
NETFLIX
RxJS + Redux + React = Amazing
Side Effect Management with RxJS
Jay Phelps | @_jayphelps

Managing **state** stuff is **hard**

Redux makes it **simple**

(not necessarily *easy*)

Managing **async** stuff is **harder**

Some **async** is **complex regardless**
of the abstraction

RxJS makes it ***manageable***

Jay Phelps

Senior Software Engineer | NETFLIX

@_jayphelps

What is redux?

Crash Course

What is redux?

Provides predicable state management using actions and reducers

What's an "action"?

Describes **something** has (or should) happen, but they **don't specify how** it should be done

```
{
  type: 'CREATE_TODO',
  payload: 'Build my first Redux app'
}
```

What's an "reducer"?

A **pure function** that takes the **previous state**
and an **action** and returns the **new state**

What's an "reducer"?

Sometimes it returns the previous state

```
(state, action) => state
```

What's an "reducer"?

Sometimes it computes new state

```
(state, action) => state + action.payload
```

```
const counter = (state = 0, action) => {
  switch (action.type) {
    case 'INCREMENT':
      return state + 1;

    case 'DECREMENT':
      return state - 1;

    default:
      return state;
  }
};
```

Reducers handle **state transitions**, but they must be done **synchronously**.

```
const counter = (state = 0, action) => {
  switch (action.type) {
    case 'INCREMENT':
      return state + 1;

    case 'DECREMENT':
      return state - 1;

    default:
      return state;
  }
};
```

What are **async** stuff do we commonly do?

Async

- User interactions (mouse, keyboard, etc)
- AJAX
- Timers/Animations
- Web Sockets
- Work Workers, etc

Some can be handled **synchronously**

```
<button onClick={() => dispatch({ type: 'INCREMENT' })}>
  Increment
</button>
```

```
const counter = (state = 0, action) => {
  switch (action.type) {
    case 'INCREMENT':
      return state + 1;

    case 'DECREMENT':
      return state - 1;

    default:
      return state;
  }
};
```

Sometimes you need **more control**

- AJAX cancellation/composing
- Debounce/throttle/buffer/etc
- Drag and Drop
- Web Sockets, Work Workers, etc

Use **middleware** to manage **async** / side effects

Most of them use **callbacks** or **Promises**

Callbacks

The most **primitive** way to handle **async** in JavaScript

Callbacks

```
fetchSomeData((error, data) => {
  if (!error) {
    dispatch({ type: 'HERES_THE_DATA', data });
  }
});
```

Callback Hell

```
fetchSomeData(id, (error, data) => {
  if (!error) {
    dispatch({ type: 'HERES_THE_FIRST_CALL_DATA', data });

    fetchSomeData(data.parentId, (error, data) => {
      if (!error) {
        dispatch({ type: 'HERES_SOME_MORE', data });

        fetchSomeData(data.parentId, (error, data) => {
          if (!error) {
            dispatch({ type: 'OMG_MAKE_IT_STOP', data });
          }
        });
      }
    });
  }
});
```

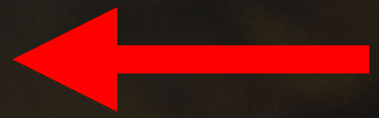
Promises

```
fetchSomeData(id)
  .then(data => {
    dispatch({ type: 'HERES_THE_FIRST_CALL_DATA', data });
    return fetchSomeData(data.parentId);
  })
  .then(data => {
    dispatch({ type: 'HERES_SOME_MORE', data });
    return fetchSomeData(data.parentId);
  })
  .then(data => {
    dispatch({ type: 'OKAY_IM_DONE', data });
  });
```

Promises

- Guaranteed future
- Immutable
- Single value
- Caching

Promises

- **Guaranteed future** 
- Immutable
- **Single value** 
- Caching

Promises

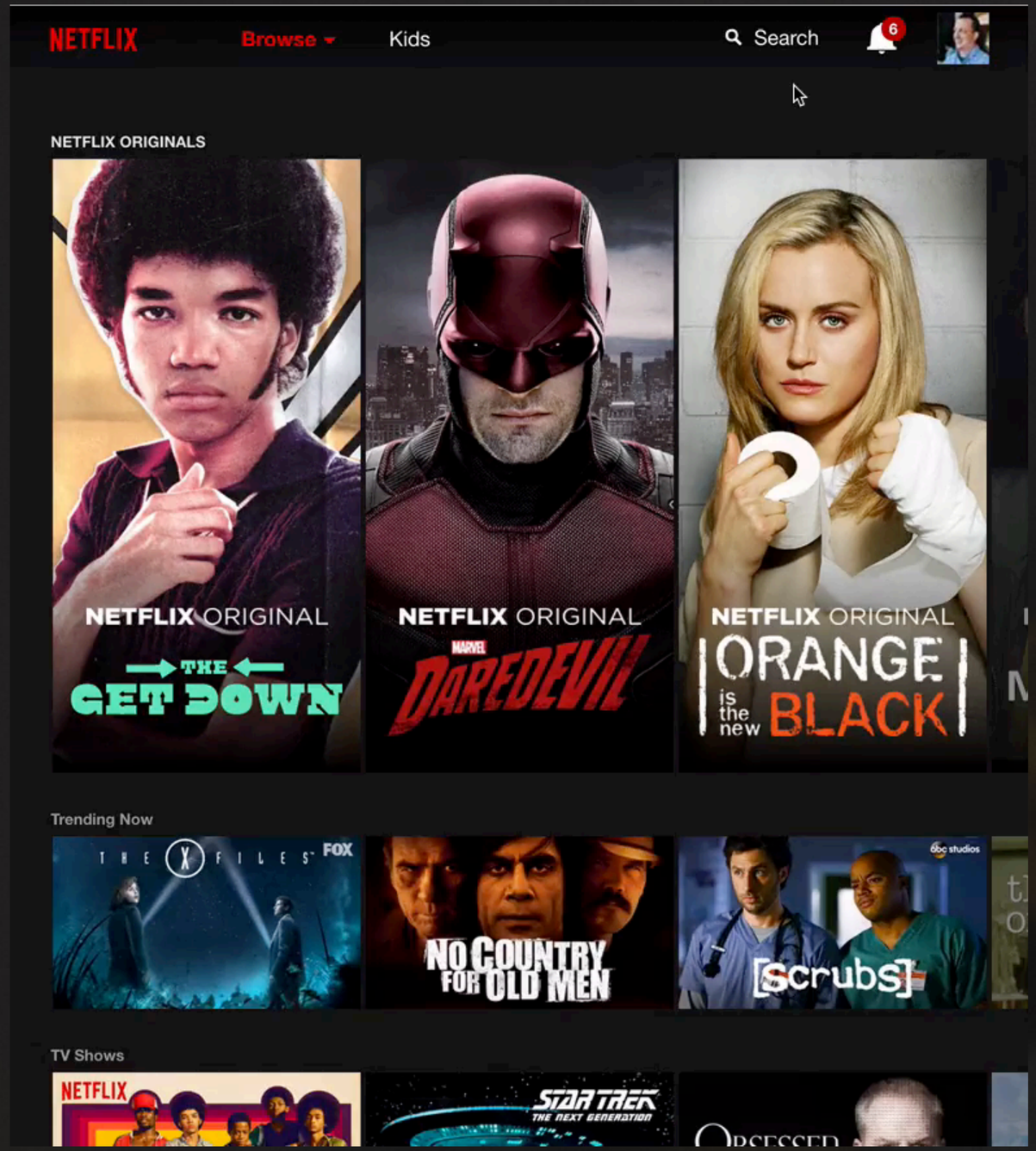
- **Guaranteed future** 
- Immutable
- **Single value**
- Caching

Promises **cannot be cancelled**

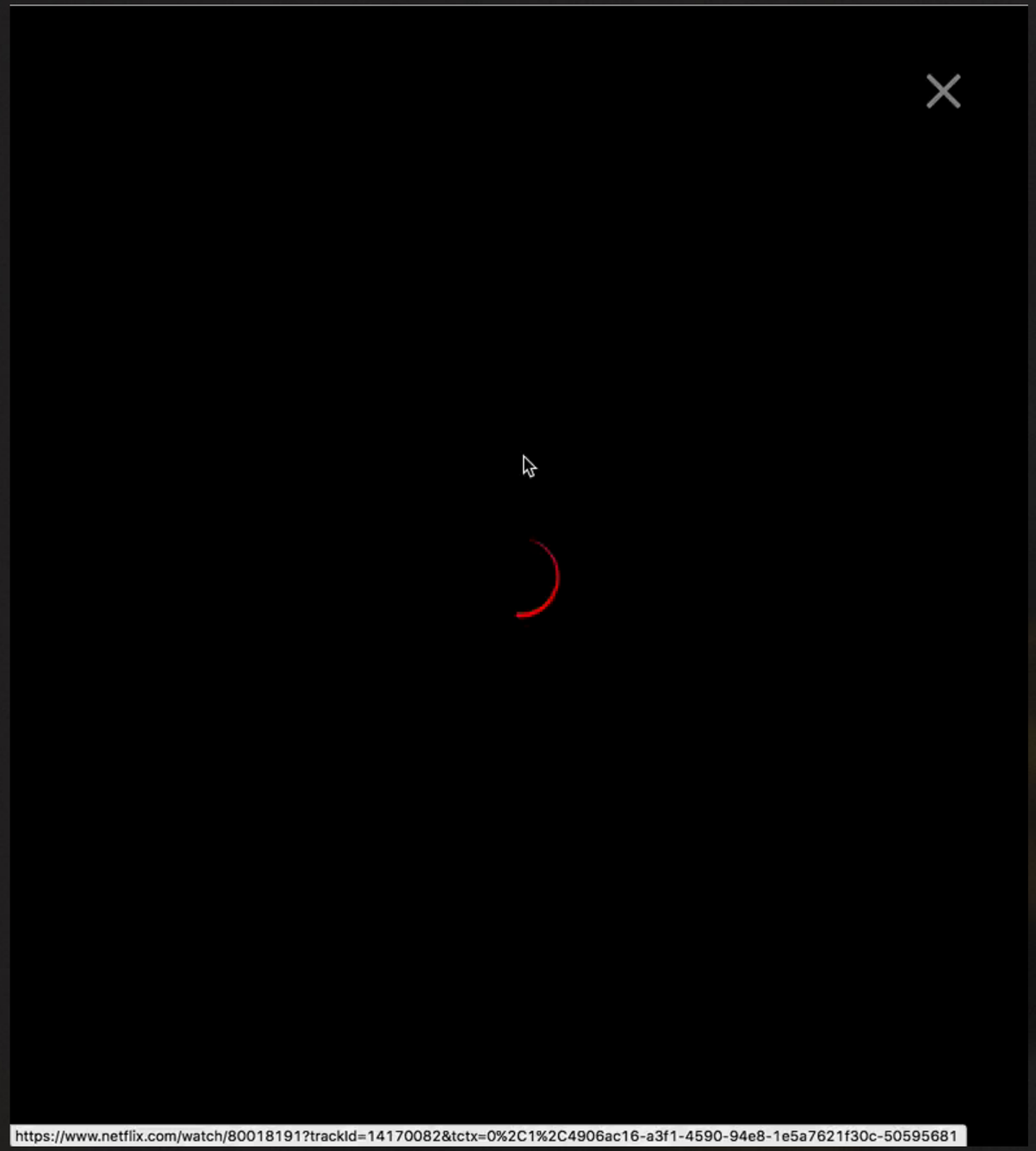
Why would you want to **cancel**?

- Changing routes/views
- Auto-complete
- User wants you to

- **Changing routes/views**
- Auto-complete
- User wants you to

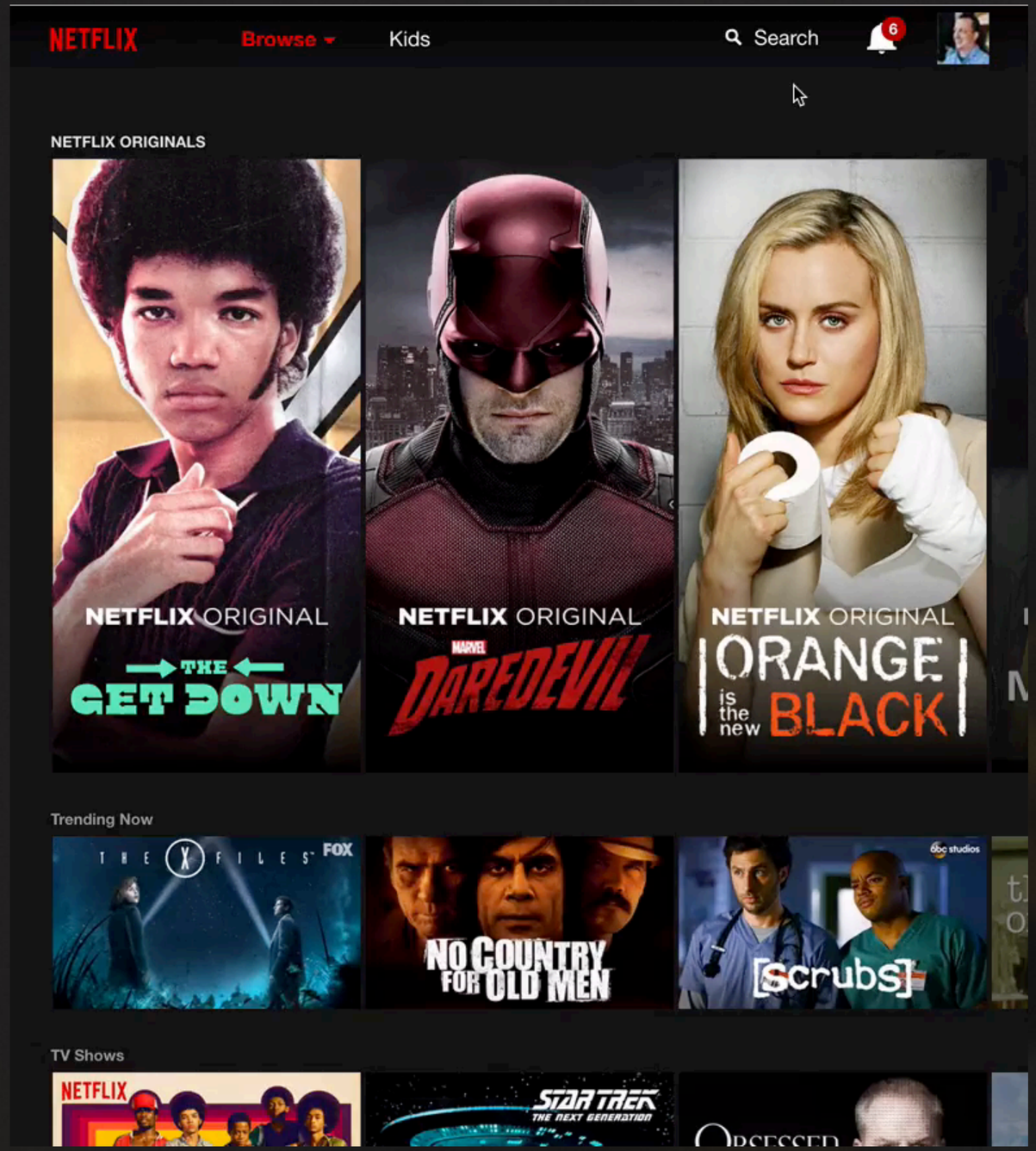

NETFLIX
Browse
Kids
Search
6
NETFLIX ORIGINALS
NETFLIX ORIGINAL
THE GET DOWN
NETFLIX ORIGINAL
MARVEL DAREDEVIL
NETFLIX ORIGINAL
ORANGE is the new BLACK
Trending Now
THE X FILES
FOX
NO COUNTRY FOR OLD MEN
abc studios
[scrubs]
TV Shows
NETFLIX
STAR TREK
THE NEXT GENERATION

Daredevil

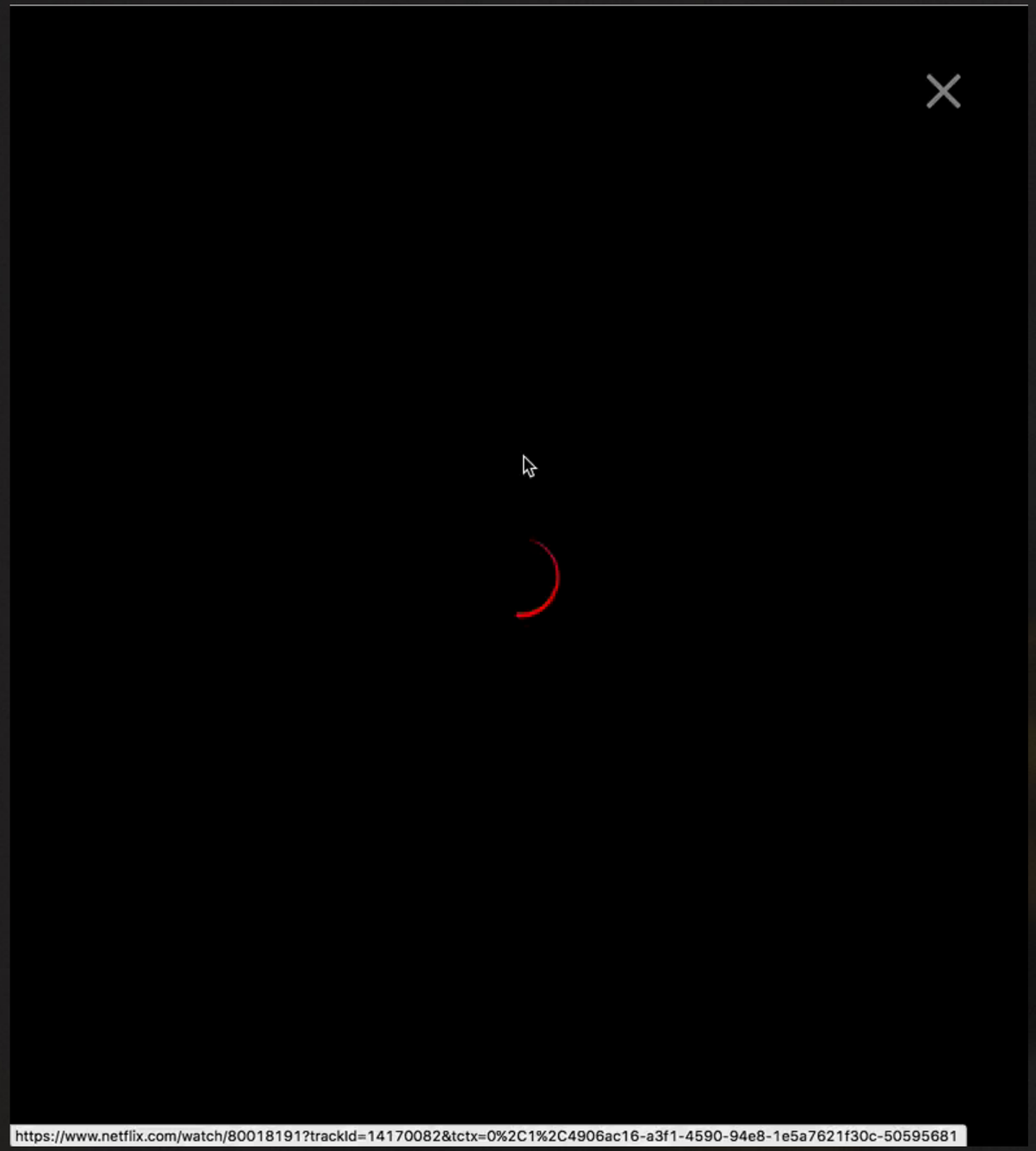


Here's Daredevil!

The Get Down


NETFLIX
Browse
Kids
Search
6
NETFLIX ORIGINALS
NETFLIX ORIGINAL
THE GET DOWN
NETFLIX ORIGINAL
MARVEL DAREDEVIL
NETFLIX ORIGINAL
ORANGE is the new BLACK
Trending Now
THE X FILES
FOX
NO COUNTRY FOR OLD MEN
abc studios
[scrubs]
TV Shows
NETFLIX
STAR TREK
THE NEXT GENERATION

Daredevil

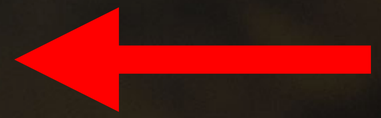

https://www.netflix.com/watch/80018191?trackId=14170082&tctx=0%2C1%2C4906ac16-a3f1-4590-94e8-1e5a7621f30c-50595681

The Get Down

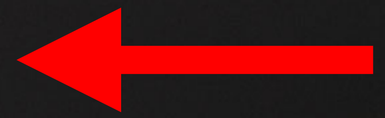
Cancelling is **common** and often **overlooked**

Promises

- **Guaranteed future**
- Immutable
- **Single value**
- Caching



Only AJAX is single value

- User interactions (mouse, keyboard, etc
- **AJAX** 
- Animations
- WebSockets, Workers, etc

What do we use?

Observables

Observables

- Stream of zero, one, or more values
- Over any amount of time
- Cancellable

Streams are a **set**, with a **dimension of time**

Being **standardized** for ECMAScript aka **JavaScript**

RxJS

“lodash for async” - Ben Lesh

Crash Course

Creating Observables

- `of('hello')`
- `from([1, 2, 3, 4])`
- `interval(1000)`
- `ajax('http://example.com')`
- `webSocket('ws://echo.websocket.com')`
- `fromEvent(button, ‘click')`
- many more...

Subscribing

```
myObservable.subscribe(
  value => console.log('next', value)
);
```

Subscribing

```
myObservable.subscribe(
  value => console.log('next', value),
  err => console.error('error', err)
);
```

Subscribing

```
myObservable.subscribe(
  value => console.log('next', value),
  err => console.error('error', err),
  () => console.info('complete!')
);
```

Observables can be *transformed*

map, filter, reduce

Observables can be *combined*

concat, merge, zip

Observables represent *time*

debounce, throttle, buffer, combineLatest

Observables are *lazy*

retry, repeat

Observables can **represent** just about **anything**

Let’s combine **RxJS** and **Redux**!

redux-observable

Side effect management for redux, using **Epics**

What is an Epic?

A **function** that takes a stream of **all actions** dispatched and returns a stream of **new actions** to dispatch

“actions in, actions out”

Sort of like this

```
// This is pseudo code, not real
function pingPong(action, store) {
  if (action.type === 'PING') {
    return { type: 'PONG' };
  }
}
```

An Epic

```
function pingPongEpic(action$, store) {
  return action$.ofType('PING')
    .map(action => ({ type: 'PONG' }));
}
```

An Epic

```
const pingPongEpic = (action$, store) =>
  action$.ofType('PING')
    .map(action => ({ type: 'PONG' }));
```

An Epic

```
const pingPongEpic = (action$, store) =>
  action$.ofType('PING')
    .delay(1000) // <- that's it
    .map(action => ({ type: 'PONG' }));
```

```
const isPinging = (state = false, action) => {
  switch (action.type) {
    case 'PING':
      return true;

    case 'PONG':
      return false;

    default:
      return state;
  }
};
```


```
const pingPongEpic = (action$, store) =>
  action$.ofType('PING')
    .delay(1000)
    .map(action => ({ type: 'PONG' }));
```

Debounced increment / decrement button

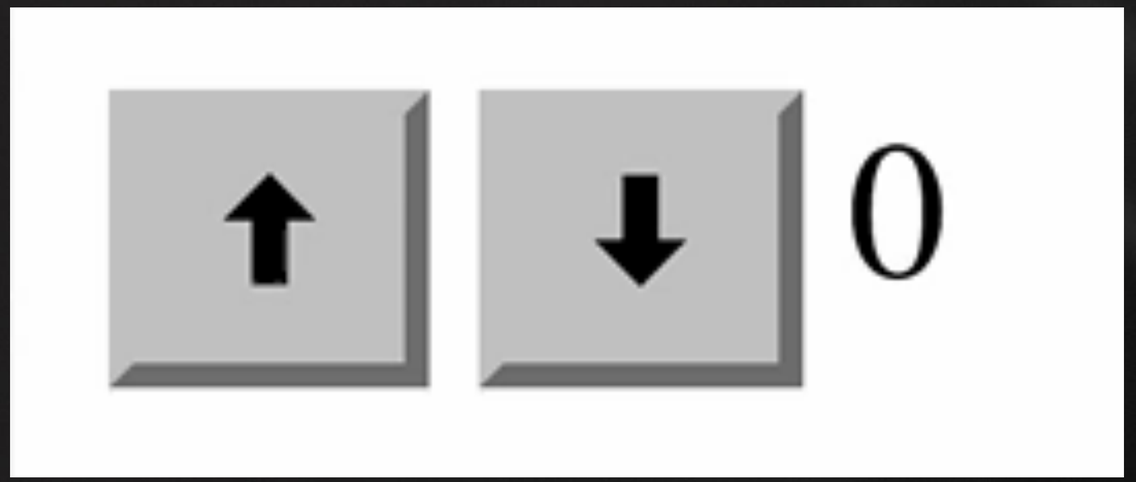
```
const counter = (state = 0, action) => {
  switch (action.type) {
    case 'INCREMENT':
      return state + 1;

    case 'DECREMENT':
      return state - 1;

    default:
      return state;
  }
};
```

```
const incrementEpic = (action$, store) =>
  action$.ofType('INCREMENT_DEBOUNCED')
    .debounceTime(1000)
    .map(() => ({ type: 'INCREMENT' }));

const decrementEpic = (action$, store) =>
  action$.ofType('DECREMENT_DEBOUNCED')
    .debounceTime(1000)
    .map(() => ({ type: 'DECREMENT' }));
```




```
const incrementEpic = (action$, store) =>
  action$.ofType('INCREMENT_DEBOUNCED')
    .debounceTime(1000)
    .map(() => ({ type: 'INCREMENT' }));

const decrementEpic = (action$, store) =>
  action$.ofType('DECREMENT_DEBOUNCED')
    .debounceTime(1000)
    .map(() => ({ type: 'DECREMENT' }));
```

Those are contrived examples, obviously

Warning: non-trivial examples ahead,
don’t struggle to read them entirely

Auto-complete

Plain JS

```
onKeyUp(e) {
  const { store } = this.props;
  const { value } = e.target.value;

  if (this.queryId) {
    clearTimeout(this.queryId);
  }

  this.queryId = setTimeout(() => {
    if (this.xhr) {
      this.xhr.abort();
    }

    const xhr = this.xhr = new XMLHttpRequest();
    xhr.open('GET', 'https://api.github.com/search/users?q=' + value);
    xhr.onload = () => {
      if (xhr.status === 200) {
        store.dispatch({
          type: 'QUERY_FULFILLED',
          payload: JSON.parse(xhr.response).items
        });
      } else {
        store.dispatch({
          type: 'QUERY_REJECTED',
          error: true,
          payload: {
            message: xhr.response,
            status: xhr.status
          }
        });
      }
    };
    xhr.send();
  }, 500);
}
```

Epic

```
const autoCompleteEpic = (action$, store) =>
  action$.ofType('QUERY')
    .debounceTime(500)
    .switchMap(action =>
      ajax('https://api.github.com/search/users?q=' + value)
        .map(payload => ({
          type: 'QUERY_FULFILLED',
          payload
        }))
    );
```

Epic

```
const autoCompleteEpic = (action$, store) =>
  action$.ofType('QUERY')
    .debounceTime(500)
    .switchMap(action =>
      ajax('https://api.github.com/search/users?q=' + value)
        .map(payload => ({
          type: 'QUERY_FULFILLED',
          payload
        }))
        .catch(payload => [{
          type: 'QUERY_REJECTED',
          error: true,
          payload
        }])
    );
```

Epic

```
const autoCompleteEpic = (action$, store) =>
  action$.ofType('QUERY')
    .debounceTime(500)
    .switchMap(action =>
      ajax('https://api.github.com/search/users?q=' + value)
        .map(payload => ({
          type: 'QUERY_FULFILLED',
          payload
        }))
        .takeUntil(action$.ofType('CANCEL_QUERY'))
        .catch(payload => [{
          type: 'QUERY_REJECTED',
          error: true,
          payload
        }])
    );
```

OK, show me ***really*** non-trivial examples

Bidirectional, multiplexed **Web Sockets**

Plain JS

```
class Example {
  @autobind
  checkChange(e) {
    const { value: key, checked } = e.target;
    this.subs = this.subs || [];

    if (checked) {
      const handler = e => {
        const data = JSON.parse(e.data);
        if (data.key === key) {
          this.updateValue(key, data.value);
        }
      };

      this.subs.push({ key, handler })

      const socket = this.getSocket(() => {
        this.setState({
          socketOpen: true
        });
        this.subs.forEach(({ key }) => socket.send(JSON.stringify({ type: 'sub', key})));
      });

      socket.addEventListener('message', handler);
    } else {
      const index = this.subs.findIndex(x => x.key === key);
      if (index !== -1) {
        this.subs.splice(index, 1);
      }

      const { socket } = this;
      if (socket && socket.readyState === 1) {
        socket.send(JSON.stringify({ type: 'unsub', key }));
        this.setInactive(key)1
        if (this.subs.length === 0) {
          socket.close();
        }
      }
    }
  }

  componentWillUnMount() {
    if (this.socket && this.socket.readyState === 1) {
      this.socket.close();
    }
  }
```

```
  getSocket(callback) {
    const { socket } = this;
    if (socket && socket.readyState === 1) {
      setTimeot(callback);
    } else {
      if (this.reconnectId) {
        clearTimeout(this.reconnectId);
      }

      socket = this.socket = new WebSocket('ws://localhost:3000');
      socket.onopen = () => {
        callback();
      };
      socket.onerror = () => {
        this.reconnectId = setTimeout(() => this.getSocket(callback), 1000);
        this.setState({ socketOpen: false });
      };
      socket.onclose = (e) => {
        if (!e.wasClean) {
          this.reconnectId = setTimeout(() => this.getSocket(callback), 1000);
        }
        this.setState({ socketOpen: false });
      };
    }

    return socket;
  }
}
```

Too much code

As an Epic

```
const socket = WebSocketSubject.create('ws://stock/endpoint');

const stockTickerEpic = (action$, store) =>
  action$.ofType('START_TICKER_STREAM')
    .mergeMap(action =>
      socket.multiplex(
        () => ({ sub: action.ticker }),
        () => ({ unsub: action.ticker }),
        msg => msg.ticker === action.ticker
      )
      .retryWhen(
        err => window.navigator.onLine ?
          Observable.timer(1000) :
          Observable.fromEvent(window, 'online')
      )
      .takeUntil(
        action$.ofType('CLOSE_TICKER_STREAM')
          .filter(closeAction => closeAction.ticker === action.ticker)
      )
      .map(tick => ({ type: 'TICKER_TICK', tick }))
    );
```

redux-observable

- Makes it easier to compose and control **complex async** tasks, over **any amount of time**
- You don't need to manage your own Rx subscriptions
- You can use redux tooling

But...

You should *probably* know **redux** and **RxJS** in advance

RxJS has a bit of a learning curve

“Reactive Programming”

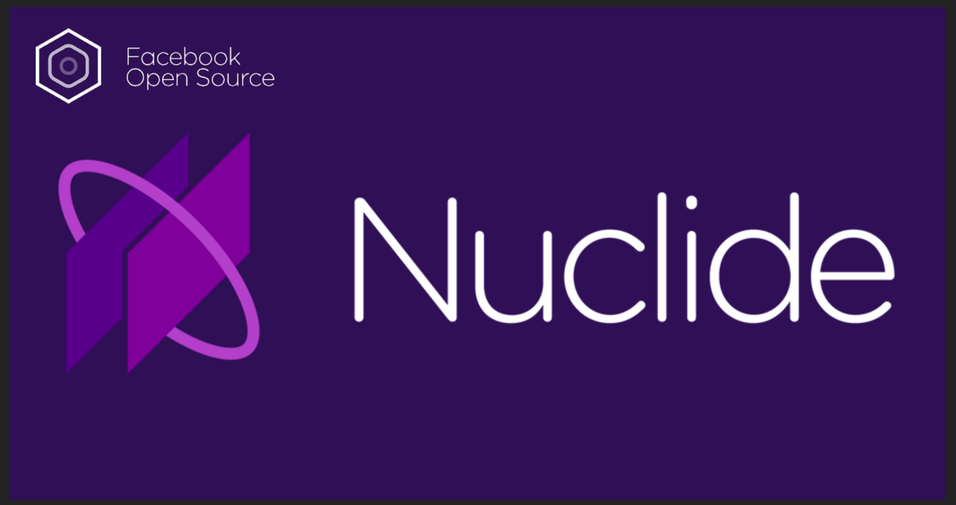
Co-author

Ben Lesh

Senior UI Engineer | NETFLIX

@benlesh

https://**redux-observable.js.org**


Facebook
Open Source
Nuclide


freeCodeCamp


slack


RANGLE.IO

NETFLIX

Thanks!

 @_jayphelps