Testing

Primary Author: Justin Renneke

General Testing Strategy

We have explored a few different testing frameworks and UIs including JUnit based JWebUnit and Selenium front end web application testing but decided setting these things up would be more effort than it was worth for a project of this size and scope. Instead, we plan to test in a systematic way by hand and according to the tests outlined in this list. Unit tests will be done during development as each function is implemented to ensure correct implementation and integration. Integration testing will be done at least before the end of each sprint and anytime a change is made to a database or a page linked to other pages is added to the site. Regression testing will be done at the end of each sprint. User acceptance testing will be done near the end of development before 'shipment/submission' of the final product.

Edge Cases

Edge cases are cases that rarely occur, but have the potential to cause failure of the function or application if unaccounted for. Testing specifically with edge cases in mind can help us bound the behavior of our functions. This bound will help give structure to our functions by providing us a 'working window' in which we can be reasonably certain that our functions will respond the way we want based on inputs that lie between the extremes of tested edge cases. To use an example we have listed in our unit tests: a username provided by the user should have a length > 0 (otherwise it is nonsensical/useless) but <= 255 (because the database field for username is limited to max 255 characters). If we test these edge cases and our function handles them correctly by rejecting them, and our function handles a typical and correct username length correctly by accepting it then it should also handle all other cases in the normal range correctly.

Typically, edge cases are handled in relation to unit testing and we plan to try and think of as many edge cases as we can. To that end, we have updated our unit tests to include as many edge cases as we can think of at this time.

User Acceptance Test Scenarios:

User acceptance testing is done by the end user to ensure all desired functionality is there and working the way the client actually wants it to work vs how the software developers think it should work. This is typically done as the last stage after other types of testing are complete and before shipment of the end product.

1. Test the outlined use cases by providing the end user a list of the use cases and any objects required to complete the tasks (for example, access to a data set for the Upload

Data Set use case).

2. This testing environment should emulate real world usage conditions.
3. The flow of the system should be tested from start to finish. Can the user create a new account, log in, upload a data set, generate or upload a manifest for the new data set, search the database to find the uploaded data set via browse manifest, view it, download it, and log out when done without any issues?
4. Test the outlined use cases by providing the end user a list of the use cases and any objects required to complete the tasks (for example, access to a data set for the Upload Data Set use case).
5. The following use cases should be tested:
    1. Browse Manifest
    2. Contribute to Existing Dataset
    3. Download Info
    4. Generate Upload Manifest
    5. Search on Manifest
    6. Upload Data Set

Unit Test Scenarios:

The testing of single methods or functions during the coding process to validate functionality.
NOTE: We know that unit tests are meant to be run on specific, individually coded functions. However, due to the nature of the website and the languages we are using (read: php), it is often difficult to separate out individual coded function into a self contained unit for testing. So, while many of these unit tests lean towards the integration side of things due to database interactions, we have chosen to frame our unit tests around logical cases of functionality instead. These tests are conducted by hand for similar reasons.

1. Early database establishment unit tests to verify that insert, update, and delete database actions are working correctly before continuing on to further system development.

A foreign key reference might have been handled incorrectly leading to improper association of related tables
Failure cases:
-An insert, update, or delete statement is executed but a follow-up select statement does not return the expected result
-Referencing a foreign key relationship returns the incorrect result

2. Login(username, password)

Login with good info should flag the user as logged in and create the corresponding session variable with the correct access level
-Good info is constituted by a username and password/password hash registered in the database
-Bad info includes the following combinations: Good username/Bad password; Bad username/Good password; Bad username/Bad password
Login should flag a user as logged in in a session variable so logged in status persists

throughout the website
Login with bad info should initiate a notification of rejected login
Failure cases:
-Username and password/password hash that aren't in database are allowed to log in
-Good username/password are not logged in
-User gets logged in but has incorrect permission level
-Logged in user navigates to another page but is not recognized as logged in because the session variable wasn't set correctly

3. CreateNewUser(username, email, first name, last name, phone)

Ensure invalid parameters are handled correctly (i.e. things such as username too long, etc)
Ensure that new user information is input into proper database fields
Verify username matches password in database
Failure cases:
-Username is too long(>255 characters) or null
-Email is too long(>255 characters) or null
-First name is too long(>255 characters)
-Last name is too long(>255 characters) or null
-Phone is too long(>255 characters)
-SQL insert statement is executed to create a new user in the database, but a select statement attempting to return matching info fails to find it
-A search of the database fails to find a matching username and password pair

4. DeleteUser(username)

Ensure invalid parameters are handled correctly
If the user to be deleted was not found in the database, the user should be notified of this
Search database for user after execution to ensure user information is deleted
Failure cases:
-Username >255 characters or 0 characters
-The attempted deletion of a username known to not exist in the database does not return notification indicating user was not found
-An attempt is made to delete a user and that user cannot be found in the database, but the function indicates a successful deletion anyway
-A database search for a user that should have been deleted returns the user's info indicating the user wasn't actually deleted

5. UploadDataSet(Dataset url)

Bad datasets or bad links to datasets should be rejected
Check database to ensure dataset is correctly inserted into proper database field
Failure cases:
-A dataset of size 0
-A non-functional link to a dataset
-A dataset that has supposedly been inserted into the database cannot be found there with a search of the database

6. CreateManifest(StandardVersions, FirstName, LastName, UploadDate, UploadComment, UploadTitle, DsTitle, DaTimeInterval, RetrievedTimeInterval, DsDateCreated, DataSet)

Ensure required data (database table NON NULL fields) is input before attempting to create the

manifest.
Ensure invalid parameters are handled correctly (things such as a string being too long, etc)
Verify that the generated manifest returned by function is valid.
Failure cases:
-Missing NON NULL variables (FirstName, LastName, UploadDate, Dataset)
-StandardVersions >255 characters
-FirstName > 255 characters or null
-LastName >255 characters or null
-UploadDate not datetime format or null
-UploadComment > 1000 characters
-UploadTitle > 1000 characters
-DsTitle >1000 characters
-DsTimeInterval > 255 characters
-RetrievedTimeInterval > 255 characters
-DsDateCreated not datetime format
-Examination of a generated manifest JSON file does not include all data or holds inaccurate data

7.UploadManifest(FirstName, LastName, UploadComment, UploadTitle, UploadDate, DataSet, JSONFile)

Ensure required data (database table NON NULL fields) is input before attempting to create the manifest.
Ensure invalid file types are not accepted.
A file of size 0 should be rejected.
Failure cases:
-Missing NON NULL variables (FirstName, LastName, UploadDate, Dataset)
-A user tries to upload a non-JSON type file
-A user tries to upload a file of size 0

8.CreateSNC(FirstName, LastName, DateCreated, FileName)

Ensure all database NON NULL parameters are provided and other parameters are valid (i.e. FileName not too long, etc)
Ensure SNC is tied to a manifest via foreign key relation maintained in m_x_snc table
Failure cases:
-Missing NON NULL variables FirstName, LastName, or DateCreated
-Filename >255 characters
-DateCreated not of type datetime
-Database query based on manifest foreign key relation doesn't return the appropriate SNC

9.browseManifests(searchString)

A search that fails to find a match should notify the user
Failure cases:
-A search that fails to find a match doesn't notify the user

10. To be expanded as more features are implemented in future sprints...

Regression Testing:

Periodic testing done whenever changes have been made to the system to validate pre-

existing functionality is still there. In other words, test everything that was already there to make sure we didn't break anything after adding new things to the system.

1. Ensure login still works as expected
2. Ensure non-authorized user can't access any restricted pages by going directly to the corresponding URL via the navigation bar
3. Test all links to ensure they lead to the correct URL
   *Ensure all pages led to by these links load correctly
4. Test all Forms
   Check field default values Ensure only correct input type is accepted (i.e if a field is meant to accept a zip code, then it should only accept numbers)
   *Verify user input is stored and used by the system
5. Ensure database integrity

Integration Testing:

Test interaction between different units of the system that must communicate with each other. Make sure these components are interacting correctly.
When to perform: Before the end of each sprint.

1. All links within the application should go to the correct page.
2. Anytime the application is given functionality to interact with the database, the integration of these components should be tested. Correctness of update, insert, delete, etc into the database should be verified and the proper display of any information retrieved from the database should be verified.
3. The log in system should be integrated across all web pages of the application. I.e. the logged in status of a user should persist across all web pages via a session variable until the session is ended and this should be tested for anytime a new page is added to the system.
4. Anytime information is passed between web pages, it should be verified that this information is correctly received and displayed.

Verification vs Validation:

Verification answers the question, 'Are we building the system in the correct way according to the requirements and design specifications?' Is the system well-engineered and error free? This is done with unit testing, integration testing, and regression testing.

Validation answers the question, 'Are we building the right product to satisfy the customer's needs?' This is done with user acceptance testing.