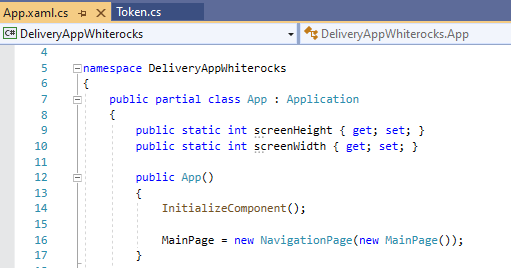
# Developer Notes

Xaml object consists of the .XAML file and .CS file, the InitializeComponent() that is always available in every .CS file will run the XAML part component of the page.

App.xaml.cs is the gateway of the apps, When the application is loaded, I initialize the Navigation page to MainPage.

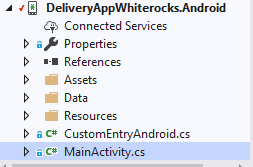
I also did create 2 static properties in App.xaml.cs those variable will be used to store the height and weight of the app.



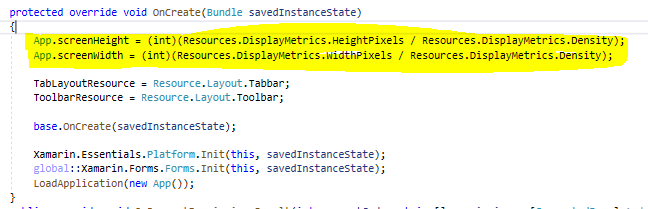
Xamarin.Forms is cross platform framework. It consist of the general part and then platform specific part.



To get each width and height of the app, and pass it into the variable in App.xaml.cs , I have to access the specific platform and MainActivity file



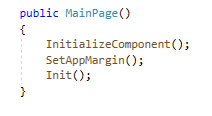
Then add this on OnCreate Method that exist in MainActivity.cs



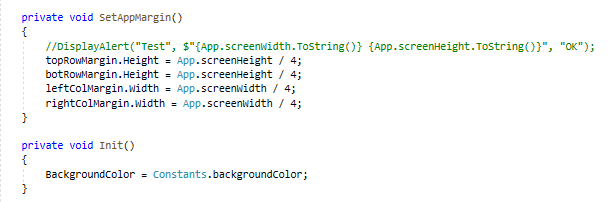
This code will pass in the data into those properties.

# Main Activity

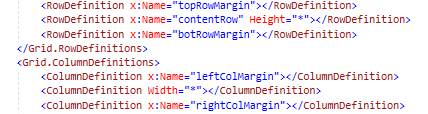
Previously we redirect the app.xaml.cs to point into MainPage.xaml



This is the constructor of mainpage, it will initialize the .XAML component of mainpage, and run 2 other methods. SetAppMargin(); and Init();

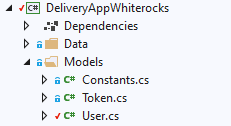


The SetAppMargin method will calculate the margin based on the app.screenHeight and app.screenWidth properties then it will set it to specific variable that exist in the MainPage.XAML

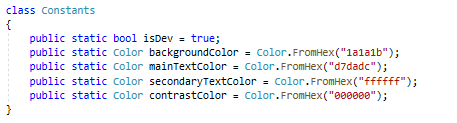


As long as you give them a name by doing x:Name=”Your name” you can access it directly in code

The Init(); function will set the background color of the page from the constants class.



I created constants class within Models folder.



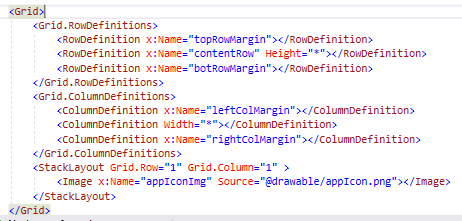
It initialize all properties that could be reused throughout the application.

Because we are adding the component in different folder than the actual application you have to reference it before using the class.



The namespace is DeliveryAppWhiterocks.Models; because it’s located in models folder.

Let’s go back to the MainPage.XAML File



There are different way to dock a component,

The most usual one is by using grid / stack layout , there are plenty more that you could use but let’s focus on this 2

To use them you just have to declare <Grid></Grid> or <StackLayout></StackLayout>

It’s much similar with the web. Unlike the Bootstrap grid that consist of 12 different column the XAML is much more flexible.

You can declare the width of the specific column and row by specifying:

**Numbers**, **\***, or **auto.**

To create a row you must define it. By doing <Grid.RowDefinitions></Grid.RowDefinitions>

Similarly with column you just have to replace the row with column <Grid.ColumnDefinitions> </Grid.ColumnDefinitions>

Inside of those tag you should also declare how many row or columns that you need. In the picture I declared 3 columns and 3 rows. You add <RowDefinition></RowDefiniton> for each row you want, <ColumnDefinition></ColumnDefinition> For each column that you want.

<Grid.RowDefinitions>

<RowDefinition></RowDefiniton>

</Grid.RowDefinitions>

Normally with a row it’s accompanied with Height attribute, for column it’s accompanied with width attribute.

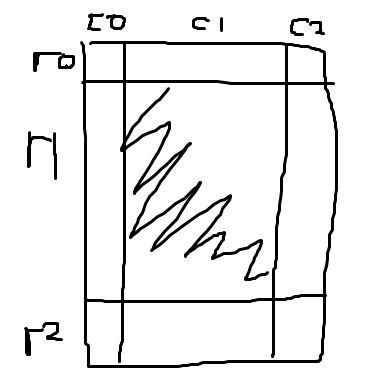
Previously I did mention that there are different way to set up a the width of the column and the height of the row.

Those things are Numeric value, \* and auto

Numeric value will set the row exactly the number , \* will take all the remaining space available and auto will set the number based on the component inside of it, if the component get bigger it will have bigger value, if the component get smaller so will the value.



Then I have this stack layout that is also exist in between the grid, you can nest grid on another grid or other component. This stack layout is being nested inside the grid.row 1 and grid.column 1, remember it starts counting from 0. Because we have 3 column and 3 row. It will results in the center.



Some picture to help visualizing.

Stacklayout works exactly like a stack, the item inside the stack will be placed from top to bottom, it’s the default. But you can change the orientation so it stacks from left to right.

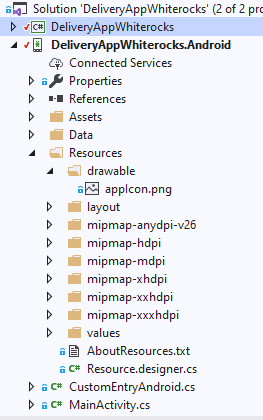


EXAMPLE ON TOP

## image



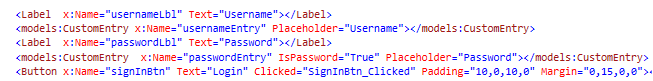
I initialize the image by opening <Image></Image> Tag, the source will be the location of the image in the platform specific path. I just copying the existing app icon with to the drawable folder.



The image in question is inside of Resources/drawable/appIcon.png

# LoginPage

You can find this inside Loginpage.xaml

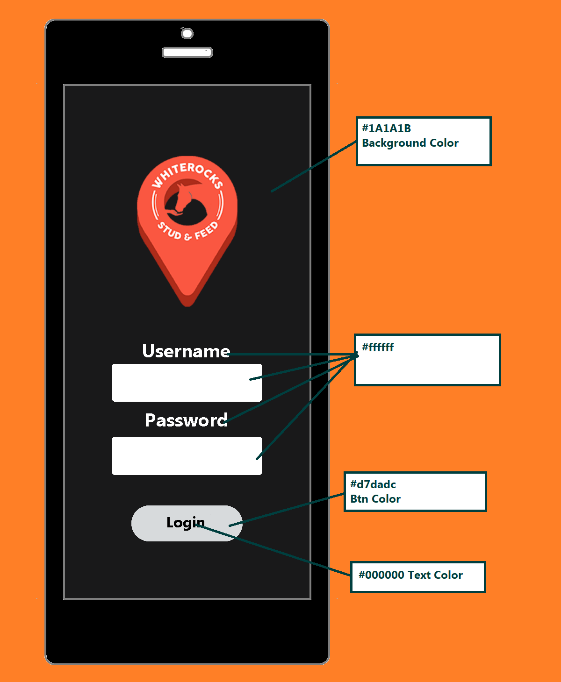


To create a label it’s very similar as what we have learned in the windows.form.app

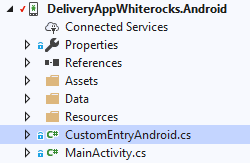
You just create the label by doing <Label></Label> it does have similar properties with the one in windows form.

<Entry></Entry> is the textbox in the windows.form.app

I did not use it because I want my entry to look similar with the wireframe, so I am just creating my own customEntry. The default one from android doesn’t allow for different textbox style.



The entry field is rectangular shape with no underline.

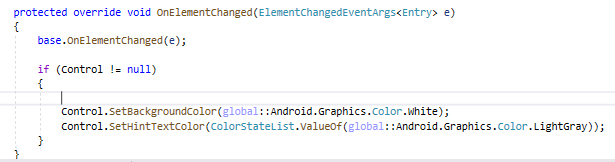


I am creating new .CS file that I called CustomEntryAndroid this will initialize the look of the Entry COMPONENT, remember ENTRY COMPONENT is just a textbox in the windows.form.app

Because I would like to override the component I have to inherit it from the package, the component is called EntryRenderer



And this is the namespace 

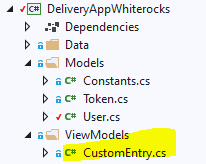


This method will alter the base control of entry inherited from EntryRenderer. The only important part is the setbackground color, in here I just set it to white. By doing this it will remove the underline that always exist and turn the background to whatever color that I want, you could also set the radius here, but since I want a rectangle box I did not do that.

The setHintTextColor is there just for the color of placeholder.



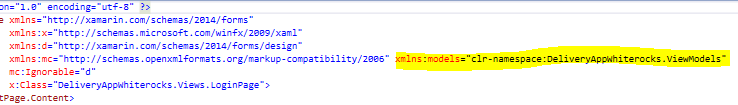
Assembly there is used to call this entry everytime it involves the CustomEntry object, that i’ve created CustomEntry class in the general part.



It’s just an empty class that act as a mediator between the general and the android part

 Since it’s located inside of ViewModels Folder you should add these namespaces in the customEntryAndroid class.

Now the last step is just adding the customEntry reference and component in the .XAML file







# Database

## Gmail

The gmail :

USERNAME: [Whiterockstud2Feed@gmail.com](mailto:Whiterockstud2Feed@gmail.com)

PASSWORD: DeliveryFeed1

## REMOTE DATABASE USED

Heliohost.org

USERNAME: nzdas123

EMAIL: Whiterockstud2Feed@gmail.com

Password: DeliveryFeed1

REMOTE CONNECTION HOST SERVER: 65.19.141.67

Database Name: nzdas123\_deliveryDatabase

Database User: nzdas123\_deliveryPerson

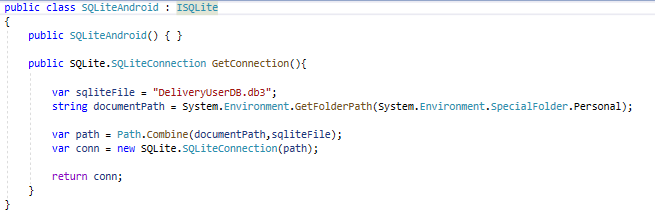
Database UserPassword: DeliveryFeed1

# Login System

## SQLite

We plan on making the application to be able to run offline as well, the user will be able to login if they have already logged previously. the sqlite will store the user information locally on their device.

I have created the ISQLite interface that implements the GetConnection function. Then I am using it in the android platform by creating a class in the android specific platform that I then called SQLiteAndroid.



The method of GetConnection in the android platform is used to get the location of the database that might have or might not have existed yet. It’s similar as what we have learned in 607 Class



For sqlite connection string to work all it needs is the location of the database.



The above assembly dependency is placed above the SQLiteAndroid class so that we could retrieve the method of the SQLiteAndroid of GetConnection(), so any call to the interface ISQLite class is pointed the specific platform

It could be observed below, the constructor userdatabasecontroller is running the dependency service that is calling the getconnection of the platform specific data.

