Pager Project

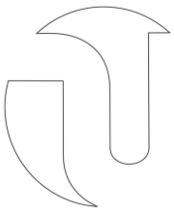
Unit Tests Specification

Abstract

This document provides an explanation of the unit tests of the web application, including the objectives of the tests and how to execute a test session to achieve the test objectives. This document also covers the test environment, coverage, instructions and evaluation criteria.

Version History

Date	Version	Author	Description
27/05/2020	4.0	Alexander Jonsson	Formatting before this becomes a PDF file.
26/05/2020	3.0	Alexander Jonsson	Formatting and more text.
25/05/2020	2.0	Alexander Jonsson	Added all tests.
08/04/2020	1.0	Alexander Jonsson	Start.



IVAR JACOBSON CONSULTING

An Essential Unified Process Document

Table of Contents

This page has been left blank intentionally.

1 Introduction

1.1 Document Purpose

The purpose of this document provide a explanation of the unit tests, including the objectives of the test and how to execute a test session to achieve the test objectives.

1.2 Document Scope

The scope of this document is limited to consideration of:

- Objective of unit tests
- Expected behaviour of the unit tests
- Test environment, coverage and instructions
- Criteria of evaluation

This scope of this document does not include:

- Documenting and analyzing the test results resulting from test executions that utilize this test specification
- The overall verification and validation strategy for the solution, including any other tests that need to be specified and executed to ensure that the solution is fit for purpose

1.3 Document Overview

This document contains the following sections:

- **Introduction** - to the testing in general, environment, coverage and etc.
- **Test Environment** - hardware and software environment in which to perform the test, including preloaded system state data and reference data
- **Test Coverage** – defines scope coverage in terms of the Test Cases or Unit Tests that are run as part of the test
- **Test Instructions** – the procedure that must be followed to execute the test, including sequencing and parallelism of test case execution and the test execution control points and observation points.
- **Evaluation Criteria** - details of how the test results are evaluated.
- **Test References** – identifies the tests that run in the test session.
- **References** – provides full reference details for all documents, white papers and books referenced by this document.

2 Environment, coverage, instructions

Where	GitHub.
When	When a new Pull Request is made.
Why (Test Purpose)	To ensure that essential parts of the web application is working as intended.

2.1 Test Environment

The unit tests runs in a Node environment. To manually initiate the tests, Node.js must be installed on the host and be used through a CLI. [1]

2.2 Test Coverage

The tests cover all Redux actions in combination with Firebase functions to ensure that the Redux states are behaving as they are intended to do. If Redux states does not behave properly, one cannot be assured that the web application works as intended. [2]

2.3 Test Instructions

If you as a user would like to manually initiate the tests, please do the following [1]:

1. Navigate to the root folder of the project through a command line interface.
2. Install Node Packet Manager on your system.
3. Navigate to the root folder of the project.
4. Install all necessary dependencies with `` `npm install` ``

   4.1. Do `` `npm audit fix` `` if necessary.
5. Run the tests with `` `npm test` ``
6. Let the tests run...
7. See results

During the run (and after) you should be prompted with which of the tests passed, the runtime for each test and total runtime, if one or more of tests failed you will be prompted with information to help you understand why the test failed.

2.4 Evaluation Criteria

When evaluating the tests, it is important to make sure that no test returns as failed, having one or more test fails means that important parts of the system is not working as intended. This could lead to unforeseen errors and bugs which could harm the intended user's experience with the web application. Some parts might not necessarily be code related but database related, if this is the case it is important to make sure that the database and the connection to it is working as intended.

3 Test References

This section references the tests that run during the test session and what they are intended to do.

3.1 userActions.js

3.1.1 cannot authenticate the user with wrong credentials

```
it('cannot authenticate the user with wrong credentials', () => {
      const expectedActions = [{ type:"ERROR_LOGIN" }]
      const store = mockStore({ type: [] })
      return store.dispatch(userAction
       .submitUser("wrong@wrong.no", "notcorrectpsw"))
       .then(() => {
        expect(store.getActions()).toEqual(expectedActions);
      })
    })
```

This test makes sure that to authenticate you must use valid credentials. The login process is expected to fail.

3.1.2 can authenticate the user with correct credentials

```
it('can authenticate the user with correct credentials', () => {
      const expectedActions = [{ type:"LOGIN", pay-
load: "test@test.test" }]
      const store = mockStore({ type: [] })
      return store.dispatch(userAction
       .submitUser("test@test.test", "testingtesting"))
       .then(() => {
        expect(store.getActions()).toEqual(expectedActions)
      })
    })
```

Like the previous test, this test makes sure that to authenticate you must use valid credentials. The login process is expected to succeed.

3.1.3 can return the current user

```
it('can return the current user', () => {
      const expectedActions = []
      const expectedValue = "test@test.test"
      const store = mockStore({ type: [] })
      let data = store.dispatch(userAction
       .checkUser("test@test.test", "testingtesting"))

      expect(store.getActions()).toEqual(expectedActions)
      expect(data).toEqual(expectedValue)
    })
```

This test makes sure that Firebase is able to correctly identify the currently logged in user. Expected to return the used test account.

3.2 messageAction.js

3.2.1 can send message to database

```
it('can send message to database', () => {
    const expectedActions = [{ type:"SEND" }]
    const store = mockStore({ type: [] })
    return store.dispatch(messageAction
       .sendMsg("test", "hello test!")).then(data => {
      expect(store.getActions()).toEqual(expectedActions)
    })
  })
```

This test makes sure that as a logged in user, you are able to send a message to the Firebase database. Expected to send a message.

3.2.2 cannot send message with length less than 2

```
it('cannot send message with length less than 2', () => {
    const expectedActions = [{ type:"ERROR_SEND" }]
    const store = mockStore({ type: [] })
    store.dispatch(messageAction.sendMsg("test", "a"))
    expect(store.getActions()).toEqual(expectedActions)
  })
```

This test makes sure that messages with a length of less than two is not being sent. Expected to not send the message at all.

3.2.3 can fetch correctly from database

```
it('can fetch correctly from database', () => {
    const expectedActions = [{ type:"FETCH" }]
    const store = mockStore({ type: [] })
    return store.dispatch(messageAction
       .fetchMsg("test")).then(data => {
      expect(store.getActions()).toEqual(expectedActions)
      expect(data[0].text).toEqual("hello test!")
    })
  })
```

This test makes sure that the available test message is being fetched correctly from the database. Expected to read and check if the fetched message equals the previously sent one (see “2.2.1 can send message to database”.)

3.3 settingsAction.js

3.3.1 can fetch general settings

```
it('can fetch general settings', () => {
    const expectedActions = [{type: "FETCH_GENERAL"}];
    const store = mockStore({ type: [] });
    return store.dispatch(settingsAction.fetchGeneral()).then(data => {
      expect(store.getActions()).toEqual(expectedActions);
      expect(data.general.amount_msg).toEqual(3);
      expect(data.general.colour).toEqual("inherit");
    });
  });
```

Makes sure that fetching the general settings is possible and that they are correct as well. Expected to get the default general settings.

3.3.2 can update general settings

```
it('can update general settings', () => {
    const expectedActions = [{type: "UPDATE_GENERAL"}];
    const store = mockStore({type: []});

    return store.dispatch(settingsAction
       .updateGeneral("red", 5)).then(data => {
      expect(store.getActions()).toEqual(expectedActions);
      return store.dispatch(settingsAction
       .fetchGeneral()).then(data => {
        expect(data.general.amount_msg).toEqual(5);
        expect(data.general.colour).toEqual("red");
      })
      .then(() => {
        //this resets the values for next test run
        store.dispatch(settingsAction.updateGeneral("inherit", 3))
      })
    })
  })
```

Makes sure that the general settings can be updated and changed. Expected to be able to change the general settings and fetch the newly updated settings.

3.4 userAction.js & messageAction.js

3.4.1 can log out user

```
it('can log out user', () => {
        const expectedActions = [
           {type: "SIDEBAR_LOGOUT"},
           {type: "LOGOUT"}
       ]
        const store = mockStore({ type: [] })
        store.dispatch(userAction.logoutUser())
        expect(store.getActions()).toEqual(expectedActions)
      })
```

Checks if the user can logout. Expected to log out the user from the current session.

3.4.2 does not send any messages when signed out

```
it('does not send any messages when signed out', () => {
          const expectedActions = [{type:"ERROR_SEND"}]
          const store = mockStore({ type: [] })
          store.dispatch(messa-
geAction.sendMsg("test", "does not work!"))
          expect(store.getActions()).toEqual(expectedActions)
      })
```

Makes sure that no messages can be sent while logged out (i.e not logged in to a valid account.) Expected to fail at sending the message.

3.4.3 does not fetch when signed out

```
it('does not fetch when signed out', () => {
          const expectedActions = [{type:"ERROR_FETCH"}]
          const store = mockStore({ type: [] })
          store.dispatch(messageAction.fetchMsg())
          expect(store.getActions()).toEqual(expectedActions)
      })
```

Makes sure that no messages can be fetched while not logged in to a valid account (i.e logged out.) Expected to not be able to fetch messages from the database.

3.5 Account related tests

3.5.1 cannot change password with WRONG credentials

```
it('cannot change password with WRONG credentials', () => {
      const expectedActions = [{type: "ERROR_AUTH"}];
      const store = mockStore({type: []});
      return firebase.auth().createUserWithEmailAndPassword("remo-
ver@test.com", "testingtesting")
        .then(() => { return firebase.auth().signInWithEmailAndPass-
word("remover@test.com", "testingtesting")
            .then(() => { return store.dispatch(settingsAction.change-
Password("remover@test.com", "WRONG_PSW","shouldNotHappen"))
              .then(() => { return expect(store.getAct-
ions()).toEqual(expectedActions) })
            })
        })
    })
```

Checks if the password gets changed if you enter the wrong credentials for changing the account password. If the password changes the test is expected to fail.

3.5.2 can change password with CORRECT credentials

```
it('can change password with CORRECT credentials', () => {
      const expectedActions = [{type: "PSW_CHANGED"}];
      const store = mockStore({type: []});
      return firebase.auth().signInWithEmailAndPassword("remo-
ver@test.com", "testingtesting")
        .then(() => { return store.dispatch(settingsAction.changePass-
word("remover@test.com", "testingtesting","newPassword"))
          .then(() => { return expect(store.getActions()).toEqual(ex-
pectedActions) })
      })
    })
```

Checks if the password gets changed if you enter the correct credentials for changing the account password. The password of the account is expected to be changed.

3.5.3 cannot remove account with WRONG credentials

```
it('cannot remove account with WRONG credentials', () => {
      const expectedActions = [{type: "ERROR_AUTH"}];
      const store = mockStore({type: []});
      return firebase.auth().signInWithEmailAndPassword("remo-
ver@test.com", "newPassword")
        .then(() => { return store.dispatch(settingsAction.remove-
Account("remover@test.com", "WRONG_PSW"))
          .then(() => { return expect(store.getActions()).toEqual(ex-
pectedActions) })
      })
    })
```

Makes sure that you are not able to delete an account when entering invalid credentials. Expected to not delete the account from the database.

3.5.4 can remove account with CORRECT credentials

```
it('can remove account with CORRECT credentials', () => {
      const expectedActions = [{type: "USER_DELETED"}];
      const store = mockStore({type: []});
       return firebase.auth().signInWithEmailAndPassword("remo-
ver@test.com", "newPassword")
          .then(() => { return store.dispatch(settingsAction.remove-
Account("remover@test.com", "newPassword"))
            .then(() => { return expect(store.getActions()).toEqual(ex-
pectedActions) })
        })
    })
```

Makes sure that you are able to delete an account when entering valid credentials. The account is expected to be deleted.

3.6 sidebarAction.js

3.6.1 can toggle the sidebar

```
it('can toggle the sidebar', () => {
    const expectedActions = [{ type:"SIDEBAR" }]
    const store = mockStore({ type: [] })
    store.dispatch(sidebarAction.toggle())
    expect(store.getActions()).toEqual(expectedActions)
  })
```

Makes sure that the user is able to toggle the sidebar. The expected action checks that the “action type” is of the sidebar action and nothing else.

Appendix A - References

[1] Node Packet Manager: https://www.npmjs.com/

[2] GitHub Project: https://github.com/adamliliemark/ii1302-webapp