Minimalistic Flat Modern GUI Template

Documentation

Minimalistic Flat Modern GUI Template

Documentation

Section

AUTHOR

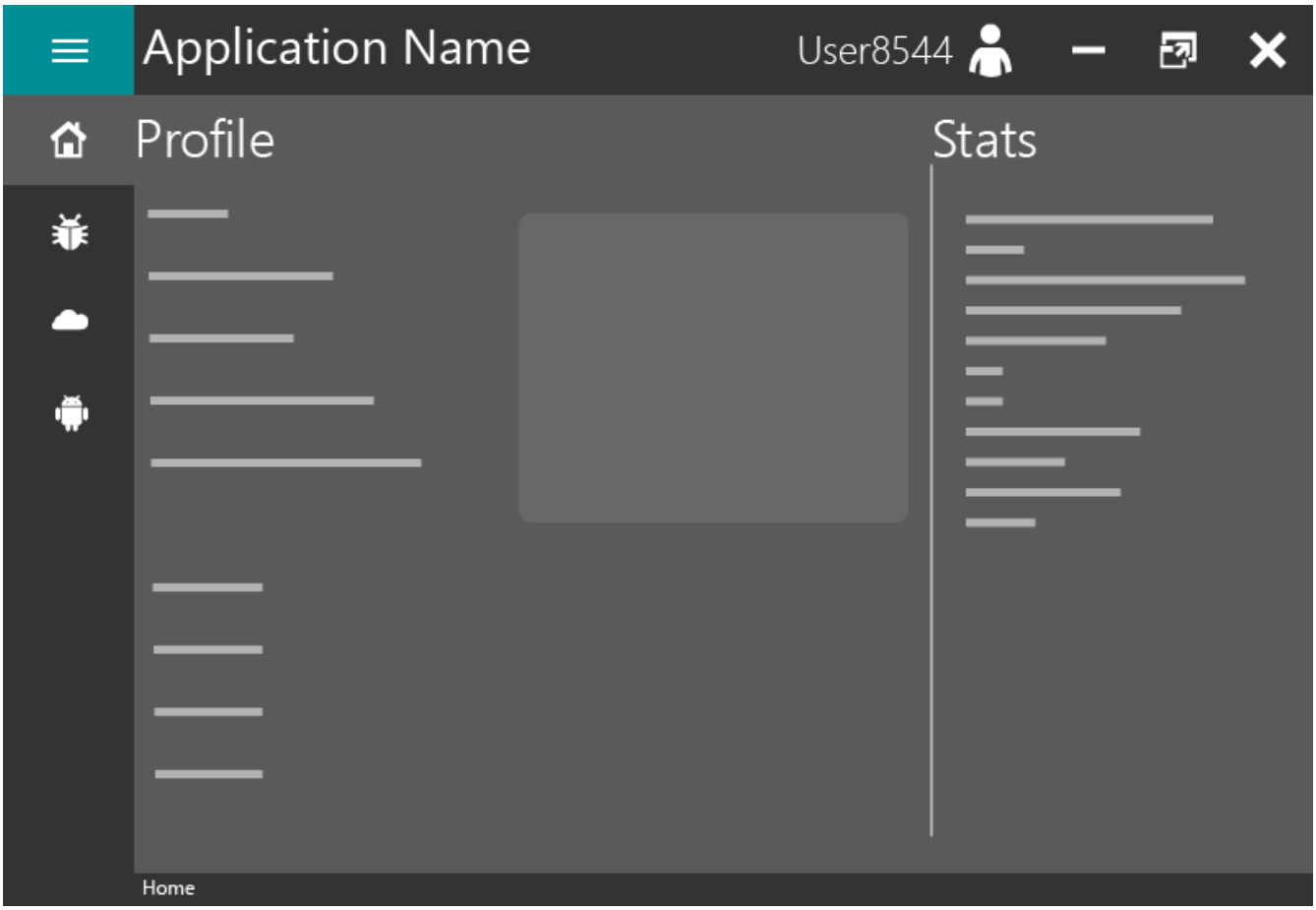
😃 Hi am Anjal.P, from the beginning of coding I was not a frequent GUI user, and I am quite new to this space, but coding different projects in CLI made to rethink, whether to make the same application in GUI or not, an initial attempt was unsuccessful, but finally, I made to crack the PySide2 GUI learning. My way was tough as many users don't code real GUI completely in Python, but the ease of which I found in making a GUI application was worth the time. From creating my first GUI application to this project, I was somehow interested in making the GUI looks better with some modern touch to it, with my knowledge in Adobe Illustrator, I simply began to make some initial designs in Illustrator, which motivated me to make a painstaking task of creating a GUI with all custom component.

As I proceeded, my try searching the internet, found lots of GUI application, but none of the templates was found, then I thought why not create the GUI template and so the developers can simply make use of this template in their next project without investing much time on the looks, this project aims to solve this issue. You can just open the `.ui` file and modify the looks, code the functions, and integrate with your application in the `main.py` and `ui_function.py` file, and that's it you made a simple yet powerful GUI application.

LICENSE

INTRODUCTION

A **Free** to use, Beautiful, Feature Rich, Fully Customizable Flat Modern GUI Template Using **Pyside2** designed in **Qt Designer**, supported for Windows/Linux/Mac OS, Incorporating widgets like: Buttons, Progress Bar, Custom Tabs and many more.



Check below to see more Images of the GUI.

Highlights

- Flat Minimalist Design.
- Dual Tone Color Scheme.
- Dedicated Menu Bar.
- Dedicated About Page.
- Toggle Menu Button.
- Custom Top Bar With Custom Minimize, Maximize, Restore and Close Buttons.
- Custom Widgets:
    - Push Buttons
    - Progress Bar
    - Radio Buttons
    - Check Boxes
    - Text Edit Field
    - Horizontal and Vertical Sliders

    - Horizontal and Vertical Scroll Bar
    - Combo Box.
- Custom Dialog Box with customizable Heading, Message to Display, Buttons to Display.
- Custom Error Box with customizable Heading, Message to Display and Button.

Prerequisites

- Intermediate Python User
- Know about Pyside2/PyQt5/PyQt4 or has been using any other GUI package(refer to Resource section).
- Install PySide2, Qt Designer
- Comfortable in using Qt Designer.

  If you are completely new to GUI in python, then I suggest you to quickly head forward to the Resource section for help.

Quick Start

Clone/Fork the Repository to your PC, open the `/exe` folder, and run the `App.exe` file to experience the GUI in a glance.

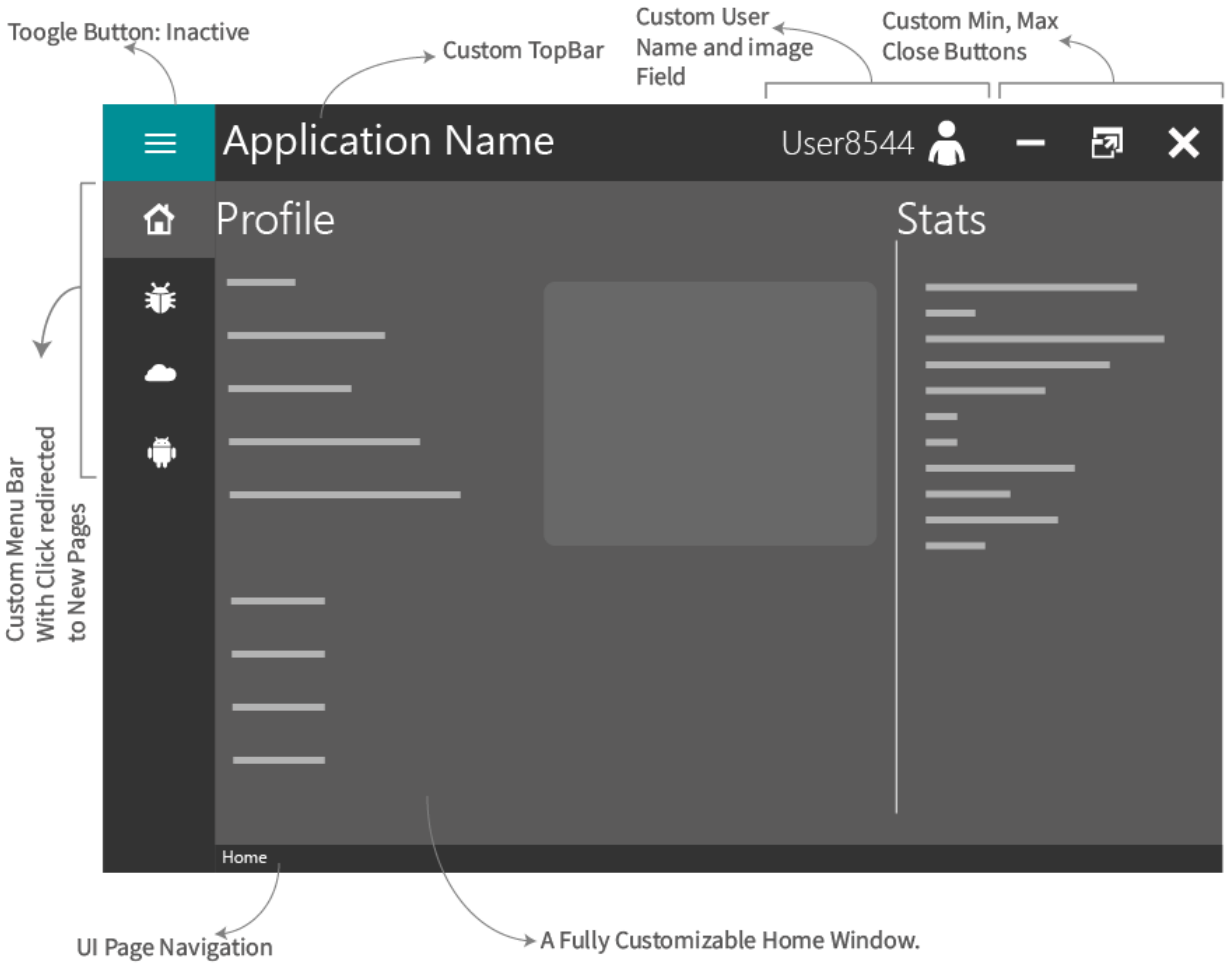
- If you have installed the PySide2, then try running the `main.py`

  ```
  C:/User/home/minimalistic-flat-modern-ui>python3 main.py
  ```

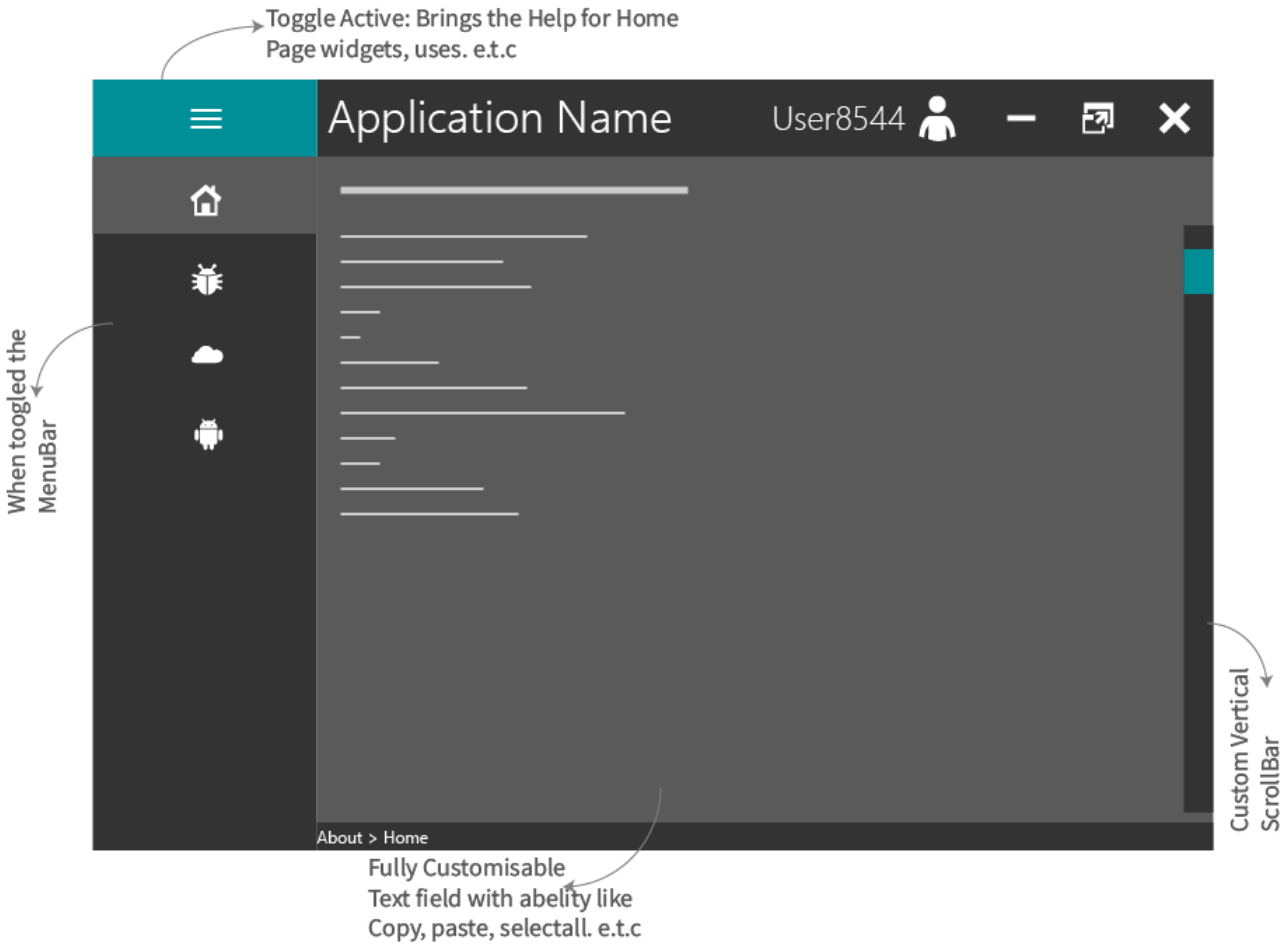
  from your favourite terminal.
- To modify the Main Window Design open the `ui_main.py` file in **Qt Designer**, make necessary and then action of the widget can be coded in the `main.py` and `ui_function.py` files .
- To Modify the Dialog Box Design open the `ui_dialog.ui` file in Qt Designer, for coding related to Dialog Box move to `class` `dialogUi` in `main.py`. Same applies for the Error Box.
- All the Custom Stylesheet used in this Project is provided in the `Documentation.pdf` file. Use them where ever required.
- Check the `images` folder to see the GUI Images.

GUI Widgets and Controls

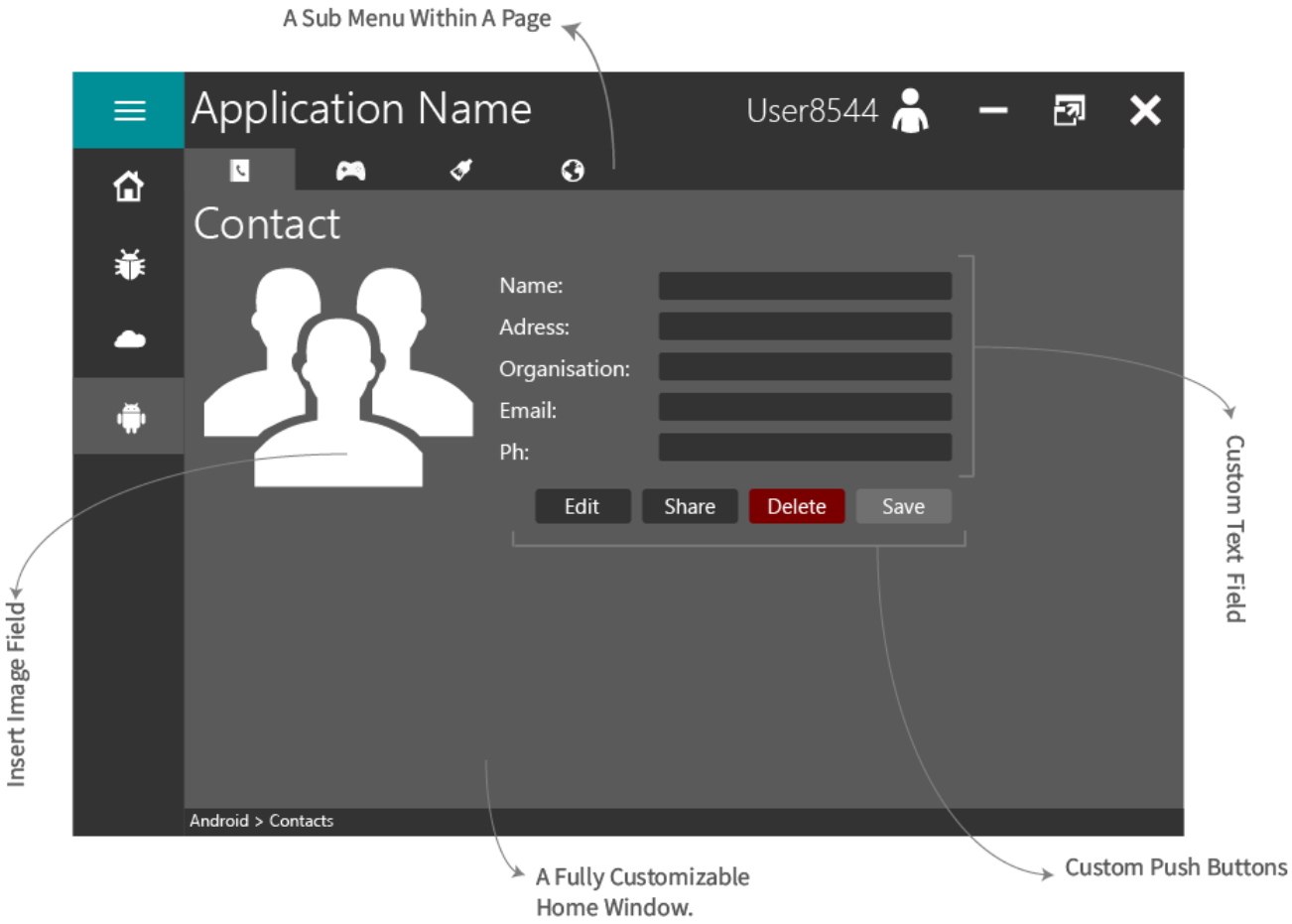
Home Page



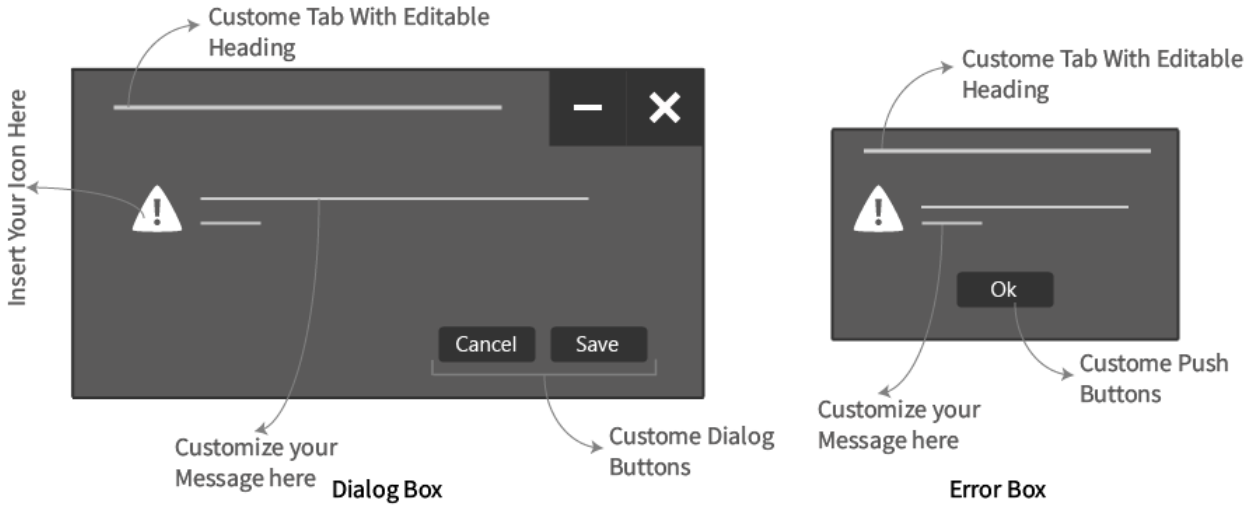
Toggle About



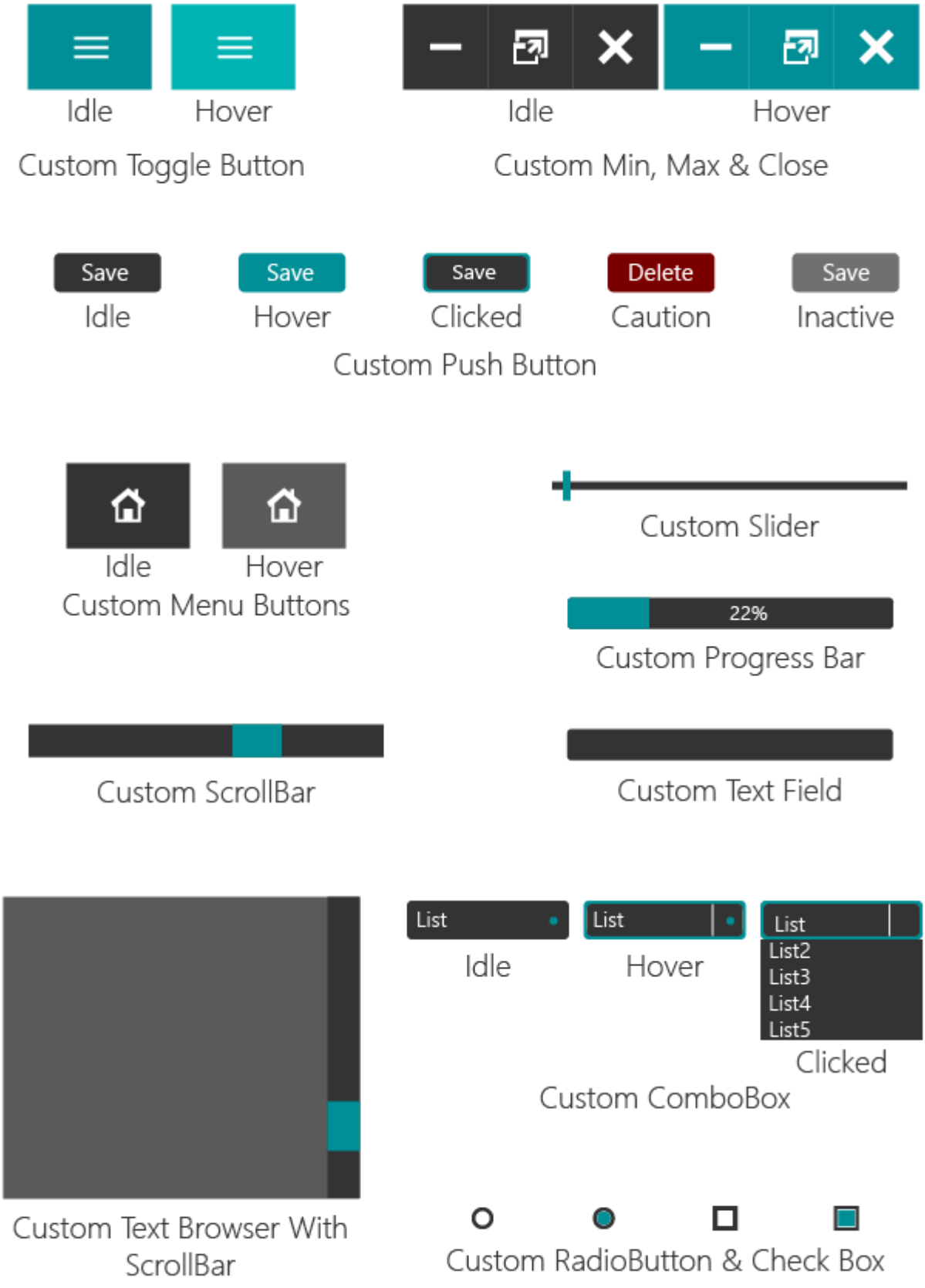
Page Within a Page



Dialog Box and Error Box

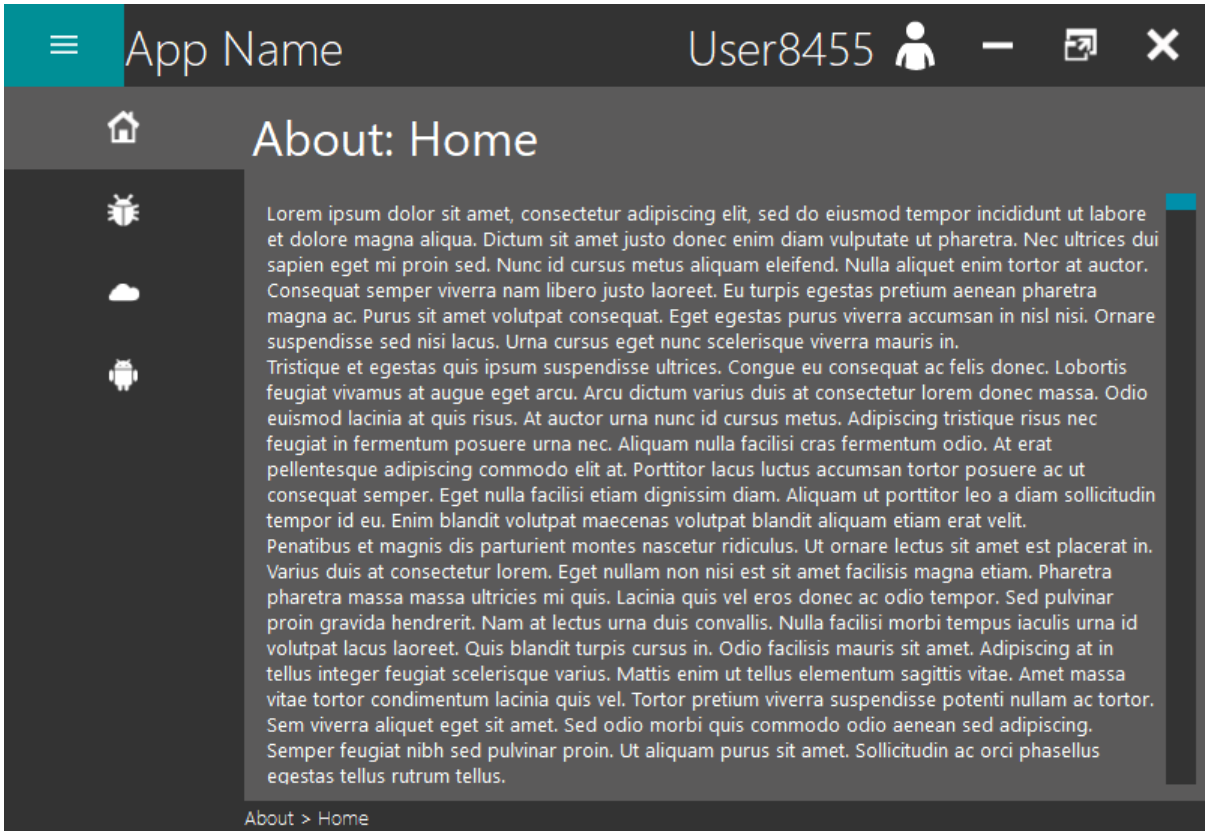


Widgets

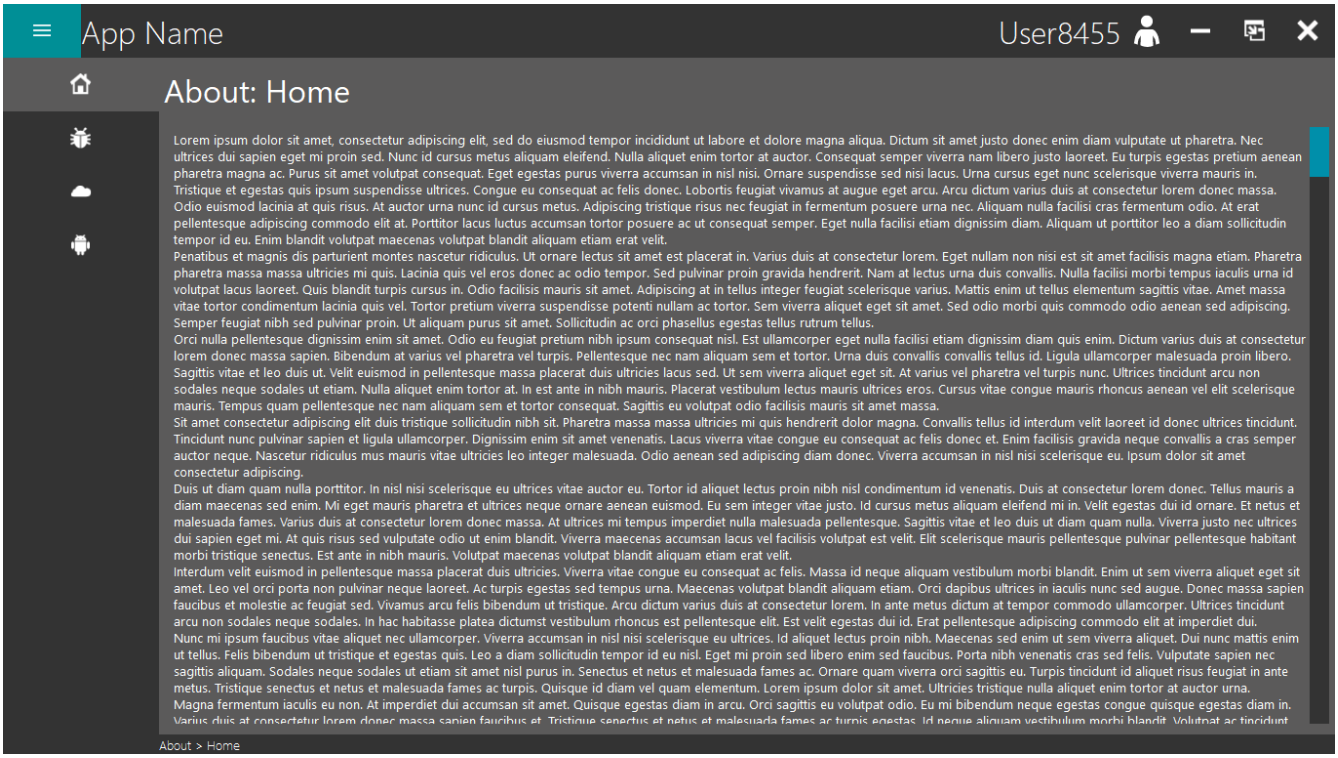


Stock Size and Full Screen

Stock Size



Full Screen



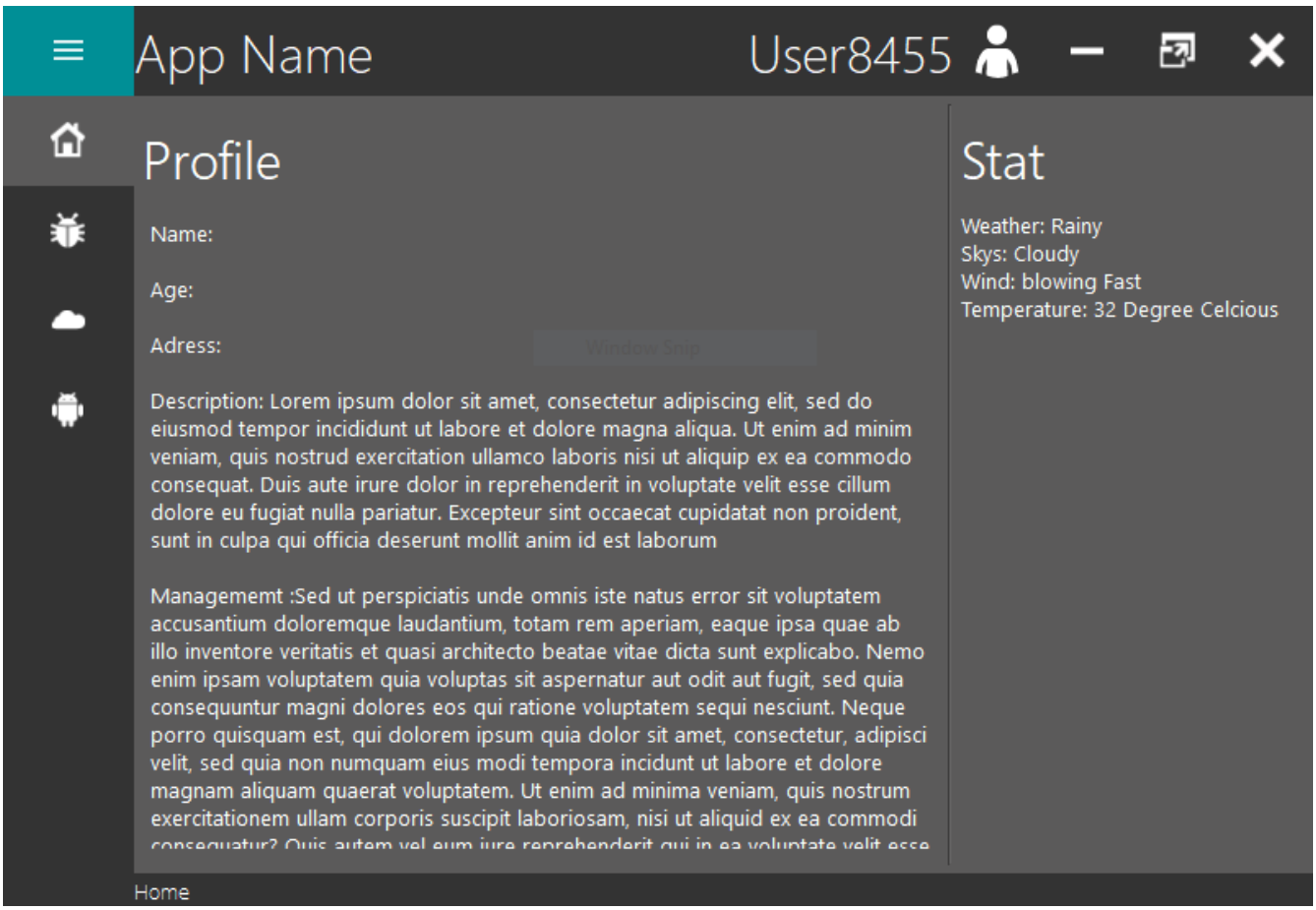
Resources

- Python Library used: Pyside2
- Qt Designer : Download
- Python Basic Pyside2 Programming:
  - Parwiz Forogh PySide2 GUI Tutorial in his YouTube Channel: **One of the Best for Beginners**.
  - GeekForGeek: best Guides for PySide2/PyQt5.

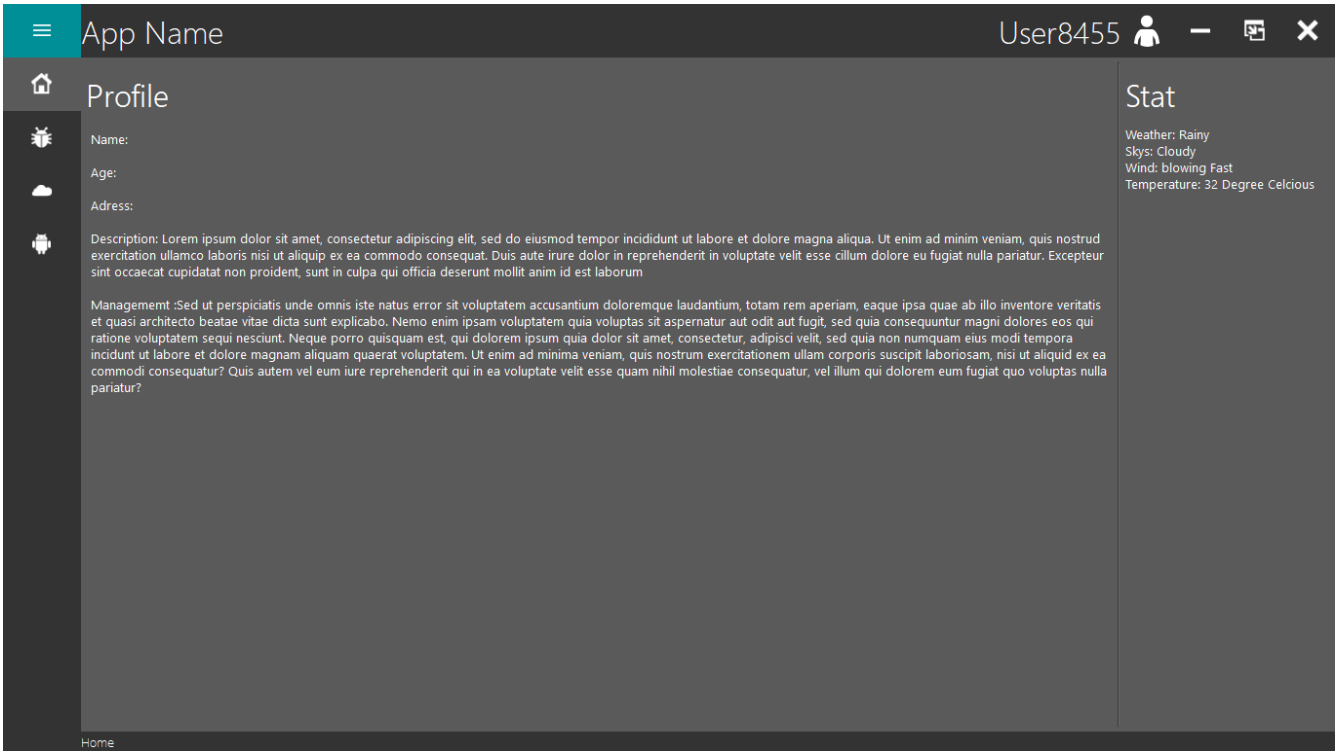
    - Tutorial Point PyQt Tuturials: Even though they are for the PyQt4, all the code works same(except some).
- Pyside2 Stylesheet Documentation: Qt For Python

Gallery

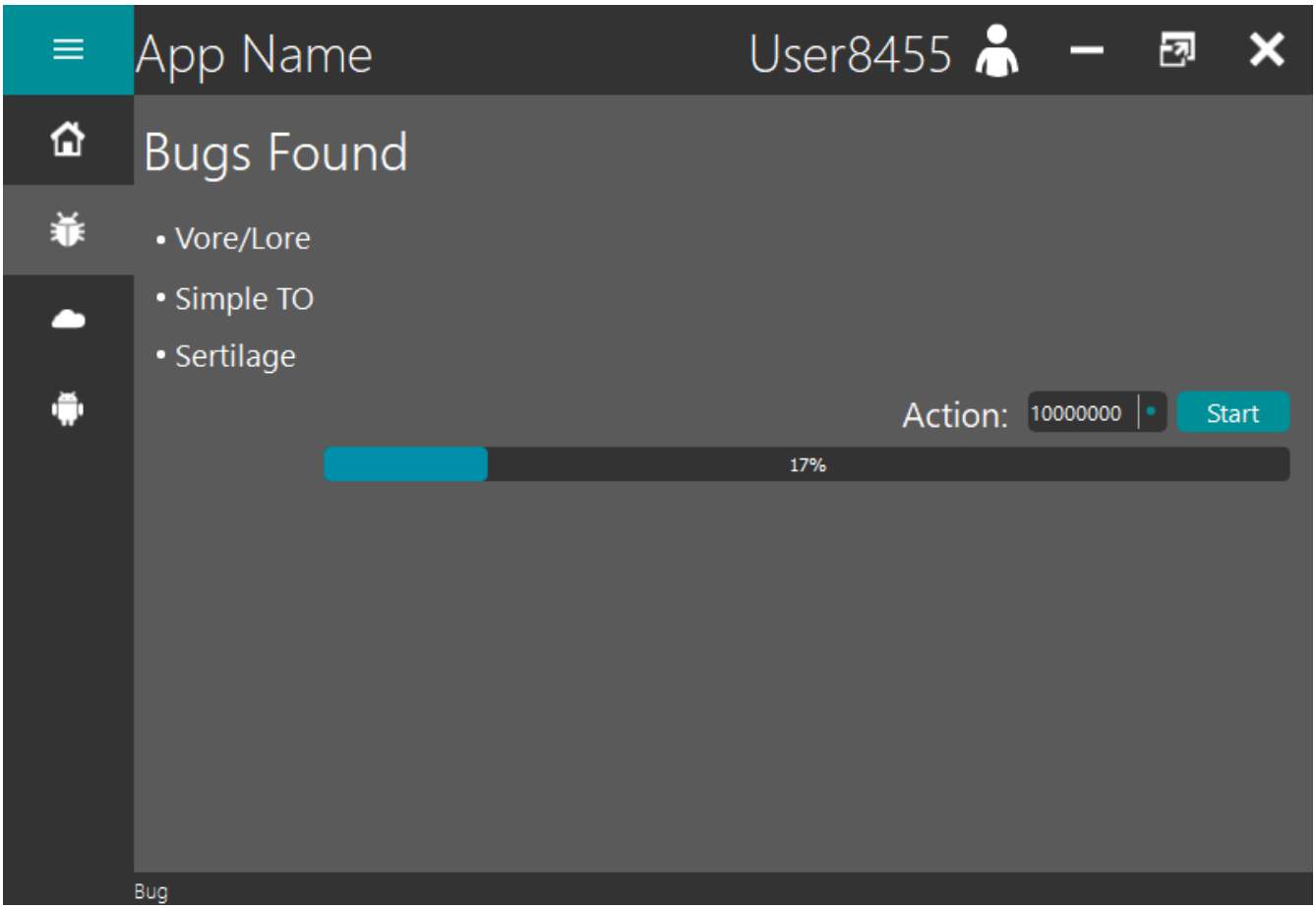
UI Home(view all your profile here):



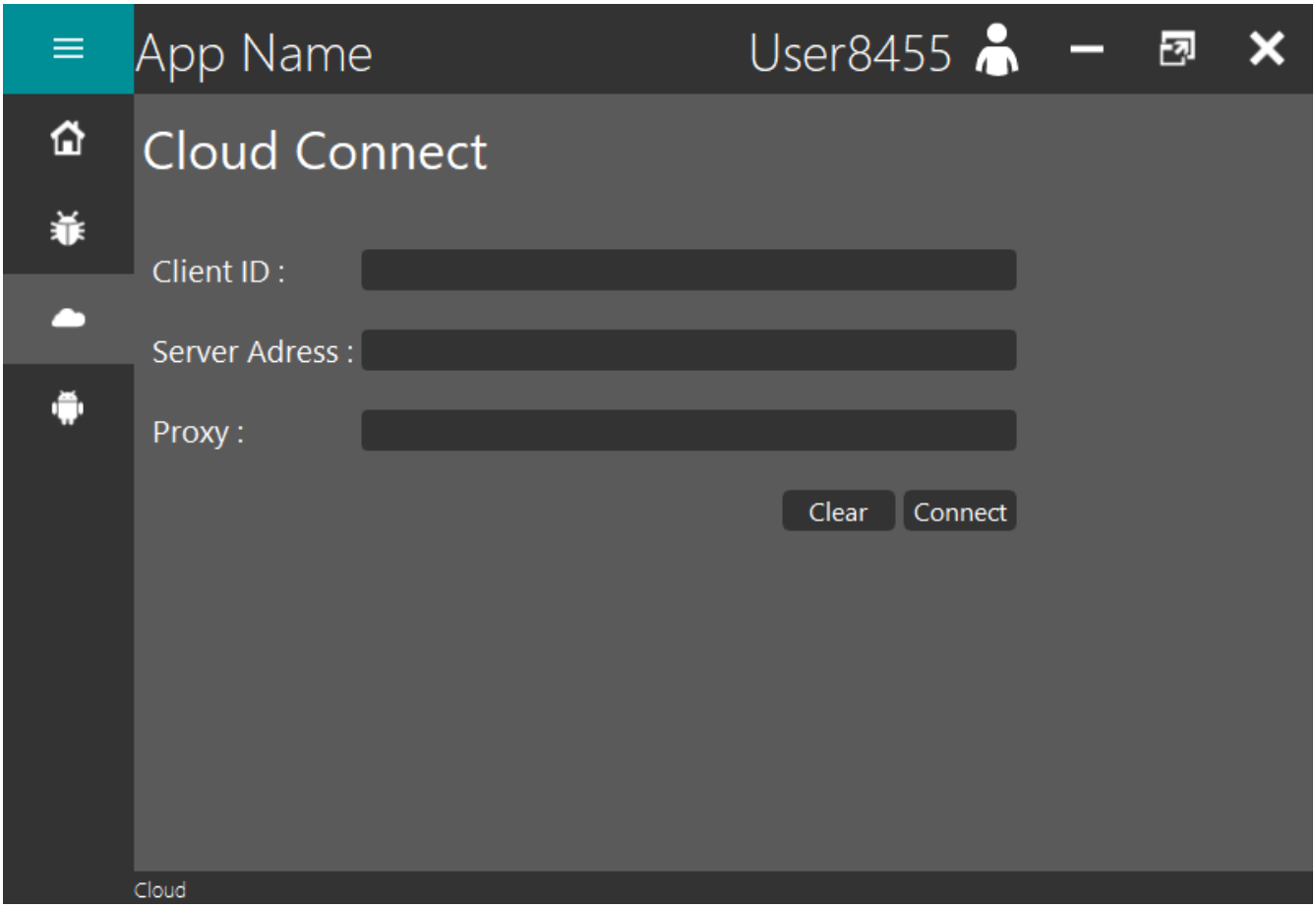
Home Full Screen:



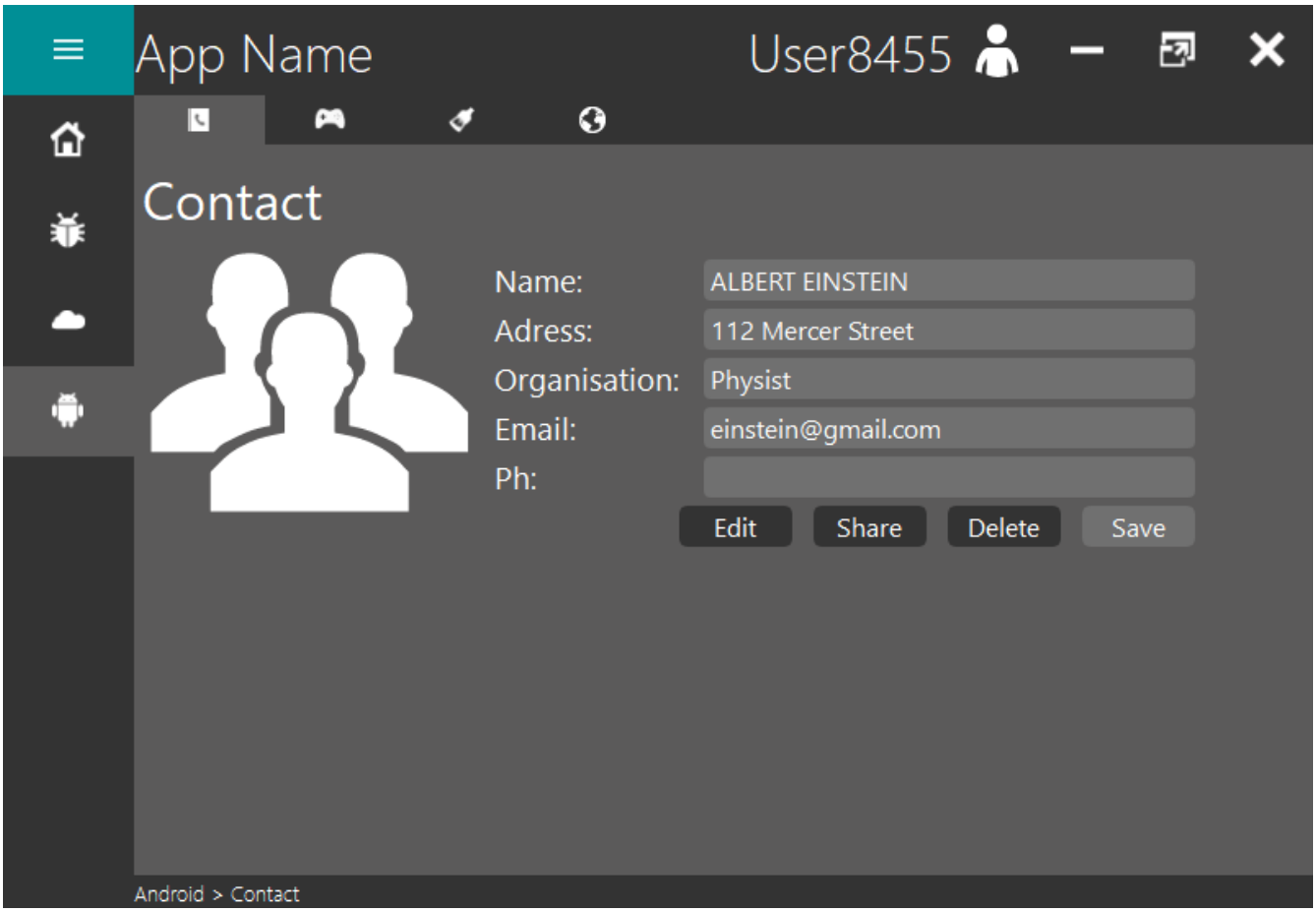
Page Bug(with PushButton, progress bar, combobox):



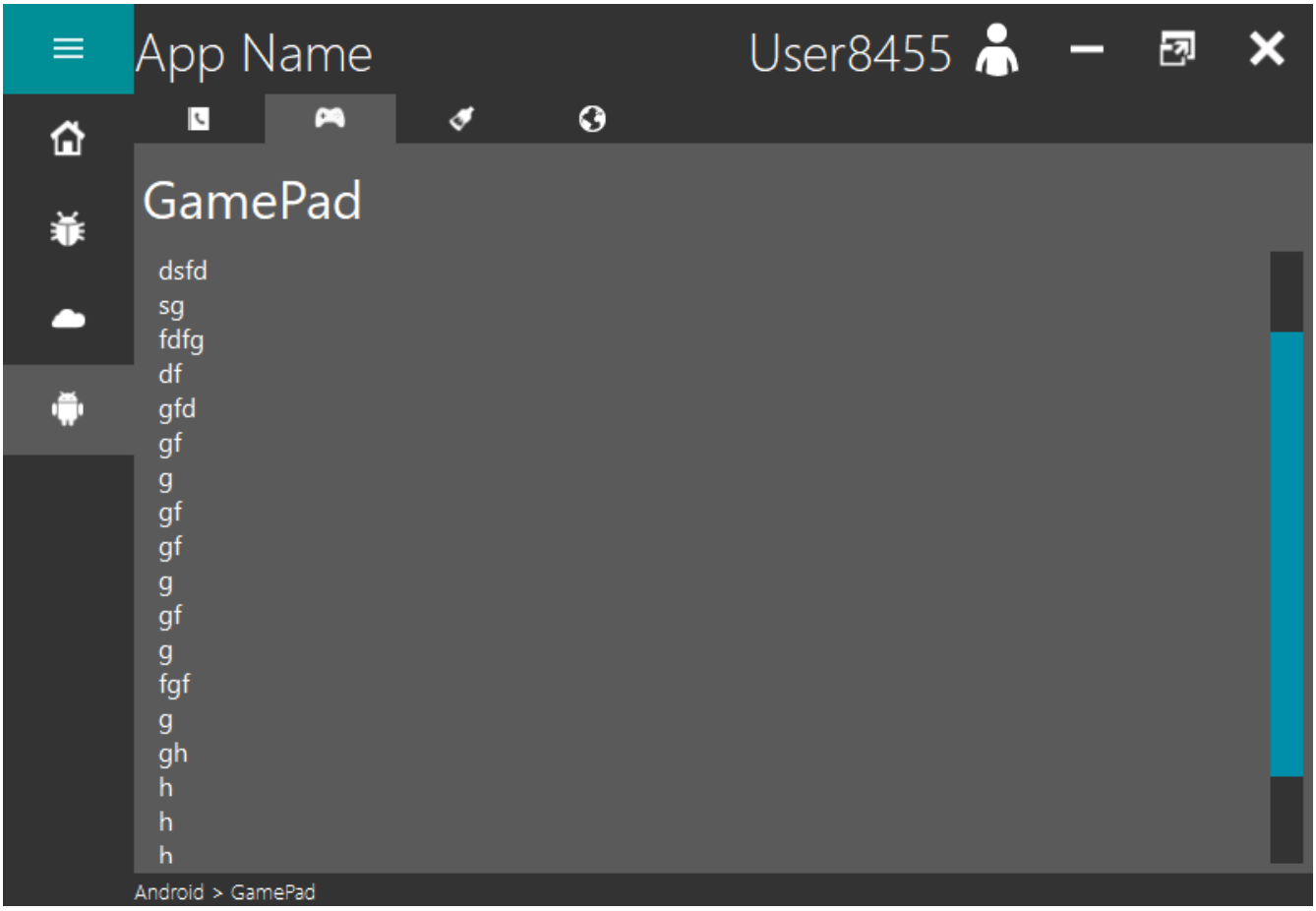
Page Cloud(with text field):



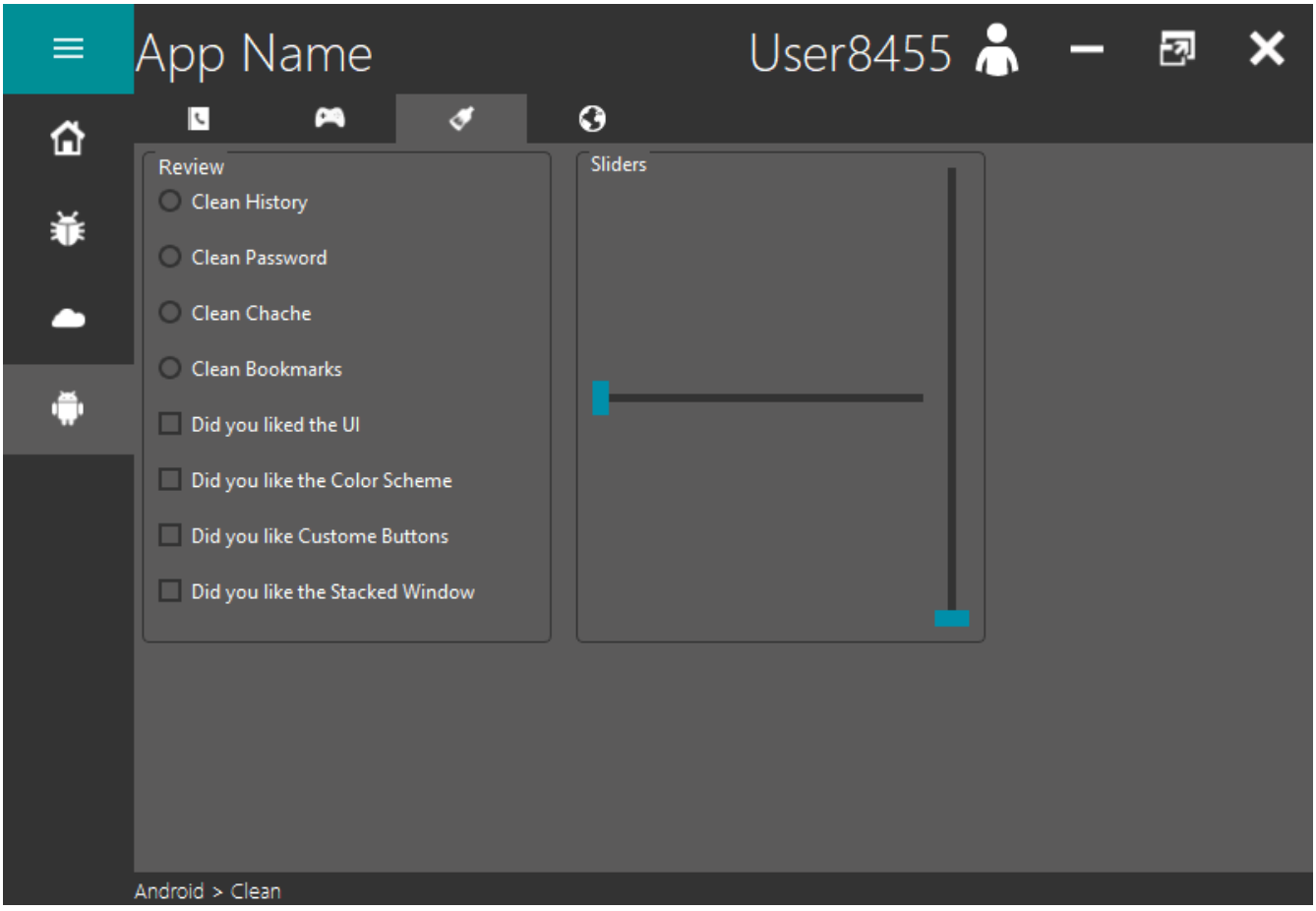
Page Android(stack widget within a widget):



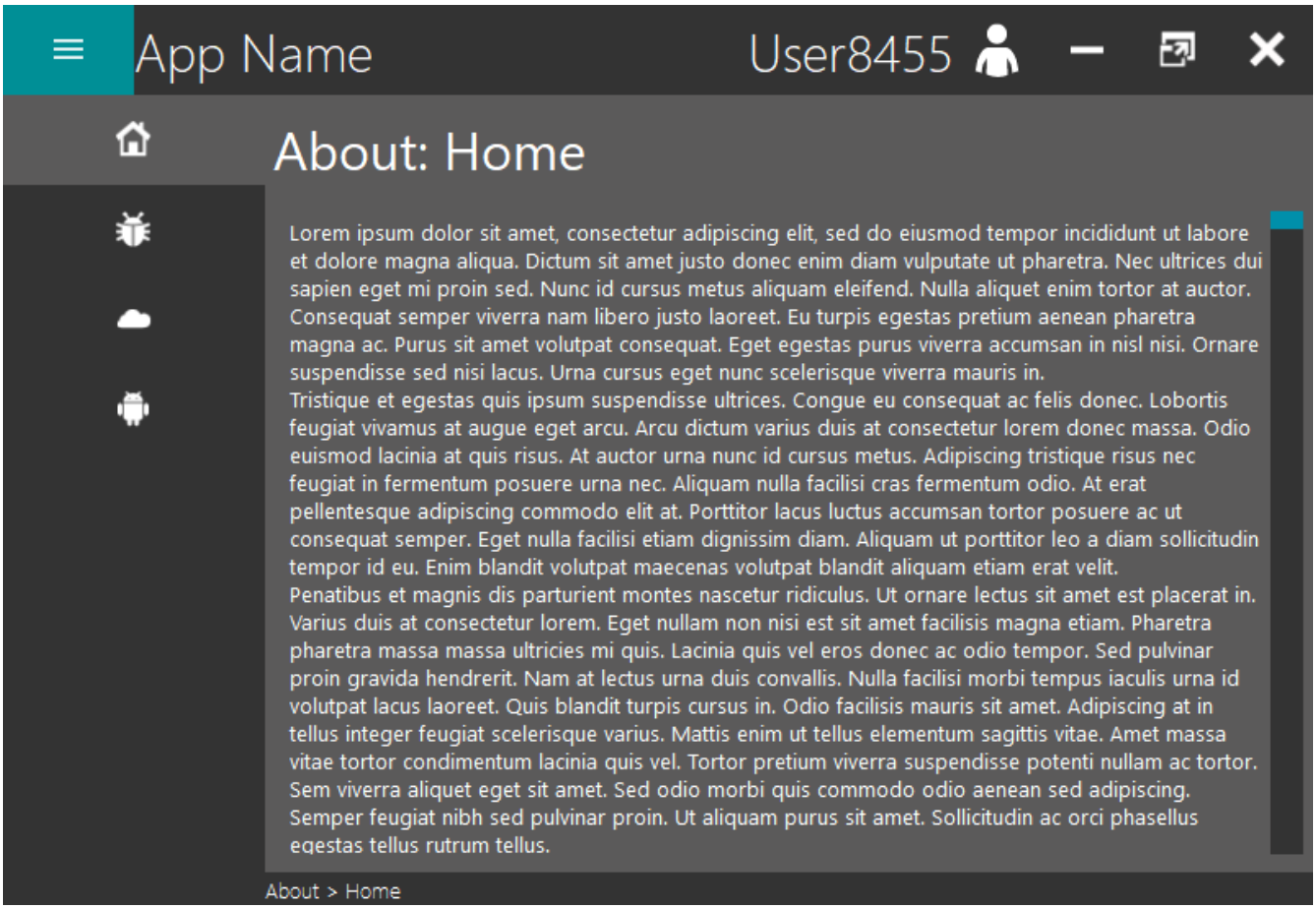
Page Game(within main Page Android)(with text browser and scroll bar):



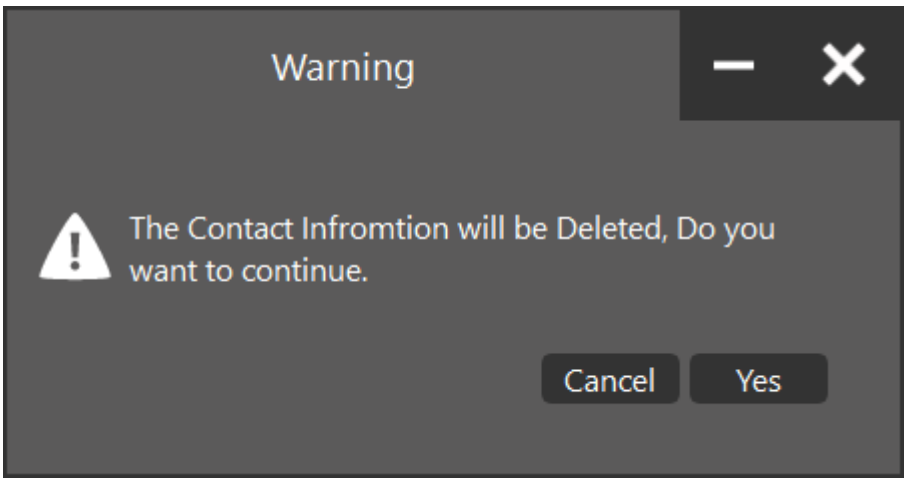
Page: Clean(with checkbox, radiobutton and sliders):



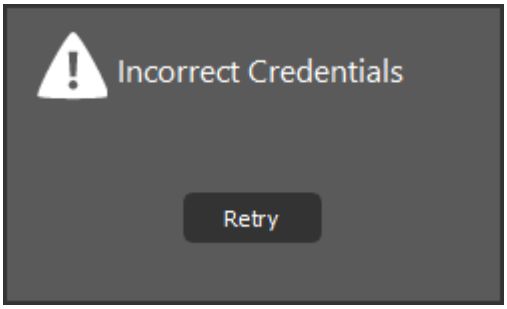
Page About(toggle button:)



DialogBox:



Error Window:


Incorrect Credentials
Retry

My Approach To GUI Design

GUI Design is a tedious job, especially if you are developing a PC software as there involves coding for different OS in the market and fewer tools for you to get-go. That's where the Qt framework comes in, it is supported in the two most powerful languages C++ and Python, can run in Windows, Linux, macOS, Android. Since Python has a couple of GUI designers like Tkinter, Kivy, and many more, Qt stands out as it has vast documentation and a large community to support it.

What I like most about the Qt is that the designer it comes with, which makes the GUI design just like a drag and drop process, and this gives you a big relief in arranging the widget across the window. This ease to construct a GUI makes men like me to customize the GUI in terms of looks and create custom widgets, new layouts, and much more. My GUI creation routine consist of mainly 3 processes:

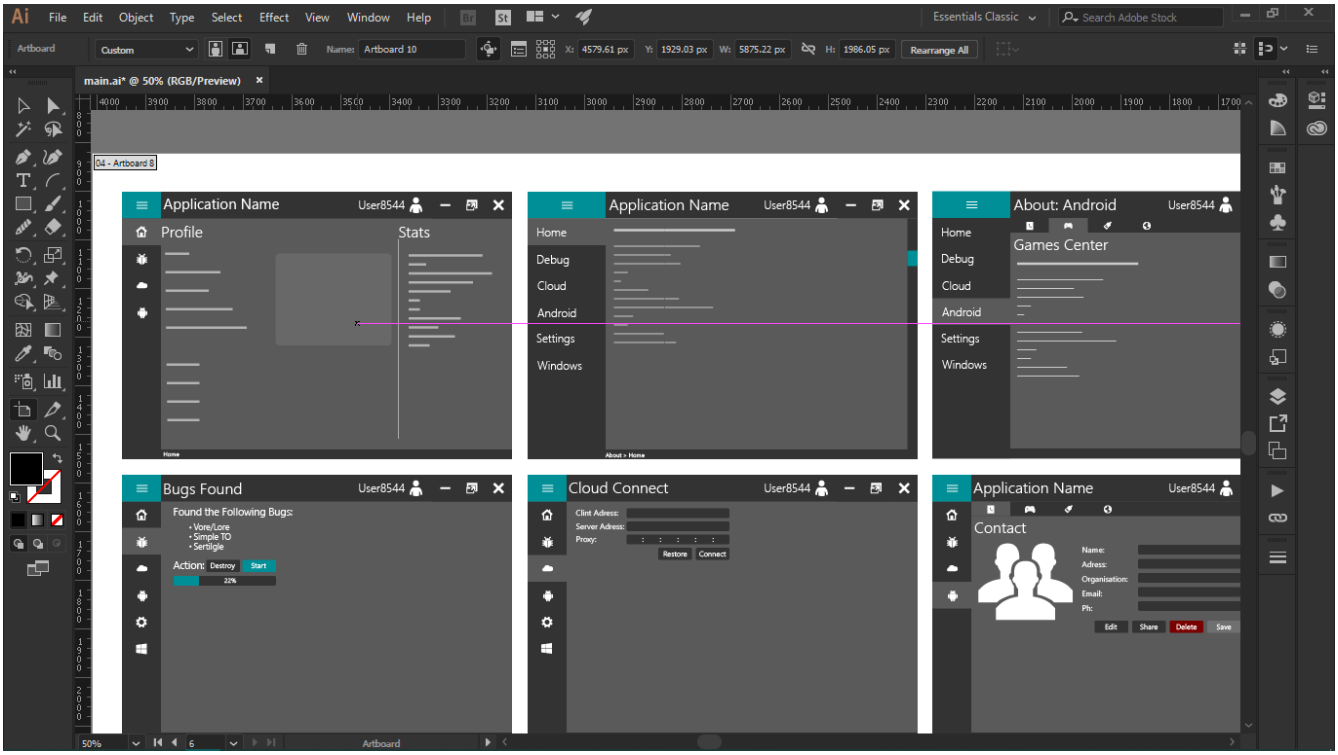
- Make a sketch of the overall GUI based on the application you want to run it.
- Implement the GUI design you made in a sketching application(my favorite: Adobe Illustrator).
- After confirming the design, make the idea a real application using Qt Designer and Python.

Sketching in your mind is the process where you think what you want in your application like: how many buttons, how many windows, text field required, and much more. Then our next process of creating the GUI using a designing software, this process is a long one where you spend time on where to place the button, which color do I use here, trying a different style of the widget and in this process you come to a conclusion of a GUI.

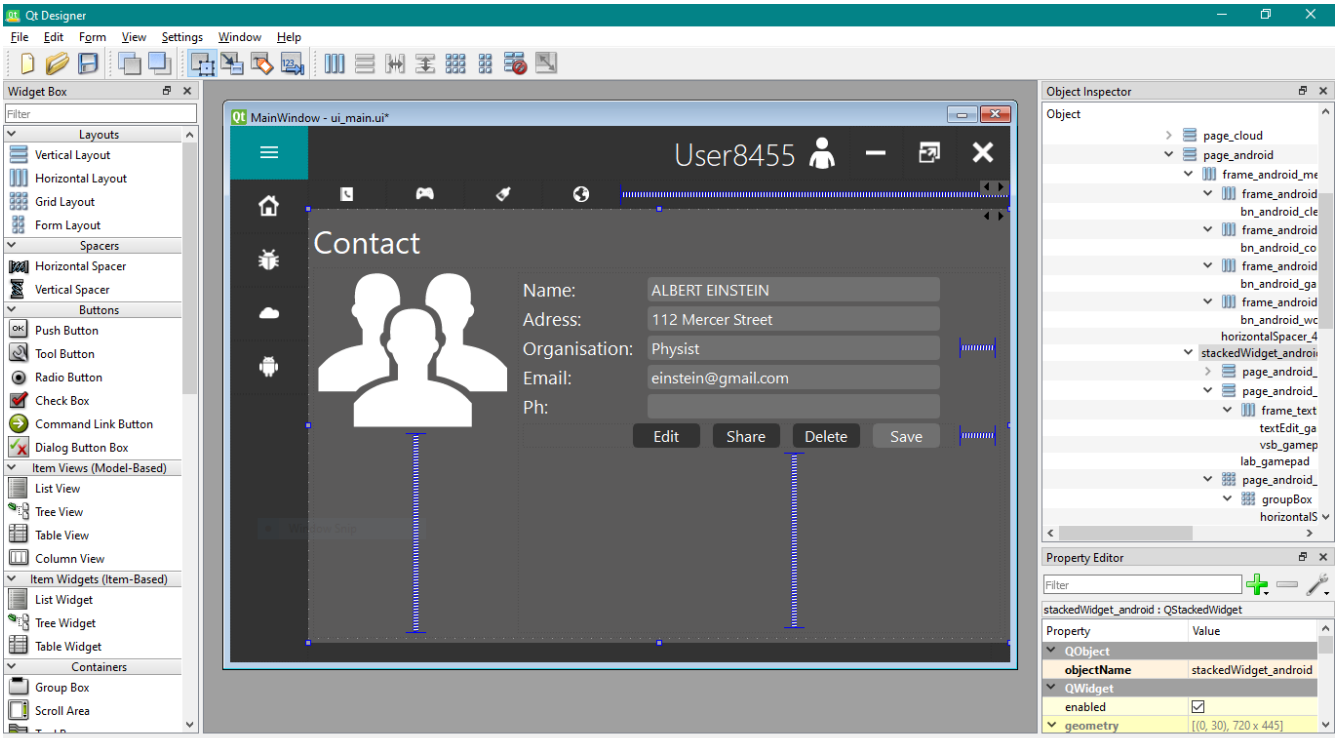
The final task is that you implement this final GUI design to a real application using the Qt designer, Python. This way it makes us follow the GUI we designed in the designer and can result in a completely easy, satisfying GUI making process. I mainly use pen and paper method to sketch the GUI in my mind, Adobe Illustrator is a great way to design the GUI, but it is a paid one, it need not like this alone, you can go for a free yet one of the best Inkscape, which is available for both Windows and Linux.

Photos:

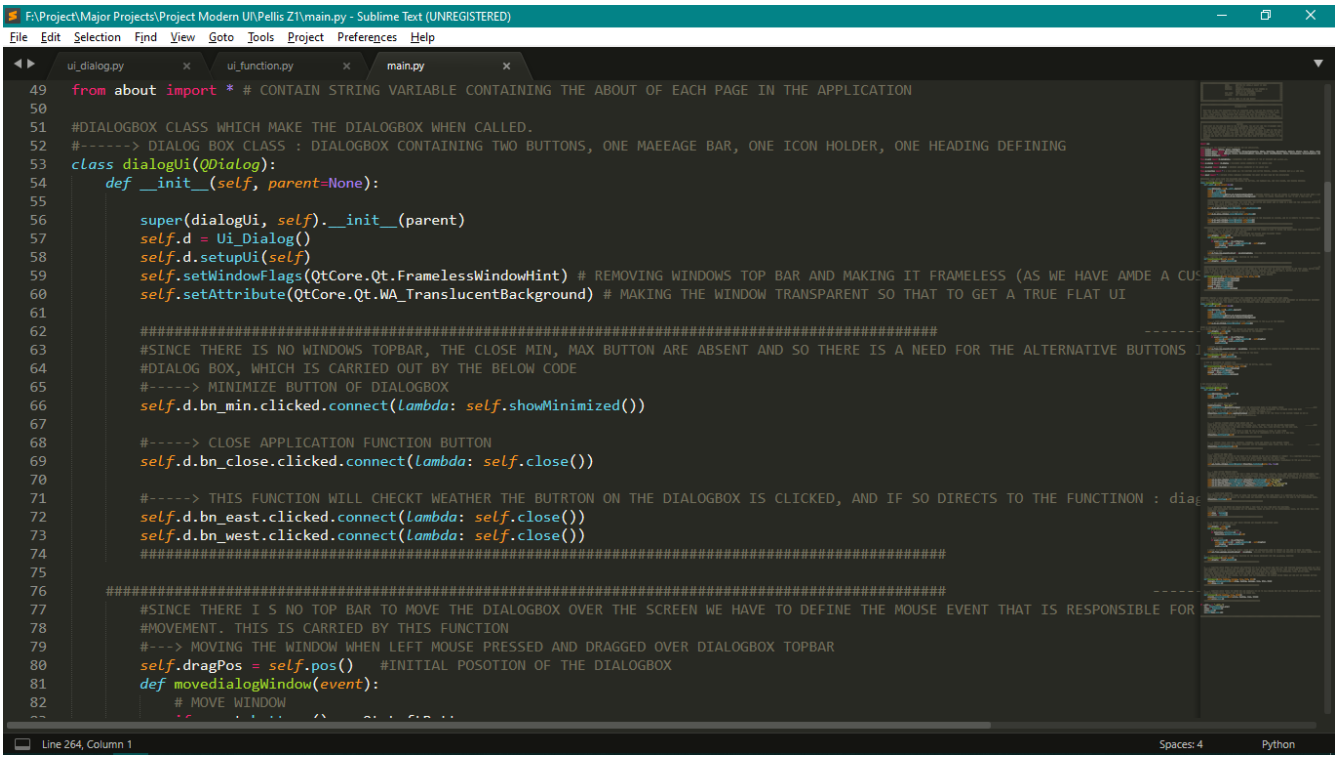
Working on GUI design in Illustrator



Recreating the GUI design in Qt Designer:



Coding the stuff:



I know this seems a long process, but trust me this is the best approach to GUI design in terms of producing better design.

In terms of designing of the GUI in Qt designer, i used to do all the stuff including the size, icons, label, style sheet, position, arrangement in Qt designer itself so that i can focus more on the functioning of these widget in the coding side.

Finally the GUI has the following files as the output:

- `main.py`
- `ui_function.py`

- `ui_main.py` generated from `ui_main.ui` file using *pyside2-uic*
- `ui_dialog.py` generated from `ui_dialog.ui`
- `ui_error.py` generated from `ui_error.ui`

`main.py` file is where all the code for the main window, dialog box, error box resides, it is also where the input entered by the user is registered. The `ui_function.py` is place where the action of you application takes place, this layout makes the code much more readable. You can place your codes for your software solution here.

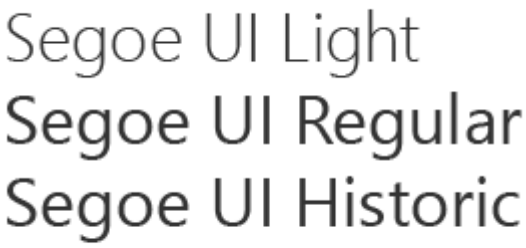
Opening the `ui_main.ui` in the Qt Designer help you with the modification of the GUI.

Layout Of GUI

To make use of the GUI as fast as possible and with ease, I have made a layout of the whole GUI containing all the frames, widgets in an infographics way with its object name used. This makes your way to find the object name of the particular part(widget or frame) in GUI by just looking into the below diagrams.

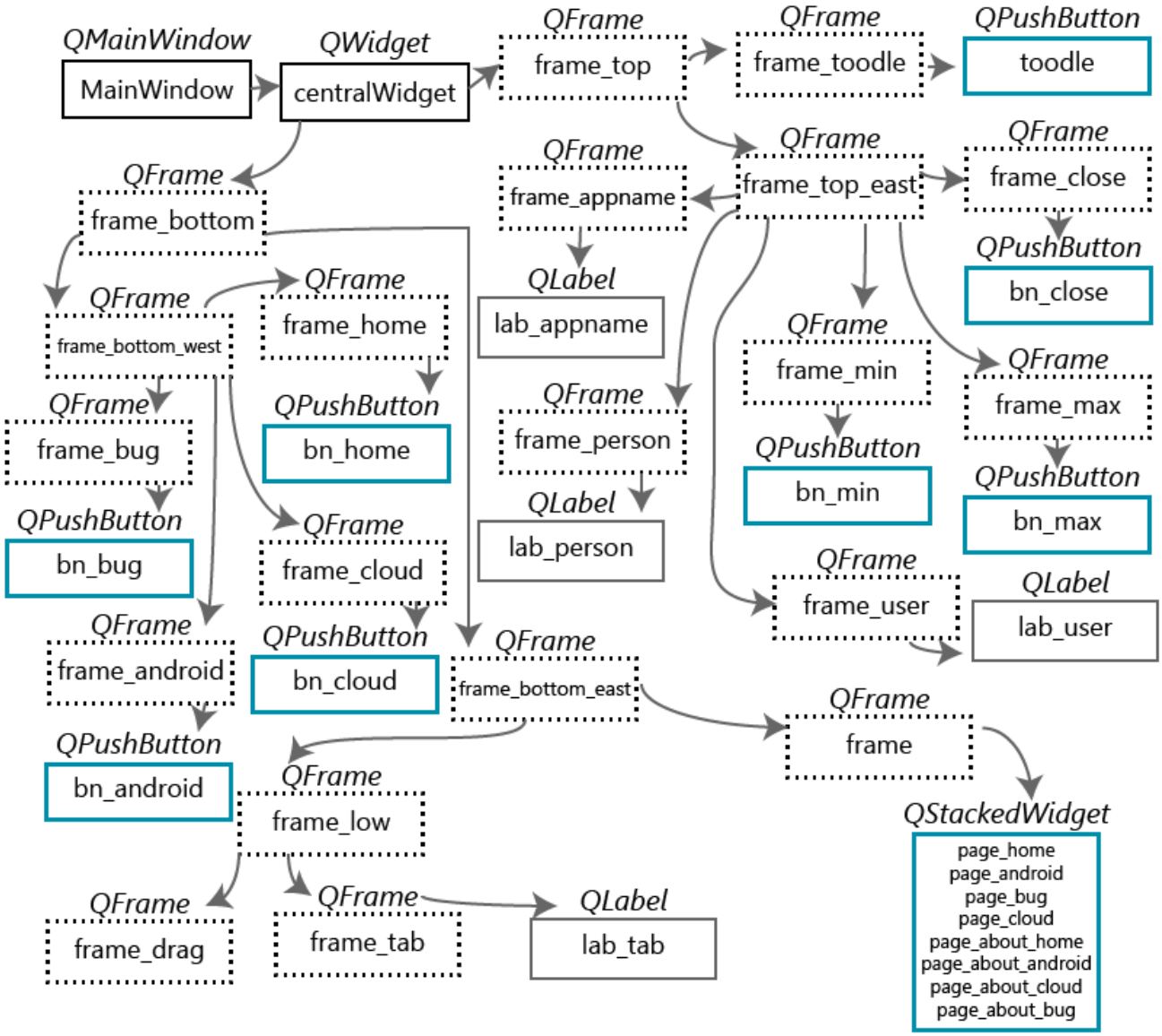
Here each object is represented as a rectangle box, with its class on top of the rectangle and object name inside the rectangle. Frames are represented in the dotted rectangle, widgets in blue colored rectangle, and others in a solid rectangle. Just below it you can see the GUI with name of the object pointing with a arrow into them.

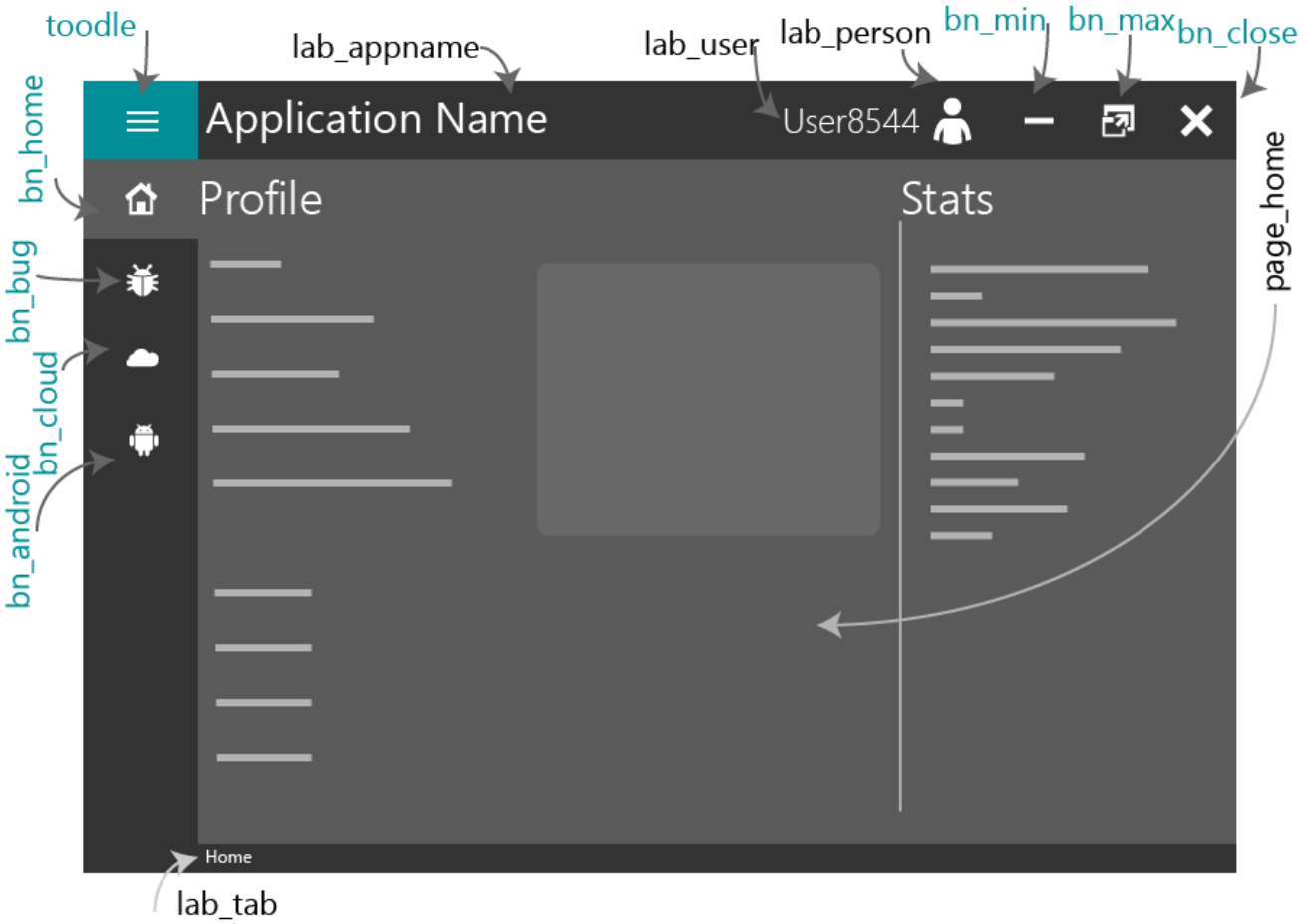
Let's start with the Color Scheme and Fonts:



Use the color code to modify the corners of your GUI. Now lets move to the actual layout of the GUI

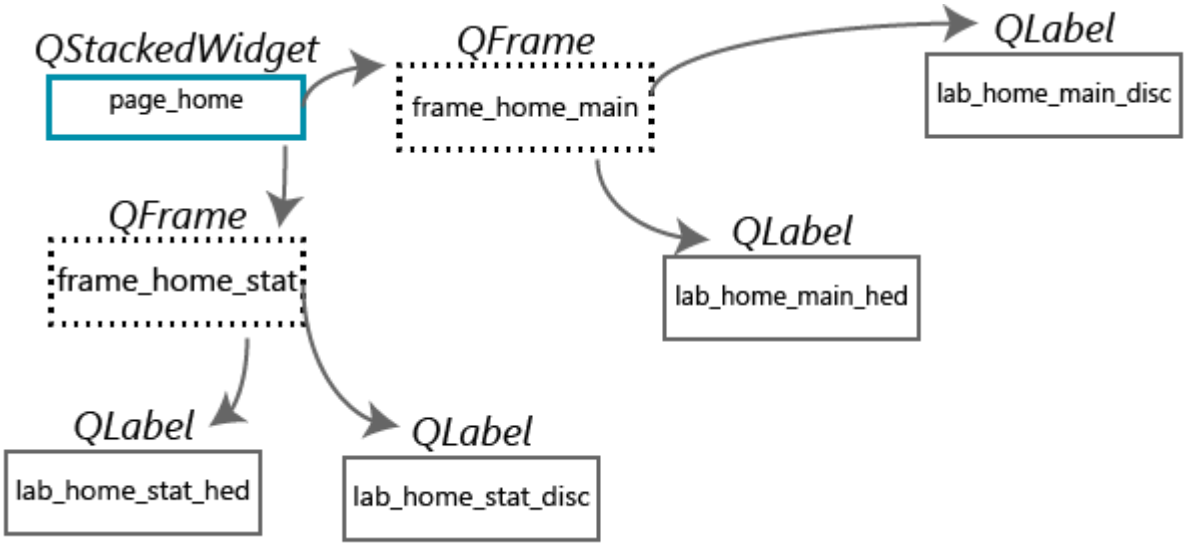
MainWindow Layout:

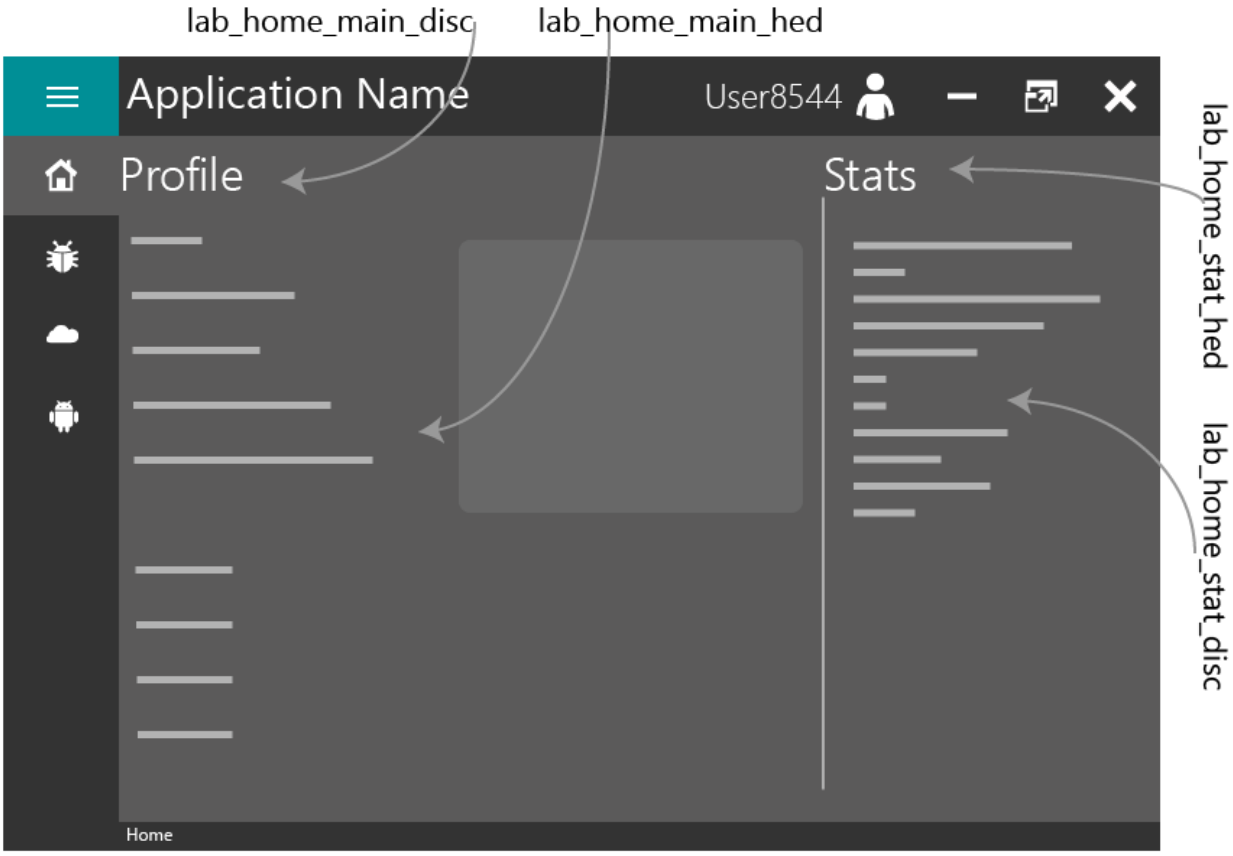

QMainWindow
MainWindow
QWidget
centralWidget
QFrame
frame_top
QFrame
frame_toodle
QPushButton
toodle
QFrame
frame_bottom
QFrame
frame_appname
QFrame
frame_top_east
QFrame
frame_close
QPushButton
bn_close
QFrame
frame_bottom_west
QFrame
frame_home
QLabel
lab_appname
QFrame
frame_min
QFrame
frame_max
QFrame
frame_bug
QPushButton
bn_home
QFrame
frame_person
QPushButton
bn_min
QPushButton
bn_max
QPushButton
bn_bug
QFrame
frame_cloud
QLabel
lab_person
QFrame
frame_user
QLabel
lab_user
QFrame
frame_android
QPushButton
bn_cloud
QFrame
frame_bottom_east
QFrame
frame
QPushButton
bn_android
QFrame
frame_low
QStackedWidget
page_home
page_android
page_bug
page_cloud
page_about_home
page_about_android
page_about_cloud
page_about_bug
QFrame
frame_drag
QFrame
frame_tab
QLabel
lab_tab



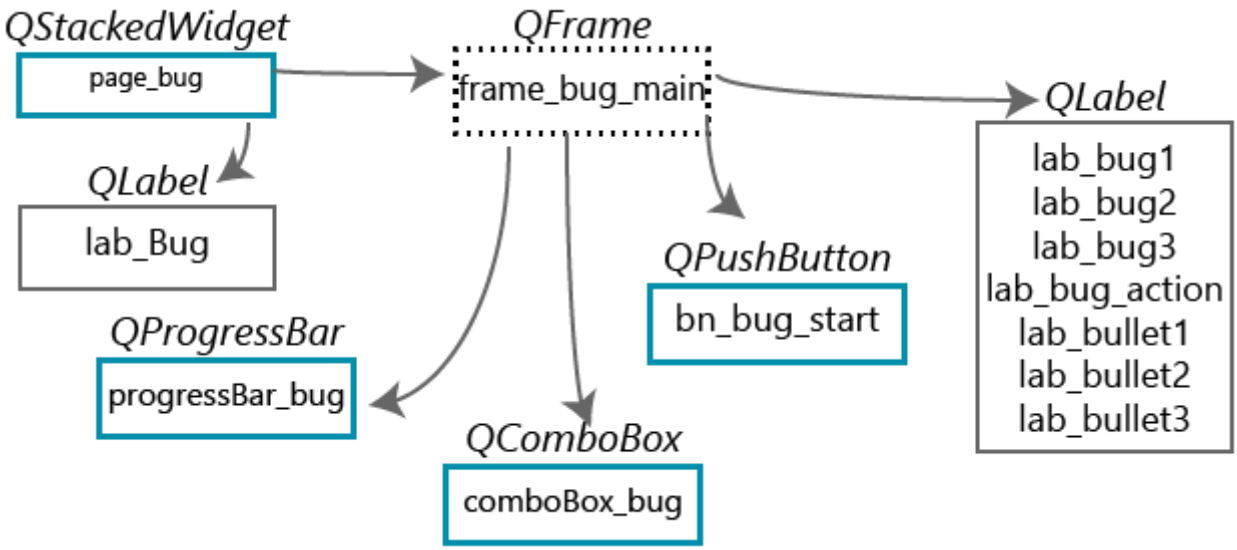
Since the Layout is so big as it contains many stacked widget pages inside it, I have used the space below to fill the rest of the individual Pages in the stack widget with each layout illustration. So refer each of this layout along with the GUI image below to refer to the corresponding pages

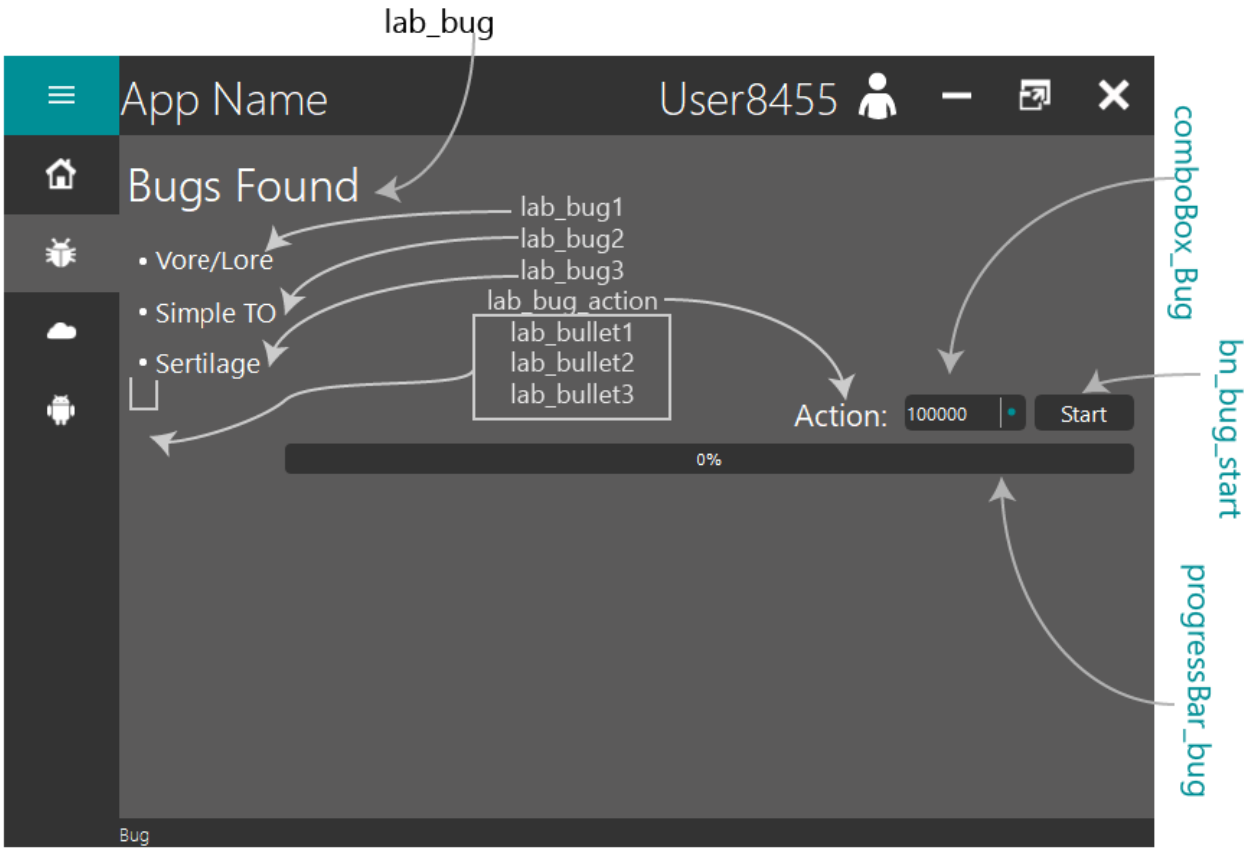
Page Home Layout



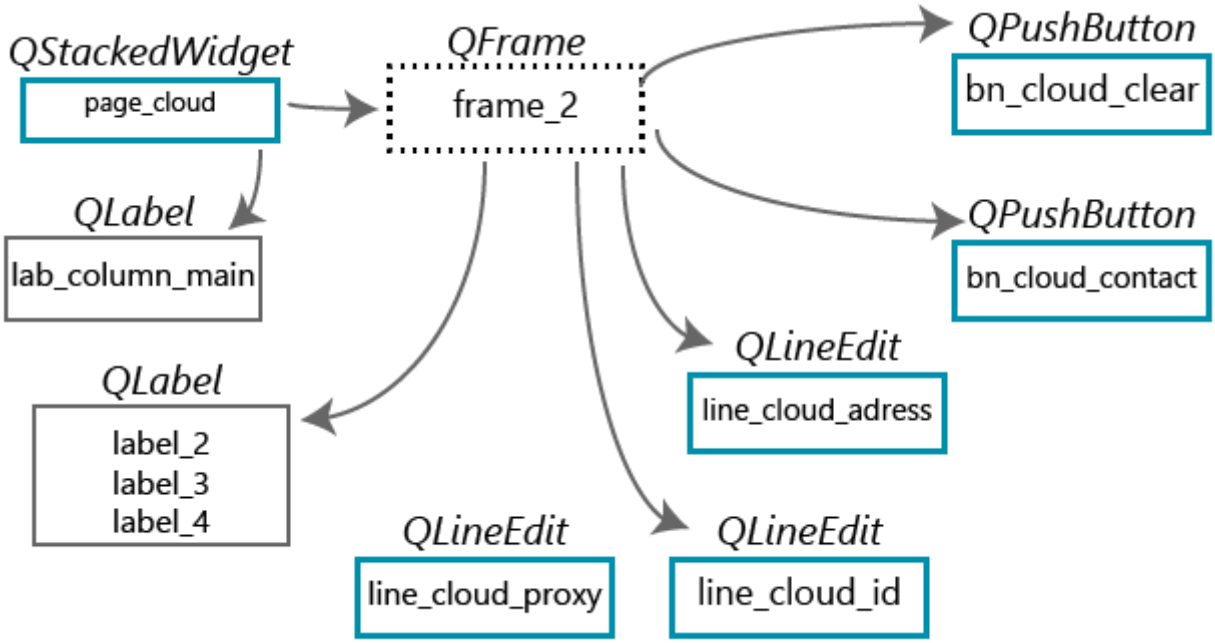


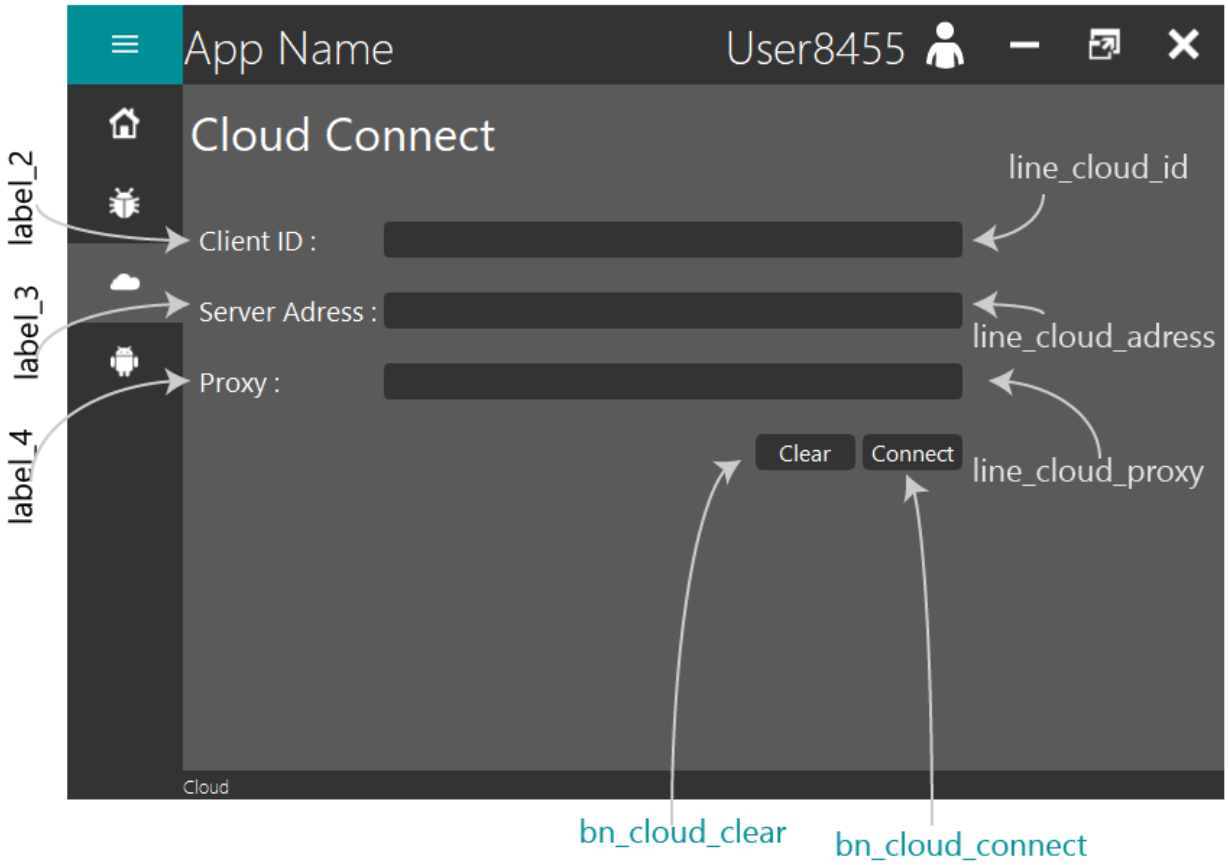
Page Bug Layout



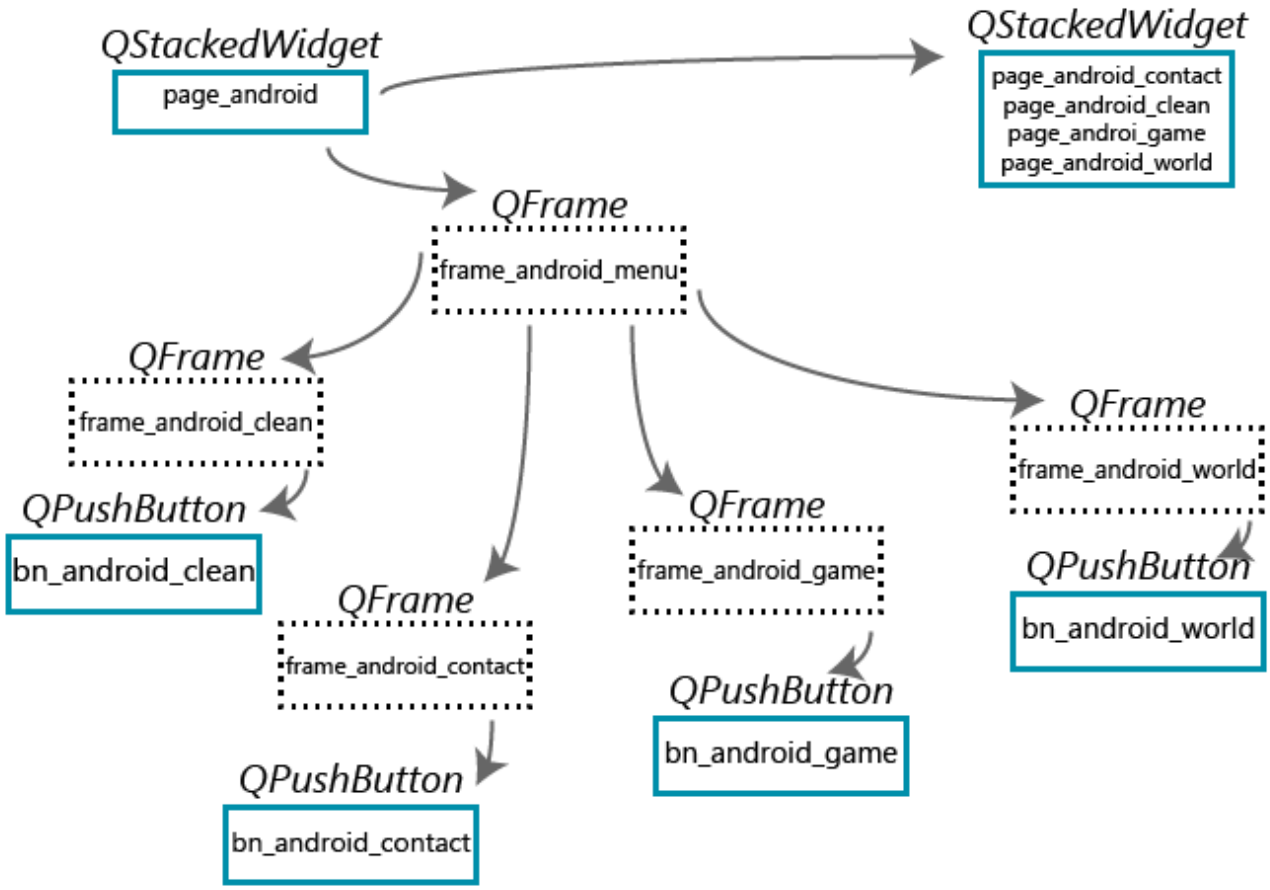


Page Cloud Layout





Page Android Layout



The last two pages are not given as their functionality in the `main.py` is not used, so if you want to implement this widget then copy the stylesheet and code the tools from the each widget codes given below.

Modify the GUI

Adding a New Menu Button

- Adding a new menu button also requires a new Page to be added in the stackWidget, which connect the menu button pressed event to this page. So follow the following procedure given below:
- Open the `ui_main.ui` file in the QT Designer and locate the frame : `frame_bottom_west`, where you can see the rest of the menu buttons there, add a new frame inside this frame, and give it a layout: Horizontal
- Copy one of the existing menu button to this frame, and modify the icon, object name in the 'Property Editor' tab.
- Then head over to the `main.py` file and at line 205 you can see something like this:

```
        #----> MENU BUTTON PRESSED EVENTS
        #NOW SINCE OUR DEMO APPLICATION HAS ONLY 4 MENU BUTTONS: Home, Bug, Android, Cloud,
WHEN USER PRESSES IT THE FOLLOWING CODE              ---------(C8)
        #REDIRECTS IT TO THE ui_function.py FILE buttonPressed() FUNCTION TO MAKE THE
NECESSERY RESPONSES TO THE BUTTON PRESSED.
        #IT TAKES SELF AND THE BUTTON NAME AS THE RGUMENT, THIS IS ONLY TO RECOGNISE WHICH
BUTTON IS PRESSED BY THE buttonPressed() FUNCTION.
        self.ui.bn_home.clicked.connect(lambda: UIFunction.buttonPressed(self, 'bn_home'))
        self.ui.bn_bug.clicked.connect(lambda: UIFunction.buttonPressed(self, 'bn_bug'))
        self.ui.bn_android.clicked.connect(lambda: UIFunction.buttonPressed(self,
'bn_android'))
        self.ui.bn_cloud.clicked.connect(lambda: UIFunction.buttonPressed(self,
'bn_cloud'))
        ###############################################################
```

- Bottom to this code add the new button function code:

```
self.ui.button_object_name_here.connect(lambda: UIFunction.buttonPressed(self,
'bn_object_name_here'))
```

- This code directs the button clicked event to the file `ui_function.py` function: *buttonPressed* in the class *UIFunction* which carries out all the code for changing the current stackedwidget page to the new created page.
- To create a new page in the stackedWidget: in the Qt Designer find the widget with object name : `stackedWidget` and right click upon it and click the 'Insert Page' option add two pages as one is for holding the widget and other one is for the 'About' i.e. the helpme about the new added page. Rename the pages in the 'Property Editor' respectively. Add all the other widget you wanted to the main button page.
- To open this page when the new button is pressed, open the `ui_function.py` file and insert the following line of code in the function `UIFunction.buttonPressed()` :

```
elif buttonName=='bn_object_name_here':
            if self.ui.frame_bottom_west.width()==80 and index!=6: #index of the new page
you added
                self.ui.stackedWidget.setCurrentWidget(self.ui.page1_name_here)
                self.ui.lab_tab.setText("page_heading here")
                self.ui.frame_cloud.setStyleSheet("background:rgb(91,90,90)") # SETS THE
BACKGROUND OF THE CLICKED BUTTON TO LITER COLOR THAN THE REST

            elif self.ui.frame_bottom_west.width()==160 and index!=2:#index of the 2nd new
page you added   # ABOUT PAGE STACKED WIDGET
                self.ui.stackedWidget.setCurrentWidget(self.ui.page2_about_name_here)
                self.ui.lab_tab.setText("About > page_heading_here")
                self.ui.frame_cloud.setStyleSheet("background:rgb(91,90,90)") # SETS THE
BACKGROUND OF THE CLICKED BUTTON TO LITER COLOR THAN THE REST
```

Adding New Widgets:

Head over to the Qt Designer and copy the widget you want to use, paste it in desired location, change the shape, alignment, stylesheet, object name and others in the 'Property Editor'. Head over to the `main.py` file and start editing the changes. Widget specific action to do is given in the Section: *PySide2 Widget Codes and StyleSheet*

PySide2 Widget Codes and StyleSheet

PushButton

> Push Button is used in this project in 3 main areas:

- In Close, Min, Max Buttons
- In Menu Buttons
- In Normal function buttons like: Save, Delete, Restore, Remove, Ok e.t.c

> Each of this button is not initialized in the `main.py` file as we have created the button, assigned an icon, assigned the text, set Max and Min size, set the stylesheet in the Qt Designer. This makes the process of widget initiation, style setting way easy. But process when the button is clicked is defined in our `main.py` file. This Applies to all the widget used in this project

> The following codes is used to address a pushButton

```
#self.ui is the object to class MainWindow()
self.ui.pushButton = QPushButton(self.ui.frame_name) #initialise the button inside
frame_name
self.ui.pushButton.setObjectName("btn_name")
self.ui.btn_name.setMaximumSize(QSize(55, 55)) #in px
self.ui.btn_name.setMinimumSize(QSize(20, 20)) # in px
self.ui.pushButton.setEnabled(True) #Enables the button
self.ui.pushButton.setEnable(False) #Disables the button
self.ui.pushButton.setText("Start")
self.ui.pushButton.setStyleSheet("{ stylesheet }") #This is where you set the color
parameter in idel, clicked, hover e.t.c see below
self.ui.pushButton.setIcon("location/icon.png")
self.ui.pushButton.setIconSize(QSize(22,22)) #in px
self.ui.pushButton.setFlat(True)
self.ui.pushButton.setToolTip("Name")
self.ui.pushButton.setFont(ft_obj) #ft_obj = QFont(), ft_obj.setFamily("Segoe
UI"),ft_obj.setPointSize(12)
```

> Stylesheet for each type of pushButton is given below.

```
/* STYLESHEET FOR PUSH BUTTON: CLOSE, MINIMIZE, MAXIMIZE*/
QPushButton {
    border: none;
    background-color: rgba(0,0,0,0);
}
QPushButton:hover {
    background-color: rgb(0,143,150);
}
QPushButton:pressed {
    background-color: rgba(0,0,0,0);
}
/* SEE THE EFFECT BELOW*/
```


```
/* STYLESHEET FOR PUSH BUTTON: MENU BUTTON */
QPushButton {
    border: none;
    background-color: rgba(0,0,0,0);
}
QPushButton:hover {
    background-color: rgb(91,90,90);
}
QPushButton:pressed {
    background-color: rgba(0,0,0,0);
}
/* SEE THE EFFECT BELOW*/
```


```
/* STYLESHEET FOR PUSH BUTTON: START, SAVE, CLOSE, RESTORE, CONNECT, EDIT */
QPushButton {
    border: 2px solid rgb(51,51,51);
    border-radius: 5px;
    color:rgb(255,255,255);
    background-color: rgb(51,51,51);
}
QPushButton:hover {
    border: 2px solid rgb(0,143,150);
    background-color: rgb(0,143,150);
}
QPushButton:pressed {
    border: 2px solid rgb(0,143,150);
    background-color: rgb(51,51,51);
}
QPushButton:disabled {
    border-radius: 5px;
    border: 2px solid rgb(112,112,112);
    background-color: rgb(112,112,112);
}
/* SEE THE EFFECT BELOW*/
```


```
/* STYLESHEET FOR PUSH BUTTON: DELETE, REMOVE */
QPushButton {
    border: 2px solid rgb(51,51,51);
    border-radius: 5px;
    color:rgb(255,255,255);
```

```
    background-color: rgb(51,51,51);
}
QPushButton:hover {
    border: 2px solid rgb(112,0,0);
    background-color: rgb(112,0,0);
}
QPushButton:pressed {
    border: 2px solid rgb(112,0,0);
    background-color: rgb(51,51,51);
}
QPushButton:disabled {
    border-radius: 5px;
    border: 2px solid rgb(112,112,112);
    background-color: rgb(112,112,112);
}
/* SEE THE EFFECT BELOW*/
```

> For more info on push Button refer to the official page: QPushButton

Progress Bar

> Same as a push Button we don't initiate the progress bar in our `main.py` file, but we took the shortcut in the Qt Designer.

> The following codes is used to address a progress Bar

```
self.ui.progressBar_bug = QProgressBar(self.ui.frame_name) #init progress bar object
self.ui.progressBar_bug.setObjectName(u"pB_Name") #changing the obj name
self.ui.progressBar_bug.setEnabled(True)
self.ui.progressBar_bug.setEnabled(False)
self.ui.progress_bar.setRange(min, max) #set the max and min range
self.ui.progress_bar.setMaximum(value)
self.ui.progress_bar.setMinimum(value)
self.ui.progressBar_bug.setStyleSheet("{ }")
self.ui.progressBar_bug.setValue(num) #progress value to var i.e. number from max to
minimum this runs the progress bar. Put this in your programme where you want the progress
bar to process.
self.ui.progressBar_bug.setAlignment(Qt.AlignCenter)
self.ui.progressBar_bug.setTextVisible(True) #percentage of progress
self.ui.progressBar_bug.setOrientation(Qt.Horizontal) #orientation
self.ui.progressBar_bug.setInvertedAppearance(False) #if True invertd the progress abr
```

> Stylesheet for Progress Bar:

```
/*Progress Bar Style Sheet*/
QProgressBar
{
    color:rgb(255,255,255);
    background-color :rgb(51,51,51);
    border : 2px;
    border-radius:4px;
}
```

```
QProgressBar::chunk{
    border : 2px;
    border-radius:4px;
    background-color:rgb(0,143,170);
}
/* SEE THE EFFECT BELOW*/
```

> For more info on push Button refer to the official page: QProgressBar

Combo Box

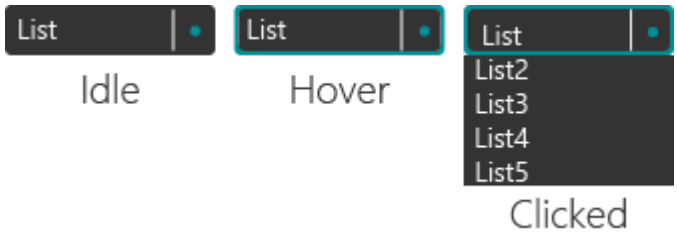
> Combo Box Code:

```
self.ui.comboBox_bug = QComboBox(self.frame_bug_main) #creating combobox object
self.ui.comboBox_bug.addItem("10000") #adds item to the list
self.ui.comboBox_bug.setObjectName(u"comboBox_Name")
self.ui.comboBox_bug.setMaximumSize(QSize(16777215, 25)) #16777215 is the max size in Qt
Creator
self.ui.comboBox_bug.setFont(font2)
self.ui.comboBox_bug.setStyleSheet()
self.ui.comboxBox_bug.setItemText(index_value, item) #Indexing the item added in the
combobox, a way to refer the item.
self.ui.comboBox_bug.currentIndex() #gives the index of the current displaying item.
```

> Stylesheet for Combo Box:

```
QComboBox {
    border: 2px solid rgb(51,51,51);
    border-radius: 5px;
    color:rgb(255,255,255);
    background-color: rgb(51,51,51);
}
QComboBox:hover {
    border: 2px solid rgb(0,143,170);
    border-radius: 5px;
    color:rgb(255,255,255);
    background-color: rgb(0,143,170);
}
QComboBox:!editable, QComboBox::drop-down:editable {
    background: rgb(51,51,51);
}
QComboBox:!editable:on, QComboBox::drop-down:editable:on {
    background:rgb(51,51,51);
}
QComboBox:on { /* shift the text when the popup opens */
    padding-top: 3px;
    padding-left: 4px;
}
QComboBox::drop-down {
    subcontrol-origin: padding;
    subcontrol-position: top right;
    width: 15px;
```

```
    border-left-width: 1px;
    border-left-color: darkgray;
    border-left-style: solid; /* just a single line */
    border-top-right-radius: 5px; /* same radius as the QComboBox */
    border-bottom-right-radius: 5px;
}
QComboBox::down-arrow {
    image: url(icons/1x/arrow.png);
}
QComboBox::down-arrow:on { /* shift the arrow when popup is open */
    top: 1px;
    left: 1px;
}
QComboBox::drop-down {
    background:rgb(51,51,51);
}
/* SEE THE EFFECT BELOW*/
```



> For More details in Combo Box Refer: QComboBox

RadioBox

> Radio Box code:

```
self.ui.radioButton = QRadioButton(self.ui.frame_name) #creating radiobutton object
self.ui.radioButton.setObjectName(u"radioButton_name")
self.ui.radioButton.setFont(font) #where it is a QFont object. Refer to QFont area
self.ui.radioButton.setAutoExclusive(True) #sets the radiobutton to Multiple choice pick
one mode.
self.ui.radioButton.setStyleSheet("{ --- }")
self.ui.radioButton.toggled.connect("any function") #check if the radio button when it is
toggle and executes the function.
```

> Stylesheet for the Radio Button:

```
/*STYLESHEET FOR PADIOBUTTON*/
QRadioButton {
    background:rgb(91,90,90);
    color:white;
}
QRadioButton::indicator {
    width:10px;
    height:10px;
    border-radius: 7px;
}
QRadioButton::indicator:checked {
    background-color:rgb(0,143,170);
    border: 2px solid rgb(51,51,51);
```

```
}
QRadioButton::indicator:unchecked {
    background-color:rgb(91,90,90);
    border:2px solid rgb(51,51,51);
}
/* SEE THE EFFECT BELOW*/
```

> For more details on the RadioButton Refer to the Official Page: QRadioButton

Check Box

> Check Box code:

```
self.ui.checkBox = QCheckBox(self.ui.frame_name)
self.ui.checkBox.setObjectName(u"checkBox")
self.ui.checkBox.setFont(font8)
self.ui.checkBox.setStyleSheet()
self.ui.checkBox.setTristate(False) # Three times clickable
self.ui.checkBox.stateChanged.connect("function to execute")
```

> Stylesheet for CheckBox:

```
/* STYLE SHEET FOR THE CHECKBOX*/
QCheckBox {
    color:rgb(255,255,255);
}
QCheckBox::indicator {
    width: 10px;
    height: 10px;
}
QCheckBox::indicator:unchecked {
    border:2px solid rgb(51,51,51);
    background:rgb(91,90,90);
}
QCheckBox::indicator:unchecked:pressed {
    border:2px solid rgb(51,51,51);
    background:rgb(0,143,170);
}
QCheckBox::indicator:checked {
    background-color:rgb(0,143,170);
    border: 2px solid rgb(51,51,51);
}
QCheckBox::indicator:checked:pressed {
    border:2px solid rgb(51,51,51);
    background:rgb(91,90,90);
}
/* SEE THE EFFECT BELOW*/
```

> For more details refer to Official Page: QCheckBox

Slider

> Slider Code:

```
self.ui.horizontalSlider_2 = QSlider(self.ui.frame_name)
self.ui.horizontalSlider_2.setObjectName(u"horizontalSlider_name")
self.ui.horizontalSlider_2.setStyleSheet()
self.ui.verticalSlider.setTracking(True)
self.ui.verticalSlider.setOrientation(Qt.Vertical)
self.ui.verticalSlider.setInvertedAppearance(False)
self.ui.verticalSlider.setInvertedControls(False)
self.ui.verticalSlider.setValue(5) #manually setting the value of the the silder.
self.ui.verticalSlicer.value() #GIVES THE LIVE VALUE ONLY ONCE IT SI CALLED
self.ui.verticalSlider.valueChanged.connect(function) #this call the function whenever the
slider si moved
#PLZ Refer to the official page for detailed reference.
```

> Stylesheet for the Slider:

```
/*SLIDER: VERTICAL*/
QSlider::groove:vertical {
    background: red;
    width:5px
}
QSlider::handle:vertical {
    height: 10px;
    background:rgb(0,143,170);
    margin:0 -8px
}
QSlider::add-page:vertical {
    background:rgb(51,51,51);
}
QSlider::sub-page:vertical {
    background:rgb(51,51,51);
}
```

```
/*SLIDER: HORIZONDAL*/
QSlider::groove:horizontal {
    height:5px;
    background: rgb(51,51,51);
}
QSlider::handle:horizontal {
    background:rgb(0,143,170);
    width: 10px;
    margin:-8px 0
}
QSlider::add-page:horizondal {
    background:rgb(51,51,51);
}
QSlider::sub-page:horizondal {
    background:rgb(51,51,51);
}
/* SEE THE EFFECT BELOW*/
```

> For further reference go to: QSlider

Scroll Bar

> Scroll Bar Code:

```
self.ui.vsb_about_home = QScrollBar(self.ui.frame_name)
self.ui.vsb_about_home.setObjectName(u"vsb_about_home")
self.ui.vsb_about_home.setStyleSheet()
self.ui.vsb_about_home.setOrientation(Qt.Vertical)
#setting the scroll bar to the textfiled, or any scrollable area: use the code:
self.ui.textEdit.setVerticalScrollBar(self.ui.vsb_about_home) #here we set the textEdit
scrollable area with the vertical scroll bar: vsb_about_home
```

> Stylesheet:

```
/*STYLESHEET FOR VERTICAL SCROLL BAR*/
QScrollBar:vertical {
    background:rgb(51,51,51);
    width:20px;
    margin: 0px 0px 0px 0px;
}
QScrollBar::handle:vertical {
    background:rgb(0,143,170);
}
QScrollBar::add-page:vertical {
    background:rgb(51,51,51);
}
QScrollBar::sub-page:vertical {
    background:rgb(51,51,51);
}
/* SEE THE EFFECT BELOW: WITH A SCROLLABLE TEXT AREA*/
```

> For further reference: QScrollBar

Line Edit

> Line Edit Code:

```
self.ui.lineEdit = QLineEdit(self.ui.frame_2)
self.ui.lineEdit.setObjectName(u"line_name")
self.ui.lineEdit.setMinimumSize(QSize(400, 25))
self.ui.lineEdit.setMaximumSize(QSize(500, 25))
self.ui.lineEdit.setFont(font6)
self.ui.lineEdit.setStyleSheet()
self.ui.lineEdit.setText("test")
self.ui.lineEdit.setEnabled(True)
self.ui.lineEdit.text() #THIS GIVES THE STRING ENTERED IN THE TEXT FIELD BY THE USER, IN
str
#CHECK THE OFFICIL PAGE AS THERE IS A LOT OF FUNCTIONALITY IN THE LINEEDIT WHICH IS SO
USEFUL GIVEN BEAUTIFULLY THERE LIKE SETTING * WHEN PASSWORD IS ENTERED, DRAG AND DROP IN
THE LINE FIELD.E.T.C.
```

> StyleSheet of Line Edit:

```
QLineEdit {
    color:rgb(255,255,255);
    border:2px solid rgb(51,51,51);
    border-radius:4px;
    background:rgb(51,51,51);
}
QLineEdit:disabled {
    color:rgb(255,255,255);
    border:2px solid rgb(112,112,112);
    border-radius:4px;
    background:rgb(112,112,112);
}
/* SEE THE EFFECT BELOW: WITH A SCROLLABLE TEXT AREA*/
```

> For further reference: QLineEdit

Support

😃 Support project by forking the repo or sharing this to other developers.

😃 Support like this motivates me to do more creative, work for Open Source.

👉Check out my other Projects in the My GitHub Profile

- Hiding Files inside an image: Project Image Steganography
- Backup your PC and Android wirelessly using the Ultra Backup Software.

*** ----------THANK YOU FOR VIEWING THIS PROJECT----------***