Xamarin XWM (TUFCv3)

TODO

Contents

[Introduction 3](#_Toc113775921)

[Create a new GitHub branch 4](#_Toc113775922)

[Create TUFCv3 5](#_Toc113775923)

[Update Software 6](#_Toc113775924)

[Run the app 8](#_Toc113775925)

[Install MySqlConnector 9](#_Toc113775926)

[Temporary layout for MainPage() 12](#_Toc113775927)

[Test commands for SQL 12](#_Toc113775928)

[Layout 12](#_Toc113775929)

[Temporary XAML layout to test SQL commands … 13](#_Toc113775930)

[Create and execute MySqlConnector commands 14](#_Toc113775931)

[Designing for layouts for small devices 16](#_Toc113775932)

[Return values using MySqlConnector 16](#_Toc113775933)

[Troubleshooting ‘Unable to connect to any of the specified MySQL Hosts’ 21](#_Toc113775934)

[MVVM: *‘Model, View, ViewModel’* 23](#_Toc113775935)

[Create MVVM structure in an application 23](#_Toc113775936)

[Move MySqlConnector Tests 24](#_Toc113775937)

[Xamarin Navigation 27](#_Toc113775938)

[Navigate to MySqlConn.xaml 27](#_Toc113775939)

[Replace the MySqlConnector test code, in located in MainPage 28](#_Toc113775940)

[Add a Navigation method to MainPage 29](#_Toc113775941)

[MainPage tidied up 30](#_Toc113775942)

[Create the Xamarin object model ‘User’ 32](#_Toc113775943)

[Create the Class User 32](#_Toc113775944)

[Create private properties for ‘User’ 33](#_Toc113775945)

[Create Public properties for ‘User’ 33](#_Toc113775946)

[Create public properties for ‘User’ 34](#_Toc113775947)

[Create the method SetProperty() for ‘User’ 35](#_Toc113775948)

[Generic methods 36](#_Toc113775949)

[CallerMemberName 36](#_Toc113775950)

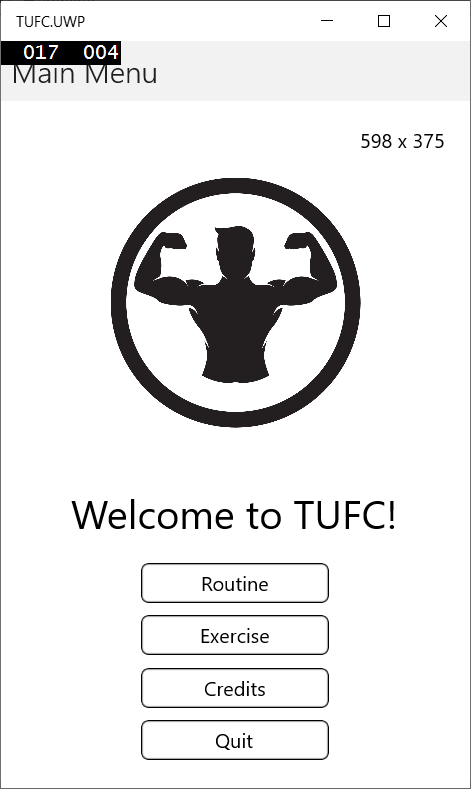
[Invoke the event PropertyChanged 36](#_Toc113775951)

# Introduction

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

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

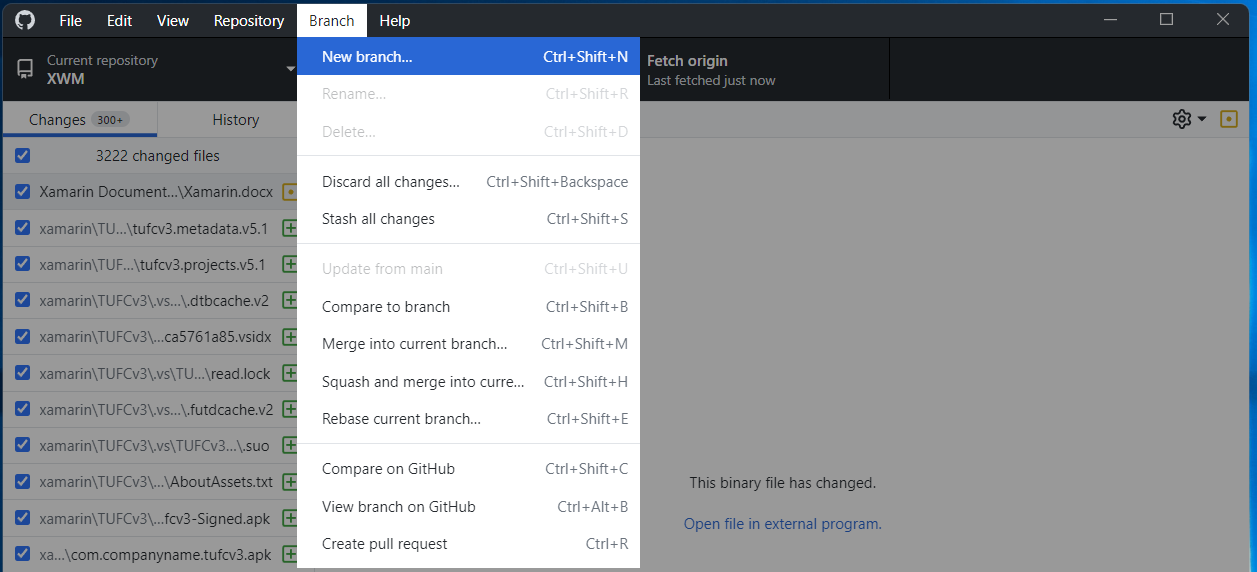
XWM is demonstration *(including ‘how-to’ documentation)*

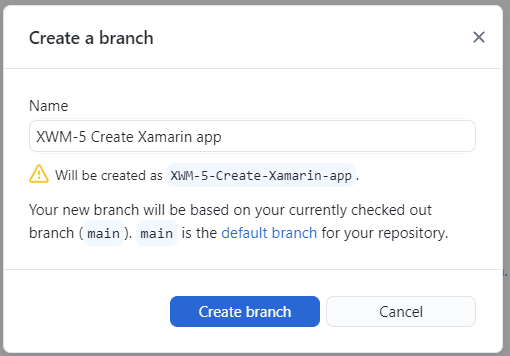
bringing these technologies together.

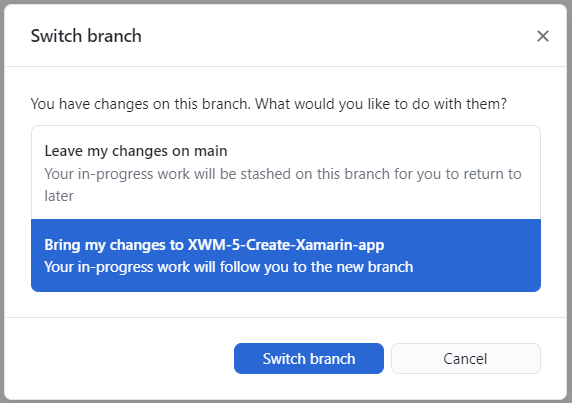
# Create a new GitHub branch

Before saving Xamarin software to GitHub

create a new branch.



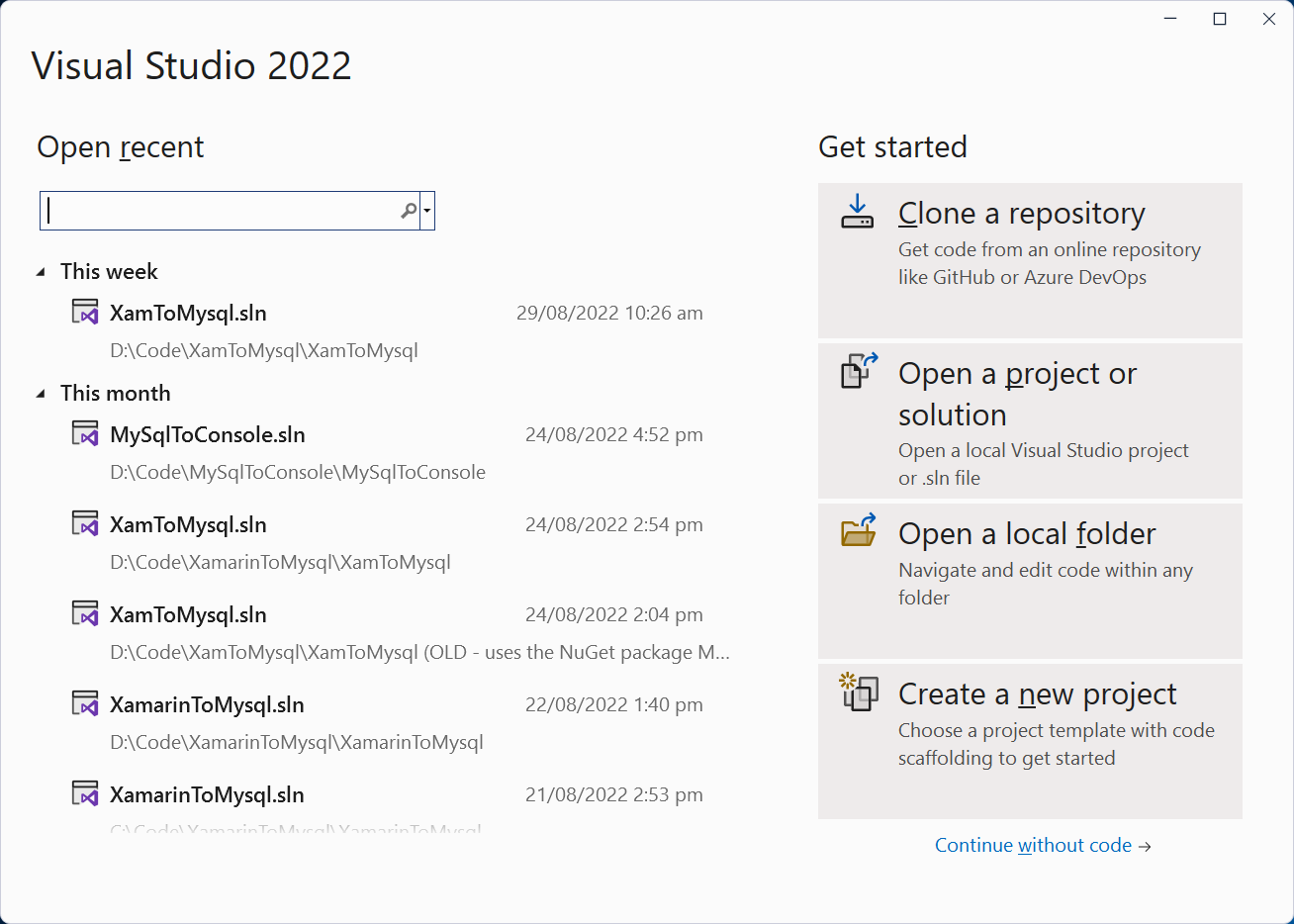


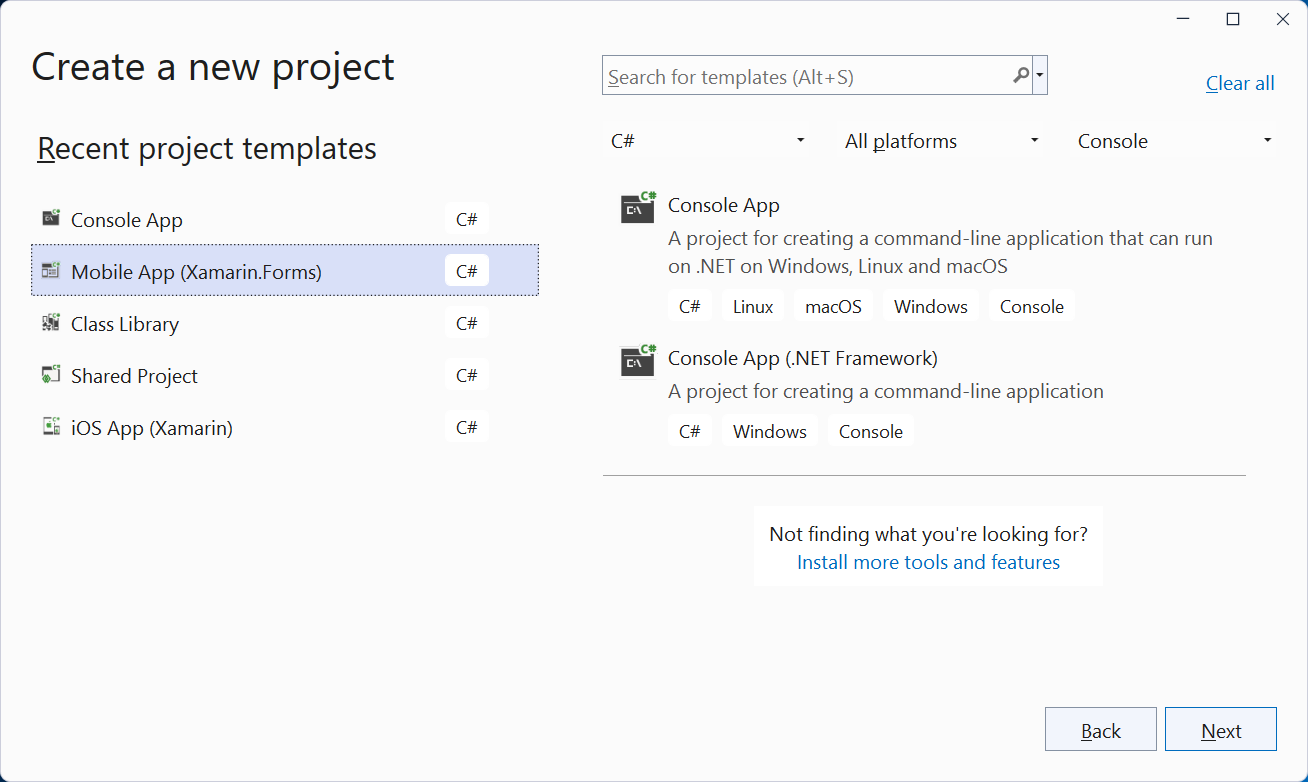


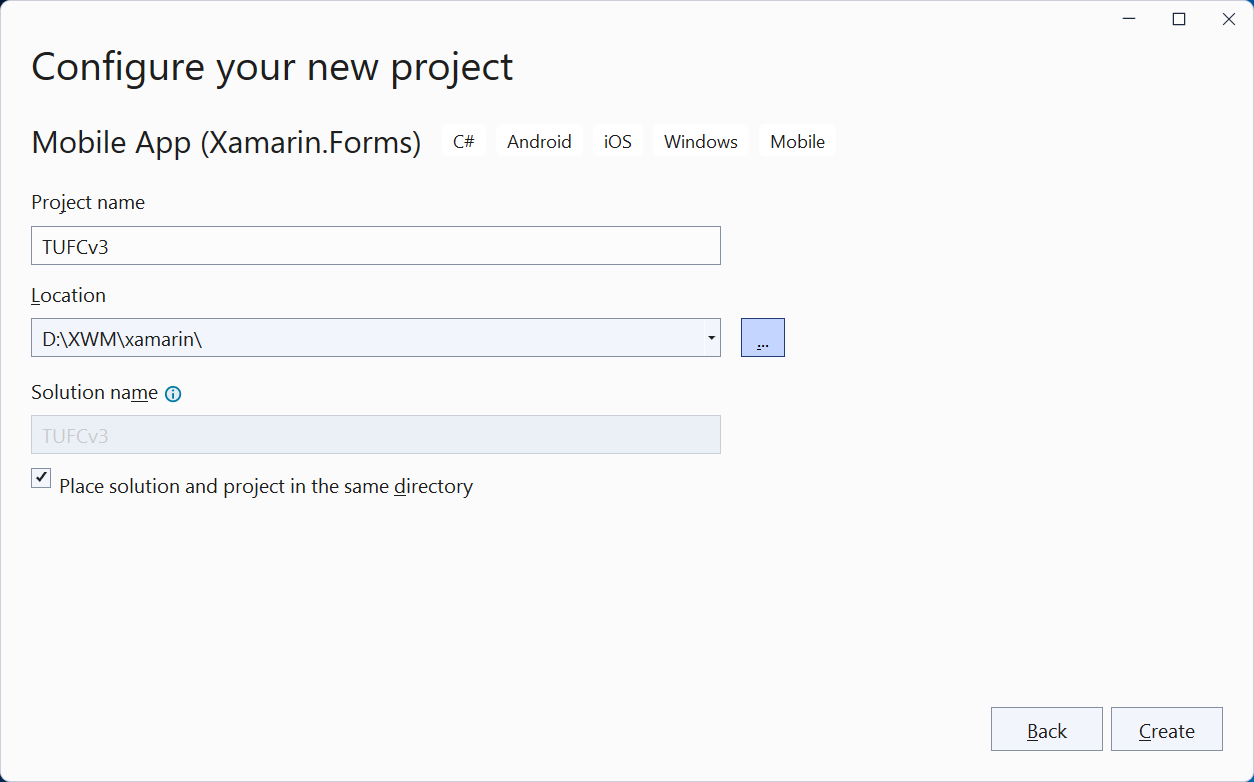
# Create TUFCv3

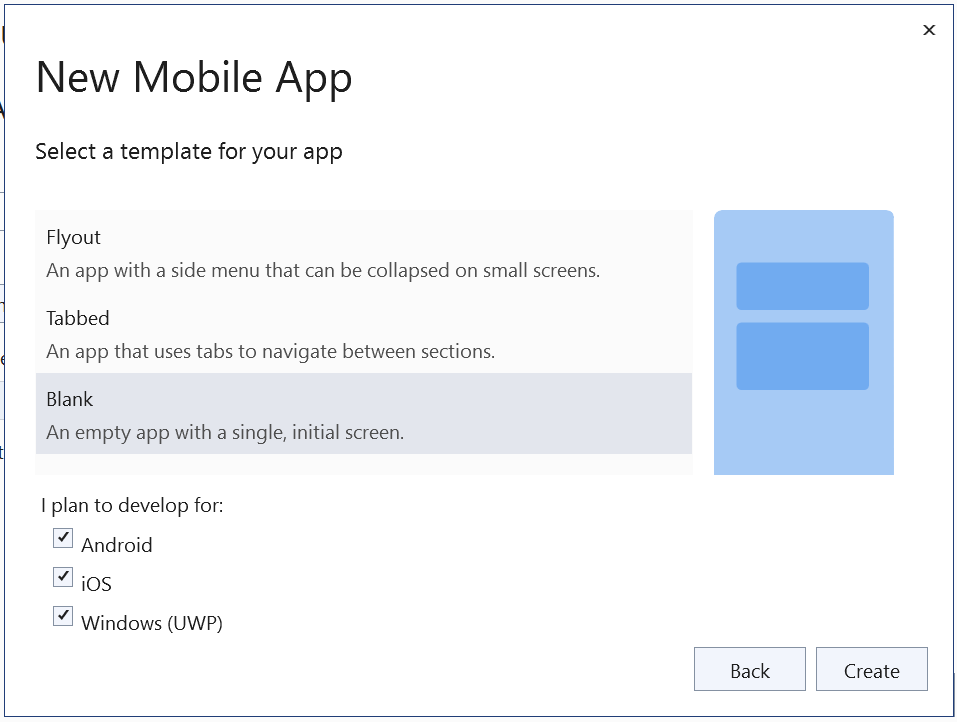
Create the Xamarin project TUFCv3

in the folder D:\XWM\xamarin









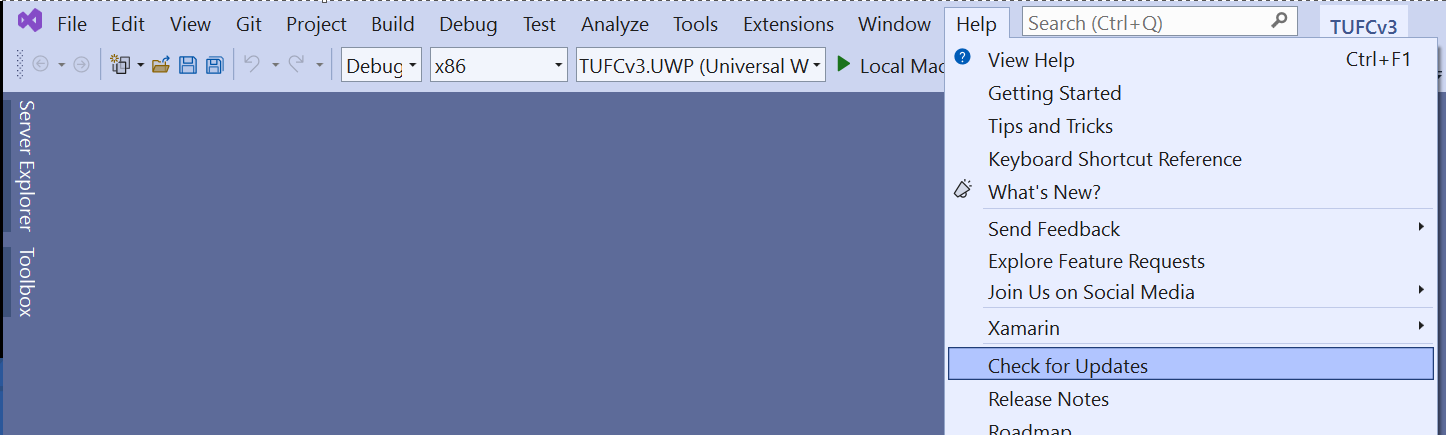
*Next, Update NuGet and run the application …*

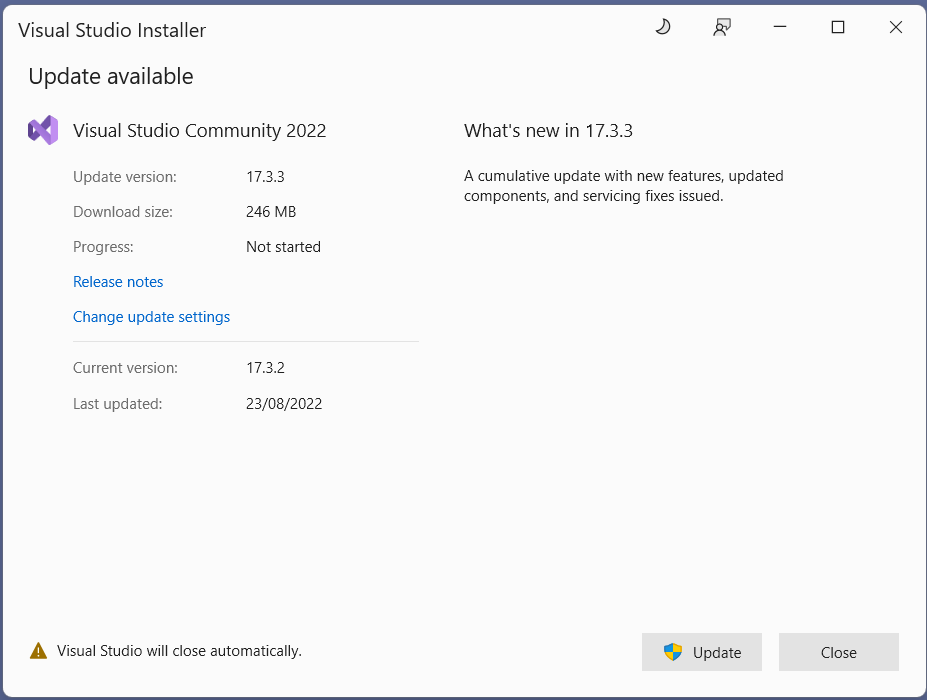
## Update Software

Before running the application check for updates on:

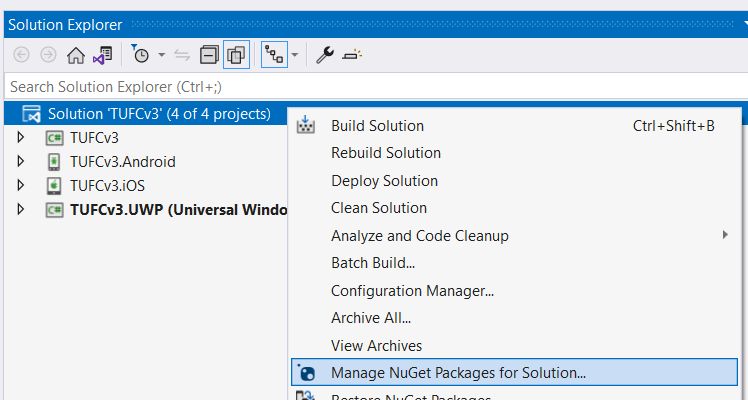
* Visual Studio
* NuGet
* Android
* Mac

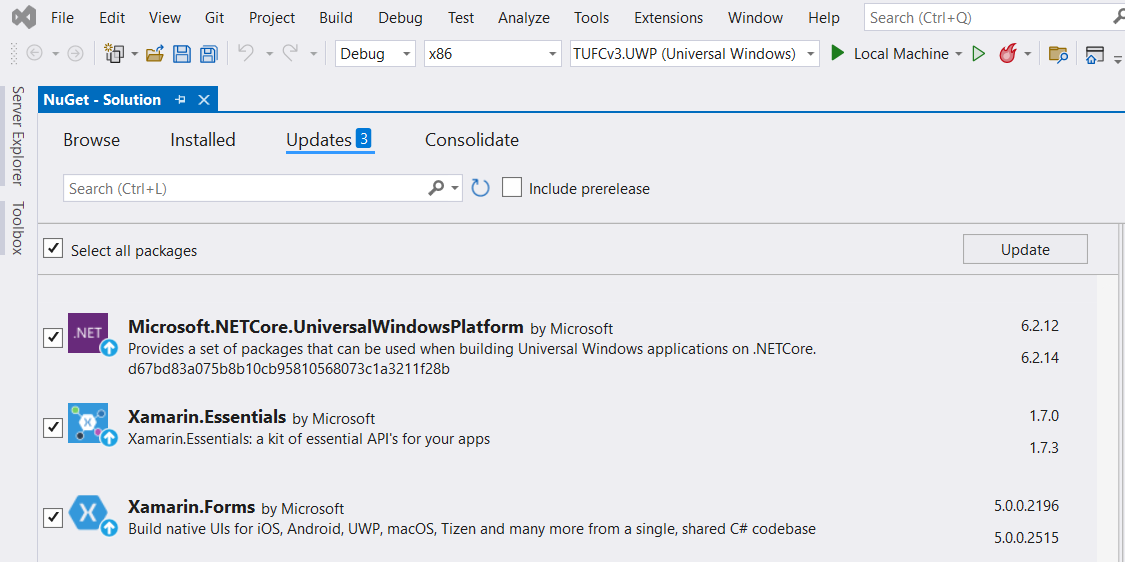
#### Update Visual Studio





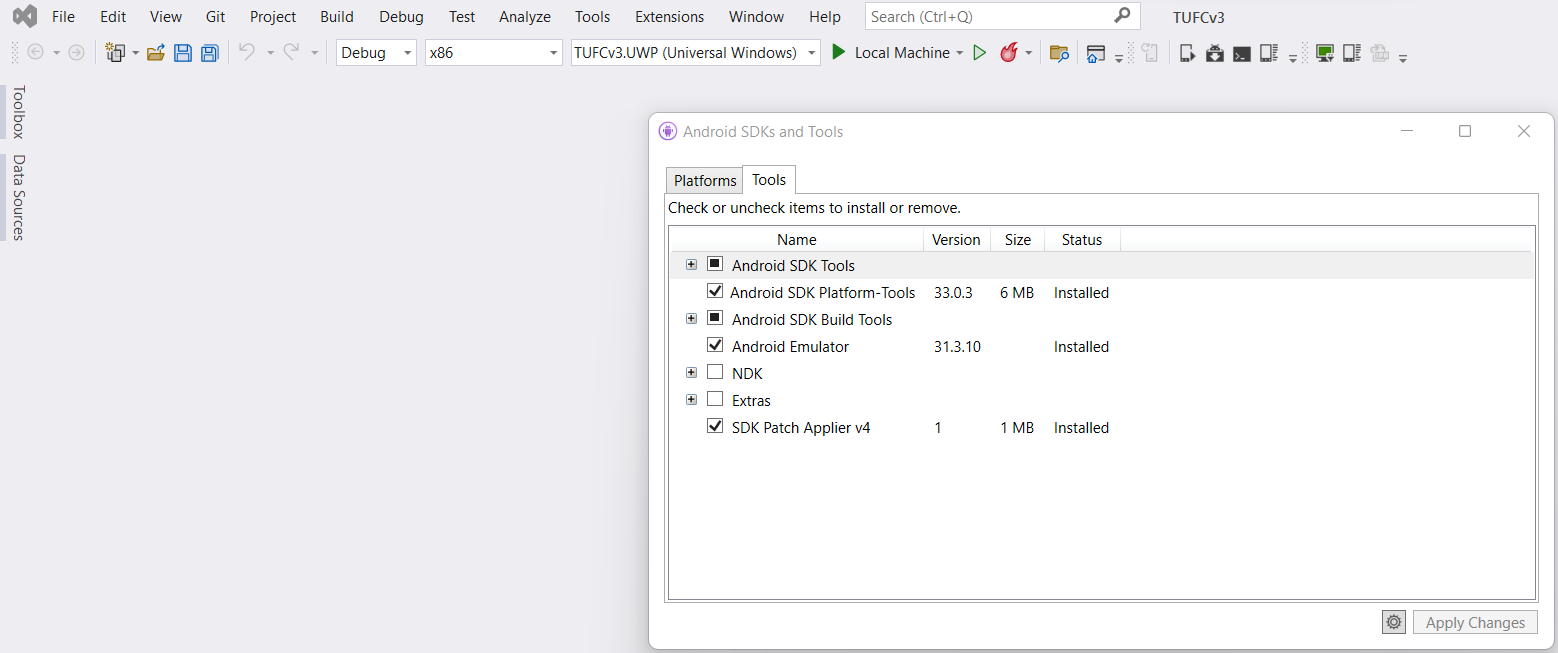
#### NuGet





*Next, Android SDK updates …*

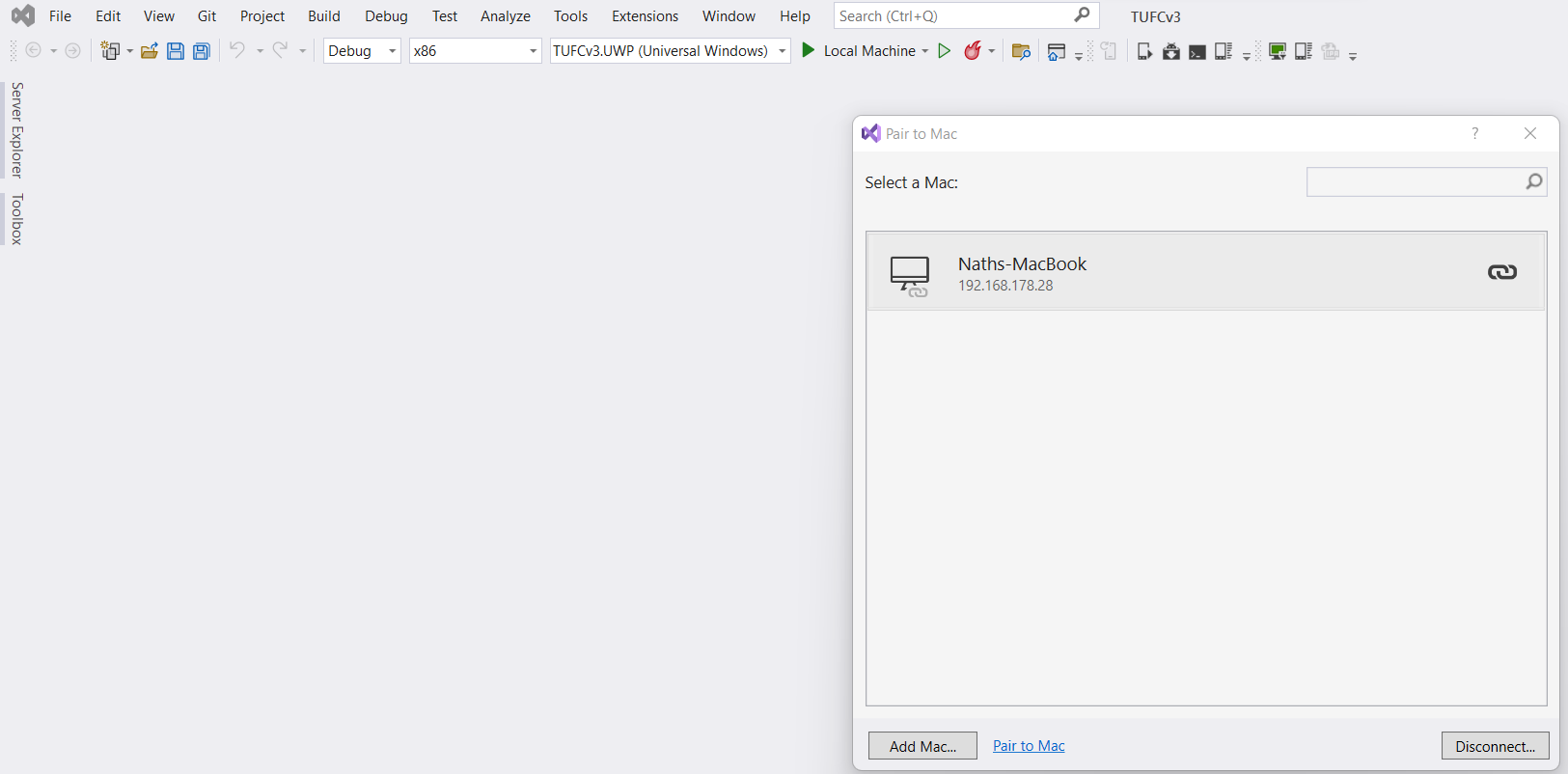
#### Android – Check the SDK manager for updates



#### 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

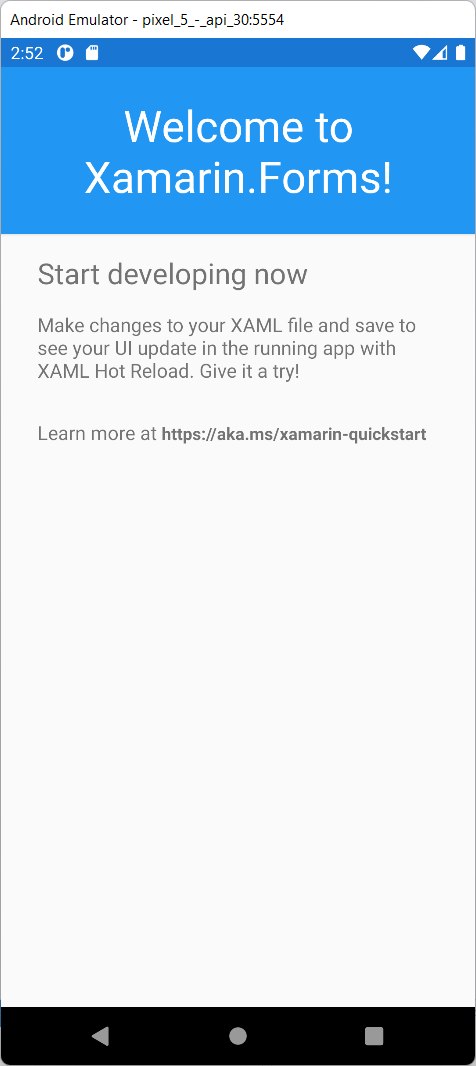
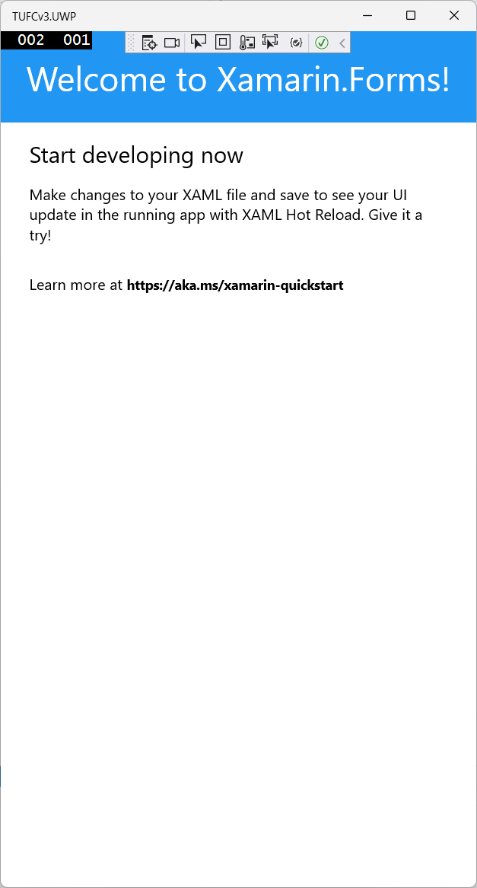
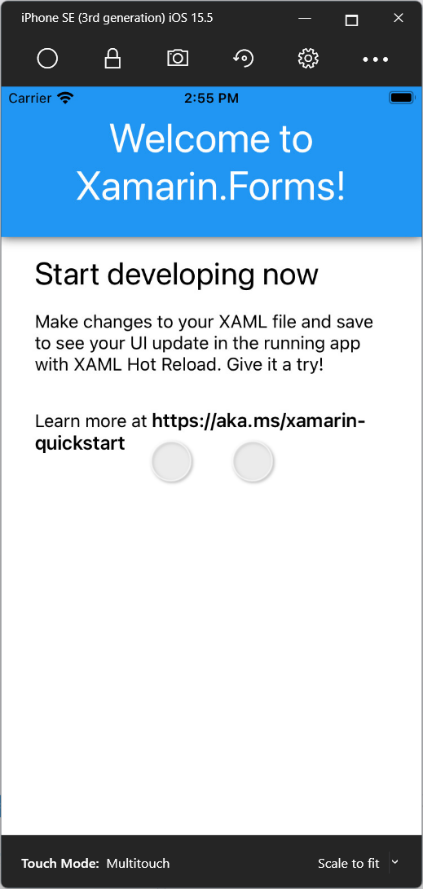
and software is up to date, run the application.

#### 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/

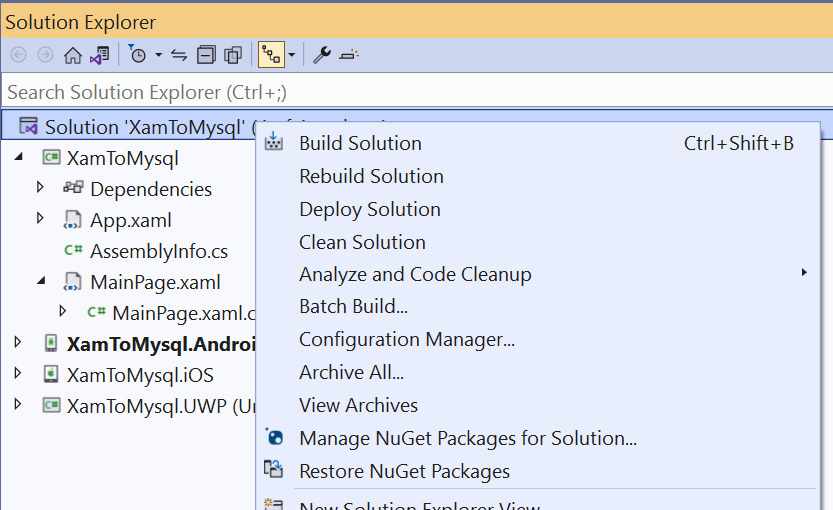
To allow the Xamarin application to connect to the MySQL database

install the NuGet package MySqlConnector

#### Example - Install MySqlConnector using NuGet

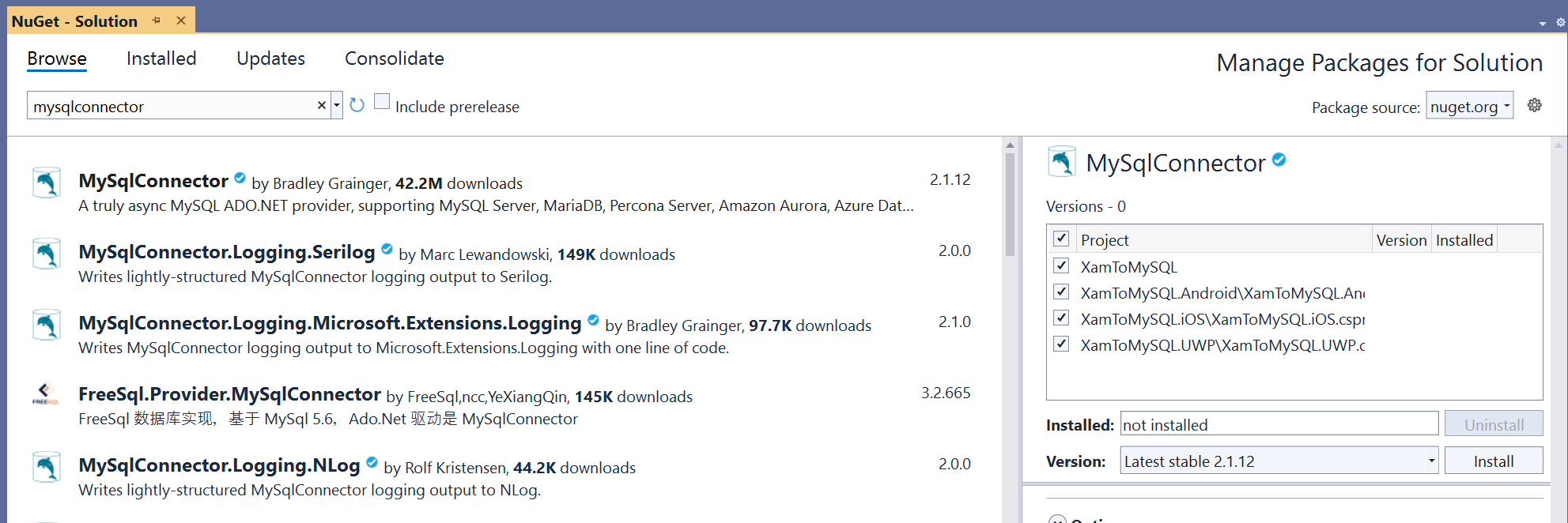
Right click the solution

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");

}

}

}

}

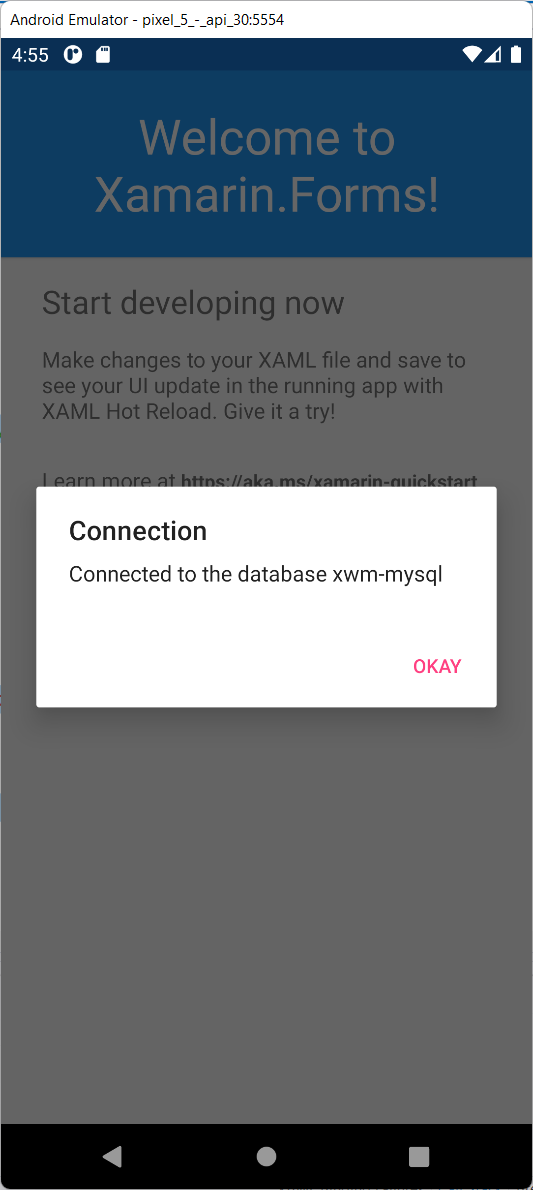
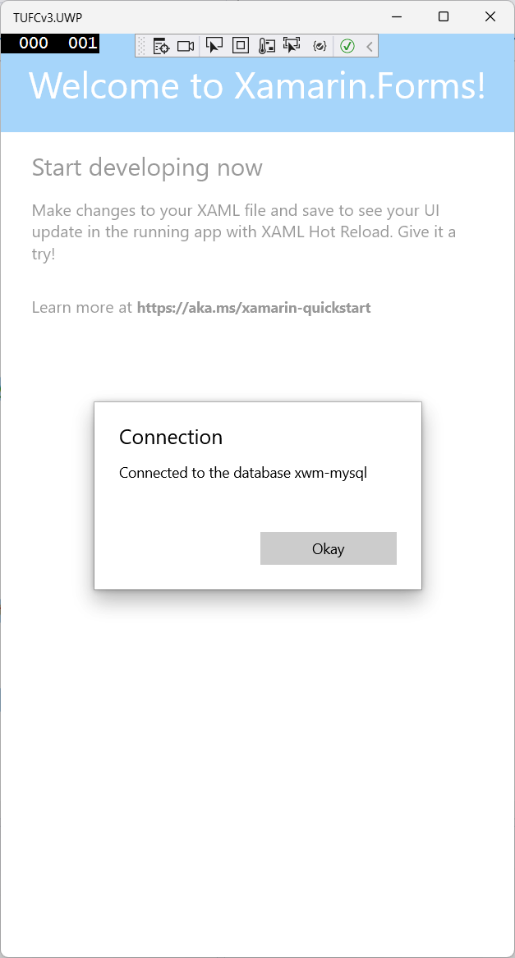
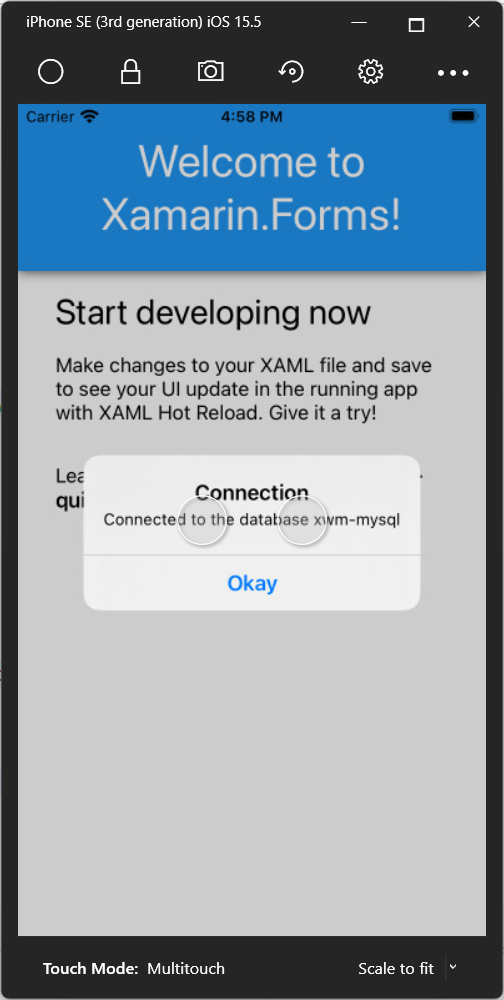
*Next, Test the connection …*

#### 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>

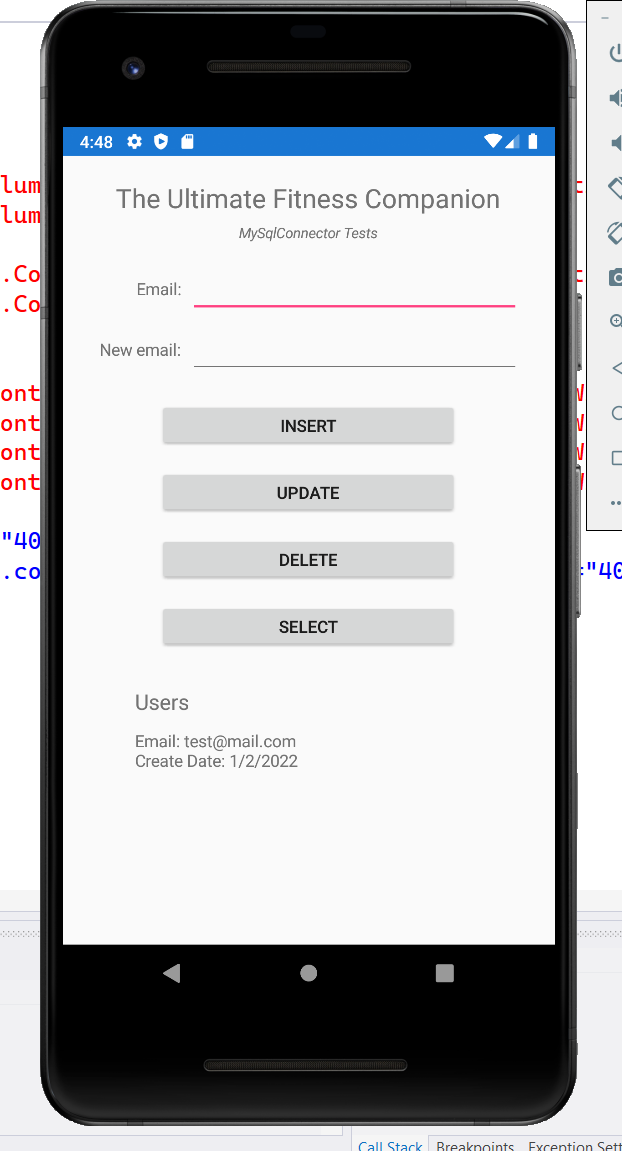
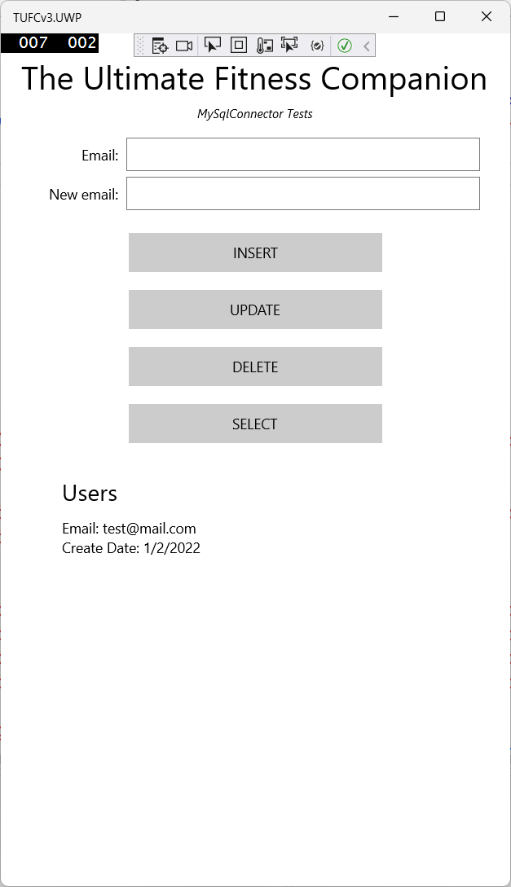
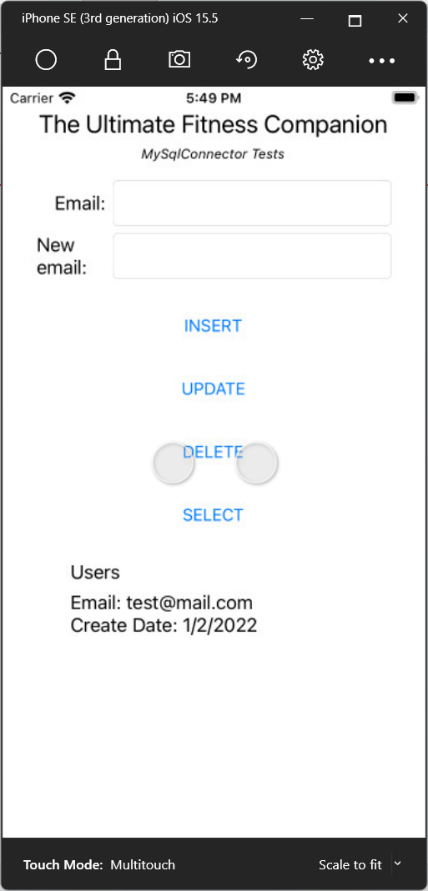
</ContentPage>

#### 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";

}

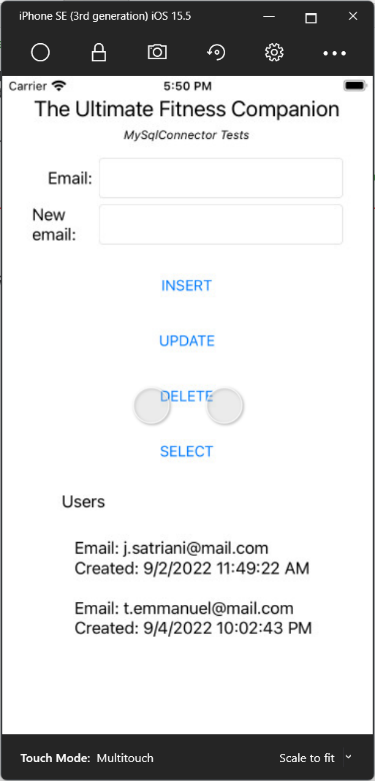
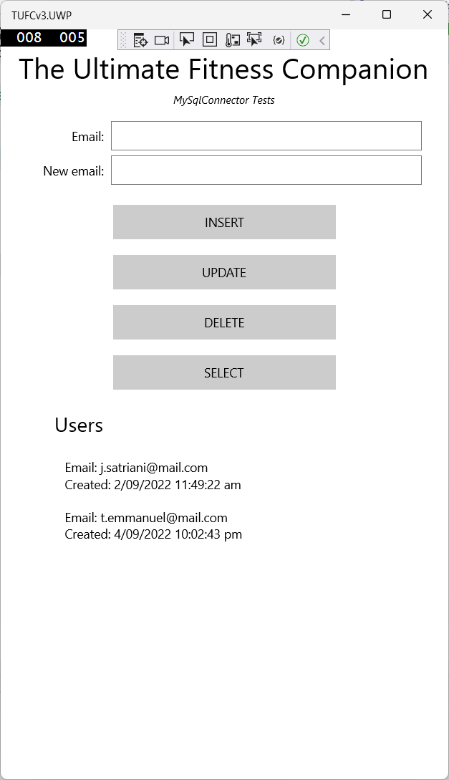
lblUsers.Text = displayedString;

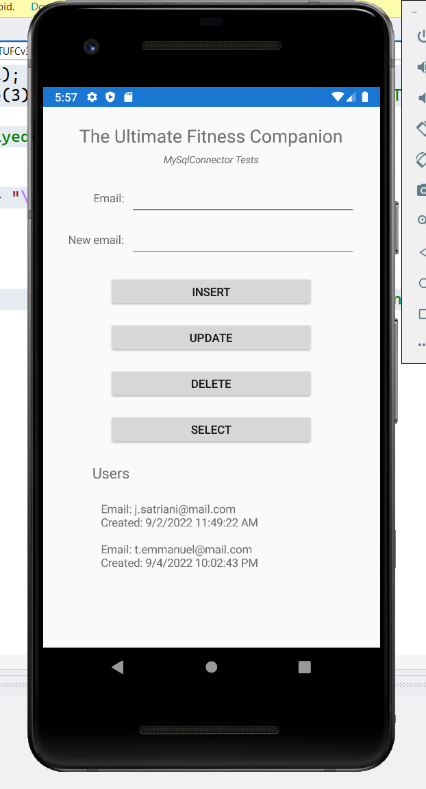
connection.Close(); // Close the connection

}

}

}

*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

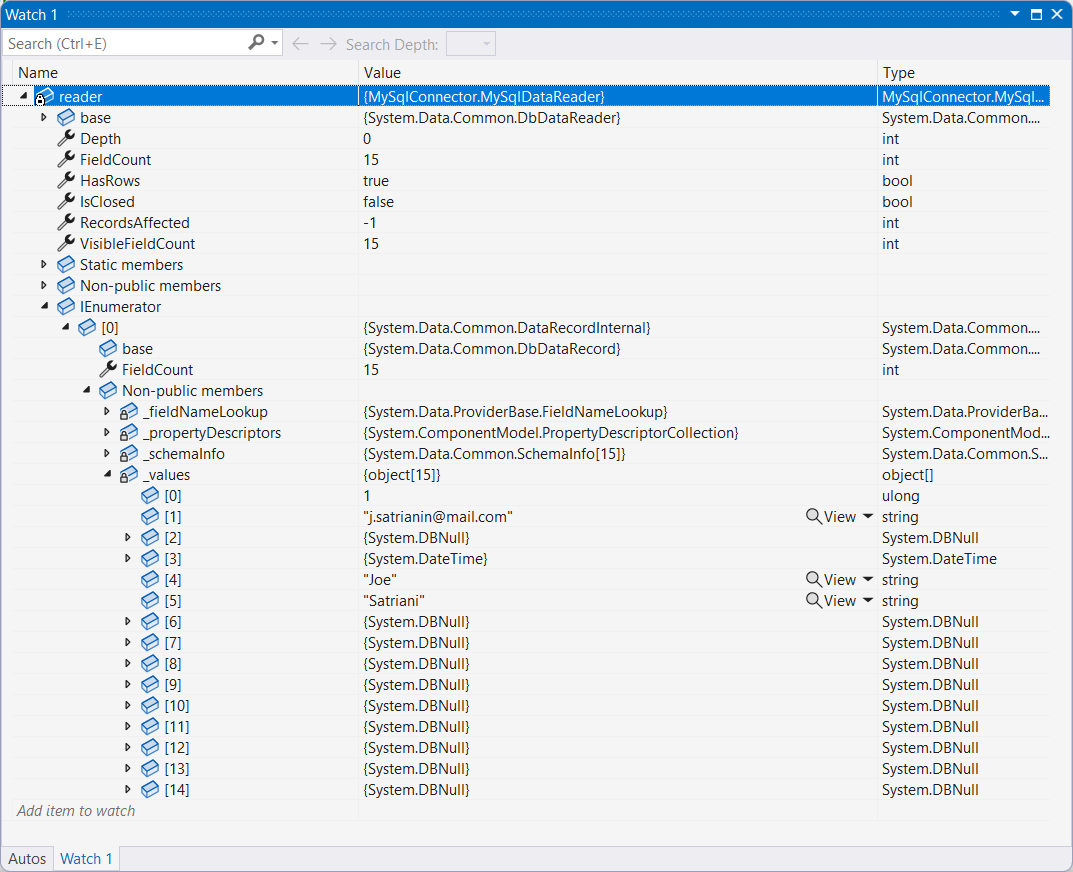
are nested arrays.

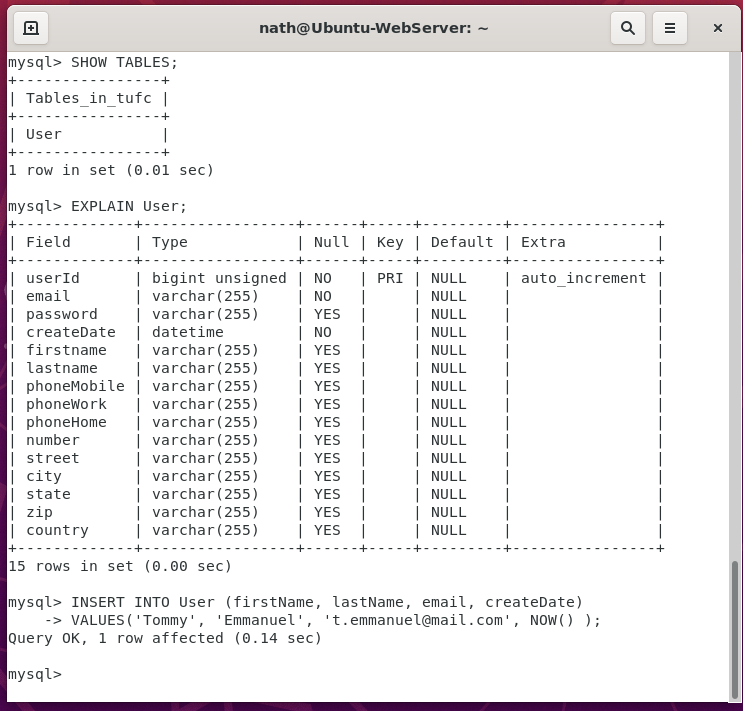
The outer array is a list of all the returned users

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)

{

}

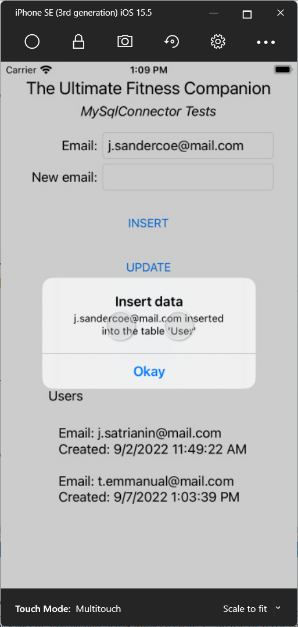
private async void OnSelectClick(object sender, EventArgs e)

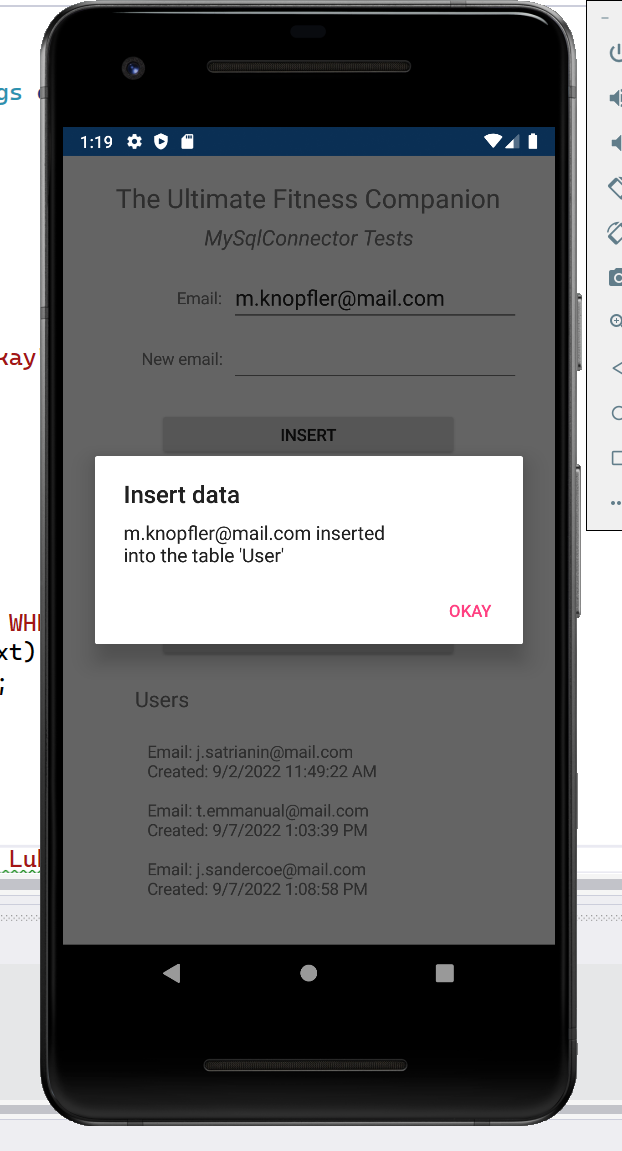
{

}

}

}

*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())

{

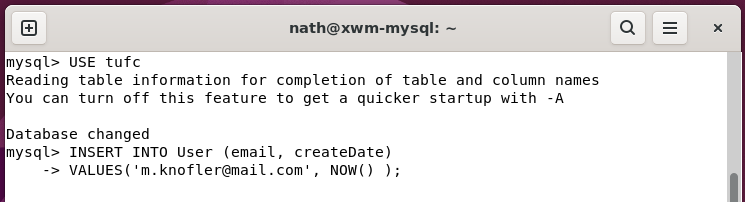
cmd.Connection = connection;

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)

{

}

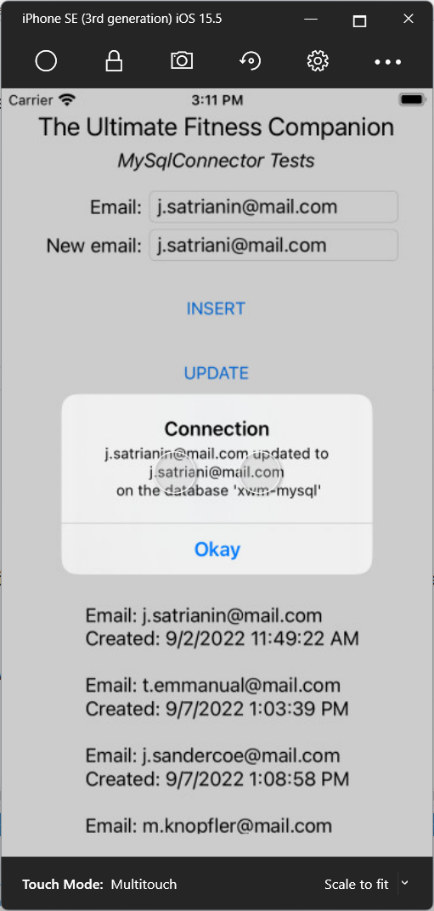
private async void OnSelectClick(object sender, EventArgs e)

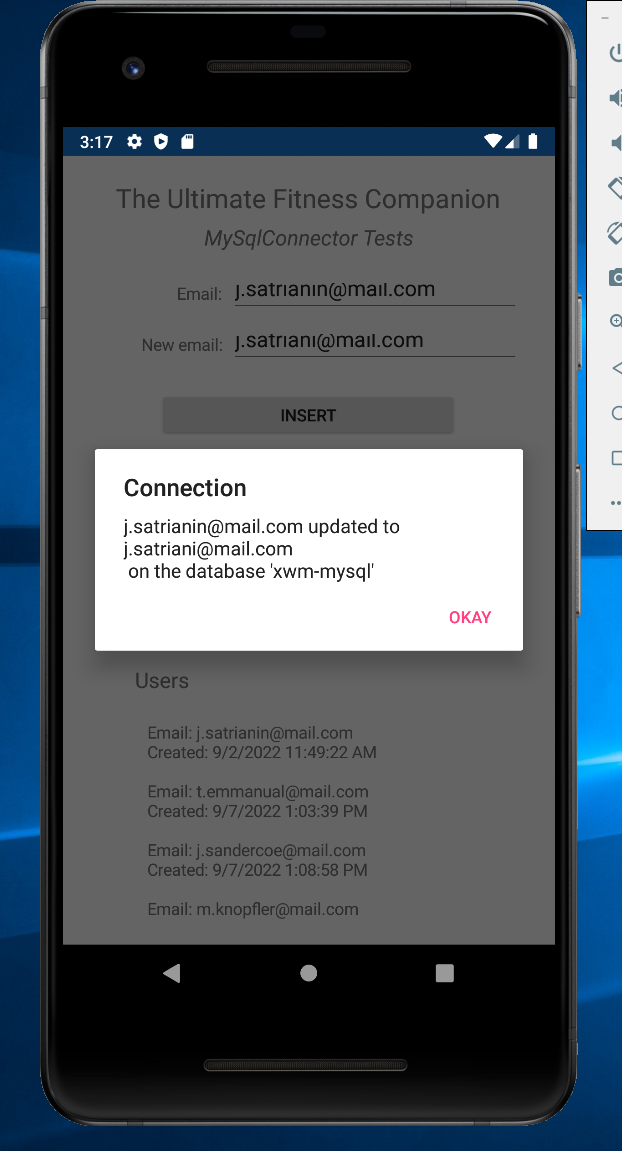
{

}

}

}

*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)

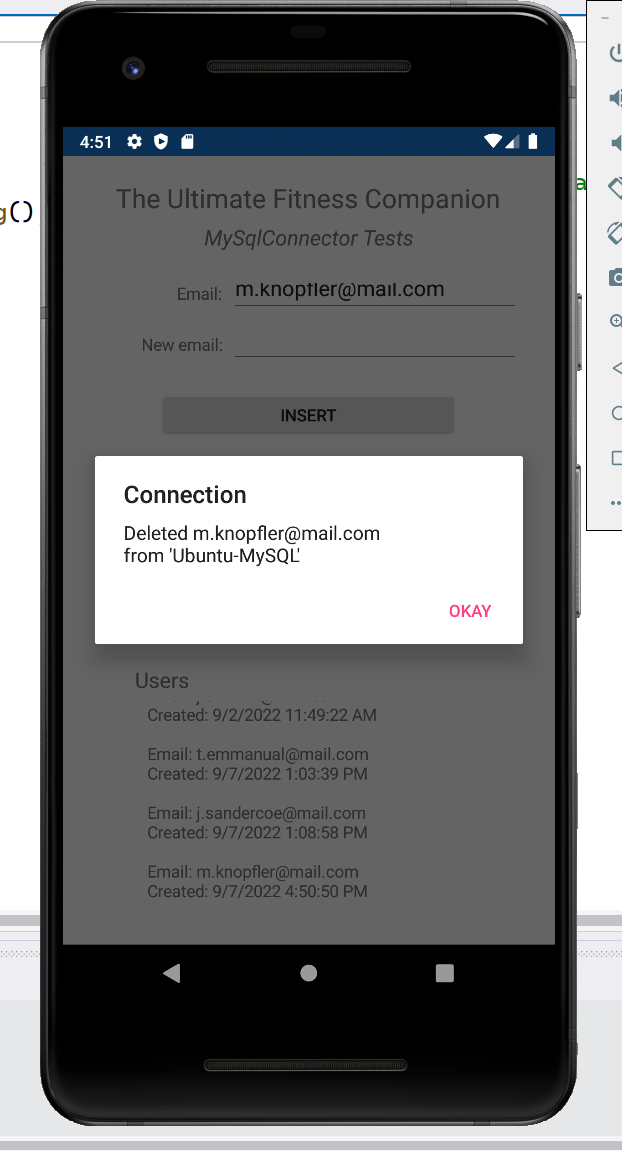
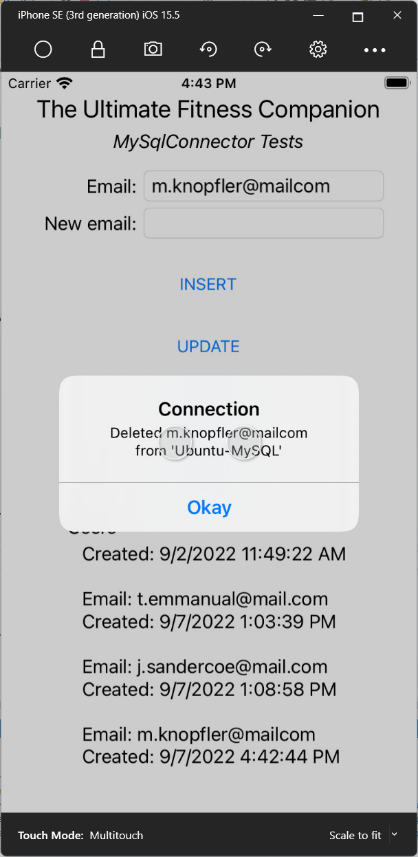
{

}

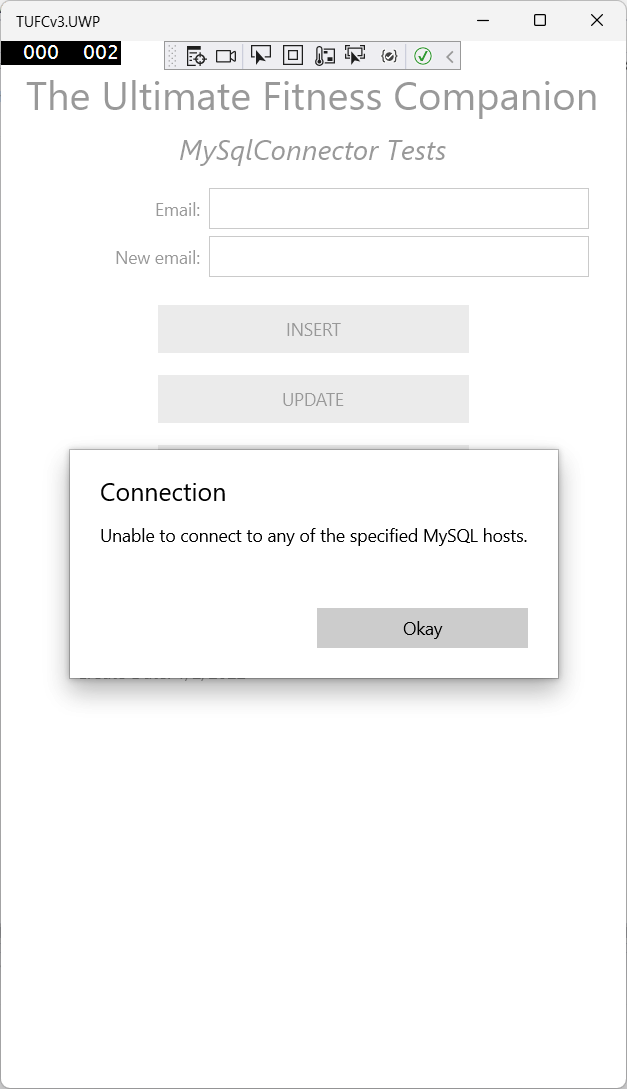
}

}

*Android iOS*



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



# MVVM: *‘Model, View, ViewModel’*

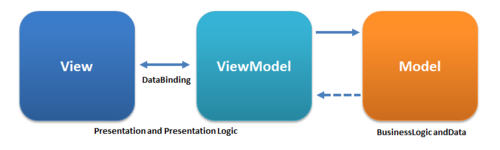
The next step in I need in the Xamarin application TUFCv3

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

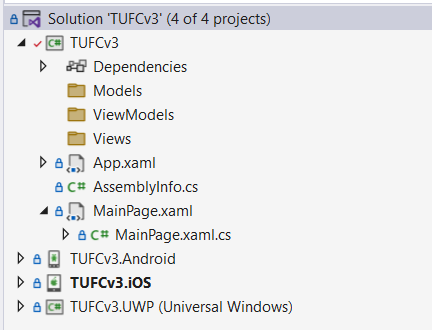
which allows the graphical user interface and the business logic

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.

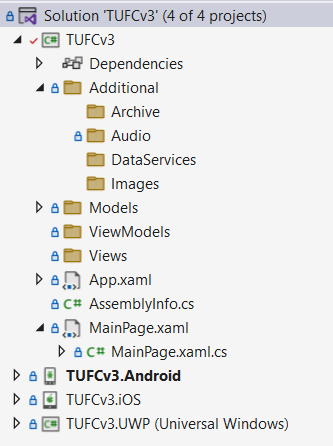


In addition to the folders for MVVM

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

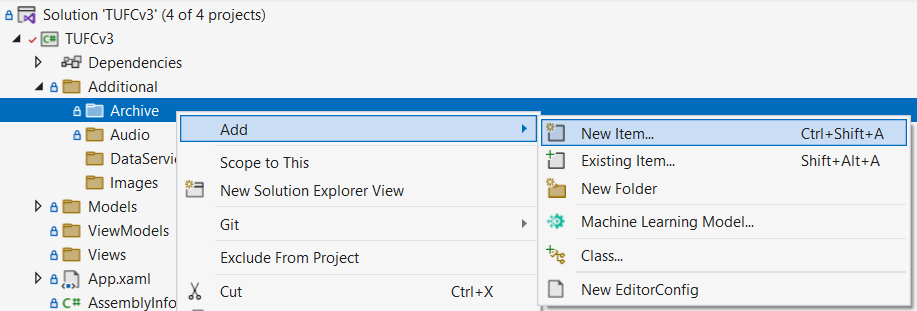
and may be helpful while doing diagnostics.

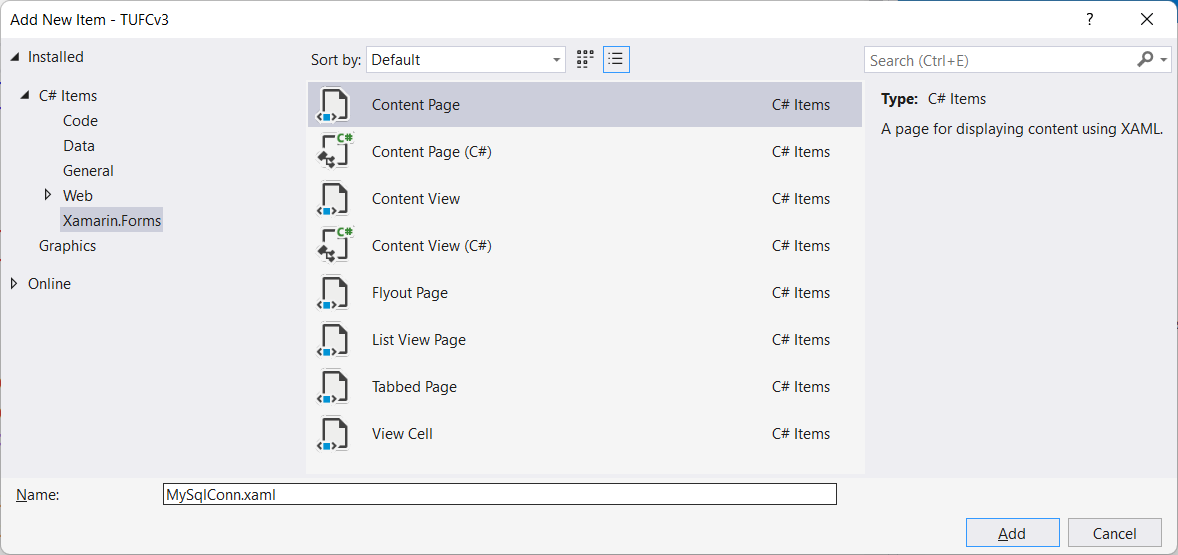
so will copy the original code to the folder Additional.Archive

#### 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)

}

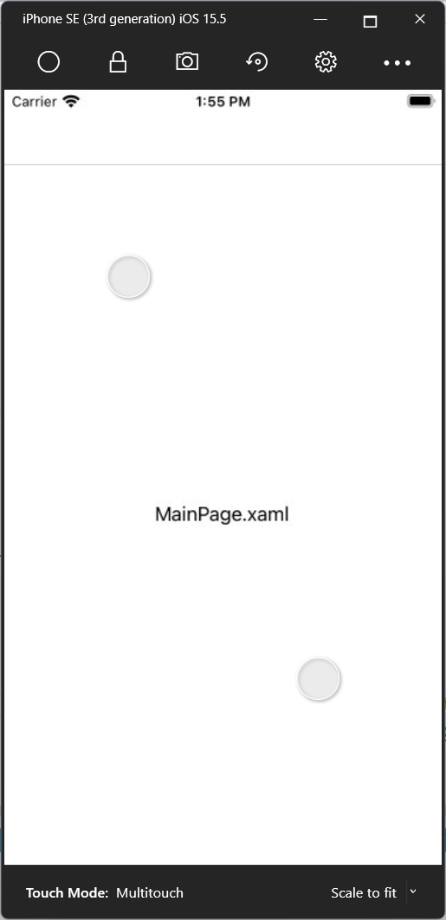
}

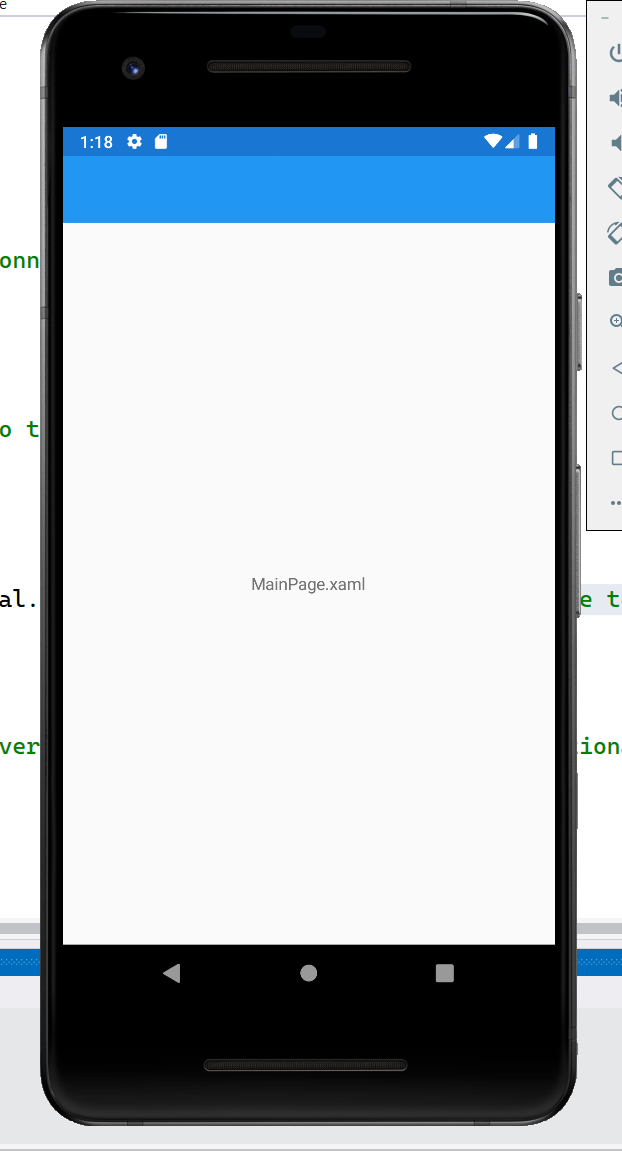
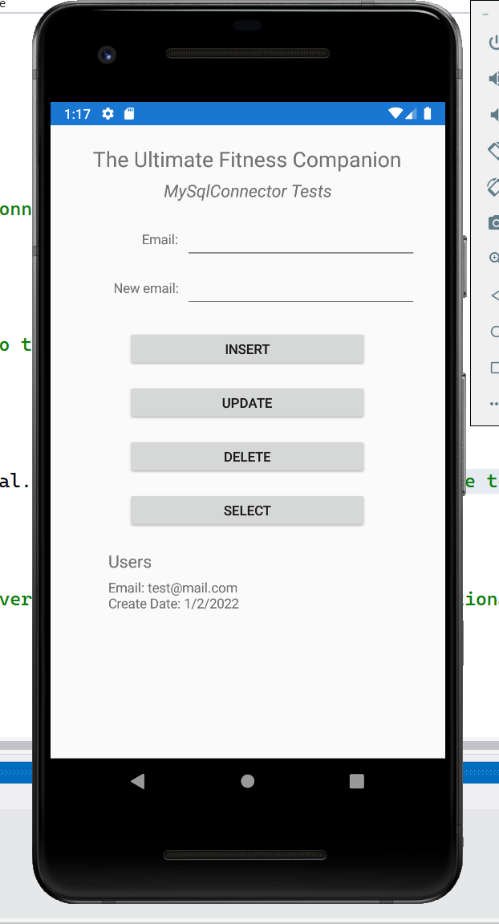
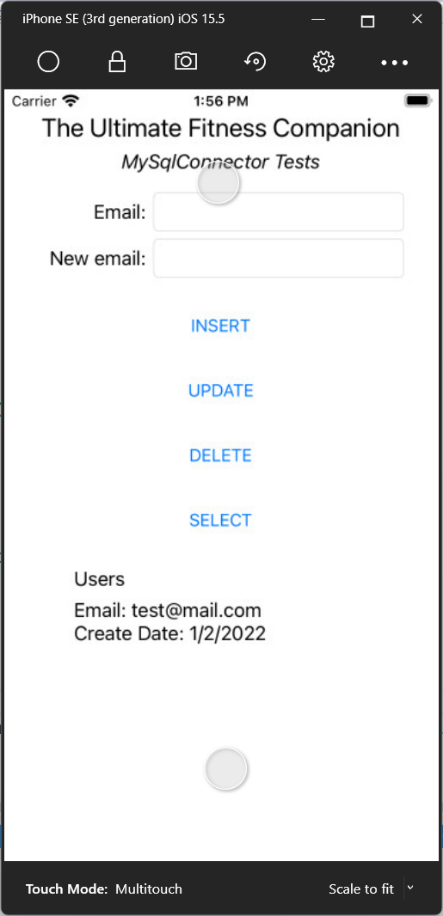
#### Test – Test navigation from ‘MainPage.xaml’ to ‘MySqlConn.xaml’

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 the Xamarin object model ‘User’

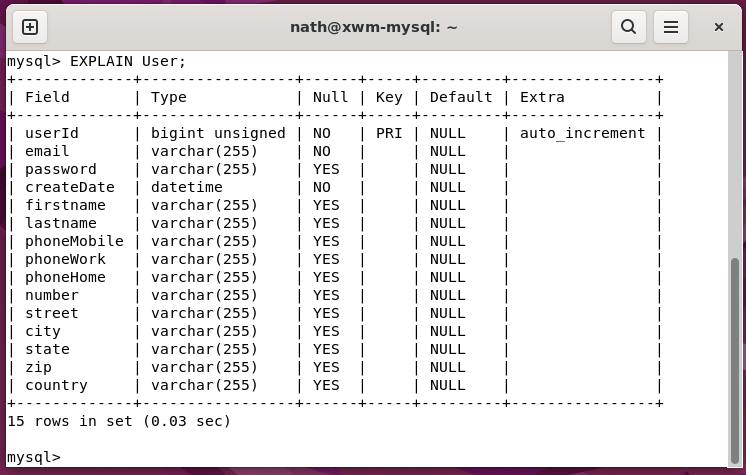
The first object model to create is User

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’

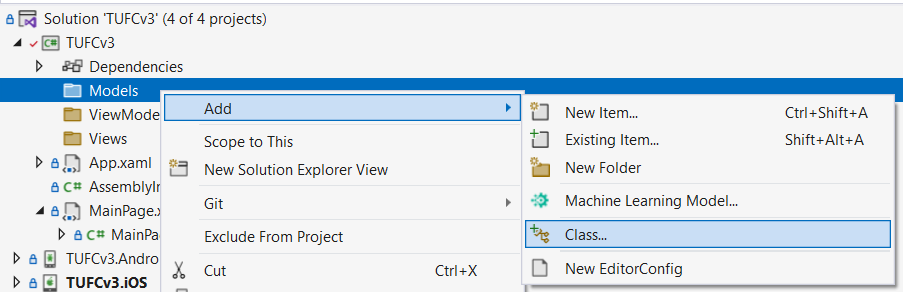


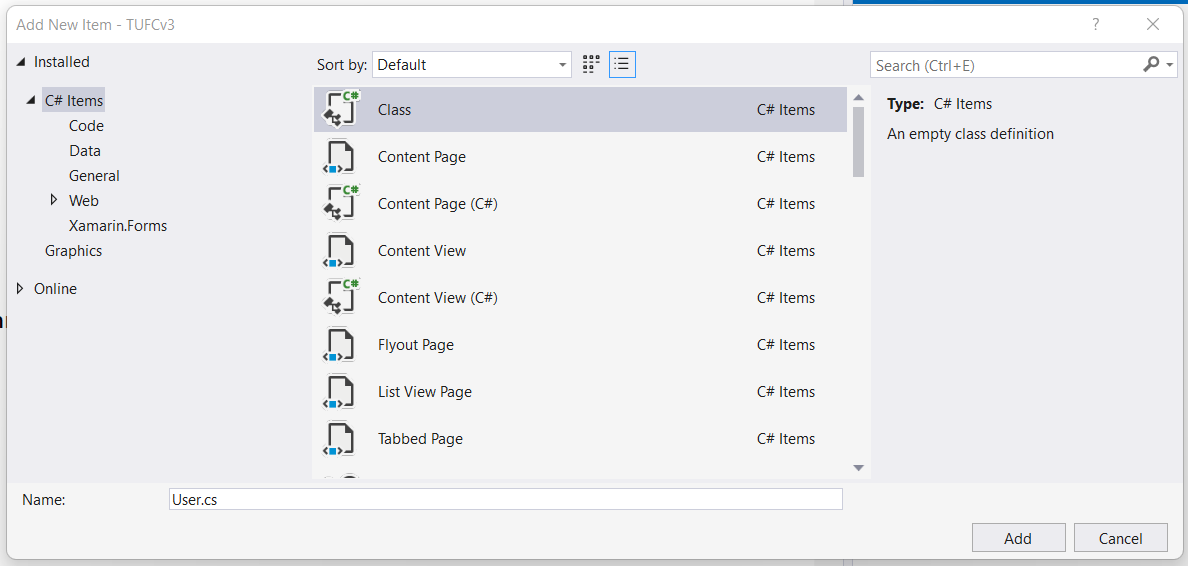
### Create the Class User

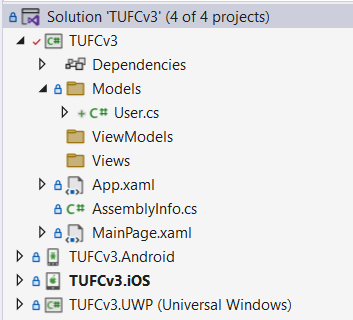
Create the class User

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.

This updates the 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() for ‘User’

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.

Also, when a public property updates

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); }

}

To update both the private variable and the devices screen

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

when a public property updates

#### 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 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

[CallerMemberName] *(highlighted above)* is an attribute

which allows you to obtain the method or property name

of the caller to the method.

You can find this attribute named CallerMemberNameAttribute

under the namespace System.Runtime.CompilerServices

You can apply the attribute [CallerMemberName] to an optional parameter

that has a default value.

You must specify an explicit default value for the optional parameter.

### 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 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;

}