Testing Redux Saga Applications

Daniel Stern
CODE WHISPERER

@danieljackstern github.com/danielstern

Testing

Methods for testing Redux Saga applications

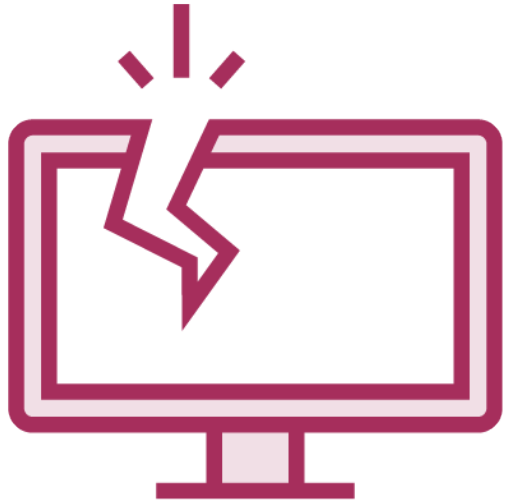
Creating testable sagas

Writing tests for our application

Testing Redux Saga Applications

Testing Redux Saga Applications

Tests need to avoid making real AJAX calls



Effects do not do anything unless *run* by Redux Saga

call()

call effect must be used instead of yielding directly to API methods

```
function* mySaga (sessionKey){
  let users = yield    api.fetchUsers(sessionKey);
}
```

- Saga is untestable, invoking the generator will actually call API

```
function* mySaga (sessionKey){
  let users = yield call([
    api,
    api.fetchUsers
  ],(sessionKey);
}
```

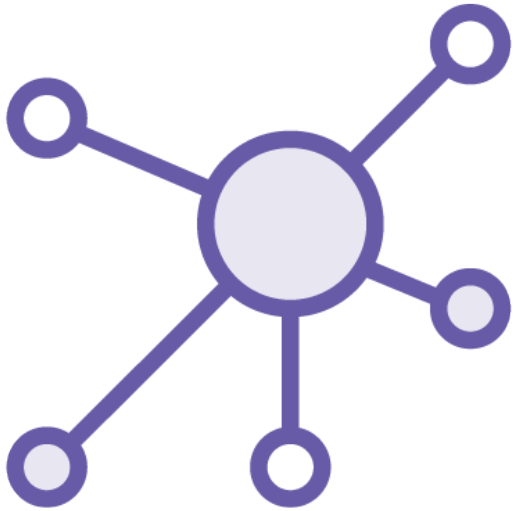
- Testable saga yields a *call* effect, no outside APIs are called except when run in Redux Saga
- Passing array to call allows context to be specified for Methods

Methods for Testing Redux Saga Applications

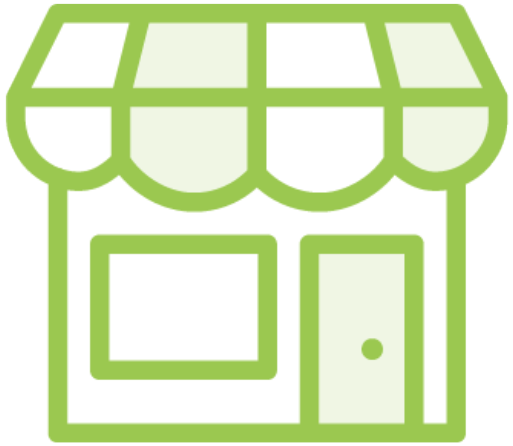
Testing Redux Saga Applications (Official Method)

Saga is executed as plain generator

Tests pass mock values to *next()*



Structure of effects is tested against expected values



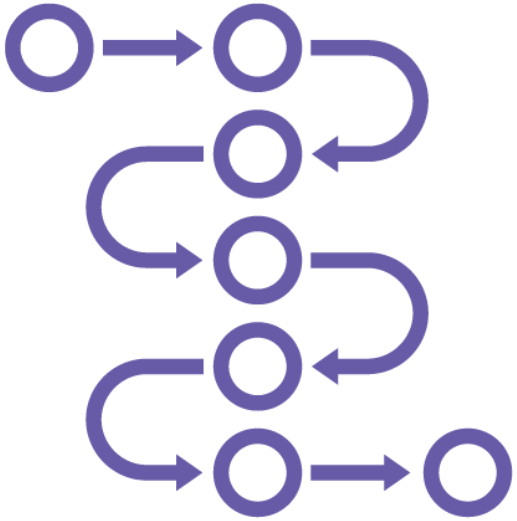
store is never used

“...the [official way of] testing sagas is wrong and puts emphasis on exact ‘implementation’, and not [on ‘results’].”

- Contributor on Redux-Saga GitHub page

Testing Redux Saga Applications (Alternate Method)

Mock store and application state are created



Entire saga is run from beginning to end

At completion, new state is compared to expected value

APIs must be injected as dependencies

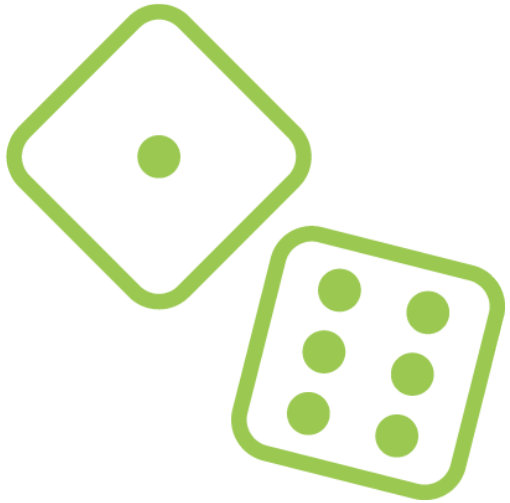
Testing Method Comparison

Standard (Unit Tests)	Alternate (End-to-End Tests)
Requires that *call* be used for functions	*Call* usage recommended but not required
Cannot test application state against expected values	Can test application state against expected values
Outside APIs can be imported with no special considerations	Any outside APIs must be injected as dependencies
Tests are brief and simple to set up	Tests are complex and require preparation of mock store and APIs
Test fails if yielded effects do not match expected values	Test fails if final application state does not match expected values

Implementing Unit Tests Within the Application

Implementing Unit Tests

Inject mock server response using *yield*



Compare generated *put* effect to expected values

Note that outside APIs are not being called (*call* usage)

Implementing Unit Tests for the Current User Saga

Demo

Create test for current user saga

next() is called manually for each step of generator execution

Yielded *call* and *put* effects will be tested against expected values

Implementing Unit Tests for the Item Quantity Saga

Demo

Create tests for item quantity saga

Inject successful server response and test output against expected values

Inject *unsuccessful* server response and test output against expected values

Module Summary

Summary

Redux Saga Effects prevent side-effects from actually occurring outside of Redux Saga

Sagas must be written to use *call* instead of directly invoking API methods

Dependencies do not need to be injected for unit tests

Effects can be easily tested against expected values

Support for E2E-driven methodology exists within community