

Amber Clark  
App4Learn Code

App4Learn

Table of Contents

[Solution 1](#_Toc416726648)

[Example Prog 1](#_Toc416726649)

[App.config 1](#_Toc416726650)

[Packages.config 2](#_Toc416726651)

[Classes 2](#_Toc416726652)

[Constants 5](#_Toc416726653)

[Databaseclasses 7](#_Toc416726654)

[Helpers 20](#_Toc416726655)

[Inputclasses 22](#_Toc416726656)

[Interfaces 60](#_Toc416726657)

[Procedure Return Types 63](#_Toc416726658)

[Question Types 64](#_Toc416726659)

[Views 65](#_Toc416726660)

[Project Tests 106](#_Toc416726661)

[App.config 106](#_Toc416726662)

[Packages.config 106](#_Toc416726663)

[New Quest Type Tests 107](#_Toc416726664)

[Training Tests 150](#_Toc416726665)

[HelpersTests.vb 172](#_Toc416726666)

[LoginInpputTests.vb 175](#_Toc416726667)

[NewQuestionInputTests.vb 181](#_Toc416726668)

[NewUserInputTests.vb 200](#_Toc416726669)

[UserClassTests.vb 208](#_Toc416726670)

# Solution

## Example Prog

### App.config

<?xml version="1.0" encoding="utf-8"?>

<configuration>

<startup>

<supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5.1" />

</startup>

<connectionStrings>

<!--<add name="AppConnect" connectionString="Data Source=JIM-KIRK\MYSQLDB;Database=App4Learn;Trusted\_Connection=True;" providerName="System.Data.SqlClient" />-->

</connectionStrings>

</configuration>

### Packages.config

<?xml version="1.0" encoding="utf-8"?>

<packages>

<package id="Dapper" version="1.38" targetFramework="net451" />

<package id="Dapper.Tvp" version="1.0.0" targetFramework="net451" />

<package id="DapperWrapper" version="0.3.0.0" targetFramework="net451" />

<package id="Regextra" version="0.3.0.0" targetFramework="net451" />

</packages>

### Classes

#### DemonstrationStep.vb

Namespace Classes

''' <summary>

''' The individual Demonstration Step class.

''' The type input uses a list of these objects to keep track of all the demo steps.

''' </summary>

''' <remarks>The specific Demo Steps.</remarks>

Public Class DemonstrationStep

Property StepDetails As String

Property StepRegEx As String

Property StepMark As Integer

End Class

End Namespace

#### UserClass.vb

Imports ExampleProg.Interfaces

Namespace Classes

''' <summary>

''' The User class.

''' Does everything to do with the user.

''' Creates new users.

''' Stores log in and log out information.

''' </summary>

''' <remarks>The User class.</remarks>

Public Class UserClass

Implements IUserClass

Property UserId As Integer Implements IUserClass.UserId

Property UserName As String Implements IUserClass.UserName

Property Password As String Implements IUserClass.Password

Property FirstName As String Implements IUserClass.FirstName

Property MiddleName As String Implements IUserClass.MiddleName

Property LastName As String Implements IUserClass.LastName

Property IsValidUser As Boolean Implements IUserClass.IsValidUser

''' <summary>

''' Checks to see if the user name already exists in the database.

''' </summary>

''' <param name="dbConnection"></param>

''' <returns>True or false.</returns>

''' <remarks></remarks>

Public Function DoesUserNameAlreadyExists(dbConnection As IDbConnector) As Boolean Implements IUserClass.DoesUserNameAlreadyExists

Dim result As Boolean

result = dbConnection.DoesUserNameAlreadyExists(UserName)

Return result

End Function

''' <summary>

''' Is the user name already in use. Is it available.

''' </summary>

''' <param name="dbConnection"></param>

''' <returns>True or false.</returns>

''' <remarks></remarks>

Public Function IsUserNameInDatabase(dbConnection As IDbConnector) As Boolean Implements IUserClass.IsUserNameInDatabase

Dim result As Boolean

result = dbConnection.IsUserNameAvailable(UserName)

Return result

End Function

''' <summary>

''' Creates a new user in the database.

''' </summary>

''' <param name="dbConnection"></param>

''' <returns>True or false.</returns>

''' <remarks></remarks>

Public Function CreateNewUser(dbConnection As IDbConnector) As Boolean Implements IUserClass.CreateNewUser

Dim createdNewUser As Boolean

createdNewUser = dbConnection.CreateNewUser(Me)

Return createdNewUser

End Function

''' <summary>

''' Gets the user id from the log in details.

''' </summary>

''' <param name="dbConnection"></param>

''' <returns>The user id.</returns>

''' <remarks></remarks>

Public Function Login(dbConnection As IDbConnector) As Boolean Implements IUserClass.Login

Dim isLoggedIn As Boolean

' Validation is done in the log in input class.

UserId = dbConnection.GetUserIdOfValidUser(UserName, Password)

Dim tempUser = dbConnection.GetUserDetails(Me)

If (tempUser.UserName <> Nothing) Then

FirstName = tempUser.FirstName

MiddleName = tempUser.MiddleName

LastName = tempUser.LastName

UserName = tempUser.UserName

Password = tempUser.Password

IsValidUser = True

isLoggedIn = True

End If

Return isLoggedIn

End Function

''' <summary>

''' Logs out the current user. Just resets all the variables.

''' </summary>

''' <remarks></remarks>

Public Sub Logout() Implements IUserClass.Logout

UserId = 0

UserName = String.Empty

Password = String.Empty

FirstName = String.Empty

MiddleName = String.Empty

LastName = String.Empty

IsValidUser = False

End Sub

End Class

End Namespace

### Constants

#### CommonConstants.vb

Namespace Constants

''' <summary>

''' A collection of constants that are used throughout the application.

''' </summary>

''' <remarks>Constants.</remarks>

Public Class CommonConstants

Public Const NotAValidInput = "A value is required."

Public Const NotAValidInteger = "A number is required."

Public Const UserAlreadyExists = "That user name already exists."

Public Const UserIsAvailable = "That user name is available."

Public Const UserNameAndPasswordDoNotMatch = "User name or password are not recognized."

Public Const ValidWebAddresRegEx = "((https?://|www\.)([-\w\.]+)+(:\d+)?(/([\w/\_\.]\*(\?\S+)?)?)?)"

Public Const WebAddressNotValid = "The web address entered is not of a recognized standard."

Public Const InsertNewQuestionError = "The question was not inserted due to problems with the subject, curriculum, keystage or explanation."

Public Const InsertNewDemoStepError = "The demonstration step was not inserted."

Public Const NewQuestionInsertedCorrectly = "The new question has been inserted."

Public Const TrainingExplanationIdNotSuppliedError = "An explanation was not suppied. No details are available."

End Class

End Namespace

#### DatabaseConstants

Namespace Constants

''' <summary>

''' The Database Constants that are used throughout the application.

''' Namely the stored procedure names used in the database.

''' </summary>

''' <remarks>Database Constants.</remarks>

Public Class DatabaseConstants

Public Const ApplicationConfigString = "AppConnect"

Public Const DBConnecting = "Connecting..."

Public Const DBFailedToConnect = "Failed."

Public Const DBConnectedSuccesful = "Success."

Public Const TestDBConnection = "users.TestConnection"

Public Const CheckIfUserIsAvailble = "users.CheckIfUserIsAvailable"

Public Const CheckIfUserIsInDatabase = "users.CheckIfUserIsInDatabase"

Public Const ValidateUser = "users.ValidateUser"

Public Const GetUserDetails = "users.GetUserDetails"

Public Const CreateANewUser = "users.CreateNewUser"

Public Const GetTheListOfSubjects = "curriculum.GetListOfSubjects"

Public Const GetTheListOfCurriculums = "curriculum.GetListOfCurriculum"

Public Const GetTheListOfKeyStages = "curriculum.GetListOfKeyStages"

Public Const GetTheListOfExplanations = "curriculum.GetListOfExplanations"

Public Const GetTheFilteredListOfCurriculums = "curriculum.GetFilteredListOfCurriculum"

Public Const GetTheFilteredListOfExplanationsBySubject = "curriculum.GetFilteredListOfExplanationsBySubject"

Public Const GetTheFilteredListOfExplanationsByCurriculum = "curriculum.GetFilteredListOfExplanationsByCurriculum"

Public Const GetExplanationDescription = "curriculum.GetExplanationDetails"

Public Const InsertNewSubject = "curriculum.InsertNewSubject"

Public Const InsertNewCurriculum = "curriculum.InsertNewCurriculum"

Public Const InsertNewKeyStage = "curriculum.InsertNewKeyStage"

Public Const InsertNewExplanation = "curriculum.InsertNewExplanation"

Public Const InsertNewQuestion = "questions.InsertNewQuestion"

Public Const InsertNewDemoSteps = "questions.InsertNewDemonstrationStep"

Public Const GetTrainingExplanationDetails = "questions.GetTrainingExplanationDetails"

Public Const GetTrainingListOfQuestionIds = "questions.GetListOfQuestionIds"

Public Const GetTrainingDemonstrationStepDetails = "questions.GetTrainingDemoStepDetailsList"

Public Const GetFullTrainingImmitationSteps = "questions.GetImmitationSteps"

End Class

End Namespace

### Databaseclasses

#### DatabaseConnectorWrapper.vb

Imports Dapper

Imports DapperWrapper

Imports ExampleProg.Classes

Imports ExampleProg.Constants

Imports ExampleProg.Interfaces

Imports ExampleProg.ProcedureReturnTypes

Imports ExampleProg.QuestionTypes

Namespace DatabaseClasses

''' <summary>

''' The Database Connector Wrapper.

''' All the calls to the database go through this class.

''' It wraps the Dapper Factory, to keep the database access to one single point.

''' </summary>

''' <remarks>Database Connection.</remarks>

Public Class DatabaseConnectorWrapper

Implements IDbConnector

Private ReadOnly \_dbExecutorFactory As IDbExecutorFactory

Private \_databaseConnectionStatus As String

''' <summary>

''' The basic constructor.

''' </summary>

''' <param name="dbExecutorFactory"></param>

''' <remarks></remarks>

Public Sub New(dbExecutorFactory As IDbExecutorFactory)

\_dbExecutorFactory = dbExecutorFactory

TestDatabaseConnection()

End Sub

''' <summary>

''' A simple check for the db connection.

''' </summary>

''' <remarks></remarks>

Private Sub TestDatabaseConnection()

Dim result As Boolean

Try

Dim db = \_dbExecutorFactory.CreateExecutor()

result = db.Query(Of Boolean)(DatabaseConstants.TestDBConnection, commandType:=CommandType.StoredProcedure).FirstOrDefault()

Catch ex As Exception

result = False

End Try

\_databaseConnectionStatus = If(result, DatabaseConstants.DBConnectedSuccesful, DatabaseConstants.DBFailedToConnect)

End Sub

''' <summary>

''' Gets the db connection status.

''' </summary>

''' <returns>True or false.</returns>

''' <remarks></remarks>

Public Function DBConnectionStatus() As String Implements IDbConnector.DBConnectionStatus

Return \_databaseConnectionStatus

End Function

#Region "UserConnections"

''' <summary>

''' DB script to check if the user name is availble.

''' </summary>

''' <param name="userName"></param>

''' <returns>True or false.</returns>

''' <remarks></remarks>

Public Function IsUserNameAvailable(userName As String) As Boolean Implements IDbConnector.IsUserNameAvailable

Dim result As Boolean

Try

Dim db = \_dbExecutorFactory.CreateExecutor()

Dim dynamicParameters = New DynamicParameters()

dynamicParameters.Add("@UserName", userName)

result = db.Query(Of Boolean)(DatabaseConstants.CheckIfUserIsAvailble, dynamicParameters, commandType:=CommandType.StoredProcedure).FirstOrDefault()

Catch ex As Exception

result = False

End Try

Return result

End Function

''' <summary>

''' DB check if the user name already exists.

''' </summary>

''' <param name="userName"></param>

''' <returns>True or false.</returns>

''' <remarks></remarks>

Public Function DoesUserNameAlreadyExists(userName As String) As Boolean Implements IDbConnector.DoesUserNameAlreadyExists

Dim result As Boolean

Try

Dim db = \_dbExecutorFactory.CreateExecutor()

Dim dynamicParameters = New DynamicParameters()

dynamicParameters.Add("@UserName", userName)

result = db.Query(Of Boolean)(DatabaseConstants.CheckIfUserIsInDatabase, dynamicParameters, commandType:=CommandType.StoredProcedure).FirstOrDefault()

Catch ex As Exception

result = False

End Try

Return result

End Function

''' <summary>

''' DB get for the user id.

''' </summary>

''' <param name="userName"></param>

''' <param name="password"></param>

''' <returns>The id of the user from the details.</returns>

''' <remarks></remarks>

Public Function GetUserIdOfValidUser(userName As String, password As String) As Integer Implements IDbConnector.GetUserIdOfValidUser

Dim userId As Integer

Try

Dim db = \_dbExecutorFactory.CreateExecutor()

Dim dynamicParameters = New DynamicParameters()

dynamicParameters.Add("@UserName", userName)

dynamicParameters.Add("@Password", password)

userId = db.Query(Of Integer)(DatabaseConstants.ValidateUser, dynamicParameters, commandType:=CommandType.StoredProcedure).FirstOrDefault()

Catch ex As Exception

userId = -1

End Try

Return userId

End Function

''' <summary>

''' DB get for the full user details.

''' </summary>

''' <param name="user"></param>

''' <returns>The user.</returns>

''' <remarks></remarks>

Public Function GetUserDetails(user As IUserClass) As UserClass Implements IDbConnector.GetUserDetails

Try

Dim db = \_dbExecutorFactory.CreateExecutor()

Dim dynamicParameters = New DynamicParameters()

dynamicParameters.Add("@UserID", user.UserId)

user = db.Query(Of UserClass)(DatabaseConstants.GetUserDetails, dynamicParameters, commandType:=CommandType.StoredProcedure).FirstOrDefault()

Catch ex As Exception

user = New UserClass

End Try

Return CType(user, UserClass)

End Function

''' <summary>

''' DB create a new user in the database.

''' </summary>

''' <param name="user"></param>

''' <returns>True or false.</returns>

''' <remarks></remarks>

Public Function CreateNewUser(user As IUserClass) As Boolean Implements IDbConnector.CreateNewUser

' Validation for all of the values here are done in the NewUserInput class.

Dim success As Boolean

Try

Dim db = \_dbExecutorFactory.CreateExecutor()

Dim dynamicParameters = New DynamicParameters()

dynamicParameters.Add("@UserName", user.UserName)

dynamicParameters.Add("@Password", user.Password)

dynamicParameters.Add("@FirstName", user.FirstName)

dynamicParameters.Add("@MiddleName", user.MiddleName)

dynamicParameters.Add("@LastName", user.LastName)

db.Execute(DatabaseConstants.CreateANewUser, dynamicParameters, commandType:=CommandType.StoredProcedure)

success = True

Catch ex As Exception

success = False

End Try

Return success

End Function

#End Region

#Region "Insert Question"

''' <summary>

''' DB get - gets the list of subjects in the database.

''' </summary>

''' <returns></returns>

''' <remarks></remarks>

Public Function GetListOfSubjects() As IEnumerable(Of SubjectListType) Implements IDbConnector.GetListOfSubjects

Dim list As IEnumerable(Of SubjectListType)

Try

Dim db = \_dbExecutorFactory.CreateExecutor()

list = db.Query(Of SubjectListType)(DatabaseConstants.GetTheListOfSubjects, commandType:=CommandType.StoredProcedure)

Catch ex As Exception

list = New List(Of SubjectListType)

End Try

Return list

End Function

''' <summary>

''' DB get - gets the list of curriculums in the database.

''' </summary>

''' <returns></returns>

''' <remarks></remarks>

Public Function GetListOfCurriculum() As IEnumerable(Of CurriculumListType) Implements IDbConnector.GetListOfCurriculum

Dim list As IEnumerable(Of CurriculumListType)

Try

Dim db = \_dbExecutorFactory.CreateExecutor()

list = db.Query(Of CurriculumListType)(DatabaseConstants.GetTheListOfCurriculums, commandType:=CommandType.StoredProcedure)

Catch ex As Exception

list = New List(Of CurriculumListType)

End Try

Return list

End Function

''' <summary>

''' DB get - gets the list of key stages in the database.

''' </summary>

''' <returns></returns>

''' <remarks></remarks>

Public Function GetListOfKeyStages() As IEnumerable(Of KeyStageListType) Implements IDbConnector.GetListOfKeyStages

Dim list As IEnumerable(Of KeyStageListType)

Try

Dim db = \_dbExecutorFactory.CreateExecutor()

list = db.Query(Of KeyStageListType)(DatabaseConstants.GetTheListOfKeyStages, commandType:=CommandType.StoredProcedure)

Catch ex As Exception

list = New List(Of KeyStageListType)

End Try

Return list

End Function

''' <summary>

''' DB get - gets the list of explanations in the database.

''' </summary>

''' <returns></returns>

''' <remarks></remarks>

Public Function GetListOfExplanations() As IEnumerable(Of ExplanationListType) Implements IDbConnector.GetListOfExplanations

Dim list As IEnumerable(Of ExplanationListType)

Try

Dim db = \_dbExecutorFactory.CreateExecutor()

list = db.Query(Of ExplanationListType)(DatabaseConstants.GetTheListOfExplanations, commandType:=CommandType.StoredProcedure)

Catch ex As Exception

list = New List(Of ExplanationListType)

End Try

Return list

End Function

''' <summary>

''' DB get - gets a list of curriculumns associated with a specific subject.

''' </summary>

''' <param name="subjectID"></param>

''' <returns></returns>

''' <remarks></remarks>

Public Function GetUpdatedListOfCurriculum(subjectID As Integer) As IEnumerable(Of CurriculumListType) Implements IDbConnector.GetUpdatedListOfCurriculum

Dim list As IEnumerable(Of CurriculumListType)

Try

Dim db = \_dbExecutorFactory.CreateExecutor()

Dim dynamicParameters = New DynamicParameters()

dynamicParameters.Add("@SubjectID", subjectID)

list = db.Query(Of CurriculumListType)(DatabaseConstants.GetTheFilteredListOfCurriculums, dynamicParameters, commandType:=CommandType.StoredProcedure)

Catch ex As Exception

list = New List(Of CurriculumListType)

End Try

Return list

End Function

''' <summary>

''' DB get - gets a list of explanations associated with a specific subject.

''' </summary>

''' <param name="subjectID"></param>

''' <returns></returns>

''' <remarks></remarks>

Public Function GetUpdatedListOfExplanations(subjectID As Integer) As IEnumerable(Of ExplanationListType) Implements IDbConnector.GetUpdatedListOfExplanations

Dim list As IEnumerable(Of ExplanationListType)

Try

Dim db = \_dbExecutorFactory.CreateExecutor()

Dim dynamicParameters = New DynamicParameters()

dynamicParameters.Add("@SubjectID", subjectID)

list = db.Query(Of ExplanationListType)(DatabaseConstants.GetTheFilteredListOfExplanationsBySubject, dynamicParameters, commandType:=CommandType.StoredProcedure)

Catch ex As Exception

list = New List(Of ExplanationListType)

End Try

Return list

End Function

''' <summary>

''' DB get - gets a list of explanations associated with a specific curriculum.

''' </summary>

''' <param name="curriculumID"></param>

''' <returns></returns>

''' <remarks></remarks>

Public Function GetUpdatedExplanationsByCurriculum(curriculumID As Integer) As IEnumerable(Of ExplanationListType) Implements IDbConnector.GetUpdatedExplanationsByCurriculum

Dim list As IEnumerable(Of ExplanationListType)

Try

Dim db = \_dbExecutorFactory.CreateExecutor()

Dim dynamicParameters = New DynamicParameters()

dynamicParameters.Add("@CurriculumID", curriculumID)

list = db.Query(Of ExplanationListType)(DatabaseConstants.GetTheFilteredListOfExplanationsByCurriculum, dynamicParameters, commandType:=CommandType.StoredProcedure)

Catch ex As Exception

list = New List(Of ExplanationListType)

End Try

Return list

End Function

''' <summary>

''' DB get - gets the explanation details based on the id provided.

''' </summary>

''' <param name="explanationID"></param>

''' <returns></returns>

''' <remarks></remarks>

Public Function GetExplanationDetailsById(explanationID As Integer) As String Implements IDbConnector.GetExplanationDetailsById

Dim result As String

Try

Dim db = \_dbExecutorFactory.CreateExecutor()

Dim dynamicParameters = New DynamicParameters()

dynamicParameters.Add("@ExplanationID", explanationID)

result = db.Query(Of String)(DatabaseConstants.GetExplanationDescription, dynamicParameters, commandType:=CommandType.StoredProcedure).FirstOrDefault()

Catch ex As Exception

result = String.Empty

End Try

Return result

End Function

''' <summary>

''' DB - inserts a new subject.

''' </summary>

''' <param name="subjectText"></param>

''' <returns></returns>

''' <remarks></remarks>

Public Function InsertNewSubject(subjectText As String) As Integer Implements IDbConnector.InsertNewSubject

Dim subjectID As Integer

Try

Dim db = \_dbExecutorFactory.CreateExecutor()

Dim dynamicParameters = New DynamicParameters()

dynamicParameters.Add("@SubjectDetail", subjectText)

subjectID = db.Query(Of Integer)(DatabaseConstants.InsertNewSubject, dynamicParameters, commandType:=CommandType.StoredProcedure).FirstOrDefault()

Catch ex As Exception

subjectID = 0

End Try

Return subjectID

End Function

''' <summary>

''' DB - inserts a new curriculum.

''' </summary>

''' <param name="subjectID"></param>

''' <param name="curriculumText"></param>

''' <returns></returns>

''' <remarks></remarks>

Public Function InsertNewCurriculum(subjectID As Integer, curriculumText As String) As Integer Implements IDbConnector.InsertNewCurriculum

Dim curriculumID As Integer

Try

Dim db = \_dbExecutorFactory.CreateExecutor()

Dim dynamicParameters = New DynamicParameters()

dynamicParameters.Add("@SubjectID", subjectID)

dynamicParameters.Add("@CurriculumDetail", curriculumText)

curriculumID = db.Query(Of Integer)(DatabaseConstants.InsertNewCurriculum, dynamicParameters, commandType:=CommandType.StoredProcedure).FirstOrDefault()

Catch ex As Exception

curriculumID = 0

End Try

Return curriculumID

End Function

''' <summary>

''' DB - inserts a new key stage.

''' </summary>

''' <param name="keyStageText"></param>

''' <returns></returns>

''' <remarks></remarks>

Public Function InsertNewKeyStage(keyStageText As String) As Integer Implements IDbConnector.InsertNewKeyStage

Dim keyStageID As Integer

Try

Dim db = \_dbExecutorFactory.CreateExecutor()

Dim dynamicParameters = New DynamicParameters()

dynamicParameters.Add("@KeyStageDetails", keyStageText)

keyStageID = db.Query(Of Integer)(DatabaseConstants.InsertNewKeyStage, dynamicParameters, commandType:=CommandType.StoredProcedure).FirstOrDefault()

Catch ex As Exception

keyStageID = 0

End Try

Return keyStageID

End Function

''' <summary>

''' DB - inserts a new explanation.

''' </summary>

''' <param name="curriculumID"></param>

''' <param name="explanationTitle"></param>

''' <param name="explanationText"></param>

''' <returns></returns>

''' <remarks></remarks>

Public Function InsertNewExplanation(curriculumID As Integer, explanationTitle As String, explanationText As String) As Integer Implements IDbConnector.InsertNewExplanation

Dim explanationID As Integer

Try

Dim db = \_dbExecutorFactory.CreateExecutor()

Dim dynamicParameters = New DynamicParameters()

dynamicParameters.Add("@CurriculumID", curriculumID)

dynamicParameters.Add("@Title", explanationTitle)

dynamicParameters.Add("@Description", explanationText)

explanationID = db.Query(Of Integer)(DatabaseConstants.InsertNewExplanation, dynamicParameters, commandType:=CommandType.StoredProcedure).FirstOrDefault()

Catch ex As Exception

explanationID = 0

End Try

Return explanationID

End Function

''' <summary>

''' DB - inserts a new question.

''' </summary>

''' <param name="question"></param>

''' <returns></returns>

''' <remarks></remarks>

Public Function InsertNewQuestion(question As QuestionInsertType) As Integer Implements IDbConnector.InsertNewQuestion

Dim questionID As Integer

Try

Dim db = \_dbExecutorFactory.CreateExecutor()

Dim dynamicParameters = New DynamicParameters()

dynamicParameters.Add("@SelectedSubjectID", question.SelectedSubjectID)

dynamicParameters.Add("@SelectedCurriculumID", question.SelectedCurriculumID)

dynamicParameters.Add("@SelectedKeyStageID", question.SelectedKeyStageID)

dynamicParameters.Add("@SelectedExplanationID", question.SelectedExplanationID)

dynamicParameters.Add("@NewQuestion", question.NewQuestion)

dynamicParameters.Add("@NewAnswer", question.NewAnswer)

dynamicParameters.Add("@AmountOfStepsSet", question.AmountOfStepsSet)

dynamicParameters.Add("@TotalMarksSet", question.TotalMarksSet)

dynamicParameters.Add("@NewFalseAnswerA", question.NewFalseAnswerA)

dynamicParameters.Add("@NewFalseAnswerB", question.NewFalseAnswerB)

dynamicParameters.Add("@NewFalseAnswerC", question.NewFalseAnswerC)

dynamicParameters.Add("@NewSearchString", question.NewSearchString)

dynamicParameters.Add("@NewWebLink", question.NewWebLink)

questionID = db.Query(Of Integer)(DatabaseConstants.InsertNewQuestion, dynamicParameters, commandType:=CommandType.StoredProcedure).FirstOrDefault()

Catch ex As Exception

questionID = 0

End Try

Return questionID

End Function

''' <summary>

''' DB - inserts a new demonstration step.

''' </summary>

''' <param name="demo"></param>

''' <param name="questionID"></param>

''' <returns></returns>

''' <remarks></remarks>

Public Function InsertNewDemoStep(demo As DemonstrationStepType, questionID As Integer) As Boolean Implements IDbConnector.InsertNewDemoStep

Dim insertedCorrect As Boolean

Try

Dim db = \_dbExecutorFactory.CreateExecutor()

Dim dynamicParameters = New DynamicParameters()

dynamicParameters.Add("@DemonstrationStageInsert", demo.DemonstrationStageInsert)

dynamicParameters.Add("@RegExInsert", demo.RegExInsert)

dynamicParameters.Add("@StageMarkInsert", demo.StageMarkInsert)

dynamicParameters.Add("@QuestionID", questionID)

db.Query(DatabaseConstants.InsertNewDemoSteps, dynamicParameters, commandType:=CommandType.StoredProcedure).FirstOrDefault()

insertedCorrect = True

Catch ex As Exception

insertedCorrect = False

End Try

Return insertedCorrect

End Function

#End Region

#Region "Training Section"

''' <summary>

''' DB get - gets the explanation details for training.

''' </summary>

''' <param name="explanationID"></param>

''' <returns></returns>

''' <remarks></remarks>

Public Function GetExplanationDetailsForTraining(explanationID As Integer) As GetExplanationDetailsType Implements IDbConnector.GetExplanationDetailsForTraining

Dim explanation As GetExplanationDetailsType

Try

Dim db = \_dbExecutorFactory.CreateExecutor()

Dim dynamicParameters = New DynamicParameters()

dynamicParameters.Add("@SelectedExplanationID", explanationID)

explanation = db.Query(Of GetExplanationDetailsType)(DatabaseConstants.GetTrainingExplanationDetails, dynamicParameters, commandType:=CommandType.StoredProcedure).FirstOrDefault()

Catch ex As Exception

explanation = New GetExplanationDetailsType

End Try

Return explanation

End Function

''' <summary>

''' DB get - gets a list of question ids for training.

''' </summary>

''' <param name="subjectID"></param>

''' <param name="curriculumID"></param>

''' <param name="keyStageID"></param>

''' <param name="explanationID"></param>

''' <returns></returns>

''' <remarks></remarks>

Public Function GetListOfQuestionIdsForTraining(subjectID As Integer, curriculumID As Integer, keyStageID As Integer, explanationID As Integer) As IEnumerable(Of Integer) Implements IDbConnector.GetListOfQuestionIdsForTraining

Dim list As IEnumerable(Of Integer)

Try

Dim db = \_dbExecutorFactory.CreateExecutor()

Dim dynamicParameters = New DynamicParameters()

dynamicParameters.Add("@SubjectID", subjectID)

dynamicParameters.Add("@CurriculumID", curriculumID)

dynamicParameters.Add("@KeyStageID", keyStageID)

dynamicParameters.Add("@ExplanationID", explanationID)

list = db.Query(Of Integer)(DatabaseConstants.GetTrainingListOfQuestionIds, dynamicParameters, commandType:=CommandType.StoredProcedure)

Catch ex As Exception

list = New List(Of Integer)()

End Try

Return list

End Function

''' <summary>

''' DB get - gets the list of demo steps for the question.

''' </summary>

''' <param name="questionID"></param>

''' <returns></returns>

''' <remarks></remarks>

Public Function GetListOfDemoStepDetails(questionID As Integer) As IEnumerable(Of String) Implements IDbConnector.GetListOfDemoStepDetails

Dim list As IEnumerable(Of String)

Try

Dim db = \_dbExecutorFactory.CreateExecutor()

Dim dynamicParameters = New DynamicParameters()

dynamicParameters.Add("@QuestionID", questionID)

list = db.Query(Of String)(DatabaseConstants.GetTrainingDemonstrationStepDetails, dynamicParameters, commandType:=CommandType.StoredProcedure)

Catch ex As Exception

list = New List(Of String)

End Try

Return list

End Function

''' <summary>

''' DB get - gets a list of imitation stage steps.

''' </summary>

''' <param name="questionID"></param>

''' <returns></returns>

''' <remarks></remarks>

Public Function GetImmitationStageSteps(questionID As Integer) As IEnumerable(Of ImmitationStageListType) Implements IDbConnector.GetImmitationStageSteps

Dim list As IEnumerable(Of ImmitationStageListType)

Try

Dim db = \_dbExecutorFactory.CreateExecutor()

Dim dynamicParameters = New DynamicParameters()

dynamicParameters.Add("@QuestionID", questionID)

list = db.Query(Of ImmitationStageListType)(DatabaseConstants.GetFullTrainingImmitationSteps, dynamicParameters, commandType:=CommandType.StoredProcedure)

Catch ex As Exception

list = New List(Of ImmitationStageListType)

End Try

Return list

End Function

#End Region

End Class

End Namespace

### Helpers

#### Helpers.vb

Imports System.Text

Namespace Helpers

''' <summary>

''' A Helper class.

''' Used like a static class in C#.

''' A set of basic functions that are re-used throughout the application.

''' To prevent code being copied throughout.

''' </summary>

''' <remarks>A set of helper functions.</remarks>

Public Module Helpers

Public Function IsTextNotNull(value As String) As Boolean

Dim result As Boolean

result = True

If (String.IsNullOrWhiteSpace(value)) Then

result = False

End If

Return result

End Function

''' <summary>

''' Checking for a null or zero.

''' If neither, then true is returned.

''' </summary>

''' <param name="value"></param>

''' <returns>True or false.</returns>

Public Function IsDigitNotNullOrZero(value As Integer) As Boolean

Dim result As Boolean

result = True

If (value = Nothing OrElse value < 1) Then

result = False

End If

Return result

End Function

''' <summary>

''' Goes through the array of booleans to find a single false.

''' </summary>

''' <param name="array"></param>

''' <returns>If there is one false, then it returns false.</returns>

''' <remarks></remarks>

Public Function IsThereAFalseInArray(array As Array) As Boolean

Dim result As Boolean = True

Dim tempArray(array.Length) As Boolean

array.CopyTo(tempArray, 0)

For index As Integer = 0 To (array.Length - 1)

If Not (tempArray(index)) Then

result = False

End If

Next

Return result

End Function

''' <summary>

''' Changes the enter key into a tab key.

''' </summary>

''' <param name="e"></param>

''' <returns></returns>

''' <remarks></remarks>

Public Function PerformTabOnEnter(ByVal e As KeyEventArgs) As Boolean

Dim isEnter As Boolean

isEnter = False

If (e.KeyCode = Keys.Enter) Then

isEnter = True

SendKeys.Send("{TAB}")

Else

Return isEnter

End If

e.SuppressKeyPress = True 'this will prevent ding sound

Return isEnter

End Function

''' <summary>

''' Formats the search string into one fully formatted string.

''' </summary>

''' <param name="array"></param>

''' <returns>The formatted string.</returns>

''' <remarks></remarks>

Public Function FormatSearchString(array As Array) As StringBuilder

Dim formattedString = New StringBuilder

For Each item As String In array

Dim index As Int32 = array.IndexOf(array, item)

If Not (item = "N/A") Then

formattedString.Append("""")

formattedString.Append(item)

formattedString.Append("""")

If Not (index = array.Length - 1) Then

formattedString.Append(" + ")

End If

End If

Next

Return formattedString

End Function

End Module

End Namespace

### Inputclasses

#### NewQuestionTypes

##### AnswerTypeInput.vb

Imports ExampleProg.Constants

Namespace InputClasses.NewQuestionTypes

''' <summary>

''' The answer type input process.

''' </summary>

''' <remarks></remarks>

Public Class AnswerTypeInput

Property CorrectAnswer As String

Property InCorrectAnswerA As String

Property InCorrectAnswerB As String

Property InCorrectAnswerC As String

Property InvalidCorrectAnswerMessage As String

Property InvalidInCorrectAnswerAMessage As String

Property InvalidInCorrectAnswerBMessage As String

Property InvalidInCorrectAnswerCMessage As String

Property IsValidCorrectAnswer As Boolean

Property IsValidInCorrectAnswerA As Boolean

Property IsValidInCorrectAnswerB As Boolean

Property IsValidInCorrectAnswerC As Boolean

Property CanProceed As Boolean

''' <summary>

''' The processor for the class.

''' Declars and sets up all the variables.

''' Sets the flags and messages based upon the values passed in.

''' </summary>

''' <param name="answer"></param>

''' <param name="wrongAnswerA"></param>

''' <param name="wrongAnswerB"></param>

''' <param name="wrongAnswerC"></param>

''' <remarks></remarks>

Public Sub Process(answer As String, wrongAnswerA As String, wrongAnswerB As String, wrongAnswerC As String)

CorrectAnswer = answer

InCorrectAnswerA = wrongAnswerA

InCorrectAnswerB = wrongAnswerB

InCorrectAnswerC = wrongAnswerC

InvalidCorrectAnswerMessage = String.Empty

InvalidInCorrectAnswerAMessage = String.Empty

InvalidInCorrectAnswerBMessage = String.Empty

InvalidInCorrectAnswerCMessage = String.Empty

IsValidCorrectAnswer = False

IsValidInCorrectAnswerA = False

IsValidInCorrectAnswerB = False

IsValidInCorrectAnswerC = False

SetValidTextFlags()

SetInvalidMessages()

SetCanProceed()

End Sub

''' <summary>

''' Sets the valid flags.

''' </summary>

''' <remarks></remarks>

Private Sub SetValidTextFlags()

IsValidCorrectAnswer = Helpers.IsTextNotNull(CorrectAnswer)

IsValidInCorrectAnswerA = Helpers.IsTextNotNull(InCorrectAnswerA)

IsValidInCorrectAnswerB = Helpers.IsTextNotNull(InCorrectAnswerB)

IsValidInCorrectAnswerC = Helpers.IsTextNotNull(InCorrectAnswerC)

End Sub

''' <summary>

''' Sets the invalid messages.

''' </summary>

''' <remarks></remarks>

Private Sub SetInvalidMessages()

InvalidCorrectAnswerMessage = If(IsValidCorrectAnswer, String.Empty, CommonConstants.NotAValidInput)

InvalidInCorrectAnswerAMessage = If(IsValidInCorrectAnswerA, String.Empty, CommonConstants.NotAValidInput)

InvalidInCorrectAnswerBMessage = If(IsValidInCorrectAnswerB, String.Empty, CommonConstants.NotAValidInput)

InvalidInCorrectAnswerCMessage = If(IsValidInCorrectAnswerC, String.Empty, CommonConstants.NotAValidInput)

End Sub

''' <summary>

''' Sets the can proceed flag if all is okay.

''' </summary>

''' <remarks></remarks>

Private Sub SetCanProceed()

Dim arrayOfFlags(3) As Boolean

arrayOfFlags(0) = IsValidCorrectAnswer

arrayOfFlags(1) = IsValidInCorrectAnswerA

arrayOfFlags(2) = IsValidInCorrectAnswerB

arrayOfFlags(3) = IsValidInCorrectAnswerC

CanProceed = Helpers.IsThereAFalseInArray(arrayOfFlags)

End Sub

End Class

End Namespace

##### CurriculumTypeInput.vb

Imports ExampleProg.Constants

Imports ExampleProg.ProcedureReturnTypes

Namespace InputClasses.NewQuestionTypes

''' <summary>

''' The curriculum type input class.

''' </summary>

''' <remarks></remarks>

Public Class CurriculumTypeInput

Property CurriculumID As Integer

Property CurriculumText As String

Property IsNewCurriculum As Boolean

Property InvalidMessage As String

Property CanProceed As Boolean

Property CurriculumList As IEnumerable(Of CurriculumListType)

''' <summary>

''' The processor for the class.

''' Declares the variables.

''' Sets the flags and messages.

''' </summary>

''' <param name="text"></param>

''' <remarks></remarks>

Public Sub Process(text As String)

CurriculumText = text

CurriculumID = 0

IsNewCurriculum = False

InvalidMessage = String.Empty

CanProceed = False

SetCurriculumAndMessageFlags()

SetCurriculumID()

End Sub

''' <summary>

''' Sets the flags.

''' </summary>

''' <remarks></remarks>

Private Sub SetCurriculumAndMessageFlags()

CanProceed = Helpers.Helpers.IsTextNotNull(CurriculumText)

InvalidMessage = If(CanProceed, String.Empty, CommonConstants.NotAValidInput)

End Sub

''' <summary>

''' Sets the curriculum id.

''' </summary>

''' <remarks></remarks>

Private Sub SetCurriculumID()

CurriculumID = (From curriculum In CurriculumList

Where curriculum.CurriculumDetails.ToLower().Equals(CurriculumText.ToLower())

Select curriculum.ID).FirstOrDefault()

If (CurriculumID = 0) Then

IsNewCurriculum = True

End If

End Sub

End Class

End Namespace

##### DemonstrationTypeInput.vb

Imports ExampleProg.Classes

Imports ExampleProg.Constants

Imports Regextra

Namespace InputClasses.NewQuestionTypes

''' <summary>

''' The Demonstration Type Input class.

''' Does all of the validation for the demo steps.

''' Collects and maintains a list of the demo steps.

''' </summary>

''' <remarks>The Demo Step.</remarks>

Public Class DemonstrationTypeInput

Private \_demonstrationDetails As String

Private \_demonstrationStepMark As Integer

Property DemonstrationTotalMarks As Integer

Property RegExDetails As String

Property AmountOfSteps As Integer

Property DemonstrationList As List(Of DemonstrationStep)

Property IsValidDemoDetails As Boolean

Property InvalidDemoDetailsMessage As String

Property IsValidRegExDetails As Boolean

Property InvalidRegExDetailsMessage As String

Property IsValidMark As Boolean

Property InvalidMarkMessage As String

Property CanInsertDetailsToList As Boolean

''' <summary>

''' Resets the values to their defaults.

''' </summary>

''' <remarks></remarks>

Public Sub ResetValuesToDefaults()

\_demonstrationDetails = String.Empty

RegExDetails = String.Empty

\_demonstrationStepMark = 0

DemonstrationList = New List(Of DemonstrationStep)

AmountOfSteps = DemonstrationList.Count

IsValidDemoDetails = False

IsValidRegExDetails = False

InvalidDemoDetailsMessage = String.Empty

InvalidRegExDetailsMessage = String.Empty

End Sub

''' <summary>

''' Gets the demonstration details.

''' </summary>

''' <returns></returns>

''' <remarks></remarks>

Public Function GetDemoDetails() As String

Return \_demonstrationDetails

End Function

''' <summary>

''' Inserts the next step details.String`

''' </summary>

''' <param name="demoDetails"></param>

''' <remarks></remarks>

Public Sub InsertNextStepDetails(demoDetails As String)

\_demonstrationDetails = demoDetails

IsValidDemoDetails = Helpers.IsTextNotNull(\_demonstrationDetails)

InvalidDemoDetailsMessage = If(IsValidDemoDetails, String.Empty, CommonConstants.NotAValidInput)

End Sub

''' <summary>

''' Sets the reg ex for the demonstration step.

''' Uses a third party plug in to generate the reg ex.

''' </summary>

''' <remarks></remarks>

Public Sub SetRegEx()

RegExDetails = PassphraseRegex.That.IncludesText(\_demonstrationDetails).ToRegex().Pattern

SetRegExFlags()

End Sub

''' <summary>

''' Updates the reg ex if the user prefers a different pattern.

''' </summary>

''' <param name="newRegEx"></param>

''' <remarks></remarks>

Public Sub UpdateRegEx(newRegEx As String)

RegExDetails = newRegEx

SetRegExFlags()

End Sub

''' <summary>

''' Insert the mark for the step.

''' </summary>

''' <param name="mark"></param>

''' <remarks></remarks>

Public Sub InsertMark(mark As String)

IsValidMark = Integer.TryParse(mark, \_demonstrationStepMark)

InvalidMarkMessage = If(IsValidMark, String.Empty, CommonConstants.NotAValidInteger)

SetCanInsert()

End Sub

''' <summary>

''' Sets the reg ex flags.

''' </summary>

''' <remarks></remarks>

Private Sub SetRegExFlags()

IsValidRegExDetails = Helpers.IsTextNotNull(RegExDetails)

InvalidRegExDetailsMessage = If(IsValidRegExDetails, String.Empty, CommonConstants.NotAValidInput)

End Sub

''' <summary>

''' Sets the can insert flag if all is okay.

''' </summary>

''' <remarks></remarks>

Private Sub SetCanInsert()

Dim arrayOfIsValidTypes(2) As Boolean

arrayOfIsValidTypes(0) = IsValidDemoDetails

arrayOfIsValidTypes(1) = IsValidRegExDetails

arrayOfIsValidTypes(2) = IsValidMark

CanInsertDetailsToList = Helpers.IsThereAFalseInArray(arrayOfIsValidTypes)

End Sub

''' <summary>

''' Adds the new details to the list.

''' </summary>

''' <returns></returns>

''' <remarks></remarks>

Public Function AddDetailsToList() As Boolean

Dim result As Boolean

Dim demoStep = New DemonstrationStep

If (CanInsertDetailsToList) Then

demoStep.StepDetails = \_demonstrationDetails

demoStep.StepRegEx = RegExDetails

demoStep.StepMark = \_demonstrationStepMark

DemonstrationList.Add(demoStep)

AmountOfSteps = DemonstrationList.Count

DemonstrationTotalMarks += \_demonstrationStepMark

result = True

Else

result = False

End If

Return result

End Function

''' <summary>

''' Deletes the selected step.

''' </summary>

''' <param name="stepDetail"></param>

''' <returns></returns>

''' <remarks></remarks>

Public Function RemoveStepFromList(stepDetail As String) As Boolean

Dim result = False

' Check the count.

AmountOfSteps = DemonstrationList.Count

' Delete the step.

DemonstrationList = DemonstrationList \_

.Where(Function(item) item.StepDetails <> stepDetail) \_

.ToList()

If (DemonstrationList.Count < AmountOfSteps) Then

result = True

' Reset the count of steps.

AmountOfSteps = DemonstrationList.Count

End If

Return result

End Function

End Class

End Namespace

##### ExplanationTypeInput.vb

Imports ExampleProg.Constants

Imports ExampleProg.Interfaces

Imports ExampleProg.ProcedureReturnTypes

Namespace InputClasses.NewQuestionTypes

''' <summary>

''' The explanation type input.

''' For validating all of the explanation details entered by the user.

''' </summary>

''' <remarks>For validating the explanations.</remarks>

Public Class ExplanationTypeInput

Private ReadOnly \_dbConnector As IDbConnector

Property ExplanationID As Integer

Property ExplanationTitle As String

Property ExplanationText As String

Property IsNewExplanation As Boolean

Private Property IsValidTitle As Boolean

Private Property IsValidDetail As Boolean

Property InvalidTitleMessage As String

Property InvalidDetailMessage As String

Property CanProceed As Boolean

Property ExplanationList As IEnumerable(Of ExplanationListType)

''' <summary>

''' The basic constructor.

''' </summary>

''' <param name="dbConnector"></param>

''' <remarks></remarks>

Public Sub New(dbConnector As IDbConnector)

\_dbConnector = dbConnector

ExplanationList = \_dbConnector.GetListOfExplanations()

End Sub

''' <summary>

''' The processor for the class.

''' </summary>

''' <param name="title"></param>

''' <param name="text"></param>

''' <remarks></remarks>

Public Sub Process(title As String, text As String)

ExplanationTitle = title

ExplanationID = 0

ExplanationText = text

IsNewExplanation = False

IsValidDetail = False

IsValidTitle = False

InvalidTitleMessage = String.Empty

InvalidDetailMessage = String.Empty

CanProceed = False

SetExplanationTitleFlags()

SetExplanationID()

SetExplanationDetails()

SetExplanationValidFlags()

SetMessageFlags()

End Sub

''' <summary>

''' Sets the title flag.

''' </summary>

''' <remarks></remarks>

Private Sub SetExplanationTitleFlags()

IsValidTitle = Helpers.IsTextNotNull(ExplanationTitle)

End Sub

''' <summary>

''' Sets the explanation id from the explanation set.

''' </summary>

''' <remarks></remarks>

Private Sub SetExplanationID()

ExplanationID = (From explan In ExplanationList

Where explan.ExplanationDetail.ToLower().Equals(ExplanationTitle.ToLower())

Select explan.ID).FirstOrDefault()

If (ExplanationID = 0) Then

IsNewExplanation = True

End If

End Sub

''' <summary>

''' Sets the details for the explantion from the details in the database.

''' </summary>

''' <remarks></remarks>

Private Sub SetExplanationDetails()

If Not (IsNewExplanation) Then

ExplanationText = \_dbConnector.GetExplanationDetailsById(ExplanationID)

End If

End Sub

''' <summary>

''' Sets the explanation valid flags.

''' </summary>

''' <remarks></remarks>

Private Sub SetExplanationValidFlags()

IsValidDetail = Helpers.IsTextNotNull(ExplanationText)

Dim arrayOfIsValidTypes(1) As Boolean

arrayOfIsValidTypes(0) = IsValidTitle

arrayOfIsValidTypes(1) = IsValidDetail

CanProceed = Helpers.IsThereAFalseInArray(arrayOfIsValidTypes)

End Sub

''' <summary>

''' Sets the message flags.

''' </summary>

''' <remarks></remarks>

Private Sub SetMessageFlags()

InvalidTitleMessage = If(IsValidTitle, String.Empty, CommonConstants.NotAValidInput)

InvalidDetailMessage = If(IsValidDetail, String.Empty, CommonConstants.NotAValidInput)

End Sub

End Class

End Namespace

##### KeyStageTypeInput.vb

Imports ExampleProg.Constants

Imports ExampleProg.ProcedureReturnTypes

Namespace InputClasses.NewQuestionTypes

''' <summary>

''' The key stage input class for inserting a new key stage.

''' </summary>

''' <remarks></remarks>

Public Class KeyStageTypeInput

Property KeyStageID As Integer

Property KeyStageText As String

Property IsNewKeyStage As Boolean

Property InvalidMessage As String

Property CanProceed As Boolean

Property KeyStageList As IEnumerable(Of KeyStageListType)

''' <summary>

''' The processor for the class.

''' </summary>

''' <param name="text"></param>

''' <remarks></remarks>

Public Sub Process(text As String)

KeyStageText = text

KeyStageID = 0

IsNewKeyStage = False

InvalidMessage = String.Empty

CanProceed = False

SetKeyStageAndMessageFlags()

SetKeyStageID()

End Sub

''' <summary>

''' Sets the message flags.

''' </summary>

''' <remarks></remarks>

Private Sub SetKeyStageAndMessageFlags()

CanProceed = Helpers.Helpers.IsTextNotNull(KeyStageText)

InvalidMessage = If(CanProceed, String.Empty, CommonConstants.NotAValidInput)

End Sub

''' <summary>

''' Sets the key stage id from the key stage details in the database.

''' </summary>

''' <remarks></remarks>

Private Sub SetKeyStageID()

KeyStageID = (From keystage In KeyStageList

Where keystage.KeyStageDetail.ToLower().Equals(KeyStageText.ToLower())

Select keystage.ID).FirstOrDefault()

If (KeyStageID = 0) Then

IsNewKeyStage = True

End If

End Sub

End Class

End Namespace

##### QuestionTypeInput.vb

Imports ExampleProg.Constants

Imports System.Text.RegularExpressions

Namespace InputClasses.NewQuestionTypes

''' <summary>

''' The question input class.

''' </summary>

''' <remarks></remarks>

Public Class QuestionTypeInput

Property QuestionText As String

Property WebAddressText As String

Property SearchString As String

Private \_subjectText As String

Private \_curriculumText As String

Private \_keyStageText As String

Private \_explanationText As String

Property InvalidQuestionMessage As String

Property InvalidWebAddressMessage As String

Property InvalidSearchStringMessage As String

Property IsValidQuestion As Boolean

Private Property IsValidWebAddress As Boolean

Property IsValidSerchString As Boolean

Property CanProceed As Boolean

''' <summary>

''' The processor for the class.

''' Declares the variables and defaults.

''' Calls the methods to process the new question.

''' </summary>

''' <param name="text"></param>

''' <param name="webAddress"></param>

''' <param name="subject"></param>

''' <param name="curriculum"></param>

''' <param name="keyStage"></param>

''' <param name="explanation"></param>

''' <remarks></remarks>

Public Sub Process(text As String, webAddress As String, subject As String, curriculum As String, keyStage As String, explanation As String)

QuestionText = text

WebAddressText = webAddress

SearchString = String.Empty

\_subjectText = subject

\_curriculumText = curriculum

\_keyStageText = keyStage

\_explanationText = explanation

IsValidQuestion = False

IsValidSerchString = False

SetTextFlag()

SetSearchString()

SetSearchFlag()

SetValidWebAddres()

SetInvalidMessages()

SetCanProceed()

End Sub

''' <summary>

''' Sets the text flag.

''' </summary>

''' <remarks></remarks>

Private Sub SetTextFlag()

IsValidQuestion = Helpers.IsTextNotNull(QuestionText)

End Sub

''' <summary>

''' Sets the search string, and formats it.

''' </summary>

''' <remarks></remarks>

Private Sub SetSearchString()

' Send the inputs to a string formatter.

Dim arrayOfStrings(3) As String

arrayOfStrings(0) = \_subjectText

arrayOfStrings(1) = \_curriculumText

arrayOfStrings(2) = \_keyStageText

arrayOfStrings(3) = \_explanationText

SearchString = Helpers.FormatSearchString(arrayOfStrings).ToString()

End Sub

''' <summary>

''' Sets the search flag in the string is okay.

''' </summary>

''' <remarks></remarks>

Private Sub SetSearchFlag()

Dim arrayOfFlags(3) As Boolean

arrayOfFlags(0) = Helpers.IsTextNotNull(\_subjectText)

arrayOfFlags(1) = Helpers.IsTextNotNull(\_curriculumText)

arrayOfFlags(2) = Helpers.IsTextNotNull(\_keyStageText)

arrayOfFlags(3) = Helpers.IsTextNotNull(\_explanationText)

IsValidSerchString = Helpers.IsThereAFalseInArray(arrayOfFlags)

End Sub

''' <summary>

''' Checks the web addres is a valid address, based on its formatting.

''' </summary>

''' <remarks></remarks>

Private Sub SetValidWebAddres()

' We don't care if it is null.

If (Helpers.IsTextNotNull(WebAddressText)) Then

Dim regexObj As Regex = New Regex(CommonConstants.ValidWebAddresRegEx)

IsValidWebAddress = regexObj.IsMatch(WebAddressText)

Else

IsValidWebAddress = True

End If

End Sub

''' <summary>

''' Sets the invalid messages.

''' </summary>

''' <remarks></remarks>

Private Sub SetInvalidMessages()

InvalidQuestionMessage = If(IsValidQuestion, String.Empty, CommonConstants.NotAValidInput)

InvalidWebAddressMessage = If(IsValidWebAddress, String.Empty, CommonConstants.WebAddressNotValid)

InvalidSearchStringMessage = If(IsValidSerchString, String.Empty, CommonConstants.NotAValidInput)

End Sub

''' <summary>

''' Sets the can proceed flag if all is okay.

''' </summary>

''' <remarks></remarks>

Private Sub SetCanProceed()

Dim arrayOfBools(2) As Boolean

arrayOfBools(0) = IsValidQuestion

arrayOfBools(1) = IsValidWebAddress

arrayOfBools(2) = IsValidSerchString

CanProceed = Helpers.IsThereAFalseInArray(arrayOfBools)

End Sub

End Class

End Namespace

##### SubjectTypeInput.vb

Imports ExampleProg.Constants

Imports ExampleProg.ProcedureReturnTypes

Namespace InputClasses.NewQuestionTypes

''' <summary>

''' The new subject type input class.

''' </summary>

''' <remarks></remarks>

Public Class SubjectTypeInput

Property SubjectID As Integer

Property SubjectText As String

Property IsNewSubject As Boolean

Property InvalidMessage As String

Property CanProceed As Boolean

Property SubjectList As IEnumerable(Of SubjectListType)

''' <summary>

''' The processor for the class.

''' Declares the variables.

''' Calls the methods to set it up.

''' </summary>

''' <param name="text"></param>

''' <remarks></remarks>

Public Sub Process(text As String)

SubjectText = text

SubjectID = 0

IsNewSubject = False

InvalidMessage = String.Empty

CanProceed = False

SetSubjectAndMessageFlags()

SetSubjectID()

End Sub

''' <summary>

''' Sets the falgs.

''' </summary>

''' <remarks></remarks>

Private Sub SetSubjectAndMessageFlags()

CanProceed = Helpers.Helpers.IsTextNotNull(SubjectText)

InvalidMessage = If(CanProceed, String.Empty, CommonConstants.NotAValidInput)

End Sub

''' <summary>

''' Sets the subject id from the details in the database.

''' </summary>

''' <remarks></remarks>

Private Sub SetSubjectID()

SubjectID = (From subject In SubjectList

Where subject.SubjectDetail.ToLower().Equals(SubjectText.ToLower())

Select subject.ID).FirstOrDefault()

If (SubjectID = 0) Then

IsNewSubject = True

End If

End Sub

End Class

End Namespace

#### Training

##### DemonstrationTraining.vb

Imports ExampleProg.Interfaces

Namespace InputClasses.Training

''' <summary>

''' The Demonstration stage of the training section.

''' </summary>

''' <remarks>Needs to keep track of the question ID and pass it up for the next stage.</remarks>

Public Class DemonstrationTraining

Private Property DBConnection As IDbConnector

Private \_subjectID As Integer

Private \_curriculumID As Integer

Private \_keyStageID As Integer

Private \_explanationID As Integer

Property QuestionID As Integer

Private \_questionsList As IEnumerable(Of Integer)

Private \_demonstrationList As IEnumerable(Of String)

Property DemonstrationStep As Integer

Property DemonstrationStepDetails As String

Property CountOfDemonstrationSteps As Integer

Property HasDemoStepsBeenSetUp As Boolean

''' <summary>

''' The constructor.

''' </summary>

''' <param name="dbConnector">The database connection.</param>

''' <remarks>Sets the variables.</remarks>

Public Sub New(dbConnector As IDbConnector)

DBConnection = dbConnector

End Sub

''' <summary>

''' Initialises the variables.

''' </summary>

''' <param name="subjectId"></param>

''' <param name="curriculumId"></param>

''' <param name="keyStageId"></param>

''' <param name="explanationId"></param>

''' <remarks></remarks>

Public Sub Initialise(subjectId As Integer, curriculumId As Integer, keyStageId As Integer, explanationId As Integer)

\_subjectID = subjectId

\_curriculumID = curriculumId

\_keyStageID = keyStageId

\_explanationID = explanationId

HasDemoStepsBeenSetUp = False

End Sub

''' <summary>

''' Set the random question id that will be used for the rest of the steps.

''' </summary>

''' <remarks>Finds the question id for the demo we are going to use.</remarks>

Public Sub SetRandomQuestionID()

' First we need to get a list of question ID's based on the other variables.

\_questionsList = DBConnection.GetListOfQuestionIdsForTraining(\_subjectID, \_curriculumID, \_keyStageID, \_explanationID)

' Then we need to randomise the list and select only 1.

Dim randomiser As New Random()

Dim randomId = randomiser.Next(0, \_questionsList.Count())

' We then set that to the QuestionID.

QuestionID = \_questionsList.ElementAt(randomId)

End Sub

''' <summary>

''' Sets up the demonstration list, steps and details ready for training.

''' </summary>

''' <remarks></remarks>

Public Sub SetUpDemoSteps()

' Check that QuestionID has been set.

If (QuestionID > 0) Then

\_demonstrationList = DBConnection.GetListOfDemoStepDetails(QuestionID)

If (\_demonstrationList.Any()) Then

CountOfDemonstrationSteps = \_demonstrationList.Count()

DemonstrationStep = 0

DemonstrationStepDetails = \_demonstrationList.ElementAt(DemonstrationStep)

HasDemoStepsBeenSetUp = True

End If

Else

HasDemoStepsBeenSetUp = False

End If

End Sub

''' <summary>

''' Updating the step details for the next one in the list.

''' </summary>

''' <remarks></remarks>

Public Sub NextStep()

If (DemonstrationStep < (CountOfDemonstrationSteps - 1)) Then

DemonstrationStep = DemonstrationStep + 1

DemonstrationStepDetails = \_demonstrationList.ElementAt(DemonstrationStep)

End If

End Sub

''' <summary>

''' Updating the step details for the previous one in the list.

''' </summary>

''' <remarks></remarks>

Public Sub PreviousStep()

If (DemonstrationStep > 0) Then

DemonstrationStep = DemonstrationStep - 1

DemonstrationStepDetails = \_demonstrationList.ElementAt(DemonstrationStep)

End If

End Sub

''' <summary>

''' Checking to see if we have viewed all of the demo steps.

''' </summary>

''' <returns>True or False.</returns>

''' <remarks></remarks>

Public Function CanProceed() As Boolean

Dim result As Boolean

If (DemonstrationStep = (CountOfDemonstrationSteps - 1)) Then

result = True

Else

result = False

End If

Return result

End Function

End Class

End Namespace

##### ExplanationTraining.vb

Imports ExampleProg.Constants

Imports ExampleProg.Interfaces

Namespace InputClasses.Training

''' <summary>

''' Gets the explanation details for the training.

''' </summary>

''' <remarks>A simple class.</remarks>

Public Class ExplanationTraining

Property ExplanationTitle As String

Property ExplanationDetails As String

Private Property DBConnection As IDbConnector

''' <summary>

''' Basic constructor for the training explanation class.

''' </summary>

''' <param name="explanationID">The id of the explanation to be displayed.</param>

''' <param name="dbConnector">The database connection.</param>

''' <remarks>Sets the title and details based on the explanation id.</remarks>

Public Sub New(explanationID As Integer, dbConnector As IDbConnector)

DBConnection = dbConnector

If (explanationID > 0) Then

SetExplanationDetailsFromId(explanationID)

Else

ExplanationTitle = CommonConstants.TrainingExplanationIdNotSuppliedError

End If

End Sub

''' <summary>

''' Gets the details from the database and sets the properties.

''' </summary>

''' <param name="explanationID">The id of the explanation to be displayed.</param>

''' <remarks>Sets the details of the explanation.</remarks>

Private Sub SetExplanationDetailsFromId(explanationID As Integer)

Dim explanationDetailsType = DBConnection.GetExplanationDetailsForTraining(explanationID)

ExplanationTitle = explanationDetailsType.Title

ExplanationDetails = explanationDetailsType.DescriptionOfExplanation

End Sub

End Class

End Namespace

##### ImmitationTraining.vb

Imports ExampleProg.Interfaces

Imports ExampleProg.ProcedureReturnTypes

Imports System.Text.RegularExpressions

Namespace InputClasses.Training

''' <summary>

''' The immitation section of training.

''' </summary>

''' <remarks></remarks>

Public Class ImmitationTraining

Private ReadOnly \_dbConnection As IDbConnector

Private \_questionID As Integer

Property DemonstrationList As IEnumerable(Of ImmitationStageListType)

Property DemonstrationStep As Integer

Property DemonstrationStepDetails As String

Property DemonstrationStepRegEx As String

Property DemonstrationStepMark As Integer

Property CountOfDemonstrationSteps As Integer

Property HasDemoStepsBeenSetUp As Boolean

Property DemoInput As String

Property IsDemoInputCorrect As Boolean

Property HasImmitationBeenCompleted As Boolean

''' <summary>

''' The basic constructor.

''' </summary>

''' <param name="dbConnection">The db connection.</param>

''' <remarks></remarks>

Public Sub New(dbConnection As IDbConnector)

\_dbConnection = dbConnection

HasImmitationBeenCompleted = False

End Sub

''' <summary>

''' Gets the demo steps for the immitation stage.

''' </summary>

''' <remarks></remarks>

Public Sub GetDemonstrationList(questionID As Integer)

\_questionID = questionID

If (\_questionID > 0) Then

\_DemonstrationList = \_dbConnection.GetImmitationStageSteps(\_questionID)

HasDemoStepsBeenSetUp = SetImmitationStage()

Else

HasDemoStepsBeenSetUp = False

End If

End Sub

''' <summary>

''' Sets the items from the entry in the list.

''' </summary>

''' <returns>If possible, it returns true, else false.</returns>

''' <remarks></remarks>

Private Function SetImmitationStage() As Boolean

Dim result As Boolean

If (\_demonstrationList.Any()) Then

CountOfDemonstrationSteps = \_demonstrationList.Count()

DemonstrationStep = 0

SetImmitationValues(DemonstrationStep)

result = True

Else

result = False

End If

Return result

End Function

''' <summary>

''' Processes the input from the user.

''' </summary>

''' <param name="userInput">The string input from the user.</param>

''' <remarks></remarks>

Public Sub ProcessInput(userInput As String)

DemoInput = userInput

If (HasDemoStepsBeenSetUp) Then

IsDemoInputCorrect = Regex.IsMatch(DemoInput, DemonstrationStepRegEx)

If (IsDemoInputCorrect) Then

NextStep()

End If

End If

End Sub

''' <summary>

''' Sets up the next step of immitation.

''' </summary>

''' <remarks></remarks>

Private Sub NextStep()

If (DemonstrationStep < (CountOfDemonstrationSteps - 1)) Then

DemonstrationStep = DemonstrationStep + 1

SetImmitationValues(DemonstrationStep)

Else

HasImmitationBeenCompleted = True

End If

End Sub

''' <summary>

''' Sets up the items based on the current step.

''' </summary>

''' <param name="currentStep">The specific step in the list.</param>

''' <remarks></remarks>

Private Sub SetImmitationValues(currentStep As Integer)

DemonstrationStepDetails = \_demonstrationList.ElementAt(currentStep).DescriptionOfStage

DemonstrationStepRegEx = \_demonstrationList.ElementAt(currentStep).RegEx

DemonstrationStepMark = \_demonstrationList.ElementAt(currentStep).StageMark

End Sub

End Class

End Namespace

##### SelectTrainingInput.vb

Imports ExampleProg.Interfaces

Imports ExampleProg.ProcedureReturnTypes

Namespace InputClasses.Training

''' <summary>

''' Select the training from the list of options.

''' </summary>

''' <remarks></remarks>

Public Class SelectTrainingInput

Implements ISelectTrainingInput

Property SubjectList As IEnumerable(Of SubjectListType) Implements ISelectTrainingInput.SubjectList

Property CurriculumList As IEnumerable(Of CurriculumListType) Implements ISelectTrainingInput.CurriculumList

Property KeyStageList As IEnumerable(Of KeyStageListType) Implements ISelectTrainingInput.KeyStageList

Property ExplanationList As IEnumerable(Of ExplanationListType) Implements ISelectTrainingInput.ExplanationList

Private Property DBConnection As IDbConnector

Property SelectedSubjectID As Integer Implements ISelectTrainingInput.SelectedSubjectID

Property SelectedCurriculumID As Integer Implements ISelectTrainingInput.SelectedCurriculumID

Property SelectedKeyStageID As Integer Implements ISelectTrainingInput.SelectedKeyStageID

Property SelectedExplanationID As Integer Implements ISelectTrainingInput.SelectedExplanationID

Property SelectedSubjectText As String Implements ISelectTrainingInput.SelectedSubjectText

Property SelectedCurriculumText As String Implements ISelectTrainingInput.SelectedCurriculumText

Property SelectedKeyStageText As String Implements ISelectTrainingInput.SelectedKeyStageText

Property SelectedExplanationText As String Implements ISelectTrainingInput.SelectedExplanationText

''' <summary>

''' The basic constructor.

''' </summary>

''' <param name="dbConnector"></param>

''' <remarks></remarks>

Public Sub New(dbConnector As IDbConnector)

DBConnection = dbConnector

InitialiseVariables()

PopulateSubjectList()

PopulateKeyStageList()

End Sub

''' <summary>

''' Initalises all the variables.

''' </summary>

''' <remarks></remarks>

Private Sub InitialiseVariables()

SelectedSubjectID = 0

SelectedCurriculumID = 0

SelectedKeyStageID = 0

SelectedExplanationID = 0

End Sub

#Region "Set up lists"

''' <summary>

''' Gets the list of subjects from the database.

''' </summary>

''' <remarks></remarks>

Public Sub PopulateSubjectList() Implements ISelectTrainingInput.PopulateSubjectList

SubjectList = DBConnection.GetListOfSubjects()

End Sub

''' <summary>

''' Gets the list of curriculum from the database.

''' </summary>

''' <remarks></remarks>

Public Sub PopulateCurriculumList() Implements ISelectTrainingInput.PopulateCurriculumList

CurriculumList = DBConnection.GetListOfCurriculum()

End Sub

''' <summary>

''' Gets the list of key stages from the database.

''' </summary>

''' <remarks></remarks>

Public Sub PopulateKeyStageList() Implements ISelectTrainingInput.PopulateKeyStageList

KeyStageList = DBConnection.GetListOfKeyStages()

End Sub

''' <summary>

''' Gets the list of explanations from the database.

''' </summary>

''' <remarks></remarks>

Public Sub PopulateExplanationList() Implements ISelectTrainingInput.PopulateExplanationList

ExplanationList = DBConnection.GetListOfExplanations()

End Sub

''' <summary>

''' Gets the updated curriculum list from the subject selected.

''' </summary>

''' <remarks></remarks>

Public Sub PopulateUpdatedCurriculumList() Implements ISelectTrainingInput.PopulateUpdatedCurriculumList

CurriculumList = DBConnection.GetUpdatedListOfCurriculum(SelectedSubjectID)

End Sub

''' <summary>

''' Gets the updated explanation list from the curriculum selected.

''' </summary>

''' <remarks></remarks>

Public Sub PopulateUpdatedExplanationList() Implements ISelectTrainingInput.PopulateUpdatedExplanationList

ExplanationList = DBConnection.GetUpdatedExplanationsByCurriculum(SelectedCurriculumID)

End Sub

#End Region

#Region "Set ID's"

''' <summary>

''' Sets the subject id from the details in the database.

''' </summary>

''' <remarks></remarks>

Public Sub SetSubjectID() Implements ISelectTrainingInput.SetSubjectID

SelectedSubjectID = (From subject In SubjectList

Where subject.SubjectDetail.ToLower().Equals(SelectedSubjectText.ToLower())

Select subject.ID).FirstOrDefault()

End Sub

''' <summary>

''' Sets the curriculum id from the details in the database.

''' </summary>

''' <remarks></remarks>

Public Sub SetCurriculumID() Implements ISelectTrainingInput.SetCurriculumID

SelectedCurriculumID = (From curric In CurriculumList

Where curric.CurriculumDetails.ToLower().Equals(SelectedCurriculumText.ToLower())

Select curric.ID).FirstOrDefault()

End Sub

''' <summary>

''' Sets the key stage id from the details in the database.

''' </summary>

''' <remarks></remarks>

Public Sub SetKeyStageID() Implements ISelectTrainingInput.SetKeyStageID

SelectedKeyStageID = (From keyStage In KeyStageList

Where keyStage.KeyStageDetail.ToLower().Equals(SelectedKeyStageText.ToLower())

Select keyStage.ID).FirstOrDefault()

End Sub

''' <summary>

''' Sets the explanation id from the details in the database.

''' </summary>

''' <remarks></remarks>

Public Sub SetExplanationID() Implements ISelectTrainingInput.SetExplanationID

SelectedExplanationID = (From explanation In ExplanationList

Where explanation.ExplanationDetail.ToLower().Equals(SelectedExplanationText.ToLower())

Select explanation.ID).FirstOrDefault()

End Sub

#End Region

End Class

End Namespace

#### Logininput.vb

Imports ExampleProg.Constants

Imports ExampleProg.Interfaces

Namespace InputClasses

''' <summary>

''' The Log In Input class.

''' Used to validate the user name and password.

''' ToDo: sort out the encryption of the password.

''' </summary>

''' <remarks>Validates the user name and password.</remarks>

Public Class LogInInput

Property LoginUserName As String

Property LoginPassword As String

Private Property LogInUserId As Integer

Property IsUserNameValid As Boolean

Property IsUserNameInDatabase As Boolean

Property IsPasswordValid As Boolean

Property DoUserNameAndPasswordMatch As Boolean

Property UserNameInvalidMessage As String

Property PasswordInvalidMessage As String

Property OverallInvalidMessage As String

Private \_arrayOfValidity(3) As Boolean

''' <summary>

''' The validity of the log in details.

''' Sets an array and then checks them.

''' </summary>

''' <param name="p1"></param>

''' <value></value>

''' <returns></returns>

''' <remarks></remarks>

Public Property ArrayOfValidity(p1 As Integer) As Boolean

Get

Return \_arrayOfValidity(p1)

End Get

Private Set(value As Boolean)

\_arrayOfValidity(p1) = value

End Set

End Property

''' <summary>

''' The basic constructor.

''' Declares the variables.

''' Sets the flags.

''' Calls the methods to set everything up.

''' </summary>

''' <param name="username"></param>

''' <param name="password"></param>

''' <remarks></remarks>

Public Sub New(username As String, password As String)

Dim array(3) As Boolean

\_arrayOfValidity(3) = array(3)

LoginUserName = username

LoginPassword = password

LogInUserId = 0

IsUserNameValid = False

IsUserNameInDatabase = False

IsPasswordValid = False

DoUserNameAndPasswordMatch = False

UserNameInvalidMessage = String.Empty

PasswordInvalidMessage = String.Empty

OverallInvalidMessage = String.Empty

End Sub

''' <summary>

''' Sets the basic flags from the input detais.

''' </summary>

''' <remarks></remarks>

Public Sub SetBasicValidStates()

IsUserNameValid = Helpers.Helpers.IsTextNotNull(LoginUserName)

IsPasswordValid = Helpers.Helpers.IsTextNotNull(LoginPassword)

End Sub

''' <summary>

''' Checks if the user name is already in the database.

''' </summary>

''' <param name="dbConnector"></param>

''' <remarks></remarks>

Public Sub SetUserNameInDatabase(dbConnector As IDbConnector)

IsUserNameInDatabase = dbConnector.DoesUserNameAlreadyExists(LoginUserName)

End Sub

''' <summary>

''' Checks if the user name and password match what is in the database.

''' </summary>

''' <param name="dbConnector"></param>

''' <remarks></remarks>

Public Sub SetDoUserNameAndPasswordMatch(dbConnector As IDbConnector)

LogInUserId = dbConnector.GetUserIdOfValidUser(LoginUserName, LoginPassword)

If (LogInUserId <> 0) Then

DoUserNameAndPasswordMatch = True

End If

End Sub

''' <summary>

''' Sets the array from the boolean flags.

''' </summary>

''' <remarks></remarks>

Public Sub SetArrayOfValidStates()

ArrayOfValidity(0) = IsUserNameValid

ArrayOfValidity(1) = IsUserNameInDatabase

ArrayOfValidity(2) = IsPasswordValid

ArrayOfValidity(3) = DoUserNameAndPasswordMatch

End Sub

''' <summary>

''' Sets the invalid messages.

''' </summary>

''' <remarks></remarks>

Public Sub SetInvalidMessages()

UserNameInvalidMessage = If((IsUserNameValid Or IsUserNameInDatabase), String.Empty, CommonConstants.NotAValidInput)

PasswordInvalidMessage = If(IsPasswordValid, String.Empty, CommonConstants.NotAValidInput)

OverallInvalidMessage = If(DoUserNameAndPasswordMatch, String.Empty, CommonConstants.UserNameAndPasswordDoNotMatch)

End Sub

''' <summary>

''' Checks to find any false flags in the array.

''' </summary>

''' <returns></returns>

''' <remarks></remarks>

Public Function IsThereAFalseInTheInput() As Boolean

Dim testArray(3) As Boolean

testArray(0) = ArrayOfValidity(0)

testArray(1) = ArrayOfValidity(1)

testArray(2) = ArrayOfValidity(2)

testArray(3) = ArrayOfValidity(3)

Return Helpers.Helpers.IsThereAFalseInArray(testArray)

End Function

End Class

End Namespace

#### NewQuestionInput.vb

Imports ExampleProg.InputClasses.NewQuestionTypes

Imports ExampleProg.Interfaces

Imports System.Threading

Imports ExampleProg.Classes

Imports ExampleProg.Constants

Imports ExampleProg.QuestionTypes

Namespace InputClasses

''' <summary>

''' The New Question Input class.

''' Uses an interface, so it can be mocked for testing.

''' Instantiates all of the different new question types.

''' Validates using the different types and then inserts the new question into the database.

''' </summary>

''' <remarks>The New Question Input class.</remarks>

Public Class NewQuestionInput

Implements INewQuestionInput

#Region "Properties"

Public Property SubjectType As SubjectTypeInput Implements INewQuestionInput.SubjectType

Public Property CurriculumType As CurriculumTypeInput Implements INewQuestionInput.CurriculumType

Public Property KeyStageType As KeyStageTypeInput Implements INewQuestionInput.KeyStageType

Public Property ExplanationType As ExplanationTypeInput Implements INewQuestionInput.ExplanationType

Public Property QuestionType As QuestionTypeInput Implements INewQuestionInput.QuestionType

Public Property AnswerType As AnswerTypeInput Implements INewQuestionInput.AnswerType

Public Property DemonstrationType As DemonstrationTypeInput Implements INewQuestionInput.DemonstrationType

Private Property DBConnection As IDbConnector

Public Property CanSubmit As Boolean Implements INewQuestionInput.CanSubmit

Public Property NewQuestionInsertMessage As String Implements INewQuestionInput.NewQuestionInsertMessage

Public Property InsertedSuccessfully As Boolean Implements INewQuestionInput.InsertedSuccessfully

#End Region

#Region "Constructor and setup of lists"

''' <summary>

''' The basic constructor.

''' </summary>

''' <param name="dbConnector"></param>

''' <remarks></remarks>

Public Sub New(dbConnector As IDbConnector)

DBConnection = dbConnector

SetInputTypes()

End Sub

''' <summary>

''' Sets up the different input classes.

''' </summary>

''' <remarks></remarks>

Private Sub SetInputTypes()

SubjectType = New SubjectTypeInput()

CurriculumType = New CurriculumTypeInput()

KeyStageType = New KeyStageTypeInput()

ExplanationType = New ExplanationTypeInput(DBConnection)

QuestionType = New QuestionTypeInput()

AnswerType = New AnswerTypeInput()

DemonstrationType = New DemonstrationTypeInput()

DemonstrationType.ResetValuesToDefaults()

CanSubmit = False

NewQuestionInsertMessage = String.Empty

InsertedSuccessfully = False

End Sub

''' <summary>

''' Sets the lists from the database.

''' Uses multi-threading to populate the lists.

''' :-)

''' </summary>

''' <remarks></remarks>

Public Sub SetListsFromDatabase() Implements INewQuestionInput.SetListsFromDatabase

Dim thread1 As New Thread(AddressOf PopulateSubjectList)

Dim thread2 As New Thread(AddressOf PopulateCurriculumList)

Dim thread3 As New Thread(AddressOf PopulateKeyStageList)

Dim thread4 As New Thread(AddressOf PopulateExplanationList)

thread1.Start()

thread2.Start()

thread3.Start()

thread4.Start()

thread1.Join()

thread2.Join()

thread3.Join()

thread4.Join()

End Sub

''' <summary>

''' Populates the subject list from thread 1.

''' </summary>

''' <remarks></remarks>

Private Sub PopulateSubjectList()

SubjectType.SubjectList = DBConnection.GetListOfSubjects()

End Sub

''' <summary>

''' Populates the curriculum list from thread 2.

''' </summary>

''' <remarks></remarks>

Private Sub PopulateCurriculumList()

CurriculumType.CurriculumList = DBConnection.GetListOfCurriculum()

End Sub

''' <summary>

''' Populates the key stage list from thread 3.

''' </summary>

''' <remarks></remarks>

Private Sub PopulateKeyStageList()

KeyStageType.KeyStageList = DBConnection.GetListOfKeyStages()

End Sub

''' <summary>

''' Populates the explanation list from thread 4.

''' </summary>

''' <remarks></remarks>

Private Sub PopulateExplanationList()

ExplanationType.ExplanationList = DBConnection.GetListOfExplanations()

End Sub

#End Region

#Region "Subject Changes"

''' <summary>

''' Updates the lists from the subjects.

''' Uses multi-threading again.

''' </summary>

''' <remarks></remarks>

Public Sub UpdateOtherLists() Implements INewQuestionInput.UpdateOtherLists

Dim thread1 As New Thread(AddressOf UpdateCurriculumList)

Dim thread2 As New Thread(AddressOf UpdateExplanationList)

thread1.Start()

thread2.Start()

thread1.Join()

thread2.Join()

End Sub

''' <summary>

''' Updates the curriculum list.

''' </summary>

''' <remarks></remarks>

Private Sub UpdateCurriculumList()

CurriculumType.CurriculumList = DBConnection.GetUpdatedListOfCurriculum(SubjectType.SubjectID)

End Sub

''' <summary>

''' Updates the explanation list.

''' </summary>

''' <remarks></remarks>

Private Sub UpdateExplanationList()

ExplanationType.ExplanationList = DBConnection.GetUpdatedListOfExplanations(SubjectType.SubjectID)

End Sub

#End Region

#Region "Curriculum Changes"

''' <summary>

''' Updates the explanation list from the curriculum selected.

''' </summary>

''' <remarks></remarks>

Public Sub UpdateExplanation() Implements INewQuestionInput.UpdateExplanation

ExplanationType.ExplanationList = DBConnection.GetUpdatedExplanationsByCurriculum(CurriculumType.CurriculumID)

End Sub

#End Region

''' <summary>

''' Process the details in the class.

''' </summary>

''' <remarks></remarks>

Public Sub Process() Implements INewQuestionInput.Process

SetCanSubmit()

End Sub

''' <summary>

''' Sets the can submit flag from the other class can proceed flags.

''' All need to be true for the can submit flag to be set to true.

''' Otherwise the new question will not be created.

''' </summary>

''' <remarks></remarks>

Private Sub SetCanSubmit()

Dim arrayOfFlags(6) As Boolean

arrayOfFlags(0) = SubjectType.CanProceed

arrayOfFlags(1) = CurriculumType.CanProceed

arrayOfFlags(2) = KeyStageType.CanProceed

arrayOfFlags(3) = ExplanationType.CanProceed

arrayOfFlags(4) = QuestionType.CanProceed

arrayOfFlags(5) = AnswerType.CanProceed

arrayOfFlags(6) = DemonstrationType.CanInsertDetailsToList

CanSubmit = Helpers.IsThereAFalseInArray(arrayOfFlags)

End Sub

''' <summary>

''' Inserts the new question into the database with all of the demo steps.

''' </summary>

''' <remarks>Inserts the question.</remarks>

Public Sub InsertNewQuestion() Implements INewQuestionInput.InsertNewQuestion

' Check if parts are new, and insert them.

InsertNewQuestionTypes()

' Check that we do not have any nulls or 0 in any of the IDs.

Dim canContiune As Boolean = SetCanContiune()

If (canContiune) Then

Dim question = SetUpQuestionInsertTypes()

' Insert the new question first and get the id of the record inserted.

Dim questionID = DBConnection.InsertNewQuestion(question)

Dim isQuestionValid = Helpers.IsDigitNotNullOrZero(questionID)

If (isQuestionValid) Then

Dim insertedOkay As Boolean = GetInsertedOkay(questionID)

If Not (insertedOkay) Then

NewQuestionInsertMessage = CommonConstants.InsertNewDemoStepError

Else

NewQuestionInsertMessage = CommonConstants.NewQuestionInsertedCorrectly

InsertedSuccessfully = True

End If

Else

NewQuestionInsertMessage = CommonConstants.InsertNewQuestionError

End If

Else

NewQuestionInsertMessage = CommonConstants.InsertNewQuestionError

End If

End Sub

''' <summary>

''' Inserts the new subjects, curriculums, key stage and explanation.

''' If they are not new, it keeps the original ID's set before.

''' </summary>

''' <remarks>For setting the ID's.</remarks>

Private Sub InsertNewQuestionTypes()

If (SubjectType.IsNewSubject) Then

SubjectType.SubjectID = DBConnection.InsertNewSubject(SubjectType.SubjectText)

End If

If (CurriculumType.IsNewCurriculum) Then

CurriculumType.CurriculumID = DBConnection.InsertNewCurriculum(SubjectType.SubjectID, CurriculumType.CurriculumText)

End If

If (KeyStageType.IsNewKeyStage) Then

KeyStageType.KeyStageID = DBConnection.InsertNewKeyStage(KeyStageType.KeyStageText)

End If

If (ExplanationType.IsNewExplanation) Then

ExplanationType.ExplanationID = DBConnection.InsertNewExplanation(CurriculumType.CurriculumID, ExplanationType.ExplanationTitle, ExplanationType.ExplanationText)

End If

End Sub

''' <summary>

''' Checks that all of the ID's are set.

''' </summary>

''' <returns>If OK to carry on.</returns>

''' <remarks>Getting ready for the rest of the insert steps.</remarks>

Private Function SetCanContiune() As Boolean

Dim arrayOfFlags(3) As Boolean

arrayOfFlags(0) = Helpers.IsDigitNotNullOrZero(SubjectType.SubjectID)

arrayOfFlags(1) = Helpers.IsDigitNotNullOrZero(CurriculumType.CurriculumID)

arrayOfFlags(2) = Helpers.IsDigitNotNullOrZero(KeyStageType.KeyStageID)

arrayOfFlags(3) = Helpers.IsDigitNotNullOrZero(ExplanationType.ExplanationID)

Return Helpers.IsThereAFalseInArray(arrayOfFlags)

End Function

''' <summary>

''' Sets up the variables into the database user defined type.

''' </summary>

''' <returns>The question insert type.</returns>

''' <remarks>Sets it all up from all of the different objects.</remarks>

Private Function SetUpQuestionInsertTypes() As QuestionInsertType

Dim newQuestion = New QuestionInsertType

newQuestion.SelectedSubjectID = SubjectType.SubjectID

newQuestion.SelectedCurriculumID = CurriculumType.CurriculumID

newQuestion.SelectedKeyStageID = KeyStageType.KeyStageID

newQuestion.SelectedExplanationID = ExplanationType.ExplanationID

newQuestion.NewQuestion = QuestionType.QuestionText

newQuestion.NewAnswer = AnswerType.CorrectAnswer

newQuestion.AmountOfStepsSet = DemonstrationType.AmountOfSteps

newQuestion.TotalMarksSet = DemonstrationType.DemonstrationTotalMarks

newQuestion.NewFalseAnswerA = AnswerType.InCorrectAnswerA

newQuestion.NewFalseAnswerB = AnswerType.InCorrectAnswerB

newQuestion.NewFalseAnswerC = AnswerType.InCorrectAnswerC

newQuestion.NewSearchString = QuestionType.SearchString

newQuestion.NewWebLink = QuestionType.WebAddressText

Return newQuestion

End Function

''' <summary>

''' Inserts the demo steps into the database.

''' Keeps a records of any false returns.

''' Then goes through that list to return true or false.

''' </summary>

''' <param name="questionID"></param>

''' <returns>True or false.</returns>

''' <remarks>Sets the demo steps and checks all is OK.</remarks>

Private Function GetInsertedOkay(questionID As Integer) As Boolean

' Setting up initial checks.

Dim insertedList = New List(Of Boolean)

'Dim isQuestionValid = Helpers.IsDigitNotNullOrZero(questionID)

Dim isDemoStepsCountValid = Helpers.IsDigitNotNullOrZero(DemonstrationType.DemonstrationList.Count)

'insertedList.Add(isQuestionValid)

insertedList.Add(isDemoStepsCountValid)

' Now for the demo inserts.

If (isDemoStepsCountValid) Then

' Insert all of the demo steps using the id of the new question.

For Each demo As DemonstrationStep In DemonstrationType.DemonstrationList

Dim demoStep = New DemonstrationStepType

demoStep.DemonstrationStageInsert = demo.StepDetails

demoStep.RegExInsert = demo.StepRegEx

demoStep.StageMarkInsert = demo.StepMark

insertedList.Add(DBConnection.InsertNewDemoStep(demoStep, questionID))

Next

End If

' So now we check that all went ok...

Return Helpers.IsThereAFalseInArray(insertedList.ToArray())

End Function

End Class

End Namespace

#### NewUserInput.vb

Imports ExampleProg.Constants

Imports ExampleProg.Interfaces

Namespace InputClasses

''' <summary>

''' The New User Input class.

''' Validates all of the inputs for a new user.

''' </summary>

''' <remarks>The New User class.</remarks>

Public Class NewUserInput

Property NewUserName As String

Property NewPassword As String

Property NewFirstName As String

Property NewMiddleName As String

Property NewLastName As String

Property IsUserNameValid As Boolean

Property IsUserAvailable As Boolean

Property IsPasswordValid As Boolean

Property IsFirstNameValid As Boolean

Property IsMiddleNameValid As Boolean

Property IsLastNameValid As Boolean

Property UserNameInvalidMessage As String

Property PasswordInvalidMessage As String

Property FirstNameInvalidMessage As String

Property MiddleNameInvalidmessage As String

Property LastNameInvalidMessage As String

Private ReadOnly \_arrayOfValidity(5) As Boolean

''' <summary>

''' The validity array property.

''' </summary>

''' <param name="p1"></param>

''' <value></value>

''' <returns></returns>

''' <remarks></remarks>

Public Property ArrayOfValidity(p1 As Integer) As Boolean

Get

Return \_arrayOfValidity(p1)

End Get

Private Set(value As Boolean)

\_arrayOfValidity(p1) = value

End Set

End Property

''' <summary>

''' The basic constructor.

''' Declares all the variables.

''' Sets the default flags.

''' calls the other methods to create the new user.

''' </summary>

''' <param name="userName"></param>

''' <param name="password"></param>

''' <param name="firstName"></param>

''' <param name="middleName"></param>

''' <param name="lastName"></param>

''' <remarks></remarks>

Public Sub New(userName As String, password As String, firstName As String, middleName As String, lastName As String)

Dim array(5) As Boolean

\_arrayOfValidity(5) = array(5)

NewUserName = userName

NewPassword = password

NewFirstName = firstName

NewMiddleName = middleName

NewLastName = lastName

IsUserNameValid = False

IsUserAvailable = False

IsPasswordValid = False

IsFirstNameValid = False

IsMiddleNameValid = True

IsLastNameValid = False

UserNameInvalidMessage = String.Empty

PasswordInvalidMessage = String.Empty

FirstNameInvalidMessage = String.Empty

MiddleNameInvalidmessage = String.Empty

LastNameInvalidMessage = String.Empty

End Sub

''' <summary>

''' Sets the flags from the user inputs.

''' </summary>

''' <remarks></remarks>

Public Sub SetValidStates()

IsUserNameValid = Helpers.Helpers.IsTextNotNull(NewUserName)

IsPasswordValid = Helpers.Helpers.IsTextNotNull(NewPassword)

IsFirstNameValid = Helpers.Helpers.IsTextNotNull(NewFirstName)

IsLastNameValid = Helpers.Helpers.IsTextNotNull(NewLastName)

End Sub

''' <summary>

''' Checks if the user name is already used.

''' </summary>

''' <param name="dbConnector"></param>

''' <remarks></remarks>

Public Sub SetUserNameIsUsed(dbConnector As IDbConnector)

IsUserAvailable = dbConnector.IsUserNameAvailable(NewUserName)

End Sub

''' <summary>

''' Set the array from the flags.

''' </summary>

''' <remarks></remarks>

Public Sub SetArrayOfValidStates()

ArrayOfValidity(0) = IsUserNameValid

ArrayOfValidity(1) = IsUserAvailable

ArrayOfValidity(2) = IsPasswordValid

ArrayOfValidity(3) = IsFirstNameValid

ArrayOfValidity(4) = IsMiddleNameValid

ArrayOfValidity(5) = IsLastNameValid

End Sub

''' <summary>

''' Sets the invalid messages.

''' </summary>

''' <remarks></remarks>

Public Sub SetInvalidMessages()

UserNameInvalidMessage = If(IsUserNameValid, If(IsUserAvailable, CommonConstants.UserIsAvailable, CommonConstants.UserAlreadyExists), CommonConstants.NotAValidInput)

PasswordInvalidMessage = If(IsPasswordValid, String.Empty, CommonConstants.NotAValidInput)

FirstNameInvalidMessage = If(IsFirstNameValid, String.Empty, CommonConstants.NotAValidInput)

LastNameInvalidMessage = If(IsLastNameValid, String.Empty, CommonConstants.NotAValidInput)

End Sub

''' <summary>

''' Checks if the array is valid.

''' </summary>

''' <returns></returns>

''' <remarks></remarks>

Public Function IsThereAFalseInTheInput() As Boolean

Dim testArray(5) As Boolean

testArray(0) = ArrayOfValidity(0)

testArray(1) = ArrayOfValidity(1)

testArray(2) = ArrayOfValidity(2)

testArray(3) = ArrayOfValidity(3)

testArray(4) = ArrayOfValidity(4)

testArray(5) = ArrayOfValidity(5)

Return Helpers.Helpers.IsThereAFalseInArray(testArray)

End Function

End Class

End Namespace

### Interfaces

#### IDbConnector.vb

Imports ExampleProg.Classes

Imports ExampleProg.ProcedureReturnTypes

Imports ExampleProg.QuestionTypes

Namespace Interfaces

''' <summary>

''' Database Connector Wrapper Interface.

''' </summary>

''' <remarks>Database Interface.</remarks>

Public Interface IDbConnector

Function DBConnectionStatus() As String

Function IsUserNameAvailable(userName As String) As Boolean

Function DoesUserNameAlreadyExists(userName As String) As Boolean

Function GetUserIdOfValidUser(userName As String, password As String) As Integer

Function GetUserDetails(user As IUserClass) As UserClass

Function CreateNewUser(user As IUserClass) As Boolean

Function GetListOfSubjects() As IEnumerable(Of SubjectListType)

Function GetListOfCurriculum() As IEnumerable(Of CurriculumListType)

Function GetListOfKeyStages() As IEnumerable(Of KeyStageListType)

Function GetListOfExplanations() As IEnumerable(Of ExplanationListType)

Function GetUpdatedListOfCurriculum(subjectID As Integer) As IEnumerable(Of CurriculumListType)

Function GetUpdatedListOfExplanations(subjectID As Integer) As IEnumerable(Of ExplanationListType)

Function GetUpdatedExplanationsByCurriculum(curriculumID As Integer) As IEnumerable(Of ExplanationListType)

Function GetExplanationDetailsById(explanationID As Integer) As String

Function InsertNewSubject(subjectText As String) As Integer

Function InsertNewCurriculum(subjectID As Integer, curriculumText As String) As Integer

Function InsertNewKeyStage(keyStageText As String) As Integer

Function InsertNewExplanation(curriculumID As Integer, explanationTitle As String, explanationText As String) As Integer

Function InsertNewQuestion(question As QuestionInsertType) As Integer

Function InsertNewDemoStep(demo As DemonstrationStepType, questionID As Integer) As Boolean

Function GetExplanationDetailsForTraining(explanationID As Integer) As GetExplanationDetailsType

Function GetListOfQuestionIdsForTraining(subjectID As Integer, curriculumID As Integer, keyStageID As Integer, explanationID As Integer) As IEnumerable(Of Integer)

Function GetListOfDemoStepDetails(questionID As Integer) As IEnumerable(Of String)

Function GetImmitationStageSteps(questionID As Integer) As IEnumerable(Of ImmitationStageListType)

End Interface

End Namespace

#### INewQuestionInput.vb

Imports ExampleProg.InputClasses.NewQuestionTypes

Namespace Interfaces

''' <summary>

''' The New Question Input Interface.

''' </summary>

''' <remarks>New Question Input Interface.</remarks>

Public Interface INewQuestionInput

Property SubjectType As SubjectTypeInput

Property CurriculumType As CurriculumTypeInput

Property KeyStageType As KeyStageTypeInput

Property ExplanationType As ExplanationTypeInput

Property QuestionType As QuestionTypeInput

Property AnswerType As AnswerTypeInput

Property DemonstrationType As DemonstrationTypeInput

Property CanSubmit As Boolean

Property NewQuestionInsertMessage As String

Property InsertedSuccessfully As Boolean

Sub SetListsFromDatabase()

Sub UpdateOtherLists()

Sub UpdateExplanation()

Sub Process()

Sub InsertNewQuestion()

End Interface

End Namespace

#### ISelectTrainingInput.vb

Imports ExampleProg.ProcedureReturnTypes

Namespace Interfaces

''' <summary>

''' The Select Training Input Interface.

''' </summary>

''' <remarks>Select Training Interface.</remarks>

Public Interface ISelectTrainingInput

Property SubjectList As IEnumerable(Of SubjectListType)

Property CurriculumList As IEnumerable(Of CurriculumListType)

Property KeyStageList As IEnumerable(Of KeyStageListType)

Property ExplanationList As IEnumerable(Of ExplanationListType)

Property SelectedSubjectID As Integer

Property SelectedCurriculumID As Integer

Property SelectedKeyStageID As Integer

Property SelectedExplanationID As Integer

Property SelectedSubjectText As String

Property SelectedCurriculumText As String

Property SelectedKeyStageText As String

Property SelectedExplanationText As String

Sub PopulateSubjectList()

Sub PopulateCurriculumList()

Sub PopulateKeyStageList()

Sub PopulateExplanationList()

Sub PopulateUpdatedCurriculumList()

Sub PopulateUpdatedExplanationList()

Sub SetSubjectID()

Sub SetCurriculumID()

Sub SetKeyStageID()

Sub SetExplanationID()

End Interface

End Namespace

#### IUserClass.vb

Namespace Interfaces

''' <summary>

''' The User class Interface.

''' </summary>

''' <remarks>Interface for the User class.</remarks>

Public Interface IUserClass

Property UserId() As Integer

Property UserName() As String

Property Password() As String

Property FirstName() As String

Property MiddleName() As String

Property LastName() As String

Property IsValidUser() As Boolean

Function IsUserNameInDatabase(dbConnection As IDbConnector) As Boolean

Function DoesUserNameAlreadyExists(dbConnection As IDbConnector) As Boolean

Function CreateNewUser(dbConnection As IDbConnector) As Boolean

Function Login(dbConnection As IDbConnector) As Boolean

Sub Logout()

End Interface

End Namespace

### Procedure Return Types

#### CurriculumListType.vb

Namespace ProcedureReturnTypes

Public Class CurriculumListType

Property ID As Integer

Property CurriculumDetails As String

End Class

End Namespace

#### ExplanationListType.vb

Namespace ProcedureReturnTypes

Public Class ExplanationListType

Property ID As Integer

Property ExplanationDetail As String

End Class

End Namespace

#### GetExplanationDetailsType.vb

Namespace ProcedureReturnTypes

''' <summary>

''' To match the values returned by the stored procedure.

''' </summary>

''' <remarks>Procedure return type.</remarks>

Public Class GetExplanationDetailsType

Property ID As Integer

Property Title As String

Property DescriptionOfExplanation As String

End Class

End Namespace

#### ImmitationStageListType.vb

Namespace ProcedureReturnTypes

Public Class ImmitationStageListType

Property DescriptionOfStage As String

Property RegEx As String

Property StageMark As Integer

End Class

End Namespace

#### KeyStageListType.vb

Namespace ProcedureReturnTypes

Public Class KeyStageListType

Property ID As Integer

Property KeyStageDetail As String

End Class

End Namespace

#### SubjectListType.vb

Namespace ProcedureReturnTypes

Public Class SubjectListType

Property ID As Integer

Property SubjectDetail As String

End Class

End Namespace

### Question Types

#### DemonstrationStepType.vb

Namespace QuestionTypes

Public Class DemonstrationStepType

Property DemonstrationStageInsert As String

Property RegExInsert As String

Property StageMarkInsert As Integer

End Class

End Namespace

#### QuestionInsertType.vb

Namespace QuestionTypes

Public Class QuestionInsertType

Property SelectedSubjectID As Integer

Property SelectedCurriculumID As Integer

Property SelectedKeyStageID As Integer

Property SelectedExplanationID As Integer

Property NewQuestion As String

Property NewAnswer As String

Property AmountOfStepsSet As Integer

Property TotalMarksSet As Integer

Property NewFalseAnswerA As String

Property NewFalseAnswerB As String

Property NewFalseAnswerC As String

Property NewSearchString As String

Property NewWebLink As String

End Class

End Namespace

### Views

#### App4Learn.vb

Imports System.Configuration

Imports DapperWrapper

Imports ExampleProg.Classes

Imports ExampleProg.Constants

Imports ExampleProg.DatabaseClasses

Imports ExampleProg.InputClasses

Imports ExampleProg.Interfaces

''' <summary>

''' The main Application View.

''' </summary>

''' <remarks>The main page.</remarks>

Public Class App4Learn

#Region "Properties"

Dim WithEvents \_userClass As IUserClass

Dim WithEvents \_tempUserClass As IUserClass

Dim WithEvents \_dbConnector As IDbConnector

Private \_dbConnectionString As String

#End Region

#Region "Initialize and Load"

''' <summary>

''' The on load method for the page.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Sub Login\_Load(sender As Object, e As EventArgs) Handles MyBase.Load

\_dbConnectionString = ConfigurationManager.ConnectionStrings(DatabaseConstants.ApplicationConfigString).ConnectionString

StatusStrip.Items.Item(1).Text = DatabaseConstants.DBConnecting

Initialize()

End Sub

''' <summary>

''' Setting up all the systems for the page.

''' </summary>

''' <remarks></remarks>

Private Sub Initialize()

InitializeDatabase()

SetupUsers()

RemoveMainTabs()

ClearAllItems()

End Sub

''' <summary>

''' Setting up the database connection.

''' </summary>

''' <remarks></remarks>

Private Sub InitializeDatabase()

Dim sqlExecutorFactory = New SqlExecutorFactory(\_dbConnectionString)

Dim newDatabaseConnection = New DatabaseConnectorWrapper(sqlExecutorFactory)

\_dbConnector = newDatabaseConnection

StatusStrip.Items.Item(1).Text = \_dbConnector.DBConnectionStatus.ToString()

SetStatusColour()

End Sub

''' <summary>

''' Setting up the users for the page.

''' </summary>

''' <remarks></remarks>

Private Sub SetupUsers()

Dim newUser = New UserClass

\_userClass = newUser

Dim newTempUser = New UserClass

\_tempUserClass = newTempUser

End Sub

''' <summary>

''' Setting the status colour based upon the connection status.

''' </summary>

''' <remarks></remarks>

Private Sub SetStatusColour()

StatusStrip.Items(1).ForeColor = If(StatusStrip.Items(1).Text = DatabaseConstants.DBConnectedSuccesful, Color.Green, Color.Red)

End Sub

#End Region

#Region "Login / Logout"

''' <summary>

''' The log in button click method.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Sub LoginButton\_Click(sender As Object, e As EventArgs) Handles LoginButton.Click

' Create a log in class.

Dim loginInput = New LogInInput(UserNameBox.Text, PasswordTextBox.Text)

loginInput.SetBasicValidStates()

loginInput.SetUserNameInDatabase(\_dbConnector)

loginInput.SetDoUserNameAndPasswordMatch(\_dbConnector)

loginInput.SetInvalidMessages()

loginInput.SetArrayOfValidStates()

' Clear any previous User details.

SetupUsers()

ClearLogInInvalidLabels()

UserNameErrorLabel.Text = loginInput.UserNameInvalidMessage

PasswordErrorLabel.Text = loginInput.PasswordInvalidMessage

InvalidLogOnLabel.Text = loginInput.OverallInvalidMessage

' Check if any one is invalid, then fail.

If (loginInput.IsThereAFalseInTheInput()) Then

Dim result As Boolean

' If all is okay, then set User.

' Send details to the database.

\_userClass.UserName = loginInput.LoginUserName

\_userClass.Password = loginInput.LoginPassword

result = \_userClass.Login(\_dbConnector)

If (result) Then

SetLoggedInSettings()

End If

End If

End Sub

''' <summary>

''' The log out button click.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Sub LogOutButton\_Click(sender As Object, e As EventArgs) Handles LogOutButton.Click

ClearAllItems()

\_userClass.Logout()

' Clearing all of the logged in settings.

SetupUsers()

SetLoggedOutSettings()

End Sub

''' <summary>

''' Setting up the logged in settings.

''' Flags.

''' The visible tabs.

''' Clearing the text boxes.

''' </summary>

''' <remarks></remarks>

Private Sub SetLoggedInSettings()

UserNameOfCurrentUser.Text = \_userClass.UserName

MainTabControl.TabPages(0).Text = "Log out"

LoginButton.Visible = False

LogOutButton.Visible = True

AddMainTabs()

MainTabControl.SelectTab(2)

ClearLogInTextBoxes()

End Sub

''' <summary>

''' Setting the logged out systems.

''' Flags.

''' The non-visible tabs.

''' </summary>

''' <remarks></remarks>

Private Sub SetLoggedOutSettings()

UserNameOfCurrentUser.Text = \_userClass.UserName

MainTabControl.TabPages(0).Text = "Log in"

LoginButton.Visible = True

LogOutButton.Visible = False

MainTabControl.SelectTab(0)

RemoveMainTabs()

End Sub

#End Region

#Region "New User"

''' <summary>

''' The new user button click.

''' The process to create a new user.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Sub NewUserButton\_Click(sender As Object, e As EventArgs) Handles NewUserButton.Click

' Create a new user input class.

Dim userInput = New NewUserInput(NewUserNameBox.Text, NewUserPasswordBox.Text, FirstNameBox.Text, MiddleNameBox.Text, LastNameBox.Text)

userInput.SetValidStates()

userInput.SetUserNameIsUsed(\_dbConnector)

userInput.SetInvalidMessages()

userInput.SetArrayOfValidStates()

' Create a new \_tempUserClass.

\_tempUserClass = New UserClass

ClearInvalidLabels()

If (userInput.IsUserAvailable And userInput.IsUserNameValid) Then

NewUserNameValidation.ForeColor = Color.Green

Else

NewUserNameValidation.ForeColor = Color.Red

End If

NewUserNameValidation.Text = userInput.UserNameInvalidMessage

NewUserPasswordValidation.Text = userInput.PasswordInvalidMessage

NewUserFirstValidation.Text = userInput.FirstNameInvalidMessage

NewUserLastValidation.Text = userInput.LastNameInvalidMessage

' Check if any one is invalid, then fail.

If (userInput.IsThereAFalseInTheInput()) Then

Dim result As Boolean

' If all is okay, then set \_tempUserClass.

' Send details to the database.

\_tempUserClass.UserName = userInput.NewUserName

\_tempUserClass.Password = userInput.NewPassword

\_tempUserClass.FirstName = userInput.NewFirstName

\_tempUserClass.MiddleName = userInput.NewMiddleName

\_tempUserClass.LastName = userInput.NewLastName

result = \_tempUserClass.CreateNewUser(\_dbConnector)

If (result) Then

MainTabControl.SelectTab(0)

ClearNewUserTextBoxes()

End If

End If

End Sub

#Region "Extra Commands and functions"

''' <summary>

''' Clears the text boxes for the new user page.

''' </summary>

''' <remarks></remarks>

Private Sub ClearNewUserTextBoxes()

NewUserNameBox.Text = String.Empty

NewUserPasswordBox.Text = String.Empty

FirstNameBox.Text = String.Empty

MiddleNameBox.Text = String.Empty

LastNameBox.Text = String.Empty

End Sub

''' <summary>

''' Clears the invalid labels for the new user page.

''' </summary>

''' <remarks></remarks>

Private Sub ClearInvalidLabels()

NewUserNameValidation.Text = String.Empty

NewUserPasswordValidation.Text = String.Empty

NewUserFirstValidation.Text = String.Empty

NewUserLastValidation.Text = String.Empty

End Sub

#End Region

#End Region

#Region "Start Training"

''' <summary>

''' The start training click button.

''' Starts the new view for training.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Sub StartTrainingButton\_Click(sender As Object, e As EventArgs) Handles StartTrainingButton.Click

WindowState = FormWindowState.Minimized

SelectTrainingView.ShowDialog()

WindowState = FormWindowState.Normal

End Sub

#End Region

#Region "New Question"

''' <summary>

''' The start new question button click.

''' Starts the new question page.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Sub StartButton\_Click(sender As Object, e As EventArgs) Handles StartButton.Click

WindowState = FormWindowState.Minimized

NewQuestionView.ShowDialog()

WindowState = FormWindowState.Normal

End Sub

#End Region

#Region "Revision"

''' <summary>

''' The revision button click.

''' Starts the revision page.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Sub Button1\_Click(sender As Object, e As EventArgs) Handles RevisionStartButton.Click

WindowState = FormWindowState.Minimized

Revision.ShowDialog()

WindowState = FormWindowState.Normal

End Sub

#End Region

#Region "Misc commands"

#Region "Clear Items"

''' <summary>

''' Clears the text boxes and labels.

''' </summary>

''' <remarks></remarks>

Private Sub ClearAllItems()

ClearLogInTextBoxes()

ClearLogInInvalidLabels()

End Sub

''' <summary>

''' Clears the combo boxes.

''' </summary>

''' <param name="box"></param>

''' <remarks></remarks>

Private Sub ClearComboBoxes(box As ComboBox)

box.Items.Clear()

box.Text = String.Empty

End Sub

''' <summary>

''' Clears the log in text boxes.

''' </summary>

''' <remarks></remarks>

Private Sub ClearLogInTextBoxes()

UserNameBox.Text = String.Empty

PasswordTextBox.Text = String.Empty

End Sub

''' <summary>

''' Clears the log in invalid labels.

''' </summary>

''' <remarks></remarks>

Private Sub ClearLogInInvalidLabels()

UserNameErrorLabel.Text = String.Empty

PasswordErrorLabel.Text = String.Empty

InvalidLogOnLabel.Text = String.Empty

End Sub

#End Region

#Region "Tab Controls"

''' <summary>

''' Removes the main page tabs.

''' </summary>

''' <remarks></remarks>

Private Sub RemoveMainTabs()

MainTabControl.TabPages.Remove(SelectTrainingPage)

MainTabControl.TabPages.Remove(InsertNewQuestion)

MainTabControl.TabPages.Remove(RevisionPage)

End Sub

''' <summary>

''' Adds the main page tabs.

''' </summary>

''' <remarks></remarks>

Private Sub AddMainTabs()

MainTabControl.TabPages.Add(SelectTrainingPage)

MainTabControl.TabPages.Add(InsertNewQuestion)

MainTabControl.TabPages.Add(RevisionPage)

End Sub

#End Region

#End Region

End Class

#### NewQuestionView.vb

Imports DapperWrapper

Imports ExampleProg.Classes

Imports ExampleProg.Constants

Imports ExampleProg.DatabaseClasses

Imports ExampleProg.InputClasses

Imports ExampleProg.Interfaces

Imports ExampleProg.ProcedureReturnTypes

Imports System.Configuration

''' <summary>

''' This is the back end of the NewQuestionView page.

''' All of the functions and button clicks on the page are executed here.

'''

''' Collects, validates and then inserts the new question into the database.

''' </summary>

''' <remarks>New Question View class.</remarks>

Public Class NewQuestionView

#Region "Properties"

Private \_newQuestionInputSetup As INewQuestionInput

Dim WithEvents \_dbConnector As IDbConnector

Private \_dbConnectionString As String

Private Property AmountOfNewQuestionTabs As Integer

Private \_currentNewQuestionTab As Integer

Private \_hasGoneBackOnNewQuestionTab As Boolean

#End Region

#Region "Load"

''' <summary>

''' When the page is loaded, this method is called. Sets the TopMost to false, so that other pages can be viewed.

''' It then initialises all of the objects that are required to insert a new question.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks>Sets up all the objects on load.</remarks>

Private Sub NewQuestionView\_Load(sender As Object, e As EventArgs) Handles MyBase.Load

\_dbConnectionString = ConfigurationManager.ConnectionStrings(DatabaseConstants.ApplicationConfigString).ConnectionString

TopMost = False

Initialize()

End Sub

''' <summary>

''' Initialises all of the objects that are required when inserting a new question.

''' Calls all the other set up methods.

''' </summary>

''' <remarks>Calls the other set up methods.</remarks>

Public Sub Initialize()

InitializeDatabase()

ClearAllNewQuestionComboBoxes()

SetNewQuestionSettings()

PopulateComboBoxesWithData()

ClearNewQuestionInvalidBoxes()

DemoListGrid.Rows.Clear()

RemoveNewQuestionTabs()

AmountOfNewQuestionTabs = NewQuestionTabControl.Controls.Count() - 1

CheckIfNewQuestionTabIsAtBounds()

\_hasGoneBackOnNewQuestionTab = False

End Sub

''' <summary>

''' Initialises the Dapper Factory.

''' Sets up the connection so that any of the other methods can connect to the database.

''' </summary>

''' <remarks>Database connection set up.</remarks>

Private Sub InitializeDatabase()

Dim sqlExecutorFactory = New SqlExecutorFactory(\_dbConnectionString)

Dim newDatabaseConnection = New DatabaseConnectorWrapper(sqlExecutorFactory)

\_dbConnector = newDatabaseConnection

End Sub

''' <summary>

''' Initialises the New Question Input object.

''' Uses that object to get a list of the subject, curriculum, key stage and explanations.

''' </summary>

''' <remarks>Creates the New Question Input object.</remarks>

Private Sub SetNewQuestionSettings()

\_newQuestionInputSetup = New NewQuestionInput(\_dbConnector)

\_newQuestionInputSetup.SetListsFromDatabase()

End Sub

#End Region

#Region "New Question"

''' <summary>

''' Used when the tab at the top of the page is changed.

''' Sets the current tab, checks if it is at the bounds.

''' Checks if the user has gone backwards.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks>Tab selection.</remarks>

Private Sub NewQuestionTab\_SelectedIndexChanged(sender As Object, e As EventArgs) Handles NewQuestionTabControl.SelectedIndexChanged

\_currentNewQuestionTab = NewQuestionTabControl.SelectedIndex

CheckIfNewQuestionTabIsAtBounds()

SetHasGoneBackwards()

End Sub

''' <summary>

''' Catches the changes to the subject selection.

''' Uses the subjects collected during initialisation.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks>Selects the subject.</remarks>

Private Sub NewQuestSubjectCombo\_SelectedIndexChanged(sender As Object, e As EventArgs) Handles NewQuestSubjectCombo.SelectedIndexChanged

DoSubjectWork()

End Sub

''' <summary>

''' Catches the key presses in the combo box.

''' Looks for the tab key being presses, then checks that a value is entered.

''' Then calls the Do Subject Work methods.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks>Catches the key press.</remarks>

Private Sub NewQuestSubjectCombo\_KeyDown(sender As Object, e As KeyEventArgs) Handles NewQuestSubjectCombo.KeyDown

Dim subjectEntered As Boolean

subjectEntered = Helpers.Helpers.PerformTabOnEnter(e)

If (subjectEntered) Then

DoSubjectWork()

End If

End Sub

''' <summary>

''' Checks the subject combo box has a value and uses that to move on.

''' Uses the Subject Input class to validate the input, and then set the process.

''' If validation fails, then it will display the error message and not allow the user to continue.

''' </summary>

''' <remarks>Subject validation.</remarks>

Public Sub DoSubjectWork()

' Need to check if we have gone backwards.

If (\_hasGoneBackOnNewQuestionTab) Then

RemoveNewQuestionTabsTo(CurriculumTabPage)

ClearTextBoxesFromTabPage(ExplanationTabPage)

ClearTextBoxesFromTabPage(QuestionTabPage)

ClearTextBoxesFromTabPage(AnswerPage)

ClearTextBoxesFromTabPage(DemonstrationTabPage)

DemoListGrid.Rows.Clear()

\_newQuestionInputSetup.DemonstrationType.ResetValuesToDefaults()

End If

\_newQuestionInputSetup.SubjectType.Process(NewQuestSubjectCombo.Text)

SubjectInvalid.Text = \_newQuestionInputSetup.SubjectType.InvalidMessage

If (\_newQuestionInputSetup.SubjectType.CanProceed) Then

AddTabPage(CurriculumTabPage)

ClearComboBoxes(NewQuestCurriculumCombo)

\_newQuestionInputSetup.UpdateOtherLists()

PopulateCurriculumComboBox()

NewQuestionTabControl.SelectTab(CurriculumTabPage)

End If

End Sub

''' <summary>

''' Catches the curriculum combo box key press.

''' Looks for the tab key and then passes the flow onto the Do Curriculum Work method.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Sub NewQuestCurriculumCombo\_KeyDown(sender As Object, e As KeyEventArgs) Handles NewQuestCurriculumCombo.KeyDown

Dim curriculumEntered As Boolean

curriculumEntered = Helpers.Helpers.PerformTabOnEnter(e)

If (curriculumEntered) Then

DoCurriculumWork()

End If

End Sub

''' <summary>

''' Catches the selected index if a new value is not entered.

''' Passes the flow onto the Do Curriculum Work method.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks>Selects the curriculum index.</remarks>

Private Sub NewQuestCurriculumCombo\_SelectedIndexChanged(sender As Object, e As EventArgs) Handles NewQuestCurriculumCombo.SelectedIndexChanged

DoCurriculumWork()

End Sub

''' <summary>

''' Does the curriculum validation.

''' Sets the error message if the input is not valid.

''' Allows the user to carry on with entering the rest of the new question.

''' </summary>

''' <remarks>Validates the curriculum input.</remarks>

Public Sub DoCurriculumWork()

' Need to check if we have gone backwards.

If (\_hasGoneBackOnNewQuestionTab) Then

RemoveNewQuestionTabsTo(KeyStageTabPage)

ClearTextBoxesFromTabPage(ExplanationTabPage)

ClearTextBoxesFromTabPage(QuestionTabPage)

ClearTextBoxesFromTabPage(AnswerPage)

ClearTextBoxesFromTabPage(DemonstrationTabPage)

DemoListGrid.Rows.Clear()

\_newQuestionInputSetup.DemonstrationType.ResetValuesToDefaults()

End If

\_newQuestionInputSetup.CurriculumType.Process(NewQuestCurriculumCombo.Text)

CurriculumInvalid.Text = \_newQuestionInputSetup.CurriculumType.InvalidMessage

If (\_newQuestionInputSetup.CurriculumType.CanProceed) Then

AddTabPage(KeyStageTabPage)

ClearComboBoxes(NewQuestExplanCombo)

\_newQuestionInputSetup.UpdateExplanation()

PopulateExplanationComboBox()

NewQuestionTabControl.SelectTab(KeyStageTabPage)

End If

End Sub

''' <summary>

''' Catches the Key Stage combo box key press.

''' Looks for the tab key.

''' Passes the flow onto the Do Key Stage Work method.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks>Key Stage combo box key press.</remarks>

Private Sub NewQuestKeyCombo\_KeyDown(sender As Object, e As KeyEventArgs) Handles NewQuestKeyCombo.KeyDown

Dim keyStageEntered As Boolean

keyStageEntered = Helpers.Helpers.PerformTabOnEnter(e)

If (keyStageEntered) Then

DoKeyStageWork()

End If

End Sub

''' <summary>

''' Collects the index of the Key Stage selected.

''' Passes the input onto the Do Key Stage Work Method.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks>Key Stage index selected.</remarks>

Private Sub NewQuestKeyCombo\_SelectedIndexChanged(sender As Object, e As EventArgs) Handles NewQuestKeyCombo.SelectedIndexChanged

DoKeyStageWork()

End Sub

''' <summary>

''' Does the Key Stage validation.

''' Dispays the error message if the input is not valid.

''' Allows the user to continue if all is OK.

''' </summary>

''' <remarks>Key Stage Validation.</remarks>

Public Sub DoKeyStageWork()

' Need to check if we have gone backwards.

If (\_hasGoneBackOnNewQuestionTab) Then

RemoveNewQuestionTabsTo(ExplanationTabPage)

ClearTextBoxesFromTabPage(ExplanationTabPage)

ClearTextBoxesFromTabPage(QuestionTabPage)

ClearTextBoxesFromTabPage(AnswerPage)

ClearTextBoxesFromTabPage(DemonstrationTabPage)

DemoListGrid.Rows.Clear()

\_newQuestionInputSetup.DemonstrationType.ResetValuesToDefaults()

End If

\_newQuestionInputSetup.KeyStageType.Process(NewQuestKeyCombo.Text)

KeyStageInvalid.Text = \_newQuestionInputSetup.KeyStageType.InvalidMessage

If (\_newQuestionInputSetup.KeyStageType.CanProceed) Then

AddTabPage(ExplanationTabPage)

NewQuestionTabControl.SelectTab(ExplanationTabPage)

End If

End Sub

''' <summary>

''' Catches the Explanation combo box key press.

''' Looks for the tab key.

''' Passes the flow onto the Do Explanation Work method.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks>Explanation key press.</remarks>

Private Sub NewQuestExplanCombo\_KeyDown(sender As Object, e As KeyEventArgs) Handles NewQuestExplanCombo.KeyDown

Dim explanationTitleEntered As Boolean

explanationTitleEntered = Helpers.Helpers.PerformTabOnEnter(e)

If (explanationTitleEntered) Then

DoExplanationWork()

End If

End Sub

''' <summary>

''' Sets the index of the selected Explanation combo box.

''' Passes the flow onto the Do Explanation Work method.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks>Sets the index of the Explanation.</remarks>

Private Sub NewQuestExplanCombo\_SelectedIndexChanged(sender As Object, e As EventArgs) Handles NewQuestExplanCombo.SelectedIndexChanged, ExplanationBox.Leave

DoExplanationWork()

End Sub

''' <summary>

''' Does the Explanation validation.

''' Displays the error message if the input is not valid.

''' Allows the user to carry on if the input is OK.

''' </summary>

''' <remarks>Explanation validation.</remarks>

Public Sub DoExplanationWork()

' Need to check if we have gone backwards.

If (\_hasGoneBackOnNewQuestionTab) Then

RemoveNewQuestionTabsTo(QuestionTabPage)

ClearTextBoxesFromTabPage(QuestionTabPage)

ClearTextBoxesFromTabPage(AnswerPage)

ClearTextBoxesFromTabPage(DemonstrationTabPage)

DemoListGrid.Rows.Clear()

\_newQuestionInputSetup.DemonstrationType.ResetValuesToDefaults()

End If

\_newQuestionInputSetup.ExplanationType.Process(NewQuestExplanCombo.Text, ExplanationBox.Text)

ExplanationInvalid.Text = \_newQuestionInputSetup.ExplanationType.InvalidTitleMessage

ExplanationDetailInvalid.Text = \_newQuestionInputSetup.ExplanationType.InvalidDetailMessage

ExplanationBox.Text = \_newQuestionInputSetup.ExplanationType.ExplanationText

If (\_newQuestionInputSetup.ExplanationType.CanProceed) Then

AddTabPage(QuestionTabPage)

End If

End Sub

''' <summary>

''' Calls the do question stuff when the user leaves the question box.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Sub QuestionBox\_Leave(sender As Object, e As EventArgs) Handles QuestionBox.Leave

DoQuestionStuff()

End Sub

''' <summary>

''' Catches the web address box key down event.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Sub WebAddressBox\_KeyDown(sender As Object, e As KeyEventArgs) Handles WebAddressBox.KeyDown

Helpers.Helpers.PerformTabOnEnter(e)

End Sub

''' <summary>

''' Catches the search string box key down event.

''' Checks the input and calls the do question stuff.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Sub SearchStringBox\_KeyDown(sender As Object, e As KeyEventArgs) Handles SearchStringBox.KeyDown

Dim searchStringEntered As Boolean

searchStringEntered = Helpers.Helpers.PerformTabOnEnter(e)

If (searchStringEntered) Then

DoQuestionStuff()

End If

End Sub

''' <summary>

''' Catches the web address leave event.

''' Calls the do question stuff.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Sub WebAddressBox\_Leave(sender As Object, e As EventArgs) Handles WebAddressBox.Leave

DoQuestionStuff()

End Sub

''' <summary>

''' The main do question process.

''' Checks if the tab has gone backwards.

''' Clears the tabs if necessary.

''' Processes the new question details.

''' </summary>

''' <remarks></remarks>

Public Sub DoQuestionStuff()

' Need to check if we have gone backwards.

If (\_hasGoneBackOnNewQuestionTab) Then

RemoveNewQuestionTabsTo(AnswerPage)

ClearTextBoxesFromTabPage(AnswerPage)

ClearTextBoxesFromTabPage(DemonstrationTabPage)

DemoListGrid.Rows.Clear()

\_newQuestionInputSetup.DemonstrationType.ResetValuesToDefaults()

End If

\_newQuestionInputSetup.QuestionType.Process(QuestionBox.Text, WebAddressBox.Text, NewQuestSubjectCombo.Text,

NewQuestCurriculumCombo.Text, NewQuestKeyCombo.Text,

NewQuestExplanCombo.Text)

QuestionInvalidLabel.Text = \_newQuestionInputSetup.QuestionType.InvalidQuestionMessage

SearchStringInvalidLabel.Text = \_newQuestionInputSetup.QuestionType.InvalidSearchStringMessage

InvalidWebAddressLabel.Text = \_newQuestionInputSetup.QuestionType.InvalidWebAddressMessage

SearchStringBox.Text = \_newQuestionInputSetup.QuestionType.SearchString

' If we can proceed, then add the tab and go to it.

If (\_newQuestionInputSetup.QuestionType.CanProceed) Then

AddTabPage(AnswerPage)

End If

End Sub

''' <summary>

''' Catches the correct answer box key down event.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Shared Sub CorrectAnswerBox\_KeyDown(sender As Object, e As KeyEventArgs) Handles CorrectAnswerBox.KeyDown

Helpers.Helpers.PerformTabOnEnter(e)

End Sub

''' <summary>

''' Catches the wrong answer a key down event.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Shared Sub InCorrectBoxA\_KeyDown(sender As Object, e As KeyEventArgs) Handles InCorrectBoxA.KeyDown

Helpers.Helpers.PerformTabOnEnter(e)

End Sub

''' <summary>

''' Catches the wrong answer b key down event.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Shared Sub InCorrectBoxB\_KeyDown(sender As Object, e As KeyEventArgs) Handles InCorrectBoxB.KeyDown

Helpers.Helpers.PerformTabOnEnter(e)

End Sub

''' <summary>

''' Catches the wrong answer c leave event.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Sub InCorrectBoxC\_Leave(sender As Object, e As EventArgs) Handles InCorrectBoxC.Leave

DoAnswerStuff()

End Sub

''' <summary>

''' Catches the wrong answer c key down event.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Sub InCorrectBoxC\_KeyDown(sender As Object, e As KeyEventArgs) Handles InCorrectBoxC.KeyDown

Dim lastAnswerEntered As Boolean

lastAnswerEntered = Helpers.Helpers.PerformTabOnEnter(e)

If (lastAnswerEntered) Then

DoAnswerStuff()

End If

End Sub

''' <summary>

''' The do answer stuff process.

''' Checks if the user has gone backwards.

''' Clears the tabs up to the next point.

''' Then processes all the answer details.

''' </summary>

''' <remarks></remarks>

Public Sub DoAnswerStuff()

' Need to check if we have gone backwards.

If (\_hasGoneBackOnNewQuestionTab) Then

RemoveNewQuestionTabsTo(DemonstrationTabPage)

ClearTextBoxesFromTabPage(DemonstrationTabPage)

DemoListGrid.Rows.Clear()

\_newQuestionInputSetup.DemonstrationType.ResetValuesToDefaults()

End If

\_newQuestionInputSetup.AnswerType.Process(CorrectAnswerBox.Text, InCorrectBoxA.Text, InCorrectBoxB.Text, InCorrectBoxC.Text)

InvalidCorrectAnswerLabel.Text = \_newQuestionInputSetup.AnswerType.InvalidCorrectAnswerMessage

InvalidInCorrectLabelA.Text = \_newQuestionInputSetup.AnswerType.InvalidInCorrectAnswerAMessage

InvalidInCorrectLabelB.Text = \_newQuestionInputSetup.AnswerType.InvalidInCorrectAnswerBMessage

InvalidInCorrectLabelC.Text = \_newQuestionInputSetup.AnswerType.InvalidInCorrectAnswerCMessage

If (\_newQuestionInputSetup.AnswerType.CanProceed) Then

AddTabPage(DemonstrationTabPage)

NewQuestionTabControl.SelectTab(DemonstrationTabPage)

End If

End Sub

''' <summary>

''' Catches the demonstration details key down event.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Sub DemonstrationDetailsBox\_KeyDown(sender As Object, e As KeyEventArgs) Handles DemonstrationDetailsBox.KeyDown

Dim demoDetailsEntered As Boolean

demoDetailsEntered = Helpers.Helpers.PerformTabOnEnter(e)

If (demoDetailsEntered) Then

CheckDemoDetails()

End If

End Sub

''' <summary>

''' Catches the reg ex box key down event.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Sub RegExBox\_KeyDown(sender As Object, e As KeyEventArgs) Handles RegExBox.KeyDown

Dim regExDetailsEntered As Boolean

regExDetailsEntered = Helpers.Helpers.PerformTabOnEnter(e)

If (regExDetailsEntered) Then

CheckRegExDetails()

End If

End Sub

''' <summary>

''' Catches the demonstration step mark box key down event.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Sub DemoStepMarkBox\_KeyDown(sender As Object, e As KeyEventArgs) Handles DemoStepMarkBox.KeyDown

Dim markEntered As Boolean

markEntered = Helpers.Helpers.PerformTabOnEnter(e)

If (markEntered) Then

CheckMarkDetails()

End If

End Sub

''' <summary>

''' Catches the demonstration details leave box event.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Sub DemonstrationDetailsBox\_Leave(sender As Object, e As EventArgs) Handles DemonstrationDetailsBox.Leave

CheckDemoDetails()

End Sub

''' <summary>

''' Catches the reg ex box leave event.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Sub RegExBox\_Leave(sender As Object, e As EventArgs) Handles RegExBox.Leave

CheckRegExDetails()

End Sub

''' <summary>

''' Catches the demonstration step mark box leave event.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Sub DemoStepMarkBox\_Leave(sender As Object, e As EventArgs) Handles DemoStepMarkBox.Leave

CheckMarkDetails()

End Sub

''' <summary>

''' Checks the demonstraion details and inserts the details into the main list.

''' </summary>

''' <remarks></remarks>

Public Sub CheckDemoDetails()

\_newQuestionInputSetup.DemonstrationType.InsertNextStepDetails(DemonstrationDetailsBox.Text)

InvalidDemonstrationLabel.Text = \_newQuestionInputSetup.DemonstrationType.InvalidDemoDetailsMessage

If (\_newQuestionInputSetup.DemonstrationType.IsValidDemoDetails) Then

\_newQuestionInputSetup.DemonstrationType.SetRegEx()

RegExBox.Text = \_newQuestionInputSetup.DemonstrationType.RegExDetails

End If

End Sub

''' <summary>

''' Checks the reg ex details and inserts the details into the list.

''' </summary>

''' <remarks></remarks>

Public Sub CheckRegExDetails()

\_newQuestionInputSetup.DemonstrationType.UpdateRegEx(RegExBox.Text)

InvalidRegExLabel.Text = \_newQuestionInputSetup.DemonstrationType.InvalidRegExDetailsMessage

End Sub

''' <summary>

''' Checks the mark details into the list.

''' If all is okay, we enable the insert button.

''' </summary>

''' <remarks></remarks>

Public Sub CheckMarkDetails()

\_newQuestionInputSetup.DemonstrationType.InsertMark(DemoStepMarkBox.Text)

InvalidDemoMarkLabel.Text = \_newQuestionInputSetup.DemonstrationType.InvalidMarkMessage

' If all ok, now we enable the insert button.

If (\_newQuestionInputSetup.DemonstrationType.CanInsertDetailsToList) Then

DemonstrationInsertButton.Enabled = True

End If

End Sub

''' <summary>

''' Inserts the demo step into the list.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Sub DemonstrationInsertButton\_Click(sender As Object, e As EventArgs) Handles DemonstrationInsertButton.Click

DoDemostrationStuff()

' If the demo inserted fine, then we need to clear the text boxes

ClearTextBoxesFromTabPage(DemonstrationTabPage)

DemonstrationInsertButton.Enabled = False

End Sub

''' <summary>

''' Gathers all the details for the demo step and inserts it into the list.

''' </summary>

''' <remarks></remarks>

Public Sub DoDemostrationStuff()

' Add demo steps to the demo type input list.

Dim insertedOkay = \_newQuestionInputSetup.DemonstrationType.AddDetailsToList()

If (insertedOkay) Then

SetDemoListGrid()

' If we have one demo step inserted, we can finish.

FinishedDemoDetailsButton.Enabled = True

End If

End Sub

''' <summary>

''' Updates the grid view with the demo step inserted.

''' </summary>

''' <remarks></remarks>

Public Sub SetDemoListGrid()

' Clear out the current rows.

DemoListGrid.Rows.Clear()

' Update the demo table row.

For Each demoStep As DemonstrationStep In \_newQuestionInputSetup.DemonstrationType.DemonstrationList

DemoListGrid.Rows.Add(demoStep.StepDetails, demoStep.StepRegEx, demoStep.StepMark)

' Update the total amount of steps.

AmountOfStepsLabel.Text = \_newQuestionInputSetup.DemonstrationType.AmountOfSteps.ToString()

Next

End Sub

''' <summary>

''' Catches the demo grid cell click event.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Sub DemoListGrid\_CellClick(sender As Object, e As DataGridViewCellEventArgs) Handles DemoListGrid.CellClick

DemoListGrid.CurrentRow.Selected = True

End Sub

''' <summary>

''' Catches the demo grid key down event.

''' If it is a delete, then we need to delete that step from the list.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Sub DemoListGrid\_KeyDown(sender As Object, e As KeyEventArgs) Handles DemoListGrid.KeyDown

If (e.KeyCode = Keys.Delete) Then

Dim currentRow = DemoListGrid.CurrentRow()

Dim currentDetail = currentRow.Cells(0).Value.ToString()

' Send the current step to newDemoTypeInput to remove the current item from

' DemostrationList.

Dim hasBeenRemoved = \_newQuestionInputSetup.DemonstrationType.RemoveStepFromList(currentDetail)

If (hasBeenRemoved) Then

SetDemoListGrid()

End If

End If

End Sub

''' <summary>

''' Catches the finished demo detail button click.

''' Sets up the review page.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Sub FinishedDemoDetailsButton\_Click(sender As Object, e As EventArgs) Handles FinishedDemoDetailsButton.Click

' Enable the next tab page.

AddTabPage(ReviewPage)

NewQuestionTabControl.SelectTab(ReviewPage)

SetReviewDetails()

End Sub

''' <summary>

''' Sets up the review page with all the details entered into the previous pages.

''' Logic is missing here if the user changes anything.

''' We could set all the values as non-editable.

''' That would mean going back through each step again, just to change one value.

''' </summary>

''' <remarks></remarks>

Private Sub SetReviewDetails()

' None of this has any logic checking in yet.

' We will need to do something if a value changes here.

ReviewSubjectIDBox.Text = \_newQuestionInputSetup.SubjectType.SubjectID.ToString()

ReviewSubjectDetailsBox.Text = \_newQuestionInputSetup.SubjectType.SubjectText

InvalidReviewSubjectLabel.Text = \_newQuestionInputSetup.SubjectType.InvalidMessage

ReviewCurriculumIDBox.Text = \_newQuestionInputSetup.CurriculumType.CurriculumID.ToString()

ReviewCurriculumDetailsBox.Text = \_newQuestionInputSetup.CurriculumType.CurriculumText

InvalidReviewCurriculumLabel.Text = \_newQuestionInputSetup.CurriculumType.InvalidMessage

ReviewKeyStageIDBox.Text = \_newQuestionInputSetup.KeyStageType.KeyStageID.ToString()

ReviewKeyStageDetailsBox.Text = \_newQuestionInputSetup.KeyStageType.KeyStageText

InvalidReviewKeyStageLabel.Text = \_newQuestionInputSetup.KeyStageType.InvalidMessage

ReviewExplanationIDBox.Text = \_newQuestionInputSetup.ExplanationType.ExplanationID.ToString()

ReviewExplanationTitleBox.Text = \_newQuestionInputSetup.ExplanationType.ExplanationTitle

ReviewExplanationDetailsBox.Text = \_newQuestionInputSetup.ExplanationType.ExplanationText

InvalidReviewExplanationLabel.Text = \_newQuestionInputSetup.ExplanationType.InvalidDetailMessage

ReviewQuestionBox.Text = \_newQuestionInputSetup.QuestionType.QuestionText

ReviewQuestionWebAddressBox.Text = \_newQuestionInputSetup.QuestionType.WebAddressText

ReviewQuestionSearchBox.Text = \_newQuestionInputSetup.QuestionType.SearchString

InvalidReviewQuestionLabel.Text = \_newQuestionInputSetup.QuestionType.InvalidQuestionMessage

ReviewCorrectAnswerBox.Text = \_newQuestionInputSetup.AnswerType.CorrectAnswer

ReviewWrongAnswerABox.Text = \_newQuestionInputSetup.AnswerType.InCorrectAnswerA

ReviewWrongAnswerBBox.Text = \_newQuestionInputSetup.AnswerType.InCorrectAnswerB

ReviewWrongAnswerCBox.Text = \_newQuestionInputSetup.AnswerType.InCorrectAnswerC

InvalidReviewAnswerLabel.Text = \_newQuestionInputSetup.AnswerType.InvalidCorrectAnswerMessage

' Clear out the current rows.

ReviewDemoGridBox.Rows.Clear()

' Update the demo table row.

For Each demoStep As DemonstrationStep In \_newQuestionInputSetup.DemonstrationType.DemonstrationList

ReviewDemoGridBox.Rows.Add(demoStep.StepDetails, demoStep.StepRegEx, demoStep.StepMark)

Next

ReviewTotalMarksBox.Text = \_newQuestionInputSetup.DemonstrationType.DemonstrationTotalMarks.ToString()

\_newQuestionInputSetup.Process()

If (\_newQuestionInputSetup.CanSubmit) Then

SubmitButton.Enabled = True

End If

End Sub

''' <summary>

''' Here we insert the new question into the database.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks>Inserting the question.</remarks>

Private Sub SubmitButton\_Click(sender As Object, e As EventArgs) Handles SubmitButton.Click

\_newQuestionInputSetup.InsertNewQuestion()

If (\_newQuestionInputSetup.InsertedSuccessfully) Then

MsgBox(\_newQuestionInputSetup.NewQuestionInsertMessage, Buttons:=MsgBoxStyle.OkOnly)

Close()

Else

MsgBox(\_newQuestionInputSetup.NewQuestionInsertMessage, Buttons:=MsgBoxStyle.OkOnly)

End If

End Sub

#End Region

#Region "Misc Functions"

''' <summary>

''' Catches the cancel button click.

''' This closes the window.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Sub NewQuestCancelButton\_Click(sender As Object, e As EventArgs) Handles NewQuestCancelButton.Click

Close()

End Sub

''' <summary>

''' Clears the invalid labels.

''' </summary>

''' <remarks></remarks>

Public Sub ClearNewQuestionInvalidBoxes()

SubjectInvalid.Text = String.Empty

CurriculumInvalid.Text = String.Empty

KeyStageInvalid.Text = String.Empty

ExplanationInvalid.Text = String.Empty

QuestionInvalid.Text = String.Empty

End Sub

''' <summary>

''' Clears the combo box selected.

''' </summary>

''' <param name="box"></param>

''' <remarks></remarks>

Public Shared Sub ClearComboBoxes(box As ComboBox)

box.Items.Clear()

box.Text = String.Empty

End Sub

''' <summary>

''' Clears all of the combo boxes.

''' </summary>

''' <remarks></remarks>

Public Sub ClearAllNewQuestionComboBoxes()

ClearComboBoxes(NewQuestSubjectCombo)

ClearComboBoxes(NewQuestCurriculumCombo)

ClearComboBoxes(NewQuestKeyCombo)

ClearComboBoxes(NewQuestExplanCombo)

End Sub

''' <summary>

''' Clears the text boxes text from the tab page.

''' </summary>

''' <param name="page"></param>

''' <remarks></remarks>

Private Sub ClearTextBoxesFromTabPage(page As TabPage)

For Each item As Control In page.Controls

If (TypeOf item Is TextBox) Then

item.Text = String.Empty

End If

Next

End Sub

#End Region

#Region "Populate Combo Boxes"

''' <summary>

''' Populates the boxes with data from the main class.

''' </summary>

''' <remarks></remarks>

Public Sub PopulateComboBoxesWithData()

PopulateSubjectComboBox()

PopulateCurriculumComboBox()

PopulateKeyStageComboBox()

PopulateExplanationComboBox()

End Sub

''' <summary>

''' Populates the subject list.

''' </summary>

''' <remarks></remarks>

Private Sub PopulateSubjectComboBox()

For Each subject As SubjectListType In \_newQuestionInputSetup.SubjectType.SubjectList

NewQuestSubjectCombo.Items.Add(subject.SubjectDetail)

Next

End Sub

''' <summary>

''' Populates the curriculum list.

''' </summary>

''' <remarks></remarks>

Private Sub PopulateCurriculumComboBox()

For Each curric As CurriculumListType In \_newQuestionInputSetup.CurriculumType.CurriculumList

NewQuestCurriculumCombo.Items.Add(curric.CurriculumDetails)

Next

End Sub

''' <summary>

''' Populates the key stage list.

''' </summary>

''' <remarks></remarks>

Private Sub PopulateKeyStageComboBox()

For Each key As KeyStageListType In \_newQuestionInputSetup.KeyStageType.KeyStageList

NewQuestKeyCombo.Items.Add(key.KeyStageDetail)

Next

End Sub

''' <summary>

''' Populates the explanation list.

''' </summary>

''' <remarks></remarks>

Private Sub PopulateExplanationComboBox()

For Each explan As ExplanationListType In \_newQuestionInputSetup.ExplanationType.ExplanationList

NewQuestExplanCombo.Items.Add(explan.ExplanationDetail)

Next

End Sub

#End Region

#Region "New Question Navigation"

''' <summary>

''' Catches the next button click.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Sub NextButton\_Click(sender As Object, e As EventArgs) Handles NextButton.Click

NewQuestionTabControl.SelectTab(\_currentNewQuestionTab + 1)

End Sub

''' <summary>

''' Catches the previous button click.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Sub PreviousButton\_Click(sender As Object, e As EventArgs) Handles PreviousButton.Click

NewQuestionTabControl.SelectTab(\_currentNewQuestionTab - 1)

End Sub

''' <summary>

''' Checks to see if the current tab is at the bounds to disable the next or previous buttons.

''' </summary>

''' <remarks></remarks>

Public Sub CheckIfNewQuestionTabIsAtBounds()

If (\_currentNewQuestionTab = 0) Then

PreviousButton.Enabled = False

Else

PreviousButton.Enabled = True

End If

If (\_currentNewQuestionTab = (AmountOfNewQuestionTabs)) Then

NextButton.Enabled = False

Else

NextButton.Enabled = True

End If

End Sub

''' <summary>

''' Checks if the user has selected a tab page 2 steps back from where they started.

''' </summary>

''' <remarks></remarks>

Public Sub SetHasGoneBackwards()

Dim gap = AmountOfNewQuestionTabs - \_currentNewQuestionTab

If (gap >= 2) Then

\_hasGoneBackOnNewQuestionTab = True

Else

\_hasGoneBackOnNewQuestionTab = False

End If

End Sub

#End Region

#Region "Tab Controls"

''' <summary>

''' Removes the tab pages from the tab control, so they get added in order.

''' The user can only go one way, and not jump around the pages.

''' </summary>

''' <remarks></remarks>

Public Sub RemoveNewQuestionTabs()

NewQuestionTabControl.TabPages.Remove(CurriculumTabPage)

NewQuestionTabControl.TabPages.Remove(KeyStageTabPage)

NewQuestionTabControl.TabPages.Remove(ExplanationTabPage)

NewQuestionTabControl.TabPages.Remove(QuestionTabPage)

NewQuestionTabControl.TabPages.Remove(AnswerPage)

NewQuestionTabControl.TabPages.Remove(DemonstrationTabPage)

NewQuestionTabControl.TabPages.Remove(ReviewPage)

End Sub

''' <summary>

''' Removes the tab pages from the control to the current tab.

''' </summary>

''' <param name="currentTab"></param>

''' <remarks></remarks>

Public Sub RemoveNewQuestionTabsTo(currentTab As TabPage)

Dim indexOfCurrentTab = NewQuestionTabControl.TabPages(currentTab.Name).TabIndex

For index As Integer = AmountOfNewQuestionTabs To indexOfCurrentTab Step -1

NewQuestionTabControl.TabPages.RemoveAt(index)

Next

AmountOfNewQuestionTabs = NewQuestionTabControl.Controls.Count() - 1

End Sub

''' <summary>

''' Adds the tab page to the control.

''' </summary>

''' <param name="nextPage"></param>

''' <remarks></remarks>

Public Sub AddTabPage(nextPage As TabPage)

Dim exists As Boolean

exists = False

For Each item As TabPage In NewQuestionTabControl.TabPages

If (item.Name = nextPage.Name) Then

exists = True

End If

Next

If Not (exists) Then

NewQuestionTabControl.TabPages.Add(nextPage)

AmountOfNewQuestionTabs = NewQuestionTabControl.Controls.Count() - 1

CheckIfNewQuestionTabIsAtBounds()

End If

End Sub

#End Region

End Class

#### Revision.vb

Imports System.Configuration

Imports DapperWrapper

Imports ExampleProg.Constants

Imports ExampleProg.DatabaseClasses

Imports ExampleProg.Interfaces

''' <summary>

''' Unfinished revision page.

''' </summary>

''' <remarks></remarks>

Public Class Revision

Dim WithEvents \_dbConnector As IDbConnector

Private \_dbConnectionString As String

''' <summary>

''' Catches the load event.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Sub Revision\_Load(sender As Object, e As EventArgs) Handles MyBase.Load

\_dbConnectionString = ConfigurationManager.ConnectionStrings(DatabaseConstants.ApplicationConfigString).ConnectionString

TopMost = False

Initialize()

End Sub

''' <summary>

''' Initialises the setting for the page.

''' </summary>

''' <remarks></remarks>

Public Sub Initialize()

InitializeDatabase()

End Sub

''' <summary>

''' Sets up the database connection for the window.

''' </summary>

''' <remarks></remarks>

Private Sub InitializeDatabase()

Dim sqlExecutorFactory = New SqlExecutorFactory(\_dbConnectionString)

Dim newDatabaseConnection = New DatabaseConnectorWrapper(sqlExecutorFactory)

\_dbConnector = newDatabaseConnection

End Sub

''' <summary>

''' Catches the cancel click event.

''' Closes the window.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Sub CancelButton\_Click(sender As Object, e As EventArgs) Handles RevisionCancelButton.Click

Close()

End Sub

End Class

#### SelectTrainingView.vb

Imports System.Configuration

Imports DapperWrapper

Imports ExampleProg.Constants

Imports ExampleProg.DatabaseClasses

Imports ExampleProg.InputClasses.Training

Imports ExampleProg.Interfaces

Imports ExampleProg.ProcedureReturnTypes

''' <summary>

''' In-complete page for selecting the training required.

''' Almost done, but needs work on the pages and completing the practice section.

''' </summary>

''' <remarks></remarks>

Public Class SelectTrainingView

Dim WithEvents \_dbConnector As IDbConnector

Private Property SelectTrainingInput As ISelectTrainingInput

Private \_dbConnectionString As String

Private \_demoTraining As DemonstrationTraining

Private \_immitationTraining As ImmitationTraining

Private \_questionID As Integer

''' <summary>

''' Catches the on load event.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Sub SelectTrainingView\_Load(sender As Object, e As EventArgs) Handles MyBase.Load

\_dbConnectionString = ConfigurationManager.ConnectionStrings(DatabaseConstants.ApplicationConfigString).ConnectionString

TopMost = False

Initialize()

End Sub

''' <summary>

''' Sets up the systems for the window.

''' </summary>

''' <remarks></remarks>

Private Sub Initialize()

ClearSelectTrainingListBoxes()

RemoveNewQuestionTabs()

HideUnusedButtons()

InitializeDatabase()

SelectTrainingInput = New SelectTrainingInput(\_dbConnector)

\_demoTraining = New DemonstrationTraining(\_dbConnector)

\_immitationTraining = New ImmitationTraining(\_dbConnector)

\_questionID = 0

PopulateSubjectComboBox()

End Sub

''' <summary>

''' Sets up the database connection.

''' </summary>

''' <remarks></remarks>

Private Sub InitializeDatabase()

Dim sqlExecutorFactory = New SqlExecutorFactory(\_dbConnectionString)

Dim newDatabaseConnection = New DatabaseConnectorWrapper(sqlExecutorFactory)

\_dbConnector = newDatabaseConnection

End Sub

''' <summary>

''' Catches the cancel button click event.

''' Closes the window.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Sub CancelButton\_Click(sender As Object, e As EventArgs) Handles SelectTrainingCancelButton.Click

Close()

End Sub

#Region "Selection"

''' <summary>

''' Catches the subject change and filteres the value through to the rest of the page.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Sub SubjectListBox\_SelectedIndexChanged(sender As Object, e As EventArgs) Handles SubjectListBox.SelectedIndexChanged

SelectTrainingInput.SelectedSubjectText = SubjectListBox.Text

SelectTrainingInput.SetSubjectID()

SelectTrainingInput.PopulateUpdatedCurriculumList()

PopulateCurriculumComboBox()

PopulateKeyStageComboBox()

SelectTrainingInput.PopulateUpdatedExplanationList()

PopulateExplanationComboBox()

CurriculumListBox.Enabled = True

KeyStageListBox.Enabled = False

ExplanationListBox.Enabled = False

ExplanationButton.Enabled = False

End Sub

''' <summary>

''' Catches the curriculum change event and filters on down.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Sub CurriculumListBox\_SelectedIndexChanged(sender As Object, e As EventArgs) Handles CurriculumListBox.SelectedIndexChanged

SelectTrainingInput.SelectedCurriculumText = CurriculumListBox.Text

SelectTrainingInput.SetCurriculumID()

KeyStageListBox.Enabled = True

ExplanationListBox.Enabled = False

ExplanationButton.Enabled = False

End Sub

''' <summary>

''' Catches the key stage changed event.

''' Filters on down.

''' We also need to ignore the key stage in the stored procedure.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Sub KeyStageListBox\_SelectedIndexChanged(sender As Object, e As EventArgs) Handles KeyStageListBox.SelectedIndexChanged

SelectTrainingInput.SelectedKeyStageText = KeyStageListBox.Text

SelectTrainingInput.SetKeyStageID()

SelectTrainingInput.PopulateUpdatedExplanationList()

PopulateExplanationComboBox()

ExplanationListBox.Enabled = True

ExplanationButton.Enabled = False

End Sub

''' <summary>

''' Catches the explanation changed event.

''' Filters on down.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Sub ExplanationListBox\_SelectedIndexChanged(sender As Object, e As EventArgs) Handles ExplanationListBox.SelectedIndexChanged

SelectTrainingInput.SelectedExplanationText = ExplanationListBox.Text

SelectTrainingInput.SetExplanationID()

ExplanationButton.Enabled = True

AddTabPage(ExplanationTabPage)

End Sub

''' <summary>

''' Catches the explanation button click.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Sub ExplanationButton\_Click(sender As Object, e As EventArgs) Handles ExplanationButton.Click

TrainingTabControl.SelectTab(ExplanationTabPage)

End Sub

#End Region

#Region "Explanation"

''' <summary>

''' Sets the explanation details tab page.

''' </summary>

''' <remarks></remarks>

Private Sub SetExplanationDetails()

Dim explanationTrainingType = New ExplanationTraining(SelectTrainingInput.SelectedExplanationID, \_dbConnector)

ExplanationTitleLabel.Text = explanationTrainingType.ExplanationTitle

ExplanationTrainingTextBox.Text = explanationTrainingType.ExplanationDetails

End Sub

''' <summary>

''' Catches the enter event to set up the rest of the process.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Sub ExplanationTabPage\_Enter(sender As Object, e As EventArgs) Handles ExplanationTabPage.Enter

SetExplanationDetails()

AddTabPage(DemonstrationTabPage)

ExplanationButton.Enabled = False

ExplanationButton.Visible = False

DemoButton.Visible = True

DemoButton.Enabled = True

End Sub

''' <summary>

''' Catches the demo button click event.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Sub DemoButton\_Click(sender As Object, e As EventArgs) Handles DemoButton.Click

TrainingTabControl.SelectTab(DemonstrationTabPage)

End Sub

#End Region

#Region "Demonstration"

''' <summary>

''' Catches the demo tab page enter event.

''' Checks the question id and if it is not set, randomises a value from the database.

''' Then sets the rest up after that.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Sub DemonstrationTabPage\_Enter(sender As Object, e As EventArgs) Handles DemonstrationTabPage.Enter

If (\_questionID < 1) Then

\_demoTraining.Initialise(SelectTrainingInput.SelectedSubjectID, \_

SelectTrainingInput.SelectedCurriculumID, SelectTrainingInput.SelectedKeyStageID, \_

SelectTrainingInput.SelectedExplanationID)

\_demoTraining.SetRandomQuestionID()

\_demoTraining.SetUpDemoSteps()

End If

If (\_demoTraining.HasDemoStepsBeenSetUp) Then

\_questionID = \_demoTraining.QuestionID

DemoStepDetailsTextBox.Text = \_demoTraining.DemonstrationStepDetails

DemoAmountSteps.Text = \_demoTraining.CountOfDemonstrationSteps.ToString()

CheckIfDemoStepIsAtBounds()

End If

End Sub

''' <summary>

''' Catches the demo next button event.

''' Moves the demo step on one.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Sub DemoNextButton\_Click(sender As Object, e As EventArgs) Handles DemoNextButton.Click

\_demoTraining.NextStep()

DemoStepDetailsTextBox.Text = \_demoTraining.DemonstrationStepDetails

CheckIfDemoStepIsAtBounds()

If (\_demoTraining.CanProceed()) Then

AddTabPage(ImmitationTabPage)

ImmitationButton.Visible = True

ImmitationButton.Enabled = True

DemoButton.Visible = False

DemoButton.Enabled = False

End If

End Sub

''' <summary>

''' Catches the demo previous step.

''' Moves the demo step back one for the user to go over the steps.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Sub DemoPreviousButton\_Click(sender As Object, e As EventArgs) Handles DemoPreviousButton.Click

\_demoTraining.PreviousStep()

DemoStepDetailsTextBox.Text = \_demoTraining.DemonstrationStepDetails

CheckIfDemoStepIsAtBounds()

If Not (\_demoTraining.CanProceed()) Then

ImmitationButton.Visible = False

ImmitationButton.Enabled = False

DemoButton.Visible = True

DemoButton.Enabled = True

End If

End Sub

''' <summary>

''' Catches the immitation button click.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Sub ImmitationButton\_Click(sender As Object, e As EventArgs) Handles ImmitationButton.Click

TrainingTabControl.SelectTab(ImmitationTabPage)

End Sub

#End Region

#Region "Immitation"

''' <summary>

''' Sets up the immitation section of the traininig.

''' Based on the question id set in the demo stage.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Sub TrainingTabControl\_SelectedIndexChanged(sender As Object, e As EventArgs) Handles TrainingTabControl.SelectedIndexChanged

If (TrainingTabControl.SelectedTab.TabIndex = ImmitationTabPage.TabIndex) Then

\_immitationTraining.GetDemonstrationList(\_questionID)

ImmitationDemoTextBox.Text = \_immitationTraining.DemonstrationStepDetails

End If

End Sub

''' <summary>

''' Cathces the immitation text box change event.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Sub ImmitationTextBox\_TextChanged(sender As Object, e As EventArgs) Handles ImmitationTextBox.TextChanged

If (ImmitationTextBox.Text.Length > 0) Then

ImmitationCheckButton.Enabled = True

Else

ImmitationCheckButton.Enabled = False

End If

End Sub

''' <summary>

''' Catches the immitation check button click event.

''' The user want to check their answer.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Sub ImmitationCheckButton\_Click(sender As Object, e As EventArgs) Handles ImmitationCheckButton.Click

\_immitationTraining.ProcessInput(ImmitationTextBox.Text)

If (\_immitationTraining.IsDemoInputCorrect) Then

If (\_immitationTraining.HasImmitationBeenCompleted) Then

AddTabPage(PracticeTabPage)

PracticeButton.Visible = True

PracticeButton.Enabled = True

ImmitationButton.Visible = False

ImmitationButton.Enabled = False

Else

ImmitationDemoTextBox.Text = \_immitationTraining.DemonstrationStepDetails

ImmitationTextBox.Text = String.Empty

End If

End If

End Sub

''' <summary>

''' Catches the practice button on click event.

''' Will display the practice page.

''' </summary>

''' <param name="sender"></param>

''' <param name="e"></param>

''' <remarks></remarks>

Private Sub PracticeButton\_Click(sender As Object, e As EventArgs) Handles PracticeButton.Click

End Sub

#End Region

#Region "Populate Combo Boxes"

''' <summary>

''' Populates the subject list.

''' </summary>

''' <remarks></remarks>

Private Sub PopulateSubjectComboBox()

SubjectListBox.Items.Clear()

For Each subject As SubjectListType In SelectTrainingInput.SubjectList

SubjectListBox.Items.Add(subject.SubjectDetail)

Next

End Sub

''' <summary>

''' Populates the curriculum list.

''' </summary>

''' <remarks></remarks>

Private Sub PopulateCurriculumComboBox()

CurriculumListBox.Items.Clear()

For Each curric As CurriculumListType In SelectTrainingInput.CurriculumList

CurriculumListBox.Items.Add(curric.CurriculumDetails)

Next

End Sub

''' <summary>

''' Populates the key stage list.

''' </summary>

''' <remarks></remarks>

Private Sub PopulateKeyStageComboBox()

KeyStageListBox.Items.Clear()

For Each key As KeyStageListType In SelectTrainingInput.KeyStageList

KeyStageListBox.Items.Add(key.KeyStageDetail)

Next

End Sub

''' <summary>

''' Populates the explanation list.

''' </summary>

''' <remarks></remarks>

Private Sub PopulateExplanationComboBox()

ExplanationListBox.Items.Clear()

For Each explan As ExplanationListType In SelectTrainingInput.ExplanationList

ExplanationListBox.Items.Add(explan.ExplanationDetail)

Next

End Sub

#End Region

#Region "Misc Functions"

''' <summary>

''' Clears the selected combo box.

''' </summary>

''' <param name="box"></param>

''' <remarks></remarks>

Private Shared Sub ClearComboBoxes(box As ListBox)

box.Items.Clear()

box.Text = String.Empty

End Sub

''' <summary>

''' Clears all the training combo boxes.

''' </summary>

''' <remarks></remarks>

Private Sub ClearSelectTrainingListBoxes()

ClearComboBoxes(SubjectListBox)

ClearComboBoxes(CurriculumListBox)

ClearComboBoxes(KeyStageListBox)

ClearComboBoxes(ExplanationListBox)

End Sub

''' <summary>

''' Removes the tab pages back to the current page.

''' </summary>

''' <remarks></remarks>

Private Sub RemoveNewQuestionTabs()

TrainingTabControl.TabPages.Remove(ExplanationTabPage)

TrainingTabControl.TabPages.Remove(DemonstrationTabPage)

TrainingTabControl.TabPages.Remove(ImmitationTabPage)

TrainingTabControl.TabPages.Remove(PracticeTabPage)

End Sub

''' <summary>

''' Hides the buttons, as they are stacked on each other in the design view.

''' </summary>

''' <remarks></remarks>

Private Sub HideUnusedButtons()

DemoButton.Visible = False

ImmitationButton.Visible = False

PracticeButton.Visible = False

End Sub

''' <summary>

''' Adds the next tab page to the control.

''' </summary>

''' <param name="nextPage"></param>

''' <remarks></remarks>

Public Sub AddTabPage(nextPage As TabPage)

Dim exists As Boolean

exists = False

For Each item As TabPage In TrainingTabControl.TabPages

If (item.Name = nextPage.Name) Then

exists = True

End If

Next

If Not (exists) Then

TrainingTabControl.TabPages.Add(nextPage)

End If

End Sub

''' <summary>

''' Checks if the demo step is at the bounds.

''' Disables the next or previous buttons.

''' </summary>

''' <remarks></remarks>

Public Sub CheckIfDemoStepIsAtBounds()

If (\_demoTraining.HasDemoStepsBeenSetUp) Then

If (\_demoTraining.DemonstrationStep < (\_demoTraining.CountOfDemonstrationSteps - 1)) Then

DemoNextButton.Enabled = True

Else

DemoNextButton.Enabled = False

End If

If (\_demoTraining.DemonstrationStep > 0) Then

DemoPreviousButton.Enabled = True

Else

DemoPreviousButton.Enabled = False

End If

End If

End Sub

#End Region

End Class

## Project Tests

### App.config

<?xml version="1.0" encoding="utf-8"?>

<configuration>

<runtime>

<assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">

<dependentAssembly>

<assemblyIdentity name="Moq" publicKeyToken="69f491c39445e920" culture="neutral" />

<bindingRedirect oldVersion="0.0.0.0-4.2.1409.1722" newVersion="4.2.1409.1722" />

</dependentAssembly>

</assemblyBinding>

</runtime>

<startup>

<supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5.1" />

</startup>

</configuration>

### Packages.config

<?xml version="1.0" encoding="utf-8"?>

<packages>

<package id="Dapper" version="1.38" targetFramework="net451" />

<package id="DapperWrapper" version="0.3.0.0" targetFramework="net451" />

<package id="Moq" version="4.2.1409.1722" targetFramework="net451" />

<package id="Regextra" version="0.3.0.0" targetFramework="net451" />

</packages>

### New Quest Type Tests

#### AnswerTypeInputTests.vb

Imports System.Text

Imports Microsoft.VisualStudio.TestTools.UnitTesting

Imports ExampleProg.InputClasses.NewQuestionTypes

Imports ExampleProg.Constants

<TestClass()> Public Class AnswerTypeInputTests

<TestMethod()> Public Sub test\_process\_correct\_answer\_sets\_valid\_flag\_to\_true()

' Arrange.

Dim answerType = New AnswerTypeInput()

' Act.

answerType.Process("yep", "nope", "um", "no way")

' Assert.

Assert.IsTrue(answerType.IsValidCorrectAnswer)

End Sub

<TestMethod()> Public Sub test\_process\_incorrect\_answer\_a\_sets\_valid\_flag\_to\_true()

' Arrange.

Dim answerType = New AnswerTypeInput()

' Act.

answerType.Process("yep", "nope", "um", "no way")

' Assert.

Assert.IsTrue(answerType.IsValidInCorrectAnswerA)

End Sub

<TestMethod()> Public Sub test\_process\_incorrect\_answer\_b\_sets\_valid\_flag\_to\_true()

' Arrange.

Dim answerType = New AnswerTypeInput()

' Act.

answerType.Process("yep", "nope", "um", "no way")

' Assert.

Assert.IsTrue(answerType.IsValidInCorrectAnswerB)

End Sub

<TestMethod()> Public Sub test\_process\_incorrect\_answer\_c\_sets\_valid\_flag\_to\_true()

' Arrange.

Dim answerType = New AnswerTypeInput()

' Act.

answerType.Process("yep", "nope", "um", "no way")

' Assert.

Assert.IsTrue(answerType.IsValidInCorrectAnswerC)

End Sub

<TestMethod()> Public Sub test\_process\_correct\_answer\_sets\_valid\_flag\_to\_false()

' Arrange.

Dim answerType = New AnswerTypeInput()

' Act.

answerType.Process(String.Empty, "nope", "um", "no way")

' Assert.

Assert.IsFalse(answerType.IsValidCorrectAnswer)

End Sub

<TestMethod()> Public Sub test\_process\_incorrect\_answer\_a\_sets\_valid\_flag\_to\_false()

' Arrange.

Dim answerType = New AnswerTypeInput()

' Act.

answerType.Process("yep", String.Empty, "um", "no way")

' Assert.

Assert.IsFalse(answerType.IsValidInCorrectAnswerA)

End Sub

<TestMethod()> Public Sub test\_process\_incorrect\_answer\_b\_sets\_valid\_flag\_to\_false()

' Arrange.

Dim answerType = New AnswerTypeInput()

' Act.

answerType.Process("yep", "nope", String.Empty, "no way")

' Assert.

Assert.IsFalse(answerType.IsValidInCorrectAnswerB)

End Sub

<TestMethod()> Public Sub test\_process\_incorrect\_answer\_c\_sets\_valid\_flag\_to\_false()

' Arrange.

Dim answerType = New AnswerTypeInput()

' Act.

answerType.Process("yep", "nope", "um", String.Empty)

' Assert.

Assert.IsFalse(answerType.IsValidInCorrectAnswerC)

End Sub

<TestMethod()> Public Sub test\_process\_correct\_answer\_sets\_invalid\_message\_to\_empty\_string()

' Arrange.

Dim answerType = New AnswerTypeInput()

' Act.

answerType.Process("yep", "nope", "um", "no way")

' Assert.

Assert.AreEqual(String.Empty, answerType.InvalidCorrectAnswerMessage)

End Sub

<TestMethod()> Public Sub test\_process\_incorrect\_answer\_a\_sets\_invalid\_message\_to\_empty\_string()

' Arrange.

Dim answerType = New AnswerTypeInput()

' Act.

answerType.Process("yep", "nope", "um", "no way")

' Assert.

Assert.AreEqual(String.Empty, answerType.InvalidInCorrectAnswerAMessage)

End Sub

<TestMethod()> Public Sub test\_process\_incorrect\_answer\_b\_sets\_invalid\_message\_to\_empty\_string()

' Arrange.

Dim answerType = New AnswerTypeInput()

' Act.

answerType.Process("yep", "nope", "um", "no way")

' Assert.

Assert.AreEqual(String.Empty, answerType.InvalidInCorrectAnswerBMessage)

End Sub

<TestMethod()> Public Sub test\_process\_incorrect\_answer\_c\_sets\_invalid\_message\_to\_empty\_string()

' Arrange.

Dim answerType = New AnswerTypeInput()

' Act.

answerType.Process("yep", "nope", "um", "no way")

' Assert.

Assert.AreEqual(String.Empty, answerType.InvalidInCorrectAnswerCMessage)

End Sub

<TestMethod()> Public Sub test\_process\_correct\_answer\_sets\_invalid\_message\_to\_not\_a\_valid\_input()

' Arrange.

Dim answerType = New AnswerTypeInput()

' Act.

answerType.Process(String.Empty, "nope", "um", "no way")

' Assert.

Assert.AreEqual(CommonConstants.NotAValidInput, answerType.InvalidCorrectAnswerMessage)

End Sub

<TestMethod()> Public Sub test\_process\_incorrect\_answer\_a\_sets\_invalid\_message\_to\_not\_a\_valid\_input()

' Arrange.

Dim answerType = New AnswerTypeInput()

' Act.

answerType.Process("yep", String.Empty, "um", "no way")

' Assert.

Assert.AreEqual(CommonConstants.NotAValidInput, answerType.InvalidInCorrectAnswerAMessage)

End Sub

<TestMethod()> Public Sub test\_process\_incorrect\_answer\_b\_sets\_invalid\_message\_to\_not\_a\_valid\_input()

' Arrange.

Dim answerType = New AnswerTypeInput()

' Act.

answerType.Process("yep", "nope", String.Empty, "no way")

' Assert.

Assert.AreEqual(CommonConstants.NotAValidInput, answerType.InvalidInCorrectAnswerBMessage)

End Sub

<TestMethod()> Public Sub test\_process\_incorrect\_answer\_c\_sets\_invalid\_message\_to\_not\_a\_valid\_input()

' Arrange.

Dim answerType = New AnswerTypeInput()

' Act.

answerType.Process("yep", "nope", "um", String.Empty)

' Assert.

Assert.AreEqual(CommonConstants.NotAValidInput, answerType.InvalidInCorrectAnswerCMessage)

End Sub

<TestMethod()> Public Sub test\_process\_sets\_can\_proceed\_to\_true()

' Arrange.

Dim answerType = New AnswerTypeInput()

' Act.

answerType.Process("yep", "nope", "um", "no way")

' Assert.

Assert.IsTrue(answerType.CanProceed)

End Sub

<TestMethod()> Public Sub test\_process\_sets\_can\_proceed\_to\_false\_with\_no\_correct\_answer()

' Arrange.

Dim answerType = New AnswerTypeInput()

' Act.

answerType.Process(String.Empty, "nope", "um", "no way")

' Assert.

Assert.IsFalse(answerType.CanProceed)

End Sub

<TestMethod()> Public Sub test\_process\_sets\_can\_proceed\_to\_false\_with\_no\_incorrect\_answer\_a()

' Arrange.

Dim answerType = New AnswerTypeInput()

' Act.

answerType.Process("yep", String.Empty, "um", "no way")

' Assert.

Assert.IsFalse(answerType.CanProceed)

End Sub

<TestMethod()> Public Sub test\_process\_sets\_can\_proceed\_to\_false\_with\_no\_incorrect\_answer\_b()

' Arrange.

Dim answerType = New AnswerTypeInput()

' Act.

answerType.Process("yep", "nope", String.Empty, "no way")

' Assert.

Assert.IsFalse(answerType.CanProceed)

End Sub

<TestMethod()> Public Sub test\_process\_sets\_can\_proceed\_to\_false\_with\_no\_incorrect\_answer\_c()

' Arrange.

Dim answerType = New AnswerTypeInput()

' Act.

answerType.Process("yep", "nope", "um", String.Empty)

' Assert.

Assert.IsFalse(answerType.CanProceed)

End Sub

End Class

#### CurriculumTypeInputTests.vb

Imports ExampleProg.Constants

Imports ExampleProg.InputClasses.NewQuestionTypes

Imports ExampleProg.ProcedureReturnTypes

Imports Microsoft.VisualStudio.TestTools.UnitTesting

Namespace NewQuestTypeTests

<TestClass()> Public Class CurriculumTypeInputTests

<TestMethod()> Public Sub process\_sets\_can\_proceed\_to\_true()

' Arrange.

Dim curriculumTypeInput = New CurriculumTypeInput()

Dim curriculum1 = New CurriculumListType()

curriculum1.ID = 1

curriculum1.CurriculumDetails = "Basic"

Dim curriculum2 = New CurriculumListType()

curriculum2.ID = 2

curriculum2.CurriculumDetails = "Advanced"

' Faking out the list.

Dim list = New List(Of CurriculumListType)

list.Add(curriculum1)

list.Add(curriculum2)

curriculumTypeInput.CurriculumList = list

' Act.

curriculumTypeInput.Process("Basic")

' Assert.

Assert.IsTrue(curriculumTypeInput.CanProceed)

End Sub

<TestMethod()> Public Sub process\_sets\_can\_proceed\_to\_false()

' Arrange.

Dim curriculumTypeInput = New CurriculumTypeInput()

' Faking out the list.

curriculumTypeInput.CurriculumList = New List(Of CurriculumListType)

' Act.

curriculumTypeInput.Process(String.Empty)

' Assert.

Assert.IsFalse(curriculumTypeInput.CanProceed)

End Sub

<TestMethod()> Public Sub process\_sets\_invalid\_message\_flag\_to\_empty\_string()

' Arrange.

Dim curriculumTypeInput = New CurriculumTypeInput()

' Faking out the list.

curriculumTypeInput.CurriculumList = New List(Of CurriculumListType)

' Act.

curriculumTypeInput.Process("hello")

' Assert.

Assert.AreEqual(String.Empty, curriculumTypeInput.InvalidMessage)

End Sub

<TestMethod()> Public Sub process\_sets\_invalid\_message\_flag\_to\_not\_a\_valid\_input()

' Arrange.

Dim curriculumTypeInput = New CurriculumTypeInput()

' Faking out the list.

curriculumTypeInput.CurriculumList = New List(Of CurriculumListType)

' Act.

curriculumTypeInput.Process(String.Empty)

' Assert.

Assert.AreEqual(CommonConstants.NotAValidInput, curriculumTypeInput.InvalidMessage)

End Sub

<TestMethod()> Public Sub process\_sets\_curriculum\_id\_correctly()

' Arrange.

Dim curriculumTypeInput = New CurriculumTypeInput()

Dim curriculum1 = New CurriculumListType()

curriculum1.ID = 1

curriculum1.CurriculumDetails = "Basic"

Dim curriculum2 = New CurriculumListType()

curriculum2.ID = 2

curriculum2.CurriculumDetails = "Advanced"

' Faking out the list.

Dim list = New List(Of CurriculumListType)

list.Add(curriculum1)

list.Add(curriculum2)

curriculumTypeInput.CurriculumList = list

' Act.

curriculumTypeInput.Process("Basic")

' Assert.

Assert.AreEqual(1, curriculumTypeInput.CurriculumID)

End Sub

<TestMethod()> Public Sub process\_sets\_curriculum\_id\_to\_zero\_if\_curriculum\_not\_in\_list()

' Arrange.

Dim curriculumTypeInput = New CurriculumTypeInput()

Dim curriculum1 = New CurriculumListType()

curriculum1.ID = 1

curriculum1.CurriculumDetails = "Basic"

Dim curriculum2 = New CurriculumListType()

curriculum2.ID = 2

curriculum2.CurriculumDetails = "Advanced"

' Faking out the list.

Dim list = New List(Of CurriculumListType)

list.Add(curriculum1)

list.Add(curriculum2)

curriculumTypeInput.CurriculumList = list

' Act.

curriculumTypeInput.Process("Manic")

' Assert.

Assert.AreEqual(0, curriculumTypeInput.CurriculumID)

End Sub

<TestMethod()> Public Sub process\_sets\_is\_new\_curriculum\_if\_curriculum\_not\_in\_list()

'Arrange.

Dim curriculumTypeInput = New CurriculumTypeInput()

Dim curriculum1 = New CurriculumListType()

curriculum1.ID = 1

curriculum1.CurriculumDetails = "Basic"

Dim curriculum2 = New CurriculumListType()

curriculum2.ID = 2

curriculum2.CurriculumDetails = "Advanced"

' Faking out the list.

Dim list = New List(Of CurriculumListType)

list.Add(curriculum1)

list.Add(curriculum2)

curriculumTypeInput.CurriculumList = list

' Act.

curriculumTypeInput.Process("Manic")

' Assert.

Assert.IsTrue(curriculumTypeInput.IsNewCurriculum)

End Sub

<TestMethod()> Public Sub process\_sets\_curriculum\_id\_correctly\_if\_input\_is\_lower\_case()

' Arrange.

Dim curriculumTypeInput = New CurriculumTypeInput()

Dim curriculum1 = New CurriculumListType()

curriculum1.ID = 1

curriculum1.CurriculumDetails = "Basic"

Dim curriculum2 = New CurriculumListType()

curriculum2.ID = 2

curriculum2.CurriculumDetails = "Advanced"

' Faking out the list.

Dim list = New List(Of CurriculumListType)

list.Add(curriculum1)

list.Add(curriculum2)

curriculumTypeInput.CurriculumList = list

' Act.

curriculumTypeInput.Process("advanced")

' Assert.

Assert.AreEqual(2, curriculumTypeInput.CurriculumID)

End Sub

<TestMethod()> Public Sub process\_sets\_curriculum\_id\_correctly\_if\_input\_is\_upper\_case()

' Arrange.

Dim curriculumTypeInput = New CurriculumTypeInput()

Dim curriculum1 = New CurriculumListType()

curriculum1.ID = 1

curriculum1.CurriculumDetails = "Basic"

Dim curriculum2 = New CurriculumListType()

curriculum2.ID = 2

curriculum2.CurriculumDetails = "Advanced"

' Faking out the list.

Dim list = New List(Of CurriculumListType)

list.Add(curriculum1)

list.Add(curriculum2)

curriculumTypeInput.CurriculumList = list

' Act.

curriculumTypeInput.Process("ADVANCED")

' Assert.

Assert.AreEqual(2, curriculumTypeInput.CurriculumID)

End Sub

<TestMethod()> Public Sub process\_sets\_subject\_id\_correctly\_if\_input\_is\_mixed\_case()

' Arrange.

Dim curriculumTypeInput = New CurriculumTypeInput()

Dim curriculum1 = New CurriculumListType()

curriculum1.ID = 1

curriculum1.CurriculumDetails = "Basic"

Dim curriculum2 = New CurriculumListType()

curriculum2.ID = 2

curriculum2.CurriculumDetails = "Advanced"

' Faking out the list.

Dim list = New List(Of CurriculumListType)

list.Add(curriculum1)

list.Add(curriculum2)

curriculumTypeInput.CurriculumList = list

' Act.

curriculumTypeInput.Process("AdVaNcEd")

' Assert.

Assert.AreEqual(2, curriculumTypeInput.CurriculumID)

End Sub

End Class

End Namespace

#### DemonstrationTypeInputTests.vb

Imports ExampleProg.Constants

Imports ExampleProg.InputClasses.NewQuestionTypes

Imports Microsoft.VisualStudio.TestTools.UnitTesting

Imports System.Text.RegularExpressions

Namespace NewQuestTypeTests

<TestClass()> Public Class DemonstrationTypeInputTests

<TestMethod()> Public Sub test\_demo\_details\_sets\_valid\_flag\_to\_true()

' Arrange.

Dim demoStep = New DemonstrationTypeInput

demoStep.ResetValuesToDefaults()

demoStep.InsertNextStepDetails("Do this to that.")

' Act.

' Assert.

Assert.IsTrue(demoStep.IsValidDemoDetails)

End Sub

<TestMethod()> Public Sub test\_demo\_details\_sets\_valid\_flag\_to\_false()

' Arrange.

Dim demoStep = New DemonstrationTypeInput

demoStep.ResetValuesToDefaults()

demoStep.InsertNextStepDetails(String.Empty)

' Act.

' Assert.

Assert.IsFalse(demoStep.IsValidDemoDetails)

End Sub

<TestMethod()> Public Sub test\_demo\_details\_sets\_invalid\_message\_to\_empty\_string()

' Arrange.

Dim demoStep = New DemonstrationTypeInput

demoStep.ResetValuesToDefaults()

demoStep.InsertNextStepDetails("Do this to that.")

' Act.

' Assert.

Assert.AreEqual(String.Empty, demoStep.InvalidDemoDetailsMessage)

End Sub

<TestMethod()> Public Sub test\_demo\_details\_sets\_invalid\_message\_to\_not\_a\_valid\_entry()

' Arrange.

Dim demoStep = New DemonstrationTypeInput

demoStep.ResetValuesToDefaults()

demoStep.InsertNextStepDetails(String.Empty)

' Act.

' Assert.

Assert.AreEqual(CommonConstants.NotAValidInput, demoStep.InvalidDemoDetailsMessage)

End Sub

<TestMethod()> Public Sub test\_insert\_demo\_details\_sets\_regex\_with\_simple\_text()

' Arrange.

Const expectedRegEx As String = "^(?=.\*Do\ this\ to\ that\.)(?!^\s|.\*\s$).+$"

Dim demoStep = New DemonstrationTypeInput

demoStep.ResetValuesToDefaults()

demoStep.InsertNextStepDetails("Do this to that.")

' Act.

demoStep.SetRegEx()

' Assert.

Assert.AreEqual(expectedRegEx, demoStep.RegExDetails)

End Sub

<TestMethod()> Public Sub test\_insert\_demo\_details\_sets\_regex\_with\_maths\_calculation()

' Arrange.

Const input As String = "2(2x - 1) + x = 8"

Dim demoStep = New DemonstrationTypeInput

demoStep.ResetValuesToDefaults()

demoStep.InsertNextStepDetails(input)

' Act.

demoStep.SetRegEx()

Dim result = Regex.IsMatch(input, demoStep.RegExDetails)

' Assert.

Assert.IsTrue(result)

End Sub

<TestMethod()> Public Sub test\_update\_regex\_sets\_flag\_to\_true()

' Arrange.

Const input As String = "Do this to that."

Const regEx As String = "^(?=.\*Do\ this\ to\ that\.)(?!^\s|.\*\s$).+$"

Dim demoStep = New DemonstrationTypeInput

demoStep.ResetValuesToDefaults()

demoStep.InsertNextStepDetails(input)

' Act.

demoStep.SetRegEx()

demoStep.UpdateRegEx(regEx)

' Assert.

Assert.IsTrue(demoStep.IsValidRegExDetails)

End Sub

<TestMethod()> Public Sub test\_update\_regex\_sets\_flag\_to\_false()

' Arrange.

Const input As String = "Do this to that."

Dim demoStep = New DemonstrationTypeInput

demoStep.ResetValuesToDefaults()

demoStep.InsertNextStepDetails(input)

' Act.

demoStep.SetRegEx()

demoStep.UpdateRegEx(String.Empty)

' Assert.

Assert.IsFalse(demoStep.IsValidRegExDetails)

End Sub

<TestMethod()> Public Sub test\_update\_regex\_sets\_invalid\_message\_to\_empty\_string()

' Arrange.

Const input As String = "Do this to that."

Const regEx As String = "^(?=.\*Do\ this\ to\ that\.)(?!^\s|.\*\s$).+$"

Dim demoStep = New DemonstrationTypeInput

demoStep.ResetValuesToDefaults()

demoStep.InsertNextStepDetails(input)

' Act.

demoStep.SetRegEx()

demoStep.UpdateRegEx(regEx)

' Assert.

Assert.AreEqual(String.Empty, demoStep.InvalidRegExDetailsMessage)

End Sub

<TestMethod()> Public Sub test\_update\_regex\_sets\_invalid\_message\_to\_not\_a\_valid\_input()

' Arrange.

Const input As String = "Do this to that."

Dim demoStep = New DemonstrationTypeInput

demoStep.ResetValuesToDefaults()

demoStep.InsertNextStepDetails(input)

' Act.

demoStep.SetRegEx()

demoStep.UpdateRegEx(String.Empty)

' Assert.

Assert.AreEqual(CommonConstants.NotAValidInput, demoStep.InvalidRegExDetailsMessage)

End Sub

<TestMethod()> Public Sub test\_insert\_mark\_sets\_flag\_to\_true()

' Arrange.

Const input As String = "Do this to that."

Const regEx As String = "^(?=.\*Do\ this\ to\ that\.)(?!^\s|.\*\s$).+$"

Const mark As String = "5"

Dim demoStep = New DemonstrationTypeInput

demoStep.ResetValuesToDefaults()

demoStep.InsertNextStepDetails(input)

' Act.

demoStep.SetRegEx()

demoStep.UpdateRegEx(regEx)

demoStep.InsertMark(mark)

' Assert.

Assert.IsTrue(demoStep.IsValidMark)

End Sub

<TestMethod()> Public Sub test\_insert\_mark\_sets\_flag\_to\_false()

' Arrange.

Const input As String = "Do this to that."

Const regEx As String = "^(?=.\*Do\ this\ to\ that\.)(?!^\s|.\*\s$).+$"

Dim demoStep = New DemonstrationTypeInput

demoStep.ResetValuesToDefaults()

demoStep.InsertNextStepDetails(input)

' Act.

demoStep.SetRegEx()

demoStep.UpdateRegEx(regEx)

demoStep.InsertMark(String.Empty)

' Assert.

Assert.IsFalse(demoStep.IsValidMark)

End Sub

<TestMethod()> Public Sub test\_insert\_mark\_sets\_flag\_to\_false\_with\_word\_input()

' Arrange.

Const input As String = "Do this to that."

Const regEx As String = "^(?=.\*Do\ this\ to\ that\.)(?!^\s|.\*\s$).+$"

Const mark As String = "hello"

Dim demoStep = New DemonstrationTypeInput

demoStep.ResetValuesToDefaults()

demoStep.InsertNextStepDetails(input)

' Act.

demoStep.SetRegEx()

demoStep.UpdateRegEx(regEx)

demoStep.InsertMark(mark)

' Assert.

Assert.IsFalse(demoStep.IsValidMark)

End Sub

<TestMethod()> Public Sub test\_insert\_mark\_sets\_flag\_to\_false\_with\_punctuation\_input()

' Arrange.

Const input As String = "Do this to that."

Const regEx As String = "^(?=.\*Do\ this\ to\ that\.)(?!^\s|.\*\s$).+$"

Const mark As String = "!£$%^&\*()"

Dim demoStep = New DemonstrationTypeInput

demoStep.ResetValuesToDefaults()

demoStep.InsertNextStepDetails(input)

' Act.

demoStep.SetRegEx()

demoStep.UpdateRegEx(regEx)

demoStep.InsertMark(mark)

' Assert.

Assert.IsFalse(demoStep.IsValidMark)

End Sub

<TestMethod()> Public Sub test\_insert\_mark\_sets\_invalid\_message\_to\_empty\_string()

' Arrange.

Const input As String = "Do this to that."

Const regEx As String = "^(?=.\*Do\ this\ to\ that\.)(?!^\s|.\*\s$).+$"

Const mark As String = "5"

Dim demoStep = New DemonstrationTypeInput

demoStep.ResetValuesToDefaults()

demoStep.InsertNextStepDetails(input)

' Act.

demoStep.SetRegEx()

demoStep.UpdateRegEx(regEx)

demoStep.InsertMark(mark)

' Assert.

Assert.AreEqual(String.Empty, demoStep.InvalidMarkMessage)

End Sub

<TestMethod()> Public Sub test\_insert\_mark\_sets\_invalid\_message\_to\_invalid\_input()

' Arrange.

Const input As String = "Do this to that."

Const regEx As String = "^(?=.\*Do\ this\ to\ that\.)(?!^\s|.\*\s$).+$"

Dim demoStep = New DemonstrationTypeInput

demoStep.ResetValuesToDefaults()

demoStep.InsertNextStepDetails(input)

' Act.

demoStep.SetRegEx()

demoStep.UpdateRegEx(regEx)

demoStep.InsertMark(String.Empty)

' Assert.

Assert.AreEqual(CommonConstants.NotAValidInteger, demoStep.InvalidMarkMessage)

End Sub

<TestMethod()> Public Sub test\_insert\_demo\_details\_updates\_total\_amount\_of\_steps()

' Arrange.

Const input As String = "Do this to that."

Const mark As String = "5"

Dim demoStep = New DemonstrationTypeInput

demoStep.ResetValuesToDefaults()

demoStep.InsertNextStepDetails(input)

demoStep.SetRegEx()

demoStep.InsertMark(mark)

' Act.

demoStep.AddDetailsToList()

' Assert.

Assert.AreEqual(1, demoStep.AmountOfSteps)

End Sub

<TestMethod()> Public Sub test\_insert\_demo\_details\_updates\_can\_insert\_details\_to\_true()

' Arrange.

Const input As String = "Do this to that."

Const mark As String = "5"

Dim demoStep = New DemonstrationTypeInput

demoStep.ResetValuesToDefaults()

demoStep.InsertNextStepDetails(input)

demoStep.SetRegEx()

demoStep.InsertMark(mark)

' Act.

demoStep.AddDetailsToList()

' Assert.

Assert.IsTrue(demoStep.CanInsertDetailsToList)

End Sub

<TestMethod()> Public Sub test\_insert\_demo\_details\_updates\_can\_insert\_details\_to\_false()

' Arrange.

Const input As String = "Do this to that."

Dim demoStep = New DemonstrationTypeInput

demoStep.ResetValuesToDefaults()

demoStep.InsertNextStepDetails(input)

demoStep.SetRegEx()

demoStep.UpdateRegEx(String.Empty)

' Act.

demoStep.AddDetailsToList()

' Assert.

Assert.IsFalse(demoStep.CanInsertDetailsToList)

End Sub

<TestMethod()> Public Sub test\_insert\_demo\_details\_does\_not\_update\_total\_amount\_of\_steps\_without\_regex()

' Arrange.

Const input As String = "Do this to that."

Dim demoStep = New DemonstrationTypeInput

demoStep.ResetValuesToDefaults()

demoStep.InsertNextStepDetails(input)

demoStep.SetRegEx()

demoStep.UpdateRegEx(String.Empty)

' Act.

demoStep.AddDetailsToList()

' Assert.

Assert.AreEqual(0, demoStep.AmountOfSteps)

End Sub

<TestMethod()> Public Sub test\_insert\_demo\_details\_does\_not\_update\_total\_amount\_of\_steps\_without\_demo\_details()

' Arrange.

Dim demoStep = New DemonstrationTypeInput

demoStep.ResetValuesToDefaults()

demoStep.InsertNextStepDetails(String.Empty)

' Act.

demoStep.AddDetailsToList()

' Assert.

Assert.AreEqual(0, demoStep.AmountOfSteps)

End Sub

<TestMethod()> Public Sub test\_reset\_values\_to\_default\_sets\_demo\_details\_to\_empty\_string()

' Arrange.

Const input As String = "Do this to that."

Dim demo = New DemonstrationTypeInput

demo.ResetValuesToDefaults()

demo.InsertNextStepDetails(input)

demo.SetRegEx()

demo.AddDetailsToList()

' Act.

demo.ResetValuesToDefaults()

' Assert.

Assert.AreEqual(String.Empty, demo.GetDemoDetails())

End Sub

<TestMethod()> Public Sub test\_reset\_values\_to\_default\_sets\_regex\_details\_to\_empty\_string()

' Arrange.

Const input As String = "Do this to that."

Dim demo = New DemonstrationTypeInput

demo.ResetValuesToDefaults()

demo.InsertNextStepDetails(input)

demo.SetRegEx()

demo.AddDetailsToList()

' Act.

demo.ResetValuesToDefaults()

' Assert.

Assert.AreEqual(String.Empty, demo.RegExDetails)

End Sub

<TestMethod()> Public Sub test\_reset\_values\_to\_default\_sets\_demo\_list\_to\_empty\_list()

' Arrange.

Const input As String = "Do this to that."

Dim demo = New DemonstrationTypeInput

demo.ResetValuesToDefaults()

demo.InsertNextStepDetails(input)

demo.SetRegEx()

demo.AddDetailsToList()

' Act.

demo.ResetValuesToDefaults()

' Assert.

Assert.AreEqual(0, demo.DemonstrationList.Count)

End Sub

<TestMethod()> Public Sub test\_remove\_step\_from\_list\_returns\_true()

' Arrange.

Const input As String = "Do this to that."

Const mark As String = "5"

Dim demo = New DemonstrationTypeInput

demo.ResetValuesToDefaults()

demo.InsertNextStepDetails(input)

demo.SetRegEx()

demo.InsertMark(mark)

demo.AddDetailsToList()

' Act.

Dim result = demo.RemoveStepFromList(input)

' Assert.

Assert.IsTrue(result)

End Sub

<TestMethod()> Public Sub test\_remove\_step\_from\_list\_returns\_false()

' Arrange.

Const input As String = "Do this to that."

Dim demo = New DemonstrationTypeInput

demo.ResetValuesToDefaults()

demo.InsertNextStepDetails(input)

demo.SetRegEx()

demo.AddDetailsToList()

' Act.

Dim result = demo.RemoveStepFromList("Not in this list.")

' Assert.

Assert.IsFalse(result)

End Sub

<TestMethod()> Public Sub test\_remove\_step\_from\_list\_returns\_true\_with\_many\_items\_in\_list()

' Arrange.

Const input As String = "Do this to that."

Const mark1 As String = "5"

Const input2 As String = "Don't do that."

Const mark2 As String = "3"

Const input3 As String = "Oh, do this then!"

Const mark3 As String = "7"

Dim demo = New DemonstrationTypeInput

demo.ResetValuesToDefaults()

demo.InsertNextStepDetails(input)

demo.SetRegEx()

demo.InsertMark(mark1)

demo.AddDetailsToList()

demo.InsertNextStepDetails(input2)

demo.SetRegEx()

demo.InsertMark(mark2)

demo.AddDetailsToList()

demo.InsertNextStepDetails(input3)

demo.SetRegEx()

demo.InsertMark(mark3)

demo.AddDetailsToList()

' Act.

Dim result = demo.RemoveStepFromList(input2)

' Assert.

Assert.IsTrue(result)

End Sub

End Class

End Namespace

#### ExplanationTypeInputTests.vb

Imports ExampleProg.Constants

Imports ExampleProg.InputClasses.NewQuestionTypes

Imports ExampleProg.Interfaces

Imports ExampleProg.ProcedureReturnTypes

Imports Microsoft.VisualStudio.TestTools.UnitTesting

Imports Moq

Namespace NewQuestTypeTests

<TestClass()> Public Class ExplanationTypeInputTests

<TestMethod()> Public Sub process\_sets\_invalid\_title\_message\_flag\_to\_empty\_string()

' Arrange.

Dim dbConnector = New Mock(Of IDbConnector)

Dim explanationTypeInput = New ExplanationTypeInput(dbConnector.Object)

' Faking out the list.

explanationTypeInput.ExplanationList = New List(Of ExplanationListType)

' Act.

explanationTypeInput.Process("hello", "Key Stage 1")

' Assert.

Assert.AreEqual(String.Empty, explanationTypeInput.InvalidTitleMessage)

End Sub

<TestMethod()> Public Sub process\_sets\_invalid\_title\_message\_flag\_to\_not\_a\_valid\_input()

' Arrange.

Dim dbConnector = New Mock(Of IDbConnector)

Dim explanationTypeInput = New ExplanationTypeInput(dbConnector.Object)

' Faking out the list.

explanationTypeInput.ExplanationList = New List(Of ExplanationListType)

' Act.

explanationTypeInput.Process(String.Empty, String.Empty)

' Assert.

Assert.AreEqual(CommonConstants.NotAValidInput, explanationTypeInput.InvalidTitleMessage)

End Sub

<TestMethod()> Public Sub process\_sets\_explanation\_id\_correctly()

' Arrange.

Dim dbConnector = New Mock(Of IDbConnector)

Dim explanationTypeInput = New ExplanationTypeInput(dbConnector.Object)

Dim explanation1 = New ExplanationListType()

explanation1.ID = 1

explanation1.ExplanationDetail = "Basic"

Dim explanation2 = New ExplanationListType()

explanation2.ID = 2

explanation2.ExplanationDetail = "Advanced"

' Faking out the list.

Dim list = New List(Of ExplanationListType)

list.Add(explanation1)

list.Add(explanation2)

explanationTypeInput.ExplanationList = list

' Act.

explanationTypeInput.Process("Basic", String.Empty)

' Assert.

Assert.AreEqual(1, explanationTypeInput.ExplanationID)

End Sub

<TestMethod()> Public Sub process\_sets\_explanation\_id\_to\_zero\_if\_key\_stage\_not\_in\_list()

' Arrange.

Dim dbConnector = New Mock(Of IDbConnector)

Dim explanationTypeInput = New ExplanationTypeInput(dbConnector.Object)

Dim explanation1 = New ExplanationListType()

explanation1.ID = 1

explanation1.ExplanationDetail = "Basic"

Dim explanation2 = New ExplanationListType()

explanation2.ID = 2

explanation2.ExplanationDetail = "Advanced"

' Faking out the list.

Dim list = New List(Of ExplanationListType)

list.Add(explanation1)

list.Add(explanation2)

explanationTypeInput.ExplanationList = list

' Act.

explanationTypeInput.Process("Manic", String.Empty)

' Assert.

Assert.AreEqual(0, explanationTypeInput.ExplanationID)

End Sub

<TestMethod()> Public Sub process\_sets\_explanation\_id\_correctly\_if\_input\_is\_lower\_case()

' Arrange.

Dim dbConnector = New Mock(Of IDbConnector)

Dim explanationTypeInput = New ExplanationTypeInput(dbConnector.Object)

Dim explanation1 = New ExplanationListType()

explanation1.ID = 1

explanation1.ExplanationDetail = "Basic"

Dim explanation2 = New ExplanationListType()

explanation2.ID = 2

explanation2.ExplanationDetail = "Advanced"

' Faking out the list.

Dim list = New List(Of ExplanationListType)

list.Add(explanation1)

list.Add(explanation2)

explanationTypeInput.ExplanationList = list

' Act.

explanationTypeInput.Process("advanced", String.Empty)

' Assert.

Assert.AreEqual(2, explanationTypeInput.ExplanationID)

End Sub

<TestMethod()> Public Sub process\_sets\_explanation\_id\_correctly\_if\_input\_is\_upper\_case()

' Arrange.

Dim dbConnector = New Mock(Of IDbConnector)

Dim explanationTypeInput = New ExplanationTypeInput(dbConnector.Object)

Dim explanation1 = New ExplanationListType()

explanation1.ID = 1

explanation1.ExplanationDetail = "Basic"

Dim explanation2 = New ExplanationListType()

explanation2.ID = 2

explanation2.ExplanationDetail = "Advanced"

' Faking out the list.

Dim list = New List(Of ExplanationListType)

list.Add(explanation1)

list.Add(explanation2)

explanationTypeInput.ExplanationList = list

' Act.

explanationTypeInput.Process("ADVANCED", String.Empty)

' Assert.

Assert.AreEqual(2, explanationTypeInput.ExplanationID)

End Sub

<TestMethod()> Public Sub process\_sets\_explanation\_id\_correctly\_if\_input\_is\_mixed\_case()

' Arrange.

Dim dbConnector = New Mock(Of IDbConnector)

Dim explanationTypeInput = New ExplanationTypeInput(dbConnector.Object)

Dim explanation1 = New ExplanationListType()

explanation1.ID = 1

explanation1.ExplanationDetail = "Basic"

Dim explanation2 = New ExplanationListType()

explanation2.ID = 2

explanation2.ExplanationDetail = "Advanced"

' Faking out the list.

Dim list = New List(Of ExplanationListType)

list.Add(explanation1)

list.Add(explanation2)

explanationTypeInput.ExplanationList = list

' Act.

explanationTypeInput.Process("AdVaNcEd", String.Empty)

' Assert.

Assert.AreEqual(2, explanationTypeInput.ExplanationID)

End Sub

<TestMethod()> Public Sub process\_sets\_invalid\_detail\_message\_flag\_to\_empty\_string()

' Arrange.

Dim dbConnector = New Mock(Of IDbConnector)

Dim explanationTypeInput = New ExplanationTypeInput(dbConnector.Object)

' Faking out the list.

explanationTypeInput.ExplanationList = New List(Of ExplanationListType)

' Act.

explanationTypeInput.Process("hello", "Key Stage 1")

' Assert.

Assert.AreEqual(String.Empty, explanationTypeInput.InvalidDetailMessage)

End Sub

<TestMethod()> Public Sub process\_sets\_invalid\_detail\_message\_flag\_to\_not\_a\_valid\_input()

' Arrange.

Dim dbConnector = New Mock(Of IDbConnector)

Dim explanationTypeInput = New ExplanationTypeInput(dbConnector.Object)

' Faking out the list.

explanationTypeInput.ExplanationList = New List(Of ExplanationListType)

' Act.

explanationTypeInput.Process(String.Empty, String.Empty)

' Assert.

Assert.AreEqual(CommonConstants.NotAValidInput, explanationTypeInput.InvalidDetailMessage)

End Sub

<TestMethod()> Public Sub process\_sets\_details\_from\_db\_if\_id\_exists()

' Arrange.

' Faking out the list.

Dim explanation1 = New ExplanationListType()

explanation1.ID = 1

explanation1.ExplanationDetail = "Basic"

Dim explanation2 = New ExplanationListType()

explanation2.ID = 2

explanation2.ExplanationDetail = "Advanced"

Dim explanationList = New List(Of ExplanationListType)

explanationList.Add(explanation1)

explanationList.Add(explanation2)

Dim dbConnector = New Mock(Of IDbConnector)

dbConnector.Setup(Function(c) c.GetListOfExplanations).Returns(explanationList)

Const explanationDetails As String = "A details explanation of Advance."

dbConnector.Setup(Function(d) d.GetExplanationDetailsById(It.IsAny(Of Integer))).Returns(explanationDetails)

Dim explanationTypeInput = New ExplanationTypeInput(dbConnector.Object)

' Act.

explanationTypeInput.Process("Advanced", String.Empty)

' Assert.

Assert.AreEqual(explanationDetails, explanationTypeInput.ExplanationText)

End Sub

<TestMethod()> Public Sub process\_sets\_can\_proceed\_to\_true()

' Arrange.

Dim dbConnector = New Mock(Of IDbConnector)

Dim explanationTypeInput = New ExplanationTypeInput(dbConnector.Object)

Dim explanation1 = New ExplanationListType()

explanation1.ID = 1

explanation1.ExplanationDetail = "KS1"

Dim explanation2 = New ExplanationListType()

explanation2.ID = 2

explanation2.ExplanationDetail = "KS2"

Const explanationDetails As String = "A details explanation of KS1."

dbConnector.Setup(Function(d) d.GetExplanationDetailsById(It.IsAny(Of Integer))).Returns(explanationDetails)

' Faking out the list.

Dim list = New List(Of ExplanationListType)

list.Add(explanation1)

list.Add(explanation2)

explanationTypeInput.ExplanationList = list

' Act.

explanationTypeInput.Process("KS1", String.Empty)

' Assert.

Assert.IsTrue(explanationTypeInput.CanProceed)

End Sub

<TestMethod()> Public Sub process\_sets\_can\_proceed\_to\_false\_with\_empty\_title()

' Arrange.

Dim dbConnector = New Mock(Of IDbConnector)

Dim explanationTypeInput = New ExplanationTypeInput(dbConnector.Object)

' Faking out the list.

explanationTypeInput.ExplanationList = New List(Of ExplanationListType)

' Act.

explanationTypeInput.Process(String.Empty, "Key Stage 1")

' Assert.

Assert.IsFalse(explanationTypeInput.CanProceed)

End Sub

<TestMethod()> Public Sub process\_sets\_can\_proceed\_to\_false\_with\_empty\_detail()

' Arrange.

Dim dbConnector = New Mock(Of IDbConnector)

Dim explanationTypeInput = New ExplanationTypeInput(dbConnector.Object)

' Faking out the list.

explanationTypeInput.ExplanationList = New List(Of ExplanationListType)

' Act.

explanationTypeInput.Process("KS1", String.Empty)

' Assert.

Assert.IsFalse(explanationTypeInput.CanProceed)

End Sub

<TestMethod()> Public Sub process\_sets\_is\_new\_explanation\_if\_explanation\_not\_in\_list()

'Arrange.

Dim dbConnector = New Mock(Of IDbConnector)

Dim explanationTypeInput = New ExplanationTypeInput(dbConnector.Object)

Dim explanation1 = New ExplanationListType()

explanation1.ID = 1

explanation1.ExplanationDetail = "Basic"

Dim explanation2 = New ExplanationListType()

explanation2.ID = 2

explanation2.ExplanationDetail = "Advanced"

' Faking out the list.

Dim list = New List(Of ExplanationListType)

list.Add(explanation1)

list.Add(explanation2)

explanationTypeInput.ExplanationList = list

' Act.

explanationTypeInput.Process("Manic", String.Empty)

' Assert.

Assert.IsTrue(explanationTypeInput.IsNewExplanation)

End Sub

<TestMethod()> Public Sub process\_sets\_explanation\_list()

' Arrange.

' Faking out the list.

Dim explanation1 = New ExplanationListType()

explanation1.ID = 1

explanation1.ExplanationDetail = "Basic"

Dim explanation2 = New ExplanationListType()

explanation2.ID = 2

explanation2.ExplanationDetail = "Advanced"

Dim explanationList = New List(Of ExplanationListType)

explanationList.Add(explanation1)

explanationList.Add(explanation2)

Dim otherList As IEnumerable(Of ExplanationListType)

otherList = explanationList

Dim dbConnector = New Mock(Of IDbConnector)

dbConnector.Setup(Function(c) c.GetListOfExplanations()).Returns(otherList)

Dim explanationTypeInput = New ExplanationTypeInput(dbConnector.Object)

' Act.

explanationTypeInput.Process("Advanced", String.Empty)

Dim result = explanationList.SequenceEqual(explanationTypeInput.ExplanationList)

' Assert.

Assert.IsTrue(result)

End Sub

End Class

End Namespace

#### KeyStageTypeInputTests.vb

Imports ExampleProg.Constants

Imports ExampleProg.InputClasses.NewQuestionTypes

Imports ExampleProg.ProcedureReturnTypes

Imports Microsoft.VisualStudio.TestTools.UnitTesting

Namespace NewQuestTypeTests

<TestClass()> Public Class KeyStageTypeInputTests

<TestMethod()> Public Sub process\_sets\_can\_proceed\_to\_true()

' Arrange.

Dim keyStageTypeInput = New KeyStageTypeInput()

Dim keystage1 = New KeyStageListType()

keystage1.ID = 1

keystage1.KeyStageDetail = "KS1"

Dim keystage2 = New KeyStageListType()

keystage2.ID = 2

keystage2.KeyStageDetail = "KS2"

' Faking out the list.

Dim list = New List(Of KeyStageListType)

list.Add(keystage1)

list.Add(keystage2)

keyStageTypeInput.KeyStageList = list

' Act.

keyStageTypeInput.Process("KS1")

' Assert.

Assert.IsTrue(keyStageTypeInput.CanProceed)

End Sub

<TestMethod()> Public Sub process\_sets\_can\_proceed\_to\_false()

' Arrange.

Dim keyStageTypeInput = New KeyStageTypeInput()

' Faking out the list.

keyStageTypeInput.KeyStageList = New List(Of KeyStageListType)

' Act.

keyStageTypeInput.Process(String.Empty)

' Assert.

Assert.IsFalse(keyStageTypeInput.CanProceed)

End Sub

<TestMethod()> Public Sub process\_sets\_invalid\_message\_flag\_to\_empty\_string()

' Arrange.

Dim keyStageTypeInput = New KeyStageTypeInput()

' Faking out the list.

keyStageTypeInput.KeyStageList = New List(Of KeyStageListType)

' Act.

keyStageTypeInput.Process("hello")

' Assert.

Assert.AreEqual(String.Empty, keyStageTypeInput.InvalidMessage)

End Sub

<TestMethod()> Public Sub process\_sets\_invalid\_message\_flag\_to\_not\_a\_valid\_input()

' Arrange.

Dim keyStageTypeInput = New KeyStageTypeInput()

' Faking out the list.

keyStageTypeInput.KeyStageList = New List(Of KeyStageListType)

' Act.

keyStageTypeInput.Process(String.Empty)

' Assert.

Assert.AreEqual(CommonConstants.NotAValidInput, keyStageTypeInput.InvalidMessage)

End Sub

<TestMethod()> Public Sub process\_sets\_key\_stage\_id\_correctly()

' Arrange.

Dim keyStageTypeInput = New KeyStageTypeInput()

Dim keystage1 = New KeyStageListType()

keystage1.ID = 1

keystage1.KeyStageDetail = "Basic"

Dim keystage2 = New KeyStageListType()

keystage2.ID = 2

keystage2.KeyStageDetail = "Advanced"

' Faking out the list.

Dim list = New List(Of KeyStageListType)

list.Add(keystage1)

list.Add(keystage2)

keyStageTypeInput.KeyStageList = list

' Act.

keyStageTypeInput.Process("Basic")

' Assert.

Assert.AreEqual(1, keyStageTypeInput.KeyStageID)

End Sub

<TestMethod()> Public Sub process\_sets\_key\_stage\_id\_to\_zero\_if\_key\_stage\_not\_in\_list()

' Arrange.

Dim keyStageTypeInput = New KeyStageTypeInput()

Dim keystage1 = New KeyStageListType()

keystage1.ID = 1

keystage1.KeyStageDetail = "Basic"

Dim keystage2 = New KeyStageListType()

keystage2.ID = 2

keystage2.KeyStageDetail = "Advanced"

' Faking out the list.

Dim list = New List(Of KeyStageListType)

list.Add(keystage1)

list.Add(keystage2)

keyStageTypeInput.KeyStageList = list

' Act.

keyStageTypeInput.Process("Manic")

' Assert.

Assert.AreEqual(0, keyStageTypeInput.KeyStageID)

End Sub

<TestMethod()> Public Sub process\_sets\_is\_new\_key\_stage\_if\_key\_stage\_not\_in\_list()

'Arrange.

Dim keyStageTypeInput = New KeyStageTypeInput()

Dim keystage1 = New KeyStageListType()

keystage1.ID = 1

keystage1.KeyStageDetail = "Basic"

Dim keystage2 = New KeyStageListType()

keystage2.ID = 2

keystage2.KeyStageDetail = "Advanced"

' Faking out the list.

Dim list = New List(Of KeyStageListType)

list.Add(keystage1)

list.Add(keystage2)

keyStageTypeInput.KeyStageList = list

' Act.

keyStageTypeInput.Process("Manic")

' Assert.

Assert.IsTrue(keyStageTypeInput.IsNewKeyStage)

End Sub

<TestMethod()> Public Sub process\_sets\_key\_stage\_id\_correctly\_if\_input\_is\_lower\_case()

' Arrange.

Dim keyStageTypeInput = New KeyStageTypeInput()

Dim keystage1 = New KeyStageListType()

keystage1.ID = 1

keystage1.KeyStageDetail = "Basic"

Dim keystage2 = New KeyStageListType()

keystage2.ID = 2

keystage2.KeyStageDetail = "Advanced"

' Faking out the list.

Dim list = New List(Of KeyStageListType)

list.Add(keystage1)

list.Add(keystage2)

keyStageTypeInput.KeyStageList = list

' Act.

keyStageTypeInput.Process("advanced")

' Assert.

Assert.AreEqual(2, keyStageTypeInput.KeyStageID)

End Sub

<TestMethod()> Public Sub process\_sets\_key\_stage\_id\_correctly\_if\_input\_is\_upper\_case()

' Arrange.

Dim keyStageTypeInput = New KeyStageTypeInput()

Dim keystage1 = New KeyStageListType()

keystage1.ID = 1

keystage1.KeyStageDetail = "Basic"

Dim keystage2 = New KeyStageListType()

keystage2.ID = 2

keystage2.KeyStageDetail = "Advanced"

' Faking out the list.

Dim list = New List(Of KeyStageListType)

list.Add(keystage1)

list.Add(keystage2)

keyStageTypeInput.KeyStageList = list

' Act.

keyStageTypeInput.Process("ADVANCED")

' Assert.

Assert.AreEqual(2, keyStageTypeInput.KeyStageID)

End Sub

<TestMethod()> Public Sub process\_sets\_key\_stage\_id\_correctly\_if\_input\_is\_mixed\_case()

' Arrange.

Dim keyStageTypeInput = New KeyStageTypeInput()

Dim keystage1 = New KeyStageListType()

keystage1.ID = 1

keystage1.KeyStageDetail = "Basic"

Dim keystage2 = New KeyStageListType()

keystage2.ID = 2

keystage2.KeyStageDetail = "Advanced"

' Faking out the list.

Dim list = New List(Of KeyStageListType)

list.Add(keystage1)

list.Add(keystage2)

keyStageTypeInput.KeyStageList = list

' Act.

keyStageTypeInput.Process("AdVaNcEd")

' Assert.

Assert.AreEqual(2, keyStageTypeInput.KeyStageID)

End Sub

End Class

End Namespace

#### QuestionTypeInputTests.vb

Imports ExampleProg.Constants

Imports ExampleProg.InputClasses.NewQuestionTypes

Imports Microsoft.VisualStudio.TestTools.UnitTesting

Namespace NewQuestTypeTests

<TestClass()> Public Class QuestionTypeInputTests

<TestMethod()> Public Sub test\_process\_sets\_text\_flag\_to\_true()

' Arrange.

Dim questionType = New QuestionTypeInput()

' Act.

questionType.Process("What is something?", "www.something.com", "Life", "Questions", "N/A", "Um..")

' Assert.

Assert.IsTrue(questionType.IsValidQuestion)

End Sub

<TestMethod()> Public Sub test\_process\_sets\_text\_flag\_to\_false()

' Arrange.

Dim questionType = New QuestionTypeInput()

' Act.

questionType.Process(String.Empty, "Life", "www.something.com", "Questions", "N/A", "Um..")

' Assert.

Assert.IsFalse(questionType.IsValidQuestion)

End Sub

' Tested more thoroughly in Helpers tests.

<TestMethod()> Public Sub test\_process\_sets\_search\_string\_correctly()

' Arrange.

Const expected As String = """Life"" + ""Questions"" + ""Um.."""

Dim questionType = New QuestionTypeInput()

' Act.

questionType.Process("What is something?", "www.something.com", "Life", "Questions", "N/A", "Um..")

' Assert.

Assert.AreEqual(expected, questionType.SearchString)

End Sub

<TestMethod()> Public Sub test\_process\_sets\_search\_flag\_to\_true()

' Arrange.

Dim questionType = New QuestionTypeInput()

' Act.

questionType.Process("What is something?", "www.something.com", "Life", "Questions", "N/A", "Um..")

' Assert.

Assert.IsTrue(questionType.IsValidSerchString)

End Sub

<TestMethod()> Public Sub test\_process\_sets\_search\_flag\_to\_false\_with\_no\_subject()

' Arrange.

Dim questionType = New QuestionTypeInput()

' Act.

questionType.Process("What is something?", "www.something.com", String.Empty, "Questions", "N/A", "Um..")

' Assert.

Assert.IsFalse(questionType.IsValidSerchString)

End Sub

<TestMethod()> Public Sub test\_process\_sets\_search\_flag\_to\_false\_with\_no\_curriculum()

' Arrange.

Dim questionType = New QuestionTypeInput()

' Act.

questionType.Process("What is something?", "www.something.com", "Life", String.Empty, "N/A", "Um..")

' Assert.

Assert.IsFalse(questionType.IsValidSerchString)

End Sub

<TestMethod()> Public Sub test\_process\_sets\_search\_flag\_to\_false\_with\_no\_key\_stage()

' Arrange.

Dim questionType = New QuestionTypeInput()

' Act.

questionType.Process("What is something?", "www.something.com", "Life", "Questions", String.Empty, "Um..")

' Assert.

Assert.IsFalse(questionType.IsValidSerchString)

End Sub

<TestMethod()> Public Sub test\_process\_sets\_search\_flag\_to\_false\_with\_no\_explanation()

' Arrange.

Dim questionType = New QuestionTypeInput()

' Act.

questionType.Process("What is something?", "www.something.com", "Life", "Questions", "N/A", String.Empty)

' Assert.

Assert.IsFalse(questionType.IsValidSerchString)

End Sub

<TestMethod()> Public Sub test\_process\_sets\_invalid\_question\_message\_to\_empty\_string()

' Arrange.

Dim questionType = New QuestionTypeInput()

' Act.

questionType.Process("What is something?", "www.something.com", "Life", "Questions", "N/A", "Um..")

' Assert.

Assert.AreEqual(String.Empty, questionType.InvalidQuestionMessage)

End Sub

<TestMethod()> Public Sub test\_process\_sets\_invalid\_question\_message\_to\_not\_a\_valid\_input()

' Arrange.

Dim questionType = New QuestionTypeInput()

' Act.

questionType.Process(String.Empty, "www.something.com", "Life", "Questions", "N/A", "Um..")

' Assert.

Assert.AreEqual(CommonConstants.NotAValidInput, questionType.InvalidQuestionMessage)

End Sub

<TestMethod()> Public Sub test\_process\_sets\_invalid\_search\_string\_to\_empty\_string()

' Arrange.

Dim questionType = New QuestionTypeInput()

' Act.

questionType.Process("What is something?", "www.something.com", "Life", "Questions", "N/A", "Um..")

' Assert.

Assert.AreEqual(String.Empty, questionType.InvalidSearchStringMessage)

End Sub

<TestMethod()> Public Sub test\_process\_sets\_invalid\_search\_string\_to\_not\_a\_valid\_input\_if\_no\_subject()

' Arrange.

Dim questionType = New QuestionTypeInput()

' Act.

questionType.Process("What is something?", "www.something.com", String.Empty, "Questions", "N/A", "Um..")

' Assert.

Assert.AreEqual(CommonConstants.NotAValidInput, questionType.InvalidSearchStringMessage)

End Sub

<TestMethod()> Public Sub test\_process\_sets\_invalid\_search\_string\_to\_not\_a\_valid\_input\_if\_no\_curriculum()

' Arrange.

Dim questionType = New QuestionTypeInput()

' Act.

questionType.Process("What is something?", "www.something.com", "Life", String.Empty, "N/A", "Um..")

' Assert.

Assert.AreEqual(CommonConstants.NotAValidInput, questionType.InvalidSearchStringMessage)

End Sub

<TestMethod()> Public Sub test\_process\_sets\_invalid\_search\_string\_to\_not\_a\_valid\_input\_if\_no\_key\_stage()

' Arrange.

Dim questionType = New QuestionTypeInput()

' Act.

questionType.Process("What is something?", "www.something.com", "Life", "Questions", String.Empty, "Um..")

' Assert.

Assert.AreEqual(CommonConstants.NotAValidInput, questionType.InvalidSearchStringMessage)

End Sub

<TestMethod()> Public Sub test\_process\_sets\_invalid\_search\_string\_to\_not\_a\_valid\_input\_if\_no\_explanation()

' Arrange.

Dim questionType = New QuestionTypeInput()

' Act.

questionType.Process("What is something?", "www.something.com", "Life", "Questions", "N/A", String.Empty)

' Assert.

Assert.AreEqual(CommonConstants.NotAValidInput, questionType.InvalidSearchStringMessage)

End Sub

<TestMethod()> Public Sub test\_process\_sets\_invalid\_web\_address\_with\_www\_to\_empty\_string()

' Arrange.

Dim questionType = New QuestionTypeInput()

' Act.

questionType.Process("What is something?", "www.something.com", "Life", "Questions", "N/A", "Um..")

' Assert.

Assert.AreEqual(String.Empty, questionType.InvalidWebAddressMessage)

End Sub

<TestMethod()> Public Sub test\_process\_sets\_invalid\_web\_address\_with\_http\_to\_empty\_string()

' Arrange.

Dim questionType = New QuestionTypeInput()

' Act.

questionType.Process("What is something?", "http://www.something.com", "Life", "Questions", "N/A", "Um..")

' Assert.

Assert.AreEqual(String.Empty, questionType.InvalidWebAddressMessage)

End Sub

<TestMethod()> Public Sub test\_process\_sets\_invalid\_web\_address\_with\_https\_to\_empty\_string()

' Arrange.

Dim questionType = New QuestionTypeInput()

' Act.

questionType.Process("What is something?", "https://www.something.com", "Life", "Questions", "N/A", "Um..")

' Assert.

Assert.AreEqual(String.Empty, questionType.InvalidWebAddressMessage)

End Sub

<TestMethod()> Public Sub test\_process\_sets\_invalid\_web\_address\_to\_not\_a\_valid\_input()

' Arrange.

Dim questionType = New QuestionTypeInput()

' Act.

questionType.Process("What is something?", "ww.umbongo.org", "Life", "Questions", "N/A", "Um..")

' Assert.

Assert.AreEqual(CommonConstants.WebAddressNotValid, questionType.InvalidWebAddressMessage)

End Sub

<TestMethod()> Public Sub test\_process\_sets\_invalid\_web\_address\_with\_rubbish\_to\_not\_a\_valid\_input()

' Arrange.

Dim questionType = New QuestionTypeInput()

' Act.

questionType.Process("What is something?", "oh dear what can the matter be", "Life", "Questions", "N/A", "Um..")

' Assert.

Assert.AreEqual(CommonConstants.WebAddressNotValid, questionType.InvalidWebAddressMessage)

End Sub

<TestMethod()> Public Sub test\_process\_sets\_invalid\_web\_address\_with\_almost\_valid\_address\_to\_not\_a\_valid\_input()

' Arrange.

Dim questionType = New QuestionTypeInput()

' Act.

questionType.Process("What is something?", "htp://ww.urgh.com/help", "Life", "Questions", "N/A", "Um..")

' Assert.

Assert.AreEqual(CommonConstants.WebAddressNotValid, questionType.InvalidWebAddressMessage)

End Sub

<TestMethod()> Public Sub test\_process\_sets\_invalid\_web\_address\_with\_almost\_valid\_address\_2\_to\_not\_a\_valid\_input()

' Arrange.

Dim questionType = New QuestionTypeInput()

' Act.

questionType.Process("What is something?", "ht:/ww.urgh.com/help", "Life", "Questions", "N/A", "Um..")

' Assert.

Assert.AreEqual(CommonConstants.WebAddressNotValid, questionType.InvalidWebAddressMessage)

End Sub

<TestMethod()> Public Sub test\_process\_sets\_invalid\_web\_address\_with\_almost\_valid\_address\_3\_to\_not\_a\_valid\_input()

' Arrange.

Dim questionType = New QuestionTypeInput()

' Act.

questionType.Process("What is something?", "htps:/ww.urgh.com", "Life", "Questions", "N/A", "Um..")

' Assert.

Assert.AreEqual(CommonConstants.WebAddressNotValid, questionType.InvalidWebAddressMessage)

End Sub

<TestMethod()> Public Sub test\_process\_sets\_invalid\_web\_address\_with\_almost\_valid\_address\_4\_to\_not\_a\_valid\_input()

' Arrange.

Dim questionType = New QuestionTypeInput()

' Act.

questionType.Process("What is something?", "http:/ww.urgh.com", "Life", "Questions", "N/A", "Um..")

' Assert.

Assert.AreEqual(CommonConstants.WebAddressNotValid, questionType.InvalidWebAddressMessage)

End Sub

<TestMethod()> Public Sub test\_process\_sets\_valid\_web\_address\_with\_null\_to\_empty\_string()

' Arrange.

Dim questionType = New QuestionTypeInput()

' Act.

questionType.Process("What is something?", String.Empty, "Life", "Questions", "N/A", "Um..")

' Assert.

Assert.AreEqual(String.Empty, questionType.InvalidWebAddressMessage)

End Sub

<TestMethod()> Public Sub test\_process\_sets\_can\_proceed\_to\_true()

' Arrange.

Dim questionType = New QuestionTypeInput()

' Act.

questionType.Process("What is something?", "https://www.something.com", "Life", "Questions", "N/A", "Um..")

' Assert.

Assert.IsTrue(questionType.CanProceed)

End Sub

<TestMethod()> Public Sub test\_process\_sets\_can\_proceed\_to\_false()

' Arrange.

Dim questionType = New QuestionTypeInput()

' Act.

questionType.Process(String.Empty, "https://www.something.com", "Life", "Questions", "N/A", "Um..")

' Assert.

Assert.IsFalse(questionType.CanProceed)

End Sub

End Class

End Namespace

#### SubjectTypeInputTests.vb

Imports ExampleProg.Constants

Imports ExampleProg.InputClasses.NewQuestionTypes

Imports ExampleProg.ProcedureReturnTypes

Imports Microsoft.VisualStudio.TestTools.UnitTesting

Namespace NewQuestTypeTests

<TestClass()> Public Class SubjectTypeInputTests

<TestMethod()> Public Sub process\_sets\_can\_proceed\_to\_true()

' Arrange.

Dim subjectTypeInput = New SubjectTypeInput()

Dim subject1 = New SubjectListType()

subject1.ID = 1

subject1.SubjectDetail = "English"

Dim subject2 = New SubjectListType()

subject2.ID = 2

subject2.SubjectDetail = "Maths"

' Faking out the list.

Dim list = New List(Of SubjectListType)

list.Add(subject1)

list.Add(subject2)

subjectTypeInput.SubjectList = list

' Act.

subjectTypeInput.Process("English")

' Assert.

Assert.IsTrue(subjectTypeInput.CanProceed)

End Sub

<TestMethod()> Public Sub process\_sets\_can\_proceed\_to\_false()

' Arrange.

Dim subjectTypeInput = New SubjectTypeInput()

' Faking out the list.

Dim list = New List(Of SubjectListType)

subjectTypeInput.SubjectList = list

' Act.

subjectTypeInput.Process(String.Empty)

' Assert.

Assert.IsFalse(subjectTypeInput.CanProceed)

End Sub

<TestMethod()> Public Sub process\_sets\_invalid\_message\_flag\_to\_empty\_string()

' Arrange.

Dim subjectTypeInput = New SubjectTypeInput()

' Faking out the list.

Dim list = New List(Of SubjectListType)

subjectTypeInput.SubjectList = list

' Act.

subjectTypeInput.Process("hello")

' Assert.

Assert.AreEqual(String.Empty, subjectTypeInput.InvalidMessage)

End Sub

<TestMethod()> Public Sub process\_sets\_invalid\_message\_flag\_to\_not\_a\_valid\_input()

' Arrange.

Dim subjectTypeInput = New SubjectTypeInput()

' Faking out the list.

Dim list = New List(Of SubjectListType)

subjectTypeInput.SubjectList = list

' Act.

subjectTypeInput.Process(String.Empty)

' Assert.

Assert.AreEqual(CommonConstants.NotAValidInput, subjectTypeInput.InvalidMessage)

End Sub

<TestMethod()> Public Sub process\_sets\_subject\_id\_correctly()

' Arrange.

Dim subjectTypeInput = New SubjectTypeInput()

Dim subject1 = New SubjectListType()

subject1.ID = 1

subject1.SubjectDetail = "English"

Dim subject2 = New SubjectListType()

subject2.ID = 2

subject2.SubjectDetail = "Maths"

' Faking out the list.

Dim list = New List(Of SubjectListType)

list.Add(subject1)

list.Add(subject2)

subjectTypeInput.SubjectList = list

' Act.

subjectTypeInput.Process("English")

' Assert.

Assert.AreEqual(1, subjectTypeInput.SubjectID)

End Sub

<TestMethod()> Public Sub process\_sets\_subject\_id\_to\_zero\_if\_subject\_not\_in\_list()

' Arrange.

Dim subjectTypeInput = New SubjectTypeInput()

Dim subject1 = New SubjectListType()

subject1.ID = 1

subject1.SubjectDetail = "English"

Dim subject2 = New SubjectListType()

subject2.ID = 2

subject2.SubjectDetail = "Maths"

' Faking out the list.

Dim list = New List(Of SubjectListType)

list.Add(subject1)

list.Add(subject2)

subjectTypeInput.SubjectList = list

' Act.

subjectTypeInput.Process("bananas")

' Assert.

Assert.AreEqual(0, subjectTypeInput.SubjectID)

End Sub

<TestMethod()> Public Sub process\_sets\_is\_new\_subject\_if\_subject\_not\_in\_list()

' Arrange.

Dim subjectTypeInput = New SubjectTypeInput()

Dim subject1 = New SubjectListType()

subject1.ID = 1

subject1.SubjectDetail = "English"

Dim subject2 = New SubjectListType()

subject2.ID = 2

subject2.SubjectDetail = "Maths"

' Faking out the list.

Dim list = New List(Of SubjectListType)

list.Add(subject1)

list.Add(subject2)

subjectTypeInput.SubjectList = list

' Act.

subjectTypeInput.Process("bananas")

' Assert.

Assert.IsTrue(subjectTypeInput.IsNewSubject)

End Sub

<TestMethod()> Public Sub process\_sets\_subject\_id\_correctly\_if\_input\_is\_lower\_case()

' Arrange.

Dim subjectTypeInput = New SubjectTypeInput()

Dim subject1 = New SubjectListType()

subject1.ID = 1

subject1.SubjectDetail = "English"

Dim subject2 = New SubjectListType()

subject2.ID = 2

subject2.SubjectDetail = "Maths"

' Faking out the list.

Dim list = New List(Of SubjectListType)

list.Add(subject1)

list.Add(subject2)

subjectTypeInput.SubjectList = list

' Act.

subjectTypeInput.Process("maths")

' Assert.

Assert.AreEqual(2, subjectTypeInput.SubjectID)

End Sub

<TestMethod()> Public Sub process\_sets\_subject\_id\_correctly\_if\_input\_is\_upper\_case()

' Arrange.

Dim subjectTypeInput = New SubjectTypeInput()

Dim subject1 = New SubjectListType()

subject1.ID = 1

subject1.SubjectDetail = "English"

Dim subject2 = New SubjectListType()

subject2.ID = 2

subject2.SubjectDetail = "Maths"

' Faking out the list.

Dim list = New List(Of SubjectListType)

list.Add(subject1)

list.Add(subject2)

subjectTypeInput.SubjectList = list

' Act.

subjectTypeInput.Process("MATHS")

' Assert.

Assert.AreEqual(2, subjectTypeInput.SubjectID)

End Sub

<TestMethod()> Public Sub process\_sets\_subject\_id\_correctly\_if\_input\_is\_mixed\_case()

' Arrange.

Dim subjectTypeInput = New SubjectTypeInput()

Dim subject1 = New SubjectListType()

subject1.ID = 1

subject1.SubjectDetail = "English"

Dim subject2 = New SubjectListType()

subject2.ID = 2

subject2.SubjectDetail = "Maths"

' Faking out the list.

Dim list = New List(Of SubjectListType)

list.Add(subject1)

list.Add(subject2)

subjectTypeInput.SubjectList = list

' Act.

subjectTypeInput.Process("MaThS")

' Assert.

Assert.AreEqual(2, subjectTypeInput.SubjectID)

End Sub

End Class

End Namespace

### Training Tests

#### DemonstrationTrainingTests.vb

Imports ExampleProg.InputClasses.Training

Imports ExampleProg.Interfaces

Imports Microsoft.VisualStudio.TestTools.UnitTesting

Imports Moq

Namespace TrainingTests

<TestClass()> Public Class DemonstrationTrainingTests

Private \_demoTraining As DemonstrationTraining

Private \_mockDb As Mock(Of IDbConnector)

#Region "Set up lists"

Private Function GetListOfIds() As IEnumerable(Of Integer)

Dim list As New List(Of Integer)

list.Add(2)

list.Add(3)

list.Add(6)

list.Add(7)

list.Add(12)

Return list

End Function

Private Function GetListOfDemos() As IEnumerable(Of String)

Dim demoList As New List(Of String)

demoList.Add("Hello")

demoList.Add("Oh dear")

demoList.Add("Goodbye")

Return demoList

End Function

#End Region

<TestMethod()> Public Sub test\_set\_random\_question\_id\_sets\_valid\_id\_with\_list\_of\_questions()

' Arrange.

\_mockDb = New Mock(Of IDbConnector)()

\_mockDb.Setup(Function(q) q.GetListOfQuestionIdsForTraining(It.IsAny(Of Integer), It.IsAny(Of Integer), It.IsAny(Of Integer), It.IsAny(Of Integer))).Returns(GetListOfIds())

' Act.

\_demoTraining = New DemonstrationTraining(\_mockDb.Object)

\_demoTraining.Initialise(1, 3, 5, 6)

\_demoTraining.SetRandomQuestionID()

Dim result = \_demoTraining.QuestionID

' Assert.

Assert.IsTrue(result = 2 OrElse result = 3 OrElse result = 6 OrElse result = 7 OrElse result = 12)

End Sub

<TestMethod()> Public Sub test\_set\_random\_question\_id\_sets\_specific\_id\_but\_not\_anything\_not\_in\_list()

' Arrange.

\_mockDb = New Mock(Of IDbConnector)()

\_mockDb.Setup(Function(q) q.GetListOfQuestionIdsForTraining(It.IsAny(Of Integer), It.IsAny(Of Integer), It.IsAny(Of Integer), It.IsAny(Of Integer))).Returns(GetListOfIds())

' Act.

\_demoTraining = New DemonstrationTraining(\_mockDb.Object)

\_demoTraining.Initialise(1, 3, 5, 6)

\_demoTraining.SetRandomQuestionID()

Dim result = \_demoTraining.QuestionID

' Assert

Assert.IsFalse(Not result = 2 AndAlso Not result = 3 AndAlso Not result = 6 AndAlso Not result = 7 AndAlso Not result = 12)

End Sub

<TestMethod()> Public Sub test\_set\_up\_demo\_steps\_sets\_has\_been\_set\_up\_to\_false()

' Arrange.

\_mockDb = New Mock(Of IDbConnector)()

\_demoTraining = New DemonstrationTraining(\_mockDb.Object)

\_demoTraining.Initialise(1, 3, 5, 6)

\_demoTraining.QuestionID = 12

' Act.

\_demoTraining.SetUpDemoSteps()

' Assert.

Assert.IsFalse(\_demoTraining.HasDemoStepsBeenSetUp)

End Sub

<TestMethod()> Public Sub test\_set\_up\_demo\_steps\_sets\_has\_been\_set\_up\_to\_false\_with\_question\_id\_less\_than\_zero()

' Arrange.

\_mockDb = New Mock(Of IDbConnector)()

\_demoTraining = New DemonstrationTraining(\_mockDb.Object)

\_demoTraining.Initialise(1, 3, 5, 6)

\_demoTraining.QuestionID = -1

' Act.

\_demoTraining.SetUpDemoSteps()

' Assert.

Assert.IsFalse(\_demoTraining.HasDemoStepsBeenSetUp)

End Sub

<TestMethod()> Public Sub test\_set\_up\_demo\_steps\_sets\_has\_been\_set\_up\_to\_false\_with\_no\_question\_list()

' Arrange.

\_mockDb = New Mock(Of IDbConnector)()

\_mockDb.Setup(Function(q) q.GetListOfDemoStepDetails(It.IsAny(Of Integer))).Returns(New List(Of String))

\_demoTraining = New DemonstrationTraining(\_mockDb.Object)

\_demoTraining.Initialise(1, 3, 5, 6)

\_demoTraining.QuestionID = 12

' Act.

\_demoTraining.SetUpDemoSteps()

' Assert.

Assert.IsFalse(\_demoTraining.HasDemoStepsBeenSetUp)

End Sub

<TestMethod()> Public Sub test\_set\_up\_demo\_steps\_sets\_has\_been\_set\_up\_to\_true\_with\_question\_list()

' Arrange.

\_mockDb = New Mock(Of IDbConnector)()

\_mockDb.Setup(Function(q) q.GetListOfDemoStepDetails(It.IsAny(Of Integer))).Returns(GetListOfDemos())

\_demoTraining = New DemonstrationTraining(\_mockDb.Object)

\_demoTraining.Initialise(1, 3, 5, 6)

\_demoTraining.QuestionID = 13

' Act.

\_demoTraining.SetUpDemoSteps()

' Assert.

Assert.IsTrue(\_demoTraining.HasDemoStepsBeenSetUp)

End Sub

<TestMethod()> Public Sub test\_set\_up\_demo\_steps\_sets\_count\_of\_steps\_with\_question\_list()

' Arrange.

Const expectedCount As Integer = 3

\_mockDb = New Mock(Of IDbConnector)()

\_mockDb.Setup(Function(q) q.GetListOfDemoStepDetails(It.IsAny(Of Integer))).Returns(GetListOfDemos())

\_demoTraining = New DemonstrationTraining(\_mockDb.Object)

\_demoTraining.Initialise(1, 3, 5, 6)

\_demoTraining.QuestionID = 13

' Act.

\_demoTraining.SetUpDemoSteps()

' Assert.

Assert.AreEqual(expectedCount, \_demoTraining.CountOfDemonstrationSteps)

End Sub

<TestMethod()> Public Sub test\_next\_step\_updates\_step\_details()

' Arrange.

\_mockDb = New Mock(Of IDbConnector)()

\_mockDb.Setup(Function(q) q.GetListOfDemoStepDetails(It.IsAny(Of Integer))).Returns(GetListOfDemos())

\_demoTraining = New DemonstrationTraining(\_mockDb.Object)

\_demoTraining.Initialise(1, 3, 5, 6)

\_demoTraining.QuestionID = 13

\_demoTraining.SetUpDemoSteps()

' Act.

\_demoTraining.NextStep()

' Assert.

Assert.AreEqual("Oh dear", \_demoTraining.DemonstrationStepDetails)

End Sub

<TestMethod()> Public Sub test\_next\_step\_updates\_current\_step()

' Arrange.

\_mockDb = New Mock(Of IDbConnector)()

\_mockDb.Setup(Function(q) q.GetListOfDemoStepDetails(It.IsAny(Of Integer))).Returns(GetListOfDemos())

\_demoTraining = New DemonstrationTraining(\_mockDb.Object)

\_demoTraining.Initialise(1, 3, 5, 6)

\_demoTraining.QuestionID = 13

\_demoTraining.SetUpDemoSteps()

' Act.

\_demoTraining.NextStep()

' Assert.

Assert.AreEqual(1, \_demoTraining.DemonstrationStep)

End Sub

<TestMethod()> Public Sub test\_next\_step\_cannot\_update\_if\_at\_count\_of\_steps()

' Arrange.

\_mockDb = New Mock(Of IDbConnector)()

\_mockDb.Setup(Function(q) q.GetListOfDemoStepDetails(It.IsAny(Of Integer))).Returns(GetListOfDemos())

\_demoTraining = New DemonstrationTraining(\_mockDb.Object)

\_demoTraining.Initialise(1, 3, 5, 6)

\_demoTraining.QuestionID = 13

\_demoTraining.SetUpDemoSteps()

' Act.

\_demoTraining.DemonstrationStep = 2

\_demoTraining.NextStep()

' Assert.

Assert.AreEqual(2, \_demoTraining.DemonstrationStep)

End Sub

<TestMethod()> Public Sub test\_previous\_step\_updates\_step\_details()

' Arrange.

\_mockDb = New Mock(Of IDbConnector)()

\_mockDb.Setup(Function(q) q.GetListOfDemoStepDetails(It.IsAny(Of Integer))).Returns(GetListOfDemos())

\_demoTraining = New DemonstrationTraining(\_mockDb.Object)

\_demoTraining.Initialise(1, 3, 5, 6)

\_demoTraining.QuestionID = 13

\_demoTraining.SetUpDemoSteps()

' Act.

\_demoTraining.DemonstrationStep = 2

\_demoTraining.PreviousStep()

' Assert.

Assert.AreEqual("Oh dear", \_demoTraining.DemonstrationStepDetails)

End Sub

<TestMethod()> Public Sub test\_previous\_step\_updates\_current\_step()

' Arrange.

\_mockDb = New Mock(Of IDbConnector)()

\_mockDb.Setup(Function(q) q.GetListOfDemoStepDetails(It.IsAny(Of Integer))).Returns(GetListOfDemos())

\_demoTraining = New DemonstrationTraining(\_mockDb.Object)

\_demoTraining.Initialise(1, 3, 5, 6)

\_demoTraining.QuestionID = 13

\_demoTraining.SetUpDemoSteps()

' Act.

\_demoTraining.DemonstrationStep = 2

\_demoTraining.PreviousStep()

' Assert.

Assert.AreEqual(1, \_demoTraining.DemonstrationStep)

End Sub

<TestMethod()> Public Sub test\_previous\_step\_cannot\_update\_if\_at\_beginning()

' Arrange.

\_mockDb = New Mock(Of IDbConnector)()

\_mockDb.Setup(Function(q) q.GetListOfDemoStepDetails(It.IsAny(Of Integer))).Returns(GetListOfDemos())

\_demoTraining = New DemonstrationTraining(\_mockDb.Object)

\_demoTraining.Initialise(1, 3, 5, 6)

\_demoTraining.QuestionID = 13

\_demoTraining.SetUpDemoSteps()

' Act.

\_demoTraining.DemonstrationStep = 0

\_demoTraining.PreviousStep()

' Assert.

Assert.AreEqual(0, \_demoTraining.DemonstrationStep)

End Sub

<TestMethod()> Public Sub test\_can\_proceed\_returns\_true\_when\_at\_count\_of\_steps()

' Arrange.

\_mockDb = New Mock(Of IDbConnector)()

\_mockDb.Setup(Function(q) q.GetListOfDemoStepDetails(It.IsAny(Of Integer))).Returns(GetListOfDemos())

\_demoTraining = New DemonstrationTraining(\_mockDb.Object)

\_demoTraining.Initialise(1, 3, 5, 6)

\_demoTraining.QuestionID = 13

\_demoTraining.SetUpDemoSteps()

' Act.

\_demoTraining.DemonstrationStep = 2

Dim result = \_demoTraining.CanProceed()

' Assert.

Assert.IsTrue(result)

End Sub

<TestMethod()> Public Sub test\_can\_proceed\_returns\_false\_when\_not\_at\_count\_of\_steps()

' Arrange.

\_mockDb = New Mock(Of IDbConnector)()

\_mockDb.Setup(Function(q) q.GetListOfDemoStepDetails(It.IsAny(Of Integer))).Returns(GetListOfDemos())

\_demoTraining = New DemonstrationTraining(\_mockDb.Object)

\_demoTraining.Initialise(1, 3, 5, 6)

\_demoTraining.QuestionID = 13

\_demoTraining.SetUpDemoSteps()

' Act.

\_demoTraining.DemonstrationStep = 1

Dim result = \_demoTraining.CanProceed()

' Assert.

Assert.IsFalse(result)

End Sub

End Class

End Namespace

#### ExplanationTrainingTests.vb

Imports ExampleProg.Constants

Imports ExampleProg.InputClasses.Training

Imports ExampleProg.Interfaces

Imports ExampleProg.ProcedureReturnTypes

Imports Microsoft.VisualStudio.TestTools.UnitTesting

Imports Moq

Namespace TrainingTests

<TestClass()> Public Class ExplanationTrainingTests

Private \_explanationTraining As ExplanationTraining

Private \_mockDb As Mock(Of IDbConnector)

#Region "Explanation Setups"

Private Function GetValidExplanationDetails() As GetExplanationDetailsType

Dim explanation = New GetExplanationDetailsType()

explanation.ID = 1

explanation.Title = "Hello"

explanation.DescriptionOfExplanation = "A very long description..."

Return explanation

End Function

Private Function GetInValidExplanationDetails() As GetExplanationDetailsType

Dim explanation = New GetExplanationDetailsType()

explanation.ID = 0

explanation.Title = String.Empty

explanation.DescriptionOfExplanation = String.Empty

Return explanation

End Function

#End Region

<TestMethod()> Public Sub test\_new\_explanation\_training\_with\_id\_returns\_explanation\_title()

' Arrange.

\_mockDb = New Mock(Of IDbConnector)()

\_mockDb.Setup(Function(e) e.GetExplanationDetailsForTraining(It.IsAny(Of Integer))).Returns(GetValidExplanationDetails())

' Act.

\_explanationTraining = New ExplanationTraining(1, \_mockDb.Object)

' Assert.

Assert.AreEqual(GetValidExplanationDetails().Title, \_explanationTraining.ExplanationTitle)

End Sub

<TestMethod()> Public Sub test\_new\_explanation\_training\_with\_id\_returns\_explanation\_details()

' Arrange.

\_mockDb = New Mock(Of IDbConnector)()

\_mockDb.Setup(Function(e) e.GetExplanationDetailsForTraining(It.IsAny(Of Integer))).Returns(GetValidExplanationDetails())

' Act.

\_explanationTraining = New ExplanationTraining(1, \_mockDb.Object)

' Assert.

Assert.AreEqual(GetValidExplanationDetails().DescriptionOfExplanation, \_explanationTraining.ExplanationDetails)

End Sub

<TestMethod()> Public Sub test\_new\_explanation\_training\_without\_id\_sets\_error\_message()

' Arrange.

\_mockDb = New Mock(Of IDbConnector)()

' Act.

\_explanationTraining = New ExplanationTraining(0, \_mockDb.Object)

' Assert.

Assert.AreEqual(CommonConstants.TrainingExplanationIdNotSuppliedError, \_explanationTraining.ExplanationTitle)

End Sub

<TestMethod()> Public Sub test\_new\_explanation\_training\_with\_id\_does\_not\_return\_correct\_explanation\_title()

' Arrange.

\_mockDb = New Mock(Of IDbConnector)()

\_mockDb.Setup(Function(e) e.GetExplanationDetailsForTraining(It.IsAny(Of Integer))).Returns(GetInValidExplanationDetails())

' Act.

\_explanationTraining = New ExplanationTraining(1, \_mockDb.Object)

' Assert.

Assert.AreNotEqual(GetValidExplanationDetails().Title, \_explanationTraining.ExplanationTitle)

End Sub

<TestMethod()> Public Sub test\_new\_explanation\_training\_with\_id\_does\_not\_return\_correct\_explanation\_details()

' Arrange.

\_mockDb = New Mock(Of IDbConnector)()

\_mockDb.Setup(Function(e) e.GetExplanationDetailsForTraining(It.IsAny(Of Integer))).Returns(GetInValidExplanationDetails())

' Act.

\_explanationTraining = New ExplanationTraining(1, \_mockDb.Object)

' Assert.

Assert.AreNotEqual(GetValidExplanationDetails().DescriptionOfExplanation, \_explanationTraining.ExplanationDetails)

End Sub

End Class

End Namespace

#### ImmitationTrainingTests.vb

Imports ExampleProg.InputClasses.Training

Imports ExampleProg.Interfaces

Imports ExampleProg.ProcedureReturnTypes

Imports Microsoft.VisualStudio.TestTools.UnitTesting

Imports Moq

Namespace TrainingTests

<TestClass()> Public Class ImmitationTrainingTests

Private \_immitation As ImmitationTraining

Private \_mockDb As Mock(Of IDbConnector)

#Region "Set up lists"

Private Function GetListOfImmitationStages() As IEnumerable(Of ImmitationStageListType)

Dim list As New List(Of ImmitationStageListType)

Dim item1 As New ImmitationStageListType

item1.DescriptionOfStage = "Hello"

item1.RegEx = "^(Hello)$"

item1.StageMark = 1

list.Add(item1)

Dim item2 As New ImmitationStageListType

item2.DescriptionOfStage = "how"

item2.RegEx = "^(how)$"

item2.StageMark = 1

list.Add(item2)

Dim item3 As New ImmitationStageListType

item3.DescriptionOfStage = "are"

item3.RegEx = "^(are)$"

item3.StageMark = 1

list.Add(item3)

Dim item4 As New ImmitationStageListType

item4.DescriptionOfStage = "you?"

item4.RegEx = "^(you)\?$"

item4.StageMark = 1

list.Add(item4)

Return list

End Function

#End Region

<TestMethod()> Public Sub test\_get\_demonstration\_list\_returns\_valid\_list()

' Arrange.

\_mockDb = New Mock(Of IDbConnector)()

\_mockDb.Setup(Function(x) x.GetImmitationStageSteps(It.IsAny(Of Integer))).Returns(GetListOfImmitationStages())

' Act.

\_immitation = New ImmitationTraining(\_mockDb.Object)

\_immitation.GetDemonstrationList(1)

' Assert.

Assert.IsTrue(\_immitation.DemonstrationList.Any())

End Sub

<TestMethod()> Public Sub test\_get\_demonstration\_list\_count\_returns\_valid\_list\_with\_four\_items()

' Arrange.

\_mockDb = New Mock(Of IDbConnector)()

\_mockDb.Setup(Function(x) x.GetImmitationStageSteps(It.IsAny(Of Integer))).Returns(GetListOfImmitationStages())

' Act.

\_immitation = New ImmitationTraining(\_mockDb.Object)

\_immitation.GetDemonstrationList(1)

' Assert.

Assert.AreEqual(4, \_immitation.DemonstrationList.Count())

End Sub

<TestMethod()> Public Sub test\_get\_demonstration\_list\_third\_item\_returns\_are\_as\_description()

' Arrange.

Const expected As String = "are"

\_mockDb = New Mock(Of IDbConnector)()

\_mockDb.Setup(Function(x) x.GetImmitationStageSteps(It.IsAny(Of Integer))).Returns(GetListOfImmitationStages())

' Act.

\_immitation = New ImmitationTraining(\_mockDb.Object)

\_immitation.GetDemonstrationList(1)

' Assert.

Assert.AreEqual(expected, \_immitation.DemonstrationList.ElementAt(2).DescriptionOfStage)

End Sub

<TestMethod()> Public Sub test\_get\_demonstration\_list\_sets\_has\_demo\_been\_set\_up\_returns\_true()

' Arrange.

\_mockDb = New Mock(Of IDbConnector)()

\_mockDb.Setup(Function(x) x.GetImmitationStageSteps(It.IsAny(Of Integer))).Returns(GetListOfImmitationStages())

' Act.

\_immitation = New ImmitationTraining(\_mockDb.Object)

\_immitation.GetDemonstrationList(1)

' Assert.

Assert.IsTrue(\_immitation.HasDemoStepsBeenSetUp)

End Sub

<TestMethod()> Public Sub test\_get\_demonstration\_list\_sets\_has\_demo\_been\_set\_up\_returns\_false()

' Arrange.

\_mockDb = New Mock(Of IDbConnector)()

' Act.

\_immitation = New ImmitationTraining(\_mockDb.Object)

\_immitation.GetDemonstrationList(1)

' Assert.

Assert.IsFalse(\_immitation.HasDemoStepsBeenSetUp)

End Sub

<TestMethod()> Public Sub test\_get\_demonstration\_list\_sets\_has\_demo\_been\_set\_up\_returns\_false\_with\_no\_question\_id()

' Arrange.

\_mockDb = New Mock(Of IDbConnector)()

' Act.

\_immitation = New ImmitationTraining(\_mockDb.Object)

\_immitation.GetDemonstrationList(-1)

' Assert.

Assert.IsFalse(\_immitation.HasDemoStepsBeenSetUp)

End Sub

<TestMethod()> Public Sub test\_set\_immitation\_sets\_count\_of\_steps\_correctly()

' Arrange.

Const expectedCount As Integer = 4

\_mockDb = New Mock(Of IDbConnector)()

\_mockDb.Setup(Function(x) x.GetImmitationStageSteps(It.IsAny(Of Integer))).Returns(GetListOfImmitationStages())

' Act.

\_immitation = New ImmitationTraining(\_mockDb.Object)

\_immitation.GetDemonstrationList(1)

' Assert.

Assert.AreEqual(expectedCount, \_immitation.CountOfDemonstrationSteps)

End Sub

<TestMethod()> Public Sub test\_set\_immitation\_sets\_reg\_ex\_correctly()

' Arrange.

Const expected As String = "^(Hello)$"

\_mockDb = New Mock(Of IDbConnector)()

\_mockDb.Setup(Function(x) x.GetImmitationStageSteps(It.IsAny(Of Integer))).Returns(GetListOfImmitationStages())

' Act.

\_immitation = New ImmitationTraining(\_mockDb.Object)

\_immitation.GetDemonstrationList(1)

' Assert.

Assert.AreEqual(expected, \_immitation.DemonstrationStepRegEx)

End Sub

<TestMethod()> Public Sub test\_set\_immitation\_sets\_stage\_mark\_correctly()

' Arrange.

Const expected As Integer = 1

\_mockDb = New Mock(Of IDbConnector)()

\_mockDb.Setup(Function(x) x.GetImmitationStageSteps(It.IsAny(Of Integer))).Returns(GetListOfImmitationStages())

' Act.

\_immitation = New ImmitationTraining(\_mockDb.Object)

\_immitation.GetDemonstrationList(1)

' Assert.

Assert.AreEqual(expected, \_immitation.DemonstrationStepMark)

End Sub

<TestMethod()> Public Sub test\_set\_immitation\_sets\_stage\_details\_correctly()

' Arrange.

Const expected As String = "Hello"

\_mockDb = New Mock(Of IDbConnector)()

\_mockDb.Setup(Function(x) x.GetImmitationStageSteps(It.IsAny(Of Integer))).Returns(GetListOfImmitationStages())

' Act.

\_immitation = New ImmitationTraining(\_mockDb.Object)

\_immitation.GetDemonstrationList(1)

' Assert.

Assert.AreEqual(expected, \_immitation.DemonstrationStepDetails)

End Sub

<TestMethod()> Public Sub test\_set\_immitation\_sets\_step\_correctly()

' Arrange.

Const expected As Integer = 0

\_mockDb = New Mock(Of IDbConnector)()

\_mockDb.Setup(Function(x) x.GetImmitationStageSteps(It.IsAny(Of Integer))).Returns(GetListOfImmitationStages())

' Act.

\_immitation = New ImmitationTraining(\_mockDb.Object)

\_immitation.GetDemonstrationList(1)

' Assert.

Assert.AreEqual(expected, \_immitation.DemonstrationStep)

End Sub

<TestMethod()> Public Sub test\_process\_input\_sets\_is\_demo\_correct\_to\_true()

' Arrange.

\_mockDb = New Mock(Of IDbConnector)()

\_mockDb.Setup(Function(x) x.GetImmitationStageSteps(It.IsAny(Of Integer))).Returns(GetListOfImmitationStages())

' Act.

\_immitation = New ImmitationTraining(\_mockDb.Object)

\_immitation.GetDemonstrationList(1)

\_immitation.ProcessInput("Hello")

' Assert.

Assert.IsTrue(\_immitation.IsDemoInputCorrect)

End Sub

<TestMethod()> Public Sub test\_process\_input\_sets\_is\_demo\_correct\_to\_false\_with\_no\_question()

' Arrange.

\_mockDb = New Mock(Of IDbConnector)()

' Act.

\_immitation = New ImmitationTraining(\_mockDb.Object)

\_immitation.GetDemonstrationList(-1)

\_immitation.ProcessInput("Hello")

' Assert.

Assert.IsFalse(\_immitation.IsDemoInputCorrect)

End Sub

<TestMethod()> Public Sub test\_process\_input\_sets\_is\_demo\_correct\_to\_false\_with\_no\_list()

' Arrange.

\_mockDb = New Mock(Of IDbConnector)()

' Act.

\_immitation = New ImmitationTraining(\_mockDb.Object)

\_immitation.GetDemonstrationList(1)

\_immitation.ProcessInput("Hello")

' Assert.

Assert.IsFalse(\_immitation.IsDemoInputCorrect)

End Sub

<TestMethod()> Public Sub test\_process\_input\_sets\_is\_demo\_correct\_to\_false\_with\_null\_input()

' Arrange.

\_mockDb = New Mock(Of IDbConnector)()

' Act.

\_immitation = New ImmitationTraining(\_mockDb.Object)

\_immitation.GetDemonstrationList(1)

\_immitation.ProcessInput(Nothing)

' Assert.

Assert.IsFalse(\_immitation.IsDemoInputCorrect)

End Sub

<TestMethod()> Public Sub test\_process\_input\_sets\_is\_demo\_correct\_to\_false\_with\_empty\_string\_input()

' Arrange.

\_mockDb = New Mock(Of IDbConnector)()

' Act.

\_immitation = New ImmitationTraining(\_mockDb.Object)

\_immitation.GetDemonstrationList(1)

\_immitation.ProcessInput(String.Empty)

' Assert.

Assert.IsFalse(\_immitation.IsDemoInputCorrect)

End Sub

<TestMethod()> Public Sub test\_next\_step\_updates\_details\_to\_next\_detail()

' Arrange.

Const expected As String = "how"

\_mockDb = New Mock(Of IDbConnector)()

\_mockDb.Setup(Function(x) x.GetImmitationStageSteps(It.IsAny(Of Integer))).Returns(GetListOfImmitationStages())

' Act.

\_immitation = New ImmitationTraining(\_mockDb.Object)

\_immitation.GetDemonstrationList(1)

\_immitation.ProcessInput("Hello")

' Assert.

Assert.AreEqual(expected, \_immitation.DemonstrationStepDetails)

End Sub

<TestMethod()> Public Sub test\_next\_step\_updates\_reg\_ex\_to\_next\_reg\_ex()

' Arrange.

Const expected As String = "^(how)$"

\_mockDb = New Mock(Of IDbConnector)()

\_mockDb.Setup(Function(x) x.GetImmitationStageSteps(It.IsAny(Of Integer))).Returns(GetListOfImmitationStages())

' Act.

\_immitation = New ImmitationTraining(\_mockDb.Object)

\_immitation.GetDemonstrationList(1)

\_immitation.ProcessInput("Hello")

' Assert.

Assert.AreEqual(expected, \_immitation.DemonstrationStepRegEx)

End Sub

<TestMethod()> Public Sub test\_next\_step\_updates\_mark\_to\_next\_mark()

' Arrange.

Const expected As Integer = 1

\_mockDb = New Mock(Of IDbConnector)()

\_mockDb.Setup(Function(x) x.GetImmitationStageSteps(It.IsAny(Of Integer))).Returns(GetListOfImmitationStages())

' Act.

\_immitation = New ImmitationTraining(\_mockDb.Object)

\_immitation.GetDemonstrationList(1)

\_immitation.ProcessInput("Hello")

' Assert.

Assert.AreEqual(expected, \_immitation.DemonstrationStepMark)

End Sub

<TestMethod()> Public Sub test\_next\_step\_updates\_step\_to\_next\_step()

' Arrange.

Const expected As Integer = 1

\_mockDb = New Mock(Of IDbConnector)()

\_mockDb.Setup(Function(x) x.GetImmitationStageSteps(It.IsAny(Of Integer))).Returns(GetListOfImmitationStages())

' Act.

\_immitation = New ImmitationTraining(\_mockDb.Object)

\_immitation.GetDemonstrationList(1)

\_immitation.ProcessInput("Hello")

' Assert.

Assert.AreEqual(expected, \_immitation.DemonstrationStep)

End Sub

<TestMethod()> Public Sub test\_next\_step\_updates\_step\_to\_next\_step\_and\_next()

' Arrange.

Const expected As Integer = 2

\_mockDb = New Mock(Of IDbConnector)()

\_mockDb.Setup(Function(x) x.GetImmitationStageSteps(It.IsAny(Of Integer))).Returns(GetListOfImmitationStages())

' Act.

\_immitation = New ImmitationTraining(\_mockDb.Object)

\_immitation.GetDemonstrationList(1)

\_immitation.ProcessInput("Hello")

\_immitation.ProcessInput("how")

' Assert.

Assert.AreEqual(expected, \_immitation.DemonstrationStep)

End Sub

End Class

End Namespace

#### SelectTrainingInputTests.vb

Imports ExampleProg.InputClasses.Training

Imports ExampleProg.Interfaces

Imports ExampleProg.ProcedureReturnTypes

Imports Microsoft.VisualStudio.TestTools.UnitTesting

Imports Moq

Namespace TrainingTests

''' <summary>

''' Tests for the Select Training Input class.

''' </summary>

''' <remarks></remarks>

<TestClass()> Public Class SelectTrainingInputTests

Private \_selectTraining As SelectTrainingInput

Private \_mockDb As Mock(Of IDbConnector)

#Region "Set up fake lists."

Private Function GetSubjectList() As List(Of SubjectListType)

Dim subjectList = New List(Of SubjectListType)

Dim subject1 = New SubjectListType()

subject1.ID = 1

subject1.SubjectDetail = "English"

Dim subject2 = New SubjectListType()

subject2.ID = 2

subject2.SubjectDetail = "Maths"

subjectList.Add(subject1)

subjectList.Add(subject2)

Return subjectList

End Function

Private Function GetCurriculumList() As List(Of CurriculumListType)

Dim curriculumList = New List(Of CurriculumListType)

Dim curr1 = New CurriculumListType()

curr1.ID = 1

curr1.CurriculumDetails = "Bananas"

Dim curr2 = New CurriculumListType()

curr2.ID = 2

curr2.CurriculumDetails = "Apples"

curriculumList.Add(curr1)

curriculumList.Add(curr2)

Return curriculumList

End Function

Private Function GetKeyStageList() As List(Of KeyStageListType)

Dim keystage1 = New KeyStageListType()

keystage1.ID = 1

keystage1.KeyStageDetail = "KS1"

Dim keystage2 = New KeyStageListType()

keystage2.ID = 2

keystage2.KeyStageDetail = "KS2"

Dim keyStageList = New List(Of KeyStageListType)

keyStageList.Add(keystage1)

keyStageList.Add(keystage2)

Return keyStageList

End Function

Private Function GetExplanationList() As List(Of ExplanationListType)

Dim explanation1 = New ExplanationListType()

explanation1.ID = 1

explanation1.ExplanationDetail = "Basic"

Dim explanation2 = New ExplanationListType()

explanation2.ID = 2

explanation2.ExplanationDetail = "Advanced"

Dim explanationList = New List(Of ExplanationListType)

explanationList.Add(explanation1)

explanationList.Add(explanation2)

Return explanationList

End Function

#End Region

<TestMethod()> Public Sub test\_populate\_subject\_list\_returns\_correct\_count()

' Arrange.

Dim subjectList As List(Of SubjectListType) = GetSubjectList()

\_mockDb = New Mock(Of IDbConnector)

' Act.

\_mockDb.Setup(Function(s) s.GetListOfSubjects()).Returns(subjectList)

\_selectTraining = New SelectTrainingInput(\_mockDb.Object)

' Assert.

Assert.AreEqual(subjectList.Count(), \_selectTraining.SubjectList.Count())

End Sub

<TestMethod()> Public Sub test\_populate\_curriculum\_list\_returns\_correct\_count()

' Arrange.

Dim curriculumList As List(Of CurriculumListType) = GetCurriculumList()

\_mockDb = New Mock(Of IDbConnector)

' Act.

\_mockDb.Setup(Function(c) c.GetListOfCurriculum()).Returns(curriculumList)

\_selectTraining = New SelectTrainingInput(\_mockDb.Object)

\_selectTraining.PopulateCurriculumList()

' Assert.

Assert.AreEqual(curriculumList.Count(), \_selectTraining.CurriculumList.Count())

End Sub

<TestMethod()> Public Sub test\_populate\_update\_curriculum\_list\_returns\_correct\_count()

' Arrange.

Dim curriculumList As List(Of CurriculumListType) = GetCurriculumList()

\_mockDb = New Mock(Of IDbConnector)

' Act.

\_mockDb.Setup(Function(c) c.GetUpdatedListOfCurriculum(It.IsAny(Of Integer))).Returns(curriculumList)

\_selectTraining = New SelectTrainingInput(\_mockDb.Object)

\_selectTraining.PopulateUpdatedCurriculumList()

' Assert.

Assert.AreEqual(curriculumList.Count(), \_selectTraining.CurriculumList.Count())

End Sub

<TestMethod()> Public Sub test\_populate\_key\_stage\_list\_returns\_correct\_count()

' Arrange.

Dim keyStageList As List(Of KeyStageListType) = GetKeyStageList()

\_mockDb = New Mock(Of IDbConnector)

' Act.

\_mockDb.Setup(Function(c) c.GetListOfKeyStages()).Returns(keyStageList)

\_selectTraining = New SelectTrainingInput(\_mockDb.Object)

' Assert.

Assert.AreEqual(keyStageList.Count(), \_selectTraining.KeyStageList.Count())

End Sub

<TestMethod()> Public Sub test\_populate\_explanation\_list\_returns\_correct\_count()

' Arrange.

Dim explanationList As List(Of ExplanationListType) = GetExplanationList()

\_mockDb = New Mock(Of IDbConnector)

' Act.

\_mockDb.Setup(Function(c) c.GetListOfExplanations()).Returns(explanationList)

\_selectTraining = New SelectTrainingInput(\_mockDb.Object)

\_selectTraining.PopulateExplanationList()

' Assert.

Assert.AreEqual(explanationList.Count(), \_selectTraining.ExplanationList.Count())

End Sub

<TestMethod()> Public Sub test\_populate\_updated\_explanation\_list\_by\_curriculum\_id\_returns\_correct\_count()

' Arrange.

Dim explanationList As List(Of ExplanationListType) = GetExplanationList()

\_mockDb = New Mock(Of IDbConnector)

' Act.

\_mockDb.Setup(Function(c) c.GetUpdatedExplanationsByCurriculum(It.IsAny(Of Integer))).Returns(explanationList)

\_selectTraining = New SelectTrainingInput(\_mockDb.Object)

\_selectTraining.PopulateUpdatedExplanationList()

' Assert.

Assert.AreEqual(explanationList.Count(), \_selectTraining.ExplanationList.Count())

End Sub

<TestMethod()> Public Sub test\_set\_subject\_id\_returns\_correct\_id()

' Arrange.

Dim subjectList As List(Of SubjectListType) = GetSubjectList()

\_mockDb = New Mock(Of IDbConnector)

' Act.

\_mockDb.Setup(Function(s) s.GetListOfSubjects()).Returns(subjectList)

\_selectTraining = New SelectTrainingInput(\_mockDb.Object)

\_selectTraining.SelectedSubjectText = "Maths"

\_selectTraining.SetSubjectID()

' Assert.

Assert.AreEqual(2, \_selectTraining.SelectedSubjectID)

End Sub

<TestMethod()> Public Sub test\_set\_curriculum\_id\_returns\_correct\_id()

' Arrange.

Dim curriculumList As List(Of CurriculumListType) = GetCurriculumList()

\_mockDb = New Mock(Of IDbConnector)

' Act.

\_mockDb.Setup(Function(s) s.GetListOfCurriculum()).Returns(curriculumList)

\_selectTraining = New SelectTrainingInput(\_mockDb.Object)

\_selectTraining.PopulateCurriculumList()

\_selectTraining.SelectedCurriculumText = "Bananas"

\_selectTraining.SetCurriculumID()

' Assert.

Assert.AreEqual(1, \_selectTraining.SelectedCurriculumID)

End Sub

<TestMethod()> Public Sub test\_set\_key\_stage\_id\_returns\_correct\_id()

' Arrange.

Dim keyStageList As List(Of KeyStageListType) = GetKeyStageList()

\_mockDb = New Mock(Of IDbConnector)

' Act.

\_mockDb.Setup(Function(s) s.GetListOfKeyStages()).Returns(keyStageList)

\_selectTraining = New SelectTrainingInput(\_mockDb.Object)

\_selectTraining.SelectedKeyStageText = "KS2"

\_selectTraining.SetKeyStageID()

' Assert.

Assert.AreEqual(2, \_selectTraining.SelectedKeyStageID)

End Sub

<TestMethod()> Public Sub test\_set\_explanation\_id\_returns\_correct\_id()

' Arrange.

Dim explanationList As List(Of ExplanationListType) = GetExplanationList()

\_mockDb = New Mock(Of IDbConnector)

' Act.

\_mockDb.Setup(Function(s) s.GetListOfExplanations()).Returns(explanationList)

\_selectTraining = New SelectTrainingInput(\_mockDb.Object)

\_selectTraining.PopulateExplanationList()

\_selectTraining.SelectedExplanationText = "Basic"

\_selectTraining.SetExplanationID()

' Assert.

Assert.AreEqual(1, \_selectTraining.SelectedExplanationID)

End Sub

End Class

End Namespace

### HelpersTests.vb

Imports Microsoft.VisualStudio.TestTools.UnitTesting

Imports ExampleProg.Helpers

''' <summary>

''' A set of tests for the Helpers class.

''' </summary>

''' <remarks>Tests.</remarks>

<TestClass()> Public Class HelpersTests

<TestMethod()> Public Sub test\_if\_text\_not\_null\_returns\_true()

' Arrange.

Dim actual

' Act.

actual = IsTextNotNull("hello")

' Assert.

Assert.IsTrue(actual)

End Sub

<TestMethod()> Public Sub test\_if\_text\_not\_null\_returns\_false()

' Arrange.

Dim actual

' Act.

actual = IsTextNotNull(String.Empty)

' Assert.

Assert.IsFalse(actual)

End Sub

<TestMethod()> Public Sub test\_is\_digit\_null\_or\_zero\_returns\_true\_with\_positive\_digit()

' Arrange,

' Act.

' Assert.

Assert.IsTrue(IsDigitNotNullOrZero(1))

End Sub

<TestMethod()> Public Sub test\_is\_digit\_null\_or\_zero\_returns\_false\_with\_null()

' Arrange,

' Act.

' Assert.

Assert.IsFalse(IsDigitNotNullOrZero(Nothing))

End Sub

<TestMethod()> Public Sub test\_is\_digit\_null\_or\_zero\_returns\_false\_with\_zero()

' Arrange,

' Act.

' Assert.

Assert.IsFalse(IsDigitNotNullOrZero(0))

End Sub

<TestMethod()> Public Sub test\_is\_digit\_null\_or\_zero\_returns\_false\_with\_negative\_digit()

' Arrange,

' Act.

' Assert.

Assert.IsFalse(IsDigitNotNullOrZero(-1))

End Sub

<TestMethod()> Public Sub test\_is\_there\_a\_false\_in\_array\_returns\_true()

' Arrange.

Dim array(6) As Boolean

Dim result

array(0) = True

array(1) = True

array(2) = True

array(3) = True

array(4) = True

array(5) = True

array(6) = True

' Act.

result = IsThereAFalseInArray(array)

' Assert.

Assert.IsTrue(result)

End Sub

<TestMethod()> Public Sub test\_is\_there\_a\_false\_in\_array\_returns\_false()

' Arrange.

Dim array(6) As Boolean

Dim result

array(0) = True

array(1) = True

array(2) = False

array(3) = True

array(4) = True

array(5) = True

array(6) = True

' Act.

result = IsThereAFalseInArray(array)

' Assert.

Assert.IsFalse(result)

End Sub

<TestMethod()> Public Sub test\_are\_all\_values\_false\_in\_array\_returns\_false()

' Arrange.

Dim array(6) As Boolean

Dim result

array(0) = False

array(1) = False

array(2) = False

array(3) = False

array(4) = False

array(5) = False

array(6) = False

' Act.

result = IsThereAFalseInArray(array)

' Assert.

Assert.IsFalse(result)

End Sub

<TestMethod()> Public Sub test\_format\_string\_returns\_nice\_string()

' Arrange.

Const expected As String = """Maths"" + ""Basics"" + ""KS1"" + ""A simple addition"""

Dim array(3) As String

array(0) = "Maths"

array(1) = "Basics"

array(2) = "KS1"

array(3) = "A simple addition"

' Act.

Dim result = FormatSearchString(array)

' Assert.

Assert.AreEqual(expected, result.ToString())

End Sub

<TestMethod()> Public Sub test\_format\_string\_returns\_nice\_string\_skipping\_na()

' Arrange.

Const expected As String = """Maths"" + ""Basics"" + ""A simple addition"""

Dim array(3) As String

array(0) = "Maths"

array(1) = "Basics"

array(2) = "N/A"

array(3) = "A simple addition"

' Act.

Dim result = FormatSearchString(array)

' Assert.

Assert.AreEqual(expected, result.ToString())

End Sub

End Class

### LoginInpputTests.vb

Imports Microsoft.VisualStudio.TestTools.UnitTesting

Imports ExampleProg.Constants

Imports ExampleProg.InputClasses

Imports ExampleProg.Interfaces

Imports Moq

''' <summary>

''' A set of tests for the Log In Input class.

''' </summary>

''' <remarks>Tests.</remarks>

<TestClass()> Public Class LogInInputTests

<TestMethod()> Public Sub test\_log\_in\_sets\_user\_name\_valid\_states\_to\_true()

' Arrange.

Dim loginInput = New LogInInput("me", "secret")

' Act.

loginInput.SetBasicValidStates()

' Assert.

Assert.IsTrue(loginInput.IsUserNameValid)

End Sub

<TestMethod()> Public Sub test\_log\_in\_sets\_user\_name\_valid\_states\_to\_false()

' Arrange.

Dim loginInput = New LogInInput(String.Empty, "secret")

' Act.

loginInput.SetBasicValidStates()

' Assert.

Assert.IsFalse(loginInput.IsUserNameValid)

End Sub

<TestMethod()> Public Sub test\_log\_in\_sets\_password\_valid\_states\_to\_true()

' Arrange.

Dim loginInput = New LogInInput("me", "secret")

' Act.

loginInput.SetBasicValidStates()

' Assert.

Assert.IsTrue(loginInput.IsPasswordValid)

End Sub

<TestMethod()> Public Sub test\_log\_in\_sets\_password\_valid\_states\_to\_false()

' Arrange.

Dim loginInput = New LogInInput("me", String.Empty)

' Act.

loginInput.SetBasicValidStates()

' Assert.

Assert.IsFalse(loginInput.IsPasswordValid)

End Sub

<TestMethod()> Public Sub test\_log\_in\_sets\_array\_of\_valid\_states\_for\_user\_name\_to\_true()

' Arrange.

Dim loginInput = New LogInInput("me", "secret")

loginInput.SetBasicValidStates()

loginInput.IsUserNameInDatabase = False

loginInput.DoUserNameAndPasswordMatch = True

' Act.

loginInput.SetArrayOfValidStates()

' Assert.

Assert.IsTrue(loginInput.ArrayOfValidity(0))

End Sub

<TestMethod()> Public Sub test\_log\_in\_sets\_array\_of\_valid\_states\_for\_password\_to\_true()

' Arrange.

Dim loginInput = New LogInInput("me", "secret")

loginInput.SetBasicValidStates()

loginInput.IsUserNameInDatabase = False

loginInput.DoUserNameAndPasswordMatch = True

' Act.

loginInput.SetArrayOfValidStates()

' Assert.

Assert.IsTrue(loginInput.ArrayOfValidity(2))

End Sub

<TestMethod()> Public Sub test\_log\_in\_sets\_invalid\_message\_for\_user\_name()

' Arrange.

Dim loginInput = New LogInInput("me", "secret")

loginInput.SetBasicValidStates()

loginInput.IsUserNameInDatabase = False

loginInput.DoUserNameAndPasswordMatch = True

' Act.

loginInput.SetInvalidMessages()

' Assert.

Assert.AreEqual(String.Empty, loginInput.UserNameInvalidMessage)

End Sub

<TestMethod()> Public Sub test\_log\_in\_sets\_invalid\_message\_for\_user\_name\_when\_null()

' Arrange.

Dim loginInput = New LogInInput(String.Empty, "secret")

loginInput.SetBasicValidStates()

loginInput.IsUserNameInDatabase = False

loginInput.DoUserNameAndPasswordMatch = True

' Act.

loginInput.SetInvalidMessages()

' Assert.

Assert.AreEqual(CommonConstants.NotAValidInput, loginInput.UserNameInvalidMessage)

End Sub

<TestMethod()> Public Sub test\_log\_in\_sets\_invalid\_message\_for\_password()

' Arrange.

Dim loginInput = New LogInInput("me", "secret")

loginInput.SetBasicValidStates()

loginInput.IsUserNameInDatabase = False

loginInput.DoUserNameAndPasswordMatch = True

' Act.

loginInput.SetInvalidMessages()

' Assert.

Assert.AreEqual(String.Empty, loginInput.PasswordInvalidMessage)

End Sub

<TestMethod()> Public Sub test\_log\_in\_sets\_invalid\_message\_for\_password\_when\_null()

' Arrange.

Dim loginInput = New LogInInput("me", String.Empty)

loginInput.SetBasicValidStates()

loginInput.IsUserNameInDatabase = False

loginInput.DoUserNameAndPasswordMatch = True

' Act.

loginInput.SetInvalidMessages()

' Assert.

Assert.AreEqual(CommonConstants.NotAValidInput, loginInput.PasswordInvalidMessage)

End Sub

<TestMethod()> Public Sub test\_log\_in\_sets\_invalid\_message\_for\_user\_name\_that\_already\_exists()

' Arrange.

Dim loginInput = New LogInInput("me", "secret")

loginInput.SetBasicValidStates()

loginInput.IsUserNameInDatabase = True

loginInput.DoUserNameAndPasswordMatch = True

' Act.

loginInput.SetInvalidMessages()

' Assert.

Assert.AreEqual(String.Empty, loginInput.UserNameInvalidMessage)

End Sub

<TestMethod()> Public Sub test\_log\_in\_sets\_invalid\_message\_for\_invalid\_user\_and\_password()

' Arrange.

Dim loginInput = New LogInInput("me", "secret")

loginInput.SetBasicValidStates()

loginInput.IsUserNameInDatabase = True

loginInput.DoUserNameAndPasswordMatch = False

' Act.

loginInput.SetInvalidMessages()

' Assert.

Assert.AreEqual(CommonConstants.UserNameAndPasswordDoNotMatch, loginInput.OverallInvalidMessage)

End Sub

<TestMethod()> Public Sub test\_set\_user\_name\_in\_db\_to\_true()

' Arrange.

Const expected As Boolean = True

Dim dbConnector = New Mock(Of IDbConnector)

dbConnector.Setup(Function(c) c.DoesUserNameAlreadyExists(It.IsAny(Of String))).Returns(expected)

Dim loginInput = New LogInInput("me", "secret")

' Act.

loginInput.SetUserNameInDatabase(dbConnector.Object)

' Assert.

Assert.IsTrue(loginInput.IsUserNameInDatabase)

End Sub

<TestMethod()> Public Sub test\_set\_user\_name\_in\_db\_to\_false()

' Arrange.

Const expected As Boolean = False

Dim dbConnector = New Mock(Of IDbConnector)

dbConnector.Setup(Function(c) c.DoesUserNameAlreadyExists(It.IsAny(Of String))).Returns(expected)

Dim loginInput = New LogInInput("me", "secret")

' Act.

loginInput.SetUserNameInDatabase(dbConnector.Object)

' Assert.

Assert.IsFalse(loginInput.IsUserNameInDatabase)

End Sub

<TestMethod()> Public Sub test\_set\_do\_user\_name\_and\_password\_match\_returns\_true()

' Arrange.

Const expected As Integer = 4

Dim dbConnector = New Mock(Of IDbConnector)

dbConnector.Setup(Function(c) c.GetUserIdOfValidUser(It.IsAny(Of String), It.IsAny(Of String))).Returns(expected)

Dim loginInput = New LogInInput("me", "secret")

' Act.

loginInput.SetDoUserNameAndPasswordMatch(dbConnector.Object)

' Assert.

Assert.IsTrue(loginInput.DoUserNameAndPasswordMatch)

End Sub

<TestMethod()> Public Sub test\_set\_do\_user\_name\_and\_password\_match\_returns\_false()

' Arrange.

Const expected As Integer = 0

Dim dbConnector = New Mock(Of IDbConnector)

dbConnector.Setup(Function(c) c.GetUserIdOfValidUser(It.IsAny(Of String), It.IsAny(Of String))).Returns(expected)

Dim loginInput = New LogInInput("me", "secret")

' Act.

loginInput.SetDoUserNameAndPasswordMatch(dbConnector.Object)

' Assert.

Assert.IsFalse(loginInput.DoUserNameAndPasswordMatch)

End Sub

<TestMethod()> Public Sub test\_is\_there\_a\_false\_in\_array\_returs\_true\_for\_all\_good()

' Arrange.

Const userID As Integer = 6

Const userExists As Boolean = True

Dim dbConnector = New Mock(Of IDbConnector)

dbConnector.Setup(Function(c) c.DoesUserNameAlreadyExists(It.IsAny(Of String))).Returns(userExists)

dbConnector.Setup(Function(c) c.GetUserIdOfValidUser(It.IsAny(Of String), It.IsAny(Of String))).Returns(userID)

Dim loginInput = New LogInInput("me", "secret")

' Act.

loginInput.SetBasicValidStates()

loginInput.SetUserNameInDatabase(dbConnector.Object)

loginInput.SetDoUserNameAndPasswordMatch(dbConnector.Object)

loginInput.SetArrayOfValidStates()

' Assert.

Assert.IsTrue(loginInput.IsThereAFalseInTheInput())

End Sub

<TestMethod()> Public Sub test\_is\_there\_a\_false\_in\_array\_returs\_false\_for\_no\_user\_name()

' Arrange.

Const userID As Integer = 6

Const userExists As Boolean = True

Dim dbConnector = New Mock(Of IDbConnector)

dbConnector.Setup(Function(c) c.DoesUserNameAlreadyExists(It.IsAny(Of String))).Returns(userExists)

dbConnector.Setup(Function(c) c.GetUserIdOfValidUser(It.IsAny(Of String), It.IsAny(Of String))).Returns(userID)

Dim loginInput = New LogInInput(String.Empty, "secret")

' Act.

loginInput.SetBasicValidStates()

loginInput.SetUserNameInDatabase(dbConnector.Object)

loginInput.SetDoUserNameAndPasswordMatch(dbConnector.Object)

loginInput.SetArrayOfValidStates()

' Assert.

Assert.IsFalse(loginInput.IsThereAFalseInTheInput())

End Sub

<TestMethod()> Public Sub test\_is\_there\_a\_false\_in\_array\_returs\_false\_for\_no\_password()

' Arrange.

Const userID As Integer = 6

Const userExists As Boolean = True

Dim dbConnector = New Mock(Of IDbConnector)

dbConnector.Setup(Function(c) c.DoesUserNameAlreadyExists(It.IsAny(Of String))).Returns(userExists)

dbConnector.Setup(Function(c) c.GetUserIdOfValidUser(It.IsAny(Of String), It.IsAny(Of String))).Returns(userID)

Dim loginInput = New LogInInput("me", String.Empty)

' Act.

loginInput.SetBasicValidStates()

loginInput.SetUserNameInDatabase(dbConnector.Object)

loginInput.SetDoUserNameAndPasswordMatch(dbConnector.Object)

loginInput.SetArrayOfValidStates()

' Assert.

Assert.IsFalse(loginInput.IsThereAFalseInTheInput())

End Sub

End Class

### NewQuestionInputTests.vb

Imports ExampleProg.Constants

Imports ExampleProg.InputClasses

Imports ExampleProg.Interfaces

Imports ExampleProg.ProcedureReturnTypes

Imports ExampleProg.QuestionTypes

Imports Microsoft.VisualStudio.TestTools.UnitTesting

Imports Moq

''' <summary>

''' A set of tests for the New Question Input class.

''' </summary>

''' <remarks>Tests.</remarks>

<TestClass()> Public Class NewQuestionInputTests

#Region "Set up functions"

Private \_newQuestion As NewQuestionInput

Private \_mockDb As Mock(Of IDbConnector)

Private Sub SetNewQuestionDefaults()

\_mockDb = New Mock(Of IDbConnector)

\_newQuestion = New NewQuestionInput(\_mockDb.Object)

End Sub

Private Sub SetSubjectStuff()

\_newQuestion.SubjectType.SubjectList = New List(Of SubjectListType)

\_newQuestion.SubjectType.Process("OMG")

End Sub

Private Sub SetCurriculumStuff()

\_newQuestion.CurriculumType.CurriculumList = New List(Of CurriculumListType)

\_newQuestion.CurriculumType.Process("Blah")

End Sub

Private Sub SetKeyStageStuff()

\_newQuestion.KeyStageType.KeyStageList = New List(Of KeyStageListType)

\_newQuestion.KeyStageType.Process("N/A")

End Sub

Private Sub SetExplanationStuff()

\_newQuestion.ExplanationType.Process("Some where", "A log list of somthing or other")

End Sub

Private Sub SetQuestionStuff()

\_newQuestion.QuestionType.Process("What is this?", "www.whatgoesthere.com", "OMG", "Blah", "N/A", "Some where")

End Sub

Private Sub SetAnswerStuff()

\_newQuestion.AnswerType.Process("a", "b", "c", "d")

End Sub

Private Sub SetDemoStuff()

\_newQuestion.DemonstrationType.InsertNextStepDetails("You take one from here and add it to that")

\_newQuestion.DemonstrationType.UpdateRegEx("$\some\ rubbish\ here\ !$")

\_newQuestion.DemonstrationType.InsertMark("5")

\_newQuestion.DemonstrationType.AddDetailsToList()

End Sub

#End Region

<TestMethod()> Public Sub test\_process\_sets\_can\_submit\_flag\_to\_true()

' Arrange.

SetNewQuestionDefaults()

SetSubjectStuff()

SetCurriculumStuff()

SetKeyStageStuff()

SetExplanationStuff()

SetQuestionStuff()

SetAnswerStuff()

SetDemoStuff()

' Act.

\_newQuestion.Process()

' Assert.

Assert.IsTrue(\_newQuestion.CanSubmit)

End Sub

<TestMethod()> Public Sub test\_process\_sets\_can\_submit\_flag\_to\_false\_with\_no\_subject()

' Arrange.

SetNewQuestionDefaults()

SetCurriculumStuff()

SetKeyStageStuff()

SetExplanationStuff()

SetQuestionStuff()

SetAnswerStuff()

SetDemoStuff()

' Act.

\_newQuestion.Process()

' Assert.

Assert.IsFalse(\_newQuestion.CanSubmit)

End Sub

<TestMethod()> Public Sub test\_process\_sets\_can\_submit\_flag\_to\_false\_with\_no\_curriculum()

' Arrange.

SetNewQuestionDefaults()

SetSubjectStuff()

SetKeyStageStuff()

SetExplanationStuff()

SetQuestionStuff()

SetAnswerStuff()

SetDemoStuff()

' Act.

\_newQuestion.Process()

' Assert.

Assert.IsFalse(\_newQuestion.CanSubmit)

End Sub

<TestMethod()> Public Sub test\_process\_sets\_can\_submit\_flag\_to\_false\_with\_no\_key\_stage()

' Arrange.

SetNewQuestionDefaults()

SetSubjectStuff()

SetCurriculumStuff()

SetExplanationStuff()

SetQuestionStuff()

SetAnswerStuff()

SetDemoStuff()

' Act.

\_newQuestion.Process()

' Assert.

Assert.IsFalse(\_newQuestion.CanSubmit)

End Sub

<TestMethod()> Public Sub test\_process\_sets\_can\_submit\_flag\_to\_false\_with\_no\_explanation()

' Arrange.

SetNewQuestionDefaults()

SetSubjectStuff()

SetCurriculumStuff()

SetKeyStageStuff()

SetQuestionStuff()

SetAnswerStuff()

SetDemoStuff()

' Act.

\_newQuestion.Process()

' Assert.

Assert.IsFalse(\_newQuestion.CanSubmit)

End Sub

<TestMethod()> Public Sub test\_process\_sets\_can\_submit\_flag\_to\_false\_with\_no\_question()

' Arrange.

SetNewQuestionDefaults()

SetSubjectStuff()

SetCurriculumStuff()

SetKeyStageStuff()

SetExplanationStuff()

SetAnswerStuff()

SetDemoStuff()

' Act.

\_newQuestion.Process()

' Assert.

Assert.IsFalse(\_newQuestion.CanSubmit)

End Sub

<TestMethod()> Public Sub test\_process\_sets\_can\_submit\_flag\_to\_false\_with\_no\_answer()

' Arrange.

SetNewQuestionDefaults()

SetSubjectStuff()

SetCurriculumStuff()

SetKeyStageStuff()

SetExplanationStuff()

SetQuestionStuff()

SetDemoStuff()

' Act.

\_newQuestion.Process()

' Assert.

Assert.IsFalse(\_newQuestion.CanSubmit)

End Sub

<TestMethod()> Public Sub test\_process\_sets\_can\_submit\_flag\_to\_false\_with\_no\_demonstration()

' Arrange.

SetNewQuestionDefaults()

SetSubjectStuff()

SetCurriculumStuff()

SetKeyStageStuff()

SetExplanationStuff()

SetQuestionStuff()

SetAnswerStuff()

' Act.

\_newQuestion.Process()

' Assert.

Assert.IsFalse(\_newQuestion.CanSubmit)

End Sub

<TestMethod()> Public Sub test\_set\_lists\_from\_db\_sets\_subject\_list()

' Arrange.

SetNewQuestionDefaults()

Dim subject1 = New SubjectListType

subject1.ID = 1

subject1.SubjectDetail = "Bananas"

Dim subject2 = New SubjectListType

subject2.ID = 2

subject2.SubjectDetail = "Apples"

Dim list = New List(Of SubjectListType)

list.Add(subject1)

list.Add(subject2)

\_mockDb.Setup(Function(c) c.GetListOfSubjects()).Returns(list)

' Act.

\_newQuestion.SetListsFromDatabase()

' Assert.

Assert.AreEqual(list, \_newQuestion.SubjectType.SubjectList)

End Sub

<TestMethod()> Public Sub test\_set\_lists\_from\_db\_sets\_curriculum\_list()

' Arrange.

SetNewQuestionDefaults()

Dim curr1 = New CurriculumListType

curr1.ID = 1

curr1.CurriculumDetails = "Bananas"

Dim curr2 = New CurriculumListType

curr2.ID = 2

curr2.CurriculumDetails = "Apples"

Dim list = New List(Of CurriculumListType)

list.Add(curr1)

list.Add(curr2)

\_mockDb.Setup(Function(c) c.GetListOfCurriculum()).Returns(list)

' Act.

\_newQuestion.SetListsFromDatabase()

' Assert.

Assert.AreEqual(list, \_newQuestion.CurriculumType.CurriculumList)

End Sub

<TestMethod()> Public Sub test\_set\_lists\_from\_db\_sets\_key\_stage\_list()

' Arrange.

SetNewQuestionDefaults()

Dim key1 = New KeyStageListType

key1.ID = 1

key1.KeyStageDetail = "N/A"

Dim key2 = New KeyStageListType

key2.ID = 2

key2.KeyStageDetail = "KS1"

Dim list = New List(Of KeyStageListType)

list.Add(key1)

list.Add(key2)

\_mockDb.Setup(Function(c) c.GetListOfKeyStages()).Returns(list)

' Act.

\_newQuestion.SetListsFromDatabase()

' Assert.

Assert.AreEqual(list, \_newQuestion.KeyStageType.KeyStageList)

End Sub

<TestMethod()> Public Sub test\_set\_lists\_from\_db\_sets\_explanation\_list()

' Arrange.

SetNewQuestionDefaults()

Dim explanation1 = New ExplanationListType

explanation1.ID = 1

explanation1.ExplanationDetail = "Carrots"

Dim explanation2 = New ExplanationListType

explanation2.ID = 2

explanation2.ExplanationDetail = "Oh something"

Dim list = New List(Of ExplanationListType)

list.Add(explanation1)

list.Add(explanation2)

\_mockDb.Setup(Function(c) c.GetListOfExplanations()).Returns(list)

' Act.

\_newQuestion.SetListsFromDatabase()

' Assert.

Assert.AreEqual(list, \_newQuestion.ExplanationType.ExplanationList)

End Sub

<TestMethod()> Public Sub test\_update\_lists\_from\_db\_updates\_curriculum\_list()

' Arrange.

SetNewQuestionDefaults()

Dim curr1 = New CurriculumListType

curr1.ID = 1

curr1.CurriculumDetails = "Bananas"

Dim curr2 = New CurriculumListType

curr2.ID = 2

curr2.CurriculumDetails = "Apples"

Dim list = New List(Of CurriculumListType)

list.Add(curr1)

list.Add(curr2)

\_mockDb.Setup(Function(c) c.GetUpdatedListOfCurriculum(It.IsAny(Of Integer))).Returns(list)

' Act.

\_newQuestion.UpdateOtherLists()

' Assert.

Assert.AreEqual(list, \_newQuestion.CurriculumType.CurriculumList)

End Sub

<TestMethod()> Public Sub test\_update\_lists\_from\_db\_updates\_explanation\_list()

' Arrange.

SetNewQuestionDefaults()

Dim explanation1 = New ExplanationListType

explanation1.ID = 1

explanation1.ExplanationDetail = "Carrots"

Dim explanation2 = New ExplanationListType

explanation2.ID = 2

explanation2.ExplanationDetail = "Oh something"

Dim list = New List(Of ExplanationListType)

list.Add(explanation1)

list.Add(explanation2)

\_mockDb.Setup(Function(c) c.GetUpdatedListOfExplanations(It.IsAny(Of Integer))).Returns(list)

' Act.

\_newQuestion.UpdateOtherLists()

' Assert.

Assert.AreEqual(list, \_newQuestion.ExplanationType.ExplanationList)

End Sub

<TestMethod()> Public Sub test\_update\_explanation\_list\_from\_db\_updates\_explanation\_list()

' Arrange.

SetNewQuestionDefaults()

Dim explanation1 = New ExplanationListType

explanation1.ID = 1

explanation1.ExplanationDetail = "Carrots"

Dim explanation2 = New ExplanationListType

explanation2.ID = 2

explanation2.ExplanationDetail = "Oh something"

Dim list = New List(Of ExplanationListType)

list.Add(explanation1)

list.Add(explanation2)

\_mockDb.Setup(Function(c) c.GetUpdatedExplanationsByCurriculum(It.IsAny(Of Integer))).Returns(list)

' Act.

\_newQuestion.UpdateExplanation()

' Assert.

Assert.AreEqual(list, \_newQuestion.ExplanationType.ExplanationList)

End Sub

' Gets any changes and re-validates all fields.

' Sets parameters for stored procedures.

' Executes stored procedure to insert new question into the database.

' Checks it is inserted successfully.

<TestMethod()> Public Sub test\_insert\_question\_sets\_inserted\_successfully\_to\_true()

' Arrange.

SetNewQuestionDefaults()

SetSubjectStuff()

SetCurriculumStuff()

SetKeyStageStuff()

SetExplanationStuff()

SetQuestionStuff()

SetAnswerStuff()

SetDemoStuff()

\_newQuestion.Process()

\_mockDb.Setup(Function(s) s.InsertNewSubject(It.IsAny(Of String))).Returns(1)

\_mockDb.Setup(Function(c) c.InsertNewCurriculum(It.IsAny(Of Integer), It.IsAny(Of String))).Returns(2)

\_mockDb.Setup(Function(k) k.InsertNewKeyStage(It.IsAny(Of String))).Returns(3)

\_mockDb.Setup(Function(e) e.InsertNewExplanation(It.IsAny(Of Integer), It.IsAny(Of String), It.IsAny(Of String))).Returns(4)

\_mockDb.Setup(Function(q) q.InsertNewQuestion(It.IsAny(Of QuestionInsertType))).Returns(5)

\_mockDb.Setup(Function(d) d.InsertNewDemoStep(It.IsAny(Of DemonstrationStepType), It.IsAny(Of Integer))).Returns(True)

' Act.

\_newQuestion.InsertNewQuestion()

' Assert.

Assert.IsTrue(\_newQuestion.InsertedSuccessfully)

End Sub

<TestMethod()> Public Sub test\_insert\_question\_sets\_new\_question\_insert\_message\_to\_success()

' Arrange.

SetNewQuestionDefaults()

SetSubjectStuff()

SetCurriculumStuff()

SetKeyStageStuff()

SetExplanationStuff()

SetQuestionStuff()

SetAnswerStuff()

SetDemoStuff()

\_newQuestion.Process()

\_mockDb.Setup(Function(s) s.InsertNewSubject(It.IsAny(Of String))).Returns(1)

\_mockDb.Setup(Function(c) c.InsertNewCurriculum(It.IsAny(Of Integer), It.IsAny(Of String))).Returns(2)

\_mockDb.Setup(Function(k) k.InsertNewKeyStage(It.IsAny(Of String))).Returns(3)

\_mockDb.Setup(Function(e) e.InsertNewExplanation(It.IsAny(Of Integer), It.IsAny(Of String), It.IsAny(Of String))).Returns(4)

\_mockDb.Setup(Function(q) q.InsertNewQuestion(It.IsAny(Of QuestionInsertType))).Returns(5)

\_mockDb.Setup(Function(d) d.InsertNewDemoStep(It.IsAny(Of DemonstrationStepType), It.IsAny(Of Integer))).Returns(True)

' Act.

\_newQuestion.InsertNewQuestion()

' Assert.

Assert.AreEqual(CommonConstants.NewQuestionInsertedCorrectly, \_newQuestion.NewQuestionInsertMessage)

End Sub

<TestMethod()> Public Sub test\_insert\_question\_sets\_inserted\_successfully\_to\_false\_with\_no\_subject()

' Arrange.

SetNewQuestionDefaults()

'SetSubjectStuff()

SetCurriculumStuff()

SetKeyStageStuff()

SetExplanationStuff()

SetQuestionStuff()

SetAnswerStuff()

SetDemoStuff()

\_newQuestion.Process()

\_mockDb.Setup(Function(s) s.InsertNewSubject(It.IsAny(Of String))).Returns(0)

\_mockDb.Setup(Function(c) c.InsertNewCurriculum(It.IsAny(Of Integer), It.IsAny(Of String))).Returns(2)

\_mockDb.Setup(Function(k) k.InsertNewKeyStage(It.IsAny(Of String))).Returns(3)

\_mockDb.Setup(Function(e) e.InsertNewExplanation(It.IsAny(Of Integer), It.IsAny(Of String), It.IsAny(Of String))).Returns(4)

\_mockDb.Setup(Function(q) q.InsertNewQuestion(It.IsAny(Of QuestionInsertType))).Returns(5)

\_mockDb.Setup(Function(d) d.InsertNewDemoStep(It.IsAny(Of DemonstrationStepType), It.IsAny(Of Integer))).Returns(True)

' Act.

\_newQuestion.InsertNewQuestion()

' Assert.

Assert.IsFalse(\_newQuestion.InsertedSuccessfully)

End Sub

<TestMethod()> Public Sub test\_insert\_question\_sets\_new\_question\_insert\_message\_to\_failure\_with\_no\_subject()

' Arrange.

SetNewQuestionDefaults()

'SetSubjectStuff()

SetCurriculumStuff()

SetKeyStageStuff()

SetExplanationStuff()

SetQuestionStuff()

SetAnswerStuff()

SetDemoStuff()

\_newQuestion.Process()

\_mockDb.Setup(Function(s) s.InsertNewSubject(It.IsAny(Of String))).Returns(0)

\_mockDb.Setup(Function(c) c.InsertNewCurriculum(It.IsAny(Of Integer), It.IsAny(Of String))).Returns(2)

\_mockDb.Setup(Function(k) k.InsertNewKeyStage(It.IsAny(Of String))).Returns(3)

\_mockDb.Setup(Function(e) e.InsertNewExplanation(It.IsAny(Of Integer), It.IsAny(Of String), It.IsAny(Of String))).Returns(4)

\_mockDb.Setup(Function(q) q.InsertNewQuestion(It.IsAny(Of QuestionInsertType))).Returns(5)

\_mockDb.Setup(Function(d) d.InsertNewDemoStep(It.IsAny(Of DemonstrationStepType), It.IsAny(Of Integer))).Returns(True)

' Act.

\_newQuestion.InsertNewQuestion()

' Assert.

Assert.AreEqual(CommonConstants.InsertNewQuestionError, \_newQuestion.NewQuestionInsertMessage)

End Sub

<TestMethod()> Public Sub test\_insert\_question\_sets\_inserted\_successfully\_to\_false\_with\_no\_curriculum()

' Arrange.

SetNewQuestionDefaults()

SetSubjectStuff()

'SetCurriculumStuff()

SetKeyStageStuff()

SetExplanationStuff()

SetQuestionStuff()

SetAnswerStuff()

SetDemoStuff()

\_newQuestion.Process()

\_mockDb.Setup(Function(s) s.InsertNewSubject(It.IsAny(Of String))).Returns(1)

\_mockDb.Setup(Function(c) c.InsertNewCurriculum(It.IsAny(Of Integer), It.IsAny(Of String))).Returns(0)

\_mockDb.Setup(Function(k) k.InsertNewKeyStage(It.IsAny(Of String))).Returns(3)

\_mockDb.Setup(Function(e) e.InsertNewExplanation(It.IsAny(Of Integer), It.IsAny(Of String), It.IsAny(Of String))).Returns(4)

\_mockDb.Setup(Function(q) q.InsertNewQuestion(It.IsAny(Of QuestionInsertType))).Returns(5)

\_mockDb.Setup(Function(d) d.InsertNewDemoStep(It.IsAny(Of DemonstrationStepType), It.IsAny(Of Integer))).Returns(True)

' Act.

\_newQuestion.InsertNewQuestion()

' Assert.

Assert.IsFalse(\_newQuestion.InsertedSuccessfully)

End Sub

<TestMethod()> Public Sub test\_insert\_question\_sets\_new\_question\_insert\_message\_to\_failure\_with\_no\_curriculum()

' Arrange.

SetNewQuestionDefaults()

SetSubjectStuff()

'SetCurriculumStuff()

SetKeyStageStuff()

SetExplanationStuff()

SetQuestionStuff()

SetAnswerStuff()

SetDemoStuff()

\_newQuestion.Process()

\_mockDb.Setup(Function(s) s.InsertNewSubject(It.IsAny(Of String))).Returns(1)

\_mockDb.Setup(Function(c) c.InsertNewCurriculum(It.IsAny(Of Integer), It.IsAny(Of String))).Returns(0)

\_mockDb.Setup(Function(k) k.InsertNewKeyStage(It.IsAny(Of String))).Returns(3)

\_mockDb.Setup(Function(e) e.InsertNewExplanation(It.IsAny(Of Integer), It.IsAny(Of String), It.IsAny(Of String))).Returns(4)

\_mockDb.Setup(Function(q) q.InsertNewQuestion(It.IsAny(Of QuestionInsertType))).Returns(5)

\_mockDb.Setup(Function(d) d.InsertNewDemoStep(It.IsAny(Of DemonstrationStepType), It.IsAny(Of Integer))).Returns(True)

' Act.

\_newQuestion.InsertNewQuestion()

' Assert.

Assert.AreEqual(CommonConstants.InsertNewQuestionError, \_newQuestion.NewQuestionInsertMessage)

End Sub

<TestMethod()> Public Sub test\_insert\_question\_sets\_inserted\_successfully\_to\_false\_with\_no\_key\_stage()

' Arrange.

SetNewQuestionDefaults()

SetSubjectStuff()

SetCurriculumStuff()

'SetKeyStageStuff()

SetExplanationStuff()

SetQuestionStuff()

SetAnswerStuff()

SetDemoStuff()

\_newQuestion.Process()

\_mockDb.Setup(Function(s) s.InsertNewSubject(It.IsAny(Of String))).Returns(1)

\_mockDb.Setup(Function(c) c.InsertNewCurriculum(It.IsAny(Of Integer), It.IsAny(Of String))).Returns(2)

\_mockDb.Setup(Function(k) k.InsertNewKeyStage(It.IsAny(Of String))).Returns(0)

\_mockDb.Setup(Function(e) e.InsertNewExplanation(It.IsAny(Of Integer), It.IsAny(Of String), It.IsAny(Of String))).Returns(4)

\_mockDb.Setup(Function(q) q.InsertNewQuestion(It.IsAny(Of QuestionInsertType))).Returns(5)

\_mockDb.Setup(Function(d) d.InsertNewDemoStep(It.IsAny(Of DemonstrationStepType), It.IsAny(Of Integer))).Returns(True)

' Act.

\_newQuestion.InsertNewQuestion()

' Assert.

Assert.IsFalse(\_newQuestion.InsertedSuccessfully)

End Sub

<TestMethod()> Public Sub test\_insert\_question\_sets\_new\_question\_insert\_message\_to\_failure\_with\_no\_key\_stage()

' Arrange.

SetNewQuestionDefaults()

SetSubjectStuff()

SetCurriculumStuff()

'SetKeyStageStuff()

SetExplanationStuff()

SetQuestionStuff()

SetAnswerStuff()

SetDemoStuff()

\_newQuestion.Process()

\_mockDb.Setup(Function(s) s.InsertNewSubject(It.IsAny(Of String))).Returns(1)

\_mockDb.Setup(Function(c) c.InsertNewCurriculum(It.IsAny(Of Integer), It.IsAny(Of String))).Returns(2)

\_mockDb.Setup(Function(k) k.InsertNewKeyStage(It.IsAny(Of String))).Returns(0)

\_mockDb.Setup(Function(e) e.InsertNewExplanation(It.IsAny(Of Integer), It.IsAny(Of String), It.IsAny(Of String))).Returns(4)

\_mockDb.Setup(Function(q) q.InsertNewQuestion(It.IsAny(Of QuestionInsertType))).Returns(5)

\_mockDb.Setup(Function(d) d.InsertNewDemoStep(It.IsAny(Of DemonstrationStepType), It.IsAny(Of Integer))).Returns(True)

' Act.

\_newQuestion.InsertNewQuestion()

' Assert.

Assert.AreEqual(CommonConstants.InsertNewQuestionError, \_newQuestion.NewQuestionInsertMessage)

End Sub

<TestMethod()> Public Sub test\_insert\_question\_sets\_inserted\_successfully\_to\_false\_with\_no\_explanation()

' Arrange.

SetNewQuestionDefaults()

SetSubjectStuff()

SetCurriculumStuff()

SetKeyStageStuff()

'SetExplanationStuff()

SetQuestionStuff()

SetAnswerStuff()

SetDemoStuff()

\_newQuestion.Process()

\_mockDb.Setup(Function(s) s.InsertNewSubject(It.IsAny(Of String))).Returns(1)

\_mockDb.Setup(Function(c) c.InsertNewCurriculum(It.IsAny(Of Integer), It.IsAny(Of String))).Returns(2)

\_mockDb.Setup(Function(k) k.InsertNewKeyStage(It.IsAny(Of String))).Returns(3)

\_mockDb.Setup(Function(e) e.InsertNewExplanation(It.IsAny(Of Integer), It.IsAny(Of String), It.IsAny(Of String))).Returns(0)

\_mockDb.Setup(Function(q) q.InsertNewQuestion(It.IsAny(Of QuestionInsertType))).Returns(5)

\_mockDb.Setup(Function(d) d.InsertNewDemoStep(It.IsAny(Of DemonstrationStepType), It.IsAny(Of Integer))).Returns(True)

' Act.

\_newQuestion.InsertNewQuestion()

' Assert.

Assert.IsFalse(\_newQuestion.InsertedSuccessfully)

End Sub

<TestMethod()> Public Sub test\_insert\_question\_sets\_new\_question\_insert\_message\_to\_failure\_with\_no\_explanation()

' Arrange.

SetNewQuestionDefaults()

SetSubjectStuff()

SetCurriculumStuff()

SetKeyStageStuff()

'SetExplanationStuff()

SetQuestionStuff()

SetAnswerStuff()

SetDemoStuff()

\_newQuestion.Process()

\_mockDb.Setup(Function(s) s.InsertNewSubject(It.IsAny(Of String))).Returns(1)

\_mockDb.Setup(Function(c) c.InsertNewCurriculum(It.IsAny(Of Integer), It.IsAny(Of String))).Returns(2)

\_mockDb.Setup(Function(k) k.InsertNewKeyStage(It.IsAny(Of String))).Returns(3)

\_mockDb.Setup(Function(e) e.InsertNewExplanation(It.IsAny(Of Integer), It.IsAny(Of String), It.IsAny(Of String))).Returns(0)

\_mockDb.Setup(Function(q) q.InsertNewQuestion(It.IsAny(Of QuestionInsertType))).Returns(5)

\_mockDb.Setup(Function(d) d.InsertNewDemoStep(It.IsAny(Of DemonstrationStepType), It.IsAny(Of Integer))).Returns(True)

' Act.

\_newQuestion.InsertNewQuestion()

' Assert.

Assert.AreEqual(CommonConstants.InsertNewQuestionError, \_newQuestion.NewQuestionInsertMessage)

End Sub

<TestMethod()> Public Sub test\_insert\_question\_sets\_inserted\_successfully\_to\_false\_with\_no\_question()

' Arrange.

SetNewQuestionDefaults()

SetSubjectStuff()

SetCurriculumStuff()

SetKeyStageStuff()

SetExplanationStuff()

'SetQuestionStuff()

SetAnswerStuff()

SetDemoStuff()

\_newQuestion.Process()

\_mockDb.Setup(Function(s) s.InsertNewSubject(It.IsAny(Of String))).Returns(1)

\_mockDb.Setup(Function(c) c.InsertNewCurriculum(It.IsAny(Of Integer), It.IsAny(Of String))).Returns(2)

\_mockDb.Setup(Function(k) k.InsertNewKeyStage(It.IsAny(Of String))).Returns(3)

\_mockDb.Setup(Function(e) e.InsertNewExplanation(It.IsAny(Of Integer), It.IsAny(Of String), It.IsAny(Of String))).Returns(4)

\_mockDb.Setup(Function(q) q.InsertNewQuestion(It.IsAny(Of QuestionInsertType))).Returns(0)

\_mockDb.Setup(Function(d) d.InsertNewDemoStep(It.IsAny(Of DemonstrationStepType), It.IsAny(Of Integer))).Returns(True)

' Act.

\_newQuestion.InsertNewQuestion()

' Assert.

Assert.IsFalse(\_newQuestion.InsertedSuccessfully)

End Sub

<TestMethod()> Public Sub test\_insert\_question\_sets\_new\_question\_insert\_message\_to\_failure\_with\_no\_question()

' Arrange.

SetNewQuestionDefaults()

SetSubjectStuff()

SetCurriculumStuff()

SetKeyStageStuff()

SetExplanationStuff()

'SetQuestionStuff()

SetAnswerStuff()

SetDemoStuff()

\_newQuestion.Process()

\_mockDb.Setup(Function(s) s.InsertNewSubject(It.IsAny(Of String))).Returns(1)

\_mockDb.Setup(Function(c) c.InsertNewCurriculum(It.IsAny(Of Integer), It.IsAny(Of String))).Returns(2)

\_mockDb.Setup(Function(k) k.InsertNewKeyStage(It.IsAny(Of String))).Returns(3)

\_mockDb.Setup(Function(e) e.InsertNewExplanation(It.IsAny(Of Integer), It.IsAny(Of String), It.IsAny(Of String))).Returns(4)

\_mockDb.Setup(Function(q) q.InsertNewQuestion(It.IsAny(Of QuestionInsertType))).Returns(0)

\_mockDb.Setup(Function(d) d.InsertNewDemoStep(It.IsAny(Of DemonstrationStepType), It.IsAny(Of Integer))).Returns(True)

' Act.

\_newQuestion.InsertNewQuestion()

' Assert.

Assert.AreEqual(CommonConstants.InsertNewQuestionError, \_newQuestion.NewQuestionInsertMessage)

End Sub

<TestMethod()> Public Sub test\_insert\_question\_sets\_inserted\_successfully\_to\_false\_with\_no\_answer()

' Arrange.

SetNewQuestionDefaults()

SetSubjectStuff()

SetCurriculumStuff()

SetKeyStageStuff()

SetExplanationStuff()

SetQuestionStuff()

'SetAnswerStuff()

SetDemoStuff()

\_newQuestion.Process()

\_mockDb.Setup(Function(s) s.InsertNewSubject(It.IsAny(Of String))).Returns(1)

\_mockDb.Setup(Function(c) c.InsertNewCurriculum(It.IsAny(Of Integer), It.IsAny(Of String))).Returns(2)

\_mockDb.Setup(Function(k) k.InsertNewKeyStage(It.IsAny(Of String))).Returns(3)

\_mockDb.Setup(Function(e) e.InsertNewExplanation(It.IsAny(Of Integer), It.IsAny(Of String), It.IsAny(Of String))).Returns(4)

\_mockDb.Setup(Function(q) q.InsertNewQuestion(It.IsAny(Of QuestionInsertType))).Returns(0)

\_mockDb.Setup(Function(d) d.InsertNewDemoStep(It.IsAny(Of DemonstrationStepType), It.IsAny(Of Integer))).Returns(True)

' Act.

\_newQuestion.InsertNewQuestion()

' Assert.

Assert.IsFalse(\_newQuestion.InsertedSuccessfully)

End Sub

<TestMethod()> Public Sub test\_insert\_question\_sets\_new\_question\_insert\_message\_to\_failure\_with\_no\_answer()

' Arrange.

SetNewQuestionDefaults()

SetSubjectStuff()

SetCurriculumStuff()

SetKeyStageStuff()

SetExplanationStuff()

SetQuestionStuff()

'SetAnswerStuff()

SetDemoStuff()

\_newQuestion.Process()

\_mockDb.Setup(Function(s) s.InsertNewSubject(It.IsAny(Of String))).Returns(1)

\_mockDb.Setup(Function(c) c.InsertNewCurriculum(It.IsAny(Of Integer), It.IsAny(Of String))).Returns(2)

\_mockDb.Setup(Function(k) k.InsertNewKeyStage(It.IsAny(Of String))).Returns(3)

\_mockDb.Setup(Function(e) e.InsertNewExplanation(It.IsAny(Of Integer), It.IsAny(Of String), It.IsAny(Of String))).Returns(4)

\_mockDb.Setup(Function(q) q.InsertNewQuestion(It.IsAny(Of QuestionInsertType))).Returns(0)

\_mockDb.Setup(Function(d) d.InsertNewDemoStep(It.IsAny(Of DemonstrationStepType), It.IsAny(Of Integer))).Returns(True)

' Act.

\_newQuestion.InsertNewQuestion()

' Assert.

Assert.AreEqual(CommonConstants.InsertNewQuestionError, \_newQuestion.NewQuestionInsertMessage)

End Sub

<TestMethod()> Public Sub test\_insert\_question\_sets\_inserted\_successfully\_to\_false\_with\_no\_demo\_steps()

' Arrange.

SetNewQuestionDefaults()

SetSubjectStuff()

SetCurriculumStuff()

SetKeyStageStuff()

SetExplanationStuff()

SetQuestionStuff()

SetAnswerStuff()

'SetDemoStuff()

\_newQuestion.Process()

\_mockDb.Setup(Function(s) s.InsertNewSubject(It.IsAny(Of String))).Returns(1)

\_mockDb.Setup(Function(c) c.InsertNewCurriculum(It.IsAny(Of Integer), It.IsAny(Of String))).Returns(2)

\_mockDb.Setup(Function(k) k.InsertNewKeyStage(It.IsAny(Of String))).Returns(3)

\_mockDb.Setup(Function(e) e.InsertNewExplanation(It.IsAny(Of Integer), It.IsAny(Of String), It.IsAny(Of String))).Returns(4)

\_mockDb.Setup(Function(q) q.InsertNewQuestion(It.IsAny(Of QuestionInsertType))).Returns(5)

\_mockDb.Setup(Function(d) d.InsertNewDemoStep(It.IsAny(Of DemonstrationStepType), It.IsAny(Of Integer))).Returns(False)

' Act.

\_newQuestion.InsertNewQuestion()

' Assert.

Assert.IsFalse(\_newQuestion.InsertedSuccessfully)

End Sub

<TestMethod()> Public Sub test\_insert\_question\_sets\_new\_question\_insert\_message\_to\_failure\_with\_no\_demo\_step()

' Arrange.

SetNewQuestionDefaults()

SetSubjectStuff()

SetCurriculumStuff()

SetKeyStageStuff()

SetExplanationStuff()

SetQuestionStuff()

SetAnswerStuff()

'SetDemoStuff()

\_newQuestion.Process()

\_mockDb.Setup(Function(s) s.InsertNewSubject(It.IsAny(Of String))).Returns(1)

\_mockDb.Setup(Function(c) c.InsertNewCurriculum(It.IsAny(Of Integer), It.IsAny(Of String))).Returns(2)

\_mockDb.Setup(Function(k) k.InsertNewKeyStage(It.IsAny(Of String))).Returns(3)

\_mockDb.Setup(Function(e) e.InsertNewExplanation(It.IsAny(Of Integer), It.IsAny(Of String), It.IsAny(Of String))).Returns(4)

\_mockDb.Setup(Function(q) q.InsertNewQuestion(It.IsAny(Of QuestionInsertType))).Returns(5)

\_mockDb.Setup(Function(d) d.InsertNewDemoStep(It.IsAny(Of DemonstrationStepType), It.IsAny(Of Integer))).Returns(False)

' Act.

\_newQuestion.InsertNewQuestion()

' Assert.

Assert.AreEqual(CommonConstants.InsertNewDemoStepError, \_newQuestion.NewQuestionInsertMessage)

End Sub

End Class

### NewUserInputTests.vb

Imports Microsoft.VisualStudio.TestTools.UnitTesting

Imports ExampleProg.Constants

Imports ExampleProg.InputClasses

Imports ExampleProg.Interfaces

Imports Moq

''' <summary>

''' A set of tests for the New User Input class.

''' </summary>

''' <remarks>Tests.</remarks>

<TestClass()> Public Class NewUserInputTests

<TestMethod()> Public Sub test\_new\_user\_sets\_user\_name\_valid\_state\_to\_true()

' Arrange.

Dim userInput = New NewUserInput("me", "secret", "Pooh", "S", "Bear")

' Act.

userInput.SetValidStates()

' Assert.

Assert.IsTrue(userInput.IsUserNameValid)

End Sub

<TestMethod()> Public Sub test\_new\_user\_sets\_user\_name\_valid\_state\_to\_false()

' Arrange.

Dim userInput = New NewUserInput(String.Empty, "secret", "Pooh", "S", "Bear")

' Act.

userInput.SetValidStates()

' Assert.

Assert.IsFalse(userInput.IsUserNameValid)

End Sub

<TestMethod()> Public Sub test\_new\_user\_sets\_password\_valid\_state\_to\_true()

' Arrange.

Dim userInput = New NewUserInput("me", "secret", "Pooh", "S", "Bear")

' Act.

userInput.SetValidStates()

' Assert.

Assert.IsTrue(userInput.IsPasswordValid)

End Sub

<TestMethod()> Public Sub test\_new\_user\_sets\_password\_valid\_state\_to\_false()

' Arrange.

Dim userInput = New NewUserInput("me", String.Empty, "Pooh", "S", "Bear")

' Act.

userInput.SetValidStates()

' Assert.

Assert.IsFalse(userInput.IsPasswordValid)

End Sub

<TestMethod()> Public Sub test\_new\_user\_sets\_first\_name\_valid\_state\_to\_true()

' Arrange.

Dim userInput = New NewUserInput("me", "secret", "Pooh", "S", "Bear")

' Act.

userInput.SetValidStates()

' Assert.

Assert.IsTrue(userInput.IsFirstNameValid)

End Sub

<TestMethod()> Public Sub test\_new\_user\_sets\_first\_name\_valid\_state\_to\_false()

' Arrange.

Dim userInput = New NewUserInput("me", "secret", String.Empty, "S", "Bear")

' Act.

userInput.SetValidStates()

' Assert.

Assert.IsFalse(userInput.IsFirstNameValid)

End Sub

<TestMethod()> Public Sub test\_new\_user\_sets\_middle\_name\_valid\_state\_to\_true()

' Arrange.

Dim userInput = New NewUserInput("me", "secret", "Pooh", "S", "Bear")

' Act.

userInput.SetValidStates()

' Assert.

Assert.IsTrue(userInput.IsMiddleNameValid)

End Sub

<TestMethod()> Public Sub test\_new\_user\_sets\_middle\_name\_valid\_state\_to\_true\_with\_null\_value()

' Arrange.

Dim userInput = New NewUserInput("me", "secret", "Pooh", String.Empty, "Bear")

' Act.

userInput.SetValidStates()

' Assert.

Assert.IsTrue(userInput.IsMiddleNameValid)

End Sub

<TestMethod()> Public Sub test\_new\_user\_sets\_last\_name\_valid\_state\_to\_true()

' Arrange.

Dim userInput = New NewUserInput("me", "secret", "Pooh", "S", "Bear")

' Act.

userInput.SetValidStates()

' Assert.

Assert.IsTrue(userInput.IsLastNameValid)

End Sub

<TestMethod()> Public Sub test\_new\_user\_sets\_last\_name\_valid\_state\_to\_false()

' Arrange.

Dim userInput = New NewUserInput("me", "secret", "Pooh", "S", String.Empty)

' Act.

userInput.SetValidStates()

' Assert.

Assert.IsFalse(userInput.IsLastNameValid)

End Sub

<TestMethod()> Public Sub test\_new\_user\_sets\_array\_of\_valid\_states\_for\_user\_name\_to\_true()

' Arrange.

Dim userInput = New NewUserInput("me", "secret", "Pooh", "S", "Bear")

userInput.SetValidStates()

' Act.

userInput.SetArrayOfValidStates()

' Assert.

Assert.IsTrue(userInput.ArrayOfValidity(0))

End Sub

<TestMethod()> Public Sub test\_new\_user\_sets\_array\_of\_valid\_states\_for\_user\_name\_already\_used\_to\_false()

' Arrange.

Dim userInput = New NewUserInput("me", "secret", "Pooh", "S", "Bear")

userInput.SetValidStates()

' Act.

userInput.SetArrayOfValidStates()

' Assert.

Assert.IsFalse(userInput.ArrayOfValidity(1))

End Sub

<TestMethod()> Public Sub test\_new\_user\_sets\_array\_of\_valid\_states\_for\_password\_to\_true()

' Arrange.

Dim userInput = New NewUserInput("me", "secret", "Pooh", "S", "Bear")

userInput.SetValidStates()

' Act.

userInput.SetArrayOfValidStates()

' Assert.

Assert.IsTrue(userInput.ArrayOfValidity(2))

End Sub

<TestMethod()> Public Sub test\_new\_user\_sets\_array\_of\_valid\_states\_for\_first\_name\_to\_true()

' Arrange.

Dim userInput = New NewUserInput("me", "secret", "Pooh", "S", "Bear")

userInput.SetValidStates()

' Act.

userInput.SetArrayOfValidStates()

' Assert.

Assert.IsTrue(userInput.ArrayOfValidity(3))

End Sub

<TestMethod()> Public Sub test\_new\_user\_sets\_array\_of\_valid\_states\_for\_middle\_name\_to\_true()

' Arrange.

Dim userInput = New NewUserInput("me", "secret", "Pooh", "S", "Bear")

userInput.SetValidStates()

' Act.

userInput.SetArrayOfValidStates()

' Assert.

Assert.IsTrue(userInput.ArrayOfValidity(4))

End Sub

<TestMethod()> Public Sub test\_new\_user\_sets\_array\_of\_valid\_states\_for\_last\_name\_to\_true()

' Arrange.

Dim userInput = New NewUserInput("me", "secret", "Pooh", "S", "Bear")

userInput.SetValidStates()

' Act.

userInput.SetArrayOfValidStates()

' Assert.

Assert.IsTrue(userInput.ArrayOfValidity(5))

End Sub

<TestMethod()> Public Sub test\_new\_user\_sets\_invalid\_message\_for\_user\_name()

' Arrange.

Dim userInput = New NewUserInput("me", "secret", "Pooh", "S", "Bear")

userInput.SetValidStates()

userInput.IsUserAvailable = True

' Act.

userInput.SetInvalidMessages()

' Assert.

Assert.AreEqual(CommonConstants.UserIsAvailable, userInput.UserNameInvalidMessage)

End Sub

<TestMethod()> Public Sub test\_new\_user\_sets\_invalid\_message\_for\_user\_name\_when\_null()

' Arrange.

Dim userInput = New NewUserInput(String.Empty, "secret", "Pooh", "S", "Bear")

userInput.SetValidStates()

' Act.

userInput.SetInvalidMessages()

' Assert.

Assert.AreEqual(CommonConstants.NotAValidInput, userInput.UserNameInvalidMessage)

End Sub

<TestMethod()> Public Sub test\_new\_user\_sets\_invalid\_message\_for\_user\_name\_when\_already\_used()

' Arrange.

Dim userInput = New NewUserInput("me", "secret", "Pooh", "S", "Bear")

userInput.SetValidStates()

userInput.IsUserAvailable = False

' Act.

userInput.SetInvalidMessages()

' Assert.

Assert.AreEqual(CommonConstants.UserAlreadyExists, userInput.UserNameInvalidMessage)

End Sub

<TestMethod()> Public Sub test\_new\_user\_sets\_invalid\_message\_for\_password()

' Arrange.

Dim userInput = New NewUserInput("me", "secret", "Pooh", "S", "Bear")

userInput.SetValidStates()

' Act.

userInput.SetInvalidMessages()

' Assert.

Assert.AreEqual(String.Empty, userInput.PasswordInvalidMessage)

End Sub

<TestMethod()> Public Sub test\_new\_user\_sets\_invalid\_message\_for\_password\_when\_null()

' Arrange.

Dim userInput = New NewUserInput("me", String.Empty, "Pooh", "S", "Bear")

userInput.SetValidStates()

' Act.

userInput.SetInvalidMessages()

' Assert.

Assert.AreEqual(CommonConstants.NotAValidInput, userInput.PasswordInvalidMessage)

End Sub

<TestMethod()> Public Sub test\_new\_user\_sets\_invalid\_message\_for\_first\_name()

' Arrange.

Dim userInput = New NewUserInput("me", "secret", "Pooh", "S", "Bear")

userInput.SetValidStates()

' Act.

userInput.SetInvalidMessages()

' Assert.

Assert.AreEqual(String.Empty, userInput.FirstNameInvalidMessage)

End Sub

<TestMethod()> Public Sub test\_new\_user\_sets\_invalid\_message\_for\_first\_name\_when\_null()

' Arrange.

Dim userInput = New NewUserInput("me", "secret", String.Empty, "S", "Bear")

userInput.SetValidStates()

' Act.

userInput.SetInvalidMessages()

' Assert.

Assert.AreEqual(CommonConstants.NotAValidInput, userInput.FirstNameInvalidMessage)

End Sub

<TestMethod()> Public Sub test\_new\_user\_sets\_invalid\_message\_for\_middle\_name()

' Arrange.

Dim userInput = New NewUserInput("me", "secret", "Pooh", "S", "Bear")

userInput.SetValidStates()

' Act.

userInput.SetInvalidMessages()

' Assert.

Assert.AreEqual(String.Empty, userInput.MiddleNameInvalidmessage)

End Sub

<TestMethod()> Public Sub test\_new\_user\_sets\_invalid\_message\_for\_first\_middle\_when\_null()

' Arrange.

Dim userInput = New NewUserInput("me", "secret", "Pooh", String.Empty, "Bear")

userInput.SetValidStates()

' Act.

userInput.SetInvalidMessages()

' Assert.

Assert.AreEqual(String.Empty, userInput.MiddleNameInvalidmessage)

End Sub

<TestMethod()> Public Sub test\_new\_user\_sets\_invalid\_message\_for\_last\_name()

' Arrange.

Dim userInput = New NewUserInput("me", "secret", "Pooh", "S", "Bear")

userInput.SetValidStates()

' Act.

userInput.SetInvalidMessages()

' Assert.

Assert.AreEqual(String.Empty, userInput.LastNameInvalidMessage)

End Sub

<TestMethod()> Public Sub test\_new\_user\_sets\_invalid\_message\_for\_last\_name\_when\_null()

' Arrange.

Dim userInput = New NewUserInput("me", "secret", "Pooh", "S", String.Empty)

userInput.SetValidStates()

' Act.

userInput.SetInvalidMessages()

' Assert.

Assert.AreEqual(CommonConstants.NotAValidInput, userInput.LastNameInvalidMessage)

End Sub

<TestMethod()> Public Sub test\_is\_there\_false\_returns\_true()

' Arrange.

Const result As Boolean = True

Dim dbConnector = New Mock(Of IDbConnector)

dbConnector.Setup(Function(c) c.IsUserNameAvailable(It.IsAny(Of String))).Returns(result)

Dim testUser = New NewUserInput("pooh", "secret", "Pooh", "M", "Bear")

testUser.SetValidStates()

testUser.SetUserNameIsUsed(dbConnector.Object)

' Act.

testUser.SetArrayOfValidStates()

' Assert.

Assert.IsTrue(testUser.IsThereAFalseInTheInput())

End Sub

<TestMethod()> Public Sub test\_is\_user\_available()

' Arrange.

Const result As Boolean = True

Dim dbConnector = New Mock(Of IDbConnector)

dbConnector.Setup(Function(c) c.IsUserNameAvailable(It.IsAny(Of String))).Returns(result)

Dim testUser = New NewUserInput("pooh", "secret", "Pooh", "M", "Bear")

' Act.

testUser.SetUserNameIsUsed(dbConnector.Object)

' Assert.

Assert.IsTrue(testUser.IsUserAvailable)

End Sub

<TestMethod()> Public Sub test\_is\_user\_available\_returns\_false()

' Arrange.

Const result As Boolean = False

Dim dbConnector = New Mock(Of IDbConnector)

dbConnector.Setup(Function(c) c.IsUserNameAvailable(It.IsAny(Of String))).Returns(result)

Dim testUser = New NewUserInput("pooh", "secret", "Pooh", "M", "Bear")

' Act.

testUser.SetUserNameIsUsed(dbConnector.Object)

' Assert.

Assert.IsFalse(testUser.IsUserAvailable)

End Sub

End Class

### UserClassTests.vb

Imports Microsoft.VisualStudio.TestTools.UnitTesting

Imports Moq

Imports ExampleProg.Classes

Imports ExampleProg.Interfaces

''' <summary>

''' A set of tests for the User class.

''' </summary>

''' <remarks>Tests.</remarks>

<TestClass()> Public Class UserClassTests

<TestMethod()> Public Sub create\_new\_user\_returns\_true()

' Arrange.

Dim dbConnector = New Mock(Of IDbConnector)

Const expected As Boolean = True

dbConnector.Setup(Function(c) c.CreateNewUser(It.IsAny(Of UserClass))).Returns(expected)

Dim user As IUserClass

user = New UserClass

user.UserName = "pooh"

user.Password = "1234"

user.FirstName = "Pooh"

user.LastName = "Bear"

' Act.

Dim result As Boolean

result = user.CreateNewUser(dbConnector.Object)

' Assert.

Assert.IsTrue(result)

End Sub

<TestMethod()> Public Sub create\_new\_user\_returns\_false()

' Arrange.

Dim dbConnector = New Mock(Of IDbConnector)

Const expected As Boolean = False

dbConnector.Setup(Function(c) c.CreateNewUser(It.IsAny(Of UserClass))).Returns(expected)

Dim user As IUserClass

user = New UserClass

user.UserName = "pooh"

user.Password = "1234"

user.FirstName = "Pooh"

user.LastName = "Bear"

' Act.

Dim result As Boolean

result = user.CreateNewUser(dbConnector.Object)

' Assert.

Assert.IsFalse(result)

End Sub

<TestMethod()> Public Sub log\_in\_returns\_true()

' Arrange.

Dim dbConnector = New Mock(Of IDbConnector)

Const id As Integer = 1

Dim user = New UserClass

user.FirstName = "Pooh"

user.LastName = "Bear"

user.UserId = 1

user.UserName = "poohB"

user.Password = "ohbother"

dbConnector.Setup(Function(c) c.GetUserIdOfValidUser(It.IsAny(Of String), It.IsAny(Of String))).Returns(id)

dbConnector.Setup(Function(d) d.GetUserDetails(It.IsAny(Of UserClass))).Returns(user)

Dim userInput = New UserClass

userInput.UserName = "poohB"

userInput.Password = "ohbother"

' Act.

Dim result As Boolean

result = userInput.Login(dbConnector.Object)

' Assert.

Assert.IsTrue(result)

End Sub

<TestMethod()> Public Sub log\_in\_returns\_false()

' Arrange.

Dim dbConnector = New Mock(Of IDbConnector)

Dim user = New UserClass

dbConnector.Setup(Function(d) d.GetUserDetails(It.IsAny(Of UserClass))).Returns(user)

Dim userInput = New UserClass

userInput.UserName = "poohB"

userInput.Password = "ohbother"

' Act.

Dim result As Boolean

result = userInput.Login(dbConnector.Object)

' Assert.

Assert.IsFalse(result)

End Sub

<TestMethod()> Public Sub log\_in\_sets\_id\_correctly()

' Arrange.

Dim dbConnector = New Mock(Of IDbConnector)

Const id As Integer = 1

Dim user = New UserClass

user.FirstName = "Pooh"

user.LastName = "Bear"

user.UserId = 1

user.UserName = "poohB"

user.Password = "ohbother"

dbConnector.Setup(Function(c) c.GetUserIdOfValidUser(It.IsAny(Of String), It.IsAny(Of String))).Returns(id)

dbConnector.Setup(Function(d) d.GetUserDetails(It.IsAny(Of UserClass))).Returns(user)

Dim userInput = New UserClass

userInput.UserName = "poohB"

userInput.Password = "ohbother"

' Act.

userInput.Login(dbConnector.Object)

' Assert.

Assert.AreEqual(1, userInput.UserId)

End Sub

<TestMethod()> Public Sub does\_user\_name\_exist\_returns\_true()

' Arrange.

Const result As Boolean = True

Dim dbConnector = New Mock(Of IDbConnector)

dbConnector.Setup(Function(c) c.DoesUserNameAlreadyExists(It.IsAny(Of String))).Returns(result)

Dim user = New UserClass

user.UserName = "tiggr"

' Act.

Dim actual = user.DoesUserNameAlreadyExists(dbConnector.Object)

' Assert.

Assert.IsTrue(actual)

End Sub

<TestMethod()> Public Sub does\_user\_name\_exist\_returns\_false()

' Arrange.

Const result As Boolean = False

Dim dbConnector = New Mock(Of IDbConnector)

dbConnector.Setup(Function(c) c.DoesUserNameAlreadyExists(It.IsAny(Of String))).Returns(result)

Dim user = New UserClass

user.UserName = "tiggr"

' Act.

Dim actual = user.DoesUserNameAlreadyExists(dbConnector.Object)

' Assert.

Assert.IsFalse(actual)

End Sub

<TestMethod()> Public Sub test\_is\_user\_name\_available\_returns\_true()

' Arrange.

Const result As Boolean = True

Dim dbConnector = New Mock(Of IDbConnector)

dbConnector.Setup(Function(c) c.IsUserNameAvailable(It.IsAny(Of String))).Returns(result)

Dim testUser = New UserClass

testUser.UserName = "tiggr"

' Act.

Dim actual = testUser.IsUserNameInDatabase(dbConnector.Object)

' Assert.

Assert.IsTrue(actual)

End Sub

<TestMethod()> Public Sub test\_is\_user\_name\_available\_returns\_false()

' Arrange.

Const result As Boolean = False

Dim dbConnector = New Mock(Of IDbConnector)

dbConnector.Setup(Function(c) c.IsUserNameAvailable(It.IsAny(Of String))).Returns(result)

Dim testUser = New UserClass

testUser.UserName = "tiggr"

' Act.

Dim actual = testUser.IsUserNameInDatabase(dbConnector.Object)

' Assert.

Assert.IsFalse(actual)

End Sub

<TestMethod()> Public Sub test\_log\_out\_sets\_user\_id\_to\_zero()

' Arrange.

Dim testUser = New UserClass

testUser.UserId = 4

testUser.FirstName = "Pooh"

testUser.LastName = "Bear"

testUser.UserName = "poohB"

testUser.Password = "secret"

' Act.

testUser.Logout()

' Assert.

Assert.AreEqual(0, testUser.UserId)

End Sub

<TestMethod()> Public Sub test\_log\_out\_sets\_user\_name\_to\_empty\_string()

' Arrange.

Dim testUser = New UserClass

testUser.UserId = 4

testUser.FirstName = "Pooh"

testUser.LastName = "Bear"

testUser.UserName = "poohB"

testUser.Password = "secret"

' Act.

testUser.Logout()

' Assert.

Assert.AreEqual(String.Empty, testUser.UserName)

End Sub

<TestMethod()> Public Sub test\_log\_out\_sets\_first\_name\_to\_empty\_string()

' Arrange.

Dim testUser = New UserClass

testUser.UserId = 4

testUser.FirstName = "Pooh"

testUser.LastName = "Bear"

testUser.UserName = "poohB"

testUser.Password = "secret"

' Act.

testUser.Logout()

' Assert.

Assert.AreEqual(String.Empty, testUser.FirstName)

End Sub

<TestMethod()> Public Sub test\_log\_out\_sets\_last\_name\_to\_empty\_string()

' Arrange.

Dim testUser = New UserClass

testUser.UserId = 4

testUser.FirstName = "Pooh"

testUser.LastName = "Bear"

testUser.UserName = "poohB"

testUser.Password = "secret"

' Act.

testUser.Logout()

' Assert.

Assert.AreEqual(String.Empty, testUser.LastName)

End Sub

<TestMethod()> Public Sub test\_log\_out\_sets\_password\_to\_empty\_string()

' Arrange.

Dim testUser = New UserClass

testUser.UserId = 4

testUser.FirstName = "Pooh"

testUser.LastName = "Bear"

testUser.UserName = "poohB"

testUser.Password = "secret"

' Act.

testUser.Logout()

' Assert.

Assert.AreEqual(String.Empty, testUser.Password)

End Sub

End Class