Unit Testing

Unit Testing

When testing React and Angular OWA, we will create a mock webservice to test requests to the database. This is useful so that we can test our code without having to be connected to an online webservice. We see a mock webservice being created in an example test cases below. For React OWAs, in addition to testing database request, tests are usually made for proper rendering and state changes.

Choose your test type

Depending on what you selected when building your web app, you will either use React or Angular to test your OWA. Below are examples of each test type that you can use for your web app.

Your test file(s) should be stored as `yourOWAfolder/test/exampleTest.js`.

React (With Redux)

React OpenMRS Open Web Apps primarily use the framework Jest for its unit testing. Setting up Jest can be found here.

All examples below are credited towards the Order Entry UI Module

Testing Actions

When testing an action, the objective is to dispatch a function into a mock store and compare the action it dispatched to what action you expected. Your function will likely make request to a database and this can be tested by mocking a request. The example below uses moxios to mock axios request. The response of the request will determine what action is dispatched.

Main code under test

```
const contextPath = window.location.href.split('/')[3];
export function fetchPatientRecord(patientUuid) {
  return dispatch => axiosInstance.get(`patient/${patientUuid}?v=custom:(patientId,uuid,patientIdentifier:
(uuid,identifier),person:(gender,age,birthdate,birthdateEstimated,personName,preferredAddress),attributes:
(value,attributeType:(name)))`)
    .then((response) => {
      dispatch({
        type: SET_PATIENT,
        patient: response.data,
      });
    })
    .catch((error) => {
      if (error.response) {
        dispatch({
          type: SET_PATIENT_FAILED,
        });
        window.location.href = `/${contextPath}`;
      }
    });
}

export function fetchPatientNote(patientUuid) {
  return dispatch => axiosInstance.get(`obs?
concept=162169AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA&patient=${patientUuid}&v=full`)
    .then((response) => {
      dispatch({
        type: SET_NOTE,
        note: response.data.results,
      });
    });
}
```

Action test

```
import {
        fetchPatientRecord,
        fetchPatientNote,
```

```
} from '../../app/js/actions/patient';
import {
        SET_PATIENT, SET_NOTE, SET_PATIENT_FAILED
} from '../../app/js/actions/actionTypes';

window.location = locationMock;

const uuid = '6cesf-4hijk-mkls';


// The describe function is for grouping related specs, typically each test file has one at the top level

describe('Patient actions', () => {
        beforeEach(() => moxios.install());
        afterEach(() => moxios.uninstall());

        // The it function represent a spec or a test case.

        it('fetch patient records', async (done) => {
                const { defaultPatient } = mockData;
                let request = moxios.requests.mostRecent();
                moxios.stubRequest(`${apiBaseUrl}/patient/${uuid}?v=custom:(patientId,uuid,patientIdentifier:
(uuid,identifier),person (gender,age,birthdate,birthdateEstimated,personName,preferredAddress),attributes:
(value,attributeType:(name)))`, {
                        status: 200,
                        response: defaultPatient
                });
                const expectedActions = [{
                type: SET_PATIENT,
                patient: defaultPatient
                }];
                const store = mockStore({});
                await store.dispatch(fetchPatientRecord(uuid))
                        .then(() => {
                                expect(store.getActions()).toEqual(expectedActions);
                        });
                done();
        });

        it('fetch patient records failure case', async (done) => {
                let request = moxios.requests.mostRecent();
                moxios.stubRequest(`${apiBaseUrl}/patient/${uuid}?v=custom:(patientId,uuid,patientIdentifier:
(uuid,identifier),person:(gender,age,birthdate,birthdateEstimated,personName,preferredAddress),attributes:
(value,attributeType:(name)))`, {
                        status: 400,
                        response: { message: "No record found" }
                });
                const expectedActions = [{
                        type: SET_PATIENT_FAILED,
                }];
                const store = mockStore({});
                await store.dispatch(fetchPatientRecord(uuid));
                        expect(store.getActions()).toEqual(expectedActions);
                done();
        });

        it('fetch patient\'s note', async (done) => {
                const { defaultNote } = mockData;
                moxios.stubRequest(`${apiBaseUrl}/obs?
concept=162169AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA&patient=${uuid}&v=full`, {
                        status: 200,
                        response: defaultNote
                });
                const expectedActions = [{
                        type: SET_NOTE,
                        note: defaultNote.results
                }];
                const store = mockStore({});
                await store.dispatch(fetchPatientNote(uuid))
                        .then(() => {
                                expect(store.getActions()).toEqual(expectedActions);
```

```
                });
                done();
        });
});
```

Testing Components

When testing a component, the objective is to check if the component gets mounted correctly and handles events correctly.

Main code under test

```
import React from 'react';
import PropTypes from 'prop-types';
import classNames from 'classnames';
import { ToolTip } from '@openmrs/react-components';
import '../../../css/grid.scss';

const formatPanelName = (panelName) => {
  const name = panelName;
  return name.replace(/panel/i, '').trim();
};

const formatToolTipData = setMembers => setMembers.map(test => test.display);

const LabPanelFieldSet = (props) => {
  const {
    selectedPanelIds, handleTestSelection, panels, labCategoryName,
  } = props;
  return (
    <fieldset className="fieldset">
      <legend>Panels</legend>
      <div className="panel-box">
        {panels.length ? (
          panels.map(panel => (
            <button
              id="panel-button"
              className={classNames('lab-tests-btn tooltip', {
                active: selectedPanelIds.includes(panel.uuid),
              })}
              type="button"
              key={`${panel.uuid}`}
              onClick={() => handleTestSelection(panel, 'panel')}>
              {formatPanelName(panel.display.toLowerCase())}
              <ToolTip
                toolTipHeader="Tests included in this panel:"
                toolTipBody={formatToolTipData(panel.setMembers)}
              />
            </button>
          ))
        ) : (
          <p>{labCategoryName} has no panels</p>
        )}
      </div>
    </fieldset>
  );
};

LabPanelFieldSet.defaultProps = {
  selectedPanelIds: [],
};

LabPanelFieldSet.propTypes = {
  handleTestSelection: PropTypes.func.isRequired,
  labCategoryName: PropTypes.string.isRequired,
  panels: PropTypes.array.isRequired,
  selectedPanelIds: PropTypes.array,
};

export default LabPanelFieldSet;
```

Component test

```
import React from 'react';
import LabPanelFieldSet from '../../../app/js/components/labOrderEntry/LabPanelFieldSet';

let props;
let mountedComponent;

const getComponent = () => {
  if (!mountedComponent) {
    mountedComponent = shallow(<LabPanelFieldSet { ...props } />);
  }
  return mountedComponent;
};

describe('Component: LabPanelFieldSet', () => {
  beforeEach(() => {
    mountedComponent = undefined;
    props = {
      handleTestSelection: jest.fn(),
      selectedPanelIds: [],
      panels: [{
        uuid: 'asampleduuid1234',
        display: 'sample',
        setMembers: [{
          uuid: 'asampleduuid1234',
          display: 'sample'
        }]
      }]
    };
  });

  it('should mount initially', () => {
    const component = getComponent();
    expect(component).toMatchSnapshot();
  });

  it('should support click for each button rendered', () => {
    const component = getComponent();
    const panelButton = component.find('#panel-button').at(0);
    panelButton.simulate('click', {});
    expect(props.handleTestSelection).toBeCalled();
  });

  it('should add class `active` to the selected panels', () => {
    props.selectedPanelIds = ['asampleduuid1234'];
    const component = getComponent();
    const activePanelButton = component.find('.active.lab-tests-btn');
    expect(activePanelButton.length).toEqual(1);
  });
```

Testing Reducers

When testing a component, the objective is to check if the reducers change the application state as intended given a specific action. Your function will most likely compare two states. This example below creates an action and initial state then passes them to the reducer.

Main code under test

```
const initialState = {
  labOrderData: {},
  errorMessage: '',
  status: {
    error: false,
    added: false,
  },
};

const createOrderReducer = (state = initialState, action) => {
  switch (action.type) {
    case 'SAVE_DRAFT_LAB_ORDER_SUCCESS': {
      const labOrderData = action.data;
      return {
        ...state,
        labOrderData,
        status: {
          added: true,
          error: false,
        },
      };
    }
    case 'SAVE_DRAFT_LAB_ORDER_FAILURE': {
      return {
        ...state,
        errorMessage: action.payload,
        status: {
          error: true,
          added: false,
        },
      };
    }
    case 'SAVE_DRAFT_LAB_ORDER_LOADING': {
      return {
        ...state,
        status: {
          ...state.status,
          loading: true,
        },
      };
    }
    default: {
      return state;
    }
  }
};

export default createOrderReducer;
```

Reducer test

```
import createOrderReducer from '../../app/js/reducers/createOrderReducer';

describe('createOrder reducer test suite', () => {
  const initialState = {
    labOrderData: {},
    errorMessage: '',
    status: {
      error: false,
      added: true,
    },
  };

  it(`sets status key error to true and sets key errorMessage to
  error message from payload on action type SAVE_DRAFT_LAB_ORDER_FAILURE `, () => {
    const action = {
      type: 'SAVE_DRAFT_LAB_ORDER_FAILURE',
      payload: 'Invalid data',
    };
    expect(createOrderReducer(initialState, action)).toEqual({
      ...initialState,
      errorMessage: 'Invalid data',
      status: {
        added: false,
        error: true,
      },
    });
  });

  it(`parses data from payload to key labOrderData and sets staus key
  added to true on action type SAVE_DRAFT_LAB_ORDER_SUCCESS `, () => {
    const data = {
      id: 1,
      name: 'Lab order',
      description: 'amoxycillinn tests',
    };
    const action = {
      type: 'SAVE_DRAFT_LAB_ORDER_SUCCESS',
      data,
    };
    expect(createOrderReducer(initialState, action)).toEqual({
      ...initialState,
      labOrderData: action.data,
      status: {
        error: false,
        added: true,
      },
    });
  });

  it('returns initial state if action type is not handled', () => {
    const action = {
      type: 'SAVE_DRAFT_LAB_ORDER',
    };
    expect(createOrderReducer(initialState, action)).toEqual(initialState);
  });
  it('should handle SAVE_DRAFT_LAB_ORDER_LOADING', () => {
    const action = {
      type: 'SAVE_DRAFT_LAB_ORDER_LOADING',
    };
    const mockState = createOrderReducer(initialState, action)
    expect(mockState.status.loading).toEqual(true);
  });
});
```

Setup

OpenMRS React OWA typically have a setup file for test to handle import that will be used across all tests. This is to help abide by DRY and avoid repetitive code. An example of this is shown below:

Order Entry UI's test setup file

```
import expect from 'expect';
import thunk from 'redux-thunk';
import configureMockStore from 'redux-mock-store';
import { configure, shallow, render, mount } from 'enzyme';
import Adapter from 'enzyme-adapter-react-16';
import sinon from 'sinon';
import moxios from 'moxios';
import axios from 'axios'
import promiseMiddleware from 'redux-promise-middleware'
import axiosInstance from '../app/js/config'

import mockData from '../__mocks__/mockData';
import locationMock from '../__mocks__/locationMock';
import store from '../app/js/redux-store';

process.env.NODE_ENV = 'test';

// React 16 Enzyme adapter
configure({ adapter: new Adapter() });

const promiseTypeSuffixes = ['LOADING', 'SUCCESS', 'FAILURE'];

const middlewares = [thunk, promiseMiddleware({promiseTypeSuffixes})];
const mockStore = configureMockStore(middlewares);
const reader = new FileReader();
moxios.install(axiosInstance)
const contextPath = window.location.href.split('/')[3];
const apiBaseUrl = `/${contextPath}/ws/rest/v1`;

require.extensions['.css'] = () => null;
require.extensions['.png'] = () => null;
require.extensions['.jpg'] = () => null;

global.expect = expect;
global.mockData = mockData;
global.locationMock = locationMock;
global.store = store;
global.apiBaseUrl = apiBaseUrl;
global.moxios = moxios;
global.mount = mount;
global.sinon = sinon;
global.shallow = shallow;
global.mockStore = mockStore;
global.navigator = {
    userAgent: 'node.js'
  };
global.document = document;


var documentRef = document;
```

In order to properly run tests and apply the setup file, Jest needs to be configured. An example configuration file is shown below:

Order Entry UI's jest.config File

```
module.exports = {
  verbose: true,
  coveragePathIgnorePatterns: [
    '/node_modules/',
    '/templates/',
    '/tests/mocks',
    '/tests/setup.js',
    'app/js/openmrs-owa-orderentry.jsx'
  ],
  testURL: 'http://localhost',
  collectCoverage: true,
  collectCoverageFrom: [
    'app/js/**/*.{js,jsx}'
  ],
  testEnvironment: "jsdom",
  roots: ['<rootDir>'],
  setupFiles: [
    '<rootDir>/tests/setup.js',
  ],
  snapshotSerializers: [
    "enzyme-to-json/serializer"
  ],
  moduleFileExtensions: [
    'js',
    'jsx'
  ],
  moduleNameMapper: {
    '\\.(jpg|jpeg|png|gif|eot|otf|webp|svg|ttf|woff|woff2|mp4|webm|wav|mp3|m4a|aac|oga)$':
        '<rootDir>/tests/mocks/fileMock.js',
    '\\.(css|scss)$': 'identity-obj-proxy'
  }
};
```

Useful Links

ReactJS Testing

Jest

Jest Documentation

Order Entry UI's Github Repo

Angular

Angular OpenMRS Open Web Apps also primarily use the framework Jasmine and the test runner Karma for its unit testing. To set up our mock backend, we use $httpbackend service. $httpbackend has the method whenGet that will create a new GET request. There needs to be enough whenGet calls that will receive all of the information that your controller expects. $rootScope needs to be created before each spec to observe model changes. $componentController creates an instance of a controller. Make sure you have the name of controller that you are testing in the parameters.

Example 1: Querying for an encounter

```
"use strict";

import '../app/js/home/home';


// The describe is a function that represents a test suite. This should encompasses tests that verify any
behaviors of a component.
describe('component: encounterComponent', () => {
  // The beforeEach is a function that is called before each spec ("it" function).
  beforeEach(angular.mock.module('home'));

  let component, scope, $componentController, $httpBackend;
  // Defining mock data as encounterList
```

```
  let encounterList = {
    "results":[{
      "uuid": "6ff4d871-a529-4b88-9cb9-761cf3c35136",
      "display": "Vitals 31/12/2015",
      "links": [
        {
          "rel": "self",
          "uri": "http://localhost:8081/openmrs-standalone/ws/rest/v1/encounter/6ff4d871-a529-4b88-9cb9-
761cf3c35136"
        }
      ]
    }]
  };

 let emptyResult = {
        "results": []
 }

  beforeEach(inject((_$httpBackend_, $rootScope, _$componentController_) => {
    $httpBackend = _$httpBackend_;
    $httpBackend.whenGET(/translation.*/).respond();
    $httpBackend.whenGET('manifest.webapp').respond(500, "");
    $httpBackend.whenGET('/ws/rest/v1/encounter?includeAll=true&q=john&v=full').respond(encounterList);
    $httpBackend.whenGET('/ws/rest/v1/encounter?q=john').respond(encounterList);
        $httpBackend.whenGET('/ws/rest/v1/encounter?includeAll=true&q=Matt&v=full').respond(emptyResult);

    scope = $rootScope.$new();
    $componentController = _$componentController_;

  }));
  // The it function represent a spec.
  it('Rest call should return the required json file', () => {

    component = $componentController('encounterComponent',
      {
        $scope: scope
      }
    );

    component.query = 'john';
    component.onsearch();
    $httpBackend.flush();
    expect(component.encounters).toEqual(encounterList.results);

  })

  // Test case 2.
  it('Rest call should return the required json file', () => {

    component = $componentController('encounterComponent',
      {
        $scope: scope
      }
    );

    component.query = 'Matt';
    component.onsearch();
    $httpBackend.flush();
    expect(component.encounters).toEqual(emptyResult.results);

  })
});
```

Example 2: Querying for a Paitient

```
"use strict";

import '../app/js/home/home';

describe('component: patientSearchComponent', () => {
  beforeEach(angular.mock.module('home'));

  let component, scope, $componentController, $httpBackend;

  const patientList = {
  "results":[{
    "uuid": "c61195ff-0e93-4799-bd49-3c1738d9c34f",
    "display": "1002C4 - John Sánchez",
    "links": [
      {
        "rel": "self",
        "uri": "http://localhost:8081/openmrs-standalone/ws/rest/v1/patient/c61195ff-0e93-4799-bd49-
3c1738d9c34f"
      }
    ]
  }]
  };

  beforeEach(inject((_$httpBackend_, $rootScope, _$componentController_) => {
    $httpBackend = _$httpBackend_;
    $httpBackend.whenGET(/translation.*/).respond();
    $httpBackend.whenGET('manifest.webapp').respond(500, "");
    $httpBackend.whenGET('/ws/rest/v1/patient?includeAll=true&q=john&v=full').respond(patientList);
    $httpBackend.whenGET('/ws/rest/v1/patient?q=john').respond(patientList);

    scope = $rootScope.$new();
    $componentController = _$componentController_;

  }));

  it('Rest call should return the required json file', () => {

    component = $componentController('patientSearchComponent',
      {
        $scope: scope
      }
      );

    component.query = 'john';
    component.onsearch();
    $httpBackend.flush();
    expect(component.patients).toEqual(patientList.results);

  })
});
```

Useful Links

Angular: Unit Testing Jasmine, Karma (step by step)

Jasmine

Karma

Angular Testing Guide

$httpBackend

$rootScope

$componentController

How to Run Unit Tests

In the root of your project directory, run the command:

Run Tests Only

```
npm run test
```

Build Project and Run Tests

```
npm run build:prod
```

Troubleshooting

Angular Owa Fails to Run Tests and Build after Generating OWA

This is a potential fix for this error message:

```
component: encounterComponent Rest call should return the required json file FAILED
Error: [$injector:modulerr] Failed to instantiate module home due to:
Error: [$injector:modulerr] Failed to instantiate module openmrs-contrib-uicommons due to:
Error: [$injector:modulerr] Failed to instantiate module openmrs-contrib-uicommons.concept-autoComplete due to:
Error: [$injector:modulerr] Failed to instantiate module ngSanitize due to:
TypeError: a is not a function
```

1) Open package.json which is in your OWA's root folder, and under dependencies, remove the "^" before the version number of each following Angular dependency: angular, angular-route, and angular-mocks.

2) In the root of your project directory, run the command:

Install Correct Angular Dependencies

```
npm install
```

3) In home.js of your project remove this code:

```
.config(['$qProvider', function ($qProvider) {
    $qProvider.errorOnUnhandledRejections(false);
}])
```

4) Re-run test and/or build