Test Plan



##### Project By:

##### Justin Goulet; Steven Clark, Jeremy Pietersma, Christian Mathias, John Orcino

##### 

##### 

##### 

##### 

##### 

##### 

Test Plan

Test-Taker Application - Team A

# Design Patterns

## ShutdownManager

ShutdownManager is very simple singleton. In order to be hooked into the shutdown process of the runtime it needed implement the thread or runnable interface. The run function used for this is not defined as static so this memberless class needed to be a singleton.

## LibraryController

LibraryController is a fully static class, with initialization, and functionally a singleton without the pattern. It is a monolithic system which only ever exists once. Since Java 8 features static initializer blocks it was possible to get rid of the entire .getinstance() part of the singleton pattern in favor of just calling the public methods of the class directly. It manages the persistent storage of backend objects across runs of the program.

# Testing

### Backend System Testing

FunctionalityTester is a class in the backend package used to provide an in-package main function from which methods of backend can be invoked for testing. A separate main statement allows testing via console IO without disturbing the real main function in DriverManager. Testing was not extensive and many errors took debugging from running actual use cases through the GUI.

MakeExample functions were part of the used backend classes. They provided a new example item of that class when called. This both helped generate sample data for testing and provided a small amount of example code for object creation.

The majority of methods of backend classes or LibraryController return a null pointer when nothing would be returned. Originally meant as a way of coding for bad input, the resulting null-pointer exceptions turned out to provide a good debugging tool in their stack traces.

Testing with the use cases revealed that there was a mistake in the implementation of a few backend classes. Tests have a transient list of references for their Questions (which are stored separately on disk) mostly as a convenience feature. When a Test was restored from the database this transient field was left as a null pointer. The function that was supposed to repopulate it failed to do so when the test had no Questions. This supposed corner case turned out to be the common case with how the UI was structured. Once it was fixed further code was added to ensure that this list is always repopulated if necessary before use.

### User Interface Testing

#### JFrame Functionality -

Within the main JFrame, run by DriverManager.java, there are two main JPanels

1. SideMenu
   1. Testing of the Side Menu necessitates the use of a user.
   2. The user must click an item from the Side menu in order for a new page to be selected.
      1. If a new page was not created, the page does not change
      2. If a page is a copy of a different page, the original is overwritten by the copy and the original displays a black view.
2. CustomPage -

This is the page that creates each panel with unique components

* Logo\_Only\_Type
  + Creates a frame with a sized logo in the center. When no image is found, it is blank.
* One\_Button\_Type
  + Basic Frame with a logo (previously loaded)
  + Single button (using CustomButton.java)
  + When no action is sided with a button, button name prints to console.
* Two\_Button\_Type
  + One\_Button\_Type, but buttons are halved in width to fit on same line
* Three\_Buton\_Type
  + Two\_Button\_Type, but now has two rows. The first row is halved button widths, with the bottom button full width.
  + CustomPage was designed to hold a max of ten buttons, adjusting the height as more are added, and ensuring when an odd number exists, the bottom is full width while the rest are halved.
  + Action Events are handled in static buttons per class.

Action Listeners for each button are accessed separately, per the class that requires them. It is each classes duty to check for their own errors as this was created just as a design tool.

* Risks:
  + Logo cannot be found, displays white
  + Current navigation controller is null - doesn’t know where to put the next page, nor retrieve the previous

3) Navigation Controller -

* Functionality:
  + Checks previous page. If no previous page found, set the current page to show and hide the back button.
  + Checks the frame title. If the title is null, display blank text.
  + Each new subpage must carry over the existing reference to the navigation controller.
  + If a view does not want to be added to the stack, we use replaceView() instead of displayView().
* Risks:
  + No logo found - Displays white
  + No title found - Displays empty string
  + Current navigation controller not set - stays on current page
  + Stack is empty for previous pages - back button is hidden
  + User clicks side menu button - stack is emptied and resets to initial view

#### Quiz Page -

Take a quiz from created test in multiple choice format.

* Make sure that a quiz has questions before allowing the user to take the quiz.
* The user must select a quiz before being able to select take quiz.
* Build frame with correct number of buttons: the first and last have 2 and everything inbetween has 3.
* Ensure that the page index counter is properly incremented upon clicking next and decremented upon clicking back.
* Ensure that selected answers are saved when backtracking through the quiz.
* Ensure that the user selects an answer to each question before being able to submit the quiz.

#### Flash Card Page -

Practice test questions by being able to look at the answers to the questions.

* Ensure that a quiz has questions before allowing the user to take the set.
* The user must select a quiz before being able to select take set.
* Ensure that the page index counter is properly incremented upon clicking next and decremented upon clicking back.
* Reset the the show/hide checkbox after displaying the answer pop-up.

Results Page -

Displays the results of the quiz.

* Ensure that each question has a valid answer.
* Ensure results are calculated only for the total number of questions in the quiz.

#### Table Data -

As this page is an extension of CustomPage, it has the same basic implementations thereof.

* Display:
  + Displays a basic table, starting from the center of the panel, branching out as more courses/tests/questions are added.
  + Uses strings as names instead of object types.
  + Each row is matched with an id grabbed from LibraryController
* Functionality:
  + When a radio button is selected, the id of the row is saved (and output to the screen).
  + Button count is set prior as a static variable as the table can contain certain buttons per its use.
* Risks:
  + If an ID cannot be found, a NullPointerException is thrown when trying to manage or delete a question
  + If a row is not selected, the class must inform the user that they must select a row in order to continue
  + If a test cannot be found, the cell can appear blank, and the id will be null.
  + If no rows were found, we must alert the user to add one.

### Analytics Testing

In the current Analytics format, there are three main functions that require testing:

#### Logging an Event -

* Accepts a tag for a user-defined event, counting the number of times it is used
* Stores that tag in a vector during program execution
  + If it is a new tag, creates new slot in the vector
  + If it is an existing tag, increments the number of times used.

Risks -

* If vector accepts too many tags, may become bulky and waste memory space/slow down execution

#### Saving Events to a File -

* Sorts through logging vectors to prepare data for writing
* Saves data for each day to a .txt file
  + If a file for that day does not yet exist, create a new file to hold information
  + If a file for that day already exists, append to that file

Risks-

* If filepath is not created and managed properly, may create null exceptions and crash programs
* If directories are not handled properly, a user directory might become flooded with .txt files
* If dated file names are not handled properly, may lead to improper incrementation of statistics (eg. November 29 2015 combining with 2016)

#### Creating a Pie Chart (With Legend) -

* Retrieve data from a text file on a particular day
  + Throws exception if the file does not exist
  + If a file does exist then stores the data onto two vectors one for type of event and the other for how many times that certain event has been accessed.
* Create a Pie Chart
  + Use data from vector and assign a corresponding color
  + Override the JComponent to create a circle with different sizes of of slice depending on the event’s visit count displaying with their corresponding color
  + Displays in a single frame
* Create Legend
  + Uses the same data when creating a Pie Chart
  + Overrides the JPanel to display a list of events with their corresponding color and count.
  + Displays in another single frame, separate from the Pie Chart frame.

Risks -

* There should be a pre existing text file in order for the Pie Chart and the Legend to be created otherwise an exception is thrown.
* If the management of the files are not in the correct file path, the AnaPieChart class might access a text file that does not have the respective form or information to the project
* If the format of the text file is different than the intended one, not using the correct delimiter as the loadData() function uses a delimiter (“-”) to store the first section of the string into the visit type Vector and the second half goes to the visit count.
  + Information may lead to change to the display of the Pie Chart or even an error due to Strings not matching to other “Strings” or Integers with other “Integers”
* if the text file is null then both frames would display white.