

Laboratory Activity 5 - Introduction to Event Handling in GUI Development

CALVIN EARL PLANTA

10/21/2024

CPE 009B - CPE21S4

Prof. Sayo

6. Supplementary Activity:

Task

Using the simple Account Registration system created in the previous laboratory activity, create the functionality that will register the account by saving it to a Database using sqllitedict or a .csv or .txt file.

The GUI program should not allow the registration to proceed if there is a field that has an empty value and notify of the missing values using a message box. Once the registration is successful a message should appear that would inform the user that the registration was successful. Use the appropriate symbol in making the message box.

registration.py

Python

```
import sys
from PyQt5.QtWidgets import QWidget, QLineEdit, QPushButton, QApplication,
QLabel, QMessageBox
from PyQt5.QtGui import QIcon
```

```
class App(QWidget):
```

```
    def __init__(self):
        super().__init__()
        self.title = "PyQt Registration"
        self.x = 200
        self.y = 200
        self.width = 300
        self.height = 300
        self.initUI()
```

```
    def initUI(self):
        self.setWindowTitle(self.title)
        self.setGeometry(self.x, self.y, self.width, self.height)
        self.setWindowIcon(QIcon('pythonico.ico'))
```

```
        self.textboxlbl = QLabel("Registration", self)
        self.textboxlbl.move(120, 20)
```

```

self.textboxlbl2 = QLabel("First name:", self)
self.textboxlbl2.move(20, 70)

self.textboxentry = QLineEdit(self)
self.textboxentry.move(80, 70)

self.textboxlbl3 = QLabel("Last name:", self)
self.textboxlbl3.move(20, 100)

self.textboxentry2 = QLineEdit(self)
self.textboxentry2.move(80, 100)

self.textboxlbl4 = QLabel("Username:", self)
self.textboxlbl4.move(20, 130)

self.textboxentry3 = QLineEdit(self)
self.textboxentry3.move(80, 130)

self.textboxlbl5 = QLabel("Password:", self)
self.textboxlbl5.move(20, 160)

self.textboxentry4 = QLineEdit(self)
self.textboxentry4.move(80, 160)
self.textboxentry4.setEchoMode(QLineEdit.Password)

self.textboxlbl6 = QLabel("Email:", self)
self.textboxlbl6.move(20, 190)

self.textboxentry5 = QLineEdit(self)
self.textboxentry5.move(80, 190)

self.button = QPushButton('Submit', self)
self.button.clicked.connect(self.submit)
self.button.move(50, 230)

self.result_label = QLabel("", self)
self.result_label.move(80, 280)
self.button2 = QPushButton('Clear', self)

self.button2.move(150, 230)
self.button2.clicked.connect(self.clear)

def submit(self):
    empty_fields = []
    if not self.textboxentry.text():
        empty_fields.append("First name")
    if not self.textboxentry2.text():
        empty_fields.append("Last name")
    if not self.textboxentry3.text():
        empty_fields.append("Username")

```

```

        if not self.textboxentry4.text():
            empty_fields.append("Password")
        if not self.textboxentry5.text():
            empty_fields.append("Email")

        if empty_fields:
            error_message = "Please fill the following fields:\n" +
"\n".join(empty_fields)
            QMessageBox.warning(self, "Input Error", error_message,
QMessageBox.Ok)
        else:
            first_name = self.textboxentry.text()
            last_name = self.textboxentry2.text()
            username = self.textboxentry3.text()
            password = self.textboxentry4.text()
            email = self.textboxentry5.text()

            with open('credentials.txt', 'a') as file:
                file.write(f"First name: {first_name}\nLast name:
{last_name}\nUsername: {username}\nPassword: {password}\nEmail: {email}")

            QMessageBox.information(self, "Success", "Registration
successful!", QMessageBox.Ok)
            print(f"User registered: {first_name}, {last_name}, {username},
{email}")

            self.clear()

        def clear(self):
            self.textboxentry.clear()
            self.textboxentry2.clear()
            self.textboxentry3.clear()
            self.textboxentry4.clear()
            self.textboxentry5.clear()

    if __name__ == '__main__':
        app = QApplication(sys.argv)
        ex = App()
        sys.exit(app.exec_())

```

main.py

```

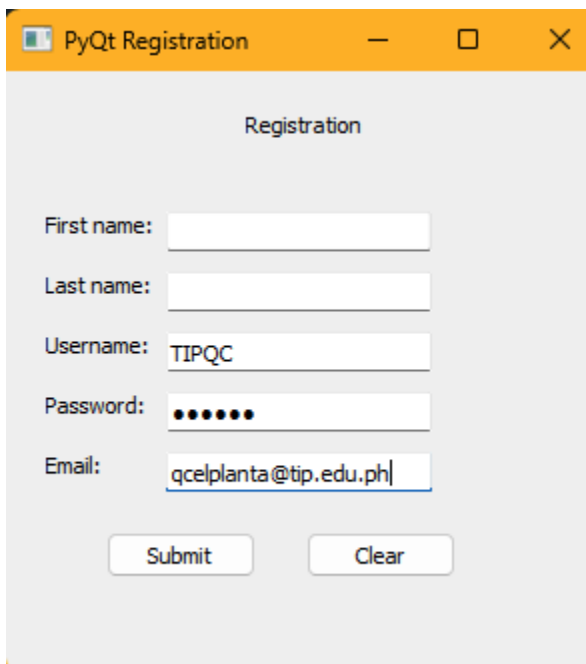
Python
import sys
from PyQt5.QtWidgets import QWidget, QLineEdit, QPushButton, QApplication,
QLabel

```

```
from PyQt5.QtGui import QIcon
from registration import App

if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = App()
    ex.show()
    sys.exit(app.exec_())
```

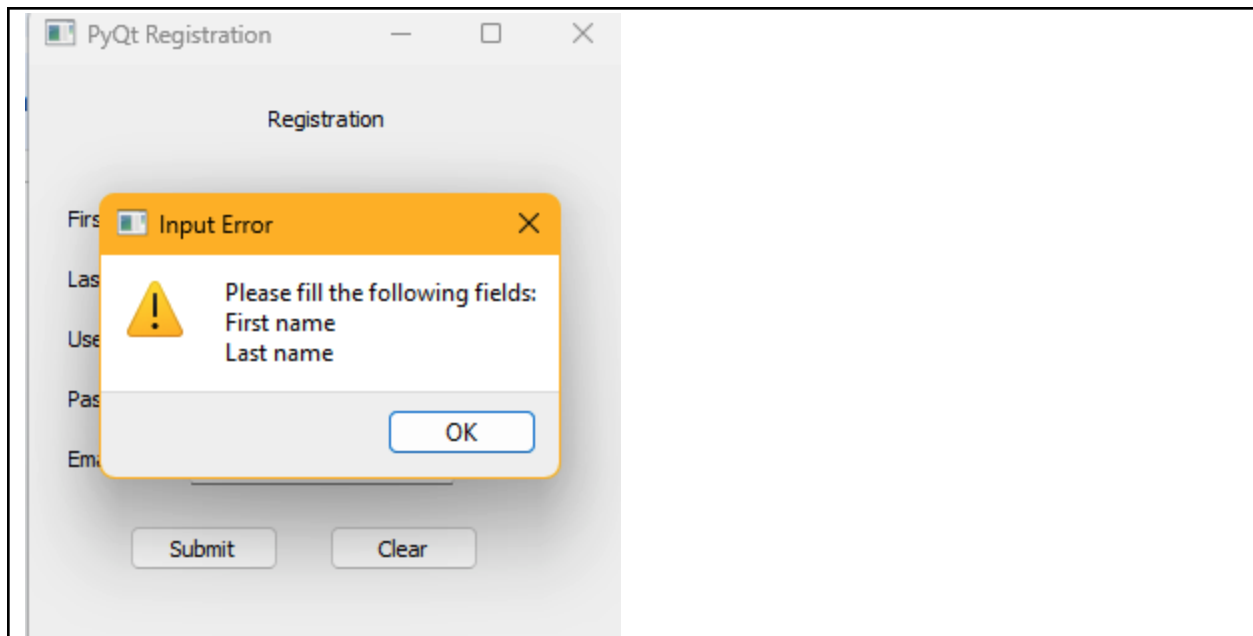
Missing fields



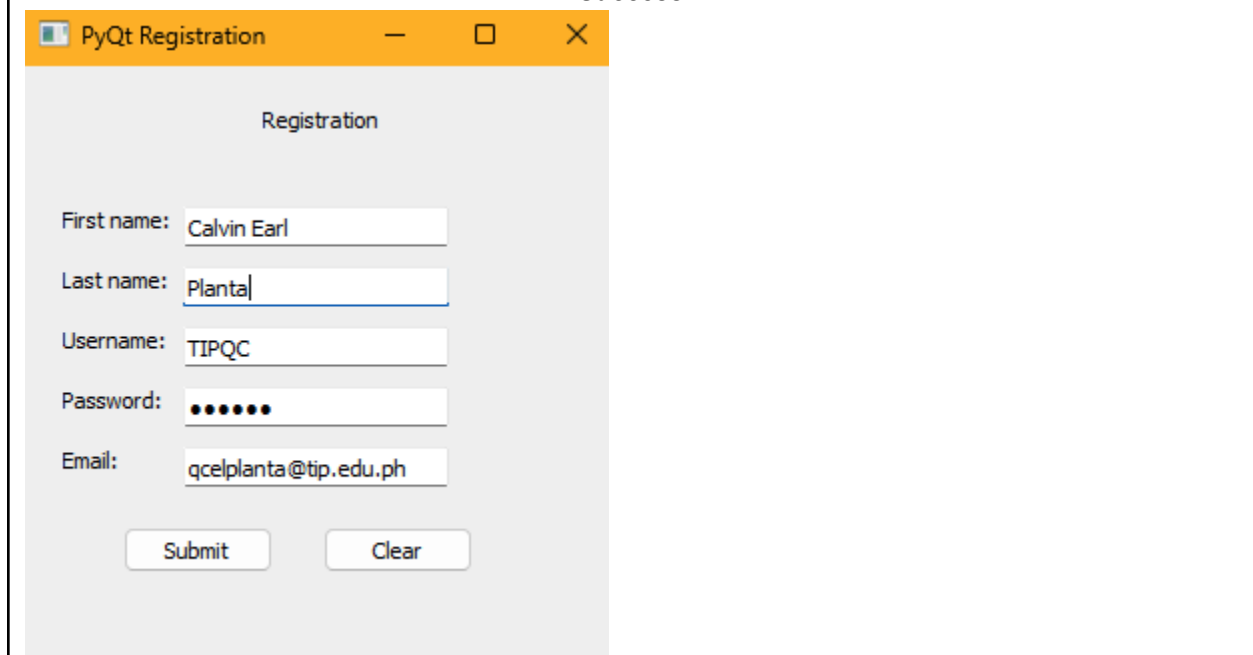
The screenshot shows a window titled "PyQt Registration" with a light gray background. The window contains a registration form with the following fields and values:

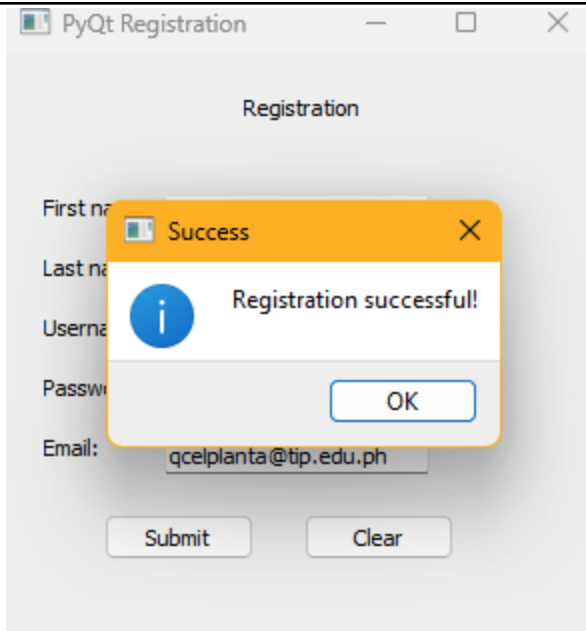
- First name:
- Last name:
- Username:
- Password:
- Email:

At the bottom of the form, there are two buttons: "Submit" and "Clear".

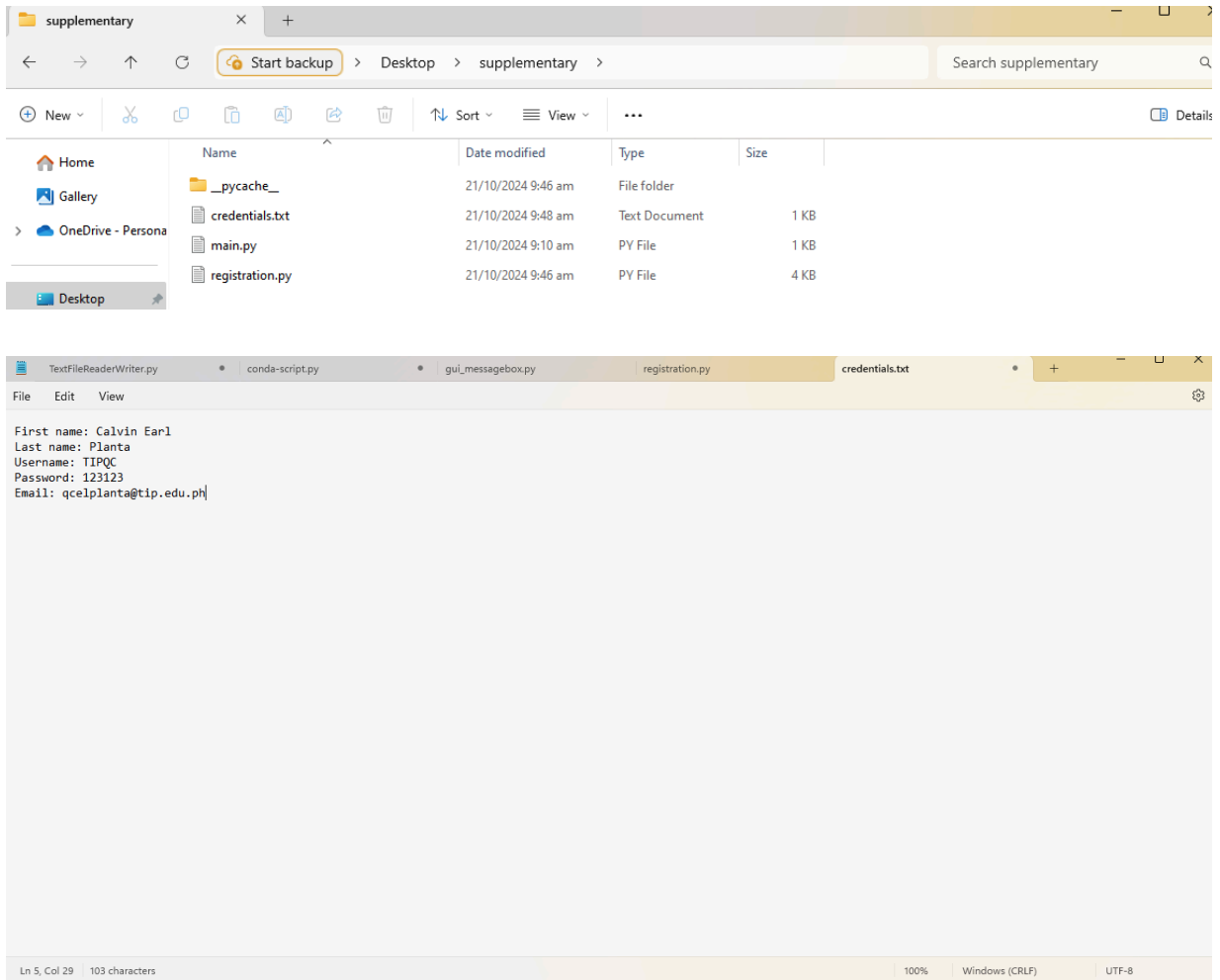


Success





Saved credentials



Questions

1. What are the other signals available in PyQt5? (give at least 3 and describe each)

- **fileSelected()** - A signal under the QFileDialog widget. It is emitted when the user selects the file.
- **textChanged()** - A signal under the QLineEdit widget. It is emitted when the user modifies the text in the field.
- **sliderMoved()** - A signal under the QSlider widget. It is emitted when the user moves the slider.

2. Why do you think that event handling in Python is divided into signals and slots?

- By dividing event handling into signals and slots, it would make more sense since the logic is clear by that concept. Signals are being emitted by objects when the user takes action, like clicking buttons or changing values to spinboxes. Slots on the other hand, are the methods that are called in response to the signal. By dividing event-handling into these two, we could create organized applications.

3. How can message boxes be used to provide a better User Experience or how can message boxes be used to make a GUI Application more user-friendly?

- By utilizing message boxes, it can help avoid confusions by specifying what is happening when the program is running. Accompanied with the appropriate symbols for each message box, it can help clarify the error the user has made, or if the user has successfully executed their desired action, further lessening confusions and guiding them in their decision makings.

4. What is Error-handling and how was it applied in the task performed?

- Error-Handling is the response or management of errors that may occur whenever a program is being executed. In the task that I performed, error-handling is shown evidently when I tried to implement the program wherein the submission of the registration credentials will not be accepted if there are any fields skipped. I was able to implement error-handling by using a message box pop up that tells what went wrong upon entering the user's input in the registration window.

5. What may be the reasons behind the need to implement error handling?

- Implementation of error-handling in programs is essential because it helps us debug our programs if errors occur. Additionally, it also helps us in managing how to handle unexpected situations just in case instances where the flow of the program changes, causing errors in the program. Lastly, it makes the program more user-friendly, creating less confusions and frustrations whenever an error occurs in the program.

7. Conclusion:

Event handling is critical to the interactive nature of applications. We can design responsive elements like buttons and text entries by understanding event-handlers like clicks and key presses. This newfound knowledge has improved my programming skills and has broadened my knowledge in OOP in creating interactive applications. All things considered, event handling is crucial to GUI creation and allows for more creative options.