Beta.py

```python
import time
import threading
import RPi.GPIO as GPIO
import Adafruit_ADS1x15
from PyQt5.QtWidgets import *
from PyQt5 import QtGui
from PyQt5.QtCore import QRect, Qt, QTimer, QSize
from PyQt5 import QtCore, QtWidgets
from PyQt5.QtGui import QPixmap,QIcon
import sys
import Adafruit_DHT
from matplotlib.backends.backend_qt5agg import FigureCanvas
from matplotlib.figure import Figure
from matplotlib.backends.backend_qt5agg import (NavigationToolbar2QT as NavigationToolbar)
import datetime
import serial
import csv



adc = Adafruit_ADS1x15.ADS1115()

GAIN = 4
count = 50
maxVal = 0
amp = 0.001
prevAmp = 0.001

global sourceState
global maxTime
global maxCurrent

sourceState = 0
maxTime = 30
maxcurrent = 15

global currentArray
global currentTime
global timeArray
currentArray = []
timeArray = []

global stop_threads

global myFont
myFont = "Linux Biolinum O"
global lineList
global currentFontItem
currentFontItem = 11

global tempUnit
tempUnit = "C"

global timeFormat
timeFormat = "%I:%M:%S %p"
global currentTimeItem
currentTimeItem = 0
```

```python

global dateFormat
dateFormat = "%a %d-%m-%Y"
global currentDateItem
currentDateItem = 0

global tripTime
tripTime = ""

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(21, GPIO.OUT)
GPIO.output(21, GPIO.LOW)
GPIO.setup(26, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

class Window(QMainWindow):


    def __init__(self):
        super().__init__()

        global myFont
        global timeFormat

        title = "Relay Test Unit"
        left = 2
        top = 30
        width = 796
        height = 450

        self.setWindowTitle(title)
        self.setGeometry(left,  top, width, height)
        self.setFixedSize(width, height)

        wid = QtWidgets.QWidget(self)
        self.setCentralWidget(wid)
        self.setStyleSheet("background-color: white;")

        mainVBox = QVBoxLayout()
        topBox = QHBoxLayout()
        button1Box = QHBoxLayout()
        button2Box = QHBoxLayout()
        bottomBox = QHBoxLayout()

        self.labelTemp = QLabel(self)
        self.labelTemp.setAlignment(Qt.AlignLeft)
        self.labelTemp.setAlignment(Qt.AlignTop)
        self.labelTemp.setFixedHeight(40)
        self.labelTemp.setText("Temp: 36 °C")
        self.labelTemp.setFont(QtGui.QFont(myFont))
        self.labelTime = QLabel(self)
        self.labelTime.setAlignment(Qt.AlignRight)
        self.labelTime.setFixedHeight(40)
        self.labelTime.setText("Time")
        self.labelTime.setFont(QtGui.QFont(myFont))

        self.btn1 = QToolButton()
        self.btn1.setToolButtonStyle(Qt.ToolButtonTextUnderIcon)
        self.btn1.setFixedSize(110,100)
        self.btn1.setText("MCB Test")
        self.btn1.setFont(QtGui.QFont(myFont, 9))
        self.btn2 = QToolButton()
        self.btn2.setToolButtonStyle(Qt.ToolButtonTextUnderIcon)
        self.btn2.setFixedSize(110,100)
        self.btn2.setText("OCR Test")
        self.btn2.setFont(QtGui.QFont(myFont, 9))
        self.btn3 = QToolButton()
        self.btn3.setToolButtonStyle(Qt.ToolButtonTextUnderIcon)
        self.btn3.setFixedSize(110,100)
        self.btn3.setText("Logged Data")
        self.btn3.setFont(QtGui.QFont(myFont, 9))
        self.btn4 = QToolButton()
        self.btn4.setToolButtonStyle(Qt.ToolButtonTextUnderIcon)
        self.btn4.setFixedSize(110,100)
        self.btn4.setText("Settings")
        self.btn4.setFont(QtGui.QFont(myFont, 9))
        self.btn5 = QToolButton()
        self.btn5.setToolButtonStyle(Qt.ToolButtonTextUnderIcon)
        self.btn5.setFixedSize(110,100)
        self.btn5.setText("How To Use")
        self.btn5.setFont(QtGui.QFont(myFont, 9))
        self.btn6 = QToolButton()
        self.btn6.setToolButtonStyle(Qt.ToolButtonTextUnderIcon)
        self.btn6.setFixedSize(110,100)
        self.btn6.setText("About")
        self.btn6.setFont(QtGui.QFont(myFont, 9))
        self.btn7 = QPushButton()
        self.btn7.setFixedSize(60,55)
        pixmapMCB = QPixmap("MCB.jpg")
        pixmapcurrentRelay = QPixmap("Current Relay.jpg")
        pixmapVoltRelay = QPixmap("Voltage Relay.jpg")

        pixmapEmpty = QPixmap("Empty.jpg")

        pixmapDataLog = QPixmap("DataLog.jpg")
        pixmapSettings = QPixmap("Settings.jpg")
        pixmapUManual = QPixmap("User Manual.jpg")
        pixmapAbout = QPixmap("About.jpg")
        pixmapClose = QPixmap("Close.png")

        self.btn1.setIcon(QIcon(pixmapMCB))
        self.btn1.setIconSize(QSize(106,80))
        self.btn2.setIcon(QIcon(pixmapcurrentRelay))
        self.btn2.setIconSize(QSize(106,80))
        self.btn3.setIcon(QIcon(pixmapDataLog))
        self.btn3.setIconSize(QSize(106,80))
        self.btn4.setIcon(QIcon(pixmapSettings))
        self.btn4.setIconSize(QSize(106,80))
        self.btn5.setIcon(QIcon(pixmapUManual))
        self.btn5.setIconSize(QSize(106,80))
        self.btn6.setIcon(QIcon(pixmapAbout))
        self.btn6.setIconSize(QSize(106,80))
        self.btn7.setIcon(QIcon(pixmapClose))
        self.btn7.setIconSize(QSize(50,50))

        self.btn1.clicked.connect(self.MCBTest)
        self.btn2.clicked.connect(self.relayTest)
        self.btn4.clicked.connect(self.settings)
        self.btn5.clicked.connect(self.howToUse)
        self.btn6.clicked.connect(self.about)
        self.btn7.clicked.connect(self.closeFunction)

        topBox.addWidget(self.labelTemp)
        topBox.addWidget(self.labelTime)

        button1Box.addWidget(self.btn1)
        button1Box.addWidget(self.btn2)
        button1Box.addWidget(self.btn3)
        button2Box.addWidget(self.btn4)
        button2Box.addWidget(self.btn5)
        button2Box.addWidget(self.btn6)
        bottomBox.addWidget(self.btn7)

        mainVBox.addLayout(topBox)
        mainVBox.addLayout(button1Box)
        mainVBox.addLayout(button2Box)
        mainVBox.addLayout(bottomBox)

        clockTimer = QTimer(self)
        clockTimer.timeout.connect(self.showTime)
        clockTimer.start(1000)
        self.showTime()








        wid.setLayout(mainVBox)
        self.show()


    def showTime(self):
        now = datetime.datetime.now()
        now = now.strftime(dateFormat+"\n"+timeFormat)
        self.labelTime.setText(now)

    def MCBTest(self):
        self.w = mcbWindow()
        self.w.show()

    def relayTest(self):
        self.w = relayWindow()
        self.w.show()

    def settings(self):
        self.w = settingsWindow()
        self.w.show()
        self.close()

    def howToUse(self):
        self.w = howtoWindow()
        self.w.show()

    def about(self):
        self.w = aboutWindow()
        self.w.show()

    def closeFunction(self):
        self.close()

```

```python
class mcbWindow(QMainWindow):
    global sourceState
    global maxTime
    global maxCurrent
    global myFont

    global timeFormat
    global dateFormat


    sourceState = 0
    maxTime = 30
    maxcurrent = 15
    def __init__(self):
        super().__init__()
        self.setWindowTitle("MCB Test")
        left = 2
        top = 30
        width = 796
        height = 450
        self.setGeometry(left,  top, width, height)
        self.setFixedSize(width, height)

        wid = QtWidgets.QWidget(self)
        self.setCentralWidget(wid)
        self.setStyleSheet("background-color: white;")

        mainVBox = QVBoxLayout()
        topBox = QHBoxLayout()
        bottomBox = QHBoxLayout()
        graphVBox = QVBoxLayout()
        settingsVBox = QVBoxLayout()
        buttonHBox = QHBoxLayout()

        #Designing TopBox
        self.labelTemp = QLabel(self)
        self.labelTemp.setAlignment(Qt.AlignLeft)
        self.labelTemp.setAlignment(Qt.AlignTop)
        self.labelTemp.setFixedHeight(40)
        self.labelTemp.setText("Temp: 36 °C")
        self.labelTemp.setFont(QtGui.QFont(myFont))
        self.labelTime = QLabel(self)
        self.labelTime.setAlignment(Qt.AlignRight)
        self.labelTime.setFixedHeight(40)
        self.labelTime.setText("Time")
        self.labelTime.setFont(QtGui.QFont(myFont))

        topBox.addWidget(self.labelTemp)
        topBox.addWidget(self.labelTime)

        clockTimer = QTimer(self)
        clockTimer.timeout.connect(self.showTime)
        clockTimer.start(1000)
        self.showTime()

        #Designing Graph anf Buttons
        self.plotMCB = MatplotlibWidget()
        self.plotMCB.setFixedSize(550,350)
        graphVBox.addWidget(self.plotMCB)

        self.backButton = QToolButton()
        self.backButton.setFixedSize(60,30)
        self.backButton.clicked.connect(self.closeFunction)
        self.onButton = QToolButton()
        self.onButton.setFixedSize(60,30)
        self.onButton.setText("On")
        self.onButton.setFont(QtGui.QFont(myFont))
        self.onButton.clicked.connect(self.onButtonFunction)
        self.onTimeButton = QToolButton()
        self.onTimeButton.setFixedSize(90,30)
        self.onTimeButton.setText("On+Time")
        self.onTimeButton.setFont(QtGui.QFont(myFont))
        self.onTimeButton.clicked.connect(self.onTimeButtonFunction)
        self.offButton = QToolButton()
        self.offButton.setFixedSize(60,30)
        self.offButton.setText("Off")
        self.offButton.setFont(QtGui.QFont(myFont))
        #print(myFont)
        self.offButton.clicked.connect(self.offButtonFunction)

        pixmapBack = QPixmap("Back.png")
        self.backButton.setIcon(QIcon(pixmapBack))
        self.backButton.setIconSize(QSize(60,35))

        buttonHBox.addWidget(self.backButton)
        buttonHBox.addWidget(self.onButton)
        buttonHBox.addWidget(self.onTimeButton)
        buttonHBox.addWidget(self.offButton)
        graphVBox.addLayout(buttonHBox)

        #Designing Settings
        currentLabel = QLabel(self)
        currentLabel.setText("Maximum Current")
        currentLabel.setFont(QtGui.QFont(myFont, 12))
        currentLabel.setAlignment(Qt.AlignCenter | Qt.AlignTop)
        self.currentSelector = QComboBox()
        self.currentSelector.setFont(QtGui.QFont(myFont, 12))
        self.currentSelector.addItem("2.5 Amps")
        self.currentSelector.addItem("5 Amps")
        self.currentSelector.addItem("7.5 Amps")
        self.currentSelector.addItem("10 Amps")
        self.currentSelector.addItem("12.5 Amps")
        self.currentSelector.addItem("15 Amps")
        self.currentSelector.addItem("17.5 Amps")
        self.currentSelector.addItem("20 Amps")
        self.currentSelector.addItem("22.5 Amps")
        self.currentSelector.addItem("25 Amps")
        self.currentSelector.addItem("27.5 Amps")
        self.currentSelector.addItem("30 Amps")
        self.currentSelector.addItem("32.5 Amps")
        self.currentSelector.addItem("35 Amps")
        self.currentSelector.addItem("37.5 Amps")
        self.currentSelector.addItem("40 Amps")
        self.currentSelector.addItem("42.5 Amps")
        self.currentSelector.addItem("45 Amps")
        self.currentSelector.setCurrentIndex(5)
        self.currentSelector.currentIndexChanged.connect(self.currentChange)

        timeLabel = QLabel(self)
        timeLabel.setText("Maximum Time")
        timeLabel.setFont(QtGui.QFont(myFont, 12))
        timeLabel.setAlignment(Qt.AlignCenter)
        self.timeSelector = QComboBox()
        self.timeSelector.setFont(QtGui.QFont(myFont, 12))
        self.timeSelector.addItem("5 Sec")
        self.timeSelector.addItem("10 Sec")
        self.timeSelector.addItem("15 Sec")
        self.timeSelector.addItem("20 Sec")
        self.timeSelector.addItem("25 Sec")
        self.timeSelector.addItem("30 Sec")
        self.timeSelector.addItem("35 Sec")
        self.timeSelector.addItem("40 Sec")
        self.timeSelector.addItem("45 Sec")
        self.timeSelector.addItem("50 Sec")
        self.timeSelector.addItem("55 Sec")
        self.timeSelector.addItem("60 Sec")
        self.timeSelector.addItem("65 Sec")
        self.timeSelector.addItem("70 Sec")
        self.timeSelector.addItem("75 Sec")
        self.timeSelector.addItem("80 Sec")
        self.timeSelector.addItem("85 Sec")
        self.timeSelector.addItem("90 Sec")
        self.timeSelector.setCurrentIndex(5)
        self.timeSelector.currentIndexChanged.connect(self.timeChange)

        self.timeVal = QLabel(self)
        self.currentVal = QLabel(self)
        self.timeVal.setText("Time: ------- Sec")
        self.currentVal.setText("Current: ----- Amps")
        self.timeVal.setFont(QtGui.QFont(myFont, 12))
        self.timeVal.setAlignment(Qt.AlignCenter)
        self.currentVal.setFont(QtGui.QFont(myFont, 12))
        self.currentVal.setAlignment(Qt.AlignCenter)

        updateGraphButton = QToolButton()
        updateGraphButton.setText("Update Graph")
        updateGraphButton.setFont(QtGui.QFont(myFont))
        updateGraphButton.clicked.connect(self.updateGraph)

        logButton = QToolButton()
        logButton.setText("LogData")
        logButton.setFont(QtGui.QFont(myFont))
        logButton.setFixedSize(113,30)
        logButton.clicked.connect(self.logData)

        resetButton = QToolButton()
        resetButton.setText("Reset")
        resetButton.setFont(QtGui.QFont(myFont))
        resetButton.setFixedSize(113,30)
        resetButton.clicked.connect(self.resetFunction)

        self.status = QLabel(self)
        self.status.setText("Status: MCB Test Interface Initiated")
        self.status.setStyleSheet("color: black")
        self.status.setAlignment(Qt.AlignLeft)
        self.status.setFont(QtGui.QFont(myFont, 12))

        settingsVBox.addWidget(currentLabel)
        settingsVBox.addWidget(self.currentSelector)
        settingsVBox.addWidget(timeLabel)
        settingsVBox.addWidget(self.timeSelector)
        settingsVBox.addWidget(self.currentVal)
        settingsVBox.addWidget(self.timeVal)
```

```
         settingsVBox.addWidget(updateGraphButton, alignment=Qt.AlignCenter)
         settingsVBox.addWidget(logButton, alignment=Qt.AlignCenter)
         settingsVBox.addWidget(resetButton, alignment=Qt.AlignCenter)
         settingsVBox.addWidget(self.status)



         bottomBox.addLayout(graphVBox)
         bottomBox.addLayout(settingsVBox)
         mainVBox.addLayout(topBox)
         mainVBox.addLayout(bottomBox)


         wid.setLayout(mainVBox)
         self.show()

     def showTime(self):
        now = datetime.datetime.now()
        now = now.strftime(dateFormat+"\n"+timeFormat)
        self.labelTime.setText(now)

     def onButtonFunction(self):
        global sourceState
        global amp
        GPIO.output(21, GPIO.HIGH)
        sourceState = 1
        self.timeVal.setText("Time: ------- Sec")
        self.status.setText("Status: Source Turned On Manually")
        self.status.setStyleSheet("color: black")
        amp = 0.001
        while(True):
            QtCore.QTimer.singleShot(10, self.currentSensing)

            time.sleep(0.05)
            if(sourceState == 0):
               self.status.setText("Status: Source Turned Off Manually")
               self.status.setStyleSheet("color: black")
               break
            if(amp > maxcurrent):
                self.status.setText("Status: Maximum Current Reached ")
                self.status.setStyleSheet("color: red")
                GPIO.output(21, GPIO.LOW)
                sourceState = 0
                self.currentVal.setText("Current: {:0.2f} Amps".format(amp))
                break
            self.currentVal.setText("Current: {:0.2f} Amps".format(amp))
            QtCore.QCoreApplication.processEvents()
        amp = 0.001

     def run(self):
         global currentTime
         global stop_threads
         start = time.time()
         while True:
             if stop_threads:
                 break
             #print('thread running')
             end = time.time()
             currentTime = end-start
             self.timeVal.setText("Time: {:0.2f} Sec".format(currentTime))
             time.sleep(0.09)
```

```python

    def onTimeButtonFunction(self):
        global sourceState
        global maxTime
        global maxcurrent
        global stop_threads
        global currentTime
        global amp
        global prevAmp
        global tripTime
        amp = 0.001
        prevAmp = 0.001
        GPIO.output(21, GPIO.HIGH)
        sourceState = 1
        self.status.setText("Status: Source Turned On Manually")
        self.status.setStyleSheet("color: black")

        stop_threads = False
        t1 = threading.Thread(target = self.run)
        t1.start()
        while(True):
            QtCore.QTimer.singleShot(0, self.currentSensing)
            if(amp > 0):
                if (prevAmp - amp <= 2):
                    prevAmp = amp
            if(amp <= 0):
                self.status.setText("Status: MCB Tripped Successfully")
                now = datetime.datetime.now()
                now = now.strftime(dateFormat+"   "+timeFormat)
                tripTime = now
                self.status.setStyleSheet("color: red")
                GPIO.output(21, GPIO.LOW)
                sourceState = 0
                amp = 0.001
                break
            if(sourceState == 0):
                self.status.setText("Status: Source Turned Off Manually")
                self.status.setStyleSheet("color: black")
                break
            if(amp > maxcurrent):
                self.status.setText("Status: Maximum Current Reached ")
                self.status.setStyleSheet("color: red")
                GPIO.output(21, GPIO.LOW)
                sourceState = 0
                self.currentVal.setText("Current: {:0.2f} Amps".format(prevAmp))
                break
            if(currentTime >= maxTime):
                self.status.setText("Status: Maximum Time Reached     ")
                self.status.setStyleSheet("color: red")
                GPIO.output(21, GPIO.LOW)
                sourceState = 0
                self.currentVal.setText("Current: {:0.2f} Amps".format(prevAmp))
                break

            QtCore.QCoreApplication.processEvents()
            self.currentVal.setText("Current: {:0.2f} Amps".format(prevAmp))
        amp = 0.001
        #prevAmp = 0.001
        stop_threads = True
        t1.join()
        print('thread killed')


     def currentSensing(self):
         global GAIN
         global count
         global maxVal
         global amp
         global prevAmp
         series = [0]*count
         value = 0
         for i in range (count):
             try:
                 value = adc.read_adc(0, gain=GAIN)
                 time.sleep(0.001)
                 series[i] = value
             except:
                 print("ADC Read Error")
             QtCore.QCoreApplication.processEvents()
         maxVal = max(series)
         #amp = 0.0014*maxVal + 0.4
         amp = -0.00000000000003*(maxVal**3) + 0.0000000023*(maxVal**2) + 0.0014*maxVal + 0.4
         amp = round(amp,1)

         if(amp < 0.3): amp = 0

         QtCore.QCoreApplication.processEvents()

     def offButtonFunction(self):
        global sourceState
        GPIO.output(21, GPIO.LOW)
        self.currentVal.setText("Current: ----- Amps")
        self.timeVal.setText("Time: ------- Sec")
        sourceState = 0

     def updateGraph(self):
         global currentArray
         global timeArray
         global currentTime
         global prevAmp
         currentArray.append(prevAmp)
         timeArray.append(currentTime)
         print(currentArray)
         print(timeArray)
         self.plotMCB.axes.cla()
         self.plotMCB.axes.set_ylim( bottom = 0, top = 50)
         self.plotMCB.axes.set_xlim( left = 0, right = 50)
         self.plotMCB.axes.set_title("TCC Curve")
         self.plotMCB.axes.set_ylabel("Time (Sec)")
         self.plotMCB.axes.set_xlabel("Current (Amps)")
         self.plotMCB.axes.plot(currentArray,timeArray)
         self.plotMCB.draw()



     def currentChange(self,i):
        global maxcurrent
        maxcurrent = (i+1)*2.5
        #print((i+1)*2.5)

     def timeChange(self,i):
        global maxTime
```

```
        maxTime = (i+1)*5
        #print((i+1)*5)

    def resetFunction(self):
        self.timeVal.setText("Time: ------- Sec")
        self.currentVal.setText("Current: ----- Amps")
        self.status.setText("Status: MCB Test Interface Initiated")
        self.status.setStyleSheet("color: black")
        self.plotMCB.axes.cla()
        self.plotMCB.axes.set_ylim( bottom = 0, top = 50)
        self.plotMCB.axes.set_xlim( left = 0, right = 50)
        self.plotMCB.axes.set_title("TCC Curve")
        self.plotMCB.axes.set_ylabel("Time (Sec)")
        self.plotMCB.axes.set_xlabel("Current (Amps)")
        self.plotMCB.draw()

    def logData(self):
        global prevAmp
        global currentTime
        global tripTime

        print(tripTime)




    def closeFunction(self):
        global sourceState
        global prevAmp
        global amp
        GPIO.output(21, GPIO.LOW)
        sourceState = 0
        global currentArray
        global timeArray
        currentArray = []
        timeArray = []
        prevAmp = 0.001
        amp = 0.001
        self.close()

class relayWindow(QMainWindow):
    global sourceState
    global maxTime
    global maxCurrent
    global myFont

    global dateFormat
    global timeFormat

    sourceState = 0
    maxTime = 30
    maxcurrent = 15
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Relay Test")
        left = 2
        top = 30
        width = 796
        height = 450
        self.setGeometry(left,  top, width, height)
        self.setFixedSize(width, height)
```

```python

        wid = QtWidgets.QWidget(self)
        self.setCentralWidget(wid)
        self.setStyleSheet("background-color: white;")

        mainVBox = QVBoxLayout()
        topBox = QHBoxLayout()
        bottomBox = QHBoxLayout()
        graphVBox = QVBoxLayout()
        settingsVBox = QVBoxLayout()
        buttonHBox = QHBoxLayout()

        #Designing TopBox
        self.labelTemp = QLabel(self)
        self.labelTemp.setAlignment(Qt.AlignLeft)
        self.labelTemp.setAlignment(Qt.AlignTop)
        self.labelTemp.setFixedHeight(40)
        self.labelTemp.setText("Temp: 36 °C")
        self.labelTemp.setFont(QtGui.QFont(myFont))
        self.labelTime = QLabel(self)
        self.labelTime.setAlignment(Qt.AlignRight)
        self.labelTime.setFixedHeight(40)
        self.labelTime.setText("Time")
        self.labelTime.setFont(QtGui.QFont(myFont))

        topBox.addWidget(self.labelTemp)
        topBox.addWidget(self.labelTime)

        clockTimer = QTimer(self)
        clockTimer.timeout.connect(self.showTime)
        clockTimer.start(1000)
        self.showTime()

        #Designing Graph anf Buttons
        self.plotMCB = MatplotlibWidget()
        self.plotMCB.setFixedSize(550,350)
        graphVBox.addWidget(self.plotMCB)

        self.backButton = QToolButton()
        self.backButton.setFixedSize(60,30)
        self.backButton.clicked.connect(self.closeFunction)
        self.onButton = QToolButton()
        self.onButton.setFixedSize(60,30)
        self.onButton.setText("On")
        self.onButton.setFont(QtGui.QFont(myFont))
        self.onButton.clicked.connect(self.onButtonFunction)
        self.onTimeButton = QToolButton()
        self.onTimeButton.setFixedSize(90,30)
        self.onTimeButton.setText("On+Time")
        self.onTimeButton.setFont(QtGui.QFont(myFont))
        self.onTimeButton.clicked.connect(self.onTimeButtonFunction)
        self.offButton = QToolButton()
        self.offButton.setFixedSize(60,30)
        self.offButton.setText("Off")
        self.offButton.setFont(QtGui.QFont(myFont))
        self.offButton.clicked.connect(self.offButtonFunction)

        pixmapBack = QPixmap("Back.png")
        self.backButton.setIcon(QIcon(pixmapBack))
        self.backButton.setIconSize(QSize(60,35))

        buttonHBox.addWidget(self.backButton)
        buttonHBox.addWidget(self.onButton)
        buttonHBox.addWidget(self.onTimeButton)
        buttonHBox.addWidget(self.offButton)
        graphVBox.addLayout(buttonHBox)

        #Designing Settings
        currentLabel = QLabel(self)
        currentLabel.setText("Maximum Current")
        currentLabel.setFont(QtGui.QFont(myFont, 12))
        currentLabel.setAlignment(Qt.AlignCenter | Qt.AlignTop)
        self.currentSelector = QComboBox()
        self.currentSelector.setFont(QtGui.QFont(myFont, 12))
        self.currentSelector.addItem("2.5 Amps")
        self.currentSelector.addItem("5 Amps")
        self.currentSelector.addItem("7.5 Amps")
        self.currentSelector.addItem("10 Amps")
        self.currentSelector.addItem("12.5 Amps")
        self.currentSelector.addItem("15 Amps")
        self.currentSelector.addItem("17.5 Amps")
        self.currentSelector.addItem("20 Amps")
        self.currentSelector.addItem("22.5 Amps")
        self.currentSelector.addItem("25 Amps")
        self.currentSelector.addItem("27.5 Amps")
        self.currentSelector.addItem("30 Amps")
        self.currentSelector.addItem("32.5 Amps")
        self.currentSelector.addItem("35 Amps")
        self.currentSelector.addItem("37.5 Amps")
        self.currentSelector.addItem("40 Amps")
        self.currentSelector.addItem("42.5 Amps")
        self.currentSelector.addItem("45 Amps")
        self.currentSelector.setCurrentIndex(5)
        self.currentSelector.currentIndexChanged.connect(self.currentChange)

        timeLabel = QLabel(self)
        timeLabel.setText("Maximum Time")
        timeLabel.setFont(QtGui.QFont(myFont, 12))
        timeLabel.setAlignment(Qt.AlignCenter)
        self.timeSelector = QComboBox()
        self.timeSelector.setFont(QtGui.QFont(myFont, 12))
        self.timeSelector.addItem("5 Sec")
        self.timeSelector.addItem("10 Sec")
        self.timeSelector.addItem("15 Sec")
        self.timeSelector.addItem("20 Sec")
        self.timeSelector.addItem("25 Sec")
        self.timeSelector.addItem("30 Sec")
        self.timeSelector.addItem("35 Sec")
        self.timeSelector.addItem("40 Sec")
        self.timeSelector.addItem("45 Sec")
        self.timeSelector.addItem("50 Sec")
        self.timeSelector.addItem("55 Sec")
        self.timeSelector.addItem("60 Sec")
        self.timeSelector.addItem("65 Sec")
        self.timeSelector.addItem("70 Sec")
        self.timeSelector.addItem("75 Sec")
        self.timeSelector.addItem("80 Sec")
        self.timeSelector.addItem("85 Sec")
        self.timeSelector.addItem("90 Sec")
        self.timeSelector.setCurrentIndex(5)
        self.timeSelector.currentIndexChanged.connect(self.timeChange)

        self._toggle = True
        self.nOpenCheck = QCheckBox("Normally Open",self)
        self.nOpenCheck.setFont(QtGui.QFont(myFont))
        self.nOpenCheck.setChecked(self._toggle)
        self.nCloseCheck = QCheckBox("Normally Close",self)
        self.nCloseCheck.setFont(QtGui.QFont(myFont))
        self.nCloseCheck.setChecked(not self._toggle)
        self.nCloseCheck.clicked.connect(self.toggle)
        self.nOpenCheck.clicked.connect(self.toggle)


        self.timeVal = QLabel(self)
        self.currentVal = QLabel(self)
        self.timeVal.setText("Time: ------- Sec")
        self.currentVal.setText("Current: ----- Amps")
        self.timeVal.setFont(QtGui.QFont(myFont, 12))
        self.timeVal.setAlignment(Qt.AlignCenter)
        self.currentVal.setFont(QtGui.QFont(myFont, 12))
        self.currentVal.setAlignment(Qt.AlignCenter)


        updateGraphButton = QToolButton()
        updateGraphButton.setText("Update Graph")
        updateGraphButton.setFont(QtGui.QFont(myFont))
        updateGraphButton.clicked.connect(self.updateGraph)

        logButton = QToolButton()
        logButton.setText("LogData")
        logButton.setFont(QtGui.QFont(myFont))
        logButton.setFixedSize(113,30)

        resetButton = QToolButton()
        resetButton.setText("Reset")
        resetButton.setFont(QtGui.QFont(myFont))
        resetButton.setFixedSize(113,30)
        resetButton.clicked.connect(self.resetFunction)

        self.status = QLabel(self)
        self.status.setText("Status: Relay Test Interface Initiated")
        self.status.setStyleSheet("color: black")
        self.status.setAlignment(Qt.AlignLeft)
        self.status.setFont(QtGui.QFont(myFont, 12))

        settingsVBox.addWidget(currentLabel)
        settingsVBox.addWidget(self.currentSelector)
        settingsVBox.addWidget(timeLabel)
        settingsVBox.addWidget(self.timeSelector)
        settingsVBox.addWidget(self.nOpenCheck, alignment=Qt.AlignCenter)
        settingsVBox.addWidget(self.nCloseCheck, alignment=Qt.AlignCenter)
        settingsVBox.addWidget(self.currentVal)
        settingsVBox.addWidget(self.timeVal)
        settingsVBox.addWidget(updateGraphButton, alignment=Qt.AlignCenter)
        settingsVBox.addWidget(logButton, alignment=Qt.AlignCenter)
        settingsVBox.addWidget(resetButton, alignment=Qt.AlignCenter)
        settingsVBox.addWidget(self.status)



        bottomBox.addLayout(graphVBox)
        bottomBox.addLayout(settingsVBox)
        mainVBox.addLayout(topBox)
        mainVBox.addLayout(bottomBox)


        wid.setLayout(mainVBox)
        self.show()

    def showTime(self):
       now = datetime.datetime.now()
       now = now.strftime(dateFormat+"\n"+timeFormat)
       self.labelTime.setText(now)

    def onButtonFunction(self):
       global sourceState
       global amp
       GPIO.output(21, GPIO.HIGH)
       sourceState = 1
       self.timeVal.setText("Time: ------- Sec")
       self.status.setText("Status: Source Turned On Manually")
       self.status.setStyleSheet("color: black")
       amp = 0.001
       while(True):
           QtCore.QTimer.singleShot(10, self.currentSensing)

           time.sleep(0.05)
           if(sourceState == 0):
              self.status.setText("Status: Source Turned Off Manually")
              self.status.setStyleSheet("color: black")
              break
           if(amp > maxcurrent):
               self.status.setText("Status: Maximum Current Reached ")
               self.status.setStyleSheet("color: red")
               GPIO.output(21, GPIO.LOW)
               sourceState = 0
               self.currentVal.setText("Current: {:0.2f} Amps".format(amp))
               break
           self.currentVal.setText("Current: {:0.2f} Amps".format(amp))
           QtCore.QCoreApplication.processEvents()
       amp = 0.001

    def run(self):
        global currentTime
        global stop_threads
        start = time.time()
        while True:
            if stop_threads:
                break
            #print('thread running')
            end = time.time()
            currentTime = end-start
            self.timeVal.setText("Time: {:0.2f} Sec".format(currentTime))
            time.sleep(0.09)

    def onTimeButtonFunction(self):
       global sourceState
       global maxTime
       global maxcurrent
       global stop_threads
       global currentTime
       global amp
       global prevAmp
       amp = 0.001
```

```
        prevAmp = 0.001
        GPIO.output(21, GPIO.HIGH)
        sourceState = 1
        self.status.setText("Status: Source Turned On Manually")
        self.status.setStyleSheet("color: black")

        stop_threads = False
        t1 = threading.Thread(target = self.run)
        t1.start()
        while(True):
            QtCore.QTimer.singleShot(0, self.currentSensing)
            if(amp > 0):
                if (prevAmp - amp <= 2):
                    prevAmp = amp
            if(self.nOpenCheck.isChecked()):
                if(GPIO.input(26) == GPIO.HIGH):
                    self.status.setText("Status: Relay Tripped Successfully")
                    self.status.setStyleSheet("color: red")
                    GPIO.output(21, GPIO.LOW)
                    sourceState = 0
                    amp = 0.001
                    break
            if(self.nCloseCheck.isChecked()):
                if(GPIO.input(26) == GPIO.LOW):
                    self.status.setText("Status: Relay Tripped Successfully")
                    self.status.setStyleSheet("color: red")
                    GPIO.output(21, GPIO.LOW)
                    sourceState = 0
                    amp = 0.001
                    break

            if(sourceState == 0):
                self.status.setText("Status: Source Turned Off Manually")
                self.status.setStyleSheet("color: black")
                break
            if(amp > maxcurrent):
                self.status.setText("Status: Maximum Current Reached ")
                self.status.setStyleSheet("color: red")
                GPIO.output(21, GPIO.LOW)
                self.currentVal.setText("Current: {:0.2f} Amps".format(prevAmp))
                sourceState = 0
                break
            if(currentTime >= maxTime):
                self.status.setText("Status: Maximum Time Reached     ")
                self.status.setStyleSheet("color: red")
                GPIO.output(21, GPIO.LOW)
                self.currentVal.setText("Current: {:0.2f} Amps".format(prevAmp))
                sourceState = 0
                break

            QtCore.QCoreApplication.processEvents()
            self.currentVal.setText("Current: {:0.2f} Amps".format(prevAmp))
        amp = 0.001
        stop_threads = True
        t1.join()

    def currentSensing(self):
        global GAIN
        global count
        global maxVal
        global amp
        global prevAmp
        series = [0]*count
        value = 0
        for i in range (count):
            try:
                value = adc.read_adc(0, gain=GAIN)
                time.sleep(0.001)
                series[i] = value
            except:
                print("ADC Read Error")
            QtCore.QCoreApplication.processEvents()
        maxVal = max(series)
        #amp = 0.0014*maxVal + 0.4
        amp = -0.00000000000003*(maxVal**3) + 0.0000000023*(maxVal**2) + 0.0014*maxVal + 0.4
        amp = round(amp,1)

        if(amp < 0.3): amp = 0

        QtCore.QCoreApplication.processEvents()

    def offButtonFunction(self):
       global sourceState
       self.status.setText("Status: Source Turned Off Manually")
       self.status.setStyleSheet("color: black")
       GPIO.output(21, GPIO.LOW)
       self.currentVal.setText("Current: ----- Amps")
       self.timeVal.setText("Time: ------- Sec")
       sourceState = 0

    #@pyqtSlot()
    def toggle(self):
        self._toggle = not self._toggle
        self.nOpenCheck.setChecked(self._toggle)
        self.nCloseCheck.setChecked(not self._toggle)

    def updateGraph(self):
        global currentArray
        global timeArray
        global currentTime
        global prevAmp
        currentArray.append(prevAmp)
        timeArray.append(currentTime)
        print(currentArray)
        print(timeArray)
        self.plotMCB.axes.cla()
        self.plotMCB.axes.set_ylim( bottom = 0, top = 50)
        self.plotMCB.axes.set_xlim( left = 0, right = 50)
        self.plotMCB.axes.set_title("TCC Curve")
        self.plotMCB.axes.set_ylabel("Time (Sec)")
        self.plotMCB.axes.set_xlabel("Current (Amps)")
        self.plotMCB.axes.plot(currentArray,timeArray)
        self.plotMCB.draw()



    def currentChange(self,i):
       global maxcurrent
       maxcurrent = (i+1)*2.5
       #print((i+1)*2.5)

    def timeChange(self,i):
       global maxTime
       maxTime = (i+1)*5
       #print((i+1)*5)

    def resetFunction(self):
       self.timeVal.setText("Time: ------- Sec")
       self.currentVal.setText("Current: ----- Amps")
       self.status.setText("Status: MCB Test Interface Initiated")
       self.status.setStyleSheet("color: black")
       self.plotMCB.axes.cla()
       self.plotMCB.axes.set_ylim( bottom = 0, top = 50)
       self.plotMCB.axes.set_xlim( left = 0, right = 50)
       self.plotMCB.axes.set_title("TCC Curve")
       self.plotMCB.axes.set_ylabel("Time (Sec)")
       self.plotMCB.axes.set_xlabel("Current (Amps)")
       self.plotMCB.draw()


    def closeFunction(self):
       global sourceState
       global prevAmp
       global amp
       GPIO.output(21, GPIO.LOW)
       sourceState = 0
       global currentArray
       global timeArray
       currentArray = []
       timeArray = []
       prevAmp = 0.001
       amp = 0.001
       self.close()

class MatplotlibWidget(FigureCanvas):
    def __init__(self, parent=None, xlim = 50,ylim = 50, hold=False):
        super(MatplotlibWidget, self).__init__(Figure())
        self.setParent(parent)
        self.figure = Figure(figsize=(5, 1),dpi = 95)
        self.canvas = FigureCanvas(self.figure)

        self.axes = self.figure.add_subplot(111)

        self.axes.set_title("TCC Curve")
        self.axes.set_ylabel("Time (Sec)")
        self.axes.set_xlabel("Current (Amps)")
        self.axes.set_xbound(lower = 0, upper = xlim)
        self.axes.set_ylim( bottom = 0, top = 50)

class settingsWindow(QMainWindow):

    def __init__(self):
        global lineList
        global myFont
        global dateFormat
        global timeFormat
        global currentTimeItem
        global currentDateItem
        global currentFontItem

        super().__init__()
        self.setWindowTitle("Settings")
        left = 2
        top = 30
        width = 796
        height = 450
        self.setGeometry(left,  top, width, height)
        self.setFixedSize(width, height)
        wid = QtWidgets.QWidget(self)
        self.setCentralWidget(wid)
        self.setStyleSheet("background-color: white;")

        mainVBox = QVBoxLayout()

        #Designing TopBox
        topBox = QHBoxLayout()
        self.labelTemp = QLabel(self)
        self.labelTemp.setAlignment(Qt.AlignLeft)
        self.labelTemp.setAlignment(Qt.AlignTop)
        self.labelTemp.setFixedHeight(40)
        self.labelTemp.setText("Temp: 36 °C")
        self.labelTemp.setFont(QtGui.QFont(myFont))
        self.labelTime = QLabel(self)
        self.labelTime.setAlignment(Qt.AlignRight)
        self.labelTime.setFixedHeight(40)
        self.labelTime.setText("Time")
        self.labelTime.setFont(QtGui.QFont(myFont))

        topBox.addWidget(self.labelTemp)
        topBox.addWidget(self.labelTime)

        clockTimer = QTimer(self)
        clockTimer.timeout.connect(self.showTime)
        clockTimer.start(1000)
        self.showTime()

        fontBox = QHBoxLayout()
        fontLabel = QLabel()
        fontLabel.setText("Select Font:     ") #5 Spaces After
        fontLabel.setFont(QtGui.QFont(myFont))
        fontList = QComboBox()
        with open("fontList") as f:
              lineList = [line.rstrip('\n') for line in open("fontList")]
        fontList.addItems(lineList)
        fontList.setFont(QtGui.QFont(myFont))
        fontList.setCurrentIndex(currentFontItem)
        fontList.currentIndexChanged.connect(self.fontChange)
        fontBox.addWidget(fontLabel, alignment=Qt.AlignRight)
        fontBox.addWidget(fontList, alignment=Qt.AlignLeft)

        speedBox = QHBoxLayout()
        speedLabel = QLabel()
        speedLabel.setText("Set Fan Speed:     ")
        speedLabel.setFont(QtGui.QFont(myFont))
        speedSlider = QSlider(Qt.Horizontal)
        speedSlider.setFocusPolicy(Qt.StrongFocus)
        speedSlider.setTickPosition(QSlider.TicksBothSides)
        speedSlider.setFixedSize(150,30)
        speedSlider.setMinimum(50)
        speedSlider.setMaximum(100)
        speedSlider.setValue(70)
        speedSlider.setTickInterval(5)
        speedSlider.setSingleStep(5)
        #speedSlider.valueChanged.connect(self.valuechange)
        speedBox.addWidget(speedLabel, alignment=Qt.AlignRight)
        speedBox.addWidget(speedSlider, alignment=Qt.AlignLeft)

        tempBox = QHBoxLayout()
        tempLabel = QLabel()
        tempLabel.setText("Temperature Unit:      ")
        tempLabel.setFont(QtGui.QFont(myFont))
        tempList = QComboBox()
        tempList.addItem("°C")
        tempList.addItem("°F")
        tempList.setFont(QtGui.QFont(myFont))
        tempList.currentIndexChanged.connect(self.tempChange)
        tempBox.addWidget(tempLabel, alignment=Qt.AlignRight)
        tempBox.addWidget(tempList, alignment=Qt.AlignLeft)

        dateBox = QHBoxLayout() #DateFormat dd/mm/yyyy
        dateLabel = QLabel()
        dateLabel.setText("Date Format:     ")
        dateLabel.setFont(QtGui.QFont(myFont))
        dateList = QComboBox()
        dateList.setFont(QtGui.QFont(myFont))
        dateList.addItem("dd/mm/yyyy")
        dateList.addItem("dd/mm/yy")
        dateList.addItem("dd/mmmm/yyyy")
        dateList.addItem("dd/mmmm/yy")
        dateList.addItem("mm/dd/yyyy")
        dateList.addItem("mm/dd/yy")
        dateList.addItem("mmmm/dd/yyyy")
        dateList.addItem("mmmm/dd/yy")
        dateList.setCurrentIndex(currentDateItem)
        dateList.currentIndexChanged.connect(self.dateChange)
        dateBox.addWidget(dateLabel, alignment=Qt.AlignRight)
        dateBox.addWidget(dateList, alignment=Qt.AlignLeft)

        timeBox = QHBoxLayout() #TiemFormat 12/24Hr
        timeLabel = QLabel()
        timeLabel.setText("Time Format:     ")
        timeLabel.setFont(QtGui.QFont(myFont))
        timeList = QComboBox()
        timeList.setFont(QtGui.QFont(myFont))
        timeList.addItem("12hr")
        timeList.addItem("24hr")
        timeList.setCurrentIndex(currentTimeItem)
        timeList.currentIndexChanged.connect(self.timeChange)
        timeBox.addWidget(timeLabel, alignment=Qt.AlignRight)
        timeBox.addWidget(timeList, alignment=Qt.AlignLeft)

        saveButton = QToolButton()
        saveButton.setText("Save Settings")
        saveButton.setFont(QtGui.QFont(myFont))
        saveButton.setFixedSize(113,30)
        saveButton.clicked.connect(self.saveSettings)






        mainVBox.addLayout(topBox)
        mainVBox.addLayout(fontBox)
        mainVBox.addLayout(speedBox)
        mainVBox.addLayout(tempBox)
        mainVBox.addLayout(timeBox)
        mainVBox.addLayout(dateBox)
        mainVBox.addWidget(saveButton)


        wid.setLayout(mainVBox)

        self.show()

    def showTime(self):
       now = datetime.datetime.now()
       now = now.strftime(dateFormat+"\n"+timeFormat)
       self.labelTime.setText(now)

    def fontChange(self,i):
        global myFont
        global lineList
        global currentFontItem

        #print(lineList[i])
        myFont = lineList[i]
        currentFontItem = i

    def tempChange(self,i):
        global tempUnit

        if(i==0): tempUnit = "C"

        if(i==1): tempUnit = "F"

    def dateChange(self,i):
        global dateFormat
        global currentDateItem

        if(i==0): dateFormat="%a %d-%m-%Y"
        if(i==1): dateFormat="%a %d-%m-%y"
        if(i==2): dateFormat="%a %d-%b-%Y"
        if(i==3): dateFormat="%a %d-%b-%y"
        if(i==4): dateFormat="%a %m-%d-%Y"
        if(i==5): dateFormat="%a %m-%d-%y"
        if(i==6): dateFormat="%a %b-%d-%Y"
        if(i==7): dateFormat="%a %b-%d-%y"



        currentDateItem= i


    def timeChange(self,i):
        global timeFormat
        global currentTimeItem

        if(i==0): timeFormat = "%I:%M:%S %p"

        if(i==1): timeFormat = "%H:%M:%S"

        currentTimeItem = i

    def saveSettings(self):

```

```python
        self.w = Window()
        self.w.show()
        self.close()

class howtoWindow(QScrollArea):

    def __init__(self):

        global myFont
        super().__init__()
        self.setWindowTitle("How To Use")
        left = 2
        top = 30
        width = 796
        height = 450
        self.setGeometry(left,  top, width, height)
        self.setFixedSize(width, height)
        self.setStyleSheet("background-color: white;")

        widget = QWidget()
        layout = QVBoxLayout(widget)
        layout.setAlignment(Qt.AlignTop)

        Title = QLabel("How To Use")
        Title.setFont(QtGui.QFont(myFont,26,QtGui.QFont.Bold))
        layout.addWidget(Title,alignment=Qt.AlignCenter)

        Title0 = QLabel("Power Supply:")
        Title0.setFont(QtGui.QFont(myFont,16,QtGui.QFont.Bold))
        layout.addWidget(Title0)

        description0 = QLabel("Following conditions should be met before powering the equipment\n"\
                             "• Supply Voltage is reliable 220V (+-10%)\n"\
                             "• Supply Frequency is reliable 50Hz (+-2%)\n"\
                             "• Voltage Stabilizer should be used in case of unreliable supply\n"\
                              "• Supply should be free of harmonics (Do not operate on UPS)")
        description0.setFont(QtGui.QFont(myFont,13))
        layout.addWidget(description0)

        Title1 = QLabel("Terminals Detail:")
        Title1.setFont(QtGui.QFont(myFont,16,QtGui.QFont.Bold))
        layout.addWidget(Title1)

        description = QLabel("There are 4 terminals on back and 6 terminals on front which are marked
accordingly\n"\
                             ">>> Back Terminals\n"\
                             "    • Two back terminals are used to power the variac\n"\
                             "    • Two back terminals are used to get output from variac\n"\
                             ">>> Front Terminals\n"\
                             "    • Two front terminals are rated at 40 Amps\n"\
                             "    • Two front terminals are rated at 15 Amps\n"\
                             "    • Two front terminals are used to get state of relay contacts\n")
        description.setFont(QtGui.QFont(myFont,13))
        layout.addWidget(description)

        Title2 = QLabel("MCB Test:")
        Title2.setFont(QtGui.QFont(myFont,16,QtGui.QFont.Bold))
        layout.addWidget(Title2)

        description2 = QLabel("MCBs upto 40 Amps can be tested with this equipment (1/2 Pole)\n\n"\
                             "• Make connections with the MCB\n"\
                             "• Turn MCB on while keeping variac at 0\n"\
                             "• Go to MCB Test Window\n"\
                             "• Turn on the injection by pressing \"On\" button\n"\
                             "• Set desired current by incresing variac voltage\n"\
                              "• Turn off the injection after achieving desired current\n"\
                              "• Wait for a second\n"\
                              "• Again turn on the injection by pressing \"On + Time\" button\n"\
                              "• Wait for the MCB to trip\n"\
                              "• Get results")
        description2.setFont(QtGui.QFont(myFont,13))
        layout.addWidget(description2)

        Title3 = QLabel("Relay Test:")
        Title3.setFont(QtGui.QFont(myFont,16,QtGui.QFont.Bold))
        layout.addWidget(Title3)

        description3 = QLabel("Single Phase Relays upto 40 Amps can be tested with this 
equipment\n\n"\
                             "• Keep Variac at 0\n"\
                             "• Make connections with the Relay\n"\
                             "• Go to Relay Test Window\n"\
                             "• Set the Initial state of relay i.e. NO/NC\n"
                             "• Turn on the injection by pressing \"On\" button\n"\
                             "• Set desired current by incresing variac voltage\n"\
                              "• Turn off the injection after achieving desired current\n"\
                              "• Wait for a second\n"\
                              "• Again turn on the injection by pressing \"On + Time\" button\n"\
                              "• Wait for the Relay to trip\n"\
                              "• Get results")
        description3.setFont(QtGui.QFont(myFont,13))
        layout.addWidget(description3)

        howto1L = QLabel(self)
        howto1P = QPixmap('howto1.PNG')
        howto1L.setPixmap(howto1P)
        howto1L.setFixedSize(488,400)
        layout.addWidget(howto1L,alignment=Qt.AlignCenter)

        Title4 = QLabel("Current/Time Settings:")
        Title4.setFont(QtGui.QFont(myFont,16,QtGui.QFont.Bold))
        layout.addWidget(Title4)

        description4 = QLabel("There are various current and time ranges given in the equipment to 
protect the device being tested\n\n"\
                             "• Select the Maximum current you want to inject\n"\
                             "• Select the Maximum time for test operation\n"\
                             "• The equipment will automatically abort test if current value is 
increased from preset value\n"\
                             "• The equipment will automatically abort test if test time in increased 
from preset value")

        description4.setFont(QtGui.QFont(myFont,13))
        layout.addWidget(description4)

        howto2L = QLabel(self)
        howto2P = QPixmap('howto2.PNG')
        howto2L.setPixmap(howto2P)
        #howto2L.setFixedSize(488,400)
        layout.addWidget(howto2L,alignment=Qt.AlignCenter)

        Title5 = QLabel("Graph and Data Logging:")
        Title5.setFont(QtGui.QFont(myFont,16,QtGui.QFont.Bold))
        layout.addWidget(Title5)

        description5 = QLabel("The equipment has plotting  and data logging capabilities \n\n"\
                              "• After successful conduction of test, press \"Update Graph\" button to show values on graph\n"\
                              "• The plotting feature is available for both Relays and MCB\n"\
                              "• Press \"Log Data\" button to log the most resent result\n"\
                              "• All the logged results can be viewed in \"Logged Data\" window on Main Screen\n"
                               "• Reset button is used to clear the Test Screen\n"\
                               "• \"Status\" shows the most recent state of equipment")

        description5.setFont(QtGui.QFont(myFont,13))
        layout.addWidget(description5)

        howto3L = QLabel(self)
        howto3P = QPixmap('howto3.PNG')
        howto3L.setPixmap(howto3P)
        #howto2L.setFixedSize(488,400)
        layout.addWidget(howto3L,alignment=Qt.AlignCenter)

        self.backButton = QToolButton()
        self.backButton.setFixedSize(60,30)
        self.backButton.clicked.connect(self.closeFunction)
        pixmapBack = QPixmap("Back.png")
        self.backButton.setIcon(QIcon(pixmapBack))
        self.backButton.setIconSize(QSize(60,35))
        layout.addWidget(self.backButton)

        self.setWidget(widget)
        self.show()

    def closeFunction(self):
       self.close()

class aboutWindow(QScrollArea):

    def __init__(self):

        global myFont
        super().__init__()
        self.setWindowTitle("About")
        left = 2
        top = 30
        width = 796
        height = 450
        self.setGeometry(left,  top, width, height)
        self.setFixedSize(width, height)
        self.setStyleSheet("background-color: white;")

        widget = QWidget()
        layout = QVBoxLayout(widget)
        layout.setAlignment(Qt.AlignTop)

        descriptionTitle = QLabel("Description")
        descriptionTitle.setFont(QtGui.QFont(myFont,26,QtGui.QFont.Bold))
        layout.addWidget(descriptionTitle ,alignment=Qt.AlignCenter)

        description = QLabel("Most test units currently used are based on power electronics circuitry and hence are very expensive. Most\n"\
                              "industries cannot afford such very expensive equipment and hence they hire third parties for relay testing \n"\
                             "which is also expensive. This relay testing unit is economical, user friendly and can test over current relays, \n"\
                             "earth fault relays and reverse power relays along with the wide range of miniature circuit breakers. The test set \n"\
                             "is designed to perform the secondary injection testing by artificially injecting the fault currents in controlled \n"\
                             "manner and find out the tripping time of protective relays and miniature circuit breakers.\nFeatures of equipment are; \n\n"\
                             "• User Friendly Graphical User Interface\n"\
                             "• Rigid Equipment Design\n"\
                             "• Reliable for use in Industrial Settings\n"\
                             "• Manual Cooling System\n"\
                             "• Built-in Protection System\n"\
                             "• TCC Curve Plotting\n"\
                             "• Report Generator\n"\
                             "• Wide Range of Settings\n")


        description.setFont(QtGui.QFont(myFont,13))
        layout.addWidget(description)

        specificationTitle = QLabel("Specifications")
        specificationTitle.setFont(QtGui.QFont(myFont,26,QtGui.QFont.Bold))
        layout.addWidget(specificationTitle ,alignment=Qt.AlignCenter)
        specificationTable = QTableWidget()
        specificationTable.setRowCount(11)
        specificationTable.setColumnCount(2)
        specificationTable.setFixedSize(312,350)
        specificationTable.setHorizontalHeaderLabels(("Specification","Value"))
        specificationTable.horizontalHeaderItem(0).setFont(QtGui.QFont(myFont,13,QtGui.QFont.Bold))
        specificationTable.horizontalHeaderItem(1).setFont(QtGui.QFont(myFont,13,QtGui.QFont.Bold))

        specificationTable.setItem(0,0,QTableWidgetItem("Rated Supply"))
        specificationTable.setItem(0,1,QTableWidgetItem("220 V"))
        specificationTable.setItem(1,0,QTableWidgetItem("Operating Temperature (Min)"))
        specificationTable.setItem(1,1,QTableWidgetItem("0 °C"))
        specificationTable.setItem(2,0,QTableWidgetItem("Operating Temperature (Max)"))
        specificationTable.setItem(2,1,QTableWidgetItem("70 °C"))
        specificationTable.setItem(3,0,QTableWidgetItem("Max Current Output"))
        specificationTable.setItem(3,1,QTableWidgetItem("40 Amps"))
        specificationTable.setItem(4,0,QTableWidgetItem("Max Terminal Voltage"))
        specificationTable.setItem(4,1,QTableWidgetItem("24 Volts"))
        specificationTable.setItem(5,0,QTableWidgetItem("Operating Time (0-5 Amps)"))
        specificationTable.setItem(5,1,QTableWidgetItem("30 Mins"))
        specificationTable.setItem(6,0,QTableWidgetItem("Operating Time (5-10 Amps)"))
        specificationTable.setItem(6,1,QTableWidgetItem("20 Mins"))
        specificationTable.setItem(7,0,QTableWidgetItem("Operating Time (10-15 Amps)"))
        specificationTable.setItem(7,1,QTableWidgetItem("15 Mins"))
        specificationTable.setItem(8,0,QTableWidgetItem("Operating Time (15-20 Amps)"))
        specificationTable.setItem(8,1,QTableWidgetItem("10 Mins"))
        specificationTable.setItem(9,0,QTableWidgetItem("Operating Time (20-30 Amps)"))
        specificationTable.setItem(9,1,QTableWidgetItem("5 Mins"))
        specificationTable.setItem(10,0,QTableWidgetItem("Operating Time (above 30 Amps)"))
        specificationTable.setItem(10,1,QTableWidgetItem("1 Min"))
        specificationTable.resizeColumnsToContents()
        specificationTable.resizeRowsToContents()
        specificationTable.verticalHeader().setVisible(False)
        layout.addWidget(specificationTable,alignment=Qt.AlignCenter)

        groupTitle = QLabel("\nManufacturer Details")
        groupTitle.setFont(QtGui.QFont(myFont,26,QtGui.QFont.Bold))
        layout.addWidget(groupTitle ,alignment=Qt.AlignCenter)

        descriptionGroup = QLabel("This test set is manufactured by Group-44  Batch 2015-16\n\n"
                             "• Omama Zaheen (EE-15136)\n"\
                             "• Muhamamd Raaid khan (EE-15141)\n"\
                             "• Hammad Junaid (EE-15146)\n"\
                             "• Uzair Ali Khan (EE-15156)\n\n"\
                             "Under the supervision of:\n\n"\
                             "• Internal Examiner: Muhammad Farooq Siddiqui\n"\
                             "• External Examiner: Muhammad Humaid Saeed\n"\
                             )


        descriptionGroup.setFont(QtGui.QFont(myFont,13))
        layout.addWidget(descriptionGroup)

        self.backButton = QToolButton()
        self.backButton.setFixedSize(60,30)
        self.backButton.clicked.connect(self.closeFunction)
        pixmapBack = QPixmap("Back.png")
        self.backButton.setIcon(QIcon(pixmapBack))
        self.backButton.setIconSize(QSize(60,35))
        layout.addWidget(self.backButton)

        self.setWidget(widget)
        self.show()

    def closeFunction(self):
       self.close()

if __name__ == "__main__":

    App = QApplication(sys.argv)
    print("Window Initiated")
    window = Window()
    sys.exit(App.exec())

```