Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Informační systém fit centra

Jiří Vrbas

Diplomová práce

2019

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2018/2019**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Jiří Vrbas**
Osobní číslo:	**I16247**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Informační systém fit centra**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování:

Cílem této práce bude návrh informačního systému a vytvoření funkční aplikace, která bude obsahovat nástroje sloužící pro evidenci a řízení fitcentra. Systém bude umožňovat tyto funkcionality: přístup uživatelů do systému podle jejich práv, evidenci vstupů, evidenci a správu pernamentek, evidenci a správu uživatelů, evidenci a obsluhu skladu a zobrazení statistik. V teoretické části by se měl student zabývat problematikou Informačního systému (IS), členění IS a rozboru jednotlivých tipů. Dále by měl student představit použité technologie a hlouběji probrat využití Entity Frameworku, využití Windows Presentation Foundation(WPF) knihovny pro tvorbu grafického rozhraní a použití Model View ViewModel(MVVM) návrhového vzoru. Pro vytvoření aplikace bude využit jazyk C#/.Net a SQL databáze.

Rozsah grafických prací:

Rozsah pracovní zprávy: **50-60 stran**

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

Seznam odborné literatury:

ARLOW, Jim a Ila NEUSTADT. UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky, Vyd. 1. Překlad Bogdan Kiszka. Brno: Computer Press, 2007, 567 s. ISBN 978-80-251-1503-9
SOMMERVILLE, Ian. Softwarové inženýrství, Brno: Computer Press, 2013, 680 s. ISBN 978-80-251-3826-7
PETZOLD, Charles, Mistrovství ve Windows Presentation Foundation: [aplikace = kód markup], Brno: Computer Press, 2008. Mistrovství. ISBN 978-802-5121-412

Vedoucí diplomové práce: **Ing. Karel Šimerda**
Katedra softwarových technologií

Datum zadání diplomové práce: **22. října 2018**
Termín odevzdání diplomové práce: **18. května 2019**

Ing. Zdeněk Němec, Ph.D.
děkan

Univerzita Pardubice Česká republika

prof. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 17. listopadu 2018

Prohlašuji:

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v univerzitní knihovně. Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 12. 5. 2019

Jiří Vrbas

Poděkování:

Chtěl bych poděkovat Ing. Karlu Šimerdovi za odborné konzultace, které mě vedly k úspěšnému dokončení diplomové práce.

ANOTACE

Diplomová práce se zabývá vývojem informačního systému pro fitness centrum. V práci jsou uvedeny dostupné softwarové produkty zabývající se touto problematikou. Jsou zde popsány použité komponenty a technologie. Součástí práce je vlastní implementace informačního systému v programovacím jazyce C#/.Net. Jednotlivé části systému jsou zde popsané a je vysvětlena jejich funkce.

KLÍČOVÁ SLOVA

informační systém, fitness, C#, SQL databáze, .Net, Entity Framework, MVVM návrhový vzor, Xaml

TITLE

Information system for fitness centrum

ANNOTATION

The thesis is focused on an information system for the fitness center. All available software products concerning these problems are included. Used components and technologies are described here. The content of this work is my own implementation of information system in programming language C#/.Net. The system components are described here and their functions are explained.

KEYWORDS

information system, fitness, C#, SQL database, .Net, Entity Framework, MVVM pattern, Xaml

OBSAH

SEZNAM ILUSTRACÍ

SEZNAM TABULEK

SEZNAM ZKRATEK A ZNAČEK

API	Application Programming Interface
GPS	Global Positioning System
HTML	Hypertext Markup Language
IS	Informační Systém systém
MVVM	Model View ViewModel
SQL	Structured Query Language
UI	User Interface
UML	Unified Modeling Language
WF	Windows Forms
WPF	Windows Presentation Foundation
XML	Extensible Markup Language

ÚVOD

V současné době se zdravý životní styl spojený s aktivním pohybem celosvětově stává stále populárnějším. Existuje spousta velkých společností provozujících fitness centra, ale i menších podnikatelů, kteří poskytují prostory pro jakýkoliv druh cvičení. Vždy je potřeba nějakým způsobem řídit klíčové procesy spojené s provozováním takového podniku. K tomu slouží informační systémy spravující veškeré aktivity ve fitness a dalších sportovních či relaxačních centrech. Právě takovým systémem pro řízení fitness centra se zabývá tato práce.

Během posledních několika let se podílím na vývoji informačních systémů z různých odvětví. Bohužel se vždy jednalo o systémy, které jsou vyvíjeny dlouhou dobu, a tudíž jsem neměl možnost se zapojit do procesu vývoje od samého počátku produktu. Proto jsem si navrhl toto téma, abych měl možnost vytvořit kompletní informační systém od zrodu myšlenky až po konečné nasazení produktu k zákazníkovi. Jelikož systémy, na kterých jsem doposud pracoval, byly vyvíjeny pomocí zastaralé technologie Windows Forms, chtěl bych čtenáři poskytnout popis a ukázku modernějších technologií v této oblasti. Proto se v práci zabývám moderním přístupem k vývoji informačních systémů zaměřeným na platformu Windows.

Hlavním cílem této práce je vytvoření funkčního a přehledného systému, který by mohl být použit v podnicích zabývajících se provozováním fitness centra. Systém je určen pro menší provozovny, kde si majitelé nechtějí pořizovat robustní systémy, často finančně nákladné a obsahující spoustu funkcí, které by zůstaly nevyužity. Dále práce obsahuje představení technologií používaných při tvorbě moderních desktopových aplikací.

Teoretická část obsahuje rešerši dostupných řešení a jejich porovnání se systémem vyvíjeným v rámci této diplomové práce. Jsou zde popsány metodiky vývoje softwaru a jeho životní cykly. Dále jsou zde popsány jednotlivé přístupy k vývoji systémů obecně a rozebrán návrh vlastní metodiky použitý pro tvorbu praktické části této práce.

Praktická část popisuje návrh a implementaci systému. Nejdříve je popsána analýza jako první fáze vývoje systému a jsou zde rozebrány uživatelské příběhy jako požadavky na výsledný systém. Poté jsou zde popsány použité technologie, odůvodnění jejich použití, případné ukázky kódu a použité knihovny třetích stran. Následuje popis struktury databázového schématu a popis jednotlivých tabulek.

Dále je rozebrána struktura celého projektu a je také popsáno a odůvodněno rozdělení částí systému. Nechybí popisy jeho jednotlivých modulů včetně ukázky jejich grafického rozložení a popisu funkcionality. Konec praktické části je věnován příručce k instalaci systému a ukázce zdrojových kódů. V závěru práce jsou shrnuty vytyčené cíle a jsou zde také uvedena možná rozšíření systému.

1 INFORMAČNÍ SYSTÉM FITNESS CENTRA

Informační systém jako takový je integrovaná sada komponentů pro sběr, ukládání a zpracování dat a pro poskytování informací, znalostí a digitálních produktů. Aby mohly obchodní společnosti a další organizace provádět a řídit své operace, spolupracovat se svými zákazníky a dodavateli a soutěžit na trhu, spoléhají se na informační systémy.

Řídicí systémy ve fitness centrech již dávno neslouží pouze k prodeji služeb a správnému vytištění účtenek. Dnes jsou tyto systémy tvořeny tak, aby byly výkonným pomocníkem při řízení a správě těchto center. Základní a prvořadou funkcí jakéhokoliv systému je vždy zjednodušení a zrychlení práce.

1.1 Dostupná řešení na trhu

V dnešní době je nabídka systémů zaměřených pouze na fitness centra poměrně malá. Na trhu se jich několik nachází, ale jsou robustní a poskytují funkce pro širokou škálu oborů. Pokud by zákazník chtěl menší systém splňující pouze jeho požadavky, nezbývá mu nic jiného, než si takovýto systém nechat udělat na míru. Takový druh systému poté bývá dosti finančně náročný. Na výběr jsou jak webové aplikace, tak jako v případě této práce desktopové aplikace. V další kapitole budou rozebrány nejvíce propagované systémy, informace o nich pochází z oficiálních stránek nebo přímo od tvůrců systému.

1.1.1 Clubspire

Systém Clubspire je vyvíjený a spravovaný společností InspireCZ. Jedná se o robustní informační systém, který zaštiťuje nepřeberné množství funkcionalit potřebných ke správě podniků zaměřených nejen na fitness, ale také například tanečních studií, aquaparků, wellness středisek, ski areálů, obchodů a restaurací.

Protože tento systém pomáhá se správou širokého spektra aktivit zákazníka, v režimu ENTERPRISE poskytuje velké množství funckionalit, které jsou v běžném provozu fitness centra nepotřebné. Tento systém je široce modulovatelný a každý zákazník si může v rámci svého předplatného povolit pouze ty moduly, které sám potřebuje. Všechny typy členství jsou v tomto případě zpoplatněné, tudíž zákazník platí pravidelně každý měsíc poplatek za provozování systému [30].

Mezi hlavní součásti systému patří:

1. marketingové nástroje,
2. kontroling provozu,
3. správa členství a permanentek,
4. strategické řízení,
5. odbavení a rezervace.

Vzhledem k velikosti je celý systém rozdělen do několika předplatných, přičemž každé z nich rozšiřuje to stávající o další moduly. Mezi ně patří například [30]:

1. rezervační systém,
2. e-platby,
3. e-mail a SMS,
4. ubytování,
5. permanentky,
6. řízení vstupů,
7. členství,
8. pokladní systém,
9. skladové hospodářství,
10. webové rezervace,
11. manažerské přehledy.

Clubspire je pravděpodobně nejkomplexnější systém, který je k dostání na českém trhu a který je vyvíjen českou společností. Spojuje jak desktopové, tak webové prostředí a zcela jistě uspokojí potřeby většiny zákazníků. Pro menší podniky bude ale nejspíše finančně náročný a nastavení tak rozsáhlého systému není jednoduché.

1.1.2 AFit

Systém AFit je vyvíjený společností EMDAT, která sídlí v Brně. V současné době se systém už dále nerozvíjí, ale provozovatel nadále poskytuje plnou zákaznickou podporu. Tento systém je zaměřený především na oblast sportovních aktivit a relaxačních služeb. Jako většina podobných systémů je i tento placený a je rozdělený do jednotlivých verzí. Omezení jednotlivých verzí je vázáno na velikost databáze a počet uživatelů, kteří smí systém využívat [31].

Základní verze se nazývá Lite a nabízí základní služby, jako jsou [31]:

1. evidence zákazníků,
2. evidence permanentek,
3. evidence skladu,
4. pokladna,
5. správa zboží,
6. adresář kontaktů.

Systém dále nabízí verze Lite PLUS, Standard, Profi a Profi PLUS. Každá z dalších verzí nabízí další přidané moduly, které rozšiřují celkové funkce systému.

Tento systém sice poskytuje většinu potřebných funkcí pro správu podniku, ale ve verzích Lite je dosti jednoduchý, a zákazník by proto mohl narazit na různé nedostatky. Také grafická stránka systému je zastaralá a odpovídá systémům tvořeným před 15 lety.

1.1.3 Member PRO

Systém Member PRO je další z menších systémů podporujících vedení a správu sportovních zařízení, studií a klubů. Tento systém je vyvíjený firmou Luxart s.r.o, která se zabývá především prodejem osvětlovací techniky, ale v průběhu let si vybudovala vlastní oddělení zaměřené na vývoj softwaru a jedním z jejích děl je právě systém Member PRO [32].

Tento systém poskytuje základní funkce, které by mohl cílový zákazník využívat. Mezi nejvyužívanější funkcionality patří [32]:

1. správa klientů,
2. správa aktivit,
3. správa skladu,
4. prodej zboží,
5. prodej služeb,
6. správa recepce,
7. rezervace,
8. statistiky.

Tento systém je poněkud zastaralý a v dnešní době graficky nedostačující. Funkce systému ovšem dokáží splnit většinu požadavků, které jsou na tento typ systémů kladeny.

1.1.4 Ostatní systémy

Výše byly popsány systémy určené výhradně pro českého zákazníka, ale na trhu jsou i systémy, které jsou vyvíjeny mimo Českou republiku a které necílí pouze na český trh. Mezi takové systémy patří například:

1. GUBI,
2. Perfect Gym,
3. Rhino fit,
4. Vagaro,
5. iGym.

1.2 Porovnání s ostatními systémy

Níže naleznete tabulku Tabulka 1 – Porovnání s ostatními systémy, kde můžete vidět srovnání systémů dostupných na českém trhu a systému vyvíjeného v rámci této diplomové práce. Z tabulky je vidět, že mnou navržený systém se může svými funkcemi vyrovnat již zavedeným systémům.

Tabulka 1 – Porovnání s ostatními systémy

Funkce	FitIS	Clubspire	Afit	Member PRO
Prodej	Ano	Ano	Ano	Ano
Statistiky	Ano	Ano	Ano	Ano
Správa permanentek	Ano	Ano	Ano	Ano
Správa zákazníků	Ano	Ano	Ano	Ano
Správa uživatelů	Ano	Ano	Ano	Ne
Správa skladů	Ano	Ano	Ne	Ne
Správa zboží	Ano	Ne	Ne	Ne
Rozesílání zpravodajů	Ano	Ano	Ne	Ne

2 METODIKY VÝVOJE SOFTWARU

Návrh softwarových řešení je stejně jako návrhy jiných složitých systémů poměrně obtížnou disciplínou, a to zejména při vývoji produktu, na němž pracují větší skupiny vývojářů, analytiků, testerů apod. Veškeré činnosti je nutné koordinovat a každý člen týmu by měl přesně znát, co má dělat a jak bude jeho práce propojena s ostatními členy týmu. K tomuto účelu slouží různé nástroje a postupy, jež jsou součástí právě metodiky vývoje. Tyto metodiky jsou založeny na takzvaném modelování s využitím různých nástrojů k modelování určeným, například UML, CASE nástroje a další [1].

2.1 Životní cykly vývoje softwaru

Životní cykly vývoje softwaru neboli Software Developement Life Cycle (SDLC), se skládá z jednotlivých fází neboli etap. Jedná se o posloupnost jednotlivých kroků, které vedou k dokončení výsledného softwaru. SDLC je proces používaný softwarovým průmyslem pro navrhování, vývoj a testování vysoce kvalitního softwaru. Klade si za cíl vyrábět vysoce kvalitní software, který splňuje nebo překračuje očekávání zákazníků, dosahuje dokončení v časech a odhadech nákladů. Je to rámec definující úkoly prováděné v každém kroku procesu vývoje softwaru. Skládá se z podrobného plánu, který popisuje, jak vyvíjet, udržovat, nahrazovat a měnit nebo vylepšovat konkrétní software. Životní cyklus definuje metodiku pro zlepšení kvality softwaru a celkového procesu vývoje [14].

Typický životní cyklus vývoje softwaru se skládá z následujících fází:

1. Plánování a analýza požadavků – Provádí jej vedoucí členové týmu se vstupy od zákazníka, obchodního oddělení, průzkumů trhu a odborníků v oboru. Tyto informace jsou pak využity k plánování základního projektového přístupu a k provedení studie proveditelnosti produktu v ekonomické, provozní a technické oblasti.
2. Definování požadavků – Po provedení analýzy požadavků je dalším krokem jasně definovat a dokumentovat požadavky na produkt a získat je od zákazníka nebo analytiků. To se provádí pomocí dokumentu SRS (Software Requirement Specification), který se skládá ze všech požadavků na výrobek, jež mají být navrženy a vyvinuty během životního cyklu projektu.

3. Návrh architektury produktu – SRS je referencí pro produktové architekty, aby přišli s nejlepší architekturou pro produkt, který má být vyvinut. Na základě požadavků specifikovaných v SRS je obvykle navrženo a zdokumentováno více než jeden návrhový přístup pro architekturu produktu.
4. Vývoj produktu – V této fázi SDLC začíná vlastní vývoj produktu. Pokud je návrh prováděn podrobným a organizovaným způsobem, může být psaní kódu provedeno bez větších potíží.
5. Testování produktu – Tato fáze je obvykle podmnožinou všech fází jako u moderních SDLC modelů, testovací aktivity jsou většinou zapojeny do všech fází. Tato etapa se však týká pouze zkušebního stadia výrobku, kde jsou vady produktu hlášeny, sledovány, fixovány a testovány.
6. Nasazení a údržba – Jakmile je produkt otestován a připraven k jeho nasazení, je oficiálně uvolněn. Někdy se nasazení produktu děje ve fázích podle obchodní strategie. Produkt může být nejprve vydán v omezeném segmentu a testován v reálném podnikatelském prostředí. Na základě zpětné vazby může být uvolněn tak, jak je, nebo s navrhovanými vylepšeními v segmentu cílového trhu. Po uvedení výrobku na trh je jeho údržba prováděna pro stávající zákaznickou základnu.

Jsou definovány a navrženy různé modely životního cyklu vývoje softwaru, sledované během tohoto procesu. Každý model navazuje na řadu kroků, které jsou jedinečné pro svůj typ, aby zajistily úspěch v procesu vývoje softwaru. Tyto modely jsou blíže popsány v kapitole 2.2.

Při vývoji softwaru je nutné zajistit správné řízení a koordinaci činností členů týmu. Čím složitější projekt je, tím je kladen větší důraz na správné složení týmů. V jednotlivých týmech je každému členovi přiřazena jeho role, pracovní náplň a kompetence.

Týmy se během procesu vývoje mohou lišit a měnit složení. Toto odlišení může být závislé na použité metodice nebo na tom, v jaké fázi vývoje se projekt nachází. Často také členové týmu zastávají více rolí [1]. Jednotlivé role je možné zařadit do určité kategorie profese spojené s vývojem softwaru:

1. Manažer projektu – zodpovídá za plánování, organizování a realizaci projektu.
2. Vedoucí týmu – má na starosti řízení celého týmu a plánování jednotlivých kroků v rámci projektu. Také spolupracuje s projektovým manažerem a odpovídá za výsledky práce celého týmu.

3. Analytik – zpracovává specifikaci požadavků na systém, a to do takových detailů, aby bylo možné zpracovat kvalitní návrh produktu.
4. Softwarový architekt – navrhuje technologie a postupy, s jejichž pomocí bude systém vyvíjen.
5. Návrhář – spolupracuje s analytikem a odpovídá za návrh systému, aby byl v souladu s požadavky. Také převádí uživatelské požadavky na systémové požadavky.
6. Vývojář – odpovídá za tvorbu programového kódu.
7. Tester – odpovídá za přípravu testovacích scénářů a za samotné testování systému.
8. Dokumentarista – zpracovává výslednou dokumentaci.
9. Správce – zajišťuje funkčnost a bezpečnost během provozu.

2.2 Specifické přístupy metodik softwarového vývoje

Od té doby, co se začaly informační technologie rozmáhat a začaly se vyvíjet různé typy softwarů, bylo vyvinuto mnoho vývojových postupů. V dalších kapitolách bude uveden popis nejčastěji používaných.

2.2.1 Vodopádový přístup

Jedná se o sekvenční vývojový proces, tedy o posloupnost navazujících etap. Tento přístup je spíše teoretický a slouží jako základní myšlenkový postup pro studium etap životního cyklu. Při tomto přístupu je kladen velký důraz na plánování, termíny a rozpočty [24].

Výhody:

1. jednoduchý z hlediska řízení,
2. nejlepší struktura výsledného produktu, pokud se požadavky nemění.

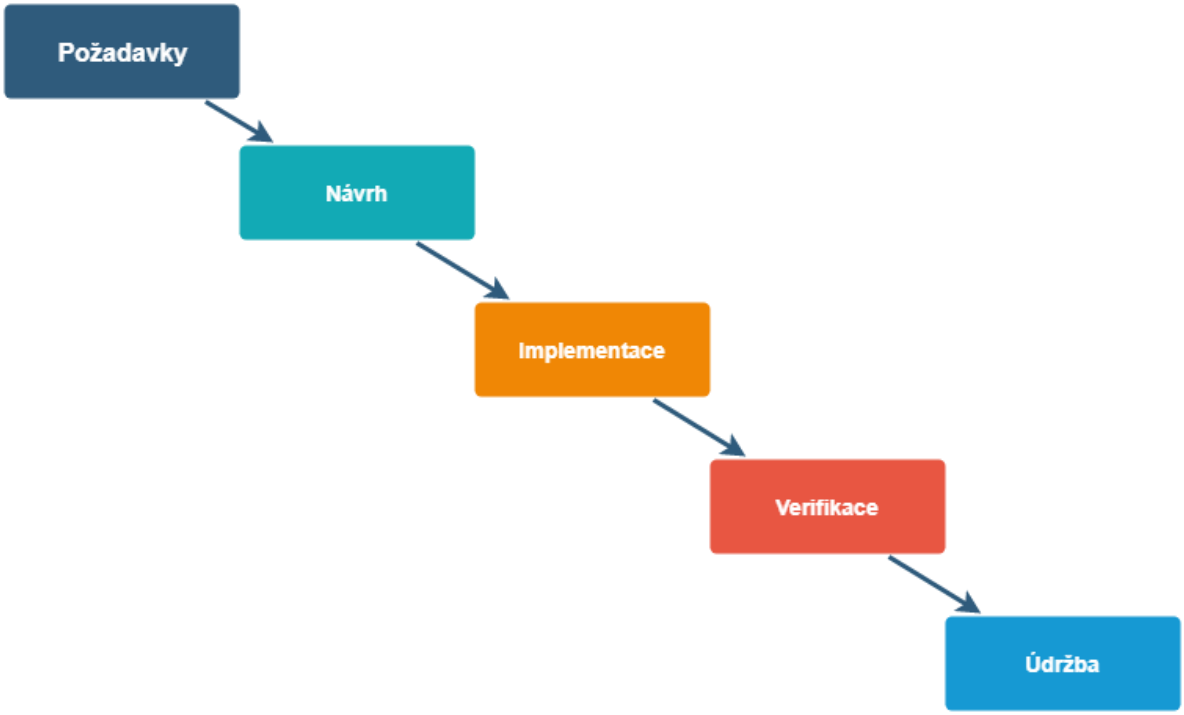
Nevýhody:

1. zákazník nedokáže stanovit všechny požadavky předem,
2. pokud se požadavky změní, prodlouží se doba realizace,
3. zákazník dostane spustitelnou verzi až na konci celého procesu, a tudíž jsou případné nedostatky odhaleny příliš pozdě [25].

Fáze přístupu:

1. specifikace zadání a stanovení požadavků,
2. analýza a návrh SW,

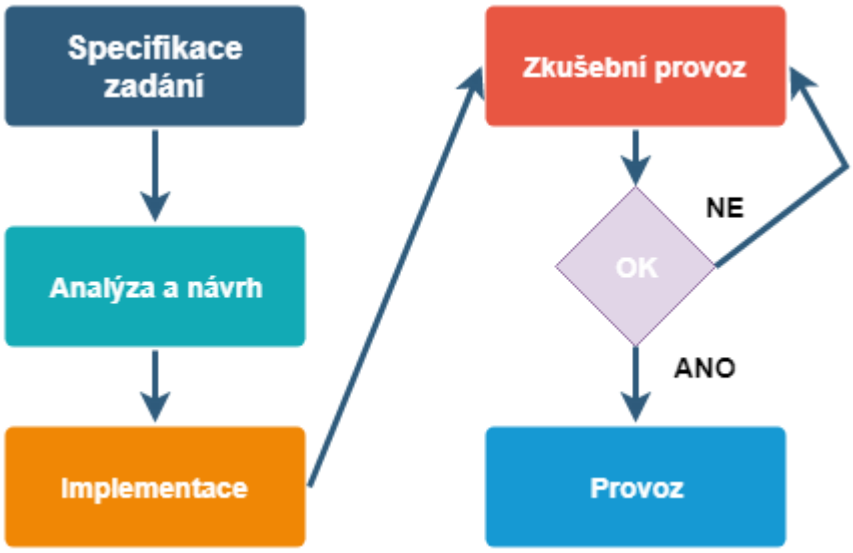
3. implementace a nasazení SW,
4. testování a integrace,
5. provoz a údržba.



Obrázek 1 – Fáze přístupu – vodopád

2.2.2 Prototypový přístup

Jedná se o přístup k vývoji systému, kde jsou v průběhu vývoje vytvářeny takzvané prototypy. Prototypem je myšlena částečná implementace produktu, která by měla reprezentovat všechna vnější rozhraní systému. Zákazník se s prototypem seznámí a testuje jej. Na základě připomínek jsou upravovány specifikace a požadavky na systém až do podoby výsledného systému. Jednotlivé stupně vývoje prototypu jsou plánované a jeho vývoj je řízen podle předem stanovených pravidel [1].



Obrázek 2 – Fáze přístupu – prototyp

2.2.3 Inkrementální přístup

Inkrementální přístup částečně využívá vodopádový model. Jedná se o kombinaci sekvenčních a iteračních metodik softwarového vývoje. Je to přístup, kdy jednotlivé části jsou vyvíjeny podle vlastního modelu a pomocí rozdělení celého projektu na tyto části se zaručí provádění jednotlivých změn v rámci implementace.

Obecné požadavky jsou definovány dříve, než je přikročeno k vývoji pomocí menších přírůstků. Výchozí koncept, analýza požadavků, design architektury a systémové jádro jsou zakončeny implementací prototypu jako funkčního systému [24].

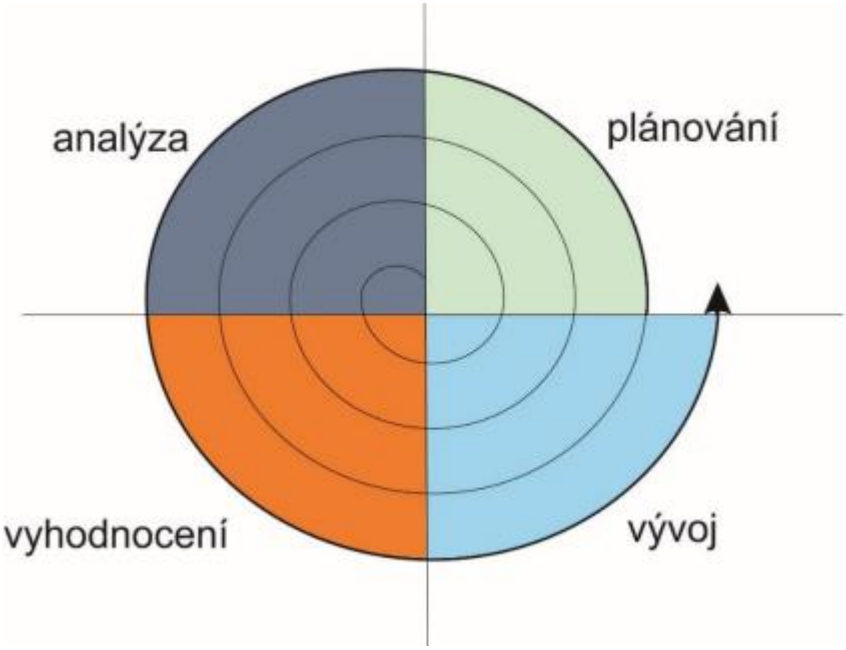
2.2.4 Iterativní přístup

Tento přístup zahrnuje vývoj SW v iteracích, což jsou opakování procesu vývoje v měnícím se kontextu. V průběhu každé iterace je vytvořena reálná část produktu, který má zákazník možnost po každé iteraci vyzkoušet, zhodnotit a zkontrolovat výsledek se svými požadavky. Díky tomu jsou mnohem rychleji odhaleny případné chyby či nedostatky systému a vývojový tým na ně může v další iteraci reagovat. Zadavatel je členem týmu vývoje a podílí se na kontrolních bodech vývoje.

2.2.5 Spirální přístup

Spirálový model je odvozen od vodopádového modelu. Kombinuje prvky designového a prototypového přístupu. Zaměřuje se na analýzu rizik a jejich minimalizaci tak, že je projekt rozdělen na menší segmenty a jsou umožněny změny během procesu vývoje. Každý cyklus

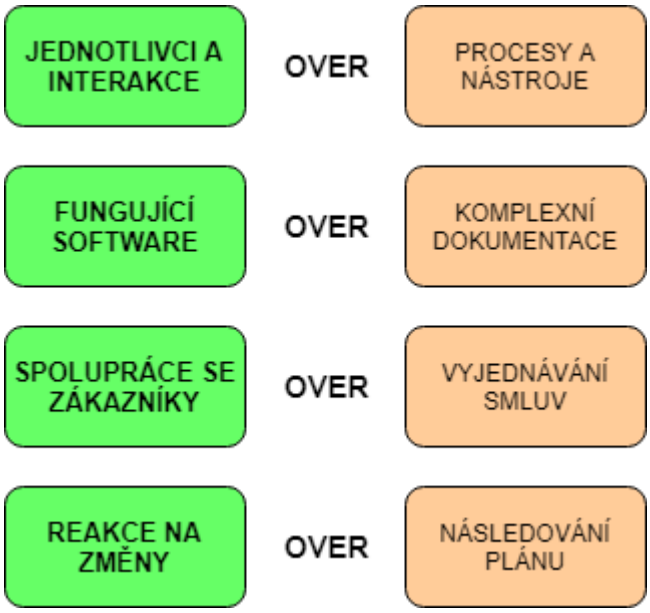
spirály spouští stejný sled kroků. Pokud chceme přejít do další fáze vývoje, je nutné provést důslednou analýzu. Rizikem je myšlena libovolná událost či situace, která by mohla mít dopad na projekt a která by jej mohla ohrozit. Každému riziku je v průběhu analýzy přirazena nebezpečnost a pravděpodobnost výskytu [24; 25].



Obrázek 3 – Fáze přístupu – spirála

2.2.6 Agilní přístup

Cílem agilního přístupu je zejména lepší organizace práce. Odlišné agilní přístupy vedou k odlišným produktům, protože považují jiné aspekty produktu za klíčové. Tento přístup vychází z „Agile Manifesto“, jenž vznikl na bázi dlouholetých zkušeností špičkových vývojářů a analytiků. Hlavní principy agilního manifestu jsou zobrazeny na Obrázek 4 – Agilní manifest.



Obrázek 4 – Agilní manifest

Díky tomuto přístupu dokáže manažer řídit více zaměstnanců, větší produkt nebo se více zaměřit na kvalitu. Z minulosti je známo, kolik práce je tým schopen udělat v rozsahu jedné iterace, a to díky tomu, že se dělají časové odhady na co nejkratší úkoly. Díky agilnímu přístupu je také mnohem jednodušší reagovat na změny zadání a zrychlí se doručování produktu zákazníkovi. Pomocí automatizovaných testů je zaručena kvalita dodaného produktu.

Agilní přístup k řízení projektů je interaktivní, pružný a přírůstkový. V praxi to znamená těsnou a neustálou spolupráci mezi projektovým týmem, který vytváří průběžné prototypy, a mezi zákazníkem, který dává zpětnou vazbu, na jejímž základě se upřesňuje zadání. Agilní řízení projektů se proto uplatňuje u velmi komplexních systémů, u kterých se detailní požadavky tvoří nebo upřesňují průběžně na základě zkušeností s prototypy z jednotlivých iterací.

Při agilních metodách práce se realizují malé porce výsledků (prototypy) v každém vývojovém cyklu v těsné spolupráci se zákazníkem [26].

Metody agilního přístupu [27]:

1. adaptivní vývoj softwaru,
2. disciplinované agilní doručení,
3. metoda dynamického vývoje systému,
4. vývoj řízený funkcemi,
5. Lean Kanban,

6. SCRUM,
7. programováním řízené testy,
8. extrémní programování.

2.3 Návrh vlastní metodiky

Metodiky popsané v kapitole 2.2 jsou vhodné především pro skupiny více lidí nebo pro celé vývojové týmy. Jelikož je tato práce vyvíjena pouze jednou osobou, není možné řídit se přímo jednou metodikou. V takovém případě je zapotřebí zvolit vhodný způsob vývoje pomocí různých postupů, vedoucích k dokončení projektu.

Během tvorby celého systému bylo vybráno a vyzkoušeno několik postupů, avšak ne všechny se nakonec jevily jako správné, a proto od nich bylo upuštěno. Zhodnocení použitých postupů naleznete v Tabulka 2 – Použité postupy [28].

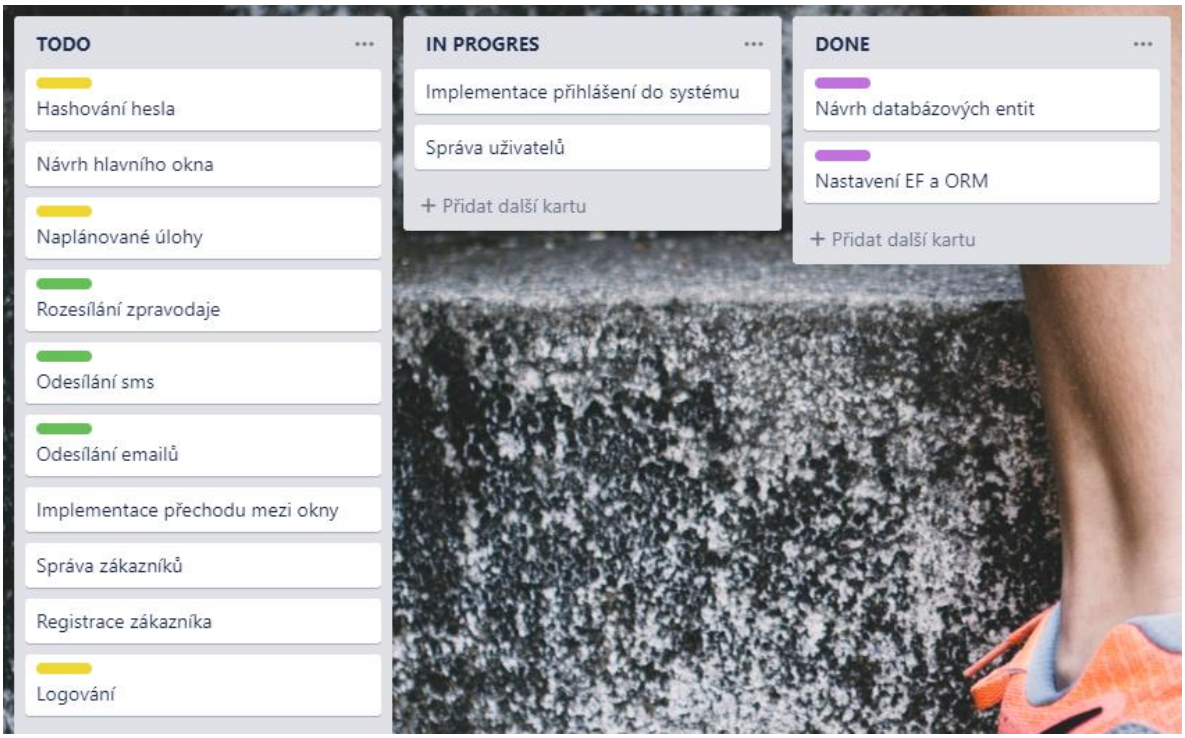
Tabulka 2 – Použité postupy

Číslo postupu	Popis	Zhodnocení
1	Kanban	Užitečné
2	Prioritizovaná tabulka požadavků na funkce	Neužitečné
3	Definice akceptačních kritérií	Užitečné
4	Používat řízení verzí	Užitečné
5	Vytváření demoverzí	Neužitečné
6	Používání externích knihoven	Užitečné
7	Správný návrh systému	Užitečné
8	Používání automatických testů	Neužitečné
9	Refaktorování kódu	Užitečné
10	Extrémní programování	Neužitečné
11	Užití správných nástrojů	Užitečné

Ze všech vyzkoušených postupů se mi jako nejefektivnější jeví Kanban, díky kterému jsem byl schopen celý systém dokončit ve stanoveném čase a kvalitě. Kanban je japonské slovo pro „vizuální signál" neboli „kartičky". Vizuální znázornění celého procesu denně pomáhá lidem při komunikaci a umožňuje další optimalizace.

Základními pravidly pro použití této metody jsou:

1. Vizualizovat – Kanban se obvykle vykresluje jako tabulka. Ukázku takovéto tabulky, která byla použita při tvorbě systému fitness centra, můžete vidět na Obrázek 5 – Kanban tabulka projektu.
2. Minimalizovat čas průchodu – Pokud je čas průchodu menší, zrychlí se dokončení úkolu, ale ztrácí se efektivita. Naopak pokud je čas průchodu zvolen větší, dokončení úkolu bude trvat déle a osoba pracující na daném úkolu může ztrácet soustředění.
3. Limitovat WIP (Work In Progress) – Jedná se o počet úkolů, na kterých se pracuje ve stejnou chvíli.



Obrázek 5 – Kanban tabulka projektu

Tato tabulka byla vytvořena pomocí nástroje Trello. Tabulka obsahuje v prvním sloupci frontu úkolů, které je potřeba dokončit. Jsou seřazené podle priority a můžou být barevně odlišeny podle typu. V dalším sloupci se nachází úkoly, na kterých se aktuálně pracuje. Pro evidenci času, který byl na úkolu stráven, je možné použít například tečky symbolizující počet dní strávených na úkolu. Poslední sloupec znázorňuje úkoly, jež byly již dokončeny.

V této kapitole byla popsána možná řešení problému vývoje softwaru jednou osobou. Jako optimální řešení se mi jeví použít upravené části agilního přístupu ke konstrukci procesního rámce, který pak určuje procesy vývoje. To vývojáři dovolí, soustředit se na podstatné úkoly a efektivně si zorganizovat práci. Dále je potřeba tento přístup doplnit o další postupy, které vyhovují konkrétnímu vývojáři. Za důležité však považuji efektivní správu času, psaní čistého kódu a návrh jednoduché architektury. Také použití vhodných nástrojů může vést k efektivnější práci [28].

3 VLASTNÍ ŘEŠENÍ IS

Praktická část této práce je zaměřena na tvorbu aplikace informačního systému fitness centra. Aplikace umožňuje zpracovávat úkoly související se správou a řízením fitness centra, evidenci a správu uživatelů, permanentek, skladových zásob a dalších funkcí potřebných k provozu. Dále lze jejím prostřednictvím získat výstupy ve formě statistik nebo rozesílání zpravodaje. V této kapitole je aplikace rozebrána a jsou popsány použité technologie a vysvětleny dílčí části systému. K práci jsou také přiloženy kompletní zdrojové kódy a příručka k instalaci.

3.1 Analýza a uživatelské příběhy

Před započetím vývoje bylo nutné provést předběžnou analýzu a specifikaci. Cílem bylo sestavit základní rámec požadavků na systém. Během této fáze byly sesbírány požadavky uživatelů a byly zjištěny požadované vstupní a výstupní informace. Dále byl vytvořen seznam možných problémů, jejich důsledků a nástin řešení. Výsledkem této části práce byla specifikace, účel systému, identifikace uživatelů a jejich zásadní požadavky, definice částí systému a návrh jejich řešení, odhady datové základny, technického a softwarového zajištění.

Důležitou částí vývoje systému je analýza. Během ní je nutné důkladně probrat každou část systému, požadavků a předchozích problémů. Chyby, které se udělají nebo neodhalí během analýzy, jsou velice zásadní, protože se dotýkají samotné struktury systému a později je náročné a pracné jejich odstranění. To s sebou nese časovou a tím pádem i finanční náročnost [2].

Na konci analýzy by měl mít analytik dokonalý přehled o systému. Musí vědět, pro koho je určen, kdo s ním bude pracovat, pro jaká data je určen a jakým způsobem se s nimi bude pracovat.

Součástí analýzy jsou v našem případě uživatelské příběhy. Ve vývoji softwaru a produktovém managementu je uživatelský příběh přirozeným popisem jedné nebo více funkcí softwarového systému. Uživatelské příběhy jsou často psány z pohledu koncového uživatele nebo uživatele systému. V závislosti na projektu mohou být napsány různými zúčastněnými stranami včetně klientů, uživatelů, manažerů nebo členů vývojového týmu.

Při psaní uživatelských příběhů je možné vycházet z několika často používaných šablon:

1. Connextra – `Jako <role uživatele> chci < mít schopnost>, abych <něco mohl>`,
2. Chris Matts – `Abych <něco získal> jako <role uživatele>, můžu <něco udělat>`,
3. Elias Weldemichael – `Jako <role uživatele>, mohu <získat>, protože <chci něco udělat>`,
4. Five Ws – `Jako <někdo> <kdy> <kde> chci, protože <něco>`,
5. Rachel Davies' – `Jako <někdo>, můžu <něco> abych mohl <něco>`.

Uživatelské příběhy, které jsou součástí analýzy, jsou zobrazeny v následující tabulce spolu s odkazy na kapitoly, kde je dokázáno, že byl daný uživatelský příběh splněn. Pro jejich vytvoření byla zvolena šablona Connextra.

Tabulka 3 – Uživatelské příběhy

Číslo	Popis	Kapitola	Splněno
1	Jako obsluha chci mít možnost registrovat nového uživatele do systému, aby mohl dostat kartičku pro vstup do posilovny.	3.6.4	Ano
2	Jako obsluha chci zaregistrovat uživatele při vstupu do posilovny, abychom věděli, kdy uživatel přišel.	3.6.3	Ano
3	Jako obsluha chci zaregistrovat uživatele při odchodu z posilovny, abychom věděli, kdy odešel.	3.6.3	Ano
4	Jako obsluha chci dobít uživateli vstupy na jeho účet, aby mohl vstoupit do posilovny.	3.6.8	Ano
5	Jako obsluha chci prodat neregistrovanému uživateli jeden vstup do posilovny, aby mohl přijít i nezaregistrovaný člověk.	3.6.3	Ano
6	Jako obsluha chci nastavit uživateli časovou permanentku, aby si nemusel hlídat vstupy do posilovny.	3.6.7	Ano
7	Jako obsluha chci být upozorněn, pokud uživateli docházejí vstupy, nebo se blíží čas vypršení časové permanentky, abych mohl uživatele upozornit.	3.6.3	Ano
8	Jako obsluha chci mít možnost prodat naše zboží zákazníkovi.	3.6.5	Ano
9	Jako manažer chci mít přehled o tom, kdo je aktuálně přítomen v posilovně, abychom mohli sledovat jednotlivé zákazníky.	3.6.3	Ano
10	Jako manažer chci mít přehled o tom, kolik lidí bylo v posilovně v určité dny, abychom mohli používat informace k jednotlivým dnům.	3.6.6	Ano

11	Jako manažer chci mít možnost nastavovat ceny a platnost jednotlivých permanentek, abych mohl ovlivňovat cenovou politiku systému.	3.6.7	Ano
12	Jako skladník chci mít přehled, kolik jednotek zboží se nachází na skladu, abych mohl případně zásoby doplnit.	3.6.11	Ano
13	Jako skladník chci mít možnost přidat nové zboží do skladu, abychom mohli rozšířit sortiment k prodeji.	3.6.15	Ano
14	Jako skladník chci mít možnost doplnit množství zboží, abychom měli neustále co prodávat.	3.6.15	Ano
15	Jako manažer chci mít možnost spravovat zaměstnance, abychom mohli přijímat a propouštět nové zaměstnance	3.6.10	Ano
16	Jako manažer chci mít možnost přiřadit novému zaměstnanci práva k přístupu do systému, aby zaměstnanci neměli možnost přístupu do celého systému.	3.6.10	Ano
17	Jako manažer chci mít možnost vytvořit zpravodaj, abych mohl informovat zákazníky o akcích fitness centra.	3.6.13	Ano

3.2 Použité technologie

Jelikož výsledkem praktické části práce je informační systém, který je určen pro počítače s operačním systémem Windows od verze 98 a vyšší, byly zvoleny technologie nejvíce vyhovující tomuto typu aplikací. Zdrojový kód je psán v jazyku C# za použití frameworku .Net, jenž poskytuje základní funkcionality a knihovny potřebné pro běh nejen desktopových aplikací, ale také třeba webových aplikací. Nejnižší nainstalovaná verze .Net frameworku potřebná pro běh tohoto systému je 4.6.1. Systém si ukládá data do Microsoft SQL databáze, kterou může mít každá instance systému vlastní, nebo může být pro více podniků společná. Popisem jednotlivých technologií, jejich představením, případně ukázkami použití se zabývají další kapitoly této práce.

3.2.1 C#/.Net

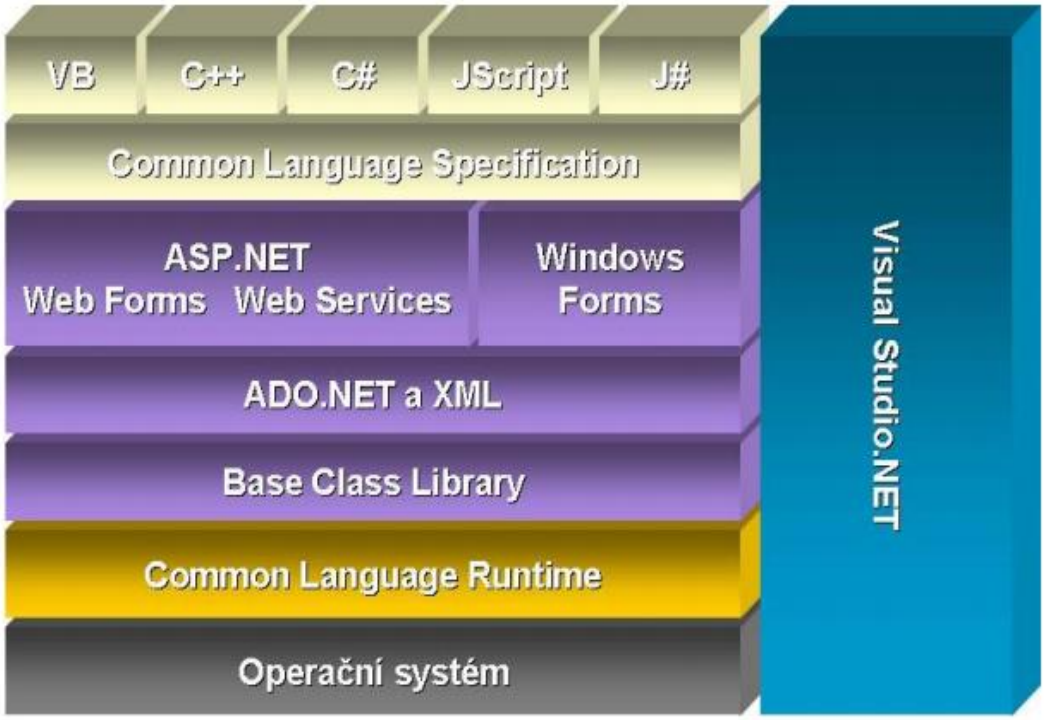
Pro vývoj celého systému byl použit programovací jazyk C# a jeho technologie. Jazyk C# vyvinula firma Microsoft. Byl představen spolu s celým vývojovým prostředím .Net, což jsou vlastně čtyři základní věci: jazyk, Visual Studio[1], CLR[2] a knihovny[3]. Struktura frameworku je znázorněna na obrázku Obrázek 6 – Architektura .Net Framework. Jak název napovídá, vychází tento jazyk v mnohém z programovacího jazyka C/C++, ale v mnoha ohledech je daleko bližší programovacímu jazyku Java [3; 15].

[1] Prostředí, ve kterém píšeme zdrojový kód a které pomáhá s vývojem
[2] Virtuální stroj, interpretuje CIL do instrukcí fyzického procesoru
[3] Předpřipravené sady struktur a komponent

Základní charakteristiky:

1. čistě objektově orientovaný,
2. obsahuje nativní podporu komponentového programování,
3. obsahuje pouze jednoduchou dědičnost s možností mnohonásobné implementace rozhraní,
4. vedle členských dat a metod přidává vlastnosti a události,
5. správa paměti je automatická (o korektní uvolňování zdrojů se stará garbage collector),
6. podporuje zpracování chyb pomocí výjimek,
7. zajišťuje typovou bezpečnost a podporuje řízení verzí,
8. podporuje atributové programování.

.Net Framework funguje jako základ, na němž lze vytvářet software. Jeho jádro je založené na principech objektově orientovaného programování a všechny služby zpřístupňuje široké škále programovacích jazyků. To znamená, že není podstatné, ve kterém programovacím jazyce komponenty vytváříme, případně jaké komponenty používáme. Tento framework dále řeší problémy s bezpečností, nasazováním a instalací aplikací.



Obrázek 6 – Architektura .Net Framework [29]

Pro snadnější správu balíčků využívá .Net framework balíčkovací systém NuGet. NuGet je důležitý nástroj pro libovolnou platformu moderního vývoje, pomocí kterého můžete vytvářet,

sdílet a využívat užitečné kódy od různých vývojářů. Takový kód je často seskupen do balíčků, které obsahují zkompilovaný kód společně s dalším obsahem potřebným v projektech využívajících tyto balíčky.

Pro platformu .Net, je tento mechanismus podporovaný společností Microsoft a definuje, jak jsou balíčky pro .Net vytvořeny, hostovány a využívány a poskytuje nástroje pro každou z těchto rolí. Protože NuGet podporuje privátní hostitele spolu s nuget.org veřejnými hostiteli, můžete sdílet kód, který je určen výhradně pro organizaci, vaše projekty, nebo pracovní skupinu [4].

3.2.2 MS-SQL

Jazyk SQL neboli Structured Query Language byl vyvinut firmou IBM na počátku 70. let jako dotazovací jazyk pro práci s velkými databázemi. Cílem tvůrců bylo vyvinout takový nástroj, který by jim umožnil vybírat data z databáze přesně podle jejich individuálních požadavků a přitom byl co nejjednodušší. Jedná se o databázový programovací jazyk, který umožňuje definovat data, dotazovat se na ně a manipulovat s nimi.

MS-SQL je takzvaná relační databáze, což je databáze založená na tabulkách. Každá tabulka obsahuje položky jednoho typu. Položky jsou ukládány na jednotlivé řádky, sloupce pak označují vlastnosti, které položky mají. Tento typ databáze je typovaný, tudíž každý sloupec může obsahovat hodnoty pouze jednoho typu.

Vlastnosti MS-SQL:

1. atomicita – operace v transakci se provedou jako jedna nedělitelná operace,
2. konzistence – stav databáze po dokončení transakce je vždy konzistentní,
3. izolace – jednotlivé operace jsou izolované a navzájem se neovlivňují,
4. trvanlivost – všechna data jsou okamžitě zapsána na trvanlivá úložiště.

Operace prováděné jazykem SQL:

1. Jazyk pro definici dat (DDL) – Těmito příkazy se vytvářejí struktury databáze – tabulky, indexy, pohledy a další objekty. Vytvořené struktury lze také upravovat, doplňovat a mazat. Mezi tyto příkazy patří CREATE, ALTER, DROP.
2. Jazyk pro manipulaci s daty (DML) – Jsou to příkazy pro získání dat z databáze a pro jejich úpravy. Mezi tyto příkazy patří SELECT, INSERT, UPDATE, MERGE, DELETE, EXPLAIN, SHOW.
3. Jazyk pro řízení dat (DCL) – Do této skupiny patří příkazy pro nastavování přístupových práv. Mezi tyto příkazy patří GRANT, REVOKE.
4. Jazyk datových transakcí (DTL) – Do této skupiny patří příkazy pro řízení transakcí. Mezi tyto příkazy patří START TRANSACTION, COMMIT, ROLLBACK.

3.2.3 Windows Presentation Foundation

Widnows Presentation Foundation (WPF) je framework pro kompletní tvorbu formulářových aplikací, který je součástí .Net frameworku od verze 3.0. Poskytuje širokou škálu hotových formulářových prvků a umožňuje různorodé stylování vzhledu aplikace. Díky architektuře je také možné velice jednoduše vytvářet vlastní prvky, které Microsoft nazval controls. Může se jednat například o vlastní tlačítka, pole, posuvníky nebo tabulky. Kromě WPF je v .Net frameworku stále možné používat starší technologii nazvanou Windows Forms (WF).

Postupem času jsou kladeny stále větší nároky na grafický design a v tomto ohledu přestala původní technologie WF stačit. Hlavní důvody zavedení nové technologie WPF:

1. V poslední době začali uživatelé mnohem více využívat mobilní zařízení, ale u WF aplikací je problém přizpůsobit jejich fyzické velikosti kvůli slabé podpoře DPI. Není možné používat jednu a tu samou aplikaci například na mobilu, tabletu a počítači. Proto WPF zavedlo jako novou jednotku délky Device Independent Pixel (DIP) a čistě vektorovou grafiku [5].
2. WF funguje tak, že si ke každému ovládacímu prvku uchovává jeho absolutní pozici vzhledem k formuláři, což může být u složitějších formulářů nedostačující. Proto se Microsoft inspiroval z webového vývoje a jazyka HTML zavedením vlastního jazyka pro definici grafického rozhraní nazvaného XAML, viz kapitola 3.2.4, který vychází z jazyka XML [5].

3. WF využívá pro vykreslování grafiky rozhraní Windows (GDI), které je pomalé a přináší mnoho omezení. WPF používá k vykreslování formulářů technologii Direct3D, což je rozhraní pro akcelerovanou grafiku. Díky tomu WPF aplikace běží rychleji a tolik nezatěžují procesor. WPF umožňuje programátorovi nastylovat komponentu přesně tak, jak potřebuje, může například do tlačítka přidat obrázek či rozbalovací menu nebo celé tlačítko může animovat [5].

Nedílnou součástí WPF je Dispatcher. Tato komponenta zajišťuje, že se úkoly vykonávají po sobě v rámci jednoho hlavního vlákna. Jedná se tedy o frontu úkolů. Obecně k uživatelskému prostředí v systému Windows zpravidla přistupujeme vždy z jednoho vlákna. Takové vlákno se nazývá hlavní vlákno UI a takový přístup tedy nazýváme jedno-vláknový. Díky tomuto přístupu se nemusíme starat o celou řadu synchronizačních problémů.

Pokud chceme, aby se v rámci uživatelského prostředí cokoliv stalo, přidáme úkol do Dispatcheru a on sám zajistí jeho spuštění v hlavním vlákně UI ve správný čas [18]. Máme tak jistotu, že se nebudou nikdy vykonávat dva úkoly týkající se uživatelského prostředí současně. Mezi tyto operace patří hlavně:

1. vykreslování,
2. události uživatelského vstupu,
3. změny vlastností.

Výhodou je, že o Dispatcheru programátor nemusí vůbec vědět. Je to vnitřní princip WPF. Hlavní nevýhodou tohoto přístupu je stav, kdy některá operace v Dispatcheru trvá příliš dlouhou dobu. V takovém případě dojde k „zamrznutí“ uživatelského rozhraní. Dispatcher bude totiž v takovém případě čekat na dokončení požadované operace a v tomto okamžiku před dokončením nemůže být provedena žádná další interakce s uživatelem. Je však možné tento stav obejít a operaci, která by mohla trvat delší dobu, spustit v novém vlákně. V takovém případě ale nemůžeme ovládat uživatelské rozhraní z jiného vlákna, protože by aplikace skončila chybou. V případě potřeby interakce s uživatelským rozhraním je tedy nutné tuto činnost opět přenechat hlavnímu vláknu [6; 20].

I když WPF nabízí velké množství komponent, občas je potřeba upravit jejich chování, nebo změnit jejich vzhled. Stejně tak tomu bylo v případě systému vyvíjeného v této práci. Bylo zapotřebí vytvořit vlastní menu tlačítka, informační boxy nebo dotazovací okna. Díky variabilitě komponent toho bylo poměrně lehce dosaženo, jak je vidět na obrázku

Chyba! Nenalezen zdroj odkazů.. Tento informační box se zobrazí v horní části okna, pokud v systému nastane situace, kdy by měl být uživatel informován o nějaké akci, ať už úspěšné, nebo i neúspěšné.

Obrázek 7 - Informační box

Pro zjednodušení používání byla vytvořena statická metoda `Show`, která automaticky vypočítá umístění informačního boxu. Vstupním parametrem je zde `MsqBoxType` určující, jestli se jedná o informaci, chybu nebo varování. Podle toho je okno barevně podbarveno pro rychlejší orientaci uživatele. Dalším parametrem je zde řetězec textu, který se v okně zobrazí.

```
public static void Show(MsgBoxType type, string text){
            var top = Application.Current.MainWindow.Top;
            var left = Application.Current.MainWindow.Left;
            var width = Application.Current.MainWindow.Width;

            var position = new Point(left + width - 400, top);

            Show(type, text, position);
}
public static void Show(MsgBoxType type, string text, Point position){
      switch (type) {
                case MsgBoxType.Error:
                    new MsgBoxInfo(text, Brushes.Red, position).Show();
                    break;
                case MsgBoxType.Warning:
                    new MsgBoxInfo(text, Brushes.Yellow,position).Show();
                    break;
                case MsgBoxType.Info:
                    new MsgBoxInfo(text, Brushes.Green, position).Show();
                    break;
      }
}
```

3.2.4 XAML

XAML je zkratka pro eXtensible Application Markup Language. Jedná se o variaci jazyka XML, kterou vytvořil Microsoft pro vytváření uživatelského rozhraní WPF nebo Silverlight aplikací. Často se však můžeme setkat s jeho alternativním způsobem využití, jako je definice konfigurace aplikace nebo popisu procesů. Tento jazyk slouží k usnadnění zápisu. Vše, co lze pomocí něj udělat, můžeme stejně zapsat i v jazycích C# nebo Visual Basic. Ve WPF tento jazyk slouží k sestavení objektového stromu elementů uživatelského prostředí, protože pracuje s objektovým modelem .Net – buď s komponenty, které jsou dodávané .Net frameworkem,

nebo jakoukoliv podstrčenou knihovnou, na kterou se ze XAML souboru odkážeme pomocí XML jmenných prostorů [7; 19].

Jmenné prostory hrají v jazyce XAML významnou roli. Definujeme skrze ně, jaké datové typy budeme v zápisu XAML používat.

```
<catel:Window x:Class="FitIS.Views.MainWindowView"
              xmlns:catel="http://schemas.catelproject.com"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
              xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
              xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
              xmlns:lib="clr-namespace:FitIS.Lib;assembly=FitIS.Lib"
              Title="Fit IS"
              WindowState="Maximized"
              AllowsTransparency="True"
              Background="{StaticResource CBlue}"
              WindowStartupLocation="CenterScreen"
              Visibility="{Binding WindowVisibility, Mode=TwoWay}">
<Grid>
</Grid>
</catel:Window>
```

Nejprve můžeme vidět otevření kořenového elementu `catel:Window`. Jedná se o okno z knihovny Catel viz kapitola 3.3.1. Následuje deklarace napojení XAML souboru se souborem obsahujícím kód, v tomto případě se jedná o soubor `FitIS.Views.MainWindowView.cs`. Následuje import názvů začínajícím vždy slovem `xmlns`, kde za dvojtečkou následuje volitelná zkratka knihovny. Následuje nastavení atributů okna, jako jsou titulek, výchozí stav okna a výchozí pozice okna. Na příkladu můžeme vidět i bindování atributu `Visibility`, který je napojen na atribut ViewModelu určující viditelnost či neviditelnost okna, nebo nastavení pozadí pomocí statické proměnné `CBlue`, deklarované v souboru starající se o grafické nastavení systému. V samotném obsahu okna je poté povolen jeden element typu `Grid`.

Každý element může mít libovolný počet vlastností, ale může si zvolit jen jednu vlastnost zvanou content (obsah). Tato vlastnost určuje, co vše můžeme zapsat dovnitř elementu.

Například následující zápisy vytvoří stejný popisek s textem: „Ahoj!“

```
<Label Content="Ahoj!"/>
```

```
<Label>Ahoj!</Label>
```

Naopak následující zápis není povolen právě proto, že element může obsahovat pouze jednou definovanou vlastnost content.

```
<Label Content="Ahoj!">Ahoj!</Label>
```

Díky této vlastnosti je také možné umístit dovnitř elementu například jiný element. V další ukázce je vidět, jak je do tlačítka přidán grid obsahující obrázek a popisek.

```
<Button>
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*"/>
            <ColumnDefinition Width="Auto"/>
        </Grid.ColumnDefinitions>
        <TextBlock Grid.Column="1" TextWrapping="Wrap"
TextAlignment="Center" VerticalAlignment="Center"/>
        <Image Grid.Column="0" />
    </Grid>
</Button>
```

Některé třídy dědící z třídy `Panel` mohou tímto způsobem vkládat objekty do sebe, a to proto, že mají jako content nastavenou vlastnost `Children`, která je typu `UIElementCollection` a umožňuje tak nést kolekci elementů uživatelského prostředí. Takto je v systému řešeno například dynamické menu.

```
<ItemsControl ItemsSource="{Binding MenuButtons}">
      <ItemsControl.ItemsPanel>
                <ItemsPanelTemplate>
                  <StackPanel Orientation="Vertical" IsItemsHost="True">
                </ItemsPanelTemplate>
      </ItemsControl.ItemsPanel>
      <ItemsControl.ItemTemplate>
                <DataTemplate>
                    <lib:UCMenuBtn/>
                </DataTemplate>
      </ItemsControl.ItemTemplate>
</ItemsControl>
```

V kódu můžeme vidět, že pro vytvoření dynamického menu byla zvolena komponenta `ItemsControl`, která může obsahovat více dílčích komponent. V tomto případě je jako nosič komponent zvolen `StackPanel`, kde jsou obsažena jednotlivá tlačítka.

3.2.5 Návrhový vzor Model View ViewModel

Model View ViewModel (MVVM) je návrhový vzor pro WPF aplikace. Poskytuje ucelené řešení, jak oddělit logiku aplikace od uživatelského rozhraní. Výsledná aplikace má pak méně kódu, je mnohem lépe testovatelná, vše je mnohem přehlednější a případné změny nejsou implementační noční můrou. MVVM odděluje data, stav aplikace a uživatelské rozhraní. Tento návrhový vzor umožňuje vývojářům a návrhářům uživatelských rozhraní mnohem jednodušší spolupráci při vývoji stejné části aplikace.

Ve většině desktopových aplikací probíhá na pozadí proces a okna slouží ke konzumaci informací pocházejících z tohoto procesu. Je žádoucí, aby byl proces na okně nezávislý. To umožňuje zavření okna bez jeho ovlivnění a jednoduché sdílení informací mezi více okny. Hlavní myšlenkou návrhového vzoru MVVM je vytvořit třídu, která si drží stav aplikace. Tuto třídu nazýváme ViewModel. Uživatelské rozhraní, které vykresluje ovládací prvky, se dotazuje ViewModelu na data a naopak, zadá-li uživatel do uživatelského rozhraní nějaké údaje, automaticky se promítnou do ViewModelu. WPF je pro toto použití uzpůsobeno, protože díky bindingu lze deklarativně napojit uživatelské rozhraní na ViewModel. Z toho vyplývá, že nejdůležitější třídou je právě ViewModel [8].

Výhody použití MVVM:

1. Pokud existuje implementační model, který zapouzdřuje stávající business logiku, může být obtížné nebo nebezpečné ho měnit. V tomto scénáři se ViewModel tváří jako adaptér pro třídy Modelu a umožňuje vývojáři vyvarovat se zásadních změn třídy Modelu.
2. Vývojáři mohou vytvořit Unit testy pro ViewModel a Model bez zobrazení View.
3. Uživatelské rozhraní může být zcela změněno bez zásahu do kódu za předpokladu, že zobrazení je zcela implementováno v XAML.
4. Návrháři a vývojáři mohou pracovat souběžně a nezávisle na jednotlivých komponentách, aniž by si navzájem zasahovali a ovlivňovali kód.

Při používání návrhového vzoru MVVM platí některé konvence, které by se měly dodržovat. Jednou z nich je pojmenovávání tříd v rámci projektu. Všechny třídy starající se o zobrazování informací by měly končit slovem View, obvykle se jedná o okna nebo formuláře. Programátor by také měl minimalizovat, nebo úplně vynechat psaní kódu do tříd pohledů, takovéto části pohledu se říká code-behind. Jedná se o třídy pohledů, ve kterých by se neměla nacházet žádná funkcionalita. Všechna tato funkcionalita by měla být přesunuta do ViewModelů

1. Model

Popisuje data, se kterými aplikace pracuje. Model nesmí o stavu ovládacích prvků nic vědět. Jedná se o jednoduché třídy s atributy. V těchto třídách by se neměl nacházet žádný přebytečný kód řešící business logiku aplikace.

2. View

Reprezentuje uživatelské rozhraní v jazyce XAML. Může se jednat o okno aplikace, stránku nebo ovládací prvek. View je zodpovědné za definici struktury, rozložení a vzhled, který se zobrazí uživateli na obrazovce. Obvykle se za pohledy považují `Windows`, `Page` nebo `UserControl`. Nicméně View může být také reprezentováno pomocí data template, které specifikuje vizuální element reprezentující zobrazený objekt. Data template jako pohled nemají žádný code-behind a jsou bindována na specifický typ ViewModelu.

Existuje několik možností pro spuštění kódu v ViewModelu v reakci na interakci s View, jako jsou kliknutí na tlačítko nebo změna vybrané položky například v rozbalovacím seznamu. Pokud ovládací prvek podporuje Commands, pak takový Command může být navázán na vlastnost ICommand v ViewModelu. Při vyvolání příkazu ve View se zároveň vyvolá stejná událost ve ViewModelu.

Jako programátor se jistě dostanete do situace, kdy navázané vlastnosti mezi View a ViewModelem nebudou stejného datového typu. V takovém případě se dají použít takzvané konvertory. Konvertor je třída sloužící jako převodník hodnot podle předem stanovené definice. Tyto malé třídy, které implementují rozhraní `IValueConverter,` se budou chovat jako prostředníci a přeloží hodnotu mezi zdrojem a cílem. Každá taková třída musí obsahovat dvě metody, a to metodu `Convert` pro dopředný převod a `ConvertBack` pro zpětný převod hodnot. Tyto konvertory jsou často implementovány pro tyto účely:

1. Máte číselnou hodnotu, ale chcete zobrazit nulové hodnoty jedním způsobem a pozitivní čísla jiným způsobem.
2. Chcete zkontrolovat zaškrtávací tlačítko na základě hodnoty, ale hodnota je řetězec jako „true“ nebo „false“ namísto logické hodnoty.
3. Velikost souboru je v bytech, ale chcete jej zobrazit jako bajty, kilobajty, megabajty nebo gigabajty na základě toho, jak velký je.

4. Můžete například zkontrolovat zaškrtávací políčko založené na logické hodnotě, ale chcete, aby byl obrácen, aby bylo zaškrtnuto políčko zaškrtávacího tlačítka, pokud je hodnota nepravdivá a není zaškrtnuta, pokud je hodnota „true“.
5. Můžete použít konvertor pro generování obrazu pro zdroj obrázku, na základě hodnoty jako zelený znak pro „true“ nebo červený znak pro „false“.
6. Chcete zobrazit, či skrýt celou komponentu na základě booleovského výrazu. Ukázka implementace třídy takovéhoto konvertoru:

```
public class BoolToVisibleOrHidden : IValueConverter {
        public BoolToVisibleOrHidden() { }
        public bool Collapse { get; set; }
        public object Convert(object value, Type targetType, object
parameter, System.Globalization.CultureInfo culture) {
            bool bValue = (bool)value;
            if (bValue)
                return Visibility.Visible;
            else {
                if (Collapse)
                    return Visibility.Collapsed;
                else
                    return Visibility.Hidden;
            }
        }
        public object ConvertBack(object value, Type targetType, object
parameter, System.Globalization.CultureInfo culture) {
            Visibility visibility = (Visibility)value;
            if (visibility == Visibility.Visible)
                return true;
            else
                return false;
        }
    }
```

3. ViewModel

Spojuje Model a View a drží stav aplikace. Ovládací prvky jsou pomocí bindingu spojeny s ViewModelem a čerpají z něj svůj obsah. Aby se jeho vlastnosti propagovaly do View, musí implementovat rozhraní `INotifyPropertyChanged` nebo v případě kolekcí `INotifyCollectionChanged`. Obsahuje obsluhu volání business objektů systému. ViewModel je testovatelný nezávisle na View nebo Modelu.

3.2.6 Entity Framework

Entity Framework (EF) je open-source ORM[4] Framework pro .Net aplikace podporované společností Microsoft. Umožňuje vývojářům pracovat s daty pomocí objektů třídy specifických pro danou doménu bez zaměření na základní tabulky a sloupce, kde jsou tato data uložena. S EF mohou vývojáři pracovat na vyšší úrovni abstrakce, když se zabývají daty a mohou vytvářet a udržovat datově orientované aplikace s menším kódem ve srovnání s tradičními aplikacemi.

Funkce EF [9; 22]:

1. Multiplatformost – EF Core je multiplatformní, a může tedy běžet jak na Windows, Linux tak Mac operačních systémech.
2. Modelování – EF vytváří Entity Data Model (EDM) založený na entitách Plain Old CLR Object (POCO).
3. Dotazování – EF umožňuje používat LINQ dotazy k získávání dat z podkladové databáze. Poskytovatel databáze převede tyto LINQ dotazy do dotazovacího jazyka specifického pro použitou databázi. EF také umožňuje provádět surové dotazy SQL přímo do databáze.
4. Sledování změn – EF sleduje změny, ke kterým došlo u instancí entit, které je třeba odeslat do databáze. Pokud se tedy změnila struktura našich entit, EF nás na to upozorní.
5. Ukládání – EF provede příkazy insert, update a delete do databáze na základě změn, ke kterým došlo při volání metody SaveChanges() nebo v případě asynchronního volání SaveChangesAsync().
6. Souběžnost – EF používá ve výchozím nastavení ochranu proti přepsání dat jinými uživateli.
7. Transakce – EF provádí automatickou správu transakcí při dotazování nebo ukládání dat. Poskytuje také možnosti přizpůsobení správy transakcí.
8. Ukládání do mezipaměti – EF zahrnuje první úroveň ukládání do mezipaměti. Opakované dotazování tedy vrátí data z mezipaměti namísto časově náročnějšího dotazu do databáze.
9. Vestavěné konvence – EF se řídí konvencemi a obsahuje sadu výchozích pravidel, která automaticky konfigurují model EF.

[4] Object Relation Mapper – zajišťuje automatickou konverzi dat mezi relační databází a objektově orientovaným programovacím jazykem

10. Konfigurace – EF umožňuje konfigurovat model pomocí anotace atributů nebo Fluent API pro potlačení výchozích konvencí.
11. Migrace – EF poskytuje sadu migračních příkazů, které lze provádět v konzoli správce balíčku NuGet nebo v rozhraní příkazového řádku.

Entity framework poskytuje tři možné přístupy pro vývoj. Prvním přístupem je Database-First, při kterém EF vygeneruje kontext i entity pomocí průvodce EDM integrovaného ve Visual Studiu, nebo vykonáním EF příkazů. Dalším přístupem je Code-First, který se používá, pokud ještě nemáme navrženou databázi pro naši aplikaci. Programátor nejprve vytvoří své entity a třídu kontextu a poté z těchto tříd vytvoří databázi pomocí migrace. Tento přístup je vhodný zejména tehdy, nemá-li programátor dostatečné zkušenosti s návrhem databáze. Posledním přístupem je takzvaný Model-First. V tomto přístupu nejprve vytvoříte entity, vztahy a hierarchie dědičnosti přímo ve vizuálním návrháři integrovaném ve Visual Studiu a poté vytvoříte entity, třídu kontextu a skript databáze z vizuálního modelu.

Pokud si programátor zvolí jako přístup vývoje Code-First pravděpodobně se nevyhne změnám datového modelu během vývoje. Aby tyto změny struktury byly snadno převedeny i do databáze a neovlivnily stávající data, používá EF takzvané migrace. Jedná se o přírůstkové aktualizace schématu databáze [9; 23].

Pro správné použití migrací je potřeba nejprve nad naším databázovým kontextem migrace povolit pomocí vyvolání příkazu `enable-migrations` v příkazovém řádku Package Manager Console. Jakmile se příkaz úspěšně provede, vytvoří se složka s třídou konfigurace, která je poděděna ze třídy `DbMigrationConfiguration`. V této třídě je možné ovlivňovat nastavení migrace. Zároveň se v databázi vytvoří tabulka uchovávající informace o jednotlivých migracích. Díky tomu je také možné vracet provedené změny zpět do stavu před kteroukoliv migrací. Pokud budeme měnit jakoukoliv datovou entitu a neprovedeme migraci, EF při spuštění aplikace upozorní na změnu datového modelu. Proto je po každé takovéto změně nutné provést příkaz `add-migration Název_migrace`, tím se vytvoří nová migrace databáze a poté už pouze promítneme danou migraci do databáze příkazem `update-database`. EF automaticky vygeneruje celý potřebný kód, který si můžeme dále upravit. Ukázka takového kódu je zobrazena níže.

```
public partial class Newsletter : DbMigration {
        public override void Up() {
            CreateTable(
                "dbo.Newsletters",
                c => new
                    {
                        NewsLetterId = c.Int(nullable: false, identity:
true),
                        SendDate = c.DateTime(nullable: false),
                        Canal = c.Int(nullable: false),
                        Subject = c.String(),
                        IsHtml = c.Boolean(nullable: false),
                        Text = c.String(),
                        Sended = c.Boolean(nullable: false),
                    })
                .PrimaryKey(t => t.NewsLetterId);
            AddColumn("dbo.Customers", "Newsletter_NewsLetterId", c =>
c.Int());
            CreateIndex("dbo.Customers", "Newsletter_NewsLetterId");
            AddForeignKey("dbo.Customers", "Newsletter_NewsLetterId",
"dbo.Newsletters", "NewsLetterId");
        }
        public override void Down() {
            DropForeignKey("dbo.Customers", "Newsletter_NewsLetterId",
"dbo.Newsletters");
            DropIndex("dbo.Customers", new[] { "Newsletter_NewsLetterId"
});
            DropColumn("dbo.Customers", "Newsletter_NewsLetterId");
            DropTable("dbo.Newsletters");
        }
    }
```

Jsou zde vidět dvě funkce, a to `Up` a `Down`. Nejprve se provede funkce `Down`, která obsahuje kód pro smazání dat a struktury databáze. Poté se provede funkce `UP`, která obsahuje kód pro vytvoření databáze. V ukázce můžeme vidět třídu, kterou nám EF vygeneroval. Název třídy je stejný jako první parametr příkazu `add-migration`. Nejprve se tedy provede metoda `Down`, která v tomto případě zruší cizí klíč, index a sloupec z tabulky Customers. Protože v této migraci bude zapotřebí velkých změn v tabulce Newsletters, je potřeba původní tabulku také zrušit. Následuje provedení metody `Up,` ve které se nejprve vytvoří nová tabulka Newsletters s potřebnými sloupečky a nastaví se primární klíč. Poté se přidá sloupec a cizí klíč do tabulky Customers a vytvoří se nový index nad tímto sloupcem. Všechen tento kód byl vygenerován EF a programátor nemusí mít žádné hlubší databázové znalosti.

Vzhledem k tomu, že jazyk C# používá dědičnost a EF vytváří tabulky pro každou konkrétní entitu, je nutné zohlednit tyto vztahy i při návrhu jednotlivých entit. Pro tyto potřeby byly vyvinuty 3 možné přístupy:

1. Tabulka na hierarchii (TPH): Jedna tabulka pro celou třídu hierarchie. Tabulka obsahuje sloupec diskriminátoru, který rozlišuje mezi třídami. Toto je výchozí strategie mapování dědičnosti v rámci Entity Framework.
2. Tabulka na typ (TPT): Tato tabulka je samostatnou tabulkou pro každou třídu domén.
3. Tabulka na konkrétní třídu (TPC): Pokud zdědíte abstraktní třídu ve více konkrétních třídách, pak budou vlastnosti abstraktní třídy součástí každé tabulky konkrétní třídy.

V rámci implementace systému fitness centra bylo potřeba rozlišit mezi vstupovou a časovou permanentkou. Pro tyto potřeby byla vytvořena abstraktní třída `Ticket` a potomci této třídy `EntryTicket` a `SessionTicket`. V rámci vytvoření databázových tabulek byl zvolen přístup TPH. Na obrázku Obrázek 9 – Databázový model můžete vidět, vytvořenou databázovou tabulku Tickets obsahující sloupec Discriminator určující, o jaký typ permanentky se jedná. EF poté vytvoří instanci právě té třídy, která je uvedená v hodnotě tohoto sloupce.

Neodmyslitelnou součástí EF je možnost Lazy loadingu. Lazy loading zajišťuje načítání relačních vazeb až v době, kdy k nim přistoupíme. EF však umožňuje tuto funkci pomocí nastavení vypnout. Pokud chceme, aby se k objektu načítaly i objekty odkazované, stačí mít lazy loading zapnutý a do definice atributu odkazovaného objektu přidat klíčové slovo virtual.

Pro dotazování se do databáze je používaný dotazovací jazyk LINQ což je poměrně sofistikovaná a rozsáhlá technologie. Její název pochází z anglického Language INtegrated Query. LINQ je součástí C# od verze 3.0 a .NET frameworku 3.5. Od novějších verzí běží dokonce na více vláknech, což zvyšuje efektivitu této technologie. LINQ je velmi podobný jazyku SQL, je to tedy jazyk deklarativní. Programu sdělíme, co hledáme, a už nás moc nezajímá, jakým způsobem pro nás data opravdu vyhledá. Výhodou integrace LINQ do C# je syntaktická kontrola dotazů při překladu programu [10]. Dotaz pro vytažení všech zákazníků systému, kteří se jmenují Karel, seřazených podle příjmení by poté mohl být v C# napsán třeba takto:

```
var customers = _context
      .Customers
      .Where(cust=>cust.Firstname = „Karel“)
      .OrderBy(cust=>cust.Lastname)
```

Tento příkaz jde však zapsat i takzvaným „SQL like“ způsobem, který může být lépe čitelný pro databázové specialisty.

```
var customers = from cust in _context.Customers
      where cust.Firstname = „Karel“
      orderby cust.Lastname
      select cust;
```

3.3 Použité externí knihovny

V rámci aplikace bylo použito několik knihoven třetích stran usnadňujících různé části programu. Tyto knihovny jsou dostupné z balíčkovacího manažeru NuGet.

3.3.1 Catel

Catel je platforma pro vývoj aplikací se zaměřením na MVVM. Cílem bylo poskytnout kompletní sadu modulárních funkcí pro aplikace napsané v technologii .Net od klienta k serveru.

Tato knihovna se skládá ze dvou menších samostatných knihoven Catel.Core, tedy samotného jádra, a Catel.MVVM pro podporu funkcí spojených s návrhovým vzorem MVVM. Knihovna jádra podporuje například validaci argumentů, ukládání do paměti, IoC[5], posílání zpráv, reflexi a serializaci. Knihovna pro podporu návrhového vzoru MVVM poskytuje funkcionality pro auditování, vlastní kolekce, příkazy, konvertory a automatickou validaci. Dále také poskytuje třídy služeb, jako jsou například [11]:

1. LocationService – služba umožňující vývojářům využívat GPS,
2. MessageService – služba pro zobrazení informačních oken,
3. NavigationService – služba pro přecházení mezi jednotlivými stránkami aplikace,
4. OpenFileService – služba pro práci se soubory,
5. PleaseWaitService – služba pro zobrazení „Prosím počkejte“ okna,
6. ProcessService – služba pro práci s procesy systému,
7. SaveFileService – služba pro ukládání souborů,
8. SchedulerService – služba pro naplánované úlohy,
9. SelectDirectoryService – služba pro práci s adresáři,
10. UIVisualizerService – služba pro práci s modálními okny,

[5] IoC neboli Inversion of Control je technika pro vkládání závislostí mezi jednotlivými komponentami programu tak, aby jedna komponenta mohla používat druhou, aniž by na ni měla v době sestavování programu referenci

11. VibrateService – služba pro spuštění či ukončení vibrací zařízení,
12. ViewExportService – služba pro export pohledů.

Při správném používání návrhového vzoru MVVM by se měly dodržovat konvence pojmenování jednotlivých tříd. Všechny názvy pohledů by měly končit slovem View, datové třídy slovem Model a třídy obsluhující logiku aplikace slovem ViewModel. Díky této konvenci je poté možné pomocí Catel frameworku automaticky namapovat datové kontexty. V případě, že se názvy tříd liší, je možné vytvořit vlastní mapování.

```
var vMLocator = ServiceLocator.Default.ResolveType<IViewModelLocator>();
vMLocator.NamingConventions.Add("NazevAsembly.ViewModels.[VW]ViewModel");
```

Pokud se u názvů tříd není na co napojit, je zde ještě možnost manuálního propojení kontextu.

```
var vMLocator = ServiceLocator.Default.ResolveType<IViewModelLocator>();
vMLocator.Register(typeof(MyView), typeof(MyViewModel));
```

3.3.2 Live charts

Jedná se o knihovnu určenou pro programovací jazyk C#, která na platformách UWP, WinForm nebo WPF pomůže s vykreslením nejrůznějších grafů. Tato knihovna byla navržena tak, aby se dala snadno používat a aby výsledný graf byl automaticky aktualizován a animován. Programátorovi nabízí velice variabilní konfiguraci, díky které může výsledný graf nabrat téměř jakoukoliv podobu [21].

3.3.3 Log4Net

Knihovna Log4Net od Apache je nástroj, který umožňuje programové výstupy do různých výstupních cílů. Tato knihovna je používána k logování jakýchkoliv událostí v aplikacích. Log4Net umožňuje měnit protokolování za běhu aplikace bez nutnosti úprav binárních souborů. Pomocí konfiguračního souboru si uživatel dokáže nastavit, do jakého výstupu či výstupů chce logování směřovat, jejich formát a případně které typy chyb budou logovány. Tato knihovna je navržena tak, aby maximalizovala rychlost a flexibilitu logování.

Podporované cíle výstupu [12]:

1. databáze,
2. barevný výstup do terminálu,
3. ASP kontext,
4. aplikační terminál,

5. windows prohlížeč událostí,
6. soubor,
7. paměť,
8. služba Windows Messenger,
9. vzdálený syslog pomocí UDP přenosu,
10. e-mail.

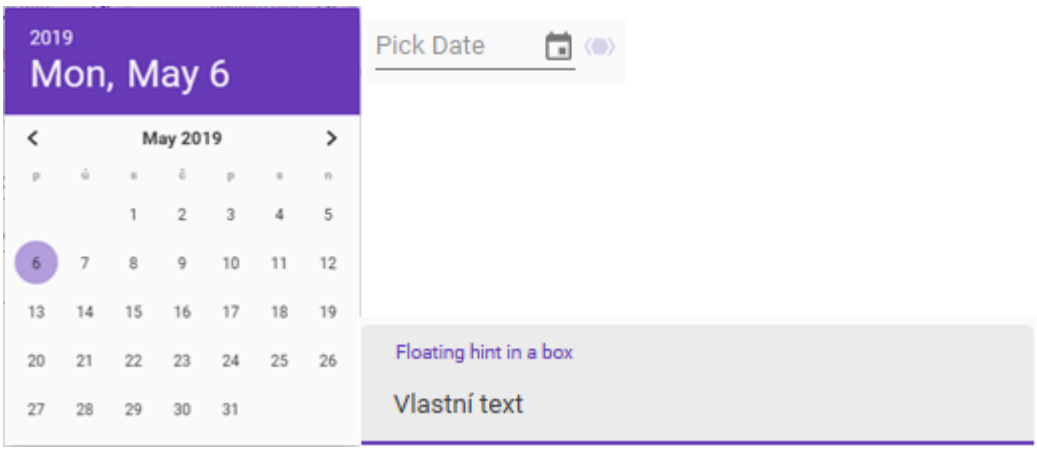
3.3.4 Material design toolkit

Tato knihovna slouží k vytváření graficky přívětivých aplikací v prostředí jazyka XAML viz kapitola 3.2.4. Jedná se o jednu z nejčastěji používaných GUI knihoven pro WPF. Umožňuje programátorovi navrhovat esteticky přívětivé aplikace bez větší znalosti grafického designu.

Mezi podporované funkce patří [13]:

1. styly a variace pro většinu standardních ovládacích prvků WPF,
2. mnoho dalších doplňkových ovládacích prvků pro podporu estetiky,
3. snadná konfigurace palety barevného designu materiálu při návrhu i v době chodu,
4. přechody API pro snadné sestavení GUI animací,
5. funguje samostatně a je také kompatibilní s dalšími populárními knihovnami WPF, MahApps a Dragablz.

Použití prvků této knihovny je velice jednoduché a po grafické stránce jsou mnohem přívětivější, než základní komponenty WPF. Na obrázku **Chyba! Nenalezen zdroj odkazů.** jsou zobrazeny komponenty DateTimePicker a TextBox, které poskytuje tato knihovna.



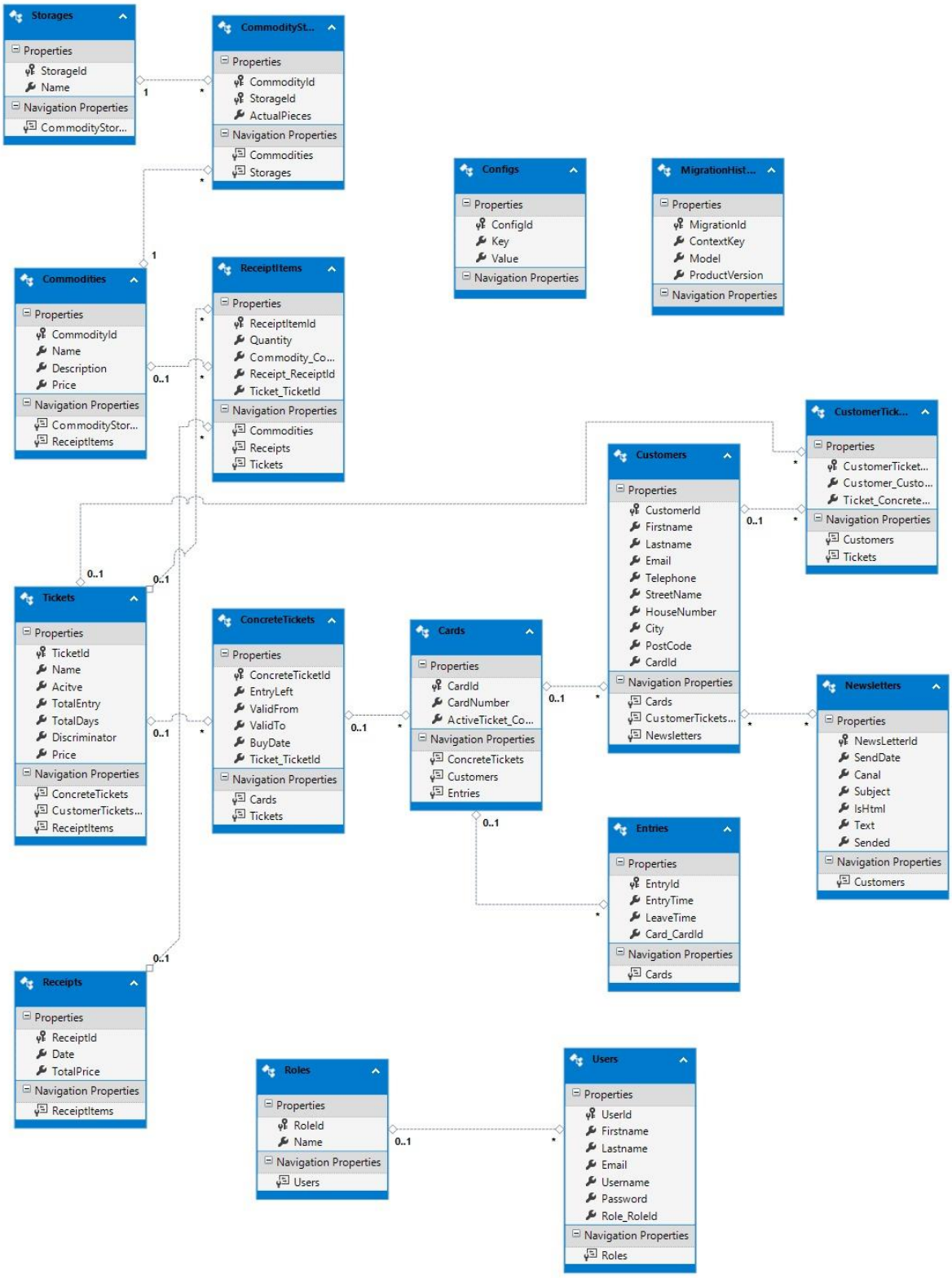
Obrázek 8 – Material design ukázka

Pro vložení těchto prvků do formuláře je zapotřebí nastavit atribut `Style` na požadovaný styl použité komponenty, jak vidíme na následujícím kódu. Programátor stále může ovlivňovat grafický vzhled těchto komponent přepsáním požadovaných atributů.

```
<DatePicker
  Width="100"
  materialDesign:HintAssist.Hint="Zvolte datum"
  Style="{StaticResource MaterialDesignFloatingHintDatePicker}" />
<TextBox
    Style="{StaticResource MaterialDesignFilledTextFieldTextBox}"
    VerticalAlignment="Top"
    AcceptsReturn="True"
    TextWrapping="Wrap"
    MaxWidth="400"
    materialDesign:HintAssist.Hint="Text zobrazený v komponentě"
    IsEnabled="{Binding Path=IsChecked,
ElementName=MaterialDesignFilledTextFieldTextBoxEnabledComboBox}" />
```

3.4 Databáze

Databázový model se skládá z 15 tabulek a je navržen dle platných předpisů pro návrh databáze. Jelikož byl pro implementaci tohoto systému využit Entity framework, tak pro samotný databázový model byl využit Code-First přístup. Díky tomu byly nejprve naprogramovány entity přímo v C# projektu a poté byla automaticky vygenerována struktura databáze. Pro znázornění struktury byl však pomocí nástroje integrovaného v prostředí Visual Studio vytvořen diagram databázových tabulek viz Obrázek 9 – Databázový model.



Obrázek 9 – Databázový model

3.4.1 Popis jednotlivých tabulek

Configs – Tato tabulka obsahuje konfigurační nastavení aplikace. Položka Key určuje klíč konfigurace a v aplikaci je reprezentována výčtovým typem. Hodnota tohoto klíče je potom šifrována, aby bylo zabráněno možné manipulaci s nastavením systému.

Storages – Tabulka uchovávající informace o jednotlivých skladech v systému. Tyto sklady jsou definovány jménem.

CommodityStorages – V této tabulce se nachází informace o tom, kolik kusů zboží je dostupno v jednotlivých skladech. V tabulce se nacházejí dva cizí klíče, a to do tabulky Commodity pro identifikaci typu zboží a do tabulky Storage pro určení, kterému skladu zboží náleží.

Commodities – Tabulka uchovávající informace o prodávaném zboží. Každé zboží je definováno svým názvem, popiskem a cenou za kus.

Receipts – Tabulka ukládající informace o vydaných účtenkách. Nachází se zde atributy data a času, kdy byla účtenka vydána, a celková cena účtenky.

ReceiptItems – Tabulka uchovávající informace o jedné položce účtenky. V tabulce se nachází cizí klíče do tabulky Receipt pro určení, které účtence položka náleží, a množství prodaných položek. Protože položka na účtence může být buď zboží, nebo permanentka, nachází se zde další dva nulové cizí klíče do tabulky Commodities pro určení prodávaného zboží a do tabulky Tickets pro určení prodávané permanentky.

Tickets – V této tabulce se uchovávají informace o permanentkách používaných v systému. U každé permanentky evidujeme jméno, cenu a indikátor aktivní, který určuje, zda je permanentka stále použitelná. Jelikož v systému evidujeme dva typy permanentek, a to časové nebo vstupové, nachází se zde atribut vygenerovaný EF nazvaný Discriminator, určující o který typ permanentky se jedná. Dále se zde v závislosti na typu permanentky nachází atribut určující maximální počet vstupů v případě vstupových permanentek a atribut určující počet dní v případě vstupových permanentek.

ConcreteTickets – V této tabulce se nachází informace o konkrétních permanentkách uživatelů. Nachází se zde informace o datu koupě permanentky, cizí klíč do tabulky Tickets určující, o jaký typ permamenntky se jedná, a na to navazující atributy počtu zbylých vstupů v případě vstupové permanentky, nebo datum platnosti v případě vstupové permanentky.

Customers – Tabulka uchovávající informace o zákaznících. V této tabulce se nachází atributy jako křestní jméno, příjmení, kontaktní informace jako e-mail, telefon, poštovní směrovací číslo, město, ulice a číslo domu. Dále se zde nachází cizí klíč do tabulky Cards přiřazující každému zákazníkovi jednu kartičku pro vstup.

Cards – Tabulka uchovávající informace o konkrétních kartičkách uživatelů. Každá kartička má jedinečný identifikátor a dále cizí klíč do tabulky ConcreteTickets určující, jaká je aktivní permanentka k dané kartě.

CustomerTicketsHistories – V této tabulce se nachází historické informace o zakoupených permanentkách zákazníkem. Z toho důvodu tu jsou cizí klíče do tabulky Tickets určující, jakou permanentku zákazník zakoupil, a do tabulky Customers určující jaký zákazník ji zakoupil.

Entries – Tabulka uchovávající informace o vstupu zákazníka do fitness centra. V této tabulce se nachází atributy EntryTime a LeaveTime určující datum a čas příchodu či odchodu zákazníka. Dále je zde cizí klíč do tabulky Cards určující. na kterou kartičku byl daný vstup či odchod vykonán.

Users – Tabulka uchovávající informace o uživatelích systému. Každý uživatel je definován jedinečným přihlašovacím jménem a heslem. Heslo je v databázi zašifrováno jednosměrným šifrovacím algoritmem. Dále se ke každému uživateli uchovávají informace jako křestní jméno, příjmení a případně e-mail pro kontaktování uživatele. Je zde také cizí klíč do tabulky Roles, stanovující každému uživateli jedinečnou roli v systému.

Roles – V této tabulce se uchovávají informace o rolích v systému. Každá role je jednoznačně identifikována pomocí jména. Podrobný popis rolí a jaké mají v systému práva, jsou uvedeny v tabulce Tabulka 4 – Povolené moduly podle role uživatele.

Newsletters – Tabulka uchovávající informace o zpravodajích. V této tabulce se nachází atribut čas a datum, kdy bude, nebo byl daný zpravodaj odeslaný, zda byl odeslán nebo zda nastala chyba, o jaký typ komunikace se jedná (SMS, e-mail), předmět a text zprávy. Dále v případě, že se jedná o e-mail, může uživatel zvolit, aby byla zpráva ve formátu HTML a v takovém případě může mít e-mail i grafické prvky.

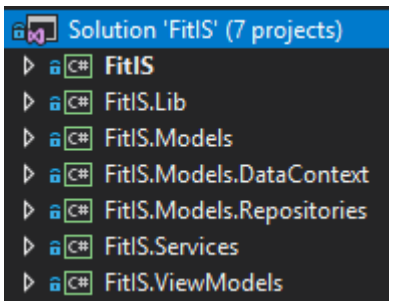
NewsletterCustomers – Tato tabulka byla automaticky vygenerována EF a nachází se zde informace o tom, kterému zákazníkovi má být daný zpravodaj zaslán. Jsou zde tedy cizí klíče do tabulky Customers a Newsletters.

MigrationHistory – Tato tabulka byla také automaticky vygenerována EF a nachází se zde informace o provedených migracích databáze viz kapitola 3.2.6. Jsou zde atributy jako název migrace, identifikátor kontextu databáze, verze EF a také celý databázový model v binární podobě.

3.5 Struktura aplikace a popis jednotlivých balíčků

Architektura aplikací je specifikována na základě obchodních a funkčních požadavků. To zahrnuje definování interakce mezi aplikačními balíky, databázemi a middlewarovými systémy z hlediska funkčního pokrytí. To pomáhá identifikovat problémy s integrací nebo mezery ve funkčním pokrytí. Pak může být vypracován plán migrace pro systémy, které jsou na konci životního cyklu softwaru nebo které mají vlastní technologická rizika.

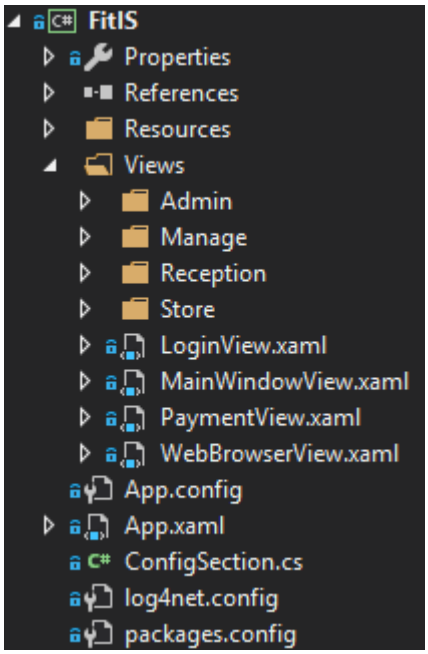
Při návrhu architektury projektu byl brán zřetel na to, aby struktura aplikace vyhovovala standardům MVVM a byla do budoucna lehce spravovatelná a rozšiřitelná. Výsledná architektura je znázorněna na obrázku Obrázek 10 – Struktura projektu a v dalších kapitolách budou popsány jednotlivé knihovny a jejich funkce.



Obrázek 10 – Struktura projektu

V projektu je oddělena databázová vrstva, s business logikou a prezentační vrstvou. Díky tomu je do budoucna mnohem jednodušší vytvořit například mobilní verzi systému bez zásahu do struktury dat nebo business logiky systému. Také při změně v kterékoliv části systému se na cílovou stanici, kde systém běží, nemusí v případě oprav nahrávat celý systém znovu, ale mohou se nahrát pouze zkompilované knihovny. Rozdělení těchto vrstev je ilustrováno v – UML diagram balíčků, zobrazující strukturu aplikace a interakce mezi jednotlivými knihovnami [16; 17].

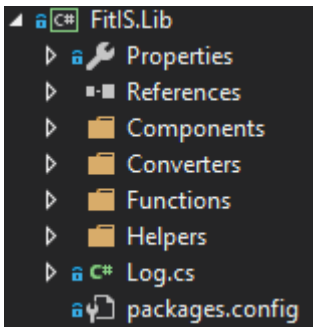
3.5.1 FitIS – hlavní spouštěcí program



Obrázek 11 – Struktura FitIS

Středobodem systému je knihovna FitIS, ve které se také nachází spouštěcí třída celého systému. Dále se zde nachází konfigurační soubory databáze a logovacího mechanizmu. Velice důležitá je zde třída `ConfigSection` ve které se nachází konfigurace všech repositářů, služeb a propojení View s ViewModely. Dále se zde nachází zdroje jako například obrázky použité v aplikaci. Nejdůležitější adresář je zde adresář View, ve kterém jsou obsaženy jednotlivé vizuální komponenty systému. V tomto adresáři jsou rozděleny podle toho, které části systému se týkají, tudíž se zde nachází adresář Admin, Manage, Reception, Store a poté sdílené pohledy například pro platbu v systému, hlavní okno aplikace, přihlašovací okno nebo integrovaný webový prohlížeč.

3.5.2 Knihovna FitIS.Lib



Obrázek 12 – Struktura Lib

V projektu se nachází knihovna FitIS.Lib, která obsahuje pomocné komponenty přístupné ve všech ostatních knihovnách systému. V této knihovně se nachází několik podsložek.

Složka Components obsahuje vlastní implementace vizuálních komponent využitých v systému. Mezi tyto komponenty patří dotazovací okno, informativní okénko, které uživateli zobrazuje informace o provedených akcích, případně chybách nebo upozorněních a také vlastní implementace tlačítek použitých v menu.

Složka Converters obsahuje jednotlivé implementace takzvaných konvertorů, blíže popsaných v kapitole 3.2.5.

Dále se zde nachází složka Functions, ve které jsou umístěny statické třídy pro volání komponent ze složky Components. Nalezneme zde také třídu MsgPipe, která slouží pro přeposílání zpráv mezi třídami. Tato funkcionalita využívá frontu zpráv systému. V jedné třídě zaregistrujeme odchyt této zprávy a z jiné třídy tuto zprávu můžeme odeslat. V systému je předpřipraven objekt obsahující informace o zprávě. V zásadě jsou v systému rozlišeny čtyři druhy zpráv:

1. None – zpráva obsahuje prostý text,
2. ChangeView – zpráva obsahuje informaci o změně pohledu,
3. ChangeVisibility – zpráva obsahuje informaci o změně viditelnosti pohledu,
4. RefreshViewModel – po přijetí zprávy tohoto typu provede ViewModel přenačtení všech dat.

Každý takový systém by měl logovat akce uživatelů, případně chyby, které v systému nastaly. Proto se v této knihovně nachází statická třída Log, která se stará o logování akcí systému. Toto logování zaznamenává případné chyby do textového souboru. Jsou zde zaznamenána také přihlášení uživatelů a zásadní operace s databází, jako jsou přidání, odebrání nebo úprava databázového záznamu.

Další důležitou částí této knihovny je obsah složky Helpers, která obsahuje pomocné funkce systému jako například hashovací a dehashovací funkce pro nastavení a přihlášení do systému. Pro tuto funkcionalitu byl použit algoritmus Rijndael. Dále se zde nachází pomocná třída pro validaci HTML, která je použita při vytváření zpravodaje. Dalším souborem je třída starající se o správu obrázků použitých v aplikaci. Pro každý obrázek je vytvořen jedinečný výčtový typ, jenž je použit při volání metody pro navrácení obrázku. Je tak zamezeno použití obrázků, které

nebyly dodány spolu se systémem. V následující ukázce je znázorněno použití této třídy při vytváření menu tlačítek s obrázky.

```
new UCMenuBtn(allBtn.Select(x => x.Name).ToList(), "Odhlásit", "odhlasit",
ResourceHelper.GetImageUrl(Img.Logout), false);
```

Jedná se o konstruktor tlačítka menu, kde jako vstupní parametry jsou:

1. seznam rolí, pro něž se tlačítko zobrazí,
2. text tlačítka,
3. parametr, s jehož pomocí je zavolána potřebná funkcionalita po stisku tlačítka,
4. fyzická cesta k obrázku,
5. indikátor, zda se na tlačítku zobrazí i text.

3.5.3 Knihovna FitIS.Models

V této knihovně se nachází pouze definice entit použitých v systému. Tyto entity odpovídají tabulkám v databázi a jsou mapovány pomocí ORM Entity Frameworku. Tyto entity jsou poté využívány napříč celým systémem. Jedná se o POCO třídy neboli Plain Old C# Object, obsahují tudíž pouze definici atributů a jejich datových tipů.

Další knihovnou spadající pod tuto větev je knihovna FitIS.Models.DataContext. Zde se nachází kontext starající se o spojení s databází a obsahující definici kolekcí, samotná lokální databáze pro případy testování a složka, do níž se ukládají migrace databáze viz kapitola 3.2.6.

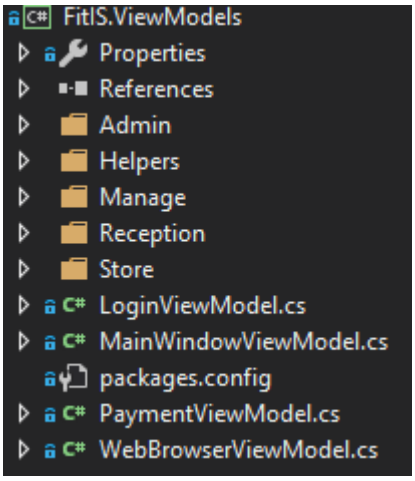
Poslední knihovna, která spadá pod tuto větev, je knihovna FitIS.Models.Repositories. V této knihovně se nachází definice rozhraní, která jsou nadstavbou pro práci s jednotlivými kolekcemi entit. Tato rozhraní jsou poté použita v knihovně služeb kapitola 3.5.4. Díky tomu je kompletně odstíněna databázová vrstva. Nejdůležitější rozhraní je IRepository<T>. Toto rozhraní implementují ostatní rozhraní pro práci s jednotlivými entitami. Jedná se o generické rozhraní, které poskytuje základní funkcionality pro práci s repozitáři, jako jsou například zpřístupnění všech entit daného typu, vyhledání jedné entity dle klíče, přidání nové entity, editace a mazání entit.

3.5.4 Knihovna FitIS.Services

Pro separaci výkonné části systému byla vytvořena knihovna FitIS.Services, která obsahuje rozhraní a implementaci těch částí systému, které se starají o business logiku aplikace. Třídy v této knihovně jsou rozděleny do logických celků tak, aby jejich případné použití bylo co

nejjednodušší a staralo se o ucelený celek funkcionalit. Nachází se zde tedy třídy pracující převážně s repozitáři, ale také služba starající se o vytvoření menu, autentizaci uživatelů v systému, rozesílání e-mailů a SMS zpráv, správu naplánovaných úloh a práci s konfigurací aplikace.

3.5.5 Knihovna FitIS.ViewModels



Obrázek 13 – Struktura bussines logiky

Poslední důležitou knihovnou je knihovna FitIS.ViewModels, ve které se nachází ViewModely, které zpracovávají a poskytují data prezenční vrstvě. Pro rychlejší a přehlednější práci s touto knihovnou je struktura totožná s knihovnou z kapitoly 3.5.1. Nalézají se zde tedy složky pro funkcionality přizpůsobené jednotlivým rolím v systému.

Jelikož ne vždy chceme zobrazit stejnou strukturu dat, jaká je v databázi, mohou se zde nacházet ještě podsložky nazvané Models obsahující konkrétní implementace použité pro specifický ViewModel. Tyto třídy jsou poté mapovány pomocí mapovacích funkcí obsažených ve složce Helpers, která obsahuje třídu Mapper. Tato třída obsahuje statické metody starající se o mapování mezi dvěma třídami. Ukázka takového mapování je přiložena v příloze

Příloha B – Mapování tříd .

Ve složce Helper se dále nachází třídy Generators pro generování náhodných sekvencí znaků, použitých například při generování čísla karty zákazníka, nebo pro vytvoření defaultního hesla nového uživatele. Dále je zde obsažena třída starající se o vytváření a správu nových oken aplikace.

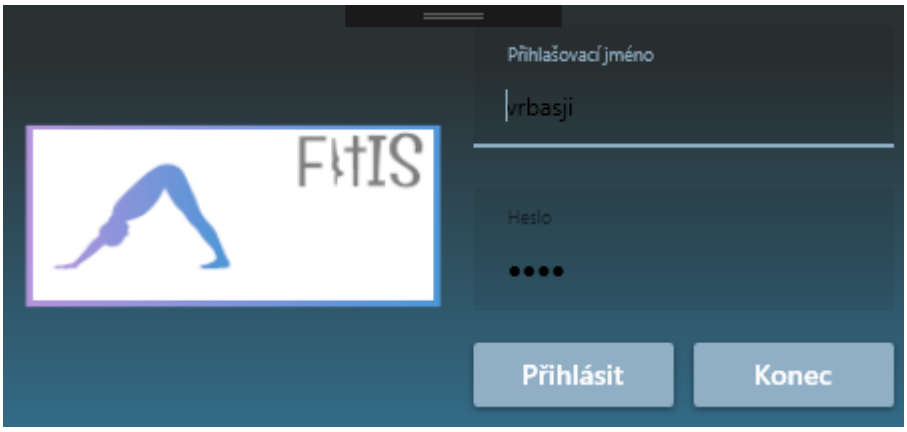
3.6 Moduly a popis funkcionalit aplikace

Celá aplikace je rozdělena do několika samostatných částí, takzvaných modulů.

Níže budou jednotlivé moduly popsány a bude představena jejich vizuální podoba.

3.6.1 Přihlášení

Při spuštění aplikace nebo při odhlášení se zobrazí okno pro přihlášení uživatele. Uživatel zadá přihlašovací jméno, heslo a proběhne jeho ověření. Při správném ověření je uživatel přihlášen do systému a jsou mu dále zpřístupněny funkcionality aplikace podle role, která mu byla přiřazena.

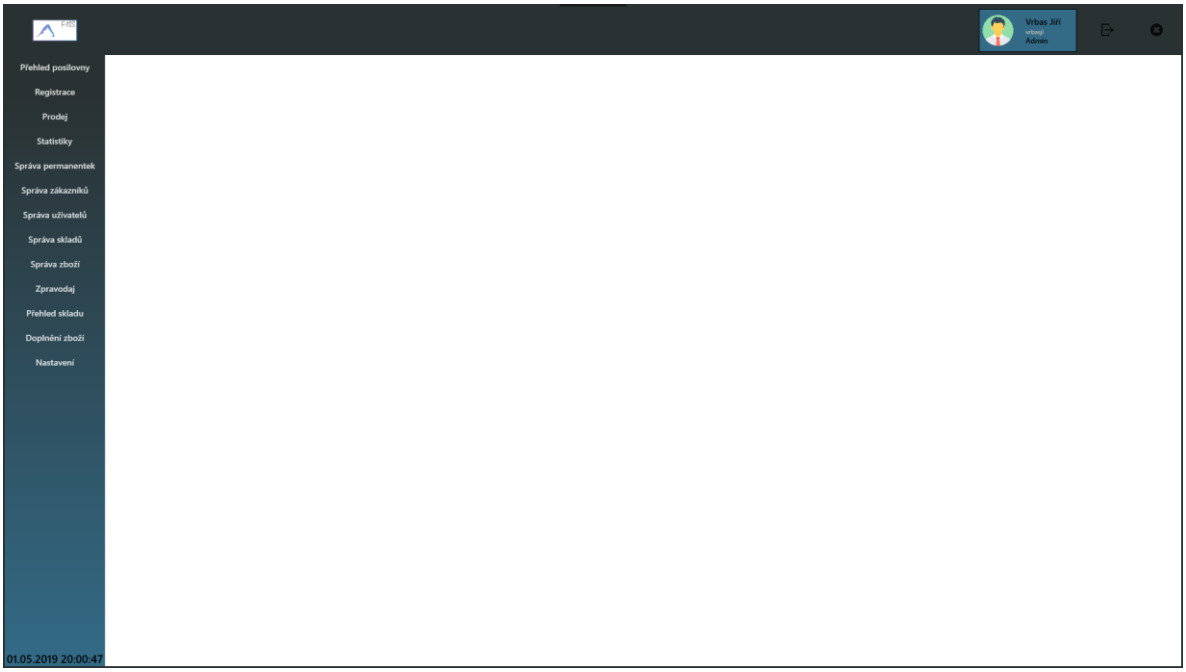


Obrázek 14 – Okno přihlášení

3.6.2 Hlavní okno

Po úspěšném přihlášení je uživateli zobrazeno hlavní okno aplikace. V jeho horní části se nachází lišta s logem informačního systému na levé straně a v pravé části se nachází informační panel. V tomto panelu je obrázek symbolizující uživatele, dále pak jeho celé jméno, přihlašovací jméno a název role, pod kterou je do systému přihlášen. Vpravo od informačního panelu se nachází tlačítka pro odhlášení uživatele a také pro ukončení aplikace. V levé části hlavního okna se nachází dynamické menu, do kterého jsou přidávána

jednotlivá tlačítka dle toho, pod jakou rolí je uživatel do systému přihlášen. Tím je také zabráněno zpřístupnění těch částí systému, do nichž by uživatel neměl mít přístup. Jelikož aplikace zabírá celou plochu monitoru, nepředpokládá se, že by uživatel používal koncové zařízení ještě jiným způsobem. Z toho důvodu je pod menu tlačítky aplikace zobrazen aktuální čas a datum systému. Zbytek prostoru je vyplněn dynamickými prvky, v nichž jsou zobrazeny jednotlivé funkcionality systému.



Obrázek 15 – Hlavní okno aplikace

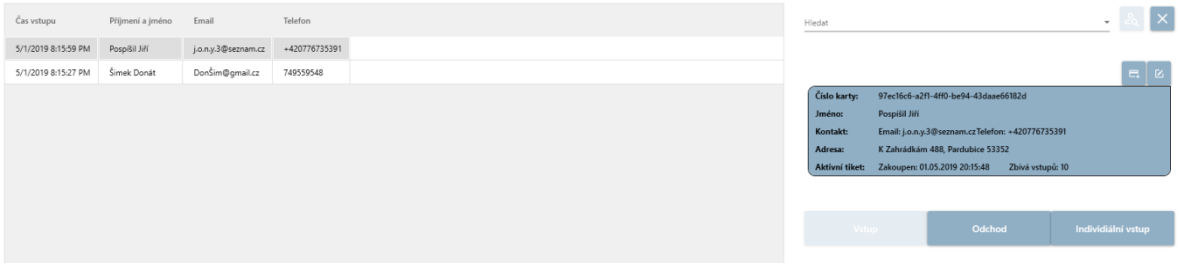
3.6.3 Přehled posilovny

Okno přehledu posilovny poskytuje uživateli evidovat vstupy a odchody zákazníků z fitness centra, dále má možnost také vidět přehled zákazníků, kteří se ve fitness centru nacházejí. V levé části obrazovky se nachází přehled zákazníků, kteří do systému vstoupili. V pravé části má uživatel možnost hledat mezi všemi zákazníky registrovanými v systému. Toto vyhledávání je umožněno například podle jména, příjmení nebo čísla karty. V případě potřeby je zde možnost rozšíření o čtečku karet, aby mohl zákazník zaevidovat vstup či odchod sám bez asistence člověka na recepci. Do pole pro vyhledávání je možné buď přímo vepsat vyhledávací řetězec, nebo po kliknutí na toto pole se zobrazí rozbalovací seznam se všemi zákazníky.

Po vyhledání zákazníka se zobrazí jeho informace pod vyhledávacím polem. Mezi těmito informacemi je například číslo karty, celé jméno, kontaktní informace, adresa a informace o aktivní permanentce. V případě, že zákazník nemá aktivní žádnou permanentku, je zde tato

informace také zobrazena a je nutné ji pro úspěšný vstup zákazníka do systému dobít. Dobití je možné pomocí ikony kreditní karty se symbolem plus nad polem s informacemi, poté je aplikace přesměrována na okno v kapitole 3.6.8. Vedle ikony pro dobití je ikona pro úpravu informací o zákazníkovi. Pomocí tohoto tlačítka je aplikace přesměrována do okna z kapitoly 3.6.9, kde je možné tyto informace upravit.

Pod oknem s informacemi jsou tlačítka pro evidování vstupu, nebo odchodu zákazníka. Tato tlačítka jsou deaktivována, pokud není vybrán žádný zákazník. Z důvodu povolení vstupu i neregistrovaným zákazníkům je zde tlačítko pro vstup neregistrovaného zákazníka. Po stisknutí tohoto tlačítka se otevře okno pro prodej jednotlivého vstupného viz okno z kapitoly 3.6.5. Po zaplacení účtenky je zaevidován vstup, ale nezobrazí se v seznamu aktuálních návštěvníků. Tento vstup je zaevidován pouze pro účely vyúčtování a také pro správnost počítaných statistik.



Obrázek 16 – Přehled posilovny

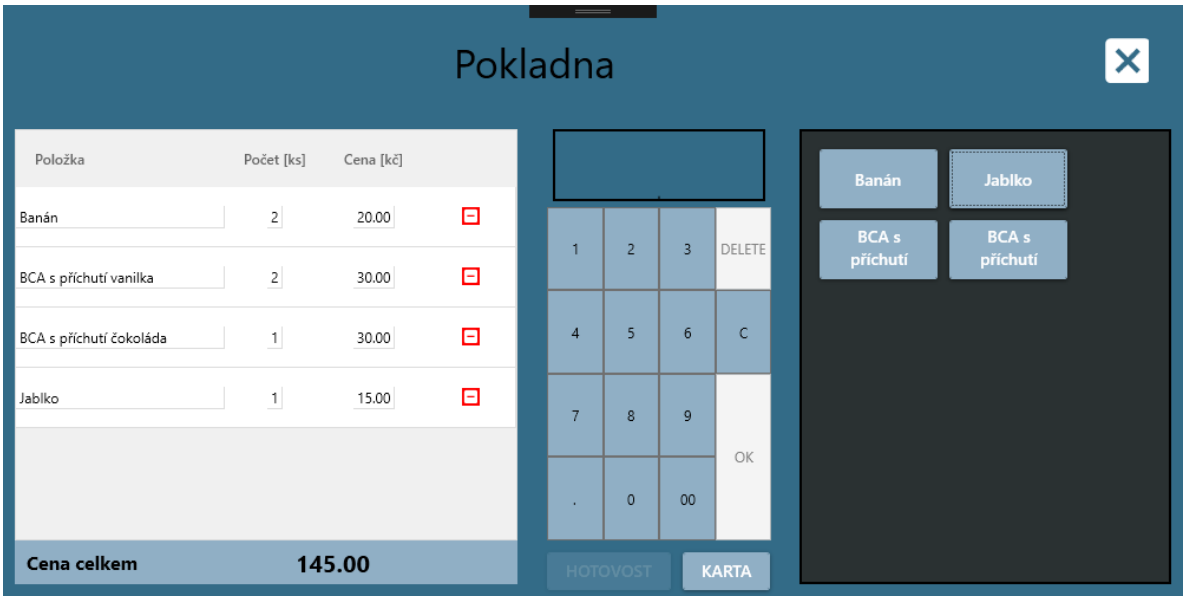
3.6.4 Registrace zákazníka

Okno pro registraci zákazníka obsahuje pouze vstupní pole pro jméno, příjmení, e-mail, telefon, ulici, město, číslo domu a poštovní směrovací číslo. Všechny tyto údaje jsou povinné. Poté je možné zákazníka „vytvořit“ po stisknutí tlačítka Registrovat. Pokud registrace proběhla v pořádku, v horní části aplikace se zobrazí informativní hláška a je možné přejít k zakoupení permanentky uživateli do okna v kapitole 3.6.8.

Obrázek 17 – Okno registrace zákazníka

3.6.5 Prodej zboží

Modul pro prodej zboží je důležitou součástí systému. Toto okno je vyvoláno, pokud si chce zákazník koupit nějaké občerstvení, novou permanentku nebo při jakékoliv činnosti spojené s platbou. Toto okno je rozděleno na tři části. V pravé části se nachází tlačítka s přednastavenými položkami, které je možné zakoupit. Po kliknutí na položku se automaticky přidá k účtence. V prostřední části se nachází modul platby. V případě platby za hotové zde recepční zadá hodnotu bankovek, které obdrží od zákazníka. Ta musí být větší než celková cena na účtence. Poté je možné platbu uskutečnit. Systém uživatele upozorní, kolik má zákazníkovi vrátit, a platbu zaeviduje. Je zde také připravena možnost platby kreditní kartou, ale tato funkcionalita zatím nebyla implementována. V levé části obrazovky se nachází přehled samotné účtenky. Je zde tabulka položek účtenky, kde vidíme název položky na účtence, počet kusů a celkovou cenu za tyto položky. Jednoduchým kliknutím, můžeme položky přidávat, či odebírat. Je zde také uvedena celková cena účtenky.



Obrázek 18 – Okno pokladny

3.6.6 Statistiky

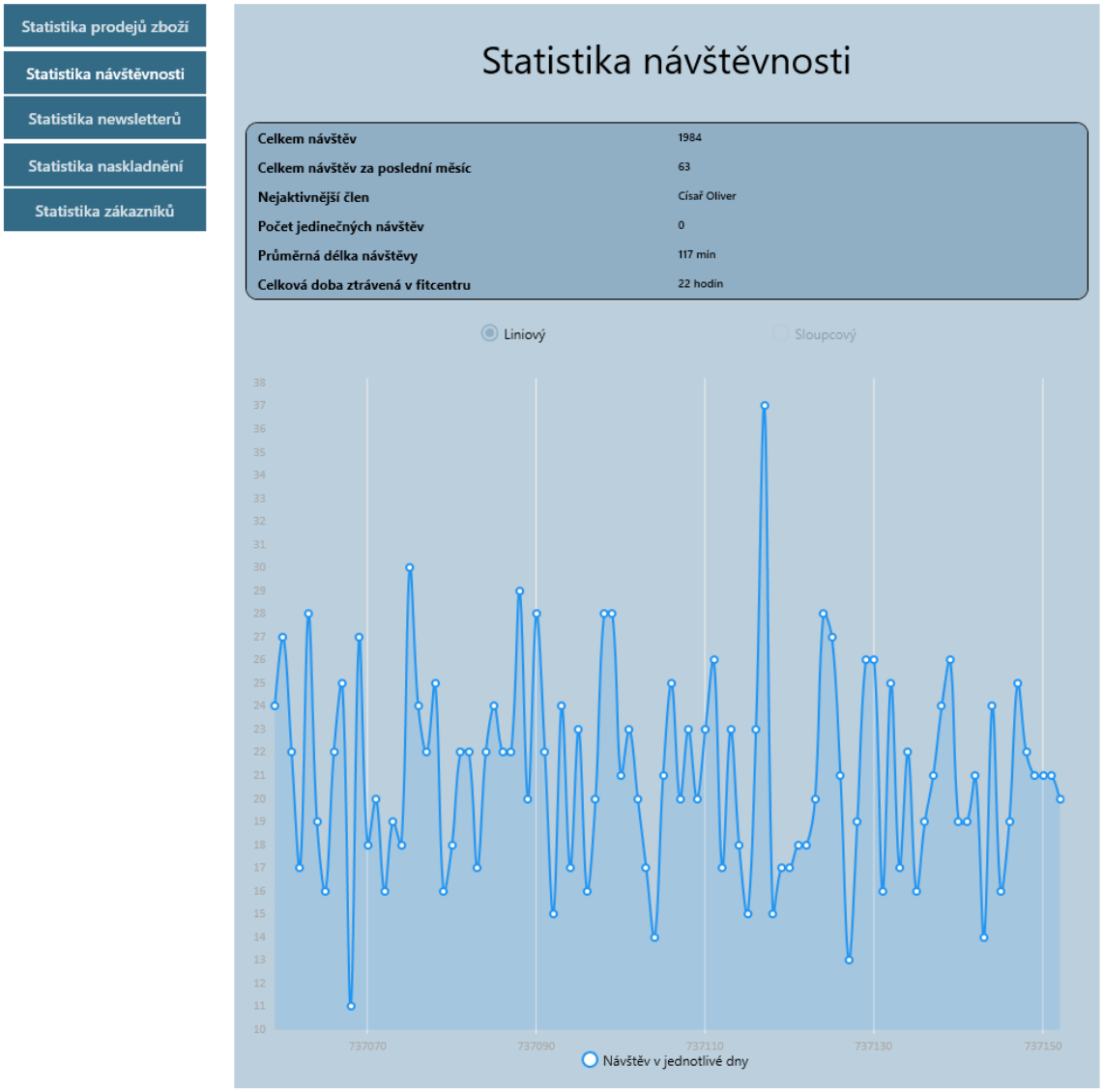
Dalším důležitým modulem je modul statistik, který je dostupný pouze uživateli s rolí manažer. V tomto modulu se nachází několik dílčích statistik s přehlednými grafy rozdělenými do logických celků.

Po levé straně se nachází tlačítka, s jejichž pomocí se zobrazí podrobnější statistiky ve formě textu a také v grafické formě, a to buď jako liniový graf, nebo jako graf sloupcový. Mezi tyto statistiky patří:

1. Statistika prodejů zboží – zobrazuje počet prodaných kusů zboží, celkovou cenu prodaného zboží, nejdražší objednávku, průměrný počet položek na účtence, nejvíce prodávané zboží a nejvíce prodávané permanentky. V grafu jsou poté zobrazeny prodeje v jednotlivé dny.
2. Statistika návštěvnosti – zobrazuje celkový počet návštěv, celkový počet návštěv za poslední měsíc, jméno nejaktivnějšího člena, počet jednorázových vstupů, průměrnou délku návštěvy a celkovou dobu, jakou zákazníci strávili ve fitness centru. V grafu jsou poté zobrazeny počty návštěv v jednotlivé dny.
3. Statistika zpravodajů – zobrazuje počet odeslaných zpravodajů, celkový počet oslovených zákazníků, průměrný počet oslovených zákazníků, celkem odeslaných e-mailů a celkem odeslaných SMS. V grafu jsou poté zobrazeny počty odeslaných zpravodajů v jednotlivých dnech.

4. Statistika naskladnění – zobrazuje celkový počet zboží ve skladech, název zboží, kterého je ve skladu nejvíce a nejméně, a průměrný počet zboží na skladě. V grafu je poté zobrazena naplněnost jednotlivých skladů.
5. Statistika zákazníků – zobrazuje počet registrovaných zákazníků, počet aktivních vstupů za poslední měsíc, počet jedinečných měst, kde mají zákazníci trvalý pobyt, poměr vstupních a časových permanentek, celkový počet nespotřebovaných vstupů a počet vyřazených permanentek. V grafu jsou poté zobrazeny počty permanentek zakoupených v jednotlivých dnech.

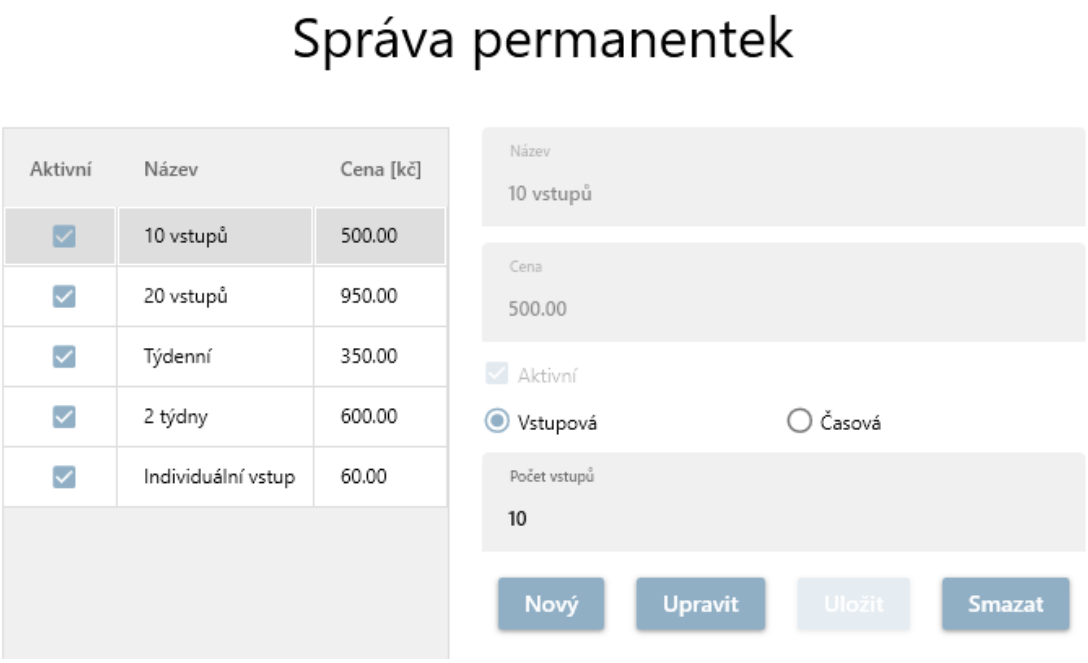
V případě potřeby je možné implementovat další statistiky na přání klienta.



Obrázek 19 – Okno statistik

3.6.7 Správa permanentek

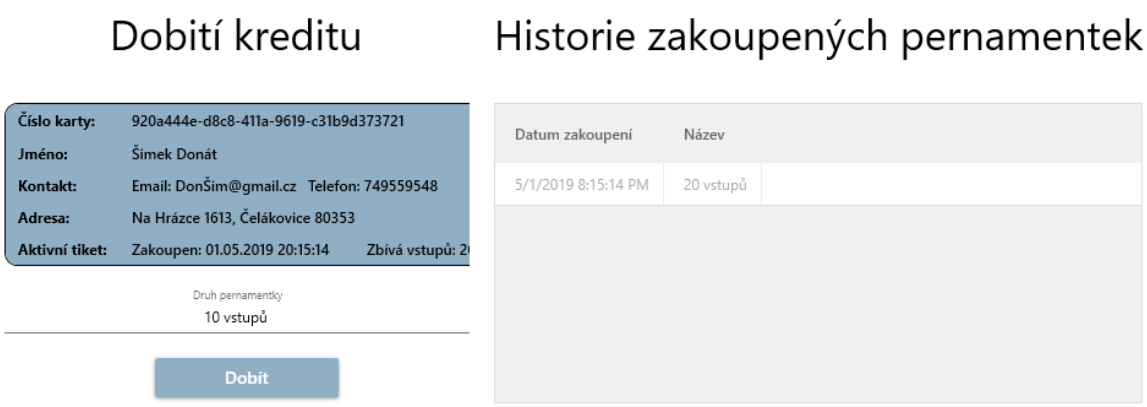
Modul správy permanentek je klasický číselník. V levé části se nachází tabulka s přehledem všech permanentek v systému. Je zde uvedeno, zda je permanentka aktivní, její název a cena. Uživatel má možnost vytvořit, smazat nebo upravit permanentku. Při vytváření a úpravě je nutné v pravé části obrazovky zadat název, cenu a zvolit, zda bude permanentka aktivní. To určuje, zda budou moci zákazníci tuto permanentku koupit. Dále je pak možné zvolit, jestli se bude jednat o vstupovou nebo časovou permanentku. U vstupové permanentky je dále možnost zadat počet vstupů a u časové pernamentky dobu platnosti ve dnech.



Obrázek 20 – Správa permanentek

3.6.8 Prodej permanentek

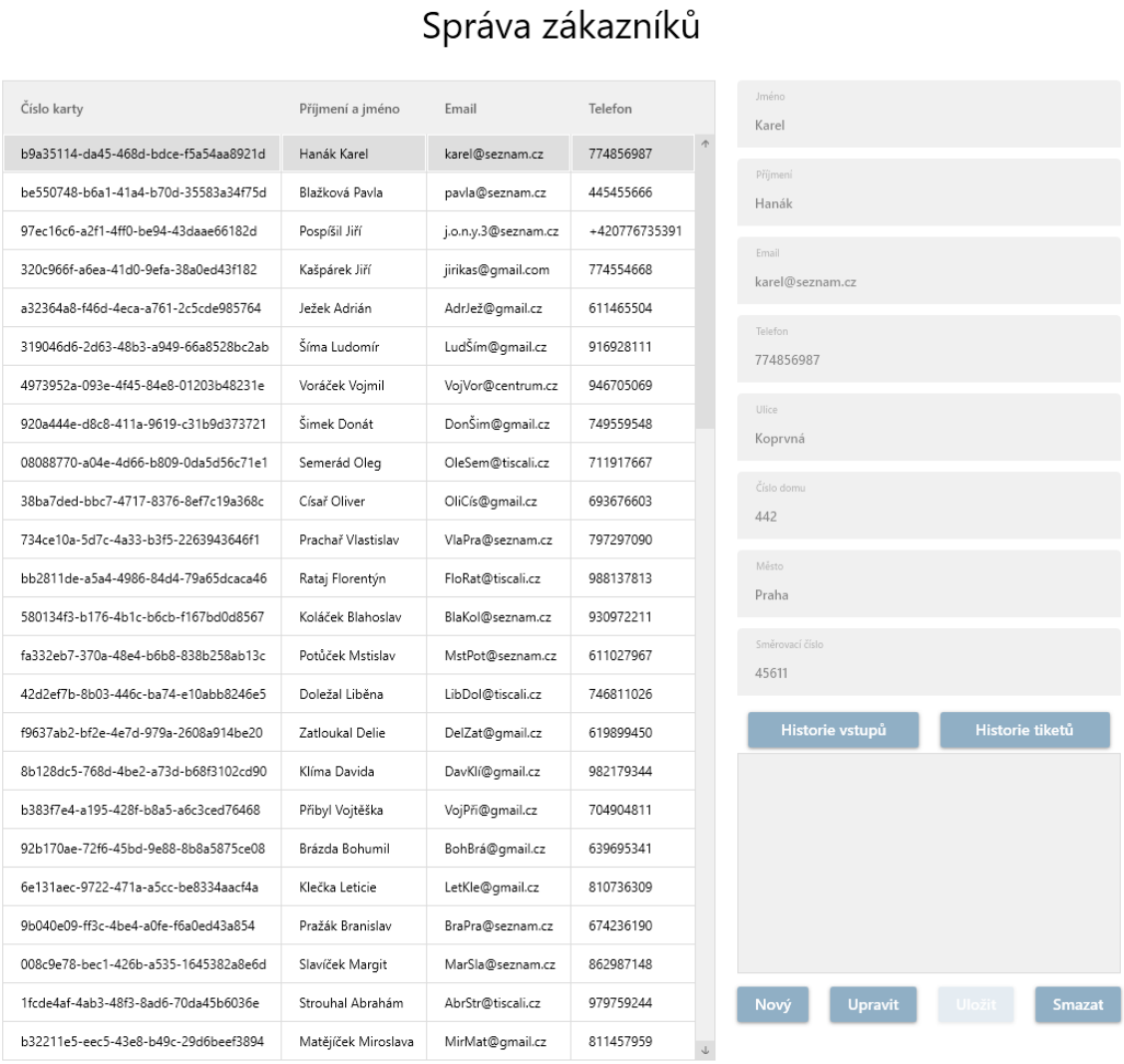
Modul pro prodej permanentek je používán, pokud zákazníkovi vypršela platnost permanentky nebo vyčerpal všechny vstupy. V tomto okně může recepční zákazníkovi dobít novou permanentku. V levé části okna se nachází detailní informace o zákazníkovi, jako jsou číslo karty, celé jméno, kontaktní informace, adresa a informace o aktivní permanentce. Pod těmito informacemi se nachází rozbalovací menu se všemi aktivními permanentkami v systému. Po zvolení možnosti k dobití permanentky se uživateli zobrazí klasické okno pro platbu viz kapitola 3.6.5. Po úspěšném zaplacení je zákazníkovi karta dobita a může znovu vstoupit do fitness centra. V pravé části okna má uživatel možnost vidět historii zakoupených permanentek.



Obrázek 21 – Okno dobití kreditu

3.6.9 Správa zákazníků

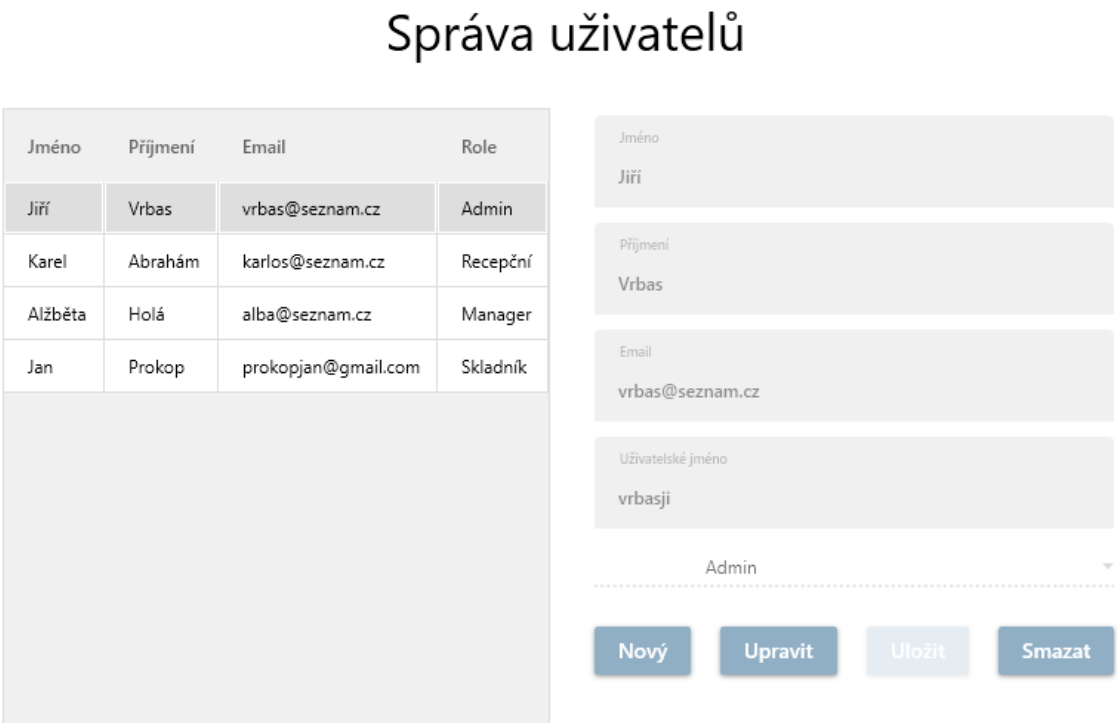
Modul správy zákazníků je další z číselníků. V levé části se nachází tabulka s přehledem všech zákazníků. Je zde uvedeno číslo karty zákazníka, jeho celé jméno, e-mail a telefon. Uživatel má možnost vytvořit, smazat nebo upravit zákazníka. Při vytváření a úpravě je nutné v pravé části obrazovky zadat jméno, příjmení, e-mail, telefon a kontaktní údaje. Pod poli pro zadávání se nachází dvě tlačítka pro zobrazení historie vstupů uživatele a historie zakoupených permanentek. U historie vstupů je zobrazen čas příchodu, čas odchodu a počet minut strávených danou návštěvu ve fitness centru. V historii permanentek může uživatel vidět, jakou permanentku měl zákazník v minulosti zakoupenou.



Obrázek 22 – Správa zákazníků

3.6.10 Správa uživatelů

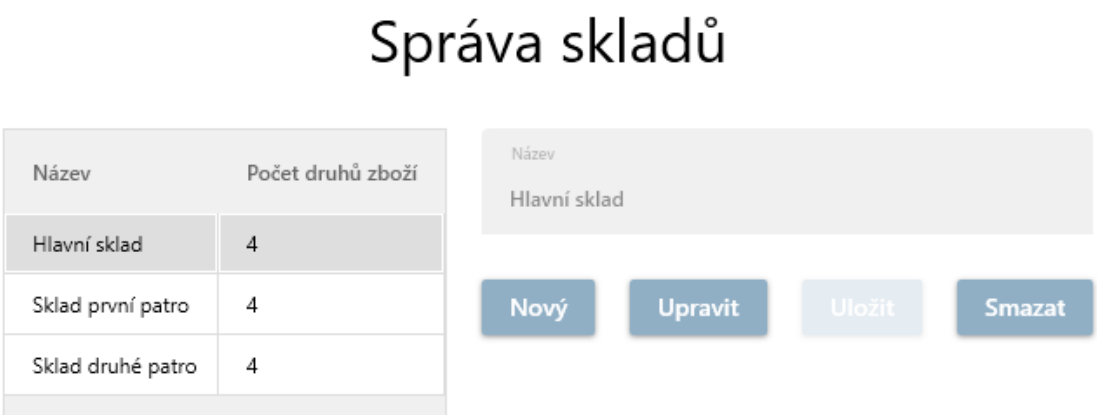
Modul správy uživatelů je další z číselníků. V levé části se nachází tabulka s přehledem všech uživatelů. Je zde uvedeno jméno, příjmení, uživatelské jméno a role uživatele. Uživatel s právy pro správu má možnost vytvořit, smazat nebo upravit jiného uživatele. Při vytváření a úpravě je nutné v pravé části obrazovky zadat jméno, příjmení, uživatelské jméno, e-mail a vybrat roli. Po založení nového uživatele je v informačním okně zobrazeno přihlašovací jméno a vygenerované heslo.



Obrázek 23 – Správa uživatelů

3.6.11 Správa skladů

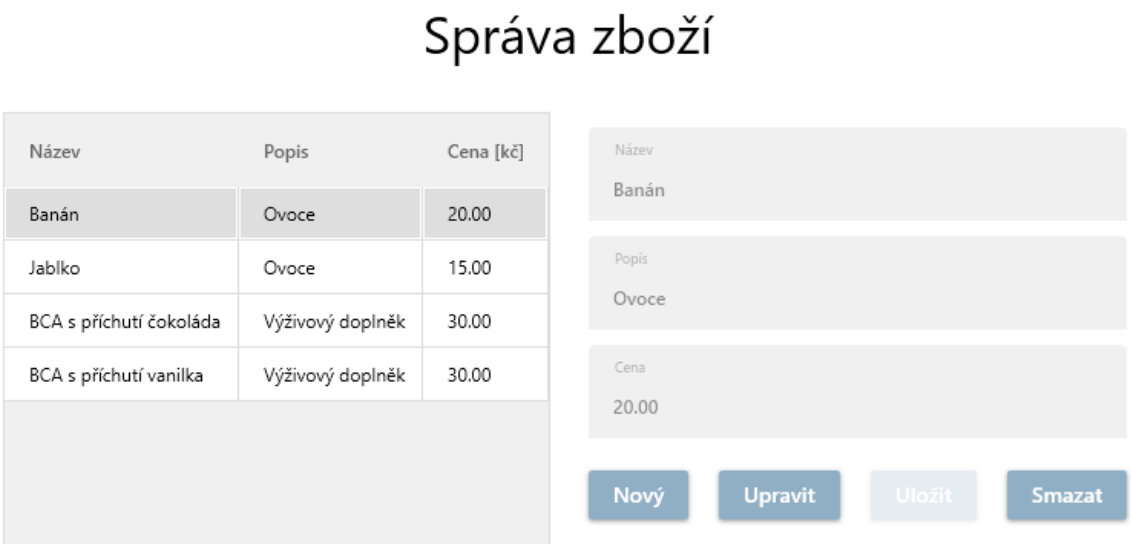
Modul správy skladů je klasický číselník. V levé části se nachází tabulka s přehledem všech skladů v systému. Je zde uvedeno jméno skladu a počet jednotlivých druhů zboží nacházejících se ve skladu. Uživatel má možnost vytvořit, smazat nebo upravit sklad. Při vytváření a úpravě je nutné v pravé části obrazovky zadat pouze název skladu.



Obrázek 24 – Správa skladů

3.6.12 Správa zboží

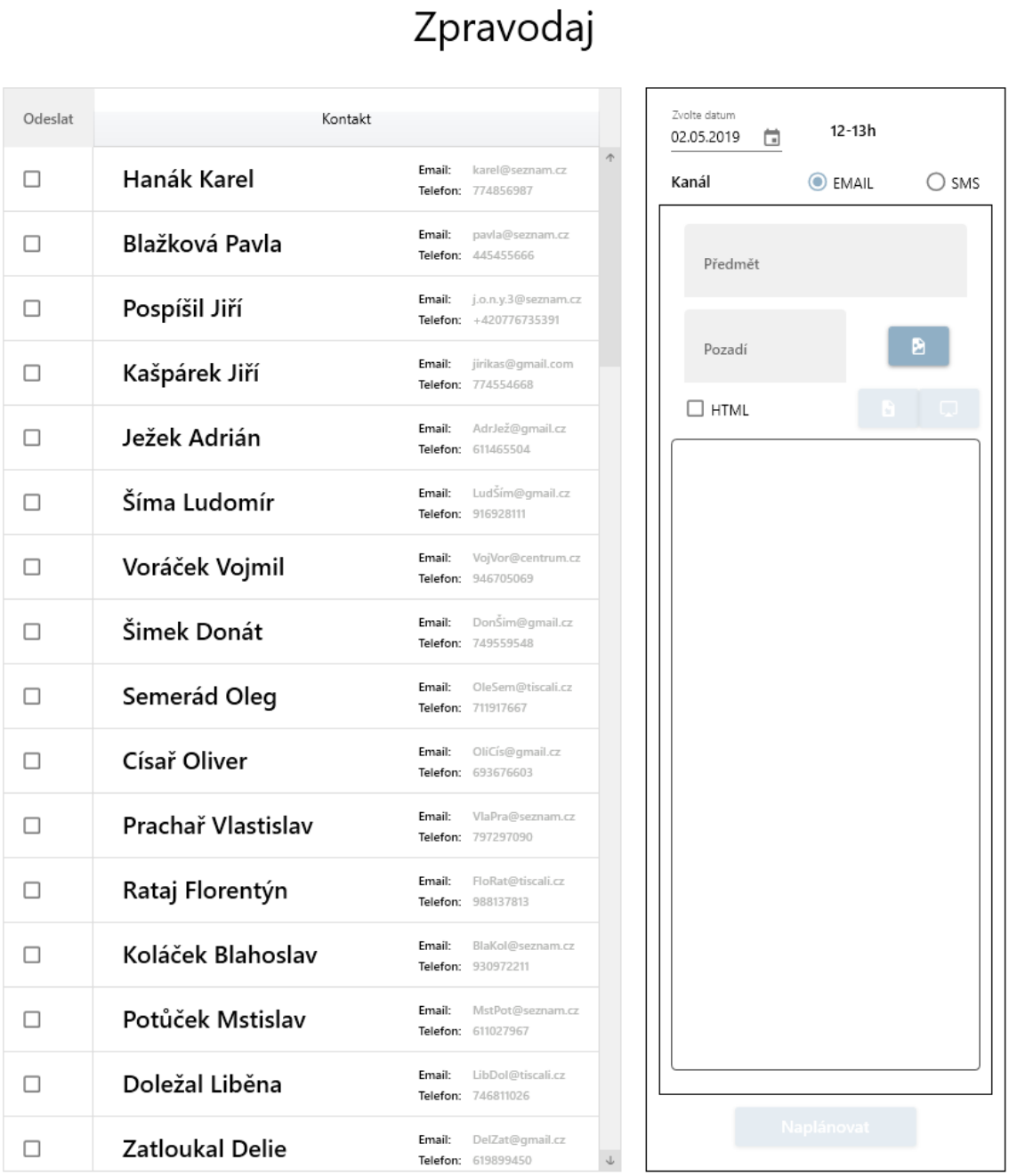
Modul správy zboží je další z číselníků. V levé části se nachází tabulka s přehledem všeho zboží. Je zde uveden název, popis a cena zboží v korunách. Uživatel má možnost vytvořit, smazat nebo upravit zboží. Při vytváření a úpravě je nutné v pravé části obrazovky zadat název, popis a cenu zboží.



Obrázek 25 – Správa zboží

3.6.13 Rozeslání zpravodaje

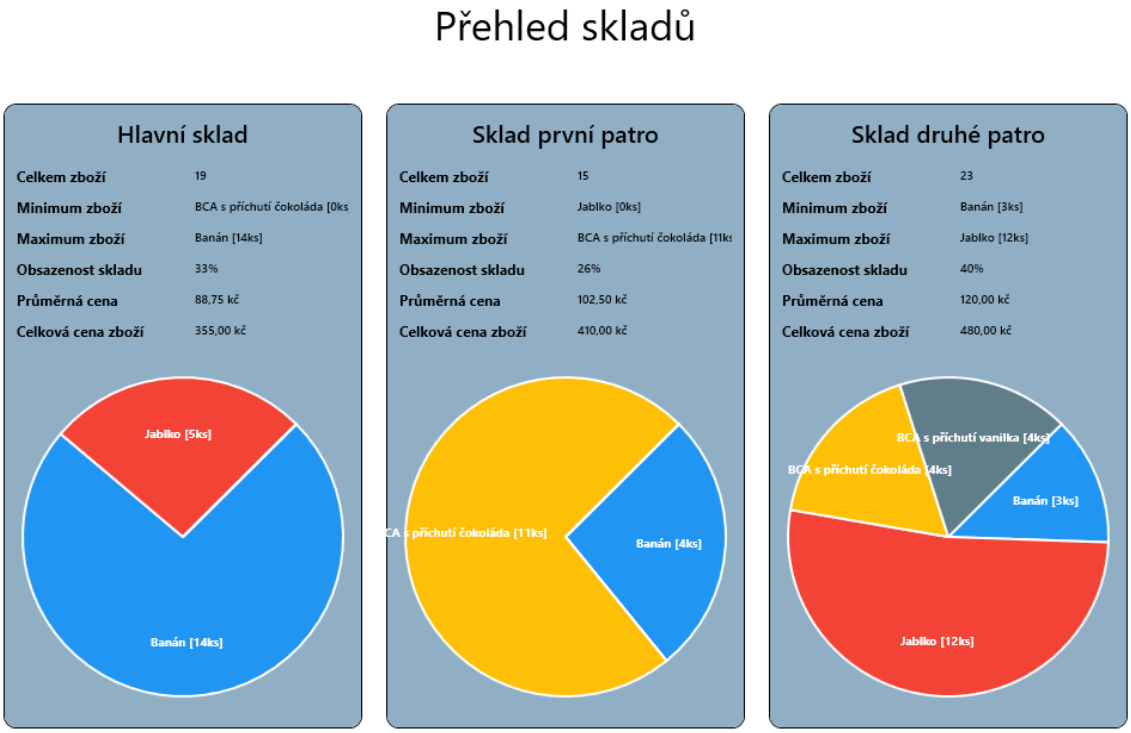
Dalším modulem je rozesílání zpravodaje. V tomto okně si může manažer naplánovat rozesílání informativního zpravodaje. V levé části okna uživatel zvolí zákazníky, kterým chce zpravodaj zaslat. Poté v levé části okna dochází k samotnému naplánování. Nejprve je nutné zvolit datum odeslání, poté způsob, jakým budou zákazníci informováni. Momentálně je možné zákazníky kontaktovat formou e-mailu, nebo SMS zprávy. Pokud uživatel zvolí formu SMS zprávy, je nutné vyplnit pouze obsah zprávy. Pokud uživatel zvolí formu e-mailu, je nutné zvolit předmět zprávy a obsah. Je zde také možnost zaslat e-mail ve formě HTML pro graficky přívětivější design. V tomto případě uživatel může vyplnit pole pro tělo e-mailu nebo importovat HTML soubor pomocí tlačítka pro import. Poté jsou zde tlačítka pro zobrazení nebo validaci daného e-mailu. Pokud má být e-mail ve formě HTML, je nutné, aby byl validní. V případě zobrazení se otevře nové okno s přesným vzhledem výsledného e-mailu. Po kompletním nastavení stačí pouze stisknout tlačítko Naplánovat a systém vytvoří naplánovanou úlohu, která ve zvolený čas odešle zpravodaj zvoleným zákazníkům.



Obrázek 26 – Okno zpravodaje

3.6.14 Přehled skladů

V modulu pro přehled skladů se nachází informace o jejich naplněnosti. Jsou zde seřazeny všechny sklady a informace o nich. Pro každý sklad je zde uveden celkový počet zboží, název zboží, jehož je na daném skladu nejméně i s počtem kusů, název zboží, jehož je na daném skladu naopak nejvíce, opět i s počtem kusů, procentuální obsazenost skladu, průměrná cena zboží na skladě a celková cena zboží na skladě. Dále je zde zobrazen graf pro každý sklad, který na koláčovém grafu ukazuje obsazenost skladu jednotlivými typy zboží.



Obrázek 27 – Okno přehled skladů

3.6.15 Doplnění zboží

Tento modul slouží skladníkovi k tomu, aby mohl jednoduše do systému doplnit zboží. Okno obsahuje filtrování skladu, ve kterém uživatel bude doplňovat zboží, a vedle toho uživatel může vyfiltrovat zboží, jež ve skladu už dochází, nebo už došlo. Pod filtry se nachází tabulka s informacemi o zboží. Obsahuje název zboží, popis, cenu v korunách a počet kusů na skladě. V tabulce jsou také podbarveny červeně vykoupené položky, případně žlutě je podbarveno zboží v omezeném počtu kusů. V případě potřeby změny počtu kusů uživatel vybere zboží z tabulky a v pravé části do připraveného pole zadá počet kusů, poté může tento počet buď přidat, nebo odebrat. Změny se ihned promítnou do zdrojové tabulky.

Obrázek 28 – Okno plnění skladů

3.6.16 Nastavení

V modulu nastavení se nachází globální nastavení celého systému. V současné době zde nalezneme nastavení účtu, odkud jsou odesílány e-maily a také SMS zprávy. Do této části aplikace má možnost vstoupit pouze administrátor. Při nastavení e-mailu je nutné zadat jméno odesílatele, e-mailovou adresu, z níž budou e-maily odeslány a také heslo pro přístup k e-mailu. Pro testovací účely byl použit e-mail registrovaný u společnosti Google. Pro odesílání SMS zpráv je nutné zadat také jméno odesílatele a klíč k API. SMS brána je napojená na API od britského poskytovatele control.txtlocal. V základním balíčku je povoleno zaslat až 100 zpráv denně zdarma. Na přání klienta je možné doplnit další způsoby odeslání e-mailů nebo SMS zpráv.

Obrázek 29 – Nastavení

3.7 Uživatelské role

V celé aplikaci je definováno několik uživatelských rolí. Každý uživatel musí mít přidělenu právě jednu z nich, jinak by mu nebylo umožněno se do systému přihlásit.

Rozdělení rolí v tabulce Tabulka 4 – Povolené moduly podle role uživatele jsou následující:

1. Admin – účet s tímto oprávněním má přístup do celé aplikace a může v systému provádět jakékoliv operace.
2. Recepční – u uživatele s touto rolí se předpokládá největší interakce se zákazníky, tudíž může zákazníka do systému zaregistrovat, dále zaevidovat jeho vstup a odchod z posilovny, dobít mu novou permanentku nebo prodat jakékoliv zboží.
3. Manažer – uživatel s touto rolí se stará o nastavení systému, může tedy vytvářet nové uživatele, zákazníky, sklady, zboží a permanentky. Dále může rozesílat zpravodaje. Nedílnou součástí je také možnost prohlížet si nejrůznější statistiky systému.
4. Skladník – tato role je určena pro uživatele, který se stará o sklad a zásoby zboží. Může tedy spravovat prodávané zboží a doplňovat ho do skladu.

Tabulka 4 – Povolené moduly podle role uživatele

Modul/Role	Admin	Recepční	Manažer	Skladník
Přehled posilovny	Ano	Ano	Ano	Ne
Registrace zákazníka	Ano	Ano	Ne	Ne
Prodej zboží	Ano	Ano	Ne	Ne
Statistiky	Ano	Ne	Ano	Ne
Správa permanentek	Ano	Ne	Ano	Ne
Prodej permanentek	Ano	Ano	Ne	Ne
Správa uživatelů	Ano	Ne	Ano	Ne
Správa skladů	Ano	Ne	Ano	Ano
Správa zboží	Ano	Ne	Ano	Ano
Rozesílání zpravodaje	Ano	Ne	Ano	Ne
Přehled skladu	Ano	Ne	Ano	Ano
Doplnění zboží	Ano	Ne	Ne	Ano
Nastavení	Ano	Ne	Ano	Ne

V databázi jsou pro účely testování vytvořeni uživatelé, díky nimž je možno aplikaci vyzkoušet pod všemi dostupnými uživatelskými rolemi:

Tabulka 5 – Testovací uživatelé

Role	Přihlašovací jméno	Heslo
Admin	vrbasji	1234
Recepční	karlos	1234
Manažer	hoal	1234
Skladník	prja	1234

3.8 Instalační příručka

Pro správné fungování celého systému je potřeba správně nastavit operační systém, na kterém aplikace poběží, případně server, na kterém bude umístěna databáze. Nejprve je důležité vytvořit novou databázi na databázovém serveru nebo lokálním počítači a s ní související databázové tabulky. Díky architektuře systému může být databáze na serveru odděleném od aplikace obsahující uživatelské rozhraní. Celý systém byl testován na Microsoft SQL Serveru 2016. Po vytvoření databáze je nutné změnit v konfiguračním souboru App.config umístěném

v domovské složce aplikace hodnotu klíče FitConStr. Tato hodnota musí odpovídat připojovacímu řetězci k vytvořené databázi.

Dále pro správné fungování vstupů neregistrovaných zákazníků je zapotřebí vložit dva záznamy o jednorázových vstupech, a to do tabulky Customers a Tickets. Je zde také přiložen skript pro vytvoření uživatele s rolí administrátor, s jehož pomocí se bude možné přihlásit do systému a provést základní nastavení systému a vytvořit ostatní uživatele. Přihlašovací jméno je `Admin` a heslo `Admin`. Skripty pro vložení záznamů jsou přiloženy v Příloha A – SQL se zdrojovými daty. Na koncové stanici, kde bude systém provozován, je nutné mít nainstalovanou verzi .Net frameworku ve verzi nejméně 4.6.1 a počítač musí být připojen k internetu. Všechny přiložené zdrojové soubory je nutné mít připraveny v samostatné složce. V ní se poté automaticky vytvoří složka Logs, kde se budou nacházet soubory s logováním jednotlivých funkcí systému. Pro zjednodušení zprovoznění systému je součástí instalační aplikace.

Po úspěšném přihlášení do systému bude moci uživatel provést základní nastavení. To se týká především oblasti v modulu Nastavení, kde je nutné pro správné fungování zasílání zpravodaje nastavit potřebné údaje pro zasílání e-mailů a SMS. Po úspěšném nastavení systému může být aplikace spuštěna souborem FitIS.exe, který je součástí přiložených souborů.

4 ZÁVĚR

Proces vývoje informačního systému není zcela jistě jednoduchou záležitostí. I když již mám nějaké zkušenosti s programováním informačních systémů, jsem rád, že během psaní této diplomové práce jsem si mohl vyzkoušet, jak se takový systém tvoří od samotného počátku produktu.

Podle analýzy v teoretické části práce bylo zjištěno, že na trhu existuje pouze pár komerčních řešení, která ne vždy musí vyhovovat požadavkům zákazníka. Proto se zákazníci často uchylují k informačnímu systému vytvořenému přímo na míru. Toto řešení poskytuje zákazníkovi systém přesně podle jeho představ a požadavků, avšak často je vytvoření takového systému finančně náročné, což může menší podniky donutit pořídit si již připravené řešení. Teoretická část dále obsahuje popis jednotlivých fází vývoje softwaru. Také byly prozkoumány jednotlivé přístupy k vývoji softwaru a popsána metodika zvolená pro tvorbu této práce.

Výsledkem praktické části práce je kompletní systém, který by mohl být nasazen do jakéhokoliv fitness centra a mohl by být bez větších problémů ihned zařazen do provozu. Systém splňuje všechny požadavky, které byly při analýze zjištěny, a byl také rozšířen o některé funkcionality, jež by takový systém mohl poskytovat. Během tvorby systému jsem se zaměřil hlavně na čistotu kódu a počítal jsem i s budoucím rozšířením systému. Proto se do budoucna počítá například s napojením celého systému na elektronickou evidenci tržeb, napojení platby kartou, vytvoření rezervačního portálu, kde by se zákazníci mohli registrovat na lekce poskytované fitness centrem nebo na napojení čtečky karet, čímž by se usnadnila práce recepční.

Při programování praktické části jsem často narážel na různá omezení, ať už ze strany použitých komponent, nebo softwarového návrhu. Bylo zapotřebí promyslet celou strukturu projektu a zvolit správný postup při implementaci dílčích částí systému. Jako programátor jsem si prohloubil znalosti používaných technologií a zjistil jsem, že je zde neustále prostor pro zlepšování.

Cíle stanovené v úvodu práce byly splněny, a i když se technologie v oblasti programování neustále vyvíjí, tak postupy popsané v této práci budou jistě pár let použitelné pro vývoj jakýchkoliv aplikací pro platformu Windows.

5 POUŽITÁ LITERATURA

[1] MARTINŮ, J. a P. ČERMÁK. Metodiky vývoje software. Olomouc: Moravská vysoká škola Olomouc, o.p.s., 2018. Dostupné také z: https://mvso.cz/wp-content/uploads/2018/02/Metodiky-v%C3%BDvoje-software-studijn%C3%AD-text.pdf

[2] ŠMÍD, V. Životní cyklus informačního systému. In: *Fi.muni.cz* [online]. © 2017 [cit. 2019-03-03]. Dostupné z: https://www.fi.muni.cz/~smid/mis-zivcyk.htm

[3] BĚHÁLEK, M. *Programovací jazyk C#* [online]. Ostrava: VŠB-TU Ostrava [cit. 2019-03-03]. Dostupné z: http://www.cs.vsb.cz/behalek/vyuka/pcsharp/text.pdf

[4] Microsoft. Úvod do NuGet. *Microsoft.com* [online]. 10. 1. 2018 [cit. 2019-03-03]. Dostupné z: https://docs.microsoft.com/cs-cz/nuget/what-is-nuget

[5] ČÁPKA, D. Lekce 1 - Úvod do WPF (Windows Presentation Foundation). In: *Itnetwork.cz* [online]. 2015 [cit. 2019-03-03]. Dostupné z: https://www.itnetwork.cz/csharp/formulare/wpf/c-sharp-tutorial-wpf-uvod-a-prvni-formularova-aplikace/

[6] JECHA, T. Architektura wpf – dispatcher. In: *Dotnestportal.cz* [online]. 26. 1. 2012 [cit. 2019-03-03]. Dostupné z: https://www.dotnetportal.cz/clanek/197/Architektura-WPF-Dispatcher

[7] JECHA, T. Jazyk XAML. In: *Dotnestportal.cz* [online]. 2. 2. 2012 [cit. 2019-03-03]. Dostupné z: https://www.dotnetportal.cz/clanek/198/Jazyk-XAML

[8] DAJBYCH, D. Mvvm: model-view-viewmodel. In: *Dotnetportal.cz* [online]. 21. 4. 2009 [cit. 2019-03-03]. Dostupné z: https://www.dotnetportal.cz/clanek/4994/MVVM-Model-View-ViewModel

[9] Entity Framework Tutorial. What is Entity Framework? *Entityframeworktutorial.net* [online]. 2018 [cit. 2019-03-03]. Dostupné z: https://www.entityframeworktutorial.net/what-is-entityframework.aspx

[10] ČÁPKA, D. Lekce 7 - LINQ v C# - revoluce v dotazování. In: *Itnetwork.cz* [online]. 2013 [cit. 2019-03-03]. Dostupné z: https://www.itnetwork.cz/csharp/kolekce-a-linq/c-sharp-tutorial-linq-dotazy

[11] Catena Logic. About. *Catelproject.com* [online]. 2018 [cit. 2019-03-03]. Dostupné z: https://www.catelproject.com/

[12] Apache Software Foundation. What is Apache log4net™ *Apache.org* [online]. 10. 1. 2018 [cit. 2019-03-03]. Dostupné z: https://logging.apache.org/log4net/

[13] Material Design In XAML. Material Design In XAML Toolkit. *Materialdesigninxaml.net* [online]. 2018 [cit. 2019-03-03]. Dostupné z: http://materialdesigninxaml.net/

[14] Tutorials Point. SDLC – Overview. *Tutorialspoint.com* [online]. 2019 [cit. 2019-03-03]. Dostupné z: https://www.tutorialspoint.com/sdlc/sdlc_overview.htm

[15] WIENHOLT, N. *Maximizing .NET performance.* New York: Springer-Verlag, 2004. ISBN 1-59059-141-0.

[16] KUNAL, C. et al. *Visual Studio .Net, UML, and MSF*. New York: Apress. ISBN 9781590593684.

[17] ARLOW, J. a I. NEUSTADT. *UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky.* 1. vyd. Brno: Computer Press, 2007. ISBN 978-80-251-1503-9.

[18] PETZOLD, C. *Mistrovství ve Windows Presentation Foundation: [aplikace = kód markup]*. Brno: Computer Press, 2008. ISBN 978-802-5121-412.

[19] LIBERTY, J., J. GALLOWAY a P. JAPIKSE. *Windows 8.1 Development with XAML and C#*. New York: Apress, 2014. ISBN 978-1-4302-4048-8.

[20] WEIT, A. *Learn WPF MVVM -XAML,C# and the MVVM pattern,* New York: Lulu.com. ISBN 9781326847999.

[21] Live Charts. Documentation. *Lvecharts.cz* [online]. 2018 [cit. 2019-03-03]. Dostoupné z: https://lvcharts.net/App/examples/wpf/start

[22] LERMAN, J. a MILLER, R. *Programming_Entity_Framework,* Sebastopol, CA: O'Reilly. ISBN 978-1-449-31294-7.

[23] Learn Entity Framework Core. What is Entity Framework Core? *Learnentityframewerkcore.com* [online]. 2019 [cit. 2019-03-03]. Dostupné z: https://www.learnentityframeworkcore.com/

[24] JOHNNY. Klasické a agilní metodiky vývoje software. In: *My.cz* [online]. 2008 [cit. 2019-03-03]. Dostupné z: http://blog.nny.cz/740763-klasicke-a-agilni-metodiky-vyvoje-software.php

[25] MARKOVÁ, E. Řízení projektů. *Slideplayer.cz* [online]. 2018 [cit. 2019-03-03]. Dostupné z: https://slideplayer.cz/slide/2870812/

[26] SLOUP, M. Metodiky vývoje software – účel, obsah metodiky;příklad sekvenční, iterativní, agilní metodiky, CMMI. In: *Arcao.com* [online]. 2011 [cit. 2019-03-03]. Dostupné z: https://zcu.arcao.com/_statnice_ing_fav/nis/wiki_zvesela/06.htm

[27] Management Mania. Agilní metodiky řízení vývoje software. *Managementmania.com* [online]. 23. 12. 2016 [cit. 2019-03-03]. Dostupné z: https://managementmania.com/cs/agilni-metodiky-rizeni-vyvoje-software

[28] BECHNÝ, O. *Návrh metodiky vývoje softwaru z pohledu samostatného vývojáře.* Brno, 2010. Diplomová práce. Masarykova univerzita, Fakulta informatiky. Vedoucí práce Jan Pavlovič. Dostupné také z: https://is.muni.cz/th/bwr0t/thesis.pdf

[29] BĚHÁLEK, M. Kapitola 1. .NET Framework. In: *Vsb.cz* [online]. 2007 [cit. 2019-03-03]. Dostupné z: http://www.cs.vsb.cz/behalek/vyuka/pcsharp/text/ch01s01.html

[30] Rezervační systém pro vaše sportovní centrum. *Clubspire* [online]. 2016 [cit. 2019-05-16]. Dostupné z: https://www.clubspire.cz/o-produktu

[31] AFit - software pro vaše sportovní centrum. *AFit* [online]. 2008 [cit. 2019-05-16]. Dostupné z: http://www.emdat.cz/afit/

[32] MemberPRO - systém pro fitness, hotely, studia a kluby. *MemberPRO* [online]. 2004 [cit. 2019-05-16]. Dostupné z: http://www.memberpro.cz/popis_systemu.aspx

6 PŘÍLOHY

Příloha A – SQL se zdrojovými daty

```
INSERT INTO [Roles] ([Name]) VALUES ('Recepční');
GO
INSERT INTO [Roles] ([Name]) VALUES ('Manager');
GO
INSERT INTO [Roles] ([Name]) VALUES ('Skladník');
GO
INSERT INTO [Roles] ([Name]) VALUES ('Admin');
GO
INSERT INTO [Users]([Firstname],[Lastname],[Email],[Username]
           ,[Password],[Role_RoleId])
     VALUES('Admin','Admin','Admin','Admin','1f30DopfrVc6pkH8Z6LfQw==',4)
GO
INSERT [Users] ([Firstname], [Lastname], [Email], [Username], [Password],
[Role_RoleId]) VALUES (N'Jiří', N'Vrbas', N'vrbas@seznam.cz', N'vrbasji',
N'kKKC0h37k1zgEkZLN47KtA==', 4)
GO
INSERT [Users] ([Firstname], [Lastname], [Email], [Username], [Password],
[Role_RoleId]) VALUES (N'Karel', N'Abrahám', N'karlos@seznam.cz',
N'karlos', N'kKKC0h37k1zgEkZLN47KtA==', 1)
GO
INSERT [Users] ([Firstname], [Lastname], [Email], [Username], [Password],
[Role_RoleId]) VALUES (N'Alžběta', N'Holá', N'alba@seznam.cz', N'hoal',
N'kKKC0h37k1zgEkZLN47KtA==', 2)
GO
INSERT [Users] ([Firstname], [Lastname], [Email], [Username], [Password],
[Role_RoleId]) VALUES (N'Jan', N'Prokop', N'prokopjan@gmail.com', N'prja',
N'kKKC0h37k1zgEkZLN47KtA==', 3)
GO
INSERT [Tickets] ([Name], [Acitve], [TotalEntry], [TotalDays], [Discriminator],
[Price]) VALUES ( N'Individuální vstup', 1, 1, NULL, N'EntryTicket', CAST(60.00 AS
Decimal(18, 2)))
INSERT INTO [Customers]([Firstname],[Lastname],[Email]
,[Telephone],[StreetName],[HouseNumber],[City],[PostCode]
,[CardId]) VALUES ('Jednorázový','Jednorázový',null,null,null
,null,null,null,null)
```

Příloha B – Mapování tříd modelu

```
        public static TicketModel Map(Ticket ticket) {
            TicketModelType type;
            string totDays = null;
            string totEntry = null;
            if (ticket is SessionTicket) {
                type = TicketModelType.SessionTicket;
                totDays = (ticket as SessionTicket).TotalDays.ToString();
            }
            else {
                type = TicketModelType.EntryTicket;
                totEntry = (ticket as EntryTicket).TotalEntry.ToString();
            }
            return new TicketModel {
                Acitve = ticket.Acitve,
                Name = ticket.Name,
                TotalDays = totDays,
                TotalEntry = totEntry,
                TicketId = ticket.TicketId,
                TicketType = type,
                Price = ticket.Price };
        }
        public static Ticket Map(TicketModel tm) {
            if (tm.TicketType == TicketModelType.EntryTicket) {
                return new EntryTicket {
                    Acitve = tm.Acitve,
                    Name = tm.Name,
                    TotalEntry = Convert.ToInt32(tm.TotalEntry),
                    TicketId = tm.TicketId,
                    Price = tm.Price };
            }
            if (tm.TicketType == TicketModelType.SessionTicket) {
                return new SessionTicket {
                    Acitve = tm.Acitve,
                    Name = tm.Name,
                    TotalDays = Convert.ToInt32(tm.TotalDays),
                    TicketId = tm.TicketId,
                    Price = tm.Price };
            }
            return null;
        }
```

Příloha C – UML diagram balíčků