|  |  |  |
| --- | --- | --- |
|  | | |
| **Závěrečná studijní práce**  **dokumentace** | | |
| **Time Tracker** | | |
| Jan Beníšek | | |
| Obsah obrázku hodiny, kruh, Grafika, klipart  Popis byl vytvořen automaticky | | |
|  | |  |
| **Obor:** | 18-20-M/01 INFORMAČNÍ TECHNOLOGIE  se zaměřením na počítačové sítě a programování | |
| **Třída:**  **Školní rok:** | IT4  2024/2025 | |

**Poděkování**

*Rád bych poděkoval rodině za podporu a pánům učitelům Ing. Petru Grussmannovi a Mgr. Marku Lučnému za jejich pomoc s projektem, jelikož mi pomohli se směřováním projektu.*

Prohlašuji, že jsem závěrečnou práci vypracoval samostatně a uvedl veškeré použité   
informační zdroje.

Souhlasím, aby tato studijní práce byla použita k výukovým účelům na Střední průmyslové   
a umělecké škole v Opavě, Praskova 399/8.

V Opavě 15. 1. 2025

*podpis autora práce*

**ABSTRACT**

Tato aplikace, nazvaná **TimeTracker**, je navržena pro monitorování uživatelských aktivit na počítači. Program sleduje aktivní okna, dobu jejich aktivity a zobrazuje jejich název, Id procesu – PID a využití systémových prostředků (CPU, RAM). Aplikace se zaměřuje na primárně uživatelsky přívětivé aplikace, jako jsou webové prohlížeče a běžné programy, a vylučuje systémové procesy, které nemají uživatelské rozhraní. Vypisuje také zaznamenanou aktivitu do souboru CSV.

TimeTracker umožňuje zobrazení aktuálně běžících aplikací v reálném čase s automatickým obnovenímv základu sekundu lze to však změnit. Poskytuje tabulkové zobrazení dat, které lze řadit podle různých kritérií, například podle času aktivace nebo využití zdrojů. Aplikace podporuje neomezený počet záznamů s možností scrollování a statickým oknem, aby byla data snadno přístupná.

Program se spouští při startu systému a je možné jej otevřít kliknutím na ikonu v systémové liště. TimeTracker je určen pro detailní analýzu a přehled využití počítače, a to jak pro individuální potřeby uživatele, tak pro účely správy času.

Celkově aplikace přispívá k lepšímu pochopení toho, jak jsou zdroje systému využívány a jaký čas je věnován konkrétním úkolům.

Klíčová slova: Monitorování aktivit, využití CPU a RAM, CSV export, správa času

**ABSTRACT**

This application, called TimeTracker, is designed to monitor user activity on a computer. The program tracks active windows, their activity time and displays their name, pro-process Id - PID and system resource usage (CPU, RAM). The application focuses on pri-marily user-friendly applications such as web browsers and common programs, and excludes system processes that do not have a user interface. It also outputs logged activity to a CSV file.

TimeTracker allows you to view the currently running applications in real time with automatic refreshin the base second however this can be changed. It provides a tabular display of data that can be sorted by various criteria, such as activation time or resource usage. The application supports unlimited records with scrolling and a static window to make the data easily accessible.

The program starts at system boot and can be opened by clicking on the icon in the system tray. TimeTracker is designed for detailed analysis and overview of computer usage, both for individual user needs and for time management purposes.

Overall, the application contributes to a better understanding of how system resources are used and what time is spent on specific tasks.

Key words: Activity monitoring, CPU and RAM usage, CSV export, time management

OBSAH

[Úvod 5](#_Toc187670905)

[1 Úvod do sledování aktivity a zdrojů 6](#_Toc187670906)

[1.1 Aplikace na sledování aktivity a zdrojů 6](#_Toc187670907)

[1.2 Princip fungování aplikace 6](#_Toc187670908)

[2 Využité technologie 7](#_Toc187670909)

[2.1 Programovací jazyk 7](#_Toc187670910)

[2.2 Knihovna pro sledování procesů 7](#_Toc187670911)

[2.3 Knihovna pro GUI 7](#_Toc187670912)

[2.4 Knihovna pro snímky obrazovky 7](#_Toc187670913)

[2.5 Knihovna pro uživatelská nastavení 7](#_Toc187670914)

[3 Způsoby řešení a použité postupy 9](#_Toc187670915)

[3.1 Sledování aktivních oken 9](#_Toc187670916)

[3.2 Sběr systémových informací 10](#_Toc187670917)

[3.3 Pořízení screenshotů 11](#_Toc187670918)

[3.4 Grafické uživatelské rozhraní (GUI) 12](#_Toc187670919)

[3.5 Nastavení aplikace 12](#_Toc187670920)

[3.6 Více vláknové zpracování a plánování úkolů 14](#_Toc187670921)

[3.7 Zápis do csv souboru 15](#_Toc187670922)

[3.8 Automatické spuštění aplikace při startu systému 15](#_Toc187670923)

[3.9 Filtrování tabulky 16](#_Toc187670924)

[3.10 Struktura aplikace 17](#_Toc187670925)

[4 Výsledky řešení, výstupy, uživatelský manuál 18](#_Toc187670926)

[4.1 Výsledky řešení 18](#_Toc187670927)

[4.2 Výstupy Aplikace 18](#_Toc187670928)

[4.3 Uživatelský manuál 18](#_Toc187670929)

[4.4 Splnění a nesplněné cíle 19](#_Toc187670930)

[Závěr 20](#_Toc187670931)

[Seznam použitýCH INFORMAČNÍCH ZDROJů 21](#_Toc187670932)

Úvod

Problematika sledování a analýzy využití uživatelských aplikací je v současnosti klíčová pro efektivní správu času a optimalizaci pracovních procesů. Rostoucí počet aplikací a procesů běžících na osobních počítačích klade nároky na přesné monitorování systémových zdrojů a uživatelské aktivity. Tato práce se zaměřuje na vývoj a implementaci aplikace

TimeTracker, která poskytuje podrobný přehled o aktivních uživatelských aplikacích, jejich čase používání a využití systémových prostředků.

Hlavním cílem práce je vytvoření nástroje, který umožní uživatelům nebo administrátorům sledovat aplikace v reálném čase, analyzovat využití CPU, RAM a pomocí automatických screenshotů a výpisů aktivity spravovat jejich aktivitu efektivněji.

Práce si klade za cíl poskytnout užitečný nástroj pro sledování aktivity, který může být použit jak v pracovním, tak osobním prostředí, a zároveň inspirovat k dalšímu rozvoji podobných systémů.

# Úvod do sledování aktivity a zdrojů

## Aplikace na sledování aktivity a zdrojů

Existuje celá řada aplikací určených k monitorování systémových zdrojů a aktivit, které uživatelům umožňují sledovat využití procesoru (CPU), operační paměti (RAM) a jednotlivé běžící procesy. Mezi nejběžnější nástroje patří Správce úloh (Windows) a Monitor aktivity (macOS), které poskytují základní přehled o systému. Pokročilejší aplikace, jako například Process Explorer nebo htop, umožňují podrobnější analýzu výkonu a detailní monitoring jednotlivých procesů. Tyto nástroje obvykle nesledují čas strávený v konkrétních aplikacích a neobsahují funkce pro zaznamenávání aktivit uživatele.

## Princip fungování aplikace

Moje aplikace se zaměřuje na sledování času stráveného v jednotlivých aplikacích a využití systémových zdrojů (CPU, RAM) v reálném čase. Aplikace pravidelně (například každé 3 sekundy) načítá aktuální data o běžících procesech a jejich zatížení systémovými prostředky. Na základě těchto údajů identifikuje aktivní aplikaci, zaznamenává její dobu aktivace a dobu trvání používání. Systém automaticky filtruje procesy bez uživatelského rozhraní a systémové služby, což zajišťuje přehlednost dat, ale tuto funkci lze upravit. Uživatel může seřadit data podle různých kritérií, jako je čas aktivace, zatížení CPU nebo délka používání aplikace. Dále je k dispozici funkce pro pořizování screenshotů také spuštěna dle času, který správce nebo uživatel určí, která umožňuje zachytit, co uživatel v daný čas na obrazovce dělá, což zvyšuje efektivitu při sledování a správě produktivity lze také změnit místo kde se budou screenshoty ukládat a je k dispozici také funkce výpisu zaznamenaných aktivit do souboru csv.

# Využité technologie

## Programovací jazyk

Pro vývoj aplikace byl zvolen programovací jazyk Python, který je ideální pro práci s ope-račními systémy a umožňuje rychlý vývoj aplikací s přehledným kódem. Python poskytuje širokou podporu pro práci s operačními prostředími Windows i macOS, což bude v budoucnu klíčové pro dosažení multiplatformní kompatibility aplikace.

## Knihovna pro sledování procesů

Pro monitorování systémových procesů a získávání informací o využití zdrojů byla využita knihovna psutil. Tato knihovna umožňuje snadné získání informací o běžících procesech, zatížení CPU, využití paměti RAM a uživatel pod kterým proces běží dalších systémových zdrojích. Knihovna psutil je velmi efektivní a umožňuje načítat data o procesech v reálném čase, což je nezbytné pro správné fungování aplikace.

## Knihovna pro GUI

Pro vývoj grafického uživatelského rozhraní byla použita knihovna PyQt5, která je založena na frameworku Qt a poskytuje široké možnosti pro tvorbu moderních desktopových aplikací. V aplikaci byla využita především pro správu hlavního okna, systémové lišty a interaktivního menu, což umožňuje ovládat aplikaci přímo z ikony v liště. PyQt5 také umožňuje vytváření tabulek a ovládacích prvků, což bylo klíčové pro zobrazení informací o systémech a procesech, a umožnilo snadnou integraci s dalšími moduly, jako je psutil pro sledování aktivních oken a využití systémových prostředků.

## Knihovna pro snímky obrazovky

Funkce pro pořizování screenshotů byla implementována pomocí knihovny pyscreenshot, která je součástí Pythonu pro snadné pořizování snímků obrazovky na různých operačních systémech. Tato knihovna umožňuje jednoduchý přístup k funkcionalitě pro zachycení celého obrazovky nebo aktivních oken a následné uložení těchto snímků v různých formátech, včetně PNG. V aplikaci byla použita pro pořízení screenshotů každých 10 sekund a jejich uložení do specifikované složky na disku.

## Knihovna pro uživatelská nastavení

Pro implementaci uživatelského nastavení byla použita knihovna PyQt5, která umožňuje snadné vytvoření interaktivních dialogových oken. Klíčové vlastnosti využité pro nastavení zahrnují:

QDialog: Slouží jako základní komponenta pro dialogová okna. Umožňuje snadnou tvorbu modálních oken, ve kterých uživatel může měnit nastavení aplikace.

QSpinBox: Tento prvek umožňuje uživatelům zadávat číselné hodnoty, například interval sledování procesů nebo interval pro pořizování screenshotů. Umožňuje také definovat minimální a maximální hodnoty, což zvyšuje uživatelskou přívětivost.

QPushButton: Tlačítka pro uložení nebo zrušení změn umožňují uživateli snadno interagovat s nastavením.

Díky PyQt5 je dialogové okno nejen funkční, ale také snadno přizpůsobitelné a rozšiřitelné o další parametry, jako je výběr cesty pro ukládání screenshotů nebo přepínač pro deaktivaci určité funkce. Tato technologie umožňuje vytvořit intuitivní rozhraní pro správu konfigurace aplikace.

# Způsoby řešení a použité postupy

## Sledování aktivních oken

Pro sledování aktuálně aktivních oken v systému macOS je využita třída ActivityTracker. Tento modul využívá NSWorkspace a NSWorkspaceDidActivateApplicationNotificati-on, což jsou součásti Cocoa frameworku, který je dostupný pouze na macOS. Pomocí těch-to technologií aplikace monitoruje aktivitu aplikací na pozadí a zaznamenává změny mezi aktivními okny.

* Signál windowChanged: Třída ActivityTracker emituje signál, který informuje o změně aktivního okna. Tento signál je následně zachycen v hlavní třídě aplikace (Menu), kde se zaktualizují informace o aktivním okně.
* Sledování změn okna: Jakmile dojde ke změně aktivního okna, aplikace zazname-ná čas poslední aktivace a dobu trvání aktivity, což je klíčová funkcionalita pro zá-znam aktivních oken.

class ActivityTracker(QObject):

*# Signál emitující změnu okna (jméno aktuálního okna)*

windowChanged = pyqtSignal(str)

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

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

self.current\_window = None *# Ukládá jméno aktuálního okna*

*# Nastavení sledování změn aktivního okna pomocí NSWorkspace*

self.workspace = NSWorkspace.sharedWorkspace()

self.notification\_center = self.workspace.notificationCenter()

self.notification\_center.addObserver\_selector\_name\_object\_(

self,

"applicationActivated:",*# Metoda, která bude volána při aktivaci okna*

NSWorkspaceDidActivateApplicationNotification,

*# Notifikace pro aktivaci okna*

None

)

def applicationActivated\_(self, notification):

*# Získání jména aktuálního aktivního okna*

active\_app = self.workspace.frontmostApplication()

window\_name = active\_app.localizedName() if active\_app else "Unknown"

*# Pokud není okno rozpoznáno, nastaví "Unknown"*

*# Vyvolá signál, jen pokud se změnilo aktivní okno*

if window\_name != self.current\_window:

self.current\_window = window\_name

*# Aktualizuje jméno aktuálního okna*

self.windowChanged.emit(window\_name)

*# Vyvolá signál, že okno bylo změněno*

## Sběr systémových informací

V aplikaci se používá knihovna psutil k získávání informací o procesech běžících na systému, jako jsou:

* CPU a RAM: Monitorujeme využití CPU a paměti pro každý proces pomocí psutil.process\_iter(), která vrací informace o běžících procesech včetně jejich názvů, PID a využití systémových prostředků.
* Filtrace uživatelských procesů: Aplikace filtruje systémové procesy (např. kernel\_task, launchd), které nemají žádnou interakci s uživatelem, a zobrazuje pouze procesy spojené s běžnými aplikacemi. Tato filtrace se provádí v metodě is\_user\_process.
* Zobrazení v tabulce: Informace o procesech jsou zobrazeny v tabulce, která je součástí GUI. Každý řádek zobrazuje název aplikace, její poslední aktivaci, dobu aktivace, CPU a RAM využití, a PID procesu.

def update\_process\_info(self):

*# Zakáže řazení během aktualizace, aby nedošlo k nekonzistencím*

self.table\_widget.setSortingEnabled(False)

self.table\_widget.setRowCount(0) *# Vymaže stávající řádky*

processes = {}

for p in psutil.process\_iter(['pid', 'name', 'cpu\_percent', 'memory\_percent', 'username']):

try:

pid = p.info['pid'] *# Získání ID procesu*

process\_name = p.info['name'] *# Získání názvu procesu*

cpu\_usage = p.info['cpu\_percent'] *# Získání procent využití CPU*

ram\_usage = p.info['memory\_percent']

*# Získání procent využití RAM*

username = p.info['username'] or "Unknown" *# Získání uživatele*

*# Uložení aktuálního textu z vyhledávacího pole*

filter\_text = self.search\_box.text().lower()

if self.is\_user\_process(pid, process\_name) and cpu\_usage is not None and ram\_usage is not None:

ram\_usage = round(ram\_usage, 2)

if process\_name in processes:

processes[process\_name]['cpu\_percent'] += cpu\_usage

processes[process\_name]['memory\_percent'] += ram\_usage

else:

processes[process\_name] = {

'name': process\_name, *# Uložení názvu*

'cpu\_percent': cpu\_usage, *# Uložení procent CPU*

'memory\_percent': ram\_usage, *# Uložení procent RAM*

'pid': pid, *# Uložení ID uživatele*

'username': username *# Uložení uživatele*

}

## Pořízení screenshotů

Pro pořízení screenshotů aplikace využívá knihovnu pyscreenshot. Pořízení screenshotu je naplánováno pomocí QTimer, který každých 300 sekund spustí metodu take\_screenshot.  
Čas, za který je metoda spuštěna lze změnit.

* Ukládání screenshotů: Screenshoty jsou ukládány do výchozí složky (nebo do specifikované složky) s názvem souboru obsahujícím časovou značku.
* Zabezpečení cesty: Před pořízením screenshotu je zajištěno, že složka pro uložení obrázků existuje. Pokud ne, je vytvořena pomocí metody os.makedirs.

class ScreenshotManager:

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

*# Výchozí cesta pro screenshoty*

self.screenshot\_directory = "/tmp/screenshot"

self.ensure\_directory\_exists(self.screenshot\_directory)

def ensure\_directory\_exists(self, directory):

*# Vytvoří složku, pokud neexistuje*

os.makedirs(directory, exist\_ok=True)

def set\_screenshot\_directory(self, new\_directory):

*# Změní výchozí složku pro ukládání screenshotů*

self.screenshot\_directory = new\_directory

self.ensure\_directory\_exists(self.screenshot\_directory)

def take\_screenshot(self):

try:

os.makedirs(self.screenshot\_directory, exist\_ok=True)

*# Zajistí vytvoření složky, pokud neexistuje*

timestamp = QDateTime.currentDateTime().toString

("yyyy.MM.dd-HH-mm-ss") *# Získání časového razítka*

filename = f"screenshot-{timestamp}.png"

file\_path = os.path.join(self.screenshot\_directory, filename)

*# Spojení cesty a názvu souboru*

*# Pořízení snímku obrazovky pomocí pyautogui*

screenshot = pyautogui.screenshot()

screenshot.save(file\_path) *# Uloží screenshot na specifikované místo*

print(f"Screenshot uložen do: {file\_path}")

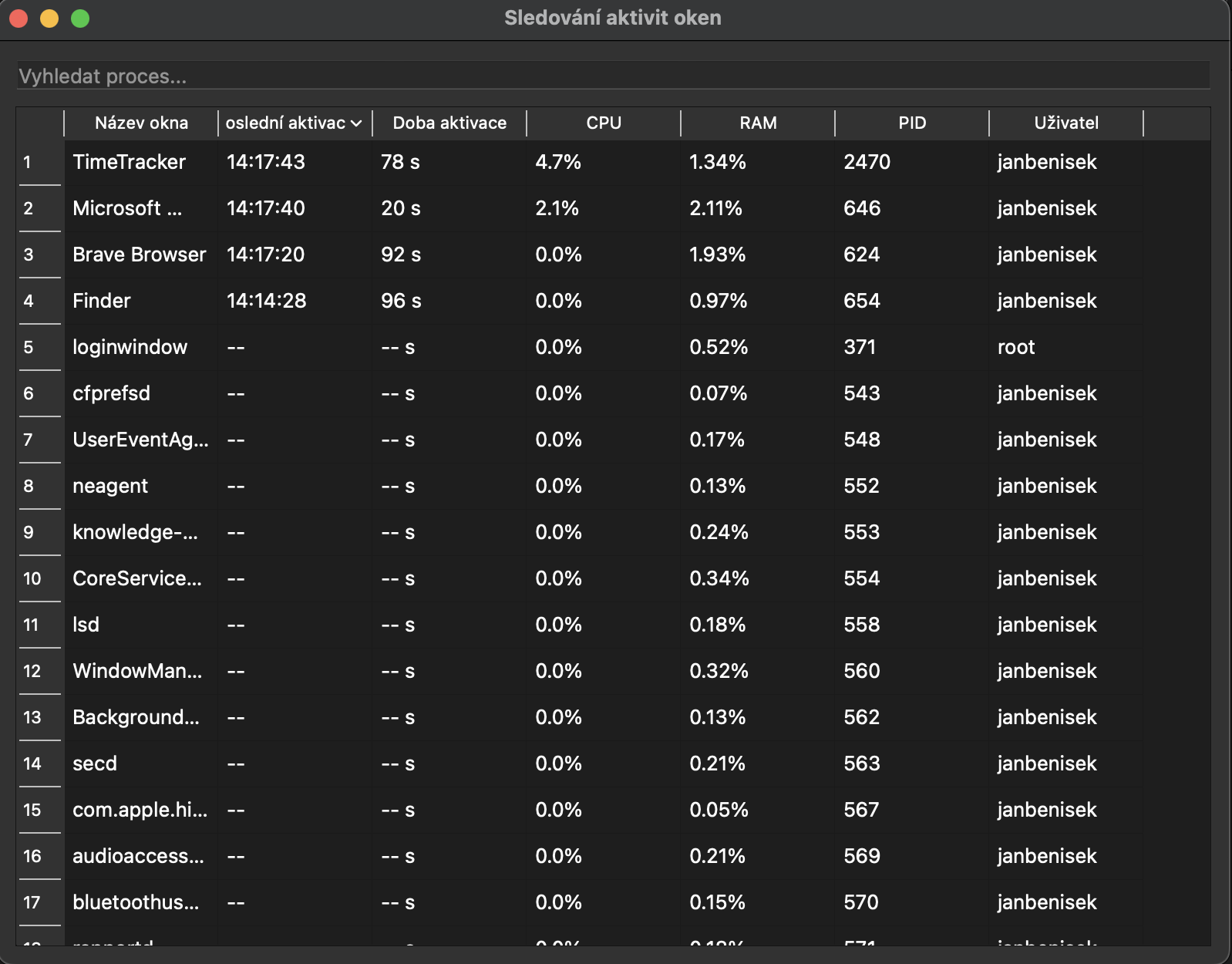
except Exception as e:

print(f"Chyba při pořizování screenshotu: {e}")

## Grafické uživatelské rozhraní (GUI)

Pro GUI je použita knihovna **PyQt5**. Aplikace vytváří hlavní okno s tabulkou, která zobrazuje aktivní procesy a jejich systémové informace. Dále využívá **QSystemTrayIcon** pro zobrazení ikony v systémové liště, kde je k dispozici kontextové menu.

* Tray Ikona a Menu: Aplikace běží na pozadí, a to buď jako minimalizované okno, nebo s aktivním oknem. K dispozici je možnost otevřít hlavní okno nebo aplikaci ukončit.
* Aktualizace Tabulky: Tabulka se automaticky aktualizuje každou sekundu, aby zobrazovala aktuální data o procesech. Řazení dat je povoleno, což umožňuje uživatelskou interakci s tabulkou, jako je třídění podle jména procesu, CPU, RAM nebo času poslední aktivace.



Obrázek : Ukázka GUI

## Nastavení aplikace

Nastavení aplikace lze upravit prostřednictvím dialogového okna v PyQt5. Uživatel může nastavit interval sledování oken a interval pořizování screenshotů. Může také změnit kde se statistikami v souboru csv nebo screenshot uloží.

* **Interval sledování oken:** Určuje, jak často aplikace kontroluje aktivní okna (1–60 sekund).
* **Interval pořizování screenshotů:** Nastavuje, jak často budou screenshoty pořízeny (1–60 minut).
* **Uložení nebo zrušení změn:** Změny se uloží po kliknutí na „Uložit“, nebo lze kliknout na „Zrušit“ pro návrat k původním hodnotám.
* **Změna cesty pro uložení screenshotů a csv:** lze vybrat novou cestu místo defaultního adresáře TMP.

Nastavení jsou aplikována v hlavní logice aplikace.

*# Interval sledování oken*

self.tracking\_label = QLabel("Interval sledování oken (v sekundách):")

self.tracking\_spinbox = QSpinBox()

self.tracking\_spinbox.setRange(1, 60)

self.tracking\_spinbox.setValue(self.tracking\_interval)

layout.addWidget(self.tracking\_label)

layout.addWidget(self.tracking\_spinbox)

*# Interval screenshotů*

self.screenshot\_label = QLabel("Interval screenshotů (v minutách):")

self.screenshot\_spinbox = QSpinBox()

self.screenshot\_spinbox.setRange(1, 60)

self.screenshot\_spinbox.setValue(self.screenshot\_interval)

layout.addWidget(self.screenshot\_label)

layout.addWidget(self.screenshot\_spinbox)

*# Interval logování do CSV*

self.csv\_label = QLabel("Interval logů do CSV (v minutách):")

self.csv\_spinbox = QSpinBox()

self.csv\_spinbox.setRange(1, 60)

self.csv\_spinbox.setValue(self.csv\_interval)

layout.addWidget(self.csv\_label)

layout.addWidget(self.csv\_spinbox)

*# Cesta pro ukládání screenshotů*

self.screenshot\_path\_label = QLabel("Cesta pro ukládání screenshotů:")

self.screenshot\_path\_edit = QLineEdit(self.screenshot\_path)

self.screenshot\_browse\_button = QPushButton("Vybrat...")

self.screenshot\_browse\_button.clicked.connect

(self.browse\_screenshot\_path)

layout.addWidget(self.screenshot\_path\_label)

layout.addWidget(self.screenshot\_path\_edit)

layout.addWidget(self.screenshot\_browse\_button)

*# Cesta pro ukládání CSV logů*

self.csv\_path\_label = QLabel("Cesta pro ukládání CSV logů:")

self.csv\_path\_edit = QLineEdit(self.csv\_path)

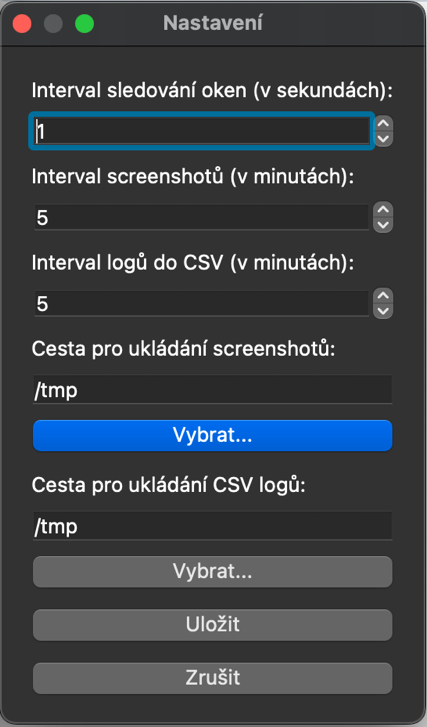
self.csv\_browse\_button = QPushButton("Vybrat...")

self.csv\_browse\_button.clicked.connect(self.browse\_csv\_path)

layout.addWidget(self.csv\_path\_label)

layout.addWidget(self.csv\_path\_edit)

layout.addWidget(self.csv\_browse\_button)



Obrázek : Ukázka nastavení aplikace

## Více vláknové zpracování a plánování úkolů

Aplikace používá QTimer k plánování pravidelných úkolů, jako je aktualizace informací o procesech nebo pořizování screenshotů. Tento přístup umožňuje, aby aplikace běžela hladce na pozadí, aniž by blokovala hlavní vlákno pro interakci s GUI.

*# Nastavení časovačů*

self.update\_timer = QTimer(self)

*# Časovač pro aktualizaci informací o procesech*

self.update\_timer.timeout.connect(self.update\_process\_info)

*# Připojení časovače k metodě aktualizace*

self.update\_timer.start(1000) *# Spuštění časovače každou sekundu*

self.screenshot\_timer = QTimer(self)

*# Časovač pro pořizování screenshotů*

self.screenshot\_timer.timeout.connect(self.take\_screenshot)

*# Připojení k metodě pořizování screenshotů*

self.screenshot\_timer.start(300000)

*# Spuštění každých 5 minut (300 000 ms)*

self.csv\_timer = QTimer(self) *# Časovač pro zápis do CSV*

self.csv\_timer.timeout.connect(self.write\_to\_csv)

*# Připojení k metodě zápisu do CSV*

self.csv\_timer.start(300000) *# Spuštění každých 5 minut (300 000 ms)*

## Zápis do csv souboru

Tato část aplikace umožňuje ukládání a načítání dat o aktivitě uživatele do souboru activity\_log.csv.

* Zápis do CSV: Funkce write\_to\_csv ukládá data o aktivních oknech, přičemž:
* Zapisují se pouze okna s nenulovou dobou aktivity.
* Pokud soubor neexistuje, vytvoří se nový s hlavičkou: „Aplikace“ a „Čas (s)“.
* Čas aktivity se sčítá s existujícími hodnotami a zaokrouhluje na celé sekundy.
* Po zápisu je uživatel informován v konzoli.
* Načítání z CSV: Funkce read\_csv načítá data do slovníku, kde klíčem je název aplikace a hodnotou celkový čas v sekundách. Pokud soubor neexistuje, vrací prázdný slovník.

Obsah obrázku text, snímek obrazovky, Písmo, číslo

Popis byl vytvořen automaticky

Obrázek 3: Ukázka CSV souboru

## Automatické spuštění aplikace při startu systému

Pro umožnění automatického spuštění aplikace TimeTracker při startu systému je využita funkce create\_launch\_agent. Tato funkce vytváří soubor .plist v adresáři

Library/LaunchAgents, který obsahuje konfiguraci pro automatické spuštění aplikace.

* Vytvoření souboru .plist: Funkce kontroluje, zda již existuje soubor s konfigurací pro automatické spuštění. Pokud ne, vytvoří nový soubor s definovanými parametry, jako je cesta k aplikaci a nastavení pro spuštění při načtení systému.
* Dotaz na Uživatelovu Volbu: Pokud soubor .plist neexistuje, je uživateli zobrazen dialog, ve kterém může zvolit, zda chce aplikaci spouštět při startu. Na základě odpovědi uživatele je buď soubor vytvořen, nebo aplikace nebude nastavena na automatické spuštění. Tento proces zajišťuje, že aplikace bude vždy připravena k okamžitému použití po startu systému, pokud uživatel tuto možnost povolí.

Obsah obrázku text, hodiny, snímek obrazovky, multimédia

Popis byl vytvořen automaticky

Obrázek 4: popup okno po prvním spuštění

## Filtrování tabulky

Aplikace umožňuje filtrování tabulky na základě textu zadaného do vyhledávacího pole. Funkce porovnává text zadaný uživatelem se všemi položkami v tabulce, ignoruje velká a malá písmena, a skryje řádky, které neodpovídají zadanému filtru. Tento přístup zajišťuje rychlé a snadné vyhledávání konkrétních záznamů bez nutnosti procházet celou tabulku.

def filter\_table(self): *# Metoda pro filtrování tabulky*

filter\_text = self.search\_box.text().lower()

*# Načte text z vyhledávacího pole a převede na malá písmena*

for row in range(self.table\_widget.rowCount()):

*# Projde všechny řádky tabulky*

match = False *# Inicializuje proměnnou pro kontrolu shody*

for column in range(self.table\_widget.columnCount()):

*# Projde všechny sloupce v aktuálním řádku*

item = self.table\_widget.item(row, column)

*# Načte položku v aktuálním řádku a sloupci*

if item and filter\_text in item.text().lower():

*# Pokud položka existuje a obsahuje text z vyhledávání*

match = True *# Nastaví shodu na True*

break *# Přeruší cyklus, protože shoda byla nalezena*

self.table\_widget.setRowHidden(row, not match)

*# Skryje nebo zobrazí řádek podle výsledku shody*

## Struktura aplikace

TimeTracker/

|

|----gui/ # Složka obsahující skripty pro grafické uživatelské rozhraní.

| |----menu.py # Definice hlavního menu aplikace, které zajišťuje interakci.

| |----settings.py # Skript pro okno nastavení, kde uživatel může upravit konfigurace.

|

|----utils/ # Složka s pomocnými moduly.

| |----activity\_tracker.py # Modul pro sledování aktivních oken.

| |----screenshot.py # Modul pro pořizování screenshotů z aktivních oken.

|

|----main.py # Hlavní skript aplikace, který spouští aplikaci.

|----README.md # Obsahuje popis jejího fungování a návod na instalaci a použití.

|----requirements.txt # Specifikuje potřebné knihovny pro správný běh aplikace.

|----.gitignore # Určuje, které soubory a složky mají být ignorovány při verzování kódu.

# Výsledky řešení, výstupy, uživatelský manuál

## Výsledky řešení

Aplikace byla navržena tak, aby efektivně sledovala aktivní procesy, monitorovala jejich využití systémových zdrojů (CPU, RAM, Čas) a poskytovala uživateli přehledné statistiky. Hlavní funkce zahrnují:

Monitorování aktivních oken a procesů s intervalem nastavitelným uživatelem.

Pořizování screenshotů v pravidelných intervalech.

Vypisování statistik do CSV souboru.

Přehlednou tabulku s možností řazení podle různých parametrů (CPU, RAM, čas aktivace).

## Výstupy Aplikace

Tabulka procesů: Hlavní okno aplikace zobrazuje seznam všech aktivních procesů s informacemi o využití zdrojů a čase aktivace.

Screenshoty: Pořizované snímky obrazovky jsou ukládány do specifikované složky, název obsahuje časovou značku.

Log dat: Aplikace umožňuje export statistik sledování do souboru pro další analýzu.

## Uživatelský manuál

Spuštění aplikace:

Aplikaci spusťte kliknutím na ikonu nebo nastavením pro automatické spuštění při startu systému.

Spuštění aplikace přes vscode:

Pro případ spuštění z konzole a po případné úpravě kódu lze takto:

1.git clone <https://github.com/Benisekjan/TimeTracker> - naklonovaní repozitáře

2. python -m venv .env – založení virtuálního prostředí

3.source .env/bin/activate – aktivace virtuálního prostředí

4.pip install -r requirements.txt – instalace knihoven

5.python3 main.py – spuštění aplikace

Build aplikace: pyinstaller --windowed --name "TimeTracker" --icon=icons/icon.icns --add-data "icons:icons" main.py

Po spuštění se aplikace zobrazí jako ikona v systémové liště.

Používání aplikace

Hlavní okno: Otevřete kliknutím na ikonu v liště a výběrem možnosti „Zobrazit“.

Nastavení: Intervaly pro sledování a screenshoty lze upravit v dialogovém okně „Nastavení“.

Ukončení: Aplikaci ukončíte z kontextového menu ikonky v systémové liště.

Klíčové funkce:

Tabulka procesů: Kliknutím na hlavičku sloupce lze data seřadit.

Screenshoty: Jsou automaticky ukládány na disk v přednastaveném intervalu.

Export dat: Statistiky lze exportovat ve formátu CSV přes hlavní menu.

Tato aplikace umožňuje snadné sledování aktivit v systému a poskytuje užitečné nástroje pro analýzu výkonu.

## Splnění a nesplněné cíle

**Splněné cíle**

Aplikace splňuje klíčové požadavky zadané na začátku projektu:

* **Monitorování procesů a aktivních oken:** Aplikace efektivně sleduje aktivní procesy, zobrazuje jejich systémové využití a umožňuje řazení dat v tabulce.
* **Pořizování screenshotů:** Automatické ukládání screenshotů v nastavených intervalech s možností změny intervalu.
* **Uživatelské nastavení:** Dialogové okno umožňuje upravit intervaly sledování a screenshotů a ukládání do vámi určené složky.
* **Přehledné GUI:** Hlavní okno zobrazuje data v tabulce s možností seřazení podle různých parametrů.
* **Výpis strávených času v aplikacích:** Vypisuje dle určeného intervalu do souboru CSV.
* **Spouštění při startu systému:** Automatické spuštění aplikace při startu systému.

**Nesplněné cíle a další vylepšení**

Během vývoje nebyly implementovány některé rozšířené funkce, které by zvýšily uživatelský komfort a přizpůsobení aplikace:

1. **Detailnější uživatelské nastavení:**
   * Možnost úplného vypnutí pořizování screenshotů.
   * Možnost úplného vypnutí vypisování statistik do CSV souboru.
2. **Pokročilé monitorování:**
   * Možnost výběru aplikací, které budou sledovány.
3. **Rozšíření funkcí:**
   * Zavedení možnosti označit aplikace jako oblíbené („pin“).
   * Ikony aplikací vedle názvu, pokud ikonu mají.
4. **Uložení nastavení při dalším spuštění:**
   * Uložení nastavení ať už screenshotů nebo exportu dat nebo strávený čas v aplikacích momentálně se nastavení ukládá pouze po dobu kdy je aplikace spuštěna.

Tyto funkce představují potenciál pro budoucí verze aplikace, čímž by došlo k jejímu výraznému vylepšení a přizpůsobení širšímu spektru uživatelů.

# Závěr

Cílem projektu bylo sledovat systémové zdroje pomocí jazyka Python.

Aplikace umožnuje uživateli sledovat čas strávený v jednotlivých aplikacích a případně si zkontrolovat co se dělo každých 5 minut díky screenshotům plochy nebo výpisům do CSV a také množství zdrojů jež jednotlivé procesy využívají.

V budoucnu bych chtěl přidat například ukládání nastavení což vidím jako hlavní nesplněný cíl.

Odkaz na github: https://github.com/Benisekjan/TimeTracker/

Seznam použitýCH INFORMAČNÍCH ZDROJů

[1] OpenAI. ChatGPT [online]. Poslední revize 20. 12. 2024 [cit. 2024-12-20]. Dostupné z: <https://chatgpt.com/>.

[2] Qt for Python Documentation [online]. Poslední revize 20. 12. 2024 [cit. 2024-12-20]. Dostupné z: <https://doc.qt.io/qtforpython-6/>.

[3] psutil Documentation [online]. Poslední revize 20. 12. 2024 [cit. 2024-12-20]. Dostupné z: <https://psutil.readthedocs.io/en/latest/#>.

[4] PyAutoGUI Documentation – Screenshot Functionality [online]. Poslední revize 20. 12. 2024 [cit. 2024-12-20]. Dostupné z: [https://pyautogui.readthedocs.io/en/latest/screenshot.html#](https://pyautogui.readthedocs.io/en/latest/screenshot.html).

[5] Apple Inc. AppKit Documentation [online]. Poslední revize 20. 12. 2024 [cit. 2024-12-20]. Dostupné z: <https://developer.apple.com/documentation/appkit>.