**ABSOLUTELY ANONYMOUS**

Improving student, lecturer communication

By

### ZAYNE FRAUENDORF, October 2016

2016

2016

October

August

Contents

[Context diagram 3](#_Toc459971675)

[Brief description of the system 3](#_Toc459971676)

[Timeline 4](#_Toc459971677)

[Description and priveleges of roles 4](#_Toc459971678)

[Objectives and specifications of methods implemented 4](#_Toc459971679)

[Data structures userd 6](#_Toc459971680)

[UML CLass Diagram 7](#_Toc459971681)

[Use Case for the application 7](#_Toc459971682)

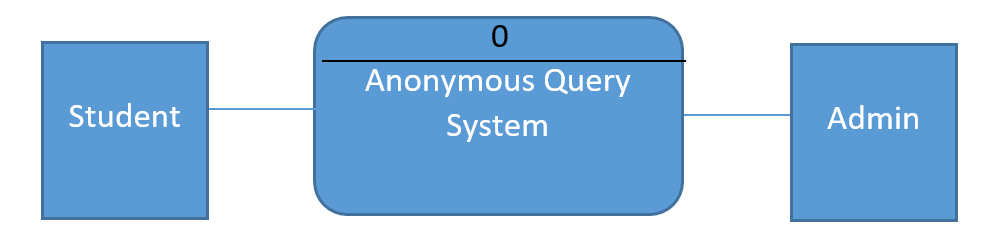
[Screen shots with test data 8](#_Toc459971683)

[FInal code for each class 9](#_Toc459971684)

[Integration test results 10](#_Toc459971685)

[References 11](#_Toc459971686)

# Roles and Responsibilities of our Team Members:



# Brief Description

Absolutely Anonymous is a system designed for the purpose of giving students the ability to contact their lecturer anonymously. Anonymous feedback is generally a little bit more honest than face to face interactions between a student and their lecturer and hence the need for this systems design.

With Absolutely Anonymous, a student can select their desired course, create queries which pertain to that course and those queries are readable by the administrator (Lecturer) who then can work to provide a solution. The student can also see a list of other queries which have been asked in anonymity which helps when ten students have the same question.

# Description of privileges and roles

|  |  |
| --- | --- |
| Privelages | Role |
| Add and view queries anonymously | User |
| Add users, courses + filter and delete queries | Admin |
|  |  |
|  |  |

User

The user is a role within the Absolutely Anonymous System which can access and see a list of queries made anonymously by other students from the “List” tab.

A user can add a new query from the “Query” tab. There they select the specified course which their query will belong to, enter their query and click the add request button. None of this functionality requires any identification from the user.

The user can see the about page

Admin

Upon entering their password - The administrator can perform additional functionality which includes filtering the courses to monitor which is done from the “Monitor tab”, this manipulates the list under the “Feedback” tab where an admin can also select an item from the list and choose to delete that item.

The administrator can add new administrators and add new courses under the “Admin” tab. This also requires re-entry of their administration accounts password.

The admin can see the about page

# Objectives and specification of methods implemented

|  |  |
| --- | --- |
| Method | Class |
| AddCourse(); | **AdminAddCourseWindowViewModel** |
| AddUser(); | **AdminAddUserWindowViewModel** |
| ShowAdminDialog(),ShowUserDialog(),IsLoggedIn(); | **AdminViewModel** |
| OnPropertyChanged(); | **BaseViewModel** |
| ConfirmDeletion(), GetFilteredQueries(); | **FeedbackViewModel** |
| ShowAbout(), ShowMonitor(), ShowQuuery(), ShowAdmin(), ShowFeedback(), ShowList(), CreateViewModel(), CreateModelDependecies(); | **MainWindowViewModel** |
| Apply(), ConfigureDefaultCourses(); | **MonitorViewModel** |
| AddQuery(); | **QueryViewModel** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Admin Add Course Window View Model

**void AddCourse();**

Method checks if the “Course” property on the same class is null, if so it creates a new course and assigns it into a variable called “newCourse” which is then passed as an argument to the AddCourse(); method on the CourseList class. Once passing the argument the method calls courseList.Save(); which is another method on the CourseList class. (Note: courseList being dealt with here is referring to the interface of the actual implementation of CourseList class)

Admin Add User Window View Model

**void AddUser();**

Method checks if the “Admin” property on the same class is null, if so it creates a new user and assigns it into a variable called “newAdmin” which is then passed as an argument to the AddAdmin(); method on the AdminList class. Once passing the argument the method calls adminList.Save(); which is another method on the AdminList class. (Note: adminList being dealt with here is referring to the interface of the actual implementation of AdminList class)

Admin View Model

**void ShowAdminDialog();**

Method creates new instance of an AdminAddCourseWindow, it then assigns that window to it’s view model and assigns the data context of the view to the viewmodel before calling the ShowDialog method on the view.

**void ShowUserDialog();**

Method creates new instance of an AdminAddUserWindow, it then assigns that window to it’s view model and assigns the data context of the view to the viewmodel before calling the ShowDialog method on the view.

**bool IsLoggedIn();**

Method assigns a method from the AdminList class adminList.GetAll(); to a local variable called allAdmins. Method then iterates over each admin in the local variable allAdmins and checks on each iteration if the password of the admin in the existing list of admins is the same as the Password property on the class (which is bound to the password textbox the user enters). On checking if true then IsLoggedIn(); returns true otherwise this Boolean method returns false.

Base View Model

**void OnPropertyChanged();**

Method implements a method which is on the INotifyPropertyChanged interface which is widely used on MVVM projects. Have implemented this method in the Base View Model class so that all the viewmodels can inherit the INotifyPropertyChanged interface through this Base View Model without having to implement the interface themselves.

Feed Back View Model

**MessageBoxResult ConfirmDeletion();**

Method prompts a message box to show, providing the user with a choice to continue deletion of an existing query or not, method then returns that result.

**GetFilteredQueries();**

Method takes in a list of strings from a parameter called courses which holds an argument that’s passed to it. Assigns a new list of type “Query” into a variable called result, then assigns a list of existing queries via the queryList.GetAll(); method which is defined in the “QueryList” class, it then iterates over each item in the string list passed into the method and for each item it adds a new item to the local variable result if the course name in the passed in list of strings matches to the existing course names. It then clears the filteredQueries variable and iterates over result adding each item in result to the filteredQueries collection.

Main Window View Model

**ShowAbout();**

Assigns the data context of the AboutUCView to the about view model and assigns the CurrentView user control property on the class to show the view.

**ShowMonitor();**

Assigns the data context of the MonitorUCView to the about view model and assigns the CurrentView user control property on the class to show the view.

**ShowQuery();**

Assigns the data context of the QueryUCView to the about view model and assigns the CurrentView user control property on the class to show the view.

**ShowAdmin();**

Assigns the data context of the AdminUCView to the about view model and assigns the CurrentView user control property on the class to show the view.

**ShowFeedback();**

Assigns the data context of the FeedbackUCView to the about view model and assigns the CurrentView user control property on the class to show the view.

**ShowList();**

Assigns the data context of the ListUCView to the about view model and assigns the CurrentView user control property on the class to show the view.

**CreateViewModels();**

Assigns all the new instances of view models to pre-defined variables in order to not new an instance up every time.

**CreateViewModelDependencies();**

Assigns instances of List class repositories to pre-defined variables in order not to new an instance up every time.

Monitor View Model

**Apply();**

Creates a new list of strings called “tempCourses” and instantiates it. Then creates a new variable called allAdmins and assigns that to a collection of items which is retrieved via the GetAll(); method on the Admin List class. Method then iterates over each admin in allAdmins which has got a list of all existing administrators assigned to it now and checks If their password matches to the Password property of the class which is being entered by the user as it’s bound to the textbox. Method then defines a courseArray variable which is assigned to a Split version of courses inside the Courses property which are split by the (,) comma. Method then iterates over each of the items inside the courseArray which now contains all the courses split into their own array item rather than one line of courses. For each of those split course items it defines and assigns a local variable of type KeyValuePair called “allCourses” which gets each of the keys and values of the courses which are currently existing. Method then iterates over each of these to check if the existing course once trimmed and lowered matches with the course the user enters and if so adds Course Name of the matching course to the temporary list of strings defined earlier in the same method. Lastly the Selected Course property on the class is assigned to the tempCourses local variable and the method returns.

**ConfigureDefaultCourses();**

Creates a variable of type key value pair and gets assigned by the GetAll() method inside the Course List class. The GetAll() method returns a list of courses which are in existence. This method then creates another local variable which is assigned by the course name of each course in existence. Method then creates another variable called defaultCourses of type string which is assigned to each of the course names joined by a comma. Method then assigns the property to defaultCourses which displays the multiple courses inside the textbox as that’s what the property is bound to.

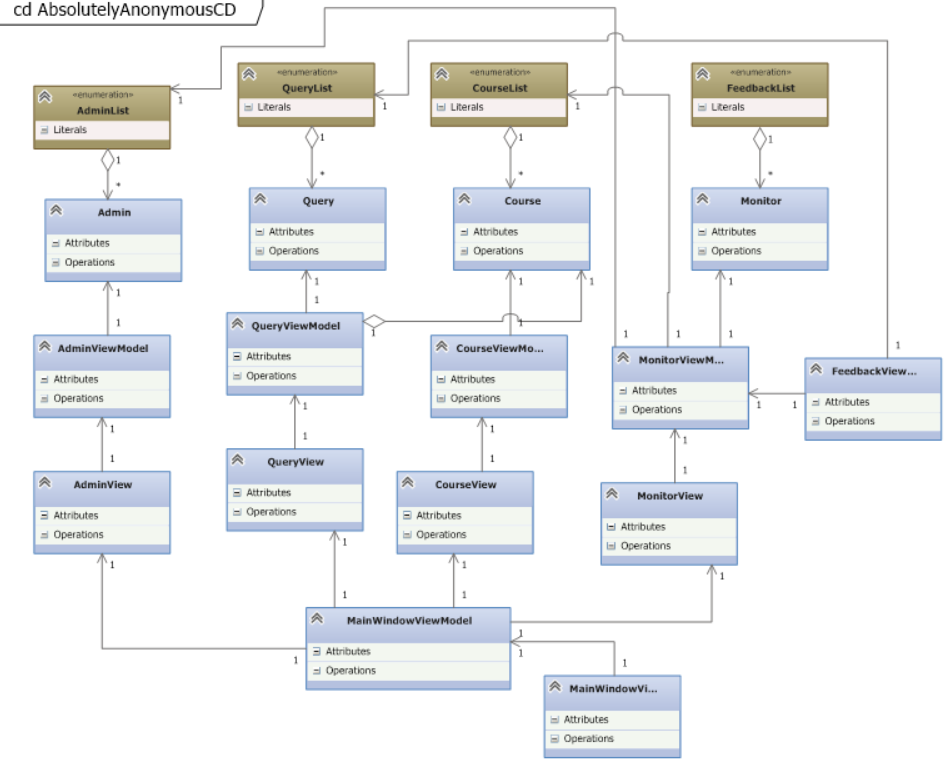
Query View Model

**AddQuery();**

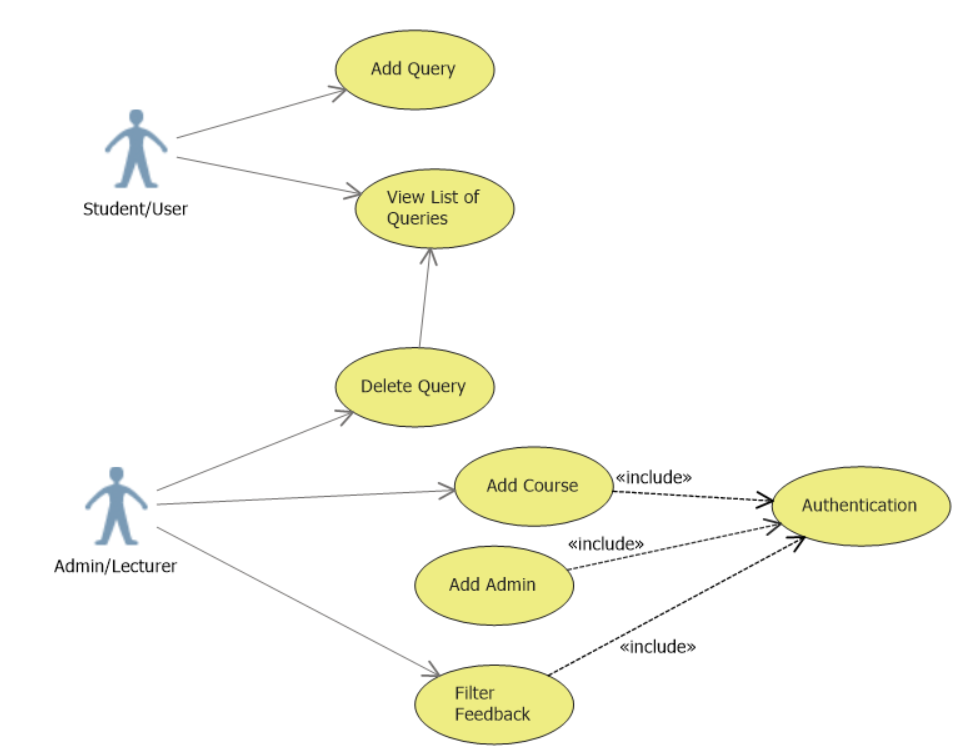
Checks if the values of the SelectedCourse and Query properties are not null, if so it creates a new query else it pops a message box showing the issue.

# 

# UML class diagram



# Use Cases:

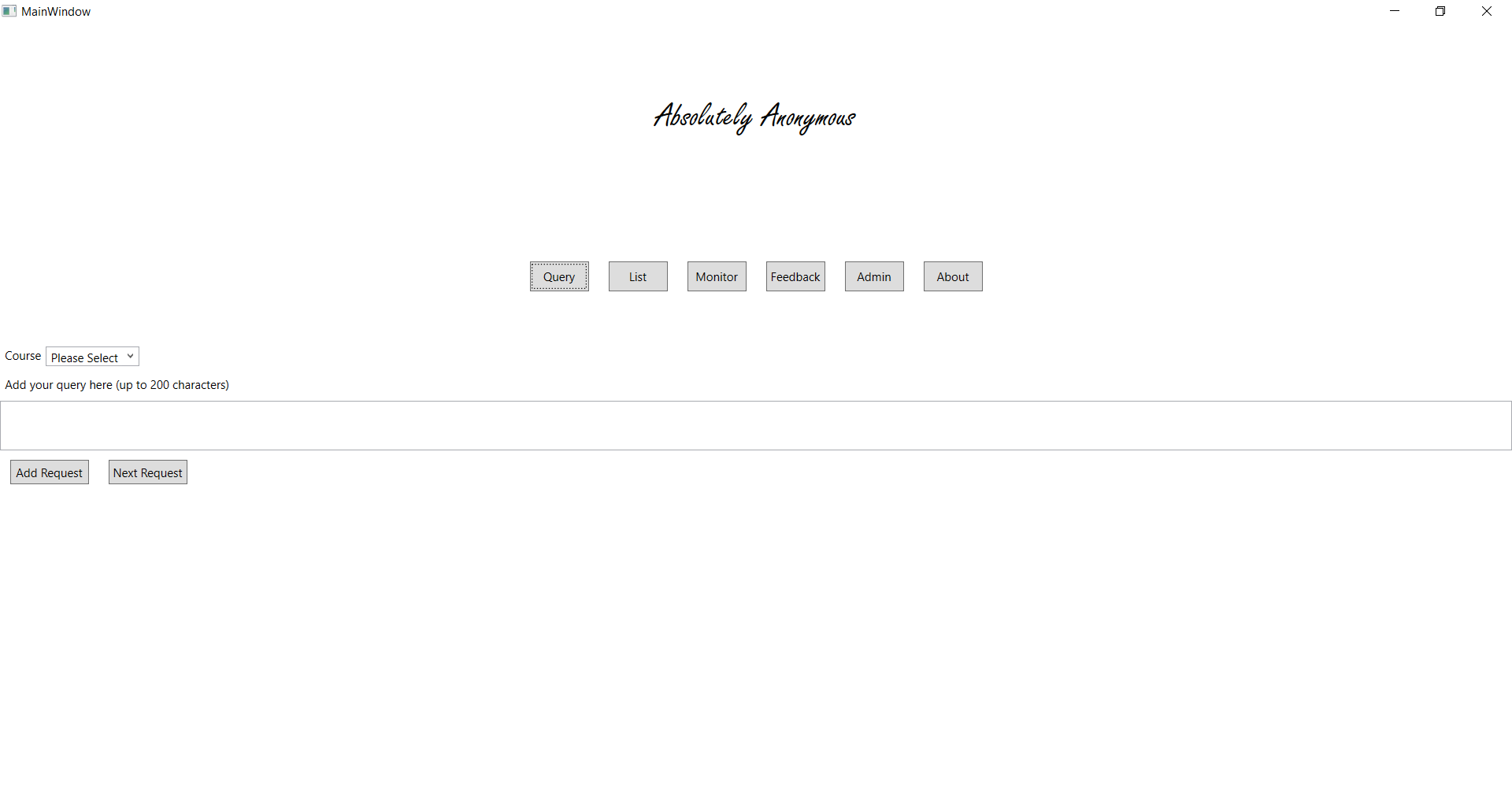


# Screen shots with test data

Absolutely Anonymous Screen Shots and Test Data

The Main Menu:

The Query Tab First Screen:



Selected course Engineering, and added the query “My Engineering query”

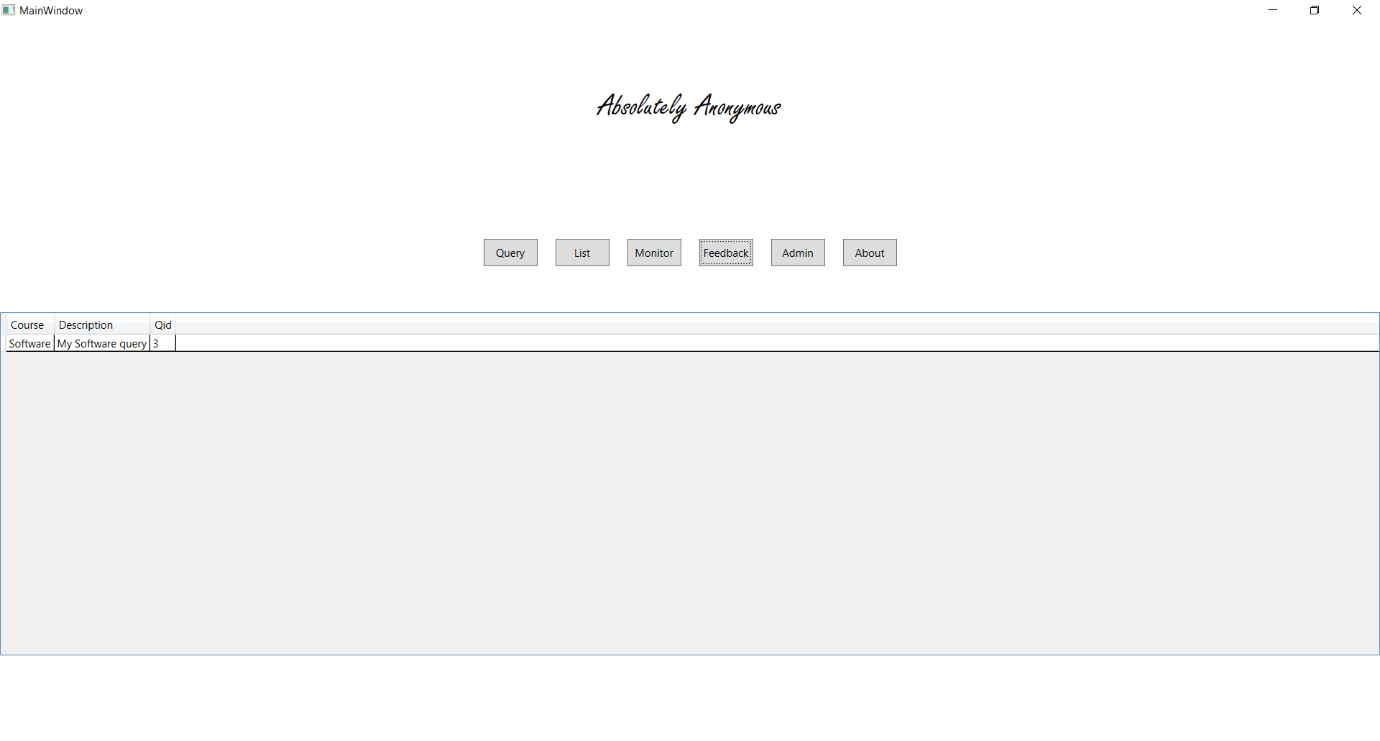
Query Added to the List of Queries under the “List” tab.

Entered password “hubrid” (Shown for display purposes) and left all courses as default in the textbox which displays every current course there is separated by a comma. Clicked Ok.

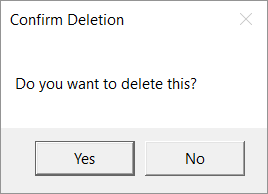
All current queries showing in feedback



Selected only “Software” in my courses box. Removed all other course names. Clicked “OK”.

Feedback now only shows the software query and has removed all the others from the current data grid.

Clicked on the item in the data grid and a message box has popped up

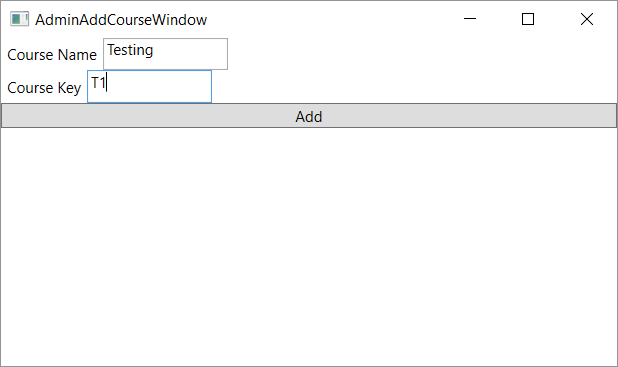


Clicked “No” and the query is still there

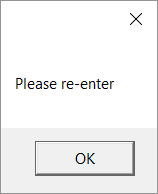


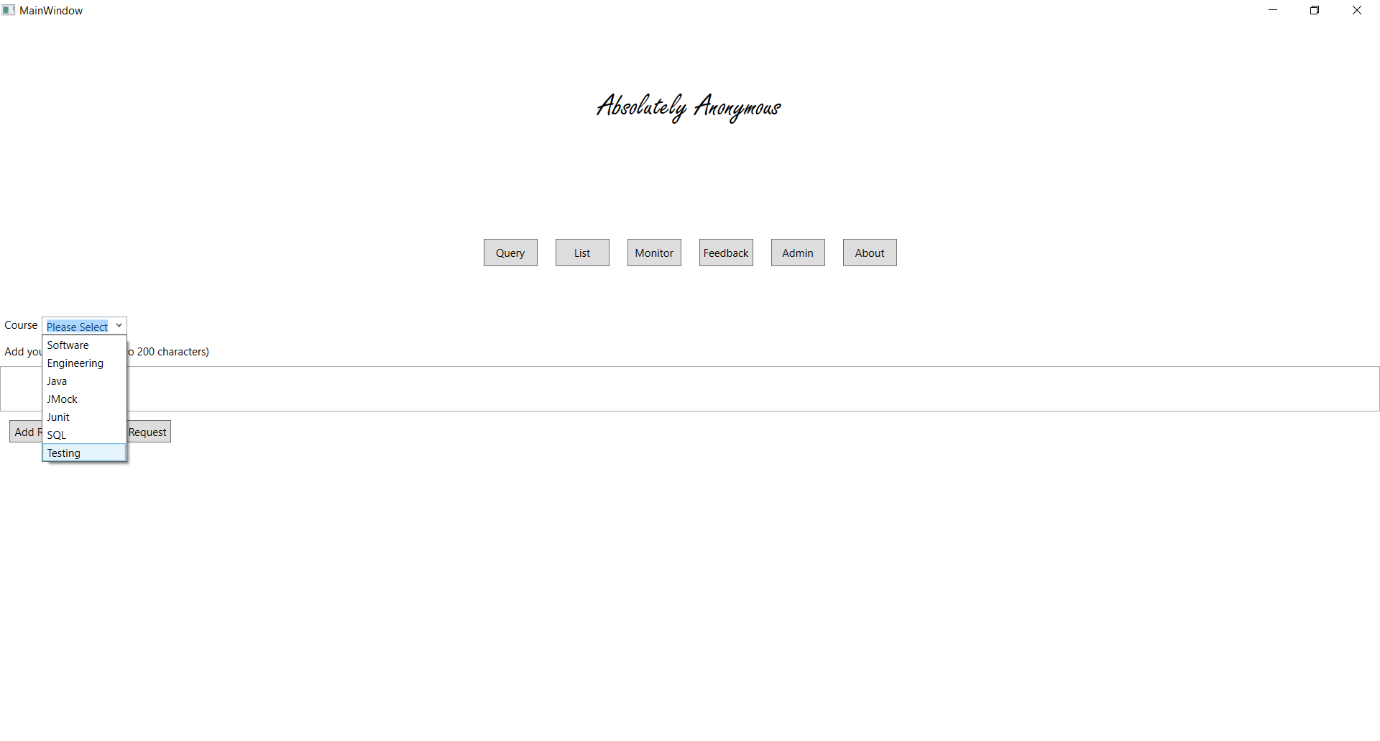
Clicked “Yes” and the query is gone



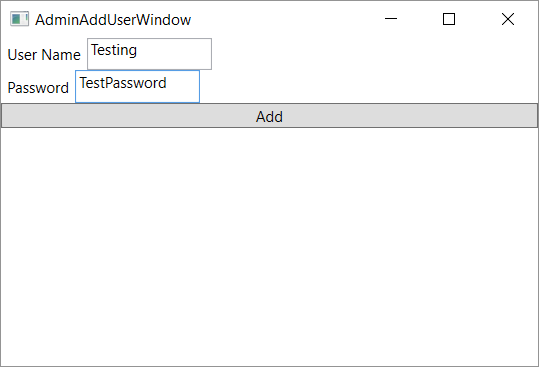
In the “Admin” tab, entered password “hubrid” (shown for display purposes) and selected Add Course which pops a new window added “Testing” in the name and course key as “T1”, then clicked Add.

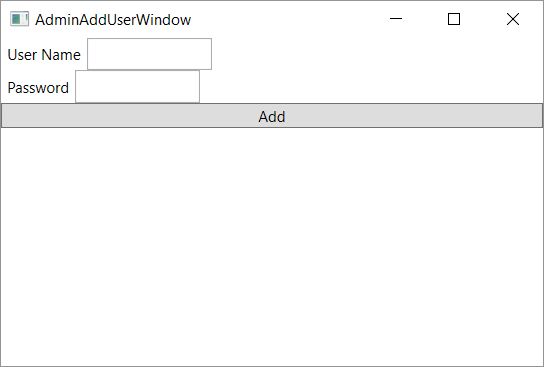
Entered incorrect password on purpose to ensure an error is displayed.

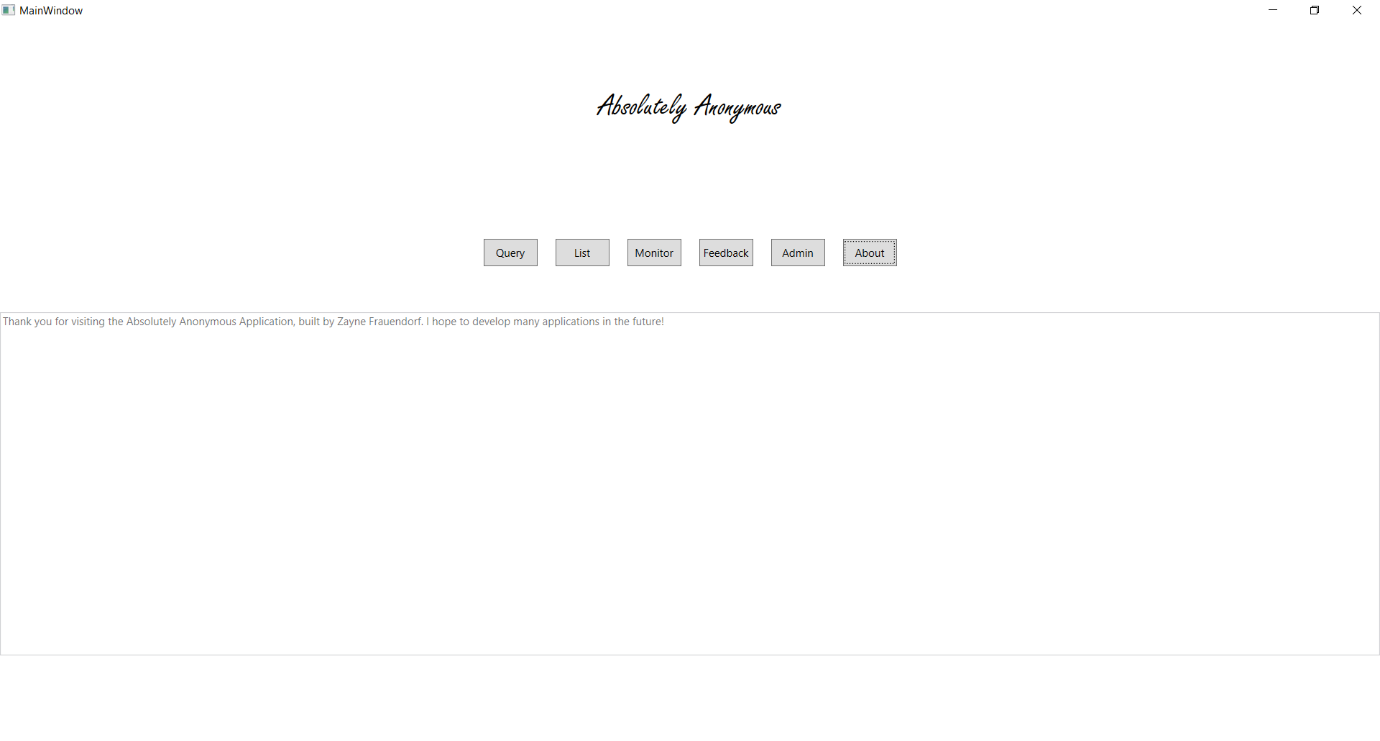


Proof of added course is that it’s now inside the drop down on the Query Window. Shown below.

In the “Admin” tab, entered password “hubrid” (shown for display purposes) and selected Add User which pops a new window added “Testing” as the username and “TestPassword” as the Password and clicked “Add”



Proof that it works, you can now access admin features using the “TestPassword” password.

Navigated to the “About” tab and the about message shows in the user control.