Using Redux to Manage State, Effects, and Routing at Jet.com

About Me

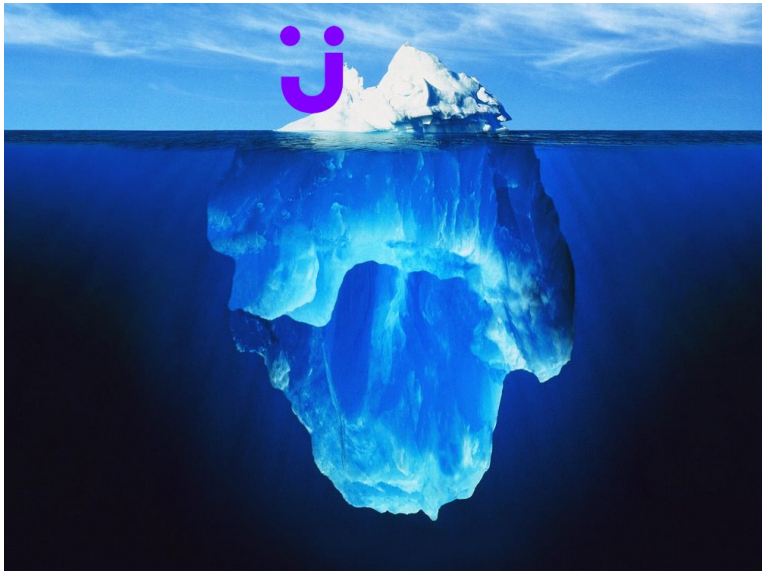
- Blake Zimmerman

- Software Engineering Intern
- Currently on Platform Tooling team

- Code and slides for this talk can be found at:
  github.com/blakezimmerman/redux-presentation

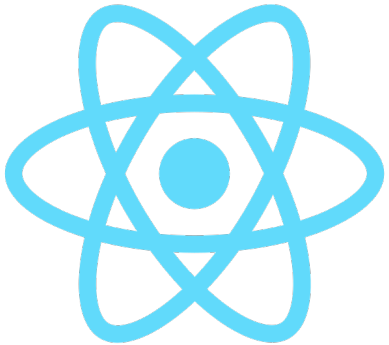
About This Talk

- Redux is a vast topic
- Foundational patterns for some of our web apps



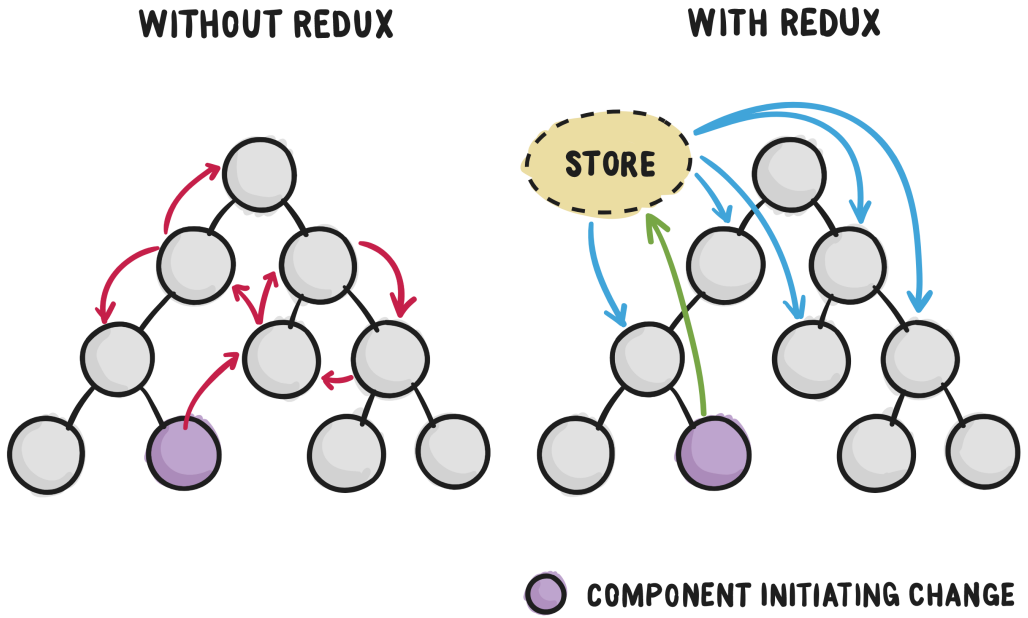
Preface

- React is a front-end JavaScript framework
- React composes components to build UIs
- TypeScript is a typed superset of JavaScript
- All code samples will be client-side code in TypeScript



What is Redux?

- Redux is a state container
  - State is global
  - State is immutable
  - Entire state of the app
  - State is a single object tree



Overview

- **State**
  - react-redux (github.com/reactjs/react-redux)
- **Effects**
  - redux-observable (github.com/redux-observable/redux-observable)
- **Routing**
  - redux-first-router (github.com/faceyspacey/redux-first-router)

State

State

- Is fundamental to most interactive web apps
- Needs to be managed predictably
- Changes are initiated through action dispatches
- Actions received by pure reducers

What is an Action?

```
5   // Action
6
7   export interface Action<T> {
8     type: string,
9     payload: T
10  }
```

Actions... In Action!

```
3   // Action Constants
4   export const INCREMENT_COUNTER = 'INCREMENT_COUNTER';
5   export const DECREMENT_COUNTER = 'DECREMENT_COUNTER';
6
7   // Action Creators
8   export const incrementCounter: (num: number) => Action<number> =
9     (num) => ({
10      type: INCREMENT_COUNTER,
11      payload: num
12    });
13
14  export const decrementCounter: (num: number) => Action<number> =
15    (num) => ({
16      type: DECREMENT_COUNTER,
17      payload: num
18    });
```

Our First Reducer

```
5   export interface AppState {
6     counter: number;
7   }
8
9   const counter = (state = 0, action: Action<number>) => {
10    switch (action.type) {
11      case INCREMENT_COUNTER:
12        return state + action.payload;
13      case DECREMENT_COUNTER:
14        return state - action.payload;
15      default:
16        return state;
17    }
18  };
19
20  export const app = combineReducers({
21    counter
22  });
```

Add State and Actions to Props

```
31  const mapStateToProps = (state: State) => ({
32    location: state.location,
33    app: state.app
34  });
35
36  const mapDispatchToProps = (dispatch: Dispatch<Action<any>>) => ({
37    increment: (num: number) => dispatch(incrementCounter(num)),
38    decrement: (num: number) => dispatch(decrementCounter(num))
39  });
40
41  export default connect(mapStateToProps, mapDispatchToProps)(App);
```

Our First Component

```
17  const App = (props: Props) => {
18    const incrementByTwo = () => props.increment(2);
19    const decrementByTwo = () => props.decrement(2);
20
21    return (
22      <div>
23        <div>Count by twos!</div>
24        <button onClick={incrementByTwo}>+</button>
25        <div>Current Value: {props.app.counter}</div>
26        <button onClick={decrementByTwo}>-</button>
27      </div>
28    )
29  };
```

Our Finished Counter

Count by twos!

+

Current Value: 0

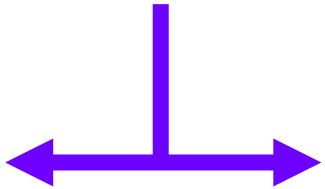
-

Count by twos!

+

Current Value: -2

-



Count by twos!

+

Current Value: 2

-

Reducing Boilerplate At Jet


```
13  // Action Creator
14
15  export type ActionDispatcher<T> = (payload?: T) => Action<T | undefined>;
16
17  export const actionCreator: <T>(type: string) => ActionDispatcher<T> =
18    (type) => (payload?) => ({ type, payload });
19
20  export const getType: (action: ActionDispatcher<any>) => string =
21    (action) => action().type;
```

```
5   export const INCREMENT_COUNTER = actionCreator('INCREMENT_COUNTER');
6   export const DECREMENT_COUNTER = actionCreator('DECREMENT_COUNTER');
```

Our New Reducer

```
switch (action.type) {
  case INCREMENT_COUNTER:
    return state + action.payload;
  case DECREMENT_COUNTER:
    return state - action.payload;
  default:
    return state;
}
```

Old

```
switch (action.type) {
  case getType(INCREMENT_COUNTER):
    return state + action.payload;
  case getType(DECREMENT_COUNTER):
    return state - action.payload;
  default:
    return state;
}
```

New

Updating mapDispatchToProps

```
36  const mapDispatchToProps = (dispatch: Dispatch<Action<any>>) => ({
37    increment: (num: number) => dispatch(INCREMENT_COUNTER(num)),
38    decrement: (num: number) => dispatch(DECREMENT_COUNTER(num))
39  });
```

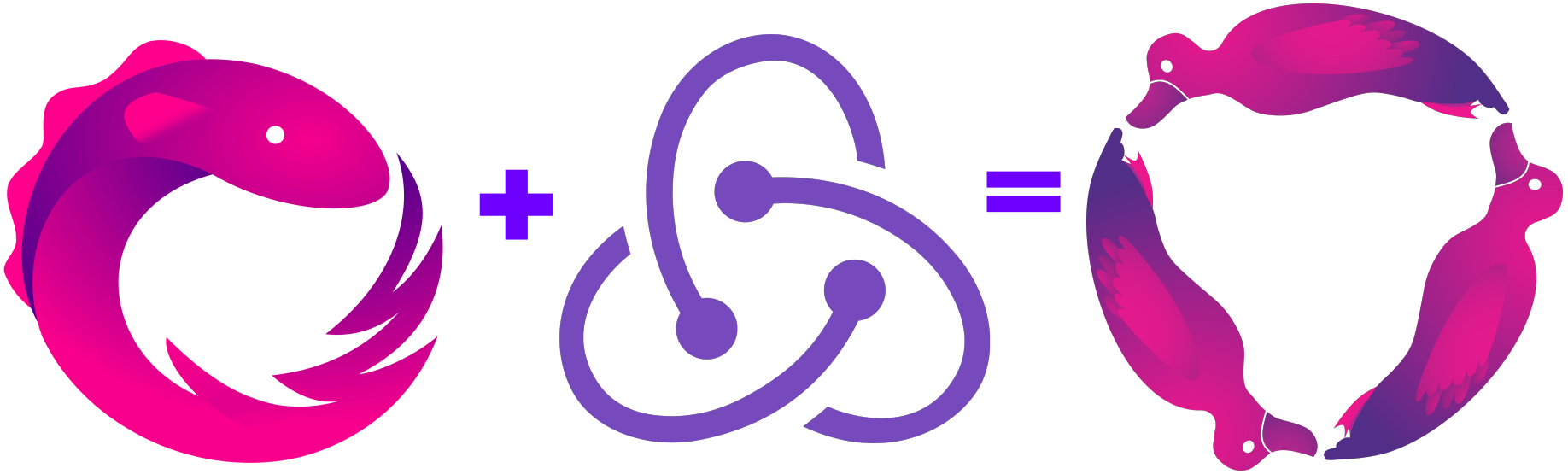
```
41  const mapDispatchToProps = ({
42    increment: INCREMENT_COUNTER,
43    decrement: DECREMENT_COUNTER
44  });
```

Shorthand if args of function match those of action creator

That’s how we manage state!

But what if I want to do something that’s asynchronous?...

Effects



Effects

- Many ways to model async actions in redux
- We prefer an effect model (i.e. actions performed as reactions to other actions)
- Redux Observable does this by means of “Epics”
- First we must start small...

Building Actions with Actions

```
// Async Action Creator

export interface AsyncActionDispatcher<T, U> {
  PENDING: ActionDispatcher<T>;
  SUCCESS: ActionDispatcher<U>;
  FAILURE: ActionDispatcher<string>;
}

export const asyncActionCreator: <T, U>(type: string) => AsyncActionDispatcher<T, U> =
  (type) => ({
    PENDING: actionCreator(type + '_PENDING'),
    SUCCESS: actionCreator(type + '_SUCCESS'),
    FAILURE: actionCreator(type + '_FAILURE')
  });
```

Our First Async Action

```
31    export const FETCH_DATA = asyncActionCreator<void, Object>('FETCH_DATA');
```

...well that was easy

Async Actions in Production

```
export const progressStepper = actionCreator('PROGRESS_STEPPER');
export const aggregate = actionCreator.async<fromModels.RequestAggregate, fromModels.DisplayGroupAggregate>('DISPLAY_GROUP_AGGREGATE');
export const split = actionCreator.async<fromModels.SplitRequest, string>('SUBMIT_SPLIT');
export const title = actionCreator.async<fromModels.UpdateTitleRequest, string>('DISPLAY_GROUP_TITLE');
export const base = actionCreator.async<fromModels.UpdateBaseRequest, string>('DISPLAY_GROUP_BASE');
export const skuTitle = actionCreator.async<fromModels.UpdateSkuTitleRequest, string>('UPDATE_GROUPED_SKU_TITLE');
export const removeSkus = actionCreator.async<fromModels.RemoveSkusRequest, string>('REMOVE_GROUPING_SKUS');
export const activeSubGroup = actionCreator.async<JetSkuId[], fromModels.ActiveSubGroupResponse[]>('ACTIVE_SUB_GROUP_FOR_SKUS');
export const add = actionCreator.async<fromModels.AddSkusRequest, string>('ADD_SKUS');
export const attributes = actionCreator.async<fromModels.UpdateAttributesRequest, string>('ATTRIBUTES');
export const append = actionCreator.async<fromModels.UpdateAppendRequest, string>('APPEND');
export const normalize = actionCreator.async<fromModels.UpdateNormalizeRequest, string>('NORMALIZE');
export const supergroup = actionCreator.async<{groupingId: string}, fromModels.GroupingSku[]>('SUPERGROUP');
export const cancel = actionCreator.async<null, string>('CANCEL');
export const submit = actionCreator.async<null, string>('SUBMIT');
export const resetGrouping = actionCreator('RESET');
```

Over time our code doesn't get more complex.
We just have more of it.

Our Actions Need a Reducer

```
40  // Async Action Reducer
41
42  export interface AsyncReducerState<T> {
43    pending: boolean;
44    result: T | undefined;
45    error: string | undefined;
46  }
47
48  export type AsyncReducer<T> = (state: AsyncReducerState<T>, action: Action<any>) => AsyncReducerState<T>;
49
50  const asyncInitialState: AsyncReducerState<any> = {
51    pending: false,
52    result: undefined,
53    error: undefined
54  };
```

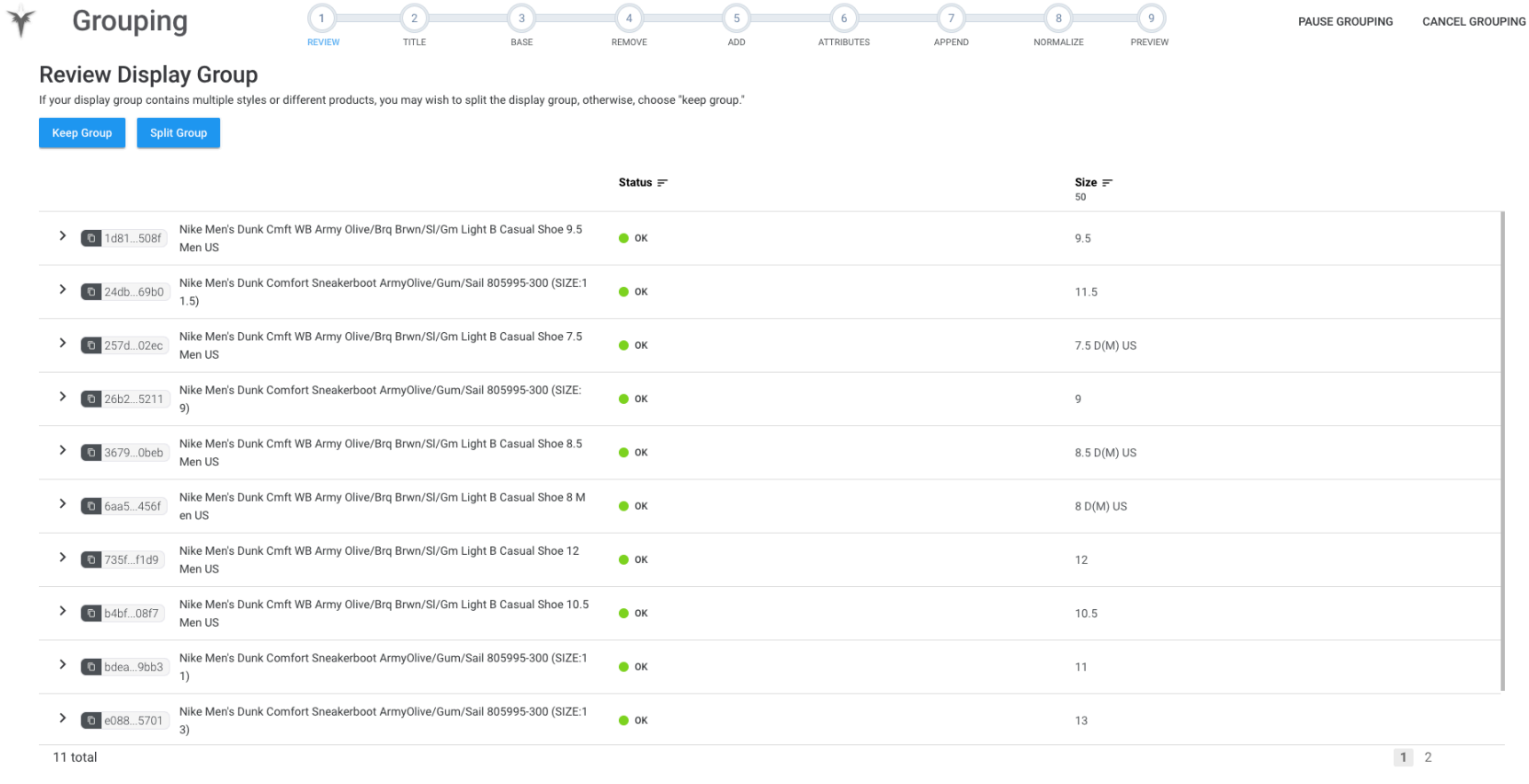
Helper Functions to the Rescue!


```
export const asyncReducer: <T>(asyncAction: AsyncActionDispatcher<any, T>) => AsyncReducer<T> =
  (asyncAction) =>
    (state = asyncInitialState, action) => {
      switch (action.type) {
        case (getType(asyncAction.PENDING)):
          return {
            ...state,
            pending: true
          };
        case (getType(asyncAction.SUCCESS)):
          return {
            ...state,
            pending: false,
            result: action.payload,
            error: undefined
          };
        case (getType(asyncAction.FAILURE)):
          return {
            ...state,
            pending: false,
            error: action.payload
          };
        default:
          return state;
      }
    };
```

Adding Our Async Action Reducer

```
16  const data = asyncReducer<Object>(FETCH_DATA);
17
18  export const app = combineReducers({
19    counter,
20    data
21  });
22
23  export interface AppState {
24    counter: number;
25    data: AsyncReducerState<Object>
26  }
```

Meanwhile in Production...



Reducers in Production

```
export const _reducer = combineReducers({
  doneStages: stepperReducer,
  aggregate: createAsyncReducer(fromActions.aggregate),
  title: createAsyncReducer(fromActions.title),
  base: createAsyncReducer(fromActions.base),
  skuTitle: createAsyncReducer(fromActions.skuTitle),
  removeSkus: createAsyncReducer(fromActions.removeSkus),
  activeSubGroup: createAsyncReducer(fromActions.activeSubGroup),
  conflict: fromRemove.reducer,
  attributes: createAsyncReducer(fromActions.attributes),
  add: fromAdd.reducer,
  variants: fromAttributes.reducer,
  orderedVariants: fromAppend.reducer,
  append: createAsyncReducer(fromActions.append),
  normalize: createAsyncReducer(fromActions.normalize),
  cancel: createAsyncReducer(fromActions.cancel),
  submit: createAsyncReducer(fromActions.submit),
  split: createAsyncReducer(fromActions.split),
  supergroup: createAsyncReducer(fromActions.supergroup),
  splitStage: fromSplitStage.reducer
});
```

What We Would Have Had...

```
export const asyncReducer: <T>(asyncAction: AsyncActionDispatcher<any, T>) => AsyncReducer<T> =
  (asyncAction) =>
    (state = asyncInitialState, action) => {
      switch (action.type) {
        case (getType(asyncAction.PENDING)):
          return {
            ...state,
            pending: true
          };
        case (getType(asyncAction.SUCCESS)):
          return {
            ...state,
            pending: false,
            result: action.payload,
            error: undefined
          };
        case (getType(asyncAction.FAILURE)):
          return {
            ...state,
            pending: false,
            error: action.payload
          };
        default:
          return state;
      }
    };
```

What We Have Now...

```
16    const data = asyncReducer<Object>(FETCH_DATA);
```

Add Async Action to Component Props

```
52  const mapDispatchToProps = ({
53    increment: INCREMENT_COUNTER,
54    decrement: DECREMENT_COUNTER,
55    fetchDataPending: FETCH_DATA.PENDING
56  });
```

Integrate Async Action with Component

```
25      const fetchData = () => props.fetchDataPending();
```

```
35          <div style={appStyles.data}>
36            <button onClick={fetchData}>Fetch Data</button>
37            <div>
38              {'Current Data: '}
39              {props.app.data.pending ? 'Pending ... ' :
40                props.app.data.result ? props.app.data.result.toString() : 'None'
41              }
42            </div>
43          </div>
```

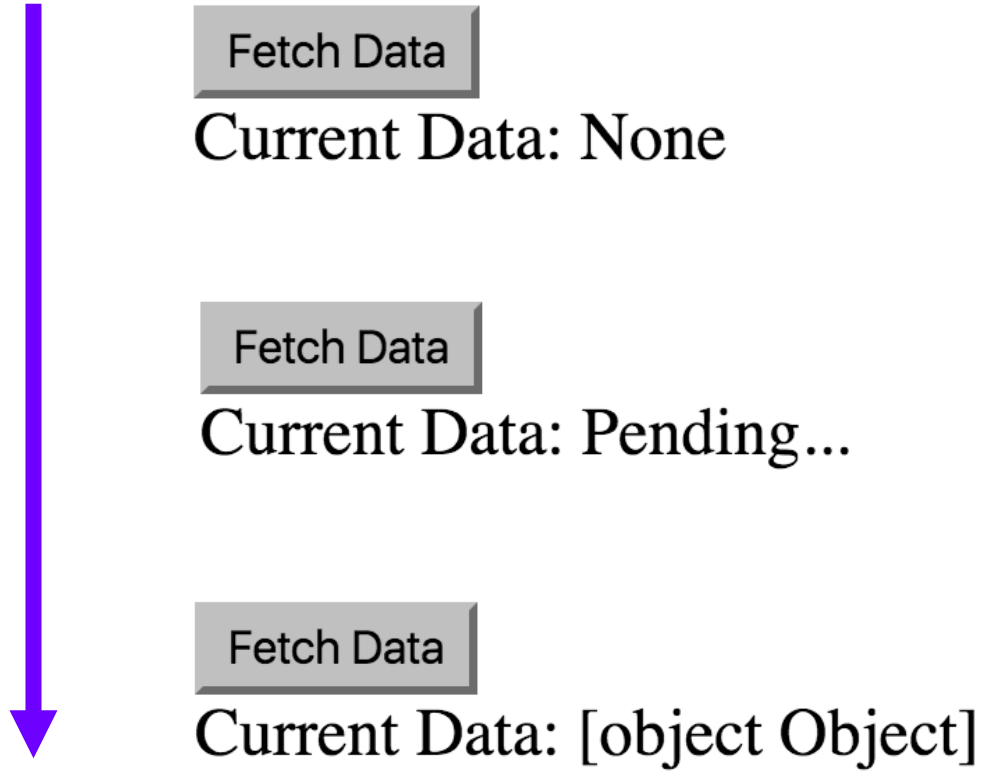
So where’s the asynchronous code?...

Async code is ‘Epic’!

```
86  export type Epic = (actions$: ActionsObservable<Action<any>>) => Observable<Action<any>>;
```

```
7   const fetchDataEpic: Epic = (actions$) =>
8     actions$.ofType(getType(FETCH_DATA.PENDING)).pipe(
9       debounceTime(500),
10      mergeMap((action) =>
11        axios.get('https://jsonplaceholder.typicode.com/posts/1')
12          .then((response) => FETCH_DATA.SUCCESS(response.data))
13          .catch((err) => FETCH_DATA.FAILURE(err.response.data))
14      )
15    );
16
17  export const appEpic = combineEpics(
18    fetchDataEpic
19  );
```

Our Finished Data Fetcher



Effects in Production

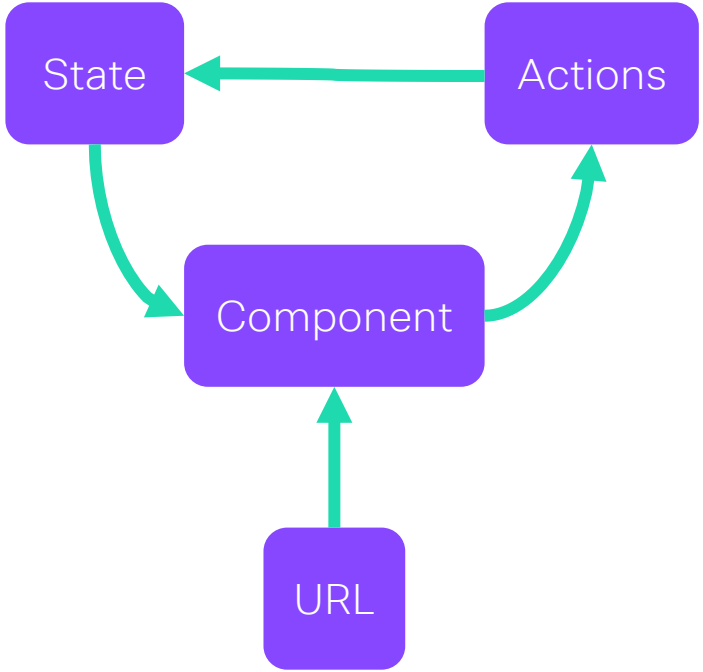
```
162   @Effect() public skuAttributes$ = this.actions$
163     .ofType(fromActions.aggregate.done.type)
164     .switchMap(() => this.store.select(fromSelect.selectUniqueAttributesIds))
165     .distinctUntilChanged()
166     .map((ids: AttributeId[]) => new RequestAttributesLookup(ids));
167
168   @Effect() public initializeCurrentVariants$ = this.actions$
169     .ofType(fromActions.aggregate.done.type)
170     .switchMap(() => this.store.select(fromSelect.selectVariantDimensions))
171     .filter(notNull)
172     .first()
173     .map(map(idProp))
174     .map((ids: VariantDimensionId[]) => initializeVariants(ids));
```

Doing More with Redux...

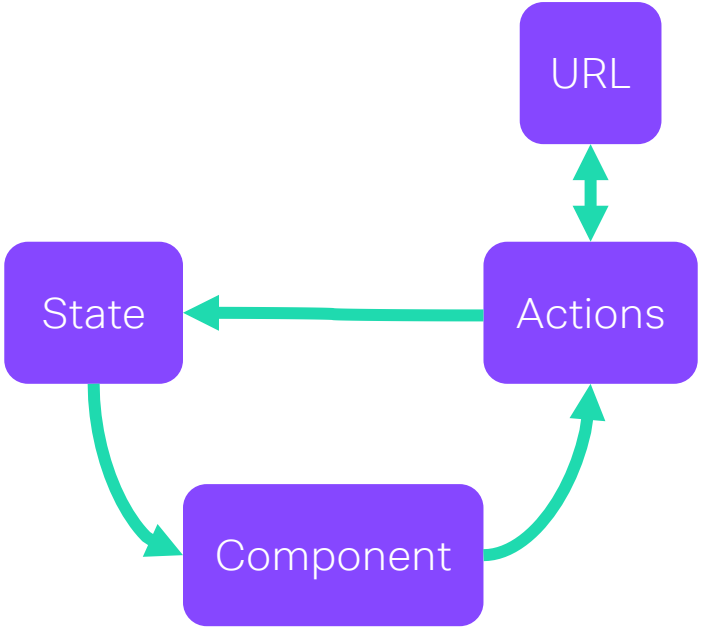
Routing with Redux

- Routing is critical to many single page apps
- Most routers lead to two sources of truth:
  - Redux State
  - Current URL
- ‘Redux First Router’ treats the url as redux state
- Simply dispatch actions to route

Routing



Redux Routing



Setting Up The Router

```
5   const history = createHistory();
6
7   const routesMap = {
8     HOME: '/',
9     PAGE_ONE: '/page-one',
10    PAGE_TWO: '/page-two'
11  };
12
13  const { reducer, middleware, enhancer } = connectRoutes(history, routesMap);
14
15  export { reducer as routerReducer };
16  export { middleware as routerMiddleware };
17  export { enhancer as routerEnhancer };
```

Rendering Components from Routes

```
46    const GetPage = () => {
47      switch (props.location.pathname) {
48        case ('/'): return <ActionsDemo />;
49        case ('/page-one'): return <PageOne />;
50        case ('/page-two'): return <PageTwo />;
51        default: return <NotFound />;
52      }
53    };
54
55    return (
56      <GetPage />
57    );
58  };
```

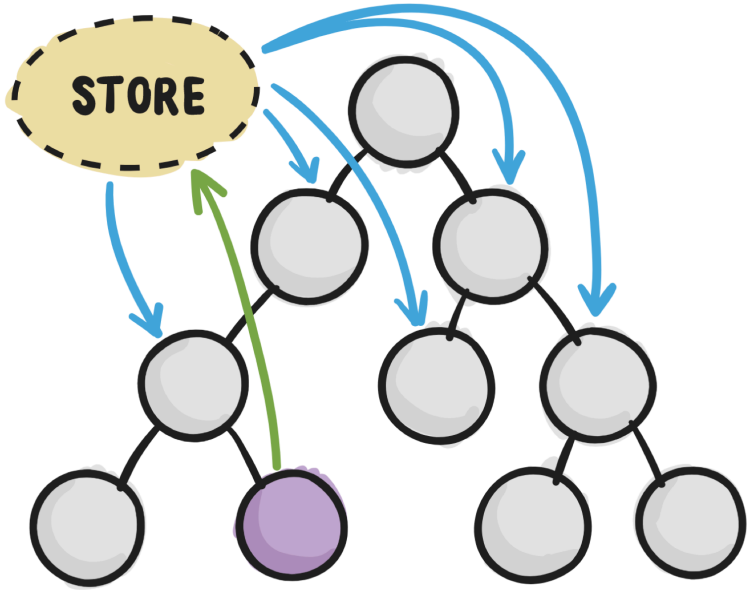
Example Component With Links

```
4   const PageOne = () => (
5     <div>
6       <div>
7         Page One
8       </div>
9       <div><Link to='/'>Go to Home</Link></div>
10      <div><Link to='/page-two'>Go to Page Two</Link></div>
11    </div>
12  );
```

There's So Much More...

- Redux First Router could be its own presentation
- Many ways to use it
- Works with universal web apps
- Worth looking further into

So how do we create our store?



The Root Reducer

```
// Create root reducer

import { combineReducers } from 'redux';
import { LocationState } from 'redux-first-router';
import { routerReducer as location } from 'router/router';
import { app, AppState } from 'app/app.reducer';

export interface State {
  location: LocationState;
  app: AppState;
}

const rootReducer = combineReducers({
  location,
  app
});
```

Epic Middleware

```
19  // Create epic middleware
20
21  import { combineEpics, createEpicMiddleware } from 'redux-observable';
22  import { appEpic } from 'app/app.epics';
23
24  const rootEpic = combineEpics(appEpic);
25
26  const epicMiddleware = createEpicMiddleware(rootEpic);
```

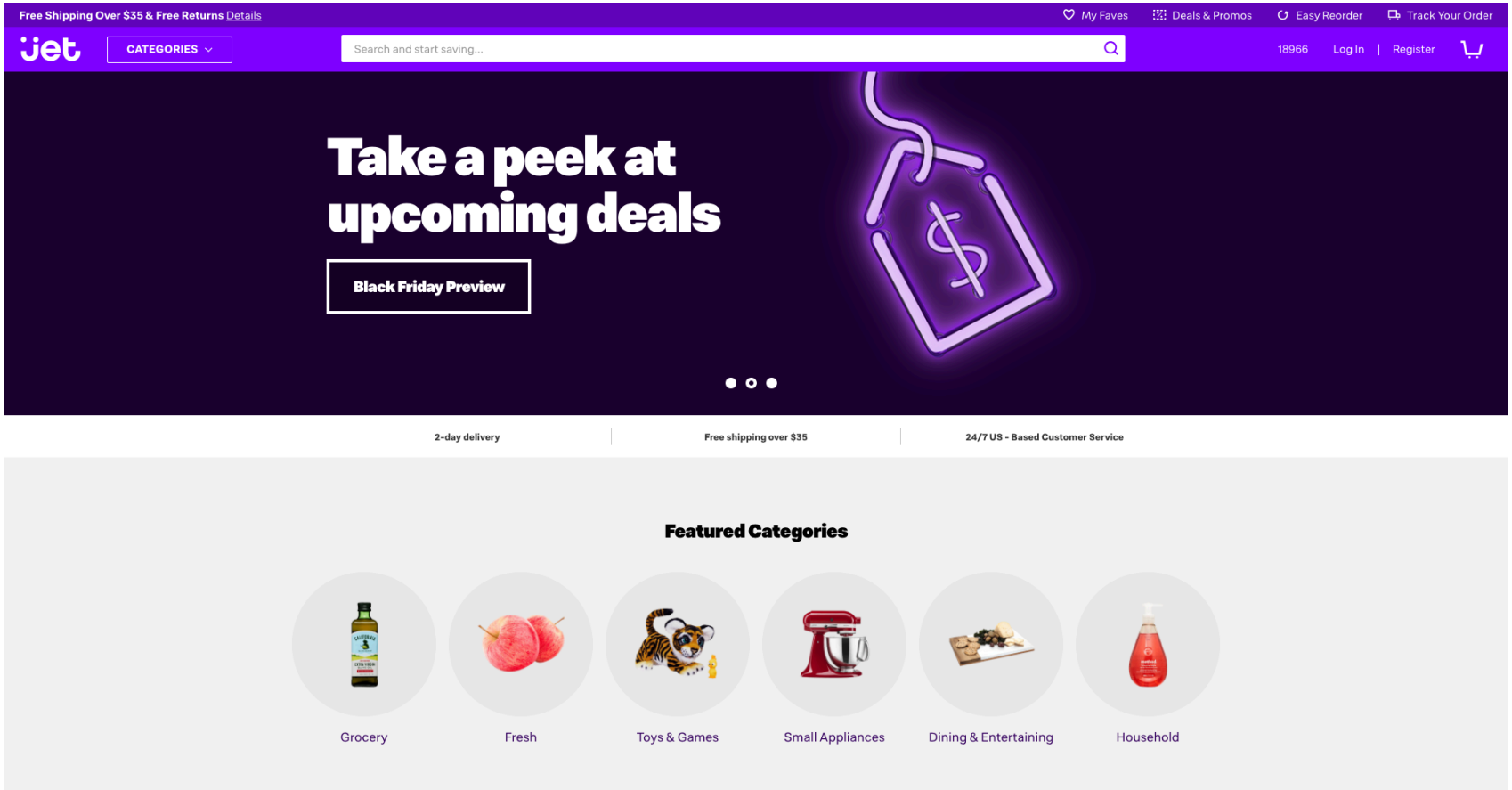
Create The Store

```
29  // Create store
30
31  import { createStore, applyMiddleware, compose } from 'redux';
32  import { routerMiddleware, routerEnhancer } from 'router/router';
33  import logger from 'redux-logger';
34
35  const store = createStore(
36    rootReducer,
37    compose(
38      routerEnhancer,
39      applyMiddleware(epicMiddleware, routerMiddleware, logger)
40    )
41  );
```

Render The App

```
44  // Render app
45
46  import * as React from 'react';
47  import { render } from 'react-dom';
48  import { Provider } from 'react-redux';
49  import App from 'app/app';
50
51  render(
52    <Provider store={store}>
53      <App />
54    </Provider>,
55    document.getElementById('root')
56  );
```

Enjoy the Result



Thank You!

Any Questions?

Code and slides for this talk can be found at:
github.com/blakezimmerman/redux-presentation