"""

Interface to utilize Accounting Tool

"""

import sys

import csv

from PyQt5.QtWidgets import \*

from PyQt5.QtGui import \*

from PyQt5.QtCore import \*

from DBManager import DBManager

from LogManager import LOG

import json

from PyQt5.QtWidgets import QMessageBox

class Find(QDialog):

def \_\_init\_\_(self,root, parent=None):

self.root = root

QDialog.\_\_init\_\_(self,self.root, parent)

self.initUI()

def initUI(self):

self.lb1 = QLabel("Search for: ", self)

self.lb1.setStyleSheet("font-size: 15px; ")

self.lb1.move(10, 10)

self.te = QTextEdit(self)

self.te.move(10, 40)

self.te.resize(250, 25)

self.src = QPushButton("Find", self)

self.src.move(270, 40)

self.lb2 = QLabel("Replace all by: ", self)

self.lb2.setStyleSheet("font-size: 15px; ")

self.lb2.move(10, 80)

self.rp = QTextEdit(self)

self.rp.move(10, 110)

self.rp.resize(250, 25)

self.rpb = QPushButton("Replace", self)

self.rpb.move(270, 110)

self.opt1 = QCheckBox("Case sensitive", self)

self.opt1.move(10, 160)

self.opt1.stateChanged.connect(self.CS)

self.opt2 = QCheckBox("Whole words only", self)

self.opt2.move(10, 190)

self.opt2.stateChanged.connect(self.WWO)

self.close = QPushButton("Close", self)

self.close.move(270, 220)

self.close.clicked.connect(self.Close)

self.setGeometry(300, 300, 360, 250)

def CS(self, state):

global cs

if state == QtCore.Qt.Checked:

cs = True

else:

cs = False

def WWO(self, state):

global wwo

print(wwo)

if state == QtCore.Qt.Checked:

wwo = True

else:

wwo = False

def Close(self):

self.close()

class App(QMainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

## 1

self.title = "Vendor Details"

self.initUI()

def initUI(self):

self.setWindowTitle(self.title)

# create a central widget and everything to it

centralWidget = CentralWidget(self)

self.setCentralWidget(centralWidget)

self.statusBar().showMessage('--')

self.showMaximized()

self.show()

findAction = QAction(QIcon("icons/find.png"),"Find",self)

findAction.setStatusTip("Find words in your document")

findAction.setShortcut("Ctrl+F")

findAction.triggered.connect(self.Find)

self.toolbar = self.addToolBar("Options")

self.toolbar.addAction(findAction)

def Find(self):

global f

find = Find(self)

find.show()

def handleFind():

f = find.te.toPlainText()

print(f)

if cs == True and wwo == False:

flag = QTextDocument.FindBackward and QTextDocument.FindCaseSensitively

elif cs == False and wwo == False:

flag = QTextDocument.FindBackward

elif cs == False and wwo == True:

flag = QTextDocument.FindBackward and QTextDocument.FindWholeWords

elif cs == True and wwo == True:

flag = QTextDocument.FindBackward and QTextDocument.FindCaseSensitively and QTextDocument.FindWholeWords

self.text.find(f, flag)

def handleReplace():

f = find.te.toPlainText()

r = find.rp.toPlainText()

text = self.text.toPlainText()

newText = text.replace(f, r)

self.text.clear()

self.text.append(newText)

class CentralWidget(QWidget):

def \_\_init\_\_(self, parent):

super(QWidget, self).\_\_init\_\_(parent)

# self.layout = QVBoxLayout(self)

# Initialize Tab Screens

self.tabs = QTabWidget()

# HOME TAB CODE

self.homeTab = HomeWidget(self)

self.horizontalGroupBox = QGroupBox()

self.layout = QGridLayout(self)

self.layout.addWidget(self.homeTab.tableWidget, 0, 0)

exportButton = QPushButton('Export')

exportButton.clicked.connect(self.onExportClickEvent)

self.layout.addWidget(exportButton, 1, 1)

self.setLayout(self.layout)

@pyqtSlot()

def onExportClickEvent(self):

try:

with open("Report.csv", "w") as csvFile:

csvWriter = csv.writer(csvFile, delimiter=',')

dbManager = DBManager()

dbManager.openConnection()

dbManager.initDB()

# create table to store the tenant/owner maintenance data

tableName = ""

## 2

tableCommand = '''SELECT \* from vendor\_details'''

csvWriter.writerows(dbManager.selectData(tableName, tableCommand))

msg = QMessageBox()

msg.setText("Report Generated!");

msg.exec\_()

except csv.Error as csve:

LOG.error(csve)

class HomeWidget(QWidget):

def \_\_init\_\_(self, parent):

super(QWidget, self).\_\_init\_\_(parent)

self.createTable()

self.drawTable()

def createTable(self):

# Create table

self.tableWidget = QTableWidget()

self.tableWidget.setSortingEnabled(True)

##3

self.tableWidget.setRowCount(10)

self.tableWidget.setColumnCount(6)

## 4

self.tableWidget.setHorizontalHeaderLabels(["Vendor ID", "Vendor Name", "Address",

"City","State", "Phone Number"])

# self.tableWidget.resizeColumnsToContents()

self.tableWidget.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)

self.tableWidget.doubleClicked.connect(self.onFlatClickEvent)

def insertIntoTable(self, row):

for rowItem in row:

tableWidgetItem = QTableWidgetItem(str(rowItem))

tableWidgetItem.setFlags(Qt.ItemIsSelectable | Qt.ItemIsEnabled)

self.tableWidget.setItem(self.rowIndex, self.columnIndex, tableWidgetItem)

self.columnIndex += 1

def drawTable(self):

dbManager = DBManager()

dbManager.openConnection()

dbManager.initDB()

## 5

tableName = "vendor\_details"

tableCommand = '''SELECT \* from vendor\_details'''

self.rowIndex = 0

self.columnIndex = 0

for row in dbManager.selectData(tableName, tableCommand):

# insert the rows into table

self.insertIntoTable(row)

self.rowIndex += 1

self.columnIndex = 0

def redrawTable(self):

self.tableWidget.clearContents()

self.drawTable()

@pyqtSlot()

def onFlatClickEvent(self):

for currentQTableWidgetItem in self.tableWidget.selectedItems():

if currentQTableWidgetItem.column() is 0: # this is to ensure that we are manipulating user data only based on primary key

print(currentQTableWidgetItem.row(), currentQTableWidgetItem.column(), currentQTableWidgetItem.text())

dialog = Dialog(currentQTableWidgetItem.text())

dialog.exec\_()

self.redrawTable()

LOG.info("Dialog Execution is done....Hurray !! Re-draw table")

class Dialog(QDialog):

flatNumber = None

def \_\_init\_\_(self, selectedText):

super(Dialog, self).\_\_init\_\_()

self.flatNumber = selectedText

self.createFormGroupBox()

buttonBox = QDialogButtonBox(QDialogButtonBox.Ok | QDialogButtonBox.Cancel)

buttonBox.accepted.connect(self.onAcceptEvent)

buttonBox.rejected.connect(self.reject)

mainLayout = QVBoxLayout()

mainLayout.addWidget(self.formGroupBox)

mainLayout.addWidget(buttonBox)

self.setLayout(mainLayout)

self.setWindowTitle("Flat No: " + self.flatNumber)

self.show()

def createFormGroupBox(self):

self.formGroupBox = QGroupBox()

layout = QFormLayout()

# Apply changes to everything, it is upto users to be careful while changing the values

# fetch the occupant name from db and put it here

dbManager = DBManager()

dbManager.openConnection()

dbManager.initDB()

##6

tableName = "vendor\_details"

tableCommand = "SELECT \* FROM vendor\_details WHERE v\_id=?"

searchKey = (self.flatNumber,)

flatDetails = dbManager.selectData(tableName, tableCommand, searchKey)

LOG.info("Selected Flat Details: %s" % (flatDetails,))

self.OccupantNameLineEdit = QLineEdit()

self.OccupantNameLineEdit.setText(flatDetails[1])

layout.addRow(QLabel("Occupants Name: "), self.OccupantNameLineEdit)

# calendar for date of payment

self.dateOfPaymentWidget = QCalendarWidget()

self.dateOfPaymentWidget.setGridVisible(True)

self.dateOfPaymentWidget.setFirstDayOfWeek(Qt.Monday)

self.dateOfPaymentWidget.clicked[QDate].connect(self.setSelectedDate)

self.dateOfPaymentLabel = QLabel("Date of Payment: ")

self.dateOfPaymentValueLabel = QLabel("")

self.dateOfPaymentValueLabel.setText(self.dateOfPaymentWidget.selectedDate().toString("yyyy-MM-dd"))

layout.addRow(self.dateOfPaymentLabel, self.dateOfPaymentValueLabel)

layout.addRow(self.dateOfPaymentWidget)

# Mode of payment

self.modeOfPaymentComboBox = QComboBox()

modeOfPayment = ["Cash", "NEFT", "IMPS", "RTGS", "cheque"]

for mode in modeOfPayment:

self.modeOfPaymentComboBox.addItem(mode)

layout.addRow(QLabel("Mode of Payment: "), self.modeOfPaymentComboBox)

# TODO: calculate remaining amount automatically

# for now just group them provide the remaining the reference table

self.amountPaidLineEdit = QLineEdit()

self.amountPaidLineEdit.setText(str(flatDetails[3]))

layout.addRow(QLabel("Amount Paid: "), self.amountPaidLineEdit)

self.amountDueLineEdit = QLineEdit()

self.amountDueLineEdit.setText(str(flatDetails[4]))

layout.addRow(QLabel("Amount Due: "), self.amountDueLineEdit)

# self.textFieldEnabled = QCheckBox()

# layout.addRow(QLabel("Check this box to edit the text fields"), self.textFieldEnabled)

# # Disable text field

# # self.OccupantNameLineEdit.setDisabled(self.textFieldEnabled.checkState() == Qt.Unchecked)

# # self.amountPaidLineEdit.setDisabled(self.textFieldEnabled.checkState() == Qt.Unchecked)

# # self.amountDueLineEdit.setDisabled(self.textFieldEnabled.checkState() == Qt.Unchecked)

# self.textFieldEnabled.stateChanged.connect(self.enableTextField)

self.formGroupBox.setLayout(layout)

def setSelectedDate(self, dateOfPayment):

self.dateOfPaymentValueLabel.setText(dateOfPayment.toString("yyyy-MM-dd"))

def getOccupantName(self):

return self.OccupantNameLineEdit.text()

def getDateOfPayment(self):

return self.dateOfPaymentValueLabel.text()

def getAmountPaid(self):

return self.amountPaidLineEdit.text()

def getAmountDue(self):

with open('UserSettings.json') as datastore:

initialUserSettings = json.load(datastore)

LOG.info("INITIAL USER SETTINGS: " + json.dumps(initialUserSettings))

currentMaintenanceAmt = initialUserSettings["currentMaintenanceAmt"]

return currentMaintenanceAmt - float(self.amountPaidLineEdit.text())

def getModeOfPayment(self):

return self.modeOfPaymentComboBox.currentText()

def enableTextField(self, state):

self.OccupantNameLineEdit.setEnabled(state != Qt.Unchecked)

self.amountPaidLineEdit.setEnabled(state != Qt.Unchecked)

self.amountDueLineEdit.setEnabled(state != Qt.Unchecked)

def onAcceptEvent(self):

# fetch values from the field and populate the data base, also close the dialog

# fetch values in corresponding variables

occupantName = self.getOccupantName()

dateOfPayment = self.getDateOfPayment()

amountPaid = self.getAmountPaid()

amountDue = self.getAmountDue()

paymentMode = self.getModeOfPayment()

flatDetails = (occupantName, dateOfPayment, amountPaid, amountDue, paymentMode, self.flatNumber)

print(str(flatDetails))

dbManager = DBManager()

dbManager.openConnection()

dbManager.initDB()

tableName = "vendor\_details"

tableCommand = """UPDATE vendor\_details SET OCCUPANT\_NAME=?,DATE=?,AMT\_PAID=?,AMT\_DUE=?,MODE\_OF\_PAYMENT=? WHERE FLATNO=?"""

dbManager.updateData(tableName, tableCommand, flatDetails)

self.close()

if \_\_name\_\_ == "\_\_main\_\_":

app = QApplication(sys.argv)

ex = App()

sys.exit(app.exec\_())