

# ZÁVĚREČNÁ STUDIJNÍ PRÁCE

## dokumentace

### Time Tracker

Jan Beníšek



**Obor:** 18-20-M/01 INFORMAČNÍ TECHNOLOGIE  
se zaměřením na počítačové sítě a programování

**Třída:** IT4  
**Školní rok:** 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 31. 12. 2023

---

*podpis autora práce*

## ANOTACE

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 obnovením v 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.

## **OBSAH**

<b>ÚVOD.....</b>	<b>5</b>
<b>1    TEORETICKÁ A METODICKÁ VÝCHODISKA.....</b>	<b>6</b>
1.1    APLIKACE NA SLEDOVÁNÍ AKTIVITY A ZDROJŮ .....	6
1.2    PRINCIP FUNGOVÁNÍ APLIKACE .....	6
<b>2    VYUŽITÉ TECHNOLOGIE .....</b>	<b>7</b>
2.1    PROGRAMOVACÍ JAZYK .....	7
2.2    KNIHOVNA PRO SLEDOVÁNÍ PROCESŮ .....	7
2.3    KNIHOVNA PRO GUI .....	7
2.4    KNIHOVNY PRO SNÍMKY OBRAZOVKY .....	7
2.5    KNIHOVNA PRO UŽIVATELSKÁ NASTAVENÍ .....	7
<b>3    ZPŮSOBY ŘEŠENÍ A POUŽITÉ POSTUPY.....</b>	<b>9</b>
3.1    SLEDOVÁNÍ AKTIVNÍCH OKEN .....	9
3.2    SBĚR SYSTÉMOVÝCH INFORMACÍ.....	9
3.3    POŘÍZENÍ SCREENSHOTŮ .....	10
3.4    GRAFICKÉ UŽIVATELSKÉ ROZHRAŇÍ (GUI) .....	11
3.5    NASTAVENÍ APLIKACE.....	12
3.6    VÍCE VLÁKNOVÉ ZPRACOVÁNÍ A PLÁNOVÁNÍ ÚKOLŮ.....	13
<b>4    VÝSLEDKY ŘEŠENÍ, VÝSTUPY, UŽIVATELSKÝ MANUÁL.....</b>	<b>15</b>
4.1    VÝSLEDKY ŘEŠENÍ .....	15
4.2    VÝSTUPY APLIKACE.....	15
4.3    UŽIVATELSKÝ MANUÁL .....	15
4.4    SPLNĚNÍ A NESPLNĚNÉ CÍLE.....	16
<b>ZÁVĚR .....</b>	<b>17</b>
<b>SEZNAM POUŽITÝCH INFORMAČNÍCH ZDROJŮ .....</b>	<b>18</b>

## ÚVOD

Text úvodu

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.

## 1 TEORETICKÁ A METODICKÁ VÝCHODISKA

### 1.1 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.

### 1.2 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.

## **2 VYUŽITÉ TECHNOLOGIE**

### **2.1 Programovací jazyk**

Pro vývoj aplikace byl zvolen programovací jazyk Python, který je ideální pro práci s operačními 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.

### **2.2 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.

### **2.3 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.

### **2.4 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.

### **2.5 Knihovna pro Uživatelská nastavení**

Pro implementaci uživatelského nastavení byla použita knihovna PyQt5, která umožňuje snadné vytvoření interaktivních dialogových oken. Klíčové vlastnosti využívané 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.



### 3 ZPŮSOBY ŘEŠENÍ A POUŽITÉ POSTUPY

#### 3.1 Sledování Aktivních Oken

Pro sledování aktuálně aktivních oken v systému macOS je využita třída ActivityTracker. Tento modul využívá NSWorkspace a NSWorkspaceDidActivateApplicationNotification, což jsou součástí Cocoa frameworku, který je dostupný pouze na macOS. Pomocí těchto 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 zaznamená čas poslední aktivace a dobu trvání aktivity, což je klíčová funkcionality pro záznam 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
```

#### 3.2 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
                    }

        except (psutil.NoSuchProcess, psutil.AccessDenied, psutil.ZombieProcess):
            continue
```

### 3.3 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

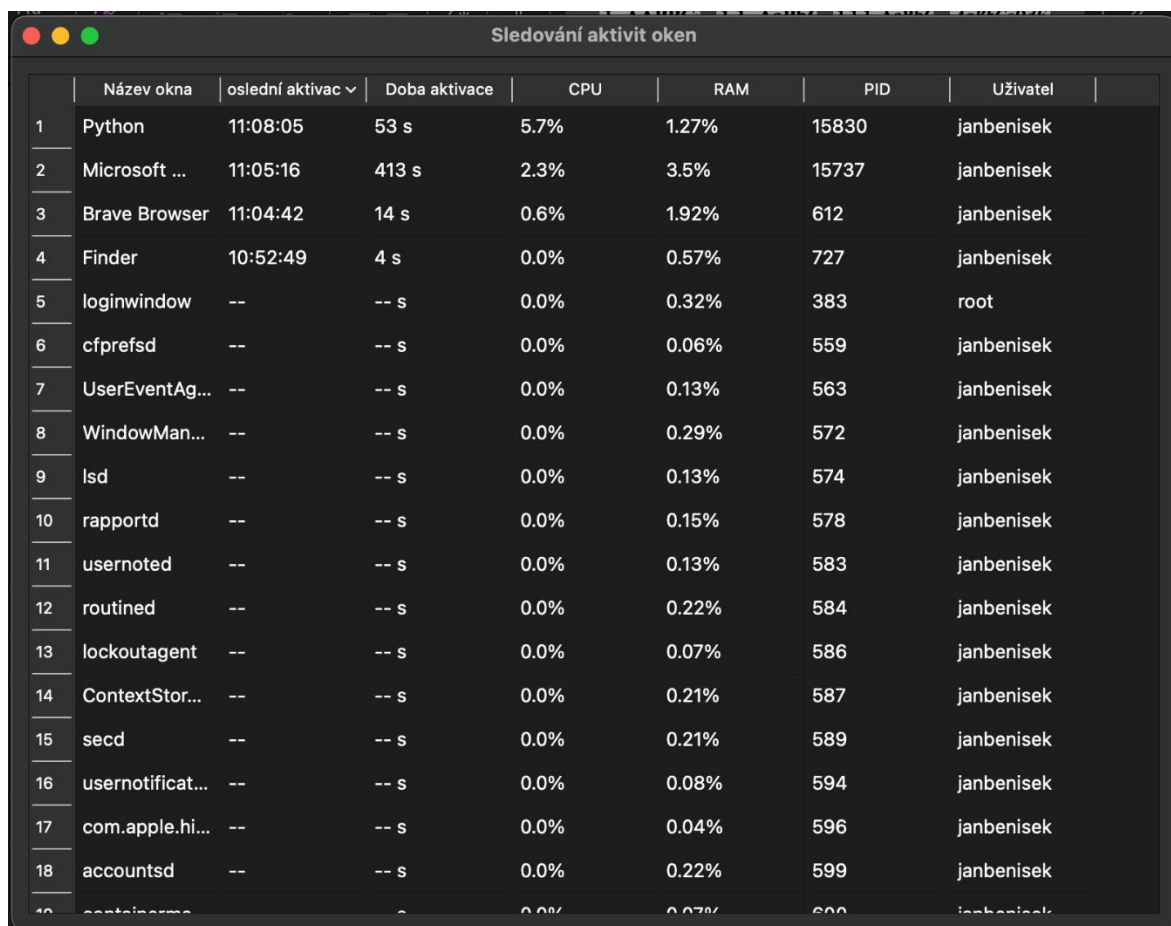
            print(f"Screenshot uložen do: {file_path}")

        except Exception as e:
            print(f"Chyba při pořizování screenshotu: {e}")
```

### 3.4 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.



	Název okna	oslední aktivac	Doba aktivace	CPU	RAM	PID	Uživatel
1	Python	11:08:05	53 s	5.7%	1.27%	15830	janbenisek
2	Microsoft ...	11:05:16	413 s	2.3%	3.5%	15737	janbenisek
3	Brave Browser	11:04:42	14 s	0.6%	1.92%	612	janbenisek
4	Finder	10:52:49	4 s	0.0%	0.57%	727	janbenisek
5	loginwindow	--	-- s	0.0%	0.32%	383	root
6	cfprefsd	--	-- s	0.0%	0.06%	559	janbenisek
7	UserEventAg...	--	-- s	0.0%	0.13%	563	janbenisek
8	WindowMan...	--	-- s	0.0%	0.29%	572	janbenisek
9	lsd	--	-- s	0.0%	0.13%	574	janbenisek
10	rapportd	--	-- s	0.0%	0.15%	578	janbenisek
11	usernoted	--	-- s	0.0%	0.13%	583	janbenisek
12	routined	--	-- s	0.0%	0.22%	584	janbenisek
13	lockoutagent	--	-- s	0.0%	0.07%	586	janbenisek
14	ContextStor...	--	-- s	0.0%	0.21%	587	janbenisek
15	sec	--	-- s	0.0%	0.21%	589	janbenisek
16	usernotificat...	--	-- s	0.0%	0.08%	594	janbenisek
17	com.apple.hi...	--	-- s	0.0%	0.04%	596	janbenisek
18	accounts	--	-- s	0.0%	0.22%	599	janbenisek
19	containerd	--	-- s	0.0%	0.07%	600	janbenisek

### 3.5 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 pořizovány (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)

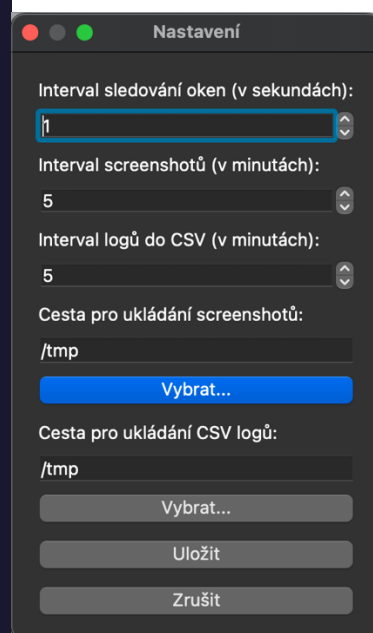
# Interval screenshotů
self.screenshot_label = QLabel("Interval screenshotů (v minutách):")
self.screenshot_spinbox = QSpinBox()
self.screenshot_spinbox.setRange(1, 60)
self.screenshot_spinbox.setValue(self.screenshot_interval)
layout.addWidget(self.screenshot_label)
layout.addWidget(self.screenshot_spinbox)

# 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)

```



### 3.6 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)

```

### 3.7 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 Uživatelskou 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.

```
# Funkce pro vytvoření souboru .plist pro automatické spuštění
def create_launch_agent():
    plist_path = Path.home() / 'Library/LaunchAgents/com.timetracker.startup.plist'

    # Cesta k aplikaci
    app_path = os.path.join(os.path.dirname(__file__), 'TimeTracker.app/Contents/MacOS/TimeTracker')

    if not plist_path.exists():
        plist = {
            'Label': 'com.timetracker.startup',
            'ProgramArguments': [
                app_path,
            ],
            'RunAtLoad': True,
        }
        with plist_path.open('wb') as plist_file:
            plistlib.dump(plist, plist_file)
        print("TimeTracker bude spuštěn při startu systému.")
    else:
        print("You, před 11 hodinami • Automatické spuštění při startu")
        print("TimeTracker již byl nastaven na spuštění při startu.")
```

### 3.8 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.
```

## 4 VÝSLEDKY ŘEŠENÍ, VÝSTUPY, UŽIVATELSKÝ MANUÁL

### 4.1 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).

### 4.2 Výstupy Aplikace

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

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

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

### 4.3 Uživatelský Manuál

Spuštění Aplikace:

Aplikaci spustíte 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> - Naklonování 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 screenshots 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.

Screenshots: Jsou automaticky ukládány na disk v přednastaveném intervalu.

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

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

## 4.4 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
- **Spuštění 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 pochopit více jazyk python a sledováním systémových zdrojů také více operační systém

Aplikace umožňuje 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

Je hodně věcí, co bych chtěl v budoucnu přidat jako 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] ponty. pyscreenshot – Python Screenshot Module [online]. Poslední revize 20. 12. 2024 [cit. 2024-12-20]. Dostupné z: <<https://github.com/ponty/pyscreenshot/tree/3.1>>.
- [5] Apple Inc. AppKit Documentation [online]. Poslední revize 20. 12. 2024 [cit. 2024-12-20]. Dostupné z: <<https://developer.apple.com/documentation/appkit>>.