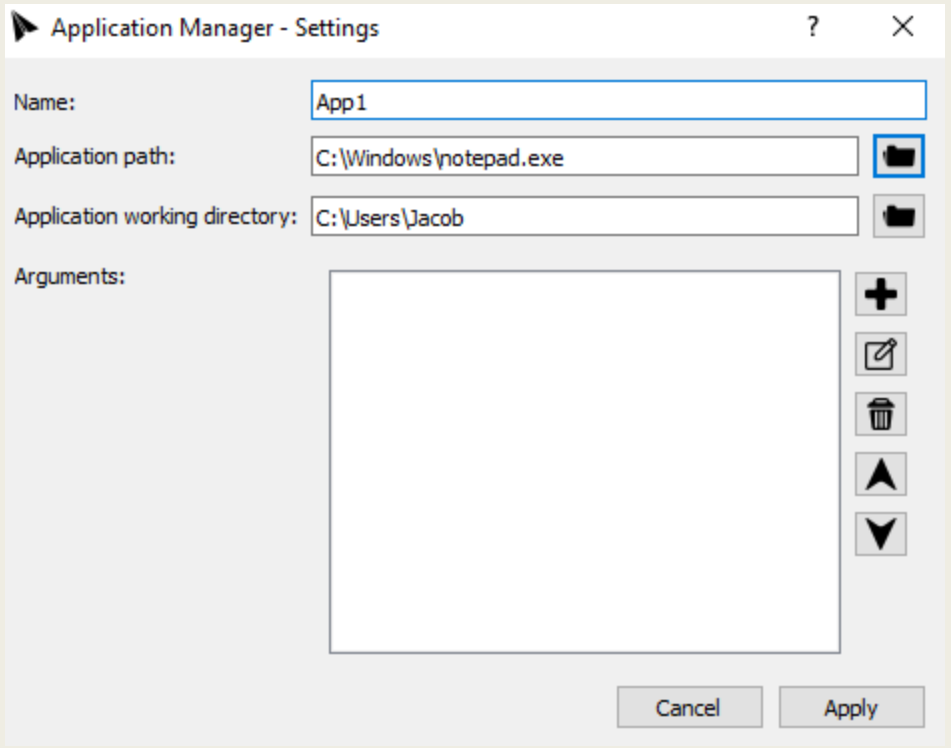
MENADŻER APLIKACJI DLA SYSTEMU LINUX

DIANA MIROWSKA

Przedmiotem prezentacji będzie:

- - struktura i plan projektu,
- - biblioteka PyQt5,
- - biblioteka json,
- - moduł os,
- - moduł subprocess.

Okienkowy menadżer uruchamiania skryptów i aplikacji dla systemu Linux



Dlaczego taki projekt?

- Ułatwienie codziennej pracy.
- Czytelniejsza diagnostyka systemu Linux.
- Przystępny interface.
- Rozwój znajomości języka Python.
- Zapoznanie z metodami serializacji danych.

Struktura projektu

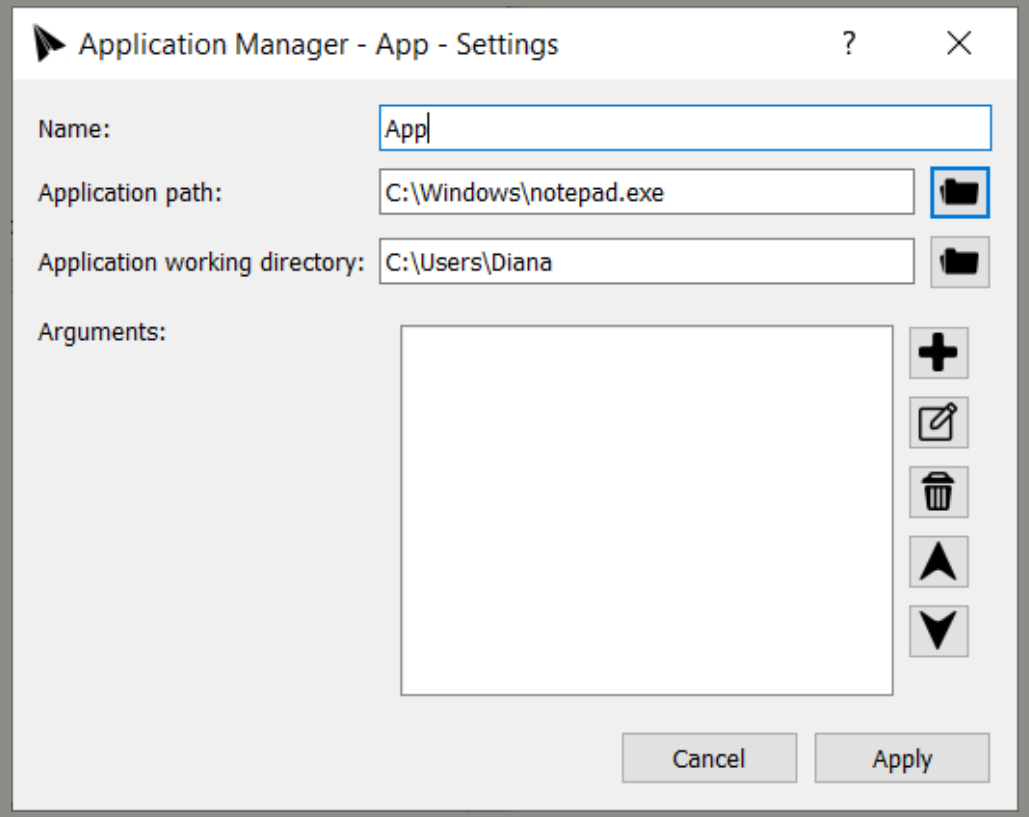
- Bazuje na klasach dziedziczących po QWidget (pośrednio lub bezpośrednio)
- Klasy okien zawierają w sobie widok (zbiór widżetów) oraz logikę aplikacji

Komponenty aplikacji:

- Okno główne z listą aplikacji oraz kontrolkami do zarządzania
- Okno konfiguracji/dodawania launchera aplikacji – konfiguracja nazwy, ścieżek oraz argumentów wykonania programu.
- Okno monitorowania procesów - wyniki programu *top*

PyQt5

- Port biblioteki Qt 5 dla języka Python
- Biblioteka wspierająca tworzenie aplikacji okienkowych.
- Opiera się o hierarchiczny model obiektowości rodzic-dzieci.
- Widżet jest podstawową klasą, z której dziedziczą wszystkie elementy GUI.
- Wspiera mechanizm sygnałów i slotów (funkcja *connect*)



QFormLayout

```
layout = QFormLayout()
self.setLayout(layout)

self.name_field_edit = QLineEdit()
self.name_field_edit.setText(app_data.name)

layout.addRow(QLabel("Name:"), self.name_field_edit)
```

QHBoxLayout

```
path_hbox = QHBoxLayout()
path_hbox.addWidget(self.path_field_edit)
path_hbox.addWidget(path_button)
```

QPushButton

```
self.path_field_edit = QLineEdit()
self.path_field_edit.setText(str(app_data.path))

path_button = QPushButton()
path_button.setIcon(QIcon('icons/folder.png'))
path_button.pressed.connect(self.showFilePath)
```

```
def showFilePath(self):

    file_path = QFileDialog.getOpenFileName(self, 'Open file', str(Path.home()))

    if file_path[0]:
        self.path_field_edit.setText(file_path[0])
```

- QPushButton – kontrolka przycisku.
- Ustawienie tekstu przycisku poprzez przekazanie go jako parametru konstruktora
- Ustawianie ikonki przycisku funkcją *setIcon()*
- Podpięcie funkcji *self.showFilePath* do zdarzenia kliknięcia

pyqtSignal

- Tworzenie własnego zdarzenia:

```
clicked = pyqtSignal(<lista typów>)
```

- Podpinanie funkcji do zdarzenia:

```
def clicked_event_handler(<parametry>):
    # kod obsługi zdarzenia
clicked.connect(clicked_event_handler)
```

- Wywoływanie zdarzenia:

```
clicked.emit(<parametry>)
```

Biblioteka JSON

JSON – uniwersalny, prosty format zapisu danych, przyjazny dla użytkownika.

Przechowywanie danych dotyczących aplikacji i parametrów uruchomieniowych.

Serializacja i deserializacja danych.

```python
class ConfigurationJSONEncoder(json.JSONEncoder):
    def default(self, o):
        if isinstance(o, ApplicationData):
            return o.as_dict()
        if isinstance(o, Path):
            return str(o)
        else:
            return json.JSONEncoder.default(self, o)
```

WŁASNY ENCODER JSON

Moduł OS

- Moduł umożliwiający interakcję z systemem operacyjnym
- Pozyskiwanie ścieżki dostępu.
- Sprawdzenie czy podana ścieżka istnieje.
- Sprawdzenie czy ścieżka wskazuje na plik bądź folder.
- Wskazanie domyślnej ścieżki uruchamiania aplikacji.

```
51      def run_clicked(self):
52          try:
53              subprocess.Popen(executable=str(self.app_data.path),
54                               args=self.app_data.arguments,
55                               cwd=self.app_data.work_dir,
56                               startupinfo=subprocess.STARTUPINFO(dwFlags=subprocess.DETACHED_PROCESS))
57          except (ValueError, OSError, subprocess.SubprocessError) as e:
58              QMessageBox.critical(self, "Error!", f"Program execution error: {str(e)}", QMessageBox.Ok)
```

MODUŁ SUBPROCESS

Biblioteka do uruchamiania, zarządzania i komunikacji z procesami.

Zadania

Zadanie 1:

- Dodaj nowe okno podobne do TopViewWindow zwracające wynik komendy `'cat /proc/meminfo'`
- Dodaj możliwosć otwarcia okna w menu *Tools*

Zadanie 2:

- Dodaj nowe pole 'description' do ApplicationData - uwzględnij pole w konstruktorze I funkcjach odpowiedzialnych za serializację: as_dict, from_dict
- Dodaj wyświetlanie pola 'description' w ApplicationWidget
- Dodaj możliwość edycji pola jako QLineEdit w ApplicationSettingsWidget
- Powyższe punkty można zrealizować analogicznie do pola *name*

- Dokumentacja PyQt5: https://doc.qt.io/qtforpython/api.html

DZIĘKUJĘ ZA UWAGĘ