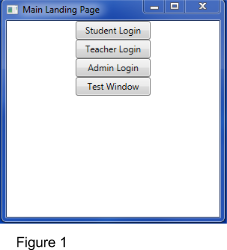
Austin Blackman

11/6/14

Senior Design

Component and Interface Design

The Collaborative Student Lab Manual exists as a locally installed Windows WPF application which has the ability to run on many types of Windows devices. The Collaborative Student Lab Manual (which will be referred to as CSLM for the remainder of the document) is implemented using a simple and intuitive series of WPf windows which manage their displays based on accessibility and use cases.

The initial window that will be presented to all users upon application launch is the Main Landing Page (Figure 1). This landing page will redirect the specific user to their user specific logon screens and subsequent landing pages. While the main landing page logic is simple and largely a series of re-direct statements, there is considerable logic taking place once a specific user successfully logs in.

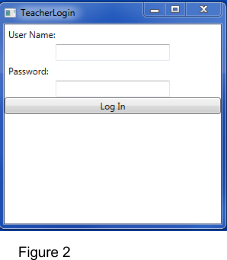
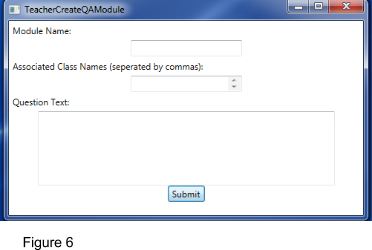
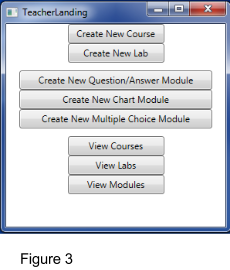
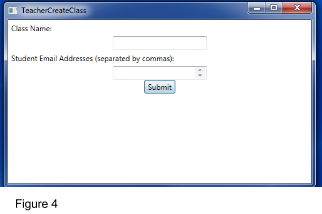
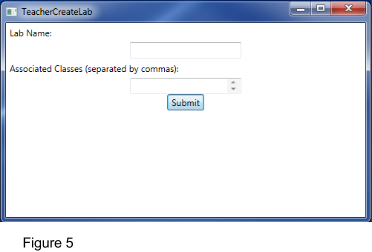
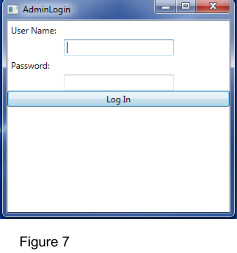
The first user case described is that of a Teacher user. Noted in Figure 2, the Teacher user is logged on using User name and Password fields, which will be discussed further when speaking about Admin roles. There is a key aspect taking place here where the static “user object” is being assigned based on which type of user logs in. The static user object works by containing a field of each type of user, and since each of these users implements a unique getType method, the static user object deciphers which field to actually fill with the user. This important tracking logic takes place on logon rather than clicking the Main Landing Page because the program keeps track of the specific User object rather than just which type of user. This enables the program to autofill certain attributes of new material being created. 

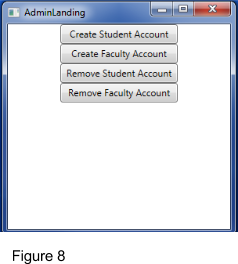
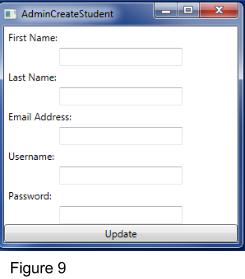
Figure 3 depicts the Teacher Landing Page. When a teacher log on successfully (with valid credentials), their login Window closes and the Teacher Landing Page is brought to the front of the screen. The landing page consists of all actions available to a Teacher user. The teacher can create a new course, create a new lab, create one of three different types of modules, and then view/remove any of these classes, labs, and modules which they created. Specifics regarding the implementation of these actions are described below, however it is relevant to note at this point that all of these actions are in a sense bound to the static user object described previously. What that means is that any item that the teacher creates will contain primary key information for the teacher who created it. In the case of CSLM, email addresses are used as a primary search key (although the database itself also provides unique identifiers for every entry in the entire collection. For example, Figure 4 represents the first option available to Teachers, which is the creation of a new course. A course exists in CSLM as merely an association, or what would be described as a dedicated join table in a relational database. A teacher builds a new course by inputting a class name which is forced to be unique as this will serve as a unique identifier in the future. In addition, the teacher will input the email addresses of all students who the teacher wants to be “enrolled” in the course. By becoming enrolled, the student objects in the database will be updated with any additions to the material of the course (this will be described in the module creation). When the teacher presses the submit button, a new course is created with the course name, the email address of the creating teacher (from the static user object described earlier), as well as an enumeration of all student email addresses enrolled. In addition, each student document (the data set corresponding to a student object in the database) who is listed as enrolled in this course will have the course name (a primary key) added to their list of courses. 

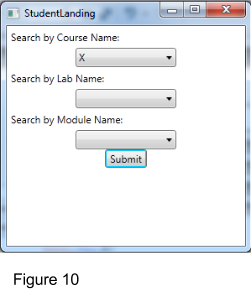
An example of Module creation is depicted in Figure 6. This represents a “QAModule” which is short for Question-Answer module. The creation of modules and their relations to the different users of the application is essential to the nature of CSLM. When a Teacher user creates a module, they input the Module name (again, this is forced unique as it will user to reference the entire module document), any associated courses (this module can be used for numerous courses across a teacher’s workload, as well as the “Question” part of question-answer pair. The only part that is different between the creation of QAModule verus the creation of Chart Module or a Multiple Choice module is the last piece where the QAModule has an input for question text. When the Teacher user presses submit, a Module document is written to the database directly modelled after the Document datatype in the application.

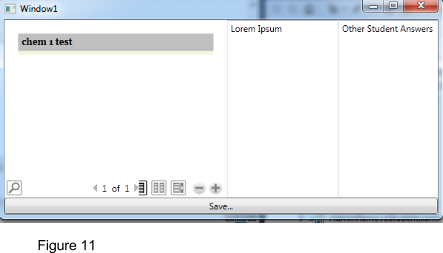
This direct mapping facilitates a unique advantage of using C# and Mongo together. CSLM utilizes C#’s integrated LINQ library which facilitates a large range of database interactions. By building documents for storage in the database that directly map to datatypes in the application (same field names, same expected field types), the application can make direct casts and treat generically typed products of a query as an instance of the datatype that they map to. Direct object mapping is best explained in the case of a Module, but is also used to map specific users when queried. Direct object mapping and use of the LINQ library are side topics, but a relevant interjection when speaking to the input of a Module in the database.

A series of complex queries are run when a new Module is submitted. First off, the new Module exists as an independent document entry in the database. This will serve as a “reference standard” version of the document that can be compared to if changes are made. Additionally, local copies of this module are provided for each student who is enrolled in one of the courses that the teacher has submitted to contain the module. After this query is complete, all student document entries that are deemed “relevant” (a term used in the coding for the application to describe a student that requires a copy of the document entry) contain an instance of the module that was created. Furthermore, a teacher can edit the reference copy which is stored in its own document and upon submission of the edit, all student copies will be edited as well. Also important to note is that the module itself is created with the answer fields left blank. On a QAModule this means an empty string (for the student to fill out on their own), but on a Chart Module this is a 2d array and on a Multiple Choice Module this is an enumerable of invalid answers (-1 in this case). 

Lastly for the teacher, the creation of a Lab is somewhat like the creation of a course. The lab itself has a forced primary name, as well as a one to many association to courses, and a one to many association with modules, all my unique name. This has little impact on how things in the backend work, but due to direct object mapping, the application is able to create a Lab object out of these entries to display information to a student in a sorted and organized way.

The admin role is miniscule in comparison to the teacher and student roles, however touching upon this user’s creation of different users is beneficial to understanding the nature of users in the application. The admin login page (Figure 7) checks Admin User credentials just like the Student and Teacher users. While the static user object keeps track of the admin user that is logged in for the session, there is nothing else to the admin user besides a user name and a password. The landing page for the Admin User (Figure 8) shows all of the actions available to the administrator. The administrator can add and remove both student and teacher (faculty) users. It is important to note that these are the only actions available to the administrator. It is up to the teacher to manager their own content. Students and teachers are removed by their email address, but Figure 9 shows how users are added. Figure 9 itself shows the creation of a student user, however the creation of a teacher user requires the same information. These are considered to be the initialized values on creation. They are not however the only values contained by the user object (as has been described earlier. What happens is that all other fields pertaining to the user whether they are courses enrolled in and modules for students, or courses taught for teachers are filled in as empty sets of the appropriate type. This enables direct object mapping for both new users as well as users that have information associated with them.

The student user is the last user that needs to be touched on. Again, the QAModule will serve as an example of how students work through material provided by the Teacher. Students log into the system using the same Username and Password screen as the Teachers and Administrators. Upon successful logon, the static user object contains the exact student user who is currently logged in. The Student Landing Page (Figure 10) allows students to search for work either by Course, Lab, or directly by Module. Searching by Course or by Lab will bring the Student to subsequent landing pages to narrow down which lab or which module in the lab the student wants to complete. The module completion page is shown in Figure 11.

As Figure 11 shows, the question portion of the QAModule is highlighted in grey on the left. This is the piece that the Teacher had input when they built the Module at the beginning of it’s lifecycle. The First textbox if for the Student user to input their own answer. On the end, another textbox shows the answers provided by other students. The algorithm for providing these answers is simple. The the page is loaded, a query is run to find the answer field for each student’s Modules document where the Module’s name is the same as the module that the current student (static user object) is working on). A selection statement discards all entries that are blank (meaning that the given student has not filled out that particular module yet). 

The format schema for the module completion windows is persistent across all types of modules. The teacher provided prompt appears to the left. The student completes their own work in the middle and on the right is a display of other student’s work.

The chart module is the most helpful as far as displaying other student’s work is concerned. The chart module will not display each student’s exact answers. Instead, it will produce an increasingly complete average of the what a student can expect to get for each attempt. For example, if the instructions are for a student to record ten datapoints three different times and then average the datapoints across the three attempts, then the student would see ten averages to work with. In simpler terms, there will be as many averages displayed as there are rows in the table constructed.

The static user object is persistent until another user logs in off of the main landing page. This is designed to allow one student to complete some work, and then another student (presumably in another session of the same course) can log in off of the main landing page without so much as closing the application.