https://youtu.be/aGaclbJgdGE

Analysis

Problem Background

Since the pandemic forced schools to deliver their curriculums remotely, there has been a growing need for online educational resources, and it is this new demand that has inspired this project. There are currently various educational applications already on the market, Hegarty Maths and Quizlet to name a few, yet there is a distinct lack of fun and engaging software available for the humanities. The existing software available seems to have large gaps in its demographic and overall tone, some like Hegarty maths are highly educational and helpful but seem clinical and lack personality. Whilst, the well-known Kahoot! is certainly entertaining and engaging but is limited in its format of multiple choice questions and true or false, thereby restricting itself to answers that have clearly defined correct answers, which is not the case with most humanities. In fact, most humanities subjects seem to be limited in the software that they can use, either having to use ill-suited software or word processors to make their own resources. This 'deficit' was discussed with a small group of students, as the realisation that they lacked online materials to support their studies, was revealed. Interactivity is the main goal of this project, to have an application that explores not commonly considered subjects like geography in a new and entertaining way to appeal to students. Partially inspired by the Driving Theory 4 in 1 app that provides the usual question and answer learning format combined with a fun engaging game to learn road signs, this project will equally have both the conventional quiz for users to answer but also a fun game element to excite students. This multi-media c# project will ensure that a high-level of intractability is maintained throughout in order to cement students knowledge. Due to the purpose of the project, a fun and lighthearted design will be used, as well as clear and readable content, so that it is easily accessible and understandable.

Potential Client/User

The main group that will be consulted throughout this project, in addition to the project supervisor, will be students from both years 13 and 11. These people are the best group to ask, as they will be able to comment on the quality of the application and whether or not it appeals and engages the main demographic, which are pupils looking to revise, hence the suitability of asking exam year groups. Due to this, the program must be easily accessible to all students, regardless of their computer literacy, meaning that all processes must be simplified for the end users.

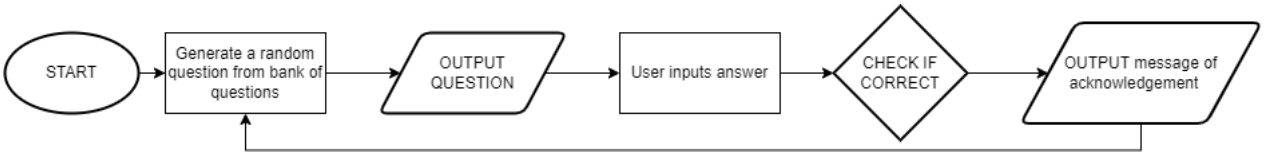
Research Methods

Researching the Problem

Due to the nature of this project, the opinions of students are extremely important, to ensure that it fulfils the purpose of being a useful but engaging resource for students to use. The findings from the initial interview (Appendix 1.1) seem to reiterate the need for an aesthetically pleasing, fun resource and yet useful application that actively encourages students to learn. It also raised the possibility of helping students to regulate their revision, to make their studying more productive.

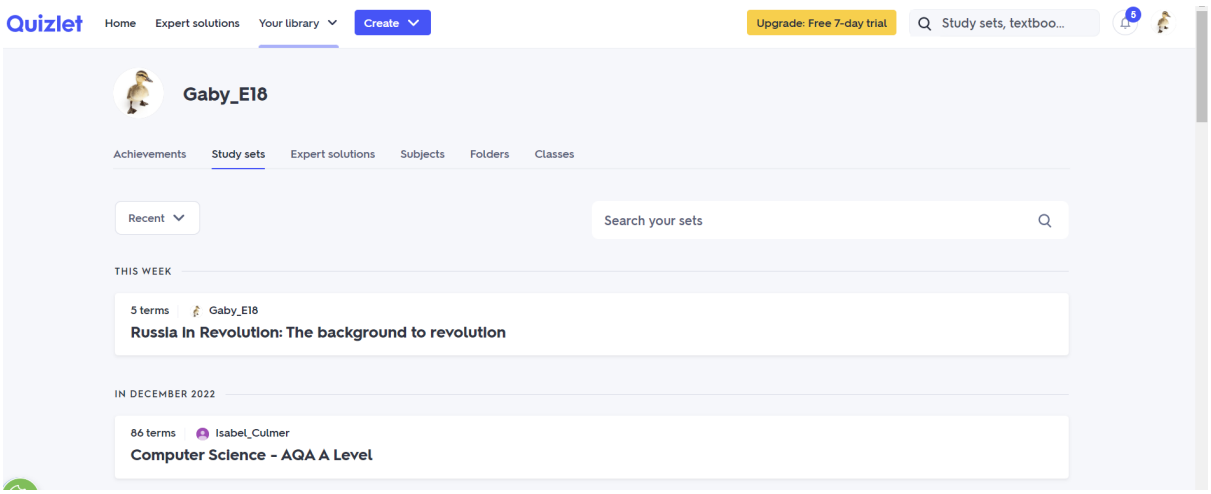
Building on the findings of the interview, a questionnaire (1.2) was produced, in order to get a greater range of responses, to supplement and support the general understanding of what students wanted from this software. The questionnaire seemed to echo the interview, that having an interactive system would be beneficial. Additionally, the idea that essay-based subjects and humanities were severely uncatered for, and that these features should be considered. However, it is clear that the questionnaire did not ask the correct questions, as the responses received were generalised and did not give information that was completely useful. In fact, the interview was much better at gathering information as it was possible to ask for elaboration.

The Current System

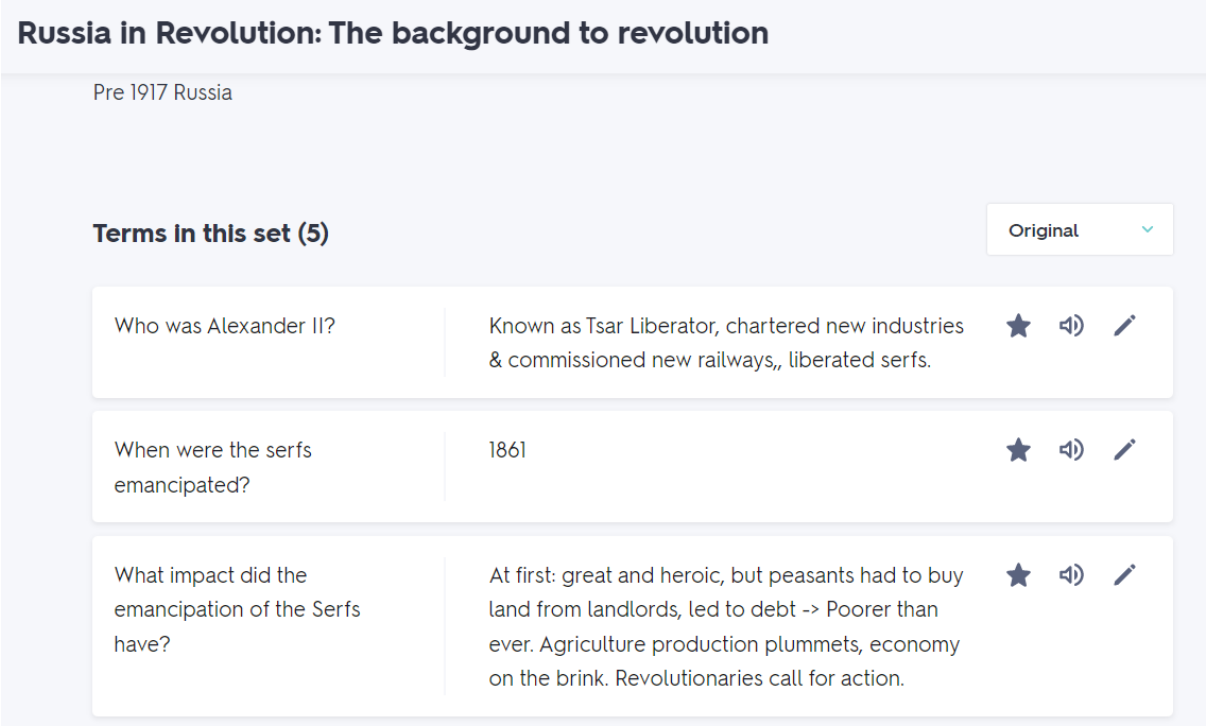


The current systems in place tend to have the above framework, which is having an existing bank of questions that are randomly selected and then the user's answers are validated. This generic algorithm explains why many systems have fallen into a bland and robotic style of revision, that does not engage or enthral. The current systems below, some of which have been referred to previously, demonstrate some features that may be considered, as they are proven to be useful for revision, equally, some add an element of excitement.

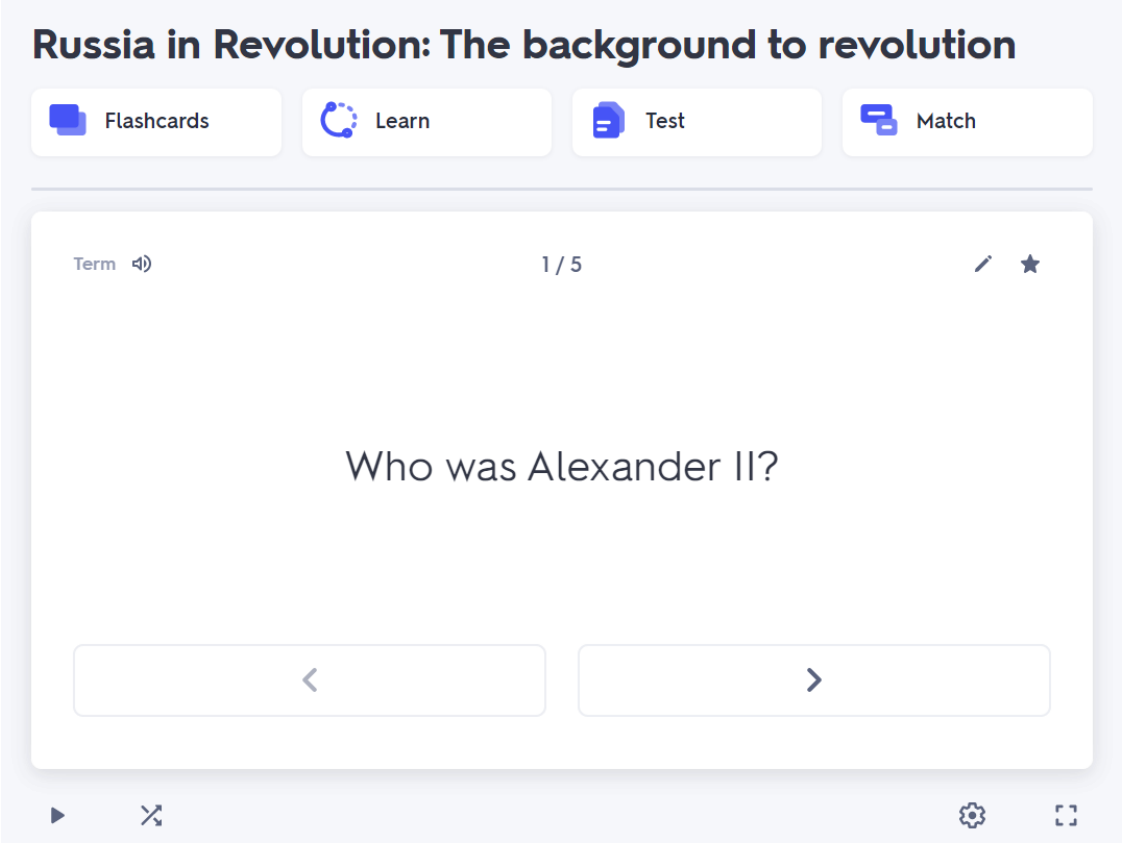
QUIZLET:
Arguably the most well known application from the list of existing systems, and one mentioned by many of the responses in the questionnaire. This system relies on a large online community that uploads sets, and allows users to contribute to this library of information, meaning that information may be inaccurate, but is bespoke and fulfils the exact needs of the users. Equally important is the way this information is delivered and learnt, with a variety of different modes, such as digital flashcards, a game similar to snap and conventional quizzes. This variety is something that this project will aspire to, as it is clear that this is what has made Quizlet so successful and retains the focus of its users.



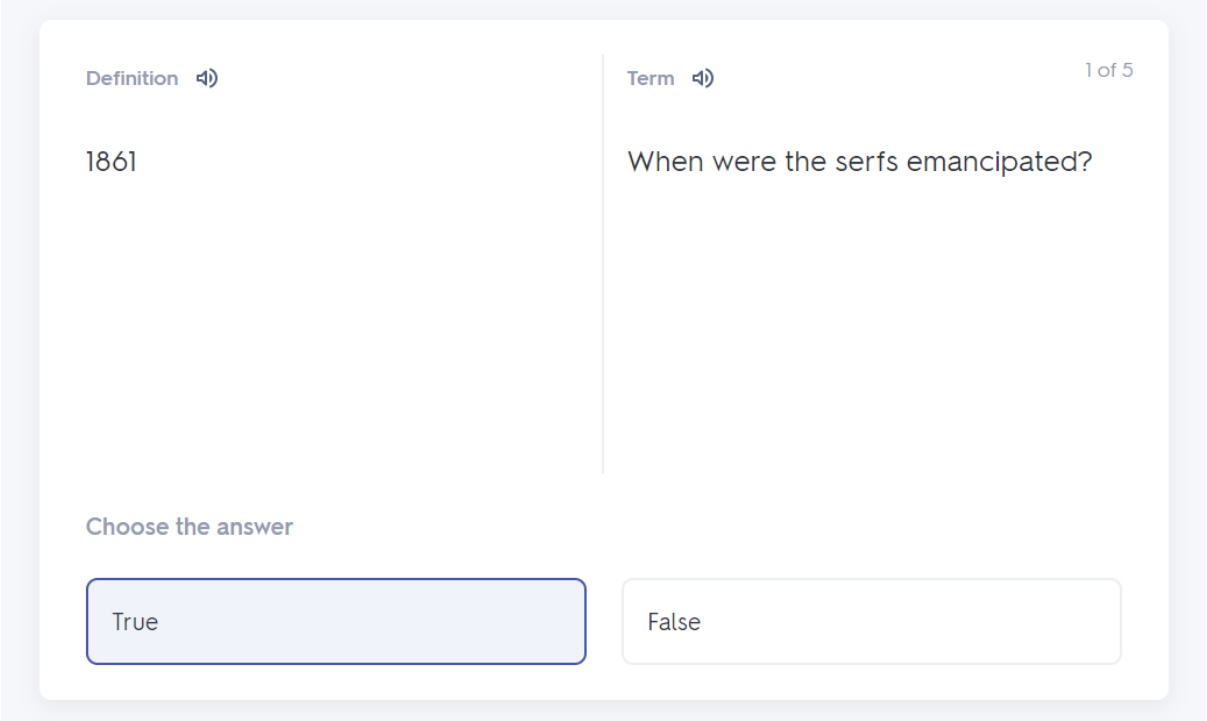
Quizlet Screenshot 1/7 - Has the list of sets, in order of most recently used, has a search and filters system to find sets easily, many drop downs to navigate website, simple clean design.



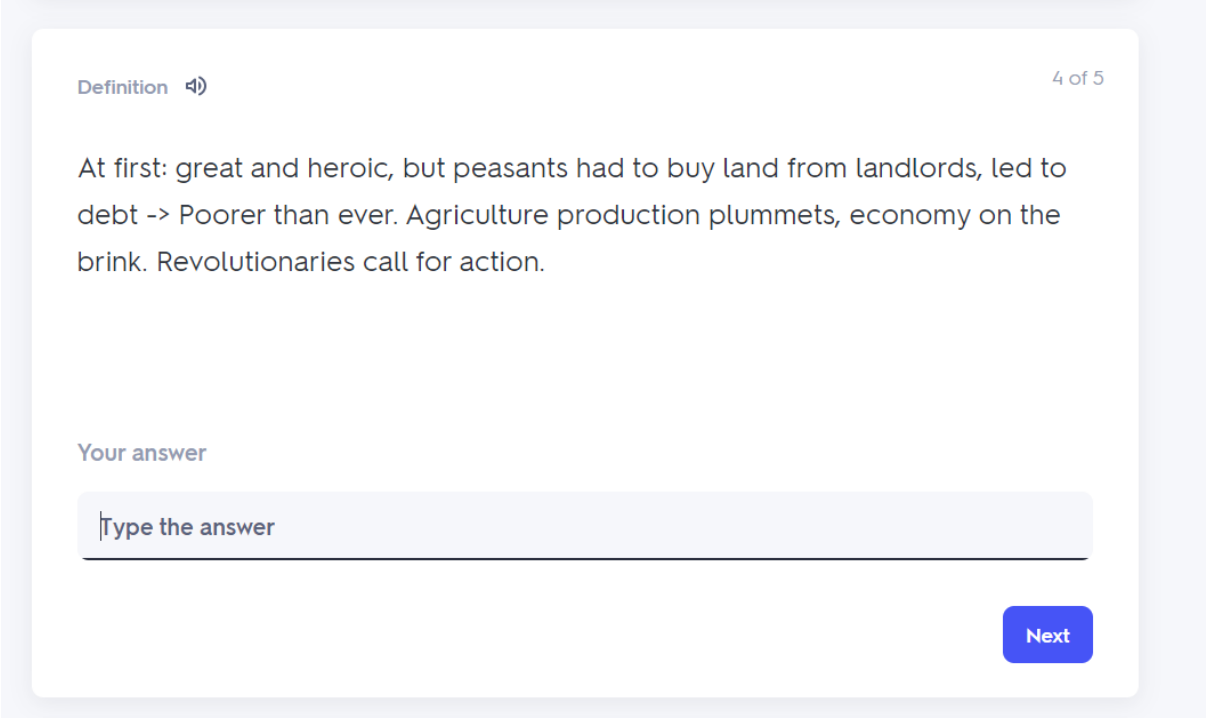
Quizlet Screenshot 2/7 - Able to scroll through the set and can be edited or read aloud.



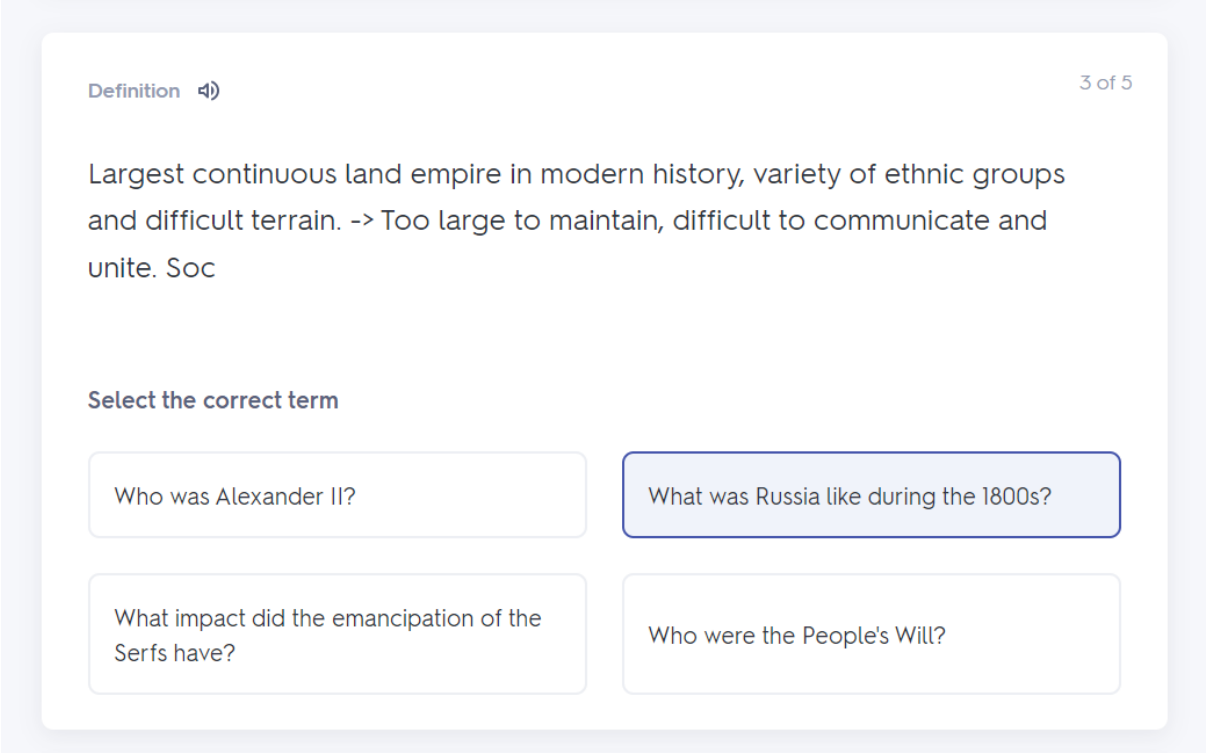
Quizlet Screenshot 3/7 - An example of different learning methods, this shows the digital flashcards.



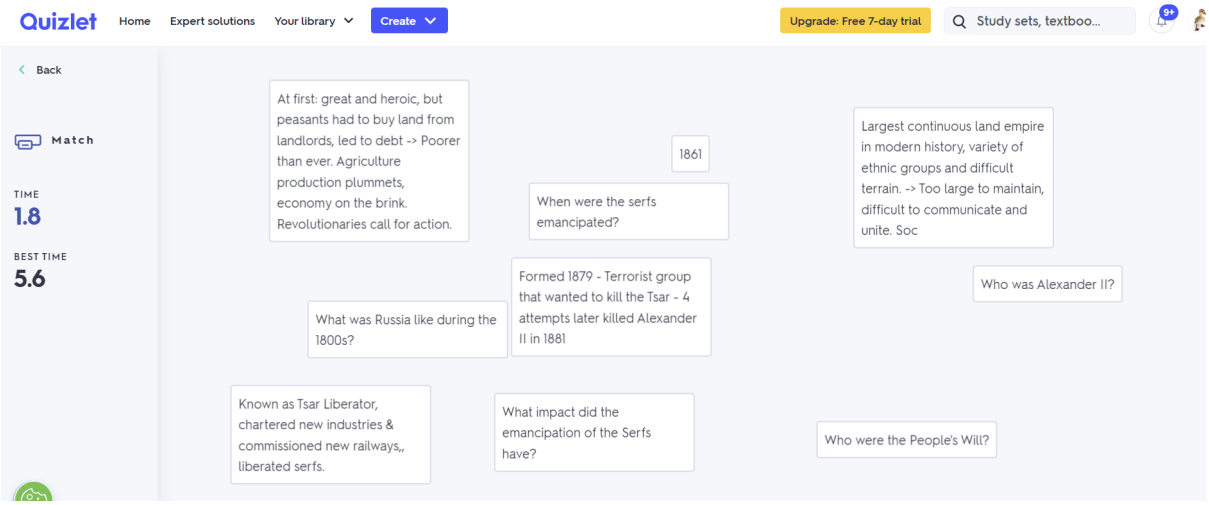
Quizlet Screenshot 4/7 - Question type, True or False



Quizlet Screenshot 5/7 - Question type, type the correct answer.



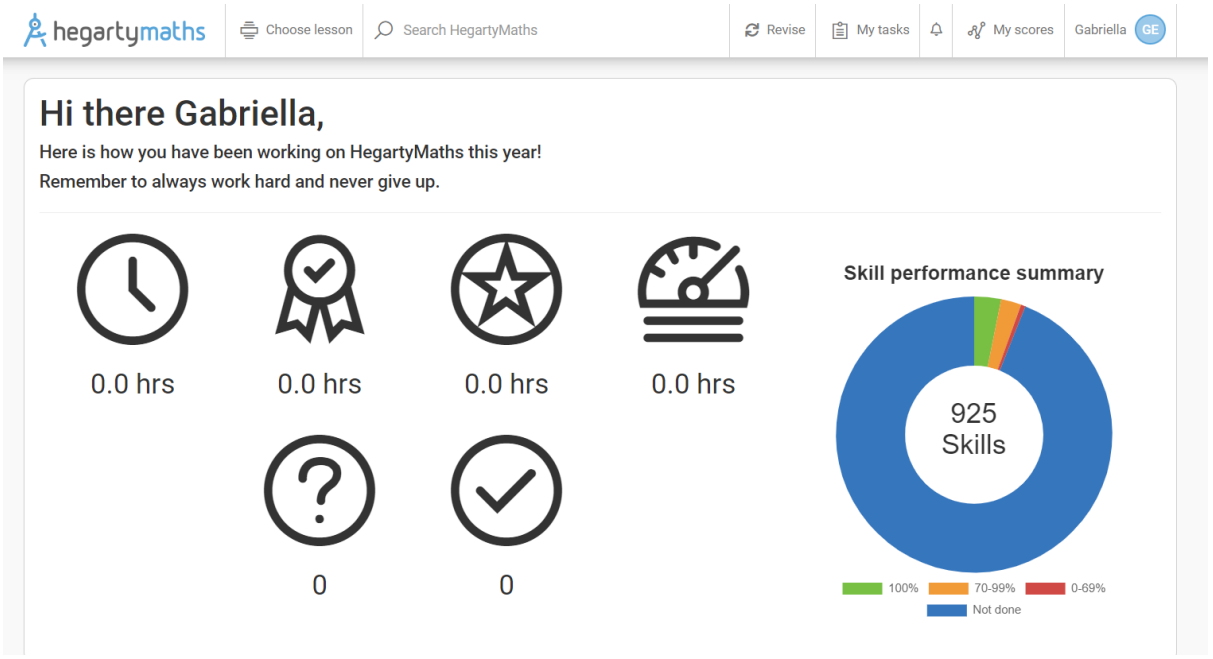
Quizlet Screenshot 6/7 - Question type, multiple choice



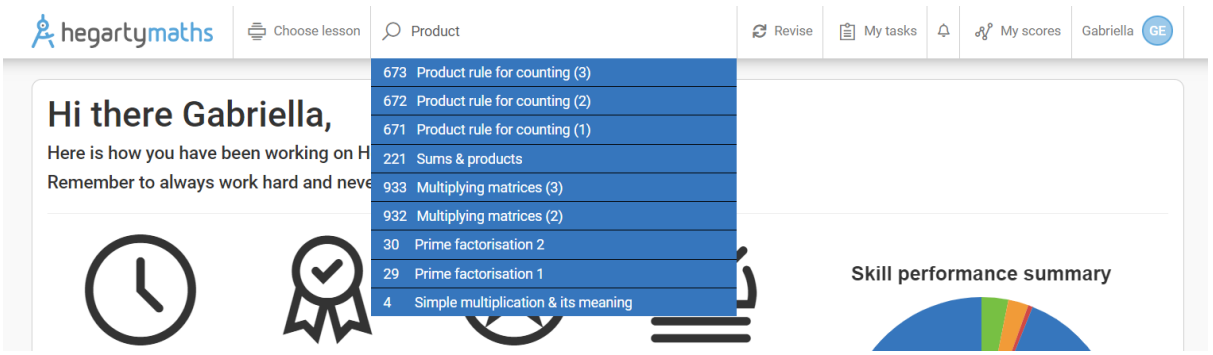
Quizlet Screenshot 7/7 - Example of the match game.

HEGARTY MATHS:
Hegarty Maths is an educational resource that is used for studying mathematics, it uses a bank of existing questions and exercises from professionals. This guarantees that the material is accurate, and has accompanying videos that provide explanations and demonstrations. This enables students to secure their knowledge and are able to track their progress and compare it to their previous attempts as well as the average of all users, this adds a competitive edge that encourages students to improve their scores and times. However, there is a distinct lack of customisation, other than the content chosen, every users' experience seems to be the exact same. Indeed, it is not possible to add material and, of course, it is limited to only focusing on Maths.

Name: Gabriella Emerson Candidate Number: 7346 Centre Number: 61853

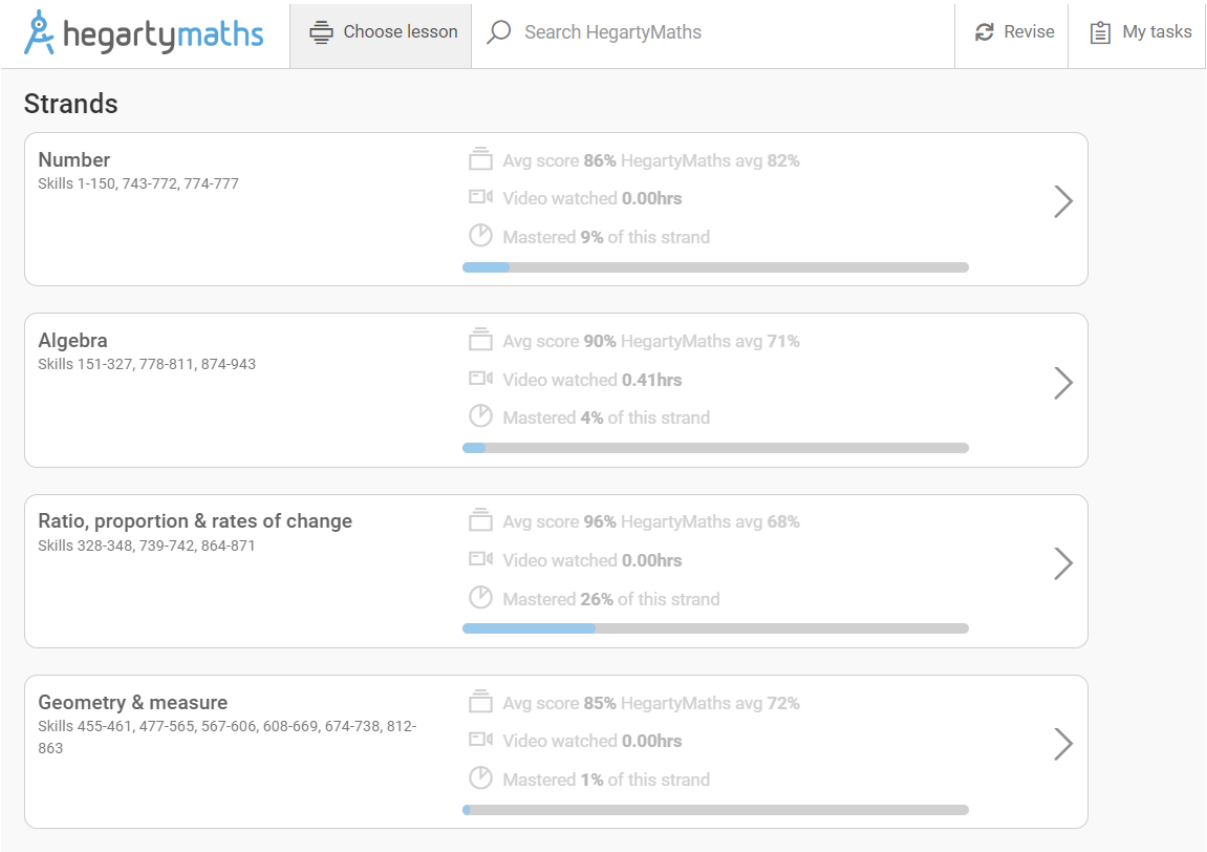


HegartyMaths Screenshot 1/8 - The welcome page, includes a list of statistics, of both the week and overall.

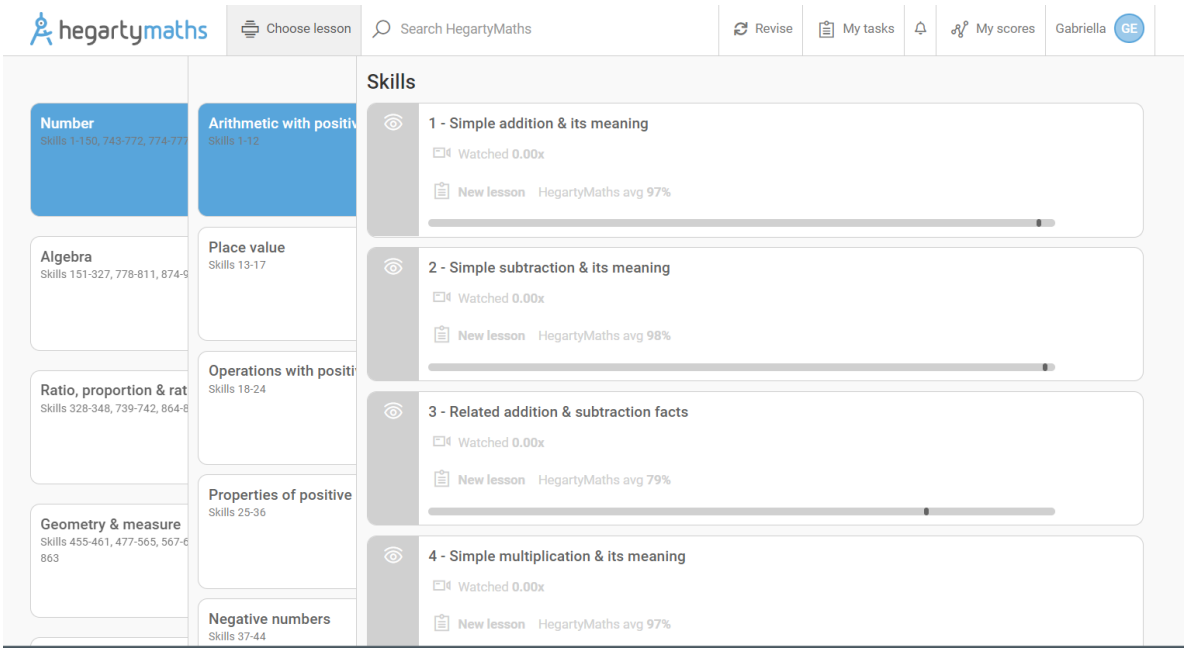


HegartyMaths Screenshot 2/8 - Suggestion of related topics when searching.

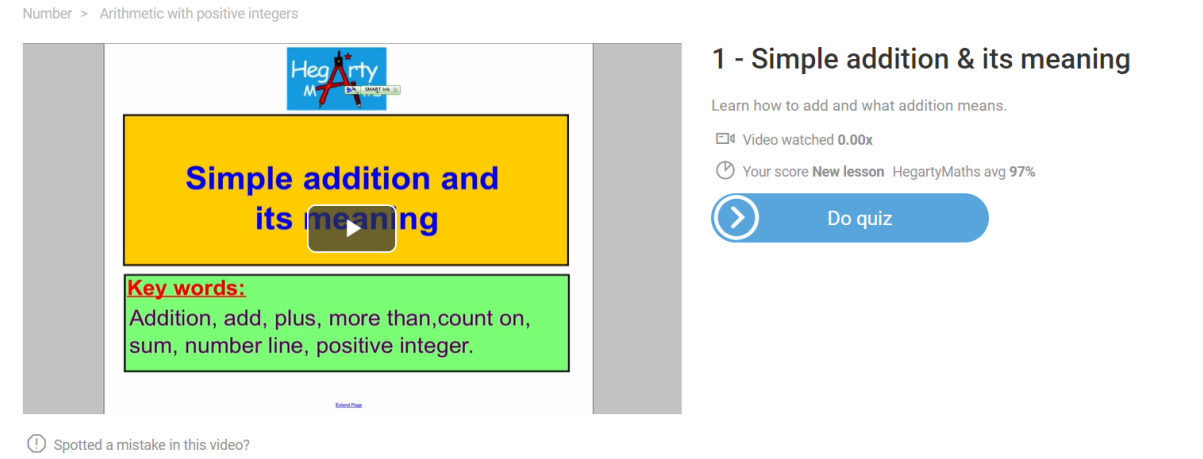
Name: Gabriella Emerson Candidate Number: 7346 Centre Number: 61853



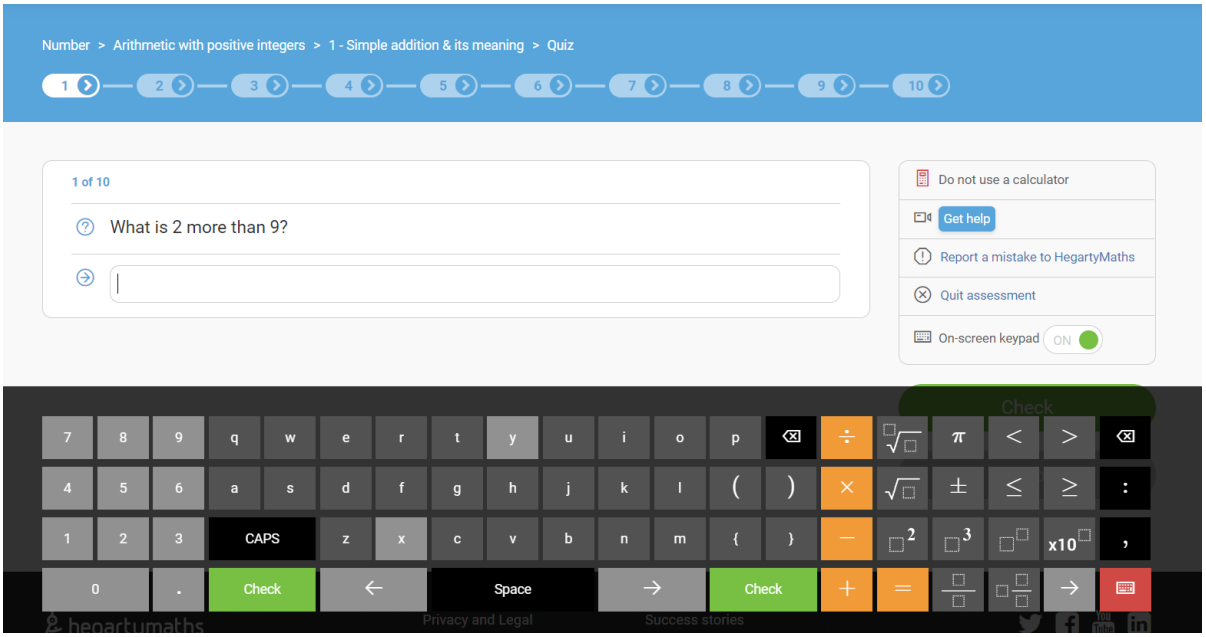
HegartyMaths Screenshot 3/8 - List of topics with users progress and statistics.



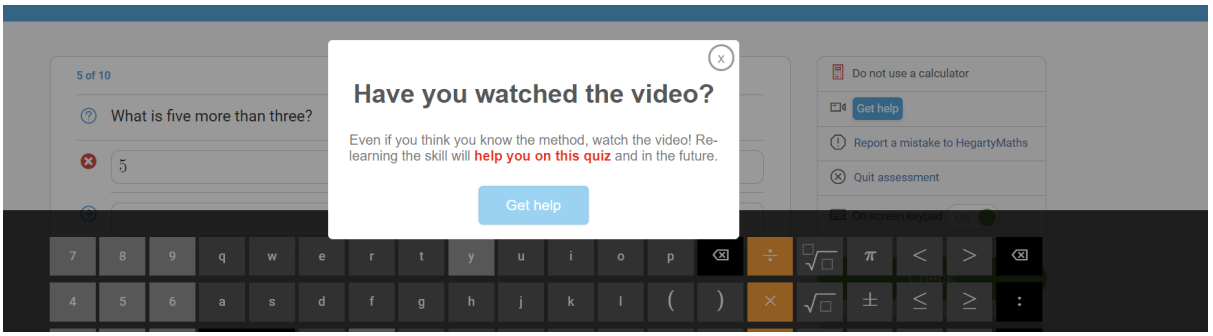
HegartyMaths Screenshot 4/8 - The lessons in clearly defined sections and the statistics associated with them.



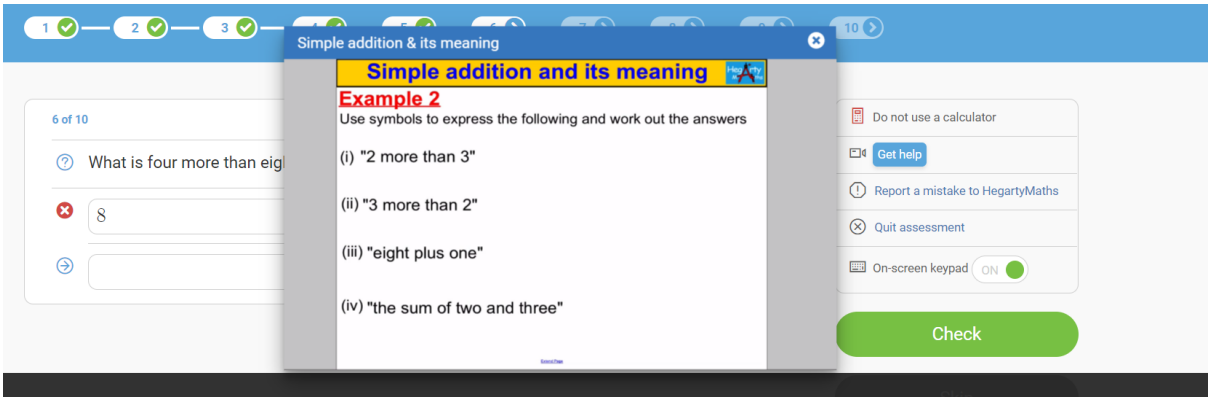
HegartyMaths Screenshot 5/8 - Option to learn from video before completing questions.



HegartyMaths Screenshot 6/8 - Question example, keyboard to ensure all mathematical symbols are available.

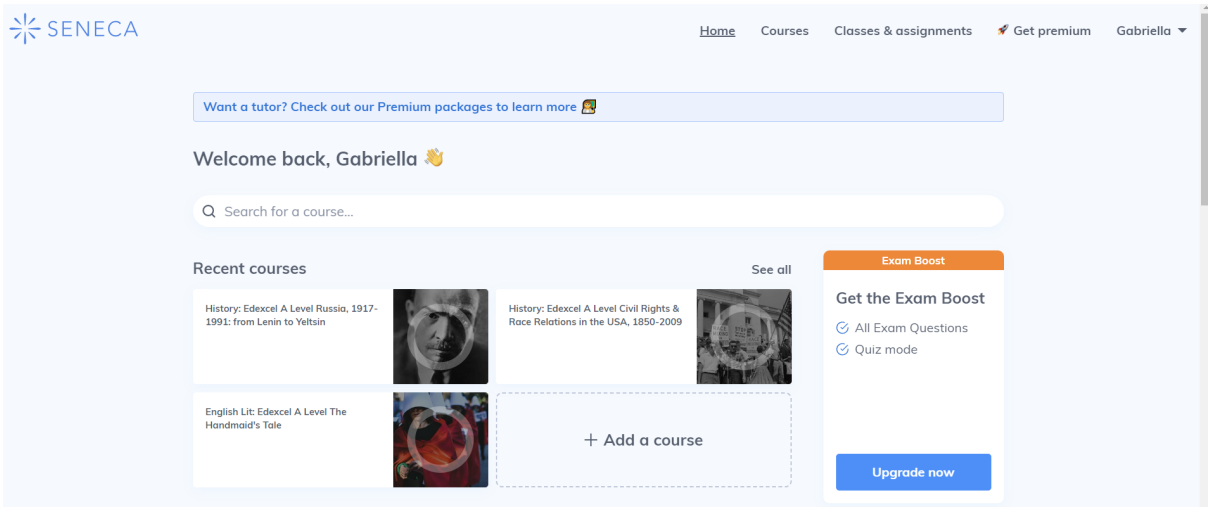


HegartyMaths Screenshot 7/8 - When questions are answered incorrectly, a pop-up window appears, suggesting to watch a video to understand the content.



HegartyMaths Screenshot 8/8 - If chosen, the video appears in a pop-up window that can be dragged around.

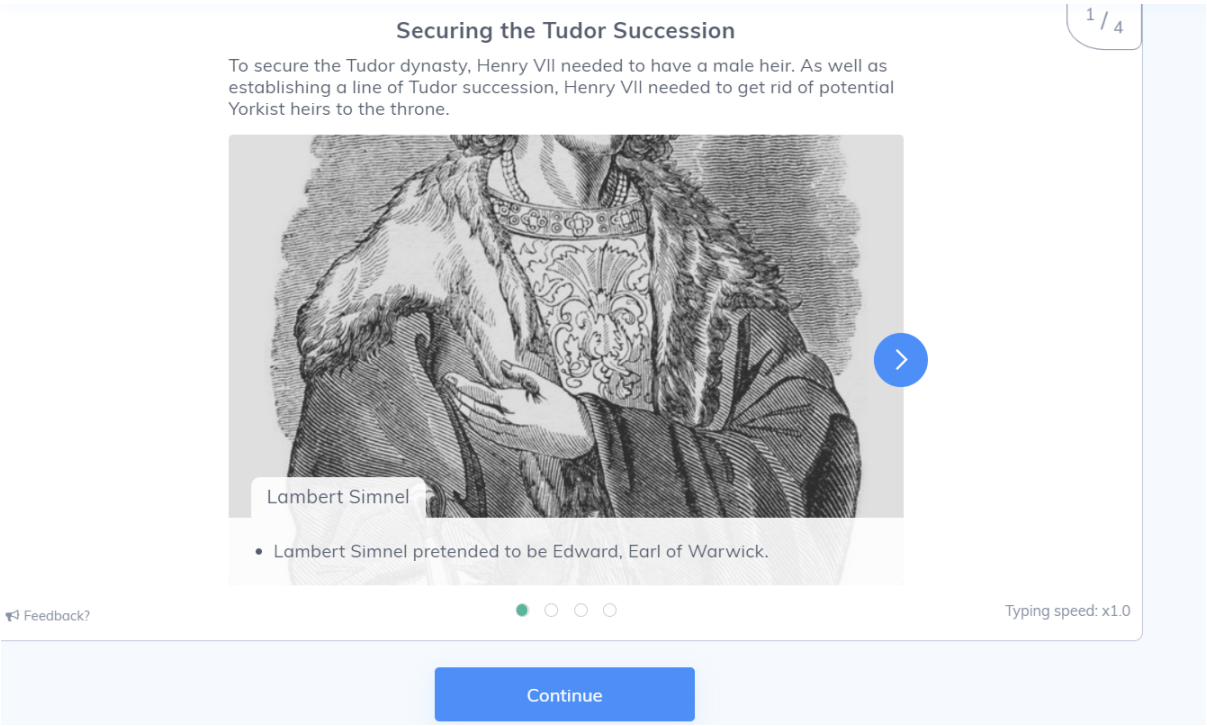
SENECA:
Seneca Learning is an online learning platform that is used for revision, providing a range of subjects that can be studied. Seneca uses the concept of association, with one of its most notable features being the use of a memory palace, which assigns units learnt in well-known geographical locations to aid in remembering the content. In addition, the combination of fun motivational comments with gifs and tracking their work encourages students to study. Seneca also includes different question types, with different levels of difficulty, which aids in engaging the users, as the amount of concentration required varies. Unfortunately, there are no games, as there are with Quizlet, and equally there is no way to add custom material, which limits you to the library of Seneca and its paywalls.



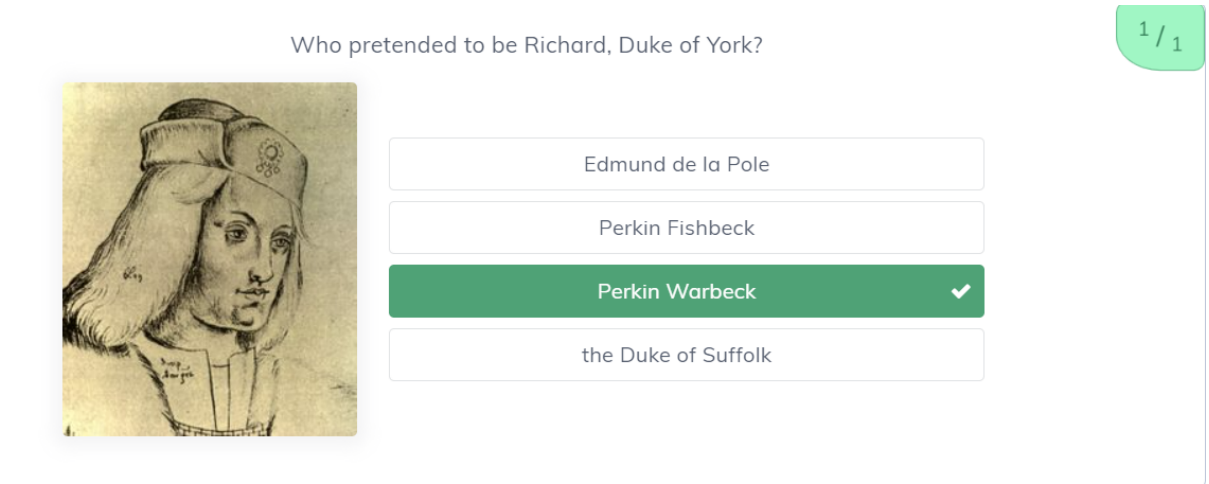
Seneca Screenshot 1/16 - The welcome screen that lists the most recent courses for convenience.

Name: Gabriella Emerson Candidate Number: 7346 Centre Number: 61853

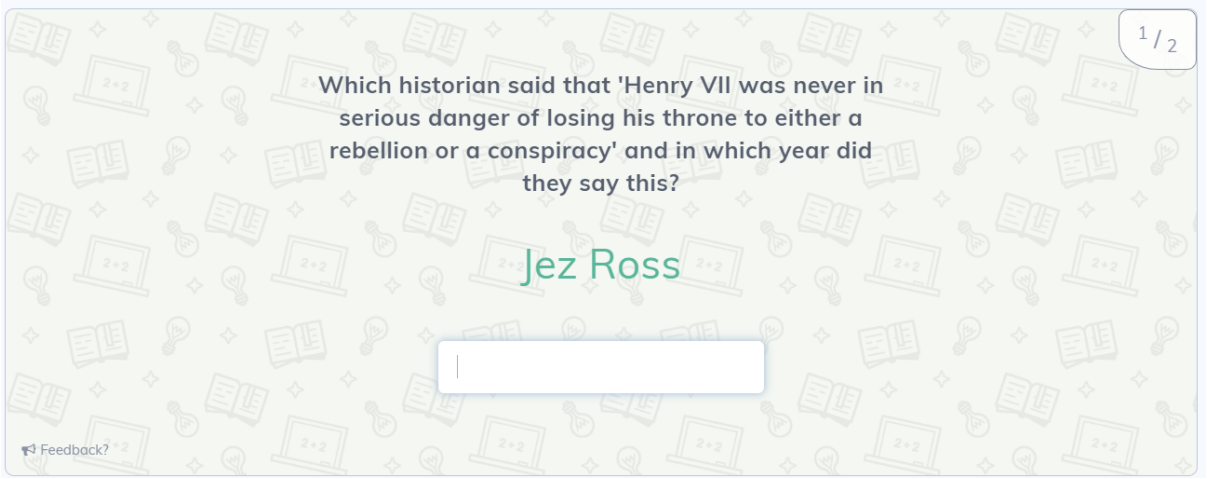
Seneca Screenshots 2/16 - When a course is selected, the statistics for the course is shown.



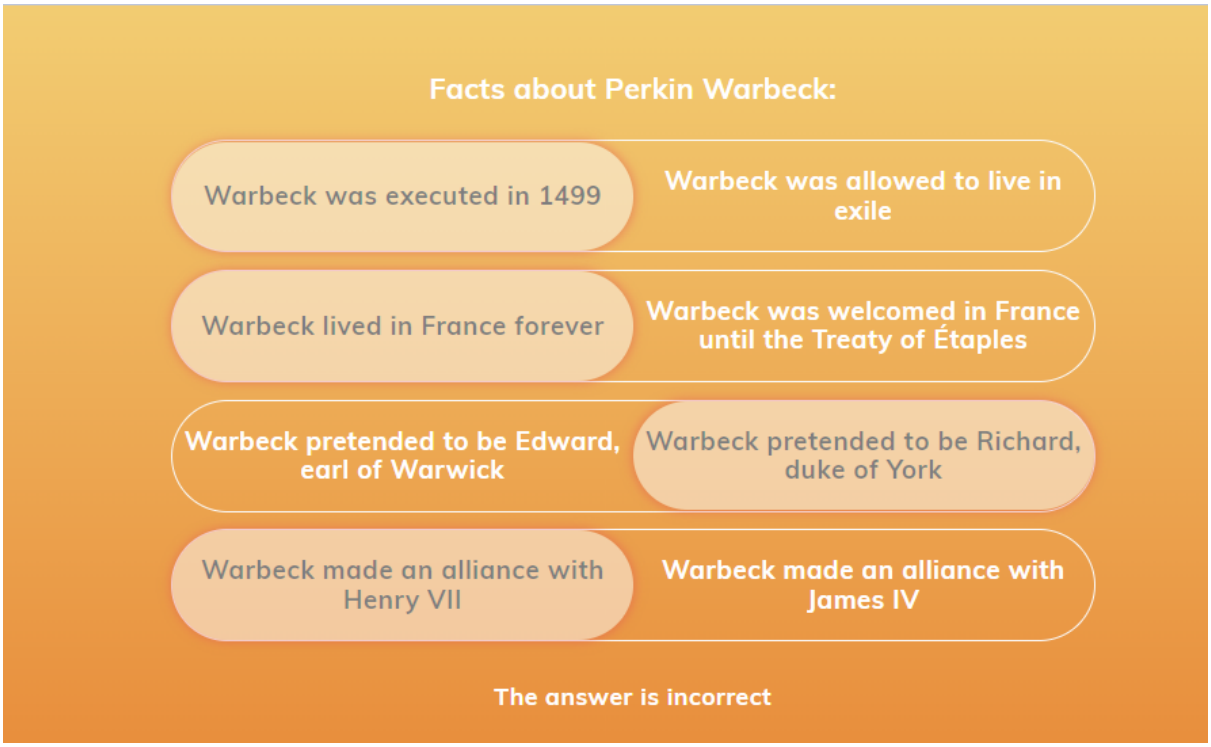
Seneca Screenshot 3/16 - Example of information, carousel of images with text



Seneca Screenshots 4/16 - Question Type: Multiple Choice

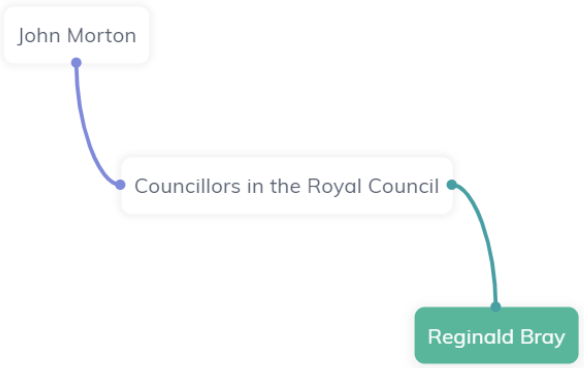


Seneca Screenshot 5/16 - Question type: Two Parts



Seneca Screenshot 6/16 - Question type: Sliders, hot and cold.

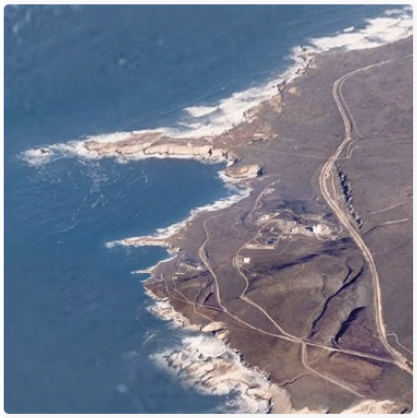
1 / 1



Seneca Screenshot 7/16 - Question type: Fill in the blanks

1 / 1

Henry VII's Regional Hold



Henry VII had a much stronger hold over areas in the south and east of England than the north.

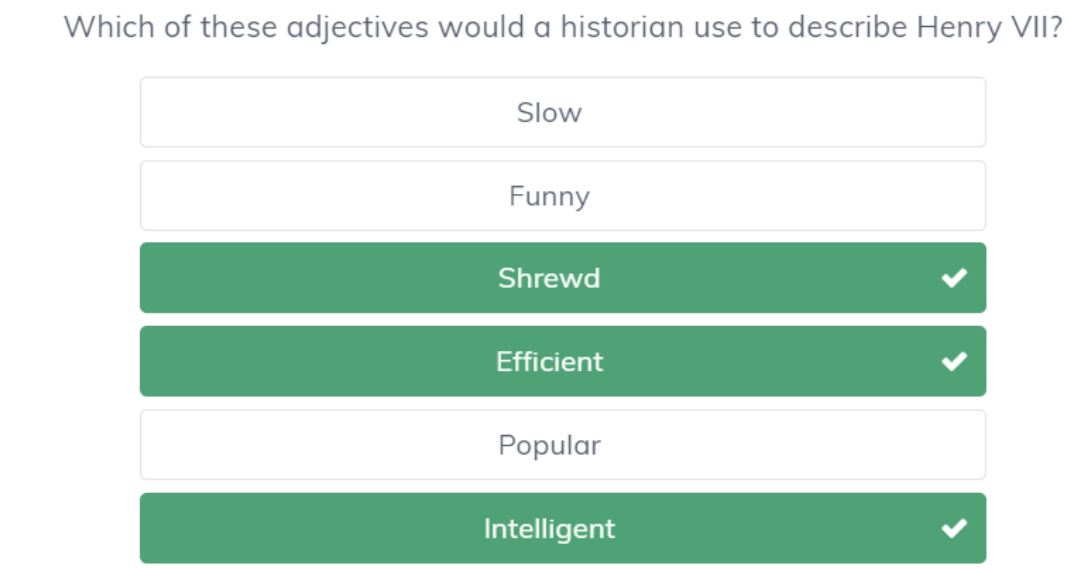
Seneca Screenshot 8/16 - Question Type: Fill in the blanks

2 / 2

In which years did Henry VII issue Acts of Attainder?

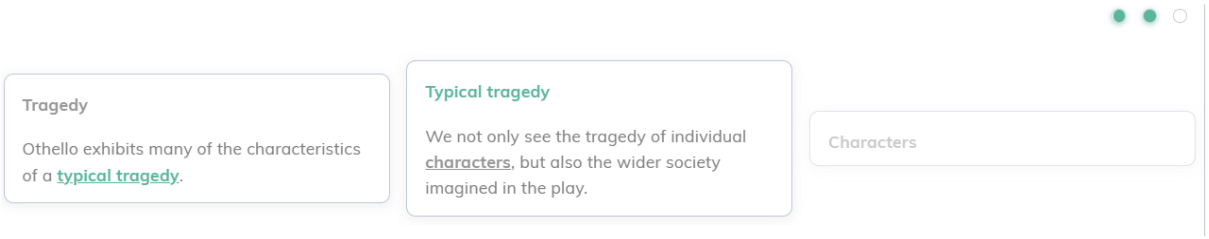
1487	*For the Battle of Stoke.*
1495	*To rally against Sir William Stanley.*
1489	*For the Northern Rebellion.*

Seneca Screenshot 9/16 - Question Type: Timeline



Seneca Screenshot 10/16 - Question type: Multiple answers

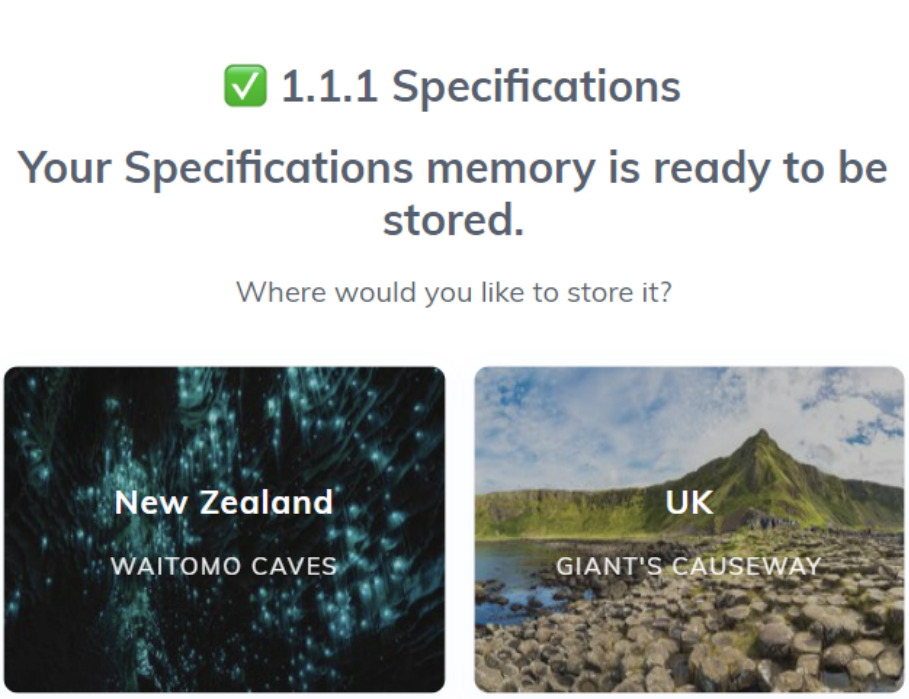
Seneca Screenshot 11/16 - Question type: standard question and answer



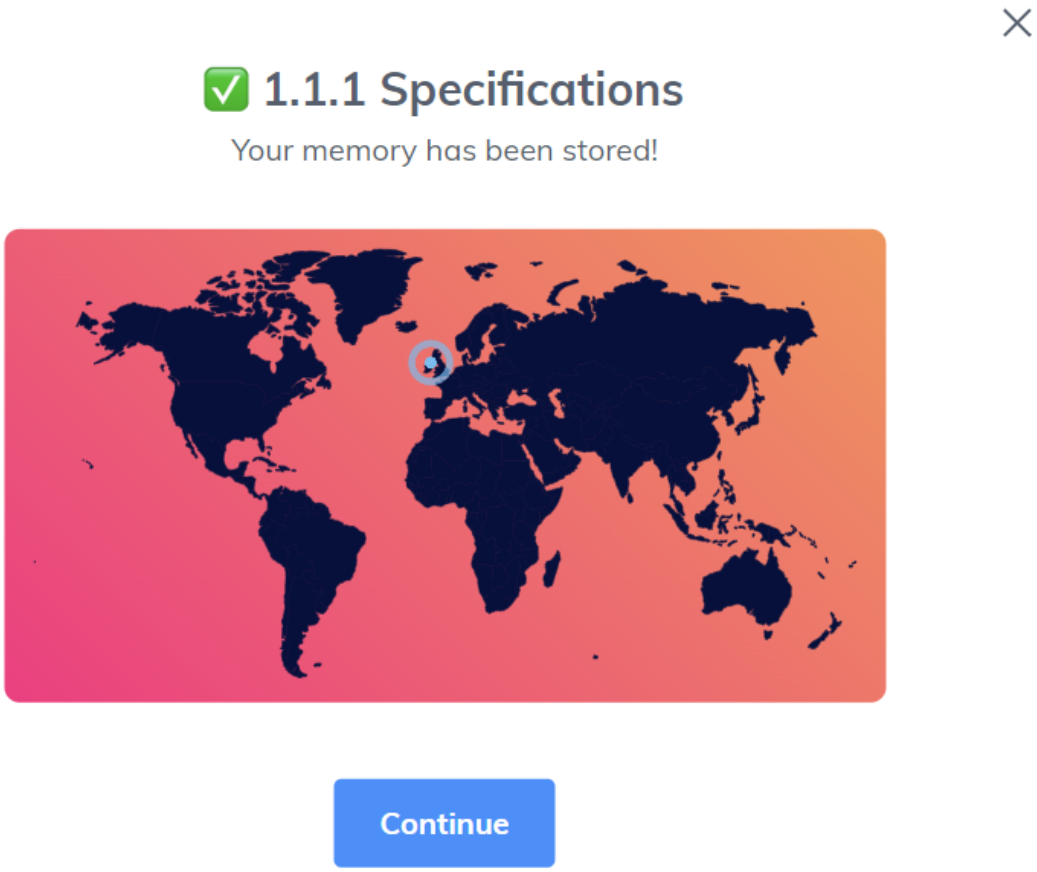
Seneca Screenshot 12/16 - Able to cycle through related information.

Seneca Screenshot 13/16 - Shows the statistics for that session

Seneca Screenshot 14/16 - Motivational notes with gif

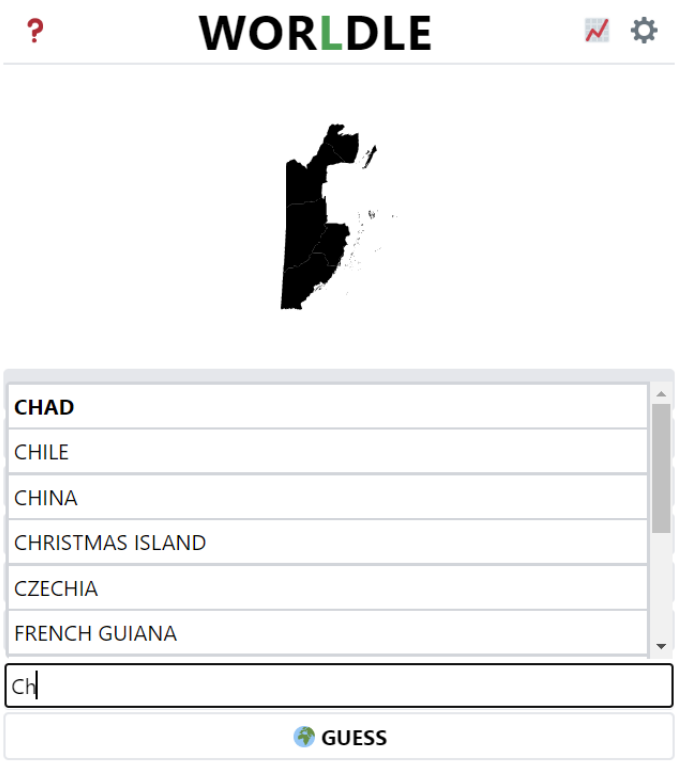


Seneca screenshot 15/16 - Get to pick where the lesson is ‘stored’

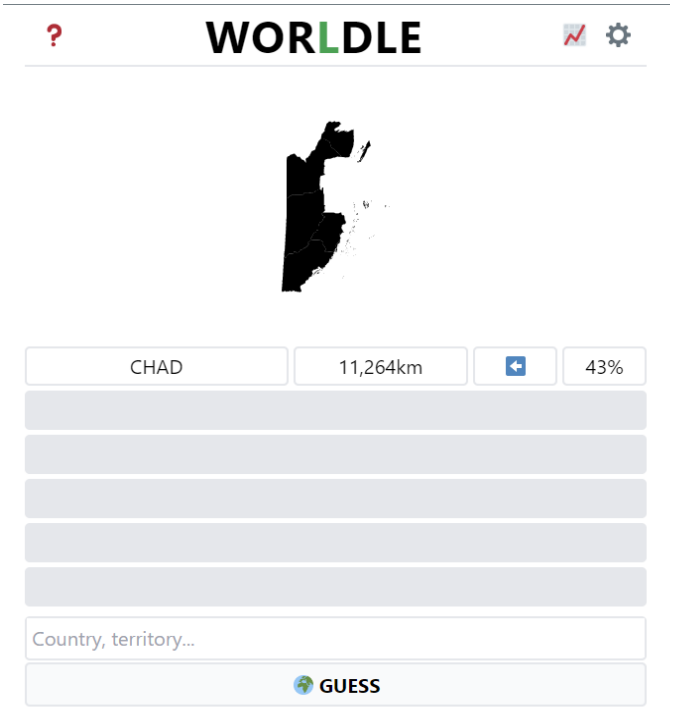


Seneca Screenshot 16/16 - Shows the location of where the session has been ‘stored’

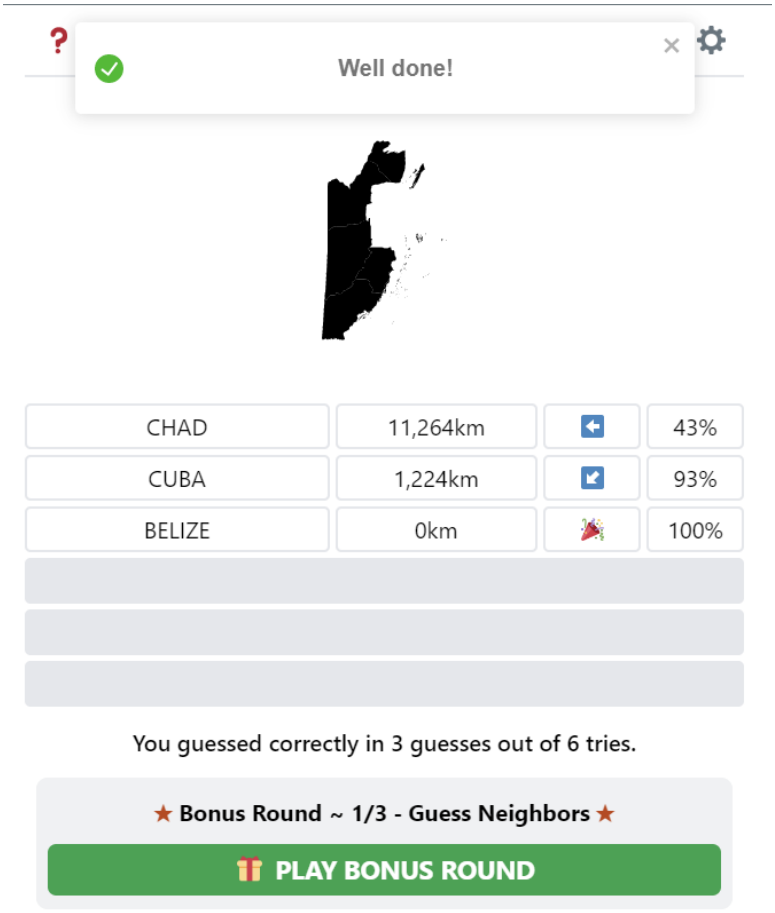
WORLDLE:
Unlike the other applications, Worldle - inspired by the global phenomenon Wordle - is not an educational application, its primary purpose is to entertain its users, yet it still manages to be informative by getting its users to actively participate and use their existing knowledge. The simplistic yet an innovative take on the childhood game Hot and Cold demonstrates the different forms that learning and nurturing knowledge can take. It doesn't seem very complex and works in a way not dissimilar to the method of using a bank of randomised questions, yet the additional element of comparing the distance and direction gives an additional element of complexity.



WORLDLE Screenshot 1/3 - Randomly generates a country, when text is entered, countries are suggested.



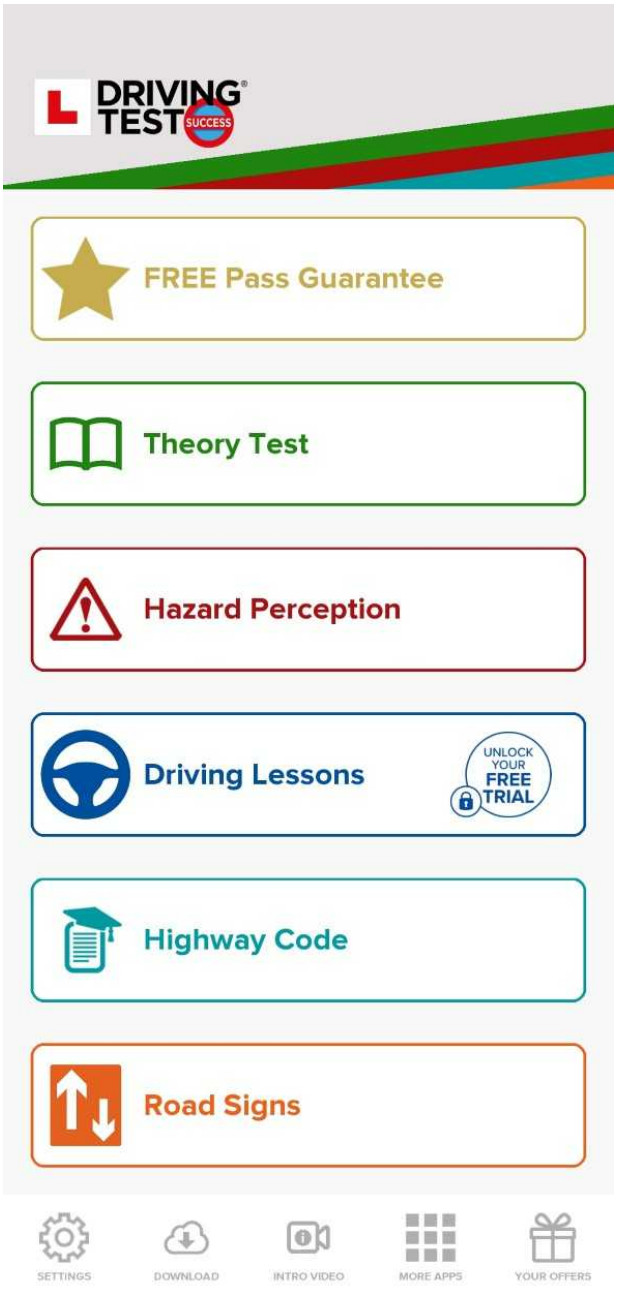
WORLDLE Screenshot 2/3 - Once a country is guessed, the distance and direction from the correct country is added.



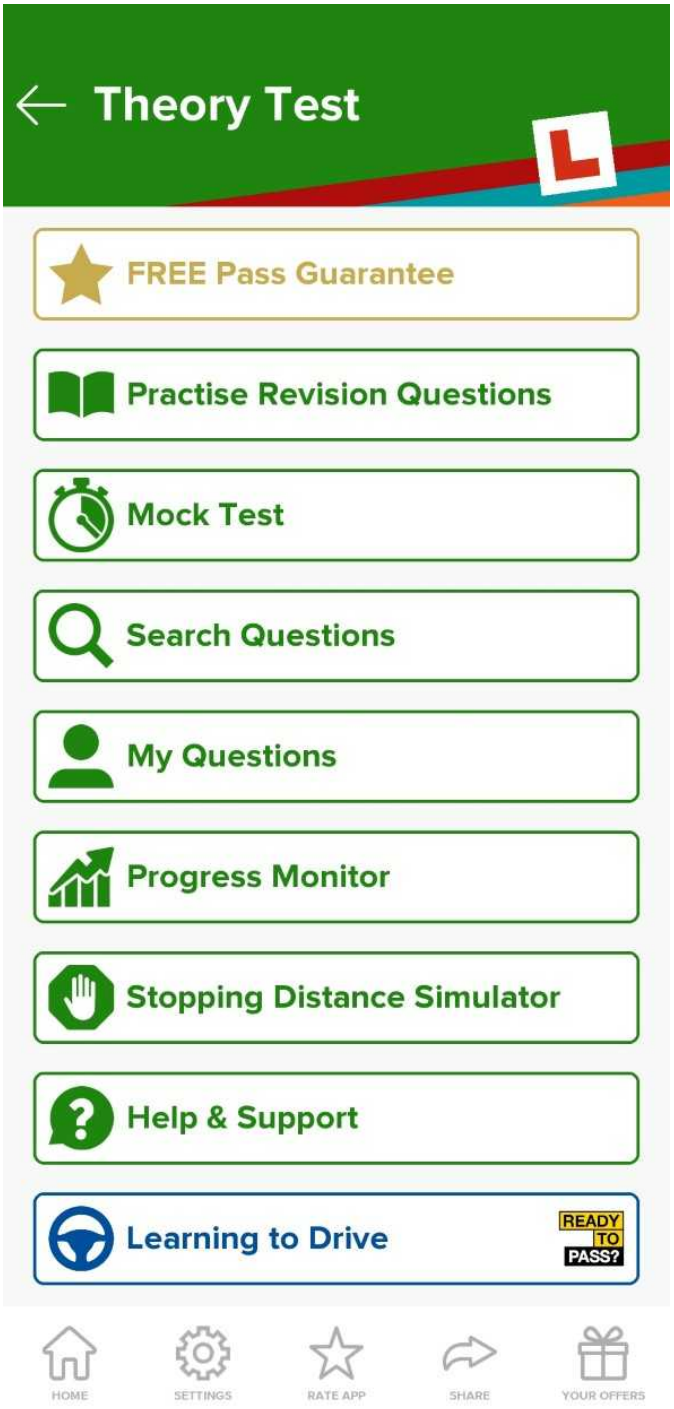
WORLDLE Screenshot 3/3 - Win state, has a window with ‘well done’

DRIVING 4 IN 1 THEORY:

While not as prolific as the other systems, there is an undeniable charm to the 4 in 1 app, which helps its users to revise for their Driving Theory tests. Like Quizlet, it has a range of ways to deliver the content, using the standard system of having a bank of questions that are randomly cycled through. The most interesting feature is the road sign game, which makes users actively want to learn and re-visit the material. Yet, there is an underlying issue of the way the information is presented throughout the app, it is very basic, and is hindered by the lack of content available for the questions, as such, it is very common to repeat the same questions.



Driving 4 in 1 Screenshot 1/6 - First page, brightly coloured with links to other pages



Driving 4 in 1 Screenshot 2/6 - Example of subpage

Driving 4 in 1 Screenshot 3/6 - The game - fun and interactive, break from conventional question types, helps to engage and retain information

Driving 4 in 1 Screenshot 4/6 - Point system and check points.

Driving 4 in 1 Screenshot 5/6 - Example of Question, Correct answer.

Driving 4 in 1 Screenshot 6/6 - Example of Question, incorrect answer.

Identification of User Needs

The input from the students has made the requirements for this project clear. The feedback has conveyed that the interactivity of the software is a must, as well as having a variety of question styles to cater to the needs of different subjects, such as the unique challenge of making questions for essay subjects or the ability to have images and diagrams. The interactivity could also take the form of a game to provide a light-hearted and entertaining element, this could equally be achieved with a cheerful house-style throughout the project. As well as this, the ability to customise their experiences seems to be a popular option, particularly by uploading their own material, a feature that should definitely be considered. Another feature that was mentioned was the ability to track their progress, whether that is their score or the amount of time they have dedicated to studying.

Objectives

1. LOGIN SYSTEM

   a. Users should be able to sign up and have their details added to the database.
   b. Users should be prevented from having the same username as one that already exists in the database.
   c. Passwords should have a validation check, such as having a special character or being at least 8 characters long.
   d. Existing users should be able sign in by checking their username and password against the login database.
2. QUIZ SYSTEM
   a. Questions and answers should be stored in a database where they are organised by subjects in separate databases.
   b. Users should be able to make their own questions, being able to edit them as well as read them - all of this should be stored in the Question Database.
   c. Users should be able to customise their quiz: being able to customise the number of questions and which question bank they answer from.
   d. Questions should be randomly selected based on the Question ID and then read from the database.
   e. Users' answers should be checked by reading the database.
   f. Ensure that there are different question types/ styles for the user to answer, e.g.) fill in the blank, timelines, hot and cold sliders or multiple choice.
3. GLOBE
   a. An interactive map that can be dragged around in order to navigate the continents, reminiscent of Google Maps.
   b. Countries and their information should be stored in a database.
   c. The User should be able to search through the database and retrieve the country's information, once the information is retrieved, the info should be displayed as a fact file.
4. MATCHING GAME
   a. Information should be retrieved from the Question Database
   b. Users should be able to drag and drop terms and have this action validated by reading the Question Database to check their answer.
   c. Users should be able to customise the game: choose how many pairs they match up, how many rounds, what question bank they use.
5. SCORES SYSTEM
   a. There should be a database that stores each user's scores for every game they play, e.g.) quiz, matching ect.
   b. Users should be able to get their high score for each game.
   c. Users should be able to get the mean average score for each game.
   d. The User should also be able to get the percentile score and overall ranking compared to all other users for each game.

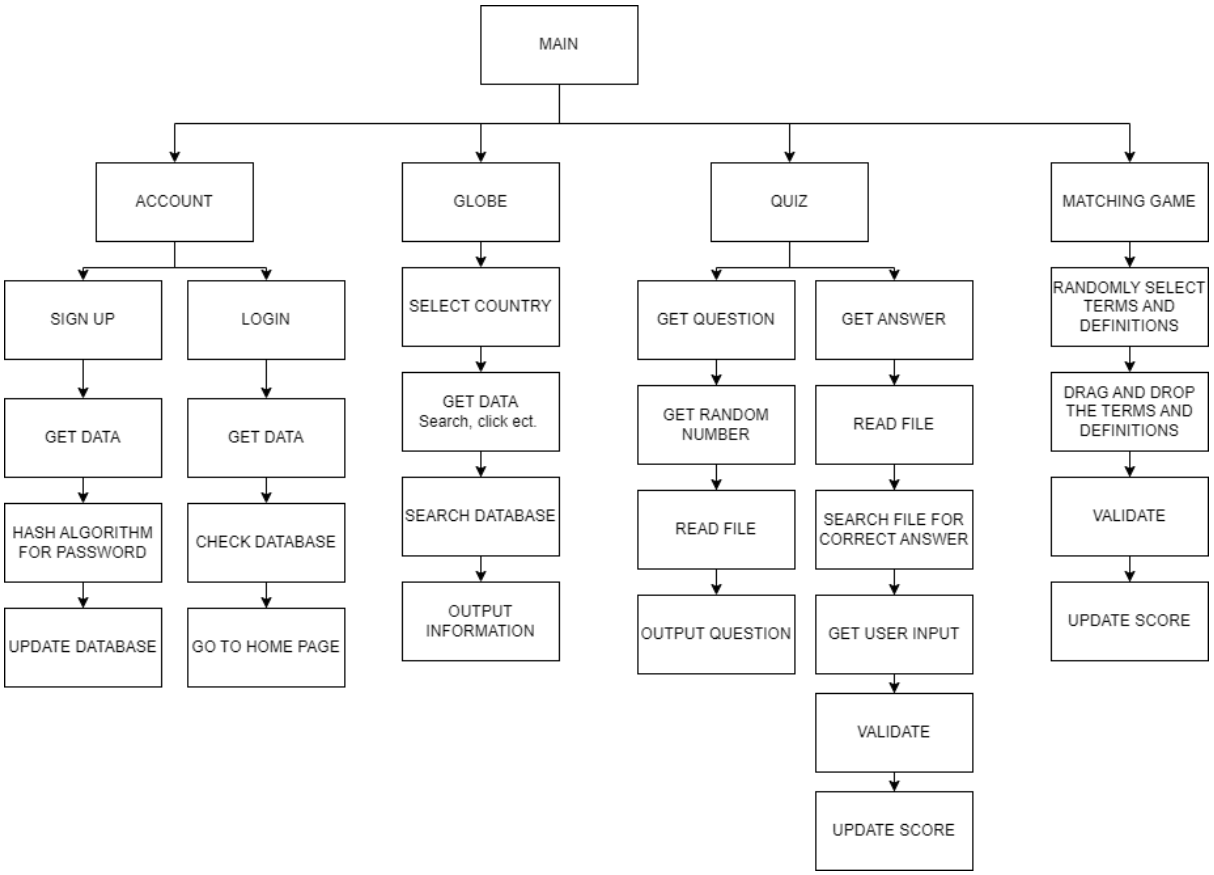
Justification of Chosen Solution

While it may have been possible for certain aspects of the project to be in a console application, such as the login system and basic quiz, other elements, such as the drag and drop game and the map could not be achieved without a basic graphical user interface. As such, it seemed to narrow down the different languages that could be used to complete this project.

One solution could be the blend of Javascript and HTML, two languages that I am relatively familiar with. This would undoubtedly be capable of providing an interactive and visual interface to ensure that students would be engaged with the software. Yet, this languages are generally used for front-end development, and as such a back-end language would need to be used in order to carry out the heavy processing, such as the maintenance of the database, as there is no common way to connect a Javascript client to the SQL database. The solution to this could be to have a separate C# back-end, but given the fact that C# is capable of being both front and back-end when combined with XAML, it seemed more logical to have a cohesive project that limited the number of languages used. As well as this, I am far more familiar with C# than HTML and Javascript, making it more reasonable to focus on the complexity of the project, rather than spending time on relearning Javascript. Indeed, C# was the most reasonable language for this project, not only because I am more comfortable with it, but also because it is powerful enough to maintain the front-end and also the complex procedures and databases.

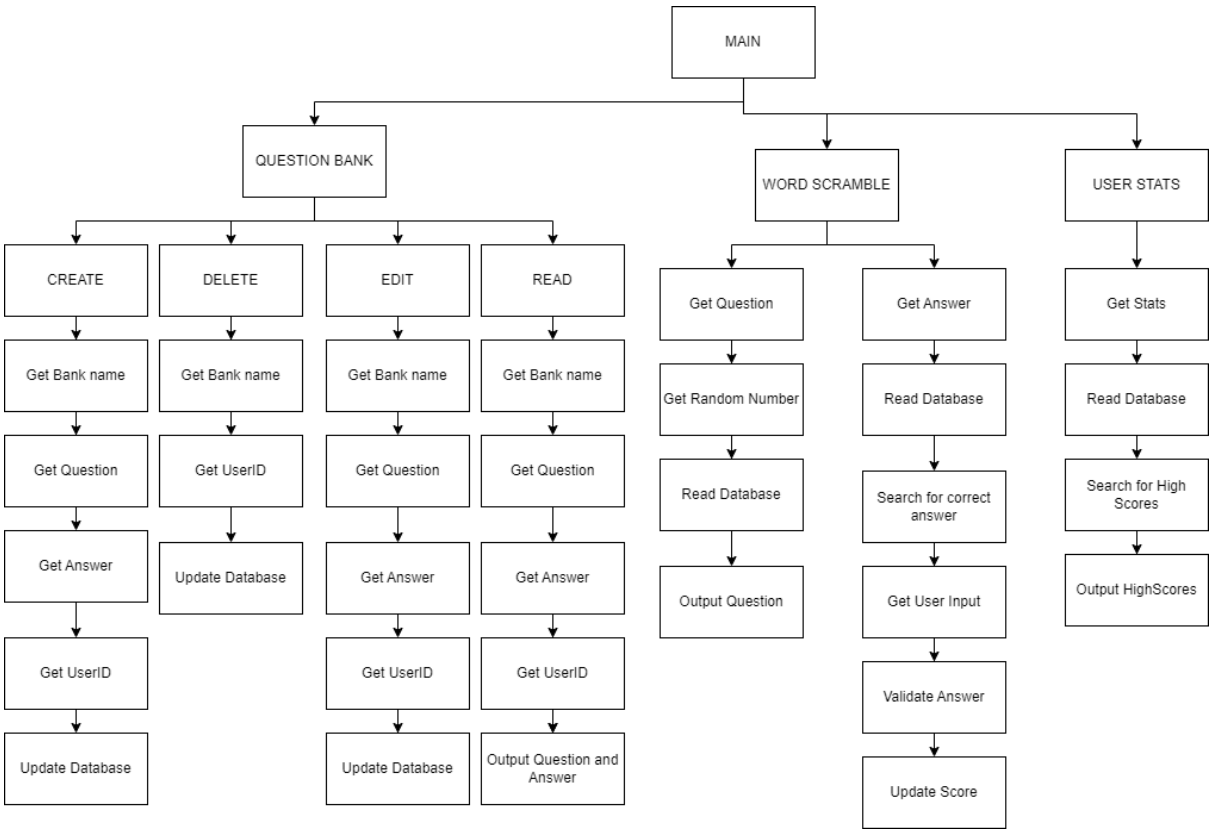
Design

Hierarchy Chart



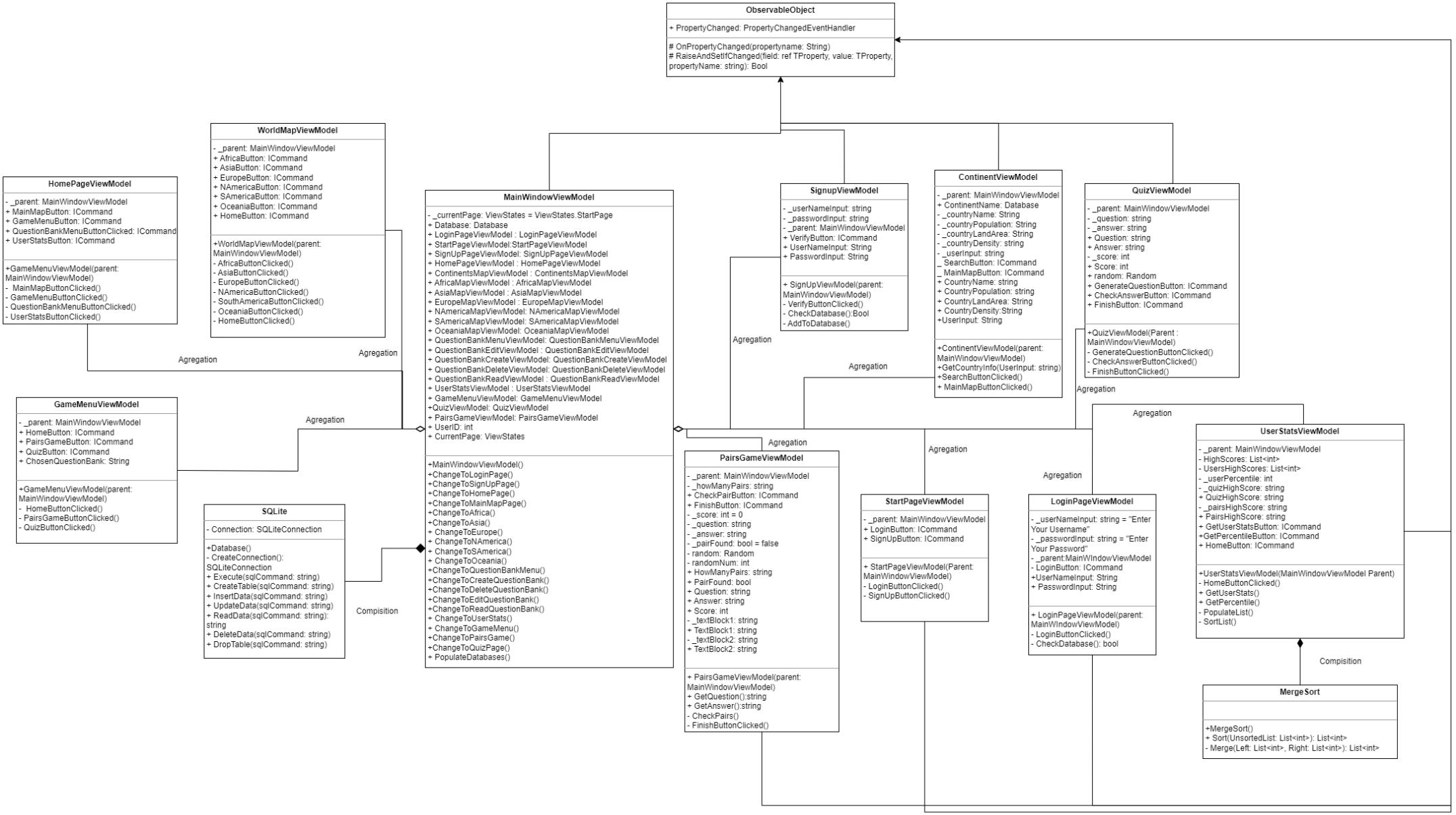
Name: Gabriella Emerson Candidate Number: 7346 Centre Number: 61853

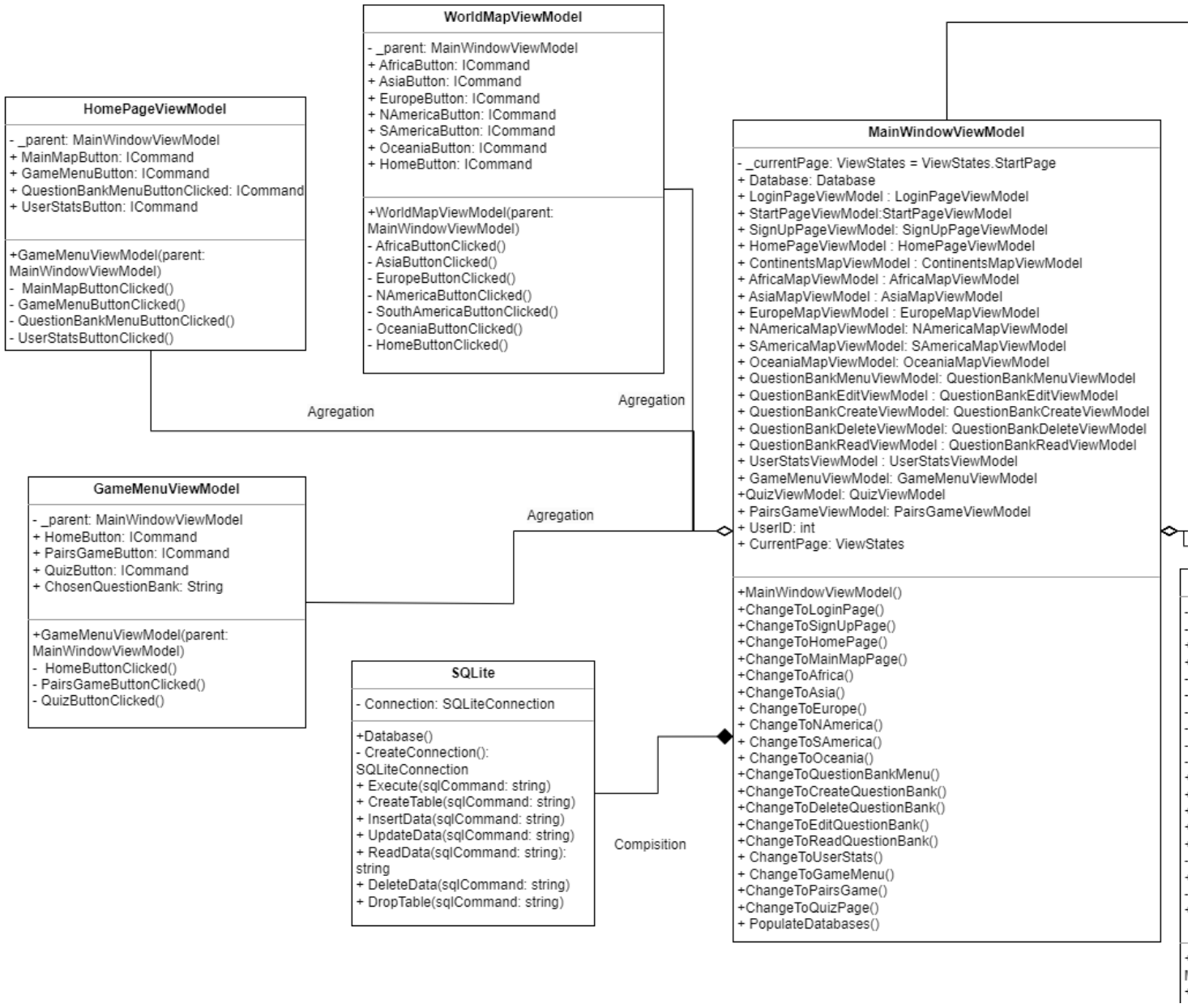
HIERARCHY CHART ADDITIONS:

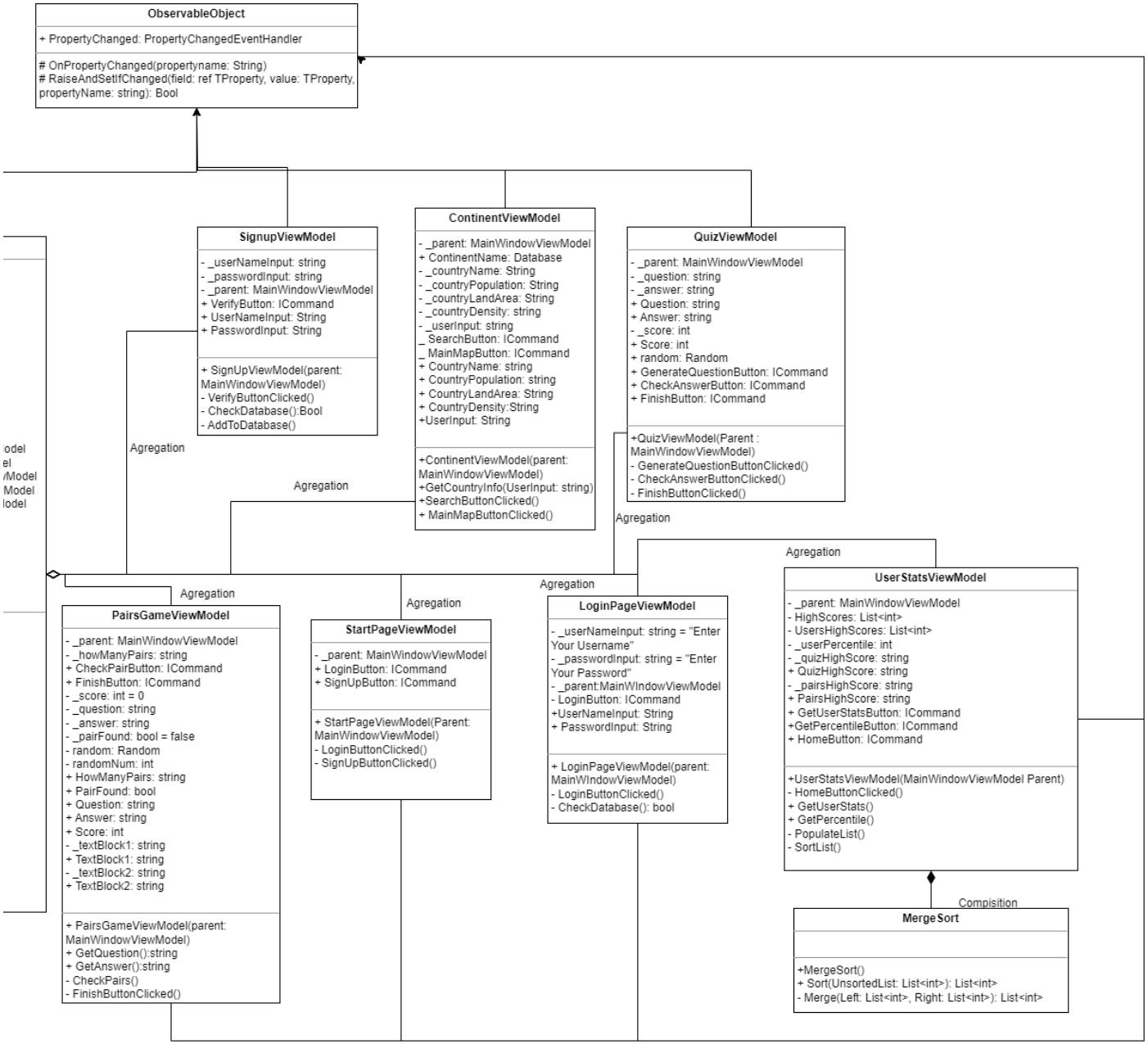


OOP Class Design

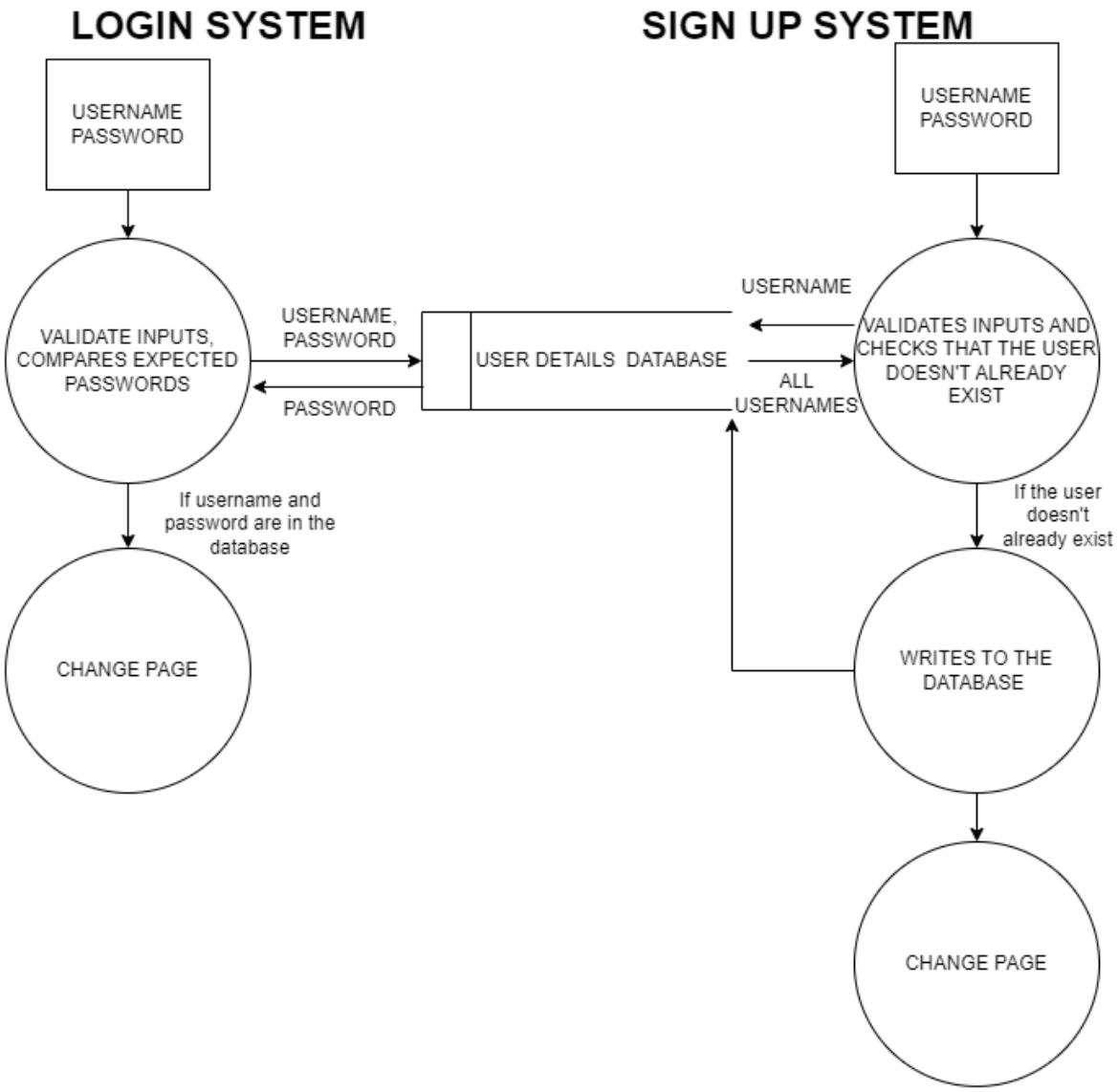
- Class Diagram

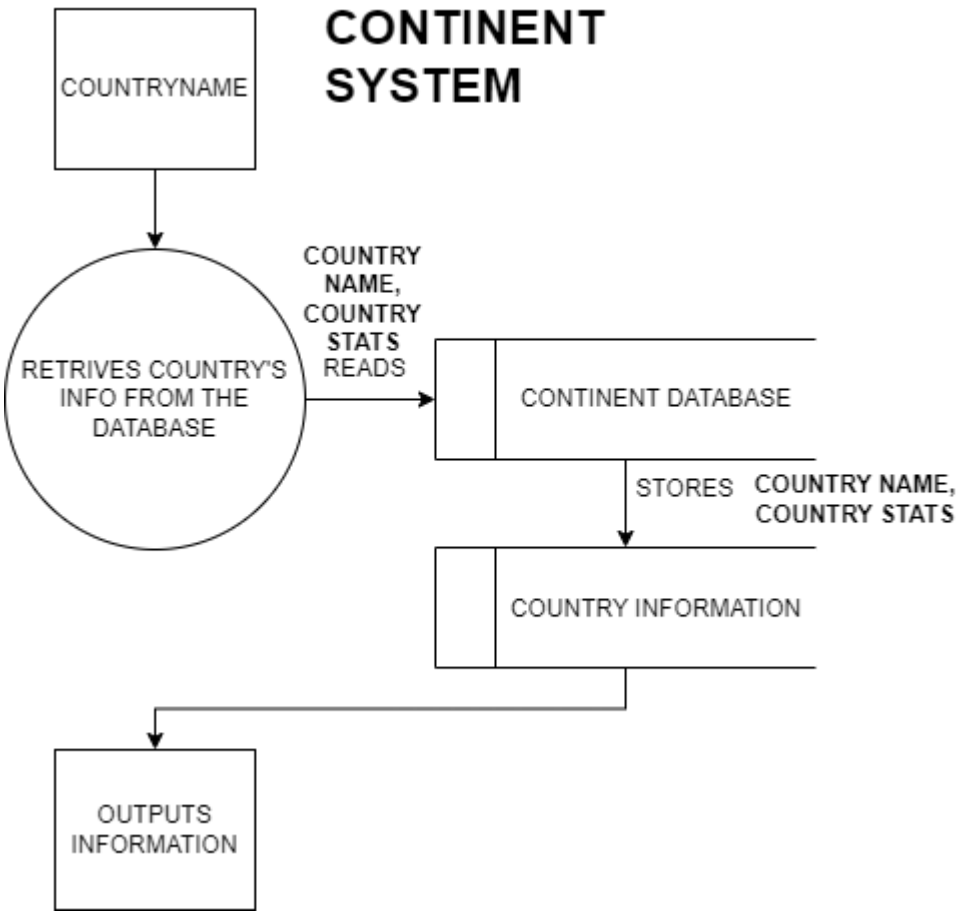

ObservableObject
+ PropertyChanged: PropertyChangedEventHandler
# OnPropertyChanged(propertyname: String)
# RaiseAndSetIfChanged(field: ref TProperty, value: TProperty, propertyName: string): Bool
WorldMapViewModel
- _parent: MainWindowViewModel
+ AfricaButton: ICommand
+ AsiaButton: ICommand
+ EuropeButton: ICommand
+ NAmericaButton: ICommand
+ SAmericaButton: ICommand
+ OceaniaButton: ICommand
+ HomeButton: ICommand
+WorldMapViewModel(parent: MainWindowViewModel)
- AfricaButtonClicked()
- AsiaButtonClicked()
- EuropeButtonClicked()
- NAmericaButtonClicked()
- SouthAmericaButtonClicked()
- OceaniaButtonClicked()
- HomeButtonClicked()
HomePageViewModel
- _parent: MainWindowViewModel
+ MainMapButton: ICommand
+ GameMenuButton: ICommand
+ QuestionBankMenuButtonClicked: ICommand
+ UserStatsButton: ICommand
+GameMenuViewModel(parent: MainWindowViewModel)
- MainMapButtonClicked()
- GameMenuButtonClicked()
- QuestionBankMenuButtonClicked()
- UserStatsButtonClicked()
MainWindowViewModel
- _currentPage: ViewStates = ViewStates.StartPage
+ Database: Database
+ LoginPageViewModel : LoginPageViewModel
+ StartPageViewModel:StartPageViewModel
+ SignUpPageViewModel: SignUpPageViewModel
+ HomePageViewModel : HomePageViewModel
+ ContinentsMapViewModel : ContinentsMapViewModel
+ AfricaMapViewModel : AfricaMapViewModel
+ AsiaMapViewModel : AsiaMapViewModel
+ EuropeMapViewModel : EuropeMapViewModel
+ NAmericaMapViewModel: NAmericaMapViewModel
+ SAmericaMapViewModel: SAmericaMapViewModel
+ OceaniaMapViewModel: OceaniaMapViewModel
+ QuestionBankMenuViewModel: QuestionBankMenuViewModel
+ QuestionBankEditViewModel : QuestionBankEditViewModel
+ QuestionBankCreateViewModel: QuestionBankCreateViewModel
+ QuestionBankDeleteViewModel: QuestionBankDeleteViewModel
+ QuestionBankReadViewModel : QuestionBankReadViewModel
+ UserStatsViewModel : UserStatsViewModel
+ GameMenuViewModel: GameMenuViewModel
+QuizViewModel: QuizViewModel
+ PairsGameViewModel: PairsGameViewModel
+ UserID: int
+ CurrentPage: ViewStates
+MainWindowViewModel()
+ChangeToLoginPage()
+ChangeToSignUpPage()
+ChangeToHomePage()
+ChangeToMainMapPage()
+ChangeToAfrica()
+ChangeToAsia()
+ ChangeToEurope()
+ ChangeToNAmerica()
+ ChangeToSAmerica()
+ ChangeToOceania()
+ChangeToQuestionBankMenu()
+ChangeToCreateQuestionBank()
+ChangeToDeleteQuestionBank()
+ChangeToEditQuestionBank()
+ChangeToReadQuestionBank()
+ ChangeToUserStats()
+ ChangeToGameMenu()
+ChangeToPairsGame()
+ChangeToQuizPage()
+ PopulateDatabases()
SignupViewModel
- _userNameInput: string
- _passwordInput: string
- _parent: MainWindowViewModel
+ VerifyButton: ICommand
+ UserNameInput: String
+ PasswordInput: String
+ SignUpViewModel(parent: MainWindowViewModel)
- VerifyButtonClicked()
- CheckDatabase():Bool
- AddToDatabase()
ContinentViewModel
- _parent: MainWindowViewModel
+ ContinentName: Database
- _countryName: String
- _countryPopulation: String
- _countryLandArea: String
- _countryDensity: string
- _userInput: string
_ SearchButton: ICommand
_ MainMapButton: ICommand
+ CountryName: string
+ CountryPopulation: string
+ CountryLandArea: String
+ CountryDensity:String
+UserInput: String
+ContinentViewModel(parent: MainWindowViewModel)
+GetCountryInfo(UserInput: string)
+SearchButtonClicked()
+ MainMapButtonClicked()
QuizViewModel
- _parent: MainWindowViewModel
- _question: string
- _answer: string
+ Question: string
+ Answer: string
- _score: int
+ Score: int
+ random: Random
+ GenerateQuestionButton: ICommand
+ CheckAnswerButton: ICommand
+ FinishButton: ICommand
+QuizViewModel(Parent : MainWindowViewModel)
- GenerateQuestionButtonClicked()
- CheckAnswerButtonClicked()
- FinishButtonClicked()
GameMenuViewModel
- _parent: MainWindowViewModel
+ HomeButton: ICommand
+ PairsGameButton: ICommand
+ QuizButton: ICommand
+ ChosenQuestionBank: String
+GameMenuViewModel(parent: MainWindowViewModel)
- HomeButtonClicked()
- PairsGameButtonClicked()
- QuizButtonClicked()
SQLite
- Connection: SQLiteConnection
+Database()
- CreateConnection(): SQLiteConnection
+ Execute(sqlCommand: string)
+ CreateTable(sqlCommand: string)
+ InsertData(sqlCommand: string)
+ UpdateData(sqlCommand: string)
+ ReadData(sqlCommand: string): string
+ DeleteData(sqlCommand: string)
+ DropTable(sqlCommand: string)
PairsGameViewModel
- _parent: MainWindowViewModel
- _howManyPairs: string
+ CheckPairButton: ICommand
+ FinishButton: ICommand
- _score: int = 0
- _question: string
- _answer: string
- _pairFound: bool = false
- random: Random
- randomNum: int
+ HowManyPairs: string
+ PairFound: bool
+ Question: string
+ Answer: string
+ Score: int
- _textBlock1: string
+ TextBlock1: string
- _textBlock2: string
+ TextBlock2: string
+ PairsGameViewModel(parent: MainWindowViewModel)
+ GetQuestion():string
+ GetAnswer():string
- CheckPairs()
- FinishButtonClicked()
StartPageViewModel
- _parent: MainWindowViewModel
+ LoginButton: ICommand
+ SignUpButton: ICommand
+ StartPageViewModel(Parent: MainWindowViewModel)
- LoginButtonClicked()
- SignUpButtonClicked()
LoginPageViewModel
- _userNameInput: string = "Enter Your Username"
- _passwordInput: string = "Enter Your Password"
- _parent:MainWIndowViewModel
- LoginButton: ICommand
+UserNameInput: String
+ PasswordInput: String
+ LoginPageViewModel(parent: MainWIndowViewModel)
- LoginButtonClicked()
- CheckDatabase(): bool
UserStatsViewModel
- _parent: MainWindowViewModel
- HighScores: List<int>
- UsersHighScores: List<int>
- _userPercentile: int
- _quizHighScore: string
+ QuizHighScore: string
- _pairsHighScore: string
+ PairsHighScore: string
+ GetUserStatsButton: ICommand
+GetPercentileButton: ICommand
+ HomeButton: ICommand
+UserStatsViewModel(MainWindowViewModel Parent)
- HomeButtonClicked()
+ GetUserStats()
+ GetPercentile()
- PopulateList()
- SortList()
MergeSort
+MergeSort()
+ Sort(UnsortedList: List<int>): List<int>
- Merge(Left: List<int>, Right: List<int>): List<int>
Agregation
Compisition

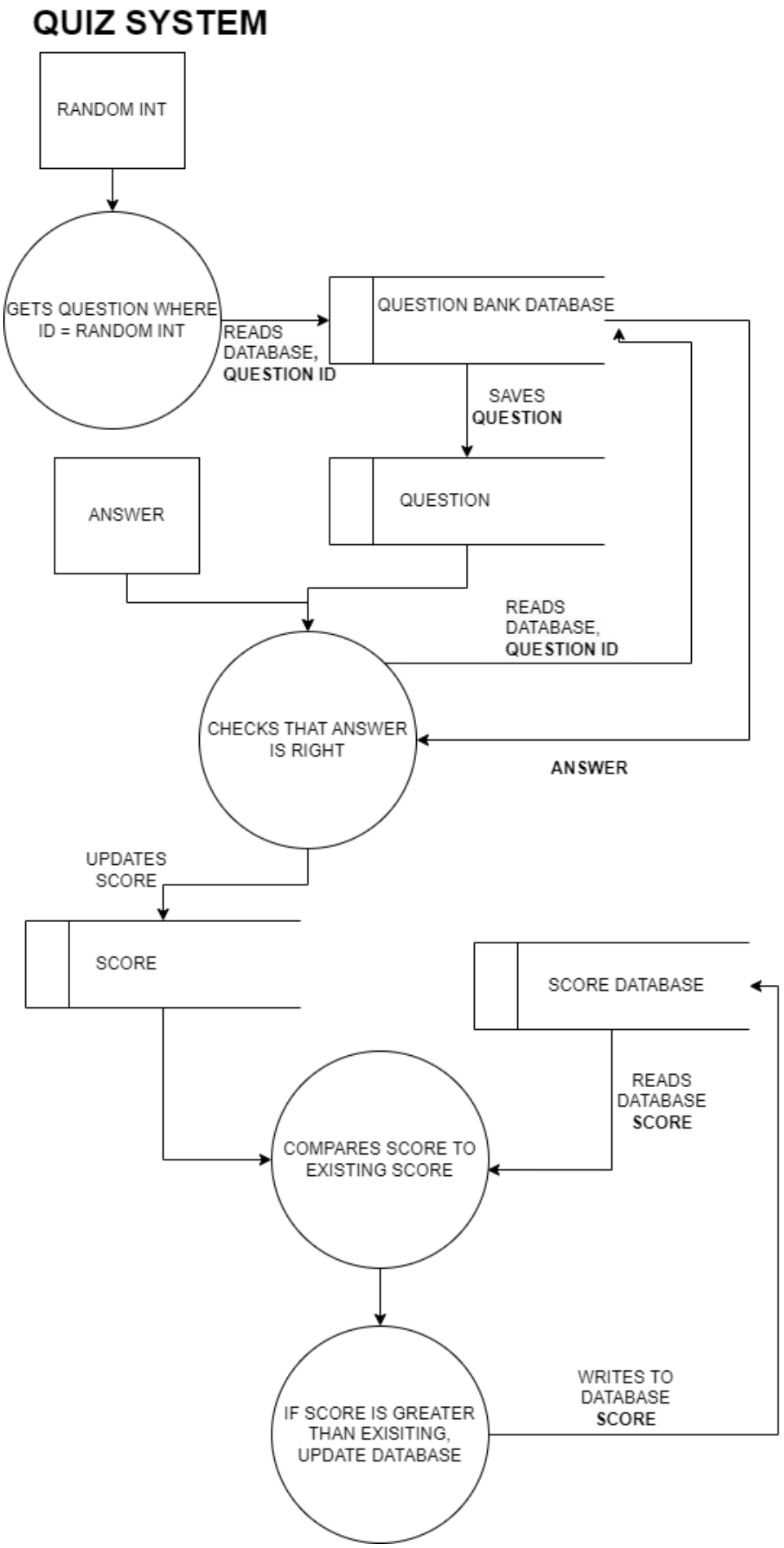

WorldMapViewModel
- _parent: MainWindowViewModel
+ AfricaButton: ICommand
+ AsiaButton: ICommand
+ EuropeButton: ICommand
+ NAmericaButton: ICommand
+ SAmericaButton: ICommand
+ OceaniaButton: ICommand
+ HomeButton: ICommand
+WorldMapViewModel(parent: MainWindowViewModel)
- AfricaButtonClicked()
- AsiaButtonClicked()
- EuropeButtonClicked()
- NAmericaButtonClicked()
- SouthAmericaButtonClicked()
- OceaniaButtonClicked()
- HomeButtonClicked()
HomePageViewModel
- _parent: MainWindowViewModel
+ MainMapButton: ICommand
+ GameMenuButton: ICommand
+ QuestionBankMenuButtonClicked: ICommand
+ UserStatsButton: ICommand
+GameMenuViewModel(parent: MainWindowViewModel)
- MainMapButtonClicked()
- GameMenuButtonClicked()
- QuestionBankMenuButtonClicked()
- UserStatsButtonClicked()
MainWindowViewModel
- _currentPage: ViewStates = ViewStates.StartPage
+ Database: Database
+ LoginPageViewModel : LoginPageViewModel
+ StartPageViewModel:StartPageViewModel
+ SignUpPageViewModel: SignUpPageViewModel
+ HomePageViewModel : HomePageViewModel
+ ContinentsMapViewModel : ContinentsMapViewModel
+ AfricaMapViewModel : AfricaMapViewModel
+ AsiaMapViewModel : AsiaMapViewModel
+ EuropeMapViewModel : EuropeMapViewModel
+ NAmericaMapViewModel: NAmericaMapViewModel
+ SAmericaMapViewModel: SAmericaMapViewModel
+ OceaniaMapViewModel: OceaniaMapViewModel
+ QuestionBankMenuViewModel: QuestionBankMenuViewModel
+ QuestionBankEditViewModel : QuestionBankEditViewModel
+ QuestionBankCreateViewModel: QuestionBankCreateViewModel
+ QuestionBankDeleteViewModel: QuestionBankDeleteViewModel
+ QuestionBankReadViewModel : QuestionBankReadViewModel
+ UserStatsViewModel : UserStatsViewModel
+ GameMenuViewModel: GameMenuViewModel
+QuizViewModel: QuizViewModel
+ PairsGameViewModel: PairsGameViewModel
+ UserID: int
+ CurrentPage: ViewStates
+MainWindowViewModel()
+ChangeToLoginPage()
+ChangeToSignUpPage()
+ChangeToHomePage()
+ChangeToMainMapPage()
+ChangeToAfrica()
+ChangeToAsia()
+ ChangeToEurope()
+ ChangeToNAmerica()
+ ChangeToSAmerica()
+ ChangeToOceania()
+ChangeToQuestionBankMenu()
+ChangeToCreateQuestionBank()
+ChangeToDeleteQuestionBank()
+ChangeToEditQuestionBank()
+ChangeToReadQuestionBank()
+ ChangeToUserStats()
+ ChangeToGameMenu()
+ChangeToPairsGame()
+ChangeToQuizPage()
+ PopulateDatabases()
SQLite
- Connection: SQLiteConnection
+Database()
- CreateConnection(): SQLiteConnection
+ Execute(sqlCommand: string)
+ CreateTable(sqlCommand: string)
+ InsertData(sqlCommand: string)
+ UpdateData(sqlCommand: string)
+ ReadData(sqlCommand: string): string
+ DeleteData(sqlCommand: string)
+ DropTable(sqlCommand: string)
Agregation
Agregation
Agregation
Compisition

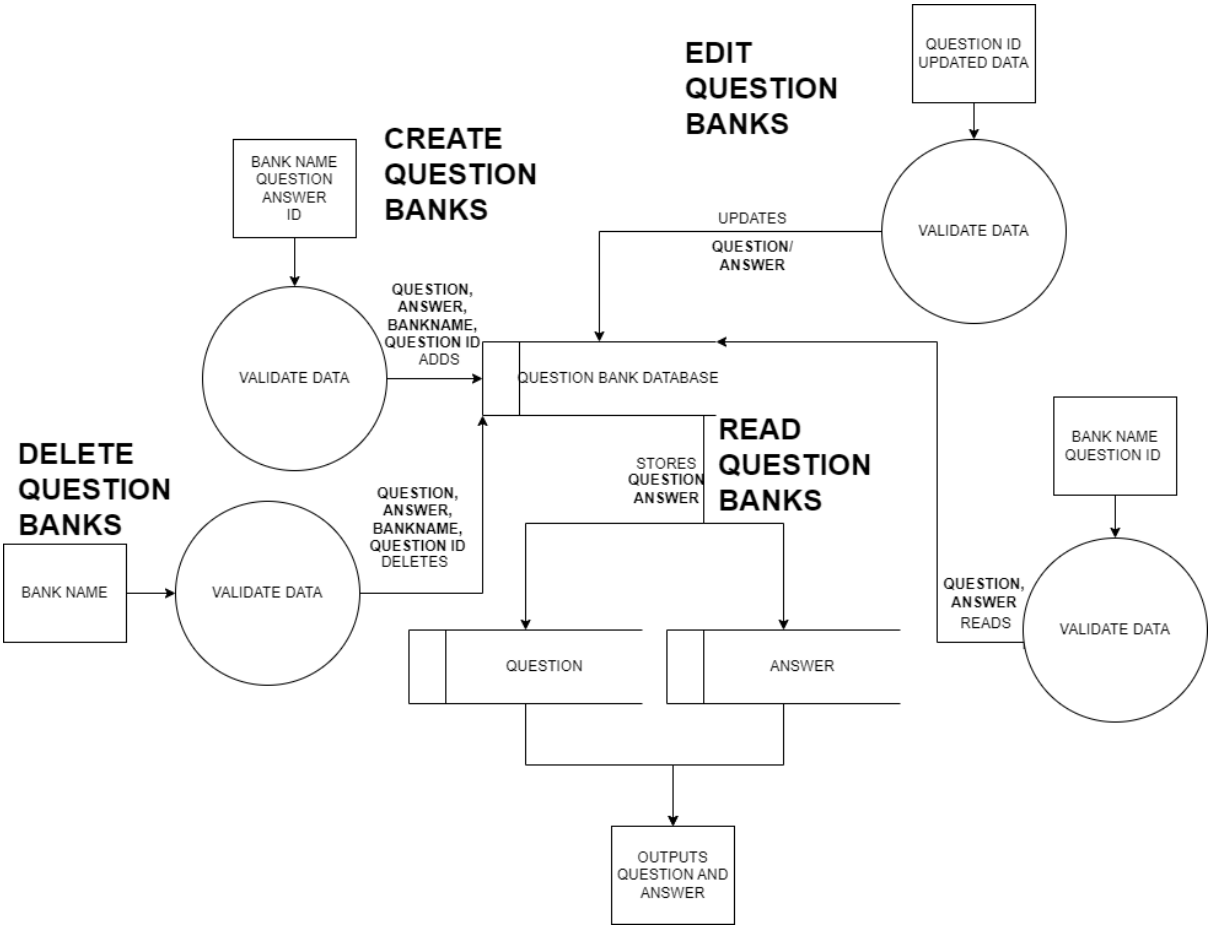

ObservableObject
+ PropertyChanged: PropertyChangedEventHandler
# OnPropertyChanged(propertyname: String)
# RaiseAndSetIfChanged(field: ref TProperty, value: TProperty, propertyName: string): Bool
SignupViewModel
- _userNameInput: string
- _passwordInput: string
- _parent: MainWindowViewModel
+ VerifyButton: ICommand
+ UserNameInput: String
+ PasswordInput: String
+ SignUpViewModel(parent: MainWindowViewModel)
- VerifyButtonClicked()
- CheckDatabase():Bool
- AddToDatabase()
ContinentViewModel
- _parent: MainWindowViewModel
+ ContinentName: Database
- _countryName: String
- _countryPopulation: String
- _countryLandArea: String
- _countryDensity: string
- _userInput: string
_ SearchButton: ICommand
_ MainMapButton: ICommand
+ CountryName: string
+ CountryPopulation: string
+ CountryLandArea: String
+ CountryDensity:String
+UserInput: String
+ContinentViewModel(parent: MainWindowViewModel)
+GetCountryInfo(UserInput: string)
+SearchButtonClicked()
+ MainMapButtonClicked()
QuizViewModel
- _parent: MainWindowViewModel
- _question: string
- _answer: string
+ Question: string
+ Answer: string
- _score: int
+ Score: int
+ random: Random
+ GenerateQuestionButton: ICommand
+ CheckAnswerButton: ICommand
+ FinishButton: ICommand
+QuizViewModel(Parent : MainWindowViewModel)
- GenerateQuestionButtonClicked()
- CheckAnswerButtonClicked()
- FinishButtonClicked()
Agregation
Agregation
Agregation
Agregation
Agregation
Agregation
PairsGameViewModel
- _parent: MainWindowViewModel
- _howManyPairs: string
+ CheckPairButton: ICommand
+ FinishButton: ICommand
- _score: int = 0
- _question: string
- _answer: string
- _pairFound: bool = false
- random: Random
- randomNum: int
+ HowManyPairs: string
+ PairFound: bool
+ Question: string
+ Answer: string
+ Score: int
- _textBlock1: string
+ TextBlock1: string
- _textBlock2: string
+ TextBlock2: string
+ PairsGameViewModel(parent: MainWindowViewModel)
+ GetQuestion():string
+ GetAnswer():string
- CheckPairs()
- FinishButtonClicked()
StartPageViewModel
- _parent: MainWindowViewModel
+ LoginButton: ICommand
+ SignUpButton: ICommand
+ StartPageViewModel(Parent: MainWindowViewModel)
- LoginButtonClicked()
- SignUpButtonClicked()
LoginPageViewModel
- _userNameInput: string = "Enter Your Username"
- _passwordInput: string = "Enter Your Password"
- _parent:MainWIndowViewModel
- LoginButton: ICommand
+UserNameInput: String
+ PasswordInput: String
+ LoginPageViewModel(parent: MainWIndowViewModel)
- LoginButtonClicked()
- CheckDatabase(): bool
UserStatsViewModel
- _parent: MainWindowViewModel
- HighScores: List<int>
- UsersHighScores: List<int>
- _userPercentile: int
- _quizHighScore: string
+ QuizHighScore: string
- _pairsHighScore: string
+ PairsHighScore: string
+ GetUserStatsButton: ICommand
+GetPercentileButton: ICommand
+ HomeButton: ICommand
+UserStatsViewModel(MainWindowViewModel Parent)
- HomeButtonClicked()
+ GetUserStats()
+ GetPercentile()
- PopulateList()
- SortList()
Compisition
MergeSort
+MergeSort()
+ Sort(UnsortedList: List<int>): List<int>
- Merge(Left: List<int>, Right: List<int>): List<int>

Data Flow Diagram




CONTINENT SYSTEM
COUNTRYNAME
RETRIVES COUNTRY'S INFO FROM THE DATABASE
COUNTRY NAME, COUNTRY STATS READS
CONTINENT DATABASE
STORES
COUNTRY NAME, COUNTRY STATS
COUNTRY INFORMATION
OUTPUTS INFORMATION

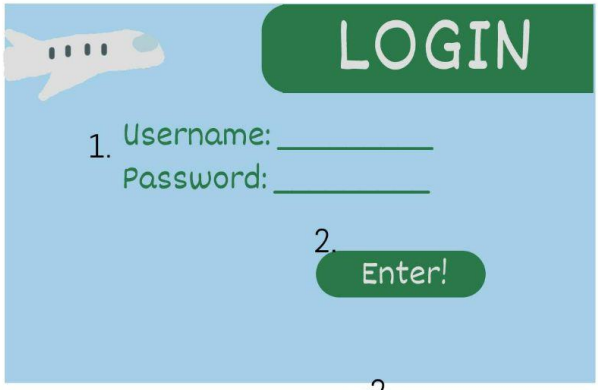

QUIZ SYSTEM
RANDOM INT
GETS QUESTION WHERE ID = RANDOM INT
READS DATABASE, QUESTION ID
QUESTION BANK DATABASE
SAVES QUESTION
ANSWER
QUESTION
READS DATABASE, QUESTION ID
CHECKS THAT ANSWER IS RIGHT
ANSWER
UPDATES SCORE
SCORE
SCORE DATABASE
READS DATABASE SCORE
COMPARES SCORE TO EXISTING SCORE
IF SCORE IS GREATER THAN EXISITING, UPDATE DATABASE
WRITES TO DATABASE SCORE



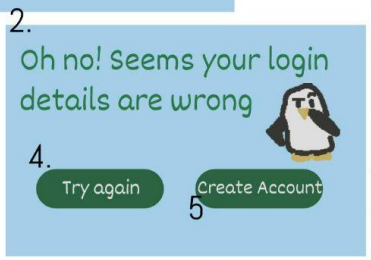
User Interface

START PAGE:
1. BUTTON - is linked to the LOGIN PAGE
2. BUTTON - is linked to the CREATE ACCOUNT PAGE

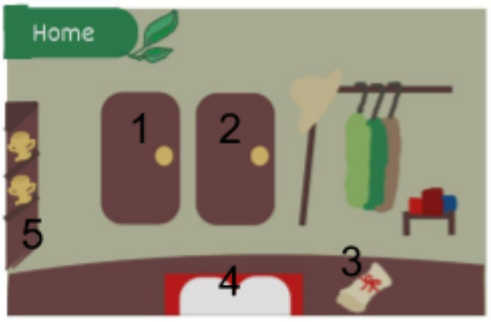
CREATE ACCOUNT PAGE:
1. TEXT BOXES - User inputs Username and Password
2. BUTTON - Adds details to database and is linked to the HOME PAGE



LOGIN PAGE:
1. TEXT BOXES - User inputs Username and Password
2. BUTTON - Checks database and is linked to the HOME PAGE

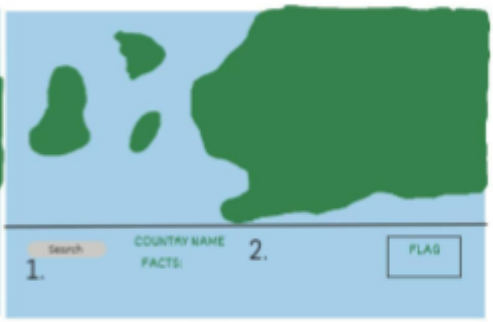


POP-UP WINDOWS
When the incorrect details are entered for the CREATE ACCOUNT PAGE (1) and LOGIN PAGE (2) with BUTTONS that link you back to that page (3/4) or to CREATE ACCOUNT (5) or LOGIN PAGE (6)



HOME PAGE:
1. BUTTON - linked to the QUIZ PAGE
2. BUTTON - linked to the MATCHING GAME
3. BUTTON - linked to the MAP
4. BUTTON - linked to USER DETAILS
5. BUTTON - linked to SCORES

MAP:
Interactive map that links to each continent page

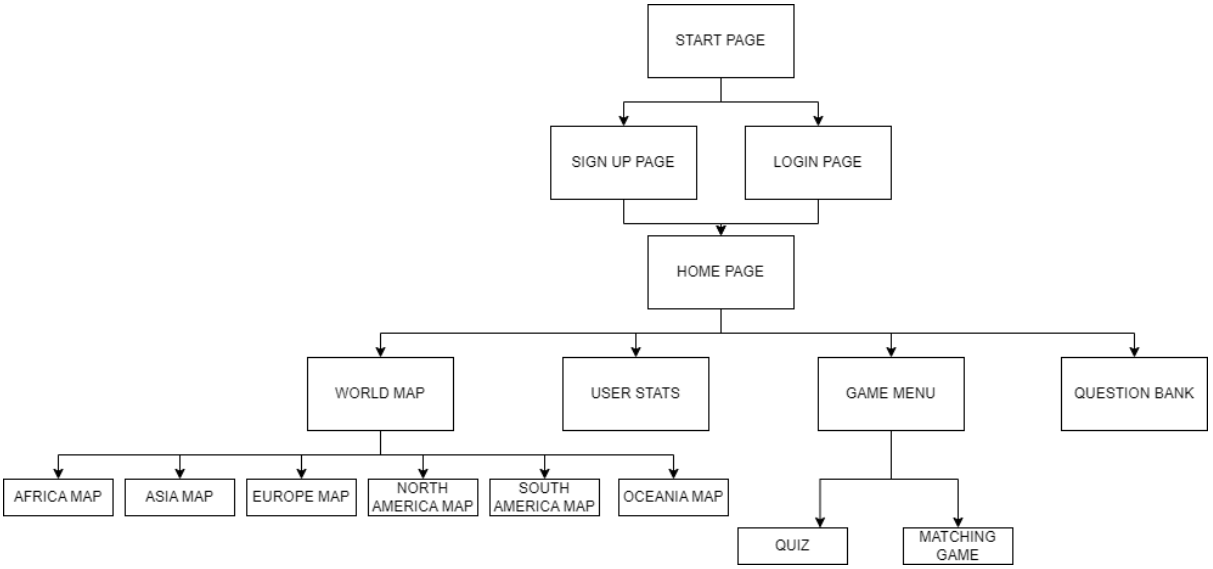


EXAMPLE CONTINTENT PAGE:
1. SEARCH BAR - Look for countries
2. OUTPUTS - infomation of the country

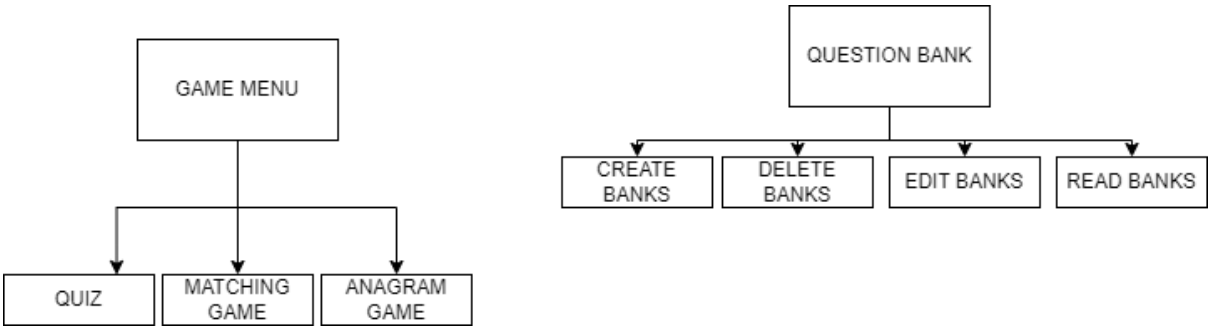
QUIZ PAGE:
Will randomly generate the questions, allow the user to enter their answer and update the score accordingly

MATCHING GAME:
Randomly generate Questions and Answers, and will allow the user to drag pairs together. Will keep track of the time and their points

PAGE NAVIGATION



Additions:



IPSO Chart

START PAGE

INPUT - User selects either the LOGIN button or the SIGN UP button	PROCESS - When pressed, each button calls a method to change the page by updating Current Page which is linked to the View States.
STORAGE - The Current View State on the Main Window is changed to the correct int value (e.g. 1 (Login) or 2 (Sign up))	OUTPUT - The page will change to either the LOGIN PAGE or the SIGN UP PAGE.

SIGN UP PAGE

INPUT - Username - Password	PROCESS - When the User inputs details, they will be validated, ensuring that they are a unique user in the database and they are at least 5 characters long.

	- If it is valid it will update the value of the current page.
STORAGE - If it passes the criteria, they will be saved to the User Login Database, and assigned a unique UserID, as well as creating a field in the HighScores Database which links to their ID. - Current Page will change to the corresponding viewstates value.	OUTPUT - If the user already exists, it will provide the user the option to go to the LOGIN PAGE, or to retry again. - If their sign up is successful, it will change to the HOME PAGE.

LOGIN PAGE

INPUT - Username - Password	PROCESS - When the User inputs details, they will be validated, ensuring that they are an existing user in the database and that their password and username matches. - If it is valid it will update the value of the current page.
STORAGE - If it passes the criteria,their UserID will be saved, to make accessing their data in other databases easier. - Current Page will change to the corresponding viewstates value.	OUTPUT - If the login fails it will provide the user the option to go to the SIGN UP PAGE, or to retry again. - If their login is successful, it will change to the HOME PAGE.

HOME PAGE

INPUT - User selects 1 of 4 buttons: MAP button, QUESTION BANK MENU button, GAME MENU button, USER STATS button.	PROCESS - When pressed, each button calls a method to change the page by updating Current Page which is linked to the View States.
STORAGE - The Current View State on the Main Window is changed to the correct int value (e.g. 1 (Login) or 2 (Sign up))	OUTPUT - The page will change to either the MAP PAGE, QUESTION BANK MENU, GAME MENU or USER STATS PAGE.

MAIN MAP PAGE

INPUT - User selects 1 of 6 buttons: AFRICA button, ASIA button, EUROPE button, NORTHAMERICA button, SOUTHAMERICA button,	PROCESS - When pressed, each button calls a method to change the page by updating Current Page which is linked to the View States.

OCEANIA button.	
STORAGE - The Current View State on the Main Window is changed to the correct int value (e.g. 1 (Login) or 2 (Sign up))	OUTPUT - The page will change to either the AFRICA PAGE, ASIA PAGE, EUROPE PAGE, NORTHAMERICA PAGE, SOUTHAMERICA PAGE or OCEANIA PAGE.

CONTINENT PAGES (Same for all)

INPUT - SelectedCountry - HOME BUTTON	PROCESS - When the User inputs the selected country, it will be validated to ensure it is in the database. - If HOME BUTTON is pressed, a method is called to change the page by updating the Current Page which is linked to the View States.
STORAGE - If it is a valid country, the database is read and the information about the country is saved to a variable to make it easier to output. - HOME BUTTON: The Current View State on the Main Window is changed to the correct int value (e.g. 1 (Login) or 2 (Sign up))	OUTPUT - If the country is valid, then the country's information will be displayed to the screen. - If not, a message will appear asking the user to enter a valid country. - HOME BUTTON: The page will change to the HOME PAGE.

QUESTION BANK MENU

INPUT - User selects 1 of 5 buttons: CREATE button, DELETE button, EDIT button, READ button or HOME button.	PROCESS - When pressed, each button calls a method to change the page by updating Current Page which is linked to the View States.
STORAGE - The Current View State on the Main Window is changed to the correct int value (e.g. 1 (Login) or 2 (Sign up))	OUTPUT - The page will change to either the CREATE BANK PAGE, DELETE BANK PAGE, EDIT BANK PAGE, READ BANK PAGE or HOME PAGE.

CREATE QUESTION BANK PAGE

INPUT - QuestionBankName - Question - Answer	PROCESS - All inputs cannot be null, if they all contain content then it is valid. - If HOME BUTTON is pressed, a

- HOME BUTTON	method is called to change the page by updating the Current Page which is linked to the View States.
STORAGE - If it passes the criteria, the Bank name, the question and answer will be saved to the Question Bank Database, alongside the User's ID, as well as being assigned a unique Question ID. - HOME BUTTON: The Current View State on the Main Window is changed to the correct int value (e.g. 1 (Login) or 2 (Sign up))	OUTPUT - If the input is invalid, a message will appear prompting the user to re-enter the incorrect values. - If the HOME BUTTON is pressed the page will change to the HOME PAGE.

DELETE QUESTION BANK PAGE

INPUT - QuestionBankName - HOME BUTTON	PROCESS - Question Bank Name cannot be null and must exist in the Question Bank Database, belonging to the User - if all of these conditions it is a valid input. - If HOME BUTTON is pressed, a method is called to change the page by updating the Current Page which is linked to the View States.
STORAGE - If it passes the criteria, ALL fields with the Bank name that belong to that user will be deleted from the database. - HOME BUTTON: The Current View State on the Main Window is changed to the correct int value (e.g. 1 (Login) or 2 (Sign up))	OUTPUT - If the input is invalid, a message will appear prompting the user to re-enter the incorrect values. - If the input is valid, a message will appear prompting the user to confirm the deletion. - If the HOME BUTTON is pressed the page will change to the HOME PAGE.

EDIT QUESTION BANK PAGE

INPUT - QuestionBankName - SelectedQuestion - NewQuestion - NewAnswer - HOME BUTTON	PROCESS - A valid Question Bank Name from the database must be entered, and a valid question from that Bank must also be entered. - The user may then enter at a New Question, New Answer or both. - If HOME BUTTON is pressed, a method is called to change the page by updating the Current Page which is linked to the View States.

STORAGE - If it passes the criteria, the question/answer will be updated in the database, where the Bank name matches the selected bank name and belongs to the User. - HOME BUTTON: The Current View State on the Main Window is changed to the correct int value (e.g. 1 (Login) or 2 (Sign up))	OUTPUT - If the input is invalid, a message will appear prompting the user to re-enter the incorrect values. - If the HOME BUTTON is pressed the page will change to the HOME PAGE.

READ QUESTION BANK PAGE

INPUT - QuestionBankName - SelectedQuestion - HOME BUTTON	PROCESS - A valid Question Bank Name from the database must be entered, and a valid question from that Bank must also be entered. - If HOME BUTTON is pressed, a method is called to change the page by updating the Current Page which is linked to the View States.
STORAGE - If it is a valid input, the database is read and the answer to the question is saved to a variable to make it easier to output. - HOME BUTTON: The Current View State on the Main Window is changed to the correct int value (e.g. 1 (Login) or 2 (Sign up))	OUTPUT - If all the inputs are valid, then the answer will be displayed to the screen. - If not, a message will appear asking the user to correct their incorrect inputs. - HOME BUTTON: The page will change to the HOME PAGE.

GAME MENU PAGE

INPUT - User selects 1 of *4* buttons: QUIZ button, PAIRS GAME button, HOME button, *Word Scramble* button.	PROCESS - When pressed, each button calls a method to change the page by updating Current Page which is linked to the View States.
STORAGE - The Current View State on the Main Window is changed to the correct int value (e.g. 1 (Login) or 2 (Sign up))	OUTPUT - The page will change to either the QUIZ PAGE, PAIRS GAME PAGE, HOME PAGE or *Word Scramble* PAGE.

QUIZ PAGE

INPUT - UserAnswer - HOME BUTTON	PROCESS - A random question is generated from the database and the correct

	answer is compared to the User's answer. - If HOME BUTTON is pressed, a method is called to change the page by updating the Current Page which is linked to the View States.
STORAGE - If the answer is correct, then a bool value is assigned true and a score is incremented by 1. - HOME BUTTON: The Current View State on the Main Window is changed to the correct int value (e.g. 1 (Login) or 2 (Sign up))	OUTPUT - If the user is correct a message displays, which shows their score increasing. - If not, a message will appear showing the correct answer. - HOME BUTTON: The page will change to the HOME PAGE.

PAIRS GAME PAGE

INPUT - Text1 - Text2 - HOME BUTTON	PROCESS - The user drags various text blocks to match the correct values, the content of each block is compared to the database and is validated accordingly. - If HOME BUTTON is pressed, a method is called to change the page by updating the Current Page which is linked to the View States.
STORAGE - If there is a match, then a bool value is assigned true and a score is incremented by 1. - HOME BUTTON: The Current View State on the Main Window is changed to the correct int value (e.g. 1 (Login) or 2 (Sign up))	OUTPUT - If there is a match, a message displays, which shows their score increasing. - If not, a message will appear showing the correct answer. - HOME BUTTON: The page will change to the HOME PAGE.

USER STATS PAGE

INPUT - SelectedHighScore - HOME BUTTON	PROCESS - A valid category from the HighScore Database must be selected and then compared to the other user's scores in order to calculate the user's percentile. - If HOME BUTTON is pressed, a method is called to change the page by updating the Current Page which is linked to the View States.
STORAGE - Once the user has selected a valid	OUTPUT - Will display the user's highscores to

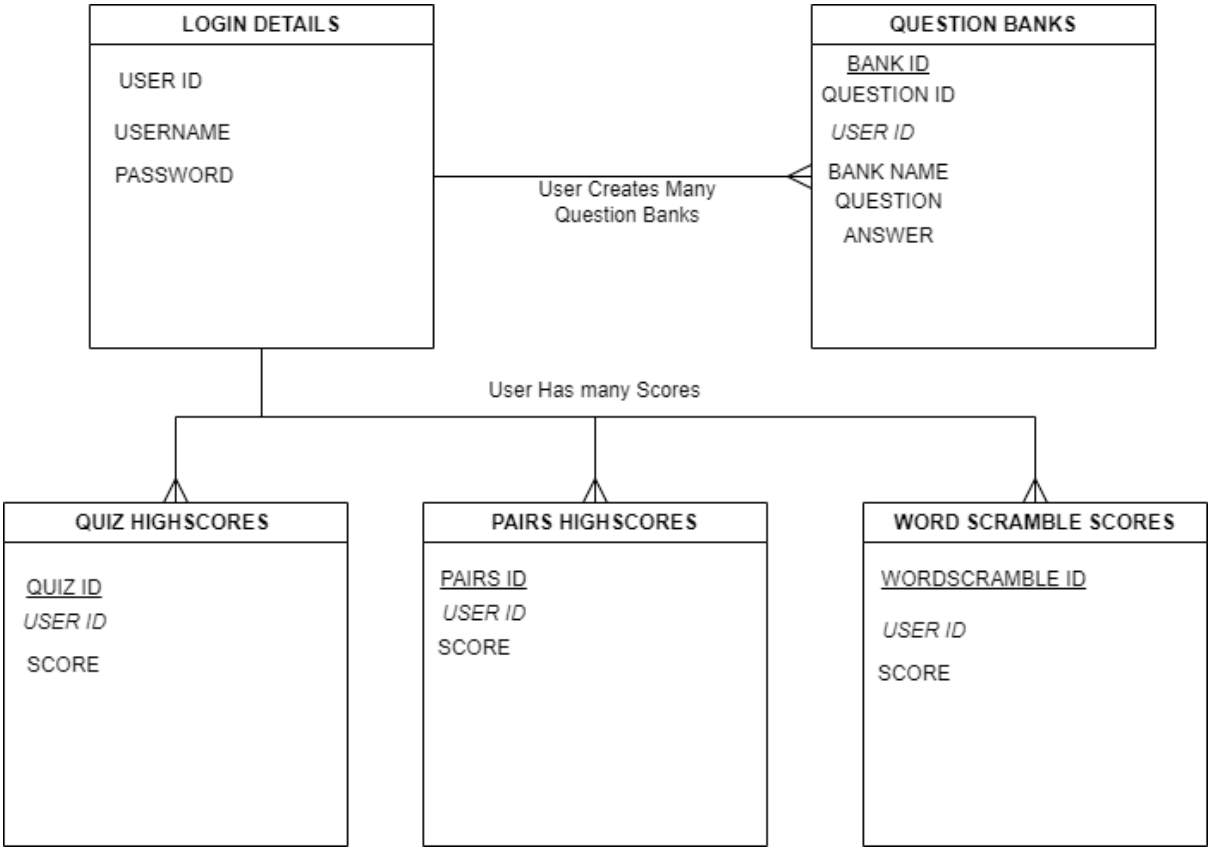
category, the percentile will be stored in order to make it easier to output the information. - HOME BUTTON: The Current View State on the Main Window is changed to the correct int value (e.g. 1 (Login) or 2 (Sign up))	screen. - If a valid category is selected, then the percentile of the user is displayed. - HOME BUTTON: The page will change to the HOME PAGE.

LATER ADDITION:

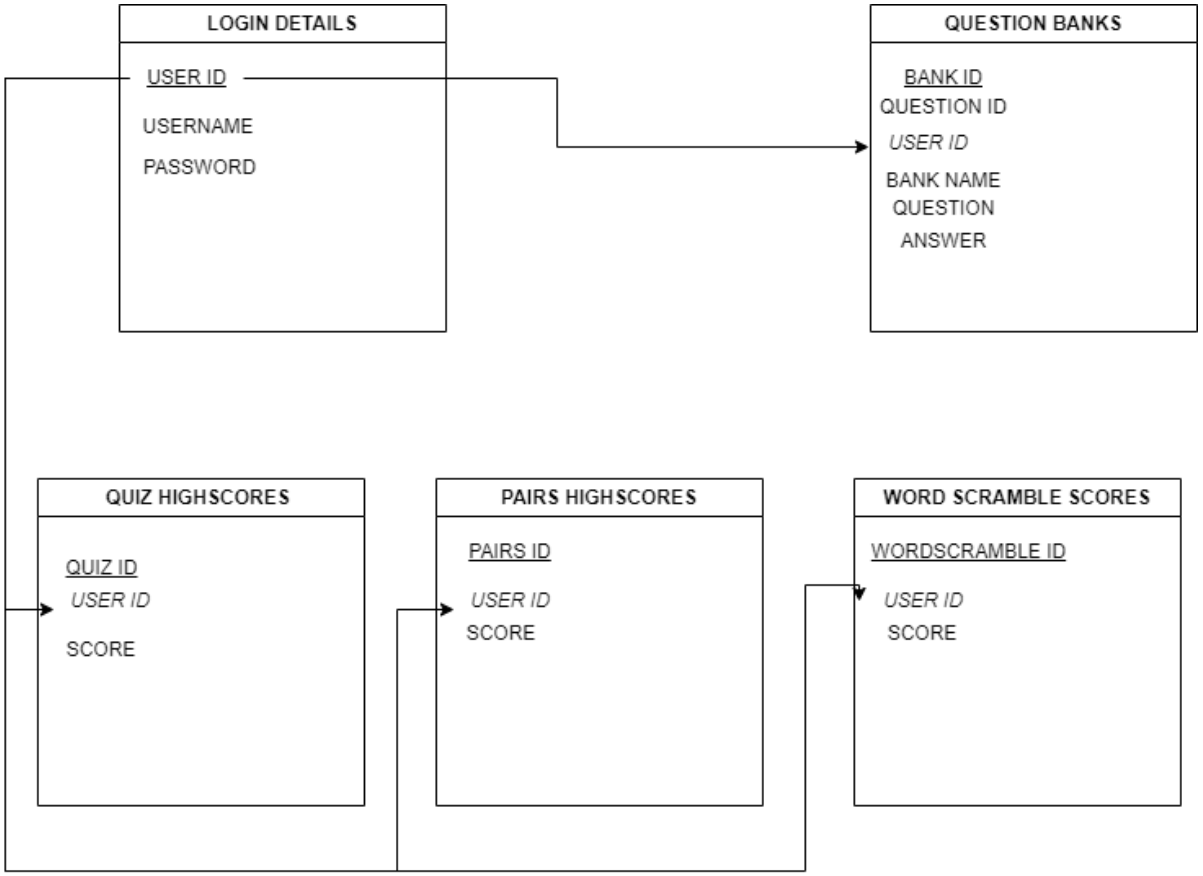
WORD SCRAMBLE PAGE

INPUT - UserAnswer - HOME BUTTON	PROCESS - A random country is generated from a random continent database and then is scrambled. - The correct answer is compared to the User's answer. - If HOME BUTTON is pressed, a method is called to change the page by updating the Current Page which is linked to the View States.
STORAGE - If the answer is correct, then a bool value is assigned true and a score is incremented by 1. - HOME BUTTON: The Current View State on the Main Window is changed to the correct int value (e.g. 1 (Login) or 2 (Sign up))	OUTPUT - If the user is correct a message displays, which shows their score increasing. - If not, a message will appear showing the correct answer. - HOME BUTTON: The page will change to the HOME PAGE.

Entity-Relationship Diagram



Relational Diagram



Normalised Data Structures

FOREIGN KEY PRIMARY KEY

LOGIN DETAILS

Field Name Data	Data Type	Validation	Example
UserID (PRIMARY)	Integer	Assigned automatically	1
Username	String	Cannot already exist, cannot be null	User1
Password	String	Must be at least 5 characters long	Pass1

QUESTION BANKS

Field Name Data	Data Type	Validation	Example
Bank ID (PRIMARY)	Integer	Assigned automatically	1

UserID	Integer	Foreign Key from User Details	1
BankName	String	Cannot be null	Practice
QuestionID	Integer	Assigned automatically	1
Question	String	Cannot be null	Bonjour
Answer	String	Cannot be null	Hello

QUIZ HIGH SCORE

Field Name Data	Data Type	Validation	Example
QuizID (PRIMARY)	Integer	Assigned automatically	2
UserID	Integer	Foreign Key from User Details	1
Score	Integer	Cannot be null	10

PAIRS HIGH SCORE

Field Name Data	Data Type	Validation	Example
PairsID (PRIMARY)	Integer	Assigned automatically	2
UserID	Integer	Foreign Key from User Details	1
Score	Integer	Cannot be null	10

WORD SCRAMBLE HIGH SCORE

Field Name Data	Data Type	Validation	Example
WordScrambleID (PRIMARY)	Integer	Assigned automatically	2
UserID	Integer	Foreign Key from User Details	1
Score	Integer	Cannot be null	10

CONTINENT - Separate Database for each Continent

Field Name Data	Data Type	Validation	Example
ID (PRIMARY)	Integer	Assigned automatically	1
CountryName	String	Cannot be null and is automatically assigned unless it already exists.	Angola

Population	String	Cannot be null and is automatically assigned unless it already exists.	32866272
LandArea	String	Cannot be null and is automatically assigned unless it already exists.	1246700
Density	String	Cannot be null and is automatically assigned unless it already exists.	26
Flag	**File path**	Cannot be null and is automatically assigned unless it already exists.	**"C:\...\NEA\Images\AngolaFlag.png"**

Flag is now removed

Algorithms

MERGE SORT

```
SORT(List)
IF(List.COUNT > 1)
        INT mid = List.COUNT / 2
        FOR i = 0 TO mid
                LeftList.ADD(List[i])
        NEXT i
        FOR i = mid TO List.COUNT
                RightList.ADD(List[i])
        SORT(LeftList)
        SORT(RightList)
        MERGE(LeftList, RightList)
RETURN List

MERGE(LeftList, RightList)
INT LeftPosition = 0
INT RightPostion = 0
WHILE LeftPosition < LeftList.COUNT AND RightPosition < RightList.COUNT
        IF LeftList[LeftPosition] < RightList[RightPosition]
                Result.ADD(LeftList[LeftPosition])
                LeftPostition ++
        ELSE IF LeftList[LeftPosition] > RightList[RightPosition]
                Result.ADD(RightList[RightPosition])
                RightPosition ++
FOR i = LeftPosition TO LeftList.COUNT
        Result.ADD(Left[i])
FOR i = RightPosition TO RightList.COUNT
        Result.ADD(Right[i])
RETURN Result
```

PERCENTILE CALCULATE

```
CALCULATE()
Highscores: LIST
UserID: int
UserHighScore: int
Position: int
FOR i = 0 TO (SELECT COUNT(USERID) FROM HighScores)
        Highscores.ADD(SELECT MAX(UserScore) From HighScores WHERE USERID = i)
        IF UserID == i THEN
                UserHighscore = SELECT MAX(UserScore) From HighScores WHERE
USERID = i
NEXT i
MERGESORT(Highscores)
FOR i = 1 TO Highscores.Count
        IF UserHighScore == Highscores[i] THEN
                Position = i
                Break Loop
NEXT i
INT Percentile = (Position/ Highscores.Count) * 100

RETURN Percentile
```

PERMUTATION OF A STRING

```
PERMUTE(char[] Word, int start, int end, ref List<string> list)
IF start = end
        list.ADD(Word.ToString)
ELSE
        FOR i = start TO end
                SWAP(Word[start], Word[i])
                PERMUTE(Word, start + 1, end, list)
                SWAP(Word[start], Word[j])
        NEXT i

SWAP(a, b)
CHAR temp = a
a = b
b = temp
```

System Security and Integrity of Data

Despite there being little personal information stored by the application, it seemed worthwhile to explore the hashing capabilities within C#, in order to hash the users' passwords. The mathematical process to make an effective hash algorithm can be taxing and due to the nature of the project, it is not a good use of time to create a unique one, when an inbuilt one would work just as effectively. The inbuilt System.Security.Cryptography with the class SHA256Managed seems to be an effective solution, that will be considered.

File Structure and Organisation, Record Structure and Processing

Text files will be used to automatically populate the necessary databases, such as the different continent's databases and the default questions. As well as this the continents will each have their own image, which equally needs to be stored within the project.

The text files will be stored by having each line containing all the relevant information for that record separated by columns.

E.G.) AFRICA TEXT FILE
Angola, 32866272, 1246700, 26
Which equals: CountryName, Population, Land Area and Density.

E.G.) DEFAULT QUESTION TEXT FILE
France's Capital, Paris
Equals: Question, Answer

Technical Solution

Code

INSTANCES OF SQL:

- SQLite: P.145 - 147
- Main Window: P.60, P.63 - 64
- Login Page: P. 72 - 73
- Sign Up Page: 68 -70
- Africa (example of all continents): P. 82 - 3
- Quiz: P.113 - 4
- Pairs Game: P.110 - 1
- CheckPairCommand: P. 151
- Word Scramble: P.117, 119
- Question Bank Create: P.124 - 5
- Question Bank Delete: P. 127 - 8
- Question Bank Edit: P. 131 - 3
- Question Bank Read: P. 136 - 7

EXAMPLE OF POLYMORPHISM:

- Check Pair Command: P.150

EXAMPLE OF MERGE SORT:

- MergeSort Class: P.148 - 150
- User Stats Page: P. 143

EXAMPLE OF RECURSIVE ALGORITHM:

- Word Scramble: 119 - 20

MAIN WINDOW CODE:

XAML:

```
<Window x:Class="NEA_Project.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:NEA_Project"
        xmlns:page="clr-namespace:NEA_Project.Pages"
        xmlns:viewmodel="clr-namespace:NEA_Project.ViewModels"
        xmlns:constant="clr-namespace:NEA_Project.Constants"
        d:DataContext="{d:DesignInstance Type=viewmodel:MainWindowViewModel}"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="800">
       <Grid>
    <Grid.ColumnDefinitions>
       <ColumnDefinition Width="1*"/>
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
       <RowDefinition Height="1*"/>
    </Grid.RowDefinitions>

<!-- All the UserControl definitions represent each page and allow them to be displayed on the main window, it also sets their binding to the ViewStates and the data context to their respective view models, outline for UserControl, designed by me - just copied and pasted-->
    <UserControl Grid.Column="0" Grid.Row="0">
```

```
        <UserControl.Style>
         <Style TargetType="UserControl">
                  <Setter Property="Visibility" Value="Collapsed" />
                  <Style.Triggers>
                  <DataTrigger Binding="{Binding Path=CurrentPage}" Value="{x:Static
constant:ViewStates.StartPage}">
                  <Setter Property="Visibility" Value="Visible" />
                  </DataTrigger>
                  </Style.Triggers>
         </Style>
        </UserControl.Style>

        <page:StartPage DataContext="{Binding StartPageViewModel}"/>
      </UserControl>

      <UserControl Grid.Column="0" Grid.Row="0">
        <UserControl.Style>
         <Style TargetType="UserControl">
                  <Setter Property="Visibility" Value="Collapsed" />
                  <Style.Triggers>
                  <DataTrigger Binding="{Binding Path=CurrentPage}" Value="{x:Static
constant:ViewStates.LoginPage}">
                  <Setter Property="Visibility" Value="Visible" />
                  </DataTrigger>
                  </Style.Triggers>
         </Style>
        </UserControl.Style>

        <page:LoginPage DataContext="{Binding LoginPageViewModel}"/>
      </UserControl>

      <UserControl Grid.Column="0" Grid.Row="0">
        <UserControl.Style>
         <Style TargetType="UserControl">
                  <Setter Property="Visibility" Value="Collapsed" />
                  <Style.Triggers>
                  <DataTrigger Binding="{Binding Path=CurrentPage}" Value="{x:Static
constant:ViewStates.SignUpPage}">
                  <Setter Property="Visibility" Value="Visible" />
                  </DataTrigger>
                  </Style.Triggers>
         </Style>
        </UserControl.Style>

        <page:SignUpPage DataContext="{Binding SignUpPageViewModel}"/>
      </UserControl>

      <UserControl Grid.Column="0" Grid.Row="0">
        <UserControl.Style>
         <Style TargetType="UserControl">
                  <Setter Property="Visibility" Value="Collapsed" />
                  <Style.Triggers>
                  <DataTrigger Binding="{Binding Path=CurrentPage}" Value="{x:Static
constant:ViewStates.HomePage}">
                  <Setter Property="Visibility" Value="Visible" />
                  </DataTrigger>
```

```
                </Style.Triggers>
            </Style>
        </UserControl.Style>

        <page:HomePage DataContext="{Binding HomePageViewModel}"/>
    </UserControl>

    <UserControl Grid.Column="0" Grid.Row="0">
        <UserControl.Style>
            <Style TargetType="UserControl">
                <Setter Property="Visibility" Value="Collapsed" />
                <Style.Triggers>
                <DataTrigger Binding="{Binding Path=CurrentPage}" Value="{x:Static
constant:ViewStates.ContinentsMap}">
                <Setter Property="Visibility" Value="Visible" />
                </DataTrigger>
                </Style.Triggers>
            </Style>
        </UserControl.Style>

        <page:ContinentsMap DataContext="{Binding ContinentsMapViewModel}"/>
    </UserControl>

    <UserControl Grid.Column="0" Grid.Row="0">
        <UserControl.Style>
            <Style TargetType="UserControl">
                <Setter Property="Visibility" Value="Collapsed" />
                <Style.Triggers>
                <DataTrigger Binding="{Binding Path=CurrentPage}" Value="{x:Static
constant:ViewStates.AfricaMap}">
                <Setter Property="Visibility" Value="Visible" />
                </DataTrigger>
                </Style.Triggers>
            </Style>
        </UserControl.Style>

        <page:AfricaMap DataContext="{Binding AfricaMapViewModel}"/>
    </UserControl>

    <UserControl Grid.Column="0" Grid.Row="0">
        <UserControl.Style>
            <Style TargetType="UserControl">
                <Setter Property="Visibility" Value="Collapsed" />
                <Style.Triggers>
                <DataTrigger Binding="{Binding Path=CurrentPage}" Value="{x:Static
constant:ViewStates.AsiaMap}">
                <Setter Property="Visibility" Value="Visible" />
                </DataTrigger>
                </Style.Triggers>
            </Style>
        </UserControl.Style>

        <page:AsiaMap DataContext="{Binding AsiaMapViewModel}"/>

    </UserControl>
```

```
        <UserControl Grid.Column="0" Grid.Row="0">
            <UserControl.Style>
             <Style TargetType="UserControl">
                        <Setter Property="Visibility" Value="Collapsed" />
                        <Style.Triggers>
                        <DataTrigger Binding="{Binding Path=CurrentPage}" Value="{x:Static
constant:ViewStates.EuropeMap}">
                        <Setter Property="Visibility" Value="Visible" />
                        </DataTrigger>
                        </Style.Triggers>
             </Style>
            </UserControl.Style>

            <page:EuropeMap DataContext="{Binding EuropeMapViewModel}"/>
             </UserControl>

        <UserControl Grid.Column="0" Grid.Row="0">
            <UserControl.Style>
             <Style TargetType="UserControl">
                        <Setter Property="Visibility" Value="Collapsed" />
                        <Style.Triggers>
                        <DataTrigger Binding="{Binding Path=CurrentPage}" Value="{x:Static
constant:ViewStates.NAmericaMap}">
                        <Setter Property="Visibility" Value="Visible" />
                        </DataTrigger>
                        </Style.Triggers>
             </Style>
            </UserControl.Style>

            <page:NAmericaMap DataContext="{Binding NAmericaMapViewModel}"/>
        </UserControl>

        <UserControl Grid.Column="0" Grid.Row="0">
            <UserControl.Style>
             <Style TargetType="UserControl">
                        <Setter Property="Visibility" Value="Collapsed" />
                        <Style.Triggers>
                        <DataTrigger Binding="{Binding Path=CurrentPage}" Value="{x:Static
constant:ViewStates.SAmericaMap}">
                        <Setter Property="Visibility" Value="Visible" />
                    </DataTrigger>
                        </Style.Triggers>
             </Style>
            </UserControl.Style>

            <page:SAmericaMap DataContext="{Binding SAmericaMapViewModel}"/>
        </UserControl>

        <UserControl Grid.Column="0" Grid.Row="0">
            <UserControl.Style>
             <Style TargetType="UserControl">
                        <Setter Property="Visibility" Value="Collapsed" />
                        <Style.Triggers>
                        <DataTrigger Binding="{Binding Path=CurrentPage}" Value="{x:Static
constant:ViewStates.OceaniaMap}">
                        <Setter Property="Visibility" Value="Visible" />
```

```
                </DataTrigger>
                </Style.Triggers>
            </Style>
        </UserControl.Style>

        <page:OceaniaMap DataContext="{Binding OceaniaMapViewModel}"/>
    </UserControl>

    <UserControl Grid.Column="0" Grid.Row="0">
        <UserControl.Style>
            <Style TargetType="UserControl">
                <Setter Property="Visibility" Value="Collapsed" />
                <Style.Triggers>
                <DataTrigger Binding="{Binding Path=CurrentPage}" Value="{x:Static
constant:ViewStates.QuestionBankMenu}">
                <Setter Property="Visibility" Value="Visible" />
                </DataTrigger>
                </Style.Triggers>
            </Style>
        </UserControl.Style>

        <page:QuestionBankMenu DataContext="{Binding QuestionBankMenuViewModel}"/>
    </UserControl>

    <UserControl Grid.Column="0" Grid.Row="0">
        <UserControl.Style>
            <Style TargetType="UserControl">
                <Setter Property="Visibility" Value="Collapsed" />
                <Style.Triggers>
                <DataTrigger Binding="{Binding Path=CurrentPage}" Value="{x:Static
constant:ViewStates.QuestionBankEdit}">
                <Setter Property="Visibility" Value="Visible" />
                </DataTrigger>
                </Style.Triggers>
            </Style>
        </UserControl.Style>

        <page:QuestionBankEditPage DataContext="{Binding QuestionBankEditViewModel}"/>
    </UserControl>

    <UserControl Grid.Column="0" Grid.Row="0">
        <UserControl.Style>
            <Style TargetType="UserControl">
                <Setter Property="Visibility" Value="Collapsed" />
                <Style.Triggers>
                <DataTrigger Binding="{Binding Path=CurrentPage}" Value="{x:Static
constant:ViewStates.QuestionBankCreate}">
                <Setter Property="Visibility" Value="Visible" />
                </DataTrigger>
                </Style.Triggers>
            </Style>
        </UserControl.Style>

        <page:QuestionBankCreatePage DataContext="{Binding QuestionBankCreateViewModel}"/>
    </UserControl>
```

```
        <UserControl Grid.Column="0" Grid.Row="0">
            <UserControl.Style>
                <Style TargetType="UserControl">
                        <Setter Property="Visibility" Value="Collapsed" />
                        <Style.Triggers>
                        <DataTrigger Binding="{Binding Path=CurrentPage}" Value="{x:Static
constant:ViewStates.QuestionBankDelete}">
                         <Setter Property="Visibility" Value="Visible" />
                        </DataTrigger>
                        </Style.Triggers>
                </Style>
            </UserControl.Style>

            <page:QuestionBankDeletePage DataContext="{Binding QuestionBankDeleteViewModel}"/>
        </UserControl>

        <UserControl Grid.Column="0" Grid.Row="0">
            <UserControl.Style>
                <Style TargetType="UserControl">
                        <Setter Property="Visibility" Value="Collapsed" />
                        <Style.Triggers>
                        <DataTrigger Binding="{Binding Path=CurrentPage}" Value="{x:Static
constant:ViewStates.QuestionBankRead}">
                        <Setter Property="Visibility" Value="Visible" />
                        </DataTrigger>
                        </Style.Triggers>
                </Style>
            </UserControl.Style>

            <page:QuestionBankReadPage DataContext="{Binding QuestionBankReadViewModel}"/>
        </UserControl>

        <UserControl Grid.Column="0" Grid.Row="0">
            <UserControl.Style>
                <Style TargetType="UserControl">
                        <Setter Property="Visibility" Value="Collapsed" />
                        <Style.Triggers>
                        <DataTrigger Binding="{Binding Path=CurrentPage}" Value="{x:Static
constant:ViewStates.UserStatsPage}">
                        <Setter Property="Visibility" Value="Visible" />
                        </DataTrigger>
                        </Style.Triggers>
                </Style>
            </UserControl.Style>

            <page:UserStatsPage DataContext="{Binding UserStatsViewModel}"/>
        </UserControl>

        <UserControl Grid.Column="0" Grid.Row="0">
            <UserControl.Style>
                <Style TargetType="UserControl">
                        <Setter Property="Visibility" Value="Collapsed" />
                        <Style.Triggers>
                        <DataTrigger Binding="{Binding Path=CurrentPage}" Value="{x:Static
constant:ViewStates.GameMenuPage}">
                        <Setter Property="Visibility" Value="Visible" />
```

```
                    </DataTrigger>
                    </Style.Triggers>
            </Style>
          </UserControl.Style>

          <page:GameMenuPage DataContext="{Binding GameMenuViewModel}"/>
      </UserControl>

      <UserControl Grid.Column="0" Grid.Row="0">
          <UserControl.Style>
            <Style TargetType="UserControl">
                    <Setter Property="Visibility" Value="Collapsed" />
                    <Style.Triggers>
                    <DataTrigger Binding="{Binding Path=CurrentPage}" Value="{x:Static
constant:ViewStates.QuizPage}">
                    <Setter Property="Visibility" Value="Visible" />
                    </DataTrigger>
                    </Style.Triggers>
            </Style>
          </UserControl.Style>

          <page:QuizPage DataContext="{Binding QuizViewModel}"/>
      </UserControl>

      <UserControl Grid.Column="0" Grid.Row="0">
          <UserControl.Style>
            <Style TargetType="UserControl">
                    <Setter Property="Visibility" Value="Collapsed" />
                    <Style.Triggers>
                    <DataTrigger Binding="{Binding Path=CurrentPage}" Value="{x:Static
constant:ViewStates.WordScramblePage}">
                    <Setter Property="Visibility" Value="Visible" />
                    </DataTrigger>
                    </Style.Triggers>
            </Style>
          </UserControl.Style>

          <page:WordScramblePage DataContext="{Binding WordScrambleViewModel}"/>
      </UserControl>

      <UserControl Grid.Column="0" Grid.Row="0">
          <UserControl.Style>
            <Style TargetType="UserControl">
                    <Setter Property="Visibility" Value="Collapsed" />
                    <Style.Triggers>
                    <DataTrigger Binding="{Binding Path=CurrentPage}" Value="{x:Static
constant:ViewStates.PairsGamePage}">
                    <Setter Property="Visibility" Value="Visible" />
                    </DataTrigger>
                    </Style.Triggers>
            </Style>
          </UserControl.Style>
          <!-- As well as the usual definitions, the pairs game also includes definitions to the dependency
properties-->
          <page:PairsGamePage DataContext="{Binding PairsGameViewModel}" CheckPairCommand
="{Binding CheckPairCommand}"
```

```
                    TextBlockContains ="{Binding TextBlockContains}" TextBlock2Contains ="{Binding
TextBlock2Contains}"
                PairFound="{Binding PairFound}" ParentVM="{Binding ParentVM}"/>
    </UserControl>
        </Grid>
</Window>
```

CODE BEHIND:

```
using NEA_Project.Pages;
using NEA_Project.ViewModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
namespace NEA_Project
{
        /// <summary>
        /// Interaction logic for MainWindow.xaml
        /// </summary>
        public partial class MainWindow : Window
        {
            public MainWindow()
            {
                InitializeComponent();
                var vm = new MainWindowViewModel();
                this.DataContext = vm;
            }
        }
}
```

VIEWMODEL:

```
using NEA_Project.Constants;
using NEA_Project.Helpers;
using NEA_Project.Pages;
using SQLDatabase;
using System;
using System.Collections;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Security.Cryptography;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Input;
namespace NEA_Project.ViewModels
{
```

```
public class MainWindowViewModel : ObservableObject
{
    //Initialise
    private ViewStates _currentPage = ViewStates.StartPage;

    public Database Database = new Database();
    public LoginPageViewModel LoginPageViewModel { get; set; }
    public StartPageViewModel StartPageViewModel { get; set; }
    public SignUpPageViewModel SignUpPageViewModel { get; set; }
    public HomePageViewModel HomePageViewModel { get; set; }

    public ContinentsMapViewModel ContinentsMapViewModel { get; set; }
    public AfricaMapViewModel AfricaMapViewModel { get; set; }
    public AsiaMapViewModel AsiaMapViewModel { get; set; }
    public EuropeMapViewModel EuropeMapViewModel { get; set; }
    public NAmericaMapViewModel NAmericaMapViewModel { get; set; }
    public SAmericaMapViewModel SAmericaMapViewModel { get; set; }
    public OceaniaMapViewModel OceaniaMapViewModel { get; set; }

    public QuestionBankMenuViewModel QuestionBankMenuViewModel { get; set; }
    public QuestionBankEditViewModel QuestionBankEditViewModel { get; set; }
    public QuestionBankCreateViewModel QuestionBankCreateViewModel { get; set; }
    public QuestionBankDeleteViewModel QuestionBankDeleteViewModel { get; set; }
    public QuestionBankReadViewModel QuestionBankReadViewModel { get; set; }
    public UserStatsViewModel UserStatsViewModel { get; set; }

    public GameMenuViewModel GameMenuViewModel { get; set; }
    public QuizViewModel QuizViewModel { get; set; }
    public WordScrambleViewModel WordScrambleViewModel { get; set; }
    public PairsGameViewModel PairsGameViewModel { get; set; }


    public int UserID { get; set; }


    public ViewStates CurrentPage
    {
       get => _currentPage;
       set
       {
        RaiseAndSetIfChanged(ref _currentPage, value);
       }
    }
    private string _currentQuestionBank = "Default";
    public string CurrentQuestionBank { get =>_currentQuestionBank; set {
_currentQuestionBank = value; } }
    //Constructor
    public MainWindowViewModel()
    {
       //CREATES THE DATABASES
       Database.CreateTable("LoginDetails", "UserID INT, UserNames VARCHAR(20),
Passwords VARCHAR(500)");
       Database.CreateTable("QuestionBanks", "UserID INT, BankName VARCHAR(100),
QuestionID INT, Question VARCHAR(100), Answer VARCHAR(150)");
       Database.CreateTable("QuizScores", "QuizID INT, UserID INT, Score VARCHAR(100)");
       Database.CreateTable("PairsScores", "PairsID INT, UserID INT, Score VARCHAR(100)");
```

```
            Database.CreateTable("WordScrambleScores", "WordScrambleID INT, UserID INT, Score
VARCHAR(100)");
            PopulateQuestionBank();
            LoginPageViewModel = new LoginPageViewModel(this);
            StartPageViewModel = new StartPageViewModel(this);
            SignUpPageViewModel = new SignUpPageViewModel(this);
            HomePageViewModel = new HomePageViewModel(this);
            ContinentsMapViewModel = new ContinentsMapViewModel(this);
            AfricaMapViewModel = new AfricaMapViewModel(this);
            AsiaMapViewModel = new AsiaMapViewModel(this);
                EuropeMapViewModel = new EuropeMapViewModel(this);
            NAmericaMapViewModel = new NAmericaMapViewModel(this);
            SAmericaMapViewModel = new SAmericaMapViewModel(this);
            OceaniaMapViewModel = new OceaniaMapViewModel(this);
            QuestionBankMenuViewModel = new QuestionBankMenuViewModel(this);
            QuestionBankEditViewModel = new QuestionBankEditViewModel(this);
            QuestionBankCreateViewModel = new QuestionBankCreateViewModel(this);
            QuestionBankDeleteViewModel = new QuestionBankDeleteViewModel(this);
            QuestionBankReadViewModel = new QuestionBankReadViewModel(this);
            UserStatsViewModel = new UserStatsViewModel(this);
            GameMenuViewModel = new GameMenuViewModel(this);
            QuizViewModel = new QuizViewModel(this);
            WordScrambleViewModel = new WordScrambleViewModel(this);
            PairsGameViewModel = new PairsGameViewModel(this);


        }

        //When called the Current Page is assigned the ViewStates of the corresponding UserControl,
this enables the MainWindow to update
        public void ChangeToLoginPage()
        {

            CurrentPage = ViewStates.LoginPage;
        }

        public void ChangeToSignUpPage()
        {

            CurrentPage = ViewStates.SignUpPage;
        }

        public void ChangeToHomePage()
        {
            CurrentPage = ViewStates.HomePage;
        }

        public void ChangeToContinentsMap()
        {
            CurrentPage = ViewStates.ContinentsMap;
        }

        public void ChangeToAfricaMap()
        {
            CurrentPage = ViewStates.AfricaMap;
        }
```

```
public void ChangeToAsiaMap()
{
    CurrentPage = ViewStates.AsiaMap;
}

public void ChangeToEuropeMap()
{
    CurrentPage = ViewStates.EuropeMap;
}
        public void ChangeToNAmericaMap()
{
    CurrentPage = ViewStates.NAmericaMap;
}
public void ChangeToSAmericaMap()
{
    CurrentPage = ViewStates.SAmericaMap;
}
public void ChangeToOceaniaMap()
{
    CurrentPage = ViewStates.OceaniaMap;
}

public void ChangeToQuestionBankMenuPage()
{
    CurrentPage = ViewStates.QuestionBankMenu;
}

public void ChangeToQuestionBankEditPage()
{
    CurrentPage = ViewStates.QuestionBankEdit;
}
public void ChangeToQuestionBankCreatePage()
{
    CurrentPage = ViewStates.QuestionBankCreate;
}
public void ChangeToQuestionBankDeletePage()
{
    CurrentPage = ViewStates.QuestionBankDelete;
}
public void ChangeToQuestionBankReadPage()
{
    CurrentPage = ViewStates.QuestionBankRead;
}
public void ChangeToUserStatsPage()
{
    CurrentPage = ViewStates.UserStatsPage;
}
public void ChangeToGameMenuPage()
{
    CurrentPage = ViewStates.GameMenuPage;
}
public void ChangeToQuizPage()
{
    CurrentPage = ViewStates.QuizPage;
}
```

```
        public void ChangeToWordScramblePage()
        {
            CurrentPage = ViewStates.WordScramblePage;
        }
        public void ChangeToPairsGamePage()
        {
            CurrentPage = ViewStates.PairsGamePage;
             }


        //Using the inbuilt Hashing function SHA256, gets string input, converts into an array of ascii value, returns the hashed value
        public byte[] Hashing(string userInput)
        {
            byte[] ascii = new byte[userInput.Length];
            for (int i = 0; i < userInput.Length; i++)
            {

              ascii[i] = (byte)userInput[i];
            }

            HashAlgorithm sha = SHA256.Create();
            byte[] result = sha.ComputeHash(ascii);
            for (int j = 0; j < result.Length; j++)
            {
              Console.WriteLine(result[j]);
            }
            return result;
        }

        //Gets the which continent, then reads a text file, which then is split into different arrays, before being added to the database
        public void PopulateCountriesDatabase(string Continent)
        {
            string TopFolder = @"../../../Content";
            string filename = Continent + ".txt";
            string fullPath = Path.Combine(TopFolder, filename);
            string[] lines = File.ReadAllLines(fullPath);
            List<string[]> test = new List<string[]>();
            Database.CreateTable($"{Continent}", "ID INT, CountryName VARCHAR(60),Population VARCHAR(100), LandArea VARCHAR(100), Density VARCHAR(100)");
            int count = 0;
            foreach (var item in lines)
            {
              test.Add(item.Split(","));
            }
            if (Database.GetSize(Continent, "ID", "") == 0)
            {
              foreach (var item in test)
              {
                        Database.InsertData($"{Continent}", "ID, CountryName,Population,LandArea,Density", $"{count},'{item[0]}','{item[1]}','{item[2]}','{item[3]}'");
                        count += 1;
              }
            }
```

```
        }

        //Reads a text file (containing the default questions), then is split into different arrays, before being added a list of arrays
        public List<string[]> GetTheText()
        {
            List<string[]> FileContent = new List<string[]>();
            string TopFolder = @"../../../Content";
            string filename = "Default.txt";
            string fullPath = Path.Combine(TopFolder, filename);
            string[] lines = File.ReadAllLines(fullPath);
            int length = 0;
            foreach (var item in lines)
            {
             item[length].ToString().Trim();
             FileContent.Add(item.Split(","));
            }
            return FileContent;
        }

        //Using the default text file, the Questions and Answers are saved to the database, alongside a unique Question ID
        public void PopulateQuestionBank()
        {

            int size = Database.GetSize("QuestionBanks", "QuestionID", "");
            //If Question Banks is empty then populate the database
            if (size == 0)
            {
             List<string[]> FileContent = GetTheText();
             for (int i = 0; i < FileContent.Count; i++)
             {
                        Database.InsertData($"QuestionBanks", "UserID,BankName,QuestionID,Question,Answer", $"0,'Default', '{i + 1}','{FileContent[i][0]}','{FileContent[i][1]}'");

             }
            }

             }


        //This method is used mostly by the continent pages, and returns the file paths of the image name that is passed through
        public string GetImageFilePath(string ImageName)
        {
            List<string[]> FileContent = new List<string[]>();
            string TopFolder = @"../../../Images";
            string filename = $"{ImageName}.png";
            string fullPath = Path.Combine(TopFolder, filename);
            return fullPath;
        }
    }
}
```

START PAGE CODE:

XAML:

```
<UserControl x:Class="NEA_Project.Pages.StartPage"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:local="clr-namespace:NEA_Project"
             xmlns:viewmodels="clr-namespace:NEA_Project.ViewModels"
             d:DataContext="{d:DesignInstance Type=viewmodels:StartPageViewModel}"
             mc:Ignorable="d"
             d:DesignHeight="450" d:DesignWidth="800"
             >
            <Grid Background="White">
        <!-- Buttons with databinding that link to the page's view model  -->
        <Button Content = "Log in" Height = "30" Width = "80"
                Command="{Binding LoginButtonClickedCommand}"/>
        <TextBlock Text = "Don't have an account?"
         Height = "30" Margin = "210,180,122,109"/>
         <Button Content = "Sign Up" Height = "20" Width = "50" Margin = "245,164,122,109"
             Command= "{Binding SignUpButtonClickedCommand}"/>
             </Grid>
</UserControl>
```

CODE BEHIND:

```
using NEA_Project.Constants;
using NEA_Project.ViewModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace NEA_Project.Pages
{
            /// <summary>
            /// Interaction logic for StartPage.xaml
            /// </summary>
            public partial class StartPage : UserControl
            {
                  public StartPage()
                  {
                     InitializeComponent();
                  }
            }
}
```

VIEWMODEL:

```
using NEA_Project.Helpers;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Input;

namespace NEA_Project.ViewModels
{
        public class StartPageViewModel : ObservableObject
        {
            //Initialise
            private MainWindowViewModel _parent;
            public ICommand LoginButtonClickedCommand { get; }
            public ICommand SignUpButtonClickedCommand { get; }
            //Constructor
            public StartPageViewModel(MainWindowViewModel parent)
            {
                _parent = parent;
                LoginButtonClickedCommand = new SimpleCommand(_ => LoginButtonClicked());
                SignUpButtonClickedCommand = new SimpleCommand(_ => SignUpButtonClicked());
            }
            //when the button is pressed, the method in MainWindowViewModel is called to change the
        page.
            private void LoginButtonClicked()
            {
                _parent.ChangeToLoginPage();
            }
            //when the button is pressed, the method in MainWindowViewModel is called to change the
        page.
            private void SignUpButtonClicked()
            {
                _parent.ChangeToSignUpPage();

            }
        }
}
```

SIGN UP CODE:

XAML:

```
<UserControl x:Class="NEA_Project.Pages.SignUpPage"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:local="clr-namespace:NEA_Project.Pages"
             mc:Ignorable="d"
             d:DesignHeight="450" d:DesignWidth="800"
             >

        <Grid Background="White">
      <!-- Databinding that link to the view model: Username and Password TextBox, allow user to input and
button is linked to a command -->
```

```
        <StackPanel>
            <TextBlock Text="HI THIS IS SIGN UP :D"/>
            <StackPanel Orientation="Horizontal" Margin="0,20,0,20">
             <TextBlock Text="USERNAME:"/>
             <TextBox VerticalAlignment ="Center" Width="200" Text="{Binding UserNameInput ,
Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"/>
            </StackPanel>
            <StackPanel Orientation="Horizontal" Margin="0,20,0,20">
             <TextBlock Text="PASSWORD:"/>
             <TextBox VerticalAlignment ="Center" Width="200" Text="{Binding PasswordInput,
Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"/>
            </StackPanel>
            <Button Content = "Verify" Height = "30" Width = "80"
              Command="{Binding VerifyButtonClickedCommand}"/>
        </StackPanel>
            </Grid>
</UserControl>
```

CODE BEHIND:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace NEA_Project.Pages
{
            /// <summary>
            /// Interaction logic for SignUpPage.xaml
            /// </summary>
            public partial class SignUpPage : UserControl
            {
                  public SignUpPage()
                  {
                     InitializeComponent();
                  }
            }
}
```

VIEWMODEL:

```
using NEA_Project.Helpers;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net.Mail;
using System.Text;
using System.Threading.Tasks;
```

```
using System.Windows;
using System.Windows.Input;

namespace NEA_Project.ViewModels
{
        public class SignUpPageViewModel : ObservableObject
        {
            //Initialise
            private string _userNameInput = "Enter Your Username Here...";
            private string _passwordInput = "Enter Your Password Here...";
            private MainWindowViewModel _parent;
            public ICommand VerifyButtonClickedCommand { get; }

            //Constructor
            public SignUpPageViewModel(MainWindowViewModel parent)
            {
               _parent = parent;
               VerifyButtonClickedCommand = new SimpleCommand(_ => VerifyButtonClicked());
            }

            public string UserNameInput
            {
               get => _userNameInput;
               set
               {
                RaiseAndSetIfChanged(ref _userNameInput, value);
               }
            }

            public string PasswordInput
            {
               get => _passwordInput;
               set
               {
                RaiseAndSetIfChanged(ref _passwordInput, value);
               }
            }

            //validates the users inputs to ensure they are valid and then adds them to the database
        accordingly
            private void VerifyButtonClicked()
            {
               //Makes sure the username is not null and the password is at least 5 characters long
                    if (_userNameInput != null && _passwordInput.Length > 4) {
                 if (CheckDatabase())
                 {
                            //username already exists: gives the user the option to go to the login page or to
        try again
                            MessageBoxResult result = MessageBox.Show("Hi friend, seems this username
        already exists, would you like to login?", "My App", MessageBoxButton.YesNo,
        MessageBoxImage.Question, MessageBoxResult.No);
                            if (result == MessageBoxResult.Yes)
                        {
                           _parent.ChangeToLoginPage();
                            }
                            else
```

```
                {
                MessageBox.Show("Oh well, remember that usernames are case sensitive and
should be unique to you!", "My App");

                }
            }
            else
            {
                AddToDatabase();
            }
        }
    }

    //check database ensure that the username is not already taken
    private bool CheckDatabase()
    {
        try
        {
            if (_parent.Database.ReadData("LoginDetails", "UserNames", $"Usernames =
'{_userNameInput}'", 1)[0] == _userNameInput)
            {
                return true;
            }
            else
            {
                return false;
            }
        }
        catch (Exception)
        {


        }
        return false;

    }

    //Adds the user to the database, hashing their password, assigning them an unique ID and
adding a record in highscore
    //database to save their scores.
    private void AddToDatabase()
    {
        int DataBaseSize = 0;

        try
        {
            DataBaseSize = _parent.Database.GetSize("LoginDetails", "UserID", "");
        }
        catch (Exception)
        {


        }
        DataBaseSize += 1;
        byte[] hashedPassword = _parent.Hashing(_passwordInput);
        string stringHashedPassword = String.Join(" ", hashedPassword);
```

```
                _parent.Database.InsertData("LoginDetails", "UserID, UserNames, Passwords",
        $"{DataBaseSize},'{_userNameInput}', '{stringHashedPassword}'");
                _parent.Database.InsertData("UserStats","UserID, HighScore1,HighScore2,HighScore3",
        $"{DataBaseSize}, 0, 0, 0");
                _parent.ChangeToHomePage();
            }
        }
}
```

LOGIN CODE:

XAML:

```
<UserControl x:Class="NEA_Project.Pages.LoginPage"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:local="clr-namespace:NEA_Project.Pages"
        mc:Ignorable="d"
        d:DesignHeight="450" d:DesignWidth="800"
        >

        <Grid>
    <!-- Databinding that link to the view model: Username and Password TextBox, allow user to input and
button is linked to a command -->
    <StackPanel>
      <TextBlock Text="THIS IS THE LOGIN PAGE"/>
      <StackPanel Orientation="Horizontal" Margin="0,20,0,20">
       <TextBlock Text="USERNAME:"/>
       <TextBox VerticalAlignment ="Center" Width="200" Text="{Binding UserNameInput,
Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"/>
      </StackPanel>
      <StackPanel Orientation="Horizontal" Margin="0,20,0,20">
       <TextBlock Text="PASSWORD:"/>
       <TextBox VerticalAlignment ="Center" Width="200" Text="{Binding PasswordInput,
Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"/>
      </StackPanel>
      <Button Content = "Log in" Height = "30" Width = "80"
        Command="{Binding LoginButtonClickedCommand}"/>
    </StackPanel>
        </Grid>
</UserControl>
```

CODE BEHIND:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
```

```
using System.Windows.Shapes;

namespace NEA_Project.Pages
{
        /// <summary>
        /// Interaction logic for LoginPage.xaml
        /// </summary>
        public partial class LoginPage : UserControl
        {
            public LoginPage()
            {
                InitializeComponent();
            }
        }
}
```

VIEWMODEL:

```
using NEA_Project.Helpers;
using SQLDatabase;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Controls.Primitives;
using System.Windows.Input;
namespace NEA_Project.ViewModels
{
        public class LoginPageViewModel : ObservableObject
        {
            //Initialise
            private string _userNameInput = "Enter Your Username Here...";
            private string _passwordInput = "Enter Your Password Here...";
            private MainWindowViewModel _parent;

            public ICommand LoginButtonClickedCommand { get; }
            //Constructor
            public LoginPageViewModel(MainWindowViewModel parent)
            {
                _parent = parent;
                LoginButtonClickedCommand = new SimpleCommand(_ => LoginButtonClicked());
            }

            public string UserNameInput
            {
                get => _userNameInput;
                set
                {
                 RaiseAndSetIfChanged(ref _userNameInput, value);
                }
            }

            public string PasswordInput
            {
```

```
            get => _passwordInput;
            set
            {
              RaiseAndSetIfChanged(ref _passwordInput, value);
            }
        }

        private void LoginButtonClicked()
        {
            //Validates users input, assigns reads database and assigns correct UserID then changes
to home page
            if (CheckDataBase())
            {
              string UserID = _parent.Database.ReadData("LoginDetails", "UserID", $"UserNames =
'{_userNameInput}'", 1)[0];
              _parent.UserID = Int32.Parse(UserID);
              _parent.ChangeToHomePage();
            }
            else
            {
              //incorrect input: given the choice to sign up (taken to sign up page) or have another go
              MessageBoxResult result = MessageBox.Show("Hi friend, seems like you're not on our
system, would you like to sign up?", "My App", MessageBoxButton.YesNo,
MessageBoxImage.Question, MessageBoxResult.No);
              if (result == MessageBoxResult.Yes)
              {
                  _parent.ChangeToSignUpPage();
              }
              else
              {
                      MessageBox.Show("Oh well, remember that passwords and usernames are case
sensitive!", "My App");

              }

            }


        }

        private bool CheckDataBase()
        {
            //uses the hashing method in MainWindowViewModel on the user's password input.
              byte[] hashedPassword = _parent.Hashing(_passwordInput);
            string stringHashedPassword = String.Join("", hashedPassword);
            List<string> temp = new List<string>();
            string correctPassword = "";
            try
              {

                      temp = _parent.Database.ReadData("LoginDetails", "Passwords", $"UserNames
             = '{_userNameInput}'", 1);
                      // this is to handle the exception if Read data returns nothing.
                      if (temp.Count > 0)
                      {
```

```
                                correctPassword = _parent.Database.ReadData("LoginDetails",
                    "Passwords", $"UserNames = '{_userNameInput}'", 1)[0];
                            }

                        }
                        catch (Exception)
                        {


                        }

                        //compared the users input to the password in the database
                        if (stringHashedPassword == correctPassword)
                        {
                         return true;
                        }

                        return false;
                    }

            }
}
```

HOME PAGE CODE:

XAML:

```
<UserControl x:Class="NEA_Project.Pages.HomePage"

             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

             xmlns:local="clr-namespace:NEA_Project.Pages"

             mc:Ignorable="d"

             d:DesignHeight="450" d:DesignWidth="800">


            <Grid Background="White">

        <!-- Databinding that link to the view model: buttons linked to commands that allow navigation -->

        <StackPanel>

            <TextBlock Text="THIS IS THE HOME PAGE!!!"></TextBlock>

            <Button Content = "Map!" Height = "30" Width = "80"
```

```
            Command="{Binding MapButtonClickedCommand}"/>
        <Button Content = "Question Bank Menu!" Height = "30" Width = "80"
            Command="{Binding QuestionBankMenuButtonClickedCommand}"/>
        <Button Content = "Games!" Height = "30" Width = "80"
            Command="{Binding GameMenuButtonClickedCommand}"/>
        <Button Content = "User Stats!" Height = "30" Width = "80"
            Command="{Binding UserStatsButtonClickedCommand}"/>
    </StackPanel>
        </Grid>
</UserControl>
```

CODE BEHIND:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace NEA_Project.Pages
{
        /// <summary>
```

```
        /// Interaction logic for HomePage.xaml
        /// </summary>
        public partial class HomePage : UserControl
        {
            public HomePage()
            {
                InitializeComponent();
            }
        }
}
```

VIEWMODEL:

```
using NEA_Project.Helpers;
using SQLDatabase;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Controls;
using System.Windows.Input;

namespace NEA_Project.ViewModels
{
        public class HomePageViewModel : ObservableObject
        {
            //Initialise
            private MainWindowViewModel _parent;
```

```
        public ICommand MapButtonClickedCommand { get; }

        public ICommand QuestionBankMenuButtonClickedCommand { get; }


        public ICommand GameMenuButtonClickedCommand { get; }

        public ICommand UserStatsButtonClickedCommand { get; }

        //Constructor

        public HomePageViewModel(MainWindowViewModel parent)

        {

            _parent = parent;

            MapButtonClickedCommand = new SimpleCommand(_ => MapButtonClicked());

            QuestionBankMenuButtonClickedCommand = new SimpleCommand(_ =>
QuestionBankButtonClicked());

            GameMenuButtonClickedCommand = new SimpleCommand(_ =>
GameMenuButtonClicked());

            UserStatsButtonClickedCommand = new SimpleCommand(_=>UserStatsButtonClicked());

        }


        //when the button is pressed, the method in MainWindowViewModel is called to change the
page.

        private void MapButtonClicked()

        {

            _parent.ChangeToContinentsMap();

        }

        //when the button is pressed, the method in MainWindowViewModel is called to change the
page.

        private void QuestionBankButtonClicked()

        {

            _parent.ChangeToQuestionBankMenuPage();

        }

        //when the button is pressed, the method in MainWindowViewModel is called to change the
page.
```

```
        private void GameMenuButtonClicked()
        {
            _parent.ChangeToGameMenuPage();
        }
        //when the button is pressed, the method in MainWindowViewModel is called to change the page.
        private void UserStatsButtonClicked()
        {
            _parent.ChangeToUserStatsPage();
        }
    }
}
```

CONTINENT MAP CODE:

XAML:

```
<UserControl x:Class="NEA_Project.Pages.ContinentsMap"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:local="clr-namespace:NEA_Project.Pages"
        mc:Ignorable="d"
        d:DesignHeight="450" d:DesignWidth="800">
        <Grid>
    <!-- Databinding: Buttons that allow you to navigate to the different continent pages and a home button, as well as an image
    with its source that is also bound to the ImageFilepath in the ViewModel-->

    <Grid.Background>
        <ImageBrush ImageSource="{Binding ImageFilePath}"/>
    </Grid.Background>

     <Button Content="Back To Home" Margin="350,5,315,343" Command="{Binding HomePageButtonCommandClicked}" Width="85" Height="40" />
    <Button Margin="50,50,520,217" Opacity="0.05" Command="{Binding NAmericaButtonClickedCommand}"/>
    <Button Margin="210,260,500,15" Opacity="0.05" Command="{Binding SAmericaButtonClickedCommand}"/>
    <Button Margin="362,100,330,250" Opacity="0.05" Command="{Binding EuropeButtonClickedCommand}"/>
```

```
        <Button Margin="368,220,340,100" Opacity="0.05" Command="{Binding
AfricaButtonClickedCommand}" />
        <Button Margin="485,100,100,150" Opacity="0.05" Command="{Binding
AsiaButtonClickedCommand}"/>
        <Button Margin="600,315,50,45" Opacity="0.05" Command="{Binding
OceaniaButtonClickedCommand}"/>
            </Grid>
</UserControl>
```

CODE BEHIND:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace NEA_Project.Pages
{
            /// <summary>
            /// Interaction logic for ContinentsMap.xaml
            /// </summary>
            public partial class ContinentsMap : UserControl
            {
                  public ContinentsMap()
                  {
                     InitializeComponent();
                  }
            }
}
```

VIEWMODEL:

```
using NEA_Project.Helpers;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Input;

namespace NEA_Project.ViewModels
{
            public class ContinentsMapViewModel
            {
                  //Initialise
                  private MainWindowViewModel _parent;
```

```
        private string _imageFilePath;
        public ICommand AfricaButtonClickedCommand { get; }
        public ICommand AsiaButtonClickedCommand { get; }
        public ICommand EuropeButtonClickedCommand { get; }
        public ICommand NAmericaButtonClickedCommand { get; }
        public ICommand SAmericaButtonClickedCommand { get; }
        public ICommand OceaniaButtonClickedCommand { get; }
        public ICommand HomePageButtonCommandClicked { get; }
        //Constructor
        public ContinentsMapViewModel (MainWindowViewModel parent)
        {
            _parent = parent;
                //assigns the image file path by passing the relevant image name through the
GetImageFilePath from the MainWindowViewModel.
            _imageFilePath = _parent.GetImageFilePath("Map_Continents");
            AfricaButtonClickedCommand = new SimpleCommand(_ => AfricaButtonClicked());
            AsiaButtonClickedCommand = new SimpleCommand(_ => AsiaButtonClicked());
            EuropeButtonClickedCommand = new SimpleCommand(_ => EuropeButtonClicked());
            NAmericaButtonClickedCommand = new SimpleCommand(_ =>
NAmericaButtonClicked());
            SAmericaButtonClickedCommand = new SimpleCommand(_ =>
SAmericaButtonClicked());
            OceaniaButtonClickedCommand = new SimpleCommand(_ => OceaniaButtonClicked());
            HomePageButtonCommandClicked = new SimpleCommand(_ =>
HomePageButtonClicked());


        }

        //Assigns the image file path
        public string ImageFilePath { get => _imageFilePath; }

        //assigning public to private, allows them to change if updated whilst the program is running
(RaiseAndSetIfChanged from Observable Objects.)
        private void AfricaButtonClicked()
        {
            _parent.ChangeToAfricaMap();
        }
        //when the button is pressed, the method in MainWindowViewModel is called to change the
page.
        private void AsiaButtonClicked()
        {
            _parent.ChangeToAsiaMap();
        }
        //when the button is pressed, the method in MainWindowViewModel is called to change the
page.
        private void EuropeButtonClicked()
        {
            _parent.ChangeToEuropeMap();
        }
        //when the button is pressed, the method in MainWindowViewModel is called to change the
page.
        private void NAmericaButtonClicked()
        {
            _parent.ChangeToNAmericaMap();
        }
```

```
            //when the button is pressed, the method in MainWindowViewModel is called to change the
        page.
            private void SAmericaButtonClicked()
            {
                _parent.ChangeToSAmericaMap();
            }
            //when the button is pressed, the method in MainWindowViewModel is called to change the
        page.
            private void OceaniaButtonClicked()
            {
                _parent.ChangeToOceaniaMap();
            }
            //when the button is pressed, the method in MainWindowViewModel is called to change the
        page.
            private void HomePageButtonClicked()
            {
                _parent.ChangeToHomePage();
            }
        }
}
```

AFRICA MAP CODE:

XAML:

```
<UserControl x:Class="NEA_Project.Pages.AfricaMap"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:local="clr-namespace:NEA_Project.Pages"
xmlns:viewmodels="clr-namespace:NEA_Project.ViewModels" d:DataContext="{d:DesignInstance
Type=viewmodels:AfricaMapViewModel}"
        mc:Ignorable="d"
        d:DesignHeight="450" d:DesignWidth="800">
        <Grid>
    <!--Databinding to the view model: ComboBox, that the user can type into and search for countries, as
well as textblocks that display
    the information of the selected country, a button that triggers a search and a button that returns to
home, as well as an image
    with its source that is also bound to the ImageFilepath in the ViewModel -->
    <StackPanel>
        <Image Source="{Binding ImageFilePath}" Height="325" Width="383"/>
        <StackPanel Orientation="Horizontal">
         <ComboBox
          IsEditable="true"
          Width ="200"
          Text="{Binding UserInput}"
          ItemsSource="{Binding Countries}"
          SelectedValue="{Binding Path=UserInput,Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
                />
         <Button Content="Search Country" Command="{Binding SearchButtonCommand}"></Button>
         <TextBlock Text="NAME" Margin="30,1,1,1"></TextBlock>
         <TextBlock Text="{Binding CountryName}" Width="100"/>
        </StackPanel>
```

```
        <StackPanel Orientation="Horizontal">
         <TextBlock Text="POPULATION "></TextBlock>
         <TextBlock Text="{Binding CountryPopulation}" Width="100"></TextBlock>
         <TextBlock Text="LAND AREA "></TextBlock>
         <TextBlock Text="{Binding CountryLandArea}" Width="100"></TextBlock>
         <TextBlock Text="POPULATION DENSITY "></TextBlock>
           <TextBlock Text="{Binding CountryDensity}" Width="100"></TextBlock>

        </StackPanel>
        <Button Content="Back To Main Map" Command="{Binding MapButtonCommand}" Width="200"/>

    </StackPanel>
      </Grid>
</UserControl>
```

CODE BEHIND:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace NEA_Project.Pages
{
        /// <summary>
        /// Interaction logic for AfricaMap.xaml
        /// </summary>
        public partial class AfricaMap : UserControl
        {
            public AfricaMap()
            {
                InitializeComponent();
            }
        }
}
```

VIEWMODEL:

```
using NEA_Project.Helpers;
using SQLDatabase;
using System;
using System.Collections.Generic;
using System.Diagnostics.Metrics;
using System.Dynamic;
using System.Linq;
using System.Net.Security;
```

```
using System.Security.Policy;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Input;

namespace NEA_Project.ViewModels
{
        public class AfricaMapViewModel:ObservableObject
        {
            //Initialise
            private MainWindowViewModel _parent;
            public Database Africa { get; }
            private string _countryName;
            private string _countryPopulation;
            private string _countryLandArea;
            private string _countryDensity;
            private string _userInput;
            private string _imageFilePath;
            private List<string> _countries = new List<string>();

            public ICommand SearchButtonCommand { get; }
            public ICommand MapButtonCommand { get; }
            //constructor
            public AfricaMapViewModel(MainWindowViewModel parent)
            {
                _parent = parent;
                //assigns the image file path by passing the relevant image name through the
        GetImageFilePath from the MainWindowViewModel.
                _imageFilePath = _parent.GetImageFilePath("AfricanMap");
                //will only populate database if its empty
                if (_parent.Database.GetSize("Africa","ID","") == 0)
                {
                    _parent.PopulateCountriesDatabase("Africa");
                }
                PopulateList();
                SearchButtonCommand = new SimpleCommand(_ => SearchButtonClickedCommand());
                MapButtonCommand = new SimpleCommand(_ => GoToMapPageCommand());
            }
            //Assigns the image file path
            public string ImageFilePath { get => _imageFilePath; }

            //assigning public to private, allows them to change if updated whilst the program is running
        (RaiseAndSetIfChanged from Observable Objects).
            public string CountryName { get=> _countryName; set { RaiseAndSetIfChanged(ref
        _countryName, value); } }
            public string CountryPopulation { get => _countryPopulation; set { RaiseAndSetIfChanged(ref
        _countryPopulation, value); } }
            public string CountryLandArea { get => _countryLandArea; set { RaiseAndSetIfChanged(ref
        _countryLandArea, value); } }
            public string CountryDensity { get => _countryDensity; set {RaiseAndSetIfChanged(ref
        _countryDensity, value); } }
            public string UserInput { get=> _userInput; set {RaiseAndSetIfChanged(ref _userInput, value);
        } }
            public List<string> Countries { get => _countries; }

            //Once a user has selected a Country, the database will be read and retrieves the information
```

```
        public void GetCountryInfo()
        {
            List<string> CountryInfo = _parent.Database.ReadData("Africa", "CountryName,
    Population, LandArea, Density", $"CountryName LIKE '{UserInput}'", 4);
            if (CountryInfo.Count > 0)
            {
              CountryName = CountryInfo[0];
              CountryPopulation = CountryInfo[1];
              CountryLandArea = CountryInfo[2];
              CountryDensity = CountryInfo[3];
            }

        }

        //when the button is clicked GetCountryInfo is called
        public void SearchButtonClickedCommand()
        {
            GetCountryInfo();
        }

        //In order to populate the ComboBox, there must be a list of all the possible countries saved,
    this reads the database
        //and saves all the names to the list.
        private void PopulateList()
              {

            for (int i = 0; i < _parent.Database.GetSize("Africa", "ID", ""); i++)
            {
              string country = _parent.Database.ReadData("Africa", "CountryName", $"ID = {i}",1)[0];
              _countries.Add(country);
            }

        }
        //when the button is pressed, the method in MainWindowViewModel is called to change the
    page.
        public void GoToMapPageCommand()
        {
            _parent.ChangeToContinentsMap();
        }
    }
}
```

ASIA MAP CODE:

XAML:

```
<UserControl x:Class="NEA_Project.Pages.AsiaMap"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:local="clr-namespace:NEA_Project.Pages"
xmlns:viewmodels="clr-namespace:NEA_Project.ViewModels" d:DataContext="{d:DesignInstance
Type=viewmodels:AsiaMapViewModel}"
        mc:Ignorable="d"
        d:DesignHeight="450" d:DesignWidth="800">
```

```
        <Grid>
    <!--Databinding to the view model: ComboBox, that the user can type into and search for countries, as well as textblocks that display
    the information of the selected country, a button that triggers a search and a button that returns to home, as well as an image
    with its source that is also bound to the ImageFilepath in the ViewModel -->
    <StackPanel>
      <Image Source="{Binding ImageFilePath}" Height="325" Width="720"/>
      <StackPanel Orientation="Horizontal">
            <ComboBox
        IsEditable="true"
        Width ="200"
        Text="{Binding UserInput}"
        ItemsSource="{Binding Countries}"
        SelectedValue="{Binding Path=UserInput,Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
                />
        <Button Content="Search Country" Command="{Binding SearchButtonCommand}"></Button>
        <TextBlock Text="NAME" Margin="30,1,1,1"></TextBlock>
        <TextBlock Text="{Binding CountryName}" Width="100"/>
      </StackPanel>
      <StackPanel Orientation="Horizontal">
        <TextBlock Text="POPULATION "></TextBlock>
        <TextBlock Text="{Binding CountryPopulation}" Width="100"></TextBlock>
        <TextBlock Text="LAND AREA "></TextBlock>
        <TextBlock Text="{Binding CountryLandArea}" Width="100"></TextBlock>
        <TextBlock Text="POPULATION DENSITY "></TextBlock>
        <TextBlock Text="{Binding CountryDensity}" Width="100"></TextBlock>

      </StackPanel>
      <Button Content="Back To Main Map" Command="{Binding MapButtonCommand}" Width="200"/>
    </StackPanel>

        </Grid>
</UserControl>
```

CODE BEHIND:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
namespace NEA_Project.Pages
{
        /// <summary>
        /// Interaction logic for AsiaMap.xaml
        /// </summary>
```

```
        public partial class AsiaMap : UserControl
        {
            public AsiaMap()
            {
                InitializeComponent();
            }
        }
}
```

VIEWMODEL:

```
using NEA_Project.Helpers;
using SQLDatabase;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Input;
using static System.Data.Entity.Infrastructure.Design.Executor;

namespace NEA_Project.ViewModels
{
        public class AsiaMapViewModel : ObservableObject
        {
            //initialise
            private MainWindowViewModel _parent;
            public Database Asia { get; }
            private string _countryName;
            private string _countryPopulation;
            private string _countryLandArea;
            private string _countryDensity;
            private string _userInput;
            private string _imageFilePath;
            private List<string> _countries = new List<string>();
            public ICommand SearchButtonCommand { get; }
            public ICommand MapButtonCommand { get; }

            //constructor
            public AsiaMapViewModel(MainWindowViewModel parent)
            {
                _parent = parent;
                    //assigns the image file path by passing the relevant image name through the
        GetImageFilePath from the MainWindowViewModel.
                _imageFilePath = _parent.GetImageFilePath("AsiaMap");
                //will only populate database if its empty
                if (_parent.Database.GetSize("Asia", "ID", "") == 0)
                {
                    _parent.PopulateCountriesDatabase("Asia");
                }
                PopulateList();
                SearchButtonCommand = new SimpleCommand(_ => SearchButtonClickedCommand());
                MapButtonCommand = new SimpleCommand(_ => GoToMapPageCommand());

            }
            //Assigns the image file path
```

```
        public string ImageFilePath { get => _imageFilePath; }

        //assigning public to private, allows them to change if updated whilst the program is running
    (RaiseAndSetIfChanged from Observable Objects.)
        public string CountryName { get => _countryName; set { RaiseAndSetIfChanged(ref
    _countryName, value); } }
        public string CountryPopulation { get => _countryPopulation; set { RaiseAndSetIfChanged(ref
    _countryPopulation, value); } }
        public string CountryLandArea { get => _countryLandArea; set { RaiseAndSetIfChanged(ref
    _countryLandArea, value); } }
        public string CountryDensity { get => _countryDensity; set { RaiseAndSetIfChanged(ref
    _countryDensity, value); } }
        public string UserInput { get => _userInput; set { RaiseAndSetIfChanged(ref _userInput,
    value); } }
        public List<string> Countries { get => _countries; }

        //Once a user has selected a Country, the database will be read and retrieves the information
        public void GetCountryInfo()
            {
            List<string> CountryInfo = _parent.Database.ReadData("Asia", "CountryName, Population,
    LandArea, Density", $"CountryName LIKE '{UserInput}'", 4);
            if (CountryInfo.Count > 0)
            {
              CountryName = CountryInfo[0];
              CountryPopulation = CountryInfo[1];
              CountryLandArea = CountryInfo[2];
              CountryDensity = CountryInfo[3];
            }
        }

        //when the button is clicked GetCountryInfo is called
        public void SearchButtonClickedCommand()
        {
            GetCountryInfo();
        }
        //In order to populate the ComboBox, there must be a list of all the possible countries saved,
    this reads the database
        //and saves all the names to the list.
        private void PopulateList()
        {

            for (int i = 0; i < _parent.Database.GetSize("Asia", "ID",""); i++)
            {
              string country = _parent.Database.ReadData("Asia", "CountryName", $"ID = {i}",1)[0];
              _countries.Add(country);
            }

        }
        //when the button is pressed, the method in MainWindowViewModel is called to change the
    page.
        private void GoToMapPageCommand()
        {
            _parent.ChangeToContinentsMap();
        }
    }
}
```

EUROPE MAP CODE:

XAML:

```
<UserControl x:Class="NEA_Project.Pages.EuropeMap"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:local="clr-namespace:NEA_Project.Pages"
xmlns:viewmodels="clr-namespace:NEA_Project.ViewModels" d:DataContext="{d:DesignInstance
Type=viewmodels:EuropeMapViewModel}"
             mc:Ignorable="d"
             d:DesignHeight="450" d:DesignWidth="800">
              <Grid>
        <!--Databinding to the view model: ComboBox, that the user can type into and search for countries, as
well as textblocks that display
        the information of the selected country, a button that triggers a search and a button that returns to
home, as well as an image
        with its source that is also bound to the ImageFilepath in the ViewModel -->
        <StackPanel>
           <Image Source="{Binding ImageFilePath}" Height="325" Width="692"/>
           <StackPanel Orientation="Horizontal">
            <ComboBox
             IsEditable="true"
             Width ="200"
             Text="{Binding UserInput}"
             ItemsSource="{Binding Countries}"
             SelectedValue="{Binding Path=UserInput,Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
                       />
            <Button Content="Search Country" Command="{Binding SearchButtonCommand}"></Button>
            <TextBlock Text="NAME" Margin="30,1,1,1"></TextBlock>
            <TextBlock Text="{Binding CountryName}" Width="100"/>
           </StackPanel>
           <StackPanel Orientation="Horizontal">
                    <TextBlock Text="POPULATION "></TextBlock>
            <TextBlock Text="{Binding CountryPopulation}" Width="100"></TextBlock>
            <TextBlock Text="LAND AREA "></TextBlock>
            <TextBlock Text="{Binding CountryLandArea}" Width="100"></TextBlock>
            <TextBlock Text="POPULATION DENSITY "></TextBlock>
            <TextBlock Text="{Binding CountryDensity}" Width="100"></TextBlock>

           </StackPanel>
           <Button Content="Back To Main Map" Command="{Binding MapButtonCommand}" Width="200"/>
        </StackPanel>
              </Grid>
</UserControl>
```

CODE BEHIND:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
```

```
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace NEA_Project.Pages
{
        /// <summary>
        /// Interaction logic for EuropeMap.xaml
        /// </summary>
        public partial class EuropeMap : UserControl
        {
            public EuropeMap()
            {
                InitializeComponent();
            }
        }
}
```

VIEWMODEL:

```
using NEA_Project.Helpers;
using SQLDatabase;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Input;

namespace NEA_Project.ViewModels
{
        public class EuropeMapViewModel:ObservableObject
        {
            //initialise
            private MainWindowViewModel _parent;
            public Database Europe { get; }
            private string _countryName;
            private string _countryPopulation;
            private string _countryLandArea;
            private string _countryDensity;
            private string _imageFilePath;
            private string _userInput;

            private List<string> _countries = new List<string>();
            public ICommand SearchButtonCommand { get; }
            public ICommand MapButtonCommand { get; }
            //constructor
            public EuropeMapViewModel(MainWindowViewModel parent)
            {
                _parent = parent;
```

```
            //assigns the image file path by passing the relevant image name through the
GetImageFilePath
            //from the MainWindowViewModel.
            _imageFilePath = _parent.GetImageFilePath("EuropeanMap");
            //will only populate database if its empty
            if (_parent.Database.GetSize("Europe", "ID", "") == 0)
            {
                _parent.PopulateCountriesDatabase("Europe");
            }
            PopulateList();
            SearchButtonCommand = new SimpleCommand(_ => SearchButtonClickedCommand());
            MapButtonCommand = new SimpleCommand(_ => GoToMapPageCommand());
        }
        //Assigns the image file path
        public string ImageFilePath { get => _imageFilePath; }

        //assigning public to private, allows them to change if updated whilst the program is running
(RaiseAndSetIfChanged from Observable Objects.)
        public string CountryName { get => _countryName; set { RaiseAndSetIfChanged(ref
_countryName, value); } }
        public string CountryPopulation { get => _countryPopulation; set { RaiseAndSetIfChanged(ref
_countryPopulation, value); } }
        public string CountryLandArea { get => _countryLandArea; set { RaiseAndSetIfChanged(ref
_countryLandArea, value); } }
        public string CountryDensity { get => _countryDensity; set { RaiseAndSetIfChanged(ref
_countryDensity, value); } }
        public string UserInput { get => _userInput; set { RaiseAndSetIfChanged(ref _userInput,
value); } }
        public List<string> Countries { get => _countries; }

        //Once a user has selected a Country, the database will be read and retrieves the information
        public void GetCountryInfo()
        {
            List<string> CountryInfo = _parent.Database.ReadData("Europe", "CountryName,
Population, LandArea, Density", $"CountryName LIKE '{UserInput}'", 4);
                if (CountryInfo.Count > 0)
            {
              CountryName = CountryInfo[0];
              CountryPopulation = CountryInfo[1];
              CountryLandArea = CountryInfo[2];
              CountryDensity = CountryInfo[3];
            }
        }

        //when the button is clicked GetCountryInfo is called
        public void SearchButtonClickedCommand()
        {
            GetCountryInfo();
        }

        //In order to populate the ComboBox, there must be a list of all the possible countries saved,
this reads the database and saves all the names to the list.
        private void PopulateList()
        {
            for (int i = 0; i < _parent.Database.GetSize("Europe", "ID", ""); i++)
            {
```

```
                string country = _parent.Database.ReadData("Europe", "CountryName", $"ID = {i}", 1)[0];
                _countries.Add(country);
            }

        }

        //when the button is pressed, the method in MainWindowViewModel is called to change the
    page.
        private void GoToMapPageCommand()
        {
            _parent.ChangeToContinentsMap();
        }
    }
}
```

NORTH AMERICA MAP CODE:

XAML:

```
<UserControl x:Class="NEA_Project.Pages.NAmericaMap"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:local="clr-namespace:NEA_Project.Pages"
xmlns:viewmodels="clr-namespace:NEA_Project.ViewModels" d:DataContext="{d:DesignInstance
Type=viewmodels:NAmericaMapViewModel}"
        mc:Ignorable="d"
        d:DesignHeight="450" d:DesignWidth="800">
        <Grid>
    <!--Databinding to the view model: ComboBox, that the user can type into and search for countries, as
well as textblocks that display
    the information of the selected country, a button that triggers a search and a button that returns to
home, as well as an image
    with its source that is also bound to the ImageFilepath in the ViewModel -->
    <StackPanel>
        <Image Source="{Binding ImageFilePath}" Height="325" Width="603"/>
        <StackPanel Orientation="Horizontal">
         <ComboBox
          IsEditable="true"
          Width ="200"
            Text="{Binding UserInput}"
          ItemsSource="{Binding Countries}"
          SelectedValue="{Binding Path=UserInput,Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
                />
         <Button Content="Search Country" Command="{Binding SearchButtonCommand}"></Button>
         <TextBlock Text="NAME" Margin="30,1,1,1"></TextBlock>
         <TextBlock Text="{Binding CountryName}" Width="100"/>
        </StackPanel>
        <StackPanel Orientation="Horizontal">
         <TextBlock Text="POPULATION "></TextBlock>
         <TextBlock Text="{Binding CountryPopulation}" Width="100"></TextBlock>
         <TextBlock Text="LAND AREA "></TextBlock>
         <TextBlock Text="{Binding CountryLandArea}" Width="100"></TextBlock>
         <TextBlock Text="POPULATION DENSITY "></TextBlock>
         <TextBlock Text="{Binding CountryDensity}" Width="100"></TextBlock>
```

```
        </StackPanel>
        <Button Content="Back To Main Map" Command="{Binding MapButtonCommand}" Width="200"/>
    </StackPanel>
      </Grid>
</UserControl>
```

CODE BEHIND:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace NEA_Project.Pages
{
        /// <summary>
        /// Interaction logic for NAmericaMap.xaml
        /// </summary>
        public partial class NAmericaMap : UserControl
        {
            public NAmericaMap()
            {
                InitializeComponent();
            }
        }
}
```

VIEWMODEL:

```
using NEA_Project.Helpers;
using SQLDatabase;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Input;

namespace NEA_Project.ViewModels
{
        public class NAmericaMapViewModel:ObservableObject
        {
            //Initialise
            private MainWindowViewModel _parent;
            public Database NAmerica { get; }
            private string _countryName;
            private string _countryPopulation;
```

```
        private string _countryLandArea;
        private string _countryDensity;
        private string _imageFilePath;
        private string _userInput;
        private List<string> _countries = new List<string>();
        public ICommand SearchButtonCommand { get; }
        public ICommand MapButtonCommand { get; }
        //Constructor
        public NAmericaMapViewModel(MainWindowViewModel parent)
        {
            _parent = parent;
            //assigns the image file path by passing the relevant image name through the
GetImageFilePath from the MainWindowViewModel.
            _imageFilePath = _parent.GetImageFilePath("NAmericaMap");
            //will only populate database if its empty
            if (_parent.Database.GetSize("NorthAmerica", "ID", "") == 0)
            {
                _parent.PopulateCountriesDatabase("NorthAmerica");
            }
            PopulateList();
            SearchButtonCommand = new SimpleCommand(_ => SearchButtonClickedCommand());
            MapButtonCommand = new SimpleCommand(_ => GoToMapPageCommand());
        }
        //Assigns the image file path
        public string ImageFilePath { get => _imageFilePath; }

        //assigning public to private, allows them to change if updated whilst the program is running
(RaiseAndSetIfChanged from Observable Objects.)
        public string CountryName { get => _countryName; set { RaiseAndSetIfChanged(ref
_countryName, value); } }
        public string CountryPopulation { get => _countryPopulation; set { RaiseAndSetIfChanged(ref
_countryPopulation, value); } }
        public string CountryLandArea { get => _countryLandArea; set { RaiseAndSetIfChanged(ref
_countryLandArea, value); } }
        public string CountryDensity { get => _countryDensity; set { RaiseAndSetIfChanged(ref
_countryDensity, value); } }
        public string UserInput { get => _userInput; set { RaiseAndSetIfChanged(ref _userInput,
value); } }
        public List<string> Countries { get => _countries; }

        //Once a user has selected a Country, the database will be read and retrieves the information
        public void GetCountryInfo()
        {
            List<string> CountryInfo = _parent.Database.ReadData("NorthAmerica", "CountryName,
Population, LandArea, Density", $"CountryName LIKE '{UserInput}'", 4);
            if (CountryInfo.Count > 0)
            {
              CountryName = CountryInfo[0];
              CountryPopulation = CountryInfo[1];
              CountryLandArea = CountryInfo[2];
              CountryDensity = CountryInfo[3];
            }
        }

        //when the button is clicked GetCountryInfo is called
        public void SearchButtonClickedCommand()
```

```
            {
                GetCountryInfo();
            }
            //In order to populate the ComboBox, there must be a list of all the possible countries saved,
        this reads the database and saves all the names to the list.
            private void PopulateList()
            {

                for (int i = 0; i < _parent.Database.GetSize("NorthAmerica", "ID",""); i++)
                {
                 string country = _parent.Database.ReadData("NorthAmerica", "CountryName", $"ID =
        {i}",1)[0];
                 _countries.Add(country);
                }

            }
            //when the button is pressed, the method in MainWindowViewModel is called to change the
        page.
            private void GoToMapPageCommand()
            {
                _parent.ChangeToContinentsMap();
            }
        }
}
```

SOUTH AMERICA MAP CODE:

XAML:

```
<UserControl x:Class="NEA_Project.Pages.SAmericaMap"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:local="clr-namespace:NEA_Project.Pages"
        mc:Ignorable="d"
        d:DesignHeight="450" d:DesignWidth="800">
         <Grid>
      <!--Databinding to the view model: ComboBox, that the user can type into and search for countries, as
well as textblocks that display
      the information of the selected country, a button that triggers a search and a button that returns to
home, as well as an image
      with its source that is also bound to the ImageFilepath in the ViewModel -->
      <StackPanel>
        <Image Source="{Binding ImageFilePath}" Height="325" Width="309"/>
        <StackPanel Orientation="Horizontal">
         <ComboBox
          IsEditable="true"
          Width ="200"
          Text="{Binding UserInput}"
          ItemsSource="{Binding Countries}"
          SelectedValue="{Binding UserInput, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"
                  />
         <Button Content="Search Country" Command="{Binding SearchButtonCommand}"/>
         <TextBlock Text="NAME" Margin="30,1,1,1"></TextBlock>
         <TextBlock Text="{Binding CountryName}" Width="100"/>
        </StackPanel>
```

```
        <StackPanel Orientation="Horizontal">
          <TextBlock Text="POPULATION "></TextBlock>
          <TextBlock Text="{Binding CountryPopulation}" Width="100"></TextBlock>
          <TextBlock Text="LAND AREA "></TextBlock>
          <TextBlock Text="{Binding CountryLandArea}" Width="100"></TextBlock>
          <TextBlock Text="POPULATION DENSITY "></TextBlock>
          <TextBlock Text="{Binding CountryDensity}" Width="100"></TextBlock>
        </StackPanel>

        <Button Content="Back To Main Map" Command="{Binding MapButtonCommand}" Width="200"/>
    </StackPanel>
      </Grid>
</UserControl>
```

CODE BEHIND:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace NEA_Project.Pages
{
        /// <summary>
        /// Interaction logic for SAmericaMap.xaml
        /// </summary>
        public partial class SAmericaMap : UserControl
        {
            public SAmericaMap()
            {
                InitializeComponent();
            }
        }
}
```

VIEWMODEL:

```
using NEA_Project.Helpers;
using SQLDatabase;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
using System.Windows.Input;

namespace NEA_Project.ViewModels
{
        public class SAmericaMapViewModel:ObservableObject
        {
            //Initialise
            private MainWindowViewModel _parent;
            public Database SAmerica { get; }
            private string _countryName;
            private string _countryPopulation;
            private string _countryLandArea;
            private string _countryDensity;
            private string _imageFilePath;
            private string _userInput;
            private List<string> _countries = new List<string>();
            public ICommand SearchButtonCommand { get; }
            public ICommand MapButtonCommand { get; }

            //Constructor
            public SAmericaMapViewModel(MainWindowViewModel parent)
            {
                _parent = parent;
                //assigns the image file path by passing the relevant image name through the
        GetImageFilePath from the MainWindowViewModel.
                _imageFilePath = _parent.GetImageFilePath("SAmericaMap");
                //will only populate database if its empty
                if (_parent.Database.GetSize("SouthAmerica", "ID", "") == 0)
                {
                    _parent.PopulateCountriesDatabase("SouthAmerica");
                }
                PopulateList();
                SearchButtonCommand = new SimpleCommand(_ => SearchButtonClickedCommand());
                MapButtonCommand = new SimpleCommand(_ => GoToMapPageCommand());
            }
            //Assigns the image file path
            public string ImageFilePath { get => _imageFilePath; }

            //assigning public to private, allows them to change if updated whilst the program is running
        (RaiseAndSetIfChanged from Observable Objects.)
            public string CountryName { get => _countryName; set { RaiseAndSetIfChanged(ref
        _countryName, value); } }
            public string CountryPopulation { get => _countryPopulation; set { RaiseAndSetIfChanged(ref
        _countryPopulation, value); } }
            public string CountryLandArea { get => _countryLandArea; set { RaiseAndSetIfChanged(ref
        _countryLandArea, value); } }
            public string CountryDensity { get => _countryDensity; set { RaiseAndSetIfChanged(ref
        _countryDensity, value); } }
            public string UserInput { get => _userInput; set { RaiseAndSetIfChanged(ref _userInput,
        value); } }
            public List<string> Countries { get => _countries; }

            //Once a user has selected a Country, the database will be read and retrieves the information
            public void GetCountryInfo()
            {
```

```
            List<string> CountryInfo = _parent.Database.ReadData("SouthAmerica", "CountryName,
        Population, LandArea, Density", $"CountryName LIKE '{UserInput}'", 4);
            if (CountryInfo.Count > 0)
            {
              CountryName = CountryInfo[0];
              CountryPopulation = CountryInfo[1];
              CountryLandArea = CountryInfo[2];
              CountryDensity = CountryInfo[3];
            }
        }

        //when the button is clicked GetCountryInfo is called
        public void SearchButtonClickedCommand()
        {
            GetCountryInfo();
        }
        //In order to populate the ComboBox, there must be a list of all the possible countries saved,
        this reads the database and saves all the names to the list.
        private void PopulateList()
        {

            for (int i = 0; i < _parent.Database.GetSize("SouthAmerica", "ID",""); i++)
            {
              string country = _parent.Database.ReadData("SouthAmerica", "CountryName", $"ID = {i}",
        1)[0];
              _countries.Add(country);
            }

        }
        //when the button is pressed, the method in MainWindowViewModel is called to change the
        page.

        private void GoToMapPageCommand()
        {
            _parent.ChangeToContinentsMap();
        }

    }
}
```

OCEANIA MAP CODE:

XAML:

```
<UserControl x:Class="NEA_Project.Pages.OceaniaMap"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:local="clr-namespace:NEA_Project.Pages"
xmlns:viewmodels="clr-namespace:NEA_Project.ViewModels" d:DataContext="{d:DesignInstance
Type=viewmodels:OceaniaMapViewModel}"
        mc:Ignorable="d"
        d:DesignHeight="450" d:DesignWidth="800">
        <Grid>
```

```
        <!--Databinding to the view model: ComboBox, that the user can type into and search for countries, as
well as textblocks that display
        the information of the selected country, a button that triggers a search and a button that returns to
home, as well as an image
        with its source that is also bound to the ImageFilepath in the ViewModel -->
        <StackPanel>
            <Image Source="{Binding ImageFilePath}" Height="325" Width="771"/>
            <StackPanel Orientation="Horizontal">
             <ComboBox
              IsEditable="true"
              Width ="200"
              Text="{Binding UserInput}"
              ItemsSource="{Binding Countries}"
              SelectedValue="{Binding Path=UserInput,Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
                        />
             <Button Content="Search Country" Command="{Binding SearchButtonCommand}"></Button>
             <TextBlock Text="NAME" Margin="30,1,1,1"></TextBlock>
             <TextBlock Text="{Binding CountryName}" Width="100"/>
            </StackPanel>
            <StackPanel Orientation="Horizontal">
             <TextBlock Text="POPULATION "></TextBlock>
             <TextBlock Text="{Binding CountryPopulation}" Width="100"></TextBlock>
             <TextBlock Text="LAND AREA "></TextBlock>
             <TextBlock Text="{Binding CountryLandArea}" Width="100"></TextBlock>
             <TextBlock Text="POPULATION DENSITY "></TextBlock>
             <TextBlock Text="{Binding CountryDensity}" Width="100"></TextBlock>

            </StackPanel>
            <Button Content="Back To Main Map" Command="{Binding MapButtonCommand}" Width="200"/>
        </StackPanel>
            </Grid>
</UserControl>
```

CODE BEHIND:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace NEA_Project.Pages
{
        /// <summary>
        /// Interaction logic for OceaniaMap.xaml
        /// </summary>
        public partial class OceaniaMap : UserControl
```

```
        {
            public OceaniaMap()
            {
                InitializeComponent();
            }
        }
}
```

VIEWMODEL:

```
using NEA_Project.Helpers;
using SQLDatabase;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Input;

namespace NEA_Project.ViewModels
{
        public class OceaniaMapViewModel : ObservableObject
        {
            //Initialise
            private MainWindowViewModel _parent;
            public Database Oceania { get; }
            private string _countryName;
            private string _countryPopulation;
            private string _countryLandArea;
            private string _countryDensity;
            private string _imageFilePath;
            private string _userInput;
            private List<string> _countries = new List<string>();
            public ICommand SearchButtonCommand { get; }
            public ICommand MapButtonCommand { get; }
            //Constructor
            public OceaniaMapViewModel(MainWindowViewModel parent)
            {
                _parent = parent;
                //assigns the image file path by passing the relevant image name through the
        GetImageFilePath from the MainWindowViewModel.
                _imageFilePath = _parent.GetImageFilePath("OceaniaMap");
                //will only populate database if its empty
                if (_parent.Database.GetSize("Oceania", "ID", "") == 0)
                {
                    _parent.PopulateCountriesDatabase("Oceania");
                }
                PopulateList();
                SearchButtonCommand = new SimpleCommand(_ => SearchButtonClickedCommand());
                MapButtonCommand = new SimpleCommand(_ => GoToMapPageCommand());
            }
            //Assigns the image file path
            public string ImageFilePath { get => _imageFilePath; }

            //assigning public to private, allows them to change if updated whilst the program is running
        (RaiseAndSetIfChanged from Observable Objects.)
```

```
        public string CountryName { get => _countryName; set { RaiseAndSetIfChanged(ref
    _countryName, value); } }
        public string CountryPopulation { get => _countryPopulation; set { RaiseAndSetIfChanged(ref
    _countryPopulation, value); } }
        public string CountryLandArea { get => _countryLandArea; set { RaiseAndSetIfChanged(ref
    _countryLandArea, value); } }
        public string CountryDensity { get => _countryDensity; set { RaiseAndSetIfChanged(ref
    _countryDensity, value); } }
        public string UserInput { get => _userInput; set { RaiseAndSetIfChanged(ref _userInput,
    value); } }
        public List<string> Countries { get => _countries; }

        //Once a user has selected a Country, the database will be read and retrieves the information
        public void GetCountryInfo()
        {
            List<string> CountryInfo = _parent.Database.ReadData("Oceania", "CountryName,
    Population, LandArea, Density", $"CountryName LIKE '{UserInput}'", 4);
            if (CountryInfo.Count > 0)
            {
              CountryName = CountryInfo[0];
              CountryPopulation = CountryInfo[1];
              CountryLandArea = CountryInfo[2];
              CountryDensity = CountryInfo[3];
            }
        }

        //when the button is clicked GetCountryInfo is called
        public void SearchButtonClickedCommand()
        {
            GetCountryInfo();
        }
        //In order to populate the ComboBox, there must be a list of all the possible countries saved,
    this reads the database and saves all the names to the list.
        private void PopulateList()
        {

            for (int i = 0; i < _parent.Database.GetSize("Oceania", "ID", ""); i++)
            {
              string country = _parent.Database.ReadData("Oceania", "CountryName", $"ID = {i}", 1)[0];
              _countries.Add(country);
            }

        }
        //when the button is pressed, the method in MainWindowViewModel is called to change the
    page.
        private void GoToMapPageCommand()
        {
            _parent.ChangeToContinentsMap();
        }
    }
}
```

GAME MENU CODE:

XAML:

```
<UserControl x:Class="NEA_Project.Pages.GameMenuPage"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
```

```
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:local="clr-namespace:NEA_Project.Pages"
        mc:Ignorable="d"
        d:DesignHeight="450" d:DesignWidth="800">
         <Grid>
    <!--Data binding with viewmodel, a ComboBox which displays all possible question bank names
available to that user as well as
    buttons for navigation and refresh button that repopulates the list that is bound to the ComboBox-->
    <StackPanel>
      <TextBlock Text="HI THIS IS THE MENU :D"/>
      <TextBlock Text="QUESTION:"/>
      <ComboBox
        Width ="200"
        ItemsSource="{Binding Path=QuestionBanks}"
        SelectedValue="{Binding Path=selectedValue,Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
                />
      <Button Content = "Quiz" Height = "30" Width = "130"
        Command="{Binding QuizButtonClickedCommand}"/>
      <Button Content = "Pairs" Height = "30" Width = "130"
        Command="{Binding PairsGameButtonClickedCommand}"/>
      <Button Content = "Word Scramble" Height = "30" Width = "130"
        Command="{Binding WordScrambleButtonClickedCommand}"/>

      <Button Content = "HOME PAGE" Height = "30" Width = "130"
        Command="{Binding HomeButtonClickedCommand}"/>
      <Button Content = "REFRESH" Height = "30" Width = "130"
        Command="{Binding RefreshButtonClickedCommand}"/>
    </StackPanel>
        </Grid>
</UserControl>
```

CODE BEHIND:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
namespace NEA_Project.Pages
{
        /// <summary>
        /// Interaction logic for GameMenuPage.xaml
        /// </summary>
        public partial class GameMenuPage : UserControl
        {
```

```
        public GameMenuPage()
        {
            InitializeComponent();
        }
    }
}
```

VIEWMODEL:

```
using NEA_Project.Helpers;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Input;

namespace NEA_Project.ViewModels
{
    public class GameMenuViewModel:ObservableObject
    {
        //Initialise
        MainWindowViewModel _parent;
        public ICommand HomeButtonClickedCommand { get; }
        public ICommand PairsGameButtonClickedCommand { get; }
        public ICommand WordScrambleButtonClickedCommand { get; }
        public ICommand QuizButtonClickedCommand { get; }
        public ICommand RefreshButtonClickedCommand { get; }

        private ObservableCollection<string> _questionBanks = new ObservableCollection<string>();

        //Constructor
        public GameMenuViewModel(MainWindowViewModel Parent)
        {
            _parent = Parent;
            HomeButtonClickedCommand = new SimpleCommand(_ => HomeButtonClicked());
            PairsGameButtonClickedCommand = new SimpleCommand(_ =>
    PairsGameButtonClicked());
            WordScrambleButtonClickedCommand = new SimpleCommand(_ =>
    WordScrambleButtonClicked());
            QuizButtonClickedCommand = new SimpleCommand(_ => QuizButtonClicked());
            RefreshButtonClickedCommand = new SimpleCommand(_ => RefreshButtonClicked());
        }
        public ObservableCollection<string> QuestionBanks { get => _questionBanks; set {
    RaiseAndSetIfChanged(ref _questionBanks, value); } }

        public string selectedValue { get => _parent.CurrentQuestionBank;
            set { _parent.CurrentQuestionBank = value; } }

        //In order to populate the ComboBox, there must be a list of all the question banks saved, this
    reads the database
        //and saves all the names to the list.
        private void populateList()
        {
            List<string> hi = _parent.Database.ReadData("QuestionBanks", "BankName", $"USERID =
    {_parent.UserID} OR UserID = 0", 1);
```

```
                foreach (string s in hi)
                {
                 if (!(_questionBanks.Contains(s)))
                 { _questionBanks.Add(s); }
                }

            }
            //when button is clicked, populateList is called.
            private void RefreshButtonClicked()
            {
                populateList();
            }
                    //when the button is pressed, the method in MainWindowViewModel is called to change
        the page.
            private void HomeButtonClicked()
            {
                _parent.ChangeToHomePage();
            }
            //when the button is pressed, the method in MainWindowViewModel is called to change the
        page.
            private void PairsGameButtonClicked()
            {
                _parent.ChangeToPairsGamePage();
            }
            //when the button is pressed, the method in MainWindowViewModel is called to change the
        page.
            private void WordScrambleButtonClicked()
            {
                _parent.ChangeToWordScramblePage();
            }
            //when the button is pressed, the method in MainWindowViewModel is called to change the
        page.
            private void QuizButtonClicked()
            {
                _parent.ChangeToQuizPage();
            }
        }
}
```

PAIRS GAME CODE:

Based on SingletonSean's drag and drop

XAML:

```
<UserControl x:Class="NEA_Project.Pages.PairsGamePage"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:local="clr-namespace:NEA_Project.Pages"
        mc:Ignorable="d"
        d:DesignHeight="450" d:DesignWidth="800">
         <Grid>
      <!--Data binding to the view model: finish button and refresh button, TextBox that the user enters how
many pairs they'd like
      and a Score textblock that outputs the user's score. A canvas is also defined, which is the area where
the
```

```
        drag and drop game takes place -->
        <Grid.RowDefinitions>
            <RowDefinition Height="1*"/>
            <RowDefinition Height="5*"/>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="1*"/>
        </Grid.ColumnDefinitions>
        <StackPanel Grid.Row="0" Orientation="Horizontal" Background="LightSeaGreen">
            <Button Content="Finish :)" Height="30" Width="80" Command="{Binding
FinishButtonCommand}"/>
            <TextBlock Text=" Enter how many Pairs you'd like: " VerticalAlignment ="Center"/>
            <TextBox Name="NumOfPairs" VerticalAlignment ="Center" Width="30" Height="30" Text="{Binding
HowManyPairs, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"/>
            <Button Content="Refresh" Height="32" Width="50" Margin="10,0"
Click="RefreshButton_Click"></Button>
            <TextBlock Text="SCORE" VerticalAlignment ="Center"/>
            <TextBlock Name="Score" VerticalAlignment ="Center" HorizontalAlignment="Right" Width="200"
Text="{Binding Score, UpdateSourceTrigger=PropertyChanged}"/>
        </StackPanel>
        <Canvas x:Name="canvas" Background="White" AllowDrop="True" DragOver="canvas_DragOver"
Drop="canvas_Drop" Grid.Row="1" Width="800" Height="375">

        </Canvas>
            </Grid>
</UserControl>
```

CODE BEHIND:

```
using NEA_Project.ViewModels;
using NEA_Project.Commands;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace NEA_Project.Pages
{
            /// <summary>
            /// Interaction logic for PairsGamePage.xaml
            /// </summary>
            public partial class PairsGamePage : UserControl
            {

        //Declaring the Dependency Properties:
        public static readonly DependencyProperty CheckPairCommandProperty =
DependencyProperty.Register("CheckPairCommand",
            typeof(ICommand), typeof(PairsGamePage), new PropertyMetadata(null));

        public ICommand CheckPairCommand
        {
```

```
        get { return (ICommand)GetValue(CheckPairCommandProperty); }
        set { SetValue(CheckPairCommandProperty, value); }
    }

    public static readonly DependencyProperty ParentProperty =
        DependencyProperty.Register("ParentVM", typeof(MainWindowViewModel),
typeof(PairsGamePage),
         new FrameworkPropertyMetadata(null,
FrameworkPropertyMetadataOptions.BindsTwoWayByDefault));

          public MainWindowViewModel ParentVM
    {
        get { return (MainWindowViewModel)GetValue(ParentProperty); }
        set { SetValue(ParentProperty, value); }
    }


    public List<TextBlock> textblocks = new List<TextBlock>();
        public int whichElement;
    //Constructor
    public PairsGamePage()
    {
        InitializeComponent();
    }

    public static readonly DependencyProperty TextBlockContainsProperty =
        DependencyProperty.Register("TextBlockContains", typeof(string), typeof(PairsGamePage),
         new FrameworkPropertyMetadata(string.Empty,
FrameworkPropertyMetadataOptions.BindsTwoWayByDefault));

    public string TextBlockContains
    {
        get { return (string)GetValue(TextBlockContainsProperty); }
        set { SetValue(TextBlockContainsProperty, value); }
    }


    public static readonly DependencyProperty TextBlock2ContainsProperty =
        DependencyProperty.Register("TextBlock2Contains", typeof(string), typeof(PairsGamePage),
         new FrameworkPropertyMetadata(string.Empty,
FrameworkPropertyMetadataOptions.BindsTwoWayByDefault));

    public string TextBlock2Contains
    {
        get { return (string)GetValue(TextBlock2ContainsProperty); }
        set { SetValue(TextBlock2ContainsProperty, value); }
    }

    public static readonly DependencyProperty PairFoundProperty =
        DependencyProperty.Register("PairFound", typeof(bool), typeof(PairsGamePage),
         new FrameworkPropertyMetadata(false,
FrameworkPropertyMetadataOptions.BindsTwoWayByDefault));

    public bool PairFound
    {
        get { return (bool)GetValue(PairFoundProperty); }
        set { SetValue(PairFoundProperty, value); }
    }

    //populate the canvas with textboxes with questions and answers.
    private void CreateTextBlocks()
    {

        Random random = new Random();
        //Databinding for the text blocks
```

```
PairsGameViewModel myDataObject = new PairsGameViewModel(ParentVM);
Binding Question = new Binding("Question");
Question.Source = myDataObject;
Binding Answer = new Binding("Answer");
Answer.Source = myDataObject;
int Pairs = 0;
bool validNumber = false;
if (NumOfPairs.Text != null)
{
 //only happens if the user inputs in the text box
 try
 {
            Pairs = Int32.Parse(NumOfPairs.Text);
            validNumber = true;
            //must be an int
  }
 catch (Exception)
 {

            MessageBox.Show("Not the right data type!");
 }
}


if (Pairs != 0 && validNumber)
{
 //if the user input more that 0 and it is a valid number
 for (int i = 0; i < Pairs; i++)
 {
            int height = (int)canvas.Height;
            int width = (int)canvas.Width;
            myDataObject.Change = true;
            //gets random number based on the dimension of the canvas
            int randomLeftTB = random.Next(0, width);
            int randomTopTB = random.Next(0, height);
            int randomLeftTB1 = random.Next(0, width);
            int randomTopTB1 = random.Next(0, height);
            TextBlock textblock = new TextBlock();
            TextBlock textblock1 = new TextBlock();
            //binds the textboxes content to Question and Answer in the ViewModel.
      textblock.SetBinding(TextBlock.TextProperty, Question);
      textblock1.SetBinding(TextBlock.TextProperty, Answer);
            //gives the textboxes definitive heights (which helps with the HitBoxes)
            textblock.Width = 75;
            textblock1.Width = 75;
            textblock1.Height = 75;
            textblock.Height = 75;
      canvas.Children.Add(textblock);
      canvas.Children.Add(textblock1);
            textblocks.Add(textblock);
            textblocks.Add(textblock1);
            //Randomises the position of the textblocks
            Canvas.SetLeft(textblock, randomLeftTB);
            Canvas.SetTop(textblock, randomTopTB);
            Canvas.SetLeft(textblock1, randomLeftTB1);
            Canvas.SetTop(textblock1, randomTopTB1);
            myDataObject.Change = false;
 }

 //Assigns the MouseMove Method to each textblock.
 for (int i = 0; i < textblocks.Count; i++)
 {
            textblocks[i].MouseMove += MouseMove;
 }
}
```

```
    }

    private void GetElement()
    {
        for (int i = 0; i < textblocks.Count; i++)
        {
         if (textblocks[i].IsMouseOver)
         {
                    //checks which element (textblock) is currently being clicked on
                    whichElement = i;
         }

        }
    }


    //event handler, changes the textblock's position based on the mouse's position
    private void canvas_DragOver(object sender, DragEventArgs e)
    {
        Point dropPosition = e.GetPosition(canvas);
        GetElement();
        Canvas.SetLeft(textblocks[whichElement], dropPosition.X);
        Canvas.SetTop(textblocks[whichElement], dropPosition.Y);
    }


    //event handler, when the left mouse button is pressed, it starts the drag and drop proccess
    private void MouseMove(object sender, MouseEventArgs e)
    {
        GetElement();
        if (e.LeftButton == MouseButtonState.Pressed)
        {
            DragDrop.DoDragDrop(textblocks[whichElement], textblocks[whichElement],
DragDropEffects.Move);

        }
    }

    //triggered when an item is dropped on the canvas, creates a hit box for the dragged item and then
iterates through all the other
    //textblocks, seeing if their hit box intersects with the dragged textblock, if they do the
CheckPairCommand is called and if its
    // a pair, then the textblocks are removed and the canvas is updated. if the canvas count is 1 then the
user is displayed a message
    // to let them know all pairs have been found
    private void canvas_Drop(object sender, DragEventArgs e)
    {
        Rect DraggedHitBox = new Rect(Canvas.GetLeft(textblocks[whichElement]),
Canvas.GetTop(textblocks[whichElement]), textblocks[whichElement].Width,
textblocks[whichElement].Height);

        for (int i = 0; i < textblocks.Count -1; i++)
            {
         if (textblocks[whichElement] != textblocks[i])
         {

                    Rect PairedHitBox = new Rect(Canvas.GetLeft(textblocks[i]),
Canvas.GetTop(textblocks[i]), textblocks[i].Width, textblocks[i].Height);
            if (DraggedHitBox.IntersectsWith(PairedHitBox))
                {
                TextBlockContains = textblocks[i].Text;
                TextBlock2Contains = textblocks[whichElement].Text;
                CheckPairCommand?.Execute(null);
```

```
                    if (PairFound)
                    {
                    TextBlock temp1 = textblocks[i];
                    TextBlock temp2 = textblocks[whichElement];
                    canvas.Children.Remove(temp1);
                     canvas.Children.Remove(temp2);
                     textblocks.Remove(temp1);
                     textblocks.Remove(temp2);
                    this.UpdateLayout();
                       }
                    if (canvas.Children.Count == 1)
                    {
                    MessageBox.Show("Finished :)");
                    }

                    }
            }

          }

      }

      //when the button is pressed, all the text boxes are removed and then new textblocks are created
      private void RefreshButton_Click(object sender, RoutedEventArgs e)
      {
         for (int i = textblocks.Count - 1; i >= 0; i--)
               {
            canvas.Children.Remove(textblocks[i]);
          textblocks.RemoveAt(i);
         }
         CreateTextBlocks();

      }
          }
}
```

VIEWMODEL:

```
using NEA_Project.Commands;
using NEA_Project.Helpers;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Input;

namespace NEA_Project.ViewModels
{
          public class PairsGameViewModel : ObservableObject
          {
                //initialise
                MainWindowViewModel _parent;
                private string _howManyPairs;
                public ICommand CheckPairCommand { get; }
                public ICommand FinishButtonCommand { get; }
                private int _score = 0;
                private string _question = "";
                private string _answer = "";
                private bool _pairFound = false;
                Random random = new Random();
                int randomNum { get; set; }
```

```
        private bool _change = false;
        //constructor
        public PairsGameViewModel(MainWindowViewModel Parent)
        {
            //as the codebehind is tightly coupled, this prevents _parent being overwritten
            if (_parent == null)
            {
             _parent = Parent;
            }


            CheckPairCommand = new CheckPairCommand(this, _parent);
            FinishButtonCommand = new SimpleCommand(_ => FinishButtonClicked());
        }

        //Most of these public properties are part of the dependency properties of the code behind
        public MainWindowViewModel ParentVM { get { return _parent; }
            set
            {
             _parent = value;
               OnPropertyChanged(nameof(_parent));
            }
        }

        public string HowManyPairs
        {
            get => _howManyPairs;
            set
            {
             //value = "pls";
             RaiseAndSetIfChanged(ref _howManyPairs, value);
            }
        }

        public bool Change
        {
            get => _change;
            set
            {
             RaiseAndSetIfChanged(ref _change, value);
             if (Change)
             {
             //Everytime Change is assigned true (Happens in the code behind), it will get a new
question and answer.
                        _question = GetQuestion();
                        _answer = GetAnswer();
                    }
            }
        }

        public bool PairFound
        {
            get => _pairFound;
            set
            {
```

```
            RaiseAndSetIfChanged(ref _pairFound, value);
        }
    }

    public string Question
    {
        get => _question;
    }

    public string Answer
    {
        get => _answer;
    }


    private string _textBlockContains;
    public string TextBlockContains
    {
        get
        {
         return _textBlockContains;
        }
        set
        {
         RaiseAndSetIfChanged(ref _textBlockContains, value);
        }
    }


    private string _textBlock2Contains;
    public string TextBlock2Contains
    {
        get
        {
         return _textBlock2Contains;
        }
        set
        {
         RaiseAndSetIfChanged(ref _textBlock2Contains, value);

        }
    }

    public int Score
    {
        get => _score;
        set { RaiseAndSetIfChanged(ref _score, value); ; }
    }

    //when generate question button is clicked, the method first distinguishes which question bank
they are using, if its the default
    // the ID needs to change because of the way it is saved in the database
    //then a random question is read from the database.

    public string GetQuestion()
    {
```

```
            int ID;
            if (_parent.CurrentQuestionBank == "Default")
            {
              ID = 0;
            }
            else
            {
              ID = _parent.UserID;
            }
            randomNum = random.Next(1, _parent.Database.GetSize("QuestionBanks", "QuestionID",
$"WHERE BankName = '{_parent.CurrentQuestionBank}' AND UserID = {ID}") + 1);

            return _parent.Database.ReadData("QuestionBanks", "Question", $"QuestionID =
{randomNum} AND BankName = '{_parent.CurrentQuestionBank}' AND UserID = {ID}",1)[0];
        }

        //Returns the answer for the current question first assigns ID depending on whether
        // the current question bank is the "Default" bank or not. Then, it queries the database to
retrieve the answer
        public string GetAnswer()
        {
            int ID;
            if (_parent.CurrentQuestionBank == "Default")
            {
              ID = 0;
            }
            else
            {
              ID = _parent.UserID;
            }

            return _parent.Database.ReadData("QuestionBanks", "Answer", $"Question LIKE
'{_question}' AND QuestionID = {randomNum} AND BankName = '{_parent.CurrentQuestionBank}'
AND UserID = {ID}",1)[0];
        }


        //When the finish button is clicked, the highscore database is read and the existing score is
saved
        //this is then compared to the new score, if the new score is greater, it replaces the old score
and
        // the page is changed to the Game Menu.
        private void FinishButtonClicked()
        {
            int existingScore = 0;
            try
            {
              string DatabaseScore = _parent.Database.ReadData("UserStats", "HighScore2",
$"USERID = {_parent.UserID}", 1)[0];
              existingScore = Int32.Parse(DatabaseScore);
            }
            catch (Exception)
            {


            }
```

```
                if (existingScore < Score)
                {
                  _parent.Database.UpdateData("UserStats", $"HighScore2 = {Score}", $"USERID =
        {_parent.UserID}");
                }
                Score = 0;
                _parent.ChangeToGameMenuPage();
            }
        }
}
```

QUIZ CODE:

XAML:

```
<UserControl x:Class="NEA_Project.Pages.QuizPage"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
         xmlns:local="clr-namespace:NEA_Project.Pages"
         mc:Ignorable="d"
         d:DesignHeight="450" d:DesignWidth="800">

        <Grid>
    <!--Databinding with viewmodel, Textblocks displays the question and the score, a textbox allow the
user to input their answer
    and 3 buttons: 1 that generate a question, 1 that checks answer and one that goes back to game
menu-->
    <StackPanel>
      <TextBlock Text="HI THIS IS SIGN UP :D"/>
      <StackPanel Orientation="Horizontal" Margin="0,20,0,20">
       <TextBlock Text="QUESTION:"/>
       <TextBlock VerticalAlignment ="Center" Width="200" Text="{Binding UserNameInput,
UpdateSourceTrigger=PropertyChanged}"/>
        <TextBlock VerticalAlignment ="Center" HorizontalAlignment="Right" Width="200" Text="{Binding
Score, UpdateSourceTrigger=PropertyChanged}"/>

      </StackPanel>
      <StackPanel Orientation="Horizontal" Margin="0,20,0,20">
       <TextBlock Text="ANSWER:"/>
       <TextBox VerticalAlignment ="Center" Width="200" Text="{Binding PasswordInput,
Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"/>
      </StackPanel>
      <Button Content = "Generate Question" Height = "30" Width = "120"
         Command="{Binding VerifyButtonClickedCommand}"/>
      <Button Content = "Check Answer" Height = "30" Width = "80"
         Command="{Binding CheckAnswerCommand}"/>
      <Button Content="Finish :)" Height="30" Width="80" Command="{Binding
FinishButtonCommand}"/>
    </StackPanel>
        </Grid>
</UserControl>
```

CODE BEHIND:

```
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace NEA_Project.Pages
{
        /// <summary>
        /// Interaction logic for QuizPage.xaml
        /// </summary>
        public partial class QuizPage : UserControl
        {
            public QuizPage()
            {
                InitializeComponent();
            }
        }
}
```

VIEWMODEL:

```
using NEA_Project.Helpers;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Input;

namespace NEA_Project.ViewModels
{
        public class QuizViewModel : ObservableObject
        {
            //Initialise
            MainWindowViewModel _parent;
            private string _question = "";
            private string _userInput = "";
            private int _score = 0;
            public Random random = new Random();

            public ICommand VerifyButtonClickedCommand { get; }
            public ICommand CheckAnswerCommand { get; }
            public ICommand FinishButtonCommand { get; }
            //Constructor
            public QuizViewModel(MainWindowViewModel Parent)
            {
                _parent = Parent;
```

```
            VerifyButtonClickedCommand = new SimpleCommand(_ =>
GenerateQuestionButtonClicked());
            CheckAnswerCommand = new SimpleCommand(_ => CheckAnswer());
            FinishButtonCommand = new SimpleCommand(_ => FinishButtonClicked());
        }
        public string Question
        {
            get => _question;
            set
            {
             RaiseAndSetIfChanged(ref _question, value);
            }
        }

        public string UserInput
        {
            get => _userInput;
            set
            {
             RaiseAndSetIfChanged(ref _userInput, value);
            }
        }

        public int Score
        {
            get => _score;
            set { RaiseAndSetIfChanged(ref _score, value); }
        }


        private void GenerateQuestionButtonClicked()
        {

            Question = GetQuestion();

        }

        //When the finish button is clicked, the highscore database is read and the existing score is
saved
        //this is then compared to the new score, if the new score is greater, it replaces the old score
and
        // the page is changed to the Game Menu.
        private void FinishButtonClicked()
        {
            int NumberOfScores = _parent.Database.GetSize("QuizScores", "UserID", $"WHERE
UserID = {_parent.UserID}");
            NumberOfScores += 1;
            _parent.Database.InsertData("QuizScores", "QuizID, UserID, Score",
$"{NumberOfScores},{_parent.UserID},'{Score}'");
            Score = 0;
            _parent.ChangeToGameMenuPage();
        }

        //when the check answer button is pressed, the user's input is compared with the actual
answer
        //the score is adjusted as fit and a message is displayed
```

```
        public void CheckAnswer()
        {

            string answer = UserInput.Trim();
            List<string> CorrectAnswer = _parent.Database.ReadData("QuestionBanks", "Answer",
$"Question LIKE '{Question}' AND BankName = '{_parent.CurrentQuestionBank}' AND (UserID =
{_parent.UserID} OR UserID = 0)",1);
            //Error checking to avoid the program crashing in case the question has not been
generated.
            if (CorrectAnswer.Count > 0) {
             if (answer == CorrectAnswer[0].Trim())
             {
                MessageBox.Show("yay!");
                    Score += 1;
             }
             else
             {
                    MessageBox.Show($"neigh </3 {CorrectAnswer[0]}");
                    Score -= 1;
             }
            }
                 else
            {
             MessageBox.Show("Please generate a question first!!!");
            }

            GetQuestion();
        }

        //when generate question button is clicked, the method first distinguishes which question bank
they are using, if it's the default
        // the ID needs to change because of the way it is saved in the database
        //then a random question is read from the database.
        public string GetQuestion()
        {
            int ID;
            if (_parent.CurrentQuestionBank == "Default")
            {
             ID = 0;
            }
            else
            {
             ID = _parent.UserID;
            }
            int question = random.Next(1, _parent.Database.GetSize("QuestionBanks", "QuestionID",
$"WHERE BankName = '{_parent.CurrentQuestionBank}' AND UserID = {ID}") + 1);
            return _parent.Database.ReadData("QuestionBanks", "Question", $"QuestionID =
{question} AND BankName = '{_parent.CurrentQuestionBank}' AND UserID = {ID}", 1)[0];

        }
    }
}
```

WORD SCRAMBLE CODE:

XAML:

```
<UserControl x:Class="NEA_Project.Pages.WordScramblePage"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:local="clr-namespace:NEA_Project.Pages"
             mc:Ignorable="d"
             d:DesignHeight="450" d:DesignWidth="800">
              <Grid>
        <!--Data binding with view model: Textblock linked to a scrambled country, text box that allows user to input their answer,
        and 5 buttons: 1) generates question 2 & 3) give hints, 4) Checks answer, 5) go back the game menu-->
        <StackPanel>
            <TextBlock Text="Scrambled Country :D"/>
            <StackPanel Orientation="Horizontal" Margin="0,20,0,20">
             <TextBlock Text="Scrambled Country:"/>
             <TextBlock VerticalAlignment ="Center" Width="200" Text="{Binding ScrambledWord, UpdateSourceTrigger=PropertyChanged}"/>
              <TextBlock VerticalAlignment ="Center" HorizontalAlignment="Right" Width="200" Text="{Binding Score, UpdateSourceTrigger=PropertyChanged}"/>
            </StackPanel>
            <StackPanel Orientation="Horizontal" Margin="0,20,0,20">
             <TextBlock Text="ANSWER:"/>
             <TextBox VerticalAlignment ="Center" Width="200" Text="{Binding UserInput, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"/>
            </StackPanel>
            <Button Content = "Generate Question" Height = "30" Width = "120"
                Command="{Binding GenerateQuestionButtonClickedCommand}"/>
            <Button Content = "Hint1" Height = "30" Width = "120"
                Command="{Binding Hint1ButtonClickedCommand}"/>
            <Button Content = "Hint2" Height = "30" Width = "120"
                Command="{Binding Hint2ButtonClickedCommand}"/>
            <Button Content = "Check Answer" Height = "30" Width = "80"
                Command="{Binding CheckAnswerCommand}"/>
            <Button Content="Finish :)" Height="30" Width="80" Command="{Binding FinishButtonCommand}"/>
        </StackPanel>
              </Grid>
</UserControl>
```

CODE BEHIND:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
```

```
using System.Windows.Shapes;

namespace NEA_Project.Pages
{
        /// <summary>
        /// Interaction logic for WordScramblePage.xaml
        /// </summary>
        public partial class WordScramblePage : UserControl
        {
            public WordScramblePage()
            {
                InitializeComponent();
            }
        }
}
```

VIEWMODEL:

```
using NEA_Project.Helpers;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Input;

namespace NEA_Project.ViewModels
{
        public class WordScrambleViewModel : ObservableObject
        {
            //Initialise
            MainWindowViewModel _parent;
            private string _answer;
            private string _userInput;
            private string _continent;
            private string _scrambledWord;
            private string _userNameInput = "Enter Your Username Here...";
            private string _passwordInput = "Enter Your Password Here...";
            private int _score = 0;
            public int UserID = 1;
            public Random random = new Random();

            public ICommand GenerateQuestionButtonClickedCommand { get; }
            public ICommand CheckAnswerCommand { get; }
            public ICommand FinishButtonCommand { get; }

            public ICommand Hint1ButtonClickedCommand { get; }
            public ICommand Hint2ButtonClickedCommand { get; }
            //Constructor
            public WordScrambleViewModel(MainWindowViewModel Parent)
            {
                _parent = Parent;
                GenerateQuestionButtonClickedCommand = new SimpleCommand(_ =>
        GenerateQuestionButtonClicked());
                CheckAnswerCommand = new SimpleCommand(_ => CheckAnswer());
```

```
            FinishButtonCommand = new SimpleCommand(_ => FinishButtonClicked());
            Hint1ButtonClickedCommand = new SimpleCommand(_=> Hint1ButtonClicked());
            Hint2ButtonClickedCommand = new SimpleCommand((_ => Hint2ButtonClicked()));
        }

        public string Answer { get { return _answer; } set {  _answer = value; } }

        //RaiseAndSetIfChanged -> Allows the variables to update
        public string ScrambledWord
        {
            get => _scrambledWord;
            set
            {
             RaiseAndSetIfChanged(ref _scrambledWord, value);
            }
        }

        public string UserInput
        {
            get => _userInput;
            set
            {

             RaiseAndSetIfChanged(ref _userInput, value);
            }
        }

        public int Score
        {
            get => _score;
            set { RaiseAndSetIfChanged(ref _score, value); }
        }


        private void GenerateQuestionButtonClicked()
        {

            GetAnagram();

        }
        //When the Finished button is clicked, the new score is compared to the
        // existing one in the database and if it is greater the database is
        // updated and the page changes to the home page
        private void FinishButtonClicked()
        {
            int NumberOfScores = _parent.Database.GetSize("WordScrambleScores", "UserID",
$"WHERE UserID = {_parent.UserID}");
            NumberOfScores += 1;
            _parent.Database.InsertData("WordScrambleScores", "WordScrambleID, UserID, Score",
$"{NumberOfScores},{_parent.UserID},'{Score}'");
            Score = 0;
            _parent.ChangeToGameMenuPage();
        }

        private void Hint1ButtonClicked()
        {
```

```
    MessageBox.Show($"This country is in: {_continent}");
}

private void Hint2ButtonClicked()
{
    MessageBox.Show($"This country starts with: {_answer[0]}");
}

//The users answer is compared to the correct
//answer and the score is updated accordingly
public void CheckAnswer()
{
    try
    {

     if (UserInput != null && Answer != null)
     {
                //checks that both user input and answer are not null
                if (UserInput.Trim().ToUpper() == Answer.Trim().ToUpper())
                {
                MessageBox.Show("yay!");
                Score += 1;
          }
                else
                {
                MessageBox.Show($"neigh </3 {Answer}");
                Score -= 1;
                }

     }

    }
    catch (Exception)
    {


    }


}


private void GetAnagram()
{
    //Gets random number and depending on the
    //number, this returns a random country
    Random random = new Random();

    int randomContinent = random.Next(6);

    string tableName = string.Empty;
    switch (randomContinent)
    {
     case 0:
          tableName = "Africa";
                break;
```

```
                case 1:
                        tableName = "Asia";
                        break;
                case 2:
                        tableName = "Europe";
                        break;
                case 3:
                        tableName = "NorthAmerica";
                        break;
                case 4:
                        tableName = "SouthAmerica";
                        break;
                case 5:
                        tableName = "Oceania";
                        break;
                default:
                        break;
            }
            _continent = tableName;
            //get the size of the database and then get a random country because
            int NoCountries = _parent.Database.GetSize($"{tableName}", "ID","");
            int RandomCountry = random.Next(NoCountries);
            List<string> GetAnswer = _parent.Database.ReadData($"{tableName}", "CountryName",
$"ID = {RandomCountry}", 1);
            if (GetAnswer.Count > 0)
            {
              _answer = GetAnswer[0].ToLower();
            }
            //Get the list of all possible combinations of the Country name
            // Gets rid of the answer from the possible list to avoid that being selected randomly
            //Randomly selects a scrambled world
            List<string> ScrambledWords = new List<string>();
            Permute(_answer.ToCharArray(), 0, _answer.Length - 1, ref ScrambledWords);
            ScrambledWords.Remove(_answer);
            int randomInt = random.Next(ScrambledWords.Count);
            ScrambledWord = ScrambledWords[randomInt];

        }

        //RECURSIVE ALGORITHM
        private void Permute(char[] arr, int start, int end, ref List<string> list)
        {
            //when a new combination is found, its added to the list
            if (start == end)
            {

              list.Add(new string(arr));
            }
            else
            {
              //cycles through the word (arr) and swaps each character
              for (int j = start; j <= end; j++)
              {
                        Swap(ref arr[start], ref arr[j]);
                    Permute(arr, start + 1, end, ref list);
                        Swap(ref arr[start], ref arr[j]);
```

```
                }
               }
            }

            static void Swap(ref char a, ref char b)
            {
               char temp = a;
               a = b;
               b = temp;
            }
        }
}
```

QUESTION BANK MENU CODE:

XAML:

```
<UserControl x:Class="NEA_Project.Pages.QuestionBankMenu"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:local="clr-namespace:NEA_Project.Pages"
             mc:Ignorable="d"
             d:DesignHeight="450" d:DesignWidth="800">
            <Grid Background="White">
      <!-- Databinding with viewmodel: buttons for navigation-->
      <StackPanel>
         <TextBlock Text="HI THIS IS THE MENU :D"/>

         <Button Content = "Create Question Bank" Height = "30" Width = "130"
            Command="{Binding CreateButtonClickedCommand}"/>
         <Button Content = "Delete Question Bank" Height = "30" Width = "130"
            Command="{Binding DeleteButtonClickedCommand}"/>
         <Button Content = "Edit Questions" Height = "30" Width = "130"
            Command="{Binding EditButtonClickedCommand}"/>
         <Button Content = "Read Questions" Height = "30" Width = "130"
            Command="{Binding ReadButtonClickedCommand}"/>
              <Button Content = "HOME PAGE" Height = "30" Width = "130"
            Command="{Binding HomeButtonClickedCommand}"/>
      </StackPanel>
          </Grid>
</UserControl>
```

CODE BEHIND:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
```

```
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace NEA_Project.Pages
{
        /// <summary>
        /// Interaction logic for QuestionBankMenu.xaml
        /// </summary>
        public partial class QuestionBankMenu : UserControl
        {
            public QuestionBankMenu()
            {
                InitializeComponent();
            }
        }
}
```

VIEWMODEL:

```
using NEA_Project.Helpers;
using System;
using System.Collections.Generic;
using System.Diagnostics.Contracts;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Input;

namespace NEA_Project.ViewModels
{
        public class QuestionBankMenuViewModel
        {
            //initialise
            private MainWindowViewModel _parent;
            public ICommand HomeButtonClickedCommand { get; }
            public ICommand CreateButtonClickedCommand { get; }
            public ICommand DeleteButtonClickedCommand { get; }
            public ICommand ReadButtonClickedCommand { get; }
            public ICommand EditButtonClickedCommand { get; }

            //constructor
            public QuestionBankMenuViewModel(MainWindowViewModel parent)
            {
                _parent = parent;
                HomeButtonClickedCommand = new SimpleCommand(_ => HomeButtonClicked());
                CreateButtonClickedCommand = new SimpleCommand(_ => CreateButtonClicked());
                DeleteButtonClickedCommand = new SimpleCommand(_ => DeleteButtonClicked());
                ReadButtonClickedCommand = new SimpleCommand(_ => ReadButtonClicked());
                EditButtonClickedCommand = new SimpleCommand(_ => EditButtonClicked());
            }

            //when the button is pressed, the method in MainWindowViewModel is called to change the
        page.
```

```
        private void HomeButtonClicked()
        {
            _parent.ChangeToHomePage();
        }
        //when the button is pressed, the method in MainWindowViewModel is called to change the page.
        private void CreateButtonClicked()
        {
            _parent.ChangeToQuestionBankCreatePage();

        }
        //when the button is pressed, the method in MainWindowViewModel is called to change the page.
        private void DeleteButtonClicked()
        {
            _parent.ChangeToQuestionBankDeletePage();

        }
        //when the button is pressed, the method in MainWindowViewModel is called to change the page.
        private void ReadButtonClicked()
        {
            _parent.ChangeToQuestionBankReadPage();
        }
        //when the button is pressed, the method in MainWindowViewModel is called to change the page.
        private void EditButtonClicked()
        {
            _parent.ChangeToQuestionBankEditPage();
        }

    }
}
```

CREATE QUESTION BANK CODE:

XAML:

```
<UserControl x:Class="NEA_Project.Pages.QuestionBankCreatePage"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:local="clr-namespace:NEA_Project.Pages"
             mc:Ignorable="d"
             d:DesignHeight="450" d:DesignWidth="800">
    <Grid>
    <!--Databinding with viewmodel: 3 textboxes that allow the user to input info, Buttons to add to database and a back button-->
    <StackPanel>
        <TextBlock Text="HI THIS IS CREATE QUESTION BANK PAGE :D"/>
        <StackPanel Orientation="Horizontal" Margin="0,20,0,20">
            <TextBlock Text="QUESTION BANK NAME"/>
            <TextBox VerticalAlignment ="Center" Width="200" Text="{Binding BankName, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"/>
```

```
            <Button Content = "CreateFile" Height = "30" Width = "120"
              Command="{Binding CreateQuestionBankCommand}"/>
          </StackPanel>
          <StackPanel Orientation="Horizontal" Margin="0,20,0,20">
           <TextBlock Text="QUESTION"/>
           <TextBox VerticalAlignment ="Center" Width="200" Text="{Binding Question, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"/>
          </StackPanel>
          <StackPanel Orientation="Horizontal" Margin="0,20,0,20">
           <TextBlock Text="ANSWER"/>
           <TextBox VerticalAlignment ="Center" Width="200" Text="{Binding Answer, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"/>
             <Button Content = "ADD QUESTION" Height = "30" Width = "120"
              Command="{Binding AddToQuestionBankCommand}"/>
          </StackPanel>
          <Button Content = "BACK TO MENU" Height = "30" Width = "120"
             Command="{Binding MenuButtonClickedCommand}"/>
       </StackPanel>
           </Grid>
</UserControl>
```

CODE BEHIND:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace NEA_Project.Pages
{
          /// <summary>
          /// Interaction logic for QuestionBankCreatePage.xaml
          /// </summary>
          public partial class QuestionBankCreatePage : UserControl
          {
                public QuestionBankCreatePage()
                {
                   InitializeComponent();
                }
          }
}
```

VIEWMODEL:

```
using NEA_Project.Helpers;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Input;

namespace NEA_Project.ViewModels
{
        public class QuestionBankCreateViewModel
        {
            //initialise
            MainWindowViewModel _parent;
            private string _BankName = "Default";
            private string _question = String.Empty;
            private string _answer = String.Empty;

            public ICommand MenuButtonClickedCommand { get; }
            public ICommand AddToQuestionBankCommand { get; }

            //constructor
            public QuestionBankCreateViewModel(MainWindowViewModel Parent)
            {
               _parent = Parent;
               MenuButtonClickedCommand = new SimpleCommand(_ => MenuButtonClicked());
               AddToQuestionBankCommand = new SimpleCommand(_ =>
        AddQuestionsAndAnswers());
            }

            public string BankName { get => _BankName; set { _BankName = value; } }
            public string Question { get => _question; set { _question = value; } }
            public string Answer { get => _answer; set { _answer = value; } }

            //adds questions and answers to the question bank database, as long as all conditions are
        met (answer and question can't be null
            //and bank name can't be null or "Default"). Then a unique QuestionID is calculated before the
        question and answer are added.
            public void AddQuestionsAndAnswers()
            {
               int QuestionID = 0;
               if (BankName == String.Empty || BankName == "Default")
               {
                 MessageBox.Show("Please Enter a suitable Question Bank Name! (Default is already in
        use)");

               }
               else
               {
                 if (Answer == String.Empty || Question == String.Empty)
                 {
                      MessageBox.Show("Oh no! Please enter both your Question and answer!");
                 }
```

```
                else
                {
                        try
                        {
                        QuestionID = NumberOfQuestions();
                        //NumberOfQuestions can sometimes return null.
                        }
                        catch (Exception)
                        {


                        }

                         try
                        {
                        QuestionID += 1;
                        //SQL INSERT INTO DATABASE
                       _parent.Database.InsertData($"QuestionBanks", "UserID, BankName,
        QuestionID,Question, Answer", $"{_parent.UserID},'{BankName}',{QuestionID},'{Question}',
        '{Answer}'");
                        }
                        catch (Exception)
                        {
                        }

                 }
                }
            }

                    //calculates the number of questions in a given question bank (to help calculate the
        Question ID)
            public int NumberOfQuestions()
            {
                return _parent.Database.GetSize("QuestionBanks", "QuestionID", $"WHERE UserID =
        {_parent.UserID} AND BankName = '{BankName}'");
            }
            //when the button is pressed, the method in MainWindowViewModel is called to change the
        page.
            private void MenuButtonClicked()
            {
                _parent.ChangeToQuestionBankMenuPage();
            }
        }
}
```

DELETE QUESTION BANK CODE:

XAML:

```
<UserControl x:Class="NEA_Project.Pages.QuestionBankDeletePage"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:local="clr-namespace:NEA_Project.Pages"
        mc:Ignorable="d"
```

```
        d:DesignHeight="450" d:DesignWidth="800">
        <Grid>
    <!--Databinding with view model: A ComboBox with a list of all question banks and buttons, 1) deletes,
2) refreshes & repopulates
    list in combobox 3) back to menu-->
    <StackPanel>
      <TextBlock Text="HI THIS IS DELETE QUESTION BANK PAGE :D"/>
      <StackPanel Orientation="Horizontal" Margin="0,20,0,20">
       <TextBlock Text="QUESTION:"/>
       <ComboBox
        Width ="200"
        ItemsSource="{Binding Path=QuestionBanks}"
        SelectedValue="{Binding Path=selectedValue,Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
                 />
        <TextBlock VerticalAlignment ="Center" Width="200" Text="{Binding UserNameInput,
UpdateSourceTrigger=PropertyChanged}"/>
        <TextBlock VerticalAlignment ="Center" HorizontalAlignment="Right" Width="200" Text="{Binding
Score, UpdateSourceTrigger=PropertyChanged}"/>
      </StackPanel>
      <Button Content = "Delete" Height = "30" Width = "120"
        Command="{Binding CheckingButton}"/>
      <Button Content = "Refresh" Height = "30" Width = "80"
        Command="{Binding RefreshButtonClickedCommand}"/>
      <Button Content = "BACK TO MENU" Height = "30" Width = "120"
        Command="{Binding MenuButtonClickedCommand}"/>
    </StackPanel>
        </Grid>
</UserControl>
```

CODE BEHIND:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace NEA_Project.Pages
{
        /// <summary>
        /// Interaction logic for QuestionBankDeletePage.xaml
        /// </summary>
        public partial class QuestionBankDeletePage : UserControl
        {
            public QuestionBankDeletePage()
            {
```

```
                InitializeComponent();
            }
        }
}
```

VIEWMODEL:

```
using NEA_Project.Helpers;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Input;

namespace NEA_Project.ViewModels
{
        public class QuestionBankDeleteViewModel : ObservableObject
        {
            //initialise
            MainWindowViewModel _parent;
            private ObservableCollection<string> _questionBanks = new ObservableCollection<string>();
            private string _selectedValue = "";
            public ICommand CheckingButton { get; }
            public ICommand MenuButtonClickedCommand { get; }
            public ICommand RefreshButtonClickedCommand { get; }
            //constructor
            public QuestionBankDeleteViewModel(MainWindowViewModel Parent)
            {
                _parent = Parent;
                PopulateList();
                MenuButtonClickedCommand = new SimpleCommand(_ => MenuButtonClicked());
                CheckingButton = new SimpleCommand(_ => DeleteButton());
                RefreshButtonClickedCommand = new SimpleCommand(_ => RefreshButtonClicked());
            }
            public ObservableCollection<string> QuestionBanks { get => _questionBanks; set {
        RaiseAndSetIfChanged(ref _questionBanks, value); } }

            public string selectedValue { get => _selectedValue; set { _selectedValue = value; } }

            //method used to delete a question bank dependent on user's input and confirms the decision
            private void DeleteButton()
            {

                if (MessageBox.Show($"Delete {_selectedValue} question bank?", "Question",
        MessageBoxButton.YesNo, MessageBoxImage.Warning) == MessageBoxResult.Yes)
                {
                  //confirms the users choice
                    _parent.Database.DeleteData("QuestionBanks", $"BankName = '{selectedValue}'");
                    _questionBanks.Remove(_selectedValue);
                  MessageBox.Show("Successfully Deleted!");
                }
```

```
            }
            //when pressed, this button calls the method populateList
            public void RefreshButtonClicked()
            {
                PopulateList();
            }

            //In order to populate the ComboBox, there must be a list of all the possible countries saved,
        this reads the database
            //and saves all the names to the list.
            private void PopulateList()
            {

                List<string> hi = _parent.Database.ReadData("QuestionBanks", "BankName", $"USERID =
        {_parent.UserID}",1);
                foreach (string s in hi)
                {
                 if (!(_questionBanks.Contains(s)))
                 { _questionBanks.Add(s); }
                }

            }
            //when the button is pressed, the method in MainWindowViewModel is called to change the
        page.
            private void MenuButtonClicked()
            {
                _parent.ChangeToQuestionBankMenuPage();
            }
        }
}
```

EDIT QUESTION BANK CODE:

XAML:

```
<UserControl x:Class="NEA_Project.Pages.QuestionBankEditPage"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:local="clr-namespace:NEA_Project.Pages"
        mc:Ignorable="d"
        d:DesignHeight="450" d:DesignWidth="800">
         <Grid>
    <!--Data binding with View Model: Comboboxes with all the question banks, all questions of selected
question bank and all answers
    of selected question and 2 textboxes that allow the users to input and 4 buttons, 1) update question, 2)
update answer, 3) back
    to home page 4) refresh button: repopulate list in comboboxes-->
    <StackPanel>
      <TextBlock Text="HI THIS IS EDIT QUESTION BANK PAGE :D"/>
      <StackPanel Orientation="Horizontal" Margin="0,20,0,20">
       <TextBlock Text="QUESTION BANK:"/>
       <ComboBox
        Width ="200"
        ItemsSource="{Binding Path=QuestionBankNames}"
```

```
            SelectedValue="{Binding Path=SelectedQuestionBankName,Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
                    />

          <Button Content = "Generate Question" Height = "30" Width = "120"
            Command="{Binding SelectQuestionBankCommand}"/>
        </StackPanel>

        <StackPanel Orientation="Horizontal">
          <StackPanel Orientation="Horizontal" Margin="0,20,0,20">
                    <TextBlock Text="QUESTION"/>
                    <ComboBox
            Width ="200"
            ItemsSource="{Binding Path=Questions}"
            SelectedValue="{Binding Path= SelectedQuestion ,Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
                    />
                    <Button Content = "Generate Question" Height = "30" Width = "120"
            Command="{Binding SelectQuestionCommand}"/>
          </StackPanel>
          <StackPanel Orientation="Horizontal" Margin="10,20,0,20">
                    <TextBlock Text="NEW QUESTION"/>
                    <TextBox VerticalAlignment ="Center" Width="200" Text="{Binding UpdatedQuestion,
Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"/>
                    <Button Content = "UPDATE QUESTION" Height = "30" Width = "120"
            Command="{Binding UpdateQuestionCommand}"/>
          </StackPanel>
        </StackPanel>

        <StackPanel Orientation="Horizontal">
          <StackPanel Orientation="Horizontal" Margin="8,20,0,20">
                    <TextBlock Text="ANSWER"/>
                    <ComboBox
            Width ="200"
            ItemsSource="{Binding Path=Answers}"
            SelectedValue="{Binding Path=SelectedAnswer,Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
                    />
          </StackPanel>
          <StackPanel Orientation="Horizontal" Margin="40,20,0,20">
                    <TextBlock Text="NEW ANSWER"/>
                    <TextBox VerticalAlignment ="Center" Width="200" Text="{Binding UpdatedAnswer,
Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"/>
                    <Button Content = "UPDATE ANSWER" Height = "30" Width = "120"
            Command="{Binding UpdateAnswerCommand}"/>
           </StackPanel>
        </StackPanel>
        <Button Content = "BACK TO MENU" Height = "30" Width = "120"
            Command="{Binding MenuButtonClickedCommand}"/>
        <Button Content = "REFRESH" Height = "30" Width = "120"
            Command="{Binding RefreshButtonClickedCommand}"/>
      </StackPanel>
        </Grid>
</UserControl>
```

CODE BEHIND:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
namespace NEA_Project.Pages
{
        /// <summary>
        /// Interaction logic for QuestionBankEditPage.xaml
        /// </summary>
        public partial class QuestionBankEditPage : UserControl
        {
            public QuestionBankEditPage()
            {
                InitializeComponent();
            }
        }
}
```

VIEWMODEL:

```
using NEA_Project.Helpers;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Input;

namespace NEA_Project.ViewModels
{
        public class QuestionBankEditViewModel : ObservableObject
        {
            //Initialise
            MainWindowViewModel _parent;
            private ObservableCollection<string> _questionBank = new ObservableCollection<string>();
            private ObservableCollection<string> _questions = new ObservableCollection<string>();
            private ObservableCollection<string> _answers = new ObservableCollection<string>();
            private string _selectedQuestionBankName = "";
            private string _selectedQuestion = "";
            private string _selectedAnswer = "";
            private string _updatedQuestion = "";
            private string _updatedAnswer = "";
            public ICommand SelectQuestionBankCommand { get; }
            public ICommand UpdateQuestionCommand { get; }
            public ICommand UpdateAnswerCommand { get; }
```

```
        public ICommand SelectQuestionCommand { get; }
        public ICommand MenuButtonClickedCommand { get; }
        public ICommand RefreshButtonClickedCommand { get; }
        //Constructor
        public QuestionBankEditViewModel(MainWindowViewModel Parent)
        {
            _parent = Parent;
            populateList();
            MenuButtonClickedCommand = new SimpleCommand(_ => MenuButtonClicked());
            SelectQuestionBankCommand = new SimpleCommand(_ => SelectQuestionBank());
            UpdateQuestionCommand = new SimpleCommand(_ => UpdateQuestion());
            UpdateAnswerCommand = new SimpleCommand(_ => UpdateAnswer());
            SelectQuestionCommand = new SimpleCommand(_ => SelectQuestion());
            RefreshButtonClickedCommand = new SimpleCommand(_ => RefreshButtonClicked());
        }

        public ObservableCollection<string> QuestionBankNames { get => _questionBank; }
        public ObservableCollection<string> Questions { get => _questions; set {
RaiseAndSetIfChanged(ref _questions, value); } }
        public ObservableCollection<string> Answers { get => _answers; set {
RaiseAndSetIfChanged(ref _answers, value); } }

        public string SelectedQuestionBankName { get => _selectedQuestionBankName; set {
_selectedQuestionBankName = value; } }
        public string SelectedQuestion { get => _selectedQuestion; set { RaiseAndSetIfChanged(ref
_selectedQuestion, value); } }
        public string SelectedAnswer { get => _selectedAnswer; set { RaiseAndSetIfChanged(ref
_selectedAnswer, value); } }
        public string UpdatedQuestion { get => _updatedQuestion; set { _updatedQuestion = value; }
}
        public string UpdatedAnswer { get => _updatedAnswer; set { _updatedAnswer = value; } }

        //In order to populate the ComboBox, there must be a list of all the possible countries saved,
this reads the database
        //and saves all the names to the list.
        private void populateList()
        {
            List<string> hi = _parent.Database.ReadData("QuestionBanks", "BankName", $"USERID =
{_parent.UserID}", 1);
            foreach (string s in hi)
            {
             if (!(_questionBank.Contains(s)))
             { _questionBank.Add(s); }
            }


             }
        //when button is pressed it calls the method populateList
        private void RefreshButtonClicked()
        {
            populateList();
        }

        //when the button is pressed, the method in MainWindowViewModel is called to change the
page.
        private void MenuButtonClicked()
```

```
        {
            _parent.ChangeToQuestionBankMenuPage();
        }

        //this method selects a question bank based on the user input, then reads the questions from
the
        //selected bank and adds them to a list of questions to populate the question combobox.
        public void SelectQuestionBank()
        {
            _questions.Clear();
            _answers.Clear();
            _selectedAnswer = String.Empty;
            _selectedQuestion = String.Empty;
            MessageBox.Show($"{SelectedQuestionBankName}");
            List<string> question = _parent.Database.ReadData("QuestionBanks", "Question",
$"UserID = {_parent.UserID} AND BankName = '{SelectedQuestionBankName}'",1);
            foreach (string s in question)
            {
             if (!(_questions.Contains(s)))
             { _questions.Add(s); }
            }
        }
        //reads the database and adds them to a list of answers to populate the answer combobox
        public void SelectQuestion()
        {
            List<string> answer = _parent.Database.ReadData("QuestionBanks", "Answer", $"UserID =
{_parent.UserID} AND BankName = '{SelectedQuestionBankName}' AND Question =
'{SelectedQuestion}' ", 1);
            foreach (string s in answer)
            {
             if (!(_answers.Contains(s)))
             { _answers.Add(s); }
            }
        }

        //Updates a question in the question bank database. Checks whether all
        //required fields are filled in, retrieves the ID of the selected question
        //, and calls then updates the question in the database.
        public void UpdateQuestion()
        {
            if (SelectedQuestionBankName == String.Empty)
            {
             MessageBox.Show("No bank selected!");

            }
            else if (SelectedQuestion == String.Empty)
            {
             MessageBox.Show("Oh no! Please select your Question!");
            }
            else if (UpdatedQuestion == String.Empty)
            {
             MessageBox.Show("Oh no! Please enter your new Question!");
            }
            else
            {
```

```
                string ID = _parent.Database.ReadData("QuestionBanks", "QuestionID", $"Question =
        '{SelectedQuestion}' AND UserID = {_parent.UserID} AND BankName =
        '{SelectedQuestionBankName}'",1)[0];
                int QuestionID = Int32.Parse(ID);
                _parent.Database.UpdateData("QuestionBanks", $"Question = '{UpdatedQuestion}'",
        $"UserID = {_parent.UserID} AND BankName = '{SelectedQuestionBankName}' AND QuestionID
        = {QuestionID}");
                    }
                UpdatedQuestion = String.Empty;
            }
            //Updates an answer in the question bank database. Checks whether all
            //required fields are filled in, retrieves the ID of the selected question
            //, and calls then updates the answer in the database.
            public void UpdateAnswer()
            {
                if (SelectedQuestionBankName == String.Empty)
                {
                 MessageBox.Show("No bank selected!");
                }
                else if (SelectedAnswer == String.Empty)
                {
                 MessageBox.Show("Oh no! Please select your Answer!");
                }
                else if (UpdatedAnswer == String.Empty)
                {
                 MessageBox.Show("Oh no! Please enter your new Answer!");
                }
                else
                {
                 string ID = _parent.Database.ReadData("QuestionBanks", "QuestionID", $"Question =
        '{SelectedQuestion}' AND UserID = {_parent.UserID} AND BankName =
        '{SelectedQuestionBankName}'", 1)[0];
                int QuestionID = Int32.Parse(ID);
                _parent.Database.UpdateData("QuestionBanks", $"Answer = '{UpdatedAnswer}'",
        $"UserID = {_parent.UserID} AND BankName = '{SelectedQuestionBankName}' AND QuestionID
        = {QuestionID}");
                }
                UpdatedAnswer = String.Empty;
            }
        }
}
```

READ QUESTION BANK CODE:

XAML:

```
<UserControl x:Class="NEA_Project.Pages.QuestionBankReadPage"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:local="clr-namespace:NEA_Project.Pages"
        mc:Ignorable="d"
        d:DesignHeight="450" d:DesignWidth="800">
         <Grid>
    <!--Data binding with view model: ComboBox with all possible question banks and then another
    combobox with all possible questions as well as 5 buttons-->
    <StackPanel>
```

```
        <TextBlock Text="HI THIS IS EDIT QUESTION BANK PAGE :D"/>
        <StackPanel Orientation="Horizontal" Margin="0,20,0,20">
         <TextBlock Text="QUESTION BANK:"/>
         <ComboBox
          Width ="200"
          ItemsSource="{Binding Path=QuestionBankNames}"
          SelectedValue="{Binding Path=SelectedQuestionBankName,Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
                  />

         <Button Content = "Generate Question" Height = "30" Width = "120"
           Command="{Binding SelectQuestionBankCommand}"/>
        </StackPanel>

        <StackPanel Orientation="Horizontal">
         <StackPanel Orientation="Horizontal" Margin="0,20,0,20">
                  <TextBlock Text="QUESTION"/>
                  <ComboBox
          Width ="200"
          ItemsSource="{Binding Path=Questions}"
          SelectedValue="{Binding Path= SelectedQuestion ,Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
                   />
                  <Button Content = "Generate Question" Height = "30" Width = "120"
           Command="{Binding SelectQuestionCommand}"/>
         </StackPanel>
         <StackPanel Orientation="Horizontal" Margin="10,20,0,20">
                  <TextBlock Text="ANSWER: "></TextBlock>
                  <TextBlock Text="{Binding Answer}" Width="100"></TextBlock>
                  <Button Content = "SHOW ANSWER" Height = "30" Width = "120"
          Command="{Binding ShowAnswerCommand}"/>
         </StackPanel>
        </StackPanel>

        <Button Content = "BACK TO MENU" Height = "30" Width = "120"
           Command="{Binding MenuButtonClickedCommand}"/>
        <Button Content = "REFRESH" Height = "30" Width = "120"
           Command="{Binding RefreshButtonClickedCommand}"/>
     </StackPanel>
         </Grid>
</UserControl>
```

CODE BEHIND:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
```

```
using System.Windows.Shapes;

namespace NEA_Project.Pages
{
        /// <summary>
        /// Interaction logic for QuestionBankReadPage.xaml
        /// </summary>
        public partial class QuestionBankReadPage : UserControl
        {
            public QuestionBankReadPage()
            {
                InitializeComponent();
            }
        }
}
```

VIEWMODEL:

```
using NEA_Project.Helpers;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Input;

namespace NEA_Project.ViewModels
{
        public class QuestionBankReadViewModel : ObservableObject
        {
            //Initialise
            MainWindowViewModel _parent;
            private ObservableCollection<string> _questionBank = new ObservableCollection<string>();
            private ObservableCollection<string> _questions = new ObservableCollection<string>();
            private ObservableCollection<string> _answers = new ObservableCollection<string>();
            private List<string> _test = new List<string>();

            private string _selectedQuestionBankName = "";
            private string _selectedQuestion = "";

            public ICommand SelectQuestionBankCommand { get; }
            public ICommand MenuButtonClickedCommand { get; }
            public ICommand RefreshButtonClickedCommand { get; }
            public ICommand ShowAnswerCommand { get; }

            //Constructor
            public QuestionBankReadViewModel(MainWindowViewModel Parent)
                 {
                _parent = Parent;
                MenuButtonClickedCommand = new SimpleCommand(_ => MenuButtonClicked());
                RefreshButtonClickedCommand = new SimpleCommand(_ => RefreshButtonClicked());
                SelectQuestionBankCommand = new SimpleCommand(_ => SelectQuestionBank());
                ShowAnswerCommand = new SimpleCommand(_ => ShowAnswer());
            }
```

```
        public ObservableCollection<string> QuestionBankNames { get => _questionBank; }
            public ObservableCollection<string> Questions { get => _questions; set {
RaiseAndSetIfChanged(ref _questions, value); } }
        public ObservableCollection<string> Answers { get => _answers; set {
RaiseAndSetIfChanged(ref _answers, value); } }

        public string SelectedQuestionBankName { get => _selectedQuestionBankName; set {
_selectedQuestionBankName = value; } }
        public string SelectedQuestion { get => _selectedQuestion; set { RaiseAndSetIfChanged(ref
_selectedQuestion, value); } }

        //when the button is pressed, the method in MainWindowViewModel is called to change the
page.
        private void MenuButtonClicked()
        {
            _parent.ChangeToQuestionBankMenuPage();
        }
             //when button is pressed, method populateList is called
        private void RefreshButtonClicked()
        {
            populateList();
        }

        //In order to populate the ComboBox, there must be a list of all the possible countries saved,
this reads the database
        //and saves all the names to the list.
        private void populateList()
        {
            List<string> hi = _parent.Database.ReadData("QuestionBanks", "BankName", $"USERID =
{_parent.UserID} OR UserID = 0", 1);
            foreach (string s in hi)
            {
              if (!(_questionBank.Contains(s)))
              { _questionBank.Add(s); }
            }


        }

        //this method selects a question bank based on the user input, then reads the questions from
the
        //selected bank and adds them to a list of questions to populate the question combobox.
        public void SelectQuestionBank()
        {
            _questions.Clear();
            _answers.Clear();
            _selectedQuestion = String.Empty;
            MessageBox.Show($"{SelectedQuestionBankName}");
            int ID;
            if (SelectedQuestionBankName == "Default")
            {
              ID = 0;
            }
            else
            {
```

```
                ID = _parent.UserID;
            }
            List<string> question = _parent.Database.ReadData("QuestionBanks", "Question",
        $"BankName = '{SelectedQuestionBankName}' AND USERID = {ID}", 1);
            foreach (string s in question)
            {
             if (!(_questions.Contains(s)))
             { _questions.Add(s); }
            }

        }
        //reads the database and displays a message box of the answer
        public void ShowAnswer()
        {
            int ID;
            if (SelectedQuestionBankName == "Default")
            {
             ID = 0;
            }
            else
            {
             ID = _parent.UserID;
            }
            List<string> GetAnswer = _parent.Database.ReadData("QuestionBanks", "Answer",
        $"Question = '{SelectedQuestion}' AND BankName = '{SelectedQuestionBankName}' AND
        USERID = {ID}", 1);
            //This prevents an out of range error, as once again there is a possibility that the ReadData
        method returns nothing.
            if (GetAnswer.Count > 0)
            {
             string answer = GetAnswer[0];
             MessageBox.Show(answer);
            }
            else
            {
             MessageBox.Show("Please Selected a Question Bank AND a Question :)");
            }

        }
    }
}
```

USER STATS PAGE CODE:

XAML:

```
<UserControl x:Class="NEA_Project.Pages.UserStatsPage"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:local="clr-namespace:NEA_Project.Pages"
        mc:Ignorable="d"
        d:DesignHeight="450" d:DesignWidth="800">
        <Grid Background="White">
    <!--Data binding with viewmodel: Textblocks that display the users different stats, a button that triggers
the display and
```

```
        a ComboBox that shows the 3 different categories of highscores, 3 average scores - both with a
button that updates the information
        and button that triggers percentile calculation and displays
        result and home button-->
        <StackPanel>
            <TextBlock Text="USER STATS PAGE" />
            <StackPanel Orientation="Horizontal">
             <TextBlock Text="QUIZ HIGH SCORE" Margin="10,0"></TextBlock>
             <TextBlock Text="{Binding QuizHighScore}" Width="40" Margin="10,0"></TextBlock>
             <TextBlock Text="PAIRS HIGH SCORE" Margin="10,0"></TextBlock>
             <TextBlock Text="{Binding PairsHighScore}" Width="40" Margin="10,0"></TextBlock>
             <TextBlock Text="WORD SCRAMBLE HIGH SCORE" Margin="10,0"></TextBlock>
             <TextBlock Text="{Binding WordHighScore}" Width="100" Margin="10,0"></TextBlock>
               <Button Content="Get Stats" Command="{Binding GetUserStatsCommand}" Width="50"
Height="30"/>

            </StackPanel>

            <StackPanel Orientation="Horizontal">
             <TextBlock Text="AVERAGE QUIZ SCORE" Margin="5,0"></TextBlock>
             <TextBlock Text="{Binding QuizScoreAverage}" Width="40" Margin="5,0"></TextBlock>
             <TextBlock Text="AVERAGE PAIRS SCORE" Margin="5,0"></TextBlock>
             <TextBlock Text="{Binding PairsScoreAverage}" Width="40" Margin="5,0"></TextBlock>
             <TextBlock Text="AVERAGE WORD SCRAMBLE SCORE" Margin="5,0"></TextBlock>
             <TextBlock Text="{Binding WordScrambleScoreAverage}" Width="100" Margin="5,0"></TextBlock>
             <Button Content="Get Average" Command="{Binding GetAverageCommand}" Width="70"
Height="30"/>

            </StackPanel>

            <StackPanel Orientation="Vertical">
             <ComboBox
                        Width ="200" Height="20"
                        ItemsSource="{Binding Path=HighScoresCategories}"
                        SelectedValue="{Binding Path=selectedValue,Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
                        />
                    <Button Content="Get Ranking" Command="{Binding GetPercentagesCommand}"
Width="200" Height="30"/>

            </StackPanel>
            <Button Content = "HOME PAGE" Height = "30" Width = "130"
              Command="{Binding HomeButtonClickedCommand}"/>

        </StackPanel>
            </Grid>
</UserControl>
```

CODE BEHIND:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace NEA_Project.Pages
{
    /// <summary>
    /// Interaction logic for UserStatsPage.xaml
    /// </summary>
    public partial class UserStatsPage : UserControl
    {
        public UserStatsPage()
        {
            InitializeComponent();
        }
    }
}
```

VIEWMODEL:

```
using NEA_Project.Helpers;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Input;

namespace NEA_Project.ViewModels
{
    public class UserStatsViewModel : ObservableObject
    {
        //Initialise
        MainWindowViewModel _parent;
        private double _userPercentile;
        private int _quizScoreAverage = 0;
        private int _pairsScoreAverage = 0;
        private int _wordScrambleAverage = 0;
        public MergeSort MergeSort;
        private string _quizHighScore;
        private string _pairsHighScore;
        private string _wordScrambleHighScore;
        private List<string> _highScoreCategories = new List<string>()
        { "Quiz Scores", "Pairs Scores", "Word Scramble Scores"};
        private string _selectedHighScore;
        public ICommand GetUserStatsCommand { get; }
        public ICommand GetPercentagesCommand { get; }
```

```
        public ICommand HomeButtonClickedCommand { get; }
        public ICommand GetAverageCommand { get;  }
        //Constructor
        public UserStatsViewModel(MainWindowViewModel Parent)
        {
            _parent = Parent;
            MergeSort = new MergeSort();
            HomeButtonClickedCommand = new SimpleCommand(_ => HomeButtonClicked());
            GetUserStatsCommand = new SimpleCommand(_=> GetUserStats());
            GetPercentagesCommand = new SimpleCommand(_ => GetPercentile());
            GetAverageCommand = new SimpleCommand(_ => GetAverageScore());
        }
        //when the button is pressed, the method in MainWindowViewModel is called to change the
page.
        private void HomeButtonClicked()
        {
            _parent.ChangeToHomePage();
        }
        public string QuizHighScore { get => _quizHighScore; set { RaiseAndSetIfChanged(ref
_quizHighScore, value); } }
        public string PairsHighScore { get => _pairsHighScore; set { RaiseAndSetIfChanged(ref
_pairsHighScore, value); } }
        public string WordHighScore { get => _wordScrambleHighScore; set {
RaiseAndSetIfChanged(ref _wordScrambleHighScore, value); } }
        public int QuizScoreAverage { get => _quizScoreAverage; set { RaiseAndSetIfChanged(ref
_quizScoreAverage, value); } }
        public int PairsScoreAverage { get => _pairsScoreAverage; set { RaiseAndSetIfChanged(ref
_pairsScoreAverage, value); } }
        public int WordScrambleScoreAverage { get => _wordScrambleAverage; set {
RaiseAndSetIfChanged(ref _wordScrambleAverage, value); } }

        public List<string> HighScoresCategories { get => _highScoreCategories; }

        public string selectedValue
        {
                get => _selectedHighScore;
            set { _selectedHighScore = value; }
        }

        public void GetUserStats()
        {
            //Reads database and assigns values, using the SELECT MAX() SQL Function
            List<string> QuizScores = _parent.Database.ReadData("QuizScores", "MAX(Score)",
"USERID = 1", 1);
            List<string> PairsScores = _parent.Database.ReadData("PairsScores", "MAX(Score)",
"USERID = 1", 1);
            List<string> WordScrambleScores = _parent.Database.ReadData("WordScrambleScores",
"MAX(Score)", "USERID = 1", 1);
            if (QuizScores.Count != 0)
            {
             QuizHighScore = QuizScores[0];
            }
            else
            {
             QuizHighScore = "0";
            }
```

```
            if (PairsScores.Count != 0)
            {
             PairsHighScore = PairsScores[0];
            }
            else
            {
             PairsHighScore = "0";
            }
                  if (WordScrambleScores.Count != 0)
            {
             WordHighScore = WordScrambleScores[0];
            }
            else
            {
             WordHighScore = "0";
            }



        }



        //populates a list of high scores for each user in a database table

        private void GetAverageScore()
        {
            //Get how many scores the user has.
            int NumberOfQuizScores = _parent.Database.GetSize("QuizScores", "QuizID", $"WHERE
UserID = {_parent.UserID}");
            int NumberOfPairsScores = _parent.Database.GetSize("PairsScores", "PairsID",
$"WHERE UserID = {_parent.UserID}");
            int NumberOfWordScrambleScores = _parent.Database.GetSize("WordScrambleScores",
"WordScrambleID", $"WHERE UserID = {_parent.UserID}");

            //Retrieves the User's scores
            List<string> QuizScores = _parent.Database.ReadData("QuizScores", "Score", $"USERID
= {_parent.UserID}", 1);
            List<string> PairsScores = _parent.Database.ReadData("PairsScores", "Score", $"USERID
= {_parent.UserID}", 1);
            List<string> WordScrambleScores = _parent.Database.ReadData("WordScrambleScores",
"Score", $"USERID = {_parent.UserID}", 1);

            int TotalQuizScore = 0;
            int TotalPairsScore = 0;
            int TotalWordScrambleScore = 0;

            //Foreach adds every score for each category.
            foreach (var item in QuizScores)
            {
             TotalQuizScore += Int32.Parse(item);
            }

            foreach (var item in PairsScores)
            {
             TotalPairsScore += Int32.Parse(item);
```

```
            }

            foreach (var item in WordScrambleScores)
            {
              TotalWordScrambleScore += Int32.Parse(item);
            }

            //Try Catches a 0 division -> important if the user has never played a game before.
            try
            {
              QuizScoreAverage = TotalQuizScore / NumberOfQuizScores;
            }
            catch (Exception)
            {

            }

            try
            {
              PairsScoreAverage = TotalPairsScore / NumberOfPairsScores;
            }
            catch (Exception)
            {

            }

            try
            {
                WordScrambleScoreAverage = TotalWordScrambleScore /
NumberOfWordScrambleScores;
            }
            catch (Exception)
            {

            }



        }
            void GetPercentile()
        {
            int NumberOfScores = 0;
            List<int> HighScores = new List<int>();
            int UserHS = 0;
            //Switch statement to determine the selected category:
            switch (_selectedHighScore)
            {
              case "Quiz Scores":
                        //Getting the number of users NOT number of scores in the database:
                        NumberOfScores = _parent.Database.GetSize("QuizScores", "DISTINCT
UserID", "");
                        for (int i = 1; i < NumberOfScores + 1; i++)
                        {
                        //Iterates through each user and get their high score using the SELECT MAX()
SQL Function and this is added to HighScores List.
```

```
                    List<string> HighScore = _parent.Database.ReadData("QuizScores",
"MAX(Score)", $"USERID = {i}", 1);
                HighScores.Add(Int32.Parse(HighScore[0]));
                    if (i == _parent.UserID)
                    {
                    //If the current user is the same as the logged-in user,
                    //their high score is stored in the "UserHS" variable.
                    UserHS = Int32.Parse(HighScore[0]);
                    }

                    }
                    break;
            case "Pairs Scores":
                    NumberOfScores = _parent.Database.GetSize("PairsScores", "DISTINCT
UserID", "");
                    for (int i = 1; i < NumberOfScores + 1; i++)
                    {

                     List<string> HighScore = _parent.Database.ReadData("PairsScores",
"MAX(Score)", $"USERID = {i}", 1);
                HighScores.Add(Int32.Parse(HighScore[0]));
                    if (i == _parent.UserID)
                    {
                     UserHS = Int32.Parse(HighScore[0]);
                    }

                    }
                    break;
            case "Word Scramble Scores":
                    NumberOfScores = _parent.Database.GetSize("WordScrambleScores",
"DISTINCT UserID", "");
                    for (int i = 1; i < NumberOfScores + 1; i++)
                    {

                    List<string> HighScore = _parent.Database.ReadData("WordScrambleScores",
"MAX(Score)", $"USERID = {i}", 1);
                HighScores.Add(Int32.Parse(HighScore[0]));
                    if (i == _parent.UserID)
                    {
                    UserHS = Int32.Parse(HighScore[0]);
                    }

                    }
                    break;
             default:
                    break;
        }

        //HighScores is then passed through the Sort Method.
        List<int> SortedHighScores = MergeSort.Sort(HighScores);

        int position = -1;
        for (int i = 0; i < SortedHighScores.Count; i++)
        {
         if (SortedHighScores[i] == UserHS)
           {
```

```
                        //Searches the sorted "HighScores" list for the user's high score
                        //and stores their position
                        position = i;
                        break;
                    }
                }
                if (position == 0)
                {
                    _userPercentile = 0;
                }
                else if (position == -1)
                {
                    MessageBox.Show("Oh no! :o Make sure you've selected a category!");
                }
                else
                {
                    //Calculates the user's percentile by dividing position by the total of highscores
                    //and multiplying the result by 100.
                    double temp = (double)position / (double)SortedHighScores.Count;
                      double Percentile = temp * 100;
                    _userPercentile = Percentile;
                }
                if (position != -1)
                {
                    MessageBox.Show($"You are better than {_userPercentile}% of users :)");
                }


            }


        }
}
```

VIEW STATES:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace NEA_Project.Constants
{
    //Assigning an integer to a variable which has the same name as each user control.
    public enum ViewStates
    {
    StartPage = 0,
    LoginPage = 1,
    SignUpPage = 2,
     HomePage = 3,
    ContinentsMap = 4,
    AfricaMap = 5,
```

```
        AsiaMap = 6,
        EuropeMap = 7,
        NAmericaMap = 8,
        SAmericaMap = 9,
        OceaniaMap = 10,
        QuestionBankMenu = 11,
        QuestionBankEdit = 12,
        QuestionBankCreate = 13,
        QuestionBankDelete = 14,
        QuestionBankRead = 15,
        UserStatsPage = 16,
        GameMenuPage = 17,
        QuizPage = 18,
        WordScramblePage = 19,
        PairsGamePage = 20,
        }
}
```

SQLITE:

```
//Based on: https://www.codeguru.com/dotnet/using-sqlite-in-a-c-application/
using System;
using System.Collections.Generic;
using System.Data.SQLite;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
 namespace SQLDatabase
{

        public class Database
        {
            //Intitialise
            private SQLiteConnection Connection { get; set; }
            //Constructor
            public Database()
            {
                Connection = CreateConnection();;

            }

            private SQLiteConnection CreateConnection()
            {

                SQLiteConnection sqlite_conn;
                // Creates the connection:
                sqlite_conn = new SQLiteConnection("Data Source=database.db; " +
                 "Version = 3; New = True; Compress = True; ");
                // Open the connection:
                try
                {
                 sqlite_conn.Open();
                }
                catch (Exception ex)
                {

                }
                return sqlite_conn;
            }

            public void Execute(string query)
            {
```

```
            //All the methods that do not return anything, such as create table, are all executed here.
            SQLiteCommand sqlite_cmd;
            sqlite_cmd = Connection.CreateCommand();
            sqlite_cmd.CommandText = query;
            sqlite_cmd.ExecuteNonQuery();

        }

        //The methods that don't return anything all have the same format:
        //Create SQL Statement by concatenating
        //Call Execute Method
        public void CreateTable(string tablename, string tableRequirements)
        {
            string Createsql = $"CREATE TABLE IF NOT EXISTS {tablename}({tableRequirements});";
            Execute(Createsql);
        }

        public void InsertData(string tablename, string columns, string values)
        {
            string Insertsql = $"INSERT INTO {tablename}({columns}) VALUES({values}); ";
            Execute(Insertsql);
        }

        public void UpdateData(string tablename, string set, string condition)
        {
            string Updatesql = $"UPDATE {tablename} SET {set} WHERE {condition};";
            Execute(Updatesql);

        }
        public void DeleteData(string tablename, string condition)
        {
            string Deletesql = $"DELETE FROM {tablename} WHERE {condition};";
            Execute(Deletesql);
        }

        public void Droptable(string tablename)
        {
            string Dropsql = $"DROP TABLE {tablename};";
            Execute(Dropsql);
        }

        public List<string> ReadData(string tablename, string Column, string condition, int NoColums)
        {
            //USING THE SELECT SQL FUNCTION
            SQLiteDataReader sqlite_datareader;
            SQLiteCommand sqlite_cmd;
            //myreader - Holds any data that is returned select
            List<string> myreader = new List<string>();
            sqlite_cmd = Connection.CreateCommand();
            sqlite_cmd.CommandText = $"SELECT {Column} FROM {tablename} WHERE
{condition};";

            sqlite_datareader = sqlite_cmd.ExecuteReader();
            while (sqlite_datareader.Read())
            {
                    // The for loop makes sure that multiple fields from the same record can be returned
             for (int i = 0; i < NoColums; i++)
             {
                    //as not all of the fields are strings, the catch handles any of the integer fields
                    try
                    {
                    //adds i (field) to myreader
                  myreader.Add(sqlite_datareader.GetString(i));
                    }
                    catch (Exception)
```

```
                {
                    //however, just because it's not a string, it doesn't guarantee that it is an integer,
        and this handles any
                    //non-integers
                    try
                    {
                    int temp = (sqlite_datareader.GetInt32(i));
                     myreader.Add(temp.ToString());
                    }
                    catch (Exception)
                    {


                    }


                    }
                }

               }

               return myreader;
           }


           public int GetSize(string tablename, string ID, string Condition)
           {
               //USING THE SELECT COUNT() FUNCTION
               //This gets the size of a database and returns the value
               SQLiteDataReader sqlite_datareader;
               SQLiteCommand sqlite_cmd;
               int Count = 0;
               sqlite_cmd = Connection.CreateCommand();
               sqlite_cmd.CommandText = $"SELECT COUNT({ID}) FROM {tablename} {Condition};";
               Count = Convert.ToInt32(sqlite_cmd.ExecuteScalar());
               sqlite_datareader = sqlite_cmd.ExecuteReader();

               return Count;
           }
       }
}
```

SIMPLE COMMAND:

```
// Based on: Learn Microsoft
using System;
using System.Windows.Input;
namespace NEA_Project.Helpers
{
    // Both classes are used to define what commands can be assigned to UI Elements
    public class SimpleCommand : SimpleCommand<object>
    {
        public SimpleCommand(Action<object> action) : base(action)
        {
        }
    }

    public class SimpleCommand<T> : ICommand where T : class
    {
```

```
        private readonly Action<T> _action;

        public SimpleCommand(Action<T> action)
        {
            _action = action;
            CanExecuteChanged = (sender, args) => { };
        }

        public bool CanExecute(object parameter)
        {
            return true;
        }

        public void Execute(object parameter)
        {
            _action.Invoke((T)parameter);
        }

        public event EventHandler CanExecuteChanged;

    }
}
```

OBSERVABLE OBJECTS:

```
//Based on: Microsoft Learn
using System.Collections.Generic;
using System.ComponentModel;
using System.Runtime.CompilerServices;
namespace NEA_Project.Helpers
{
    public abstract class ObservableObject : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;
        protected virtual void OnPropertyChanged([CallerMemberName] string propertyName = null)
        {
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
        }
        /// Raises a property changed event for the property named
        /// Call this in a property setter like so:
        /// RaiseAndSetIfChanged(ref _privateField, value)

        protected virtual bool RaiseAndSetIfChanged<TProperty>(ref TProperty field,
            TProperty value,[CallerMemberName] string propertyName = null)
        {
            if (EqualityComparer<TProperty>.Default.Equals(field, value))
                return false;

            field = value;
            OnPropertyChanged(propertyName);
            return true;
        }

    }
}
```

MERGE SORT:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace NEA_Project.Helpers
{
        public class MergeSort
        {
            public MergeSort()
            {
            }
            public List<int> Sort(List<int> Unsorted)
            {
               if (Unsorted.Count > 1)
               {
                int mid = Unsorted.Count / 2;
                List<int> leftHalf = new List<int>();
                List<int> rightHalf = new List<int>();
                // Copy elements into the left and right
                for (int i = 0; i < mid; i++)
                {
                        leftHalf.Add(Unsorted[i]);
                }
                for (int i = mid; i < Unsorted.Count; i++)
                {
                        rightHalf.Add(Unsorted[i]);
                }
                // Recursively sort the left and right
                Sort(leftHalf);
                Sort(rightHalf);
                // Merge the sorted lists
                Unsorted.Clear();
                  Unsorted.AddRange(Merge(leftHalf, rightHalf)); ;
               }
               return Unsorted;
            }

            private static List<int> Merge(List<int> left, List<int> right)
            {
               List<int> result = new List<int>();
               int leftPosition = 0;
               int rightPositon = 0;
               //Keep adding to the result list until either the left or right list has finished
               while (left.Count > leftPosition && right.Count > rightPositon)
               {
                if (left[leftPosition] < right[rightPositon])
                {
                    result.Add(left[leftPosition]);
                        leftPosition += 1;
                }
                else
                {
                    result.Add(right[rightPositon]);
                        rightPositon += 1;
```

```
                }
                }
                //These for loops make sure that the remaining values are added to the result list
                for (int i = leftPosition; i < left.Count; i++)
                {
                 result.Add(left[i]);
                }
                for (int i = rightPositon; i < right.Count; i++)
                {
                 result.Add(right[i]);
                }
                return result;
            }
    }
}
```

CHECK PAIR COMMAND:

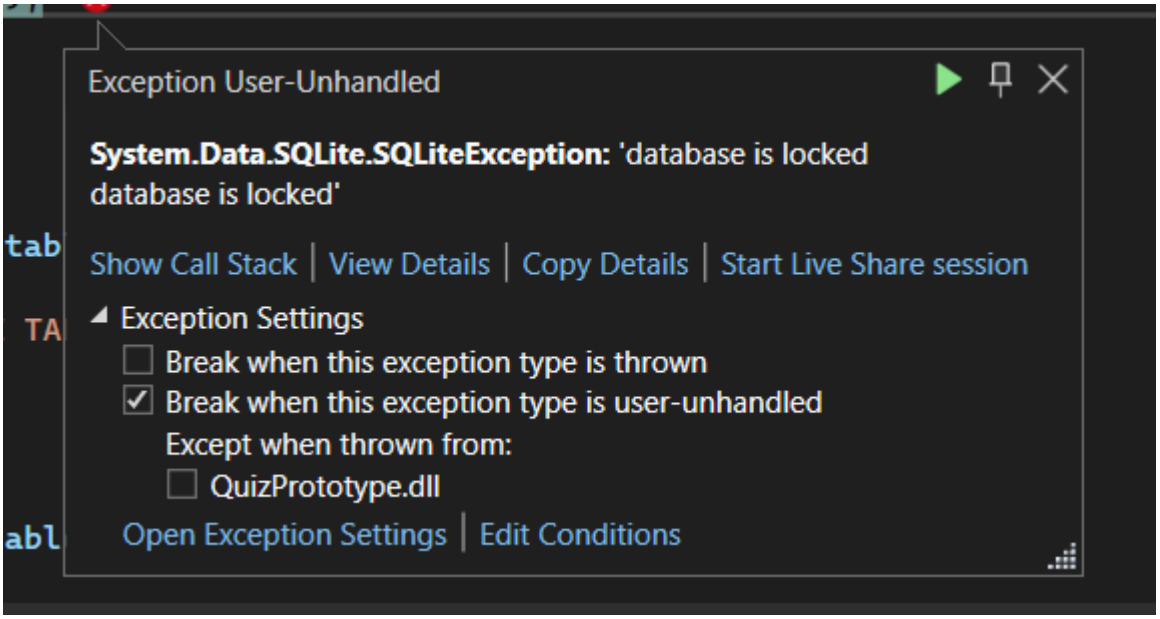
```
using MVVMEssentials.Commands;
using NEA_Project.ViewModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;

namespace NEA_Project.Commands
{
        public class CheckPairCommand : CommandBase
        {
            //Initialising attributes
            private readonly PairsGameViewModel _vm;
            private MainWindowViewModel _parent;
            private bool found = false;
            //constructor
            public CheckPairCommand(PairsGameViewModel vm, MainWindowViewModel parent)
            {

                _vm = vm;
                _parent = parent;
            }

            //This method overrides the execute method of CommandBase and is how the command
        actually executes POLYMORPHISM.
            public override void Execute(object parameter)
            {
                _vm.PairFound = false;
                RightPair();
                //check to see if a pair has been found, then the score is updated accordingly and a
        message box is displayed
                if (found)
                {
                 MessageBox.Show("Pair!");
                 _vm.Score += 1;
                 _vm.PairFound = true;
                }
                else
                {
                 MessageBox.Show($"Not Pair");
                 _vm.Score -= 1;
                }
```

```
        }

        private bool RightPair()
        {
            // This compares the content of the two textboxes with the expected answers in the
database

            int ID;
            if (_parent.CurrentQuestionBank == "Default")
            {
             ID = 0;
             //Because of the way that the Question banks are saved, this statement means
             //that it is possible for all users to access the Default Question Bank
            }
            else
            {
             ID = _parent.UserID;
            }
            found = false;
            //returns the expected answers, but because you don't know which textblock
            //is in the "Answer" column and which is in the "Question", have to
            // create 2 variables to cover both scenarios

            string textblock2Contains = _vm.TextBlock2Contains;
            //READS THE DATABASE AND STORES THE ANSWER
            List<string> CheckWithTextBlock = _parent.Database.ReadData("QuestionBanks",
"Answer", $"Question LIKE '{_vm.TextBlockContains}' AND BankName =
'{_parent.CurrentQuestionBank}' AND UserID = {ID}", 1);
            string textblockContains = _vm.TextBlockContains;
            List<string> CheckWithTextBlock2 = _parent.Database.ReadData("QuestionBanks",
"Answer", $"Question LIKE '{_vm.TextBlock2Contains}' AND BankName =
'{_parent.CurrentQuestionBank}' AND UserID = {ID}", 1);

            // Because the ReadData method sometimes returns null
            // this IF avoids an exception error as it
            // only compares the textblocks content with the answer that is not null.

            if (CheckWithTextBlock.Count == 0 && CheckWithTextBlock2.Count == 0)
            {
             //HOWEVER, there may be an instance where the content of both textblocks is stored in
the question field of the DATABASE
             //meaning that, if both the answers read from the DATABASE are null, then the question
fields are read and compared to the
             //conent in the textblocks
             List<string> CheckWithTextBlockQ = _parent.Database.ReadData("QuestionBanks",
"Question", $"Answer LIKE '{_vm.TextBlockContains}' AND BankName =
'{_parent.CurrentQuestionBank}' AND UserID = {ID}", 1);
             List<string> CheckWithTextBlock2Q = _parent.Database.ReadData("QuestionBanks",
"Question", $"Answer LIKE '{_vm.TextBlockContains}' AND BankName =
'{_parent.CurrentQuestionBank}' AND UserID = {ID}", 1);
             //READING and storing from database.
             if (CheckWithTextBlockQ.Count == 0)
             {
                    if (textblockContains == CheckWithTextBlock2Q[0])
                    {
                    found = true;
                    }
             }
             else
             {
                    if (textblock2Contains == CheckWithTextBlockQ[0]) found = true;

             }
            }
```

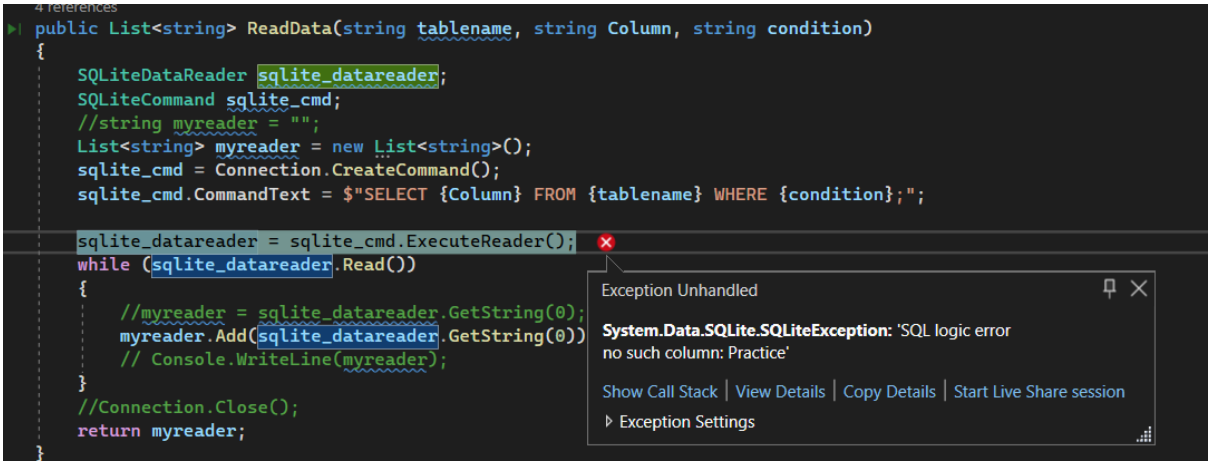
```
                else if (CheckWithTextBlock.Count == 0)
                {
                 if (textblockContains == CheckWithTextBlock2[0])
                        {
                           found = true;
                 }
                }
                else
                {
                 if (textblock2Contains == CheckWithTextBlock[0]) found = true;

                }

                return found;

            }
        }
}
```

Testing

Ongoing Testing

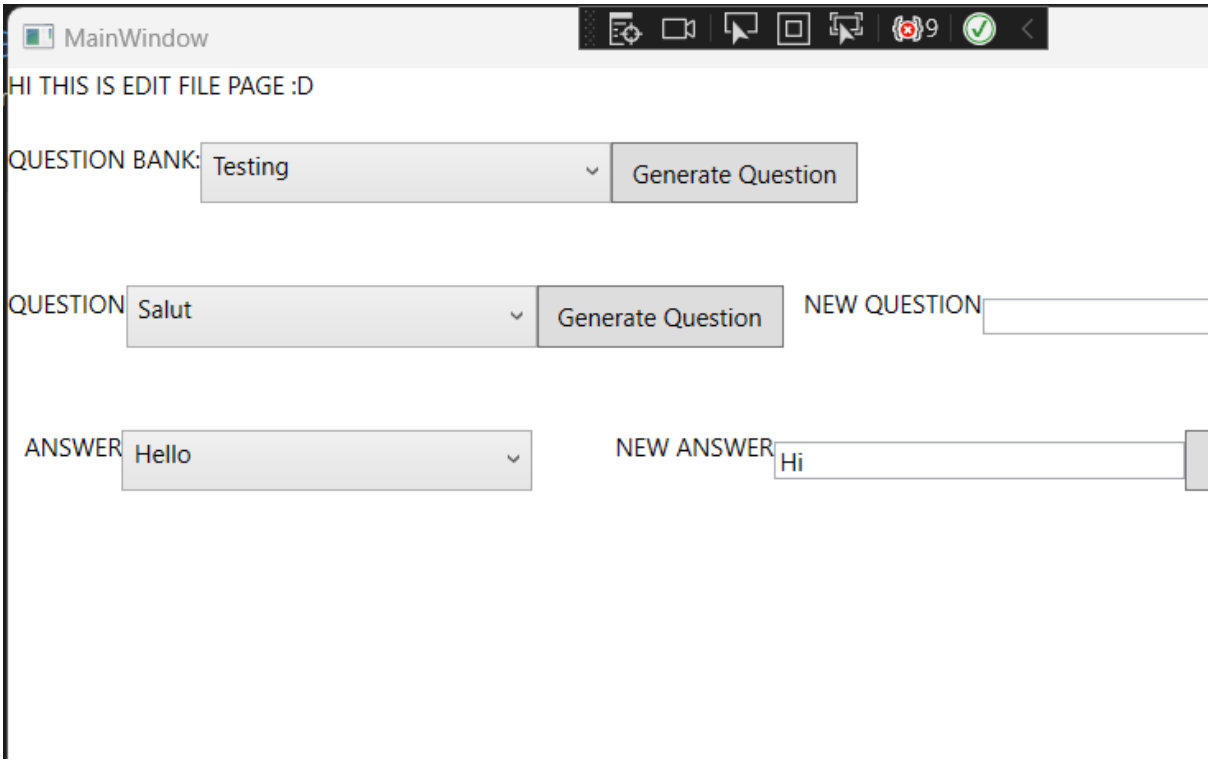


This issue came up numerous times and this was due to the nature of SQLite and the database file already being in use, even if the connections were methodically closed. This is due to the fact that a new instantiation of the database class would cause the locking error, and ultimately was not necessary to create new tables.



Another common issue was SQL Syntax, because the SQL Statements when in C# were saved as Strings, I initially ended up missing the ‘’ that were necessary for the SQL Syntax, but as the project progressed, it became more habitual.

QUESTION BANK EDIT PAGE ERRORS:



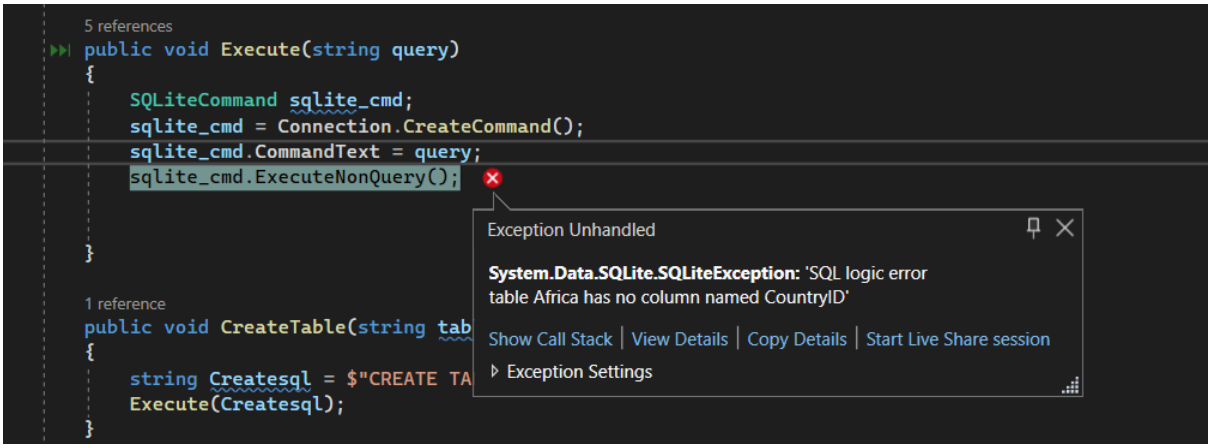
- QUESTION BANK EDIT PAGE: Whilst functional, the ComboBoxes were not updating dynamically.

```
public class EditFilePageViewModel: ObservableObject
{
    private MainWindowViewModel _parent;
    private List<string> _questionBank = new List<string>();
    private List<string> _questions = new List<string>();
    private List<string> _answer = new List<string>();
    private List<string> _test = new List<string>();
    private string _selectedQuestionBankName = "hi";
    private string _selectedQuestion = "hi";
    private string _selectedAnswer = "hi";
```

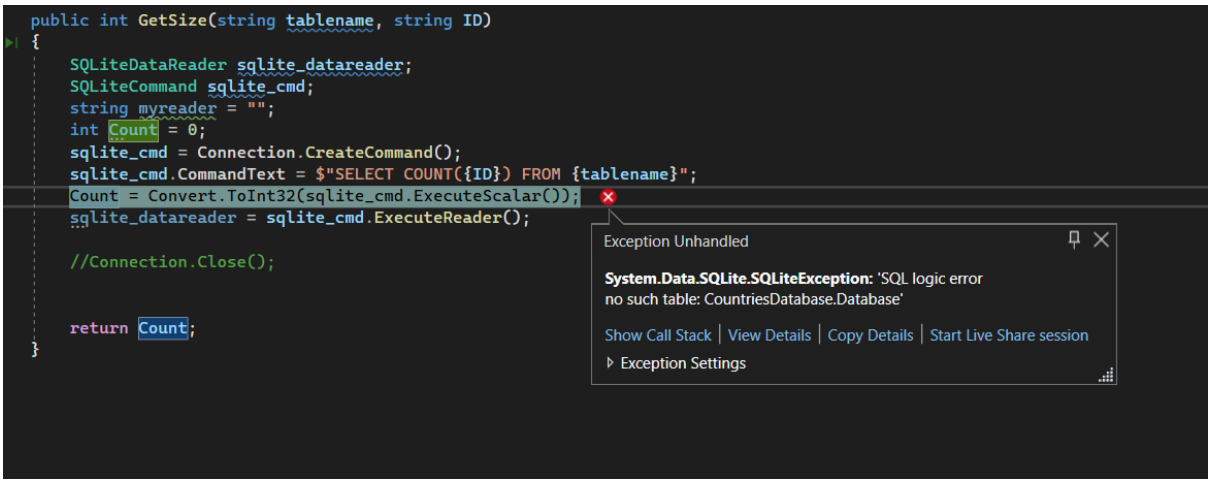
```
public class EditFilePageViewModel: ObservableObject
{
    private MainWindowViewModel _parent;
    private ObservableCollection<string> _questionBank = new ObservableCollection<string>();
    private ObservableCollection<string> _questions = new ObservableCollection<string>();
    private ObservableCollection<string> _answers = new ObservableCollection<string>();
    private List<string> _test = new List<string>();
    private string _selectedQuestionBankName = "";
    private string _selectedQuestion = "";
    private string _selectedAnswer = "";
    private string _updatedQuestion = "";
    private string _updatedAnswer = "";
```

- In order to update dynamically, the List<string> is changed to an Observable collection.

COUNTRY DATABASE ERRORS:



- COUNTRIES DATABASE, when updating the structure of the database, needed to drop databases first, especially with new columns being added.

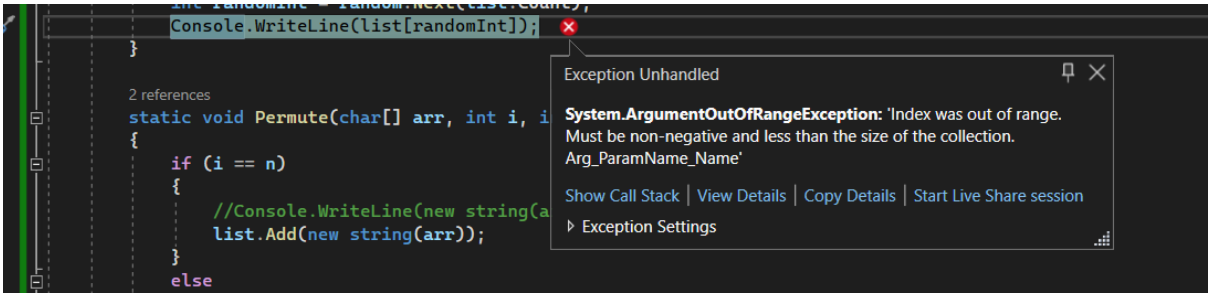


- Incorrect Table name, in an attempt to remove repeating data.

FIXED:

```
int randomContinent = random.Next(databases.Count);
int NoCountries = 0;
switch (randomContinent)
{
    case 0:
        NoCountries = databases[randomContinent].GetSize($"Africa", "CountryID");
        break;
    case 1:
        NoCountries = databases[randomContinent].GetSize($"Asia", "CountryID");
        break;
    case 2:
        NoCountries = databases[randomContinent].GetSize($"Europe", "CountryID");
        break;
    case 3:
        NoCountries = databases[randomContinent].GetSize($"NorthAmerica", "CountryID");
        break;
    case 4:
        NoCountries = databases[randomContinent].GetSize($"SouthAmerica", "CountryID");
        break;
    case 5:
        NoCountries = databases[randomContinent].GetSize($"Oceania", "CountryID");
        break;
    default:
        break;
}
```

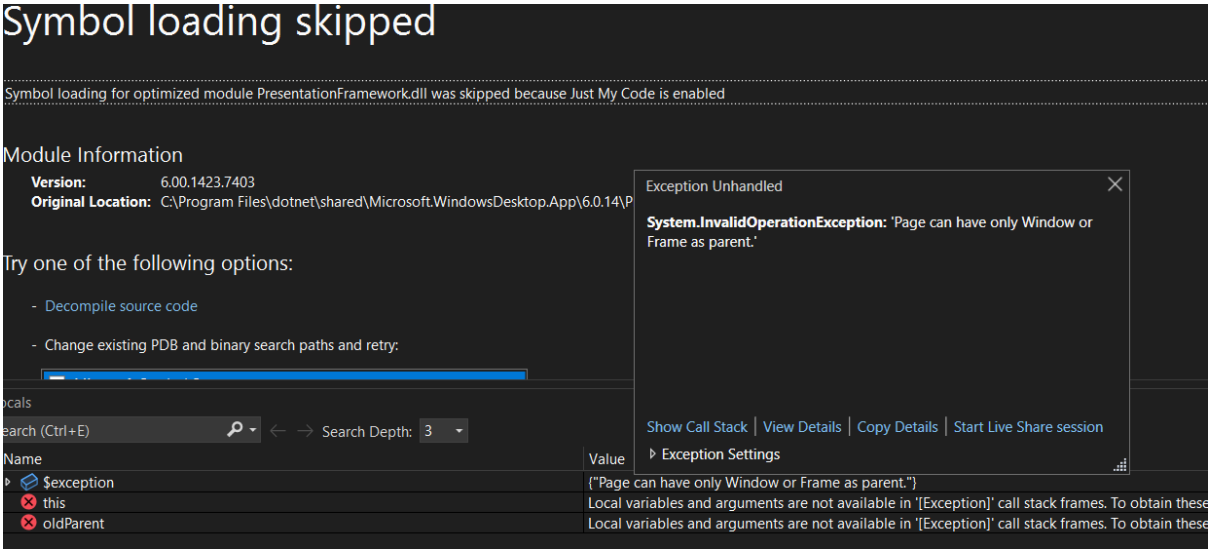
- Using a SWITCH/CASE statement to alter the table names.



- PERMUTE FUNCTION not working: not returning a country name, because it can't handle if multiple strings are returned (in this practice program, each time the program was run, every country was added again).

- PERMUTE FUNCTION incorrect parameters for NEXT function -> Country IDs start at 1 NOT 0. -> original was int RandomCountry = random.Next(NoCountries) -> Needed random.Next(1, list.Count + 1), as the first is inclusive and then 2nd is exclusive.

GAME PAGE ERROR:




```
<Page x:Class="NEA_Project.Pages.QuizPage"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      xmlns:local="clr-namespace:NEA_Project.Pages"
      mc:Ignorable="d"
      d:DesignHeight="450" d:DesignWidth="800"
      Title="QuizPage">

    <Grid>

    </Grid>
</Page>
```

- GAME MENU PAGE: Crashed when opened, caused by the QUIZ PAGE being a Page not User Control.

Name: Gabriella Emerson Candidate Number: 7346 Centre Number: 61853

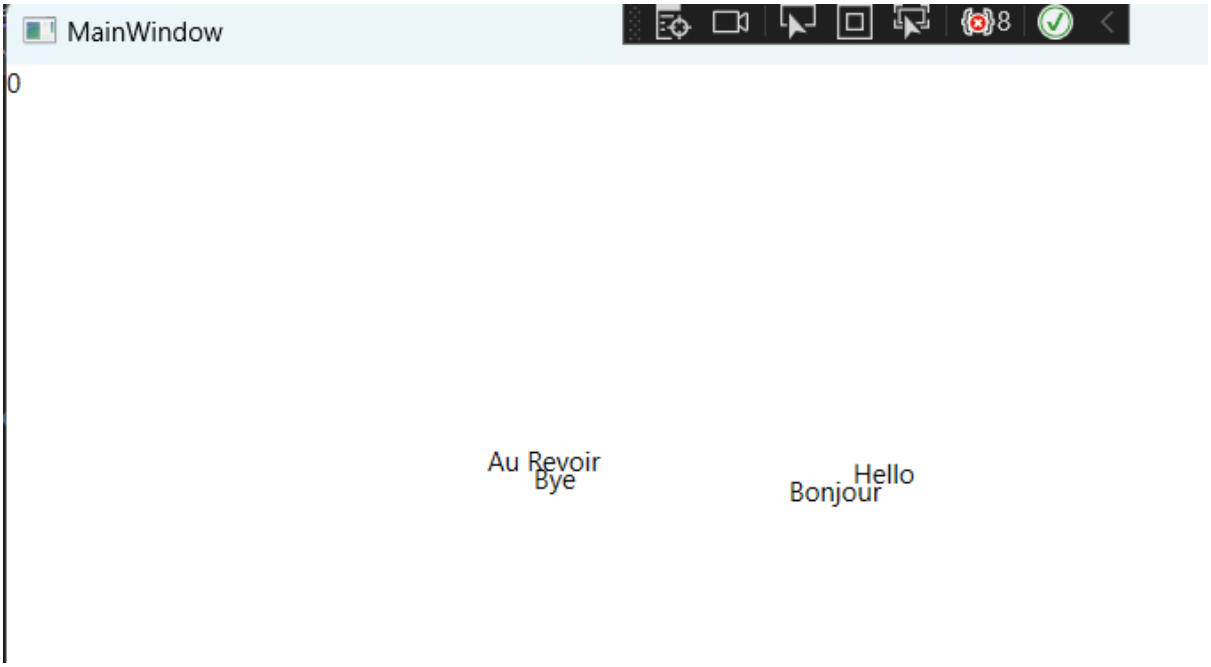
FIXED:

```
16  namespace NEA_Project.Pages
17  {
18      /// <summary>
19      /// Interaction logic for QuizPage.xaml
20      /// </summary>
        2 references
21      public partial class QuizPage : UserControl
22      {
            0 references
23          public QuizPage()
24          {
25              InitializeComponent();
26          }
27      }
28  }
29
```

```
1   <UserControl x:Class="NEA_Project.Pages.QuizPage"
2                xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3                xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4                xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
5                xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
6            xmlns:local="clr-namespace:NEA_Project.Pages"
7            mc:Ignorable="d"
8            d:DesignHeight="450" d:DesignWidth="800">
9
10       <Grid>
11
12       </Grid>
13   </UserControl>
14
```

PAIRS GAME PAGE ERRORS:

ERROR: NOTHING HAPPENS WHEN HIT



FIXED: Missing the Data Binding for the dependency properties

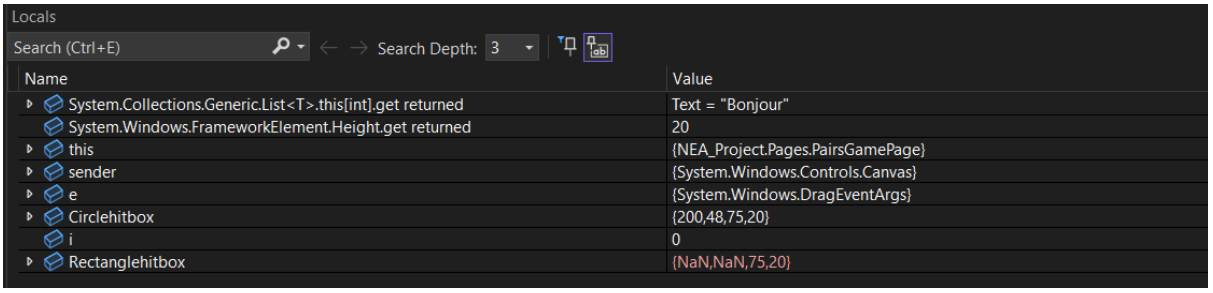
```
<UserControl Grid.Column="0" Grid.Row="0">
    <UserControl.Style>
        <Style TargetType="UserControl">
            <Setter Property="Visibility" Value="Collapsed" />
            <Style.Triggers>
                <DataTrigger Binding="{Binding Path=CurrentPage}" Value="{x:Static constant:ViewStates.PairsGamePage}">
                    <Setter Property="Visibility" Value="Visible" />
                </DataTrigger>
            </Style.Triggers>
        </Style>
    </UserControl.Style>

    <page:PairsGamePage DataContext="{Binding PairsGameViewModel}" CheckPairCommand ="{Binding CheckPairCommand}"
                        TextBlockContains ="{Binding TextBlockContains}" TextBlock2Contains ="{Binding TextBlock2Contains}"
                        PairFound="{Binding PairFound}"/>
</UserControl>
```

ERROR: TextBoxes would match with invisible boxes
FIXED: Removed items from the canvas.

```
private void RefreshButton_Click(object sender, RoutedEventArgs e)
{
    for (int i = textblocks.Count - 1; i >= 0; i--)
    {
        canvas.Children.Remove(textblocks[i]);
        textblocks.RemoveAt(i);
    }
    CreateTextBlocks();

}
```

ERROR: Pairs does not use selected question bank for the Question and Answers, it is over written in the PairsPage.xaml.cs
FIXED: Use dependency property on the parent

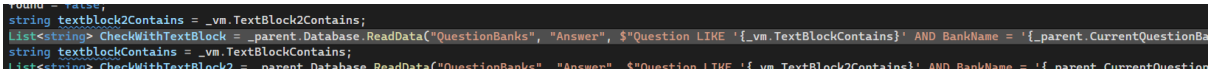
ERROR: Pairs are not returning as valid -> Hitbox is giving 2 Not a Number errors



FIXED: Properly declared position of textblock and added a random starting position too.

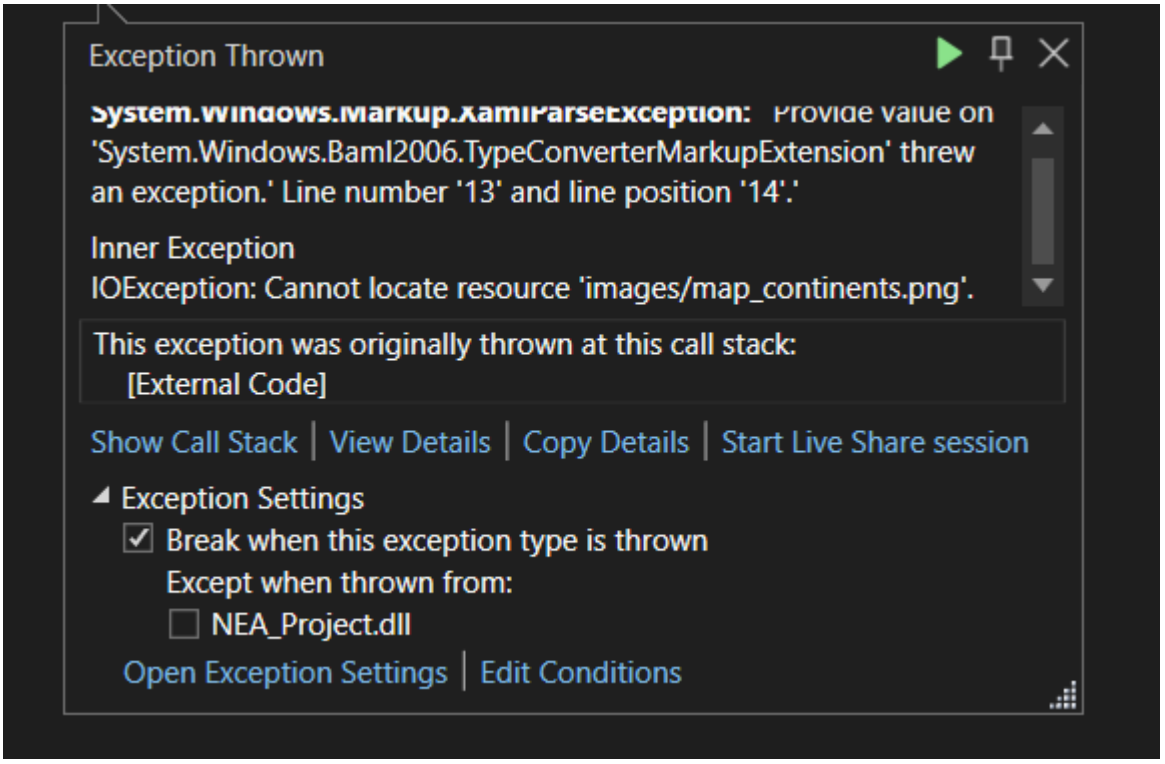
```
for (int i = 0; i < 2; i++)
{
    int height = (int)canvas.Height;
    int width = (int)canvas.Width;
    myDataObject.Change = true;
    int randomNum = random.Next(0, test.Count);
    int randomLeft = random.Next(0, width);
    int randomTop = random.Next(0, height);
    TextBlock textblock = new TextBlock();
    TextBlock textblock1 = new TextBlock();
    textblock.SetBinding(TextBlock.TextProperty, Question);
    textblock1.SetBinding(TextBlock.TextProperty, Answer);
    textblock.Width = 75;
    textblock1.Width = 75;
    textblock1.Height = 75;
    textblock.Height = 75;
    canvas.Children.Add(textblock);
    canvas.Children.Add(textblock1);
    textblocks.Add(textblock);
    textblocks.Add(textblock1);
    Canvas.SetLeft(textblock, randomLeft);
    Canvas.SetTop(textblock, randomTop);
    myDataObject.Change = false;
}
```

ERROR: _vm.TextBlockContains is empty



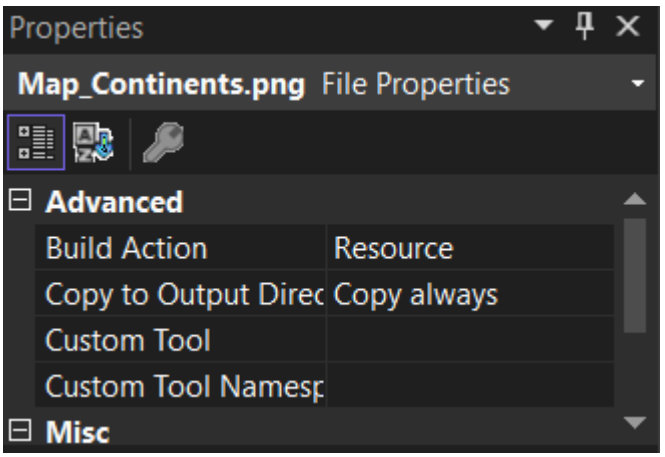
FIXED: TextBlockContainsProperty was missing TWO WAY BINDING

IMAGE DISPLAY ERRORS:



- Relative path does not work for the images on map (this is from main map)

FIXED:

- Added a method in MAIN WINDOW to get the image file paths and bound the image sources to a FilePath Variable
- Changed Image Properties to Resource.
- 

Test Table

Test No	Objectives	Section / Form	Test	Expected Result	Test Type	Result
1	1 - Login System	START PAGE	Sign Up Button Pressed	Changes page to Sign Up Page	Normal	Pass

2	1 - Login System	START PAGE	Login in button pressed	Changes page to Login Page	Normal	Pass
3	1 - Login System	SIGN UP PAGE	Verify button pressed	Changes page to Home Page	Normal - new username	Pass
4	1 - Login System	SIGN UP PAGE	Verify button pressed	Pop-up window	Normal - existing username	Pass
5	1 - Login System	SIGN UP POP UP	Yes Button Pressed	Changes page to login Page	Normal	Pass
6	1 - Login System	SIGN UP POP UP	No Button Pressed	Goes back to Sign Up page	Normal	Pass
7	1 - Login System	LOGIN PAGE	Login in button pressed	Changes to Home Page	Normal - correct username and password	Pass
8	1 - Login System	LOGIN PAGE	Login in button pressed	Pop-up window	Normal - incorrect username or password	Pass
9	1 - Login System	LOGIN POP UP	Yes Button Pressed	Changes to Sign up Page	Normal	Pass
10	1 - Login System	LOGIN POP UP	No Button Pressed	Goes back to Login Page	Normal	Pass
11		HOME PAGE	Map Button	Changes to Map Page	Normal	Pass
12		HOME PAGE	Question Bank Menu Button	Changes to Question Bank Menu Page	Normal	Pass
13		HOME PAGE	Game Menu button	Changes to Game Menu Page	Normal	Pass
14		HOME PAGE	User Stats button	Changes to User Stats Page	Normal	Pass
15	3 - Globe	MAP PAGE	North America button	Changes to North America Page	Normal	Pass
16	3 - Globe	MAP PAGE	South America button	Changes to South America Page	Normal	Pass
17	3 - Globe	MAP PAGE	Africa button	Changes to Africa Page	Normal	Pass
18	3 - Globe	MAP PAGE	Europe button	Changes to Europe Page	Normal	Pass

19	3 - Globe	MAP PAGE	Asia button	Changes to Asia Page	Normal	Pass
20	3 - Globe	MAP PAGE	Oceania button	Changes to Oceania Page	Normal	Pass
21	3 - Globe	AFRICA PAGE	Search Country Button	Country's information is displayed	Normal	Pass
22	3 - Globe	AFRICA PAGE	Search Country Button	Nothing should happen, program shouldn't crash	Erroneous - no valid country selected	Pass
23	3 - Globe	AFRICA PAGE	Home Button Pressed	Changes to Home Page	Normal	Pass
24	3 - Globe	ASIA PAGE	Search Country Button	Country's information is displayed	Normal	Pass
25	3 - Globe	ASIA PAGE	Search Country Button	Nothing should happen, program shouldn't crash	Erroneous - no valid country selected	Pass
26	3 - Globe	ASIA PAGE	Home Button Pressed	Changes to Home Page	Normal	Pass
27	3 - Globe	EUROPE PAGE	Search Country Button	Country's information is displayed	Normal	Pass
28	3 - Globe	EUROPE PAGE	Search Country Button	Nothing should happen, program shouldn't crash	Erroneous - no valid country selected	Pass
29	3 - Globe	EUROPE PAGE	Home Button Pressed	Changes to Home Page	Normal	Pass
30	3 - Globe	NORTH AMERICA PAGE	Search Country Button	Country's information is displayed	Normal	Pass
31	3 - Globe	NORTH AMERICA PAGE	Search Country Button	Nothing should happen, program shouldn't crash	Erroneous - no valid country selected	Pass
32	3 - Globe	NORTH AMERICA PAGE	Home Button Pressed	Changes to Home Page	Normal	Pass
33	3 - Globe	SOUTH AMERICA PAGE	Search Country Button	Country's information is displayed	Normal	Pass

34	3 - Globe	SOUTH AMERICA PAGE	Search Country Button	Nothing should happen, program shouldn't crash	Erroneous - no valid country selected	Pass
35	3 - Globe	SOUTH AMERICA PAGE	Home Button Pressed	Changes to Home Page	Normal	Pass
36	3 - Globe	OCEANIA PAGE	Search Country Button	Country's information is displayed	Normal	Pass
37	3 - Globe	OCEANIA PAGE	Search Country Button	Nothing should happen, program shouldn't crash	Erroneous - no valid country selected	Pass
38	3 - Globe	OCEANIA PAGE	Home Button Pressed	Changes to Home Page	Normal	Pass
39	2 - Quiz System	GAME MENU	Question Bank ComboBox	A Question bank is selected and the selected question bank variable is updated	Normal	Pass
40	2 - Quiz System	GAME MENU	Quiz Button Pressed	Changes to Quiz Page	Normal	Pass
41	2 - Quiz System	GAME MENU	Pairs Game Button Pressed	Changes to Pair Game Page	Normal	Pass
42	2 - Quiz System	GAME MENU	Word Scramble Game Button Pressed	Changes to Word Scramble Page	Normal	Pass
43	2 - Quiz System	GAME MENU	Refresh Button Pressed	Updates the ComboBox (most noticable after a new question bank has been made)	Normal	Pass
44	2 - Quiz System	GAME MENU	Home Button Pressed	Changes to Home Page	Normal	Pass
45	2 - Quiz System	QUIZ	Generate Question Button Pressed	Random Question from selected question bank is displayed	Normal	Pass

46	2 - Quiz System	QUIZ	Check Answer Button Pressed	Score will increase and message acknowleging correct answer	Normal - Correct Answer	Pass
47	2 - Quiz System	QUIZ	Check Answer Button Pressed	Score will decrease and a message showing the correct answer	Normal - Inncorrect Answer	Pass
48	2 - Quiz System	QUIZ	Check Answer Button Pressed	Error message will appear alterting the user they have not generated a question	Erroneous - No Question generated	Pass
49	2 - Quiz System	QUIZ	Finish Button Pressed	Changes to the Game Menu Page	Normal	Pass
50	4 - Matching Game	PAIRS	Finish Button Pressed	Changes to Game Menu Page	Normal	Pass
51	4 - Matching Game	PAIRS	Drag textboxes together	Message displayed to the screen, acknowledging that the user's found a pair and the score increases	Normal - Correct Answer	Pass
52	4 - Matching Game	PAIRS	Drag textboxes together	Message displayed to the screen telling the user that their answer is incorrect and the score decreases	Normal - Incorrect Answer	Pass
53	4 - Matching Game	PAIRS	Refresh Button Pressed	A number of randomly generated questions are output to the screen, depending on the number entered in the How Many Pairs Text Box	Normal	Pass

54	4 - Matching Game	PAIRS	Refresh Button Pressed	Message box telling the user they've input incorrect datatype appears	Erroneous	Pass
55	2 - Quiz System	WORD SCRAMBLE	Generate Question Button Pressed	Random Scrambled Country Name is displayed	Normal	Pass
56	2 - Quiz System	WORD SCRAMBLE	Check Answer Button Pressed	Score will increase and message acknowleging correct answer	Normal - Correct Answer	Pass
57	2 - Quiz System	WORD SCRAMBLE	Check Answer Button Pressed	Score will decrease and a message showing the correct answer	Normal - Inncorrect Answer	Pass
58	2 - Quiz Sytem	WORD SCRAMBLE	Check Answer Button Pressed	Error is handled, the program doesn't crash	Erroneous - No Question generated	Pass
59	2 - Quiz System	WORD SCRAMBLE	Finish Button Pressed	Changes to the Game Menu Page	Normal	Pass
60	2 - Quiz System	WORD SCRAMBLE	Hint1 Button Pressed	Message box showing what continent the country is in	Normal	
61	2 - Quiz System	WORD SCRAMBLE	Hint2 Button Pressed	Message box showing what letter the country begins with	Normal	Pass
62	2 - Quiz System	QUESTION BANK MENU	Create Question Bank Button Pressed	Changes to Create Question Bank Page	Normal	Pass
63	2 - Quiz System	QUESTION BANK MENU	Delete Question Bank Button Pressed	Changes to Delete Question Bank Page	Normal	Pass
64	2 - Quiz System	QUESTION BANK MENU	Edit Questions Button Pressed	Changes to Edit Question Bank Page	Normal	Pass

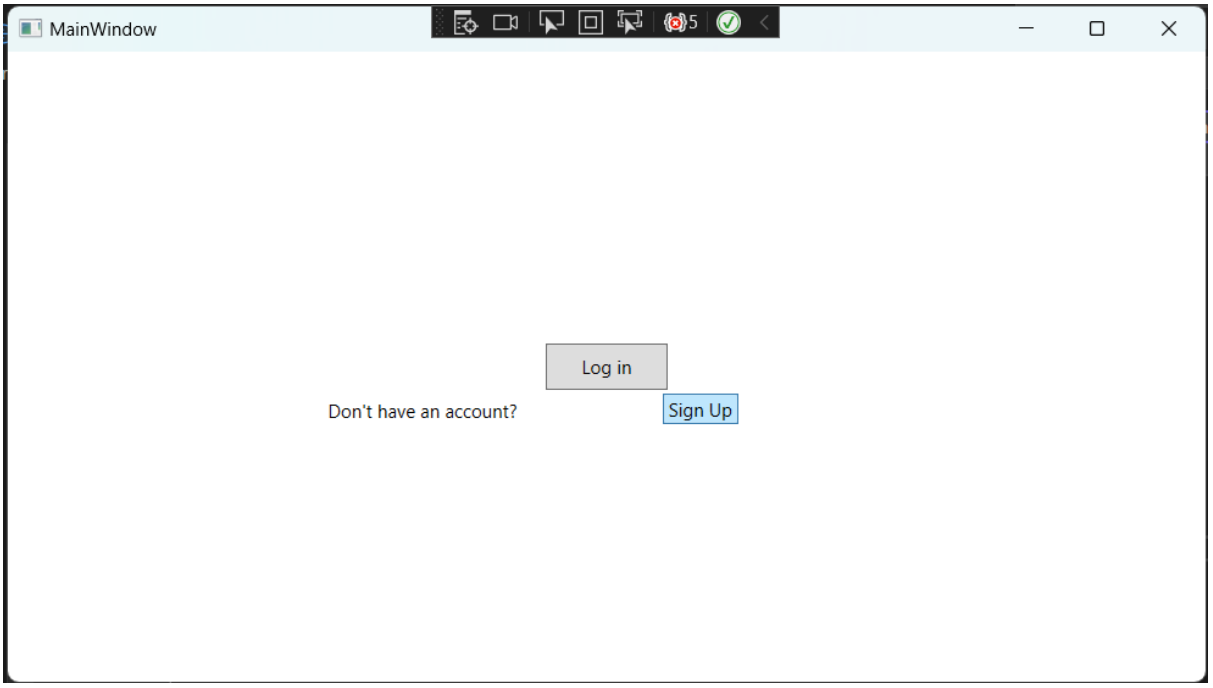
65	2 - Quiz System	QUESTION BANK MENU	Read Question Banks Button Pressed	Changes to Read Question Bank Page	Normal	Pass
66	2 - Quiz System	QUESTION BANK MENU	Home Button Pressed	Changes to Home Page	Normal	Pass
67	2 - Quiz System	QUESTION BANK CREATE	Add Question Button Pressed	No error message should appear	Normal	Pass
68	2 - Quiz System	QUESTION BANK CREATE	Add Question Button Pressed	Error message displayed	Erroneous - No Question Bank Selected	Pass
69	2 - Quiz System	QUESTION BANK CREATE	Add Question Button Pressed	Error message displayed	Erroneous - Either Question or Answer is Null	Pass
70	2 - Quiz System	QUESTION BANK CREATE	Back To Menu Button Pressed	Changes to Question Bank Menu Page	Normal	Pass
71	2 - Quiz System	QUESTION BANK DELETE	Delete Button Pressed	Message Box appears, asking user to confirm	Normal	Pass
72	2 - Quiz System	QUESTION BANK DELETE	Refresh Button Pressed	Updates the ComboBox	Normal	Pass
73	2 - Quiz System	QUESTION BANK DELETE	Back To Menu Button Pressed	Changes to Question Bank Menu Page	Normal	Pass
74	2 - Quiz System	QUESTION BANK EDIT	Generate Question Bank Button Pressed	Message box to notify the user which bank has been selected	Normal	Pass
75	2 - Quiz System	QUESTION BANK EDIT	Update Question	No Error Message	Normal	Pass
76	2 - Quiz System	QUESTION BANK EDIT	Update Question	Error message - telling user to selected a question Bank	Erroneous	Pass
77	2 - Quiz System	QUESTION BANK EDIT	Update Question	Error message - telling user to	Erroneous	Pass

				selected a question		
78	2 - Quiz System	QUESTION BANK EDIT	Update Answer	No Error Message	Normal	Pass
79	2 - Quiz System	QUESTION BANK EDIT	Update Answer	Error message - telling user to selected a question Bank	Erroneous	Pass
80	2 - Quiz System	QUESTION BANK EDIT	Update Answer	Error message - telling user to selected an answer	Erroneous	Pass
81	2 - Quiz System	QUESTION BANK EDIT	Update Answer	Error message - telling user to selected an answer	Erroneous	Pass
82	2 - Quiz System	QUESTION BANK EDIT	Back To Menu Button Pressed	Changes to Question Bank Menu Page	Normal	Pass
83	2 - Quiz System	QUESTION BANK EDIT	Refresh Button Pressed	Updates the ComboBox	Normal	Pass
84	2 - Quiz System	QUESTION BANK READ	Generate Question Bank Button Pressed	Message box displaying the question bank name selected	Normal	Pass
85	2 - Quiz System	QUESTION BANK READ	Show Answer Button Pressed	Message box with the answer	Normal	Pass
86	2 - Quiz System	QUESTION BANK READ	Show Answer Button Pressed	Message box displaying an error message, informing the user they haven't selected a question or a bank	Erroneous - no data selected (Question Bank)	Pass
87	2 - Quiz System	QUESTION BANK READ	Show Answer Button Pressed	Message box displaying an error message, informing the user they haven't selected a question or a bank	Erroneous - no data selected (Question)	Pass

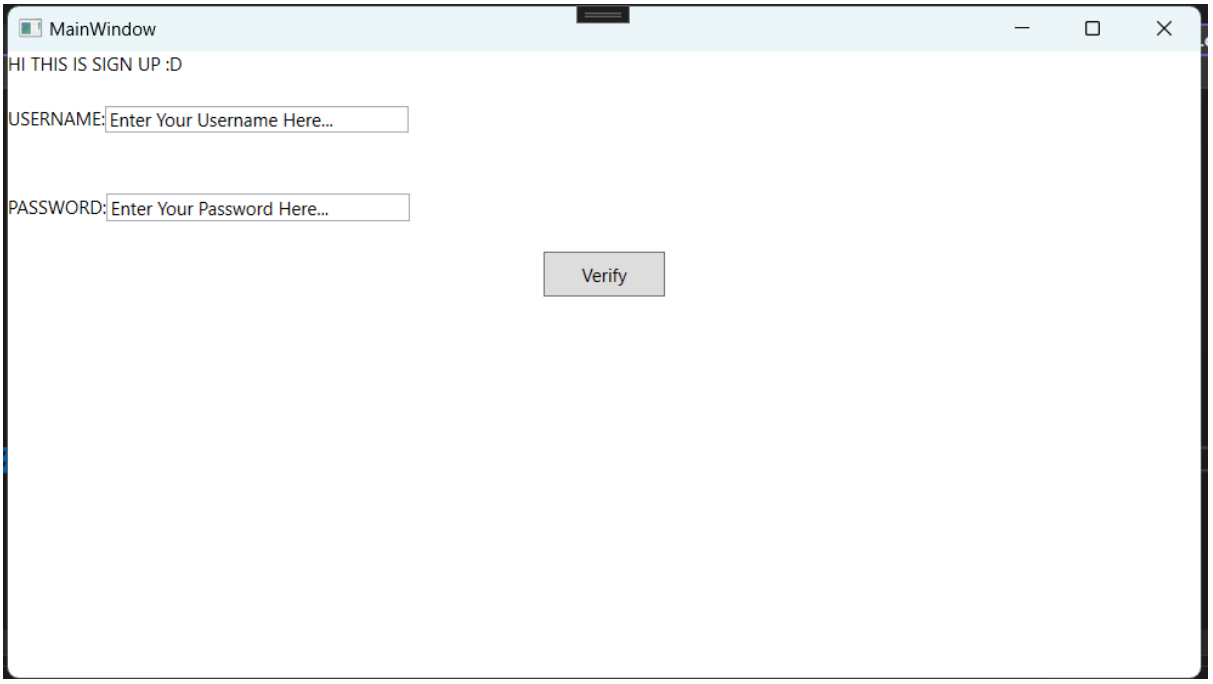
88	2 - Quiz System	QUESTION BANK READ	Back To Menu Button Pressed	Changes to Question Bank Menu Page	Normal	Pass
89	2 - Quiz System	QUESTION BANK READ	Refresh Button Pressed	Updates the ComboBox	Normal	Pass
90	5 - Score System	USERSTATS PAGE	Get Stats Button Pressed	The Highest Score for the user for each game is displayed to the screen	Normal	Pass
91	5 - Score System	USERSTATS PAGE	Get Average Button Pressed	The Average scores for the user for each game is displayed to the screen	Normal	Pass
92	5 - Score System	USERSTATS PAGE	Get Ranking Button Pressed	A message box appears displaying the user's percentile score for the selected game	Normal - Quiz Scores	Pass
93	5 - Score System	USERSTATS PAGE	Get Ranking Button Pressed	A message box appears displaying the user's percentile score for the selected game	Normal - Pairs Scores	Pass
94	5 - Score System	USERSTATS PAGE	Get Ranking Button Pressed	A message box appears displaying the user's percentile score for the selected game	Normal - Word Scramble Scores	Pass
95	5 - Score System	USERSTATS PAGE	Get Ranking Button Pressed	A message box appears with an error message	Erroneous - No category selected	Pass
96	5 - Score System	USERSTATS PAGE	Home Page Button Pressed	Changes to the Home Page	Normal	Pass

Evidence of Test Results

TEST 1:

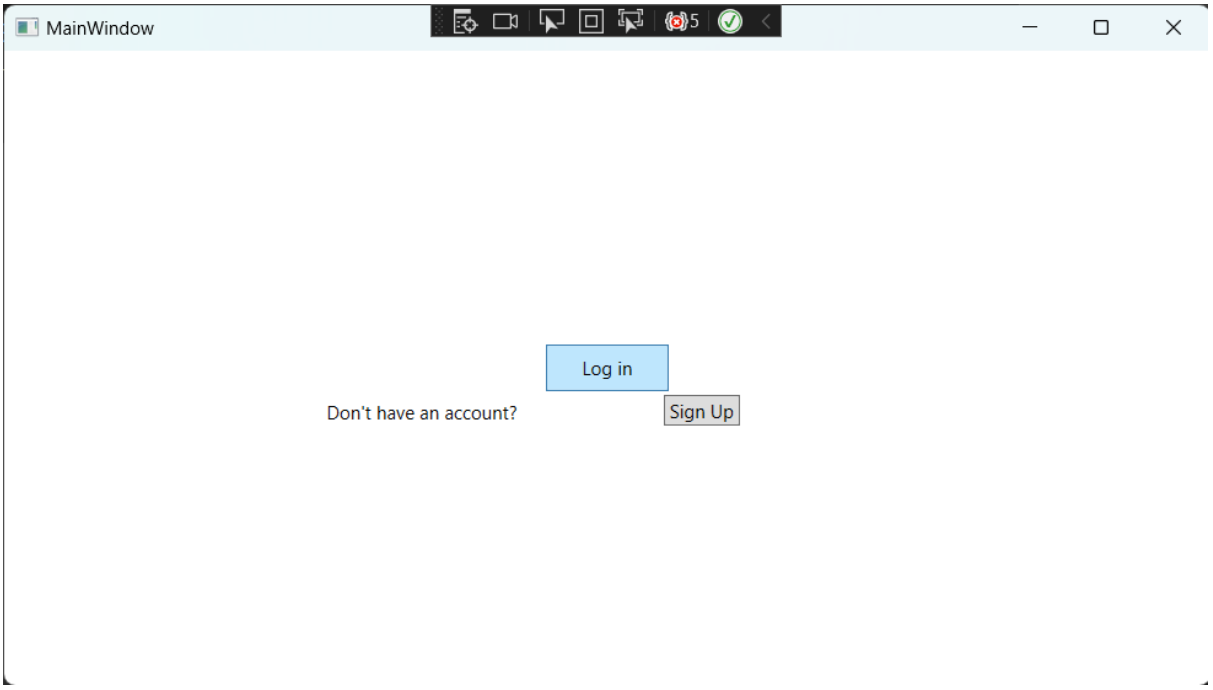


- START PAGE: Sign Up Button Clicked

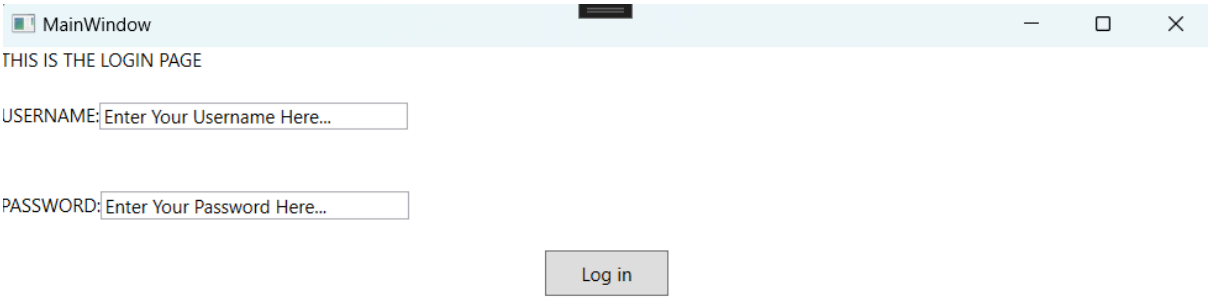


- Successfully changed to the SIGN UP PAGE

TEST 2:

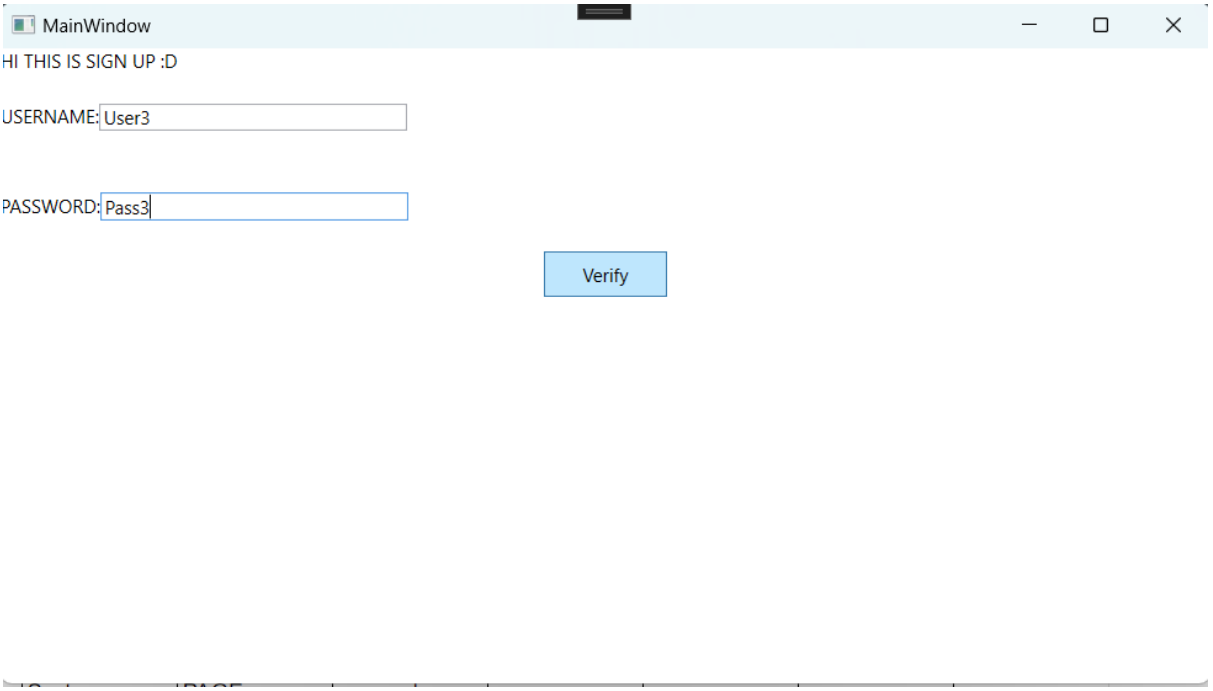


- START PAGE: Login Button Clicked

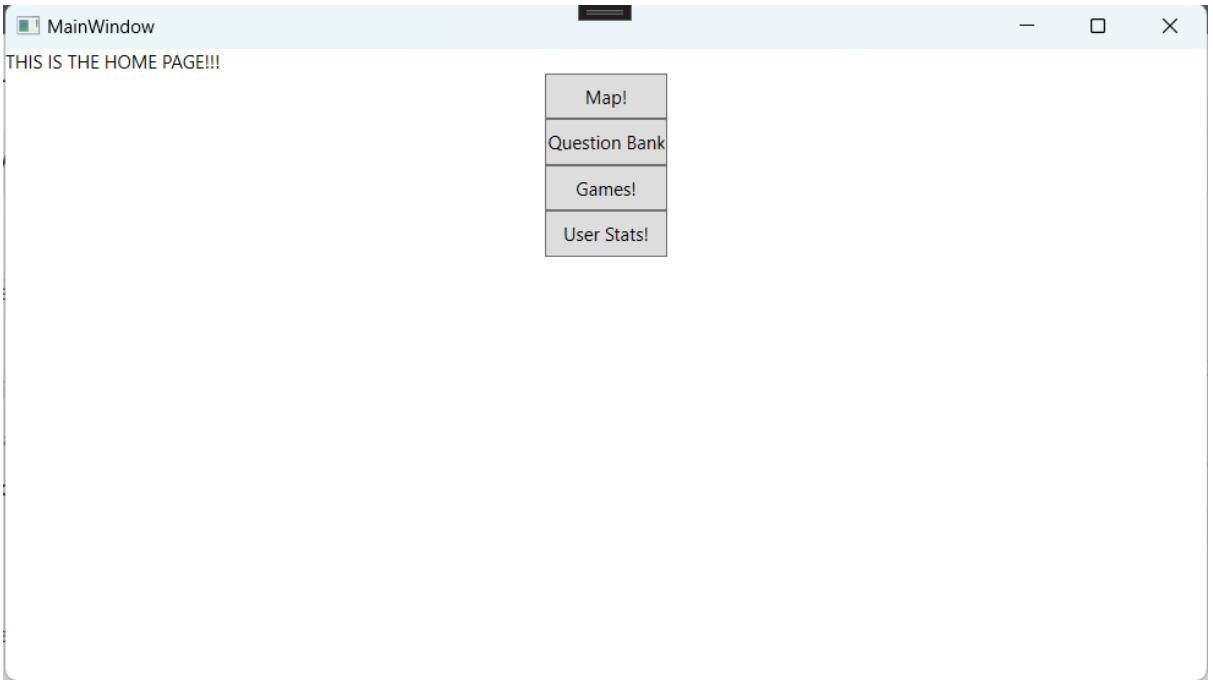


- Successfully changed to the LOGIN PAGE

TEST 3:

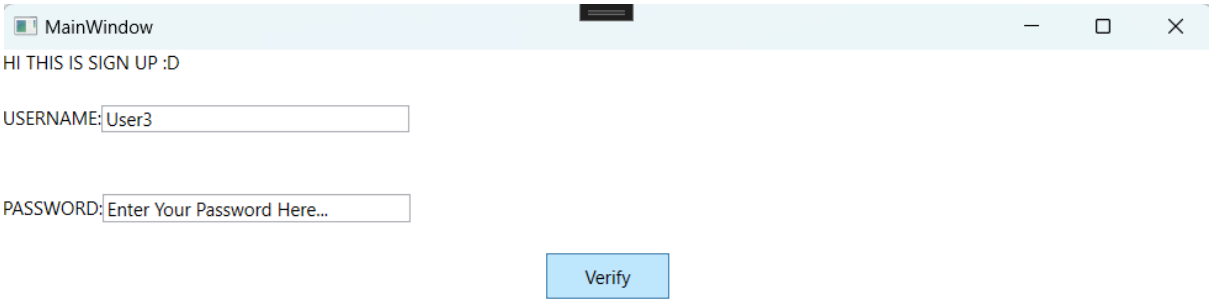


- SIGN UP PAGE: New User details entered and verify button pressed.

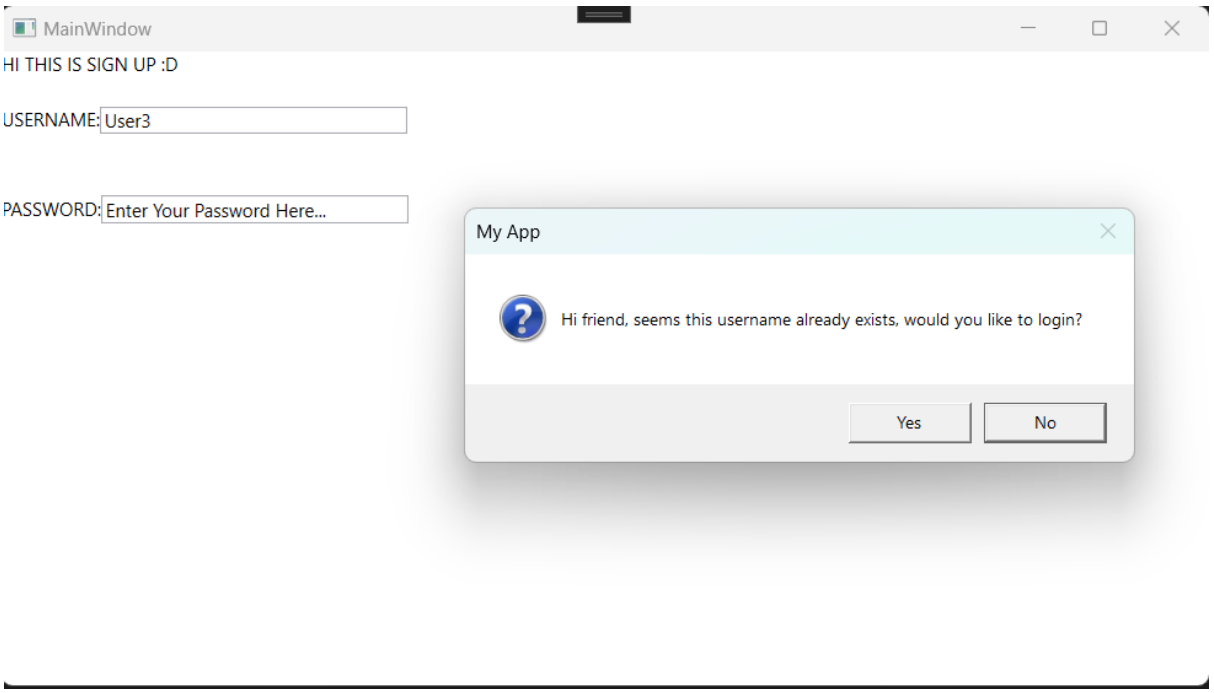


- Successfully changed to the HOME PAGE

TEST 4:

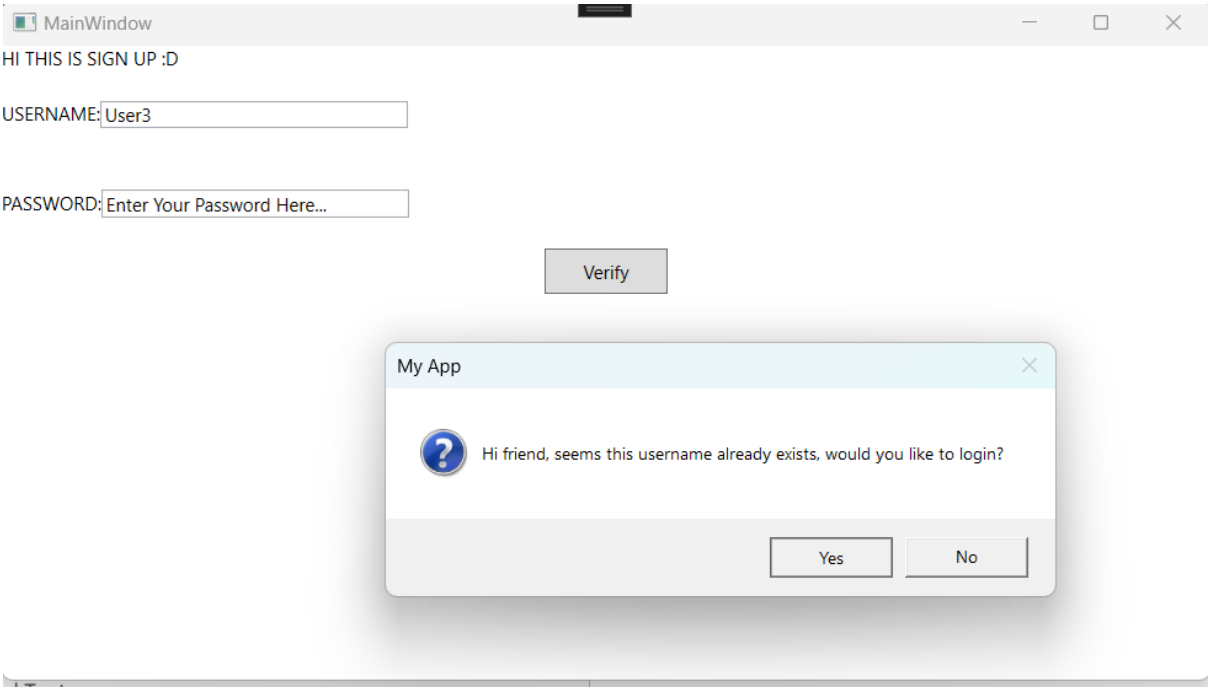


- SIGN UP PAGE: Existing username is entered and verify button is clicked

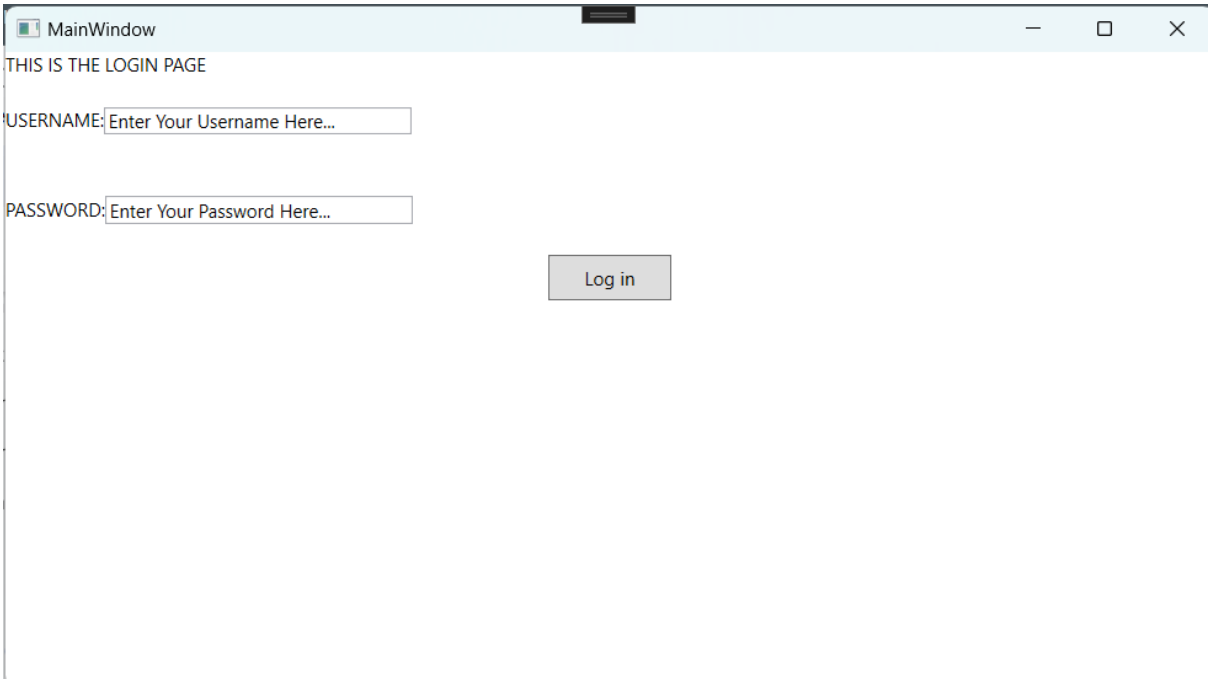


- Successfully displays a message box alerting users that the username already exists.

TEST 5:

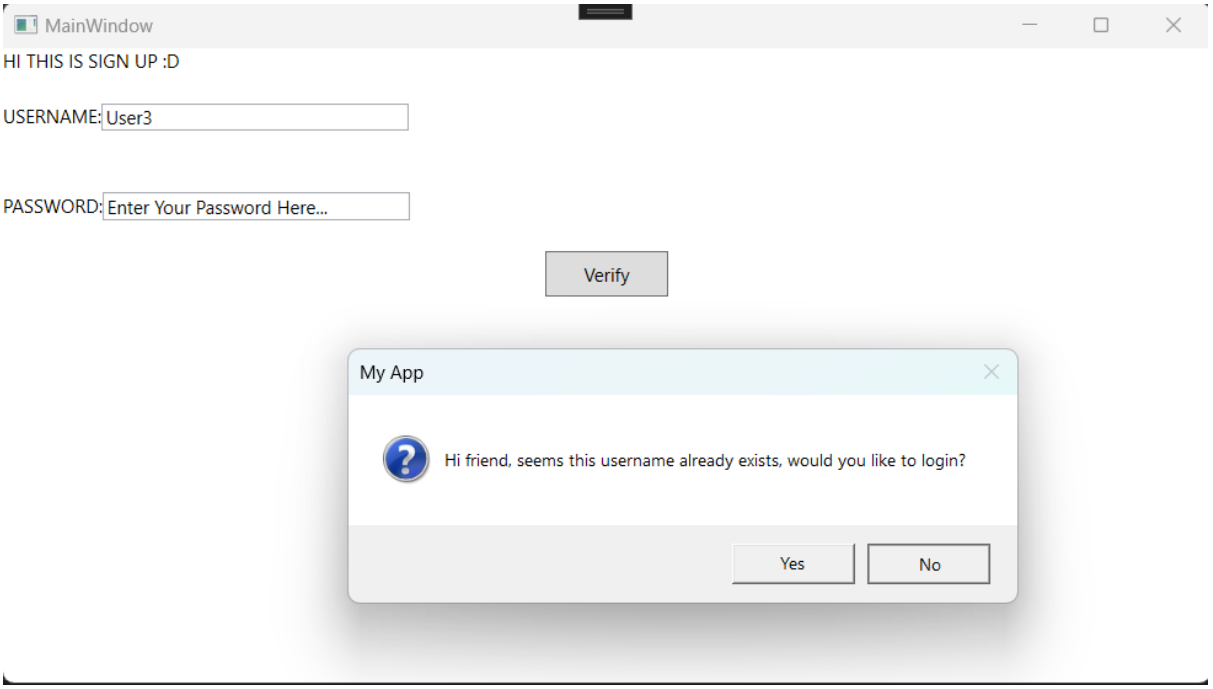


- SIGN UP PAGE POP-UP: 'Yes' Button pressed

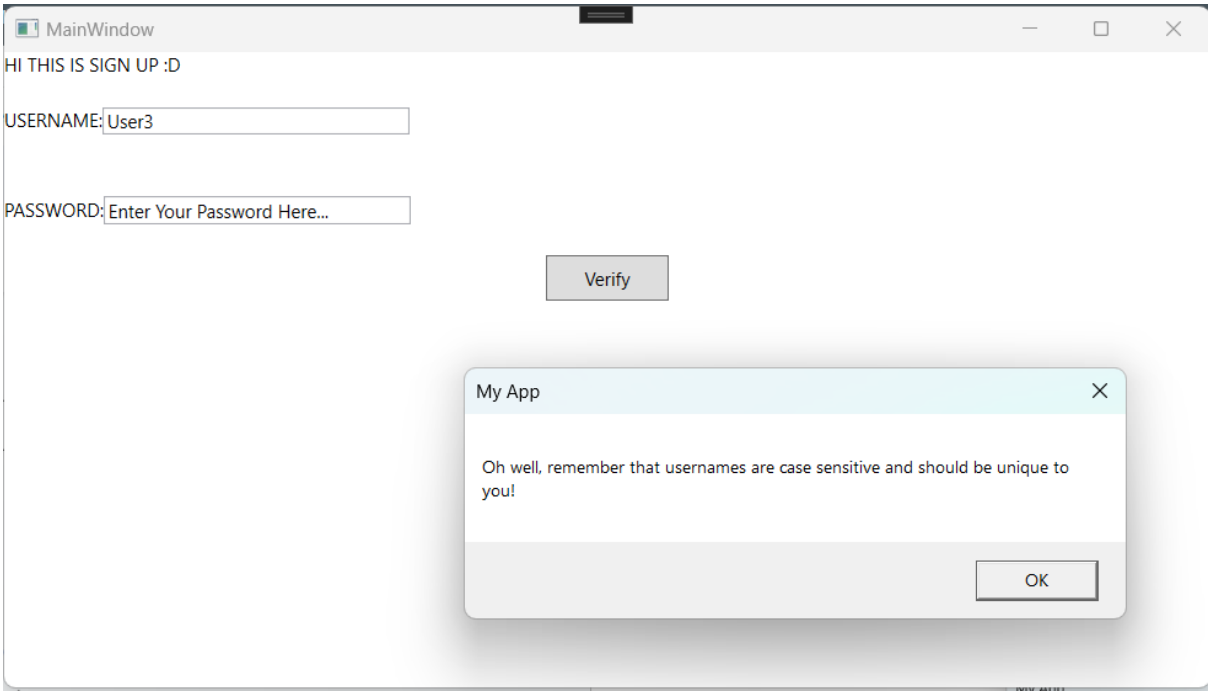


- Successfully changes to the LOGIN PAGE.

TEST 6:

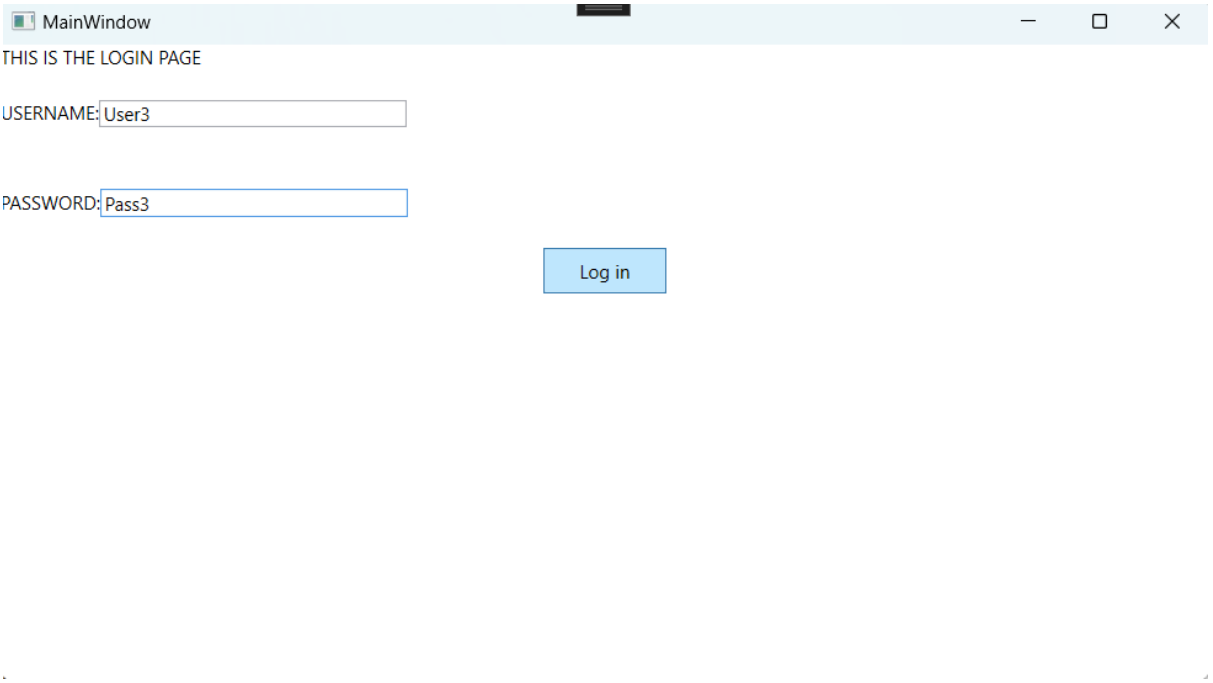


- SIGN UP POP-UP: 'No' Button is pressed.

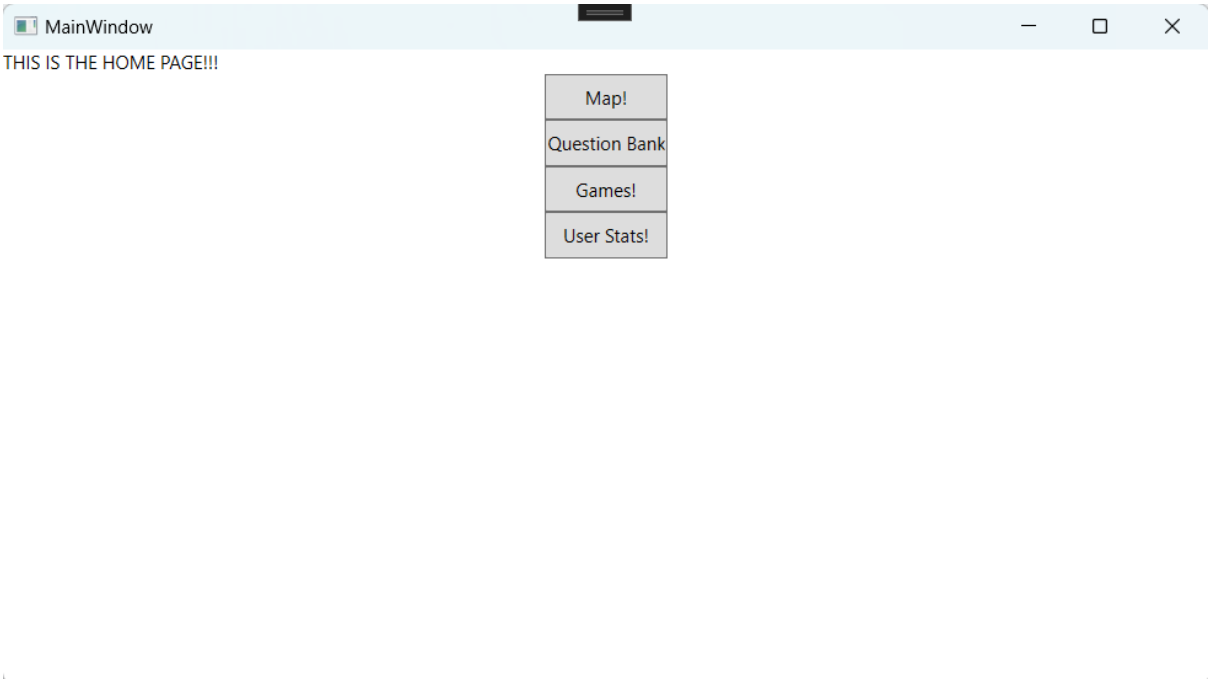


- Message box pops up with a reminder and stays on the SIGN UP PAGE.

TEST 7:

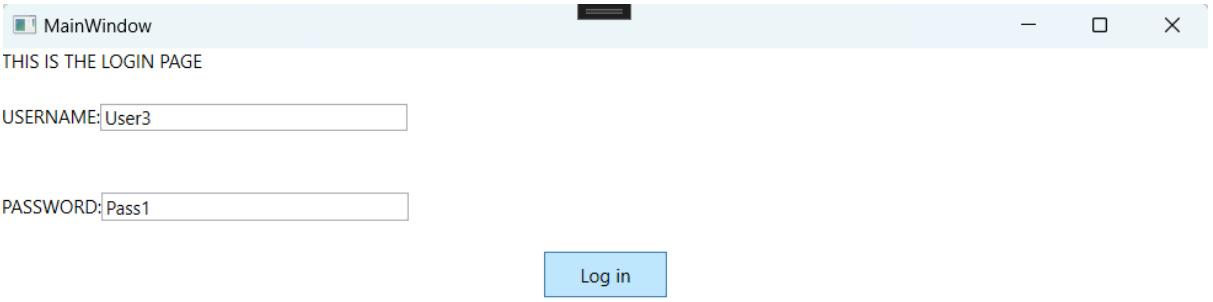


- LOGIN PAGE: Correct Username and Password entered and Login Button pressed.

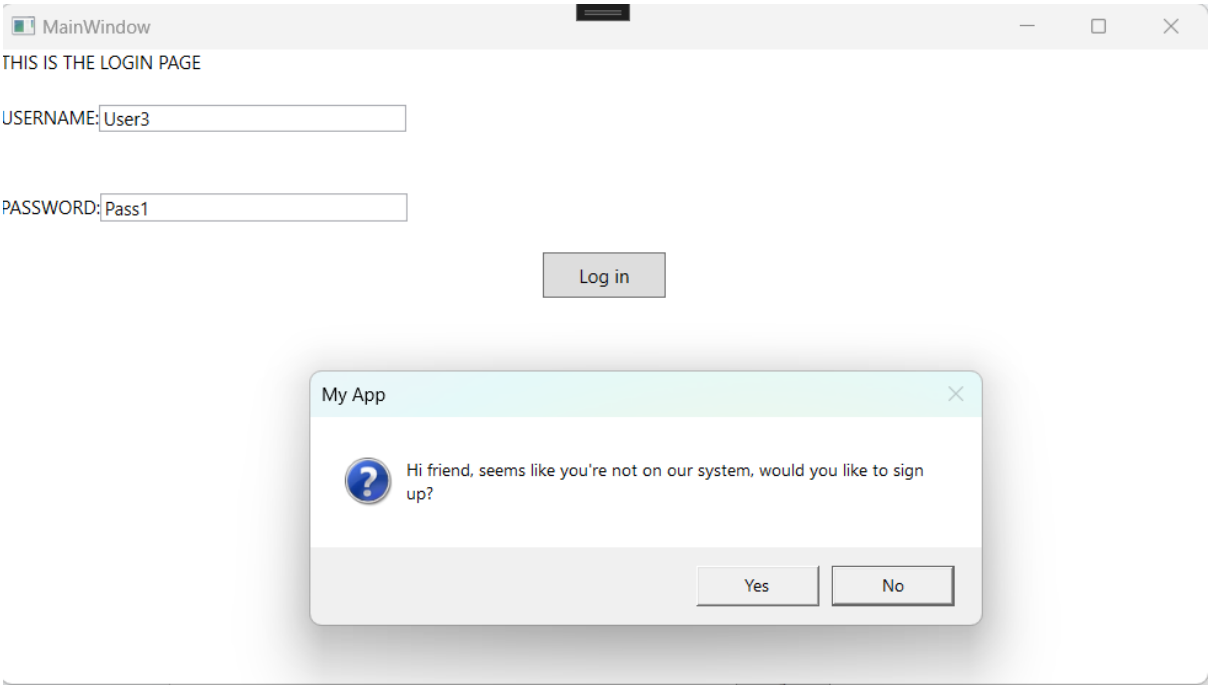


- Successfully changed to the HOME PAGE

TEST 8:

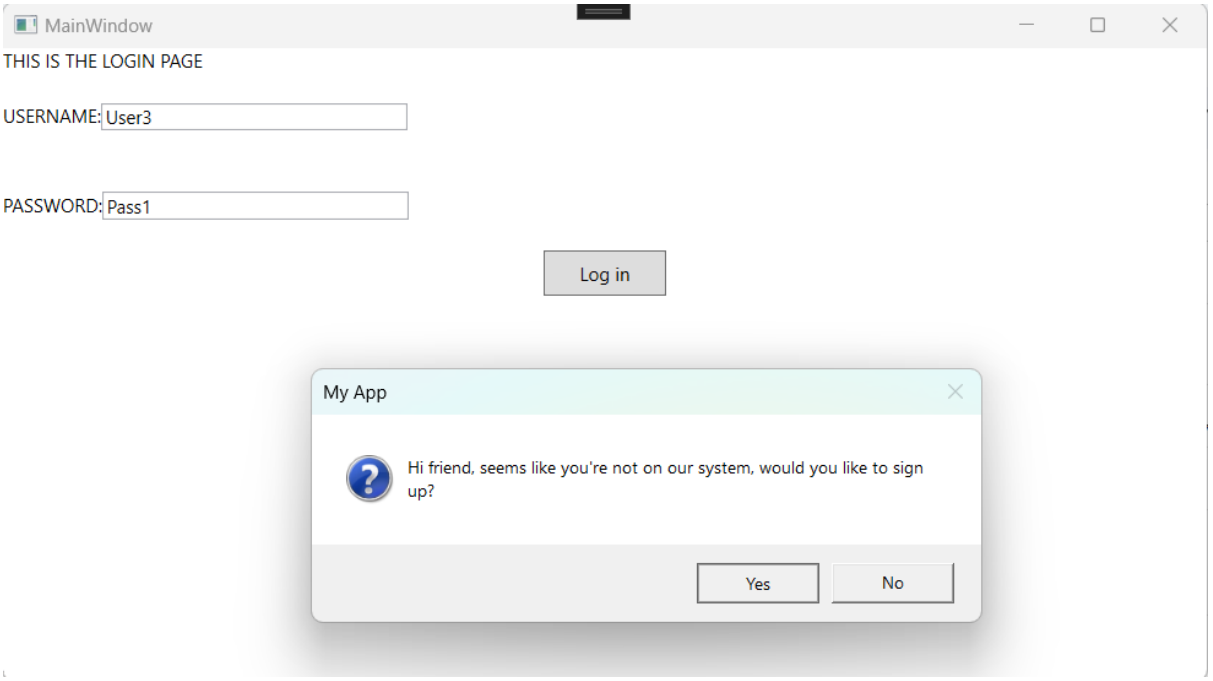


- LOGIN PAGE: Login Button Pressed.

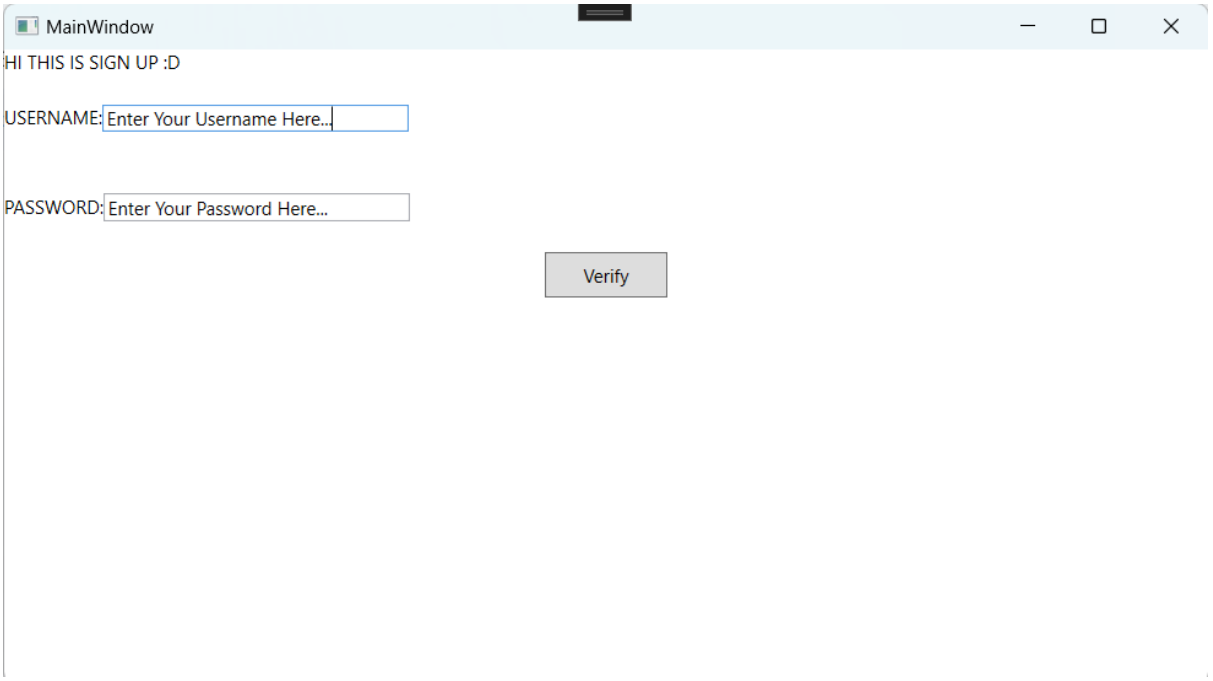


- Pop up successfully displayed, alerting the user that they've entered incorrect credentials.

TEST 9:

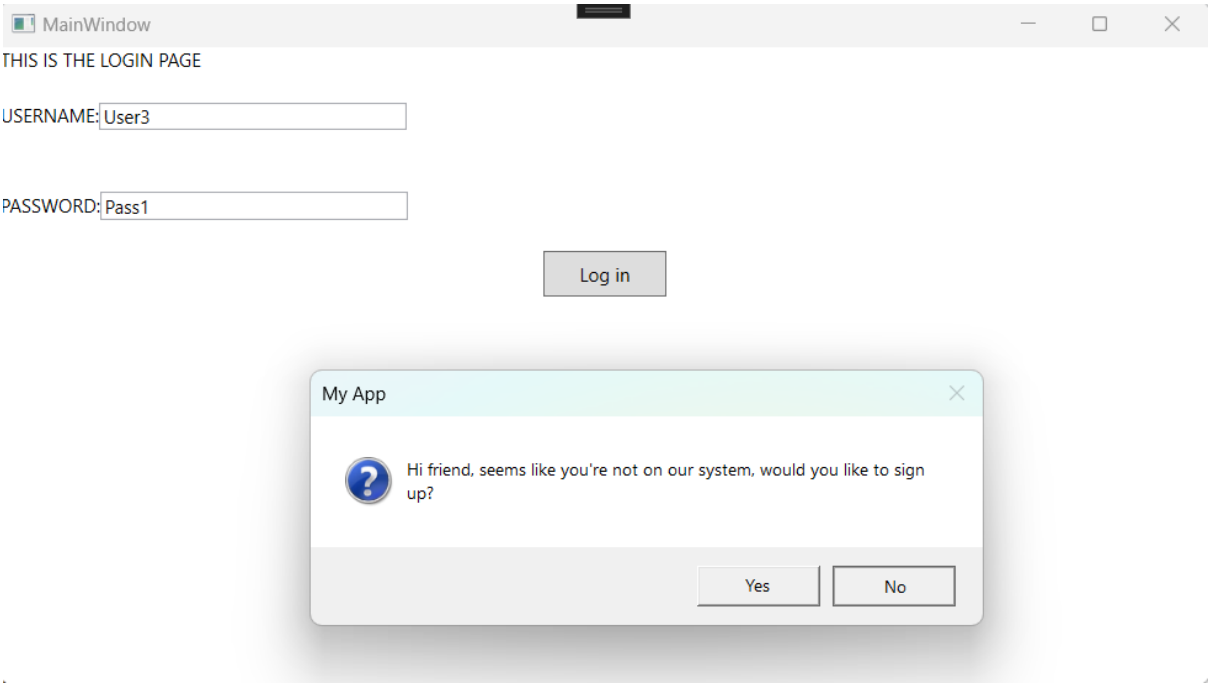


- LOGIN POP-UP: ‘Yes’ Button is pressed.

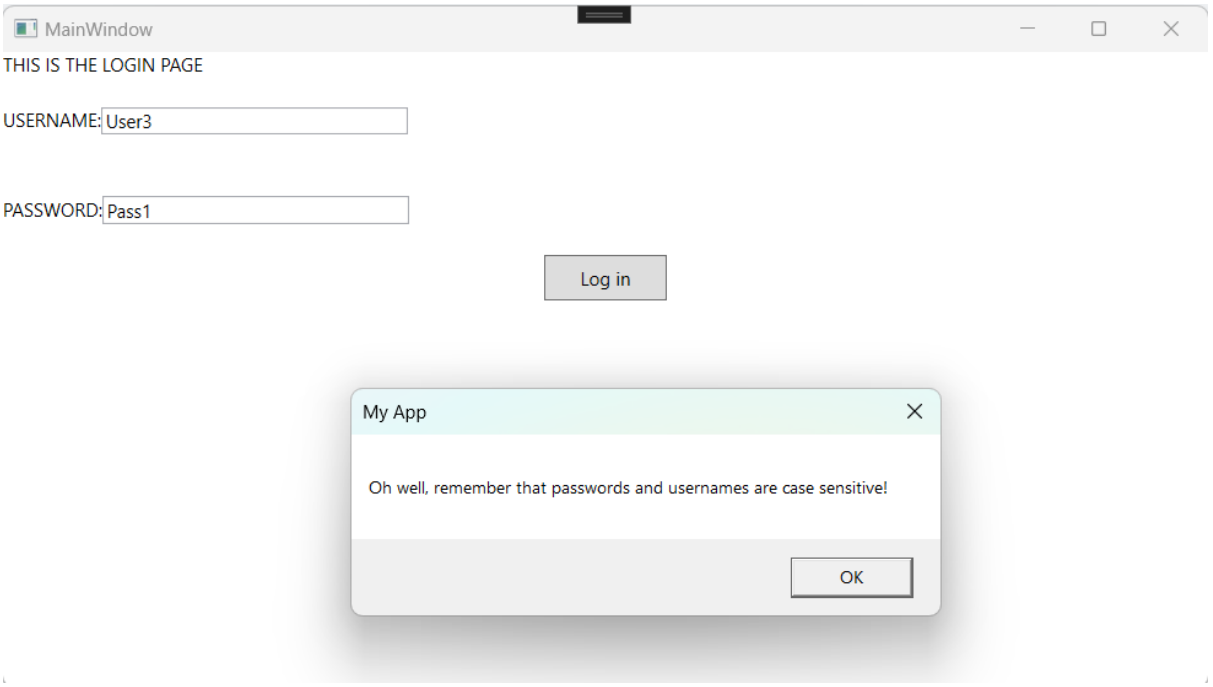


- The page is successfully changed to the SIGN UP PAGE.

TEST 10:

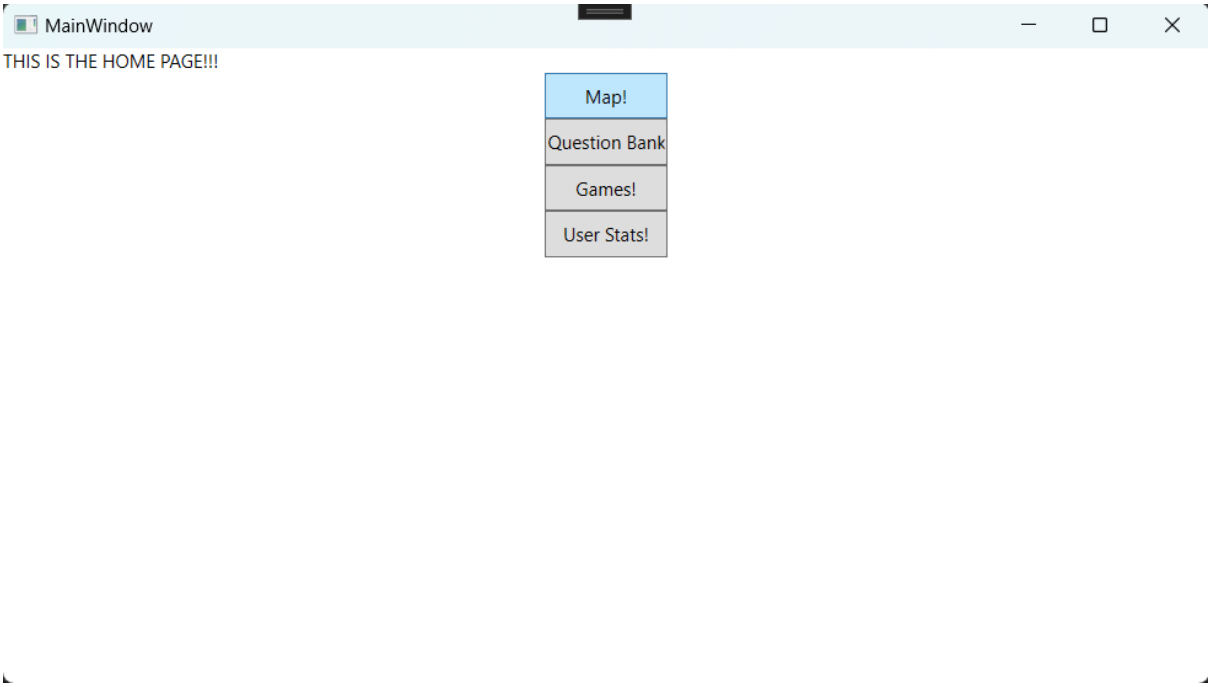


- LOGIN POP-UP: 'No' Button Pressed



- Pop-up that reminds the user to enter details correctly is successfully displayed and remains on the LOGIN PAGE.

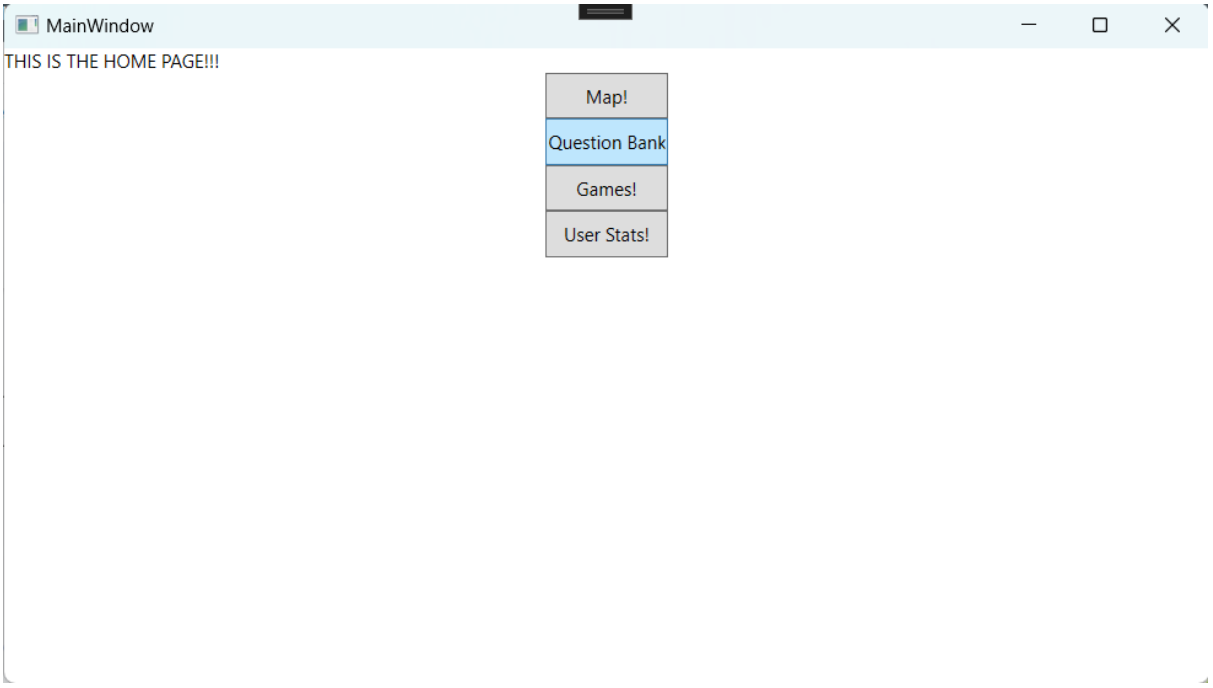
TEST 11:



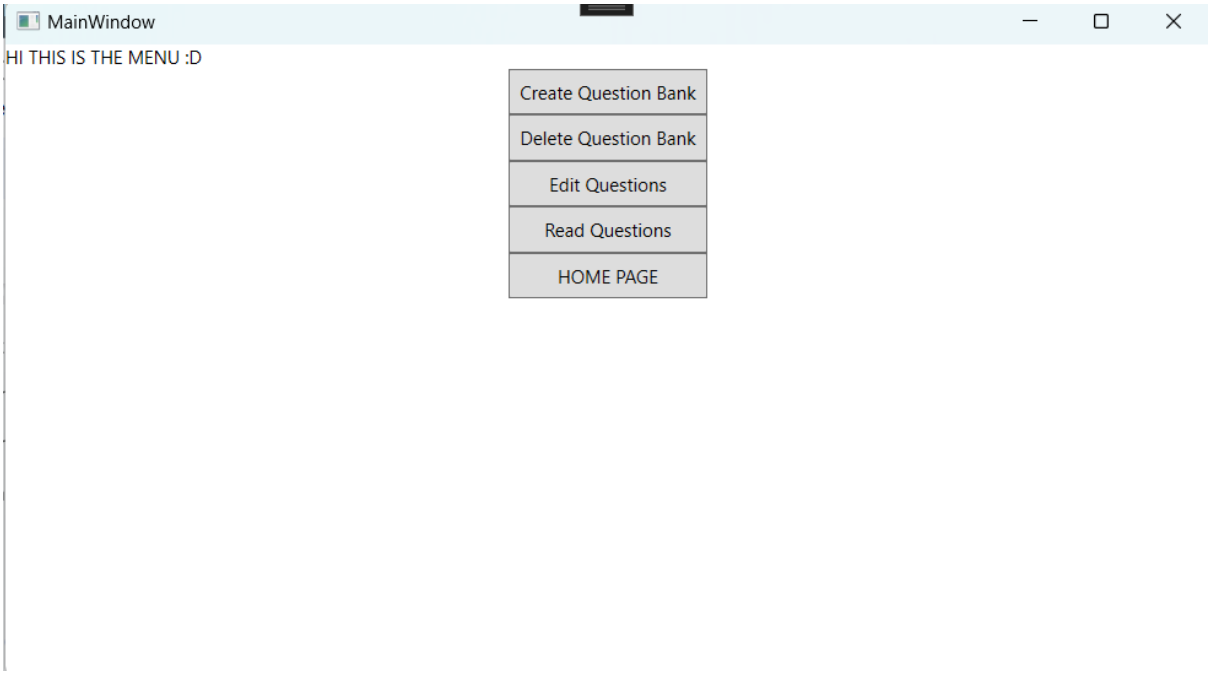
- HOME PAGE: Map Button Pressed.

- Successfully changes to the MAP PAGE.

TEST 12:

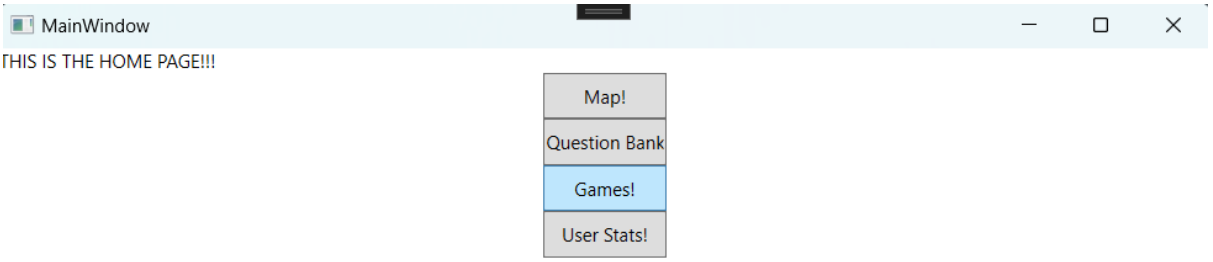


- HOME PAGE: Question Bank Button Pressed

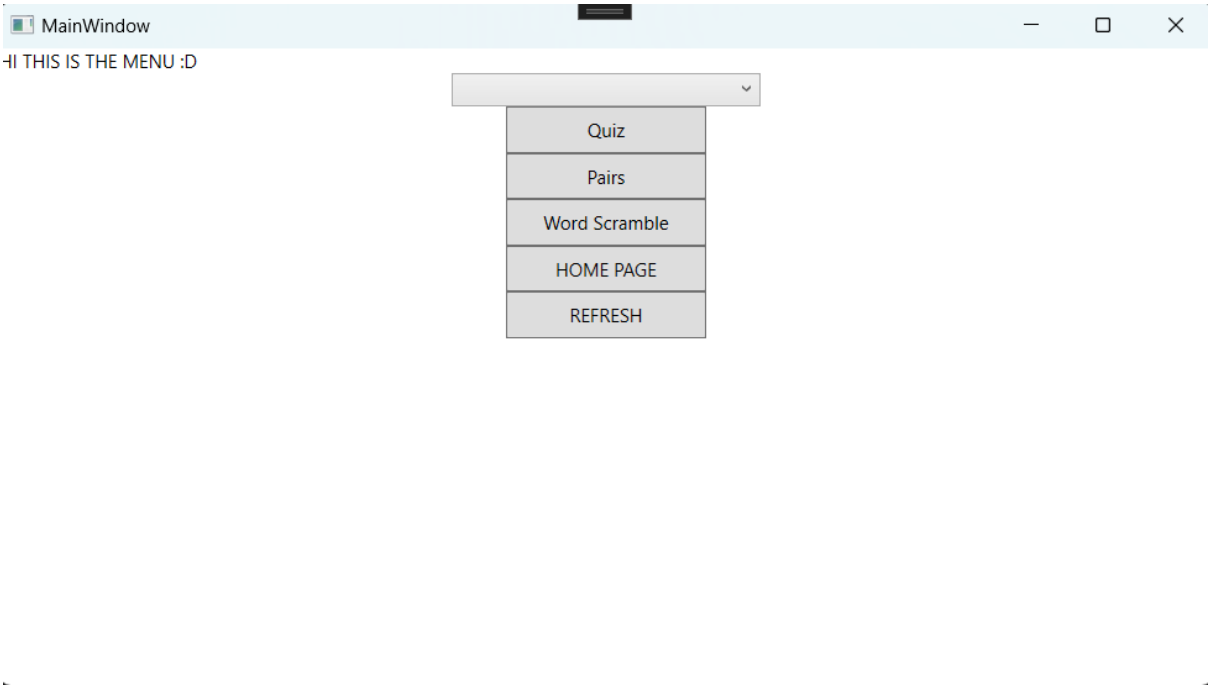


- Successfully changes to the QUESTION BANK MENU PAGE

TEST 13:

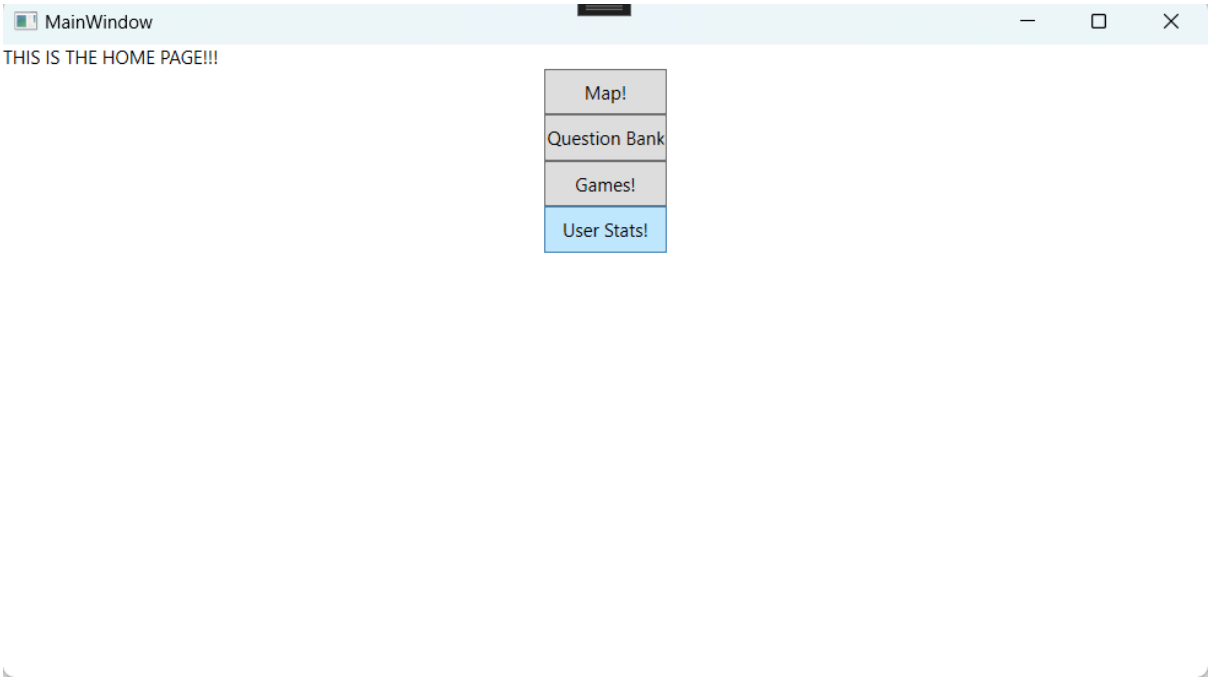


- HOME PAGE: Games Button Pressed.

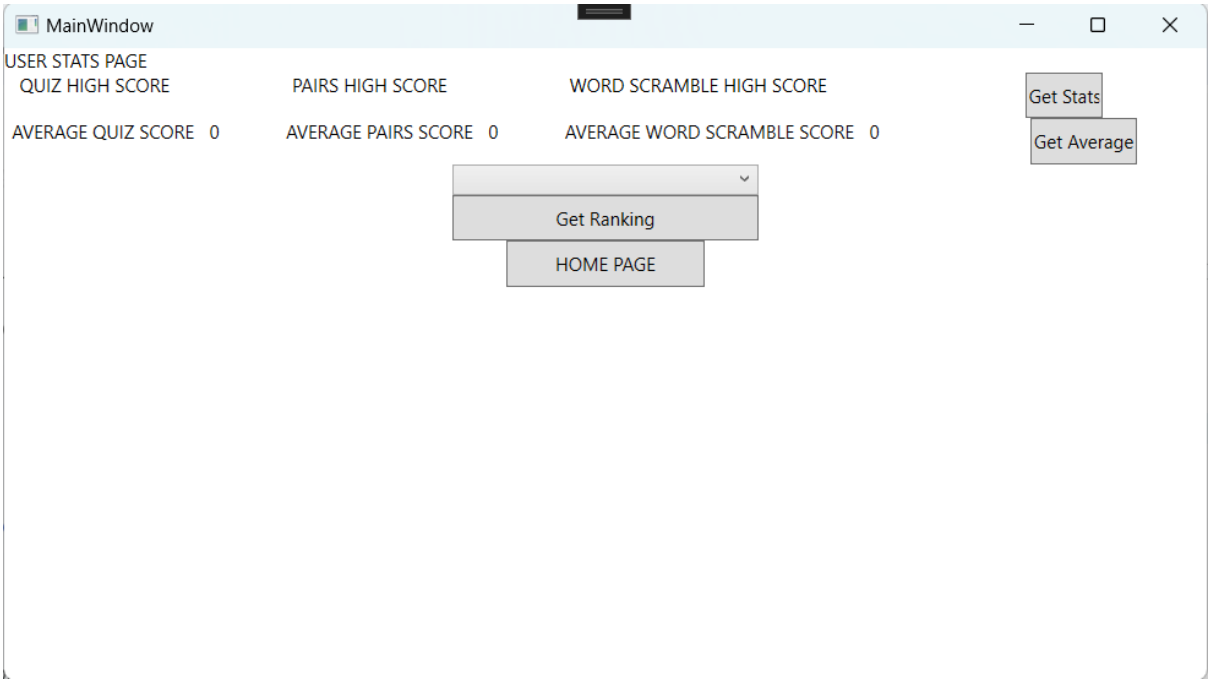


- Successfully Changed to the GAMES MENU PAGE

TEST 14:



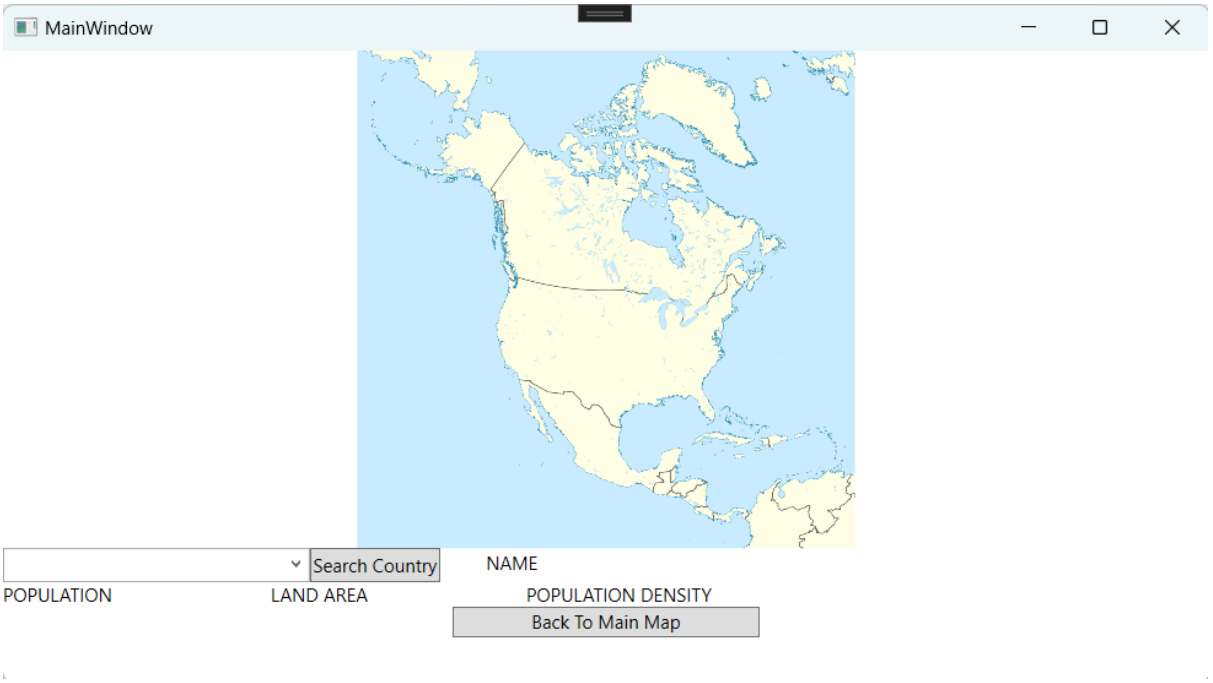
- HOME PAGE: Uses Stats Button Pressed.



- The page has successfully changed to USER STATS PAGE.

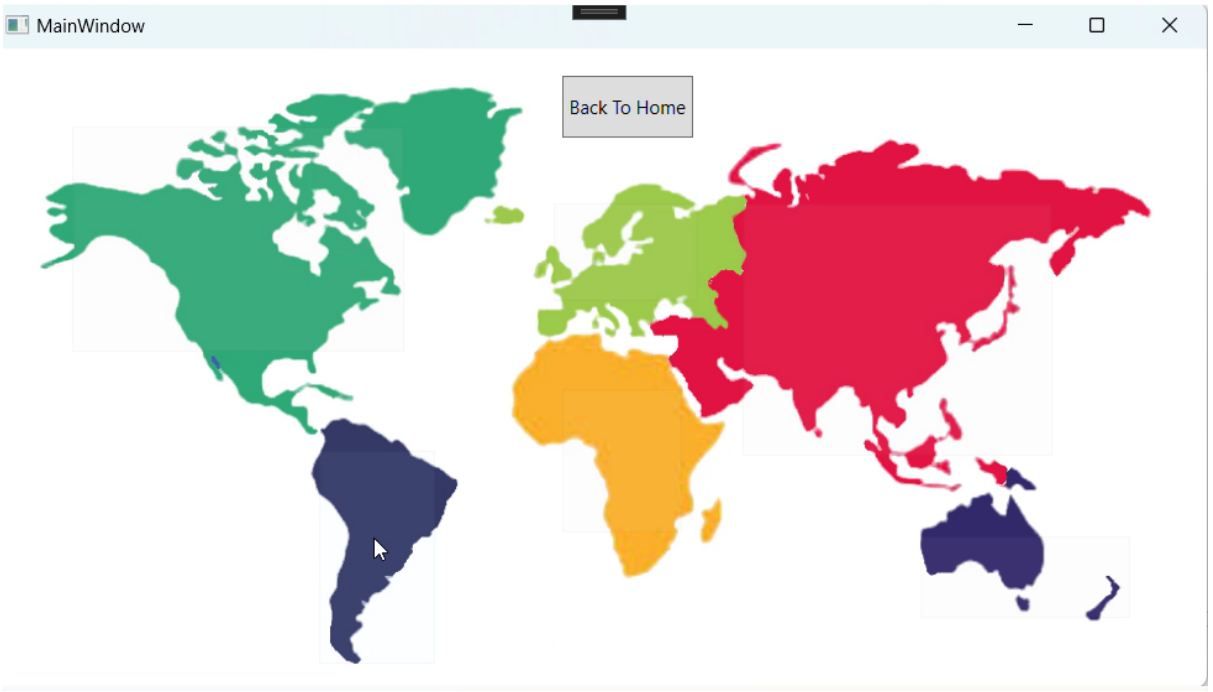
TEST 15:

- MAP PAGE: North America Button Clicked



- Successfully changed to NORTH AMERICA PAGE

TEST 16:

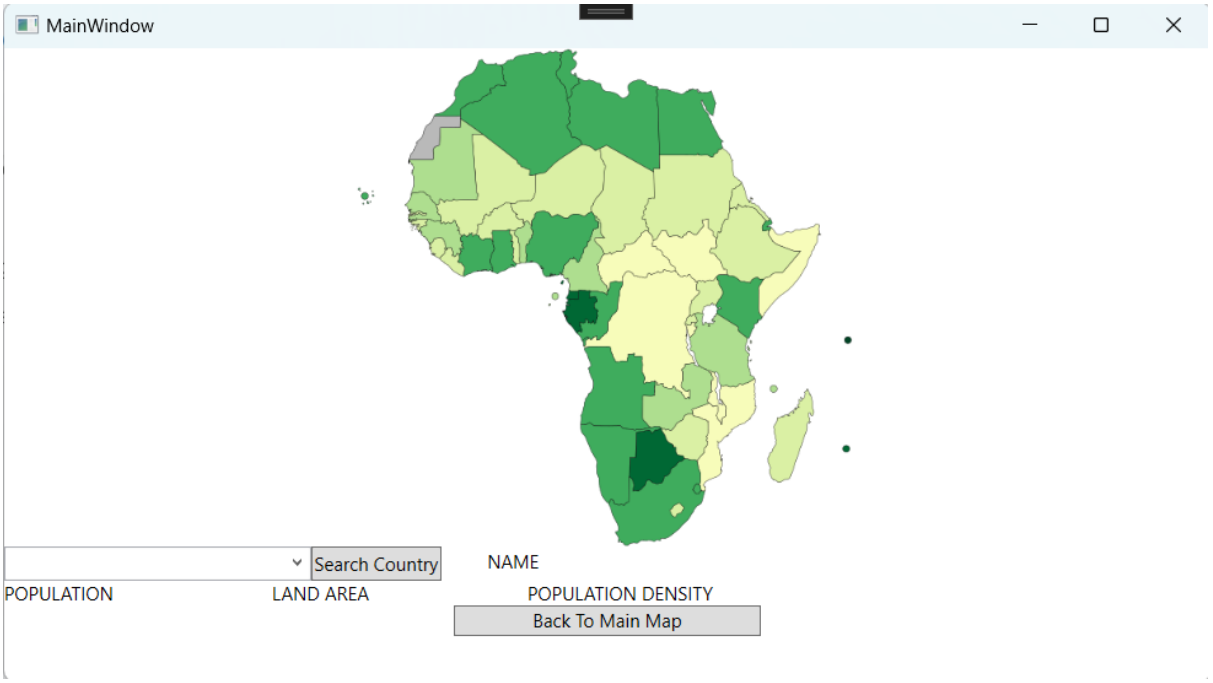


- MAP PAGE: South America Button Clicked

- Successfully changed to SOUTH AMERICA PAGE

TEST 17:

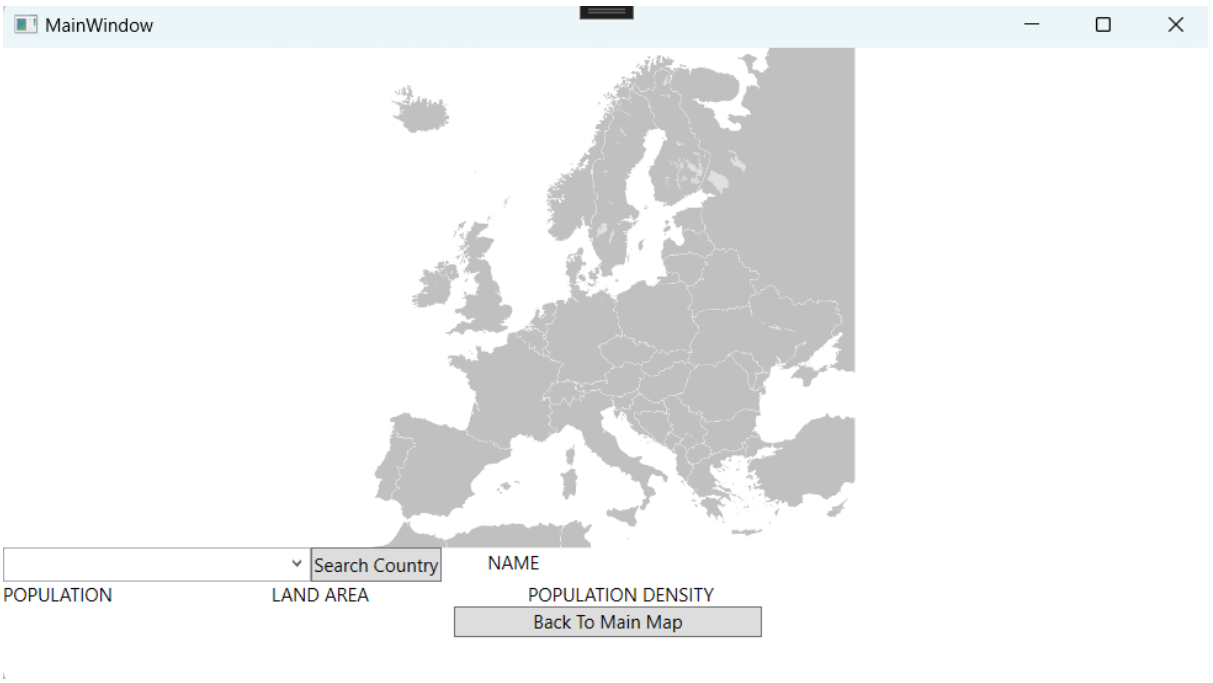
- MAP PAGE: Africa Button Clicked



- Successfully changed to AFRICA PAGE

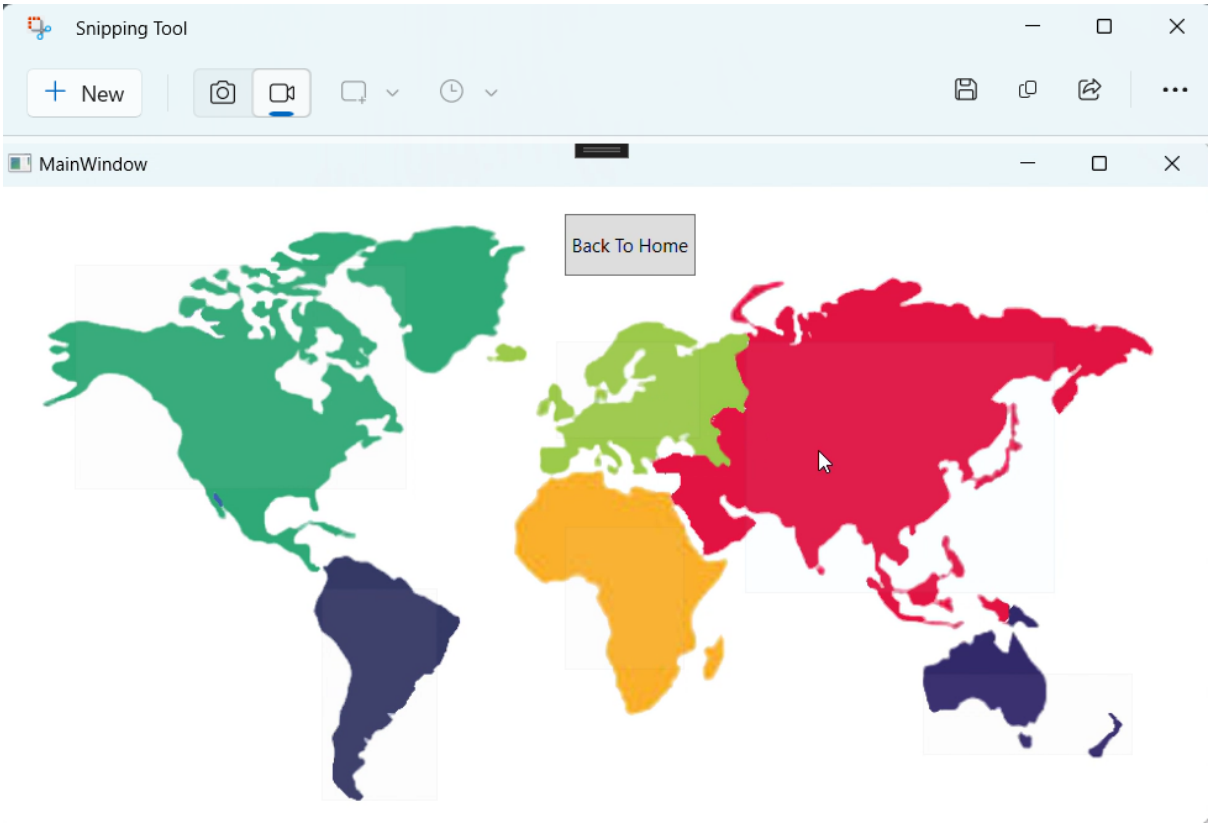
TEST 18:

- MAP PAGE: Europe Button Clicked

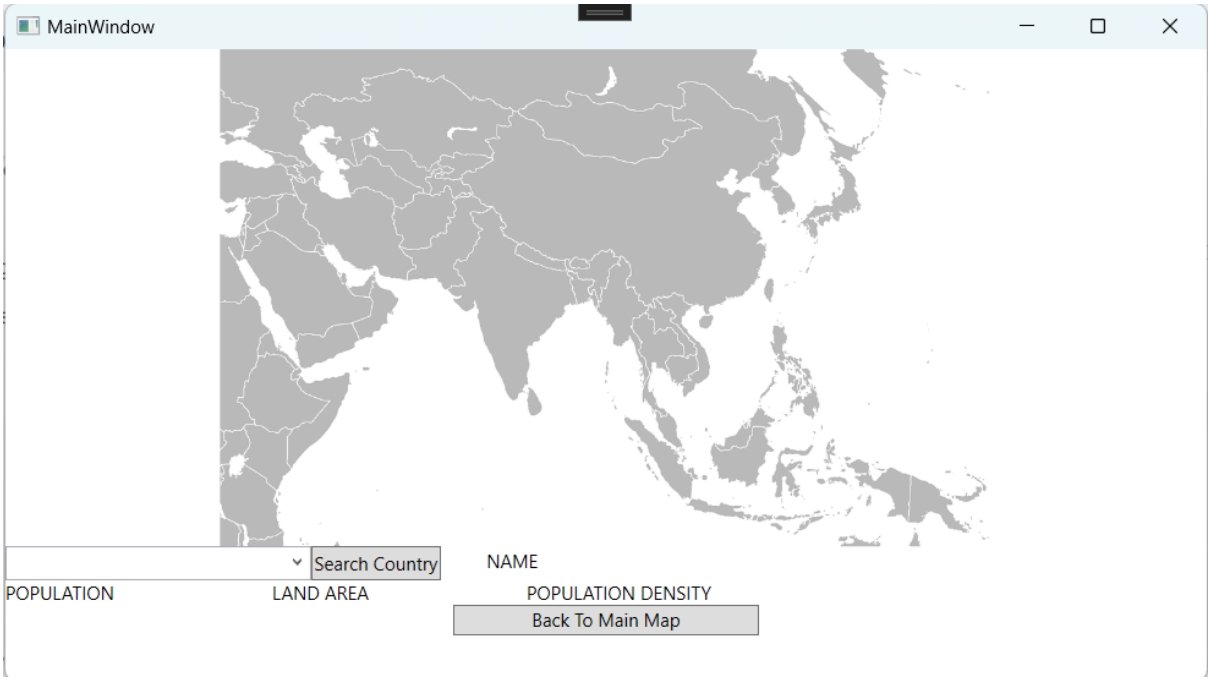


- Successfully changed to EUROPE PAGE

TEST 19:



- MAP PAGE: Asia Button Clicked

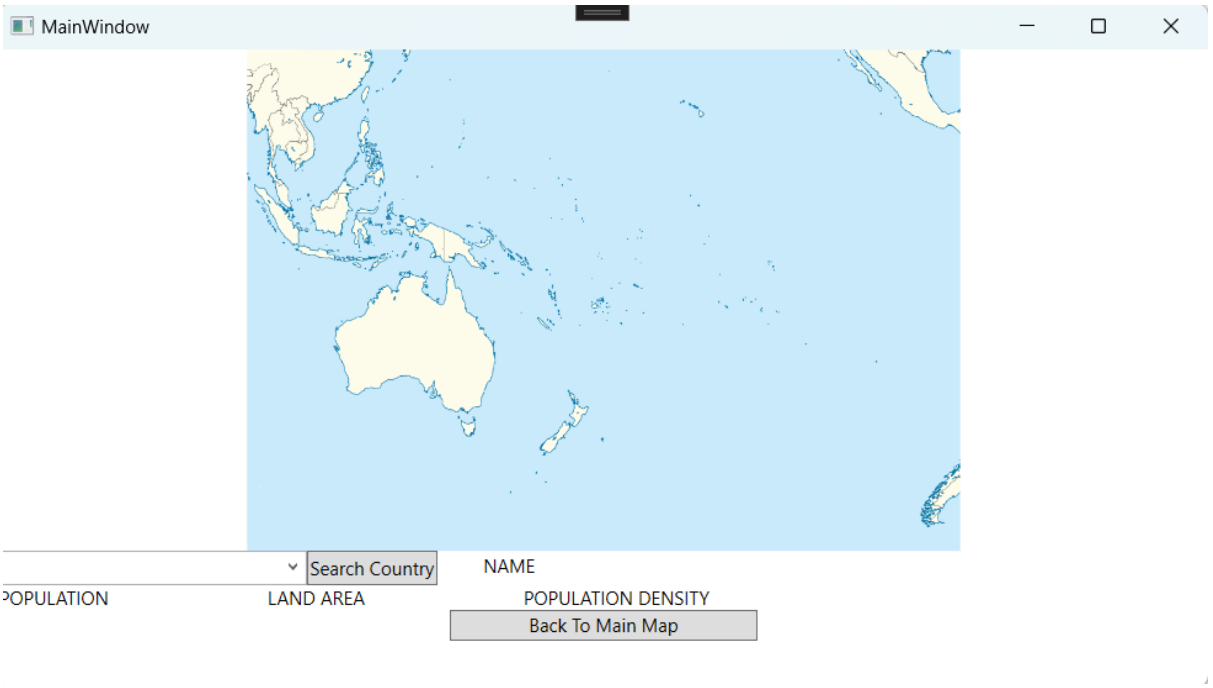


- Successfully changed to ASIA PAGE

TEST 20:

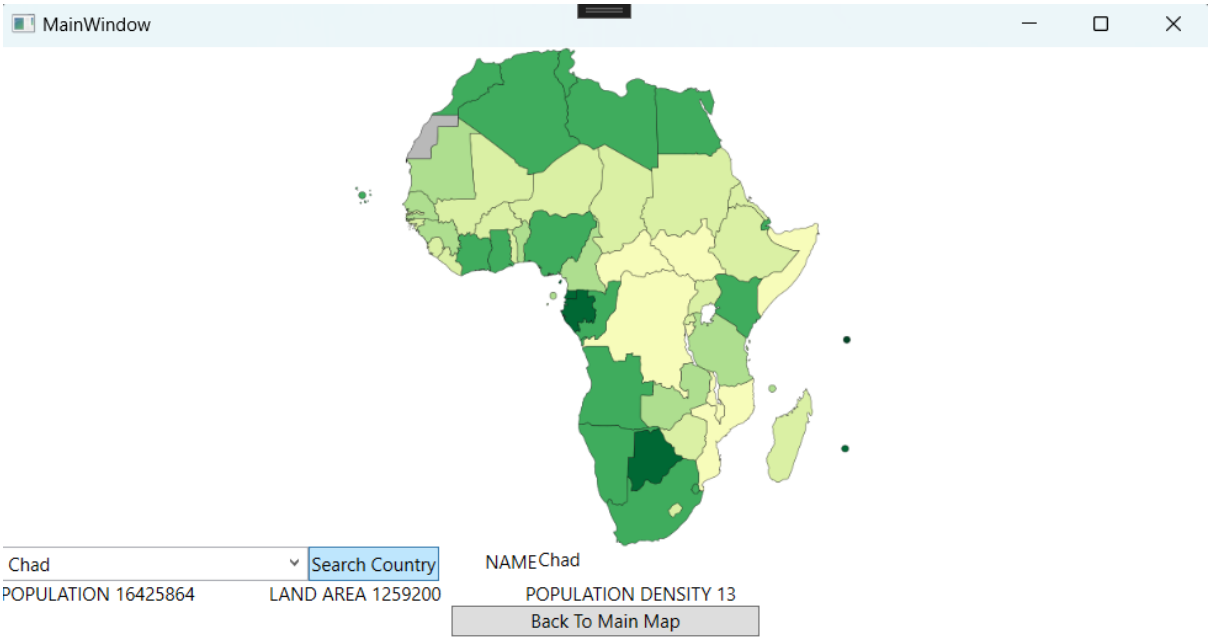
Name: Gabriella Emerson Candidate Number: 7346 Centre Number: 61853

- MAP PAGE: Oceania Button Clicked



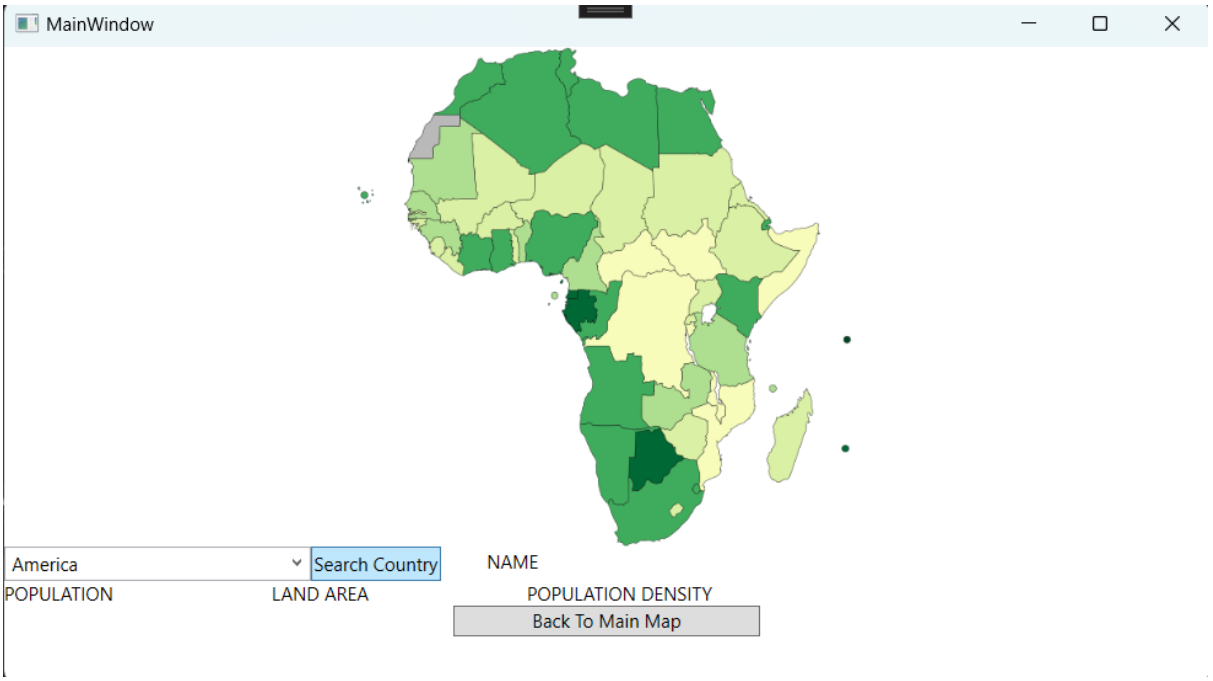
- Successfully changed to OCEANIA PAGE

TEST 21:



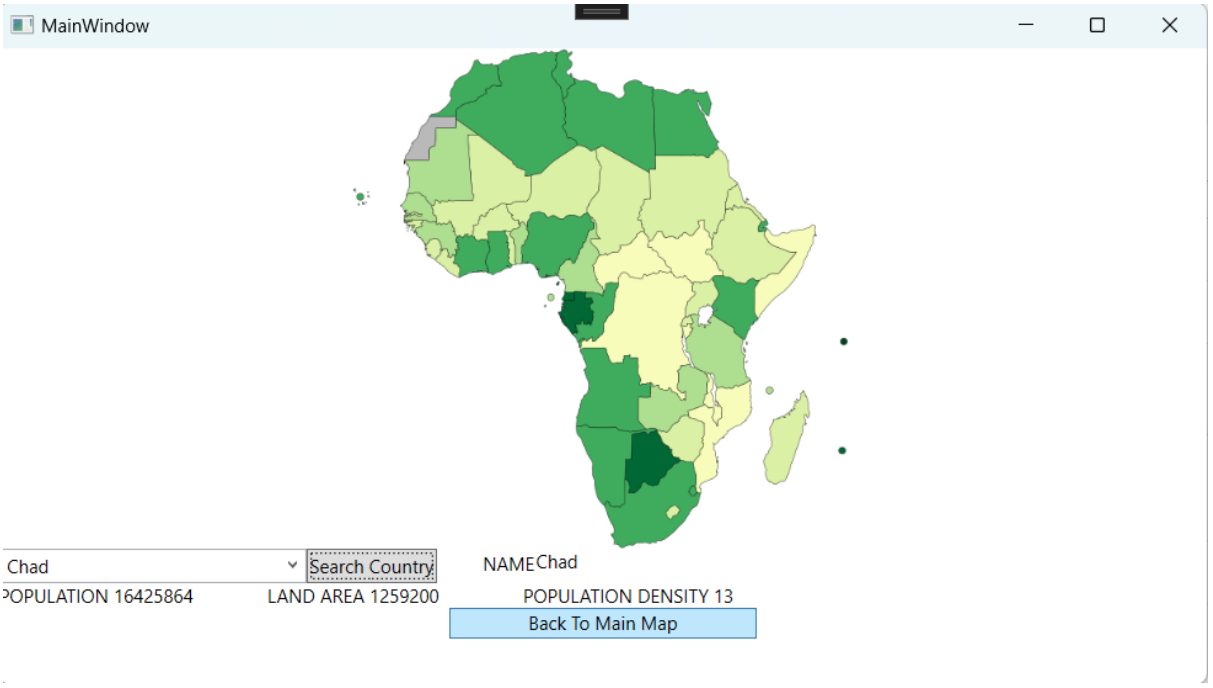
- AFRICA PAGE: Valid Country is selected and the search button is pressed, then the information is displayed to the screen.

TEST 22:

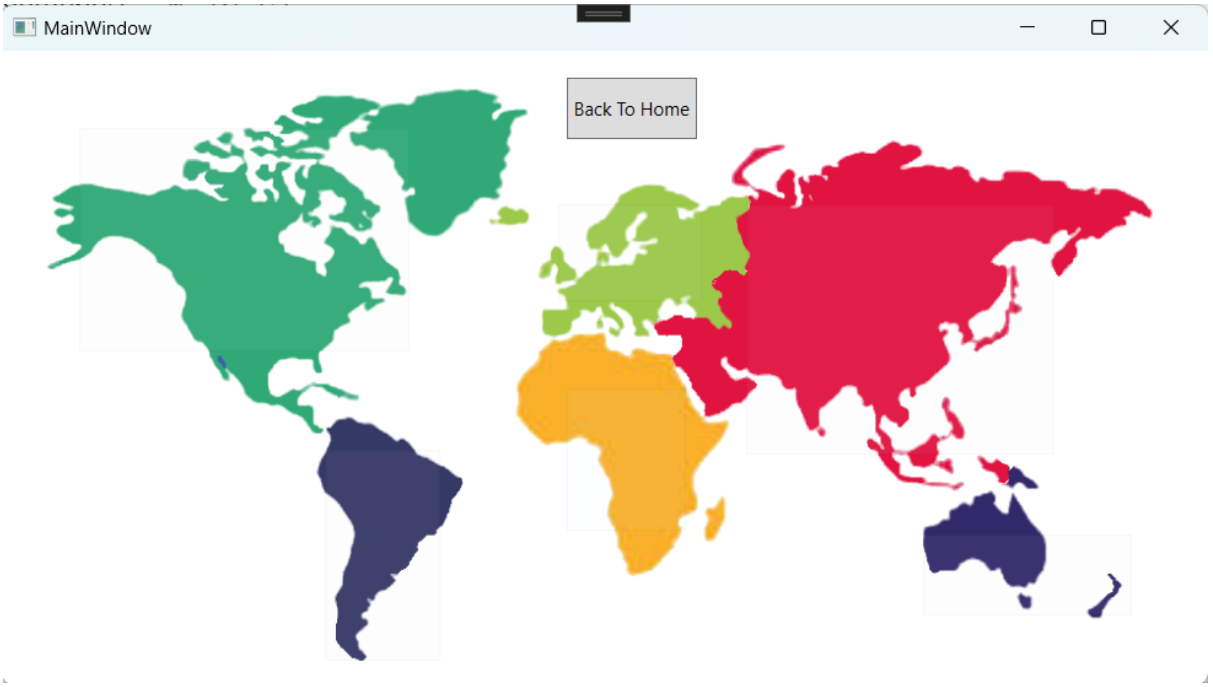


- AFRICA PAGE: An invalid country is entered and the search button is pressed, without the program crashing.

TEST 23:

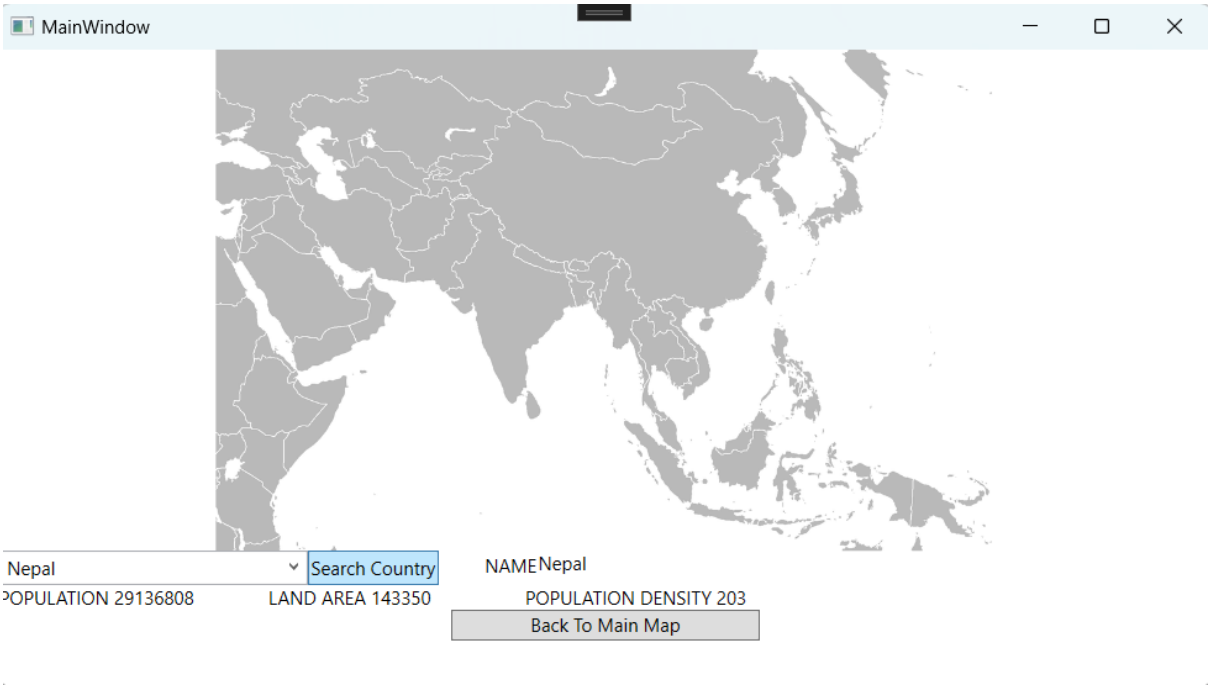


- AFRICA PAGE: Back to Main Map Button Pressed



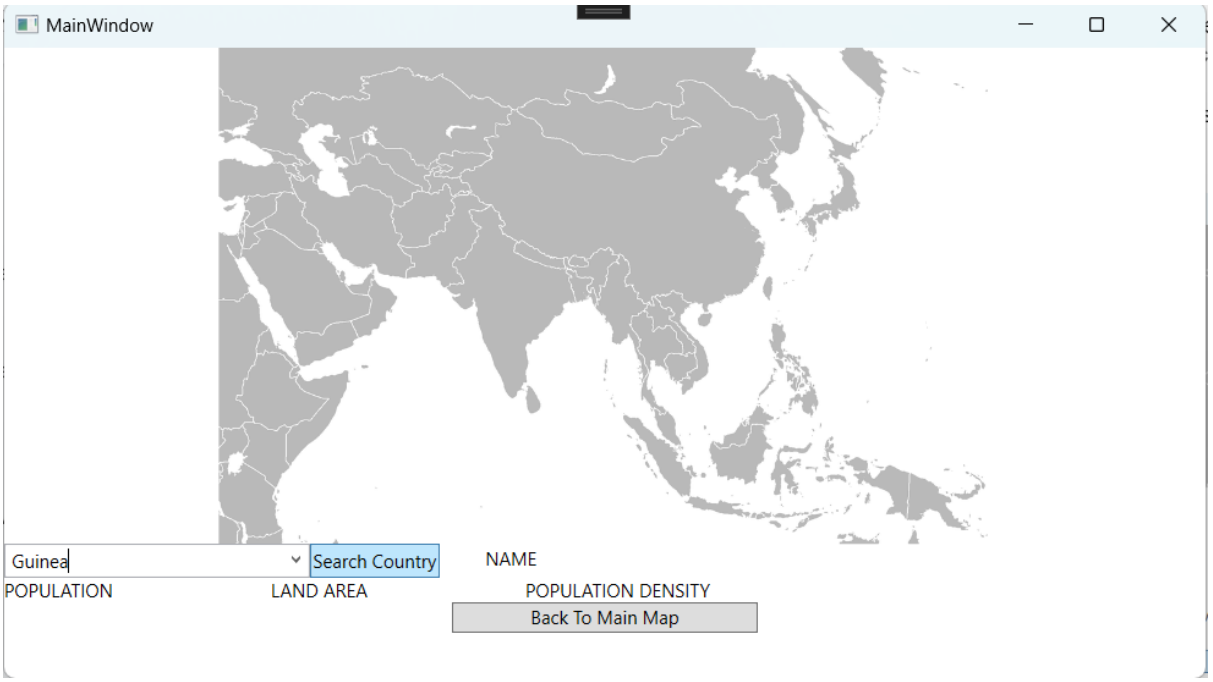
- Changes back to the MAP PAGE

TEST 24:



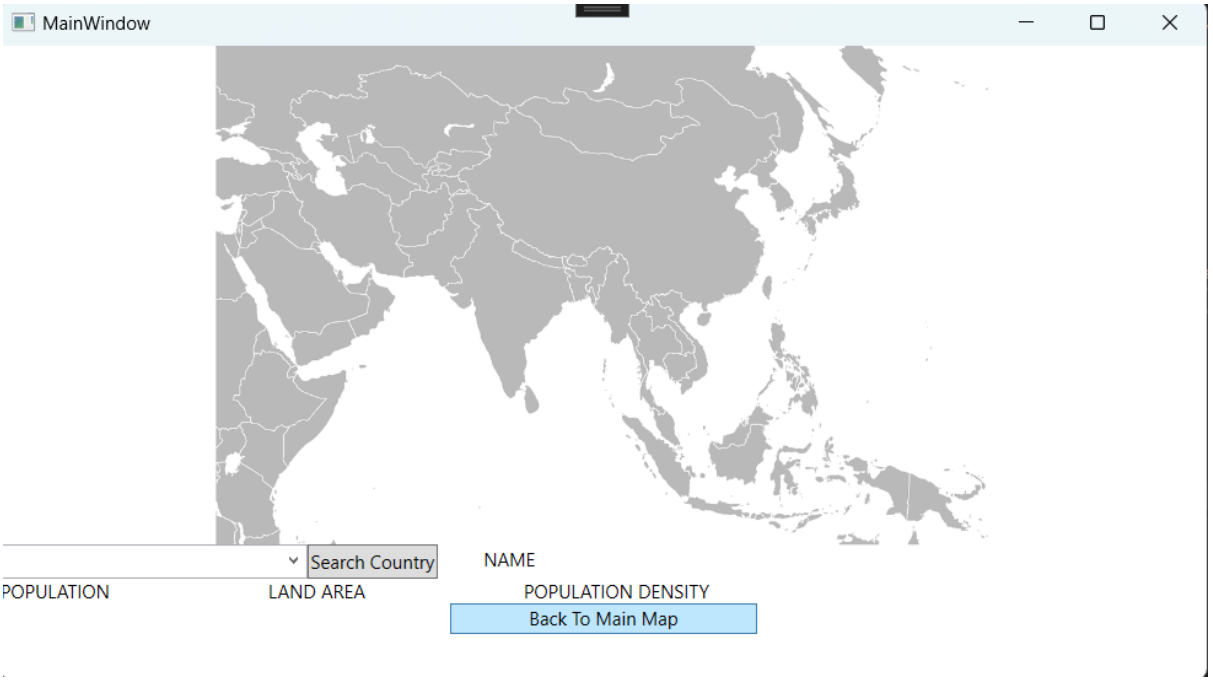
- ASIA PAGE: Valid Country is selected and the search button is pressed, then the information is displayed to the screen.

TEST 25:



- ASIA PAGE: An invalid country is entered and the search button is pressed, without the program crashing.

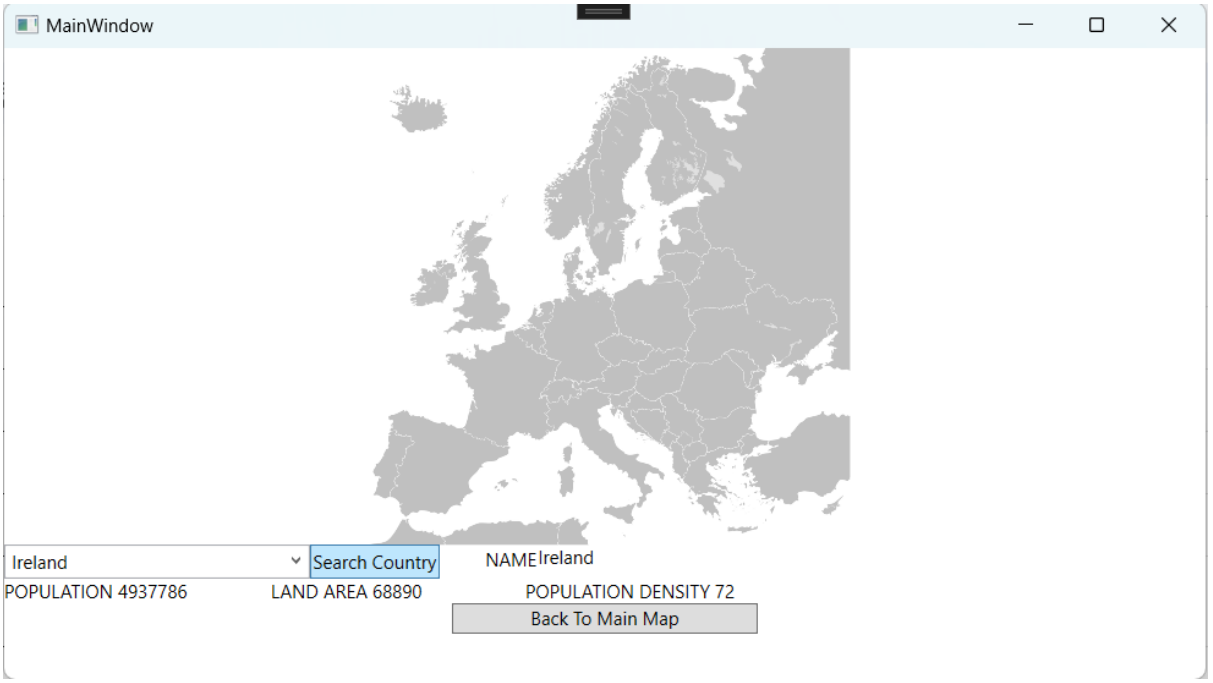
TEST 26:



- ASIA PAGE: Back to Main Map Button Pressed

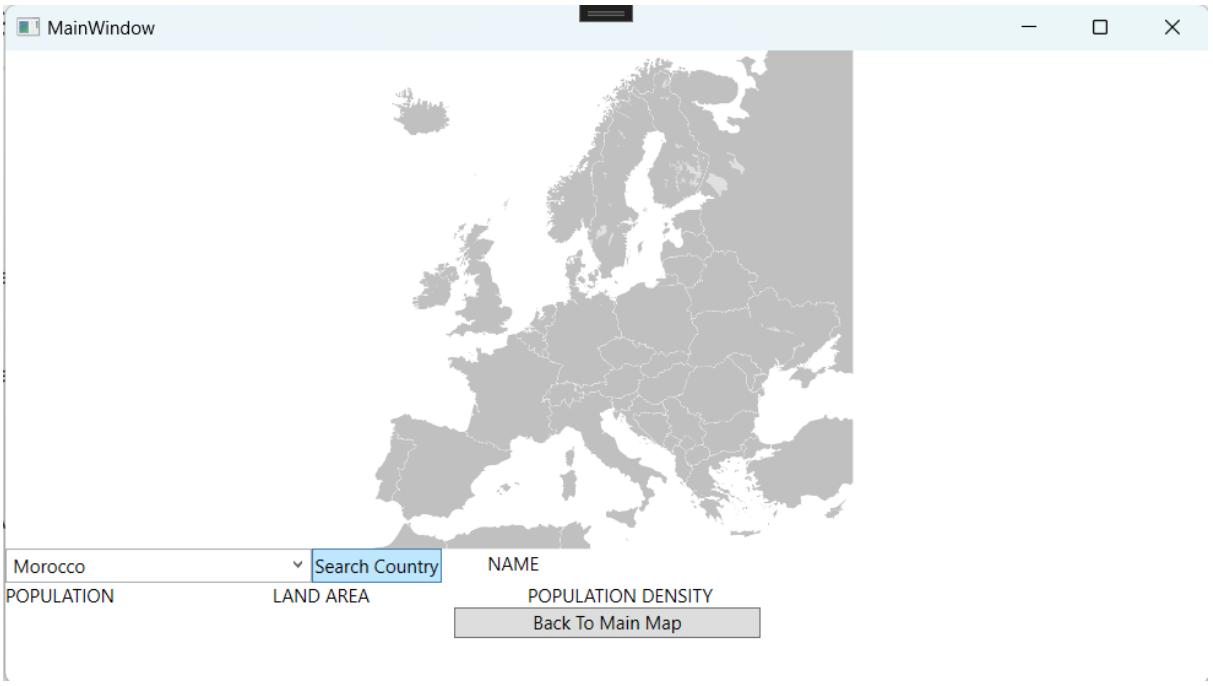
- Changes to the MAP PAGE.

TEST 27:



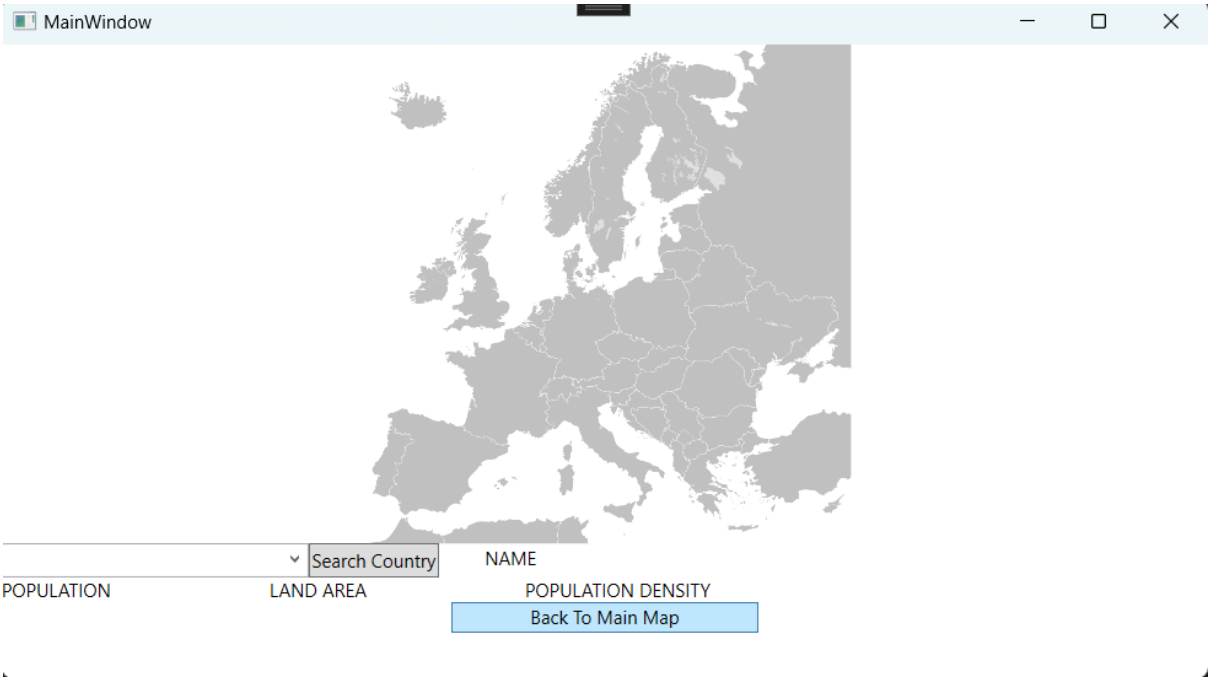
- EUROPE PAGE: Valid Country is selected and the search button is pressed, then the information is displayed to the screen.

TEST 28:



- EUROPE PAGE: An invalid country is entered and the search button is pressed, without the program crashing.

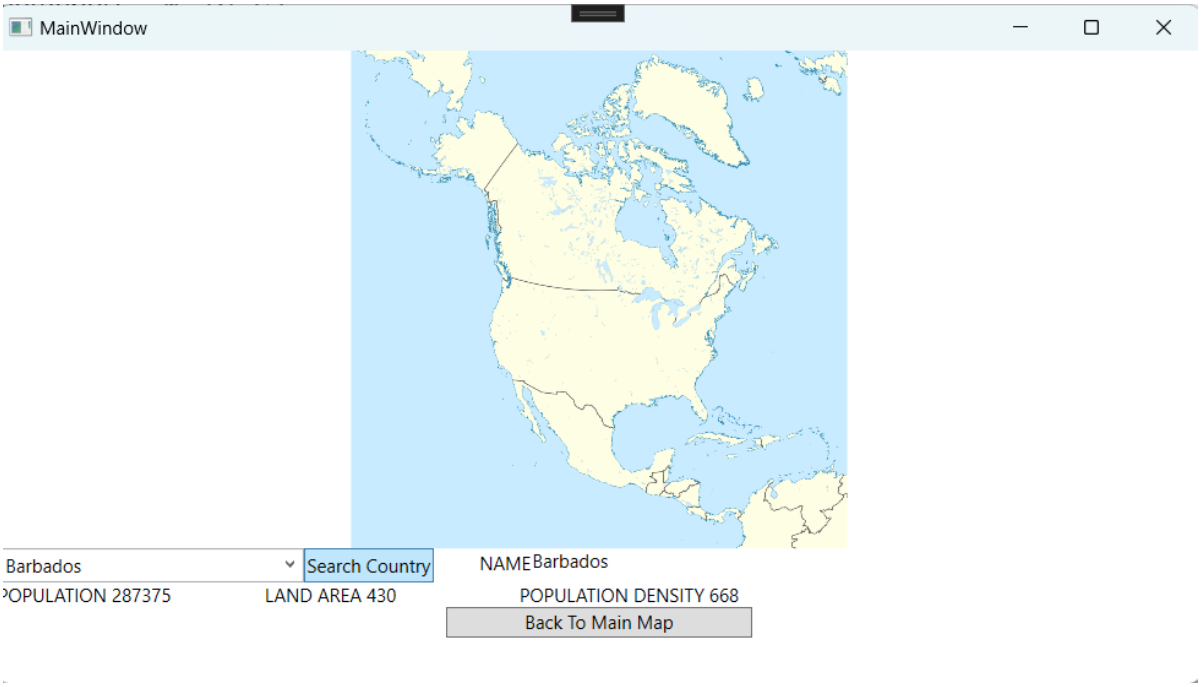
TEST 29:



- EUROPE PAGE: Back To Main Map Button Pressed

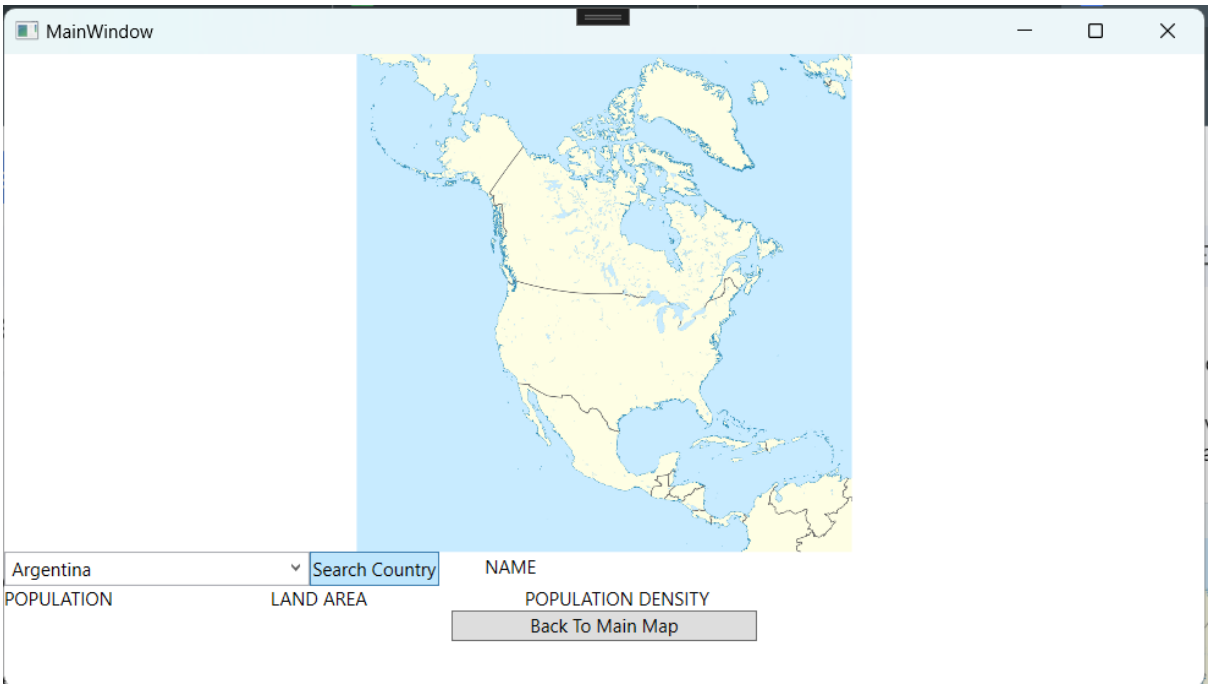
- Changes to the MAP PAGE

TEST 30:



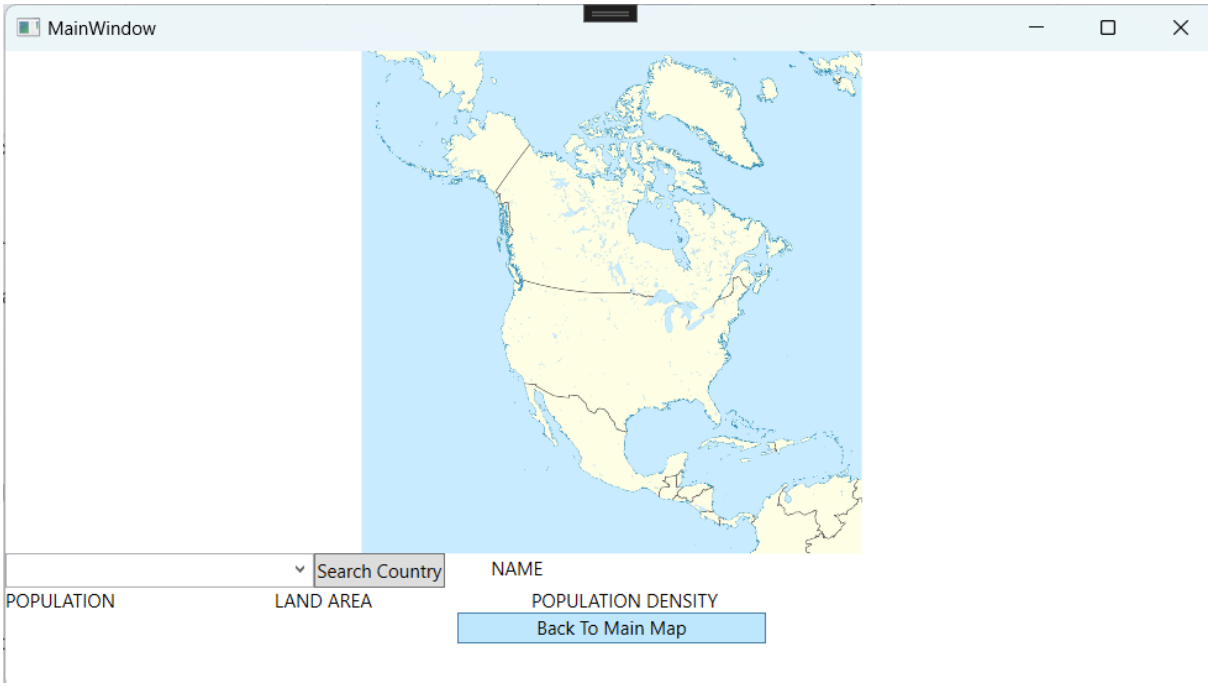
- NORTH AMERICA PAGE: Valid Country is selected and the search button is pressed, then the information is displayed to the screen.

TEST 31:



- NORTH AMERICA PAGE: An invalid country is entered and the search button is pressed, without the program crashing.

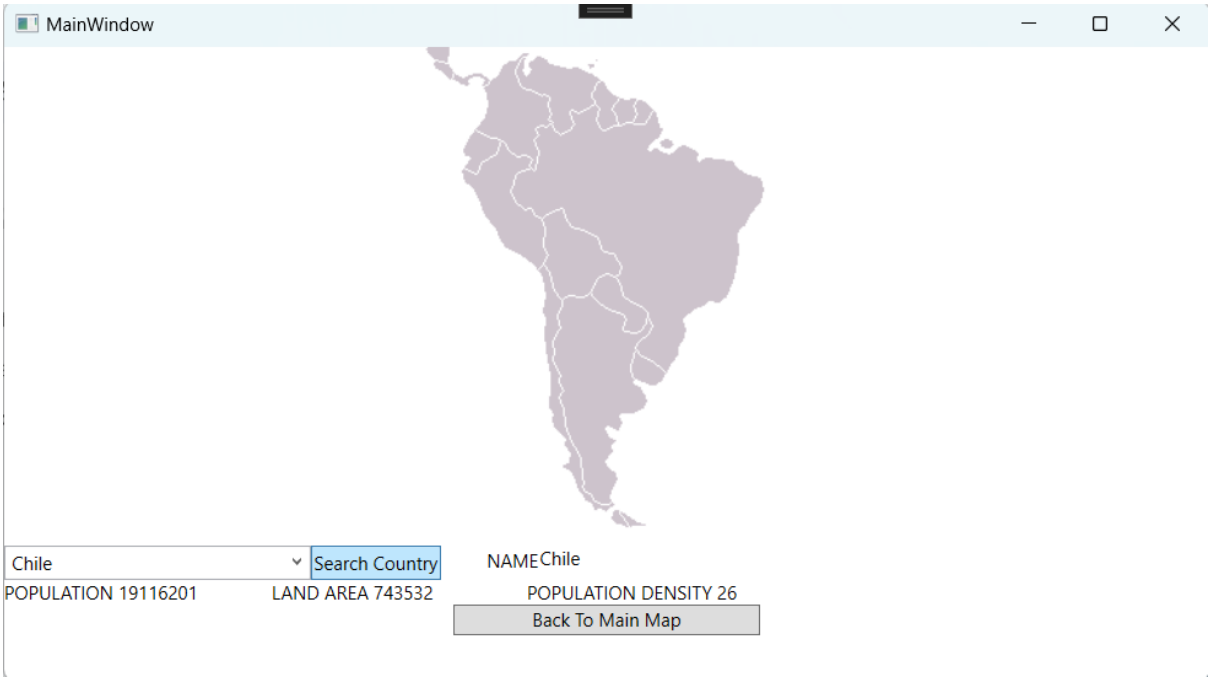
TEST 32:



- NORTH AMERICA PAGE: Back To Main Map Button Pressed

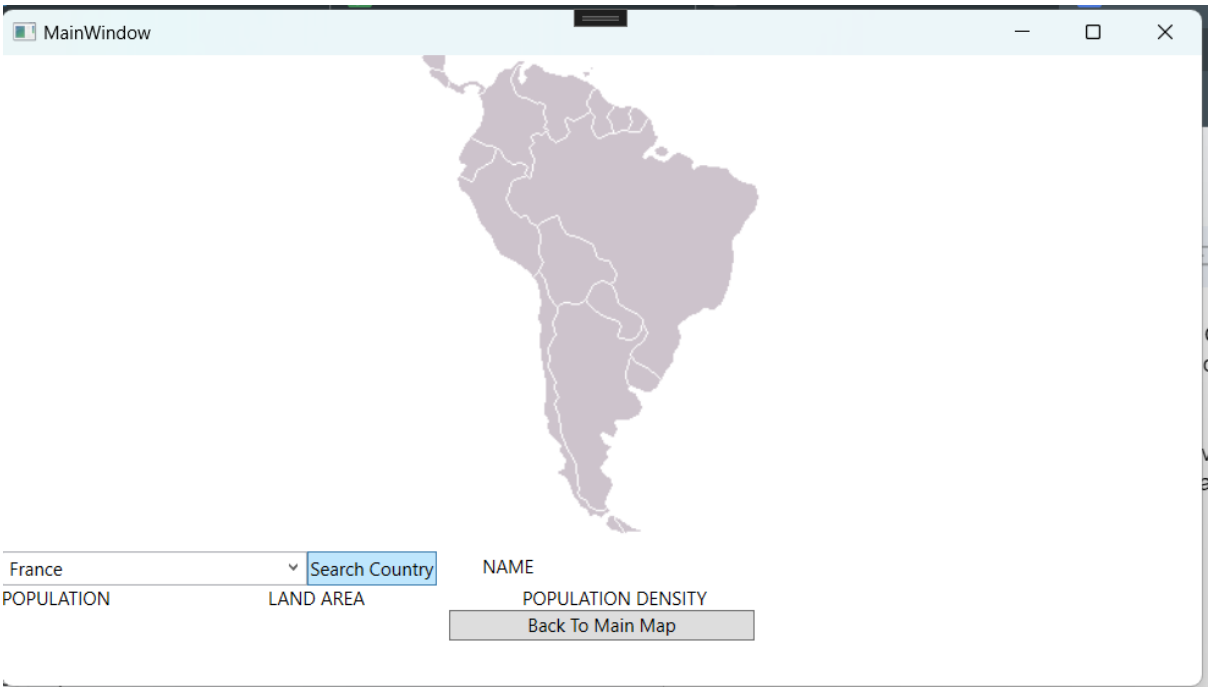
- Changes to the MAP PAGE

TEST 33:



- SOUTH AMERICA PAGE: Valid Country is selected and the search button is pressed, then the information is displayed to the screen.

TEST 34:



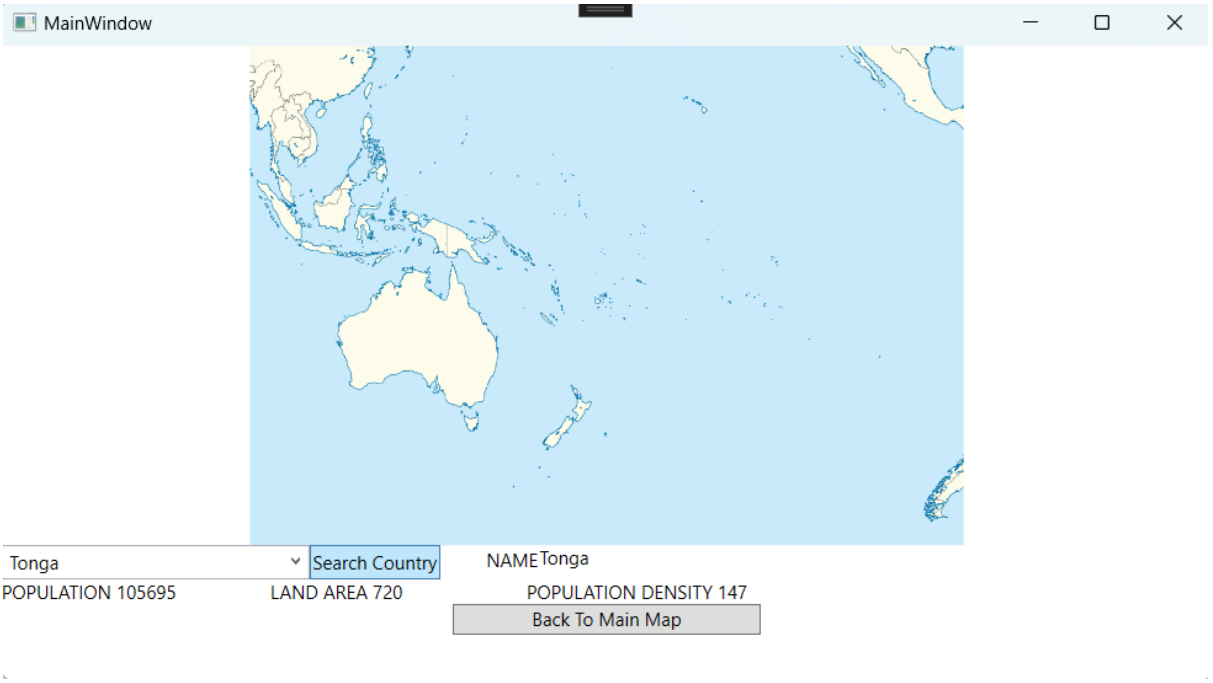
- SOUTH AMERICA PAGE: An invalid country is entered and the search button is pressed, without the program crashing.

TEST 35:

- SOUTH AMERICA PAGE: Back To Main Map Button Pressed

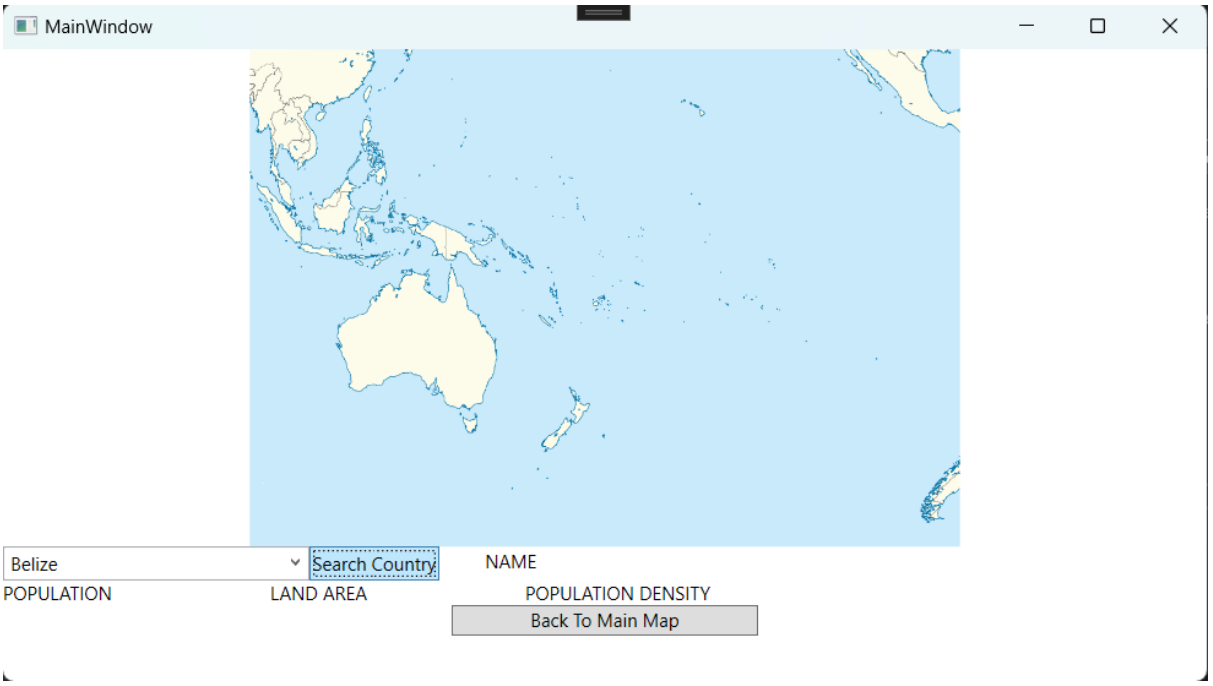
- Changes to the MAP PAGE

TEST 36:



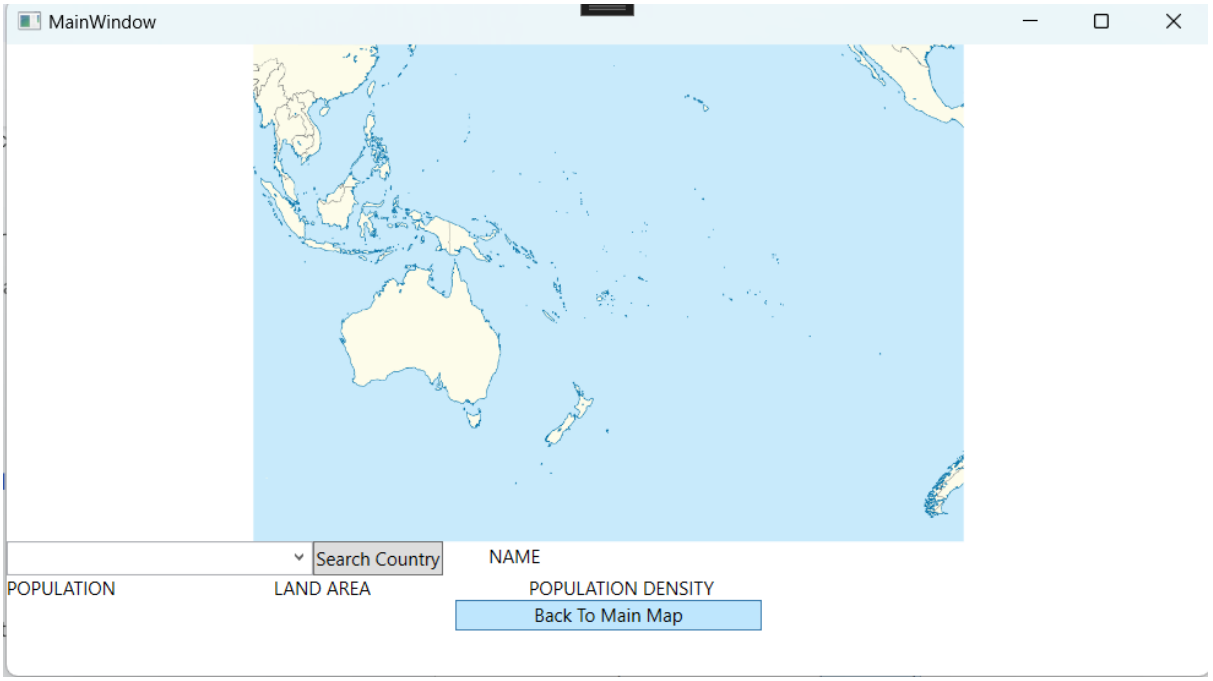
- OCEANIA PAGE: Valid Country is selected and the search button is pressed, then the information is displayed to the screen.

TEST 37:



- OCEANIA PAGE: An invalid country is entered and the search button is pressed, without the program crashing.

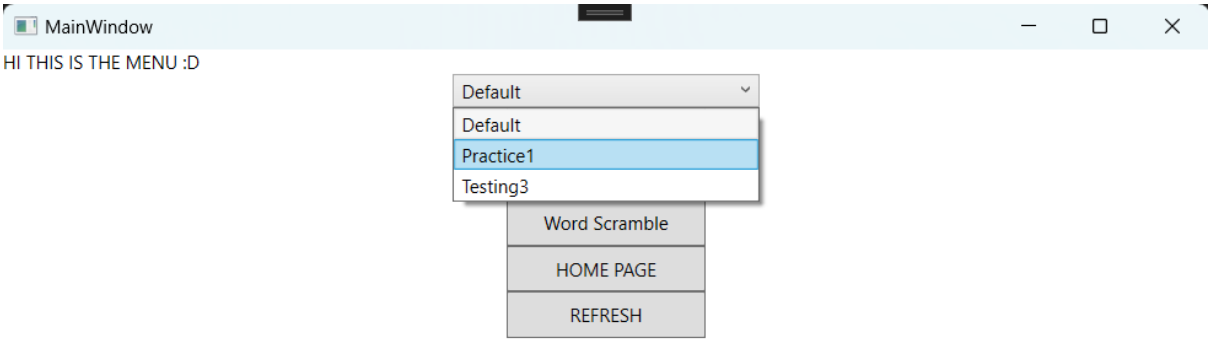
TEST 38:



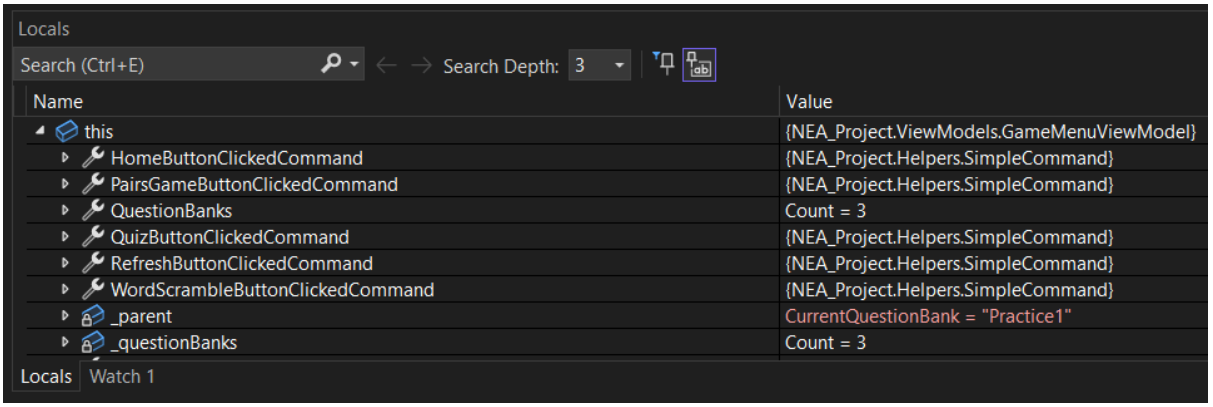
- OCEANIA PAGE: Back To Main Map Button Pressed

- Changes to the MAP PAGE

TEST 39:



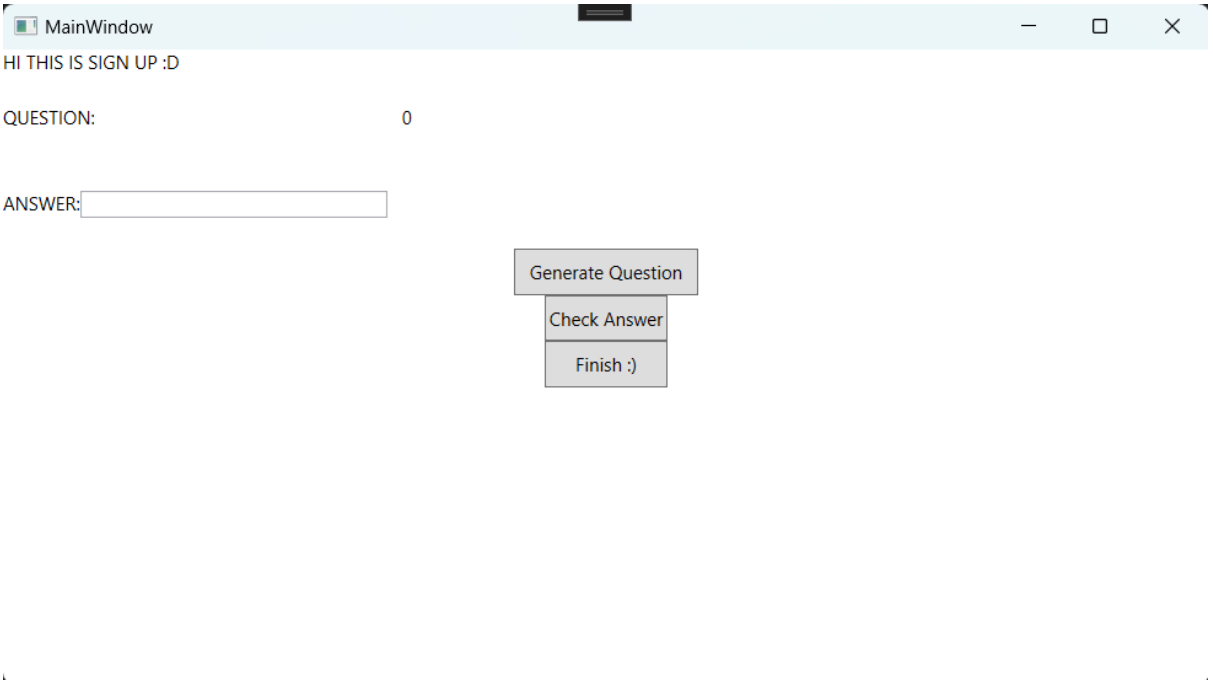
- GAMES MENU: Combobox is dropped down and Practice1 questions are selected.



- Successfully updates the Current question bank variable.

TEST 40:

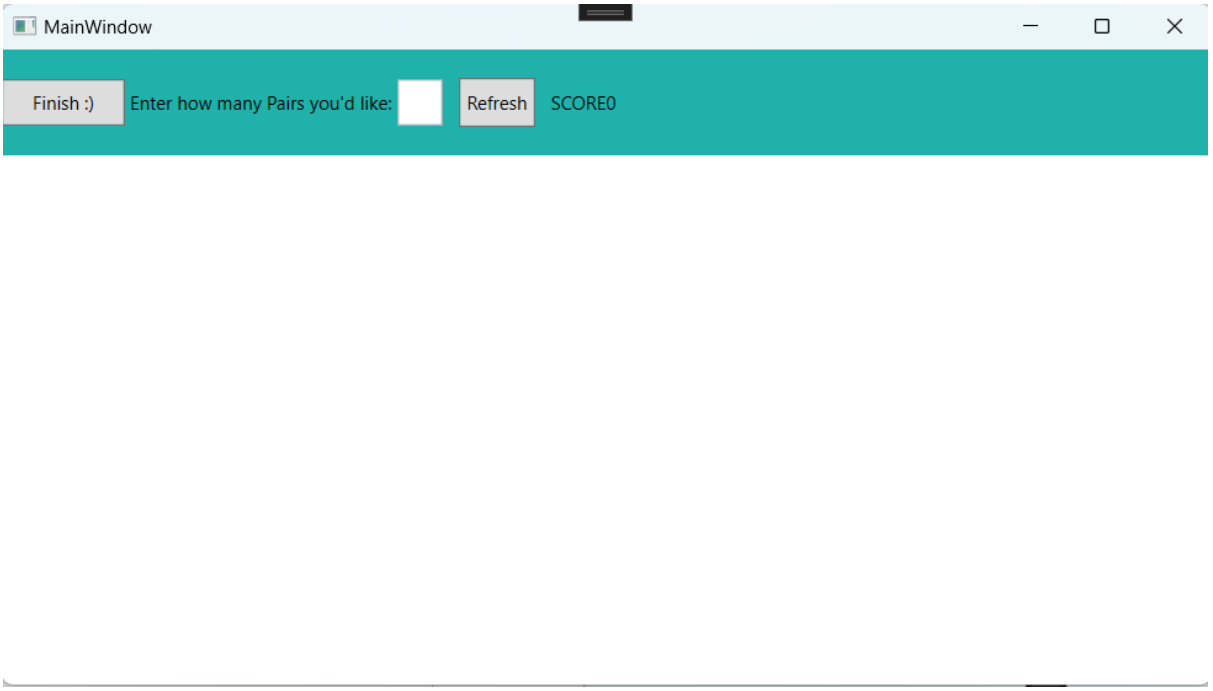
- GAME MENU PAGE: Quiz Button Pressed



- Changes to the QUIZ PAGE

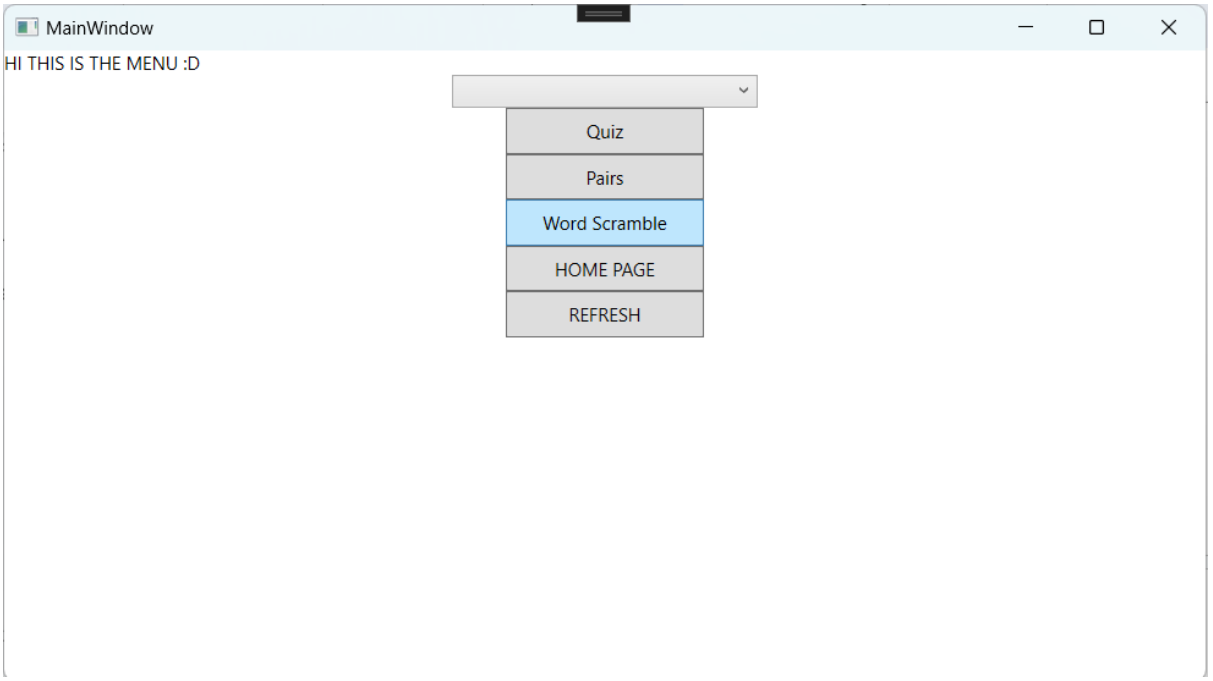
TEST 41:

- GAME MENU PAGE: Pairs Button is pressed

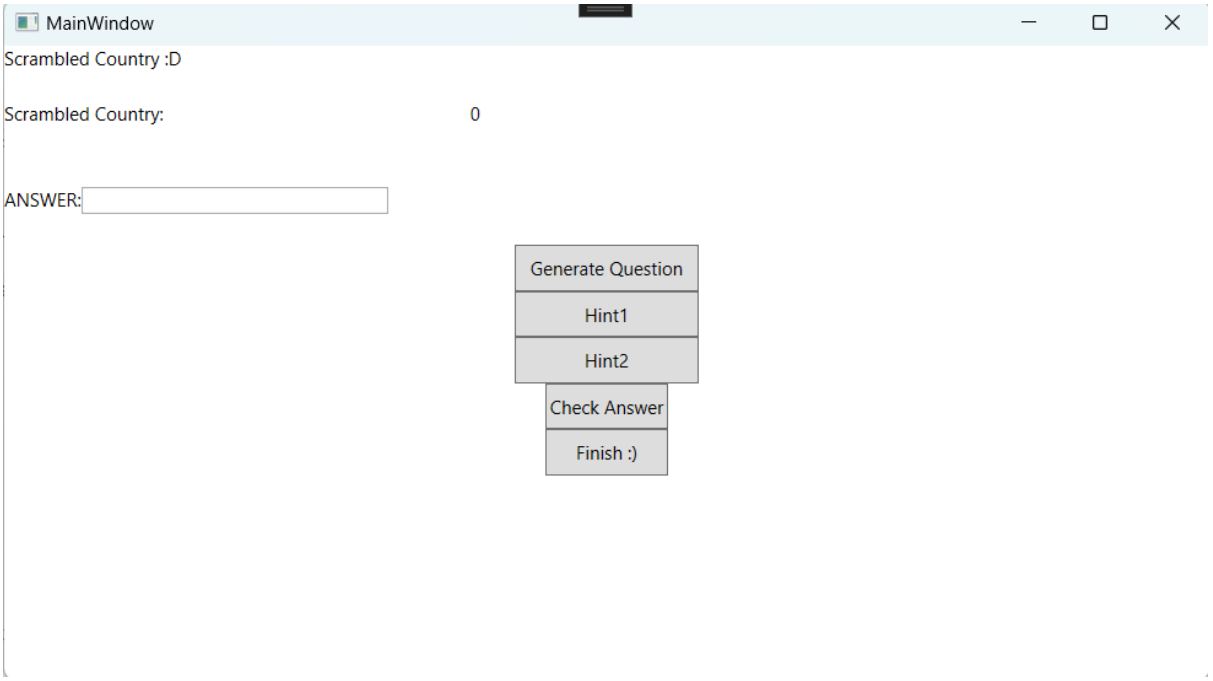


- Changes to PAIRS GAME PAGE

TEST 42:

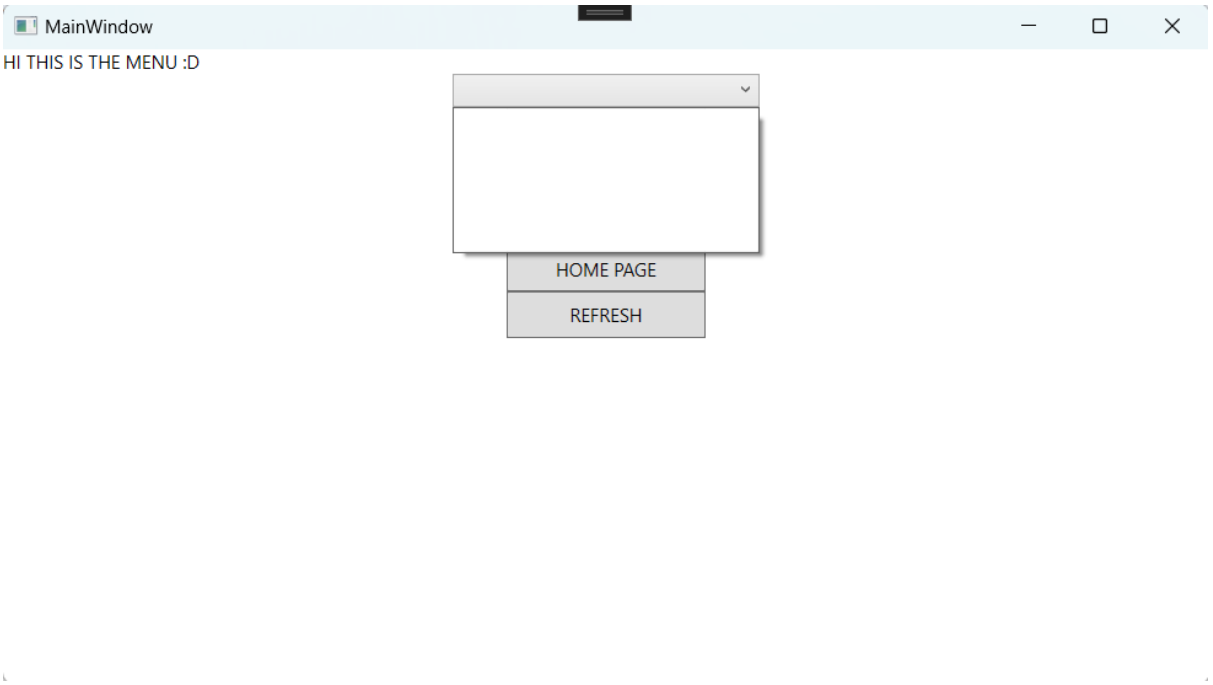


- GAME MENU PAGE: Word Scramble Button is pressed

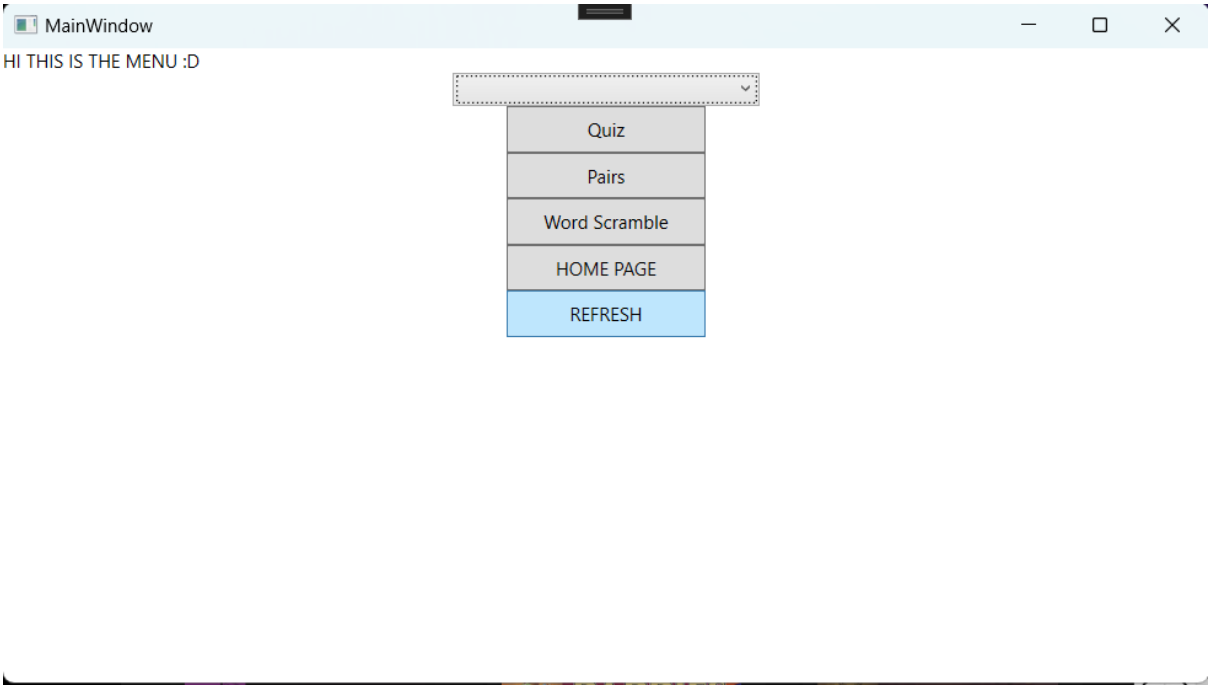


- Changes to WORD SCRAMBLE PAGE

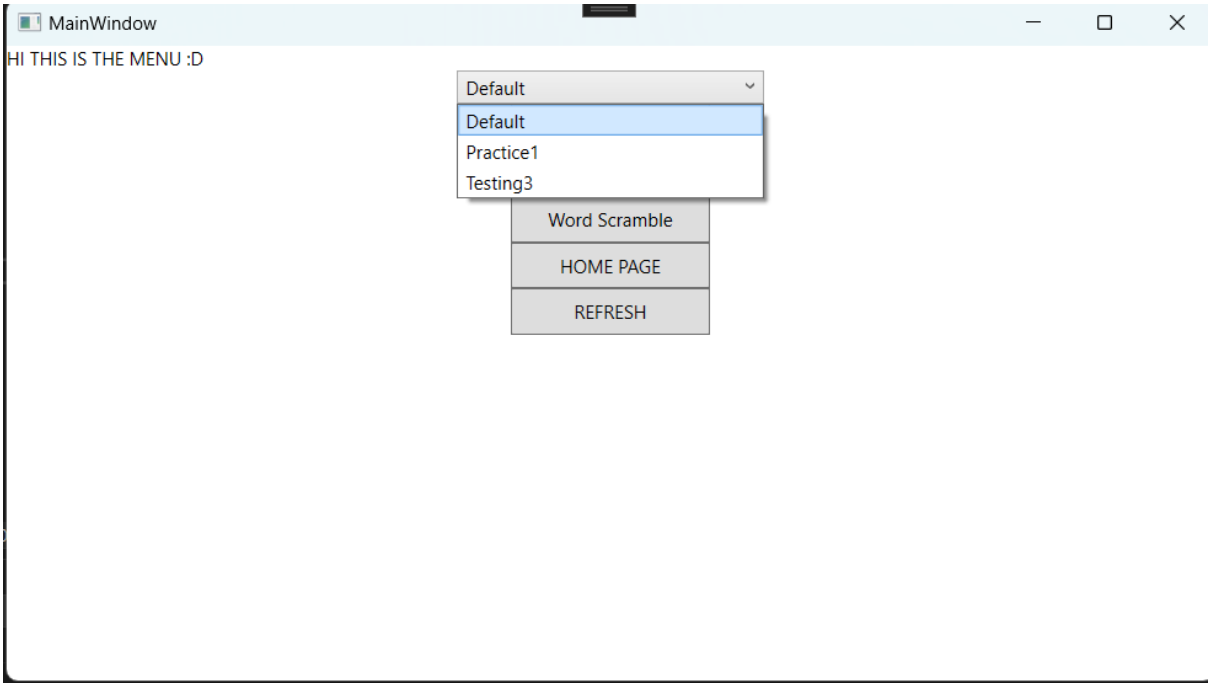
TEST 43:



- GAME MENU PAGE: ComboBox is empty

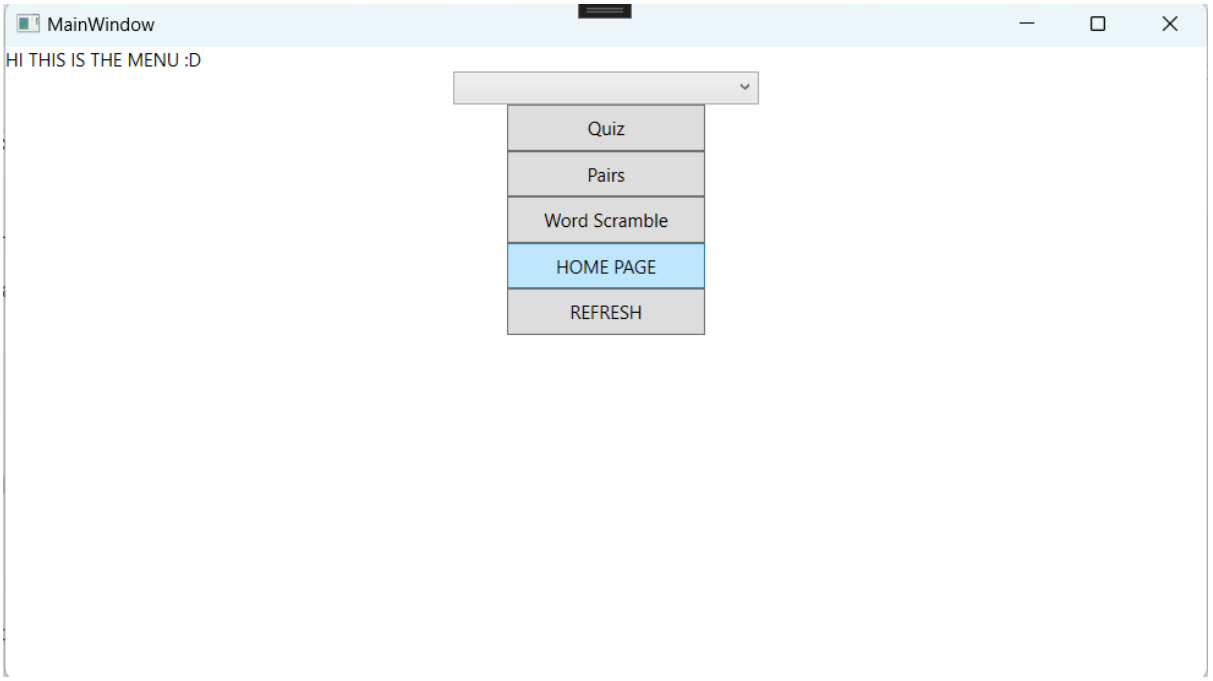


- GAME MENU PAGE: Refresh Button Pressed

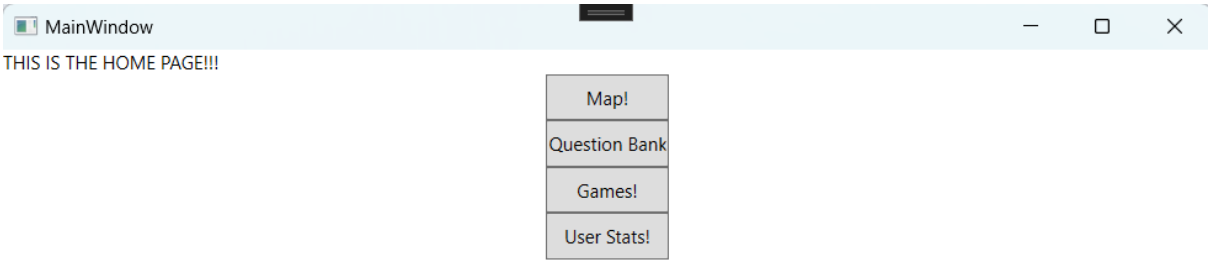


- The ComboBox is updated

TEST 44:

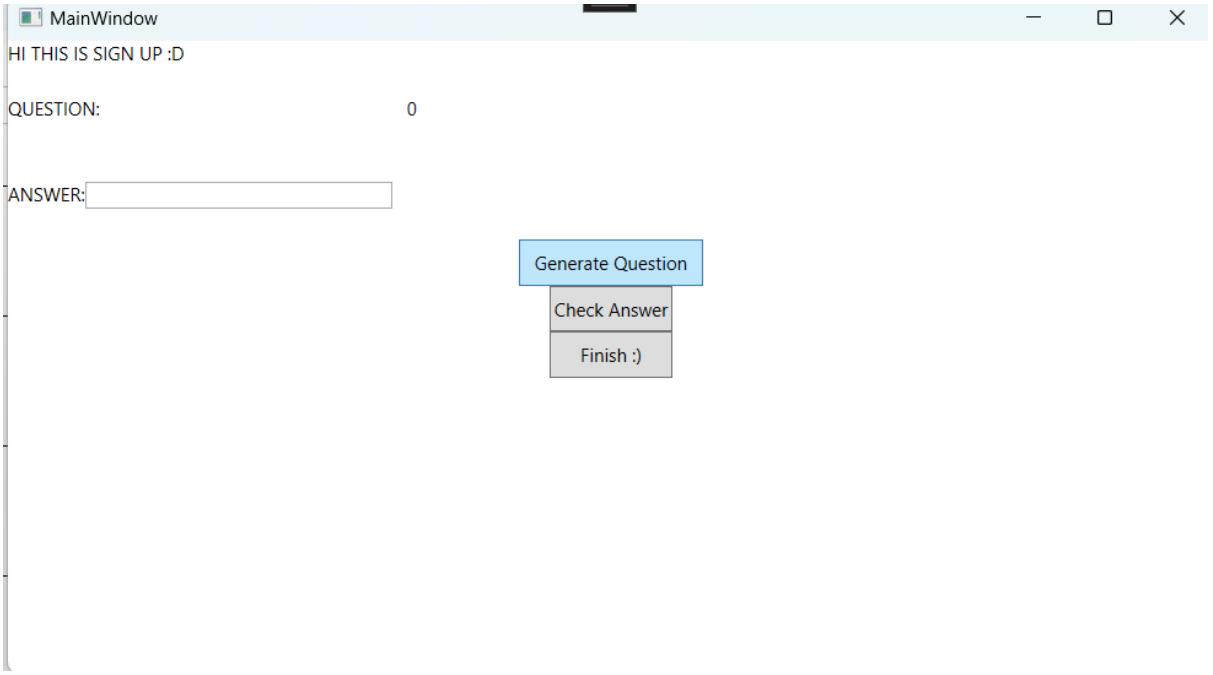


- GAME MENU PAGE: Home Button is pressed

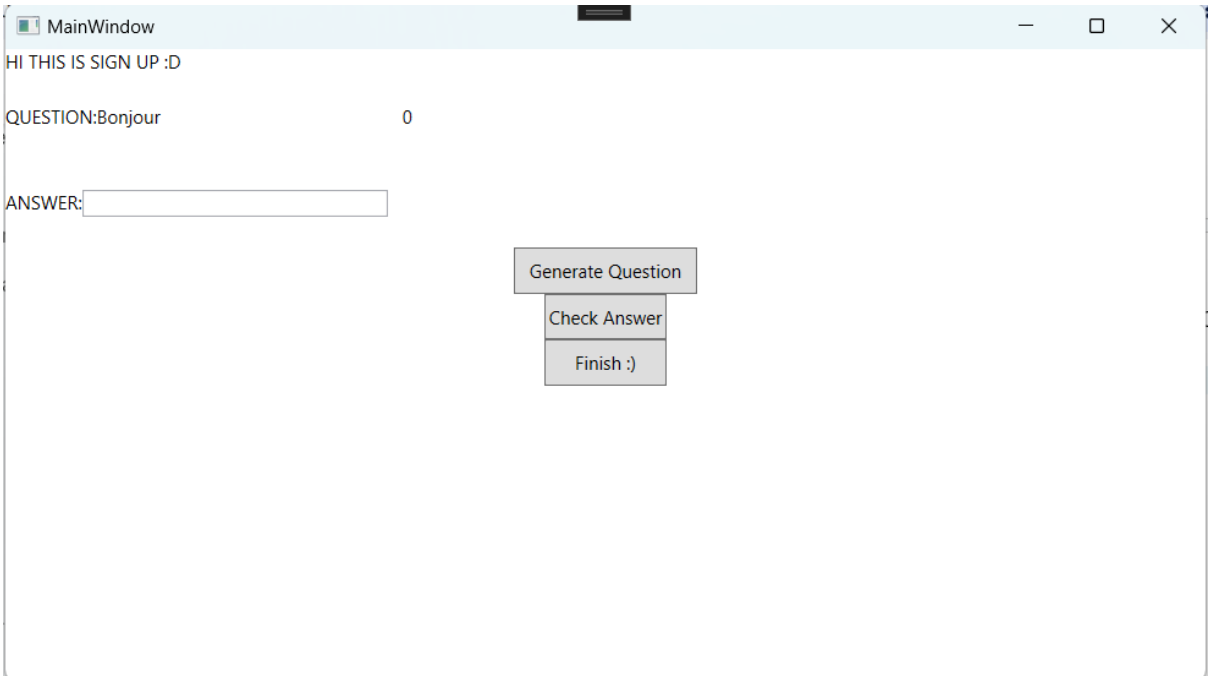


- Changes to HOME PAGE

TEST 45:

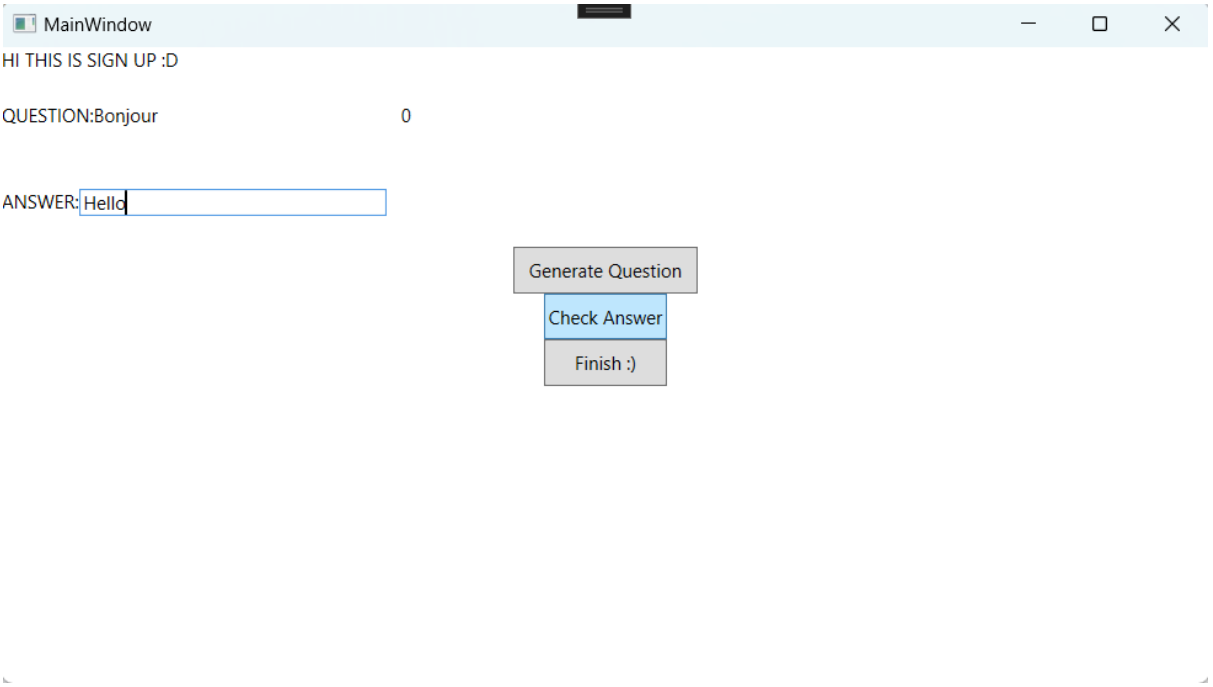


- QUIZ PAGE: Generate Question Button is pressed

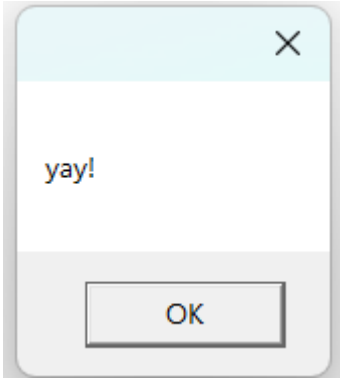


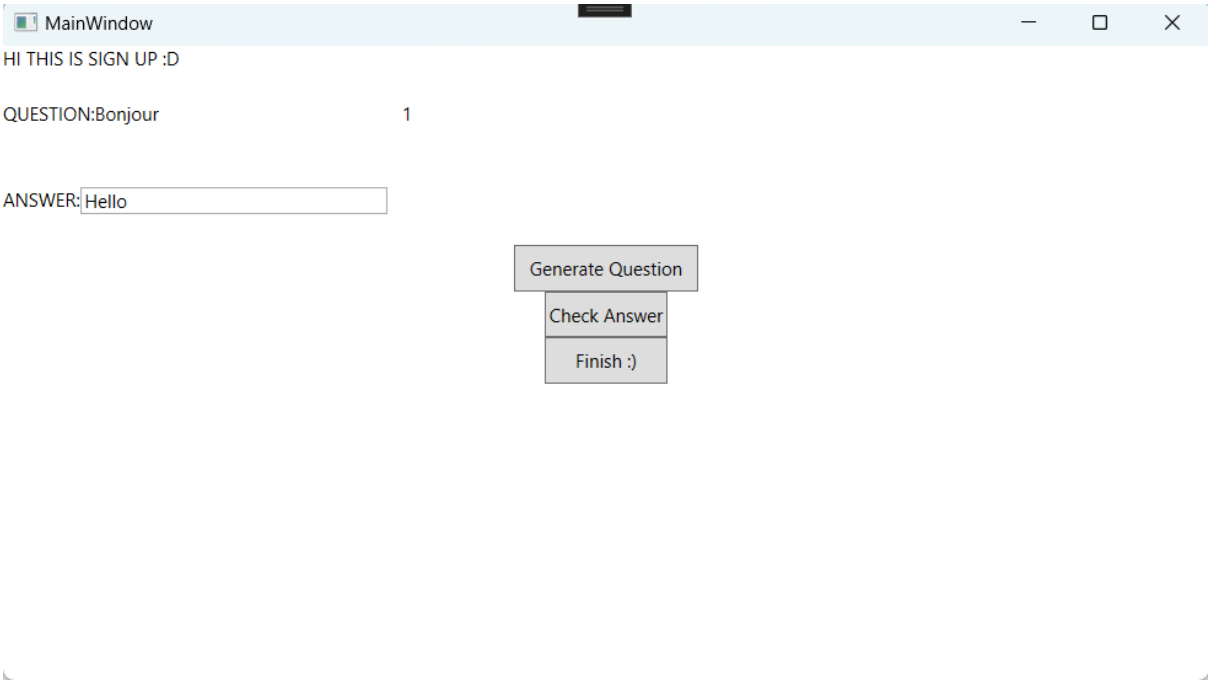
- A random question is generated.

TEST 46:



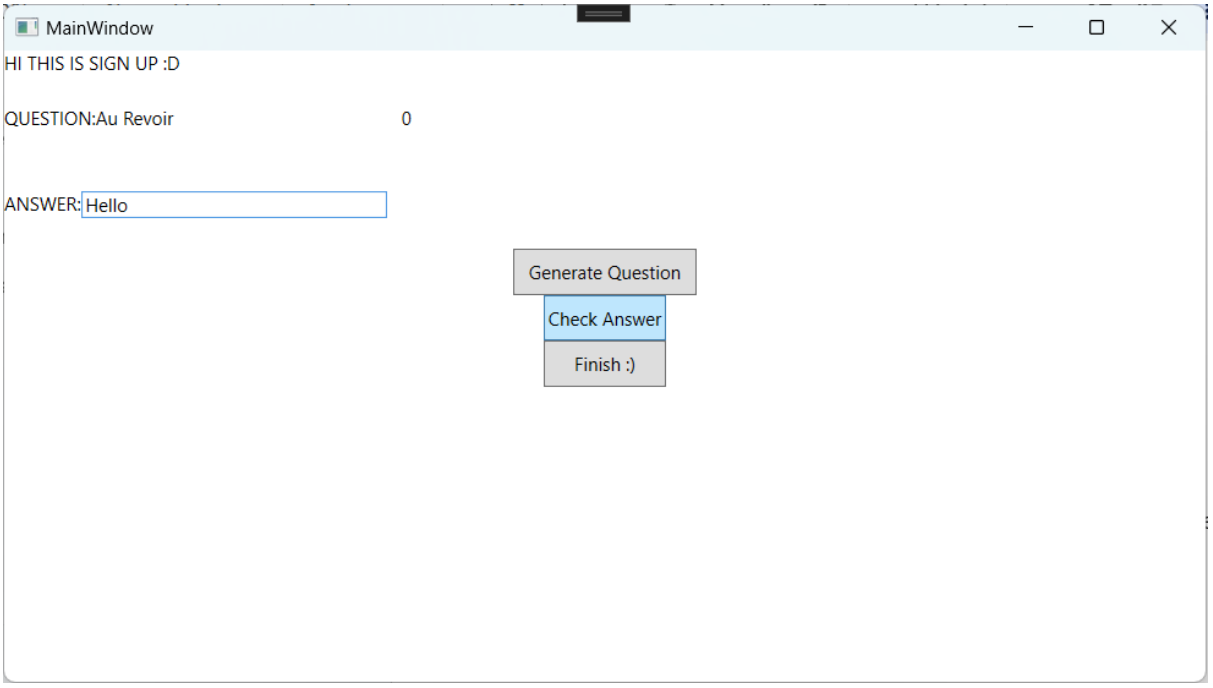
- QUIZ PAGE: Correct answer is entered and Check answer button is pressed.



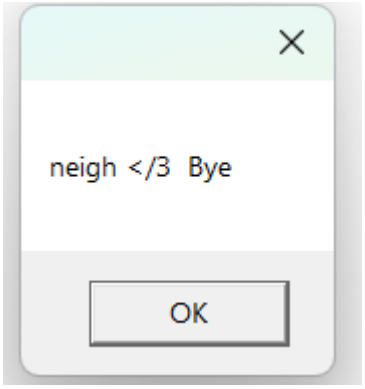


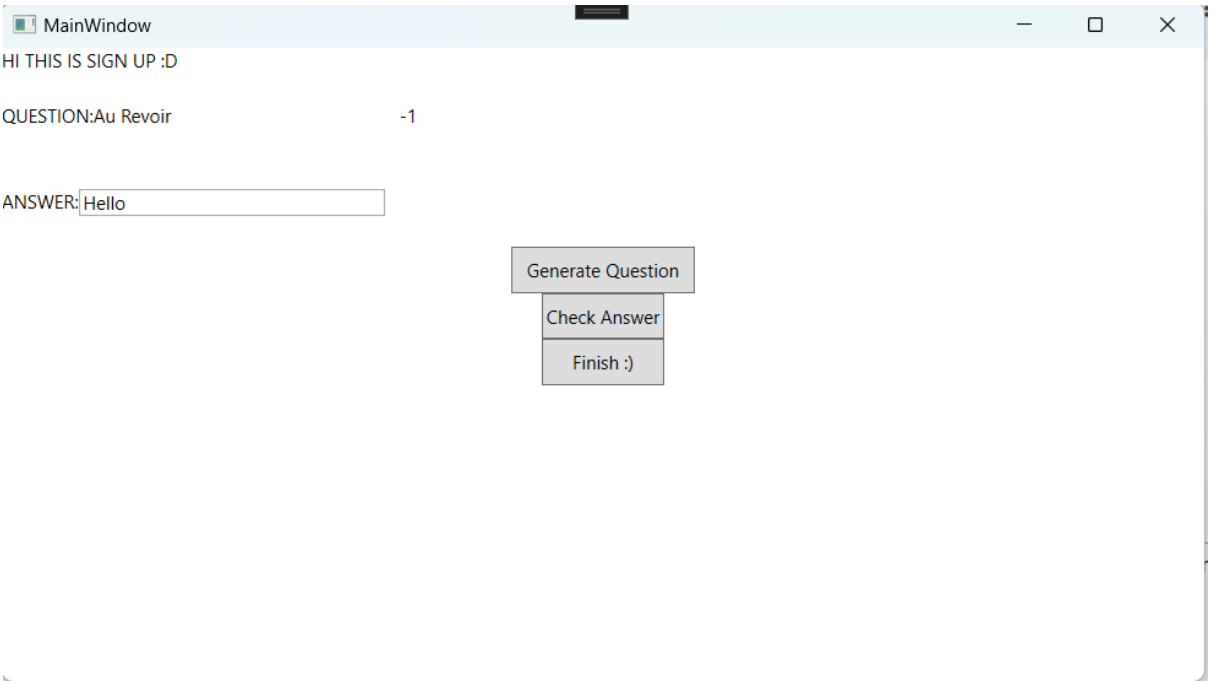
- A message box acknowledges the correct answer and Score increases by 1.

TEST 47:



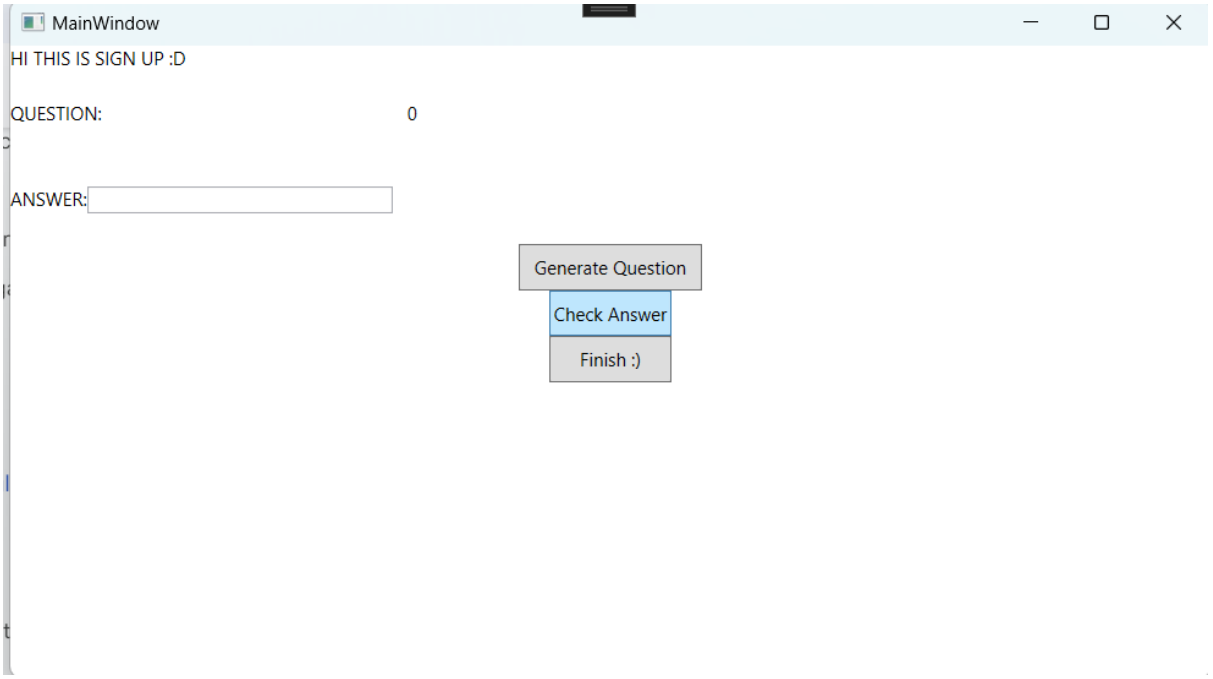
- QUIZ PAGE: Incorrect answer is entered and Check Answer Button is pressed.



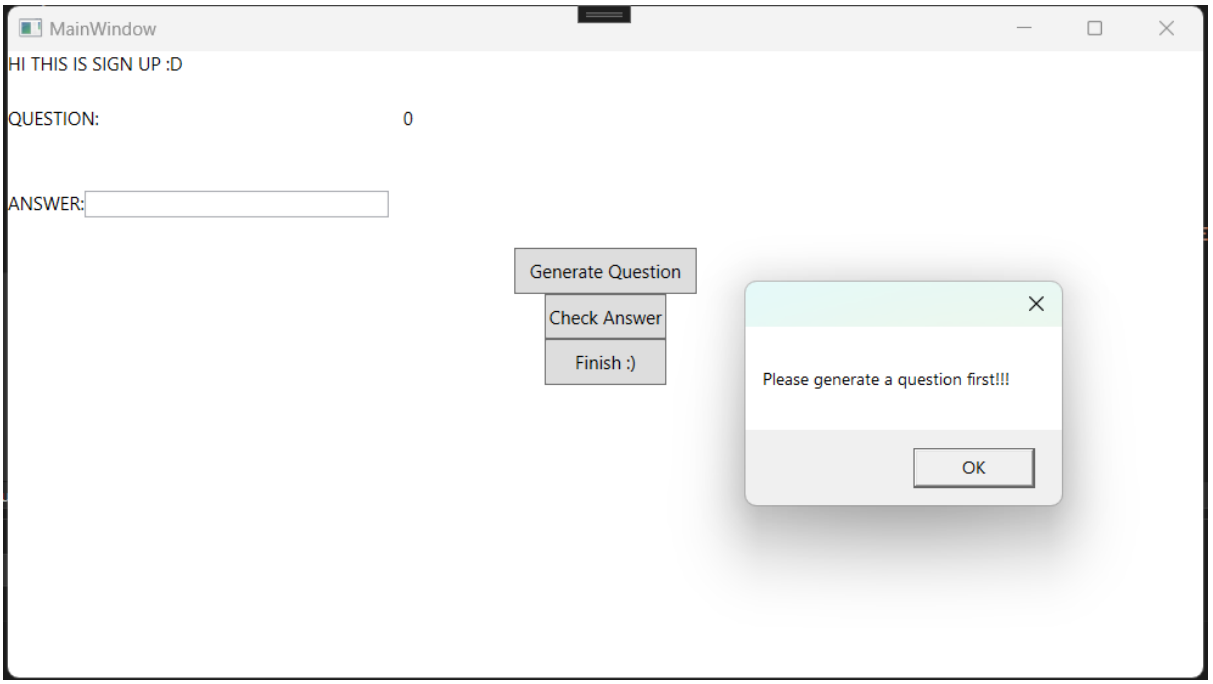


- A message box reveals the correct answer and Score decreases by 1.

TEST 48:

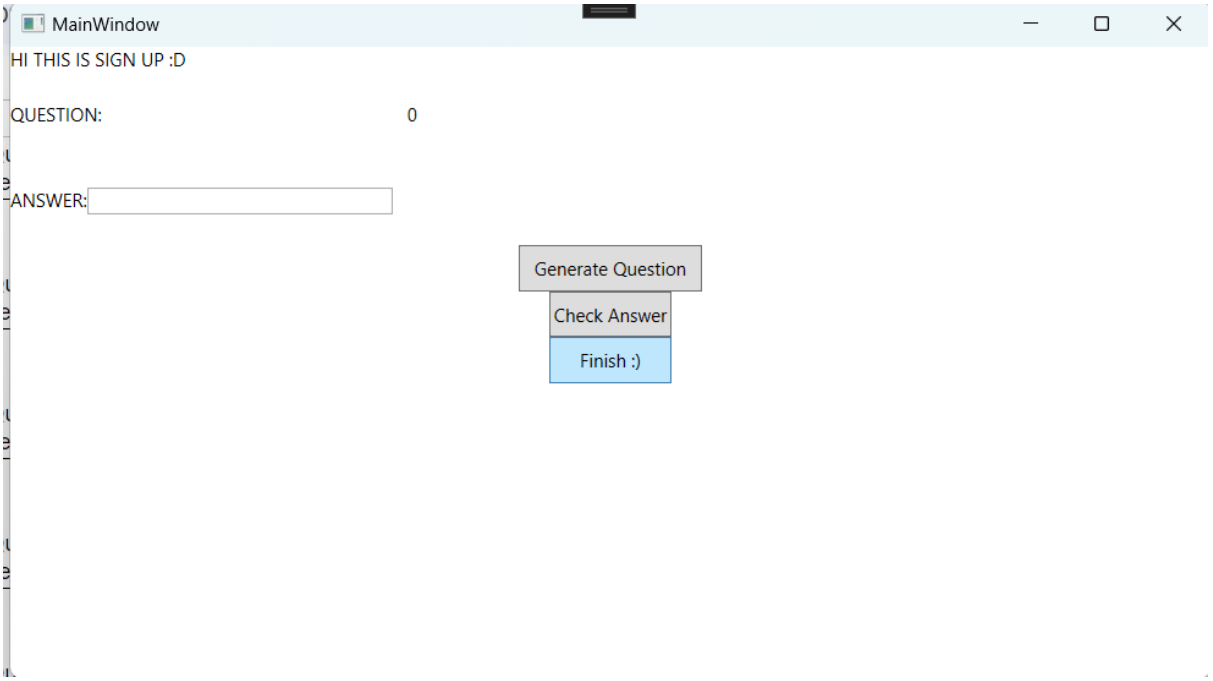


- QUIZ PAGE: Check Answer button is pressed, without generating a question.

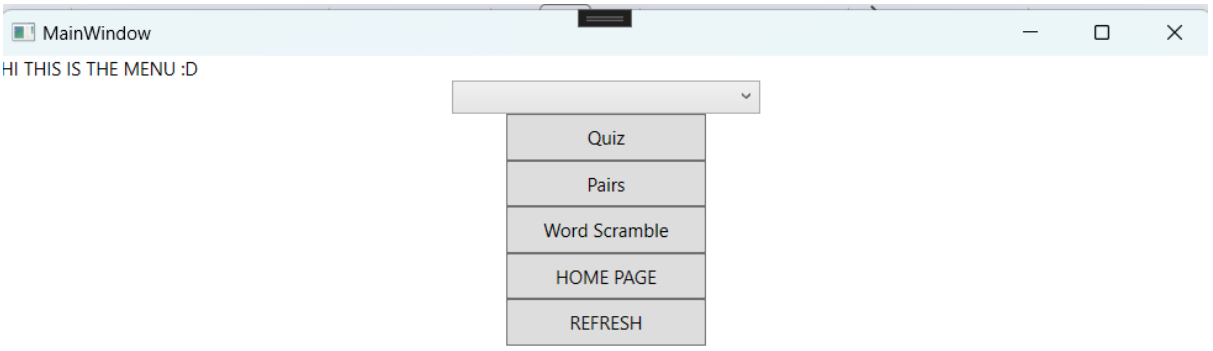


- A message box appears reminding the user to generate a question.

TEST 49:

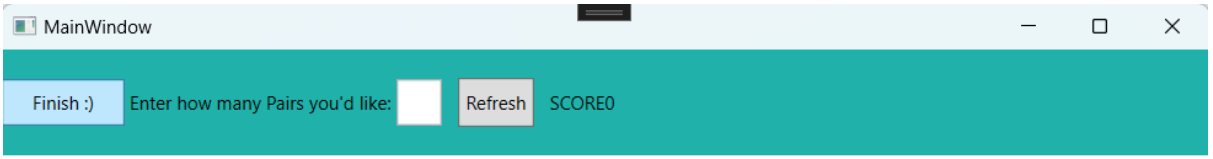


- QUIZ PAGE: Finish Button Pressed

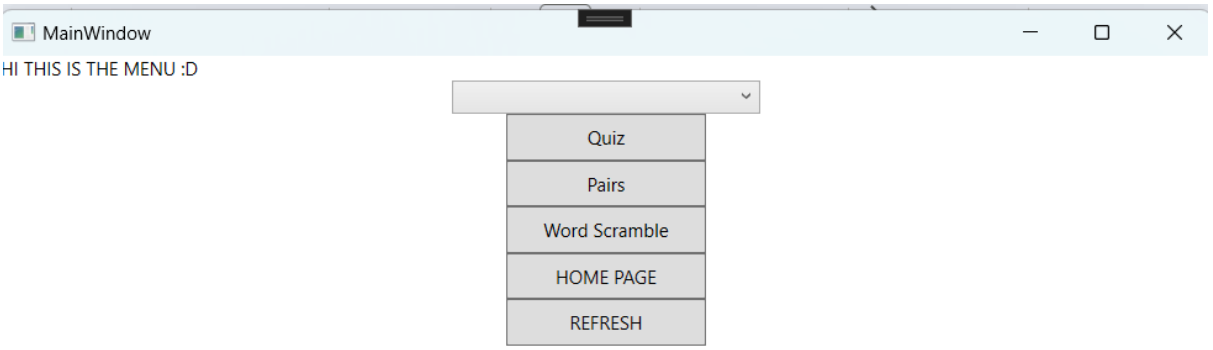


- Changes to Game Menu Page

TEST 50:

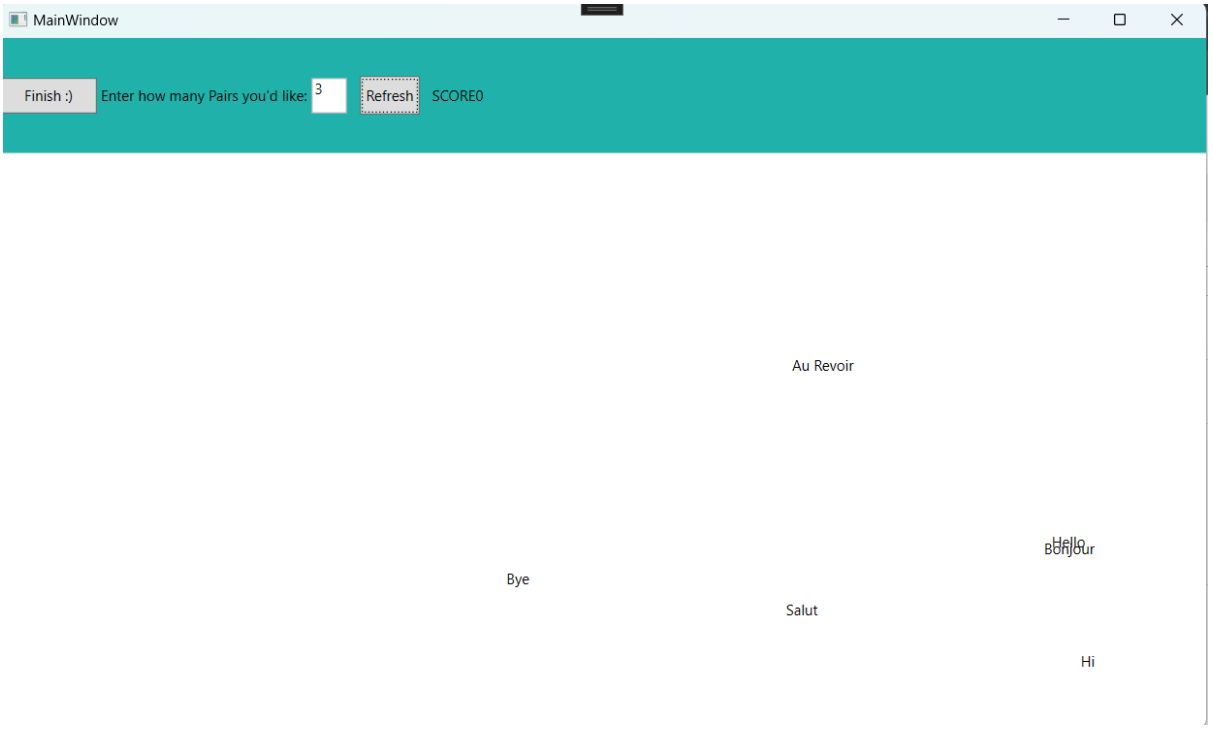


- PAIRS GAME PAGE: Finish Button is pressed.

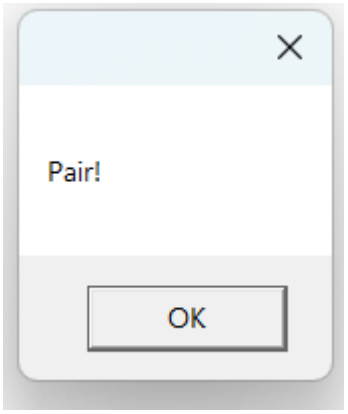


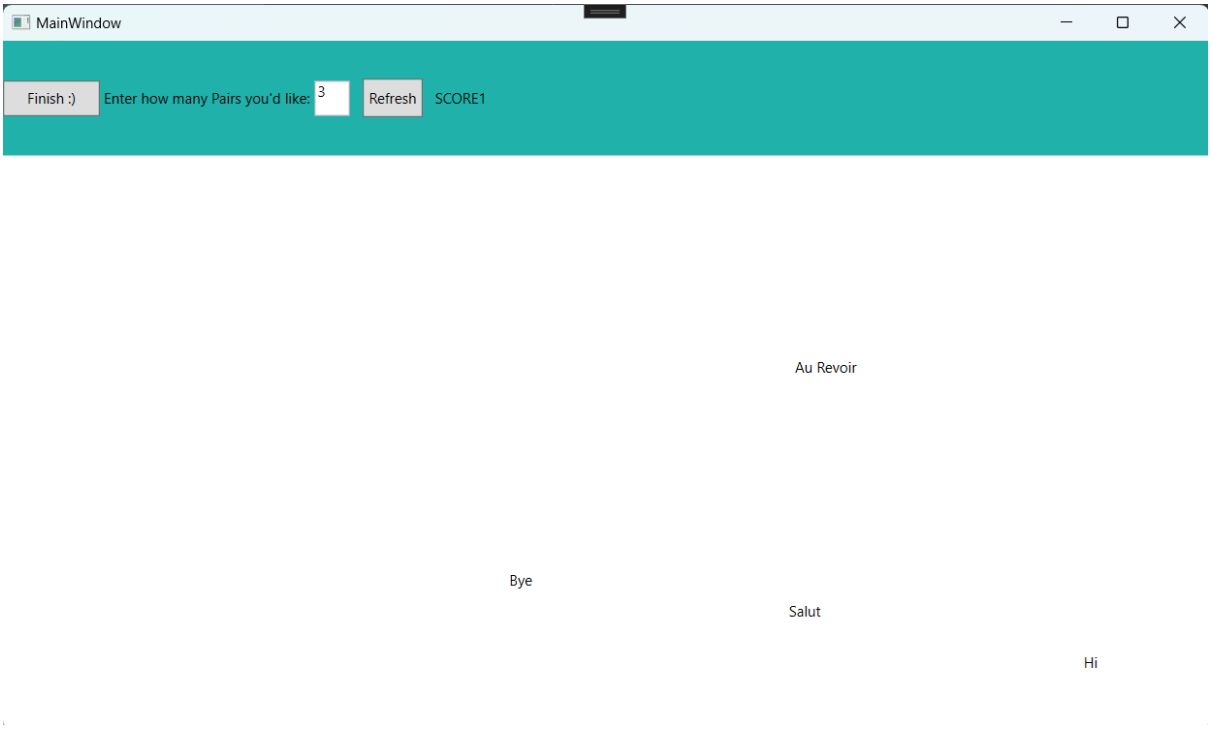
- Changes to Game Menu Page

TEST 51:



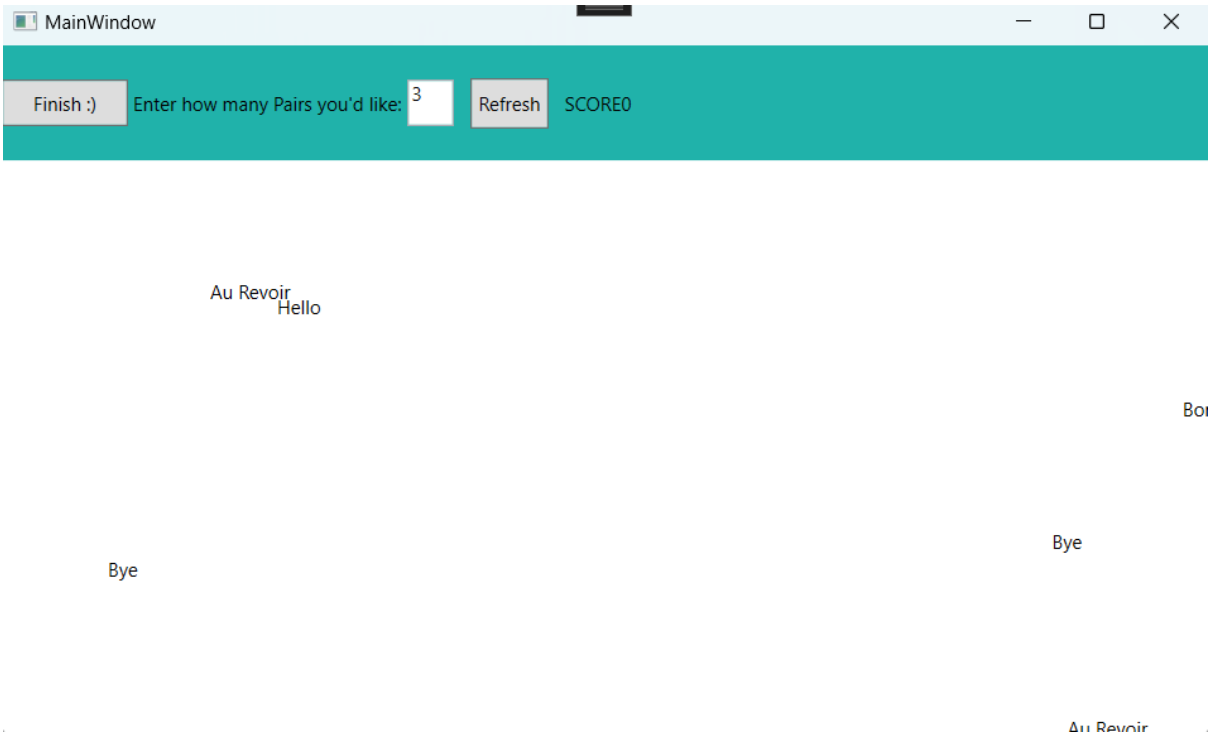
- PAIRS GAME: Drag correct pairs together (Hello & Bonjour)



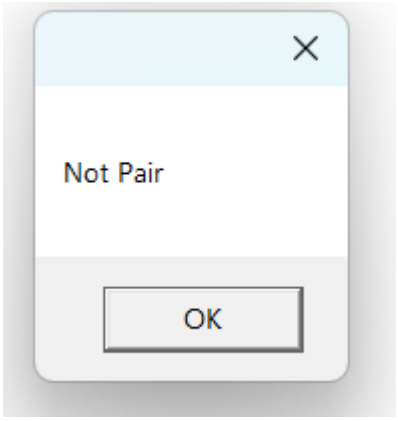


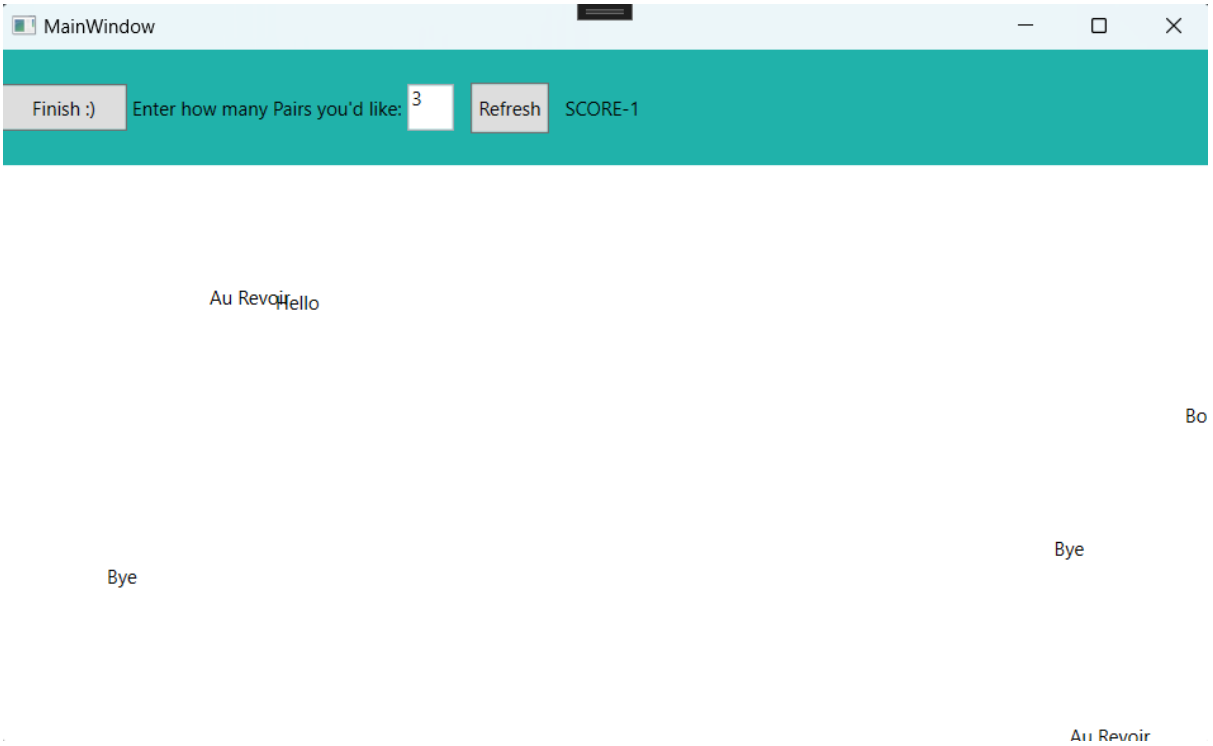
- Pop-up window to acknowledge correct answer, score increases by 1 and correct pair disappears.

TEST 52:



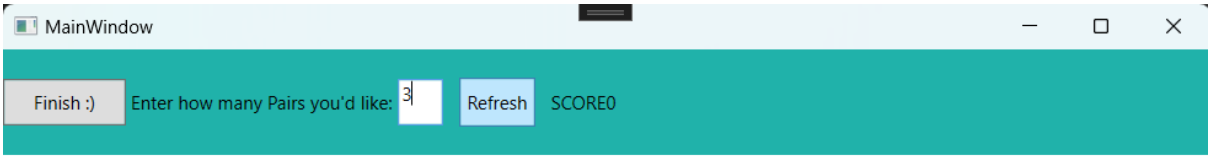
- PAIRS GAME: Drag correct pairs together (Hello & Bonjour)



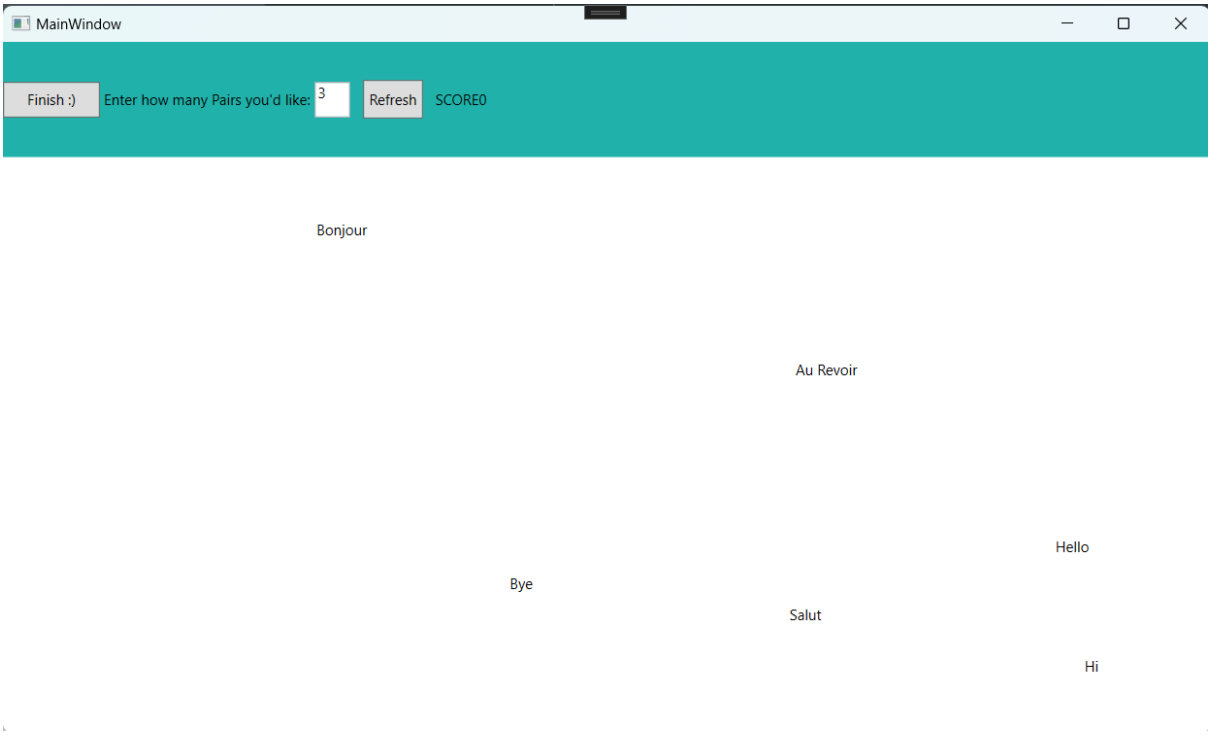


- Pop-Up shows that it is not a pair and the score decreases by 1.

TEST 53:

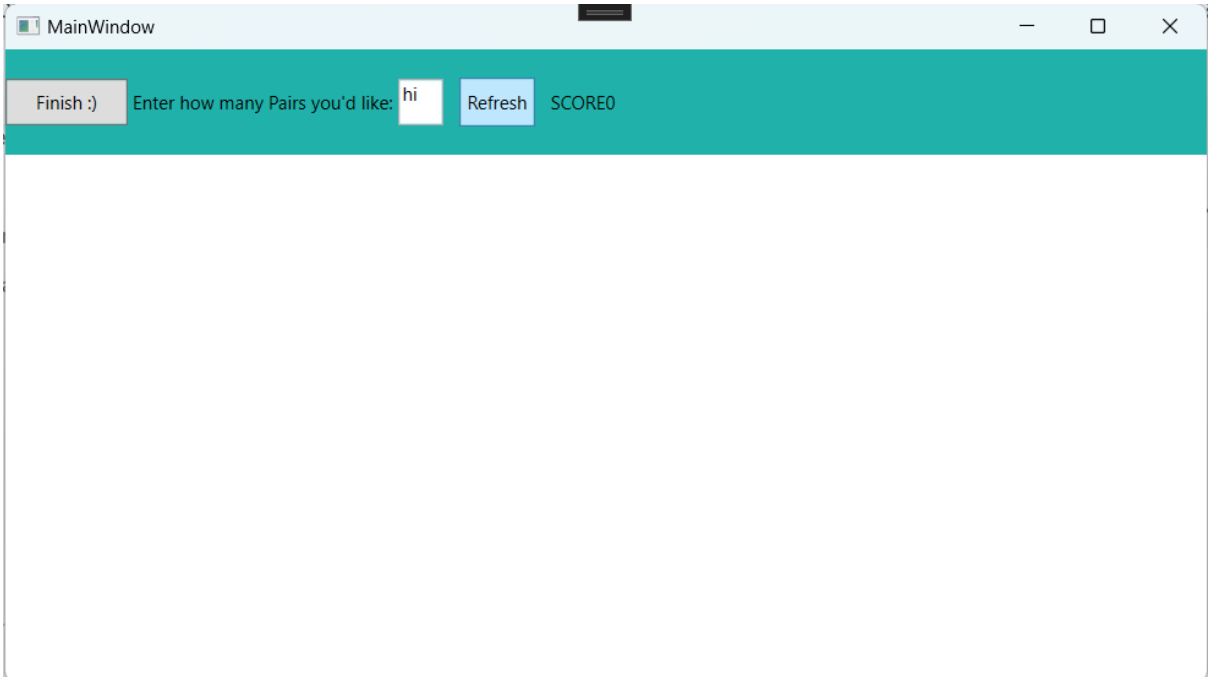


- PAIRS GAME PAGE: Correct data type entered and Refresh button pressed.

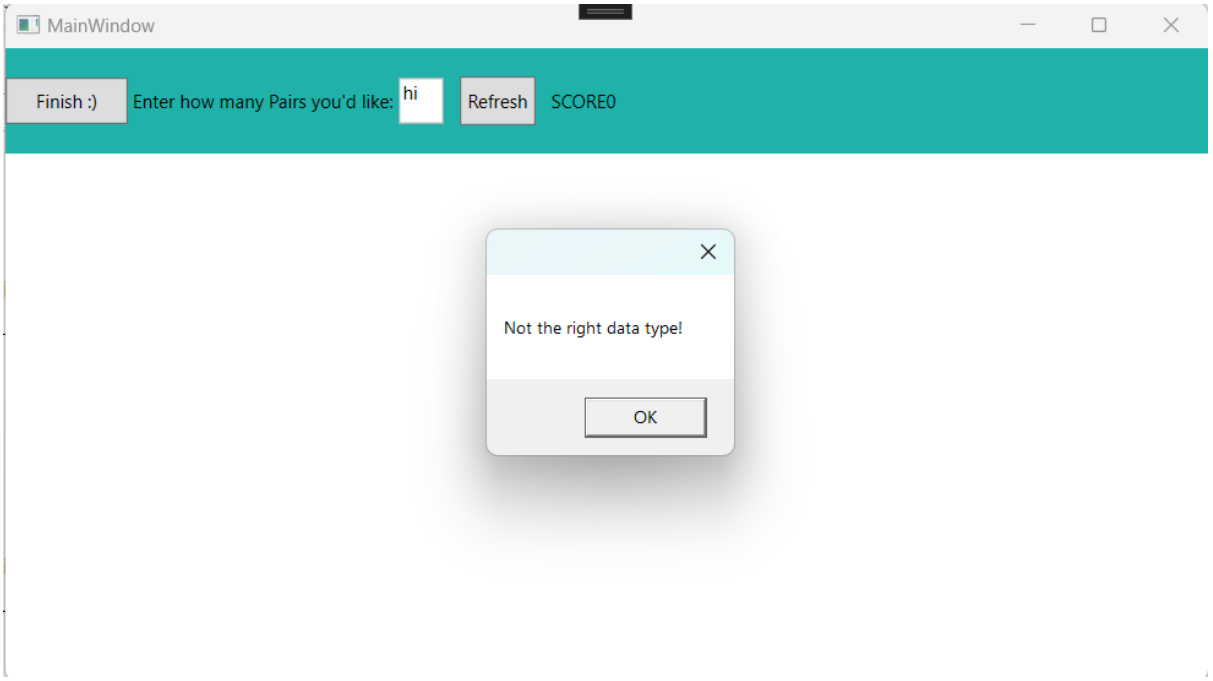


- Pairs randomly generated and displayed in random positions.

TEST 54:

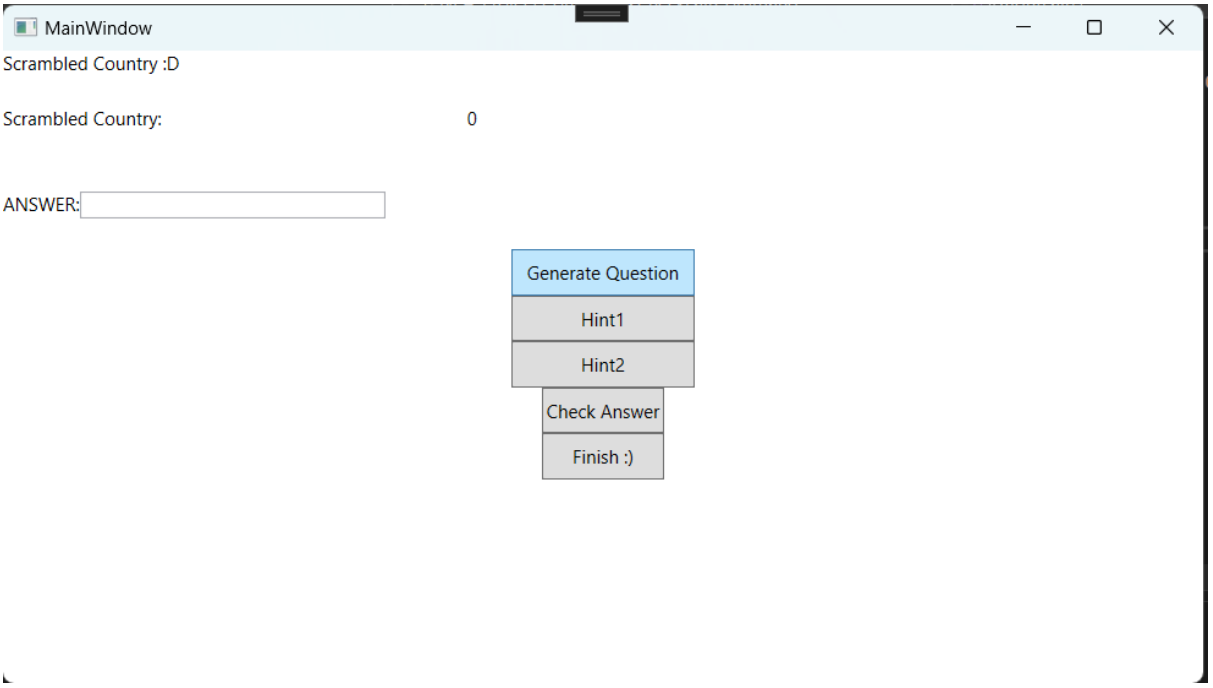


- PAIRS GAME PAGE: Incorrect data type and Refresh button pressed

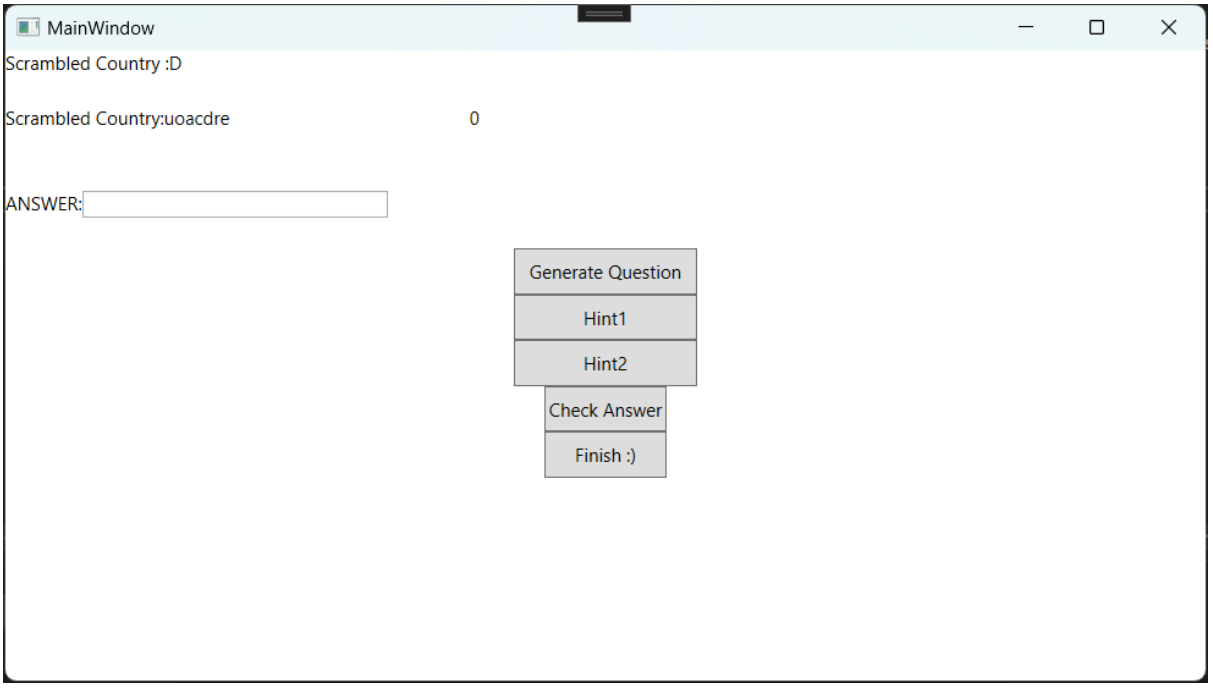


- Message Box Appears to alert the user.

TEST 55:

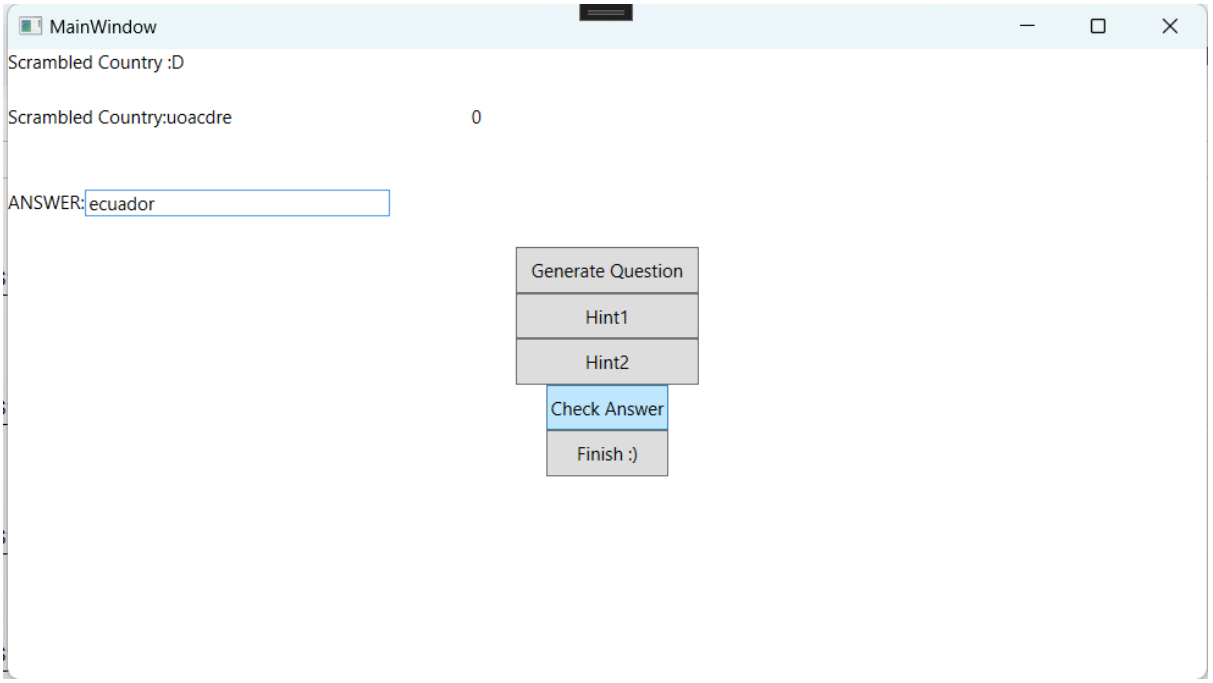


- WORD SCRAMBLE PAGE: Generate Question Button Pressed.

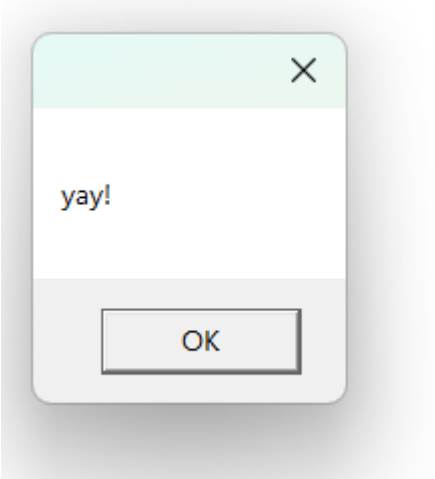


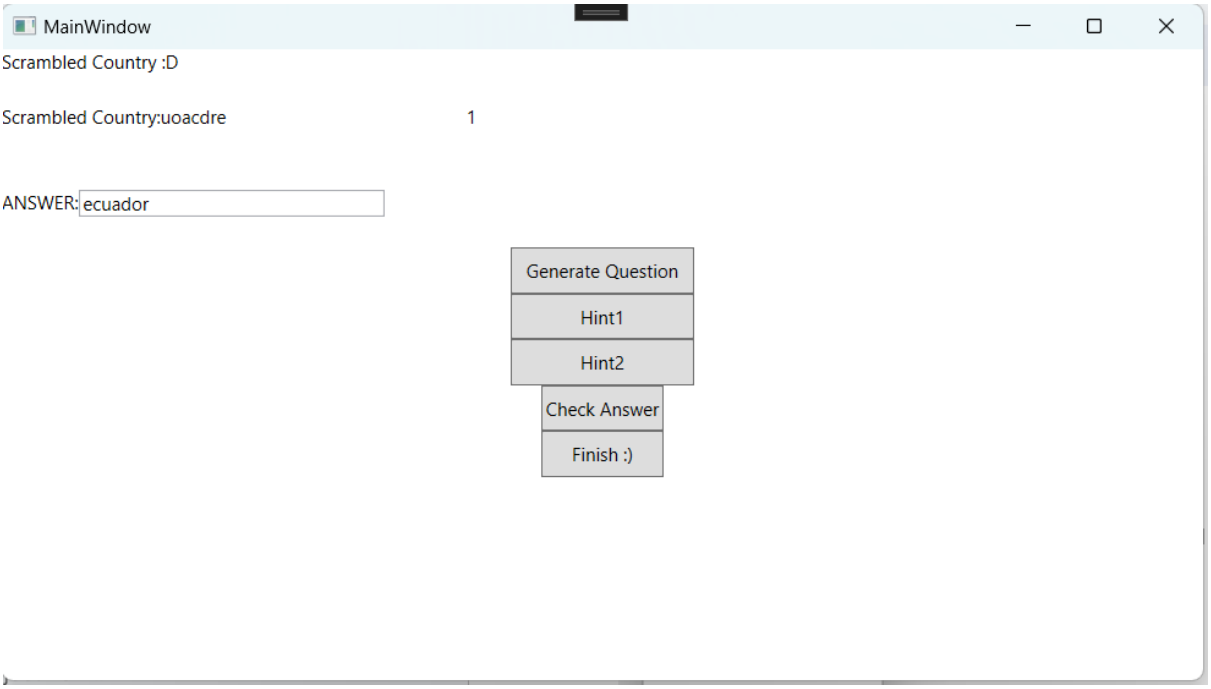
- Displays a random scrambled country name.

TEST 56:



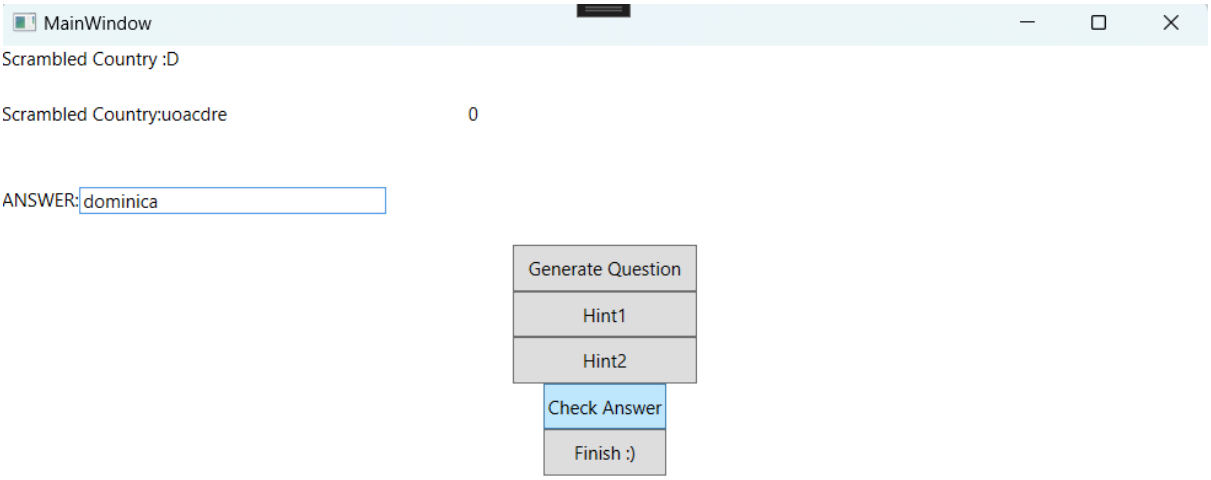
- WORD SCRAMBLE PAGE: Correct Answer entered and Check Answer Pressed.



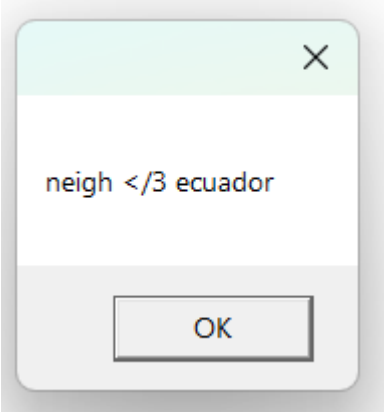


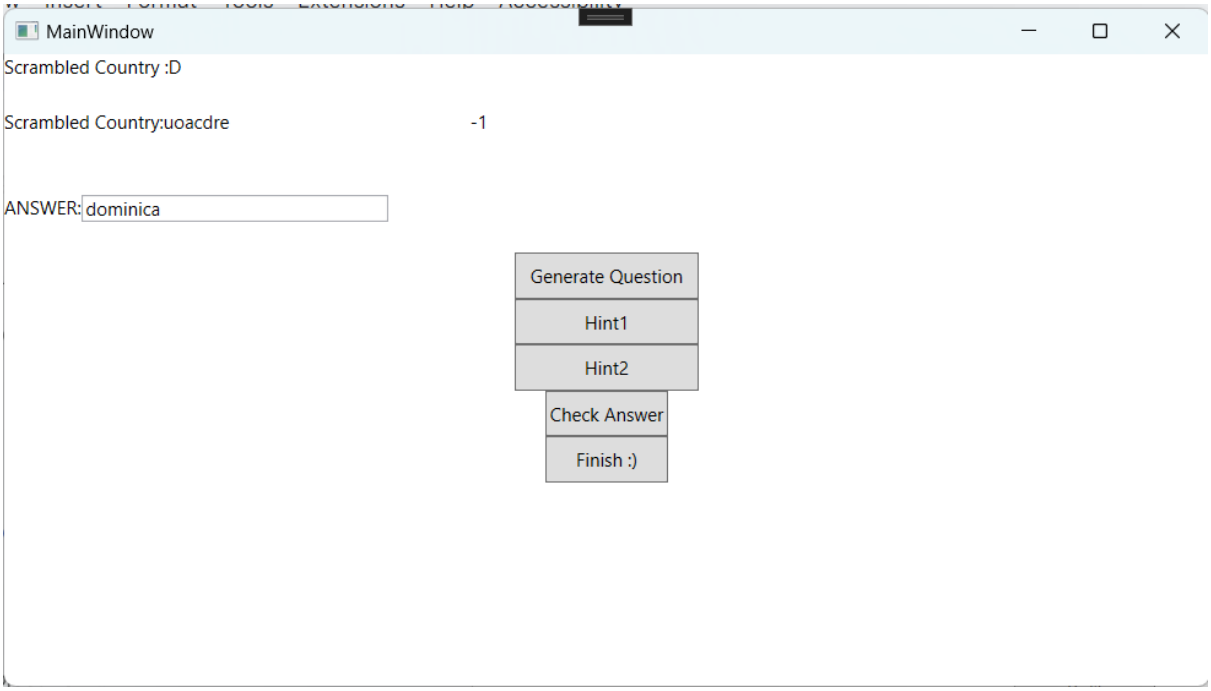
- Message box appears to indicate the correct answer and score increases by 1.

TEST 57:



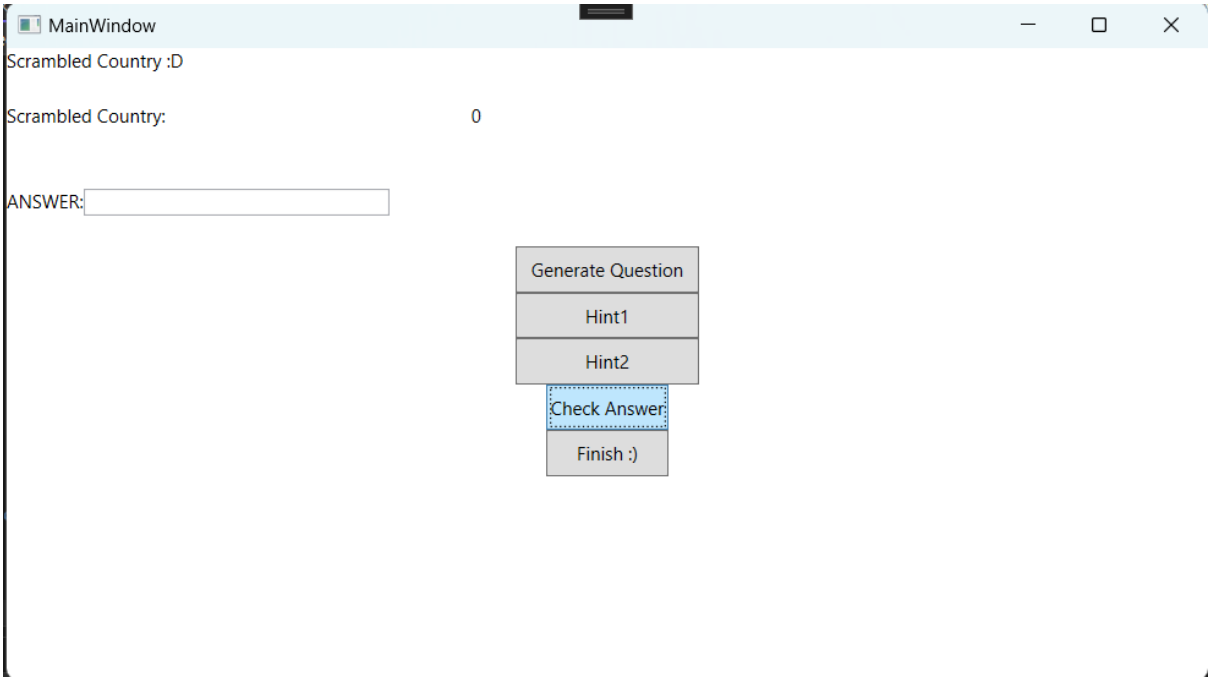
- WORD SCRAMBLE PAGE: Incorrect Answer entered and Check Answer Pressed.





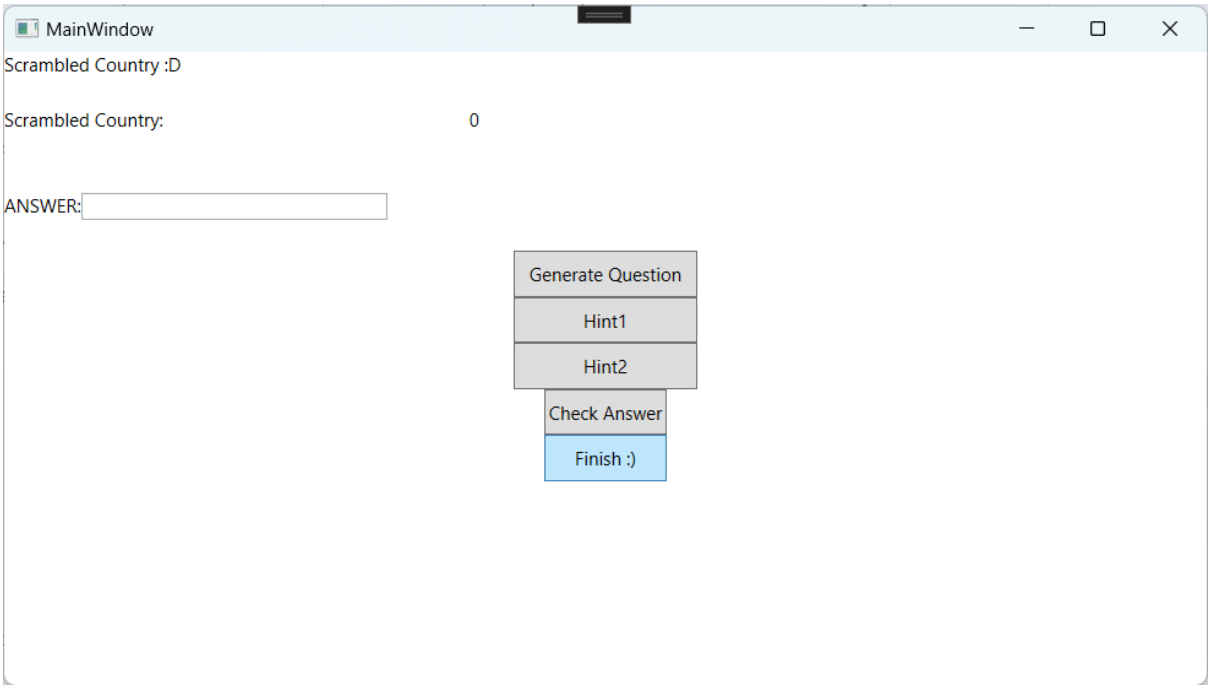
- Message box appears to indicate an incorrect answer and score decreases by 1.

TEST 58:

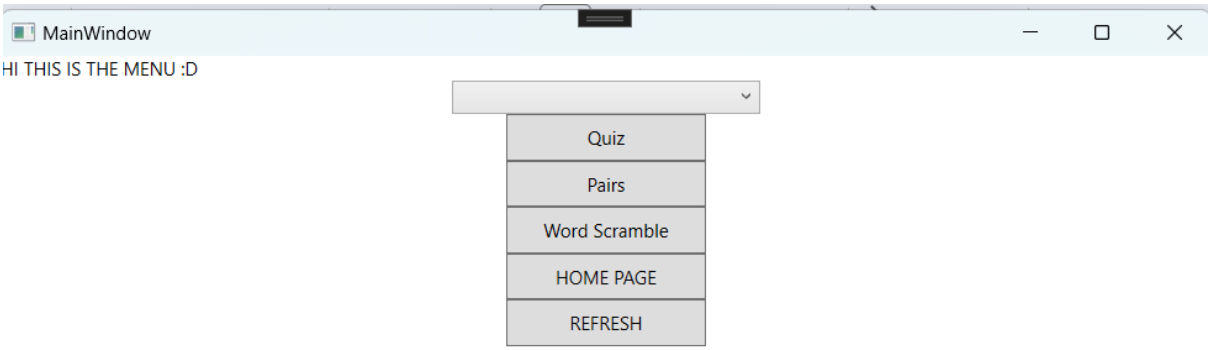


- WORD SCRAMBLE PAGE: When the Check answer button is pressed and no question is generated, the program doesn't crash.

TEST 59:

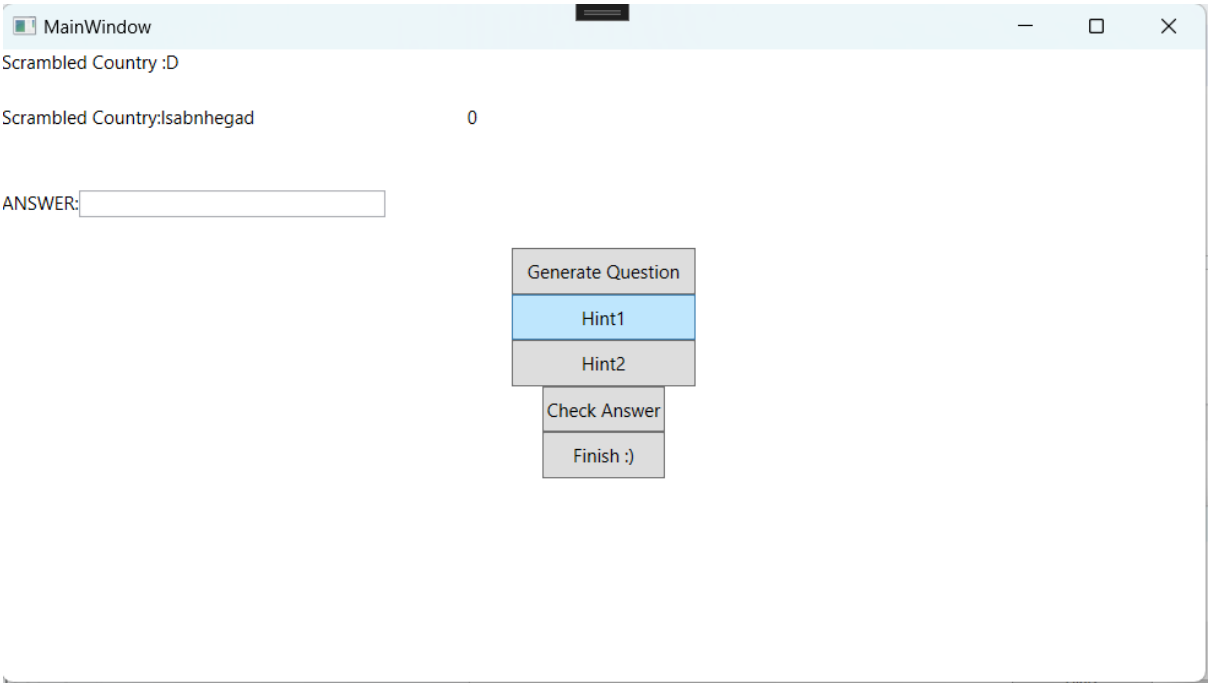


- WORD SCRAMBLE PAGE: Finish Button is pressed

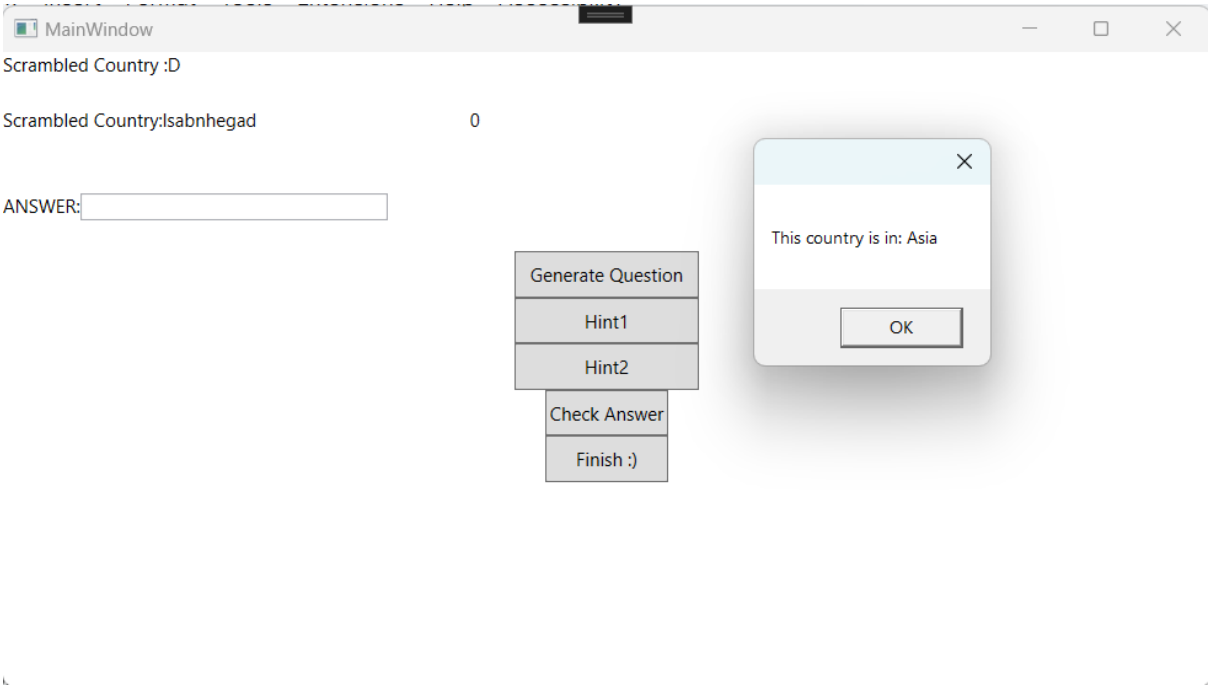


- Changes to the GAME MENU PAGE.

TEST 60:

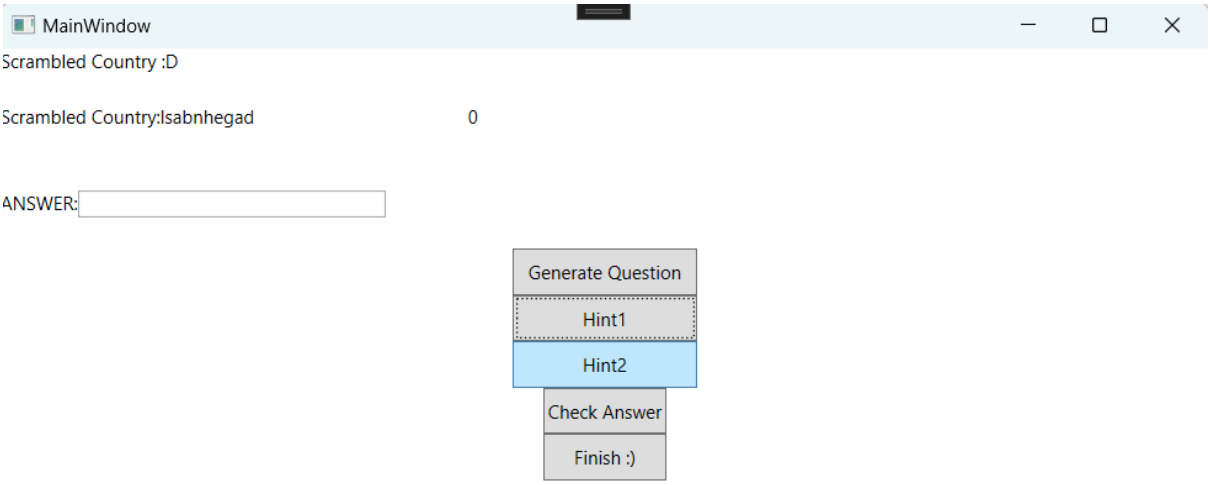


- WORD SCRAMBLE PAGE: Hint 1 Button is pressed

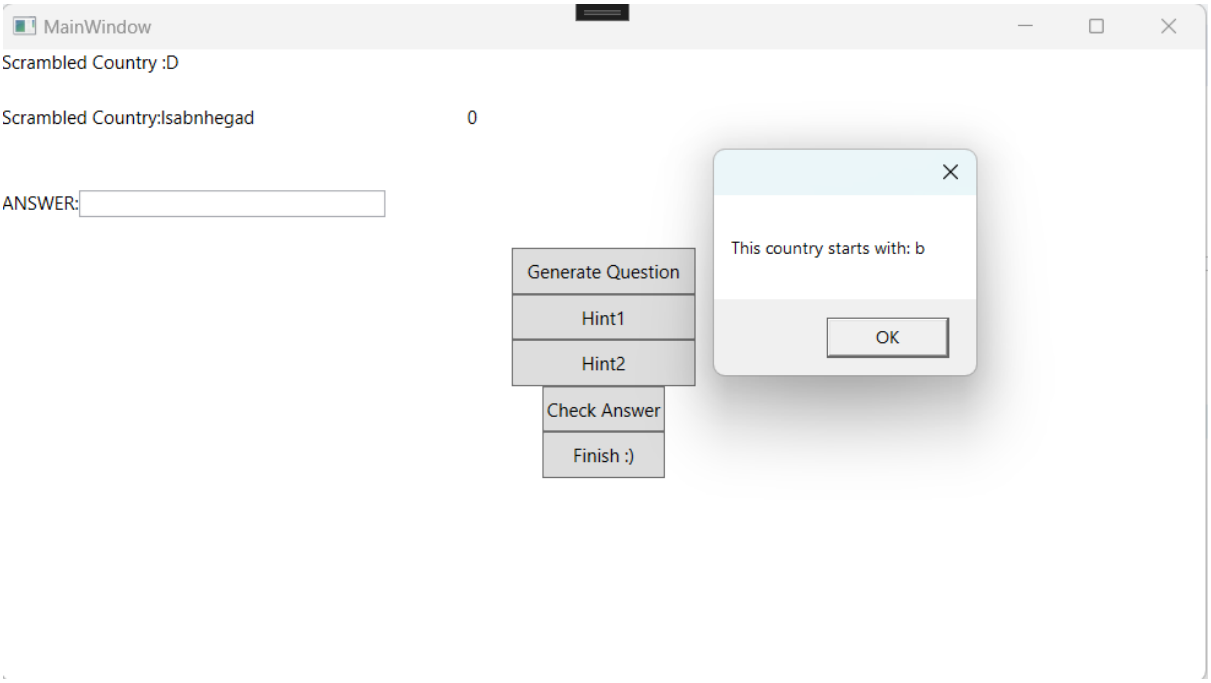


- Message Box with hint is displayed

TEST 61:

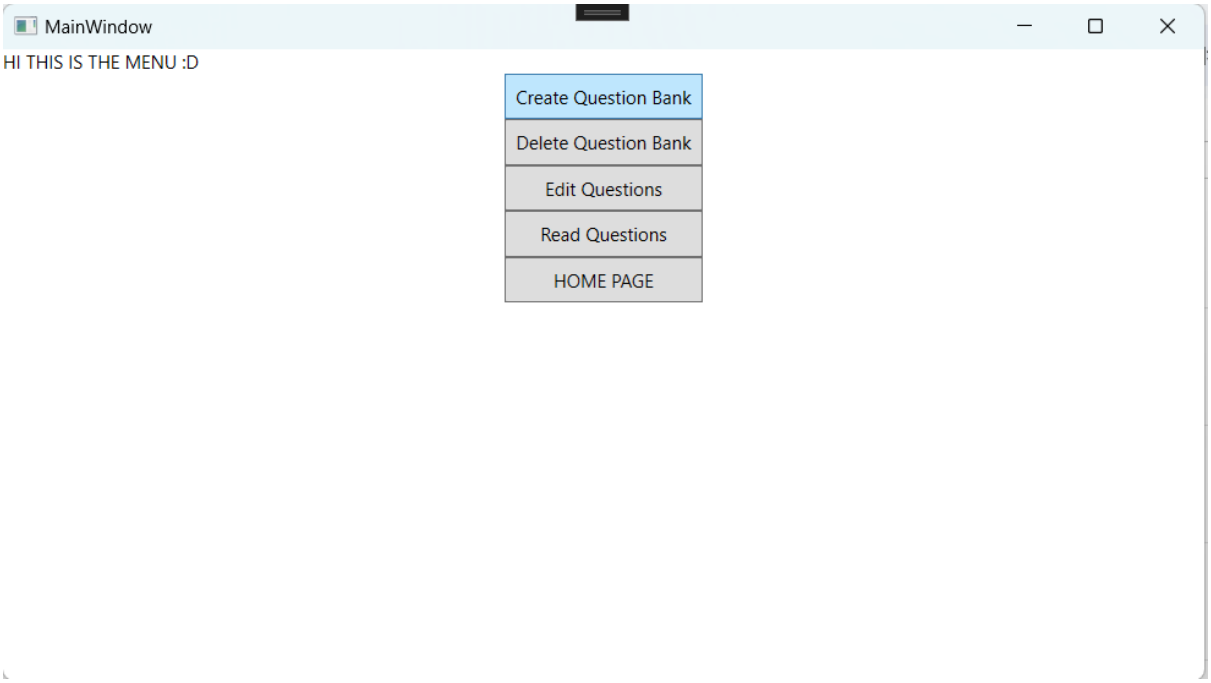


- WORD SCRAMBLE PAGE: Hint 2 Button is pressed

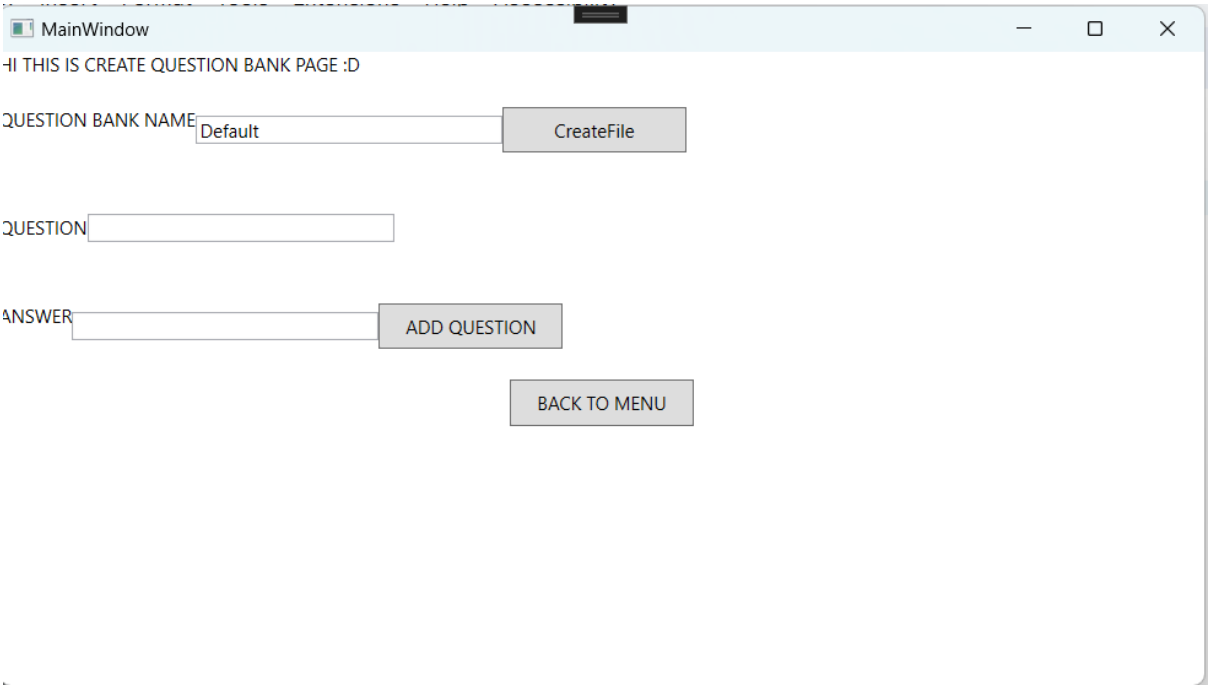


- Message Box with hint is displayed

TEST 62:



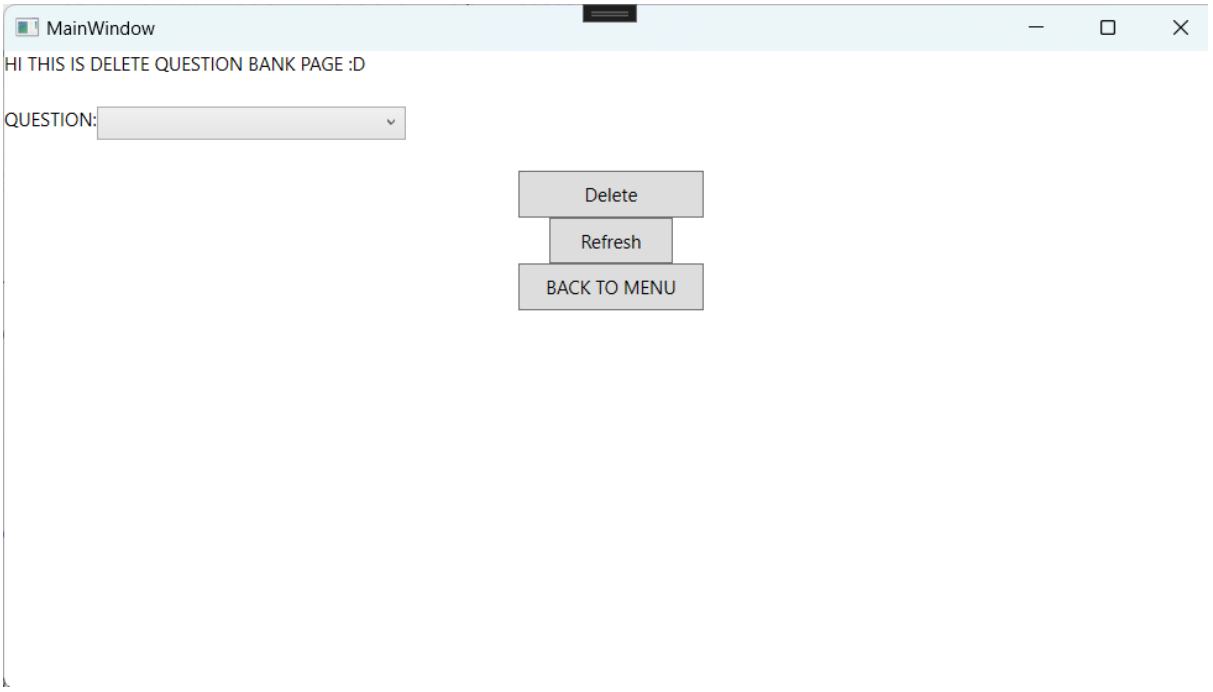
- QUESTION BANK MENU PAGE: Create Button Pressed



- Changed to QUESTION BANK CREATE PAGE

TEST 63:

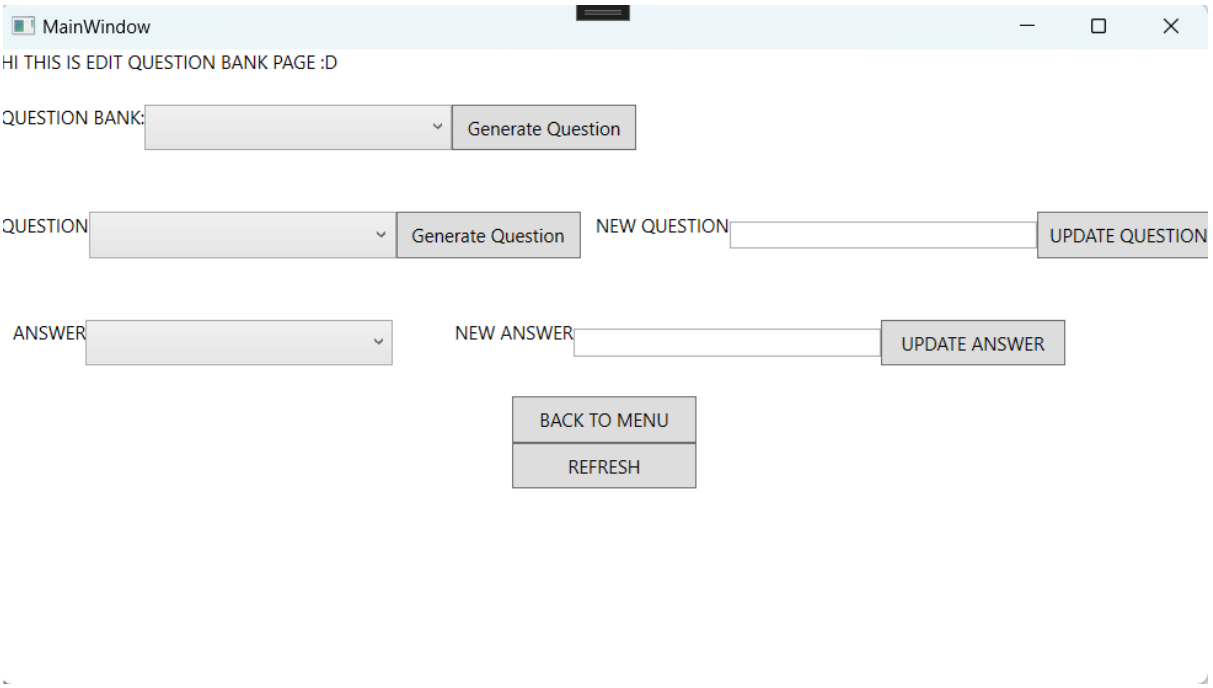
- QUESTION BANK MENU PAGE: Delete Button Pressed



- Changed to QUESTION BANK DELETE PAGE

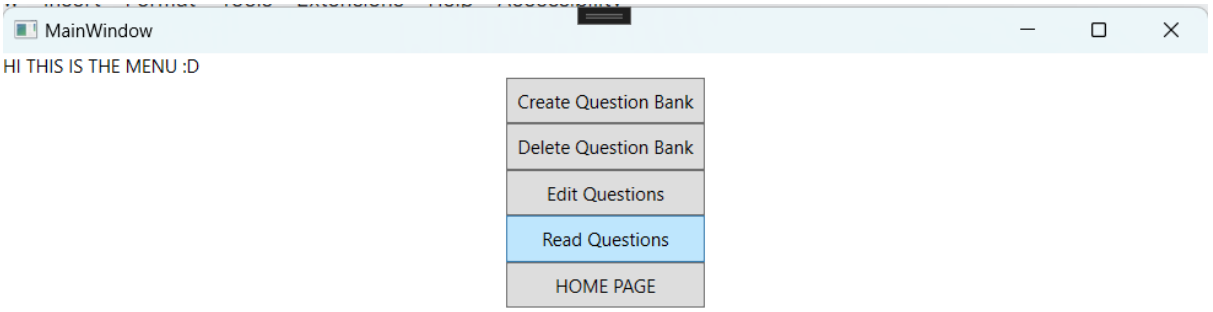
TEST 64:

- QUESTION BANK MENU PAGE: Edit Button Pressed

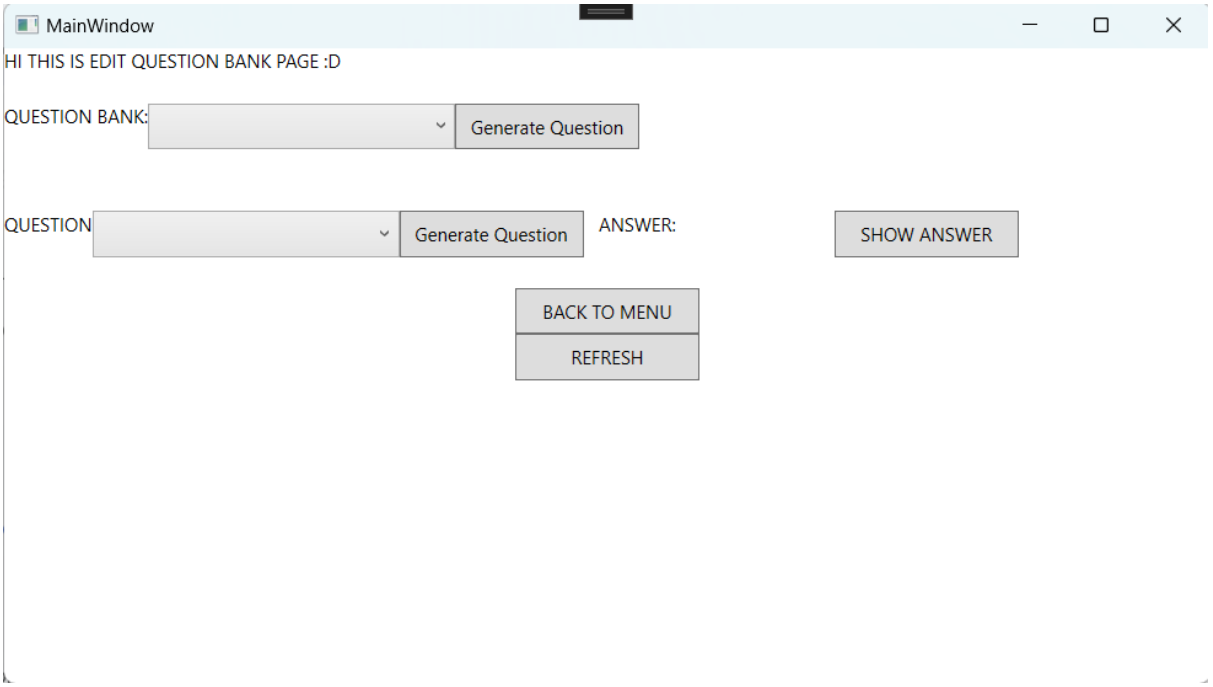


- Changed to QUESTION BANK EDIT PAGE

TEST 65:

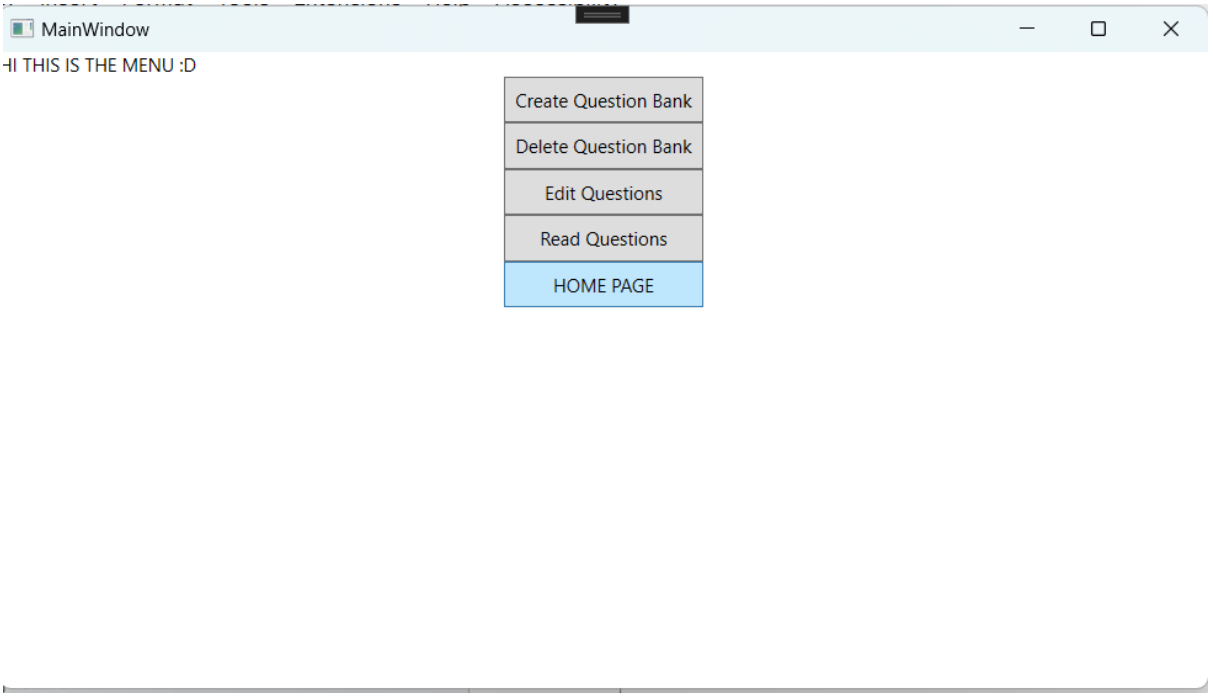


- QUESTION BANK MENU PAGE: Read Button Pressed

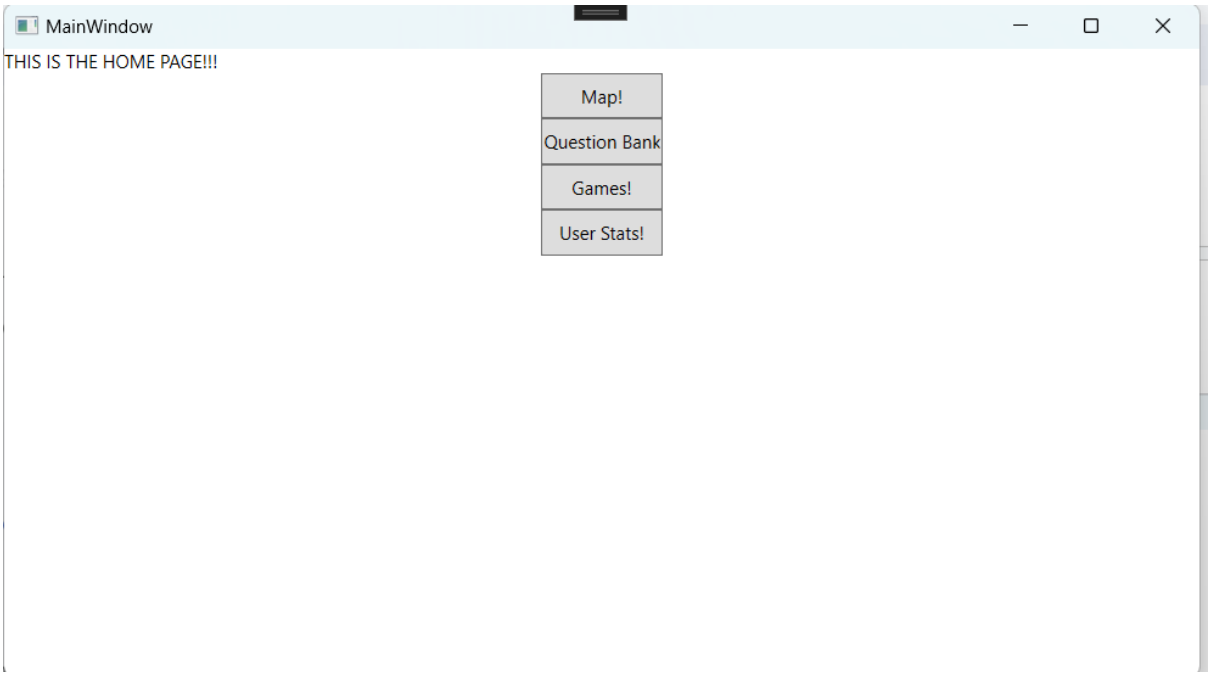


- Changed to QUESTION BANK READ PAGE

TEST 66:

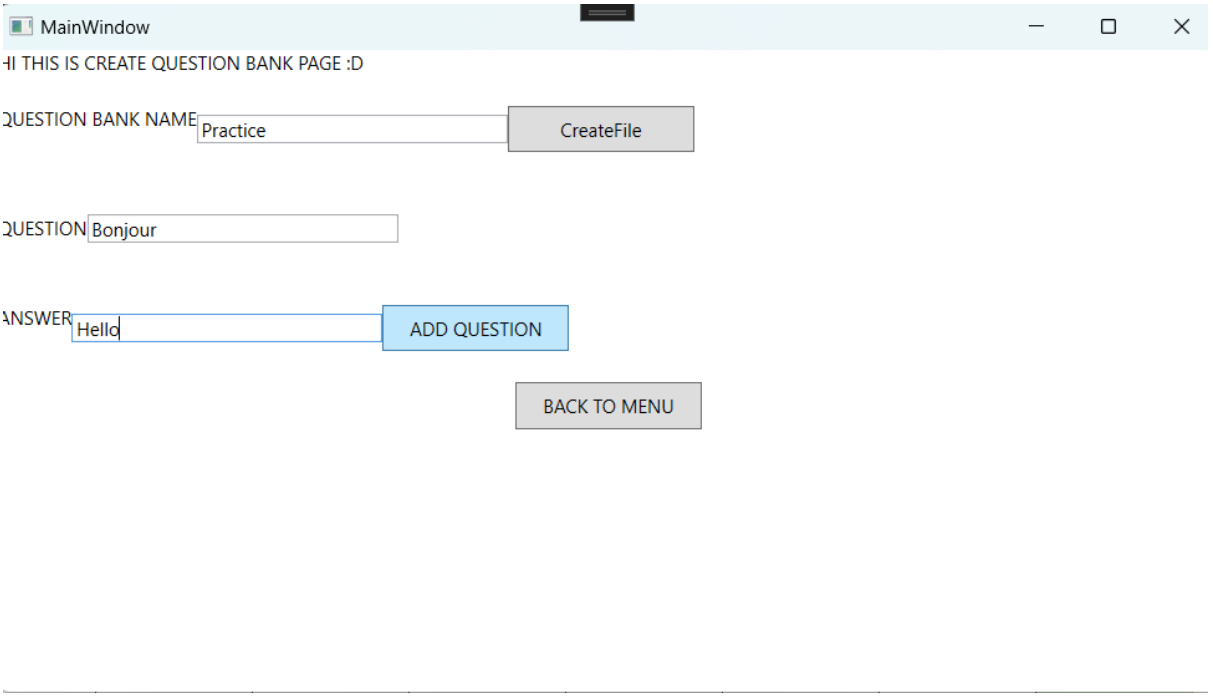


- QUESTION BANK MENU PAGE: Home Button Pressed



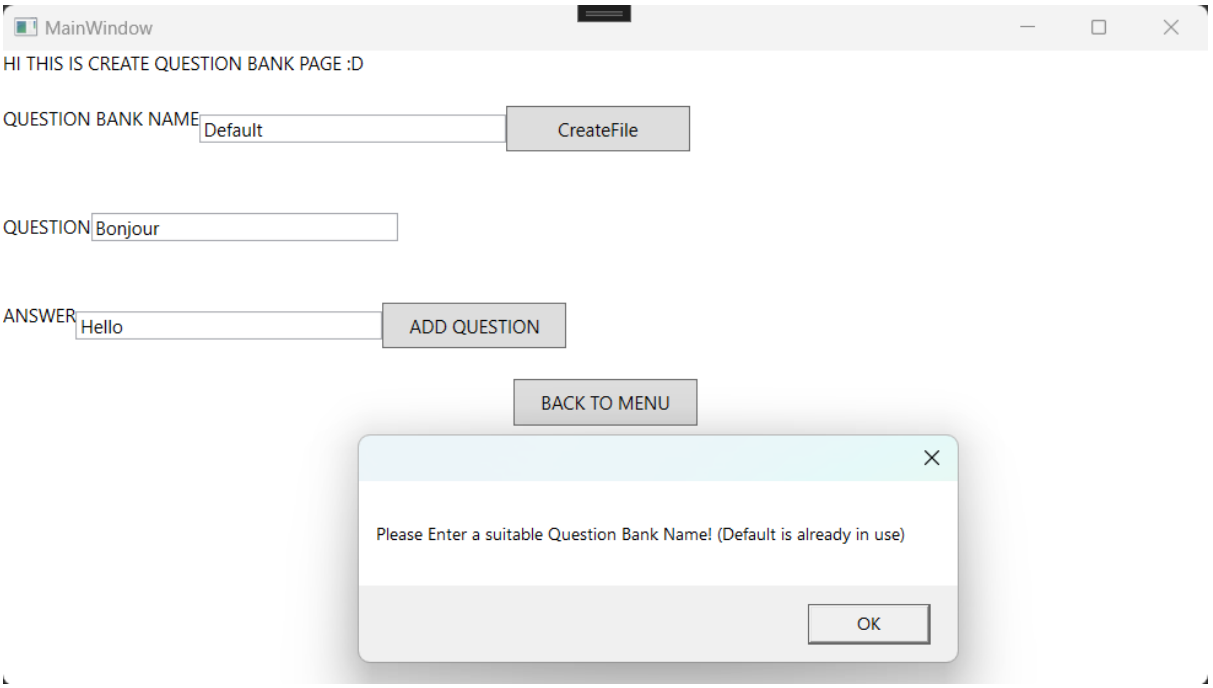
- Changed to HOME PAGE

TEST 67:



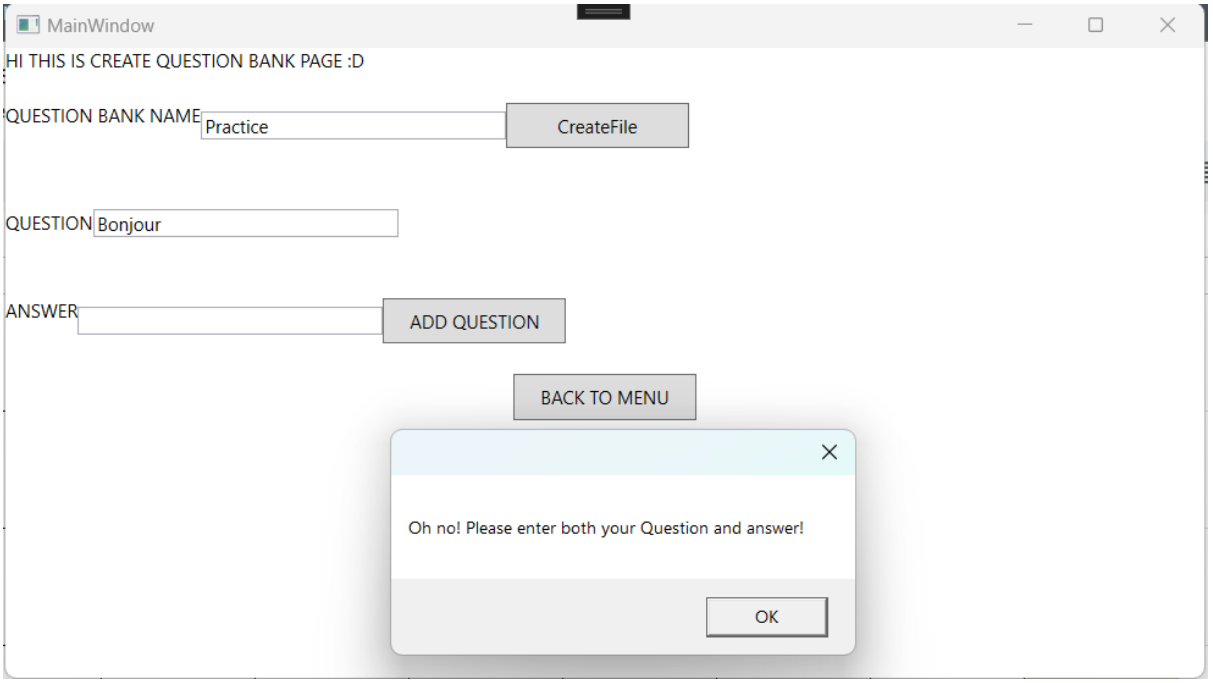
- QUESTION BANK CREATE PAGE: All correct inputs, no error message appears.

TEST 68:



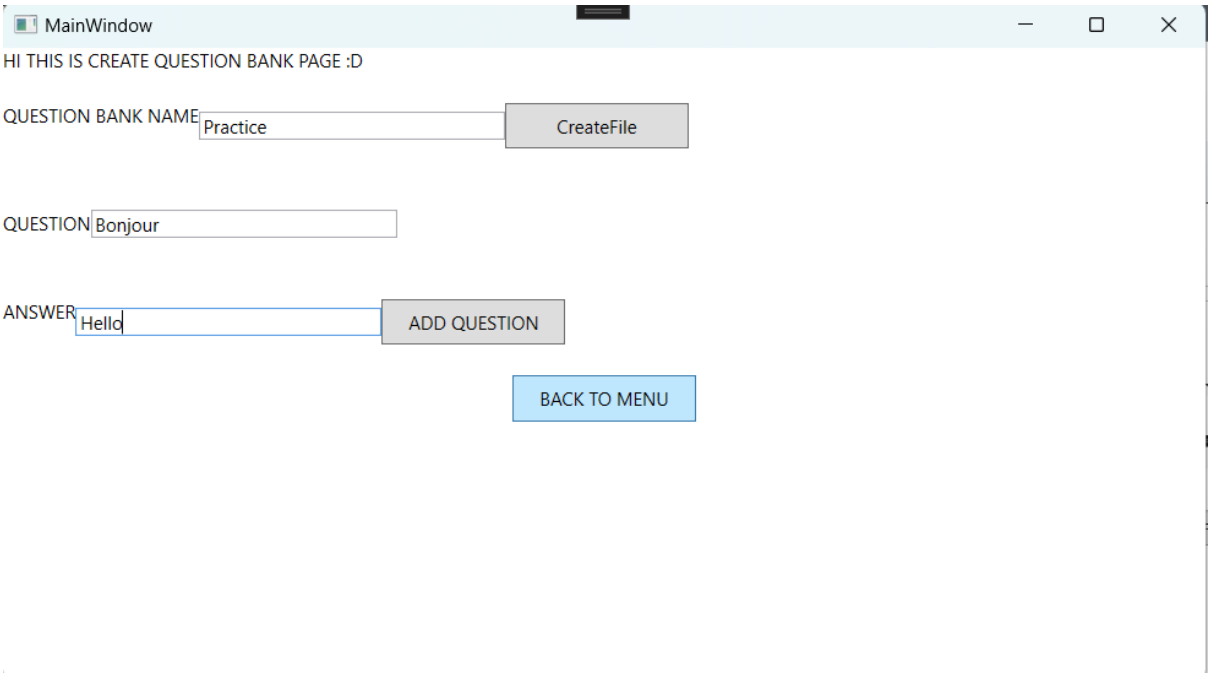
- QUESTION BANK CREATE PAGE: Pop-up when incorrect Question Bank Name is used and Add question is pressed

TEST 69:

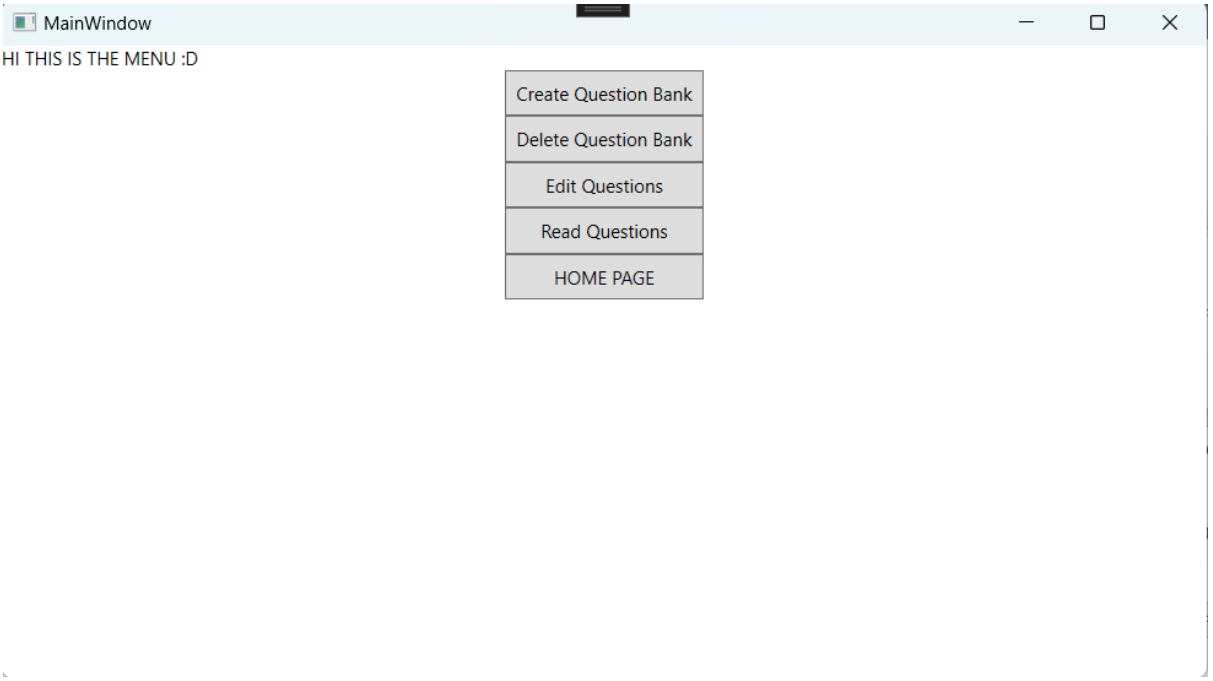


- QUESTION BANK CREATE PAGE: Pop-Up when either Question or Answer is null and Add Question is pressed.

TEST 70:

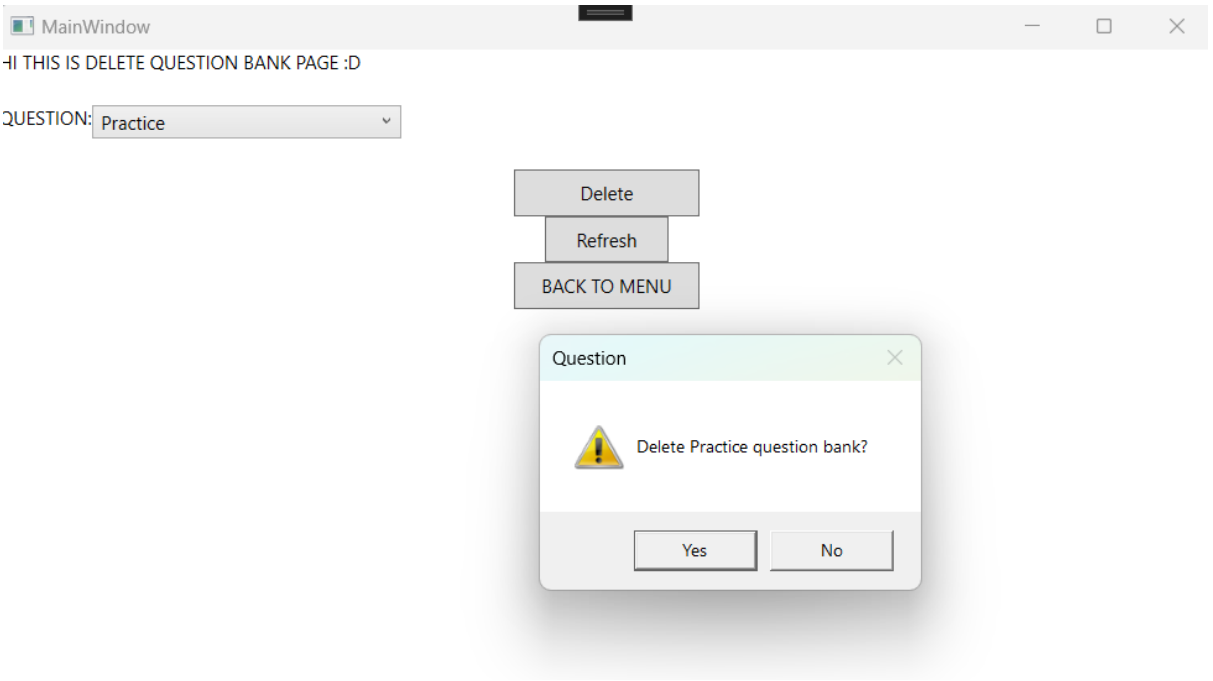


- QUESTION BANK CREATE PAGE: Back to Menu Pressed.

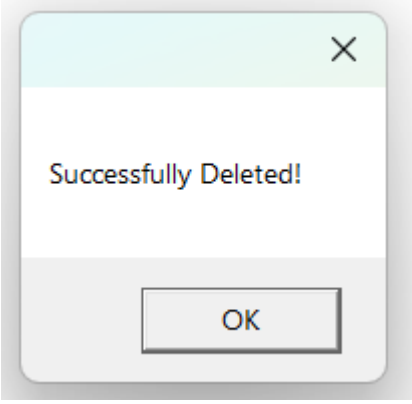


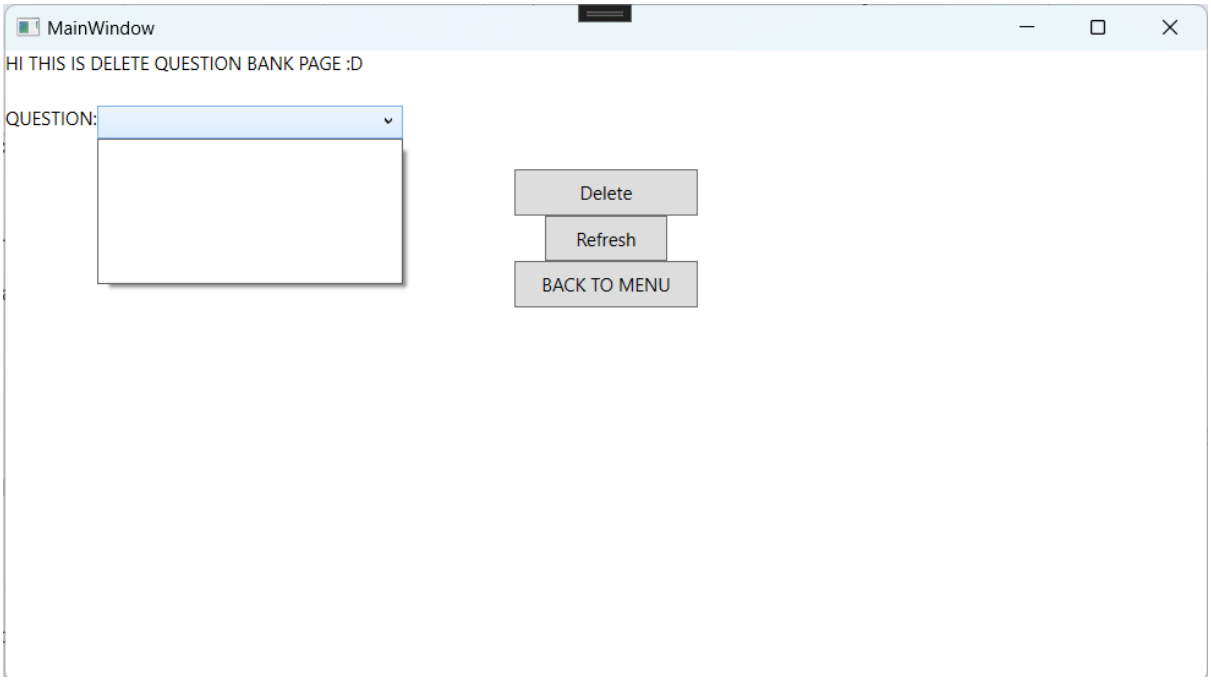
- Changed to the Question Bank Menu.

TEST 71:



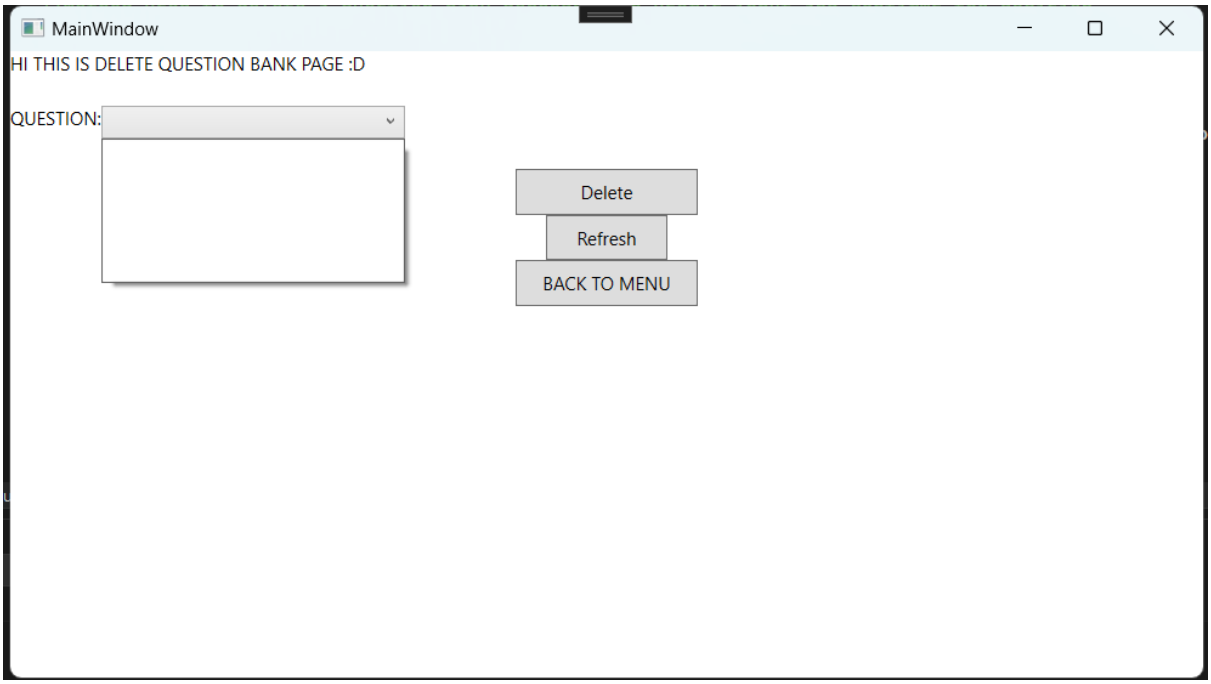
- QUESTION BANK DELETE PAGE: When the Delete Button is pressed, a message box appears, asking the user to confirm.



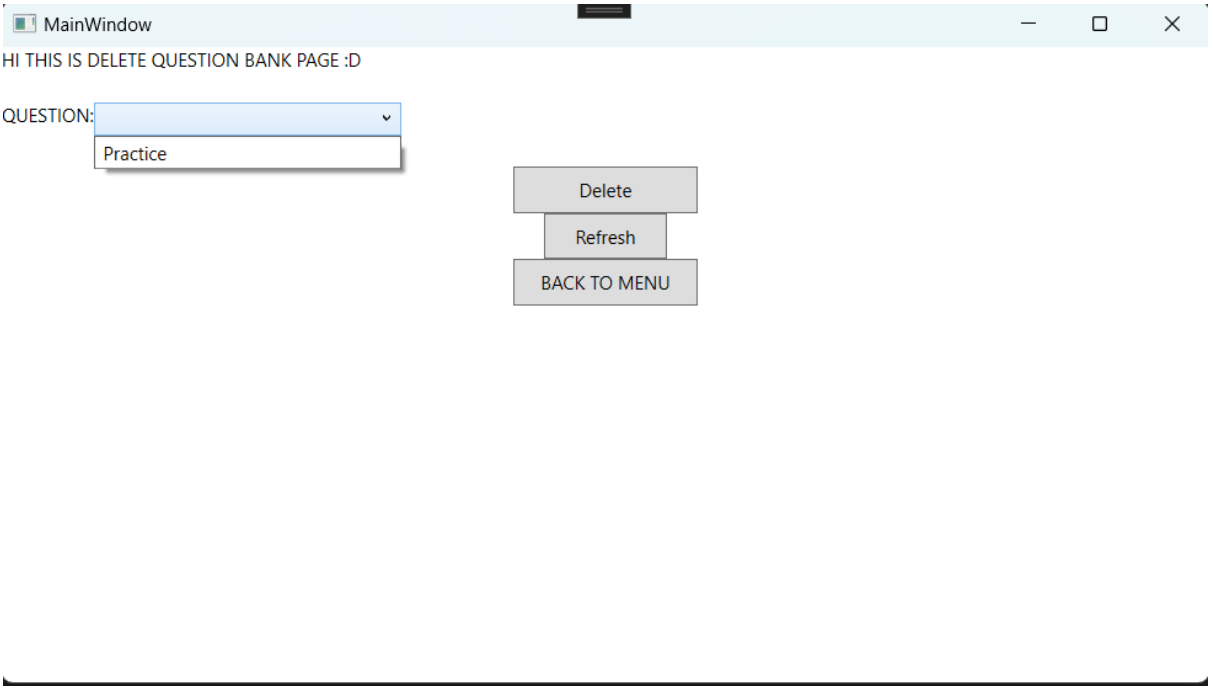


- Pop-Up Window indicating bank is deleted and now deleted from the Combobox.

TEST 72:

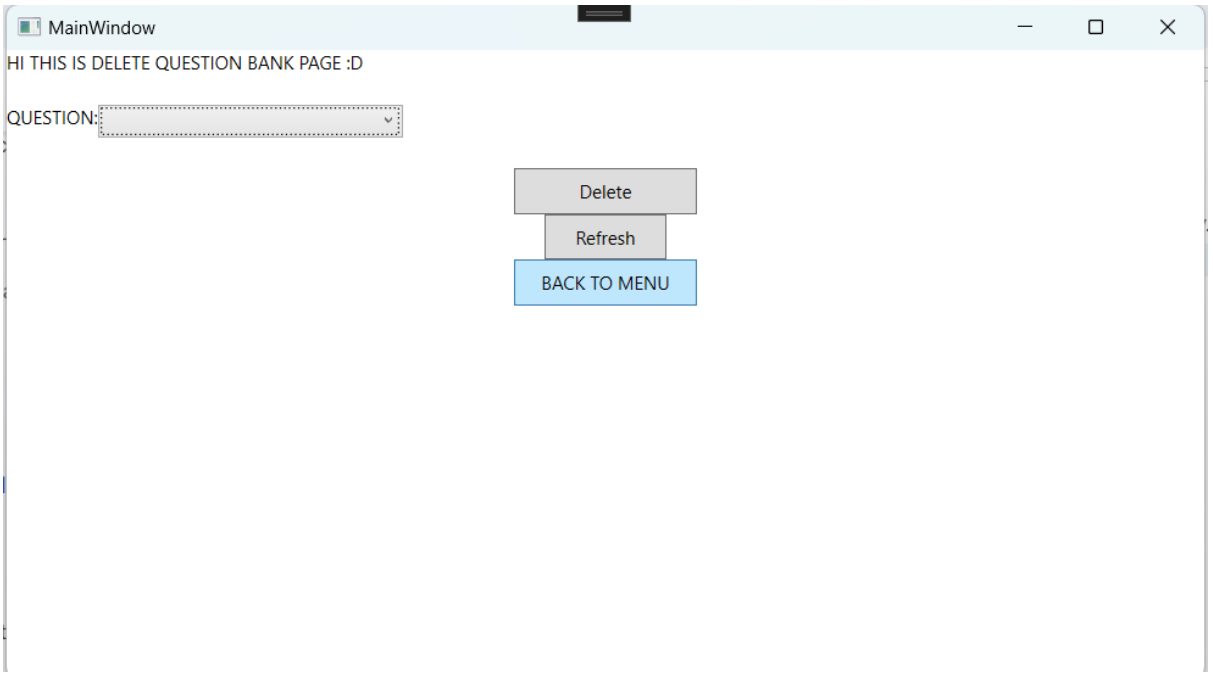


- QUESTION BANK DELETE PAGE: Combobox is empty.

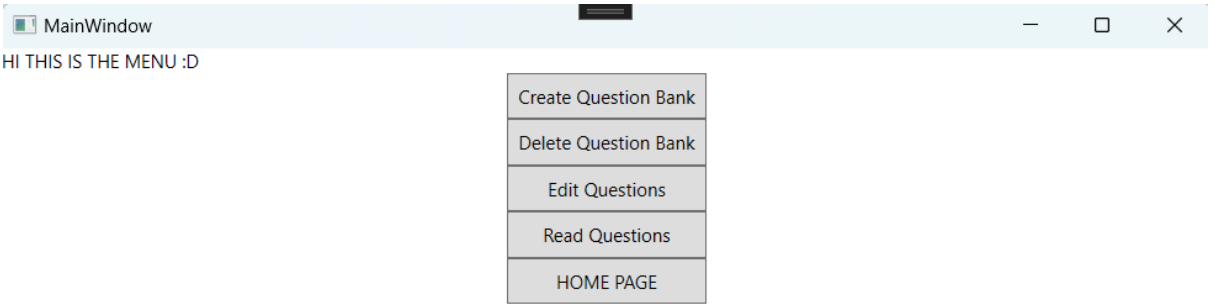


- After the Refresh Button is pressed, ComboBox is populated.

TEST 73:

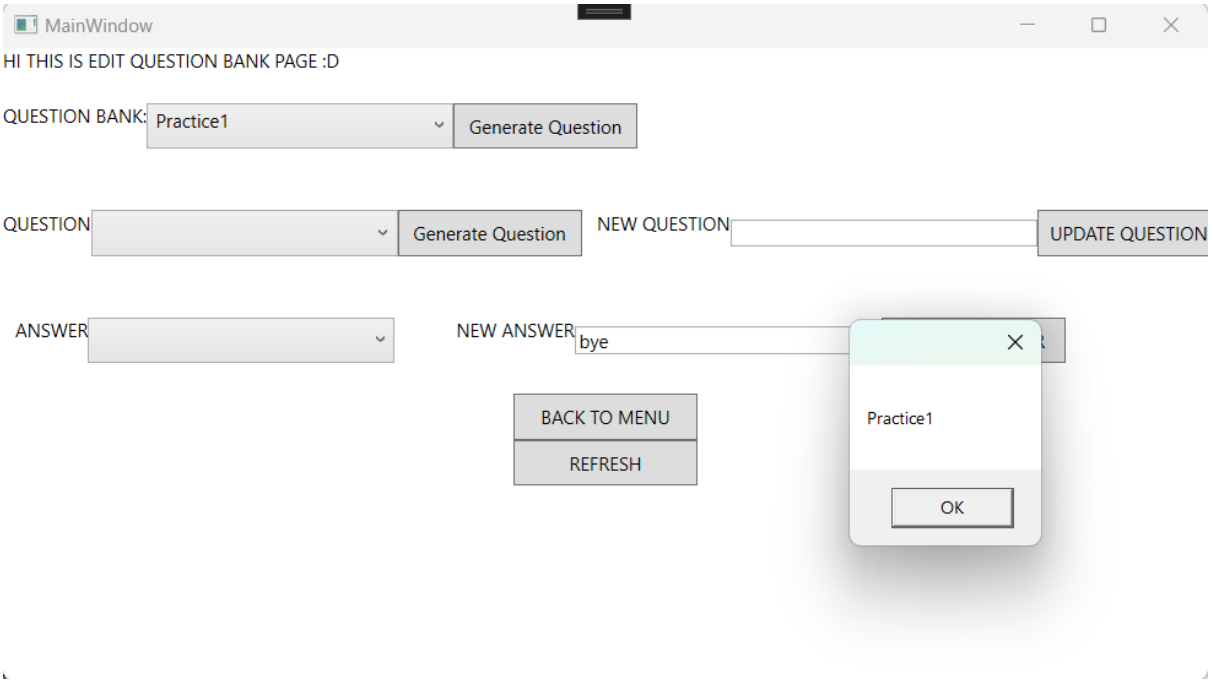


- QUESTION BANK DELETE PAGE: Back to Menu Button Pressed.



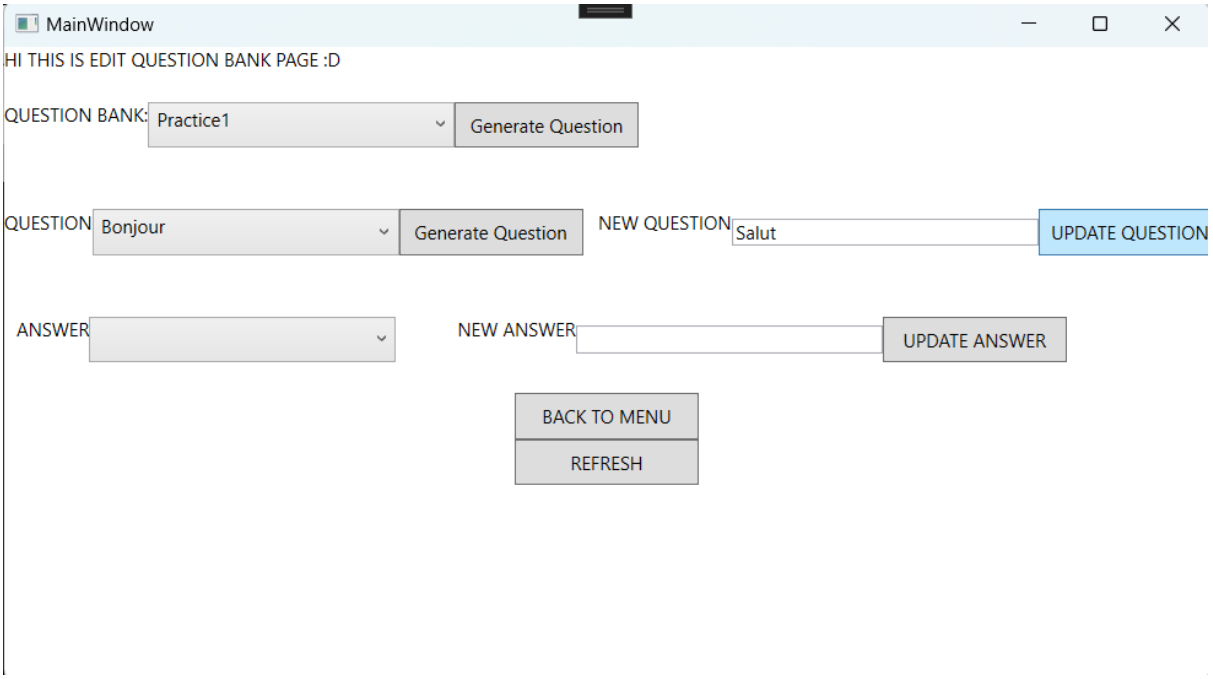
- Changes to the QUESTION BANK MENU PAGE.

TEST 74:



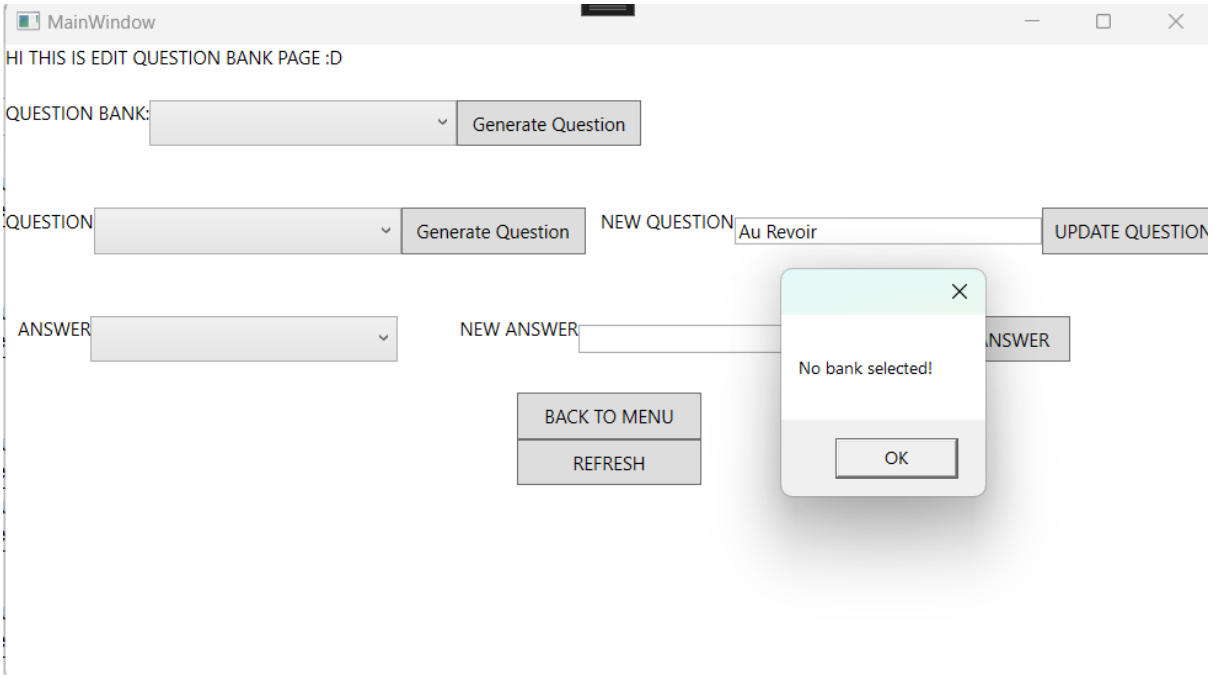
- QUESTION BANK EDIT PAGE: Message box that displays which question bank has been selected after the Generate Question Bank Button has been pressed.

TEST 75:



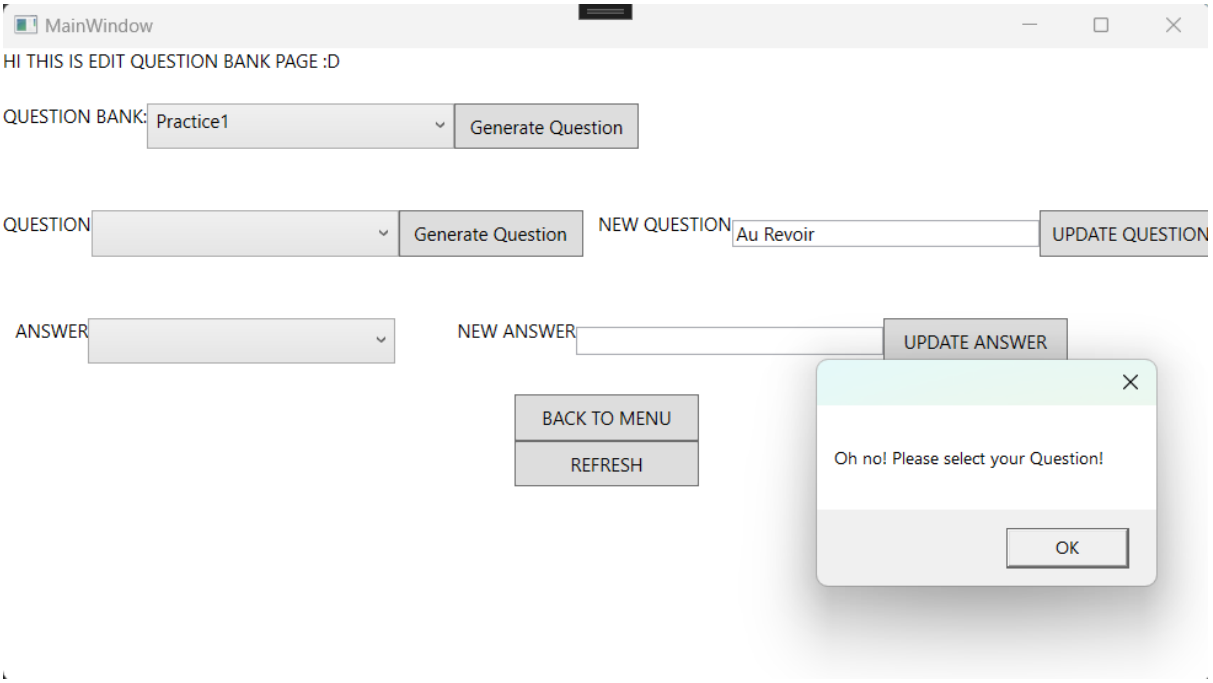
- QUESTION BANK EDIT PAGE: When Update Question Button is pressed, no error message appears.

TEST 76:



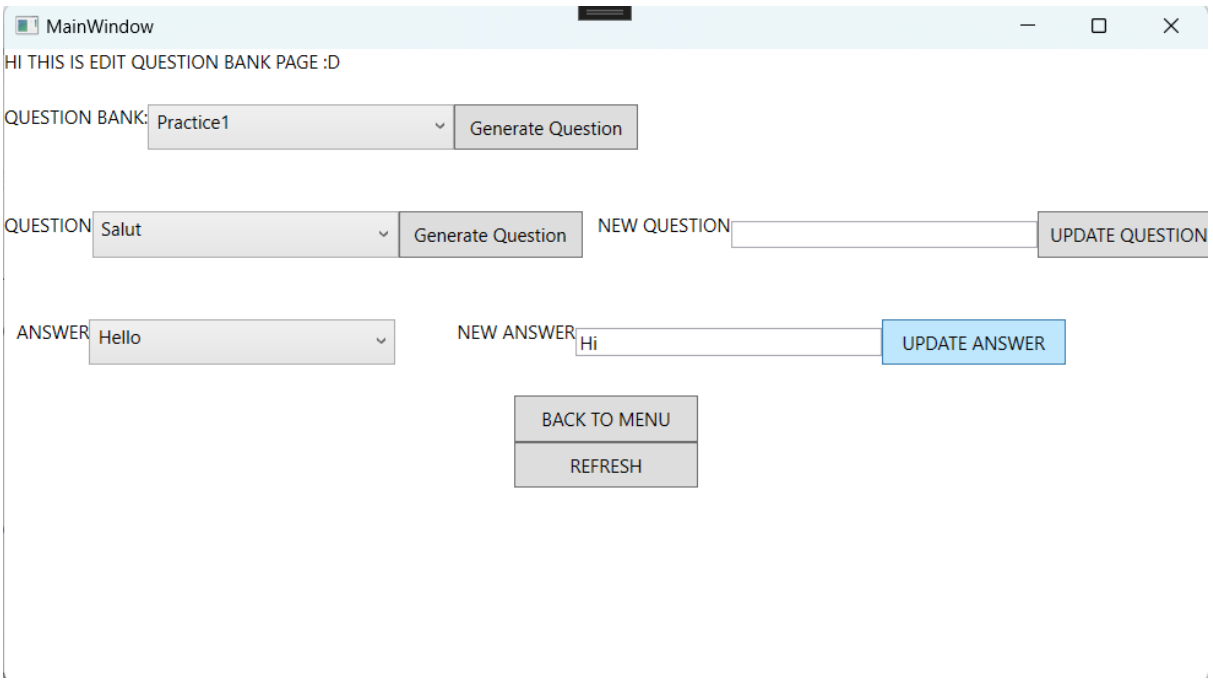
- QUESTION BANK EDIT PAGE: Pop-Up appears, when the user presses the Update Question button without selecting a bank.

TEST 77:



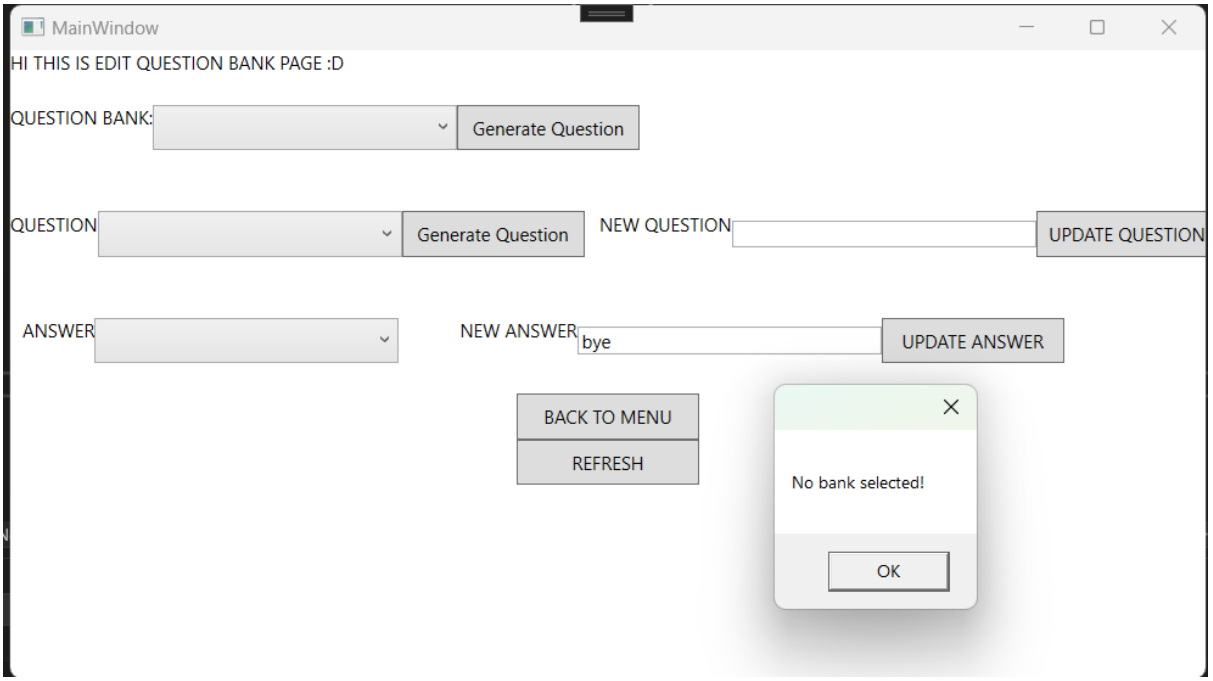
- QUESTION BANK EDIT PAGE: Pop-Up appears, when the user presses the Update Question button without selecting a Question.

TEST 78:



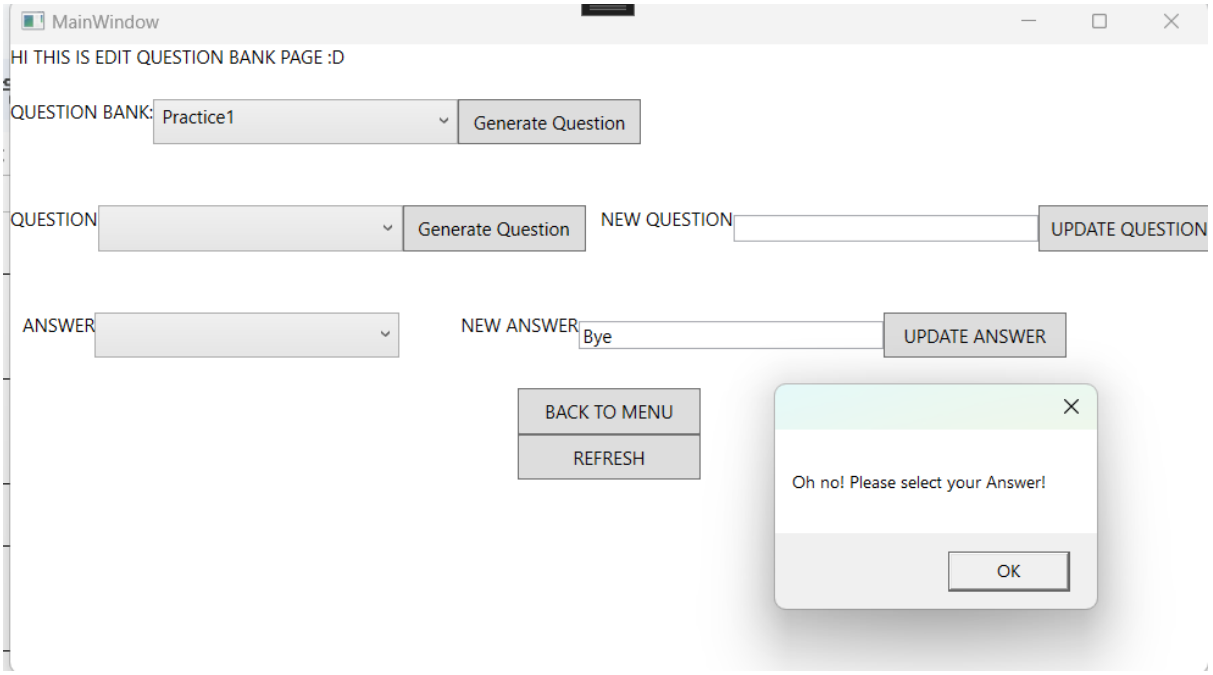
- QUESTION BANK EDIT PAGE: When Update Answer Button is pressed, no error message appears.

TEST 79:



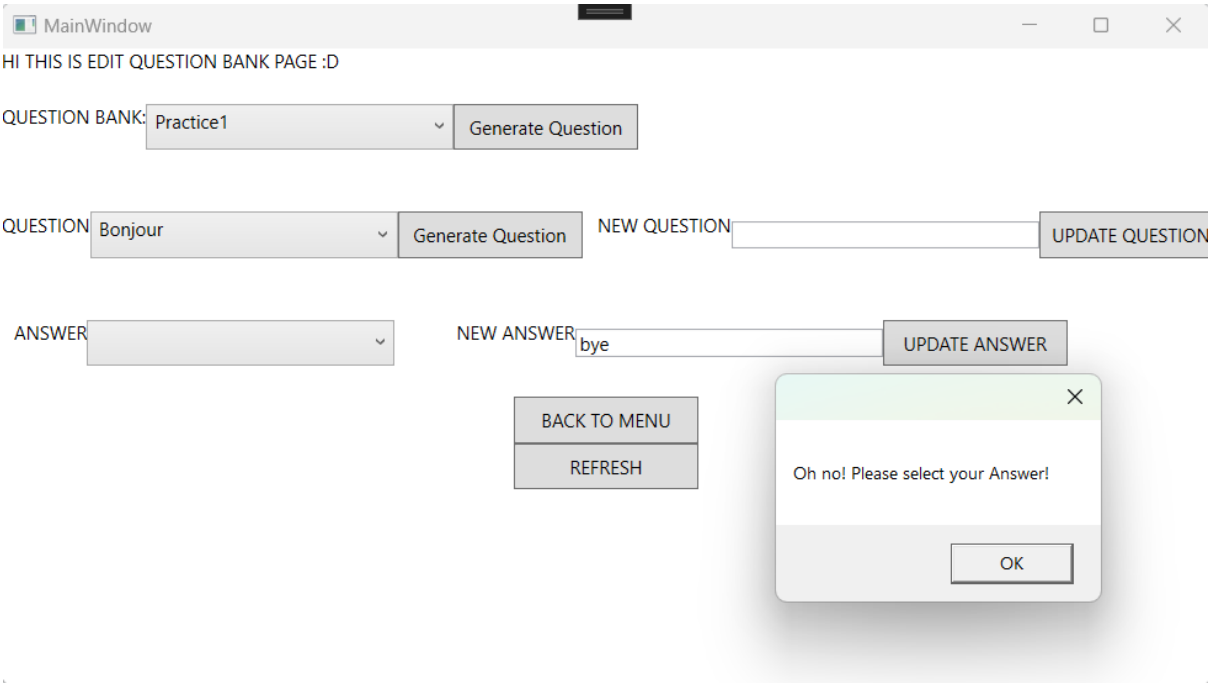
- QUESTION BANK EDIT PAGE: Pop-Up appears, when the user presses the Update Question button without selecting a bank.

TEST 80:



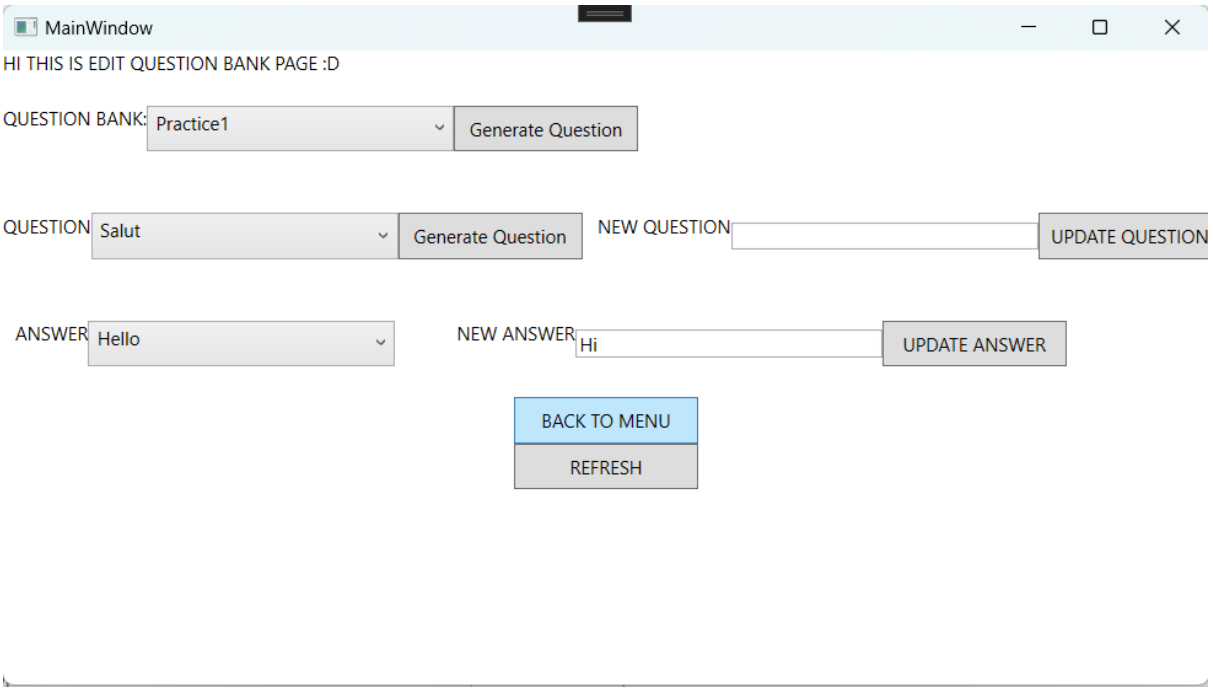
- QUESTION BANK EDIT PAGE: Pop-Up appears, when the user presses the Update Question button without selecting a question or answer.

TEST 81:

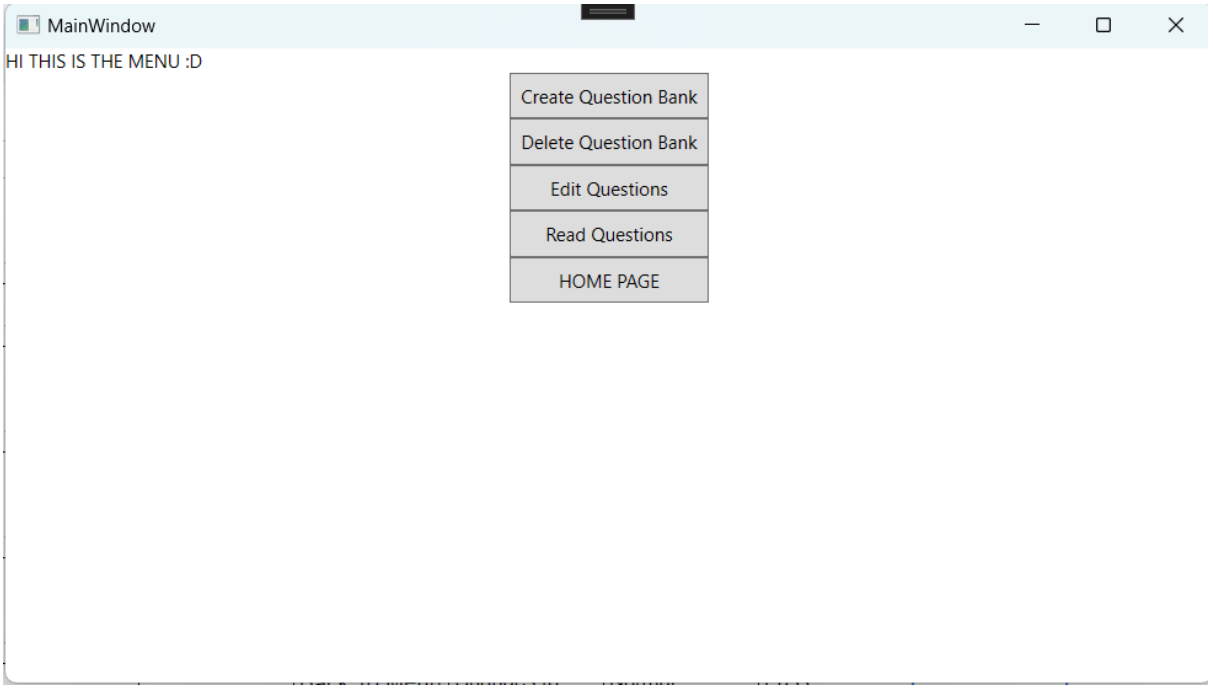


- QUESTION BANK EDIT PAGE: Pop-Up appears, when the user presses the Update Question button without selecting an answer.

TEST 82:

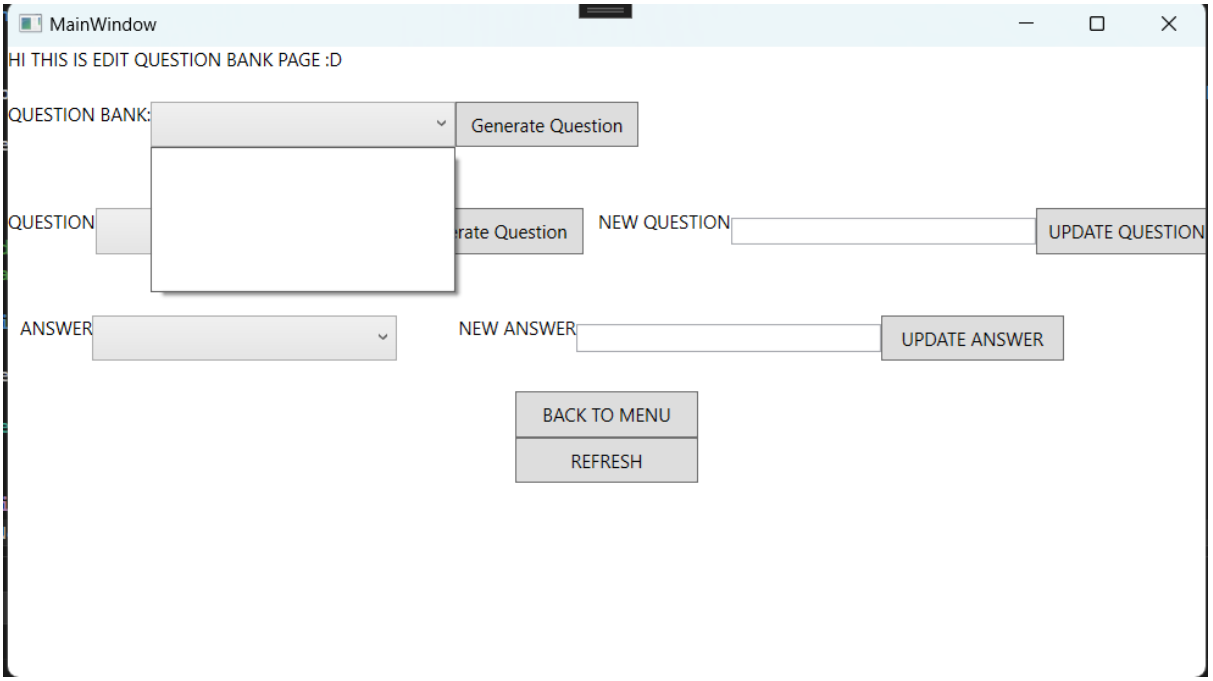


- QUESTION BANK EDIT PAGE: Back to Menu Button Pressed.

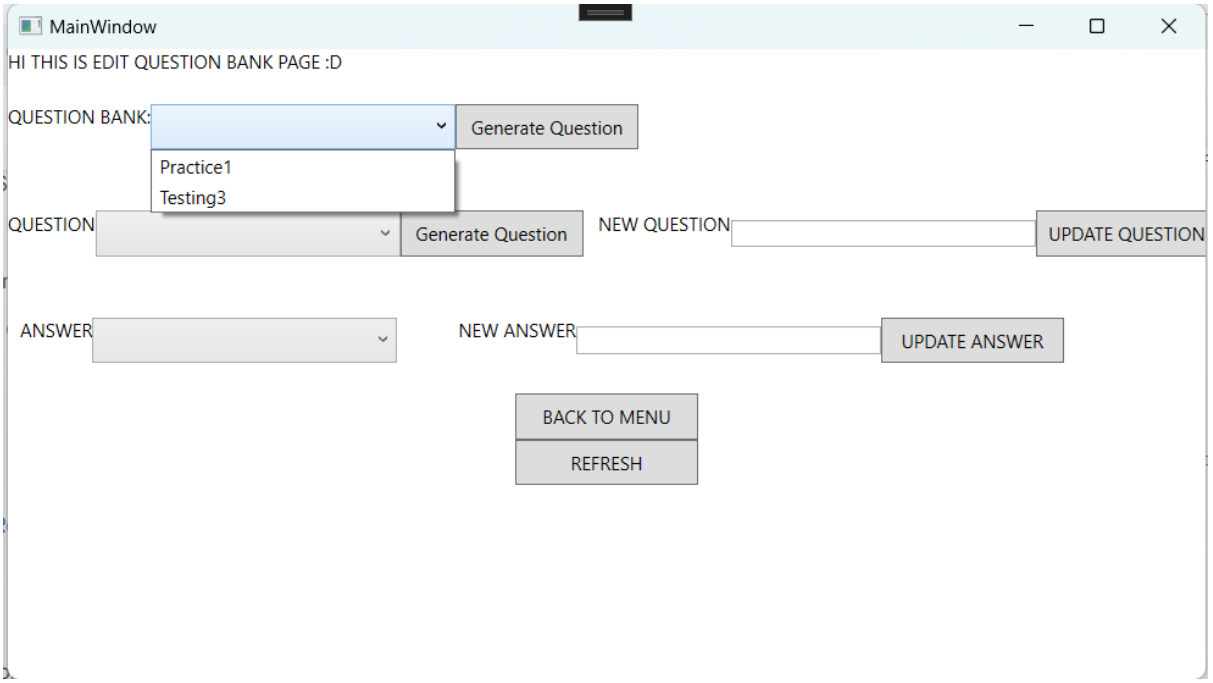


- Changes Page to the QUESTION BANK MENU PAGE.

TEST 83:

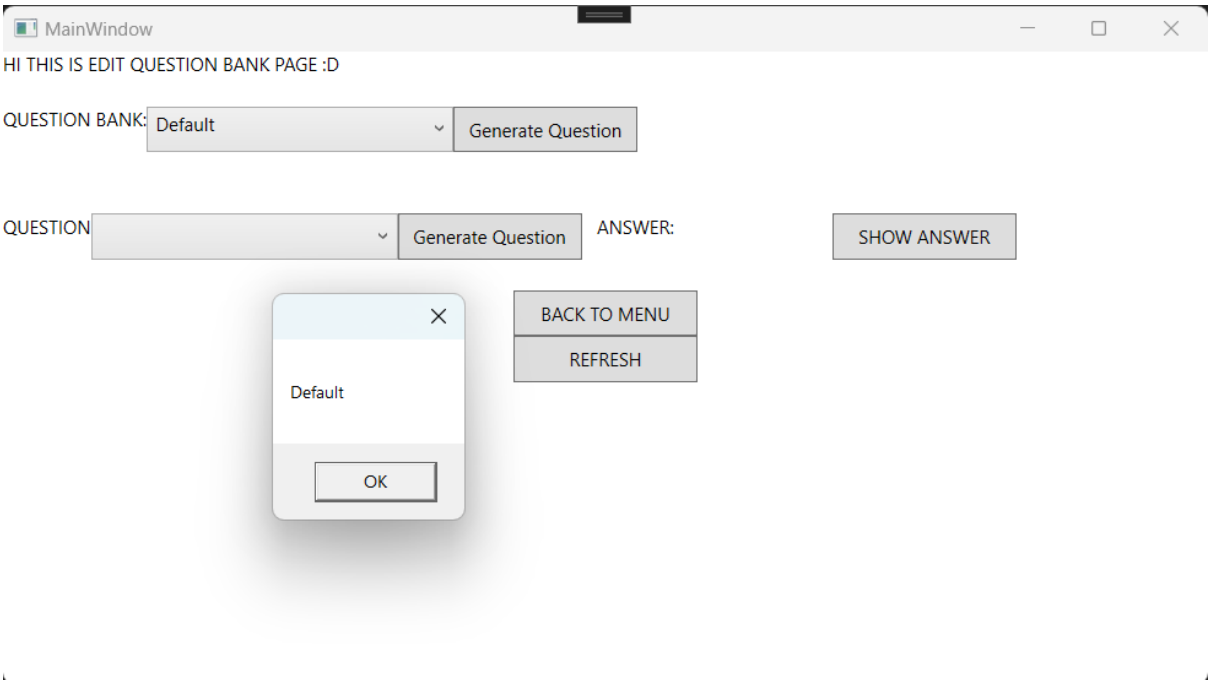


- QUESTION BANK EDIT PAGE: The ComboBox is empty.



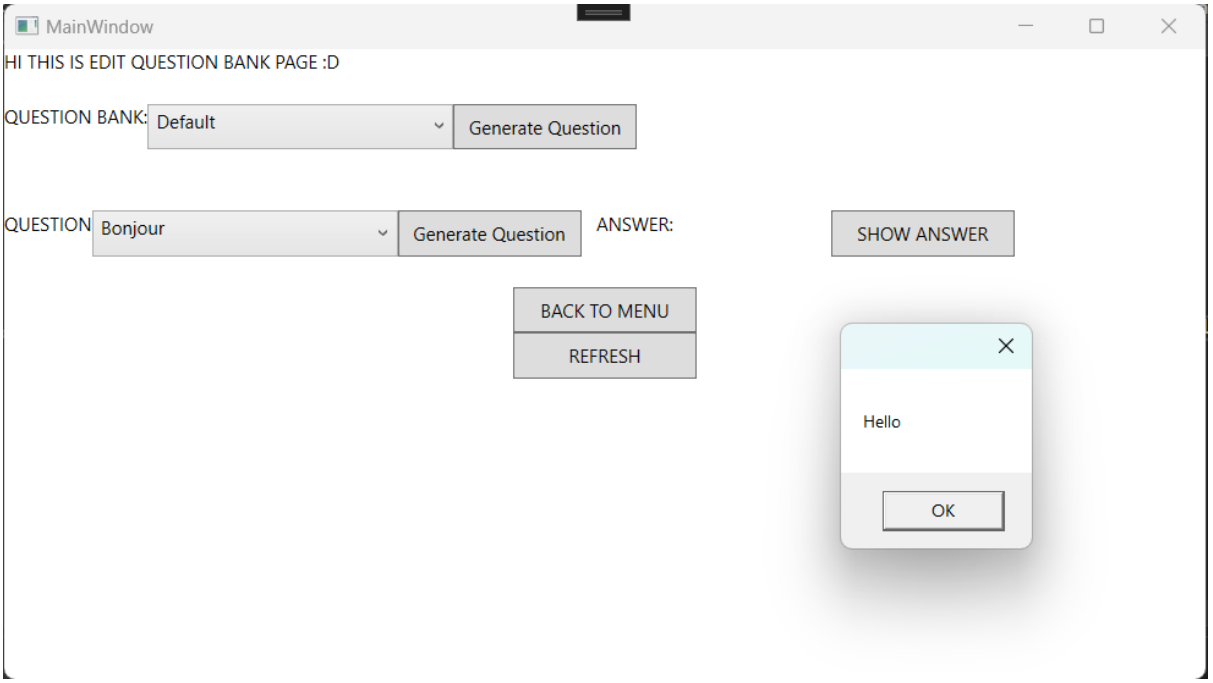
- Once the Refresh button is pressed, the ComboBox is populated.

TEST 84:



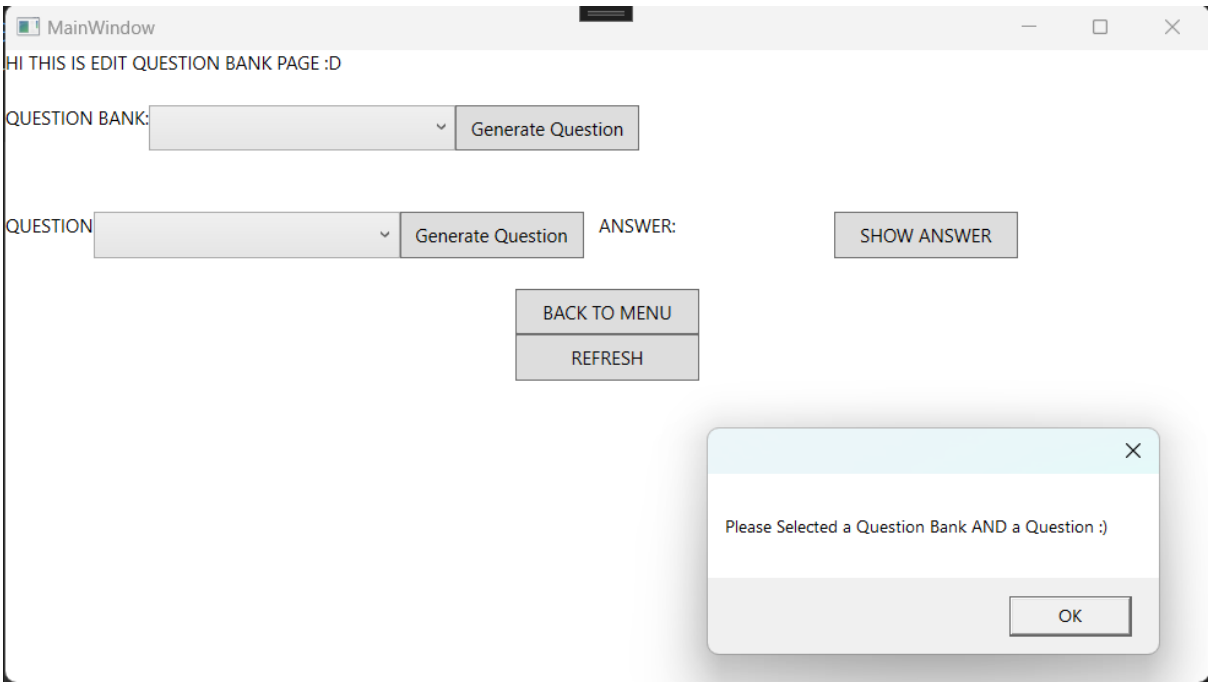
- QUESTION BANK EDIT PAGE: Message box that displays which question bank has been selected after the Generate Question Bank Button has been pressed.

TEST 85:



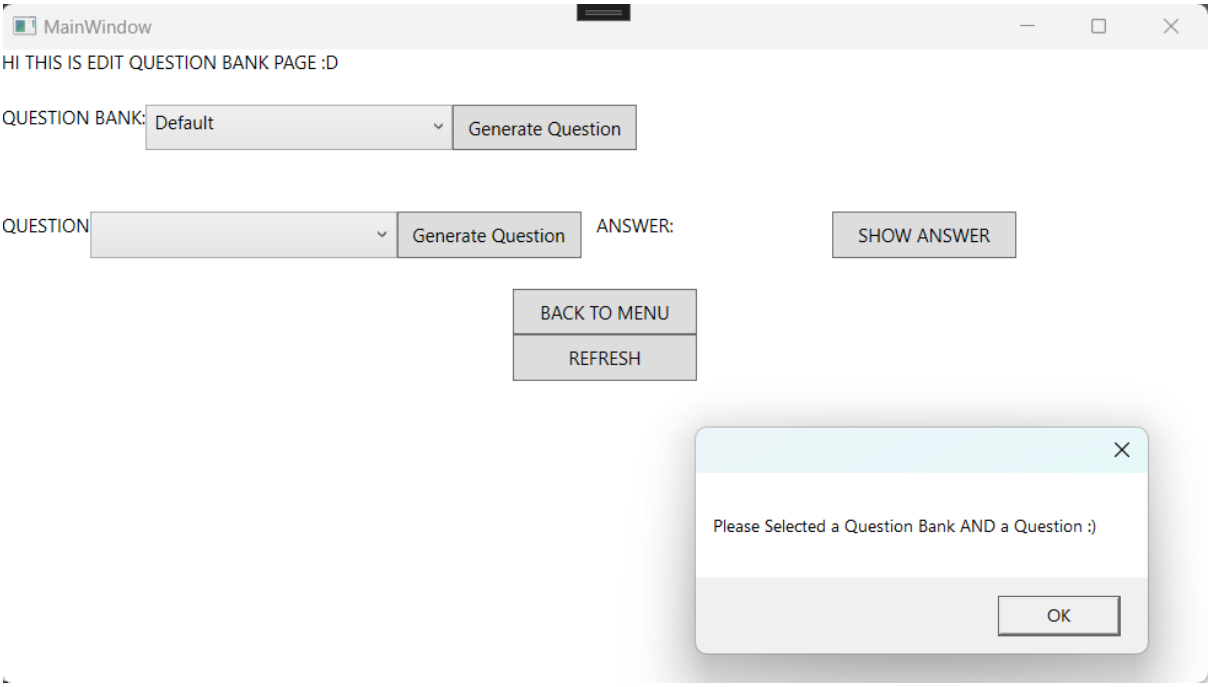
- QUESTION BANK EDIT PAGE: Once the Question Bank and Question has been selected and the Show Answer button pressed, a pop-up appears with the answer on screen.

TEST 86:



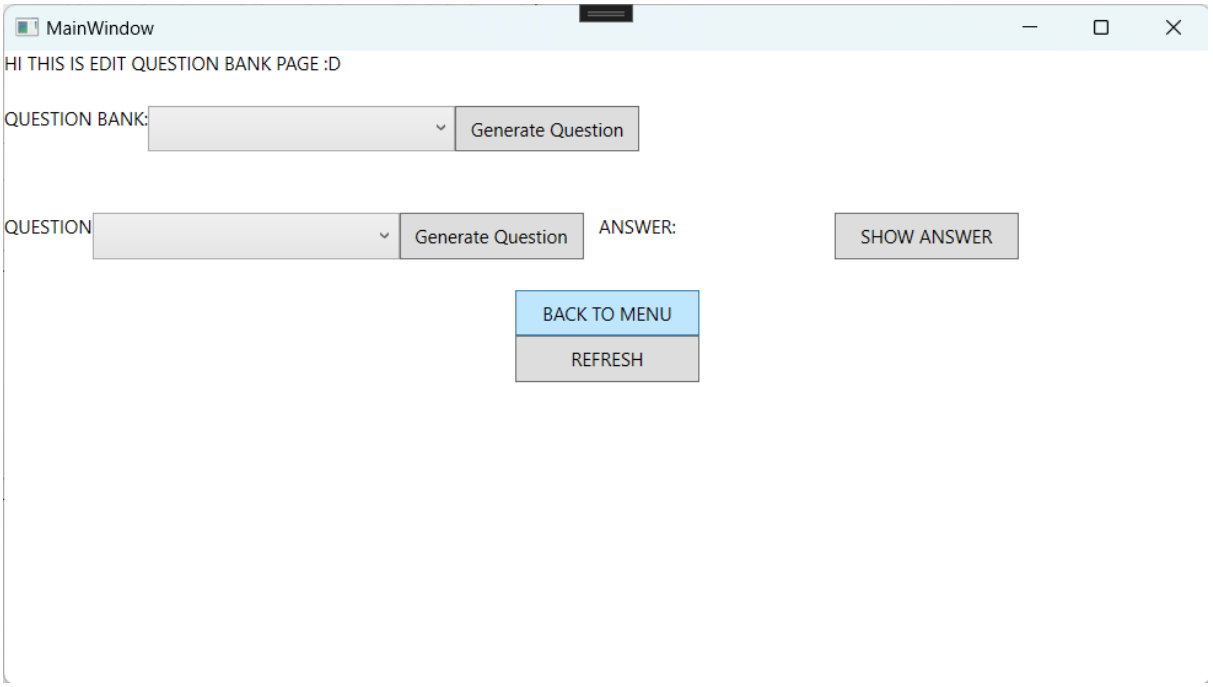
- QUESTION BANK READ PAGE: When the Show Answer button is pressed, and no question bank is selected, an error message appears.

TEST 87:

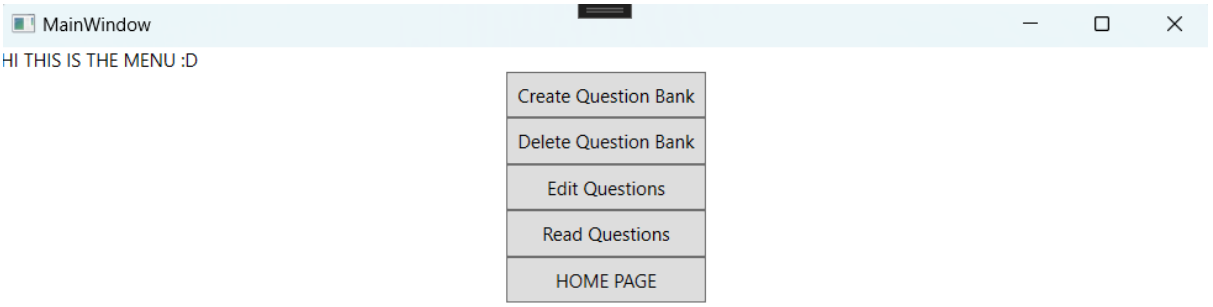


- QUESTION BANK READ PAGE: When the Show Answer button is pressed, and no question is selected, an error message appears.

TEST 88:

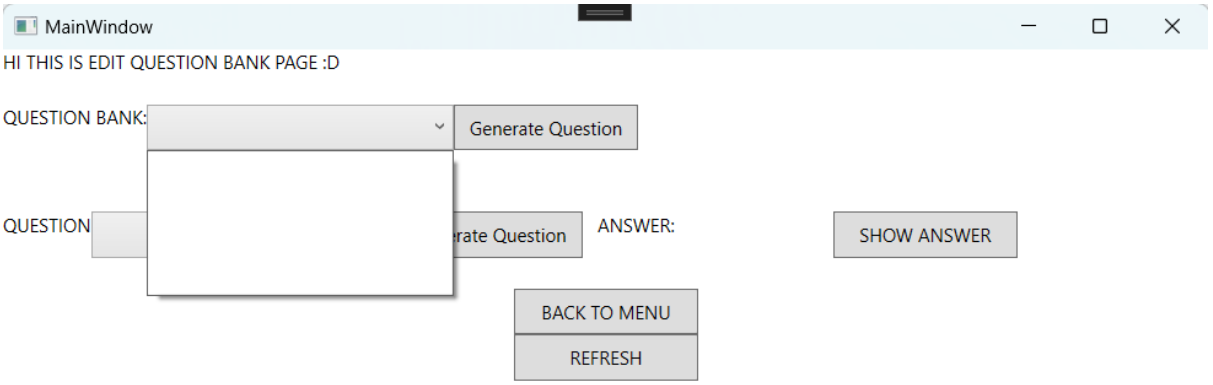


- QUESTION BANK READ PAGE: Back to Menu Button pressed

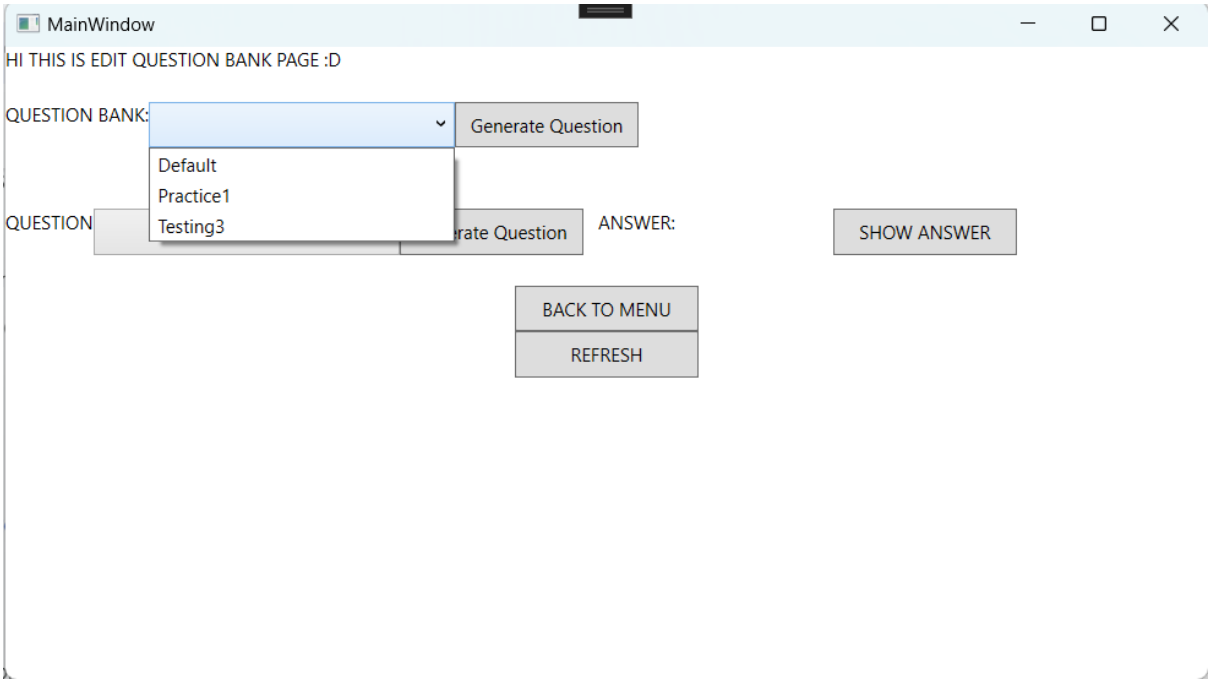


- Page is changed to the QUESTION BANK MENU PAGE.

TEST 89:

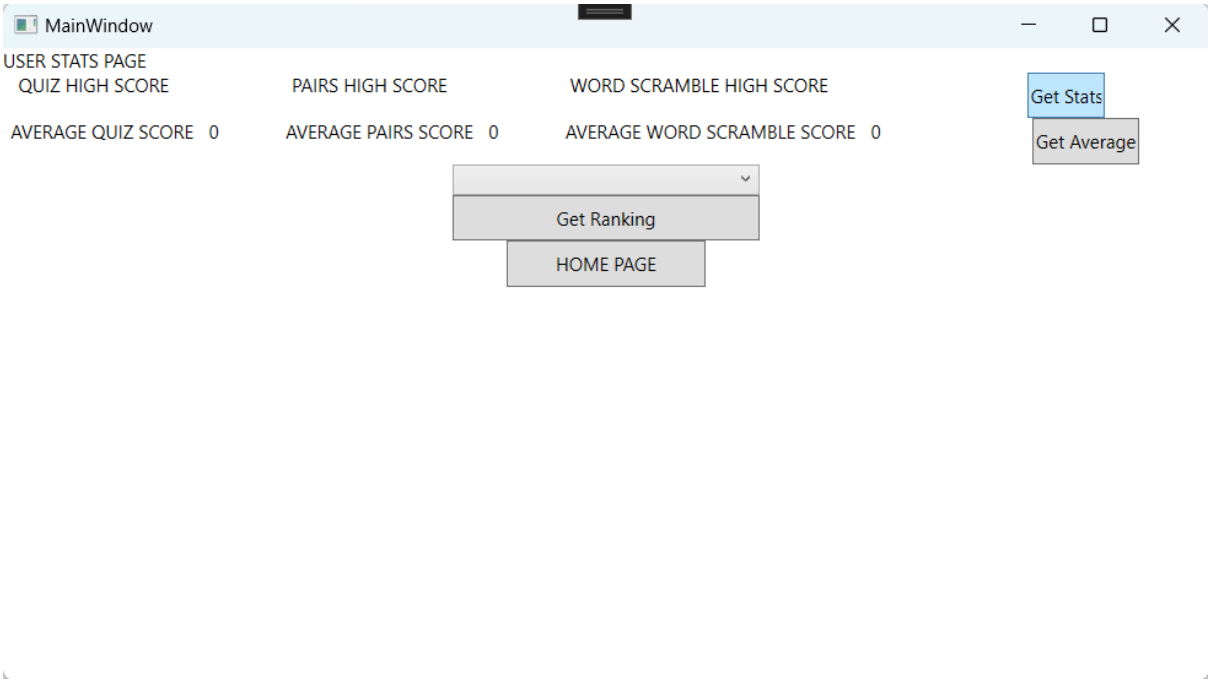


- QUESTION BANK READ PAGE: ComboBox is empty.

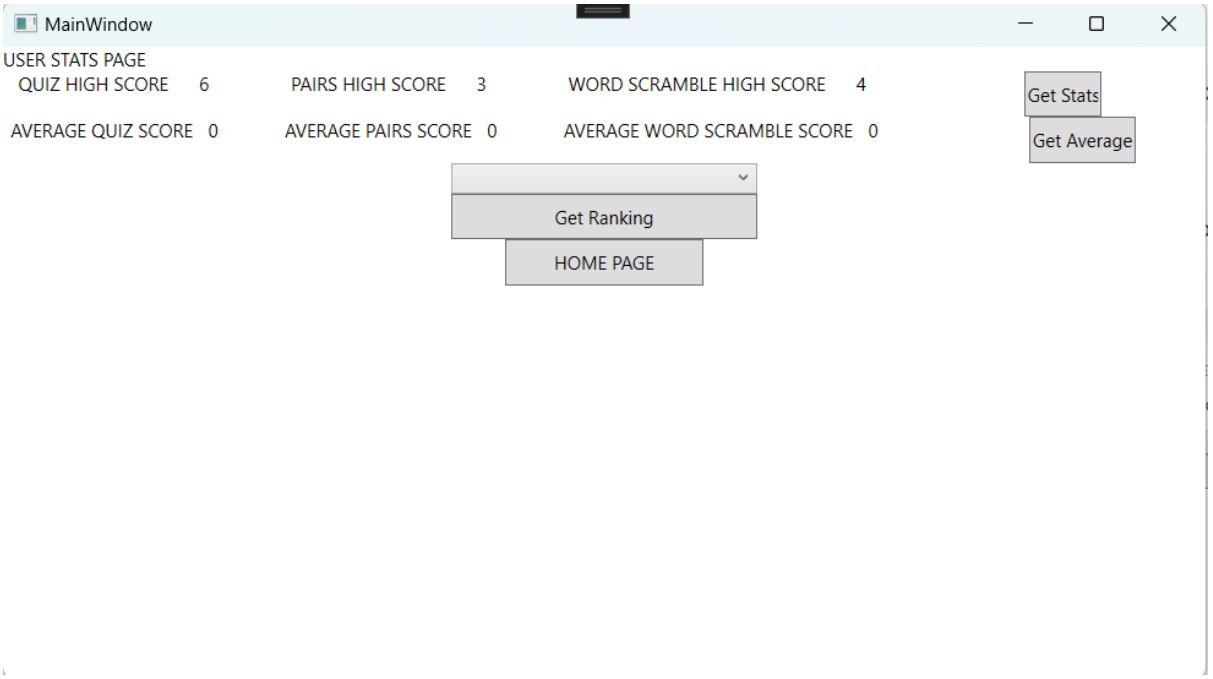


- When the Refresh button is pressed, the ComboBox is populated.

TEST 90:

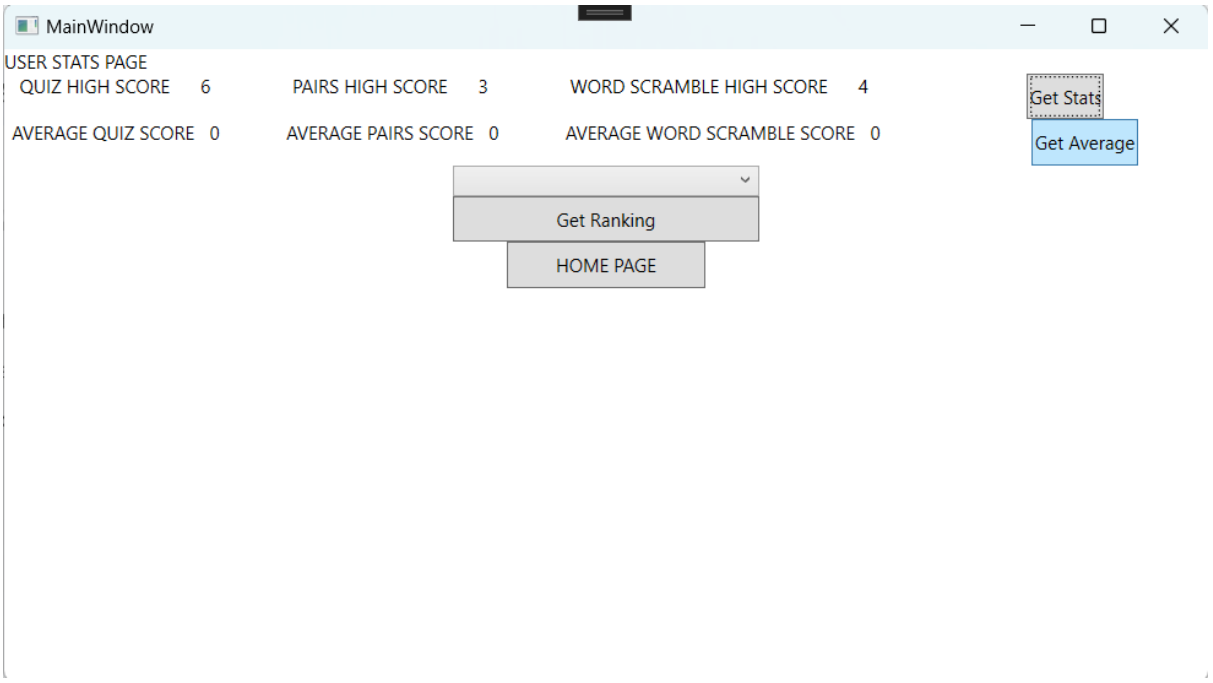


- USER STATS PAGE: Get Stats Button is pressed.

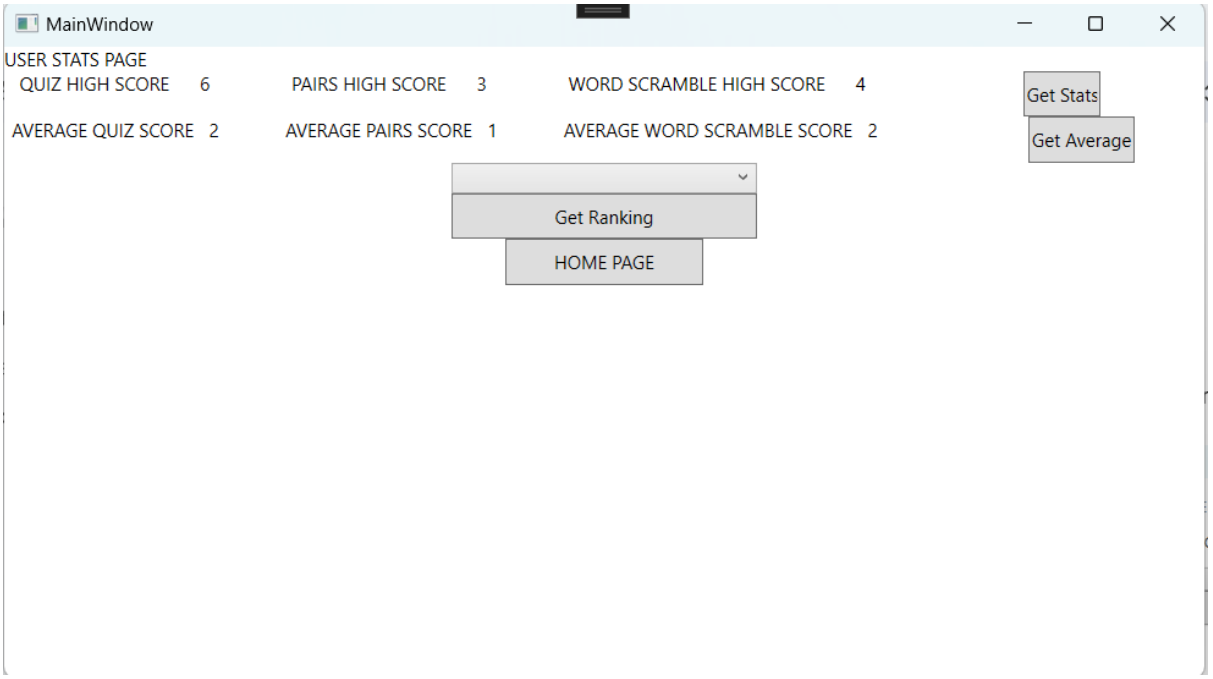


- The Highest scores for the user in each game are displayed.

TEST 91:

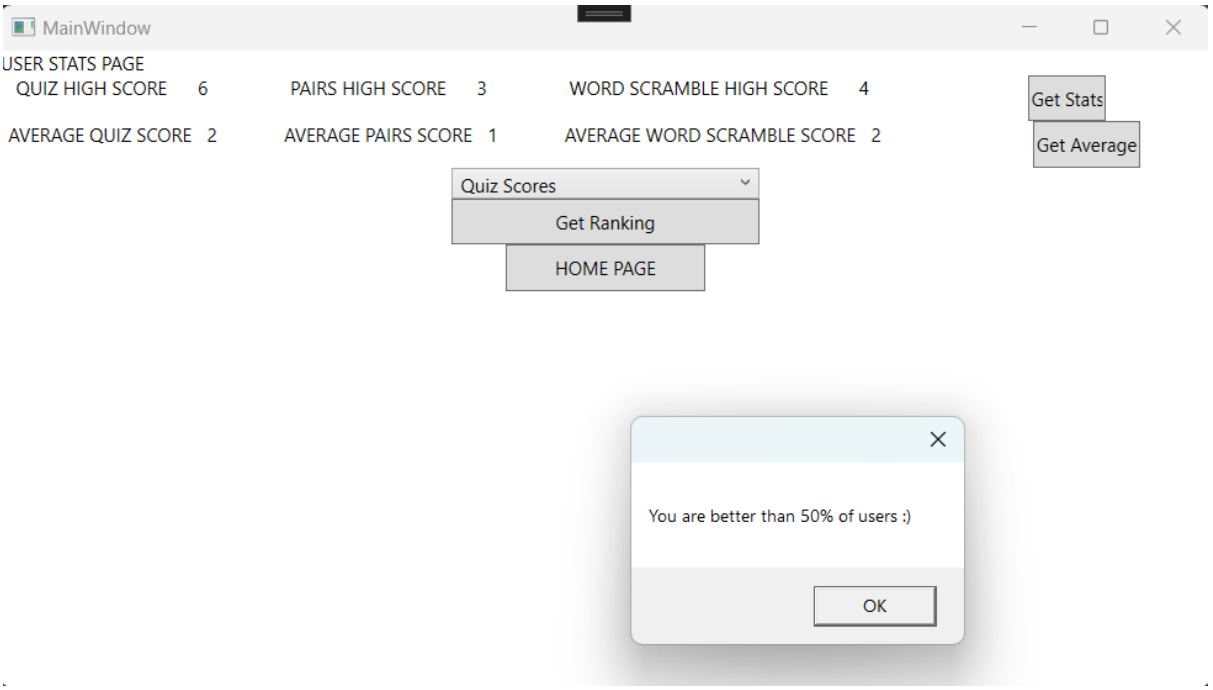


- USER STATS PAGE: GET Average Button is pressed.



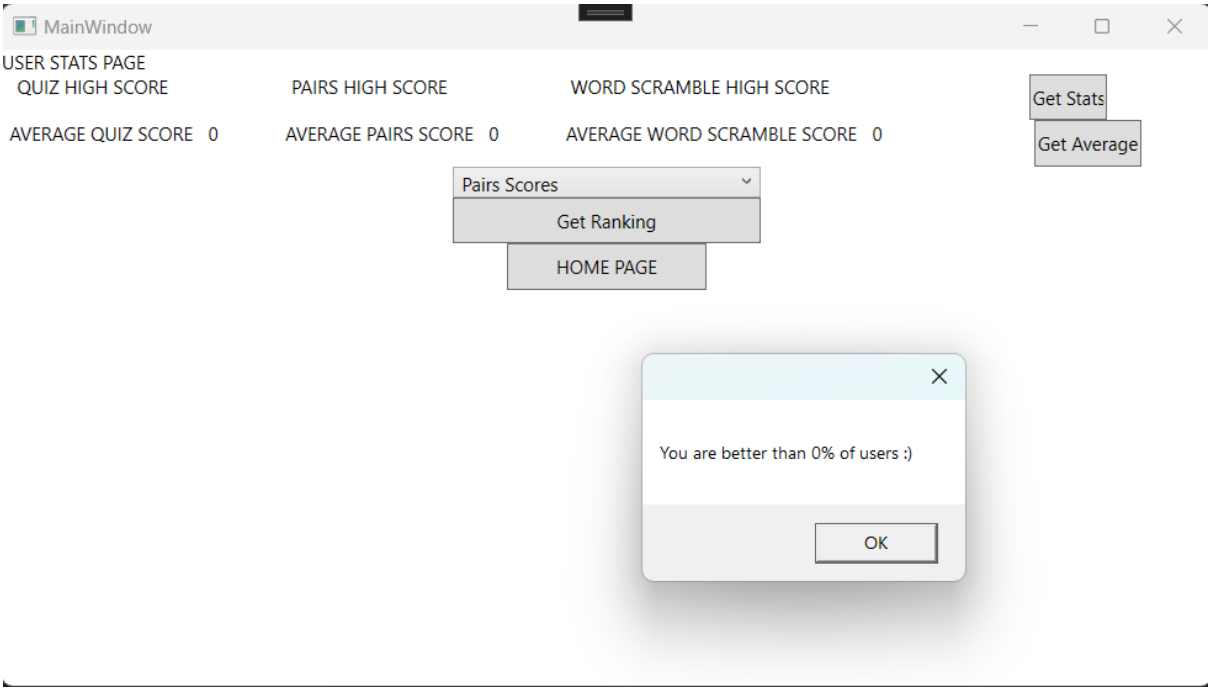
- The Mean average of each score for the user is displayed.

TEST 92:



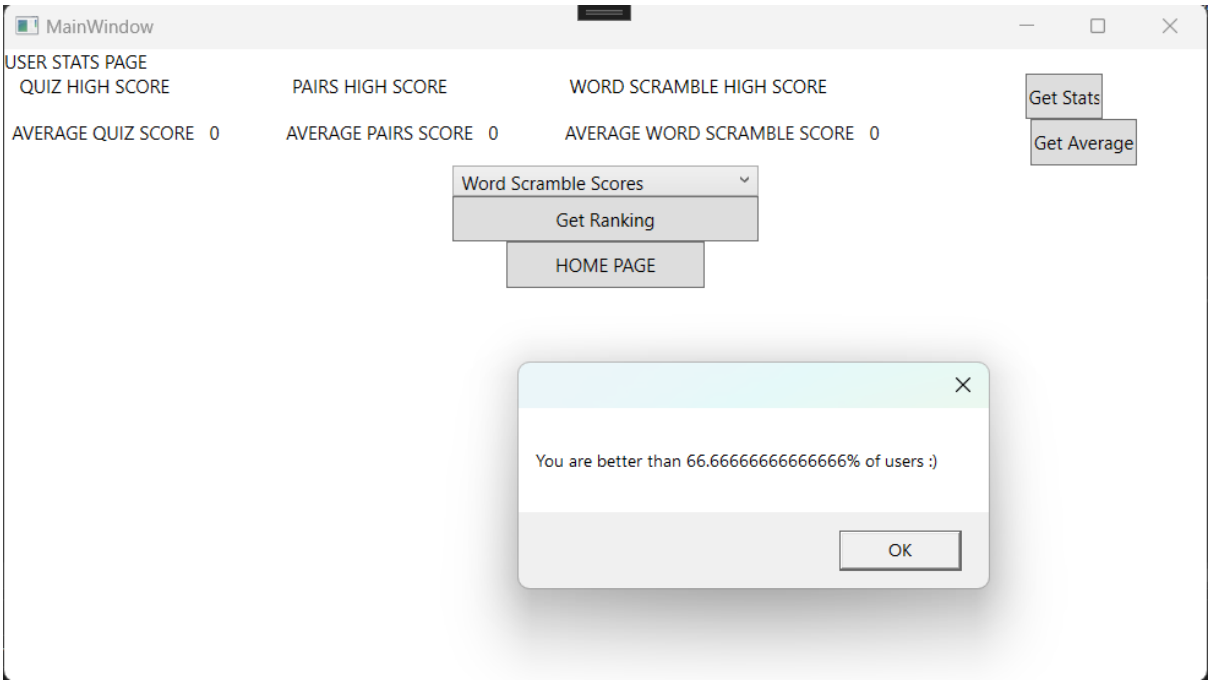
- USER STATS PAGE: When the Get Ranking Button is clicked and the Quiz Score category (any valid category) is selected.

TEST 93:



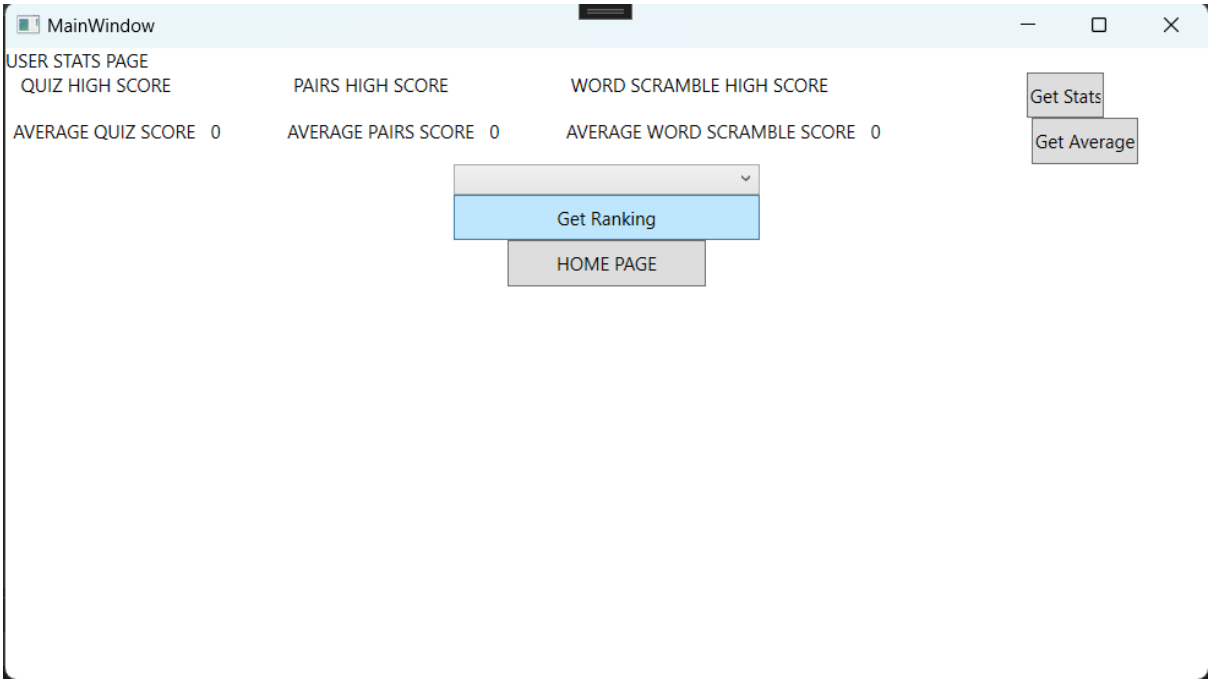
- USER STATS PAGE: When the Get Ranking Button is clicked and the Pairs Score category (any valid category) is selected.

TEST 94:

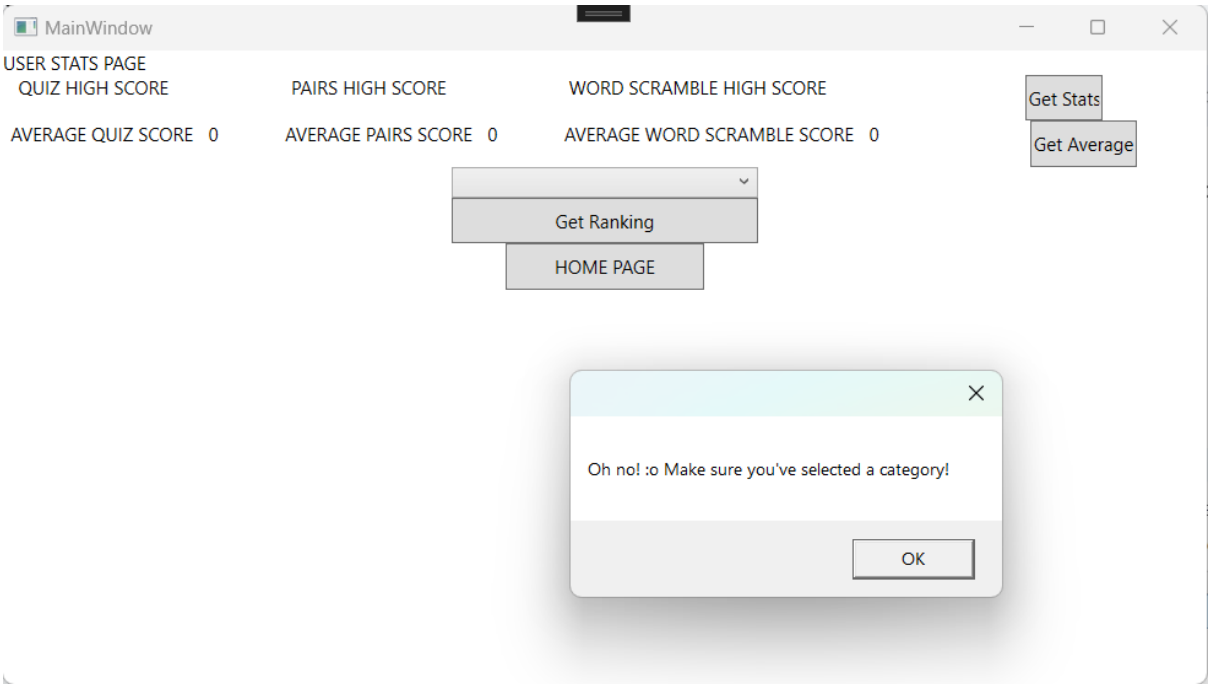


- USER STATS PAGE: When the Get Ranking Button is clicked and the Word Scramble Score category (any valid category) is selected.

TEST 95:

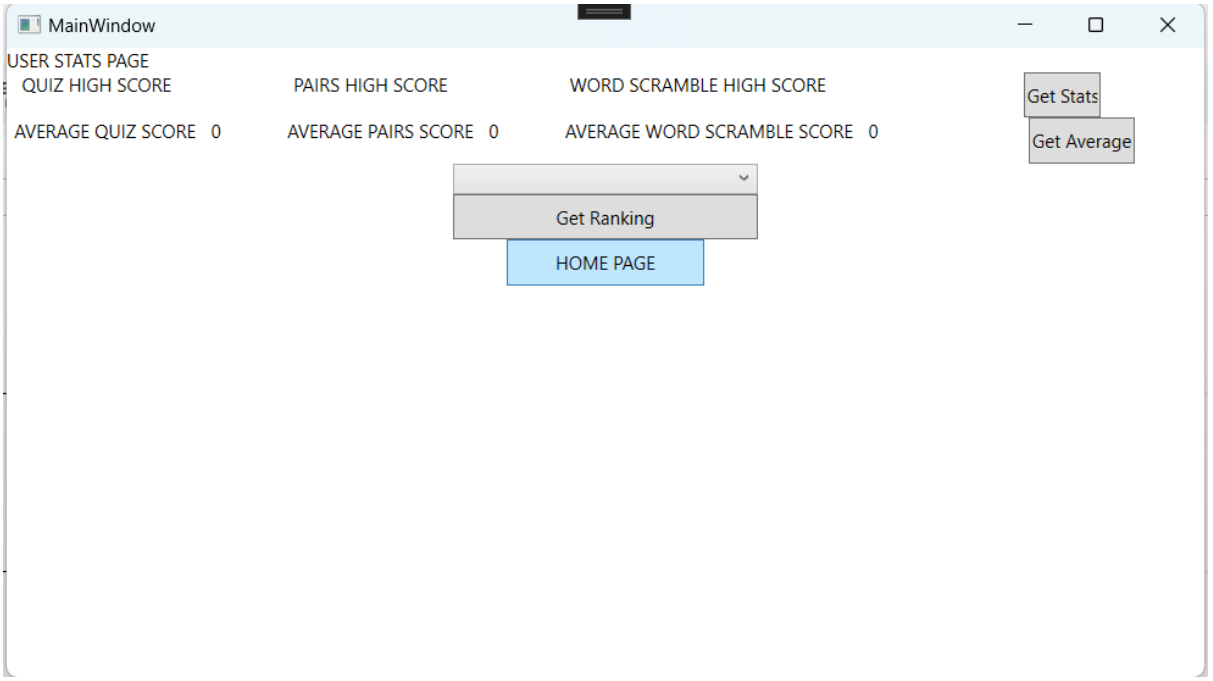


- USER STATS PAGE: Get Ranking Button is pressed without any category selected in ComboBox.

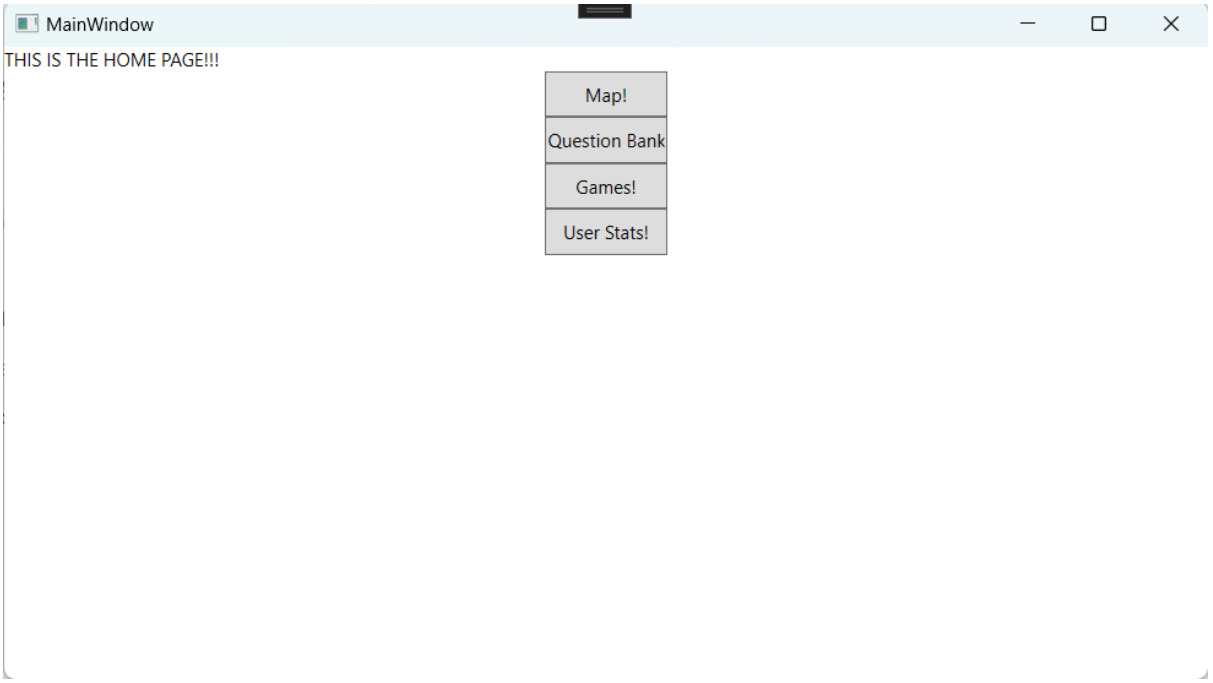


- Pop-up appears to remind the user to select a category first.

TEST 96:



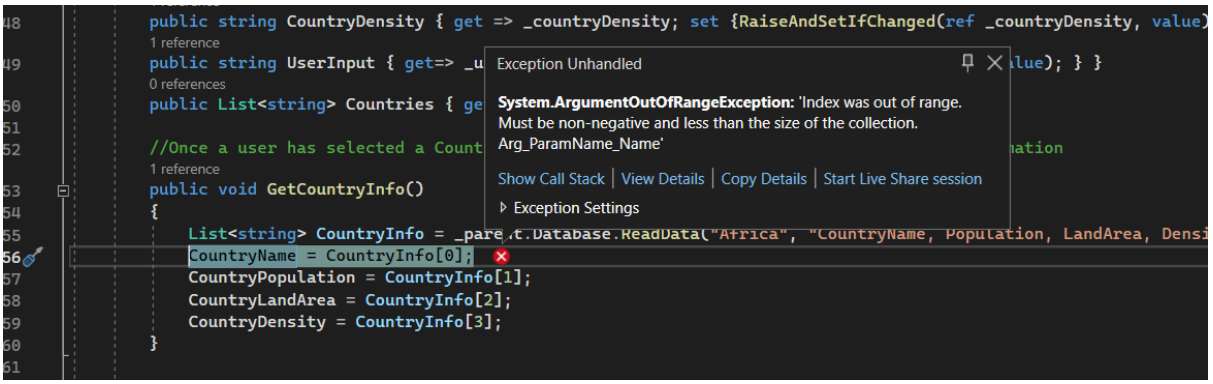
- USER STATS PAGE: Home Button is pressed.



- Changes to Home Page.

Failed Tests

TEST 22, 25, 28, 31, 34, 37:



Error:
An Exception is thrown because CountryInfo.Count is 0, because the SQL SELECT function returned nothing. This means that there needs to be a way to manage a user input of a country that does not exist.

```
CountryName = CountryInfo[0];
CountryPopulation = CountryInfo[1];
CountryLandArea = CountryInfo[2];
CountryDensity = CountryInfo[3];
```

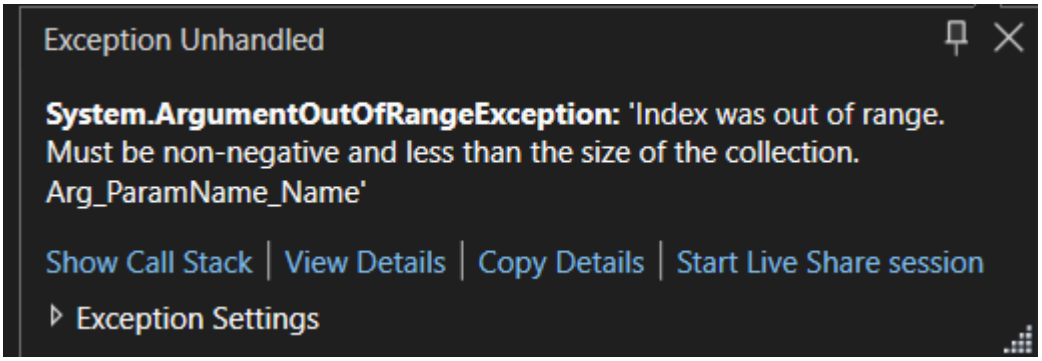
Fixed:

```
public void GetCountryInfo()
{
    List<string> CountryInfo = _parent.Database
    if (CountryInfo.Count > 0)
    {
        CountryName = CountryInfo[0];
        CountryPopulation = CountryInfo[1];
        CountryLandArea = CountryInfo[2];
        CountryDensity = CountryInfo[3];
    }

}
```

An IF statement is implemented to ensure that the program doesn't crash, only allowing the CountryInfo's data to be assigned IF it is not null.

TEST 48:

Error:



Much like the issue with searching the countries, the program would crash due to attempting to access an empty list:

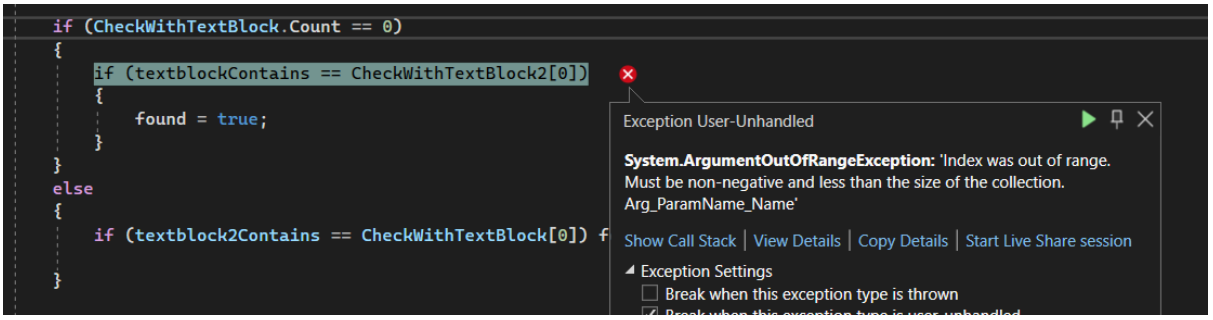
```
string CorrectAnswer = _parent.Database.ReadData("QuestionBanks", "Answer", $"Question LIKE '{Question}' AND BankName = '{_parent.CurrentQuestionBank}' AND (UserID = {_parent.UserID} OR UserID = 0)",1)[0];
```

This occurs if the user attempts to check their answer without generating a question first on both the QUIZ PAGE.

Fixed:

```
if (CorrectAnswer.Count > 0) {
        if (answer == CorrectAnswer[0].Trim())
        {
        MessageBox.Show("yay!");
        Score += 1;
        }
        else
        {
                MessageBox.Show($"neigh </3 {CorrectAnswer}");
        Score -= 1;
        }
}
else
{
        MessageBox.Show("Please generate a question first!!!");
}
```

TEST 52:



The problem lies in the fact that it only compares the text blocks with Answers NOT question, so if two Questions are compared, there is no way to handle this.

```
string textblock2Contains = _vm.TextBlock2Contains;

        List<string> CheckWithTextBlock = _parent.Database.ReadData("QuestionBanks", "Answer", $"Question LIKE '{_vm.TextBlockContains}' AND BankName = '{_parent.CurrentQuestionBank}' AND UserID = {ID}", 1);

        string textblockContains = _vm.TextBlockContains;

         List<string> CheckWithTextBlock2 = _parent.Database.ReadData("QuestionBanks", "Answer", $"Question LIKE '{_vm.TextBlock2Contains}' AND BankName = '{_parent.CurrentQuestionBank}' AND UserID = {ID}", 1);
```

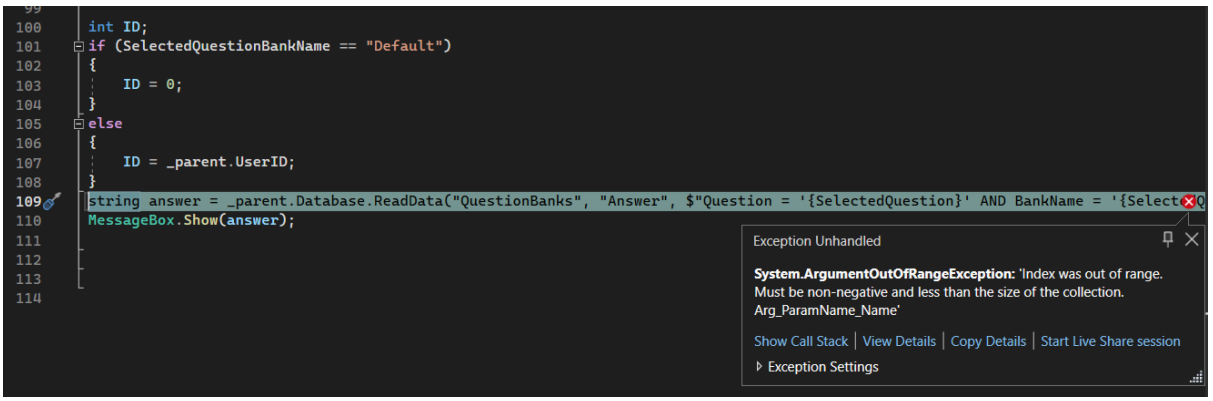
FIXED:

Though a messy solution, it solves the problem of two answers being paired and two questions being paired.

```
if (CheckWithTextBlock.Count == 0 && CheckWithTextBlock2.Count == 0)
        {
         List<string> CheckWithTextBlockQ = _parent.Database.ReadData("QuestionBanks", "Question", $"Answer LIKE '{_vm.TextBlockContains}' AND BankName = '{_parent.CurrentQuestionBank}' AND UserID = {ID}", 1);
         List<string> CheckWithTextBlock2Q = _parent.Database.ReadData("QuestionBanks", "Question", $"Answer LIKE '{_vm.TextBlockContains}' AND BankName = '{_parent.CurrentQuestionBank}' AND UserID = {ID}", 1);
         if (CheckWithTextBlockQ.Count == 0)
         {
                 if (textblockContains == CheckWithTextBlock2Q[0])
                 {
                 found = true;
                 }
         }
         else
         {
                 if (textblock2Contains == CheckWithTextBlockQ[0]) found = true;

         }
        }
```

```
else if (CheckWithTextBlock.Count == 0)
{
 if (textblockContains == CheckWithTextBlock2[0])
 {
            found = true;
 }
}
else
{
   if (textblock2Contains == CheckWithTextBlock[0]) found = true;

 }
```

TEST 86 & 87



Once again, there is an exception that is unhandled due to attempting an empty list, the fact that these errors are numerous, is no doubt due to the alteration of the SQLite ReadData method that saw the return type change from a String to List<String>.

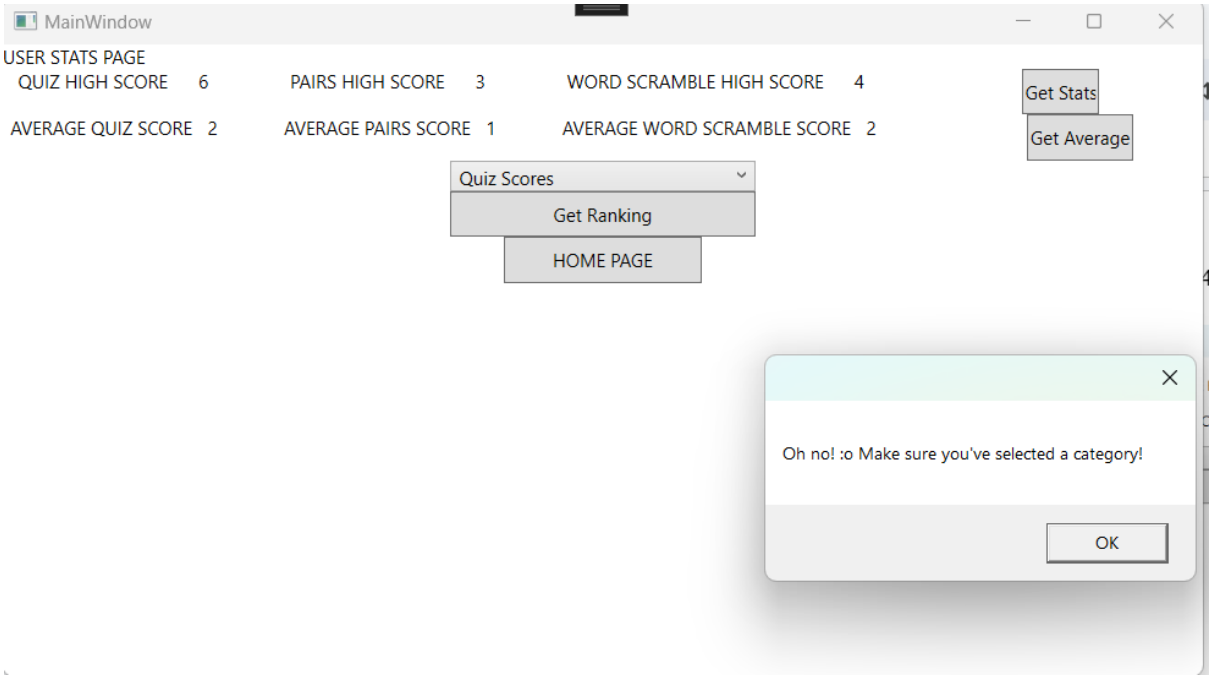
```
string answer = _parent.Database.ReadData("QuestionBanks", "Answer", $"Question =
'{SelectedQuestion}' AND BankName = '{SelectedQuestionBankName}' AND USERID = {ID}", 1)[0];
```

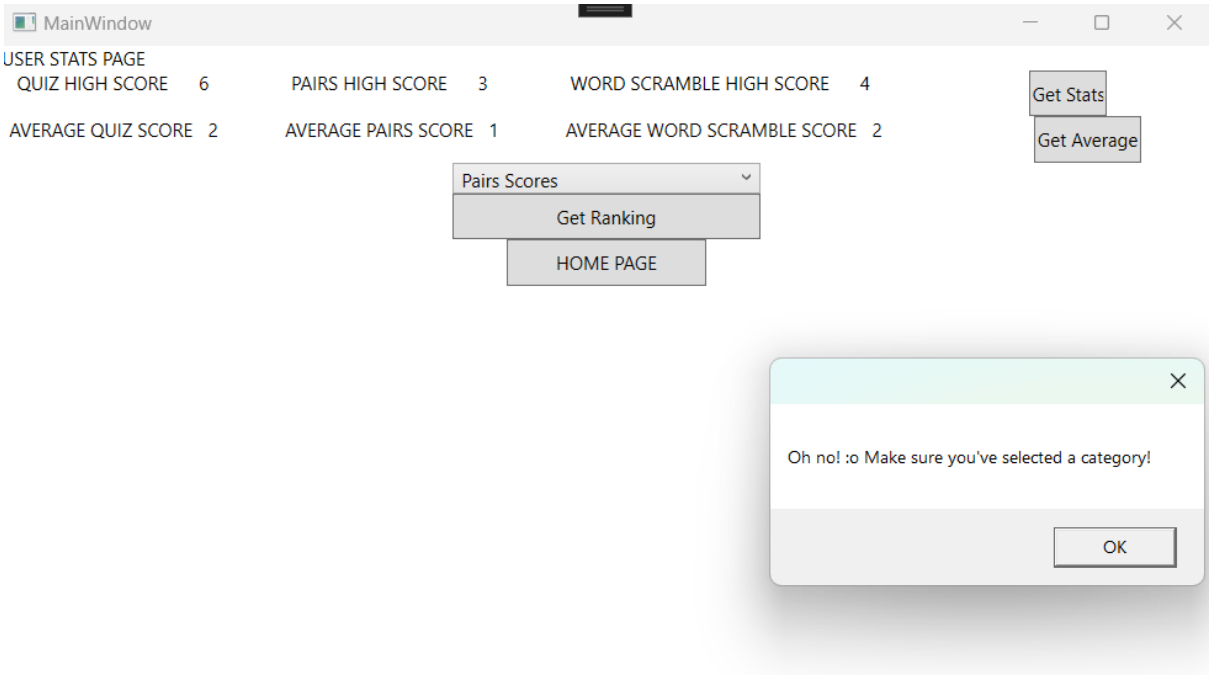
FIXED:

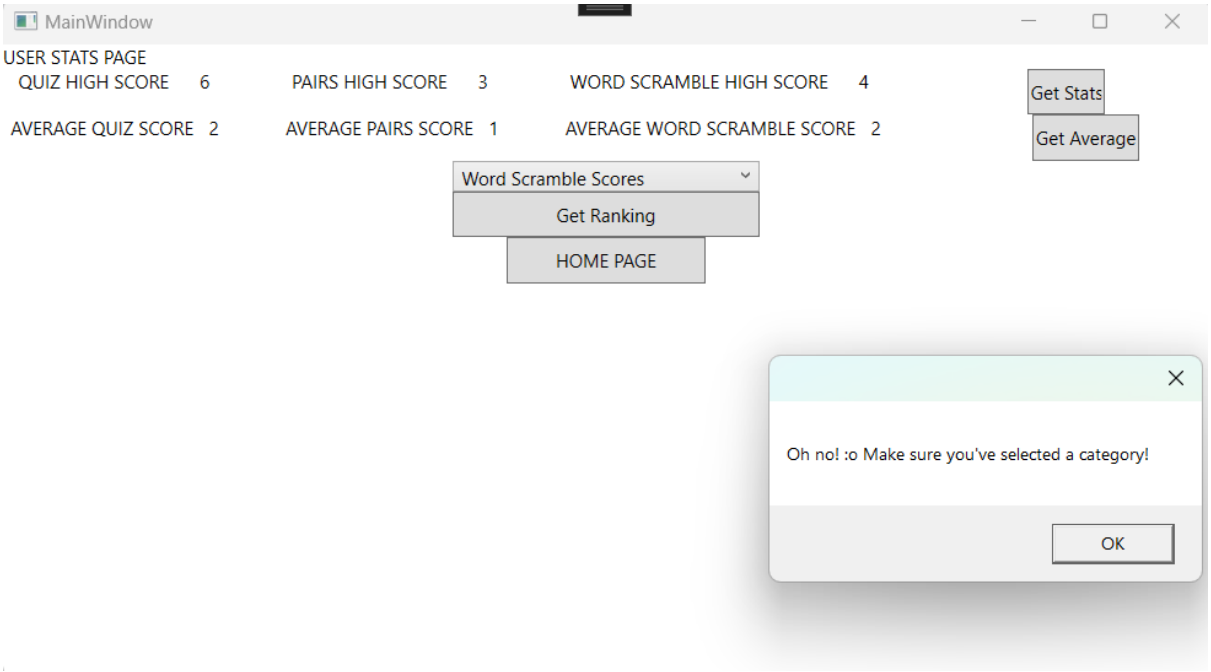
```
public void ShowAnswer()
    {
        int ID;
        if (SelectedQuestionBankName == "Default")
        {
          ID = 0;
        }
        else
        {
          ID = _parent.UserID;
        }
```

```
        List<string> GetAnswer = _parent.Database.ReadData("QuestionBanks", "Answer", $"Question =
'{SelectedQuestion}' AND BankName = '{SelectedQuestionBankName}' AND USERID = {ID}", 1);

        if (GetAnswer.Count > 0)

        {

         string answer = GetAnswer[0];

         MessageBox.Show(answer);

        }

        else

        {

         MessageBox.Show("Please Selected a Question Bank AND a Question :)");

        }


  }
```

TEST 92, 93, 94:


MainWindow
USER STATS PAGE
QUIZ HIGH SCORE 6
PAIRS HIGH SCORE 3
WORD SCRAMBLE HIGH SCORE 4
AVERAGE QUIZ SCORE 2
AVERAGE PAIRS SCORE 1
AVERAGE WORD SCRAMBLE SCORE 2
Get Stats
Get Average
Quiz Scores
Get Ranking
HOME PAGE
Oh no! :o Make sure you've selected a category!
OK


MainWindow
USER STATS PAGE
QUIZ HIGH SCORE 6
PAIRS HIGH SCORE 3
WORD SCRAMBLE HIGH SCORE 4
AVERAGE QUIZ SCORE 2
AVERAGE PAIRS SCORE 1
AVERAGE WORD SCRAMBLE SCORE 2
Get Stats
Get Average
Pairs Scores
Get Ranking
HOME PAGE
Oh no! :o Make sure you've selected a category!
OK



ERROR: Simple Logic Error with the SWITCH/CASE Statement, where the strings in the case did not match those stored in the ComboBox:

```
switch (_selectedHighScore)
        {
          case "Quiz HighScore":
                    NumberOfScores = _parent.Database.GetSize("QuizScores", "DISTINCT UserID", "");
                    for (int i = 1; i < NumberOfScores + 1; i++)
                    {

                  List<string> HighScore = _parent.Database.ReadData("QuizScores", "MAX(Score)",
$"USERID = {i}", 1);
                  HighScores.Add(Int32.Parse(HighScore[0]));
                    if (i == _parent.UserID)
                    {
                            UserHS = Int32.Parse(HighScore[0]);
                    }

                    }
                    break;
          case "Pairs HighScore":
                    NumberOfScores = _parent.Database.GetSize("PairsScores", "DISTINCT UserID", "");
                    for (int i = 1; i < NumberOfScores + 1; i++)
                    {

                    List<string> HighScore = _parent.Database.ReadData("PairsScores", "MAX(Score)",
$"USERID = {i}", 1);
                  HighScores.Add(Int32.Parse(HighScore[0]));
                    if (i == _parent.UserID)
                    {
                    UserHS = Int32.Parse(HighScore[0]);
                    }

                    }
                    break;
          case "Word Scramble HighScore":
                    NumberOfScores = _parent.Database.GetSize("WordScrambleScores", "DISTINCT
UserID", "");
                    for (int i = 1; i < NumberOfScores + 1; i++)
                {
```

```
                List<string> HighScore = _parent.Database.ReadData("WordScrambleScores",
"MAX(Score)", $"USERID = {i}", 1);
                HighScores.Add(Int32.Parse(HighScore[0]));
                if (i == _parent.UserID)
                {
                UserHS = Int32.Parse(HighScore[0]);
                }

                }
                break;
        default:
                break;
        }
```

Fixed:

```
switch (_selectedHighScore)
        {
        case "Quiz Scores":
                NumberOfScores = _parent.Database.GetSize("QuizScores", "DISTINCT UserID", "");
                for (int i = 1; i < NumberOfScores + 1; i++)
                {

                List<string> HighScore = _parent.Database.ReadData("QuizScores", "MAX(Score)",
$"USERID = {i}", 1);
                HighScores.Add(Int32.Parse(HighScore[0]));
                if (i == _parent.UserID)
                {
                UserHS = Int32.Parse(HighScore[0]);
                }

                }
                break;
        case "Pairs Scores":
                NumberOfScores = _parent.Database.GetSize("PairsScores", "DISTINCT UserID", "");
                for (int i = 1; i < NumberOfScores + 1; i++)
                {

                List<string> HighScore = _parent.Database.ReadData("PairsScores", "MAX(Score)",
$"USERID = {i}", 1);
                HighScores.Add(Int32.Parse(HighScore[0]));
                if (i == _parent.UserID)
                {
                UserHS = Int32.Parse(HighScore[0]);
                }

                }
                break;
        case "Word Scramble Scores":
                NumberOfScores = _parent.Database.GetSize("WordScrambleScores", "DISTINCT
UserID", "");
                for (int i = 1; i < NumberOfScores + 1; i++)
                {

                List<string> HighScore = _parent.Database.ReadData("WordScrambleScores",
"MAX(Score)", $"USERID = {i}", 1);
                HighScores.Add(Int32.Parse(HighScore[0]));
                if (i == _parent.UserID)
                {
                UserHS = Int32.Parse(HighScore[0]);
                }

                }
                break;
        default:
```

```
                break;
        }
```

Evaluation

Comparison of Project Performance against Objectives

1. LOGIN SYSTEM
    a. Users should be able to sign up and have their details added to the database.

This Objective has been met. When the user opens the application, there is the option to select the sign up page. Once on the sign up page, the user can input a username and password, once this is validated the user's details are added to the database with a UserID.

    b. Users should be prevented from having the same username as one that already exists in the database.

This Objective has been met. The user will not be able to go any further until they input a username that doesn't already exist, instead they are given a message that gives them the option to try again or they can be taken to the login page.

    c. Passwords should have a validation check, such as having a special character or being at least 8 characters long.

This objective has partially been achieved. There is a validation to check that it is at least 5 characters long, but due to testing and time constraints, it is not as complex as originally planned.

    d. Existing users should be able sign in by checking their username and password against the login database.

This objective has been met. On the start the user can select the login page, where they can input their username and password. These are then validated and then are taken to the Home Page, if it is invalid there is a message that gives them the option to try again or they can be taken to the sign up page.

2. QUIZ SYSTEM
    a. Questions and answers should be stored in a database where they are organised by subjects in separate databases.

This objective has partially been achieved. Each pair of questions and answers are assigned to UserIDs, QuestionIDs and Question Bank Name. The default questions are assigned by a UserID of 0 and are accessible by all users. But instead of being in separate databases, all question banks are saved on one database, accessed via composite keys - Question Bank Name and UserID.

    b. Users should be able to make their own questions, being able to edit them as well as read them - all of this should be stored in the Question Database.

This objective has been met. The Create Question bank page, allows users to write to the database. The edit question bank page allows users to make changes to existing questions and answers. The users are also able to delete question banks, via the delete question bank page.

c. Users should be able to customise their quiz: being able to customise the number of questions and which question bank they answer from.

This objective has been met. On the Game Menu Page, the users can use the ComboBox to select which questions they would like to use for the quiz and pairs game. If the user doesn't select an option, then the default question bank is used. On the quiz page, the user is able to keep clicking the generate question button as many times as they'd like. In the pair game the user can pick how many pairs they want to match and equally can refresh the page and re-enter the number of pairs they want. The word scramble page, like the quiz page, also allows the user to keep clicking the generate button as many times as they would like.

d. Questions should be randomly selected based on the Question ID and then read from the database.

This objective has been met. Each game mode: the Quiz, the pairs and word scramble, all use random integers to access the information via indexes.

e. Users' answers should be checked by reading the database.

This objective has been met. The users are able to type the answers on the quiz page and the word scramble page. Using the databinding, which connects the values input in the xaml to the viewmodels, which is where the validation takes place. The user's input is compared to the expected answer, which has been read from the database.

f. Ensure that there are different question types/ styles for the user to answer, e.g.) fill in the blank, timelines, hot and cold sliders or multiple choice.

This objective has partially been achieved. Instead of being automatically assigned, the different question types have been achieved by having different game modes: the quiz, the pair game and the word scramble.

3. GLOBE
    a. An interactive map that can be dragged around in order to navigate the continents, reminiscent of Google Maps.

This objective has partially been achieved. On the continents map is relatively interactive, but unlike google maps, which can be dragged around and zoomed in, this is far more simplistic. Instead, there are buttons that allow the user to select a continent, but cannot be zoomed in further.

b. Countries and their information should be stored in a database.

To populate the database, a text file for each continent is read and using this content, each continent database is written to. The Continent Database has unique Country IDs, Country Name and the Country's statistics.

c. The User should be able to search through the database and retrieve the country's information, once the information is retrieved, the info should be displayed as a fact file.

This objective has been met. Using the ComboBox, it provides a list of all countries in a given continent, which can be scrolled through or the user can search for a country. Once a user has selected a country via the ComboBox, a search button is then clicked, which triggers a method that reads the database and retrieves that country's information, then displaying its statistics.

4. MATCHING GAME
    a. Information should be retrieved from the Question Database

This objective has been met. The text boxes are populated with random questions and answers from a selected question bank.

b. Users should be able to drag and drop terms and have this action validated by reading the Question Database to check their answer.

This objective has been met. In the Pairs Game, the user is able to drag and drop the text boxes and once they intersect, a validation occurs, which then returns whether or not it's a valid pair.

c. Users should be able to customise the game: choose how many pairs they match up, how many rounds, what question bank they use.

This objective has been met. On the Game Menu Page, the user can either use the ComboBox to select which question bank to use, if the user doesn't select an option, the default question bank. In the pair game the user can pick how many pairs they want to match and equally can refresh the page and re-enter the number of pairs they want. The word scramble page, like the quiz page, also allows the user to keep clicking the genera

5. SCORES SYSTEM
    a. There should be a database that stores each user's scores for every game they play, e.g.) quiz, matching ect.

This objective has been met. For each game there is a validation check to see if the user's answer is correct, after this the score will change, accordingly, if it is correct it will increase and if it is wrong it will decrease. Once the user clicks the finish button, the new score is stored in the database.

b. Users should be able to get their high score for each game.

This objective has been met. In order to get the highest score for each game for each user, the SQL function MAX(). Once this is found, it is stored in a variable and then displayed to the screen.

c. Users should be able to get the mean average score for each game.

This objective has been met. Using the SQL function SELECT, each game database is read, and all of the scores for the current user are stored in a temporary variable before the mean average score is calculated and displayed to the screen.

d. The User should also be able to get the percentile score and overall ranking compared to all other users for each game.

This objective has been met. All the highest scores for each user are read from each database, using the MAX() function and are stored in a list. A merge sort is performed on the list and then the current user's score's position is found. Using this information, the percentile is calculated, comparing the current user to all users in the database.

User Feedback

To see whether or not the application actually serves its purpose, a group interview was conducted (Appendix 2.1). It was natural to gain the insight of a group of exam year students, as this would be the intended audience, and it was extremely reassuring that the feedback was largely positive, with no critiques of the functionality or features. In fact, the students enjoyed the variety of the game modes and the competitive aspect of the users stats page. However, the students were helpful in giving constructive feedback, highlighting the need to have clearer instructions throughout the application and equally demonstrating the importance of having an aesthetically pleasing application. In addition, the group interview allowed certain flaws to be highlighted that had not been noticed, notably the fact that there is no way to exit the application or any way to log out.

Possible Extensions

Whilst the bare-bones of this project is satisfactory, there is no denying there are numerous points for improvement. The entire project revolved around interactivity and, unfortunately, it has not materialised. There is a level of interactivity that can be achieved through aesthetics, making this one of the key points I'd focus on, improving the cosmetics, having more colours and potentially including a mascot to provide encouragement and increase engagement inspired by the feedback from the group interview.

Continuing on the theme of interactivity, it would be far better to have the interactive map that was originally envisioned. Indeed, creating a map more similar to Google maps would have been ideal, but ultimately unrealistic, so a compromise would need to be made. Having a draggable image that could wrap around would provide a level of interactivity as well as instead of transparent buttons, assigning coordinates to the image that the user could click on to change to the different continents pages.

Conclusion

The feedback has supported the idea that this system is viable and would actually be useful to the users, especially if the feedback was implemented. Reassuringly, most of the changes that would be needed are rather minor and definitely could have been achieved, if not for the time constraints.

Equally, the inclusion of new ideas, such as a mascot and a reward system would increase the longevity and reusability of the application, and would have provided an interesting challenge to try and implement.

The timing was a major hindering factor, as I personally mismanaged time, and at some points underestimated different tasks, which negatively impacted my work plan, meaning I was catching up for the majority of the project.

As well as this, whilst I was confident with C# whilst using console applications, using xaml and a WPF project was extremely difficult and definitely cut into my time far more than I was expecting.

Despite this, I undoubtedly achieved all the main objectives of this project and the mostly positive feedback I received from my target users proves this.

Bibliography

SQLite Help:
https://www.codeguru.com/dotnet/using-sqlite-in-a-c-application/

Inspiration for Permutations:
https://stackoverflow.com/questions/12477339/finding-anagrams-for-a-given-word
https://www.interviewbit.com/blog/permutations-of-the-given-string/

Hashing:
https://www.sean-lloyd.com/post/hash-a-string/

Observable Objects:
https://learn.microsoft.com/en-us/dotnet/desktop/wpf/data/how-to-implement-property-change-notification?view=netframeworkdesktop-4.8

Simple Commands:
https://learn.microsoft.com/en-us/previous-versions/visualstudio/visual-studio-2013/dn235445(v=vs.120)

Drag and Drop:
https://www.youtube.com/watch?v=THV5BW5WW_o&t=517s

Appendix

Appendix 1.1.: Interview with Student M

What kind of things do you use to revise, do you use any particular apps?

Student M: *Yeah, I like to use Quizlet for a lot of my revision, in particular I like using flashcards and things like that. I also find having more interactive things like videos and audio really useful, as well as games like snap to match definitions to key words. So just a variety of revision techniques is good.*

How often do you spend revising, do you find it easy to regulate breaks and track your progress?

Student M: *That's actually something I struggle with, I end up studying for like 8 hours in one day and then nothing the next, being able to keep momentum and have productive study is something I need help with.*

What do you find encouraging when you're learning?

Student M: *I'm a visual learner, so I like having a lot of fun and colourful material. I often add drawings or colour code my notes to help me remember and engage me with the content.*

Appendix 1.2: Questionnaire

1.) What system do you use to revise/learn and if applicable, what applications do you use (E.G. word processors, revision apps)?

Student R: Quizlet

Student H: Revision Apps e.g.) Quizlet, Seneca

2.) What are the benefits of your current software?

Student R: It has multiple different modes and it is easy to make flashcards

Student H: I can use spaced repetition to get knowledge in my long term memory.

3.) What are the drawbacks of the software?

Student R: No real space for writing longer answers which is frustrating for someone doing essay subjects, plus you have to pay to access other people's sets once you've viewed them a few times

Student H: You can't have full access to features without paying a subscription.

4.) What features in these applications do you like?

Student R: Being able to study material in different forms, rather than just one type of question/mode, and also turning the flashcards into interactive games.

Student H: The test feature on quizlet means that you can check if you have learn everything

5.) What new features would be the most important?

Student R: It would be great if there was a specific section for making essay plans and writing out longer answers which could be broken down into smaller sections

Student H: Images and diagrams with the flashcards

6.)What existing features are the most useful?

Student R: The variety of question types - e.g.) asking for the definition of a term and then asking for the term based on the definition.

Student H: On Seneca you can redo the lesson as many times as you like to improve your knowledge.

7.) Would an interactive system encourage you to revise more?

Student R: Yes.

Student H: Yes

8.) Do you think a variety of question types would keep you engaged with the content?

Student R: Yes

Student H: Yes

Any other Comments?

Student R: N/A

Student H: N/A

Appendix 2.1: Group Interview (Paraphrased)

1.) What are some positive aspects of this application?

Student R: *I really liked the fact that there are different games, the variety of having the word scramble and pairs was a cool feature, it definitely helped keep it interesting*.

Student D: *For me, I really enjoyed the fact that there were competitive highscores that compared you to other users. This feature would definitely encourage me to keep getting better scores.*

Student M: *My favourite feature was the Hi Friend pop up box, because it allowed you to go to the sign up page without you having to go back yourself, in fact, the whole project is all very easy to navigate, which is definitely a plus.*

2.) Are there any noticeable flaws you'd like to comment on and how you'd improve the application?

Student R: *It's less of a problem because we've had you to guide us, but there aren't many instructions on the application, and so it is clear on formatting or what you actually need to do to make the question banks or how to play the games.*

Student D: *I agree that instructions are needed, but I also noticed that there is no way to leave or log out from the application, other than just forcing it to close.*

Student M: *I personally think that improving the look of the application would also be a good idea, with bright colours to make it more engaging. I also like the idea of having a mascot or something like a rewards system to help motivate users.*

Appendix 3.1: Africa Database, viewed in DB Browser (Example for all continents)

	ID	CountryName	Population	LandArea	Density
	Filter	Filter	Filter	Filter	Filter
1	0	Algeria	43851044	2381740	18
2	1	Angola	32866272	1246700	26
3	2	Benin	12123200	112760	108
4	3	Botswana	2351627	566730	4
5	4	Burkina Faso	20903273	273600	76
6	5	Burundi	11890784	25680	463
7	6	Côte d Ivoire	26378274	318000	83
8	7	Cabo Verde	555987	4030	138
9	8	Cameroon	26545863	472710	56
10	9	Central African Republic	4829767	622980	8
11	10	Chad	16425864	1259200	13
12	11	Comoros	869601	1861	467
13	12	The Republic of the Congo	5518087	341500	16
14	13	Democratic Republic of the Congo	89561403	2267050	40
15	14	Djibouti	988000	23180	43
16	15	Egypt	102334404	995450	103
17	16	Equatorial Guinea	1402985	28050	50
18	17	Eritrea	3546421	101000	35
19	18	Eswatini	1160164	17200	67
20	19	Ethiopia	114963588	1000000	115
21	20	Gabon	2225734	257670	9
22	21	Gambia	2416668	10120	239
23	22	Ghana	31072940	227540	137

	ID	CountryName	Population	LandArea	Density
	Filter	Filter	Filter	Filter	Filter
24	23	Guinea	13132795	245720	53
25	24	Guinea-Bissau	1968001	28120	70
26	25	Kenya	53771296	569140	94
27	26	Lesotho	2142249	30360	71
28	27	Liberia	5057681	96320	53
29	28	Libya	6871292	1759540	4
30	29	Madagascar	27691018	581795	48
31	30	Malawi	19129952	94280	203

	ID	CountryName	Population	LandArea	Density
	Filter	Filter	Filter	Filter	Filter
32	31	Mali	20250833	1220190	17
33	32	Mauritania	4649658	1030700	5
34	33	Mauritius	1271768	2030	626
35	34	Morocco	36910560	446300	83
36	35	Mozambique	31255435	786380	40
37	36	Namibia	2540905	823290	3
38	37	Niger	24206644	1266700	19
39	38	Nigeria	206139589	910770	226
40	39	Rwanda	12952218	24670	525
41	40	Sao Tome and Principe	219159	960	228
42	41	Senegal	16743927	192530	87
43	42	Seychelles	98347	460	214
44	43	Sierra Leone	7976983	72180	111
45	44	Somalia	15893222	627340	25
46	45	South Africa	59308690	59308690	1213090
47	46	South Sudan	11193725	610952	18
48	47	Sudan	43849260	1765048	25
49	48	Tanzania	59734218	885800	67
50	49	Togo	8278724	54390	152
51	50	Tunisia	11818619	155360	76
52	51	Uganda	45741007	199810	229
53	52	Zambia	18383955	743390	25
54	53	Zimbabwe	14862924	386850	38

Appendix 3.2: Login Details Database, viewed in DB Browser

	UserID	UserNames	Passwords
	Filter	Filter	Filter
1	1	User1	230 4 251 32 114 194 134 209 251 9...
2	2	User2	66 202 105 143 78 166 12 180 150 8...

Appendix 3.3: Quiz Scores Database, viewed in DB Browser (Example for all Games)

	QuizID	UserID	Score
	Filter	Filter	Filter
1	1	1	4
2	2	1	3
3	3	1	0
4	1	2	2
5	4	1	0
6	5	1	1
7	6	1	6
8	7	1	6

Appendix 3.4: Question Bank Database, viewed in DB Browser

	UserID	BankName	QuestionID	Question	Answer
	Filter	Filter	Filter	Filter	Filter
1	0	Default	1	Bonjour	Hello
2	0	Default	2	Salut	Hi
3	0	Default	3	Au Revoir	Bye
4	1	Practice1	1	Salut	Hi
5	1	Testing3	1	Bonjour	Hello
6	1	Testing1	1	French Capital	Paris
7	1	Testing1	2	Italian Capital	Rome