John Loranger

April 7, 2015

CIS 234A

Assignment 1: Project Intro

# Select Stories

## N-Item Sort: Admin Set-up

## N-Item Sort: User Taking Test

## Ordering Comparisons

## Back Button

# Clarifying Questions

## Story – Foundation - N-Item Sort – Admin Set-Up

### Paragraph 1 statement: “no items currently exist in the system”.

#### What is the maximum number of items?

#### For Items, what is the min/max character count?

#### If the user enters a word phrase that is too long for a given line, should we wrap to a next line, or scroll the line horizontally?

#### If the user enters a word of phrase that is too long for a given line and we are wrapping to next line, do we break on spell-check determined whole words?

#### If the user is entering a word or phrase, should it be spell and/or grammar checked?

#### Should we strip off item-preceding, item-succeeding and excessive inter-word whitespace? Example: “ The for ran ” becomes “The fox ran” ?

#### Can the admin enter duplicate items?

#### Can the admin edit or remove items they just entered?

### Paragraph 2 statement: “some items already exist in system”.

#### Can Admin edit existing items?

#### Can Admin selectively remove existing items?

#### Can Admin clear (remove all) existing items?

#### Should system accept, but ignore duplicates to existing or new-this-session entries?

#### Should system accept, and add-in duplicates (same items appears multiple times)?

#### Paragraph 3 statement: “fewer than 2 items have been entered”

#### Should the Finish Button become disabled while/after user enters third item, and re-enabled once fourth item is entered?

#### Should the Finish Button be enabled whenever an even number of items are entered but disabled when an odd number is entered?

#### Should we have a tooltip on Finish Button or elsewhere indicating 2 items or more must be entered for the Finish Button to enable?

#### Should we have a note on the entry screen or some graphical grouping to assist user in knowing they must enter an even number of items to enable Finish Button??

#### Paragraph 4 Statement: “at least 2 items have been entered”

#### See Paragraph 3 questions.

#### Story - Foundation - N-Item Sort - User Taking Test

#### Paragraph 1 statement: “some number of items configured”

#### How is the number of items configured determined?

#### If the test terminates early, should answers thus far be saved or discarded? If we do save, should we preserve the fact that the results are from a partial test?

#### If we save an partial test, should those partial results be included in any statistics/reports?

#### Should we differentiate partial tests based differentiator of user hitting ‘Cancel’ vs user simply not finishing due to session interrupt (such as network failure or session timeout)?

#### If the test terminates early, can the user resume, perhaps by supplying test-session ID and/or via non-expired browser session?

#### How should test-session-ID be generated and what format, properties and size should it have? Discussion points include use of a sequencer: sequencer id characters, character count and initial value. Additional id discussion includes use of: date/time encoding, person’s name (security implications) or other id.

#### Will the test-session-ID appear on any user screens, admin screens or reports?

#### If the application is ever re-installed, should there be a way to configure a sequence-based test-session ID to preserve the existing sequence of ID’s. For example say the last user test had ID TEST151, we re-install the software, should there be a way to configure the system so the next test gets TEST152 instead of TEST001?

#### Paragraph 2 statement: “a user taking the test, with a unique test-session.”

#### When we store the Item & choice result, should we also store associated the test id, and sequence within test, with it?

#### If duplicate items allowed, do we need any special handling or data collection of user answers? Example: Some sort of marker indicating user answered differently given same two choices at different times in a test session.

#### Paragraph 3 statement: “a user taking test”.. “and done w/questions”

#### Should we show user a running completed/togo tally of questions while taking test.

#### Should we warn the user that ‘You have one questions left’ prior to last question.

#### Once user has completed all test questions, do they have the option to cancel and throw away results?

#### What summary information should we show user after taking test?

#### Story – Ordering Comparisons

#### Paragraph 1 statement: “some comparisons have been made already”

#### Must every item be present in every test?

#### How is the number of questions on a test determined, and is there any relation to the number of items available?

#### Should the comparison pairs be the same for each test?

#### Should the sequence of comparison pairs be the same for each test?

#### Should we include only unique items in the set of comparison pairs before introducing duplicate items?

#### THEN CLAUSE: Weighting:

#### Does this mean that the system will never choose B, D, and only choose from A, C, and E?

#### THEN CLAUSE: This weighting creates a sequential presentation pattern where the user will see unique item comparisons early in the test, and repeat item comparisons later. Is this ok, or should we investigate alternative sequencing methods?

#### Given set {A, B,C ,D ,E }: items B&D have each been compared 6 times, A&C 3 times, and E once? Do we extract comparison pairs randomly, or is there a weighting based on view count sets

#### Paragraph 2 ‘THEN the system will select as the first item’

#### This introduces a presentation pattern where the user will be shown a set of unique comparisons early in the test, then later more duplicates. Is this bias ok? A way around this would be to determine the set of comparison pairs based on this formula, and then randomize the pair's presentation sequence.

#### Story – Back Button

#### Paragraph 1 statement: “a user in the middle of a test”

#### Should the on-screen prompts specifically direct the user with detailed messages such as “I see you’ve entered data. You can change if you like, then hit Next to Save”? Or, should we keep prompts minimal and assume user can figure out Back/Next saving through observation?

#### Paragraph 3 statement: “AND clicks ‘Back Button’ again”

#### Should the user be able to ‘go back’ as many times as they wish?

#### 

# User Interface Sketches

## N-Item Sort: Admin Set-up

## Initial State. Showing screen initial no-data state, which shows status message indicating user must enter at least two items.

## Adding items. User types phrase into lower box and clicks Add button. Note that Enter Two items message is gone and Finish enabled.

## Opening with items already in database. Existing items are grayed out.

## 

## N-Item Sort: User Taking Test.

## Taking Test - Compare Screen. Use clicks one of three radio choice buttons then (as applicable) Next to move forward within test or Back to revisit answers. Choosing a radio button enables Next button. Hitting Back or Next saves choice.

## Taking Test. Right after user selected Radio Button.

## On the last item, the user will have Next replaced with Submit Test.

## Taking Test - Completion Screen. User received message affirming completion.

## Ordering Comparisons

## Ordering Comparison - Taking Test with “Apples, Oranges, Bananas, Plums. First comparison is randomly Compare Apples to Oranges.

## Ordering Comparison - Taking Test with “Apples, Oranges, Bananas. Apples and Oranges already seen, so now compare Bananas to Plums.

## Ordering Comparison - Taking test with “Apples, Oranges, Bananas. All items already compared once, so re-use Apples, and compare to Bananas, an item Apples not previously compared to.

## Ordering Comparison - Taking test with “Apples, Oranges, Bananas. Apples and Bananas already used twice, Oranges and Plums only once, so compare using Oranges and Plums.

## Back Button

## Back Button - When user is on item 1 of the test, the back and buttons are disabled, the radio buttons are initially unchecked and the user sees a Choose message. They must click one of three buttons.

## Back Button - Once the user has selected a radio button, the Next button is Enabled. Since this is the first item, Back is still disabled. The user can either change his radio button selection or click Next to for next item.

## Back Button - Now the user just clicked Next, and we are on Comparison #2 of 4. The user at this point could either hit Back, or select a Radio button then hit either Back or Next. If the User hits Back right now, then no answer will be saved for #2. If the user selects a radio button then hits Back or Next, then an answer will be saved for #2.

## Back Button - The user just hit Back on the previous screen. They see the answer they chose earlier. They may change this answer if they choose, and hit Next to save. They may also just hit Next and leave their answer unchanged. Back is disabled because this is item#1. If it were any other then Back would be enable.

## Back Button - The user just clicked Next. Now they are back on Comparison #2, and they have a choice of either going back to Comparison #1, changing Comparison #2 and hitting Next (if they had selected anything before hitting Back), or leaving Comparison #2 unchanged and hitting Next. In this case the user had not chosen anything on Comparison #2 before hitting Back, so Next is disabled until they choose.

# Implementation Notes

## Story N-Item Sort: Admin Set-up.

## Will need to secure Admin Web page or local GUI screen.

## Need to determine how to create/maintain admin userid’s

## Display of existing items may be linear list, but should explore more dense formats such as tables in case comparison suite to large.

## If Comparator is to grow to support pictures etc, then single column display with ‘detail view/edit’ screen may work. Along these lines, it is likely that Item will be an abstract Java class with specialized children for String, Picture, Song specializations as possibilities.

## Try to determine if an item ID is necessary to maintain and if other item enrichment data, such as presentation Order is going to be needed now or in the future. ID will likely be based on client input, and possibly database id fields. Make sure database determined ID Fields can translate to other databases.

## Determine if Admin needs any delete functionality.

## Determine if Admin needs ability to save ‘working sets’ of items rather than just having one set..

## Potential Classes initially identified:

## Item

## List of Item

## Data structure to present list of items.

## Admin Login Object with security features for Admin user.

## Potential Database Tables.

## UserId Table with columns userid, password, role.

## Item Table with columns for item\_name, possible item\_Id and order\_or\_entry.

## Story N-Item Sort: User Taking Test.

## Potential Classes Identified:

## User Login object with Name, email fields

## Item object to contain comparison item data. May be as simple as a String, may be an abstract Data item which could specialize to ntring name, picture, or audio file. Will use Java Comparator object to compare items.

## Answer object, which contains Java enums of LEFT, CANT\_DECIDE, or RIGHT to save user’s answer.

## ComparisonPair which contains 2 Items.

## TestQuestion which contains Comparison Pair, Answer, and possibly a display sequence scalar.

## ScreenSequence - possible sequence scalar class to hold value indicating TestQuestion sequence in TestQuestionList. (future-tbd) May be tied to Item table and may even have an Admin entry/edit to help Admin specify sequence of presentation.

## TestQuestionList which will likely be data structure containing sequence of TestQuestions for Test. Data structure will possibly be a doubly linked list or indexed array.

## TestId which will contain a unique Test Id.

## TestResults – tbd – possibly a data structure showing summarized and/or derived test results. To be used in end screen reporting and may tie in with results analysis screens yet to be determined.

## ComparisonTest which will contain User Login, TestId, Login, and TestQuestionList.

## GUITestScreen – GUI view to display questions to user.

## GUIResultsScreen – GUI view to dislay results.

## Potential Database Tables:

## ComparisonTest table to store test data. It may have links to Questions Table, UserId Table. Not sure if the data structure for this needs to be backed by database yet – probably not.

## Userid table (already mentioned)

## Item Table (already mentioned).

## Story N-Item Sort: Ordering Comparisons.

## ItemSet object which will contain a set an Items.

## ItemToItemSetMap object which will contain a Map of Item to ItemSet. Each item will have an associated map of items, possibly indexed by comparison count, that will be used in determining the set of candidate items for comparison.

## Will possibly use java util Set, Map, Comparator.

## Potential Database Tables:

## May choose to recreate these sets & maps from data stored in ComparisonTest table.

## Need for database storage for these sets/maps is tbd. It may be easy enough extract info from ComparisonTest table or we may need a more involved table that actually stores these sets/maps. Need for recreation of these data structures will also be determined by need to store in-progress tests in case user ‘drops off’ during a test and wants to re-attach and continue.

## Story N-Item Sort: Back Button.

## As mentioned in User Taking Test, Data structure within ComparisonTest supporting forward/back and test. May want to have ComparisonTest maintain list of helpful messages (or even state objects) that will change as user moves Back/Next over list of questions.

## Database Tables: None identified, with possible exception of database needing to store ‘test state’. See Ordering Comparisons database notes.

## Techology Architecture, initial thoughts & questions.

## Will use MYSQL for database with stored procedures for all data layer access. Stored Procedures will support exception throwing up to Java database access layer.

## Will use Java layer for database access layer.

## Will use Java layer for business object layer.

## Will use Java layer for GUI Services. These provide a service interface for Javascript/HTML clients to call.

## (initial thought - tbd) Will use HTML/Javascript for presentation layer. Presentation layer will handle all user interaction and interact exclusively with Java GUI Services layer.

## (with instructor permission) May use Bootstrap HTML libraries to assist in developing GUI.

## Architecture influencing questions:

## Does user want a web or desktop GUI based application?

## How distributed does the app need to be? Is it one-terminal in-office, or should it be a 24/7 access server with web clients?

## If app is to be truly web distributed, should we consider J2EE?

## Platforms and Software:

## IntelliJ for development environment for HTML, Javascript, Java development.

## MYSQL database. Environment tbd. Possibly simple command line script environment.

## Git for revision control.

## (tbd) Maven for build tool.

## (tbd) JUnit/Hamcrest for Java testing, tbd for GUI and database.

## Developer workstations: Per developer choice of Mac or Windows.

## Documentation tools: Microsoft Word, Pencil or Visio for GUI mockups (tbd), UML tool modeling tool is TBD.