**HSBC Fund Rebalancing Project**

Test Plan

4/1/2019

Version 1.3

**Team REST**

Aurora (Yutian) Qiu

Cathy Leung

Leo Cheng

Leo (Jixing) Li

Shaw (Xiao) Lu

Stanley Ye

Tanya (Yi) Tan

**TABLE OF CONTENTS**

[**REVISION HISTORY**](#_u6rkx3gi9uj9) **3**

[**STAKEHOLDERS**](#_vuezx64jm3t) **3**

[**SUMMARY**](#_hjsm1ff1natx) **3**

[Testing Outline](#_y77fvurhzhcg) 3

[Testing Tool](#_5zbb7gz2tbac) 4

[Risk-Based Testing Approach](#_7o8zot81emuk) 4

[Testing Cycle](#_vgmt0thg926s) 5

[Acceptance Criteria](#_fg8u0rkyoevn) 6

[Resourcing and Separation of Duties](#_lbn2wyl9fge) 6

[**FUNCTIONAL TEST PLAN**](#_x5u0l8hx0kbh) **6**

[Header Validation](#_qv9p4hnplqko) 6

[Allocations Validation](#_mvzz786ogzjc) 7

[Deviation Validation](#_5czy54dbwfzb) 7

[Prepare Rebalance Recommendation](#_b767ykwp057r) 7

[Database Integration](#_yh824ehnxt26) 7

[Mock Fund System Integration](#_y5qok8b5l4zs) 7

[Create Initial Preference for a Portfolio](#_5bb3mb5h21ii) 7

[Get Initial Preference of a Portfolio](#_uec0kcfdb69m) 8

[Update Deviation of a Portfolio](#_n0c2qgjsjb18) 8

[Create Rebalance Recommendation Transactions for a Portfolio](#_rbw26pw3wk0j) 8

[Update Allocations of a Portfolio](#_dm1i9hrrdti2) 8

[Execute Recommendation](#_djjf3oaacb02) 8

[**NON-FUNCTIONAL TEST PLAN**](#_c5rpsdy8g2ak) **9**

[Performance Testing](#_m3adgcb1ffb) 9

[Capacity Testing](#_bc5d9zjwtsep) 9

[Cross-Browser Testing](#_2u5zn9llvbdb) 9

[Code Quality Testing](#_ij29guyw8rfv) 10

[**TEST SCRIPTS**](#_rg52xodoec64) **10**

[**UAT TEST SCRIPT**](#_vjb35lyftslo) **19**

[**SECURITY TESTING AND TEST DATA APPROACH**](#_apea1oqdty6a) **20**

[**REGRESSION TESTING**](#_dodzr8fedh41) **21**

# 

# **REVISION HISTORY**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **DATE** | **Name** | **Description of Change** |
| 1.3 | 4/1/2019 | Team REST | Final version |
| 1.2 | 3/15/2019 | Team REST | Added more details in the UAT and Non-Functional Test Plan |
| 1.1 | 3/14/2019 | Team REST | Added inputs and outputs for endpoint tests |
| 1.0 | 3/13/2019 | Team REST | Initial Draft |

# **STAKEHOLDERS**

Categories:

1. Agree with contents
2. Agree, subject to incorporation of comments
3. Disagree, comments included

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Approver Name/Title** | **Signature** | **Sign Date** | **Subject to Category** | **Comments** |
| Wilson Fung |  |  |  |  |
| Clifford Lee |  |  |  |  |
| Jerry Jim |  |  |  |  |

# **SUMMARY**

## **Testing Outline**

To fully test our system, we utilize a combination of unit tests, system Integration test (SAT) and user acceptance test (UAT) to test the functional requirements of the system. We also utilize a mixture of static testing and manual testing for the non-functional requirements. Testing is done through white box unit tests and black-box integration tests. We have acknowledged the risk of conducting white box unit tests is that we may ignore certain edge cases during development and testing. For integration testing, we also face the risk that the test outcomes may be difficult to reproduce. Tests can become redundant and even in the best situations, testing may rarely cover all software paths. We have automated SAT on the backend and will be using a manual testing approach to test the integration between UI and the backend. For UAT, we will provide a complete list of acceptance criteria to test the business flow. In addition, static testing will be performed to examine code coverage.

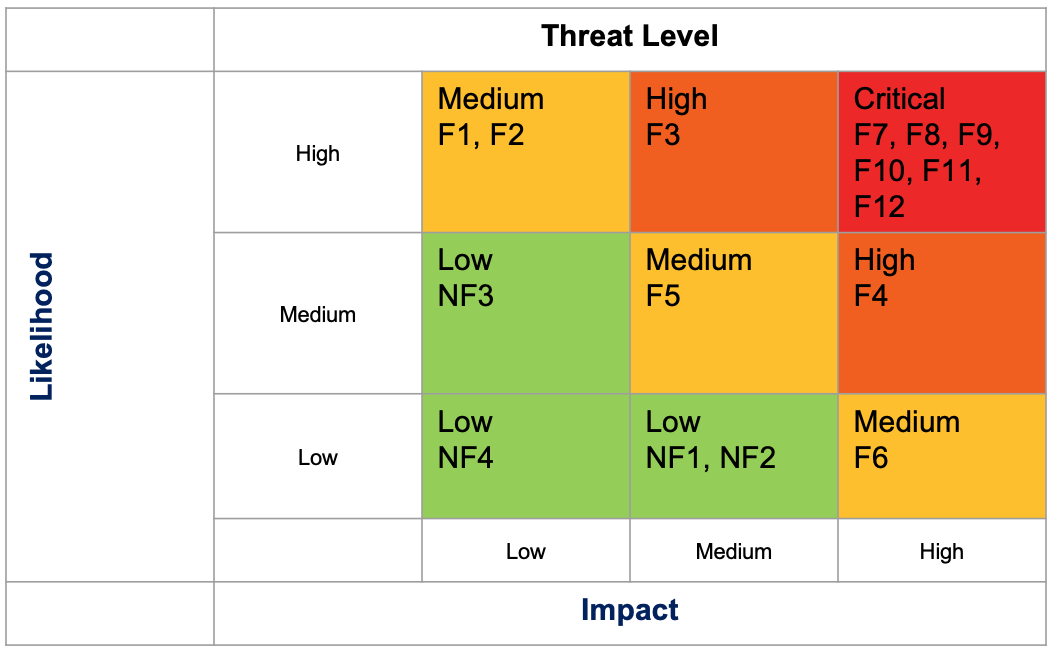
## **Testing Tool**

1. **Junit 5**: For unit tests, integration tests and SAT, Junit 5 is utilized as the test runner and assertion library.
2. **Shazamcrest 0.11**: As a supplement, Shazamcrest 0.11 is used as customized assertion matchers.
3. **IntelliJ IDEA Ultimate 2018.3.5**: For static analysis such as code coverage, we use IntelliJ IDEA.
4. **Trello**: For bug tracking and task assignment, we utilized Trello to create subtasks and assign to designated personnel.
5. **LoadRunner**: For some parts of non-functional testing, we rely on LoadRunner.

## **Risk-Based Testing Approach**

Using a risk-based approach for our testing, we first assess our project in terms of the following risk categories: product quality risk, technical risk as well as business risk. For product quality risks, we may face the risk of not being able to fulfill all our functional and non-functional requirements. The likelihood of such risk is high and the impact is high. As some of our endpoints are dependent on others, we will prioritize these endpoints in testing in order to ensure that all other endpoints that depend on them can be delivered. We will also maintain strong and detailed documentation on all API endpoints to mitigate such risk. It’s crucial to ensure the quality of every requirement in order to deliver a successful project. For technical risk, we face the risk of having an unsuccessful deployment on Google Cloud. The likelihood of such risk is medium-high and the impact is very high. We will utilize the WIP link to conduct frequent deployment and apply continuous integration to mitigate such risk. For business risk, we face the risk of not integrating all required features into a user-friendly UI that provide great customer experience for HSBC clients. The likelihood of such risk is medium, however, the impact is very high. To mitigate such risk, we will define clear acceptance criteria for all UI test scenarios and conduct manual testings in the early stage of our test cycles.

In order to align the testing approach with the identified risks above, we have applied the risk assessment matrix to evaluate all test components and prioritize those with higher risks into the early stage of our test cycles. The matrix is shown below:



## **Testing Cycle**

As our system works with multiple external systems including database and HSBC fund system, we have a relatively high chance of system corruption during development. Therefore, we start with performing Unit Testing to ensure the functionalities exempting any external connection. The next step is to perform Integration Testing to explicitly test the performance of our system interacting with the database and with the HSBC fund system. After those are been settled, black box SAT comes in to test each endpoint at a higher level. Taking such order as well as the risk assessment matrix into consideration, we have defined the following three distinct testing cycles:

**Cycle 1**: Automated testing upon check-in to master:

* Upon every checking to master automated test suite on all functional requirements will be run (F1 to F12). Greater attention will be given to components that have higher risks identified in the priority matrix.
* The purpose of the first cycle is to identify and solve the high defects of the system via automated tests that have been created to identify critical bugs.

**Cycle 2**: Manual testing on UI, feature modification and UAT

* All UI features will be tested via manual testing based on the defined acceptance criteria. UAT will be conducted as well.
* Upon changes to features in this stage, manual testing will be conducted to make sure the new change comply with the original development.
* The purpose of the second cycle is to identify and solve remaining high and medium defects, remove gaps in the scripts and obtain performance results.

**Cycle 3**: Non-functional testing and regression testing

* Before the release, all three non-functional testings: performance testing, web browsing testing and code quality testing will be run. UAT will be conducted by TAs or the stakeholders. Regression testing will be run to ensure the successful delivery of the project.

## **Acceptance Criteria**

We have defined the acceptance criteria for each testing cycle as the following:

* **Cycle 1**: Acceptance criteria for automated tests is 100% of all automated tests passing.
* **Cycle 2**: Acceptance criteria for manual testing on the UI is that all manual tests should meet the expected output. If bugs are found and the risk is thought to be minimal, as per the team’s discretion, then the release can move forward.
* **Cycle 3**: Acceptance criteria for non-functional testing should be acceptable for most use-cases. If any issues are found they can be noted and addressed in further releases if necessary unless they are bottlenecks to performing any of the required functionality in more than 50% of use cases. At this point, all regression testing should be passed to ensure the quality of the project.

## **Resourcing and Separation of Duties**

Our team has been separated into Testing Team 1 and Testing Team 2:

**Testing Team 1**: Leo C, Leo Li, Cathy, Stanley and Tanya.

* They are in charge of Cycle 1 and non-functional testings in Cycle 3 in the testing phase.
* Cross-examination is implemented where developers cross-test each other’s features.

**Testing Team 2**: Shaw and Aurora.

* They are in charge of Cycle 2 in the testing phase.

In addition, stakeholders and TAs commence UAT as part of Cycle 2 testing.

# **FUNCTIONAL TEST PLAN**

## **Header Validation**

* 1. Purpose: to check whether our functions correctly validate the header contains “**x-custid**” with none empty value.
  2. Testing technique: white box automated unit testing.
  3. Anticipated scripts: 5 tests are performed, 2 of which are happy paths and 3 are non-happy paths.

## **Allocations Validation**

* 1. Purpose: to verify if our functions correctly validate the allocations.
  2. Testing technique: white box automated unit testing.
  3. Anticipated scripts: 4 tests are performed, one of which is a happy path and 3 are non-happy paths.

## **Deviation Validation**

* 1. Purpose: to verify that our functions correctly set a deviation value in the boundary of [0, 5].
  2. Testing technique: white box automated unit testing.
  3. Anticipated scripts: 6 tests are performed, 3 of which are happy paths and 3 are non-happy paths.

## **Prepare Rebalance Recommendation**

* 1. Purpose: to check our functions successfully calculate and construct a set of recommendation transactions by examining the preferred initial percentage and current holding of a portfolio.
  2. Testing technique: white box automated testing.
  3. Anticipated scripts: 2 tests are performed, one is a happy path and the other is a non-happy path.

## **Database Integration**

1. Purpose: to test the database connection as well as the correct behaviour of database related functions.
2. Testing technique: white box automated integration testing.
3. Anticipated scripts: 10 tests are performed, one of which is non-happy paths and the rest are happy paths.

## **Mock Fund System Integration**

* 1. Purpose: to test the interaction between our system and mock fund system as well as correct parsing of data fetched from the mock fund system.
  2. Testing technique: white box automated integration testing.
  3. Anticipated scripts: single happy path integration test is performed.

## **Create Initial Preference for a Portfolio**

* 1. Purpose: to check whether our functions create an initial preference with the valid portfolio id, deviation, fund id, the percentage for the valid customer id
  2. Testing technique: black box automated integration testing, manual integration testing.
  3. Anticipated scripts: 11 tests will be performed, including 4 manual integration tests and 7 black box automated integration tests. 2 of which are happy paths and 9 of which are unhappy paths.

## **Get Initial Preference of a Portfolio**

* 1. Purpose: to check whether our function retrieves the correct portfolio preference on the given portfolio id.
  2. Testing technique: black box automated integration testing, manual integration testing.
  3. Anticipated scripts: 2 tests, including 1 manual integration test and 1 black box automated integration test. Both tests are happy paths.

## **Update Deviation of a Portfolio**

* 1. Purpose: to verify that our function updates valid deviation to a portfolio.
  2. Testing technique: black box automated integration testing, manual integration testing.
  3. Anticipated scripts: 4 tests, including 2 manual integration tests and 2 black box automated integration test. For the manual integration tests, 1 test is a happy path and 1 is an unhappy path. The black box automated integration tests consist of 1 happy path and 1 unhappy path.

## **Create Rebalance Recommendation Transactions for a Portfolio**

* 1. Purpose: to verify that our functions generate a set of recommendation to rebalance the portfolio.
  2. Testing technique: black box automated integration testing, manual integration testing.
  3. Anticipated scripts: 3 tests, including 2 manual integration tests and 1 black box automated integration tests. All 3 tests are happy paths.

## **Update Allocations of a Portfolio**

* 1. Purpose: to verify that our function updates valid allocation to a portfolio
  2. Testing technique: black box automated integration testing, manual integration testing.
  3. Anticipated scripts: 3 tests, including 2 manual integration tests and 1 black box automated integration test. One of the manual integration tests is an unhappy path. All the rest tests are happy paths.

## **Execute Recommendation**

* 1. Purpose: to verify that our function applies the list of recommendations to rebalance the portfolio.
  2. Testing technique: black box automated integration testing, manual integration testing.
  3. Anticipated scripts: 3 tests, including 1 manual integration test and 2 black box automated integration test. One of the black box automated integration test is an unhappy path. All tests are happy paths.

# **NON-FUNCTIONAL TEST PLAN**

## **Performance Testing**

1. Purpose: to test the overall performance of the system.
2. Testing technique: manual testing
3. Details: When a page is loaded, and a button is clicked and redirects to another page, the time takes will be recorded and evaluated, as a user goes through the workflow. The workflow is identified in the UAT Test Script. We will repeat the workflow 4 times, and take the average of all page load time as our page load time.
4. Expected output:  
   1) Each time we will give the same input data, and the responses for 4 times should be stable and the same. Otherwise, the performance test fails on stability.

2) If the page load time is under 3 seconds, then the performance is good. If it takes 3-10 seconds, this is acceptable but not ideal. If it takes more than 10 seconds, then the performance test fails. The scale is based on a study about the average page load from 2018. (Source:https://www.machmetrics.com/speed-blog/average-page-load-times-websites-2018/)

## **Capacity Testing**

* 1. Purpose: to test the capacity of the system.
  2. Testing technique: manual testing.
  3. Details: With the provided 3 test users: nxqa3cu9r6, yv9q6aodfa, t8ej8u8q5n, we conduct a concurrent performance testing using three laptops to log in with individual customer ids and input different data. We will access each page and click buttons on the page at the same time.
  4. Expected output: If the data inputted by each user is correctly saved and retrieved for each user, then the capacity test pass. Otherwise, the test fails.

## **Cross-Browser Testing**

* 1. Purpose: to test how the system performs on different browsers.
  2. Testing technique: manual testing.
  3. Details: UI may look different on different browsers. For cross-browser testing, we open our web pages and go through each page on selected trending browsers, Chrome and Firefox.
  4. Expected output: If the UI is displayed the same and as what we expected on both browsers, then cross-browser test pass. Otherwise, the test fails.

## **Code Quality Testing**

* 1. Purpose: to test the overall quality of the system.
  2. Testing technique: static testing.
  3. Details: Our code quality testing is taken care of by the IntelliJ IDE Ultimate 2018.3.5. It provides static code analysis feature. The analysis included 1) Finding probable bugs 2) Locating the “dead” code 3) Detecting performance issues 4) Improving code structure and maintainability 5) Conforming to coding guidelines and standards 6) Conforming to specifications. When opening our source code in IntelliJ, it automatically scans the code and highlights the lines that have code quality issue. Since all of the members are using IntelliJ IDE Ultimate as a development tool, our code is tested every time we implement tasks.
  4. Expected output: when a technical user opens the source code in IntelliJ IDE Ultimate, there is no code quality related notification, then the test pass. Otherwise, the test fails.

# **TEST SCRIPTS**

1. **Header Validation** 
   1. isValidHeaderTrue()

Input: a header with a key of "x-custid" and alphanumeric customer id ("x-custid": nxqa3cu9r6)

Process: unit test

Expected output: true assertion pass

* 1. isValidHeaderTrue()

Input: a header with a key of "x-custid" and alphanumeric customer id ("x-custid": lalala)

Process: unit test

Expected output: true assertion pass

* 1. isValidHeaderFalse()

input: a header that does not contain "x-custid" as a key

Process: unit test

Expected output: false assertion pass

* 1. isValidHeaderFalse()

input: a header that has no value associated with "x-custid" key ("x-custid": “” )

Process: unit test

Expected output: false assertion pass

* 1. isValidHeaderFalse()

input: an empty header

Process: unit test

Expected output: false assertion pass

1. **Allocations Validation**
2. isValidAllocationsTrue()

Input: a set of valid allocations with all funds appears in the associated holdings

Process: unit test

Expected output: true assertion pass

1. isValidAllocationsFalseNotSumTo100()

Input: a set of allocations in which the funds’ percentage do not sum to 100

Process: unit test

Expected output: false assertion pass

1. isValidAllocationsFalseInvalidFundId()

Input: a set of allocations in which one of the funds not appear in the associated holdings

Process: unit test

Expected output: false assertion pass

* 1. isValidAllocationsFalseNegaivePercentage()

Input: a set of allocations in which one of the funds has negative percentage allocation

Process: unit test

Expected output: false assertion pass

1. **Deviation Validation**
2. isValidDeviationTrue() (test with in-bound deviation value)

Input: 1

Process: unit test

Expected output: true assertion pass

1. isValidDeviationFalse() (test with negative deviation value)

Input: -1

Process: unit test

Expected output: false assertion pass

1. IsValidDeviationTrue() (test with boundary deviation value)

Input: 0

Process: unit test

Expected output: true assertion pass

1. isValidDeviationTrue() (test with boundary deviation value)

Input: 5

Process: unit test

Expected output: true assertion pass

1. isValidDeviationFalse() (test with boundary deviation value)

input: 5.1

Process: unit test

Expected output: false assertion pass

1. isValidDeviationFalse() (test with out-of-bound deviation value)

Input: 6

Process: unit test

Expected output: false assertion pass

1. **prepare Rebalance Recommendations**
2. prepareRecommendationTest()

(test when the difference between the allocations and the actual holdings is outside of the deviation range set by customer.)

input: a set of holdings, a portfolio and an empty recommendation.

Process: unit test

Expected output: the true match assertion between the computed recommendation and the expected recommendation matched by fields passes.

1. prepareRecommendationTest\_withinDeviation()

(test when the difference between the allocations and the actual holdings is within the deviation range set by customer.)

input: a set of holdings, a portfolio and an empty recommendation.

Process: unit test

Expected output: the true match assertion between the computed recommendation and empty recommendation passes.

1. **Database Integration**
2. insertAndDeletePortfolioRecord() (test the correct behaviour of portfolio insertion and deletion)

Input: a valid portfolio

Process: integration test

Expected output: true assertion pass. The Portfolio exists in database after insertion and been removed after deletion.

1. doesInitialPreferenceExistTest() (test if the function correctly checks the existence of a portfolio in database)

input: a valid portfolio

Process: integration test

Expected output: true assertion pass for portfolio that exists in database; false assertion pass for portfolio that does not exist in database;

1. deleteAndInsertAllocationsTest() (test the correct behaviour of allocation insertion and deletion)

input: a set of valid allocations

Process: integration test

Expected output: true assertion pass. The allocations exist in database after insertion and been removed after deletion.

1. insertAndDeleteRecommendationsTest() (test the correct behaviour of recommendation insertion and deletion)

Input: A valid portfolio and recommendation id

Process: integration test

Expected output: true assertion pass. The recommendation exists in database after insertion and been removed after deletion.

1. insertAndDeleteTransactionsTest() (test the correct behaviour of transaction insertion and deletion)  
   Input: A valid portfolio, recommendation id and list of transactions

Process: integration test

Expected output: true assertion pass. The transactions associated with the given recommendation id exist in database after insertion and been removed after deletion.

1. updateTransactionsTest() (test the correct behaviour of update transaction)  
   Input: A valid recommendation id and old/new list of transactions  
   Process: Integration test  
   Expected output: true assertion pass. The transactions associated with the given recommendation id has been updated

1. **Mock Fund System Integration**
2. getHoldingsTest200()

Input: a header that contains a valid customer id(“nxqa3cu9r6”); a valid portfolio id (“2517972”).

Process: integration test

Expected output: a set of holdings associated with the customer id; true assertion on return object type pass.

1. **Create Initial Preference for a Portfolio**

Black-box integration tests:

1. createInitalPreferenceTest404InvalidPortfolioId()

Input: valid customer id nxqa3cu9r6 in the header and an HTTP request with an initial preference with non-existing portfolio id (“1”) in the request body

Process: integration test

Expected output: status code 404 with an error message “Portfolio id doesn’t exist”

1. createInitalPreferenceTest400InvalidDeviation()

Input: valid customer id nxqa3cu9r6 in the header and an HTTP request with an initial preference with a deviation not in [0,5] (“6”) in the request body and valid customer id nxqa3cu9r6 in the header

Process: integration test

Expected output: status code 400 bad request with an error message “your deviation should be in [0,5]”

1. createInitalPreferenceTest400InvalidFundId()

Input: valid customer id nxqa3cu9r6 in the header and a HTTP request with an initial preference with a non-existing fund id (“234561”) for portfolio 2517972 in the request body

Process: integration test

Expected output: status code 400 bad request with an error message “Input funds are not contained in your holdings or your portfolio preference should sum up to 100 and contain no negative number”

1. createInitalPreferenceTest400NotSumTo100()

Input: valid customer id nxqa3cu9r6 in the header and a HTTP request with an initial preference with allocations doesn’t sum up to 100 ("23456", 25) and ("23457", 70) in the request body

Process: integration test

Expected output: status code 400 bad request with an error message “Input funds are not contained in your holdings or your portfolio preference should sum up to 100 and contain no negative number”

1. createInitialPrererenceTest400NegativePercentage()

Input: valid customer id nxqa3cu9r6 in the header and a HTTP request with an initial preference with allocations contain negative percentage ("23456", -1) and ("23457", 101) in the request body

Process: integration test

Expected output: status code 400 bad request with an error message “Input funds are not contained in your holdings or your portfolio preference should sum up to 100 and contain no negative number”

1. createInitalPreferenceTest404InvalidCustomerHeader()

Input:header has non-existing customer id (“lalala”) along with a HTTP request with a valid initial preference

Process: integration test

Expected output: status code 404 not found

1. createInitalPreferenceTest200()

Input: an request with a valid initial preference in request body and valid customer id in the headers (when a portfolio doesn’t have an initial preference and an initial preference is given with a deviation range from [0,5], type is either “fund” or “category”, and allocations contains fund ids from under this portfolio holding and percentages for allocations sum up to 100)

Process: integration test

Expected output: 200 status code and initial preference data in the response body

Manual integration tests:

1. Create Initial Preference with Invalid Deviation

Given: User logs in with customer id nxqa3cu9r6 on the Home page. Click the GO button to the Portfolios page. Click on a portfolio that doesn’t have an initial preference. This leads the user to the Initial Portfolio page. On Initial Portfolio page, click Create Initial Preference button to go to the Creation Initial Preference page.

When: User enters an invalid deviation (a negative deviation, a deviation that’s bigger than 5 and is an integer, a deviation that’s bigger than 5 and is a double), and clicks outside the box

Then: The deviation box is highlighted in red. An error message “Invalid deviation” is shown

1. Create Initial Preference with Percentage not sum up to 100

Given: User logs in with customer id nxqa3cu9r6 on the Home page. Click the GO button to the Portfolios page. Click on a portfolio that doesn’t have an initial preference. This leads the user to the Initial Portfolio page. On Initial Portfolio page, click Create Initial Preference button to go to the Creation Initial Preference page.

When: User sets an initial preference with allocations doesn’t sum up to 100("23456", 25 and "23457", 70), and clicks outside the box

Then: The percentage boxes are highlighted in red. An error message “Invalid Percentage” is shown

1. Create Initial Preference with Negative Percentage

Given: User logs in with customer id nxqa3cu9r6 on the Home page. Click the GO button to the Portfolios page. Click on a portfolio that doesn’t have an initial preference. This leads the user to the Initial Portfolio page. On Initial Portfolio page, click Create Initial Preference button to go to the Creation Initial Preference page

When: User sets the allocation with a negative percentage ("23456", -1 and "23457", 101), and clicks outside the box

Then: The percentage boxes are highlighted in red. An error message “Invalid Percentage” is shown

1. Create Initial Preference

Given: User logs in with customer id yv9q6aodfa on the Home page. Click the GO button to the Portfolios page. Click on a portfolio that doesn’t have an initial preference. This leads the user to the Initial Portfolio page. On Initial Portfolio page, click Create Initial Preference button to go to the Creation Initial Preference page

When: User enters all valid data and click the Save button

Then: User is directed to the Portfolio page with correct holding information displayed

1. **Get Initial Preference of a Portfolio**

Black-box integration test:

* 1. getInitalPreference200()

Input: an HTTP GET request with a valid portfolio id (“2517972”) and valid customer id in the header(“nxqa3cu9r6”)

Process: integration test

Expected output: status code 200 and retrieved initial preference data under this portfolio

Manual integration test:

* 1. Get Initial Preference

Given: User logs in with customer id nxqa3cu9r6 on the Home page. Click the GO button to the Portfolios page. Click on a portfolio that doesn’t have an initial preference. This leads the user to the Initial Portfolio page.

When: Click on the portfolio with id 2517972.

Then: User is directed to the Portfolio screen with all the initial preference information displayed.

1. **Update Deviation of a Portfolio**

Black-box integration test:

* 1. deviationPUT200()

Input: a PUT request with a valid portfolio id in the request URL and a valid deviation ([0,5]) in the request body

Process: integration test

Expected out: status code 200 along with new deviation amount in request body

1. deviationPUT400()

Input: a PUT request with a valid portfolio id in the request URL and an invalid deviation in the request body

Process: integration test

Expected out: status code 400

Manual integration test:

* 1. Update Deviation with a valid input

Given: A user clicks the Update button on the Portfolio page and is directed to the Deviation Update page

When: The user enters a deviation ([0,5]) in the input box and clicks the Save button

Then: The user is redirected back to the Portfolio screen with the new deviation listed

* 1. Update Deviation with an invalid input

Given: A user clicks the Update button on the the Portfolio page and is directed to the Deviation Update page

When: The user enters a deviation (a negative deviation, a deviation that’s bigger than 5 and is an integer, a deviation that’s bigger than 5 and is a double) in the input box and clicks outside the input field

Then: The deviation box is highlighted in red. An error message “Invalid deviation” is shown

1. **Create Rebalance Recommendations for a Portfolio**

Black-box integration test:

* 1. getRecommendationPOST200()

Input: a valid customer id nxqa3cu9r6 in the header and a valid portfolio id 2517972 in the request URL

Process: integration tests

Expected out: status code 200 along with a recommendation that could balance the customer’s holdings back to his or her mix asset preference.

Manual integration test:

* 1. Create Rebalance Recommendations Transactions

Given: A user with id nxqa3cu9r6 goes to the Portfolio page and wants to rebalance the portfolio 2517972

When: The user clicks on the Rebalance button

Then: A list of rebalance recommendations transactions is displaced on the Rebalance page

* 1. Create Rebalance Recommendation when the Difference between the Current Holding and the Initial Preference is within the Deviation Range:

Given: A customer has a portfolio where the difference between the current holding and the initial preference is within the deviation range

When: The user clicks on the Rebalance button

Then: The user will not be directed to the Rebalance page. A message “Your portfolio is balanced.” will be displaced

1. **Update Allocations of a Portfolio**

Black-box integration test

* 1. allocationsPUT200()

Input: a valid allocations associated with a portfolio id

Process: integration test

Expected out: status code 200 and returned updated allocations

Manual integration test

1. Update Allocation with a valid input

Given: A user clicks the Update button on the Portfolio page for the allocation update and is directed to the Allocation Update page

When: The user enters a valid allocation in the input box and clicks the Save button

Then: The user is redirected back to the Portfolio screen with the new deviation listed

1. Update Deviation with an invalid input

Given: A user clicks the Update button on the the Portfolio page and is directed to the Allocation Update page

When: The user enters an allocation (percentage not summed up to 100, negative percentage) in the input box and clicks outside the input field

Then: The allocation box is highlighted in red. An error message “Invalid allocation” is shown

1. **Execute Recommendation**

Black-box integration test

* 1. executeRecommendation\_200()  
     Input: A header containing a valid customer id (“nxqa3cu9r6”), a valid portfolio id (“2517972”), a recommendation id (“613”) and a list of transactions for that recommendation.

Process: Integration test

Expected output: status code 200 and returns the customer’s updated holdings.

* 1. setRecommendation\_400()  
     Input: A header containing a valid customer id (“nxqa3cu9r6”), a valid portfolio id (“2517972”), a recommendation id (“613”) and an empty list of transactions for that recommendation.

Process: Integration test

Expected output: status code 400

Manual integration test

1. Execute recommendation with success

Given: The user is on the Rebalance page and wants to execute the list of rebalance recommendations transactions

When: The user clicks the execute button

Then: The user is directed back to the Portfolio page with the updated portfolio

# **UAT TEST SCRIPT**

* Log in with customer id nxqa3cu9r6 on the Home page. Click the Sign In button to show the Portfolios page. The Portfolios page should have 2 portfolios with ids 2517972 and 579565575.
* On the Portfolios page, click on portfolio id 2517972which has an initial preference in the database. The user should be directed to the Portfolio page. The Portfolio page should show holdings for two funds: 1132 units of 23456 and 1341 units of 23457 under portfolio id 2517972. Also, it should show the user the initial preference where fund 23456 occupies 30% and fund 23457 occupies 70%, the deviation is 5.0, and the type is “fund”.
* If the user’s current holding under this portfolio has deviated off the initially preferred percentages by over the allowed deviation amount, a Rebalance button will show up on the page. For portfolio id 2517972, the current holding for fund 23456 occupies around 60% and fund 23457 occupies around 40%. Therefore the Rebalance button should appear. Clicking the Rebalance button should direct the user to the Rebalance page.
* On the Rebalance page, it displays a set of rebalancing recommendation transactions to buy 183 units of fund 23456 to increase its allocation from 60% to 70% and to sell 329 units of fund 23457 to decrease its allocation from 40% to 30%. Click the Execute button to execute the recommendation, once the execution succeeds, this directs the user back to the Portfolio page and the current holding under portfolio id 2517972 should be updated and fund 23456 should occupy approximately 70% and fund 23457 should occupy approximately 30% of total holding.
* On the Portfolio page, click the Back button to go back to the Portfolios page, the Portfolios page should still show 2 portfolio ids 2517972 and 579565575. Clicking on portfolio id 579565575 which doesn’t have an initial preference should redirect the user to the Initial Portfolio page.
* On the Initial Portfolio page, the user should see the current holding information: some units of fund 23459, some units of fund 23458, some units of fund 23503 and some units of fund 23456 under portfolio id 579565575, and there is no initial preference information. Click on the create initial preference button. A form will pop up on the screen. Enter the data to finish the initial preference (i.e. enter 0 for deviation input box and select “fund” for type. Under allocations, enter 30 for fund 23459, 10 for fund 23458, 40 for fund 23503 and 20 for 23456), then click the Save button to complete creating an initial preference. The form will be closed.
* The Portfolios page is revised and a pie chart of the target allocations shows on the right side of the page . The target allocation pie chart shows the initial preference where fund 23459 occupies 30%, fund 23458 occupies 10%, fund 23503 occupies 40% and 23456 occupies 20%, the deviation is 0, and the type is “fund”.
* On the Portfolio page, clicking Update Deviation button will trigger a pop-up window contains an Update Deviation form, where the user can enter a value in [0,5] in the input box. Click the Save button, the form is closed and the title of the target allocation pie chart contains the new deviation value the user just entered.
* On Portfolio page, clicking Update Allocations button will trigger a pop-up window contains an Update Preference form, where the user can enter a valid percentage number for each fundId (23503,23456,23459, 23458). All the input percentages should be positive numbers and sum up to 100. Clicking the Save button, the form is closed and the target allocation pie chart contains the new allocations values the user just entered.

# **SECURITY TESTING AND TEST DATA APPROACH**

1. **Security testing**

In our project, each request towards our API endpoints requires a header that contains a customer id. Therefore, identity verification would be done by confirming that the customer id actually exists and matches the records. To achieve this, we verify the existence of that customer id on mock fund system before providing any further service. In our implementation, each endpoint does the check and is expected to return HTTP NOT\_FOUND error when it receives a non-existing customer id.

1. **Test data**

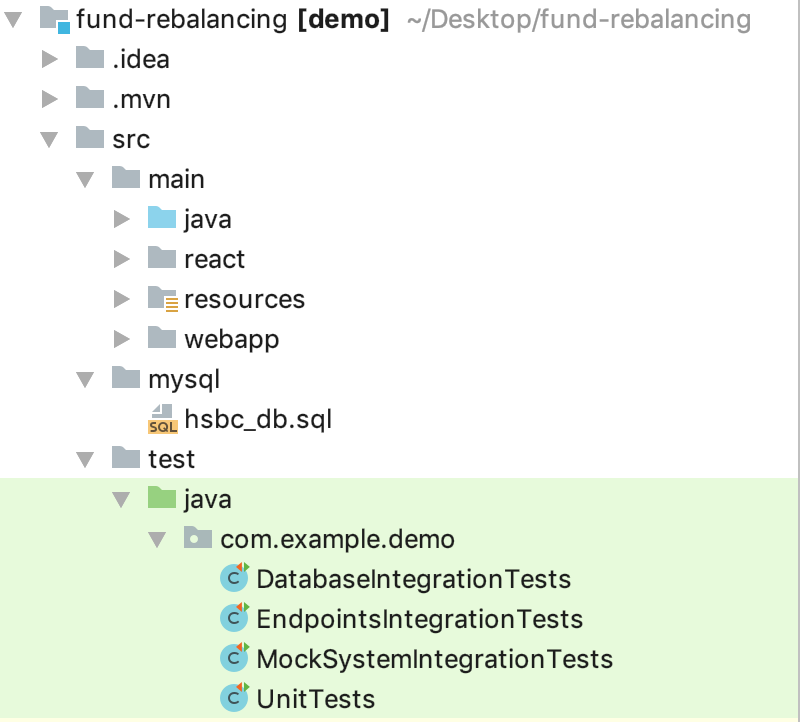
For unit testing and database integration testing, since we do not intend to leverage the HSBC fund system, we mock our test data mainly for cleanness and variation to serve each test case. For instance, the deviation values are designed to test boundary cases and different preference allocations are designed to test recommendations giving in different cases.

For HSBC fund system integration test, SAT and UAT, we leverage HSBC provided test data to ensure a correct connection with mock fund system and the proper behaviour of our endpoints in a production-like testing environment.

# **REGRESSION TESTING**

All unit and integration tests from the Functional Test Plan will be compiled in the regression testing package. More details about the tests can be found in Functional Test Plan and Test Script Appendix sections.

**TEST SCRIPT APPENDIX**

Source: <https://gitlab.com/cpsc319-2018w2/hsbc/rest/fund-rebalancing/tree/master/src/test/java/com/example/demo>  


1. Header Validation /UnitTests
2. Allocations Validation /UnitTests
3. Deviation Validation /UnitTests
4. Prepare Rebalance Recommendation /UnitTests
5. Database Integration /DatabaseIntegrationTests
6. Mock Fund System Integration /MockSystemIntegrationTests
7. Create Initial Preference for a Portfolio Test /EndpointsIntegrationTests
8. Get Initial Preference of a Portfolio Test /EndpointsIntegrationTests
9. Update Deviation of a Portfolio Test /EndpointsIntegrationTests
10. Create Rebalance Recommendation for a Portfolio Test /EndpointsIntegrationTests
11. Update Allocations of a Portfolio Test /EndpointsIntegrationTests
12. Execute Recommendation Test: /EndpointsIntegrationTests