Formidable

Event Driven Redux

NationJS Frontrunners React 2019
Dillon Mulroy - Software Engineer @ Formidable

Formidable Open Source

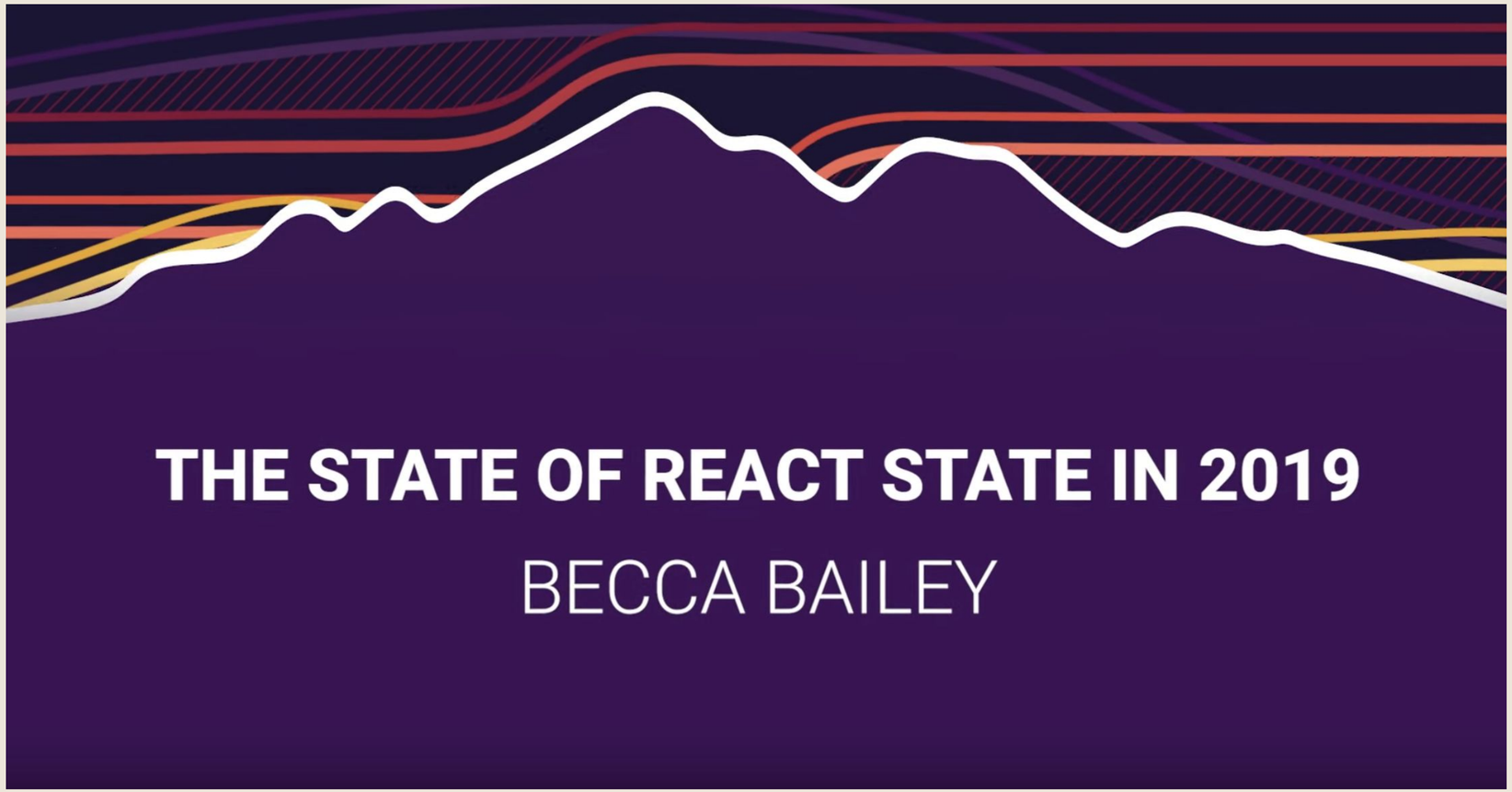
Slide Deck

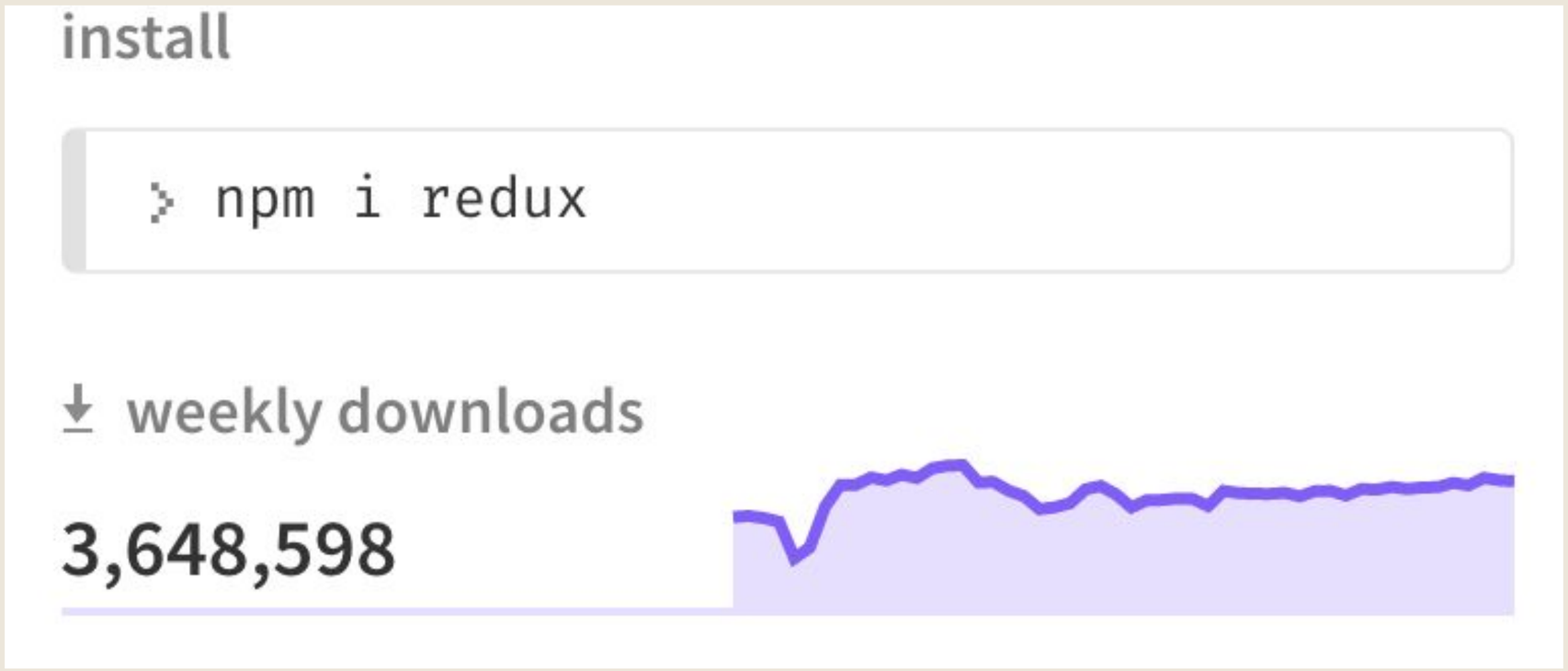
bit.ly/event-driven-redux-njs

The state of state management

This is not a talk about ‘ Why Redux’

If you’re interested in what the current and future landscape looks like, the pros and cons of different choices, and thoughtful analysis, checkout Becca Bailey’s talk from React Conf this year.





Source: https://www.npmjs.com/package/redux (11/23/19)

Redux is not going away or dying

- ~16 million monthly downloads
- Well known and established patterns + APIs in the React community
  - Easy to hire for and onboard juniors
- Battle tested and time proven for building and shipping production level applications

A little background on why this talk


Dan Abramov
@dan_abramov
Reading some Redux example code I wrote four years ago and I have no effing idea what's going on there
5:47 PM · Nov 4, 2019 · Twitter Web App
421 Retweets
2.9K Likes


Dan Abramov
@dan_abramov
Why is this stuff in five different files and constants SHOUTING at me
5:49 PM · Nov 4, 2019 · Twitter Web App
31 Retweets
548 Likes

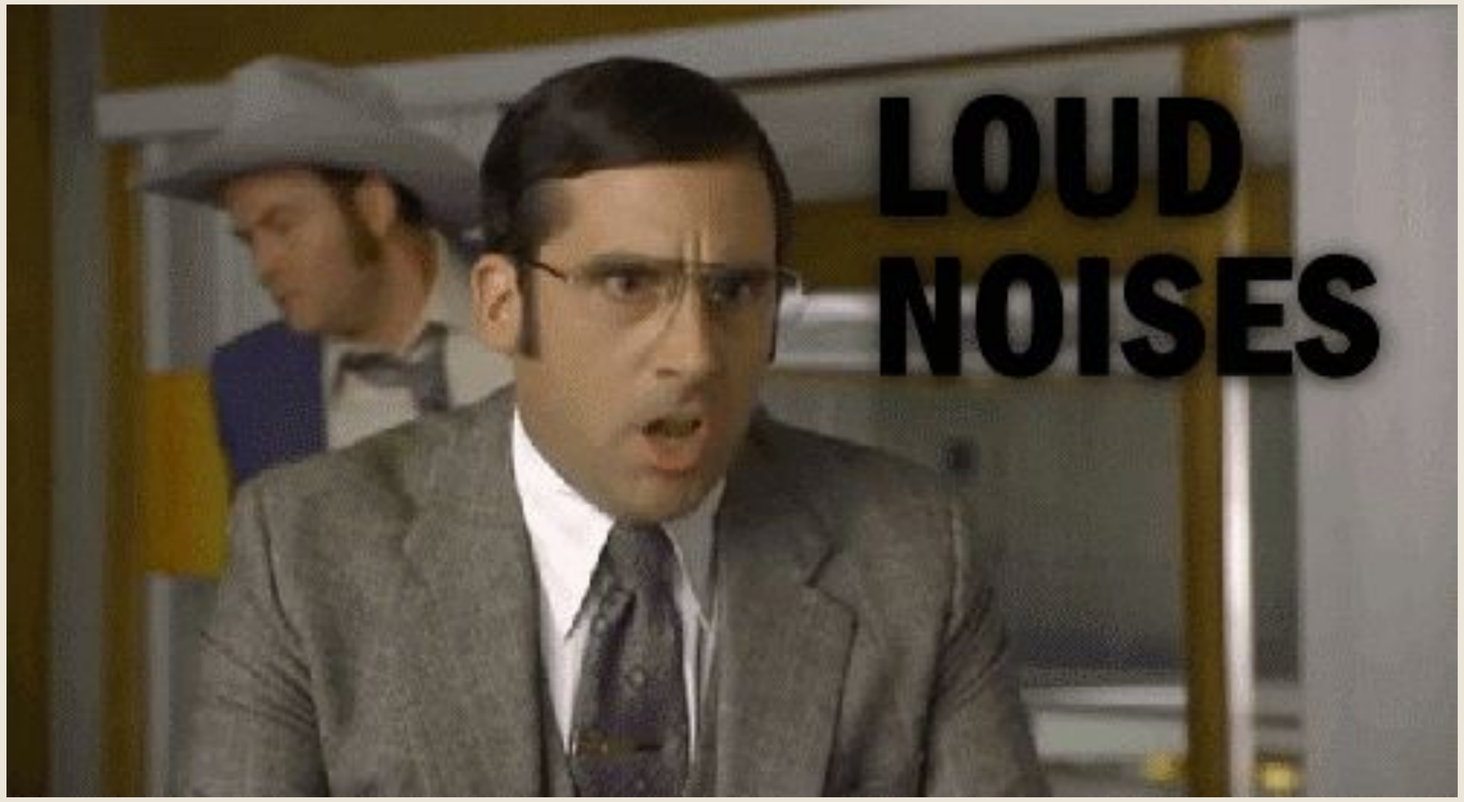
My experiences with Redux

So. Many. Files. Large PRs for even small features

My experiences with Redux

Developer exhaustion due to extensive boilerplate

My experiences with Redux



Needless verbosity (CONSTANTS_EVERYWHERE) that makes code harder to read and follow

My experiences with Redux

Reducers end up as simple “setters”

My experiences with Redux

My component APIs (specifically callback props) started looking and reading like reducer setters or thunk actions

My experiences with Redux

What state belongs in Redux and what state should be local???

It doesn't have to be this way

Idiomatic Redux Blog Series

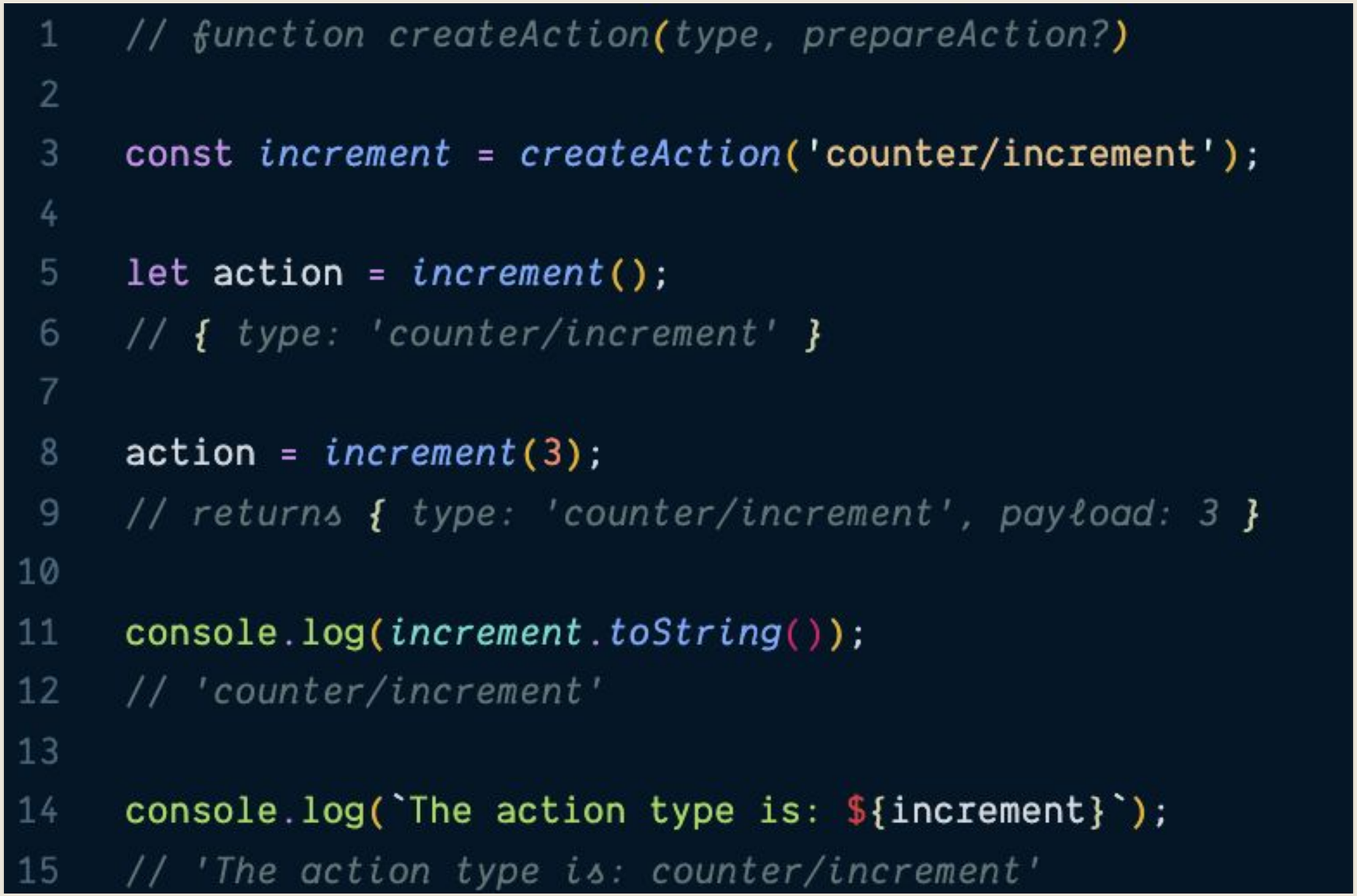
- Idiomatic Redux: The Tao of Redux, Part 1 - Implementation and Intent

- Idiomatic Redux: The Tao of Redux, Part 2 - Practice and Philosophy

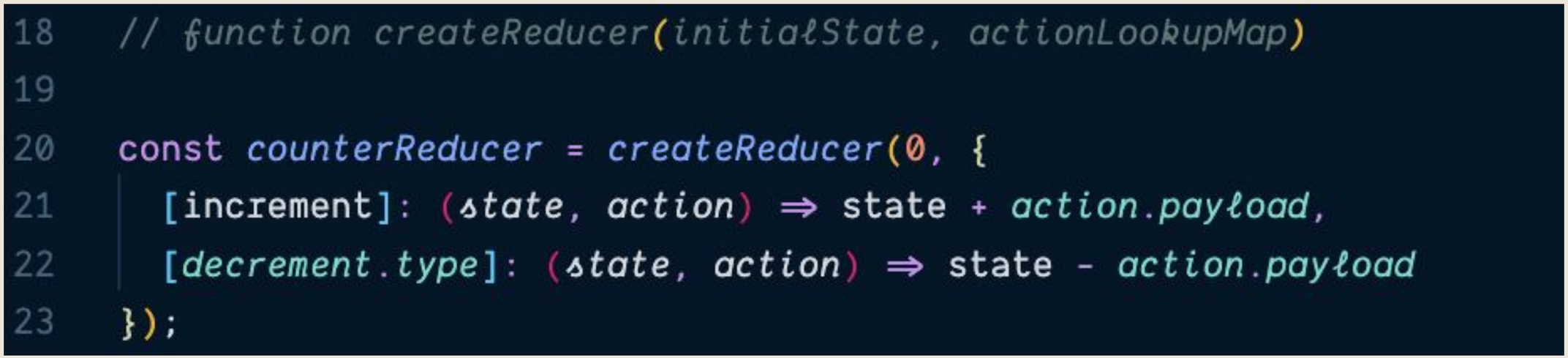
- Idiomatic Redux: Redux Toolkit 1.0

Redux Toolkit (RTK)

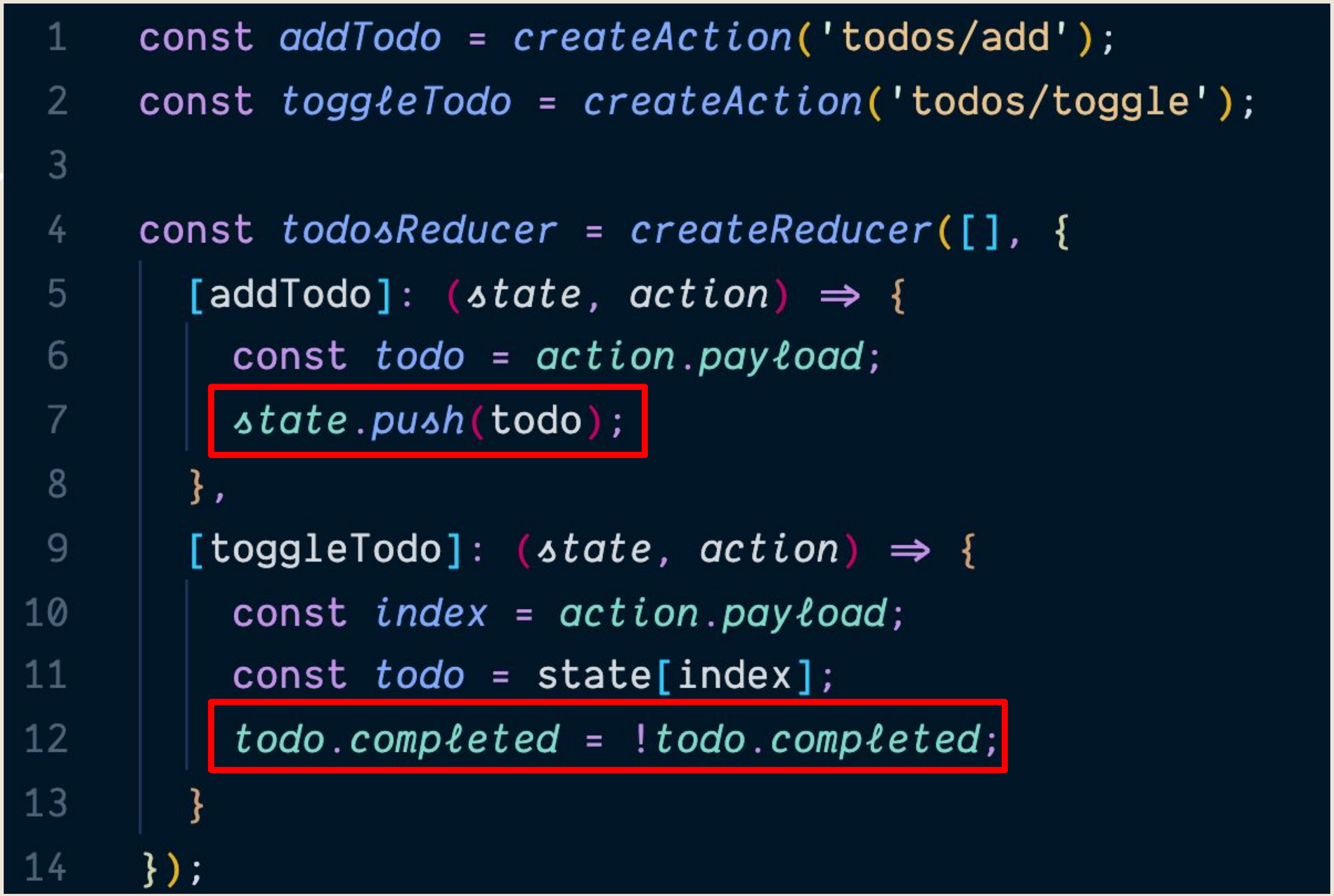
- The officially recommended way to start with and use Redux
- Formerly known as Redux Starter Kit
- Easy to integrate into existing project
- API
  - configureStore
  - createAction
  - createReducer
  - createSlice
- https://redux-toolkit.js.org/



```
// function createAction(type, prepareAction?)

const increment = createAction('counter/increment');

let action = increment();
// { type: 'counter/increment' }

action = increment(3);
// returns { type: 'counter/increment', payload: 3 }

console.log(increment.toString());
// 'counter/increment'

console.log(`The action type is: ${increment}`);
// 'The action type is: counter/increment'
```



```
// function createReducer(initialState, actionLookupMap)

const counterReducer = createReducer(0, {
  [increment]: (state, action) => state + action.payload,
  [decrement.type]: (state, action) => state - action.payload
});
```



```
1   const addTodo = createAction('todos/add');
2   const toggleTodo = createAction('todos/toggle');
3
4   const todosReducer = createReducer([], {
5     [addTodo]: (state, action) => {
6       const todo = action.payload;
7       state.push(todo);
8     },
9     [toggleTodo]: (state, action) => {
10      const index = action.payload;
11      const todo = state[index];
12      todo.completed = !todo.completed;
13    }
14  });
```

react-redux hooks

- useDispatch
- useSelector
- Removes the need for the connect Higher Order Component

```
1   const mapStateToProps = state => ({
2     value: state.counter
3   });
4
5   const mapDispatchToProps = dispatch => ({
6     onIncrement: () => dispatch(increment(1)),
7     onDecrement: () => dispatch(decrement(1))
8   });
9
10  export default connect(mapStateToProps, mapDispatchToProps)(Counter);
```

```tsx
const CounterContainer = () => {
  const dispatch = useDispatch();
  const value = useSelector(state => state.counter); // Reselect compatible

  const onIncrement = () => dispatch(increment(1));
  const onDecrement = () => dispatch(decrement(1));

  return (
    <Counter
      value={value}
      onIncrement={onIncrement}
      onDecrement={onDecrement}
    />
  );
};
```

Redux architecture; A closer look

React

Responsibilities

- Presentation layer and UI of our applications
- Describe APIs (component props) that must be fulfilled to render meaningful data and interactions for our users

Knowledge of the system

- Should have little to no knowledge of Redux or async middleware

The Store

Responsibilities

- Storing “slices” of state
- Composing our reducers to act on those slices of state.

Knowledge of the system

- Reducers

```
// store
{ entities, features }

// entities/playlists
{
  344352: { id: 344352, name: 'Workout playlist', songIds: [...] },
  454353: { id: 454353, name: 'Jazz vibes', songIds: [...] },
  ...
}

// entities/songs
{
  583628: { id: 583628, name: 'Thunderstruck', artist: 'AC/DC', ... },
  395032: { id: 395032, name: 'Friday', artist: 'Rebecca Black', ... },
  ...
}

// features/playlist
{ playlistId: 344352, selectedSongId: 583628 }

// features/modal
{ open: false, activeModal: null }
```

Reducers

Responsibilities

- Applying state mutations to the state that it's responsible for
- Outside of asynchronous side effects, the majority of our business logic should be captured and live here

Knowledge of the system

- The slice of state it acts on
- Actions that trigger state mutations to occur

Selectors

Responsibilities

- Accessing state from various “slices” in the store
- Applying transformations to extracted state to prepare it for being consumed by our component APIs

Knowledge of the system

- The store and its slices of state
- Component APIs/props

Containers

Responsibilities

- Stitching together selectors and actions to fulfill component APIs

Knowledge of the system

- Available Selectors
- Available Actions
- Component APIs/props

Actions

Responsibilities

- Plain JS Objects that adhere to the Flux Standard Action (FSA) spec
- Communication between containers, reducers, and potentially async middleware

Knowledge of the system

- Reducers

  OR

- Containers/Component API

Knowledge Matrix

Rows have knowledge of columns

	Store	Reducers	Selectors	Containers	Actions
Store	-	✔️	🚫	🚫	🚫
Reducers	✔️	-	🚫	🚫	✔️
Selectors	✔️	🚫	-	✔️	🚫
Containers	🚫	🚫	✔️	-	✔️
Actions	🚫	✔️ OR 🚫	🚫	🚫 OR ✔️	-

Actions as commands

- Knowledge of reducers
- Used as API to “command” reducers what to do
- The ‘type’ is typically seen as VERB_NOUN
- Typically map 1:1 to a reducer
- Business logic has a tendency to leak into our containers
- Pairs well with Redux Thunk

```
// actions.js
const removeFromPlaylist = createAction('playlist/removeFromPlaylist');
const setSelectedSong = createAction('playlistView/setSelectedSong');
const openModal = createAction('modal/openModal');
const closeModal = createAction('modal/closeModal');
```

```
// reducers.js
const modalInitialState = { open: false, activeModal: null };

const modalReducer = createReducer(modalInitialState, {
  [openModal]: (state, action) => ({ open: true, activeModal: action.payload }),
  [closeModal]: () => modalInitialState
});

const playlistInitialState = {};

const playlistReducer = createReducer(playlistInitialState, {
  [removeFromPlaylist]: (state, action) => {
    if (state[action.payload.playlistId]) {
      return state[action.payload.playlistId].filter(id => id !== songId);
    }
  }
});

const playlistViewInitialState = { playlistId: null, selectedSongId: null };

const playlistViewReducer = createReducer(playlistViewInitialState, {
  [setSelectedSong]: (state, action) => {
    state.selectedSongId = action.payload;
  }
});
```

```
// playlist/container.js
const mapDispatchToProps = dispatch => ({
  onRemoveSongClick: songId => {
    dispatch(openModal('removeSongModal'));
    dispatch(setSelectedSong(songId));
  }
});

// removeSongModal/container.js
const mapDispatchToProps = (dispatch, { playlistId, songId }) => ({
  onRemoveClick: () => {
    dispatch(removeFromPlaylist({ playlistId, songId }));
    dispatch(closeModal());
    dispatch(setSelectedSong(null));
  }
});
```

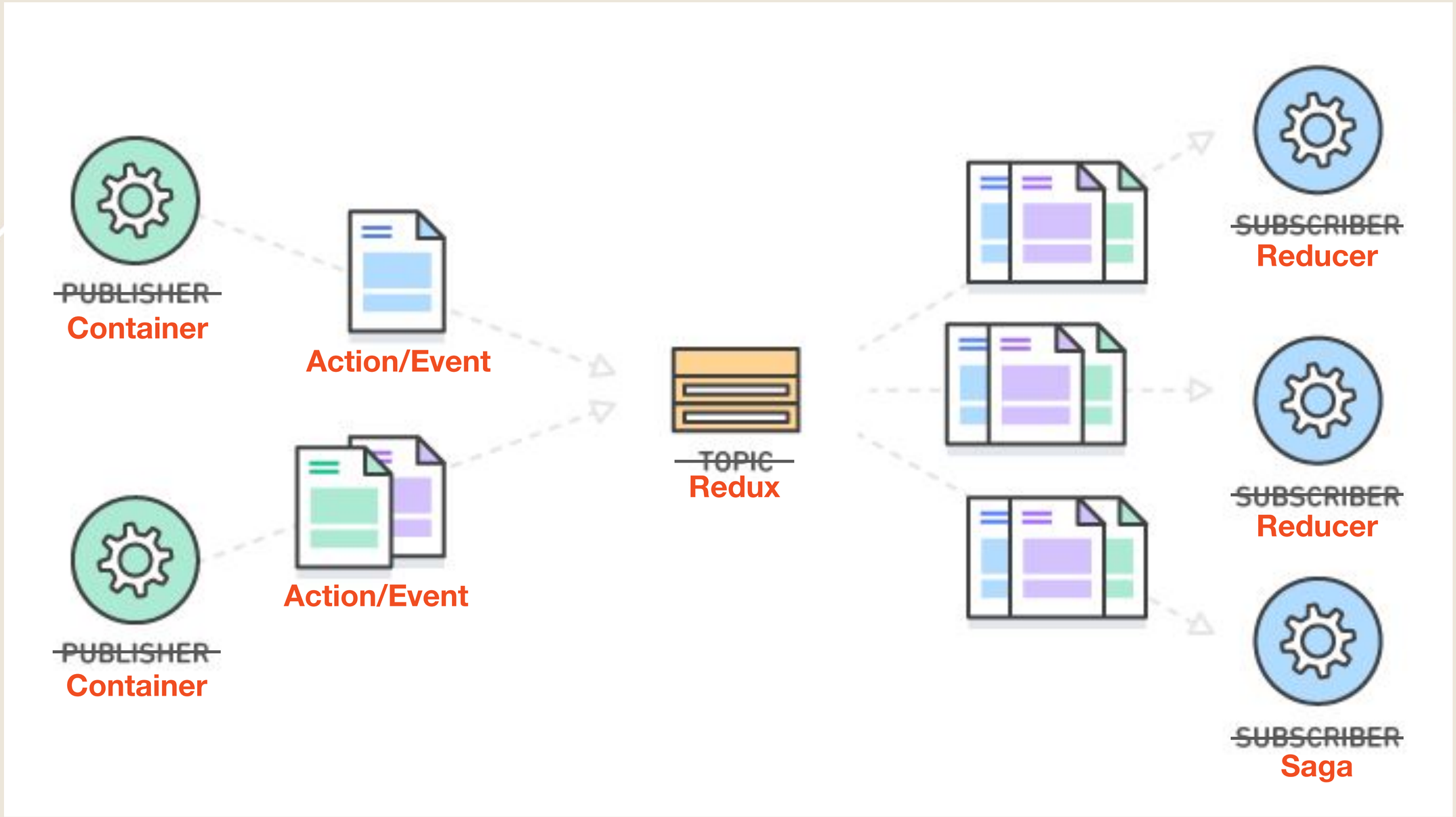
Actions as events

- Knowledge of containers/component API
- Describe “events” that have occured
- The ‘type’ is past tense
- Can map 1:M reducers
- Isolates the role of containers to fulfilling our component APIs/props
- Pairs well with Redux Saga

```
// actions.js
const removeSongClicked = createAction('playlistView/removeSongClicked');
const removeClicked = createAction('removeSongModal/removeClicked');
```

```
// reducers.js
const modalInitialState = { open: false, activeModal: null };

const modalReducer = createReducer(modalInitialState, {
  [removeSongClicked]: () => ({ open: true, activeModal: 'removeSongModal' }),
  [removeClicked]: () => ({ open: false, activeModal: null })
});

const playlistInitialState = {};

const playlistReducer = createReducer(playlistInitialState, {
  [removeClicked]: (state, action) => {
    if (state[action.payload.playlistId]) {
      return state[action.payload.playlistId].filter(id => id !== songId);
    }
  }
});

const playlistViewInitialState = { playlistId: null, selectedSongId: null };

const playlistViewReducer = createReducer(playlistViewInitialState, {
  [removeSongClicked]: (state, action) => {
    state.selectedSongId = action.payload;
  }
});
```

```
// playlist/container.js
const mapDispatchToProps = dispatch => ({
  onRemoveSongClick: songId => dispatch(removeSongClicked(songId))
});

// removeSongModal/container.js
const mapDispatchToProps = (dispatch, { playlistId, songId }) => ({
  onRemoveClick: () => dispatch(removeClicked({ playlistId, songId }))
});
```



Source: https://aws.amazon.com/pub-sub-messaging/

An event driven workflow

Storybook

↓

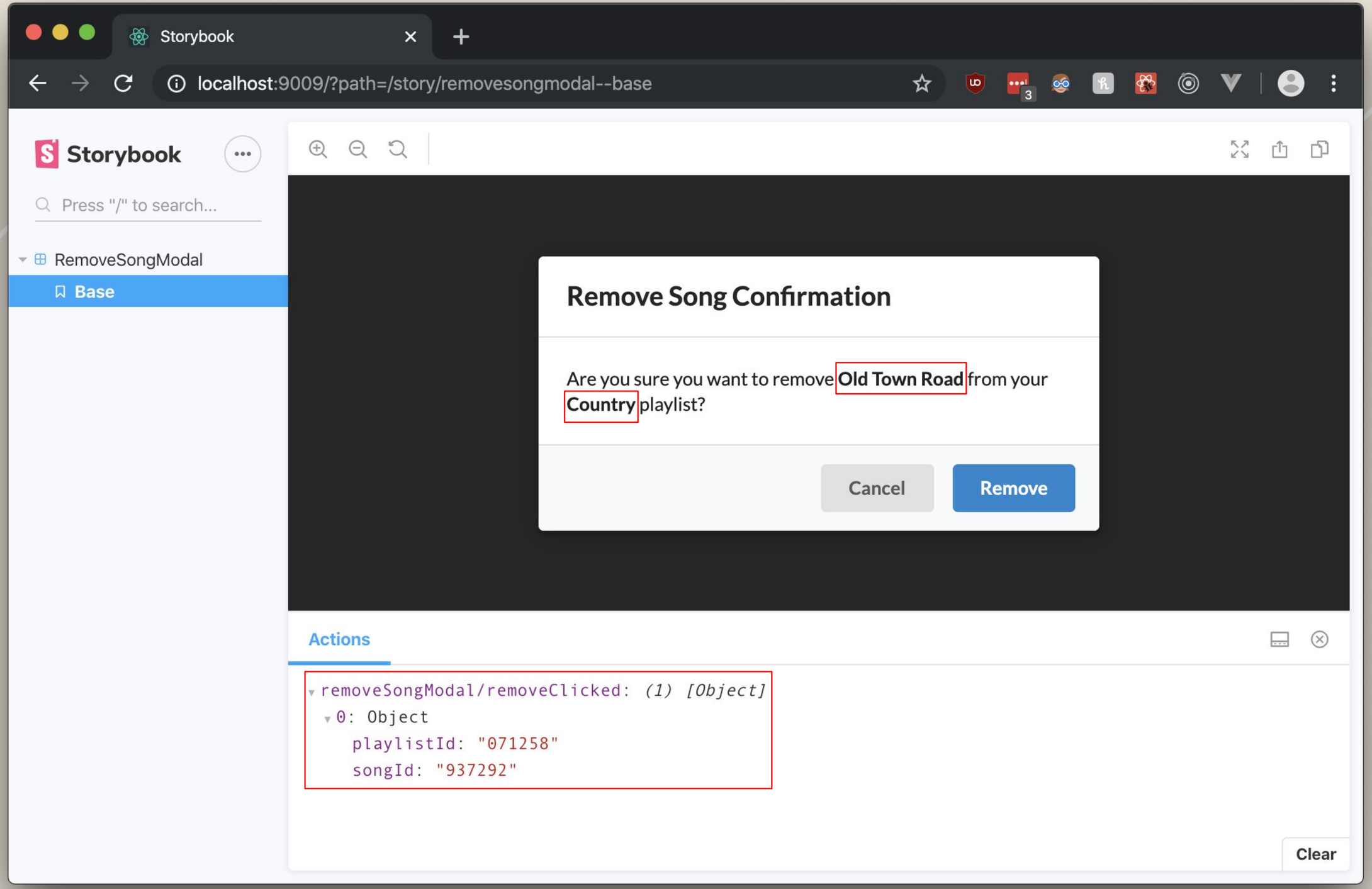
Actions/Events

↓

Container

↓

Reducers/Sagas


Storybook
localhost:9009/?path=/story/removesongmodal--base
Storybook
Press "/" to search...
RemoveSongModal
Base
Remove Song Confirmation
Are you sure you want to remove Old Town Road from your Country playlist?
Cancel
Remove
Actions
removeSongModal/removeClicked: (1) [Object]
0: Object
playlistId: "071258"
songId: "937292"
Clear

Redux Docs & Style Guide

- Model Actions as Events, Not Setters
- Allow Many Reducers to Respond to the Same Action
- Avoid Dispatching Many Actions Sequentially

In Summary

- Use react-redux hook and RTK to eliminate boilerplate
- Shift your mental modal to an event driven pub/sub model
- Fewer actions, isolated business logic, easier tests, and more scalable code

Thank you!