Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Diplomová práce

Návrh pojišťovnického systému

Jan Močko

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Jan Močko

Informatika

Název práce

Návrh pojišťovnického systému

Název anglicky

Design of the insurance system

Cíle práce

Cílem práce je analýza původního IS a následné zpracování návrhu nového IS včetně dokumentace, který reflektuje negativní body z analýzy původního IS. Nový IS by měl představovat all-in-one řešení, které v praxi představuje zjednodušení workflow uživatelů včetně poskytnutí snadné platformy pro správu pojistných smluv a jejich vyplácení bez potřeby integrací externích systémů.

Metodika

Práce bude složena ze dvou částí. První teoretickou část představí literární rešerše odborné literatury pro danou problematiku. Budou zde popsány přístupy, které budou podkladem pro část praktickou.

Praktická část bude složena ze slovního zhodnocení slabých částí původního IS a návrhu nového řešení s ohledem na analýzu původního IS. Návrh bude prováděn pomocí notace UML a BPMN.

Doporučený rozsah práce

60-80 stran

Klíčová slova

XML, ASP, Třída, Duplicitní transakce, Požadavek, Delegace, Role

---

Doporučené zdroje informací

Barlow, Mike. Evolving Architectures of FinTech. Sebastopol : O'Reilly, 2016. ISBN 9781491967768.

Martin, Robert C.; Clean Architecture: A Craftsman's Guide to Software Structure and Design. 1st Edition. New York: Pearson, 2017. ISBN 978-0134494166.

MILES, Rob. Exam Ref 70-483 Programming in C#. 2nd edition. New York: Pearson Education, 2018. ISBN 978-1509306985.

Předběžný termín obhajoby

2023/24 LS – PEF

Vedoucí práce

Ing. Martin Pelikán, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 10. 9. 2023

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 3. 11. 2023

doc. Ing. Tomáš Šubrt, Ph.D.

Děkan

V Praze dne 30. 03. 2024

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Návrh Pojišťovnického systému" jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 31.3. 2024 _____

Poděkování

Rád bych touto cestou poděkoval Ing. Martinu Pelikánovi Ph.D. za vedení mé diplomové práce za pečlivou zpětnou vazbu, cenné rady a čas, který mi byl věnován během tvorby této práce. Dále bych rád poděkovat společnosti SABO za možnou spolupráci na projektu v rámci této diplomové práce.

Návrh pojišťovnického systému

Abstrakt

Diplomová práce analyzuje stávající informační systém využívaný v pojišťovně a navrhuje jeho náhradu systémem novým, který slibuje efektivnější platformu pro posuzování událostí a správu smluv pro zaměstnance pojišťovny.

Teoretická část poskytuje přehled o IT aplikacích v pojišťovnictví, zkoumá problematiku návrhu odolných informačních systémů (IS), a představuje moderní přístupy a metodiky v softwarovém inženýrství a vývoji.

Praktická část se věnuje detailní analýze současné platformy, včetně vytvoření doposud chybějící dokumentace, identifikace a popisu klíčových problematických oblastí pomocí textové i grafické dokumentace, využívající nástroje jako UML a BPMN. Na základě této analýzy byly formulovány požadavky pro návrh nového systému, rozpracované skrze uživatelské scénáře, které byly v rámci návrhu nadefinovány v celém znění, a nakonec byl návrh vizualizován pomocí UML a BPMN diagramů s důrazem na zjištění z analýzy. Práce vyústila ve vytvoření dokumentace existujícího systému, detailní návrh nového systému, včetně definice klíčových procesů pomocí BPMN, a grafického návrhu uživatelského rozhraní v nástroji Figma.

Klíčová slova: Třída, Transakce, Delegace, Role, Proces, Entita, Objekt, Událost

Design of an Insurance System

Abstract

This thesis analyzes the existing information system used in an insurance company and proposes its replacement with a new system that promises a more efficient platform for event assessment and contract management for insurance company employees.

The theoretical part provides an overview of IT applications in insurance, explores the issue of designing resilient information systems (IS), and introduces modern approaches and methodologies in software engineering and development.

The practical part focuses on a detailed analysis of the current platform, including the creation of previously missing documentation, identification, and description of key problematic areas using both textual and graphical documentation, utilizing tools such as UML and BPMN. Based on this analysis, requirements for the design of the new system were formulated, elaborated through user scenarios that were defined in full in the design phase, and ultimately, the design was visualized using UML and BPMN diagrams with an emphasis on findings from the analysis. The work resulted in the creation of documentation for the existing system, a detailed design of the new system, including the definition of key processes using BPMN, and the graphical design of the user interface in Figma.

Keywords: Class, Transaction, Delegation, Role, Process, Entity, Object, Event

Obsah

1 Úvod

Pojišťovnické informační systémy mají klíčovou roli v efektivním a spolehlivém fungování pojišťovacích společností. Tyto systémy umožňují správu pojistných smluv, řízení škodních událostí, finanční analýzy a další důležité procesy spojené s pojišťovnictvím. V dnešním konkurenčním prostředí je důležité, aby pojišťovny disponovaly moderním a efektivním informačním systémem, který dokáže usnadnit a zrychlit každodenní práci zaměstnanců, zvýšit kvalitu poskytovaných služeb a minimalizovat chybovost.

Motivací pro výběr tématu "Návrh a implementace pojišťovnického informačního systému" je právě potřeba vyhodnotit stávající informační systém v pojišťovně a identifikovat jeho nedostatky a slabá místa. Na základě této analýzy bude vytvořen návrh nového IS, který bude reflektovat požadavky a potřeby pojišťovny a přinést zlepšení ve všech relevantních oblastech.

Systém, který je předmětem analýzy je komplikovaný, a hlavně nesystematicky navrhnutý. Spousta funkcionalit původního systému není navrhnuta robustně a způsobuje nepříjemné problémy, které je za potřebí řešit často ručními zásahy do databáze.

Tyto problémy sahají až do situací, kdy dojde k vytvoření duplicitních transakcí, nebo dokonce zaseknutí transakcí ve stavech, ze kterých se nemohou dostat a následně není možné jej vyplatit.

Tato práce by měla poukázat na tyto problémy a navrhnout robustnější řešení pro problematické části a principy aplikace.

Cílem této práce je poskytnout pojišťovně návrh nového all-in-one řešení, které zjednoduší a zefektivní workflow uživatelů. Nový informační systém bude sloužit jako jednotné a integrované prostředí pro správu pojistných smluv, škodních událostí, financí a dalších souvisejících procesů. Implementace systému na moderních technologiích a s využitím solidního databázového řešení umožní rychlé a spolehlivé zpracování dat a poskytne uživatelům snadný přístup k potřebným informacím.

Význam tématu spočívá ve zlepšení výkonnosti a konkurenceschopnosti pojišťovny. Nový informační systém by měl přinést řadu výhod, jako je zvýšená efektivita práce, rychlejší reakce na požadavky klientů, snížení chybovosti a zvýšení spokojenosti

zákazníků. Navrhovaný systém by měl také umožnit snadnou integraci s externími systémy a poskytovat širokou škálu analytických nástrojů pro lepší rozhodování na základě dat.

Celkově lze konstatovat, že návrh a implementace pojišťovnického informačního systému představuje významný krok k modernizaci a zlepšení procesů v pojišťovnictví. Tato práce nabízí příležitost pro zdokonalení stávajících postupů a zajištění vyšší efektivity a konkurenceschopnosti pojišťovací společnosti.

2 Cíl práce a metodika

2.1 Cíl práce

Cílem této diplomové práce je zhodnotit původní pojišťovnický informační systém a na základě toho vypracovat návrh nového informačního systému, který adresuje nedostatky a problémy identifikované v původním systému. Nový informační systém bude sloužit jako all-in-one řešení, které přináší zjednodušení pracovních postupů pro uživatele a poskytuje snadnou platformu pro správu pojistných smluv a jejich vyplácení, aniž by bylo nutné integrovat externí systémy.

Hlavními cíli práce jsou:

1. Analýza původního informačního systému: Provést důkladnou analýzu původního pojišťovnického informačního systému a identifikovat jeho slabé části a nedostatky. Zhodnotit negativní body, které ovlivňují efektivitu, výkonnost a uživatelskou přívětivost systému.
2. Návrh nového informačního systému: Na základě analýzy původního systému vypracovat návrh nového informačního systému, který reflektuje identifikované nedostatky. Navrhnout integrované prostředí pro správu pojistných smluv, škodních událostí, financí a dalších procesů, které jsou klíčové pro pojišťovny.
3. Zhodnocení a vyhodnocení: Porovnat výsledný informační systém s původním systémem a zhodnotit dosažené vylepšení a zlepšení pracovních postupů. Analyzovat efektivitu nového systému a jeho přínosy pro pojišťovnu.

Cíle této práce jsou zaměřeny na vypracování kvalitního a efektivního informačního systému pro pojišťovnu, který zvýší produktivitu zaměstnanců, zlepší služby poskytované klientům a přispěje ke zvýšení konkurenceschopnosti pojišťovací společnosti.

2.2 Metodika

Metodika k této diplomové práce vychází z nasbíraných poznatků z literárních zdrojů, které jsou zpracovány v literární části práce. Tyto poznatky se zaměřují na tvorby informačních systémů s přihlédnutím do tématické oblasti pojišťoven.

V další částí je vypracována analýza, dokumentace a popis původního systému. Tato analýza je zároveň vstupem pro návrh nového informačního systému.

Analýza původního systému je prováděna textovým popisem a grafickým znázorněním důležitých částí pomocí notace UML a BPMN. Taktéž se zde uplatňovala forma rozhovoru při analýze potíží od klíčových uživatelů systému včetně osobního testování pro ověření chyb, které byly nahlášeny. Z této analýzy byly nadefinovány uživatelské scénáře a směr, kterým by se návrh IS měl ubírat.

Následně je proveden návrh nového systému, kde jsou znovu použity techniky softwarového inženýrství pro návrh architektury a procesů systému. Pomocí datového modelování jsou zde popsány hlavní modely. Vzhledem k složitostí procesů je pro znázornění datových toků použito BPMN. V návrhu je taktéž navržena architektura technologií informačního systému.

Na závěru je práce doplněna o návrh možného uživatelského rozhraní, které naplňuje požadavky stanovené v případech užití a respektuje předtím navržené modely. Jedná se o grafický návrh, který byl prováděn pomocí programu Figma.

3 Teoretická východiska

3.1 IS v pojišťovacím sektoru

Pojišťovnictví je odvětví ekonomiky, které se zabývá poskytováním pojistných produktů a služeb za účelem převzetí rizika od klientů a ochrany jejich majetku, zdraví, života nebo jiných hodnot. Hlavním principem pojišťovnictví je rozložení rizika mezi větší skupinu pojištěných osob, které přispívají finančními příspěvky (pojišťovacími příspěvky) do společného fondu, ze kterého jsou vypláceny pojistné plnění v případě nastání pojištěné události. (Slavík, 2013)

Pojišťovnictví se vyznačuje několika specifickými rysy:

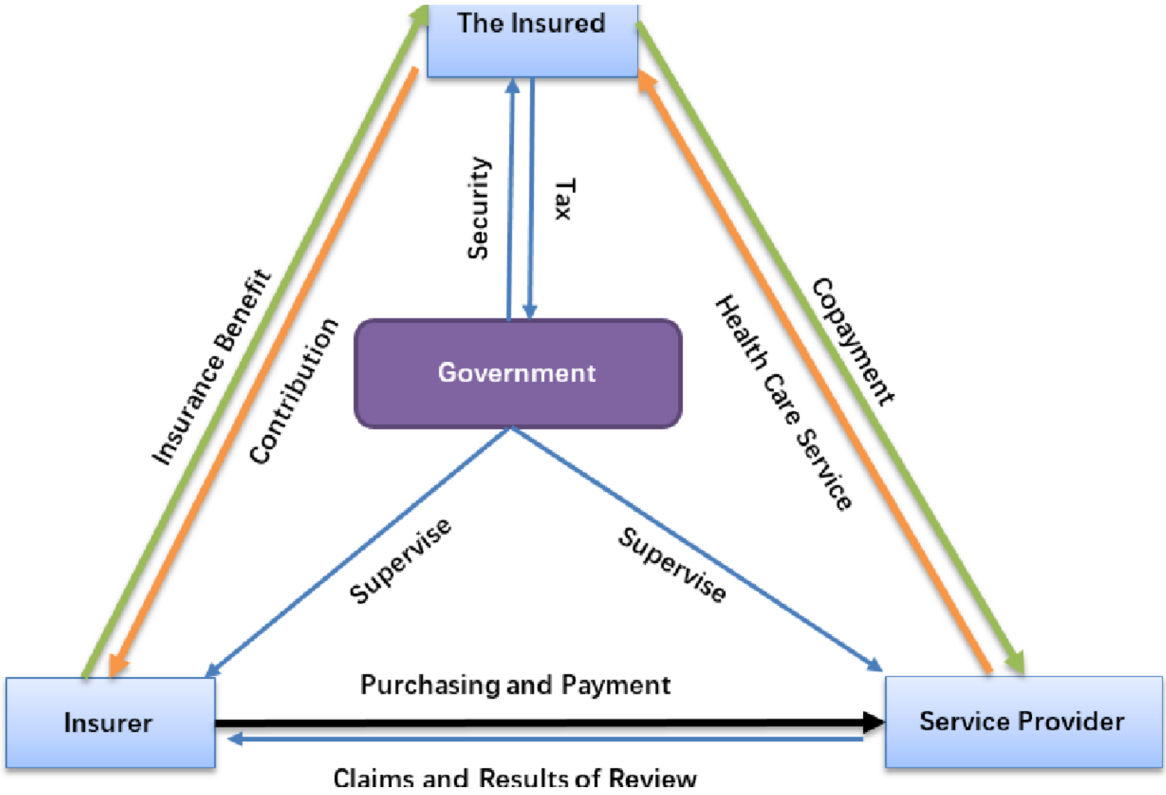
1. Pojištění a pojistné smlouvy: Pojišťovnictví je založeno na uzavírání pojistných smluv mezi pojišťovnou a pojištěnou osobou. Tyto smlouvy stanovují podmínky pojištění, rozsah pojistných krytí, výši pojistného plnění a další důležité aspekty.
2. Prémie a pojistné plnění: Pojištěná osoba (pojištěný) platí pojišťovně prémii (pojistné), která je finančním příspěvkem za poskytnutou pojistnou ochranu. Pokud nastane pojištěná událost v souladu s podmínkami pojistné smlouvy, pojišťovna vyplácí pojistné plnění (indemnizaci) pojištěné osobě.
3. Riziko a hodnocení rizika: Pojišťovna provádí analýzu rizika spojeného s poskytovanými pojistnými produkty. Hodnocení rizika zahrnuje analýzu pravděpodobnosti nastání pojištěné události a jejího dopadu. Na základě tohoto hodnocení je stanovena výše pojistného a podmínky pojistné smlouvy.
4. Aktuárská činnost: Aktuáři jsou specialisté, kteří se zabývají matematickým a statistickým modelováním rizik a výpočtem pojistných tarifů. Provádějí analýzu statistických dat, pravděpodobnostní výpočty a stanovují ceny pojistných produktů.
5. Regulace a dozor: Pojišťovnictví je většinou přísně regulováno státem nebo jinými orgány dozoru. Cílem regulace je zajištění stability a důvěry v pojišťovací trh, ochrana zájmů pojištěných osob a spravedlivá soutěž mezi pojišťovnami.

Pojišťovnictví je důležitou součástí moderního ekonomického systému a poskytuje ochranu před riziky a nepředvídatelnými událostmi. Tato kapitola je důležitá pro

pochopení stávajícího systému a vyplývají z ní další klíčové pojmy pro pochopení návrhu a implementace nového pojišťovnického informačního systému. (Slavík, 2013)

Informační systémy (IS) hrají v pojišťovnictví klíčovou roli a mají zásadní vliv na efektivitu, konkurenceschopnost a schopnost pojišťoven plnit své závazky vůči klientům. Tato kapitola se zaměří na význam a různé aspekty role informačních systémů v pojišťovnictví. (Yazi, Chunji, Yang, 2021)

Obrázek 1 - Pyramida pojišťovacího systému



Zdroj: Yazi, Chunji, Yang, 2021

3.1.1 Automatizace a zpracování dat

Automatizace a zpracování dat jsou zásadní pro efektivní fungování pojišťovacích společností. S narůstajícím objemem informací spojených s pojistnými smlouvami, klienty a škodami se manuální zpracování stává neudržitelné. Informační systémy poskytují centralizovanou databázi pro všechny relevantní údaje a umožňují jejich rychlé a přesné zpracování. To zahrnuje registraci nových pojistek, správu plateb, řízení pojistných

událostí a správu investičního portfolia. Díky automatizaci lze snížit riziko chyb z manuálního zpracování a zvýšit rychlost a přesnost transakcí. (boost.ai, 2023)

Automatizace rovněž umožňuje rychlé a efektivní generování reportů a analýzu dat pro řízení rizik. Toto je klíčové pro aktuárskou práci, která je základem pro určení cen pojistných produktů a rezerv na pokrytí budoucích škod. Informační systémy tak výrazně přispívají k operativní efektivitě a redukci nákladů, což má pozitivní dopad na ziskovost a konkurenceschopnost pojišťovny. (leadsquared.com, 2023)

3.1.2 Řízení rizik

Řízení rizik je dalším klíčovým prvkem v pojišťovnictví, kde informační systémy hrají zásadní roli. Pojišťovny čelí různým druhům rizik, od pojistných až po investiční. Informační systémy umožňují sběr, analýzu a správu dat nezbytných pro hodnocení a řízení těchto rizik. Aktuáři využívají informační systémy k vytváření modelů pro analýzu rizik a odhad budoucích ztrát, což umožňuje rychlou reakci na změny na trhu a přizpůsobení pojistných tarifů a strategií řízení rizik. Tyto systémy také usnadňují sledování portfolia pojistných smluv a předpovídání potenciálních problémů. (boost.ai, 2023)

V budoucnosti se očekává, že automatizace změní rozsah úkolů v pojišťovnictví. Například pracovníci na pozici likvidátorů pojistných událostí se mohou více soustředit na zlepšování zákaznických zkušeností, zatímco podnikoví aktuáři mohou pracovat více s datovou vědou a pokročilou analytikou. Strategie talentu v pojišťovnictví se bude muset přizpůsobit, kde bude důležité přeškolování a rozvoj stávajících zaměstnanců. (mckinsey.com, 2020)

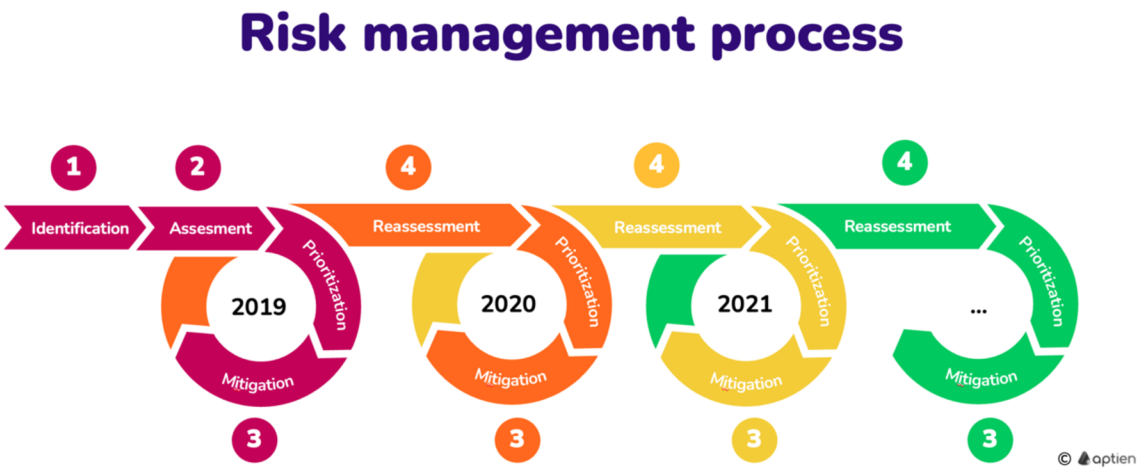
Automatizované zpracování dokumentů v pojišťovnictví (IDP) výrazně zlepšuje efektivitu a přesnost při manipulaci s daty. IDP umožňuje extrakci dat z různých zdrojů, což zlepšuje přesnost a efektivitu dat. Technologie také usnadňuje porovnání dokumentů s vysokou přesností, což je nezbytné pro procesy, jako je ověřování politik proti externím datovým zdrojům. Automatizace těchto procesů vede ke zvýšení spokojenosti zákazníků a

věrnosti, zatímco zároveň poskytuje úspory nákladů a zlepšení efektivity pro pojišťovací společnost. (docketry.com, 2022)

3.1.3 Služby zákazníkům

V oblasti služeb zákazníkům v pojišťovnictví hrají informační systémy stále důležitější roli. Moderní pojišťovny využívají AI a strojové učení pro lepší porozumění

Obrázek 2 - Proces řízení rizik



Zdroj: aptien.com, 2024

potřebám svých klientů, což jim umožňuje nabízet personalizované pojišťovací produkty. Inteligentní avatary na sociálních sítích nebo webových stránkách jsou schopny interaktivně komunikovat s potenciálními zákazníky, což vede k vytváření přizpůsobených pojistných plánů, které mohou zvýšit úspěšnost prodejních snah (InsurTech Magazine).

Pojišťovny, které modernizovaly své IT systémy, dosahují podstatně vyšší produktivity operací a nižších IT nákladů na politiku ve srovnání s těmi, které stále využívají starší systémy. Moderní IT systémy také umožňují lepší automatizaci a přesnější zpracování pojistných událostí, například spojením systémů politik a nároků pro lepší shodu pojistných klauzulí a událostí s nároky (McKinsey).

3.1.4 Hodnocení rizik a podpora rozhodování

AI a ML technologie v pojišťovnictví také zlepšují efektivitu procesů pojistných nároků analýzou vzorců chování zákazníků a detekcí podezřelých akcí a potenciálně podvodných nároků, což může vést k lepším cenám pro zákazníky (InsurTech Magazine).

V oblasti rychlého řízení nároků, pojišťovna Lemonade navrhla systém s umělou inteligencí, který dokáže vyřešit nárok během několika sekund. Tento systém využívá algoritmy, které "chápou" povahu nároků, jejich závažnost a zda je uživatel v nouzovém stavu, a dokonce se snaží posoudit pravděpodobnost podvodného nároku (TJIP).

Kromě výše uvedeného, IT modernizace v pojišťovnictví také zahrnuje snižování nákladů na úpravu škod prostřednictvím automatizace a zvýšení přesnosti zpracování nároků, například propojením systémů politik a nároků pro lepší shodu politik a událostí s nároky (McKinsey).

Některé pojišťovny dokonce investují do robo-poradců, kteří poskytují pohodlný přístup k informacím 24/7 a mohou poskytovat přístup k službám správy majetku, které byly dříve vyhrazeny pouze pro velmi bohaté, během několika minut (TJIP).

3.1.5 Odpovídající regulace a dohled

Pokud jde o správu politik a podporu rozhodování, AI usnadňuje administrativní úkoly, jako je správa politik, výpočet předplatného a obnovení politik, a navíc zahrnuje prvky hraní her, které povzbuzují držitele politik k adopci zdravějšího životního stylu, čímž činí správu politik interaktivnější a atraktivnější (InsurTech Magazine).

Strategické partnerství a společné podniky jsou dalším způsobem, jak pojišťovny rozšiřují své stávající schopnosti a splňují očekávání spotřebitelů. Taková partnerství mohou být klíčem k růstu a úspěchu zákazníků (TJIP).

Celkově jsou informační systémy a moderní technologie, jako je AI a ML, klíčové pro poskytování vysoce personalizovaných služeb zákazníkům, zlepšení zkušeností zákazníků, rychlého zpracování nároků a efektivní správy politik v pojišťovnictví.

3.1.6 Inovace a konkurenceschopnost

V konkurenčním prostředí pojišťovacího odvětví jsou inovace klíčovým faktorem pro udržení a zvyšování konkurenceschopnosti. Informační systémy (IS) umožňují pojišťovnám inovovat v různých směrech:

1. Vývoj nových pojistných produktů a služeb: Moderní IS umožňují pojišťovnám rychle reagovat na změny trhu a vytvářet nové, flexibilní produkty, které lépe odpovídají potřebám klientů. Analýza dat a modelovací techniky pomáhají lépe porozumět tržním potřebám a identifikovat příležitosti pro inovace. (McKinsey, 2019)
2. Zlepšení zákaznického zážitku: IS umožňují pojišťovnám poskytovat lepší a personalizovaný servis klientům. Portály pro klienty, chatboti a další technologie zvyšují dostupnost a rychlost komunikace se zákazníky, což vede k vyšší spokojenosti a loajalitě klientů. (McKinsey, 2021)
3. Zvýšení produktivity operací a snížení nákladů: Modernizací IT mohou pojišťovny výrazně zvýšit produktivitu operací a snížit IT náklady. Například pojišťovny s modernizovanými IT systémy mohou dosahovat o 40 procent vyššího počtu politik na plný pracovní úvazek a mohou mít IT náklady na politiku o 41 procent nižší než ty s tradičními IT systémy. (McKinsey, 2019)
4. Přizpůsobení se digitálním trendům: Pojišťovny musí své provozní modely přizpůsobit modernizaci jádrových IT systémů. To vyžaduje integrovaný transformační přístup, který zjednodušuje jádrové systémy a může vést k měřitelné efektivitě, spokojenosti zákazníků a udržitelným zlepšením. (McKinsey, 2019)
5. Využití technologických trendů: Přijetí nových technologických trendů, jako je aplikovaná AI, distribuovaná infrastruktura a budoucnost konektivity, transformuje způsob, jakým pojišťovny fungují, nabízejí nové úrovně služeb, personalizace produktů a řízení rizik. Tyto trendy umožňují pojišťovnám přecházet k prediktivnějším modelům, kde se zákaznická data používají k informování různých interakcí s pojistníky. (McKinsey, 2021)
6. Podpora inovací napříč organizací: Pro udržitelný růst se doporučuje přesunout zdroje z běžných podnikových úkolů na průlomové inovační iniciativy, vyvíjet

rozdílné cesty vývoje produktů a vytvářet hodnotové nabídky, které zahrnují nové přístupy k zapojení zákazníků a distribuci. (McKinsey, 2022)

Shrnuto, IS jsou hnací silou inovací v pojišťovacím odvětví a klíčem k udržení konkurenceschopnosti. Tyto systémy umožňují rychle reagovat na tržní změny, nabízet moderní služby klientům a zlepšovat interní procesy, což má pozitivní vliv na výsledky a budoucnost pojišťovací společnosti. (McKinsey, 2022)

3.2 Softwarové inženýrství

Vývoj informačních systémů (IS) v pojišťovnictví je momentálně ovlivňován několika významnými technologickými trendy. Umělá inteligence (AI) a strojové učení (ML) se stávají klíčovými technologiemi, které zásadně přetvářejí základní procesy v pojišťovnictví, jako je distribuce, underwriting, nároky a servis, a to tím, že umožňují předvídatelnější a kvalitnější zákaznické kontakty. Přestože mnoho pojišťoven experimentuje s AI, jen málo z nich dosud tyto schopnosti skutečně využívá napříč celým podnikem. Využití cloudových řešení a distribuované infrastruktury se zvyšuje, což umožňuje pojišťovnám rychlejší uvedení nových produktů a vytváření lepšího zákaznického servisu, a také podporuje zpracování obrovských datových sad, které jsou pro moderní pojišťovnictví nezbytné. (Barlow M., 2016)

IoT a pokročilé možnosti připojení, jako je 5G, umožňují sdílení dat v reálném čase a pomáhají pojišťovnám poskytovat služby v reálném čase. Přechod k IoT a používání telematiky umožňuje pojišťovnám posunout se k modelům založeným na využití, které poskytují přesnější hodnocení rizik a cenové modelování založené na skutečném chování zákazníků. Dále, důležité bezpečnostní standardy, jako jsou blockchain a zero-trust security, pomáhají vytvářet odolné sítě, které chrání před kybernetickými útoky. A konečně, nízko kódové/no-kódové platformy nabízejí pojišťovnám větší agilitu a růst díky snadnému spravování pojišťovacích platform, rychlému nasazení aktualizací a vývoji nových produktů. (adacta-fintech.com, 2024)

3.2.1 Databázové technologie

Databázové technologie jsou nezbytné pro sběr, správu a analýzu dat, které řídí rozhodovací procesy a zvyšují efektivitu a ziskovost organizací. Relační databáze

(RDBMS) dominují od 80. let a vyznačují se efektivním přístupem ke strukturovaným informacím prostřednictvím tabulek a řádků. Objektově orientované databáze pak informace prezentují ve formě objektů, podobně jako v objektově orientovaném programování. (businesstechweekly.com, 2022)

Distribuované databáze obsahují soubory umístěné na různých místech, což umožňuje ukládání dat na více počítačích v různých fyzických lokalitách nebo po celých sítích. Data warehousy (datové sklady) jsou speciálně navrženy pro rychlé dotazování a analýzu a slouží jako centrální úložiště pro data. NoSQL databáze umožňují ukládání a manipulaci s nestrukturovanými a polostrukturovanými daty, což je užitečné pro webové aplikace, které se stávají stále složitějšími. (oracle.com, 2023)

Kromě toho moderní databázové systémy, jako jsou cloudové databáze a multimodelové databáze, kombinují různé typy modelů databází do jediného, integrovaného backendu, čímž umožňují zpracování různých typů dat. Dokumentové databáze jsou moderním způsobem ukládání dat ve formátu JSON namísto tradičních řádků a sloupců. (oracle.com, 2023)

Samostatnou kapitolou jsou autonomní databáze, které využívají cloudovou technologii a strojové učení k automatizaci mnoha rutinních úkolů potřebných pro správu databází, jako je ladění, zabezpečení, zálohování, aktualizace a další běžné správní úkoly. Tyto autonomní databáze přinášejí výhody jako zvýšení výkonu, snížení nákladů a zlepšení zabezpečení dat. (oracle.com, 2023)

Výběr správného typu databáze pro organizaci závisí na tom, jak organizace plánuje data využívat. RDBMS mohou poskytovat centrální systémy pro efektivní správu všech podnikově kritických dat. Na druhou stranu, NoSQL databáze mohou být vhodné pro organizace, které potřebují pracovat s velkými objemy nestrukturovaných dat a vyžadují flexibilitu a škálovatelnost. Pro organizace je také klíčové zvážit, jak budou data sdílena a získávána, jaký mají rozpočet a zda se jejich požadavky pravděpodobně budou měnit nebo růst. (businesstechweekly.com, 2022)

3.2.2 Cloud computing a virtualizace

V oblasti cloud computingu a virtualizace dochází v současnosti k významným inovacím, které ovlivňují různé průmyslové odvětví včetně finančních služeb, výroby, technologií a zdravotnictví. Cloud computing nabízí podnikům flexibilní a škálovatelné IT

zdroje bez potřeby vlastní infrastruktury, což je obzvláště významné pro odvětví, kde je třeba reagovat na rychlé změny a vysoké nároky na datové zpracování. Služby jako Amazon Web Services (AWS) a Microsoft Azure nabízejí širokou škálu cloudových řešení, včetně veřejných, soukromých a hybridních cloudů, které umožňují podnikům efektivně spravovat své aplikace a služby. (pluralsight.com, 2023)

Virtualizace je klíčovou součástí cloudových služeb a zahrnuje vytváření virtuálních verzí fyzických výpočetních zdrojů, jako jsou servery a sítě. Tato technologie umožňuje podnikům efektivně využívat své hardwarové zdroje tím, že na jednom fyzickém stroji běží několik virtuálních instancí. Tím se snižují náklady a zvyšuje se efektivita správy serverů a aplikací. Pro malé a střední podniky (SMB) je virtualizace výhodná, protože snižuje kapitálové výdaje a provozní náklady, minimalizuje dobu nefunkčnosti a zvyšuje produktivitu, což vede k rychlejšímu poskytování zdrojů a aplikací (cyberlinkasp.com, 2023)

Cloudové technologie a virtualizace představují klíčové komponenty pro moderní informační systémy, které umožňují rychlou škálovatelnost, flexibilitu a zlepšení kontinuity a odolnosti IT služeb. Tyto inovace jsou stále důležitější, jak organizace hledají způsoby, jak se adaptovat na ekonomické tlaky a měnící se požadavky trhu. (pluralsight.com, 2023)

3.2.3 Front end technologie

Kapitola 3.2.4 se věnuje front-endovým technologiím, které jsou klíčové pro vývoj softwarových aplikací, se zaměřením na uživatelské rozhraní a interakci s uživatelem. Tato oblast je neustále v pohybu, s novými technologiemi a trendy, které mění způsob, jakým jsou webové stránky a aplikace vytvářeny.

Uživatelské rozhraní je často první kontaktní bod uživatele s aplikací, což z něj činí klíčový prvek pro uživatelskou spokojenost a úspěšnost aplikace. Kvalita a intuitivnost uživatelského rozhraní mají zásadní dopad na celkový zážitek uživatele. (Autor, 2024)

Mezi hlavní technologie používané ve front-endu patří HTML, CSS a JavaScript. Tyto technologie jsou doplněny o frameworky a knihovny, jako jsou React, Angular a Vue.js, které umožňují vytvářet dynamické a interaktivní webové aplikace. Tyto frameworky jsou stále populární a vývojáři by si měli osvojit jejich používání pro budoucí projekty. (fronttribe.com, 2023)

Trendy ve front-end vývoji v roce 2024 zahrnují mobilní design jako prioritu, vzhledem k rostoucímu počtu uživatelů přistupujících k internetu prostřednictvím mobilních zařízení. Dále je důležitý responsivní web design, dark mode, microinteractions a motion design. Vývojáři by měli také věnovat pozornost vývoji designových systémů pro udržení konzistence v designu a vývoji. (romexsoft.com, 2023)

V roce 2023 a dále se očekává, že budou v front-end vývoji populární technologie jako WebAssembly pro výkonnostně efektivní vykonávání kódu v prohlížeči a GraphQL pro efektivní komunikaci s backendovými službami. Serverless computing je další trend, který umožňuje vývojářům se soustředit na psaní kódu bez nutnosti spravovat servery. (romexsoft.com, 2023)

Front-end vývojáři hrají klíčovou roli v celém procesu vývoje, kde spolupracují s UI/UX designéry a back-end vývojáři. Jejich úkolem je implementovat uživatelské rozhraní podle designových specifikací a zajistit jeho přístupnost a optimalizaci výkonu. (Autor, 2024)

Front-end vývoj bude nadále hrát klíčovou roli v softwarovém inženýrství. Vývojáři by měli neustále sledovat nové technologie a trendy, jako je AI, virtuální a rozšířená realita, a přizpůsobovat své dovednosti, aby zůstali konkurenceschopní na trhu práce. (fronttribe.com, 2023)

3.2.4 Backend Technologie

Backend technologie představují klíčový aspekt v procesu vývoje softwarových aplikací, který se zaměřuje na implementaci serverové části aplikace, manipulaci s daty a zajištění komunikace mezi klienty a serverem. V této kapitole se podrobněji zabýváme technologiemi, frameworky a metodologiemi používanými při vývoji backendu v rámci softwarového inženýrství. (opensudo.org, 2023)

Backend technologie hrají klíčovou roli v celkovém fungování softwarové aplikace. Jsou zodpovědné za zpracování požadavků klientů, provádění obchodní logiky, správu a manipulaci s daty v databázích a zajištění bezpečnosti a výkonnosti aplikace. Kvalitní a efektivní backend je základním předpokladem pro stabilitu, škálovatelnost a úspěšnost aplikace. (medium.com, 2022)

Pro vývoj backendu existuje mnoho různých technologií, programovacích jazyků, frameworků a databázových systémů. Mezi nejpoužívanější patří:

Programovací jazyky:

- Java: Objektově orientovaný programovací jazyk s širokou škálou frameworků pro vývoj backendových aplikací, jako je například Spring nebo Hibernate.
- Python: Interpretovaný programovací jazyk s jednoduchou syntaxí a silnými knihovnami pro vývoj webových aplikací, jako je například Django nebo Flask.
- JavaScript/Node.js: Skriptovací jazyk běžící na straně serveru, který umožňuje vývoj asynchronních a škálovatelných aplikací pomocí frameworků jako je Express nebo NestJS.
- C#: Moderní objektově orientovaný jazyk vyvinutý společností Microsoft, který je často používán ve spojení s frameworkem .NET pro vývoj aplikací pro platformu Windows.

Frameworky:

- Spring Boot: Framework pro vývoj aplikací v jazyce Java, který zajišťuje rychlý a efektivní vývoj backendových aplikací s minimální konfigurací.
- Django: High-level framework pro vývoj webových aplikací v jazyce Python, který poskytuje kompletní sadu nástrojů pro vytváření robustních aplikací.
- Express.js: Minimální a flexibilní framework pro vývoj webových aplikací v jazyce JavaScript, který je často používán ve spojení s platformou Node.js.
- ASP.NET Core: Framework pro vývoj webových aplikací v jazyce C#, který poskytuje škálovatelný a výkonný model pro vytváření moderních backendových aplikací.

Stejně jako v u front-end vývoje se pro efektivní organizaci a řízení vývoje backendu se často používají různé metodologie, které pomáhají zajistit kvalitu, stabilitu a předvídatelnost procesu vývoje. Mezi nejpoužívanější metodologie ve vývoji patří:

- Agilní metodiky a DevOPS: Stejně jako u front-endového vývoje jsou oblíbené tyto přístupy, které jsou často uplatňovány na celé vývojové týmy včetně front-end i back-end týmů. (medium.com, 2023)

- Test Driven Development (TDD): Metodologie, která klade důraz na psaní testů před implementací kódu, což umožňuje zvýšit kvalitu kódu a snížit počet chyb. (Autor, 2024)

Obrázek 3 - Agilní vývoj

Zdroj: indevlab.com, 2021

Predikce budoucnosti backendového vývoje ukazují na další rozvoj technologií jako jsou mikro služby, serverless architektury a pokročilá integrace s IoT zařízeními. Tyto technologie budou formovat novou generaci backendových systémů, které budou ještě škálovatelnější, výkonnější a bezpečnější. (opensudo.org, 2023)

Každý z těchto bodů by byl rozvinut do hlubší analýzy a diskuse, přičemž by byly přidány podrobné informace, příklady, případové studie a odkazy na zdroje a výzkumy. Pro tuto kapitolu by bylo vhodné konzultovat akademické články, oficiální dokumentaci technologií a názory předních expertů v oboru, aby byla zajištěna její odbornost a relevanci. (opensudo.org, 2023)

C# a framework .NET

C# je moderní, objektově orientovaný jazyk, který se vyvinul jako klíčový prvek platformy .NET od Microsoftu. Díky své výkonnosti a integraci s různými nástroji a knihovnami je C# preferovanou volbou pro mnohé backendové aplikace, zejména ty, které jsou určené pro prostředí Windows. .NET framework je oslavován pro jeho schopnost

běhu na různých platformách, včetně Windows, macOS a Linuxu, což přináší vývojářům výhody jako zlepšený výkon a možnost paralelního nasazení různých verzí. (Microsoft.com, 2023)

ASP.NET Core, součást rodiny .NET, je příkladem cross-platformního, vysoko-výkonného frameworku určeného pro moderní, cloudem podporované, internetově propojené aplikace. Umožňuje snadnou integraci s populárními front end JavaScript frameworky jako jsou Angular, React, Vue a poskytuje rozsáhlou podporu pro vývoj webových API a služeb. Vývojáři si mohou vybrat, zda chtějí své aplikace nasadit na cloud nebo on-premise, což dává flexibilitu v rozhodování o architektuře a nasazení. (Miles, R., 2018)

.NET je také uznáván pro jeho produktivitu, bezpečnost, spolehlivost a schopnost automatické správy paměti pomocí garbage collectoru. Poskytuje rozsáhlou sadu knihoven, které mají širokou funkcionalitu a jsou optimalizovány pro výkon na různých operačních systémech a architekturách čipů. (Microsoft.com, 2024)

V rámci .NET ekosystému, Microsoft a komunita pravidelně aktualizují platformu, aby zajistili, že uživatelé nasazují do provozu bezpečné a spolehlivé aplikace. .NET je udržován mnoha organizacemi, které pracují na tom, aby .NET mohl běžet na různých operačních systémech a byl neustále aktualizován. Nové verze .NET jsou vydávány každoročně v listopadu a jsou pravidelně aktualizovány každý měsíc na Patch Tuesday. (Microsoft.com, 2024)

Vzhledem k těmto aspektům je C# spolu s .NET frameworkem silnou volbou pro backend vývoj aplikací, které vyžadují spolehlivost, škálovatelnost a výkonnost. Ať už se jedná o cloudové aplikace, interaktivní webové služby nebo rozsáhlé podnikové systémy, C# a .NET poskytují nástroje a schopnosti k vytvoření robustních řešení, která mohou růst společně s vaším podnikáním a adaptovat se na měnící se požadavky trhu. (Microsoft.com, 2024)

Obrázek 4 - Příklad Struktury C# a frameworku .NET ve backend vývoji

```
using System;

class Program
{
    static void Main()
    {
        Console.WriteLine("Hello, World!");
    }
}
```

Zdroj: Autor, 2024

3.3 Jazyk UML: Unifikovaný Modelovací Jazyk

Unifikovaný modelovací jazyk (UML) je standardizovaný vizuální modelovací jazyk, který pomáhá softwarovým vývojářům vizualizovat a konstruovat nové systémy. Nejedná se o programovací jazyk, ale o sadu pravidel určených speciálně pro kreslení diagramů, podobně jako je modrá plán pro softwarové inženýry. UML byl vyvinut v 90. letech 20. století Grady Boochem, Ivarem Jacobsonem a Jamesem Rumbaughem v společnosti Rational Software a později přijat jako standard skupinou Object Management Group. UML prošel několika revizemi, přičemž UML 2.0 nahradilo verzi 1.5 v roce 2005, a další aktualizace pokračovaly až do 10. let 21. století. (gliffy.com, 2020)

Diagramy UML se dělí na dva hlavní typy: strukturální a behaviorální. Strukturální diagramy, jako jsou třídní diagramy, objektové diagramy a balíčkové diagramy, popisují statickou strukturu systému. Behaviorální diagramy, včetně sekvenčních diagramů a diagramů případů užití, popisují dynamiku a chování systému. Každý typ diagramu má své specifické notace a účely, od zobrazování architektury a závislostí softwarových komponent po modelování fyzických aspektů softwarových systémů. (Booch, 2005)

UML je důležitý pro vývoj softwaru, protože poskytuje jasný a standardizovaný způsob reprezentace architektury a designu softwarových systémů. Je užitečný nejen v dokumentaci objektově orientovaného designu, ale také v širším souboru designové dokumentace. I když UML samo o sobě není vývojovou metodou, bylo navrženo tak, aby bylo kompatibilní s předními metodami objektově orientovaného vývoje softwaru té doby,

jako je například Boochova metoda a Racionalizovaný jednotný proces (RUP). (staruml.io, 2023)

Je důležité rozlišovat mezi modelem UML a sadou diagramů systému. Model představuje úplný pohled na systém, zatímco diagram je jeho částečná grafická reprezentace. Sada diagramů nemusí nutně pokrývat celý model a smazání diagramu nemění model. (omg.org, 2017)

Souhrnně lze říci, že UML nabízí komplexní rámec pro modelování softwarových systémů a poskytuje softwarovým vývojářům nezbytné nástroje pro vizualizaci, dokumentaci a konstrukci složitých softwarových architektur a procesů. (visual-paradigm.com, 2020)

3.3.1 Historie UML

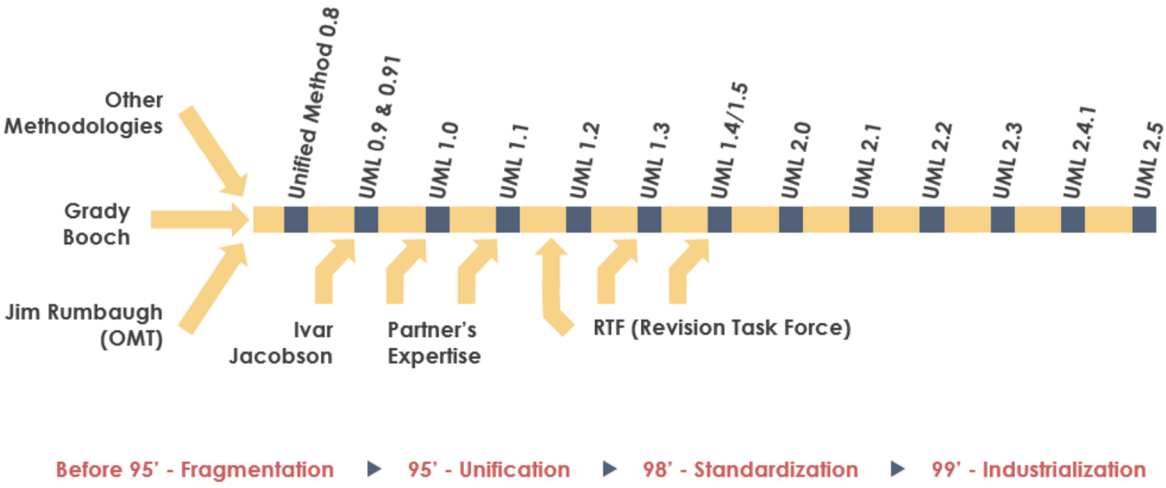
Unifikovaný modelovací jazyk, známý pod zkratkou UML, se vyvíjel jako odpověď na potřebu sjednocení a standardizace v modelování softwaru. Tato potřeba se stala zjevnou v průběhu 80. a 90. let, kdy se softwarový průmysl začal rychle rozvíjet a byla potřeba lépe porozumět složitým systémům a usnadnit komunikaci mezi různými týmy.

V roce 1994 Grady Booch, James Rumbaugh a Ivar Jacobson, kteří pracovali pro společnost Rational Software, začali na společném projektu, jehož cílem bylo sloučení jejich individuálních přístupů k modelování softwaru do jediného, univerzálního modelovacího jazyka. Tito tři průkopníci přinesli do projektu každý svou odbornost a zkušenosti, které získali při vyvíjení svých vlastních metod: Boochova metoda, Object Modeling Technique (OMT) od Rumbaugh a Object-Oriented Software Engineering (OOSE) od Jacobsona. (visual-paradigm-com, 2020)

Výsledkem jejich spolupráce bylo první vydání UML v roce 1996, které kombinovalo nejlepší prvky jejich předchozích prací. Tato verze byla představena jako otevřený standard, což znamenalo, že byl k dispozici všem vývojářům a organizacím bez nutnosti placení licenčních poplatků. Tento otevřený přístup velmi pomohl k rychlému rozšíření UML. (visual-paradigm-com, 2020)

V roce 1997 byl UML oficiálně přijat jako standard organizací Object Management Group (OMG), která je mezinárodní konsorciem vedoucím vývoj a publikaci standardů v oblasti softwarového inženýrství. OMG od té doby převzala zodpovědnost za udržování a aktualizaci UML, což zahrnovalo vydání UML 2.0 v roce 2005, které přineslo významné změny a rozšíření. (visual-paradigm-com, 2020)

Obrázek 5 - Časová osa Historie UML



Zdroj: visual-paradigm-com, 2020

3.3.2 Vývoj UML

Vývoj Unifikovaného modelovacího jazyka (UML) byl významným krokem ve vývoji softwarového inženýrství, jehož cílem bylo poskytnout standardizovaný způsob vizualizace a konstrukce softwarových systémů. UML, vyvinutý v 90. letech týmem v Rational Software a později přijatý jako standard skupinou Object Management Group (OMG), umožňuje vývojářům popisovat strukturu, chování, interakce a procesy systému na různých úrovních abstrakce. (Autor, 2024)

Sekvenční a stavové diagramy, které jsou součástí UML, poskytují nástroje k detailnímu popisu interakcí mezi objekty a vizualizaci jejich životního cyklu. Tato vlastnost UML je významná pro pochopení a předvídání chování systému ve vývoji. (omg.org, 2014)

Integrace UML do Model-Driven Architecture (MDA) OMG představovala další významný krok ve vývoji UML. MDA je přístup navržený OMG, který zdůrazňuje modelování jako klíčový prvek softwarového vývoje, a UML se stalo jeho zásadní součástí. To znamená, že pomocí MDA mohou být modely vytvořené v UML realizovány na téměř jakékoliv platformě, včetně .NET, což umožňuje odstranění vazby mezi obchodní a technickou logikou a nezávislý vývoj každého aspektu systému. (omg.org, 2014)

Vývoj UML byl rovněž ovlivněn potřebou větší flexibility a lepší škálovatelnosti, s neustále se zvyšujícím důrazem na formalizaci jazyka. MDA poskytuje směrnice pro strukturování softwarových specifikací, které jsou vyjádřeny jako modely, což umožňuje oddělení podnikové a aplikační logiky od technologie platformy. Díky tomu mohou být modely realizovány na prakticky jakékoliv platformě a dokumentují obchodní funkčnost a chování aplikace odděleně od technologicky specifického kódu, což izoluje jádro aplikace od neustálého cyklu technologických změn. (omg.org, 2014)

Podrobnosti o MDA a jeho integraci s UML lze nalézt na oficiálních stránkách OMG, kde se také diskutuje o přínosech a výzvách spojených s přijetím tohoto přístupu. Tato platforma také zahrnuje informace o dalších nástrojích a specifikacích spojených s MDA a UML, jako je MetaObject Facility (MOF) a XML Metadata Interchange (XMI). (omg.org, 2014)

3.3.3 Základní struktura UML

Základní struktura UML zahrnuje tři hlavní typy prvků: předměty, relace a diagramy (Fowler, 2009).

Předměty představují základní stavební bloky modelu a zahrnují třídy, rozhraní, případy užití a další. Každý předmět má své chování, které se vyjadřuje slovesy, a může být doplněn poznámkami nebo seskupením do balíčků (Ambler, 2005).

Relace definují vztahy mezi jednotlivými objekty v modelu a mohou být závislosti, asociace, generalizace nebo realizace. Každý typ relace má svou specifickou grafickou reprezentaci a význam (Arlow, 2007).

Diagramy jsou grafickým zobrazením prvků a vztahů v modelu a umožňují různé pohledy na systém. Existují různé typy diagramů, které lze rozdělit do tří hlavních skupin: diagramy struktury, diagramy chování a diagramy interakce (Arlow, 2007).

3.3.4 Obecné mechanismy

UML poskytuje čtyři základní mechanismy pro modelování: specifikace, ornamenty, podskupiny a mechanismy rozšiřitelnosti.

a) Specifikace: Specifikace jsou textovými popisy prvků diagramu a poskytují základní informace potřebné k porozumění modelu (Arlow, 2007).

b) Ornamenty: Ornamenty jsou symboly používané k vyjádření vlastností prvků a umožňují zjednodušení komplexních struktur (Ambler, 2005).

c) Podskupiny: Podskupiny umožňují různé pohledy na řešenou problematiku a rozlišují se podle typu klasifikátorů a instance nebo rozhraní a implementace (Arlow, 2007).

d) Mechanismy rozšiřitelnosti: Mechanismy rozšiřitelnosti poskytují možnosti rozšíření UML modelů a zahrnují omezení, stereotypy a označené hodnoty (Arlow, 2007).

3.3.5 Architektura

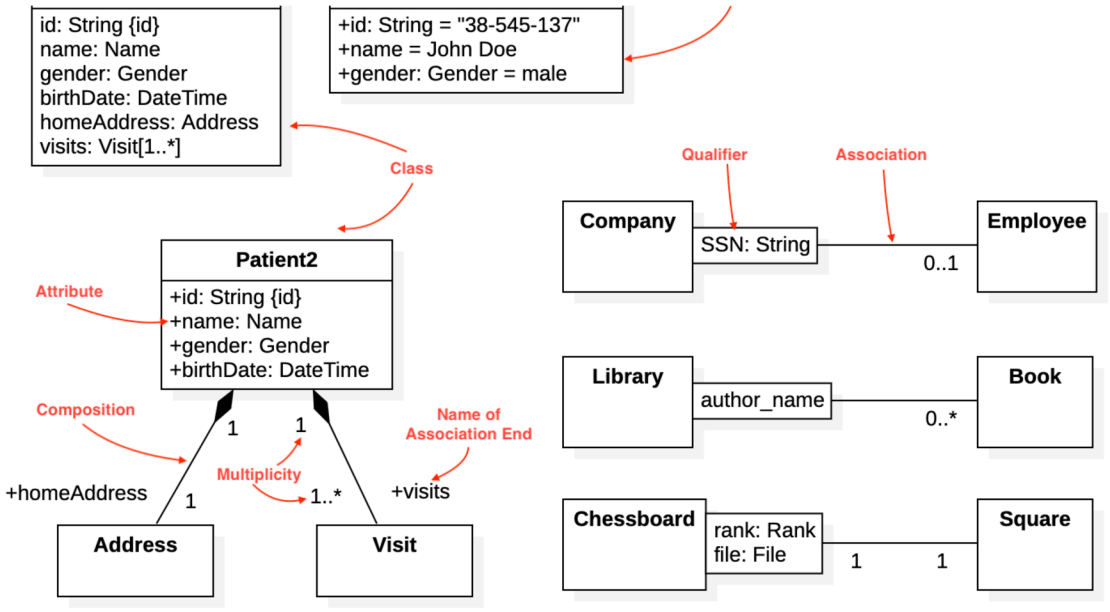
Architektury umožňují zachytit podstatné aspekty vývojově komplexních systémů pomocí různých pohledů a kombinací diagramů, čímž vzniká komplexní architektura systému. Architekturu lze chápat jako kombinaci různých pohledů na systém (Arlow, 2007).

UML poskytuje širokou škálu nástrojů a technik pro modelování softwarových systémů, které mohou být využity v různých fázích vývoje od analýzy až po implementaci a údržbu. Tyto techniky a nástroje jsou důležité pro efektivní a systematické řešení problémů v oblasti softwarového inženýrství (Fowler, 2009).

3.3.6 Modelování v UML

UML poskytuje strukturovaný přístup k modelování softwarových systémů, který umožňuje vytvářet abstraktní reprezentace systémů, analyzovat jejich strukturu a chování a komunikovat s různými stakeholdery vývoje. Modelování v UML se často provádí pomocí různých typů diagramů, které umožňují vizualizaci různých aspektů systému z různých úhlů pohledu.

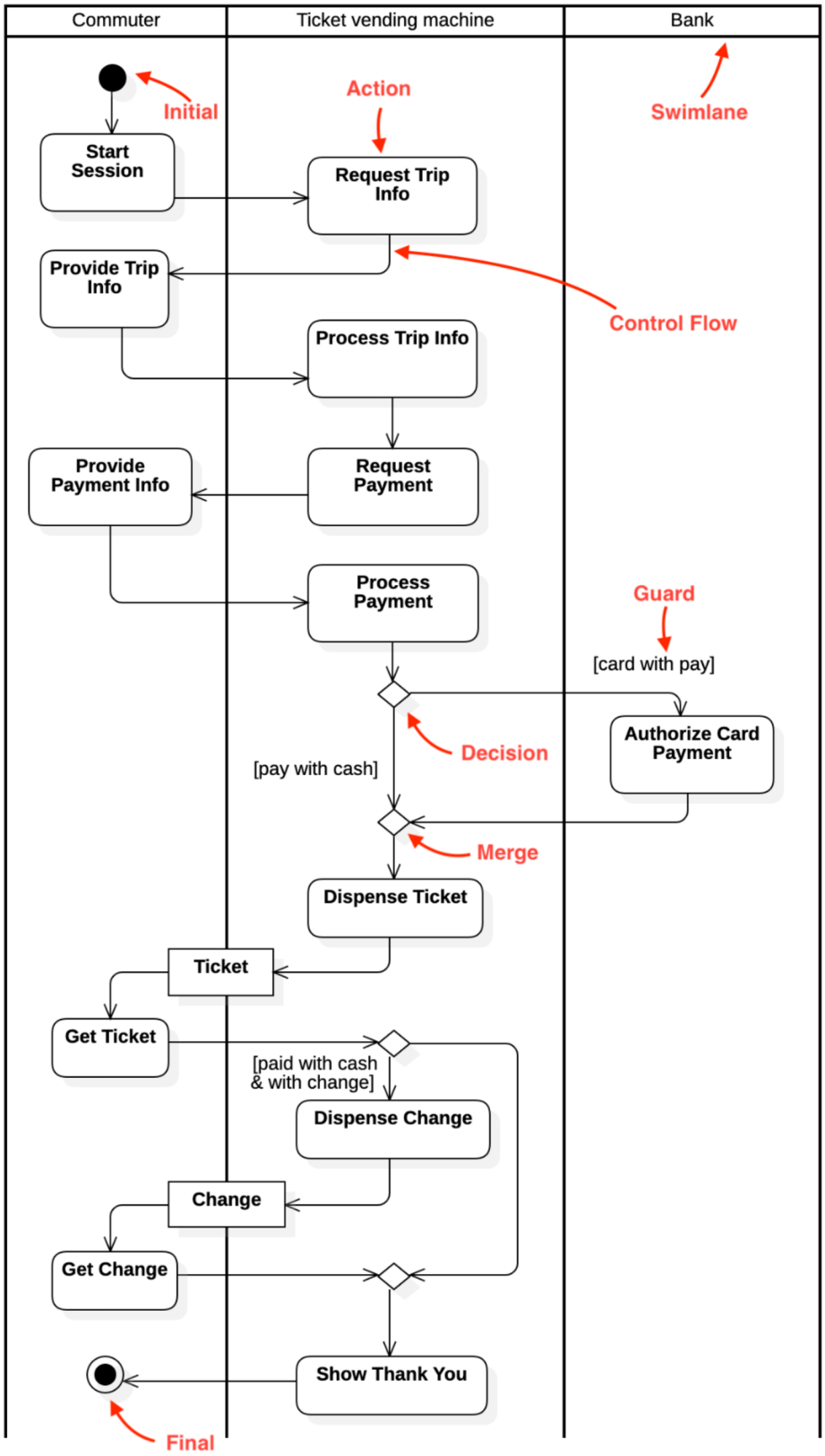
Obrázek 6 - Příklad UML



Zdroj: medium.com, 2022

Diagramy struktury se zaměřují na statické aspekty systému a zachycují jeho stavbu a organizaci. Mezi nejčastěji používané diagramy struktury patří diagram tříd, objektový diagram, diagram balíčků, diagram komponent, diagram nasazení a diagram složené struktury. Tyto diagramy umožňují vizualizaci tříd, objektů, balíčků, komponent, jejich vztahů a fyzického nasazení systému (Arlow, 2007).

Obrázek 7 - Příklad activity diagramu



Zdroj: quizizz.com, 2022

Diagramy chování se zaměřují na dynamické aspekty systému a zachycují jeho chování a interakce mezi jednotlivými komponentami. Mezi diagramy chování patří diagram aktivit, diagram případů užití a diagram stavového automatu. Tyto diagramy umožňují modelovat chování systému v různých situacích a scénářích (Arlow, 2007).

Diagramy interakce se zaměřují na komunikaci a interakce mezi jednotlivými objekty nebo komponentami systému. Mezi diagramy interakce patří diagram komunikace, diagram časování, sekvenční diagram a stručný diagram interakce. Tyto diagramy umožňují vizualizaci zpráv a událostí posílaných mezi jednotlivými objekty a komponentami systému v čase (Arlow, 2007).

3.3.7 Využití UML v praxi

UML nachází široké využití v praxi při vývoji softwarových systémů. Je používán při analýze a návrhu systémů, při dokumentaci systémů a komunikaci s různými stakeholdery vývoje, jako jsou zákazníci, vývojové týmy a manažeři projektů. UML umožňuje komplexní modelování systémů, což přispívá k lepšímu porozumění jejich struktury a chování a usnadňuje přechod od konceptuálního návrhu k implementaci (Fowler, 2009).

Díky standardizaci a rozsáhlému ekosystému nástrojů a metodik je UML široce používaným nástrojem v oblasti softwarového inženýrství. Využívá se jak při vývoji malých a středních softwarových aplikací, tak i při vývoji velkých a komplexních systémů, jako jsou informační systémy, softwarové platformy nebo distribuované aplikace (OMG, 2017).

3.3.8 Budoucnost UML

S rozvojem technologií a nových přístupů k vývoji softwarových systémů se mění i role a význam UML. Nástroje a metodiky založené na UML jsou stále důležité pro modelování a analýzu softwarových systémů, ale v dnešní době se stále častěji objevují nové přístupy a techniky, které doplňují nebo nahrazují tradiční modelovací nástroje. Mezi tyto nové přístupy patří například agilní metodiky vývoje, modelování doménově řízené architektury (DDD) nebo přístupy založené na automatizovaném testování a kontinuální integraci (CI/CD).

Nicméně, i přes tyto nové trendy je UML stále silným a užitečným nástrojem pro modelování a analýzu softwarových systémů a zůstává nedílnou součástí softwarového

inženýrství (Fowler, 2009). S vývojem a inovacemi v oblasti softwarového inženýrství bude pravděpodobně UML nadále evoluovat a přizpůsobovat se novým požadavkům a trendům v odvětví.

3.4 Objektově orientované programování

Objektově orientované programování (OOP) má své kořeny v raných 60. letech, kdy výzkum na MIT přinesl průlom v použití instancí a objektů. Jako první programovací jazyk využívající objekty je uznáván Simula 67, který byl navržen v Norsku Kristenem Nygaardem a Ole-Johanem Dahlem pro účely simulací. Simula 67 byla průkopníkem nejen v konceptu třídy, ale také v uvedení instancí třídy. Termín "objektově orientované programování" byl poprvé použit ve společnosti Xerox PARC ve vztahu k jazyku Smalltalk, který byl inspirován projektem Simula 67, ale byl navržen tak, aby byl dynamický, umožňoval změny, vytváření nebo mazání objektů, což bylo odlišné od tehdy běžných statických systémů. (Stojkovic, 2023)

Vývoj OOP ovlivnila potřeba adresovat otázky strukturovaných dat, což vedlo k rozvoji specializovaných programovacích jazyků. Tony Hoare ve své práci „Record Handling“ z roku 1966 představil komplexní přístup k správě datových struktur v jazycích pro všeobecné použití, což výrazně ovlivnilo další vývoj OOP, včetně představení konceptu "referenčního typu", který je v podstatě ukazatelem na konkrétní záznam v paměti. (Stojkovic, 2023)

Simula 67 poskytla inspiraci pro velké množství dalších programovacích jazyků a byla významná pro vývoj grafických uživatelských rozhraní a vývoj event-driven programování. V 80. letech se OOP stalo prominentním, přičemž klíčovou roli sehrál C++. V moderní době je OOP základem mnoha programovacích jazyků, od Fortranu a BASICu po Pascal, a výzkum v této oblasti pokračuje s cílem dále rozvíjet tento přístup. (exforsys.com, 2006)

Různé části v OOP provádějí akce na předmětech ze skutečného světa a vytvářejí skutečné interakce mezi lidmi a stroji. Strategie je výhodná pro kolaborativní vývoj, kdy jsou projekty rozděleny do skupin kvůli organizaci objektově orientovaného softwaru. Opětovné použití kódu, škálovatelnost a efektivita jsou další výhody OOP. (exforsys.com, 2006)

První fází OOP je shromáždit všechny objekty, se kterými chce programátor pracovat, a určit jejich vztahy, což je proces známý jako datové modelování. Data a funkce se kombinují a vytvářejí objekt z datové struktury. Programátoři mohou také navázat spojení mezi několika objekty. Objekty mohou například získávat vlastnosti od jiných objektů. Člověk je přímočará ilustrace předmětu. (apollotechnical.com, 2022)

Logicky by se dalo předpokládat, že člověk bude mít jméno. To by bylo považováno za vlastnictví dané osoby. Další věc, kterou můžete od někoho očekávat, je jeho schopnost dělat, jako je chůze nebo řízení. Někdo to může považovat za jednu z metod člověka. Objekty slouží jako rámec pro objektově orientovaný programovací kód. (apollotechnical.com, 2022)

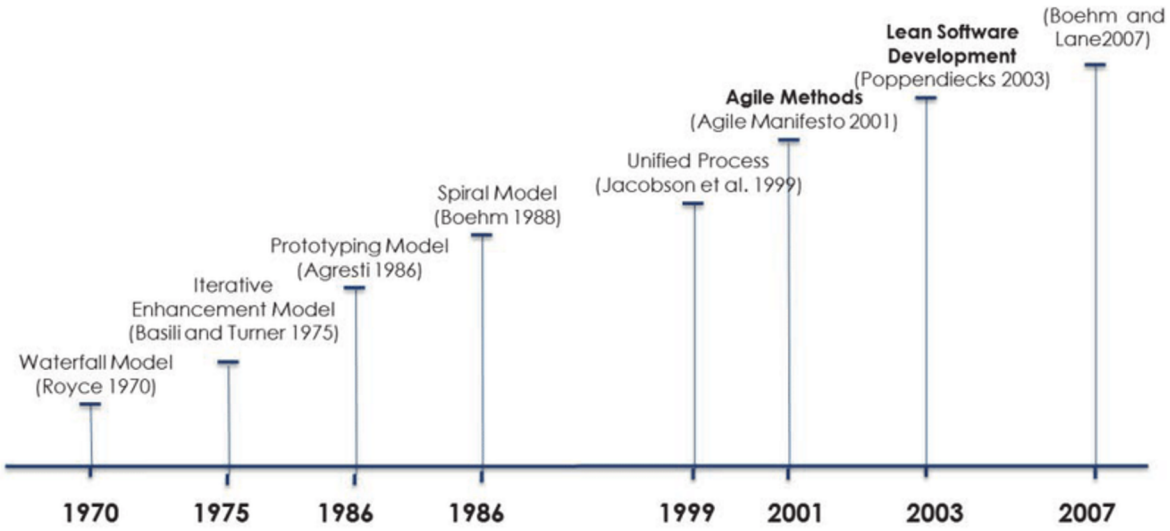
Jakmile jsou vaše objekty na svém místě, můžete použít jejich interakce k dosažení požadovaného výsledku. Zvažte možnost představení, kde někdo nasedne do auta a jede s ním z bodu A do bodu B. Začněte předměty jako osoba nebo vozidlo, jak byste je popsali. (apollotechnical.com, 2022)

Použití způsobů je jedním z příkladů: člověk může řídit auto a auto může řídit. Aby jednotlivec mohl řídit, musíte shromáždit své předměty tak, aby byly všechny na jednom místě. Když je objekt identifikován, je mu přiřazena třída objektů, která popisuje typ dat, která obsahuje, a sekvencuje logiku, která by mohla data jakýmkoli způsobem upravit. Metoda je jakákoli konkrétní logická sekvence. S jasně specifikovanými rozhraními známými jako zprávy mohou objekty komunikovat. (exforsys.com, 2006)

3.4.1 Vývoj OOP

Objektově orientované programování (OOP) představuje klíčový milník ve vývoji softwarového inženýrství. Tento přístup nevznikl náhodně, ale je výsledkem dlouhodobého vývoje v oblasti programování. OOP je dnes široce podporováno většinou moderních programovacích jazyků a přináší nový pohled na strukturu a funkce softwarových programů. Tento přístup by se měl uplatňovat ve všech typech projektů, od jednoduchých utilit až po složité databázové systémy. OOP není jen soubor technik nebo doporučených postupů; je to především nový způsob myšlení, nový pohled na řešení problémů a nová éra v softwarovém vývoji. (researchgate.net, 2022)

Obrázek 8 - Historie OOP



Zdroj: researchgate.net

3.4.2 Historie a vývoj programování

Vývoj programovacích paradigmat za posledních 40 let ukazuje, jak se výrazně změnil způsob, jakým píšeme software. Počáteční počítače byly omezené svým výkonem a s nimi i jejich software. S rychlým rozvojem hardwaru (například dle Moorova zákona, který předpovídá zdvojnásobení počtu tranzistorů v mikroprocesorech každé dva roky) se zvyšovala potřeba složitějšího a výkonnějšího softwaru. (researchgate.net, 2022)

Strojový kód

Toto rané paradigma zahrnovalo přímé programování v instrukcích specifických pro daný hardware, což bylo brzy nahrazeno flexibilnějšími přístupy. (Cocca, 2022)

Nestrukturované paradigma

Připomínající assembler, tento přístup zahrnoval lineární soubor instrukcí, který se vykonával sekvenčně. Přestože toto paradigma umožňovalo větší čitelnost a hardwarovou nezávislost, stále trpělo mnoha omezeními, například nadužíváním příkazu GOTO. (researchgate.net, 2022)

Strukturované programování

Toto paradigma zavedlo použití cyklů a větvení, což přineslo lepší organizaci a čitelnost kódu. Programy byly strukturovány do funkcí a metod, ale stále zde byla omezení v modifikaci a znovupoužití kódu. (Cocca, 2022)

Přechod k OOP

S narůstající složitostí a rozsahem softwarových projektů se objevila potřeba nového přístupu, což vedlo k rozvoji objektově orientovaného programování. OOP je inspirováno průmyslovou revolucí a ideou využívání standardizovaných komponent. Tento přístup se zaměřuje na znovu použitelnost, modularitu a zapouzdření funkcionalit do objektů. (Cocca, 2022)

3.4.3 Význam OOP

OOP přináší řadu výhod, včetně zvýšení přehlednosti kódu, usnadnění údržby a rozšíření softwaru, a podporuje znovu použitelnost kódu. Omyl, že OOP se má používat pouze pro složité systémy, je nesprávný; i jednoduché aplikace mohou těžit z principů OOP.

OOP představuje klíčový přístup v moderním softwarovém vývoji. Jeho principy a metodiky nejenže usnadňují práci programátorům, ale také přispívají k vytváření robustnějšího, flexibilnějšího a udržitelnějšího softwaru. (Half, 2023)

3.4.4 Klíčové koncepty OOP

Třída (Class)

Třída je základní jednotkou C++, která otevírá dveře k objektově orientovanému programování. Je to uživatelem definovaný datový typ, který lze přistupovat a používat vytvořením instance této třídy. Obsahuje své vlastní datové členy a členské funkce. Třída je srovnatelná s plánem objektu. V třídách se nacházejí jak členské funkce, tak datové členy. Datové členy uvnitř třídy jsou manipulovány pomocí těchto členských funkcí. (codeacademy.com, 2023)

Objekt (Object)

V okamžiku vytvoření třídy je definicí první objekt. Instance třídy existuje v objektu. Zajímavé je, že systém nealokuje žádný paměťový prostor, když je třída specifikována, ale je alokován, když je instancována, tj. když je objekt vytvořen. Věci v reálném světě mají společné vlastnosti stav a chování. Objekt své chování skrývá prostřednictvím metod a uchovává své informace v atributech. (Half, 2023)

Syntaxe (Syntax)

Principy, které specifikují, jak je jazyk strukturován, jsou známé jako syntax. V programovacích jazycích (na rozdíl od přirozených jazyků, jako je angličtina) je syntax souborem pravidel, která definují a řídí, jak jsou slova, interpunkce a symboly organizovány v programovacím jazyce. Bez syntaxe je téměř nemožné pochopit sémantiku nebo význam jazyka. Kompilátor nebo interpretátor nebudou schopni pochopit kód, pokud se nedodržuje syntax jazyka. (Techterms.com, 2011)

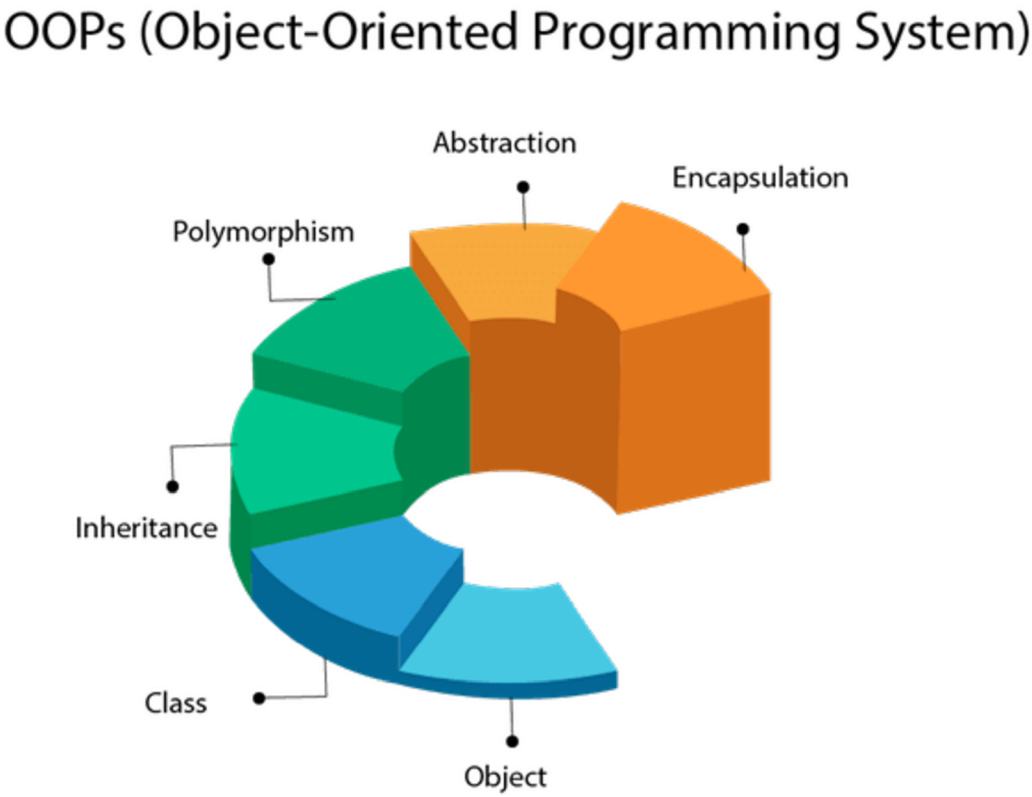
Zapouzdření (Encapsulation)

Zapouzdření je proces seskupení funkcí a dat do jediné entity. Pro přístup k těmto datovým členům musí být rozsah členské funkce nastaven na "public", zatímco rozsah datových členů musí být nastaven na "private". Podle této teorie obsahuje položka veškeré důležité informace; pouze malá podmnožina je zpřístupněna vnějšímu světu. Každý objekt má privátní třídu, která obsahuje jeho implementaci a stav.

Polymorfismus (Polymorphism)

Polymorfismus umožňuje více třídám používat stejný název metody, což zahrnuje také předefinování metod pro odvozené třídy. Existují dva typy polymorfismu: polymorfismus za běhu a polymorfismus za kompilace. Kromě toho, že mají několik forem, jsou objekty vytvořeny tak, aby měly společné chování. Aby se zabránilo psaní duplicitního kódu, software určí, které využití nebo význam je požadován pokaždé, když je použit objekt z rodičovské třídy. (Khanna, 2021)

Obrázek 10 - OOP struktura



Zdroj: medium.com, 2020

Dědičnost (Inheritance)

V nejširším smyslu dědičnost odkazuje na proces získávání vlastností. Jeden objekt v OOP dědí vlastnosti jiného objektu. Vývojáři mohou znovu používat běžné funkčnosti při zachování jasné hierarchie tím, že přiřadí vztahy a podtřídy mezi objekty. Tato vlastnost OOP urychluje vývoj a zajišťuje větší přesnost tím, že vyžaduje podrobnější prozkoumání dat. Vztah mezi rodičem a potomkem je znázorněn prostřednictvím dědičnosti. (programiz.com, 2022)

Abstrakce (Abstraction)

Jedním z konceptů OOP v Javě je abstrakce, což je čin reprezentace klíčových vlastností bez zahrnutí podpůrných informací. Je to metoda pro vývoj nového datového typu vhodného pro konkrétní aplikaci. Vyhneme se poskytování nadbytečných nebo nepodstatných faktů a zobrazujeme pouze přesnou část, kterou uživatel požadoval. Je to klíčové, protože vás to zabraňuje dělat stejný úkol více než jednou. (Quanit, 2021)

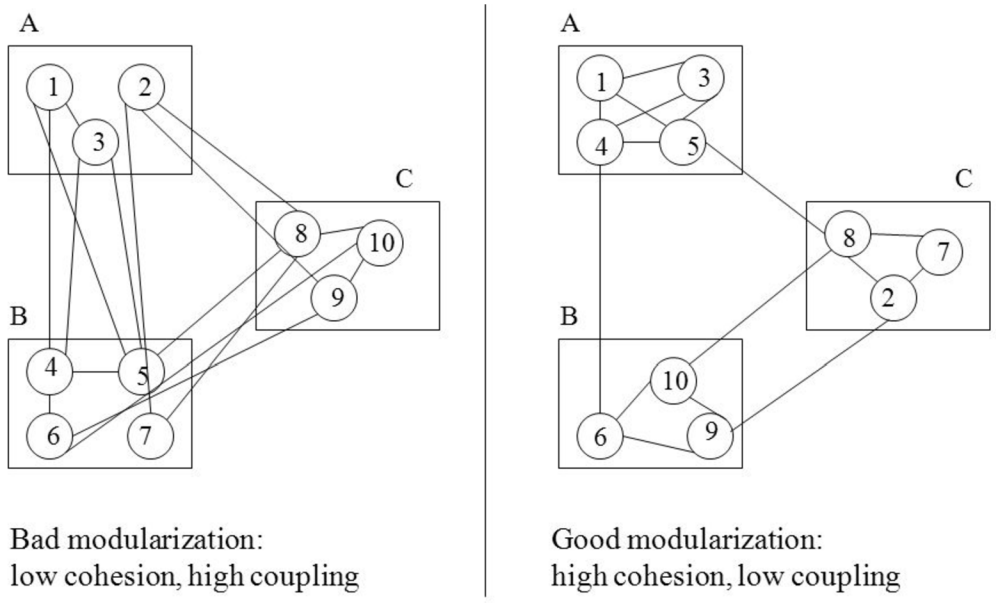
Spojení (Coupling)

Spojení popisuje míru, do jaké je jeden softwarový prvek spojen s jiným. Softwarové prvky mohou být třída, balíček, komponenta, podsystém nebo systém. Označuje úroveň znalosti, kterou má jeden objekt nebo třída o druhém. To znamená, že pokud jedna třída změní své atributy, závislé změny v druhém se také změní. Velikost vzájemné závislosti mezi dvěma třídami určí, jak tyto změny nastanou. (Quanit, 2021)

Soudržnost (Cohesion)

Soudržnost třídy je určena tím, jak úzce a smysluplně jsou její metody a vlastnosti navzájem spojeny, a také tím, jak intenzivně se zaměřují na provedení jednoho, jasně definovaného cíle pro systém. Je to míra, jak úzce se zaměřují povinnosti třídy. Třídy s nízkou soudržností jsou obtížné udržovat, protože jejich metody a vlastnosti logicky nesouvisí jedna s druhou. (devopedia.org, 2019)

Obrázek 11 - Spojení x Soudržnost



Zdroj: Viva differences, 2019

Agregace (Aggregation)

V této metodě má každý objekt odlišný životní cyklus. Vlastnictví však brání dětskému objektu stát se součástí jiného rodičovského objektu. Agregace v Javě znázorňuje vazbu mezi objektem, který obsahuje další objekty, a je slabou asociací.

Ilustruje to vztah mezi součástí a celkem, kde část může existovat bez celku. Agregace je zvláštní typ sémanticky slabé vazby, která nastane, když jsou kombinovány nesouvisející věci. (Kanjilal, 2018)

Kompozice (Composition)

Kompozice je asociace, která znázorňuje vztah mezi částí a celkem, kde část nemůže existovat bez celku. Agregace může mít různé formy, včetně kompozice. Jelikož dětské objekty nemají životní cyklus, všechny automaticky zanikají, když zanikne rodičovský objekt. V žádné kompozici mezi dvěma entitami nemůže jeden objekt existovat bez druhého. V důsledku toho jsou obě entity závislé jedna na druhé ve své kompozici.

Modularita (Modularity)

Modulární design odkazuje na rozdělení systému na mnoho funkčních částí (označovaných jako moduly), které lze kombinovat k vytvoření rozsáhlejší aplikace. Modularita a zapouzdření jsou neoddělitelně propojeny. Při mapování zapouzdřených abstrakcí do skutečných, fyzických modulů lze vidět vysokou soudržnost uvnitř modulů a omezenou interakci mezi moduly nebo spojení jako definici modularity. (Quanit, 2021)

Konstruktory a metody (Constructors and methods)

Konstruktor je speciální druh podprogramu volaného k vytvoření objektu. Nastavuje nový objekt k použití a často přijímá argumenty od konstruktoru k nastavení potřebných členských proměnných. V OOP je metoda procedura spojená se zprávou a objektem. Stavová data objektu a jeho chování tvoří jeho rozhraní, které popisuje, jak ho mohou používat jeho různí spotřebitelé. Metoda je činnost objektu parametrizovaná spotřebitelem. (scaler.com, 2023)

3.4.5 Výhody OOP

Přes vzestup různých programovacích modelů zůstává OOP populární v DevOps. Důvodem jsou následující výhody, které poskytuje:

Umožňuje opakované použití kódu

Myšlenka dědičnosti je jedním z klíčových konceptů, které objektově orientované programování nabízí. Atributy třídy mohou být předávány dědičností, což eliminuje potřebu duplikace úsilí. Tím se předejde problémům spojeným s opakovaným psaním stejného kódu. (scaler.com, 2023)

Díky zavedení myšlenky tříd lze část kódu použít tolikrát, kolik je v programu potřeba. Dětská třída, která využívá metodu dědičnosti, zdědí pole a metody rodičovské třídy. Lze snadno upravit dostupné metody a hodnoty rodičovské třídy. (scaler.com, 2023)

Zvyšuje produktivitu ve vývoji softwaru

Můžeme vytvářet programy z předem napsaných, propojených modulů, místo aby bylo nutné začínat od nuly, což ušetří čas a zvýší produktivitu. Díky jazyku OOP můžeme software rozdělit na zvládnutelné, diskrétní problémy. Objektově orientované programování je modulární, protože umožňuje dělení práce při tvorbě programů založených na objektech.

Je také rozšiřitelné, protože můžete přidávat nové charakteristiky a akce k objektům. Objekty lze využívat v několika aplikacích. Objektově orientované programování zvyšuje produktivitu vývoje softwaru ve srovnání s konvenčními procedurálními programovacími technikami díky modularitě, rozšiřitelnosti a opakovanému použití. (2023, developer.com)

Zjednodušuje řešení problémů

Při použití objektově orientovaného programování je řešení problémů jednodušší, protože uživatel ví, kde v kódu hledat zdroj problému. Jelikož chyba ukáže, kde je problém, není třeba prohlížet další části kódu. Všechny objekty v objektově orientovaném programování (OOP) jsou samostatné, což je jedna z výhod používání zapouzdření. DevOps inženýři a vývojáři získávají mnoho výhod z tohoto multimodálního chování, protože nyní mohou pracovat na několika projektech najednou s výhodou vyhýbání se duplikaci kódu. (2023, developer.com)

Posiluje bezpečnost

Pro udržení bezpečnosti aplikace a poskytování klíčových dat k prohlížení filtrujeme omezená data prostřednictvím mechanismů skrývání dat a abstrakce. Koncept datové

abstrakce v OOP umožňuje zobrazit uživateli pouze malé množství dat, což je jedna ze silných stránek OOP.

Když jsou přístupné pouze nezbytné informace, zbytek není. To umožňuje udržovat bezpečnost. Další sada výhod OOP v konceptu abstrakce Javy se používá ke skrytí složitosti před ostatními uživateli a zobrazení informací o prvku podle požadavků. (2023, developer.com)

Zjednodušuje údržbu kódu

Software založený na objektovém programování je z hlediska kódu jednodušší na údržbu. Díky modularitě designu je možné v případě problémů aktualizovat část systému bez potřeby rozsáhlých úprav. Navíc můžete upravit již existující objekty a vytvořit nové.

Tato schopnost by byla výhodná pro jakýkoli programovací jazyk; brání uživatelům, aby museli znovu vykonávat práci různými způsoby. Udržování a aktualizace stávajících kódů přidáním nových změn je vždy jednoduché a šetří čas. Jelikož lze vytvářet nové objekty s pouze malými variacemi od starých, je jednoduché udržovat a modifikovat stávající kód. (Boyles, 2023)

Zabraňuje opakování dat

Redundantní data se vztahují k datům, která byla zdvojena. V důsledku toho jsou stejné informace opakovány. Redundance dat je považována za výhodu v objektově orientovaném programování. Například uživatel by si přál funkci srovnatelnou s téměř všemi třídami.

V takových případech může uživatel vytvořit třídy s podobnou funkcionalitou a zdědit je, když je to nutné. Značnou výhodou OOP je redundance dat. Uživatelé, kteří chtějí podobnou funkci ve více třídách, mohou napsat standardní definice tříd pro tyto funkce a zdědit je. (Boyles, 2023)

Vede k flexibilnímu kódu

Polymorfismus je myšlenka, která umožňuje flexibilitu. Následující výhody polymorfismu pro vývojáře jsou rozšiřitelnost a jednoduchost. Jednou z výhod OOP je polymorfismus, který umožňuje, aby kus kódu existoval ve více verzích. Například můžete jednat odlišně, pokud se změní prostředí nebo situace.

Podívejme se na jednoduchý příklad. Na trhu bude osoba jednat jako zákazník; ve škole bude osoba jednat jako student; a doma bude osoba jednat jako syn nebo dcera. Zde tato samá osoba vykazuje různá chování v závislosti na prostředí. (Boyles, 2023)

Řeší problémy již v rané fázi

Další výhodou objektově orientovaného programování je, že může účinně řešit problémy tím, že je rozdělí na menší části. Stává se dobrým programovacím zvykem rozložit složitý problém na jednodušší části nebo komponenty. S touto informací OOP využívá funkci, která rozděluje programový kód na menší, lépe zvládnutelné kousky vyvíjené jednotlivě.

Jakmile je problém rozložen, můžete jednotlivé kusy znovu použít k řešení dalších problémů. Navíc můžete vyměnit menší kódy za moduly se stejným rozhraním a implementačními detaily. (Boyles, 2023)

Poskytuje výhody v designu

Významným vývojem v softwarovém inženýrství byl objektově orientovaný vývoj. Mimo jiné slibuje zkrácení doby vývoje a poskytuje firmám konkurenční výhodu. Designovou výhodou, kterou uživatelé zažijí díky OOP, je snadnost, s jakou mohou věci navrhovat a opravovat, a snížení rizik, pokud nějaká existují.

Zde vyžadují objektově orientované programy dlouhou a důkladnou fázi designu od designérů, která produkuje lepší návrhy s menším počtem chyb. Po dosažení určitých základních omezení je jednodušší programovat všechny ne-OOP samostatně. (Boyles, 2023)

Snížení nákladů na vývoj

Použití objektově orientovaného přístupu umožňuje snížit některé přímé náklady spojené se systémy, včetně údržby a vývoje. Opakované použití softwaru také snižuje cenu vývoje. Ve většině případů je na objektově orientovanou analýzu a design vynaloženo více času a úsilí, což snižuje celkové náklady na vývoj.

Obecně je cena za vylepšení snížena, protože je obvykle věnováno více úsilí na specifické hodnocení a plánování článků. Náklady na vývoj jsou obecně sníženy, protože

je obvykle věnováno více času a úsilí na objektově orientovanou analýzu a design. (Boyles, 2023)

3.5 Metodologie návrhu IS

3.5.1 Principy a Metody návrhu IS

V této části se zaměřujeme na principy a metody používané při návrhu informačních systémů. Hlavním cílem je vytvoření systémů, které jsou efektivní, bezpečné, uživatelsky přívětivé a technologicky udržitelné. Mezi klíčové přístupy patří:

- Strukturované Metody: Tyto metody, jako je například model vodopádu, poskytují systematický přístup k vývoji IS, kde každá fáze projektu má svůj začátek a konec. Vyznačují se důrazem na počáteční plánování a podrobnou specifikaci požadavků.
- Objektově Orientovaný Návrh: Tento přístup se zaměřuje na modelování systému jako sady vzájemně propojených objektů. Objektově orientovaný design umožňuje větší flexibilitu a snadnější údržbu softwaru.
- Agilní Metodiky: Agilní přístupy jako Scrum a Kanban jsou flexibilní a přizpůsobivé, umožňující rychlou reakci na změny požadavků během vývojového procesu. Tyto metody podporují iterativní vývoj a časté testování.
- Lean Vývoj: Lean metodologie se zaměřuje na minimalizaci plýtvání zdrojů a zefektivnění procesů. Je založena na principech štíhlé výroby a neustálého zlepšování.

Tyto metodiky jsou aplikovány v různých kombinacích a variacích v závislosti na specifikách projektu, potřebách klienta a charakteristice vývojového týmu. (Frost, Pike, Kenyo, Pels, 2011)

3.5.2 Uživatelské požadavky a funkční specifikace

Fáze definice uživatelských požadavků a funkčních specifikací je klíčová pro úspěšný návrh informačních systémů. Tento proces zahrnuje:

- Sběr Požadavků: Interakce se zainteresovanými stranami pro získání komplexního přehledu o potřebách a očekáváních. To zahrnuje rozhovory, dotazníky, pozorování a pracovní skupiny.
- Analýza Požadavků: Kritické hodnocení a klasifikace získaných požadavků. To pomáhá identifikovat skutečné potřeby uživatelů a rozlišit mezi základními požadavky a těmi, které jsou méně důležité nebo realizovatelné.
- Specifikace Požadavků: Transformace analyzovaných požadavků do strukturovaného dokumentu, který slouží jako základ pro následný design a vývoj systému.
- Validace a Verifikace: Zajištění, že specifikace odpovídají uživatelským potřebám a jsou technicky realizovatelné.

Tento proces je iterativní a často zahrnuje opakované konzultace se zainteresovanými stranami pro upřesnění a potvrzení požadavků. (Frost, Pike, Kenyo, Pels, 2011)

3.5.3 Architektura a datový model navrhovaného IS

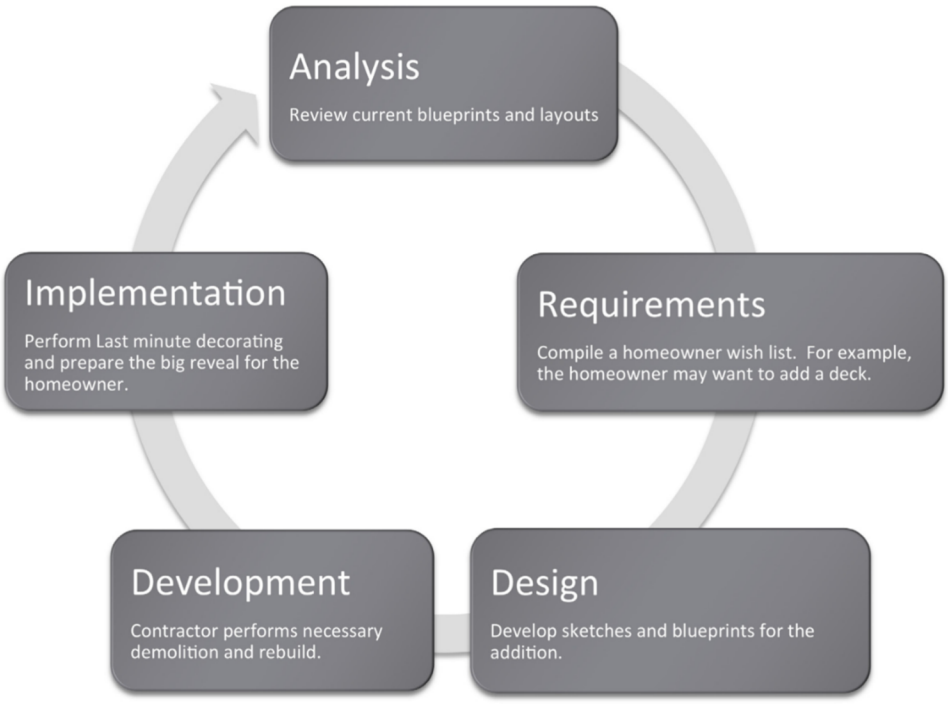
Architektura informačního systému (IS) a jeho datový model jsou zásadním stavebním kamenem celého návrhu. Tyto prvky určují, jakým způsobem budou data organizována, jak bude IS strukturován a jaké procesy bude schopen provádět. Následující body se věnují těmto aspektům:

- Typy Architektur: Rozlišujeme několik typů architektur informačních systémů, jako například klient-server, vícevrstvou (n-tier), mikro služby (microservices) apod. Každý z těchto přístupů má své výhody a nevýhody a volba závisí na konkrétních potřebách projektu.
- Struktura Datového Modelu: Datový model definuje, jak jsou data organizována v systému. Zahrnuje popis entit (objektů), jejich atributů (vlastností) a vztahů mezi nimi. Tento model může být hierarchický, relační, objektově orientovaný nebo grafový, v závislosti na charakteru dat.
- Normalizace Dat: V relačních databázích je normalizace procesem, který zajišťuje efektivní organizaci dat a minimalizuje redundanci. To je důležité pro zachování konzistence a integrity dat.

- Principy Řízení Dat: Definice, jak budou data ukládána, upravována a získávána. Zahrnuje práva přístupu, zabezpečení dat, zálohování a obnovení dat.
- Technologie a Nástroje: Výběr technologií a nástrojů pro implementaci architektury a datového modelu. To může zahrnovat výběr databázového systému, jazyka programování, frameworků a dalších komponent.
- Optimalizace Výkonu: Zajištění, že architektura a datový model jsou navrženy tak, aby dosahovaly optimálního výkonu systému. To zahrnuje rychlý přístup k datům, efektivní zpracování a minimalizaci zpoždění.
- Škálovatelnost a Rozšiřitelnost: Navržení systému tak, aby byl škálovatelný, což znamená, že lze zvyšovat jeho výkon s růstem požadavků. Rozšiřitelnost zase umožňuje snadnou integraci nových funkcí a komponent.

Architektura a datový model jsou základními stavebními kameny, které ovlivňují všechny další části návrhu informačního systému. Jejich správný výběr a návrh jsou klíčové pro dosažení funkcionalit, výkonu a udržitelnosti systému. (ibm.com, 2023. Martin, Robert C., 2017)

Obrázek 12 - Cyklus vývoje systémů



Zdroj: collegesidekick.com, 2024

4 Vlastní práce

4.1 Předmluva k vlastní práci

Vlastní práce se zaměřuje na zhodnocení existujícího systému pojišťovny. V původním smyslu práce mělo jít o zhodnocení nedostatků a problematických částí systému. Toto hodnocení představuje východisko pro následný návrh aplikace, který bude nabídnut pojišťovně k budoucímu vývoji.

Během tvoření analýzy, popisu a hrubého zadání bylo spolupracováno s interním týmem pojišťovny. Bohužel však během procesu tvoření práce došlo k přerušení spolupráce ze strany pojišťovny, a proto se v této práci neobjevuje název dané společnosti ani žádné konkrétní údaje, které by mohly vést k přesnému určení klienta. Taktéž došlo k přerušení ještě před vypracováním finálních návrhů, tím pádem se jedná o návrhy, které ze strany klienta nedostaly zpětnou vazbu.

V následujících kapitolách jsou tedy všechny popsané skutečnosti realitou, která již existuje a je podložena již existujícím systémem. Zadání požadavků bylo specifikováno s klientem v rámci quality assurance služby.

V místech kde není specifikováno bylo dodržováno zadání klienta. V úskalí systému, kde byly nejasnosti jsou vysvětleny a je v těchto případech řešení vždy řešeno podle standardů čisté architektury a tzv. „best practices“.

4.2 Analýza původního informačního systému

Tato kapitola se věnuje důkladné analýze původního informačního systému (dále jen OPS), který sloužil jako základ pro naši práci na návrhu a implementaci nového pojišťovnického informačního systému. Analýza existujícího systému je klíčovým krokem při plánování a implementaci nového IS, protože nám umožňuje získat hlubší porozumění stávajícímu stavu, identifikovat jeho silné a slabé stránky a určit oblasti, které vyžadují zlepšení.

Hlavní cíl v této kapitole je poskytnout komplexní pohled na OPS, který zahrnuje jeho strukturu, funkce, procesy a výkon. Budeme se zabývat klíčovými aspekty, jako jsou datové modely, uživatelská rozhraní, procesy zpracování transakcí a provozní výkonnost.

Důrazem bude také identifikace nedostatků a výzev, kterým původní systém čelil, a které byly impulsem pro zahájení tohoto projektu.

Tato analýza bude sloužit jako základ pro návrh nového informačního systému, který má za cíl přinést výrazná zlepšení v efektivitě, výkonu a schopnosti našeho pojišťovacího podniku plnit potřeby našich klientů v dnešním konkurenčním prostředí. Začneme tedy detailním rozborem.

4.2.1 Popis původního systému

Webová aplikace pro správu odchozích klientských plateb představuje systém, který usnadňuje manipulaci s platebními transakcemi a jejich vyplácením klientům na základě jejich preferencí. Tato aplikace přijímá platební transakce z různých zdrojových systémů, jako jsou systémy Oxx a Lxx (názvy vynechány z důvodu anonymity podniku), a umožňuje jejich zpracování a rozdělení mezi různé příjemce. Důležité je, že tato aplikace nepřímo nekomunikuje s bankovními systémy, ale připravuje platební dávky k vyplacení prostřednictvím různých platebních kanálů. V následující části je prozkoumáno, co tato aplikace konkrétně umožňuje.

V rámci zkoumané aplikace uživatelé mohou využívat širokou škálu funkcí, které zajišťují správu finančních operací. Klíčovým prvkem je schopnost pohodlně přerozdělit vyplácené částky mezi více příjemců a zároveň sjednotit identické platby z různých transakcí do jednoho srozumitelného celku.

Uživatelé mají možnost aktivně schvalovat jednotlivé transakce, což přináší zvýšenou kontrolu nad procesem a zajišťuje bezpečnost finančních transakcí. Další zajímavou funkcí naší aplikace je schopnost připravovat platební dávky pro různé platební kanály, což podporuje vyplácení finančních prostředků.

Sledování historie transakcí patří mezi klíčové aspekty naší aplikace, umožňující uživatelům podrobně monitorovat a analyzovat veškeré provedené platby. Tato funkcionalita má zásadní význam nejen pro archivaci údajů, ale i pro auditní účely.

Aplikace dále disponuje pokročilými auditními funkcemi, které umožňují uživatelům sledovat a kontrolovat všechny prováděné operace, zajišťující tak transparentnost a důvěryhodnost v celém procesu finanční správy.

Slovník pojmů

- OPS – Systém odchozích plateb (z anglického názvu Outgoing Payment System).
- PAS – Zdrojový systém (Oxx/Lxx).
- OWS – Lustrační systém.
- DWH – Lustrační systém (z datového skladu).
- Transakce – Platební transakce, která má být vyplacena.
- Příjemce – Klient nebo jiná osoba, která má obdržet vyplacenou částku, jeden platební příkaz.
- Spojení plateb – Sloučení více stejných příjemců z více transakcí do jedné položky pro vyplacení.
- Workflow – Stavový stroj, který řídí životní cyklus transakce.
- SSO – Jediné přihlášení, z anglického Single Sign-On.
- AD – Active Directory – adresářové služby LDAP od Microsoft.
- TWS – Treasury Workstation – systém pro vyplácení peněz.

4.2.2 Správa uživatelů a přihlašování

Přihlášení do této aplikace probíhá bez nutnosti ručního zadávání uživatelského jména a hesla. Aplikace využívá tzv. jediného přihlášení (SSO), což znamená, že je nutné, aby uživatel byl již předem přihlášen do domény XXX (Z důvodu mlčenlivosti není jméno společnosti zveřejněno – dále XXX). Po ověření tohoto přihlášení se systém kontroluje, zda má uživatel přístupová oprávnění pro tuto aplikaci. Různé funkce aplikace budou uživateli zpřístupněny na základě rolí a oprávnění, které má přiřazeny.

Aby mohl uživatel získat přístup k této aplikaci, musí být součástí jedné nebo více Active Directory (AD) skupin, které jsou automaticky synchronizovány s aplikací OPS. Každá z těchto AD skupin definuje aplikační role a přidělená oprávnění.

4.2.3 Uživatelské role

- PAS operátor
    - Zodpovědný za přípravu příchozích transakcí ze zdrojových systémů PAS pro vyplacení.

    - Má právo upravovat transakce podle potřeby.
    - Schopen efektivně rozdělovat a spojovat platby podle specifikací.
    - Individuálně deleguje svou transakci na jiného uživatele, zvyšuje tím flexibilitu pracovního procesu.
- Vedoucí
    - Má oprávnění jednotlivě nebo hromadně delegovat transakce operátorům ve svém týmu.
    - Schvaluje transakce provedené operátory ve svém týmu, což zajišťuje kontrolu a bezpečnost platebních operací.
- Hlavní vedoucí
    - Sdílí oprávnění s vedoucím.
    - Má možnost zobrazit různé reporty pro komplexní analýzu finančních transakcí.
    - Schvaluje transakce s vysokými částkami, které již prošly schválením vedoucího.
- FIN operator
    - Zodpovědný za přiřazení jednotlivých plateb do platebních souborů.
    - Uzavírá platební soubory a provádí jejich export pro bankovní systémy.
    - Importuje výpisy z bankovních systémů zpět do aplikace OPS.
    - Má možnost manuálně zadat úspěšnost provedených plateb.
- Reader
    - Speciální uživatel s přístupem k prohlížení všech transakcí, avšak bez oprávnění ke změnám nebo úpravám.
- Administrátor (aplikace)
    - Má přístup k celkové konfiguraci aplikace pro optimalizaci provozu.
    - Oprávněn upravovat systémové číselníky podle potřeby.
    - Má právo přiřazovat uživatele do týmů pro efektivní správu pracovních skupin.
- Administrátor (podpora IT)
    - Má přístup k celkové konfiguraci aplikace s omezeným přístupem k úpravám systémových číselníků.

- Oprávněn přiřazovat uživatele do týmů pro efektivní organizaci pracovních skupin.

4.2.4 Pohledy aplikace a jejich očekávané funkce

Na každém pohledu aplikace je aktivní hlavní menu. Toto menu funguje jako centrální bod, kde uživatelé rychle lokalizují odkazy na hlavní seznamy a moduly aplikace. Tento přehledný panel zjednodušuje každodenní práci poskytujíc rychlý přístup ke klíčovým částem systému. V rámci aplikace se můžeme potkat s následujícími pohledy / moduly.

Níže jsou tyto moduly sepsány. Všechny moduly, které vykazovaly v minulosti systému problematické chování, či nefunkčnost, jsou zvýrazněny červenou barvou.

- **PAS Dashboard** poskytuje přehled transakcí v systému, zejména pro operátory a vedoucí. Operátoři zde vidí transakce, které mají přiřazené, zatímco vedoucí mají přístup k transakcím svého týmu.
- **FIN Dashboard** je zaměřen na finanční operátory a zahrnuje seznam plateb čekajících na vyplacení. To efektivně usnadňuje správu finančního toku v rámci aplikace.
- **Dearchivace** představuje modul, který umožňuje prohledávat archivované transakce a případně provádět jejich dearchivaci. Tato funkce je užitečná při vyhledávání historických dat.
- **Reporty – Transakce čekající na schválení** zahrnují seznam transakcí, které vyžadují schválení od vedoucích, což je klíčové pro řízení procesu schvalování transakcí.
- **Reporty – Odmítnuté návrhy na sloučení** zobrazují transakce, které byly automaticky navrženy k sloučení, ale byly následně zamítnuty operátory. Tato informace pomáhá identifikovat a řešit případné problémy s automatickým spojováním transakcí.
- **Reporty – Audit**, který poskytuje detailní audit všech změn v databázi, umožňující sledovat, kdo a kdy provedl určité operace.
- **Reporty – Neaktivní uživatelé** představuje speciální report zobrazující aktivity uživatelů, kteří měli být v určitý den neaktivní. Tato informace je důležitá pro sledování pracovní produktivity.

- **Reporty – Historie plateb** umožňuje zobrazit historický seznam plateb, což je užitečné pro sledování platební historie a historických dat.
- **Reporty – Stav systému** obsahuje speciální reporty, které umožňují uživatelům sledovat aktuální stav systému, včetně výkonnostních metrik a dalších důležitých informací.
- **Reporty – Log systému** je místo, kde můžete nalézt systémový log aplikace. Tato informace je užitečná při diagnostice problémů a monitorování provozu.
- **Konfigurace – Role** obsahuje číselník uživatelských rolí, určujících oprávnění různých uživatelů v systému.
- **Konfigurace – Uživatelé** Poskytuje přehled všech uživatelů systému, včetně jejich rolí, týmů a limitů, což je důležité pro správu uživatelských účtů.
- **Konfigurace – SLA** (Service Level Agreements): Určuje limity pro schvalování transakcí, což je klíčové pro dodržování dohodnutých časových rámů.
- **Konfigurace – Limity** Definují limity pro schvalování transakcí a osoby oprávněné k jejich schvalování.
- **Konfigurace – Daňové sazby** Obsahují sazby daní pro účely zdanění transakcí ze systému LifeFit a stanovují datum jejich platnosti.
- **Konfigurace – Finanční účty** Seznamuje uživatele se všemi finančními účty používanými pro automatické účetní operace.
- **Konfigurace – Banky** Obsahuje číselník bank s jejich kódy, TWS směrováním a SWIFT kódy, což je důležité pro provádění bankovních operací.
- **Konfigurace – Bankovní poplatky** Poskytuje informace o bankovních poplatcích pro různé platební kanály, což může ovlivnit náklady spojené s transakcemi.
- **Konfigurace – Měny** Zahrnují číselník měn rozpoznávaných systémem OPS, což je důležité pro mezinárodní transakce.
- **Konfigurace – Výjimky v kalendáři (svátky)** Obsahují informace o kalendářních svátcích, které mohou ovlivnit provoz a termíny.
- **Konfigurace – Šablony dopisů (kódy)** Slouží k definování tiskových šablon dopisů pro komunikaci s klienty.

- **Konfigurace – Šablony emailů** Obsahují šablony notifikačních emailů, které mohou být automaticky odesílány uživatelům.
- **Konfigurace – Typ transakce** Číselník různých typů transakcí používaných v systému.
- **Konfigurace – Parametry systému** Zahrnují speciální systémové parametry, které ovlivňují chování aplikace.

Je tedy na první pohled zřejmé, že velká část aplikace je problematická. Právě v těchto chvílích přichází na řadu analýza rizik, rozsáhlosti problémů a výpočtu (spíše odhadů) nákladů pro obnovu systémů (nejen toho, který zkoumáme, ale i systémů, které byly zmíněny výše (Oxx, Lxx)). Rozsáhlost těchto analýz, nedostatek informací a omezenost zaměření však znemožňuje tuto analýzu provést v rámci této práce. Pracuji tedy s úvahou, že došlo k ustanovení rozhodnutí stávající legacy systémy nahradit za jeden komplexní celek. Následující kapitola je tedy zaměřena na kritické části původního systému, které je třeba mít na paměti při návrhu nového systému

4.3 Identifikace a popis problémů původního IS

V předchozí kapitole byl věcně popsán abstraktní směr a smysl systému. Tato kapitola se však věnuje již konkrétním problematickým celkům a otázkám fungování systému. Tyto problémy je třeba vzít v úvahu pro nový navrhovaný system.

Problémy byly popsány na základě rozhovorů s 16 business uživateli na straně zákazníka jako prioritní body pro vyřešení. Tyto rozhovory probíhaly otevřenou formou s otázkami.:

1. Jaké problémy vidíte v existujícím systému?
2. Navrhujete nějaké konkrétní změny v systému?
3. Existuje proces, který je zbytečně složitý? Lze jej upravit v souladu pravidel společnosti?
4. Jaký je Váš názor na sjednocení doteď oddělených systémů?
5. Jaká je Vaše zkušenost ze spolupráce s dodavateli primárních systémů?
6. Jaký je rozdíl mezi operátorem a likvidátorem?
7. Volná konverzace na téma inovace interního systému.

Z těchto otázek vznikly odpovědi, ze kterých byly sepsány jednotné podkapitoly, které referují o zkušenostech hlavních zaměstnanců, kteří patří do skupiny testerů na straně zákazníka. Tito uživatelé jsou zkušení ve svém oboru a vědí co je potřeba pro zajištění stabilního a rychlého workflow.

4.3.1 Kolize rolí

V předchozí kapitole lze najít vypsané role a daná privilegia. V rámci systému však docházelo k určitým komplikacím. Tyto komplikace byly způsobeny převážně jiným chováním a jinými potřebami různých oddělení. Aplikace totiž byla využívána celou organizací napříč týmy.

V těchto situacích se stávalo, že dle politiky daného oddělení docházelo občas k nedostatečným, či naopak nadměrným právům uživatelů. Jak ale takové role nastavit?

V dané situaci a rozmanitosti týmů by tedy dávalo smysl nastavovat pravomoci dynamicky a ne fixně. To by však vyžadovalo velmi komplexní strukturalizaci jednotlivých částí aplikace. V našem případě by dávalo smysl existující role rozkouskovat na více menších dílů, čím zvýšíme granulitu práv a jsme schopni přesněji dané potřeby splňovat.

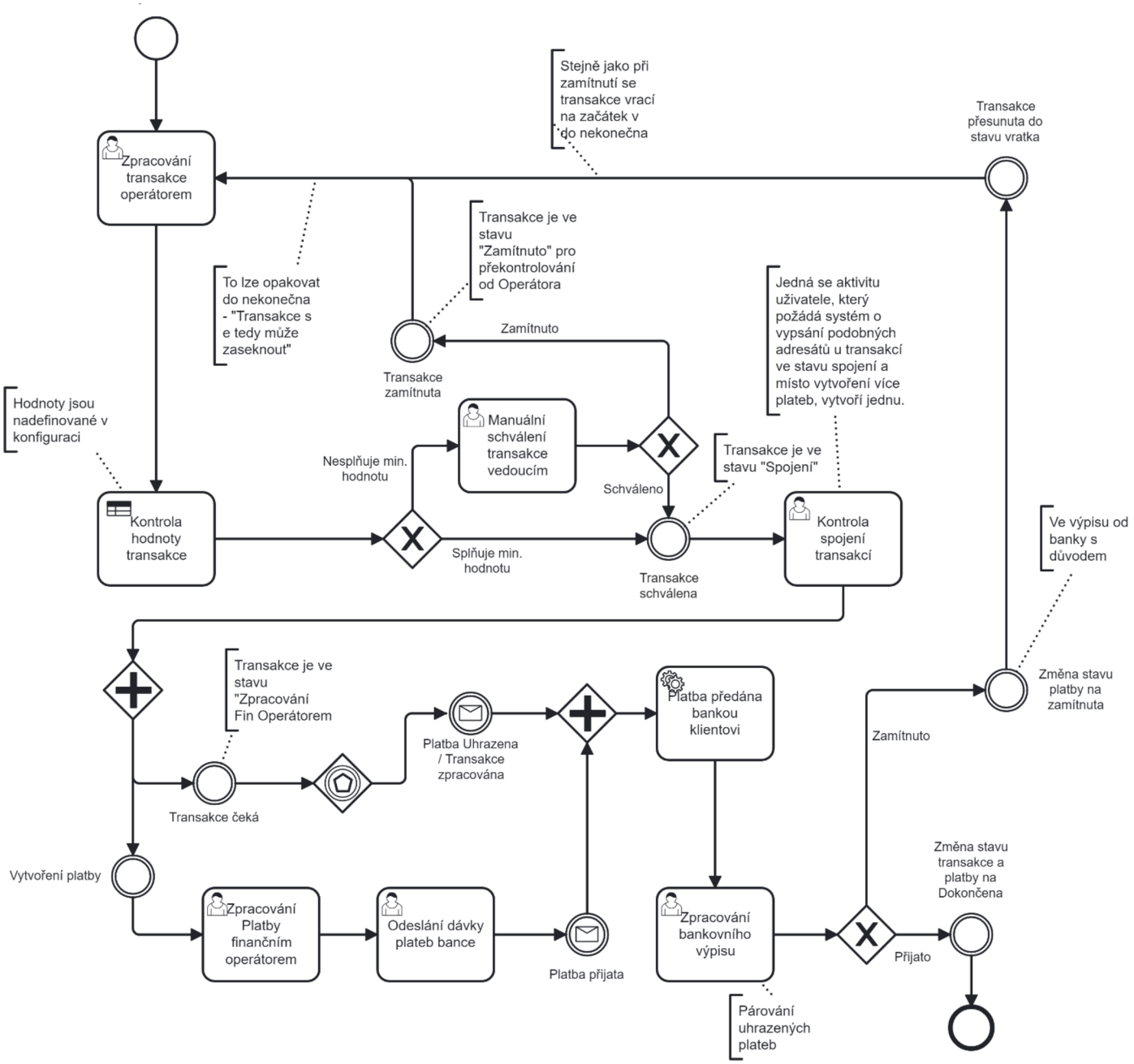
4.3.2 Komplikovaný životní cyklus transakcí

Transakce v našem systému mají poměrně jednoduchý životní cyklus. To však přestane platit v momentě kdy si uvědomíme celý proces od přijmutí po vyplacení. Během procesu se totiž tok rozštěpí z transakce na transakci a platbu i přes to, že z modelu tříd, který je dostupný v kapitole 4.3.9 to není úplně zřejmé. V původním systému totiž byla platba v kódu definována jako položka ve finančním exportu, která měla své stavy – více o platbách v kapitole 4.3.4. To má svým způsobem určitý důvod, ale výhoda dvou entit oproti držení dvou asynchronních toků, které jsou na sobě závislé a může pak docházet k zaseknutí, zde přestává dávat smysl.

Na obrázku níže je možné vidět zjednodušený proces schvalování transakcí ze stávajícího systému.

BPMN diagram níže popisuje nejjednodušší možné zobrazení procesu, bez znázornění kontroly lustrace osob z DWH, kontrol exekucí apod... Největším problémem zde je zapasování spojování plateb a následné vratky nad stejnými transakcemi.

Obrázek 13 – Proces 1 – BPM diagram procesu výplaty původního systému



Zdroj: Autor, 2024

V minulosti se zde často stávalo, že každá implementační změna nad entitou transakcí měla negativní vliv na entitu plateb. Tuto vysokou míru chybovosti omezíme sjednocením těchto entit a výrazným zjednodušením workflow.

Je nutno také říci, že v nově navrženém systému bude muset být ponechán systém schvalování transakcí při nedosažení limitu dané role. Standartně se jedná o situaci, kdy daná role má nastavený strop… pokud je tento strop překonán, transakce přejde do stavu čekající na schválení od nadřízeného. Nicméně z rozhovorů vyplynulo, že každá transakce, která vznikla stejně byla vyplacena, protože zde neexistoval konec při zamítnutí. Zamítnutí výplaty by tedy mělo probíhat již na úrovni vyhodnocení události.

4.3.3 Stavy transakce

V navázání na předchozí kapitolu je vhodné popsat si samotné stavy transakcí. Stavy transakce v OPS jsou celkem složité a je jich spoustu, zde vidí uživatelé prostor na zjednodušení a sjednocení stavů, přičemž by každý stav byl reprezentován různým chováním např. při zobrazení detailu transakce.

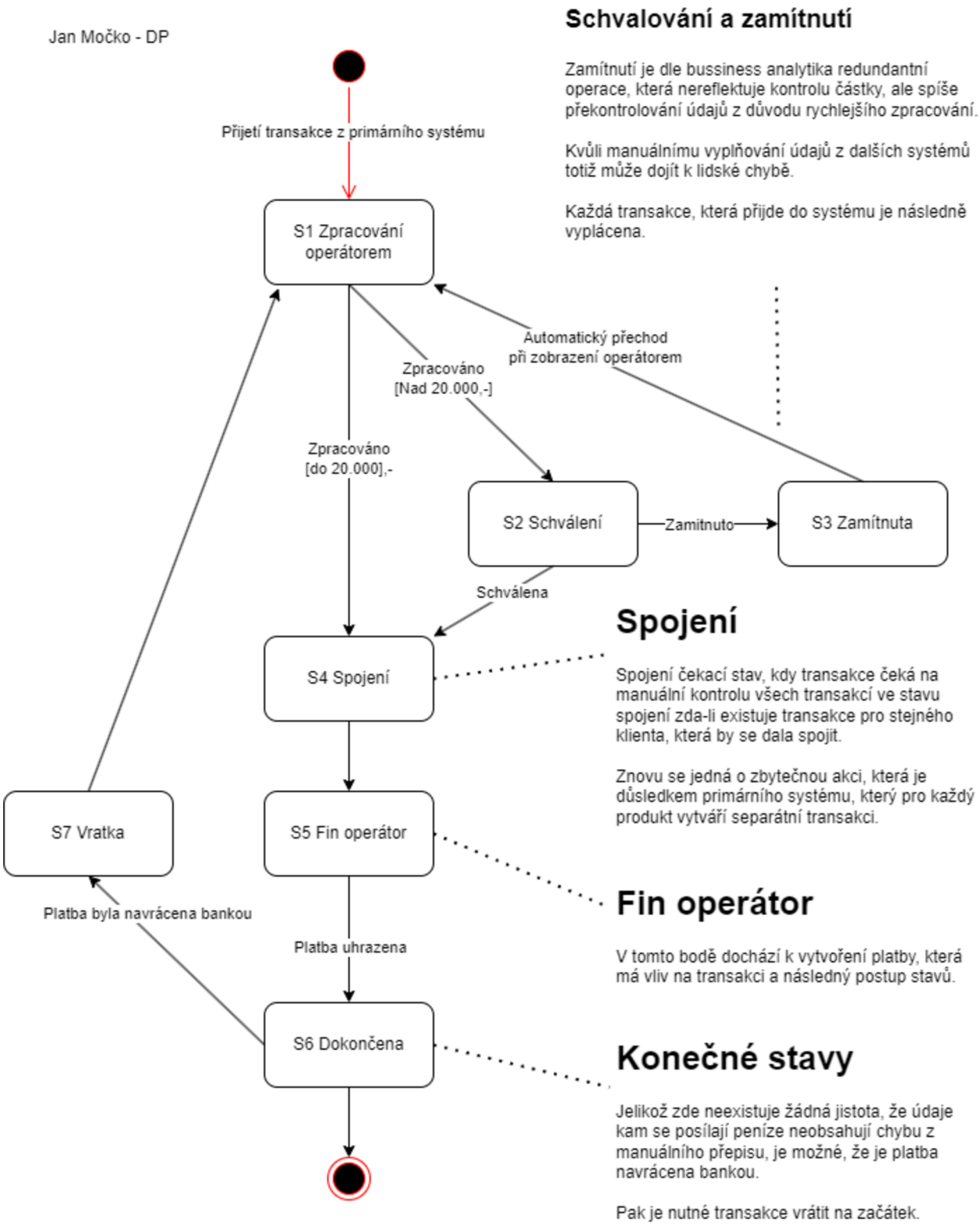
Dále by také přibyly stavy, které má platba a došlo by taktéž k přejmenování.

Momentálně existují tyto stavy transakcí.

a. Operátor (zpracování operátorem): Operátor má možnost změnit povolené položky transakce, přidat příjemce (platby) a určit, jakými platebními kanály budou peníze poslány. Operátor také vytváří a řeší aktivní aletry, které mohou blokovat další zpracování transakce. Po dokončení může transakci přesunout do dalšího stavu, což zahrnuje kontrolu údajů v systémech OWS a DWH a generování alertů, které musí operátor vyřešit.
b. Čekání na schválení. Tento stav popisuje situaci, kdy je transakce po zpracování operátorem v hodnotě převyšující limit, který je nastaven v konfiguraci systému. Schválení transakce je provedeno vedoucím původního operátora.
c. Spojování je stavem kdy se čeká na manuální kontrolu spojení transakcí. To znamená, že se čeká, zda se neobjeví ve zpracování další transakce pro stejného adresáta. Může se totiž stát, že klient využívá plnění z více než jedné smlouvy, pak jsou tyto plnění do systému vytvářeny jako oddělené transakce. Jedná se však o zbytečnou a problematickou funkcionalitu, protože spojením systémů bychom byli schopni reálnou částku k vyplacení sčítat.

d. Stav „Fin“ popisuje stav, kdy k transakci byla vytvořena příslušná platba, která má své workflow a své vlastní stavy. Popis workflow plateb a transakcí je popsán na obrázku výše.
e. Vratka popisuje stav, kdy byla platba provedena ze strany banky, ale z nějakého důvodu nebyla úspěšně provedena. Často tyto situace plynou například ze špatně vyplněného bankovního účtu. Do stavu vratky se pak mohou transakce vracet cyklicky do konce věků.

Obrázek 14 – Stavový diagram 1 – UML Diagram stavů transakce v původním systému



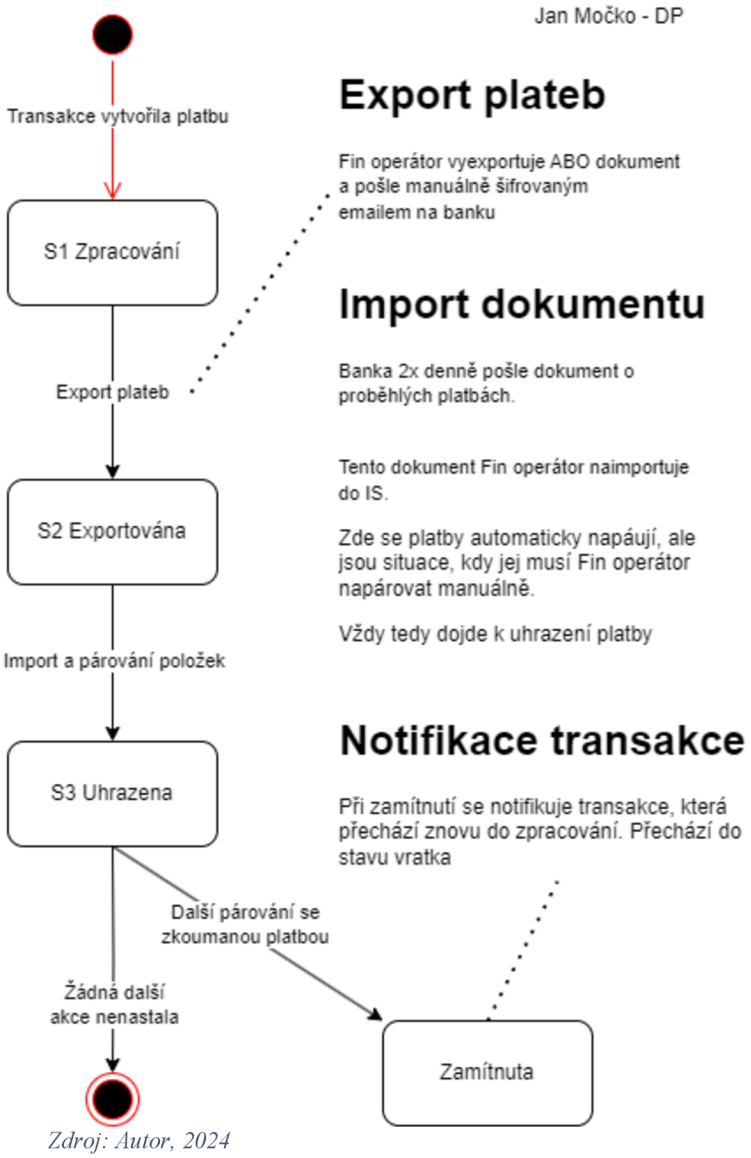
Zdroj: Autor, 2024

4.3.4 Platba vs Transakce

Jak je již zřejmé z výše popsaných kapitol, zkoumaný systém má celkem složitý workflow v poměru s očekávaným výsledkem. Jednoduše řečeno – Systém přijme transakci z primárního systému a po provedení potřebných kroků, je k transakci vytvořena příslušná platba.

Obě entity jsou na sobě úzce závislé a při určitých „edge“ scénářích může docházet k netriviálním chybám, které můžou mít za příčinu např. nevyplacení správné částky, či vyúčtování špatných (opakovaně strhnutých poplatků). To však v novém řešením budeme eliminovat jednodušším navržením entit a procesem zpracování a vyplácení.

Obrázek 15 - Stavový diagram 2 – UML Diagram stavů plateb v původním systému



Zdroj: Autor, 2024

4.3.5 Tři systémy pro jeden cíl

Taktéž již bylo zmíněno, že naše zkoumaná aplikace momentálně funguje pouze jako koncový bod pro vytváření platebních dokumentů. To však ke složitosti celkového ekosystému nedává praktický smysl.

Momentálně jsou transakce vytvářeny ve dvou „legacy“ systémech, které jsou používány dlouhou dobu. Komunikace je pouze jednosměrná, a proto pokud transakce dorazí do zkoumaného systému, tak se již nikdy nemůže vrátit.

Každý omyl primárních systémů, které jsou dodávány dalšími externími dodavateli (je těžké monitorovat a trackovat původ chyb) je nutné manuálními zásahy opravovat přímo v DB, což je i z pohledu auditu problematické. Jedná se například o duplicitní transakce, které jeden ze zmíněných systémů čas od času nepochopitelně zasílal.

Reálná úspora námahy a nákladů by zde byla docílena sjednocením systémů pro správu klientových produktů a vyplácení. To je samozřejmě z praktického hlediska složitější pro náhradu. Avšak je třeba dodat, že odklady nepříjemných změn a inovací se mohou negativně projevit na následných změnách v budoucnosti.

4.3.6 Problematika daní

Daně se v našem systému netýkají každého produktu, ale pouze těch, u kterých je registrován výdělek v rámci výplat, které se netýkají škodných událostí. Takové produkty jsou samozřejmě tématikou pro jiný obor. Avšak z pohledu architektů a návrhářů takových systémů je třeba definovat správnou cestu pro výpočet daní.

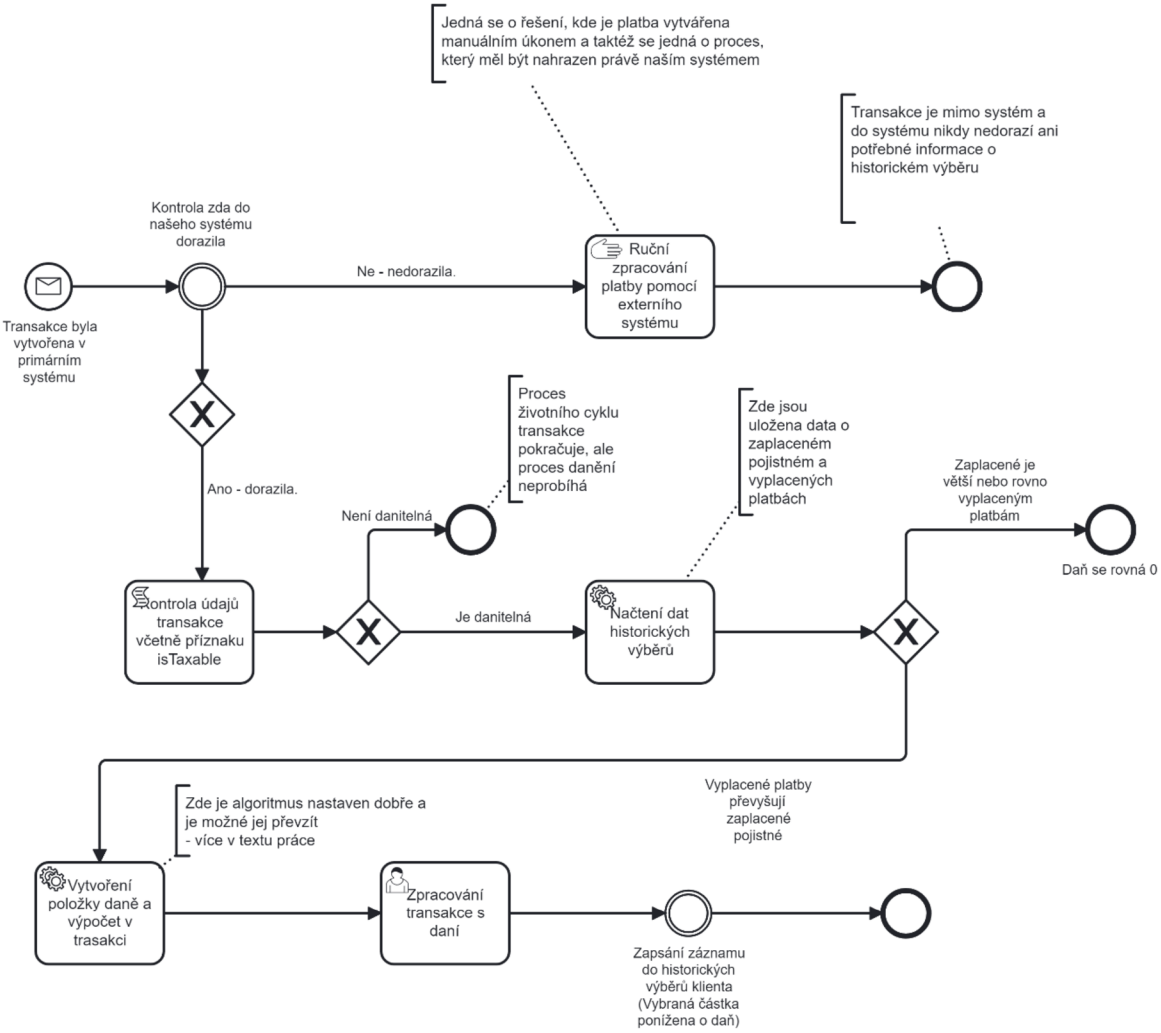
Momentální výpočet by byl v pořádku, avšak nastávají situace, které jsou způsobeny občasným „ztracením transakce“ mezi primárním a naším systémem. Jedná se o situaci, kdy primární systém zaeviduje transakci k výplatě. Ta ale nikdy do zkoumaného systému nedorazí. V rámci tohoto problému bylo prováděno spoustu analýz a „bug trackingů“, ale nikdy se nepodařilo najít jednoznačný původ chyby. Ať už na straně primárních systémů, bránou, přes kterou se transakce propisuje do systému, či ve zkoumaném systému. Taková transakce je potom vyplácena mimo systém, tzv. ručním způsobem v náhradním nástroji.

Výpočet totiž potřebuje brát v úvahu historicky provedené výplaty, které se odečítají od zaplacených „prémií“. Pokud tedy chybí ručně vyplacená transakce, potom je následně

špatně vypočten a uhrazená daň. Tento problém může tedy mít vážné existenční následky a je třeba ho jednou pro vždy vyřešit.

Na obrázku níže je možné vidět rozvětvení metodiky zpracovávání transakcí, které je problematické. Úděl původního systému měl plně nahradit ruční zpracování, které bylo dlouhé a drahé. Nicméně se stále toto řešení vyskytuje kvůli problémovosti primárního systému doručovat transakce ve 100% případech. Bohužel se zde jedná o nespolehlivého dodavatele se kterým klient není schopen nic dořešit a ukončit spolupráci kvůli skutečnosti, že se jedná o dceřinou společnost. Proto je tento problém jako jeden z mnoha řešitelný pouze kompletní náhradou systémů.

Obrázek 16 - BPMN diagram - Proces danění



Zdroj: Autor

4.3.7 Delegace transakcí

V původním systému je představena akce tzv. „delegování“. Tato funkcionalita teoreticky představuje velmi logickou, avšak problematickou funkcionalitu. Původní systém totiž pracoval s myšlenkou, že operátor, který zpracoval transakci v primárním systému, taktéž figuroval v námi zkoumaném systému.

Transakce byla tedy přiřazena na daného operátora a ostatní jej nemohli vidět. Delegování mělo umožnit přeřadit transakci na jiného operátora. To je samozřejmě užitečné, ale systém přiřazování transakcí na určitou osobu je celkem problematické například v situaci kdy transakce, která byla několikrát vrácena ze strany banky a zpracování tudíž trvalo delší časový interval, mohla skončit ve stavu vratky po ukončení pracovního poměru se zaměstnancem. To mohlo vyvrcholit v zaseknutí transakce a následné zapomenuté vyplácení.

4.3.8 Mnoho manuálních úkonů

Původní myšlenka systému bylo vytvoření platformy, která má za úkol co nejefektivněji dostat peníze od pojišťovny ke klientovi. To však často nebylo pravdou z důvodu vysoké míry manuálních úkonů ze strany zaměstnanců (operátorů).

Zvýšením automatizovaných kroků, můžeme zmírnit chybovost manuálních úkonů, mezi které patřilo například manuální vyplňování čísla účtu, adresy apod. u klienta. Taktéž chyběl kontaktní kanál mezi pojišťovnou a klientem v případě potíží.

Nový systém by tento problém měl odstranit tím, že bude mít všechny údaje na jednom místě z první ruky od klienta. Nový workflow taktéž zajistí to, aby se výplata nezasekla a po neúspěšném provedení platby bude klient kontaktován, protože tento krok bude po zaměstnanci vyžadován pro opětovné zpracování transakce.

4.3.9 Neúměrná složitost aplikace

Původní aplikace se na první pohled zdá velice komplexní. Také tomu tak je, ale většina aplikace buď nefunguje, nebo se ani nevyžaduje její funkčnost, či je zbytečně složitá. To je zřejmé taktéž z designu modelu tříd na obrázku níže.

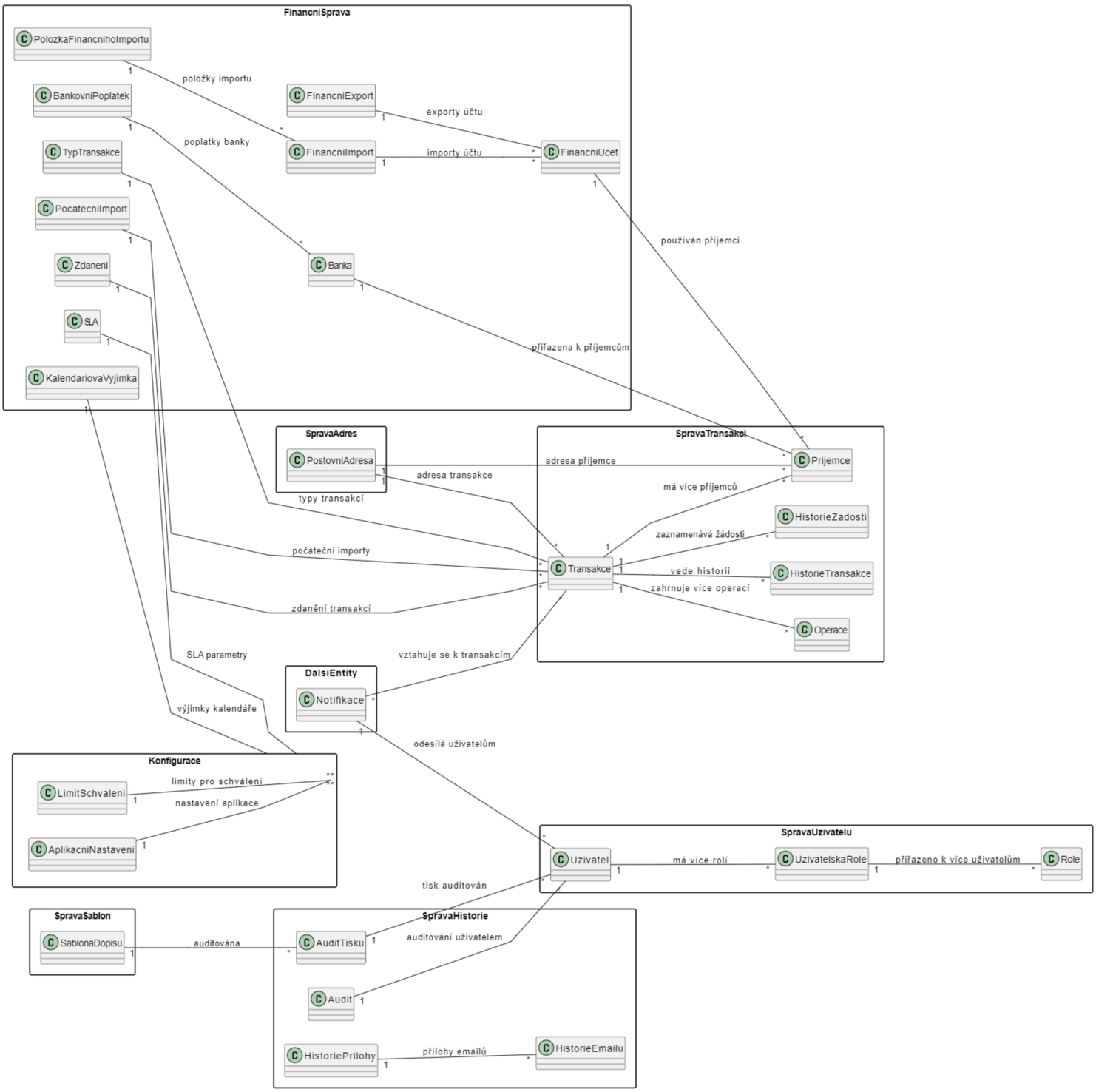
Tento model je však stále značným zjednodušením opravdového systému, vzhledem k tomu, že byly pro přehlednost vynechány vlastnosti daných tříd. Taktéž byly vynechány

některé třídy, které jsou však nepodstatné pro další zkoumání. Samotný model byl vytvářen z kódu inicializace aplikace, který je přiložen v přílohách na konci pod názvem priloga_migrace_dp_mocko.

Takové situace se často v systémech, které jsou vyvíjeny dlouhodobě, stávají a jedná se o běžný problém takových projektů. Důvody proč takové situace nastávají nejsou vždy zcela jasné, ale může jít například o nedostatečnou dokumentaci systému, či časté rotace vývojářů aplikace.

Situace je však řešitelná zjednodušením, které přinese přehlednost uživatelům, ale také budoucím vývojářům, správcům a stakeholderům aplikace.

Obrázek 17 - ER diagram 1 - Model původního systému



Zdroj: Autor, 2024

4.4 Návrh nového IS

V rámci předchozích kapitol byla představena myšlenka a zásadní problémy stávajícího systému. Tato myšlenka by měla se být vším všudy zachována. Problémy původního systému jsou tedy hlavním odrazovým můstkem pro očekávané změny.

Hlavní změnou systému bude sjednocení původních tří systémů do jednoho. Bude se jednat o aplikaci, která bude mít za úkol držet informace o klientech, správu a komunikaci s klientem, a v poslední řadě provádění a následný monitoring výplat.

4.4.1 Specifické nového funkce systému

Uzavírat smlouvy

- Implementace formulářů pro zadání parametrů nových smluv.
- Validace vstupních údajů a podmínek produktů.
- Elektronické podpisování a archivace smluv.
- Notifikace klientů o uzavření nebo změnách smluv.

Provádět výpočet plnění

- Algoritmy pro výpočet výše plnění založené na specifikacích smluv a typu události.
- Možnost manuální korekce výpočtů likvidátorem.
- Automatizované zpracování nároků z událostí a výplat.

Provádět výpočet daně

- Výpočet daně z plnění v souladu s aktuálními daňovými předpisy.
- Generování daňových přiznání a souhrnů pro interní účely i pro klienty.
- Integrace s daňovými kalkulačkami a účetními systémy.

Posílat upozornění

- Automatické generování a odesílání upozornění klientům a zaměstnancům.
- Konfigurovatelné šablony pro různé typy notifikací (email, SMS, v systému).
- Sledování doručení a otevření upozornění.

Schvalovat a zamítat události / výplaty

- Workflow pro revizi a schvalování událostí a nároků na výplaty.
- Role a oprávnění pro různé úrovně schvalování.
- Auditovatelný záznam o všech akcích a rozhodnutích.

Komunikovat s datovým skladem

- Integrace s databázemi a datovými sklady pro extrakci a zpracování dat.
- Rozhraní pro import a export dat, podpora různých datových formátů.
- Zabezpečení přenosu dat a ochrana osobních údajů.
- V rámci návrhu neimplementováno z důvodu neposkytnutých informacích o struktuře a neznámých endpointů.

Komunikovat s bankovním API pro provedení výplaty

- Napojení na bankovní API pro automatické zpracování transakcí.
- Řízení výplat na základě schválených nároků a událostí.
- Zabezpečení finančních transakcí a sledování jejich stavu.
- Příklad Api - https://www.fio.cz/bankovni-sluzby/api-bankovnictvi
- V rámci návrhu neimplementováno z důvodu neprovedeného rozhodnutí o poskytovateli.

4.4.2 Pohledy nového systému

Na základě komunikace s projektovým manažerem a business analytikem pojišťovny byly v rámci zadání vydefinovány pohledy, které by daná aplikace měla obsahovat včetně stručného popisu zobrazení.

Toto zadání je následně vypracování v podobě návrhu designu, který se držel bodů každého pohledu ze zadání. Tyto návrhy jsou k dispozici v přílohách pod číselným označením, které odpovídá jednotlivým bodům níže.

- 1-A/B Přihlášení (Klient / Zaměstnanec)
    - Oddělené přihlášení pro klienta a zaměstnance
    - Zaměstnanci ve společnosti standartně používají k autentizaci PKI kartu. (PKI je podmínkou vzhledem k povaze systému).

    - Systém, jakým způsobem bude probíhat registrace a přihlašování z technického pohledu je v rukou pojišťovny a interního know-how. Přihlášení tedy nebude pokryto technickými detaily.
- 2-A/B/C Registrace / Úprava klienta (Poradce / Klient / Administrátor)
    - Registrace klienta probíhá s poradcem na pobočce.
    - Registrace je úspěšná po vyplnění povinných údajů.
    - V rámci registrace je nutné ověřit klienta SMS ověřením.
    - V rámci registrace dojde k napojení primárního bankovního účtu klienta, tímto způsobem bude zajištěna jistota existujícího účtu. Toto ověření by mělo probíhat např. bankovní identitou jako např. u portálu občana.
    - Dokončením registrace souhlasí klient s podmínkami.
    - Úpravu klienta by měl být schopen provádět klient sám s následným schválením od Administrátora.
    - Přímou úpravu klienta by měl být schopen provádět pouze administrátor.
- 3-A Detail účtu klienta – sekce "O účtu" (Klient)
    - Klient by měl být po přihlášení přesměrován na detail jeho účtu.
    - Zde by měly být dostupné základní informace o klientovi, jeho produktech a přidruženém poradci.
    - Poradci jsou vybírání na základě lokality klienta a poradce.
    - Klient by měl být schopen z této obrazovky kontaktovat klienta.
    - Kontaktování klienta by mělo spustit rutinu k odeslání sms na klienta. (Logika nenadefinována)
- 3-B Detail účtu klienta – sekce "O účtu" (Poradce / Administrátor)
    - Poradce by měl být schopen zobrazit si detail klienta se stejnými informacemi jako klient při navigaci ze seznamu klientů.
    - Poradce by měl mít možnost upravovat stávající produkty klienta a uzavírat nové smlouvy.
    - Administrátor by měl mít možnost zobrazit stejný pohled bez možnosti sjednávat.
- 4-A Detail účtu klienta – sekce „Bankovní spojení" (Klient)

    - Klient by měl být schopen si zobrazit existující spojení, které je možné přejmenovat. Všechny úpravy by měli být přepisovány pomocí bankovní identity.
    - V případě neaktivního spojení bude účet deaktivován.
    - Lze také přidat nové spojení.
- 5-A Seznam klientů (Poradce / Administrátor)
    - Jedná se o seznam, kde lze filtrovat a vyhledávat.
    - Přístup má poradce s možným proklikem na detail klienta, či sjednání nové služby pro daného klienta.
    - Administrátor by měl mít možnost zobrazit si stejný pohled. Nicméně bez možnosti uzavírat nové smlouvy.
- 6-A Sjednání / Úprava služby klienta (Poradce)
    - Tato sekce by měla umožnit vytvářet, či upravovat existující smlouvy daného klienta.
    - Pro tyto úkony je nutná přítomnost klienta, z toho důvodu je třeba přidat ověření SMS jako při registraci.
- 7-A Mé služby (Klient)
    - Klient by měl mít možnost zobrazit si seznam svých smluv.
    - Měl by mít možnost kontaktovat poradce.
    - Měl by mít možnost překliknout se na detail své smlouvy.
    - Měl by zde být také proklik na nabízené služby.
- 8-A Nabízené služby (Klient / Poradce)
    - Jedná se o seznam všech nabízených produktů
    - Uživatel by měl být schopen překliknout se na detail služby.
- 9-A Detail služby (Klient / Poradce)
    - Detail služby by měl poskytnout stručný popis produktu.
    - Mělo by být možné si stáhnout dokumenty – například celé znění produktu.
    - Na detailu služby je třeba zjednodušeně odprezentovat produkt, včetně zobrazení základních krytí.
    - Taktéž by zde měla být možnost odeslat notifikaci poradci pro pomoc.

- 10-A Nová událost (Klient)
    - Klient by měl mít možnost vytvořit novou událost.
    - Vytvoření události by mělo být jednoduché, ačkoliv by zde taktéž měla být možnost notifikovat poradce pro případnou pomoc.
    - V rámci události bude vyplněn základní popis – datum, popis situace a případné podklady pro budoucí posouzení.
    - Měla by zde být také informace o případném zažádání doplnění údajů ze strany likvidátora.
    - Události by měli být editovatelné a každá úprava by měla být možná uložit.
- 10-B Vyhodnocení události (Likvidátor)
    - Likvidátor má možnost zobrazit si odeslanou událost.
    - K této události by měl být schopen zažádat o doplňující informace. (Tato funkcionalita by měla zaslat email klientovi)
    - Má možnost schvalovat a zamítat události.
    - Má možnost zobrazit si přiložené dokumenty.
    - K vyrozumění přidává likvidátor svůj komentář.
- 11-A Mé události (Klient)
    - Klient by měl mít možnost navigovat na seznam svých minulých, či přítomných událostí.
    - Měly by se zde objevovat všechny události klienta.
    - Taktéž by měla existovat možnost překliku do detailu existující události.
    - Design detailu by měl zůstat stejný jako při vytváření nové.
    - Hlavní rozdíl by měl být v nemožnosti editovat některá pole, či vizuální změně stavu.
- 11-B Seznam událostí (Likvidátor)
    - Na této obrazovce budou k dispozici všechny události, které k sobě nemá přiřazený žádný likvidátor nebo všechny události, které jsou přiřazeny na daného likvidátora.
    - Lze zobrazit všechny (i proběhlé události, které si likvidátor může zobrazit na detailu.

    - Pokud si chce likvidátor zobrazit nepřevzatou událost, tak se mu po zobrazení přiřadí.
- 12-A Seznam výplat (Administrátor)
    - Seznam výplat představuje seznam všech vyplacených událostí.
    - Tato tabulka slouží pro stopování výplat.
    - Na vyžádání ji lze poskytnout přes administrátora.
- 13-A Log akcí
    - Zde se zapisuje každý úkon v aplikaci.
    - Vždy je daná akce označena časovým označením.
- 13-B Log chyb
    - Log chyb představuje podobnou funkci, jako předchozí pohled.
    - Místo akcí se zde mají vyskytovat chybové hlášky systému.

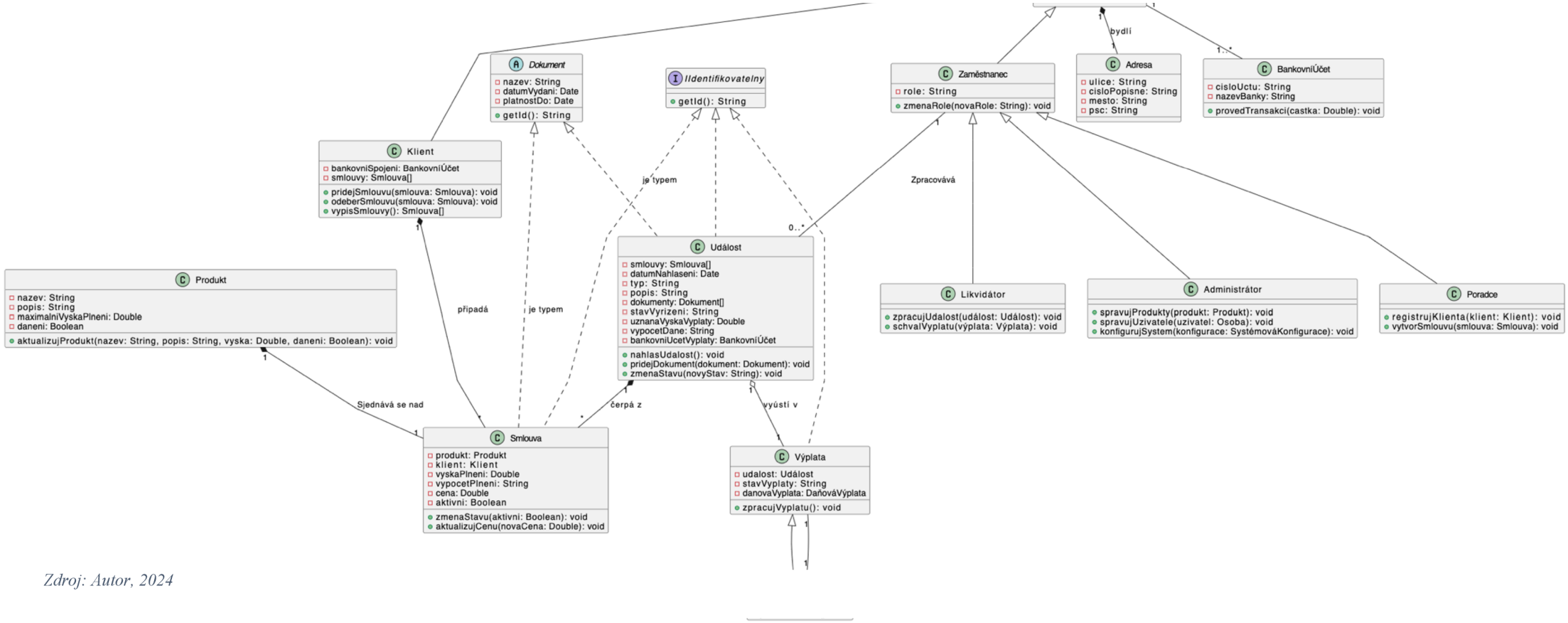
Při tvorbě designu byl brán zřetel na přehlednost, jednoduchost a zadání. Design pravděpodobně nezobrazuje všechny položky, které by se mohly objevit v našem systému a bere v úvahu i „Případy užití“ z kapitoly 4.4.8. Nicméně od klienta nebyla získána zpětná vazba k designu před skončením spolupráce. Celý design je dostupný v přílohách na konci práce, nebo na webu níže: https://www.figma.com/file/RfjRr3qdD05xe0VuJkhwpa/Untitled?type=whiteboard&node-id=0-1&t=8gMWQjTvNTpp5qBZ-0

4.4.3 Model tříd

Na obrázku níže je zobrazen návrh modelu tříd pro nový systém, který projektuje výše zmíněné body. Tento model však zůstává lehce zjednodušeným z důvodu přehlednosti. Při implementaci systému by jistě přibyly nové pomocné třídy. Nicméně při návrhu byly navrženy určité míry abstrakce, které jsou základem čisté architektury dle publikace od (M. Shaw; R. DeLine; G. Zelesnik, 1997).

Tento model vyplývá ze zadání a měl by splňovat potřeby systému. Taktéž byl brán zřetel na momentální problém z kapitoly 4.3.9 - zbytečné komplexnosti systému. Snahou tedy bylo eliminovat nepotřebné třídy, které komplikovali celou stabilitu systému.

Obrázek 18 - ER diagram 2 - Model navrhovaného systému



Zdroj: Autor, 2024

4.4.4 Datový slovník

Osoba (Abstraktní)

Osoba je hlavní třídou našeho systému, avšak se v tomto případě bude jednat pouze o třídu abstraktní. Osoba disponuje celým jménem, telefonním číslem, adresou a datumem narození. Osoba je dále využívána pro entitu klienta a zaměstnance.

Klient : Osoba

Třída klienta představuje osobu, která má u pojišťovny sjednán jeden, či více produktů. Objekt klienta a účet, který klient může využívat je vytvořen po sjednání na pobočce. Tím zajistíme, že klient má přidružený alespoň jednu smlouvu nabízeného produktu.

Produkt se ruší vždy k poslednímu dni v měsíci, proto bude zavedeno pravidlo promazání uživatelů bez produktů každý první den v měsíci. Nejdříve se však bude jednat o soft delete příznakem isDeleted na dobu do následujícího daňujícího období. Tímto způsobem zvýšíme i efektivitu využívání úložiště v databázi a zároveň udržíme potřebná data pro daňové období.

Klient bude kromě objektů smluv, disponovat poděděnými údaji jako je celé jméno, telefonní číslo, adresa. Navíc bude taktéž disponovat bankovním spojením, které bude následně využíváno pro výplaty plnění.

Zaměstnanec : Osoba

Třída zaměstnance dědí z abstraktní třídy osob, přičemž kromě poděděných atributů bude mít důležitý atribut role. Ten je představen kolekcí enumů většího počtu rolí, které spolu vzájemně nekolidují a poskytují vyšší granulitu než v původním systému.

Zaměstnanec je dále rozdělen podle specializace. Tato šňůra dědění zajišťuje případné propisování změn a dodržuje pravidla čistého designu.

Produkt

Produkt představuje entitu, která je využívána v entitě klienta. Samotný produkt v našem případě bude mít atributy pro název, popis produktu a maximální výši plnění.

Taktéž bude disponovat pro zjednodušení názvosloví atributem booleanu danění, které bude nabývat hodnot ano nebo ne.

Smlouva

Třída smlouvy představuje logický most mezi produktem a klientem. Jedná se o produkt, který klient využívá a je pro daného klienta přidružen. Samotná smlouva odkazuje na objekty produktu a klienta, nad kterými je uzavřena, nastavenou výši plnění, formuli pro výpočet plnění, cenu a informaci, zda je smlouva aktivní.

Aktivita smlouvy je řízena buď tím, zda nebyla smlouva ze strany klienta zrušena, nebo nebyla uhrazena měsíční částka. Tato kontrola by byla prováděna dotazem na datový sklad společnosti, kde jsou ukládána tato data.

Událost

Klient bude mít možnost nahlašovat událost, tato událost představuje druhé rozšíření původního systému po nabízení produktů. Díky tomuto kroku totiž původní systém rozšíříme o samoobslužný servis pro klienty.

Události bude náležet kolekce objektů smluv, nad kterými chceme událost nahlašovat, datum nahlášení události, typ události, popis události, dodatečné dokumenty, stav vyřízení události, uznanou výši k výplatě a výpočet daně – tato část je provedena až po změně stavu události na „Uznáno“. Mimo již zmíněné atributy bude zde taktéž atribut pro jiný bankovní účet pro provedení výplaty. Události posuzuje osoba, která je zaměstnancem v roli likvidátor.

Výplata

Původní entity transakcí a plateb jsem sjednotil do jedné vzhledem ke změně povahy a struktury sytému. Tímto způsobem bude odstraněn nejeden problém původního systému, které zapříčinili nejen problematický workflow, ale zvýšenou chybovost.

Tato třída bude obsahovat objekt události, stav výplaty a ID daňové výplaty, pokud se jednalo o zdanitelnou výplatu. Tím, že zde používáme celé objekty událostí, tak jsme si usnadnili přístup k informacím a zároveň ponechali funkcionalitu.

Výplaty procházejí stejným způsobem zpracování jako v původním systému, takže uživatelé, kteří jsou zaměstnanci v roli operátor provádí zpracování výplat a vedoucí výplaty nad stanovenou hranici v konfiguraci tyto výplaty schvalují.

Daňová výplata

Jedná se o entitu, která vzniká u zdanitelných produktů po uznání události současně s výplatou pro klienta. Jedná se o efektivnější workflow, který odstraňuje potenciální chybovost původního systému.

4.4.5 Nové role

Jak již bylo zmíněno ve čtvrté kapitole, změny se budou taktéž týkat práci s rolemi, kvůli kolizím. Tato situace je řešena revizí a změně hierarchie rolí, které se zaměřují na jednotlivé funkce, a ne na logické celky jako to bylo v původním systému.

Vzhledem k tomu, že nový systém není tak komplexní, jak by se mohlo zdát, tak je toto řešení zcela vyhovující a zároveň v dostatečné míře přehledné.

Role jsou odvozené od abstraktní třídy Osoba a následně zaměstnanec. Týká se tedy pouze zaměstnanců a jsou následující:

a.) Likvidátor – Provádí všechny úkony, které doteď prováděl tzv. operátor

b.) Administrátor – Provádí administraci nad aplikací

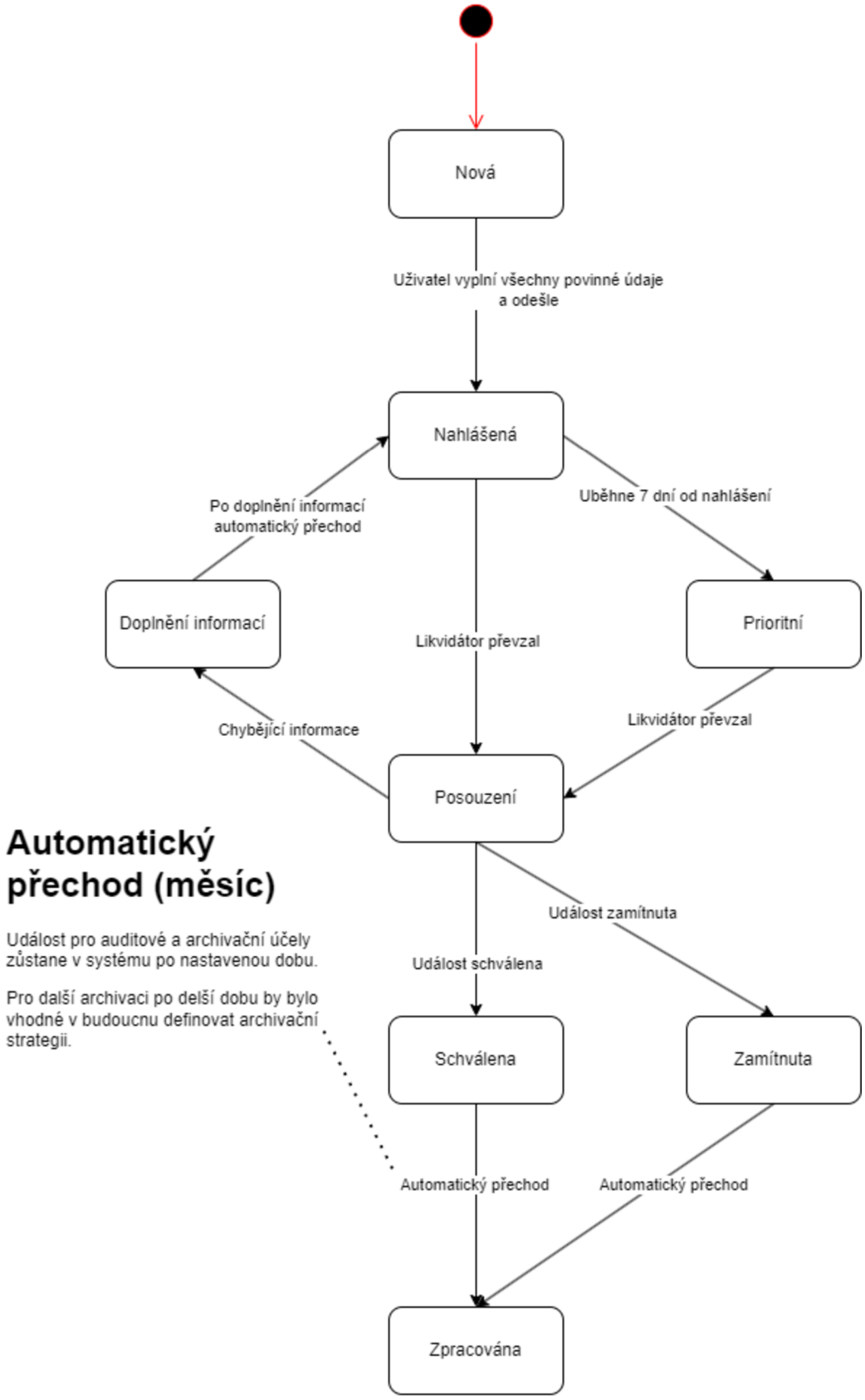
c.) Poradce – Provádí akce nad uživatelem, které se týkají registrace a správy smluv.

4.4.6 Stavový model

Navržený systém popisuje třídy, které jsou nezbytné pro naplnění požadavků na systém. Taktéž velmi zjednodušuje workflow a nepracuje se složitými stavy. Kromě složitých stavů využívá předávání zodpovědností při ukončení. Pro příklad je níže zobrazen jednoduchý stavový diagram Události.

Z diagramu níže je zřejmé, že zde není žádná výhybka a workflow je velmi jednoduché. Co však zřejmé z diagramu není, je to, že před uzavřením bude likvidátor vyhodnocovat, zda bude uznána či nikoliv. Pokud bude vybrána možnost uznání, pak se vytvoří entita platby, která bude zpracována mimo samotnou třídu událostí.

Obrázek 19 - Stavový model 2 - Stavový model událostí navrhnovaného systému



Zdroj: Autor, 2024

Pokud tedy dojde k vytvoření výplaty, událost dojde do stavu uzavřená. Stejný scénář by nastal v situaci, kdy bude událost zamítnutá. Tato situace je většinou řešena s likvidátorem na přímo komunikací se zákazníkem.

Tímto přístupem dodržování solid principů zjednodušíme celý systém a zamezíme výskytu chyb, nekonečných cyklů a neočekávaných problémů. Taktéž je nutné dodat, že schvalování je tedy prováděno nad třídou událost, což je drobná změna oproti původnímu systému a následná výplata popisuje budoucí platbu.

4.4.7 Technický návrh

Technický návrh je vytvořen na základě literární části práce a taktéž na základě znalosti prostředí klienta, který upřednostňovala služby založené na produktech od společnosti Microsoft, primárně kvůli partnerství a internímu know-how. Další podmínkou je také zjednodušený návrh autentizace z důvodu zachování interního know-how, to by si pojišťovna v případě použití nasadila sama.

Jediná podmínka, která tedy ovlivňuje samotné řešení je vyhledání a použití produktů, které vyhovují danému požadavku. Na základě tohoto zjištění byl návrh uskutečněn.

Touto podmínkou jsme tedy omezeni na nástroje, které jsou dle literární část vhodné pro korporátní prostředí a jedná se o velmi kvalitní nástroje, které jsou vhodné na tvorbu projektu tohoto typu vzhledem k modernosti a udržitelnosti řešení. V rámci technologického stacku je navrženo použít následující.

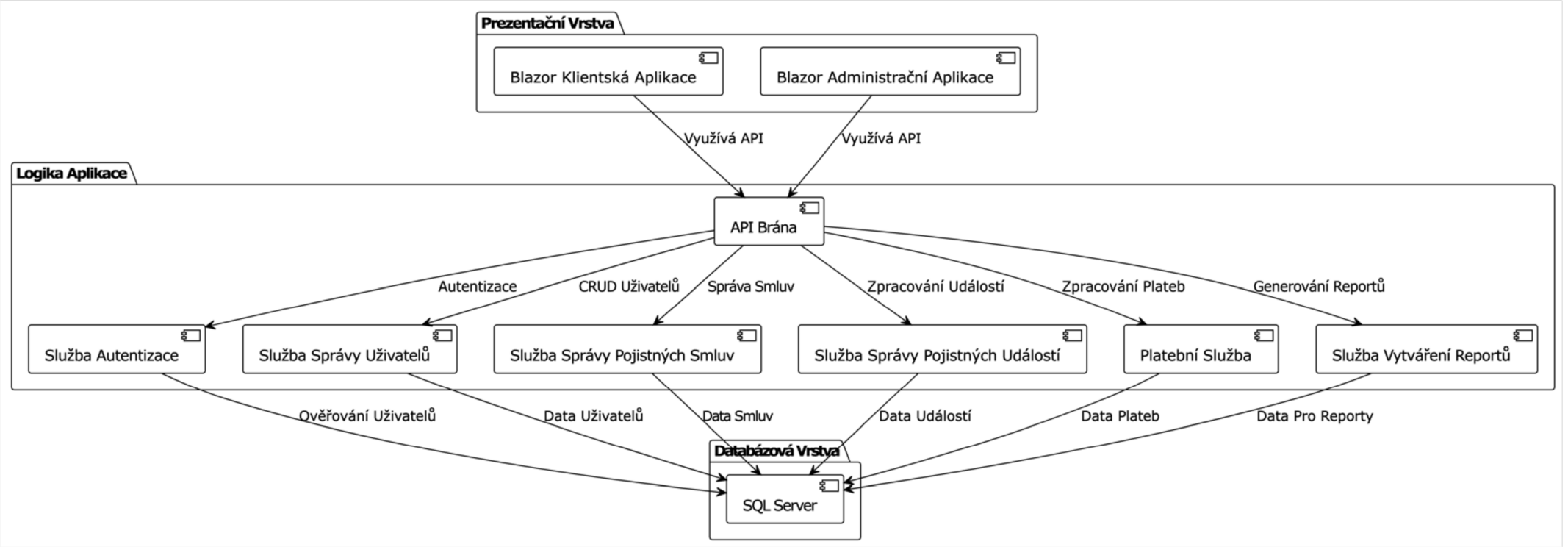
- Programovací jazyk a framework: C# s .NET 8 pro vývoj backendu. .NET 8 byl zvolen pro jeho výkon, bezpečnostní vlastnosti a podporu cross-platform vývoje.
- Databáze: Microsoft SQL Server pro správu relačních dat s využitím Entity Framework Core pro ORM, což usnadní manipulaci s daty.
- Front end: Blazor pro vytváření interaktivních uživatelských rozhraní s využitím C# a .NET, což umožní sdílet logiku mezi serverem a klientem a minimalizovat použití Javascriptu.
- API a Middleware: ASP.NET Core Web API pro vývoj RESTful API, které bude sloužit jako most mezi front endem a backendem.

- Identita a Bezpečnost: ASP.NET Core Identity pro autentizaci a autorizaci, doplněné o OAuth 2.0 a OpenID Connect pro integraci s externími poskytovateli identity. (Vzhledem k zadání se návrh drží tzv. best practices, dle nástrojů, které jsou dokumentovány na webu curity.io)
- Cloud a Hosting: Azure App Services pro hosting webových aplikací a Azure SQL Database pro databáze, zajišťující vysokou dostupnost a bezpečnost.
- DevOps a CI/CD: Azure DevOps pro automatizaci buildů, testování a nasazování s využitím Git pro verzování kódu.

Návrh architektury zahrnuje třívrstvou strukturu: prezentace, business logika a datová vrstva, doplněnou o API vrstvu pro komunikaci mezi klientem a serverem.

- Prezentační Vrstva
  - Blazor Client App: Zajišťuje interakci s uživatelem a prezentaci dat.
- Business Logika
  - NET 8.0 Services: Obsahuje veškerou logiku aplikace, zpracování pravidel a manipulaci s daty.
- Datová Vrstva
  - MSSQL Database: Ukládá data a je přístupná přes Entity Framework Core
- API Vrstva
  - ASP.NET Core Web API: Propojuje front end s business logikou, umožňuje CRUD operace a integraci s externími systémy.

Obrázek 20 - Navržená architektura systému



Zdroj: Autor

Pro umožnění budoucího růstu dat a zabezpečení budou využívány tyto služby.

- Azure Load Balancer: Pro distribuci zátěže a zajištění vysoké dostupnosti.
- Azure SQL Database: Nabízí automatické škálování a zálohování.
- HTTPS: Pro šifrování komunikace.
- Azure Active Directory & Identity Server 4: Pro správu uživatelů a bezpečný přístup.

4.4.8 Use case scénáře systému

V rámci této kapitoly jsou popsány případy užití navrhovaného systému. Tyto případy užití by měly reprezentovat myšlenku systému jako celku.

Tyto USE cases odrážejí funkčnost a jednoduchost systému při dodržení požadavků, které byly na počátku projektu definovány.

Tabulka 1 – Seznam Případů užití

UC	Název use case	Popis
UC1	Přihlášení uživatele	Umožňuje uživatelům přihlásit se do systému.
UC2	Registrace klienta	Poradce na pobočce registruje nového klienta do systému.
UC3	Vytvoření pojistné smlouvy	Umožňuje klientům nebo poradcům vytvořit a podepsat pojistnou smlouvu.
UC4	Hlášení pojistné události klientem	Klienti mohou hlásit pojistné události prostřednictvím samoobslužného rozhraní.
UC5	Správa pojistných událostí	Likvidátor zpracovává a vyhodnocuje nahlášené pojistné události.
UC6	Výplata pojistného plnění	Systém zpracovává a provádí výplaty schválených pojistných plnění.
UC7	Správa uživatelských účtů	Administrátoři spravují uživatelské účty v systému.
UC8	Správa pojistných produktů	Administrátoři mohou přidávat, upravovat nebo mazat pojistné produkty v systému.
UC9	Monitorování a reportování	Systém poskytuje reporty a monitorovací nástroje pro sledování aktivity a výkonu.
UC10	Nahlásit událost	Klienti nahlašují události, které se následně zpracovávají
UC11	Mé události	Klienti si mohou prohlížet jejich události.
UC12	Seznam výplat	Administrátoři mohou sledovat průběh plateb pro případné řešení ticketů.

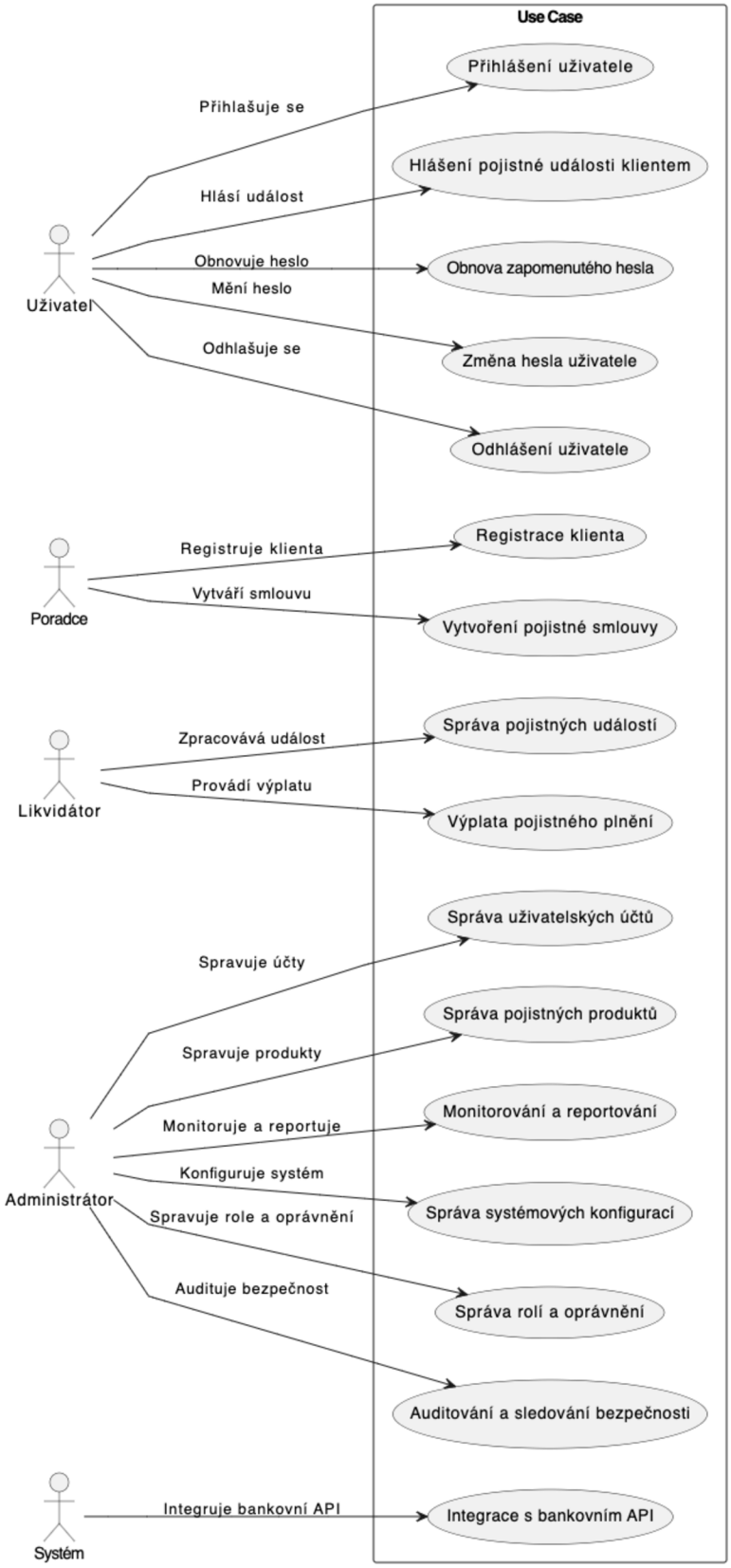
UC	Název use case	Popis
UC13	Log akcí a log chyb	Administrátoři mohou vyhledávat a sledovat aktivity uživatelů a systému.
UC14	Integrace s bankovním API	Systém se integruje s bankovními API pro zpracování transakcí.
UC15	Správa rolí a oprávnění	Administrátoři přidělují a spravují role a oprávnění uživatelů v systému.
UC16	Auditování a sledování bezpečnosti	Systém zaznamenává a audituje bezpečnostní události a přístupy.

Zdroj: Autor, 2023

Níže jsou detailněji rozepsány tři nejzásadnější případy užití tvořeného systému doplněné o diagramy aktivit. Zbytek případů užití je dostupný na konci práce v druhé sekci příloh pod designem.

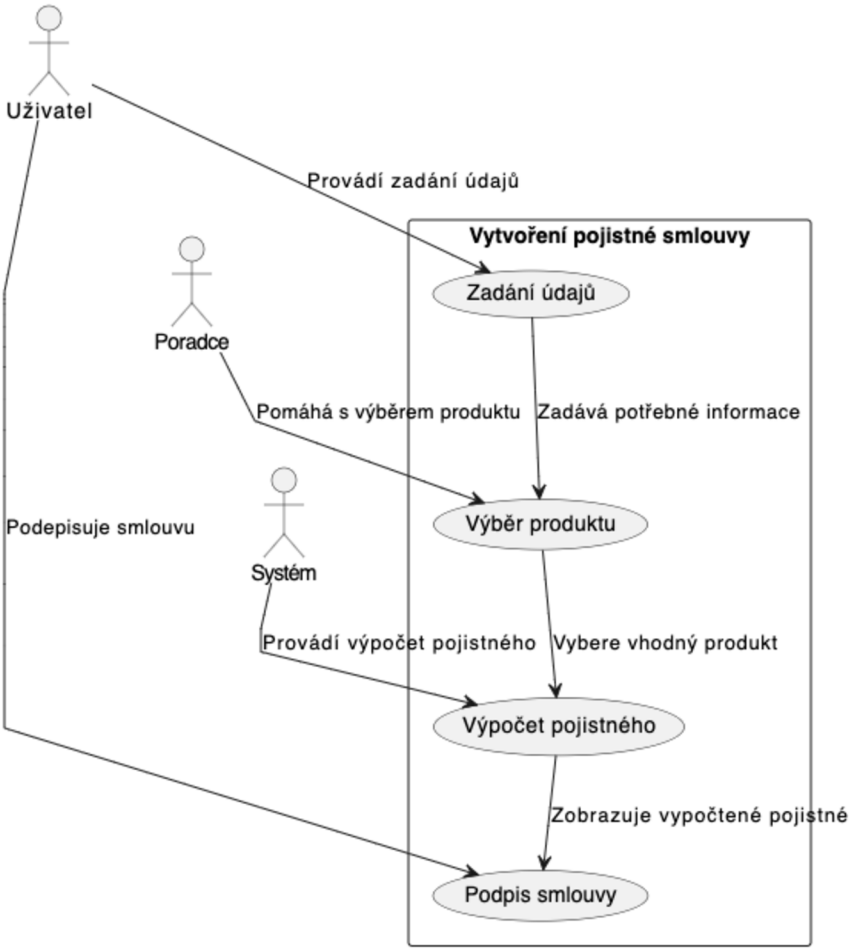
Případy užití a design by měl být blízký k naplnění. Nicméně některé detaily návrhu by měly být při případné budoucí implementaci vyjasněny s potenciálním zákazníkem.

Obrázek 21 - Use case diagram navrhovaného systému



Zdroj: Autor, 2023

Obrázek 22 - Dekompozice UC - vytvoření pojistné smlouvy



Zdroj: Autor, 2023

UC2 – Registrace klienta do systému

Aktéři

- Poradce: Zaměstnanec pojišťovny, který je odpovědný za registraci nových klientů.
- Klient: Osoba, která se stává klientem pojišťovny.
- Systém

Podmínky před spuštěním

- Poradce je přihlášen do systému na pobočce.
- Klient je fyzicky přítomen na pobočce a má připraveny všechny potřebné dokumenty a informace pro registraci.

Základní tok

1. Klient obdrží smlouvu o poskytování služeb a souhlas s ochranou osobních údajů k podpisu.
2. Po podpisu dokumentů systém uloží údaje a smlouvy a aktivuje uživatelský účet.
3. Poradce vybere v systému možnost pro přidání nového klienta.
4. Klient poskytne všechny potřebné informace (jméno, adresa, datum narození, kontaktní údaje atd.).
5. Poradce vloží informace do systému.
6. Systém ověří unikátnost a validitu zadaných údajů.
7. Pokud jsou údaje správné a unikátní, systém vytvoří nový uživatelský účet.
8. Systém vygeneruje uživatelské jméno a heslo pro klienta.
9. Poradce informuje klienta o podrobnostech účtu a provádí instruktáž k používání samoobslužného rozhraní.
10. Klient je informován o úspěšné registraci a možnostech následného používání služeb pojišťovny.

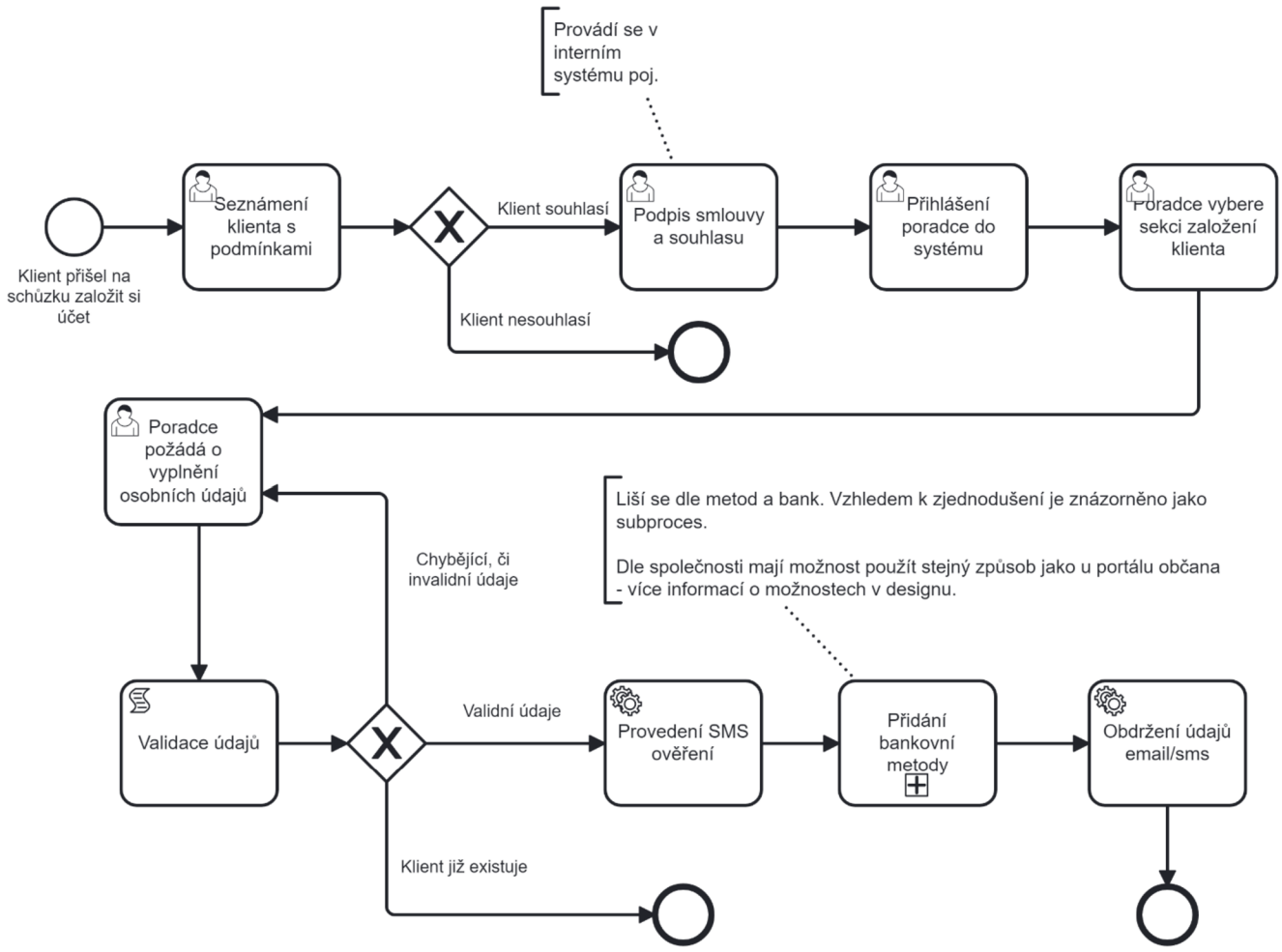
Alternativní tok bodu 4

1. Pokud systém identifikuje problém s údaji (duplicita, neplatnost atd.), poradce je vyzve k opravě.
2. Klient nebo poradce opraví údaje. Tok pokračuje bodem č. 5.

Podmínky po spuštění

- Klient má aktivní uživatelský účet v systému pojišťovny.
- Klient má podrobné informace o tom, jak přistupovat k účtu a jak využívat služby pojišťovny.
- Všechny relevantní dokumenty jsou podepsány a uloženy jak v papírové, tak v elektronické formě.

Obrázek 23 - BPMN diagram 2 - UC2 – proces registrace klienta do systému



Zdroj: Autor, 2024

UC3 – Vytvoření pojistné smlouvy

Aktéři

- Klient: Osoba, která chce uzavřít pojistnou smlouvu.
- Systém
- Databáze

Podmínky před spuštěním

- Klient musí být zaregistrován a přihlášen do systému.
- Systém musí být dostupný a funkční.
- Klient musí mít veškeré potřebné informace a dokumenty potřebné pro uzavření smlouvy.

Základní tok

1. Poradce se přihlásí do systému.
2. Poradce s klientem zvolí typ pojištění.
3. Poradce seznámí klienta s podmínkami produktu.
4. Poradce s klientem vyplní potřebné údaje pro smlouvu.
5. Pokud jsou data validní, systém vypočítá cenu pojištění.
6. Klient potvrdí detaily smlouvy a akceptuje cenu.
7. Klient potvrdí identitu přes SMS ověření.
8. Klient elektronicky podepíše pojistnou smlouvu.
9. Systém uloží smlouvu do databáze smluv.
10. Klient obdrží potvrzení o úspěšném uzavření smlouvy.

Alternativní tok bodu 3-A

1. Klient nesouhlasí s podmínkami smlouvy.
2. Klient si chce si vybrat jiný produkt. Tok se vrací do bodu č. 2.

Alternativní tok bodu 3-B

1. Klient nesouhlasí s podmínkami smlouvy.
2. Klient nechce pokračovat. Tok končí.

Alternativní tok bodu 6-A

1. Pokud klient nesouhlasí s cenou nebo detaily smlouvy, může požádat o revizi nebo změny.
2. Systém umožní klientovi provést změny a aktualizovat nabídku. Tok pokračuje bodem č. 5.

Alternativní tok bodu 6-B

1. Pokud klient nesouhlasí s cenou nebo detaily smlouvy, chce jiný produkt.
2. Systém umožní vybrat jiný produkt. Tok se vrací do bodu č. 2.

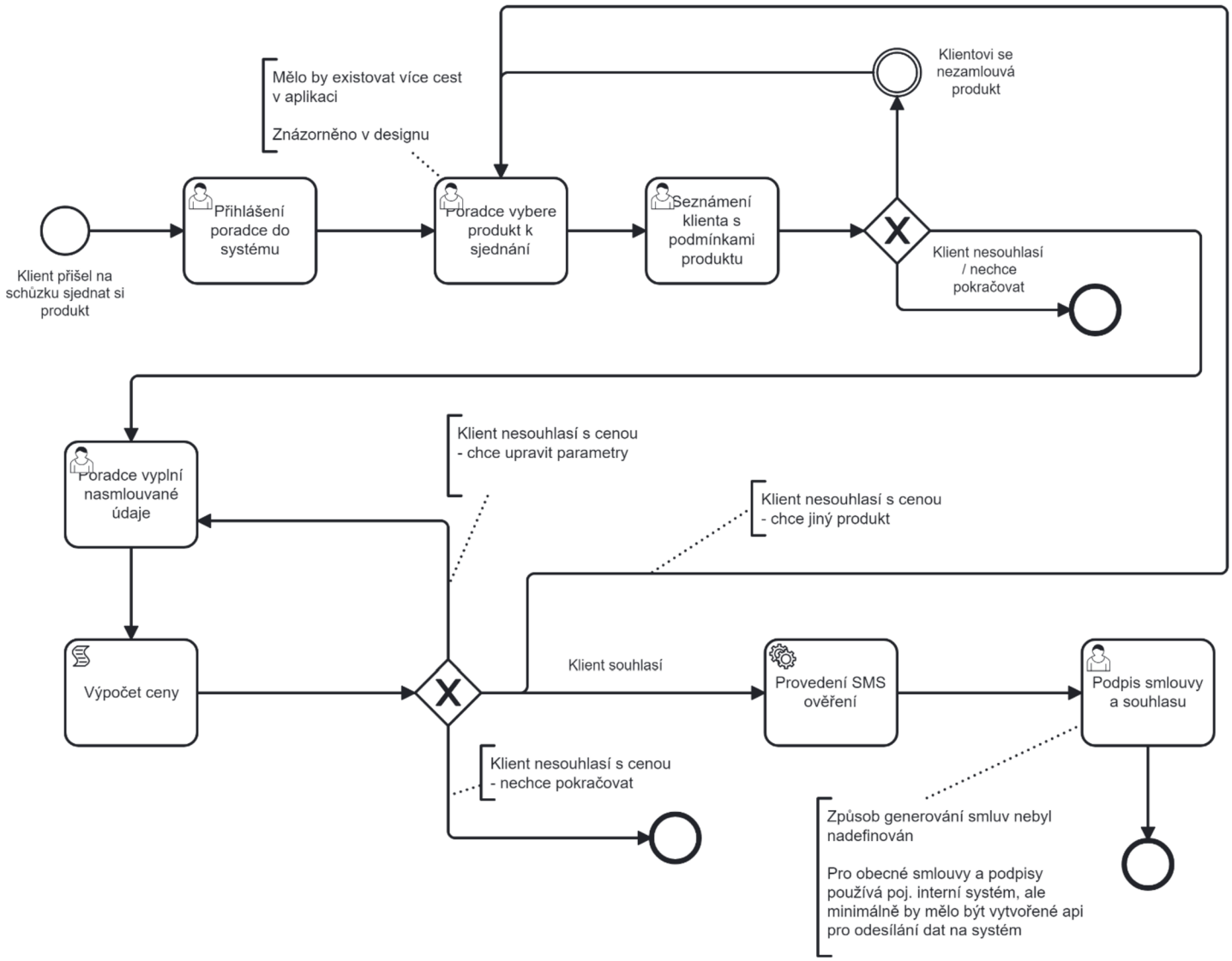
Alternativní tok bodu 6-C

1. Pokud klient nesouhlasí s cenou nebo detaily smlouvy.
2. Klient nechce pokračovat. Tok končí.

Podmínky po spuštění

- Pojistná smlouva je aktivní a uložena v systému.
- Klient má přístup k digitální kopii pojistné smlouvy.
- Systém má zaznamenanou veškerou komunikaci a dokumentaci související s procesem uzavření smlouvy.

Obrázek 24 - BPMN diagram 3 - UC3 Proces Vytvoření pojistné smlouvy



Zdroj: Autor, 2024

UC4 - Hlášení pojistné události klientem

Aktéři

- Klient
- Systém

Podmínky před spuštěním

- Klient je přihlášen a má platnou pojistnou smlouvu.

Základní tok

1. Klient zvolí v uživatelském rozhraní možnost "Nahlásit událost".
2. Vyplní formulář s detaily pojistné události, včetně popisu, data a přiložení případných dokladů.
3. Odešle formulář ke zpracování.
4. Systém potvrdí přijetí hlášení a informuje klienta o dalším postupu.
5. Alternativní tok:
6. Pokud klient nemůže poskytnout všechny potřebné informace, systém mu umožní uložit hlášení jako koncept a dokončit ho později.

Podmínky po dokončení

- Pojistná událost je zaznamenána v systému a připravena k dalšímu zpracování.

UC6 – Výplata pojistného plnění

Aktéři

- Klient
- Likvidátor (zaměstnanec pojišťovny odpovědný za vyřizování pojistných událostí)
- Systém
- Databáze
- Bankovní API

Podmínky před spuštěním

- Klient je přihlášen do samoobslužného rozhraní.
- Likvidátor je přihlášen do systému.
- Smlouva a pojistná událost jsou již zaznamenány v systému.
- Pojistná událost byla likvidátorem uznána k plnění.

Základní tok

1. Likvidátor získá seznam schválených výplat pro zpracování v systému.
2. Likvidátor vybere pojistnou událost ze seznamu pro provedení výplaty.
3. Systém zobrazí detaily události a spojené smlouvy pro kontrolu.
4. Likvidátor zadá výši pojistného plnění podle podmínek smlouvy a události.
5. Systém provede výpočet daně z pojistného plnění (pokud je zdanitelné).
6. Systém generuje návrh transakce pro výplatu a související daňovou výplatu.
7. Likvidátor schválí transakci.
8. Systém zašle pokyn k výplatě přes bankovní API.
9. Bankovní API potvrdí provedení transakce.
10. Systém zaznamená provedení výplaty a aktualizuje stav události.
11. Systém odesílá notifikaci klientovi o výplatě plnění.

Alternativní tok bodu 4

1. Pokud výše plnění není jednoznačná, likvidátor žádá o další dokumentaci nebo informace.
2. Klient poskytne požadované informace. Tok pokračuje bodem č. 5.

Alternativní tok bodu 7

1. Pokud likvidátor neschválí transakci, systém vyzve k doplnění informací nebo k opravě výpočtu.
2. Likvidátor opraví informace a znovu schválí transakci. Tok pokračuje bodem č. 8.

Alternativní tok bodu 9

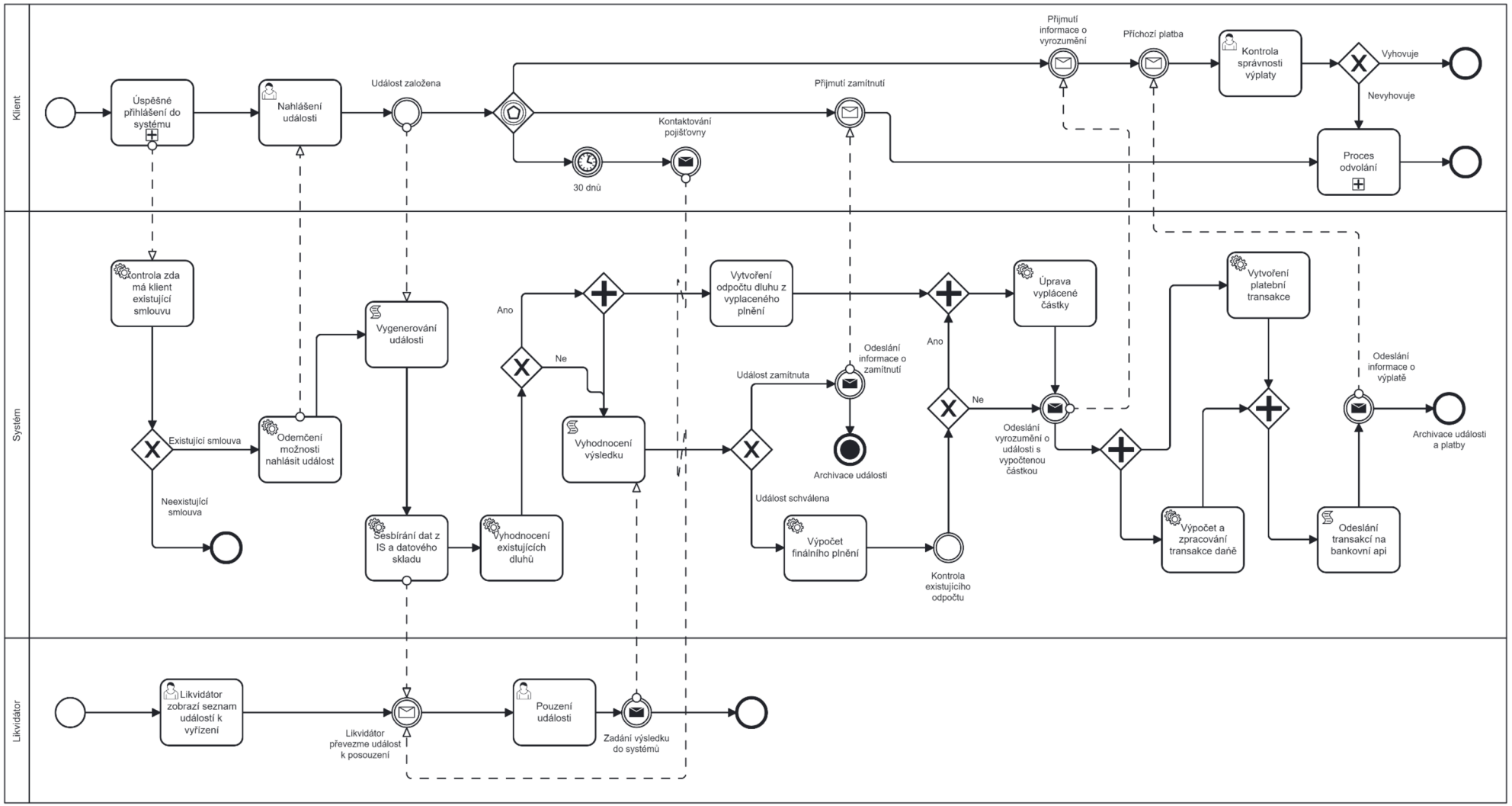
1. Pokud bankovní API nevrátí potvrzení, systém zkusí transakci odeslat znovu.
2. Pokud opětovné odeslání selže, likvidátor je upozorněn na potíže s transakcí a kontaktuje banku.

Podmínky po spuštění

- Výplata byla klientovi úspěšně zaslána.
- Stav události je aktualizován na "Vyplaceno".
- Klient byl informován o výplatě.

Na diagramu níže je možné vidět proces výplaty od nahlášení po vyplacení události. Tento diagram by měl demonstrovat změny worflow z původního systému na nový koncept. I přes určitou abstrakci je zjevné, že výrazně došlo k zjednodušení procesu, který by měl zajišťovat pohodlnost klientům a zaměstnancům pojišťovny. Diagram se týká jak UC 4, tak i UC6 a je spojen právě pro možnost porovnání.

Obrázek 25 - BPMN diagram 4 – UC4 + UC6 - Proces kompletního hlavního proudu aplikace (Od nahlášení – po vyplacení)



Zdroj: Autor, 2024

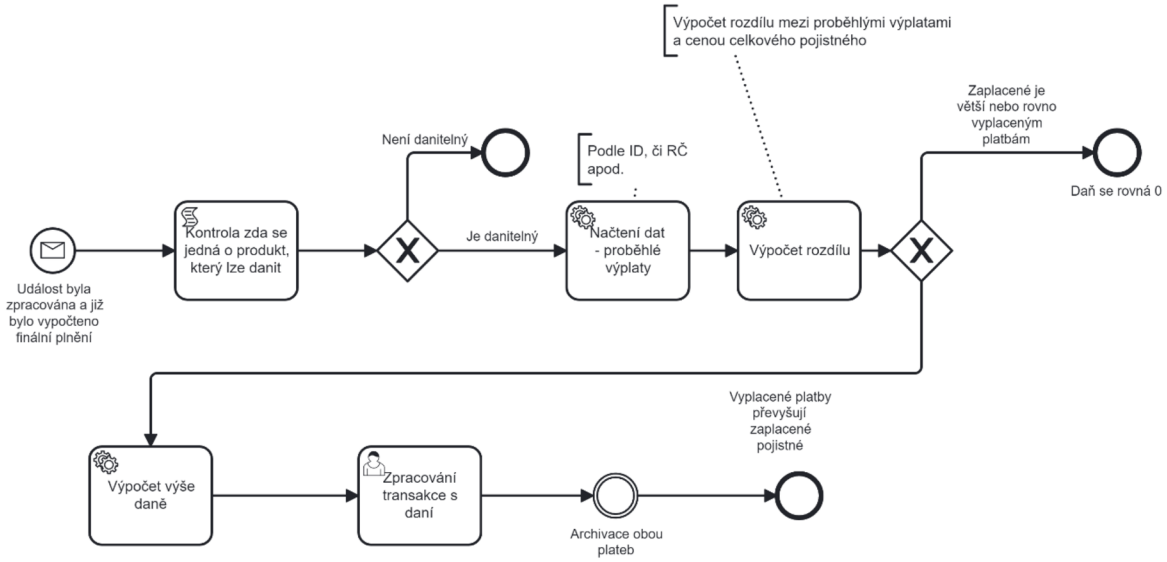
4.4.9 Návrh nového daňového procesu

Posledním úkolem, který je třeba vyřešit aby byly splněny akceptační kritéria je popsání fungování nového daňového modulu v navrhovaném systému. Ten byl nastaven tak, aby jej pracovníci nemuseli řešit a proces probíhal v pozadí.

Vzhledem ke změně přístupu při navrhování tohoto systému se jedná o jednoduchý proces, který se vyvaruje původní příčině chybovosti a zpracovává všechny informace vně systému. Systém má přístup k ceně pojistného i k historickým platbám. Je tedy nyní jednoduché po schválení platby takovou daň vypočítat bez zbytečných „Edge casů".

Nový proces je znázorněn v následujícím BPMN diagramu. Je také viditelné, že jsem ponechal základní myšlenku původního daňového modulu, která nebyla problematická a je znovupoužitelná.

Obrázek 26 - BPMN Diagram - Proces danění nového systému



Zdroj: Autor

5 Hodnocení výsledků

Tato práce byla zaměřena na návrh informačního systému, který původně měl být realizován. Během vypracování práce však bohužel došlo ke přerušení spolupráce mezi společností ve které byla práce tvořena a klientem (pojišťovnou), který se rozhodl pokračovat jinou cestou. Práce však tedy pokračovala s udržením zadání a myšlenky napravit chyby původního řešení.

Nejedná se tedy o hodnocení existujícího systému, ale zhodnocení řešení, které bylo navrženo oproti chování stávajícího systému.

5.1 Hodnocení analýzy původního systému

Původní systém byl testován skupinou cílových uživatelů, kteří jsou zaměstnanci klienta a odvádí práci s tímto systémem na denní bázi. Taktéž bylo prováděno osobní testování a analýza, při které byl sestaven a popsán pohled na systém ke kterému chyběla detailní dokumentace. Tato dokumentace byla sestavena v kapitole analýzy původního systému a akceptována klientem.

Kromě textové dokumentace byla popsána struktura systému zjednodušeným modelem tříd bez metod, který byl sestaven z migračního skriptu databáze. Taktéž byl vykreslen problematický proces výplaty, danění, a následně diagram změn stavů transakcí a plateb.

Dále byly sesbírány problémy, se kterými se klíčoví uživatelé potkávali a byly označené za prioritní k řešení. Tyto problémy byly následně přeformulovány pro potřeby tyto práce do formy textů z původních odrážkových bodů. Jednalo se o výstupní zadání vyřešit dané problémy v novém systému.

5.2 Hodnocení návrhu informačního systému

Návrh informačního systému byl založen na myšlence původního systému, který měl být nahrazen novou multifunkční platformou. Návrh odrážel problematické chování původního systému, kde se vyskytovaly primárně procesní problémy, které nebyly nijak specificky nadefinovány před tvorbou práce a bylo třeba pro ně navrhnout nové procesy toky a řešení.

V rámci návrhu byl vytvořen model tříd, který popisuje strukturu nového systému a lze jej porovnat s původním systémem. Byly vytvořeny uživatelské scénáře na základě zadání jako důležitý výstup pro případnou budoucí implementaci. Taktéž byly pro nejpodstatnější uživatelské scénáře popsány BPMN diagramy, které byly zvoleny kvůli komplexitě procesů namísto diagramů aktivit. Tento přístup přinesl jednoznačnost, čistotu a možnost porovnání s původním systémem.

K navrženému řešení byl také navržen diagram architektury včetně navržených technologií, které odpovídají zadání klienta se zaměřením na technologie od společnosti Microsoft. Navíc nad rámec technického návrhu byly taktéž navrženy pohledy pro nový systém, který byl tvořen tak, aby splňoval scénáře případů užití.

Ověření splnění všech požadavků bylo provedeno pomocí mapování případů užití na požadavky, které vyplynuly z analýzy a navrhovaného designu aplikace.

5.3 Budoucí vývoj

Jak již bylo zmíněno, tak bohužel došlo k ukončení spolupráce před dokončením samotné práce. Nicméně v případě, že by byl návrh použit, tak by bylo již jen třeba daný systém naimplementovat a dodefinovat sekundární uživatelské cíle systému, které by mohly být například v oblasti nastavení aplikace, či personalizace UI.

Systém byl navržen, tak aby splňoval procesy a chování dané pojišťovny. Věřím však, že by byl znovupoužitelný pro jiného klienta v dané oblasti.

5.4 Porovnání s aktuálním řešením

Motivace této práce vycházela právě z nefunkčního řešení klienta, které způsobovalo spoustu problémů a manuálních úkonů. Věřím tedy, že navržené řešení odpovídá vyšším standardům než to původní.

Nicméně se klient rozhodl pokračovat ve vývoji původního systému u jiného dodavatele cca v září roku 2023. Tento dodavatel však daný systém do pořádku stále nedal a věřím, že navrhované řešení by bylo jednodušší, multifunční a levnější řešení pro zkoumaného klienta.

6 Závěr

Diplomová práce "Návrh Pojišťovnického systému" představuje důkladné zkoumání a inovativní přístup k řešení problémů stávajícího pojišťovnického informačního systému prostřednictvím návrhu, vývoje a implementace nového, efektivnějšího a technologicky pokročilého systému.

Výsledky této diplomové práce představují významný krok vpřed v oblasti pojišťovnických informačních systémů v dané společnosti. Nově navržený systém adresně řeší identifikované nedostatky stávajícího systému a přináší řadu zlepšení:

1. Zvýšená efektivita a produktivita: Automatizace rutinních úkolů a zlepšené procesní řízení v novém systému značně zvyšují efektivitu a produktivitu pracovníků, což vede k rychlejšímu a přehlednějšímu zpracování pojistných událostí.
2. Vylepšená uživatelská přívětivost: S ohledem na moderní uživatelské rozhraní a intuitivní navigaci poskytuje nový systém výrazně lepší uživatelskou zkušenost jak pro zaměstnance pojišťovny, tak pro její klienty.
3. Flexibilita a škálovatelnost: Díky moderní architecture, jednoduchosti a použití aktuálních technologií byl nový systém dobře připraven na budoucí rozšiřování a integraci s dalšími technologiemi a systémy.

V rámci diplomové práce bylo primárním cílem navrhnout a hodnotit informační systém pro pojišťovnický sektor, který by přinášel významná zlepšení oproti stávajícímu systému. Hlavním směrem bylo zaměření na automatizaci procesů, zlepšení uživatelské přívětivosti, zvýšení bezpečnosti a spolehlivosti, a nabídnout flexibilitu a škálovatelnost pro budoucí rozvoj.

Analýza původního systému ukázala, že existují významné nedostatky ve způsobu, jakým jsou informace a procesy spravovány. Vytvořená dokumentace a modely tříd ukazují na potřebu zásadní restrukturalizace a zlepšení funkcionalit. Navrhovaný systém odráží poznatky získané z této analýzy a přináší inovativní řešení, které slibuje výrazné zlepšení oproti původnímu stavu.

Návrh informačního systému je postaven na robustních technologických základech a předpokládá využití moderních softwarových principů a praxí. Zahrnuje detailně propracované uživatelské scénáře a procesní modely, které zajišťují, že systém bude

splňovat všechny požadavky a očekávání. Integrace s technologiemi od společnosti Microsoft dále zvyšuje důvěru v stabilitu a budoucí podporu systému.

Navzdory nepředvídaným obtížím v průběhu spolupráce s původním klientem bylo možné práci úspěšně dokončit a předložit návrh, který přináší nové možnosti pro danou společnost. Věřím, že prezentované řešení je nejen vyššího standardu než stávající systém, ale také nabízí platformu pro další rozvoj a adaptaci na budoucí potřeby.

Budoucí vývoj systému by měl zahrnovat průběžné testování a validaci s reálnými uživateli, aby se zajistilo, že všechny procesy a funkce splňují očekávání a požadavky trhu. Důraz by měl být kladen také na rozšíření funkcionalit, integraci s dalšími technologiemi a platformami, a průzkum nových oblastí, kde by systém mohl přinést další hodnotu.

Závěrem lze konstatovat, že prezentovaný návrh informačního systému představuje významný krok vpřed ve vývoji a implementaci pokročilých řešení ve společnosti. Práce poskytuje solidní základ pro budoucí vývoj a inovace v tomto dynamickém a neustále se vyvíjejícím odvětví. S nadějí očekávám, že tyto myšlenky a návrhy budou sloužit jako inspirace pro další rozvoj a zlepšení, které povedou k efektivnějšímu, bezpečnějšímu a uživatelsky přívětivějšímu prostředí v případě návratu klienta.

7 Seznam použitých zdrojů

Adacta, 2024. Navigating Digital Transformation in Commercial Insurance [online]. [cit. 25-8-2023]. Dostupné z: https://blog.adacta-fintech.com/digital-transformation-underwriting-workbench

Agaraval N, 2024. A Framework of Intelligent Process Automation in Insurance [Online]. [cit. 10-1-2024]. Dostupné z: https://www.leadsquared.com/industries/insurance/intelligent-process-automation-in-insurance-industry/.

apollotechnical.com, 2022. Why object-oriented programming matters [online]. [cit. 25-8-2023]. Dostupné z: https://www.apollotechnical.com/why-object-oriented-programming-matters/

Aptien.com, 2024. Risk management [online]. [cit. 10-1-2024]. Dostupné z: https://aptien.com/en/kb/articles/risk-management

Barlow, Mike. Evolving Architectures of FinTech. Sebastopol : O'Reilly, 2016. ISBN 9781491967768.

Booch G., Rumbaugh J., Jacobson I., 2005. Unified Modeling Language User Guide, The (2 ed.). Boston: Addison-Wesley. ISBN 8024745933

Boost.ai, 2023. The state of the insurance industry in 2023 [online]. [cit. 15-12-2023]. Dostupné z: https://boost.ai/blog/insurance-automation/

Boyles C., 2023. Object Oriented Programming: Concepts, Benefits, and Best OOP Language [online]. [cit. 25-8-2023]. Dostupné z: https://uniquesoftwaredev.com/object-oriented-programming-concepts-benefits-and-best-oop-languages/

Cocca G., 2022. Programming Paradigms – Paradigm Examples for Beginners [online]. [cit. 25-8-2023]. Dostupné z: https://www.freecodecamp.org/news/an-introduction-to-programming-paradigms/

codecademy.com, 2023. What Is Object-Oriented Programming? [online]. [cit. 25-8-2023]. Dostupné z: https://www.codecademy.com/resources/blog/object-oriented-programming/

curity.io, 2024. OpenID Connect Client with .NET [online]. [cit. 25-8-2023]. Dostupné z: https://curity.io/resources/learn/dotnet-openid-connect-website/

cyberlinkasp.com, 2018. Understanding the role of virtualization in cloud computing [online]. [cit. 25-8-2023]. Dostupné z: https://www.pluralsight.com/blog/cloud/industry-verticals-cloud-computing-2023

Devopedia.com, 2019. Coupling vs Cohesion [online]. [cit. 25-8-2023]. Dostupné z: https://devopedia.org/cohesion-vs-coupling

Docketry, 2020. How insurance companies benefit from automated document processing [online]. [cit. 1-11-2023]. Dostupné z: https://docketry.ai/blogs/how-insurance-companies-benefit-from-automated-document-processing/

Edmonson James, 2022. Advantages of Databases: Why are databases important to businesses? [online]. [cit. 25-8-2023]. Dostupné z: https://www.businesstechweekly.com/operational-efficiency/data-management/databases-advantages-benefits/

Ellingrud K., Kimura A., Quinn B., Ralph J., 2022. Five steps to improve innovation in the insurance industry [online]. [cit. 11-9-2023]. Dostupné z: https://www.mckinsey.com/industries/financial-services/our-insights/five-steps-to-improve-innovation-in-the-insurance-industry

exforsys.com, 2023. The History of Object Oriented Programming [online]. [cit. 25-8-2023]. Dostupné z: https://www.exforsys.com/tutorials/oops-concepts/the-history-of-object-oriented-programming.html

fronttribe.com, 2023. Front-End Development Trends: A Comprehensive Guide for 2023 [online]. [cit. 25-8-2023]. Dostupné z: https://www.fronttribe.com/stories/front-end-development-trends-2023-guide

Frost R, Pike J., Kenyo L., Pels S., 2011. Business Information Systems: Design an App for That. [cit. 25-8-2023]. Ohio: Saylor Foundation. ISBN 13: 9781453311578 nebo online dostupné z: https://open.umn.edu/opentextbooks/textbooks/46

Gliffy.com, 2023. What is UML? Everything You Need to Know About Unified Modeling Language [online]. [cit. 25-8-2023]. Dostupné z: https://www.gliffy.com/blog/what-is-uml-everything-you-need-to-know-about-unified-modeling-language

Half R., 2023. 4 Advantages of Object-Oriented Programming [online]. [cit. 25-8-2023]. Dostupné z: https://www.roberthalf.com/us/en/insights/career-development/4-advantages-of-object-oriented-programming

Hyde, R., 2022. What it gets right, What it gets wrong and How it could be useful [online]. [cit. 25-8-2023]. Dostupné z: https://medium.com/codex/alphacode-what-it-gets-right-what-it-gets-wrong-and-how-it-could-be-useful-c9fd9cfc2dc8

IBM.com, 2012. What is a data architecture [online]. [cit. 25-8-2023]. Dostupné z: https://www.ibm.com/topics/data-architecture

Kanjilal J., 2018. Association, aggregation and composition in OOP explained [online]. [cit. 25-8-2023]. Dostupné z: https://www.infoworld.com/article/3029325/exploring-association-aggregation-and-composition-in-oop.html

Karan, G, 2024. Front-End Development Trends to Watch in 2024 [online]. [cit. 25-8-2023]. Dostupné z: https://www.credera.com/insights/front-end-development-trends-watch-2024/?redirect=true&referred=tadigital

Khanna M., 2021. Encapsulation in OOP: Definition and Examples [online]. [cit. 25-8-2023]. Dostupné z: https://scoutapm.com/blog/what-is-encapsulation

Krishnakantan K., Lansing J., Münstermann B. Olesen P.B., 2019. IT modernization in insurance: Three paths to transformation [online]. [cit. 12-9-2023]. Dostupné z: https://www.mckinsey.com/industries/financial-services/our-insights/it-modernization-in-insurance-three-paths-to-transformation

Krishnakantan K., McElhaney D., Milinkovich N., Pradhan A., 2021. Transforming the talent model in the insurance industry [online]. [cit. 10-8-2023]. Dostupné z: https://www.mckinsey.com/industries/financial-services/our-insights/how-top-tech-trends-will-transform-insurance

Li, Yazi, Chunji Lu, Yang Liu, 2020. Medical Insurance Information Systems in China: Mixed Methods Study. JMIR Medical Informatics 8 [online]. [cit. 11-12-2023]. Dostupné z: https://api.semanticscholar.org/CorpusID:221562302

Mahipal Nehra, 2019. Top 10 Frameworks for Web Application Development [online]. [cit. 25-8-2023]. Dostupné z: https://www.decipherzone.com/blog-detail/top-web-frameworks

Martin, Robert C.; Clean Architecture: A Craftsman's Guide to Software Structure and Design. 1st Edition. New York: Pearson, 2017. ISBN 978-0134494166.

McKinsey, 2020. How top tech trends will transform insurance [online]. [cit. 11-8-2023]. Dostupné z: https://www.mckinsey.com/industries/financial-services/our-insights/transforming-the-talent-model-in-the-insurance-industry

Microsoft.com, 2022. Introduction to .NET [online]. [cit. 25-8-2023]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/core/introduction

Microsoft.com, 2023. Overview of ASP.NET Core [online]. [cit. 25-8-2023]. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-8.0

MILES, Rob. Exam Ref 70-483 Programming in C#. 2nd edition. New York: Pearson Education, 2018. ISBN 978-1509306985.

Omg.org, 2014. The architecture of choice for a changing world [online]. [cit. 25-8-2023]. Dostupné z: https://www.omg.org/mda/

Omg.org, 2017. UML 2.5.1 Infrastructure [online]. [cit. 25-8-2023]. Dostupné z: https://www.omg.org/spec/UML/2.5.1/About-UML

opensudo.org, 2023. The Future of Backend Development: Trends and Predictions for 2024 [online]. [cit. 25-8-2023]. Dostupné z: https://opensudo.org/the-future-of-backend-development-trends-and-predictions-for-2024/

oracle.com, 2023. What Is a Database? [online]. [cit. 25-8-2023]. Dostupné z: https://www.oracle.com/database/what-is-database/

Per C., 2011. Syntax – Tech term [online]. [cit. 25-8-2023]. Dostupné z: https://techterms.com/definition/syntax

pluralsight.com, 2023. How are different industries using cloud tech in 2023? [online]. [cit. 25-8-2023]. Dostupné z: https://www.pluralsight.com/blog/cloud/industry-verticals-cloud-computing-2023

Programiz.com, 2023. Java Inheritance [online]. [cit. 25-8-2023]. Dostupné z: https://www.programiz.com/java-programming/inheritance

Quanit M., 2021. Software Engineering Principle — Coupling & Cohesion [online]. [cit. 25-8-2023]. Dostupné z: https://dev.to/mquanit/software-engineering-principle-coupling-cohesion-1mma

Schmitz A., 2012. Bussiness information systems design. [cit. 25-8-2023]. Washington: Saylor Academy. Dostupné z: https://www.scaler.com/topics/oops-advantages/

Shaw M., DeLine R., Zelesnik G., 1996. "Abstractions and implementations for architectural connections," Proceedings of International Conference on Configurable Distributed Systems, Annapolis. [cit. 25-8-2023]. Dostupné z: https://ieeexplore.ieee.org/abstract/document/509340

SLAVÍK, J., 2013. Finanční průvodce nefinančního manažera. Česko: Grada. 8024745933

Staruml.io, 2023. Class Diagram [online]. [cit. 25-8-2023]. Dostupné z: https://docs.staruml.io/working-with-uml-diagrams/class-diagram

Stojkovic, N., 2023. The Genesis of Object-Oriented Programming (OOP): A Historical Overview [online]. [cit. 25-8-2023]. Dostupné z: https://brightmarbles.io/blog/object-oriented-programming-history/

Sushant G., 2023. Advantages and Disadvantages of OOP [online]. [cit. 25-8-2023]. Dostupné z: https://www.scaler.com/topics/oops-advantages/

visual-paradigm.com, 2023. What is Unified Modeling Language (UML)? [online]. [cit. 25-8-2023]. Dostupné z: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/

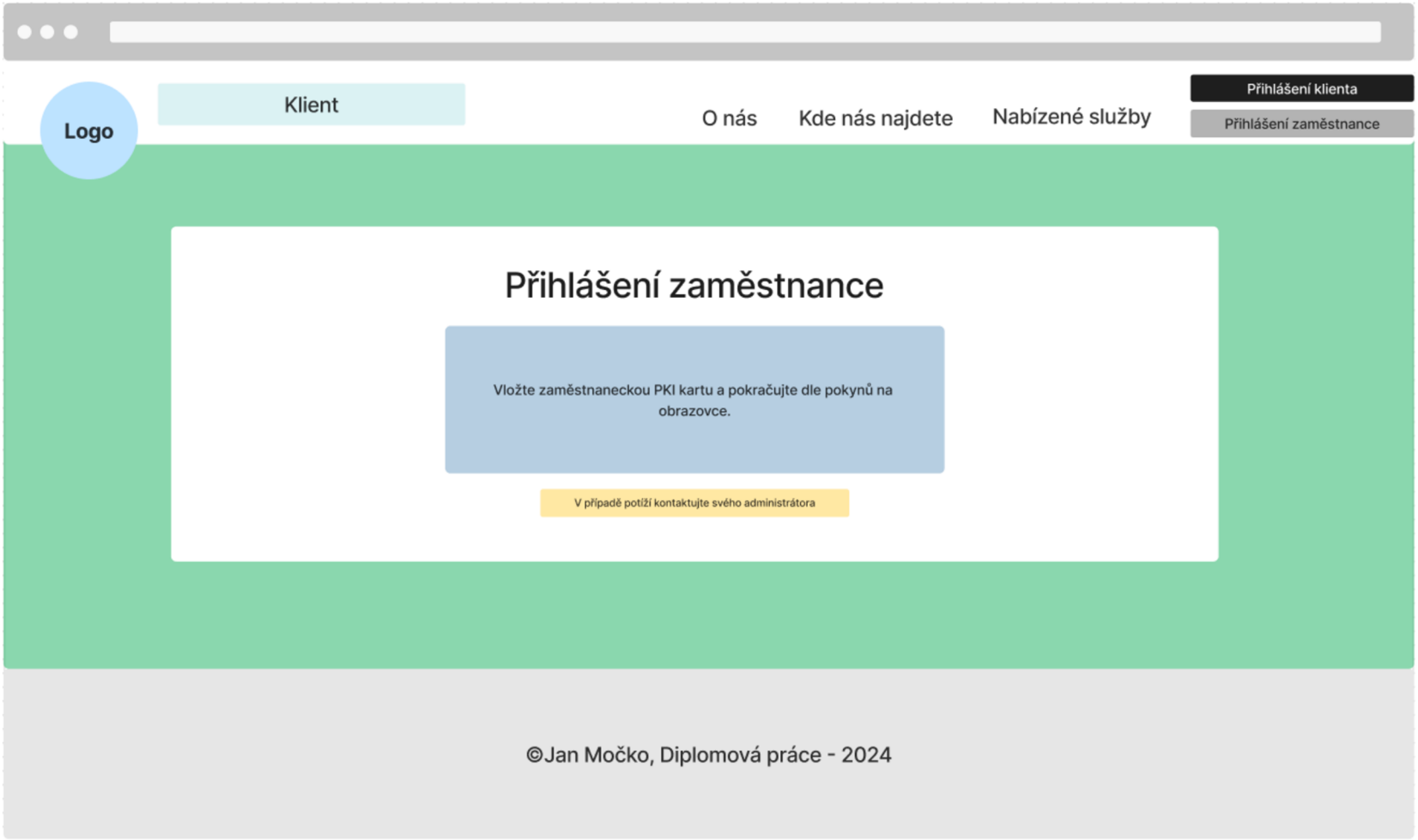
Yurchyshyn, Y, 2023. Trends and Future of Front-End Web Development [online]. [cit. 25-8-2023]. Dostupné z: https://www.romexsoft.com/blog/front-end-development-trends/

8 Seznam obrázků; Seznam tabulek

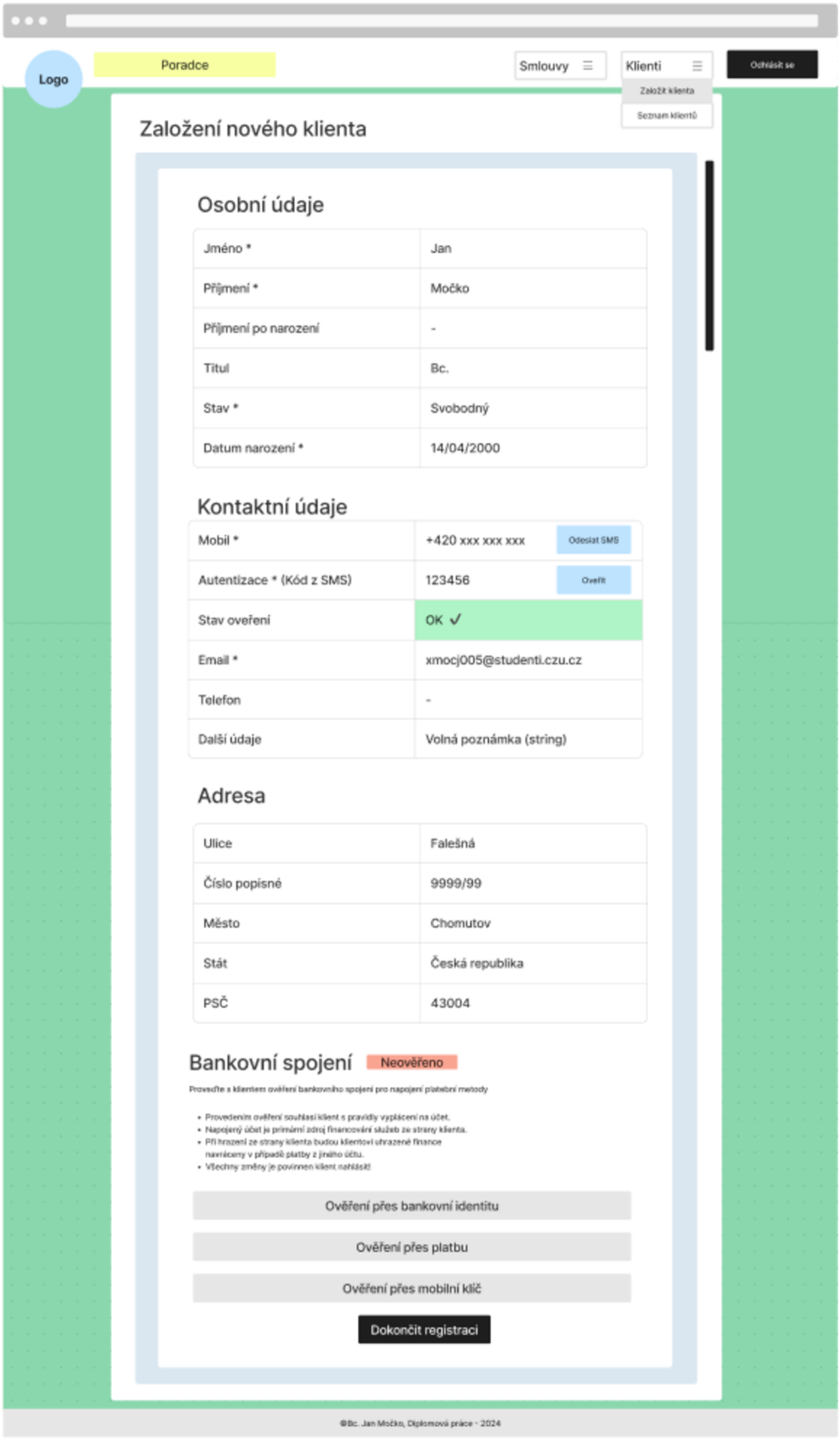
Přílohy

Návrh Wireframe a Designu

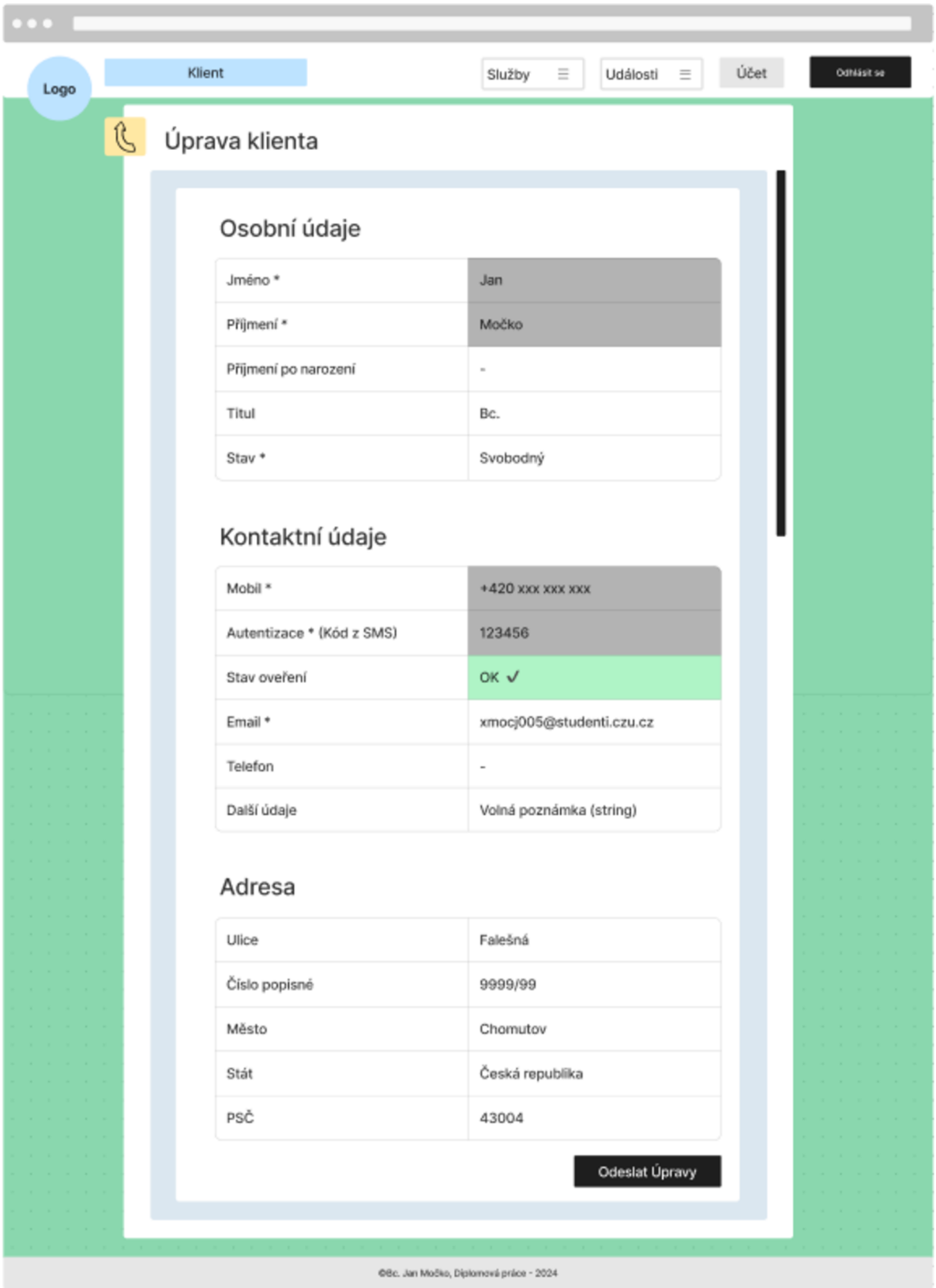
1-A Přihlášení / Klient



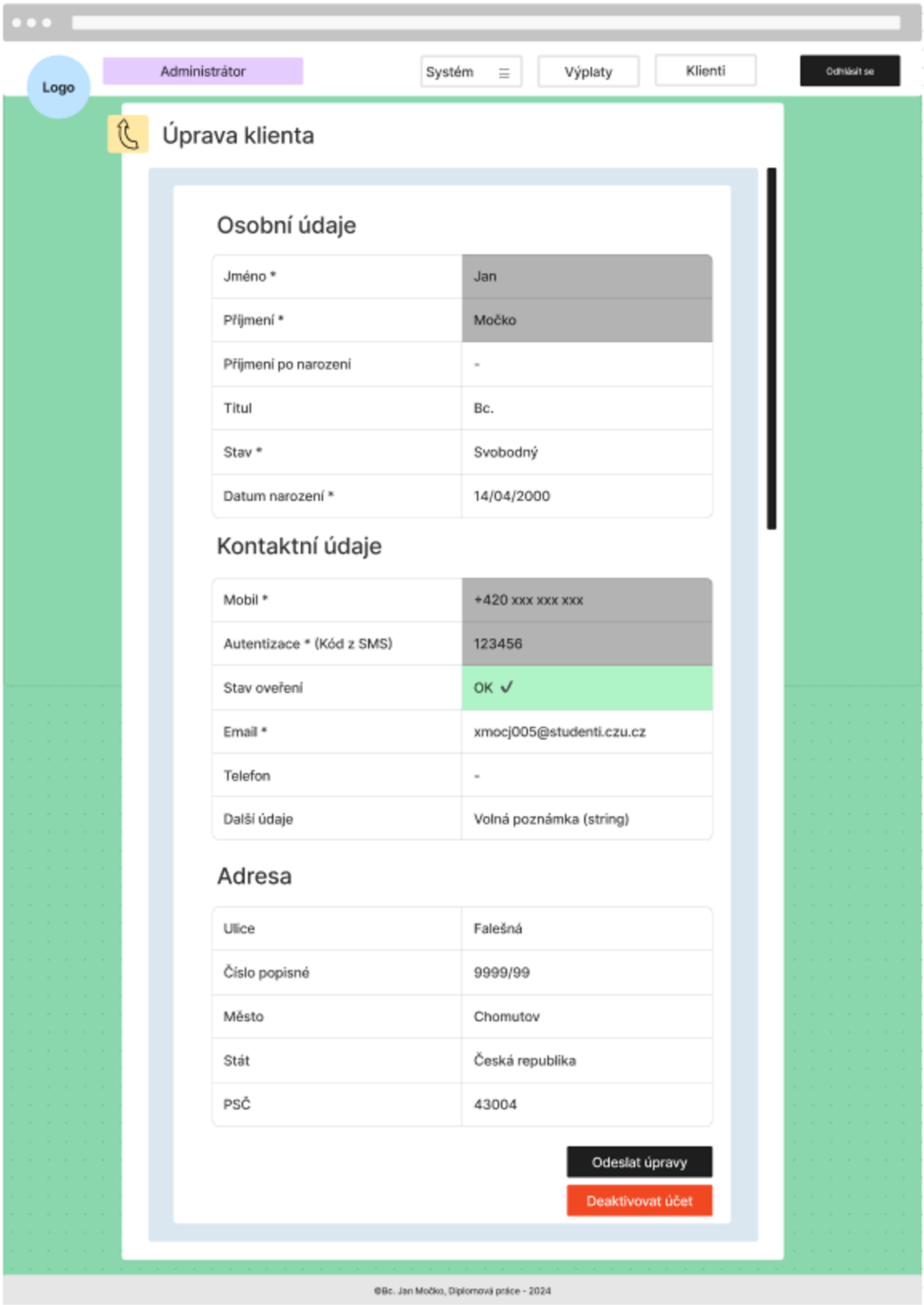
1-B Přihlášení / Zaměstnanec



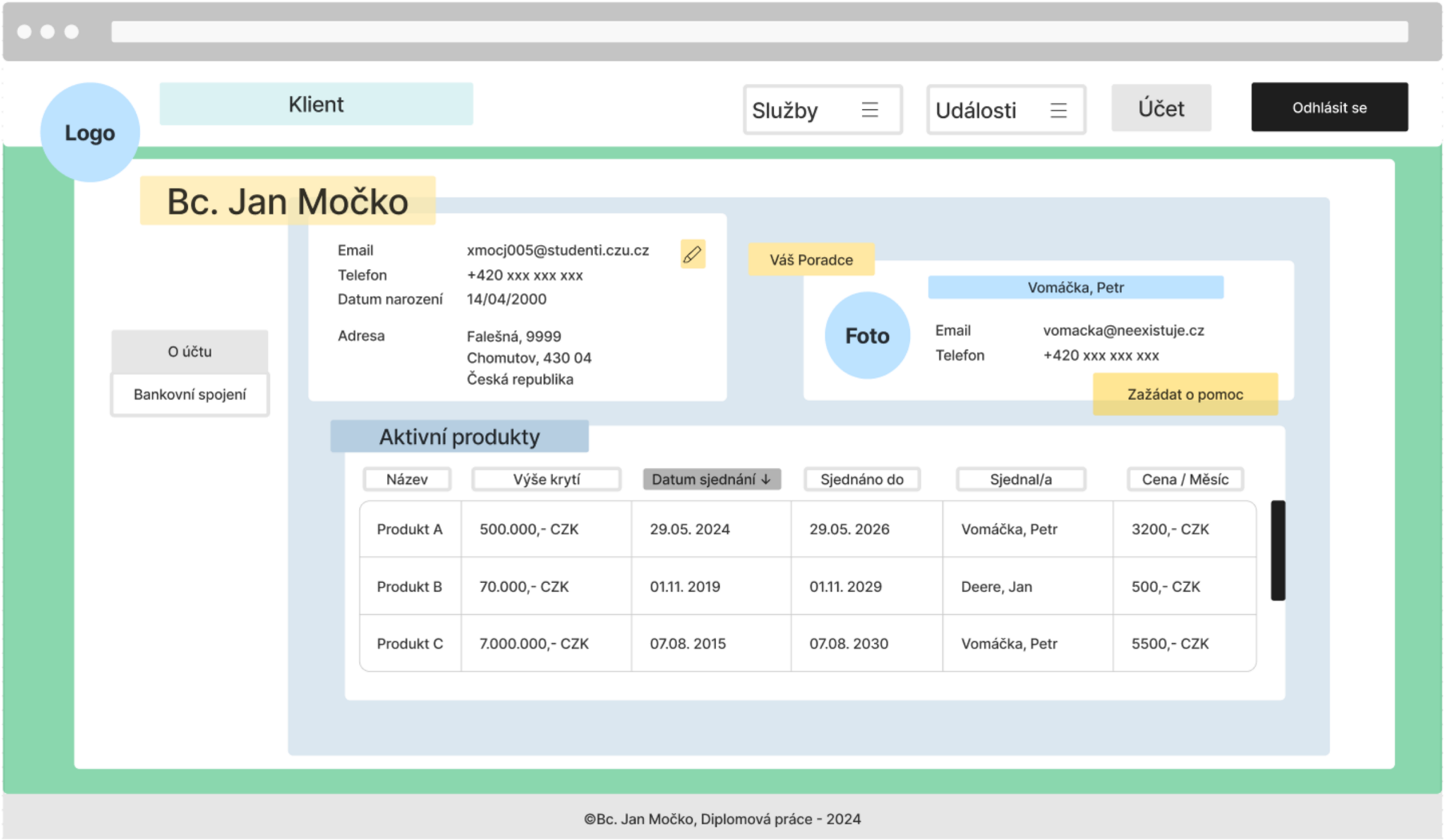
2-A Registrace klienta / Poradce



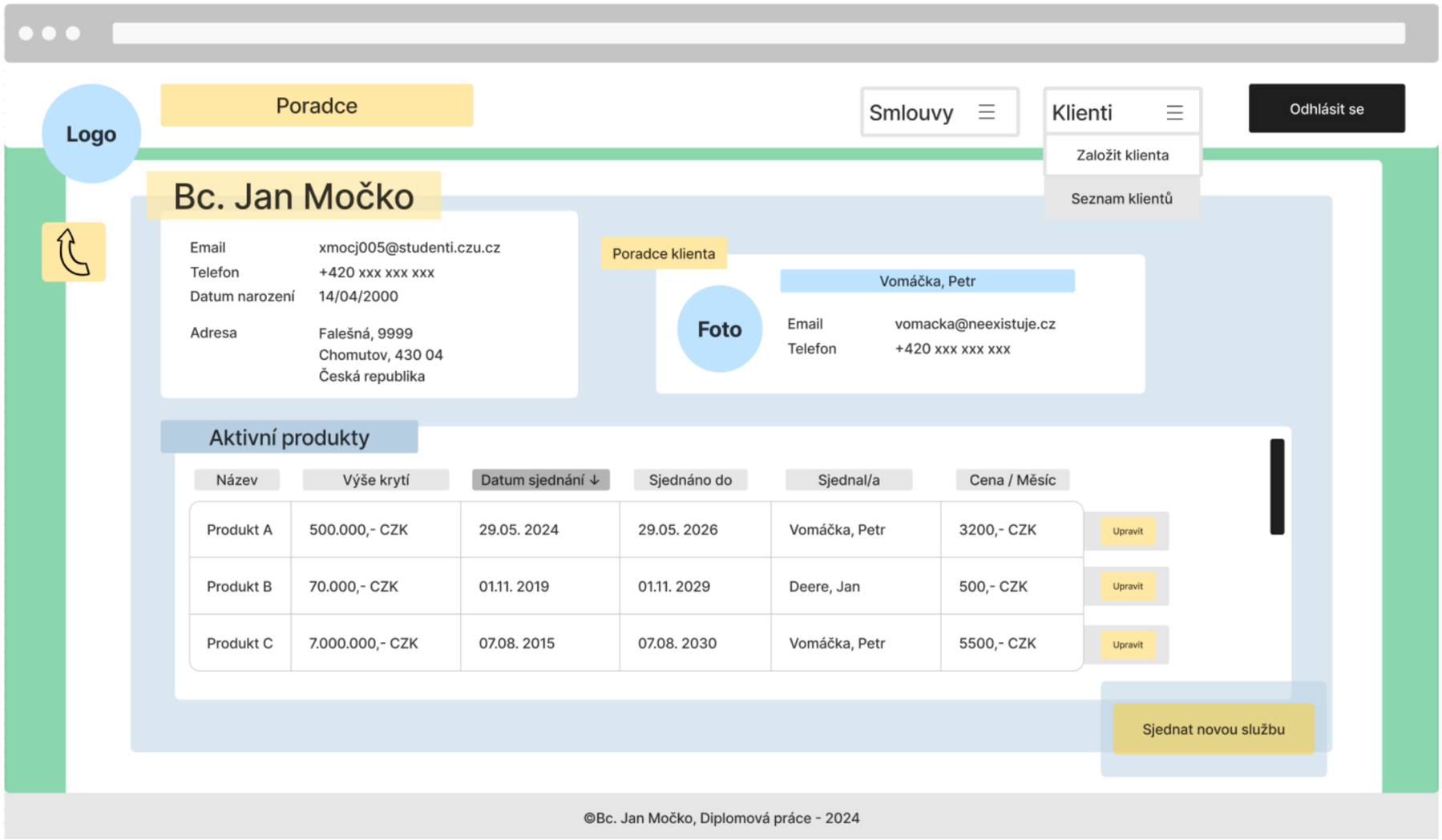
2-B Úprava klienta / Klient



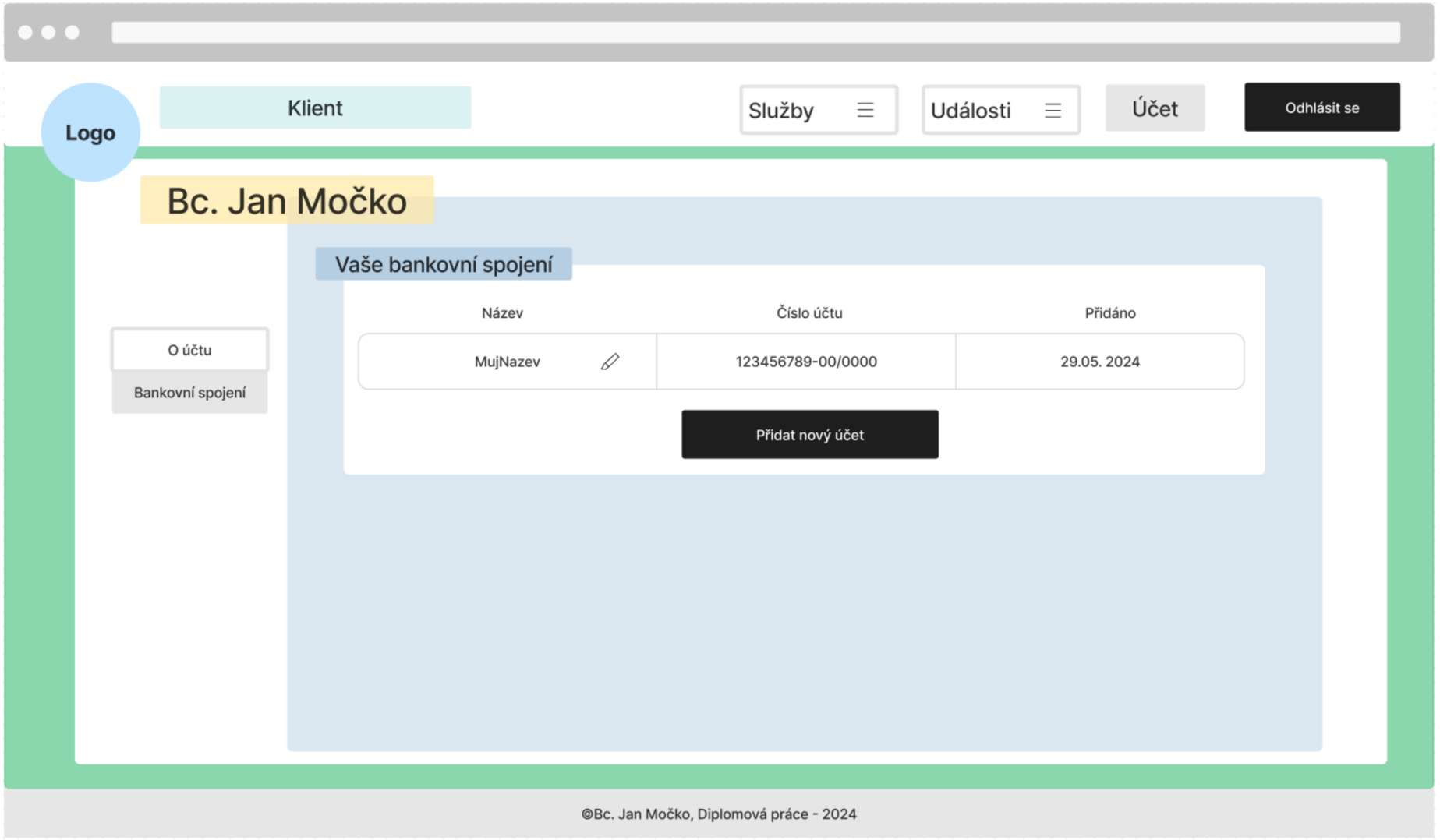
2-C Úprava klienta / Administrátor



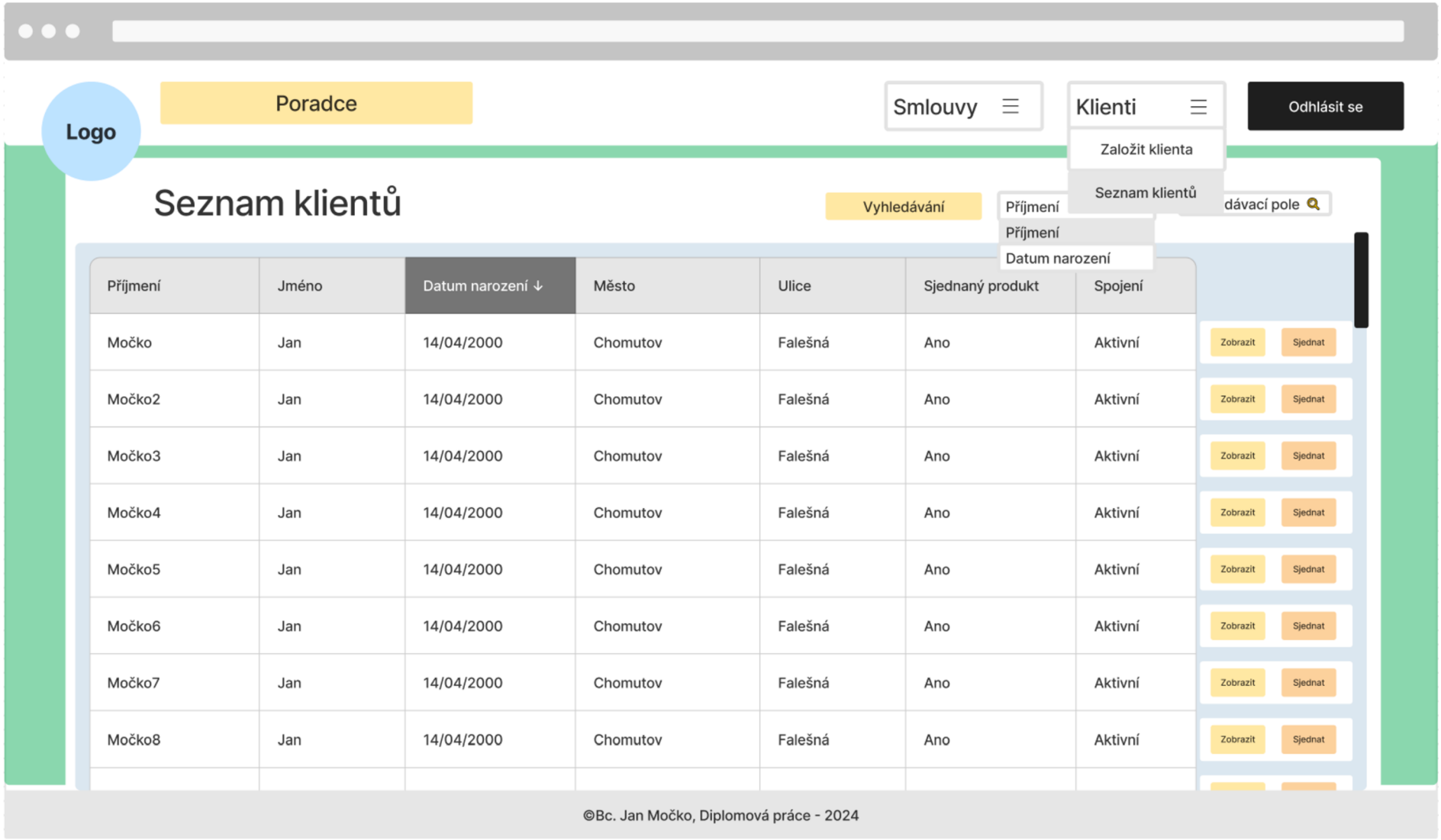
3-A Detail účtu - o účtu / Klient



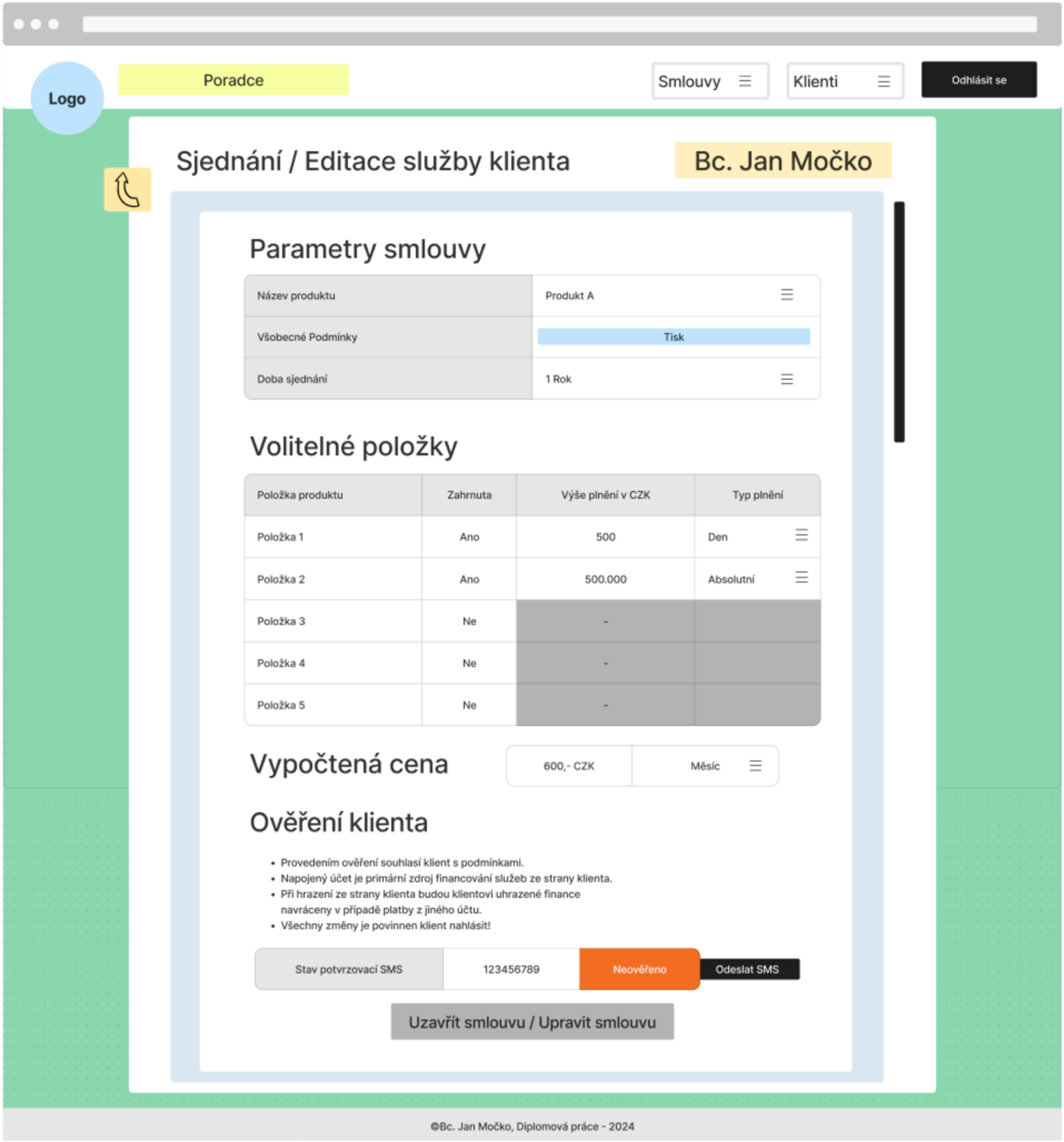
3-B Detail účtu / Poradce



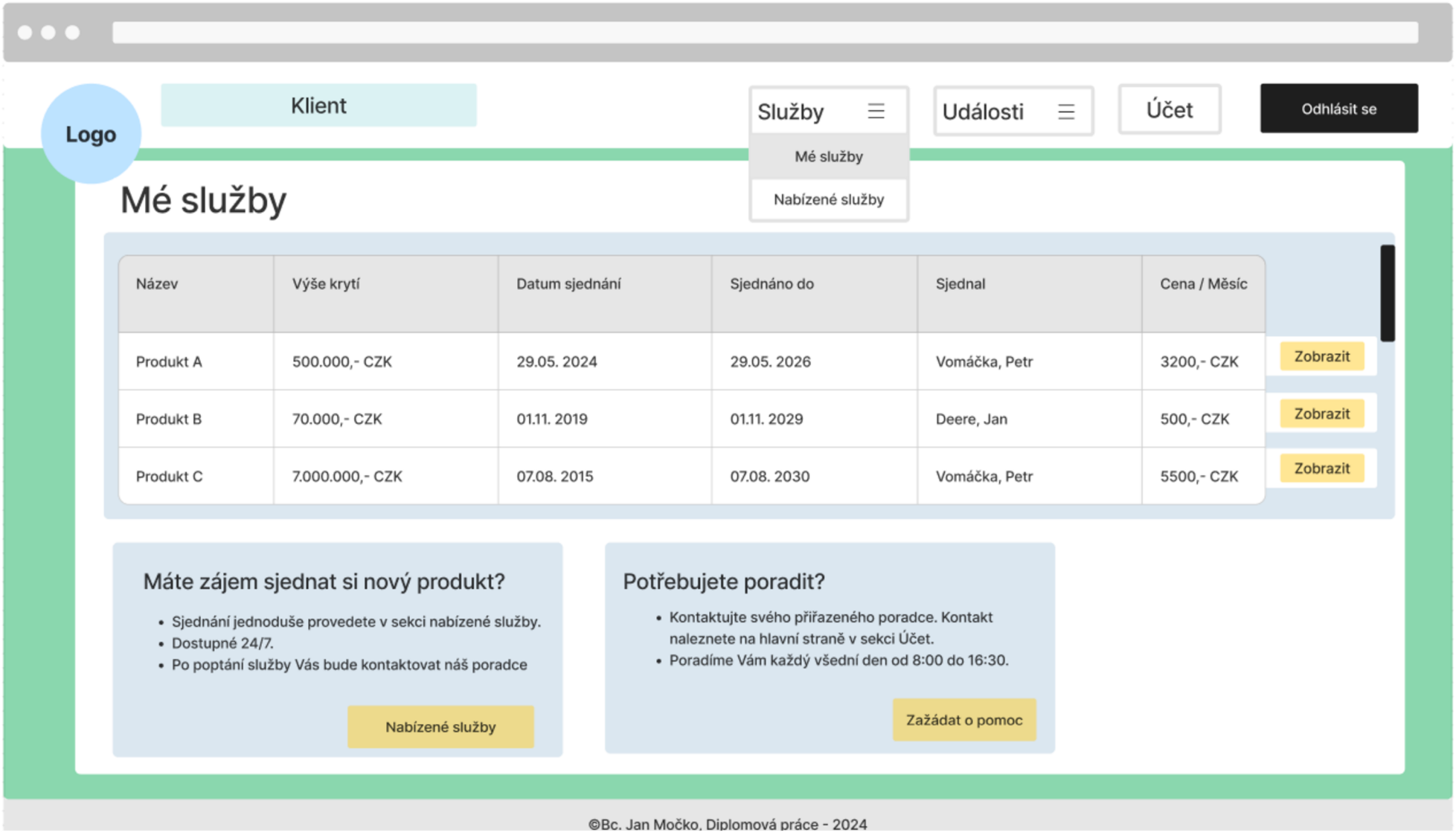
4-A Detail účtu - Bankovní spojení / Klient



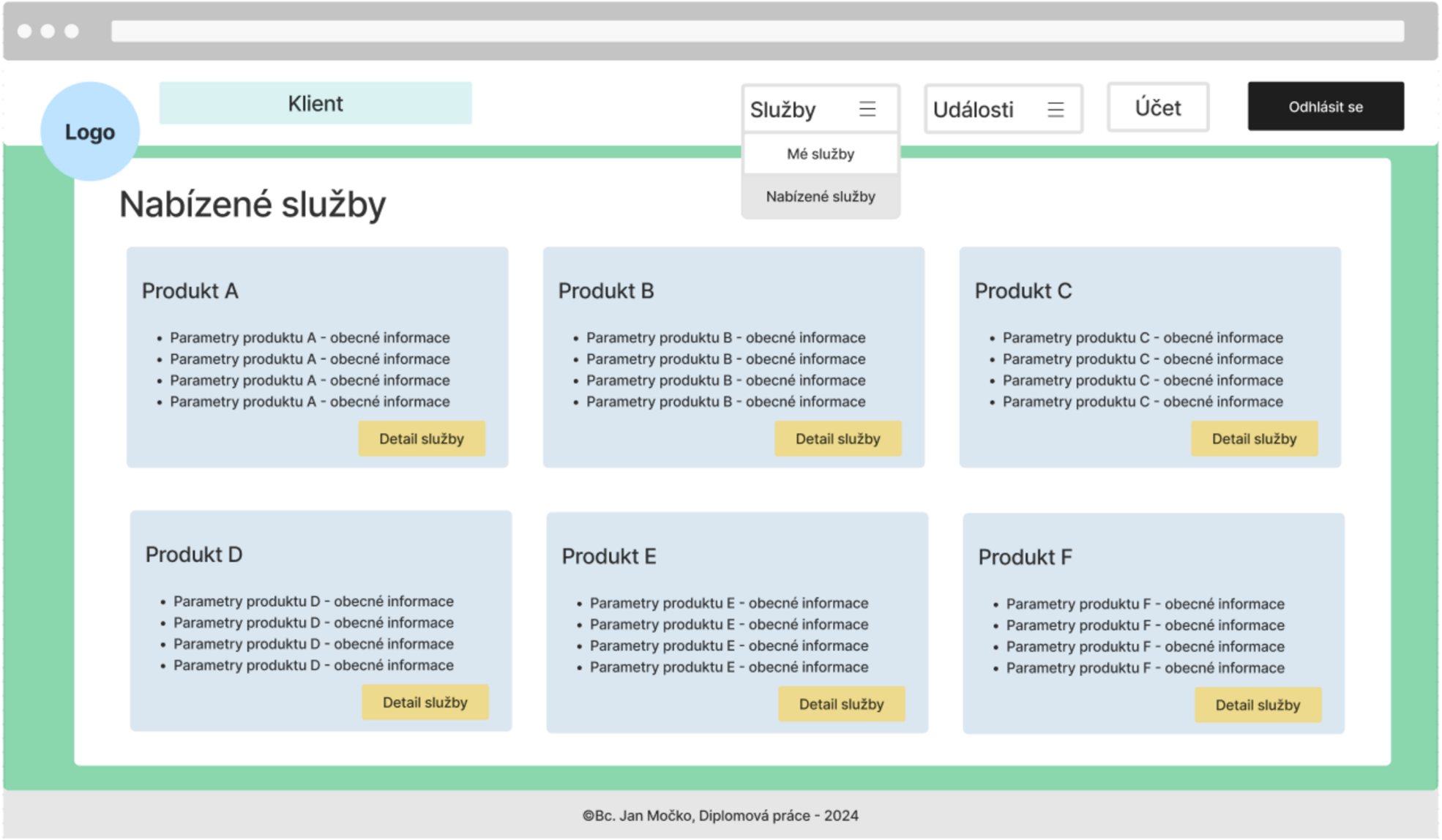
5-A Seznam klientů / Poradce / Administrátor



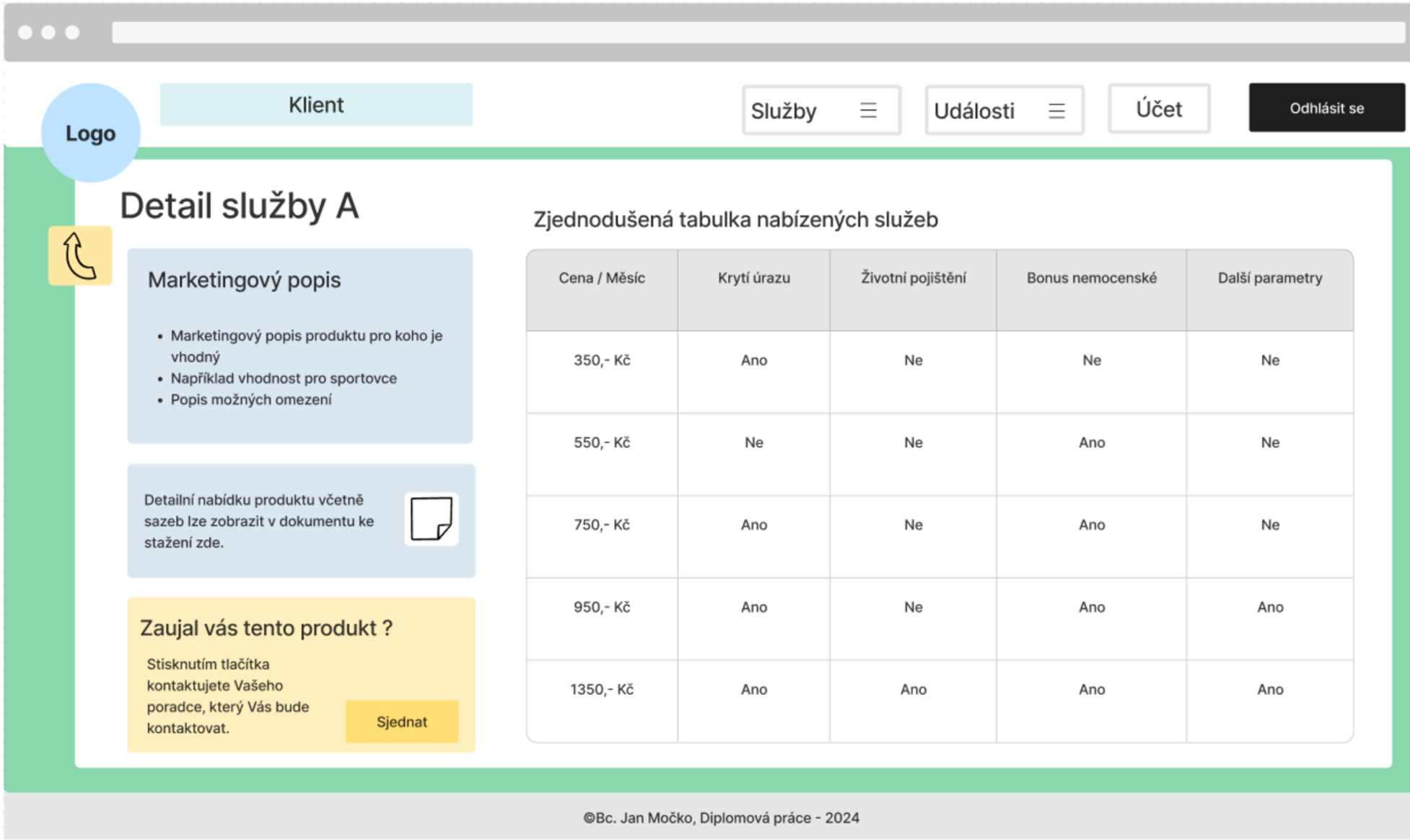
6-A Sjednání - Úprava služby / Poradce



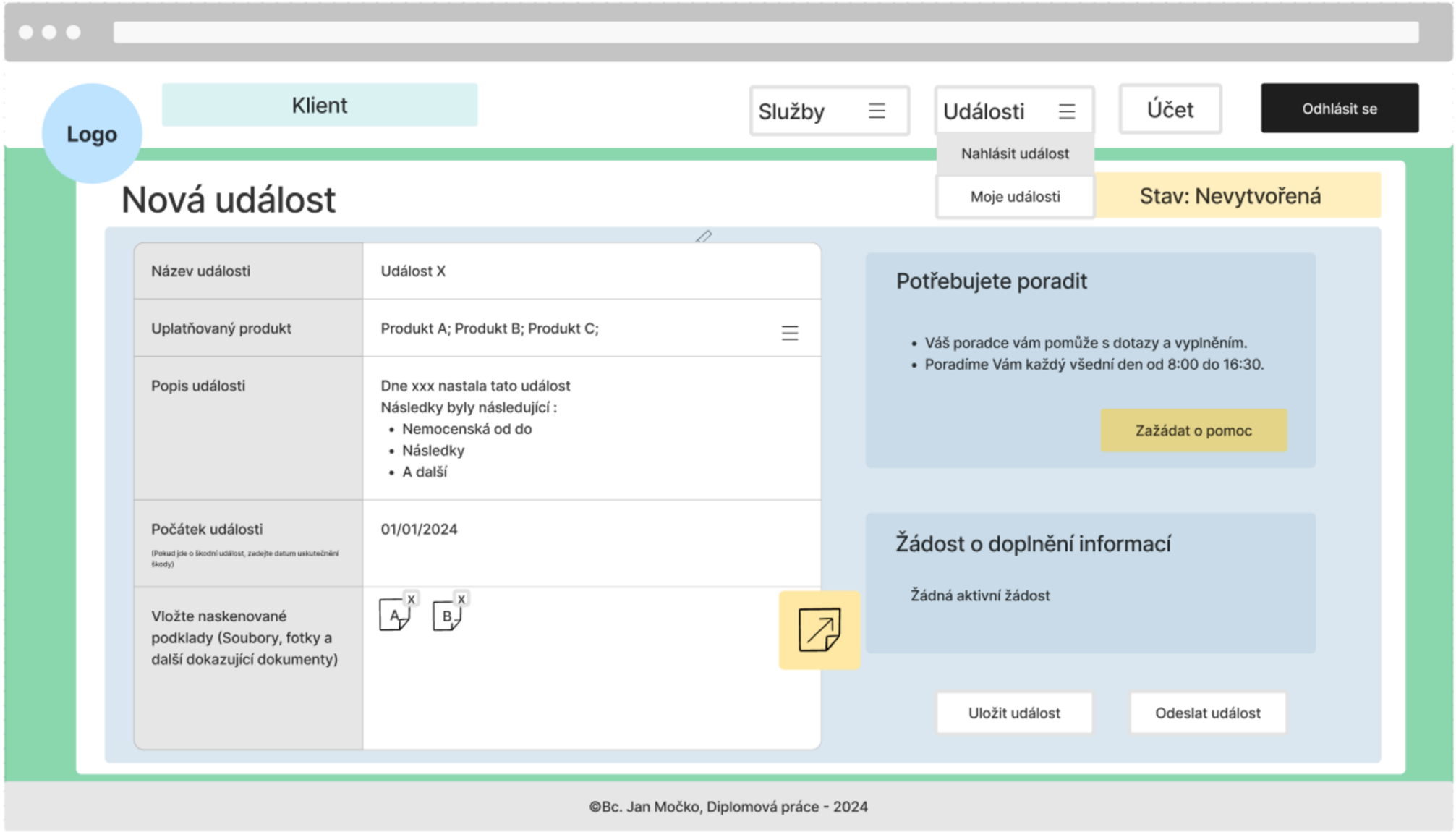
7-A Mé služby / klient



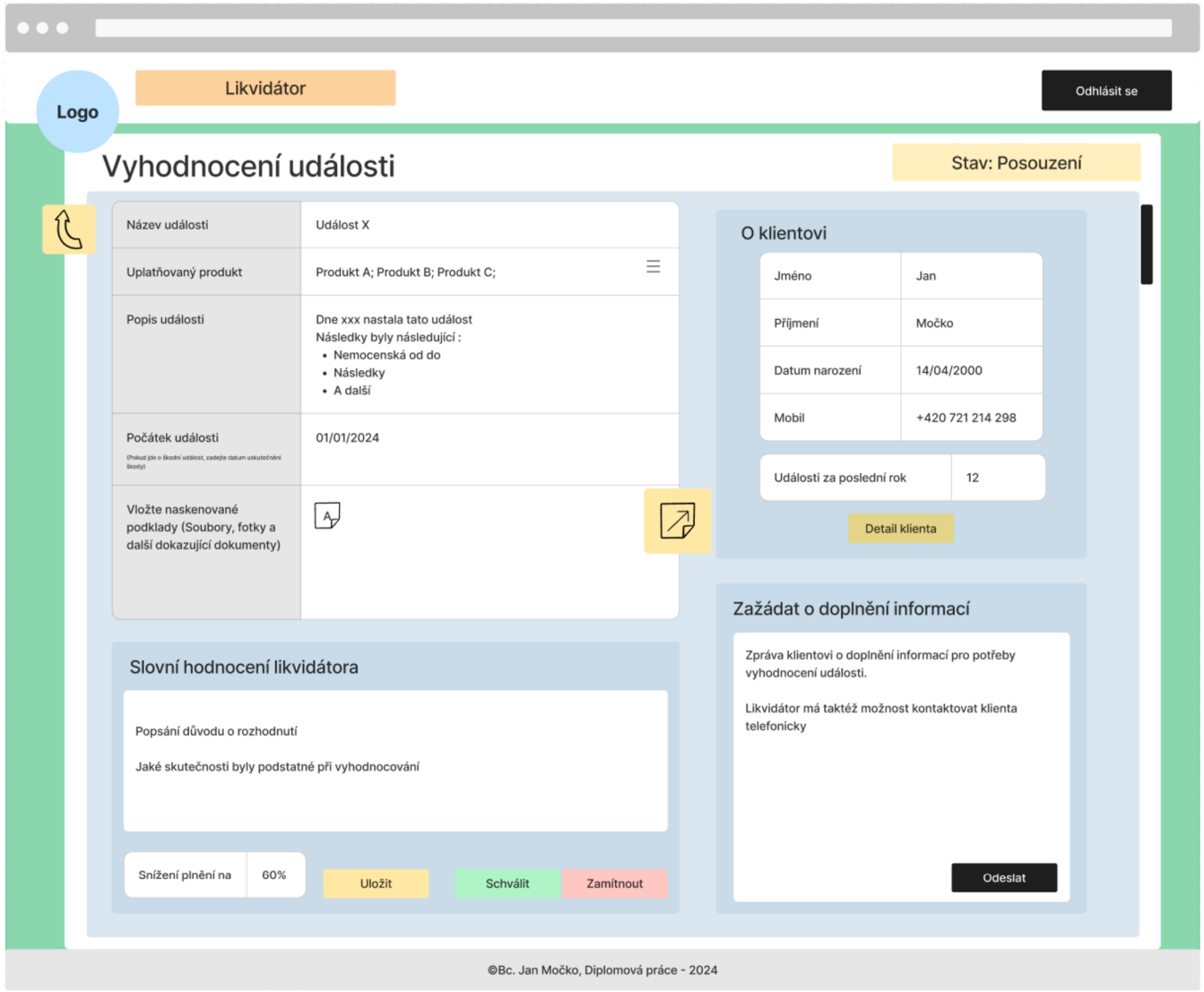
8-A Nabízené služby / klient



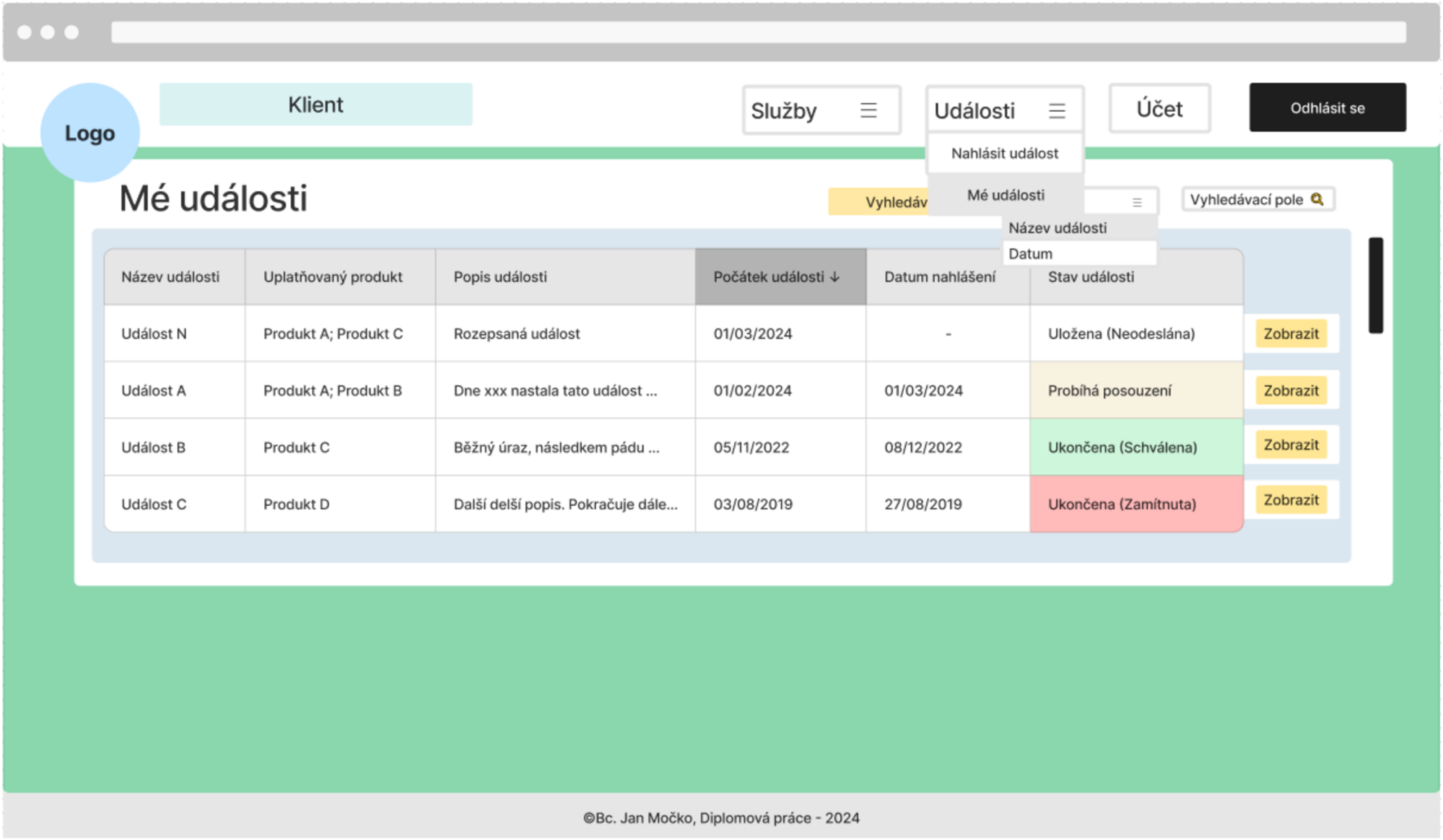
9-A Detail služby/ klient



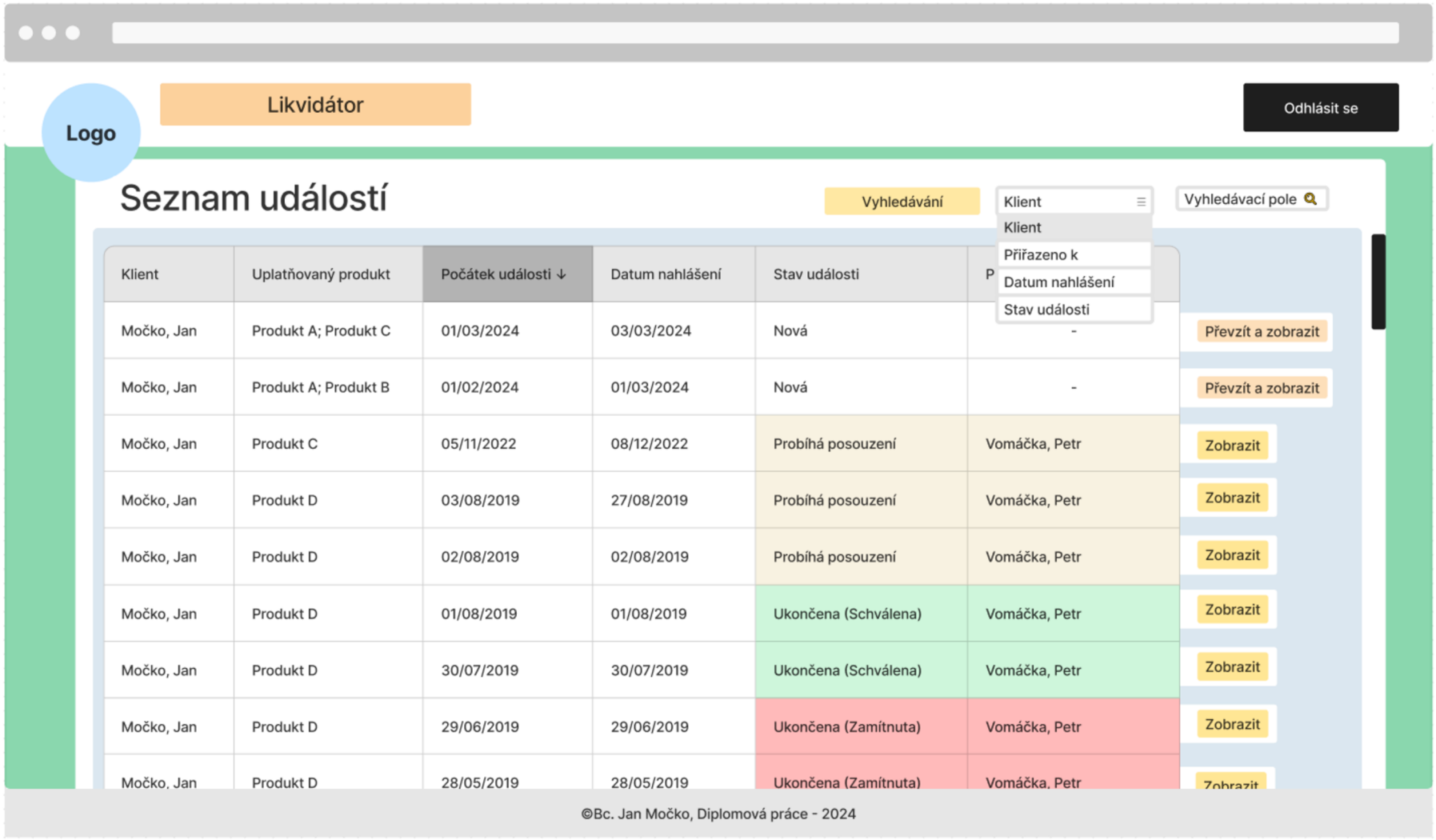
10-A Nahlásit událost / klient



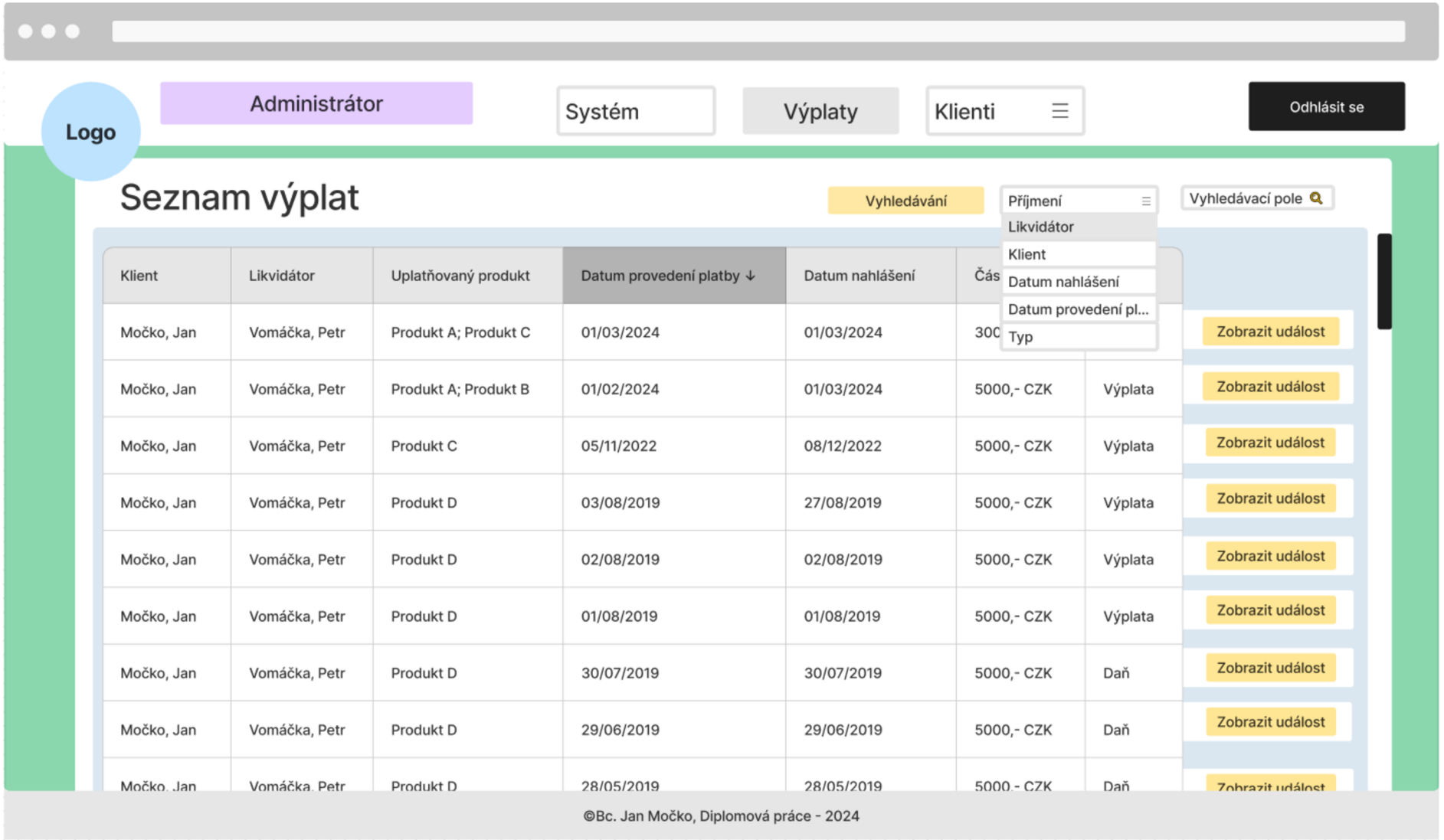
10-B Vyhodnocení události / klient



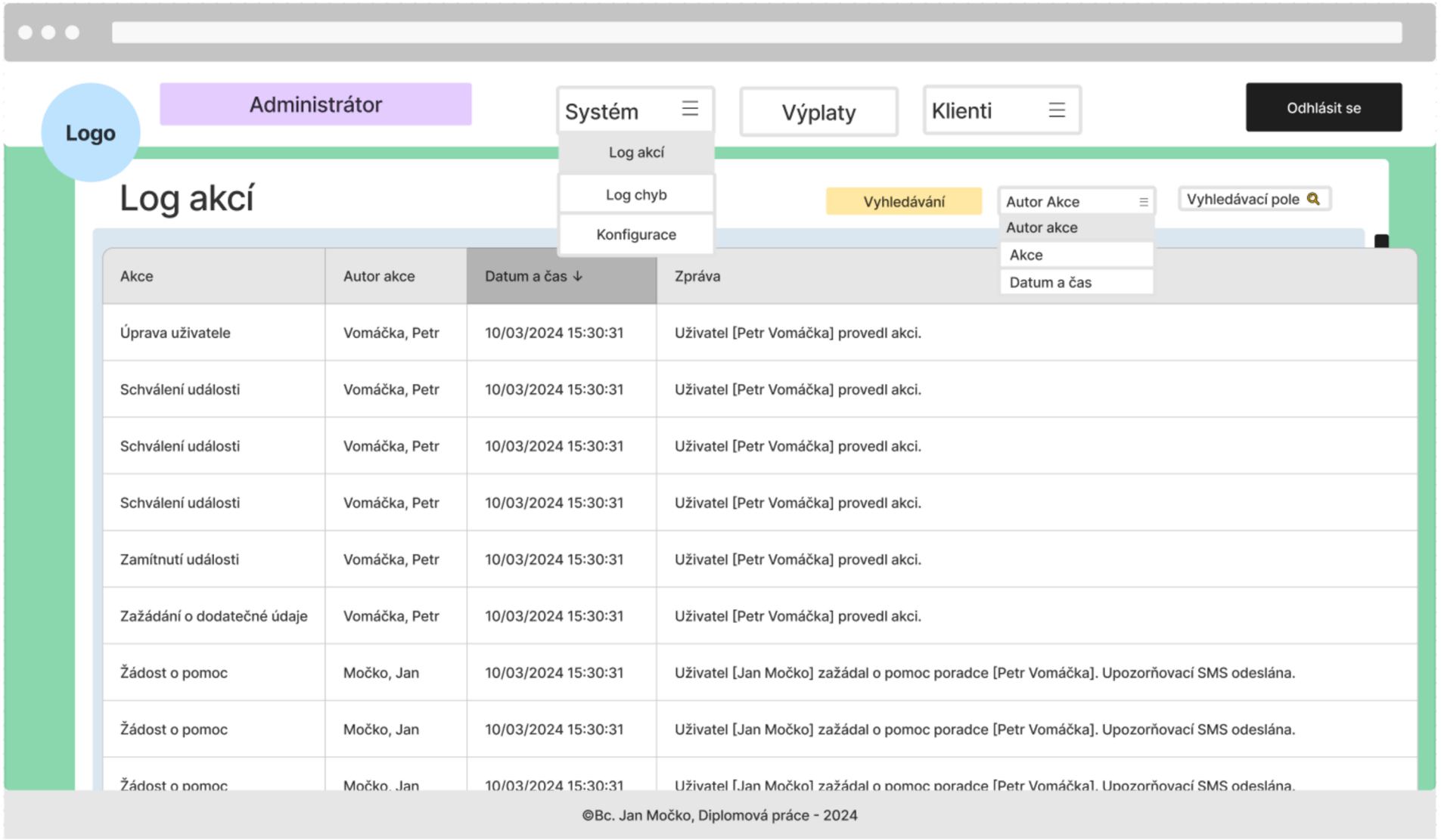
11-A Mé Události/ klient



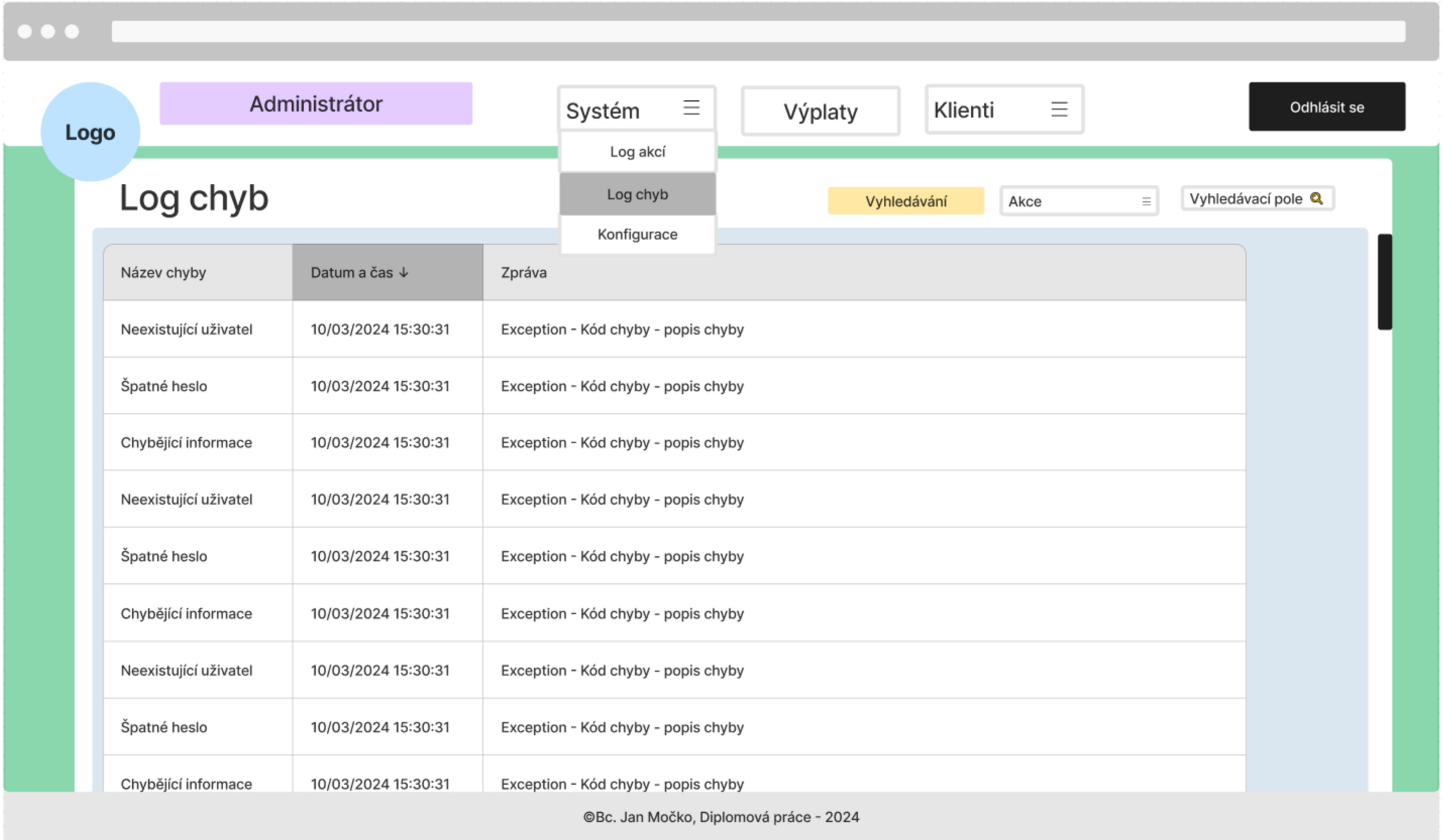
11-B Seznam událostí / Likvidátor



12-A Seznam výplat / administrátor



13-A Log akcí / administrátor



13-B Log chyb / administrátor

Případy užití

UC1 - Přihlášení uživatele

Aktéři

- Klient
- Zaměstnanec
- Systém

Podmínky před spuštěním

- Uživatel má vytvořený účet.

Základní tok

1. Uživatel otevře stránku pro přihlášení.
2. Zadá své uživatelské jméno a heslo.
3. Systém ověří údaje.
4. Při úspěšném ověření systém uživatele přesměruje na domovskou stránku.
5. Uživatel má nyní přístup k svému účtu.

Alternativní tok

- Pokud ověření selže, systém zobrazí chybové hlášení a nabídne možnost obnovit heslo nebo opakovat zadání údajů.

Podmínky po spuštění

- Uživatel je úspěšně přihlášen do systému a má přístup k funkcím systému odpovídajícím jeho roli (klient nebo zaměstnanec).

UC2 – Registrace klienta do systému

Aktéři

- Poradce: Zaměstnanec pojišťovny, který je odpovědný za registraci nových klientů.
- Klient: Osoba, která se stává klientem pojišťovny.
- Systém

Podmínky před spuštěním

- Poradce je přihlášen do systému na pobočce.
- Klient je fyzicky přítomen na pobočce a má připraveny všechny potřebné dokumenty a informace pro registraci.

Základní tok

11. Klient obdrží smlouvu o poskytování služeb a souhlas s ochranou osobních údajů k podpisu.
12. Po podpisu dokumentů systém uloží údaje a smlouvy a aktivuje uživatelský účet.
13. Poradce vybere v systému možnost pro přidání nového klienta.
14. Klient poskytne všechny potřebné informace (jméno, adresa, datum narození, kontaktní údaje atd.).
15. Poradce vloží informace do systému.
16. Systém ověří unikátnost a validitu zadaných údajů.
17. Pokud jsou údaje správné a unikátní, systém vytvoří nový uživatelský účet.
18. Systém vygeneruje uživatelské jméno a heslo pro klienta.
19. Poradce informuje klienta o podrobnostech účtu a provádí instruktáž k používání samoobslužného rozhraní.
20. Klient je informován o úspěšné registraci a možnostech následného používání služeb pojišťovny.

Alternativní tok bodu 4

3. Pokud systém identifikuje problém s údaji (duplicita, neplatnost atd.), poradce je vyzve k opravě.
4. Klient nebo poradce opraví údaje. Tok pokračuje bodem č. 5.

Podmínky po spuštění

- Klient má aktivní uživatelský účet v systému pojišťovny.
- Klient má podrobné informace o tom, jak přistupovat k účtu a jak využívat služby pojišťovny.
- Všechny relevantní dokumenty jsou podepsány a uloženy jak v papírové, tak v elektronické formě.

UC3 – Vytvoření pojistné smlouvy

Aktéři

- Klient: Osoba, která chce uzavřít pojistnou smlouvu.
- Systém
- Databáze

Podmínky před spuštěním

- Klient musí být zaregistrován a přihlášen do systému.
- Systém musí být dostupný a funkční.
- Klient musí mít veškeré potřebné informace a dokumenty potřebné pro uzavření smlouvy.

Základní tok

11. Poradce se přihlásí do systému.
12. Poradce s klientem zvolí typ pojištění.
13. Poradce seznámí klienta s podmínkami produktu.
14. Poradce s klientem vyplní potřebné údaje pro smlouvu.
15. Pokud jsou data validní, systém vypočítá cenu pojištění.
16. Klient potvrdí detaily smlouvy a akceptuje cenu.
17. Klient potvrdí identitu přes SMS ověření.
18. Klient elektronicky podepíše pojistnou smlouvu.
19. Systém uloží smlouvu do databáze smluv.
20. Klient obdrží potvrzení o úspěšném uzavření smlouvy.

Alternativní tok bodu 3-A

3. Klient nesouhlasí s podmínkami smlouvy.
4. Klient si chce si vybrat jiný produkt. Tok se vrací do bodu č. 2.

Alternativní tok bodu 3-B

3. Klient nesouhlasí s podmínkami smlouvy.
4. Klient nechce pokračovat. Tok končí.

Alternativní tok bodu 6-A

3. Pokud klient nesouhlasí s cenou nebo detaily smlouvy, může požádat o revizi nebo změny.
4. Systém umožní klientovi provést změny a aktualizovat nabídku. Tok pokračuje bodem č. 5.

Alternativní tok bodu 6-B

3. Pokud klient nesouhlasí s cenou nebo detaily smlouvy, chce jiný produkt.
4. Systém umožní vybrat jiný produkt. Tok se vrací do bodu č. 2.

Alternativní tok bodu 6-C

3. Pokud klient nesouhlasí s cenou nebo detaily smlouvy.
4. Klient nechce pokračovat. Tok končí.

Podmínky po spuštění

- Pojistná smlouva je aktivní a uložena v systému.
- Klient má přístup k digitální kopii pojistné smlouvy.
- Systém má zaznamenanou veškerou komunikaci a dokumentaci související s procesem uzavření smlouvy.

UC4 - Hlášení pojistné události klientem

Aktéři

- Klient
- Systém

Podmínky před spuštěním

- Klient je přihlášen do systému.
- Klient má aktivní pojistnou smlouvu.

Základní tok

1. Klient vybere možnost "Nahlásit událost".
2. Vyplní formulář s informacemi o události, včetně datumu, popisu a případných důkazů.
3. Odešle formulář.
4. Systém potvrdí přijetí hlášení a informuje klienta o dalším postupu.

Alternativní tok

- Pokud klient nemůže dokončit formulář, může jej uložit jako koncept a dokončit později.

Podmínky po spuštění

Hlášení pojistné události bylo úspěšně přijato systémem. Klient je informován o přijetí hlášení a očekává další instrukce nebo rozhodnutí o pojistném plnění.

UC5 - Správa pojistných událostí

Aktéři

- Likvidátor
- Systém

Podmínky před spuštěním

- Likvidátor je přihlášen do systému.

Základní tok

1. Likvidátor zobrazí seznam nahlášených pojistných událostí.
2. Vybere událost k posouzení.
3. Prozkoumá podrobnosti a dostupné důkazy.
4. Rozhodne o výši pojistného plnění.
5. Zadá rozhodnutí do systému, který následně informuje klienta.

Alternativní tok

- Pokud jsou potřeba další informace, likvidátor požádá klienta o doplnění údajů nebo dokumentů.
- Toto jsou podrobné scénáře pro vybrané use cases, které reflektují základní interakce mezi uživateli a systémem. Tyto popisy mohou sloužit jako východisko pro další rozvoj a detailní specifikaci systému.

Podmínky po spuštění

- Pojistná událost byla zpracována likvidátorem.
- Rozhodnutí o výši pojistného plnění bylo zaznamenáno do systému, a klient je informován o výsledku.
- Systém aktualizuje status pojistné události odpovídajícím způsobem.

UC6 – Výplata pojistného plnění

Aktéři

- Klient
- Likvidátor (zaměstnanec pojišťovny odpovědný za vyřizování pojistných událostí)
- Systém
- Databáze
- Bankovní API

Podmínky před spuštěním

- Klient je přihlášen do samoobslužného rozhraní.
- Likvidátor je přihlášen do systému.
- Smlouva a pojistná událost jsou již zaznamenány v systému.
- Pojistná událost byla likvidátorem uznána k plnění.

Základní tok

1. Likvidátor získá seznam schválených výplat pro zpracování v systému.
2. Likvidátor vybere pojistnou událost ze seznamu pro provedení výplaty.
3. Systém zobrazí detaily události a spojené smlouvy pro kontrolu.
4. Likvidátor zadá výši pojistného plnění podle podmínek smlouvy a události.
5. Systém provede výpočet daně z pojistného plnění (pokud je zdanitelné).
6. Systém generuje návrh transakce pro výplatu a související daňovou výplatu.
7. Likvidátor schválí transakci.
8. Systém zašle pokyn k výplatě přes bankovní API.
9. Bankovní API potvrdí provedení transakce.
10. Systém zaznamená provedení výplaty a aktualizuje stav události.
11. Systém odesílá notifikaci klientovi o výplatě plnění.

Alternativní tok bodu 4

3. Pokud výše plnění není jednoznačná, likvidátor žádá o další dokumentaci nebo informace.

4. Klient poskytne požadované informace. Tok pokračuje bodem č. 5.

Alternativní tok bodu 7

3. Pokud likvidátor neschválí transakci, systém vyzve k doplnění informací nebo k opravě výpočtu.
4. Likvidátor opraví informace a znovu schválí transakci. Tok pokračuje bodem č. 8.

Alternativní tok bodu 9

3. Pokud bankovní API nevrátí potvrzení, systém zkusí transakci odeslat znovu.
4. Pokud opětovné odeslání selže, likvidátor je upozorněn na potíže s transakcí a kontaktuje banku.

Podmínky po spuštění

- Výplata byla klientovi úspěšně zaslána.
- Stav události je aktualizován na "Vyplaceno".
- Klient byl informován o výplatě.

UC7 - Správa uživatelských účtů

Aktéři

- Administrátor
- Systém

Podmínky před spuštěním

- Administrátor je přihlášen do systému.

Základní tok

1. Administrátor přejde do sekce správy účtů.
2. Administrátor vybere možnost "Spravovat uživatelské účty".
3. Systém zobrazí seznam uživatelských účtů.
4. Pro přidání nového účtu zvolí "Přidat nového uživatele".
5. Vyplní požadované údaje a potvrdí vytvoření účtu.
6. Systém ověří údaje a přidá nový uživatelský účet.
7. Pro úpravu účtu vybere uživatele ze seznamu a aktualizuje požadované údaje.
8. Pro odstranění účtu vybere uživatele a potvrdí jeho vymazání.

Alternativní toky

- Pokud administrátor zjistí chybu v údajích uživatele, může vybrat možnost "Opravit údaje" a provést nezbytné změny.

Podmínky po spuštění

- Údaje uživatele jsou aktualizovány, přidány nebo odstraněny.

UC8 - Správa pojistných produktů

Aktéři

- Administrátor
- Systém

Podmínky před spuštěním

- Administrátor je přihlášen do systému.

Základní tok

1. Administrátor přejde do sekce správy produktů.
2. Administrátor vybere možnost "Spravovat pojistné produkty".
3. Systém zobrazí seznam pojistných produktů.
4. Pro přidání nového produktu zvolí "Přidat nový produkt".
5. Vyplní požadované údaje o produktu a potvrdí jeho přidání.
6. Systém ověří údaje a přidá nový pojistný produkt do nabídky.
7. Pro úpravu produktu vybere produkt ze seznamu a provede požadované změny.
8. Pro odstranění produktu vybere produkt a potvrdí jeho vymazání.

Alternativní toky

- Pokud administrátor chce upravit pokročilá nastavení produktu, může vybrat produkt a vstoupit do pokročilého režimu úprav.

Podmínky po spuštění

- Produkt je přidán, upraven nebo odstraněn z nabídky produktů.
- Pokud byste chtěli tyto use cases rozpracovat ještě detailněji nebo přidat specifické kroky nebo podmínky, dejte mi, prosím, vědět, a já přizpůsobím scénáře vašim potřebám.

UC9 - Monitorování a reportování

Aktéři

- Administrátor

Podmínky před spuštěním

- Administrátor je přihlášen do systému.

Základní tok

1. Administrátor otevře sekci "Reporty" v administračním rozhraní.
2. Vybere typ reportu ze seznamu dostupných reportů: "Log akcí" nebo "Log chyb".
3. Specifikuje parametry pro filtraci reportu, jako je datumový rozsah nebo konkrétní kategorie událostí.
4. Systém zpracuje požadavek a zobrazí report s relevantními daty.
5. Administrátor může report prohlížet online nebo si ho stáhnout pro další analýzu.

Alternativní tok

- Pokud nejsou k dispozici žádná relevantní data pro zvolené kritéria, systém informuje administrátora a nabídne možnost upravit filtraci.

Podmínky po spuštění

- Administrátor má přehled o aktuálním stavu systému nebo identifikovaných problémech na základě zvoleného reportu.

UC10 - Nahlásit událost

Aktéři

- Klient
- Systém

Podmínky před spuštěním

- Klient je přihlášen do systému.

Základní tok

1. Klient přistupuje k formuláři pro nahlášení události v systému.
2. Vyplňuje všechny potřebné informace o události, včetně popisu, data a místa události, a případně přikládá důkazní materiály.
3. Odesílá vyplněný formulář.
4. Systém potvrdí přijetí nahlášení a informuje klienta o přijetí nahlášení a o dalším postupu.

Alternativní tok

- Pokud systém zjistí chybu v informacích nebo chybějící údaje, upozorní klienta a požádá o doplnění nebo opravu informací.

Podmínky po spuštění

- Událost je úspěšně nahlášena v systému a čeká na další zpracování likvidátorem.

UC11 - Mé Události

Aktéři

- Klient
- Systém

Podmínky před spuštěním

- Klient je přihlášen do systému.

Základní tok

1. Klient přistupuje k sekci "Mé události".
2. Systém zobrazí seznam všech událostí, které klient nahlásil, včetně jejich aktuálního stavu zpracování.
3. Klient vybere jednu z událostí pro zobrazení detailnějších informací.
4. Systém zobrazí detaily vybrané události, včetně popisu, data nahlášení, aktuálního stavu a případných výsledků likvidace.

Alternativní tok

- Pokud nejsou k dispozici žádné nahlášené události, systém informuje klienta, že nemá žádné aktuálně zpracovávané události.

Podmínky po spuštění

- Klient má přehled o svých nahlášených pojistných událostech a je informován o jejich aktuálním stavu a výsledcích.

UC12 - Seznam výplat

Aktéři

- Administrátor
- Systém

Podmínky před spuštěním

- Administrátor je přihlášen do systému.

Základní tok

1. Administrátor otevře sekci "Seznam výplat".
2. Systém zobrazí seznam všech výplat pojistného plnění.
3. Administrátor může vyhledávat výplaty podle kritérií, jako jsou datum, jméno klienta nebo stav výplaty.
4. Pro zobrazení detailů o konkrétní výplatě vybere administrátor výplatu ze seznamu.
5. Systém zobrazí detailní informace o vybrané výplatě, včetně částky, data, stavu a případných poznámek.

Alternativní tok

- Pokud administrátor potřebuje upravit informace o výplatě, může tak učinit prostřednictvím funkce pro úpravu, která mu umožní změnit detaily výplaty.

Podmínky po spuštění

- Administrátor má aktualizovaný přehled o výplatách pojistného plnění, včetně všech provedených změn nebo doplnění.

UC13 - Log akcí a Log chyb

Aktéři:

- Administrátor
- Systém

Podmínky před spuštěním

- Administrátor je přihlášen do systému.

Základní tok (Log akcí)

1. Administrátor přistupuje k sekci "Log akcí".
2. Systém zobrazí seznam všech akcí provedených uživateli systému, včetně časového razítka a popisu akce.
3. Administrátor může filtrovat logy podle uživatele, data, nebo typu akce pro lepší přehlednost.
4. Administrátor prozkoumá logy akcí, aby identifikoval neobvyklé nebo neautorizované aktivity.

Základní tok (Log chyb)

1. Administrátor přistupuje k sekci "Log chyb".
2. Systém zobrazí seznam chyb, které systém zaznamenal, včetně časového razítka a popisu problému.
3. Administrátor může prozkoumat detaily každé chyby a určit potřebné kroky pro její řešení.
4. Případně může administrátor přidělit úkol pro opravu chyby technickému týmu.

Podmínky po spuštění

- Administrátor získal přehled o akcích a chybách v systému, což mu umožňuje podniknout kroky k jejich řešení a zlepšení bezpečnosti a stability systému.

UC14 - Integrace s bankovním API

Aktéři

- Systém
- Bankovní API

Podmínky před spuštěním

- Systém je správně nakonfigurován pro komunikaci s bankovním API.

Základní tok

1. Systém iniciuje požadavek na transakci (např. výplatu pojistného plnění) prostřednictvím bankovního API.
2. Poskytne všechny potřebné informace pro transakci, včetně částky, údajů o účtu příjemce a případně dalších specifických informací požadovaných bankou.
3. Bankovní API zpracuje požadavek a vrátí odpověď, zda byla transakce úspěšně provedena nebo odmítnuta.
4. Systém zaznamená výsledek transakce a aktualizuje stav pojistné události nebo plnění v systému.

Podmínky po spuštění

- Transakce byla úspěšně zpracována bankovním API, a systém má zaznamenány všechny relevantní informace o provedené transakci.

UC15 - Správa rolí a oprávnění

Aktéři

- Administrátor
- Systém

Podmínky před spuštěním

- Administrátor je přihlášen do systému.

Základní tok

1. Administrátor otevře sekci pro správu rolí a oprávnění.
2. Prohlíží seznam dostupných rolí a jejich aktuální oprávnění.
3. Pro přidání nové role vybere možnost "Přidat roli" a definuje její oprávnění.
4. Pro úpravu oprávnění stávající role vybere roli a upraví její oprávnění.
5. Pro odstranění role vybere stávající roli a potvrdí její vymazání.
6. Systém provede změny a aktualizuje konfiguraci rolí a oprávnění.

Podmínky po spuštění

- Role a oprávnění jsou aktualizovány dle změn provedených administrátorem, což ovlivní přístupová práva uživatelů v systému.

UC16 - Auditování a sledování bezpečnosti

Aktéři

- Systém
- Administrátor

Podmínky před spuštěním

- Systém je nakonfigurován pro zaznamenávání a auditování bezpečnostních událostí.

Základní tok

- Systém automaticky zaznamenává všechny bezpečnostní události, jako jsou přihlášení uživatelů, pokusy o neautorizovaný přístup a změny systémových nastavení.
- Administrátor pravidelně přistupuje k logům bezpečnostních událostí prostřednictvím administračního rozhraní.
- Administrátor prochází a analyzuje záznamy k identifikaci podezřelých nebo neautorizovaných aktivit.
- V případě zjištění bezpečnostního incidentu administrátor podnikne odpovídající kroky k řešení a zabezpečení systému.

Alternativní tok

- Administrátor může nastavit upozornění nebo pravidla pro automatické informování o určitých typech událostí, což usnadní rychlou reakci na potenciální hrozby.

Podmínky po spuštění

- Bezpečnostní události jsou průběžně monitorovány a auditovány, což zvyšuje celkovou bezpečnost a odolnost systému proti hrozbám.