SIDE EFFECTS MADE SIMPLE WITH

REDUX-SAGAS

WHY SAGAS?

REACT IS (CAN BE) PURE

- Pure functions
- Pure components

WHAT ARE PURE FUNCTIONS?

PURE FUNCTIONS ARE…

- Deterministic
  - Given the same input, they will return the same output
  - Do not rely on external mutable state
- No side-effects / Immutable
  - Do not mutate any external state

NON-PURE FUNCTIONS

```
// non-pure function (non-deterministic and mutable)
function increment(obj) {
  obj.count = obj.count ? object.count + incrementVal : incrementVal;
  return obj;
}

> var incrementVal = 1;
> const myObject = { count: 41 };
> let incrementedObject = increment(myObject);
> console.log(incrementedObject.count); // 42
> console.log(myObject.count); // 42

> incrementVal = 10;
> incrementedObject = increment(myObject);
> console.log(incrementedObject.count); // 52
> console.log(myObject.count); // 52
```

NON-PURE FUNCTIONS

```
// still a non-pure function (deterministic but still mutable)
function increment(obj, incrementVal) {
  obj.count = obj.count ? object.count + incrementVal : incrementVal;
  return obj;
}

> const incrementVal = 1;
> const myObject = { count: 41 };
> let incrementedObject = increment(myObject, incrementVal);
> console.log(incrementedObject.count); // 42
> console.log(myObject.count); // 42
```

PURE FUNCTIONS

```
// pure function (deterministic and immutable)
function increment(obj, incrementVal) {
  return {
    ...obj,
    count: obj.count ? object.count + incrementVal : incrementVal,
  }
}

> const incrementVal = 1;
> const myObject = { count: 41 };
> let incrementedObject = increment(myObject, incrementVal);
> console.log(incrementedObject.count); // 42
> console.log(myObject.count); // 41

> incrementedObject = increment(myObject, incrementVal);
> console.log(incrementedObject.count); // 42
> console.log(myObject.count); // 41
```

WHY PURE FUNCTIONS?

PURE FUNCTIONS ARE…

- Easier to Understand
  - No “hidden” state (what you see is what you get)
- Easier to Test
  - Given the same argument values, they will always return the same result (deterministic)
- Easier to Refactor
  - Pure functions are more independent, they can be moved around more easily, and with unit testing, can be refactored with greater confidence.

OK. BACK TO REDUX-SAGAS

REACT IS (CAN BE) PURE

- Pure functions
- Pure components

REDUX IS PURE

- Pure actions
- Pure reducers

WAIT...WHAT IS REDUX?

WHAT IS REDUX?

- Redux is a state management system for JavaScript apps

REDUX THREE PRINCIPLES

- Single source of truth
  - All application state is stored within a single store
- State is read-only
  - The only way to change the state is to dispatch / emit an action
- Changes are made with pure functions
  - Transformations to the state occur within pure reducers

IN SUMMARY...

SUMMARY

- Our functions are pure
- Our React components are pure
- Our Redux actions and reducers are pure
- but…what about our API calls?
  - we can’t guarantee that our API calls are going to be pure operations

SO WHERE TO PUT THE SIDE-EFFECTS?

REDUX-SAGA IS A LIBRARY THAT AIMS TO MAKE SIDE EFFECTS (I.E. ASYNCHRONOUS THINGS LIKE DATA FETCHING AND IMPURE THINGS LIKE ACCESSING THE BROWSER CACHE) IN REACT/REDUX APPLICATIONS EASIER AND BETTER.

https://redux-saga.js.org/

WHAT ARE SAGAS?

WHAT ARE SAGAS?

- Sagas are ES6 generator leveraged Ajax handlers

WHAT ARE GENERATORS?

WHAT ARE GENERATORS?

- Generators are functions which can be exited and later re-entered.

```
// my generator
function* myGenerator() {
  yield;
  console.log('fizz');
  yield;
  console.log('buzz');
  yield;
  console.log('fizz buzz');
}

> const gen = myGenerator();
> gen.next();
> gen.next();
// fizz
> gen.next();
// buzz
> gen.next();
// fizz buzz
```

WHAT ARE GENERATORS?

- Generators are functions which can be exited and later re-entered.

```
// my generator
function* myGenerator() {
  yield;
  console.log('fizz');
  yield;
  console.log('buzz');
  yield;
  console.log('fizz buzz');
}

const gen = myGenerator();
> gen.next();
> gen.next();
// fizz
> gen.next();
// buzz
> gen.next();
// fizz buzz
```

- Generators are functions which can be exited and later re-entered.

```
// my generator
function* myGenerator() {
  yield;
  console.log('fizz');
  yield;
  console.log('buzz');
  yield;
  console.log('fizz buzz');
}

const gen = myGenerator();
> gen.next();
> gen.next();
// fizz
> gen.next();
// buzz
> gen.next();
// fizz buzz
```

- Generators are functions which can be exited and later re-entered.

```
// my generator
function* myGenerator() {
  yield;
  console.log('fizz');
  yield;
  console.log('buzz');
  yield;
  console.log('fizz buzz');
}

> const gen = myGenerator();
  gen.next();
> gen.next();
// fizz
> gen.next();
// buzz
> gen.next();
// fizz buzz
```

- Generators are functions which can be exited and later re-entered.

```
// my generator
function* myGenerator() {
  yield;
  console.log('fizz');
  yield;
  console.log('buzz');
  yield;
  console.log('fizz buzz');
}

> const gen = myGenerator();
  gen.next();
> gen.next();
// fizz
> gen.next();
// buzz
> gen.next();
// fizz buzz
```

- Generators are functions which can be exited and later re-entered.

```
// my generator
function* myGenerator() {
  yield;
  console.log('fizz');
  yield;
  console.log('buzz');
  yield;
  console.log('fizz buzz');
}

> const gen = myGenerator();
> gen.next();
> gen.next();
// fizz
> gen.next();
// buzz
> gen.next();
// fizz buzz
```

- Generators are functions which can be exited and later re-entered.

```
// my generator
function* myGenerator() {
  yield;
  console.log('fizz');
  yield;
  console.log('buzz');
  yield;
  console.log('fizz buzz');
}

> const gen = myGenerator();
> gen.next();
> gen.next();
// fizz
> gen.next();
// buzz
> gen.next();
// fizz buzz
```

- Generators are functions which can be exited and later re-entered.

```
// my generator
function* myGenerator() {
  yield;
  console.log('fizz');
  yield;
  console.log('buzz');
  yield;
  console.log('fizz buzz');
}

> const gen = myGenerator();
> gen.next();
> gen.next();
// fizz
> gen.next();
// buzz
> gen.next();
// fizz buzz
```

- Generators are functions which can be exited and later re-entered.

```
// my generator
function* myGenerator() {
  yield;
  console.log('fizz');
  yield;
  console.log('buzz');
  yield;
  console.log('fizz buzz');
}

> const gen = myGenerator();
> gen.next();
> gen.next();
// fizz
  gen.next();
// buzz
> gen.next();
// fizz buzz
```

- Generators are functions which can be exited and later re-entered.

```
// my generator
function* myGenerator() {
  yield;
  console.log('fizz');
  yield;
  console.log('buzz');
  yield;
  console.log('fizz buzz');
}

> const gen = myGenerator();
> gen.next();
> gen.next();
// fizz
> gen.next();
// buzz
> gen.next();
// fizz buzz
```

- Generators are functions which can be exited and later re-entered.

```
// my generator
function* myGenerator() {
  yield;
  console.log('fizz');
  yield;
  console.log('buzz');
  yield;
  console.log('fizz buzz');
}

> const gen = myGenerator();
> gen.next();
> gen.next();
// fizz
> gen.next();
// buzz
> gen.next();
// fizz buzz
```

- Generators are functions which can be exited and later re-entered.

```
// my generator
function* myGenerator() {
  yield;
  console.log('fizz');
  yield;
  console.log('buzz');
  yield;
  console.log('fizz buzz');
}

> const gen = myGenerator();
> gen.next();
> gen.next();
// fizz
> gen.next();
// buzz
> gen.next();
// fizz buzz
```

- Generators are functions which can be exited and later re-entered.

```
// my generator
function* myGenerator() {
  yield;
  console.log('fizz');
  yield;
  console.log('buzz');
  yield;
  console.log('fizz buzz');
}

> const gen = myGenerator();
> gen.next();
> gen.next();
// fizz
> gen.next();
// buzz
> gen.next();
// fizz buzz
```

- Generators are functions which can be exited and later re-entered.

```
// my generator
function* myGenerator() {
  yield;
  console.log('fizz');
  yield;
  console.log('buzz');
  yield;
  console.log('fizz buzz');
}

> const gen = myGenerator();
> gen.next();
> gen.next();
// fizz
> gen.next();
// buzz
> gen.next();
// fizz buzz
```

HOW ARE SAGAS USED?

SAGAS 'WATCH' FOR DISPATCHED ACTIONS

- Given an action type: `FETCH_ALL_THE_THINGS`
- We can create a saga: `watchFetchAllTheThings`
- that will be triggered whenever `FETCH_ALL_THE_THINGS` is dispatched

SAMPLE COMPONENT

SAMPLE COMPONENT

```
// my component
import React from 'react';
import PropTypes from 'prop-types';
import { bindActionCreators } from 'redux';
import { connect } from 'react-redux';
import { fetchAllTheThings as fetchAllTheThingsAction } from './actions';

function mySuperAwesomeComponent({ fetchAllTheThings }) {
  return (
    <div onClick={event => fetchAllTheThings()}>
      Fetch the Things
    </div>
  );
}

mySuperAwesomeComponent.propTypes = {
  fetchAllTheThings: PropTypes.func.isRequired,
};

function mapDispatchToProps(dispatch) {
  return bindActionCreators({
    fetchAllTheThings: fetchAllTheThingsAction,
  }, dispatch);
}

export default connect(null, mapDispatchToProps)(mySuperAwesomeComponent);
```

SAMPLE COMPONENT

```
1   // my component
2   import React from 'react';
3   import PropTypes from 'prop-types';
4   import { bindActionCreators } from 'redux';
5   import { connect } from 'react-redux';
6   import { fetchAllTheThings as fetchAllTheThingsAction } from './actions';
7
8   function mySuperAwesomeComponent({ fetchAllTheThings }) {
9     return (
10      <div onClick={event => fetchAllTheThings()}>
11        Fetch the Things
12      </div>
13    );
14  }
15
16  mySuperAwesomeComponent.propTypes = {
17    fetchAllTheThings: PropTypes.func.isRequired,
18  };
19
20  function mapDispatchToProps(dispatch) {
21    return bindActionCreators({
22      fetchAllTheThings: fetchAllTheThingsAction,
23    }, dispatch);
24  }
25
26  export default connect(null, mapDispatchToProps)(mySuperAwesomeComponent);
```

SAMPLE COMPONENT

```
1  // my component
2  import React from 'react';
3  import PropTypes from 'prop-types';
4  import { bindActionCreators } from 'redux';
5  import { connect } from 'react-redux';
6  import { fetchAllTheThings as fetchAllTheThingsAction } from './actions';
7
8  function mySuperAwesomeComponent({ fetchAllTheThings }) {
9    return (
10     <div onClick={event => fetchAllTheThings()}>
11       Fetch the Things
12     </div>
13   );
14 }
15
16 mySuperAwesomeComponent.propTypes = {
17   fetchAllTheThings: PropTypes.func.isRequired,
18 };
19
20 function mapDispatchToProps(dispatch) {
21   return bindActionCreators({
22     fetchAllTheThings: fetchAllTheThingsAction,
23   }, dispatch);
24 }
25
26 export default connect(null, mapDispatchToProps)(mySuperAwesomeComponent);
```

SAMPLE COMPONENT

```
// my component
import React from 'react';
import PropTypes from 'prop-types';
import { bindActionCreators } from 'redux';
import { connect } from 'react-redux';
import { fetchAllTheThings as fetchAllTheThingsAction } from './actions';

function mySuperAwesomeComponent({ fetchAllTheThings }) {
  return (
    <div onClick={event => fetchAllTheThings()}>
      Fetch the Things
    </div>
  );
}

mySuperAwesomeComponent.propTypes = {
  fetchAllTheThings: PropTypes.func.isRequired,
};

function mapDispatchToProps(dispatch) {
  return bindActionCreators({
    fetchAllTheThings: fetchAllTheThingsAction,
  }, dispatch);
}

export default connect(null, mapDispatchToProps)(mySuperAwesomeComponent);
```

SAMPLE COMPONENT

```
1  // my component
2  import React from 'react';
3  import PropTypes from 'prop-types';
4  import { bindActionCreators } from 'redux';
5  import { connect } from 'react-redux';
6  import { fetchAllTheThings as fetchAllTheThingsAction } from './actions';
7
8  function mySuperAwesomeComponent({ fetchAllTheThings }) {
9    return (
10     <div onClick={event => fetchAllTheThings()}>
11       Fetch the Things
12     </div>
13   );
14 }
15
16 mySuperAwesomeComponent.propTypes = {
17   fetchAllTheThings: PropTypes.func.isRequired,
18 };
19
20 function mapDispatchToProps(dispatch) {
21   return bindActionCreators({
22     fetchAllTheThings: fetchAllTheThingsAction,
23   }, dispatch);
24 }
25
26 export default connect(null, mapDispatchToProps)(mySuperAwesomeComponent);
```

SAMPLE SAGA

SAMPLE SAGA

```
1   // my saga
2   import { call, takeLatest, put } from 'redux-saga/effects';
3   import { request } from './request';
4   import { transformTheData } from './mappers';
5   import * as actions from './actions';
6
7   function* fetchAllTheThings() {
8     try {
9       const response = yield call(request, '/fetchThings', 'GET');
10      if (response.status < 400) {
11        const transformedData = yield call(transformTheData, response.data);
12        yield put(actions.storeAllTheThings(transformedData));
13      } else {
14        yield put(actions.handleError());
15      }
16    }
17  }
18
19  function* watchFetchAllTheThings() {
20    yield takeLatest(FETCH_ALL_THE_THINGS, fetchAllTheThings);
21  }
22
23  export { watchFetchAllTheThings, fetchAllTheThings };
```

SAMPLE SAGA

```
1   // my saga
2   import { call, takeLatest, put } from 'redux-saga/effects';
3   import { request } from './request';
4   import { transformTheData } from './mappers';
5   import * as actions from './actions';
6
7   function* fetchAllTheThings() {
8     try {
9       const response = yield call(request, '/fetchThings', 'GET');
10      if (response.status < 400) {
11        const transformedData = yield call(transformTheData, response.data);
12        yield put(actions.storeAllTheThings(transformedData));
13      } else {
14        yield put(actions.handleError());
15      }
16    }
17  }
18
19  function* watchFetchAllTheThings() {
20    yield takeLatest(FETCH_ALL_THE_THINGS, fetchAllTheThings);
21  }
22
23  export { watchFetchAllTheThings, fetchAllTheThings };
```

SAMPLE SAGA

```
1   // my saga
2   import { call, takeLatest, put } from 'redux-saga/effects';
3   import { request } from './request';
4   import { transformTheData } from './mappers';
5   import * as actions from './actions';
6
7   function* fetchAllTheThings() {
8     try {
9       const response = yield call(request, '/fetchThings', 'GET');
10      if (response.status < 400) {
11        const transformedData = yield call(transformTheData, response.data);
12        yield put(actions.storeAllTheThings(transformedData));
13      } else {
14        yield put(actions.handleError());
15      }
16    }
17  }
18
19  function* watchFetchAllTheThings() {
20    yield takeLatest(FETCH_ALL_THE_THINGS, fetchAllTheThings);
21  }
22
23  export { watchFetchAllTheThings, fetchAllTheThings };
```

SAMPLE SAGA

```
1   // my saga
2   import { call, takeLatest, put } from 'redux-saga/effects';
3   import { request } from './request';
4   import { transformTheData } from './mappers';
5   import * as actions from './actions';
6
7   function* fetchAllTheThings() {
8     try {
9       const response = yield call(request, '/fetchThings', 'GET');
10      if (response.status < 400) {
11        const transformedData = yield call(transformTheData, response.data);
12        yield put(actions.storeAllTheThings(transformedData));
13      } else {
14        yield put(actions.handleError());
15      }
16    }
17  }
18
19  function* watchFetchAllTheThings() {
20    yield takeLatest(FETCH_ALL_THE_THINGS, fetchAllTheThings);
21  }
22
23  export { watchFetchAllTheThings, fetchAllTheThings };
```

SAMPLE SAGA

```
1   // my saga
2   import { call, takeLatest, put } from 'redux-saga/effects';
3   import { request } from './request';
4   import { transformTheData } from './mappers';
5   import * as actions from './actions';
6
7   function* fetchAllTheThings() {
8     try {
9       const response = yield call(request, '/fetchThings', 'GET');
10      if (response.status < 400) {
11        const transformedData = yield call(transformTheData, response.data);
12        yield put(actions.storeAllTheThings(transformedData));
13      } else {
14        yield put(actions.handleError());
15      }
16    }
17  }
18
19  function* watchFetchAllTheThings() {
20    yield takeLatest(FETCH_ALL_THE_THINGS, fetchAllTheThings);
21  }
22
23  export { watchFetchAllTheThings, fetchAllTheThings };
```

SAMPLE SAGA

```
1   // my saga
2   import { call, takeLatest, put } from 'redux-saga/effects';
3   import { request } from './request';
4   import { transformTheData } from './mappers';
5   import * as actions from './actions';
6
7   function* fetchAllTheThings() {
8     try {
9       const response = yield call(request, '/fetchThings', 'GET');
10      if (response.status < 400) {
11        const transformedData = yield call(transformTheData, response.data);
12        yield put(actions.storeAllTheThings(transformedData));
13      } else {
14        yield put(actions.handleError());
15      }
16    }
17  }
18
19  function* watchFetchAllTheThings() {
20    yield takeLatest(FETCH_ALL_THE_THINGS, fetchAllTheThings);
21  }
22
23  export { watchFetchAllTheThings, fetchAllTheThings };
```

SAMPLE SAGA

```
1   // my saga
2   import { call, takeLatest, put } from 'redux-saga/effects';
3   import { request } from './request';
4   import { transformTheData } from './mappers';
5   import * as actions from './actions';
6
7   function* fetchAllTheThings() {
8     try {
9       const response = yield call(request, '/fetchThings', 'GET');
10      if (response.status < 400) {
11        const transformedData = yield call(transformTheData, response.data);
12        yield put(actions.storeAllTheThings(transformedData));
13      } else {
14        yield put(actions.handleError());
15      }
16    }
17  }
18
19  function* watchFetchAllTheThings() {
20    yield takeLatest(FETCH_ALL_THE_THINGS, fetchAllTheThings);
21  }
22
23  export { watchFetchAllTheThings, fetchAllTheThings };
```

SAMPLE SAGA

```
1   // my saga
2   import { call, takeLatest, put } from 'redux-saga/effects';
3   import { request } from './request';
4   import { transformTheData } from './mappers';
5   import * as actions from './actions';
6
7   function* fetchAllTheThings() {
8     try {
9       const response = yield call(request, '/fetchThings', 'GET');
10      if (response.status < 400) {
11        const transformedData = yield call(transformTheData, response.data);
12        yield put(actions.storeAllTheThings(transformedData));
13      } else {
14        yield put(actions.handleError());
15      }
16    }
      }
18
19  function* watchFetchAllTheThings() {
20    yield takeLatest(FETCH_ALL_THE_THINGS, fetchAllTheThings);
21  }
22
23  export { watchFetchAllTheThings, fetchAllTheThings };
```

TESTING SAGAS

TESTING SAGAS

```
1   // saga test
2   import { call, takeLatest, put } from 'redux-saga/effects';
3   import { request } from './request';
4   import { transformTheData } from './mappers';
5   import * as actions form './actions';
6
7   describe('fetchAllTheThings sagas', () => {
8     describe(watchFetchAllTheThings, () => {
9       it('calls takelatest on FETCH_ALL_THE_THINGS action', () => {
10        const generator = watchFetchAllTheThings();
11        let next = generator.next();
12        expect(next.value).toEqual(takeLatest(FETCH_ALL_THE_THINGS, fetchAllTheThings));
13        next = generator.next();
14        expect(next).toEqual({ done: true, value: undefined });
15      });
16    });
17
18    describe('fetchAllTheThings', () => {
19      let generator;
20      let next;
21      beforeEach(() => {
22        generator = fetchAllTheThings();
23        next = generator.next();
24        expect(next.value).toEqual(call(request, '/fetchThings', 'GET'));
25      });
26
27      it('handles successful api calls', () => {
28        const response = {
29          status: 200,
30          data: { we: 'win!' },
31        };
32        const transformedData = { showTrophy: true };
33
34        next = generator.next(response);
35        expect(next).toEqual(call(transformTheData, response.data));
36        next = generator.next(transformedData);
37        expect(next).toEqual(put(actions.storeAllTheThings(transformedData)));
38        next = generator.next();
39        expect(next).toEqual({ done: true, value: undefined });
40      });
41
42      it('handles unsuccessful api calls', () => {
43        const response = {
44          status: 403,
45        };
46
47        next = generator.next(response);
48        expect(next).toEqual(put(actions.handleError()));
49        next = generator.next();
50        expect(next).toEqual({ done: true, value: undefined });
51      })
52    });
53  }
```

TESTING SAGAS

```
1   // saga test
2   import { call, takeLatest, put } from 'redux-saga/effects';
3   import { request } from './request';
4   import { transformTheData } from './mappers';
5   import * as actions form './actions';
6
7   describe('fetchAllTheThings sagas', () => {
8     describe(watchFetchAllTheThings, () => {
9       it('calls takelatest on FETCH_ALL_THE_THINGS action', () => {
10        const generator = watchFetchAllTheThings();
11        let next = generator.next();
12        expect(next.value).toEqual(takeLatest(FETCH_ALL_THE_THINGS, fetchAllTheThings));
13        next = generator.next();
14        expect(next).toEqual({ done: true, value: undefined });
15      });
16    });
17
18    describe('fetchAllTheThings', () => {
19      let generator;
20      let next;
21      beforeEach(() => {
22        generator = fetchAllTheThings();
23        next = generator.next();
24        expect(next.value).toEqual(call(request, '/fetchThings', 'GET'));
25      });
26
27      it('handles successful api calls', () => {
28        const response = {
29          status: 200,
30          data: { we: 'win!' },
31        };
32        const transformedData = { showTrophy: true };
33
34        next = generator.next(response);
35        expect(next).toEqual(call(transformTheData, response.data));
36        next = generator.next(transformedData);
37        expect(next).toEqual(put(actions.storeAllTheThings(transformedData)));
38        next = generator.next();
39        expect(next).toEqual({ done: true, value: undefined });
40      });
41
42      it('handles unsuccessful api calls', () => {
43        const response = {
44          status: 403,
45        };
46
47        next = generator.next(response);
48        expect(next).toEqual(put(actions.handleError()));
49        next = generator.next();
50        expect(next).toEqual({ done: true, value: undefined });
51      })
52    });
53  }
```

TESTING SAGAS

```
1   // saga test
2   import { call, takeLatest, put } from 'redux-saga/effects';
3   import { request } from './request';
4   import { transformTheData } from './mappers';
5   import * as actions form './actions';
6
7   describe('fetchAllTheThings sagas', () => {
8     describe(watchFetchAllTheThings, () => {
9       it('calls takelatest on FETCH_ALL_THE_THINGS action', () => {
10        const generator = watchFetchAllTheThings();
11        let next = generator.next();
12        expect(next.value).toEqual(takeLatest(FETCH_ALL_THE_THINGS, fetchAllTheThings));
13        next = generator.next();
14        expect(next).toEqual({ done: true, value: undefined });
15      });
16    });
17
18    describe('fetchAllTheThings', () => {
19      let generator;
20      let next;
21      beforeEach(() => {
22        generator = fetchAllTheThings();
23        next = generator.next();
24        expect(next.value).toEqual(call(request, '/fetchThings', 'GET'));
25      });
26
27      it('handles successful api calls', () => {
28        const response = {
29          status: 200,
30          data: { we: 'win!' },
31        };
32        const transformedData = { showTrophy: true };
33
34        next = generator.next(response);
35        expect(next).toEqual(call(transformTheData, response.data));
36        next = generator.next(transformedData);
37        expect(next).toEqual(put(actions.storeAllTheThings(transformedData)));
38        next = generator.next();
39        expect(next).toEqual({ done: true, value: undefined });
40      });
41
42      it('handles unsuccessful api calls', () => {
43        const response = {
44          status: 403,
45        };
46
47        next = generator.next(response);
48        expect(next).toEqual(put(actions.handleError()));
49        next = generator.next();
50        expect(next).toEqual({ done: true, value: undefined });
51      })
52    });
53  }
```

TESTING THE WATCHER

```
it('calls takelatest on FETCH_ALL_THE_THINGS action', () => {
  const generator = watchFetchAllTheThings();
  let next = generator.next();
  expect(next.value).toEqual(takeLatest(FETCH_ALL_THE_THINGS, fetchAllTheThings));
  next = generator.next();
  expect(next).toEqual({ done: true, value: undefined });
});
```

```
function* watchFetchAllTheThings() {
  yield takeLatest(FETCH_ALL_THE_THINGS, fetchAllTheThings);
}
```

TESTING THE WATCHER

```
it('calls takelatest on FETCH_ALL_THE_THINGS action', () => {
  const generator = watchFetchAllTheThings();
  let next = generator.next();
  expect(next.value).toEqual(takeLatest(FETCH_ALL_THE_THINGS, fetchAllTheThings));
  next = generator.next();
  expect(next).toEqual({ done: true, value: undefined });
});
```

```
function* watchFetchAllTheThings() {
  yield takeLatest(FETCH_ALL_THE_THINGS, fetchAllTheThings);
}
```

TESTING THE WATCHER

```
it('calls takelatest on FETCH_ALL_THE_THINGS action', () => {
  const generator = watchFetchAllTheThings();
  let next = generator.next();
  expect(next.value).toEqual(takeLatest(FETCH_ALL_THE_THINGS, fetchAllTheThings));
  next = generator.next();
  expect(next).toEqual({ done: true, value: undefined });
});
```

```
function* watchFetchAllTheThings() {
  yield takeLatest(FETCH_ALL_THE_THINGS, fetchAllTheThings);
}
```

TESTING THE WATCHER

```
it('calls takelatest on FETCH_ALL_THE_THINGS action', () => {
  const generator = watchFetchAllTheThings();
  let next = generator.next();
  expect(next.value).toEqual(takeLatest(FETCH_ALL_THE_THINGS, fetchAllTheThings));
  next = generator.next();
  expect(next).toEqual({ done: true, value: undefined });
});
```

```
function* watchFetchAllTheThings() {
  yield takeLatest(FETCH_ALL_THE_THINGS, fetchAllTheThings);
}
```

TESTING THE WATCHER

```
it('calls takelatest on FETCH_ALL_THE_THINGS action', () => {
  const generator = watchFetchAllTheThings();
  let next = generator.next();
  expect(next.value).toEqual(takeLatest(FETCH_ALL_THE_THINGS, fetchAllTheThings));
  next = generator.next();
  expect(next).toEqual({ done: true, value: undefined });
});
```

```
function* watchFetchAllTheThings() {
  yield takeLatest(FETCH_ALL_THE_THINGS, fetchAllTheThings);
}
```

TESTING THE SAGA

```
beforeEach(() => {
  generator = fetchAllTheThings();
  next = generator.next();
  expect(next.value).toEqual(call(request, '/fetchThings', 'GET'));
});

it('handles successful api calls', () => {
  const response = {
    status: 200,
    data: { we: 'win!' },
  };
  const transformedData = { showTrophy: true };

  next = generator.next(response);
  expect(next).toEqual(call(transformTheData, response.data));
  next = generator.next(transformedData);
  expect(next).toEqual(put(actions.storeAllTheThings(transformedData)));
  next = generator.next();
  expect(next).toEqual({ done: true, value: undefined });
});
```

```
function* fetchAllTheThings() {
  try {
    const response = yield call(request, '/fetchThings', 'GET');
    if (response.status < 400) {
      const transformedData = yield call(transformTheData, response.data);
      yield put(actions.storeAllTheThings(transformedData));
    } else {
      yield put(actions.handleError());
    }
  }
}
```

TESTING THE SAGA

```
beforeEach(() => {
  generator = fetchAllTheThings();
  next = generator.next();
  expect(next.value).toEqual(call(request, '/fetchThings', 'GET'));
});

it('handles successful api calls', () => {
  const response = {
    status: 200,
    data: { we: 'win!' },
  };
  const transformedData = { showTrophy: true };

  next = generator.next(response);
  expect(next).toEqual(call(transformTheData, response.data));
  next = generator.next(transformedData);
  expect(next).toEqual(put(actions.storeAllTheThings(transformedData)));
  next = generator.next();
  expect(next).toEqual({ done: true, value: undefined });
});
```

```
function* fetchAllTheThings() {
  try {
    const response = yield call(request, '/fetchThings', 'GET');
    if (response.status < 400) {
      const transformedData = yield call(transformTheData, response.data);
      yield put(actions.storeAllTheThings(transformedData));
    } else {
      yield put(actions.handleError());
    }
  }
}
```

TESTING THE SAGA

```
beforeEach(() => {
  generator = fetchAllTheThings();
  next = generator.next();
  expect(next.value).toEqual(call(request, '/fetchThings', 'GET'));
});

it('handles successful api calls', () => {
  const response = {
    status: 200,
    data: { we: 'win!' },
  };
  const transformedData = { showTrophy: true };

  next = generator.next(response);
  expect(next).toEqual(call(transformTheData, response.data));
  next = generator.next(transformedData);
  expect(next).toEqual(put(actions.storeAllTheThings(transformedData)));
  next = generator.next();
  expect(next).toEqual({ done: true, value: undefined });
});
```

```
function* fetchAllTheThings() {
  try {
    const response = yield call(request, '/fetchThings', 'GET');
    if (response.status < 400) {
      const transformedData = yield call(transformTheData, response.data);
      yield put(actions.storeAllTheThings(transformedData));
    } else {
      yield put(actions.handleError());
    }
  }
}
```

TESTING THE SAGA

```
beforeEach(() => {
  generator = fetchAllTheThings();
  next = generator.next();
  expect(next.value).toEqual(call(request, '/fetchThings', 'GET'));
});

it('handles successful api calls', () => {
  const response = {
    status: 200,
    data: { we: 'win!' },
  };
  const transformedData = { showTrophy: true };

  next = generator.next(response);
  expect(next).toEqual(call(transformTheData, response.data));
  next = generator.next(transformedData);
  expect(next).toEqual(put(actions.storeAllTheThings(transformedData)));
  next = generator.next();
  expect(next).toEqual({ done: true, value: undefined });
});
```

```
function* fetchAllTheThings() {
  try {
    const response = yield call(request, '/fetchThings', 'GET');
    if (response.status < 400) {
      const transformedData = yield call(transformTheData, response.data);
      yield put(actions.storeAllTheThings(transformedData));
    } else {
      yield put(actions.handleError());
    }
  }
}
```

TESTING THE SAGA

```
beforeEach(() => {
  generator = fetchAllTheThings();
  next = generator.next();
  expect(next.value).toEqual(call(request, '/fetchThings', 'GET'));
});

it('handles successful api calls', () => {
  const response = {
    status: 200,
    data: { we: 'win!' },
  };
  const transformedData = { showTrophy: true };

  next = generator.next(response);
  expect(next).toEqual(call(transformTheData, response.data));
  next = generator.next(transformedData);
  expect(next).toEqual(put(actions.storeAllTheThings(transformedData)));
  next = generator.next();
  expect(next).toEqual({ done: true, value: undefined });
});
```

```
function* fetchAllTheThings() {
  try {
    const response = yield call(request, '/fetchThings', 'GET');
    if (response.status < 400) {
      const transformedData = yield call(transformTheData, response.data);
      yield put(actions.storeAllTheThings(transformedData));
    } else {
      yield put(actions.handleError());
    }
  }
}
```

TESTING THE SAGA

```
beforeEach(() => {
  generator = fetchAllTheThings();
  next = generator.next();
  expect(next.value).toEqual(call(request, '/fetchThings', 'GET'));
});

it('handles successful api calls', () => {
  const response = {
    status: 200,
    data: { we: 'win!' },
  };
  const transformedData = { showTrophy: true };

  next = generator.next(response);
  expect(next).toEqual(call(transformTheData, response.data));
  next = generator.next(transformedData);
  expect(next).toEqual(put(actions.storeAllTheThings(transformedData)));
  next = generator.next();
  expect(next).toEqual({ done: true, value: undefined });
});
```

```
function* fetchAllTheThings() {
  try {
    const response = yield call(request, '/fetchThings', 'GET');
    if (response.status < 400) {
      const transformedData = yield call(transformTheData, response.data);
      yield put(actions.storeAllTheThings(transformedData));
    } else {
      yield put(actions.handleError());
    }
  }
}
```

TESTING THE SAGA

```
beforeEach(() => {
  generator = fetchAllTheThings();
  next = generator.next();
  expect(next.value).toEqual(call(request, '/fetchThings', 'GET'));
});

it('handles successful api calls', () => {
  const response = {
    status: 200,
    data: { we: 'win!' },
  };
  const transformedData = { showTrophy: true };

  next = generator.next(response);
  expect(next).toEqual(call(transformTheData, response.data));
  next = generator.next(transformedData);
  expect(next).toEqual(put(actions.storeAllTheThings(transformedData)));
  next = generator.next();
  expect(next).toEqual({ done: true, value: undefined });
});
```

```
function* fetchAllTheThings() {
  try {
    const response = yield call(request, '/fetchThings', 'GET');
    if (response.status < 400) {
      const transformedData = yield call(transformTheData, response.data);
      yield put(actions.storeAllTheThings(transformedData));
    } else {
      yield put(actions.handleError());
    }
  }
}
```

TESTING THE SAGA

```
beforeEach(() => {
  generator = fetchAllTheThings();
  next = generator.next();
  expect(next.value).toEqual(call(request, '/fetchThings', 'GET'));
});

it('handles successful api calls', () => {
  const response = {
    status: 200,
    data: { we: 'win!' },
  };
  const transformedData = { showTrophy: true };

  next = generator.next(response);
  expect(next).toEqual(call(transformTheData, response.data));
  next = generator.next(transformedData);
  expect(next).toEqual(put(actions.storeAllTheThings(transformedData)));
  next = generator.next();
  expect(next).toEqual({ done: true, value: undefined });
});
```

```
function* fetchAllTheThings() {
  try {
    const response = yield call(request, '/fetchThings', 'GET');
    if (response.status < 400) {
      const transformedData = yield call(transformTheData, response.data);
      yield put(actions.storeAllTheThings(transformedData));
    } else {
      yield put(actions.handleError());
    }
  }
}
```

TESTING THE SAGA

```
beforeEach(() => {
  generator = fetchAllTheThings();
  next = generator.next();
  expect(next.value).toEqual(call(request, '/fetchThings', 'GET'));
});

it('handles successful api calls', () => {
  const response = {
    status: 200,
    data: { we: 'win!' },
  };
  const transformedData = { showTrophy: true };

  next = generator.next(response);
  expect(next).toEqual(call(transformTheData, response.data));
  next = generator.next(transformedData);
  expect(next).toEqual(put(actions.storeAllTheThings(transformedData)));
  next = generator.next();
  expect(next).toEqual({ done: true, value: undefined });
});
```

```
function* fetchAllTheThings() {
  try {
    const response = yield call(request, '/fetchThings', 'GET');
    if (response.status < 400) {
      const transformedData = yield call(transformTheData, response.data);
      yield put(actions.storeAllTheThings(transformedData));
    } else {
      yield put(actions.handleError());
    }
  }
}
```

TESTING THE SAGA

```
beforeEach(() => {
  generator = fetchAllTheThings();
  next = generator.next();
  expect(next.value).toEqual(call(request, '/fetchThings', 'GET'));
});

it('handles successful api calls', () => {
  const response = {
    status: 200,
    data: { we: 'win!' },
  };
  const transformedData = { showTrophy: true };

  next = generator.next(response);
  expect(next).toEqual(call(transformTheData, response.data));
  next = generator.next(transformedData);
  expect(next).toEqual(put(actions.storeAllTheThings(transformedData)));
  next = generator.next();
  expect(next).toEqual({ done: true, value: undefined });
});
```

```
function* fetchAllTheThings() {
  try {
    const response = yield call(request, '/fetchThings', 'GET');
    if (response.status < 400) {
      const transformedData = yield call(transformTheData, response.data);
      yield put(actions.storeAllTheThings(transformedData));
    } else {
      yield put(actions.handleError());
    }
  }
}
```

TESTING THE SAGA

```
beforeEach(() => {
  generator = fetchAllTheThings();
  next = generator.next();
  expect(next.value).toEqual(call(request, '/fetchThings', 'GET'));
});

it('handles successful api calls', () => {
  const response = {
    status: 200,
    data: { we: 'win!' },
  };
  const transformedData = { showTrophy: true };

  next = generator.next(response);
  expect(next).toEqual(call(transformTheData, response.data));
  next = generator.next(transformedData);
  expect(next).toEqual(put(actions.storeAllTheThings(transformedData)));
  next = generator.next();
  expect(next).toEqual({ done: true, value: undefined });
});
```

```
function* fetchAllTheThings() {
  try {
    const response = yield call(request, '/fetchThings', 'GET');
    if (response.status < 400) {
      const transformedData = yield call(transformTheData, response.data);
      yield put(actions.storeAllTheThings(transformedData));
    } else {
      yield put(actions.handleError());
    }
  }
}
```

VOILÁ

IN SUMMARY…

- Redux-Sagas…
  - offer a solution for asynchronous React / Redux code
  - make asynchronous code look synchronous
  - are easy to test
  - allow the rest of your application to be written with pure functions and components

THANK YOU!

LINKS

- Redux
  - https://redux.js.org/
- Redux-Sagas
  - https://redux-saga.js.org/
- Me (Scott Fletcher)
  - https://www.linkedin.com/in/sfletche/
- This Presentation
  - https://github.com/sfletche/presentations