UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY
a INFORMATIKY

BAKALÁŘSKÁ PRÁCE

2023 Adam Tomášek

Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Tvorba grafických aplikací v jazyce Python

Bakalářská práce

2023 Adam Tomášek

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2022/2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Adam Tomášek**
Osobní číslo:	**I20165**
Studijní program:	**B0688A140009 Informační technologie**
Téma práce:	**Tvorba grafických aplikací v jazyce Python**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování

Cílem bakalářské práce je vytvoření manuálu pro tvorbu grafických aplikací v jazyce Python. V teoretické části práce budou popsány dostupné grafické frameworky a jejich vzájemné porovnání. V praktické části práce budou vytvořeny jednoduché aplikace za použití popisovaných frameworků.

Rozsah pracovní zprávy: **min. 30 stran**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná**

Seznam doporučené literatury:

PECINOVSKÝ, Rudolf. *Začínáme programovat v jazyku Python*. 2. přepracované a rozšířené vydání. Praha: Grada Publishing, 2022. Začínáme s. ISBN 978-80-271-3609-4.

WILKES, Matthew. *Advanced Python development: using powerful language features in real-world applications*. [United States]: Apress, [2020]. ISBN 978-1-4842-5792-0.

SUMMERFIELD, Mark. *Python 3: výukový kurz*. 2. vydání. Přeložil Lukáš KREJČÍ. Brno: Computer Press, 2021. ISBN 978-80-251-5030-6.

Vedoucí bakalářské práce: **Ing. Martin Pozdílek, Ph.D.**
Katedra informačních technologií

Datum zadání bakalářské práce: **16. prosince 2022**
Termín odevzdání bakalářské práce: **12. května 2023**

L.S.

Ing. Zdeněk Němec, Ph.D. v.r.
děkan

Ing. Jan Panuš, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 28. února 2023

Prohlašuji:

Práci s názvem tvorba grafických aplikací v jazyce Python jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 7/2019 Pravidla pro odevzdávání, zveřejňování a formální úpravu závěrečných prací, ve znění pozdějších dodatků, bude práce zveřejněna prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 02. 05. 2023 Adam Tomášek

PODĚKOVÁNÍ

Rád bych poděkoval panu inženýru Miroslavu Dvořákovi a panu doktoru Martinu Pozdílkovi cenné rady a pomoc, kterou mi poskytli při tvorbě mé bakalářské práce.

ANOTACE

Tato bakalářská práce poskytuje porovnání čtyř grafických frameworků v Pythonu: PyQt5, Kivy, WxPython a Tkinter. Zahrnuje teoretické představení frameworků, jejich instalaci na Windows a tvorbu layoutu pro aplikace. Praktická část se zaměřuje na vývoj TODO listu a kalkulačky, demonstrujících funkčnost a možnosti každého frameworku. Práce je vhodná pro IT vývojáře a studenty.

KLÍČOVÁ SLOVA

Python, Kivy, PyQt5, WxPython, Tkinter

TITLE

Creating graphical applications in Python

ANNOTATION

This bachelor thesis provides a comparison of four Python graphical frameworks: PyQt5, Kivy, WxPython, and Tkinter. It includes a theoretical introduction to the frameworks, their installation on Windows, and the creation of layouts for applications. The practical part focuses on developing a TODO list and a calculator, demonstrating the functionality and capabilities of each framework. The work is suitable for IT developers and students.

KEYWORDS

Python, Kivy, PyQt5, WxPython, Tkinter

OBSAH

SEZNAM ILUSTRACÍ a TABULEK

SEZNAM ZKRATEK

AGPL	Affero General Public License
API	Application Programming Interface
BE	Back End
BSD	Berkeley Software Distribution
CSS	Cascading Style Sheets
ES	OpenGL for Embedded Systems
FE	Front End
FPS	Frame per second
GPL	General Public License
HEX	Hexadecimal
LGP	Lesser General Public License
MIT	Massachusetts Institute of Technology
NFC	Near Field Communication
OpenGL	Open Graphics Library
OS	Operating System
RGBA	Red Green Blue Alpha
SQL	Structured Query Language
SVG	Scalable Vector Graphics
TCL	Tool Command Language/Tool Kit
TK	Tool Kit UI User Interface
XML	Extensible Markup Language

ÚVOD

Tato bakalářská práce se zaměřuje na manuál a porovnání čtyř různých grafických frameworků v jazyce Python: PyQt5, Kivy, WxPython a Tkinter. V teoretické části práce jsou představeny jednotlivé frameworky a provedeno jejich porovnání z různých hledisek. Mezi faktory, které jsou při porovnání frameworků zohledněny, patří například výkonnost, komunita, licence a cena a kompatibilita s Pythonem 3. Nadále je zde ukázán jednoduchý manuál pro instalaci jednotlivých frameworků v operačním systému Windows a vytvoření jednoduchého layoutu pro aplikaci.

V praktické části práce jsou vytvořeny dvě aplikace – TODO list a kalkulačka.

TODO list slouží k jednoduché ukázce funkcionality každého frameworku, stylování a ovládání samotné aplikace a porovnání paměťových a výkonnostních nároků pro každý framework. Kalkulačka poté pro hlubší prozkoumání kódu a porovnání vlastností a postupů mezi frameworky.

1. TEORETICKÁ ČÁST

1.1.Dostupné grafické frameworky

1.1.1. PyQt5

Qt je sada knihoven a vývojových nástrojů v jazyce C++, která obsahuje platformě nezávislé abstrakce pro grafická uživatelská rozhraní, sítě, vlákna, regulární výrazy, databáze SQL, SVG, OpenGL, XML, uživatelská a aplikační nastavení, polohové a lokalizační služby, komunikaci na krátkou vzdálenost (NFC a Bluetooth), prohlížení webu, 3D animace, grafy, 3 D vizualizaci dat a propojení s obchody s aplikacemi. PyQt5 implementuje více než 1000 těchto tříd jako sadu modulů Pythonu.

PyQt5 se skládá ze samotného PyQt5 a řady doplňků, které odpovídají dalším knihovnám Qt. Každý z nich je k dispozici jako zdrojová distribuce *sdist* a předkompilovaný soubor, který obsahuje spustitelný kód a další potřebné soubory pro instalaci a spuštění na operačních systémech Windows, Linux a MacOS.

PyQt5 podporuje platformy Windows, Linux, UNIX, Android, macOS a iOS a vyžaduje Python ve verzi 3.5 nebo novější. Pro starší verze Pythonu v3 a pro verzi Pythonu v2.7 je zde stále starší alternativa PyQt4. [1]

1.1.2. Tkinter

Balíček Tkinter (rozhraní Tk) je standardní rozhraní jazyka Python pro sadu grafických nástrojů Tcl/Tk. Balíky Tk i Tkinter jsou k dispozici na většině unixových platforem včetně systému MacOS a také v systémech Windows.

Spuštění příkazu *python -m* Tkinter z příkazového řádku otevře okno demonstrující jednoduché rozhraní *Tk*, které vás informuje o tom, že Tkinter je ve vašem systému správně nainstalován, a také ukáže, jaká verze Tcl/Tk je nainstalována, takže si můžete přečíst dokumentaci Tcl/Tk specifickou pro tuto verzi.

Tkinter podporuje řadu verzí Tcl/Tk, které jsou sestaveny buď s podporou vláken, nebo bez ní. Oficiální binární verze Pythonu obsahuje Tcl/Tk 8.6 s vlákny. Další informace o podporovaných verzích najdete ve zdrojovém kódu modulu *_thinker*.

Tkinter není tenký *wrapper*, ale přidává poměrně hodně vlastní logiky, aby bylo prostředí přizpůsobenější programovacím metodám v jazyce Python. Tato dokumentace se soustředí na

tyto doplňky a změny a v podrobnostech, které se nezměnily, odkazuje na oficiální dokumentaci Tcl/Tk. [2]

1.1.3. Kivy

Kivy je open-source multiplatformní knihovna pro vývoj grafického uživatelského rozhraní v jazyce Python, kterou lze spustit v systémech iOS, Android, Windows, OS X a GNU/Linux. Pomáhá vyvíjet aplikace, které využívají inovativní více dotykové uživatelské rozhraní. Základní myšlenkou Kivy je umožnit vývojáři vytvořit aplikaci jednou a používat ji na všech zařízeních, díky čemuž je kód opakovaně použitelný a nasaditelný, což umožňuje rychlý a snadný návrh interakce a rychlé prototypování. [3]

1.1.4. WxPython

WxPython je multiplatformní sada nástrojů grafického uživatelského rozhraní pro programovací jazyk Python. Umožňuje programátorům v jazyce Python jednoduše a snadno vytvářet programy s robustním a vysoce funkčním grafickým uživatelským rozhraním. Je implementován jako sada rozšiřujících modulů jazyka Python, které obalují komponenty grafického uživatelského rozhraní populární multiplatformní knihovny WxWidgets, která je napsána v jazyce C++.

Stejně jako Python a WxWidgets je i WxPython Open Source, což znamená, že je zdarma k použití pro kohokoli a zdrojový kód je k dispozici komukoli k nahlédnutí a úpravám. a kdokoli může do projektu přispívat opravami nebo vylepšeními.

WxPython je multiplatformní sada nástrojů. To znamená, že stejný program poběží na více platformách bez nutnosti úprav. v současné době jsou podporovány tyto platformy: Microsoft Windows, Mac OS X a MacOS a Linux nebo jiné Unixům podobné systémy s knihovnami GTK2 nebo GTK3. Ve většině případů se na každé platformě používají nativní widgety, které zajišťují 100% nativní vzhled aplikace.

Vzhledem k jednoduchosti jazyka Python, jsou programy WxPython jednoduché, snadno se píšou a jsou srozumitelné. [4]

1.2. Porovnání frameworků

V této části práci se lze dočíst o porovnání frameworků z hlediska komunity, licence a kompatibility. Další porovnání jako je funkcionalita, výkonnost a paměťové požadavky jsou v praktické části.

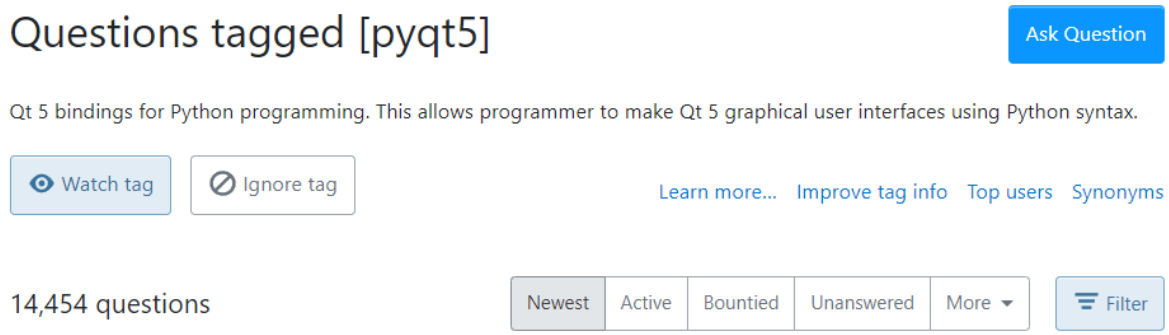
1.2.1. Komunita

Komunity okolo jednotlivých frameworků není možné kvantifikovat. Existuje příliš mnoho stránek a možností vyhledávání na to, aby výsledek byl u všech frameworků konzistentní. Rozhodl jsem se tedy zaměřit se spíše na frameworky v jednotlivých komunitách.

Stackoverflow.com

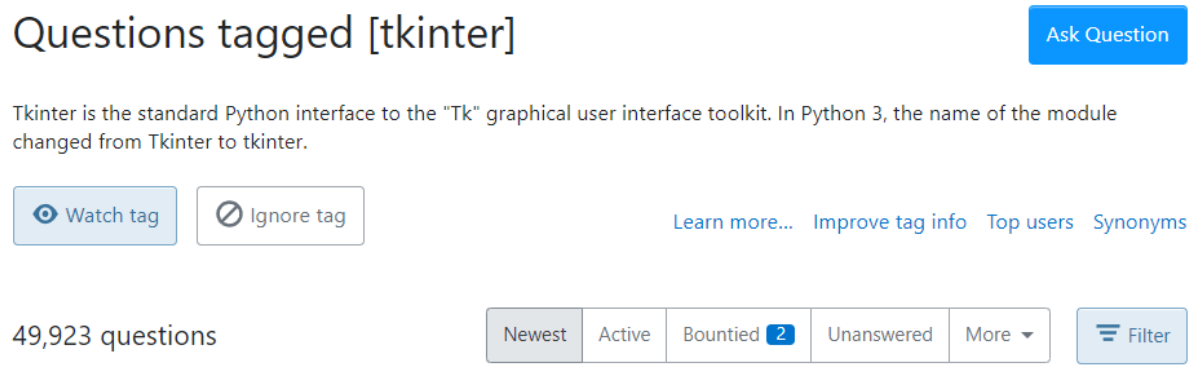
Stack Overflow jako komunita umožňuje vyhledávání podle tagu, k určitému dni (viz. citace obrázků) jsem provedl vyhledávání podle charakteristických tagů pro jednotlivé frameworky.

PyQt5



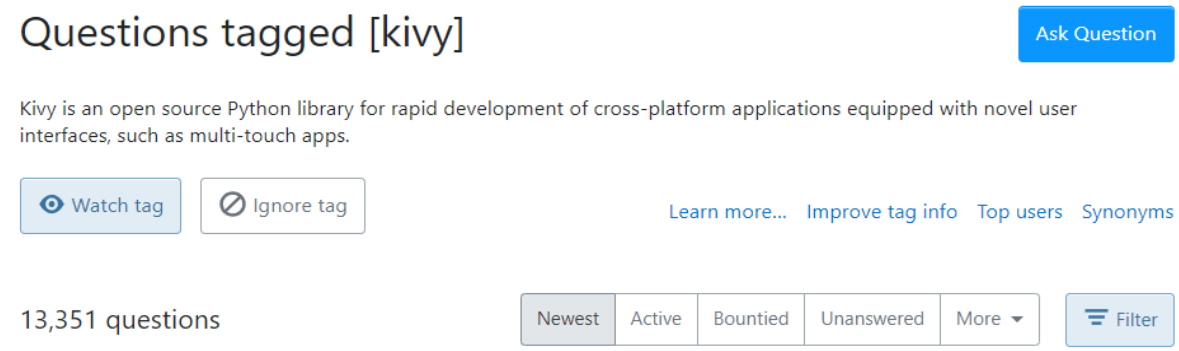
Obrázek 1 PyQt5 TAG – Stack Overflow [5]

Tkinter



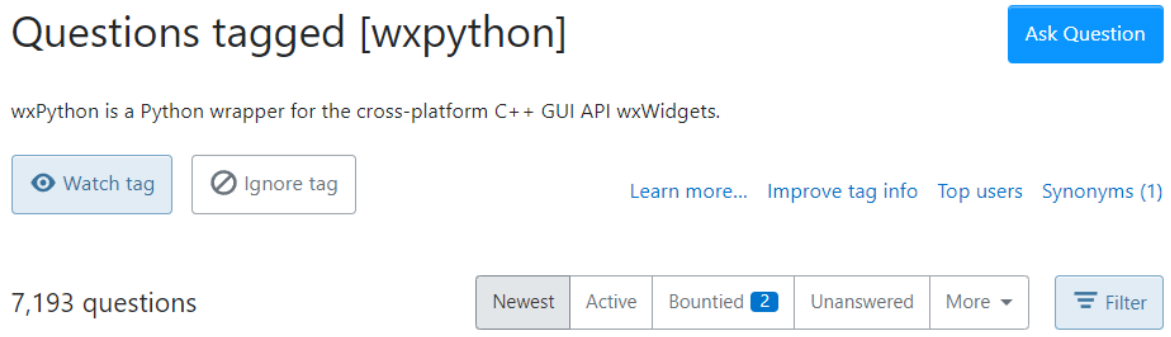
Obrázek 2 Tkinter TAG – Stack Overflow [6]

Kivy



Obrázek 3 Kivy TAG – Stack Overflow [7]

WxPython



Obrázek 4 WxPython TAG – Stack Overflow [8]

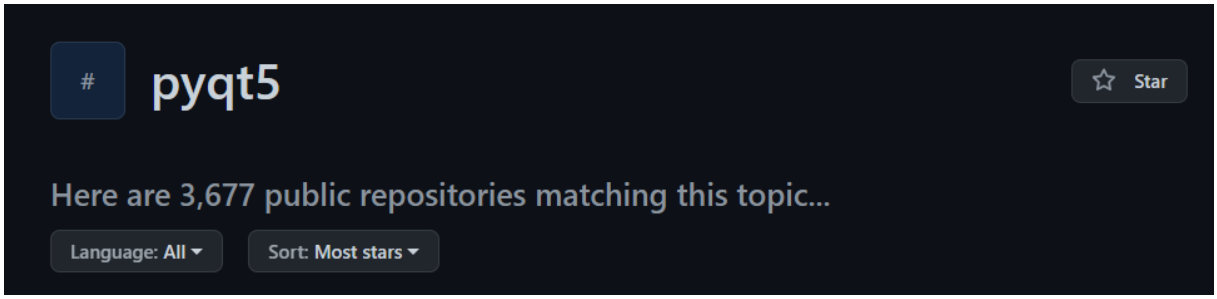
Shrnutí

Pokud jde o pokrytí na Stack Overflow, podle mé analýzy má Tkinter nejvíce otázek (přes 14 000), následovaný PyQT5 (přes 14 000), Kivy (přes 13 000) a WxPythonem (přes 7 000). To by mohlo naznačovat, že Tkinter je nejpopulárnější a nejpoužívanější knihovna GUI pro Python, ale také že má nejvíce problémů nebo nejasností. Tkinter je také velmi oblíbený a snadno použitelný pro jednoduché projekty. WxPython a Kivy a PyQt5 jsou méně časté, ale stále mají dostatek zdrojů a podpory.

Github.com

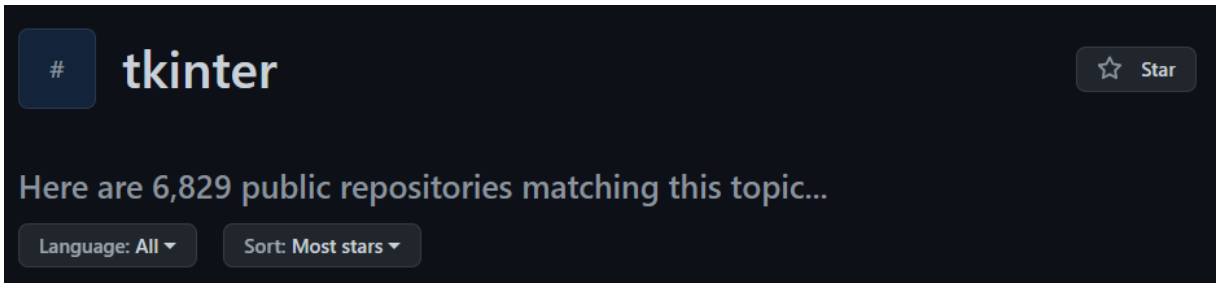
Obdobně jako u „stackoveflow.com“ se i komunita na stránce „github.com“ dá kvantifikovat za pomocí tagů. v tomto případě využiji k měření počet repositářů, které obsahují tag pro jednotlivé frameworky.

PyQt5



Obrázek 5 PyQt5 TAG – Github [9]

Tkinter

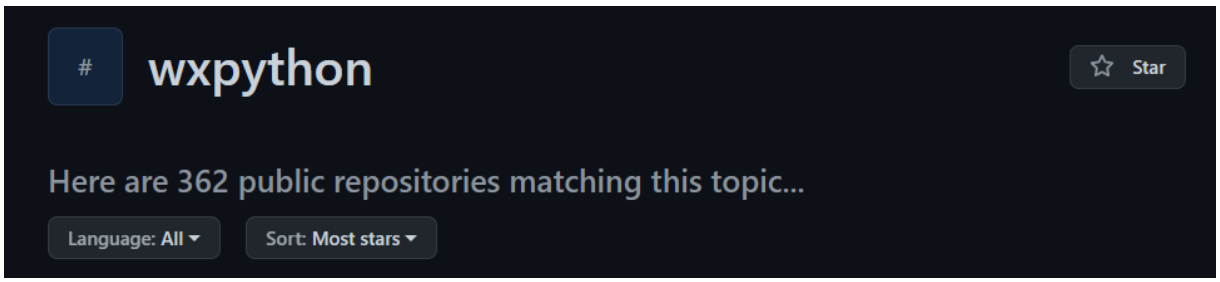


Obrázek 6 Tkinter TAG – Github [10]

Kivy

Obrázek 7 Kivy TAG – Github [11]

WxPython



Obrázek 8 WxPython TAG – Github [12]

Shrnutí

Tkinter je nejpopulárnější s téměř dvojnásobným počtem repozitářů (6 829) ve srovnání s PyQt5, který má 3 677 repozitářů. Kivy, i když je zaměřený na inovativní multi-touch rozhraní, má významně méně repozitářů (1,198), což naznačuje menší, ale specifickou komunitu vývojářů. WxPython je s 362 repozitáři nejméně zastoupeným frameworkem, což může odrážet jeho specifičtější použití nebo preferenci vývojářů pro jiné nástroje. Celkově tyto čísla ukazují, že komunita vývojářů Pythonu preferuje Tkinter pro jeho jednoduchost a dostupnost, zatímco PyQt a Kivy jsou vyhledávané pro jejich rozšířené možnosti a podporu pro sofistikovanější aplikace.

1.2.2. Licence a cena

MIT licence

Licence MIT je jedna z nejpopulárnější licencí v IT v době psaní této semestrální práce. Mezi nejpopulárnější software, který používá tuto licenci je vedle zde zmíněného Kivy i Ruby on Rails, jQuery a Node.js.

Licence patří do mezi pasivní open source licence, což podobně jako u ostatních pasivních licencí znamená, že má pouze velmi málo omezení. Velký rozdílem od obdobných licencí např. (GPL v3 a AGPL) je, že nevyžaduje, aby veškerá vybudovaná díla založená na komponentech s touto licencí používala jí i zachovala. Tedy dílo založené na AGPL musí být taktéž pod licencí AGPL. Celé znění licence má pouze 172 slov. Mezi další velmi oblíbené využívané vedle MIT licence jsou BSD a Apache License 2.0. [14]

Každý, kdo získá kopii souvisejícího softwaru, má právo software využívat, kopírovat, upravovat, slučovat, distribuovat, publikovat, poskytovat a prodávat kopie softwaru. Jediné omezení pro použití softwaru spočívá v tom, že na jakákoliv díla vyplývající z komponent, která používají tuto licenci musí být dáno stejné upozornění na autorská práva. [15]

GPL licence

GPL licence je jedna z prvních volných licencí (první vydání bylo již v 25.2.1989 [17]), která umožňuje volné využívání softwaru, volné měnění softwaru, volné sdílení softwaru a volnou změnu softwaru. Vlastností je samozřejmě mnohem více, ale tyto 4 jsou podle oficiálních zdrojů ty nejzákladnější.

Jedinou výraznější podmínkou je, že v případě použití softwaru pod touto licencí k jakékoliv tvorbě, je povinností, aby byl vzniklý produkt taktéž pod licencí GPL. [18]

LGPL licence

LGPL licence je velmi podobná GPL licenci. „L“ v „L“GPL znamená „Lesser“. Vznikla jako reakce na svazující GPL licenci. Autoři softwaru, měli problém s bezpodmínečnou „dědičností“ GPL licence.

Ve zkratce, hlavním rozdílem mezi GPL licencí je za určitých podmínek možnost šíření softwaru s pouhými linky na licence původních projektů bez nutnosti zdědění GPL licence. [20]

Kivy

Kivy je 100 % pod licencí MIT [13]

PyQt5

PyQt5 je ze základu pod GPL, nebo pod LGPL licencí. Sami autoři však doporučují GPL licenci vhledem k větší rozmanitosti některých komponentů.

Nicméně nelze použít ani jednu z těchto licencí, je zde stále možnost PyQt komerční licence. [16]

Tkinter

Tkinter má podobně jako WxPython svou vlastní licenci založenou na open source. Na rozdíl od předchůdce, se již neodráží o LGPL licenci, nicméně si rozšíření specifikuje sám. [21]

WxPython

WxPython je pod licencí „knihovny WxWindows“.

Podle autora to však znamená, že základní licencí je LGPL. Pokud jsou však díla v binární formě, můžou být distribuovány dle vlastních podmínek. [19]

Shrnutí

MIT licence je nejlepší volbou pro softwarové frameworky kvůli její flexibilitě a minimálním omezením. Umožňuje široké využití softwaru, včetně komerčního, bez striktních pravidel o dědičnosti licence, což je výhodné ve srovnání s GPL a LGPL licencemi. Tato licence je ideální pro různé projekty, jak ukazuje její využití v populárních frameworkách jako Kivy.

1.2.3. Kompatibilita s Pythonem v3

Kivy

Kivy je kompatibilní s Pythonem v3. Kivy nabízí také možnost využít přidanou technologii OpelGL ES 2 pro hardwarově akcelerované vykreslování. [22]

Co se týče funkčnosti s verzí Pythonu 2.7, Kivy nabízí řešení v podobě úpravy v kód. [23]

WxPython

WxPython je od verze 4 která byla vydána v roce 2018 plně kompatibilní s Pythonem v3. [24]

Kromě verze Python v3 nabízí plnou kompatibilitu i se staršími verzemi pythonu na kterých byl tento framework postaven již od základu. [25]

PyQt5

PyQt5 podle oficiální dokumentace nabízí podporu pouze pro Python v3. [26]

Toto omezení se však týká pouze verze 5. PyQt4 jakožto knihovna postavená na Qt4 má plnou kompatibilitu jak s Pythonem v3, tak i s Pythonem v2. [27]

Tkinter

Tkinter je kompatibilní s Pythonem v3 i Pythonem v2.

Při migraci aplikace naprogramované v Pythonu v2 na Python v3 budou však potřebné určité změny, jelikož framework není v jednotlivých verzích naprosto totožný. [28]

Shrnutí

Kivy je kompatibilní s Pythonem v3 a nabízí podporu pro OpenGL ES 2, zatímco pro Python 2.7 vyžaduje úpravy kódu. WxPython je od verze 4 (2018) plně kompatibilní s Pythonem v3, přičemž zachovává kompatibilitu se staršími verzemi Pythonu. PyQt5 podporuje výhradně Python v3, ale jeho předchůdce PyQt4 je kompatibilní jak s Pythonem v3, tak v2. Tkinter funguje s oběma verzemi Pythonu, ale při migraci z Pythonu v2 na v3 mohou být potřeba změny v kódu.

2. PRAKTICKÁ ČÁST

2.1.Manuál

2.1.1. PyQt5

Instalace

```
pip install pyqt5
```

Tento příkaz instaluje PyQt5, což je sada Python modulů pro vývoj grafických uživatelských rozhraní (GUI). Tento příkaz je nutná zadat v příkazovém řádku.

Import potřebných modulů frameworku.

```
import sys

from PyQt5.QtWidgets import QApplication, QLabel, QWidget
```

import sys: Importuje modul sys, který je součástí standardní knihovny Pythonu a poskytuje přístup k některým proměnným a funkcím, které silně interagují s interpretem Pythonu.

from PyQt5.QtWidgets import QApplication, QLabel, QWidget: Importuje třídy QApplication, QLabel a QWidget z modulu QtWidgets. QApplication spravuje běh a konfiguraci aplikace, QLabel se používá pro zobrazení textu a QWidget je základní třída pro všechny objekty uživatelského rozhraní.

Inicializace

```
app = QApplication(sys.argv)
```

Vytváří instanci QApplication. Tato třída spravuje celkové chování aplikace a je potřebná pro všechny aplikace PyQt. ***sys.argv*** je seznam argumentů příkazové řádky.

Vytvoření hlavního okna aplikace a nastavení jeho vlastností

```
window = QWidget()

window.setWindowTitle('Hello World - PyQt5')

window.setGeometry(100, 100, 200, 50)
```

window = QWidget(): Vytvoří hlavní okno aplikace. QWidget je základní třída pro všechny objekty uživatelského rozhraní.

window.setWindowTitle('Hello World - PyQt5'): Nastavuje titulek okna.

window.setGeometry(100, 100, 200, 50): Určuje polohu a velikost okna (x, y, šířka, výška).

Přidání labelu s textem

```
label = QLabel('Hello, World!', parent=window)

label.move(50, 15)
```

label = QLabel('Hello, World!', parent=window): Vytvoří nový label (nápis) s textem "Hello, World!" a přidá ho do hlavního okna.

label.move(50, 15): Nastavuje pozici labelu v rámci okna.

Zobrazení okna a spuštění aplikace

```
window.show()

sys.exit(app.exec_())
```

window.show(): Zobrazí hlavní okno.

sys.exit(app.exec_()): Spouští hlavní událostní smyčku aplikace a ukončí program s návratovým kódem, když je okno zavřeno. ***app.exec_()*** zpracovává události a vrací kontrolu, až když je aplikace ukončena.

2.1.2. Tkinter

Instalace

Tkinter je součástí standardní knihovny Pythonu a nemusí se instalovat.

Import potřebných modulů frameworku.

```
import tkinter as tk
```

import tkinter as tk: Importuje modul tkinter a přejmenovává jej na tk pro snazší přístup. Tkinter je používán pro vytváření grafických uživatelských rozhraní (GUI) v Pythonu.

Vytvoření hlavního okna aplikace a nastavení jeho vlastností

```
window = tk.Tk()

window.title("Hello World - Tkinter")
```

window = tk.Tk(): Vytváří hlavní okno aplikace. Třída Tk je základní třídou v Tkinteru pro vytváření oken.

window.title("Hello World - Tkinter"): Nastavuje titulek hlavního okna.

Přidání labelu s textem

```
greeting = tk.Label(text="Hello, World!")

greeting.pack()
```

greeting = tk.Label(text="Hello, World!"): Vytváří widget Label (nápis) s textem "Hello, World!".

greeting.pack(): Přidává widget *Label* do okna a spravuje jeho umístění. Metoda *pack()* je jednou z několika metod geometrického manažera v Tkinteru, která kontroluje, jak jsou widgety umístěny v okně.

Zobrazení okna a spuštění aplikace

```
window.mainloop()
```

window.mainloop(): Spouští hlavní událostní smyčku aplikace. Tato metoda čeká na události, jako jsou kliknutí myší nebo stisky kláves, a zpracovává je. Udržuje tak aplikaci běžící a reagující na uživatelský vstup, dokud okno není zavřeno.

2.1.3. Kivy

Instalace

```
pip install kivy
```

Import potřebných modulů frameworku

```
from kivy.app import App

from kivy.uix.label import Label
```

from kivy.app import App: Importuje třídu App z modulu kivy.app. Tato třída slouží jako základ pro tvorbu aplikací v Kivy.

from kivy.uix.label import Label: Importuje třídu Label z modulu kivy.uix.label. Label je widget používaný pro zobrazení textu.

Definice třídy aplikace

```
class HelloWorldApp(App):

    def build(self):

        return Label(text='Hello, World!')
```

Tato část kódu definuje třídu ***HelloWorldApp***, která dědí z třídy ***App*** poskytované Kivy.

Metoda ***build*** je přepsána tak, aby vracela instanci Label s textem "Hello, World!". Tato metoda je volána automaticky, když Kivy spustí aplikaci, a má za úkol nastavit počáteční stav uživatelského rozhraní aplikace.

Zobrazení okna a spuštění aplikace

```
if __name__ == '__main__':
    HelloWorldApp().run()
```

Tento kód ověří, zda je skript spuštěn jako hlavní program (a ne jako modul v jiném skriptu) pomocí ***if __name__ == '__main__':***.

Pokud ano, vytvoří instanci třídy ***HelloWorldApp*** a spustí ji pomocí metody ***run***. Tato metoda spouští aplikaci a zahajuje její událostní smyčku, což umožňuje zpracovávat události, jako jsou doteky nebo kliknutí.

2.1.4. wxPython

Instalace

```
pip install wxPython
```

Import potřebných modulů frameworku

```
import wx
```

import wx: Importuje wxPython, což je modul, který poskytuje nástroje pro vytváření grafického uživatelského rozhraní.

Vytvoření hlavního okna aplikace a nastavení jeho vlastností

```
app = wx.App(False)
frame = wx.Frame(None, wx.ID_ANY, "Hello World - wxPython")
```

app = wx.App(False): Vytváří instanci aplikace wxPython. Argument False říká, že wxPython nemá přesměrovávat stdout a stderr do okna.

frame = wx.Frame(None, wx.ID_ANY, "Hello World - wxPython"): Vytváří hlavní okno aplikace. ***wx.Frame*** je základní třída pro vytváření oken. První argument (***None***) určuje, že okno nemá nadřazený objekt, ***wx.ID_ANY*** povoluje wxPythonu automaticky vygenerovat identifikátor, a poslední argument je titulek okna.

Přidání labelu s textem

```
panel = wx.Panel(frame)

text = wx.StaticText(panel, label="Hello, World!", pos=(40,40))
```

panel = wx.Panel(frame): Vytváří panel uvnitř hlavního okna. Panel je kontejner, který může obsahovat další widgety.

text = wx.StaticText(panel, label="Hello, World!", pos=(40,40)): Vytváří statický text (label) s určeným textem a pozicí uvnitř panelu.

Zobrazení okna a spuštění aplikace

```
frame.Show(True)

app.MainLoop()
```

frame.Show(True): Zobrazuje hlavní okno.

app.MainLoop(): Spouští hlavní smyčku aplikace, která zpracovává události, jako jsou kliknutí myši nebo stisky kláves, a udržuje aplikaci běžící až do jejího ukončení.

2.2. TODO list

2.2.1. Úvod

Tato část bakalářské práce je věnována aplikaci TODO listu. Jedná se o pouhou ukázku možností jednotlivých frameworků, rozšířené informace a vnitřních fungováních a postupech realizací je rozvinuta v části Kalkulátor. TODO list se věnuje spíše ukázce funkcionality, výkonnosti a paměťových nároků.

2.2.2. Struktura a responzivita

Když pomineme vyskakovací modální okna, celou aplikaci tvoří pouze jediné okno. Hlavní kontejner v okně obsahuje 2 druhy prvků – tlačítka a *listView*. *Listview* slouží pro ukládání, označování a v jednom případě i manipulaci s množinou úkolů. Tlačítka dále tvoří ovládací prvky aplikace.

2.2.3. Styly

V případě frameworku **PyQt5** byla možnost využít CSS stylování. Tedy na rozdíl od ostatních implementací je zde umožněno využít technologii strukturovaného stromu která je známá z vývoje webu – tedy identifikace komponenty a přidání požadovaných vzhledových vlastností. Takové řešení umožnilo výrazně propracovanější a příjemnější vzhled aplikace, než tomu je u ostatních frameworků.

V případě tohoto frameworku také stálo za úvahu použití Qt designeru – softwaru pro grafické modelování layoutů, nicméně kvůli problémům s responzivitou bylo nakonec zvoleno CSS řešení.

```
QPushButton {
    background-color: #e7e7e7;
    color: #3498db;
    padding: 12px 20px;
    border: 1px solid #bdc3c7;
    font-size: 14px;
    font-weight: bold;
    border-radius: 8px;
    cursor: pointer;
}
```

Obrázek 9 PyQt5 – TODO – styly

Řešení podobné CSS stylům šlo využít ještě u frameworku **Kivy**. Podobně jako PyQt5 lze stylování nastavit kaskádovým jazykem, nicméně zde je tomuto účelu použit vlastní jazyk *Kv language*.

```
Label:
    adaptive_height: True
    text: root.text
    font_size: '17dp'
    color: '#42cbf5'
    size_hint_y: None
    pos_hint: {'center_y': 0.5}
    bold: True
```

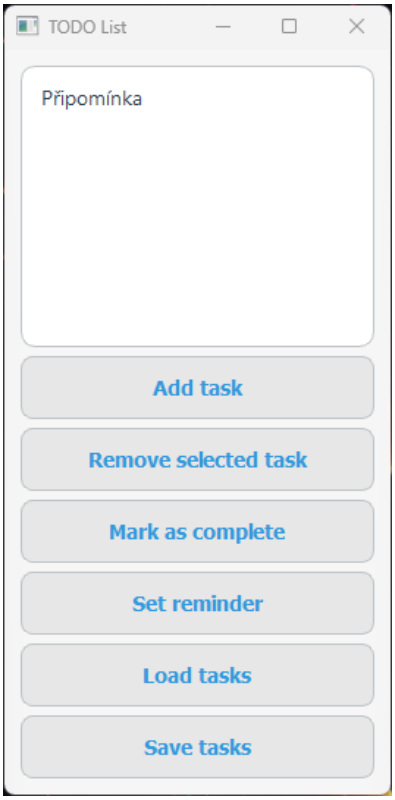
Obrázek 10 Kivy – TODO – styly

Bohužel u ostatních frameworků neexistuje řešení textového stylování, nebo externího softwaru. Každý styl musí být přímo nastaven jednotlivým komponentám.

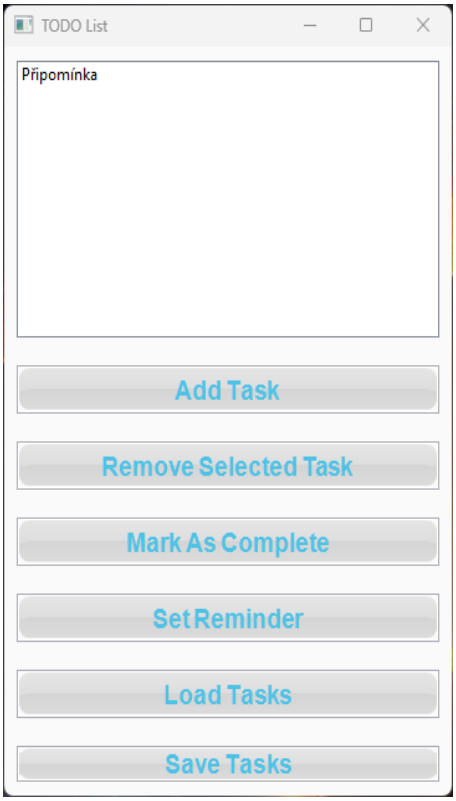
```
btnDone = Button(self.root, text='Mark as complete', width=2, font=font, highlightthickness=0,
                 background=self.bg_color, foreground=self.fg_color, highlightbackground='black',
                 command=lambda: self.markAsDone())
btnDone.pack(fill=X, padx=20, pady=10)

btnNotify = Button(self.root, text='Set reminder', width=2, font=font, highlightthickness=0,
                   background=self.bg_color, foreground=self.fg_color, highlightbackground='black',
                   command=lambda: self.setNotification())
btnNotify.pack(fill=X, padx=20, pady=10)
```

Obrázek 11 WxPython – TODO – styly

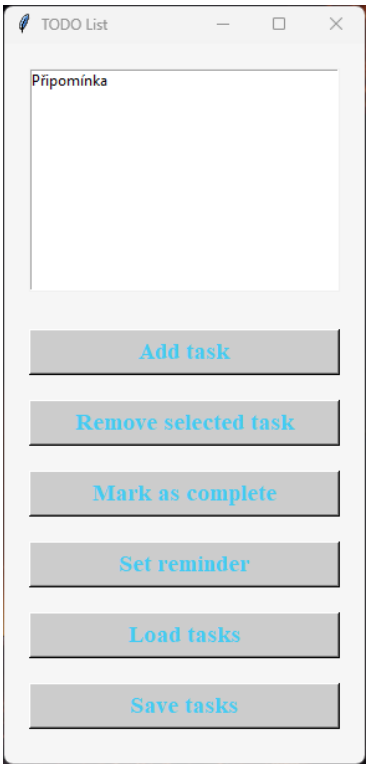
2.2.4. Vzhled



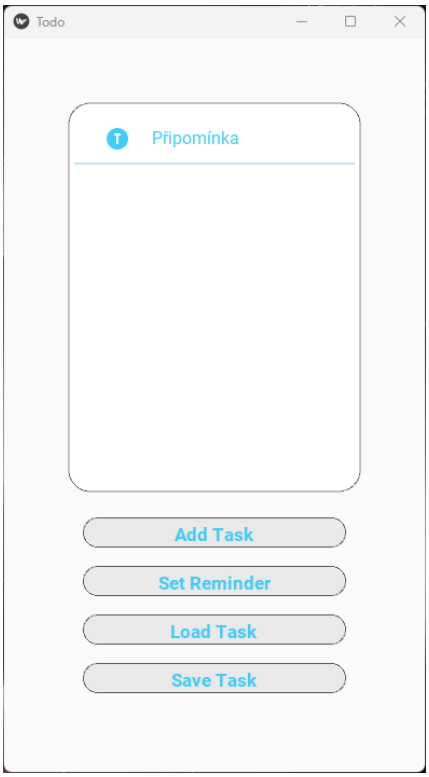
Obrázek 14 PyQt5 – TODO – Aplikace



Obrázek 15 Tkinter – TODO – Aplikace



Obrázek 12 Tkinter – TODO – Aplikace

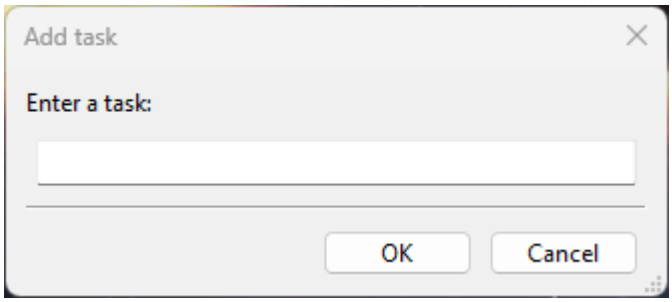


Obrázek 13 Kivy – TODO – Aplikace

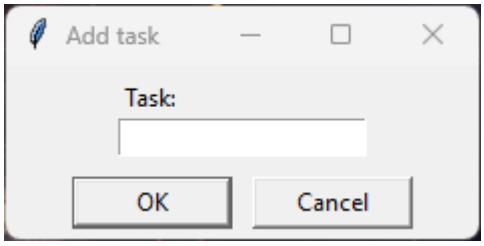
2.2.5. Funkcionalita tlačítek

Tlačítko Add task

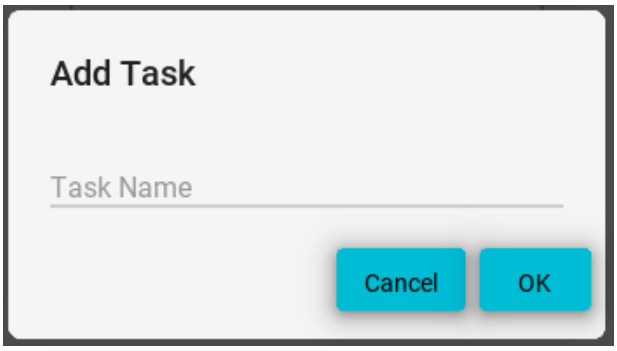
Jednoduché tlačítko, po jehož zmáčknutí se objeví modální okno s textovým polem pro zadání názvu úkolu.



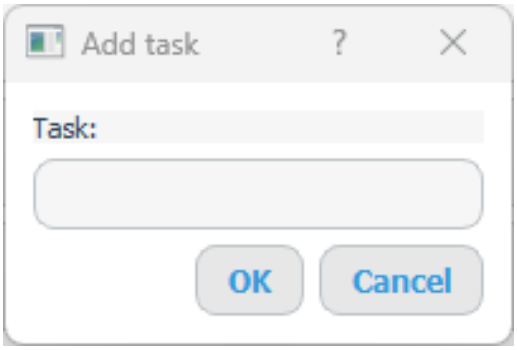
Obrázek 19 WxPython – TODO – Add task window



Obrázek 17 Tkinter – TODO – Add task window



Obrázek 16 Kivy – TODO – Add task window



Obrázek 18 PyQt5 – TODO – Add task window

U všech frameworků s výjimkou Kivy se jedná o vyskakovací okna oddělená od hlavní aplikace. V případě Kivy Windows neregistruje nové okno a požadavek se ukazuje v rámci hlavní aplikace.

Po kliknutá na tlačítko „OK“ se volba potvrdí a okno se zavře, při kliknutí na tlačítko „Cancel“ se žádná ze změn neaplikuje.

Tlačítko Remove selected task

Toto tlačítko obsahují všechny implementace s výjimkou Kivy. Před kliknutím je třeba vybrat úkol ze seznamu. Pokud je seznam prázdný, neprovedou se žádné změny. Po stisknutí nelze vrátit změny, ani samotná akce není doprovázena žádným varovným upozorněním.

Implementace Kivy je výrazně rozdílná od ostatních frameworků. Jelikož Kivy je uzpůsoben i pro dotykové obrazovky, byla tato aplikace zvolena pro ukázku této funkčnosti.

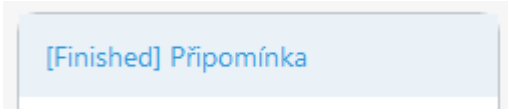
Při použití aplikace na PC, je třeba zvolený úkol zakliknout a potáhnout vpravo. Díky animaci se zobrazí ikonka koše a po puštění myši, se samotný úkol z listu vymaže.

Obrázek 20 Kivy – TODO – Remove task

Tlačítko Mark as complete

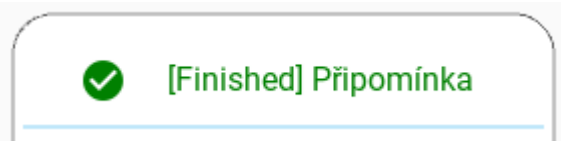
V rámci funkcionality je tato akce přirovnatelná k tlačítku „Remove selected task“. Obdobně jako u tohoto příkladu je bez návratová, nenásleduje jí žádné modální okno s upozorněním a implementace v Kivy je opět trochu odlišná.

Akce vyžaduje výběr úkolu ze seznamu a kliknutí na tlačítko. Výsledkem je přidání textu „[Finished]“ k vybranému úkolu.



Obrázek 21 PyQt5 – TODO – Mark as complete window

Zástupcem tlačítka pro framework Kivy je akce při kliknutí na ikonku „T“ u jednotlivých úkolů. Změní se jak ikona, tak barva textu.



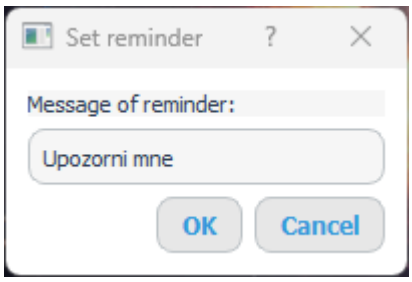
Obrázek 22 Kivy – TODO – Mark as complete window

Tlačítko Set reminder

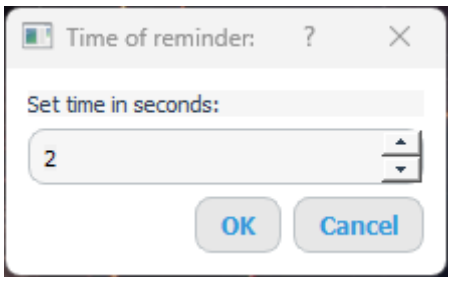
Tlačítko „Set reminder“ má ve všech frameworcích stejnou implementaci, s lehce rozdílným zadáním. Po stisknutí tlačítka, má uživatel možnost zadat text upozornění a čas v sekundách, kdy má upozornění vyskočit.

Odlišné implementace spočívají pouze v počtu vyskakovacích oken, které umožňují zadání.

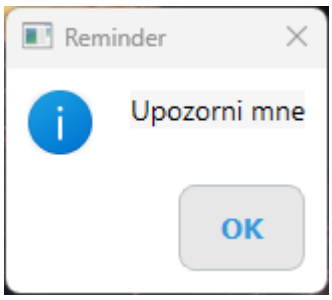
V případě PyQt5 proces zadávání tvoří 2 okna, jedno pro text upozornění a druhé pro číselné zadání počtu sekund.



Obrázek 23 PyQt5 – TODO – Reminder 1. window



Obrázek 24 PyQt5 – TODO – Reminder 2. window



Obrázek 25 PyQt5 – TODO – Reminder alert

V případě WxPythonu a Tkinteru je řešení obdobné, v druhém vyskakovacím okně lze ovšem s chybou zadat i jiný text než pouze čísla.

Jediným rozdílem je aplikace Kivy, kde je zadání jak textu, tak času tvořeno pouze jedním oknem.

Tlačítko Load tasks a Save tasks

Poslední dvě tlačítka v layoutu se starají o perzistenci úkolů. Po kliknutí se zobrazí systémové okno pro načtení, případně uložení.

2.2.6. Paměťové nároky

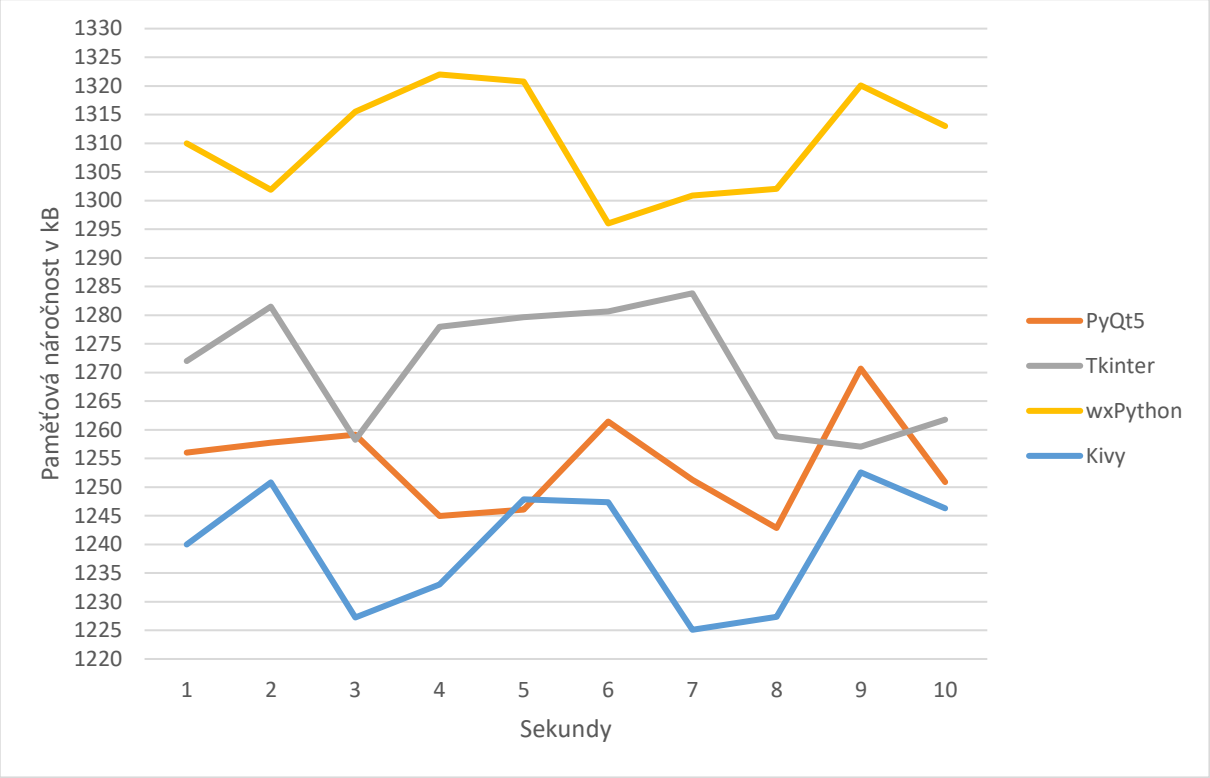
Nejjednodušším postupem, jak změřit paměťové nároky pro jednotlivé frameworky bylo vytvoření jednoduchého scriptu, který by automaticky simuloval aktivitu uživatele. Důvodem, proč je nutné tento postup automatizovat je nekonzistence zadávání – prodlevy mezi jednotlivými úkony by mohly negativně ovlivnit výsledky měření.

Zvolený časový rámec byl 10 sekund při kterých script nahodile (avšak pro všechny frameworky stejně) **přidával úkoly, odebíral úkoly a označoval je jako dokončené**. Tato aktivita byla následně zopakována v 10 iteracích.

Bohužel, odebírání dat by bylo složitější pro automatizaci. Řešením byl záznam správce úloh společně s logy testovacích scriptu, který umožnil odebírat paměťovou náročnost pro každý framework v přesně daném intervalu s odchylkou nižších stovek milisekund.

Data byla sesbírána, jednotlivé iterace zprůměrovány po sekundách a zaneseny a grafu.

Výsledky:



Obrázek 26 TODO – porovnání paměti

Vyhodnocení a shrnutí

Výsledky měření napovídají, že paměťová náročnost se v případě aplikace TODO liší pouze v desítkách kilobajtů. V měření se neprojevila složitost jednotlivých frameworků – dost pravděpodobně vzhledem k malé složitosti aplikace TODO, avšak v 9 sekundě (kdy nastávalo odebírání úkolů ze seznamu) je jasně vidět nárůst u frameworků Kivy, wxPython a PyQt5. Tento výsledek naznačuje, že na rozdíl od frameworku Tkinter je u zmíněných frameworků paměťově mnohem složitější aktualizovat stav listu úkolů.

Závěrem tedy je, že v případě jednodušších aplikací nebude ani tolik záležet na výběru frameworku, jako na preferencích programátora.

2.2.7. Výkonnost

Pro porovnání výkonnosti u každého frameworku byl zvolen stejný postup jako v předchozí kapitole, tedy měření po 10 sekundách v 10 iteracích na základě automatizovaného scriptu se stejnou metodou sběru dat.

Bohužel v případě výkonnosti se nepodařilo nasbírat věrohodná data. Důvodem je, že používaná sestava pro testování byla příliš výkonná pro simulaci jakéhokoliv zatížení. Byl vyzkoušen i postup manuálního testování – v tomto případě bylo do každého frameworku přidáno 10 000 úkolů – znovu však bez jakékoliv změny v FPS.

2.3. Kalkulátor

2.3.1. Úvod do problematiky

Jedním z primárních cílů této bakalářské práce je provést srovnání různých frameworků. Avšak, jak je již patrné z předchozích kapitol, existuje mnoho hledisek a perspektiv, z nichž lze na frameworky nahlížet. Je možné, že jeden framework disponuje rozsáhlou dokumentací, ale zároveň vyžaduje placenou licenci. Naopak, druhý framework může nabízet bezplatnou licenci, ale jeho komunita nemusí být tak rozsáhlá.

Na teoretické úrovni lze frameworky rozdělit do skupin. Komplikace však nastávají při praktickém využití, neboť neexistuje univerzální metrika, škála nebo bodový systém, který by umožňoval srovnání implementací.

Po důkladném zvážení jsem se rozhodl, že v této části práce se nebudu soustředit na srovnávání konkrétního kódu. Místo toho se zaměřím na to, jak se každá implementace dokáže přiblížit jednomu vizuálnímu vzoru a to PyQt5.

Praktická část práce se tedy zaměří hlavně na vizuální aspekty jednotlivých komponent a na s nimi související vlastnosti, které budou následovat ukázky částí implementací pro jednotlivé frameworky.

2.3.2. Spuštění aplikace a struktura Front-Endu

Pro co největší usnadnění a přehlednost, je kompletní aplikace tvořena čtyřmi FE (Front End) a jedním BE (Back End).

Spuštění aplikace

Uživatel má tedy při spouštění možnost, zvolit si z jednoho z frameworků argumentem z příkazové řádky.

```python
def main():
    # Zkontroluje, zda byl předán argument příkazové řádky
    if len(sys.argv) < 2:
        print("Usage: python main.py [pyqt|kivy|tkinter|wx]")
        sys.exit(1)
    # Načte argument a převede ho na malá písmena pro porovnání
    calc_type = sys.argv[1].lower()
    # Podle typu kalkulačky vytvoří a spustí instanci
    if calc_type == "pyqt":
        # Vytvoří se instance QApplicaton a CalculatorPyqt
        app = QApplication(sys.argv)
        calc = CalculatorPyqt()
        # Spustí se aplikace pomocí app.exec_()
        sys.exit(app.exec_())
    elif calc_type == "kivy":
        # Spustí se aplikace CalculatorApp
        from gui.kivy.CalculatorKivy import CalculatorApp
        CalculatorApp().run()
    elif calc_type == "tkinter":
        # Vytvoří se instance Calculator a spustí se metoda run()
        calc = Calculator()
        calc.run()
    elif calc_type == "wx":
        # Vytvoří se instance CalculatorWx
        CalculatorWx()
    else:
        # Pokud byl předán neplatný argument, vytiskne se chybové hlášení
        print("Invalid calculator type. Please choose from [pyqt|kivy|tkinter|wx].")
        sys.exit(1)
```

Obrázek 27 Spuštění - screenshot kódu

Struktura aplikace

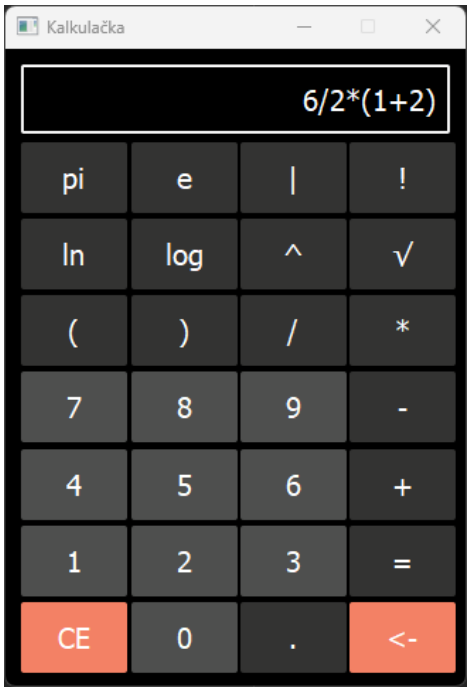
Z důvodu potřeby komunikace všech čtyř FE částí aplikace s jedinou BE částí bylo nezbytné najít vhodné řešení, jak zajistit konzistentní postup volání. Tento problém jsem vyřešil využitím jednotného rozhraní (interface), které slouží jako průsečík pro umístění kódu frameworku tak, aby bylo dosaženo optimální funkčnosti**.** Toto řešení zajistilo nejen konzistentnější kód pro GUI, ale také umožnilo přehlednější ukázky v následujících kapitolách.

```python
class ICalculator(ABC):
    # Abstraktní metoda pro vytvoření a nastavení UI kalkulačky. Obsahuje inicializace jednotlivých
    # komponent a v některých případech, pokud to technologie vyžadovala, i nastavení některých stylů.
    @abstractmethod
    def initUI(self):
        pass

    # Abstraktní metoda pro vytvoření tlačítek kalkulačky.
    # Pro každé tlačítko volá metodu create_button s textem a názvem tlačítka (pokud existuje).
    @abstractmethod
    def create_buttons(self):
        pass

    # Abstraktní metoda, která přidává tlačítka do layoutu pro uspořádání tlačítek na widgetu.
    @abstractmethod
    def add_buttons_to_layout(self):
        pass

    # Abstraktní metoda pro zobrazení kritického modální okna, které informuje uživatele o chybě.
    @abstractmethod
    def show_critical_messagebox(self, type_of_exception: str, text: str):
        pass

    # Abstraktní metoda pro vytvoření tlačítka s daným textem a názvem.
    # Nastavuje pevnou velikost a připojuje signál k slotu, který aktualizuje obsah okna pro výpočet.
    @abstractmethod
    def create_button(self, text, name=None):
        pass

    # Abstraktní metoda pro přidání stylů k widgetům.
    # V případě frameworku PyQt5 je řešení pomocí technologie CSS, ostatní frameworky mají vlastní řešení.
    @abstractmethod
    def add_styles(self):
        pass

    # Jediná metoda stejná pro všechny frameworky.
    # Tato metoda je z většiny používána jako callback při stisknutí tlačítka,
    # jednotlivé implementace budou ukázány v dalších kapitolách
    def button_clicked(self, text_of_button: str, actual_text: str) -> str:
        """Funkční kód"""
```

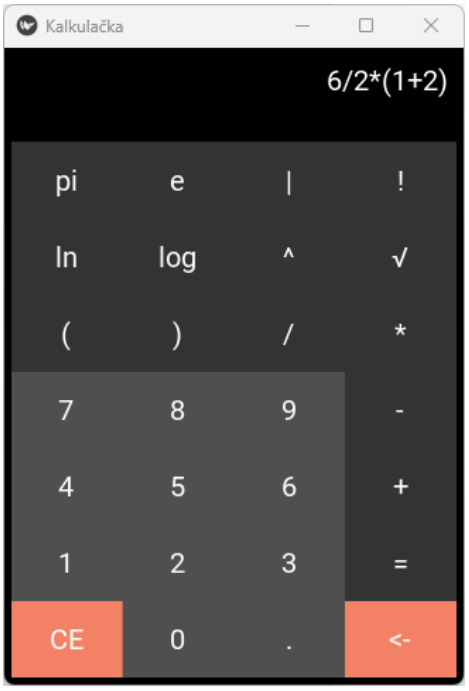
Obrázek 28 ICalculator - screenshot kódu

Jak je vidět na obrázku výše, je využíváno 6 abstraktních metod. Každé řešení využívající jednotlivé frameworky implementuje právě tento interface. Konkrétní popis implementací je nastíněn v dalších kapitolách.

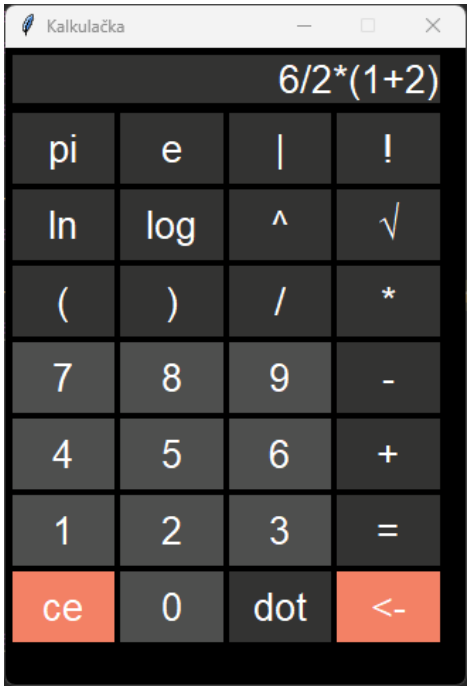
2.3.3. Porovnání vzhledu



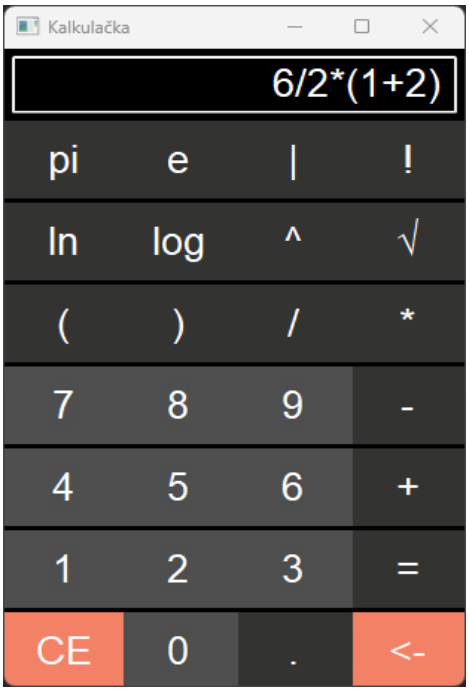
Obrázek 29 PyQt5 – screenshot



Obrázek 30 Kivy – screenshot



Obrázek 31 Tkinter – screenshot



Obrázek 32 WxPython – screenshot

2.3.4. Odchylky

Přejíždění myši po komponentě

Velkým rozdílem, který není na obrázcích výše vidět je, že v případě frameworku Kivy nefunguje změna barvy při přejetí nad tlačítkem. Původní myšlenkou bylo, že při přejetí nad jednotlivými tlačítky se komponenta změní podle nastavení. Nicméně u frameworku Kivy je toto nastavení sice možné, ale nejde realizovat se zvoleným rozhraním. Řešení vyžaduje použití „stromu widget“, který umožňuje nahrát strukturu a některá nastavení komponent.

Dalším problematickým frameworkem, je WxPython. I přes veškeré snahy, se mi nepodařilo upravit změnu barvy. Oficiálním postupem je přidat tlačítkům tlačítku atributy *EVT_ENTER_WINDOW* a *EVT_LEAVE_WINDOW* s callbacky na příslušné metody (implementace na obrázku 21 v následujících kapitolách). Tento postup je sice v kódu realizovaný, ale také naprosto neúčinný. Dohledal jsem pouze jeden postup s použitím vlastního eventu, který ovšem také není funkční. Po delším řešení této problematiky, jsem se vzhledem k neexistujícím dalším informacím rozhodl ponechat kód v aktuálním stavu s vírou, že bude tato chyba opravena v dalších verzích.

Mezery mezi komponentami

Viditelným rozdílem mezi v obrázcích výše, je absence *marginu* mezi jednotlivými tlačítky u frameworků Kivy a WxPython**.**

V případě frameworku Kivy jsou *wignety* umístěny relativním uspořádání k rodiči. Je tedy velmi obtížené, použít *margin* pro jednotlivé komponenty jako jsou v tomto případě tlačítka, nebo výstup pro příklad.

V případě frameworku WxPython je řešení značně snazší, lze použít metodu *SetMargins*. Nicméně i přes veškeré snažení, se mi toto řešení nepodařilo spojit s korektním zarovnáním komponent do okna, zvolil jsem tedy variantu bez *marginu*.

V ostatních frameworcích je přejíždění již funkční.

Barvy komponent

Tento rozdíl je spíše technického rázu než vizuálního.

Frameworku Kivy nepodporuje HEX formát barev. Příkladem jsou numerická tlačítka kalkulačky se standardním nastavením pozadí u ostatních frameworků na „#4e4f4e“.

Případě frameworku Kivy musela být pro stejnou barvu použita RGBA n-tice

„(0.3058823529411765, 0.30980392156862746, 0.3058823529411765, 1)“

Kivy nabízí vlastní API pro převod barev z formátu HEX na formát RGBA, ale výsledky po převodu nejsou nikdy 100% stejné jako původní vzhled HEX barvy. Tato skutečnost je při bližším zkoumání patrná i na obrázcích výše.

Zarovnání komponentů

Snad největším rozdílem viditelným na obrázcích je u frameworku Tkinter.

I přes veškerou snahu, se mi nepovedlo pozměnit zarovnání komponentů na střed. Řešení by mělo existovat, ale bohužel všechny použité metody se minuly účinkem.

Řešení buď vyžadují kompletní změnu implementace, nebo rozbíjejí ostatní funkčnosti frameworku.

V ostatních frameworcích je zarovnání již funkční.

2.3.5. Odchytávání stisku tlačítka

V této kapitole se zaměřím na představení ukázek kódu pro různé zajímavé části uživatelského rozhraní (GUI).

Kivy

```
def create_button(self, text, name=None):
    button = Button(
        text=text,
        background_normal='',
    )
    button.name = name or text
    button.bind(on_release=lambda button: self.button_clicked_proxy(button.name, self.display.text))
    return button

def button_clicked_proxy(self, text_of_button: str, actual_text: str):
    self.display.text = self.button_clicked(text_of_button, actual_text)
```

Obrázek 33 Kivy – tvorba tlačítka

Kód zahrnuje dvě metody – jednu zděděnou z rozhraní a druhou pomocnou. Metoda *button_clicked_proxy* odchytává stisk tlačítka vytvořeného pomocí metody *create_button*, která je navázána na tuto událost prostřednictvím metody *bind*. Metoda využívá anonymní funkci s klíčovým slovem *lambda*, která volá metodu *button_clicked_proxy* s parametry *text_of_button* a *actual_text*. Proxy metoda aktualizuje text v aplikaci tím, že volá metodu *button_clicked* s těmito parametry. Metoda *button_clicked_proxy* tedy slouží k odchytávání stisku tlačítka a aktualizaci zobrazeného textu v aplikaci.

Implementace Kivy musela upravit původní zamýšlenou mechaniku rozhraní. Použití *proxy* metody bylo nutné pro úpravu *display.text*, protože na rozdíl od například PyQt5 editace komponenty *TextInput* neprobíhá metodou, ale úpravou atributu, což v lambda funkci není možné.

PyQt5

```
def create_button(self, text, name=None):
    button = QPushButton(text)
    button.setFixedSize(75, 50)
    button.clicked.connect(lambda: self.line_edit.setText(self.button_clicked(text, self.line_edit.text())))
    setattr(self, "button_" + (name if name else text), button)
```

Obrázek 34 PyQt5 – tvorba tlačítka

Kód definuje metodu *create_button* pro vytváření tlačítek v aplikaci. Stisk tlačítka je odchycen pomocí metody *clicked* a propojení s anonymní funkcí vytvořenou pomocí klíčového slova *lambda*. Anonymní funkce přijímá dva parametry – *text* (text tlačítka) a *self.line_edit.text* (aktuální textový řetězec v poli pro vstup) – a poté volá metodu *button_clicked* z rozhraní s těmito parametry. Kód tak zajišťuje, že po stisku tlačítka se spustí funkce *button_clicked* z rozhraní s příslušnými parametry.

Editace textu komponenty QLineEdit probíhá metodou setText.

WxPython

```
def create_button(self, text: str, nothing=None) -> wx.Button:
    btn = wx.Button(self.panel, label=text)
    btn.SetMinSize((80, 55))
    btn.SetMaxSize((80, 55))
    btn.Bind(wx.EVT_BUTTON,
             lambda click: self.example.SetLabel(self.button_clicked(btn.GetLabel(), self.example.GetLabel())))
    return btn
```

Obrázek 35 WxPython – tvorba tlačítka

Kód definuje metodu *create_button* pro vytváření tlačítek v aplikaci.

Metoda vytváří tlačítko s pevnou velikostí a propojuje ho s funkcí, která aktualizuje textový widget v aplikaci. Stisk tlačítka je odchycen pomocí metody *bind* a anonymní funkce vytvořené pomocí klíčového slova *lambda*.

Editace textu komponenty *TextCtrl* probíhá metodou *SetLabel*.

Zajímavostí v jazyce Python oproti Javě je hlavička metody. Na rozdíl od Javy, kde je rozhraní považováno za pevné smluvní rozhraní a jména argumentů nelze měnit, v Pythonu to možné je. To umožnilo změnu atributu *name* na *nothing*, což lépe vystihuje jeho význam, jelikož v této implementaci není jméno tlačítka pro funkčnost potřebné.

Tkinter

```
def create_button(self, text, nothing=None) -> tk.Button:
    ## Funkční kód
    button = tk.Button(
        ## Funkční kód
        command=lambda: self.button_clicked_proxy(text, self.display.get())
    )
    ## Funkční kód
    return button

def button_clicked_proxy(self, text_of_button: str, actual_text: str) -> str:
    self.actual_text = self.button_clicked(text_of_button, actual_text)
```

Obrázek 36 Tkinter – tvorba tlačítka

Funkce *create_button* slouží k vytvoření tlačítka v GUI s daným textem. Toto tlačítko může být stisknuto a vyvolá metodu *button_clicked_proxy* s dvěma argumenty – *text* tlačítka a *self.display.get* - aktuálním textem zobrazeném v poli v GUI.

Funkce button_clicked_proxy působí jako prostředník mezi tlačítkem a funkcí *button_clicked*. Tento prostředník předává text tlačítka a aktuální text zobrazený v GUI do funkce *button_clicked*, která provede určité operace a vrátí nový text, který se pak zobrazí v GUI.

Shrnutí

V přehledu těchto implementací lze vidět, že odchytávání stisku tlačítka a aktualizace zobrazeného textu v aplikaci probíhá podobně napříč různými frameworky, i když s drobnými odlišnostmi. Významné je, že každý z frameworků používá metodu nebo anonymní funkci (lambda) pro zachycení stisku tlačítka a propojení s příslušnou funkcí či metodou. Navíc, každý z frameworků upravuje zobrazený text v aplikaci pomocí specifických metod či atributů, jako jsou *setText*, *SetLabel* nebo úprava atributu.

2.3.6. Nastavení stylů

Kivy

```
def add_styles(self):
    Window.background_color = (0.8, 0.8, 0.8, 1)
    self.display.background_color = (0, 0, 0, 1)
    self.display.foreground_color = (1, 1, 1, 1)
    self.display.font_size = 20
    for button in self.buttons:
        button.font_size = 20
        button.color = (1, 1, 1, 1)
        if button.name == 'CE' or button.name == '<-':
            button.background_color = (0.9529411764705882, 0.5058823529411764, 0.396078431372549, 1)
        elif button.name in '0123456789.':
            button.background_color = (0.3058823529411765, 0.30980392156862746, 0.3058823529411765, 1)
        else:
            button.background_color = (0.2, 0.2, 0.19607843137254902, 1)
```

Obrázek 37 Kivy – styly

Funkce *add_styles* slouží k přidání vlastních stylů pro grafické komponenty. Vzhledem k tomu, že úpravy se provádějí na základě změn atributů v jednotlivých komponentách, byla původní implementace stylování v Kivy upravena. Původně se plánovalo, že všechny styly budou přidány pomocí funkce *add_styles*, ale později se ukázalo jako efektivnější vložit základní styly, jako například zarovnání fontů v *TextInput*, při inicializaci v metodě *initUI*. V kapitole vizuální porovnání byl popsán velký problém spočívající v tom, že Kivy používá formát barev RGBA namísto HEX formátu, který je běžný v ostatních frameworcích. Příkladem na obrázku je

n-tice (0.9529411764705882, 0.5058823529411764, 0.396078431372549, 1)

zastupující barvu #f38165, používanou pro stejná tlačítka v ostatních frameworcích.

PyQt5

```
def add_styles(self):
    # Funkční kód
    line_edit_style = """
        height: 200px;
        padding: 5px;
        text-align: right;
    """
    num_buttons_style = """
    QPushButton {
         background-color: #4e4f4e;
         border-radius: 2px;
    }
    QPushButton:hover {
         background-color: rgba(51,51,50,255);
    }
    """
    # Funkční kód
    self.line_edit.setStyleSheet(line_edit_style)
    self.button_0.setStyleSheet(num_buttons_style)
    self.button_1.setStyleSheet(num_buttons_style)
    # Funkční kód
```

Obrázek 38 PyQt5 – styly

Funkce *add_styles* slouží k přidání vlastních stylů pro grafické komponenty. Na rozdíl od Kivy je většina implementací stylů nastavena právě v metodě *add_styles*. Pyqt5 je jediný z testovaných frameworků, který umožňuje modifikaci vzhledu pomocí CSS. Modifikace všech komponent probíhá pomocí metody *setStyleSheet*, která přijímá styl ve formátu řetězce jako vstup. Na implementaci *hover* je jasně patrné, že vývojáři frameworku umožnili i využívání aktivního stylování kaskádových stylů. Tato vlastnost umožnila výrazně snazší řešení změny barvy při přejíždění tlačítek, než tomu bylo v ostatních implementacích.

WxPython

```
def add_styles(self):
    self.example.SetForegroundColour("white")
    self.example.SetBackgroundColour("#000000")
    for child in self.panel.GetChildren():
        if isinstance(child, wx.Button):
            child.SetFont(wx.Font(20,
                                  wx.FONTFAMILY_DEFAULT,
                                  wx.FONTSTYLE_NORMAL,
                                  wx.FONTWEIGHT_NORMAL))
            child.SetForegroundColour("#ffffff")
            child.Bind(wx.EVT_ENTER_WINDOW, self.on_mouseover)
            child.Bind(wx.EVT_LEAVE_WINDOW, self.on_mouseleave)

    self.panel.SetSizer(self.sizer)
```

Obrázek 39 WxPython – styly

Funkce *add_styles* má za úkol přidat uživatelsky definované styly pro grafické komponenty. v případě frameworku WxPython se pro dosažení tohoto cíle využívá kombinace metod *add_styles* a *initUI*. Určitá část stylů je definována v samotné metodě *add_styles*, zatímco jiná část je nastavena v metodě *initUI*. Při vývoji s použitím frameworku WxPython je nutné mít hluboké znalosti daných postupů, neboť se jedná o framework, který při určitých případech, jako je právě stylování, vyžaduje velkou pozornost a pečlivost. Pokud jsou některé styly nastaveny až po inicializaci komponenty, nebudou změny v GUI viditelné. V takovém případě je tedy nezbytné přidat styly již před vložením komponenty do layoutu.

Tkinter

```
def add_styles(self):
    style = ttk.Style()

    style.configure("TButton", font=("Arial", 20), background="#333332", foreground="white")

    style.map(
        "TButton",
        background=[("active", "#4e4f4e")],
        foreground=[("active", "white")]
    )

    # Funkční kód
```

Obrázek 40 Tkinter – styly

Úkolem funkce *add_styles* je přidání uživatelsky definovaných stylů pro grafické komponenty v Tkinter frameworku. Tkinter je odlišný od předchozích frameworků tím, že ke konfiguraci stylů používá speciální objekt *Style*, který musí být nejprve nakonfigurován metodou *configure*. Metoda *configure* přijímá argumenty, jako je typ cíle (v tomto případě *TButton*) a následující argumenty ve formě stylů.

Na obrázku výše lze také vidět metodu *map*, která byla použita k řešení změn barev tlačítek při přejetí myší. Pro účely tohoto obrázku byl ukázkový kód značně zkrácen, protože použití objektu "Style" výrazně prodloužilo samotnou metodu více než u předchozích případů.

ZÁVĚR

V této bakalářské práci jsem provedl rozsáhlé srovnání čtyř populárních frameworků pro vývoj grafického uživatelského rozhraní v Pythonu: PyQt5, Tkinter, Kivy a WxPython. Zjištění ukazují, že každý framework má své specifické silné a slabé stránky, které ovlivňují jeho vhodnost pro různé typy projektů.

PyQt5 se vyznačuje vysokou stabilitou a výkonností, je vhodný pro složitější aplikace a nabízí podporu pro CSS stylování.

Na druhé straně Tkinter, s jeho jednoduchostí a nižší výkonností, je ideální pro menší a jednodušší projekty.

Kivy vyniká ve vývoji aplikací s bohatým uživatelským rozhraním a podporou multiplatformního nasazení.

WxPython poskytuje solidní výkon a nativní vzhled aplikací napříč platformami.

Z hlediska komunitní podpory a dokumentace jsou všechny frameworky dobře podporovány, s aktivními komunitami na Stack Overflow a GitHub. Licence a cena také hrají důležitou roli při výběru frameworku, s různými možnostmi od open-source licencí po komerční licence.

Na základě provedeného srovnání lze doporučit PyQt5 pro vývojáře hledající vysoký výkon a rozsáhlé možnosti stylizace, Tkinter pro jednoduché projekty a rychlý vývoj, Kivy pro aplikace s bohatým uživatelským rozhraním a multiplatformní potřeby a WxPython pro projekty, kde je důležitý nativní vzhled napříč různými operačními systémy.

Výběr nejvhodnějšího frameworku by měl být vždy založen na specifických požadavcích a cílech projektu, stejně jako na preferencích a zkušenostech vývojáře.

Citovaná literatura

[1] PyQt Documentation v5.15.4 Introduction. In: *Www.riverbankcomputing.com* [online]. Edinburgh: Riverbank Computing Limited, 2022 [cit. 2023-03-04]. Dostupné z: https://www.riverbankcomputing.com/static/Docs/PyQt5/introduction.html

[2] Tkinter — Python interface to Tcl/Tk. In: *Docs.python.org* [online]. Wilmington, Delaware: Python Software Foundation, 2022 [cit. 2023-03-04]. Dostupné z: https://docs.python.org/3/library/tkinter.html

[3] What is Kivy?. In: *Geeks for Geeks* [online]. Noida, Uttar Pradesh: Geeks for Geeks, 2020 [cit. 2023-03-04]. Dostupné z: https://www.geeksforgeeks.org/what-is-kivy/

[4] Welcome to wxPython!. In: *Wxpython.org* [online]. San Francisco: The wxPython Team, 2022 [cit. 2023-03-04]. Dostupné z: https://wxpython.org/pages/overview/

[5] Stackoverflow pyqt5 tag. In: *Stackoverflow* [online]. New York: Stack Exchange Inc, 2023 [cit. 2023-03-04]. Dostupné z: https://stackoverflow.com/questions/tagged/pyqt5

[6] Stackoverflow tkinter tag. In: *Stackoverflow* [online]. New York: Stack Exchange Inc, 2023 [cit. 2023-03-04]. Dostupné z: https://stackoverflow.com/questions/tagged/tkinter

[7] Stackoverflow kivy tag. In: *Stackoverflow* [online]. New York: Stack Exchange Inc, 2023 [cit. 2023-03-04]. Dostupné z: https://stackoverflow.com/questions/tagged/kivy

[8] Stackoverflow wxpython tag. In: *Stackoverflow* [online]. New York: Stack Exchange Inc, 2023 [cit. 2023-03-04]. Dostupné z: https://stackoverflow.com/questions/tagged/wxpython

[9] Github pyqt5 tag. In: *Github.com* [online]. San Francisco: GitHub, Inc., 2023 [cit. 2023-03-04]. Dostupné z: https://github.com/topics/pyqt5

[10] Github tkinter tag. In: *Github.com* [online]. San Francisco: GitHub, Inc., 2023 [cit. 2023-03-04]. Dostupné z: https://github.com/topics/tkinter

[11] Github kivy tag. In: *Github.com* [online]. San Francisco: GitHub, Inc., 2023 [cit. 2023-03-04]. Dostupné z: https://github.com/topics/kivy

[12] Github wxpython tag. In: *Github.com* [online]. San Francisco: GitHub, Inc., 2023 [cit. 2023-03-04]. Dostupné z: https://github.com/topics/wxpython

[13] Kivy. In: *Https://kivy.org/* [online]. San Francisco: Kivy organization, 2023 [cit. 2023-03-05]. Dostupné z: https://kivy.org/

[14] MIT License. In: *Fossa* [online]. San Francisco: FOSSA, Inc., 2023 [cit. 2023-03-05]. Dostupné z: https://fossa.com/blog/open-source-licenses-101-mit-license/

[15] What are the MIT license terms and conditions?. In: *Snyk* [online]. Boston: Snyk Limited, 2023 [cit. 2023-03-05]. Dostupné z: https://snyk.io/learn/what-is-mit-license/

[16] License FAQ. In: *Riverbankcomputing* [online]. Edinburgh: Riverbank Computing Limited, 2023 [cit. 2023-03-05]. Dostupné z: https://riverbankcomputing.com/commercial/license-faq

[17] GNU General Public License, version 1. In: *Gnu* [online]. Boston: Free Software Foundation, 2022 [cit. 2023-03-05]. Dostupné z: https://www.gnu.org/licenses/old-licenses/gpl-1.0.html

[18] The Foundations of the GPL. In: *Gnu* [online]. Boston: Free Software Foundation, 2022 [cit. 2023-03-05]. Dostupné z: https://www.gnu.org/licenses/quick-guide-gplv3.html

[19] Licence. In: *Wxpython* [online]. San Francisco: The wxPython Team, 2022 [cit. 2023-03-05]. Dostupné z: https://wxpython.org/pages/license/index.html

[20] Why you shouldn't use the Lesser GPL for your next library. In: *Gnu* [online]. Boston: Free Software Foundation, 2022 [cit. 2023-03-05]. Dostupné z: https://www.gnu.org/licenses/why-not-lgpl.html

[21] Tcl/Tk License Terms. In: *Tcl* [online]. San Francisco: Tlc Developer Xchange, 2020 [cit. 2023-03-05]. Dostupné z: https://www.tcl.tk/software/tcltk/license.html

[22] Kivy - Technical FAQ. In: *Kivy - Technical FAQ* [online]. San Francisco: Kivy organization, 2017 [cit. 2023-04-23]. Dostupné z: https://kivy.org/doc/stable/faq.html

[23] Kivy compatibility module. In: *Kivy.org* [online]. San Francisco: Kivy organization, 2023 [cit. 2023-04-23]. Dostupné z: https://kivy.org/doc/stable/api-kivy.compat.html

[24] WxPython migration. In: *Docs.wxpython* [online]. San Francisco: The wxPython Team, 2018 [cit. 2023-04-23]. Dostupné z: https://docs.wxpython.org/MigrationGuide.html

[25] Realpython - wxpython. In: *Realpython* [online]. Kanada: Dan Bader, 2019 [cit. 2023-04-23]. Dostupné z: https://realpython.com/python-gui-with-wxpython/

[26] Python 3 - Pyqt5 - kompatibilita. In: *Pypi* [online]. Edinburgh, South Australia, Australia: Riverbank Computing Limited, 2023 [cit. 2023-04-30]. Dostupné z: https://pypi.org/project/PyQt5/

[27] Python 3 - Pyqt5 - kompatibilita (Python2). In: *Wiki.python* [online]. Netherlands: Python Software Foundation, 2021 [cit. 2023-04-30]. Dostupné z: https://wiki.python.org/moin/PyQt

[28] Tkinter - kompatibilita. In: *Docs.python.org* [online]. Wilmington, Delaware: Python Software Foundation, 2023 [cit. 2023-04-30]. Dostupné z: https://docs.python.org/3/howto/pyporting.html

[29] Příklad. In: *Facebook* [online]. Menlo Park: facebook, 2018 [cit. 2023-04-29]. Dostupné z: https://www.facebook.com/horseedcollege/photos/a.612078505848876/931387300584660/

[30] Kivy - dokumentace. In: *Kivy* [online]. San Francisco: Kivy organization, 2023 [cit. 2023-04-30]. Dostupné z: https://kivy.org/doc/stable/

[31] WxPython - dokumentace. In: *Wxpython* [online]. San Francisco: The wxPython Team, 2023 [cit. 2023-04-30]. Dostupné z: https://docs.wxpython.org/

[32] PyQt5 - dokumentace. In: *Riverbankcomputing* [online]. Edinburgh: Riverbank Computing Limited, 2022 [cit. 2023-04-30]. Dostupné z: https://www.riverbankcomputing.com/static/Docs/PyQt5/

[33] Tkinter - dokumentace. In: *Docs.python* [online]. Wilmington Delaware: Python Software Foundation, 2022 [cit. 2023-04-30]. Dostupné z: https://docs.python.org/3/

[34] PyQT5 - instalace. In: *Pythonguis* [online]. Netherlands: Martin Fitzpatrick, 2019 [cit. 2023-05-02]. Dostupné z: https://www.pythonguis.com/installation/install-pyqt-windows/

[35] Kivy - Instalace - windows. In: *Kivy* [online]. San Francisco: Kivy organization, 2023 [cit. 2023-05-02]. Dostupné z: https://kivy.org/doc/stable-1.11.1/installation/installation-windows.html

[36] Kivy - Instalace - linux. In: *Kivy* [online]. San Francisco: Kivy organization, 2023 [cit. 2023-05-02]. Dostupné z: https://kivy.org/doc/stable-1.10.1/installation/installation-linux.html

[37] Wxpython - Instalace - Windows + Linux. In: *Wxpython* [online]. San Francisco: The wxPython Team, 2022 [cit. 2023-05-02]. Dostupné z: https://wxpython.org/pages/downloads/

[38] Tkinter - Instalace - Windows. In: *Docs.python* [online]. Wilmington, Delaware: Python Software Foundation, 2023 [cit. 2023-05-02]. Dostupné z: https://docs.python.org/3/library/tkinter.html

[39] EBNF. In: *Dzone* [online]. Severní Karolína, USA: DZone MVB, 2017 [cit. 2023-05-03]. Dostupné z: https://dzone.com/articles/ebnf-how-to-describe-the-grammar-of-a-language