Univerzita Pardubice

Fakulta ekonomicko-správní

Vytvoření aplikace pro ukládání big dat v Pythonu

Diplomová práce

2025 Bc. Tomáš Fráňa

Univerzita Pardubice
Fakulta ekonomicko-správní
Akademický rok: 2024/2025

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Tomáš Fráňa**
Osobní číslo:	**E23040**
Studijní program:	**N0613A140041 Aplikovaná informatika – Data Science pro business**
Téma práce:	**Vytvoření aplikace pro ukládání big dat v Pythonu**
Zadávající katedra:	**Ústav systémového inženýrství a informatiky**

Zásady pro vypracování

Cílem práce je charakterizovat současné přístupy k ukládání big dat, porovnat existující nástroje a metody, provést sběr a předzpracování dat, navrhnout modelové postupy, a nakonec implementovat celé řešení prostřednictvím aplikace v Pythonu.

Osnova:

- Vymezení, přehled a porovnání nástrojů a metod.
- Příprava, sběr a předzpracování dat.
- Návrh modelových postupů.
- Implementace aplikace v Pythonu.
- Vyhodnocení výsledků a diskuse.

Rozsah pracovní zprávy: **cca 50 stran**
Rozsah grafických prací:
Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

CIELEN, Davy; MEYSMAN, Arno D. B.; ALI, Mohamed. *Introducing Data Science: Big data, machine learning, and more, using Python tools*. Shelter Island: Manning, 2016. ISBN 9781633430037.
GANDOMI, Amir; HAIDER, Murtaza. Beyond the hype: Big data concepts, methods, and analytics. *International Journal of Information Management*, 2015, 35.2: 137-144.
HENDL, Jan. *Big data: věda o datech – základy a aplikace*. Praha: Grada Publishing, 2021. Průvodce. ISBN 978-80-271-3031-3.
JIN, Xiaolong, et al. Significance and challenges of big data research. *Big Data Research*, 2015, 2.2: 59-64.
JOHNSON, Jeff S.; FRIEND, Scott B.; LEE, Hannah S. Big data facilitation, utilization, and monetization: Exploring the 3Vs in a new product development process. *Journal of Product Innovation Management*, 2017, 34.5: 640-658.
MCKINNEY, Wes. *Python for data analysis: data wrangling with Pandas, NumPy, and IPython*. Second edition. Sebastopol: O'Reilly, 2018. ISBN 978-1-491-95766-0.

Vedoucí diplomové práce: **RNDr. Ing. Oldřich Horák, Ph.D.**
Ústav systémového inženýrství a informatiky

Datum zadání diplomové práce: **1. září 2024**
Termín odevzdání diplomové práce: **30. dubna 2025**

L.S.

prof. Ing. Jan Stejskal, Ph.D. v.r.
děkan

prof. Ing. Petr Hájek, Ph.D. v.r.
garant studijního programu

V Pardubicích dne 1. září 2024

Prohlášení:

Prohlašuji:

Práci s názvem Vytvoření aplikace pro ukládání big dat v Pythonu 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 30.04.2025

Bc. Tomáš Fráňa

PODĚKOVÁNÍ

Můj dík patří všem kantorům, kteří mi jakoukoli mírou pomáhali při vypracovávání této práce. I přes všechna úskalí, která na této cestě nastala jsem rád, že jsem se mohl spolehnout na jejich pomoc. Jsem velice vděčný za doporučení opravdu kvalitní odborné literatury, kterou bych mohl použít i při jakékoli práci na podobné téma a velmi mi pomohla při zpracování rešerší a pochopení základních problémů a principů v mojí práci obsažených. Velice si vážím také všech přátel, známých a kolegů, kteří mi jakkoli pomohli nasměrovat práci směrem, kterým se vydala, a díky nim je praktický přínos mé práce podstatně větší, než jsem na začátku očekával.

ANOTACE

Práce se zaměřuje na současné přístupy k ukládání big dat a porovnání existujících nástrojů a metod. Hlavní část se věnuje sběru a předzpracování dat, návrhu modelových postupů a implementaci komplexního řešení formou aplikace v jazyce Python. Klíčovým přínosem je návrh a realizace uživatelsky přívětivé aplikace využívající otevřené technologie a knihovny Pythonu, která poskytuje dostupné a snadno využitelné řešení pro jednotlivce a malé či střední podniky, jež nepotřebují nebo si nemohou dovolit robustní big data platformy.

KLÍČOVÁ SLOVA

Big data, Python, ukládání dat, předzpracování dat, JSON, NoSQL

TITLE

Developing an application for big data storage in Python

ANNOTATION

The thesis focuses on current approaches to big data storage and the comparison of existing tools and methods. The main part is dedicated to data collection and preprocessing, the design of model procedures, and the implementation of a comprehensive solution in the form of a Python-based application. The key contribution of the thesis lies in the design and development of a user-friendly application utilizing open technologies and Python libraries, providing an accessible and easily applicable solution for individuals and small to medium-sized enterprises that do not require or cannot afford robust big data platforms.

KEYWORDS

Big data, Python, data storage, data preprocessing, JSON, NoSQL

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM ZKRATEK A ZNAČEK

API	Application Programming Interface
BPMN	Business Process Model and Notation
CSV	Comma-Separated Values
GFS	Google File System
GUI	Graphical User Interface
HDFS	Hadoop Distributed File System
IoT	Internet of Things
JSON	JavaScript Object Notation
NLP	Natural Language Processing
PDF	Portable Document Format
QFS	Quantcast File System
SQL	Structured Query Language
TSV	Tab-Separated Values
XML	Extensible Markup Language
YAML	Yet Another Markup Language
YARN	Yet Another Resource Negotiator

ÚVOD

V posledních letech se problematika ukládání velkých objemů různorodých dat (big data) stává stále aktuálnější. S rostoucím množstvím dat, která jsou generována v různých oblastech, jako jsou sociální sítě, senzory, obchodní transakce a mnohé další, se zvyšuje potřeba efektivních přístupů pro jejich správu. Jedním z hlavních problémů při hledání a návrhu vhodných přístupů a řešení jsou různé charakteristiky (vlastnosti) big dat. Kromě velkých objemů těchto dat, která jsou navíc často uložena v nepropojených a nespolupracujících systémech, je nutné řešit také různorodost dat co se týká jejich formátů, struktury a zdrojů, což může výrazně komplikovat jejich integraci, analýzu a využívání pro rozhodovací procesy.

Jedním z klíčových problémů při práci s big daty je proto jejich předzpracování. Big data jsou zpravidla dostupná v různých nestrukturovaných nebo polostrukturovaných formátech, které je nutné upravit do takové podoby, aby byly vhodné pro následnou analýzu. Tento proces zahrnuje různé kroky, jako je čištění dat, jejich transformace a optimalizace pro ukládání. Ačkoliv lze nalézt velké množství platforem, nástrojů a služeb, které lze k těmto činnostem využít, tak stále chybí jednoduchá řešení využívající otevřené technologie a přístupy.

Cílem této práce je charakterizovat současné přístupy k ukládání big dat, porovnat existující nástroje a metody, provést sběr a předzpracování dat, navrhnout modelové postupy, a nakonec implementovat celé řešení prostřednictvím aplikace v Pythonu. Hlavní motivací tohoto přístupu je, aby výsledná aplikace využívala otevřené technologie a vhodné knihovny programovacího jazyka Python, a zároveň nabídla snadno ovladatelné uživatelské rozhraní. Navržená aplikace by měla sloužit jak jednotlivcům, tak malým a středním podnikům, které nepotřebují komplexní a robustní platformy, nástroje a služby pro správu big dat, nebo si je nemohou finančně dovolit.

První kapitola této práce se věnuje vymezení základních pojmů se zaměřením na jejich vývoj a propojení. Druhá kapitola se zabývá přehledem a porovnáním nástrojů a metod pro ukládání a předzpracování big dat. Jsou zde prozkoumány jejich výhody a nevýhody a jejich praktické využití. Důraz je kladen na porovnání vybraných komerčních nástrojů a otevřených řešení vhodných pro ukládání dat. Následuje kapitola věnující se metodice práce, ve které je vizuálně znázorněna posloupnost kroků vedoucích ke splnění cíle práce. Další kapitola práce se zabývá zdroji dat a přístupy, které jsou používány pro přípravu, sběr a předzpracování big dat. Je zde řešeno, jak jsou tato data předzpracována a jakým způsobem je lze efektivně uložit. Na základě těchto poznatků jsou poté navrženy modelové postupy, které jsou implementovány v aplikaci

vytvořené v Pythonu. Tato aplikace je navržena tak, aby efektivně předzpracovávala a ukládala velké datové soubory, přičemž k tomu využívá formát JavaScript Object Notation (JSON). JSON je nativně podporován mnoha NoSQL databázemi a jinými systémy, které jsou běžně používány pro ukládání a správu big dat.

V závěru práce jsou zhodnoceny celkové přínosy vytvořených modelů a aplikace. Je provedena analýza jejich efektivity a navrženy možnosti pro jejich další vylepšení. Dále jsou diskutovány také budoucí vývoj v oblasti předzpracování a ukládání big dat a představena možná rozšíření funkcionalit aplikace pro další praktické využití. Výsledky této práce mohou být využity jak jednotlivci, tak malými nebo středními podniky, které poptávají otevřená řešení pro efektivní ukládání big dat, tak zároveň mohu sloužit jako základ pro další výzkum a vývoj v oblasti big dat a jejich efektivního předzpracování a ukládání.

1 VYMEZENÍ ZÁKLADNÍCH POJMŮ

V této kapitole jsou vymezeny základní pojmy týkající se řešené problematiky.

1.1 Big data

Big data představují rozsáhlé a komplexní datové sady, které svou velikostí, rychlostí generování a strukturální různorodostí přesahují možnosti tradičních databázových systémů a vyžadují nové přístupy k jejich ukládání a celkově práci s nimi (Hendl, 2021). Tyto datové sady vznikají v reálném čase nebo téměř v reálném čase a pocházejí z různých zdrojů, jako jsou sociální sítě, chytré senzory, mobilní zařízení, online transakce, vědecké simulace či internet věcí (Internet of Things, IoT). Klíčovým rysem big data je jejich různorodost, která zahrnuje strukturovaná data (např. databázové záznamy), nestrukturovaná data (texty, obrázky, videa) a polostrukturovaná data jako JSON nebo Extensible Markup Language (XML). K popisu big dat se využívá několik klíčových vlastností. Původně se hovořilo o tzv. 3V konceptu, který zahrnuje Volume, Velocity a Variety (Cielen et al., 2016; Hendl, 2021). Gandomi a Haider (2015) ve své práci pracují s charakteristikami big dat známými jako 5V (Volume, Velocity, Variety, Veracity, Value). Dnes těchto konceptů založených na popisu pomocí „V“ vlastností existuje množství, nicméně v praxi se zpravidla pracuje s 5V (Saeed a Husamaldin, 2021).

Mezi základní vlastnosti se řadí (Hendl, 2021; Gandomi a Haider, 2015; Saeed a Husamaldin, 2021):

- **Objem** (Volume) odkazuje na enormní množství dat, které vyžaduje specializované úložné technologie a datové modely.
- **Rychlost** (Velocity) zdůrazňuje potřebu zpracování dat téměř v reálném čase, aby byla zachována jejich relevance a užitečnost.
- **Různorodost** (Variety) reflektuje rozmanitost datových formátů – od strukturovaných databázových záznamů po nestrukturovaný obsah, jako jsou texty, obrázky či videa, což komplikuje jejich integraci a analýzu.
- **Pravdivost** (Veracity) se zaměřuje na kvalitu a spolehlivost dat, přičemž roste potřeba validace a čištění, aby bylo možné eliminovat nepřesnosti a šum.
- **Hodnota** (Value) vyjadřuje potenciální přínos dat, jehož realizace závisí na efektivitě použitých metod zpracování a analýzy.

1.2 Typy dat

V rámci big dat se rozlišují takzvané „V“ charakteristiky mezi něž patří i „Variety“, tedy různorodost. Různorodost dat tedy poukazuje na to, že data pocházejí z více zdrojů nebo mají jiný typ, případně formát. Logicky posléze nastává problém se zpracováním takovýchto dat, protože k nim nelze přistupovat zcela stejně. Například data ze zdrojů jako jsou záznamy porad, anebo zasedání dozorčích rad budou naprosto odlišné od dat získávaných z firemní komunikace, kterou reprezentují například emaily. Pro správné předzpracování a způsob uložení big dat je tedy nutné chápat jednotlivé typy dat (Cielen et al., 2016; Hendl, 2021; Khan et al., 2014).

Strukturovaná data jsou organizována do pevně definovaného formátu, což umožňuje snadné třídění, ukládání a vyhledávání. Nejčastěji jsou tato data uložena v relačních databázích, kde mají podobu tabulek obsahujících řádky a sloupce s přesně definovanými atributy. Tento formát umožňuje efektivní správu a analýzu pomocí Structured Query Language (SQL) dotazů, které poskytují rychlý přístup k požadovaným informacím (Cielen et al., 2016; Hendl, 2021).

Nestrukturovaná data nemají pevně definovanou strukturu, což komplikuje jejich třídění, vyhledávání a analýzu. Do této kategorie spadají formáty jako textové dokumenty, e-maily, obrázky, videa a zvukové nahrávky. Jelikož tato data nelze jednoduše uložit do tradičních databází, vyžadují pokročilé analytické metody, jako je strojové učení, aby bylo možné z nich extrahovat relevantní informace (Cielen et al., 2016). Přestože jejich analýza je náročná, nestrukturovaná data představují významný zdroj poznatků, zejména v oblastech jako sociální sítě, zákaznické recenze či multimediální obsah, kde jsou tradiční metody neefektivní.

Polostrukturovaná data se nacházejí na pomezí mezi strukturovanými a nestrukturovanými daty. Obsahují určité organizované prvky, ale jejich struktura není pevně dána jako u relačních databází. Typickými příklady jsou XML a JSON, které uchovávají data v hierarchické, samo-popisující podobě (Buneman, 1997). Tyto formáty jsou často používány pro přenos dat mezi systémy a Application Programming Interface (API), protože umožňují snadnou integraci mezi různými platformami. JSON je obzvláště populární díky své jednoduché syntaxi a efektivitě, která snižuje množství přenášených dat oproti XML (Hendl, 2021; Pezoa, 2016).

Přirozený jazyk je specifickou kategorií nestrukturovaných dat, která zahrnuje texty psané v lidském jazyce. Jeho analýza spadá do oblasti zpracování přirozeného jazyka (Natural Language Processing, NLP), která využívá metody strojového učení k interpretaci významu textu. NLP zahrnuje techniky jako analýza sentimentu, rozpoznávání entit nebo strojový

překlad, které umožňují automatizované zpracování obrovského množství textových dat (Cielen et al., 2016). Přirozený jazyk se objevuje například v sociálních médiích, zákaznických recenzích či právních dokumentech, kde jeho analýza pomáhá identifikovat trendy, vzorce chování a klíčová témata.

Audiovizuální data zahrnují obrazové a zvukové soubory, například fotografie, videa, hlasové nahrávky či hudbu. Tato data jsou nestrukturovaná a jejich analýza vyžaduje pokročilé metody zpracování, jako je rozpoznávání obrazu a zvuku. V praxi se využívají v bezpečnostních systémech pro rozpoznávání tváří, v analýze chování zákazníků či v automatizovaném titulkování videí (Cielen et al., 2016; Hendl, 2021). S rostoucím objemem multimediálního obsahu na internetu nabývá zpracování audiovizuálních dat na stále větším významu, přičemž technologie jako hluboké učení umožňují stále přesnější analýzu obrazu i zvuku.

Strojově vytvořená data jsou generována automaticky bez lidského zásahu, často prostřednictvím senzorů, logovacích systémů nebo IoT zařízení. Tato data mohou být strukturovaná i nestrukturovaná a obvykle vznikají ve velkých objemech a vysoké rychlosti. Typickými příklady jsou záznamy z průmyslových senzorů monitorujících výkon strojů, telemetrická data z vozidel nebo logy serverů sledující aktivitu uživatelů (Cielen et al., 2016). Strojově vytvořená data hrají zásadní roli v automatizaci, prediktivní analýze a optimalizaci provozu, zejména v odvětvích jako je průmyslová výroba, chytrá města a IoT.

1.3 Ukládání big dat

Tradiční relační databázové systémy často nejsou dostatečně flexibilní pro práci s big daty, což vedlo k rozvoji NoSQL databází, cloudových úložišť a distribuovaných souborových systémů, jako jsou Hadoop Distributed File System (HDFS) (Saeed a Husamaldin, 2021; Wiese, 2015). Proto při ukládání big dat hraje klíčovou roli efektivní správa a organizace datových úložišť, což umožňuje nejen škálovatelnost, ale také optimalizaci výkonu analytických operací. Upadhyay et al. (2021) zdůrazňují význam moderních úložných infrastruktur, které dokážou zpracovávat rozsáhlé objemy dat a zároveň zajistit jejich dostupnost a integritu. Hybridní přístupy, kombinující on-premise infrastrukturu s cloudovými službami, poskytují organizacím větší kontrolu nad daty a zároveň využívají výhod cloudové flexibility (Strohbach et al., 2016). Důležitou roli hraje také edge computing, který umožňuje zpracování dat blíže k jejich zdroji, čímž snižuje latenci a zvyšuje efektivitu centrálních úložišť (Khan et al., 2014; Wiese, 2015).

Navzdory technologickému pokroku čelí ukládání big dat řadě výzev. Klíčovým problémem je efektivní organizace a indexování dat, které umožňuje rychlé vyhledávání a analýzu big dat (Naeem et al., 2022). Dalším zásadním aspektem je bezpečnost a ochrana citlivých dat, jelikož s rostoucím množstvím uchovávaných informací se zvyšuje i riziko kybernetických hrozeb a neoprávněného přístupu (Johnson et al., 2017). Upadhyay et al. (2021) zdůrazňují, že správně implementovaná infrastruktura pro ukládání big dat nejen zlepšuje zabezpečení, ale umožňuje také automatizované zálohování a obnovu dat, čímž minimalizuje riziko ztráty kritických dat. Kromě bezpečnosti je významnou výzvou také nákladová efektivita, zejména v souvislosti s dlouhodobou archivací a správou životního cyklu dat (Strohbach et al., 2016). Přesto však vhodně zvolené technologie a strategie ukládání umožňují organizacím efektivněji využívat analytické nástroje, zlepšovat rozhodovací procesy a vytvářet nové obchodní příležitosti (Jin et al., 2015).

1.4 Python

Python je vysokoúrovňový, interpretovaný a objektově orientovaný programovací jazyk, který je široce využíván pro vývoj softwaru, vědecké výpočty, analýzu dat a automatizaci úloh. Je známý svou jednoduchou a čitelnou syntaxí, což jej činí oblíbeným jak mezi začátečníky, tak mezi zkušenými vývojáři. Díky rozsáhlé standardní knihovně a podpoře velkého množství rozšíření je vhodný pro různé oblasti, včetně webového vývoje, umělé inteligence, strojového učení a práce s databázemi. Python hraje důležitou roli v oblasti ukládání big dat, a to díky své flexibilitě, bohaté knihovně nástrojů a podpoře distribuovaných systémů. Umožňuje efektivní přístup k databázím, správu souborových systémů a integraci s cloudovými platformami (Cielen et al., 2016; McKinney, 2017).

Cielen et al. (2016) zdůrazňují, že efektivní ukládání big dat vyžaduje formáty a optimalizované databázové systémy, které umožňují rychlé načítání, zápis a analýzu dat. Python podporuje jak relační databáze (např. PostgreSQL, MySQL), tak i NoSQL databáze (např. MongoDB, Apache Cassandra), které jsou vhodné pro práci s nestrukturovanými a polostrukturovanými daty. McKinney (2017) klade důraz na význam sloupcově orientovaných formátů, jako je Parquet, které umožňují efektivní ukládání datových souborů s vysokou kompresí a optimalizovaným dotazováním. Pro ukládání velkých objemů dat v binární podobě Python nabízí podporu pro HDF5, který umožňuje efektivní ukládání hierarchicky strukturovaných datových souborů a jejich rychlé načítání při analýze (McKinney, 2017).

2 PŘEHLED A POROVNÁNÍ NÁSTROJŮ A METOD

Kapitola se zaměřuje na přehled a porovnání nástrojů a metod pro ukládání big dat.

2.1 Datová centra a úložiště

V současnosti dochází k exponenciálnímu nárůstu objemu digitálních dat, což klade významné nároky na infrastrukturu pro jejich předzpracování a ukládání. Velké technologické společnosti, jako je Google, shromažďují obrovské množství informací a využívají pokročilé algoritmy pro jejich analýzu. Nicméně i přes vysoký výpočetní výkon moderních systémů začínají narážet na limity zpracování dat, jejichž objem přesahuje praktické možnosti ukládání a analýzy (Gilder, 2021). Velkokapacitní datová centra využívaná těmito korporacemi jsou propojena rozsáhlou sítí optických kabelů a jejich energetická náročnost představuje jedno z klíčových ekologických a ekonomických témat současnosti (Gao et al., 2023; Wu et al., 2016). Spotřeba energie těchto zařízení již konkuruje tradičním průmyslovým sektorům a náklady na provoz datových center významně ovlivňují cenu správy a uchovávání dat (Ahmed et al., 2021; Gilder, 2021).

V tomto kontextu se stává stále naléhavějším hledání efektivnějších způsobů ukládání a správy dat, zejména s využitím otevřených technologií, které by umožnily redukci objemu ukládaných dat při zachování jejich integrity a dostupnosti.

2.2 Souborové systémy a formáty

Souborové systémy a formáty hrají klíčovou roli při ukládání a práci s big daty, přičemž jejich výběr závisí na specifických požadavcích aplikací, jako je škálovatelnost a efektivita ukládání. Ploché soubory (Flat Files) představují nejjednodušší formu ukládání dat bez vnitřní hierarchie – typicky ve formátech Comma-Separated Values (CSV) a Tab-Separated Values (TSV). Tyto formáty jsou široce využívány v databázových systémech a datových úložištích, ale jejich nevýhodou je omezená flexibilita při reprezentaci složitějších vztahů mezi daty (Mehmood et al., 2019). Pro strukturovanější ukládání se využívají polostrukturované soubory, jako JSON, XML nebo Yet Another Markup Language (YAML), které umožňují hierarchickou organizaci dat a efektivní výměnu informací mezi aplikacemi a API (Buneman, 1997; Pezoa, 2016).

V prostředí big dat se využívají distribuované souborové systémy, které umožňují zpracování obrovských objemů dat rozložených napříč více uzly v clusterech. Tyto systémy jsou navrženy tak, aby zajistily škálovatelnost, redundanci a odolnost vůči výpadkům, což jsou klíčové

požadavky při práci s rozsáhlými datovými sadami (Hendl, 2021; Thanh et al., 2008). Jedním z nejznámějších distribuovaných souborových systémů je HDFS, který je optimalizovaný pro paralelní zpracování velkých souborů a efektivní využití výpočetních zdrojů. HDFS rozděluje soubory do bloků, které jsou distribuovány mezi jednotlivé uzly v clusteru, a využívá replikaci dat k minimalizaci rizika ztráty informací při selhání hardwaru. Jeho klíčovou výhodou je spolupráce s výpočetním frameworkem Apache Hadoop, což umožňuje efektivní distribuované výpočty a škálování dle potřeb organizací pracujících s big daty (Saeed a Husamaldin, 2021; Sun et al., 2023).

Alternativou k HDFS je Quantcast File System (QFS), navržený pro vysoký výkon a efektivní správu v prostředí big dat. QFS klade důraz na nízkou latenci při přístupu k souborům a vysokou efektivitu využití úložného prostoru. V porovnání s HDFS poskytuje lepší správu metadat a optimalizuje čtení/zápis souborů pro analytické a strojově učící úlohy. Podle studie Thanh et al. (2008) je QFS často využíván v aplikacích s požadavky na nízkou latenci a rychlou odezvu, přičemž jeho architektura umožňuje škálování od malých clusterů až po rozsáhlé datové infrastruktury. Kromě HDFS a QFS existují i další distribuované souborové systémy, jako jsou Google File System (GFS), Ceph nebo Lustre, které poskytují různé úrovně škálovatelnosti a optimalizace výkonu v závislosti na konkrétním použití (Sun et al., 2023). Moderní trendy ukazují, že distribuované souborové systémy se stále více integrují s cloudovými technologiemi a výpočetními frameworky jako je např. Apache Spark (Pallamala a Rodrigues, 2022).

2.3 NoSQL databáze a jejich principy

NoSQL databáze jsou další možností pro řešení ukládání a práci s big daty. Oproti tradičnějším relačním SQL databázím, které byly a jsou velmi často vyhledávanou volbou, mají NoSQL databáze řadu jiných vlastností (Gupta et al., 2017; Thakare et al., 2023). SQL databáze jsou relační systémy určené pro strukturovaná data, která jsou organizována v tabulkách s pevně definovaným schématem. Záznamy v těchto tabulkách jsou rozlišovány pomocí primárního klíče a pro správu a manipulaci s daty je používán jazyk SQL. S postupem času, vývojem technologií a zvyšováním objemu ukládaných dat vyvstal problém s tím, že v klasických SQL databázích vznikalo tolik relačních tabulek, že začaly vznikat problémy (Hendl, 2021; Wiese, 2015).

Problémy SQL databází, kterými je škálovatelnost a neschopnost práce s nestrukturovanými a polostrukturovanými daty vyřešil nástup NoSQL databází. NoSQL databáze jsou navrženy tak, že umožňují prakticky nekonečný růst uložených dat. Samozřejmě v rámci schopností

fyzických technologií. Ač by se z názvu mohlo zdát, že NoSQL opouštějí zcela od jazyka SQL, není tomu úplně tak. „No“ v rámci NoSQL neznamená „ne“ ale „not only“. NoSQL databáze mají tedy obdobu či vlastní SQL pro potřeby architektury daného typu NoSQL databáze (Cielen et al., 2016; Hendl, 2021).

NoSQL databází je hned několik druhů. Hendl (2021) zmiňuje organizační typy (datové modely): klíč-hodnota, dokumenty a grafy. Cielen et al. (2016) kromě těchto typů zmiňuje i sloupcové, streaming, NewSQL a SQL on Hadoop. Za obecně přijímané typy se považují klíč-hodnota, dokumentové, sloupcové a grafové, jak uvádí ve své práci Thakare et al. (2023).

Databáze typu klíč-hodnota nabízejí extrémně rychlý přístup k datům díky jednoduchému mapování klíčů na hodnoty. Tento model je často využíván v systémech, kde je potřeba nízká latence a vysoká výkonnost při čtení a zápisu. Klíče slouží jako identifikátory a hodnota je přiřazená ke klíči, hodnota může být zároveň dalším klíčem. Klíčem může být například identifikátor uživatele, web-session nebo nákupního košíku, hodnota poté preference uživatele nebo záznamy o daném web-session či košíku. Příkladem těchto databází jsou Redis nebo DynamoDB (Hendl, 2021; Thakare et al., 2023).

Dokumentově orientované databáze umožňují ukládat data ve formě dokumentů, obvykle ve formátu JSON nebo XML, což poskytuje větší flexibilitu a eliminuje nutnost pevně daného schématu. Tento přístup umožňuje snadnou práci s polostrukturovanými daty a je ideální pro aplikace, kde se struktura dat často mění. Jedním z nejrozšířenějších příkladů dokumentových databází je MongoDB, které umožňuje efektivní dotazování, agregaci dat a snadnou integraci s moderními webovými technologiemi (Hendl, 2021; Thakare et al., 2023).

Sloupcové databáze ukládají data ve sloupcích namísto tradičních řádků, což umožňuje rychlejší analýzu big dat. Data jsou ukládána sekvenčně po sloupcích, což znamená, že hodnoty stejného sloupce jsou uloženy společně, čímž se minimalizuje počet čtení z úložiště při dotazování na konkrétní atributy. Tento přístup umožňuje efektivní indexaci sloupců, což výrazně snižuje náklady na vstupně-výstupní operace při zpracování dotazů, které přistupují pouze k určitým sloupcům dat. Mezi nejznámější sloupcové databáze patří například HBase nebo Cassandra (Thakare et al., 2023).

Grafové databáze se používají tam, kde je důležité modelovat složitá propojení mezi entitami. Na rozdíl od tradičních tabulkových nebo dokumentových přístupů pracují s uzly a vztahy mezi nimi, což umožňuje efektivní reprezentaci a analýzu propojených dat. Je vhodné je použít tam, kde je dáván na vztahy opravdový důraz. Tento model je hojně využíván například v sociálních

sítích (Facebook, LinkedIn atd.) nebo systémech dopravy. Mezi nejznámější grafové databáze patří Neo4j (Cielen et al., 2016; Hendl, 2021; Thakare et al., 2023). Tabulka 1 pak zachycuje porovnání jednotlivých typů NoSQL databází podle vybraných kritérií.

Tabulka 1: Porovnání typů NoSQL databází. Zdroj: Gupta et al. (2017).

Typ (datový model)	**Výkon dotazů**	**Škálovatelnost dat**	**Flexibilita schématu**	**Struktura databáze**	**Složitost hodnot**
Klíč-hodnota	Vysoký	Vysoká	Vysoká	Primární klíč s přidruženou hodnotou	Žádná
Sloupcová databáze	Vysoký	Vysoká	Střední	Řádek obsahující více sloupců	Nízká
Dokumentová databáze	Vysoký	Variabilní (vysoká)	Vysoká	JSON ve stromové struktuře	Nízká
Grafová databáze	Variabilní	Variabilní	Vysoká	Graf – entity a jejich vztahy	Vysoká

2.4 Vybrané komerční nástroje

2.4.1 Google BigTable

Google BigTable je distribuovaný systém pro ukládání strukturovaných a polostrukturovaných dat, který se odlišuje od tradičních relačních databází odlišným datovým modelem. Je navržen pro škálovatelnost a vysoký výkon při práci s rozsáhlými datovými sadami. Již v roce 2008 ho využívalo více než 60 aplikací Googlu, včetně Google Earth a webového indexování, protože dokáže efektivně řešit úlohy, které nejsou vhodné pro standardní relační databáze. Vývoj BigTable byl zahájen v roce 2003 jako reakce na narůstající objemy dat v internetových aplikacích. Například Google Earth pracuje s více než 70 terabajty dat, což klade vysoké nároky na jejich správu a dostupnost. BigTable využívá distribuované uzly pro ukládání dat a od svého vzniku neustále roste (Chang et al., 2008).

BigTable využívá vícerozměrnou seřazenou mapu (Multi-dimensional Sorted Map), což mu poskytuje flexibilitu při práci s datovým modelem. Úložnou vrstvu tvoří GFS, který umožňuje vysokou škálovatelnost a spolehlivost. Data jsou ukládána do menších bloků (Chubby files),

které jsou replikovány na více uzlech v rámci systému. Tato strategie zajišťuje vysokou dostupnost a spolehlivost při zpracování dat v reálném čase (Chang et al., 2008; Kalid et al., 2017).

2.4.2 Amazon DynamoDB

Amazon DynamoDB je NoSQL databáze, která poskytuje vysoký výkon, nízkou latenci a automatickou škálovatelnost, což ji činí ideální volbou pro aplikace s náročnými požadavky na dostupnost a spolehlivost. Díky své bezserverové architektuře eliminuje nutnost manuální správy databázové infrastruktury a umožňuje automatické přizpůsobení výkonu podle aktuální zátěže. Přístup k datům probíhá výhradně na základě primárního klíče, což zajišťuje rychlé dotazování bez nutnosti složitých relačních spojení (Kalid et al., 2017; Sivasubramanian, 2012).

DynamoDB využívá distribuovanou architekturu, která kombinuje automatické dělení dat (sharding) a replikaci pro dosažení maximální odolnosti vůči výpadkům. Data jsou ukládána pomocí konzistentního hashování, které rovnoměrně distribuuje požadavky mezi uzly a tím zajišťuje optimální rozložení zátěže i při extrémně vysokém počtu operací (Sivasubramanian, 2012). Díky horizontální škálovatelnosti může DynamoDB bezproblémově růst s potřebami aplikací, aniž by docházelo k výkonnostním omezením. Navíc integrace s dalšími webovými službami Amazon rozšiřuje její využitelnost – například DynamoDB Streams umožňuje real-time zpracování změn v databázi, což je klíčové pro transakční systémy, analytické aplikace a aplikace vyžadující okamžité reakce na změny (Kalid et al., 2017; Remala et al., 2024).

2.5 Vybraná otevřená řešení

2.5.1 Apache Hadoop

HDFS je nejčastěji používaným distribuovaným souborovým systémem v oblasti big dat. (Cielen et al., 2016; Holdsworth, 2024). Apache Hadoop umožňuje efektivní správu rozsáhlých datových souborů tím, že je distribuuje mezi více zařízení, kterými mohou být specializované servery, ale také zcela běžné počítače, čímž je možné dosáhnout levnější realizace. Tato architektura umožňuje pracovat s daty téměř stejně snadno jako na lokálním souborovém systému, přestože se ve skutečnosti mohou nacházet na tisících serverech (Cielen et al., 2016; Hendl, 2021; Holdsworth, 2024).

Jádrem Hadoopu je několik klíčových komponent. HDFS zajišťuje distribuované ukládání dat, zatímco Yet Another Resource Negotiator (YARN) spravuje zdroje v rámci clusteru. Kromě

toho MapReduce umožňuje distribuované zpracování dat. Apache Hadoop ekosystém se však neomezuje pouze na tyto základní komponenty – zahrnuje také nástroje jako Hive, který umožňuje práci se strukturovanými daty pomocí dotazů založených na SQL, HBase, což je databáze s distribuovaným ukládáním využívající řádky, sloupce a třetí hodnotu k odlišení různých hodnot, a Mahout, který poskytuje framework pro strojové učení založený na knihovnách jazyka Java (Cielen et al., 2016; Hendl, 2021). Dále sem patří nástroje jako Oozie pro správu workflow, Flume pro sběr logů, Sqoop pro výměnu dat, Ranger pro zabezpečení nebo Zookeeper pro koordinaci clusteru (Cielen et al., 2016; Holdsworth, 2024).

2.5.2 Apache Spark

Apache Spark je open-source framework pro distribuované zpracování dat, který umožňuje rychlou analýzu big dat. Na rozdíl od Apache Hadoop podporuje interaktivní dotazy, streamové zpracování a iterativní výpočty, což jej činí vhodným pro strojové učení a analytické úlohy. Spark je kompatibilní s různými systémy správy clusterů, včetně nativního Spark Cluster, Hadoop YARN a Apache Mesos, a může pracovat s širokou škálou úložných řešení, jako je HDFS, Cassandra nebo Amazon S3. Původně byl vyvinut na UC Berkeley AMPLab v roce 2009, v roce 2013 se stal součástí Apache Software Foundation a od té doby patří mezi nejaktivnější projekty v oblasti big dat (Ibtisum et al., 2023; Salloum et al., 2016).

Jednou z hlavních výhod Sparku je jeho výkon, který v mnoha scénářích převyšuje Apache Hadoop. Například Spark SQL, nástupce projektu Shark, dokáže provádět SQL dotazy až 80x rychleji než Hive při práci s daty uloženými v paměti a 5x rychleji při čtení z HDFS. Podobně ve strojovém učení dosahuje Spark 100x rychlejšího zpracování u logistické regrese a 30x rychlejšího běhu algoritmu k-means oproti Hadoopu. Kromě výkonu nabízí Spark také snadné API pro Scala, Java a Python, což usnadňuje jeho využití pro širokou škálu aplikací, od tradiční analýzy dat po pokročilé umělé inteligence a prediktivní modelování (Ibtisum et al., 2023; Salloum et al., 2016).

2.5.3 Apache Parquet

Apache Parquet je open-source sloupcově orientovaný formát souborů, vyvinutý Twitterem a Clouderou pro ekosystém Hadoop, inspirovaný technologií Google Dremel. Díky sloupcové struktuře umožňuje výrazně lepší kompresi a efektivnější zpracování dotazů, zejména při přístupu pouze k vybraným sloupcům, čímž snižuje zátěž na vstupně-výstupní operace. Mezi jeho klíčové optimalizace patří statistiky sloupcových bloků (například min/max hodnoty),

které ve spojení s predicate pushdown strategií umožňují filtrování dat již v raných fázích zpracování, což minimalizuje objem načítaných dat. Parquet také využívá pokročilé kompresní algoritmy, jako je Run Length Encoding pro ukládání opakujících se hodnot v kompaktní podobě a Bitpacking, který efektivně zpracovává ne-byte velikostní hodnoty, čímž zvyšuje výkon procesoru a optimalizuje datové operace. Díky těmto vlastnostem se Parquet stal standardem pro efektivní ukládání a analýzu velkých objemů dat v distribuovaných systémech (Vohra, 2016).

Pro optimalizaci ukládání big dat by organizace měly implementovat vhodné kompresní techniky a správné formáty souborů, přičemž Apache Parquet je jedním z nejefektivnějších řešení pro jejich ukládání s využitím Apache Hadoop a Apache Spark. Parquet je nativně podporován oběma technologiemi, což umožňuje efektivní distribuované zpracování a analýzu dat. V prostředí HDFS jeho sloupcově orientovaná struktura snižuje množství čtených dat při dotazování a zároveň umožňuje vyšší kompresi oproti tradičním řádkovým formátům, čímž snižuje nároky na úložný prostor. V Apache Spark Parquet optimalizuje výkon dotazů, protože umožňuje predicate pushdown, tedy filtrování dat už při načítání, což výrazně zrychluje analytické operace. Díky těmto vlastnostem se Parquet stal standardním úložným formátem pro organizace využívající big data technologie k business intelligence, strojovému učení a datové analytice (Singh et al., 2025).

2.5.4 Apache Cassandra

Apache Cassandra je vysoce škálovatelná NoSQL databáze, která umožňuje efektivní práci s big daty. Díky své schopnosti zvládat big data je široce využívána v korporátní sféře. Otevřená verze Cassandry byla představena v roce 2009 a původně sloužila k pokrytí potřeb společností jako Google, Facebook a Amazon v oblasti správy rozsáhlých datových infrastruktur. V současnosti se Cassandra uplatňuje v moderních podnicích, které potřebují spolehlivou a flexibilní správu dat. Oproti BigTable a DynamoDB poskytuje vyšší flexibilitu při řešení chyb a zpracování různých druhů dat. Cassandra kombinuje metody používané jinými NoSQL databázemi a díky tomu zajišťuje vysokou dostupnost a škálovatelnost (Carpenter a Hewitt, 2022; Kalid et al., 2017).

Apache Cassandra vyniká distribuovanou architekturou, která eliminuje single points of failure a zajišťuje vysokou dostupnost i při velké zátěži. Díky peer-to-peer replikaci a horizontální škálovatelnosti lze snadno přidávat nové uzly bez přerušení provozu. Cassandra podporuje tunable consistency, což umožňuje nastavení úrovně konzistence dle potřeb aplikace – od silné

až po eventuální konzistenci. Tato flexibilita činí Cassandru ideální pro IoT, finanční služby a datovou analytiku, kde je klíčová nízká latence a spolehlivost (Carpenter a Hewitt, 2022).

2.6 Porovnání nástrojů a metod

Tabulka 2 obsahuje porovnání vybraných komerčních a otevřených nástrojů, které lze zařadit do kategorie NoSQL databází. Shrnutí bylo provedeno na základě studie relevantních zdrojů jako je Bagga a Sharma (2021), Chang et al. (2008), Chebotko et al. (2015), Kalid et al. (2017) a Weintraub (2014).

Srovnání databází BigTable, DynamoDB a Apache Cassandra ukazuje rozdíly zejména ve škálovatelnosti, dostupnosti a metodě práce s daty. BigTable vyniká vysokým výkonem a efektivním skenováním strukturovaných dat pomocí dynamicky dělených hash tabulek, zatímco Cassandra poskytuje kontinuální dostupnost, vysoký výkon a lineární škálovatelnost díky konzistentnímu hashování. DynamoDB upřednostňuje konzistenci, je optimalizována pro webové aplikace a nabízí masivní škálovatelnost. Z hlediska odolnosti proti selhání vynikají BigTable a Cassandra, přičemž API všech tří databází se liší podle typu operací.

Tabulka 2: Porovnání vybraných NoSQL databází. Zdroj: vlastní.

NoSQL databáze / Charakteristiky	**BigTable**	**DynamoDB**	**Cassandra**
Výkon systému	Velmi vysoký výkon s efektivním skenováním	Optimalizováno modelem průchodnosti	Velmi vysoký výkon
Škálovatelnost	Vysoká	Masivní a plynulá škálovatelnost	Lineární škálovatelnost
Dostupnost	Přístup k většině aktuálních datových sad	Konzistentní	Kontinuální
Operace s daty	Čtení / zápis (řádek, sloupec nebo časové razítko)	Přístup pomocí klíče a zápis na disk	Čtení / zápis
Datové sady	Strukturovaná data	Strukturovaná a nestrukturovaná data	Strukturovaná a nestrukturovaná data

NoSQL databáze / Charakteristiky	BigTable	DynamoDB	Cassandra
Metoda dělení dat	Dynamicky dělené hash tabulky	Konzistentní hashování	Konzistentní hashování
Odolnost proti selhání	ANO	NE	ANO
API	Scan, get, del, put	Put, get	Get, put

Pro porovnání metod byly do přehledu k NoSQL datovým modelům přidány i další přístupy, tzn. distribuované souborové systémy a cloudová řešení. Shrnutí bylo provedeno na základě studie relevantních zdrojů jako je Cielen et al. (2016), Kalid et al. (2017), Strohbach et al. (2016), Sun et al. (2023), Thanh et al. (2008) a Wiese (2015). Porovnání zachycuje Tabulka 3.

Na základě porovnání distribuovaných souborových systémů, NoSQL datových modelů a cloudových řešení lze konstatovat, že každý z těchto přístupů nabízí odlišné výhody podle povahy použití. Distribuované souborové systémy jsou vhodné zejména pro zpracování big dat v dávkovém režimu a vědeckých aplikacích. Vyznačují se hierarchickou strukturou souborů a horizontální škálovatelností, avšak jejich správa vyžaduje pokročilé odborné znalosti. Naproti tomu NoSQL modely jsou navrženy pro webové aplikace v reálném čase, podporují flexibilní schémata (např. dokumentová, klíč–hodnota) a škálují horizontálně s důrazem na eventuální konzistenci. Jsou tedy vhodné tam, kde je prioritou rychlý přístup k datům a vysoká dostupnost. Cloudová řešení poskytují automatické škálování, vysokou nákladovou efektivitu a plnou správu poskytovatelem, což z nich činí volbu pro organizace hledající flexibilitu a minimální provozní zátěž.

Tabulka 3: Porovnání vybraných metod ukládání big dat. Zdroj: vlastní.

Metody ukládání big dat / Charakteristiky	**Distribuované souborové systémy**	**NoSQL datové modely**	**Cloudová řešení**
Primární využití	Ukládání velkých objemů dat, dávkové zpracování, vědecký výzkum	Webové aplikace v reálném čase, flexibilita schémat, vysoká dostupnost	Úsporné ukládání, archivace, integrace se službami cloudu
Datový model	Ukládání souborů s hierarchickou adresářovou strukturou	Flexibilní schémata: klíč–hodnota, dokumentové, sloupcové, grafové modely	Objektové úložiště pro strukturovaná, polostrukturovaná a nestrukturovaná data
Škálovatelnost	Horizontální škálování pomocí distribuovaných uzlů	Horizontální škálování s eventuální konzistencí	Automatické škálování dle potřeby
Výkon	Optimalizováno pro vysokou propustnost při dávkovém zpracování	Optimalizováno pro nízkou latenci a přístup v reálném čase	Optimalizováno pro nákladovou efektivitu a bezproblémovou integraci
Správa	Vyžaduje odborné znalosti pro správu a optimalizaci	Spravováno databázovými administrátory	Plně spravováno poskytovateli cloudových služeb

Nakonec byly porovnány vybrané nástroje s ohledem na metody ukládání big dat a jejich vazbu na NoSQL databáze. Kromě již výše uvedených zdrojů byly využity ještě Hendl (2021), Holdsworth (2024), Ibtisum et al. (2023) a Salloum et al. (2016). Tabulka 4 shrnuje klíčové charakteristiky tří hlavních technologií – Apache Hadoop, MapReduce a Apache Spark. Zaměřuje se na jejich strukturu, způsob zpracování dat, podporu různých úložišť, výkon, rozšiřitelnost, možnosti integrace s dalšími nástroji i výhody a nevýhody jednotlivých řešení. Cílem této tabulky je přehledně ilustrovat rozdíly mezi tradičním dávkovým zpracováním (MapReduce), komplexním ekosystémem pro ukládání a výpočet (Hadoop) a moderním přístupem k rychlému zpracování dat v paměti (Spark).

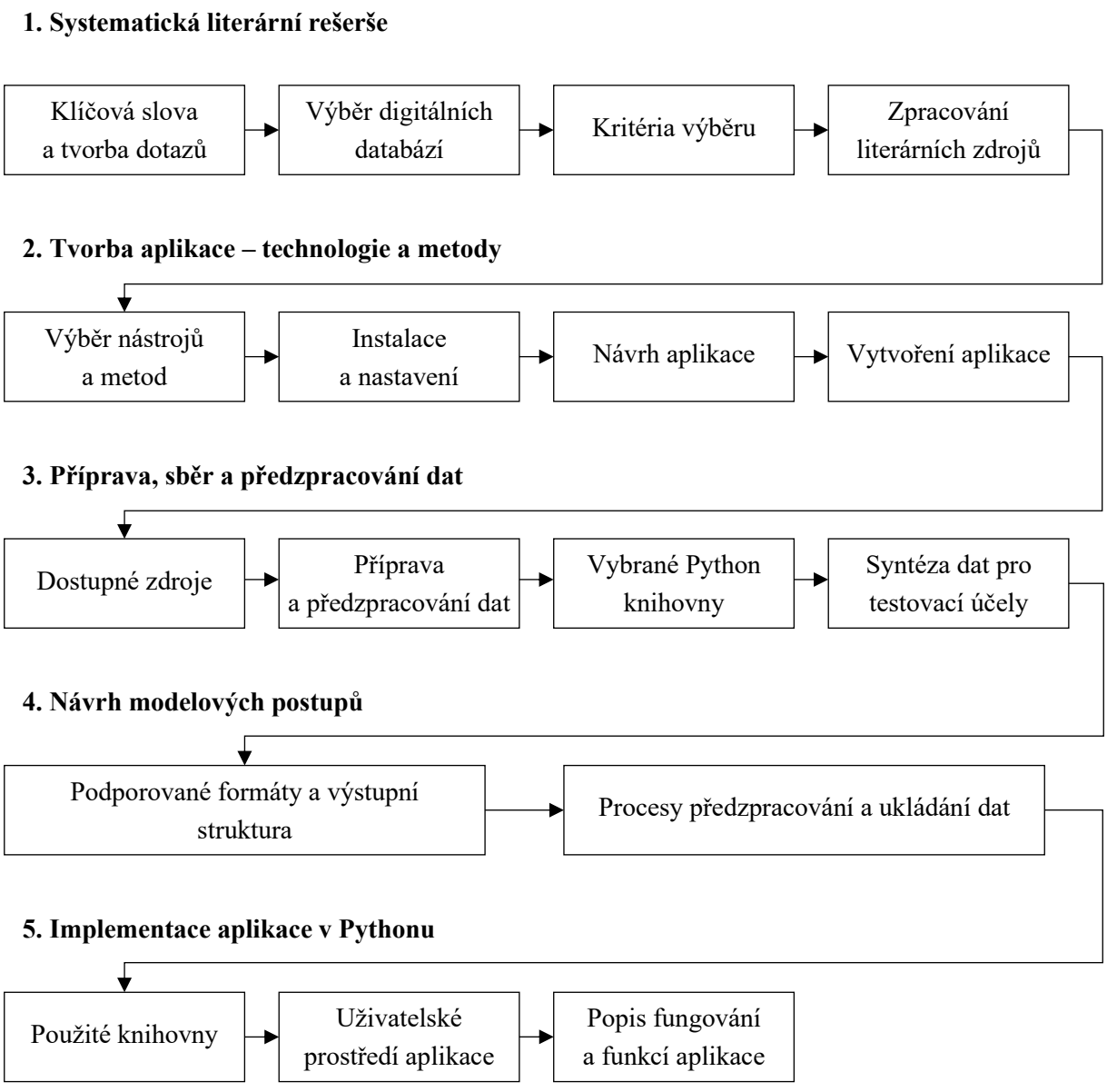
Tabulka 4: Porovnání Apache Hadoop, MapReduce a Apache Spark. Zdroj: vlastní.

Charakteristiky	**Apache Hadoop**	**MapReduce**	**Apache Spark**
Typ nástroje	Rámec a ekosystém pro distribuované ukládání a zpracování dat	Výpočetní model (součást Hadoopu)	Rámec pro distribuované zpracování dat
Úložiště	HDFS	Neposkytuje úložiště – zpracovává data z HDFS nebo jiných zdrojů	Podpora různých úložišť: HDFS, Cassandra, Amazon S3
Způsob zpracování	Distribuované zpracování pomocí MapReduce	Dávkové zpracování dat ve dvou fázích: Map a Reduce	In-memory zpracování, podpora dávkového i streamového výpočtu
Správa zdrojů	YARN	Správa výpočtů v rámci MapReduce	Vlastní cluster manager, podporuje i YARN a Apache Mesos
Podpora SQL dotazů	Hive – převod SQL na MapReduce	Nepřímá (pomocí Hive)	Spark SQL – výrazně rychlejší než Hive
Výkon	Výkon závislý na diskovém vstup/výstup a MapReduce	Efektivní pro dávkové úlohy, pomalé pro iterace	Vysoce výkonný – až 100x rychlejší pro některé úlohy
Podpora streamingu	Ne	Ne	Ano
Podpora strojového učení	Mahout (nad MapReduce, Java knihovny)	Omezeně (přes Mahout)	MLLib – výkonný modul pro strojové učení
Rozšiřitelnost	Vysoká – ekosystém zahrnuje Hive, HBase, Mahout, Oozie apod.	Součást Hadoopu, málo modulární	Velmi modulární – snadná integrace různých knihoven a jazyků

Charakteristiky	Apache Hadoop	MapReduce	Apache Spark
Výhody	Robustní systém pro big data, podpora běžného hardwaru, otevřený ekosystém	Jednoduchý, odolný, dobře škáluje	Extrémní výkon, flexibilita, API pro více jazyků, umělá inteligence
Nevýhody	Nižší výkon ve srovnání se Sparkem, složitější pro vývojáře	Pomalý při iteracích a streamování	Vyšší paměťové nároky, může být náročnější na správu

3 METODIKA PRÁCE

Níže je popsán metodický postup řešení cíle práce, tzn. *charakterizovat současné přístupy k ukládání big dat, porovnat existující nástroje a metody, provést sběr a předzpracování dat, navrhnout modelové postupy, a nakonec implementovat celé řešení prostřednictvím aplikace v Pythonu.* Je rozdělen do 5 částí, kde každá z nich se skládá z dílčích kroků, viz Obrázek 1.



Obrázek 1: Metodický postup řešení cíle práce. Zdroj: vlastní.

Pro vymezení požadavků na aplikaci a vytvoření přehledu aktuálních trendů, a především existujících nástrojů a metod byla použita systematická literární rešerše. S ohledem na zjištěné požadavky, tzn. přístupy k řešení ukládání big dat a související výzvy, byly zvoleny vhodné

nástroje a metody, které lze použít pro tvorbu aplikace. Následně bylo nutné provést přípravu, sběr a předzpracování dat. Klíčový kroky zde bylo vymezení dostupných dat a výběr Python knihoven, které jsou pro předzpracování a ukládání dat nejvhodnější. Nakonec byla provedena syntéza dat pro testovací účely. Další krokem byl návrh modelových postupů, tzn. konkrétní kroky a metody, pomocí kterých se budou data předzpracovávat a ukládat. Poslední krokem je implementace aplikace v Pythonu a popis jejich funkcí.

3.1 Systematická literární rešerše

Pro účely systematické literární rešerše v této práci jsou využívány principy založené na zjištěních Kitchenham et al. (2009), kteří ve své studii zaměřené na oblast softwarového inženýrství navrhují metodologii systematických přehledů literatury. Tato metodologie je široce aplikovatelná i v jiných oblastech a zahrnuje tři klíčové fáze: (1) plánování systematické literární rešerše, (2) provedení rešerše a (3) analýzu a syntézu výsledků. V rámci plánování je klíčovým krokem vymezení kroků, které směřují k dosažení cílů diplomové práce. Na základě těchto kroků byly definovány tři tematické oblasti související s problematikou ukládání big dat.

První oblast se zaměřuje na aktuální trendy a metody ukládání velkých dat, přičemž klíčová slova vedla k formulaci dotazu: "big data storage" OR "big data storing" AND "method" OR "approach" AND "trend". Druhá oblast se soustředí na nástroje a platformy pro ukládání big dat, pro což byl použit vyhledávací dotaz: "big data storage" OR "big data storing" AND "tool" OR "platform" OR "service" AND "comparison" OR "overview" OR "survey". Třetí oblast se zaměřuje na návrh aplikace v Pythonu pro předzpracování a ukládání big dat, s vyhledávacím dotazem: "big data storage" OR "big data storing" AND "application" OR "solution" AND "Python" AND "development". První dvě oblasti poskytují teoretický i praktický rámec pro pochopení problematiky big dat a jejich ukládání, zatímco třetí oblast je klíčová pro vývoj vlastní aplikace, jelikož se zaměřuje na technologické aspekty implementace.

Dalším zásadním krokem bylo vybrání vhodných digitálních zdrojů, kde byly formulované dotazy aplikovány. Pro zajištění kvalitní literatury byly zvoleny vědecké databáze, konkrétně Scopus, Web of Science a Google Scholar, které nabízejí široký přístup k odborným vědeckým článkům a studiím. Při vyhledávání byly výsledky podrobeny dalšímu filtrování, aby bylo zajištěno, že klíčová slova odpovídají tématu výzkumu a vyskytují se v názvu, abstraktu nebo klíčových slovech dané publikace. V souladu s doporučeními Kitchenham et al. (2009) byla jako další kritéria zvolena:

1. časové omezení, a to posledních 20 let, kdy v roce 2004 byl publikován programovací model MapReduce, který je považován za počátek problematiky (Dean a Ghemawat, 2004),
2. relevantní vědní obory – computer science, decision sciences, engineering, information systems, software, economics, a business a management,
3. dostupnost plného textu publikací – upřednostněny byly open access publikace nebo zdroje dostupné prostřednictvím institucionálního předplatného Univerzity Pardubice.

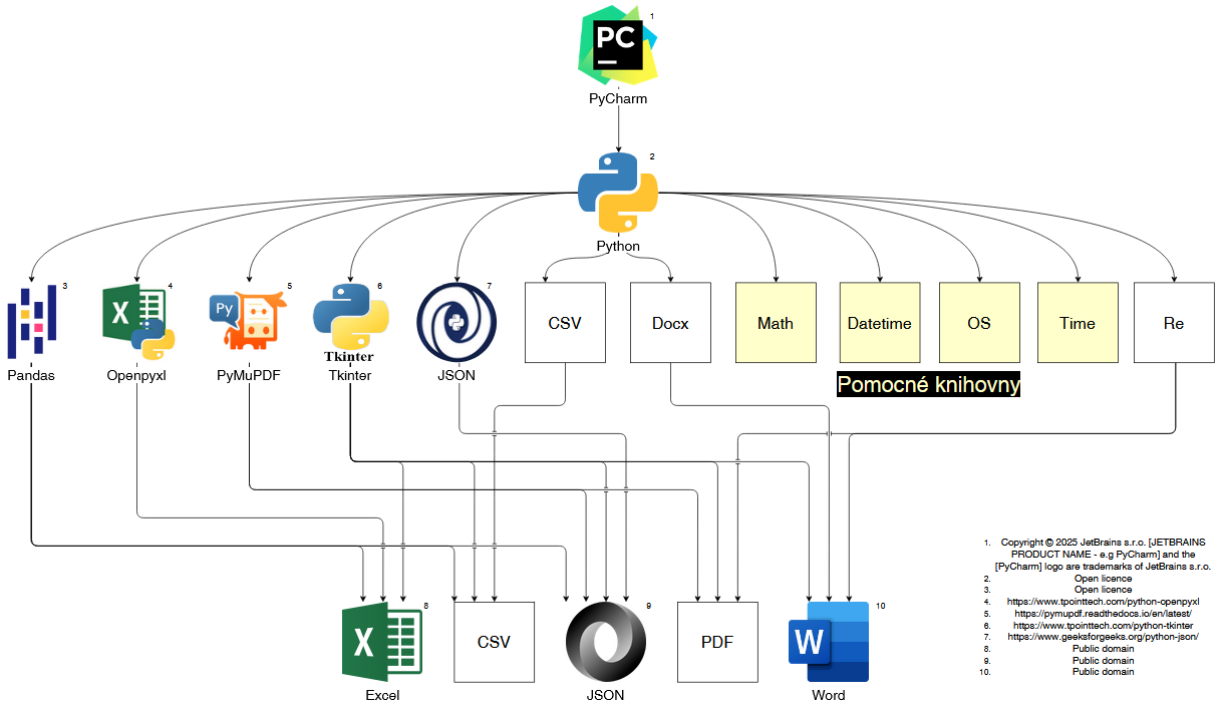
Na základě těchto metodologických kroků byla sestavena konečná sada relevantní literatury, která byla dále systematicky analyzována a roztříděna podle tematických oblastí výzkumu. Kvalitativní a kvantitativní analýza výsledků pomohla identifikovat nejčastěji používané přístupy, nejnovější trendy v oblasti ukládání big dat a nejvhodnější technologie pro návrh aplikace.

3.2 Tvorba aplikace

3.2.1 Výběr nástrojů a platforem

Na základě literární rešerše, zjištění aktuální praxe a praktického aspektu výběru současných nástrojů a metod byly určeny technologie a postupy, které jsou při tvorbě aplikace využity. Jako nejvhodnější bylo vybráno jedno z nejrozšířenějších programovacích prostředí od pražské firmy JetBrains – PyCharm, které nabízí přehledné prostředí pro práci s programovacím jazykem Python. Firma nabízí placenou i komunitní verzi, zároveň i licenci zdarma pro studenty. Prostředí pomáhá při hledání syntaktických chyb, při kontrolách aktuálních verzí knihoven nebo například při automatické nápovědě syntaxí knihovních funkcí (JetBrains, 2025; Pecinovský, 2021).

V prostředí jazyka Python existuje široká škála dostupných řešení pro vývoj aplikací. V rámci této práce byly využity výhradně otevřené knihovny jazyka Python, které umožňují plnou flexibilitu při tvorbě kódu a zajišťují, že výsledná aplikace zůstane dostupná pro bezplatné použití. Použití komerčních nástrojů by mohlo omezit její otevřenost a dostupnost. Při návrhu bylo klíčové zvolit vhodné uživatelské prostředí s ohledem na potřeby aplikace a očekávání uživatelů. Pro zajištění intuitivní interakce a snadného ovládání bylo implementováno grafické uživatelské rozhraní (Graphical User Interface, GUI) pomocí knihovny Tkinter. Obrázek 2 zachycuje technologický stack aplikace.



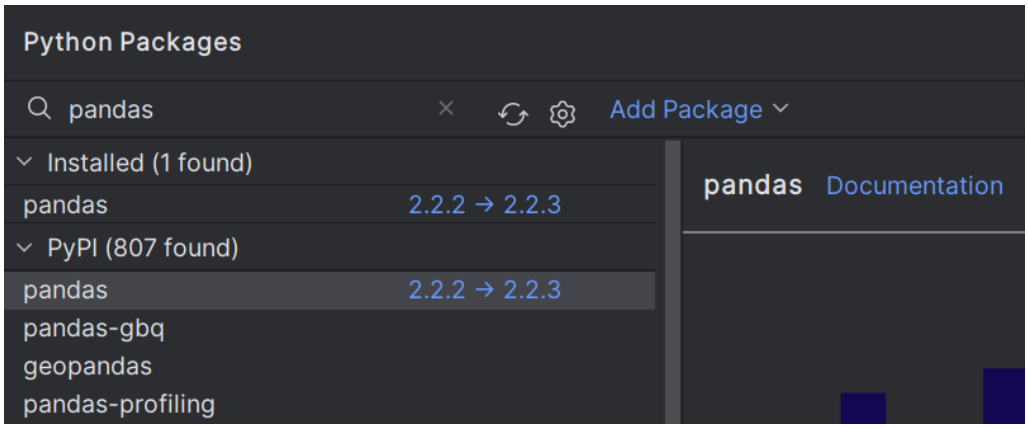
Obrázek 2: Technologický stack aplikace. Zdroj: vlastní.

3.2.2 Instalace a nastavení

Při instalaci vývojového prostředí PyCharm lze postupovat podle návodu od firmy JetBrains, který je dostupný na stránce: https://www.jetbrains.com/help/pycharm/installation-guide.html. Návod nabízí několik možných postupů instalace a uvádí systémové požadavky. Za minimální požadavky je uváděno mít 2GB volné RAM paměti, 3.5GB volného místa na úložišti, rozlišení monitoru minimálně 1024x768 a dále minimální verze jednotlivých operačních systémů. Pro platformu Windows je to verze Windows 10 1809, pro maxOS verze 12.0 a například pro Ubuntu nebo Fedora distribuce Linuxu je to verze jádra 6.x, Gnome nebo KDE prostřední a X Windows Systém (X11).

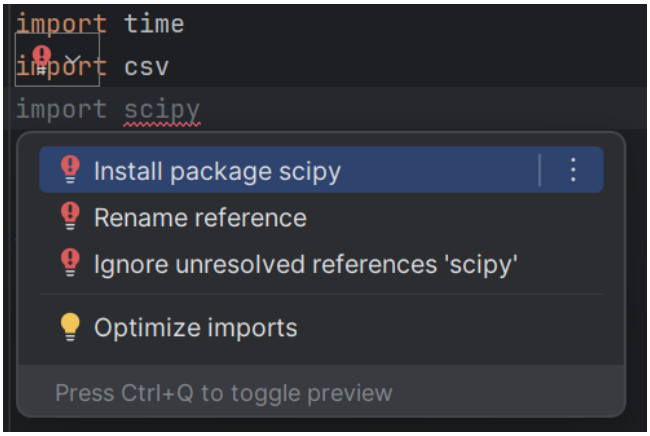
Je rovněž nutné zohlednit i případné samotné nároky vyvíjené aplikace. Vývoj aplikace probíhal na operačním systému Windows, a proto zde bude popsán postup právě pro tento operační systém. Vývojáři doporučují využít nástroje JetBrains Toolbox, jakožto správce produktů JetBrains. Ovšem pro účely práce byla zvolena samostatná instalace, protože bylo potřeba pouze vývojové prostředí PyCharm a nikoli balík více produktů. Následně stačí postupovat podle návodu. Je třeba stáhnout .exe instalační soubor a spustit ho na zařízení (pro jiné operační systémy se postup může lišit). Dále pak postupovat dle průvodce instalací. Po instalaci prostředí je vhodné nainstalovat Python knihovny. Návod k instalaci Python balíčků

je dostupný na adrese: https://www.jetbrains.com/help/pycharm/installing-uninstalling-and-upgrading-packages.html#packages-tool-window, kde je popsáno fungování správce balíčků Python, včetně způsobu jeho spuštění. Po spuštění se v rámci prostředí objeví okno, viz Obrázek 3. Pro potřeby ukázky je zobrazeno vyhledávání konkrétní knihovny, kde jsou vidět nalezené instalované knihovny a další vyhledané knihovny z repositáře, které je možné nainstalovat do projektu. Zároveň tento manažer informuje o možnosti aktualizovat knihovnu na novější verzi.



Obrázek 3: Vyhledávání knihoven Python. Zdroj: vlastní.

Ovšem v rámci potřeb toho projektu byl zvolen způsob skrze nápovědu, kde samotné prostředí kontroluje, zda jsou knihovny z kódu i nainstalovány, viz Obrázek 4. Skrze našeptávač tak lze doinstalovat všechny potřebné knihovny pro správné fungování kódu aplikace. To znamená, že i pokud vývojář zapomene knihovny nainstalovat přes manažera, stále ho prostředí naviguje k instalaci nedostupných knihoven. Při případném využití zdrojového kódu aplikace v rámci prostředí PyCharm by tedy nemělo dojít k chybám s nedostupností knihoven, na kterou by uživatel nebyl upozorněn.



Obrázek 4: Kontrola existence Python knihoven. Zdroj: vlastní.

3.2.3 Návrh aplikace

Návrh aplikace je založen na přehledu literatury, která nabízí pokyny a doporučení pro vývoj aplikace zaměřené na ukládání big dat v jazyce Python. Hlavními zdroji jsou Cielen et al. (2016) a McKinney (2017), ale aplikace se rovněž odkazuje na publikace od autorů Pecinovský (2022), Computer Science Academy (2020) nebo Stewart a Mommert (2023), které byly použity pro teoretický základ k syntaxi, sémantice a vlastnostem jazyka Python.

Na základě této literatury byl zvolen přístup k vývoji. Jak uvádí například Cielen et al. (2016): „*Neobjevujte znovu kolo*“, proto jsou v aplikaci využívány již existující knihovny. Není důvod řešit funkce, které již někdo implementoval. V tom se shodují i další autoři, kteří taktéž vyzdvihují velký počet Python knihoven a nepřeberným množstvím funkcí a nástrojů. Dalším aspektem při návrhu aplikace byla struktura kódu. McKinney (2017) zdůrazňuje fakt, že hluboce vnořené struktury snižují čitelnost kódu. Je tedy žádoucí mít funkce jednodušší, aby byly případné chyby snáze detekovatelné a vývojář nemusel procházet vrstvy logických funkcí. Stejně tak autor zastává názor, že v rámci jazyka Python nejsou delší kódy na škodu, spíše naopak, a potvrzuje výhody méně vnořených (plochých) kódů.

Při návrhu aplikace byl také kladen důraz na funkcionality a takové prostředí, které bude efektivním nástrojem pro řešení problémů spojených s ukládáním big dat. K tomu bylo nezbytné podrobně analyzovat potřeby cílové skupiny, což umožnilo definovat klíčové funkce, které byly implementovány s ohledem na reálné scénáře využití. Při návrhu architektury aplikace byl kladen důraz na využití frameworku Tkinter pro uživatelské rozhraní. Integrace grafického uživatelského rozhraní umožňuje přístupnost aplikace i pro uživatele bez hlubších technických znalostí, což odpovídá doporučením Cielen et al. (2016) týkajících se uživatelské přívětivosti analytických nástrojů. Proto bylo navrženo rozhraní, které klade důraz na jednoduchost a přehlednost.

3.2.4 Vytvoření aplikace

Po vyjasnění očekávaného výstupu začal vývoj samotné aplikace. Vývoj byl zaměřen na vytvoření programu umožňujícího zpracování velkých datových souborů a jejich optimalizované ukládání. Prvním krokem bylo navržení architektury aplikace, která měla zajistit možnost rozšíření podle budoucích potřeb uživatelů a systémů. Bylo potvrzeno, že aplikace bude napsána v jazyce Python, a to s využitím klíčových knihoven pro zpracování dat, jako jsou Pandas, Openpyxl, PyMuPDF a další.

Základem vytvořené aplikace je systém pro převody různých formátů souborů do jednotného standardizovaného výstupu, primárně do formátu JSON. Práce byla zahájena implementací podpory pro zpracování tabulkových dat, konkrétně Excelových a CSV souborů. Byla využita knihovna Pandas, která umožnila snadnou manipulaci s datovými rámci a jejich export do požadovaných formátů. Při testování se ukázalo, že některé soubory obsahují rozdílné formátování, například rozdílné oddělovače sloupců v CSV souborech. Proto byl přidán mechanismus, který umožňuje uživateli určit formát souboru pro přizpůsobení zpracování dat.

Další část vývoje byla zaměřena na zpracování Portable Document Format (PDF) dokumentů. Byla implementována podpora pro extrakci textu z PDF pomocí knihovny PyMuPDF, která se ukázala jako výkonná a přesná pro práci s textem v PDF souborech. V průběhu vývoje byla rovněž implementována funkcionalita slučování JSON souborů. Bylo nutné zajistit, aby data pocházející z různých zdrojů byla sloučena do jednotné struktury bez ztráty důležitých informací. Byl tedy navržen systém, který umožnil slučování datových struktur na základě předdefinovaných pravidel.

Součástí vývoje aplikace bylo i vytvoření jednoduchého uživatelského rozhraní umožňujícího snadné ovládání jednotlivých funkcí. Byla zvolena knihovna Tkinter, která poskytuje základní grafické prvky potřebné pro interaktivní práci s aplikací. Rozhraní umožňuje uživatelům vybrat vstupní soubory, nastavit parametry zpracování a následně spustit samotný převod nebo slučování dat jedním kliknutím.

Celý proces vývoje byl iterativní, kdy byly postupně testovány a zdokonalovány jednotlivé moduly aplikace. Klíčovým aspektem byla optimalizace výkonu, především u velkých datových sad, kde bylo nutné minimalizovat nároky na paměť a zpracování. Byly implementovány techniky, jako je zpracování dat po blocích, místo načítání celých souborů do paměti najednou, což umožnilo efektivnější práci s rozsáhlými daty.

4 PŘÍPRAVA, SBĚR A PŘEDZPRACOVÁNÍ DAT

V této kapitole je popsán proces získání testovacích big dat a jejich příprava a předzpracování.

4.1 Dostupné zdroje

Dostupné zdroje dat hrají klíčovou roli v zpracování big dat, přičemž je možné využít jak veřejně dostupné databáze – zde jsou data zpravidla dostupná jako big open nebo big open linked data (Lnenicka a Komarkova, 2019), tak vlastní datové sady. Cielen et al. (2016) uvádějí, že organizace již často disponují rozsáhlými databázemi, které mohou být použity pro analytické účely, a pokud interní data nestačí, lze využít otevřené datové zdroje nebo komerční databáze dostupné prostřednictvím třetích stran.

Mezi hlavní dostupné zdroje dat patří interní data organizací, jako jsou datové sklady, datová jezera či firemní databáze. Dále existují externí otevřené zdroje dat, například Data.gov, World Bank Open Data, Freebase.org nebo národní datové portály, na kterých lze nalézt otevřená data veřejné správy. V případě nedostupnosti vhodných dat lze využít i komerční datové služby, které poskytují přístup k specializovaným databázím, například v oblasti finanční analýzy nebo průzkumu trhu. Podobné závěry uvádí i Ikegwu et al. (2022), přestože se spíše zaměřuje na průmyslové využití takových dat. Potvrzují, že hlavními zdroji jsou interní data organizací, ale zaměřují se na data v rámci IoT a dalších zdrojích, které v rámci průmyslu mohou generovat data, se kterými lze dále pracovat.

4.2 Příprava a předzpracování dat

Příprava a předzpracování dat je komplexní proces zahrnující několik fází, které transformují surová data do vhodných struktur a vstupů pro další fáze. Tento proces obvykle začíná detekcí a opravou chyb v datech, pokračuje kombinací dat z různých zdrojů a končí jejich přetvořením do formátu vhodného pro modelování a analýzu. Jak uvádějí Cielen et al. (2016), je vhodné chyby odstraňovat co nejdříve v procesu získávání dat, protože pozdní opravy zvyšují náklady a riziko špatných rozhodnutí na základě chybných údajů. Mezi běžné typy chyb, které je nutné během předzpracování řešit, patří nadbytečné znaky, rozdíly ve velikostech znaků (velké vs. malé znaky), hodnoty mimo rozsah, chybějící hodnoty nebo odlehlé hodnoty. Například redundantní mezery na konci řetězců mohou způsobit neshody při spojování datových tabulek, zatímco špatně formátovaná velikost písmen může vést k duplicitám.

Důležitou součástí předzpracování dat je také práce s chybějícími hodnotami. Tyto hodnoty nemusí být nutně chybné, ale je nezbytné je zpracovat – například pomocí přiřazení průměru, náhradou hodnotou nula, nebo vložením hodnoty podle rozložení dat. Rozhodnutí o způsobu řešení chybějících hodnot by mělo vždy vycházet z kontextu konkrétního datasetu. Podobné závěry uvádí i studie od autorů Kwak a Kim (2017), která potvrzuje důležitost práce s daty, zvláště chybějícími hodnotami, a upozorňuje na možné zkreslení, pokud budou prázdné hodnoty nesprávně nahrazeny.

4.3 Vybrané Python knihovny

Knihovna Pandas představuje základní nástroj pro práci se strukturovanými daty v Pythonu. Poskytuje datové struktury jako DataFrame a Series, které umožňují efektivní manipulaci s tabulkovými daty. Pandas podporuje operace jako filtrování, agregace, spojování tabulek či práce s časovými řadami. McKinney (2017) uvádí, že Pandas kombinuje výkonnost NumPy s flexibilitou relačních databází a tabulkových procesorů, čímž poskytuje robustní prostředí pro přípravu a čištění dat. Cielen et al. (2016) pak zdůrazňují jeho roli při načítání, kombinování a transformaci datových zdrojů v rámci datově-vědního procesu. Pecinovský (2023) ve své knize popisuje mnoho praktických využití knihovny. Mimo jiné potvrzuje práci s tabulkovými daty včetně filtrování a zmiňuje popisné funkce jako je například describe().

Knihovna NumPy je základem numerických výpočtů v Pythonu. Poskytuje vícerozměrné pole ndarray, nad kterým lze provádět vektorové operace, statistické výpočty a další výkonné transformace dat. NumPy funguje jako základní vrstva mnoha dalších knihoven, včetně Pandas a scikit-learn. McKinney (2017) přímo uvádí, že pro numerická data je NumPy mnohem efektivnější než vestavěné Python seznamy a že je vhodný zejména pro práci s velkými objemy dat díky své paměťové úspornosti a výkonnosti. Cielen et al. (2016) považují NumPy za jeden ze základních stavebních bloků data science stacku v Pythonu. Pecinovský (2023) připisuje knihovně stejně důležitou roli při vědeckých výpočtech a zmiňuje velkou obsáhlost funkcí.

Knihovna scikit-learn se využívá pro účely modelování, predikce a přiřazení chybějících hodnot. Tato knihovna poskytuje širokou škálu algoritmů pro klasifikaci, regresi, shlukování a nástroje pro výběr modelů, škálování, transformaci a zpracování dat. McKinney (2017) zmiňuje, že scikit-learn je jednou z nejspolehlivějších a nejrozšířenějších knihoven pro strojové učení v Pythonu. Cielen et al. (2016) dodávají, že scikit-learn je preferovaným nástrojem mnoha datových vědců díky své jednoduchosti a integraci se zbytkem Python ekosystému.

Knihovna matplotlib slouží jako nízkoúrovňový nástroj pro tvorbu 2D grafů, zatímco seaborn nad ní staví a nabízí jednodušší rozhraní s důrazem na statistické vizualizace. Vizualizace dat je důležitou součástí průzkumné analýzy. Podle McKinney (2017) je seaborn vhodný nejen pro pokročilé uživatele, ale i pro začátečníky, protože již samotný import upraví výchozí vzhled grafů v matplotlib, což zlepší estetiku a čitelnost výstupů. Cielen et al. (2016) zmiňují využití vizualizace v rámci průzkumné fáze a uvádějí tyto knihovny jako vhodné pro práci s datovými grafy jako jsou histogramy nebo krabicové grafy. Pecinovský (2023) zmiňuje, že knihovna matplotlib využívá univerzální GUI, jakými je třeba Tkinter.

Faker je Python balíček určený ke generování falešných, ale realisticky vypadajících dat. Je inspirována obdobnými knihovnami z jiných jazyků a v datové vědě nachází uplatnění zejména při anonymizaci dat, testování aplikací nebo plnění databází syntetickými záznamy. Mezi běžné scénáře použití, které uvádí i oficiální dokumentace, patří například zaplňování vývojových databází a generování dat pro testovací účely.

Princip knihovny spočívá ve využití tzv. providerů – balíčků dat, ze kterých Faker náhodně generuje výstupy. Tyto balíčky mohou být komunitní nebo si uživatel může vytvořit vlastní. Kromě toho knihovna podporuje lokalizaci, takže při generování reflektuje jazykové a kulturní rozdíly – například česká jména, názvy měst nebo formát adres. Podporované datové typy zahrnují jména, adresy, měny, firmy, telefonní čísla, datumy a mnoho dalších.

Faker navíc umožňuje pracovat s realistickým výskytem hodnot. Například častější jména v populaci mají větší pravděpodobnost výskytu, což lze ale volitelně vypnout pro zvýšení výkonu. V takovém případě mají všechny hodnoty stejnou pravděpodobnost výběru (Faker, 2025).

4.4 Syntéza dat pro testovací účely

Vzhledem k tomu, že nebyl nalezen volně dostupný soubor dat vhodný pro testování funkcí navržené aplikace, bylo rozhodnuto využít knihovnu Faker pro generování syntetického testovacího souboru typu big data. Byl vytvořen skript, který využívá lokalizaci této knihovny pro české prostředí, což umožňuje generovat náhodné entity jako jsou firmy, adresy či osoby s realistickými místopisnými údaji. Skript inicializuje české jazykové prostředí knihovny a následně generovaná data vkládá do předem připravených šablon faktur. Numerické hodnoty jsou vytvářeny pomocí pseudonáhodného generátoru, zatímco textová data jsou generována přímo pomocí funkcí knihovny Faker. Výsledné datové soubory jsou ukládány do definované

složky a uživatel má možnost určit počet i typ generovaných výstupů. Program umožňuje vytvářet tisíce souborů během několika sekund, což simuluje zátěžové scénáře odpovídající práci s big daty.

Podobná řešení se v oblasti testování aplikací objevují stále častěji. Carrola (2022) například upozorňuje, že práce s citlivými daty je často omezena přístupovými právy, přestože vývojáři potřebují s daty pracovat. V jeho studii se proto přistoupilo k tvorbě syntetických dat zcela od nuly za použití knihovny Factory Boy, která staví právě na knihovně Faker. Ke stejným závěrům dochází i studie Mazzarino et al. (2023), která se věnuje práci s osobními údaji. I zde byla knihovna Faker využita pro tvorbu syntetických dat s cílem nahradit citlivé informace ve strukturovaných datových sadách. Autoři konstatují, že takto vytvořená data nejen chrání soukromí, ale zároveň si zachovávají sémantickou konzistenci původních datových struktur.

Knihovna Faker je tak dnes běžně využívána nejen v průmyslové praxi, ale i ve vědeckých studiích – a to jak pro testování, tak pro výuku a vývoj datových nástrojů. Rozsah dat, který lze tímto způsobem vytvořit, závisí především na implementaci a výpočetních možnostech cílového systému.

5 NÁVRH MODELOVÝCH POSTUPŮ

Tato kapitola popisuje návrh modelových postupů pro aplikaci zaměřenou na zpracování velkých objemů dat (big data). Cílem je navrhnout aplikaci tak, aby podporovala hned několik formátů jako je Excel, CSV, Word a PDF, přičemž hlavním výstupním formátem je JSON. Tento formát je zvolen nejen pro svou univerzálnost a snadnou čitelnost, ale také pro jeho podporu v moderních databázích. Aplikace bude schopna provádět extrakci dat z různých souborů, jejich validaci, konverzi a v případě potřeby i sloučení více zdrojů do jednoho výstupního souboru.

5.1 Podporované formáty a výstupní struktura

Aplikace bude podporovat práci s několika základními formáty. Každý z těchto formátů má svá specifika, která je nutné při zpracování vzít v potaz. Excelové soubory (XLS, XLSX) jsou široce používány v korporátním prostředí a často obsahují strukturovaná data v tabulkové podobě. Ovšem ne vždy je tomu tak a data je potřeba zpracovat do podoby, kterou lze efektivně využít. Jejich zpracování zahrnuje možnost výběru konkrétních buněk nebo oblastí podle uživatelských preferencí. Pro čtení těchto souborů se budou využívat knihovny jako Pandas a openpyxl, které umožňují efektivní extrakci dat a jejich následné zpracování.

Dalším podporovaným formátem je CSV, který se vyznačuje jednoduchostí a širokou podporou v různých softwarových nástrojích. Při načítání CSV souborů je nutné zohlednit různé oddělovače hodnot (například ; nebo ,). Převod CSV do JSON formátu je relativně jednoduchý, protože klíčovou roli hraje správné mapování jednotlivých sloupců na požadovanou strukturu.

Zpracování PDF a Word dokumentů je oproti předchozím formátům složitější, protože nemají oproti předchozím formátům pevné oddělovače, ani souřadnicovou strukturu. Výsledná data jsou následně transformována do JSON formátu na základě uživatelem zadaných klíčových slov.

JSON jako výstupní formát byl zvolen z několika důvodů. Jeho hlavní výhodou je snadná čitelnost a podpora v široké škále databázových systémů. JSON je také velmi dobře škálovatelný a umožňuje snadné rozšíření o další atributy bez nutnosti změn v datovém modelu.

5.2 Procesy předzpracování a ukládání dat

Zpracování dat v aplikaci bude probíhat v několika na sebe navazujících krocích, které zajistí správnou transformaci vstupních souborů do požadovaného výstupu.

5.2.1 Výběr a načítání souborů

Prvním krokem v procesu transformace dat bude vždy výběr souborů, které mají být použity. Uživatel může vybrat jednotlivé soubory nebo celou složku. Tento proces se může opakovat, protože cesty k souborům budou ukládány do seznamu a uživatel tak bude moci soubory libovolně přidávat. Uživatel bude mít přehled o načtených souborech skrze tlačítko informací, které mu zobrazí informace o počtu načtených souborů a vypíše mu v dialogovém okně několik názvů souborů pro případnou validaci.

5.2.2 Transformace dat

Po vybrání a načtení všech souborů, které chce uživatel transformovat následuje volba způsobu transformace. Těch bude hned několik a budou se dát rozdělit na dva typy. Transformace v rámci převodníkových a slučovacích funkcí a na transformace v rámci hlavních nástrojů aplikace. Pro první zmíněné se bude jednat o to, že vybrané soubory pro transformaci zadá uživatel v rámci nástroje a bez dalšího nastavení zvolí kam výsledné soubory uložit. Zde totiž není vyžadován jakýkoli vstup od uživatele a aplikace tyto převody a změny provádí automaticky. Ovšem pro hlavní prvky aplikace budou nutné uživatelské vstupy pro pochopení toho, co uživatel vyžaduje a zamýšlí dělat s načtenými daty. Zde se aplikace dělí na dvě části a záleží na tom, jaká data chce uživatel v daném kroku zpracovat. Pro Excelovské soubory musí uživatel zvolit požadovaná data pro extrakci, konkrétně buňky, a nastavit výsledné atributy. V rámci textových dokumentů jako je Word nebo PDF musí uživatel zvolit kvůli absenci struktury klíčová slova. Aplikace tak klíčová slova vyhledá v datech a přidělí vzniklým atributům dané hodnoty.

5.2.3 Export a uložení výsledků

Jakmile uživatel absolvoval veškeré potřebné kroky k samotné transformaci dat a transformace se úspěšně dokončila, zbývá výsledek exportovat. To bude uživatel provádět jen za pomocí tlačítka pro uložení, kde zvolí kam si přeje výsledný soubor uložit. Pokud proběhlo zpracování dat korektně, jsou data jen převedena do výsledného formátu JSON a uložena na místo zvolené

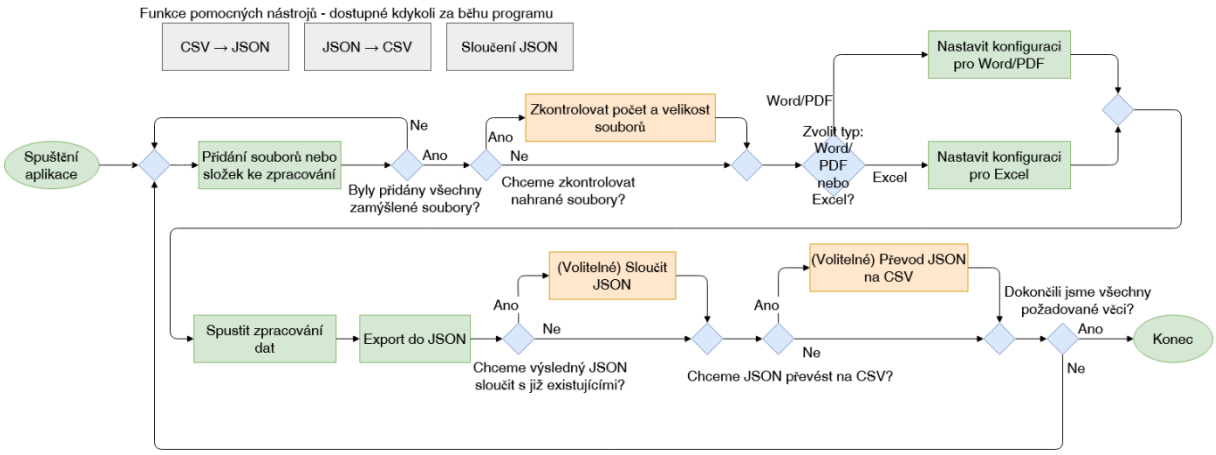
uživatelem. Tento krok by mohl být posledním krokem, jelikož jsou data již převedena a uložena. Uživatel má ovšem možnost s daty dále pracovat i v rámci aplikace. Pokud uvážíme případ, že by uživatel zpracovával data několikrát z několika různých zdrojů, v jiných formátech a podobně, má aplikace funkci slučování JSON souborů. Ta zajistí to, že bude uživatel schopný mít opravdu ve výsledku jen jeden jediný JSON soubor pro případný import do dalších systémů. Pro další využití bude implementován i převod JSON formátu do CSV, aby se ještě více rozšířilo možné využití programu.

5.2.4 Shrnutí funkcí a postupů

Aplikace disponuje tedy celou řadou funkcionalit, které napomáhají hlavnímu cíli, tj. ukládání big dat tím, že data transformují a manipulují s nimi pro dosažení co nejvyšší efektivity. Stěžejní funkcionality aplikace by tak šlo vypsat v následujícím přehledu:

- Převod CSV souborů do JSON formátu
    - Nastavení oddělovačů CSV a desetinného znaménka
- Slučování JSON souborů
- Převod JSON souborů na formát CSV
- Zprocesování tabulkových dat Excel
    - *Nastavení pro práci s Excel soubory*
    - *Export nastavení pro práci s Excel soubory*
    - *Import nastavení pro práci s Excel soubory*
- Zprocesování textových dat Word a PDF
    - *Nastavení pro práci s Word soubory*
    - *Export nastavení pro práci s Word soubory*
    - *Import nastavení pro práci s Word soubory*
- Export procesovaných dat do formátu JSON

Na diagramu (Obrázek 5) jsou vyobrazeny funkcionality aplikace, jejich návaznost a postup při práci se samotným programem v rámci procesu předzpracování dat. Pro tvorbu diagramu byla použita notace Business Process Model and Notation (BPMN). Její použití bylo zvoleno proto, aby potenciální uživatelé z řad podniků mohli lépe pochopit, jak aplikace funguje, a zároveň jednotlivé kroky srozumitelněji zasadit do kontextu podnikových procesů.



Obrázek 5: BPMN diagram procesů fungování aplikace. Zdroj: vlastní.

6 IMPLEMENTACE APLIKACE V PYTHONU

V této kapitole je podrobně popsána implementace aplikace v Pythonu.

6.1 Použité knihovny

Během vývoje aplikace je běžné, že se postupně přidávají nové funkce, které často vyžadují implementaci dalších knihoven pro jejich správné fungování. Tento jev je typický i pro tuto aplikaci. Python nabízí širokou škálu knihoven, které umožňují efektivní řešení různých úkolů. Pro práci s různými typy souborů je nutné využít specializované knihovny, zatímco pro manipulaci se soubory a implementaci specifických algoritmů jsou vyžadovány knihovny odlišného zaměření. Aplikace, určená pro předzpracování a ukládání big dat v různých formátech, jako jsou JSON, CSV, PDF, Word nebo Excel, využívá jak specializované knihovny zaměřené na konkrétní formáty, tak i knihovny širšího rozsahu, například pro práci se soubory či s časovými daty.

6.1.1 Standardní knihovny Pythonu

Knihovna math poskytuje základní matematické funkce, které zahrnují výpočty exponenciálních, logaritmických a trigonometrických operací (Python, 2025d). V aplikaci se využívá hlavně pro výpočty související s převodem velikosti souborů do čitelnější podoby, konkrétně funkcí *math.log* a *math.floor*, které se používají k přepočtu velikosti souborů z bajtů na kilobajty, megabajty a další jednotky. Díky tomu může aplikace uživateli zobrazit souhrnné informace o souborech v přehledném formátu.

Knihovna Tkinter je používaným nástrojem pro tvorbu grafických uživatelských rozhraní v Pythonu a poskytuje podklad pro tvorbu oken, tlačítek, textových polí a dalších interaktivních prvků (Python, 2025h). V aplikaci hraje zásadní roli, protože umožňuje uživateli pohodlnou interakci s aplikací. Pomocí Tkinter bylo vytvořeno hlavní okno aplikace, ve kterém uživatel může vybírat soubory i složky pro zpracování, nastavovat parametry aplikace a zobrazovat informace o zpracovaných souborech. Dále byl použit pro vyskakovací okna zobrazující upozornění a chybové hlášky. Kromě toho aplikace využívá *tkinter.Toplevel()* pro vytvoření samostatných dialogových oken, například pro zobrazení informací o vybraných souborech.

Modul tkinter.ttk je rozšířením Tkinter, které přináší vylepšené a modernější vzhledy widgetů a jejich širší možnosti konfigurace. V aplikaci byl použit zejména pro vytvoření progress baru,

který uživateli vizuálně ukazuje průběh zpracování souborů. Tento prvek je důležitý zejména při práci s velkým množstvím dat, protože poskytuje uživateli zpětnou vazbu o průběhu procesu a zajišťuje, že aplikace během výpočtů nebudí dojem „zamrznutí“.

Knihovna time umožňuje práci s časovými údaji, včetně získávání aktuálního času a měření časových intervalů (Python, 2025g). V aplikaci je využita především k měření doby trvání jednotlivých operací, jako je zpracování souborů. Při spuštění procesu se zaznamená časový údaj pomocí *time.time()* a po dokončení se opětovně načte aktuální čas, přičemž rozdíl mezi těmito hodnotami slouží k výpočtu doby trvání zpracování. Tento údaj se následně zobrazí uživateli jako informace o tom, jak dlouho celý proces trval.

Modul os poskytuje funkce pro interakci s operačním systémem, včetně manipulace se soubory a adresáři (Python, 2025e). V aplikaci se využívá zejména k načítání souborů ze složek, kde umožňuje rekurzivní procházení adresářové struktury a získávání cest k souborům pomocí *os.walk()*. Dále umožňuje získávání informací o velikosti souborů pomocí *os.path.getsize()*, což je zásadní pro výpočet celkové velikosti vybraných datových souborů, která se zobrazuje uživateli.

Knihovna JSON slouží k práci s formátem JSON a umožňuje kódování i dekódování tohoto formátu (Python, 2025c). V aplikaci se tato knihovna využívá k načítání a ukládání dat ve formátu JSON, což umožňuje export zpracovaných souborů do strukturované podoby. JSON soubory jsou vytvářeny na základě extrahovaných nebo vložených dat z různých formátů, jako jsou PDF, Word, Excel nebo CSV. Zároveň se formát JSON využívá pro ukládání a načítání uživatelské konfigurace.

Modul re umožňuje práci s regulárními výrazy, které se často používají pro pokročilé zpracování textu, hledání vzorů a extrakci konkrétních informací (Python, 2025f). V aplikaci je využíván pro extrakci dat z textu na základě klíčových slov, kde pomocí regulárních výrazů vyhledává konkrétní hodnoty a přiřazuje je odpovídajícím atributům.

Knihovna datetime poskytuje nástroje pro práci s datovými a časovými hodnotami (Python, 2025b). V aplikaci se využívá především pro generování časových razítek, která se používají k logování nezpracovaných souborů. Každý záznam o nezpracovaných souborech je v rámci názvu opatřen časovým údajem, což nám poskytuje snadný přehled o tom, kdy daný záznam vznikl.

Modul CSV umožňuje čtení a zápis souborů ve formátu CSV (Python, 2025a). V aplikaci je využit pro převod dat z JSON do CSV, což umožňuje export strukturovaných dat do formátu s pevným oddělovačem, který je běžně využíván v mnoha dalších aplikacích pro zápis dat.

6.1.2 Externí knihovny Pythonu

Knihovna Pandas je jedním z nejvýznamnějších nástrojů pro analýzu a manipulaci s daty v Pythonu. Poskytuje výkonné struktury, jako je DataFrame, které umožňují efektivní práci s tabulkovými daty (McKinney, 2017; Pandas, 2024). V aplikaci se využívá hlavně k načítání a zpracování CSV a Excel souborů, kde následně umožňuje jejich konverzi do strukturovaného JSON formátu. Díky tomu mohou být data snadno ukládána do NoSQL databází nebo dalších systémů.

Knihovna fitz (PyMuPDF) je rozhraní pro MuPDF a umožňuje čtení, analýzu a extrakci textu z PDF souborů (PyMuPDF, 2025). V aplikaci je využívána k extrakci textu z PDF dokumentů, což je zásadní při práci s fakturami a jinými dokumenty obsahujícími klíčové informace. Aplikace dokáže z těchto dokumentů získat údaje, které uživatel požaduje a za pomoci již dříve zmíněných knihoven a modelů jako je re dokáže z extrahovaného textu získat požadovaná data.

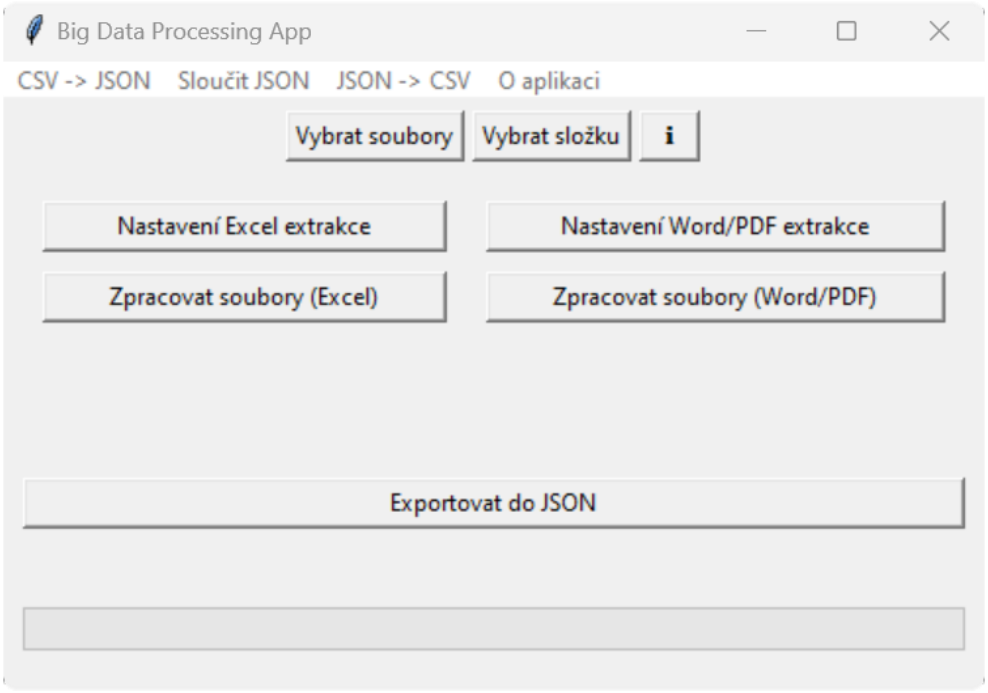
Modul docx (python-docx) umožňuje práci s dokumenty ve formátu Microsoft Word (.docx) (Python-docx, 2024). V aplikaci se využívá k extrakci textových dat z Word dokumentů, přičemž získaná data jsou následně analyzována a strukturována do JSON formátu. Využívá se zde taktéž podobného principu jako u PDF souborů a nedílnou součástí procesu jsou další knihovny.

Knihovna openpyxl je standardním nástrojem pro práci s Excel soubory ve formátu .xlsx (Openpyxl, 2024). Aplikace ji využívá k extrakci hodnot z jednotlivých buněk v Excelu na základě uživatelem zadaných parametrů. To umožňuje specifikovat, která data mají být vybrána a v jakém formátu mají být uložena. Tímto způsobem je uživatel schopen specifikovat, jaká data jsou důležitá. Je to klíčový aspekt pro redukci objemu dat.

6.2 Uživatelské prostředí aplikace

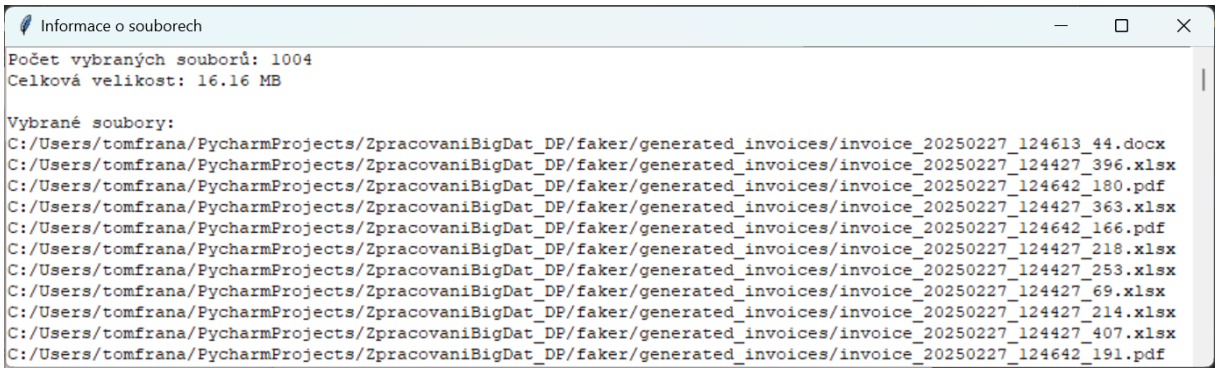
Uživatelské rozhraní aplikace je navrženo s důrazem na jednoduchost a přehlednost ovládání. Hlavní okno aplikace umožňuje uživateli snadno vybírat soubory, nastavovat parametry práce se soubory a spouštět jednotlivé operace pomocí přehledně rozložených ovládacích prvků. V horní části okna se nachází menu s několika záložkami, které umožňují přístup k hlavním

funkcím aplikace (Obrázek 6). Jedná se o záložky: „*CSV → JSON*“ pro převod CSV souborů do formátu JSON. „*Sloučit JSON*“ pro slučování více JSON souborů do jednoho souhrnného. „*JSON → CSV*“ pro převod dat z JSON formátu do CSV. A poslední „*O aplikaci*“, kde se nachází sekce poskytující informace o verzi aplikace, autorovi a je zde dostupný kontakt.



Obrázek 6: Uživatelské rozhraní aplikace. Zdroj: vlastní.

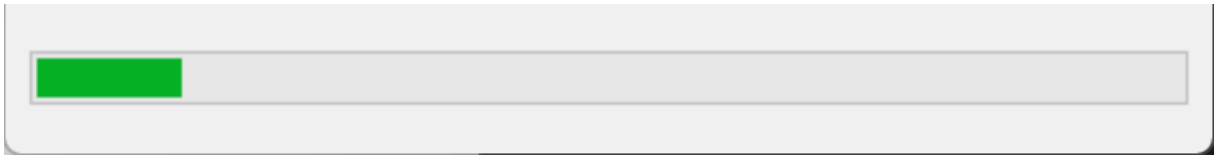
Pod hlavním menu se nachází hlavní panel aplikace, kde se nachází ovládací prvky pro hlavní funkce aplikace. Nahoře jsou tlačítka „*Vybrat soubory*“, „*Vybrat složku*“ a tlačítko informací. Skrze první dvě tlačítka může uživatel vybrat libovolné množství souborů, které chce do aplikace nahrát. Uživatel má možnost pomocí výběru více souborů vybrat konkrétní soubory, nebo pro potřeby velkého množství dat zvolit výběr složky. Takto může zvolit libovolný počet cest ke složkám i souborům, vše je ukládáno do seznamu souborů. Následně může uživatel skrze tlačítko informací, označené písmenem *i*, zjistit, zda se správně načetly všechny soubory. Samozřejmě pro velké množství dat nelze zkontrolovat každý jeden soubor, proto výpis obsahuje na prvním místě údaje o počtu načtených souborů a jejich celkovou velikost (Obrázek 7). Uživatel si tak ověří, že se načetl takový počet dat, který zamýšlel.



Obrázek 7: Informace o souborech. Zdroj: vlastní.

Následující prvky slouží pro uživatelské nastavení práce se soubory. Na levé straně jsou ovládací prvky pro práci s tabulkovými daty ve formátu Excel (.xls, .xlsx) a na pravé straně jsou ovládací prvky pro práci s textovými formáty typu Word (.docx) a PDF. Pod tlačítky k nastavení extrakce se nachází tlačítka pro zpracování souborů. Tlačítka jsou oddělená kvůli principům fungování jednotlivých metod, které budou popsány blíže v následující kapitole, která se bude zabývat právě popisem samotných funkcí aplikace.

Ve spodní části aplikace se nachází dva prvky. První je tlačítko „*Exportovat do JSON*“, které slouží k uložení dat do formátu JSON pro případné další použití. Po stisknutí tlačítka je uživatel vyzván k vybrání místa pro uložení. Druhým prvkem je tzv. progress bar (Obrázek 8), který slouží k ukazování průběhu zpracování dat. Jak se postupně bude zpracovávat soubor dat vložených do aplikace, bude se více zabarvovat zeleně. Při dosažení 100 % se uživateli zobrazí informace o úspěšném dokončení a výsledný soubor bude uložen.



Obrázek 8: Progress bar. Zdroj: vlastní.

Aplikace využívá jednoduché grafické rozhraní založené na knihovně Tkinter, které poskytuje základní vizuální prvky pro práci s daty. Všechny hlavní funkce jsou dostupné přímo z úvodní obrazovky, což usnadňuje použití i pro méně zkušené uživatele.

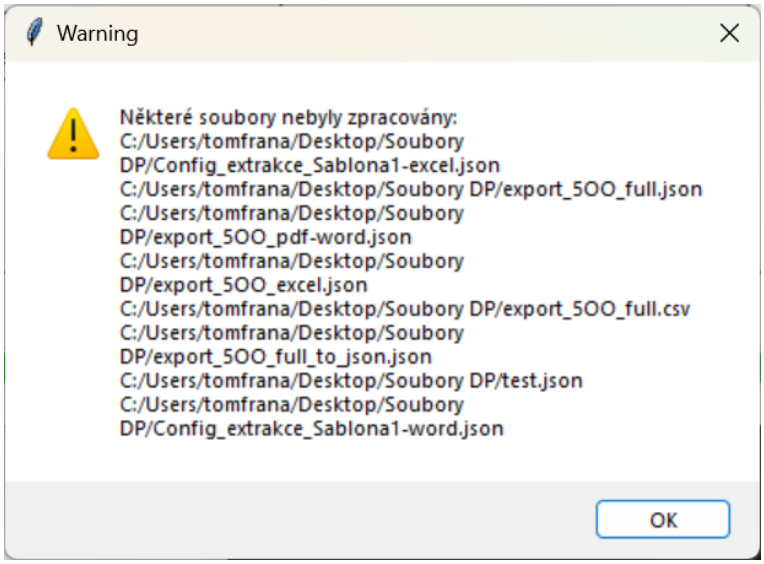
6.3 Funkce aplikace

Hlavním cílem aplikace je nabídnout jednoduchý, otevřený a moderní nástroj v jazyce Python, který je určen pro práci se soubory a zaměřuje se na předzpracování a ukládání big dat. Aplikace

je navržena tak, aby poskytla efektivní a snadno použitelný nástroj pro uživatele, kteří potřebují pracovat s velkými soubory dat. V souladu s předchozí kapitolou a na základě vymezených potřeb byly identifikovány klíčové funkce, které jsou implementovány do aplikace.

6.3.1 Kontrola vstupních dat

Tato funkcionalita, na rozdíl od ostatních, nebyla popsána v předchozích kapitolách, protože se jednalo jen o výčet funkcionalit, které vidí samotný uživatel. V rámci algoritmů je toho v programu podstatně více. Například se počítá s ověřováním vstupních souborů. Pakliže uživatel nahraje jakkoli vadný soubor (nepodporovaný typ, vadná a nečitelná data), aplikace je chráněna, a uživatel je o této chybě informován pomocí chybového logu a chybovou hláškou (Obrázek 9). Tudíž je schopen identifikovat data, která jsou poškozená, a nejsou ve výsledném exportu. Uživatel je chybovými hláškami nebo oznamovacími okny upozorňován prakticky v celé aplikaci. Nemělo by se tedy stát, že uživatel zadá zpracování dat bez toho, aby je nahrál. Uživatel je také ujišťován v případě správného importu nastavení jak pro Excel, tak i Word soubory a podobně. Aplikace se snaží minimalizovat lidskou chybu, ale nezavrhuje lidský faktor a uživateli dává plnou moc nad tím, jak s daty pracovat.



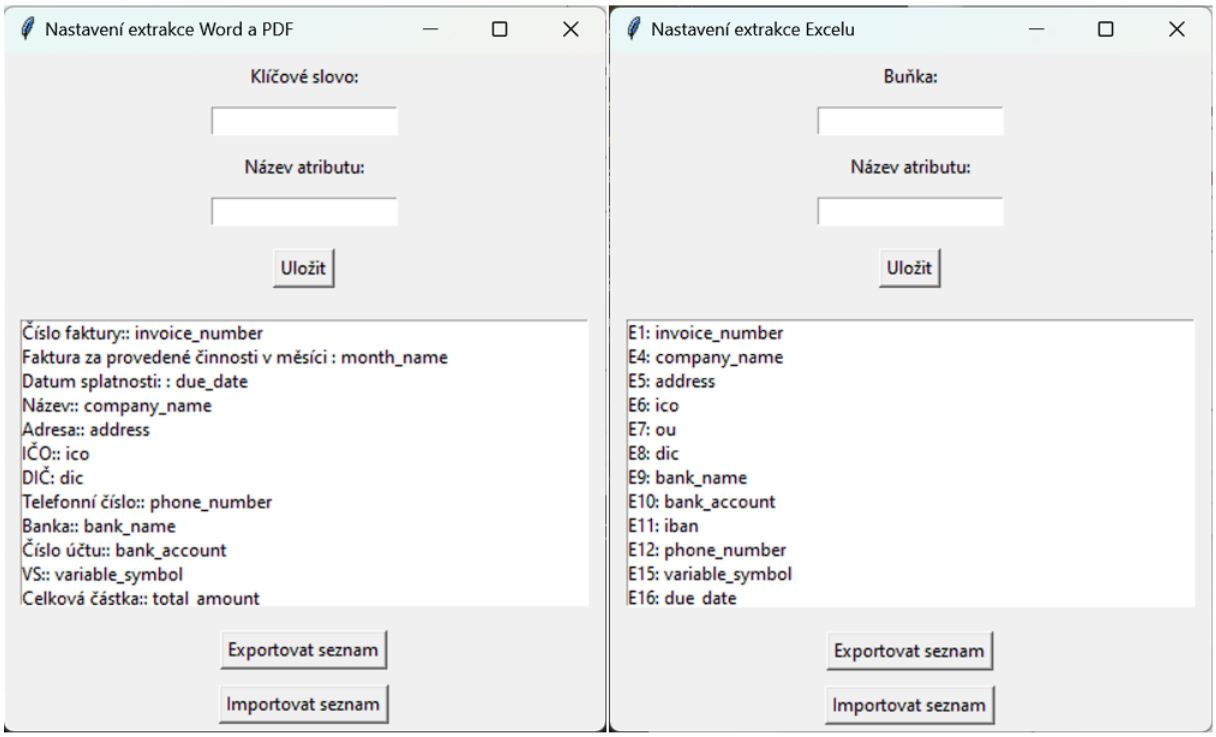
Obrázek 9: Chybové hlášení aplikace. Zdroj: vlastní.

6.3.2 Zpracování Excel souborů

Při práci s Excel soubory se využívá knihovna Pandas ve spojení s openpyxl, což umožňuje načítání, manipulaci a přípravu dat pro následný export do formátu JSON. Pro uživatele je práce s touto funkcionalitou velice jednoduchá. Předpokládá se, že vstupní soubory jsou již korektně

nahrány do aplikace. Uživatel poté skrze jednoduchý formulář nastaví parametry pro práci se soubory. Aplikace po uživateli vyžaduje v případě Excel souborů pouze kód buňky a název atributu, pod kterým budou následně data uložena ve výstupním formátu. Aplikace kontroluje správnost formátu zadaných buněk. U atributu záleží na rozhodnutí uživatele, jak atribut pojmenuje a jak bude následně uložen ve výstupním formátu. Uživatel má samozřejmě možnost tento seznam uložit (exportovat), aby ho mohl použít pro stejný typ souborů v budoucnu (Obrázek 10).

Pro načtení stačí zvolit možnost importu a nahrát konfigurační soubor ve formátu JSON. Konfigurační soubor je tedy možné upravovat i mimo samotnou aplikaci, což umožňuje větší svobodu pro zkušené uživatele. Díky této kombinaci lze načíst uživatelem definované oblasti tabulek, přizpůsobit jejich strukturu a pracovat s nimi. Samozřejmě je lze exportovat do JSON tak, aby odpovídaly požadovaným formátům pro další analýzu či integraci do dalších systémů.



Obrázek 10: Nastavení extrakce pro různé typy souborů. Zdroj: vlastní.

6.3.3 Zpracování Word a PDF souborů

Při zpracování souborů formátů Word a PDF se využívají knihovny python-docx a PyMuPDF, které umožňují extrahovat data ze souborů .docx a .pdf a stejně jako u Excel souborů je díky nim možné připravit data pro následný export do formátu JSON. Zde je postup velice podobný tomu, který uživatel vykonává u Excel souborů. Předpokládá se, že data jsou načtena správně.

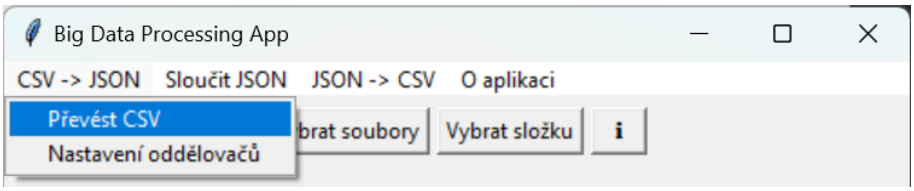
Uživatel otevře nastavení pro práci s Word a PDF soubory pomocí příslušného tlačítka. Zde oproti Excel nastavení nenastavuje buňky, řádky, ani nic podobného, ale klíčová slova. Klíčová slova jsou následně díky regulérním výrazům ve zpracovaném textu nalezena a dané hledané hodnoty uloženy. Klíčová slova byla vybrána kvůli většímu uživatelskému pohodlí – uživatel pouze zadá klíčová hledaná slova a název atributu podle jeho volby a aplikace nalezne odpovídající hodnoty a uloží je. Obrázek 10 zachycuje rozhraní pro tato nastavení.

Díky tomu, že se skrze knihovny převádí a ukládají soubory typu Word a PDF do textové podoby, je v těchto datech následně možné vyhledávat skrze již zmíněné výrazy. Výstupem je tedy stejný formát jako pro Excel soubory, kdy se uloží uživatelem zadaný atribut a k němu náležící hodnota z dokumentu, nalezena skrze klíčová slova. Samozřejmě je zde naprosto stejná možnost exportu, importu i úpravy konfiguračního souboru, pro případné další využití na obdobná data v budoucnu.

6.3.4 Pomocné nástroje

Nedílnou součástí aplikace jsou také pomocné nástroje. Ty slouží jako důležité prvky aplikace. Nachází se v horním menu aplikace a nabízí hlavně manipulaci se strukturovanými typy dat. První funkce, kterou aplikace nabízí, je prostý, ale užitečný převod, ze strukturovaného formátu CSV do formátu JSON. Aplikace tuto funkcionalitu nabízí kvůli rychlému a dostupnému převodu pro potřeby uživatele, aby nemusel využívat externích řešení. Pokud by uživatel totiž zvolil cizí řešení, které by například nepodporovalo český jazyk, ztratil by většinu hodnoty dat a data by ve své podstatě zcela znehodnotil. V rámci aplikace by se takové věci stávat neměly a pro bezpečnou práci s tímto typem souborů byla funkce zavedena.

Další velice podstatnou funkcí je slučování JSON souborů. Zde se vychází z praxe, kde budou dostupná předzpracovaná data z mnoha souborů jiných formátů, která ovšem bude uživatel chtít mít na jednom místě a v jednom celku. K tomu slouží právě tato funkce. Zvolené JSON soubory sloučí do jednoho velkého souboru, který je následně připravený pro další práci nebo využití v dalších systémech. Poslední funkcionalitou, která byla do aplikace přidána je funkce pro převod JSON formátu do CSV formátu (Obrázek 11). Zde se předpokládá využití v aplikacích, ve kterých je CSV vhodnějším formátem. To je například Microsoft Excel či jiné tabulkové editory, kde v případě objemů dat zvládnutelnými těmito aplikacemi lze nad daty provádět různé analýzy, ať už se jedná o Excel nebo jiné aplikace jako PowerBI, ve kterých lze s těmito daty pracovat. Tato funkce je tedy vhodná pro kancelářskou práci. Kromě toho se také jedná o významné snížení objemu dat kvůli rozdílným vlastnostem JSON a CSV.



Obrázek 11: Převod na CSV. Zdroj: vlastní.

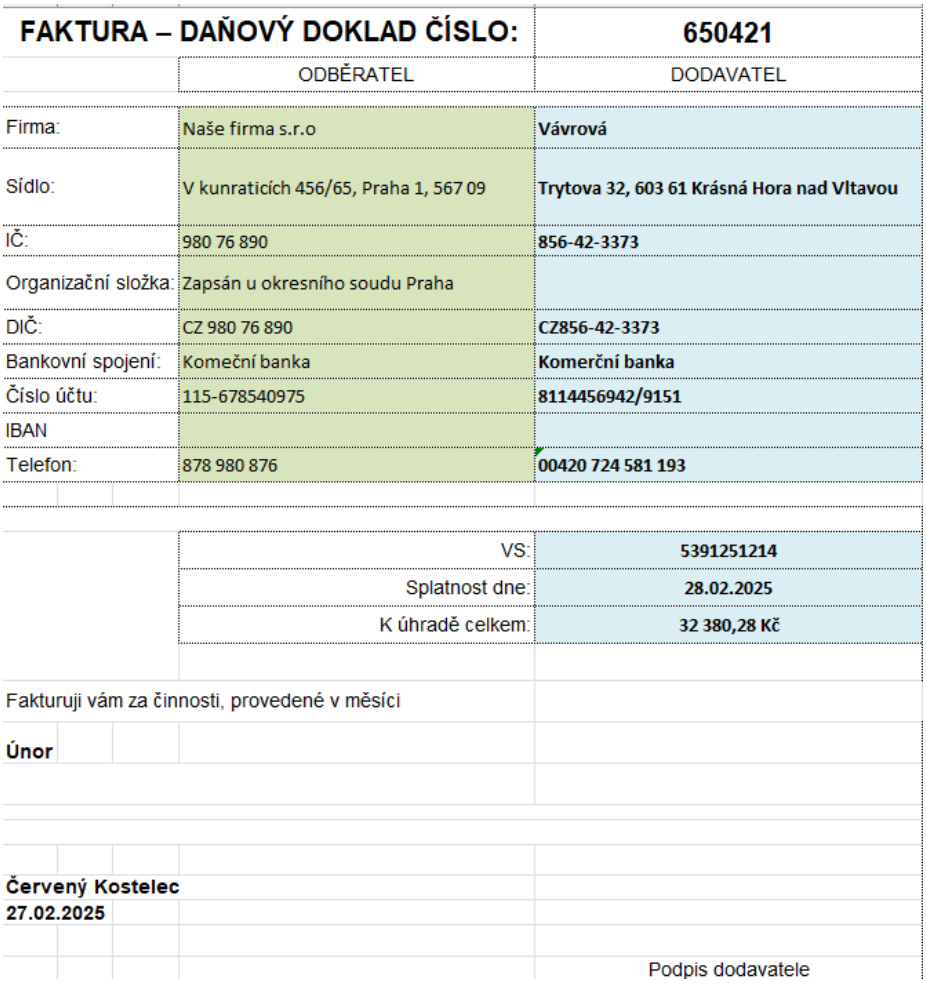
7 VYHODNOCENÍ VÝSLEDKŮ A DISKUSE

V této kapitole jsou zhodnoceny výsledky diplomové práce v podobě navržené aplikace a dále diskutovány její přínosy a možná rozšíření.

7.1 Ukázka výstupu aplikace

Tato kapitola se zaměřuje na konkrétní ukázku zpracování dat a demonstruje celý proces od načtení vstupního souboru až po generování finálního výstupu ve formátech, které jsou uloženy a připraveny k dalšímu využití. Cílem této ukázky je ilustrovat, jak aplikace efektivně extrahuje klíčová data z velkého množství dokumentů a převádí je do strukturované podoby, ve které jsou následně uložena a mohou být dále použita pro analýzy, reportování nebo integraci s jinými systémy.

Obrázek 12 zobrazuje jeden z tisíců fakturačních dokumentů, které aplikace dokáže zpracovávat. Tento dokument obsahuje řadu důležitých informací, jako jsou název dodavatele, datum vystavení faktury, datum splatnosti, celková částka a další relevantní údaje, které jsou nezbytné pro účetní, daňovou evidenci nebo finanční analýzu. Uživatel zadá do aplikace požadované parametry, které definují klíčové položky pro extrakci, a spustí proces zpracování dat. Aplikace následně automaticky detekuje relevantní údaje, které jsou extrahovány a převedeny do požadovaného formátu, čímž se výrazně zjednodušuje a zrychluje celý proces zpracování faktur.



FAKTURA – DAŇOVÝ DOKLAD ČÍSLO:		650421
	ODBĚRATEL	DODAVATEL
Firma:	Naše firma s.r.o	**Vávrová**
Sídlo:	V kunraticích 456/65, Praha 1, 567 09	**Trytova 32, 603 61 Krásná Hora nad Vltavou**
IČ:	980 76 890	**856-42-3373**
Organizační složka:	Zapsán u okresního soudu Praha	
DIČ:	CZ 980 76 890	**CZ856-42-3373**
Bankovní spojení:	Komeční banka	**Komerční banka**
Číslo účtu:	115-678540975	**8114456942/9151**
IBAN		
Telefon:	878 980 876	**00420 724 581 193**

VS:	**5391251214**
Splatnost dne:	**28.02.2025**
K úhradě celkem:	**32 380,28 Kč**

Fakturuji vám za činnosti, provedené v měsíci

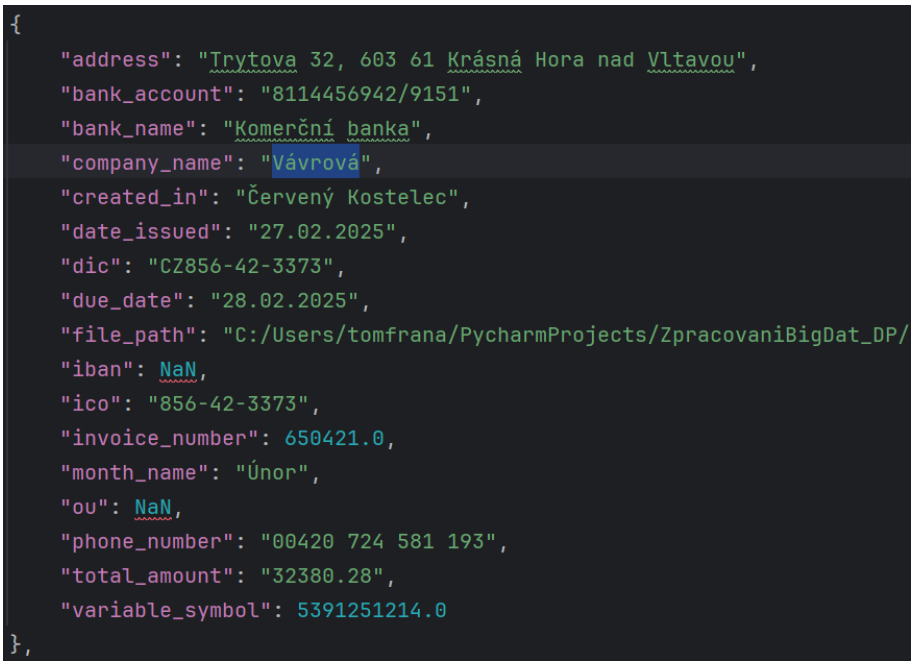
Únor

Červený Kostelec
27.02.2025

Podpis dodavatele

Obrázek 12: Ukázka vstupních dat ve formě faktury. Zdroj: vlastní.

Jakmile aplikace dokončí zpracování všech faktur, výsledná data jsou převedena uživatelem do formátu JSON, jak ukazuje Obrázek 13. Formát JSON je ideální pro ukládání a následné automatizované zpracování dat, neboť poskytuje jasně strukturovanou reprezentaci informací, která umožňuje snadnou integraci s databázemi či analytickými nástroji. V JSON výstupu jsou zobrazeny všechny klíčové údaje z faktury, a to v souladu s konfigurací uživatele. Každá extrahovaná hodnota je uložena pod odpovídajícím klíčem, což značně usnadňuje následnou práci s daty. Kromě toho je do výstupu přidán atribut obsahující cestu k původnímu souboru, což umožňuje pozdější sledování původu dat. Tento atribut je možné při analýze v případě potřeby odstranit.



Obrázek 13: Převod faktury do formátu JSON. Zdroj: vlastní.

Obrázek 14 zobrazuje stejná data, tentokrát však převedená do formátu CSV, což umožňuje jejich snadnou manipulaci v tabulkových programech, jako je Microsoft Excel nebo Google Sheets. Formát CSV je široce využíván především v případech, kdy je potřeba sdílet data mezi různými systémy nebo provádět analýzu ve známém prostředí, které uživatelům usnadňuje práci. Tento formát je také vhodný pro rychlou manipulaci s daty, protože je méně náročný na úložný prostor než JSON, což ho činí efektivním pro práci s rozsáhlými soubory. Formát CSV je navíc snadno čitelný pro běžné kancelářské aplikace, což znamená, že není potřeba specializovaného softwaru pro jeho otevření a analýzu.

address	bank_account	bank_name	company_name	created_in	date_issued	dic	due_date
Podohradská 862, 247 78 Kunovice	7759336892/1084	Komerční banka	Kratochvílová k.s.	Bystřice pod Hostýnem	27.02.2025	CZ686-52-4350	28.02.2025
Havlíčkova 4, 607 41 Kamenice nad Lipou	8049736545/5867	UniCredit Bank	Novák Bartoš v.o.s.	Zbiroh	27.02.2025	CZ586-73-7935	28.02.2025
Hnězdenská 59, 380 58 Mirovice	4087668252/7263	MONETA Money Bank	Bartošová	Liberec	27.02.2025	CZ591-87-2634	28.02.2025
Durychova 4, 426 82 Letohrad	9244444630/7609	Česká spořitelna	Matějka Stejskal v.o.s.	Police nad Metují	27.02.2025	CZ051-89-8977	28.02.2025
Trytova 32, 603 61 Krásná Hora nad Vltavou	8114456942/9151	Komerční banka	Vávrová	Červený Kostelec	27.02.2025	CZ856-42-3373	28.02.2025
Palackého 315, 354 51 Nová Paka	1147200443/4854	ČSOB	Urban Bílek o.s.	Hluboká nad Vltavou	27.02.2025	CZ504-83-5778	28.02.2025
Cerhenická 447, 134 70 Švihov	8162310063/4257	Komerční banka	Kovářová	Rosice	27.02.2025	CZ893-98-9613	28.02.2025
Rudoltická 7, 388 71 Vizovice	4130013146/1053	Česká spořitelna	Hájek	Kašperské Hory	27.02.2025	CZ183-53-2900	28.02.2025
Náměstí Prezidenta Masaryka 96, 290 88 Přebuz	9923983983/6713	Česká spořitelna	Nováková Svobodová k.s.	Hrádek	27.02.2025	CZ500-69-7825	28.02.2025
Moravanů 4, 654 53 Benešov nad Ploučnicí	8074189857/2383	UniCredit Bank	Čermák Bednář s.r.o.	Jilemnice	27.02.2025	CZ008-46-8447	28.02.2025
Jeruzalémská 26, 528 04 Netolice	9979011537/8161	UniCredit Bank	Toman v.o.s.	Chvaletice	27.02.2025	CZ821-89-9455	28.02.2025
Vokrojova 35, 680 30 Kouřim	5335149134/6388	Air Bank	Holubová Fialová a.s.	Zruč nad Sázavou	27.02.2025	CZ997-91-1145	28.02.2025
Lebeděvova 572, 542 81 Stráž pod Ralskem	3993471199/7374	Fio banka	Soukupová a.s.	Nasavrky	27.02.2025	CZ356-78-7817	28.02.2025
Lovčenská 8, 486 61 Horní Jiřetín	8108448622/7168	UniCredit Bank	Valenta k.s.	Semily	27.02.2025	CZ892-85-3419	28.02.2025
Palackého 4, 159 24 Městec Králové	2268684134/3342	Raiffeisenbank	Bartošová	Smečno	27.02.2025	CZ550-79-9348	28.02.2025
Teplická 4, 425 42 Jihlava	3846559224/9752	Česká spořitelna	Veselý	Slatiňany	27.02.2025	CZ293-48-0483	28.02.2025
Vajgarská 54, 417 53 Česká Skalice	6501568259/2129	Česká spořitelna	Ježek	Desná	27.02.2025	CZ144-75-2307	28.02.2025
K Dýmači 967, 698 47 Bělá nad Radbuzou	5626711910/2315	UniCredit Bank	Vlčková	Konice	27.02.2025	CZ818-04-4602	28.02.2025
Smolenská 659, 341 65 Lázně Kynžvart	7248276523/7891	UniCredit Bank	Lišková	Ivanovice na Hané	27.02.2025	CZ923-44-6733	28.02.2025
Františka Černého 35, 275 84 Bechyně	7659338944/7945	Fio banka	Švecová	Plzeň	27.02.2025	CZ128-99-4479	28.02.2025
Slatiňanská 72, 647 30 Svoboda nad Úpou	7504159016/1624	Komerční banka	Štěpánková Pospíšil k.s.	Hostouň	27.02.2025	CZ651-28-0466	28.02.2025
Libuňská 41, 192 71 Studénka	3078446756/4911	UniCredit Bank	Stejskal	Nechanice	27.02.2025	CZ662-67-6336	28.02.2025

Obrázek 14: Převod faktury do formátu CSV. Zdroj: vlastní.

Tento proces ukazuje přínos aplikace při práci s velkým objemem dat. Uživatel má možnost vybrat si mezi formátem JSON a CSV v závislosti na svých potřebách a požadavcích cílových systémů. Možnost volby výstupního formátu poskytuje flexibilitu, která uživatelům zajišťuje maximální efektivitu při integraci a sdílení dat. Zpracování probíhá rychle a efektivně, což výrazně šetří čas, který by byl jinak vynaložen na manuální přepisování informací z dokumentů. Díky této automatizaci se snižuje riziko lidské chyby a zvyšuje se celková produktivita při práci s big daty.

7.2 Redukce velikosti souborů dat

Jedním z klíčových přínosů aplikace je její schopnost podstatně snížit velikost zpracovávaných datových souborů. Při převodu dokumentů, jako jsou faktury ve formátu PDF, Excel tabulky nebo Word dokumenty, se často vyskytuje velké množství metadat, formátovacích prvků a jiných informací, které nejsou relevantní pro další analýzu. Aplikace se proto zaměřuje na extrakci pouze těch údajů, které uživatel specifikuje, což umožňuje dosáhnout značného zmenšení výsledného objemu dat.

Pro lepší představu o efektivitě tohoto procesu byla provedena analýza čtyř modelových případů, ve kterých byl zaznamenán počáteční objem dat před zpracováním a finální velikost výstupního souboru. Tyto výsledky shrnuje následující Tabulka 5. Zároveň je nutné zmínit, že analýza může být ovlivněna použitým hardwarem a konfigurací zařízení obecně. Modelové případy byly realizovány v podmínkách operačního systému Windows 11 Pro 23H2 s procesorem AMD Ryzen 5 7530U 2 GHz s integrovanou grafickou kartou, 16 GB operační paměti a úložištěm Samsung MZAL4512HBLU-00BL2 M.2 NVMe SSD s kapacitou 512 GB.

Tabulka 5: Analýza efektivity zpracování dat – srovnání počátečního a výsledného objemu dat. Zdroj: vlastní.

Zpracovávané soubory	**Objem dat**	**Výsledný objem**	**Snížení**
500 Excel souborů (Šablona 1) 300 Word a 200 PDF souborů (Šablona 1)	16,2 MB	JSON 700 kB CSV 204 kB	JSON -96 % CSV -98 %
10 000 Excel souborů (Šablona 3)	11,71 GB	JSON 6,7 MB CSV 2,96 MB	JSON -99,94 % CSV -99,98 %
10 000 Excel souborů (Šablona 3) 10 000 Word souborů (Šablona 1)	11,86 GB	JSON 13 MB CSV 5,94 MB	JSON -99,89 % CSV -99,95 %
10 000 Excel souborů (Šablona 3) 10 000 Word souborů (Šablona 1) 10 000 PDF souborů (Šablona 1)	12,13 GB	JSON 16,1 MB CSV 7,92 MB	JSON -99,87 % CSV -99,94 %

Z tabulky je patrné, že aplikace dosahuje úspory velikosti souborů i o více jak 99 %, což představuje skutečně významné snížení objemu dat při zachování jejich důležité informační hodnoty. Šablony typu 1 obsahují pouze základní grafické prvky, a i přesto je zde vidět výrazná úspora velikosti. Šablony typu 3 obsahují kromě základních prvků také obrázek simulující logo společnosti. Zde je vidět ještě razantnější snížení objemu dat. To dokazuje, jak neefektivní je ukládat data v těchto formátech, protože jejich metadata a formátovací prvky zabírají až příliš místa. Tímto způsobem lze ušetřit nejen úložný prostor, ale také snížit požadavky na přenos dat, což může být zásadní při práci s velkými soubory v cloudových řešeních nebo databázových systémech.

Výše uvedené výsledky potvrzují efektivitu aplikace při optimalizaci dat a ukazují, že její použití může přinést značné výhody v oblastech, kde je potřeba pracovat s velkými objemy dokumentů a zajistit jejich rychlé a efektivní zpracování. Mimo to je lze použít i na menší objemy dat, kde nemusí být úspora dat tak zásadní, ale aplikace jistě urychlí zpracování podobných souborů, a to nejen faktur, ale i dalších typů dokumentů.

7.3 Zasazení do kontextu životního cyklu big dat

Moderní datové ekosystémy zahrnují komplexní infrastruktury, ve kterých jsou data sbírána, ukládána, zpracovávána a analyzována (Johnson et al., 2017). Efektivní ekosystém big dat by měl být modulární a škálovatelný, aby umožňoval práci s daty v reálném čase a podporoval horizontální škálování úložišť (Khan et al., 2014). Cassandra a DynamoDB jsou příklady NoSQL databází, které byly vyvinuty právě pro potřeby vysoce distribuovaných ekosystémů, kde je nutné zpracovávat obrovské objemy dat s nízkou latencí (Carpenter a Hewitt, 2022). V tomto kontextu je významné využití sloupcově orientovaných formátů, jako je Apache Parquet, který optimalizuje čtení datových bloků a minimalizuje vstupně-výstupní operace v distribuovaných systémech (Vohra, 2016). Tím se propojuje efektivní správa dat s jejich následným využitím v analytických nástrojích, jako je Apache Spark (Singh, 2025).

Životní cyklus dat zahrnuje vytváření, sběr, ukládání, zpracování, analýzu a archivaci dat, přičemž každá z těchto fází přináší specifické výzvy (Naeem et al., 2022). V kontextu big dat je zvlášť důležité efektivní řízení ukládání a přístupu k datům, což umožňuje jejich optimalizované využití v dalších fázích životního cyklu (Jin et al., 2015). Výše navržená aplikace se stává důležitým prvkem zejména ve fázi předzpracování a ukládání, kde hraje roli při přípravě dat, odstranění redundancí a optimalizaci formátů pro jejich další analýzu (Upadhyay et al., 2021). Aplikace zároveň může integrovat prvky z moderních metodik a technologických ekosystémů a fungovat jako vstupní vrstva pro další analytické systémy. Díky tomu nejen usnadňuje proces předzpracování a ukládání dat, ale zároveň umožňuje jejich efektivní využití v kontextu celého životního cyklu dat (Rahul a Banyal, 2020).

7.4 Budoucí rozšíření aplikace

Přestože aplikace již v současnosti plní svou hlavní funkci efektivního podpůrného programu pro ukládání big dat, existuje několik možných směrů, jak ji v budoucnu dále rozšířit. Tyto možnosti zahrnují rozšíření funkcionalit na základě zpětné vazby od uživatelů anebo propojení s dalšími systémy, které by umožnilo její širší nasazení v datové infrastruktuře podniků či vědeckých institucí. Jedná se například o přímý přenos dat skrze moduly či API do jiných systémů.

7.4.1 Další funkce na základě požadavků uživatelů

Další vývoj aplikace by měl být primárně řízen potřebami uživatelů a požadavky, které vyplynou z jejího reálného nasazení. Mezi potenciální nové funkcionality patří:

- **Podpora nových datových formátů** – možnost přímé práce s XML, Parquet, Avro nebo připojení k databázovým NoSQL systémům. Možnost zapracování méně populárních typů souborů, které se v praxi využívají pro dokumenty.
- **Automatizované detekce a kategorizace souborů** – aplikace by mohla automaticky rozpoznávat typy dokumentů a podle toho aplikovat odpovídající pravidla pro extrakci a uložení dat. Tato funkcionalita by byla jednoduše aplikovatelná, ale záleží na odezvě od uživatelů.
- **Zlepšená kontrola dat** – Program by v budoucnu mohl aktivně vyhledávat duplicitní záznamy, které by mohly vzniknout slučováním několika zdrojů dat. Ovšem za předpokladu, že uživatel by od těchto zdrojů vyžadoval stejná data a výstupní data by byla shodná, respektive duplicitní v pravém slova smyslu.
- **Možnost definovat uživatelské transformace** – rozšíření aplikace o možnost definování vlastních pravidel pro úpravu dat. Například to, že by uživatelé zadali synonymní atributy, které by se v případě slučování JSON souborů sloučili do jednotného názvosloví. Samozřejmě za předpokladu správného použití uživatelem a jeho správném pochopení dat, abychom nepřišli o zásadní data při převodu.
- **Napojení na monitorovací systémy** – přidání funkcí pro sledování výkonu, kapacity úložiště a vytížení aplikace v reálném čase.

Dalším důležitým aspektem budoucího rozšíření je možnost větší personalizace pro různé typy uživatelů. Aplikace by mohla nabídnout různé režimy práce podle zkušeností uživatele, například základní a pokročilý režim, možnost detailnější konfigurace exportních formátů nebo uživatelské přizpůsobení grafického rozhraní. Takové úpravy by umožnily lepší přizpůsobení aplikace různým potřebám uživatelů, od jednotlivců až po organizace s komplexními požadavky na správu big dat. Samozřejmě by se vše odvíjelo od nejvíce vyžadovaných změn ze strany reálných uživatelů, kteří by při samotné praxi přicházeli na případné obtíže při ovládání aplikace pro jejich specifické potřeby.

7.4.2 Propojení s dalšími systémy

Dalším zásadním směrem rozvoje aplikace by bylo její propojení s dalšími systémy zaměřenými na práci s big daty. Jelikož aplikace slouží primárně jako efektivní nástroj pro ukládání velkých datových souborů ve standardizovaných formátech, její další rozšíření by se mělo zaměřit především na integraci se systémy umožňujícími rychlou analýzu, distribuované zpracování a efektivní správu velkých datových objemů.

Mezi klíčové možnosti rozšíření patří:

- **Integrace s distribuovanými úložišti** – přímé napojení na technologie jako Hadoop HDFS, Apache Parquet nebo Google BigQuery, které umožňují efektivní práci s big daty. V aktuálním stavu aplikace využívá kompatibilní JSON formát právě kvůli jeho širokému využití a program se tak nemusí omezovat na specifické platformy. Samozřejmě praxe by ukázala, zda je vhodné určité systémy přímo napojit na možnosti aplikace a zlepšit uživatelský zážitek.
- **Podpora NoSQL databází** – například MongoDB nebo Cassandra, které jsou vhodné pro ukládání a dotazování na rozsáhlá, nestrukturovaná data. Skrze moduly by bylo možné aplikaci přizpůsobit tak, aby bylo možné se napřímo napojit na databázové řešení a nebylo nutné využívat formátu JSON pro převod dat jinými cestami. Tato možnost by zlepšila uživatelské pohodlí. Výběr platforem by pravděpodobně záležel na nejvíce žádaných řešeních. Napojení by mohlo být řešeno skrze integraci v aplikaci, nebo přes dodatečné moduly, které by se mohly vyvíjet nezávisle a do aplikace pouze nahrávat skrze ekosystém.
- **Optimalizace pro cloudová řešení** – umožnění přímého napojení na cloudové platformy jako AWS S3, Google Cloud Storage nebo Microsoft Azure, což by usnadnilo škálovatelnost a dostupnost uložených dat. Na základě dostupných API by mohla být aplikace přizpůsobena pro nejpopulárnější platformy a zajistit tak přímé připojení na úložiště jednotlivců nebo organizací.

Ačkoli hlavní funkcí aplikace je řešení problematiky efektivního ukládání a práce s big daty, lze ji v budoucnu rozšířit i o modul pro lepší předzpracování dat, který by umožňoval provádět základní operace nad vstupními datovými sadami před jejich uložením. Tento modul by mohl zajišťovat automatickou kontrolu duplicit, filtraci či agregaci dat, což by vedlo k dalšímu zefektivnění celého procesu práce s big daty. I zde se samozřejmě předpokládá jistá uživatelská

znalost, kterou by ovšem mohly řešit dříve zmíněné základní nebo rozšířená uživatelská prostředí.

Další zajímavou možností je propojení s analytickým systémem pro pokročilou práci s big daty, který vyvinul kolega Holeček jakožto svoji diplomovou práci. Pokud by byl vytvořen modul umožňující efektivní komunikaci mezi těmito dvěma aplikacemi, mohlo by dojít k propojení, kdy by výše navržená aplikace sloužila jako řešení úložiště a zároveň by poskytovala přímý vstup pro analýzy prováděné systémem, který vytvořil kolega Holeček. Toto propojení by umožnilo rychlou a efektivní analýzu dat přímo z uložených souborů, bez nutnosti zdlouhavých konverzí či manuálního přenosu dat mezi jednotlivými systémy. Jelikož je na vstupu využíván formát CSV, který se dále převádí do formátu Parquet, neměl by být tento modul problematický. Obě aplikace by měly být schopny vzájemně komunikovat, jelikož jedna již umožňuje exportování dat ve formátu CSV a druhá ho již umí načítat.

Automatizace práce s daty by mohla být rozšířena o automatické exporty a importy dat, které by umožnily propojení s dalšími nástroji bez nutnosti manuálního zásahu. Tím by bylo možné například pravidelně synchronizovat databáze s externími zdroji dat, exportovat data ve standardizovaných formátech pro další analýzu nebo přímé napojení na analytické nástroje.

ZÁVĚR

Cílem této diplomové práce byl návrh a implementace aplikace v programovacím jazyce Python, která je schopna efektivně zpracovávat a ukládat rozsáhlé objemy dat, tzv. big data, přičemž byl kladen důraz na využití otevřených technologií a knihoven. Hlavním záměrem práce bylo vytvořit funkční nástroj umožňující zpracování dat z různorodých souborových formátů (např. Excel, CSV, PDF, Word) a jejich následnou transformaci do strukturovaného formátu JSON, případně CSV, který je dále využitelný v moderních datových systémech, včetně NoSQL databází.

V teoretické části práce byly nejprve vymezeny základní pojmy z oblasti big dat, přičemž byly klasifikovány typy dat (mj. strukturovaná, polostrukturovaná a nestrukturovaná) a popsány principy jejich uchovávání. Dále byly podrobně analyzovány nástroje a metody pro práci s big daty. Byly především popsány souborové systémy, datová úložiště a NoSQL databáze. Byla provedena rešerše komerčních řešení (např. Google Bigtable, Amazon DynamoDB) i open-source nástrojů jako Apache Hadoop, Apache Spark, Apache Parquet a Apache Cassandra. Následně byly tyto nástroje a přístupy porovnány na základě kritérií určených literární rešerší.

Metodická část se zaměřila na popis systematické literární rešerše a metodiky návrhu a vývoje aplikace. Popsán byl výběr konkrétních nástrojů a platforem, instalace potřebného softwaru a konfigurace prostředí. Detailně byl popsán návrh struktury aplikace, volba architektury a implementační kroky vedoucí k vytvoření funkčního systému. Součástí této části byla také analýza zdrojů dat, včetně veřejných a neveřejných datasetů, a způsoby jejich využití pro testovací účely. Popsána byla rovněž metodika syntézy dat pomocí knihovny Faker, která umožnila simulaci realistických vstupních dat pro účely testování a validace funkcí aplikace.

V další části práce byly popsány moderní přístupy k předzpracování a ukládání dat. Byla představena podoba výstupní struktury aplikace, podpora různých formátů a celý proces zpracování – od výběru souboru přes jeho načtení, transformaci dat až po export a uložení výsledků. Procesy byly vizualizovány pomocí vývojového diagramu, který umožnil lepší pochopení jednotlivých fází. Následně byla rozebrána samotná implementace aplikace, včetně podrobného popisu použitých knihoven. Ty byly rozděleny na standardní knihovny jazyka Python (např. os, math, datetime) a externí knihovny zaměřené na specifické formáty souborů (pandas, openpyxl, python-docx, PyMuPDF a další).

Pozornost byla věnována rovněž uživatelskému rozhraní aplikace a popisu jeho funkčností. Byly popsány jednotlivé části rozhraní, použité grafické prvky a procesy, které jsou jimi spouštěny. Aplikace umožňuje kontrolu vstupních dat, jejich validaci, extrakci z konkrétních oblastí dokumentů a následnou transformaci. Podrobně byly popsány moduly pro zpracování souborů Excel, Word a PDF, jakož i pomocné nástroje sloužící ke slučování JSON souborů nebo převodu mezi formáty CSV a JSON. Tyto pomocné funkce rozšiřují praktické využití aplikace a umožňují uživatelům komplexní práci s daty bez nutnosti využívat další nástroje.

Závěrečná část práce se věnovala testování, vyhodnocení výsledků a diskusi. Byla představena ukázka výstupu aplikace, která dokumentovala celý proces – od původních vstupních nestrukturovaných dat až po strukturovaný výstup. Byla provedena analýza účinnosti aplikace z hlediska redukce velikosti dat, kdy bylo u testovacích souborů dosaženo zmenšení až o více než 99 %, což potvrzuje vysokou efektivitu zvoleného přístupu. Aplikace tak významně přispívá ke snížení nároků na úložiště a zrychlení následného zpracování.

Dále byla diskutována otázka zasazení aplikace do širšího kontextu životního cyklu big dat, včetně možností její integrace do existujících datových toků. V rámci návrhů budoucího rozšíření byly uvedeny dva hlavní směry vývoje: (1) rozšíření funkcionality na základě zpětné vazby uživatelů (např. o podporu dalších formátů jako XML, Parquet, či Avro), a (2) propojení s externími systémy, zejména NoSQL databázemi (např. MongoDB, Cassandra), případně dalšími projekty umožňujícími analýzu nebo vizualizaci dat.

Na základě dosažených výsledků lze konstatovat, že stanovený cíl diplomové práce byl naplněn. Aplikace představuje praktický nástroj pro předzpracování a ukládání big dat, který je snadno využitelný v různých oblastech – od akademického výzkumu přes podnikové nasazení až po individuální použití. Díky modularitě, otevřenému kódu a použití běžně dostupných knihoven je řešení snadno rozšiřitelné a adaptovatelné podle potřeb konkrétního uživatele. Navržený přístup potvrzuje, že i pomocí open-source nástrojů je možné vyvinout robustní, efektivní a uživatelsky přívětivý systém pro práci s rozsáhlými datovými soubory.

POUŽITÁ LITERATURA

AHMED, Kazi Main Uddin; BOLLEN, Math HJ; ALVAREZ, Manuel. A review of data centers energy consumption and reliability modeling. *IEEE Access*, 2021, 9: 152536-152563.

BAGGA, Simmi; SHARMA, Anil. A comparative study of NoSQL databases. In: *Recent Innovations in Computing: Proceedings of ICRIC 2020*. Springer Singapore, 2021. s. 51-61.

BUNEMAN, Peter. Semistructured data. In: *Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*. 1997. s. 117-121.

CARPENTER, Jeff; HEWITT, Eben. *Cassandra: The Definitive Guide*. Sebastopol: O'Reilly Media, Inc., 2022.

CARROLA, Anthony. *Synthesizing Realistic Substitute Data for a Law Enforcement Database using a Python Library*. West Virginia University, 2022.

CIELEN, Davy; MEYSMAN, Arno D. B.; ALI, Mohamed. *Introducing Data Science: Big data, machine learning, and more, using Python tools*. Shelter Island: Manning, 2016.

COMPUTER SCIENCE ACADEMY. *Python for Data Analysis: A Complete Guide for Beginners, Including Python Statistics and Big Data Analysis*. Milton Keynes: Computer Scince Academy, 2020.

DEAN, Jeffrey; GHEMAWAT, Sanjay. MapReduce: Simplified data processing on large clusters. In: *Proceedings of the 6th Conference on Operating Systems Design & Implementation, Vol. 6 (OSDI'04)*. USENIX Association, 2004, s. 1-13.

FAKER. In: Github.com [online]. 2025. [cit. 2025-02-15]. Dostupné z: https://github.com/joke2k/faker

GANDOMI, Amir; HAIDER, Murtaza. Beyond the hype: Big data concepts, methods, and analytics. *International Journal of Information Management*, 2015, 35.2: 137-144.

GAO, Jianyong, et al. Energy Efficiency and Green Computing in Large-scale Data Centers. In: *2023 International Conference on Evolutionary Algorithms and Soft Computing Techniques (EASCT)*. IEEE, 2023. s. 1-6.

GILDER, George F. *Konec éry Google: úpadek big dat a vzestup ekonomiky blockchainu*. Brno: Zoner Press, 2021.

GUPTA, Adity, et al. NoSQL databases: Critical analysis and comparison. In: *2017 International conference on computing and communication technologies for smart nation (IC3TSN)*. IEEE, 2017. s. 293-299.

HENDL, Jan. *Big data: věda o datech – základy a aplikace*. Praha: Grada Publishing, 2021. Průvodce.

HOLDSWORTH, Jim. What is Hadoop Distributed File System (HDFS)? In: IBM. IBM.com [online]. 2024 [cit. 2025-03-15]. Dostupné z: https://www.ibm.com/think/topics/hdfs

CHANG, Fay, et al. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 2008, 26.2: 1-26.

CHEBOTKO, Artem; KASHLEV, Andrey; LU, Shiyong. A big data modeling methodology for Apache Cassandra. In: *2015 IEEE International Congress on Big Data*. IEEE, 2015. s. 238-245.

IBTISUM, Sifat, et al. A comparative analysis of big data processing paradigms: Mapreduce vs. apache spark. *World Journal of Advanced Research and Reviews*, 2023, 20.1: 1089-1098.

IKEGWU, Anayo Chukwu, et al. Big data analytics for data-driven industry: a review of data sources, tools, challenges, solutions, and research directions. *Cluster Computing*, 2022, 25.5: 3343-3387.

JETBRAINS. PyCharm – Quick start guide. [online]. 2025. [cit. 2025-03-07]. Dostupné z: https://www.jetbrains.com/help/pycharm/quick-start-guide.html

JIN, Xiaolong, et al. Significance and challenges of big data research. *Big Data Research*, 2015, 2.2: 59-64.

JOHNSON, Jeff S.; FRIEND, Scott B.; LEE, Hannah S. Big data facilitation, utilization, and monetization: Exploring the 3Vs in a new product development process. *Journal of Product Innovation Management*, 2017, 34.5: 640-658.

KALID, Sultana, et al. Big-data NoSQL databases: A comparison and analysis of "Big-Table","DynamoDB", and "Cassandra". In: *2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA)*. IEEE, 2017. s. 89-93.

KHAN, Nawsher, et al. Big data: survey, technologies, opportunities, and challenges. *The Scientific World Journal*, 2014, 2014.1: 712826.

KITCHENHAM, Barbara, et al. Systematic literature reviews in software engineering–a systematic literature review. *Information and software technology*, 2009, 51.1: 7-15.

KWAK, Sang Kyu; KIM, Jong Hae. Statistical data preparation: management of missing values and outliers. *Korean Journal of Anesthesiology*, 2017, 70.4: 407.

LNENICKA, Martin; KOMARKOVA, Jitka. Big and open linked data analytics ecosystem: Theoretical background and essential elements. *Government Information Quarterly*, 2019, 36.1: 129-144.

MAZZARINO, Simona; MINIERI, Andrea; GILLI, Luca. Nerpii: A python library to perform named entity recognition and generate personal identifiable information. In: *Proceedings of the Seventh Workshop on Natural Language for Artificial Intelligence (NL4AI 2023) co-located with 22th International Conference of the Italian Association for Artificial Intelligence (AI* IA 2023)*. CEUR Workshop Proceedings, 2023. s. 1-7.

MCKINNEY, Wes. *Python for data analysis: data wrangling with Pandas, NumPy, and IPython*. Second edition. Sebastopol: O'Reilly, 2017.

MEHMOOD, Hassan, et al. Implementing big data lake for heterogeneous data sources. In: *2019 IEEE 35th international conference on data engineering workshops (icdew)*. IEEE, 2019. s. 37-44.

NAEEM, Muhammad, et al. Trends and future perspective challenges in big data. In: *Advances in intelligent data analysis and applications: Proceeding of the sixth euro-China conference on intelligent data analysis and applications, 15–18 October 2019, Arad, Romania*. Springer Singapore, 2022. s. 309-325.

OPENPYXL. *Openpyxl - A Python library to read/write Excel 2010 xlsx/xlsm files*. [online]. 2024. [cit. 2025-03-22]. Dostupné z: https://openpyxl.readthedocs.io/en/stable/index.html

PALLAMALA, Rajesh Kumar; RODRIGUES, Paul. An investigative testing of structured and unstructured data formats in big data application using apache spark. *Wireless Personal Communications*, 2022, 122.1: 603-620.

PANDAS. *Pandas documentation*. [online]. 2024. [cit. 2025-03-22]. Dostupné z: https://pandas.pydata.org/docs/index.html

PECINOVSKÝ, Rudolf. *Python-knihovny pro práci s daty: pro verzi 3.11*. Praha: Grada Publishing a.s., 2023.

PECINOVSKÝ, Rudolf. *Python: kompletní příručka jazyka pro verzi 3.10*. Praha: Grada Publishing a.s., 2021.

PECINOVSKÝ, Rudolf. *Začínáme programovat v jazyku Python*. Praha: Grada Publishing a.s., 2022.

PEZOA, Felipe, et al. Foundations of JSON schema. In: *Proceedings of the 25th international conference on World Wide Web*. 2016. s. 263-273.

PYMUPDF. *PyMuPDF documentation*. [online]. 2025. [cit. 2025-03-22]. Dostupné z: https://pymupdf.readthedocs.io/en/latest/index.html

PYTHON. *Csv – CSV File Reading and Writing*. [online]. 2025a. [cit. 2025-03-18]. Dostupné z: https://docs.python.org/3/library/csv.html

PYTHON. *Datetime – Basic date and time types*. [online]. 2025b. [cit. 2025-03-18]. Dostupné z: https://docs.python.org/3/library/datetime.html

PYTHON. *Json – JSON encoder and decoder*. [online]. 2025c. [cit. 2025-03-07]. Dostupné z: https://docs.python.org/3/library/json.html

PYTHON. *Math – Mathematical functions*. [online]. 2025d. [cit. 2025-03-07]. Dostupné z: https://docs.python.org/3/library/math.html

PYTHON. *Os – Miscellaneous operating system interfaces.* [online]. 2025e. [cit. 2025-03-07]. Dostupné z: https://docs.python.org/3/library/os.html

PYTHON. *Re – Regular expression operations*. [online]. 2025f. [cit. 2025-03-18]. Dostupné z: https://docs.python.org/3/library/re.html

PYTHON. *Time – Time access and conversions*. [online]. 2025g. [cit. 2025-03-07]. Dostupné z: https://docs.python.org/3/library/time.html

PYTHON. *Tkinter – Python interface to Tcl/Tk.* [online]. 2025h. [cit. 2025-03-07]. Dostupné z: https://docs.python.org/3/library/tkinter.html

PYTHON-DOCX. *Python-docx documentation*. [online]. 2024. [cit. 2025-03-22]. Dostupné z: https://python-docx.readthedocs.io/en/latest/

RAHUL, Kumar; BANYAL, Rohitash Kumar. Data life cycle management in big data analytics. *Procedia Computer Science*, 2020, 173: 364-371.

REMALA, Rajesh, et al. Optimizing Data Management Strategies: Analyzing Snowflake and DynamoDB for SQL and NoSQL. *International Journal of Management, IT & Engineering*, 2024, 14.8: 46-52.

SAEED, Nagham; HUSAMALDIN, Laden. Big data characteristics (V's) in industry. *Iraqi Journal of Industrial Research*, 2021, 8.1: 1-9.

SALLOUM, Salman, et al. Big data analytics on Apache Spark. *International Journal of Data Science and Analytics*, 2016, 1.3: 145-164.

SINGH, Sukhpreet, et al. Comparative analysis of Apache Hadoop and Apache Spark for business intelligence. *AIP Conference Proceedings*, 2025, 2025, 3224.1: 020040.

SIVASUBRAMANIAN, Swaminathan. Amazon dynamoDB: a seamlessly scalable non-relational database service. In: *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. 2012. s. 729-730.

STEWART, John M.; MOMMERT, Michael. *Python for scientists*. Cambridge: Cambridge University Press, 2023.

STROHBACH, Martin, et al. Big data storage. In: *New Horizons for a Data-Driven Economy: A Roadmap for Usage and Exploitation of Big Data in Europe*. Cham: Springer International Publishing, 2016. s. 119-141.

SUN, Xudong, et al. Survey of distributed computing frameworks for supporting big data analysis. *Big Data Mining and Analytics*, 2023, 6.2: 154-169.

THAKARE, Atul, et al. NoSQL databases: modern data systems for big data analytics-features, categorization and comparison. *International Journal of Electrical and Computer Engineering Systems*, 2023, 14.2: 207-216.

THANH, Tran Doan, et al. A taxonomy and survey on distributed file systems. In: *2008 Fourth international conference on networked computing and advanced information management*. IEEE, 2008. s. 144-149.

UPADHYAY, Shubham, et al. Analytics and Storage of Big Data. In: *Proceedings of the International Semantic Intelligence Conference 2021 (ISIC 2021)*. 2021. s. 202-210.

VOHRA, Deepak. Apache parquet. In: *Practical Hadoop Ecosystem: A Definitive Guide to Hadoop-Related Frameworks and Tools*. Apress, Berkeley, 2016. s. 325-335.

WEINTRAUB, Grisha. Dynamo and BigTable—Review and comparison. In: *2014 IEEE 28th Convention of Electrical & Electronics Engineers in Israel (IEEEI)*. IEEE, 2014. s. 1-5.

WIESE, Lena. *Advanced data management: for SQL, NoSQL, cloud and distributed databases*. Walter de Gruyter GmbH & Co KG, 2015.

WU, Jinsong, et al. Big data meet green challenges: Greening big data. *IEEE Systems Journal*, 2016, 10.3: 873-887.

SEZNAM PŘÍLOH

Příloha I: Zdrojový kód – Hlavní aplikace

```
import math
import tkinter as tk
from tkinter import filedialog, ttk, messagebox
import pandas as pd
import os
import json
import fitz  # PyMuPDF
from docx import Document
from datetime import datetime
import re
from openpyxl import load_workbook
import time
import csv


# Inicializace hlavního okna
root = tk.Tk()
root.title("Big Data Processing App")
root.geometry("500x300")

# Globální proměnné
files_set = set()
processed_data = {}
excel_extraction_settings = []
word_extraction_settings = []

# Globální nastavení
settings = {
    'delimiter': ';',
    'decimal': ','
}

# Funkce pro výběr složky a souborů v ní
def select_folder():
    folder_path = filedialog.askdirectory()
    if folder_path:
        for root_dir, _, files in os.walk(folder_path):
            for file in files:
                file_path = os.path.join(root_dir, file).replace("\\", "/")
                files_set.add(file_path)

# Funkce pro výběr souborů
def select_files():
    file_paths = filedialog.askopenfilenames()
    if file_paths:
        files_set.update(file_paths)

# Zobrazení informací o souborech
def show_selected_files_info():
    if not files_set:
        messagebox.showinfo("Info", "Žádné soubory nebyly vybrány.")
        return

    total_size = convert_size(sum(os.path.getsize(file) for file in
files_set))
    message = (
        f"Počet vybraných souborů: {len(files_set)}\n"
        f"Celková velikost: {total_size}\n\n"
```

```
            "Vybrané soubory:\n" + "\n".join(files_set)
        )

        info_window = tk.Toplevel()
        info_window.title("Informace o souborech")
        info_window.geometry("600x400")

        text_box = tk.Text(info_window, wrap='word')
        scroll_bar = tk.Scrollbar(info_window, command=text_box.yview)
        text_box.config(yscrollcommand=scroll_bar.set)

        text_box.insert('1.0', message)
        text_box.config(state='disabled')

        text_box.pack(side='left', fill='both', expand=True)
        scroll_bar.pack(side='right', fill='y')

    # Převod velikosti souboru
    def convert_size(size_bytes):
        if size_bytes == 0:
            return "0B"
        size_name = ("B", "kB", "MB", "GB", "TB")
        i = int(math.floor(math.log(size_bytes, 1024)))
        p = math.pow(1024, i)
        s = round(size_bytes / p, 2)
        return f"{s} {size_name[i]}"

    # Funkce pro převod času
    def format_elapsed_time(elapsed_time):
        if elapsed_time < 60:
            return f"{int(elapsed_time)} sekund"
        elif elapsed_time < 3600:
            minutes = int(elapsed_time // 60)
            seconds = int(elapsed_time % 60)
            return f"{minutes} minut a {seconds} sekund"
        else:
            hours = int(elapsed_time // 3600)
            minutes = int((elapsed_time % 3600) // 60)
            return f"{hours} hodin a {minutes} minut"

    # Načtení a export JSON
    def load_json(file_path):
        with open(file_path, 'r', encoding='utf-8') as file:
            return json.load(file)

    def export_to_json(data):
        json_file_path = filedialog.asksaveasfilename(defaultextension=".json",
    filetypes=[("JSON files", "*.json")])
        if not json_file_path:
            messagebox.showinfo("Info", "Export byl zrušen.")
            return

        with open(json_file_path, 'w', encoding='utf-8') as json_file:
            json.dump(data, json_file, ensure_ascii=False, indent=4)

        messagebox.showinfo("Success", "Data byla exportována do JSON
    souboru.")

    # Funkce pro nastavení
    def open_settings():
        settings_window = tk.Toplevel(root)
```

```
    settings_window.title("Nastavení oddělovačů")
    settings_window.geometry("300x200")

    delimiter_label = tk.Label(settings_window, text="CSV Delimiter:")
    delimiter_label.pack(pady=5)

    delimiter_entry = tk.Entry(settings_window)
    delimiter_entry.insert(0, settings['delimiter'])
    delimiter_entry.pack(pady=5)

    decimal_label = tk.Label(settings_window, text="CSV Decimal:")
    decimal_label.pack(pady=5)

    decimal_entry = tk.Entry(settings_window)
    decimal_entry.insert(0, settings['decimal'])
    decimal_entry.pack(pady=5)

    def save_settings():
        settings['delimiter'] = delimiter_entry.get() or ';'
        settings['decimal'] = decimal_entry.get() or ','
        settings_window.destroy()

    save_button = tk.Button(settings_window, text="Uložit",
command=save_settings)
    save_button.pack(pady=10)

# Funkce pro export seznamu extrahovaných buněk
def export_extraction_settings(extraction_settings):
    file_path = filedialog.asksaveasfilename(defaultextension=".json",
filetypes=[("JSON files", "*.json")])
    if not file_path:
        return
    with open(file_path, 'w', encoding='utf-8') as file:
        json.dump(extraction_settings, file, ensure_ascii=False, indent=4)
    messagebox.showinfo("Export", "Nastavení extrakce bylo úspěšně
exportováno.")

# Funkce pro import seznamu extrahovaných buněk
def import_extraction_settings(extraction_list, extraction_settings):
    file_path = filedialog.askopenfilename(filetypes=[("JSON files",
"*.json")])
    if not file_path:
        return
    with open(file_path, 'r', encoding='utf-8') as file:
        imported_settings = json.load(file)
    extraction_settings.clear()
    extraction_settings.extend(imported_settings)
    extraction_list.delete(0, tk.END)
    for setting in extraction_settings:
        if extraction_settings == excel_extraction_settings:
            extraction_list.insert(tk.END, f"{setting['cell']}:
{setting['attribute']}")
        else:
            extraction_list.insert(tk.END, f"{setting['keyword']}:
{setting['attribute']}")
    messagebox.showinfo("Import", "Nastevní extrakce bylo úspěšně
importováno.")

# Funkce pro nastavení extrakce z Excelu
def open_excel_extraction_settings():
    def save_excel_extraction():
```

```
        cell = cell_entry.get()
        attribute = attribute_entry.get()
        if not cell or not attribute:
            messagebox.showwarning("Warning", "Musíte zadat buňku i název
atributu.")
            return

        if not re.match(r'^[A-Za-z]+\d+$', cell):
            messagebox.showerror("Error", "Buňka musí být ve správném
formátu, například A1 nebo B2.")
            return

        excel_extraction_settings.append({'cell': cell, 'attribute':
attribute})
        extraction_list.insert(tk.END, f"{cell}: {attribute}")

        cell_entry.delete(0, tk.END)
        attribute_entry.delete(0, tk.END)

    def load_existing_settings():
        extraction_list.delete(0, tk.END)
        for setting in excel_extraction_settings:
            extraction_list.insert(tk.END, f"{setting['cell']}:
{setting['attribute']}")

    extraction_window = tk.Toplevel(root)
    extraction_window.title("Nastavení extrakce Excelu")
    extraction_window.geometry("400x450")

    cell_label = tk.Label(extraction_window, text="Buňka:")
    cell_label.pack(pady=5)

    cell_entry = tk.Entry(extraction_window)
    cell_entry.pack(pady=5)

    attribute_label = tk.Label(extraction_window, text="Název atributu:")
    attribute_label.pack(pady=5)

    attribute_entry = tk.Entry(extraction_window)
    attribute_entry.pack(pady=5)

    save_button = tk.Button(extraction_window, text="Uložit",
command=save_excel_extraction)
    save_button.pack(pady=10)

    extraction_list = tk.Listbox(extraction_window)
    extraction_list.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

    export_button = tk.Button(extraction_window, text="Exportovat seznam",
command=lambda: export_extraction_settings(excel_extraction_settings))
    export_button.pack(pady=5)

    import_button = tk.Button(extraction_window, text="Importovat seznam",
command=lambda: import_extraction_settings(extraction_list,
excel_extraction_settings))
    import_button.pack(pady=5)

    load_existing_settings()

def open_word_extraction_settings():
    def save_word_extraction():
```

```
        keyword = keyword_entry.get()
        attribute = attribute_entry.get()
        if not keyword or not attribute:
            messagebox.showwarning("Warning", "Musíte zadat klíčové slovo i
název atributu.")
            return

        word_extraction_settings.append({'keyword': keyword, 'attribute':
attribute})
        extraction_list.insert(tk.END, f"{keyword}: {attribute}")

        keyword_entry.delete(0, tk.END)
        attribute_entry.delete(0, tk.END)

    def load_existing_settings():
        extraction_list.delete(0, tk.END)
        for setting in word_extraction_settings:
            extraction_list.insert(tk.END, f"{setting['keyword']}:
{setting['attribute']}")

    extraction_window = tk.Toplevel(root)
    extraction_window.title("Nastavení extrakce Word a PDF")
    extraction_window.geometry("400x450")

    keyword_label = tk.Label(extraction_window, text="Klíčové slovo:")
    keyword_label.pack(pady=5)

    keyword_entry = tk.Entry(extraction_window)
    keyword_entry.pack(pady=5)

    attribute_label = tk.Label(extraction_window, text="Název atributu:")
    attribute_label.pack(pady=5)

    attribute_entry = tk.Entry(extraction_window)
    attribute_entry.pack(pady=5)

    save_button = tk.Button(extraction_window, text="Uložit",
command=save_word_extraction)
    save_button.pack(pady=10)

    extraction_list = tk.Listbox(extraction_window)
    extraction_list.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

    export_button = tk.Button(extraction_window, text="Exportovat seznam",
command=lambda: export_extraction_settings(word_extraction_settings))
    export_button.pack(pady=5)

    import_button = tk.Button(extraction_window, text="Importovat seznam",
command=lambda: import_extraction_settings(extraction_list,
word_extraction_settings))
    import_button.pack(pady=5)

    load_existing_settings()

# Nástroj pro zpracování strukturovaných CSV a Excel souborů
def open_structured_data_tool():

        structured_file_path = filedialog.askopenfilename(filetypes=[("CSV
or Excel files", "*.csv *.xls *.xlsx")])
        if not structured_file_path:
            return
```

```
        if structured_file_path.endswith('.csv'):
            df = pd.read_csv(structured_file_path,
delimiter=settings['delimiter'], decimal=settings['decimal'])
        else:
            df = pd.read_excel(structured_file_path)

        json_data = df.to_dict(orient='records')
        export_to_json(json_data)

# Zpracování Excel souborů
def process_excel(file_path):
    wb = load_workbook(file_path, data_only=True)
    sheet = wb.active

    if not excel_extraction_settings:
        messagebox.showerror("Error", "Nebyly zadány žádné buňky pro
extrakci.")
        return {}

    extracted_data = {}
    for setting in excel_extraction_settings:
        cell = setting['cell']
        attribute = setting['attribute']
        try:
            value = sheet[cell].value
            extracted_data[attribute] = value
        except Exception:
            extracted_data[attribute] = None

    return extracted_data

# Zpracování PDF a DOC souborů
def process_doc(file_path):
    with fitz.open(file_path) as doc:
        text = "".join(page.get_text() for page in doc)

    return extract_data_by_keywords(text)

# Zpracování souborů
def process_files(file_type):
    global processed_data
    if not files_set:
        messagebox.showerror("Error", "Nejprve vyberte soubory.")
        return

    start_time = time.time()
    progress_bar['value'] = 0
    progress_bar['maximum'] = len(files_set)

    processed_data = {}
    skipped_files = []
    for idx, file_path in enumerate(files_set):
        ext = os.path.splitext(file_path)[1].lower()
        try:
            # Stejné funkce, jinak pracují s daty. Vyžadován proto vstup
pro rozlišení
            if ext in ['.xls', '.xlsx'] and file_type == "excel":
                processed_data[file_path] = process_excel(file_path)
            elif ext == '.pdf' and file_type == "wrd/pdf":
                processed_data[file_path] = process_doc(file_path)
```

```
            elif ext == '.docx' and file_type == "wrd/pdf":
                processed_data[file_path] = process_doc(file_path)
            else:
                skipped_files.append(file_path)
        except Exception as e:
            skipped_files.append(f"{file_path} (Error: {str(e)})")
        finally:
            progress_bar['value'] = idx + 1
            root.update_idletasks()

    end_time = time.time()
    elapsed_time = end_time - start_time

    elapsed_time_str = format_elapsed_time(elapsed_time)

    if skipped_files:
        skipped_message = "\n".join(skipped_files)
        log_unprocessed_files(skipped_files)
        messagebox.showwarning("Warning", f"Některé soubory nebyly
zpracovány:\n{skipped_message}")

    files_set.clear()
    messagebox.showinfo("Success", f"Zpracování souborů
dokončeno.\nZpracování trvalo: {elapsed_time_str}")


def merge_json_files():
    # Výběr JSON souborů
    file_paths = filedialog.askopenfilenames(filetypes=[("JSON Files",
"*.json")])
    if not file_paths:
        messagebox.showinfo("Info", "Nebyl vybrán žádný soubor.")
        return

    merged_data = {}

    for file_path in file_paths:
        try:
            with open(file_path, "r", encoding="utf-8") as file:
                data = json.load(file)

                if isinstance(data, dict):
                    merged_data.update(data)  # Přímé sloučení bez vnoření
                else:
                    messagebox.showwarning("Upozornění", f"Soubor
{file_path} obsahuje nepodporovanou strukturu.")
        except Exception as e:
            messagebox.showerror("Chyba", f"Chyba při čtení souboru
{file_path}: {e}")

    # Výběr cesty pro uložení výsledného JSON
    save_path = filedialog.asksaveasfilename(defaultextension=".json",
filetypes=[("JSON Files", "*.json")])
    if save_path:
        with open(save_path, "w", encoding="utf-8") as output_file:
            json.dump(merged_data, output_file, ensure_ascii=False,
indent=4)
        messagebox.showinfo("Úspěch", f"JSON byl úspěšně sloučen a uložen
do {save_path}")

# Logování nezpracovaných souborů
```

```
def log_unprocessed_files(skipped_files):
    timestamp = datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
    log_filename = f"Unprocessed_files_{timestamp}.txt"
    with open(log_filename, 'w', encoding='utf-8') as log_file:
        log_file.write("\n".join(skipped_files))

# Funkce pro informace o aplikaci
def about_app():
    messagebox.showinfo("O aplikaci", "Big Data Processing App\nVerze:
1.0\nAutor: Tomáš Fráňa\nst58229@upce.cz")

def extract_text_from_pdf(pdf_path):
    text_data = []
    try:
        doc = fitz.open(pdf_path)
        for page in doc:
            text_data.append(page.get_text())
        return "\n".join(text_data)
    except Exception as e:
        print(f"Chyba při zpracování PDF: {e}")
        return None

def extract_text_from_docx(docx_path):
    try:
        doc = Document(docx_path)
        return "\n".join([para.text for para in doc.paragraphs if
para.text.strip()])
    except Exception as e:
        print(f"Chyba při zpracování Word souboru: {e}")
        return None

def extract_data_by_keywords(text):
    extracted_data = {}
    for setting in word_extraction_settings:
        pattern = rf"{setting['keyword']}\s*[:\-]?\s*(.+)"
        attribute = setting['attribute']
        match = re.search(pattern, text, re.IGNORECASE)
        if match:
            extracted_data[attribute] = match.group(1).strip()
    return extracted_data

def json_to_csv():
    json_file = filedialog.askopenfilename(filetypes=[("JSON Files",
"*.json")])
    if not json_file:
        messagebox.showinfo("Info", "Nebyl vybrán žádný soubor.")
        return

    csv_file = filedialog.asksaveasfilename(defaultextension=".csv",
filetypes=[("CSV Files", "*.csv")])
    if not csv_file:
        return

    try:
        # Načtení JSON souboru
        with open(json_file, "r", encoding="utf-8") as file:
            data = json.load(file)

        if not isinstance(data, dict):  # Ověříme, že JSON obsahuje slovník
(ne seznam)
            messagebox.showerror("Chyba", "JSON soubor musí být slovník,
```

```
kde klíče jsou názvy souborů.")
            return

        structured_data = []
        header = set(["file_path"])  # Hlavička obsahuje i file_path

        # Převedeme JSON strukturu na list of dicts
        for file_path, attributes in data.items():
            if isinstance(attributes, dict):  # Ověříme, že hodnoty jsou
slovníky
                header.update(attributes.keys())  # Přidáme klíče do
hlavičky
                record = {"file_path": file_path, **attributes}  # Přidáme
cestu k souboru
                structured_data.append(record)

        # Seřadíme klíče pro konzistentní výstup
        header = sorted(header)

        # Zápis do CSV
        with open(csv_file, "w", encoding="utf-8-sig", newline="") as file:
            writer = csv.DictWriter(file, fieldnames=header)
            writer.writeheader()
            writer.writerows(structured_data)

        messagebox.showinfo("Úspěch", f"JSON byl úspěšně převeden a uložen
jako CSV: {csv_file}")

    except Exception as e:
        messagebox.showerror("Chyba", f"Chyba při převodu JSON na CSV:
{e}")

# GUI prvky
menu_bar = tk.Menu(root)
root.config(menu=menu_bar)

csv_menu = tk.Menu(menu_bar, tearoff=0)
menu_bar.add_cascade(label="CSV -> JSON", menu=csv_menu)
csv_menu.add_command(label="Převést CSV",
command=open_structured_data_tool)
csv_menu.add_command(label="Nastavení oddělovačů", command=open_settings)

menu_bar.add_command(label="Sloučit JSON", command=merge_json_files)
menu_bar.add_command(label="JSON -> CSV", command=json_to_csv)
menu_bar.add_command(label="O aplikaci", command=about_app)

frame_select = tk.Frame(root)
frame_select.pack(pady=5)

file_button = tk.Button(frame_select, text="Vybrat soubory",
command=select_files)
folder_button = tk.Button(frame_select, text="Vybrat složku",
command=select_folder)
info_button = tk.Button(frame_select, text="i",
command=show_selected_files_info, width=3)

file_button.pack(side="left", padx=2)
folder_button.pack(side="left", padx=2)
info_button.pack(side="left", padx=2)

frame_main = tk.Frame(root)
```

```
frame_main.pack(pady=10, padx=10, fill="both", expand=True)

frame_left = tk.Frame(frame_main)
frame_left.pack(side="left", padx=10, fill="both", expand=True)

frame_right = tk.Frame(frame_main)
frame_right.pack(side="right", padx=10, fill="both", expand=True)

# Tlačítka pro zpracování Excel
btn_settings_excel = tk.Button(frame_left, text="Nastavení Excel extrakce",
command=open_excel_extraction_settings)
btn_settings_excel.pack(pady=5, fill="x")

btn_process_excel = tk.Button(frame_left, text="Zpracovat soubory (Excel)",
command=lambda: process_files("excel"))
btn_process_excel.pack(pady=5, fill="x")

# Tlačítka pro zpracování Word/PDF
btn_settings_word = tk.Button(frame_right, text="Nastavení Word/PDF
extrakce", command=open_word_extraction_settings)
btn_settings_word.pack(pady=5, fill="x")

btn_process_word = tk.Button(frame_right, text="Zpracovat soubory
(Word/PDF)", command=lambda: process_files("wrd/pdf"))
btn_process_word.pack(pady=5, fill="x")

# Tlačítko pro export JSON
btn_export_json = tk.Button(root, text="Exportovat do JSON",
command=lambda: export_to_json(processed_data))
btn_export_json.pack(pady=20, padx=10, fill="x")

# Progress bar
progress_bar = ttk.Progressbar(root, orient="horizontal", length=100,
mode="determinate")
progress_bar.pack(pady=20, padx=10, fill="x")


# Spuštění aplikace
root.mainloop()
```

Příloha II: Zdrojový kód – Faker skript

```
import os
import random
from faker import Faker
from openpyxl import load_workbook
from fpdf import FPDF
from docx import Document
from datetime import datetime


def generate_invoices(num_invoices, invoice_type,
output_folder="generated_invoices"):
    # Inicializace Faker pro česká data
    fake = Faker("cs_CZ")
    os.makedirs(output_folder, exist_ok=True)

    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")  # Unikátní čas
pro sérii faktur

    # Šablony pro generování faktur
    excel_template = "Sablona_Excel_Faktura3.xlsx"
    word_template = "Sablona_Word_Faktura1.docx"

    # Oprava kvůli generování PDF
    font_path = "C:\\Windows\\Fonts\\arial.ttf"  # Cesta k TTF souboru na
Windows
    if not os.path.exists(font_path):
        font_path = "/usr/share/fonts/truetype/dejavu/DejaVuSans.ttf"  #
Alternativa pro Linux/Mac

    # Samotné generování faktur
    for i in range(num_invoices):
        invoice_number = random.randint(100000, 999999)
        date_issued = datetime.today().strftime("%d.%m.%Y")
        due_date = (datetime.today().replace(day=28)).strftime("%d.%m.%Y")
        company_name = fake.company()
        address = fake.address().replace("\n", ", ")
        total_amount = round(random.uniform(500, 50000), 2)
        ico = fake.ssn()
        bank_list = [
            "Česká spořitelna", "Komerční banka", "ČSOB", "MONETA Money
Bank",
            "Fio banka", "Air Bank", "UniCredit Bank", "Raiffeisenbank"
        ]
        bank_name = random.choice(bank_list)
        bank_account = f"{random.randint(1000000000,
9999999999)}/{random.randint(1000, 9999)}"  # Číslo bankovního účtu ve
formátu XXXXXXXX/YYYY
        variable_symbol = random.randint(1000000000, 9999999999)
        city_name = fake.city()
        phone_number = fake.phone_number()
        months_cs = {
            "January": "Leden", "February": "Únor", "March": "Březen",
            "April": "Duben", "May": "Květen", "June": "Červen",
            "July": "Červenec", "August": "Srpen", "September": "Září",
            "October": "Říjen", "November": "Listopad", "December":
"Prosinec"
        }
        month_name = months_cs[datetime.now().strftime("%B")]
```

```
        if invoice_type == "excel":
            # Úprava Excel šablony
            wb = load_workbook(excel_template)
            sheet = wb.active

            # configurovatelné
            sheet["C1"] = invoice_number

            # Dodavatel
            sheet["E4"] = company_name
            sheet["E5"] = address
            sheet["E6"] = ico
            sheet["E7"] = f"CZ{ico}"
            sheet["E9"] = bank_name
            sheet["E10"] = bank_account
            sheet["E11"] = ""
            sheet["E13"] = phone_number

            sheet["B15"] = variable_symbol
            sheet["B16"] = due_date
            sheet["B17"] = total_amount

            sheet["F16"] = month_name

            sheet["B21"] = city_name
            sheet["B22"] = date_issued

            # Uložení Excel šablony
            excel_file = os.path.join(output_folder,
f"invoice_{timestamp}_{i + 1}.xlsx")
            wb.save(excel_file)
            print(f"Generovaná faktura uložena jako: {excel_file}")

        elif invoice_type == "word":
            # Úprava Word šablony
            doc = Document(word_template)
            for paragraph in doc.paragraphs:
                for run in paragraph.runs:  # Zachová formátování a obrázky
                    run.text = run.text.replace("{INVOICE_NUMBER}",
str(invoice_number))
                    run.text = run.text.replace("{DATE_ISSUED}",
date_issued)
                    run.text = run.text.replace("{DUE_DATE}", due_date)
                    run.text = run.text.replace("{COMPANY_NAME}",
company_name)
                    run.text = run.text.replace("{ADDRESS}", address)
                    run.text = run.text.replace("{TOTAL_AMOUNT}",
f"{total_amount} Kč")
                    run.text = run.text.replace("{MONTH_NAME}", month_name)
                    run.text = run.text.replace("{ICO}", ico)
                    run.text = run.text.replace("{DIC}", f"CZ{ico}")
                    run.text = run.text.replace("{PHONE_NUMBER}",
phone_number)
                    run.text = run.text.replace("{BANK_NAME}", bank_name)
                    run.text = run.text.replace("{BANK_ACCOUNT}",
bank_account)
                    run.text = run.text.replace("{VS}",
f"{variable_symbol}")
                    run.text = run.text.replace("{CITY_NAME}", city_name)
```

```
            word_file = os.path.join(output_folder,
f"invoice_{timestamp}_{i + 1}.docx")
            doc.save(word_file)
            print(f"Generovaná faktura uložena jako: {word_file}")

        elif invoice_type == "pdf":
            # Úprava PDF šablony s UTF-8 podporou
            pdf = FPDF()
            pdf.add_page()
            pdf.add_font('CustomFont', '', font_path, uni=True)
            pdf.set_font("CustomFont", '', 16)
            pdf.cell(200, 10, txt="Faktura", ln=True, align='C')
            pdf.set_font("CustomFont", '', 12)
            pdf.cell(200, 10, txt=f"Faktura č. {invoice_number}", ln=True)
            pdf.cell(200, 10, txt=f"Datum vystavení: {date_issued}",
ln=True)
            pdf.cell(200, 10, txt=f"Datum splatnosti: {due_date}", ln=True)
            pdf.cell(200, 10, txt=f"Dodavatel: {company_name}", ln=True)
            pdf.cell(200, 10, txt=f"IČO: {ico}", ln=True)
            pdf.cell(200, 10, txt=f"Adresa: {address}", ln=True)
            pdf.cell(200, 10, txt=f"Bankovní účet: {bank_account}
({bank_name})", ln=True)
            pdf.cell(200, 10, txt=f"Variabilní symbol: {variable_symbol}",
ln=True)
            pdf.cell(200, 10, txt=f"Telefon: {phone_number}", ln=True)
            pdf.cell(200, 10, txt=f"Celková částka: {total_amount} Kč",
ln=True)
            pdf_file = os.path.join(output_folder, f"invoice_{timestamp}_{i
+ 1}.pdf")
            pdf.output(pdf_file)
            print(f"Generovaná faktura uložena jako: {pdf_file}")

        else:
            print("Neplatný typ faktury.")
            break


if __name__ == "__main__":
    invoice_type = input("Zadejte typ faktury (excel/word/pdf):
").strip().lower()
    num_invoices = int(input("Kolik faktur chcete vygenerovat? "))

    generate_invoices(num_invoices, invoice_type)
```