ToiletPaper #111

Testing a saga made easy

Author: Robert Gruner / Software Engineer / Office Leipzig

✕ Problem

You are currently working on a single page application with React? You use Redux for state management? You even managed to encapsulate your asynchronous work steps and side effects (API calls, access to browser caches etc.) with redux-saga? In this case, it might look like that:

```
export default function* userConfirmationsSaga() {
    yield all([takeEvery(FETCH_USER_CONFIRMATION_START, fetchUserConfirmations), /*here may be more*/]);
}
function* fetchUserConfirmations() {
    try {
        const accessToken = yield select(getAccessTokenFromStore);
        const requestUrl = yield select(getUserConfirmationUrlFromStore);
        const response = yield call(fetch, requestUrl, {});
        yield put(fetchUserConfirmationSuccess(response));
    } catch (e) {
        yield put(fetchUserConfirmationMissing());
    }
}
```

Now, all you need is one further step to achieve mastery: how do you test the paths/cases to make sure that the thing does what it's supposed to do?

✓ Solution

Uncle Jeremy Fairbank offers something nice – and it's called a redux-saga-test-plan. Using this toy, you can test your saga integration. I.e. within a test run, it is possible to control the saga via mock Redux actions and special providers so that it takes a certain course (and hopefully triggers an expected Redux action). Using the above saga, a special success test might look like this:

➔ Example

```
describe('userConfirmationsSaga - when fetching user confirmation', () => {
     it('eventually dispatches success action', () => {
        const payload = { term_of_use_confirmation: new Date().toISOString() };
        return expectSaga(userConfirmationsSaga)
            .dispatch({ type: FETCH_USER_CONFIRMATION_START })
            .provide([
                [select(getAccessTokenFromStore), ''],
                [select(getUserConfirmationUrlFromStore), ''],
                [matchers.call.fn(fetch), payload]
            ])
            .put({ type: FETCH_USER_CONFIRMATION_SUCCESS, payload })
            .silentRun();
    });
});
```

- **.dispatch** controls which of the "observed" actions (FETCH_USER_CONFIRMATION_START) is being reacted to.
- Within **.provide**, everything that affects the saga is mocked – i.e. any selections from the Redux Application State or calls of other functions.
- **.put** specifies which Redux action is expected. The test only turns green if it is actually triggered.

+ Further Aspects

- https://twitter.com/elpapapollo
- It is also easily possible to define the Redux State for the start of a test sequence. This requires the .withState-method.