

Systém správy událostí aplikací

Bc. Martin Domanský



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2021/2022

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	Bc. Martin Domanský
Osobní číslo:	A20143
Studijní program:	N0613A140022 Informační technologie
Specializace:	Softwarové inženýrství
Forma studia:	Kombinovaná
Téma práce:	Systém správy událostí aplikací
Téma práce anglicky:	Event Management System of Applications

Zásady pro vypracování

1. Analyzujte možnosti ukládání záznamů událostí aplikací.
2. Navrhněte systém umožňující zefektivnit správu událostí aplikací.
3. Realizujte navržený systém.
4. Pro vytvořený systém implementujte grafické uživatelské rozhraní.
5. Analyzujte další možnosti rozvíjení systému a jeho napojení na systémy řízení projektů a správy vývoje software.

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. Dokumentace k jazyku C# – začínáme, kurzy, referenční dokumentace | Microsoft Docs. [online]. Copyright © Microsoft 2021 [cit. 17.11.2021]. Dostupné z: <https://docs.microsoft.com/cs-cz/dotnet/csharp/>
2. Dokumentace k ASP.NET Microsoft Docs. [online]. Copyright © Microsoft 2021 [cit. 17.11.2021]. Dostupné z: <https://docs.microsoft.com/cs-cz/aspnet/core/?view=aspnetcore-6.0>
3. Best of 2018: Log Monitoring and Analysis: Comparing ELK, Splunk and Graylog – DevOps.com. DevOps – The Web's Largest Collection of DevOps Content [online]. Copyright © 2021 [cit. 17.11.2021]. Dostupné z: <https://devops.com/log-monitoring-and-analysis-comparing-elk-splunk-and-graylog/>
4. Aspektově orientované programování – Logování. Největší český web zaměřený na .NET framework [online]. Copyright © 2021 [cit. 17.11.2021]. Dostupné z: <https://www.dotnetportal.cz/blogy/16/Martin-Dybal/6395/Aspektove-orientovane-programovani-Logovani>
5. Protokolování a trasování – .NET Core | Microsoft Docs. [online]. Copyright © Microsoft 2021 [cit. 17.11.2021]. Dostupné z: <https://docs.microsoft.com/cs-cz/dotnet/core/diagnostics/logging-tracing#ilogger-and-logging-frameworks>

Vedoucí diplomové práce:

prof. Mgr. Roman Jašek, Ph.D., DBA
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **3. prosince 2021**

Termín odevzdání diplomové práce: **23. května 2022**

doc. Mgr. Milan Adámek, Ph.D. v.r.
děkan



prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 24. ledna 2022

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomové práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude uložen v příruční knihovně Fakulty aplikované informatiky. Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má Univerzita Tomáše Bati ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne

.....

podpis studenta

ABSTRAKT

Tato diplomová práce se zabývá strojovým zpracováním informací z logovaných událostí chování aplikací. Pojednává o alternativním přístupu k reportování vnitřního chování a aktuálního stavu běžících aplikací. Vizualizace logovaných informací je efektivní způsob monitorování aplikací. Cílem je vyvinout systém pro zpracování, analýzu a vizualizaci událostí logovaných v rámci běhu aplikací. Díky tomuto systému bude možné logované události dále zpracovávat a vyhodnocovat. Oddělení kontroly kvality aplikací umožní zefektivnit jejich činnost a poskytnout detailnější informace oddělení vývoje.

Klíčová slova: .NET, automatizace, logování, databáze

ABSTRACT

This thesis deals with the machine processing of information from logged application behavior events. It discusses an alternative approach to reporting the internal behavior and current state of running applications. Visualization of logged information is an effective way of monitoring applications. The goal is to develop a system for processing, analyzing and visualizing logged events within running applications. With this system, the logged events can be further processed and evaluated. It will enable the application quality control department to streamline their operations and provide more detailed information to the development department.

Keywords: .NET, automation, logging, database

Rád bych poděkoval prof. Mgr. Roman Jašek Ph.D., DBA za vedení práce a cenné připomínky. Zvláště bych rád poděkoval své rodině a přátelům za projevenou podporu a trpělivost v celém průběhu psaní práce.

OBSAH

ÚVOD	9
CÍL.....	10
1 LOGOVÁNÍ UDÁLOSTÍ APLIKACÍ	11
1.1 STRUKTURA LOGOVACÍHO ZÁZNAMU	11
1.2 LOGOVACÍ ANALYZÁTORY	13
2 AUTOMATIZACE PROCESŮ	15
2.1 ZAVEDENÍ AUTOMATIZACE PROCESŮ	15
2.2 PŘÍNOSY AUTOMATIZACE PROCESŮ	16
3 VYBRANÉ TECHNOLOGIE	17
3.1 JAZYK C#	17
3.2 ASP.NET.....	17
3.2.1 Web APIs.....	19
3.2.2 REST API.....	19
3.3 MICROSOFT SQL SERVER	20
3.4 JIRA	20
3.5 ELK STACK	22
4 ARCHITEKTURA SYSTÉMU	23
4.1 PREZENTAČNÍ VRSTVA	23
4.1.1 Komunikační rozhraní	24
4.1.2 Grafické rozhraní	24
4.2 APLIKAČNÍ VRSTVA	28
4.2.1 Webové rozhraní.....	29
4.2.2 Business core	31
4.2.3 Databázové rozhraní	32
4.3 DATOVÁ VRSTVA.....	34
5 AUTOMATIZAČNÍ PROCESY SYSTÉMU	35
5.1 PROCES ZPRACOVÁNÍ LOGOVANÝCH DAT	35
5.1.1 Normalizace dat	36
5.1.2 Slučovací pravidla.....	37
5.2 PROCES ZPRACOVÁNÍ ÚKOLŮ Z JIRA	39
5.3 PROCES SPRÁVY UDÁLOSTÍ	40
5.4 UDÁLOSTI	41
5.5 STAVY UDÁLOSTÍ	41

6	TESTOVÁNÍ	44
6.1	AUTOMATIZOVANÉ TESTY	44
6.2	UŽIVATELSKÉ TESTOVÁNÍ	45
7	DISKUZE	46
	ZÁVĚR	48
	SEZNAM POUŽITÉ LITERATURY	49
	SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK	51
	SEZNAM OBRÁZKŮ	52
	SEZNAM PŘÍLOH	53

ÚVOD

Logování je nezbytnou součástí informačního světa a využívá se jak při analýze hardwaru, tak i softwaru. Logované záznamy usnadňují analýzu chování dané aplikace.

U každé aplikace můžeme analyzovat její typické vnitřní a vnější chování. Zatímco analýza vnějšího chování vychází z pozorování grafického rozhraní nebo konzolových výstupů, vnitřní chování aplikací nelze jednoduše pozorovat ani analyzovat. Pro analýzu vnitřního chování aplikací můžeme využít různých specializovaných softwarových nástrojů.

Vnitřní chování je důležité při vývoji a správě aplikací. Pomáhá odhalovat neočekávané stavy a události vzniklé při běhu programu. Znalostí vnitřního chování můžeme rozlišit, jestli aplikace nereaguje z důvodu, že je zaneprázdněná obsluhou aktuální úlohy nebo se nachází v chybovém stavu a vyžaduje odborný zásah programátora. Své vnitřní chování může každá aplikace jednoduše reportovat pomocí logovacích záznamů, které popisují vnitřní stavy a události dané aplikace. Analýza těchto logů může být prováděna jednoduše pomocí lidského faktoru nebo pokročile pomocí vhodné míry automatizace.

Tématem této práce je pochopení problematiky zpracování a vyhodnocení zaznamenaných událostí aplikací. Na základě poznatků bude vytvořen nástroj, který usnadní programátorům a také administrátorům ve zpracování těchto událostí. Nástroj umožní jednodušší pochopení chybného vnitřního chování při analýze logovaných záznamů. Celý proces evidování událostí, detekci problémů, reportování chyb, analýzy a řešení chyb lze zjednodušit automatizací procesů, o kterou se může starat vhodně navržený systém. Automatizované procesy šetří lidskou práci, minimalizují výskyt procesních chyb a důsledkem je i snížení finančních nákladů. Důležitou součástí je vizualizace zpracovaných informací, která zdůrazňuje asociace mezi událostmi a umožní tak i odhalení kritických míst programů nebo aplikací a jejich následnou optimalizaci.

CÍL

Hlavním cílem této diplomové práce je vytvořit funkční prototyp systému, který umožní zefektivnit analýzu chybových událostí aplikací. Tohoto cílu by mělo být možné dosáhnout pomocí automatizovaného zpracovávání logovaných záznamů aplikací a pomocí cílené automatizace procesů souvisejících s analýzou chybových událostí aplikací. Prototyp systému by měl být otestován v zadavatelské firmě a bude zpracovávat logované data z aplikací vyvíjených firmou.

Dalším cílem práce je snížit potřebu lidských zdrojů při reportování chybových událostí vhodnou mírou automatizace. Pokud vytvořený systém bude umět komunikovat se externími systémy pro správu úkolů, bude možné automatizovat proces zakládání nových úkolů na základě analyzované chybové události. Jednotná forma úkolů zvyšuje čitelnost úkolů pro řešitelé úkolů.

Výsledkem práce by měl být nástroj, který zefektivňuje zpracování logů a pomáhá identifikovat rozsah i frekvenci výskytů problémových událostí.

1 LOGOVÁNÍ UDÁLOSTÍ APLIKACÍ

Logování událostí je v počítačové terminologii činnost, při které se zaznamenávají informace o činnostech a běhu aplikace. Při logování se generují záznamy obsahující potřebné informace o aplikaci a tyto záznamy nazýváme *logy*. Logy mohou být ukládány do logovacích souborů a nebo databází. *Log* je tedy záznam, který obsahuje informaci o stavu aplikace v daný moment.

V praxi se můžeme setkat s více způsoby, jak vytvářet, ukládat a zpracovávat logy aplikací. O tvorbu logů se mohou starat speciální generátory logů, ale mohou být realizovány jednoduchým textovým výpisem do konzole. Ukládat logy je možné na lokální úložiště nebo vzdálené databázové servery. V případě posílání logovaných záznamů po síti lze standardní síťové protokoly TCP nebo UDP. U lokálního zaznamenávání logů do souboru, je možné ukládat do nestrukturovaného textového souboru (.txt), strukturovaného souboru (.xml, .json) nebo databázového souboru (například SQLite). Zpracování logů může být výhodné ručně pomocí lidských zdrojů, částečně automatizované, kdy je možné filtrovat zpracované logované záznamy nebo automatizované, které je více popsáno v podkapitole 1.2 a kapitole 5. [1, 2]

Pro každý software existuje typické vnitřní a vnější chování. Zatímco vnější chování lze přímo pozorovat a analyzovat, například pomocí dostupných grafických či textových výstupů, vnitřní chování software nelze jednoduše pozorovat a analyzovat jej můžeme pouze pomocí specializovaných softwarových nástrojů.

Vnitřní chování software je důležité při vývoji a správě aplikací, protože pomáhá odhalovat neočekávané stavy a události vzniklé při běhu programu. Může se tak odhalit, jestli aplikace nereaguje protože je zaneprázdněná obsluhou aktuální úlohy, očekává podnět z externího zdroje nebo se nachází v chybovém stavu a vyžaduje odborný zásah. Logování událostí pomáhá retrospektivně analyzovat postupné události, které vedly k jednotlivým stavům aplikace.

1.1 Struktura logovacího záznamu

Logovací záznamy mohou sloužit pro administrátory i pro programátory. Mohou být obecně informativní o stavu aplikace, ale mohou obsahovat i plno dodatečných informací. Různé dodatečné informace pomáhají programátorům s pochopením problému a trasováním chyb. Míra dodatečných informací nesmí příliš komplikovat čitelnost logu nebo jeho zpracování.

Pro každou aplikaci může být nejvhodnější jiná struktura logovaných záznamů. Pro lepší orientaci v logovacích záznamech je vhodnější, aby logy dodržovaly běžné konvence a standardní struktury. Obecná struktura logovaných záznamů se skládá ze tří částí:

typ záznamu, časová značka záznamu a obsah záznamu.

Typ záznamu

Vypovídá o důležitosti dané zprávy. Používá se pro filtrování obsahu logovaných záznamů.

Základní typy záznamů logu, které se nejběžněji se používají jsou:

- DEBUG - záznamy obsahující detailní popis chování aplikace a jejích prvků; využívají se pro ladění aplikace.
- INFO - informativní záznamy, které obecně popisují běžné chování aplikace.
- WARN - záznamy popisují nestandardní chování aplikace; upozorňují na podstatné změny v běhu aplikace.
- ERROR - záznamy popisují vzniklou chybu nebo neočekávané chování aplikace.

V případě potřeby se používají i další typy záznamů:

- MONITORING - monitorovací záznamy; využívají se ke statistickým účelům
- TRACE - trasovací záznamy; využívají se pro ladění aplikace nebo pochopení chování běhu aplikace.
- FATAL - záznamy popisují kritické problémy, které mohou způsobit neočekávané chování aplikace

Časová značka záznamu

Informuje o době, kdy byla logovaná událost zapsána do logu. Pro přesnost časové značky mohou být dostačující jednotky sekund, ale existují i případy, kdy je vhodnější zaznamenávat události s přesností na tisíce sekund. Logovací záznam může obsahovat i časovou značku momentu, ve kterém nastala reportovaná událost, kvůli přesnějšímu řetězení záznamů. V takovém případě se tato časová značka zapisuje do obsahu zprávy.

Obsah záznamu

Obsahuje popis reportované události. Popis události může být pouze textový, ale může obsahovat i serializované objekty do formátu XML či JSON. Obsah záznamu může být doplněný o další informace, které jsou důležité pro popis události. Můžeme zde nalézt informaci o názvu vlákna, které zpracovává danou úlohu, *SessionId*, které identifikuje sekci ve které se algoritmus nachází, a také i *StackTrace* vzniklé chyby.

Struktura logů by měla odpovídat účelu, které má plnit logovací soubor. Pro administraci aplikace postačí formát prostého textu, kdy obsah logovacího záznamu obsahuje

```
2022/04/13 01:00:23.657|INFO| Processing platform 'TSutil'
2022/04/13 01:00:23.885|INFO| Processing platform 'TSutil'
2022/04/13 01:00:24.008|WARN| On path '\\TSUTIL.xxxxx.lab\AS' cannot find 'ASConfig.xml' or 'ApplicationServer.xml'
2022/04/13 01:00:24.042|WARN| On path '\\TSUTIL.xxxxx.lab\AS' cannot find 'AssemblyList.xml' or 'Version.log'
2022/04/13 01:00:24.042|ERROR| Cannot get config file of platform 'TSutil'!
2022/04/13 01:00:24.042|ERROR| Something gone wrong while checking page of platform 'TSutil'!
```

Obrázek 1.1 Ukázka jednoduchého logu

pouze stručný popis aktuálního stavu aplikace, a jaká událost nastala v aplikaci. Příliš detailní popis snižuje čitelnost logů a pro účely administrace aplikace není užitečný. V případě, že logy budou pomáhat programátorovi k pochopení vzniklého problému a reprodukování chyby, bude vhodnější aby obsahová část logu byla doplněna o další užitečné informace. Větší množství informací úměrně snižuje čitelnost logu, proto formát prostého textu nemusí být dostačující. V takovém případě se využívají formáty se samopopisnou strukturou nebo snadně serializovatelné formáty jako třeba XML nebo JSON. Ve vhodných případech lze použít i netextový formát, například binární BSON, nebo zapisovat logovací záznamy přímo do databáze.

1.2 Logovací analyzátoři

Nejvýznamnější součástí logovací infrastruktury jsou *logovací analyzátoři*. Jedná se o specializované programy, které vizualizují, prohledávají a zkoumají logované data. Mohou být stěžejní částí při interpretaci logovaných událostí a také interpretují logované informace z logovaných dat.

Logovacích analyzátorů můžeme nalézt mnoho typů, ale s určitou mírou abstrakce je lze rozdělit na dvě základní skupiny. První skupinou jsou real-time analyzátoři, které upozorňují na aktuální stav aplikace. Druhou skupinou jsou off-line analyzátoři, které se starají o statistickou analýzu.

Real-time analyzátoři nepřetržitě monitorují chování aplikace a poskytují přehled, který je potřeba k optimalizaci výkonu sítě, minimalizaci prostoru pro útoky, zvýšení bezpečnosti a zlepšení správy zdrojů. Nestačí však jen vědět, jak monitorovat chování aplikace. Důležité je také zvážit zdroje dat pro patřičné monitorovací nástroje. Real-time analyzátoři nemusí monitorovat pouze chování aplikace, ale mohou monitorovat i síťový provoz nebo zdroje běžícího systému. Přehledy vytvořené real-time analyzátoři mohou být využity pro pozdější analýzu problémů.

Off-line analyzátoři umožňují sledovat a odhadovat trendy, detekovat anomálie a naleznou využití i při zpětném řešení bezpečnostních incidentů. Analyzátoři mohou pracovat s logy v surovém formátu, ale kvůli variabilitě různých formátů může být analýza nad nestrukturalizovanými daty značně komplikovaná. Zjednodušení zpracování logů řeší normalizace dat, tedy transformace dat do strukturované podoby. Při

normalizaci dat může docházet ke ztrátě části informací z logů ale i doplnění dalších informací, které nebyly součástí nestrukturalizovaných dat.

2 AUTOMATIZACE PROCESŮ

Standardním postupem pro odhalování chyb v logovaných záznamech je denní procházení logovacích souborů a hledání případných chyb. Tato činnost je časově náročná a každý den vyžaduje zapojení někdy i více lidí. Při nalezení chyby následuje pokus o její reprodukci, který není vždy úspěšný. Po bližším prozkoumání dostupných informací založí pracovník úkol v systému pro správu úkolů, který je následně přiřazen na programátora. Úkol může pracovník doplnit o detaily z logů nebo o vlastní poznámky.

Tento přístup má mnoho slabých míst, kdy prvním z nich je nutnost výběru proškoleného a schopného člověka vhodného pro tuto práci. Velkou roli zde hraje lidský faktor, kdy může dojít k nenalezení některých chybových hlášek nebo dokonce onemocnění pověřeného pracovníka. Rovněž každý člověk má své postupy a způsoby zpracování událostí, což pro vede k nejednotným zápisům o událostech v systému pro správu úkolů.

Naproti tomu automatizované zpracování logů umožňuje zakládat úkoly automaticky v jednotném formátu a odprošťuje od komplikací spojených s lidským faktorem. Automatizace procesů plní většinou hned tři funkce: automatizuje procesy, centralizuje informace a snižuje požadavky na vstupy od lidí. Jejím cílem je odstranění úzkých hrdel (tzv. bottle neck), omezení chyb a omezení ztráty dat. [3]

Hlavním přínosem je odstranění lidských vstupů, což snižuje chybovost, zvyšuje rychlost dodávek, zvyšuje kvalitu, minimalizuje náklady a zjednodušuje procesy.

2.1 Zavedení automatizace procesů

Prvním krokem při zavádění automatizace je identifikace úkolů a procesů, které lze automatizovat. Jedná se o procesy, které se často opakují a vyžadují zapojení lidských zdrojů.

Hlavní společné rysy automatizovatelných procesů:

- Časová náročnost
- Opakovatelnost
- Ovlivnění dalších navazujících procesů
- Standardizovanost
- Nutnost zapojení více lidí

Druhým a velmi důležitým krokem je stanovení cíle. Tento krok obnáší vytvoření jasné představy o tom, kterým směrem se bude vykonávání procesů dále směřovat a

jaké výhody to bude přinášet. S tímto krokem se váže také následující krok, spočívající ve volbě vhodného nástroje.

Před učiněním samotného rozhodnutí je zapotřebí provést průzkum stávajícího řešení, se kterým mohou výrazně pomoci například odborníci na automatizaci procesů. Výsledkem této části je výběr jednoduchého a snadno pochopitelného řešení, které bude rovněž flexibilní a bude odpovídat stanoveným požadavkům.

Nejdůležitějším krokem je řízení změn, jež nastane při přechodu na novou automatizaci. Jedná se o zdlouhavou část, která vyžaduje proškolení zaměstnanců, zavedení nových pravidel a také získávání zpětné vazby.

Právě pravidelné získávání zpětné vazby a sledování klíčových ukazatelů výkonosti je rozhodujících z dlouhodobého hlediska. Podchycení razantních změn v čas umožňuje plynulejší přechod a časnější přizpůsobení automatizace.

2.2 Přínosy automatizace procesů

Mezi hlavní přínosy patří úspora času, kdy dochází k rapidnímu snížení úkolů vykonávaných ručně. Místo toho se mohou pracovníci soustředit na náročnější úkony vyžadující jejich pozornost, zatímco dosavadní denně vykonávaná rutina je odbavována automaticky za pomoci nově zavedených procesů.

Nezanedbatelným přínosem je také snížení nákladů plynoucí z kombinace nižší chybovosti a vyšší rychlosti zpracování. Úkony nejsou nijak ovlivňovány lidským faktorem, popřípadě je tento vliv snížen na nezbytné minimum. Díky tomu dochází k menšímu množství chyb a zpoždění, které by jinak ovlivnily další řetězec následujících procesů.

3 VYBRANÉ TECHNOLOGIE

Pro vývoj systému byl zvolen jazyk C# , protože se jedná o moderní a populární vyšší programovací jazyk. U vyšších programovacích jazyků je typické využití vyšší míry abstrakce pomocí různých frameworků a knihoven. Proto tvorbu RESTful webového rozhraní byl použit framework ASP.NET (viz 3.2) a architektonický vzor MVC. Ukládání strukturovaných dat s paralelním přístupem k datům je vhodné pro relační databáze. Z tohoto důvodu byl jako databázový server preferován Microsoft SQL Server (viz 3.3).

Upřednostnění technologií od firmy Microsoft je dáno tím, že vývoj těchto technologií je předvídatelný a existuje stabilní silná komunita vývojářů, kteří mají podíl na dalším rozvoji.

3.1 Jazyk C#

Jazyk C# (vyslovuje se "See Sharp") je moderní, objektově orientovaný a typově bezpečný programovací jazyk. Umožňuje vývojářům vytvářet mnoho typů bezpečných a robustních aplikací, které běží v prostředí .NET. Jazyk C# má své kořeny v rodině jazyků C a bude intuitivní programátorům v jazycích C, C++, Java a JavaScript. [4]

Mnoho funkcí jazyka C# pomáhá vytvářet robustní a odolné aplikace. Garbage collection automaticky získává zpět paměť obsazenou nedosažitelnými nepoužívanými objekty. Nulovatelné typy chrání před proměnnými, které neodkazují na alokované objekty. Zpracování výjimek poskytuje strukturovaný a rozšiřitelný přístup k detekci a obnově chyb. Lambda výrazy podporují techniky funkcionálního programování. Syntaxe LINQ (Language Integrated Query) vytváří společný vzor pro práci s daty z libovolného zdroje. Podpora jazyka pro asynchronní operace poskytuje syntaxi pro vytváření distribuovaných systémů. Jazyk C# má jednotný typový systém. Všechny typy jazyka C# , včetně primitivních typů, jako jsou `int` a `double`, dědí z jediného kořenového objektového typu *object*. Všechny typy sdílejí sadu společných operací. Hodnoty libovolného typu lze ukládat, přenášet a pracovat s nimi konzistentním způsobem. Kromě toho jazyk C# podporuje jak uživatelsky definované referenční typy, tak hodnotové typy. Umožňuje dynamické přidělování objektů a řádkové ukládání lehkých struktur. Podporuje generické metody a typy, které zajišťují vyšší typovou bezpečnost a výkon. Poskytuje iterátory, které umožňují implementátorům tříd kolekcí definovat vlastní chování klientského kódu. [4, 5]

3.2 ASP.NET

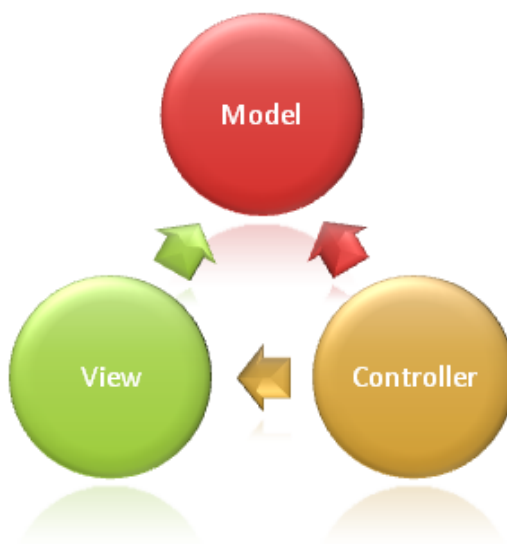
ASP.NET je open-source framework pro tvorbu webových aplikací na straně serveru, který je určen pro vývoj webových stránek. Byl vyvinut společností Microsoft, aby

umožnil programátorům vytvářet dynamické webové stránky, aplikace a služby. Název je zkratkou Active Server Pages Network Enabled Technologies. ASP.NET je postaven na běhovém prostředí Common Language Runtime (CLR), což programátorům umožňuje psát kód ASP.NET pomocí libovolného podporovaného jazyka platformy .NET. Rozšiřující rámec ASP.NET umožňuje jeho komponentám zpracovávat zprávy SOAP, REST či gRPC. [6]

ASP.NET Core MVC poskytuje efektivní způsob tvorby dynamických webových stránek založený na architektonických vzorech. Obsahuje také mnoho funkcí, které umožňují rychlý vývoj a umožňují vytvářet sofistikované aplikace využívající nejnovější webové standardy. [7]

Architektonický vzor MVC (Model-View-Controller) popisuje tři typy komponent: Model, View a Controller. Tento vzor odděluje jednotlivé komponenty a vymezuje jejich odpovědnosti v rámci aplikace. Uživatelské požadavky jsou směřovány na Controller, který se stará o práci s Modelem a provádí uživatelské akce a/nebo načítá výsledky dotazů. Controller určuje, které View se zobrazí uživateli, a poskytuje z data Modelu, která View požaduje.

Následující schéma ukazuje princip znázorňuje, jak na sebe vzájemně odkazují hlavní komponenty architektonického vzoru MVC:



Obrázek 3.1 Vzor MVC

Toto vymezení odpovědností pomáhá škálovat aplikaci z hlediska složitosti, protože je snazší kódovat, ladit a testovat něco (model, view nebo controller), co má jedinou úlohu. Je obtížnější aktualizovat, testovat a ladit kód, který má závislosti rozprostřené ve dvou nebo více z těchto tří oblastí. Například logika uživatelského rozhraní má tendenci se měnit častěji než business logika. Pokud jsou prezentační kód a business logika spojeny v jednom objektu, musí být objekt obsahující business logiku upraven

při každé změně uživatelského rozhraní. To může zanášet další chyby do systému a vyžaduje to opětovné testování business logiky po každé minimální změně uživatelského rozhraní. [7]

3.2.1 Web APIs

Framework ASP.NET Core MVC je vhodnou platformou pro tvorbu webových stránek a má také dobrou podporu pro tvorbu webových rozhraní API. Lze vytvářet služby, které budou mít širokou podporu klientů včetně prohlížečů a mobilních zařízení. Framework obsahuje podporu pro vyjednávání obsahu HTTP s vestavěnou podporou pro formátování dat jako JSON nebo XML. [7]

Webové rozhraní API je aplikační programové rozhraní pro webový server nebo webový prohlížeč. Jedná se o koncept vývoje webu, který se obvykle omezuje na klient-skou stranu webové aplikace (včetně všech používaných webových frameworků), a proto obvykle nezahrnuje detaily implementace webového serveru nebo prohlížeče. Na straně serveru je programové rozhraní sestávající z jednoho nebo více veřejně přístupných koncových bodů definovaného systému zpráv typu request–response, obvykle vyjádřeného ve formátu JSON nebo XML, které je vystaveno prostřednictvím webu (nejčastěji prostřednictvím webového serveru založeného na protokolu HTTP).

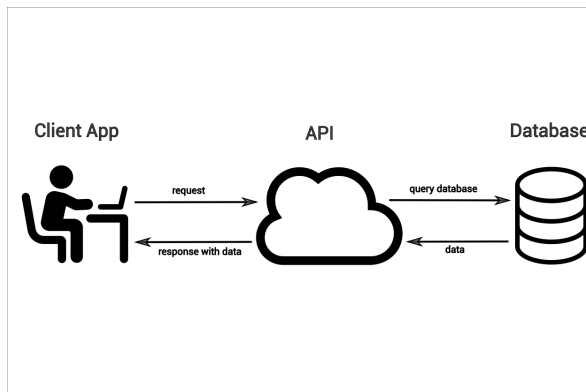
3.2.2 REST API

REST je architektonický styl, který lze využít při návrhu a vývoji rozhraní pro komunikaci skrze internet pomocí protokolu HTTP. Definuje omezení a pravidla, které vývojáři API radí jak vytvářet architekturu webového rozhraní a jak reagovat na konkrétní požadavky. REST si klade důraz na škálovatelnost interakcí mezi komponentami, jednotná rozhraní, nezávislé nasazení komponent a vytvoření vícevrstvé architektury. Cílem REST je zvýšit výkon, škálovatelnost, jednoduchost, modifikovatelnost, přehlednost, přenositelnost a spolehlivost. [8, 9]

REST nelze považovat za protokol ani standard, ale spíše rozšířeným a uznávaným souborem pokynů pro vytváření bezstavových, spolehlivých webových rozhraní. Webové rozhraní, které dodržuje omezení REST, můžeme označit jako RESTful rozhraní. Hlavičky a parametry HTTP požadavku mohou obsahovat důležité identifikační informace o metadatech požadavku, autorizaci, identifikátoru zdroje a atd.

U RESTful webových služeb RESTful požadavky na URI zdroje vyvolávají odpověď ve formátu HTML, XML, JSON nebo jiném formátu. Odpověď může například potvrdit změněnu stavu prostředku, ale také může obsahovat hypertextové odkazy na další zdroje. Nejběžnějším protokolem pro tyto požadavky a odpovědi je protokol HTTP. Poskytuje základní CRUD operace, konkrétně metody GET, POST, PUT a DELETE.

Použitím bezstavového protokolu a standardních operací usilují systémy RESTful o rychlý výkon, spolehlivost a možnost růstu díky opakovanému použití komponent, které lze spravovat a aktualizovat bez vlivu na systém jako celek, a to i za jeho chodu.



Obrázek 3.2 Funkce REST API

3.3 Microsoft SQL Server

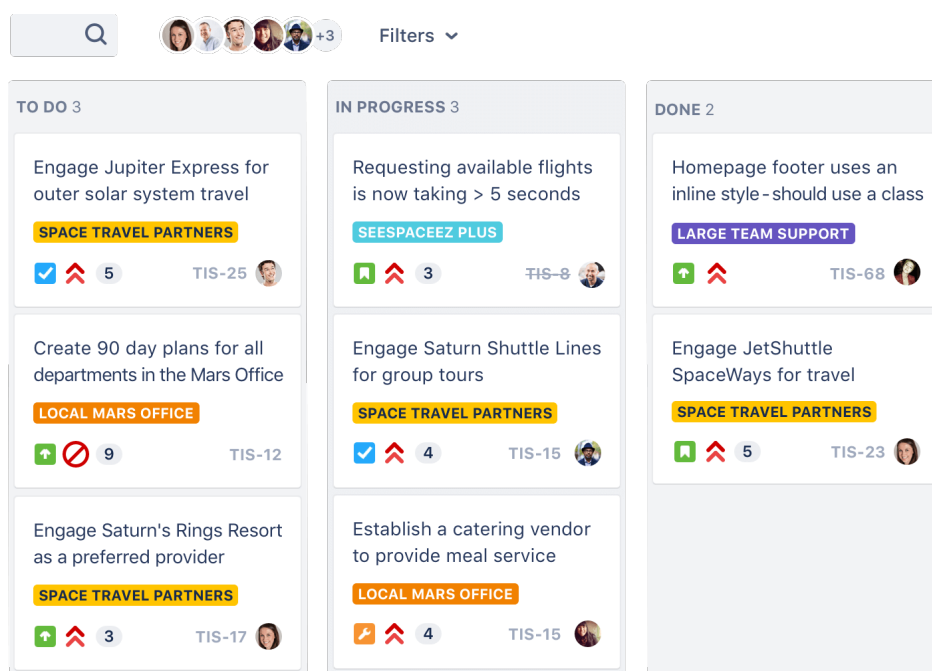
Microsoft SQL Server je relační systém pro správu databází vyvinutý společností Microsoft. Jako databázový server je to softwarový produkt, jehož hlavní funkcí je ukládání a načítání dat podle požadavků jiných softwarových aplikací.

Relační databáze se skládá z kolekce tabulek, které uchovávají určitou sadu strukturovaných dat. Tabulky obsahují množinu řádků, které označujeme jako záznamy, a sloupců, které označujeme jako atributy. Každý sloupec v tabulce je určen k ukládání určitého typu informací, například textových řetězců, časových značek, čísel nebo identifikačních klíčů. Data jednotlivých atributů mohou být unikátní v rámci tabulky v daném sloupci. Záznamy v tabulkách mohou být spolu vzájemně propojeny pomocí identifikačních klíčů.

3.4 Jira

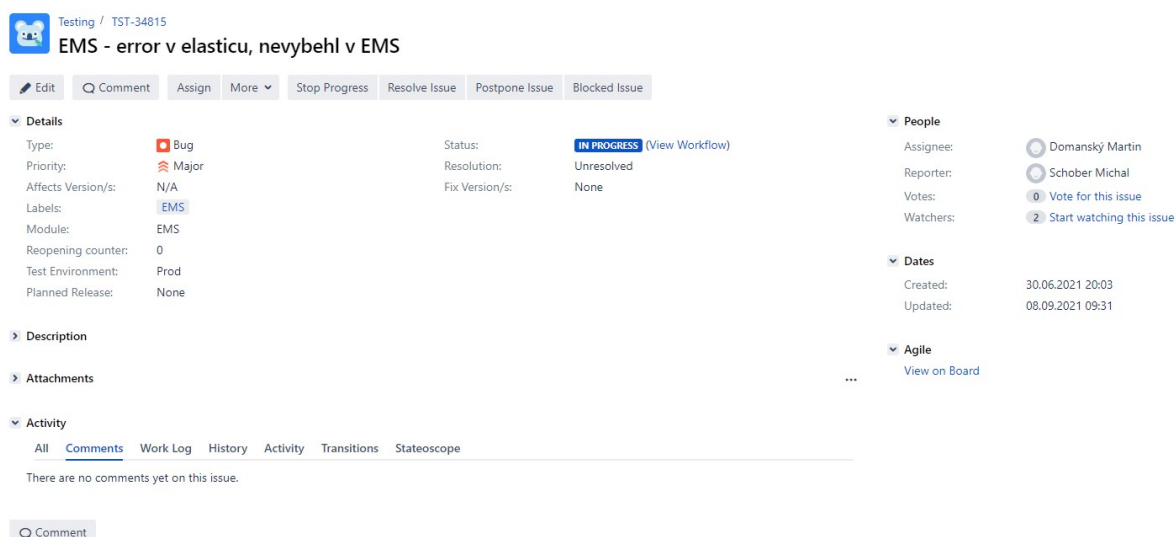
Jira je softwarový nástroj vyvíjený společností Atlassian. Původně byla navržena jako nástroj pro sledování chyb a problémů. Dnes lze využít také pro správu práce od správy požadavků a testovacích případů až po agilní vývoj softwaru. Na obrázku 3.3 je ukázka tabule sloužící pro plánování a monitorování úkolů na projektu. Při vývoji softwaru je umožněno flexibilní řízení projektů a monitorování plnění vytvořených úkolů. Jira je součástí rodiny produktů, které mají pomáhat týmům řídit práci na projektu. [10, 11]

Pro práci s úkoly v systému Jira se běžně využívá grafického rozhraní, viz obrázek 3.4. Grafické rozhraní je přizpůsobeno pro intuitivní práci uživatelů v systému. Pro práci se systémem Jira mimo grafické rozhraní je poskytováno privátní RESTful webové



Obrázek 3.3 Ukázka plánování úkolů v Jira [10]

rozhraní. Toto rozhraní poskytuje základní poskytuje základní CRUD operace pro práci s projekty a úkoly.



Obrázek 3.4 Detail úkolu v Jira

Vyhledávání a filtrování úkolů na projektech je umožněno pomocí vlastního dotazovacího jazyka JQL. Teto dotazovací jazyk je v systému využívám pro vyhledávání souvisejících úkolů pro konkrétní logovanou událost. Pro vytvoření nového úkolu a modifikace existujícího úkolu postačují základní CRUD operace webového rozhraní. [12]

3.5 ELK Stack

Elasticsearch je distribuovaný, bezplatný a otevřený vyhledávací a analytický stroj pro všechny typy dat. Byl vytvořen na platformě Apache Lucene a je specifickým zástupcem NoSQL vyhledávacích databází. Poskytuje distribuovaný plnotextový vyhledávač s podporou více uživatelů, webovým rozhraním HTTP a dokumenty JSON bez schémat. [13, 14]

Elasticsearch je známý pro své jednoduché rozhraní REST API, distribuovanou povahu, rychlost a škálovatelnost a je ústřední součástí sady ELK Stack. Jedná se o sadu nástrojů pro příjem, obohacování, ukládání, analýzu a vizualizaci dat. Sada ELK Stack zahrnuje kolekci nástrojů, z nichž nejvýznamnější jsou Elasticsearch, Logstash a Kibana. [15]

Kibana je nástroj pro vizualizaci a správu dat pro Elasticsearch. Poskytuje histogramy, čárové grafy, koláčové grafy a mapy v reálném čase. Součástí Kibany jsou také pokročilé aplikace, jako je Canvas, která uživatelům umožňuje vytvářet vlastní dynamické infografiky na základě jejich dat, a Elastic Maps pro vizualizaci geoprostorových dat. [16]

Logstash se používá k agregaci a zpracování dat a jejich odesílání do Elasticsearch. Umožňuje na straně serveru zpracovávat data z více zdrojů současně a obohacovat a transformovat je před jejich indexováním do Elasticsearch. [17]

4 ARCHITEKTURA SYSTÉMU

Vytvořený systém pro správu chybových událostí byl implementován v jazyce C# a jeho logické celky odpovídají stylu třívrstvé architektury. Třívrstvá architektura je dobře známá architektura softwarových aplikací, která rozděluje aplikaci do tří logických (někdy i fyzických) výpočetních vrstev. Prezentační vrstva neboli uživatelské rozhraní, aplikační vrstva, kde se řeší aplikační logika, a datová vrstva, kde se ukládají a spravují data spojená s aplikací.

Hlavní výhodou třívrstvé architektury je, že každá vrstva může být nasazena na vlastní infrastrukturu i na samostatném operačním systému, dle funkčních požadavků. Každou vrstvu lze provozovat na jednom sdíleném serverovém hardwaru či virtuálním serveru, ale v případě potřeby je možné každou vrstvu provozovat na vyhrazeném serveru nebo dokonce na více serverech. Služby každé vrstvy lze přizpůsobit a optimalizovat bez dopadu na ostatní vrstvy. Každá vrstva může být současně vyvíjena samostatným vývojovým týmem. Jednotlivé vrstvy lze podle potřeby aktualizovat nebo rozšiřovat, aniž by to mělo vliv na ostatní vrstvy. [18]

Po desetiletí byla třívrstvá architektura převládající architekturou pro aplikace typu klient-server. Dnes je tato architektura preferována z důvodu jednoduché modernizace aplikací s využitím cloudových nativních technologií, jako jsou kontejnery a mikroslužby, a migrace do cloudu. [18]

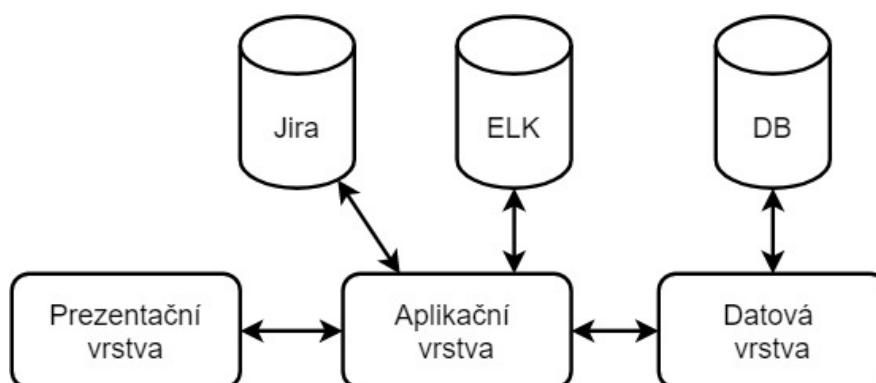
Vytvořený systém využívá prvky třívrstvé architektury a je rozdělen na tři samostatné celky:

- **Prezentační vrstva 4.1** - zobrazuje informace pro uživatele pomocí grafického rozhraní; validuje vstupy od uživatele
- **Aplikační vrstva 4.2** - obsahuje logiku aplikace; řeší o zpracování dat; komunikuje se vzdálenými servery Jira a ELK
- **Datová vrstva 4.3** - uchovává data; zaručuje jejich konzistenci

V aplikaci s třívrstvou architekturou probíhá veškerá komunikace přes aplikační vrstvu. Prezentační a datová vrstva spolu nemohou komunikovat přímo.

4.1 Prezentační vrstva

Prezentační vrstvu tvoří uživatelské rozhraní a komunikační rozhraní aplikace, kde s aplikací komunikuje koncový uživatel. Jejím hlavním účelem je zobrazovat informace uživateli a shromažďovat informace od uživatele. Tato nejvyšší vrstva může běžet například ve webovém prohlížeči, jako desktopová aplikace nebo grafické uživatelské rozhraní (GUI).



Obrázek 4.1 Architektura aplikace

4.1.1 Komunikační rozhraní

Prezentační vrstva komunikuje s aplikační vrstvou pomocí webového rozhraní REST API, které poskytuje aplikační vrstva. Komunikace je realizována pomocí technologie ASP.NET MVC a obsahuje několik samostatných kontrolerů, kdy každý z kontrolerů obsluhuje vymezenou část požadavků. Jednotlivé kontrolery se starají o obsluhu požadavku vztahující se k určité komponentě (událost, slučovací pravidlo, autentizace, atd). Komunikace probíhá pomocí standardního protokolu HTTP a tělo zprávy je reprezentováno formátem JSON.

4.1.2 Grafické rozhraní

Grafické uživatelské rozhraní (GUI) je rozhraní, jehož prostřednictvím uživatel komunikuje s elektronickými zařízeními, například počítače. Využívá různých ikon, nabídek a dalších vizuálních ukazatelů nebo grafických zobrazení. Grafická rozhraní graficky zobrazují informace a související uživatelské ovládací prvky. Na rozdíl od textových rozhraní, kde jsou údaje a příkazy výhradně v textu.

Grafická uživatelská rozhraní jsou ostatně vytvořena tak, aby byla dostatečně intuitivní a mohli je ovládat i relativně nekvalifikovaní pracovníci, kteří nemají žádné znalosti programovacího jazyka. Spíše než na strojovou orientaci jsou dnes standardem v programování softwarových aplikací, protože jejich návrh je vždy zaměřen na uživatele.

GUI je vytvořeno na privátních technologiích firmy TollNet a.s. Grafické prvky a další elementy vychází z privátních knihoven, které jsou vytvořeny pomocí jazyků HTML, CSS a javascript.

Webové grafické rozhraní prezentační vrstvy obsahuje dvě zobrazení. První zobrazení obsluhuje autentizaci uživatele v systému. Uživatel musí zadat své přihlašovací údaje, které se ověřují vůči externímu systému adresářových služeb (Active Directory). Druhé zobrazení obsluhuje požadavky na práci s hlavními entitami systému - událostmi

aplikací a slučovacími pravidly (viz 5.1.2).

Po úspěšné autentizaci do systému si může uživatel zvolit ze tří stránek pomocí bočního menu. První stránka řeší přehledy událostí aplikací, druhá stránka řeší přehledy slučovacích pravidel a poslední stránka umožňuje vyhledávání událostí dle provázaných úloh ze systému Jira.

Správa událostí

Stránka správy událostí vizualizuje nejaktuálnější události, které jsou evidované v systému. Nalézají se zde dvě přehledové tabulky.

První tabulku můžeme vidět na obrázku 4.2. Tato tabulka obsahuje přehled událostí dle nastavených filtrů. Bez nastavených filtrů se v tabulce chronologicky zobrazuje přehled nejaktuálnější událostí. Uživatel může zobrazené události filtrovat podle platformy projektu (GroupId), názvu události (Summary), časové značky (Last update), Modulu aplikace (Module), počtu výskytu (Count) a nebo stavu události (Status).

Events						
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Id	Group Id	Summary	Last update(UTC)	Module	Count	Status
838302	EMS	asng-testing-muo-*. ScrollId is null or empty	13.5.2022 23:35:51	EMS	9247	New
927384	ITBILLCZ	ExportOnlineRefundOrderItemPolling GetStatusCallback result=OK wcfMethodResult=[MethodResult=0x0900	13.5.2022 23:35:48	AR	899	InReview
854172	TACefbo	Could not find a part of the path 'Z:\Backup\EF\Images'.	13.5.2022 23:30:00	EF	29	New
854171	TACefbo	Could not find a part of the path 'Z:\Backup\EF\STR'.	13.5.2022 23:30:00	EF	29	New
929716	ITevin	AdjustedBill.FiscalVerificationNumber must not be empty and must consist of three strings delimited	13.5.2022 23:24:15	BE	26	InReview
925177	ITBILLCZ	WfGenerateBillsServiceLogic.ContinueWF method parameters	13.5.2022 23:15:12	BE	32	InReview
925178	ITBILLCZ	WfGenerateBillsServiceLogic.ContinueWF: Missing next step result=TransitionNotFound for result=Sysste	13.5.2022 23:15:12	BE	32	InReview
925176	ITBILLCZ	Missing next step result=TransitionNotFound for result=SystemError workflow=10 stepId=100 wfInstance	13.5.2022 23:15:12	BE	32	New
925175	ITBILLCZ	ReprocessBillSessionServiceLogic.ReprocessBillsInSession method parameters	13.5.2022 23:15:11	BE	32	InReview
925174	ITBILLCZ	ReprocessBillSessionServiceLogic.ReprocessBillsInSession: WrongBillSessionStatus expectedStatus=Repr	13.5.2022 23:15:11	BE	32	New
929563	ITevin	No such host is known. (fwproxy.infra.evin.si:3128) ---> System.Net.Http.HttpRequestException: No su	13.5.2022 23:13:01	AS	4	New
929562	ITevin	FileTransfer failed because=.	13.5.2022 23:13:01	AS	4	New
927665	TACutil	CalculateDailyConsumptionPollingServiceLogic.CalculateDailyConsumptionPolling method parameters	13.5.2022 23:10:19	BE	276	InReview
927664	TACutil	CalculateDailyConsumptionPollingServiceLogic.CalculateDailyConsumptionPolling: TryCalculateDailyCons	13.5.2022 23:10:19	BE	276	InReview
925166	ITBILLCZ	RateTollDataRecordServiceLogic.RateTollDataRecordPolling method parameters	13.5.2022 23:08:16	RE	545	InReview
925164	ITBILLCZ	ProcessRatedTollEventCzServiceLogic.ProcessRatedEventCz method parameters	13.5.2022 23:08:16	BE	545	InReview
925165	ITBILLCZ	RateTollDataRecordServiceLogic.RateTollDataRecordPolling: EndRating ProcessRatedEventCzCallback resu	13.5.2022 23:08:16	RE	545	InReview
925163	ITBILLCZ	ProcessRatedTollEventCzServiceLogic.ProcessRatedEventCz: ProcessRatedEventCz Callback fail	13.5.2022 23:08:16	BE	545	InReview
872828	FABILLCZ	ExportEetsMessagePollingServiceLogic.ExportEetsMessagePolling: InitializeExport failed with error re	13.5.2022 23:07:38	EDE	9119	InReview
872827	FABILLCZ	GetExportFileContent() failed for integrationLogId=92411 with errorMessage=File doesnt exist.	13.5.2022 23:07:38	EDE	9119	New

Obrázek 4.2 Přehled událostí v systému

Druhou tabulku můžeme vidět na obrázku 4.3. Tato tabulka vizualizuje detail konkrétní události. Nalézají se zde:

- pojmenované atributy logované události
- vazby na prolinkované úkoly v systému Jira (Linked issues)
- odkazy na obhobné události na jiných platformách (Similar events)
- seznam souvisejících událostí (Linked events)
- seznam historických SID (Service ID history)

Obrázek 4.3 Detail události

Správa slučovacích pravidel

Slučovací pravidla pomáhají redukovat množství zpracovávaných událostí a dále jsou popsány v kapitole 5.1.2. Aplikování aktivních slučovacích pravidel provádí systém implicitně, ale pouze na nově evidované události. Pokud existuje požadavek na aplikování slučovacího pravidla na již zpracované události, je nutné tento proces spustit explicitně administrátorem. Tento proces lze explicitně spustit pouze pro jedno konkrétní slučovací pravidlo. Pokud by se mělo explicitně aplikovat více slučovacích pravidel, bude proces spuštěn postupně pro každé pravidlo až po dokončení předchozího procesu. Konkurentní aplikace více slučovacích pravidel by v případě kolize nebo překrytí více slučovacích pravidel mohlo poškodit integritu procesu nebo databáze a následně ztratit dat.

Stránka pro správu slučovacích pravidel umožňuje správci vytvářet, editovat a monitorovat slučovací pravidla, která jsou v systému evidována. Přehledová tabulka (obrázek 4.4) zobrazuje stránkovaný seznam slučovacích pravidel v systému. Zobrazuje počet aplikování pravidla na evidované události, pseudo-regulární výraz, dle kterého se pravidlo aplikuje a informaci (sloupec Ignored), jestli je pravidlo v systému aktivní či nikoliv.

Grouping rules				
Id	Application count	Regular expression	Ignored	Priority
305	132	^RR013 \[internalId = \d+, threadId = \d+\] StReport rendering failed on exception.*	false	1
306	259	^RR013 \[internalId = \d+, threadId = \d+\] Report processing failed.*	false	1
307	0	^GetExportFileContent() failed for integrationLogId=\d+ with errorMessage=File doesnt exist.*	false	1
308	0	^File F5FCZ_1[0-9]-\.\TZ+\.\cof\,tmp cannot be copied to z:\Paywell\Output\F5FCZ_1\cof because a file with the same name already exists.*	false	1
309	20403	^ME\.(d+) \[MessageId = \d+\] Sending of the mail failed.*	false	1
310	20403	^ME\.(d+) \[MessageId = \d+\] Sending of the email failed -> rename the file back.*	false	1
311	0	^ParseImportFileTimestamp: cannot parse timestamp from file name Test_.*	false	1
313	156	^StimulsoftServiceLogic.PrecompileLayout: Template with id =\d+ cannot be downloaded from DB. Stored procedure ended with odpResult = ExecuteCommandError.*	false	1
314	0	^Temporary authentication failure. \[mailbox.etc.skytoll:[0-9]-\] \[0-9-\] \[NO\].*	false	1
315	784	^GuiDocumentManagementServiceLogic.GetDocumentStatus: document \d+ isn't generated.*	false	1
317	11693	^CodebookServiceLogic.GetValueDocGenerationAutoClean: No data for \[DocumentId: \d+ GenerationStyleCode: 1\].*	false	1
318	479	^d+-SendSmsBulkToInfobip error: InfobipApi.SendSmsBulk exception=.*	false	1
319	0	^GetExportFileContent() failed for integrationLogId=\d+ with errorMessage=File doesnt exist.*	false	0
320	357501	^Authorization failed for \[origSelId=\d+, origASId=\d+, origSelType=\[a-zA-Z\]+\], destSelId=, interfaceName=\[a-zA-Z\]+\], methodId=\d+, invokeId=.*	false	0
321	0	^ProcessEmailCallback() failed, result= \[A-Za-Z+\] methodResult= \d+, mailIndex = .*	false	0
322	2175	^d+ ServerTimeoutCollection: Deleting on timeout key=\[origSelId=\d+, destSelId=\d+, interfaceName=\[a-zA-Z\]+\], methodId=\d+, invokeId=.*	true	0
323	0	^Authorization failed for \[origSelId=\d+, origASId=\d+, origSelType=\[a-zA-Z\]+\] destSelId=, interfaceName=\[a-zA-Z\]+\], methodId=\d+, invokeId=.*	false	0
324	18179	^(\[A-Z\]+\).(d+) ServerTimeoutCollection: Deleting on timeout key=\[origSelId=, destSelId=\d+, interfaceName=\[a-zA-Z\]+\], methodId=\d+, invokeId=.*	true	0
325	0	^GetExportFileContent() failed for integrationLogId=\d+ with errorMessage=File doesnt exist.*	false	0
327	20	^Cannot find running job for serviceLogicId:.*	false	0

Obrázek 4.4 Přehled slučovacích pravidel v systému

Detail slučovacího pravidla (obrázek 4.5) umožňuje editaci, odstranění nebo aplikaci konkrétního slučovacího pravidla. Editací slučovacího pravidla můžeme měnit jeho pseudo-regulární výraz, platformu na kterou se aplikuje, prioritu při implicitní aplikaci a nebo jej zneaktivnit, viz obrázek 4.6. V případě požadavku na aplikaci pravidla se před spuštěním samotného procesu vyhodnocuje počet událostí, na které bude mít konkrétní slučovací pravidlo vliv. Jedná se o bezpečnostní mechanismus, který uživatele informuje o míře dopadů konkrétního slučovacího pravidla. V případě, že vyhodnocení dopadu slučovacího pravidla trvá příliš dlouhou dobu, je možné tento bezpečnostní mechanismus obejít a vynutit si explicitní aplikaci konkrétního slučovacího pravidla. Aplikace slučovacího pravidla je nevratný proces a proto je proces vytvoření, modifikace a aplikace slučovacího pravidla omezený pouze administrátorům systému.

Grouping rule			
Modify	Remove	Apply	Force apply
Close			
Id: 310		Group Id: 0	Priority: 1
Regular expression: ^ME\.(d+) \[MessageId = \d+\] Sending of the email failed -> rename the file back.*		Ignored: False	
		Application count: 20403	
ChangedOn: 14.5.2022 19:53:49		ChangedBy: Domanský Martin	
Modify	Remove	Apply	Force apply
Close			

Obrázek 4.5 Detail slučovacího pravidla

Grouping rule			
Save			
Close			
Id: 317		Group Id: ITeOBU	* Priority: 1
* Regular expression: ^CodebookServiceLogic.GetValueDocGenerationAutoClean: No data for \[DocumentId: \d+ GenerationStyleCode: 1\].*		Ignored: <input type="checkbox"/>	
ChangedOn: 14.5.2022 19:53:49		ChangedBy: Domanský Martin	
Save			
Close			

Obrázek 4.6 Editace slučovacího pravidla

Vyhledávání událostí

Při vývoji aplikace mohou nastat situace, kdy stejná chyba v aplikaci způsobuje mnoho různých logovaných chybových událostí. Tyto události nemusí mít mezi sebou přímou vazbu, například ServiceID, díky které by systém mohl vyhodnotit jejich souvislost. Nicméně v systému Jira může existovat úkol, který řeší zmíněnou chybu a tento úkol mají prolinkovány všechny události, které způsobuje zmíněná chyba. V takovémto případě je pro administrátora vhodné, aby si mohl zobrazit přehled událostí, které mají vazbu na konkrétní úkol v systému Jira.

Tato stránka slouží pro vyhledávání seznamu událostí, které mají prolinkovaný zadaný úkol ze systému Jira. Úkol je definován unikátním klíčem, který přiděluje systém Jira. Prolinkování úkolů ze systému Jira na evidované události je prováděno implicitně v procesu zpracování nových událostí nebo na explicitní vyžádání pro konkrétní logovanou událost, viz kapitola 5.1.

Stránku tvoří přehledová tabulka (obrázek 4.7), která zobrazí seznam událostí s vazbou na konkrétní úkol ze systému Jira. Jednotlivé položky obsahují stejné informace jako v přehledová tabulka na stránce správy událostí. Zobrazené události lze filtrovat a umožňují proklik na detail konkrétní události.

Id	Group Id	Summary	Last update(UTC)	Module	Count	Status
925157	ITBILLCZ	ExportEetsMessagePollingServiceLogic.ExportEetsMessagePolling: InitializeExport failed with error re	13.5.2022 22:32:44	EDE	29430	InReview
872828	FABILLCZ	ExportEetsMessagePollingServiceLogic.ExportEetsMessagePolling: InitializeExport failed with error re	13.5.2022 22:03:40	EDE	9113	InReview
808051	FABILLCZ	ExportEetsMessagePollingServiceLogic.ExportEetsMessagePolling: method parameters	13.5.2022 22:00:34	EDE	260409	InReview
461226	ITBOUCZ	ExportEetsMessagePollingServiceLogic.ExportEetsMessagePolling: InitializeExport failed with error re	13.5.2022 16:37:04	EDE	17186	InReview
829531	STBILLCZ	ExportEetsMessagePollingServiceLogic.ExportEetsMessagePolling: method parameters	13.5.2022 10:07:14	EDE	5358418	InReview
848411	STBILLCZ	ExportEetsMessagePollingServiceLogic.ExportEetsMessagePolling: InitializeExport failed with error re	13.5.2022 10:07:14	EDE	4113589	InReview
924888	ITbill	ExportEetsMessagePollingServiceLogic.ExportEetsMessagePolling: method parameters	13.5.2022 6:53:47	EDE	119	InReview
924889	ITbill	ExportEetsMessagePollingServiceLogic.ExportEetsMessagePolling: InitializeExport failed with error re	13.5.2022 6:53:47	EDE	119	InReview
926291	AUTOBILLCZ	ExportEetsMessagePollingServiceLogic.ExportEetsMessagePolling: InitializeExport failed with error re	10.5.2022 9:02:02	EDE	24	InReview
925640	STOBUZCZ	ExportEetsMessagePollingServiceLogic.ExportEetsMessagePolling: InitializeExport failed with error re	21.4.2022 8:42:37	EDE	24792	InReview
798831	STBILLCZ	OL RecalculateStockOperationCounterOneDay: stock_management_pkg.calc_stock_oper_counter_one day resul	13.10.2020 14:13:55	OL	3252	ReadyForDeploy
798832	STBILLCZ	OL: stock_management_pkg.calc_stock_oper_counter_one day after	13.10.2020 14:13:55	OL	3252	ReadyForDeploy
798833	STBILLCZ	RecalculateStockOperationCounterOneDayServiceLogic.RecalculateStockOperationCounterOneDay: stock_man	13.10.2020 14:13:55	OL	3252	ReadyForDeploy
798835	STBILLCZ	RecalculateStockOperationCounterOneDayServiceLogic.RecalculateStockOperationCounterOneDay: method par	13.10.2020 14:13:55	OL	3257	ReadyForDeploy
799810	STBILLCZ	Procedure OL: stock_management_pkg.calc_stock_oper_counter_one day ended with odpResultCode=Timeout	13.10.2020 14:13:55	AS	2556	ReadyForDeploy

Obrázek 4.7 Vyhledávání událostí dle klíče úkolu

4.2 Aplikační vrstva

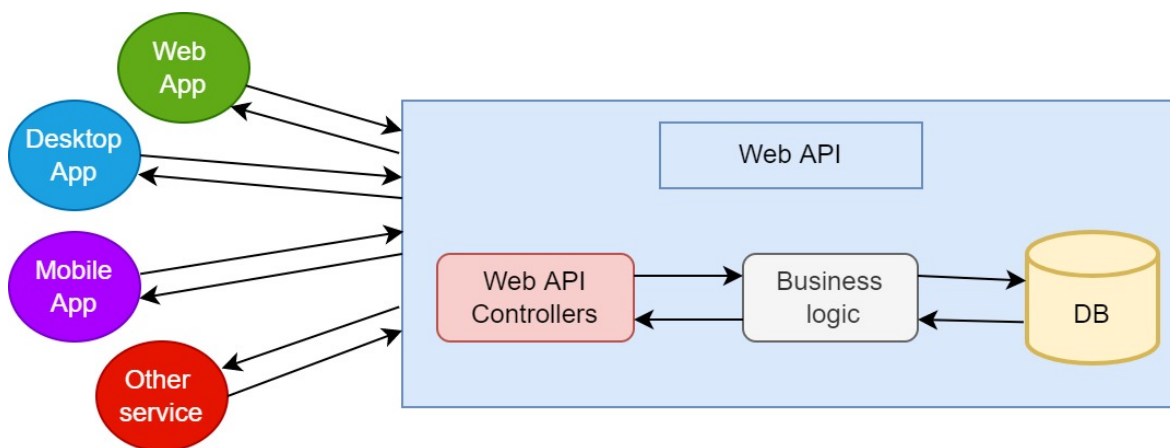
Aplikační vrstva, známá také jako logická vrstva, je srdcem aplikace. V této vrstvě se zpracovávají informace získané z prezentační vrstvy - někdy oproti dalším informacím v datové vrstvě - pomocí business logiky, tedy specifické sady business pravidel. Aplikační vrstva může také přidávat, mazat nebo upravovat data v datové vrstvě.

Nejvýznamnější částí je aplikační vrstva, která v sobě spojuje logiku a stěžejní funkce aplikace. Tato vrstva se stará o zpracování dat, logování, komunikaci s externími sys-

témy a také o komunikaci s ostatními vrstvami aplikace. Je implementována pomocí několika samostatných částí, které se obsluhují jednotlivé logické operace.

4.2.1 Webové rozhraní

Webové rozhraní je implementováno jako samostatná část aplikační vrstvy, která obsluhuje komunikaci aplikační vrstvy s prezentační vrstvou. Pro realizaci webového rozhraní bylo využito technologie ASP.NET Core MVC (viz 3.2), která umožňuje vytváření REST API služeb založených na protokolu HTTP. Implementované webové RESTful rozhraní je platformě nezávislé, proto s ním lze komunikovat pomocí jakékoliv platformy, jak lze vidět na obrázku 4.8.



Obrázek 4.8 Diagram Web API

Rozhraní REST API podporuje základní sadu CRUD operací, pomocí HTTP metod GET, PUT, POST a DELETE. Pro webové zdroje se využívá JSON reprezentace, protože jde o přehledný a jednoduchý datový formát. Jeho datová struktura je textový řetězec, což ulehčuje testování API. Webové rozhraní je bezstavové, proto nezáleží na předchozích dotazech a odpověď serveru vždy reaguje na konkrétní příchozí požadavek. Tento přístup je velmi vhodný pro paralelní zpracování dorazů na webové rozhraní. Každý dotaz je obsluhován v rámci individuálního kontextu, který vytváří framework ASP.NET.

Webové rozhraní obsahuje mnoho samostatných kontrolerů, kdy každý z kontrolerů obsluhuje vymezenou část požadavků. Jednotlivé kontrolery můžeme rozdělit do tří skupin podle entity, jejíž požadavky obsluhují.

Události

První skupina kontrolerů obsluhuje požadavky na události. Požadavky na API se mohou týkat konkrétní události nebo skupinu událostí.

Žádosti, které obsluhuje WebAPI pro konkrétní události:

- získání dat o události
- získání přehledu podobných událostí
- změna komentáře události
- změna stavu události
- opětovné vyhledání souvisejících úkolů v systému Jira
- vytvoření nového úkolu v systému Jira
- provázání události s konkrétním úkolem ze systému Jira
- zrušení provázání události s konkrétním úkolem ze systému Jira
- zrušení provázání události se všemi provázanými úkoly ze systému Jira

Žádosti, které obsluhuje WebAPI pro skupiny událostí:

- získání přehledu událostí dle daného filtru
- získání reportu o událostech dle daného filtru
- získání přehledu událostí mající vazbu na konkrétní úkol ze systému Jira
- odstranění skupiny událostí

Slučovací pravidla

Druhá skupina kontrolerů obsluhuje požadavky na slučovací pravidla. Požadavky na API se mohou týkat konkrétního slučovacího pravidla nebo skupinu slučovacích pravidel.

Žádosti, které obsluhuje WebAPI pro konkrétní slučovací pravidlo:

- získání dat o slučovacím pravidle
- získání přehledu událostí na které lze aplikovat konkrétní slučovací pravidlo
- aplikování konkrétního slučovacího pravidla
- vytvoření slučovacího pravidla
- změna vlastností slučovacího pravidla
- odstranění konkrétního slučovacího pravidla

Žádosti, které obsluhuje WebAPI pro skupinu slučovacích pravidel:

- získání přehledu slučovacích pravidel dle daného filtru
- odstranění skupiny slučovacích pravidel

Úkoly

Třetí skupina kontrolerů obsluhuje požadavky týkající se konkrétních úkolů ze systému Jira. Tato skupina obsahuje pouze jediný kontroler, který předává URL adresu pro konkrétní úkol ze serveru Jira identifikovaný pomocí jeho unikátního identifikačního klíče.

4.2.2 Business core

Business core je nejdůležitější část aplikační vrstvy. Stará se o komunikaci s externími servery a nachází se zde i business logika pro zpracování událostí aplikací. Mezi externími servery nalezneme server s databází Elasticsearch s normalizovanými logovanými daty, server pro správu úkolů Jira a privátní mail server pro odesílání emailových zpráv.

Pro business logiky jsou zde stěžejní tři procesy. První proces se stará o zpracování nových dat z Elasticsearch databáze a tyto data dále mapuje na evidované události aplikací. Druhý proces obstarává vyhledání souvisejících úkolů ze serveru Jira ke konkrétní události. Třetí proces řeší správu stavů jednotlivých událostí, reportování o událostech přidělenému správci a redukci a likvidaci nadbytečných u jednotlivých událostí. Procesy jsou dále detailněji popsány v kapitolách 5.1, 5.2 a 5.3.

Všechny tři procesy jsou navzájem nezávislé a v ideálním případě se navzájem nijak neovlivňují. U kritických sekcí, ve kterých by mohla nastat kolize jednotlivých procesů jsou využity zámky pro vstup do těchto kritických sekcí. Nastane-li situace, kdy chce více procesů vstoupit do kritické sekce, dochází za pomoci softwarových zámků k pozastavení toho procesu, který požádal o vstup do kritické sekce později. Praktický dopad této situace může být prodloužení doby cyklu pozastaveného procesu. K předcházení deadlocku se využívá timeout, který je specifický pro jednotlivé procesy zvlášť. Uvolnění a restartování procesu je realizováno pomocí CancellationToken, který si předávají volané asynchronní metody.

ElasticApi

Pro napojení na externí databázi Elasticsearch je implementováno ve třídě *ElasticApi*, které vnitřně využívá instanci třídy *ElasticClient* z knihovny *Nest*. Třída *ElasticApi* implementuje interface *IElasticApi*, které zveřejňuje asynchronní metodu *SearchLogsByScrollAsync* a kolekce názvů indexů v Elasticsearch pro jednotlivé platformy.

Asynchronní metoda *SearchLogsByScrollAsync* obstarává načtení dat z databáze Elasticsearch pro konkrétní platformu. Data, která se načítají lze omezit pomocí časové značky od - do a typu logovacích dat (konkrétně Severity).

JiraApi

Vytvořený systém umožňuje pro každou zpracovanou událost vyhledat informaci, jestli existuje nebo v minulosti existovat úkol, ve které se řešila problematika vzniklé události. Pro tuto funkcionalitu je nutné napojit se na vhodný systém pro správu úloh. Metody pro práci se systémem pro správu úloh popisuje interface *IIssueServerApi* a konkrétně pro server Jira jej implementuje třída *JiraApi*.

Třída *JiraApi* implementuje metody pro práci s úkoly uloženými v systému Jira. Pro komunikaci se systémem Jira se využívá externího RESTful webového rozhraní. Toto rozhraní poskytuje základní CRUD operace pro práci s úkoly v jednotlivých projektech.



Obrázek 4.9 Prolinkované úkoly s událostí

MailApi

Třída *MailApi* implementuje rozhraní pro komunikaci s externím mail serverem. Tato třída umožňuje systému odesílat emailové zprávy pomocí protokolu SMTP (Simple Mail Transfer Protocol). Odesílání zprávy se provádí asynchronně a systém nečeká na potvrzení doručení zprávy. Díky tomu odesílání zpráv nezatěžuje hlavní procesy systému. Funkcionalita SMTP protokolu není implementována ve třídě *MailApi*, ale využívá funkcionalit knihovny *System.Net.Mail*.

U emailových zpráv jsou podporovány všechny možnosti SMTP protokolu, tedy zprávu lze poslat v textovém formátu i formátu HTML. Je možné stejnou zprávu poslat více příjemcům a dokonce využít i skryté kopie. Zpráva může obsahovat i přílohy, tedy soubory navázané na emailovou zprávu. Omezení maximální velikosti přílohy nebo typu souboru řeší externí mail server.

IP Adresa externího mail serveru a port pro SMTP zprávy jsou minimální požadavky pro funkci odesílání emailových zpráv. Dále je možné nastavit SSL šifrování, přístupové údaje pro mail server nebo X509 certifikáty. Nastavení pro funkci odesílání emailových zpráv je konfigurovatelné a načítá se z konfiguračního souboru.

4.2.3 Databázové rozhraní

Databázové rozhraní je implementováno jako samostatná část aplikační vrstvy, která obsluhuje komunikaci aplikační vrstvy s datovou vrstvou. Pro realizaci databázového rozhraní byla využita knihovna *MDDM.DatabaseBase*, která se stará o fyzickou komunikaci s MSSQL databází pomocí příslušných protokolů a serializuje data získaná z databázového serveru. Tato knihovna umožňuje pro každou CRUD operaci provádět v samostatné databázové transakci a tím omezit přístupy do kritických sekcí. Data-

bázové transakce splňují vlastnosti ACID a tedy atomicitu operací, konzistenci dat, izolovanost operací a trvalost provedených změn.

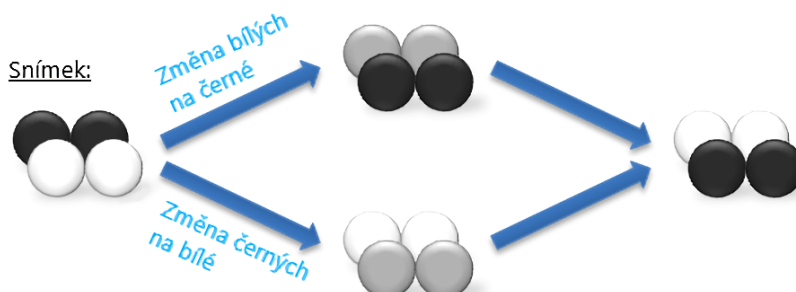
U databázových transakcí jsou povoleny izolace snímků. Tato funkcionality umožňuje vytvoření konzistentního snímku databáze při zahájení transakce. Všechna čtení prováděná v rámci transakce načtou hodnoty, které existovaly v době zahájení transakce. Aktualizovaná data po začátku transakce se neprojeví u operacích prováděných uvnitř transakce. Tento mechanismus eliminuje riziko nepotvrzeného čtení dat z databáze. Samotná transakce úspěšně potvrzená pouze tehdy, pokud žádná z aktualizací, které provedla, není v rozporu se souběžnými aktualizacemi provedenými od tohoto snímku.

Funkcionality izolace snímků je již běžné rozšíření v relačních i nerelačních databázích, například Oracle, MySQL, PostgreSQL, MongoDB i MSSQL. Hlavním důvodem pro používání izolace snímků je to, že umožňuje vyšší výkon než serializovatelnost paralelních zápisů dat. Izolace snímků se vyhýbá většině anomálií souběhu, které umí řešit serializovatelnost paralelních zápisů dat. V praxi je izolace snímků implementována v rámci řízení souběžnosti více verzí (MVCC), kde jsou udržovány generační hodnoty jednotlivých datových položek (verzí). MVCC je běžný způsob, jak umožnit vyšší paralelizaci a výkonnost tím, že při každém zápisu objektu databáze se vygeneruje nová verze objektu. Transakce mohou provádět operace čtení několika posledních relevantních verzí (každého objektu).

Serializovatelnost:



Snímek:



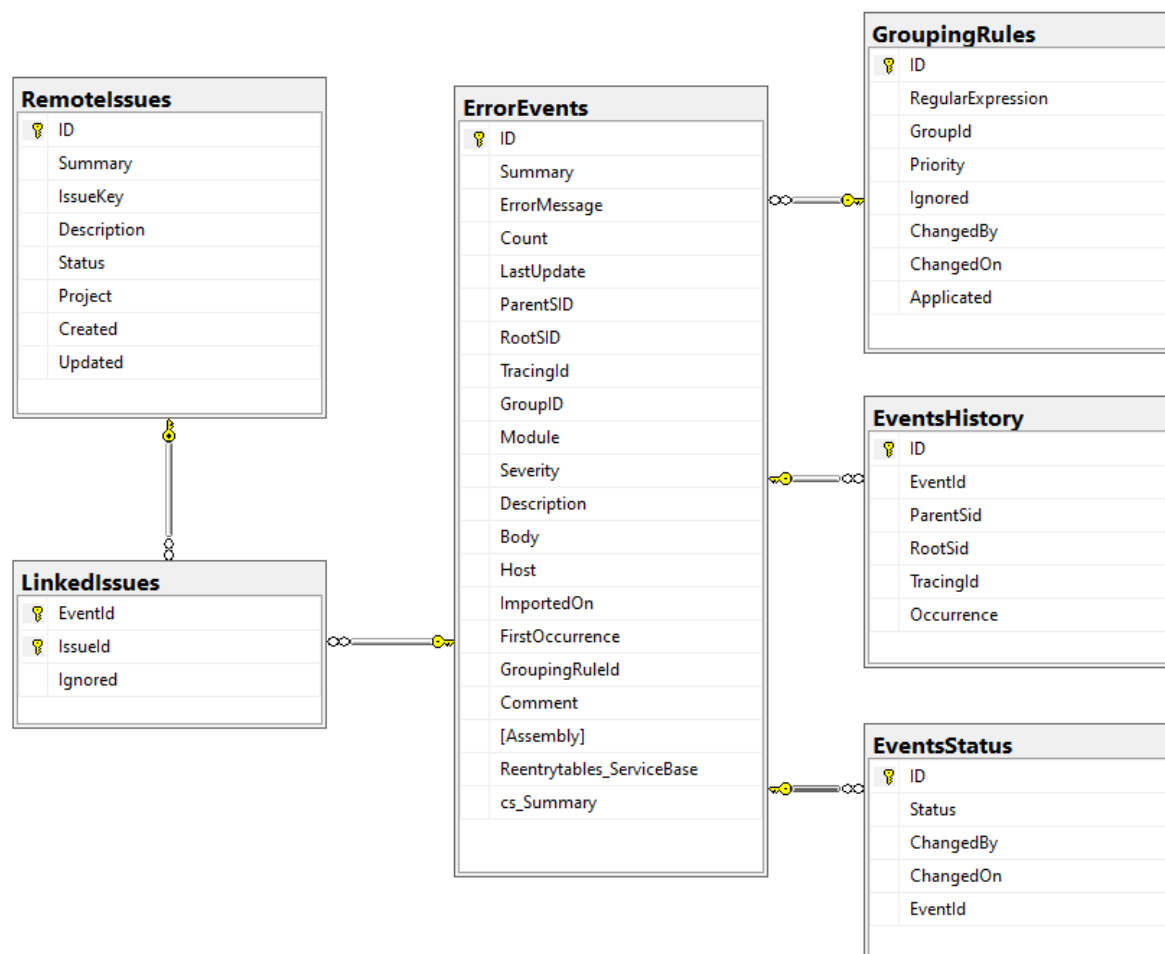
Obrázek 4.10 Serializovatelnost vs Izolace snímků

4.3 Datová vrstva

Datová vrstva, někdy nazývaná jako databázová vrstva, se stará o fyzický přístup k datům. Řeší se zde uložení a správa informací zpracovávaných aplikací. Může se jednat o relační systém správy databází nebo o databázový server. Pro uchování dat v databázové vrstvě je využíván Microsoft SQL Server, viz 3.3.

V databázi se ukládají tři hlavní entity: události (tabulka ErrorEvents), úkoly (tabulka RemoteIssues) a slučovací pravidla (tabulka GroupingRules). Kromě zmíněných tabulek v databázi existuje ještě vazební tabulka LinkedIssues, která propojuje evidované události a uložené kopie úkolů ze systému Jira.

Tabulka ErrorEvents obsahuje základní informace pro každou evidovanou událost v systému. Na tabulku ErrorEvents jsou skrze cizí klíče navázány tabulky EventsHistory, která eviduje historii ServiceId záznamů, a EventsStatus, která eviduje změny stavů dané události. Schéma databáze je popsáno na obrázku 4.11.



Obrázek 4.11 Schéma databáze

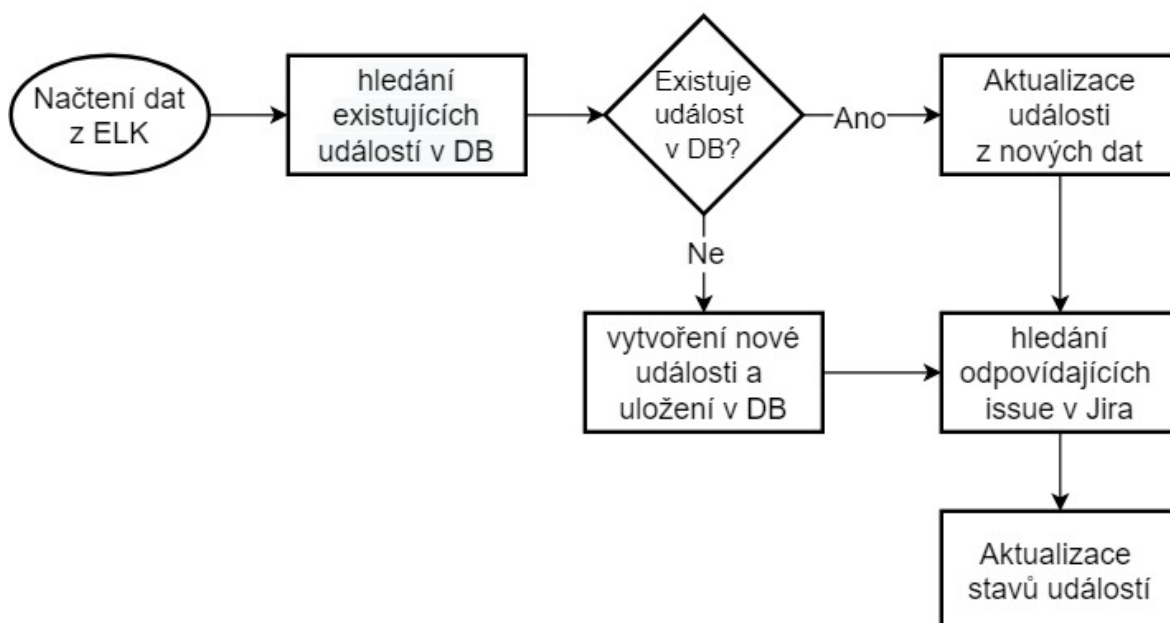
5 AUTOMATIZAČNÍ PROCESY SYSTÉMU

Automatizované zpracování logovaných dat je nejvíce podstatnou částí systému. Hlavní logika systému, nacházející se v aplikační vrstvě, obsahuje tři důležité procesy.

První proces automatizuje zpracování logovaných dat do evidovaných událostí, tento proces je dále popsán v kapitole 5.1. Druhý proces automatizuje zpracování úkolů, které jsou evidovány v systému Jira, tento proces je dále popsán v kapitole 5.2. Třetí proces automatizuje správu a reportování událostí, tento proces je dále popsán v kapitole 5.3.

5.1 Proces zpracování logovaných dat

Stěžejní částí práce je problematika zpracování dat. Na obrázku je znázorněn proces zpracování dat, který se periodicky opakuje každou hodinu. Proces zpracování logovaných dat implicitně spouští proces zpracování úkolů z Jira kvůli aktualizaci informací o dostupných úkolech v systému Jira, viz kapitola 5.2.



Obrázek 5.1 Proces zpracování dat

Na začátku procesu zpracování logovaných dat se nová data načítají z databáze Elasticsearch. Zde jsou data již uložena v normalizovaném stavu, viz kapitola 5.1.1. Elasticsearch si svá data ukládá v clusterech, které jsou identifikovány pomocí unikátních indexů. Pro každý index, ze kterého se mají zpracovávat data, si systém ukládá časovou značku. Časová značka pro jednotlivý index se aktualizuje hodnotou časové značky z posledního zpracovaného logovaného záznamu. U indexů, do kterých se logy, z více projektů, importují pomocí dávkových souborů, může nastat situace, kdy nově přidané logované záznamy mají starší časovou značku než je uložená aktuální časová

značka pro daný index. Taková situace by mohla způsobit ztrátu informací, protože logované data se starší časovou značkou by systém nikdy nezpracoval. Pro indexy s importovanými logovanými záznamy se kromě časové značky posledního logovaného záznamu ukládá také časová značka posledního importovaného záznamu.

Pro každý z načtených logovaných záznamů se v databázi dohledávají odpovídající slučovací pravidla (viz kapitola 5.1.2). Pokud slučovací pravidlo bylo již aplikováno na existující událost, použijí se data z logovaného záznamu pro aktualizaci dat dané události. Aktualizují se časové značky výskytu, *ServiceId*. a počítadlo výskytů dané události. V případě, kdy vyhovující slučovací pravidlo nebylo doposud aplikováno na existující událost nebo logovanému záznamu nevyhovuje žádné ze slučovacích pravidel, vytvoří se z dat logovaného záznamu nová událost a uloží se do systému.

Před následujícím krokem proces čeká na dokončení procesu aktualizace informací o dostupných úkolech v systému Jira. Následně se totiž pro všechny aktuální události dohledávají v Jira navázané úkoly. V tomto kroku se systém snaží pro každou událost nalézt takové úkoly, které obsahují části chybové hlášky nebo popisu chyby dané události. Pokud systém nalezne úkol, který má v názvu, popisu, přílohách nebo komentářích informace o dané události, implicitně se vytvoří vazba mezi úkolem a událostí.

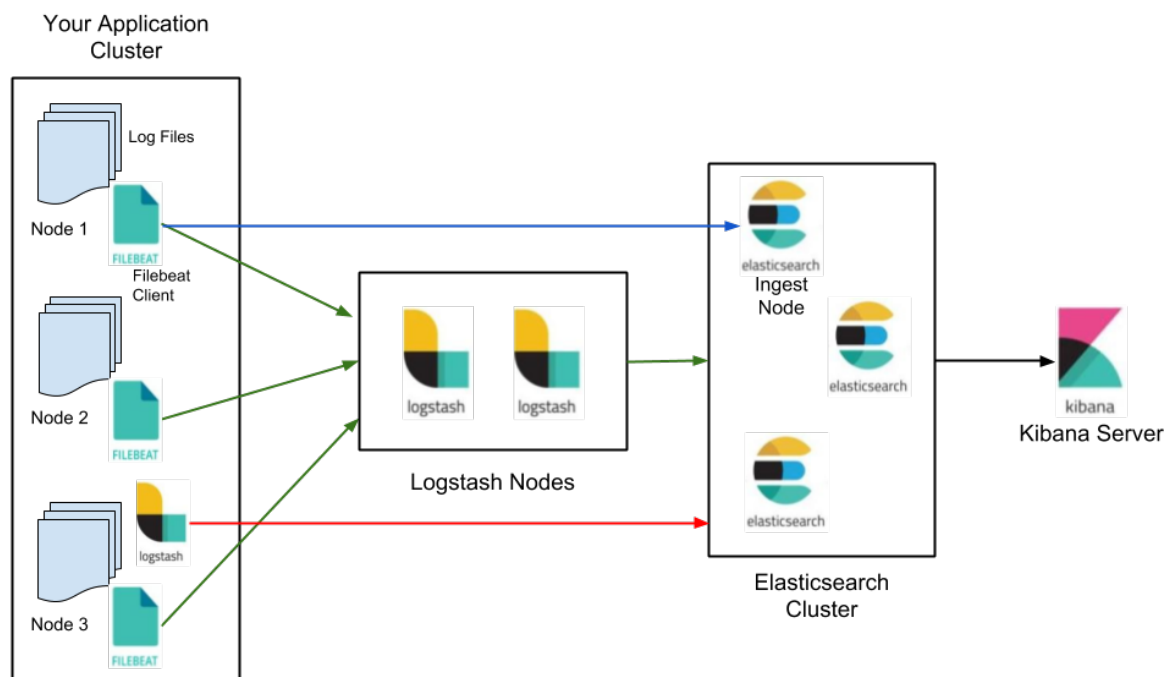
V posledním kroku se pro aktuální události vyhodnocuje aktualizace stavu. Nové události se zakládají se stavem *New*. Pokud se pro událost ve stavu *New* nalezne vazba na úkol v systému Jira, změní se stav události na *InReview*. Pro ostatní události události se aktualizace stavů neprovádí.

Logované data musí být před zpracováním normalizována a importována do databáze Elasticsearch. O import dat se může starat samotná aplikace přímo nebo mohou být importovány pomocí aplikace Logstash.

5.1.1 Normalizace dat

Logované data existují nejčastěji v podobě čistého textu (obrázek 1.1) a obsah logu nemusí mít vždy pevnou strukturu. Z tohoto důvodu je nutné před zpracováním dat provést normalizaci logovaných dat. Normalizaci dat může řešit aplikace Logstash, která se stará o parsování dat do strukturalizovaného formátu JSON. Poté jsou data uložena do Elasticsearch databáze, viz obrázek 5.2. Další možností je, aby aplikace zapisovala své logy přímo do databáze Elasticsearch. V takovém případě však musí být normalizátor logovacích dat součástí implementace generátoru logů.

Normalizace dat přináší benefit sjednocení různorodých formátů dat a také možnost rozšířit obsah logů o doplňující data aplikace, které nejsou součástí logů. Například se může jednat o identifikační kód aplikace, verzi sestavení, název hostitelského stroje a další informace o prostředí, ve kterém aplikace běží. Při normalizaci a importu dat se



Obrázek 5.2 Možnosti normalizace dat[13]

volí index klastru Elasticsearch databáze, dle příslušné konfigurace.

5.1.2 Slučovací pravidla

Proces systému, který se stará o načtení a zpracování nových dat, může v případě velkého množství vstupních dat trvat příliš dlouho. Nejjednodušším řešením nedostatku výkonu je horizontální nebo vertikální škálování. Vertikální škálování vyžaduje nový hardware nebo další zdroje zvyšující výkon systému. Výhodou vertikálního škálování je, že vyžaduje minimální zásahy na straně software. Nevýhodou mohou být případné komplikace s pořízením a instalací nového hardware. Horizontální škálování nevyžaduje nový hardware, ale oproti tomu vyžaduje značné zásahy na straně software. Oba způsoby škálování lze kombinovat, ikdyž většinou se tak neděje stejnou měrou. Horizontální i vertikální škálování má své limity. [19, 20].

Výše popsanou situaci řeší v systému cílená redukce a agregace vstupních dat, která vychází ze zpětné analýzy existujících událostí. Redukcí dat je snaha o eliminování duplicitních informací a nebo neukládání dat, které lze odvodit z jiných dat a informací. Při agregaci dat se využívá faktu, že stejná chyba většinou generuje opakující se chybové záznamy. Tyto opakované chybové záznamy sice obsahují jiné data, ale získané informace jsou vždy totožné. U opakujících se záznamů stejné události postačuje uchovat pouze nejnovější informace a počítat počet výskytů. Tento přístup lze jednoduše aplikovat na události, u kterých popis chyby neobsahuje proměnlivou složku. Pro

případy s proměnlivou složkou jsou vytvořeny slučovací pravidla.

Funkčně jsou slučovací pravidla realizována pomocí logiky regulárních výrazů [21]. Lze u nich aplikovat všechny typické syntaktické konstrukce:

- **Varianty** - '|' svislá čára, roura
- **Seskupování** - '()' kulaté závorky
- **Libovolný znak** - '.' tečka ; '[a-z0-9]' množina znaků ; '[^a-z0-9]' negace množiny znaků ; '\\kód' definované množiny znaků
- **Počet opakování** - '?' nepovinná část ; '*' žádný nebo libovolný počet opakování ; '+' jedno a více opakování ; 'm,n' yadaný počet opakování
- **Začátek a konec** - '^' pouze na začátku řetězce ; '\$' pouze na konci řetězce
- **Speciální znaky** - '\\' zpětné lomítko

Obecně existují tři typické použití slučovacích pravidel:

Prefix

Pro množinu záznamů, které obsahují proměnlivou složku pouze na konci textu události postačuje užití slučovacího pravidla jako prefixový výraz.

- "^Could not find file 'Z:\\Backup\\EF\\Images\\.*"

Výše zmíněné slučovací pravidlo se aplikuje na následující události.

- "Could not find file 'z:\\Excel\\Exports\\1210.csv"
- "Could not find file 'z:\\DMS\\Nevygenerovany_dokument_info.pdf'."
- "Could not find file 'z:\\AS\\X'.File name: 'C:\\AS\\X'"

Regulární výraz

Pro množinu záznamů, které neobsahují proměnlivou složku pouze na konci textu události, je nutné definovat slučovací pravidlo jako regulární výraz. Literály regulárního výrazu určují, které části textu jsou proměnlivou složkou.

- "^Performance: Normal - -> High CPU=[\\d\\.]+ DB CPU=[\\d\\.]+ AAS=[\\d\\.]+"

Výše zmíněné slučovací pravidlo se aplikuje na následující události.

- "Performance: Normal - -> High CPU=9.378 DB CPU=75.914 AAS=.0456"

- "Performance: Normal - -> High CPU=7.442 DB CPU=20.516 AAS=.0821"
- "Performance: Normal - -> High CPU=9.135 DB CPU=56.683 AAS=.061"
- "Performance: Normal - -> High CPU=5.273 DB CPU=100 AAS=.066"

Vzorový formát

Speciálním případem slučovacího pravidla je vzorový formát. V případě, kdy není žádoucí agregací dat přijít o informaci obsaženou v proměnlivé složce dat, lze definovat vzor slučovacího pravidla. Z tohoto vzoru budou generovány nová pravidla dle dostupných dat. Vzorový literál je definován jako regulární výraz, který je uzavřen do složených závorek.

- "^Authorization failed for \[origSelId=\d+, origSelType=[a-zA-Z\\.]+\]"

Výše zmíněné slučovací pravidlo se aplikuje na následující události.

- "Authorization failed for [origSelId=16778700, origSelType=MI.Service.Processing"
- "Authorization failed for [origSelId=50614759, origSelType=ME.Plugin.Wcf.Processing"
- "Authorization failed for [origSelId=22168626, origSelType=AS.Plugin.Wcf.ISelfCare"

Výsledkem bude vytvoření a aplikování následujících pravidel:

- "^Authorization failed for \[origSelId=\d+, origSelType=MI.Service.Processing"
- "^Authorization failed for \[origSelId=\d+, origSelType=ME.Plugin.Wcf.Processing"
- "^Authorization failed for \[origSelId=\d+, origSelType=AS.Plugin.Wcf.ISelfCare"

5.2 Proces zpracování úkolů z Jira

Proces zpracování úkolů z Jira se stará o procesní komunikaci se systémem Jira. Pro evidované události prohledává názvy, popisy, přílohy a komentáře v úlohách v systému Jira. Analýzou textu se proces snaží nalézt souvislosti mezi evidovanými událostmi a existujícími úlohami. Tento proces je implicitně spuštěn v rámci procesu zpracování dat, ale je možné jej explicitně spustit z grafického rozhraní, viz kapitola 4.1.2. Proces může běžet najednou pouze v jedné instanci. Pokud se sejde více požadavků na explicitní nebo implicitní spuštění procesu, jsou všechny požadavky ignorovány až do doby dokončení procesu. Ignorování požadavků na spuštění procesu je v tomto případě regulární postup, protože dokončením procesu vzniká aktuální obraz systému Jira. Navíc

rollback činností v rámci procesu a jeho restart není možný bez poškození integrity dat v databázi.

V původním záměru bylo, aby systém vždy pro každou událost mohl prohledat systém Jira a případně nalézt tam takové úkoly, které evidují vazbu na danou událost. S rostoucím počtem zpracovávaných událostí se násobně zvyšovalo vytížení systému Jira a způsobovalo to nedostupnost systému pro běžné uživatele. Z tohoto důvodu si proces stahuje nové úkoly ze systému Jira a aktualizuje ty existující. Tento proces prohledávání systému Jira si ukládá časovou značku posledního prohledávání systému Jira a nové prohledávání systému Jira zpracovává pouze změny v úkolech, které mají novější časovou značku *LastUpdate*. Časová značka procesu se aktualizuje podle nejnovější hodnoty *LastUpdate* z načtených úkolů.

5.3 Proces správy událostí

Proces správy událostí se stará o automatizaci zpracování událostí. Proces se spouští opakovaně jednou za den a plní účel noční údržby systému. Není možné explicitně vynutit jeho provedení, protože se jedná o implicitní údržbu systému. U událostí, které se v době běhu údržby nachází ve stavu *Removed*, se postará o bezpečné odstranění záznamů z databáze. Odstranění záznamů je trvalé, ale proces popsáný v kapitole 5.1 zvládne znovu vytvořit odstraněnou událost, pokud k tomu budou existovat patřičné data (data v Elasticsearch). Znovu vytvořené události neobsahují poznámky ani historii ServiceID, ale je možné u nich obnovit (znovu vyhledat) související úkoly v systému Jira.

Činnost procesu lze rozdělit do čtyř samostatných celků, které se provádějí postupně a izolovaně vůči sobě.

V prvním celku se ze systému odstraňují již nepotřebné události. Nepotřebné události jsou takové události, kterou se nacházejí ve stavu *Removed*. Do tohoto stavu se mohou události dostat dvěma způsoby. První způsob je explicitní změna stavu události, která je iniciována od uživatele nebo administrátora systému, skrze grafické rozhraní. Druhý způsob, jak se může událost dostat do stavu *Removed*, je implicitní nastavení systémem. To nastává ve chvíli, kdy se událost nachází ve stavu *Ready for deploy* a nelze dohledat nové úkoly v systému Jira, které by se nacházely v jímém stavu než *Closed*.

V druhém celku se vytvářejí denní či týdenní reporty. Obsah, frekvenci i adresáty reportů lze konfigurovat. Reporty se vždy posílají skrze email a zapisují se do spreadsheet dokumentu *xlsx*. Reportují se statistiky událostí pro dané platformy. Reportují se i události, u kterých brzy dojde k implicitnímu odstranění ze systému. Mezi reporty lze zahrnout i informace, které jsou potenciálně podstatné pro administrátory systému.

Třetí celek kontroluje existující události a vyhodnocuje, jestli je žádoucí tyto události dále uchovávat v systému nebo jestli se jedná o nepotřebné události a může u nich nastavit stav *Removed*. Nepotřebné události jsou takové, u kterých není možné dohledat navázané úkoly a uplynula minimální doba nutná pro retenci takových událostí v systému.

V posledním čtvrtém celku se ze systému odstraňují nadbytečná data událostí. Pro zpětnou analýzu událostí se uchovává v systému historie pouze určitého množství SessionId identifikátorů. Pokud se v systému nachází u některé z evidovaných událostí více historických záznamů, nadbytečné záznamy budou odstraněny. Dále se na evidované události, které nejsou ve stavu *Removed*, implicitně aplikují slučovací pravidla. Toto chování zde existuje pro situace, kdy slučovací pravidlo je přidáno do systému zpětně analýzou událostí, ale nebylo explicitně aplikováno. Implicitní aplikace slučovacích pravidel v procesu zmíněném v kapitole 5.1 nemá vliv na události již evidované v systému.

5.4 Události

Systém zpracovává logované data do událostí, se kterými dále pracuje a které vizualizuje. Existující události mohou odpovídat jednomu konkrétnímu záznamu v logovacím souboru, ale mohou obsahovat i data z více logovaných záznamů.

Události jsou navrženy tak, aby každá událost minimálně popisovala chybovou nebo jinou nestandardní situaci v chování systému. Teoreticky je možné, aby byly události vytvářené pro každý záznam v logu, ale přínos takového rozhodnutí bude minimální. Události tedy obsahují obecnou či detailní definici reportované chyby a dále obsahují informace, které lze využít k dohledání dalších souvisejících záznamů chování aplikace.

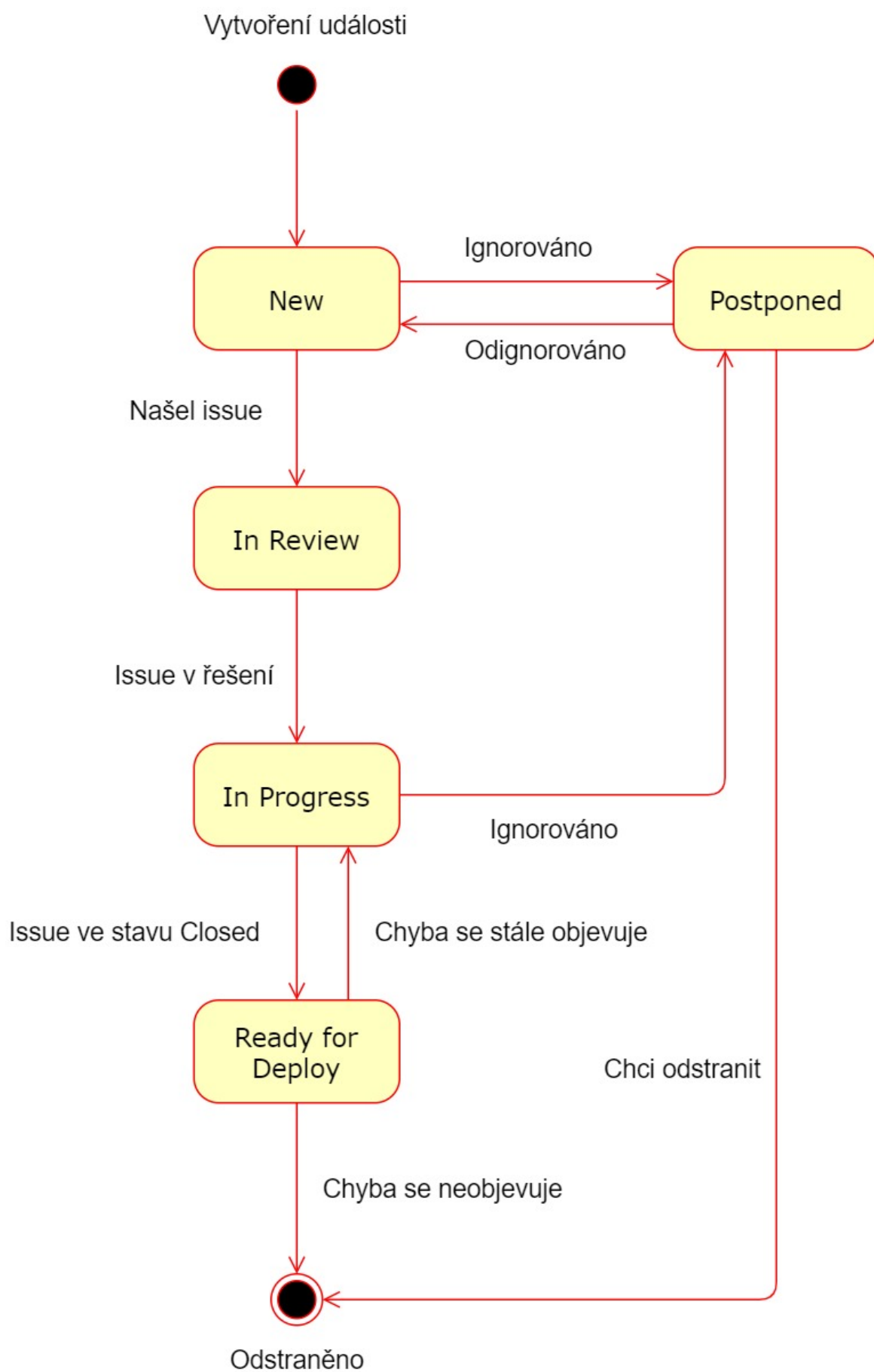
Každá událost si udržuje svůj vnitřní stav. Tento stav je důležitý pro uživatele i pro automatizované procesy v systému. Stavy jsou více popsány v kapitole 5.5. Pro každou událost lze vyhledat obdobné události, tedy události které jsou nejspíše způsobeny stejnou chybou, ale nacházejí se v jiném modulu nebo na jiné platformě. Pomocí informace o SessionId lze nalézt další události, které vznikly při stejném běhu aplikace a mohou mít na sebe vliv. To je důležitá informace kvůli tomu, že není nutné reportovat každou událost, ale postačuje vytvořit jeden úkol v systému Jira pro všechny související události. Pro každou událost je nejvíce podstatná kolekce souvisejících úkolů v systému Jira. Tato informace dává uživateli vědět, jestli je daná chyba již reportovaná a v jakém stavu se nachází řešení problému. Vizualizace události je zobrazena na obrázku 4.3.

5.5 Stavy událostí

Pro každou událost je stěžejní její vnitřní stav. Hodnoty stavů, které mohou události nabývat, a přechody mezi nimi názorně zobrazuje obrázek 5.3.

Nově vytvořené události se nacházejí ve stavu *New*. Pokud je automatickým procesem pro událost nalezen související úkol v systému Jira nebo je explicitně založen nový úkol, tak u takové události proběhne implicitně změna stavu na hodnotu *InReview*. Přejít ze stavu *InReview* na stav *InProgress* proběhne implicitně, pokud u dané události existuje alespoň jeden související úkol v systému Jira, který je aktuálně v řešení. Explicitně je možné z uživatelského rozhraní požádat o změnu u události ve stavu *InReview* na stav *InProgress*. Pro stavy *New* a *InProgress* existuje explicitní změna stavu události na stav *Postponed*, která může být iniciována z uživatelského rozhraní. Tento stav se využívá pro události, které se v systému často evidují, ale nezpůsobují chybu v systému ani nejsou způsobeny přímo žádným chybným chováním. Evidence takových událostí může být žádoucí z důvodu počítání jejich výskytu. Ze stavu *Postponed* je možné u události změnit stav na hodnotu *New* nebo událost nechat odstranit ze systému.

U událostí, které se nachází ve stavu *InProgress* a všechny související úkoly ze systému Jira jsou uzavřeny, proběhne implicitní změna stavu na hodnotu *Ready to Deploy*. Tento stav uživatele informuje, že evidovaná chyba je již zpracována a opravena. Pokud se nadále opakuje výskyt dané chyby v logovaných datech a pro danou událost bude nalezena nový neuzavřený úkol v systému Jira, proběhne implicitní změna stavu na hodnotu *InProgress*.



Obrázek 5.3 Stavový diagram událostí

6 TESTOVÁNÍ

Významnou součástí vývoje software je testování. Testování jako takové zdaleka není záležitost pouze hotového produktu, ale probíhá skrze celý vývojový proces. Při testování se snažíme odhalit programátorské chyby, které mohou vzniknout při psaní kódu. Dále se snažíme odhalit logické chyby, které vznikají chybným pochopením požadavku či nedostatečnou analýzou problému.

Ve fázi testování produktu sledujeme tři různé dimenze, které si dáváme za cíl testování. První dimenze se zaměřuje na to co testujeme za jednotku. Zde můžeme zahrnout jednotkové testy, integrační testy a systémové testy. Tyto testy ověřují, že úprava v systému neměla vliv na funkčnost dané jednotky a neovlivnila integritu systému ani dat. Druhá dimenze se zaměřuje na to, jaký aspekt testujeme. Zde se aplikují výkonové, bezpečnostní i funkční testy. Dle názvu je zřejmé, že zmíněné testy hodnotí zmíněné aspekty systému. Oproti jednotkovým testům se zde nehodnotí izolované funkce systému v bezpečném prostředí, ale klade se důraz na chování systému v reálném prostředí s reálnými hrozbami i obtížemi. Poslední třetí dimenze hodnotí, s jakým cílem testujeme daný systém. Zde zahrnujeme například regresní testy, které využívají funkčních i nefunkčních testů, aby odhalili, jestli provedené změny měly nějaký vliv oproti předchozí verzi systému. Můžeme zde zařadit i akceptační testy, které hodnotí, jestli chování systému odpovídá specifikaci. Akceptační testy se mohou zaměřovat i na testování určité jednotky systému, proto je můžeme zařadit do více skupin.

6.1 Automatizované testy

Nejběžnější automatické testy jsou jednotkové testy. Podpora jednotkových testů je integrovaná ve většině vývojových prostředích jazyka C# a díky tomu je jejich tvorba a spouštění jednoduché. Pro jednotkové testy je charakteristické, že kontrolují samostatnou komponentu systému, nikoliv celek jako takový. Díky jednoduchosti jsou jednotkové testy rychlé ve vyhodnocení daného testu. Jednotkové testy jsou izolované a nesdílí zdroje s jinými jednotkovými testy, proto je možné je pouštět paralelně napříč systémem. Paralelní běh více testů a rychlost jednotlivých jednotkových testů z nich dělá téměř ideální nástroj pro hodnocení stavu vyvíjeného systému.

S rostoucím množstvím pokrytí systému jednotkovými testy získáme přesnější přehled o systému. Budeme lépe znát, které metody se chovají dle očekávání a které nikoliv. Nevýhodou velkého množství jednotkových testů je rostoucí náročnost údržby jednotlivých testů. Výraznější změny v systému mohou mít dopady na velkém množství jednotkových testů, proto se hojně využívají pouze v počátcích vývoje. V pozdějších fázích vývoje se více spoléhá na integrační a systémové testy, které se lépe udržují aktuální.

Integrační a systémové testy ověřují, jestli vnitřní i vnější chování systému odpovídá specifikacím. Tyto testy většinou vznikají až v pozdější fázi vývoje, kdy jsou již definované komunikační rozhraní mezi komponenty a tyto komponenty jsou již ve stavu, kdy mohou spolu komunikovat.

Testování vytvořeného systému proběhlo na dvou úrovních. První úrovní byly automatizované testy, které vznikaly v době, kdy nebylo možné systém otestovat jako celek. Mezi automatizované testy byly zahrnuty i negativní jednotkové testy a nefunkční testy. Tyto sady testů ověřují, že jednotlivé metody a komponenty neobsahují chybné chování. Negativní testy mohou například testovat nedefinované stavy a hodnoty parametrů mimo obor hodnot. Druhou úrovní testů bylo ruční testování uživateli systému, které je dále popsáno v kapitole 6.2.

6.2 Uživatelské testování

Uživatelské testování bylo prováděno reálnými lidmi a bylo spojeno s testováním vůči specifikaci. Testování probíhalo v pozdější fázi vývoje, kdy změny v systému již nebyly tolik dynamické. Uživatelé se snažily testovat aplikaci obdobně jako koncový zákazník a využívalo se i nedeterministického chování uživatele v systému.

Největší nevýhoda uživatelského testování byla časová náročnost tohoto procesu. Tester se musel seznámit s úpravami oproti poslední verzi a do testovaného scénáře zařadit testování těch částí systému, na které mohly mít dopad provedené změny. Problematické byly také chyby, které se neodhalily při prvotním zanesení do systému a později komplikovaly další testování. Výhodou ručního testování bylo vytváření scénářů (sady úkonů), které kopírují typické chování uživatele v aplikaci. Scénáře bylo možné snadno vytvářet i měnit a navíc je mohl každý tester pochopit a provádět trochu jinak. Tím se docílilo širšího otestování více komponent systému a zároveň se ověřovalo, jestli je chování uživatele v systému dle předchozích analýz.

Pro testování byly využity reálné logované data z různých firemních i soukromých projektů. U soukromých projektů bylo možné integrovat funkcionalitu normalizace a importu dat do Elasticsearch databáze. Oproti tomu u firemních projektů bylo nutné pro normalizaci a import dat využít nástroj Logstash a tyto činnosti provádět mimo hlavní instanci projektu. Výsledky testování jsou popsány v kapitole 7.

7 DISKUZE

Tato kapitola se věnuje zhodnocení vytvořeného systému a ukazuje další možná využití, kterými je možné vytvořený systém rozšířit.

Výsledkem vývoje, popsanych v předchozích kapitolách, je systém, který umožňuje zpracovat logované data, vizualizuje informace získané z dostupných dat a automatizuje mnoho procesů, které byly původně doménou lidské práce. Plánovaným záměrem práce bylo demonstrovat pomocí automatizace procesů využití informačních technologií při správě a analýze logů a také vyzdvihnout neefektivnost tohoto procesu pokud je realizován pouze lidskými zdroji. Lidské zdroje jsou totiž omezené, nákladné a velice náchylné k chybovosti. Není však správné říct, že správu a analýzu logů lze jednoduše automatizovat a omezit vliv lidských zdrojů v tomto procesu.

Uživatelské testování a reálné nasazení systému bylo realizováno ve firmě, která se stará o vývoj software na zakázku. Existoval u ní citelný problém s nedostatkem lidských zdrojů na testování a analýzu chyb. Nedostatečné lidské zdroje a vysoké nároky na analýzu chyb zvyšovaly množství chyb, které byly reportovány z produkčního prostředí zákazníkem. Pozdější odhalení chyb zákazníkem způsobovalo časově náročnější analýzu problému a více časově zatěžovalo programátory. Důsledkem byla vyšší míra chyb oproti očekávání managementu, větší míra administrace prací na projektu a také dražší vývoj software.

Reálné nasazení systému vedlo ve zmíněné firmě k následujícím změnám:

- U projektů, u kterých to mělo smysl, se rozšířilo množství logovaných dat. Horší čitelnost logu neměla na automatizované zpracování logovaných záznamů vliv, ale více dostupných dat zvýšilo kvalitu analýzy evidovaných událostí.
- Zvýšilo se množství reportovaných chyb, protože systém zpracovával všechny reportované chyby a žádné nevynechával na rozdíl od lidského faktoru.
- Nárůstem počtu reportovaných chyb se ze začátku nasazení systému zvýšila zátěž programátorů. V brzké době se zátěž programátorů začala citelně snižovat. Chyby z úkolů se programátorům vracely dříve a programátor nemusel vynaložit tolik času jako předtím na asimilaci s reportovaným problémem.
- Zvýšila se kvalita analýzy chyb, protože testerům ubyla práce s vyhledáváním chyb. Více času bylo možné věnovat analýze problému. Reportované úkoly měly vždy standardizovaný styl (vytváření úkolů systémem) co vedlo k rychlejší nalezení podstatných informací v úkolu.
- Snížilo se požadované množství lidských zdrojů na analýzu chyb a nadbytečné lidské zdroje byly přeškoleny na tvorbu automatizovaných testů.

- Vyšší míra automatizovaných testů pomohla odhalit více chyb.
- Management získal nástroj, který vizualizuje míru a kadenci chyb na jednotlivých modulech a platformách.
- Snížilo se množství chyb, které byly reportovány z produkčního prostředí zákazníkem.
- Snížila se míra administrace na projektu, protože nebylo nutné tak často komunikovat se zákazníkem kvůli chybnému chování.
- Snížily se náklady na vývoj software.

Z výsledků testování a zpětné vazby reálných uživatelů bylo nasazení systému dle očekávání úspěšné a přispělo k vyšší produktivitě práce. Systém svou automatizací procesů nejen že snížil potřebnou míru lidské práce, ale i zvýšil potenciál lidí, kteří byli přeškoleni na jinou činnost s vyšší přidanou hodnotou a vyšším ohodnocením vykonané práce.

Zvýšení efektivity práce po nasazení systému měly vliv na zvýšení množství vyvíjeného software, což vedlo k vyšším nárokům na systém a vyšším objemům zpracovávaných dat. Pro další rozšíření systému bude nutné přepracovat architekturu vytvořeného systému a umožnit vyšší míru horizontálního škálování systému. Třívrstvá architektura umožňuje přepracování aplikační vrstvy do vhodnějšího architektonického stylu bez větších dopadů na zbylé části systému.

Pro vytvořený systém by bylo možné navrhnout a implementovat další prvky, které mohou zvýšit míru automatizace procesů, ale podstatnější jsou reálné požadavky na systém. Vytváření nových funkcionalit, které nemají vliv na reálné problémy uživatelů systému, jsou spíše bezpředmětné.

ZÁVĚR

Cílem této práce bylo vytvořit systém pro správu událostí aplikací, který pomocí vhodné vizualizace dat a automatizace procesů umožní zefektivnit detekci a analýzu aplikačních problémů. V práci jsem se snažil identifikovat a pochopit problémy, které snižují efektivitu zpracování logů, a implementovat opatření, které tyto problémy odstraňují nebo jejich dopady zmírňují.

Analýza možností ukládání záznamů událostí aplikací vedla k zavedení normalizace dat. Normalizace dat umožňuje ponechat formát logů jako prostý text, ale může být i ve formátu JSON či XML, pokud je to výhodnější. Podstatné je, aby byly logy ve strojově zpracovatelném formátu. Normalizací lze také doplnit dodatečné informace, které nejsou součástí logovacího souboru, protože by snižovaly čitelnost souboru pro administrátory. Například pomocí ID vlákna nebo SessionID, je možné vytvářet vazby mezi souvisejícími událostmi, které nejsou v logu zapsány lineárně. Vizualizace vazebních událostí pak pomáhá usnadnit analýzu problému.

Automatické prolinkování událostí do systému správy úkolů (Jira) umožní zjistit, jestli již existuje issue, které daný problém aktuálně řeší nebo v minulosti řešilo. Zakládání issue je také z části automatizováno a do vytvořeného issue se zapisují dostupné informace o dané události. Tabulková vizualizace dat s příslušnými filtry zjednodušuje přehled nad aktuálním stavem zvolené aplikace či modulu.

Výsledný systém zjednodušuje podstatnou část procesu zpracování logovaných dat aplikací. Je vytvořeno prostředí, které je interaktivní a pomáhá zefektivnit analýzu chyb. Rutinní činnosti, jako jsou zakládání nových issue, hledání existujících issue v Jira či zkoumání rozsahu a četnosti výskytu dané chyby jsou v systému vhodně automatizovány.

SEZNAM POUŽITÉ LITERATURY

- [1] Dybal, M.: Aspektově orientované programování – Logování. Největší český web zaměřený na .NET framework [online]. <https://www.dotnetportal.cz/blogy/16/Martin-Dybal/6395/Aspektove-orientovane-programovani-Logovani>, 2021, [cit. 2022-05-10].
- [2] Protokolování a trasování – .NET Core | Microsoft Docs. [online]. <https://docs.microsoft.com/cs-cz/dotnet/core/diagnostics/logging-tracingilogger-and-logging-frameworks>, 2021, [cit. 2022-05-10].
- [3] TIBCO: What is Process Automation? [online]. <https://www.tibco.com/reference-center/what-is-process-automation>, [cit. 2022-05-10].
- [4] A tour of the C# language [online]. <https://docs.microsoft.com/cs-cz/dotnet/csharp/tour-of-csharp/>, 2022, [cit. 2022-04-06].
- [5] Dokumentace k jazyku C# – začínáme, kurzy, referenční dokumentace [online]. <https://docs.microsoft.com/cs-cz/dotnet/csharp/>, 2021, [cit. 2022-04-06].
- [6] ASP.NET [online]. <https://en.wikipedia.org/wiki/ASP.NET>, 2022, [cit. 2022-05-10].
- [7] Overview of ASP.NET Core MVC [online]. <https://docs.microsoft.com/en-us/aspnet/core/mvc/overview>, 2022, [cit. 2022-05-10].
- [8] Fielding, R. T.: Architectural Styles and the Design of Network-based Software Architectures [online]. <https://www.ibm.com/cloud/learn/three-tier-architecture>, 2000, [cit. 2022-05-10].
- [9] Understanding APIs [online]. <https://www.ibm.com/cloud/learn/three-tier-architecture>, 2018, [cit. 2022-05-10].
- [10] Jira Software [online]. <https://www.atlassian.com/software/jira>, 2022, [cit. 2022-05-10].
- [11] DB-Engines Ranking of Search Engines [online]. <https://db-engines.com/en/ranking/search+engine>, 2022, [cit. 2022-05-10].
- [12] Radigan, D.: JQL: the most flexible way to search Jira [online]. <https://www.atlassian.com/blog/jira-software/jql-the-most-flexible-way-to-search-jira-14>, 2020, [cit. 2022-05-10].

-
- [13] Hunch, D.: Log shipping through ELK [online]. <https://www.digihunch.com/2018/09/log-shipping-through-elk/>, 2018, [cit. 2022-05-10].
- [14] Best of 2018: Log Monitoring and Analysis: Comparing ELK, Splunk and Graylog - DevOps.com. DevOps - The Web's Largest Collection of DevOps Content [online]. <https://devops.com/log-monitoring-and-analysis-comparing-elk-splunk-and-graylog/>, 2021, [cit. 2022-05-10].
- [15] Search everything, anywhere [online]. <https://www.elastic.co/enterprise-search/>, [cit. 2022-05-10].
- [16] Your window into the Elastic Stack [online]. <https://www.elastic.co/kibana/>, [cit. 2022-05-10].
- [17] Elastic Docs [online]. <https://www.elastic.co/guide/index.html>, [cit. 2022-05-10].
- [18] Three-Tier Architecture [online]. <https://www.ibm.com/cloud/learn/three-tier-architecture>, [cit. 2022-05-10].
- [19] Trend s názvem „horizontální škálování“: kdy má smysl a o co vlastně jde? [online]. <https://vshosting.cz/blog/trend-s-nazvem-horizontalni-skalovani-kdy-ma-smysl-a-o-co-vlastne-jde>, [cit. 2022-05-10].
- [20] Hunch, D.: Vertikální navýšení kapacity vs. škálování na více instancí [online]. <https://azure.microsoft.com/cs-cz/overview/scaling-out-vs-scaling-up/overview>, 2018, [cit. 2022-05-10].
- [21] Friedl, J. E. F.: *Mastering Regular Expressions*. O'Reilly Media, 2005, ISBN 978-0-596-00289-3.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

TCP	Transmission Control Protocol
UDP	User Datagram Protocol
XML	Extensible Markup Language
JSON	JavaScript Object Notation
BSON	Binary JSON
ASP.NET	Active Server Pages Network Enabled Technologies
MVC	Model-view-controller
.NET	Network Enabled Technologies
LINQ	Language Integrated Quer
SOAP	Simple Object Access Protocol
REST	Representational state transfer
gRPC	Google Remote Procedure Call
CLR	Common Language Runtime
API	Application Programming Interface
HTTP	Hypertext Transfer Protocol
SQL	Structured Query Language
CRUD	Create, Read, Update, Delete
JQL	Jira Query Language
ELK	Elasticsearch, Kibana, Logstash
GUI	Graphic User Interface
SID	Service ID
WebAPI	Webové API
SMTP	Simple Mail Transfer Protocol).
ACID	Atomic, Consistent, Isolated, Durable
ID	Identifikace

SEZNAM OBRÁZKŮ

Obr. 1.1.	Ukázka jednoduchého logu	13
Obr. 3.1.	Vzor MVC	18
Obr. 3.2.	Funkce REST API	20
Obr. 3.3.	Ukázka plánování úkolů v Jira [10]	21
Obr. 3.4.	Detail úkolu v Jira	21
Obr. 4.1.	Architektura aplikace	24
Obr. 4.2.	Přehled událostí v systému.....	25
Obr. 4.3.	Detail události.....	26
Obr. 4.4.	Přehled slučovacích pravidel v systému.....	27
Obr. 4.5.	Detail slučovacího pravidla.....	27
Obr. 4.6.	Editace slučovacího pravidla.....	27
Obr. 4.7.	Vyhledávání událostí dle klíče úkolu	28
Obr. 4.8.	Diagram Web API	29
Obr. 4.9.	Prolinkované úkoly s událostí	32
Obr. 4.10.	Serializovatelnost vs Izolace snímků	33
Obr. 4.11.	Schéma databáze	34
Obr. 5.1.	Proces zpracování dat.....	35
Obr. 5.2.	Možnosti normalizace dat[13]	37
Obr. 5.3.	Stavový diagram událostí.....	43

SEZNAM PŘÍLOH

P I. Obsah přiloženého CD

PŘÍLOHA P I. OBSAH PŘILOŽENÉHO CD

- `./doc/` (adresář) - elektronická verze této technické zprávy v $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$
- `./src/` (adresář) - zdrojové soubory aplikace