VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

HERNÍ ENGINE PRO ITIL TRENAŽÉR

AN ITIL SIMULATOR GAME ENGINE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. MARTIN PUČÁLKA**
AUTHOR

VEDOUCÍ PRÁCE **RNDr. MAREK RYCHLÝ, Ph.D.**
SUPERVISOR

BRNO 2018

Abstrakt

Tato diplomová práce je zaměřená na knihovnu Information Technology Infrastructure Library (ITIL). Cílem práce bylo analyzovat, navrhnout a implementovat herní engine, který by umožnil simulaci provozu IT služeb v reálném či zrychleném čase anebo v tazích. Základem enginu je tvůrčí mód, ve kterém je možné vytvářet vlastní služby a specifikovat jejich chování během provozu tak, jako kdyby je používali reální uživatelé. Druhou částí enginu je mód hráče a jednoduchý service desk. Pomocí nich se hráči mohou starat o plynulý provoz služeb, které jim byly svěřeny. To jim nabízí možnost praktické výuky a procvičování postupů popsaných v knihovně ITIL.

Abstract

This master's thesis is focused on Information Technology Infrastructure Library (ITIL). Objective of the project was to analyze, design and implement a game engine, which would provide simulation of IT service operation in real or accelerated time or in turns. Basic part of the engine is a creators mode, which allows users to create custom IT services and specify their behaviour during operation, like the service would be used in real. Another part of the engine is a players mode and simple service desk. In this mode, players can take care of fluent operation of their services. Thanks to this, they can learn and train practices, which are described in ITIL.

Klíčová slova

ITIL, engine, simulátor, hra, výukový software, provoz služeb, IT služba, service desk, web, workflow, Javascript, Node.js, Socket.io, MongoDB

Keywords

ITIL, engine, simulator, game, learning software, service operation, IT service, service desk, web, workflow, Javascript, Node.js, Socket.io, MongoDB

Citace

PUČÁLKA, Martin. *Herní engine pro ITIL trenažér*. Brno, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce RNDr. Marek Rychlý, Ph.D.

Herní engine pro ITIL trenažér

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana RNDr. Marka Rychlého, Ph.D.. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Martin Pučálka
21. května 2018

Poděkování

Rád bych poděkoval vedoucímu práce, RNDr. Marku Rychlému, Ph.D. za jeho rady, nápady a čas věnovaný konzultacím.

Obsah

Kapitola 1

Úvod

Pohled na informační technologie, jakožto na produkt dodávaný zákazníkům, prochází evolucí. Obchodní model, kdy je produkt nabízen jako komodita, je stále častěji nahrazován strategií, kdy je nabízeným produktem služba. Tento trend je patrný například u software, kdy stále více výrobců mění způsob licencování svých programů. Zaplacení jednorázového poplatku je nahrazováno pravidelnými paušálními poplatky. Klasické "krabicové"verze produktů jsou vytlačovány cloudovými službami. Dokonce i hardware, který se jeví čistě jako zboží, je nyní nabízen formou služby. U vybraných modelů prodejci nabízí možnost kdykoliv zboží vrátit, garantují expresní servis či pravidelnou výměnu stávajícího modelu za modernější. Tento obchodní model hraje ještě větší roli v obchodním sektoru, kde zákazníkem může být například průmyslová firma. Nabízí možnost outsourcingu celých IT odvětví a s tím spojené výhody. Zejména jde o možnost soustředit se na hlavní činnost podniku, rychlé nasazení řešení, eliminaci rizik spjatých s vlastním řešením, snížení operativních nákladů, predikovatelné nebo kontrolovatelné výdaje.

Tato práce se zabývá knihovnou ITIL, která je zaměřena právě na řízení IT služeb. Cílem práce je vyvinout herní engine, který umožní vytvářet simulace provozu služeb tak, jak jej knihovna popisuje. V rámci práce bude vytvořen i jednoduchý service desk, který je nedílnou součástí provozu služeb a pomocí něhož budou moci hráči vytvořené simulace obsluhovat. Práce tak částečně navazuje na diplomovou práci ITIL trenažér se snahou zaměřit se zejména na provoz služeb, který je pro tvorbu simulace nejlépe uchopitelný. Má přinést rozšíření simulace o běh v reálném či zrychleném čase, možnost zasílání oznámení service desku na email nebo třeba jednodušší definici chování služby.

Kapitola 2 se věnuje představení knihovny, její historii a postupnému vývoji. Uvedeny jsou zde její hlavní principy a výhody, které její zavedení přináší. Stručně popsáno je všech pět hlavních částí aktuální verze této knihovny včetně hlavních procesů. S ohledem na téma práce je největší pozornost věnována fázi provozu služeb. V kapitole 3 jsou popsány principy workflow a významné nástroje pro jejich modelování. Část 4 je věnována webovým aplikacím, zejména platformě Node.js. Kapitola 5 popisuje existující řešení her či simulátorů zaměřených na ITIL a specifikuje požadavky na vlastní řešení. Kapitola 6 představuje analýzu a návrh realizace vlastní aplikace. V kapitole 7 následuje stručný popis zvolených technologií a seznámení s implementací. Testování a vyhodnocení výsledné aplikace je popsáno v kapitole 8. V závěru (kapitola 9) jsou potom shrnuty dosažené cíle práce, výsledky vyhodnocení a možnosti dalšího vývoje.

Kapitola 2

ITIL - Information Technology Infrastructure Library

Název ITIL je zkratkou pro Information Technology Infrastructure Library. Jedná se o sbírku knih, které poskytují soubor nejlepších praktik pro správu a řízení služeb IT.

Službu lze chápat jako prostředek pro poskytování hodnoty zákazníkům prostřednictvím výstupů, kterých zákazník chce dosáhnout bez vlastnictví specifických nákladů a rizik.[5] Služba IT je založena na informačních technologiích a jejím účelem je poskytovat podporu obchodních služeb, procesů či funkcí zákazníka. Zákazníky, tedy odběratele služeb, lze rozlišit na interní či externí. Dle významu služby pro zákazníka se rozlišuje, zda se jedná o službu orientovanou na zákazníka, která přímo podporuje jeho obchodní procesy nebo o tzv. podpůrnou službu, která není spjata přímo s podnikáním zákazníka, ale je důležitá jako podpora služeb orientovaných na zákazníka.[5] Mimo zákazníka se rozlišují také pojmy uživatel a dodavatel. Uživatel je osoba, která službu přímo využívá. Uživatel a zákazník mohou být rozdílné osoby. Dodavatelem je potom třetí strana, která je zodpovědná za poskytování služeb. Příkladem služeb mohou být například provoz ERP systému, webových stránek, servis hardwaru nebo poskytování internetového připojení.

Za službu lze tedy označit širokou škálu činností. ITIL je obecný rámec, který neobsahuje konkrétní řešení. Naopak je dostatečně obecný a flexibilní, aby jej bylo možné aplikovat na různá odvětví IT. Při jeho implementaci často není nutné zavádět všechny zmíněné procesy a aplikovat vše, co ITIL nabízí. Mnohem důležitější je znát cílovou oblast, respektovat požadavky zákazníků a poskytovatelů služeb a z knihovny ITIL využít jen to, co je v daném případě užitečné.

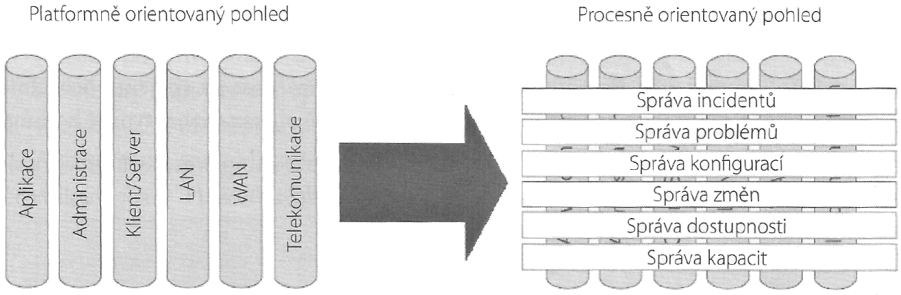
Často uváděným přínosem zavedení doporučených praktik je vyhnutí se tzv. znovu vynalézání kola, což vede ke snížení nákladů, zvýšení efektivity a účinnosti při poskytování služeb. Dalšími výhodami jsou transparentnost nákladů, kontrolovatelnost výsledků pomocí definovaných metrik, orientace služeb na potřeby obchodních procesů a zlepšení komunikace, což ve výsledku přináší vyšší kvalitu a spokojenost zákazníků s poskytovanými službami.[4]

2.1 Historie a obecné informace

Vznik knihovny sahá do 80. let a vychází ze snahy Velké Británie snížit náklady na IT. Za jejím prvním vydáním stála britská agentura CCTA (Central Computer and Telecomunications Agency). Zpočátku tato knihovna čítala přes 40 knih a v té době nabízela

nejobsáhlejší popis procesů pro výstavbu organizace služeb IT. Původně se zaměřovala na optimalizaci dosavadních procesů dle nejlepších praktik. V následujících letech pak prošla postupným vývojem, kdy byla přizpůsobována potřebám průmyslu a dynamicky měnícímu se odvětví IT.[4]

To se promítlo do vývoje druhé verze knihovny, označované jako ITIL V2, který započal koncem 90. let pod vedením CCTA. Z původního vládního úřadu CCTA vznikla v roce 2000 Office Government Commerce (OGC), která je vlastníkem knihovny pod názvem Cabinet Office ITIL dodnes. Během vývoje bylo vydáno několik titulů, z nichž největší úspěch zaznamenal prvně vydaný Service Support. Vývoj druhé verze byl završen vydáním poslední publikace v roce 2006.[1] Významnou změnou byl nový pohled na správu služeb, který přešel od platformně orientovaného k procesně orientovanému.

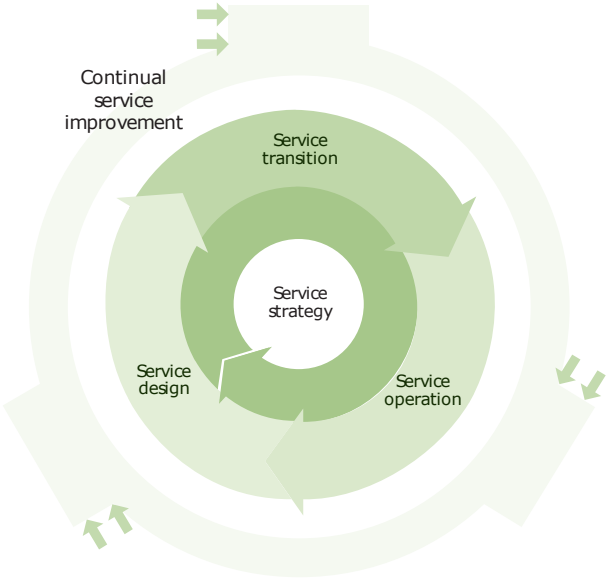


Obrázek 2.1: Znázornění přechodu od platformně orientovaného pohledu k pohledu orientovanému procesně. [4]

V červenci 2007 byl zveřejněn ITIL V3, který je dosud poslední verzí. Od roku 2011 je dostupná ITIL Edice 2011. Jedná se o aktualizaci třetí verze, která přináší sjednocení terminologie, větší konzistenci a srozumitelnost popisovaných oblastí.

Během svého dlouhého vývoje se z něj stal prakticky standard, který odkazuje na řadu dalších rámců, metod či standardů, mezi které patří například standard ISO 20000, metoda správy kvality Six Sigma nebo metoda pro správu projektů PRINCE2.

V současnosti tvoří základ knihovny pět publikací, které mapují celý životní cyklus služby – od jejího vzniku přes návrh, nasazení, provoz až po vyřazení služby z portfolia poskytovaných služeb. Jsou v nich popsány procesy řízení IT služeb, jejich návaznosti, role a zodpovědnosti v těchto procesech, metriky vyhodnocení a také návody pro průběžné zlepšování procesů. Mimo tyto klíčové publikace existuje také množství dodatečných materiálů, označovaných jako Complementary Guidance, které se zabývají například uvedením do problematiky řízení služeb, nabízí rady pro speciální odvětví apod.[4]



Obrázek 2.2: Schéma zobrazuje jednotlivé fáze ITIL a jejich návaznosti. [5]

2.2 Strategie služeb

První z pěti hlavních publikací se zabývá fází nazývanou Strategie služeb. Tato fáze se prolíná celým životním cyklem služeb. Definuje jakým způsobem plánuje poskytovatel nabízet své služby, aby dodal požadované výstupy svým zákazníkům, a tím naplnil své cíle. Klade se zde důraz na to, aby poskytovaná služba přinášela zákazníkovi hodnotu a poskytované služby byly mapovány na zákazníkovy výstupy.

V souvislosti s tímto mapováním se zde zavádí pojem portfolio služeb. Jedná se o repositář informací o všech službách organizace, který se základně dělí na frontu služeb, katalog služeb a vyřazené služby. Fronta služeb obsahuje služby, které zatím nejsou aktivní. Jsou například jen plánované nebo ve vývoji. V katalogu služeb jsou aktuálně nabízené služby. Na rozdíl od ostatních kategorií bývá tento katalog dostupný pro zákazníky. Do kategorie vyřazených služeb pak spadají ty služby, jejichž poskytování je už ukončováno. Dělení však může být i podrobnější. Cílem je mít přehled o nabízených službách organizace (minulých, současných i budoucích) a díky tomu lépe pokrýt potřeby zákazníků na trhu, mít přehled o konkurenci, možnostech cenové politiky apod.[20]

Ve fázi Strategie služeb definuje ITIL pět stěžejních procesů [24], [2]:

Strategické řízení má za úkol posoudit nabídky poskytovatele, schopnosti, konkurenci a tržní potenciál a na základě toho vyvinout strategii pro poskytování služeb zákazníkům. Proces je také zodpovědný za zajištění plnění této strategie.

Správa portfolia služeb je proces, který má na starosti správu služeb a jejich průchod portfoliem služeb. Od jejich financování, návrhu, vývoje a testování ve frontě služeb, přes kontrolu očekávaných výsledků v katalogu služeb až po konec životnosti služby a přechod do kategorie vyřazených služeb. Proces zajišťuje, že poskytované služby jsou v souladu se strategií řízení služeb. Umožňuje dynamicky a transparentně řídit investice do zdrojů. Cílem je maximalizovat poskytovanou hodnotu a současně řídit rizika a náklady.

Správa financí se stará o účtování (Accounting), rozpočtování (Budgeting) a zpoplatnění (Charging) – tzv. ABC správy financí. Účtování odpovídá na otázku, kolik stojí provoz jednotlivých služeb. Rozpočtování zajišťuje dostupnost finančních prostředků na provoz smluvených služeb a zpoplatnění má za úkol srozumnění zákazníků s poplatky za poskytované služby.

Správa požadavků má na starosti porozumění, předvídání a ovlivnění zákazníkových požadavků na službu. Ve spolupráci se Správou kapacit zajišťuje schopnost jejich plnění.

Správa obchodních vztahů udržuje vztahy se zákazníky. Analyzuje požadavky stávajících a potenciálních zákazníků a zajišťuje, aby byly vyvíjeny služby tak, aby těmto požadavkům vyhovovaly.

2.3 Návrh služeb

Účelem této fáze je návrh a tvorba hodnotných služeb v souladu se strategií služeb. Služby by měly být navrženy efektivně tak, aby v průběhu jejich životního cyklu byla požadována jen minimální vylepšení.

Při návrhu služby hraje důležitou roli tzv. dohoda o úrovni služeb (Service Level Agreement, SLA). Jedná se o písemnou dohodu mezi poskytovatelem služeb a zákazníkem. Měla by obsahovat srozumitelné a jednoznačné informace ohledně poskytované služby, mezi které patří například popis služby, ujednání o dostupnosti služby, výkonu, uživatelské podpoře, nacenění apod.

Informace o hlavních procesech popsaných v této fázi byly čerpány z [10] a [24].

Koordinace návrhu je proces zajišťující konzistenci a efektivitu návrhu služeb, informačních systémů pro správu služeb, architektur, technologií, procesů, informací a metrik. Důležitá je koordinace všech aktivit v rámci návrhu pomocí projektů, změn, dodavatelů a podpůrných týmů. Součástí je i řízení plánování, zdrojů a dovedností včetně správy konfliktů.

Správa katalogu služeb má na starosti vytvoření a údržbu katalogu služeb. Zodpovídá za to, že se v něm nachází přesné informace o všech provozovaných službách včetně těch, které jsou připraveny na uvedení do provozu. Proces poskytuje aktuální informace pro všechny ostatní procesy řízení služeb: detaily služeb, aktualní stav, závislosti služeb atd.

Správa úrovně služeb je zodpovědná za evidenci požadavků na služby, zejména za jejich smluvní ukotvení v podobě dohody o úrovni služeb (Service Level Agreement, SLA). Požadavky zákazníků se mohou měnit nebo vznikat nové. Důležité je, aby byly sjednané požadavky plněny, proto tento proces dále zajišťuje průběžnou kontrolu dohodnuté úrovně služeb.

Správa dostupnosti má za úkol definovat, analyzovat, plánovat, měřit a vylepšovat všechny aspekty dostupnosti služeb. Dostupnost je spolu se spolehlivostí a udržovatelností jedním z měřitelných aspektů. Lze je vypočítat následovně[20]:

$$dostupnost = \frac{sjednaný\ čas\ dostupnosti\ služby - čas\ výpadku\ služby}{sjednaný\ čas\ dostupnosti\ služby} * 100$$

$$spolehlivost^{1} = \frac{dostupnost\ v\ hodinách - čas\ výpadku\ služby\ v\ hodinách}{počet\ přerušení}$$

$$udržovatelnost = \frac{čas\ výpadku\ služby\ v\ hodinách}{počet\ přerušení}$$

Proces se pomocí proaktivních či reaktivních činností stará o to, že jsou tyto ukazatele na úrovni smluvené se zákazníkem.

Správa kapacit zabezpečuje, že kapacity služeb a infrastruktury IT splňují smluvené požadavky. K úkolům tohoto procesu patří například vytvoření informačního systému správy kapacit, plánu kapacit nebo provádění proaktivních měření za účelem zlepšení výkonnosti.

Správa kontinuity služeb IT má na starosti rizika, která by mohla mít vážný dopad na poskytování služby. Musí dohlížet na to, aby byl poskytovatel vždy schopný dodávky služby s minimálními odsouhlasenými parametry. Docílit toho lze redukcí vysokých rizik na přijatelnou mez nebo tvorbou plánů pro případnou obnovu dodávky služeb na původní úroveň.

Správa bezpečnosti informací dbá na zachování důvěrnosti, integrity a dostupnosti informací, dat a IT služeb. Činnosti v rámci tohoto procesu by se měly řídit komplexní bezpečnostní politikou, která by měla podléhat pravidelné revizi.

Správa dodavatelů zajišťuje, že všechny kontrakty s dodavateli podporují obchodní potřeby a že všichni dodavatelé dodržují své smluvní závazky.

2.4 Přechod služeb

Přechod služeb má za cíl plánovat a řídit změny služeb účinně a efektivně, spravovat rizika týkající se nových, měněných či ukončovaných služeb a zajistit, že prováděné změny vytváří očekávanou hodnotu pro zákazníka. V [4] jsou popsány následující procesy:

Plánování a podpora přechodu Účelem procesu je poskytnout celkové plánování pro přechod služby a koordinace potřebných zdrojů.

Správa změn je proces správy životního cyklu změn. Změnou je myšleno přidání, modifikace či odstranění služby nebo některé z jejích komponent. Životní cyklus změny zahrnuje například požadavek na změnu, analýzu rizik, schválení, naplánování, koordinaci provedení či přezkoumání výsledků změny. Proces má za úkol minimalizovat počet výpadků a incidentů zapříčiněných prováděním změn. Změny by měly probíhat rychle a koordinovaně dle standartizovaných postupů.

[1] průměrný čas mezi selháními

Správa aktiv a konfigurací má poskytovat spolehlivé informace o aktivech potřebných pro poskytování služeb. Důležitým prvkem je zde logický model IT infrastruktury složený z konfiguračních položek, jejich vlastností a vztahů mezi nimi. Konfigurační položka (Configuration Item, CI) je definována jako jakákoliv komponenta, která musí být spravována kvůli dodávce IT služby. Typicky jde o IT službu jako takovou, potřebný hardware, software, budovy, personál či dokumentaci.

Správa releasů a provozního nasazení plánuje, řídí a kontroluje přesun release do testovacího a provozního prostředí. Release (označován jako release balíček) je sada schválených změn IT služby. Může tedy obsahovat hardware, software, dokumentaci, procesy a další komponenty potřebné k implementaci změn. Cílem procesu je vytvořit, otestovat, verifikovat a nasadit release balíček do provozu.

Validace a testování služby zabezpečuje, že nové a změněné služby splňují zákazníkova očekávání. Testují se vlastnosti, přinášející zákazníkovi hodnotu, tedy záruka a použitelnost.

Vyhodnocení změn posuzuje velké změny nebo nové služby předtím, než jsou implementovány. Mělo by zahrnovat zhodnocení zamýšlených účinků změny, předvídání vedlejších účinků změny, identifikaci rizik apod.

Správa znalostí je proces, který má za úkol shromažďovat, analyzovat a sdílet data, znalosti a informace napříč organizací. Tomu může dopomáhat systém správy znalostí o službách. Proces má budovat znalostní bázi organizace, která je důležitá pro proces průběžného zlepšování služeb.

2.5 Provoz služeb

Po návrhu a zavedení služby následuje její provoz. Během něj je třeba službu monitorovat, kontrolovat stanovené ukazatele, hlídat jestli splňuje stanovené SLA atd. Pro zákazníka je tato část nejviditelnější. V komunikaci s uživateli se zde totiž řeší vzniklé chyby nebo nové požadavky na službu. Ty jsou hlášeny přes jediné kontaktní místo (Single Point of Contact, SPoC) – tzv. service desk. Mimo to může service desk sloužit k poskytování informací či k dotazování zákaníků nebo uživatelů na spokojenost se službou. [4]

Na spokojenost se službou má service desk značný vliv. Jedná se totiž o jediné rozhraní mezi uživateli a IT, proto může utvářet dojem o celé IT organizaci. Service desk s dobrou dostupností či krátkou průměrnou dobou řešení problémů může kompenzovat nedostatky služby. Naproti tomu nedostupný či špatně fungující service desk může kazit dojem a spokojenost zákazníka s jinak perfektní službou. [20]

Kontakt se service deskem může být různý, a to pomocí telefonu, emailu nebo třeba webového chatu. Service desk může být jediný pro všechny ogranizační jednotky (centrální), pro každé oddělení zvlášť (lokální) nebo rozdělený na různých místech, ale dostupný přes jeden bod (virtuální). Dále může být dělen dle specializace pracovníků na service desk nižší úrovně (základní, obecné znalosti) až po specializované service desky. Řešení požadavků uživatelů pak může eskalovat napříč těmito úrovněmi, od první úrovně podpory až po ty specializované. [4]

Správa událostí

Je proces, jehož hlavní úlohou je identifikovat, evidovat, analyzovat a reagovat na události. Událost je změna stavu, která má významný vliv na řízení IT služby nebo konfiurační položky. Pojem je také používan ve významu výstrahy nebo upozornění od IT služby, konfigurační položky nebo monitorovacího nástroje. Události obvykle vyžadují, aby pracovník provozu IT provedl nějakou činnost, a často vedou k registraci incidentu. Události mohou být různého typu, například:

- informační události - uživatel se přihlásil do systému, email dorazil příjemci
- varovné události - malá kapacita uložiště, velká odezva serveru
- výjimky - detekován nelegální software, pokus o přihlášení špatným heslem

Efektivní provoz služeb je založen na znalosti stavu infrastruktury služby a detekci odchylek od normálního či očekávaného chování, proto je proces správy událostí tak důležitý. Incidenty jsou často spjaty s několika událostmi, proto evidence a analýza událostí může přinést také včasnou detekci incidentů. [23]

Správa incidentů

Je proces zodpovědný za správu životního cyklu všech incidentů. Incident je neplánované přerušení IT služby, snížení její kvality nebo selhání konfigurační položky, které zatím IT službu neovlivnilo. Incidenty mohou být rozpoznány technickými pracovníky, detekovány a nahlášeny monitorovacími nástroji, uživateli nebo dodavateli a partnery třetích stran. Je vhodné, aby bylo co nejvíce incidentů detekováno monitorovacími nástroji v procesu správy událostí a uživatelé se dostávali do styku s incidenty co nejméně. Proces by měl incident včas detekovat, evidovat a obnovit běžný provoz služby co nejrychleji, a tím minimalizovat nepříznivý dopad na zákazníka. [23]

Důležitý je kontakt s uživatelem, aby měl možnost sledovat průběh řešení incidentu. Součástí zpracování incidentu je jeho identifikace, zařazení do kategorie a přidělení priority, která rozhoduje o způsobu přiřazení zdrojů. Následuje úvodní diagnóza, řešení v první úrovni a případná eskalace. Po vyřešení a zotavení se z incidentu následuje jeho uzavření ohlašovatelem či automaticky. Proces se snaží najít co nejrychleji řešení incidentu a minimalizovat dobu výpadku služby, podrobnější analýza kvůli příčinám incidentu nebo hrozbě jeho opakování, již spadá pod správu problémů. [20]

Správa problémů

Problém je definován jako příčina jednoho nebo více incidentů. Účelem tohoto procesu je správa všech problémů a jejich životního cyklu od identifikace, prověření, dokumentaci až po případné odstranění. Snaží se minimalizovat negativní dopad incidentů a problémů, které jsou způsobeny chybami IT infrastruktury a proaktivně jim předcházet. Aby toho bylo možné docílit, hledají se příčiny incidentů, dokumentují se známé chyby a iniciují se akce ke zlepšení nebo napravení těchto situací. Známá chyba je v tomto případě definována jako problém, který má popsanou příčinu a řešení. Proces by měl také ověřovat, jestli byly příčiny incidentu odstraněny a nedochází již k dalším. Objevuje se zde silné propojení na proces správy znalostí, zejména skrze databázi známých chyb (Known Error Database, KEDB), která je využívána v obou procesech. Ta by měla umožnit ukládání znalostí o incidentech a problémech, o tom jak byly překonány, aby je bylo možné rychleji rozeznat a vyřešit pokud se objeví znovu. [23]

Plnění požadavků

Významnou rolí service desku je možnost ohlašování požadavků na službu. Požadavek může být například žádost o změnu hesla, žádost o nainstalování nového software nebo třeba žádost o informace ohledně služby. ITIL proto zavádí tento proces, který má za úkol spravovat životní cyklus všech požadavků na službu. Díky němu je uživatelům a zákazníkům k dispozici komunikační kanál pro všeobecnou podporu, pomocí kterého mohou klást dotazy, požadavky, návrhy na zlepšení či stížnosti. Proces se stará o jejich profesionální zpracování, a tím se snaží udržovat vysokou spokojenost s poskytovanou službou. [23]

Správa přístupů

Je proces udělující autorizovaným uživatelům práva k využívání služeb, zatímco neautorizovaným uživatelům je přístup zamítnut. Jedná se tedy o realizaci zásad a postupů definovaných ve správě bezpečnosti informací. Úkolem je efektivně odpovídat na žádosti o poskytnutí přístupu, změnám oprávnění nebo omezení přístupu a udržovat důvěrnost, integritu, nepopiratelnost a dostupnost aktiv. [23]

2.6 Neustálé zlepšování služeb

Tento proces má na starosti přizpůsobování IT služeb měnícím se požadavkům a odhalování možností jejich optimalizace. Typickými aktivitami jsou zde periodické průzkumy spokojenosti zákazníků, analýza provozních dat a možností ke zlepšení, provádění auditů, ladění metrik atd. Snahou je dosáhnout vyšší kvality služeb, snížit náklady a cenu služeb, a tím dosáhnout větší konkurenceschopnosti. [20]

Kapitola 3

Workflow

V této kapitole bude vysvětlen pojem workflow. Dále budou stručně popsány vybrané nástroje pro jeho vytváření a oblasti využití těchto nástrojů.

3.1 Pojem workflow

Pojem workflow se používá v mnoha oblastech a významech. Z toho pramení také množství odlišných definic, se kterými se lze v literatuře setkat. Často se jedná o abstraktní definice, jako například ve zdroji [8]:

Definice 1 *Workflow je posloupnost opakovatelných činností, které je třeba provádět k dokončení požadovaného úkolu.*

Typické je použití tohoto pojmu v kontextu modelování podnikových procesů. V této souvislosti zdroj [13] popisuje workflow jako:

Definice 2 *Tok informací v podnikovém procesu a jejich automatizované řízení.*

Další definici uvádí organizace Workflow Management Coalition[1] [13]:

Definice 3 *Worflow znamená automatizaci celého nebo části podnikového procesu, během něhož jsou dokumenty, informace nebo úkoly předávány podle procedurálních pravidel od jednoho účastníka k dalšímu tak, aby proces dále pokračoval.*

Přestože se uvedené definice značně liší, lze z nich vyčíst, že pojem workflow je spjat s prováděním nějaké komplexní činnosti (procesu, úlohy apod.). Někdy je dokonce definice workflow a procesu prakticky totožná, viz [8] a [14]. Hlavní rozdíl mezi workflow a jinými způsoby zápisu procesu, postupu či algoritmu, bývá spatřován ve vizuální reprezentaci. Pro workflow jsou typická schémata založená na grafu.

3.2 Techniky modelování workflow

Mezi oblasti využití pojmu či principu workflow patří řešení počítačových problémů, analýza a návrh software, popis podnikových procesů, různé simulace a jiné. V různých oblastech

[1] Celosvětová organizace Workflow Management Coalition (WfMC) vznikla v roce 1993. Sdružuje vývojáře, dodavatele, konzultanty a členy akademických kruhů. Jejím cílem je dosáhnout standardizace pojmů a medotik spojených s oblastí workflow technik při modelování podnikových procesů. [25]

jsou různé také požadavky na formální základ systému pro modelování a s tím související možnosti softwarové interpretace modelu. V některých případech je při vytváření workflow prioritou, aby jej bylo možné snadno a rychle vytvořit a následně pochopit bez dalších speciálních znalostí či nástrojů. Jindy může být žádoucí, aby měl vytvořený model vysokou vyjadřovací sílu, bylo možné jej interpretovat v nějakém systému či simulačním nástroji a požadavky na jednoduchost a intuitivnost práce s modelem jsou upozaděny. Některé nástroje se zase snaží kombinovat oba tyto přístupy. Z důvodů odlišných oblastí aplikace workflow se tak s postupem času začaly používat také různé modelovací techniky.

Unified Modeling Language (UML)

Jazyk UML je široce rozšířený v oblasti softwarového inženýrství. Za dobu jeho existence se z něj stal standard, který poskytuje řadu modelovacích nástrojů. Mezi ně patří například diagram aktivit. Ten se používá pro popis dynamických aspektů systému. Kromě oblasti vývoje software se používá také k modelování obchodních (business) procesů a workflow. Podobným diagramem je stavový diagram. Oba ukazují posloupnost stavů, které nastávají v čase a ukazují podmínky způsobující přechody mezi těmito stavy. K dispozici jsou konstrukce ke složitějšímu řízení toku, jako třeba rozhodování na základě podmínky nebo větvení umožňující modelovat paralelní průběh. [3]

Petriho sítě

Zatímco UML diagramy stavů a aktivit slouží především pro vizualizaci a vzájemné dorozumění mezi vývojáři či procesními specialisty, zápis nějakého procesu prostřednictvím Petriho sítí je používán spíše pro možnost jeho následné interpretace. Petriho sítě umožňují vizuální znázornění a tvorbu modelu a přitom mají formální matematický základ. Díky tomu nachází uplatnění v různých oblastech modelování a simulace diskrétních systémů (například systémů hromadné obsluhy). Existuje množství jejich variant, například C/E (Condition / Event) Petriho sítě, P/T (Place / Transition) Petriho sítě, CPN Barevné (Coloured) Petriho sítě a jiné. [6]

Business Process Modeling Notation

Business Process Modeling Notation (BPMN) je grafická notace, která slouží k modelování podnikových procesů. Jedná se o formalizovaný nástroj založený na teorii Petriho sítí vyšší úrovně [27]. Jde o soubor grafických objektů a pravidel, podle nichž mohou být mezi sebou spojovány. Vznik BPMN byl iniciován snahou vytvořit notaci, která bude čitelná všemi účastníky životního cyklu procesu (business analytici, techničtí vývojáři, analytici monitorující procesy atd.). Díky množství nástrojů, které tuto notaci využívají, se stala prakticky standardem pro modelování podnikových procesů.

Výhodou této notace je, že je na jednu stranu snadná na pochopení a používání, ale na druhou stranu nabízí možnost modelování i komplexních procesů. Důležitou vlastností je rovněž převod mezi návrhem procesu v BPMN a jeho implementací v BPEL (Business Process Execution Language), BPML (Business Process Modeling Language) nebo jiném jazyce pro interpretaci procesů. BPMN definuje převod jednotlivých elementů a sekvencí těchto elementů do jazyka BPEL. Díky tomu je možné model procesu manuálně převést do jeho spustitelné podoby. Kvůli volnosti modelování v BPMN není možné tento převod do BPEL provádět zcela automaticky. Existují však nástroje, které tuto funkci nabízejí, ale za cenu určitých omezení při samotném modelování procesu. [26]

Event-driven Process Chain

Event-driven Process Chain (EPC) patří k velmi rozšířeným technikám pro popis procesu. Především proto, že se stala součástí systémů jako SAP, ARIS či Microsoft Visio. Motivací ke vzniku bylo vytvoření jednoduchého nástroje pro popis podnikového procesu tak, aby mohl být používán širokou veřejností zabývající se touto problematikou.

Základním principem je řetězení událostí a aktivit do schématu. Událost definuje vstupní podmínku provedení aktivity. Ukončení aktivity pak definuje další událost (výstupní podmínku), na kterou mohou navazovat další aktivity. Každá aktivita je tak ohraničena dvěmi událostmi. Tím je definován její začátek a konec. Ke spojování aktivit a událostí lze použít také logické spojky AND, OR a XOR, které mohou mít význam rozdělení nebo naopak sloučení toku. [27].

Kapitola 4

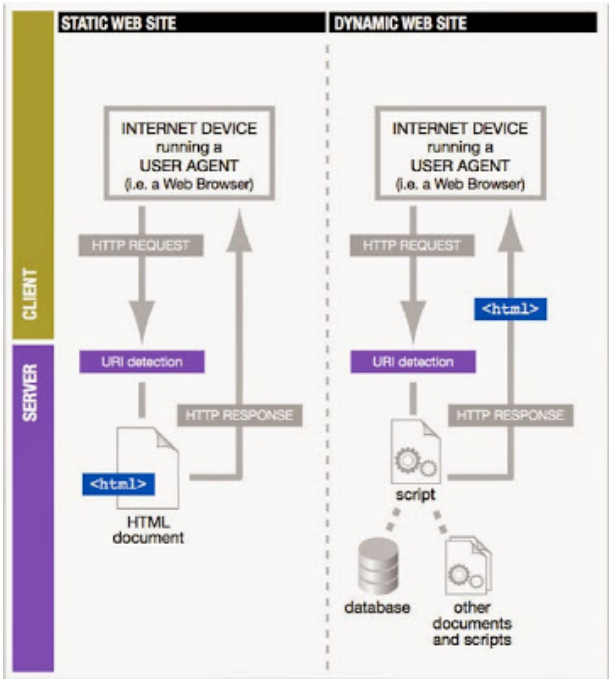
Webové aplikace

V této kapitole budou popsány principy webových aplikaci se zaměřením na technologie, které jsem v projektu použil. Je zde vysvětlen princip architektury rozdělené na část klienta a serveru. Dále je popsána komunikace mezi těmito stranami, zejména pak architektura REST. S ohledem na způsob implementace výsledného systému je hlavní část kapitoly věnována platformě Node.js a databázi MongoDB.

4.1 Model klient-server

Webové aplikace jsou typicky založené na architektuře klient-server. Serverová část zde poskytuje hlavní logiku aplikace. Přijímá, vyřizuje a odpovídá na požadavky klientů. Zároveň také řídí přístup k datům, která bývají centralizována na straně serveru. Klientskou část představuje webový prohlížeč. Ten poskytuje grafické uživatelské rozhraní, které vykresluje dokumenty obdržené ze serveru a zprostředkovává komunikaci se serverem.

Dokumenty, které server zasílá klientovi mohou být pevně definovány tvůrcem a uloženy na serveru. Takovové stránky jsou označovány jako statické. Druhou variantou jsou dynamické webové stránky. V tom případě nejsou HTML dokumenty pevně definovány tvůrcem, ale jsou generovány serverem na základě požadavku klienta. To umožňuje na základě stejných požadavků generovat odlišné dokumenty, které mohou být například naplněny různými údaji z databáze apod. Rozdíl mezi statickým a dynamickým webem ilustruje obrázek 4.1.



Obrázek 4.1: Rozdíl mezi statickým a dynamickým webem. [12]

Dynamické generování výsledného dokumentu probíhá pomocí šablon. Šablonou je HTML dokument (nebo dokument vytvořený v jiném značkovacím jazyce), který může být doplněn o meta značky. Skrze tyto značky je vkládán dynamický obsah, nebo jsou použity ve významu řídících příkazů (podmínek, cyklů), kterými se generování obsahu řídí. [12]

Pro grafické zobrazení výsledného dokumentu uživateli prohlížečem se využívají další významné technologie, jako například kaskádové styly (Cascading Style Sheets, CSS). Jedná se o jazyk, který popisuje způsob zobrazení elementů obsažených v dokumentech typu HTML, XHTML nebo XML. Pro dynamické chování vykreslených webových stránek se pak využívá JavaScript. Ten je možné použít pro asynchronní komunikaci se serverem, interaktivní chování vykreslených dokumentů, tvorbu animací a dalších efektů.

Knihovna jQuery

Pro usnadnění práce s HTML dokumentem na straně klienta existuje javascriptová knihovna jQuery. Jejím velkým přínosem je, že její funkce pracují napříč různými prohlížeči stejně. Díky tomu vývojář nemusí ošetřovat specifika různých prohlížečů a používat různé funkce tak, aby dosáhl stejného výsledku. Mimo to je k dispozici také množství pluginů pro tvorbu uživatelského rozhraní. Mezi ně patří například nástroj pro tvorbu modálních oken, obrázkových galerií, kalendářů a jiné.

4.2 Komunikace mezi klientem a serverem

Pro komunikaci mezi dvěma stranami existuje množství přístupů. Například technologie Java RMI (Java Remote Method Invocation) nebo DCOM (Distributed Component Object Model) jsou určeny pro vytváření provázaných distribuovaných systémů. Implementace obou stran jsou v těchto systémech na sobě závislé. Pro vývoj tak nelze počítat s volbou

libovolných technologií. Z těchto důvodů se při vývoji webových aplikací využívá univerzálních způsobů komunikace. Ty splňují webové služby, které možňují komunikaci mezi systémy bez ohledu na technologie, v jakých jsou implementovány. Za zástupce těchto principů komunikace lze označit protokol SOAP a architekturu REST. [17]

Architektura REST

Architektura REST může být aplikována v různých protokolech, nicméně nejznámější je její použití v kombinaci s HTTP protokolem pro vytváření webových API. REST slouží pro jednotný a snadný přístup ke zdrojům. Zdrojem jsou obvykle data uložená na serveru. Ta mohou být uložena v nějakém interním formátu, obvykle v databázi. Proto může být rozdíl mezi zdrojem samotným a jeho reprezentací, která je odesílána jako odpověď na požadavek klientovi. Server neodesílá přímo výpis konkrétního zdroje, ale namísto toho odesílá data, která jsou zabalená například v JSON nebo HTML. Základním principem je, že každý zdroj má svůj vlastní identifikátor URI (Uniform Resouce Identifier) a jsou definovány čtyři základní metody pro přístup k nim. Ty jsou známé pod označením CRUD, tedy vytvoření nových dat (Create), získání požadovaných dat (Read), úpravu (Update) a smazání (Delete). Ty jsou v HTTP protokolu implementovány pomocí odpovídajících metod [16]:

- GET - metoda pro získání zdroje. Data klientova požadaku jsou umístěna v URI konkrétního zdroje.
- POST - metoda pro vytvoření nových dat. Ve chvíli volání není známý přesný identifikátor zdroje, protože zdroj ještě neexistuje. Proto se pro POST používá domluvený společný identifikátor (např. /users/save). Data ze strany klienta bývají odesílána v těle požadavku.
- DELETE - metoda pro mazání zdroje. Zdroj ke smazání je určen URI.
- PUT - úprava zdroje. Data ze strany klienta bývají odesílána v těle požadavku.

Obousměrná komunikace ve webových aplikacích

Dalším hlediskem, jakým lze na komunikaci mezi dvěma stranami nahlížet, je její směr. Webové aplikace pracují běžně na principu dotaz-odpověď. To znamená, že jakmile má klient nějaký požadavek, zašle jej na server. Ten jej zpracuje a zašle odpověď. Hlavním rysem tohoto způsobu je, že komunikaci vždy iniciuje klient.

To však nemusí být dostatečné pro všechny typy aplikací. Někdy je žádoucí, aby komunikaci zahájil server. Může se jednat o aplikace pracující v reálném čase, například chatová komunikace více uživatelů. V ní chce být každý z uživatelů informován o nové zprávě ihned, jakmile je někým odeslána. Jednoduché a intuitivní řešení jak toho docílit, je periodické dotazování se na server. Toto řešení není příliš efektivní, protože dotazy jsou často zbytečné (ke změně informací nedošlo) a jejich pravidelné opakování tak zbytečně zatěžuje server. Tento přístup je označován jako *polling*.

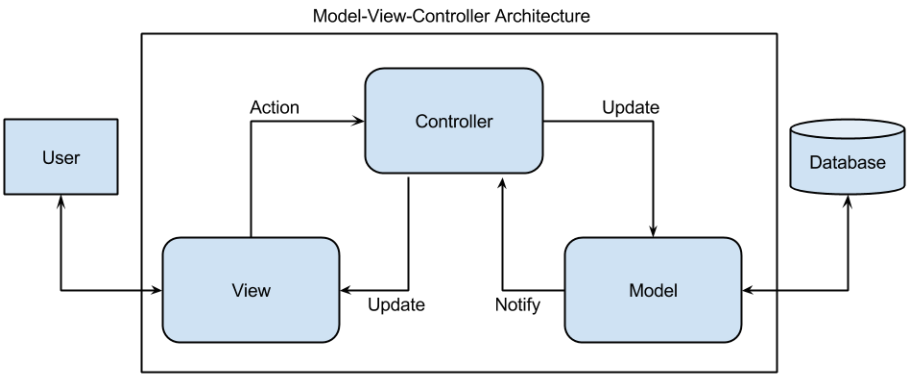
Modifikace tohoto přístupu bývá označována jako *long-polling* a funguje tak, že klient opět zahajuje spojení se serverem, které je v tomto případě udržováno do doby, než má server dostupná požadovaná data. Po odeslání těchto dat je spojení ukončeno a klient musí spojení vytvořit znovu. Pro potřebu komunikace v reálném čase vzniklo množství podobných technik, které měly na straně serveru jedno společné – množství vláken neukončených spojení.

Další možností je použití protokolu WebSocket. Ten nabízí plně duplexní (obousměrný) komunikační kanál pomocí protokolu TCP. Navázání komunikace zahajuje klient HTTP žádostí, která obsahuje příznak, že klient požaduje použití protokolu WebSocket. Pokud server vyhoví, pokračuje komunikace namísto HTTP protokolem WS nebo jeho zabezpečenou verzí WSS.

Ve srovnání s přístupem *long-polling* má protokol WebSocket řadu výhod. Metody založené na dlouhotrvajících požadavcích HTTP totiž mohou mít problémy kvůli omezení počtu současných dotazů na konkrétní doménu a především kvůli různým příčinám nepředpokládaného ukončení tohoto spojení. Dlouhotrvající spojení totiž může ukončit třeba sám klientův prohlížeč, některé firewally nebo proxy servery. WebSocket je navíc úspornější co se týče výpočetních prostředků, které by byly jinak alokovány pro udržování živých spojení. Nevýhodou WebSocket může být, že se jedná o novější přístup, který navíc vyžaduje podporu jak na straně klienta, tak i serveru. Nicméně v současné době je již ve většině hlavních prohlížečů podporován a také na straně serveru je pro práci s ním k dispozici řada frameworků. [22, 15]

4.3 Frameworky pro tvorbu webových aplikací

Pro vývoj serverových částí aplikací se používá množství programovacích jazyků, resp. frameworků. Od základu, bez použití nějakého frameworku, se komplexní webové aplikace vytváří jen zřídka. Použití nějakého frameworku totiž značně usnadňuje práci při vývoji. Významnou výhodou bývá podpora architektury MVC (Model-View-Controller). Ta rozděluje systém na tři logické části. Model představuje data aplikace, View uživatelské rozhraní a Controller reprezentuje logiku aplikace. Toto rozdělení přináší snížení závislostí jednotlivých komponent tak, aby úprava v jedné části znamenala co nejmenší zásah do dalších částí. Kód je tak snáze udržovatelný, na různých částech mohou lépe spolupracovat různí vývojáři, v různých fázích projektu, zlepšuje se možnost testování a jiné praktické aspekty vývoje software.



Obrázek 4.2: Schéma ukazuje architekturu MVC se znázorněnými vazbami na uživatele a databázi. [11]

Alternativou ke vzoru MVC je MVP (Model-View-Presenter). Ten se mírně liší způsobem komunikace mezi jednotlivými částmi. V případě MVC je aktualizace modelu prováděna přímo z View, kdežto u MVP View komunikuje s modelem nepřímo přes Presenter, který na základě akcí uživatele aktualizuje View i Model.

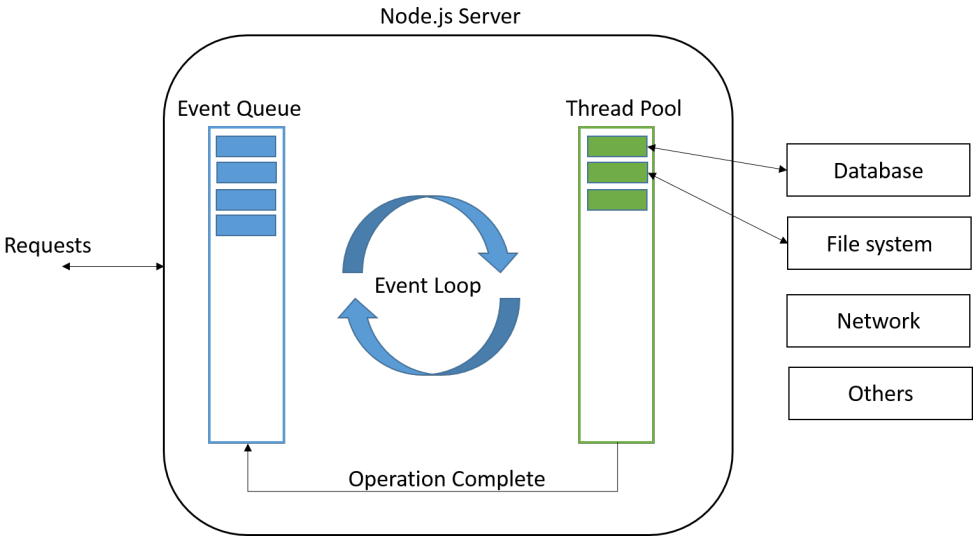
Další důležitou výpomoc poskytují frameworky při práci s databází, kdy nad ní vytváří jakousi mezivrstvu. Typické je použití různých ORM (Object-Relational Mapping) nástrojů, které provádí mapování objektů daného programovacího jazyka do relační databáze. Mezi další benefity použití frameworků patří vyšší míra zabezpečení aplikace díky ošetření běžných programátorských chyb a podpora nástrojů pro testování a ladění.

Běžně se pro vývoj webových aplikací používají jazyky PHP (např. frameworky Laravel, Symfony), C# (ASP.NET), Java (Spring), Node.js (Express) a jiné.

Node.js

Node.js je javascriptové běhové prostředí, jehož základem je javascriptový interpret V8 od Google Chrome. Přestože byl JavaScript dříve používán zejména na straně klienta, nyní se je jeho použití na serveru celkem rozšířené.

Node.js je vystavěn na událostmi řízené architektuře. V takové architektuře je tok programu dán vzniklými událostmi, jako třeba nějakou akcí uživatele (klik myši, stisk klávesy) nebo přijetím zprávy od jiného programu či vlákna. Tento přístup je známý z aplikací obsluhující GUI. Základem je smyčka obsluhující události. Ta v případě Node.js iteruje frontou událostí a zpětných volání dokončených operací. Jakmile se v průběhu programu narazí na nějakou I/O operaci (například načítání dat z disku), tak se její vykonání deleguje na další vlákno. Hlavní smyčka obsluhující události mezitím pokračuje dál. Jakmile je I/O operace dokončena, zařadí se její zpětné volání do fronty. Smyčka událostí následně zpětné volání zpracuje a poskytne výsledek, viz obrázek 4.3. Díky tomu jsou I/O operace vykonávány asynchronně a spuštěná aplikace může zpracovávat požadavky klientů v jednom vlákně.



Obrázek 4.3: Smyčka pro zpracování událostí. [19]

Tímto přístupem se Node.js liší od běžných webových serverů (například Apache), které vytvářejí nové vlákno zvlášť pro zpracování každého klientského požadavku. Nevýhodou tohoto přístupu je, že v případě čekání na výsledek nějaké I/O operace je vlákno pozastaveno. Pozastavené vlákno však stále zabírá zdroje. Při větším počtu současných požadavků se tak jejich obsluha stává úzkým hrdlem aplikace, protože vzniká velké množství vláken, které jsou náročné na výpočetní zdroje, přestože některé z nich pouze čekají na výsledky I/O operací. [19]

Díky událostmi řízené architektuře a neblokujícím I/O operacím umožňuje platforma Node.js vytvářet výkonné a škálovatelné aplikace. Jako její další výhody jsou udávány:

- Jednoduchá a rychlá konfigurace a vytvoření první aplikace.
- Systém Node Package Manager (NPM), který obsahuje tisíce modulů. Ty jsou k dispozici zdarma a systém NPM umožňuje jejich snadnou instalaci, popřípadě kontrolu, jestli jsou při spouštění aplikace všechny k dispozici.
- Syntaxi známou z JavaScriptu, který je používán vývojáři na straně klienta. Odpadá tak nutnost učit se více programovacích jazyků a spolupráce backend a frontend vývojářů může být jednodušší.
- Při použití NoSQL databáze využívající ukládání objektů ve formátu JSON je možné JavaScript využít i pro práci s databází, díky čemuž je vývoj aplikace ještě přímočařejší.
- Sdílení a znovupoužití JavaScriptové kódu napříč serverem a klientem.
- Aplikace jsou multiplatformní a mohou běžet v prostředích Linux, Windows i OS X.

Hlavní uváděnou nevýhodou je, že náročné CPU výpočty snižují propustnost. V těchto případech je vhodnější použití platformy založené na více vláknech.[9]

Node.js nachází uplatnění hlavně při tvorbě jednostránkových či real-time aplikací. Typickými příklady použití jsou různé chatovací weby a aplikace, ve kterých spolupracuje více uživatelů v reálném čase. Například multiplayerové hry nebo online editory dokumentů (jako třeba Trello, Dropbox Paper nebo Google Docs). Další oblastí využití jsou služby poskytující datové streamování.

4.4 Databáze

Podle způsobu ukládání dat lze databáze rozdělit na relační a NoSQL. Relační databáze jsou tvořeny tabulkami, kde sloupce se nazývají atributy, mají předem určený datový typ a množinu hodnot, kterých mohou nabývat. Řádky představují jednotlivé záznamy pro uložení dat. Řádky v tabulce jsou jednoznačně identifikovány primárním klíčem. Cizí klíč pak udává vztah mezi dvěma tabulkami tak, že hodnota v daném sloupci jedné tabulky musí být obsažena v jiném (primárním) klíči. Databázový systém zaručuje ACID[1] změn uložených dat. Pro práci s daty se používá dotazovací jazyk SQL.

Požadavky na decentralizovanost databáze, práci s velkými objemy dat a operací, rychlost, úmyslnou redundanci pro odolnost proti výpadkům, vysoké nároky na škálovatelnost, časté změny schématu a další vedly k vývoji tzv. NoSQL (Not only SQL) databází. Existuje několik druhů NoSQL databází, např. typu klíč-hodnota, dokumentová, grafová a další. Dodržování modelu ACID v těchto databázích nebývá jednoduché, a tak místo něj bývá dodržován model BASE[2]. [21]

[1] ACID atomičnost (Atomicity) - provedeny všechny změny nebo žádná, konzistence (Consistency) - v databázi jsou pouze data dle pravidel, izolovanost (Isolation) - souběžné transakce se neovlivňují, trvalost (Durability) - uskutečněné změny nebudou ztraceny

[2] BASE (Basicaly Available, Soft state, Eventualy consistency) - aplikace je v podstatě stále dostupná v nějakém známém stavu, byť nemusí být nutně konzistentní. Případné nekonzistence jsou řešeny při čtení, zápisu nebo asynchronně.

MongoDB

MongoDB se řadí mezi dokumentové NoSQL databáze. Jedná se o multiplatformní databázi, která je vyvíjena v jazyce C++. Dokumenty se skládají z dvojic klíč-hodnota a zapisují se ve formátu JSON (JavaScript Object Notation). Ty se pro efektivější využití paměti ukládají ve formátu BSON (binární JSON). Hodnota v dokumentu může být některého ze základních datových typů, ale také dokument, pole nebo pole dokumentů. Díky tomu je možné dokumenty vnořovat do sebe. Dokumenty je možné také vzájemě odkazovat pomocí ID. K dispozici je také dědičnost, odvozování a následné rozšiřování dokumentů.

Pro definici struktury ukládaných dokumentů je možné použít balíček Mongoose. Ten umožňuje definovat objekty se silně typovaným schematem, které je mapováno na MongoDB dokumenty. Dále poskytuje širokou škálu možností, jako třeba zadání výchozí hodnoty, validaci dat, automatickou konverzi, definici indexů pro rychlejší načítání a další. [18]

Kapitola 5

Specifikace požadavků

Tato kapitola se věnuje upřesnění cílů práce a specifikaci požadavků na výsledný systém, které proběhly na začátku práce. Jsou zde uvedena již existující řešení zaměřená na nácvik provozu služeb dle ITIL, zejména práce ITIL trenažér, na kterou tato práce částečně navazuje.

5.1 Cíle práce

Cílem práce je navrhnout a implementovat herní engine, který bude umožňovat vytváření simulace provozu IT služeb. Práce částečně navazuje na aplikaci ITIL trenažér, viz 5.2. Snahou je použít koncept výukového trenažéru a rozšířit jej o možnost provádět simulaci provozu služeb v reálném a proměnlivém čase anebo v tazích. Součástí aplikaci bude také jednoduchý service desk, pomocí kterého bude možné reagovat na události, incidenty či požadavky na změnu vzniklé během provozu služby.

Herní engine – tvůrčí část

Tato část tvoří jádro práce, které bude dostupné tvůrci simulace. Engine by měl umožnit vytváření služeb. Vytvoření služby se bude skládat z návrhu infrastruktury služby a z definice jejího chování.

Pro možnost obecného návrhu služby bude možné přiřadit ke službě libovolné vlastní atributy. Každému atributu bude možné zadat jeho název, počáteční hodnotu a jednotku. Příkladem takového atributu může být třeba spokojenost uživatelů se službou, udávaná v procentech nebo třeba množství financí vynaložených na provoz služby. Infrastruktura služby bude reprezentována konfiguračními položkami zapojenými do schématu. Každá konfigurační položka bude mít svůj název, popis a ilustrační obrázek. Podobně jako ke službě, tak i ke konfiguračním položkám bude možné přiřadit libovolné vlastní atributy.

Chování služby bude popsáno pomocí workflow grafů. V uzlech těchto grafů bude možné měnit hodnoty atributů konfiguračních položek a služby, což bude simulovat změnu konfigurace služby nebo jinou událost významnou pro její provoz. Hrany budou představovat přechody mezi těmito uzly. Pomocí nich bude realizováno větvení na základě podmínky nebo časové zpoždění mezi jednotlivými uzly. Scénář provozu služby bude dán tokem tohoto grafu z počátečního uzlu do koncového.

Herní engine – hráčská část

V hráčské části si uživatelé budou moci vyzkoušet provoz služeb navržených ostatními uživateli. Každý uživatel bude moci provozovat souběžně několik služeb. Během toho bude moci monitorovat stav těchto služeb, sledovat důležité indikátory a zejména reagovat na události, incidenty či požadavky na změnu generované enginem na základě definice chování dané služby. To umožní hráči ovlivňovat průběh provozu služby. Včasná a vhodná reakce na vzniklé tikety může zachovat dostupnost služby, minimalizovat počet vzniklých incidentů a udržovat další ukazatele na smluvené úrovni.

Service desk

Interakce hráče s provozovanou službou bude probíhat skrze jednoduchý service desk. V praxi slouží service desk jako rozhraní mezi poskytovatelem a uživateli služby. Typický příklad jeho použití v reálném světě může zjednodušeně vypadat následovně:

1. Uživatel služby reaguje na nějakou vzniklou událost. Například mu přestane fungovat internetové připojení. Proto založí tiket, aby kontaktoval poskytovatele služby a ten situaci vyřešil.

2. Poskytovatel služby přijme nový tiket a snaží se požadavek uživatele vyřešit. Například opravou či výměnou HW na straně uživatele. Jakmile se to podaří, tiket je v service desku označen jako vyřešený.

V tomto scénáři simuluje chování uživatelů herní engine tím, že interpretuje workflow graf popisující chování služby. Hráč zde vystupuje v roli poskytovatele služby.

Řešení reálné situace z pohledu poskytovatele bývá komplexní úkol, který zahrnuje spolupráci s uživatelem služby. Často je zapotřebí požadavek od uživatele nejprve analyzovat a poté zvolit a realizovat vhodné řešení na základě dostupných znalostí, případně delegovat úkol na vyšší úroveň podpory. Následně se kontroluje, jestli zvolené řešení opravdu vyřešilo požadavek uživatele a teprve poté následuje uzavření požadavku uživatele (tiketu) v service desku. I přesto se časem může ukázat, že provedené řešení nebylo vyhovující, tiket je znovu otevřen a proces pokračuje. Hlavní činností v tomto procesu je především samotné provedení nějakého úkonu, kterým se požadavek uživatale řeší (v našem příkladě je to oprava či výměna HW).

Tvorba algoritmu, který by takové chování provozu služby přesně popisoval, by byla pro uživatele aplikace náročná. Pro zjednodušení proto bude moci tvůrce služby zadat ke každému tiketu, který bude služba generovat, výčtem několik možností k jeho řešení. Když potom hráč během provozu služby na takový tiket narazí, bude na něj moci reagovat volbou jednoho z těchto řešení. Zvolení řešení bude simulovat jak skutečné provedení nějakého úkonu, tak i vyřešení tiketu v service desku.

Statistiky

Pro tvůrce služby bude k dispozici také přehled klíčových údajů o provozu služeb, které vytvořil. Díky tomu bude moci sledovat, kteří hráči jeho službu provozovali a jakých hodnot významných indikátorů při tom dosáhli. K dispozici by měly být údaje o dostupnosti služby, udržovatelnosti nebo třeba spolehlivosti služby. Způsob výpočtu těchto údajů se nachází v kapitole 2.3. Tato funkcionalita má podpořit výukový charakter aplikace tak, že tvůrce služby (učitel) bude moci sledovat, kteří hráči (studenti) si provoz jeho služby vyzkoušeli.

5.2 Existující řešení

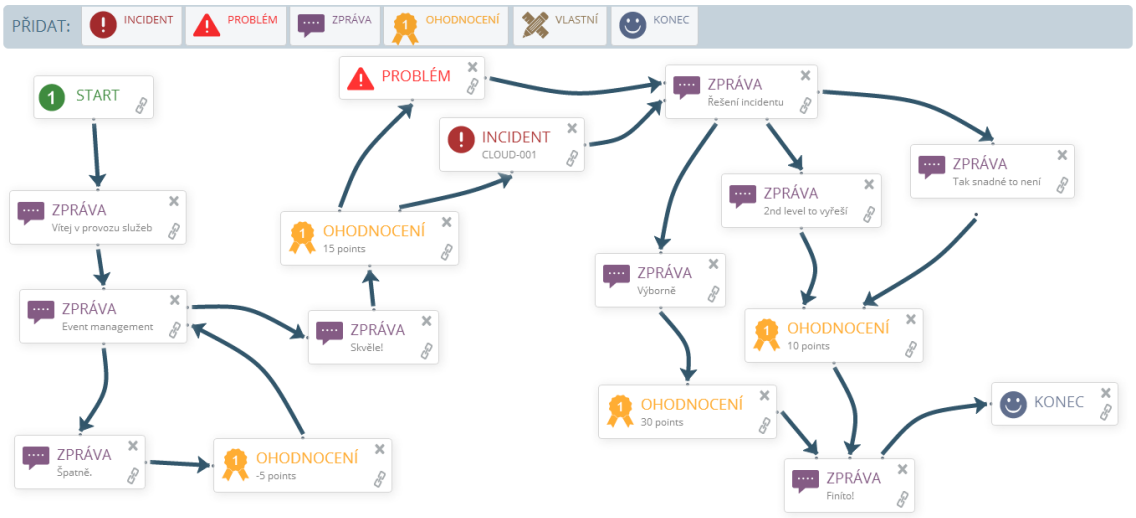
Pro výuku provozu služeb dle ITIL je v dnešní době k dispozici množství řešení. Obvyklým způsobem jsou online školení a kurzy, které bývají spjaty s přípravou na certifikační zkoušku[1]. Více prakticky orientovanou možnost představují semináře, během kterých účastníci v týmech pod vedením mentora řeší reálné problémy za pomoci specializovaného software[2]. Kombinace výukových simulátorů a reálné spolupráce několika členů v týmu poskytuje zkušenosti blížící se reálnému provozu.

Čistě programovým řešením, které se zabývá praktickým nácvikem správy IT služeb a je dostupné i bez placených kurzů, je ITIL trenažér.

ITIL trenažér

Tato aplikace byla vytvořena v rámci diplomové práce na FIT VUT v roce 2013. Je to výukový webový informační systém, který rozlišuje dva režimy – tvůrce a hráče. Tvůrce vytváří výcviky pro hráče, kteří si jejich absolvováním procvičují návrh a provoz služeb dle knihovny ITIL.

V rámci vytváření výcviku tvůrce definuje, jaké služby bude hráč navrhovat a provozovat. Pro fázi návrh služby může uvést základní atributy služby, jako například název či vlastník. Dále může definovat, z jakých konfiguračních položek se služba bude skládat a jaké jsou jejich vstupy a výstupy. Pro fázi provoz služeb má tvůrce možnost definovat chování konfiguračních položek. Ty mohou čekat a reagovat na své vstupy, produkovat nějaké výstupy, provádět určitou dobu nějakou svoji činnost, případně ji pravidelně nebo nepravidelně opakovat. Pro provoz služby je možné vytvářet tzv. scénáře. Na ty lze nahlížet jako na workflow, viz 5.1.

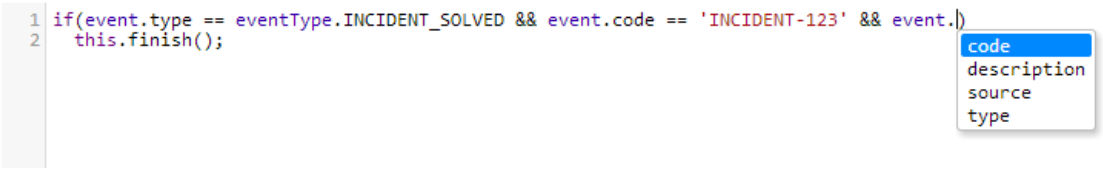


Obrázek 5.1: Workflow editor, který tvoří klíčovou část vytváření výcviku. [7]

[1] Příkladem může být kurz od Autocontu (https://edu.autocont.cz/skoleni/pro-specialisty/itil/ITIL-F3) nebo od společnosti Gopas (http://www.gopas.cz/Kurzy/Katalog-kurzu/Rizeni-IT/ITIL-2011-Foundation-ITILFV3.aspx)

[2] Zajímavá řešení jsou například Fort fantastic (https://www.fortfantastic.com/index.php/en) nebo ITIL Foundation PoleStar Simulation (https://www.globalknowledge.com/us-en/resources/resource-library/videos/itil-foundation-polestar-simulation)

Uzly zde představují provádění akcí, jako třeba vznik incidentu, problému, zobrazení informativní zprávy nebo ohodnocení za hráčův postup. Navíc mohou ovlivnit stav konfiguračních položek, například porušit její SLA. Chování scénáře (tok workflow) lze definovat podobně jako chování konfiguračních položek, a to pomocí editoru, ve kterém je možné využít množinu předdefinovaných příkazů.



Obrázek 5.2: Ukázka editoru chování včetně nápovědy klíčových slov. [7]

Hráč má za úkol tyto výcviky absolvovat. První fází výcviku je návrh služeb, který se provádí zapojením dostupných konfiguračních položek do logického schématu podle toho, jak tvůrce specifikoval jejich vstupy a výstupy. Další fází je provoz služby. Během něj hráč reaguje na vzniklé události, incidenty a problémy. K tomu má k dispozici rozhraní service desku, které umožňuje vyhledávat ve vzniklých událostech, přijímat, prohlížet a řešit incidenty a pracovat s problémy. K dispozici má také databázi známých chyb a jejich řešení. Za absolvování výcviku jsou uživatelé ohodnoceni body tak, jak to definoval tvůrce výcviku. Další metrikou jsou pro hráče finance. Tvůrce určí při vytváření výcviku cenu za pořízení a provoz jednotlivých komponent. Ke každé službě a komponentě navíc může určit vlastní atributy a jejich hraniční hodnoty, při jejichž překročení dojde k porušení SLA, a tím i ke snížení finančního výnosu za provoz služby. Při problémech s konfigurační položkou je možné ji za stanovenou cenu restartovat nebo vyměnit. Snahou hráče je dokončit výcvik s co nejvyšším bodovým i finančním ohodnocením, a dosáhnout tak efektivního poskytování služeb.

Kapitola 6

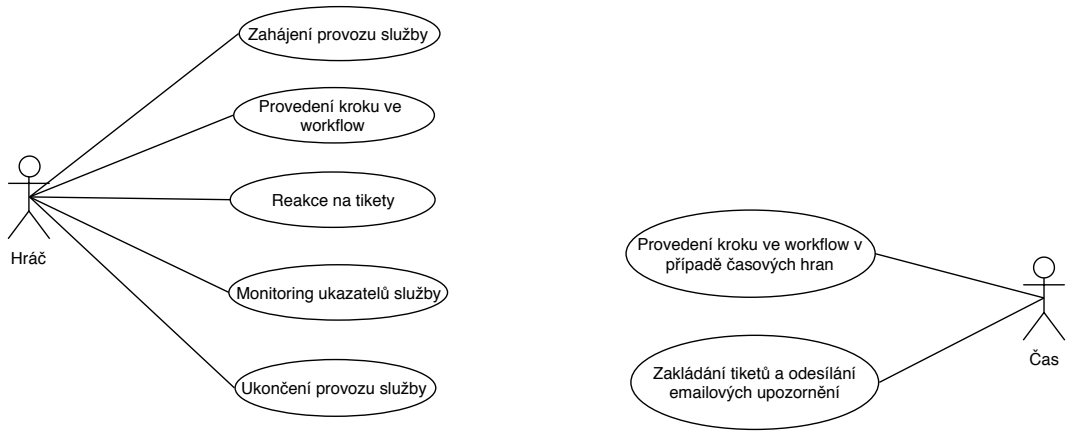
Analýza a návrh

Na základě stanovených cílů práce a specifikace požadavků jsem provedl analýzu a návrh výsledného systému. Analýza zahrnuje soupis základních funkcí, které jsou zobrazeny na diagramech případů užití. Popsán je zde návrh funkcionality vytváření služby a scénáře jejího provozu. Sekce 6.2 potom podrobněji rozebírá návrh vlastního workflow grafu, který představuje jádro enginu. Součástí kapitoly je také doménový model systému se stručným doprovodným komentářem.

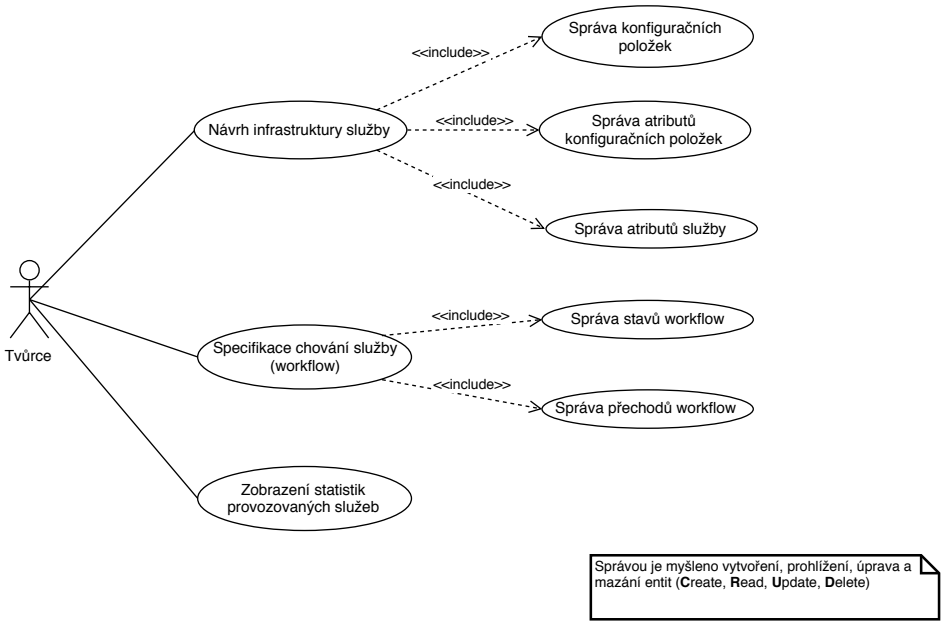
6.1 Případy užití

Při návrhu systému jsem nejdříve začal analýzou aktérů, kteří se budou v systému vyskytovat. Následně jsem sepsal funkce, které by měli mít k dispozici. To ilustrují diagramy na obrázcích 6.1 a 6.2.

Funkce znázorněné na obou diagramech vychází ze specifikace požadavků. Na diagramech se nachází dvě primární role, kterými jsou hráč a tvůrce služby. Uživatel přihlášený do systému bude moci vystupovat v obou rolích, takže možnost vytvářet a provozovat služby bude dostupná pro každého registrovaného uživatele.



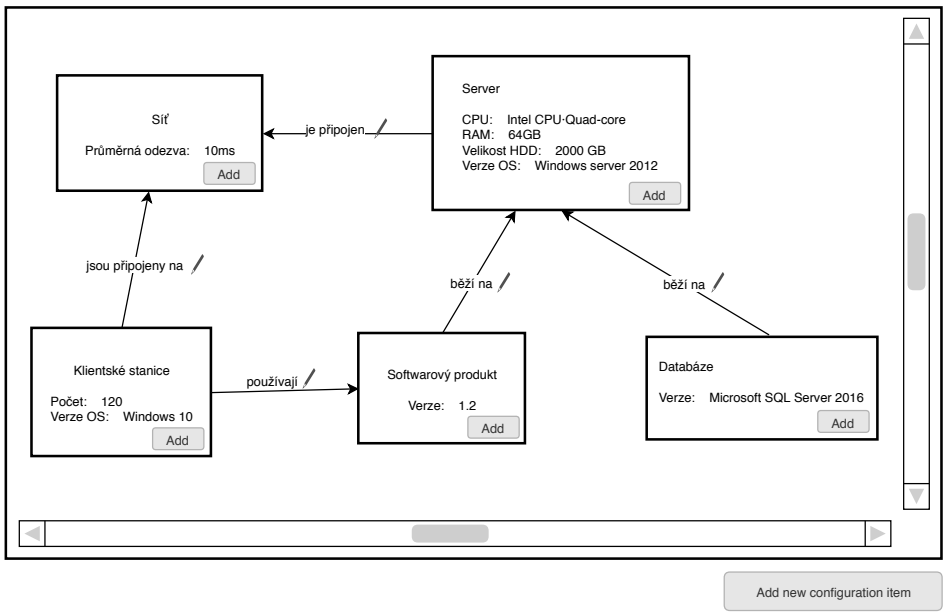
Obrázek 6.1: Diagram případů užití z pohledu provozovatele služby.



Obrázek 6.2: Diagram případů užití z pohledu tvůrce služby.

6.2 Tvorba služby

Základem herního enginu bude vytvoření služby. To zahrnuje vytvoření konfiguračních položek, jejich vlastností a také jejich zapojení do schématu infrastruktury služby. Toto zapojení bude možné vytvářet pomocí intuitivního editoru, jehož návrh ukazuje obrázek 6.3.



Obrázek 6.3: Návrh obrazovky pro vytváření konfiguračních položek, jejich vlastností a vazeb mezi nimi.

Pro takto navrženou službu bude možné definovat scénář. Scénářem je myšlen průběh provozu služby tak, jako kdyby se odehrával v realitě. To lze ilustrovat na příkladě. Službou může být poskytování nějakého softwarového produktu na míru zákazníkům. Její scénář z pohledu poskytovatele mohou představovat požadavky uživatelů na úpravy či implemenaci nových funkcí do aplikace, incidenty ohledně nestability aplikace anebo třeba běžné logované události, které pouze informují například o úspěšném provedení zálohy.

Scénář bude vytvářen pomocí workflow grafu. Tento způsob zápisu chování by měl klást na tvůrce scénáře menší nároky, na rozdíl od popisu chování v nějakém programovacím jazyku. Tvůrce nebude muset znát přesnou syntaxi zápisu všech příkazů a celý scénář uvidí graficky znázorněný ve formě workflow grafu.

Scénář provozu služby

V kapitole 3 byla popsána problematika workflow včetně existujících modelovacích nástrojů. Pro realizaci scénáře provozu služby se nabízela možnost využít některý z těchto nástrojů. Kladnou stránkou tohoto řešení by byla možnost používat k tvorbě scénáře standartizovaný modelovací jazyk se všemi výhodami, které to přináší. Uživatelé se znalostí daného nástroje by měli značně usnadněnou práci. Ostatní uživatelé by měli pro seznámení s nástrojem k dispozici jeho specifikaci, dokumentaci a množství návodů. Odpadlo by také velké množství práce s implementací nástroje pro tvorbu workflow grafu a workflow enginu pro jejich interpretaci.

Hlavní nevýhodu v použití existujícího nástroje jsem shledal v jeho obtížném přizpůsobení pro tvorbu scénáře služby. Kromě tvorby scénáře totiž bylo nutné navrhnout mechanismus pro vytváření doplňujících informací o službě tak, aby mohl být scénář zasazen do kontextu nějaké služby. Mezi tyto doplňující informace patří dle specifikace požadavků zejména konfigurace služby v podobě konfiguračních položek a jejich vlastností. Tyto údaje navíc musí být zpřístupněny pro tvůrce služby v nástroji pro modelování workflow tak, aby s nimi bylo možné manipulovat skrze modelovaný scénář. Navíc jsem se chtěl vyhnout řešení, kdy by práce s konfigurací služby byla definována pomocí zápisu v nějakém zdrojovém kódu. Scénář, jehož součástí byl zápis ve zdrojovém kódu, byl použit v práci ITIL trenažér a při hodnocení uživateli byl hodnocen jako slabší stránka aplikace [7].

Z těchto důvodů jsem se rozhodl navrhnout pro potřeby realizace scénáře provozu služby vlastní workflow. Vytvoření vlastního systému workflow obnáší implementaci nástroje pro jeho návrh a také workflow enginu, který má za úkol proces popsaný workflow grafem interpretovat.

Workflow graf

Navržený workflow se bude skládat z uzlů a orientovaných hran. V jednotlivých uzlech bude možné vyvolat vznik události, incidentu či požadavku na změnu. V uzlech bude také možné manipulovat s hodnotami navržených atributů služby a konfiguračních položek, a to pomocí jednoduchých přiřazovacích příkazů. Ty budou provedeny vždy, jakmile tok grafu dosáhne daného uzlu. Uzly budou různého typu:

Start - počáteční uzel v grafu. Při spuštění provozu služby v něm scénář začíná.

Konec - koncový uzel, neměly by z něj vést žádné hrany. Při dosažení tohoto uzlu scénář končí.

Událost - uzel, který vytvoří tiket typu událost, který se zobrazí hráči v service desku. Událost má svůj předmět a popis. Dále bude obsahovat sadu řešení, která představují možnosti reakce hráče na tuto událost.

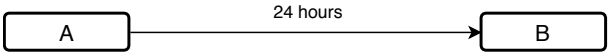
Požadavek na změnu - podobně jako událost, jen vzniklý tiket bude typu požadavek na změnu.

Incident - tiket bude mít stejné vlastnosti jako událost a požadavek na změnu, zde však navíc může tvůrce definovat, zda-li na jeho základě dochází k výpadku služby.

Stav - obecný stav, na základě kterého nevzniká žádný tiket. Bude v něm možné pracovat s hodnotami atributů, případně čekat na další krok.

Také hrany budou různého typu, přičemž pro všechny typy hran platí, že z jednoho uzlu může vycházet pouze jeden typ hran. Pro vysvětlení uvažujme přechod z uzlu A do uzlu B, případně C. Ten probíhá tak, že nejprve jsou provedeny všechny akce v uzlu A. Tyto akce představují např. odeslání incidentu nebo modifikaci hodnot atributů na základě příkazů. Poté následuje přechod do dalšího uzlu na základě typu hrany:

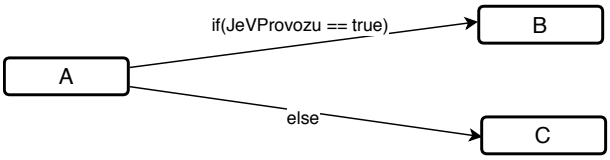
Časovaná - v uzlu A se čeká na uplynutí daného časového intervalu a potom se přejde do uzlu B. Cílový uzel musí být právě jeden.



Obrázek 6.4: Časovaná hrana.

Tento typ hrany je zde zejména z důvodu možnosti provádět simulaci v reálném či zrychleném čase.

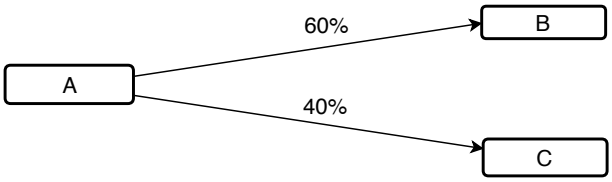
Podmínková - z uzlu A se přejde do ulzu B nebo C na základě podmínky. V podmínce může být výraz, který například porovnává hodnoty některých z atributů služby. Při použití tohoto typu přechodu musí ze stavu vycházet právě dvě hrany – `if` a `else`.



Obrázek 6.5: Podmínková hrana.

Možnou alternativou pro větvení na základě podmínky bylo vytvoření konstrukce, která by umožňovala více výstupních hran, přičemž výběr jedné z nich by se prováděl na základě jedné podmínky (konstrukce podobná bloku `switch - case`). Ve zdrojovém kódu je v tomto případě priorita při výběru jedné z hran dána pořadím v zápisu. To v grafickém znázornění není, a tak by tvůrce musel navíc definovat prioritu těchto hran. Pro jednoduchost návrhu jsem zůstal pouze u jednoduché konstrukce `if - else`, které lze v případě potřeby řadit za sebe.

Pravděpodobnostní - z uzlu A se přejde buď do uzlu B nebo C na základě stanovené pravděpodobnosti. Hran z uzlu A může vycházet více než dvě. Tento typ hran slouží k modelování jisté míry neurčitosti. Díky tomu se provoz služby nemusí chovat vždy stejně, takže pro hráče nebude snadno předvídatelný.



Obrázek 6.6: Pravděpodobnostní hrana.

Tento koncept workflow byl navržen tak, aby poskytnul tvůrci hry základní programátorské konstrukce, jakými jsou větvení na základě podmínky nebo cyklus a zároveň mu umožnil vytvářet scénáře, které se mohou chovat rozdílně při každém spuštění.

6.3 Provoz služby

Službu s vytvořeným scénářem bude možné zveřejnit. Tím se stane dostupná pro hráče, kteří si ji budou moci spustit. Tvůrce nadále zůstane jejím výhradním vlastníkem a jako jediný bude moci tuto službu upravovat a prohlížet si detaily o jejím provoz. Pro hráče zůstane implementace cizích her skryta. To je důležité proto, aby si nemohli prohlížet scénáře, předvídat jejich chování, a tím si dopomáhat k lepším výsledkům.

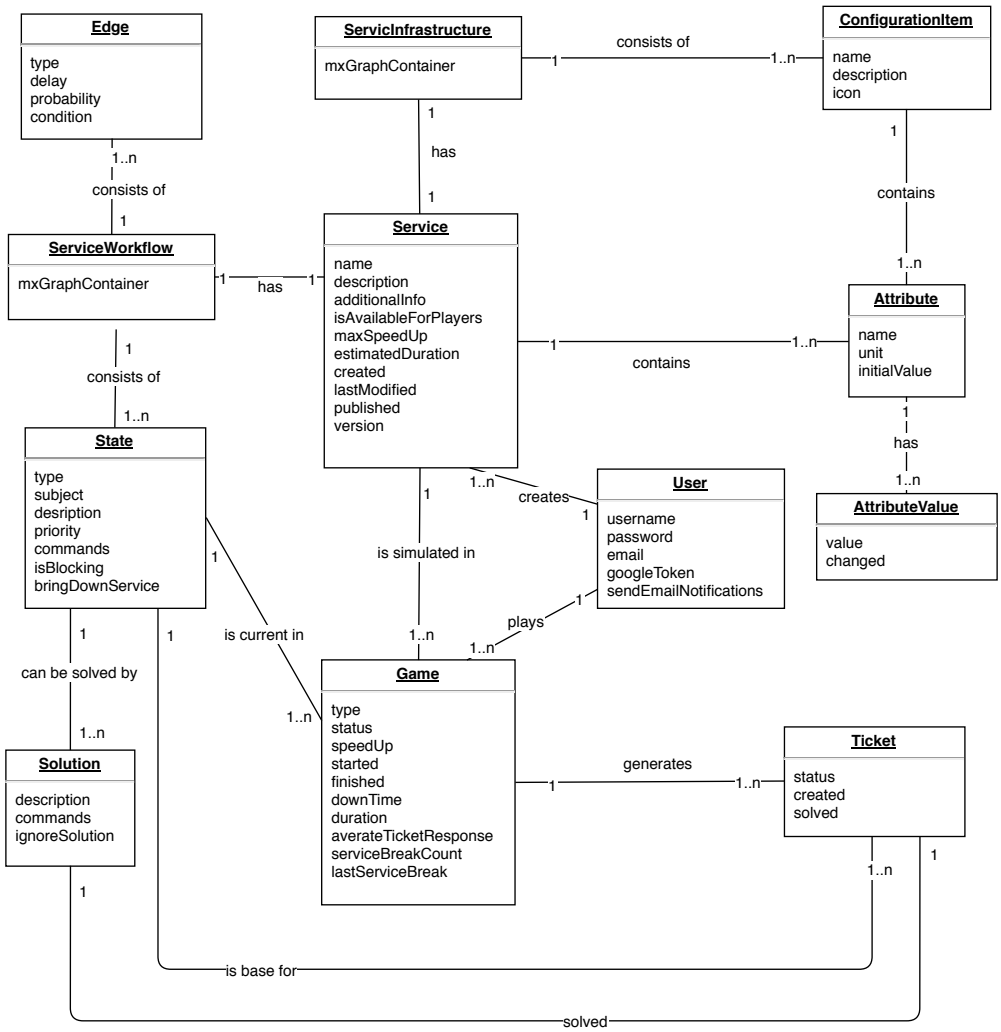
Při spuštění služby bude mít hráč na výběr, jestli bude scénář probíhat v tazích anebo v reálném či zrychleném čase. Při volbě tahů bude jeden tah odpovídat jednomu kroku ve workflow, tedy provedení hrany a přesunu z výchozího uzlu do následujícího. Hru na bázi tahů si lze představit tak, že hráč bude během hry provádět úkony v service desku a jakmile bude vše považovat za hotové, provede tah, a tím se hra posune kupředu. Na základě typu následujícího uzlu mohou hráči v service desku objevit nové tikety k řešení. Dále se na základě příkazů definovaných tvůrcem aktualizují hodnoty atributů konfiguračních položek a služeb. Mohou se také změnit hodnoty klíčových ukazatelů, jako třeba dostupnost nebo průměrný čas mezi výpadky služby. Tento princip tahů je inspirován počítačovými hrami, které jsou označovány jako tahové strategie[1]. Využívá se jej tehdy, když jsou úkony ve hře příliš složité nebo je jich velké množství na to, aby se daly zvládat v reálném čase.

Druhou možností průběhu hry bude režim v reálném či zrychleném čase. Strategické hry založené na tomto principu se nazývají realtimové[2]. Pro tento případ bude možné nastavit, kolikrát bude hra zrychlena oproti reálnému času. To usnadní práci tvůrci scénáře během vytváření workflow, protože bude stačit, aby při použití časovaných přechodů pracoval pouze s reálným časem. Navíc bude možné simulovat dlouhodobý provoz služeb, aniž by hra musela ve skutečnosti tak dlouho běžet.

[1] Například hry Civilization nebo Heroes of Might and Magic.
[2] Například Age of Empires nebo StarCraft

6.4 Doménový model systému

Entity a jejich vlastnosti, ze kterých se bude celý systém skládat, zachycuje doménový model. Významnou entitou je *Služba*, která může být vytvořena *Uživatelem*. Tato *Služba* má své *Atributy*, *Graf infrastruktury* a *Graf workflow*. *Graf infrastruktury* je složen z *Konfiguračních položek*, které rovněž obsahují své vlastní *Atributy*. Každý *Atribut* může nabývat různých *Hodnot*. *Graf workflow* se skládá ze *Stavů* a *Hran* mezi nimi. Simulace provozu *Služby* je reprezentována entitou *Hra*. *Hra* má svůj aktuální *Stav*. Přechod do následujícího *Stavu* může být podnětem ke vzniku *Tiketu*, který může mít několik *Řešení*.



Obrázek 6.7: Doménový model systému.

Kapitola 7

Implementace

Tato kapitola se zabývá implementací enginu. Jsou zde popsány některé důležité technologie a významné či zajímavé části implementace. Součástí kapitoly jsou také vybrané snímky aplikace, které ilustrují hlavní funkcionalitu systému.

7.1 Volba technologií

Navržený systém jsem se rozhodl realizovat jako webovou aplikaci. Zejména proto, že pro její spuštění bude stačit mít nainstalovaný webový prohlížeč, takže uživatelé, kteří by si chtěli ITIL trenažér vyzkoušet, nebudou muset nic stahovat ani instalovat.

Na straně serveru jsem se rozhodl použít open source platformu Node.js, protože je vhodná pro tvorbu real-time aplikací. Pro ukládání dat jsem použil NoSQL databázi MongoDB, kterou jsem zvolil především proto, že s ní lze pracovat pomocí JavaScriptu, který je využíván i ve zbytku aplikace. Aplikace je strukturovaná dle vzoru MVP, viz 4.3.

Na straně klienta využívám pro asynchronní komunikaci se serverem asynchronních požadavků AJAX (Asynchronous JavaScript and XML) a tam, kde je žádoucí, aby komunikace byla zahajována ze strany serveru, používám knihovnu Socket.io. Pro stylování dokumentů využívám frameworku Bootstrap, který usnadnuje práci při vytváření respozivního webu a nabízí množství předdefinovaných CSS pravidel pro tabulky, formuláře, tlačítka a další elementy.

Framework Express a významné balíčky

Na serverové části jsem použil framework Express. Důležitým prvkem tohoto frameworku je možnost nastavení směrování. To poskytuje překlad mezi požadovanou URL adresou a odpovídající akcí Controlleru. Express umožňuje dynamicky vykreslovat HTML stránky na základě šablon. Já jsem pro definici šablon systém Jade. V porovnání s běžným HTML se značky neuvádí do ostrých závorek. Nepoužívají se ani koncové značky a zanoření je vytvářeno odsazením. Třídy a identifikátory jdou zapsat přímo za element, ostatní atributy se zapisují do závorek tak, jak je možné vidět na obrázku 7.1. Systém Jade tak oproti standartnímu zápisu v HTML umožňuje vytvářet přehlednější a méně obsáhlý kód.

```
form.form-signin(role='form', action="/register", method="post").registerForm
    h3.form-signin-heading Sign Up here
    input.form-control(type='text', name="username", placeholder='Username', required)
    input.form-control(type='password', name="password", placeholder='Password', required)
    input.form-control(type='email', name="email", placeholder='Email', required)
    button.btn.btn-lg.btn-primary.btn-block(type='submit') Sign Up
```

```
<form role="form" action="/register" method="post" class="form-signin registerForm">
  <h3 class="form-signin-heading">Sign Up here</h3>
  <input type="text" name="username" placeholder="Username" required="required" class="form-control"/>
  <input type="password" name="password" placeholder="Password" required="required" class="form-control"/>
  <input type="email" name="email" placeholder="Email" required="required" class="form-control"/>
  <button type="submit" class="btn btn-lg btn-primary btn-block">Sign Up</button>
</form>
```

Obrázek 7.1: Ukázka zápisu kódu v šablonovacím systému Jade a jeho ekvivalent v HTML.

Pro podporu ukládání sezení jsem použil balíček Express-session. Data sezení jsou ukládána na serveru a do cookies je ukládán pouze identifikátor sezení. Ve výchozím nastavení je pro ukládání použita paměť serveru. Autoři balíčku uvádí, že toto nastavení je doporučené pouze pro vývoj a lazení, protože často dochází k úniku paměti. Pro ukládání sezení do databáze jsem proto použil balíček Conncet-mongo. Díky němu zůstane sezení i po restartu serveru, což bylo během vývoje užitečné.

Pro restart serveru jsem používal balíček Nodemon. Ten slouží k automatickému restartování aplikace v případě detekce uložení modifikovaného zdrojového souboru. Umožňouje vybrat soubory, ve kterých jsou modifikace monitorovány. Výběr lze nastavit na specifické soubory, typy souborů nebo jejich umístění.

Jak bylo popsáno v části 4.3, spousta operací probíhá v Node.js asynchronně. Nezřídka se však stává, že některé operace jsou závislé na výsledku těch předchozích. Proto je užitečné mít možnost definovat pořadí operací, případně je sychronizovat. K tomu jsem využil balíček Async, který tuto práci vývojáři značně usnadňuje.

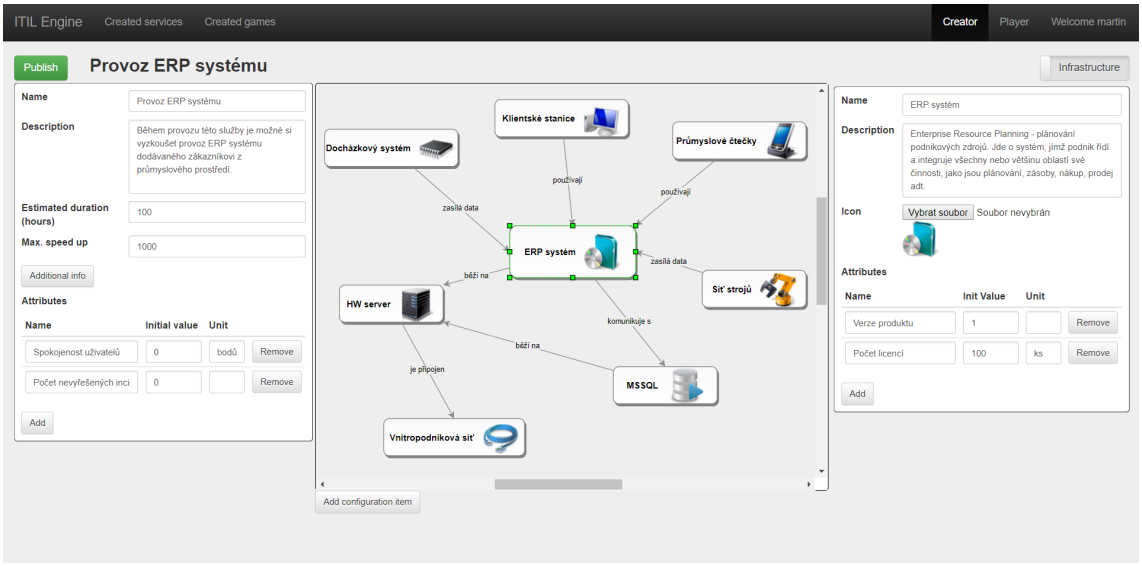
7.2 Autentizace

Pro přihlášení do systému musí mít uživatel vytvořený účet. Ten je možné vytvořit zadáním přihlašovacích údajů. Povinné údaje jsou přihlašovací jméno, heslo a email. Aby si uživatel nemusel pamatovat přihlašovací údaje do dalšího systému, implementoval jsem také přihlašování pomocí účtu třetí strany, konkrétně pomocí Google účtu.

K implementaci přihlašování využívám balíčků Passport a Passport-google-oauth20. Při přihlašování přes účet Google dojde k přesměrování na stránku s přihlašovacími údaji. Po úspěšném přihlášení jsou vráceny údaje o uživateli, včetně jeho access tokenu a emailové adresy. V případě, že se jedná o první přihlášení tohoto uživatele, tak je mu vytvořen nový účet.

7.3 Tvorba služby

Základem herního enginu byla implementace funkcionality pro vytváření služeb. První část – vytvoření základních atributů služby, konfiguračních položek a jejich zapojení do infrastruktury ukazuje snímek 7.2.



Obrázek 7.2: Obrazovka pro vytváření infrastruktury a údajů o službě.

Uprostřed obrazovky se nachází plátno pro zapojení konfiguračních položek do schématu. Plátno se v případě potřeby automaticky rozšiřuje a pohyb po něm je možný podržením pravého tlačítka myši nebo použitím scrollovacích posuvníků. Toto plátno je implementováno pomocí knihovny MxGraph.

Konfigurační položky lze přidávat tlačítkem `Add configuration item`. Přidanou konfigurační položku lze tahem myši ve schématu přemisťovat, případně měnit její velikost, pokud je svým způsobem významnější nebo se její název a ikona nevejdou do uzlu ve schématu. Tahem myši ze středu uzlu představujícího položku lze vkládat orientované hrany, které mohou vizuálně znázornit její vazbu na ostatní konfigurační položky. Hrany mohou nést textový popisek, který lze přidat dvojklikem na ně. Konfigurační položky i hrany lze odebírat jejich označením a stiskem klávesy `Delete`. Označená položka či hrana je ve schématu podbarvena zelenou barvou. V případě konfigurační položky se při označení navíc zobrazí její detail v pravé části obrazovky. V detailu lze modifikovat název položky, přidat její popis, ikonu a vlastní atributy.

U každého atributu je vyžadován jeho název a počáteční hodnota. Název atributu je dále využíván při tvorbě příkazů, které definují změnu hodnoty tohoto atributu v průběhu scénáře. Stejné názvy atributů v rámci jedné konfigurační položky by při tvorbě příkazů kolidovaly, proto nejsou povoleny. V názvu atributu jsou zakázány také speciální znaky, které by opět komplikovaly tvorbu příkazů. U atributů lze nepovinně zadat také jejich jednotku.

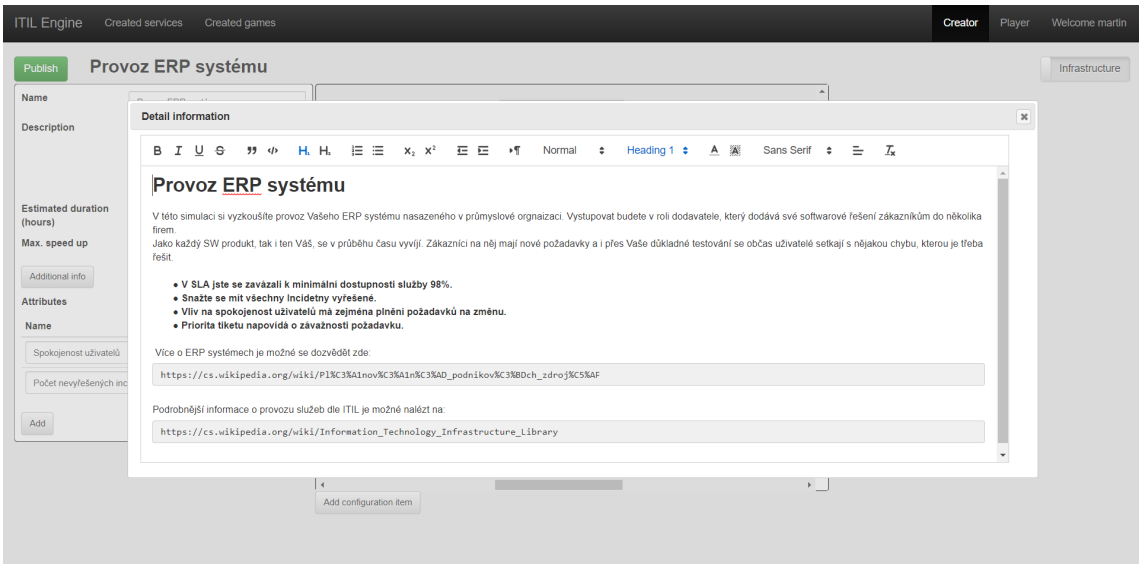
V levé straně obrazovky se nachází podrobnější údaje o službě. Je možné zadat název služby, popis a také odhadovanou dobu trvání jejího provozu (délky scénáře) v případě, že si hráč zvolí real-time provoz. Real-time provoz může hráč urychlit. Aby tvůrce mohl při definici scénáře počítat s nějakou minimální dobou trvání provozu služby, může zadat maximální zrychlení, jaké bude hráči k dispozici.

Během návrhu a konzultace prototypů aplikace jsem atributy, které by měly být u služby k dispozici, několikrát měnil. V úvahu připadaly například ekonomické údaje, jako náklady na provoz služby, zisk nebo třeba údaje spjaté s plněním SLA. Z těchto důvodů jsem se nakonec rozhodl namísto několika pevně definovaných atributů dát přednost možnosti přiřa-

zování vlastních atributů, podobně jako u konfiguračních položek. Tyto atributy jsou rovněž zpřístupněny při tvorbě příkazů, které mohou jejich hodnoty v průběhu scénáře měnit.

Doplňující informace o službě

Protože obyčejný textový popis služby nemusí být dostatečný pro úplné seznámení hráče se službou, byl k definici služby přidán tzv. rich text editor, viz obrázek 7.3.

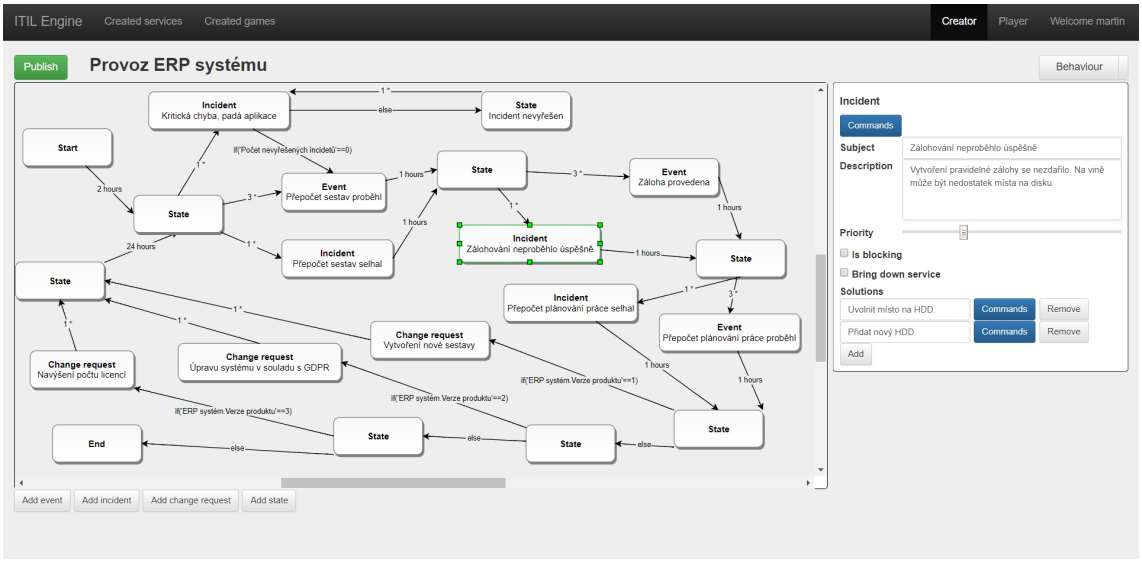


Obrázek 7.3: Textový editor pro doplňující informace o službě.

Ten umožňuje vytvoření delšího a bohatěji formátovaného textu. Pomocí něj může tvůrce poskytnout detailnější a strukturovaný popis služby. K dispozici jsou různé druhy, styly a velikosti písma, nadpisy nebo třeba hypertextové odkazy. Typicky je určen pro podrobnější informace o službě, odkaz na vlastní service desk, nastínění scénáře nebo třeba pro nápovědu k provozu služby. Takto vytvořený popis je potom hráčům k dispozici před i během provozu služby. Tento textový editor je implementován pomocí balíčku Quill. Jedná se o editor typu WYSIWYG (What you see is what you get), který ukládá data ve formátu JSON.

7.4 Tvorba scénáře provozu služby

Druhou částí vytvoření služby je definice jejího chování v průběhu jejího provozu, tedy vytvoření scénáře. Hlavní obrazovka pro tvorbu scénáře je na obrázku 7.4.



Obrázek 7.4: Hlavní obrazovka tvorby workflow grafu.

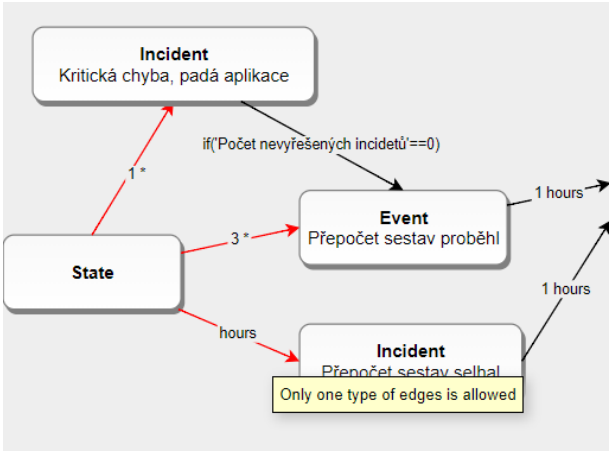
Většinu obrazovky zabírá plátno, které slouží pro tvorbu workflow grafu, jehož interpretace představuje scénář. Podobně jako při tvorbě schématu infrastruktury, i zde je využita knihovna MxGraph. Práce s plátnem je pro uživatele podobná, jako při vytváření schématu infrastruktury. Tlačítky ve spodní části obrazovky lze do grafu přidávat uzly, které v tomto případě představují stavy nebo tikety. Hrany představují přechody mezi nimi. Označením hrany nebo uzlu v grafu se na pravé straně zobrazí jejich detail.

V detailu hrany lze zvolit její typ. Při volbě časované hrany se zadává délka jejího trvání v hodinách. U pravděpodobnostní hrany lze zadat její váhu, která představuje pravděpodobnost, s jakou se hrana provede. V případě podmínkové hrany je třeba přidat podmínku, případně doplnit hranou `else`. Údaje dostupné v detailu uzlu závisí na typu uzlu. Typy uzlů a údaje, které mohou obsahovat jsou podrobněji popsány v části 6.2.

Scénář služby je reprezentován formou XML dokumentu. Tak s ním pracuje i knihovna MxGraph. Díky použití MongoDB je navíc velmi jednoduše možné graf uchovávat strukturovaně také v databázi, a to převodem z XML do JSON. Rekonstrukce grafu uloženého v databázi ve formátu JSON pak probíhá tak, že se nejprve provede převod zpět na XML, které je poté předáno knihovně MxGraph. Pro převod z XML do JSON a zpět jsem využil balíček Xml2js.

Validace workflow grafu

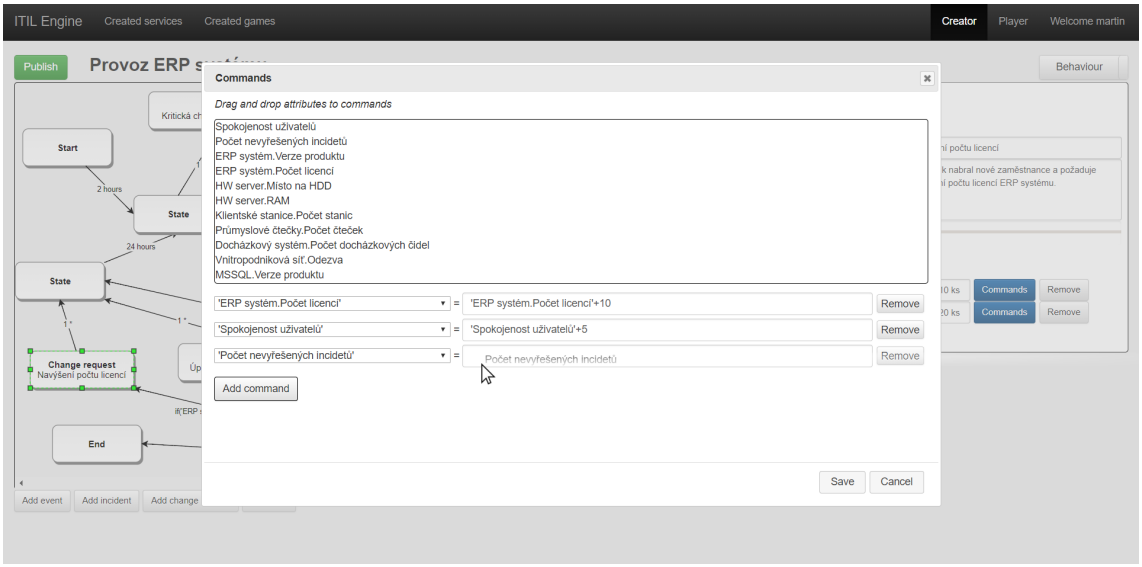
Při implementaci jsem se snažil, aby pro zápis scénáře uživatel nemusel znát syntaxi nějakého programovacího jazyka a byl pro něj co nejjednodušší. Tvorba nejpodstatnější části scénáře je tak vizualizována a probíhá pouze úpravami workflow grafu a jeho uzlů a hran. Aby se dal graf následně interpretovat, má jeho vytváření pravidla, která musí uživatel při jeho vytváření dodržovat. To se týká převážně hran. Podrobnosti o tom, jaké uzly a hrany lze přidávat a jaká pravidla pro práci s nimi platí, lze nalézt v sekci 6.2. Pokud uživatel přidá do grafu hranu, která není validní, je na to vizuálně upozorněn zvýrazněním chybné hrany, případně souvisejících hran. Najetím myši se zobrazí chybová hláška, která uživateli napovídá, o jaký typ chyby se jedná, viz obrázek 7.5.



Obrázek 7.5: Ukázka upozornění na chybu během vytváření workflow grafu.

Tvorba příkazů

Důležitou součástí tvorby scénáře je definice změn či výměn konfiguračních položek. Tyto změny konfigurace jsou realizovány prostřednictvím příkazů. Příkladem změny konfigurace služby může být navýšení kapacity HDD na serveru, doplnění barvy v tiskárně nebo třeba změna počtu licencí SW produktu (viz první příkaz na obrázku 7.6).



Obrázek 7.6: Tvorba příkazů, které umožňují změnit konfiguraci služby.

Díky možnosti přiřazovat libovolné atributy také ke službě, příkazy nemusí představovat jen změnu konfigurace u některé z položek. Mohou sloužit i k modifikaci libovolných vlastních údajů, které si tvůrce nadefinuje. Na obrázku 7.6 jsou ve druhém a třetím příkazu modifikovány atributy služby spokojenost uživatelů a počet nevyřešených incidentů, které mohou sloužit jako ukazatele pro hodnocení hráče při provozu dané služby.

Tyto příkazy je možné definovat pro každý uzel, kromě startu a konce. Uzel, který reprezentuje tiket, může mít svá řešení. Pro každé řešení je též možné definovat tyto příkazy.

Příkazy jsou tedy vykonány, pokud workflow graf přejde do následujícího uzlu (příkazy patřící k uzlu) nebo pokud hráč provede řešení nějakého tiketu (příkazy patřící k řešení).

Tímto způsobem může tvůrce modelovat typické situace vznikající během provozu služeb. Příkladem takové situace může být incident, kdy v tiskárně dojde barva. Tuto situaci lze simulovat přechodem do uzlu typu incident se stručným popisem k čemu došlo. Tento uzel může obsahovat příkaz, který způsobí změnu konfigurace a sníží atribut množství barvy v tiskárně na nula procent. Dále lze definovat řešení tohoto incidentu – doplnění barvy. Toto řešení může mít přiřazený příkaz, který množství barvy zvýší opět na sto procent.

Každý příkaz se skládá z levé a pravé strany. Aby byla definice příkazů validní, musí se na levé straně nacházet právě jeden atribut. Na pravé straně se může nacházet výraz složený z konstant či proměnných v podobě atributů. Aby si tvůrce služby nemusel pamatovat přesně názvy atributů nebo je ručně přepisovat, jsou mu při vytváření příkazů k dispozici v podobě seznamu, ze kterého je lze operací drag and drop přesouvat do jednotlivých definic příkazů. Sémantika příkazu je taková, že jakmile dojde během provozu služby na vyhodnocení příkazu, tak se vyhodnotí výraz na pravé straně a výsledek se přiřadí do proměnné na levé straně příkazu.

Zabezpečení provádění uživatelského kódu

Před uložením příkazů je provedena jejich validace. Do výrazu na pravé straně jsou za proměnné dosazeny fiktivní hodnoty a je provedeno jeho vyhodnocení. Pokud se na levé straně nenachází atribut nebo výraz na pravé straně nelze vyhodnotit jako hodnotu, není výraz validní a tvůrce jej nemůže uložit. Validace při vytváření příkazů nedokáže zachytit všechny neočekávané stavy, protože skutečné hodnoty se dosazují do výrazu až za běhu programu. Nelze tak předem vyloučit chyby jako třeba dělení nulou. Kromě chyb způsobených tvůrcem scénáře neúmyslně, bylo třeba vyhodnocení výrazů zabezpečit také proti úmyslnému vložení škodlivého kódu. Namísto příkazu složeného z konstant a proměnných služby se totiž ve výrazu může objevit například následující kód:

```
process.exit()
```

Protože je vyhodnocovaný kód závislý na vstupu od uživatele, nestačilo obyčejné použití funkce eval() a bylo zapotřebí jej vykonávat v zabezpečeném prostředí. Já za tímto účelem využil balíček vm2. Ten slouží jako tzv. sandbox pro zabezpečené provádění nedůvěryhodného kódu a měl by si poradit i s výrazem jako:

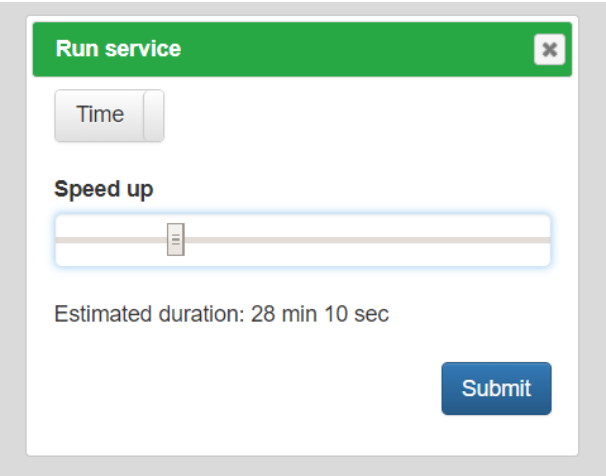
```
while (true) {}
```

Za nedůvěryhodný kód lze považovat jakýkoliv vstup zadávaný uživatelem, proto je stejného principu použito také při vyhodnocování výrazu v podmínkové hraně.

7.5 Provoz služby

Po tvorbě služby a scénáře jejího provozu je druhou hlavní částí enginu interpretace tohoto scénáře, tedy simulace provozu služby. Vytvořenou službu s validním scénářem může tvůrce publikovat. Tím ji zpřístupní ke spuštění. Publikovanou hru si mohou spustit všichni uživatelé v systému. Před spuštěním si hráč může zvolit, jestli chce provozovat službu v reálném či zrychleném čase anebo v tazích. Při volbě založené na čase má hráč k dispozici údaj o odhadované délce trvání. Odhadovanou délku trvání vyplňuje tvůrce scénáře. Její automatický výpočet by byl složitý a často zavádějící, protože tok workflow grafem může

být pokaždé jiný, a to kvůli rozdílným reakcím hráče a také pravděpodobnostním hranám, které ovlivňují průběh scénáře na základě náhody. V případě volby časového průběhu scénáře má hráč možnost délku trvání hry ovlivnit pomocí nastavení zrychlení, jak je ukázáno na obrázku 7.7.



Obrázek 7.7: Modální okno zobrazované před spuštěním služby. Slouží pro nastavení průběhu scénáře.

Po spuštění je hráč přesměrován na hlavní stránku hry (obrázek 7.8). Ta je určena pro obsluhu provozu služby. Vlevo na obrázku 7.8 lze vidět část obsahující obecné údaje vztažené ke službě. V rámečku č. 1 se nachází také vybrané ukazatele úspěšnosti provozované služby. Mezi ně patří dostupnost, spolehlivost, udržovatelnost a průměrná délka reakce na tiket. K dispozici je také nápověda, jak jsou vypočítávány. To může pomoci hráči k osvojení si těchto důležitých pojmů z oblasti ITIL. Pro tvůrce hry navíc mohou sloužit jako jedna z metrik pro hodnocení hráče. Tyto ukazatele jsou vypočítávány automaticky, aniž by s nimi tvůrce služby nějak musel pracovat. V rámečku č. 2 jsou potom atributy služby, které definoval její tvůrce. Jejich hodnoty se mohou měnit dle scénáře.

Uprostřed obrazovky je zobrazeno schéma konfiguračních položek. Kliknutím na ně lze zobrazit jejich detail včetně jejich aktuálního stavu, který je dán hodnotami jejich atributů.

V horní straně pravé části obrazovky se nachází tlačítka pro ovládání hry. Dole se pak nachází jednoduchý service desk. Skrze něj je hráč informován o nově vzniklých tiketech.

Obrázek 7.8: Obrazovka provozu služby z pohledu hráče.

Socket.io

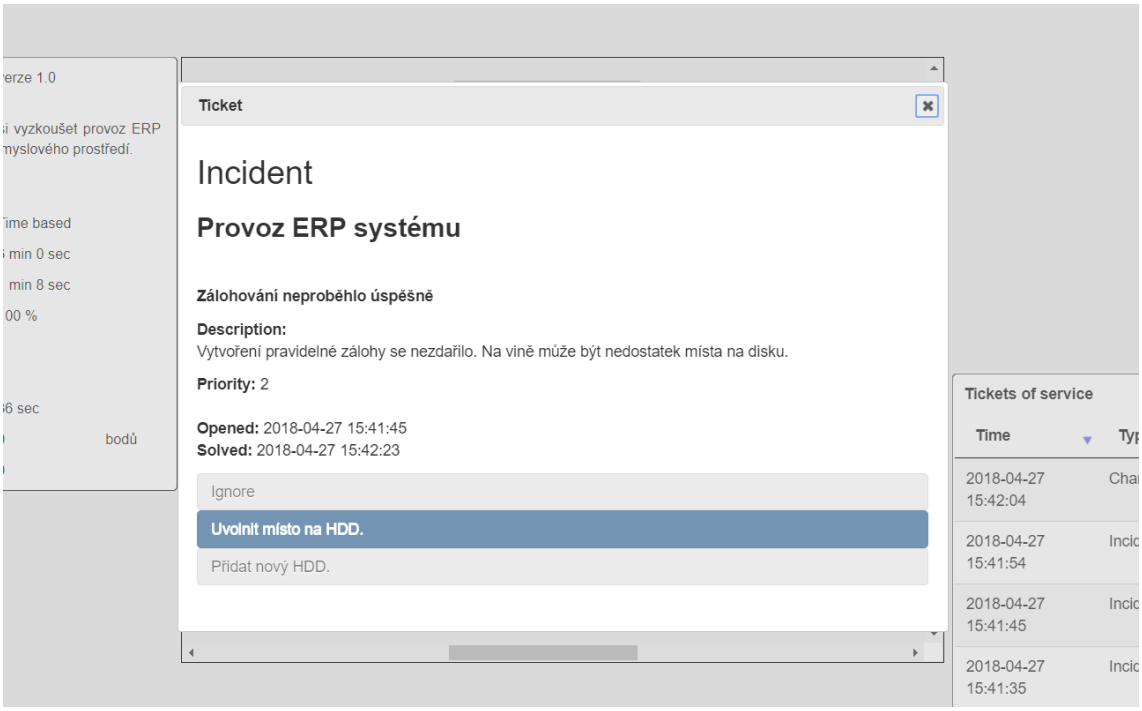
Při časovém průběhu hry vznikají na straně serveru nové tikety, přepočítávají se ukazatele ve hře, mění se konfigurace služby a jiné. To vše se děje na straně serveru na základě scénáře, aniž by bylo zapotřebí reakce uživatele. Proto bylo nutné zajistit aktuálnost údajů na straně klienta pomocí obousměrné komunikace. Některé techniky, jak toho lze u webových apliakcích docílit, byly popsány v části 4.2. Já si pro oboustranou komunikaci zvolil knihovnu Socket.io.

Komunikace probíhá tak, že při načtení stránky s hrou je ze strany klienta vyvolána událost, na základě které si server uloží dvojici ID hráče a jeho socket[1] pro pozdější komunikaci s ním. Jakmile ve hře nastane událost, na kterou má být hráč upozorněn, server jej vyhledá dle jeho identifikátoru a prostřednictvím socketu jej o události informuje.

Tikety

Tikety představují významné události ve scénáři. Jejich řešením je možné reagovat na průběh hry. Řešení tiketu je ukázáno na obrázku 7.9. Tiket obsahuje údaje tak, jak je definoval tvůrce scénáře. Dole je vidět seznam dostupných řešení, ze kterých může hráč vybírat. Navíc jsou přidány informace o času vytvoření a v případě vyřešeného tiketu také čas vyřešení a řešení, které hráč zvolil.

[1] Socket.IO není implementace protokolu WebSocket, ačkoli tento protokol využívá, když je k dispozici. Socket je v tomto případě označení pro jeho interní reprezentaci touto knihovnou.



Obrázek 7.9: Ukázka detailu tiketu.

S tikety lze pracovat prostřednictvím rozhraní REST. Pro získání tiketů ve hře je k dispozici URI:

```
GET /player/games/tickets/:id
```

Ukázka odpovědi, která obsahuje jeden z tiketů je na obrázku 7.10. V ukázce jsou pro přehlednost vynechány identifikátory dokumentů a jiné detailní informace. Vyřešení tiketu lze provést požadavkem:

```
PUT /player/games/tickets/:id
```

V těle tohoto požadavku je očekáván identifikátor hry a zvoleného řešení:

```
{
  gameId: "5afe910e7c373e5a04be5edf",
  solutionId: "5ae31ee2c6d54883a825e178"
}
```

Vyřešení tiketu může znamenat změnu konfigurace služby podle toho, jak to definoval tvůrce scénáře. Vyřešením tiketu dochází k posunu ve hře a přepočtu průměrné doby reakce na tiket. V případě, že jde o incident, který způsobil výpadek služby, dojde také k obnovení dostupnosti služby, přepočtu udržovatelnosti služby a dalších ukazatelů.

```
{
    "created": "2018-04-27T13:41:45.000Z",
    "state": {
      "subject": "Zálohovaní neproběhlo úspěšně",
      "description": "Vytvoření pravidelné zálohy se nezdařilo.
                      Na vině může být nedostatek místa na disku.",
      "priority": "2",
      "commands": [
        "'5ae31edcc6d54883a825e0da'=0"
      ],
      "isBlocking": false,
      "bringDown": false,
      "type": "Incident",
      "service": "5ae31edbc6d54883a825e0cd",
      "solutions": [
        {
          "description": "Ignore",
          "isIgnore": true,
          "state": "5ae31edcc6d54883a825e10c",
        },
        {
          "description": "Uvolnit místo na HDD.",
          "commands": [
            "'5ae31edcc6d54883a825e0da'=200"
          ],
          "isIgnore": false,
          "state": "5ae31edcc6d54883a825e10c",
        },
        {
          "description": "Přidat nový HDD.",
          "commands": [
            "'5ae31edcc6d54883a825e0da'=1000"
          ],
          "isIgnore": false,
          "state": "5ae31edcc6d54883a825e10c",
        }
      ]
    },
    "status": "solved",
    "selectedSolution": "5ae31edcc6d54883a825e127",
    "solved": "2018-04-27T13:42:23.000Z"
  },
```


Obrázek 7.10: Ukázka struktury tiketu generovaného scénářem. Některé údaje jsou pro přehlednost vynechány.

Na vznik nového tiketu může být hráč upozorněn také prostřednictvím emailu. Pro odesílání emailů je použita adresa, kterou uživatel vyplnil při registraci. Ve výchozím nastavení jsou tato upozornění vypnuta. Zapnout a upravit emailovou adresu je možné v uživatelském profilu.

Zabezpečení odesílání emailových upozornění

Upozornění jsou hráči odesílána v případě, že se mu v service desku vyskytne nová událost, požadavek na změnu či incident. Kdy se tak stane, záleží na aktivitě hráče a především na scénáři hry. Tato funkcionalita by mohla být zneužita tak, že by tvůrce scénáře vytvořil scénář s nekonečnou smyčkou, která generuje tikety. Pro případ časového průběhu hry je v aplikaci sice implementováno minimální zpoždění, které zabraňuje okamžitým přechodům mezi stavy, nicméně toto omezení je v řádu vteřin a v případě odesílání emailů by se i tak jednalo o velké množství.

Aby se předešlo zneužití mechanismu zasílání emailů, bylo implementováno omezení maximálního počtu odeslaných mailů jednomu hráči za minutu. Tuto hodnotu lze nastavit při spuštění serveru.

Kapitola 8

Testování a vyhodnocení

Tato kapitola je věnována testování a vyhodnocení aplikace. Stručně jsou zde popsány poznatky uživatelů, kterým byla aplikace dána k dispozici za účelem získání zpětné vazby.

8.1 Testování aplikace

Testování jsem prováděl manuálně pomocí programu Advanced REST client. Ten je možné nainstalovat pod prohlížečem Google Chrome. Jedná se o nástroj určený k testování aplikačního rozhraní serveru, který nabízí grafické rozhraní pro pohodlné zasílání HTTP požadavků na server a zobrazení odpovědí. Užitečnou funkcí byla zejména nápověda v podobě automatického doplňování při psaní příkazů. K dispozici je také import/export dat a integrace s cloudovým uložištěm Google Drive.

Pro experimentální ověření funkcionality a ladění frontendové části aplikace jsem používal vývojářské nástroje dostupné v Google Chrome. Kontroloval tak jsem zejména provádění javascriptového kódu, strukturu vygenerovaného HTML dokumentu a jeho správné vykreslení pomocí kaskádových stylů.

Kvůli možným rozdílům v internetových prohlížečích bylo důležitou součástí také ověření funkčnosti aplikace na vybraných zástupcích. To jsem provedl na prohlížečích Google Chrome ve verzi 66, Mozilla Firefox ve verzi 59 a Microsoft Edge verze 41.

8.2 Vyhodnocení

Pro vyhodnocení vytvořené aplikace jsem oslovil čtyři uživatele se znalostí principů ITIL. Ti byly nejprve seznámeni s aplikací. Na dvou předem vytvořených službách jim byl demonstrován koncept vytváření služby, jejího scénáře a následného provozu.

Následně dostal každý z uživatelů prostor, aby si mohl vytvořit vlastní službu a vyzkoušet si její provoz. Během toho jsem pozoroval, jak si počínají při ovládání aplikace a zodpovídal případné dotazy. Následně jsem formou strukturovaného pohovoru zjišťoval jejich názor na aplikaci. Na závěr uživatelé dostali prostor pro vyjádření vlastních poznatků a postřehů.

Z rozhovorů vyplynulo, že uživatelé, kteří měli nějakou předchozí zkušenost s prací s nástroji, které využívají workflow (SSIS z MSSQL, ARIS Designer), byly s vizualizovanou tvorbou scénáře prostřednictvím grafu spokojeni a rychle si ji osvojili. Ostatní potřebovali více nápovědy, ale ve výsledku zhodnotili, že návrh pomocí s workflow pro ně byl přijatelnější

než práce čistě se zdrojovým kódem. Uživatelé na aplikaci ocenili možnost osvěžit si pojmy a principy metodiky ITIL a za zajímavé považovali také možnost časového průběhu hry.

Z pohledu možných vylepšení jsem obdržel podněty na vylepšení vzhledu aplikace nebo přidání ukazatelů informujících o ukládání či načítání stránky. Pro uživatele by byla zajímavá také pokročilejší možnost práce s tikety, která by zahrnovala více stavů, možnost jejich znovuotevírání a další. Pokročilejší práce s tikety by mohla zahrnovat například eskalaci incidentu na výšší úroveň podpory v případě, že by se jej hráči nedařilo vyřešit. V případě možnosti hrát hru více uživateli, by pak bylo možné tikety přidělovat k řešení různým hráčům, a na průběhu hry tak spolupracovat.

Kapitola 9

Závěr

V rámci diplomové práce jsem nastudoval problematiku správy služeb dle knihovny ITIL. Za účelem tvorby enginu jsem také nastudoval principy workflow grafů, technologie pro jejich modelování a seznámil se tvorbou webových aplikací, které komunikují v reálném čase.

Ve fázi specifikace požadavků jsem analyzoval existující řešení, která jsou zaměřena na nácvik provozu IT služeb. Zaměřil jsem se hlavně na práci ITIL trenažér, protože výsledná aplikace z ní vychází. Autor v ní zmiňuje několik možností na vylepšení či rozšíření, která jsem při specifikaci požadavků zohlednil.

Část návrhu systému byla velmi zajímavá, protože ITIL je rozsáhlá knihovna a bylo důležité z ní vybrat to nejvýznamnější a zakomponovat to do enginu. Bylo třeba vymyslet a popsat, jakým způsobem bude engine pracovat, jakou roli v něm bude mít tvůrce her a jaké funkce budou mít k dispozici hráči. V rámci práce jsem navrhnul, jak budou vytvářeny hry a jak budou fungovat herní módy. Popsal jsem také způsob vytváření herních scénářů, které jsou založené na návrhu IT služeb a specifikaci jejich chování. Za tímto účelem jsem vytvořil koncepci workflow, která umožňuje manipulovat s definovanou konfigurací služby, vytvářet tikety a reagovat na jejich řešení hráčem.

Navržený systém jsem implementoval jako webovou aplikaci založenou na platformě Node.js a NoSQL databázi MongoDB. Engine nabízí nabízí možnost návrhu služby, její infrastruktury a scénáře jejího provozu pomocí workflow. Interpretace tohoto scénáře je možná v reálném či zrychleném čase nebo v tazích. Tikety generované scénářem jsou dle principů REST dostupné pomocí svého URI a mimo to s nimi hráč může pracovat pomocí jednoduchého service desku, který je součástí aplikace.

Během implementace bylo důležité naučit se pracovat s knihovnou MxGraph určenou pro tvorbu workflow grafu, vypořádat se s oboustranou komunikací v reálném čase a také s netradičním přístupem asynchronního programování na platformě Node.js

Vyhodnocení výsledků jsem provedl ve spolupráci s uživateli, kterým byla aplikace představena a následně dána k dispozici na testování. Na základě jejich poznatků by zřejmě nejzajímavějším rozšířením byla možnost spolupráce více hráčů při provozu služby, delegace či eskalace tiketů na některé ze spoluhráčů a obecně pokročilejší možnosti práce s tikety.

Výsledná aplikace může z pohledu tvůrce sloužit k návrhu vlastních IT služeb a k vytvoření modelu popisujícího jejich provoz. Simulaci takto navržené služby lze provádět v hráčské části. Během toho si hráč může prakticky vyzkoušet pojmy či postupy z knihovny ITIL, což může být využito k výukovým účelům.

Literatura

[1] *BESTPRACTICE.CZ, IT and Management Knowledge Base*. [Online; cit 12.12.2017]. URL https://www.bestpractice.cz/cs/Best-practice/-ITSM-ITIL-/-Historie-a-vyvoj-ITIL-.alej

[2] *ITIL® Service Strategy*. [Online; cit. 12.11.2017]. URL http://www.bmc.com/guides/itil-service-strategy.html

[3] *State machine diagram - diagram stavového stroje*. [Online; cit. 25.3.2018]. URL http://mpavus.wz.cz/uml/uml-b-activity-3-2-3.php

[4] BUCKSTEEG, M.; EBEL, N.; EGGERT, F.; aj.: *ITIL 2011 Stručný a srozumitelný výklad*. Computer Press, 2012, ISBN 978-80-251-3732-1.

[5] Cannon, D.; Wheeldon, D.; Lacy, S.; aj.: *ITIL® Service Strategy*. The Stationery Office, 2011, ISBN 978-0-11-331304-4.

[6] Dorda, M.: *Úvod do Petriho sítí*. [Online; cit. 25.3.2018]. URL http://homel.vsb.cz/~dor028/Nekonvencni_metody_1.pdf

[7] Dvořák, P.: *ITIL trenažér*. Diplomová práce, FIT VUT v Brně, Brno, 2013.

[8] Greene, N.: *What is a Workflow: Definition, Benefits and Examples*. [Online; cit. 20.3.2018]. URL https://tallyfy.com/what-is-a-workflow/

[9] Havlena, M.: *Node.js – How it differs from traditional web servers?* [Online; cit. 25.4.2018]. URL http://www.havlena.net/en/programming/node-js-how-it-differs-from-traditional-web-servers/

[10] Hunnebeck, L.; Rudd, C.; Lacy, S.; aj.: *ITIL® Service Design*. The Stationery Office, 2011, ISBN 978-0-11-331305-1.

[11] Kennedy, P.: *Overview of Model-View-Controller (MVC)*. [Online; cit. 30.4.2018]. URL http://www.patricksoftwareblog.com/overview-of-model-view-controller-mvc/

[12] Kit, A. C. Y.: *Static vs. Dynamic Web Sites*. [Online; cit. 25.4.2018]. URL http://adriancyk.blogspot.cz/2015/05/static-vs-dynamic-web-sites.html

[13] Kunstová, R.; Carda, A.: *Nástroj manažera pro řízení podnikových procesů*. GRADA, 2006, ISBN 80-247-0666-0.

[14] Šárka Květoňová: *Strategické řízení informačních systémů (Úvod k předmětu).* [Online; cit. 20.3.2018].
URL https://wis.fit.vutbr.cz/FIT/st/course-files-st.php?file=%2Fcourse%2FSRI-IT%2Flectures%2FSRI_1_uvod.pdf&cid=11647

[15] Malý, M.: *Komunikace v reálném čase díky Server-Sent Events a Web Sockets.* [Online; cit. 25.4.2018].
URL https://www.zdrojak.cz/clanky/kometa-prinasi-web-v-realnem-case

[16] Malý, M.: *REST: architektura pro webové API.* [Online; cit. 27.4.2018].
URL https://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/

[17] Mumbaikar, S.; Padiya, P.: *Web Services Based On SOAP and REST Principles. International Journal of Scientific and Research Publications*, 2013, ISSN 2250-3153.

[18] Munro, J.: *An Introduction to Mongoose for MongoDB and Node.js.* [Online; cit. 27.4.2018].
URL https://code.tutsplus.com/articles/an-introduction-to-mongoose-for-mongodb-and-nodejs--cms-29527

[19] Olakara, A. R.: *Understanding the Node.js event loop.* [Online; cit. 26.4.2018].
URL
http://abdelraoof.com/blog/2015/10/28/understanding-nodejs-event-loop/

[20] Rychlý, M.: *Přednášky k předmětu návrh a implementace IT služeb.* [Online; cit 12.12.2017].
URL http://www.fit.vutbr.cz/study/course-l.php.cs?id=12171

[21] Rychlý, M.; Kolář, D.: *NoSQL databáze.* [Online; cit. 26.4.2018].
URL http://www.fit.vutbr.cz/~rychly/public/docs/slides-nosql-databases/slides-nosql-databases.print.pdf

[22] Salvet, P.: *Komunikace v reálném čase díky Server-Sent Events a Web Sockets.* [Online; cit. 25.4.2018].
URL https://www.interval.cz/clanky/komunikace-v-realnem-case-diky-server-sent-events-a-web-sockets

[23] Steinberg, R.; Rudd, C.; Lacy, S.; aj.: *ITIL® Service Operation.* The Stationery Office, 2011, ISBN 978-0-11-331307-5.

[24] Thejendra, B.: *Practical IT Service Management: A concise guide for busy executives.* IT Governance Publishing, 2014, ISBN 978-1-84928-547-6.

[25] Tick, J.: *P-Graph-based Workflow Modelling.* John von Neumann Faculty of Informatics, 2007, [Online; cit. 25.3.2018].
URL https://www.uni-obuda.hu/journal/Tick_9.pdf

[26] Vašíček, P.: *Úvod do BPMN.* [Online; cit. 28.3.2018].
URL http://bpm-sme.blogspot.cz/2008/03/3-uvod-do-bpmn.html

[27] Vondrák, I.: *Metody byznys modelování.* Vysoká škola báňská - Technická univerzita Ostrava Fakulta elektrotechniky a informatiky, 2004, [Online; cit. 25.3.2018].
URL http://vondrak.cs.vsb.cz/download/Metody_byznys_modelovani.pdf