FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Aplikace pro správu smluv v zastavárně
Student:	Xuan Trung Pham
Vedoucí:	Ing. Michal Valenta, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Cílem práce je návrh a implementace aplikace pro správu smluv ve specifické doméně – provozu zastavárny.

1. Níže uvedené funkční a nefunkční požadavky rozpracujte a dopřesněte.
2. Proveďte rešerši aplikací, které se věnují této doméně, existují-li nějaké.
3. Proveďte návrh vlastního řešení.
4. Zvolte vhodnou implementační platformu.
5. Návrh implementujte formou funkčního prototypu, otestujte s budoucími uživateli.
6. Zhodnoťte výsledek, navrhněte další rozvoj.

Požadavky:
- vytváření, úprava, tisk smluv,
- rychlé vyhledávání ve smlouvách,
- kalkulace zboží na základě průzkumu cen na internetu,
- aplikace bude dostupná na více zařízeních.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 22. října 2021 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Aplikace pro správu smluv v zastavárně

Xuan Trung Pham

Katedra softwarového inženýrství
Vedoucí práce: Ing. Michal Valenta, Ph.D.

3. května 2022

Poděkování

V první řadě chci poděkovat svému vedoucímu závěrečné bakalářské práce Ing. Michalu Valentovi, Ph.D. především za jeho ochotu mě v tomto projektu podpořit a jeho trpělivost během psaní bakalářské práce.

Dále chci věnovat poděkování svým rodičům za možnost vytvoření aplikace pro správu smluv v zastavárně a také za jejich výbornou spolupráci, otevřenost a trpělivost při vytváření požadavků pro výslednou aplikaci.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 3. května 2022 . . . . . .

Odkaz na tuto práci

Pham, Xuan Trung. *Aplikace pro správu smluv v zastavárně*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2022.

Abstrakt

Tato bakalářská práce se zabývá analýzou, návrhem a implementací aplikace pro malý podnik provozující zastavárenskou činnost. Výsledná aplikace, která je nasazená do podniku, podporuje provozovatele zastavárny ve správě vedených smluv a zákazníků, kalkulaci ceny zboží na základě průzkumu internetu a analýze různých dat o smlouvách či zákazníků.

Klíčová slova desktopová aplikace, správa smluv, správa zákazníků, analytika dat, web scraping, JavaFx

Abstract

The bachelor thesis deals with analysis, design and implementation of an application for a small business running a pawn shop. The finished application that is deployed in the business itself supports pawn shop managers to manage maintained contracts and customers, calculate price of an item based on internet survey and analyse various different data about contracts or customers.

Keywords desktop application, contract management, customer management, data analytics, web scraping, JavaFx

Obsah

Úvod **1**

1 Cíl práce **3**

2 Analýza **5**
2.1 Analýza stávajících řešení 5
2.1.1 Contract Zen 5
2.1.2 PandaDoc 6
2.1.3 Concord 7
2.1.4 Závěr 7
2.2 Požadavky na aplikaci 7
2.2.1 Funkční požadavky 7
2.2.2 Nefunkční požadavky 10
2.3 Případy užití 11
2.3.1 Mapování případů užití na funkční požadavky 15

3 Návrh **17**
3.1 Doménový model 17
3.2 Architektura aplikace 18
3.2.1 Typ architektury 18
3.2.2 Technologie pro uživatelské rozhraní 20
3.2.3 Model uživatelského rozhraní 21
3.2.4 Technologie pro průzkum cen na internetu 22
3.2.5 Databáze 23
3.2.6 Zálohování dat 23
3.2.7 Závěr 24
3.3 Uživatelské rozhraní 24

4 Implementace **27**
4.1 Nástroje a technologie 27

4.1.1 GitHub . . . 27
4.1.2 Nástroje pro podporu vývoje . . . 27
4.1.3 JavaFx . . . 28
4.1.4 HtmlUnit . . . 28
4.1.5 Hibernate . . . 29
4.2 Nasazení a spuštění aplikace . . . 30
4.2.1 Apache Maven . . . 30
4.2.2 JPackage . . . 31
4.3 Testování . . . 32
4.3.1 JUnit 5 . . . 32
4.3.2 Mockito . . . 32
4.3.3 TestFx . . . 33
4.3.4 HSQLDB . . . 34
4.3.5 Uživatelské testování . . . 35
4.4 Závěr . . . 35

5 Budoucí rozšíření **37**
5.1 Offline mód . . . 37
5.1.1 Nahrazení „internetových“ částí systému . . . 37
5.1.2 Integrace . . . 38
5.2 Synchronizace dat . . . 39
5.2.1 Návrh řešení . . . 39

Závěr **43**

Literatura **45**

A Snímky výsledné aplikace **49**

B Seznam použitých zkratek **55**

C Obsah přiloženého SD **57**

Seznam obrázků

2.1 Aplikace Contract Zen 6
2.2 Aplikace PandaDoc 6

3.1 Doménový model 17
3.2 2-tier architektura 18
3.3 3-tier architektura 19
3.4 Model Supervising Controller 21
3.5 Shared Model 21
3.6 Presentation Model 22
3.7 Proces zálohování dat 23
3.8 Výsledná architektura aplikace 24
3.9 Návrh uživatelského rozhraní 1 25
3.10 Návrh uživatelského rozhraní 2 26

4.1 Výstup z terminálu po zadání příkazu mvn test 32

A.1 Snímek aplikace – Domovská stránka 49
A.2 Snímek aplikace – Zobrazení smluv 50
A.3 Snímek aplikace – Vytvoření smlouvy s web scaraping funkcí . . . 50
A.4 Snímek aplikace – Zobrazení zákazníků 51
A.5 Snímek aplikace – Vytvoření zákazníka s analytikou 51
A.6 Snímek aplikace – Zobrazení smluv a jejich stavů 52
A.7 Snímek aplikace – Zobrazení historie prodloužení smlouvy 52
A.8 Snímek aplikace – Kalkulace příjmů, výdajů a výnosu či ztráty . . 53
A.9 Snímek aplikace – Analytika smluv 53

Seznam tabulek

2.1 Tabulka mapování případů užití na funkční požadavky 16

4.1 Tabulka chyb z uživatelského testování 35

Seznam zrojových kódů

4.1 Tlačítko definované v FXML souboru 28
4.2 Funkce reagující na stisknutí tlačítka 28
4.3 Extrahování cen z Google reklam 29
4.4 Příklad užití frameworku Hibernate pro uložení entity 29
4.5 Použití shade pluginu v pom.xml 30
4.6 JPackage vytvoření instalačního balíčku pro Windows 31
4.7 JPackage vytvoření instalačního balíčku pro MacOS 31
4.8 Unit testování s frameworkem Mockito 32
4.9 Testování s frameworkem TestFx 33
4.10 Konfigurace souboru hibernate.cfg.xml pro testování 34
5.1 Konfigurace souboru hibernate_fallback_solution.cfg.xml 38
5.2 Příklad řešení přepínání mezi lokální a vzdálenou databází . . . 39
5.3 Příklad podmínky pro zavolání funkce pro překopírování 39
5.4 Příklad funkce pro překopírování dat 40

Úvod

V dnešní době drtivá většina podniků, které nějakým způsobem uzavírají dohodu se svými zákazníky, používá softwarový systém právě pro správu těchto dohod. Ať už jde o zakázky na výrobu, členství v klubech nebo samotné smlouvy, společným důvodem pro zřízení takového systému je to, že ruční správa těchto dat je při velkém množství velmi náročná.

Tomuto problému čelí i malý podnik vykonávající zastavárenskou činnost v Chebu. Protože se v zastavárně neustále podepisují nové smlouvy, mít pořádek a přehled ve všech smlouvách vyžaduje dobře vedené uložiště kopií všech podepsaných smluv. Momentálně si podnik vede uložiště smluv ve fyzické formě. Aby mohli provozovatelé hledat ve smlouvách přiměřeně rychle, vedou si různé sešity, do kterých pravidelně stručně přepisují obsah smluv, aby měli informace jednoduše k dispozici a nemuseli neustále vytahovat fyzické kopie z uložiště.

Takto vedený systém je velmi neefektivní. Proto nasazení systému pro správu smluv by bylo pro podnik a provozovatele velmi přínosné. Nejen že by práce se smlouvami byla velmi rychlá a snadná, ale také by se ušetřilo na papírových sešitech, které se používaly jako dočasné řešení.

Motivací pro výběr právě tohoto tématu bakalářské práce bylo pro autora především řešení reálného problému formou funkčního softwarového prototypu. Autorovi se také zalíbila možnost zavedení dalších funkcí nad rámec spravování smluv jako analýza měsíčních příjmů a výdajů nebo kalkulace ceny zboží na základě průzkumu cen na internetu.

V první kapitole je nejprve provedena analýza existujících podobných řešení na trhu existují-li nějaká. Po té jsou na základě konzultací s provozovateli zastavárny sestaveny funkční a nefunkční požadavky na systém a případy užití.

Ve druhé kapitole je na základě výstupu z první kapitoly popsán návrh aplikace, který obsahuje popis doménového modelu, výběr specifických technologií a zvolení architektury.

V další kapitole je provedena samotná implementace aplikace a v kapitole jsou popsány všechny technologie, knihovny a vývojová prostředí, které byly pro implementaci aplikace použity. Nakonec je tu popsáno nasazení, testování aplikace a konečné vyhodnocení implementační části.

Nakonec je v poslední kapitole stručně popsáno budoucí možné rozšíření aplikace a stručné nastínění nástrojů a postupů pro možnou implementaci.

Kapitola 1

Cíl práce

Hlavním cílem této bakalářské práce je poskytnout řešení reálného problému správy smluv v zastavárně formou funkčního prototypu, který bude řádně navržen, implementován, otestován a nasazen. Dílčím cílem je sepsání všech požadavků na systém, ze kterých se provede návrh systému.

Výstupem implementace bude funkční prototyp, který bude nasazen do podniku podle jeho potřeb.

Kapitola 2

Analýza

Tato kapitola obsahuje analytickou část práce. Nejprve se provedla rešerše aplikací, které se věnují této doméně, existují-li nějaké.

Po analýze existujících řešení na trhu se na základě konzultací s provozovateli zastavárny sepsaly všechny funkční a nefunkční požadavky na systém a případy užití.

Nakonec se z funkčních požadavků a případů užití sestavila tabulka, která přesně mapuje případy užití na funkční požadavky.

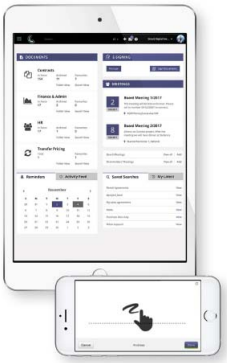
2.1 Analýza stávajících řešení

Aplikací, které řeší problém správy smluv či dokumentů, je na trhu hodně a každá z těchto aplikací přichází s různým řešením, vzhledem a různými funkcionalitami navíc nad rámec správy dokumentů, které je od své kompetice odlišují. Protože je těchto aplikací hodně, rozhodl se autor uvést analýzu tří následujících aplikací, které jsou od sebe svým vzhledem, řešením a funkcionalitami nejodlišnější.

2.1.1 Contract Zen

Contract Zen je webová aplikace, jejíž hlavní funkcí je správa smluv. Contract Zen také poskytuje řadu dalších funkcí jako jsou například různé nástroje ke spravování schůzek, elektronické podpisy a mnoho dalších. Po registraci obdrží uživatel zkušební dobu po dobu jednoho měsíce, během níž má uživatel plný přístup ke Contract Zen [1].

Navzdory tomu, že aplikace poskytuje mnoho funkcí a nástrojů, je Contract Zen na první pohled velmi matoucí a zastaralý. Uživatelské rozhraní není intuitivní a nadměrný počet funkcí a nástrojů dělají aplikaci nepřehlednou. Aplikace také neumožňuje úpravu, vytváření smluv z uložených šablon nebo export smluv pro následné tisknutí [1].

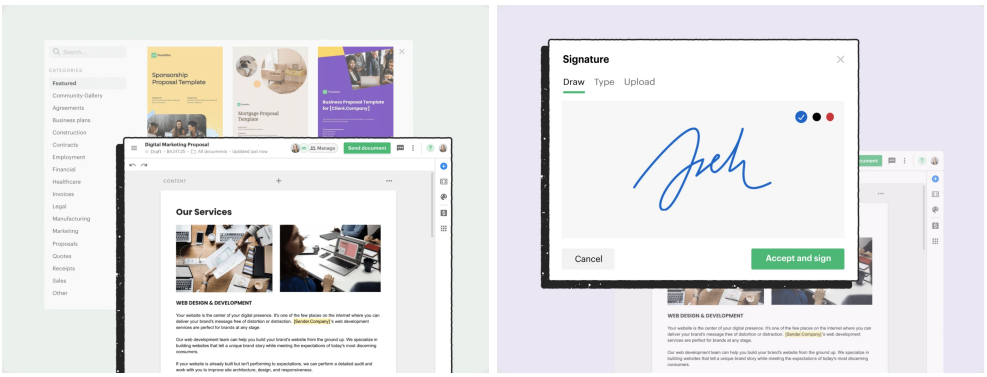


Obrázek 2.1: Aplikace Contract Zen [1]

2.1.2 PandaDoc

PandaDoc poskytuje svým uživatelům platformu pro automatizaci pracovních postupů potřebných pro udržování životního cyklu dokumentů. Tato platforma pomáhá rychle rostoucím firmám jednoduše a rychle spravovat důležité dokumenty na jednom místě od vytváření dokumentů ze šablon po uzavírání digitálních smluv elektronickým podpisem [2].

PandaDoc nabízí i bezplatnou verzi, která umožňuje neomezené nahrávání a spravování smluv, vyřizování plateb a možnost správy smluv i přes mobilní zařízení. Jak webové tak mobilní rozhraní aplikace jsou velmi moderní, responzivní a dobře navržena. Aplikace se jednoduše používá, je velmi intuitivní a poskytuje všechno potřebné ke spravování smluv. Nevýhodou bezplatné verze je nemožnost použití šablon, které jsou již zpoplatněny, a samotné vytváření smluv. Smlouvy v bezplatné verzi se mohou pouze nahrávat ve formátu PDF, Word, PowerPoint, JPG nebo PNG [2].



Obrázek 2.2: Aplikace PandaDoc [2]

2.1.3 Concord

Concord je cloudová platforma pro správu životního cyklu smluv, která kromě správy také poskytuje online vyjednávání s klientem a možnost uzavření dohody digitálním podpisem. Concord také umožňuje různé integrace například s Google Docs, Salesforce nebo Dropbox [3].

Aplikace působí velmi profesionálně s čistým a přehledným designem. Umožňuje také jednoduché vytváření nových smluv ze šablon, podepisování nebo měnit stav smluv. Concord také nabízí reporty a analytika vybraných dat týkajících se spravovaných dokumentů. Největší nevýhodou je cena, kterou je aplikace zpoplatněná. Kromě toho si uživatelé Concord nemohou zobrazit v analytice například měsíční výdaje nebo výnosy z vybraných smluv, které jsou pro firmu velmi důležitými daty [3].

2.1.4 Závěr

Drtivá většina stávajících řešení na trhu je zpoplatněna a nenabízí všechny potřebné funkcionality. Z analýzy tedy vyplývá, že z části trhu, která byla analyzována, neexistuje žádné řešení, které by zcela vyřešilo reálný problém provozovatelů zastavárny.

Výsledek této analýzy byl ale do jisté míry očekáván, jelikož se autor zabývá návrhem systému pro správu smluv ve velmi specifické doméně s ještě specifičtějšími funkcemi.

2.2 Požadavky na aplikaci

Požadavky definují obsah a rozsah práce. Dělí se na požadavky funkční, které se týkají funkcionalit systému, a nefunkční, které se týkají vlastností systému, její omezení nebo požadavky na řešení.

Všechny požadavky jsou sestaveny na základě konzultací s provozovateli zastavárny a autorova vlastní zkušenosti v procesech podniku zastavárny.

2.2.1 Funkční požadavky

Pro určení priorit funkčních požadavků je použita metoda MoSCoW [4], která rozděluje požadavky do čtyř kategorií:

- MUST HAVE – nezbytné pro akceptování systému,
- SHOULD HAVE – důležité požadavky, které přidávají systému velkou hodnotu. Mohou být ignorovány,
- COULD HAVE – požadavky, které jsou volitelné. Mohou usnadnit práci se systémem,
- WOUNT HAVE – požadavky, které nebudou implementovány.

V rámci analýzy požadavků na systém pro správu smluv se využijou pouze první tři kategorie.

2.2.1.1 FP-1 Evidence smluv

Smlouvy hrají nezbytnou roli ve fungování zastavárny. Představují vztah mezi zákazníkem (zástavcem) a zastavárnou. Jako každý podnikatel i provozovatelé zastavárny potřebují mít přehled nad svým podnikem, proto důležitou součástí systému je také stručný náhled všech smluv ve formě seznamu. Požadavky na evidenci smluv jsou:

- FP-1.1 – vytvoření smlouvy [MUST HAVE],
- FP-1.2 – čtení smlouvy [MUST HAVE],
- FP-1.3 – náhled všech smluv jako seznam [MUST HAVE],
- FP-1.4 – úprava smlouvy [MUST HAVE],
- FP-1.5 – smazání smlouvy [MUST HAVE].

2.2.1.2 FP-2 Vyhledávání smluv

Protože se v zastavárně pracuje se smlouvami dennodenně, je nezbytně nutné je umět vyhledat podle různých parametrů. Požadavky pro vyhledávání smluv jsou následující:

- FP-2.1 – vyhledávání podle datu zástavy [MUST HAVE]
- FP-2.2 – vyhledávání podle jména zákazníka [MUST HAVE]
- FP-2.3 – vyhledávání podle rodného čísla zákazníka [MUST HAVE]
- FP-2.4 – vyhledávání podle ID smlouvy [MUST HAVE]
- FP-2.5 – vyhledávání podle značky zboží [COULD HAVE]

2.2.1.3 FP-3 Potvrzení smluv

Zákazníci mohou buďto zboží vybrat včas nebo ho nemusí vybrat vůbec. Aby provozovatelé věděli jaké smlouvy jsou ještě relevantní, musí mít možnost je potvrdit a považovat je za vyřízené. Požadavky pro potvrzení smluv jsou následující:

- FP-3.1 – potvrzení o výběru zboží [MUST HAVE]
- FP-3.2 – potvrzení o propadnutí zboží [MUST HAVE]

2.2.1.4 FP-4 Evidence zákazníků

Žádný podnik se bez zákazníků neobejde, proto vedení evidence zákazníků nese velkou hodnotu. Protože se smlouva vždy vztahuje k právě jednomu zástavci, není evidence zákazníků pro akceptování systému nutná. Nicméně zavedení této evidence přináší velkou hodnotu. Požadavky na evidenci zákazníků jsou následující:

- FP-4.1 – přidání nového zákazníka [SHOULD HAVE]
- FP-4.2 – čtení údajů o zákazníkovi [SHOULD HAVE]
- FP-4.3 – úprava údajů o zákazníkovi [SHOULD HAVE]
- FP-4.4 – smazání zákazníka [SHOULD HAVE]

2.2.1.5 FP-5 Vyhledávání zákazníků

Požadavky pro vyhledávání zákazníků jsou následující:

- FP-5.1 – vyhledávání podle jména [SHOULD HAVE]
- FP-5.2 – vyhledávání podle rodného čísla zákazníka [SHOULD HAVE]

2.2.1.6 FP-6 Výpočet cen

Zastavárna je postavena na půjčování peněz, proto je nezbytně nutné mít neustále pod kontrolou kolik peněz se půjčuje každý den. Kromě výpočtu výdajů je také důležitý výpočet ceny zboží při naceňování. Protože do zastavárny chodí nejčastěji zákazníci zastavovat různou elektroniku, je obtížné vždy přesně nacenit zákazníkovo zboží bez žádné představy o průměrné ceně zboží na trhu. Požadavky pro výpočet cen jsou:

- FP-6.1 – výpočet ceny zboží na základě průzkumu cen na internetu [MUST HAVE]
- FP-6.2 – výpočet celkové ceny při výběru zboží s úroky [MUST HAVE]
- FP-6.3 – výpočet výdajů z vybraných smluv [MUST HAVE]
- FP-6.4 – výpočet příjmů z vybraných smluv [COULD HAVE]

2.2.1.7 FP-7 Notifikace

Každá smlouva je podepsána na dobu určitou. Smlouva tedy po uplynutí určité doby propadne a zastavárně nabyde vlastnické právo k danému zboží uvedené ve smlouvě. Požadavky pro notifikace jsou:

- FP-7.1 – notifikace na propadlou smlouvu [COULD HAVE]
- FP-7.2 – zobrazení notifikace [COULD HAVE]

2.2.1.8 FP-8 Tisk

Smlouva v zastavárně se zásadně uzavírá podpisem obou smluvních stran – zástavce a zastavárny. Smlouva se vyhotovuje ve dvou stejnopisech. Každá smluvní strana obdrží jeden stejnopis smlouvy. Požadavky pro tisk jsou:

- FP-8.1 – tisk smluv [MUST HAVE]
- FP-8.2 – tisk seznamu smluv [COULD HAVE]

2.2.1.9 FP-9 Analytika smluv

Jako každý podnikatel potřebuje provozovatel zastavárny umět porovnat svoje výdaje, příjmy, výnosy a ztráty v různých měsících daného roku nebo v různých letech. Proto jsou požadavky pro analytika následující:

- FP-9.1 – analýza výdajů, příjmů, výnosů a ztrát v různých měsících daného roku [MUST HAVE]
- FP-9.2 – analýza výdajů, příjmů, výnosů a ztrát v různých letech [MUST HAVE]
- FP-9.3 – analýza počtu smluv a jejich stavů v daném roce či letech [MUST HAVE]

2.2.1.10 FP-10 Analytika zákazníků

Zákazníci jsou v každém podnikání největší prioritou. Aby provozovatel zastavárny mohl zákazníkům vyjít co nejvíce vstříc, musí vědět, kolik smluv nechal či nenechal zákazník propadnout a kolik smluv má ještě momentálně validních. Požadavky pro analytika zákazníků jsou:

- FP-10.1 – analýza počtu smluv a jejich stavů u daného zákazníka [MUST HAVE]
- FP-10.2 – analýza počtu smluv v různých měsících daného roku nebo v různých letech [MUST HAVE]
- FP-10.3 – analýza příjmů ze smluv daného zákazníka [COULD HAVE]

2.2.2 Nefunkční požadavky

Seznam nefunkčních požadavků je následující:

2.2.2.1 NFP-1 Dostupnost systému na více zařízeních

Systém bude dostupný na více zařízeních najednou pro více uživatelů.

2.2.2.2 NFP-2 Rychlé vyhledávání ve smlouvách

Systém bude umožňovat rychlé vyhledávání ve smlouvách dle vybraných parametrů.

2.2.2.3 NFP-3 Rychlý výpočet ceny zboží na základě průzkumu cen na internetu

Systém bude umožňovat rychlý výpočet ceny zboží na základě průzkumu cen na internetu v řádech desítek sekund.

2.2.2.4 NFP-4 Zálohování dat

Systém bude pravidelně data zálohovat vždy jednou denně.

2.2.2.5 NFP-5 Snadná rozšiřitelnost

Systém bude snadně rozšiřitelný pro další budoucí rozvoj systému.

2.3 Případy užití

Z diskuse s provozovateli zastavárny plynne, že rozdělení rolí v systému je nepodstatné. Přístup do systému se všemi pravomocemi budou mít pouze provozovatelé zastavárny. Nicméně v případě potřeby přidání dalších rolí bude systém jednoduše rozšiřitelný.

PU-1 Vytvoření nové smlouvy pro nového zákazníka

Uživatel vytvoří novou smlouvu pro nového zákazníka. Po vytvoření smlouvy se uloží smlouva i zákazník do systému.

PU-2 Vytvoření nové smlouvy pro stálého zákazníka s novým zbožím ze šablony

Uživatel vyhledá v systému stálého zákazníka a ze šablony zákazníka vytvoří novou smlouvu s před vyplněnými údaji o zákazníkovi. Po vytvoření smlouvy se uloží smlouva do systému.

PU-3 Vytvoření nové smlouvy pro stálého zákazníka se stálým zbožím ze šablony

Uživatel vyhledá v systému smlouvu na jméno stálého zákazníka s odpovídajícím stálým zbožím a ze šablony smlouvy vytvoří smlouvu novou s před vyplněnými údaji o zákazníkovi a informacemi o zboží. Po vytvoření se uloží smlouva do systému.

PU-4 Úprava smlouvy

Uživatel vyhledá smlouvu v systému a upraví ji. Po úpravě se nově upravené změny uloží do systému.

PU-5 Prodloužení smlouvy

Uživatel vyhledá smlouvu v systému a prodlouží zákazníkovi smlouvu. Prodloužením se automaticky dopočítají a přičtou úroky a provedené změny se uloží do systému.

PU-6 Smazání smlouvy

Uživatel vyhledá smlouvu v systému a smaže ji.

PU-7 Zobrazení náhledu všech smluv jako seznam

Uživatel si zobrazí stručný náhled všech smluv, ve kterém si prochází smlouvy jako v seznamu. Seznam slouží primárně pro rychlý přehled propadlých a nepropadlých smluv.

PU-8 Vyhledání smlouvy podle ID smlouvy

Uživatel vyhledá smlouvu v systému podle ID smlouvy.

PU-9 Vyhledání smlouvy podle datu zástavy

Uživatel vyhledá smlouvy v systému podle datu zástavy.

PU-10 Vyhledání smlouvy podle jména zákazníka

Uživatel vyhledá smlouvy v systému podle jména zákazníka.

PU-11 Vyhledání smlouvy podle rodného čísla zákazníka

Uživatel vyhledá smlouvy v systému podle rodného čísla zákazníka.

PU-12 Tisk smlouvy

Uživatel buď nejprve smlouvu vyhledá v systému a vytiskne ji nebo ji vytiskne ihned po vytvoření.

PU-13 Tisk náhledu všech smluv jako seznam

Uživatel si vytiskne náhled vybraných smluv jako seznam.

PU-14 Výpočet výdajů z vybraných smluv

Uživatel si nechá vypočítat celkové výdaje z vybraných smluv.

PU-15 Výpočet celkové ceny s úroky při výběru zboží ze zastavárny

Uživatel si nechá vypočítat celkovou cenu s úroky při výběru zboží ze zastavárny při vytváření nové smlouvy nebo při prodlužování smlouvy.

PU-16 Výpočet ceny zboží na základě průzkumu cen na internetu

Při naceňování si uživatel nechá vypočítat cenu zákazníkova zboží na základě průzkumu cen na internetu.

PU-17 Vytvoření nového zákazníka

Uživatel vytvoří nového zákazníka. Po vytvoření se zákazník uloží do systému.

PU-18 Úprava údajů zákazníka

Uživatel vyhledá zákazníka v systému a upraví jeho osobní údaje.

PU-19 Smazání zákazníka

Uživatel vyhledá zákazníka v systému a smaže ho.

PU-20 Vyhledání zákazníka podle jména

Uživatel vyhledá zákazníka v systému podle jeho jména.

PU-21 Vyhledání zákazníka podle rodného čísla

Uživatel vyhledá zákazníka v systému podle jeho rodného čísla.

PU-22 Zobrazení notifikace

Uživatel si zobrazí notifikace o propadlých smlouvách. Po potvrzení smluv se notifikace automaticky vypnou.

PU-23 Potvrzení o propadnutí zboží

Uživatel potvrdí smlouvu, která zákazníkovi propadla. Po potvrzení se stav smlouvy přepíše na potvrzeno a případné notifikace se vypnou.

PU-24 Potvrzení o výběru zboží

Uživatel potvrdí smlouvu, u které si zástavce pro zboží přišel. Po potvrzení se stav smlouvy přepíše na potvrzeno.

PU-25 Analýza výdajů, příjmů, výnosů a ztrát smluv v různých měsících daného roku

Uživatel napíše rok, ze kterého chce udělat analýzu smluv, a nechá si provést analýzu výdajů, příjmů, výnosů a ztrát v různých měsících daného roku.

PU-26 Analýza výdajů, příjmů, výnosů a ztrát smluv v různých letech

Uživatel napíše interval v letech, ve kterém se má provést analýza, a nechá si provést analýzu výdajů, příjmů, výnosů a ztrát v různých letech.

PU-27 Analýza počtu smluv a jejich stavů v daném roce

Uživatel si nechá udělat analýzu počtu smluv a jejich stavů v zadaném roce.

PU-28 Analýza počtu smluv a jejich stavů v různých letech

Uživatel si nechá udělat analýzu počtu smluv a jejich stavů v zadaném intervalu.

PU-29 Analýza počtu smluv a jejich stavů u daného zákazníka

Uživatel si zobrazí počet smluv a jejich stavy u vybraného zákazníka.

PU-30 Analýza počtu smluv daného zákazníka v různých měsících daného roku

Uživatel si nechá udělat analýzu počtu smluv daného zákazníka v různých měsících daného roku.

PU-31 Analýza počtu smluv daného zákazníka v různých letech

Uživatel si nechá udělat analýzu počtu smluv daného zákazníka v zadaném intervalu.

PU-32 Analýza příjmů smluv daného zákazníka v různých měsících daného roku

Uživatel si nechá udělat analýzu příjmů smluv daného zákazníka v různých měsících daného roku.

PU-33 Analýza příjmů smluv daného zákazníka v různých letech

Uživatel si nechá udělat analýzu příjmů smluv daného zákazníka v zadaném intervalu.

2.3.1 Mapování případů užití na funkční požadavky

Pro přehlednost slouží následující mapovací tabulka 2.1, kde každý funkční požadavek musí pokrývat alespoň jeden případ užití.

Tabulka 2.1: Tabulka mapování případů užití na funkční požadavky

	FP-1	FP-2	FP-3	FP-4	FP-5	FP-6	FP-7	FP-8	FP-9	FP-10
PU-1	X									
PU-2	X									
PU-3	X									
PU-4	X									
PU-5	X									
PU-6	X									
PU-7	X									
PU-8		X								
PU-9		X								
PU-10		X								
PU-11		X								
PU-12								X		
PU-13								X		
PU-14						X				
PU-15						X				
PU-16						X				
PU-17				X						
PU-18				X						
PU-19				X						
PU-20					X					
PU-21					X					
PU-22							X			
PU-23			X							
PU-24			X							
PU-25									X	
PU-26									X	
PU-27									X	
PU-28									X	
PU-29										X
PU-30										X
PU-31										X
PU-32										X
PU-33										X

Kapitola 3

Návrh

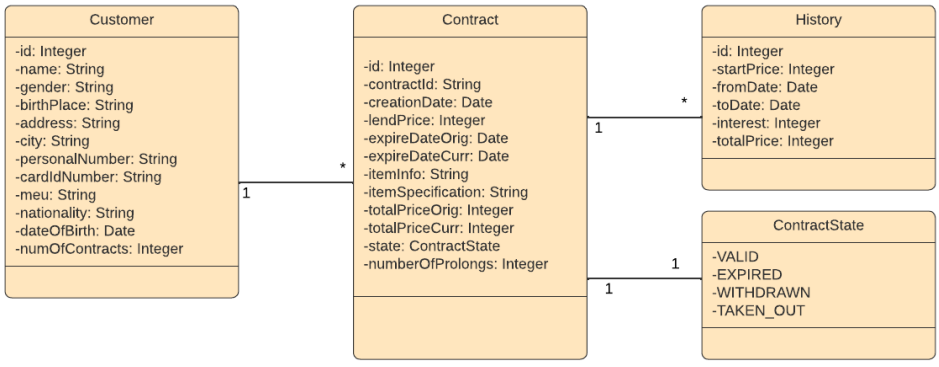
V této kapitole se popisuje celý návrh aplikace pro správu smluv a výběr hlavních technologií pro implementaci. Nejprve se popíše doménový model a jeho entity.

V další části se rozebere architektura aplikace, jaké jsou možnosti a pro jakou možnost se rozhodlo pro implementaci. Dále se stručně popíší volby a důvod pro výběr určitých technologií.

Na závěr se ve stručnosti popíše grafické rozhraní prototypu a použitý nástroj pro design.

3.1 Doménový model

V následujícím doménovém modelu 3.1 jsou popsány jednotlivé entity, jejich atributy a jejich vztahy mezi sebou.



Obrázek 3.1: Doménový model

3.2 Architektura aplikace

Další důležitou částí návrhu je architektura aplikace. Aby aplikace byla dostupná a funkční na více zařízeních, musí být od ní oddělena alespoň datová část aplikace.

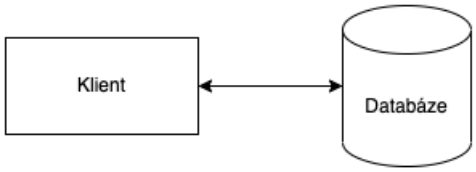
3.2.1 Typ architektury

Architekturu aplikace pro správu smluv lze navrhnout jako 2-tier (dvouvrstvou) nebo 3-tier (třívrstvou) architekturu. Každá z těchto možností přináší své výhody a nevýhody.

3.2.1.1 2-tier architektura

Jak z názvu vyplývá, 2-tier neboli dvouvrstvá architektura je architektura, která obsahuje dvě vrstvy:

1. Klient – aplikace na koncovém zařízení uživatele, která zajišťuje grafické uživatelské rozhraní a aplikační logiku,
2. Datová vrstva – databáze nasazená na server pro ukládání dat.



Obrázek 3.2: 2-tier architektura

Výhodou 2-tier aplikace je:

- jednoduchá implementace
- jednoduché nasazení
- jednoduchá údržba aplikace

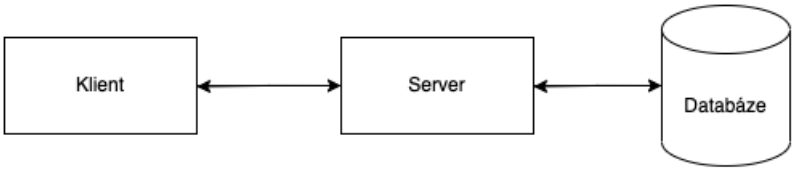
Naopak nevýhodami jsou:

- špatná škálovatelnost
- špatný výkon při velkém množství uživatelů
- výkon závislý na hardwaru uživatele

3.2.1.2 3-tier architektura

Narozdíl od 2-tier architektury, tak 3-tier neboli třívrstvá architektura obsahuje o jednu vrstvu více a člení se na:

1. Prezentační - viditelná vrstva pro uživatele, zajištuje vstupy od uživatelů a prezentuje uživatelům data,
2. Aplikační - logická vrstva, která má na starost veškerou logiku aplikace,
3. Datová - databáze na serveru pro ukládání dat.



Obrázek 3.3: 3-tier architektura

Výhodou 3-tier aplikace je:

- velmi dobrá škálovatelnost
- dobrá znovupoužitelnost kódu
- nezávislost vrstev
- dobrý výkon nezávisle na hardwaru uživatele

Hlavními nevýhodami 3-tier aplikace jsou:

- vyšší náročnost implementace

3.2.1.3 Vybraný typ architektury

Z předešlých kapitol o 2-tier a 3-tier lze říci, že výběr 3-tier architektury bude pro celkovou aplikaci, její škálovatelnost a výkon dobrým rozhodnutím. Nicméně autor se nakonec rozhodl pro 2-tier architektury z následujících důvodů:

1. pro co nejrychlejší implementaci, otestování s uživateli a nasazení prototypu do podniku se autor rozhodl více přiklonit k architektuře 2-tier pro rychlejší a snazší implementaci, testování a nasazení,

2. hlavní nevýhodou 2-tier architektury je její škálovatelnost při vysokém počtu uživatelů. Tato situace určitě nenastane, jelikož výsledný prototyp budou používat pouze provozovatelé podniku - tedy dva uživatelé každý na svém zařízení,

3. pro budoucí rozšíření aplikace o takzvané offline používání se 2-tier aplikace, která zajišťuje uživatelské rozhraní a aplikační logiku na straně klienta, jeví jako atraktivnějším řešením pro tuto funkcionalitu.

Body uvedené výše byly důvodem autorova rozhodnutí pro 2-tier architekturu namísto 3-tier architektury.

3.2.2 Technologie pro uživatelské rozhraní

Výslednou aplikaci budou provozovatelé zastavárny používat, tudíž s ní budou nějakým způsobem manipulovat. Aby tato interakce byla možná a co nejjednodušší pro koncové uživatele, musí se dobře navrhnout uživatelské rozhraní a vybrat správná technologie pro implementaci. Nejprve se vybere technologie pro implementaci. Následující seznam obsahuje všechny technologie a jejich stručný popis, u kterých se autor rozhodoval:

- **AWT** neboli Abstract Window Toolkit je knihovna grafických prvků pro platformu Java a je její součástí od JDK 1.0 [5] Knihovna nabízí nativní sadu komponent uživatelského rozhraní, robustní model zpracovávání eventů nebo různé grafické a zobrazovací nástroje [6].

- **Swing** je populární knihovnou poskytující sadu tříd pro implementaci grafických uživatelských rozhraní, která je postavena na základě AWT architektury [6]. Protože je Swing novější než AWT, nabízí knihovna širší a bohatší výběr komponent.

- **SwingX** je dle [7] rozšířením Swing API, který je součástí open source projektu swingLabs. Cílem SwingX bylo obohatit Swing o sofistikovanější komponenty a funkce, nicméně na projektu se již nepracuje. Poslední verzí SwingX byla verze 1.6.5 [8].

- **JGoodies** poskytují různé knihovny a software pro jednoduší implementaci Swing aplikací. JGoodies knihovny jako jsou Binding, Forms, `Looks`, a Validation poskytují mnoho zajímavých funkcí, nicméně `JGoodies` knihovny nemají skoro žádnou dokumentaci. [9].

- **JavaFx** [10] je open source, next generation platforma postavená na Javě, která umožňuje vývoj aplikací s moderním grafickým uživatelským rozhraním. JavaFx poskytuje vývojářům širokou nabídku různých API formou modulů jako jsou například javafx.base, javafx.controls nebo `javafx.fxml`.

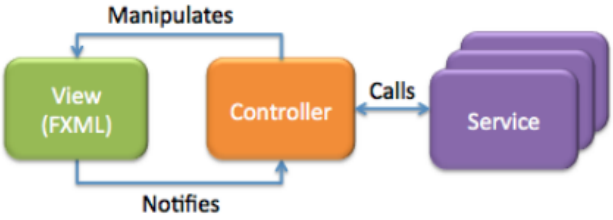
Z uvedených možností výše se AWT a SwingX již nepoužívají a JGoodies knihovny nemají skoro žádnou dokumentaci. Jedinou reálnou možností je buď Swing nebo JavaFx. Oproti Swing přináší JavaFx podporu grafů, CSS stylů a dalších technologií, které zajistí líbivost aplikace. I přes to, že od Javy 11 již JavaFx není součástí JDK, jsou výhody JavaFx stále převažující, proto se autor rozhodl pro implementaci uživatelského rozhraní v JavaFx.

3.2.3 Model uživatelského rozhraní

V dnešní době existuje již mnoho různých modelů a vzorů, které se běžně používají při implementaci funkčních uživatelských rozhraní.

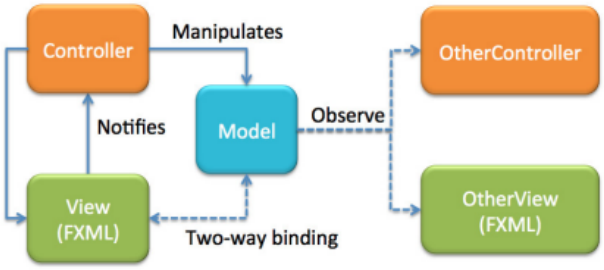
V případě JavaFx existují dle [11] několik možných modelů pro implementaci JavaFx uživatelského rozhraní:

1. **Kontrolér jako supervizor** je velmi jednoduchý model, ve kterém kontroléry obsahují veškerou logiku, interakci se servisy a aktualizaci obrazu.



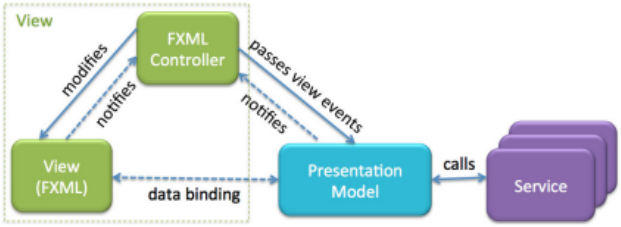
Obrázek 3.4: Model Supervising Controller [11]

2. **Společný prostředník** je model, jehož užití se vyplatí zvážit je-li aplikace komplexnější a obsahuje mnoho obrazovek. Kontroléry jsou v tomto případě stejné jako u prvního modelu, nicméně zavedením nějakého společného prostředníka se eliminuje potřeba sdílení vnitřních stavů kontrolérů mezi sebou.



Obrázek 3.5: Shared Model [11]

3. **Prezentační model** je model, který již připomíná velmi známý MVC koncept. Zbavuje kontrolérů veškeré odpovědnosti, kterou si přebírá nově přidaný model viz 3.6. V tomto případě tedy kontroléry fungují pouze jako most mezi obrazem a modelem.



Obrázek 3.6: Presentation Model [11]

Jelikož výsledná aplikace nemá velké množství obrazovek a není kladen důraz na psaní právě unit testů, rozhodl se autor v rámci návrhu použít model **Kontrolér jako supervizor**, který je pro specifikované požadavky a funkční prototyp vhodný.

3.2.4 Technologie pro průzkum cen na internetu

Jedním z požadavků na výslednou aplikaci je kalkulace ceny zboží na základě průzkumu cen na internetu. Aby toho byl výsledný prototyp schopný, musí se pro danou funkcionalitu vybrat vhodná technologie, kterou může být:

- **JSoup** je knihovna napsaná v Javě, která umožňuje pracovat s HTML soubory. JSoup poskytuje API pro načítání adres URL, extrakci a manipulaci dat z HTML [12].
- **HtmlUnit** je dle [13] „webový prohlížeč bez GUI pro Java programy“, který poskytuje API pro načítání stránek, vyplňování textových buněk nebo otevírání linků, jako každý jiný uživatel se svým „normálním“ prohlížečem.
- **Gecco**[14] je framework založený na JSoup, který poskytuje funkce pro extrakci a manipulaci dat z HTML načtených stránek. Mimo jiné Gecco podporuje také HtmlUnit rozšíření.

Jelikož všechny zmíněné technologie se v celku jednoduše používají a splnily by daný požadavek na kalkulaci ceny zboží, mohla by se v rámci této bakalářské práce použít teoreticky jakákoliv technologie. Nicméně autor se nakonec rozhodl pro HtmlUnit, jelikož umožňuje také klikat na různé elementy stránky, což by bylo velmi přínosné, kdyby byla náhodou tato funkce potřeba při kalkulaci cen na webu.

3.2.5 Databáze

Další důležitou částí aplikace je databáze, kam se ukládají veškerá potřebná data. Jelikož je aplikace dělaná pro malý podnik, který negeneruje velké množství dat, zanedbal autor možnost výběru různých typů databáze i možnost různých databází.

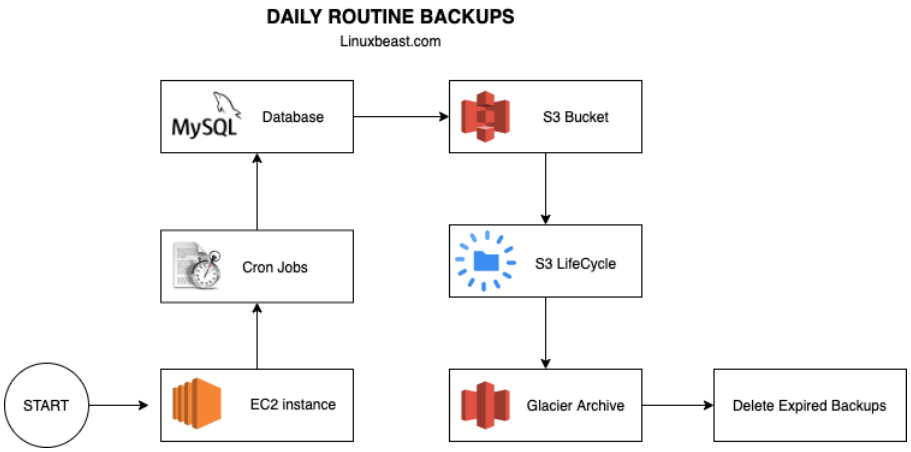
Pro databázovou část se autor rozhodl použít relační databázi MySQL od společnosti Oracle [15], se kterou má autor zkušenosti. Pro ORM se použije framework Hibernate, což také umožňuje jednoduchou výměnu databáze, kdyby byla třeba.

Pro nasazení databáze na server, je potřeba najít vhodnou platformu, která poskytuje potřebné služby pro hostování. Jednou z nejpoužívanější cloudovou platformou na světě je Amazon Web Service (AWS) [16], která má přes milion uživatelů a pro kterou se rozhodl i autor.

Konkrétně se jedná o Amazon Elastic Compute Cloud (Amazon EC2) [17], která nabízí nejširší a nejvýkonnější platformy s výběrem nejnovějších procesorů, uložišť nebo operačních systémů. Jeden z důvodu použití platformy Amazon EC2, byla možnost jeho použití v rámci AWS Free Tier na celý rok bezplatně.

3.2.6 Zálohování dat

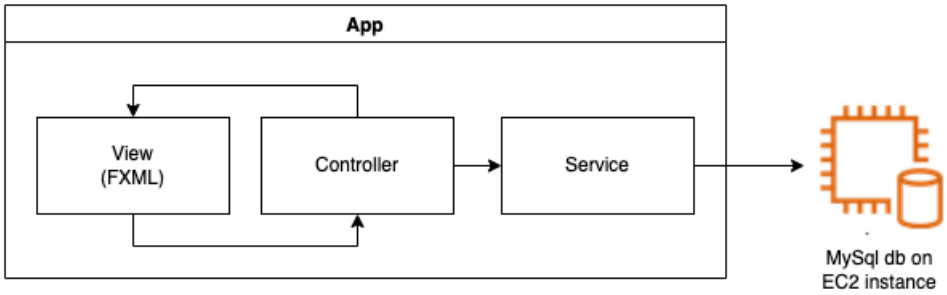
V případě, že dojde ke ztrátě dat z jakéhokoliv důvodu, přestane zastavárna fungovat. Aby se tomuto problému předešlo, musí být zavedeno pravidelné zálohování dat, které se dosáhne použitím CronJobs skriptu, Amazon S3 Bucket, S3 LifeCycle a Glacier Active přesně dle návodu na stránkách Linux Beast [18].



Obrázek 3.7: Proces zálohování dat [18]

3.2.7 Závěr

Z kapitol 3.2.1.3, 3.2.3 a 3.2.5 vyplývá výsledná architektura aplikace, podle které se bude řídit následná implementace.



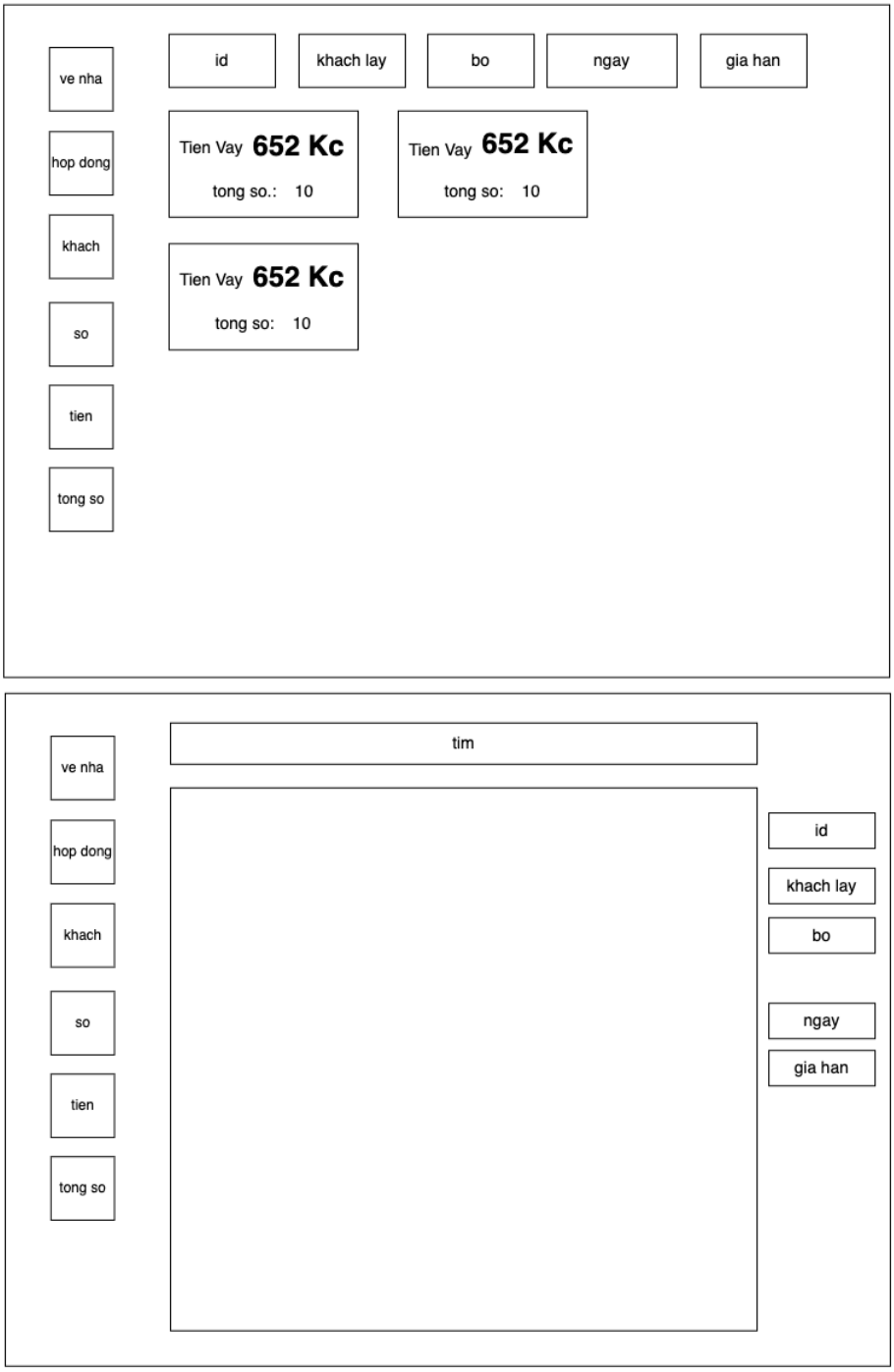
Obrázek 3.8: Výsledná architektura aplikace

3.3 Uživatelské rozhraní

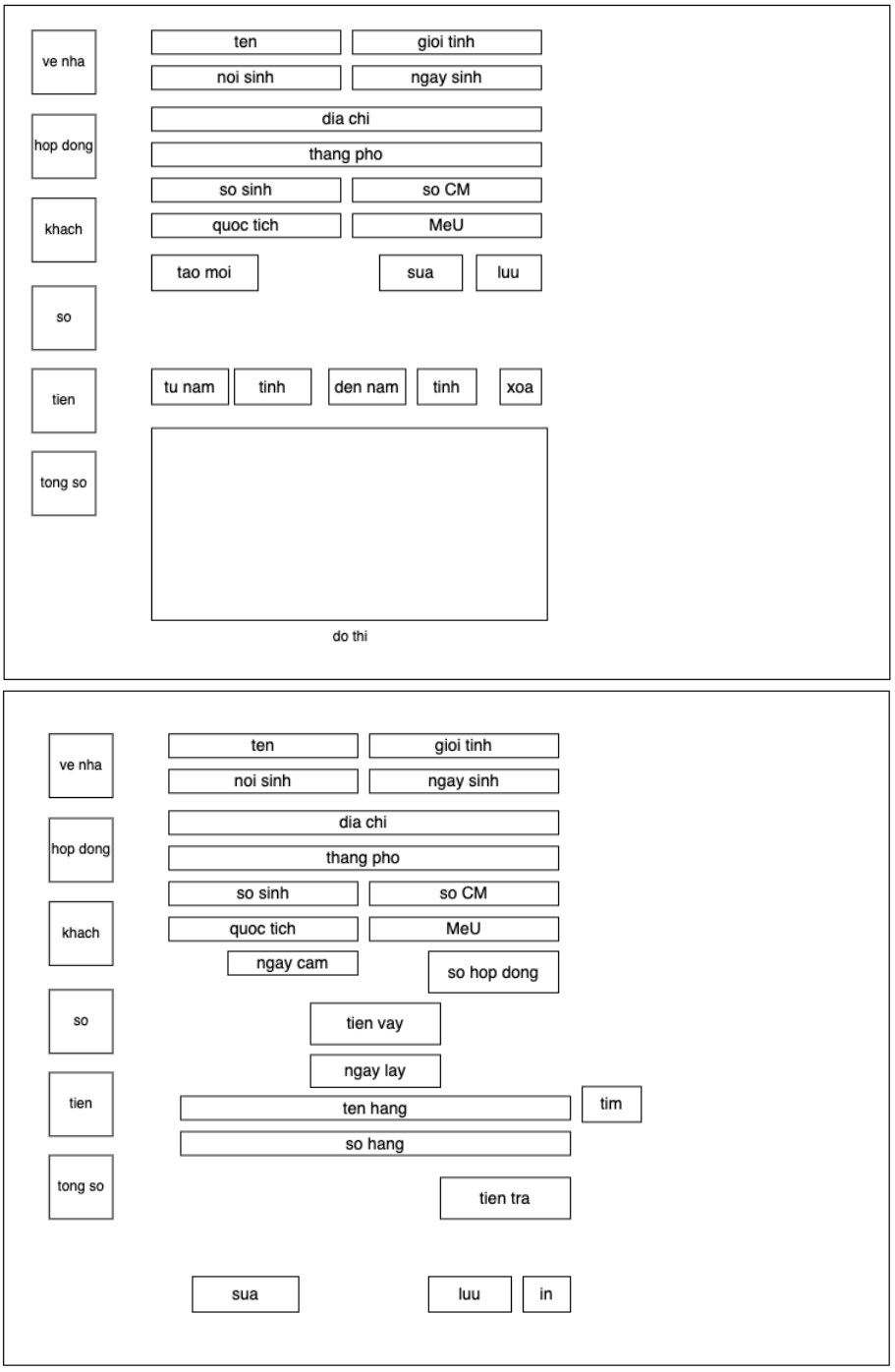
Pro kvalitnější a efektivnější konzultace s provozovateli zastavárny se při sepisování požadavků a jejich případných úprav používaly první prototypy designu uživatelského rozhraní. Tomuto prvotnímu návrhu rozpoložení elementů a funkcionalit aplikace se říká wireframe [19]. Mezi nejlepší wireframe nástroje pro rok 2022 dle [20] patří:

- **UXPin**
- **Adobe XD**
- **Axure RP**
- **Balsamiq Mockups**
- **Wireframe.cc**

Navzdory tomu, že uvedené nástroje vypadají dobře a poskytují i bezplatnou zkušební dobu, rozhodl se autor pro urychlení procesu provést návrhy obrazovek v diagrams.net [21], která je velmi intuitivní a jednoduchá k používání. Během konzultací s budoucími uživateli, si uživatelé rovnou sami posouvali, přidávali nebo mazali různé elementy obrazovky dle svých preferencí.



Obrázek 3.9: Návrh uživatelského rozhraní 1



Obrázek 3.10: Návrh uživatelského rozhraní 2

Kapitola 4

Implementace

Po návrhu přichází na řadu realizace. V této kapitole se stručně popisují veškeré technologie, nástroje a postupy použité v každé fázi implementace. Jedná se o implementaci samotnou, testování, nasazení a spuštění aplikace.

4.1 Nástroje a technologie

Během implementace se použilo mnoho nástrojů a technologií. Výběr některých z nich byl již odůvodněn v předešlých kapitolách. Pro ty, které zmíněné ještě nebyly, se autor rozhodoval podle svých zkušeností a znalostí daného nástroje či technologie.

4.1.1 GitHub

GitHub [22] je platforma, která podporuje vývoj softwaru pomocí verzovacího nástroje Git. Vývoj softwaru probíhal tak, že se pro novou funkci či sadu společných funkcí vždy vytvořila nová pracovní větev na GitHubu. Po dokončení implementace a její kontroly se následně vytvořil na GitHubu pull request pro merge s větví master. Veškeré zdrojové kódy a použité obrázky pro ikony aplikace jsou uloženy právě na GitHubu.

4.1.2 Nástroje pro podporu vývoje

Kromě platformy GitHub byly použity i další nástroje pro podporu realizace aplikace.

Prvním pomocníkem je IntelliJ IDEA od společnosti JetBrains [23]. Jedná se o vývojové prostředí pro vývoj softwaru například v programovacím jazyce Java. Během implementace se prostředí IntelliJ využilo zejména pro pohodlné psaní a editaci zdrojových kódů díky chytrým funkcím, build a spuštění aplikace jedním kliknutím pomocí nastavitelné konfigurace.

Dalším použitým nástrojem pro zjednodušení a urychlení vývoje byl Scene Builder od společnosti Gluon [24]. Scene Builder je vizuální nástroj pro podporu vytváření JavaFx grafických uživatelských rozhraní definované FXML soubory. Tento nástroj umožňuje drag&drop funkci pro velmi rychlé přidání, umístění a škálování různých tlačítek, textových polí či grafů. Tuto vizuální funkci poskytuje i prostředí IntelliJ, nicméně práce s Scene Builder od společnosti Gluon je mnohem responzivnější a rychlejší. Kromě toho se také Scene Builder dá jednoduše integrovat do vývojového prostředí IntelliJ IDEA.

4.1.3 JavaFx

Z kapitoly 3.2.2 se pro implementaci vybral framework JavaFx. Konkrétně se využily moduly javafx-controls a javafx-fxml.

První ze zmíněných modulů poskytuje vývojářům širokou nabídku textových polí, tlačítek, seznamů, tabulek, filtrů nebo různých typů grafů.

Modul javafx-fxml umožňuje vytvářet grafická uživatelská rozhraní pomocí souborů FXML, což je značkovací jazyk založen na XML pro definování uživatelských rozhraní JavaFx aplikací. Pro vytvoření vztahu mezi FXML souborem a kontrolérem stačí jednoduše přidat jméno třídy kontroléru přímo do odpovídajícího FXML souboru. Tímto se automaticky proměnné JavaFx komponent v kontroléru provážou s odpovídajícím JavaFx komponentem v FXML souboru, což umožní kontroléru manipulovat s obrazem a reagovat na vstupy od uživatelů.

```
1 <Pane id="contractsButton" onMouseClicked="#switchToContracts"
      prefHeight="35.0" prefWidth="156.0" styleClass="buttonPane">
```

Zdrojový kód 4.1: Tlačítko definované v FXML souboru

```
1 @FXML
2 private void switchToContracts(final MouseEvent event) throws
      IOException {
3   mainPane.setCenter(getPage("contracts.fxml"));
4 }
```

Zdrojový kód 4.2: Funkce reagující na stisknutí tlačítka

4.1.4 HtmlUnit

Jak již bylo nastíněno v kapitole 3.2.4, pro implementaci funkce kalkulace ceny zboží na základě průzkumu cen na internetu byl vybrán nástroj HtmlUnit. Tento nástroj umožňuje vývojářům přesně extrahovat vybraná data z HTML stránek pomocí XPath.

Pro jednoduchost a diversifikaci různých cen stejného zboží na různých stránkách a internetových obchodech využil autor reklam v internetovém vyhledávači od společnosti Google [25].

Při zadání jakéhokoliv produktu, který je vystaven a prodáván na internetu, do vyhledávače Google se ve výsledku vyhledávání objeví reklamy na zadaný produkt od různých prodejců. Tohoto faktu se využije tak, že se nejprve vyhledá dotazovaný produkt ve vyhledávači Google pomocí jednoduchého URL *https://www.google.com/search?q="jméno_produktu"*. Po načtení stránky se pak pomocí XPath extrahují potřebné ceny daného zboží z Google reklam.

Následující část kódu 4.3 přesně extrahuje hledané ceny od různých prodejců.

```
List<HtmlDivision> anchors = page.getByXPath("//div[contains(
    @class, 'rwVHAc')]");

for (HtmlElement anchor : anchors) {
  HtmlAnchor name = anchor.getFirstByXPath("./div[@role='heading
    ']/a[contains(@class, 'plantl pla-unit-title-link')]");
  HtmlElement priceElem = anchor.getFirstByXPath("./div[contains(
    @class, 'T4OwTb')]");

  prices.add(new Price(name.getTextContent(), name.
    getHrefAttribute(), formatPriceString(priceElem.
    getTextContent())));
}
```

Zdrojový kód 4.3: Extrahování cen z Google reklam

4.1.5 Hibernate

Hibernate [26] je framework napsaný v Javě, který se používá pro Objektově Relační Mapování neboli ORM. Tento framework automaticky mapuje objekty na entity v relační databázi za použití jednoduchých anotací, což ulehčuje vývojářům značnou práci při vývoji.

Samotné připojení a manipulace s databází je řešena pomocí Hibernate objektů *SessionFactory* a *Session*. *SessionFactory* je objekt pro vytváření *Session* objektů na základě konfiguračního souboru *hibernate.cfg.xml*. *Session* je objekt poskytující připojení mezi aplikací a databází.

```
public abstract class Repository<K, E> {
  protected Session session;
  protected SessionFactory FACTORY = new Configuration().
    configure("/hibernate.cfg.xml").buildSessionFactory();

  public void save(final E entity) {
    try {
      session = FACTORY.openSession();
      session.beginTransaction();
      session.save(entity);
      session.getTransaction().commit();
    } catch(final Exception e) {
      if(session != null && session.getTransaction() != null) {
        session.getTransaction().rollback();
      }
```

```
15        e.printStackTrace();
16      } finally {
17        if(session != null) {
18          session.close();
19        }
20      }
21    }
22  }
```

Zdrojový kód 4.4: Příklad užití frameworku Hibernate pro uložení entity

4.2 Nasazení a spuštění aplikace

Jak již bylo zmíněno v kapitole 3.2.5, MySQL server s databází je nasazen na platformě Amazon EC2 s instancí typu t2.micro a konfigurací `1 CPU`, `1 GiB RAM` a `25 GiB SSD`.

Pro nasazení aplikace do podniku a spuštění byly použity následující technologie a postupy.

4.2.1 Apache Maven

Apache Maven je nástroj pro správu softwarových projektů, který na základě POM souboru řídí jejich sestavení, které zahrnuje mimo jiné také stažení závislostí nebo spuštění testů [27].

Apache Maven se použije v první fázi nasazení, ve které se vytvoří JAR soubor, který je nutný pro další fáze nasazení aplikace. Toho se dosáhne použitím pluginu maven-shade-plugin, který umožní vytvoření JAR souboru z nemodulárního projektu, který používá modulární moduly.

```
1  <plugin>
2    <artifactId>maven-shade-plugin</artifactId>
3    <version>3.2.1</version>
4    <executions>
5      <execution>
6        <phase>package</phase>
7        <goals>
8          <goal>shade</goal>
9        </goals>
10       <configuration>
11         <filters>
12           <filter>
13             <artifact>*:*</artifact>
14             <excludes>
15               <exclude>module-info.class</exclude>
16             </excludes>
17           </filter>
18         </filters>
19         <transformers>
20           <transformer implementation="org.apache.maven.plugins.
     shade.resource.ManifestResourceTransformer">
```

```
21              <mainClass>
22                cz.fit.cvut.contract_manager.Main
23              </mainClass>
24            </transformer>
25          </transformers>
26        </configuration>
27      </execution>
28    </executions>
29  </plugin>
```

Zdrojový kód 4.5: Použití shade pluginu v pom.xml

4.2.2 JPackage

JPackage je dle [28] nástroj, který umožňuje generovat instalační balíčky pro modulární i nemodulární Java aplikace. Vygenerovaný balíček specifický pro určitou platformu Linux, macOS nebo Windows poskytuje uživatelům jednoduchý způsob pro instalaci a spuštění aplikací.

Protože JPackage generuje balíčky specifické pro určitou platformu, musí se aplikační balíčky vytvářet na platformě, pro kterou chceme balíček vygenerovat.

Po vygenerování instalačního balíčku pro požadovanou platformu stačí již přenést instalační balíček například pomocí USB flash disku na jakýkoliv počítač a spustit instalaci.

Pro vytvoření instalačního balíčku pro windows ve formátu MSI stačí zadat následující příkaz 4.6 do příkazové řádky.

```
1 jpackage -i ./target/ --main-class cz.fit.cvut.contract_manager.
    Main --main-jar contract_manager-1.0-SNAPSHOT.jar --type "msi
    " --name "ContractManager" --win-dir-chooser --win-shortcut
    --icon src/main/resources/pics/cm_icon.ico
```

Zdrojový kód 4.6: JPackage vytvoření instalačního balíčku pro Windows

Pro vytvoření instalačního balíčku pro MacOS ve formátu DMG stačí zadat následující příkaz 4.7 do příkazové řáadky.

```
1 jpackage -i ./target/ --main-class cz.fit.cvut.contract_manager.
    Main --main-jar contract_manager-1.0-SNAPSHOT.jar --type "dmg
    " --name "ContractManager"
```

Zdrojový kód 4.7: JPackage vytvoření instalačního balíčku pro MacOS

4.2.2.1 WiX

Ke generování instalačních balíčků pomocí JPackage na operačním systému Windows je ještě za potřebí WiX toolset, což je dle [29] soubor nástrojů, které vytvářejí instalační balíčky pro Windows z XML.

4.3 Testování

Při buildu aplikace se vždy automaticky spustí všechny testy díky pluginu maven-surefire-plugin. Aplikace je testována různými testy, které využívají různé technologie a testují různé funkcionality.

```
[INFO] Results:
[INFO]
[INFO] Tests run: 79, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  01:35 min
[INFO] Finished at: 2022-05-01T14:24:55+02:00
[INFO] ------------------------------------------------------------------------
```

Obrázek 4.1: Výstup z terminálu po zadání příkazu **mvn test**

4.3.1 JUnit 5

JUnit 5 je dle [30] testovací framework pro jazyk Java. Použitím tohoto frameworku se implementují různé testy, které jsou specifické tím, že testují správné chování části systému izolovaně od ostatních částí. Těmto testům se říká unit testy.

4.3.2 Mockito

Testování tříd unit testy není ale vždy možné, jelikož ve většině případů obsahují třídy proměnné z dalších tříd. Aby se i tyto třídy, které jsou závislé na ostatních třídách, daly testovat jako samostatná jednotka, použije se framework Mockito.

Mockito je dle [31] framework, který se používá pro mockování proměnných, aby se mohla následně vybraná třída otestovat, aniž by byla závislá na implementaci ostatních tříd. Mockito poskytuje vývojářům sadu anotací a metod, které dokáží plně simulovat chování tříd, na kterých je testovaná třída závislá. Tomuto procesu simulace chování třídy se říká mockování.

```
1  @ExtendWith(MockitoExtension.class)
2  class ContractRepositoryServiceTest {
3
4    @Mock
5    private ContractRepository contractRepository;
6
7    @InjectMocks
8    private ContractRepositoryService contractRepositoryService;
9
10   @Test
11   void shouldGetMostRecentByContractId() {
12     String contractId = "R12";
13     Contract expectedContract =
14       new Contract(contractId, new Date(10), 1000, new Date(20),
```

```
15                "Mobile", "j123", 1000);
16
17      when(
18        contractRepository.getMostRecentByContractId(contractId)
19      ).thenReturn(expectedContract);
20
21      Contract actualContract = contractRepositoryService
22        .getMostRecentByContractId("R12");
23
24      verify(contractRepository, times(1))
25        .getMostRecentByContractId(contractId);
26
27      assertEquals(actualContract, expectedContract);
28    }
29  }
```

Zdrojový kód 4.8: Unit testování s frameworkem Mockito

4.3.3 TestFx

Dalším typem testů jsou testy integrační, které testují funkčnost aplikace jako celek. Integrační testy byly implementovány pro všechny kontroléry pro otestování reakce kontrolérů na eventy simulující uživatelské vstupy, interakce kontrolérů se servisy a manipulací dat z databáze.

Protože jsou kontroléry a eventy simulující uživatelské vstupy závislé na JavaFx komponentách, využijeme testovacího frameworku TestFx, který je vhodný pro testování JavaFx aplikací. TestFx je dle [32] testovací framework pro JavaFx aplikace, který umožňuje inicializaci JavaFx komponent a poskytuje vývojářům sadu metod pro simulaci uživatelských vstupů.

```
1  @ExtendWith(ApplicationExtension.class)
2  class ContractControllerTest {
3
4    @Start
5    public void start(final Stage stage) throws Exception {
6      Parent root = FXMLLoader.load(
7        getClass().getResource("/fxml/contracts.fxml")
8      );
9
10     stage.setScene(new Scene(root));
11     stage.show();
12   }
13
14   @Test
15   void switchToCreateContractTest(final FxRobot robot) {
16     BorderPane mainPane =
17       robot.lookup("#mainPane").queryAs(BorderPane.class);
18
19     assertEquals("contractsPane", mainPane.getCenter().getId());
20
21     robot.clickOn("#createButton");
22
```

```
23      assertEquals(
24        "createContractAnchorPane",
25        mainPane.getCenter().getId()
26      );
27    }
28  }
```

Zdrojový kód 4.9: Testování s frameworkem TestFx

4.3.4 HSQLDB

Hsqldb je dle [33] SQL relační databázový systém napsaný v Javě a nabízí rychlý transakční databázový engine. Pomocí Hsqldb se dají vytvořit tabulky buďto jako in-memory nebo přímo na disku jako integrovaný databázový systém.

Tyto vlastnosti jsou vhodné právě pro testování aplikace jako celek, jelikož při testování je nepřípustné, aby se manipulovalo s databází produkční. Použití této databáze při testování je díky Hibernate snadné. Stačí správně nakonfigurovat soubor *hibernate.cfg.xml*. Databáze pro testování je nakonfigurovaná jako in-memory, což znamená, že databáze používá hlavní paměť RAM pro ukládání dat místo disku. Po dokončení testů se obsah databáze vždy automaticky zahodí.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate
       Configuration DTD 3.0//EN" "http://www.hibernate.org/dtd/
       hibernate-configuration-3.0.dtd">
3  <hibernate-configuration>
4    <session-factory>
5      <property name="hbm2ddl.auto">create-drop</property>
6
7      <property name="hibernate.dialect">
8        org.hibernate.dialect.HSQLDialect
9      </property>
10
11     <property name="connection.url">
12       jdbc:hsqldb:mem:db_test
13     </property>
14
15     <property name="connection.username">sa</property>
16     <property name="connection.password">sa</property>
17
18     <property name="connection.driver_class">
19       org.hsqldb.jdbcDriver
20     </property>
21
22     <mapping class="cz.fit.cvut.contract_manager.entity.Contract"
       />
23     <mapping class="cz.fit.cvut.contract_manager.entity.Customer"
       />
```

```
24      <mapping class="cz.fit.cvut.contract_manager.entity.History"
        />
25    </session-factory>
26  </hibernate-configuration>
```

Zdrojový kód 4.10: Konfigurace souboru hibernate.cfg.xml pro testování

4.3.5 Uživatelské testování

Jednou z velmi důležitou částí testování výsledné aplikace je uživatelské testování. Tohoto testování se účastnili budoucí uživatelé aplikace, kteří aplikaci zkoušeli používat, jako kdyby byl systém již nasazen do podniku.

Během uživatelského testování se narazilo na několik nedostatků a chyb, které se nedokázaly odchytit předešlými testy. Z testování s budoucími uživateli vzešla následující tabulka 4.1, která obsahuje popis chyb či nedostatků aplikace a jejich řešení.

Tabulka 4.1: Tabulka chyb z uživatelského testování

Popis problému	**Řešení**
Při tisknutí smlouvy se údaje o zákazníkovi a zboží vytiskly na papír ve špatném zarovnání.	Oprava daného FXML souboru
Při tisknutí smlouvy se údaje o zákazníkovi vytiskly v jiném odstínu než údaje o zboží.	Oprava proměnné *opacity* v daném FXML souboru
Při tisknutí smlouvy chybí čárka s pomlčkou za cenami.	Přidání čárky a pomlčky za cenou do funkce pro tisk.
Aplikace neakceptuje zápis datu ve formátu dd.mm.yy.	Úprava parsovací funkce pro akceptování formátů dd.mm.yyyy i dd.mm.yy
Aplikace značí smlouvy, které jsou ještě platné, jako propadlé.	Prohození znaménka při porovnání datu splatnosti.
Graf o stavech smluv neukazoval propadlé smlouvy.	Opraveno přidáním této informace do grafu.
Procenta v grafu o stavech smluv nebyla přesná.	Použití datového typu Double místo datového typu Integer.

4.4 Závěr

Výstupem implementace je funkční prototyp, jehož hlavní funkce byly otestovány jak automatickými testy tak testy uživatelskými. Všechny funkční požadavky nejvyšších priorit MUST HAVE a SHOULD HAVE byly splněny.

Jediné požadavky priority COULD HAVE, které nebyly splněny a budou doimplementovány v dalších iteracích, jsou:

- FP-7.1 - notifikace na propadlou smlouvu [COULD HAVE]
- FP-7.2 - zobrazení notifikace [COULD HAVE]
- FP-8.2 - tisk seznamu smluv [COULD HAVE]

Kapitola 5

Budoucí rozšíření

Protože je výsledná aplikace závislá na internetovém připojení, je její použití při výpadku internetu nemožné. V této situaci by budoucí uživatelé určitě ocenili určitá rozšíření aplikace, která by daný problem řešila.

Jak již bylo nastíněno, tato kapitola popisuje možná rozšíření aplikace, která by řešila problém výpadku internetového připojení. Kromě samotného popisu rozšíření se autor také pokusí stručně nastínit možný návrh a implementaci.

5.1 Offline mód

Prvním možným rozšířením aplikace je umožnění použití aplikace v takzvaném offline módu, ve kterém uživatelé mohou provádět veškeré operace a činnosti jako v normálním online režimu. Jedinou výjimkou je kalkulace ceny zboží z internetu, která v offline režimu vyřešit nepůjde.

5.1.1 Nahrazení „internetových“ částí systému

Pro dosažení funkčnosti aplikace i bez internetového připojení se musí nahradit všechny části systému, které se nenacházejí na koncovém zařízení uživatele. Protože je výsledná aplikace implementovaná jako 2-tier aplikace, která má uživatelské rozhraní a veškerou logiku na straně klienta, je databáze jedinou částí systému, kterou je potřeba nahradit.

Pro nahrazení se využije již zmíněná databáze HSQLDB, která umožňuje lokální ukládání dat na disk počítače. Pro správné chování databáze se použije následující konfigurace podle 5.1. Použitím této konfigurace bude databáze ukládat veškerá data do složky pod názvem *data*, která se nachází ve složce, kam se instalovala aplikace.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate
    Configuration DTD 3.0//EN" "http://www.hibernate.org/dtd/
    hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <property name="hbm2ddl.auto">update</property>

    <property name="hibernate.dialect">
      org.hibernate.dialect.HSQLDialect
    </property>

    <property name="connection.url">
      jdbc:hsqldb:./data/db:contract_manager_db
    </property>

    <property name="connection.username">sa</property>
    <property name="connection.password">sa</property>

    <property name="connection.driver_class">
      org.hsqldb.jdbcDriver
    </property>

    <mapping class="cz.fit.cvut.contract_manager.entity.Contract"
     />
    <mapping class="cz.fit.cvut.contract_manager.entity.Customer"
     />
    <mapping class="cz.fit.cvut.contract_manager.entity.History"
    />
  </session-factory>
</hibernate-configuration>
```

Zdrojový kód 5.1: Konfigurace souboru hibernate_fallback_solution.cfg.xml

5.1.2 Integrace

Po připravení náhradní databáze je ještě třeba toto fallback řešení integrovat do stávající aplikace. Jelikož nám framework Hibernate už řeší ORM, zbývá ještě vyřešit přepínání mezi vzdálenou databází a databází lokální.

Pro přepínání mezi dvěmi různými *SessionFactory* objekty, se implementuje nová metoda *buildSessionFactory()* viz 5.2.

Kdykoliv je potřeba manipulovat s databází, zavolá se metoda *buildSessionFactory()*, která vrátí *SesstionFactory* s konfigurací buď pro vzdálenou databázi nebo pro databázi lokální.

Bez internetového připojení trvá proces vyhození výjimky při připojení do vzdálené databáze téměř jednu minutu. Aby přepínání databází netrvalo vždy takhle dlouho, obalí se proces připojení do vzdálené databáze objektem *Future* pomocí *ExecutorService*.

```
static Session session;
static SessionFactory sessionFactory;

protected SessionFactory buildSessionFactory() {
  ExecutorService executor = Executors.newSingleThreadExecutor();
  Future<SessionFactory> future = executor.submit(
    () -> new Configuration()
      .configure("/hibernate.cfg.xml")
      .buildSessionFactory()
  );

  try {
    sessionFactory = future.get(1, TimeUnit.SECONDS);
  } catch (Exception e) {
    future.cancel(true);
    sessionFactory = new Configuration()
        .configure("/hibernate-fallback.cfg.xml")
        .buildSessionFactory();
  }

  executor.shutdownNow();
  return sessionFactory;
}
```

Zdrojový kód 5.2: Příklad řešení přepínání mezi lokální a vzdálenou databází

5.2 Synchronizace dat

Rozšíření aplikace o možnost jejího používání i bez internetového připojení určitě přináší své benefity, nicméně tato funkce rozděluje data o různých smlouvách a zákaznících do dvou různých databází. Tato situace může být pro uživatele nepříjemná, jelikož jedna z hlavních výhod aplikace je vytváření smluv z různých uložených šablon. V případě dvou různých databází se některé šablony nachází na databázi lokální a některé zase na databázi vzdálené.

Pro vyřešení tohoto problému se vyplatí zavést synchronizaci dat mezi lokální a vzdálenou databází.

5.2.1 Návrh řešení

Jelikož se lokální databáze používá v případě výpadku internetového připojení, dává smysl pro synchronizaci dat překopírovat veškerá data z lokální databáze do databáze na serveru, jakmile to bude možné. Detekce pro spuštění synchronizace je jednoduchá, stačí nahradit blok kódu v metodě *buildSessionFactory()* z 5.2 následujícím kódem 5.3.

```
try {
  SessionFactory newSessionFactory =
    future.get(1, TimeUnit.SECONDS);

```

```
  if(sessionFactory.getProperties().get("connection.url") ==
    "jdbc:hsqldb:./data/db:contract_manager_db") {
    synchronize(newSessionFactory);
  }

  sessionFactory = newSessionFactory;
} catch (Exception e) {
  future.cancel(true);
  sessionFactory = new Configuration()
        .configure("/hibernate-fallback.cfg.xml")
        .buildSessionFactory();
}
```

Zdrojový kód 5.3: Příklad podmínky pro zavolání funkce pro překopírování

Řešení metody *synchronize()* je již složitější, jelikož se musí brát v úvahu následující faktory:

1. Načtení všech entit z lokální databáze a následné uložení do databáze vzdálené nemusí být možné. Protože každá entita musí mít unikátní identifikátor, může nastat kolize identifikátorů při kopírování entit z databáze lokální na databázi vzdálenou.

2. Pro vyřešení 1. bodu se musí nějakým způsobem měnit identifikátory entit. Tím se ale mohou porušit některé vazby mezi entitami.

3. I kdyby nenastala kolize identifikátorů, překopírování všech zákazníků z lokální na vzdálenou databázi může způsobit duplikaci zákazníků v databázi na serveru.

Pro vyřešení prvního možného problému stačí načteným entitám z lokální databáze přepsat jejich identifikátor na *null*. Tím se dosáhne automatického přidělení správného identifikátoru při ukládání na vzdálenou databázi.

Jelikož se identifikátory všech entit přepíšou, musí se správně také přepsat vazby mezi dvojicemi entit *Customer-Contract* a *Contract-History*.

Možné duplikáty se zamezí tím, že se nejprve provede kontrola, jestli existuje daný zákazník ve vzdálené databázi. K tomu se využije rodné číslo zákazníka, jelikož se jedná o unikátní údaj.

Nyní zkombinováním zmíněných řešení se může funkce *synchronize()* implementovat například jako následující pseudokód 5.4.

```
void synchronize() {
  for customer in customersFromLocal {
    contracts = customer.getContracts();

    if customer not in customersFromRemote {
      customer.setId(null);
      saveToRemote(customer);
    }

```

```
10      customer = getCustomerFromRemote(
11        customer.getPersonalNumber()
12      );
13
14      for contract in contracts {
15        histories = contract.getHistories();
16
17        contract.setId(null);
18        contract.setCustomer(customer);
19        saveToRemote(contract);
20
21        contract = getMostRecentContractFromRemote(
22          contract.getContractId()
23        );
24
25        for history in histories {
26          history.setId(null);
27          history.setContract(contract);
28          saveToRemote(history);
29        }
30      }
31    }
32  }
```

Zdrojový kód 5.4: Příklad funkce pro překopírování dat

Závěr

Hlavním cílem bakalářské práce bylo poskytnutí řešení reálného problému správy smluv formou funkční aplikace. Dílčím cílem bylo také sepsání všech funkčních a nefunkčních požadavků na systém. Protože výstupem této bakalářské práce byl funkční prototyp, který byl řádně navržen, implementován, otestován a nasazen do podniku, považuji hlavní cíl práce jako splněný. Dílčí cíl považuji také za splněný, jelikož návrh aplikace vzešel právě z analýzy, během níž se sepsaly všechny funkční a nefunkční požadavky na systém.

Momentální verze aplikace určitě není verzí poslední. V dalších iteracích se doimplementují nedodělané funkční požadavky z analytické části a realizují zmíněná možná rozšíření aplikace.

Při psaní této bakalářské práce jsem se obohatil o cenné zkušenosti z oblasti analýzy, návrhu a implementace desktopové aplikace. Rozšířil jsem si obzory nejenom v různých používaných nástrojů a technologií, ale také v samotném návrhu a implementace aplikace obecně.

Literatura

[1] Contract Zen: Governance software: Contractzen Contract and Board Portal [online]. 2021, [cit. 2022-04-15]. Dostupné z: https://www.contractzen.com/

[2] PandaDoc Inc.: PandaDoc [online]. 2022, [cit. 2022-04-15]. Dostupné z: https://www.pandadoc.com/

[3] Concord: Concord Now [online]. 2022, [cit. 2022-04-15]. Dostupné z: https://www.concordnow.com/

[4] Product Plan: MoSCoW Prioritization [online]. *Product Plan*, 2021, [cit. 2022-04-15]. Dostupné z: https://www.productplan.com/glossary/moscow-prioritization/

[5] Oracle Corporaction: Java™ Platform, Standard Edition 7 API Specification - Package java.awt [online]. 2020, [cit. 2022-04-20]. Dostupné z: https://docs.oracle.com/javase/7/docs/api/java/awt/package-summary.html

[6] Oracle Corporaction: Java SE Documentation - Abstract Window Toolkit (AWT) [online]. 2005, [cit. 2022-04-20]. Dostupné z: https://docs.oracle.com/javase/7/docs/technotes/guides/awt/index.html

[7] Idris, N.: SWINGX TUTORIAL - PAINTERS [online]. *developerlife.com*, 2007, [cit. 2022-04-20]. Dostupné z: https://developerlife.com/2007/12/24/swingx-tutorial-painters/

[8] org.swinglabs.swingx [online]. 2013, [cit. 2022-04-20]. Dostupné z: https://maven.java.net/index.html

[9] JGoodies: JGoodies [online]. 2021, [cit. 2022-04-20]. Dostupné z: https://www.jgoodies.com/home/about/

[10] openjfx.io: JavaFX [online]. 2022, [cit. 2022-04-20]. Dostupné z: https://openjfx.io/index.html

[11] Kruk, G.; Alves, O. D. S.; Molinari, L.; aj.: Best Practices for Efficient Development of JavaFX Applications. In *Proc. of International Conference on Accelerator and Large Experimental Control Systems (ICALEPCS'17), Barcelona, Spain, 8-13 October 2017*, číslo 16 in International Conference on Accelerator and Large Experimental Control Systems, Geneva, Switzerland: JACoW, Jan. 2018, ISBN 978-3-95450-193-9, s. 1078–1083, doi:https://doi.org/10.18429/JACoW-ICALEPCS2017-THAPL02, https://doi.org/10.18429/JACoW-ICALEPCS2017-THAPL02. Dostupné z: http://jacow.org/icalepcs2017/papers/thapl02.pdf

[12] jsoup.org: JSoup: Java HTML Parser [online]. © 2009 - 2021, [cit. 2022-04-20]. Dostupné z: https://jsoup.org/

[13] Gargoyle Software Inc.: HtmlUnit [online]. © 2002–2022, [cit. 2022-04-20]. Dostupné z: https://htmlunit.sourceforge.io/

[14] Gecco [online]. 2022, [cit. 2022-04-20]. Dostupné z: https://github.com/xtuhcy/gecco

[15] Oracle Corporation: MySQL [online]. © 2022, [cit. 2022-04-20]. Dostupné z: https://www.mysql.com/

[16] Amazon Web Services, Inc.: Amazon AWS [online]. © 2022, [cit. 2022-04-20]. Dostupné z: https://aws.amazon.com/what-is-aws/

[17] Amazon Web Services, Inc.: Amazon EC2 [online]. © 2022, [cit. 2022-04-20]. Dostupné z: https://aws.amazon.com/ec2/

[18] Gerald: Automate MySQLdump Backup in EC2 to Amazon S3 [online]. *Linuxbeast*, 2020, [cit. 2022-04-20]. Dostupné z: https://linuxbeast.com/tutorials/aws/automate-mysqldump-backup-on-ec2-to-amazon-s3/

[19] Experience UX: What is wireframing [online]. *Experience UX*, © 2022, [cit. 2022-04-21]. Dostupné z: https://www.experienceux.co.uk/faqs/what-is-wireframing/

[20] May, T.: The best wireframe tools in 2022 [online]. *Creative Bloq*, 2021, [cit. 2022-04-21]. Dostupné z: https://www.creativebloq.com/wireframes/top-wireframing-tools-11121302

[21] diagrams.net: Diagrams.net [online]. © 2005-2021, [cit. 2022-04-21]. Dostupné z: https://www.diagrams.net/

[22] GitHub, Inc.: GitHub [online]. © 2022, [cit. 2022-04-21]. Dostupné z: https://github.com/

[23] JetBrains s.r.o.: IntelliJ IDEA [online]. © 2000-2022, [cit. 2022-04-21]. Dostupné z: https://www.jetbrains.com/idea/

[24] Gluon: Scene Builder [online]. © 2022, [cit. 2022-04-21]. Dostupné z: https://gluonhq.com/products/scene-builder/

[25] Google LLC: Google Search [online]. 2022, [cit. 2022-04-21]. Dostupné z: https://www.google.com/search/about/

[26] Red Hat: Hibernate [online]. 2022, [cit. 2022-04-21]. Dostupné z: https://hibernate.org/

[27] Apache Software Foundation: Apache Maven Project [online]. © 2002–2022, [cit. 2022-04-21]. Dostupné z: https://maven.apache.org/

[28] Oracle Corporation: Packaging Tool User's Guide [online]. © 2020, [cit. 2022-04-21]. Dostupné z: https://docs.oracle.com/en/java/javase/14/jpackage/packaging-overview.html

[29] wixtoolset.org: Wix Toolset [online]. 2021, [cit. 2022-04-21]. Dostupné z: https://wixtoolset.org/

[30] The JUnit Team: JUnit 5 User Guide [online]. 2021, [cit. 2022-04-21]. Dostupné z: https://junit.org/junit5/docs/current/user-guide

[31] mockito.org: Mokcito [online]. 2022, [cit. 2022-04-22]. Dostupné z: https://site.mockito.org/

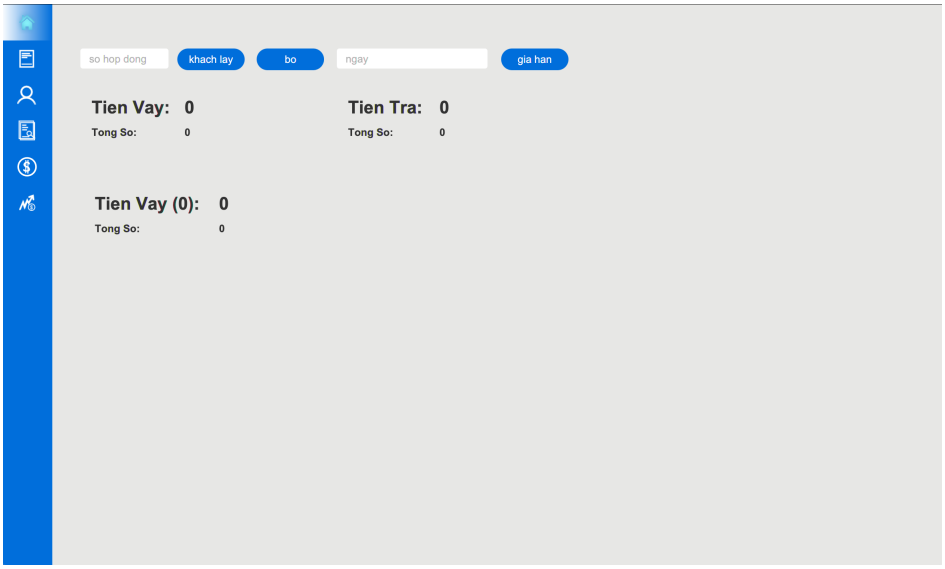
[32] TestFx: TestFx [online]. 2021, [cit. 2022-04-22]. Dostupné z: https://github.com/TestFX/TestFX/'

[33] The HSQL Development Group.: HyberSQL [online]. ©2001-2021, [cit. 2022-04-22]. Dostupné z: https://hsqldb.org/

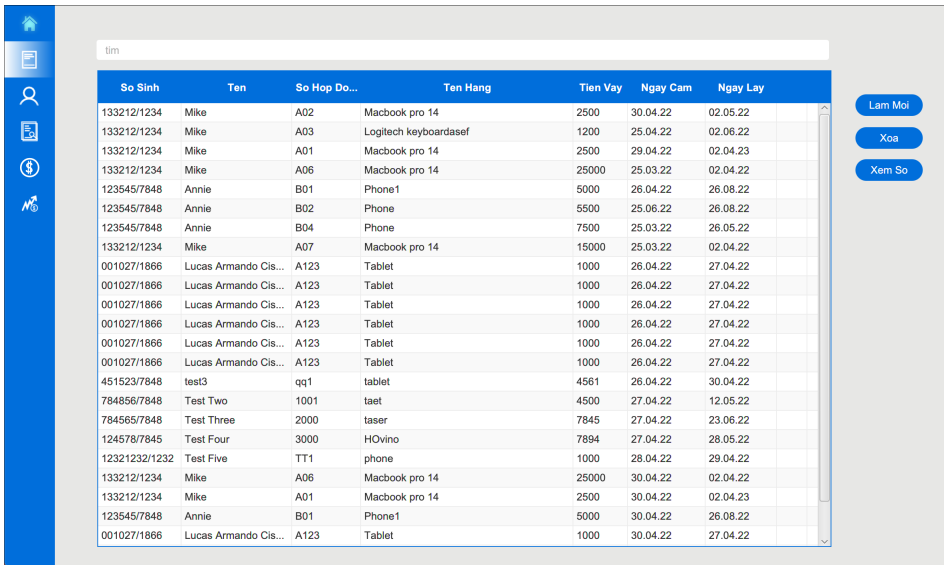
Příloha A

Snímky výsledné aplikace

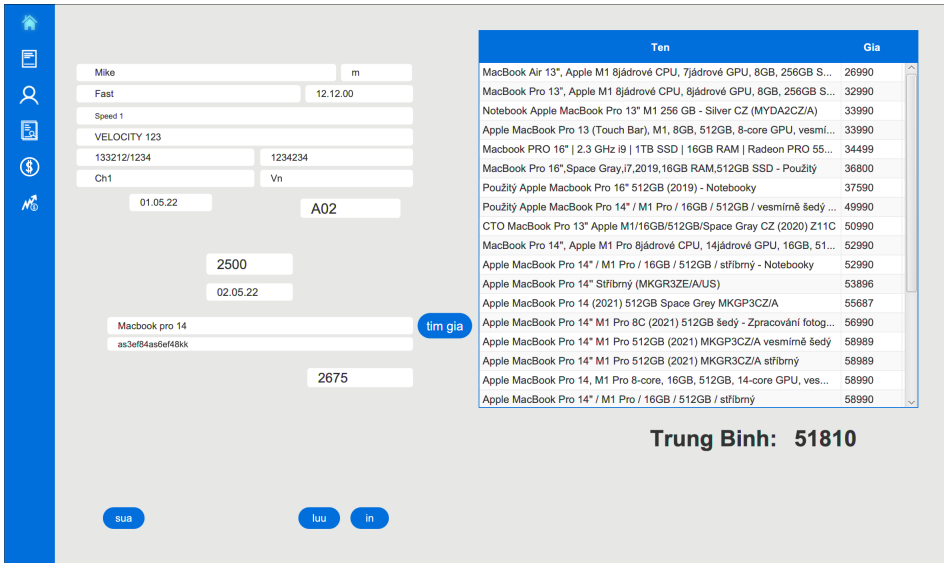
Příloha A obsahuje snímky všech obrazovek výsledné aplikace. Jelikož vzdálená databáze již obsahuje data reálných zákazníků a jejich smluv, byla aplikace spuštěna s databází lokální obsahující data na testování.



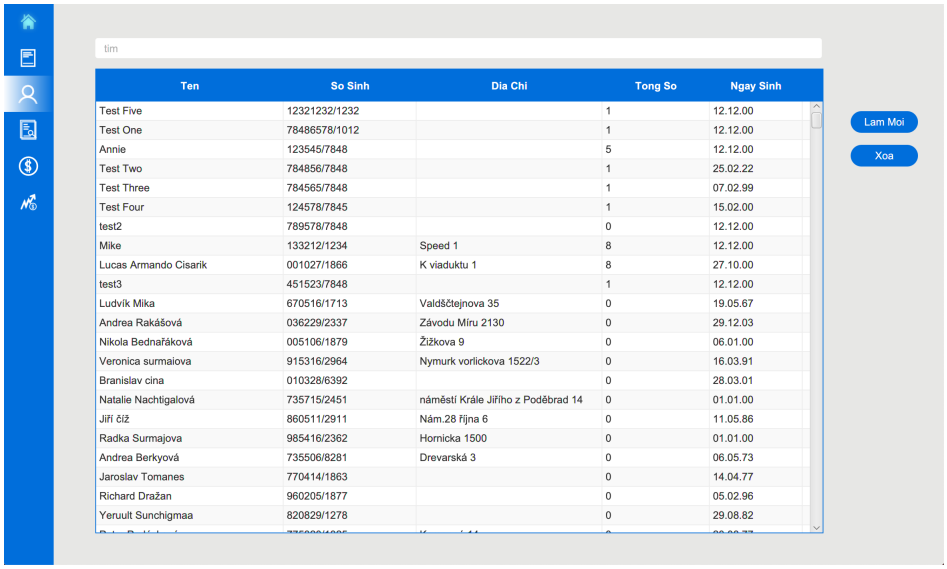
Obrázek A.1: Snímek aplikace – Domovská stránka



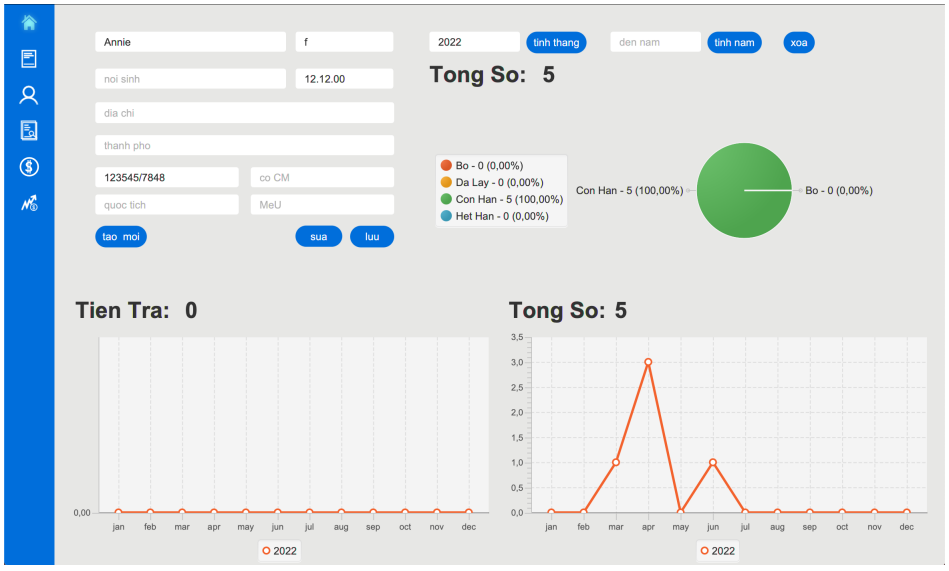
Obrázek A.2: Snímek aplikace – Zobrazení smluv



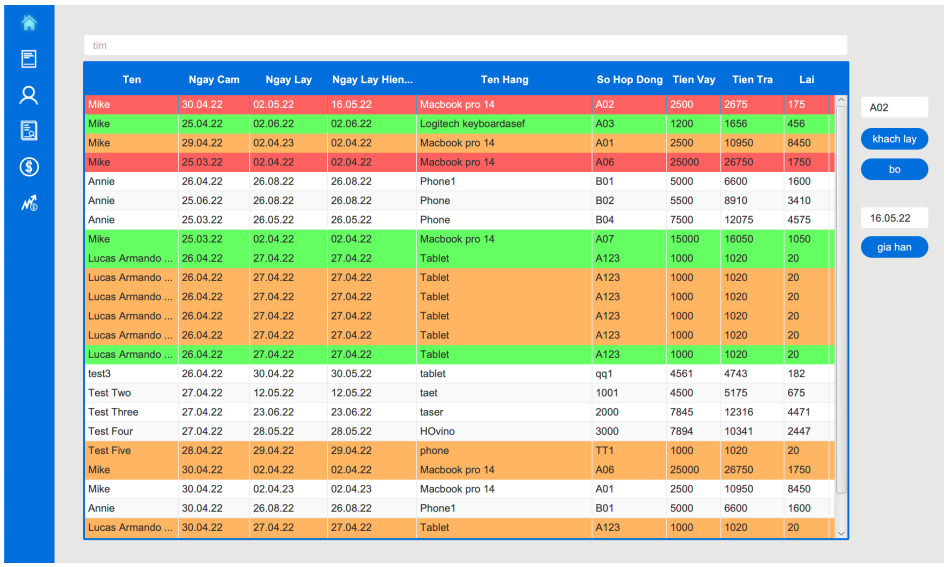
Obrázek A.3: Snímek aplikace – Vytvoření smlouvy s web scraping funkcí



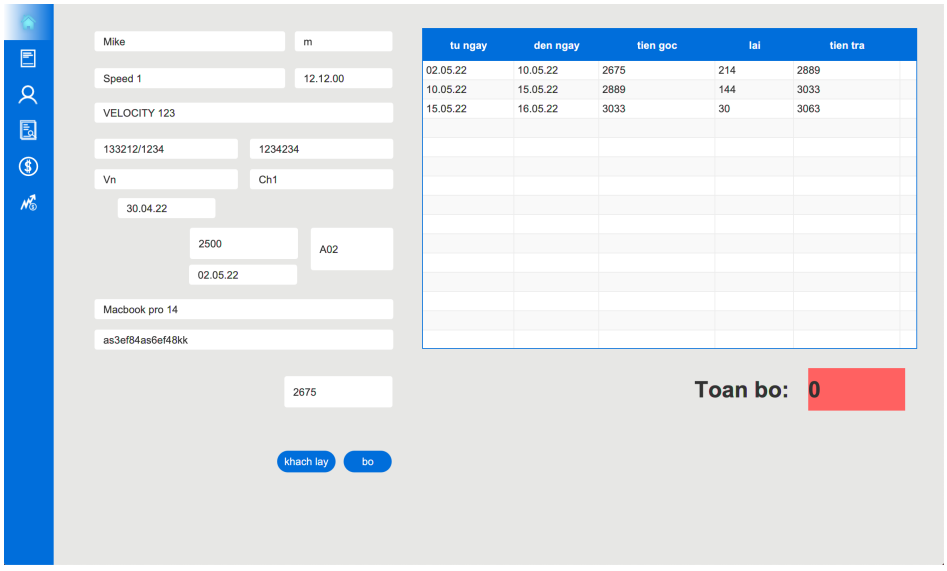
Obrázek A.4: Snímek aplikace – Zobrazení zákazníků



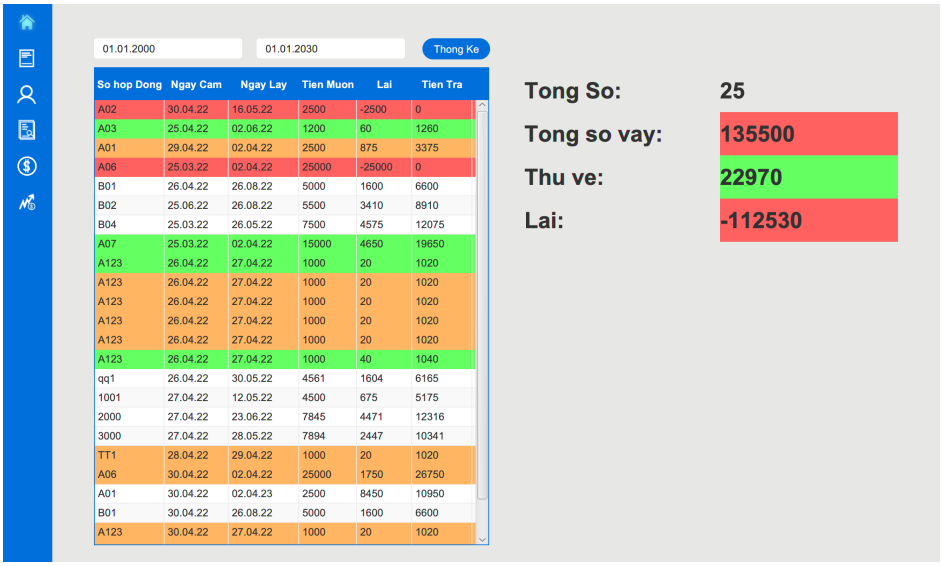
Obrázek A.5: Snímek aplikace – Vytvoření zákazníka s analytikou



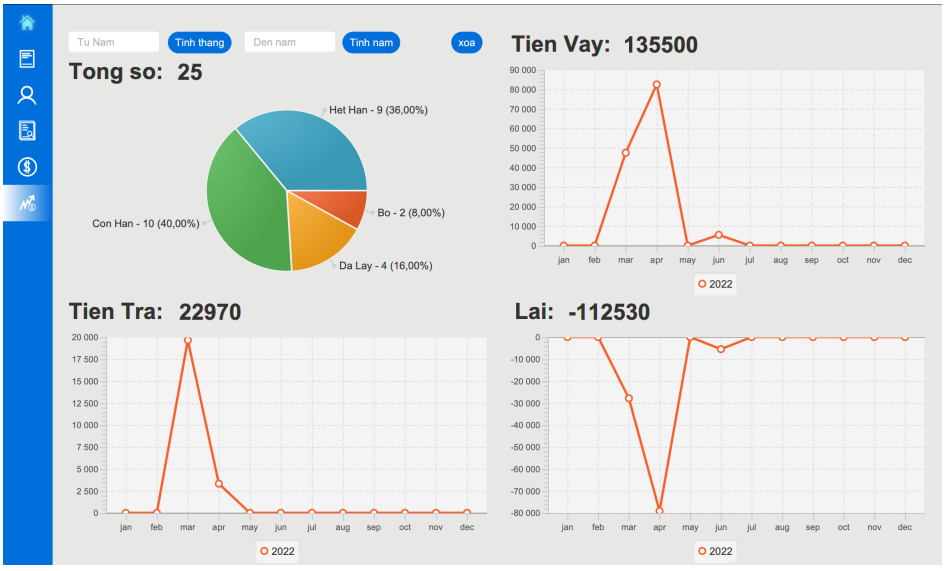
Obrázek A.6: Snímek aplikace – Zobrazení smluv a jejich stavů



Obrázek A.7: Snímek aplikace – Zobrazení historie prodloužení smlouvy



Obrázek A.8: Snímek aplikace – Kalkulace příjmů, výdajů a výnosu či ztráty



Obrázek A.9: Snímek aplikace – Analytika smluv

Příloha B

Seznam použitých zkratek

HTML Hypertext Markup Language

API Application Programming Interface

URL Uniform Resource Locator

GUI Graphical user interface

XML Extensible markup language

FXML XML-based language for JavaFx

POM Project Object Model

USB Universal Serial Bus

MSI Microsoft Windows Installer

RAM Random-Access Memory

Příloha C

Obsah přiloženého SD

- `readme.txt` ..... stručný popis obsahu SD
- `installer` ..... instalační balíčky
  - `ContractManager-1.0.msi` ..... instalační balíček pro Windows
  - `ContractManager-1.0.dmg` ..... instalační balíček pro MacOS
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `manual` ..... zdrojová forma příručky ve formátu MD
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `installation_manual.pdf` ..... instalační příručka ve formátu PDF