SSDL Systems and Software Development Lab

Cognitivity

Test Plan

Submitted by: Daniel Lyubin

Mark Erlich

Ben Hunter

Guy Shemesh

Ophir Katz

Peer Sagiv

Supervised by: Tanya Brokhman

Table of Contents

1. Introduction

1.1 Purpose

This test plan describes the testing approach and overall framework that will drive the testing of the Cognitivity project. The document introduces:

- Test Strategy: rules the test will be based on, including the givens of the project (e.g.: start / end dates, objectives, assumptions); description of the process to set up a valid test (e.g.: entry / exit criteria, creation of test cases, specific tasks to perform, scheduling, data strategy).
- Execution Strategy: describes how the test will be performed.

1.2 Project overview

TO BE TAKEN FROM HLD REPORT!

2. Test strategy

2.1 Test objectives

The objective of the test is to verify that the functionality of the Cognitivity project works according to the specifications. The final product of the test is twofold:

- A production-ready software;
- A set of stable test scripts that can be reused for Functional test execution.

2.2 Test assumptions

ADD ENVIRONMENT THAT THE DEVELOPMENT OF THE PROJECT WAS IN. e.g. Angular was approved by google before use
MySql approved by oracle before

2.3 Scope and Levels of Testing

2.3.1 Functional Test

PURPOSE: Functional testing will be performed to check the functions of application. The functional testing is carried out by feeding the input and validating the output from the application.

METHOD: The test will be performed according to Functional scripts/Test procedures with a well-defined PASS/FAIL criteria.

2.3.2 User Acceptance Test (UAT)

PURPOSE: this test allows the end users to complete one final review of the system prior to deployment.

METHOD: Will be performed manually by team members according to written test cases.

3. Validation and Defect Management

It is the responsibility of the tester to open the defects, link them to the corresponding script, assign an initial severity and status,

It is the responsibility of the developer to retest after a fix is provided and close the defect.

Defects will be categorized according to the following severity status:

Severity	Impact
1 (Critical)	● This bug is critical enough to crash the system, cause file corruption, or cause potential data loss ● It causes an abnormal return to the operating system (crash or a system failure message appears). ● It causes the application to hang and requires re-booting the system.
2 (High)	● It causes a lack of vital program functionality with workaround.
3 (Medium)	● This Bug will degrade the quality of the System. However there is an intelligent workaround for achieving the desired functionality - for example through another screen. ● This bug prevents other areas of the product from being tested. However other areas can be independently tested.

4 (Low)	• There is an insufficient or unclear error message, which has minimum impact on product use
5 (Cosmetic)	• There is an insufficient or unclear error message that has no impact on product use.

4. TEST ENVIRONMENT

….

5. Test cases

5.1 Function tests

5.1.1 Backend MVC Tests:

- Controller Tests:

  Objective: To assert that the controller classes are receiving, transferring and manipulating data properly.
  Defects categorization:
- Critical – The data of the http message was not transferred correctly, or wasn't returned correctly.
- High – Some formats was not understood in the transferred data.
  Input: A HTTP Message
  Test logic:
  1. Send a HTTP message from
  2. Check in the backend that the data is received correctly with no errors.
- Controller Tests:

  Objective: To assert that the data conversion from JSON representation to a Java object is done correctly.
  Defects categorization:

- Critical – The conversion crashes because the data was not in the correct format.
- High – The conversion is faulty because some fields of the objects were not initialized properly.

Input: JSON representation of an object (Question, TestID etc...)

Test logic:

1. Call to a conversion function on the JSON data.
2. Check that for every field, the value it holds is the correct one according to the mapping in the JSON object. i.e., if in the JSON object there was a mapping $questionID\ :\ 111$ then the object will hold the value 111 in the field questionID.

Objective: To assert that the data is being pushed and pulled correctly from the data server by the defined methods.

Enter criteria: The data server is initialized and ready.

Exit criteria: The data was pulled from the server correctly with no errors.

Defects categorization:

- Critical – The data was not pushed/pulled from the server at all.
- High – Some of the data was pushed/pulled, and some wasn't.
- Low – The data was successfully pulled from the server, but some entries are missing.

Input: Some data to push to the server (including different types of data, to different table, with different fields etc...)

Test logic:

1. Push some data to the server (doesn't require use of function, just MySQL queries on some info).
2. For each method that requests data from the server (whether it requests a single entry, or a list of entries by some criteria), call it and receive the data.
3. Test that the received data is the same as the data that was pushed in stage 1 of the test, according to the criteria.

Objective: To assert that the data on the server is being updated correctly.

Enter criteria: The data server is initialized with data (tables) and ready.

Exit criteria: The data was updated as expected.

Defects categorization:

- Critical – The data was not updated at all.
- High – The data was updated, but the old data was not erased from the data server.
- Low – The data was updated but the values are not as expected.

Input: Some data to update/delete.

Test logic:

1. Call to an update/delete method on the data.

2. assert that the data was updated correctly by pulling it and checking the fields. Or (if delete), assert that the data was deleted and is not in the server by querying it.

5.1.2 Manager’s Dashboard Tests:

The dashboard is GUI, so there will be only UAT.

- Test List test:
  - **Objective**: check if the tests shown in the list are the tests of the manager.
  - **Test Logic**:
    - Log the tests returned from the service.
    - Click on the ‘Tests’ label on the navbar.
    - Check that every test in the logged tests is in the table, and no other tests are shown.
  - **Defect categorization**
    - **High**: Data isn’t consistent
  - **Enter Criteria:** There is test information in the DB
  - **Exit Criteria:** Tests are shown in the table
- Question List test:
  - **Objective:** check if the questions shown in the list are the questions that the manager added.
  - **Test Logic:**
    - Log the returned questions from the service.
    - Click on the ‘Questions’ label on the navbar.
    - Check that every question in the logged questions is in the list, and no other questions are shown.
  - **Defect categorization**
    - **High:** Data isn’t consistent
  - **Enter Criteria:** There is question information in the DB
  - **Exit Criteria:** Questions are shown in the table
- Navigation Test:
  - **Objective**: Insure that navigating between pages is correct and doesn’t disrupt the design of the page.
  - **Test Logic:** Navigate to different pages using the labels and links available
  - **Enter Criteria:** The dashboard page has loaded
  - **Exit Criteria:** The dashboard is intact. Specifically, the navbar is in its right place and components outside the dashboard are not appearing.
  - **Defect categorization**

- **Critical:** There are components not related to the dashboard, shown in the dashboard.
- **High:** Wrong components are router or routing is not working.
- **Medium:** The navbar is not showing.

5.1.3 Questions creation and view tests:

The questions creation and view is GUI, so there will be only UAT

- **Creating a question test:**
    - **Objective:** check by the manager if he can add a new question to a test
    - **Enter criteria:** The manager is logged in and already created a test.
    - **Exit criteria:** The manager can view the question in the test.
    - **Defects categorization:**
        - Critical - the question didn't added at all to the test.
        - High - the question wasn't added correctly.
        - Low - the view isn't satisfying enough.
    - **Test logic:**
        - **Entering the form for creating a question -** The manager should enter the dashboard, and to access a specific test that he wishes to add a question, after that he needs to edit the test and to access the form for creating a question.

        - **Filling the form for creating a question -** The manager needs to fill all relevant details in the form - The question itself, type of question (multiple choice and the required organization of the answers, open question, rating question), the answers (depending on the type of the question).

        - **Submitting the form -** After finishing filling the form the manager will submit the form.

        - **Enter criteria:** The manager is logged in, and he already created a test.
- **Viewing the different types of questions:**

Objective: to show you how you should write a test plan by giving you some examples

Enter criteria: you already wrote an HLD and have the list of the functionalities your product will supply
Exit criteria: Students managed to write a test plan for their project
Defects categorization:

- Critical – no one understood what is required from him/her
- High – some section was not clear
- Low – additional examples were requested
- Cosmetic – the font is ugly

Input: ....
Test logic:

3. distribute this document to students
4. request them to go over it and get back with questionsto

5.1.4 Hibernate coherence with DB:

Objective: To make sure we can read data from db

Enter criteria: There is a db up and running
Exit criteria: The reading has finished
Defects categorization:
High – if there isn't an option to read data, the managers won't be able to analyze tests
Test logic:

1. enter data to the db, to each of the tables
2. read data from the db using hibernate
3. check that the data is the same as we entered

Objective: To make sure we can write data to db

Enter criteria: There is a db up and running
Exit criteria: The writing has finished
Defects categorization:
Critical – if there is a defect at writing, we will lose data
Test logic:

1. write data to the db using hibernate, to each of the tables
2. read data from db
3. check that the data is the same as we entered

5.1.5

5.2 UAT tests