FILESYNCER – Dokumentace projektu

Autor: Plichta
Kontaktní údaje: plichtavojta@gmail.com
Škola: SPŠE Ječná
Studijní obor: Informační technologie
Datum vypracování: 24.11.2025
Poznámka: Jedná se o školní projekt určený k demonstraci práce s více vlákny a synchronizací souborů v Pythonu.

1. Specifikace požadavků

1.1 Business Requirements

Aplikace má umožnit automatickou synchronizaci vybrané složky do cílové zálohovací složky. Uživatel očekává, že:

- Při vytvoření souboru ve *source* se automaticky objeví v *backup*.
- Při úpravě souboru ve *source* se aktualizuje i soubor v *backup*.
- Při smazání souboru ve *source* bude odstraněn i z *backup*.
- Synchronizace má probíhat automaticky bez zásahu uživatele.
- Aplikace má být spustitelná bez složité instalace.

1.2 Funkční požadavky

- Sledování změn ve složce *source_folder*.
- Detekce vytvoření, modifikace a smazání souboru.
- Převod událostí na úkoly (tasks).
- Paralelní zpracování úkolů pomocí více vláken.
- Kopírování souborů, mazání souborů v cílové složce.
- Ukládání záznamů o běhu aplikace do logu.
- Nakládání s ignorovanými soubory podle configu.

1.3 Nefunkční požadavky

- Aplikace musí běžet na Windows 10/11.
- Musí být použit Python 3.
- Program musí být spustitelný příkazem `run.cmd`.
- Zpracování I/O operací musí být bezpečné z hlediska více vláken.
- Logování musí být konzistentní, thread-safe a čitelné.

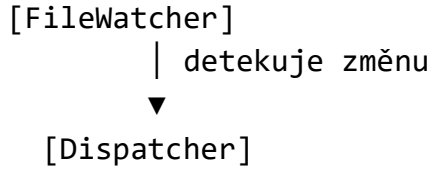
2. Architektura aplikace

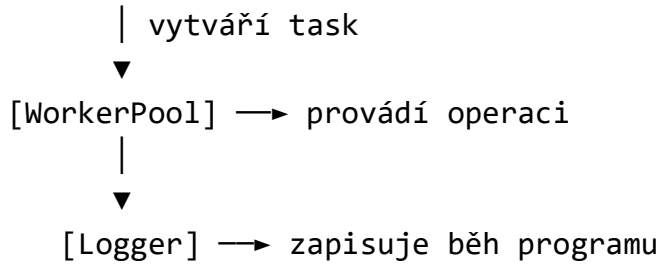
Aplikace je postavená na modularizované architektuře, kde každá část odpovídá za vlastní odpovědnost.

Hlavní komponenty:

- **SyncService**
  Centralizovaný řídicí modul, který propojuje jednotlivé části aplikace.
- **FileWatcher**
  Periodicky prochází zdrojovou složku a zjišťuje změny.
- **Dispatcher**
  Překládá změny (events) na úkoly (tasks) pro pracovní vlákna.
- **WorkerPool**
  Sada vláken, která paralelně zpracovávají kopírovací/smazovací operace.
- **Logger**
  Zajišťuje bezpečné logování událostí napříč vlákny.

Komunikace komponent





Aplikace využívá **návrhový vzor Producer-Consumer**, kde:

- Watcher = PRODUCER událostí
- Dispatcher = PRODUCER úkolů
- WorkerPool = CONSUMER úkolů

3. Běh aplikace

3.1 Use-case: vytvoření souboru

1. Uživatel uloží nový soubor do *source/*.
2. Watcher si všimne, že soubor je nový.
3. Zavolá `dispatcher.on_created(path)`.
4. Dispatcher vygeneruje task typu `copy`.
5. WorkerPool ho zpracuje (pomocí dostupného vlákna).
6. Logger zaznamená průběh.

3.2 Use-case: úprava souboru

- Probíhá stejně jako vytvoření, ale dispatcher volá `on_modified`.

3.3 Use-case: smazání souboru

- Watcher detekuje odstranění.
- Dispatcher vytvoří `delete` task.
- Worker file odstraní v cílové cestě.

3.4 Activity Diagram (textový popis)

```
Start
 │
 ▼
Watcher skenuje složku
 │
 ├── nový soubor → on_created → enqueue(copy)
 ├── změněný soubor → on_modified → enqueue(copy)
 └── smazaný soubor → on_deleted → enqueue(delete)
 │
 ▼
WorkerPool zpracuje task
 │
 ▼
Logger uloží výsledek
 │
 ▼
Opakování cyklu
```

4. Použité technologie, knihovny a protokoly

Aplikace je založena pouze na Python standardní knihovně:

- **threading** – vytvoření worker vláken
- **queue** – thread-safe fronta úkolů
- **os / shutil** – práce se soubory
- **hashlib** – volitelná kontrola integrity pomocí SHA-256
- **json** – načítání konfigurace
- **datetime** – timestampy v logu

Žádné externí knihovny nejsou použity.

5. Právní aspekty

Tento projekt je vytvořen jako **školní nekomerční práce**.
Není určena pro komerční použití.
Kód nepoužívá žádné licencované komponenty třetích stran.

6. Konfigurace

Aplikace používá soubor `config.json` v rootu projektu.

Příklad konfigurace:

```
{
    "source_folder": "source",
    "target_folder": "backup",
    "num_workers": 3,
    "log_file": "sync.log",
    "hash_verify": false,
    "ignore_patterns": []
}
```

Význam voleb:

- **source_folder** – sledovaná složka
- **target_folder** – zálohovací složka
- **num_workers** – počet vláken
- **log_file** – název log souboru
- **hash_verify** – kontrola integrity zkopírovaných dat
- **ignore_patterns** – seznam ignorovaných typů souborů (glob patterny)

7. Instalace a spuštění

1) Vyžadováno

- Windows 10/11
- Python 3.x
- Žádné externí knihovny

2) Spuštění aplikace

V PowerShellu nebo CMD:

`bin\run.cmd`

Tento skript:

- nastaví pracovní adresář
- spustí `src/main.py` pomocí Python launcheru
- nechá okno otevřené kvůli logům

8. Chybové stavy

Chybové hlášky, které mohou nastat:

Chyba	Význam
FileNotFoundError	Chybějící konfigurace nebo soubor
Hash mismatch	Kopírování proběhlo, ale obsah se liší
Worker error	Neošetřená výjimka uvnitř workeru
Permission denied	Zákaz přístupu k souboru
Invalid path	Nesprávně nastavené cesty v configu

Aplikace chyby zapisuje do logu + vypíše je na konzoli.

9. Testování a validace

Provedené testy:

- **Vytvoření souboru v source** → správná kopie v backup
- **Úprava souboru** → aktualizovaný soubor v backup
- **Smazání souboru** → odstranění v backup
- **Vytvoření podsložek** → automatické vytvoření i v backup
- **Více vláken** → paralelní kopírování funguje
- **Hash verify ON** → kontrola integrity proběhla správně
- **Ignorované soubory** → nebyly kopírovány

Výsledek:
✓ aplikace splňuje stanovené požadavky.

10. Verze a známé problémy

Verze: 1.0
Známé omezení:

- Watcher používá periodický scan (není event-based).
- Dispatcher zatím neřeší konflikty změn.
- Hashing je volitelný, ale pomalý u velkých souborů.
- Neexistuje GUI ani CLI argumenty.

Závěr

Projekt úspěšně demonstruje:

- práci s více vlákny
- synchronizaci souborů
- modularizaci kódu
- práci se vstupně-výstupními operacemi
- využití front a návrhového vzoru Producer–Consumer

Aplikace je funkční, rozšiřitelná a přehledná.