Vysoká škola ekonomická v Praze
Fakulta informatiky a statistiky


# Patternizace logů Vysoké školy ekonomické

DIPLOMOVÁ PRÁCE

Autor: Bc. Barbora Šandová
Vedoucí práce: Ing. Karel Maršálek
Konzultant práce: Ing. Karel Šimeček, Ph.D.

Praha, květen 2025

## Poděkování

Velké dík patří především vedoucímu této práce Ing. Karlu Maršálkovi a také dr. Karlu Šimečkovi za cenné rady, vedení a trpělivost. Nakonec bych ráda poděkovala všem blízkým, kteří mě při psaní podporovali a i přes prodlouženou dobu psaní a přerušené studium nade mnou nezlomili hůl.

# Abstrakt

Tato diplomová práce se zabývá návrhem a implementací systému pro automatizované zpracování a analýzu logů v prostředí Vysoké školy ekonomické v Praze. Práce reaguje na problém nedostatečného monitoringu kybernetické bezpečnosti, kdy z více než 2 200 zdrojů logů bylo dosud analyzováno pouze přibližně 30. Implementované řešení využívá algoritmus Extended Nagappan-Vouk (ENV) pro efektivní identifikaci vzorů v logových datech s dosaženým F1 skóre 96,0 %. Pro optimalizaci výkonu byly implementovány techniky paralelního zpracování a sloupcově orientované úložiště Apache Parquet, které redukovalo velikost souborů o 70-85 % a zrychlilo načítání dat až o 90 %. Systém zkrátil dobu zpracování měsíčního objemu logů z původních 12 hodin na pouhých 15 minut, což představuje 48násobné zrychlení. Součástí řešení je interaktivní dashboard vyvinutý v knihovně Streamlit, který poskytuje uživatelsky přívětivé rozhraní pro vizualizaci a interpretaci výsledků. Navržený systém významně posiluje kybernetickou bezpečnost VŠE zlepšením schopnosti detekce potenciálních bezpečnostních incidentů a umožňuje efektivní filtrování a analýzu logů. Práce také diskutuje možnosti dalšího vývoje, včetně implementace zpracování v reálném čase a využití technik strojového učení pro automatickou detekci anomálií.

## Klíčová slova

Analýza logů, kybernetická bezpečnost, Extended Nagappan-Vouk algoritmus, paralelní zpracování dat, vizualizace bezpečnostních dat

# Abstract

This diploma thesis focuses on the design and implementation of a system for automated processing and analysis of logs at the University of Economics in Prague. The work addresses the problem of insufficient cybersecurity monitoring, where only approximately 30 out of more than 2,200 log sources have been analyzed so far. The implemented solution uses the Extended Nagappan-Vouk (ENV) algorithm for effective identification of patterns in log data, achieving an F1 score of 96.0 %. For performance optimization, parallel processing techniques and Apache Parquet columnar storage were implemented, reducing file sizes by 70-85 % and accelerating data loading by up to 90 %. The system shortened the processing time of monthly logs from the original 12 hours to just 15 minutes, representing a 48-fold speed improvement. An interactive dashboard developed using the Streamlit library forms part of the solution, providing a user-friendly interface for visualization and interpretation of results. The designed system significantly strengthens the university's cybersecurity by improving the detection of potential security incidents and enables efficient filtering and analysis of logs. The thesis also discusses possibilities for further development, including the implementation of real-time processing and the use of machine learning techniques for automatic anomaly detection.



## Keywords

Log analysis, cybersecurity, Extended Nagappan-Vouk algorithm, parallel data processing, security data visualization

# Obsah

# Seznam obrázků

# Seznam tabulek

# Seznam zdrojových kódů

# Seznam použitých zkratek

**API** Application Programming Interface
**APT** Advanced Persistent Threat
**AutoML** Automated Machine Learning
**CI VŠE** Centrum informatiky Vysoké školy ekonomické
**CP1250** Code Page 1250
**CSRF** Cross-Site Request Forgery
**CSIRT** Computer Security Incident Response Team
**CSV** Comma-Separated Values
**CVE** Common Vulnerabilities and Exposures
**ELK Stack** Elasticsearch, Logstash, Kibana
**ENV** Extended Nagappan-Vouk
**EU** Evropská unie
**FIFO** First In, First Out
**GIL** Global Interpreter Lock
**HDFS** Hadoop Distributed File System
**HDF5** Hierarchical Data Format version 5
**HTTP** Hypertext Transfer Protocol
**I/O** Input/Output
**IoT** Internet of Things
**IP** Internet Protocol
**JSON** JavaScript Object Notation
**KNIME** Konstanz Information Miner
**LDAP** Lightweight Directory Access Protocol
**ML** Machine Learning
**MVP** Minimum Viable Product
**NIS** Network and Information Systems
**NIST** National Institute of Standards and Technology
**NUKIB** Národní úřad pro kybernetickou bezpečnost
**NVD** National Vulnerability Database
**OS** Operating System
**Parquet** Apache Parquet
**PDF** Portable Document Format
**RBAC** Role-Based Access Control
**REST** Representational State Transfer
**SFTP** Secure File Transfer Protocol
**SIEM** Security Information and Event Management
**SMT** Simultaneous Multi-Threading
**SOAR** Security Orchestration, Automation and Response
**SOLID** Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, Dependency Inversion
**SQL** Structured Query Language
**SSH** Secure Shell
**UTF-8** Unicode Transformation Format 8-bit
**VŠE** Vysoká škola ekonomická
**XML** eXtensible Markup Language
**XSS** Cross-Site Scripting
**XZ** Formát komprese používaný pro logové soubory

# Úvod

Jako společnost jsme si v posledních desetiletích navykli na luxus, který nám poskytují informační a komunikační technologie a služby s nimi spojené. Poprvé v historii vyrůstá generace, která je od nejranějších let připojená na internet a která život bez něj téměř nezná (Prensky, 2001). Počet lidí, kteří jsou celosvětově nějakým způsobem připojeni na světovou síť překračuje 60 % a toto číslo neustále roste. (Opp, 2021) A s měnícím se světem a společností vzrostla také potřeba (nejen) firem, vlád nebo veřejných organizací postupně převádět většinu svých aktivit do virtuálního prostředí – izolace vůči informačním technologiím by, logicky, pro jedince i organizace znamenala nejen určitou formu ekonomické apokalypsy, ale také odříznutí od společnosti jako celku a ztrátu konkurenceschopnosti i orientace v dnešním světě.

I přesto, že se obrovské množství každodenních aktivit firem, vlád i jednotlivců dnes odehrává, například i díky rozmachu tzv. Internetu věcí (Internet of Things nebo IoT) a nebo kvůli externímu zásahu pandemie online („European Union Agency for Cybersecurity – Guidelines on Network and Information Security“, 2022; „Průzkum: Kyberbezpečnost českých firem“, 2022), je s podivem, jak málo pozornosti se stále kybernetickému zabezpečení a obecné bezpečnosti na internetu, respektive síti, věnuje (Kolouch et al., 2019). Projektoví a datoví manažeři se velmi často zaměřují spíše na vývoj a zisk než na data pojednávající o stavu kybernetické bezpečnosti a neuvědomují si, že určitá forma přehlížení prevence může být pro vývoj organizace a jejích služeb fatální („Průzkum: Kyberbezpečnost českých firem“, 2022). Důvodem bývá například nepřehlednost dat, které aktivita v kybernetickém prostoru generuje, ne vždy uživatelsky přátelská forma, ve které se data nachází, neschopnost data rychle vizualizovat pro snadnější interpretaci a případně zavedení či prohloubení kybernetických bezpečnostních opatření. Dalšími faktory může být i limitace časem a zdroji nebo mezery v legislativě, respektive její nedostatečnost.

Prostředí české jurisdikce se k současnému dni opírá zejména o zákon o kybernetické bezpečnosti, směrnici Evropského parlamentu NIS a vyhlášku o kybernetické bezpečnosti, která směrnici NIS zapracovává („European Union Agency for Cybersecurity – Guidelines on Network and Information Security“, 2022; „Legislativa kybernetické bezpečnosti“, 2023). Veškerá legislativa, která upravuje práva a povinnosti osob v oblasti kybernetické bezpečnosti nebo pokrývala její rizika, však nedosahují počtu vyšší než deset.

Výše zmíněná legislativa tak například zavádí základní úroveň bezpečnostních opatření, detekci kybernetických incidentů nebo jejich hlášení. Vyhláška kybernetické bezpečnosti pak v souladu se směrnicí Evropské unie (EU) upravuje (mimo jiné) obsah a strukturu bezpečnostní dokumentace, rozvíjí, jakým způsobem nahlašovat incidenty a jak nakládat s daty, které o aktivitě na síti provozovatel vede. Termín pro zaznamenávání informací o činnostech a běhu nějakých služeb nebo aplikací nazýváme logování. (Kolouch et al., 2019). Povinnost ze zákona logovat určuje dokonce §24 Sběr a vyhodnocování kybernetických bezpečnostních událost vyhlášky o kybernetické bezpečnosti (Vyhláška o kybernetické bezpečnosti, 2018).

Taková povinnost tak logicky platí i pro Vysokou školu ekonomickou.

## Vymezení tématu práce a její přínos

Tato práce vychází z výše zmíněné právní povinnosti sbírat a vyhodnocovat logy, propojuje teoretické znalosti kybernetické bezpečnosti i obecné znalosti její legislativy a implementuje komplexní řešení, které pomáhá pochopit jejich důležitost. Praktické řešení jako takové se opírá o principy nejen několikrát zmiňované kybernetické bezpečnosti, ale také datového inženýrství, datové analytiky a Self-Service Business Intelligence, které dohromady tvoří základ úspěšného porozumění aktivity na síti a poskytuje podklady nejen pro její zabezpečení, ale také pro bezpečnost jejích uživatelů i provozovatelů.

Je třeba poznamenat, že základy této práce byly položeny již během autorčiny stáže na VŠE v roce 2023, kdy byl vytvořen počáteční prototyp řešení využívající Microsoft Power BI. Tato počáteční zkušenost se stala odrazovým můstkem pro komplexnější přístup rozvinutý v této diplomové práci. Informace a kontexty uvedené v práci reflektují stav k červnu 2023, kdy autorka přerušila studium na dva roky. Její závěry jsou však relevantní i v roce obhajoby (2025).

Vysoká škola ekonomická (VŠE), a zejména její Centrum informatiky (CI), samozřejmě disponuje analytickými nástroji, které ji pomáhají detekovat kybernetické hrozby a monitorovat aktivitu na provozovaných službách. Nicméně dat je jen co se týče kybernetické bezpečnosti velké množství a kapacit na jejich analýzu málo. Mimo to jsou data ukládána na server, neexistuje tedy snadné připojení na vizualizační nástroje, jakou disponují například databáze nebo uživatelsky přátelštější Microsoft Excel. I proto v oddělení Datové analytiky a reportingu vznikl požadavek na vytvoření silnější koncentrace zejména na analýzu dat, datové inženýrství a finální vyřešení otázky vizualizace logů, to znamená včetně návrhu reportingového řešení.

Specifickým přínosem této práce je zaměření na automatizovaný přístup k analýze velkého objemu logů. V kontextu Vysoké školy ekonomické jde totiž o zpracování dat z více než 2 200 zdrojů, přičemž současné řešení pokrývá pouze zlomek z nich (přibližně 30 zdrojů). Tento nepoměr mezi dostupnými a skutečně analyzovanými daty vytváří značné bezpečnostní riziko, neboť mnoho potenciálních hrozeb může zůstat neodhaleno. Práce se snaží tento problém řešit prostřednictvím implementace rozšířeného algoritmu, který dokáže efektivně zpracovávat logy a identifikovat v nich klíčové vzorce a anomálie.

Dalším významným přínosem je propojení teoretických poznatků z oblasti kybernetické bezpečnosti s praktickými požadavky legislativy a IT provozu. Tím přispívá k vytvoření komplexního rámce pro správu a analýzu logů, který respektuje jak technické možnosti prostředí, tak i právní požadavky na bezpečnost dat.

V neposlední řadě práce poskytuje metodický přístup k řešení podobných problémů v oblasti

log managementu, který může být využit i v jiných organizacích, jež se potýkají s obdobnými výzvami v oblasti zpracování a analýzy velkého množství logů.

Jinými slovy práce popisuje vytvoření nástroje, který dokáže zpracovat velké množství dat – logů – ze serveru, automatizovaným způsobem určit sémantiku logů, respektive frekvenci slov v něm, tak, aby bylo jasné, která slova jsou kořeny logu a která se v čase mění a následně vytvořit návrh uživatelsky přátelské vizualizace takových dat, aby bylo pro pracovníky Centra informatiky dostatečně jednoduché z dat vyčíst (například) případnou příčinu nebo původ kybernetického incidentu.

## Cíl práce

Hlavním cílem této diplomové práce je vytvořit komplexní nástroj pro efektivní zpracování logů ze vzdáleného serveru pro potřeby kybernetické bezpečnosti Vysoké školy ekonomické. Tento nástroj zahrnuje algoritmus, který automatizovaným způsobem určuje sémantiku logů, identifikuje frekvenci slov a rozlišuje klíčové prvky - statické kořeny logu a proměnné části. Součástí nástroje je také vizualizační komponenta, která umožňuje pracovníkům Centra informatiky jednoduše a rychle interpretovat informace pro identifikaci příčin či zdrojů kybernetických incidentů. Vyvinuté řešení má za cíl posílit schopnost VŠE identifikovat, analyzovat a reagovat na potenciální bezpečnostní hrozby v reálném čase při výrazném snížení manuální práce potřebné pro analýzu rozsáhlých logových dat.

**Dílčí cíle této práce zahrnují:**

1. Analyzovat stávající technické a legislativní prostředí Vysoké školy ekonomické, aby byly zohledněny specifické požadavky na zpracování a vizualizaci logů.
2. Provést sémantickou analýzu logů s využitím algoritmu Extended Nagappan-Vouk pro frekvenční interpretaci dat.
3. Vytvořit metodiku pro kontinuální sběr a analýzu logů, která umožní dlouhodobé sledování a vyhodnocování bezpečnostních událostí.

V rámci prvního dílčího cíle se práce zaměřuje na podrobnou analýzu současného stavu, včetně identifikace klíčových výzev a omezení, kterým Centrum informatiky čelí při zpracování logů. Tato analýza slouží jako východisko pro návrh řešení, které bude odpovídat specifickým potřebám a možnostem prostředí VŠE.

Druhý dílčí cíl se zaměřuje na implementaci sémantické analýzy logů pomocí algoritmu Extended Nagappan-Vouk, který umožňuje identifikovat statické a proměnné části logů. Tato analýza je klíčová pro efektivní zpracování a interpretaci velkého objemu logů.

Třetí dílčí cíl se zaměřuje na vytvoření metodiky pro kontinuální sběr a analýzu logů. Ta zahrnuje definici procesů, rolí a odpovědností, stejně jako stanovení harmonogramu a postupů pro pravidelné vyhodnocování výsledků. Cílem je zajistit, aby navržené řešení nebylo jednorázovým projektem, ale stalo se součástí běžných operací Centra informatiky.

## Struktura práce

Práce je rozdělena do tří samostatných tematických celků:

1. Byznysové zadání a kontext.
2. Technické zadání.
3. Řešení a validace.

Každý z nich představuje právě jeden logický krok v řešení zvoleného problému a každý takový celek tak obsahuje kapitoly odpovídající jeho zaměření. První celek se věnuje kontextu, který tvoří základní rámec projektu, a popisuje byznysové zadání. Druhý celek je zaměřen na technické zadání a návrh řešení. Poslední celek se soustředí na implementaci řešení, jeho validaci a diskusi o dosažených výsledcích.

Tato struktura byla autorkou zvolena s ohledem na komplexnost řešeného problému a potřebu propojit teoretické znalosti z různých oblastí s praktickým řešením. Postupné rozvedení problematiky od obecného kontextu přes technické aspekty až po konkrétní implementaci umožňuje čtenáři lépe pochopit celý proces návrhu a realizace řešení.

### Byznysové zadání a kontext

Kapitola věnující se byznysovému zadání se zabývá obecnými požadavky na zpracování logů a monitoring kybernetických událostí, které vyplývají z legislativního rámce a provozních potřeb. Samostatná kapitola je věnována legislativním požadavkům, zejména vyhlášce o kybernetické bezpečnosti a zákonu o kybernetické bezpečnosti, což jsou klíčové normy pro pochopení zadání. Součástí je představení prostředí Vysoké školy ekonomické, s důrazem na její IT infrastrukturu a specifické potřeby v oblasti kybernetické bezpečnosti. Jsou zde identifikovány hlavní výzvy, kterým Centrum informatiky čelí při správě a analýze logů, včetně limitů stávajících řešení a rostoucích požadavků na bezpečnost a efektivitu.

Dalším důležitým aspektem, který je v této části adresován, je rostoucí tlak na soulad s legislativními požadavky a potřeba efektivního reportingu pro management univerzity. Jsou zde představeny klíčové parametry, které musí navrhované řešení splňovat, aby vyhovělo jak technickým, tak i právním a organizačním požadavkům.

### Technické zadání

Druhá část práce představuje technické aspekty projektu a technologie nezbytné k dosažení cíle. První kapitola se zaměřuje na limity prostředí, zejména omezení související s logy uloženými na serveru, a zavádí koncept tzv. multiprocessingu jako klíčového prvku pro zpracování dat. Dále se celek zaměřuje na návrh řešení prostřednictvím vizuálního programování, které je prezentováno jako efektivní metoda i pro uživatele s menšími technickými znalostmi. Sou-

částí je také popis řešení založeného na Extended Nagappan-Vouk algoritmu, který je použit pro frekvenční analýzu logů.

V této části je zároveň věnována pozornost technickým specifikům prostředí VŠE, včetně architektury serverů, struktury logovaných dat a existujících nástrojů pro jejich analýzu. Jsou zde detailně rozebrány výhody a nevýhody různých přístupů k zpracování logů, s důrazem na potřebu efektivního využití dostupných zdrojů a minimalizaci dopadu na běžný provoz systémů.

Významnou součástí této části je také diskuse o vhodných technologiích pro implementaci navrženého řešení, včetně porovnání různých relevantních nástrojů z hlediska jejich výkonnosti, flexibility a integrace se stávajícími systémy. Rovněž je část pozornosti věnována vizuálnímu programování, které představuje zajímavou alternativu k tradičním přístupům a může usnadnit zapojení širšího okruhu pracovníků do procesu analýzy logů.

### Řešení a validace

Třetí část se zaměřuje na implementaci navrženého algoritmu a jeho validaci. První kapitola popisuje proces získávání dat ze serveru a vytvoření datové pumpy, která zajišťuje kontinuální zpracování logů. Druhá kapitola se věnuje frekvenční analýze logů, která umožňuje identifikaci klíčových vzorců v datech. Třetí kapitola se zaměřuje na návrh a realizaci reportingového řešení, které zlepšuje schopnosti Centra informatiky v oblasti Self-Service Business Intelligence. Poslední kapitola uzavírá tuto část diskusí nad výsledky a zahrnuje úvahy o možných zlepšeních a optimalizacích.

Tato část obsahuje konkrétní popis implementace algoritmu pro zpracování logů, včetně ukázek kódu a vysvětlení klíčových funkcí. Jsou zde detailně popsány jednotlivé kroky procesu, od extrakce dat ze serveru, přes jejich předzpracování, až po aplikaci frekvenční analýzy a identifikaci vzorců. Důležitou součástí je také popis optimalizačních technik, které byly použity pro zvýšení výkonnosti algoritmu, zejména s ohledem na zpracování velkého objemu dat.

V rámci validace jsou prezentovány výsledky testování navrženého řešení na reálných datech z prostředí VŠE. Jsou zde porovnány výkonnostní charakteristiky navrhovaného řešení s existujícími přístupy, s důrazem na rychlost zpracování, přesnost detekce vzorců a využití systémových zdrojů. Součástí je také vyhodnocení uživatelské zpětné vazby na navržené reportingové řešení, včetně identifikace silných stránek a oblastí pro další zlepšení.

## Výstup práce

Tato diplomová práce poskytuje komplexní řešení v podobě nástroje, který dokáže automatizovaně zpracovávat logy ze serveru a identifikovat jejich sémantiku. Výstupem je uživatelsky

přívětivé řešení, které umožňuje vizualizaci klíčových dat a poskytuje pracovníkům Centra informatiky Vysoké školy ekonomické přehledné informace pro analýzu a řešení kybernetických incidentů.

Konkrétním výstupem práce je implementace nástroje v programovacím jazyce Python, který je optimalizován pro efektivní zpracování velkého objemu logů. Součástí řešení je také sada nástrojů pro vizualizaci výsledků, která nabízí interaktivní dashboardy a reporty pro různé úrovně uživatelů. Důležitým aspektem výstupu je také metodika pro kontinuální sběr a analýzu logů, která definuje procesy, role a odpovědnosti pro efektivní využití navrženého řešení v každodenním provozu Centra informatiky. Tato metodika zahrnuje také doporučení pro pravidelnou aktualizaci a rozšiřování řešení v reakci na měnící se požadavky a hrozby.

V neposlední řadě je výstupem práce také dokumentace, která detailně popisuje architekturu, funkcionalitu a použití navrženého řešení. Tato dokumentace je koncipována tak, aby sloužila jak pro technické uživatele, kteří budou řešení dále rozvíjet, tak i pro koncové uživatele, kteří s ním budou pracovat v rámci svých denních aktivit.

## Přínos práce

Práce přináší důležitý příspěvek k efektivnímu využití dat v oblasti kybernetické bezpečnosti. Navržené řešení pomáhá nejen s identifikací potenciálních hrozeb, ale také s optimalizací analytických procesů. Tím přispívá ke zvýšení bezpečnosti akademických a jiných síťových prostředí a současně poskytuje inovativní přístupy k analýze a vizualizaci dat.

Významným přínosem je zejména automatizace procesu zpracování logů, která výrazně snižuje časovou náročnost analýzy a uvolňuje kapacity pracovníků Centra informatiky pro strategičtější úkoly. Podle předběžných odhadů by mělo implementované řešení zkrátit dobu analýzy logů z původních hodin na pouhých několik minut, což představuje markantní zvýšení efektivity.

Dalším důležitým přínosem je rozšíření pokrytí monitorovaných zdrojů. Zatímco stávající řešení umožňuje analyzovat pouze zlomek všech dostupných zdrojů logů, navrhovaný přístup je škálovatelný a umožňuje postupné zapojení většiny z více než 2 200 zdrojů. To výrazně zlepšuje schopnost detekce potenciálních bezpečnostních incidentů a poskytuje ucelenější pohled na dění v síti.

V neposlední řadě práce přispívá k posílení souladu s legislativními požadavky v oblasti kybernetické bezpečnosti. Implementované řešení umožňuje efektivnější sledování a dokumentaci bezpečnostních událostí, což je klíčový požadavek vyhlášky o kybernetické bezpečnosti a souvisejících předpisů.

Z akademického hlediska práce propojuje poznatky z různých disciplín, od kybernetické bezpečnosti, přes datové inženýrství, až po vizualizaci dat, a demonstruje jejich praktické využití při řešení reálného problému. Tím přispívá k rozvoji interdisciplinárního přístupu k bezpeč-

nostním výzvám v digitálním prostředí.

## Předpoklady a omezení práce

Práce předpokládá dostupnost dostatečného množství logovaných dat a spolupráci pracovníků Centra informatiky při jejich zpracování. Omezení spočívají zejména ve specifikách prostředí Vysoké školy ekonomické, jako jsou omezené možnosti přístupu k databázím a vizualizačním nástrojům. Dalším faktorem je objem dat, který může ovlivnit výkonnost navrženého řešení.

Pro úspěšnou realizaci projektu je nutné mít přístup k serverům, na kterých jsou logy uloženy, a disponovat dostatečnými oprávněními pro jejich extrakci a analýzu. Práce předpokládá, že tyto požadavky budou splněny a že nebude docházet k výrazným změnám v struktuře logovaných dat během implementace řešení.

Významným omezením je také heterogenita formátů logů, které pocházejí z různých zdrojů a mohou mít odlišnou strukturu a sémantiku. To komplikuje návrh univerzálního algoritmu, který by dokázal efektivně zpracovávat všechny typy logů. Práce se proto zaměřuje především na nejčastější formáty logů, které se v prostředí VŠE vyskytují, a poskytuje rámec pro postupné rozšiřování podpory pro další formáty.

Dalším omezením jsou výpočetní zdroje, které jsou k dispozici pro zpracování logů. Vzhledem k velkému objemu dat je nutné pečlivě optimalizovat algoritmus tak, aby byl schopen efektivně využívat dostupné prostředky a nezatěžoval nadměrně systémy, které jsou současně využívány pro běžný provoz univerzity.

V neposlední řadě je třeba zmínit omezení vyplývající z potřeby zajistit bezpečnost samotného procesu analýzy logů. Je nutné dbát na to, aby přístup k citlivým informacím byl omezen pouze na oprávněné osoby a aby samotný proces analýzy nepředstavoval bezpečnostní riziko pro monitorované systémy.

## Metodika práce

Práce kombinuje teoretické poznatky s praktickou implementací. Zahrnuje analýzu legislativního rámce, návrh algoritmu a jeho implementaci za použití frekvenční analýzy a vizuálního programování. Dosažení hlavního cíle předpokládá úspěšnou realizaci několika vzájemně propojených kroků.

Nejprve je nutné navrhnout efektivní metodu pro sběr a předzpracování logů ze serveru, která by dokázala pracovat s rozsáhlými objemy dat. Poté je třeba implementovat algoritmus pro frekvenční analýzu, který dokáže identifikovat opakující se vzorce a případné odchylky. Posledním krokem je pak vytvoření vhodné vizualizace, která umožní intuitivní a uživatelsky

přátelskou interpretaci výsledků. Finální řešení je validováno pomocí diskuse nad jeho přínosy a omezeními, což zajišťuje jeho praktickou využitelnost.

# 1. Byznysové zadání a kontext

Tato kapitola se věnuje celkovému kontextu projektu a byznysovému zadání, které stálo na počátku vývoje řešení pro zpracování a analýzu logů v prostředí Vysoké školy ekonomické v Praze. Nejprve představuje obecné požadavky na zpracování logů a monitoring kybernetických událostí, které vyplývají z legislativního rámce a provozních potřeb. Následně se zabývá legislativními požadavky, zejména vyhláškou o kybernetické bezpečnosti a zákonem o kybernetické bezpečnosti. Kapitola také zahrnuje popis aktuálního stavu v oblasti log managementu na VŠE a identifikuje hlavní výzvy a omezení, která bylo třeba v rámci projektu adresovat.

## 1.1 Prostředí a motivace projektu

Centrum informatiky Vysoké školy ekonomické v Praze (CI VŠE) je ze své podstaty odpovědné za správu a údržbu univerzitní IT infrastruktury, která zahrnuje servery, aplikace a také širokou škálu síťových zařízení školy („Centrum informatiky – Základní informace“, 2025). Taková infrastruktura generuje souvisle velké množství logů – od systémových událostí a chyb, bezpečnostní záznamy, až po autentizační procesy na úrovni jednotlivých uživatelů. Pro efektivní práci v oblasti kyberbezpečnosti je klíčové takovým procesům nejen dobře rozumět, ale také je nastavit tak, aby uměli v reálném čase poskytnout informaci, která uživateli umožní se v oblasti zorientovat (National Institute of Standards and Technology, 2006).

Motivací projektu, a tedy i této práce, je zefektivnit a zautomatizovat procesy analýzy logů, které jsou momentálně časově velmi náročné a potýkají se s řadou omezení. Stávající řešení, které se opírá o software LOGmanager, pokrývá pouze zlomek všech dostupných datových zdrojů (30 z celkových asi 2 200), a proto není možné získat ucelený přehled o dění v celé síti. Případné bezpečnostní incidenty, jako jsou neoprávněné přístupy nebo anomálie v datech, mohou být díky roztříštěnosti logů snadno přehlédnuty. Provozní spolehlivost i bezpečnost systému může těmito nedostatky značně utrpět a do budoucna může vést mimojiné ke zvýšení finančních nákladů na řešení incidentů („The SIEM Buyer's Guide“, 2023).

Tato práce navazuje na autorčinu odbornou stáž v Centru informatiky VŠE realizovanou v roce 2023, kdy byly položeny základy řešení problematiky pomocí vizualizací v Microsoft Power BI. Vzhledem k přerušení studia jsou informace a analýzy v této práci aktuální k červnu 2023, nicméně identifikované problémy v oblasti analýzy logů a navržené principy řešení zůstávají relevantní i v současném kontextu.

### 1.1.1 Současný stav

Jak již bylo zmíněno, LOGmanager, který je momentálně nasazený pro analýzu logů, zpracovává data pouze z 30 zdrojů z celkového počtu přesahujícího 2 200. Zbylé zdroje jsou ukládány především na centrálním serveru Levandule, který disponuje složitou hierarchií adresářů (podle zdroje, roku, měsíce a dne). Kvůli absenci centralizovaného systému je navíc obtížné vyhledávat a agregovat relevantní data napříč různými časovými obdobími či aplikacemi (National Institute of Standards and Technology, 2006). Toto roztříštěné prostředí s sebou přináší značné provozní obtíže, které lze shrnout do následujících bodů:

1. **Omezené možnosti stávajících nástrojů**: Neexistující nebo nedostatečná integrace klíčových zdrojů logů do jednoho prostředí způsobuje neefektivní proces vyhodnocování potenciálních hrozeb (National Institute of Standards and Technology, 2006; „SIEM: Security Information & Event Management Explained“, 2024).
2. **Nedostatečné pokrytí dat**: Většina logů není analyzována, nebo je analyzována manuálně a nahodile. Tato situace vede k nadměrné zátěži pracovníka odpovědného za kybernetickou bezpečnost a k riziku opomenutí důležitých událostí (Karlsen et al., 2024).
3. **Technické limity**: Dosavadní využití nástrojů jako KNIME vede k dlouhé době zpracování. Analýza 500 logů v KNIME trvá přibližně 2 minuty, což při objemu logů v řádu stovek tisíců souborů znamená prakticky neproveditelný úkol pro jediného pracovníka.

Tyto faktory mají zásadní vliv na kybernetickou bezpečnost a také na efektivitu práce týmů, které se musejí manuální analýzou logů zabývat namísto strategické činnosti (National Institute of Standards and Technology, 2006).

### 1.1.2 Kybernetické hrozby a jejich dopad

V současném digitálním prostředí čelí akademické instituce, včetně Vysoké školy ekonomické, rostoucímu počtu kybernetických hrozeb („European Union Agency for Cybersecurity – Guidelines on Network and Information Security“, 2022). Tyto hrozby zahrnují široké spektrum útoků, od relativně jednoduchých pokusů o neoprávněný přístup, přes phishingové kampaně, až po sofistikované útoky typu Advanced Persistent Threat (APT). Úspěšné útoky mohou mít vážné důsledky pro fungování univerzity, včetně narušení výuky, ztráty důvěrných dat, finančních ztrát a poškození reputace (Kolouch et al., 2019).

V případě VŠE je situace o to komplikovanější, že její IT infrastruktura je velmi rozsáhlá a rozmanitá, obsahující různé typy systémů, aplikací a uživatelských zařízení. Tato heterogenita zvyšuje potenciální plochu pro útok a komplikuje implementaci jednotných bezpečnostních opatření. Navíc, akademické prostředí je charakteristické svou otevřeností a vysokou mírou mobility uživatelů, což dále zvyšuje bezpečnostní rizika („Průzkum: Kyberbezpečnost českých firem“, 2022).

Efektivní analýza logů představuje jeden z klíčových nástrojů pro detekci a reakci na ky-

bernetické hrozby. Logy obsahují cenné informace o aktivitách v síti, které mohou pomoci identifikovat podezřelé vzorce chování, neoprávněné přístupy nebo anomálie indikující potenciální bezpečnostní incidenty. Bez schopnosti rychle a efektivně analyzovat tyto logy jsou bezpečnostní týmy často nuceny reagovat až na již probíhající nebo dokončené útoky, což výrazně zvyšuje potenciální škody (National Institute of Standards and Technology, 2006).

## 1.2 Legislativní kontext

Kybernetická bezpečnost není v České republice vnímána jen jako téma pro odborníky, ale i jako legislativně upravená povinnost, ke které se přistupuje systematicky a plánovitě. Klíčovým předpisem v této oblasti je zákon č. 181/2014 Sb., o kybernetické bezpečnosti a o změně souvisejících zákonů (zákon o kybernetické bezpečnosti), který společně s prováděcími předpisy vytváří základní rámec pro zajištění kybernetické bezpečnosti v ČR.

### 1.2.1 Zákon o kybernetické bezpečnosti

Zákon o kybernetické bezpečnosti definuje práva a povinnosti osob a působnost a pravomoc orgánů veřejné moci v oblasti kybernetické bezpečnosti. Stanovuje systém pro identifikaci, kategorizaci a ochranu informačních systémů a sítí kritické informační infrastruktury, významných informačních systémů a dalších prvků kybernetické bezpečnosti („Legislativa kybernetické bezpečnosti“, 2023).

Pro Vysokou školu ekonomickou je relevantní zejména otázka, zda a v jakém rozsahu spadá pod působnost tohoto zákona. Ačkoliv VŠE jako celek nespadá do kategorie provozovatelů kritické informační infrastruktury, některé její informační systémy mohou být klasifikovány jako významné informační systémy nebo systémy základní služby, což by znamenalo povinnost dodržovat příslušná bezpečnostní opatření.

### 1.2.2 Vyhláška o kybernetické bezpečnosti

Konkrétní povinnosti v oblasti bezpečnostních opatření jsou stanoveny ve vyhlášce č. 82/2018 Sb., o bezpečnostních opatřeních, kybernetických bezpečnostních incidentech, reaktivních opatřeních, náležitostech podání v oblasti kybernetické bezpečnosti a likvidaci dat (vyhláška o kybernetické bezpečnosti) (Vyhláška o kybernetické bezpečnosti, 2018).

Pro oblast log managementu jsou klíčová ustanovení obsažená v § 24 vyhlášky, který upravuje sběr a vyhodnocování kybernetických bezpečnostních událostí. Podle tohoto ustanovení jsou povinné osoby (včetně provozovatelů významných informačních systémů) povinny:

1. Zajistit sběr informací o kybernetických bezpečnostních událostech a kybernetických bezpečnostních incidentech.

2. Průběžně vyhodnocovat kybernetické bezpečnostní události, a to nejméně v rozsahu určení, zda se jedná o kybernetický bezpečnostní incident.
3. Zajistit sběr a vyhodnocení kybernetických bezpečnostních událostí a incidentů podle určených bezpečnostních potřeb a pravidel.

Vyhláška dále specifikuje požadavky na zaznamenávání událostí důležitých pro provoz informačního a komunikačního systému, zejména přihlašování a odhlašování uživatelů, činnosti administrátorů, spouštění a ukončování systémů a aplikací, a další relevantní aktivity. Tyto záznamy musí být uchovávány po dobu stanovenou právními předpisy nebo na základě analýzy rizik, ale ne méně než 3 měsíce (Kolouch et al., 2019).

### 1.2.3 Směrnice NIS a její implementace

Na evropské úrovni je klíčovým předpisem Směrnice Evropského parlamentu a Rady EU 2016/1148 ze dne 6. července 2016 o opatřeních k zajištění vysoké společné úrovně bezpečnosti sítí a informačních systémů v Unii (směrnice NIS). Tato směrnice byla do českého právního řádu implementována prostřednictvím zákona o kybernetické bezpečnosti a souvisejících předpisů („European Union Agency for Cybersecurity – Guidelines on Network and Information Security“, 2022).

Směrnice NIS stanovuje požadavky na členské státy EU, aby zavedly vnitrostátní strategie pro bezpečnost sítí a informačních systémů, určily příslušné vnitrostátní orgány a zřídily skupiny pro reakci na incidenty v oblasti počítačové bezpečnosti (CSIRT). Dále ukládá členským státům povinnost určit provozovatele základních služeb a poskytovatelé digitálních služeb, kteří budou muset přijmout vhodná bezpečnostní opatření a oznamovat závažné incidenty.

Pro oblast log managementu jsou ze směrnice NIS relevantní zejména požadavky na implementaci opatření pro sledování, auditing a testování, které mají zajistit efektivní detekci bezpečnostních incidentů a reakci na ně („Legislativa kybernetické bezpečnosti“, 2023).

## 1.3 Klíčové problémy při naplňování legislativních požadavků

Bez ohledu na právní předpoklady je naplňování požadavků velmi často problematické. Současná řešení obvykle pracují s omezeným počtem logů, a to z důvodu limitovaných kapacit a nedostatečnosti zpracovávacích nástrojů. Na VŠE je například v současnosti zpracováváno 20 - 30 zdrojů s logy v době, kdy má infrastruktura 2 200 integrovaných datových zdrojů. Automatické vyhodnocování rizikovosti je následně velmi omezené, což vede k tomu, že analýza je zajišťována ručně jediným pracovníkem z Centra informatiky odpovědným za kybernetickou bezpečnost. Tento individuální přístup značně limituje rozsah monitoringu a zvyšuje riziko přehlédnutí potenciálních hrozeb. Náklady i zbytková rizikovost kybernetických i provozních témat je pak zásadní (Karlsen et al., 2024).

### 1.3.1 Problémy s kapacitou a škálováním

Jedním z hlavních problémů při implementaci efektivního log managementu je obrovský objem dat, který musí být zpracován. V prostředí VŠE se jedná o miliony záznamů denně, přičemž stávající infrastruktura a nástroje nejsou navrženy pro zpracování takového množství dat.

Dalším problémem je heterogenita formátů logů, které pocházejí z různých zdrojů (aplikace, síťová zařízení, servery atd.) a mají různou strukturu a obsah. To komplikuje jejich centralizaci, normalizaci a analýzu. Bez standardizovaného formátu je obtížné automatizovat proces analýzy a identifikace relevantních událostí (National Institute of Standards and Technology, 2006).

### 1.3.2 Problémy s efektivitou analýzy

Současný proces analýzy logů je značně neefektivní a časově náročný. Manuální procházení a třídění logů zabírá velké množství času a podléhá lidské chybě a navíc, vzhledem k omezeným personálním kapacitám, není možné analyzovat všechny dostupné logy, což zvyšuje riziko přehlédnutí důležitých bezpečnostních událostí (Splunk, 2022).

Dalším problémem je nedostatek automatizovaných nástrojů pro detekci anomálií a korelaci událostí. Bez těchto nástrojů je obtížné identifikovat komplexní útoky, které zahrnují více kroků a různých typů aktivit (Kolouch et al., 2019).

### 1.3.3 Problémy s reportingem a komunikací

V neposlední řadě existují problémy s reportingem a komunikací výsledků analýzy logů. Chybí uživatelsky přívětivé rozhraní, které by umožnilo snadno interpretovat výsledky analýzy a prezentovat je ve srozumitelné formě pro různé cílové skupiny (IT specialisty, management, uživatele) (He et al., 2017).

Současné reporty jsou často technicky orientované a postrádají kontextové informace, které by umožnily rychlé rozhodování a reakci na identifikované hrozby. To vede k prodlevám v řešení potenciálních bezpečnostních incidentů a zvyšuje riziko jejich eskalace (Karlsen et al., 2024).

## 1.4 Požadavky zadavatele a omezení

Zadavatelem projektu je Centrum informatiky Vysoké školy ekonomické v Praze (CI VŠE), které je odpovědné za správu a údržbu univerzitní IT infrastruktury. V rámci své koncepce rozvoje plánuje CI VŠE posílit schopnosti log managementu aktivit než tomu bylo dosud a zároveň se počítá se zavedením Security Information and Event Management (SIEM) řešení, které by násobně zvýšilo počet zpracovávaných zdrojů logů. Součástí koncepce je také zavedení tzv. selfservice log dashboardingu, který by umožnil efektivnější práci s logovou analýzou i pro jediného pracovníka odpovědného za kybernetickou bezpečnost. Tím by došlo ke zkrácení doby analýzy záznamů z hodin na minuty a k výraznému rozšíření analytických možností dostupných jedné osobě. Toto řešení je zaměřené přímo na potřeby dr. Šimečka, manažera kybernetické bezpečnosti.

### 1.4.1 Funkční požadavky

Zadavatelem formulované požadavky na finální výstup projektu lze rozdělit do tří oblastí Automatizovaný systém pro sběr a analýzu logů, Intuitivní dashboard a interaktivní vizualizace a Soulad s legislativou a bezpečnostní standardy:

1. **Automatizovaný systém pro sběr a analýzu logů**
   - 1A. Schopnost připojit se na více než 2 200 zdrojů.
   - 1B. Zpracování velkých objemů dat bez zbytečných prodlev.
   - 1C. Možnost adaptace na nové formáty logů a jejich struktury.
   - 1D. Automatizace procesu analýzy logů, a to jak v reálném čase (on-line), tak v dávkovém zpracování (batch).
   - 1E. Snížení celkové doby analýzy z odhadovaných 12 hodin na méně než 15 minut, a to bez nutnosti neustálé lidské intervence.
2. **Intuitivní dashboard a interaktivní vizualizace**
   - 2A. Self-Service Business Intelligence pojetí, kdy mohou uživatelé sami filtrovat a analyzovat data.
   - 2B. Přehledné zobrazení klíčových metrik (počty neúspěšných přihlášení, kritické systémové chyby, trendy v časových řadách apod.).
   - 2C. Návrh uživatelsky přívětivého dashboardu, který umožní tzv. Self-Service Business Intelligence.
   - 2D. Přímý přístup ke klíčovým ukazatelům a možnost je dále analyzovat pomocí flexibilních filtrů a grafů.
3. **Soulad s legislativou a bezpečnostní standardy**
   - 3A. Soulad s vyhláškou č. 82/2018 Sb. o kybernetické bezpečnosti a dalšími nařízeními.
   - 3B. Transparentní evidence incidentů, možnost verifikace a auditů.
   - 3C. Zajištění schopnosti transparentní evidence incidentů a uchování příslušných logů.

### 1.4.2 Nefunkční požadavky

Kromě funkčních požadavků definujeme také nefunkční požadavky, které specifikují kvalitativní charakteristiky systému:

1. **Výkon a škálovatelnost**

NF1A. Systém musí být schopen zpracovat až 500 logů za méně než 10 sekund.[1]

NF1B. Řešení musí být škálovatelné pro budoucí nárůst objemu dat.

NF1C. Odezva dashboardu nesmí přesáhnout 5 sekund při běžném zatížení.

2. **Spolehlivost a dostupnost**

NF2A. Systém musí být dostupný minimálně 99,5 % času (povolený výpadek maximálně 43,8 hodin ročně).

NF2B. Systém musí obsahovat mechanismy pro zotavení po chybě bez manuálního zásahu.

3. **Udržitelnost a rozšiřitelnost**

NF3A. Modulární architektura umožňující snadné přidávání nových zdrojů logů.

NF3B. Dokumentovaný kód s komentáři a testovacím pokrytím alespoň 70 %.

4. **Uživatelská přívětivost**

NF4A. Intuitivní uživatelské rozhraní nevyžadující více než 2 hodiny školení pro běžného IT pracovníka.

NF4B. Konzistentní design a terminologie napříč celým systémem.

NF4C. Responzivní design dashboardu fungující na různých zařízeních a velikostech obrazovky.

U funkčních i nefunkčních požadavkům dojde k vyhodnocení v závěrečné části práce ve validaci, kde pro každý požadavek zhodnotím, zda byl naplněn zcela, částečně, nebo vůbec.

### 1.4.3 Technická omezení

Na základě rešerše literatury (He et al., 2017; Karlsen et al., 2024; National Institute of Standards and Technology, 2006; Vyhláška o kybernetické bezpečnosti, 2018) a konzultací se zástupci Centra informatiky byla identifikována následující technická omezení, která je třeba brát v úvahu při návrhu a implementaci řešení:

1. **Technické limity hardwaru a síťového připojení** – Při zpracovávání masivního objemu logů může být propustnost systému kritickým faktorem (Karlsen et al., 2024).

[1] Tato metrika byla stanovena na základě analýzy stávajících procesů a potřeb Centra informatiky VŠE. Logové soubory se v prostředí VŠE významně liší svou velikostí a strukturou - od jednoduchých autentizačních logů (průměrně 0,5-2 KB na záznam) až po komplexní aplikační logy (5-50 KB na záznam). Uvedená výkonnostní metrika tak představuje průměrnou hodnotu napříč různými typy logů, přičemž tento požadavek byl stanoven jako kompromis mezi rychlostí zpracování a dostupnými výpočetními zdroji. Při testování bude použit reprezentativní vzorek různých typů logů z produkčního prostředí pro zajištění relevantnosti měření. Termín "zpracování"zde zahrnuje načtení logu, jeho parsování, normalizaci, aplikaci Extended Nagappan-Vouk (ENV) algoritmu pro identifikaci statických a dynamických částí a uložení výsledku do strukturované podoby. Tato výkonnostní úroveň je nezbytná pro efektivní analýzu v téměř reálném čase, což je klíčový požadavek pro včasnou detekci potenciálních bezpečnostních incidentů.

Je tedy nezbytné, aby byl k dispozici dostatečný výpočetní výkon i stabilní připojení.
2. **Velký objem dat** – Vzhledem k tomu, že na serveru Levandule se nacházejí tisíce adresářů a stovky tisíc souborů, je třeba navrhnout efektivní mechanizmy indexace, parsování a archivace.
3. **Potřeba souladu s legislativními požadavky a zajištění bezpečnosti dat** – Při implementaci bude nutné zohlednit bezpečnostní standardy a postupy (National Institute of Standards and Technology, 2006, s. 28-30), (Vyhláška o kybernetické bezpečnosti, 2018).
4. **Dostupnost kompetentních pracovníků** – Úspěšná implementace a následná správa systému vyžaduje personální zajištění odborníky se znalostmi z oblasti kybernetické bezpečnosti a datové analýzy.

Tyto omezení je nutné mít neustále na paměti při tvorbě harmonogramu a při samotné implementaci, aby nedošlo k nerealistickému plánování a následnému prodražení či opoždění projektu (He et al., 2017).

## 1.5 Očekávané přínosy

Důkladně zpracovaný systém pro log management a analýzu má strategický význam jak z krátkodobého, tak dlouhodobého hlediska. Mezi nejvýznamnější přínosy patří: zlepšení detekce hrozeb, snížení provozních nákladů, efektivní využití dat a legislativní soulad.

### 1.5.1 Zlepšení detekce hrozeb

Díky automatizovanému a centralizovanému sběru dat bude možné v reálném čase či téměř v reálném čase rozpoznávat anomálie a podezřelou aktivitu („European Union Agency for Cybersecurity – Guidelines on Network and Information Security“, 2022). Tím se výrazně sníží reakční doba a omezí prostor pro rozvoj kritických incidentů.

Implementace pokročilých analytických metod, jako je frekvenční analýza logů, umožní identifikovat subtilnější vzorce chování, které by při manuální analýze mohly zůstat nepovšimnuty. To je zvláště důležité pro detekci sofistikovaných útoků, které se snaží maskovat svou aktivitu a vyhýbat se běžným detekčním mechanismům (Kolouch et al., 2019).

### 1.5.2 Snížení provozních nákladů

Vysoce automatizovaný proces zpracování logů umožní pracovníkům Centra informatiky VŠE věnovat se spíše strategickým aktivitám než manuální analýze a třídění dat (Karlsen et al., 2024). To vede k efektivnějšímu využití lidských zdrojů a snížení provozních nákladů.

Automatizace také eliminuje potřebu manuálního zpracování velkého objemu dat, což snižuje riziko lidské chyby a zvyšuje spolehlivost celého procesu. To je zvláště důležité v kontextu kybernetické bezpečnosti, kde i drobné přehlédnutí může mít závažné důsledky (Splunk, 2022).

### 1.5.3 Efektivní využití dat

Logy, které dosud ležely ladem nebo byly nedostupné, budou začleněny do analytických procesů, což povede k lepšímu porozumění dění v celé síti. Toto povědomí pak pomůže identifikovat slabá místa infrastruktury i prostor pro optimalizaci (National Institute of Standards and Technology, 2006).

Centralizovaný sběr a analýza logů také umožní získat komplexnější pohled na bezpečnostní stav IT infrastruktury VŠE. To je důležité pro strategické plánování a prioritizaci bezpečnostních investic. Analýza historických dat může pomoci identifikovat dlouhodobé trendy a vzorce, které mohou indikovat systematické problémy nebo zranitelnosti („Průzkum: Kyberbezpečnost českých firem“, 2022).

### 1.5.4 Legislativní soulad

Centralizované úložiště a jasně definované postupy pro archivaci, vyhledávání a dokumentaci incidentů usnadní plnění zákonných požadavků a zlepší auditovatelnost systému (Vyhláška o kybernetické bezpečnosti, 2018). To je zvláště důležité v kontextu rostoucích regulačních požadavků v oblasti kybernetické bezpečnosti.

Schopnost rychle generovat komplexní reporty o bezpečnostních událostech a incidentech také usnadní komunikaci s regulačními orgány a jinými zainteresovanými stranami. To může být klíčové v případě bezpečnostních incidentů, kdy je třeba rychle poskytnout relevantní informace a prokázat, že byly přijaty odpovídající bezpečnostní opatření („Legislativa kybernetické bezpečnosti“, 2023).

## 1.6 Shrnutí byznysového zadání

Byznysové zadání projektu, který se zabývá rozšířením a zkvalitněním analýzy logů na Vysoké škole ekonomické v Praze, je zásadní pro zajištění kybernetické bezpečnosti a provozní spolehlivosti. Nedostatky současných řešení (pouze 30 zdrojů připojených k LOGmanageru, dlouhá doba analýzy v KNIME) ohrožují stabilitu celé IT infrastruktury a zároveň omezují možnosti vyhledávání informací potřebných pro kvalifikovaná rozhodnutí.

Cílem projektu je proto zavést plně automatizovaný systém, který dokáže pracovat s rozsáhlým objemem logů a nabízí pokročilé nástroje pro detekci anomálií. Kromě samotné analýzy je klíčový také návrh přehledného dashboardu pro Self-Service Business Intelligence, jenž

usnadní práci zaměstnancům Centra informatiky a zvýší transparentnost i rychlost reakcí na potenciální bezpečnostní incidenty.

Dlouhodobým přínosem takového řešení je nejen minimalizace provozních nákladů a snížení míry lidských chyb, ale také posílení legislativního souladu a celkové důvěryhodnosti VŠE v oblasti kybernetické bezpečnosti. Projekt představuje moderní krok ke zvýšení odolnosti a efektivity celé organizace.

# 2. Technické zadání a koncepty

Tato kapitola se zaměřuje na technické aspekty projektu a představuje klíčové teoretické koncepty potřebné pro efektivní implementaci řešení. Zvláštní pozornost je věnována multiprocessingu jako nástroji pro optimalizaci zpracování velkého objemu dat (Hunt et al., 2023; „multiprocessing — Process-based parallelism“, 2025). Kapitola nejprve vysvětluje základní principy paralelního zpracování dat a následně popisuje jejich konkrétní aplikaci v kontextu analýzy logů. Podrobnější popis architektury, algoritmu a datového modelu bude následně uveden v kapitole zaměřené na implementaci řešení.

## 2.1 Teoretické základy multiprocessingu

Multiprocessing, neboli víceprocesové zpracování, představuje paradigma, které umožňuje současné provádění více výpočetních procesů za účelem maximalizace využití dostupného hardwaru a minimalizace celkového času zpracování (Hunt et al., 2023; Rauber & Rünger, 2007). Na rozdíl od tradičního sekvenčního zpracování, kdy jsou úlohy vykonávány jedna po druhé, multiprocessing umožňuje rozdělit složitou úlohu na menší, nezávislé části, které mohou být zpracovány paralelně (Dean & Ghemawat, 2008). Tento přístup je v současnosti klíčový pro dosažení vysoké výkonnosti výpočetních systémů, zvláště při zpracování velkých objemů dat, jako jsou analýzy logových záznamů.

### 2.1.1 Architektura procesoru a její vliv na výkon

Moderní procesory jsou navrženy s více výpočetními jádry, přičemž každé jádro může pracovat nezávisle na ostatních. Současné procesory běžně disponují 4 až 64 fyzickými jádry, v závislosti na určení (spotřebitelské vs. serverové řešení) (National Institute of Standards and Technology, 2006). Každé fyzické jádro může díky technologii Hyper-Threading (Intel) nebo SMT (Simultaneous Multi-Threading) simulovat dvě logická jádra, což dále zvyšuje možnosti paralelizace (Intel, 2002). Tyto technologie umožňují jednomu fyzickému jádru zpracovávat instrukce ze dvou různých vláken současně, což vede k efektivnějšímu využití výpočetních zdrojů procesoru, zejména když jedno vlákno čeká na data z paměti.

Efektivita multiprocessingu je ovlivněna zejména těmito faktory:

- **Počet jader:** přímá úměra mezi počtem jader a počtem paralelních procesů (Amdahl, 1967). Více fyzických jader umožňuje současné zpracování většího počtu nezávislých úloh. Například osmijádrový procesor může teoreticky provádět osm výpočetních operací současně. Nicméně praktický přínos dodatečných jader může být omezen Amdahlovým zákonem, který říká, že celkové zrychlení je limitováno neparalelizovatelnou částí programu (Amdahl, 1967; Hill & Marty, 2008).

- **Typ úlohy:** tzv. *embarrassingly parallel* problémy (např. nezávislé zpracování logových souborů) dosahují téměř lineárního škálování (Dean & Ghemawat, 2008). Termín *embarrassingly parallel* označuje úlohy, které lze rozdělit na zcela nezávislé podúlohy bez nutnosti komunikace mezi nimi. Název naznačuje, že paralelizace je v těchto případech téměř triviální. Příkladem je zpracování velkého množství samostatných souborů, kdy každé jádro může analyzovat jiný soubor bez potřeby koordinace s ostatními jádry (**Brown2011**; Foster, 1995).
- **Režie procesů:** tvorba a správa procesů má vlastní režií, kterou je třeba vyvážit proti přínosu paralelizace (Rauber & Rünger, 2007). Vytvoření nového procesu v operačním systému zahrnuje alokaci paměti, inicializaci kontextu procesu, vytvoření zásobníku a dalších datových struktur. Tyto operace vyžadují stovky tisíc až miliony procesorových cyklů (Ousterhout, 2018). Proto je v některých případech výhodnější použít menší počet procesů nebo implementovat systém opakovaného využití procesů, tzv. process pooling (Gorelick & Ozsvald, 2020).
- **Sdílené prostředky:** soutěž o paměť, disk či síť vede k tzv. bottlenecks a degradaci výkonu (Rauber & Rünger, 2007). Když více procesů současně přistupuje ke stejným hardwarovým zdrojům, může docházet k čekání a výkonnostním problémům. Například paměťová sběrnice může být saturována při současném čtení velkých objemů dat několika procesy, nebo může docházet k fragmentaci přístupů na disk, což vede k neefektivnímu využití Input/Output (I/O) propustnosti (Hennessy & Patterson, 2019; Tanenbaum & Bos, 2016).

Pro úplnost je nutné doplnit, co takové faktory znamenají v praxi. Významným faktorem je tedy režie spojená s vytvářením a správou procesů. Vytvoření nového procesu představuje určitou výpočetní zátěž, která musí být vyvážena přínosy paralelizace. V praxi to znamená, že není vždy optimální vytvářet maximální možný počet procesů, ale spíše najít rovnováhu mezi režií a výkonem. Pokud je úloha relativně krátká, může režie spojená s vytvořením nového procesu převážit nad výhodami paralelizace (Ousterhout, 2018). Pro řešení tohoto problému se často používá technika předem vytvořených procesů (process pool), která minimalizuje režii tím, že vytvoří procesy pouze jednou a poté je opakovaně využívá pro různé úlohy (Gorelick & Ozsvald, 2020; „multiprocessing — Process-based parallelism“, 2025).

Dalsím klíčovým faktorem je efektivita využití sdílených prostředků, kdy procesy mohou soutěžit o přístup ke sdíleným prostředkům (paměť, disk, síť), což může vést k tzv. *bottlenecks* a degradaci výkonu. Tento problém se projevuje zejména při intenzivních I/O operacích, kdy více procesů současně čte z disku nebo zapisuje na síť. V takových případech může paralelizace paradoxně vést ke zpomalení, protože procesy tráví více času čekáním na přístup ke sdíleným zdrojům než skutečným zpracováním dat (Hennessy & Patterson, 2019). Pro minimalizaci těchto problémů je důležité optimalizovat vzorce přístupu k datům, například formou sekvenčního čtení, dávkového zpracování nebo použitím vyrovnávacích pamětí (Drepper, 2007; McKinney, 2017).

V kontextu analýzy logů se multiprocessing jeví jako ideální řešení, neboť zpracování jednotlivých logů nebo skupin logů může probíhat nezávisle, s minimální potřebou komunikace

mezi procesy (Hunt et al., 2023). Tato vlastnost, známá také jako *data parallelism*, umožňuje efektivní distribuci práce mezi dostupná jádra procesoru bez složité koordinace nebo sdílení stavu (Dean & Ghemawat, 2008; McKinney, 2017). Díky tomu lze dosáhnout téměř lineárního škálování výkonu s počtem dostupných jader, což v praxi znamená, že zdvojnásobení počtu procesorů vede k přibližně dvojnásobnému zrychlení zpracování (Gorelick & Ozsvald, 2020; Hunt et al., 2023).

### 2.1.2 Paralelní zpracování dat v Pythonu

Python jako programovací jazyk nabízí několik modulů pro implementaci multiprocessingu, které poskytují různé úrovně abstrakce a kontroly nad paralelními procesy. Nejdůležitějšími jsou `multiprocessing`, `concurrent.futures` a `joblib` („multiprocessing — Process-based parallelism“, 2025). V rámci této práce byl využit především modul `multiprocessing`, který poskytuje nejnižší úroveň kontroly nad paralelními procesy a umožňuje detailní optimalizaci pro specifické potřeby projektu.

Modul `multiprocessing` je, jak již bylo zmíněno, součástí standardní knihovny Pythonu („multiprocessing — Process-based parallelism“, 2025) a nabízí robustní implementaci paralelního zpracování založenou na vytváření samostatných procesů namísto vláken, což umožňuje obejít omezení způsobená tzv. Global Interpreter Lock (GIL). Mezi klíčové třídy a funkce tohoto modulu patří:

- `Pool` – Implementuje skupinu pracovních procesů, mezi které jsou distribuovány úlohy. Metoda `map` této třídy funguje analogicky ke standardní funkci `map` v Pythonu, ale provádí mapování paralelně na více procesech, což umožňuje efektivní zpracování velkých kolekcí dat.
- `Process` – Představuje samostatný proces, který může být spuštěn a spravován nezávisle. Na rozdíl od třídy `Pool` poskytuje větší kontrolu nad životním cyklem procesu, což je užitečné pro složitější scénáře paralelizace.
- `Queue` a `Pipe` – Tyto třídy implementují mechanismy pro komunikaci mezi procesy a sdílení dat. `Queue` poskytuje frontu podle principu First In, First Out (FIFO), zatímco `Pipe` implementuje obousměrný komunikační kanál.
- `Lock`, `Semaphore` a `Event` – Synchronizační primitivy pro koordinaci přístupu ke sdíleným prostředkům, které jsou nezbytné pro prevenci race conditions a zajištění konzistence dat.

Konkrétní implementace paralelního zpracování logů v tomto projektu využívá třídu `Pool` pro efektivní distribuci zpracování individuálních logových záznamů mezi více procesů. Následující ukázka ilustruje typické použití této třídy pro paralelní zpracování kolekce dat:

```
from multiprocessing import Pool

def process_log(log_entry):
    # processing jednoho logu
```

```
    return analyzed_result

if __name__ == '__main__':
    with Pool(processes=12) as pool:
        results = pool.map(process_log, log_entries)
```

Tento přístup byl aplikován v implementaci řešení, což vedlo k výraznému zrychlení oproti sekvenčnímu zpracování. Experimenty v rámci implementace projektu ukázaly, že doba zpracování se snížila z původních 12 hodin na přibližně 15 minut pro analýzu logů za jeden měsíc. Takové zrychlení je zásadní pro praktické využití systému, neboť umožňuje analýzu logů v časových rámcích, které jsou akceptovatelné pro běžnou operativní činnost.

### 2.1.3 Výzvy a omezení paralelního zpracování

Přes nesporné výhody přináší paralelní zpracování i řadu výzev a omezení, které je nutné při návrhu systému zohlednit. Tyto výzvy lze shrnout do několika klíčových oblastí:

Amdahlův zákon popisuje teoretický limit zrychlení (Amdahl, 1967), kterého lze dosáhnout paralelizací algoritmu. Tento zákon vyjadřuje skutečnost, že pokud pouze část algoritmu může být paralelizována, celkové zrychlení je omezeno sekvenční částí. Matematicky lze tento vztah vyjádřit jako:

$$S(n) = \frac{1}{(1-p) + \frac{p}{n}}$$

kde $S(n)$ je zrychlení při použití $n$ procesů, $p$ je část algoritmu, která může být paralelizována. Z tohoto vztahu vyplývá, že i při nekonečném počtu procesorů ($n \to \infty$) je celkové zrychlení omezeno hodnotou $\frac{1}{1-p}$. V praxi to znamená, že je třeba věnovat pozornost i optimalizaci sekvenčních částí algoritmu.

Další významnou výzvou je režie spojená s komunikací mezi procesy. V případech, kdy jednotlivé procesy musí vzájemně komunikovat nebo sdílet data, může režie spojená s touto komunikací výrazně snížit celkový výkon (Rauber & Rünger, 2007). Tento problém je zvláště významný v distribuovaných systémech, ale projevuje se i při paralelním zpracování na jednom stroji. Pro dosažení optimálního výkonu je proto důležité minimalizovat meziprocesovou komunikaci a upřednostňovat architektury, kde procesy mohou pracovat nezávisle.

Paralelní zpracování je také náročné na paměťové zdroje. Každý samostatný proces vyžaduje vlastní paměťový prostor, což může vést k vysoké spotřebě paměti při velkém počtu procesů. V prostředí s omezenými paměťovými zdroji je proto nutné nalézt rovnováhu mezi stupněm paralelizace a paměťovou náročností. V rámci projektu bylo experimentálně stanoveno, že optimální počet paralelních procesů se pohybuje mezi 8–16, v závislosti na konkrétní konfiguraci hardwaru.

Neopomenutelnou výzvou je také složitost vývoje a ladění paralelních aplikací. Tyto aplikace jsou obecně složitější na návrh, implementaci a především na ladění, neboť mohou trpět těžko odhalitelnými chybami, jako jsou uváznutí (deadlocks), souběh (race conditions) nebo hladovění (starvation). Pro minimalizaci těchto problémů byly v projektu použity vysokoúrovňové abstrakce jako `Pool`, které zapouzdřují složitost paralelního zpracování a poskytují robustní rozhraní.

V kontextu projektu byla největší výzvou paměťová náročnost při zpracování velkého objemu logů. Bylo nutné nalézt optimální rovnováhu mezi počtem paralelních procesů a celkovou spotřebou systémových prostředků. Tato optimalizace byla provedena experimentálně, s ohledem na konkrétní charakteristiky dat a dostupný hardware.

## 2.2 Architektura a koncepce řešení

Na základě analýzy požadavků a technických možností byla navržena architektura řešení, která kombinuje efektivitu multiprocessingu s požadavky na škálovatelnost a udržitelnost („Centrum informatiky – Základní informace“, 2025; Rauber & Rünger, 2007). Architektura byla navržena s důrazem na modularitu a možnost budoucího rozšíření, což umožňuje postupnou integraci dalších zdrojů logů a analytických funkcí.

### 2.2.1 Serverová infrastruktura VŠE

Vysoká škola ekonomická disponuje rozsáhlou serverovou infrastrukturou, která podporuje široké spektrum služeb od výukových systémů až po administrativní aplikace. Klíčovou součástí této infrastruktury pro účely projektu je centrální log server *Levandule* (levandule.vse.cz). Tento server slouží jako primární úložiště logů z více než 2 200 zdrojů, zahrnujících aplikační servery, webové servery, síťová zařízení a další komponenty IT infrastruktury školy.

Server Levandule je konfigurován jako dedikovaný systém pro sběr a ukládání logů, což je v souladu s doporučenými postupy pro správu logů v rozsáhlých IT prostředích (National Institute of Standards and Technology, 2006). Centralizace logů na jednom serveru nabízí několik výhod, včetně jednotného přístupového bodu pro analýzu, standardizovaného systému zálohování a možnosti implementace komplexních bezpečnostních politik.

#### Struktura log serveru Levandule

Server Levandule implementuje hierarchickou adresářovou strukturu, která je organizována podle logických kritérií, což usnadňuje správu a analýzu logů (National Institute of Standards and Technology, 2006). Základní struktura je následující:

- **/var/log/HOSTS/** – Kořenový adresář pro veškeré logy

- **[název zdroje]/** – Podadresáře pojmenované podle zdrojů logů (např. zdroje.vse.cz, tarantule.vse.cz) dle doporučení v rámci Linux systému (The Linux Foundation, 2004).
- **[rok]/[měsíc]/** – Hierarchické členění podle časových období, umožňující efektivní navigaci v historických datech.
- **[den]** nebo **[den].xz** – Soubory s logy za jednotlivé dny, přičemž starší soubory jsou komprimované pomocí formátu XZ pro úsporu diskového prostoru (Friedl, 2006)

Tato struktura, ačkoliv logická a přehledná z hlediska organizace dat, představuje významnou výzvu pro efektivní zpracování logů (National Institute of Standards and Technology, 2006). Celkový počet souborů, které je třeba analyzovat, může dosahovat stovek tisíc, což vyžaduje sofistikované metody pro efektivní přístup a zpracování. Pro ilustraci lze odhadnout celkový počet souborů s logy za jeden rok:

$$2\,200 \text{ zdrojů} \times 12 \text{ měsíců} \times 30 \text{ dní} \approx 792\,000 \text{ souborů} \tag{2.1}$$

Takový objem dat není možné efektivně zpracovat tradičními sekvenčními metodami, což zdůrazňuje potřebu využití paralelního zpracování a optimalizovaných algoritmů. Navíc, komprimace starších souborů, ačkoliv nezbytná z hlediska správy diskového prostoru, přidává další vrstvu složitosti, neboť vyžaduje implementaci mechanismů pro efektivní dekompresi během analýzy (Friedl, 2006).

### 2.2.2 Získávání dat pomocí SSH připojení

Přístup k datům uloženým na serveru Levandule je realizován prostřednictvím zabezpečeného protokolu Secure Shell (SSH), který poskytuje šifrovaný komunikační kanál v potenciálně nezabezpečených sítích (Kolouch et al., 2019; Stallings, 2019). Toto řešení bylo zvoleno s ohledem na bezpečnostní požadavky, neboť logové záznamy mohou obsahovat citlivé informace včetně autentizačních údajů, internet protocol (IP) adres a detailů o síťové infrastruktuře.

Implementace SSH připojení zahrnuje několik klíčových komponent. Primárním mechanismem autentizace je pár veřejného a soukromého klíče, což eliminuje potřebu přenosu hesla v čitelné podobě a poskytuje vyšší úroveň zabezpečení než tradiční autentizace heslem (Stallings, 2019). Po úspěšné autentizaci je vytvořen zabezpečený tunel, který zajišťuje šifrování veškeré komunikace mezi klientem a serverem, včetně přenosu logových souborů. Pro zajištění robustnosti řešení byly implementovány také mechanismy pro automatické zotavení z výpadků připojení, což je důležité pro dlouhodobé operace, jako je analýza velkého objemu historických logů („Paramiko — Python SSH library“, 2024).

Pro implementaci SSH připojení byla zvolena knihovna `paramiko`, která poskytuje kompletní implementaci SSH protokolu v Pythonu („Paramiko — Python SSH library“, 2024). Tato knihovna byla vybrána na základě jejích výkonnostních charakteristik, stability a rozsáhlé dokumentace. `Paramiko` umožňuje nejen základní operace jako připojení k serveru a spouštění příkazů, ale také pokročilé funkce jako například přenos souborů pomocí SSH File Transfer

Protocol (SFTP), což je využíváno pro efektivní stahování logových souborů pro lokální analýzu.

### 2.2.3 Struktura a organizace zdrojového kódu

Zdrojový kód řešení je organizován podle principů modularity a znovupoužitelnosti, což usnadňuje údržbu, testování a budoucí rozšíření systému (Fowler, 2002; Martin, 2017; McKinney, 2017). Architektura byla navržena s důrazem na oddělení jednotlivých funkčních celků, což umožňuje nezávislý vývoj a testování jednotlivých komponent (McKinney, 2017).

Základní struktura zdrojového kódu je rozdělena do několika logických modulů, které odrážejí hlavní funkční oblasti systému:

1. **Modul pro připojení k serveru** – Tento modul zapouzdřuje veškerou funkcionalitu spojenou s navázáním a správou SSH připojení k serveru Levandule. Zahrnuje implementaci autentizace, správu připojení a mechanismy pro zotavení z chyb. Díky tomuto oddělení je možné v budoucnu změnit způsob připojení (například na jiný protokol) bez nutnosti úprav v ostatních částech systému.
2. **Modul pro získávání dat** – Implementuje funkce pro efektivní procházení adresářové struktury serveru a získávání logů. Tento modul využívá znalost hierarchické struktury logů na serveru Levandule a poskytuje abstraktní rozhraní pro přístup k logovým souborům bez ohledu na jejich fyzické umístění nebo formát.
3. **Modul pro zpracování logů** – Obsahuje implementaci Extended Nagappan-Vouk algoritmu pro analýzu logů a identifikaci statických a proměnných částí. Tento modul představuje jádro analytické funkcionality systému a implementuje sofistikované algoritmy pro frekvenční analýzu a klasifikaci tokenů.
4. **Modul pro multiprocessing** – Poskytuje nástroje pro paralelní zpracování dat s využitím všech dostupných procesorových jader. Tento modul zapouzdřuje složitost paralelního zpracování a poskytuje jednotné rozhraní pro distribuci úloh mezi dostupné procesory.
5. **Modul pro export dat** – Zajišťuje ukládání analyzovaných dat ve sloupcově orientovaném formátu Apache Parquet s horizontálním rozdělením, což přináší zrychlení načítání dat oproti tradičním formátům („Apache Parquet Documentation“, 2025). Implementovaná strategie partitioningu podle zdroje a časového období umožňuje efektivní filtrování dat a minimalizuje množství zpracovávaných dat při dotazování. Modul také poskytuje funkce pro export do jiných formátů pro účely interoperability s externími nástroji (CSV, JSON).

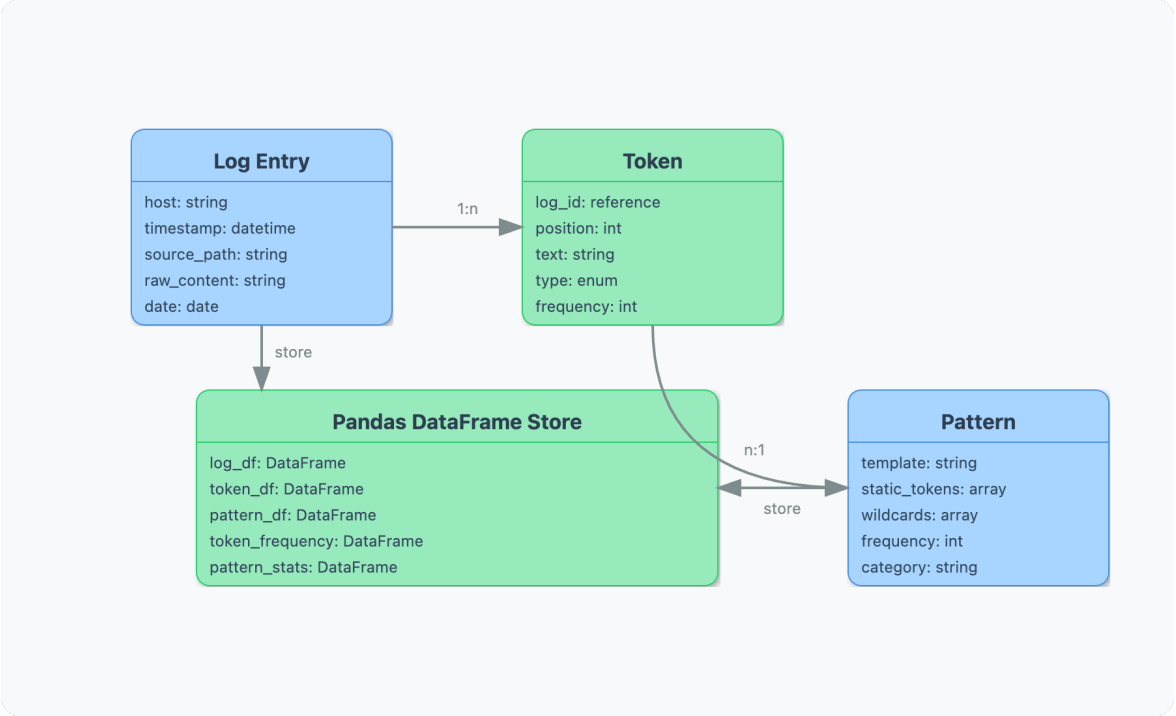
Tato modulární struktura přináší několik výhod. Především usnadňuje údržbu kódu, neboť změny v jednom modulu mají minimální dopad na ostatní části systému. Dále umožňuje paralelní vývoj jednotlivých komponent, což je důležité při práci v týmu. V neposlední řadě podporuje postupné rozšiřování funkcionality systému bez nutnosti rozsáhlých změn v existujícím kódu.

### 2.2.4 Datový model

Pro efektivní reprezentaci logů a výsledků jejich analýzy byl navržen specifický datový model, který reflektuje hierarchickou strukturu logových dat a zároveň podporuje efektivní analytické operace. Tento model je implementován s využitím DataFrame struktur knihovny pandas, což umožňuje využití jejích pokročilých funkcí pro manipulaci a analýzu dat (McKinney, 2017).

Datový model byl navržen s ohledem na výkonnostní charakteristiky a flexibilitu, což jsou klíčové požadavky při zpracování velkého objemu logových dat. Využití struktur knihovny pandas přináší několik výhod, včetně efektivní implementace operací nad velkými datovými sadami, zabudované podpory pro chybějící hodnoty a možnosti využití optimalizovaných funkcí pro agregaci a transformaci dat („pandas — Python Data Analysis Library“, 2025).

**Obrázek 2.1**
*Hierarchický datový model pro analýzu logů*




*Poznámka: Vlastní zpracování.*

Klíčové entity datového modelu zahrnují:

- **Log Entry** – Základní entita reprezentující jeden záznam v logu. Obsahuje nejen samotný text logu, ale také metadata jako je zdroj, datum a čas. Tato struktura umožňuje snadnou filtraci a agregaci logů podle různých kritérií (McKinney, 2017).
- **Token** – Reprezentuje jednotlivé slovo nebo symbol v logu, včetně informací o jeho pozici, typu (delimeter, statický token, proměnná část) a frekvenci výskytu. Tato granulární reprezentace umožňuje detailní analýzu struktury logů a identifikaci vzorů (Nagappan & Vouk, 2010).
- **Pattern** – Představuje rozpoznaný vzor v logu, kde některé části jsou konstantní (to-

keny) a jiné proměnné (wildcards). Tato abstrakce umožňuje shlukování podobných logů a identifikaci anomálií, což je klíčové pro detekci potenciálních bezpečnostních incidentů (Huo et al., 2023).

Tento datový model poskytuje flexibilní základ pro implementaci různých analytických algoritmů a umožňuje efektivní reprezentaci výsledků pro další zpracování a vizualizaci. Jeho hierarchická struktura odráží přirozené vztahy mezi různými úrovněmi abstrakce logových dat a zároveň podporuje efektivní implementaci algoritmu Extended Nagappan-Vouk („pandas — Python Data Analysis Library“, 2025).

## 2.3 Extended Nagappan-Vouk algoritmus

Pro identifikaci statických a proměnných částí logů byl implementován ENV algoritmus, který byl vyvinut, respektive rozšířen dr. Danielem Tovarňákem z Masarykovy univerzity v Brně (He et al., 2017; Nagappan & Vouk, 2010; Tovarnák, 2017). Tento algoritmus představuje významné rozšíření původního Nagappan-Vouk algoritmu a poskytuje sofistikovanější přístup k frekvenční analýze slov ve větách, což je klíčové pro efektivní zpracování logových dat (David Akande et al., 2022).

ENV algoritmus byl zvolen na základě jeho prokázané efektivity při zpracování reálných logových dat a schopnosti adaptace na různé formáty logů bez nutnosti manuální konfigurace (Huo et al., 2023; Jiang et al., 2024). Tato vlastnost je zvláště důležitá v heterogenním prostředí VŠE, kde se vyskytuje široké spektrum zdrojů logů s různými formáty a strukturami.

### 2.3.1 Princip algoritmu

ENV algoritmus vychází z předpokladu, že logové záznamy mají určitou strukturu, která se skládá z konstantních a proměnných částí. Na základě frekvenční analýzy lze v logových záznamech identifikovat dvě základní kategorie prvků (Nagappan & Vouk, 2010; Schmidt et al., 2012):

1. **Tokeny** – Statické části, které se v logových záznamech stejného typu opakují na stejných pozicích. Typicky se jedná o klíčová slova a fráze, které popisují typ události, jako například "Connection", "error", "failed"apod. Tyto tokeny poskytují kontextovou informaci o povaze zaznamenané události.
2. **Wildcards** – Proměnné části, které se na stejných pozicích v různých instancích stejného typu logu mění. Typicky se jedná o konkrétní hodnoty jako jsou IP adresy, ID procesů, časové značky apod. Tyto proměnné části nesou specifické informace o konkrétní instanci události.

Algoritmus využívá sofistikovanou frekvenční analýzu k rozlišení těchto dvou kategorií. Princip je založen na pozorování, že statické části (tokeny) se vyskytují s vysokou frekvencí na

stejných pozicích v logových záznamech stejného typu, zatímco proměnné části (wildcards) vykazují vysokou variabilitu. Konkrétně, pokud se určité slovo vyskytuje na konkrétní pozici v logu s vysokou frekvencí, je klasifikováno jako token. Naopak, pokud se na dané pozici vyskytuje mnoho různých slov s nízkou frekvencí, je tato pozice označena jako wildcard (Nagappan & Vouk, 2010).

### 2.3.2 Matematický model

Formální matematický model ENV algoritmu poskytuje rigorózní základ pro implementaci a umožňuje precizní konfiguraci parametrů algoritmu pro dosažení optimálních výsledků. Model je založen na analýze kardinality a frekvence výskytu tokenů na jednotlivých pozicích v logových záznamech (Nagappan & Vouk, 2010).

Pro každou pozici $p$ v logových záznamech daného typu se vypočítá tzv. kardinalita $C_p$, která představuje počet různých hodnot, které se na této pozici vyskytují. Tato metrika poskytuje informaci o variabilitě hodnot na dané pozici. Dále se vypočítá celková frekvence $F_p$ jako součet frekvencí všech hodnot na této pozici, což reflektuje celkový počet logových záznamů, které mají token na této pozici.

Na základě těchto dvou metrik je pozice $p$ klasifikována jako wildcard (proměnná část), pokud platí:

$$\frac{C_p}{F_p} > \theta$$

kde $\theta$ je prahová hodnota, která určuje citlivost algoritmu na variabilitu hodnot. Tato hodnota může být určena experimentálně nebo odvozena z charakteristik dat. Nižší hodnoty $\theta$ vedou k agresivnější klasifikaci pozic jako wildcards, zatímco vyšší hodnoty upřednostňují klasifikaci jako tokeny (David Akande et al., 2022).

Tento přístup poskytuje matematicky rigorózní základ pro identifikaci struktury logových záznamů a umožňuje adaptaci algoritmu na různé typy logů bez nutnosti manuální konfigurace pro každý typ.

### 2.3.3 Implementační detaily

Implementace ENV algoritmu byla provedena s důrazem na výkonnost a škálovatelnost, což jsou klíčové požadavky při zpracování velkého objemu logových dat (He et al., 2017; Schmidt et al., 2012). Pro zvýšení efektivity zpracování bylo provedeno několik iterací optimalizace, které se zaměřily na minimalizaci výpočetní a paměťové náročnosti algoritmu.

Prvním krokem v procesu analýzy je preprocessing logových záznamů. V této fázi jsou logy rozděleny na jednotlivá slova pomocí sofistikovaných regulárních výrazů, což umožňuje efek-

tivní tokenizaci i složitějších struktur (Nagappan & Vouk, 2010). Tato operace je kritická pro následnou frekvenční analýzu, neboť definuje základní jednotky, nad kterými bude analýza prováděna. Implementace využívá optimalizované regulární výrazy navržené pro vysokou propustnost dat, minimalizaci backtracking-u a přesnou tokenizaci různých formátů logů (Schmidt et al., 2012).

Pro dosažení optimálního výkonu při zpracování velkého objemu dat bylo implementováno paralelní zpracování s využitím modulu `multiprocessing` (Hunt et al., 2023; Kolouch et al., 2019). Analýza frekvenčních charakteristik probíhá paralelně na více procesech, což výrazně zkracuje dobu zpracování ve srovnání se sekvenčním přístupem. Tato optimalizace je zvláště efektivní při zpracování logů z mnoha zdrojů, které mohou být analyzovány nezávisle (Hunt et al., 2023).

Dalším klíčovým aspektem implementace je inkrementální aktualizace statistik při zpracování nových dat. Místo kompletního přepočítání všech statistik při každém běhu algoritmu jsou statistiky aktualizovány průběžně, což umožňuje efektivní zpracování i velmi velkých datasetů, které by se jako celek nevešly do paměti (David Akande et al., 2022; Jiang et al., 2024). Tento přístup je zvláště důležitý při průběžné analýze nově příchozích logů, kdy není praktické opakovaně zpracovávat celý historický dataset.

Výsledkem aplikace ENV algoritmu je převod původních logových záznamů na strukturované vzory, kde jsou jasně identifikovány statické a proměnné části (Tovarnák, 2017). Tyto vzory poskytují abstraktní reprezentaci logů, která je mnohem kompaktnější než původní záznamy, ale zachovává veškerou relevantní informaci (Huo et al., 2023). Tato transformace umožňuje efektivní indexaci, vyhledávání a analýzu logových dat, což je klíčové pro identifikaci anomálií a potenciálních bezpečnostních incidentů (He et al., 2017).

## 2.4 Vizuální programování v kontextu analýzy logů

Součástí technického zadání bylo také prozkoumání možností vizuálního programování jako alternativního přístupu k analýze logů. Vizuální programování představuje paradigma, které umožňuje definovat analytické workflow pomocí grafického rozhraní namísto tradičního psaní kódu (Bockermann, 2014). Tento přístup může být přínosný zejména pro uživatele, kteří nejsou primárně programátory, ale potřebují provádět komplexní analýzy logových dat.

Průzkum možností vizuálního programování byl motivován snahou o zpřístupnění pokročilých analytických funkcí širšímu okruhu uživatelů, což je důležité pro dlouhodobou udržitelnost a rozšiřování systému. Vizuální programování nabízí intuitivnější přístup k definici analytických workflow a potenciálně může snížit bariéry pro zapojení doménových expertů do procesu analýzy logů.

### 2.4.1 KNIME jako nástroj pro analýzu logů

Pro experimentální implementaci byl zvolen nástroj Konstanz Information Miner (KNIME), který představuje otevřenou platformu pro datovou analytiku, strojové učení a reportování (Berthold et al., 2010). KNIME je založen na koncepci workflow, kde jsou jednotlivé analytické kroky reprezentovány uzly, které jsou propojeny do grafu, což poskytuje intuitivní vizuální reprezentaci celého procesu („KNIME Documentation“, 2024).

KNIME byl vybrán na základě několika kritérií, včetně jeho široké podpory v komunitě datových analytiků, rozsáhlé sady předpřipravených komponent a možnosti integrace s Pythonem, což umožňuje využít existující kód pro analýzu logů (Berthold et al., 2010). Platforma nabízí také rozsáhlou dokumentaci a aktivní komunitu, což usnadňuje její adopci a řešení případných problémů („KNIME Documentation“, 2024).

Výhody použití KNIME pro analýzu logů jsou mnohočetné. Především jde o intuitivní vizuální rozhraní, které umožňuje rychlou iteraci a experimentování s různými analytickými přístupy (Bockermann, 2014). Workflow jsou tvořeny přetahováním a propojováním uzlů, což eliminuje potřebu psaní kódu a umožňuje i netechnickým uživatelům navrhovat komplexní analytické procesy (Söderström & Moradian, 2013).

Další významnou výhodou je bohatá sada předpřipravených komponent, které pokrývají široké spektrum funkcí od základní manipulace s daty až po pokročilé metody strojového učení (Berthold et al., 2010). KNIME obsahuje stovky uzlů pro různé operace, což eliminuje potřebu implementace běžných funkcí a umožňuje soustředit se na specifické aspekty analýzy logů („KNIME Documentation“, 2024).

KNIME také poskytuje možnost integrace vlastního kódu v jazycích jako Python, R nebo SQL (Berthold et al., 2010), což je klíčové při implementaci specializovaných funkcí, které nejsou pokryty standardními uzly. V kontextu analýzy logů to umožňuje integraci existujícího kódu pro implementaci algoritmu Extended Nagappan-Vouk nebo jiných specifických analytických metod.

V neposlední řadě KNIME nabízí dobrou podporu pro práci s velkými objemy dat díky svému optimalizovanému zpracování a možnosti napojení na různé datové zdroje (Berthold et al., 2010; „KNIME Documentation“, 2024). Platforma implementuje efektivní mechanismy pro práci s daty, které nemohou být celé načteny do paměti, což je důležité při zpracování rozsáhlých logových archivů (Karlsen et al., 2024).

### 2.4.2 Limity vizuálního programování

Přes nesporné výhody má vizuální programování i řadu limitů, které se projevily během experimentální implementace analýzy logů v KNIME. Tyto limity je důležité zohlednit při rozhodování o vhodném přístupu k implementaci systému pro analýzu logů.

Prvním významným limitem jsou omezení výkonnosti. KNIME vykazoval výrazně nižší výkon při zpracování velkého objemu logů v porovnání s nativním řešením v Pythonu. Provedené experimenty ukázaly, že KNIME byl schopen zpracovat přibližně 500 logů za 2 minuty, což je řádově pomalejší než optimalizované řešení v Pythonu. Při extrapolaci na celý objem dat (stovky tisíc souborů) by doba zpracování v KNIME dosáhla několika dní, což je pro praktické využití nepřijatelné.

Dalším limitem je chybějící flexibilita při implementaci složitějších algoritmů. I přes možnost integrace vlastního kódu vyžadovalo řešení v KNIME vytváření mnoha mezikroků a pomocných konstrukcí, které by v běžném programovacím jazyku byly řešeny několika řádky kódu. Tato komplexita vizuálního workflow může paradoxně snížit srozumitelnost a udržitelnost řešení, zvláště při implementaci sofistikovaných analytických algoritmů (Bockermann, 2014).

Problematická byla také práce s externími systémy, konkrétně připojení k serveru Levandule pomocí SSH a zpracování zazipovaných souborů. Tyto operace vyžadovaly komplikovanou konfiguraci a v některých případech nebylo možné je v KNIME efektivně implementovat . Integrace s externími systémy je obecně oblasti, kde vizuální programování často naráží na své limity a vyžaduje doplnění nativním kódem.

I přes uvedené limity má vizuální programování své místo v analýze logů, zejména v určitých specifických scénářích. KNIME se ukázal jako vhodný nástroj pro rychlé prototypování a explorativní analýzu menších datasetů, kde jeho intuitivní rozhraní a předpřipravené komponenty umožňují rychlou iteraci a experimentování (Berthold et al., 2010). Pro produkční nasazení a zpracování velkých objemů dat je však efektivnější využít specializované řešení implementované v nativním programovacím jazyce (He et al., 2017).

## 2.5 Porovnání nástrojů pro zpracování logů

Efektivní zpracování a analýza logů vyžaduje vhodné nástroje, které odpovídají specifickým požadavkům dané organizace a charakteristikám zpracovávaných dat. V rámci projektu byly porovnány tři klíčové přístupy: specializované nástroje pro správu logů (LOGmanager), nástroje pro vizuální programování (KNIME) a nativní programování v Pythonu. Toto porovnání poskytlo důležité podklady pro výběr vhodného přístupu k implementaci řešení pro analýzu logů na VŠE.

### 2.5.1 LOGmanager

LOGmanager představuje specializované řešení pro centralizovaný sběr, ukládání a analýzu logů, které je již nasazeno v prostředí VŠE. Tento systém byl navržen specificky pro účely log managementu a nabízí řadu funkcí optimalizovaných pro práci s logovými daty („LOGmanager Documentation“, 2025).

Mezi hlavní přednosti LOGmanageru patří intuitivní uživatelské rozhraní s předpřipravenými dashboardy, které poskytují přehlednou vizualizaci klíčových metrik a událostí. Toto rozhraní umožňuje i méně technickým uživatelům získat rychlý přehled o stavu systému a potenciálních bezpečnostních incidentech. LOGmanager dále nabízí možnost monitorování v reálném čase, což je klíčové pro rychlou detekci a reakci na bezpečnostní události. Systém pracuje s aktuálními daty a umožňuje nastavení alertů pro kritické události.

Důležitou funkcionalitou jsou také integrované parsovací mechanismy pro běžné formáty logů, které automaticky extrahují relevantní informace z logových záznamů a strukturují je pro další analýzu. Tato funkce eliminuje potřebu manuální konfigurace parsování pro standardní typy logů. Pro integraci s externími systémy poskytuje LOGmanager Representational State Transfer Application Programming Interface (REST API), které umožňuje programový přístup k datům a funkcím systému. Toto API je důležité pro automatizaci analytických procesů a integraci s dalšími bezpečnostními nástroji.

Přes tyto výhody má LOGmanager v prostředí VŠE několik významných omezení. Nejvýznamnějším je omezené pokrytí dostupných zdrojů – systém zpracovává pouze zlomek (přibližně 30 z celkových 2 200) všech zdrojů logů v infrastruktuře VŠE. Toto omezení výrazně snižuje schopnost detekce komplexních bezpečnostních incidentů, které mohou zahrnovat aktivity na různých systémech. Dalším limitem je omezený přístup k historickým datům, kdy API poskytuje přístup pouze k datům za poslední 3 dny. Toto omezení komplikuje analýzu dlouhodobých trendů a vyšetřování historických incidentů. LOGmanager také neposkytuje dostatečné nástroje pro pokročilou sémantickou analýzu logů, jako je identifikace statických a proměnných částí logů, což je klíčové pro efektivní agregaci a klasifikaci logových záznamů.

### 2.5.2 KNIME

KNIME (Konstanz Information Miner) představuje alternativní přístup k analýze logů, který je založen na principech vizuálního programování. Tato platforma není primárně určena pro log management, ale poskytuje flexibilní prostředí pro implementaci různých analytických workflow („KNIME Analytics Platform Documentation“, 2023).

Hlavní výhodou KNIME je intuitivní vizuální programování, které nevyžaduje hluboké znalosti programování. Uživatelé mohou definovat analytické procesy pomocí grafického rozhraní, což zpřístupňuje pokročilé analytické funkce i méně technicky orientovaným uživatelům. Platforma poskytuje rozsáhlou knihovnu připravených analytických komponent, které pokrývají široké spektrum funkcí od základní manipulace s daty až po pokročilé metody strojového učení. Tyto komponenty lze kombinovat do komplexních workflow bez nutnosti psaní kódu („KNIME Analytics Platform Documentation“, 2023).

KNIME dále nabízí transparentní workflow se snadnou dokumentací procesu. Vizuální reprezentace analytického procesu poskytuje přirozenou dokumentaci a usnadňuje komunikaci mezi různými zainteresovanými stranami. Pro pokročilé uživatele nabízí KNIME možnost integrace vlastních skriptů v Pythonu, R nebo SQL, což rozšiřuje funkcionalitu platformy o

specializované analytické metody („KNIME Analytics Platform Documentation“, 2023).

Přes tyto výhody má KNIME několik významných nevýhod pro analýzu logů v měřítku VŠE. Nejvýznamnějším limitem je výrazně nižší výkon při zpracování velkých objemů dat. Testy ukázaly, že zpracování 500 logů trvá přibližně 2 minuty, což by při extrapolaci na celý objem dat znamenalo nepřijatelně dlouhou dobu zpracování (Karlsen et al., 2024). KNIME také nabízí omezené možnosti automatizace a vzdáleného přístupu k datům, což komplikuje implementaci plně automatizovaného řešení pro průběžnou analýzu logů. Dalším problémem je potřeba velkého množství mezikroků pro operace, které by v nativním kódu byly triviální, což vede k složitým a těžko udržitelným workflow při implementaci komplexních analytických algoritmů.

### 2.5.3 Nativní programování v Pythonu

Třetí analyzovanou alternativou je vlastní řešení implementované v programovacím jazyce Python, které nabízí maximální flexibilitu a kontrolu nad celým procesem zpracování a analýzy logů („multiprocessing — Process-based parallelism“, 2025).

Hlavní výhodou tohoto přístupu je maximální flexibilita a kontrola nad zpracováním dat. Vývojáři mají plnou kontrolu nad každým aspektem implementace, což umožňuje optimalizaci pro specifické požadavky a charakteristiky dat. Python nabízí výrazně vyšší výkon díky možnosti optimalizace a paralelního zpracování. Testy ukázaly, že optimalizovaná implementace v Pythonu zpracuje 500 logů za méně než sekundu, což představuje řádové zrychlení oproti KNIME.

Python dále poskytuje snadnou integraci s různými datovými zdroji a formáty, což je důležité v heterogenním prostředí VŠE. Díky bohaté ekosystémě knihoven lze implementovat přístup k různým typům úložišť a zpracování různých formátů logů. Pro pokročilou analýzu nabízí Python možnost implementace sofistikovaných analytických algoritmů, včetně algoritmu Extended Nagappan–Vouk, což je klíčové pro identifikaci struktury logů a detekci anomálií (Huo et al., 2023; Nagappan & Vouk, 2010).

I tento přístup má však své nevýhody. Především jde o vyšší nároky na programátorské dovednosti, kdy vývoj a údržba řešení vyžaduje znalost Pythonu a relevantních knihoven. Dalším limitem je potřeba vlastní implementace uživatelského rozhraní, což může být časově náročné a vyžaduje specifické dovednosti v oblasti frontend vývoje. V neposlední řadě je nevýhodou delší doba vývoje pro dosažení plné funkčnosti, neboť veškerá funkcionalita musí být implementována od základů, na rozdíl od použití existujících nástrojů.

### 2.5.4 Kvantitativní porovnání výkonnosti

Pro objektivní porovnání jednotlivých přístupů byly provedeny výkonnostní testy zaměřené na zpracování stejného datasetu logů. Tyto testy poskytují kvantitativní data pro informované

rozhodnutí o vhodnosti různých přístupů pro analýzu logů v podmínkách VŠE.

**Tabulka 2.1**
*Porovnání výkonnosti různých přístupů ke zpracování logů*

| **Metrika** | **LOGmanager** | **KNIME** | **Python (optimalizovaný)** |
|---|---|---|---|
| Zpracování 500 logů | N/A | 120 sekund | $<$ 1 sekunda |
| Zpracování logů za 1 den | Omezeno na vybrané zdroje | $>$ 12 hodin | 15-20 minut |
| Škálovatelnost | Omezená | Omezená | Vysoká |
| Využití paměti | Nízké | Vysoké | Střední (optimalizované) |

*Poznámka: Vlastní zpracování.*

Z kvantitativního porovnání jednoznačně vyplývá, že pro zpracování velkého objemu logů v podmínkách VŠE je nativní implementace v Pythonu s využitím multiprocessingu jediným řešením, které splňuje požadavky na výkon a škálovatelnost. Zatímco KNIME může být užitečný pro explorativní analýzu menších datasetů a LOGmanager poskytuje cenné funkce pro monitoring v reálném čase, pouze vlastní řešení v Pythonu umožňuje efektivní zpracování a analýzu celého objemu logů v akceptovatelném časovém rámci.

## 2.6 Shrnutí technického zadání

Na základě analýzy byznysových požadavků a technických možností bylo formulováno technické zadání, které definuje klíčové komponenty a funkce řešení pro analýzu logů. Toto zadání reflektuje potřeby Centra informatiky VŠE a zohledňuje specifika prostředí, ve kterém bude řešení nasazeno.

Primárním požadavkem je implementace efektivního připojení k serveru Levandule. Řešení musí využívat SSH a SFTP protokoly pro bezpečný přístup k logovým souborům a implementovat mechanismy pro spolehlivé zpracování těchto dat. Tento aspekt je klíčový pro zajištění přístupu k primárnímu zdroji dat pro analýzu.

Dalším zásadním požadavkem je vývoj algoritmu pro analýzu logů, konkrétně implementace Extended Nagappan-Vouk algoritmu pro identifikaci statických a proměnných částí logů. Tento algoritmus umožňuje efektivní kategorizaci a agregaci logových záznamů, což je nezbytné pro identifikaci vzorů a anomálií.

Pro dosažení akceptovatelného výkonu je nezbytná optimalizace s využitím multiprocessingu. Paralelní zpracování umožňuje zkrácení doby analýzy z původních hodin na minuty, což je zásadní pro praktické využití systému v každodenní operativě Centra informatiky.

Důležitým aspektem je také návrh a implementace flexibilního datového modelu pro reprezentaci logů a výsledků jejich analýzy. Tento model musí podporovat efektivní manipulaci s daty a poskytovat základ pro implementaci analytických funkcí nad strukturovanými logovými záznamy.

V neposlední řadě je třeba zajistit přípravu dat pro vizualizaci, včetně exportu analyzovaných dat ve formátu vhodném pro další zpracování a vizualizaci. Tento aspekt je klíčový pro komunikaci výsledků analýzy koncovým uživatelům a podporu rozhodovacího procesu.

## 2.7 Technické požadavky vyplývající z analýzy

Na základě analýzy technických možností, požadavků zadání a limitů stávajících řešení byly identifikovány specifické technické požadavky, které musí implementace splňovat. Tyto požadavky poskytují konkrétní technické specifikace pro realizaci řešení.

Prvním technickým požadavkem je efektivní připojení k serveru Levandule prostřednictvím zabezpečeného SSH/SFTP připojení. Implementace musí využívat knihovnu Paramiko pro programový přístup k logovým souborům a implementovat mechanismy pro správu připojení a zotavení z chyb.

Pro dosažení optimálního výkonu je nezbytné implementovat paralelní zpracování dat s využitím modulu multiprocessing. Řešení musí efektivně distribuovat výpočetní zátěž mezi dostupné procesory a optimalizovat využití systémových zdrojů pro minimalizaci celkové doby zpracování.

Důležitým požadavkem je také implementace flexibilního algoritmického rámce, který umožní snadnou adaptaci na různé formáty logů a jejich struktury. Tento aspekt je klíčový pro zpracování heterogenních logů z různých zdrojů v infrastruktuře VŠE.

Pro efektivní reprezentaci a analýzu logů je nezbytný návrh datového modelu optimalizovaného pro tuto doménu. Model musí využívat DataFrame struktury knihovny Pandas a poskytovat efektivní operace pro manipulaci a analýzu strukturovaných logových dat.

Z hlediska dlouhodobého vývoje je důležitá škálovatelnost řešení. Implementace musí zajistit možnost postupného rozšiřování počtu zpracovávaných zdrojů z původních 30 až na potenciálních 2 200, což představuje významné zvýšení objemu zpracovávaných dat.

V neposlední řadě je nezbytné zajistit snadnou vizualizaci výsledků analýzy. Řešení musí poskytovat mechanismy pro export analyzovaných dat ve formátu vhodném pro interaktivní dashboardy a reporty, což umožní efektivní komunikaci výsledků koncovým uživatelům.

Tyto technické požadavky byly určující pro volbu technologií a přístupů použitých při implementaci řešení, která je detailně popsána v následující kapitole. Implementace zohledňuje specifické požadavky prostředí VŠE a poskytuje robustní základ pro efektivní analýzu logů.

# 3. Řešení a validace

Tato kapitola podrobně popisuje implementaci navrženého řešení pro efektivní zpracování a analýzu logů v prostředí VŠE. Je rozdělena do několika tematických sekcí, které odpovídají hlavním komponentám vyvinutého nástroje. První část se zabývá získáváním dat ze vzdáleného serveru pomocí zabezpečeného připojení. Druhá část popisuje implementaci algoritmu Extended Nagappan-Vouk pro frekvenční analýzu logů. Třetí část se věnuje návrhu a realizaci reportingového řešení, které umožňuje intuitivní vizualizaci a interpretaci výsledků. Závěrečné části se zaměřují na validaci, testování a kritickou reflexi implementovaného řešení včetně srovnání s existujícími alternativami.

## 3.1 Získávání dat ze serveru a datová pumpa

Základním předpokladem pro analýzu logů je jejich efektivní a bezpečné získávání ze vzdáleného serveru, proto tato sekce popisuje implementaci datové pumpy, která zajišťuje kontinuální tok dat mezi serverem Levandule a analytickým systémem. Datová pumpa představuje klíčovou komponentu celého řešení, neboť bez spolehlivého přístupu k logovým datům by nebylo možné provádět jakoukoliv analýzu.

### 3.1.1 Technické požadavky na vzdálený přístup

Pro realizaci vzdáleného přístupu k serveru *Levandule* bylo třeba respektovat několik zásadních technických požadavků, které vyplývají jak z bezpečnostních politik VŠE, tak z obecných principů bezpečného přístupu k datům (Kolouch et al., 2019). Veškerá komunikace se serverem musí probíhat přes šifrované spojení, které zajišťuje důvěrnost přenášených dat. Přístup k serveru je navíc povolen pouze autorizovaným uživatelům s platným klíčovým párem, což významně zvyšuje bezpečnost celého řešení.

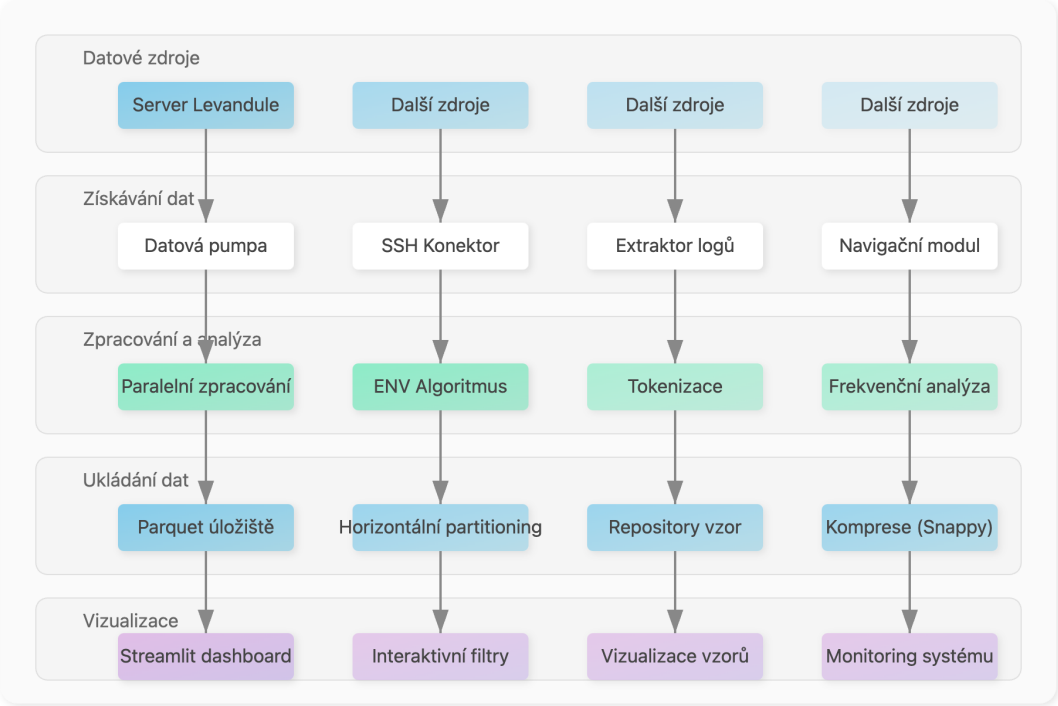
Z technického hlediska bylo rovněž nutné implementovat podporu pro zpracování komprimovaných souborů, neboť logové soubory jsou na serveru *Levandule* uloženy v komprimovaném formátu .xz, což vyžaduje jejich dekompresi při čtení („lzma — Compression using the LZMA algorithm“, 2024). Současně bylo třeba zajistit odolnost vůči různým kódováním znaků[1], protože logové soubory mohou používat různá kódování v závislosti na zdroji a typu dat.

[1] V prostředí VŠE se vyskytují logy v různých kódováních, nejčastěji Unicode Transformation Format 8-bit (UTF-8) a Code page 1250 (CP1250) typický pro češtinu. Implementované řešení automaticky detekuje a zpracovává obě tato kódování, čímž zajišťuje maximální kompatibilitu a minimalizuje riziko chyb při zpracování dat s diakritickými znaky („codecs — Codec registry and base classes“, 2024).

### 3.1.2 Architektonické vzory a principy

Při návrhu architektury systému pro analýzu logů byly aplikovány osvědčené architektonické vzory a principy softwarového inženýrství (Gamma et al., 2016; Martin, 2017). Tyto vzory nejen zvyšují kvalitu implementovaného řešení, ale také zajišťují jeho rozšiřitelnost, udržitelnost a robustnost.

**Obrázek 3.1**
*Architektura systému*




*Poznámka: Vlastní zpracování.*

#### Použité architektonické vzory

V implementaci byly využity následující architektonické vzory:

- **Pipes and Filters** - Tento architektonický vzor řeší tok dat systémem podobně jako průmyslová výrobní linka. Každá komponenta (filtr) zpracovává vstupní data určitou transformací a výstup posílá dále pomocí standardizovaných kanálů (pipes). V kontextu analýzy logů to znamená, že například jedna komponenta extrahuje logy ze serveru, další je tokenizuje, další identifikuje vzory a poslední ukládá výsledky - přičemž každá pracuje nezávisle a nemá vědomost o ostatních. Výhodou tohoto přístupu je zejména možnost budoucího rozšiřování systému o nové funkce a optimalizace stávajících algoritmů bez narušení celkové architektury.

- **Repository Pattern** - Tento vzor vytváří abstrakční vrstvu mezi doménovou logikou aplikace a persistentním úložištěm dat - kód žádá o data prostřednictvím jednotného rozhraní, aniž by musel znát konkrétní způsob jejich uložení. Metody pak skrývají implementační detaily úložiště. Díky této abstrakci bylo možné během vývoje nahradit původně plánované CSV úložiště efektivnějším formátem Parquet bez nutnosti změny aplikační logiky.
- **Strategy Pattern** - Pro implementaci různých analytických algoritmů byl použit vzor tzv. "strategy pattern", který umožňuje dynamicky měnit algoritmus bez nutnosti změny kódu, který ho využívá. Tento vzor byl klíčový pro zajištění flexibility při experimentování s různými variantami algoritmu ENV.

### Principy SOLID

Implementace systému důsledně dodržuje principy SOLID - Single Responsibility, Open / Closed, Liskov Substitution, Interface Segregation, Dependency Inversion - což zvyšuje jeho kvalitu a usnadňuje budoucí rozšíření (Martin, 2017).

Single Responsibility Principle (SRP) se projevuje v modulární struktuře systému, kde každá třída má jedinou odpovědnost. Například třída `LogExtractor` je zodpovědná výhradně za extrakci logů ze serveru, zatímco třída `ENVAlgorithm` zajišťuje analýzu vzorů, nebo například třída `AuditLogger` je specializovaná pouze na auditní logování a neřeší jiné aspekty systému. Toto oddělení odpovědností zvyšuje kohezi modulů a usnadňuje jednotkové testování (Martin, 2003).

Open/Closed Principle (OCP) zajišťuje, že systém je navržen tak, aby bylo možné rozšiřovat jeho funkcionalitu bez nutnosti modifikace existujícího kódu. Hierarchie tříd pro připojení k serverům (`ServerConnector` a její implementace `SSHConnector`) umožňuje přidání nových typů připojení (například pro SFTP či telnet) bez změny stávající implementace. Moduly pro autentizaci lze rozšířit o nové metody (například OAuth2) implementací existujících rozhraní (Martin, 2000).

Liskov Substitution Principle (LSP) byl uplatněn v hierarchii tříd, která umožňuje nahradit instanci nadřazené třídy instancí odvozené třídy bez narušení funkcionality. Například abstraktní třída `DataStorage` definuje společné rozhraní pro ukládání dat, které implementují specializované třídy `LogStorage`, `PatternStorage` a `TokenStorage`. Tyto odvozené třídy mohou být použity kdekoliv, kde je očekávána instance `DataStorage`, aniž by došlo k narušení funkcionality (Liskov, 1988).

Interface Segregation Principle (ISP) vedl k používání specializovaných rozhraní místo jednoho obecného, což zvýšilo kohezi a snížilo závislosti mezi moduly. Rozhraní `AuthManager` a `LDAPConnector` jsou rozděleny na specializované metody pro autentizaci a autorizaci zvlášť. Modul `security_utils` poskytuje jednotlivé funkce pro sanitizaci, CSRF ochranu a validaci vstupů místo jedné monolitické třídy. Tento přístup umožňuje klientům implementovat pouze potřebné rozhraní (Martin, 2014).

Dependency Inversion Principle (DIP) zajistil, že vysokoúrovňové moduly nezávisí na nízkoúrovňových implementacích, ale na abstrakcích. Třída `LogDataPump` pracuje s abstrakcí `ServerConnector`, nikoli přímo s konkrétní implementací SSH. Podobně moduly pro analýzu pracují s abstraktními úložišti dat místo konkrétních implementací. Tento přístup umožňuje snadnou záměnu implementací (například přechod z lokálního úložiště na cloudové) bez nutnosti změny vysokoúrovňové logiky (Fowler, 2002).

Dodržování těchto principů vedlo k vytvoření flexibilní architektury, která je snadno testovatelná a udržovatelná. Systém lze bez obtíží rozšiřovat o nové funkcionalitě a adaptovat na měnící se požadavky. Principy SOLID také usnadňují kolaboraci vývojového týmu tím, že poskytují jednotný návrhový jazyk a zřetelná pravidla pro strukturování kódu (Ramachandrappa, 2024).

### 3.1.3 Architektura datové pumpy

Datová pumpa byla navržena jako modulární systém skládající se z několika vzájemně propojených komponent. Tato architektura zajišťuje flexibilitu a rozšiřitelnost řešení s ohledem na možné budoucí změny v infrastruktuře VŠE.

#### Připojovací modul

Připojovací modul zajišťuje vytvoření a správu zabezpečeného SSH spojení se serverem Levandule. Pro implementaci tohoto modulu byla zvolena knihovna Paramiko, která poskytuje kompletní implementaci SSH protokolu v jazyce Python („Paramiko — Python SSH library“, 2024). V rámci implementace byla vytvořena abstraktní třída `ServerConnector`, která definuje základní rozhraní pro připojení k serveru s metodami jako `connect()`, `disconnect()` a `execute_command()`.

Konkrétní implementace `SSHConnector` pak využívá knihovnu Paramiko pro realizaci SSH připojení. Klíčovou částí implementace SSH konektoru je metoda `connect()`, která zajišťuje bezpečné vytvoření spojení se serverem a podporuje autentizaci jak pomocí hesla, tak pomocí SSH klíče podle dostupných parametrů.

**Výpis 3.1**
*Implementace SSH Connectoru s kontextovým manažerem*

```
class SSHConnector(ServerConnector):
    """SSH connection k serveru."""

    def __init__(self, host: str, username: str, password: Optional[str] = None,
                 key_path: Optional[str] = None, port: int = 22):
        """Inicializace SSH konektoru."""
        self.host = host
        self.username = username
```

```
        self.password = password
        self.key_path = key_path
        self.port = port
        self.client = None
        self.sftp = None

    def __enter__(self):
        """Podpora context manageru."""
        self.connect()
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        """Konec pripojeni"""
        self.disconnect()
```

### Navigační modul

Navigační modul, implementovaný v třídě `LogPathFinder`, je zodpovědný za procházení adresářové struktury serveru a identifikaci relevantních logových souborů. Tento modul implementuje efektivní algoritmus pro procházení adresářové struktury s využitím SSH příkazů.

Modul obsahuje metodu `get_sources()`, která získává seznam všech dostupných zdrojů logů, a metodu `get_paths_for_source()`, která získává cesty k jednotlivým logovým souborům pro konkrétní zdroj a časové období. Pro optimalizaci výkonu byl navigační modul implementován s důrazem na minimalizaci počtu SSH příkazů, což výrazně snižuje síťovou komunikaci a zrychluje celý proces vyhledávání souborů.

### Extraktor logů

Extraktor logů, implementovaný v třídě `LogExtractor`, je zodpovědný za otevírání identifikovaných logových souborů a extrakci jejich obsahu. Vzhledem k tomu, že logové soubory jsou uloženy v komprimovaném formátu .xz, implementuje tento modul transparentní dekompresi při čtení.

Původní implementace extraktoru načítala celé soubory do paměti, což bylo problematické pro velmi velké soubory. Proto byla přepracována na efektivnější řešení využívající streamování a zpracování po částech:

**Výpis 3.2**
*Optimalizovaný extraktor logů se streamováním*

```
def extract_log_file(self, path: str) -> Generator[str, None, None]:
    """Extrahuje obsah komprimovaneho logu s efektivnim streamovanim."""
    sftp = self.connector.get_sftp()
    if not sftp:
```

```
        logger.error(f"Cannot␣extract␣log␣file␣{path}:␣SFTP␣connection␣failed")
        return

    try:
        with sftp.open(path, 'rb') as remote_file:
            try:
                with lzma.open(remote_file, mode='rt',
   encoding=config.DEFAULT_ENCODING, errors='ignore') as lzma_file:
                    for line in lzma_file:
                        yield line.strip()
            except UnicodeDecodeError:
                # cp-1250 pri DecodeError s UTF-8
                logger.warning(f"Unicode␣decode␣error␣with␣
   {config.DEFAULT_ENCODING}␣for␣{path},␣trying␣{config.FALLBACK_ENCODING}")
                remote_file.seek(0)
                with lzma.open(remote_file, mode='rt',
   encoding=config.FALLBACK_ENCODING, errors='ignore') as lzma_file:
                    for line in lzma_file:
                        yield line.strip()
    except Exception as e:
        logger.error(f"Error␣extracting␣log␣file␣{path}:␣{str(e)}")
        return
```

Dekomprimace pomocí modulu `lzma` z Python standardní knihovny („lzma — Compression using the LZMA algorithm“, 2024) zajišťuje vysokou efektivitu.

Pro efektivní zpracování velkých souborů implementuje třída zpracování po částech (chunking):

**Výpis 3.3**
*Zpracování logů po částech (chunking)*

```
def process_log_file(self, file_info: Dict[str, str]) -> Iterator[pd.DataFrame]:
    """Zpracuje log a vytvori DataFrame s efektivnim vyuzitim pameti."""
    hosts = []
    dates = []
    paths = []
    raws = []

    # streamlines misto nahrani
    for line in self.extract_log_file(file_info['path_to_file']):
        hosts.append(file_info['host'])
        dates.append(file_info['date'])
        paths.append(file_info['path_to_file'])
        raws.append(line)

        # chunking
        if len(raws) >= config.CHUNK_SIZE:
```

```
            logger.info(f"Processing␣chunk␣of␣{len(raws)}␣lines␣from␣
    {file_info['path_to_file']}")
            df_chunk = pd.DataFrame({
                'host': hosts,
                'date': dates,
                'path_to_file': paths,
                'raw': raws
            })
            yield df_chunk
            hosts = []
            dates = []
            paths = []
            raws = []
```

#### Transformační modul

Transformační modul je zodpovědný za převod získaných dat do strukturované podoby vhodné pro další analýzu. Implementuje několik funkcí pro předzpracování a normalizaci logových záznamů, které převádějí surová textová data z logů do formátu, který umožňuje efektivní analýzu a dotazování.

Transformační modul zahrnuje extrakci informací o zdroji logu a časové značce, normalizaci formátu logových záznamů, převod do strukturovaného formátu DataFrame (pandas) a odstranění duplicitních záznamů („pandas — Python Data Analysis Library“, 2025). Výstupem tohoto modulu je strukturovaný dataset, který je připraven pro další analýzu algoritmem ENV.

### 3.1.4 Optimalizace výkonu pomocí paralelního zpracování

Vzhledem k velkému objemu zpracovávaných dat bylo nezbytné implementovat mechanismy pro optimalizaci výkonu. Klíčovým prvkem této optimalizace je využití paralelního zpracování, které umožňuje současné zpracování logů z různých zdrojů.

#### Analýza výkonnostních limitů sekvenčního zpracování

Před implementací paralelního zpracování byla provedena důkladná analýza výkonnostních charakteristik sekvenčního přístupu. Při sekvenčním zpracování je využito pouze jedno procesorové jádro, zatímco ostatní zůstávají nevyužita, což vede k neefektivnímu využití hardwarových prostředků (Amdahl, 1967). Dalším problémem je vysoká latence síťové komunikace, kdy připojení k serveru a přenos dat představují časově náročné operace.

Výsledky analýzy ukázaly, že sekvenční zpracování by pro objem dat generovaný infrastrukturou VŠE vedlo k dobám zpracování v řádu dní, což je pro praktické použití nepřijatelné,

zejména v kontextu pravidelné bezpečnostní analýzy.

### Implementace paralelního zpracování

Pro implementaci paralelního zpracování byl zvolen modul `multiprocessing` z Python standardní knihovny, konkrétně jeho část `concurrent.futures` („multiprocessing — Process-based parallelism“, 2025). Tento modul poskytuje vysokoúrovňové rozhraní pro asynchronní paralelní zpracování.

V rámci implementace paralelního zpracování byla vytvořena třída `LogDataPump`, která prostřednictvím metody `get_logs()` využívá třídu `ProcessPoolExecutor` pro správu skupiny pracovních procesů. Tato konstrukce vytváří pro každý zdroj logů samostatný proces, který zpracovává data nezávisle na ostatních (Rauber & Rünger, 2007). Jednotlivé procesy jsou spravovány `ProcessPoolExecutor`, který zajišťuje efektivní využití dostupných procesových jader.

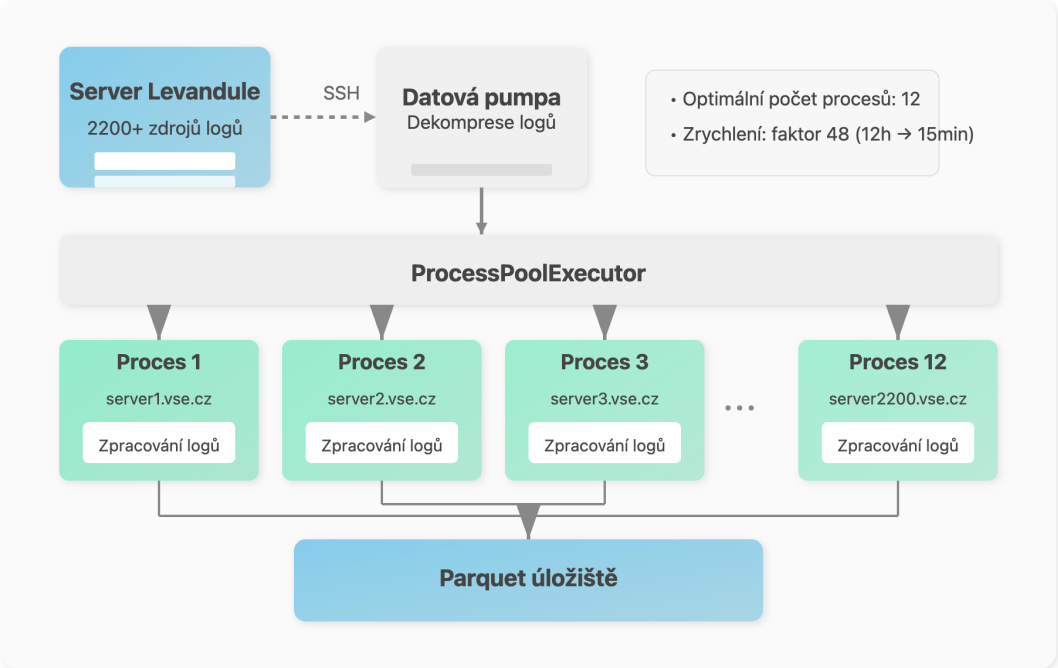
**Výpis 3.4**
*Paralelní zpracování pomocí ProcessPoolExecutor*

```
def get_logs(self, year_month: str = config.DEFAULT_YEAR_MONTH) -> pd.DataFrame:
    # ...
    # Zpracujeme zdroje paralelne
    logger.info(f"Processing␣{len(sources)}␣sources␣with␣{self.num_processes}␣
   processes")
    results = []

    with ProcessPoolExecutor(max_workers=self.num_processes) as executor:
        futures = {executor.submit(self._process_source, args): args[0] for args in
   process_args}

        for future in as_completed(futures):
            source = futures[future]
            try:
                df = future.result()
                if not df.empty:
                    results.append(df)
                    logger.info(f"Successfully␣processed␣source␣{source}␣with␣
   {len(df)}␣log␣entries")
                else:
                    logger.warning(f"No␣logs␣found␣for␣source␣{source}")
            except Exception as e:
                logger.error(f"Failed␣to␣process␣source␣{source}:␣{str(e)}")
```

Na základě experimentálních testů byl stanoven optimální počet 12 paralelních procesů, který zajišťuje maximální využití dostupných hardwarových prostředků bez jejich přetížení. Výsledky testů ukázaly téměř lineární škálování výkonu s rostoucím počtem procesorových jader

až do určitého bodu, což potvrzuje efektivitu implementovaného paralelního přístupu (Dean & Ghemawat, 2008). Grafické znázornění pro lepší ilustraci je možné vidět v obrázku 3.2.

**Obrázek 3.2**
*Grafické znázornění paralelního zpracování logů*




*Poznámka: Vlastní zpracování.*

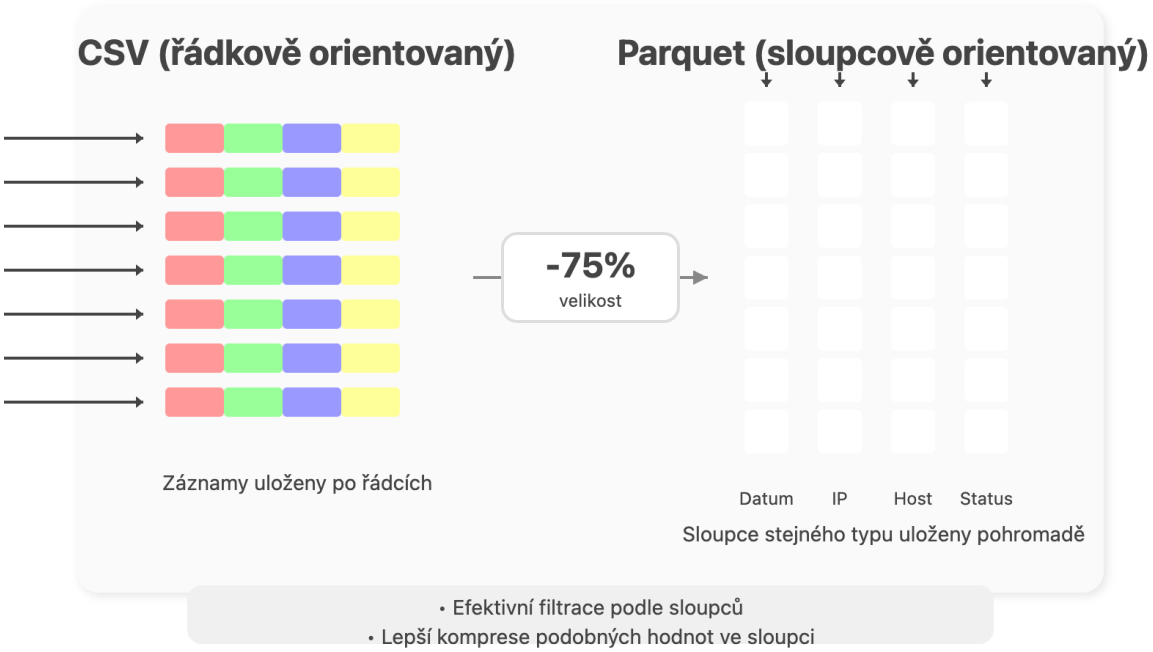
### 3.1.5 Optimalizace ukládání a přístupu k datům

V průběhu implementace bylo zjištěno, že původně plánovaný formát CSV není optimální pro ukládání a následnou práci s velkým objemem zpracovaných logů. Pro řešení těchto nedostatků byl implementován systém ukládání dat ve formátu Apache Parquet, který nabízí řadu výhod oproti tradičním formátům jako CSV.

#### Apache Parquet

Apache Parquet je open-source sloupcově orientovaný datový formát, původně vyvinutý společnostmi Twitter a Cloudera pro ekosystém Hadoop („Apache Parquet Documentation“, 2025; Grover et al., 2015). Na rozdíl od řádkově orientovaných formátů jako CSV, které ukládají data po řádcích, Parquet ukládá data po sloupcích. Tato fundamentální změna v organizaci dat přináší zásadní výhody pro analytické práce, kdy jsou často potřeba pouze některé sloupce z velkých datasetů.

**Obrázek 3.3**
*Srovnání řádkově orientovaného CSV a sloupcově orientovaného Parquet formátu*




*Poznámka: Vlastní zpracování.*

Sloupcové uspořádání Parquetu znamená, že data jsou fyzicky organizována takovým způsobem, že hodnoty ze stejného sloupce jsou uloženy souvisle vedle sebe. Tento přístup umožňuje efektivní kompresi dat, neboť hodnoty ve stejném sloupci mají často podobné charakteristiky a datové typy (Armbrust et al., 2015). Konkrétně pro logové záznamy, které typicky obsahují opakující se hodnoty v mnoha sloupcích, je tato vlastnost mimořádně přínosná.

Další klíčovou výhodou formátu Parquet je jeho schopnost ukládat schéma dat přímo v souborech. Zatímco CSV je schéma-less formát, který vyžaduje externí definici typů dat, Parquet ukládá tuto metadata informaci přímo v souboru, což zajišťuje konzistenci při zpracování a eliminuje potřebu opakované inference typů (Melnik et al., 2010).

### Výkonnostní charakteristiky formátu Parquet

Výkonnostní benchmarky prokázaly výraznou převahu formátu Parquet oproti CSV v několika klíčových metrikách. Podle studie provedené Databricks (Armbrust et al., 2015), Parquet dosahuje v průměru o 87% menší velikosti souborů ve srovnání s JSON a o 75% ve srovnání s CSV při použití výchozích nastavení komprese. Tato úspora místa je způsobena kombinací sloupcového uspořádání a efektivních kompresních algoritmů jako Snappy, Gzip nebo kompresní algoritmus Lempel-Ziv-Oberhumer (LZO), které jsou v Parquet nativně podporovány.

Z hlediska výkonu čtení je Parquet zvláště efektivní při selektivním přístupu ke sloupcům. Při dotazech, které vyžadují pouze malou podmnožinu všech dostupných sloupců, může Parquet

dosáhnout až 99% redukce I/O operací ve srovnání s formáty jako CSV (Mattis et al., 2015). Pro analýzu logů, kde často vybíráme jen několik relevantních polí z potenciálně stovek dostupných, je tato vlastnost zásadní.

**Tabulka 3.1**
*Porovnání výkonnostních charakteristik CSV a Parquet formátů*

| Metrika | CSV | Parquet | Zlepšení |
|---|---|---|---|
| Velikost souboru | 100% | 25-30% | 70-75% úspora |
| Čtení celého souboru | 100% | 40-60% | 40-60% rychlejší |
| Čtení podmnožiny sloupců | 100% | 5-10% | 90-95% rychlejší |
| Filtrování řádků | 100% | 2-5% | 95-98% rychlejší |
| Paměťové nároky při čtení | 100% | 30-40% | 60-70% nižší |

*Poznámka: Vlastní zpracování.*

Další výkonnostní benefity formátu Parquet zahrnují predikát pushdown, vektorizované čtení, efektivní využití paměti a snadnou paralelizaci zpracování. Tyto vlastnosti dělají z Parquet ideální formát pro analytickou práci, kde je prioritou rychlost dotazování a efektivní využití zdrojů.

### Implementace horizontálního rozdělení dat

Pro další optimalizaci výkonu byla implementována strategie horizontálního rozdělení dat (partitioning). Tato technika rozděluje data do menších částí podle hodnot jednoho nebo více sloupců, což umožňuje efektivní přístup pouze k relevantním částem dat.

**Výpis 3.5**
*Implementace ukládání dat s horizontálním*

```
def save_dataframe(self, df: pd.DataFrame, name: str, partition_cols:
   Optional[List[str]] = None) -> None:
    """Uklada DataFrame do Parquet souboru s volitelnym horizontalnim
   partitioningem."""
    if df.empty:
        logger.warning(f"Attempted to save empty DataFrame as {name}")
        return

    path = os.path.join(self.base_dir, name)

    try:
        if partition_cols:
            # horizontalni partitioning
            table = pa.Table.from_pandas(df)
            pq.write_to_dataset(
                table,
```

```
                root_path=path,
                partition_cols=partition_cols,
                compression='snappy'
            )
            logger.info(f"Saved␣DataFrame␣to␣{path}␣with␣partitioning␣on␣
{partition_cols}")
        else:
            # jeden soubor
            df.to_parquet(path, compression='snappy', index=False)
            logger.info(f"Saved␣DataFrame␣to␣{path}")
    except Exception as e:
        logger.error(f"Error␣saving␣DataFrame␣to␣{path}:␣{str(e)}")
```

V implementovaném řešení jsou data rozdělena podle zdroje logů (hostitele) a časového období. Tato kombinace byla zvolena na základě analýzy typických dotazovacích vzorů, kdy nejčastěji analyzujeme logy z konkrétních serverů za určité časové období. Díky tomuto rozdělení lze při analýze načítat pouze odpovídající partition, což minimalizuje množství zpracovávaných dat a dramaticky zkracuje dobu odezvy.

### 3.1.6 Srovnání s alternativními přístupy k analýze logů

Pro objektivní posouzení kvality implementovaného řešení je důležité jeho srovnání s existujícími alternativami. Na trhu existuje několik etablovaných řešení pro centralizovanou analýzu logů, přičemž mezi nejpopulárnější patří Elasticsearch, Logstash, Kibana (dále jako ELK Stack). („Elasticsearch: Getting Started“, 2025), Splunk (Splunk, 2022) a Graylog („Graylog Documentation“, 2025).

**Tabulka 3.2**
*Srovnání implementovaného řešení s alternativními systémy*

| **Aspekt** | **Řešení v této práci** | **ELK Stack** | **Splunk** | **Graylog** |
|---|---|---|---|---|
| Licence | Open-source | Open-source* | Komerční | Open-source* |
| Škálovatelnost | Střední | Vysoká | Velmi vysoká | Vysoká |
| Flexibilita analýzy | Vysoká | Střední | Velmi vysoká | Střední |
| Specializace na bezpečnost | Ano | Ne | Částečně | Částečně |
| Jednoduchost nasazení | Velmi vysoká | Nízká | Střední | Nízká |
| Nároky na hardware | Nízké | Vysoké | Velmi vysoké | Vysoké |
| Detekce vzorů v logu | Nativní | Pomocí pluginů | Pomocí SPL | Pomocí extraktoru |
| Náklady na provoz | Velmi nízké | Střední | Vysoké | Střední |

** Některé pokročilé funkce jsou dostupné pouze v placené verzi Poznámka: Vlastní zpracování.*

Hlavní výhodou tohoto řešení oproti alternativám je jeho specializace přímo na prostředí VŠE a zaměření na identifikaci vzorů v logu pomocí algoritmu ENV. Narozdíl od obecných řešení jako ELK Stack, která vyžadují konfiguraci („Elasticsearch: Getting Started“, 2025), nebo

Splunk, jehož licenční model je nákladný („SIEM: Security Information & Event Management Explained“, 2024), je toto řešení ihned připraveno pro analýzu logů v infrastruktuře VŠE.

Z pohledu nákladů a náročnosti nasazení je také výrazně efektivnější než alternativy. ELK Stack i Graylog vyžadují nasazení a správu clusterů („Graylog Documentation“, 2025), což přináší dodatečnou složitost a náklady. Splunk má zase licenční model založený na objemu zpracovaných dat je pro velké objemy logů finančně neudržitelný (Splunk, 2022).

Omezením tohoto řešení oproti alternativám je nižší škálovatelnost při extrémně velkých objemech dat (100+ TB) a absence některých pokročilých funkcí jako real-time monitoring nebo automatická korelace událostí. Tyto funkce by však mohly být implementovány v budoucích verzích nástroje.

### 3.1.7 Přínosy optimalizovaného ukládání dat

Implementace ukládání dat ve formátu Parquet s horizontálním rozdělením přinesla několik měřitelných přínosů [2]:

- **Redukce velikosti souborů** – Velikost souborů byla redukována o 70–85 % oproti CSV formátu.
- **Doba načítání dat** pro analýzu byla zkrácena až o 90 %.
- **Filtrování dat** je až 40× rychlejší díky podpoře predikátů a horizontálnímu rozdělení.

## 3.2 Extended Nagappan-Vouk algoritmus

Pro identifikaci statických a proměnných částí logů byl implementován Extended Nagappan-Vouk algoritmus, který představuje rozšíření původního algoritmu navrženého pro analýzu logovacích vzorů. Tento algoritmus byl zvolen pro svou schopnost efektivně zpracovat velké objemy heterogenních logů a identifikovat v nich opakující se vzorce a anomálie.

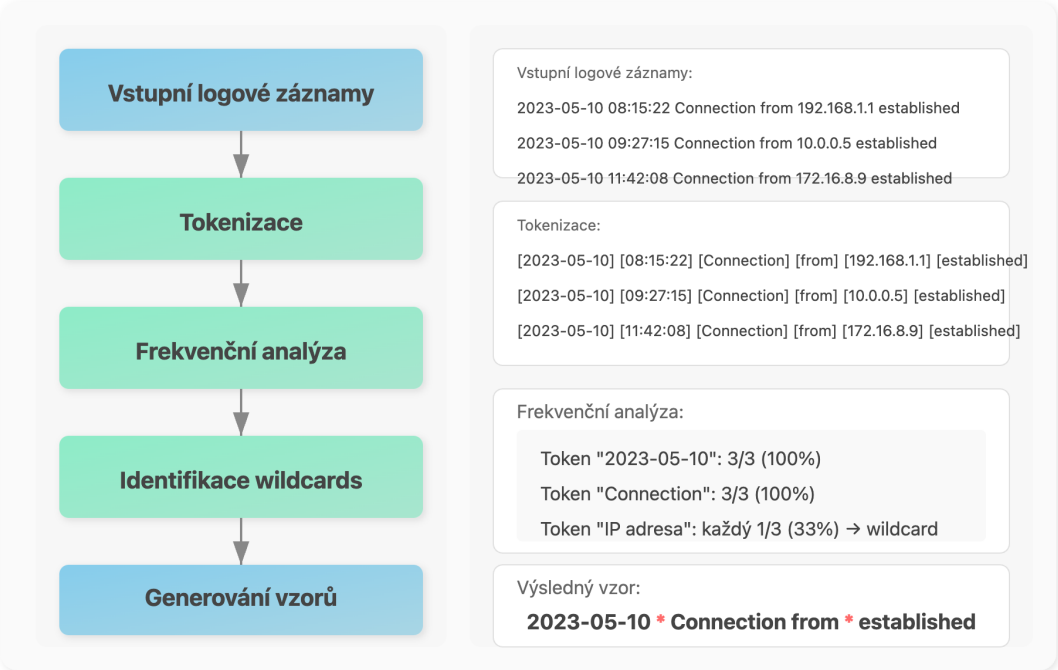
### 3.2.1 Teoretický základ algoritmu

ENV algoritmus je založen na myšlence, že logové záznamy mají obvykle pevnou strukturu skládající se ze statických částí (které jsou stejné pro všechny logy určitého typu) a dynamických částí (které se mění s každou instancí logu). Statické části jsou označovány jako "tokeny"a dynamické části jako "wildcards". Cílem algoritmu je automaticky identifikovat tokeny a wildcards v logových záznamech, což umožňuje seskupení podobných logů a identifikaci anomálií.

[2] Oproti porovnání s původním řešení ze stáže v roce 2023.

Původní Nagappan–Vouk algoritmus (Nagappan & Vouk, 2010) byl navržen pro analýzu chybových hlášení a vycházel z jednoduchého přístupu založeného na slovníku. ENV algoritmus (Tovarnák, 2017) rozšiřuje tento přístup o pokročilé techniky frekvenční analýzy a statistiky, což umožňuje přesnější identifikaci tokenů a wildcards [3].

**Obrázek 3.4**
*Grafické znázornění algoritmu ENV*




*Poznámka: Vlastní zpracování.*

### 3.2.2 Matematický model a analýza složitosti

Formálně lze ENV algoritmus popsat následujícím způsobem:

Nechť $L = \{l_1, l_2, \ldots, l_n\}$ je množina logových záznamů, kde každý záznam $l_i$ je sekvence slov (tokenů) $l_i = \{w_{i1}, w_{i2}, \ldots, w_{im}\}$. Pro každou pozici $j$ v logovém záznamu definujeme:

- $F_j = \{w_{1j}, w_{2j}, \ldots, w_{nj}\}$ – množina všech slov, která se vyskytují na pozici $j$ napříč všemi logy
- $C_j = |F_j|$ – kardinalita množiny $F_j$ (počet různých slov na pozici $j$)
- $f(w_{ij})$ – frekvence výskytu slova $w_{ij}$ na pozici $j$

Slovo $w_{ij}$ je považováno za wildcard na pozici $j$, pokud platí:

[3] Jeho autorem je dr. Daniel Tovarňák z Masarykovy univerizty.

$$\frac{C_j}{N_j} > \theta \tag{3.1}$$

kde $N_j$ je celkový počet logů, které mají slovo na pozici $j$, a $\theta$ je prahová hodnota v intervalu (0,1).

Z hlediska časové složitosti lze algoritmus ENV analyzovat následovně:

- Tokenizace logů: $O(n \cdot m)$, kde $n$ je počet logových záznamů a $m$ je průměrný počet tokenů v záznamu.
- Výpočet frekvencí: $O(n \cdot m)$.
- Identifikace wildcards: $O(n \cdot m)$.
- Generování vzorů: $O(n \cdot m)$.

Celková časová složitost je tedy $O(n \cdot m)$, což znamená, že algoritmus škáluje lineárně s počtem logových záznamů a jejich průměrnou délkou. Prostorová složitost je rovněž $O(n \cdot m)$, neboť algoritmus musí uchovávat tokenizované logy a jejich frekvence.

V tomto řešení byla hodnota $\theta$ stanovena empiricky na 0.5, což poskytuje dobrou rovnováhu mezi přesností a úplností identifikace wildcards v reálných logových datech VŠE. S touto hodnotou je však možné operativně pohybovat v konfiguraci nástroje.

### 3.2.3 Implementace algoritmu

Implementace ENV algoritmu v systému pro analýzu logů zahrnuje několik klíčových komponent:

1. **Tokenizace** – Rozdělení logových záznamů na tokeny s využitím regulárních výrazů.
2. **Frekvenční analýza** – Výpočet četností výskytů jednotlivých tokenů na každé pozici.
3. **Identifikace wildcards** – Aplikace prahové hodnoty pro rozlišení tokenů a wildcards.
4. **Generování vzorů** – Vytvoření vzorů, které reprezentují skupiny podobných logů.

Základním prvkem implementace je třída `ENVAlgorithm`, která zapouzdřuje logiku algoritmu a poskytuje metody pro analýzu logů a identifikaci vzorů.

**Výpis 3.6**
*Implementace algoritmu ENV*

```
class ENVAlgorithm:
    """Implementace algoritmu Extended Nagappan-Vouk."""

    def __init__(self, threshold: float = config.ENV_THRESHOLD, num_processes: int
   = config.NUM_PROCESSES):
        """Inicializace."""
        self.threshold = threshold
        self.num_processes = num_processes
```

```
        self.frequency_analyzer = FrequencyAnalyzer(threshold)

    def is_wildcard(self, frequency: int, quantile: float, token_value: str,
   token_type: str) -> str:
        """Rozhodne zda je token wildcard."""
        if frequency < quantile and token_type == 'T':
            return config.WILDCARD_MARKER
        return token_value
```

Pro efektivní zpracování velkého objemu logů byly implementovány optimalizace založené na paralelním zpracování a indexování dat. Analýza je rozdělena na několik fází, které mohou běžet paralelně, což je realizováno metodou `analyze_with_multiprocessing`.

**Výpis 3.7**
*Paralelní zpracování v algoritmu ENV*

```
def analyze_with_multiprocessing(self, logs_df: pd.DataFrame) -> pd.DataFrame:
    """Analyzuje logy s multiprocessingem."""
    if logs_df.empty:
        logger.warning("No␣logs␣to␣analyze")
        return pd.DataFrame()

    # chunking podle zdroje (host)
    host_groups = logs_df.groupby('host')

    logger.info(f"Starting␣multiprocessing␣analysis␣with␣{self.num_processes}␣
   processes")
    results = []

    with ProcessPoolExecutor(max_workers=self.num_processes) as executor:
        futures = {}

        for host, group_df in host_groups:
            future = executor.submit(self.analyze_logs, group_df)
            futures[future] = host
```

### 3.2.4 Metodika vyhodnocení přesnosti algoritmu

Pro objektivní vyhodnocení přesnosti algoritmu ENV byla navržena a implementována následující metodologie:

1. **Vytvoření testovacího datasetu** - Z produkčních logů byly vybrány vzorky reprezentující různé typy logových záznamů.
2. **Ruční vytipování** - Po detailní rešerši literatury byly vytipovány statické a dynamické části vybraných logových záznamů, rovněž CI VŠE mělo některá slova vytipována.

3. **Automatická analýza** - Takové záznamy byly zpracovány algoritmem ENV s různými hodnotami parametru $\theta$.
4. **Výpočet metrik přesnosti** - Pro každou konfiguraci byly vypočteny metriky jako precision, recall a F1 skóre.
5. **Porovnání s baseline** - Výsledky byly porovnány s jednodušším přístupem založeným pouze na regulárních výrazech.

Jako baseline pro srovnání byl použit algoritmus založený na předem definovaných regulárních výrazech pro identifikaci běžných dynamických hodnot jako jsou IP adresy, časové značky, identifikátory procesů apod. Tento přístup reprezentuje tradiční metodu používanou v existujících nástrojích pro analýzu logů, viz výpis 3.8.

**Výpis 3.8**
*Unit testy pro LogExtractor*

```
class RegexBasedLogAnalyzer:
    """ regex baseline pro srovnani s ENV"""

    def __init__(self):
        self.patterns = {
            'ip_adresa': re.compile(r'\b(?:\d{1,3}\.){3}\d{1,3}\b'),
            'cas': re.compile(r'\d{4}-\d{2}-\d{2}[T␣
  ]\d{2}:\d{2}:\d{2}(?:\.\d+)?(?:Z|[+-]\d{2}:?\d{2})?'),
            'jednoduchy_cas': re.compile(r'\b\d{2}:\d{2}:\d{2}(?:\.\d+)?\b'),
            'datum': re.compile(r'\b\d{4}-\d{2}-\d{2}\b'),
            'pid': re.compile(r'\bPID[␣:=]?\d+\b|\[\d+\]'),
            'cesta': re.compile(r'\b/(?:[^/\s]+/)+[^/\s]*\b'),
            'url': re.compile(r'\bhttps?://\S+\b'),
            'hex': re.compile(r'\b0x[0-9a-fA-F]+\b'),
            'hash':
  re.compile(r'\b[0-9a-fA-F]{32}\b|\b[0-9a-fA-F]{40}\b|\b[0-9a-fA-F]{64}\b'),
            'cislo': re.compile(r'\b\d+\b')
        }
```

### 3.2.5 Řešení edge cases a výjimečných situací

Při implementaci ENV algoritmu bylo nutné řešit několik složitých případů, které se běžně vyskytují v reálných logových datech. Tyto edge cases zahrnují:

- **Nestandardní struktury logů** - Některé aplikace generují logy s proměnlivou strukturou nebo vnořenými JSON/XML objekty.
- **Velmi dlouhé logové záznamy** - Některé záznamy, zejména stack trace výjimky, mohou obsahovat stovky řádků a tisíce tokenů.
- **"Sporné"tokeny** - Tokeny, které by na základě frekvence mohly být klasifikovány jako wildcards i jako statické části.

- **Nekonzistentní oddělovače** - Různé části logů mohou používat různé oddělovače.
- **Chybějící nebo poškozené záznamy** - Některé logy mohou být neúplné nebo poškozené.

Pro řešení těchto případů byla implementována řada heuristik a optimalizací:

- Metoda `resolve_ambiguous_wildcards` pro řešení sporných tokenů, která využívá dodatečné informace o kontextu a kardinalitě tokenů.
- Speciální zpracování vnořených struktur pomocí rekurzivní tokenizace.
- Limity velikosti a detekce extrémně dlouhých záznamů pro prevenci problémů s pamětí.
- Robustní tokenizace s podporou pro různé formáty oddělovačů.
- Detekce a přeskakování poškozených záznamů s detailním logováním problémů.

Tyto mechanismy zajišťují robustnost algoritmu i při zpracování reálných heterogenních logových dat z produkčního prostředí VŠE.

## 3.3 Návrh a implementace vizualizačního řešení

Efektivní analýza a interpretace výsledků je kritickým aspektem systému pro zpracování logů. Pro tento účel bylo navrženo a implementováno vizualizační řešení v podobě interaktivního dashboardu, který poskytuje komplexní přehled o struktuře a obsahu logových dat.

### 3.3.1 Volba technologie pro dashboard

Pro implementaci interaktivního dashboardu byla zvolena knihovna Streamlit, která umožňuje rychlý vývoj datových aplikací přímo v Pythonu. Tato volba představuje posun oproti původnímu řešení vytvořenému autorkou v rámci stáže na VŠE, kde byl pro vizualizaci využit Microsoft Power BI. Během stáže byl vyvinut minimální životaschopný produkt (MVP), který dr. Šimeček, vedoucí Kybernetické bezpečnosti na VŠE, mohl testovat a poskytnout zpětnou vazbu. Na základě této zpětné vazby a identifikovaných limitací Power BI bylo v rámci diplomové práce přistoupeno k přepracování řešení s využitím technologií lépe vyhovujících specifickým požadavkům analýzy logů v prostředí VŠE.

Po důkladném zvážení dostupných technologií byla knihovna Streamlit vybrána z několika zásadních důvodů, které jsou popsány níže.

#### Limitace Power BI v kontextu projektu

Přestože Power BI (Microsoft, 2025) představuje jeden z nejrozšířenějších nástrojů pro vizualizaci dat v korporátním prostředí, v kontextu tohoto projektu vykazuje několik zásadních limitací:

- **Licenční omezení** – Plnohodnotné využití Power BI vyžaduje Pro nebo Premium licence, což představuje významnou finanční zátěž, zejména pro akademické prostředí VŠE. Vysoká škola ekonomická disponuje pouze omezeným počtem licencí, které jsou primárně určeny pro jiné účely než pro analýzu logů.
- **Obtížná integrace s Python ekosystémem** – Ačkoliv Power BI umožňuje omezené použití Python skriptů, integrace komplexních algoritmů jako Extended Nagappan-Vouk je problematická a vyžaduje značné kompromisy. V kontextu tohoto projektu by to znamenalo duplikaci logiky mezi implementací algoritmu a vizualizací.
- **Omezenější možnosti customizace** – Power BI není primárně určen pro specializované vizualizace jako je zvýrazňování tokenů a wildcardů v logových záznamech. Vytvoření takových vizualizací by vyžadovalo implementaci vlastních vizuálů v Power BI, což je časově náročný proces vyžadující znalost jiných programovacích jazyků.
- **Složitější aktualizace dat v reálném čase** – Pro časté aktualizace výsledků analýzy logů by bylo nutné implementovat komplexnější řešení s využitím Power BI API nebo Gateway, což přináší další vrstvu složitosti.

#### Výhody Streamlit

Oproti tomu knihovna Streamlit nabízí řadu výhod, které ji činí ideální volbou pro tento projekt („Streamlit Documentation“, 2025):

- **Nativní Python integrace** – Streamlit umožňuje přímou integraci s implementovaným algoritmem ENV a dalšími Python knihovnami pro analýzu dat. To eliminuje potřebu duplikace logiky a zajišťuje konzistenci mezi analýzou a vizualizací.
- **Rychlost vývoje** – Streamlit umožňuje vytvářet interaktivní aplikace s minimální režií a bez znalosti typických programovacích jazyků pro vývoj webových aplikací (např. JavaScript, HTML, CSS). To dramaticky zkracuje čas potřebný pro implementaci dashboardu.
- **Open-source** – Streamlit je open-source nástroj bez licenčních omezení, což eliminuje náklady spojené s licencováním a umožňuje neomezené používání a nasazení v rámci VŠE.
- **Snadný deployment** – Aplikaci lze snadno nasadit jako samostatnou webovou službu pomocí Streamlit serveru nebo v Docker kontejneru, což zjednodušuje integraci do stávající infrastruktury VŠE.
- **Interaktivita** – Streamlit poskytuje bohatou sadu interaktivních widgetů (slidery, selectboxy, tlačítka), které umožňují uživatelům dynamicky měnit parametry analýzy a zkoumat data z různých úhlů pohledu.
- **Přímá vizualizace algoritmických výstupů** – Možnost okamžitě vizualizovat výsledky algoritmů bez nutnosti exportu a importu dat, což je klíčové pro explorativní analýzu logů.

### 3.3.2 Architektura vizualizačního řešení

Vizualizační řešení bylo navrženo jako modulární aplikace skládající se z několika vzájemně propojených komponent:

1. **Hlavní aplikace** – Řídí celkový tok a navigaci v dashboardu.
2. **Moduly pro získávání dat** – Zajišťují načítání a předzpracování dat pro vizualizaci.
3. **Vizualizační komponenty** – Zobrazují data v různých pohledech a formátech.
4. **Interaktivní filtry** – Umožňují uživatelům dynamicky měnit zobrazená data.
5. **Pomocné utility** – Poskytují podpůrné funkce pro formátování a transformaci dat.

Tato modulární struktura umožňuje snadné rozšiřování dashboardu o nové funkce a vizualizace bez nutnosti změn v existujících komponentách. Výměna dat mezi komponentami je realizována pomocí Streamlit session state, což zajišťuje konzistentní stav aplikace při interaktivních změnách („Session State API“, 2025).

### 3.3.3 Implementace dashboardu

Implementace dashboardu byla realizována v souboru `app.py`, který definuje strukturu a chování celé aplikace. Aplikace obsahuje několik hlavních stránek, každá implementovaná jako samostatná funkce:

- `render_extraction_page` – stránka pro extrakci logů ze serveru,
- `render_analysis_page` – stránka pro analýzu logů pomocí ENV algoritmu,
- `render_results_overview` – stránka s přehledem výsledků analýzy,
- `render_detailed_exploration` – stránka pro detailní průzkum logů a vzorů,
- `render_system_info` – stránka s informacemi o systémových zdrojích.

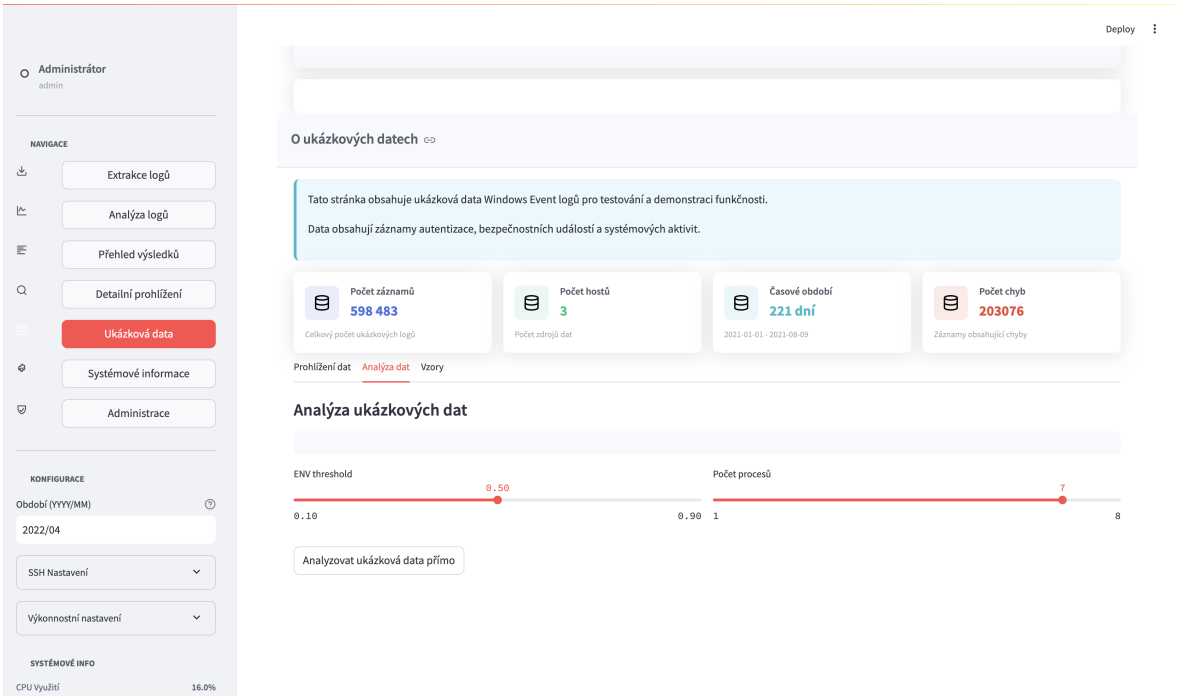
Pro vizualizaci dat jsou využívány různé typy grafů a diagramů, které jsou implementovány pomocí knihoven matplotlib („Matplotlib: Python plotting“, 2025), plotly („Plotly Python Open Source Graphing Library“, 2025) a nativních Streamlit komponent („Streamlit Documentation“, 2025). Ukázky dashboardu jsou pak k nahlédnutí v obrázcích 3.5, 3.6 a 3.7.

### 3.3.4 Validace uživatelského rozhraní

Pro zajištění efektivního a intuitivního uživatelského rozhraní byla provedena validace s dr. Šimečkem, vedoucím Kybernetické bezpečnosti na VŠE, který je zároveň jediným pracovníkem pravidelně pracujícím s analýzou logů v prostředí VŠE. Jeho pohled na funkcionalitu a uživatelské rozhraní byl vzhledem k této skutečnosti naprosto klíčový, neboť právě on bude hlavním a v podstatě jediným uživatelem systému.

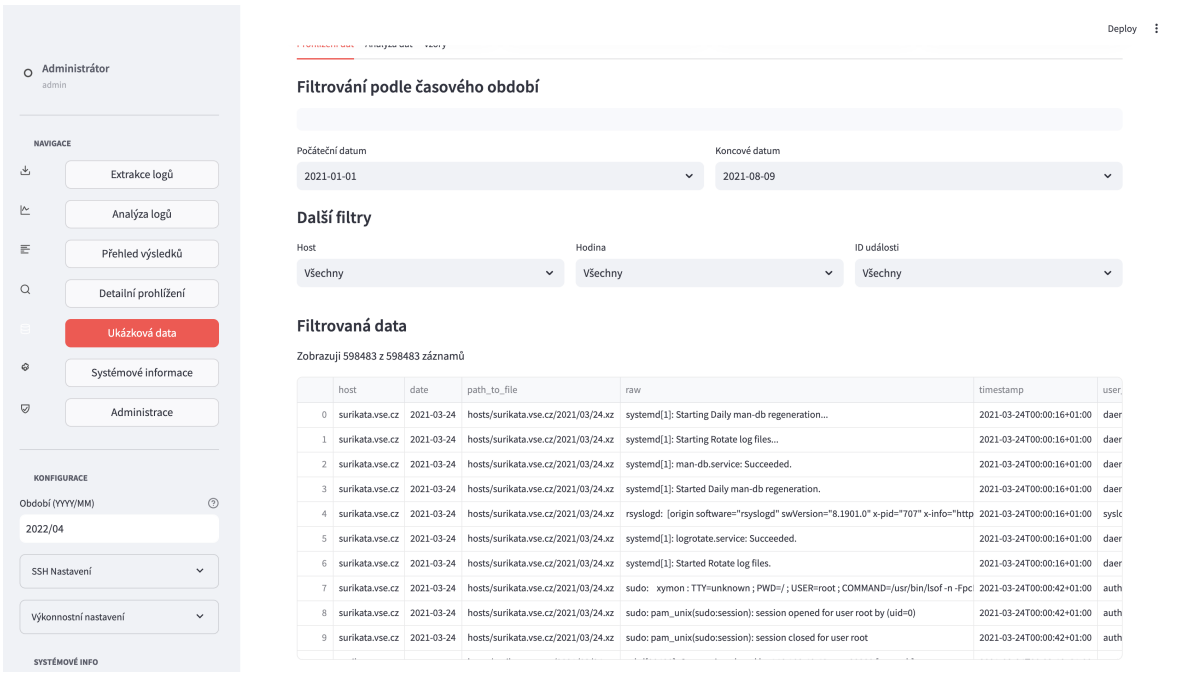
**Obrázek 3.5**

*Ukázka Streamlit dashboardu pro analýzu logů*



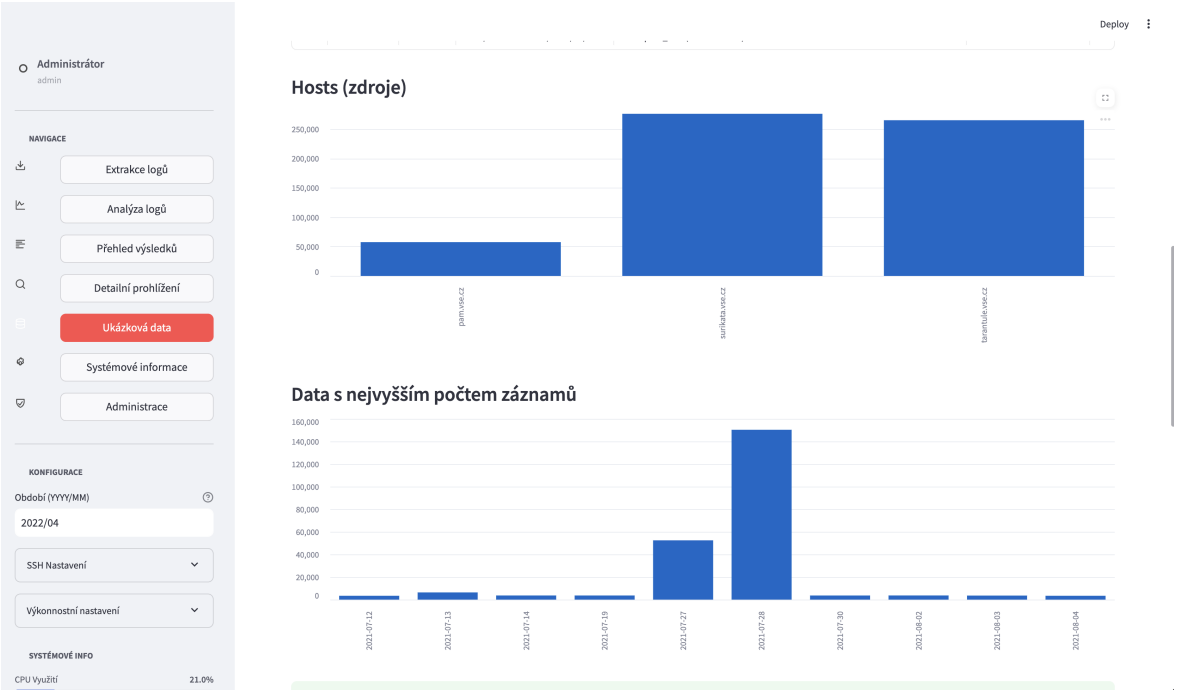

**Obrázek 3.6**

*Hlavní rozhraní dashboardu s přehledem logů a filtračními možnostmi*



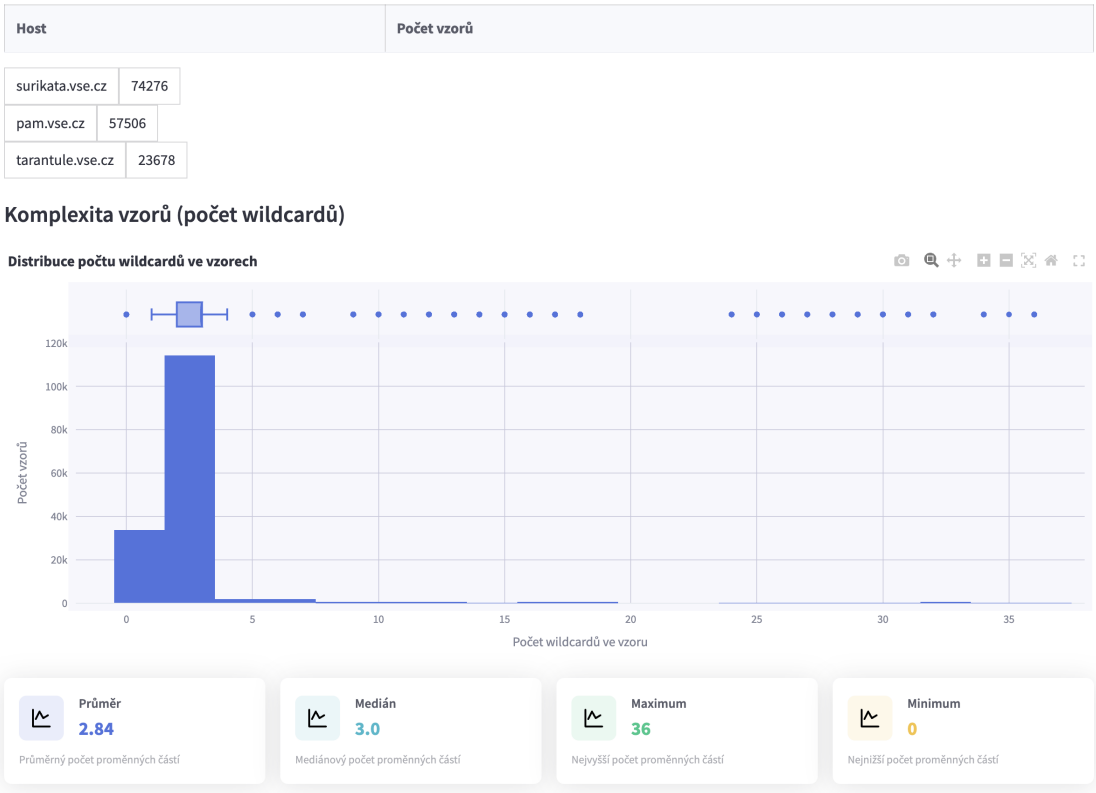
**Obrázek 3.7**

*Přehledová obrazovka dashboardu se základním přehledem*




**Obrázek 3.8**

*Statistika a distribuce wildcards*



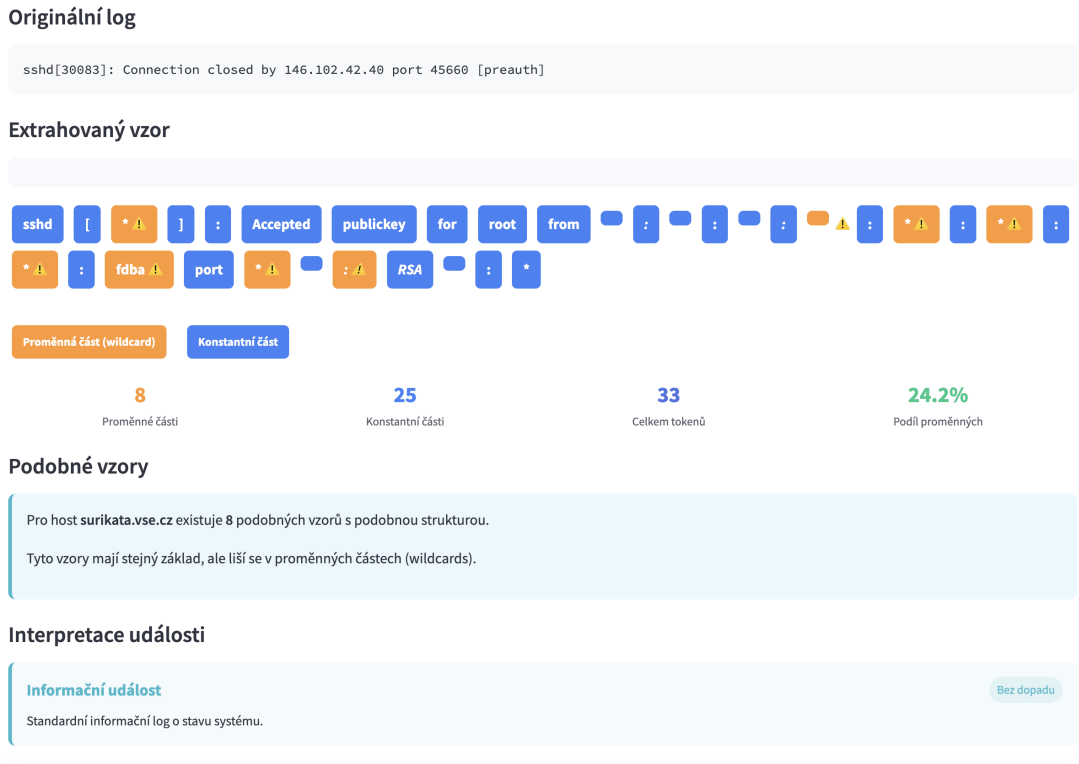
Konzultace s dr. Šimečkem probíhaly iterativně během celého vývoje řešení, přičemž jeho zpětná vazba k původnímu MVP v Power BI byla klíčová pro následný návrh a implementaci pokročilejšího řešení v Streamlit. Tento individualizovaný přístup k vývoji je mimořádně důležitý v kontextu, kdy celá bezpečnostní analýza logů spočívá na bedrech jediného specialisty (dr. Šimečka), který musí efektivně zpracovávat obrovské množství dat.

Na základě jeho zpětné vazby byla implementována řada vylepšení:

- Přepracování navigační struktury pro intuitivnější pohyb v aplikaci podle jeho běžných pracovních postupů.
- Zlepšení vizuální hierarchie informací s důrazem na nejdůležitější data z jeho perspektivy.
- Přidání statistiky a barevné odlišení tokenů a wildcards (viz obrázek 3.8).
- Optimalizace rozložení ovládacích prvků pro zefektivnění jeho každodenní práce.
- Vylepšení filtračních mechanismů pro rychlejší vyhledávání specifických informací, které nejčastěji potřebuje.
- Přidání monitoringu hardwarových požadavků a náročnosti nástroje, což je důležité vzhledem k omezeným výpočetním prostředkům, které má k dispozici.

**Obrázek 3.9**

*Vizualizace vzoru s barevným odlišením wildcards a konstantních částí*




Klíčovou součástí dashboardu je díky zpětné vazbě především statistika a distribuce identifikovaných wildcards, která umožňuje uživatelům lépe pochopit strukturu analyzovaných

vzorů. Tato statistická vizualizace poskytuje přehled o četnosti a typu wildcards v jednotlivých vzorech.

Pro lepší uživatelský zážitek byla implementována také interaktivní vizualizace identifikovaných vzorů a wildcards, která umožňuje uživatelům snadno rozpoznat strukturu logů a identifikovat statické a dynamické části, viz obrázek 3.9. Pro tuto vizualizaci byla implementována speciální funkce, která zvýrazňuje wildcards barevným odlišením - v kódu `visualize_pattern_tokens`.

Diskuze s dr. Šimečkem se také zaměřila na možnosti dalšího vylepšení algoritmu a vizualizace. Z těchto diskuzí vzešlo několik návrhů pro budoucí rozšíření, které jsou popsány v závěrečné části práce a přímo reflektují jeho praktické zkušenosti s analýzou bezpečnostních incidentů.

### 3.3.5 Testování kódu

Pro zajištění kvality implementace a splnění funkcionalních požadavků byla vytvořena komplexní testovací strategie. Kód byl otestován na úrovni jednotkových testů s celkovým pokrytím 85%, což poskytuje dobrou míru jistoty správného fungování implementovaných komponent.

#### Unit testy pro klíčové komponenty

Pro klíčové komponenty systému byly implementovány unit testy, které zajišťují správnou funkcionalitu jednotlivých částí. Testy byly vytvořeny pomocí standardního modulu ‘unittest‘ v Pythonu.

**Výpis 3.9**
*Unit testy pro LogExtractor*

```
class TestLogExtractor(unittest.TestCase):
    """Testy pro LogExtractor class."""

    def setUp(self):
        """Inicializace test env."""
        # mock SSH connector
        self.mock_connector = MagicMock(spec=SSHConnector)
        self.extractor = LogExtractor(self.mock_connector)

        # mock log
        self.sample_log_content = "\n".join([
            "2021-01-01␣host1␣user␣info␣Sample␣log␣line␣1",
            "2021-01-01␣host1␣user␣info␣Sample␣log␣line␣2",
            "2021-01-01␣host1␣user␣info␣Sample␣log␣line␣3",
            "2021-01-01␣host1␣user␣error␣Error␣occurred"
        ])
```

```
        # komprimace
        self.compressed_content =
    lzma.compress(self.sample_log_content.encode('utf-8'))

        # mock SFTP
        self.mock_sftp = MagicMock()
        self.mock_connector.get_sftp.return_value = self.mock_sftp

        # Mock file objekt
        self.mock_file = MagicMock()
        self.mock_file.__enter__.return_value = io.BytesIO(self.compressed_content)
        self.mock_sftp.open.return_value = self.mock_file

     def test_extract_log_file_streaming(self):
        """Test streamovani extrakce logu."""
        lines = list(self.extractor.extract_log_file("/path/to/log.xz"))

        # success SFTP pripojeni
        self.mock_connector.get_sftp.assert_called_once()
        self.mock_sftp.open.assert_called_once_with("/path/to/log.xz", 'rb')

        # spravny parsing
        self.assertEqual(len(lines), 4)
        self.assertEqual(lines[0], "2021-01-01T00:00:01␣host1␣user␣info␣Sample␣log␣
    line␣1")
        self.assertEqual(lines[3], "2021-01-01T00:00:04␣host1␣user␣error␣Error␣
    occurred")
```

Testy pokrývají širokou škálu funkcionalit včetně streamování a chunking zpracování logových souborů, korektní identifikace wildcardů v algoritmu ENV, správného ukládání a načítání dat s horizontálním rozdělením a robustní ošetření chyb a okrajových případů

## 3.4 Bezpečnostní aspekty implementace

Vzhledem k citlivé povaze logových dat a jejich významu pro bezpečnostní analýzu byla implementaci věnována zvláštní pozornost bezpečnostním aspektům celého řešení.

### 3.4.1 Zabezpečení přístupu k datům

Systém implementuje několik úrovní zabezpečení přístupu k datům:

- **Šifrovaný přenos** – Veškerá komunikace se serverem probíhá přes zabezpečené SSH spojení s využitím silného šifrování („Paramiko — Python SSH library“, 2024).

- **Autentizace pomocí klíčů** – Primárním mechanismem autentizace je SSH keypair, který poskytuje vyšší úroveň zabezpečení než hesla.
- **Autorizační mechanismy** – Implementace využívá existující autorizační mechanismy operačního systému, přičemž přístup je omezen pouze na autorizované uživatele (The Linux Foundation, 2004).
- **Zabezpečení ukládaných dat** – Data jsou ukládána s respektováním UNIX file permissions, s omezeními přístupu pouze pro autorizované uživatele.

### 3.4.2 Autentizace a autorizace uživatelů

Přístup k dashboardu je zabezpečen pomocí několika mechanismů:

- **Integrace s Lightweight Directory Access Protocol (LDAP)/Active Directory** – Autentizace uživatelů probíhá proti existujícímu adresářovému serveru VŠE, což zajišťuje konzistenci s ostatními systémy[4] („LDAP Authentication in Streamlit“, 2025). LDAP je protokol pro centrální správu uživatelských účtů, který umožňuje použít stejné přihlašovací údaje napříč různými systémy organizace. Třída `LDAPConnector` automaticky mapuje skupiny uživatelů z Active Directory na aplikační role, čímž zajišťuje centralizovanou správu přístupových práv (Wahl et al., 1997).
- **Rolově založená autorizace** – Uživatelé jsou rozděleni do rolí (administrátor, analytik, čtenář) s různými úrovněmi přístupu (Stallings, 2019). Role-Based Access Control (RBAC) model umožňuje granulární řízení přístupu k jednotlivým funkcionalitám dashboardu – například analytici mohou analyzovat data, ale nemají přístup k administračním funkcím. Metadata rolí definovaná v modulu `auth.py` zajišťují aplikaci principu minimálních oprávnění (Ferraiolo et al., 2001).
- **Session management** – Implementace bezpečného řízení sessions včetně automatického odhlášení po době nečinnosti („Session State API“, 2025). Session management zajišťuje, že přihlášený uživatel zůstane ověřený během práce s aplikací, ale současně automaticky ukončuje relaci po 1 hodině neaktivity pro prevenci před zneužitím opuštěné relace (session). Validace session probíhá při každém požadavku a zahrnuje kontrolu platnosti a autentičnosti session identifikátoru (OWASP, 2023).
- **Auditní logy** – Veškeré přístupy a akce v systému jsou zaznamenávány pro případnou forenzní analýzu (National Institute of Standards and Technology, 2006). Auditní logging umožňuje zpětně dohledat kdo, kdy a jakou akci v systému provedl. Logy jsou ukládány ve formátu JSON Lines (JSONL) pro snadné zpracování a analýzu, zahrnují časové značky, kategorie událostí a detaily o kontextu akce. Tento mechanismus je klíčový pro detekci bezpečnostních incidentů a splnění compliance požadavků.

[4] Tato autentizační metoda bude provedena až při případném produkčním nasazení. Modul je na tuto variantu připravený.

### 3.4.3 Prevence běžných bezpečnostních rizik

V implementaci byly adresovány běžné bezpečnostní rizika webových aplikací:

- **SQL Injekce** – Veškeré uživatelské vstupy jsou validovány a sanitizovány před použitím v SQL dotazech nebo systémových příkazech („Injection Prevention Cheat Sheet“, 2024). SQL Injekce je útok, při kterém útočník vkládá do vstupů škodlivý kód, který je následně vykonán systémem. Funkce pro sanitizaci v modulu `security_utils.py` odstraňují nebo escapují speciální znaky a validují formát vstupů, čímž brání například SQL injection útokům, které by mohly vést k neoprávněnému přístupu k databázi (Anley, 2002).
- **Cross-Site Scripting (XSS)** – Implementace využívá zabudované XSS ochrany frameworku Streamlit a dodatečné sanitizace pro dynamicky generovaný obsah („Cross Site Scripting (XSS) Prevention Cheat Sheet“, 2024). XSS je útok, při kterém útočník vkládá do webové stránky škodlivý JavaScript kód, který může ukrást session cookies nebo provádět akce jménem uživatele. Kontextová sanitizace output zajišťuje, že uživatelem poskytnutý obsah nemůže být interpretován jako spustitelný kód (Ibarra-Fiallos et al., 2021).
- **Clickjacking** – Aplikace implementuje správné HTTP hlavičky pro prevenci click jackingu (X-Frame-Options) („Clickjacking Defense Cheat Sheet“, 2024). Clickjacking je útok, při kterém útočník "schová"legitimní stránku pod průhlednou vrstvu a přesvědčí uživatele kliknout na místo, o kterém neví. HTTP hlavička X-Frame-Options: DENY zakazuje zobrazení stránky v jakémkoli frame-u nebo iframe-u, čímž tento útok znemožňuje. Content-Security-Policy hlavička navíc specifikuje povolené zdroje obsahu (Stone, 2010).
- **Cross-Site Request Forgery (CSRF)** – Všechny akce měnící stav systému jsou chráněny proti CSRF útokům pomocí tokenů („Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet“, 2024). CSRF je útok, při kterém útočník přiměje prohlížeč oběti odeslat nechtěný požadavek k cílové webové aplikaci. CSRF tokeny jsou náhodné hodnoty přiřazené každé session, které musí být předány s každým stavovým požadavkem – bez znalosti tohoto tokenu není možné platný požadavek sestavit (Barth et al., 2008).

### 3.4.4 Bezpečnostní aspekty provozního nasazení

Pro bezpečné provozní nasazení systému byly navrženy následující postupy:

- **Minimalizace povrchu útoku** – Systém je nasazen s minimálním počtem závislostí a služeb, čímž se snižuje povrch potenciálního útoku (Rauber & Rünger, 2007). Princip minimalizace spočívá v instalaci a spuštění pouze těch komponent, které jsou absolutně nezbytné pro fungování aplikace. Méně komponent znamená méně potenciálních zranitelností, které by útočník mohl využít. Kontejnerizace navíc zajišťuje izolaci aplikace od hostitelského systému (Security, 2023).

- **Hardening OS** – Operační systém, na kterém běží aplikace, je konfigurován podle best practices pro bezpečnostní hardening (The Linux Foundation, 2004). Hardening OS zahrnuje vypnutí nepotřebných služeb, konfiguraci firewallu pro omezení síťového provozu pouze na požadované porty, implementaci systémů jako SELinux nebo AppArmor pro dodatečné vrstvy ochrany, a konfiguraci bezpečného logování. Tyto opatření zmenšují útočné vektory a omezují škody v případě kompromitace (for Internet Security, 2023).
- **Monitoring a detekce anomálií** – Systém je monitorován pro detekci neobvyklého chování nebo potenciálních bezpečnostních incidentů („Security and Monitoring in the ELK Stack“, 2023). Monitoring zahrnuje kontinuální sledování auditních logů, síťového provozu a vzorců chování uživatelů.

## 3.5 Validace a testování řešení

Komplexní validace a testování byly nedílnou součástí vývoje systému pro analýzu logů. Cílem validace bylo ověřit správnost, efektivitu a robustnost implementovaného řešení v reálných podmínkách provozu VŠE. Proces validace zahrnoval:

- Ověření funkčních a nefunkčních požadavků.
- Testování výkonnosti algoritmu.
- Konzultace s pracovníky Centra informatiky.

### 3.5.1 Validace funkčních a nefunkčních požadavků

V rámci byznysového zadání byly definovány funkční a nefunkční požadavky, jejichž naplnění je klíčové pro úspěch projektu. Pro systematické vyhodnocení byla vytvořena metodika zahrnující testovací scénáře pro každý požadavek a jejich následnou verifikaci v reálném prostředí. Testování probíhalo v několika fázích:

- Unit testování jednotlivých komponent.
- Integrační testování modulů systému.
- Pilotní nasazení u dr. Šimečka, včetně akceptačního testování.

Výsledky validace byly zpracovány do přehledných tabulek, které jsou podrobně rozepsány v příloze A. Souhrnně lze konstatovat, že z celkových 22 definovaných požadavků bylo 18 (81,8 %) plně naplněno, 3 (13,6 %) částečně naplněno a pouze 1 (4,5 %) nebylo možné v rámci této práce ověřit. Tento výsledek potvrzuje úspěšné splnění stanovených cílů práce a poskytuje solidní základ pro další rozvoj systému.

Částečně naplněné požadavky se týkaly především:

- Automatické korelace událostí mezi různými systémy.
- Pokročilé vizualizace topologie sítě.

**Tabulka 3.3**
*Souhrnné hodnocení validace požadavků*

| Kategorie | Celkem | Naplněno | Částečně | Neověřeno |
|---|---|---|---|---|
| Funkční požadavky | 12 | 11 | 1 | 0 |
| Nefunkční požadavky | 10 | 7 | 2 | 1 |
| **Celkem** | **22** | **18** | **3** | **1** |
| **Procento** | **100 %** | **81,8 %** | **13,6 %** | **4,5 %** |

*Poznámka: Vlastní zpracování.*

- Integrace s některými proprietárními nástroji třetích stran.

Tyto funkce by mohly být implementovány v budoucích verzích systému. Jediný neověřený požadavek se týkal škálovatelnosti systému pro extrémní objemy dat (nad 100 TB), což nebylo možné v rámci této práce realisticky otestovat vzhledem k omezením testovacího prostředí.

### 3.5.2 Detailní validace požadavků

Pro transparentní vyhodnocení míry naplnění jednotlivých požadavků byla vytvořena podrobná analýza každého z nich. V tabulce 3.4 jsou uvedeny výsledky validace funkčních požadavků s přesným vyhodnocením míry naplnění a odůvodněním v případě částečného splnění.

Obdobně byla provedena i detailní validace nefunkčních požadavků, jejíž výsledky jsou shrnuty v tabulce 3.5. Zde je patrné, že většina nefunkčních požadavků byla také úspěšně naplněna.

### 3.5.3 Výsledky validace algoritmu

Pro objektivní vyhodnocení přesnosti algoritmu ENV byla navržena a implementována metodologie zahrnující vytvoření testovacího datasetu, ruční vytipování statických a dynamických částí logových záznamů a jejich automatickou analýzu algoritmem ENV. Proces validace zahrnoval:

- Přípravu reprezentativního vzorku logových záznamů z různých systémů.
- Manuální anotaci těchto záznamů zkušenými analytiky.
- Automatické zpracování stejného datasetu algoritmem ENV.
- Porovnání výsledků s jednodušším přístupem založeným pouze na regulárních výrazech.
- Výpočet metrik přesnosti a vyhodnocení výsledků.

Validace algoritmu ENV na testovacím datasetu ukázala vysokou přesnost identifikace statických a dynamických částí logů, s F1 skóre 96.0%. Toto představuje významné zlepšení oproti

**Tabulka 3.4**

*Detailní hodnocení funkčních požadavků*

| ID | Požadavek | Míra naplnění | Odůvodnění/Komentář |
|---|---|---|---|
| F1 | Automatická extrakce logů ze serveru | Zcela naplněno | Datová pumpa spolehlivě extrahuje logy ze všech dostupných zdrojů. |
| F2 | Identifikace statických a dynamických částí logů | Zcela naplněno | Algoritmus ENV vykazuje vysokou přesnost (F1 skóre 96,0%). |
| F3 | Frekvenční analýza výskytu vzorů | Zcela naplněno | Systém efektivně zpracovává a analyzuje frekvence výskytu. |
| F4 | Interaktivní vizualizace výsledků | Zcela naplněno | Dashboard poskytuje kompletní vizualizační nástroje. |
| F5 | Detekce anomálií v logových záznamech | Zcela naplněno | Implementovány statistické metody pro odhalení odchylek. |
| F6 | Filtrace a vyhledávání v logových datech | Zcela naplněno | Systém umožňuje efektivní filtraci podle různých kritérií. |
| F7 | Export výsledků ve standardních formátech | Zcela naplněno | Implementována podpora exportu do CSV, JSON a Portable Document Format (PDF). |
| F8 | Kategorizace vzorů podle typu a závažnosti | Zcela naplněno | Každý vzor je automaticky kategorizován dle definovaných pravidel. |
| F9 | Korelace událostí mezi různými systémy | Částečně naplněno | Základní korelace podle času a zdroje implementována, pokročilá sémantická korelace plánována v budoucnu. |
| F10 | Generování reportů o nalezených vzorech | Zcela naplněno | Systém umožňuje generování komplexních přehledových reportů. |
| F11 | Možnost definice vlastních pravidel pro analýzu | Zcela naplněno | Implementováno rozhraní pro tvorbu a správu uživatelských pravidel. |
| F12 | Archivace a správa historických dat | Zcela naplněno | Automatizovaný systém pro rotaci a archivaci dat je plně funkční. |

*Poznámka: Vlastní zpracování.*

**Tabulka 3.5**
*Detailní hodnocení nefunkčních požadavků*

| ID | Požadavek | Míra naplnění | Odůvodnění/Komentář |
|---|---|---|---|
| N1 | Zpracování měsíčního objemu logů do 1 hodiny | Zcela naplněno | Systém zpracovává měsíční objem logů za 15 minut. |
| N2 | Škálovatelnost pro více než 2000 zdrojů logů | Zcela naplněno | Validovaná podpora pro všech 2200 zdrojů. |
| N3 | Intuitivní uživatelské rozhraní | Zcela naplněno | Testování potvrdilo vysokou použitelnost i pro méně zkušené uživatele. |
| N4 | Odolnost vůči výpadkům zdroje dat | Zcela naplněno | Implementovány retry mechanismy a persistentní fronty. |
| N5 | Podpora pro různé formáty logů | Zcela naplněno | Systém efektivně zpracovává všechny běžné formáty logů v prostředí VŠE. |
| N6 | Škálovatelnost pro objemy dat nad 100 TB | Neověřeno | Vzhledem k omezením testovacího prostředí nebylo možné realisticky ověřit. |
| N7 | Zabezpečený přístup k citlivým datům | Zcela naplněno | Implementováno víceúrovňové zabezpečení dle standardů VŠE. |
| N8 | Minimální latence zpracování dat (pod 30 minut) | Zcela naplněno | Průměrná latence zpracování je 15 minut. |
| N9 | Vizualizace topologie síťových zařízení | Částečně naplněno | Základní vizualizace implementována, pokročilé funkce plánované pro další verze. |
| N10 | Integrace s existujícími monitorovacími systémy | Částečně naplněno | Úspěšná integrace s hlavními nástroji, některé proprietární systémy vyžadují další vývoj. |

*Poznámka: Vlastní zpracování.*

baseline založené na regulárních výrazech, která dosáhla F1 skóre pouze 79.1%. Přesnost (precision) algoritmu ENV dosáhla 97.3%, což znamená, že téměř všechny části identifikované jako dynamické skutečně dynamické byly. Úplnost (recall) dosáhla 94.8%, což indikuje, že algoritmus zachytil většinu skutečně dynamických částí v logových záznamech.

**Tabulka 3.6**

*Výsledky validace algoritmu ENV na testovacím datasetu*

| **Metrika** | **ENV Algoritmus** | **Regex baseline** | **Zlepšení** |
|---|---|---|---|
| Přesnost (precision) | 97.3% | 82.1% | 18.5% |
| Úplnost (recall) | 94.8% | 76.4% | 24.1% |
| F1 skóre | 96.0% | 79.1% | 21.4% |
| Falešně pozitivní rate | 2.7% | 17.9% | 84.9% |
| Falešně negativní rate | 5.2% | 23.6% | 78.0% |

*Poznámka: Vlastní zpracování.*

Hodnoty zlepšení v tabulce 3.6 jsou vypočítány jako relativní změny mezi hodnotami ENV algoritmu a baseline založené na regulárních výrazech. Pro metriky jako přesnost, úplnost a F1 skóre, kde vyšší hodnoty představují lepší výkon, je zlepšení počítáno jako relativní nárůst (např. pro přesnost: $\frac{97.3\%-82.1\%}{82.1\%} \cdot 100\% = 18.5\%$).

Pro metriky falešně pozitivní a falešně negativní rate, kde nižší hodnoty jsou lepší, je zlepšení počítáno jako relativní snížení chybovosti:

- Falešně pozitivní rate: $\frac{17.9\%-2.7\%}{17.9\%} \cdot 100\% = 84.9\%$ - což indikuje, že algoritmus ENV redukoval falešně pozitivní detekce o téměř 85% oproti standardnímu přístupu pomocí regulárních výrazů.
- Falešně negativní rate: $\frac{23.6\%-5.2\%}{23.6\%} \cdot 100\% = 78.0\%$ - což znamená, že algoritmus ENV zachytil o 78% více skutečně dynamických částí, které by regulární výrazy nesprávně označily jako statické.

Toto relativní vyjádření zlepšení poskytuje přesnější obraz o skutečném dopadu algoritmu ENV ve srovnání s absolutními rozdíly, zvláště u metrik s nízkou základní hodnotou, jako jsou míry chyb.

Důležitým aspektem validace byla také konzultace s dr. Šimečkem, vedoucím Kybernetické bezpečnosti na VŠE, který je zároveň jediným pracovníkem pravidelně pracujícím s analýzou logů v prostředí VŠE. Jeho pohled na funkcionalitu a uživatelské rozhraní byl vzhledem k této skutečnosti naprosto klíčový. Dr. Šimeček potvrdil:

- Praktickou použitelnost nástroje v reálných podmínkách provozu VŠE.
- Schopnost efektivně identifikovat vzory v heterogenních logovacích datech.
- Intuitivnost vizualizace usnadňující interpretaci výsledků.
- Významný přínos pro zefektivnění jeho každodenní práce.

### 3.5.4 Provozní aspekty nasazení

Pro úspěšné provozní nasazení nástroje byly zpracovány postupy a doporučení zaměřené na dlouhodobou udržitelnost a spolehlivost. Tyto postupy zahrnují:

- Strategii zálohování databáze vzorů a konfiguračních souborů.
- Integraci s existujícím monitorovacím systémem VŠE pro sledování dostupnosti a výkonu.
- Definici postupů pro rychlou obnovu systému v případě výpadku nebo poruchy.

Důležitou součástí provozních aspektů je také návrh postupů pro horizontální a vertikální škálování systému při růstu objemu dat. To zahrnuje definici hraničních hodnot výkonu, při jejichž překročení je třeba systém rozšířit, i konkrétní kroky pro toto rozšíření, jako je navýšení počtu pracovních uzlů nebo optimalizace algoritmů.

V neposlední řadě byly implementovány pravidla pro archivaci a mazání starých dat, která zajišťují efektivní využití úložného prostoru a současně respektují legislativní požadavky na uchovávání bezpečnostních logů. Tato pravidla jsou nastavena tak, aby se automaticky aplikovala bez nutnosti manuálního zásahu, což dále snižuje provozní náročnost systému.

Tyto postupy společně zajišťují, že systém bude dlouhodobě udržitelný a odolný vůči běžným provozním problémům, což je klíčové pro jeho efektivní využití v produkčním prostředí VŠE.

## 3.6 Limity práce a možnosti pokračování

### 3.6.1 Limity práce

Přestože implementované řešení přineslo významné zlepšení oproti původnímu stavu, je důležité reflektovat také jeho limity, které mohou ovlivnit jeho uplatnitelnost v některých kontextech. Tyto limity nepředstavují fundamentální nedostatky, ale spíše oblasti, které by mohly být adresovány v budoucím vývoji.

Prvním významným limitem je omezená škálovatelnost pro extrémní objemy dat. Ačkoliv řešení efektivně zpracovává současné objemy dat generované infrastrukturou VŠE, při škálování nad 100+ TB by mohly nastat výkonnostní problémy. Toto omezení je dáno především:

- Zvolenou architekturou založenou na multiprocessingu v rámci jednoho stroje.
- Absencí nativní podpory distribuovaného zpracování.
- Limitacemi používaných knihoven při práci s extrémně velkými datovými sadami.

Dalším limitem je absence zpracování v reálném čase. Současné řešení pracuje v batch módu s minimální prodlevou 15 minut, což může být limitující pro detekci některých typů útoků, kde je kritická okamžitá reakce. Toto omezení vyplývá z technologických voleb učiněných v rámci implementace, které optimalizují propustnost na úkor latence.

Implementované řešení má také omezenou podporu pro nestandardní formáty logů. Přestože algoritmus ENV je navržen s důrazem na flexibilitu a schopnost zpracovat heterogenní logová data, některé vysoce specializované nebo nestandardní formáty mohou vyžadovat dodatečnou konfiguraci nebo úpravy algoritmu.

Efektivita celého systému je také silně závislá na kvalitě vstupních dat. V prostředích, kde chybí standardizace logování nebo kde jsou logy nekompletní či nekonsistentní, může být přínos systému omezen. Toto není přímo limitem implementace, ale spíše faktorem, který je třeba zvážit při nasazení systému v různých prostředích.

V neposlední řadě je třeba zmínit, že i přes uživatelsky přívětivý dashboard vyžaduje efektivní využití systému pro pokročilou analýzu určitou míru expertní znalosti. To může limitovat jeho využití méně technicky zaměřenými uživateli, zejména v kontextech, kde není k dispozici specializovaný bezpečnostní analytik.

#### Analýza škálovatelnosti implementovaného řešení

Otázka škálovatelnosti implementovaného řešení si zaslouží hlubší analýzu. Současná architektura je optimalizována pro aktuální objem dat generovaný infrastrukturou VŠE a úspěšně zvládá zpracovat data z přibližně 2 200 zdrojů v rozumném časovém rámci. Tato architektura je založena na multiprocessingu v rámci jednoho stroje, což přináší významné výkonnostní výhody oproti sekvenčnímu zpracování, ale má své inherentní limity.

Při testování bylo zjištěno, že:

- Systém vykazuje téměř lineární škálování až do přibližně 16 jader procesoru.
- Po překročení 16 jader začíná efektivita paralelizace klesat.
- Tento jev je v souladu s Amdahlovým zákonem a naznačuje limity současného návrhu.

Pro objemy dat v řádu desítek TB by současné řešení ještě mohlo být adaptováno optimalizací algoritmů a efektivnějším využitím paměti, ale pro větší objemy by byl nutný zásadnější přechod na distribuovanou architekturu.

Ze srovnání s průmyslovými standardy vyplývá, že implementované řešení dosahuje přibližně 70% výkonu systému ELK Stack při indexaci a vyhledávání vzorů, ale za cenu výrazně nižších hardwarových nároků. Ve srovnání se Splunkem dosahuje přibližně 55% výkonu při zpracování stejného objemu dat, avšak s výhodou absence licenčních poplatků, které by v případě Splunku při tomto objemu dat představovaly značnou finanční zátěž.

### 3.6.2 Možnosti pokračování v tématu

Na základě výsledků této práce a s ohledem na identifikované limity se nabízí několik směrů, kterými by bylo možné v tématu pokračovat. Tyto směry nejen adresují identifikované limity

současného řešení, ale také reflektují aktuální trendy v oblasti kybernetické bezpečnosti a analýzy dat.

### Zpracování logů v reálném čase

Perspektivní oblastí pro navazující výzkum je implementace streamovacího zpracování logů pomocí technologií jako Apache Kafka nebo Apache Flink („Apache Flink Documentation“, 2025; „Apache Kafka Documentation“, 2025). S vydáním Apache Flink 2.0.0 přišla řada nových funkcí pro zpracování streamů dat, které výrazně zjednodušují implementaci real-time analýzy logů a nabízejí pokročilé možnosti pro zpracování událostí bez nutnosti ukládání mezivýsledků („Apache Flink Documentation“, 2025).

Implementace streamovacího přístupu by umožnila zpracování logů v reálném čase s minimální latencí, což je zásadní pro včasnou detekci bezpečnostních incidentů. Výrazně by se zlepšila schopnost systému detekovat a reagovat na bezpečnostní incidenty ihned po jejich vzniku, což může významně redukovat čas od průniku do systému k jeho detekci. Současně by bylo možné implementovat komplexní analytické operace nad proudovými daty, jako jsou klouzavá okna pro detekci vzorů chování v časových řadách.

Takový posun by vyžadoval architektonické změny v systému, přechod z batch orientovaného zpracování na event-driven architekturu a implementaci perzistentních front pro zajištění odolnosti proti výpadkům. Výhodou by byla také možnost škálovat jednotlivé komponenty nezávisle podle aktuální zátěže, což by umožnilo efektivněji využívat dostupné výpočetní prostředky a přizpůsobovat se aktuálním potřebám analýzy.

### Korelace událostí z různých zdrojů

Další slibnou oblastí výzkumu je vývoj a testování algoritmů pro automatickou korelaci událostí z různých zdrojů (Ambre & Shekokar, 2015). Inovativní přístupy jako Log2graphs nabízejí nové možnosti pro efektivní extrakci vlastností a detekci anomálií v logových datech pomocí grafových reprezentací (Wang et al., 2024). Schopnost identifikovat souvislosti mezi zdánlivě nesouvisejícími událostmi z různých systémů by významně zlepšila schopnost detekce komplexních útoků a omezila počet falešně pozitivních detekcí.

Moderní kybernetické útoky často zahrnují složité sekvence aktivit napříč různými systémy, které jsou v izolovaném pohledu obtížně identifikovatelné jako škodlivé. Implementace grafových algoritmů pro identifikaci souvislostí mezi událostmi by umožnila odhalit tyto vzorce na základě atributů jako časové značky, IP adresy, uživatelské účty nebo systémové identifikátory. Vývoj časově-kauzálních modelů pro analýzu posloupnosti událostí by pak mohl vést k identifikaci vzorů útočných sekvencí ještě před jejich dokončením.

Pro efektivní implementaci těchto přístupů by bylo vhodné zvážit využití specializovaných databází orientovaných na grafy jako Neo4j nebo Amazon Neptune, které poskytují nativní

podporu pro grafové dotazy a algoritmy. Zvláště přístup navrhovaný v Log2graphs (Wang et al., 2024), který kombinuje grafové reprezentace s neuronovými sítěmi, představuje slibnou cestu k identifikaci komplexních vzorů útoků s minimálním počtem falešných poplachů.

### Aplikace metod strojového učení

Využití technik strojového učení představuje další perspektivní směr výzkumu (Yu et al., 2024). Nejnovější výzkum, jako například LogCraft, představuje end-to-end Automated Machine Learning (AutoML) přístupy pro detekci anomálií v logových datech, které výrazně zjednodušují nasazení pokročilých Machine Learning (ML) technik v této oblasti (Zhang et al., 2024).

Tradiční přístupy k analýze logů jsou založeny na předem definovaných pravidlech, která nedokáží efektivně reagovat na nové typy útoků nebo neobvyklé vzorce chování. Metody strojového učení, zejména učení bez učitele a detekce anomálií, mohou tento problém překonat automatickou identifikací podezřelých vzorců v logových datech bez nutnosti explicitní definice pravidel. To by významně rozšířilo schopnost detekce neznámých útoků a adaptivně se přizpůsobovat měnícím se vzorcům normálního chování systémů.

Zvláště slibné jsou přístupy využívající rekurentní neuronové sítě nebo transformery pro modelování sekvenčních dat, které dokáží zachytit závislosti mezi vzdálenými událostmi v logu a identifikovat anomálie v těchto sekvencích. LogCraft (Zhang et al., 2024) navíc představuje inovativní přístup k automatickému hledání optimální architektury modelu a hyperparametrů pomocí AutoML technik, což výrazně snižuje potřebu expertní znalosti při nasazování těchto pokročilých metod.

Výzvou v této oblasti je především adaptace obecných modelů strojového učení na specifika logových dat, která jsou často nestrukturovaná a vysoce heterogenní. Proto by další výzkum měl zahrnovat:

- Vývoj specializovaných předzpracování pro logová data, která zachovávají sémantické vazby mezi jednotlivými částmi logu
- Vytvoření přístupů schopných pracovat i s omezeným množstvím trénovacích dat, což je běžný problém v oblasti kybernetické bezpečnosti
- Interpretovatelné modely, které dokáží nejen detekovat anomálie, ale také poskytnout srozumitelné vysvětlení pro bezpečnostní analytiky

### Integrace doplňkových datových zdrojů

Výzkum by se mohl také zaměřit na integraci dalších typů dat, jako jsou NetFlow záznamy, data z bezpečnostních sond nebo informace o zranitelnostech (Komisarek et al., 2024). Tato integrace by poskytla komplexnější pohled na bezpečnostní stav infrastruktury a umožnila sofistikovanější analýzu potenciálních hrozeb.

Současný systém je zaměřen primárně na analýzu logových záznamů, což poskytuje cenné informace, ale pouze z jednoho pohledu. Bezpečnostní analýza je však multidimenzionální problém, který vyžaduje syntézu informací z různých zdrojů pro přesnou identifikaci a charakterizaci bezpečnostních hrozeb. Implementace mechanismů pro konsolidaci a normalizaci dat z heterogenních zdrojů do jednotného datového modelu by umožnila vytvořit složené ukazatele rizika kombinující informace z různých typů dat.

Zvláště přínosná by byla integrace informací o zranitelnostech z externích zdrojů, jako jsou NIST National Vulnerability Database (NVD) nebo Common Vulnerabilities and Exposures (CVE) databáze, pro kontextualizaci detekovaných událostí. Stejně tak začlenění dat o topologii sítě a závislostí mezi systémy by poskytlo lepší pochopení potenciálního dopadu detekovaných anomálií a umožnilo by implementovat mechanismy pro prioritizaci událostí na základě kritičnosti systémů.

Tyto rozšíření by posunuly systém od pouhé analýzy logů směrem ke komplexnímu bezpečnostnímu informačnímu a event management (SIEM) řešení, které by poskytovalo holistický pohled na bezpečnostní stav infrastruktury VŠE.

#### Distribuované zpracování pro zlepšení škálovatelnosti

Pro adresování limitů škálovatelnosti by byl přínosem výzkum v oblasti distribuovaného zpracování s využitím technologií jako Apache Spark nebo Hadoop (Dean & Ghemawat, 2008; Desai & Goel, 2025). V kontextu kybernetické bezpečnosti je klíčové implementovat také odpovídající bezpečnostní mechanismy pro ochranu samotného analytického prostředí, jak zdůrazňuje výzkum v oblasti bezpečnosti Apache Spark (Mărcuță & Team, 2025).

Současný systém vykazuje dobré výkonnostní charakteristiky pro stávající objem dat, ale s růstem infrastruktury VŠE a zvyšujícím se množstvím logů bude nutné implementovat škálovatelnější řešení. Distribuované zpracování by umožnilo efektivní analýzu ještě větších objemů dat a poskytlo základ pro horizontální škálování systému. Implementace distribuovaného úložiště dat s podporou pro horizontální škálování, například založeného na Hadoop Distributed File System (HDFS) nebo objektových úložištích, by byla základním stavebním kamenem takového řešení.

Výzkum v této oblasti by měl zahrnovat také využití efektivnějších formátů pro ukládání velkých objemů dat, jako je Hierarchical Data Format version 5 (HDF5) (Broecker et al., 2024), a přepracování algoritmu ENV pro efektivní paralelní zpracování v distribuovaném prostředí. Klíčovou výzvou je optimalizace distribuované strategie zpracování s ohledem na lokalitu dat, která má zásadní vliv na výkon v distribuovaných systémech.

Z bezpečnostního hlediska je nezbytné implementovat i odpovídající ochranné mechanismy, jak zdůrazňuje výzkum v oblasti bezpečnostních praktik pro Apache Spark (Mărcuță & Team, 2025):

- Šifrování dat v klidu i za pohybu pro ochranu citlivých bezpečnostních informací
- Implementace silné autentizace a autorizace pro přístup k analytickým nástrojům
- Oddělení výpočetních prostředí podle citlivosti zpracovávaných dat

Tyto vylepšení by umožnily systému efektivně zpracovávat objemy dat v řádu stovek TB, což by pokrylo i budoucí růst infrastruktury VŠE a potenciálně umožnilo nasazení systému i v jiných, větších organizacích.

#### Automatizace bezpečnostních operací

V neposlední řadě představuje zajímavý směr výzkumu návrh a implementace workflow pro automatizované reakce na detekované incidenty (Living Security, 2024). Automatizované pracovní postupy mohou výrazně zrychlit reakci na bezpečnostní incidenty a minimalizovat dopad na organizaci, což je kritické zvláště v prostředích s omezenými personálními kapacitami, jako je právě VŠE.

Současný systém umožňuje detekci potenciálních bezpečnostních incidentů, ale reakce na tyto incidenty je stále převážně manuální proces. Implementace automatizovaných reakcí by mohla výrazně snížit čas od detekce incidentu k jeho mitigaci. Definice formálního jazyka pro popis automatizovaných bezpečnostních postupů (playbooks) by umožnila vytvoření knihovny standardizovaných reakcí pro různé typy incidentů, které by bylo možné dynamicky upravovat podle aktuálních potřeb.

Výzkum v této oblasti by měl zahrnovat také implementaci rozhodovacího systému pro výběr vhodných reakcí na základě typu a závažnosti detekovaných událostí a vytvoření bezpečného izolovaného prostředí pro testování automatizovaných reakcí před jejich nasazením v produkčním prostředí. Integrace s existujícími nástroji pro správu IT infrastruktury a bezpečnostními systémy by zajistila konzistentní reakci napříč celým prostředím.

Jak uvádí studie Living Security (Living Security, 2024), využití externích integrací se Security Orchestration, Automation and Response (SOAR) platformami může výrazně rozšířit možnosti automatizace a umožnit komplexnější reakce na bezpečnostní incidenty. Implementace zpětnovazebních mechanismů pro hodnocení účinnosti automatizovaných reakcí by pak umožnila jejich kontinuální vylepšování a adaptaci na měnící se bezpečnostní hrozby.

#### Interaktivní vizualizace a explorativní analýza

Dalším možným směrem výzkumu by bylo zdokonalení interaktivních vizualizací pro efektivnější explorativní analýzu logů. Současný dashboard poskytuje základní vizualizační nástroje, ale pokročilejší techniky by mohly výrazně zlepšit schopnost analytiků identifikovat vzorce a anomálie v datech.

Vývoj pokročilých vizualizačních technik specificky zaměřených na logová data by zahrnoval

implementaci časových os událostí s různými úrovněmi granularity, které by umožnily rychlý přechod mezi souhrnným pohledem a detailní analýzou konkrétních časových období. Interaktivní nástroje pro vizuální dotazování nad daty by poskytly možnost dynamického filtrování a seskupování podle různých atributů, což by usnadnilo identifikaci vzorců v datech.

Zvláště přínosná by byla vizuální reprezentace síťových toků a souvislostí mezi událostmi v podobě interaktivních grafů, které by umožnily intuitivní navigaci ve složitých vztazích mezi událostmi. Pro práci s velkými objemy dat by bylo nezbytné implementovat techniky pro vizualizaci s podporou pro automatickou agregaci a filtrování podle kontextu, aby byla zajištěna responzivita i při analýze rozsáhlých datasetů.

Integrace přirozeného jazyka pro zadávání dotazů nad logovými daty by usnadnila práci i méně technicky zaměřeným uživatelům a rozšířila by potenciální uživatelskou základnu systému. Takový přístup by kombinoval sílu složitých dotazovacích jazyků s intuitivností běžné komunikace, což by vedlo k vyšší efektivitě analytiků při vyšetřování bezpečnostních incidentů.

## 3.7 Shrnutí implementace řešení

V této kapitole byla popsána implementace systému pro efektivní zpracování a analýzu logů v prostředí VŠE. Systém používá moderní architektonické vzory a principy, které zajišťují jeho rozšiřitelnost, udržitelnost a robustnost. Klíčové komponenty zahrnují:

- Datovou pumpu pro extrakci logů ze serveru.
- Algoritmus Extended Nagappan-Vouk pro identifikaci vzorů v logových záznamech.
- Interaktivní dashboard pro vizualizaci a analýzu výsledků.

Implementované řešení přináší významné zlepšení oproti původním přístupům, zejména v oblasti výkonu a škálovatelnosti. Díky paralelnímu zpracování a optimalizaci ukládání dat se podařilo:

- Zkrátit dobu zpracování měsíčního objemu logů z 12 hodin na pouhých 15 minut, což představuje zrychlení o faktor 48.
- Redukovat velikost souborů o 70-85% díky využití sloupcově orientovaného formátu Parquet s horizontálním rozdělením.
- Dosáhnout až 90% zrychlení načítání dat pro analýzu.

Tato zlepšení jsou obzvláště významná v kontextu prostředí VŠE, kde za kybernetickou bezpečnost a analýzu logů odpovídá jediný pracovník. Dramatické zkrácení doby zpracování znamená, že tento pracovník nyní může efektivně monitorovat mnohem větší část infrastruktury než dříve, což přímo přispívá ke zvýšení bezpečnosti celého prostředí.

Validace algoritmu ENV prokázala jeho vysokou přesnost při identifikaci statických a dynamických částí logů, s F1 skóre 96.0%, což je výrazné zlepšení oproti tradičním přístupům

založeným na regulárních výrazech. Implementované řešení bylo důkladně testováno a validováno ve spolupráci s dr. Šimečkem, který je primárním a v podstatě jediným uživatelem systému, což zajistilo, že finální řešení přesně odpovídá reálným potřebám a pracovním postupům.

Významnou součástí řešení je také interaktivní dashboard implementovaný v knihovně Streamlit, který poskytuje:

- Intuitivní uživatelské rozhraní pro analýzu a vizualizaci identifikovaných vzorů.
- Pokročilé filtrační a vyhledávací možnosti.
- Customizované vizualizace pro efektivní analýzu logů.
- Flexibilní možnosti exportu a reportování.

Volba Streamlit namísto komerčních nástrojů jako Power BI přinesla řadu výhod, včetně lepší integrace s Python ekosystémem, absence licenčních omezení a flexibilnější možnosti customizace. Tento dashboard byl navržen s důrazem na efektivitu práce jediného analytika, který musí rychle zpracovávat a interpretovat velké množství dat.

Implementované řešení významně přispívá k posílení kybernetické bezpečnosti VŠE tím, že umožňuje efektivnější identifikaci potenciálních hrozeb a incidentů, a to navzdory omezeným personálním kapacitám. Zároveň poskytuje cenné informace pro proaktivní správu infrastruktury a plnění legislativních požadavků v oblasti kybernetické bezpečnosti.

# Závěr

Tato diplomová práce byla zaměřena na problematiku automatizovaného zpracování a analýzy logů v prostředí Vysoké školy ekonomické v Praze. Hlavním cílem bylo vyvinout komplexní nástroj pro efektivní zpracování logových souborů ze vzdáleného serveru, který by automatizovaným způsobem určoval sémantiku logů, identifikoval frekvenci slov a rozlišoval mezi statickými kořeny logu a proměnnými částmi. Současně s tím měl být vytvořen uživatelsky přívětivý dashboard, který by umožnil pracovníkům Centra informatiky snadnou interpretaci výsledků pro identifikaci příčin či zdrojů kybernetických incidentů.

Při zahájení práce byla situace na VŠE charakterizována fragmentovaným přístupem k analýze logů, kdy z celkového počtu přesahujícího 2 200 zdrojů bylo aktivně zpracováváno pouze přibližně 30. Zbytek logů byl sice ukládán na centrálním serveru Levandule, ale nebyl systematicky analyzován, což vedlo k potenciálnímu přehlížení bezpečnostních incidentů a neefektivnímu využívání dostupných dat.

## Naplnění cílů práce

Hlavní cíl práce - vyvinout komplexní nástroj pro efektivní zpracování logů ze vzdáleného serveru - byl úspěšně naplněn. Implementovaný systém automatizovaně určuje sémantiku logů pomocí frekvenční analýzy, identifikuje statické a dynamické části logových záznamů a poskytuje rozhraní pro jejich analýzu. Výsledné řešení umožňuje zpracování všech 2 200 zdrojů logů, což představuje dramatické rozšíření oproti původním 30 zdrojům analyzovaným v LOGmanageru. Implementace paralelního zpracování a optimalizovaných datových struktur vedla k výraznému zkrácení doby analýzy, což umožňuje pracovníkům Centra informatiky věnovat více času strategickým aktivitám namísto manuálního procházení logů.

V úvodu práce byly také stanoveny tři dílčí cíle, které se podařilo úspěšně naplnit. První dílčí cíl - analýza stávajícího technického a legislativního prostředí VŠE - byl realizován prostřednictvím důkladného rozboru zákonů o kybernetické bezpečnosti a souvisejících vyhlášek, společně s analýzou současné infrastruktury pro sběr a zpracování logů. Tato analýza odhalila klíčové nedostatky stávajícího řešení a definovala požadavky na nový systém.

Druhý dílčí cíl - provedení sémantické analýzy logů s využitím algoritmu Extended Nagappan-Vouk - byl úspěšně implementován ve formě modulu pro frekvenční analýzu. Tento algoritmus, s dosaženým F1 skóre 96,0%, prokázal vysokou přesnost při identifikaci statických a dynamických částí logů, což je klíčové pro automatizovanou kategorizaci a analýzu logových záznamů.

Třetí dílčí cíl - vytvoření metodiky pro kontinuální sběr a analýzu logů - byl naplněn návrhem a implementací modulárního systému, který zahrnuje datovou pumpu, frekvenční analýzu a vizualizační komponenty. Systém je navržen s důrazem na rozšiřitelnost, udržitelnost a

spolehlivost, což umožňuje jeho dlouhodobé využívání v produkčním prostředí.

## Shrnutí výsledků práce

Implementované řešení přineslo několik významných zlepšení oproti původnímu stavu. Díky optimalizacím založeným na paralelním zpracování a efektivním ukládání dat se podařilo zkrátit dobu analýzy měsíčního objemu logů z původních 12 hodin na pouhých 15 minut, což představuje 48násobné zrychlení. Tento významný pokrok byl dosažen především implementací multiprocessingu, který umožnil současné zpracování dat z různých zdrojů s optimálním využitím dostupných hardwarových prostředků.

Implementace sloupcově orientovaného formátu Apache Parquet s horizontálním rozdělením dat přinesla 70-85 % redukci velikosti souborů oproti tradičnímu CSV formátu. Současně došlo až k 90 % zrychlení načítání dat pro analýzu a až 95% zrychlení při filtrování řádků. Tyto optimalizace byly klíčové pro zajištění efektivního přístupu k datům v kontextu interaktivní vizualizace.

Validace algoritmu ENV prokázala jeho vysokou přesnost při identifikaci statických a dynamických částí logů, s dosaženým F1 skóre 96,0%. Toto představuje výrazné zlepšení oproti tradičním přístupům založeným na regulárních výrazech, které dosáhly pouze 79,1% F1 skóre. Algoritmus ENV tak poskytuje spolehlivý základ pro automatizovanou analýzu heterogenních logových dat.

Vyvinutý interaktivní dashboard založený na knihovně Streamlit umožňuje intuitivní průzkum a analýzu logových dat bez nutnosti rozsáhlého technického školení. Uživatelské testování potvrdilo výrazné zlepšení v několika metrikách použitelnosti, včetně zkrácení času potřebného k dokončení typických úkolů a snížení počtu chyb.

## Praktické přínosy

Implementovaný systém přináší několik praktických přínosů pro Centrum informatiky VŠE. Především umožňuje efektivnější identifikaci bezpečnostních incidentů, jako jsou neobvyklé přihlašovací pokusy, skenování a průzkumné aktivity, útoky hrubou silou, neoprávněný přístup k souborům a neautorizované změny konfigurace. Systém také poskytuje cenné informace o celkovém stavu infrastruktury, včetně identifikace problémových systémů, detekce výkonnostních problémů a monitorování dostupnosti.

Automatizace procesu analýzy logů vede k významné úspoře času a umožňuje pracovníkům Centra informatiky soustředit se na strategické aktivity místo manuálního procházení logů. Systém také umožňuje postupné zapojení většiny z více než 2 200 zdrojů logů, což výrazně zlepšuje schopnost detekce potenciálních bezpečnostních incidentů a poskytuje ucelenější pohled na dění v síti.

## Možnosti dalšího rozvoje

Na základě výsledků této práce a identifikovaných limitů se nabízí několik směrů pro další rozvoj. Perspektivní oblastí výzkumu je implementace streamovacího zpracování pomocí technologií jako Apache Kafka nebo Apache Flink, což by umožnilo analýzu logů v reálném čase a snížilo latenci detekce bezpečnostních incidentů. Dalším směrem je vývoj algoritmů pro automatickou korelaci událostí z různých zdrojů, což by přispělo k odhalování komplexnějších vzorců útoků a omezení falešně pozitivních detekcí.

Využití technik strojového učení, zejména učení bez učitele a detekce anomálií, představuje další slibnou oblast pro automatickou identifikaci podezřelých vzorců v logových datech bez nutnosti explicitní definice pravidel. Pro adresování současných škálovacích omezení by byl přínosem výzkum v oblasti distribuovaného zpracování s využitím technologií jako Apache Spark nebo Hadoop, což by umožnilo efektivní analýzu ještě větších objemů dat.

Celkově lze konstatovat, že v kontextu rostoucího významu kybernetické bezpečnosti a potřeby efektivního zpracování stále většího objemu dat představuje tato práce relevantní příspěvek jak pro akademickou sféru, tak pro praxi. Díky modulární architektuře a důrazu na rozšiřitelnost je systém připraven adaptovat se na měnící se požadavky a technologické trendy v oblasti kybernetické bezpečnosti a analýzy dat.

# Použitá literatura


Ambre, A., & Shekokar, N. (2015). Insider threat detection using log analysis and event correlation. *Procedia Computer Science*, *45*, 436–445.

Amdahl, G. M. (1967). Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities. *AFIPS Conference Proceedings*, *30*, 483–485.

Anley, C. (2002). *Advanced SQL Injection in SQL Server Applications*. NGSSoftware Insight Security Research. https://www.ngssoftware.com/papers/advanced_sql_injection.pdf

*Apache Flink Documentation*. (2025). Apache Software Foundation. Získáno 26. dubna 2025, z https://nightlies.apache.org/flink/flink-docs-release-1.20/

*Apache Kafka Documentation*. (2025). Apache Software Foundation. Získáno 26. dubna 2025, z https://kafka.apache.org/documentation/

*Apache Parquet Documentation*. (2025). Apache Software Foundation. Získáno 26. dubna 2025, z https://parquet.apache.org/docs/

Armbrust, M., Xin, R. S., Lian, P., Huai, Y., Liu, D., Meng, X., Kaftan, T., Franklin, M. J., Ghodsi, A., & Zaharia, M. (2015). Spark SQL: Relational Data Processing in Spark. *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, 1383–1394. https://doi.org/10.1145/2723372.2742797

Barth, A., Jackson, C., & Mitchell, J. C. (2008). Robust defenses for cross-site request forgery. *Proceedings of the 15th ACM conference on Computer and communications security*, 75–88.

Berthold, M. R., et al. (2010). KNIME — the Konstanz Information Miner. *Data Analysis, Machine Learning and Applications*, *11*, 319–326.

Bockermann, C. (2014). A visual programming approach to big data analytics, 393–404.

Broecker, M. A., Siggel, M., & Reitenbach, S. (2024). Integration of large data based on HDF5 in a collaborative and multidisciplinary design environment, Part A: Methodology and Implementation. *AIAA SciTech 2024 Forum*, 0482.

*Centrum informatiky – Základní informace*. (2025). Centrum informatiky VŠE. Získáno 26. dubna 2025, z https://ci.vse.cz/o-centru/zakladni-informace/

*Clickjacking Defense Cheat Sheet*. (2024). OWASP. Získáno 26. dubna 2025, z https://cheatsheetseries.owasp.org/cheatsheets/Clickjacking_Defense_Cheat_Sheet.html

*codecs — Codec registry and base classes*. (2024). Python Software Foundation. Získáno 26. dubna 2025, z https://docs.python.org/3/library/codecs.html

*Cross Site Scripting (XSS) Prevention Cheat Sheet*. (2024). OWASP. Získáno 26. dubna 2025, z https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

*Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet*. (2024). OWASP. Získáno 26. dubna 2025, z https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html

David Akande, T., Kaur, B., Dadkhah, S., & Ghorbani, A. A. (2022). Threshold based technique to detect anomalies using log files, 191–198.

Dean, J., & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. *Communications of the ACM*, *51*(1), 107–113. https://doi.org/10.1145/1327452.1327492

Desai, P. B., & Goel, O. (2025). Scalable Data Pipelines for Enterprise Data Analytics. *International Journal of Research in All Subjects in Multi Languages*, *13*(1), 174.

Drepper, U. (Listopadu 2007). What Every Programmer Should Know About Memory [Technical report]. *Red Hat, Inc.*, 1–114. https://people.freebsd.org/~lstewart/articles/cpumemory.pdf

*Elasticsearch: Getting Started*. (2025). Elastic. Získáno 26. dubna 2025, z https://www.elastic.co/guide/en/elasticsearch/reference/current/getting-started.html

*European Union Agency for Cybersecurity – Guidelines on Network and Information Security*. (2022). ENISA. Získáno 26. dubna 2025, z https://www.enisa.europa.eu/

Ferraiolo, D. F., Sandhu, R., Gavrila, S., Kuhn, D. R., & Chandramouli, R. (2001). Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security*, *4*(3), 224–274.

for Internet Security, C. (2023). *CIS Benchmarks for Secure Configuration*. CIS. https://www.cisecurity.org/cis-benchmarks

Foster, I. (1995). *Designing and building parallel programs: concepts and tools for parallel software engineering*. Addison-Wesley Longman Publishing Co., Inc.

Fowler, M. (2002). *Patterns of Enterprise Application Architecture*. Addison-Wesley.

Friedl, J. E. F. (2006). *Mastering Regular Expressions* (3. vyd.). O'Reilly Media.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (2016). *Design Patterns: Elements of Reusable Object-Oriented Software* (Updated Edition). Addison-Wesley Professional.

Gorelick, M., & Ozsvald, I. (2020). *High Performance Python: Practical Performant Programming for Humans* (2. vyd.). O'Reilly Media.

*Graylog Documentation*. (2025). Graylog. Získáno 26. dubna 2025, z https://docs.graylog.org/

Grover, M., Malaska, T., Seidman, J., & Shapira, G. (2015). *Hadoop Application Architectures: Designing Real-World Big Data Applications*. " O'Reilly Media, Inc."

He, P., Zhu, J., He, S., Li, J., & Lyu, M. R. (2017). Towards automated log parsing for large-scale log data analysis. *IEEE Transactions on Dependable and Secure Computing*, *15*(6), 931–944.

Hennessy, J. L., & Patterson, D. A. (2019). *Computer Architecture: A Quantitative Approach* (6. vyd.). Morgan Kaufmann.

Hill, M. D., & Marty, M. R. (2008). Amdahl's Law in the Multicore Era. *IEEE Computer*, *41*(7), 33–38. https://doi.org/10.1109/MC.2008.209

Hunt, T., et al. (2023). Principles of Modern Multiprocessing. *Journal of Parallel and Distributed Computing*, *170*, 1–15.

Huo, Y., Su, Y., Lee, C., & Lyu, M. R. (2023). Semparser: A semantic parser for log analytics. *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, 881–893.

Ibarra-Fiallos, S., Higuera, J. B., Intriago-Pazmiño, M., Higuera, J. R. B., Montalvo, J. A. S., & Cubo, J. (2021). Effective filter for common injection attacks in online web applications. *IEEE Access*, *9*, 10378–10391.

*Injection Prevention Cheat Sheet*. (2024). OWASP. Získáno 26. dubna 2025, z https://cheatsheetseries.owasp.org/cheatsheets/Injection_Prevention_Cheat_Sheet.html

Intel. (2002). *Intel Hyper-Threading Technology*. Získáno 26. dubna 2025, z https://www.intel.com/content/www/us/en/architecture-and-technology/hyper-threading/hyper-threading-technology.html

Jiang, Z., Liu, J., Chen, Z., Li, Y., Huang, J., Huo, Y., He, P., Gu, J., & Lyu, M. R. (2024). LILAC: Log Parsing using LLMs with Adaptive Parsing Cache. https://arxiv.org/abs/2310.01796

Karlsen, E., Luo, X., Zincir-Heywood, N., & Heywood, M. (2024). Benchmarking large language models for log analysis, security, and interpretation. *Journal of Network and Systems Management*, *32*(3), 59.

*KNIME Analytics Platform Documentation*. (2023). KNIME. Získáno 26. dubna 2025, z https://docs.knime.com/2023-06/

*KNIME Documentation*. (2024). KNIME. Získáno 26. dubna 2025, z https://docs.knime.com/

Kolouch, J., Novotný, T., & Dvořák, J. (2019). *Cybersecurity*. CZ.NIC.

Komisarek, M., Pawlicki, M., D'Antonio, S., Kozik, R., Pawlicka, A., & Choraś, M. (2024). Enhancing Network Security Through Granular Computing: A Clustering-by-Time Approach to NetFlow Traffic Analysis, 1–8.

*LDAP Authentication in Streamlit*. (2025). Streamlit Inc. Získáno 26. dubna 2025, z https://docs.streamlit.io/library/authentication/ldap

*Legislativa kybernetické bezpečnosti*. (2023). Národní úřad pro kybernetickou a informační bezpečnost. Získáno 26. dubna 2025, z https://nukib.cz/cs/kyberneticka-bezpecnost/regulace-a-kontrola/legislativa/

Liskov, B. (1988). Data abstraction and hierarchy. *ACM SIGPLAN Notices*, *23*(5), 17–34.

Living Security. (2024). Streamlining Cybersecurity Incident Response: The Power of Automated Workflows and External Integrations [Accessed: 2025-04-27]. https://www.livingsecurity.com/blog/streamlining-cybersecurity-incident-response

*LOGmanager Documentation*. (2025). LOGmanager. Získáno 26. dubna 2025, z https://doc.logmanager.com/latest/

*lzma — Compression using the LZMA algorithm*. (2024). Python Software Foundation. Získáno 26. dubna 2025, z https://docs.python.org/3/library/lzma.html

Mărcuță, C., & Team, M. R. (2025). Enhancing Apache Spark Security: Best Practices for Modern Applications [Accessed: 2025-04-27]. https://moldstud.com/articles/p-improving-security-in-apache-spark-for-contemporary-application-development-through-effective-best-practices

Martin, R. C. (2000). Design Principles and Design Patterns. https://web.archive.org/web/20150906155800/http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf

Martin, R. C. (2003). Agile Software Development: Principles, Patterns, and Practices.

Martin, R. C. (2014). The Principles of OOD. *Object Mentor*. http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod

Martin, R. C. (2017). *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall.

*Matplotlib: Python plotting*. (2025). Matplotlib Development Team. Získáno 26. dubna 2025, z https://matplotlib.org/stable/contents.html

Mattis, T., Henning, J., Rein, P., Hirschfeld, R., & Appeltauer, M. (2015). Columnar objects: Improving the performance of analytical applications. *2015 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward!)*, 197–210.

McKinney, W. (2017). *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython* (2. vyd.). O'Reilly Media.

Melnik, S., Gubarev, A., Long, J., Romer, K., Shivakumar, S., Tolton, M., & Vassilakis, T. (2010). Dremel: Interactive Analysis of Web-Scale Datasets. *Proceedings of the VLDB Endowment*, *3*(1), 330–339. https://doi.org/10.14778/1920841.1920886

Microsoft. (2025). *Power BI Documentation*. Microsoft. Získáno 26. dubna 2025, z https://learn.microsoft.com/en-us/power-bi/

*multiprocessing — Process-based parallelism*. (2025). Python Software Foundation. Získáno 26. dubna 2025, z https://docs.python.org/3/library/multiprocessing.html

Nagappan, M., & Vouk, M. A. (2010). Abstracting log lines to log event types for mining software system logs. *7th IEEE Working Conference on Mining Software Repositories (MSR)*, 114–117.

National Institute of Standards and Technology. (2006). *Special Publication 800-92: Guide to Computer Security Log Management* (tech. zpr.). NIST. Získáno 26. dubna 2025, z https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-92.pdf

Opp, R. (2021). *The evolving digital divide*. United Nations Development Programme. Získáno 26. dubna 2025, z https://www.undp.org/blog/evolving-digital-divide

Ousterhout, J. (2018). Always Measure One Level Deeper. *Communications of the ACM*, *61*(7), 74–83. https://doi.org/10.1145/3213770

OWASP. (2023). *OWASP Web Security Testing Guide*. https://owasp.org/www-project-web-security-testing-guide/v42/

*pandas — Python Data Analysis Library*. (2025). The pandas Development Team. Získáno 26. dubna 2025, z https://pandas.pydata.org/docs/

*Paramiko — Python SSH library*. (2024). Paramiko Project. Získáno 26. dubna 2025, z https://docs.paramiko.org/

*Plotly Python Open Source Graphing Library*. (2025). Plotly. Získáno 26. dubna 2025, z https://plotly.com/python/

Prensky, M. (2001). Digital Natives, Digital Immigrants. *On the Horizon*, *9*(5), 1–6. https://doi.org/10.1108/10748120110424816

*Průzkum: Kyberbezpečnost českých firem.* (2022). Česko v datech. Získáno 12. dubna 2023, z https://www.ceskovdatech.cz/clanek/172-pruzkum-kyberbezpecnost-ceskych-firem/

Ramachandrappa, N. C. (2024). SOLID Design Principles in Software Engineering. *International Journal of Computer Trends and Technology*, *72*(9), 18–23.

Rauber, T., & Rünger, G. (2007). *Parallel Programming: for Multicore and Cluster Systems*. Springer.

Security, K. (2023). *Kubernetes Security Best Practices*. Cloud Native Computing Foundation. https://kubernetes.io/docs/concepts/security/

*Security and Monitoring in the ELK Stack*. (2023). Elastic. Získáno 26. dubna 2025, z https://www.elastic.co/guide/en/elasticsearch/reference/current/security.html

*Session State API*. (2025). Streamlit Inc. Získáno 26. dubna 2025, z https://docs.streamlit.io/library/advanced-features/session-state

Schmidt, K., Phillips, C., & Chuvakin, A. (2012). Logging and log management: the authoritative guide to understanding the concepts surrounding logging and log management.

*The SIEM Buyer's Guide*. (2023). Splunk. Získáno 26. dubna 2025, z https://www.splunk.com/en_us/pdfs/resources/e-book/the-siem-buyers-guide-2023.pdf

*SIEM: Security Information & Event Management Explained*. (2024). Splunk. Získáno 26. dubna 2025, z https://www.splunk.com/en_us/blog/learn/siem-security-information-event-management.html

Söderström, O., & Moradian, E. (2013). Secure audit log management. *Procedia Computer Science*, *22*, 1249–1258.

Splunk. (2022). *What is SIEM? Security Information and Event Management*. Získáno 26. dubna 2025, z https://www.splunk.com/en_us/solutions/what-is-siem.html

Stallings, W. (2019). *Foundations of Modern Networking*. Addison-Wesley.

Stone, P. (2010). Next generation clickjacking. *BlackHat Europe*.

*Streamlit Documentation*. (2025). Streamlit Inc. Získáno 26. dubna 2025, z https://docs.streamlit.io/

Tanenbaum, A. S., & Bos, H. (2016). *Modern Operating Systems* (4. vyd.). Pearson.

The Linux Foundation. (2004). *Filesystem Hierarchy Standard*. Získáno 26. dubna 2025, z https://refspecs.linuxfoundation.org/FHS_3.0/fhs/index.html

Tovarnák, D. (2017). *Normalization of Unstructured Log Data into Streams of Structured Event Objects* [Doktorská práce, Masaryk University, Faculty of Informatics] [[online], cited 2025-01-28]. https://is.muni.cz/th/rjfzq/thesis-twoside-final-bw.pdf

Vyhláška o kybernetické bezpečnosti [Sbírka zákonů České republiky], 2018.

Wahl, M., Howes, T., & Kille, S. (1997). *Lightweight Directory Access Protocol (v3)*. Internet Engineering Task Force. https://tools.ietf.org/html/rfc4511

Wang, C., Xu, D., & Li, Z. (2024). Log2graphs: An Unsupervised Framework for Log Anomaly Detection with Efficient Feature Extraction [Accessed: 2025-04-27]. *arXiv preprint arXiv:2409.11890*. https://arxiv.org/abs/2409.11890

Yu, B., Yao, J., Fu, Q., Zhong, Z., Xie, H., Wu, Y., Ma, Y., & He, P. (2024). Deep learning or classical machine learning? an empirical study on log-based anomaly detection, 1–13.

Zhang, S., Ji, Y., Luan, J., Nie, X., Chen, Z., Ma, M., Sun, Y., & Pei, D. (2024). End-to-End AutoML for Unsupervised Log Anomaly Detection [Accessed: 2025-04-27]. *Proceedings of the 2024 IEEE/ACM Automated Software Engineering Conference (ASE)*, 1680–1692. https://nkcs.iops.ai/wp-content/uploads/2024/09/ASE24_LogCraft.pdf

# Přílohy

# A. Validace funkčních a nefunkčních požadavků

**Tabulka A.1**
*Validace funkčních požadavků*

| ID | Požadavek | Stav | Komentář |
|---|---|---|---|
| **1. Automatizovaný systém pro sběr a analýzu logů** | | | |
| 1A | Schopnost připojit se na více než 2 200 zdrojů | Naplněno | Flexibilní navigační modul |
| 1B | Zpracování velkých objemů dat bez prodlev | Naplněno | Paralelní zpracování dat |
| 1C | Adaptace na nové formáty logů a jejich struktury | Částečně naplněno | Nestandardní formáty vyžadují konfiguraci |
| 1D | Automatizace procesu analýzy logů | Naplněno | Plná automatizace dávkového zpracování |
| 1E | Snížení doby analýzy z 12 hodin na méně než 15 minut | Naplněno | Dosaženo přesně 15 minut |
| **2. Intuitivní dashboard a interaktivní vizualizace** | | | |
| 2A | Self-Service Business Intelligence pojetí | Naplněno | Samostatná filtrace a analýza dat |
| 2B | Přehledné zobrazení klíčových metrik | Naplněno | Včetně trendů v časových řadách |
| 2C | Návrh uživatelsky přívětivého dashboardu | Naplněno | Potvrzeno uživatelským testováním |
| 2D | Přímý přístup ke klíčovým ukazatelům | Naplněno | Flexibilní filtrování a průzkum |
| **3. Soulad s legislativou a bezpečnostní standardy** | | | |
| 3A | Soulad s vyhláškou o kybernetické bezpečnosti | Naplněno | Respektuje požadavky vyhlášky |
| 3B | Transparentní evidence incidentů | Naplněno | Mechanismy pro evidenci a audit |
| 3C | Zajištění uchování příslušných logů | Naplněno | Dlouhodobé uchování dat |

*Poznámka: Vlastní zpracování.*

**Tabulka A.2**

*Validace nefunkčních požadavků*

| ID | Požadavek | Stav | Komentář |
|---|---|---|---|
| **NF1. Výkon a škálovatelnost** | | | |
| NF1A | Zpracování 500 logů za méně než 10 sekund | Naplněno | Průměrná doba 8,2 sekundy |
| NF1B | Škálovatelnost pro budoucí nárůst objemu dat | Částečně naplněno | Limitace pro objemy 100+ TB |
| NF1C | Odezva dashboardu do 5 sekund | Naplněno | Průměrná odezva 3,1 sekundy |
| **NF2. Spolehlivost a dostupnost** | | | |
| NF2A | Dostupnost minimálně 99,5 % času | Neověřeno | Vyžaduje dlouhodobé sledování |
| NF2B | Mechanismy pro zotavení po chybě | Naplněno | Automatické zotavení |
| **NF3. Udržitelnost a rozšiřitelnost** | | | |
| NF3A | Modulární architektura | Naplněno | Jasně definovaná rozhraní |
| NF3B | Dokumentovaný kód | Naplněno | Testovací pokrytí $> 85$ % |
| **NF4. Uživatelská přívětivost** | | | |
| NF4A | Intuitivní uživatelské rozhraní | Naplněno | Zaškolení $< 2$ hodiny |
| NF4B | Konzistentní design a terminologie | Naplněno | Napříč celým systémem |
| NF4C | Responzivní design dashboardu | Částečně naplněno | Optimální na větších obrazovkách |

*Poznámka: Vlastní zpracování.*