Dank Devz - Test Plan

Continuous Integration, Integration and Unit Testing

This is a flexible test plan that is subject to change and may be updated as development progresses.

Testing Strategy

1. Refer to Git flow described in the README.
2. Upon completion of a feature branch, write and test a unit test using QtTestLib.
3. Any team member other than the author will peer review the pull-request before it is merged into the integration branch `testing`.
4. Qt Testing framework allows for multiple application profiles to be used and lets developers run the application by itself or within the testing profile which utilizes the same code library.
5. Each Developer can run the tests by changing profiles
    1. Each new feature implemented needs to have a test written for it as well.
    2. These tests will be added to a overall regression testing suite.
6. If all passes, `testing` will be merged into `develop` for the development testing.
    1. If regression test pass from each merge then develop can be merged into `master`.
7. Use a continuous integration server to poll and compile the project off the develop branch as each commit is made.
    - This provides a way for others to see if code that has been pushed is correctly compiling.

Roles and Responsibilities

- Each member is responsible for writing their own unit tests.
- Each member is responsible for verifying the tests pass before merging the feature branch into develop.
- Product owner is responsible for checking that develop is passing the regression suite.

Purpose

- Verify that database opens, adds and removes data, and is persistent between executions of the program.
- Verify that the menu items are added and removed when the add / remove buttons are clicked.
- Verify that push buttons are functional.

Features to be tested

- Database
  - Production Database
  - Testing Database

Testing tasks

1. Database
   - Testing if Database opens `testOpen()`
   - Testing adding **Valid** items : `testValidAddMenuItem()`
   - Testing adding **Invalid** items : `testInvalidAddMenuItem()`
   - Testing removing menu items : `testRemoveMenuItem()`
   - Testing getting restaurant Ids : `testGetRestaurantId()`
   - Testing add restaurant to database : `testAddRestaurant`
   - Testing removing restaurant from database : `testRemoveRestaurant()`
   - Testing getting the Id of an item : `testGetItemId()`
   - Testing get all restaurant information from the database : `testGetRestaurants()`

- Testing add items to the cart and summing up the total price of all items : `testGetCartTotal()`
- Testing admin access by entering password, and querying the database to check if the password is correct : `testAuthenticateAdmin()`
- Testing getting all the distances for a restaurant by giving the function an ID : `testGetDistanceFromRestaurantByID()`
- Testing getting all the distances for a restaurant by giving the function the restaurant name : `testGetDistanceFromRestaurantByName()`
- Testing tests getting all restaurant ids: `testGetAllRestaurantIds()`

2. Push Buttons
    - Test clickable regions.
    - Test if button image changes upon entry of clickable region.
    - Test if button image changes upon exit of clickable region.
    - Test if button click signal is registered by other parts of the program.
    - Test if button image changes upon click.

Test framework

- *QT Testing Framework*
    - Using QtTests and built in unit testing for Qt

Test environment

- UNIX / Linux / OSX / Windows
- *JetBrains TeamCity* : Continuous Integration Server

Schedule

- Upon completion of a feature branch and before integration this unit test must be successfully run.
- Unit Test per feature development
- On Commit testing
    - Using Continuous Integration server that polls the Git repository every 5

minutes.

Deployable Guarantee

- Branch `master` will always be in a deployable state.

Test design techniques - entry/exit criteria

1. Entry
    - Table `distances` contains *Predefined values*
    - Table `cart` contains *no values*
    - Table `restaurants` contains *Predefined values*
    - Table `items` contains *Predefined values*
2. Exit
    - Table `distances` contains *Predefined values*
    - Table `cart` contains *no values*
    - Table `restaurants` contains *Predefined values*
    - Table `items` contains *Predefined values*

Resource Links

- Git Respository Server : `https://www.github.com/dank-devz/effective-octo-meme`
- Continuous Integration Server : `https://teamcity.dank-developer.com:8080/`