TUFCv3 *(XWM Xamarin)*

TODO

Contents

[Introduction 4](#_Toc117087223)

[Create a new GitHub branch 5](#_Toc117087224)

[Create TUFCv3 6](#_Toc117087225)

[Update Software 7](#_Toc117087226)

[Run the app 9](#_Toc117087227)

[Install MySqlConnector 10](#_Toc117087228)

[Temporary layout for MainPage() 13](#_Toc117087229)

[Test commands for SQL 13](#_Toc117087230)

[Layout 13](#_Toc117087231)

[Temporary XAML layout to test SQL commands … 14](#_Toc117087232)

[Create and execute MySqlConnector commands 15](#_Toc117087233)

[Designing for layouts for small devices 17](#_Toc117087234)

[Return values using MySqlConnector 17](#_Toc117087235)

[Troubleshooting ‘Unable to connect to any of the specified MySQL Hosts’ 22](#_Toc117087236)

[MVVM: *‘Model, View, ViewModel’* 24](#_Toc117087237)

[Create MVVM structure in an application 24](#_Toc117087238)

[Move MySqlConnector Tests 25](#_Toc117087239)

[Xamarin Navigation 28](#_Toc117087240)

[Navigate to MySqlConn.xaml 28](#_Toc117087241)

[Replace the MySqlConnector test code, in located in MainPage 29](#_Toc117087242)

[Add a Navigation method to MainPage 30](#_Toc117087243)

[MainPage tidied up 31](#_Toc117087244)

[Create Login Page 33](#_Toc117087245)

[Add XAML Layout elements 34](#_Toc117087246)

[Add an Image 35](#_Toc117087247)

[Add an image to the solution 35](#_Toc117087248)

[Add the image to the View 36](#_Toc117087249)

[Adding MVVM buttons 38](#_Toc117087250)

[Add the Views ‘MainMenu’ and ‘NewUser’ 39](#_Toc117087251)

[Write MVVM button code 40](#_Toc117087252)

[Create the Xamarin object model ‘User’ 41](#_Toc117087253)

[Create the class User 41](#_Toc117087254)

[Create private properties for ‘User’ 42](#_Toc117087255)

[Create Public properties for ‘User’ 42](#_Toc117087256)

[Create public properties for ‘User’ 43](#_Toc117087257)

[Create the method SetProperty() 44](#_Toc117087258)

[Generic methods 45](#_Toc117087259)

[CallerMemberName 45](#_Toc117087260)

[Invoke the event PropertyChanged 45](#_Toc117087261)

[User object - complete 46](#_Toc117087262)

[Encryption using CBC 47](#_Toc117087263)

[Create the class ‘Encryption’ 47](#_Toc117087264)

[Encryption code 47](#_Toc117087265)

[Encryption constructor 48](#_Toc117087266)

[CreateEncryptor() 48](#_Toc117087267)

[EncryptString() 49](#_Toc117087268)

[DecryptString() 50](#_Toc117087269)

[TestEncryption() 51](#_Toc117087270)

[Introduce SOLID to the login classes 53](#_Toc117087271)

[SOLID Summary 53](#_Toc117087272)

[Single Responsibility Principle (SRP) 53](#_Toc117087273)

[Open Closed Principle (OCP) 53](#_Toc117087274)

[SOLID ‘S’ 54](#_Toc117087275)

[SOLID ‘O’ 55](#_Toc117087276)

[Extract interfaces for types in vmLogin 55](#_Toc117087277)

[Extract an interface for the type ‘User’ 56](#_Toc117087278)

[Converting other Instances 57](#_Toc117087279)

[Create interfaces in vmLogin 58](#_Toc117087280)

[Create interfaces in ‘AuthenticateUser.cs’ 59](#_Toc117087281)

[Create ‘IGetMySqlData’ 59](#_Toc117087282)

[‘getLoginDetails.errorMessage’ not longer works 60](#_Toc117087283)

[Create ‘IEncryptionCbcCombo’ 61](#_Toc117087284)

[Cyclic Block Cypher encryption 62](#_Toc117087285)

[Combine ‘IEncryption’ and ‘IEncryptionCbc’ 63](#_Toc117087286)

[IEncryption 63](#_Toc117087287)

[IEncryptionCbc 63](#_Toc117087288)

[Create ‘EncryptionCbcCombo’ 63](#_Toc117087289)

[Create interfaces in ‘Encryption.cs’ 65](#_Toc117087290)

[‘O’ in SOLID Complete 65](#_Toc117087291)

[Create interfaces in ‘GetLoginDetails.cs’ 66](#_Toc117087292)

[Create ‘IConnection’ 66](#_Toc117087293)

[New Errors ‘sqlCmd’ and ‘mySqlConn.connection’ 66](#_Toc117087294)

[‘L’ in SOLID 68](#_Toc117087295)

[‘I’ in SOLID 69](#_Toc117087296)

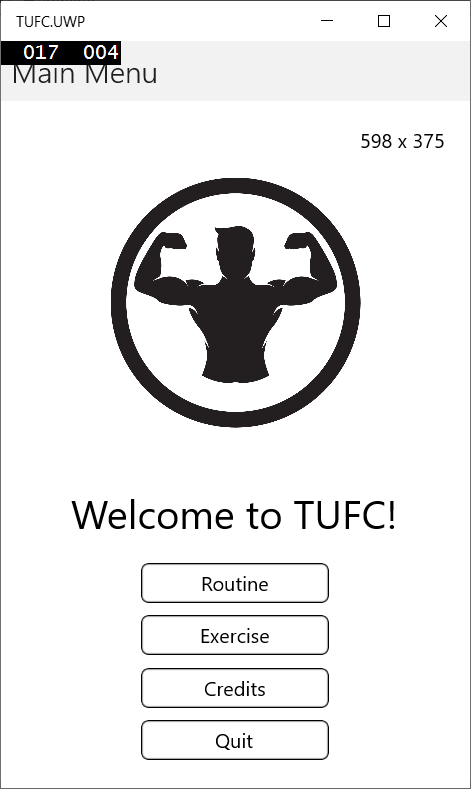
[‘D’ in SOLID 70](#_Toc117087297)

# Introduction

The Xamarin application **TUFCv3** *(The Ultimate Fitness Companion)*

is part of the project XWM *(Xamarin, Web server, MySQL)*

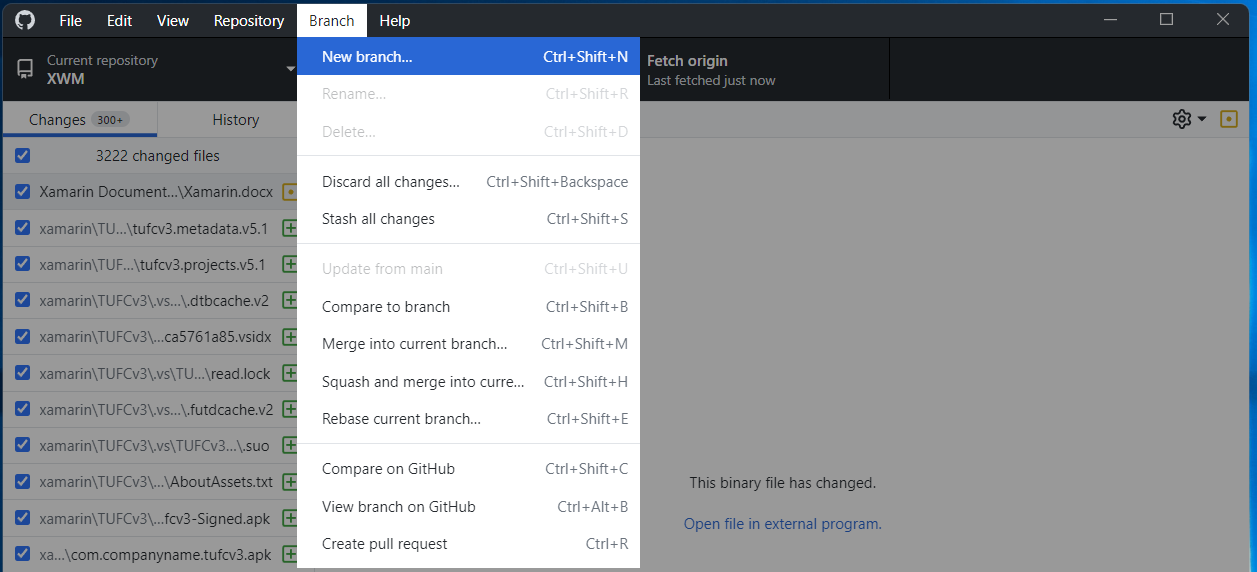
XWM is demonstration *(including ‘how-to’ documentation)*

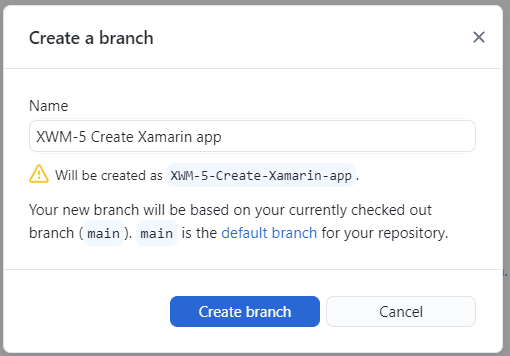
bringing these technologies together.

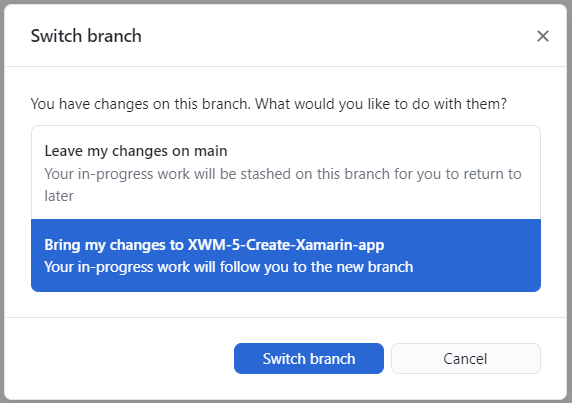
# Create a new GitHub branch

Before saving Xamarin software to GitHub

create a new branch.



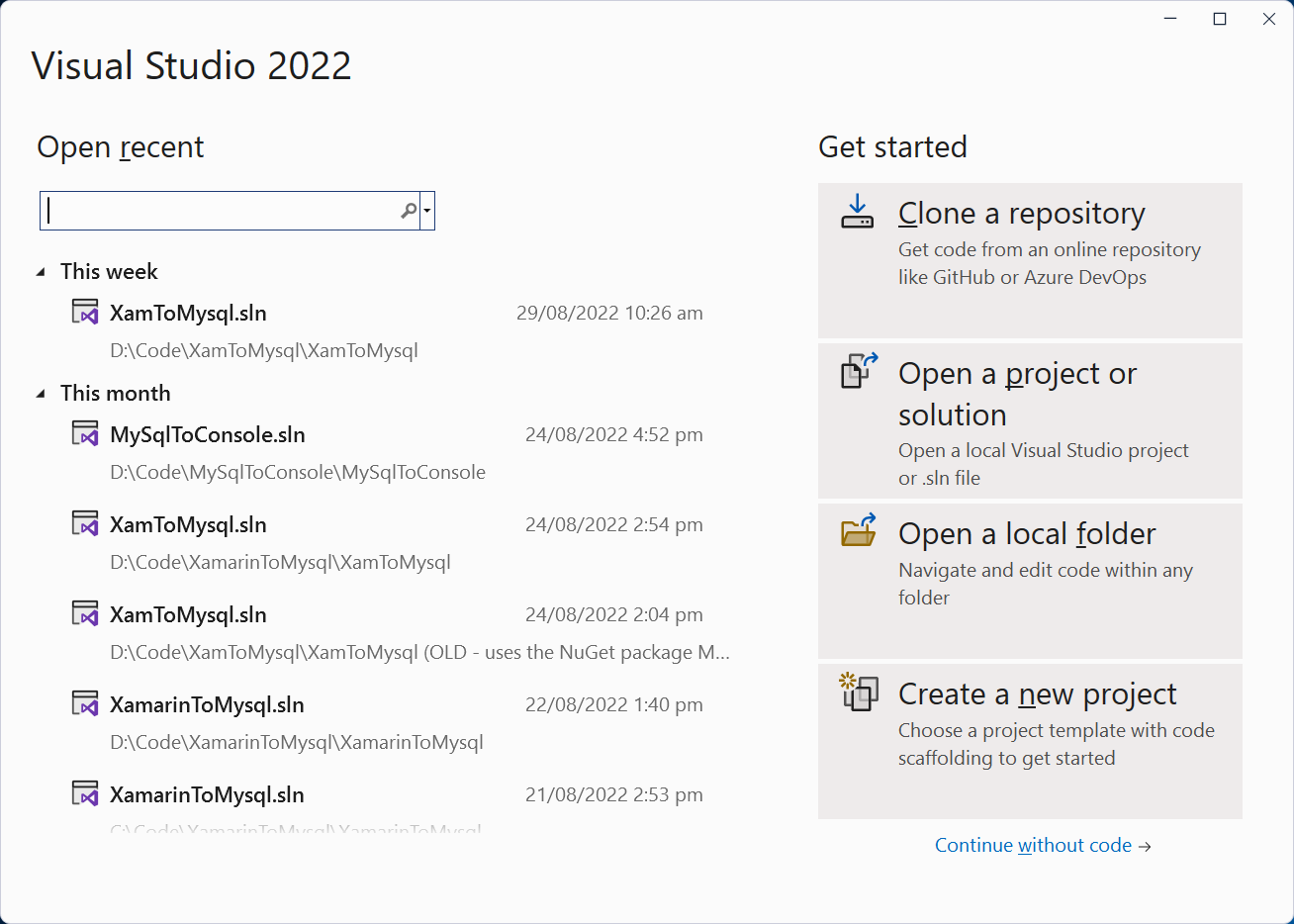


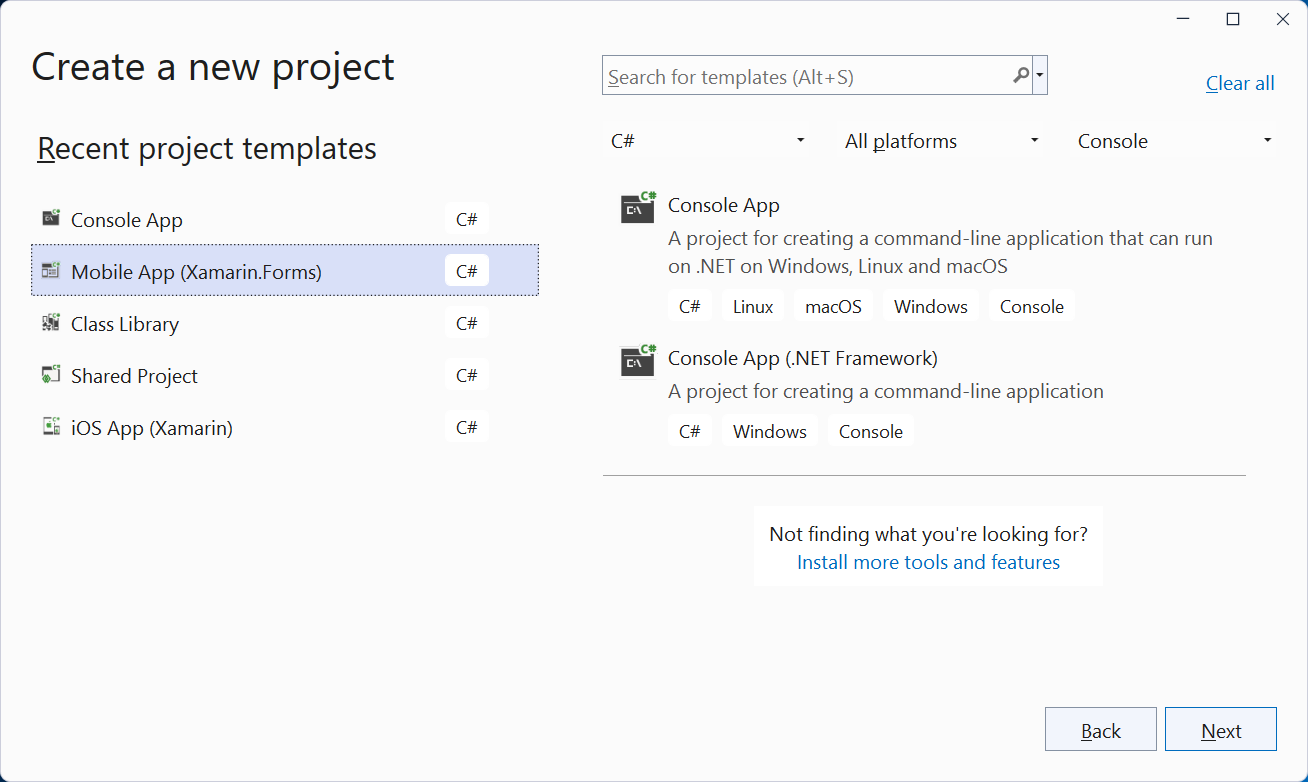


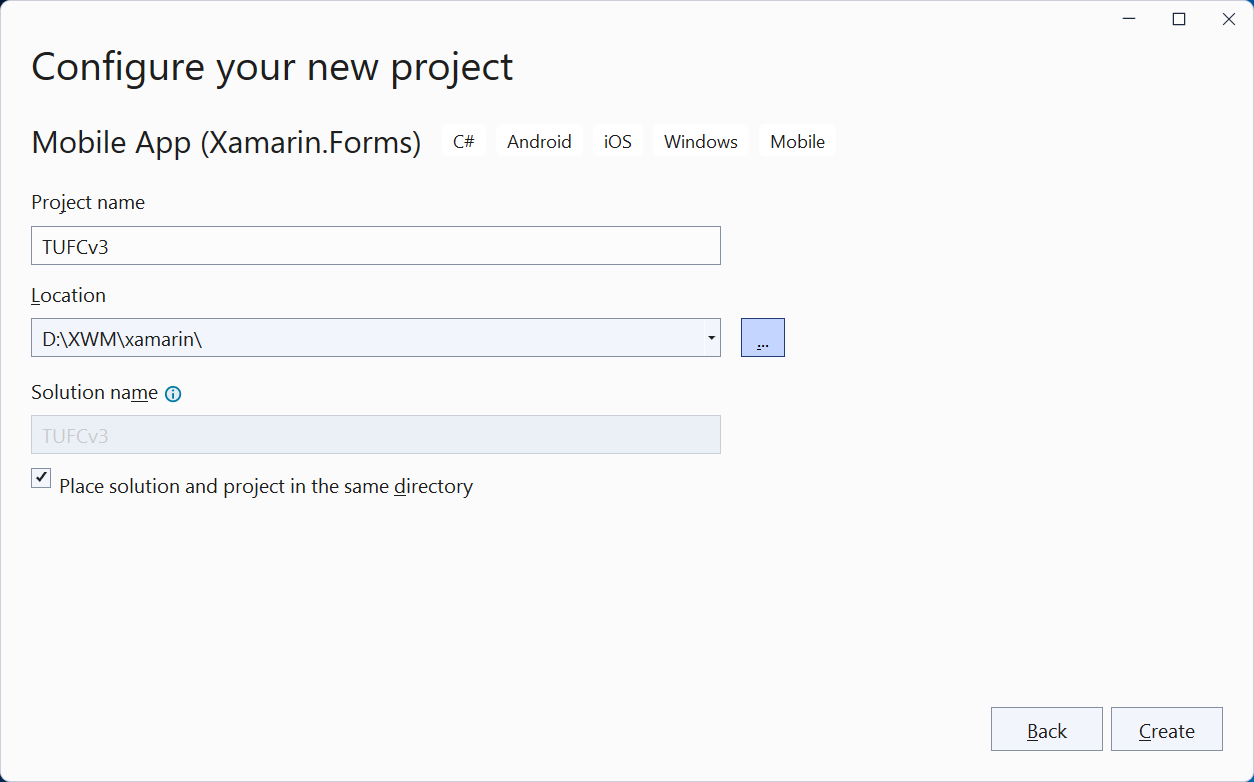
# Create TUFCv3

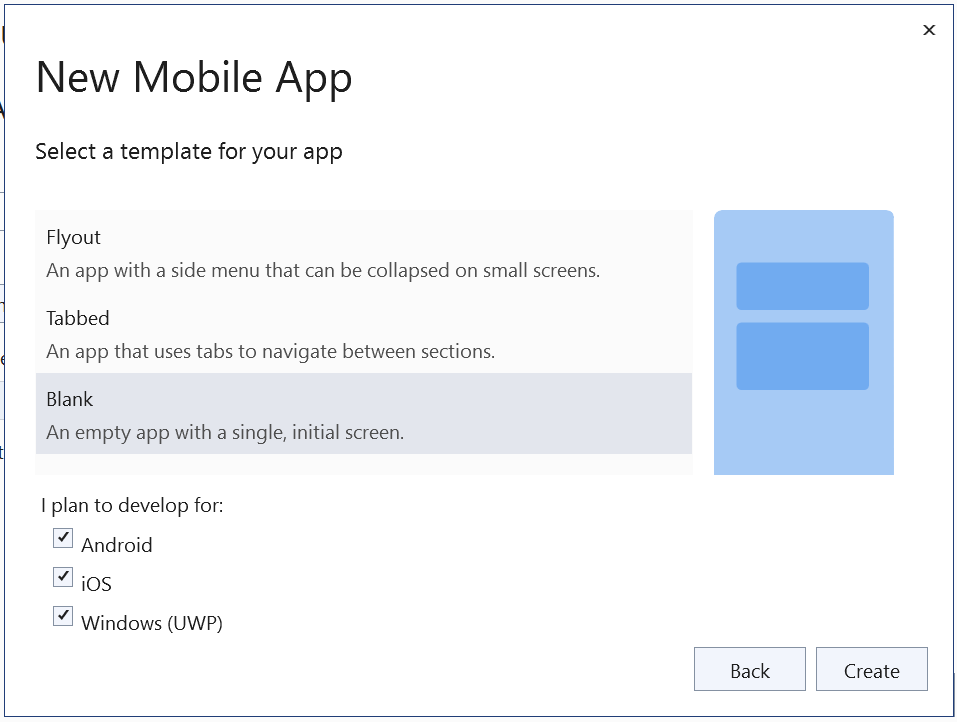
Create the Xamarin project TUFCv3

in the folder D:\XWM\xamarin









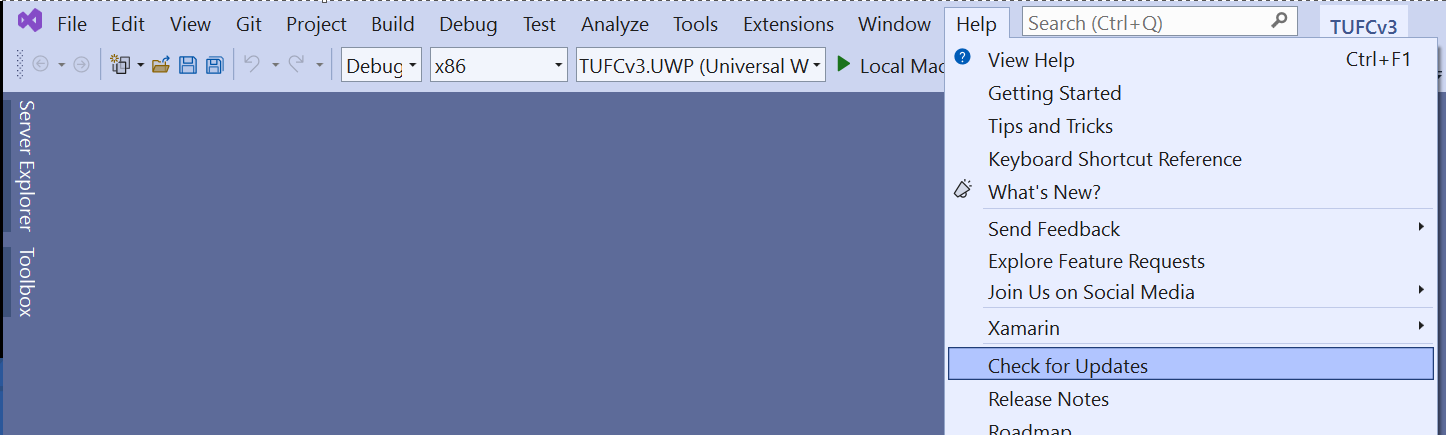
*Next, Update NuGet and run the application …*

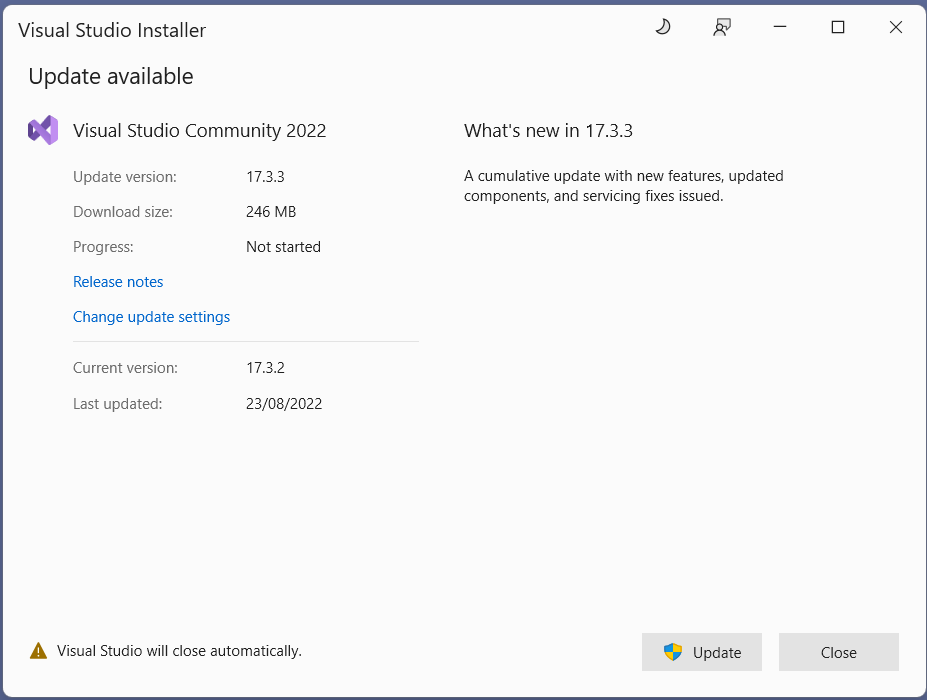
## Update Software

Before running the application check for updates on:

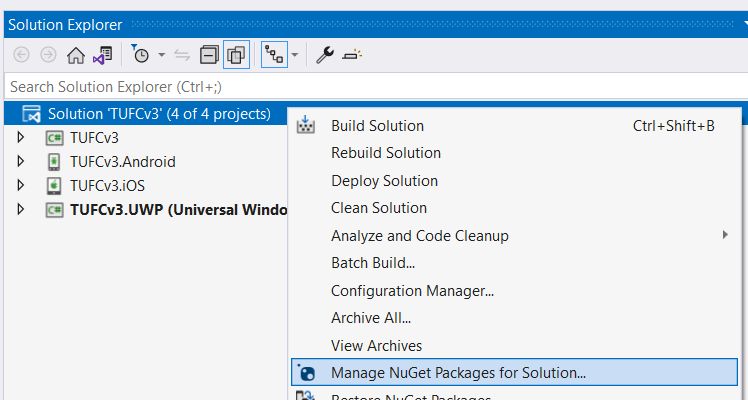
* Visual Studio
* NuGet
* Android
* Mac

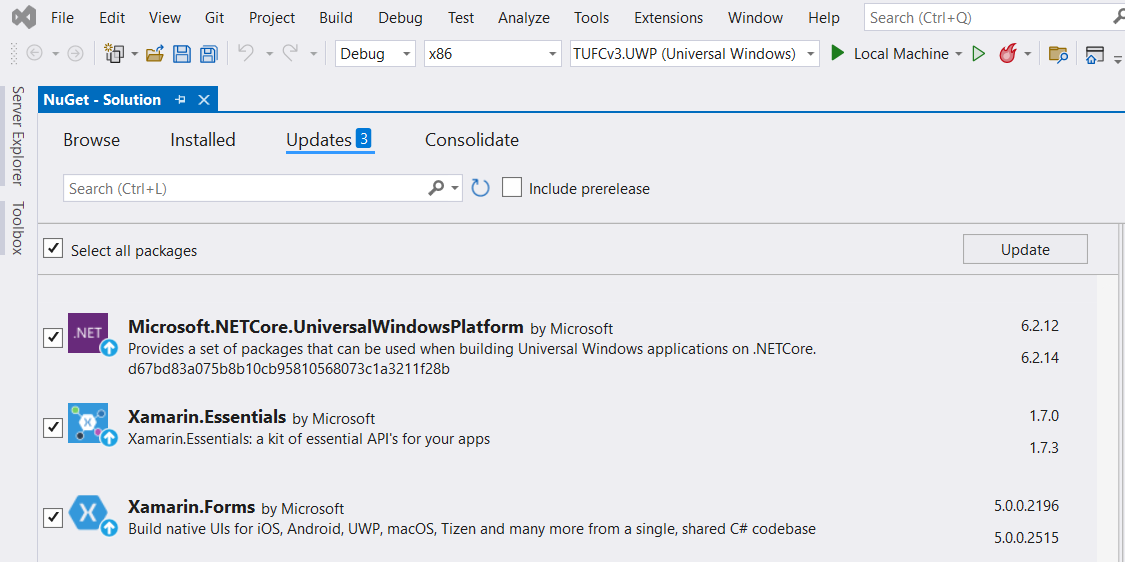
#### Update Visual Studio





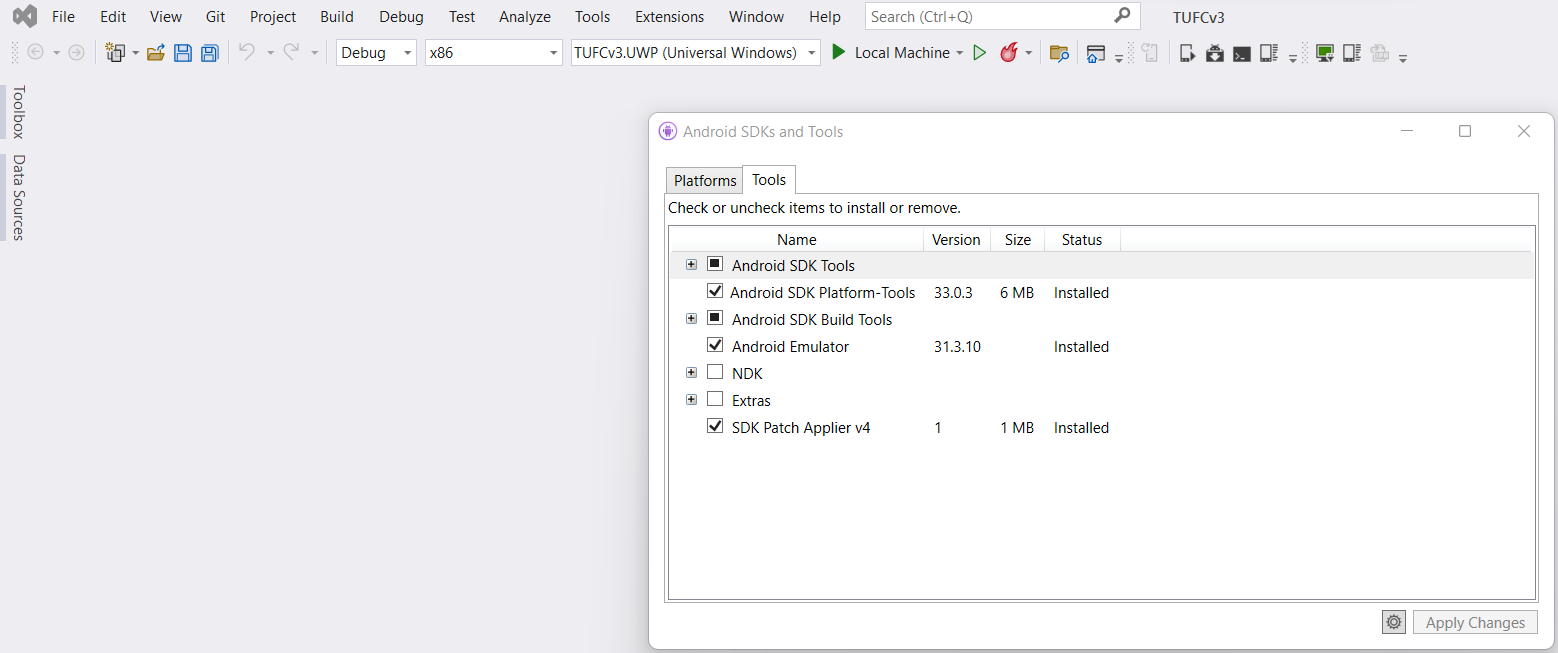
#### NuGet





*Next, Android SDK updates …*

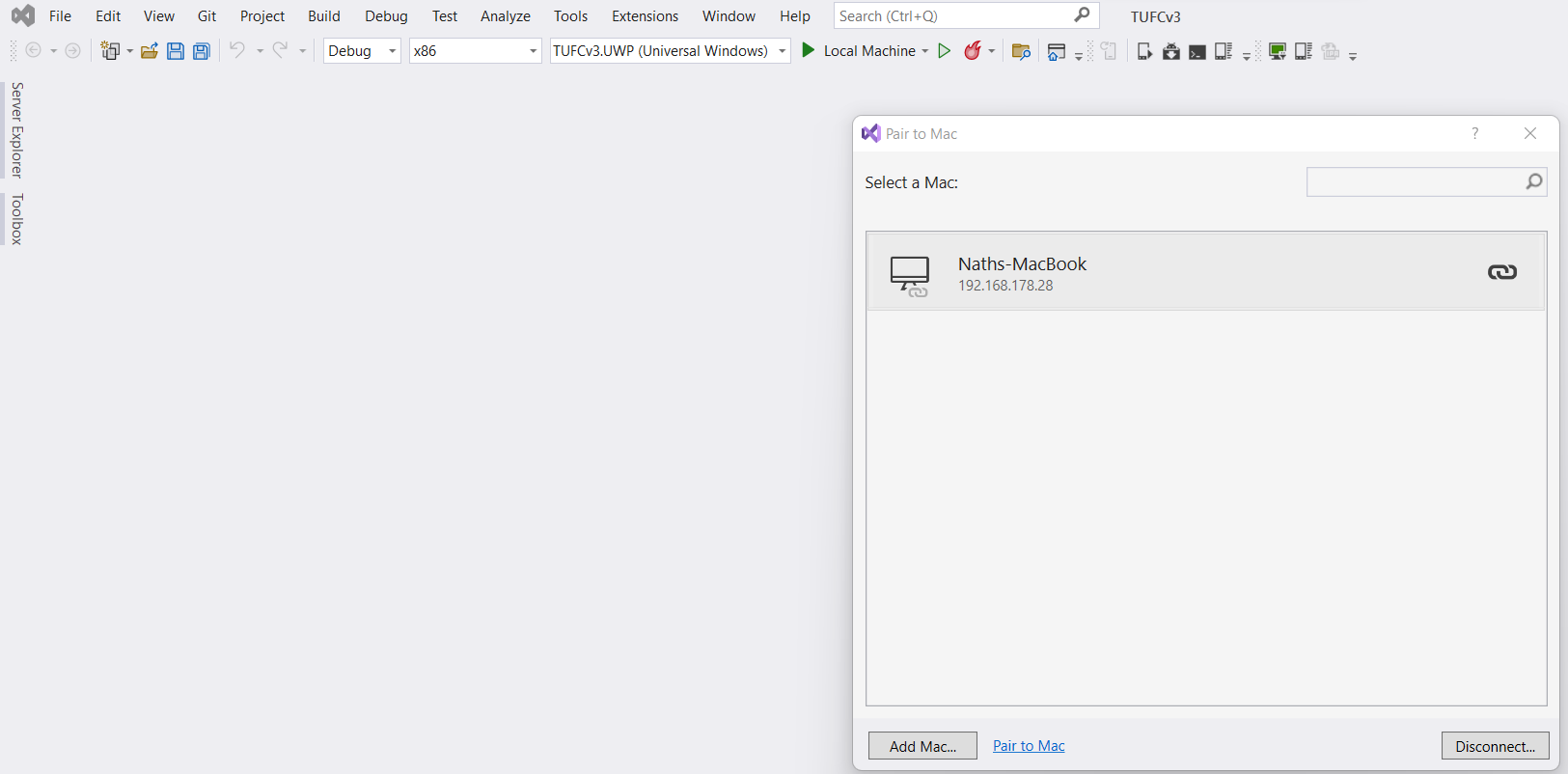
#### Android – Check the SDK manager for updates



#### Mac

When running iOS emulators on a PC, Visual Studio checks for software compatibility

while pairing with the Mac and updates Xcode versions automatically.



If the PC and Mac pair, but the emulator does not start properly

it is worth trying to update the Mac’s operating system.

Although Microsoft, Android and Apple do their best to minimise incompatibility

there may also be times when one company updates software

before compatible software is released by the others.

In this case it is usually a matter of waiting 24 hours

before compatible updates become available,

after which the Android/iOS app will build

and run correctly on the emulator.

## Run the app

Now that the solution TUFCv3 has been created

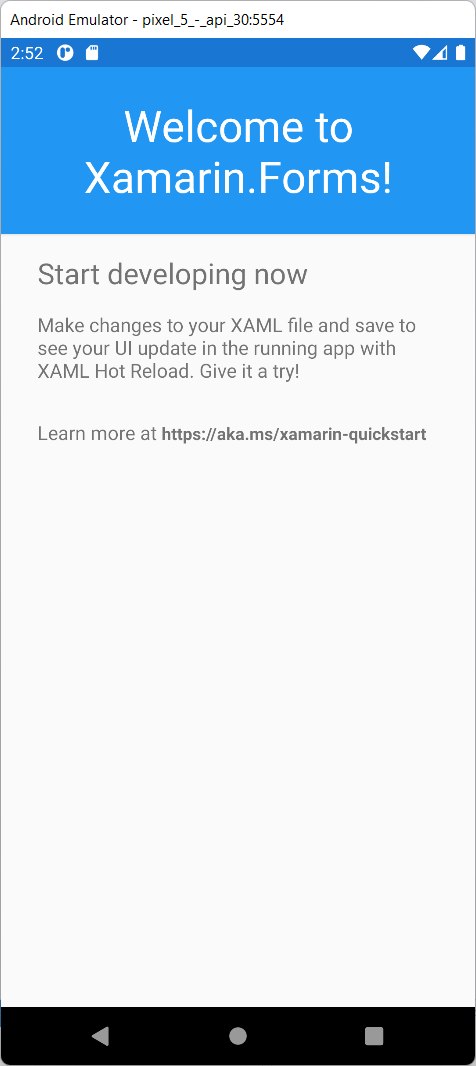
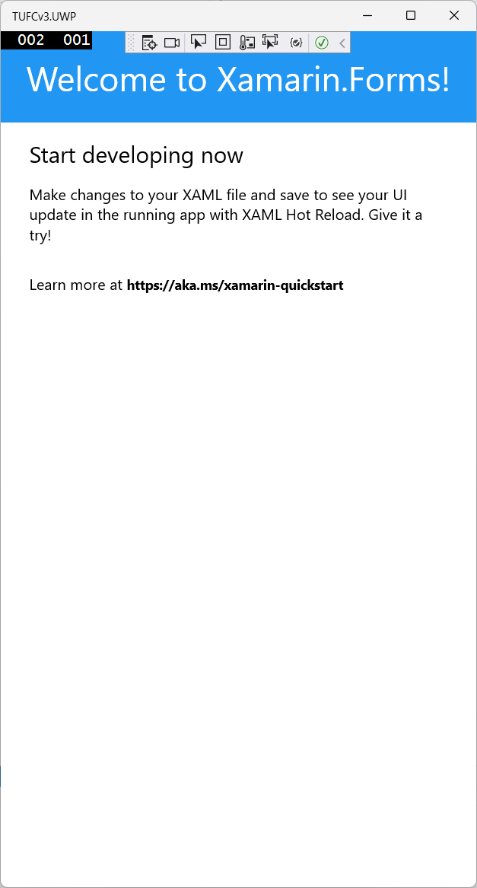
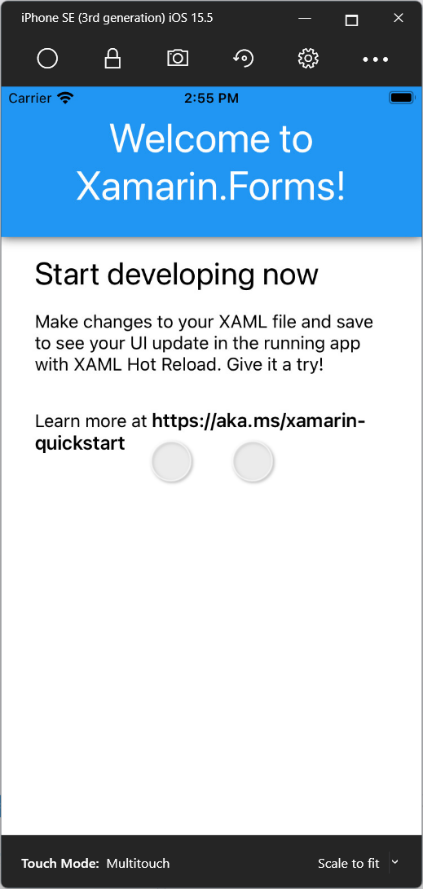
and software is up to date, run the application.

#### Test – Run the application ‘TUFCv3’

Before making any changes, run the application TUFCv3

to make sure the environment is working correctly - okay

*Android iOS UWP*



# Install MySqlConnector

https://mysqlconnector.net/

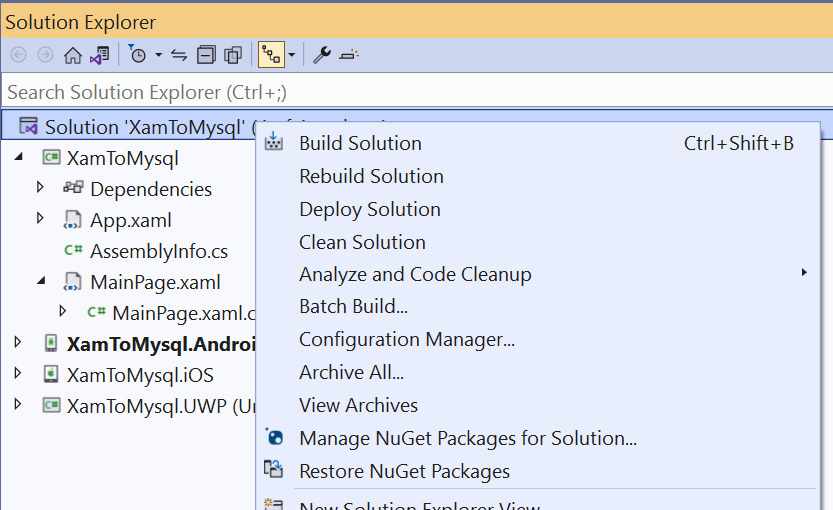
To allow the Xamarin application to connect to the MySQL database

install the NuGet package MySqlConnector

#### Example - Install MySqlConnector using NuGet

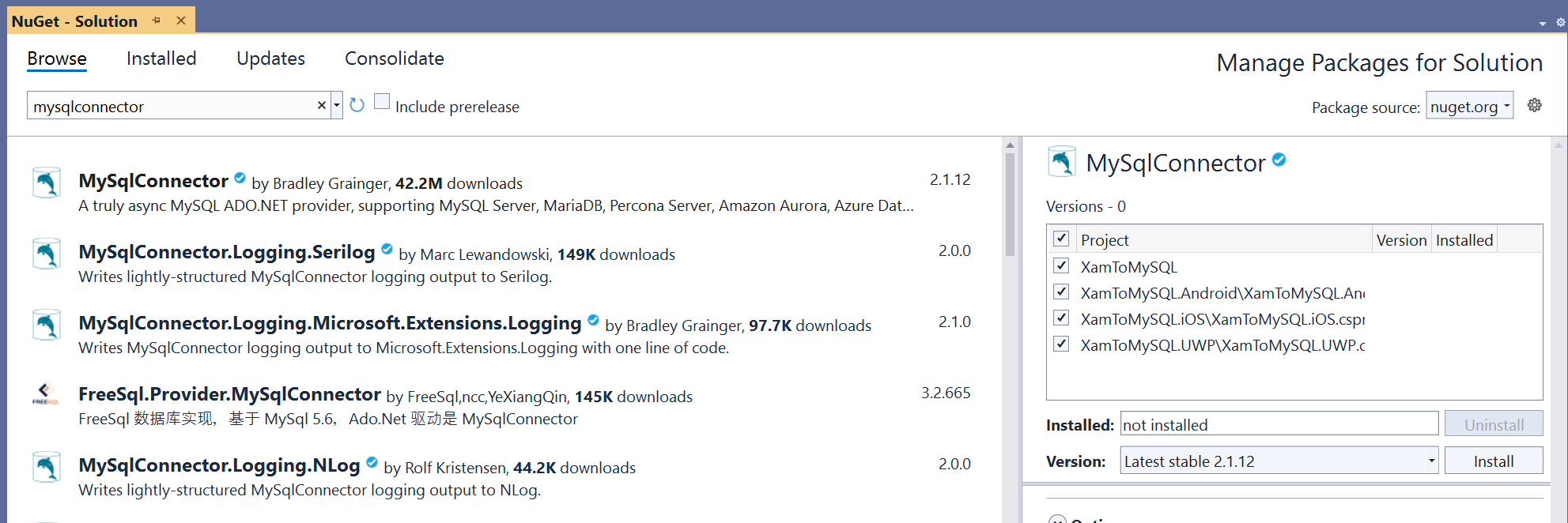
Right click the solution

and select: Manage NuGet Packages for Solution



Search for MySqlConnector

then install.



*Next, Connect to a database example using MySqlConnetor …*

#### Example - Connect to a database using MySqlConnector

In MainPage.xaml.cs

call the method ConnectToMySql()

from the constructor

using System;

using Xamarin.Forms;

namespace XamToMySQL

{

public partial class MainPage : ContentPage

{

public MainPage()

{

InitializeComponent();

ConnectToMySql();

}

public void ConnectToMySql()

{

}

}

}

In the method ConnectToMySql()

create a connection to the database

by calling the method MySqlConnection()

using the server’s connection details as arguments.

Then connect to the server.

using MySqlConnector;

using Xamarin.Forms;

namespace XamToMySQL

{

public MySqlConnection connection = new MySqlConnection();

public partial class MainPage : ContentPage

{

public MainPage()

{

InitializeComponent();

ConnectToMySql();

}

public void ConnectToMysql()

{

// Create a MySqlConnection, using the server xwm-mysql's connection details

connection = new MySqlConnection(

"Server=xwm-mysql;" +

"User ID=admin;" +

"Password=***pw***;" +

"Database=tufc");

}

}

}

In a try/catch statement open the connection

using a DisplayAlert() to indicate any connection errors.

namespace XamToMySQL

{

public partial class MainPage : ContentPage

{

public MySqlConnection connection = new MySqlConnection();

public MainPage()

{

InitializeComponent();

ConnectToMySql();

}

public void ConnectToMysql()

{

// Create a MySqlConnection, using the server xwm-mysql's connection details

connection = new MySqlConnection(

"Server=xwm-mysql;" +

"User ID=admin;" +

"Password=***pw***;" +

"Database=tufc");

try

{

connection.Open();

DisplayAlert("Connection", "Connected to the database xwm-mysql", "Okay");

connection.Close();

}

catch(Exception ex)

{

DisplayAlert("Connection", ex.Message, "Okay");

}

}

}

}

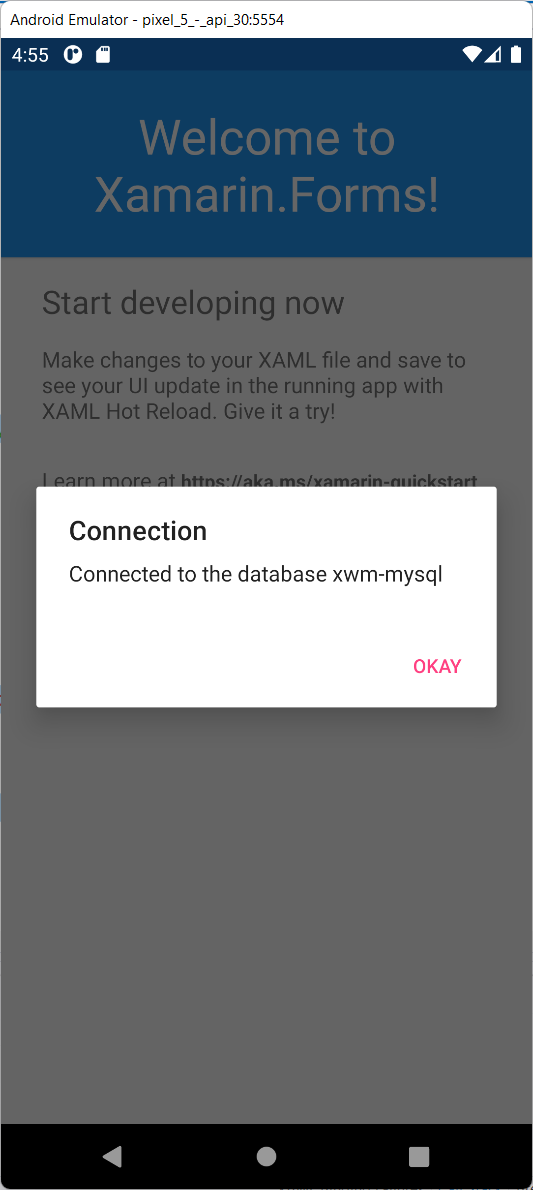
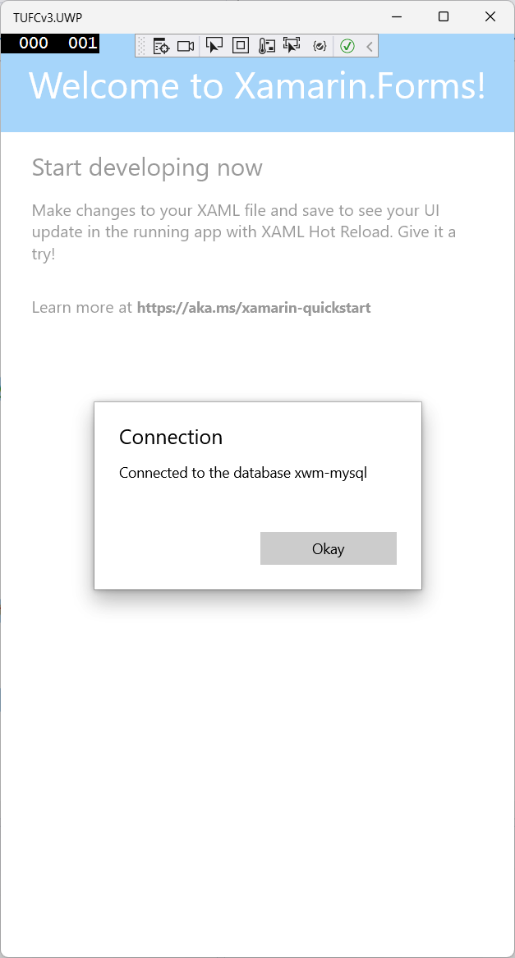
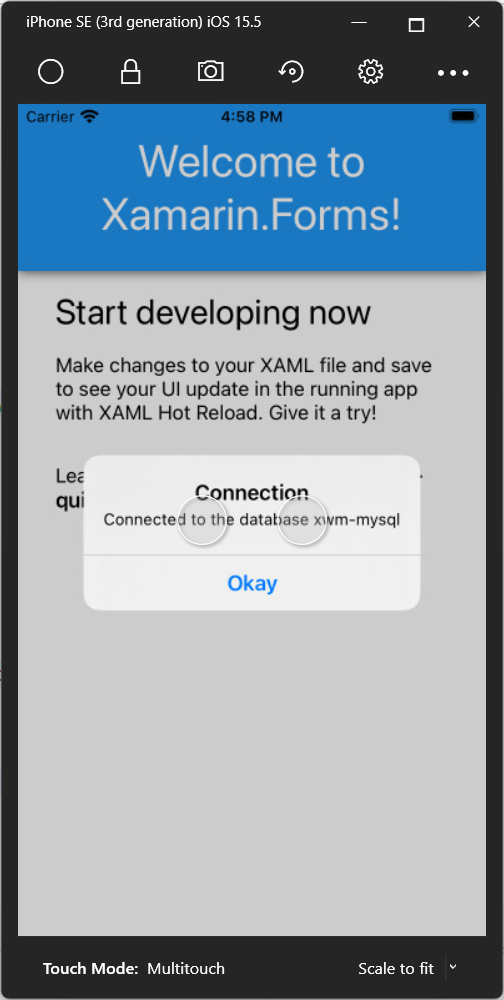
*Next, Test the connection …*

#### Test – Confirm connectivity from mobile devices to the server

Confirm connectivity from Android, iOS and UWP emulators

to the MySQL server xwm-mysql - okay

*Android iOS UWP*



*Next, Create a query …*

## Temporary layout for MainPage()

My current goal is to test interaction with the MySQL database.

This test code will be refactored into other classes

so, I don’t want to get carried away with an object model

or complicated layout, which will be modified in the near future.

Instead, I’ll create a minimal layout

that contains temporary layout controls

used while testing connectivity to the MySQL database.

### Test commands for SQL

I will test the following SQL commands *(CRUD)*

* Create INSERT
* Read SELECT
* Update UPDATE
* Delete DELETE

### Layout

Mock up the layout, before creating the XAML page.

*Title: ‘The Ultimate Fitness Companion’*

*Subtitle ’SQL command tests’*

*Label: ‘Email’ Entry: ‘Email’*

*Label: ‘New Email’ Entry: ‘newEmail’*

*Button: ‘INSERT’*

*Button: ‘UPDATE’*

*Button: ‘DELETE’*

*Button: ‘SELECT’*

*Label: ‘Users’*

*Label: (containing the list of users)*

*Next, Create the test layout for SQL commands …*

### Temporary XAML layout to test SQL commands …

#### Example – XAML, temporary layout to test SQL commands

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

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="TUFCv3.MainPage">

<StackLayout Padding="20">

<Label Text="The Ultimate Fitness Companion" FontSize="Large" HorizontalOptions="Center"/>

<Label Text="MySqlConnector Tests" FontSize="12" FontAttributes="Italic" HorizontalOptions="Center"/>

<Grid Padding="10">

<Grid.RowDefinitions>

<RowDefinition Height="\*" />

<RowDefinition Height="\*" />

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="0.2\*"/>

<ColumnDefinition Width="0.8\*"/>

</Grid.ColumnDefinitions>

<Label Text="Email:" Grid.Row="0" Grid.Column="0" HorizontalOptions="End" VerticalTextAlignment="Center"/>

<Entry x:Name="email" Grid.Row="0" Grid.Column="1" />

<Label Text="New email:" Grid.Row="1" Grid.Column="0" HorizontalOptions="End" VerticalTextAlignment="Center"/>

<Entry x:Name="newEmail" Grid.Row="1" Grid.Column="1"/>

</Grid>

<Button Clicked="OnInsertClick" Text="INSERT" HorizontalOptions="Center" HeightRequest="40" WidthRequest="250" Margin="5"/>

<Button Clicked="OnUpdateClick" Text="UPDATE" HorizontalOptions="Center" HeightRequest="40" WidthRequest="250" Margin="5"/>

<Button Clicked="OnDelectClick" Text="DELETE" HorizontalOptions="Center" HeightRequest="40" WidthRequest="250" Margin="5"/>

<Button Clicked="OnSelectClick" Text="SELECT" HorizontalOptions="Center" HeightRequest="40" WidthRequest="250" Margin="5" />

<Label Text="Users" FontSize="Medium" Padding="40,10,40,0"/>

<Label x:Name="lblUsers" Text="Email: test@mail.com &#x0a;Create Date: 1/2/2022" Padding="40,0" VerticalOptions="Start"/>

</StackLayout>

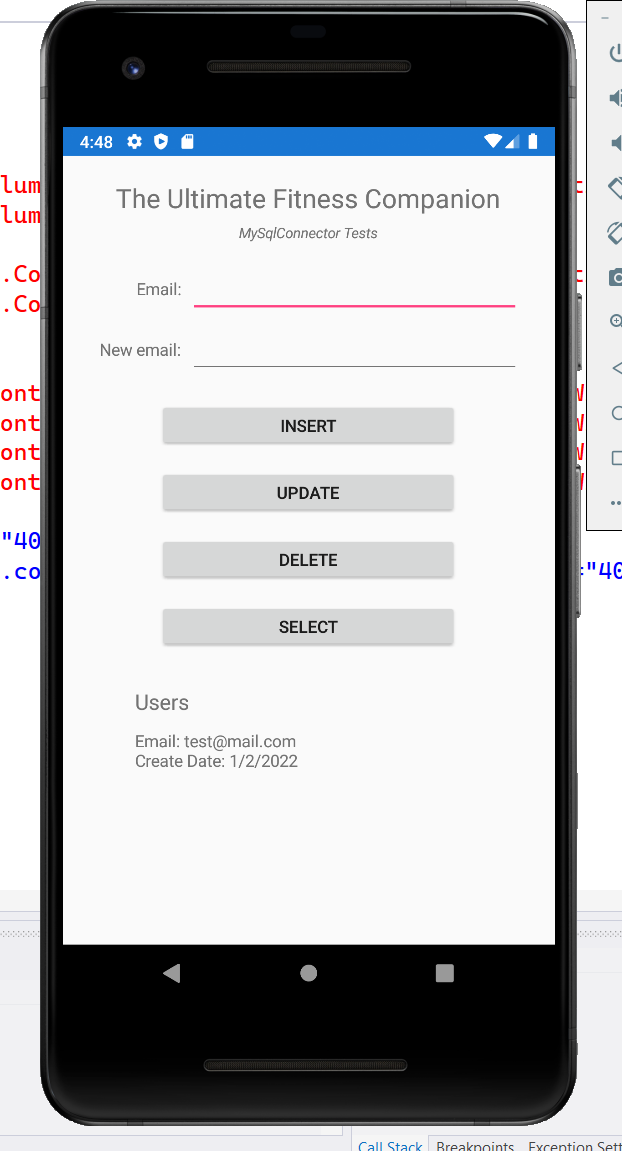
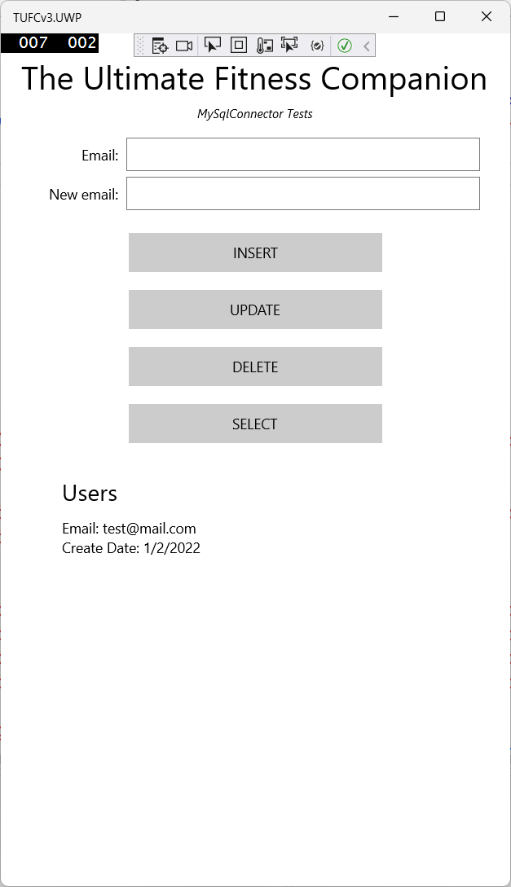
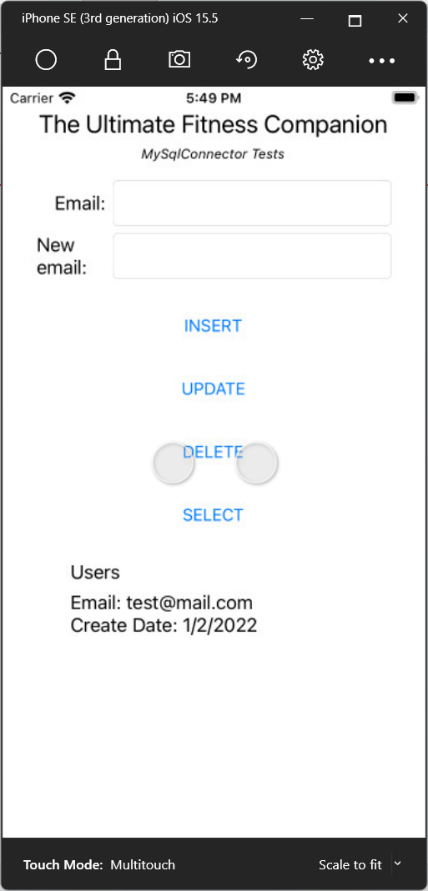
</ContentPage>

#### Test – Check temporary XAML layout

Check temporary XAML layout

to test SQL commands is working - okay

*Android iOS UWP*



## Create and execute MySqlConnector commands

I have decided to use asynchronous connections for mobile devices

while connecting to the database.

This will speed up the performance of the device, reducing hang times experienced by the user

but more importantly, it reduces the time the device is connected to the database

freeing the database for other devices queries.

*Next, MySqlConnector SELECT …*

#### Example – MySqlConnector SELECT

using System;

using Xamarin.Forms;

using MySqlConnector;

namespace TUFCv3

{

public partial class MainPage : ContentPage

{

public MySqlConnection connection = new MySqlConnection();

public MainPage()

{

InitializeComponent();

ConnectToMysql();

}

public void ConnectToMysql()

{

}

// OnSelectClick()

private async void OnSelectClick(object sender, EventArgs e)

{

// Connect to the server

try

{

connection.Open();

}

catch (Exception ex)

{

await DisplayAlert("Connection", ex.Message, "Okay");

return;

}

var query = new MySqlCommand( // Create the query

"SELECT \*" +

"FROM User",

connection);

var reader = await query.ExecuteReaderAsync();

string displayedString = "";

while (await reader.ReadAsync())

{

string email = reader.GetString(1); // User's email is already a string

string dateTime = reader.GetValue(3).ToString(); // Creation date (DateTime)

// Create the string to be displayed

displayedString +=

"\n Email: " + email +

"\n Created: " + dateTime + "\n";

}

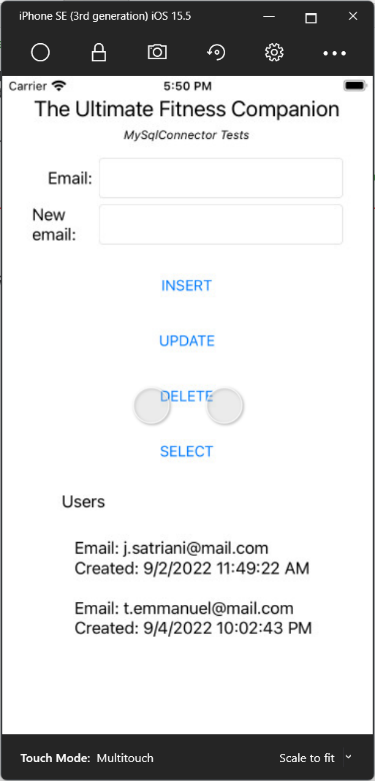
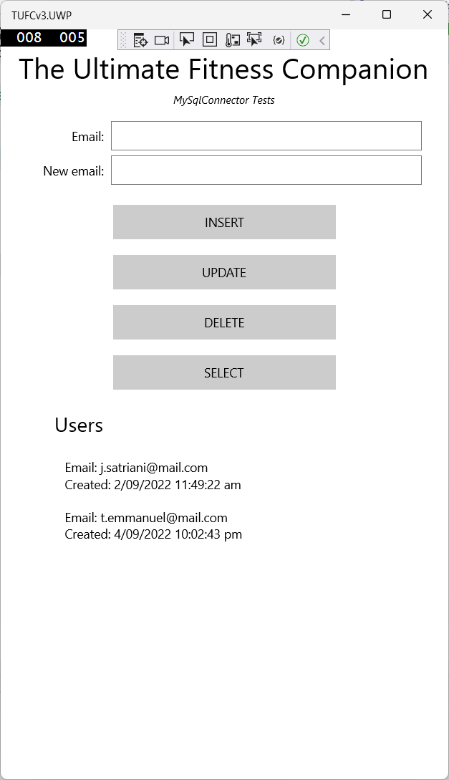
lblUsers.Text = displayedString;

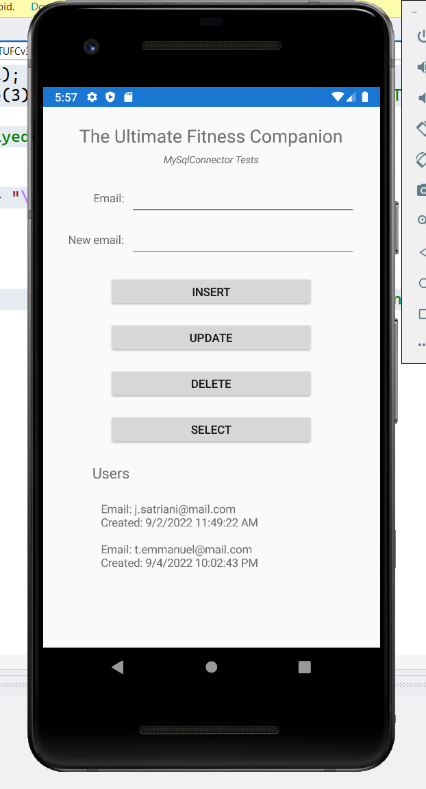
connection.Close(); // Close the connection

}

}

}

*Android iOS UWP*



Two things to note about the previous test.

* designing for layouts for small devices
* return values using MySqlConnector

### Designing for layouts for small devices

The emulators I use for testing have small screen dimensions.

They are the:

* Android ‘Pixel 2’
* Apple ‘SE3’

Designing for a small screen is easy, when you start a new project.

However, if you design layouts using emulators with larger screens

then, later discover they don’t render well on smaller devices

it can be messy to tidy up.

### Return values using MySqlConnector

The User values returned

when doing queries with MySqlConnector

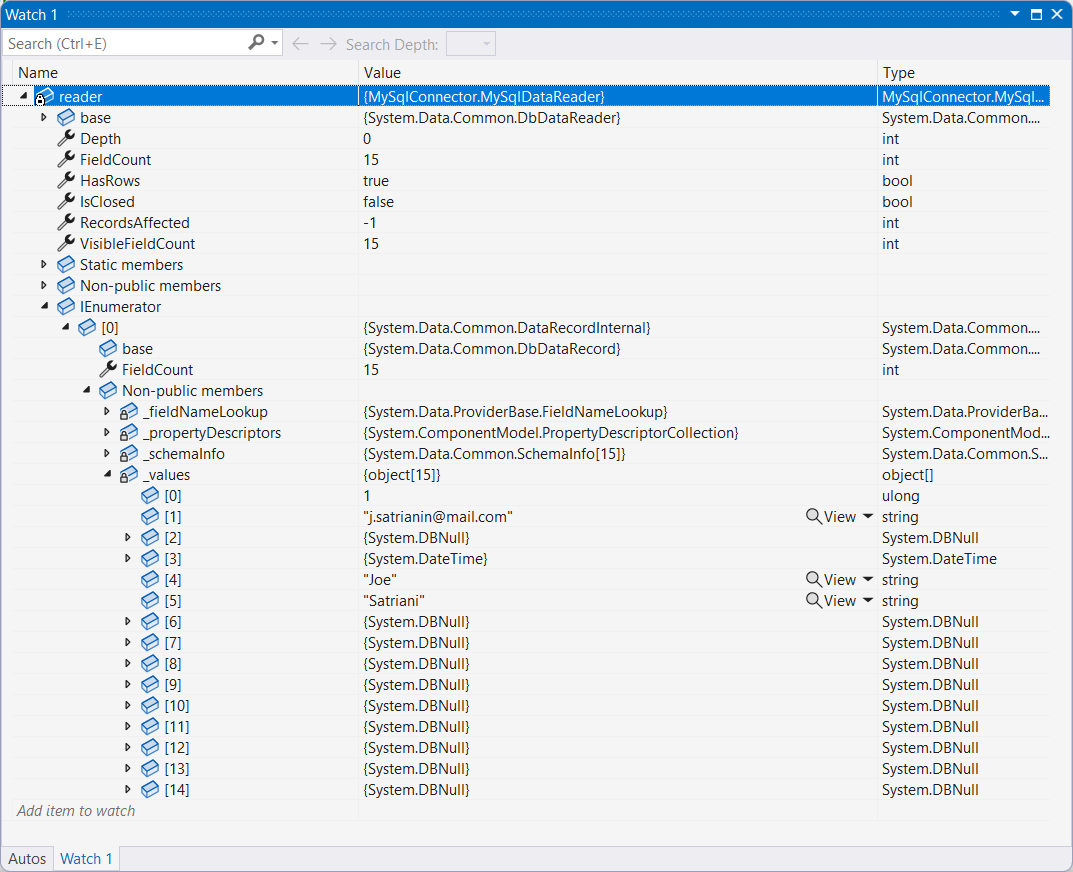
are nested arrays.

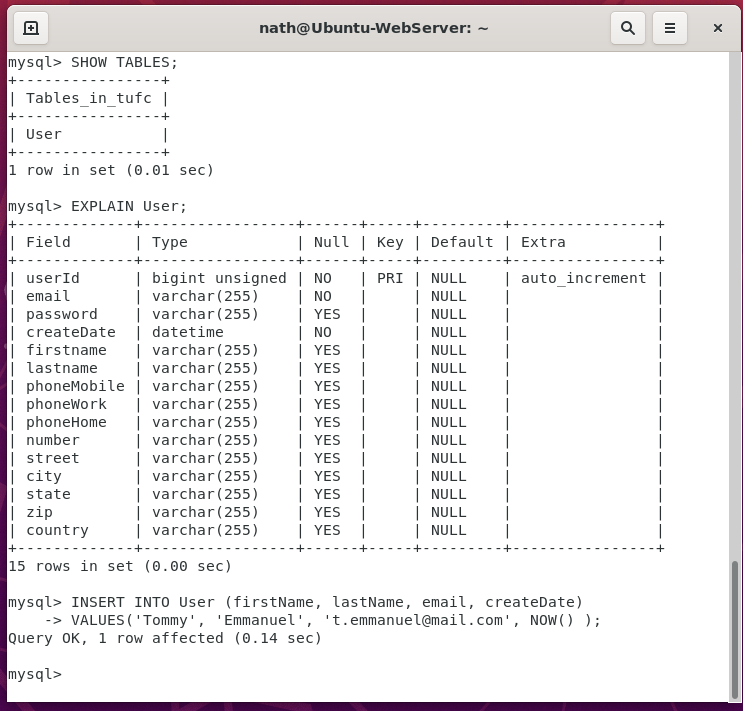
The outer array is a list of all the returned users

and the inner array is a user’s property values.

In the previous code, the MySQL User fields *(*email *and* createDate*)*

are referenced using array element numbers.

MySQL table: User Visual Studio variable: reader



#### Example – MySqlConnector INSERT

using System;

using Xamarin.Forms;

using MySqlConnector;

namespace TUFCv3

{

public partial class MainPage : ContentPage

{

public MySqlConnection connection = new MySqlConnection();

public MainPage()

{

InitializeComponent();

ConnectToMysql();

}

public void ConnectToMysql()

{

}

private async void OnInsertClick(object sender, EventArgs e)

{

// Connect to the server

try

{

connection.Open();

}

catch (Exception ex)

{

await DisplayAlert("Connection", ex.Message, "Okay");

return;

}

// Create an execute an INSERT command

using (var cmd = new MySqlCommand())

{

cmd.Connection = connection;

cmd.CommandText = "INSERT INTO User (email, createDate) VALUES (@e, @d)"; // Create a query to save data to MySQL

cmd.Parameters.AddWithValue("e", email.Text); // String from the xaml entry 'email'

cmd.Parameters.AddWithValue("d", DateTime.Now); // Current time, obtained using the function DateTime()

// Save the data

try

{

cmd.ExecuteNonQuery();

await DisplayAlert("Insert data", email.Text + " inserted \ninto the table 'User'", "Okay");

}

catch (Exception ex)

{

await DisplayAlert("Insert data", ex.Message, "Okay");

}

}

connection.Close();

}

private async void OnUpdateClick(object sender, EventArgs e)

{

}

private async void OnDeleteClick(object sender, EventArgs e)

{

}

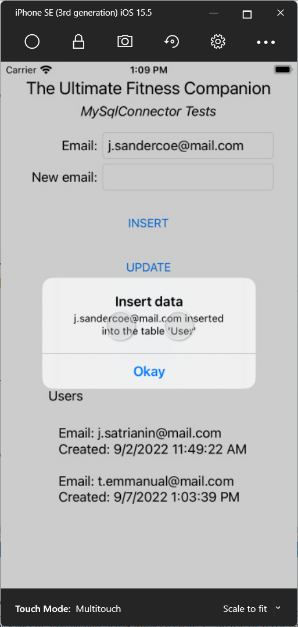
private async void OnSelectClick(object sender, EventArgs e)

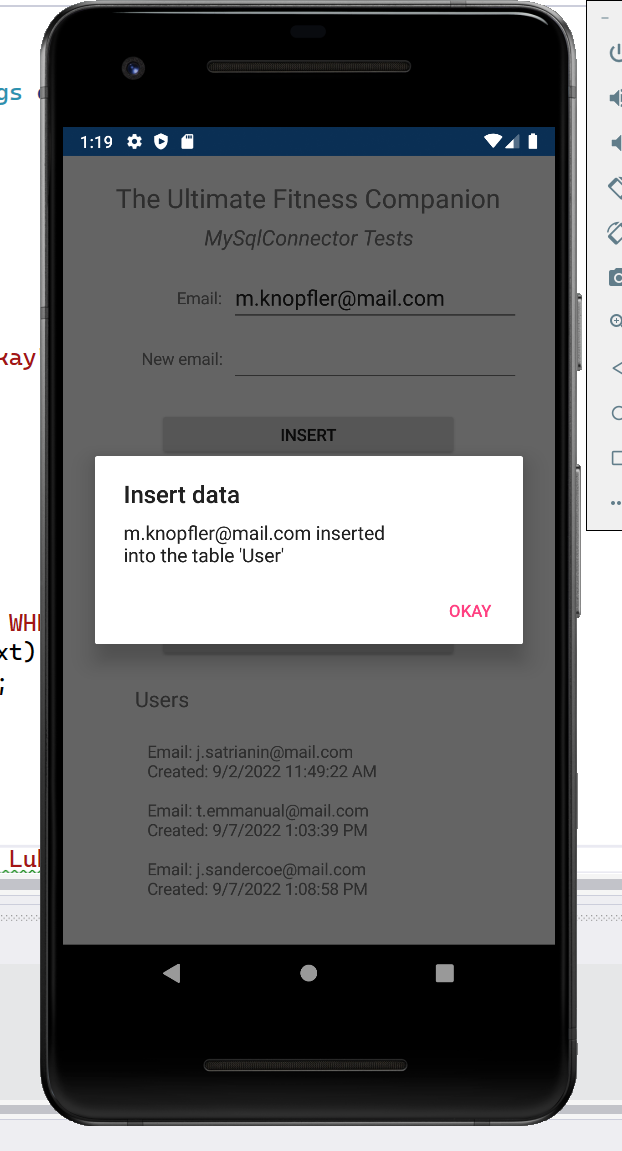
{

}

}

}

*Android iOS*



*Next, MySqlConnector command syntax …*

Note the syntax used when inserting values with MySqlConnector

// Create an execute an INSERT command

using (var cmd = new MySqlCommand())

{

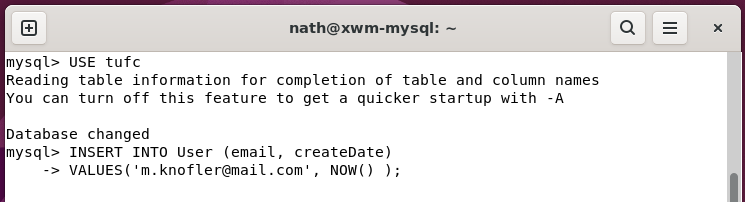
cmd.Connection = connection;

cmd.CommandText = "INSERT INTO User (email, createDate) VALUES (@e, @d)"; // Create a query to save data to MySQL

cmd.Parameters.AddWithValue("e", email.Text); // String from the xaml entry 'email'

cmd.Parameters.AddWithValue("d", DateTime.Now); // Current time, obtained using the function DateTime()

The way the values are entered not the same as a SQL command



Instead, each VALUE is assigned a parameter *(for example* @e*)*

Which is combined into the SQL command

using the method cmd.Parameters.AddWithValue(*parameter, value*)

This must be followed when using values in MySqlConnector

and ensures the correct types are inserted into the MySQL database.

#### Example – MySqlConnector UPDATE

using System;

using Xamarin.Forms;

using MySqlConnector;

namespace TUFCv3

{

public partial class MainPage : ContentPage

{

public MySqlConnection connection = new MySqlConnection();

public MainPage()

{

InitializeComponent();

ConnectToMysql();

}

public void ConnectToMysql()

{

}

private async void OnInsertClick(object sender, EventArgs e)

{

}

private async void OnUpdateClick(object sender, EventArgs e)

{

// Connect to the server

try

{

connection.Open();

}

catch (Exception ex)

{

await DisplayAlert("Connection", ex.Message, "Okay");

return;

}

using (var cmd = new MySqlCommand())

{

cmd.Connection = connection;

// Create the command UPDATE

cmd.CommandText = "UPDATE User " +

"SET email = @newEmail " +

"WHERE email = @oldEmail ";

cmd.Parameters.AddWithValue("@newEmail", newEmail.Text);

cmd.Parameters.AddWithValue("@oldEmail", email.Text);

// Execute the query

try

{

cmd.ExecuteReader();

await DisplayAlert("Connection", email.Text + " updated to " + newEmail.Text + "\n on the database 'xwm-mysql' ", "Okay");

}

catch (Exception ex)

{

await DisplayAlert("Connection", ex.Message, "Okay");

}

}

connection.Close();

}

private async void OnDeleteClick(object sender, EventArgs e)

{

}

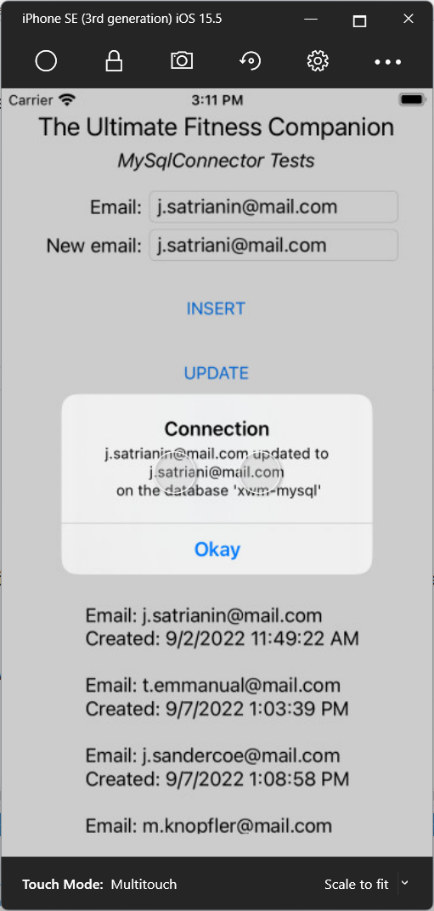
private async void OnSelectClick(object sender, EventArgs e)

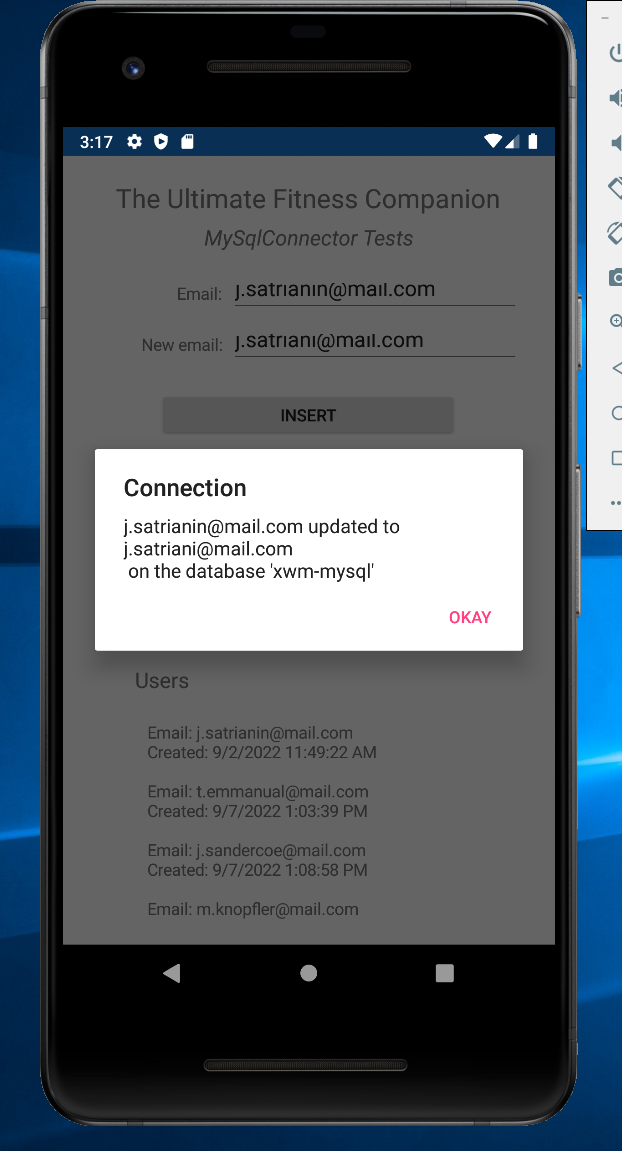
{

}

}

}

*Android iOS*



##### Example – MySqlConnector DELETE

using System;

using Xamarin.Forms;

using MySqlConnector;

namespace TUFCv3

{

public partial class MainPage : ContentPage

{

public MySqlConnection connection = new MySqlConnection();

public MainPage()

{

InitializeComponent();

ConnectToMysql();

}

public void ConnectToMysql()

{

}

private async void OnInsertClick(object sender, EventArgs e)

{

}

private async void OnUpdateClick(object sender, EventArgs e)

{

}

private async void OnDeleteClick(object sender, EventArgs e)

{

// Connect to the server

try

{

connection.Open();

}

catch (Exception ex)

{

await DisplayAlert("Connection", ex.Message, "Okay");

return;

}

using (var cmd = new MySqlCommand())

{

// Create the command to DELETE

cmd.Connection = connection;

cmd.CommandText = "DELETE FROM User " +

"WHERE email = @e";

cmd.Parameters.AddWithValue("@e", email.Text);

// Execute the query

try

{

cmd.ExecuteReader();

await DisplayAlert("Connection", "Deleted " + email.Text + "\nfrom 'Ubuntu-MySQL' ", "Okay");

}

catch (Exception ex)

{

await DisplayAlert("Connection", ex.Message, "Okay");

}

connection.Close();

}

}

private async void OnSelectClick(object sender, EventArgs e)

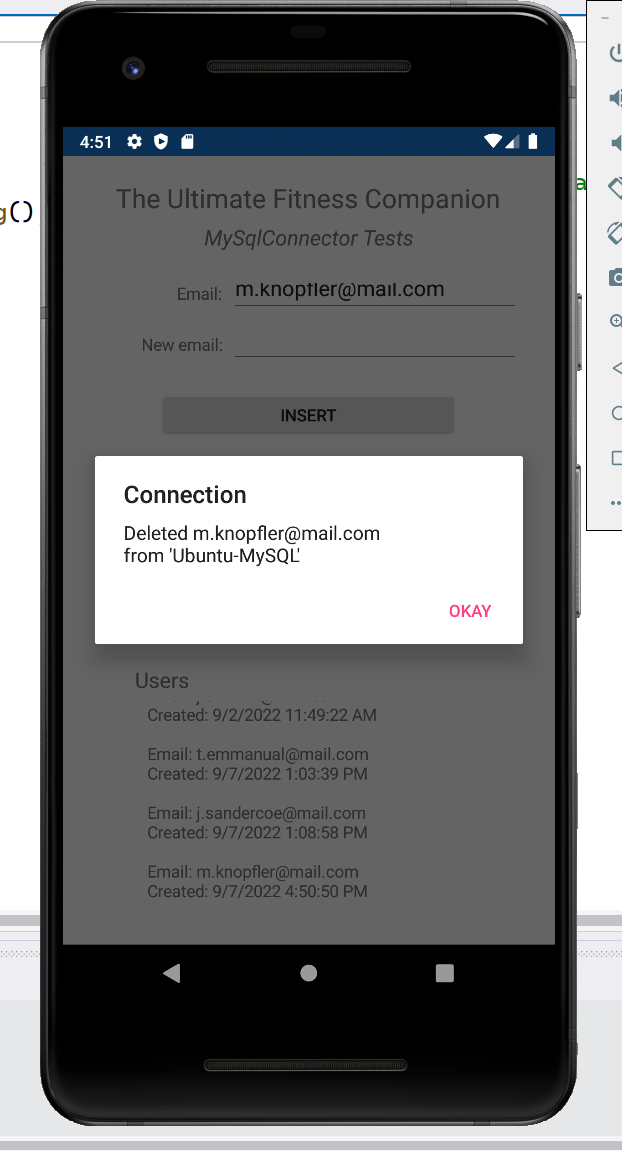
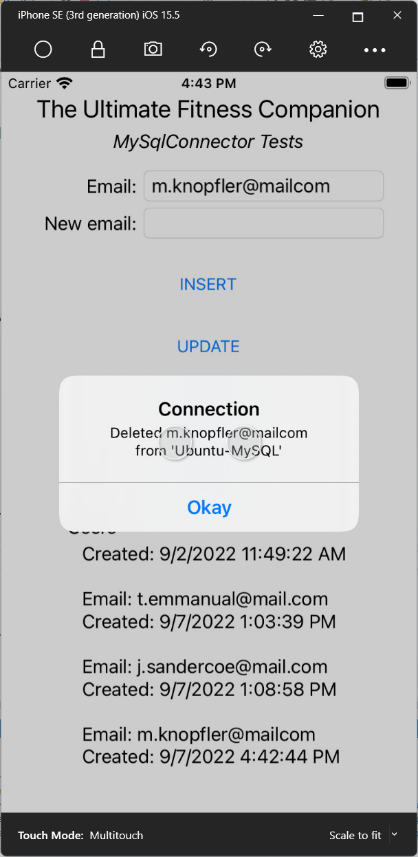
{

}

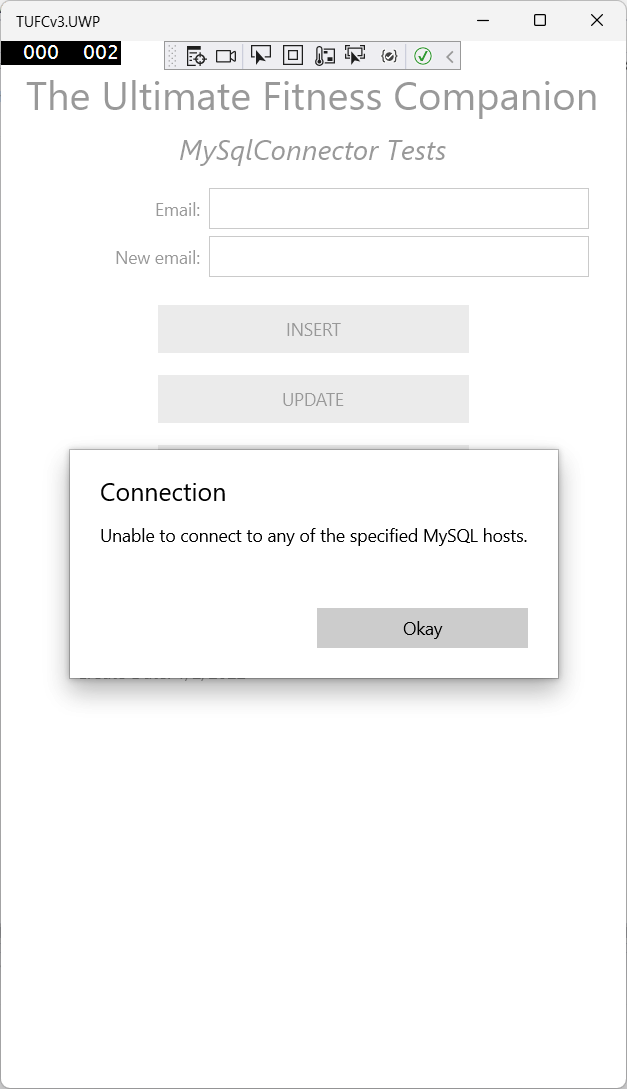
}

}

*Android iOS*



### Troubleshooting ‘Unable to connect to any of the specified MySQL Hosts’



# MVVM: *‘Model, View, ViewModel’*

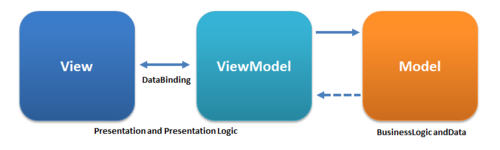
The next step in I need in the Xamarin application TUFCv3

is to create the model User

This is a good time to look at the design pattern MVVM

MVVM separates the development of the user interface (the ***View***)

from the business logic or back-end logic (the ***Model***)



*https://commons.wikimedia.org/wiki/File:MVVMPattern.png#/media/File:MVVMPattern.png*

The ***ViewModel*** exposes object models

in a way that their data can be easily managed and presented

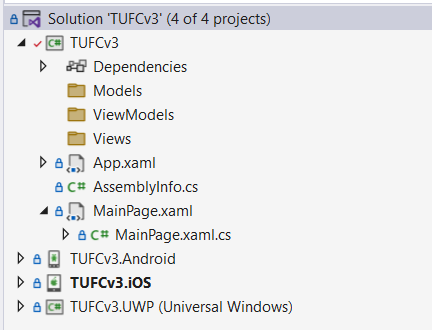
which allows the graphical user interface and the business logic

to be developed independently.

### Create MVVM structure in an application

To begin adding MVVM structure to the application TUFCv3

add folders to the shared project.

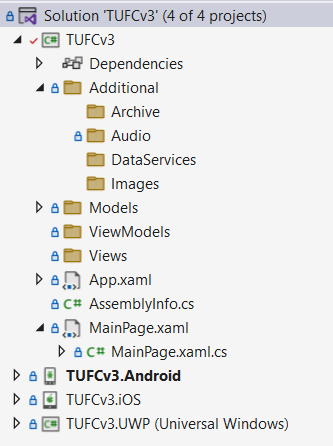


In addition to the folders for MVVM

I will also include a few additional folders

that I’ll need for this application, including:

* Audio
* Images
* DataServices *(to communicate with databases)*
* Archive *(for any example code that may come in handy in the future)*



# Move MySqlConnector Tests

It’s great to know the we can communicate

from the Xamarin app TUFCv3

to the MySQL database xwm-mysql

however, this code needs to be refactored

into the folder Additional.DataServices

I would also like to keep a copy of the original layout (XAML) and business logic (C#)

as it will be a good reference

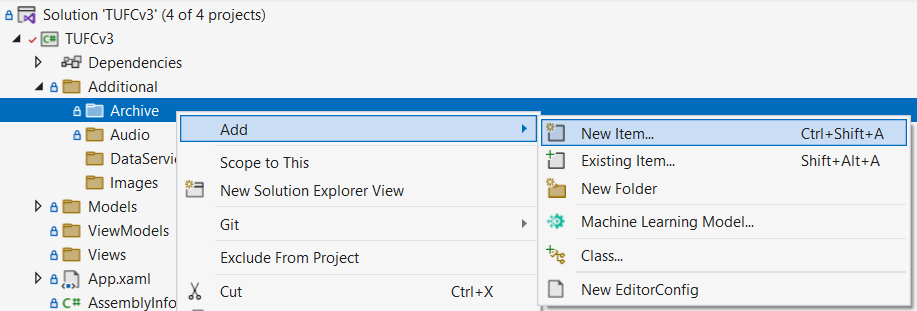
and may be helpful while doing diagnostics.

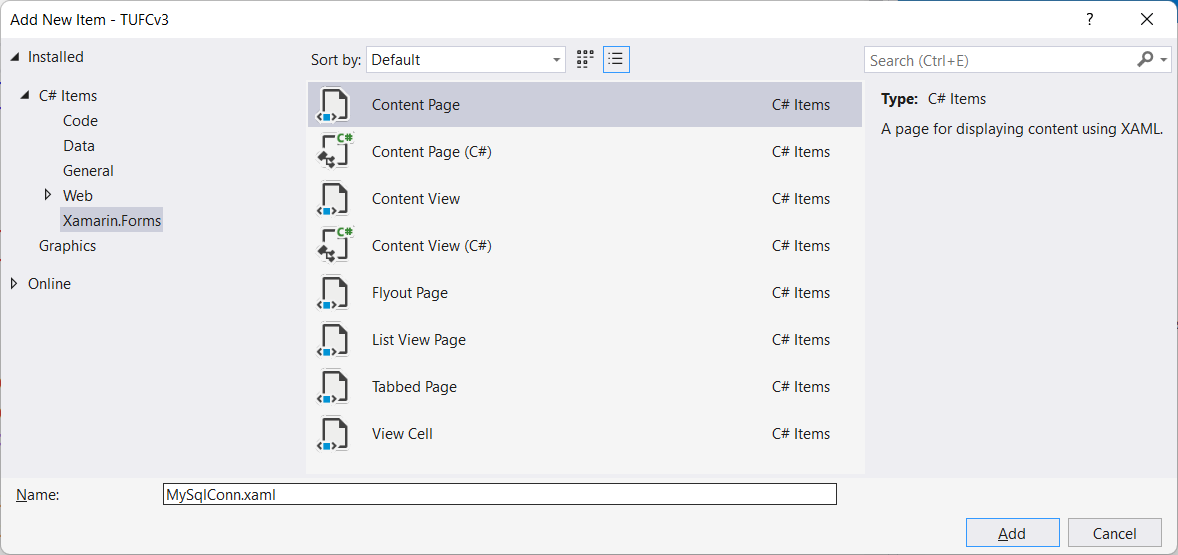
so will copy the original code to the folder Additional.Archive

#### Example – Copy MySqlConnector code to Additional.Archive

Create the new page MySqlConn.xaml

in the folder Additional.Archive





*Next, Move XAML code from* Mainpage.xaml *to* MySqlConn.xaml *…*

Replace MySqlComm.xaml code, in the section <ContentPage.Content>

with the code contained in MainPage.xaml <StackLayout>

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

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="TUFCv3.Additional.Archive.MySqlConn">

<ContentPage.Content>

<StackLayout>

<Label Text="Welcome to Xamarin.Forms!"

VerticalOptions="CenterAndExpand"

HorizontalOptions="CenterAndExpand" />

</StackLayout>

</ContentPage.Content>

<StackLayout Padding="20">

<Label Text="The Ultimate Fitness Companion" FontSize="Large" HorizontalOptions="Center"/>

<Label Text="MySqlConnector Tests" FontSize="Medium" FontAttributes="Italic" HorizontalOptions="Center"/>

<Grid Padding="10">

<Grid.RowDefinitions>

<RowDefinition Height="\*" />

<RowDefinition Height="\*" />

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="0.3\*"/>

<ColumnDefinition Width="0.7\*"/>

</Grid.ColumnDefinitions>

<Label Text="Email:" Grid.Row="0" Grid.Column="0" HorizontalOptions="End" VerticalTextAlignment="Center"/>

<Entry x:Name="email" Grid.Row="0" Grid.Column="1" />

<Label Text="New email:" Grid.Row="1" Grid.Column="0" HorizontalOptions="End" VerticalTextAlignment="Center"/>

<Entry x:Name="newEmail" Grid.Row="1" Grid.Column="1"/>

</Grid>

<Button Clicked="OnInsertClick" Text="INSERT" HorizontalOptions="Center" HeightRequest="40" WidthRequest="250" Margin="5"/>

<Button Clicked="OnUpdateClick" Text="UPDATE" HorizontalOptions="Center" HeightRequest="40" WidthRequest="250" Margin="5"/>

<Button Clicked="OnDeleteClick" Text="DELETE" HorizontalOptions="Center" HeightRequest="40" WidthRequest="250" Margin="5"/>

<Button Clicked="OnSelectClick" Text="SELECT" HorizontalOptions="Center" HeightRequest="40" WidthRequest="250" Margin="5" />

<Label Text="Users" FontSize="Medium" Padding="40,10,40,0"/>

<ScrollView>

<Label x:Name="lblUsers" Text="Email: test@mail.com &#x0a;Create Date: 1/2/2022" Padding="40,0" VerticalOptions="Start" />

</ScrollView>

</StackLayout>

</ContentPage>

Copy and paste all of the methods

from Mainpage.xaml.cs

to MySqlConn.xaml.cs

except the constructor.

*Next, Move XAML code from* Mainpage.xaml.cs *to* MySqlConn.xaml.cs *…*

Copy and paste all of the methods

from Mainpage.xaml.cs

to MySqlConn.xaml.cs

except the constructor.

using MySqlConnector;

using System;

using Xamarin.Forms;

using Xamarin.Forms.Xaml;

namespace TUFCv3.Additional.Archive

{

[XamlCompilation(XamlCompilationOptions.Compile)]

public partial class MySqlConn : ContentPage

{

public MySqlConnection connection = new MySqlConnection();

public MySqlConn()

{

InitializeComponent();

ConnectToMysql();

}

public void ConnectToMysql()

{

// Create a MySqlConnection, using the server xwm-mysql's connection details (additional options: Port=3306; SslMode=none)

connection = new MySqlConnection(

"Server=xwm-mysql;"+

"Database=tufc;" +

"User ID=admin;" +

"Password=adm1n;"

);

try

{

connection.Open();

DisplayAlert("Connection", "Connected to the database xwm-mysql", "Okay");

connection.Close();

}

catch (Exception ex)

{

DisplayAlert("Connection", ex.Message, "Okay");

}

}

private async void OnInsertClick(object sender, EventArgs e)

{

// Connect to the server

try

{

connection.Open();

}

catch (Exception ex)

{

await DisplayAlert("Connection", ex.Message, "Okay");

return;

}

// Create an execute an INSERT command

using (var cmd = new MySqlCommand())

{

cmd.Connection = connection;

cmd.CommandText = "INSERT INTO User (email, createDate) VALUES (@e, @d)"; // Create a query to save data to MySQL

cmd.Parameters.AddWithValue("e", email.Text); // String from the xaml entry 'email'

cmd.Parameters.AddWithValue("d", DateTime.Now); // Current time, obtained using the function DateTime()

// Save the data

try

{

cmd.ExecuteNonQuery();

await DisplayAlert("Insert data", email.Text + " inserted \ninto the table 'User'", "Okay");

}

catch (Exception ex)

{

await DisplayAlert("Insert data", ex.Message, "Okay");

}

}

connection.Close();

}

private async void OnUpdateClick(object sender, EventArgs e)

{

// Connect to the server

try

{

connection.Open();

}

catch (Exception ex)

{

await DisplayAlert("Connection", ex.Message, "Okay");

return;

}

using (var cmd = new MySqlCommand())

{

cmd.Connection = connection;

// Create the command UPDATE

cmd.CommandText = "UPDATE User " +

"SET email = @newEmail " +

"WHERE email = @oldEmail ";

cmd.Parameters.AddWithValue("@newEmail", newEmail.Text);

cmd.Parameters.AddWithValue("@oldEmail", email.Text);

// Execute the command

try

{

cmd.ExecuteReader();

await DisplayAlert("Connection", email.Text + " updated to " + newEmail.Text + "\non the database 'xwm-mysql' ", "Okay");

}

catch (Exception ex)

{

await DisplayAlert("Connection", ex.Message, "Okay");

}

}

connection.Close();

}

private async void OnDeleteClick(object sender, EventArgs e)

{

// Connect to the server

try

{

connection.Open();

}

catch (Exception ex)

{

await DisplayAlert("Connection", ex.Message, "Okay");

return;

}

using (var cmd = new MySqlCommand())

{

// Create the command to DELETE

cmd.Connection = connection;

cmd.CommandText = "DELETE FROM User " +

"WHERE email = @e";

cmd.Parameters.AddWithValue("@e", email.Text);

// Execute the query

try

{

cmd.ExecuteReader();

await DisplayAlert("Connection", "Deleted " + email.Text + "\nfrom 'Ubuntu-MySQL' ", "Okay");

}

catch (Exception ex)

{

await DisplayAlert("Connection", ex.Message, "Okay");

}

connection.Close();

}

}

private async void OnSelectClick(object sender, EventArgs e)

{

// Connect to the server

try

{

connection.Open();

}

catch (Exception ex)

{

await DisplayAlert("Connection", ex.Message, "Okay");

return;

}

var query = new MySqlCommand( // Create the query

"SELECT \*" +

"FROM User",

connection);

var reader = await query.ExecuteReaderAsync();

string displayedString = "";

while (await reader.ReadAsync())

{

string email = reader.GetString(1); // User's email, which is already a string

string dateTime = reader.GetValue(3).ToString(); // Creation date (DateTime)

// Create the string to be displayed

displayedString +=

"\n Email: " + email +

"\n Created: " + dateTime + "\n";

}

lblUsers.Text = displayedString;

connection.Close(); // Close the connection

}

}

}

Also add ConnectToMysql(); to the constructor

and the public variable public MySqlConnection connection = new MySqlConnection();

# Xamarin Navigation

Now that the code to test MySqlConnector

has been moved to the page Additional.Archive.MySqlConn.xaml

a way to navigate to it needs to be created.

In Xamarin, this is done using the method **Navigation()**

To navigate between pages in a Xamarin application

the way MainPage is called needs to be updated.

App.xaml.cs instantiates new NavigationPage()

with MainPage() as its argument

#### Example – Update App.xaml for page navigation

using Xamarin.Forms;

namespace TUFCv3

{

public partial class App : Application

{

public App()

{

InitializeComponent();

MainPage = new MainPage();

MainPage = new NavigationPage(new MainPage()); // Allow page navigation in the Xamarin app

}

protected override void OnStart()

{

}

protected override void OnSleep()

{

}

protected override void OnResume()

{

}

}

}

### Navigate to MySqlConn.xaml

We can now call the method Navigation()

from anywhere within the application

to move between pages.

**modal** navigation places each new page onto a stack

so, the user can navigate back and forth between pages.

**asynchronous** navigation improves the user experience

by threading application processes.

### Replace the MySqlConnector test code, in located in MainPage

Before adding page navigation

tidy up MainPage.xaml and MainPage.xaml.cs

by removing code related to MySqlConnector

In MainPage.xaml ‘comment out’ the MySqlConnector StackLayout *(previously created for MySqlConnector elements)*

and replace it with a simple ContentPage.Content

that contains a label displaying the page name “MainPage.xaml”

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

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="TUFCv3.MainPage">

<ContentPage.Content>

<StackLayout>

<Label Text="MainPage.xaml"

VerticalOptions="CenterAndExpand"

HorizontalOptions="CenterAndExpand" />

</StackLayout>

</ContentPage.Content>

<!--<StackLayout Padding="20">

<Label Text="The Ultimate Fitness Companion" FontSize="Large" HorizontalOptions="Center"/>

<Label Text="MySqlConnector Tests" FontSize="Medium" FontAttributes="Italic" HorizontalOptions="Center"/>

<Grid Padding="10">

<Grid.RowDefinitions>

<RowDefinition Height="\*" />

<RowDefinition Height="\*" />

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="0.3\*"/>

<ColumnDefinition Width="0.7\*"/>

</Grid.ColumnDefinitions>

<Label Text="Email:" Grid.Row="0" Grid.Column="0" HorizontalOptions="End" VerticalTextAlignment="Center"/>

<Entry x:Name="email" Grid.Row="0" Grid.Column="1" />

<Label Text="New email:" Grid.Row="1" Grid.Column="0" HorizontalOptions="End" VerticalTextAlignment="Center"/>

<Entry x:Name="newEmail" Grid.Row="1" Grid.Column="1"/>

</Grid>

<Button Clicked="OnInsertClick" Text="INSERT" HorizontalOptions="Center" HeightRequest="40" WidthRequest="250" Margin="5"/>

<Button Clicked="OnUpdateClick" Text="UPDATE" HorizontalOptions="Center" HeightRequest="40" WidthRequest="250" Margin="5"/>

<Button Clicked="OnDeleteClick" Text="DELETE" HorizontalOptions="Center" HeightRequest="40" WidthRequest="250" Margin="5"/>

<Button Clicked="OnSelectClick" Text="SELECT" HorizontalOptions="Center" HeightRequest="40" WidthRequest="250" Margin="5" />

<Label Text="Users" FontSize="Medium" Padding="40,10,40,0"/>

<ScrollView>

<Label x:Name="lblUsers" Text="Email: test@mail.com &#x0a;Create Date: 1/2/2022" Padding="40,0" VerticalOptions="Start" />

</ScrollView>

</StackLayout>-->

</ContentPage>

In MainPage.xaml.cs ‘comment out’ all of the MySqlConnector related code

using Xamarin.Forms;

namespace TUFCv3

{

public partial class MainPage : ContentPage

{

//public MySqlConnection connection = new MySqlConnection();

public MainPage()

{

InitializeComponent();

// ConnectToMysql();

}

//public void ConnectToMysql()

//{

// // Create a MySqlConnection, using the server xwm-mysql's connection details (additional options: Port=3306; SslMode=none)

// connection = new MySqlConnection(

// "Server=xwm-mysql;"+

// "Database=tufc;" +

// "User ID=admin;" +

// "Password=adm1n;"

// );

// try

// {

// connection.Open();

// DisplayAlert("Connection", "Connected to the database xwm-mysql", "Okay");

// connection.Close();

// }

// catch (Exception ex)

// {

// DisplayAlert("Connection", ex.Message, "Okay");

// }

//}

…

// var query = new MySqlCommand( // Create the query

// "SELECT \*" +

// "FROM User",

// connection);

// var reader = await query.ExecuteReaderAsync();

// string displayedString = "";

// while (await reader.ReadAsync())

// {

// string email = reader.GetString(1); // User's email, which is already a string

// string dateTime = reader.GetValue(3).ToString(); // Creation date (DateTime)

// // Create the string to be displayed

// displayedString +=

// "\n Email: " + email +

// "\n Created: " + dateTime + "\n";

// }

// lblUsers.Text = displayedString;

// connection.Close(); // Close the connection

//}

}

}

### Add a Navigation method to MainPage

Asynchronous code isn’t permitted in constructors.

So, from the constructor, call a new method called NavigateToMySqlConn() *(which permits asynchronous code)*

to do the page navigation.

using Xamarin.Forms;

namespace TUFCv3

{

public partial class MainPage : ContentPage

{

//public MySqlConnection connection = new MySqlConnection();

public MainPage()

{

InitializeComponent();

NavigateToMySqlConn(); // Navigate to the page MySqlComm

}

async void NavigateToMySqlConn()

{

await Navigation.PushModalAsync(new Additional.Archive.MySqlConn()); // Navigate to MySqlConn.xaml

}

// (Commented out MySqlConnector methods)

}

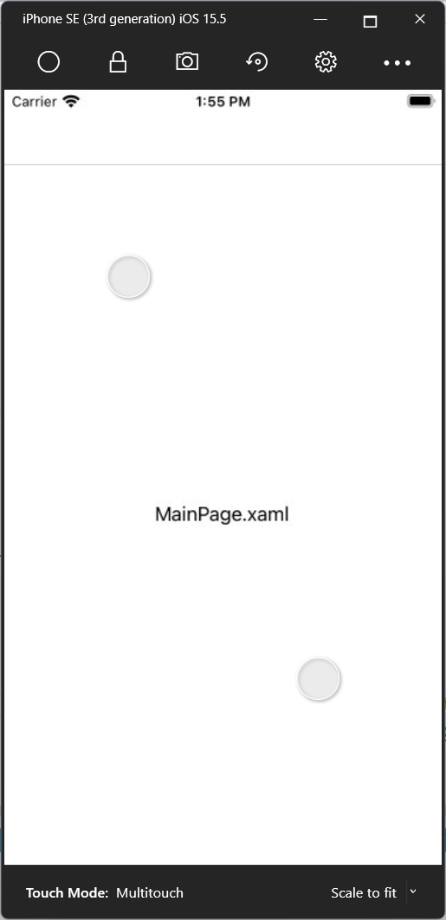
}

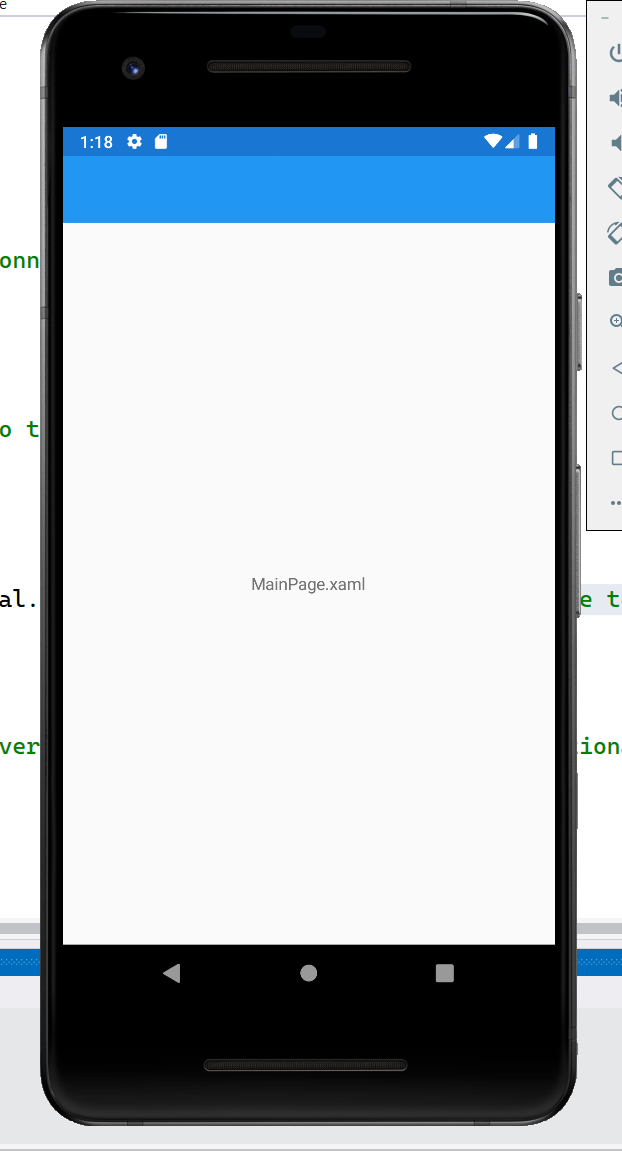
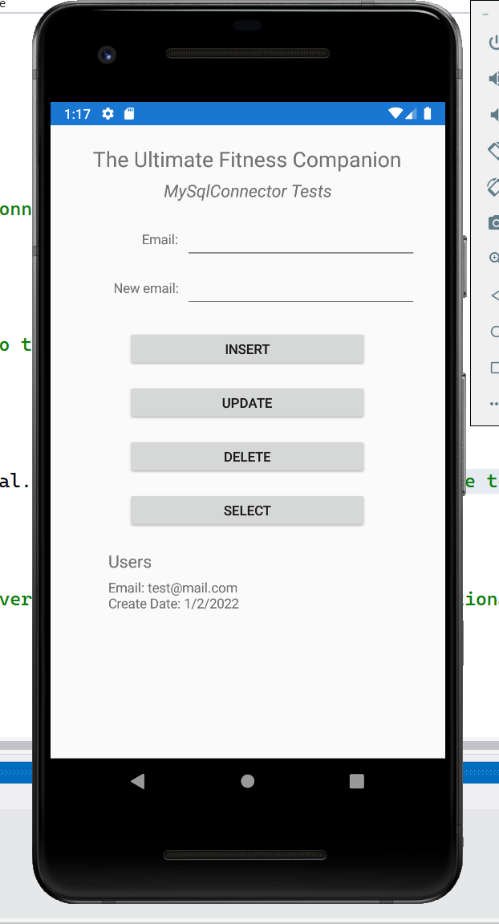
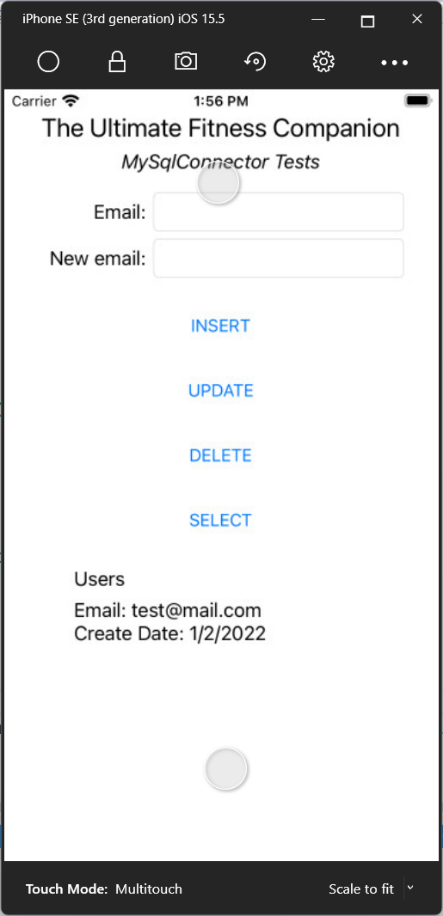
#### Test – Test navigation from ‘MainPage.xaml’ to ‘MySqlConn.xaml’

Test navigation from MainPage.xaml

to MySqlConn.xaml - okay

*Android iOS*





Now that page navigation from MainPage to MySqlConn.xaml is working

the ‘commented out’ code can be removed from MainPage

to keep application code tidy.

*Next,* MainPage *tidied up …*

### MainPage tidied up

#### MainPage.xaml

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

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="TUFCv3.MainPage">

<ContentPage.Content>

<StackLayout>

<Label Text="MainPage.xaml"

VerticalOptions="CenterAndExpand"

HorizontalOptions="CenterAndExpand" />

</StackLayout>

</ContentPage.Content>

</ContentPage>

#### MainPage.xaml.cs

using Xamarin.Forms;

namespace TUFCv3

{

public partial class MainPage : ContentPage

{

public MainPage()

{

InitializeComponent();

NavigateToMySqlConn(); // Navigate to to the page MySqlComm

}

async void NavigateToMySqlConn()

{

await Navigation.PushModalAsync(new Additional.Archive.MySqlConn()); // Navigate to MySqlConn.xaml

}

}

}

*Next, Create the object User …*

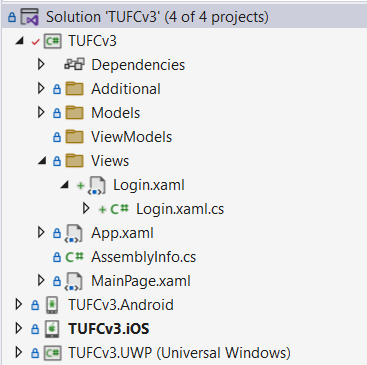
# Create Login Page

When the application opens

the first page to be displayed will be the page: Login.xaml

Create the content page Login.xaml

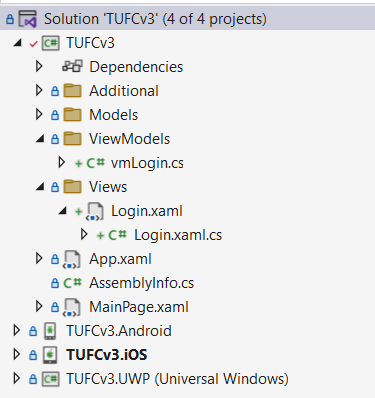
in TUFCv3.Views



All of the pages in thsis application will use the MVVM design pattern

In the folder TUFCv3.ViewModels

create the complementary class vmLogin.vm



To link the View Login and ViewModel class vmLogin

they need to be bound together.

This can be done either in C# code *(in the class Login.xaml.cs)*

or using XAML *(on the layout Login.xaml)*

For consistency I’ll bind all MVVM code using XAML.

#### Example – Bind ‘Login.xaml’ and ‘vmLogin.cs’

In Login.xaml add the namespace for vmLogin.cs

and bind vmLogin using the XAML command BindingContext

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

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="TUFCv3.Views.Login"

xmlns:ViewModels="clr-namespace:TUFCv3.ViewModels" >

<ContentPage.Content>

<StackLayout Padding="20">

<StackLayout.BindingContext>

<ViewModels:vmLogin/>

</StackLayout.BindingContext>

</StackLayout>

</ContentPage.Content>

</ContentPage>

*Next, Add XAML layout elements …*

## Add XAML Layout elements

Add the title *The Ultimate Fitness Companion*

and a label and entry

for the users email and password

#### Example – Add XAML for the Login page including title, email and password

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

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="TUFCv3.Views.Login"

xmlns:ViewModels="clr-namespace:TUFCv3.ViewModels" >

<ContentPage.Content>

<StackLayout Padding="20">

<StackLayout.BindingContext>

<ViewModels:vmLogin/>

</StackLayout.BindingContext>

<Label Text="The Ultimate &#x0a; Fitness Companion"

FontSize="Large" HorizontalTextAlignment="Center"/>

<Grid Padding="0,20">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="0.3\*"/>

<ColumnDefinition Width="0.7\*"/>

</Grid.ColumnDefinitions>

<Label Text="Email:" Grid.Row="0" Grid.Column="0" HorizontalTextAlignment="End" VerticalTextAlignment="Center" Padding="10" />

<Label Text="Password:" Grid.Row="1" Grid.Column="0" HorizontalTextAlignment="End" VerticalTextAlignment="Center" Padding="10" />

<Entry Text="{Binding user.email}" Grid.Row="0" Grid.Column="1"/>

<Entry Text="{Binding user.password}" Grid.Row="1" Grid.Column="1" IsPassword="True" />

</Grid>

</StackLayout>

</ContentPage.Content>

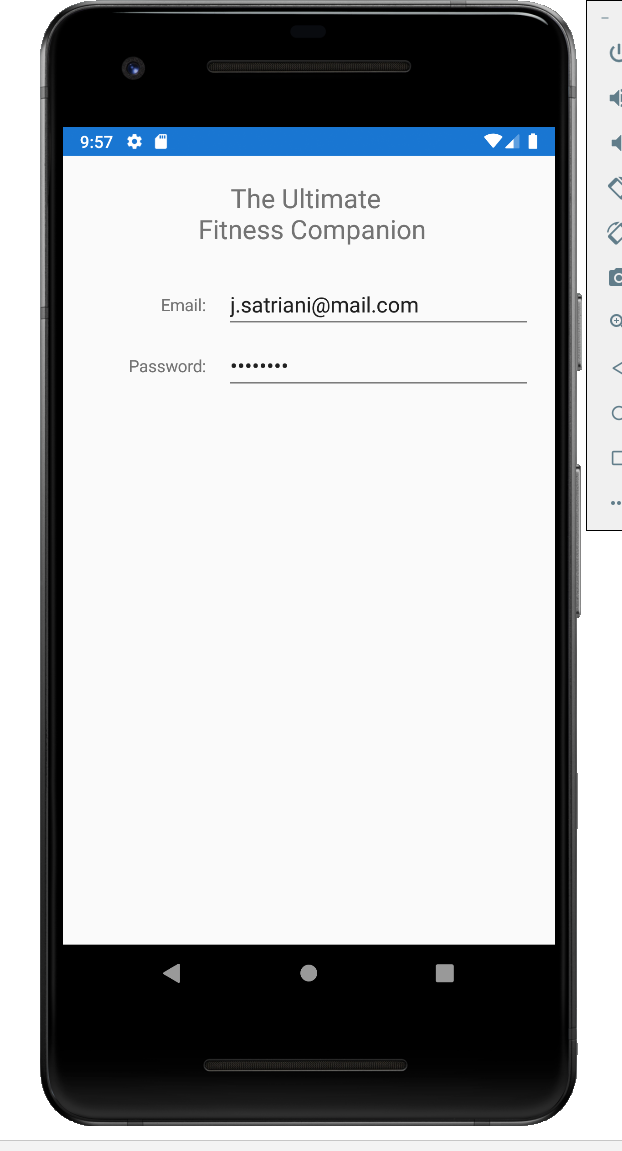
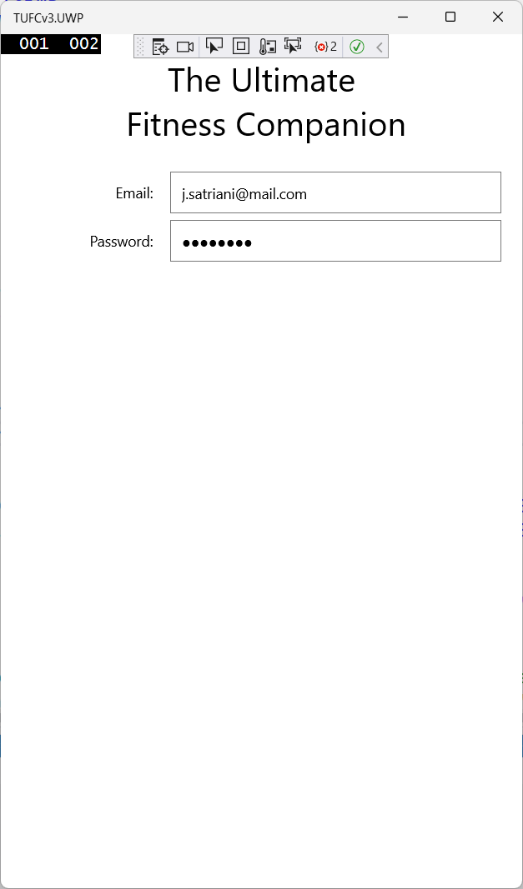
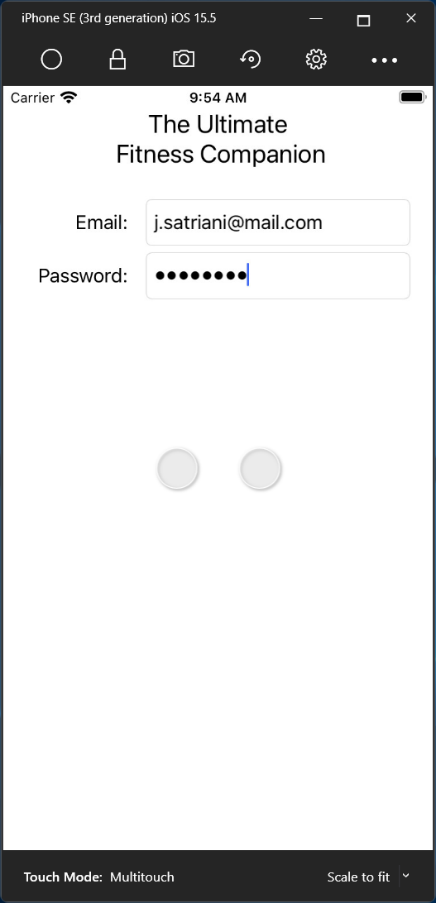
</ContentPage>

#### Test – Login page ‘Title, email and password’ are working

On the View Login.xaml test the layout of the title

and labels and entries for the user’s email and password – okay

*Android iOS UWP*



*Next, Add an Image …*

## Add an Image

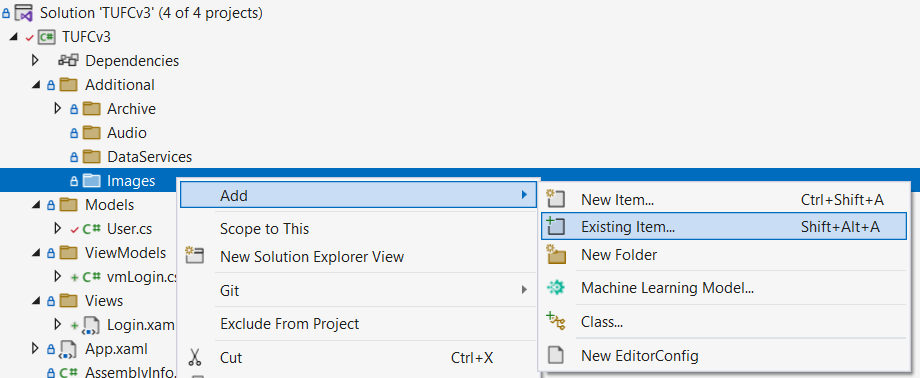
Add the image Flex

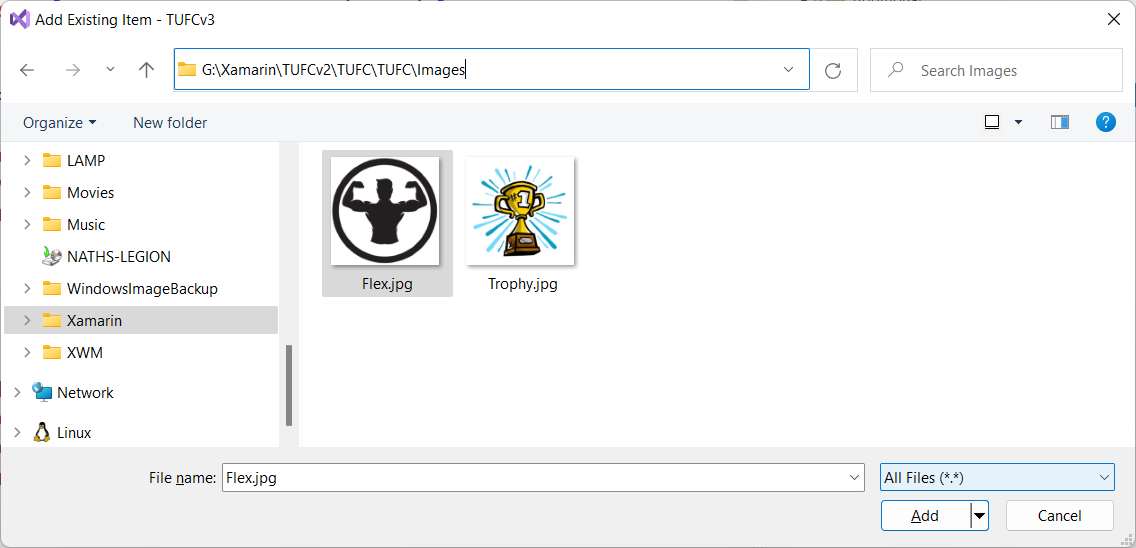
to the View Login.xaml

## Add an image to the solution

Add the image Flex

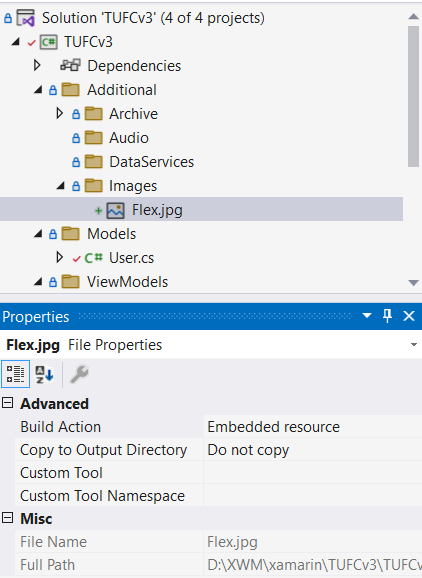
to the folder TUFCv3.Images





In the images Properties

set it’s Build Action to Embedded Resource



*Next, Add the image to the View …*

## Add the image to the View

Add the image Flex.jpg to the View Login.xaml

by adding an Image element

#### Example, Step1 - Add the image ‘Flex.jpg’ (XAML)

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

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="TUFCv3.Views.Login"

xmlns:ViewModels="clr-namespace:TUFCv3.ViewModels" >

<ContentPage.Content>

<StackLayout Padding="20">

<StackLayout.BindingContext>

<ViewModels:vmLogin/>

</StackLayout.BindingContext>

<Label Text="The Ultimate &#x0a; Fitness Companion"

FontSize="Large" HorizontalTextAlignment="Center"/>

<Image x:Name="flex" Aspect="AspectFit" HeightRequest="250" WidthRequest="250" Margin="0,25,0,10" />

<Grid Padding="0,20">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="0.3\*"/>

<ColumnDefinition Width="0.7\*"/>

</Grid.ColumnDefinitions>

<Label Text="Email:" Grid.Row="0" Grid.Column="0" HorizontalTextAlignment="End" VerticalTextAlignment="Center" Padding="10" />

<Label Text="Password:" Grid.Row="1" Grid.Column="0" HorizontalTextAlignment="End" VerticalTextAlignment="Center" Padding="10" />

<Entry Text="{Binding user.email}" Grid.Row="0" Grid.Column="1"/>

<Entry Text="{Binding user.password}" Grid.Row="1" Grid.Column="1" IsPassword="True" />

</Grid>

<!--<Button Text="Submit" HorizontalOptions="Center" HeightRequest="40" WidthRequest="250" Margin="5"/>

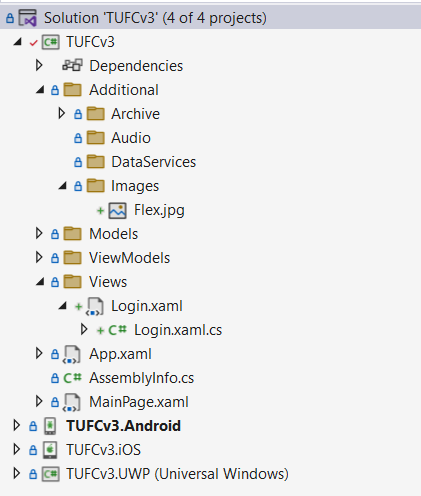
<Button Text="New User" HorizontalOptions="Center" HeightRequest="40" WidthRequest="250" Margin="5"/>-->

</StackLayout>

</ContentPage.Content>

</ContentPage>

#### Example, Step2 - Add the image ‘Flex.jpg’ (C# Code)

In Login.xaml.cs, add a call from the constructor

to a new method, called AddImages()

using Xamarin.Forms;

using Xamarin.Forms.Xaml;

namespace TUFCv3.Views

{

[XamlCompilation(XamlCompilationOptions.Compile)]

public partial class Login : ContentPage

{

public Login()

{

InitializeComponent();

AddImages();

}

}

}

In the method AddImages()

link xaml’s x:Name flex

to the image Flex.jpg

located in the folder TUFCv3.Images

using Xamarin.Forms;

using Xamarin.Forms.Xaml;

namespace TUFCv3.Views

{

[XamlCompilation(XamlCompilationOptions.Compile)]

public partial class Login : ContentPage

{

public Login()

{

InitializeComponent();

AddImages();

}

void AddImages()

{

// Set the image's source to the path of the image

// and add the View as the second parameter ie: 'typeof(Login)'

flex.Source = ImageSource.FromResource("TUFCv3.Additional.Images.Flex.jpg", typeof(Login));

}

}

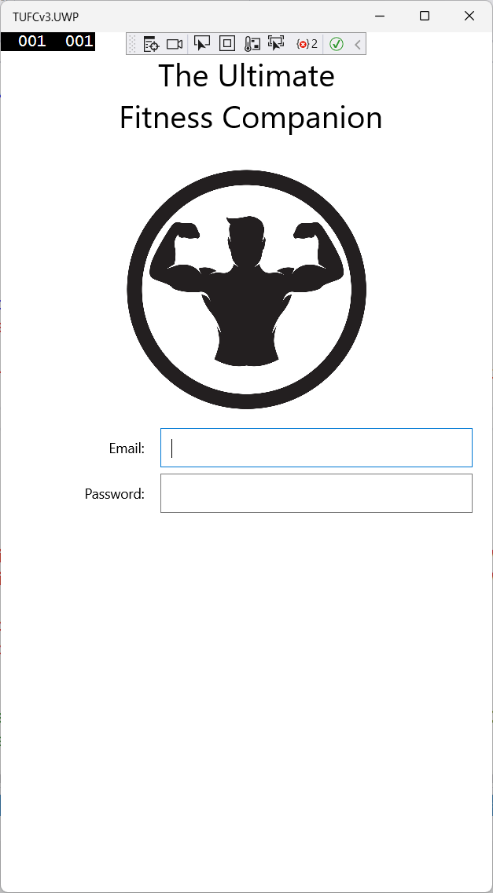
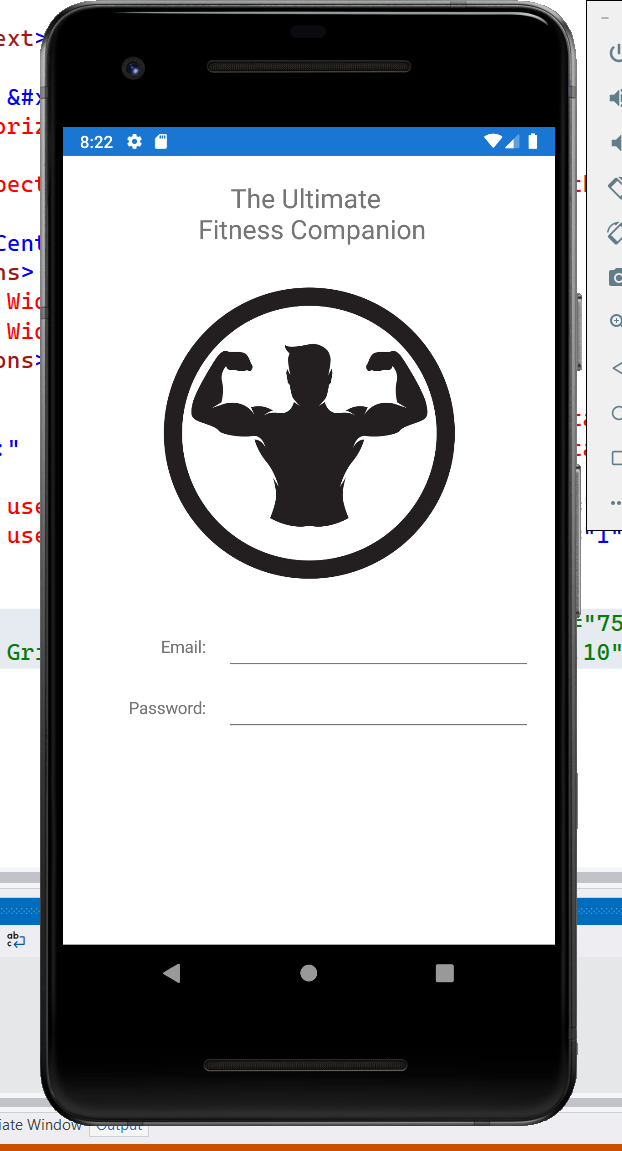
}

*Next, Text the app …*

#### Test – Check the image Flex.jpg is displayed correctly

Check the image Flex.jpg is displayed correctly – okay

*Android iOS UWP*



NOTE: Android’s default background color is grey.

I set this to white *(using the setting BackgroundColor)*

to match the image’s background.

*Next, Adding MVVM buttons …*

# Adding MVVM buttons

Add the buttons Login and New User

to the view Login.xaml

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

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="TUFCv3.Views.Login"

xmlns:ViewModels="clr-namespace:TUFCv3.ViewModels" >

<ContentPage.Content>

<StackLayout BackgroundColor="White" Padding="20" >

<StackLayout.BindingContext>

<ViewModels:vmLogin/>

</StackLayout.BindingContext>

<Label Text="The Ultimate &#x0a; Fitness Companion"

FontSize="Large" HorizontalTextAlignment="Center"/>

<Image x:Name="flex" Aspect="AspectFit" HeightRequest="250" WidthRequest="250" Margin="0,25" />

<Grid HorizontalOptions="CenterAndExpand" Padding="0,0,0,25">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="0.3\*"/>

<ColumnDefinition Width="0.7\*"/>

</Grid.ColumnDefinitions>

<Label Text="Email:" Grid.Row="0" Grid.Column="0" HorizontalTextAlignment="End" VerticalTextAlignment="Center" Padding="10" />

<Label Text="Password:" Grid.Row="1" Grid.Column="0" HorizontalTextAlignment="End" VerticalTextAlignment="Center" Padding="10" />

<Entry Text="{Binding user.email}" Grid.Row="0" Grid.Column="1"/>

<Entry Text="{Binding user.password}" Grid.Row="1" Grid.Column="1" IsPassword="True" />

</Grid>

0

<Button Text="Login" Margin="75,10" />

<Button Text="New User" Margin="75,10" />

</StackLayout>

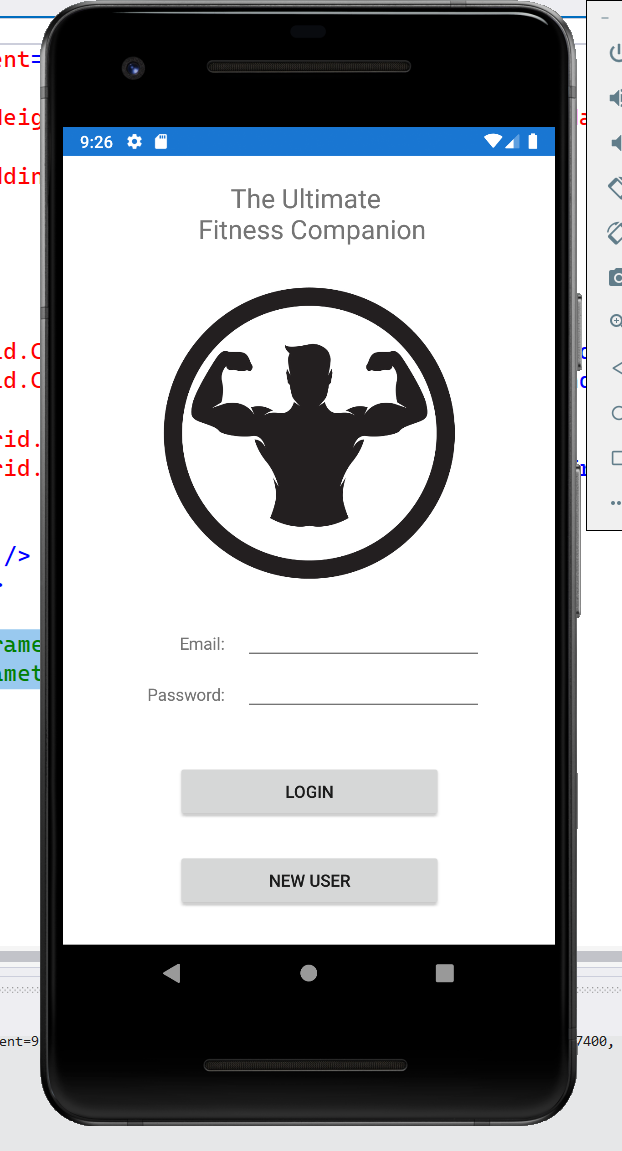
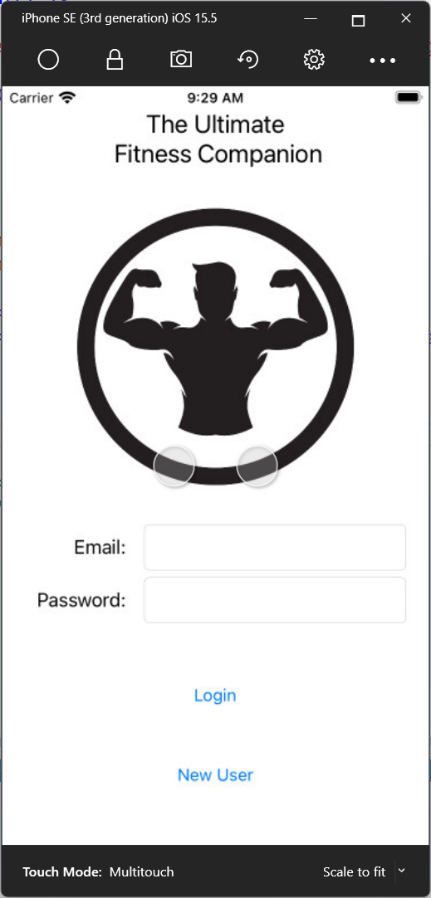
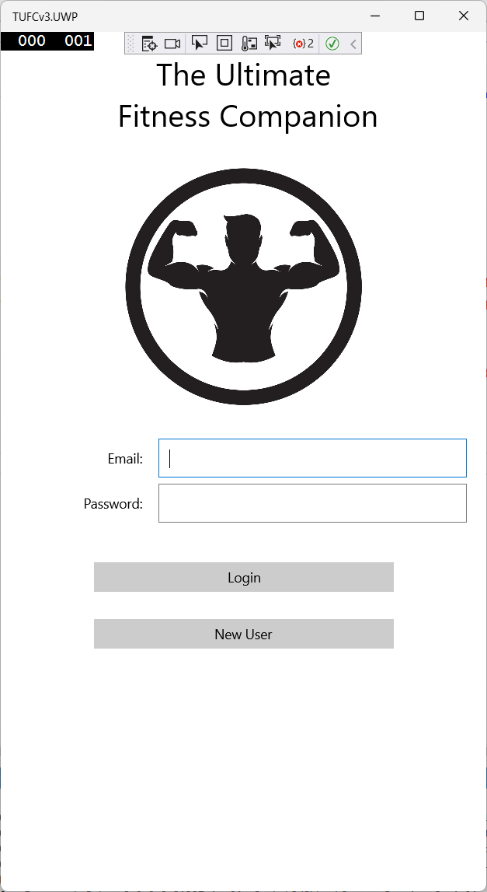
</ContentPage.Content>

</ContentPage>

#### Test – Check the buttons ‘Login’ and ‘NewUser’ display correctly

Check the buttons Login and NewUser display correctly – okay

*Android iOS Windows*



*Next, Add the pages MainMenu and New User …*

## Add the Views ‘MainMenu’ and ‘NewUser’

Before writing code for the buttons

add the ContentViews: MainMenu.xaml and NewUser.xaml

which the buttons will navigate to.

#### MainMenu

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

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="TUFCv3.Views.MainMenu">

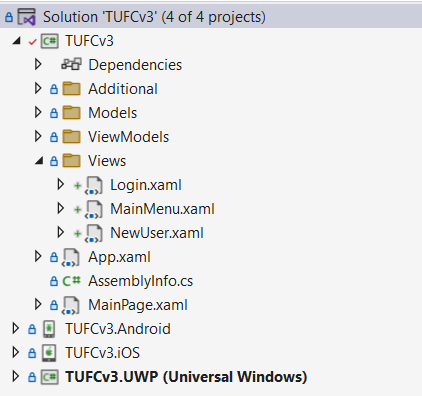
<ContentPage.Content>

<StackLayout>

<Label Text="MainMenu.xaml"

VerticalOptions="CenterAndExpand"

HorizontalOptions="CenterAndExpand" />

 </StackLayout>

</ContentPage.Content>

</ContentPage>

#### NewUser

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

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="TUFCv3.Views.NewUser">

<ContentPage.Content>

<StackLayout>

<Label Text="NewUser.xaml"

VerticalOptions="CenterAndExpand"

HorizontalOptions="CenterAndExpand" />

</StackLayout>

</ContentPage.Content>

</ContentPage>

*Next, Write MVVM button code …*

## Write MVVM button code

When using the MVVM design pattern, button Clicked cannot be used

as ViewModel code is loosely couple to the View.

Instead, use a Command inside the button

The same Command = “{Binding cmdNavigation}” is used for both buttons.

CommandParameter=”{x:Type ***Destination***}” specifies which page to navigate to.

#### Example, Step1 – In XAML add ‘Command’ and ‘CommandParameter’ for navigation buttons

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

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="TUFCv3.Views.Login"

xmlns:ViewModels="clr-namespace:TUFCv3.ViewModels"

xmlns:Views="clr-namespace:TUFCv3.Views" >

<ContentPage.Content>

<StackLayout BackgroundColor="White" Padding="20" >

<StackLayout.BindingContext>

<ViewModels:vmLogin/>

</StackLayout.BindingContext>

<Label Text="The Ultimate &#x0a; Fitness Companion"

FontSize="Large" HorizontalTextAlignment="Center"/>

<Image x:Name="flex" Aspect="AspectFit" HeightRequest="250" WidthRequest="250" Margin="0,25" />

<Grid HorizontalOptions="CenterAndExpand" Padding="0,0,0,25">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="0.3\*"/>

<ColumnDefinition Width="0.7\*"/>

</Grid.ColumnDefinitions>

<Label Text="Email:" Grid.Row="0" Grid.Column="0" HorizontalTextAlignment="End" VerticalTextAlignment="Center" Padding="10" />

<Label Text="Password:" Grid.Row="1" Grid.Column="0" HorizontalTextAlignment="End" VerticalTextAlignment="Center" Padding="10" />

<Entry Text="{Binding user.email}" Grid.Row="0" Grid.Column="1"/>

<Entry Text="{Binding user.password}" Grid.Row="1" Grid.Column="1" IsPassword="True" />

</Grid>

<Button Text="Login" Command="{Binding cmdNavigation}" CommandParameter="{x:Type Views:MainMenu}" Margin="75,10" />

<Button Text="New User" Command="{Binding cmdNavigation}" CommandParameter="{x:Type Views:NewUser}" Margin="75,10" />

</StackLayout>

</ContentPage.Content>

</ContentPage>

#### Example, Step2 – In the ViewModel instantiate ‘ICommand’ and add the ‘Command’ definition

In ViewModel code, the same command definition

is used for all navigation buttons.

To do this:

* instantiate ICommand as cmdNavigation
* call the new method DefineCommand() from the constructor
* write the command definition cmdNavigation in the new method

using Xamarin.Forms;

using System;

using System.ComponentModel;

using System.Runtime.CompilerServices;

using System.Windows.Input;

using TUFCv3.Models;

namespace TUFCv3.ViewModels

{

public class vmLogin

{

public string email { get; set; }

public string password { get; set; }

public ICommand cmdNavigation { private set; get; }

// CONSTRUCTOR

public vmLogin()

{

User user = new User();

email = "j.satriani@mail.com";

password = "P@ssword";

DefineCommands();

}

void DefineCommands()

{

// When Login is pressed, create MainMenu and naviage to it.

cmdNavigation = new Command<Type>(

execute: async (Type selectedPage) =>

{

Page page = (Page)Activator.CreateInstance(selectedPage); // Create the page

await App.Current.MainPage.Navigation.PushModalAsync(page); // and navigate to it.

});

}

}

}

# Create the Xamarin object model ‘User’

User login details will be authenticated

by comparing them to user’s email address and password

stored on the database.

The credentials entered by the user

will stored in the User instance loginUser

and compared to data returned from the database

stored the User instance databaseUser

Before doing this the object model User

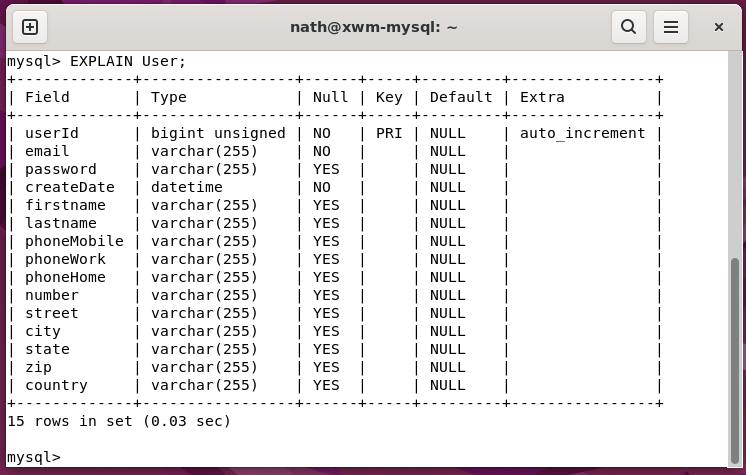
needs to be created.

The properties in the Xamarin object User

will complement fields in the table User

located in the database tufc on the MySQL database xwm-mysql

#### Image – Table ‘User’ located in the database ‘tufc’

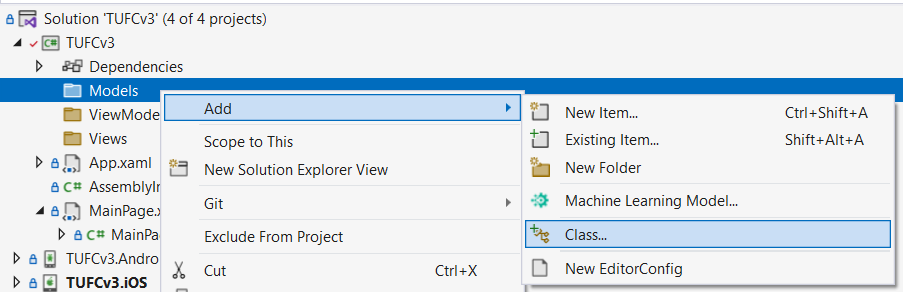


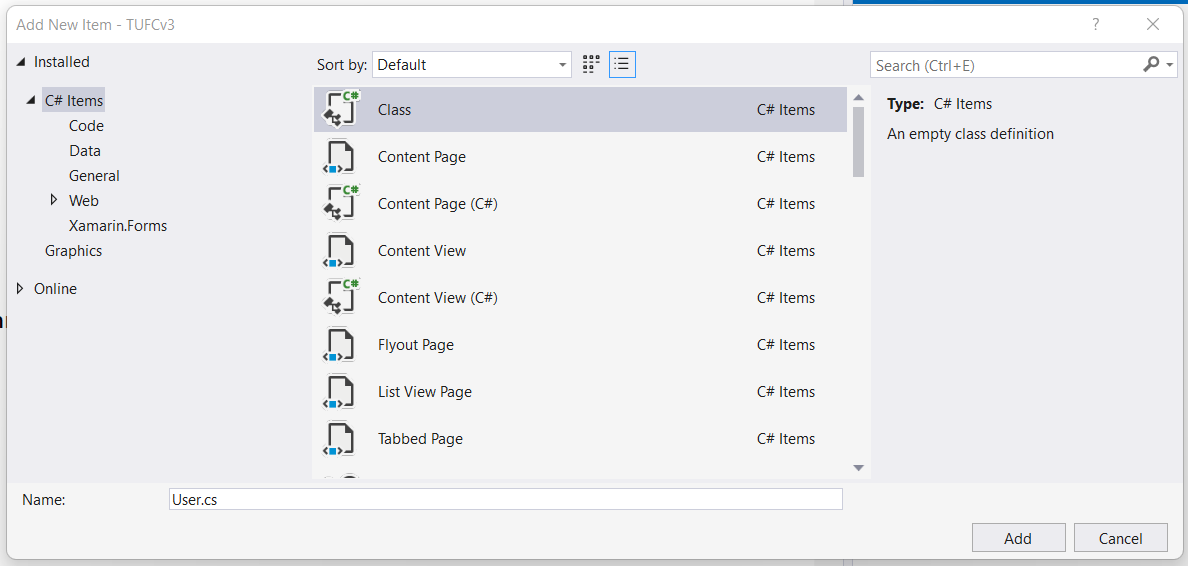
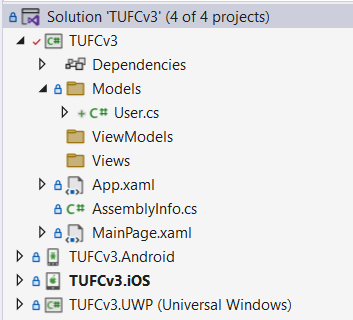
### Create the class User

Create the class User

in the folder tufc.Models

#### Example – Create private properties the Xamarin object User





### Create private properties for ‘User’

Create the private properties in Xamarin object User

to complement the fields table User

in the database tufc

using System;

namespace TUFCv3.Models

{

class User

{

private Int64 userId;

private string email;

private string password;

private DateTime createDate;

private string firstName;

private string lastName;

private string phoneMobile;

private string phoneWork;

private string phoneHome;

private string number;

private string street;

private string city;

private string state;

private string zip;

private string country;

}

}

### Create Public properties for ‘User’

Complement each private variable

with its public property.

Each time a public property is set *(for example from a* xaml Entry *field)*

the method SetProperty() is called.

SetProperty() updates the complementary private property,

and calls the method OnPropertyChanged()

to update the xaml page.

*Next, Create public properties for User …*

### Create public properties for ‘User’

To update private variable values

a complementary public property is required.

Public property setters call the method SetProperty()

to update the private variable and refresh the XAML page.

using System;

using System.Runtime.CompilerServices;

namespace TUFCv3.Models

{

class User

{

// Private Properties

private Int64 userId;

private string email;

private string password;

private DateTime createDate;

private string firstName;

private string lastName;

private string phoneMobile;

private string phoneWork;

private string phoneHome;

private string number;

private string street;

private string city;

private string state;

private string zip;

private string country;

// Public properties

// When setting public properties, call the method SetProperty()

// to update the complementry private variable and update the xaml page view.

public Int64 UserId

{

get { return userId; }

set { SetProperty(ref userId, value); }

}

public string Email

{

get { return email; }

set { SetProperty(ref email, value); }

}

public string Password

{

get { return password; }

set { SetProperty(ref password, value); }

}

public DateTime CreateDate

{

get { return createDate; }

set { SetProperty(ref createDate, value); }

}

public string FirstName

{

get { return firstName; }

set { SetProperty(ref firstName, value); }

}

public string LastName

{

get { return lastName; }

set { SetProperty(ref lastName, value); }

}

public string PhoneMobile

{

get { return phoneMobile; }

set { SetProperty(ref phoneMobile, value); }

}

public string PhoneWork

{

get { return phoneWork; }

set { SetProperty(ref phoneWork, value); }

}

public string PhoneHome

{

get { return phoneHome; }

set { SetProperty(ref phoneHome, value); }

}

public string Number

{

get { return number; }

set { SetProperty(ref number, value); }

}

public string Street

{

get { return street; }

set { SetProperty(ref street, value); }

}

public string City

{

get { return city; }

set { SetProperty(ref city, value); }

}

public string State

{

get { return state; }

set { SetProperty(ref state, value); }

}

public string Zip

{

get { return zip; }

set { SetProperty(ref zip, value); }

}

public string Country

{

get { return country; }

set { SetProperty(ref country, value); }

}

bool SetProperty<T>(ref T privateValue, T newValue, [CallerMemberName] string propertyName = null)

{

// TODO

return false;

}

}

}

### Create the method SetProperty()

There are several different property types in the object User, including:

* Int64
* string
* DateTime

When a public property updates

the complementary private variable also needs to automatically update.

The device’s screen also needs to update

which is done using an event.

To use an event in an object model

the event must be declared before public properties

#### Example – Declare the event property changed

private string phoneMobile; // private variable

// PropertyChanged

// An event handler, that updates bindings (including data on the device's screen)

// when a property changes.

public event PropertyChangedEventHandler PropertyChanged;

public string PhoneMobile // public property

{

get { return phoneMobile; }

set { SetProperty(ref phoneMobile, value); }

}

When a public property is updated

the method SetProperty() is called from the public property’s set{} *(shown above)*

which updates both the private variable and the devices screen.

#### Example – SetProperty()

// SetProperty()

// Update the private property, to match the public property

// then invoke the event handler PropertyChanged to update binding (including the screen).

// Arguments:

// 'privateValue' is the private property's current value

// 'newValue' is the public property's new value

// [CallerMemberName] 'propertyName' the calling public property's name.

bool SetProperty<T>(ref T privateValue, T newValue, [CallerMemberName] string propertyName = null)

{

if(Object.Equals(privateValue, newValue)) // If the stored and new values are the same

return false; // return without making any changes.

privateValue = newValue; // Update the private variable's value

// to match the public property's new value

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName)); // Invoke the event handler

// that updates property bindings

return true;

}

There are few interesting things happening in this method

which are described following sections.

*Next, Explain the method* SetProperty() *…*

### Generic methods

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/generics/generic-methods

Generics increase the reusability of the code

so, you don't need to write code to handle different data types.

SetProperty() is a generic

which allows different property types *(for example* int64, string *and* DateTime*)*

to all use the same method.

// SetProperty()

// Update the private property, to match the public property

// then invoke the event handler PropertyChanged to update binding (including the screen).

// Arguments:

// 'privateValue' is the private property's current value

// 'newValue' is the public property's new value

// [CallerMemberName] 'propertyName' the calling public property's name.

bool SetProperty<T>(ref T privateValue, T newValue, [CallerMemberName] string propertyName = null)

{

if(Object.Equals(privateValue, newValue)) // If the stored and new values are the same

return false; // return without making any changes.

privateValue = newValue; // Update the private variable's value

// to match the public property's new value

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName)); // Invoke the event handler

// that updates property bindings

return true;

}

### CallerMemberName

https://docs.microsoft.com/en-us/dotnet/api/system.runtime.compilerservices.callermembernameattribute?view=net-6.0

The attribute [CallerMemberName] *(highlighted above)*

allows the property name of the caller to be obtained.

You can find this attribute named CallerMemberNameAttribute

under the namespace System.Runtime.CompilerServices

### Invoke the event PropertyChanged

https://docs.microsoft.com/en-us/dotnet/api/system.componentmodel.propertychangedeventargs?view=net-6.0

To update property binding including the devices screen

the event PropertyChanged is raised when a public property changes.

The object PropertyChangedEventArgs()

specifies the name of the property that changed.

// SetProperty()

// Update the private property, to match the public property

// then invoke the event handler PropertyChanged to update binding (including the screen).

// Arguments:

// 'privateValue' is the private property's current value

// 'newValue' is the public property's new value

// [CallerMemberName] 'propertyName' the calling public property's name.

bool SetProperty<T>(ref T privateValue, T newValue, [CallerMemberName] string propertyName = null)

{

if(Object.Equals(privateValue, newValue)) // If the stored and new values are the same

return false; // return without making any changes.

privateValue = newValue; // Update the private variable's value

// to match the public property's new value

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName)); // Invoke the event handler

// that updates property bindings

return true;

}

### User object - complete

using System;

using System.Runtime.CompilerServices;

using System.ComponentModel;

namespace TUFCv3.Models

{

public class User : INotifyPropertyChanged

{

// PRIVATE Properties

private Int64 userId;

private string email;

private string password;

private DateTime createDate;

private string firstName;

private string lastName;

private string phoneMobile;

private string phoneWork;

private string phoneHome;

private string number;

private string street;

private string city;

private string state;

private string zip;

private string country;

// EVENTS

// PropertyChanged

// An event handler, that updates bindings (including data on the device's screen)

// when a property changes.

public event PropertyChangedEventHandler PropertyChanged;

// PUBLIC properties

// When setting public properties, call the method SetProperty() to update the complementry private variable

// which in turn calls the method OnPropertyChanged(), to update xmal page views.

public Int64 UserId

{

get { return userId; }

set { SetProperty(ref userId, value); }

}

public string Email

{

get { return email; }

set { SetProperty(ref email, value); }

}

public string Password

{

get { return password; }

set { SetProperty(ref password, value); }

}

public DateTime CreateDate

{

get { return createDate; }

set { SetProperty(ref createDate, value); }

}

public string FirstName

{

get { return firstName; }

set { SetProperty(ref firstName, value); }

}

public string LastName

{

get { return lastName; }

set { SetProperty(ref lastName, value); }

}

public string PhoneMobile

{

get { return phoneMobile; }

set { SetProperty(ref phoneMobile, value); }

}

public string PhoneWork

{

get { return phoneWork; }

set { SetProperty(ref phoneWork, value); }

}

public string PhoneHome

{

get { return phoneHome; }

set { SetProperty(ref phoneHome, value); }

}

public string Number

{

get { return number; }

set { SetProperty(ref number, value); }

}

public string Street

{

get { return street; }

set { SetProperty(ref street, value); }

}

public string City

{

get { return city; }

set { SetProperty(ref city, value); }

}

public string State

{

get { return state; }

set { SetProperty(ref state, value); }

}

public string Zip

{

get { return zip; }

set { SetProperty(ref zip, value); }

}

public string Country

{

get { return country; }

set { SetProperty(ref country, value); }

}

// SETPROPERTY()

// Update the private property, to match the public property

// then invoke the event handler PropertyChanged to update binding (including the screen).

// Arguments:

// 'privateValue' is the private property's current value

// 'newValue' is the public property's new value

// [CallerMemberName] 'propertyName' the calling public property's name.

bool SetProperty<T>(ref T privateValue, T newValue, [CallerMemberName] string propertyName = null)

{

if(Object.Equals(privateValue, newValue)) // If the stored and new values are the same

return false; // return without making any changes.

privateValue = newValue; // Update the private variable's value

// to match the public property's new value

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName)); // Invoke the event handler

// that updates property bindings

return true;

}

}

}

# Encryption using CBC

To encrypt the user’s password

I want to use an encryption process

that works on both the Xamarin app and the website.

I will use Cipher Block Chain *(CBC)* methods

that work in C# *(for Xamarin)* and PHP *(for the website)*

When a password is created it will be encrypted *(using CBC)*

then saved to the MySQL database xwm-mysql

In the future, when the user then logs in

the user enters their password

that is encrypted using the same algorithm.

The encrypted password the user entered during login

can then be compared with the encrypted password

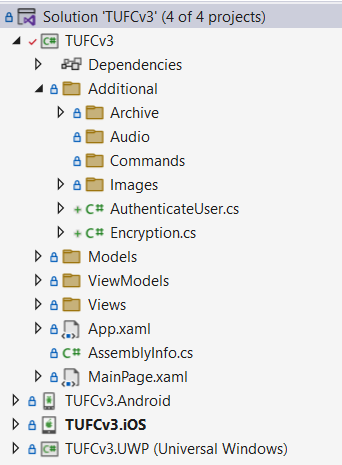
retrieved from the MySQL database.

If the encrypted passwords match, the user is authenticated

and the app navigates to MainView.xaml

## Create the class ‘Encryption’

Create the class TUFCv3.Additional.Encryption.cs



## Encryption code

When comparing the password the user enters

with the encrypted password saved on the database

it is encrypted using the same cipher, then compared.

This means the password saved on the database

never needs to be decrypted, keeping it secure.

However, while deciding on a suitable encryption method for C# and PHP

I needed to encrypt/decrypt on C# *(Xamarin)*

and compared it with output using PHP *(Website)*

and vice versa.

To do that I created complementary methods to encrypt and decrypt.

In total I created five methods in the class Encryption:

* Encryption() The class constructor
* CreateEncryptor() Sets up the cipher mode, ready for encoding/decoding
* EncryptString() Encrypts a string
* DecryptString() Decrypts a cipher
* TestEncryption() To test encryption is working, this method encodes then decodes a test string

*Next, Describe each of the methods in the class Encryption.cs …*

### Encryption constructor

The constructor for the class Encryption.cs

* creates a test password and initialization vector
* calls CreateEncryptor() to configure the cipher method
* calls TestEncryption to test encoding/decoding is working *(and is compatible with PHP encryption)*

using System;

using System.IO;

using System.Security.Cryptography;

using System.Text;

namespace TUFCv3.Additional

{

public class Encryption

{

Aes encryptor; // The encryption object, configured for CBC (includes the key and IV)

public Encryption()

{

string password = "password!";

string iv = "TopSecretVector!"; // On a live system 'iv' should be randomly generated and saved to a secure location.

CreateEncryptor(password, iv); // Create the object 'encryptor'

TestEncryption(); // Test the encryption method ('comment out', but do not delete)

}

}

}

### CreateEncryptor()

Specifies the cipher mode *(CBC)* for the object encryptor

and adds the key and iv

using System;

using System.IO;

using System.Security.Cryptography;

using System.Text;

namespace TUFCv3.Additional

{

public class Encryption

{

Aes encryptor; // The encryption object, configured for CBC (includes the key and IV)

public Encryption()

{

string password = "password!";

string iv = "TopSecretVector!"; // On a live system 'iv' should be randomly generated and saved to a secure location.

CreateEncryptor(password, iv); // Create the object 'encryptor'

TestEncryption(); // Test the encryption method ('comment out', but do not delete)

}

// CreateEncryptor()

// Create the object 'encryptor', that will be used to encode/decode messages

void CreateEncryptor(string password, string iv)

{

SHA256 sha256 = SHA256Managed.Create(); // Create a Secure Hashing Algorithm

byte[] key = sha256.ComputeHash(Encoding.ASCII.GetBytes(password)); // Create the hash key, using 'password'

byte[] ivBytes = Encoding.ASCII.GetBytes(iv); // Convert the 'iv' to a byte array

encryptor = Aes.Create(); // Create the Advanced Encryption Standard (AES) object 'encryptor'

encryptor.Mode = CipherMode.CBC; // Set encryption method to Cyclic Block Chain (CBC)

byte[] aesKey = new byte[32];

Array.Copy(key, 0, aesKey, 0, 32); // Copy the hashed password 'key' to the byte array 'aesKey'

encryptor.Key = aesKey; // Set 'aesKey' as the the encryption key

encryptor.IV = ivBytes; // and ivBytes as the initialization vector

}

}

}

*Next, Encryptstring() …*

### EncryptString()

Encrypt plain text to a cipher

using System;

using System.IO;

using System.Security.Cryptography;

using System.Text;

namespace TUFCv3.Additional

{

public class Encryption

{

Aes encryptor; // The encryption object, configured for CBC (includes the key and IV)

public Encryption()

{

string password = "password!";

string iv = "TopSecretVector!"; // On a live system 'iv' should be randomly generated and saved to a secure location.

CreateEncryptor(password, iv); // Create the object 'encryptor'

TestEncryption(); // Test the encryption method ('comment out', but do not delete)

}

// CreateEncryptor()

// Create the object 'encryptor', that will be used to encode/decode messages

void CreateEncryptor(string password, string iv)

{

SHA256 sha256 = SHA256Managed.Create(); // Create a Secure Hashing Algorithm

byte[] key = sha256.ComputeHash(Encoding.ASCII.GetBytes(password)); // Create the hash key, using 'password'

byte[] ivBytes = Encoding.ASCII.GetBytes(iv); // Convert the 'iv' to a byte array

encryptor = Aes.Create(); // Create the Advanced Encryption Standard (AES) object 'encryptor'

encryptor.Mode = CipherMode.CBC; // Set encryption method to Cyclic Block Chain (CBC)

byte[] aesKey = new byte[32];

Array.Copy(key, 0, aesKey, 0, 32); // Copy the hashed password 'key' to the byte array 'aesKey'

encryptor.Key = aesKey; // Set 'aesKey' as the the encryption key

encryptor.IV = ivBytes; // and ivBytes as the initialization vector

}

// EncryptString()

// Encrypt a message

public string EncryptString(string plainText)

{

MemoryStream memoryStream = new MemoryStream(); // Memory stream

ICryptoTransform aesEncryptor = encryptor.CreateEncryptor(); // Set the transform to encrypt

CryptoStream cryptoStream = new CryptoStream(memoryStream, aesEncryptor, CryptoStreamMode.Write); // Encryption stream

byte[] plainBytes = Encoding.ASCII.GetBytes(plainText); // Convert the original string to the byte array 'plainbytes'

cryptoStream.Write(plainBytes, 0, plainBytes.Length); // Encrypt the array using CBC

cryptoStream.FlushFinalBlock();

byte[] cipherBytes = memoryStream.ToArray();

memoryStream.Close();

cryptoStream.Close();

string cipherText = Convert.ToBase64String(cipherBytes, 0, cipherBytes.Length); // Convert the encrypted byte array to a string

return cipherText;

}

}

}

*Next, DecryptString() …*

### DecryptString()

This is not required for authentication.

However, it has been useful while testing

and comparing C# and PHP encryption.

using System;

using System.IO;

using System.Security.Cryptography;

using System.Text;

namespace TUFCv3.Additional

{

public class Encryption

{

Aes encryptor; // The encryption object, configured for CBC (includes the key and IV)

public Encryption()

{

string password = "password!";

string iv = "TopSecretVector!"; // On a live system 'iv' should be randomly generated and saved to a secure location.

CreateEncryptor(password, iv); // Create the object 'encryptor'

TestEncryption(); // Test the encryption method ('comment out', but do not delete)

}

// CreateEncryptor()

// Create the object 'encryptor', that will be used to encode/decode messages

void CreateEncryptor(string password, string iv)

{

SHA256 sha256 = SHA256Managed.Create(); // Create a Secure Hashing Algorithm

byte[] key = sha256.ComputeHash(Encoding.ASCII.GetBytes(password)); // Create the hash key, using 'password'

byte[] ivBytes = Encoding.ASCII.GetBytes(iv); // Convert the 'iv' to a byte array

encryptor = Aes.Create(); // Create the Advanced Encryption Standard (AES) object 'encryptor'

encryptor.Mode = CipherMode.CBC; // Set encryption method to Cyclic Block Chain (CBC)

byte[] aesKey = new byte[32];

Array.Copy(key, 0, aesKey, 0, 32); // Copy the hashed password 'key' to the byte array 'aesKey'

encryptor.Key = aesKey; // Set 'aesKey' as the the encryption key

encryptor.IV = ivBytes; // and ivBytes as the initialization vector

}

// EncryptString()

// Encrypt a message

public string EncryptString(string plainText)

{

MemoryStream memoryStream = new MemoryStream(); // Memory stream

ICryptoTransform aesEncryptor = encryptor.CreateEncryptor(); // Set the transform to encrypt

CryptoStream cryptoStream = new CryptoStream(memoryStream, aesEncryptor, CryptoStreamMode.Write); // Encryption stream

byte[] plainBytes = Encoding.ASCII.GetBytes(plainText); // Convert the original string to the byte array 'plainbytes'

cryptoStream.Write(plainBytes, 0, plainBytes.Length); // Encrypt the array using CBC

cryptoStream.FlushFinalBlock();

byte[] cipherBytes = memoryStream.ToArray();

memoryStream.Close();

cryptoStream.Close();

string cipherText = Convert.ToBase64String(cipherBytes, 0, cipherBytes.Length); // Convert the encrypted byte array to a string

return cipherText;

}

// DecryptString()

string DecryptString(string cipherText)

{

MemoryStream memoryStream = new MemoryStream(); // Memory stream

ICryptoTransform aesDecryptor = encryptor.CreateDecryptor(); // Set the transform to decrypt

CryptoStream cryptoStream = new CryptoStream(memoryStream, aesDecryptor, CryptoStreamMode.Write); // Encryption stream

string plainText = String.Empty; // Will contain the decrypted plain text

try

{

byte[] cipherBytes = Convert.FromBase64String(cipherText); // Convert the original string to the byte array 'cipherBytes'

cryptoStream.Write(cipherBytes, 0, cipherBytes.Length); // Decrypt the cipherBytes using CBC

cryptoStream.FlushFinalBlock();

byte[] plainBytes = memoryStream.ToArray(); // Write the decrypted bytes to an array

plainText = Encoding.ASCII.GetString(plainBytes, 0, plainBytes.Length); // Convert the decrypted byte array to a string

}

finally

{

memoryStream.Close();

cryptoStream.Close();

}

return plainText;

}

}

}

*Next, TestEncryption() …*

### TestEncryption()

Test encrypt/decrypt methods

to make sure they’re working.

using System;

using System.IO;

using System.Security.Cryptography;

using System.Text;

namespace TUFCv3.Additional

{

public class Encryption

{

Aes encryptor; // The encryption object, configured for CBC (includes the key and IV)

public Encryption()

{

string password = "password!";

string iv = "TopSecretVector!"; // On a live system 'iv' should be randomly generated and saved to a secure location.

CreateEncryptor(password, iv); // Create the object 'encryptor'

TestEncryption(); // Test the encryption method ('comment out', but do not delete)

}

// CreateEncryptor()

// Create the object 'encryptor', that will be used to encode/decode messages

void CreateEncryptor(string password, string iv)

{

SHA256 sha256 = SHA256Managed.Create(); // Create a Secure Hashing Algorithm

byte[] key = sha256.ComputeHash(Encoding.ASCII.GetBytes(password)); // Create the hash key, using 'password'

byte[] ivBytes = Encoding.ASCII.GetBytes(iv); // Convert the 'iv' to a byte array

encryptor = Aes.Create(); // Create the Advanced Encryption Standard (AES) object 'encryptor'

encryptor.Mode = CipherMode.CBC; // Set encryption method to Cyclic Block Chain (CBC)

byte[] aesKey = new byte[32];

Array.Copy(key, 0, aesKey, 0, 32); // Copy the hashed password 'key' to the byte array 'aesKey'

encryptor.Key = aesKey; // Set 'aesKey' as the the encryption key

encryptor.IV = ivBytes; // and ivBytes as the initialization vector

}

// EncryptString()

// Encrypt a message

public string EncryptString(string plainText)

{

MemoryStream memoryStream = new MemoryStream(); // Memory stream

ICryptoTransform aesEncryptor = encryptor.CreateEncryptor(); // Set the transform to encrypt

CryptoStream cryptoStream = new CryptoStream(memoryStream, aesEncryptor, CryptoStreamMode.Write); // Encryption stream

byte[] plainBytes = Encoding.ASCII.GetBytes(plainText); // Convert the original string to the byte array 'plainbytes'

cryptoStream.Write(plainBytes, 0, plainBytes.Length); // Encrypt the array using CBC

cryptoStream.FlushFinalBlock();

byte[] cipherBytes = memoryStream.ToArray();

memoryStream.Close();

cryptoStream.Close();

string cipherText = Convert.ToBase64String(cipherBytes, 0, cipherBytes.Length); // Convert the encrypted byte array to a string

return cipherText;

}

// DecryptString()

string DecryptString(string cipherText)

{

MemoryStream memoryStream = new MemoryStream(); // Memory stream

ICryptoTransform aesDecryptor = encryptor.CreateDecryptor(); // Set the transform to decrypt

CryptoStream cryptoStream = new CryptoStream(memoryStream, aesDecryptor, CryptoStreamMode.Write); // Encryption stream

string plainText = String.Empty; // Will contain the decrypted plain text

try

{

byte[] cipherBytes = Convert.FromBase64String(cipherText); // Convert the original string to the byte array 'cipherBytes'

cryptoStream.Write(cipherBytes, 0, cipherBytes.Length); // Decrypt the cipherBytes using CBC

cryptoStream.FlushFinalBlock();

byte[] plainBytes = memoryStream.ToArray(); // Write the decrypted bytes to an array

plainText = Encoding.ASCII.GetString(plainBytes, 0, plainBytes.Length); // Convert the decrypted byte array to a string

}

finally

{

memoryStream.Close();

cryptoStream.Close();

}

return plainText;

}

// TestEncryption() - only used during testing

void TestEncryption()

{

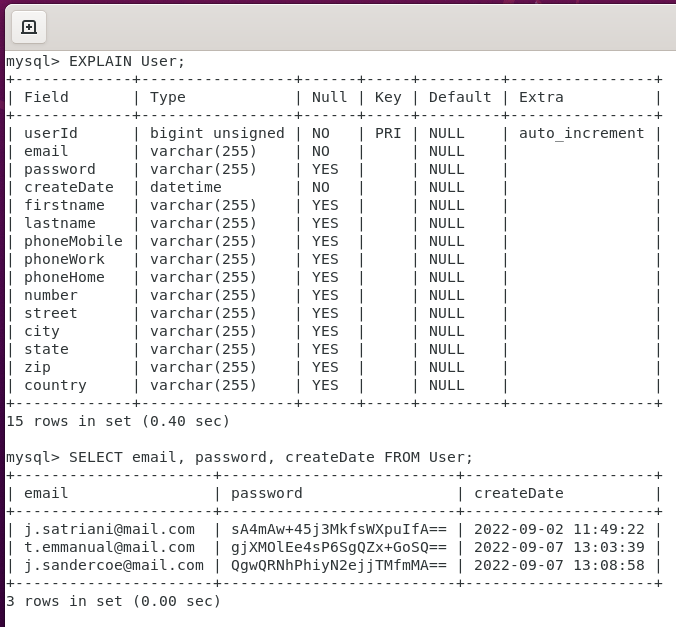
string encryptedMessage = this.EncryptString("Message to encode/decode"); // Encrypt a message

string decryptedMessage = this.DecryptString(encryptedMessage); // Decrypt the message

}

}

}



# Introduce SOLID to the login classes

I refactor will using the SOLID design pattern

to make the application more robust and easier to modify.

## SOLID Summary

### Single Responsibility Principle (SRP)

<https://www.youtube.com/watch?v=5RwhyZnVRS8&list=PLAaFb7UfyShCoS246UzZJNEiXuD8bg02e&index=1>

“*There should only be one reason to change a class”*

### Open Closed Principle (OCP)

https://www.youtube.com/watch?v=VFlk43QGEgc&list=PLAaFb7UfyShCoS246UzZJNEiXuD8bg02e&index=2

*“Objects should be open to extension*

*but closed to modification”*

#### Liskov Substitution Principle

https://www.youtube.com/watch?v=-3UXq2krhyw&list=PLAaFb7UfyShCoS246UzZJNEiXuD8bg02e&index=3

#### Interface Segregation Principle

<https://www.youtube.com/watch?v=y1JiMGP51NE&list=PLAaFb7UfyShCoS246UzZJNEiXuD8bg02e&index=4>

#### Dependency Inversion Principle

https://www.youtube.com/watch?v=NnZZMkwI6KI&list=PLAaFb7UfyShCoS246UzZJNEiXuD8bg02e&index=5

## SOLID ‘S’

The ‘O’ in solid is for:

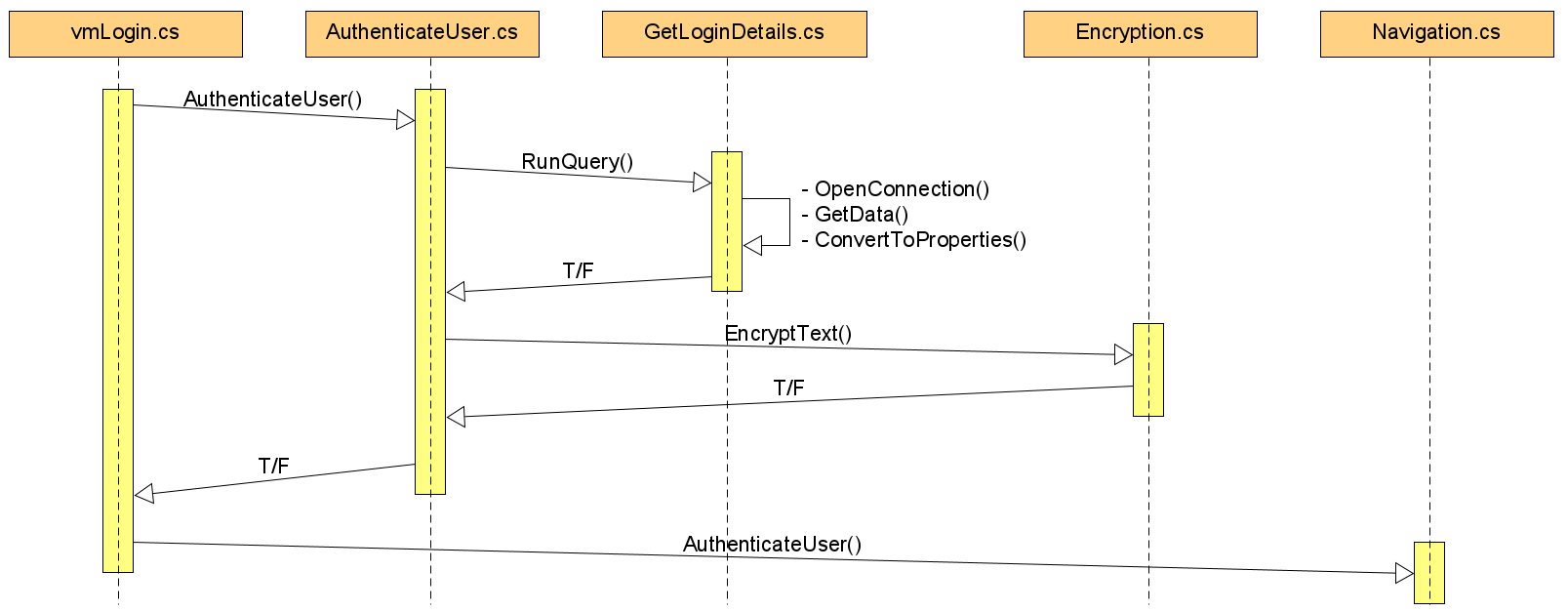
***Single Responsibility***

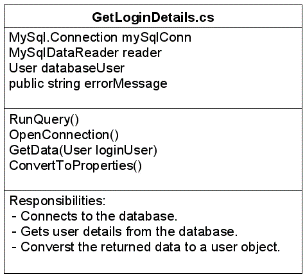
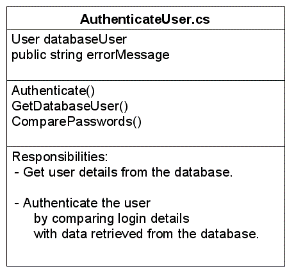
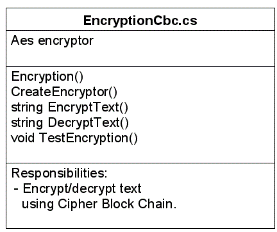
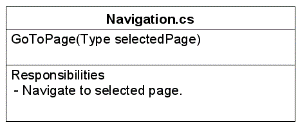
I have refactored each class used during the login process

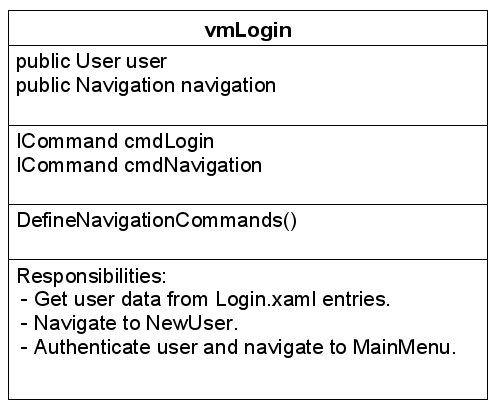
into methods which only have a single responsibility *(the ‘S’ in SOLID)*

This makes the code easy to understand

and allows each method to be unit tested.





I have also named each method with a descriptive *(but generic)* name

so interfaces can be extracted and used in the next step *(‘O’)*

in the SOLID design pattern.

*Next, SOLID ‘O’ …*

## SOLID ‘O’

The ‘O’ in solid is for:

***Open to extension***

***but closed to modification.***

This means objects in a production environment

should not be modified *(closed to modification)*

however, they can be extended.

For example, the class User may have child objects

such as technician, manager etc

Each child object contains all of the properties of the object User

and may also include additional properties and overridden methods

to extend their functionality.

An example of where this will be used in TUFCv3

includes creating the generic interface GetMySqlData

to get data from a database

however, various child objects will specify what data will be returned.

To make later steps in SOLID design pattern easier

I will create an interface for each object type

and wherever an object instance is created *(‘an object is newed up’)*

I will replace the type with its interface.

## Extract interfaces for types in vmLogin

Starting with the class vmLogin

extract an interface for each type.

*Next, Extract an interface for the type ‘User’ …*

### Extract an interface for the type ‘User’

The first type that appears in the class vmLogin

is User

using Xamarin.Forms;

using System;

using System.Windows.Input;

using TUFCv3.Models;

using TUFCv3.Additional;

namespace TUFCv3.ViewModels

{

public class vmLogin

{

// Properties

public User user { set; get; } // Login user

public Navigation navigation = new Navigation(); // Navigates to another page

// Constructor

public vmLogin()

{

user = new User(); // Instantiate the object 'user' (which is bound to Login.xaml entry fields)

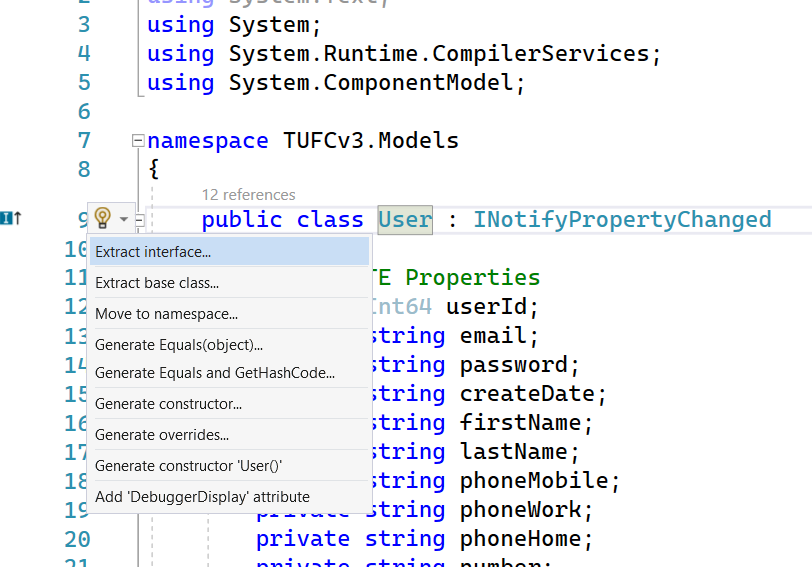
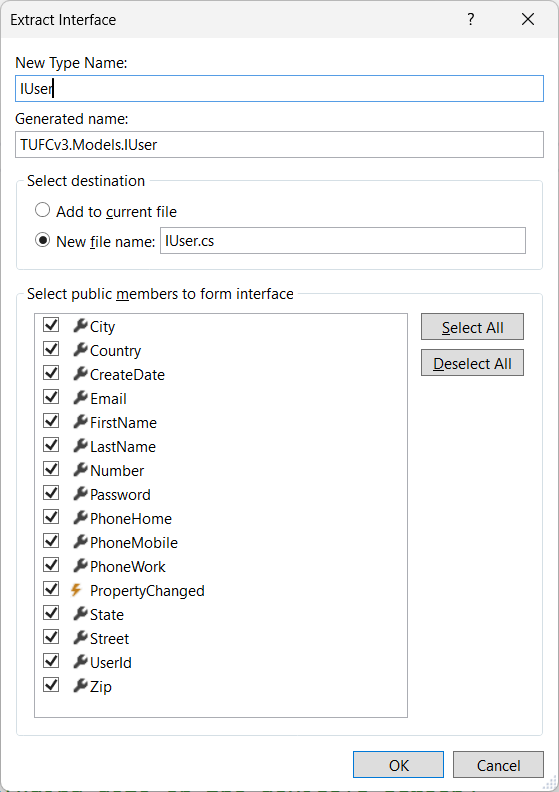
DefineNavigationCommands(); // Create navigation commands for button click events

}

// …

To create an interface for User

* open the class User *(in Models)*
* Ctrl + . the class name
* select Extract interface …
* click OK



Now that the User interface has been created

the type User in vmLogin

can be replaced by the interface IUser

public class vmLogin

{

// Properties

public User IUser loginUser { set; get; } // Login loginUser

public Navigation navigation = new Navigation(); // Navigates to another page

// Constructor

public vmLogin()

{

loginUser = new User(); // Instantiate the object 'loginUser' (which is bound to Login.xaml entry fields)

DefineNavigationCommands(); // Create navigation commands for button click events

}

Changing User to IUser causes errors

in other classes where loginUser is referenced.

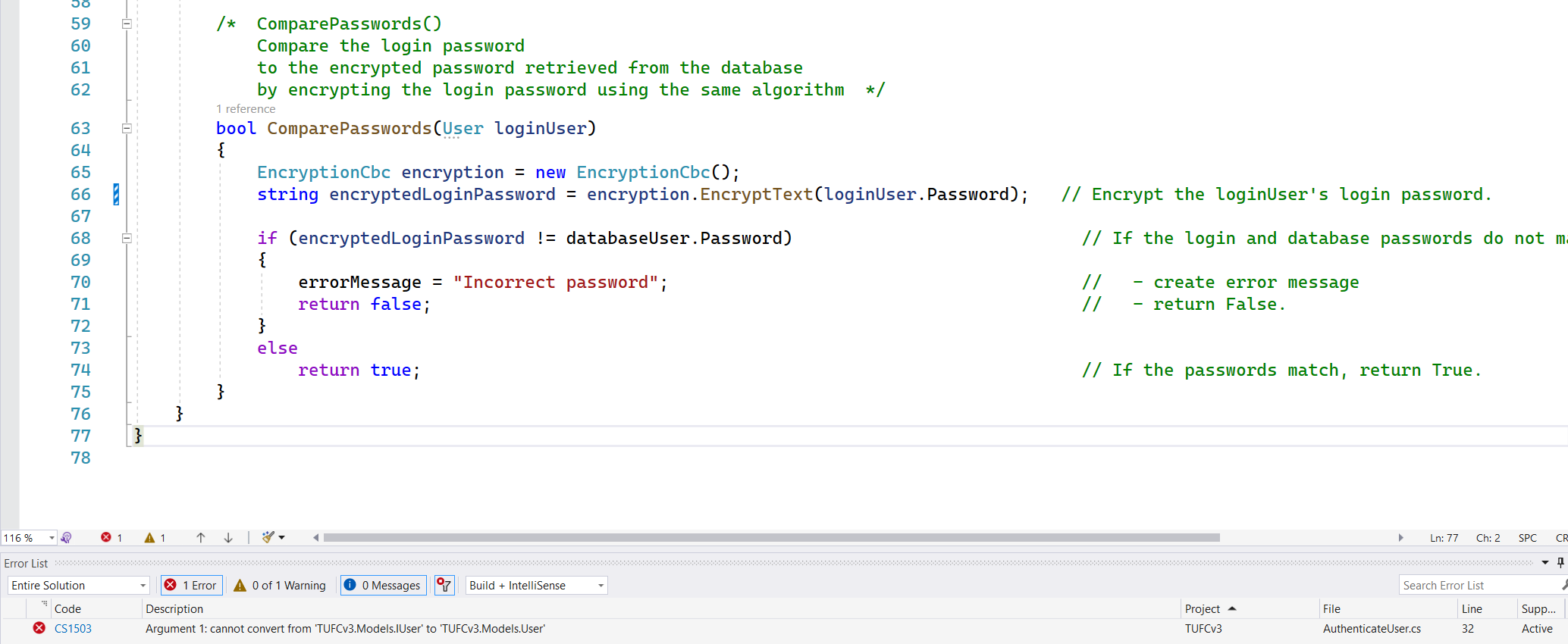
Everywhere it’s referenced it must be updated

from User to IUser

A first glance this may appear to be a big task

but Visual Studio highlights each place it needs to update

making it a simple task.



#### Test the App – Replacing ‘User’ with ‘IUser’

Replace all instances of User loginUser with IUser

Test the app - okay

I recommend testing and saving the application

before and after converting each type’s definition

to an interface.

## Converting other Instances

The same process must be used for every object instantiation.

I’ll go through each class

creating an interface for each object type

then replace each instantiation of an object

with an instance of an interface.

### Create interfaces in vmLogin

Continuing with the class vmLogin

the next object type is Navigation

The class Navigation is a small.

It is designed to reduce the clutter in ViewModel classes

by separating out navigation code.

using System;

using System.Threading.Tasks;

using Xamarin.Forms;

namespace TUFCv3.Additional.Navigation

{

public class Navigation

{

/\* GoToPage()

Naviagate to a new page \*/

public async Task GoToPage(Type selectedPage)

{

Page page = (Page)Activator.CreateInstance(selectedPage); // Create the page

await Application.Current.MainPage.Navigation.PushModalAsync(page); // and navigate to it.

}

}

}

If I create an interface for the class Navigation

its name would logically be INavigation

which would conflict with Xamarin.Forms.INavigation

Although this could be done

whenever I instantiate an INavigation object

I would have to use the entire path Additional.Navigation.INavigation

which is wordy and messy.

Instead, I renamed my navigation class from Navigation to Navigate

which removes any naming conflicts

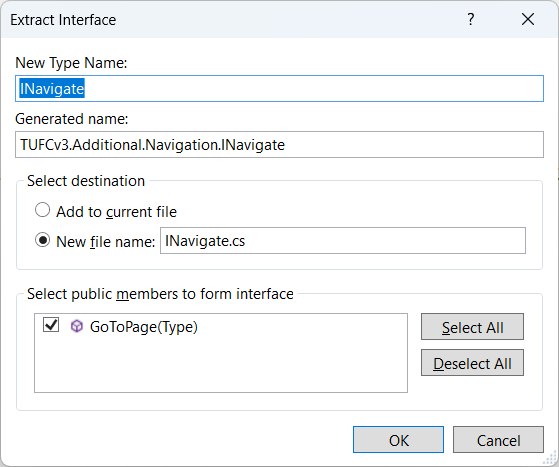
when interface INavigate is created.

To extract an interface for the renamed class Navigate.cs

press Ctrl + . on the class name

select Extract Interface

then click OK





Replace the original instantiated object Navigation navigation

with the interface INavigate navigate

using Xamarin.Forms;

using System;

using System.Windows.Input;

using TUFCv3.Additional;

using TUFCv3.Models.Users;

using TUFCv3.Additional.Navigation;

namespace TUFCv3.ViewModels

{

public class vmLogin

{

// Properties

public IUser loginUser { set; get; } // Login loginUser

public INavigate navigate = new Navigate(); // Navigates to another page

// Constructor

public vmLogin()

{

loginUser = new User(); // Instantiate the object 'loginUser' (which is bound to Login.xaml entry fields)

DefineNavigationCommands(); // Create navigate commands for button click events

}

// …

#### Test the app – Replace ‘Navigation’ with ‘INavigate’

Replace the instantiation of Navigation with INavigate

in vmLogin and test – okay

## Create interfaces in ‘AuthenticateUser.cs’

### Create ‘IGetMySqlData’

The method GetDatabaseUser()

located in the class AuthenticateUser

instantiates the object GetLoginDetails

namespace TUFCv3.Additional

{

[XamlCompilation(XamlCompilationOptions.Compile)]

public class AuthenticateUser

{

private IUser databaseUser = new User(); // User details from the database (compared to loginUser to authenticate)

public string errorMessage; // Authentication errorMessage

/\* Authenticate()

public async Task<bool> Authenticate(IUser loginUser)

{

}

/\* GetDatabaseUser()

Call the method 'RunQuery'

to get loginUser details from the database \*/

bool GetDatabaseUser(IUser loginUser)

{

MySql.GetLoginDetails getLoginDetails = new MySql.GetLoginDetails();

databaseUser = getLoginDetails.RunQuery(loginUser); // Get loginUser object from the database.

if(databaseUser == null) // If the loginUser can not be found:

{

errorMessage = getLoginDetails.errorMessage; // - create error message

return false; // - and return False.

}

return true; // If loginUser data is okay, return True.

}

The class GetLoginDetails

gets the users login details from the MySQL database.

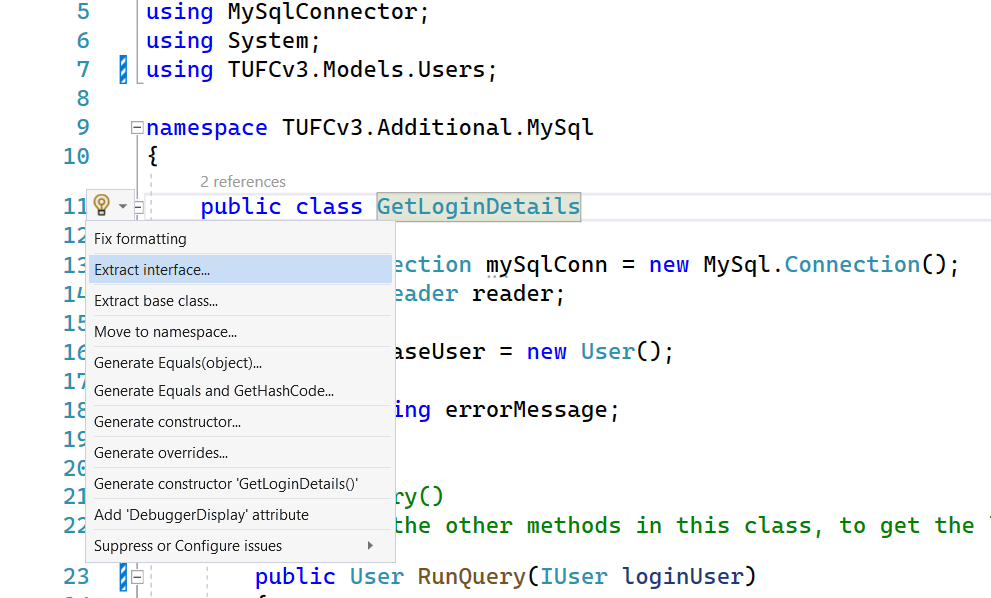
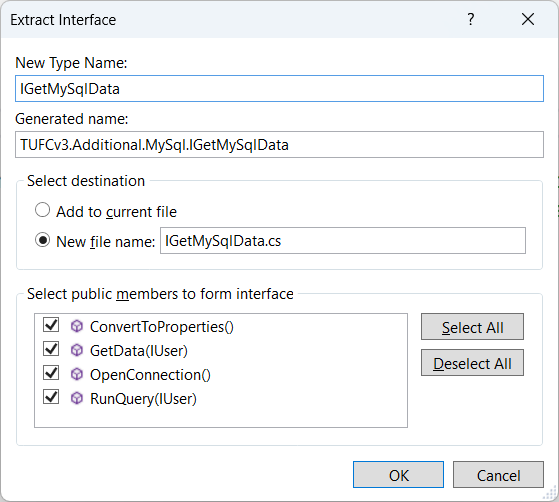
It goes through a sequence, that every MySQL query will follow:

* open a MySQL connection
* get data from the database
* convert MySQL data to C# properties

As these steps are common with every MySQL query

when I extract an interface from GetLoginDetails

and call it IGetMySqlData



The interface IGetLoginDetails can then be instantiated

instead of the type GetLoginDetails

/\* GetDatabaseUser()

Call the method 'RunQuery'

to get loginUser details from the database \*/

bool GetDatabaseUser(IUser loginUser)

{

GetLoginDetails IGetMySqlData getLoginDetails = new GetLoginDetails();

databaseUser = getLoginDetails.RunQuery(loginUser); // Get loginUser object from the database.

if(databaseUser == null) // If the loginUser can not be found:

{

errorMessage = getLoginDetails.errorMessage; // - create error message

return false; // - and return False.

}

return true; // If loginUser data is okay, return True.

}

*Next, getLoginDetails.errorMessage not longer works …*

### ‘getLoginDetails.errorMessage’ not longer works

When the type GetLoginDetails

is changed to an interface IGetMySqlData

the variable errorMessage displays an error.

This is because variables cannot be saved in an interface.

#### errorMessage error

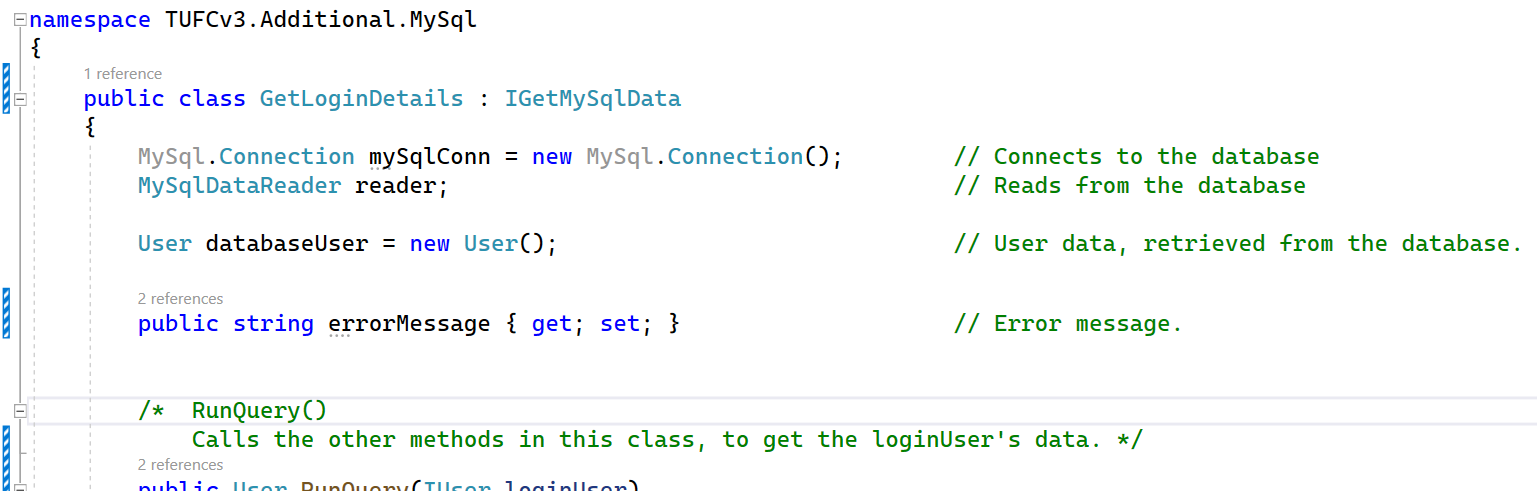


To resolve this issue:

1. Convert the variable errorMessage

located in GetLoginDetails.cs to a property.

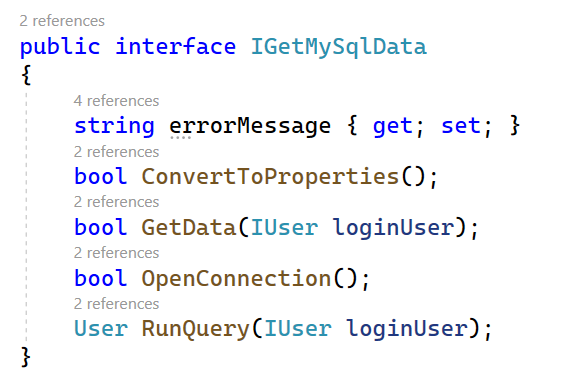
GetLoginDetails.cs



1. In the interface IGetMySqlData

include the property errorMessage { get; set; }

IGetMySqlData.cs



#### Test the application – IGetMySqlData is working

Replace the type GetLoginDetails

with IGetMySqlData – okay

## Create ‘IEncryptionCbcCombo’

The method AuthenticateUser.ComparePasswords()

instantiates the object EncryptionCbc encryption

that needs to be replaced with an interface.

/\* ComparePasswords()

Compare the login password

to the encrypted password retrieved from the database

by encrypting the login password using the same algorithm \*/

bool ComparePasswords(IUser loginUser)

{

EncryptionCbc encryption = new EncryptionCbc();

string encryptedLoginPassword = encryption.EncryptText(loginUser.Password); // Encrypt the loginUser's login password.

if (encryptedLoginPassword != databaseUser.Password) // If the login and database passwords do not match:

{

errorMessage = "Incorrect password"; // - create error message

return false; // - return False.

}

else

return true; // If the passwords match, return True.

}

As there are many methods of encryption

and the type of encryption could potentially change in the future

I’ll name a generic interface IEncryption

As not all encryption methods require password and IV

I will remove the method CreateEncryptor()

from the interface definition *(below)*

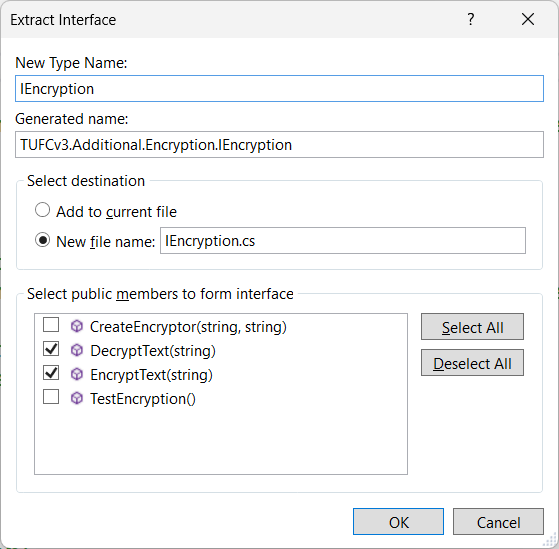
The method TestEncyption() is also not required for encryption functionality

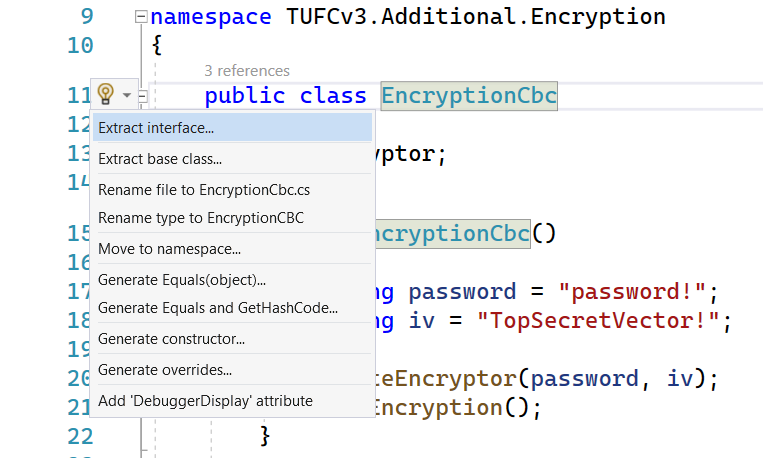
so, I will deselect this as well.

To create the interface, open the class EncryptionCbc

Ctrl + . on the classes name

and select Extract Interface …





*Next, Cyclic Block Cypher encryption …*

### Cyclic Block Cypher encryption

Cyclic Block Cypher encryption requires an encryptor

which is generated using a password and initialization vector (IV).

To capture these requirements in an interface

I made the each of these a public property.

namespace TUFCv3.Additional.Encryption

{

public class EncryptionCbc : IEncryption

{

public Aes encryptor { get; set; } // The encryption object, configured for Cipher Block Chain.

public string Password { get; set; } // CBC password

public string Iv { get; set; } // CBC IV

/\* Constructor

Get the Password and IV

Call the method CreateEncryptor() to creaete a CBC encryptor

\*/

public EncryptionCbc(string \_password = "password!", string \_iv = "TopSecretVector!")

{

string Password = \_password; // Make the password argument public

string IV = \_iv; // Make the IV argument public

CreateEncryptor(Password, Iv); // Create the object 'encryptor'

TestEncryption("Message to encrypt/decrypt"); // Test the encryption method ('comment out', but do not delete)

}

/\* CreateEncryptor()

Create and configure the object 'encryptor'

that encodes/decodes messages \*/

public void CreateEncryptor(string Password, string Iv)

{

encryptor = Aes.Create(); // Create the AES encryption object.

encryptor.Mode = CipherMode.CBC; // Set encryption method to Cyclic Block Chain.

SHA256 sha256 = SHA256.Create(); // Set the Key.

byte[] key = sha256.ComputeHash(Encoding.ASCII.GetBytes(Password));

encryptor.Key = key;

byte[] ivBytes = Encoding.ASCII.GetBytes(Iv); // Set the IV.

encryptor.IV = ivBytes;

}

// …

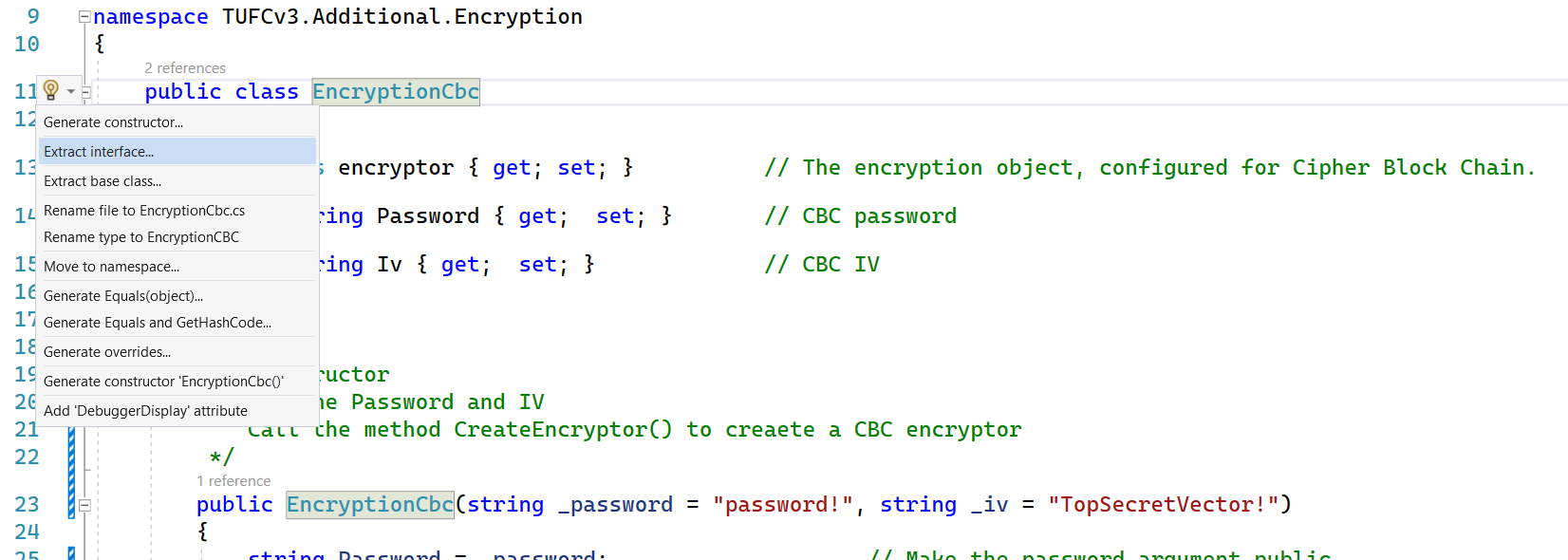
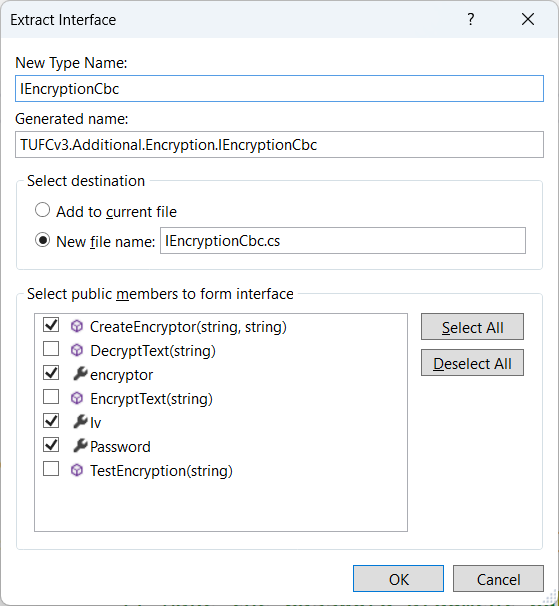
The object Aes encryptor

is created in the method CreateEncryptor() *(also shown above)*

I can now create a new interface called IEncryptionCbc

that captures things which are unique to CBC encryption

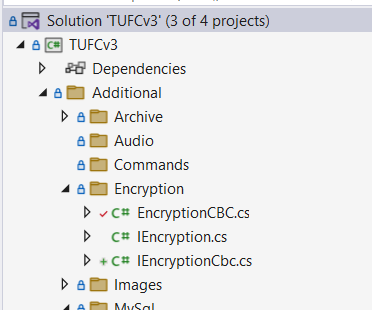
including the three properties and the method CreateEncryptor()



*Next, Combine IEncryption and IEncryptionCbc …*

### Combine ‘IEncryption’ and ‘IEncryptionCbc’

I now have two encryption interfaces IEncryption and IEncryptionCbc



### IEncryption

namespace TUFCv3.Additional.Encryption

{

public interface IEncryption

{

string DecryptText(string cipherText); // Decrypt an encryted string

string EncryptText(string plainText); // Encrypt plain text

}

}

### IEncryptionCbc

using System.Security.Cryptography;

namespace TUFCv3.Additional.Encryption

{

public interface IEncryptionCbc

{

Aes encryptor { get; set; } // Encryptor used in Cyclic Block Cypher (CBC) encryption.

string Iv { get; set; } // Initialization Vector

string Password { get; set; } // Password

void CreateEncryptor(string Password, string Iv); // Create the property 'encryptor', using the password and iv.

}

}

### Create ‘EncryptionCbcCombo’

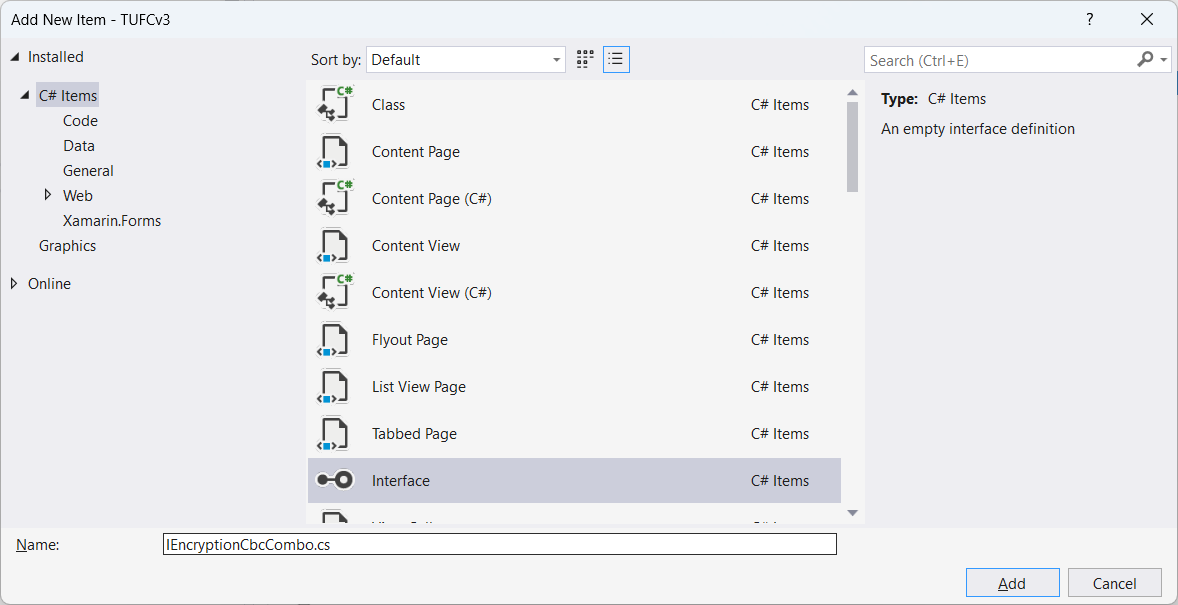
I need to combine the two Interfaces together

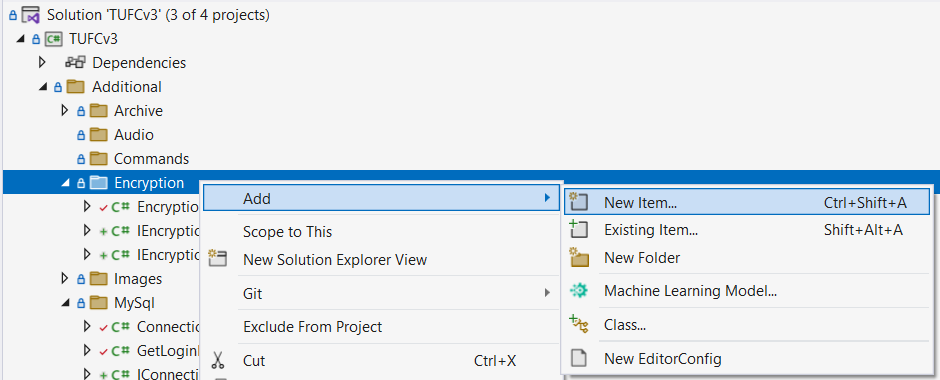
to create the interface I would like to used

called IEncrypionCbcCombo

To do this I will create a new interface called IEncrypionCbcCombo

which inherits from both of the existing interfaces IEncryption and IEncryptionCbc





The new interface IEncryptionCbcCombo

inherits from IEncryption and IEncryptionCbc

namespace TUFCv3.Additional.Encryption

{

public interface IEncryptionCbcCombo : IEncryption, IEncryptionCbc

{

}

}

It does not require any additional code.

It simply combines the two interfaces into one

that contains all of the required properties and methods

for CBC encryption.

*Next, Replace EncryptionCbc with IEncryptionCbcCombo …*

In the class AuthenticateUser

replace the instance of EncryptionCbc

with the interface IEncryptionCbcCombo, by:

1. Set the class EncryptionCbc

to inherit from IEncryptionCbcCombo

using System;

using System.IO;

using System.Security.Cryptography;

using System.Text;

namespace TUFCv3.Additional.Encryption

{

public class EncryptionCbc : IEncryptionCbcCombo

{

public Aes encryptor { get; set; } // The encryption object, configured for Cipher Block Chain.

public string Password { get; set; } // CBC password

public string Iv { get; set; } // CBC IV

/\* Constructor …

1. Replace the instance of EncryptionCbc

with the interface IEncryptionCbcCombo

in the method AuthenticateUser.ComparePasswords()

namespace TUFCv3.Additional

{

[XamlCompilation(XamlCompilationOptions.Compile)]

public class AuthenticateUser

{

// …

/\* ComparePasswords()

Compare the login password

to the encrypted password retrieved from the database

by encrypting the login password using the same algorithm \*/

bool ComparePasswords(IUser loginUser)

{

IEncryptionCbcCombo encryption = new EncryptionCbc();

string encryptedLoginPassword = encryption.EncryptText(loginUser.Password); // Encrypt the loginUser's login password.

if (encryptedLoginPassword != databaseUser.Password) // If the login and database passwords do not match:

{

errorMessage = "Incorrect password"; // - create error message

return false; // - return False.

}

else

return true; // If the passwords match, return True.

}

}

}

#### Test the app – Replaced EncryptionCbc with IEncryptionCbcCombo

After creating the interface IEncryptionCbcCombo

by combining the interfaces IEncryption and IEncryptionCbc

then replacing the instance of EncryptionCbc

with the interface IEncryptionCbcCombo

in the method Authenticate.ComparePasswords()

okay

## Create interfaces in ‘Encryption.cs’

There are several places in Encryption.cs

where new instances are created.

For example, in the method CreateEncryptor()

namespace TUFCv3.Additional.Encryption

{

public class EncryptionCbc : IEncryptionCbcCombo

{

public Aes encryptor { get; set; } // The encryption object, configured for Cipher Block Chain.

public string Password { get; set; } // CBC password

public string Iv { get; set; } // CBC IV

/\* Constructor

Get the Password and IV

Call the method CreateEncryptor() to creaete a CBC encryptor

\*/

public EncryptionCbc(string \_password = "password!", string \_iv = "TopSecretVector!")

{

string Password = \_password; // Make the password argument public

string Iv = \_iv; // Make the IV argument public

CreateEncryptor(Password, Iv); // Create the object 'encryptor'

TestEncryption("Message to encrypt/decrypt"); // Test the encryption method ('comment out', but do not delete)

}

/\* CreateEncryptor()

Create and configure the object 'encryptor'

that encodes/decodes messages \*/

public void CreateEncryptor(string Password, string Iv)

{

encryptor = Aes.Create(); // Create the AES encryption object.

encryptor.Mode = CipherMode.CBC; // Set encryption method to Cyclic Block Chain.

SHA256 sha256 = SHA256.Create(); // Set the Key.

byte[] key = sha256.ComputeHash(Encoding.ASCII.GetBytes(Password));

encryptor.Key = key;

byte[] ivBytes = Encoding.ASCII.GetBytes(Iv); // Set the IV.

encryptor.IV = ivBytes;

}

// …

These classes have been created by Microsoft.

I will never modify their content

so, there is no reason to extract interfaces

and replace the objects with interfaces.

### ‘O’ in SOLID Complete

I have now gone through each of the classes

involved with user login

and made each class them comply

with the ‘O’ in SOLID:

***Open to extension***

***But closed to modification***

*Next, ‘L’ in SOLID …*

### Create interfaces in ‘GetLoginDetails.cs’

### Create ‘IConnection’

The class GetLoginDetails

instantiates Connection mySqlConn

namespace TUFCv3.Additional.MySql

{

public class GetLoginDetails : IGetMySqlData

{

MySql.Connection mySqlConn = new MySql.Connection(); // Connects to the database

MySqlDataReader reader; // Reads from the database

User databaseUser = new User(); // User data, retrieved from the database.

public string errorMessage { get; set; } // Error message.

/\* …

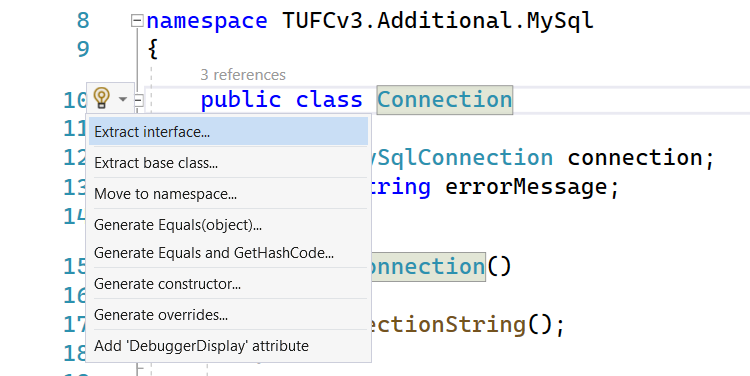
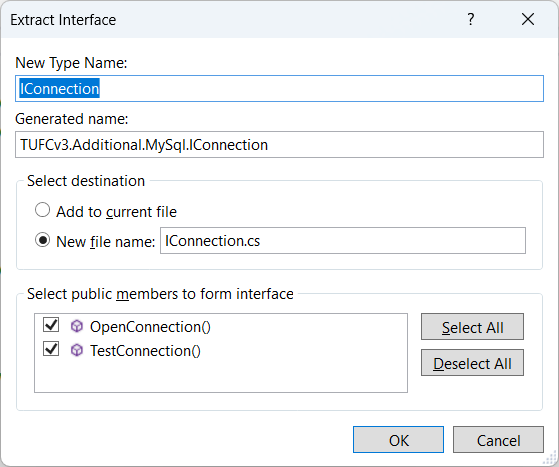
The object Connection needs to be replaced

with a new interface called IConnection

Go to the class Connection.cs

and extract a new interface, by pressing Ctrl + .

and selecting Extract Interface …



In the class GetLoginDetails

replace the object Connection

with the interface IConnection

namespace TUFCv3.Additional.MySql

{

public class GetLoginDetails : IGetMySqlData

{

MySql.Connection IConnection mySqlConn = new MySql.Connection(); // Connects to the database

MySqlDataReader reader; // Reads from the database

User databaseUser = new User(); // User data, retrieved from the database.

public string errorMessage { get; set; } // Error message.

/\* …

### New Errors ‘sqlCmd’ and ‘mySqlConn.connection’

Updating Connection to IConnection created two new errors

in the class GetLoginDetails

sqlCmd and mySqlConn.connection no longer work.



I will fix mySqlConn.connection first.

This has become erroneous because the interface IConnection

could not create the variable connection

when it was extracted from the class Connection

To resolve this:

1. The variable connection { get; set; }

needs to be converted to a property in Connection

namespace TUFCv3.Additional.MySql

{

public class Connection : IConnection

{

public MySqlConnection connection { get; set; } // Connection to the MySQL database

public string errorMessage { get; set; } // Error message

public Connection()

{

ConnectionString(); // Create the connection string

}

/\* …

1. The property connection { get; set; }

also needs to be added to the interface IConnection

namespace TUFCv3.Additional.MySql

{

public interface IConnection

{

MySqlConnection connection { get; set; }

string errorMessage { get; set; }

bool OpenConnection();

bool TestConnection();

}

}

Adding the property MySqlConnection connection {get; set;}

to Connection and IConnection

resolved both errors sqlCmd and mySqlConn.connection

#### Test the app – ‘Connection’ replaced by ‘IConnection’

Extracted the interface: IConnection

from: Connection

and added the property: MySqlConnection connection { get; set; }

In the class GetLoginDetails

replaced the instance of: MySql.Connection mySqlConn = new MySql.Connection();

with: MySql.IConnection mySqlConn = new MySql.Connection();

Tested the add - okay

## ‘L’ in SOLID

The ‘L’ in SOLID refers to the Liskov Substitution Principle

https://www.youtube.com/watch?v=-3UXq2krhyw&list=PLAaFb7UfyShCoS246UzZJNEiXuD8bg02e&index=3

*“If* ***S*** *is a subtype of type* ***T***

*then objects of type* ***T***

*may be replaced with objects of type* ***S****”*

I have already introduced created the ‘Liskov Substitution Principle’

an example is the interface [EncryptionCbcCombo](#_Create_‘IEncryptionCbcCombo’)

EncryptionCbcCombo includes two interfaces :

* IEncryption
* IEncryptionCbc

Which allows EncryptionCbcCombo *(the subtype)*

to be used wherever IEncryption is instantiated in the application.

## ‘I’ in SOLID

<https://www.youtube.com/watch?v=y1JiMGP51NE&list=PLAaFb7UfyShCoS246UzZJNEiXuD8bg02e&index=4>

The Interface Segregation Principle (ISP) states:

***“A client should not be forced to depend***

***on interfaces they do not use”***

This topic is also covered when the interface [EncryptionCbcCombo](#_Create_‘IEncryptionCbcCombo’) was created.

## ‘D’ in SOLID

<https://www.youtube.com/watch?v=NnZZMkwI6KI&list=PLAaFb7UfyShCoS246UzZJNEiXuD8bg02e&index=5>

The ‘D’ ins SOLID stand for ‘The Dependency Inversion Principle’

which states:

*“High level modules*

*should not depend on lower-level modules,*

*instead, they should both rely on abstractions.*

*Also, the abstractions should not depend upon details”*

Any time an object is instantiated *(‘newed-up’)*

in an application, the high-level module *(the class creating the instance)*

is tightly bound to the lower-level module *(the object being created)*

If the lower-level module is modified, it could affect (or even break)

the operation of the high-level object that instantiated it.

To remove tight coupling

types are replaced with interfaces

and the interfaces are all instantiated from one location.

To implement dependency inversion in TUFCv3

I’ll create the new static class Factory

and static methods which instantiate and return interfaces

Each class then calls the relevant method in Factory

to create new object instances.

## Create ‘Factory’

In the solution TUFCv3, create the static class Factory

that will instantiate all objects.