

# **Správa oprávnění vydávání klíčů v budovách**

Building Key Issue Authorisation Management System

Bc. Robert Hanzel

---

Diplomová práce  
2014



Univerzita Tomáše Bati ve Zlíně  
Fakulta aplikované informatiky

---

Univerzita Tomáše Bati ve Zlíně  
Fakulta aplikované informatiky  
akademický rok: 2013/2014

## ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Robert Hanzel**  
Osobní číslo: **A11482**  
Studijní program: **N3902 Inženýrská informatika**  
Studijní obor: **Informační technologie**  
Forma studia: **kombinovaná**

Téma práce: **Správa oprávnění vydávání klíčů v budovách**

Zásady pro vypracování:

1. Seznamte se s existujícími řešeními v dané oblasti.
2. Vypracujte seznam požadavků na řešení.
3. Proveďte návrh struktury systému.
4. Proveďte návrh pracovních procesů vč. využití elektronické komunikace.
5. Věnujte pozornost zabezpečení aplikace.
6. Realizujte všechny části navrženého systému.
7. Proveďte vyhodnocení přínosu systému.

Rozsah diplomové práce:

Rozsah příloh:

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

Seznam odborné literatury:

1. **MACDONALD, Matthew, Adam FREEMAN a Mario SZPUSZTA. ASP.NET 4 a C# 2010: tvorba dynamických stránek profesionálně. Vyd. 1. Překlad Jan Pokorný. Brno: Zoner Press, 2011, 880 s. Encyklopedie Zoner Press. ISBN 978-80-7413-131-8.**
2. **LACKO, L'uboslav. 1001 tipů a triků pro SQL. Vyd. 1. Brno: Computer Press, 2011, 416 s. ISBN 978-80-251-3010-0.**
3. **NASH, Trey. C# 2010: rychlý průvodce novinkami a nejlepšími postupy. Vyd. 1. Brno: Computer Press, 2010, 624 s. ISBN 978-80-251-3034-6**
4. **JQuery: kuchařka programátora. Vyd. 1. Brno: Computer Press, 2010, 436 s. ISBN 978-80-251-3152-7.**
5. **CASTRO, Elizabeth. HTML, XHTML a CSS: názorný průvodce tvorbou WWW stránek. Vyd. 1. Brno: Computer Press, 2007, 438 s. ISBN 978-80-251-1531-2.**
6. **PECINOVSKÝ, Rudolf. Návrhové vzory. Vyd. 1. Brno: Computer Press, 2007, 527 s. ISBN 978-80-251-1582-4.**
7. **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.**

Vedoucí diplomové práce:

**Ing. Petr Šilhavý, Ph.D.**

Ústav počítačových a komunikačních systémů

Datum zadání diplomové práce:

**21. února 2014**

Termín odevzdání diplomové práce:

**20. května 2014**

Ve Zlíně dne 21. února 2014

prof. Ing. Vladimír Vašek, CSc.  
*děkan*



doc. Mgr. Roman Jašek, Ph.D.  
*ředitel ústavu*

## **ABSTRAKT**

Program Správa oprávnění vydávání klíčů v budovách by měl sloužit k jednoduché a přehledné evidenci klíčů a jejich zápůjček. Typickým uživatelem navrženého systému mohou být recepce různých administrativních komplexů či budov. S pomocí programu je velmi snadné udržovat přehled o půjčených klíčích, zaznamenávat různé incidenty zápůjček, seskupovat osoby půjčující si klíče do konkrétních rolí, nebo si jednoduše administrovat přístup do vlastní kanceláře.

Klíčová slova: uživatel, budova, patro, místnost, klíč, zápůjčka, osoba, role, výjimka

## **ABSTRACT**

Program management authority issuing keys in buildings should be used for simple and easy registration keys and loans. Typical applications for this software is offered as a gatehouse various administrative complexes or buildings. With the help of the program is very easy to keep track of borrowed keys, record various incidents loans, personal lending group keys into specific roles or simply administer access to its offices.

Keywords: user, building, floor, room, key, loan, person, role, exception

Děkuji Ing. Petru Šilhavému, Ph.D. za odborné vedení mé diplomové práce

### **Prohlašuji, že**

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

### **Prohlašuji,**

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

Ve Zlíně 3. května 2014

.....  
podpis diplomanta

# OBSAH

<b>1</b>	<b>TEORETICKÁ ČÁST .....</b>	<b>9</b>
<b>1</b>	<b>ÚVOD .....</b>	<b>10</b>
<b>2</b>	<b>POPIS PROBLÉMU .....</b>	<b>12</b>
2.1	ZÁKLADNÍ FUNKCIONALITA APLIKACE.....	13
2.1.1	Evidované entity.....	13
2.1.2	Uživatelské účty .....	14
2.1.3	Role osob.....	15
2.1.4	Výjimky místnosti .....	15
2.1.5	Proces zapůjčování klíče .....	15
2.1.6	Práce s evidovanými objekty.....	16
2.2	ZABEZPEČENÍ APLIKACE .....	17
2.2.1	Mechanismy zabezpečení webové aplikace .....	18
2.2.1.1	Autentizace .....	18
2.2.1.2	Autorizace .....	19
2.2.1.3	Utajení a integrita dat.....	19
2.2.2	Možnosti zabezpečení ze strany webového serveru IIS .....	20
2.2.2.1	Autentizace IIS.....	20
2.2.2.2	Autorizace IIS .....	20
2.2.2.3	IIS a využití SSL .....	20
2.2.3	Možnosti zabezpečení ze strany ASP.NET.....	21
2.2.3.1	Bezpečnostní architektura ASP.NET.....	21
2.2.3.2	Autentizace Windows v ASP.NET.....	23
2.2.3.3	Formulářová autentizace v ASP.NET.....	26
2.2.3.4	Členství ASP.NET .....	28
2.2.3.5	Autorizace ASP.NET.....	30
<b>3</b>	<b>ANALYTICKÁ ČÁST .....</b>	<b>33</b>
3.1	POŽADAVKY.....	33
3.2	DIAGRAM TŘÍD .....	35
3.3	DIAGRAMY PŘÍPADŮ UŽITÍ.....	36
3.3.1	Aktéři.....	36
3.3.2	Evidence budov .....	36
3.3.3	Evidence pater .....	37
3.3.4	Evidence místností .....	38
3.3.5	Evidence klíčů .....	39
3.3.6	Evidence zápůjček.....	40
3.3.7	Evidence osob .....	41
3.3.8	Evidence rolí osob .....	42
3.3.9	Evidence výjimek .....	43
3.3.10	Evidence uživatelů .....	44
3.4	DIAGRAMY AKTIVIT.....	45
3.4.1	Evidence budov .....	45
3.4.2	Evidence zápůjček.....	52
3.4.3	Evidence výjimek .....	58

3.4.4	Evidence uživatelů .....	62
<b>II</b>	<b>PRAKTICKÁ ČÁST .....</b>	<b>69</b>
<b>4</b>	<b>ARCHITEKTURA APLIKACE.....</b>	<b>70</b>
<b>5</b>	<b>POPIS ŘEŠENÍ .....</b>	<b>73</b>
5.1	DATOVÁ VRSTVA.....	73
5.1.1	Tabulky.....	74
5.1.2	Uložené procedury .....	80
5.1.3	Uživatelské funkce .....	87
5.1.4	Uživatelské datové typy .....	87
5.2	VRSTVA WEBOVÉ SLUŽBY .....	88
5.2.1	Metody webové služby pro základní operace s daty .....	88
5.2.2	Použité datové třídy.....	92
5.3	KLIENSKÁ VRSTVA .....	98
5.3.1	Základní operace s daty .....	99
5.3.2	Databázové třídy.....	107
	<b>ZÁVĚR .....</b>	<b>114</b>
	<b>ZÁVĚR V ANGLIČTINĚ .....</b>	<b>115</b>
	<b>SEZNAM POUŽITÉ LITERATURY .....</b>	<b>116</b>
	<b>SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK .....</b>	<b>117</b>
	<b>SEZNAM OBRÁZKŮ .....</b>	<b>118</b>
	<b>SEZNAM TABULEK.....</b>	<b>123</b>

## **I. TEORETICKÁ ČÁST**

## 1 ÚVOD

Problematika evidence klíčů a jejich zápůjček je aktuální všude tam, kde dochází k častému pohybu více klíčů mezi různými osobami. Může se jednat o administrativní budovy, školy, různé úřady, nebo například také nemocnice. V těchto zařízeních se nachází společné uzamykatelné prostory (učebny, laboratoře, kanceláře, různé zasedací místnosti atp.), pro jejichž navštívení je potřeba vypůjčit si klíč k těmto prostorám a to zpravidla z nějakého centrálního úložiště – typicky vrátnice objektu.

Na těchto stanovištích většinou pracuje osoba, která má nad půjčovanými klíči dohled. Zajišťuje jejich vydávání, přijímání a také provádí určitou evidenci jak samotných klíčů, tak i jejich zápůjček. Bez použití specializovaného software se jedná o jednoduchý soupis klíčů a ruční zaznamenávání všech pohybů klíčů (zapůjčení, vrácení, ztráta). Evidence může být realizována pouze formou papírových poznámek, nebo může jít v lepším případě o určitý elektronický druh evidence s použitím například tabulkového procesoru.

V prvním případě, kdy se vše zapisuje na papírové médium, se jedná o nejjednodušší metodu, která v sobě také nese základní nedostatky. Papírové poznámky mohou být nepřehledné, nejsou odolné vůči chybným zápisům a rozhodně se vůbec nehodí pro tvorbu rychlých reportů a sestav konstruovaných z různých pohledů na data. Pokud bude někoho zajímat, které klíče jsou zrovna půjčené, nejrychlejší a nejspolehlivější způsob nespíš bude zkontrolovat vitrínu s klíči a zaznamenat si názvy neobsazených háčků po půjčených klíčích. Jestliže dostane obsluha evidenčního systému za úkol dodat informaci o všech osobách, které si v daném časovém intervalu půjčily určitý klíč, bude to pro ni znamenat projít postupně všechny záznamy od určitého místa a kontrolovat, zda se jedná o zápůjčku požadovaného klíče. Při delším časovém intervalu se bude jednat o časově náročný úkon s nejistým výsledkem, jelikož při hledání jednotlivých záznamů může obsluha snadno chybovat a některé záznamy přehlédnout. Získání téměř jakékoliv užitečné informace z papírové agendy bude problematické a zdlouhavé.

Lepším přístupem bezesporu bude využití elektronické evidence s použitím výše uvedené ho tabulkového procesoru. S tímto prostředkem lze vytvořit přehlednější agendu s možností rychlého filtrování záznamů. Stále se ovšem bude jednat o jednoduché řešení s minimální funkcionalitou. Tabulkový procesor bude jen stěží kontrolovat některé chybné uživatelské vstupy (vypůjčení již vypůjčeného klíče, zadání neexistujícího klíče nebo osoby, atd.). Takto pořízená evidence klíčů bude pouhou množinou záznamů bez těsnější

vazby mezi jednotlivými záznamy. Integrita takto pořizovaných dat je jen minimální a do značné míry závisí na přesném zadávání údajů do systému ze strany obsluhy. Tvorba výstupních sestav z takového systému je tedy snadnější než v předchozím případě papírové evidence, ale rozhodně neumožňuje nijak komplexní pohledy na pořízená data a výstupní sestavy odrážejí přesný a skutečný stav věcí také jen v případě, že byla data do systému bezchybně zadána.

Nejlepším řešením při pořizování systému evidence klíčů a zápůjček těchto klíčů se jeví sofistikovaný aplikační software. Odbourává nevýhody a nedostatky předešlých přístupů a přidává nemalé výhody. Základem takovéto aplikace bude datová základna na relační bázi, která bude udržovat pořizovaná data ve strukturované podobě a vždy v konzistentním stavu. Zajistí se tím rychlý přístup k požadovaným informacím s možností získávat rozmanité pohledy na data z hlediska modelovaných entit (budova, místnost, klíč) podle dostupných kritérií (čas, název, příslušnost k jiným entitám..). Při využití databázového serveru je možno vytvořit architekturu aplikace typu klient / server, která umožní centrální přístup k datům z mnoha klientských stanic. Což je podstatný rozdíl oproti předchozímu řešení s tabulkovým procesorem, kdy lze celou evidenci klíčů spravovat jen z jednoho místa, jelikož jsou data ukotvena lokálně na konkrétním počítači. Při zavedení webových technologií je pak možné na straně klienta využívat k přístupu k datům místo instalované klientské aplikace obyčejný webový prohlížeč a celé řešení provozovat na místní počítačové síti.

Předmětem této diplomové práce je tvorba aplikace pro evidenci klíčů a jejich zápůjček, která využívá zmíněné webové technologie ve spolupráci s databázovým serverem pro dosažení výrazné flexibility při nasazení a samotném provozu aplikace. Součástí řešení je i přidání další vrstvy aplikace ve formě webové služby, která do budoucna umožní tvorbu rozmanitých platformově navzájem rozdílných klientů. Kromě webové aplikace bude tedy možno doprogramovat další typy klientů evidenčního systému – například mobilního klienta, nebo desktopové klienty běžící na různých OS – Windows, Linux.

Při tvorbě řešení byly aplikovány znalosti použitých technologií, které autor čerpal z následujících zdrojů: tvorba webových aplikací v ASP.NET [1], dotazovací databázový jazyk SQL [2], programovací jazyk C# [3], tvorba klientských skriptů v JQuery [4], využití HTML a CSS při tvorbě webových aplikací [5], využití návrhových vzorů při návrhu a realizaci aplikací [6], využití UML při návrhu aplikací [7].

## 2 POPIS PROBLÉMU

Existující systémy řešící problematiku evidence klíčů a správu jejich zápůjček jsou různorodé aplikace lišící se jak svou architekturou, tak i způsobem a úrovní zpracování. Měl jsem možnost blíže se seznámit se třemi aplikacemi tohoto druhu.

Software Evva [8] je starší aplikací postavenou nad souborovou databází Access a běžící na platformě Windows. Jedná se o obyčejnou desktopovou aplikaci neumožňující nasazení v síťovém prostředí. Program eviduje základní entity, spadající do problematiky správy klíčů, jako jsou budovy, místnosti, klíče, nebo osoby půjčující si jednotlivé klíče. Lze zde vytvářet různé sestavy z dostupných pohledů na jednotlivé entity a takto pořízené informace program umožňuje exportovat ve formátu .xls.

Fama+ [9] od společnosti Tesco SW je aplikace pro facility management, jejíž součástí je také modul pro správu a evidenci klíčů. V tomto případě je jádro modulu univerzálně navržené s možností relativně bohaté customizace. Modul je plně integrován do celého rámce aplikace a umožňuje vzájemnou komunikaci se souvisejícími moduly. Aplikace je dodávána ve dvou možných provedeních. „Těžká“ verze je klasický klient / server pro Windows, využívající dle přání zákazníka databázový server Oracle nebo SQL Server. Oproti tomu je „lehká“ verze klasickou webovou aplikací používající na klientské straně technologii Silverlight. Modul zpracovává danou problematiku velmi detailně, eviduje podrobnosti typu zámkové vložky dveří, v rámci místnosti operuje s více dveřmi, zavádí pojem univerzálních klíčů a kromě samotných klíčů je schopen evidovat i jiné druhy zapůjčovaných objektů, například parkovací a čipové karty. Součástí modulu je i realizované rozhraní pro komunikaci s jinými softwarovými systémy.

Správa klíčového hospodářství [10] je software od společnosti Yamaco, který je dostupný k užívání zdarma. Platí se jen za systémovou podporu produktu. Architektura aplikace je také typu klient / server a databázové služby zabezpečuje databázový server Firebird. Eviduje podobné základní entity jako aplikace Evva, se širší škálou možných grafických nebo tabulkových sestav. Program umí s osobami v systému komunikovat prostřednictvím SMS zpráv a obsažená data lze importovat do systému nebo exportovat z něj prostřednictvím různých běžných souborových formátů.

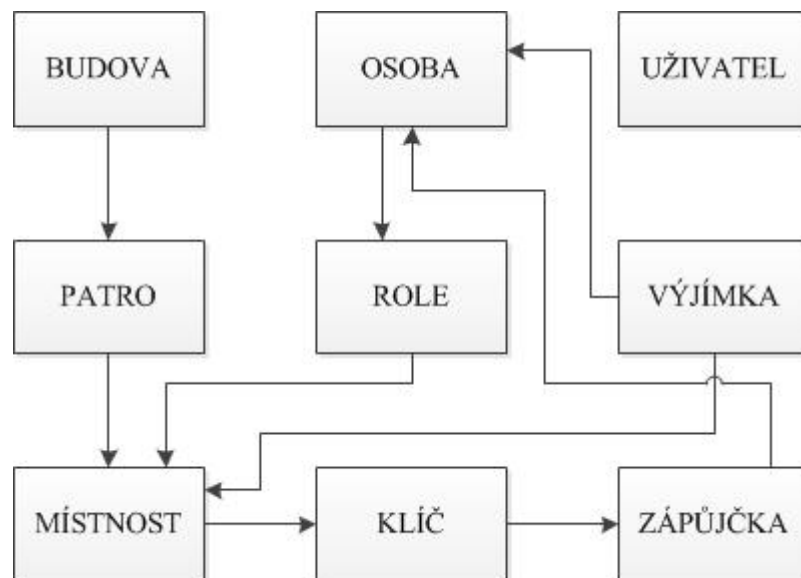
Program pro správu klíčů, který je předmětem této diplomové práce, si neklade za cíl pokrýt zpracovávanou problematiku do nejmenších detailů, umožnit tvorbu velkého

množství podrobných grafických výstupních sestav, nebo poskytovat další pokročilé funkcionality v podobě podpory rozličných importních a exportních procesů. Mělo by se jednat o základní evidenční systém, umožňující přehlednou a rychlou evidenci klíčů a zápůjček. Především by měl být kladen důraz na jednoduchost. V budoucnu bude samozřejmě možné aplikaci případně programově rozšířit o další požadované funkcionality. Součástí stávajícího řešení bude kromě evidence základních entit například možnost seskupovat osoby do různých rolí, nebo existence primární kontroly přístupu do místnosti ze strany jejího vlastníka. Na data bude mít uživatel možnost nahlížet z perspektivy jednotlivých entit a s využitím různých filtrů selektovat úzce specifické informace. Aplikace bude koncipována jako webová, spolupracující s databázovým serverem SQL Server. Velkou předností systému ovšem bude realizace webové služby, která bude komunikovat s databázovou vrstvou a poskytovat univerzální rozhraní směrem ke klientské vrstvě systému. Kromě klienta realizovaného formou webové aplikace v rámci této diplomové práce, tak bude možné do budoucna k systému doprogramovat libovolně další typy klientů, využívajících odlišné technologie, nebo dokonce běžících na různých platformách.

## **2.1 Základní funkcionality aplikace**

### **2.1.1 Evidované entity**

Aplikace bude provádět evidenci a správu základních entit souvisejících s řešeným problémem. Hierarchicky seřazené jsou objekty jednotlivých budov, pater, místností, klíčů a konečně zápůjček jednotlivých klíčů. Systém bude též evidovat objekty osob, půjčujících si klíče. Osobám budou přiřazovány jedna nebo více rolí, které budou nést informaci o místnostech, ke kterým osoba díky přiřazené roli získává oprávnění k zapůjčování klíčů. Výjimka místnosti je entita, která mimo klasická oprávnění přes role zavádí prioritní povolení nebo zákaz k vypůjčení klíčů určité místnosti pro konkrétní osobu na libovolně dlouhý časový úsek. Posledním typem evidované entity je uživatel přihlašující se do systému.



Obr. 2.1 - Evidované entity aplikace

### 2.1.2 Uživatelské účty

Z uživatelského hlediska bude aplikace podporovat 3 typy uživatelských účtů s různými kompetencemi. Administrátor bude mít oprávnění pro úplnou správu celého systému, kromě řízení prioritního přístupu k místnostem vlastníků. Obsluze zápůjček bude umožněno spravovat zápůjčky, klíče a osoby. Vlastník místnosti může ovládat přístup k místnostem, k nimž byl přiřazen administrátorem jako jejich vlastníků. Jedná se o správu výjimek k těmto místnostem. Zbytek systému se stává pro tuto roli nedostupným. Stejně tak zbylé role nebudou mít přístup do sekce vlastníků místnosti. Každý uživatel může mít přiřazeny všechny existující role.



Obr. 2.2 – Uživatelské role

### 2.1.3 Role osob

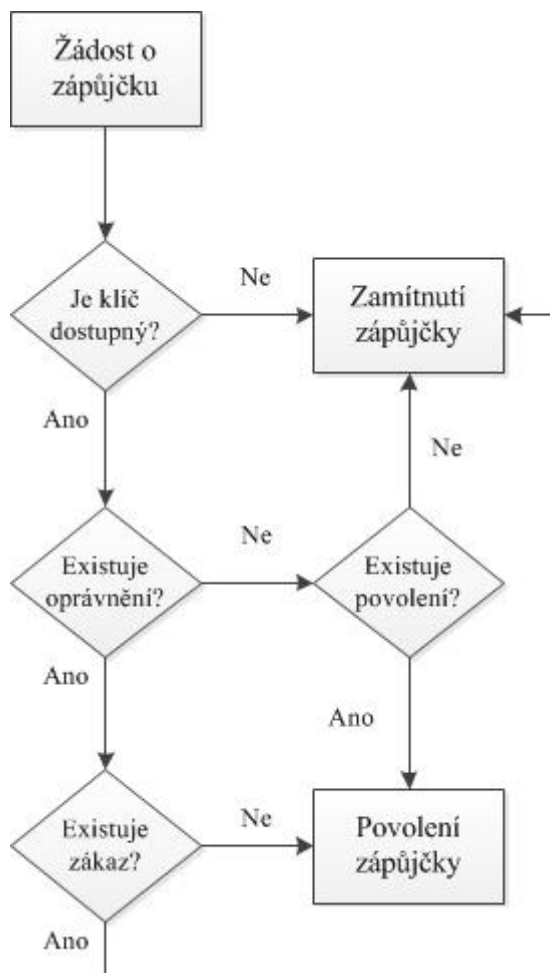
Každá osoba zanesená v systému, musí mít přiřazenu alespoň jednu roli, jinak nemá oprávnění k vypůjčení žádného klíče. Roli zakládá a případně upravuje administrátor aplikace a jedná se o seznam místností, ke kterým má osoba s touto rolí oprávnění k vypůjčení klíče. Osobě lze přiřadit libovolné množství existujících rolí.

### 2.1.4 Výjimky místnosti

Výjimka místnosti je pravidlo, které zavádí prioritní přístup k místnosti pro danou osobu. Může se jednat o povolení, nebo zákaz. Výjimka má časovou platnost. Tento druh entity zakládá a spravuje jedině vlastník místnosti, ke které je vztažena. Výjimka tedy umožňuje po určitý časový úsek rušit oprávnění osoby k místnosti, vyplývající z její role.

### 2.1.5 Proces zapůjčování klíče

Při požadavku o zapůjčení klíče určité osobě proběhne nejdříve kontrola, zda je klíč volný a dostupný k zapůjčení. Pokud je momentálně vypůjčený, nebo není k dispozici z jiných důvodů, jako je třeba ztráta nebo poškození, bude tato skutečnost obsluze oznámena a operace zapůjčení klíče nebude uskutečněna. Při dostupném klíči přijde na řadu kontrola oprávnění osoby k zapůjčení klíče. Postupně se projdou všechny role osoby a zkontroluje se přítomnost místnosti, ke které náleží zapůjčovaný klíč, alespoň v jedné z nich. Jestliže se místnost nevyskytuje ani u jedné role, znamená to absenci oprávnění k vypůjčení klíče. Potom ještě ale přijde na řadu kontrola výjimek místnosti daného klíče. Jestliže je nalezen záznam pro danou osobu, přítomný čas a jedná se o pravidlo povolovací, osoba tímto získává oprávnění k zapůjčení klíče. V opačném případě nelze osobě klíč vydat. Analogicky v momentě, kdy na základě příslušnosti k roli osoba získá oprávnění k zápůjčce, proběhne kontrola výjimek a jestliže pro osobu existuje omezující pravidlo, o oprávnění k zápůjčce tímto přichází. Pokud se potřebná výjimka nenajde, osoba má definitivně možnost klíč k zapůjčení získat.



Obr. 2.3 – Proces zapůjčení klíče

### 2.1.6 Práce s evidovanými objekty

S jednotlivými objekty, které představují instance evidovaných entit (budova, patro..) bude možno v aplikaci provádět tyto základní typy operací:

- **Zobrazení seznamu objektů**

Operace, která zobrazí seznam všech evidovaných objektů vybraného typu. Seznam bude možno různě třídit nebo filtrovat. Buď podle příslušnosti k nadřazeným objektům, nebo na základně hodnot svých atributů.

- **Zobrazení detailu objektu**

Výběrem ze seznamu bude umožněno zobrazení detailní informace vybraného objektu. Kromě základních údajů zobrazovaných o objektu v seznamu, zde můžou být přidány doplňující informace, například o počtu podřízených objektů, atd.

- **Zobrazení seznamu podřízených objektů**

Z detailu objektu lze provést operaci zobrazení seznamu všech podřízených objektů daného objektu.

- **Založení nového objektu**

Při zobrazeném seznamu podřízených objektů, bude uživateli umožněno založit nový objekt, který se automaticky naváže na nadřazený objekt, k němuž patří zobrazený seznam. Před uložením do databáze bude zkontrolováno, zda jsou všechny nově zadané hodnoty atributů objektu validní.

- **Odstranění objektu**

Z detailu objektu lze tento objekt odstranit z evidence. Zároveň s ním budou po odsouhlasení upozornění odstraněny ze systému všechny související, podřízené objekty.

- **Editace objektu**

Z detailu objektu lze tento objekt editovat, tedy měnit hodnoty jeho atributů. Před uložením údajů bude zkontrolováno, zda jsou všechny změněné hodnoty atributů objektu validní.

## 2.2 Zabezpečení aplikace

Bezpečnost vytvářeného software je samozřejmě důležitou složkou vývoje jakéhokoliv druhu aplikace, nicméně v případě webových aplikací požadavek na důkladnou implementaci bezpečnostních mechanismů ještě vzrůstá. V aplikaci správy klíčů budou použity všechny základní postupy pro zaručení jejího bezpečného provozu v reálném prostředí Internetu. Uživatelská hesla, která se ukládají do databáze, budou zabezpečena proti neoprávněnému přečtení metodou tzv. hashování. Síťová komunikace mezi klientem a webovou službou poběží na zabezpečeném komunikačním kanále, v tomto případě se využije šifrovacích mechanismů v podobě digitálních certifikátů. Obranou proti útoku typu *SQL Injection* je omezení přístupu k datům databáze výhradně přes volání uložených procedur, kdy se potřebné operace s daty uskutečňují na základě předávání potřebných hodnot do vstupních parametrů jednotlivých procedur. Při práci s klientskou webovou aplikací bude každý uživatel podléhat procesům autentizace a autorizace. Chráněné zdroje budou moci využívat jen oprávnění uživatelé.

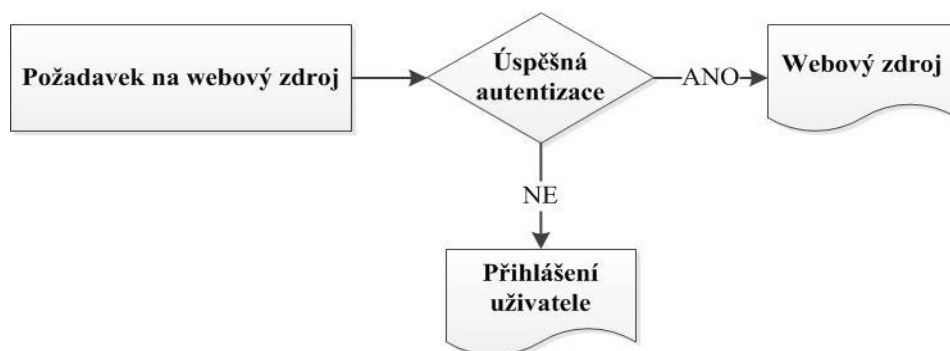
Pro bližší představu o této problematice, se bude další text věnovat všeobecně možnostem zabezpečení webových aplikací, s určitým zaměřením na ASP.NET, jelikož právě tato technologie byla zvolena pro realizaci klientské části aplikace pro správu klíčů.

### 2.2.1 Mechanismy zabezpečení webové aplikace

Při vývoji webové aplikace patří otázka bezpečnosti mezi ty nejvíce stěžejní. Bez řádného zabezpečení se aplikace stává pro své uživatele nebezpečnou – jejich důležitá data mohou být odcizena a zneužita neoprávněnými osobami, čímž jim mohou vzniknout nemalé škody. Aplikace bez vážných bezpečnostních rizik by rozhodně měla obsahovat implementaci základních bezpečnostních mechanismů, jako jsou autentizace, autorizace, utajení a integrita dat.

#### 2.2.1.1 Autentizace

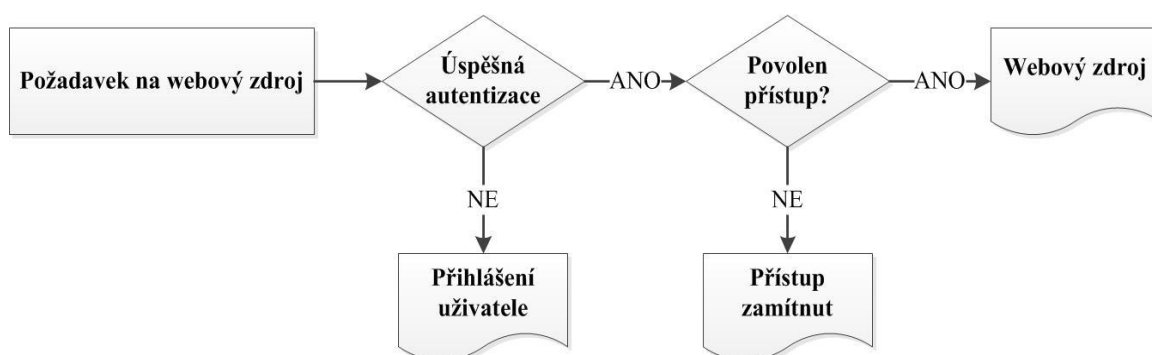
Autentizace je proces zjišťující identitu jednotlivých uživatelů aplikace. Bez řádně udělené identity není uživateli umožněn přístup do těch částí aplikace, které jsou k dispozici výhradně autentizovaným uživatelům. Princip autentizace probíhá v několika krocích. Nejprve neautentizovaný uživatel realizuje požadavek na webový server. Následně je vyzván k zadání svých přihlašovacích údajů skládajících se ze jména a hesla. Poskytnuté přihlašovací doklady uživatele jsou ověřeny webovou aplikací, nebo samotným webovým serverem, v závislosti na druhu použité autentizace. Pokud jsou přihlašovací údaje platné, je uživateli umožněn přístup k požadovanému webovému zdroji. [1]



Obr. 2.4 - Přístup k chráněnému webovému zdroji vyžadujícímu autentizaci

### 2.2.1.2 Autorizace

Tento mechanismus zajišťuje jednotlivým uživatelům přidělení přístupových práv k určitým operacím, nebo chráněným zdrojům aplikace, v závislosti na jejich identitě. Svůj zdroj můžeme zpřístupnit pouze pro vybrané uživatele. Vytvoří se pro tento účel seznam uživatelů, nebo skupin uživatelů, kteří mají právo přistupovat k vybranému zdroji a identita uživatele je potom porovnávána se seznamem. Pokud se najde shoda, je zřejmé, že uživatel má právo použít zdroj a přístup je mu umožněn. V opačném případě je uživateli přístup odepřen. [1]



Obr. 2.5 - Přístup k chráněnému webovému zdroji vyžadujícímu autorizaci

### 2.2.1.3 Utajení a integrita dat

Utajením dat rozumíme zajištění jejich ochrany před nežádoucím přečtením cizím subjektem. K tomu může dojít během výměny dat mezi serverem a klientem, nebo při přístupu do datového úložiště na serveru. K zajištění utajení dat se využívá šifrovacích technik. [1]

Integrita dat nám slouží pro ověření skutečnosti, že data nebyla během přenosu mezi serverem a klientem nikým měněna. Digitální podpisy postavené na asymetrické kryptografii, zajišťují právě tento cíl. [1]

## 2.2.2 Možnosti zabezpečení ze strany webového serveru IIS

Webový server představuje prvotní bezpečnostní bránu při přístupu k webové aplikaci. Dříve, než požadavek zpracuje runtime ASP.NET, je obslužen webovým serverem. Lze jej konfigurovat pro plnění funkcí autentizace, autorizace a důvěrnosti dat.

### 2.2.2.1 Autentizace IIS

IIS podporuje několik druhů autentizace. Jedná se o autentizaci Windows, Základní autentizaci, Digestní autentizaci, Passportovou autentizaci, a konečně Certifikátovou autentizaci prostřednictvím SSL. Výsledkem jakéhokoliv z uvedených autentizačních procesů je autentizovaný uživatel Windows. Uživatel vykonávající požadavek tedy musí mít platný účet na počítači webového serveru, či v rámci domény Windows. Pokud aplikace nevyužívá autentizaci Windows, ale například formulářovou autentizaci, musí být server nakonfigurován pro umožnění anonymního přístupu. [1]

### 2.2.2.2 Autorizace IIS

IIS umožňuje nakonfigurovat přístup k webu jenom vybraným strojům. Omezení se vztahuje na určité IP adresy. Toto řešení má smysl jen v případě, že je opravdu žádoucí, aby mělo přístup na server povoleno jen několik vybraných uživatelů. [1]

### 2.2.2.3 IIS a využití SSL

HTTP komunikace mezi serverem a klientem může být realizovaná pomocí protokolu SSL. Jde o mechanismus vytvářející zabezpečený komunikační kanál. URL takto chráněných zdrojů pak začínají prefixem `https://` místo obvyklého `http://` a data putují přes standardní port 443 oproti portu 80 využívaného pro nechráněnou komunikaci. SSL využívá k utajení dat asymetrické šifrování v kombinaci se symetrickým šifrováním. Asymetrickou šifrou je přenášen mezi serverem a klientem symetrický šifrovací klíč využívaný pro šifrování samotné komunikace. [1]

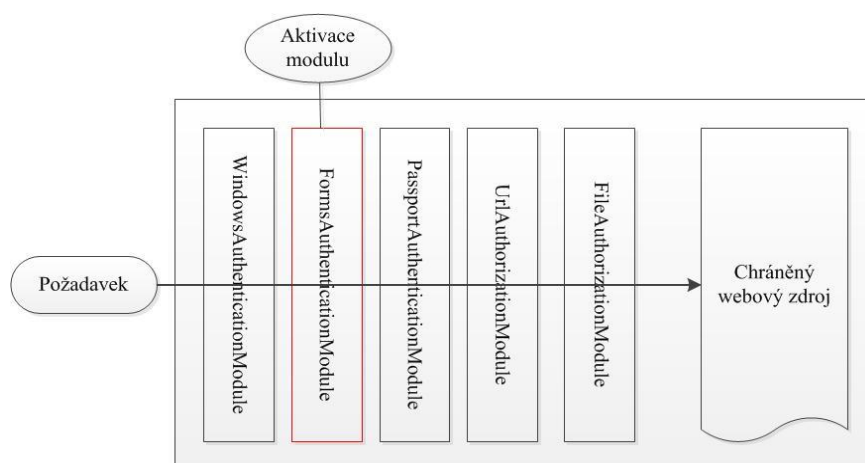
Aby mohl IIS podporovat komunikaci prostřednictvím SSL, musí mít nainstalován certifikát X.509. Implementace SSL tak spočívá v nainstalování tohoto certifikátu a příslušné konfiguraci serveru IIS.

### 2.2.3 Možnosti zabezpečení ze strany ASP.NET

V předchozí kapitole byly nastíněny možnosti zabezpečení v rámci webového serveru. Mechanismus zabezpečené komunikace je doménou právě jenom serveru IIS. Runtime ASP.NET o jeho existenci vůbec netuší. Naproti tomu autentizace je záležitostí jak webového serveru, tak i ASP.NET. Do jaké míry, to záleží na zvoleném druhu této autentizace. Formulářová autentizace je plně v kompetenci ASP.NET, zatímco různé druhy autentizace Windows pracují tak, že samotnou autentizaci provádí server a následně potom předává informace o identitě uživatele do ASP.NET. Implementace autorizačních mechanismů nám potom zajistí pro každého uživatele zpřístupnění jen těch zdrojů, na které má nárok.

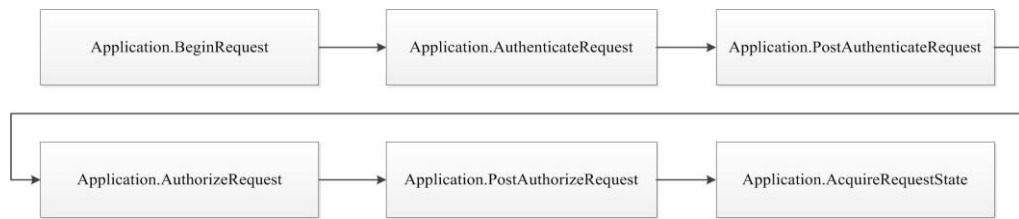
#### 2.2.3.1 Bezpečnostní architektura ASP.NET

Z důvodu bezstavové povahy protokolu http je potřeba vykonávat všechny operace související s autentizací a autorizací při každém požadavku na server. Uživatel tedy musí být autentizován a autorizován pokaždé znovu. ASP.NET se s tímto nedostatkem vyrovnává tím, že odpaluje globální aplikační události, na které jsou schopny reagovat http moduly, určené pro různé druhy autentizace a autorizace. Tyto moduly potom aktivují samotné procesy autentizace či autorizace. [1]



Obr. 2.6 – Autentizační a autorizační moduly ASP.NET

Nejdůležitější události odpalované v souvislosti se zabezpečením aplikace jsou *AuthenticateRequest* a *AuthorizeRequest*. [1]



Obr. 2.7 - Aktivace událostí zabezpečení ASP.NET

Druh použité autentizace v aplikaci lze nastavit výběrem patřičného autentizačního modulu úpravou kořenového konfiguračního souboru aplikace web.config v sekci authentication:

```
<authentication mode="[Forms/Windows/Passport]" />
```

- **FormsAuthenticationModule**

Modul pro formulářovou autentizaci, kterou lze realizovat vlastním návrhem, nebo ji lze také implementovat pomocí API pro členství.

- **WindowsAuthenticationModule**

Tento modul spolupracuje na procesu autentizace Windows s webovým serverem.

- **PassportAuthenticationModule**

Modul poskytující podporu pro autentizační službu Microsoft Passport. Pro její provoz je potřeba instalovat Passport.Net SDK.

V okamžiku, kdy je uživatel autentizován, jsou informace týkající se jeho účtu předány infrastruktuře ASP.NET. K těmto informacím lze v programovém kódu přistupovat prostřednictvím objektu *HttpContext.Current.User*. A využít je lze k realizaci své vlastní autorizace přímo v kódu aplikace. Nicméně ASP.NET nabízí pro implementaci autorizace také následující moduly:

- **UrlAuthorizationModule**

Autorizace pracující na základě tohoto modulu funguje prostřednictvím prvků `<authorization>`, které mohou být obsaženy na rozdíl od prvků `<authentication>` v

konfiguračních souborech každého z adresářů aplikace. Určují, do kterých adresářů a ke kterým souborům má aktuální uživatel umožněn přístup.

- **FileAuthorizationModule**

Tento modul ASP.NET automaticky používá v případě, že je aktivovaná autentizace Windows. Princip spočívá v kontrole přístupových práv uživatelů Windows k jednotlivým adresářům a souborům, které jsou přístupné prostřednictvím ASP.NET. Jako kontrola zabezpečení zde fungují nastavení ACL Windows. V tomto případě je nutností, aby měl každý uživatel přístupová práva pro čtení ke všem souborům webové aplikace.

Důležitým prvkem v rámci bezpečnostní architektury ASP.NET je bezpečnostní kontext. Jedná se o objekt obsahující název aktuálního uživatele spolu s informacemi spjatými s jeho účtem, jako je např. název role, do které patří. Tento objekt je přístupný z aktuálního http kontextu aplikace pomocí vlastnosti *User* objektu *HttpContext.Current*. Jedná se o objekt implementující rozhraní *IPrincipal*. *IPrincipal* obsahuje vlastnost *Identity*, která poskytuje informace o identitě uživatele. Jedná se o objekt implementující rozhraní *IIdentity*. Používané vlastnosti tohoto rozhraní jsou:

- **AuthenticationType**

Vrací typ používané autentizace ve formě řetězce.

- **IsAuthenticated**

Vrací hodnotu typu bool podle toho, zda je uživatel autentizován, či nikoli

- **Name**

Vrací jméno aktuálního uživatele ve formě řetězce.

### 2.2.3.2 Autentizace Windows v ASP.NET

Nasazení autentizace Windows má svůj význam v případech, kdy vytváříme aplikaci pro relativně méně uživatelů, typicky v prostředí intranetu. V tomto případě není potřeba implementovat nový autentizační systém. Existující uživatelé již mají vytvořeny své účty

Windows, většinou doménové, a autentizace Windows umožňuje použití těchto účtů. Vlastní autentizaci přenechává ASP.NET ke zpracování webovému serveru. Webový server požaduje přihlašovací doklady po prohlížeči, který je poskytne a tyto jsou poté namapovány na uživatelský účet Windows. Pokud je autentizace úspěšná, povolí server uživateli přístup k požadované stránce a informace o uživateli a roli jsou předány do runtime ASP.NET. Na jejich základě potom ASP.NET vytváří aktuální bezpečnostní kontext. [1]

IIS umožňuje použít jeden z následujících autentizačních mechanismů autentizace Windows:

- **Základní autentizace**

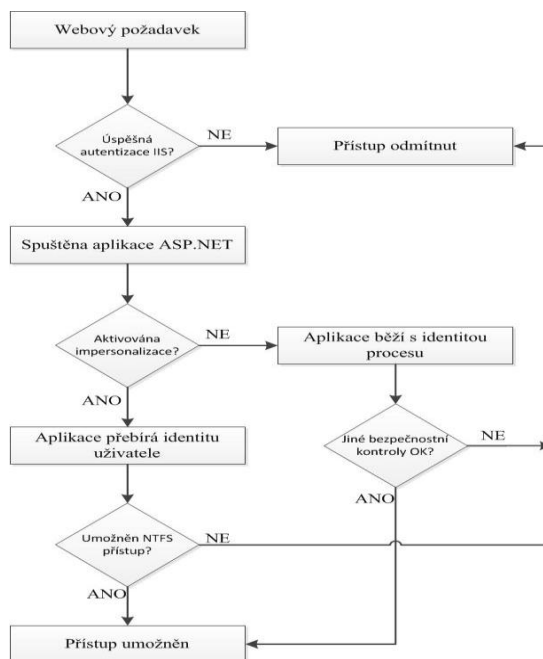
Přihlašovací údaje jsou zasílány ve formě prostého textu. Jedná se o standard jazyka HTML, který je podporován všemi prohlížeči.

- **Digestní autentizace**

Přihlašovací údaje jsou zasílány v hešované formě, takže se jedná o zabezpečený přenos.

- **Integrovaná autentizace Windows**

Identita uživatele aktuálně přihlášeného k Windows je přenášena ve formě tokenu, který prohlížeč poskytuje automaticky, bez zásahu uživatele.



Obr. 2.8 - Proces autentizace Windows

Pro zprovoznění autentizace Windows je potřeba vykonat několik kroků. Nejdříve se musí provést nastavení zabezpečení virtuálního adresáře aplikace prostřednictvím IIS. Stačí zvolit jednu z výše uvedených možností autentizace Windows. Pokud se rozhodneme, že přihlašování uživatelů nebudeme požadovat, nastavíme pro virtuální adresář možnost anonymního přístupu.

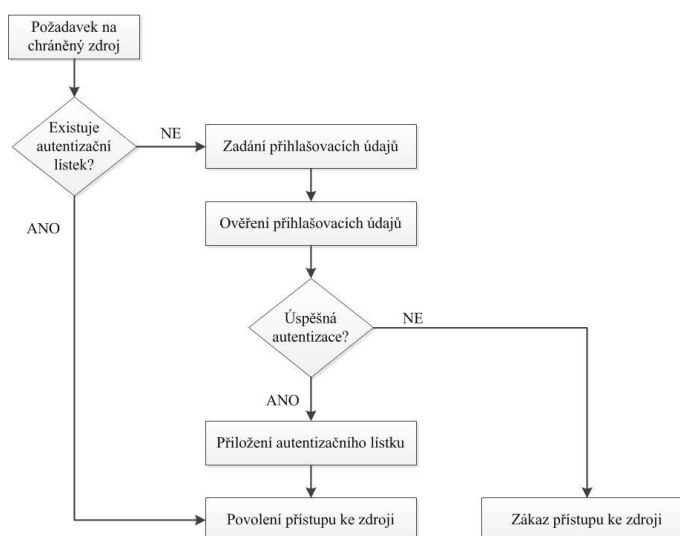
V dalším kroku je ještě nutné v konfiguračním souboru webové aplikace nastavit prvku `<authentication>` atribut `mode` na hodnotu "Windows". Tím je umožněna vzájemná spolupráce ASP.NET a IIS na realizaci autentizace Windows.

Jak již bylo dříve uvedeno, pro řádně autentizovaného uživatele zřizuje runtime ASP.NET bezpečnostní kontext. Jedná se o objekt implementující rozhraní *IPrincipal*. Identita uživatele je pak přístupná přes vlastnost *Identity*, která je instancí typu *IIdentity*. V případě autentizace Windows se jedná o instance konkrétních tříd *WindowsPrincipal* a *WindowsIdentity*.

### 2.2.3.3 Formulářová autentizace v ASP.NET

Pokud není žádoucí, aby webová aplikace pracovala s uživatelskými účty Windows, je potřeba použít formulářovou autentizaci. Musí se vytvořit přihlašovací formulář, do kterého budou uživatelé zadávat svá přihlašovací pověření, a tyto údaje budou potom porovnávány s údaji uloženými ve vlastním úložišti – typicky databázi. Po úspěšném přihlášení je potom potřeba zřídit aktuální bezpečnostní kontext. ASP.NET poskytuje pro implementaci formulářové autentizace kompletní infrastrukturu.

Formulářová autentizace využívá tzv. lístků (tickets). Jedná se o šifrovanou cookie s uživatelskými informacemi, která je klientovi zasílána pro pozdější použití. Pokud uživatel přistupuje ke zdroji vyžadujícímu autentizaci a není ještě řádně autentizován, runtime ASP.NET jej přesměruje na přihlašovací stránku. Po zadání a ověření přihlašovacích údajů je vytvořena a přiložena cookie s lístkem a uživatel je přesměrován na původně požadovanou stránku. S dalším požadavkem již klient přikládá obdržovaný lístek a na jeho základě je mu již povolen přístup ke chráněnému zdroji. [1]



Obr. 2.9 - Proces formulářové autentizace

Nejdůležitější prvek formulářové autentizace je http modul *FormsAuthenticationModule*, který umí v požadavku detekovat přítomnost autentizačního lístku. Pokud lístek není přiložen, modul automaticky uživatele přesměruje na přihlašovací stránku. Pokud modul lístek detekuje, vytvoří pro aktuálního uživatele bezpečnostní kontext. Základní třídy modulu jsou:

- **FormsAuthentication**

Hlavní třída pro práci s infrastrukturou formulářové autentizace. Má na starosti vytváření autentizačního lístku, spravuje cookie a přesměrovává uživatele z přihlašovací stránky na stránku původně požadovanou.

- **FormsAuthenticationTicket**

Tato třída reprezentuje autentizační lístek.

- **FormsIdentity**

Třída identity uživatele v systému formulářové autentizace.

Pro aktivaci formulářové autentizace je potřeba provést konfiguraci souboru `web.config` webové aplikace. Jedná se o prvek `forms` sekce `authentication`. K přihlášení uživatele je nutné vytvořit webový formulář pro zadání přihlašovacích údajů. Jedná se o jednoduchý formulář s textovými poli pro jméno, heslo a s tlačítkem pro spuštění autentizačního procesu.

Jako úložiště přihlašovacích údajů může sloužit přímo konfigurační soubor `web.config`, nebo je možné vytvořit své vlastní úložiště, typicky databázi SQL Serveru. V prvním případě se jedná o řešení pro aplikaci s nemnoha uživateli, při vyšším počtu je vhodnější použít databázi. Hesla ukládaná do konfiguračního souboru je možno hešovat.

V tlačítku pro přihlášení uživatele musí být kód pro ověření uživatele a jeho následné přesměrování na požadovanou stránku, pokud přihlášení proběhlo v pořádku. Pokud použijeme jako úložiště konfigurační soubor, pro ověření lze použít metodu `Authenticate()` třídy `FormsAuthentication`. Při použití vlastního úložiště si musí programátor ověřovací metodu vytvořit sám. Pro přesměrování uživatelů pak slouží metoda `RedirectFromLoginPage()` ze stejné třídy. Tato metoda vytváří pro uživatele autentizační lístek, šifruje informace z lístku, vytváří cookie pro uchování informací z tohoto lístku, cookie připojuje k http odpovědi a konečně přesměrovává uživatele na cílovou stránku. Odhlášení uživatele se provádí voláním metody `FormsAuthentication.SignOut()`. Poté je vhodné uživatele přesměrovat na přihlašovací stránku metodou `FormsAuthentication.RedirectToLoginPage()`.

### 2.2.3.4 Členství ASP.NET

Členství je pracovní rámec, který je strukturálně nadřazený infrastruktuře formulářové autentizace. Zapojení tohoto rámce přináší nemalé výhody. Nemusí se implementovat přihlašovací formuláře, ani vlastní úložiště přihlašovacích údajů. Schematicky se dá členství popsat následovně:



Obr. 2.10 - Struktura členství ASP.NET

Na vrcholu členství se nachází ovládací prvky pro bezpečnost, které řeší důležité zabezpečovací funkce a to zcela automaticky. Pro použití nabízených funkcí stačí přetáhnout vybraný prvek na webovou stránku a případně upravit jeho vzhled pomocí vestavěných šablon. Jedná se o prvky pro přihlašování, pro práci s hesly, průvodce pro vytvoření uživatele a stavové ovládací prvky.

API pro členství tvoří komponenty ve formě tříd, které při své práci využívají ovládací prvky pro bezpečnost. Na druhé straně je schopno API komunikovat s poskytovateli členství. Skládá se z následujících tříd:

- **Membership**

Poskytuje metody pro správu uživatelů, jejich ověřování a obnovu jejich hesel.

- **MembershipCreateUserException**

Výjimka, která je vyvolána v případě, když dojde k chybě během vytváření nového uživatele pomocí třídy *Membership*.

- **MembershipUser**

Obsahuje informace o jednotlivých uživateli.

- **MembershipUserCollection**

Kolekce prvků typu *MembershipUser*.

Poskytovatelé členství představují abstraktní vrstvu v aplikaci, která umožňuje, aby aplikace nebyla závislá na konkrétní implementaci datového úložiště. Poskytovatel realizuje přístup k datovému úložišti. Následující třídy se vztahují k poskytovatelům členství:

- **MembershipProvider**

Třída, z které se odvozují vlastní poskytovatelé členství.

- **MembershipProviderCollection**

Kolekce členů typu *MembershipProvider*.

- **SqlMembershipProvider**

Implementace poskytovatele na bázi SQL Serveru.

- **ActiveDirectoryMembershipProvider**

Poskytovatel Active Directory.

Samotná úložiště mohou být postavena na podporovaných technologiích SQL Serveru a Active Directory, ale programátor má možnost vytvořit si své vlastní úložiště za předpokladu, že k němu také vytvoří odpovídajícího poskytovatele členství. Pro zprovoznění celé infrastruktury je nutné nejdříve konfigurovat formulářovou autentizaci v souboru *web.config* a odepřít přístup anonymním uživatelům. Dále se musí připravit úložiště dat. Pokud se rozhodneme pro databázi SQL Serveru, lze použít nástroj .Net Frameworku *aspnet\_regsql.exe*, který vytvoří potřebné databázové tabulky. Pokud je úložiště připravené, je nutné dále nastavit v souboru *web.config* připojovací řetězec k databázi a použitého poskytovatele členství. Do připraveného úložiště je ještě potřeba přidat nové uživatele. Lze je vytvářet aplikačně pomocí ovládacího prvku *CreateUserWizard*, nebo lze použít administrační nástroj Visual Studia WAT. Nakonec je ještě potřeba vytvořit přihlašovací stránky aplikace. Lze zvolit řešení pomocí prvku *Login*, nebo s využitím metod z třídy *Membership*.

Třída *Membership* spolu s třídou *MembershipUser* tvoří základ podkladového API rozhraní pro členství. Všechny ovládací prvky pro bezpečnost využívají jejich členy pro realizaci vlastní funkcionality. *Membership* má na starosti vše, co se dotýká práce s uživateli. Umožňuje vytvářet nové uživatele, odstraňovat ty existující, provádět jejich aktualizace, načítat seznam všech uživatelů a také poskytovat podrobnosti o jednotlivých uživateli. K těmto operacím využívá právě třídu *MemberShipUser*, která zapouzdřuje informace o konkrétním uživateli.

### 2.2.3.5 Autorizace ASP.NET

V ASP.NET je umožněno použití dvojího typu autorizace. Souborová autorizace se používá v součinnosti s autentizací Windows. Pokud již mají uživatelé v systému Windows nastavená určitá oprávnění, která chceme využívat i v rámci webové aplikace, stačí povolit autentizaci Windows a použít souborovou autorizaci ASP.NET. Pokud v tomto rámci uživatel požaduje nějakou konkrétní webovou stránku, modul souborové autorizace *FileAuthorizationModule* ověří existující oprávnění operačního systému vztahující se k souboru .aspx požadované stránky pro účet aktuálně autentizovaného uživatele. Bez zapojení souborové autorizace by se na úrovni operačního systému kontrolovalo zabezpečení tohoto souboru jen pro pevný účet, pod kterým se kód aplikace vykonává. [1]

Autorizace URL kontroluje, zda má uživatel přístup ke konkrétním prostředkům. Rozhoduje se tedy na základě aktuálního bezpečnostního kontextu a URL požadovaného zdroje. Tento typ autorizace se aktivuje v konfiguračních souborech jednotlivých adresářů aplikace prvkem `<authorization>`.

Existují dva druhy autorizačních pravidel. Povolovací a zakazující pravidla. Každé pravidlo může být aplikováno na několik uživatelů a rolí, atribut *verbs* potom značí, o jaký typ http požadavku se jedná. Nejjednodušší pravidlo bývá používáno pro použití samotné autentizace. Aby mohla být autentizace aktivována, je nejdříve potřeba zakázat přístup ke zdroji neautentizovaným uživatelům. Pokud se tak nestane, znamená to, že je zdroj poskytován bez rozdílu všem uživatelům a autentizace je tudíž v tomto případě zbytečná.

```
<authorization><deny users= "?" /></authorization>
```

Znak "?" symbolizuje všechny uživatele s dosud neznámou identitou. Pokud tento kód vložíme do souboru `web.config` libovolného adresáře aplikace, tak při každém pokusu o přístup k souborům z tohoto adresáře ze strany anonymního uživatele bude aktivován proces autentizace. Znak "\*" potom zastupuje úplně všechny uživatele, i ty neautentizované. Soubor `machine.config` obsahuje také jedno autorizační pravidlo, využívající právě tento symbol a umožňující přístup k celé aplikaci všem uživatelům. Bývá vyhodnocováno jako úplně poslední autorizační pravidlo celé aplikace, povoluje tedy dodatečně přístup k aplikaci všem uživatelům, na které dosud nebyly uplatněny žádné restrikce.

Autorizační pravidla se v rámci adresáře vyhodnocují postupně za sebou. Pokud je na uživatele aplikováno nějaké pravidlo, potom každé další pravidlo týkající se stejného uživatele bude ignorováno. V rámci podadresářů pracuje vyhodnocovací systém autorizace následovně: Nejdříve se aplikují pravidla u konečného adresáře. Pokud je tento adresář součástí nadřazeného adresáře, aplikují se následně autorizační pravidla z tohoto výše umístěného adresáře. Jako poslední se provede výše zmiňované kořenové pravidlo ze souboru `machine.config`.

Autorizace přímo v programovém kódu je umožněna voláním metody `IsInRole()` třídy `IPrincipal`. Alternativou může být také použití třídy `PrincipalPermission`. Při vytváření instance se do konstruktoru třídy předají parametry s uživatelským jménem a názvem role. Jeden ze dvou parametrů může být nahrazen hodnotou `null`. V místě autorizační kontroly je vyvolána metoda `Demand()`. Pokud je autorizace úspěšná, pokračuje se dále. V opačném případě je vyvolána výjimka `SecurityException`.

ASP.NET podporuje rovněž komplexní infrastrukturu pro správu rolí, podobnou probíranému členství. Obsahuje funkce pro správu rolí a pro získávání rozličných informací o rolích. Jednoduché zprovoznění celé infrastruktury lze provést pomocí nástroje WAT. Vytvoří datové úložiště s propojením na databázi členství spolu s přidruženým poskytovatelem rolí a provede nezbytná nastavení infrastruktury. Lze pomocí něj provádět i veškerou administrační práci spojenou s rolemi. Nastavení poskytovatele rolí i s případnou konfigurací parametrů lze realizovat v konfiguračním souboru aplikace ve značce `<roleManager>`.

Základní třídy pro služby rolí jsou následující:

- **RoleManagerModule**

Zajišťuje přidělování rolí aktuálnímu uživateli, vytváří instance třídy *RolePrincipal*.

- **RoleProvider**

Třída reprezentující jednotlivé poskytovatele rolí.

- **SqlRoleProvider**

Třída odvozená od *RoleProvider*. Jde o implementaci pro poskytovatele založené na databázi SQL Serveru.

- **WindowsTokenRoleProvider**

Získává informace o rolích pro uživatele autentizovaného prostřednictvím autentizace Windows.

- **Roles**

Jedná se o primární rozhraní k úložišti rolí. Pomocí této třídy lze vytvářet programovou správu rolí.

- **RolePrincipal**

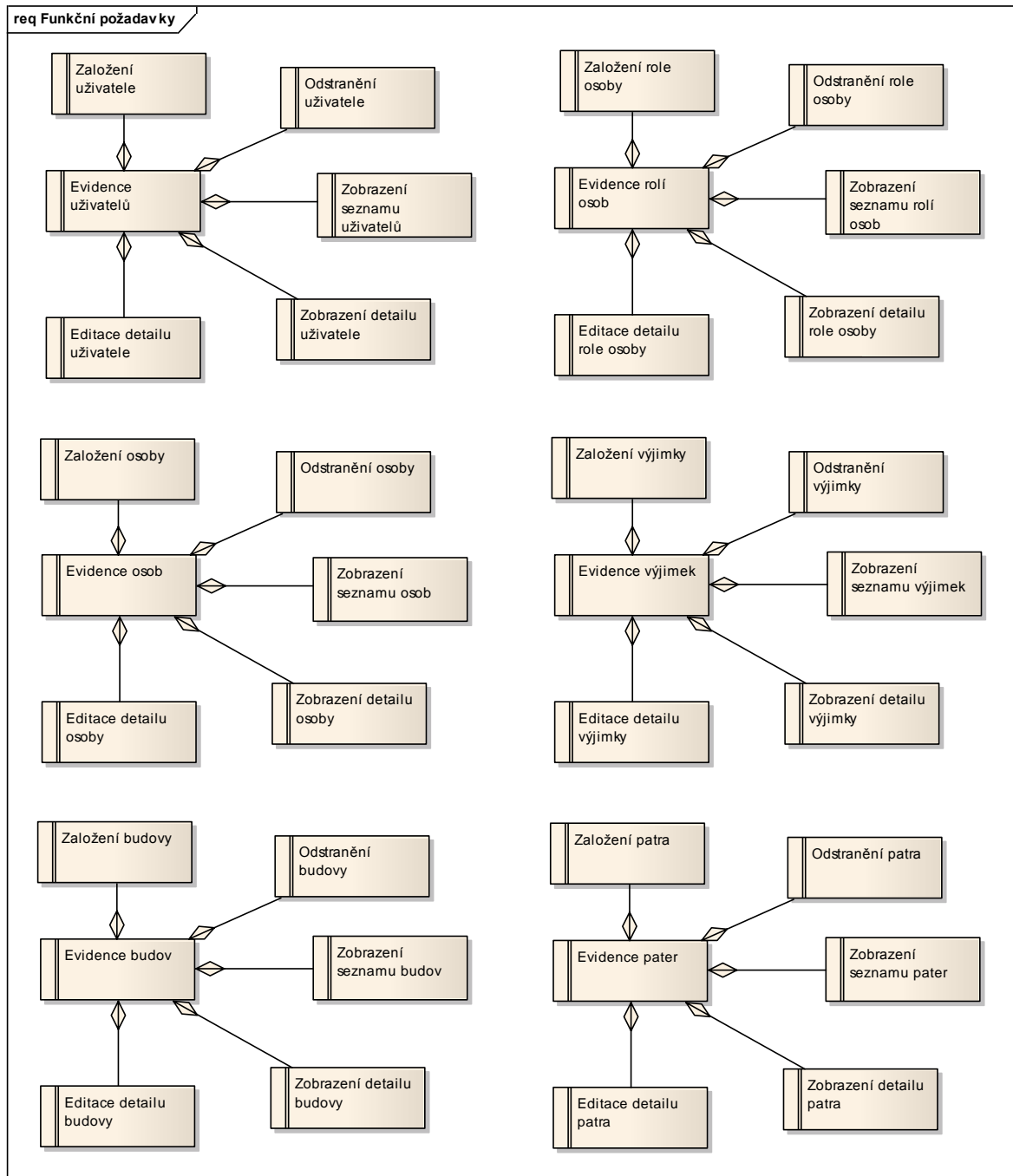
Implementace třídy *IPrincipal* spojující role s autentizovaným uživatelem.

Všechny doposud zmíněné autentizační a autorizační systémy pracují s jedním zásadním omezením: Vztahují se jen na typy souborů, s kterými umí pracovat ASP.NET, jako jsou webové stránky, webové služby, konfigurační soubory, atd. Pokud si uživatel vyžádá například nějaký html soubor, obejde tím jakoukoliv autorizační kontrolu. Pokud má takovýto požadavek projít bezpečnostním modelem ASP.NET, musí se typ požadovaného souboru namapovat na ISAPI filtr ASP.NET. Dále je ještě nutné sdělit ASP.NET, jak má při zpracování takového požadavku postupovat. To se řeší tvorbou vlastní třídy http ovladače, který bude tento typ požadavků zpracovávat.

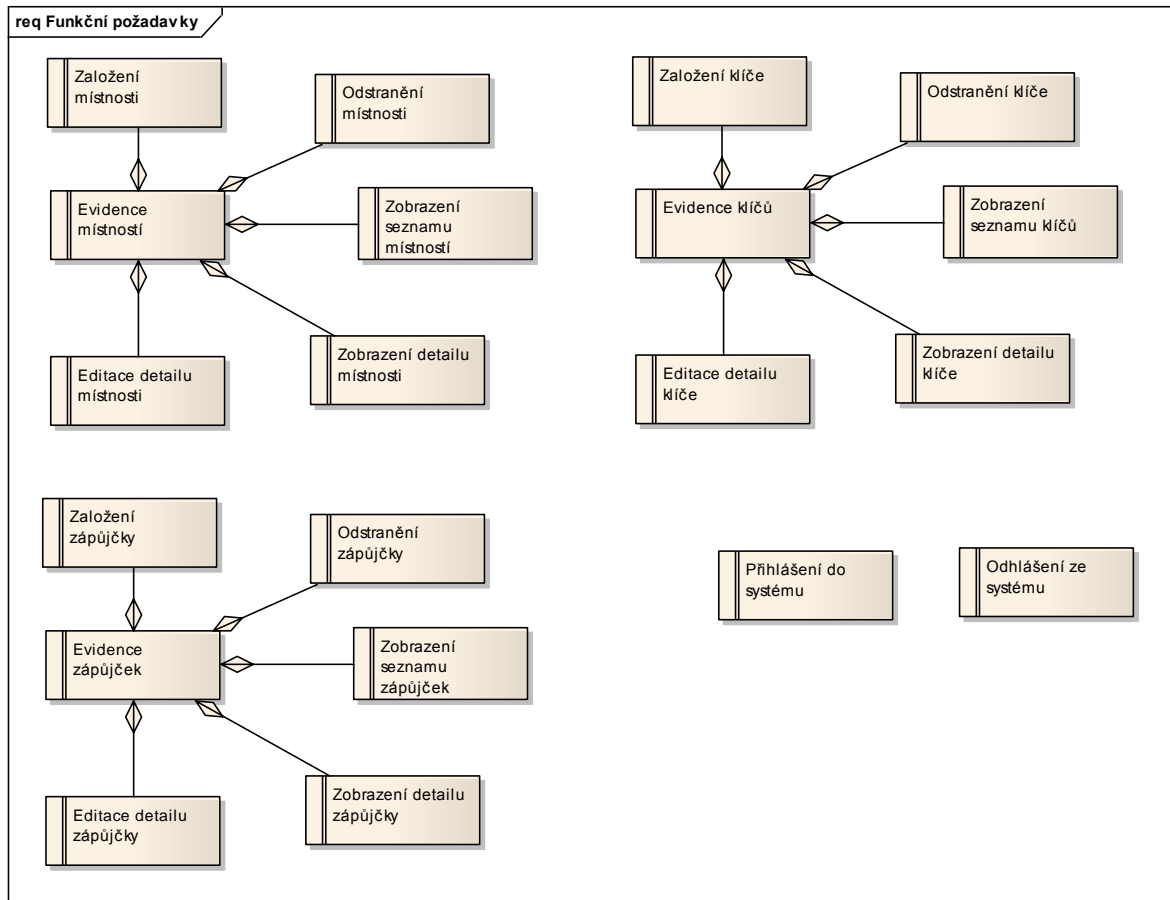
### 3 ANALYTICKÁ ČÁST

#### 3.1 Požadavky

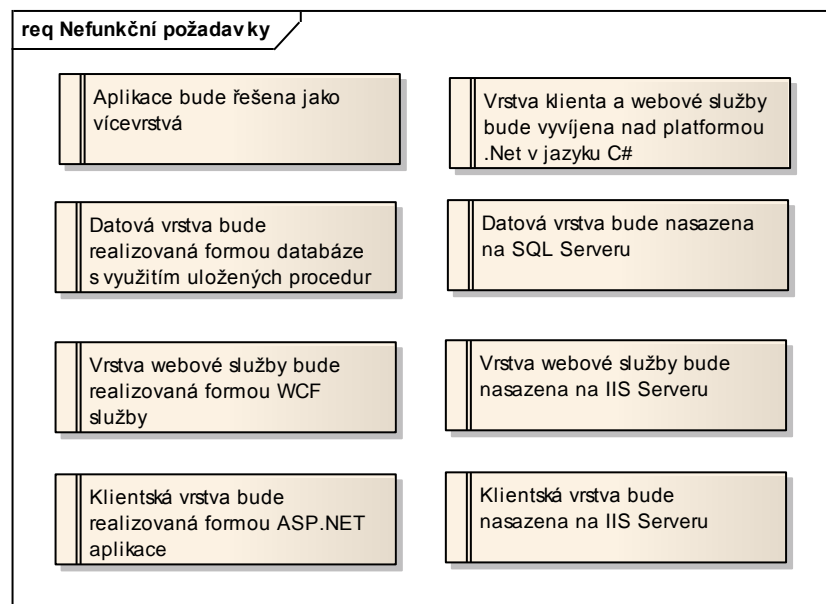
Z předchozího popisu problému vyplívají následující požadavky na aplikaci:



Obr. 3.1 – Funkční požadavky na aplikaci, část 1



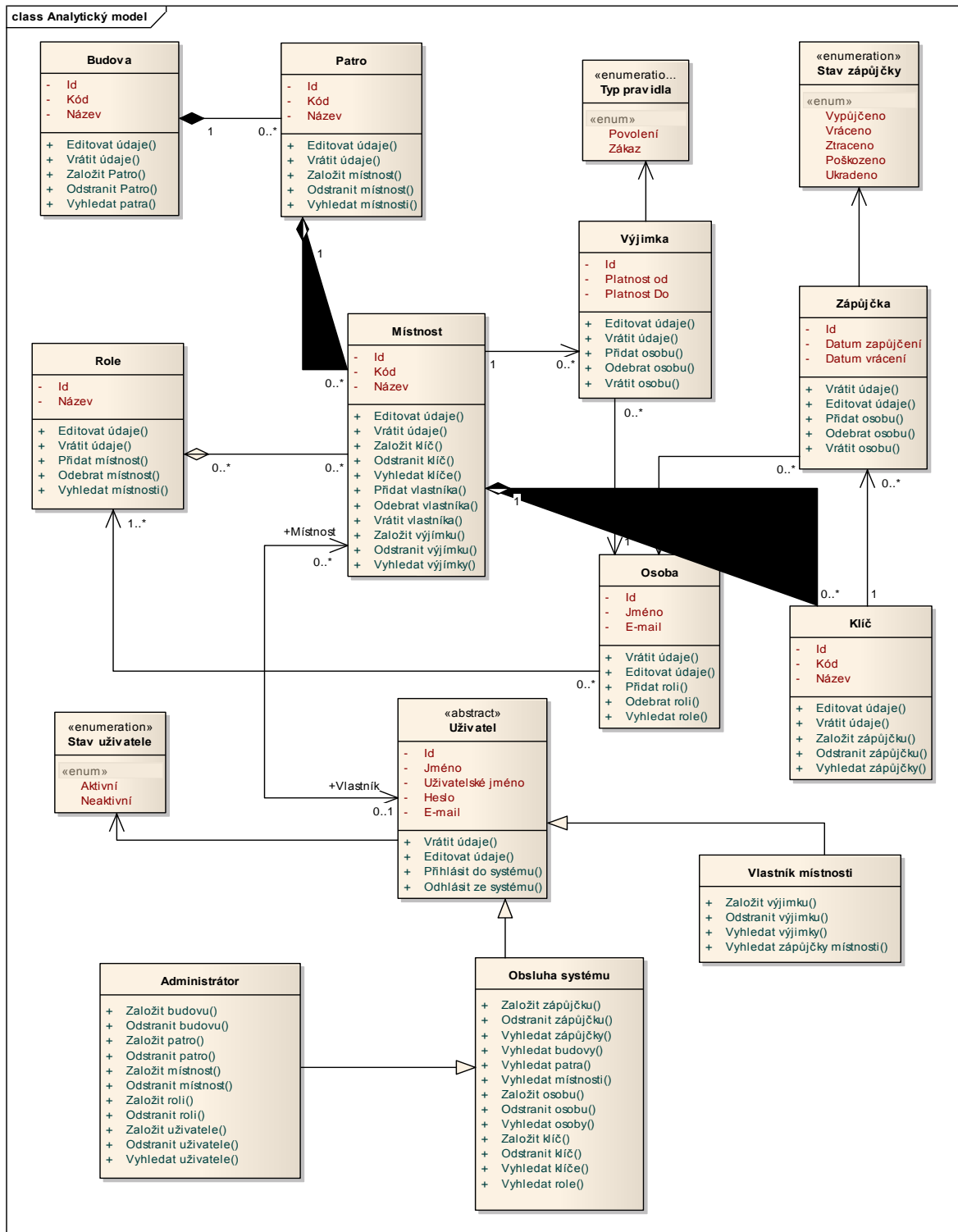
Obr. 3.2 – Funkční požadavky na aplikaci, část 2



Obr. 3.3 – Nefunkční požadavky na aplikaci

### 3.2 Diagram tříd

Při analýze funkčních požadavků na aplikaci byl zkonstruován analytický diagram tříd, který identifikuje hlavní entity aplikace, jejich vlastnosti, dostupné operace a vzájemné vztahy mezi entitami.



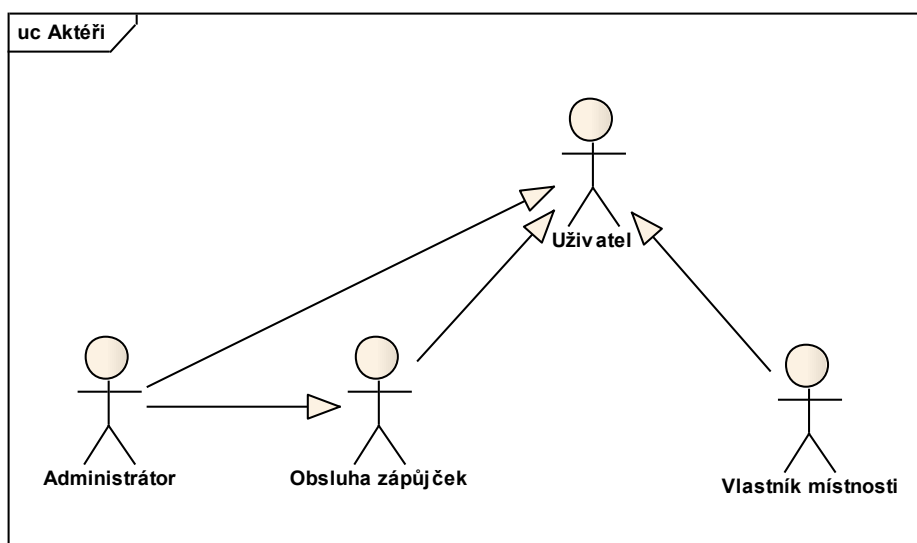
Obr. 3.4 – Analytický diagram tříd

### 3.3 Diagramy případů užití

Diagramy případů užití (use case model) popisují funkcionalitu systému z pohledu jednotlivých uživatelů, zohledňuje se zde přístup k aplikaci v kontextu jednotlivých uživatelských rolí.

#### 3.3.1 Aktéři

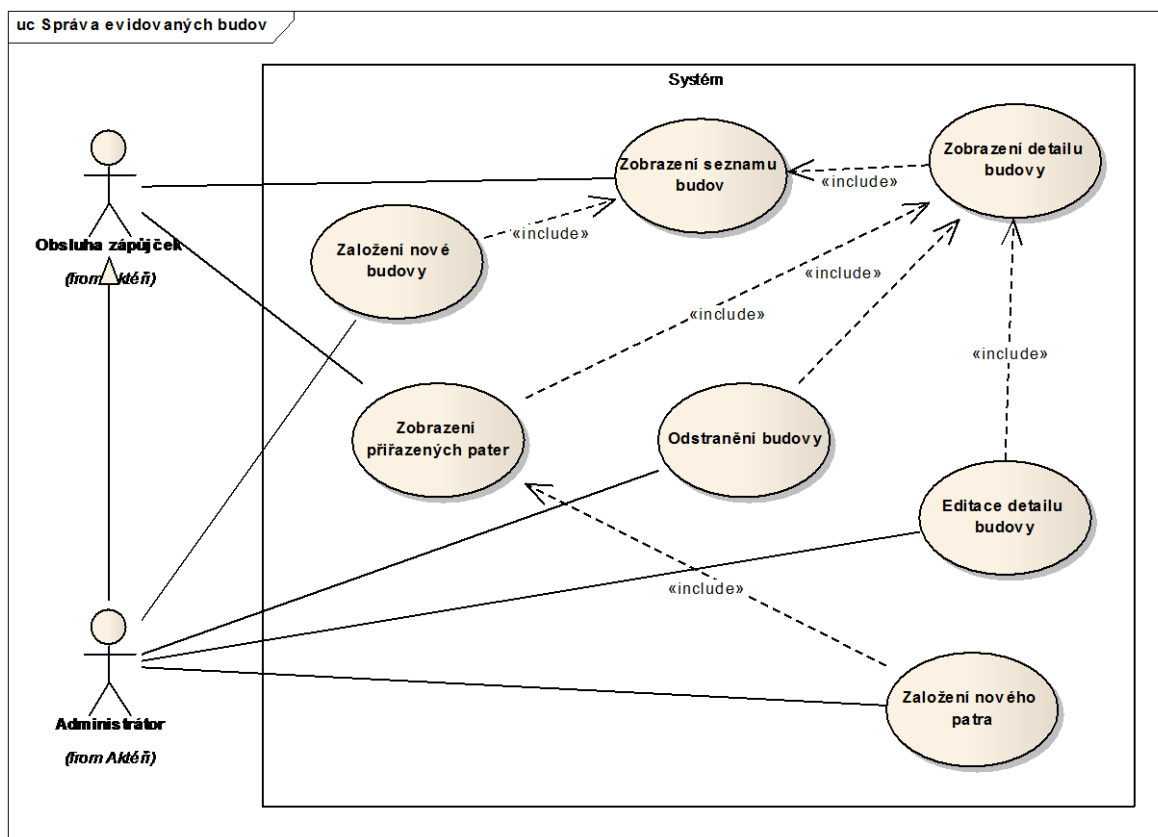
Aktéři zastupují všechny uživatele, patřící do jednotlivých rolí.



Obr. 3.5 – Aktéři

#### 3.3.2 Evidence budov

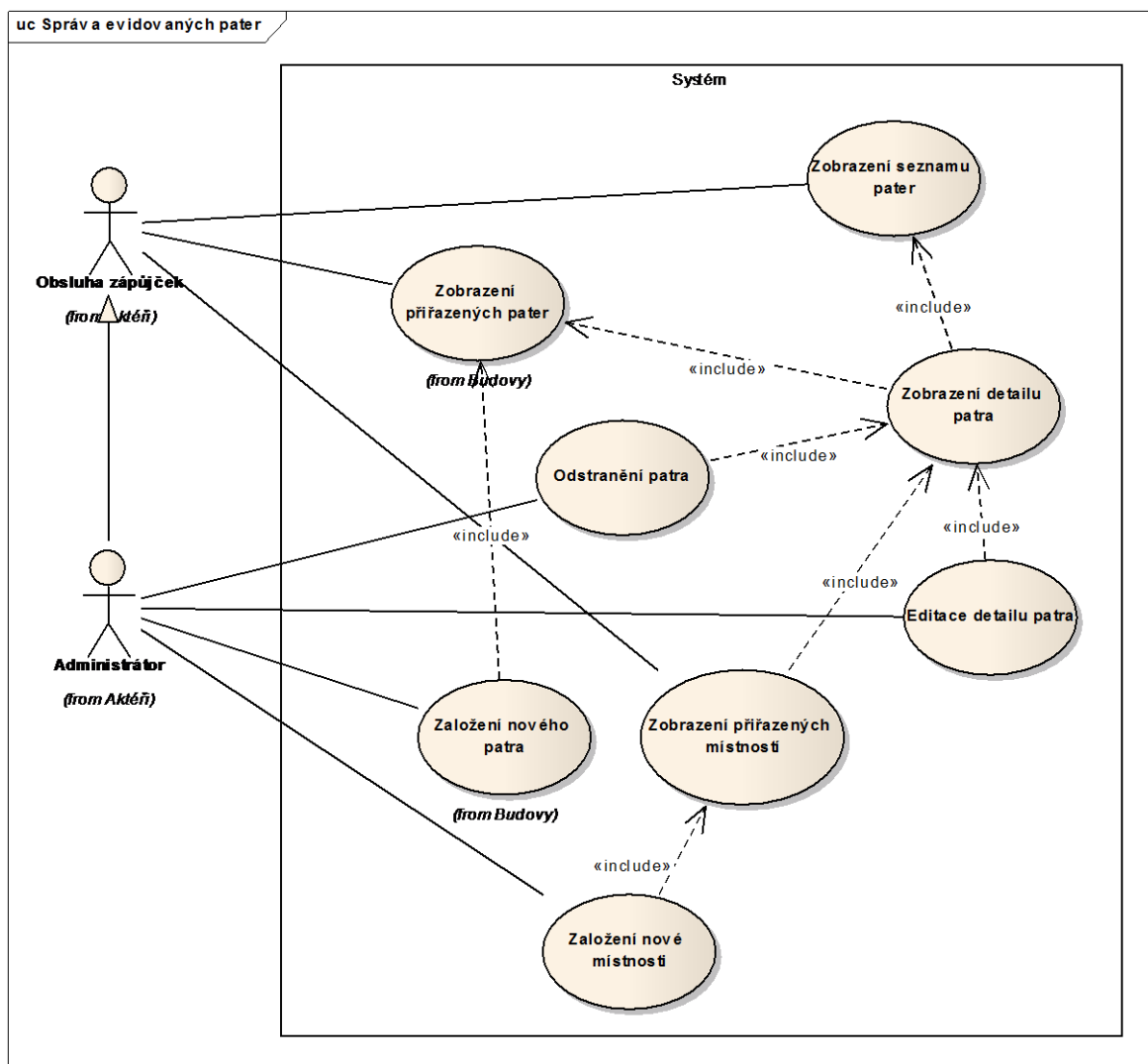
Základní operací v rámci evidence budov je zobrazení seznamu evidovaných budov. Tímto seznamem lze poté filtrovat podle dostupných kritérií (obr. 3.15, str. 46). Při zobrazeném seznamu evidovaných budov lze zakládat do evidence nové budovy (obr. 3.18, str. 48). Výběrem položky ze seznamu dojde k zobrazení detailu budovy (obr. 3.16, str. 46). Z detailu budovy má uživatel možnost provést aktualizaci údajů budovy (obr. 3.17, str. 47), odstranění budovy z evidence (obr. 3.19, str. 49), nebo zobrazení seznamu přiřazených pater budovy (obr. 3.20, str. 50). Při zobrazeném seznamu přiřazených pater lze zakládat nová patra budovy (obr. 3.21, str. 51).



Obr. 3.6 – Use case model Evidence budov

### 3.3.3 Evidence pater

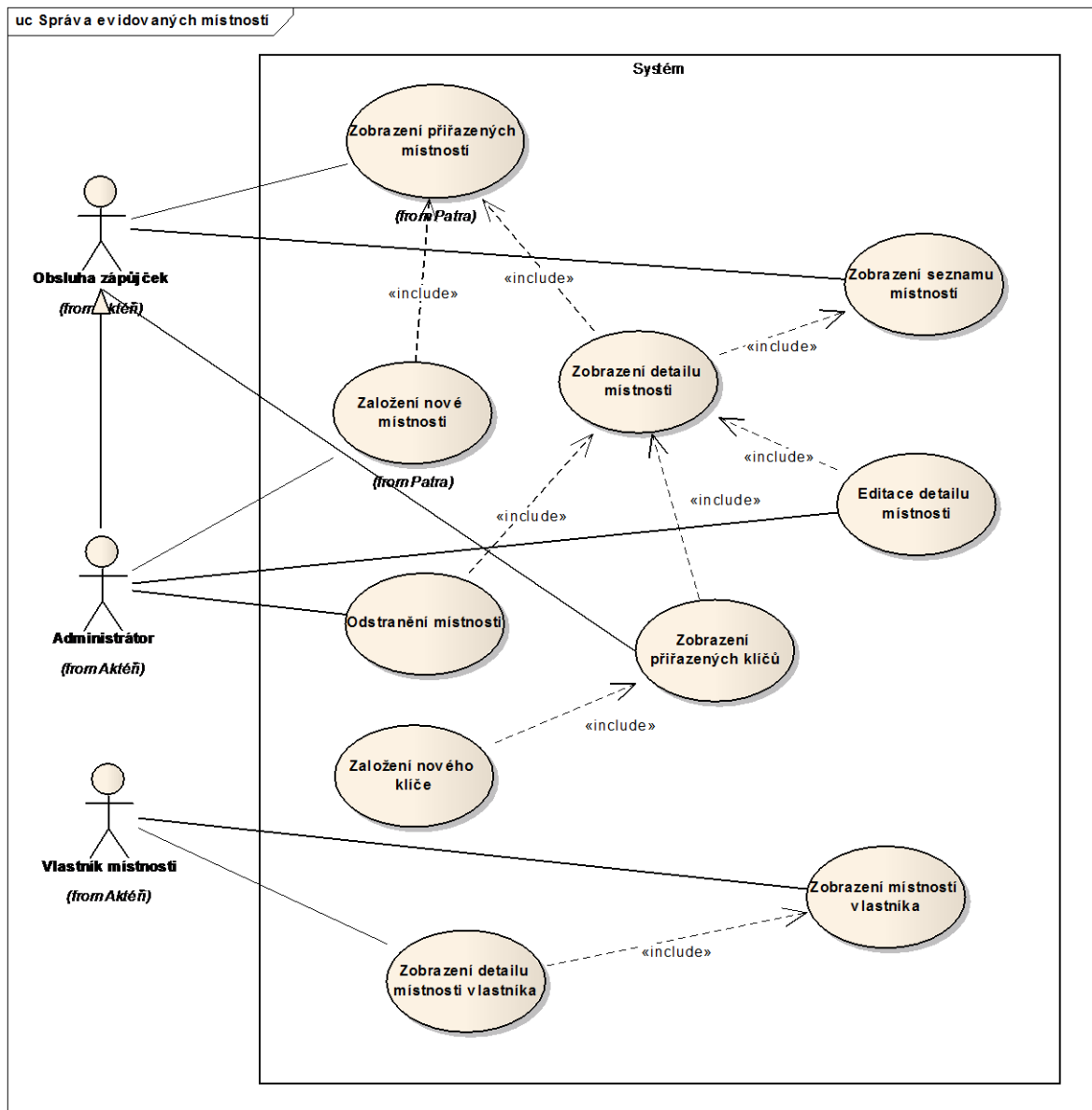
Základní operací v rámci evidence pater je zobrazení seznamu evidovaných pater. Tímto seznamem lze poté filtrovat podle dostupných kritérií. Výběrem položky ze seznamu evidovaných pater, nebo ze seznamu přiřazených pater konkrétní budovy, dojde k zobrazení detailu patra. Z detailu patra má uživatel možnost provést aktualizaci údajů patra, odstranění patra z evidence, nebo zobrazení seznamu přiřazených místností patra. Při zobrazeném seznamu přiřazených místností lze zakládat nové místnosti patra.



Obr. 3.7 – Use case model Evidence pater

### 3.3.4 Evidence místností

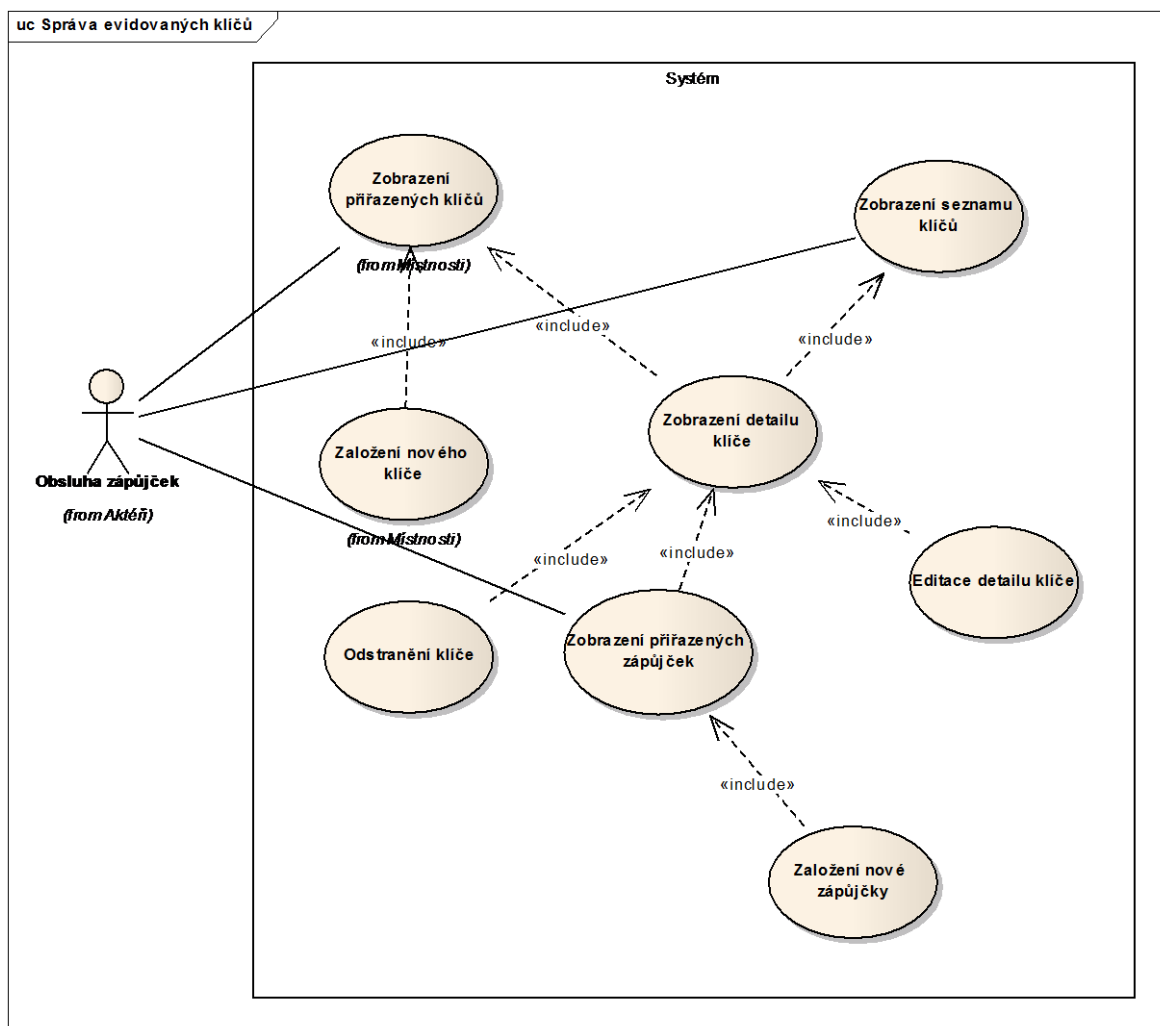
Základní operací v rámci evidence místností je zobrazení seznamu evidovaných místností. Tímto seznamem lze poté filtrovat podle dostupných kritérií. Výběrem položky ze seznamu evidovaných místností, nebo ze seznamu přiřazených místností konkrétního patra, dojde k zobrazení detailu místnosti. Z detailu místnosti má uživatel možnost provést aktualizaci údajů místnosti, odstranění místnosti z evidence, nebo zobrazení seznamu přiřazených klíčů místnosti. Při zobrazeném seznamu přiřazených klíčů lze zakládat nové klíče místnosti. Pokud je uživatel v roli vlastníka místnosti, má možnost zobrazení seznamu místností vlastníka. Výběrem položky ze seznamu se uživateli zobrazí detail místnosti.



Obr. 3.8 – Use case model Evidence místností

### 3.3.5 Evidence klíčů

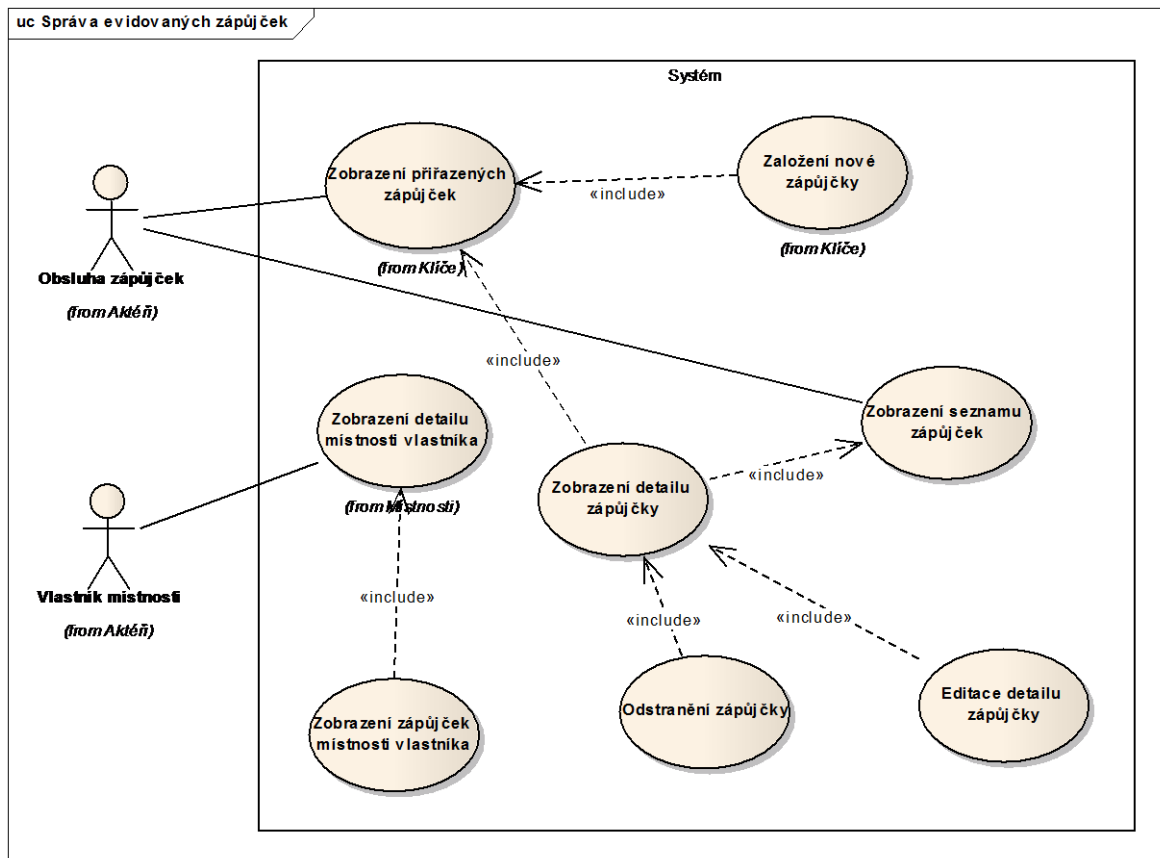
Základní operací v rámci evidence klíčů je zobrazení seznamu evidovaných klíčů. Tímto seznamem lze poté filtrovat podle dostupných kritérií. Výběrem položky ze seznamu evidovaných klíčů, nebo ze seznamu přiřazených klíčů konkrétní místnosti, dojde k zobrazení detailu klíče. Z detailu klíče má uživatel možnost provést aktualizaci údajů klíče, odstranění klíče z evidence, nebo zobrazení seznamu přiřazených zápůjček klíče. Při zobrazeném seznamu přiřazených zápůjček lze zakládat nové zápůjčky klíče (obr. 3.25, str. 55).



Obr. 3.9 – Use case model Evidence klíčů

### 3.3.6 Evidence zápůjček

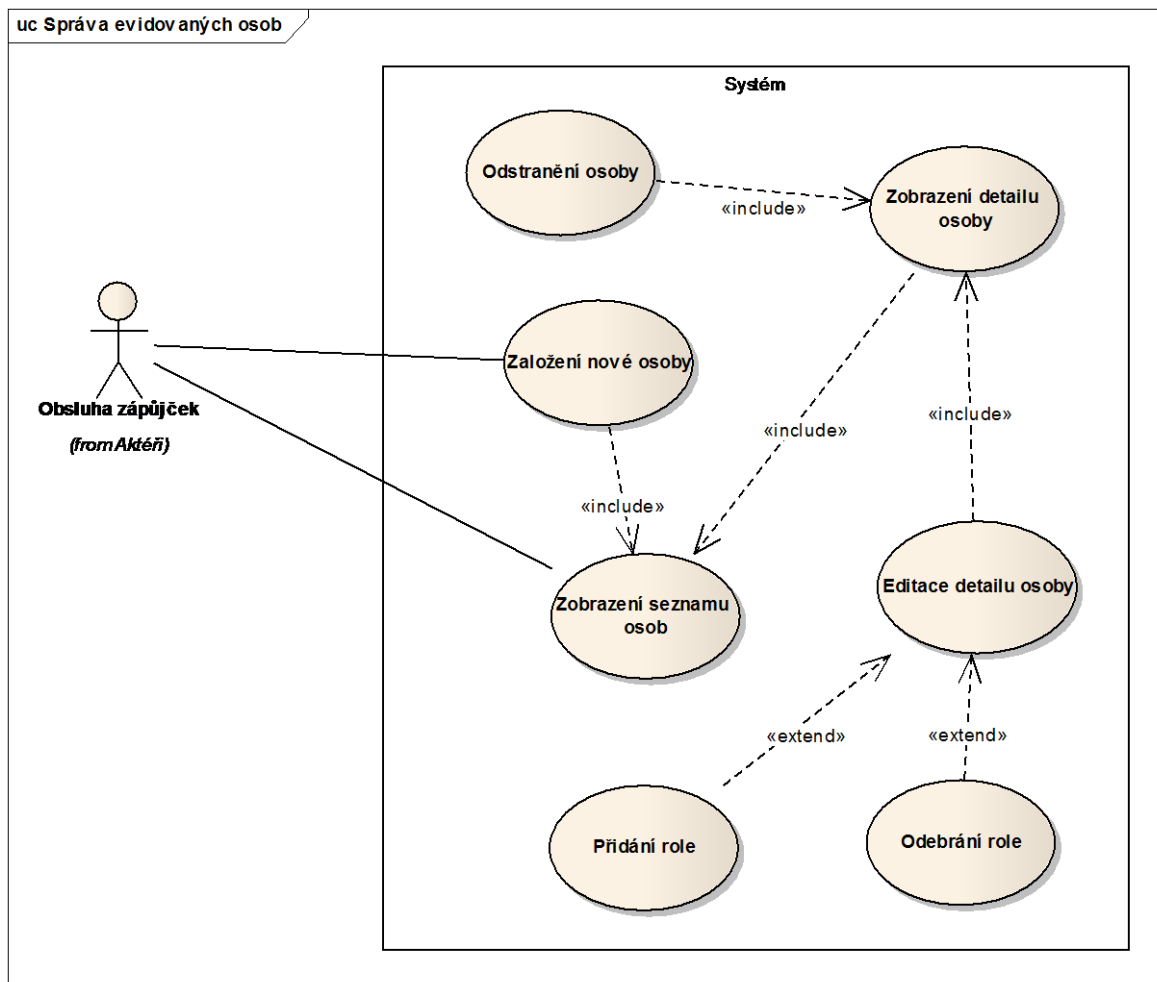
Základní operací v rámci evidence zápůjček je zobrazení seznamu evidovaných zápůjček. Tímto seznamem lze poté filtrovat podle dostupných kritérií (obr. 3.22, str. 52). Výběrem položky ze seznamu evidovaných zápůjček, nebo ze seznamu přirazených zápůjček konkrétního klíče, dojde k zobrazení detailu zápůjčky (obr. 3.24, str. 54). Z detailu zápůjčky má uživatel možnost provést aktualizaci údajů zápůjčky (obr. 3.27, str. 57), nebo odstranění zápůjčky z evidence (obr. 3.26, str. 56). Pokud má uživatel zobrazený detail místnosti vlastníka, může si nechat zobrazit všechny zápůjčky vztahující se k této místnosti (obr. 3.23, str. 53).



Obr. 3.10 – Use case model Evidence zápůjček

### 3.3.7 Evidence osob

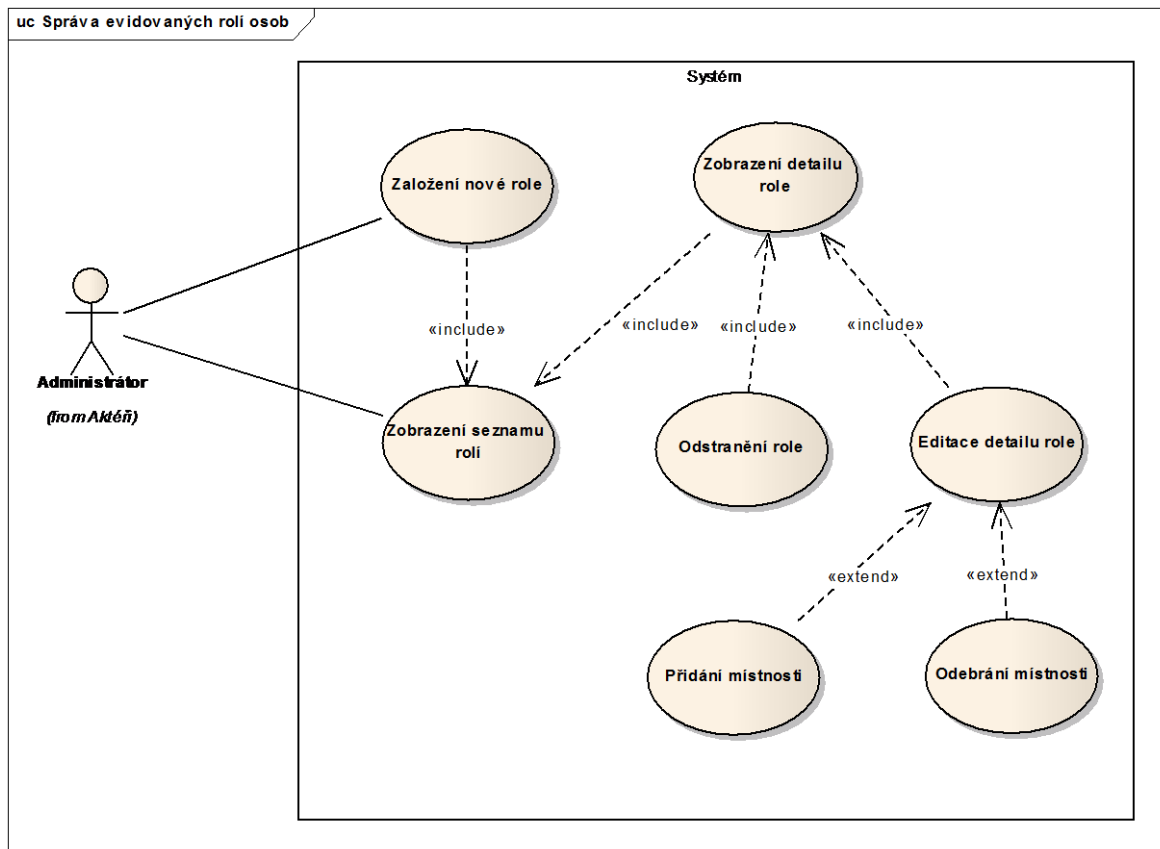
Základní operací v rámci evidence osob je zobrazení seznamu evidovaných osob. Tímto seznamem lze poté filtrovat podle dostupných kritérií. Při zobrazeném seznamu evidovaných osob lze zakládat do evidence nové osoby. Výběrem položky ze seznamu dojde k zobrazení detailu osoby. Z detailu osoby má uživatel možnost provést aktualizaci údajů osoby, nebo odstranění osoby z evidence. Součástí aktualizace údajů osoby může být přidání nebo odebrání role.



Obr. 3.11 – Use case model Evidence osob

### 3.3.8 Evidence rolí osob

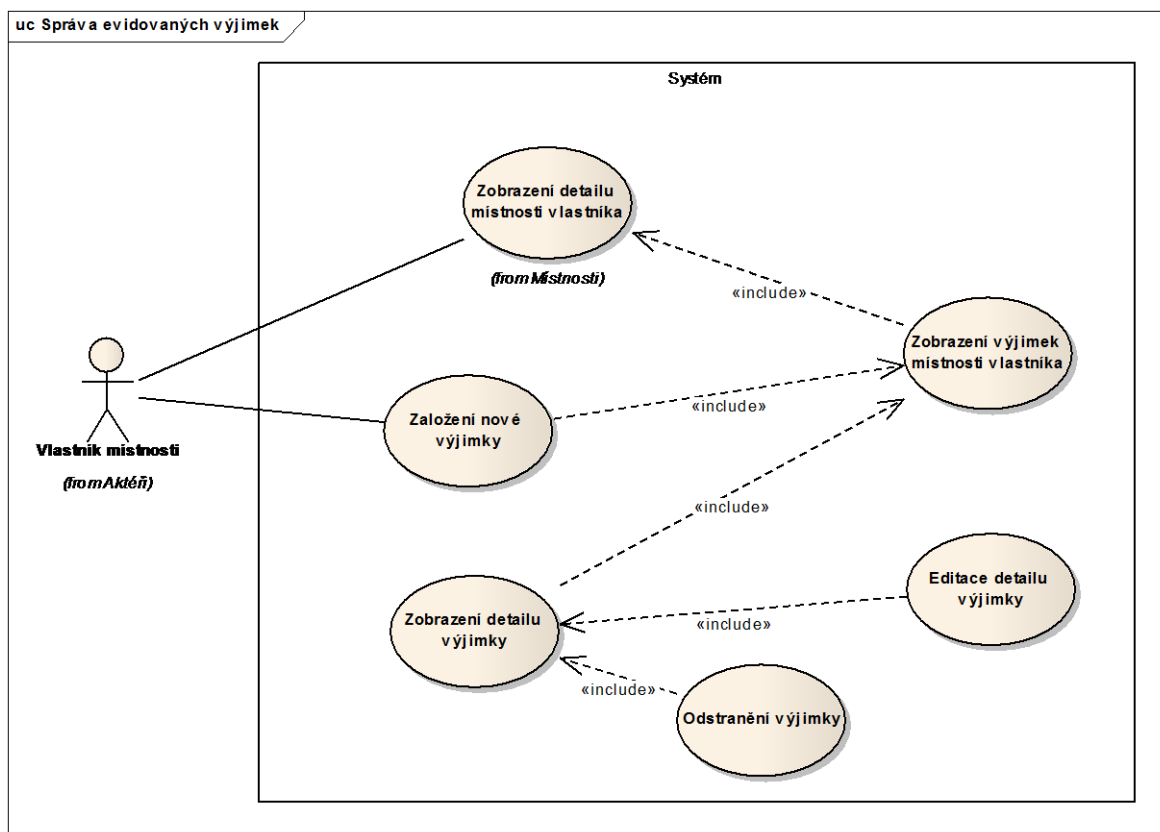
Základní operací v rámci evidence rolí osob je zobrazení seznamu evidovaných rolí osob. Tímto seznamem lze poté filtrovat podle dostupných kritérií. Při zobrazeném seznamu evidovaných rolí osob lze zakládat do evidence nové role osoby. Výběrem položky ze seznamu dojde k zobrazení detailu role osoby. Z detailu role osoby má uživatel možnost provést aktualizaci údajů role osoby, nebo odstranění role osoby z evidence. Aktualizace role osoby spočívá v úpravě seznamu místností, sdružených pod touto rolí. Místnosti lze k roli přidávat, nebo odebírat.



Obr. 3.12 – Use case model Evidence rolí osob

### 3.3.9 Evidence výjimek

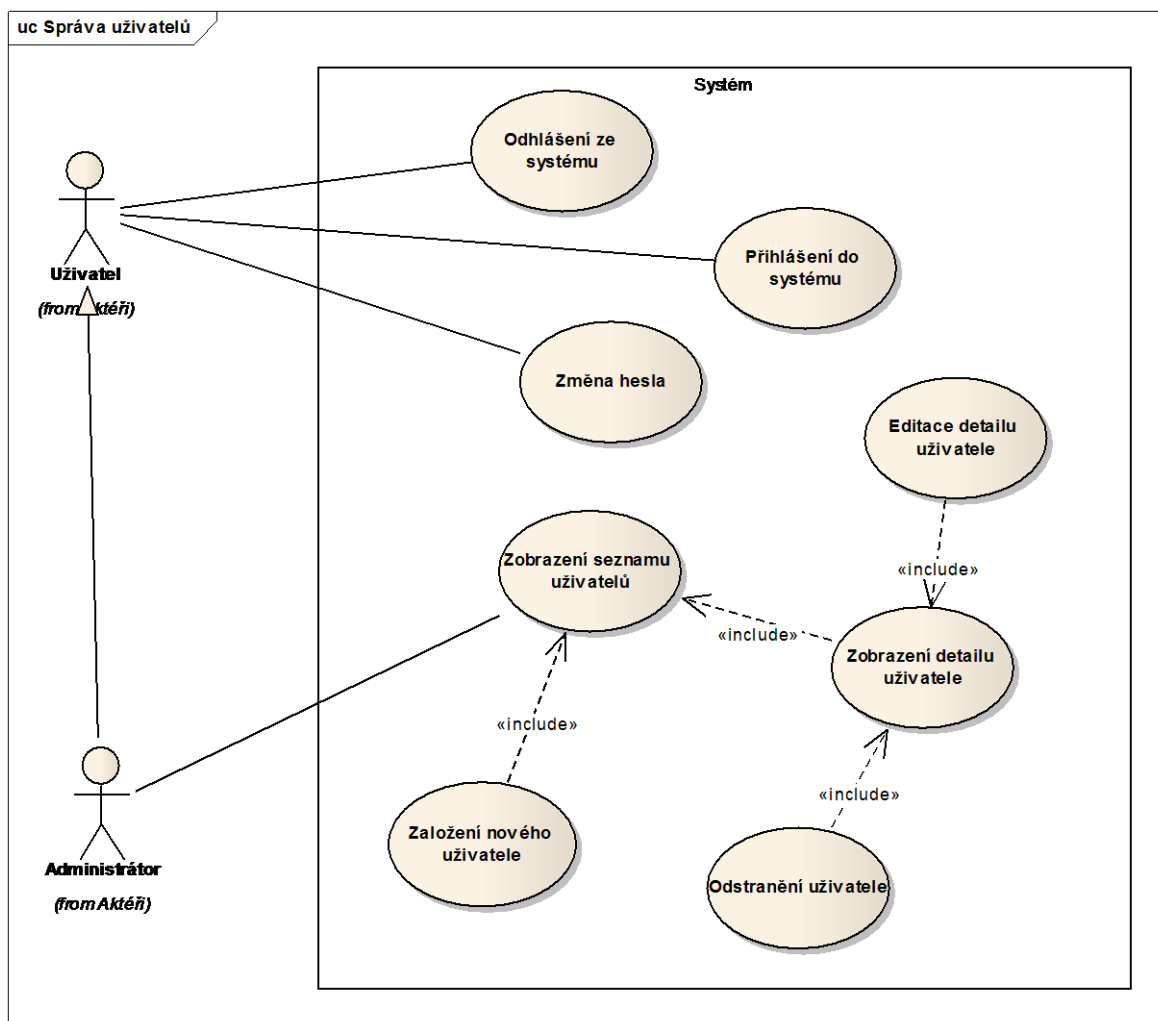
Základní operací v rámci evidence výjimek je zobrazení seznamu evidovaných výjimek místnosti vlastníka. Tímto seznamem lze poté filtrovat podle dostupných kritérií (obr. 3.28, str. 58). Při zobrazeném seznamu výjimek lze zakládat do evidence nové výjimky (obr. 3.31, str. 60). Výběrem položky ze seznamu dojde k zobrazení detailu výjimky (obr. 3.29, str. 59). Z detailu výjimky má uživatel možnost provést aktualizaci údajů výjimky (obr. 3.32, str. 61), nebo odstranění výjimky z evidence (obr. 3.30, str. 59).



Obr. 3.13 – Use case model Evidence výjimek

### 3.3.10 Evidence uživatelů

Každý uživatel aplikace má možnost přihlásit se do systému (obr. 3.33, str. 62), odhlásit se ze systému (obr. 3.34, str. 62), nebo si změnit aktuální heslo (obr. 3.35, str. 63). Základní operací v rámci evidence uživatelů je zobrazení seznamu evidovaných uživatelů. Tímto seznamem lze poté filtrovat podle dostupných kritérií (obr. 3.36, str. 64). Při zobrazeném seznamu uživatelů lze zakládat do evidence nové uživatele (obr. 3.38, str. 66). Výběrem položky ze seznamu dojde k zobrazení detailu uživatele (obr. 3.37, str. 65). Z detailu uživatele lze provést aktualizaci údajů uživatele (obr. 3.40, str. 68), nebo odstranění uživatele z evidence (obr. 3.39, str. 67).



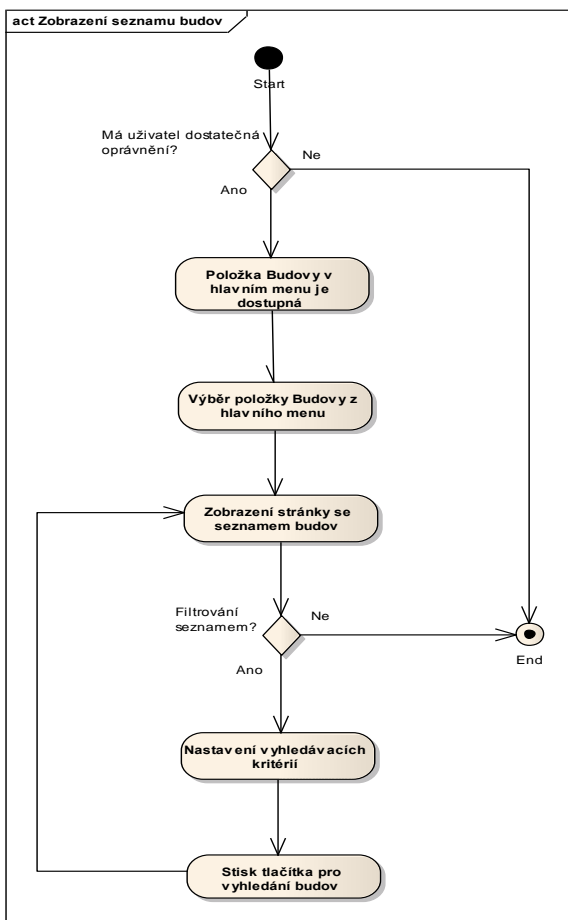
Obr. 3.14 – Use case model Evidence uživatelů

### 3.4 Diagramy aktivit

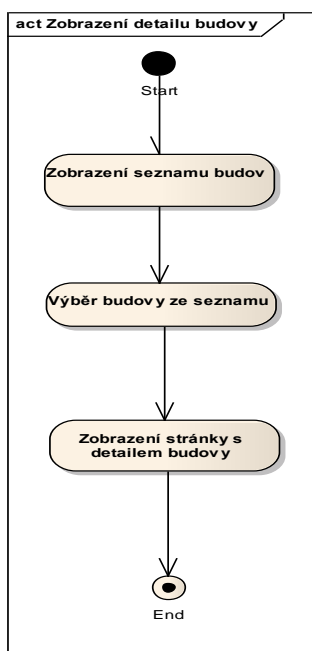
Diagramy aktivit představují modely, které zachycují posloupnost akcí, vedoucí k provedení určité operace v aplikaci. Jedná se o dynamický popis systému. Scénáře pro jednotlivé entity systému se až na detaily velmi podobají a jejich úplný výčet by v této práci zabral příliš mnoho místa. Proto zde budou uvedeny jen diagramy pro popis aktivit spojených s evidencí budov, zápůjček, výjimek a uživatelů systému.

#### 3.4.1 Evidence budov

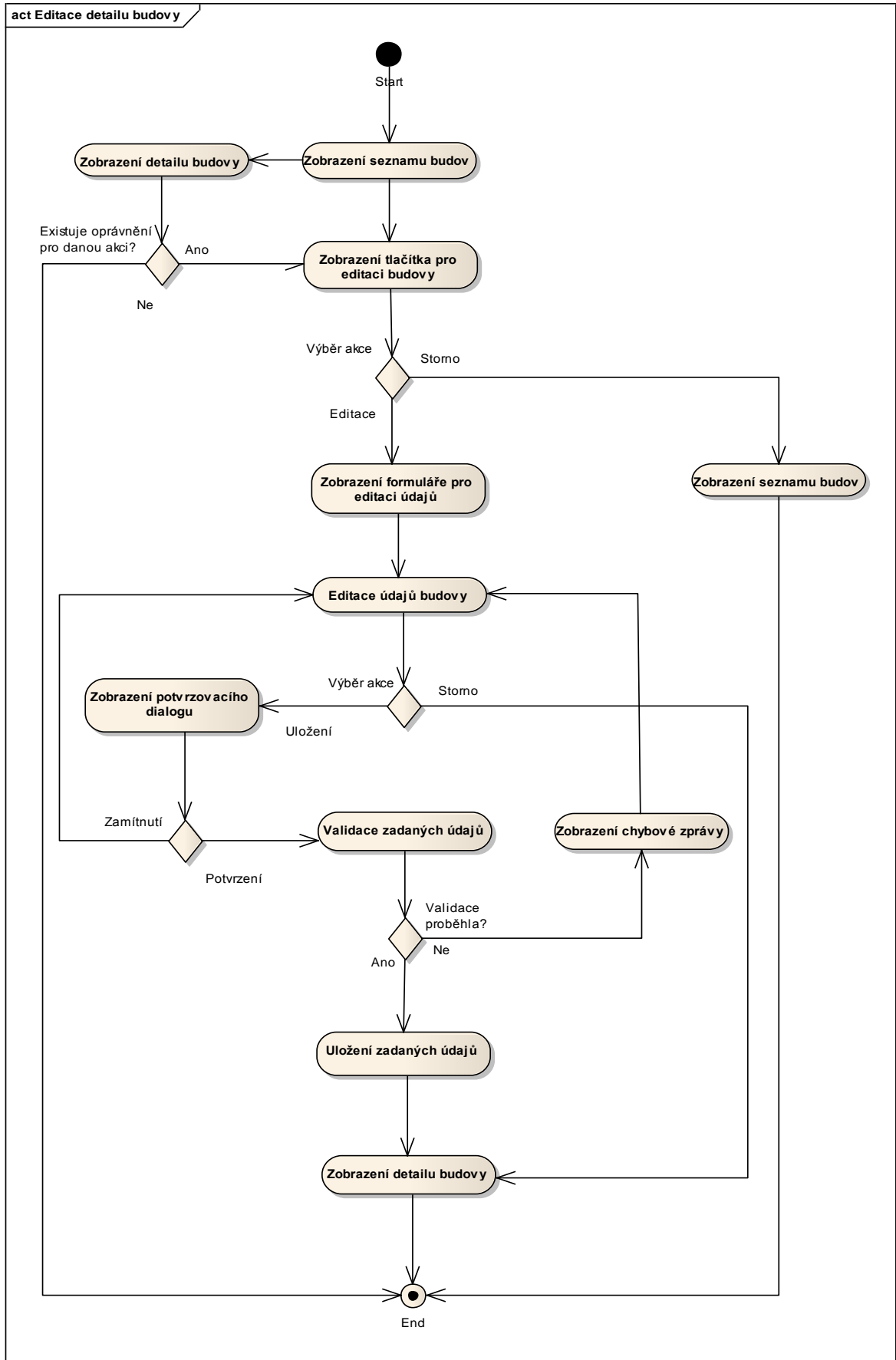
Následující diagramy popisují operace spojené s evidencí budov v systému.



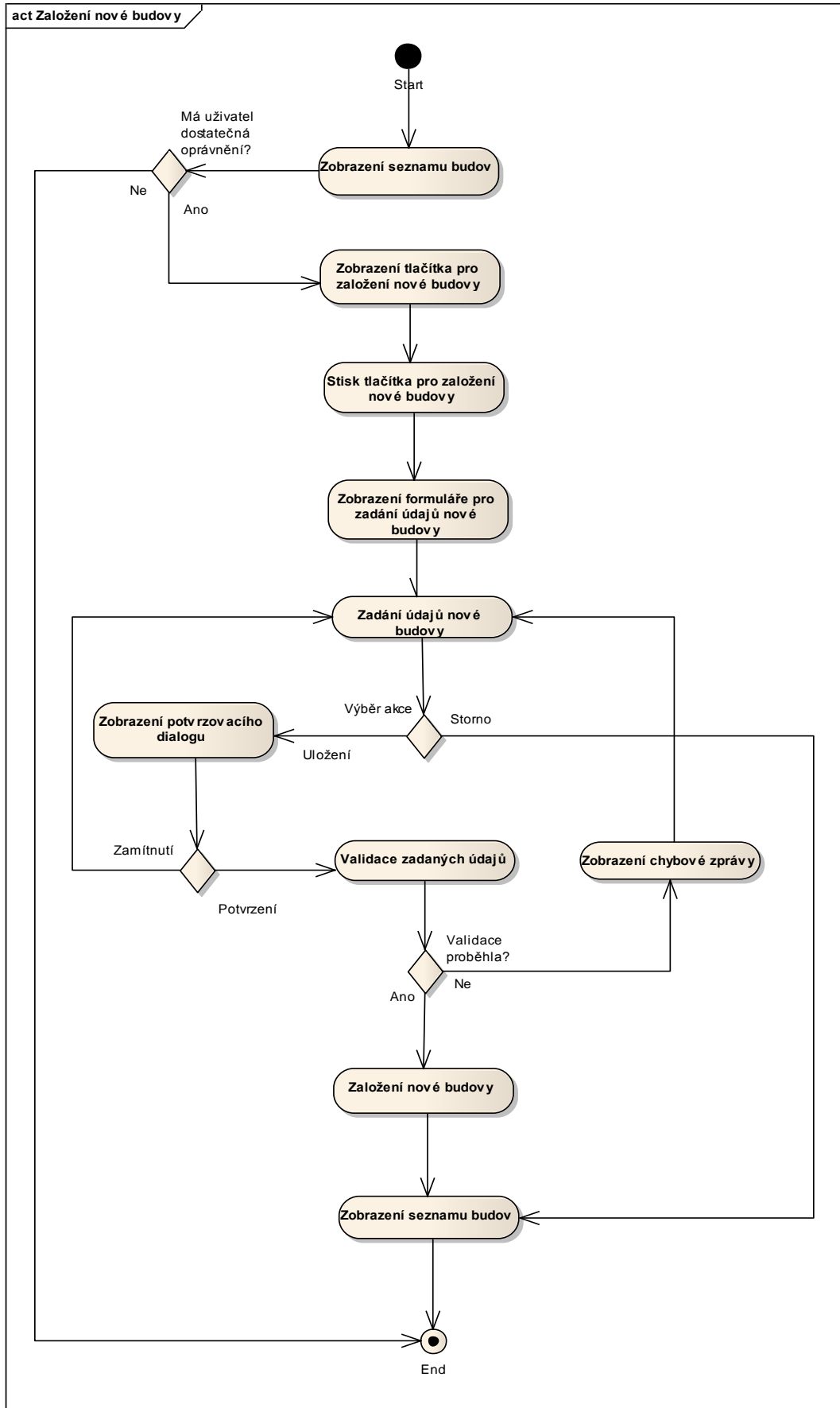
Obr. 3.15 – Diagram aktivit Zobrazení seznamu budov



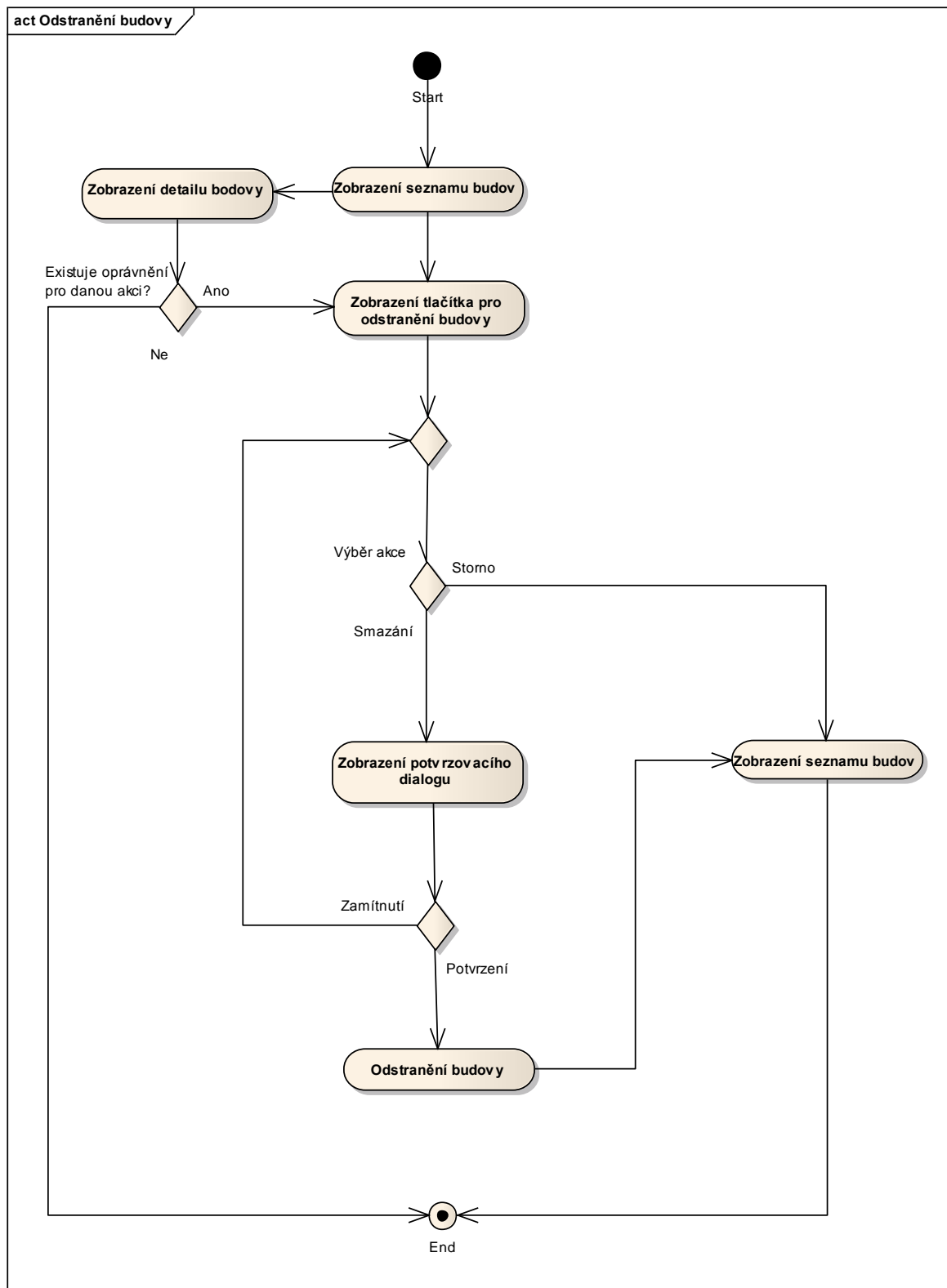
Obr. 3.16 – Diagram aktivit Zobrazení detailu budovy



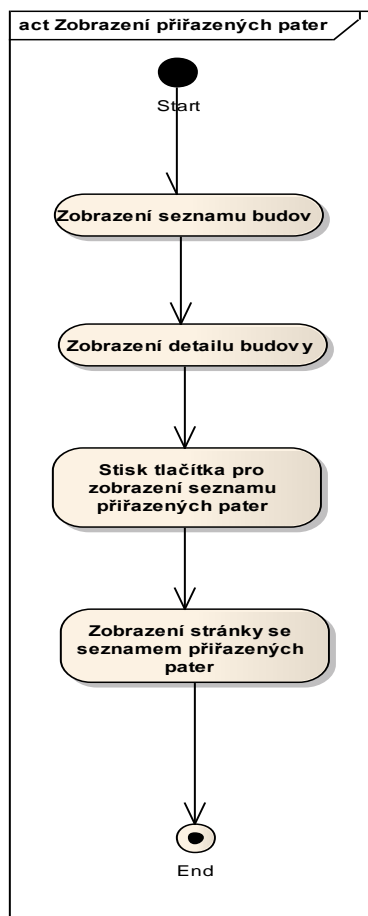
Obr. 3.17 – Diagram aktivit Editace detailu budovy



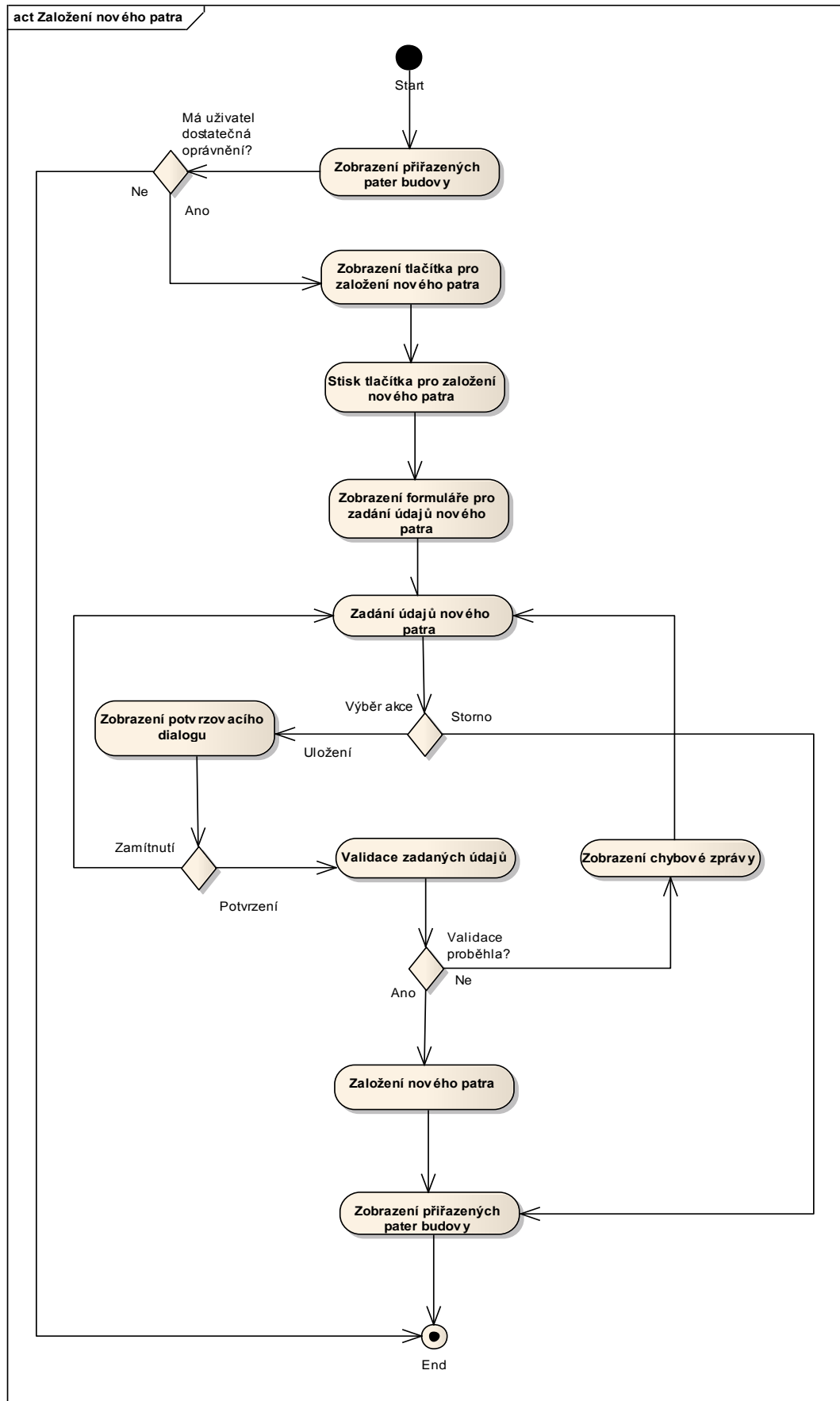
Obr. 3.18 – Diagram aktivit Založení nové budovy



Obr. 3.19 – Diagram aktivit Odstranění budovy



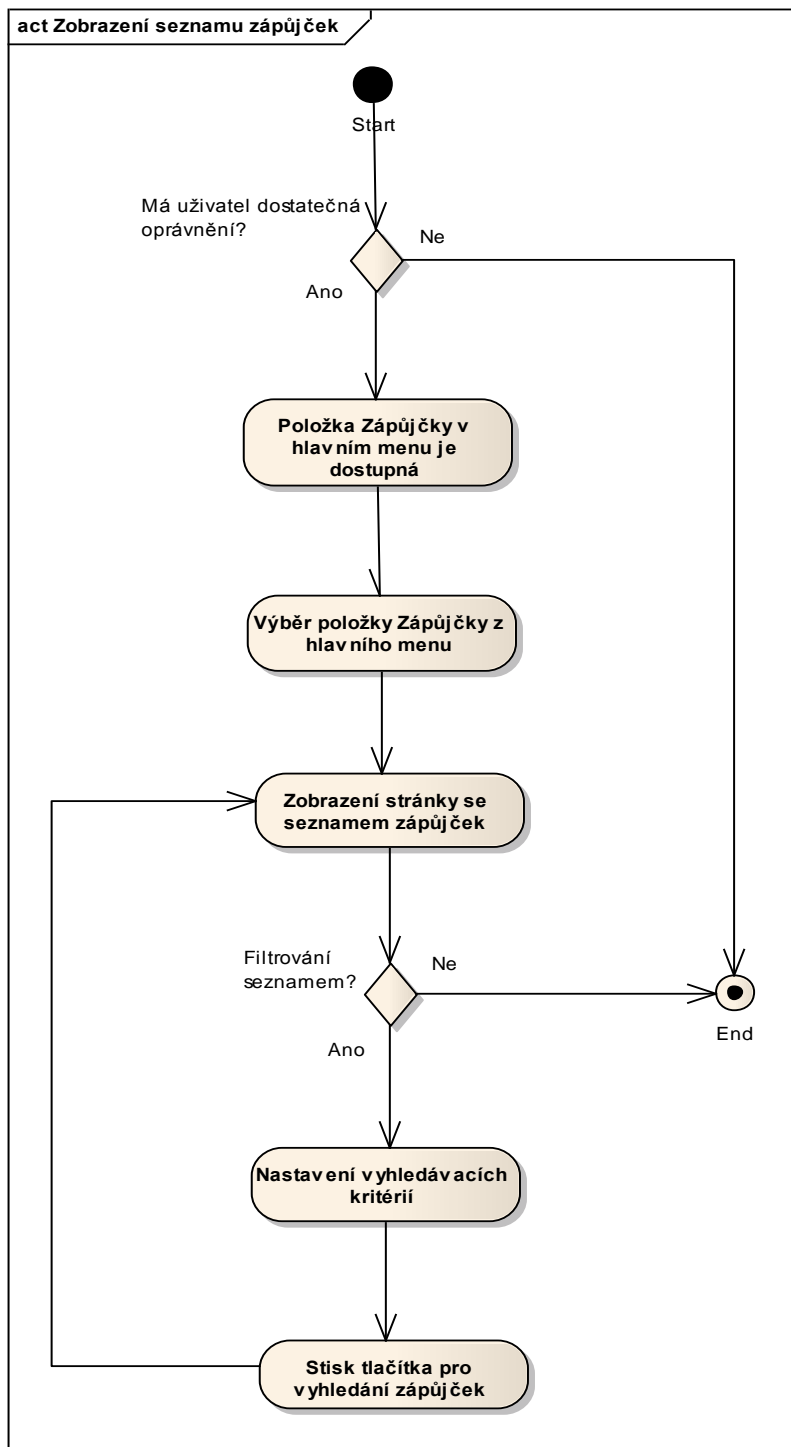
Obr. 3.20 – Diagram aktivit Zobrazení přiřazených pater budovy



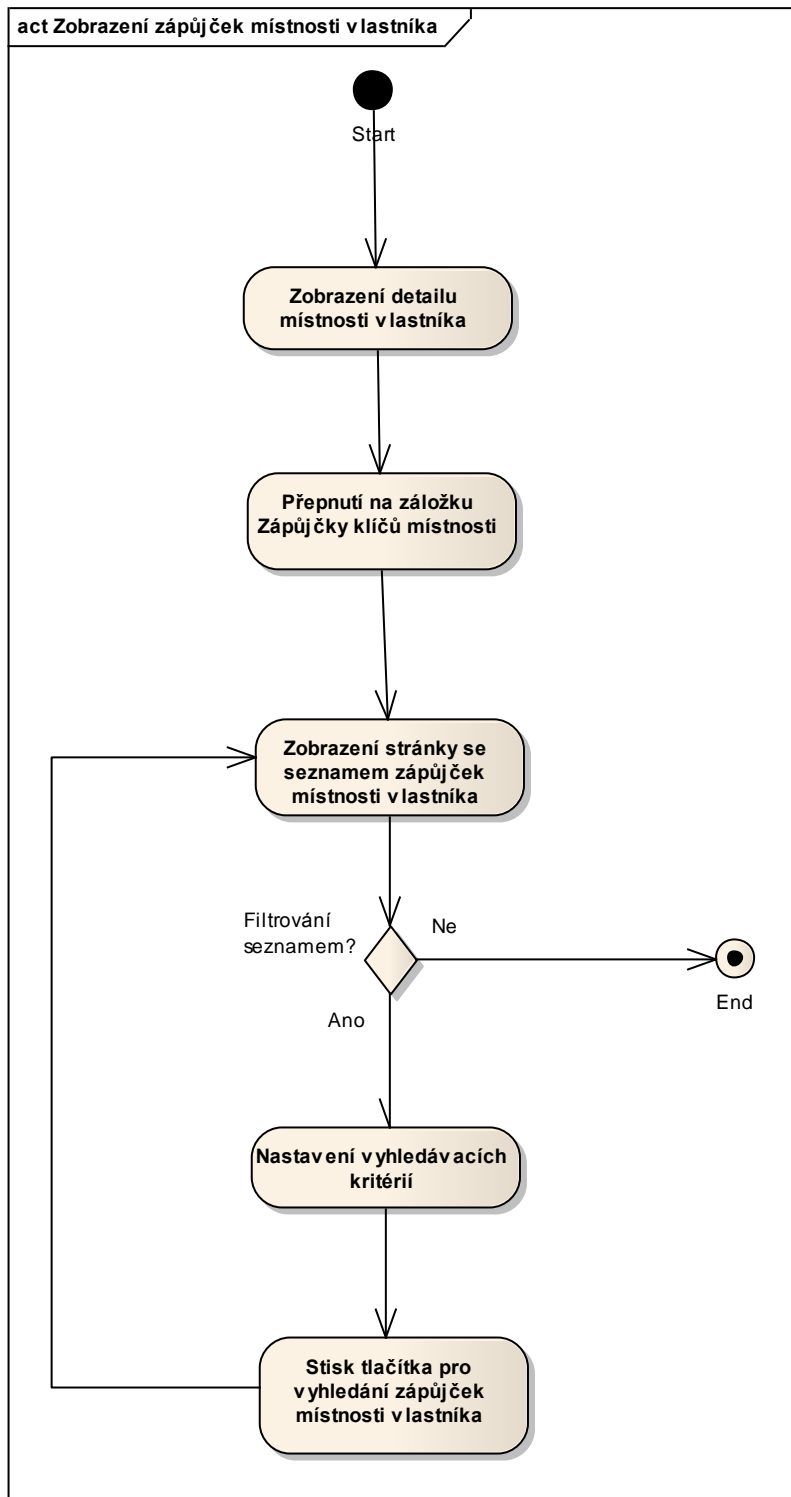
Obr. 3.21 – Diagram aktivit Založení nového patra budovy

### 3.4.2 Evidence zápůjček

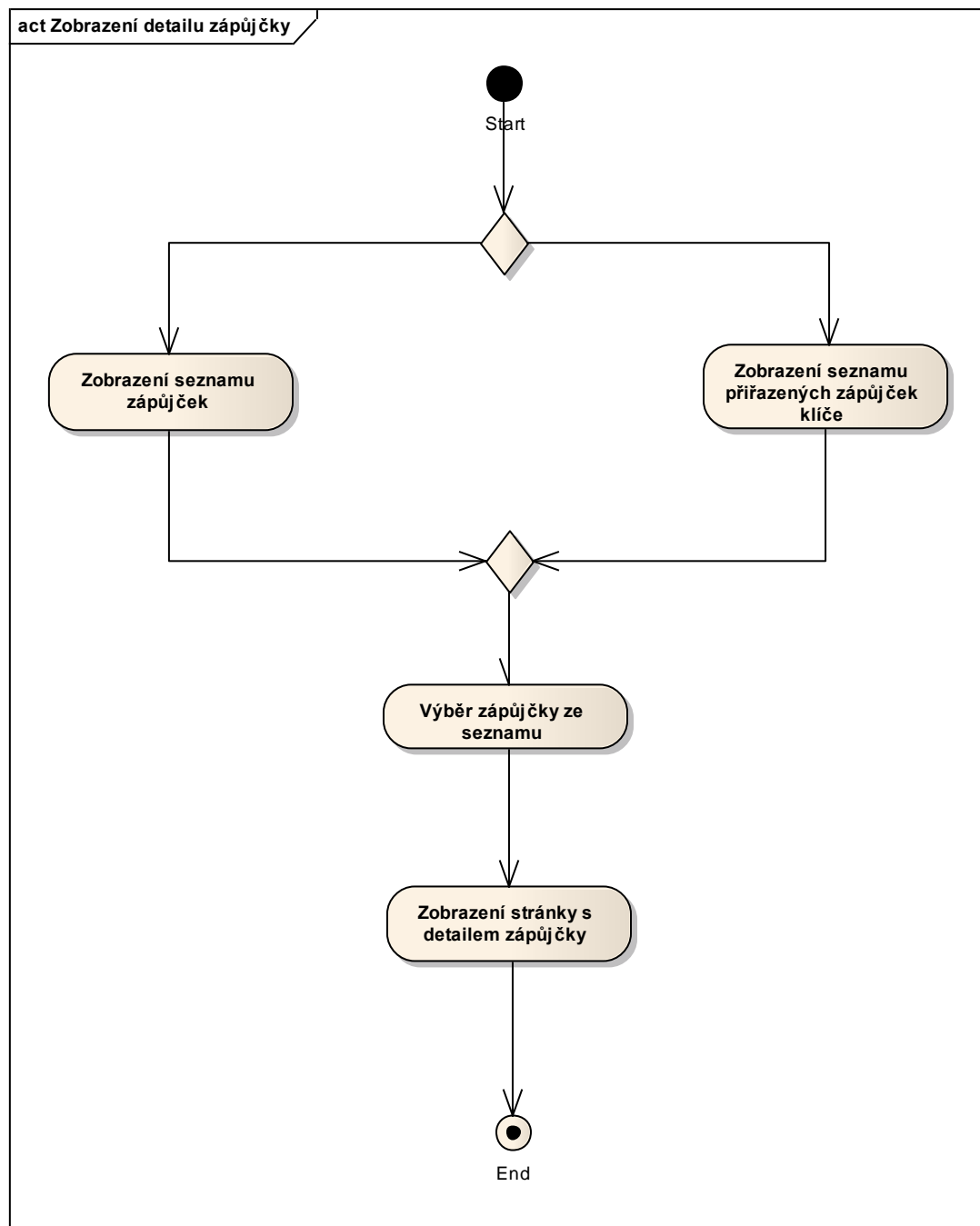
Následující diagramy popisují operace spojené s evidencí zápůjček v systému.



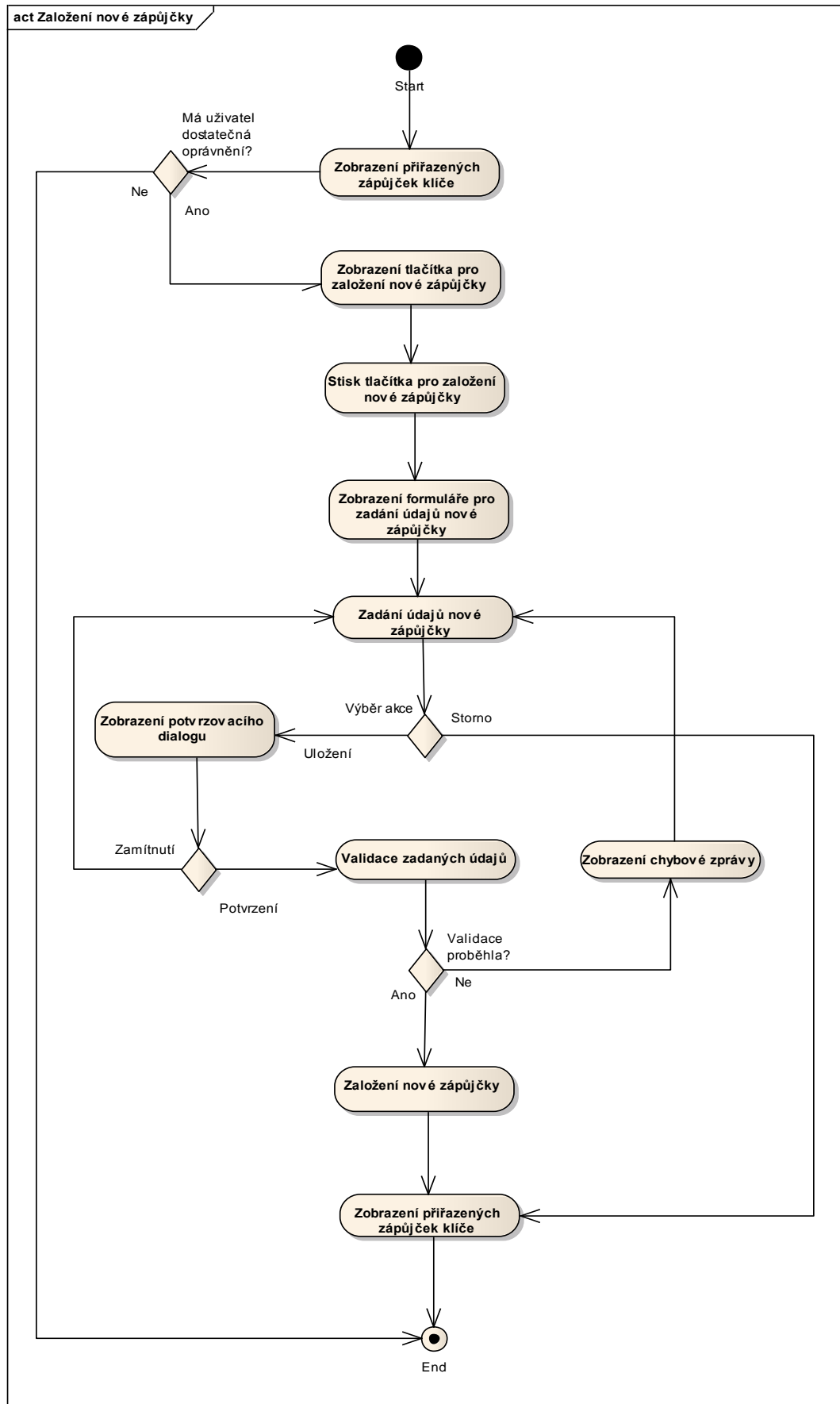
Obr. 3.22 – Diagram aktivit Zobrazení seznamu zápůjček



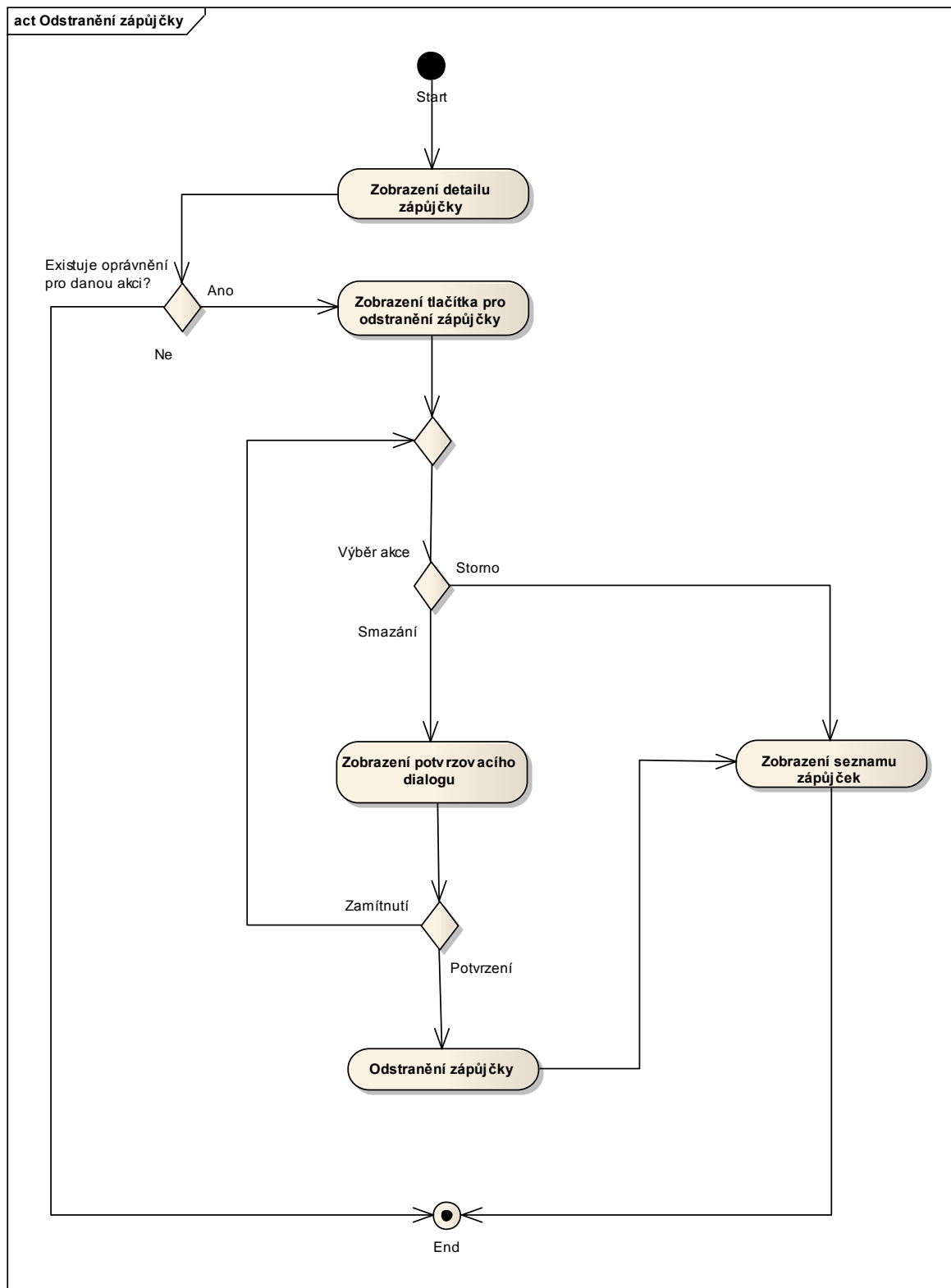
Obr. 3.23 – Diagram aktivit Zobrazení zápůjček místnosti vlastníka



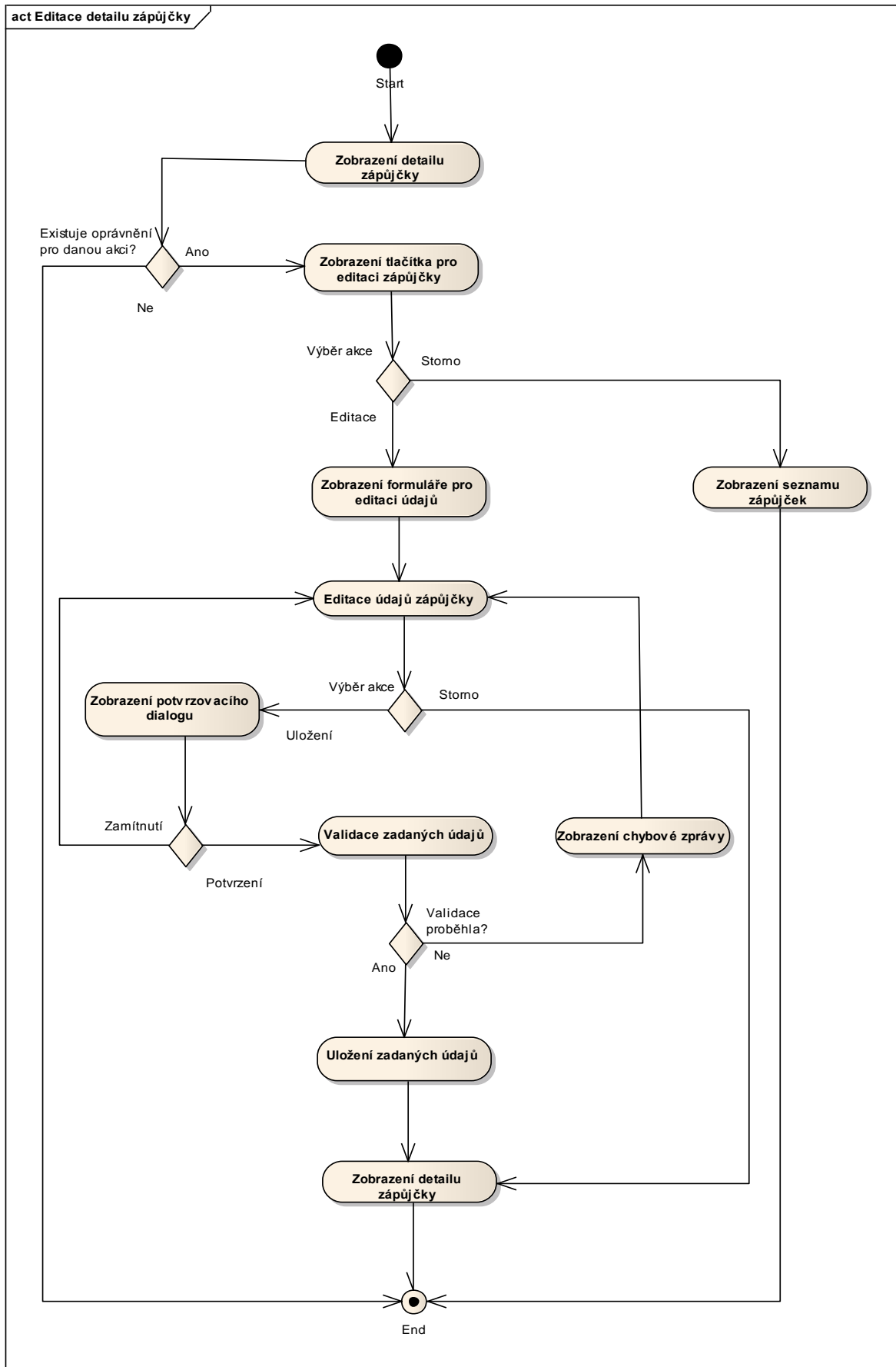
Obr. 3.24 – Diagram aktivit Zobrazení detailu zápůjčky



Obr. 3.25 - Diagram aktivit Založení nové zápůjčky



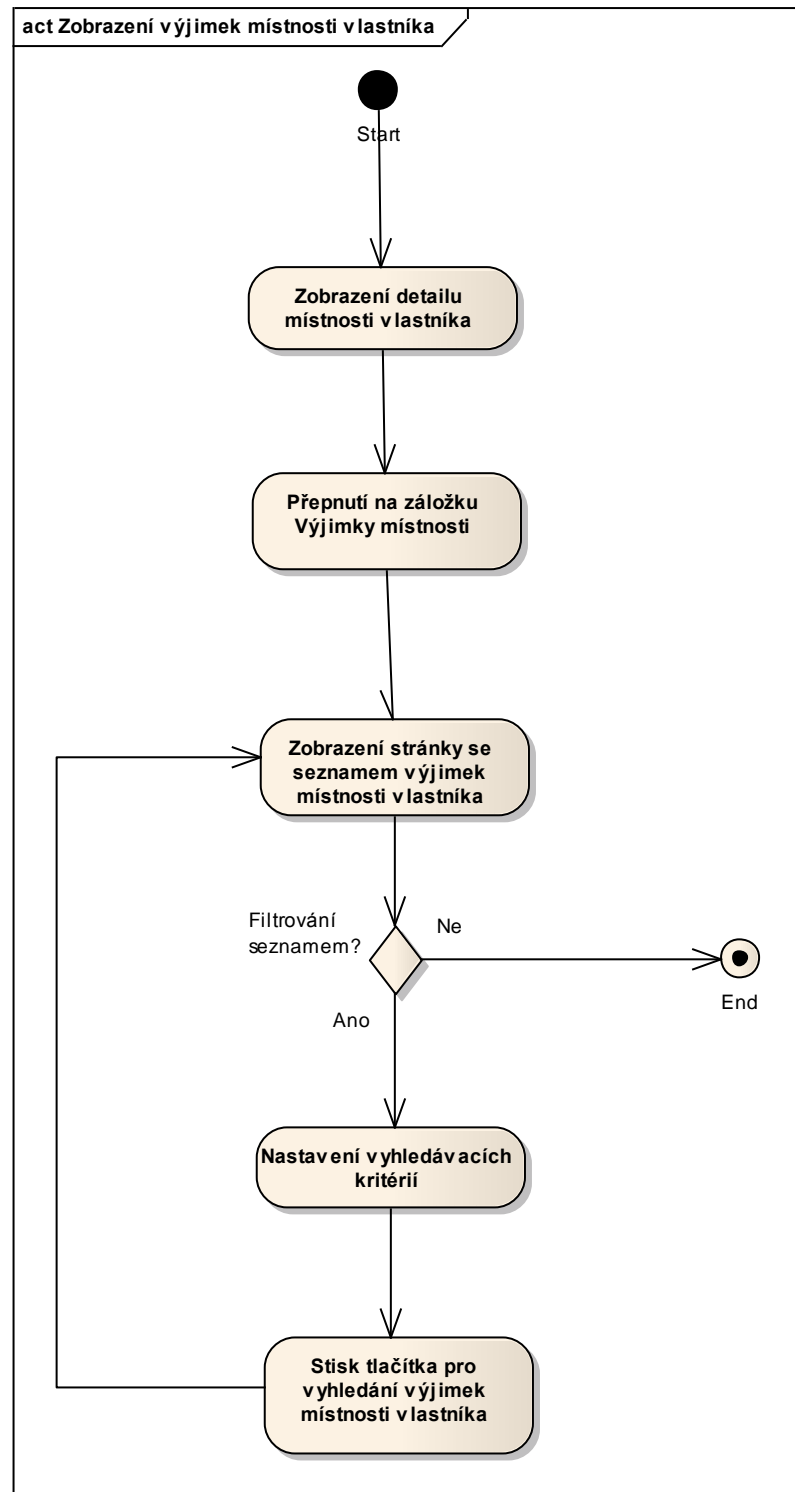
Obr. 3.26 – Diagram aktivit Odstranění zápůjčky



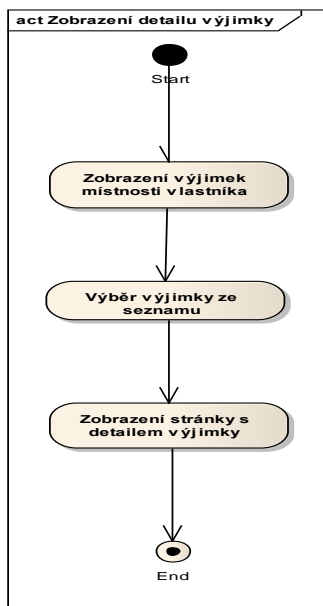
Obr. 3.27 – Diagram aktivit Editace detailu zápůjčky

### 3.4.3 Evidence výjimek

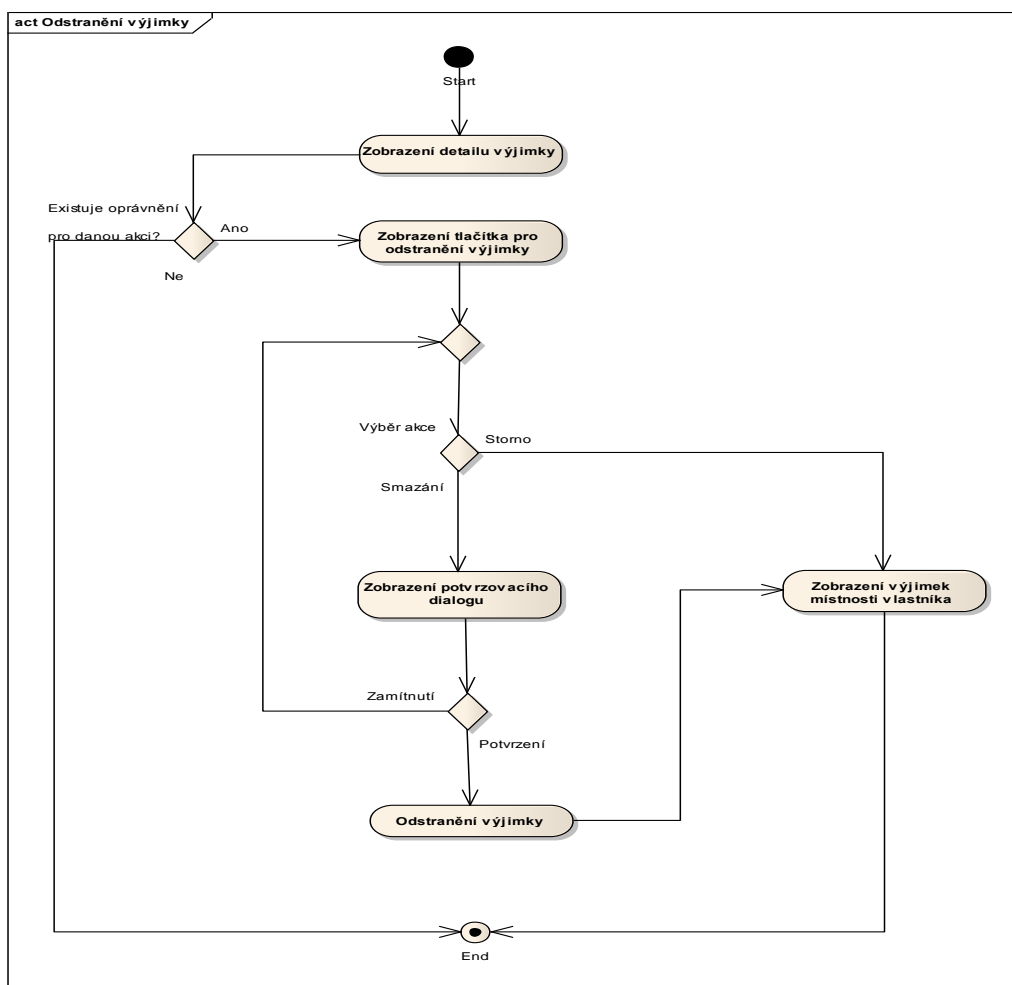
Následující diagramy popisují operace spojené s evidencí výjimek v systému.



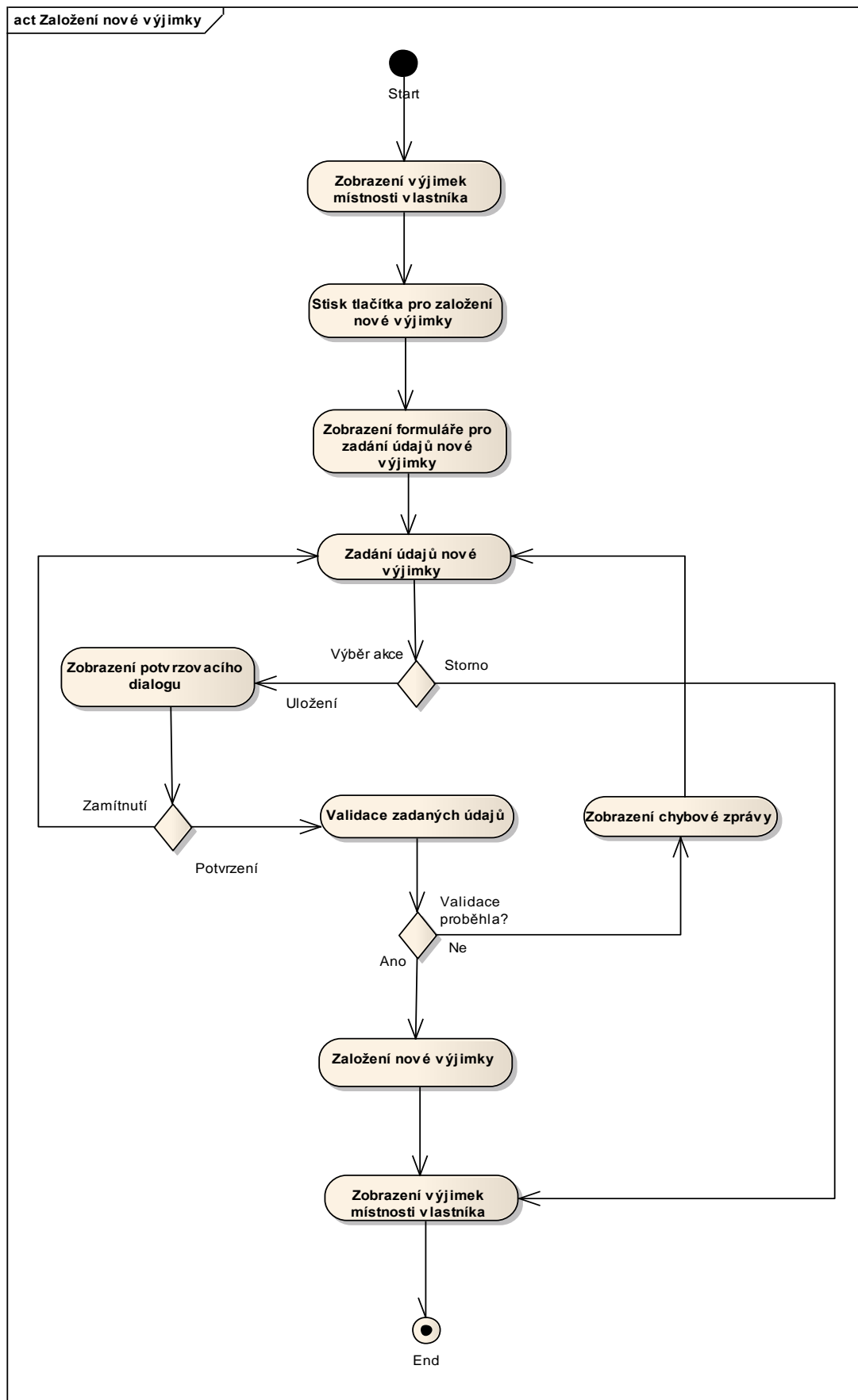
Obr. 3.28 – Diagram aktivit Zobrazení výjimek místnosti vlastníka



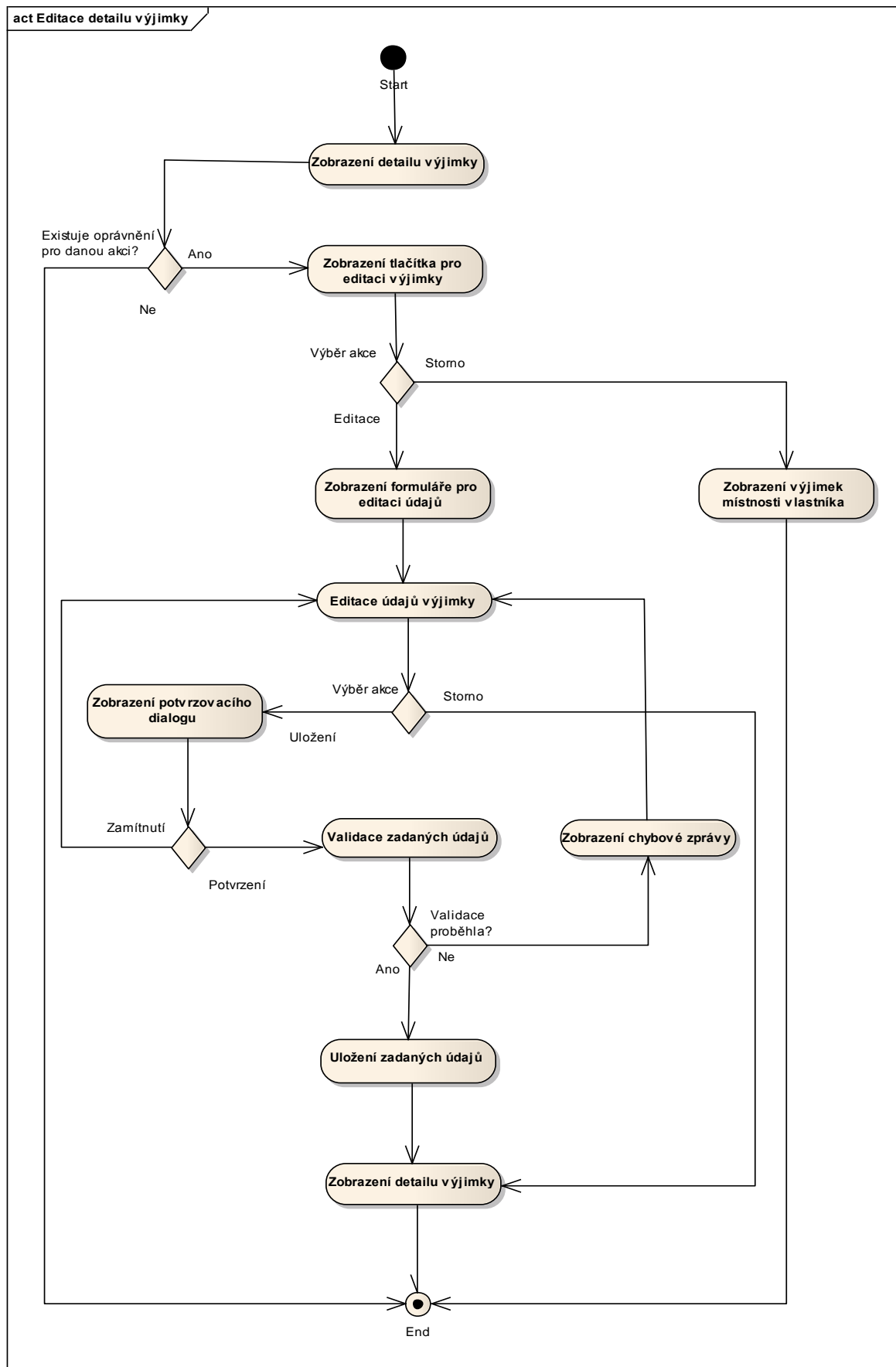
Obr. 3.29 – Diagram aktivit Zobrazení detailu výjimky



Obr. 3.30 – Diagram aktivit Odstranění výjimky



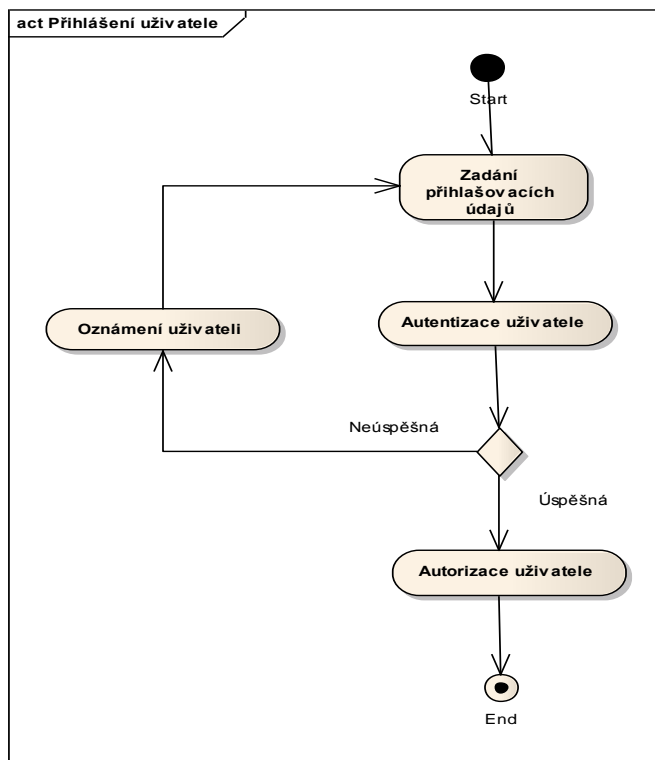
Obr. 3.31 – Diagram aktivit Založení nové výjimky



Obr. 3.32 – Diagram aktivit Editace detailu výjimky

### 3.4.4 Evidence uživatelů

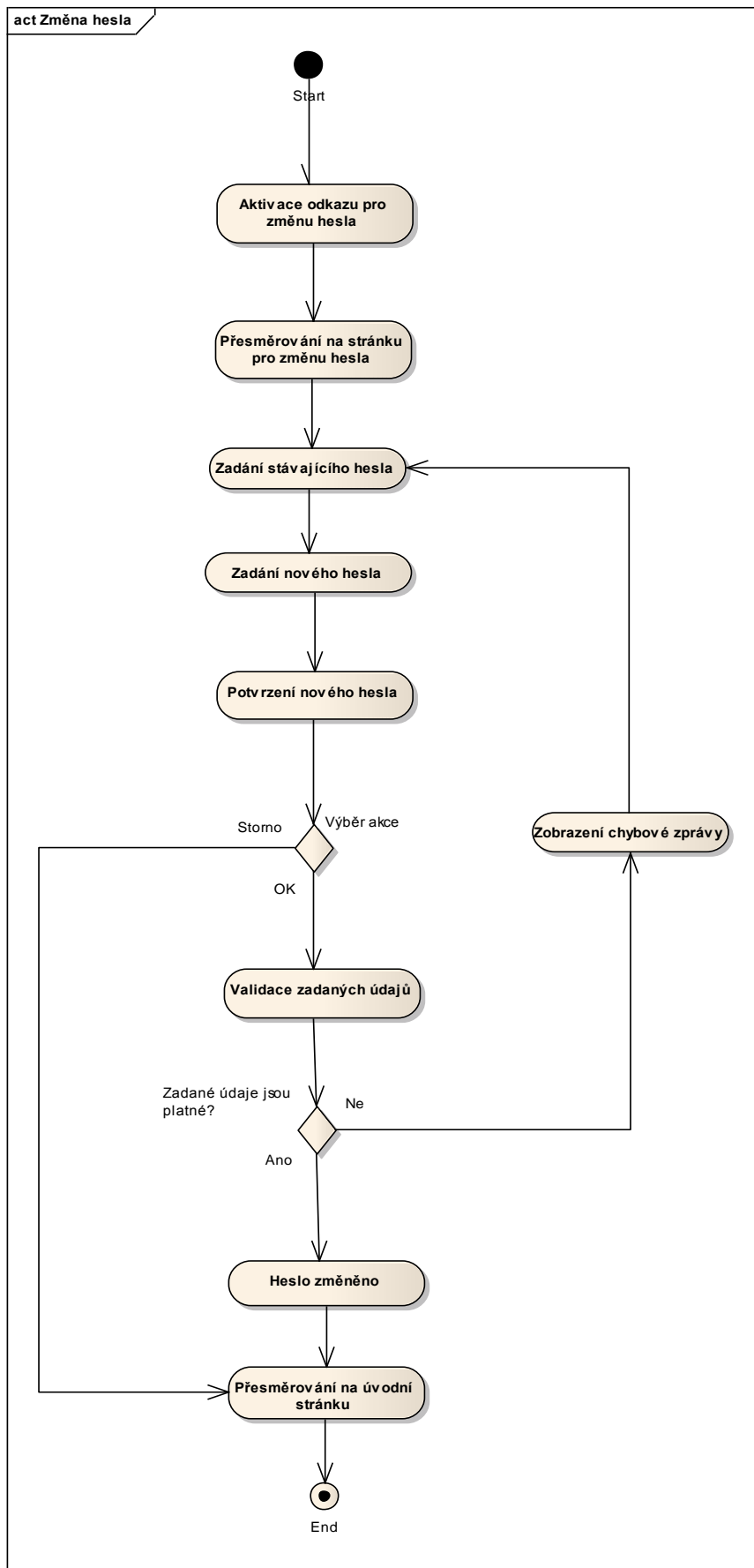
Následující diagramy popisují operace spojené s evidencí uživatelů v systému.



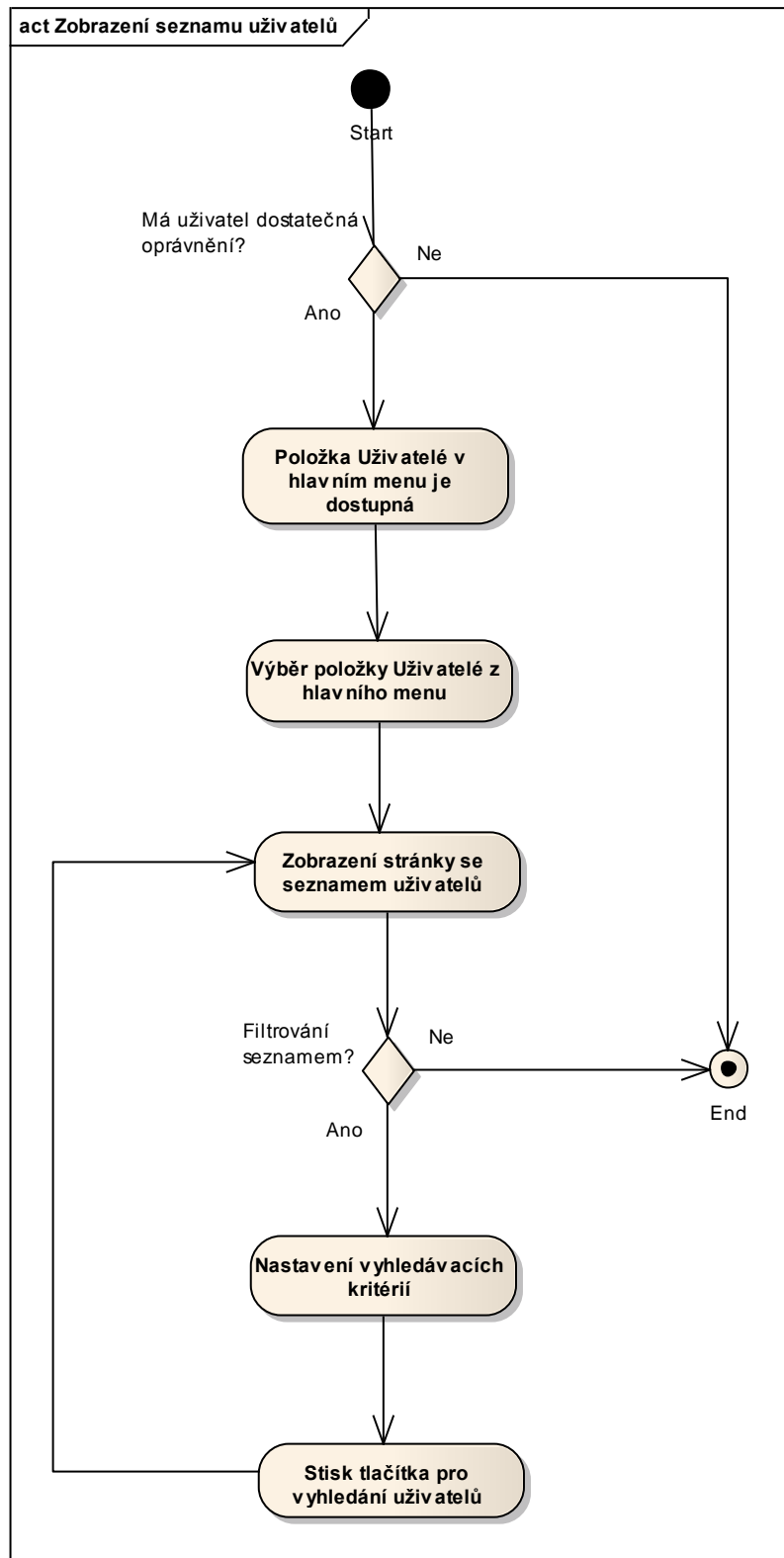
Obr. 3.33 – Diagram aktivit Přihlášení uživatele



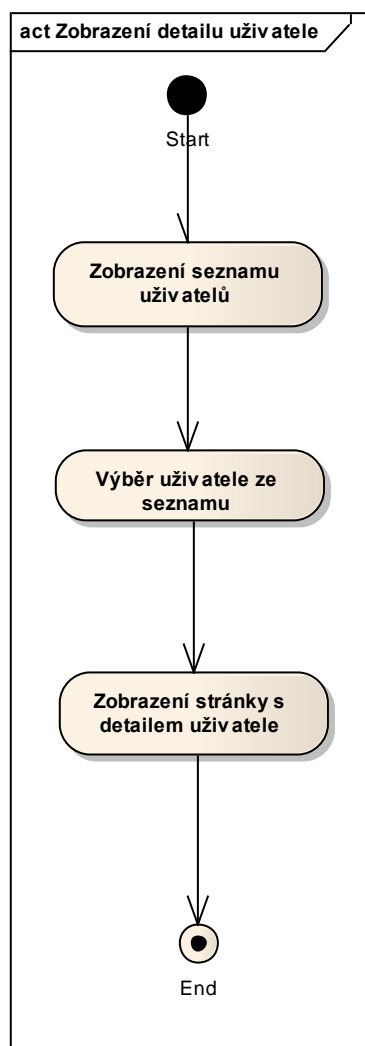
Obr. 3.34 – Diagram aktivit Odhlášení uživatele



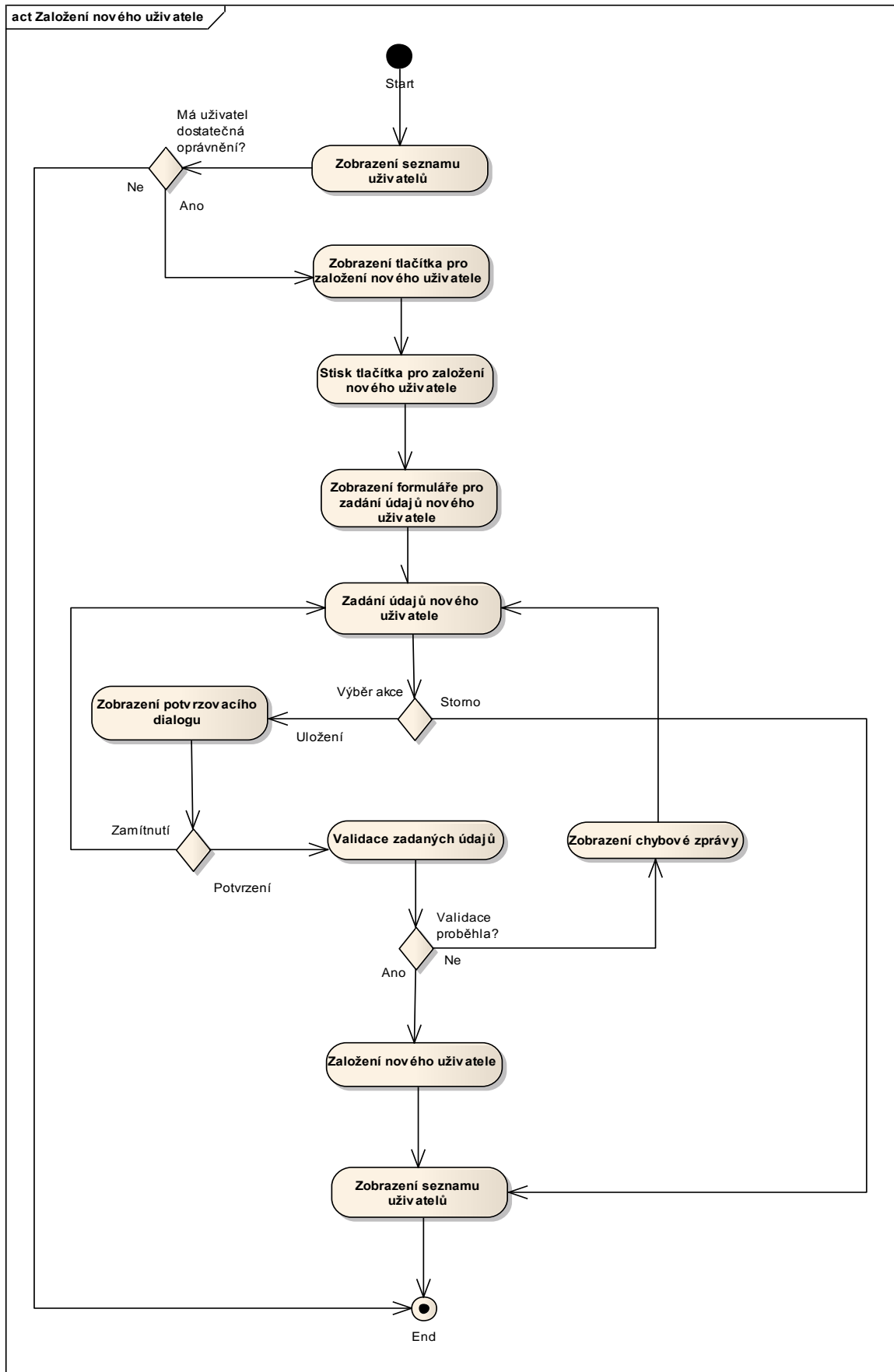
Obr. 3.35 – Diagram aktivit Změna hesla



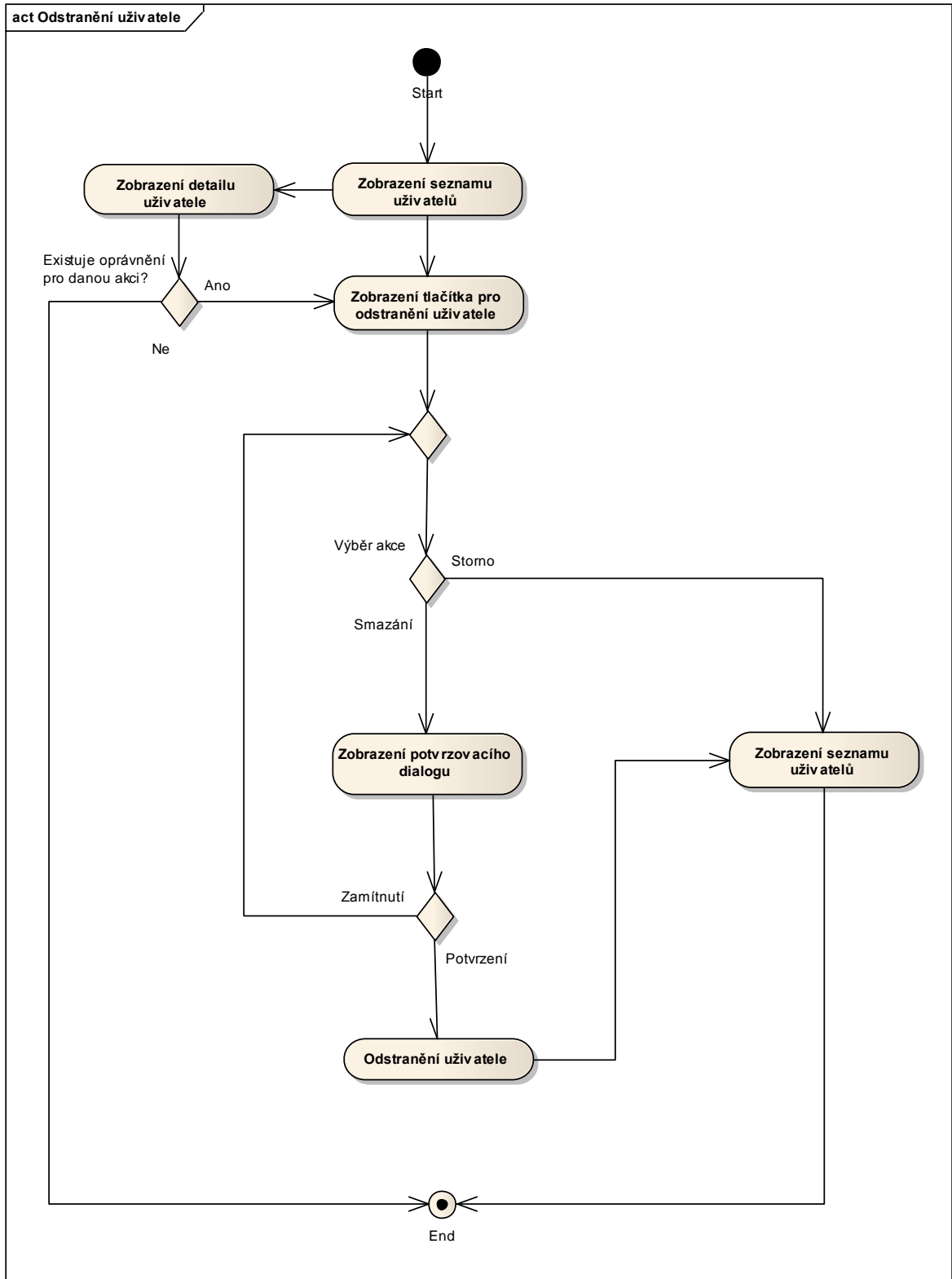
Obr. 3.36 – Diagram aktivit Zobrazení seznamu uživatelů



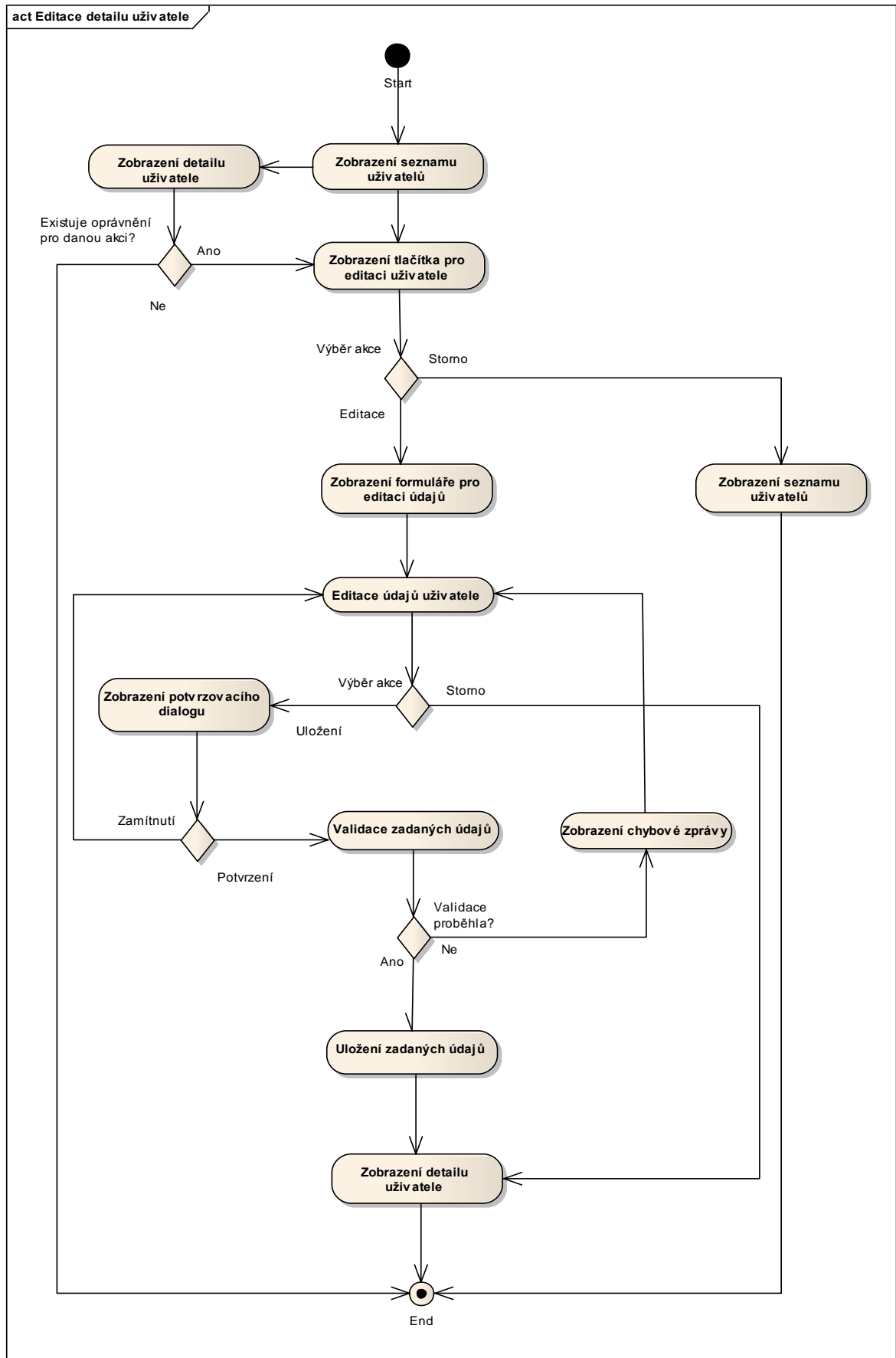
Obr. 3.37 – Diagram aktivit Zobrazení detailu uživatele



Obr. 3.38 – Diagram aktivit Založení nového uživatele



Obr. 3.39 – Diagram aktivit Odstranění uživatele



Obr. 3.40 – Diagram aktivit Editace detailu uživatele

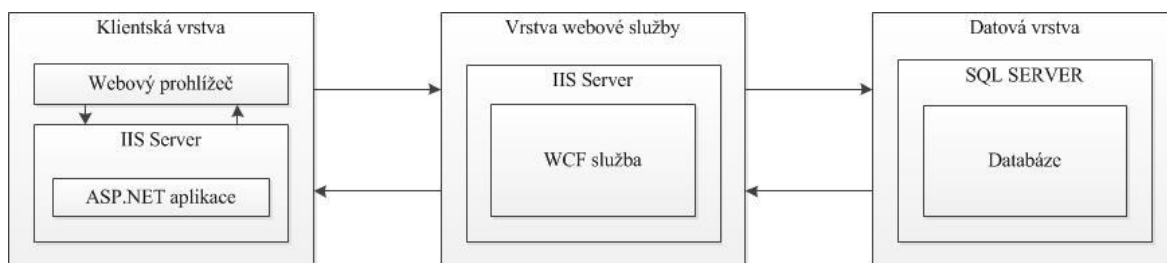
## **II. PRAKTICKÁ ČÁST**

## 4 ARCHITEKTURA APLIKACE

Aplikace je koncipována jako vícevrstvý systém. Základem je datová vrstva zastoupená databází běžící v rámci databázového serveru od společnosti Microsoft SQL Server. Při vytváření jednotlivých uložených procedur nad databází se použil procedurální databázový programovací jazyk T-SQL.

Nad datovou vrstvou je umístěna vrstva webové služby, která je prostředníkem mezi klientskou a datovou vrstvou. Konkrétněji se jedná o technologii WCF, jež je součástí platformy .NET Framework. Při vývoji služby a jejího klienta byl použit .NET Framework verze 4.5. Webová služba přijímá požadavky na data z klientské vrstvy a předává je do databáze. Získaná data z datové vrstvy jsou potom v patřičném formátu předávána zpět klientovi.

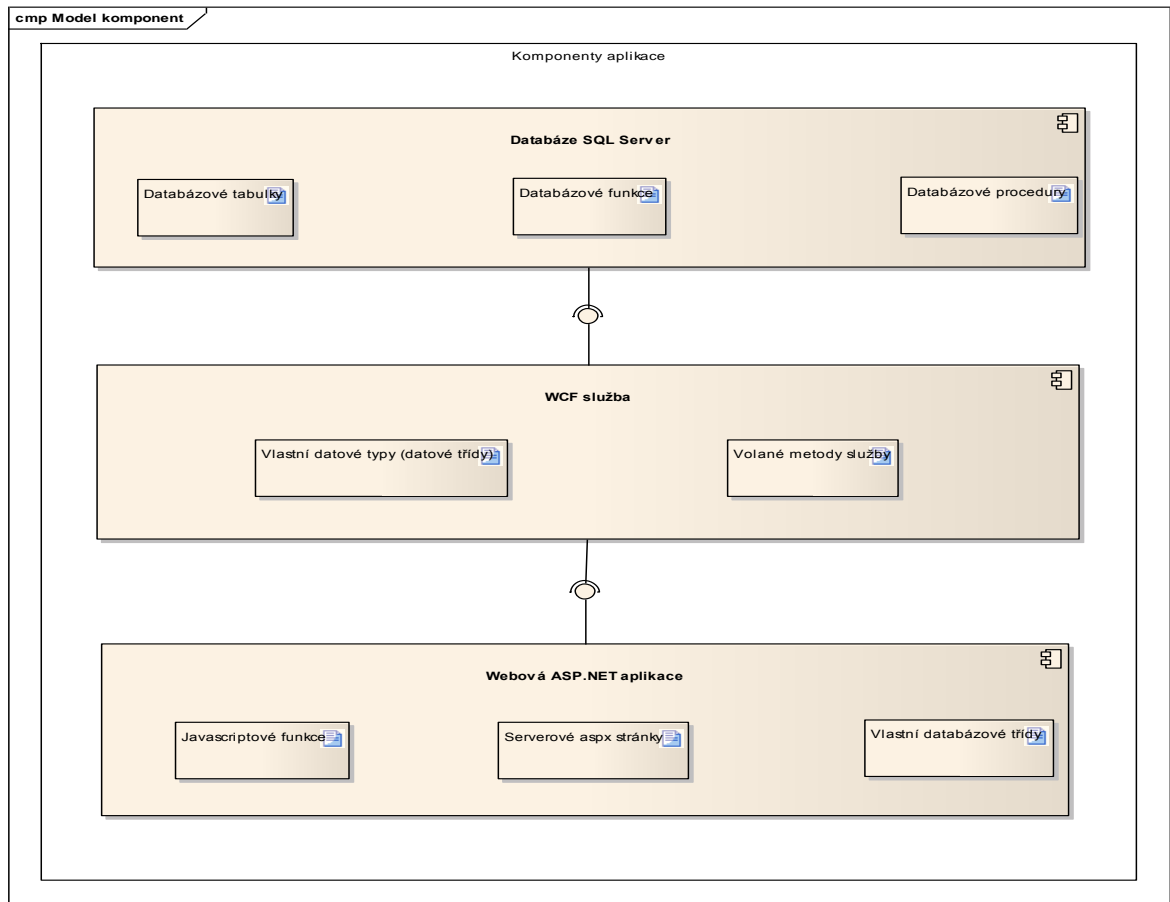
Klientskou vrstvu reprezentuje v tomto řešení webová aplikace. Pro tvorbu klienta byla zvolena technologie ASP.NET, opět součást .NET Frameworku. Samotná klientská část je v tomto případě dělena na 2 aplikační vrstvy. Serverová část má za úkol realizovat GUI aplikace a komunikaci s webovou službou. Serverový kód byl napsán v jazyce C#. Klientská část webové aplikace představuje několik funkcí napsaných ve skriptovacím jazyce Javascript. Jedná se o klientské validační funkce a několik dalších doplňkových funkcionalit na straně klienta. Pro nasazení klientské vrstvy i vrstvy služby bude použit webový server společnosti Microsoft IIS. Jako primární vývojové prostředí bylo v případě datové vrstvy zvoleno SQL Server Management Studio 2012, pro ostatní vrstvy aplikace potom Visual Studio 2013. Oba produkty pocházejí od společnosti Microsoft.



Obr. 4.1 – Architektura systému

Tab. 4.1 - Použité technologie

<b>Technologie</b>	<b>Popis</b>	<b>Použití</b>
SQL SERVER 2012	Databázový server	Datová vrstva
SQL Server Man. Studio 2012	Vývojové databázové prostředí	Datová vrstva
T-SQL	Programovací jazyk	Datová vrstva
.Net Framework 4.5	Programová platforma	Klientská vrstva, vrstva služby
Visual Studio 2013	Integrované vývojové prostředí	Klientská vrstva, vrstva služby
WCF služba	Webová služba	Vrstva služby
ASP.NET 4.5	Součást .Net pro vývoj webu	Klientská vrstva
IIS	Webový server	Klientská vrstva, vrstva služby
C#	Programovací jazyk	Klientská vrstva, vrstva služby
Javascript	Skriptovací jazyk	Klientská vrstva



Obr. 4.2 – Komponenty aplikace

## 5 POPIS ŘEŠENÍ

V této kapitole bude podrobněji popsán způsob, jakým byla aplikace vyvíjena. Bude se z větší části jednat o určitou specifikaci jednotlivých aplikačních vrstev, zahrnující v případě datové vrstvy vytvořené databázové struktury, jako jsou tabulky nebo uložené procedury, v případě vrstvy webové služby a vrstvy klienta potom programové struktury. Specifikace bude doplněna o technický popis komunikace mezi jednotlivými komponentami aplikace.

### 5.1 Datová vrstva

Základem celé aplikace je bezesporu datová vrstva, která bude uchovávat veškeré ukládané údaje. Úkolem této vrstvy je udržovat data aktuální, vždy v konzistentním stavu a správně strukturovaná tak, aby byl přístup k nim co nejefektivnější. Jedná se v tomto případě o relační databázi, běžící na databázovém serveru SQL Server. Využití řešení s databázovým serverem umožní vzdálený přístup k datové vrstvě, znamenající možnost tvorby aplikace typu klient / server, kdy jsou data centrálně přístupná v rámci databázového serveru a různý počet klientů má možnost k těmto datům souběžně přistupovat.

Samotná databáze se skládá z několika typů databázových struktur:

- **Tabulky**

Základní databázové struktury, uchovávající údaje o objektech určitého typu, v tomto řešení například budova, místnost, klíč, nebo osoba. Pro popis objektů slouží atributy tabulky, které vyjadřují jednotlivé vlastnosti objektu, jako je například název, platnost, stav, atd. Mezi tabulkami existují vztahy, realizované speciálními atributy, nazývanými jako primární a cizí klíče.

- **Uložené procedury**

Databázové programové struktury, obsahující kód v jazyce T-SQL. Uložené procedury jsou v této aplikaci jediným místem, přes které se přímo manipuluje s daty databáze. Uvnitř uložené procedury se kromě práce s daty provádí další důležité činnosti, jako je autorizace uživatele pro konkrétní operaci, nebo validace vstupních dat.

- **Databázové funkce**

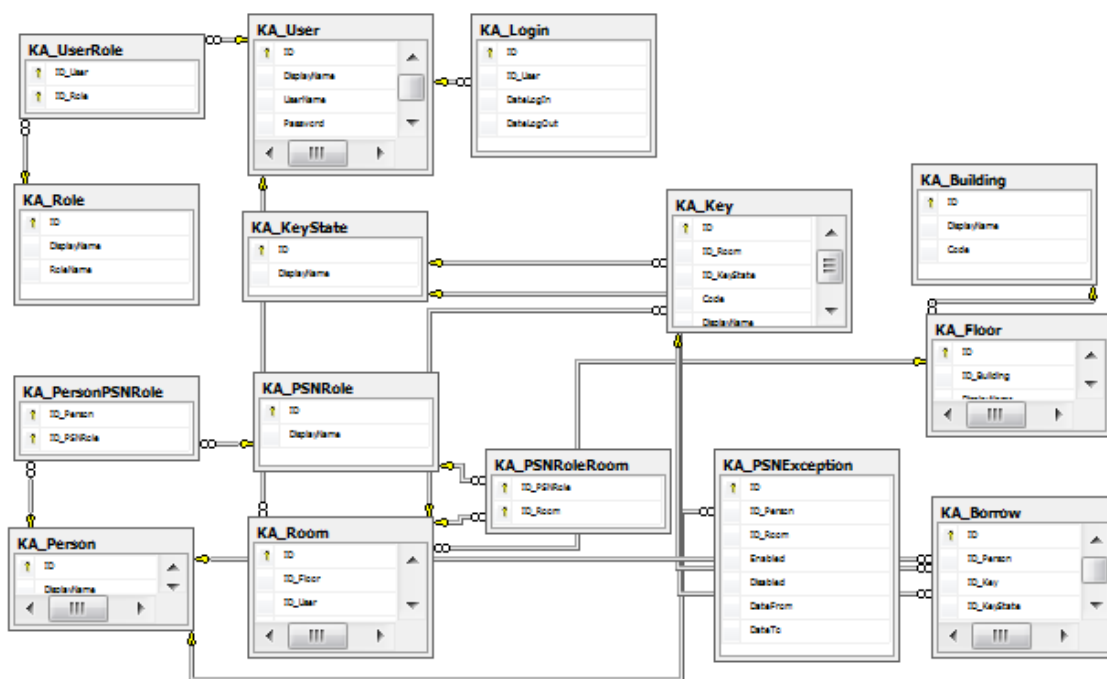
Další programové struktury, které po provedení svého kódu na rozdíl od procedur vrací konkrétní návratovou hodnotu. Může se jednat o skalární hodnotu různého datového typu, nebo o tabulkovou hodnotu. Jsou použity například pro ověření, zda je uživatel v konkrétní roli, nebo pro kontrolu zadávaného hesla uživatele.

- **Vlastní datové typy**

Kromě standardních vestavěných datových typů prostředí bylo vytvořeno a použito i několik vlastních datových typů.

### 5.1.1 Tabulky

Výsledkem databázového návrhu je následující databázové schéma:



Obr. 5.1 – Databázové schéma

Následuje soupis všech tabulek v databázi:

**KA\_User**

Tabulka, která udržuje údaje o všech uživatelích aplikace.

KA_User				
	Column Name	Condensed Type	Nullable	Description
🔑	ID	int	No	Primární klíč
	DisplayName	String:nvarchar...	No	Zobrazovaný název
	UserName	String:nvarchar...	No	Uživatelské jméno
	Password	String:nvarchar...	No	Heslo
	Email	String:nvarchar...	Yes	E-mail uživatele
	IsEnabled	bit	No	Stav

Obr. 5.2 – Tabulka KA\_User

**KA\_UserRole**

Spojovací tabulka mezi tabulkami KA\_User a KA\_Role.

KA_UserRole				
	Column Name	Condensed Type	Nullable	Description
🔑	ID_User	int	No	Cizí klíč do tabulky KA_User
🔑	ID_Role	int	No	Cizí klíč do tabulky KA_Role

Obr. 5.3 – Tabulka KA\_UserRole

**KA\_Role**

Tabulka, která udržuje údaje o všech uživatelských rolích aplikace.

KA_Role				
	Column Name	Condensed Type	Nullable	Description
🔑	ID	int	No	Primární klíč
	DisplayName	nvarchar(255)	No	Zobrazovaný název
	RoleName	nvarchar(255)	No	Jméno role

Obr. 5.4 – Tabulka KA\_Role

**KA\_Login**

Tabulka, která udržuje údaje o všech uživatelských přístupech do aplikace.

KA_Login				
	Column Name	Condensed Type	Nullable	Description
	ID	GUID:uniqueid...	No	Primární klíč
	ID_User	int	No	Cizí klíč do tabulky KA_User
	DateLogIn	datetime	No	Čas zalogování uživatele
	DateLogOut	datetime	Yes	Čas vypršení platnosti loginu uživatele

Obr. 5.5 – Tabulka KA\_Login

**KA\_Person**

Tabulka, která udržuje údaje o všech evidovaných osobách aplikace.

KA_Person				
	Column Name	Condensed Type	Nullable	Description
	ID	int	No	Primární klíč
	DisplayName	String:nvarch...	No	Zobrazovaný název
	Email	String:nvarch...	Yes	E-mail

Obr. 5.6 – Tabulka KA\_Person

**KA\_PersonPSNRole**

Spojovací tabulka mezi tabulkami KA\_Person a KA\_PSNRole.

KA_PersonPSNRole				
	Column Name	Condensed Type	Nullable	Description
	ID_Person	int	No	Cizí klíč do tabulky KA_Person
	ID_PSNRole	int	No	Cizí klíč do tabulky KA_PSNRole

Obr. 5.7 – Tabulka KA\_PersonPSNRole

**KA\_PSNRole**

Tabulka, která udržuje údaje o všech evidovaných rolích osob aplikace.

KA_PSNRole				
	Column Name	Condensed Type	Nullable	Description
🔑	ID	int	No	Primární klíč
	DisplayName	String:nvarch...	No	Zobrazovaný název

Obr. 5.8 – Tabulka KA\_PSNRole

**KA\_PSNRoleRoom**

Spojovací tabulka mezi tabulkami KA\_PSNRole a KA\_Room

KA_PSNRoleRoom				
	Column Name	Condensed Type	Nullable	Description
🔑	ID_PSNRole	int	No	Cizí klíč do tabulky KA_PSNRole
🔑	ID_Room	int	No	Cizí klíč do tabulky KA_Room

Obr. 5.9 – Tabulka KA\_PSNRoleRoom

**KA\_Building**

Tabulka, která udržuje údaje o všech evidovaných budovách aplikace.

KA_Building				
	Column Name	Condensed Type	Nullable	Description
🔑	ID	int	No	Primární klíč
	DisplayName	String:nvarch...	No	Zobrazovaný název
	Code	String:nvarch...	No	Kód

Obr. 5.10 – Tabulka KA\_Building

**KA\_Floor**

Tabulka, která udržuje údaje o všech evidovaných patrech aplikace.

KA_Floor				
	Column Name	Condensed Type	Nullable	Description
🔑	ID	int	No	Primární klíč
	ID_Building	int	No	Cizí klíč do tabulky KA_Building
	DisplayName	String:nvarch...	No	Zobrazovaný název
	Code	String:nvarch...	No	Kód

Obr. 5.11 – Tabulka KA\_Floor

**KA\_Room**

Tabulka, která udržuje údaje o všech evidovaných místnostech aplikace.

KA_Room				
	Column Name	Condensed Type	Nullable	Description
🔑	ID	int	No	Primární klíč
	ID_Floor	int	No	Cizí klíč do tabulky KA_Floor
	ID_User	int	Yes	Cizí klíč do tabulky KA_User
	Code	String:nvarch...	No	Kód
	DisplayName	String:nvarch...	No	Zobrazovaný název

Obr. 5.12 – Tabulka KA\_Room

**KA\_Key**


Tabulka, která udržuje údaje o všech evidovaných klíčích aplikace.

KA_Key				
	Column Name	Condensed Type	Nullable	Description
🔑	ID	int	No	Primární klíč
	ID_Room	int	No	Cizí klíč do tabulky KA_Room
	ID_KeyState	String:nvarch...	No	Cizí klíč do tabulky KA_KeyState
	Code	String:nvarch...	No	Kód
	DisplayName	String:nvarch...	No	Zobrazovaný název

Obr. 5.13 – Tabulka KA\_Key

**KA\_KeyState**


Tabulka, která udržuje údaje o všech možných stavech klíče a zápůjčky.

KA_KeyState				
	Column Name	Condensed Type	Nullable	Description
	 ID	String:nvarch...	No	Primární klíč
	DisplayName	String:nvarch...	No	Zobrazovaný název

Obr. 5.14 – Tabulka KA\_KeyState

**KA\_Borrow**


Tabulka, která udržuje údaje o všech evidovaných zápůjčkách klíčů aplikace.

KA_Borrow				
	Column Name	Condensed Type	Nullable	Description
	 ID	int	No	Primární klíč
	ID_Person	int	No	Cizí klíč do tabulky KA_Person
	ID_Key	int	No	Cizí klíč do tabulky KA_Key
	ID_KeyState	String:nvarch...	No	Cizí klíč do tabulky KA_KeyState
	DateFrom	datetime	No	Datum zapůjčení klíče
	DateTo	datetime	Yes	Datum vrácení klíče

Obr. 5.15 – Tabulka KA\_Borrow

**KA\_PSNEException**

Tabulka, která udržuje údaje o všech evidovaných výjimkách místností aplikace.

KA_PSNEException				
	Column Name	Condensed Type	Nullable	Description
	 ID	int	No	Primární klíč
	ID_Person	int	No	Cizí klíč do tabulky KA_Person
	ID_Room	int	No	Cizí klíč do tabulky KA_Room
	Enabled	bit	No	Zda se jedná o povolovací pravidlo
	Disabled	bit	No	Zda se jedná o zakazovací pravidlo
	DateFrom	datetime	No	Začátek platnosti
	DateTo	datetime	Yes	Konec platnosti

Obr. 5.16 – Tabulka KA\_PSNEException

### 5.1.2 Uložené procedury

Ve velké většině vytvořených uložených procedur se vyžaduje, aby byly volány v kontextu přihlášeného uživatele. To je zajištěno následujícím procesem: Při úspěšném přihlášení uživatele je do tabulky KA\_Login založen nový záznam s jednoznačným globálním identifikátorem, podle kterého se rozlišují jednotliví přihlášení uživatelé. Tato hodnota je po vytvoření vrácena zpět klientovi a opakovaně potom přikládána ke každému volání procedury, jako jeden z jejích vstupních parametrů. V proceduře se potom ověřuje, zda jde o platný uživatelský login. Každý záznam loginu má nastavenou omezenou časovou platnost. Ta zaručuje, že při delší nečinnosti ze strany uživatele dojde k jeho zneplatnění. Pokud se login vyhodnotí jako neplatný, je v proceduře vygenerována výjimka a další kód se již neprovádí. Je na vyšších vrstvách aplikace, jak vygenerovanou výjimku zpracují. V případě webové aplikace dojde k odhlášení uživatele a přesměrování na přihlašovací stránku. Pro ověření loginu je použita uživatelská funkce *IsValidLogin*.

```
ALTER PROCEDURE [dbo].[KA_Building_New]
    @ID_Login GUID,
    @Code String,
    @DisplayName String
AS
BEGIN
    declare @Message String
    declare @Error int
    set @Message = 'Při založení budovy došlo k chybě!'

    --zjištění, zda je uživatel přihlášen
    if (dbo.IsValidLogin(@ID_Login) = '0')
    begin
        set @Message = 'Uživatel byl odhlášen!'
        goto FAILED
    end
end
```

Obr. 5.17- Ukázka ověření přihlášeného uživatele na začátku procedury

V dalším kroku procedury dochází k ověření uživatelských oprávnění k provedení příslušné operace. Taktéž na základě přiloženého loginu. Například založit novou budovu může jen administrátor. V proceduře pro založení budovy je tedy potřeba ověřit, zda pod aktuálním loginem vystupuje uživatel s příslušnými právy. Pokud jsou oprávnění uživatele

nedostačující, opět je vygenerována výjimka a procedura tímto končí. Pro ověření oprávnění uživatele je použita uživatelská funkce *IsUserInRole*.

```
--zjištění, zda je uživatel v roli admin
if dbo.IsUserInRole(@ID_Login, 'admin') = '0'
begin
    set @Message =
        'Uživatel nemá dostatečná oprávnění k provedení požadované operace!'
    goto FAILED
end
```

Obr. 5.18 – Ukázka ověření práv uživatele pro provedení požadované operace

V některých procedurách je také potřeba realizovat určitou validaci vstupních dat. V případě zakládání nového objektu budovy je důležité zkontrolovat, zda již stejný kód není přidělen některé z existujících budov. Hodnota tohoto atributu musí být v rámci evidovaných budov jedinečná. Pokud nejsou vstupní data validní, opět se vygeneruje výjimka s příslušnou chybovou zprávou a je na klientské aplikaci, jak s ní bude dále zacházet.

```
--zjištění, zda je kód jedinečný
if exists(select * from KA_Building where KA_Building.Code = @Code)
begin
    set @Message = 'Budova již existuje!'
    goto FAILED
end
```

Obr. 5.19 – Ukázka validace vstupního parametru procedury

Pokud jsou vstupní data validní, může se konečně provést samotná požadovaná operace s daty. V proceduře pro založení budovy se SQL příkazem INSERT v tabulce KA\_Building vytvoří nový záznam.

```
insert into KA_Building values (@DisplayName, @Code)

if @@error <> 0
    goto FAILED
```

Obr. 5.20 – Ukázka založení nového záznamu do tabulky KA\_Building

Po provedení potřebné akce, jako je založení nebo odstranění záznamu, je ještě na konci procedury potřeba prodloužit platnost loginu na výchozí časový limit a udržovat tak uživatele přihlášeného. Pro prodloužení platnosti loginu je použita uložená procedura KA\_ExtendLogin.

```
--prodloužení platnosti loginu
exec @Error = KA_ExtendLogin @ID_Login
if @Error <> 0
    goto FAILED
```

Obr. 5.21 – Ukázka prodloužení platnosti loginu po provedení požadované operace v uložené proceduře

U jednotlivých entit aplikace jsou zakládány procedury pro každou ze základních implementovaných operací.

- **Zobrazení seznamu záznamů**

Operace, která zobrazí seznam evidovaných objektů požadovaného typu. Seznam lze filtrovat podle přiřazených atributů, nebo podle příslušnosti k nadřazeným entitám. Tyto informace jsou do procedury dodány prostřednictvím vstupních parametrů.

```
ALTER PROCEDURE [dbo].[KA_Borrow_All]
    @ID_Login GUID,
    @ID_Person int = null,
    @ID_Building int = null,
    @ID_Floor int = null,
    @ID_Room int = null,
    @ID_Key int = null,
    @ID_KeyState String = null,
    @DateFromFilter datetime = null,
    @DateToFilter datetime = null
AS
BEGIN
```

Obr. 5.22 – Ukázka definice vstupních parametrů procedury KA\_Borrow\_All pro zobrazení seznamu zápůjček

```

select
    KA_Borrow.ID,
    'Building' = KA_Building.DisplayName,
    'Floor' = KA_Floor.DisplayName,
    'Room' = KA_Room.DisplayName,
    'Key' = KA_Key.DisplayName,
    'Person' = KA_Person.DisplayName,
    'State' = KA_KeyState.DisplayName,
    KA_Borrow.DateFrom,
    KA_Borrow.DateTo
from
    KA_Borrow
    inner join KA_KeyState on KA_Borrow.ID_KeyState = KA_KeyState.ID
    inner join KA_Person on KA_Borrow.ID_Person = KA_Person.ID
    inner join KA_Key on KA_Borrow.ID_Key = KA_Key.ID
    inner join KA_Room on KA_Key.ID_Room = KA_Room.ID
    inner join KA_Floor on KA_Room.ID_Floor = KA_Floor.ID
    inner join KA_Building on KA_Floor.ID_Building = KA_Building.ID
where
    (KA_Building.ID = @ID_Building or @ID_Building is null)
    and (KA_Floor.ID = @ID_Floor or @ID_Floor is null)
    and (KA_Room.ID = @ID_Room or @ID_Room is null)
    and (KA_Key.ID = @ID_Key or @ID_Key is null)
    and (KA_Borrow.ID_KeyState = @ID_KeyState or @ID_KeyState is null)
    and (KA_Borrow.ID_Person = @ID_Person or @ID_Person is null)
    and (KA_Borrow.DateFrom >= @DateFromFilter or @DateFromFilter is null)
    and (KA_Borrow.DateFrom <= @DateToFilter or @DateToFilter is null)
order by KA_Borrow.DateFrom

```

Obr. 5.23 – Ukázka SQL příkazu v proceduře KA\_Borrow\_All pro zobrazení seznamu zápůjček na základě dodaných vstupních parametrů

- **Zobrazení detailu záznamu**

Operace pro zobrazení detailních informací jediného objektu. Do procedury vstupuje ve formě parametru kromě loginu hodnota ID záznamu, jehož detail je zobrazován.

```

ALTER PROCEDURE [dbo].[KA_Borrow_Detail]
    @ID_Login GUID,
    @ID int
AS
BEGIN

```

Obr. 5.24 - Ukázka definice vstupních parametrů procedury KA\_Borrow\_Detail pro zobrazení detailu zápůjčky

```

select
    KA_Borrow.ID,
    KA_Borrow.ID_KeyState,
    'Building' = KA_Building.DisplayName,
    'Floor' = KA_Floor.DisplayName,
    'Room' = KA_Room.DisplayName,
    'Key' = KA_Key.DisplayName,
    'Person' = KA_Person.DisplayName,
    'State' = KA_KeyState.DisplayName,
    KA_Borrow.DateFrom,
    KA_Borrow.DateTo
from
    KA_Borrow
    inner join KA_KeyState on KA_Borrow.ID_KeyState = KA_KeyState.ID
    inner join KA_Person on KA_Borrow.ID_Person = KA_Person.ID
    inner join KA_Key on KA_Borrow.ID_Key = KA_Key.ID
    inner join KA_Room on KA_Key.ID_Room = KA_Room.ID
    inner join KA_Floor on KA_Room.ID_Floor = KA_Floor.ID
    inner join KA_Building on KA_Floor.ID_Building = KA_Building.ID
where
    KA_Borrow.ID = @ID

```

Obr. 5.25 - Ukázka SQL příkazu v proceduře KA\_Borrow\_Detail pro zobrazení detailu zápůjčky na základě dodaných vstupních parametrů

- **Založení nového záznamu**

Operace pro přidání nového objektu do evidence. Formou vstupních parametrů jsou proceduře předány všechny informace o objektu, včetně hodnot cizích klíčů do nadřazených objektů.

```

ALTER PROCEDURE [dbo].[KA_Borrow_New]
    @ID_Login GUID,
    @ID_Person int,
    @ID_Key int
AS
BEGIN

```

Obr. 5.26 - Ukázka definice vstupních parametrů procedury KA\_Borrow\_New pro vytvoření nové zápůjčky

```

insert into KA_Borrow values (@ID_Person, @ID_Key, 'vypujceno', GETDATE(), null);

```

Obr. 5.27 - Ukázka SQL příkazu v proceduře KA\_Borrow\_New pro vytvoření nové zápůjčky

- **Aktualizace záznamu**

Operace pro aktualizaci stávajících údajů objektu. Do procedury jsou vstupními parametry předány všechny nové hodnoty objektu. Pro úpravu záznamu je také potřeba do procedury předat hodnotu primárního klíče, tedy ID.

```
ALTER PROCEDURE [dbo].[KA_Borrow_Edit]
    @ID_Login GUID,
    @ID int,
    @ID_State String
AS
BEGIN
```

Obr. 5.28 - Ukázka definice vstupních parametrů procedury KA\_Borrow\_Edit pro aktualizaci zápůjčky

```
--aktualizace stavu u zápůjčky i klíče
update KA_Borrow
    set ID_KeyState = @ID_State
where KA_Borrow.ID = @ID

update KA_Key
    set ID_KeyState = @ID_State
where KA_Key.ID = (select KA_Borrow.ID_Key from KA_Borrow where KA_Borrow.ID = @ID)
```

Obr. 5.29 - Ukázka SQL příkazu v proceduře KA\_Borrow\_Edit pro aktualizaci zápůjčky

- **Smazání záznamu**

Operace pro odstranění objektu z evidence. Do procedury je potřeba předat ID mazaného objektu. V případě, že se maže nadřazený objekt, součástí operace odstranění bude i smazání všech souvisejících podřízených objektů.

```
ALTER PROCEDURE [dbo].[KA_Borrow_Del]
    @ID_Login GUID,
    @ID int
AS
BEGIN
```

Obr. 5.30 - Ukázka definice vstupních parametrů procedury KA\_Borrow\_Del pro smazání zápůjčky

```
--smazání zápůjčky  
delete from KA_Borrow where KA_Borrow.ID = @ID
```

Obr. 5.31 - Ukázka SQL příkazu v proceduře KA\_Borrow\_Del pro smazání zápůjčky

Procedury operací nad objekty ostatních typů jsou v principu až na výjimky velmi podobné. Kromě uložených procedur pro zajištění základních operací s daty obsahuje databáze ještě několik dalších, jejichž použití je různé.

- **KA\_ExtendLogin**

Uložená procedura pro prodloužení platnosti loginu.

- **KA\_ChangePassword**

Uložená procedura pro změnu hesla uživatele.

- **KA\_Person\_All\_Names**

Uložená procedura pro zobrazení jmen všech evidovaných osob. Nevyžaduje volání v kontextu přihlášeného uživatele.

- **KA\_Role\_All\_Auth**

Uložená procedura, která vrací seznam přiřazených rolí uživatele. Nevyžaduje volání v kontextu přihlášeného uživatele.

- **KA\_User\_Detail\_Auth**

Uložená procedura, která vrací detail uživatele. Nevyžaduje volání v kontextu přihlášeného uživatele.

- **KA\_UserLogIn**

Uložená procedura vykonávající operaci přihlášení uživatele.

- **KA\_UserLogOut**

Uložená procedura, vykonávající operaci odhlášení uživatele.

- **KA\_ValidPassword**

Uložená procedura pro validaci hesla uživatele.

### 5.1.3 Uživatelské funkce

- **SplitStrings**

Funkce pro rozdělení řetězce na jednotlivé podřetězce na základě zadaného separátoru. Výstupem je tabulka nalezených podřetězců hlavního řetězce.

- **JoinStrings**

Opačná funkce k funkci *SplitStrings*. Na základě dodaného seznamu řetězců a separátoru vrací jeden řetězec složený z položek seznamu, které jsou odděleny separátorem. Parametr vstupního seznamu je uživatelského datového typu *StringList*.

- **GetUserID**

Vrací ID uživatele na základě dodaného loginu.

- **IsUserInRole**

Pro zadaný login a název role vrací '1', pokud je uživatel v této roli. Jinak vrací '0'.

- **IsValidLogin**

Funkce pro kontrolu platnosti zadaného loginu. Je volána skoro ve všech uložených procedurách.

- **IsValidPassword**

Funkce pro ověření správnosti zadaného hesla. Ověřuje se u uživatele, ke kterému se vztahuje login, který je spolu s heslem vstupním parametrem funkce.

### 5.1.4 Uživatelské datové typy

- **GUID**

Globální identifikátor, použitý pro ukládání jednotlivých uživatelských loginů.

- **String**

Typ, který přehledněji vyjadřuje interní datový typ *nvarchar(255)*.

- **StringList**

Typ pro uchování seznamu řetězců.

## 5.2 Vrstva webové služby

Pro realizaci vrstvy webové služby byla zvolena technologie WCF. Programově se jedná o jednotlivé metody služby, které jsou volatelné klientem služby. V tomto případě se volají z webové aplikace. Metody volají ve svém těle příslušné uložené procedury v databázi a získaná data potom předávají dále vyšší vrstvě. Pro každou požadovanou operaci s daty tedy existuje odpovídající dvojice *metoda služby – uložená procedura*. Volání uložených procedur se odehrává uvnitř chráněného bloku pro ošetření případných vzniklých výjimek při práci s datovou vrstvou. Každá výjimka je transformována na vyvolání nové výjimky typu *FaultException*. Tento mechanismus zaručí možnost bezproblémového ošetření takto vygenerované výjimky na straně klientského kódu. Pro práci s databází využívá služba vlastní programovou třídu, která v sobě zapouzdřuje standardní komponenty prostředí .NET pro přístup k datům – *SqlConnection* a *SqlCommand*. Při vytvoření instance této třídy je připojovací řetězec k databázi automaticky načten z konfiguračního souboru služby, je otevřeno připojení k databázi a typ příkazu je nastaven na volání uložených procedur. Práce s touto třídou je mnohem rychlejší než běžný přístup a při častém používání významně uspoří čas při vývoji softwaru. Z datové vrstvy dostává služba data ve formě iterovatelné struktury a v každém procházeném cyklu získává jeden řádek výsledné množiny. K hodnotám jednotlivých atributů záznamu se potom přistupuje na základě jejich názvů. Pořízená data ve formě skalárních hodnot je ovšem docela těžkopádné a nepřehledné předávat dále v nezměněné podobě. Pro přenos dat mezi službou a jejím klientem jsou místo toho použity datové třídy. Pro každý přenášený záznam určitého typu (místnost, klíč..) je vytvořena instance odpovídající třídy a zastává tak roli datové obálky záznamu. Vlastnosti objektu jsou pak plněny získanými hodnotami atributů záznamu. V případě, kdy má metoda webové služby vrátit klientovi celou množinu záznamů, výsledkem volání je kolekce objektů daného typu. Tento způsob předávání informací mezi klientem a službou je přehledný a programový kód řešící tuto funkcionalitu je snadno udržovatelný.

### 5.2.1 Metody webové služby pro základní operace s daty

Scénáře metod pro zajištění základních operací s daty jsou v případě jednotlivých evidovaných aplikačních entit velmi podobné. Společným prvkem je dodání hodnoty loginu přihlášeného uživatele přes vstupní parametr do metody z klientské vrstvy aplikace, který je poté dále předáván do příslušné uložené procedury, taktéž ve formě hodnoty

vstupního parametru. Metody pro základní operace s daty budou předvedeny na příkladu práce s objekty klíčů.

- **Založení nového objektu**

Při volání metody pro založení objektu je přes vstupní parametry předán login uživatele a zakládáný objekt.

```
//založení nového klíče
public void AddKey(string idLogin, Key key)
{
    try
    {
        using (var db = new Db("KA_Key_New"))
        {
            db.Parameters.AddWithValue("ID_Login", idLogin);
            db.Parameters.AddWithValue("ID_Room", key.ID_Room);
            db.Parameters.AddWithValue("ID_KeyState", key.ID_KeyState);
            db.Parameters.AddWithValue("Code", key.Code);
            db.Parameters.AddWithValue("DisplayName", key.DisplayName);

            db.ExecuteNonQuery();
        }
    }
    catch (Exception exp)
    {
        throw new FaultException(exp.Message);
    }
}
```

Obr. 5.32 - Ukázka metody webové služby pro založení nového klíče

- **Odstranění objektu**

Při volání metody pro odstranění objektu je přes vstupní parametry předán login uživatele a hodnota id mazaného objektu.

```
//smazání klíče
public void DeleteKey(string idLogin, int id)
{
    try
    {
        using (var db = new Db("KA_Key_Del"))
        {
            db.Parameters.AddWithValue("ID_Login", idLogin);
            db.Parameters.AddWithValue("ID", id);

            db.ExecuteNonQuery();
        }
    }
    catch (Exception exp)
    {
        throw new FaultException(exp.Message);
    }
}
```

Obr. 5.33 - Ukázka metody webové služby pro odstranění klíče

- **Aktualizace objektu**

Při volání metody pro aktualizaci objektu je přes vstupní parametry předán login uživatele a aktualizovaný objekt.

```
//editace klíče
public void EditKey(string idLogin, Key key)
{
    try
    {
        using (var db = new Db("KA_Key_Edit"))
        {
            db.Parameters.AddWithValue("ID_Login", idLogin);
            db.Parameters.AddWithValue("ID", key.ID);
            db.Parameters.AddWithValue("Code", key.Code);
            db.Parameters.AddWithValue("DisplayName", key.DisplayName);

            db.ExecuteNonQuery();
        }
    }
    catch (Exception exp)
    {
        throw new FaultException(exp.Message);
    }
}
```

Obr. 5.34 - Ukázka metody webové služby pro aktualizaci klíče

- **Zobrazení detailu objektu**

Při volání metody pro zobrazení detailu objektu je přes vstupní parametry předán login uživatele a id požadovaného objektu. Návrátovou hodnotou metody je zobrazovaný objekt.

```
//detail klíče
public Key KeyDetail(string idLogin, int id)
{
    try
    {
        using (var db = new Db("KA_Key_Detail"))
        {
            db.Parameters.AddWithValue("ID_Login", idLogin);
            db.Parameters.AddWithValue("ID", id);

            SqlDataReader reader = db.ExecuteReader();

            reader.Read();

            Key key = new Key()
            {
                ID = (Int32)reader["ID"],
                ID_Room = (Int32)reader["ID_Room"],
                ID_KeyState = reader["ID_KeyState"].ToString(),
                Code = reader["Code"].ToString(),
                DisplayName = reader["DisplayName"].ToString(),
                State = reader["State"].ToString(),
                Room = reader["Room"].ToString(),
                Floor = reader["Floor"].ToString(),
                Building = reader["Building"].ToString()
            };

            return key;
        }
    }
    exception region...
}
```

Obr. 5.35 - Ukázka metody webové služby pro zobrazení detailu klíče

- **Zobrazení seznamu objektů**

Při volání metody pro zobrazení seznamu objektů je přes vstupní parametry předán login uživatele a dále hodnoty všech podporovaných filtrovacích parametrů. Návrátovou hodnotou metody je kolekce všech zobrazovaných objektů.

```
//všechny klíče, možnost filtrování
public Key[] KeyAll(string idLogin, int? idRoom, int? idFloor, int? idBuilding, string idKeyState, string codeFilter, string displayNameFilter)
{
    List<Key> keyList = new List<Key>();
    try
    {
        using (var db = new Db("KA_Key_All"))
        {
            db.Parameters.AddWithValue("ID_Login", idLogin);
            db.Parameters.AddWithValue("ID_Room", idRoom);
            db.Parameters.AddWithValue("ID_Floor", idFloor);
            db.Parameters.AddWithValue("ID_Building", idBuilding);
            db.Parameters.AddWithValue("ID_KeyState", idKeyState);
            db.Parameters.AddWithValue("CodeFilter", codeFilter);
            db.Parameters.AddWithValue("DisplayNameFilter", displayNameFilter);

            SqlDataReader reader = db.ExecuteReader();

            while (reader.Read())
            {
                Key key = new Key()
                {
                    ID = (Int32)reader["ID"],
                    ID_Room = (Int32)reader["ID_Room"],
                    ID_KeyState = reader["ID_KeyState"].ToString(),
                    Code = reader["Code"].ToString(),
                    DisplayName = reader["DisplayName"].ToString(),
                    Room = reader["Room"].ToString(),
                    Floor = reader["Floor"].ToString(),
                    Building = reader["Building"].ToString(),
                    State = reader["State"].ToString()
                };
                keyList.Add(key);
            }
            return keyList.ToArray();
        }
    }
    catch (Exception exp)
    {
        throw new FaultException(exp.Message);
    }
}
```

Obr. 5.36 - Ukázka metody webové služby pro zobrazení seznamu vybraných klíčů

### 5.2.2 Použité datové třídy

Následuje seznam vytvořených datových tříd, určených pro přenos informací o evidovaných objektech mezi vrstvou klienta a vrstvou webové služby. Každá třída v sobě zapouzdřuje všechny potřebné údaje, popisující detailně objekty patřící do této třídy.

#### **Borrow.cs**

Třída určená pro přenos informací o zápůjčkách.

```
[DataContract]
public class Borrow
{
    public Borrow()
    { }

    [DataMember]
    public int ID { get; set; }

    [DataMember]
    public int ID_Person { get; set; }

    [DataMember]
    public int ID_Key { get; set; }

    [DataMember]
    public string ID_KeyState { get; set; }

    [DataMember]
    public string Building { get; set; }

    [DataMember]
    public string Floor { get; set; }

    [DataMember]
    public string Room { get; set; }

    [DataMember]
    public string Key { get; set; }

    [DataMember]
    public string Person { get; set; }

    [DataMember]
    public string State { get; set; }

    [DataMember]
    public DateTime DateFrom { get; set; }

    [DataMember]
    public DateTime? DateTo { get; set; }
}
```

Obr. 5.37 – Deklarace třídy Borrow.cs

## Building.cs

Třída určená pro přenos informací o budovách.

```
[DataContract]
public class Building
{
    [DataMember]
    public int ID { get; set; }

    [DataMember]
    public string Code { get; set; }

    [DataMember]
    public string DisplayName { get; set; }

    [DataMember]
    public int FloorsCount { get; set; }

    [DataMember]
    public int RoomsCount { get; set; }

    [DataMember]
    public int KeysCount { get; set; }
}
```

Obr. 5.38 – Deklarace třídy Building.cs

## Floor.cs

Třída určená pro přenos informací o patrech.

```
[DataContract]
public class Floor
{
    [DataMember]
    public int ID { get; set; }

    [DataMember]
    public int ID_Building { get; set; }

    [DataMember]
    public string Code { get; set; }

    [DataMember]
    public string DisplayName { get; set; }

    [DataMember]
    public string Building { get; set; }

    [DataMember]
    public int RoomsCount { get; set; }

    [DataMember]
    public int KeysCount { get; set; }
}
```

Obr. 5.39 – Deklarace třídy Floor.cs

## Key.cs

Třída určená pro přenos informací o klíčích.

```
[DataContract]
public class Key
{
    [DataMember]
    public int ID { get; set; }

    [DataMember]
    public int ID_Room { get; set; }

    [DataMember]
    public string ID_KeyState { get; set; }

    [DataMember]
    public string Code { get; set; }

    [DataMember]
    public string DisplayName { get; set; }

    [DataMember]
    public string State { get; set; }

    [DataMember]
    public string Room { get; set; }

    [DataMember]
    public string Floor { get; set; }

    [DataMember]
    public string Building { get; set; }
}
```

Obr. 5.40 – Deklarace třídy Key.cs

### Person.cs

Třída určená pro přenos informací o osobách.

```
[DataContract]
public class Person
{
    public Person()
    { }

    [DataMember]
    public int ID { get; set; }

    [DataMember]
    public string DisplayName { get; set; }

    [DataMember]
    public string Email { get; set; }

    [DataMember]
    public string Roles { get; set; }
}
```

Obr. 5.41 – Deklarace třídy Person.cs

### PSNException.cs

Třída určená pro přenos informací o výjimkách.

```
[DataContract]
public class PSNException
{
    [DataMember]
    public int ID { get; set; }

    [DataMember]
    public int ID_Room { get; set; }

    [DataMember]
    public int ID_Person { get; set; }

    [DataMember]
    public bool Enabled { get; set; }

    [DataMember]
    public bool Disabled { get; set; }

    [DataMember]
    public DateTime DateFrom { get; set; }

    [DataMember]
    public DateTime? DateTo { get; set; }

    [DataMember]
    public string Room { get; set; }

    [DataMember]
    public string Person { get; set; }
}
```

Obr. 5.42 - Deklarace třídy PSNException.cs

**PSNRole.cs**

Třída určená pro přenos informací o rolích osob.

```
[DataContract]
public class PSNRole
{
    public PSNRole()
    {
        Buildings = new List<HierarchyBuilding>();
    }

    [DataMember]
    public int ID { get; set; }

    [DataMember]
    public string DisplayName { get; set; }

    [DataMember]
    public List<HierarchyBuilding> Buildings { get; set; }
}
```

Obr. 5.43 – Deklarace třídy PSNRole.cs

**Role.cs**

Třída určená pro přenos informací o rolích uživatelů aplikace.

```
[DataContract]
public class Role
{
    [DataMember]
    public int ID { get; set; }

    [DataMember]
    public string DisplayName { get; set; }

    [DataMember]
    public string RoleName { get; set; }
}
```

Obr. 5.44 – Deklarace třídy Role.cs

## Room.cs

Třída určená pro přenos informací o místnostech.

```
[DataContract]
public class Room
{
    [DataMember]
    public int ID { get; set; }

    [DataMember]
    public int ID_Floor { get; set; }

    [DataMember]
    public int? ID_User { get; set; }

    [DataMember]
    public string Code { get; set; }

    [DataMember]
    public string DisplayName { get; set; }

    [DataMember]
    public string FloorCode { get; set; }

    [DataMember]
    public string Owner { get; set; }

    [DataMember]
    public string Floor { get; set; }

    [DataMember]
    public string Building { get; set; }

    [DataMember]
    public int KeysCount { get; set; }
}
```

Obr. 5.45 – Deklarace třídy Room.cs

## User.cs

Třída určená pro přenos informací o uživateli aplikace.

```
[DataContract]
public class User
{
    [DataMember]
    public int ID { get; set; }

    [DataMember]
    public string DisplayName { get; set; }

    [DataMember]
    public string UserName { get; set; }

    [DataMember]
    public string Password { get; set; }

    [DataMember]
    public string Email { get; set; }

    [DataMember]
    public bool IsEnabled { get; set; }

    [DataMember]
    public string Roles { get; set; }
}
```

Obr. 5.46 – Deklarace třídy User.cs

### 5.3 Klientská vrstva

Nejvyšší vrstva aplikace, klientská vrstva, je v tomto řešení představována webovou aplikací. Pro její vývoj byla zvolena technologie ASP.NET. Klient přímo spolupracuje s vrstvou webové služby. Postupně volá její metody, v návaznosti na požadovaných operacích. Při jakémkoliv požadavku na zobrazení, nebo nějakou jinou manipulaci s daty, je vytvořena instance proxy třídy webové služby, pomocí které lze z klienta volat patřičné metody. Klient má díky přidání referenci na assembly služby k dispozici nejen proxy třídu s množinou definovaných metod, ale zároveň i všechny datové třídy, které mají sloužit pro obousměrný přenos dat mezi klientem a službou.

Pro autentizaci a autorizaci uživatele v aplikaci je použita patřičná infrastruktura ASP.NET, která využívá k tomuto účelu navržený model, umístěný v datové vrstvě aplikace. Jedná se o tabulky *KA\_User* pro evidenci uživatelů, *KA\_Role* pro evidenci rolí uživatelů a *KA\_Login* pro práci s jednotlivými loginy uživatelů. Informace o uživateli jsou v bezstavovém prostředí webu udržovány mezi jednotlivými přístupy klienta na server ve formě cookies u klienta webové aplikace, nebo v tzv. stavu relace – slovníkové kolekci na straně serveru.

Webová aplikace obsahuje několik desítek aspx stránek, které lze považovat za uživatelské rozhraní aplikace. Obsahují rozličné ovládací prvky s různou funkcionalitou. Důležitou skupinu tvoří ovládací prvky, které lze vázat na datové zdroje. Tyto prvky jsou schopné zobrazovat uživateli důležitá data a také mu umožňují s nimi různě manipulovat. Pro datový zdroj těchto ovládacích prvků byl zvolen sofistikovaný ovládací prvek *ObjectDataSource*. Na rozdíl od běžněji používaného prvku *SqlDataSource*, který je schopen přistupovat přímo k databázovým strukturám, pracuje tato komponenta trochu odlišněji, nezávisle na podkladové datové vrstvě. Pro svou funkci vyžaduje svázání s instancí libovolné třídy. V této třídě jsou implementovány metody, které vykonávají funkcionalitu potřebnou pro zajištění základních operací s daty. Jedná se tedy o metody pro získání seznamu objektů určitého typu, detailu jediného objektu, přidání, smazání a aktualizaci objektu. Pro každou evidovanou entitu v aplikaci byla vytvořena samostatná třída tohoto druhu, tzv. databázová třída. V prvku *ObjectDataSource* se konkrétní akce s daty (*select*, *insert*, *delete*, *update*) sváže s vybranou metodou databázové třídy. Jestliže má metoda vrátit hodnotu ve formě instance nějaké třídy, nebo kolekci těchto instancí, musí být tento typ v prvku *ObjectDataSource* také uveden. Přístup k datům přes

*ObjectDataSource* je nutný z toho důvodu, že aplikace nepřistupuje k datům přímo. Musí komunikovat s webovou službou, a teprve ta kontaktuje databázi a získává z ní data. Jednotlivé metody databázových tříd tedy ve svém těle inicializují proxy třídu webové služby a na ní potom volají odpovídající metody pro získání požadovaných dat. Takto pořízená data jsou přes volání databázové metody k dispozici ovládacímu prvku *ObjectDataSource*, který je díky vazbě na prvky uživatelského rozhraní poskytuje dále přímo uživateli aplikace.

### 5.3.1 Základní operace s daty

Mezi základní operace s daty v aplikaci patří zobrazení seznamu vybraných objektů určitého typu, zobrazení detailu jediného objektu, založení nového objektu, aktualizace objektu a konečně jeho odstranění z evidence. Následuje popis realizace jednotlivých operací na straně klientské části aplikace pro objekty místností.

- **Zobrazení seznamu objektů**

Pro zobrazení seznamu objektů je použita komponenta *GridView*. Aby zobrazovala požadovaná data, musí být navázána na datový zdroj. Jedná se o prvek *ObjectDataSource*.

```
<asp:GridView ID="grdRoomAll" runat="server" DataSourceID="odsRoomAll">
  <Columns>
    <asp:BoundField DataField="ID" Visible="false" />
    <asp:HyperLinkField DataTextField="DisplayName" HeaderText="Název" DataNavigateUrlFields="ID"
      DataNavigateUrlFormatString="~/Rooms/RoomDetail.aspx?ID_Detail={0}" SortExpression="DisplayName" />
    <asp:BoundField DataField="Code" HeaderText="Kód" SortExpression="Code" />
    <asp:BoundField DataField="Floor" HeaderText="Patro" SortExpression="Floor" ItemStyle-HorizontalAlign="Right" />
    <asp:BoundField DataField="Building" HeaderText="Budova" SortExpression="Building" />
  </Columns>
</asp:GridView>
```

Obr. 5.47 – Definice ovládacího prvku *GridView* pro zobrazení místností s napojeným prvkem *ObjectDataSource*

Prvek *ObjectDataSource* ve své definici uvádí napojení na databázovou třídu a výběr metody pro uskutečnění operace zobrazení seznamu objektů.

```
<asp:ObjectDataSource ID="odsRoomAll" runat="server" TypeName="KeysManagement.DataClasses.DBRoom"
  DataObjectTypeName="DataTable" SelectMethod="RoomAll">
</asp:ObjectDataSource>
```

Obr. 5.48 – Definice prvku *ObjectDataSource* pro zobrazení místností

V kódu aspx stránky jsou prvku datového zdroje dodány hodnoty filtrovacích parametrů. Tyto hodnoty jsou získány z příslušných uživatelských prvků stránky.

```
protected void AddParametersToDataSource()
{
    odsRoomAll.SelectParameters.Add(new ControlParameter("idBuilding", "ddlBuildingAll", "SelectedValue"));
    odsRoomAll.SelectParameters.Add(new ControlParameter("idFloor", "ddlFloorAll", "SelectedValue"));
    odsRoomAll.SelectParameters.Add(new ControlParameter("displayNameFilter", "txtDisplayNameFilter", "Text"));
    odsRoomAll.SelectParameters.Add(new ControlParameter("codeFilter", "txtCodeFilter", "Text"));
}
```

Obr. 5.49 – Dodání hodnot parametrů datovému zdroji pro zobrazení místností

Po vyvolání příslušné akce, v tomto případě akce *select*, zavolá datový zdroj metodu instance, kterou má uvedenou ve své definici pro tuto akci.

```
public DataTable RoomAll(int? idBuilding, int? idFloor, string CodeFilter, string DisplayNameFilter)
{
    try
    {
        using (var service = new KMServiceClient())
        {
            List<Room> roomList = service.RoomAll(HttpContext.Current.Session["ID_Login"].ToString(), idBuilding,
            idFloor, CodeFilter, DisplayNameFilter).ToList();
            return TransferRoomListToTable(roomList);
        }
    }
    catch (Exception exp)
    {
        if (exp.Message == "Uživatel byl odhlášen!")
        {
            FormsAuthentication.SignOut();
            FormsAuthentication.RedirectToLoginPage();
        }
        else
            throw new Exception(exp.Message);
    }

    return null;
}
```

Obr. 5.50 – Definice metody RoomAll ze třídy DBRoom pro zobrazení místností

Obdržená data seznamu objektů následně datový zdroj posílá navázanému prvku *GridView*, který je zobrazí.

Název	Kód	Patro	Budova
Místnost 1 - P1B1	M1P1B1	Patro 1 - B1	Budova 1
Místnost 2 - P1B1	M2P1B1	Patro 1 - B1	Budova 1
Místnost 3 - P1B1	M3P1B1	Patro 1 - B1	Budova 1

Obr. 5.51 – Zobrazení místností ve webové aplikaci

- **Zobrazení detailu objektu**

Pro zobrazení detailu objektu je použita komponenta *FormView*. Je navázána na podobný datový zdroj, jako předcházející komponenta pro zobrazení seznamu objektů.

```
<asp:FormView runat="server" ID="fvRoom" DataSourceID="odsRoomDetail" CssClass="detail" OnPreRender="fvRoom_PreRender" >
  <ItemTemplate>
    <asp:Panel ID="panelBuilding" runat="server" GroupingText="Místnost">
      <table>
        <tr>
          <td>
            <asp:Literal ID="litBuilding" runat="server" Text="Budova:" />
          </td>
          <td style="padding-left:10px;">
            <asp:Label runat="server" ID="lblBuilding" Text="<%# Eval("Building") %>" />
          </td>
        </tr>
        <tr>
          <td>
            <asp:Literal ID="litFloor" runat="server" Text="Patro:" />
          </td>
          <td style="padding-left:10px;">
            <asp:Label runat="server" ID="lblFloor" Text="<%# Eval("Floor") %>" />
          </td>
        </tr>
      </table>
    </asp:Panel>
  </ItemTemplate>
</asp:FormView>
```

Obr. 5.52 – Část definice ovládacího prvku FormView pro zobrazení detailu místnosti

Prvek *ObjectDataSource* ve své definici uvádí napojení na databázovou třídu a výběr metody pro uskutečnění operace zobrazení detailu objektu. Součástí definice je i předání hodnoty ID požadovaného objektu prostřednictvím parametru dotazovacího řetězce stránky.

```
<asp:ObjectDataSource ID="odsRoomDetail" runat="server" TypeName="KeysManagement.DataClasses.DBRoom"
  DataObjectTypeName="Room" SelectMethod="RoomDetail">
  <SelectParameters>
    <asp:QueryStringParameter Name="id" QueryStringField="ID_Detail" />
  </SelectParameters>
</asp:ObjectDataSource>
```

Obr. 5.53 – Definice prvku ObjectDataSource pro zobrazení detailu místnosti

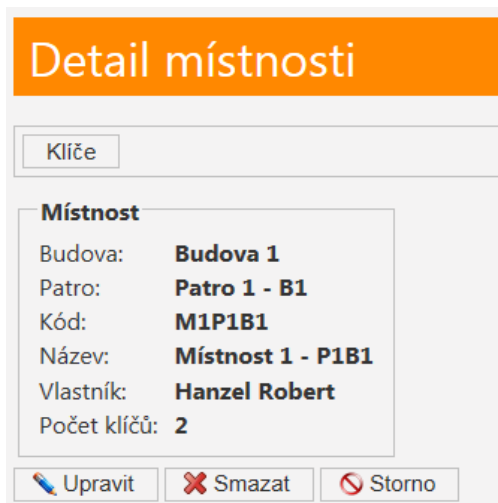
Po vyvolání příslušné akce, v tomto případě taktéž akce *select*, zavolá datový zdroj metodu instance, kterou má uvedenou ve své definici pro tuto akci.

```
public Room RoomDetail(int id)
{
    try
    {
        using (var service = new KMServiceClient())
        {
            Room room = service.RoomDetail(HttpContext.Current.Session["ID_Login"].ToString(), id);
            return room;
        }
    }
    catch (Exception exp)
    {
        if (exp.Message == "Uživatel byl odhlášen!")
        {
            FormsAuthentication.SignOut();
            FormsAuthentication.RedirectToLoginPage();
        }
        else
            throw new Exception(exp.Message);

        return null;
    }
}
```

Obr. 5.54 - Definice metody RoomDetail ze třídy DBRoom pro zobrazení detailu místnosti

Obdržená data detailu objektu následně datový zdroj posílá navázanému prvku *FormView*, který je zobrazí.




<b>Detail místnosti</b>		
<input type="button" value="Klíče"/>		
<b>Místnost</b>		
Budova:	<b>Budova 1</b>	
Patro:	<b>Patro 1 - B1</b>	
Kód:	<b>M1P1B1</b>	
Název:	<b>Místnost 1 - P1B1</b>	
Vlastník:	<b>Hanzel Robert</b>	
Počet klíčů:	<b>2</b>	
<input type="button" value="Upravit"/>	<input type="button" value="Smazat"/>	<input type="button" value="Storno"/>

Obr. 5.55 - Zobrazení detailu místnosti ve webové aplikaci

- **Založení nového objektu**

Pro zadání údajů nového objektu je použita komponenta *FormView*. Je navázána na podobný datový zdroj, jako předcházející komponenty. Automaticky mu předává zadané údaje nového objektu.



Obr. 5.56 – Formulář pro zadání údajů nové místnosti

```
<asp:FormView runat="server" ID="fvRoom" DataSourceID="odsRoomNew" CssClass="detail" DefaultMode="Insert"
  EnableViewState="true" OnPreRender="fvRoom_PreRender" OnItemInserted="fvRoom_ItemInserted" >
  <InsertItemTemplate>

    <uc:UserDialog runat="server" ID="popup" PositionX="250" PositionY="100" TargetControlID="btnSave"
      Title="Založení záznamu" Message="Opravdu si přejete založit novou místnost?" OkCancel="true"
      OkButtonCommandName="Insert" />

    <asp:Panel ID="panelRoom" runat="server">
      <table cellpadding="2">
        <tr>
          <td style="border:none; text-align:right;" >
            <asp:Label ID="lblFloor" runat="server" Text="Patro:" />
          </td>
          <td style="border:none;">
            <asp:Literal ID="litFloor" runat="server" />
          </td>
        </tr>
        <tr>
          <td style="border:none; text-align:right;" >
            <asp:Label ID="lblCode" runat="server" Text="Kód:" />
          </td>
          <td>
            <asp:TextBox runat="server" ID="txtCode" Text="<%# Bind("Code") %>" />
            <asp:RequiredFieldValidator ID="reqCode" runat="server" ControlToValidate="txtCode">

```

Obr. 5.57 – Část definice ovládacího prvku FormView pro založení nové místnosti

Prvek *ObjectDataSource* ve své definici uvádí napojení na databázovou třídu a výběr metody pro uskutečnění operace založení nového objektu.

```
<asp:ObjectDataSource ID="odsRoomNew" runat="server" TypeName="KeysManagement.DataClasses.DBRoom"
    DataObjectTypeName="KeysManagement.KMServiceReference.Room"
    InsertMethod="AddRoom" OnInserting="odsRoomNew_Inserting" >
</asp:ObjectDataSource>
```

Obr. 5.58 - Definice prvku ObjectDataSource pro založení nové místnosti

Po vyvolání příslušné akce, v tomto případě akce *insert*, zavolá datový zdroj metodu instance, kterou má uvedenou ve své definici pro tuto akci. Výsledkem volání je založení nového objektu.

```
public void AddRoom(Room room)
{
    try
    {
        using (var service = new KMServiceClient())
        {
            service.AddRoom(HttpContext.Current.Session["ID_Login"].ToString(), room);
        }
    }
    catch (Exception exp)
    {
        if (exp.Message == "Uživatel byl odhlášen!")
        {
            FormsAuthentication.SignOut();
            FormsAuthentication.RedirectToLoginPage();
        }
        else
            throw new Exception(exp.Message);
    }
}
```

Obr. 5.59 - Definice metody AddRoom ze třídy DBRoom pro založení nové místnosti

- **Aktualizace objektu**

Pro aktualizaci údajů objektu je také použita komponenta *FormView*. Je navázána na podobný datový zdroj, jako předcházející komponenty. Automaticky mu předává zadané údaje aktualizovaného objektu.

Obr. 5.60 - Formulář pro zadání údajů aktualizované místnosti

```

<asp:FormView runat="server" ID="fvRoom" DataSourceID="odsRoomEdit" CssClass="detail" DefaultMode="Edit"
  EnableViewState="true" OnItemUpdated="fvRoom_ItemUpdated" OnPreRender="fvRoom_PreRender" >
  <EditItemTemplate>

  <uc:UserDialog runat="server" ID="popup" PositionX="250" PositionY="100" TargetControlID="btnUpdate"
    Title="Aktualizace záznamu" Message="Opravdu si přejete aktualizovat údaje místnosti?" OkCancel="true"
    OkButtonCommandName="Update" />

  <asp:Panel ID="panelRoom" runat="server">
    <table cellpadding="2">
      <tr>
        <td>
          <asp:TextBox runat="server" ID="txtId" Text='<%= Bind("ID") %>' Visible="false" />
        </td>
      </tr>
      <tr>
        <td style="text-align:right;">
          <asp:Label ID="lblCode" runat="server" Text="Kód:" />
        </td>
        <td>
          <asp:TextBox runat="server" ID="txtCode" Text='<%= Bind("Code") %>' />
          <asp:RequiredFieldValidator ID="reqCode" runat="server" ControlToValidate="txtCode"
            Display="Dynamic" Text="Povinné pole!">
          </asp:RequiredFieldValidator>
        </td>
      </tr>
    </table>
  </asp:Panel>
  </EditItemTemplate>
</asp:FormView>

```

Obr. 5.61 - Část definice ovládacího prvku FormView pro aktualizaci místnosti

Prvek *ObjectDataSource* ve své definici uvádí napojení na databázovou třídu a výběr metod pro uskutečnění operací zobrazení a aktualizace objektu. Parametry pro dodání ID objektu získá datový zdroj prostřednictvím dotazovacího řetězce stránky.

```
<asp:ObjectDataSource ID="odsRoomEdit" runat="server" TypeName="KeysManagement.DataClasses.DBRoom"
  DataObjectTypeName="KeysManagement.KMServiceReference.Room" SelectMethod="RoomDetail" UpdateMethod="EditRoom"
  OnSelected="odsRoomEdit_Selected" OnUpdating="odsRoomEdit_Updating" >
  <SelectParameters>
    <asp:QueryStringParameter QueryStringField="ID_Detail" Name="ID" DbType="Int32" />
  </SelectParameters>
  <UpdateParameters>
    <asp:QueryStringParameter QueryStringField="ID_Detail" Name="ID" DbType="Int32" />
  </UpdateParameters>
</asp:ObjectDataSource>
```

Obr. 5.62 - Definice prvku *ObjectDataSource* pro aktualizaci místnosti

Po vyvolání příslušné akce, v tomto případě akce *update*, zavolá datový zdroj metodu instance, kterou má uvedenou ve své definici pro tuto akci. Výsledkem volání je aktualizace údajů objektu.

```
public void EditRoom(Room room)
{
    try
    {
        using (var service = new KMServiceClient())
        {
            service.EditRoom(HttpContext.Current.Session["ID_Login"].ToString(), room);
        }
    }
    catch (Exception exp)
    {
        if (exp.Message == "Uživatel byl odhlášen!")
        {
            FormsAuthentication.SignOut();
            FormsAuthentication.RedirectToLoginPage();
        }
        else
            throw new Exception(exp.Message);
    }
}
```

Obr. 5.63 - Definice metody *EditRoom* ze třídy *DBRoom* pro aktualizaci místnosti

- **Odstranění objektu**

Pro operaci odstranění objektu z evidence není potřeba využívat žádné ovládací prvky vázané na data, tím pádem ani žádný datový zdroj. Jednoduše se na patřičném místě ručně zavolá metoda odpovídající databázové třídy, která má odstranění objektu na starosti.

Parametrem se jí jen musí dodat ID mazaného objektu. Ve většině případů se tato hodnota získá z dotazovacího řetězce detailové stránky objektu.

```
public void DeleteRoom(int id)
{
    try
    {
        using (var service = new KMServiceClient())
        {
            service.DeleteRoom(HttpContext.Current.Session["ID_Login"].ToString(), id);
        }
    }
    catch (Exception exp)
    {
        if (exp.Message == "Uživatel byl odhlášen!")
        {
            FormsAuthentication.SignOut();
            FormsAuthentication.RedirectToLoginPage();
        }
        else
            throw new Exception(exp.Message);
    }
}
```

Obr. 5.64 - Definice metody DeleteRoom ze třídy DBRoom pro odstranění místnosti

### 5.3.2 Databázové třídy

Následuje seznam definic vytvořených databázových tříd. Pro každou evidovanou entitu je k dispozici jedna třída, která obsahuje implementaci všech metod potřebných k zajištění požadovaných operací s objekty daného typu.

#### DBBorrow.cs

Databázová třída pro zajištění operací se zápůjčkami.

```
public class DBBorrow
{
    public DBBorrow()
    { }

    private DataTable TransferBorrowListToTable(Borrow[] borrowArray)...

    public void AddBorrow(Borrow borrow)...

    public void EditBorrow(Borrow borrow)...

    public void DeletePerson(int id)...

    public Borrow BorrowDetail(int id)...

    public DataTable BorrowAll(int idKey)...

    public DataTable BorrowAll(int? idPerson, int? idRoom, int? idKey, string idKeyState,
        string dateFromFilter, string dateToFilter)...

    public DataTable BorrowAll(int? idPerson, int? idBuilding, int? idFloor, int? idRoom, int? idKey,
        string idKeyState, string dateFromFilter, string dateToFilter)...
}
```

Obr. 5.65 – Definice třídy pro práci se zápůjčkami

**DBBuilding.cs**

Databázová třída pro zajištění operací s budovami.

```
public class DBBuilding
{
    public DBBuilding()
    { }

    private DataTable TransferBuildingListToTable(List<Building> buildingList)...

    public void AddBuilding(Building building)...

    public void EditBuilding(Building building)...

    public void DeleteBuilding(int id)...

    public Building BuildingDetail(int id)...

    public DataTable BuildingAll(string codeFilter, string displayNameFilter)...

    public DataTable BuildingAll()...
}
```

Obr. 5.66 - Definice třídy pro práci s budovami

**DBFloor.cs**

Databázová třída pro zajištění operací s patry.

```
public class DBFloor
{
    public DBFloor()
    { }

    private DataTable TransferFloorListToTable(List<Floor> floorList)...

    public void AddFloor(Floor floor)...

    public void EditFloor(Floor floor)...

    public void DeleteFloor(int id)...

    public Floor FloorDetail(int id)...

    public DataTable FloorAll(int? idBuilding)...

    public DataTable FloorAll(int? idBuilding, string codeFilter, string displayNameFilter)...
}
```

Obr. 5.67 - Definice třídy pro práci s patry

**DBKey.cs**

Databázová třída pro zajištění operací s klíči.

```
public class DBKey
{
    public DBKey()
    { }

    private DataTable TransferKeyListToTable(List<Key> keyList)...

    public void AddKey(Key key)...

    public void EditKey(Key key)...

    public void DeleteKey(int id)...

    public Key KeyDetail(int id)...

    public DataTable KeyAll(int idRoom)...

    public DataTable KeyAll(int? idBuilding, int? idFloor, int? idRoom)...

    public DataTable KeyAll(int? idRoom, int? idBuilding, int? idFloor, string idKeyState,
        string CodeFilter, string DisplayNameFilter)...
}
```

Obr. 5.68 – Definice třídy pro práci s klíči

**DBKeyState.cs**

Databázová třída pro zajištění operací se stavy klíčů a zápůjček.

```
public class DBKeyState
{
    public DBKeyState()
    { }

    public KeyState[] KeyStateAll()...
}
```

Obr. 5.69 – Definice třídy pro práci se stavy klíčů a zápůjček

**DBOwnerRoom.cs**

Databázová třída pro zajištění operací s místnostmi vlastníků.

```
public class DBOwnerRoom
{
    public DBOwnerRoom()
    { }

    private DataTable TransferOwnerRoomListToTable(List<Room> roomList)...

    public DataTable OwnerRoomAll(string displayNameFilter)...
}
```

Obr. 5.70 – Definice třídy pro práci s místnostmi vlastníků

**DBPerson.cs**

Databázová třída pro zajištění operací s osobami.

```
public class DBPerson
{
    public DBPerson()
    { }

    private DataTable TransferPersonListToTable(Person[] personArray)...

    private DataTable TransferRoomListToTable(Room[] roomArray)...

    public void AddPerson(Person person)...

    public void EditPerson(Person person)...

    public void DeletePerson(int id)...

    public Person PersonDetail(int id)...

    public DataTable PersonAll()...

    public DataTable PersonAll(int? idRole, string displayNameFilter)...

    public Person[] PersonAllNames()...

    public DataTable PersonAllRooms(int idPerson)...
}
```

Obr. 5.71 – Definice třídy pro práci s osobami

### DBPSNException.cs

Třída pro zajištění operací s výjimkami.

```
public class DBPSNException
{
    public DBPSNException()
    { }

    private DataTable TransferPSNExceptionListToTable(PSNException[] psnExceptionArray)...

    public void AddPSNException(PSNException exception)...

    public void EditPSNException(PSNException exception)...

    public void DeletePSNException(int id)...

    public PSNException PSNExceptionDetail(int id)...

    public DataTable PSNExceptionAll(int idRoom, string personFilter, bool enabled, bool disabled,
        string dateFromFilter, string dateToFilter)...
}
```

Obr. 5.72 – Definice třídy pro práci s výjimkami

### DBPSNRole.cs

Třída pro zajištění operací s rolemi osob.

```
public class DBPSNRole
{
    public DBPSNRole()
    { }

    private DataTable TransferPSNRoleListToTable(PSNRole[] roleArray)...

    public void AddPSNRole(PSNRole role)...

    public void EditPSNRole(PSNRole role)...

    public void DeletePSNRole(int id)...

    public PSNRole PSNRoleDetail(int id)...

    public DataTable PSNRoleAll()...

    public DataTable PSNRoleAll(string displayNameFilter)...
}
```

Obr. 5.73 – Definice třídy pro práci s rolemi osob

**DBRole.cs**

Třída pro zajištění operací s uživatelskými rolemi.

```
public class DBRole
{
    public DBRole()
    { }

    private DataTable TransferRoleListToTable(List<Role> roleList)...

    public DataTable GetRoleAll()...

    public DataTable GetRoleAll(string userName)...
```

Obr. 5.74 – Definice třídy pro práci s uživatelskými rolemi

**DBRoom.cs**

Třída pro zajištění operací s místnostmi

```
public class DBRoom
{
    public DBRoom()
    { }

    private DataTable TransferRoomListToTable(List<Room> roomList)...

    public void AddRoom(Room room)...

    public void EditRoom(Room room)...

    public void DeleteRoom(int id)...

    public Room RoomDetail(int id)...

    public DataTable RoomAll()...

    public DataTable RoomAll(int? idFloor)...

    public DataTable RoomAll(int? idBuilding, int? idFloor)...

    public DataTable RoomAll(int? idBuilding, int? idFloor, string CodeFilter, string DisplayNameFilter)...
```

Obr. 5.75 – Definice třídy pro práci s místnostmi

**DBUser.cs**

Třída pro zajištění operací s uživateli.

```
public class DBUser
{
    public DBUser()
    { }

    private DataTable TransferUserListToTable(List<User> userList) [...]

    public void AddUser(User user) [...]

    public void EditUser(User user) [...]

    public void DeleteUser(int id) [...]

    public User UserDetails(int id) [...]

    public DataTable UserAll() [...]

    public DataTable UserAll(string displayName, int? idRole) [...]
}
```

Obr. 5.76 – Definice třídy pro práci s uživateli

## ZÁVĚR

Cílem této práce bylo vytvoření aplikace pro evidenci klíčů a zápůjček těchto klíčů. Aplikace bude úspěšně sloužit v prostředí, kde dochází k častému pohybu klíčů mezi různými osobami a vyvstává zde potřeba tuto činnost řádně evidovat. Lze ji provozovat v rámci počítačové sítě, přístup tedy není omezen pouze z jednoho stroje. Funkcionalita řešení umožňuje spravovat základní entity spojené s problematikou. Jedná se o evidenci budov, pater, místností, klíčů, zápůjček a osob. Dále je také umožněno spravovat uživatele aplikace. Užitečnou vlastností aplikace je existence vlastníka místnosti s přímou kontrolou přístupu a také možnost seskupovat evidované osoby do vlastních rolí. Aplikaci lze obsluhovat v kontextu několika uživatelských rolí. Na data lze pohlížet z různých pohledů a získávat tak ze systému velmi užitečné informace.

Důležitá byla zvolená architektura aplikace. Jedná se o vícevrstvé řešení s nasazením webové služby v rámci samostatné vrstvy. Klientem služby pak může být libovolná aplikace, běžící na libovolné platformě. Jako součást diplomové práce byla zvolena realizace klienta služby ve formě webové aplikace. Do budoucna je však možné doprogramovat další typy klientů a použití aplikace tak rozšířit například o mobilní platformu, nebo vytvořit desktopovou verzi klienta, která poběží v prostředí MS Windows, nebo Linux. Datová vrstva je realizovaná databází v rámci databázového serveru SQL Server.

Bezpečnost provozu je důležitou stránkou aplikací vhodných pro nasazení do reálného prostředí, hlavně v případě síťových řešení. Při komunikaci mezi klientem a webovou službou by měl být použit zabezpečený komunikační kanál realizovaný prostřednictvím bezpečnostních certifikátů. Hesla uživatelů jsou ukládána do databáze v zašifrované podobě. Obranou proti útoku typu *SQL Injection* je zabezpečený přístup k datům v databázi výhradně prostřednictvím uložených procedur.

Aplikace Správa oprávnění vydávání klíčů v budovách bude dobrým pomocníkem při řešení problému s evidencí a správou zapůjčování klíčů a to i v rozsáhlejších měřítku. Poskytne svým uživatelům komplexní pohled na udržovanou agendu a umožní tak efektivně spravovat danou problematiku. Nasazením do prostředí Internetu, které je umožněno použitím webových technologií při tvorbě aplikace, bude navíc možné přistupovat k aplikaci téměř odkudkoliv.

## ZÁVĚR V ANGLIČTINĚ

The aim of this work was to create an application for registration of a lending keys the keys. The application will successfully serve in an environment with frequent movement keys between different people and it raises the need for this activity properly recorded. It can be operated under computer network access is not limited to only one machine. Functionality solutions to manage basic entity associated with the topic. These are records of buildings, floors , rooms , keys, loans and people. It is also possible to manage the user's application. A useful feature of the application is the owner of existence room with direct access control and the ability to keep track of registered persons in their own roles. The application can operate on multiple user roles. The data can be viewed from different perspectives and get the system very useful information .

Also important was the chosen application architecture. It is a multi-layered solution with the deployment of web services in a separate layer. Client service can be any application running on any platform. As part of the thesis was chosen implementation of client services in the form of a web application. In the future , it is possible to create other types of clients and how to use and extend, for example, a mobile platform, or create a desktop client version that will run under MS Windows or Linux. The data layer is implemented databases in the SQL Server database server .

Operational safety is an important aspect of applications suitable for deployment in a real environment, especially in the case of network solutions. When communication between the client and the web service should be used to secure communications channel implemented through the security certificates. Users' passwords are stored in the database in encrypted form. Defense against attacks *sql-injection* is secure access to data in the database only through stored procedures.

Application Administration authority issuing keys in buildings will be a big help in solving the problem of accounting and asset lending keys and even larger scale. It provides its users with a comprehensive view of sustained agenda, allowing you to effectively manage the issue. Deploying to the Internet, which is the use of web technologies to create an application, will also be able to access the application from almost anywhere.

## SEZNAM POUŽITÉ LITERATURY

- [1] MACDONALD, Matthew, Adam FREEMAN a Mario SZPUSZTA. ASP.NET 4 a C# 2010: tvorba dynamických stránek profesionálně. Vyd. 1. Překlad Jan Pokorný. Brno: Zoner Press, 2011, 880 s. Encyklopedie Zoner Press. ISBN 978-80-7413-131-8.
- [2] LACKO, Ľuboslav. 1001 tipů a triků pro SQL. Vyd. 1. Brno: Computer Press, 2011, 416 s. ISBN 978-80-251-3010-0.
- [3] NASH, Trey. C# 2010: rychlý průvodce novinkami a nejlepšími postupy. Vyd. 1. Brno: Computer Press, 2010, 624 s. ISBN 978-80-251-3034-6
- [4] JQuery: kuchařka programátora. Vyd. 1. Brno: Computer Press, 2010, 436 s. ISBN 978-80-251-3152-7.
- [5] CASTRO, Elizabeth. HTML, XHTML a CSS: názorný průvodce tvorbou WWW stránek. Vyd. 1. Brno: Computer Press, 2007, 438 s. ISBN 978-80-251-1531-2.
- [6] PECINOVSKEÝ, Rudolf. Návrhové vzory. Vyd. 1. Brno: Computer Press, 2007, 527 s. ISBN 978-80-251-1582-4.
- [7] 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.
- [8] EVVA [online]. [cit. 2014-05-02]. Dostupné z: <http://www.evva.cz/>
- [9] Zápůjčky. *TESCO SW* [online]. [cit. 2014-05-02]. Dostupné z: [http://www.tescosw.cz/cz/moduly-fama/zapujcky/art\\_2834/zapujcky.aspx](http://www.tescosw.cz/cz/moduly-fama/zapujcky/art_2834/zapujcky.aspx)
- [10] Správa klíčového hospodářství. *Yamaco* [online]. [cit. 2014-05-02]. Dostupné z: <http://www.yamaco.cz/produkty/sprava-klicoveho-hospodarstvi.html>

## SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

OS    Operační systém

IS    Informační systém

VS    Visual Studio

**SEZNAM OBRÁZKŮ**

Obr. 2.1 - Evidované entity aplikace.....	14
Obr. 2.2 – Uživatelské role .....	14
Obr. 2.3 – Proces zapůjčení klíče.....	16
Obr. 2.4 - Přístup k chráněnému webovému zdroji vyžadujícímu autentizaci .....	18
Obr. 2.5 - Přístup k chráněnému webovému zdroji vyžadujícímu autorizaci.....	19
Obr. 2.6 – Autentizační a autorizační moduly ASP.NET .....	21
Obr. 2.7 - Aktivace událostí zabezpečení ASP.NET .....	22
Obr. 2.8 - Proces autentizace Windows .....	25
Obr. 2.9 - Proces formulářové autentizace.....	26
Obr. 2.10 - Struktura členství ASP.NET .....	28
Obr. 3.1 – Funkční požadavky na aplikaci, část 1 .....	33
Obr. 3.2 – Funkční požadavky na aplikaci, část 2 .....	34
Obr. 3.3 – Nefunkční požadavky na aplikaci.....	34
Obr. 3.4 – Analytický diagram tříd .....	35
Obr. 3.5 – Aktéři .....	36
Obr. 3.6 – Use case model Evidence budov .....	37
Obr. 3.7 – Use case model Evidence pater .....	38
Obr. 3.8 – Use case model Evidence místností.....	39
Obr. 3.9 – Use case model Evidence klíčů .....	40
Obr. 3.10 – Use case model Evidence zápůjček .....	41
Obr. 3.11 – Use case model Evidence osob.....	42
Obr. 3.12 – Use case model Evidence rolí osob .....	43
Obr. 3.13 – Use case model Evidence výjimek .....	44
Obr. 3.14 – Use case model Evidence uživatelů.....	45
Obr. 3.15 – Diagram aktivit Zobrazení seznamu budov .....	46
Obr. 3.16 – Diagram aktivit Zobrazení detailu budovy .....	46
Obr. 3.17 – Diagram aktivit Editace detailu budovy .....	47
Obr. 3.18 – Diagram aktivit Založení nové budovy .....	48
Obr. 3.19 – Diagram aktivit Odstranění budovy.....	49
Obr. 3.20 – Diagram aktivit Zobrazení přiřazených pater budovy .....	50
Obr. 3.21 – Diagram aktivit Založení nového patra budovy.....	51

Obr. 3.22 – Diagram aktivit Zobrazení seznamu zápůjček.....	52
Obr. 3.23 – Diagram aktivit Zobrazení zápůjček místnosti vlastníka.....	53
Obr. 3.24 – Diagram aktivit Zobrazení detailu zápůjčky.....	54
Obr. 3.25 - Diagram aktivit Založení nové zápůjčky.....	55
Obr. 3.26 – Diagram aktivit Odstranění zápůjčky .....	56
Obr. 3.27 – Diagram aktivit Editace detailu zápůjčky.....	57
Obr. 3.28 – Diagram aktivit Zobrazení výjimek místnosti vlastníka.....	58
Obr. 3.29 – Diagram aktivit Zobrazení detailu výjimky.....	59
Obr. 3.30 – Diagram aktivit Odstranění výjimky .....	59
Obr. 3.31 – Diagram aktivit Založení nové výjimky .....	60
Obr. 3.32 – Diagram aktivit Editace detailu výjimky .....	61
Obr. 3.33 – Diagram aktivit Přihlášení uživatele.....	62
Obr. 3.34 – Diagram aktivit Odhlášení uživatele .....	62
Obr. 3.35 – Diagram aktivit Změna hesla.....	63
Obr. 3.36 – Diagram aktivit Zobrazení seznamu uživatelů .....	64
Obr. 3.37 – Diagram aktivit Zobrazení detailu uživatele.....	65
Obr. 3.38 – Diagram aktivit Založení nového uživatele.....	66
Obr. 3.39 – Diagram aktivit Odstranění uživatele .....	67
Obr. 3.40 – Diagram aktivit Editace detailu uživatele.....	68
Obr. 4.1 – Architektura systému .....	70
Obr. 4.2 – Komponenty aplikace .....	72
Obr. 5.1 – Databázové schéma .....	74
Obr. 5.2 – Tabulka KA_User.....	75
Obr. 5.3 – Tabulka KA_UserRole .....	75
Obr. 5.4 – Tabulka KA_Role.....	75
Obr. 5.5 – Tabulka KA_Login.....	76
Obr. 5.6 – Tabulka KA_Person.....	76
Obr. 5.7 – Tabulka KA_PersonPSNRole.....	76
Obr. 5.8 – Tabulka KA_PSNRole .....	77
Obr. 5.9 – Tabulka KA_PSNRoleRoom.....	77
Obr. 5.10 – Tabulka KA_Building .....	77
Obr. 5.11 – Tabulka KA_Floor.....	78
Obr. 5.12 – Tabulka KA_Room.....	78

Obr. 5.13 – Tabulka KA_Key .....	78
Obr. 5.14 – Tabulka KA_KeyState .....	79
Obr. 5.15 – Tabulka KA_Borrow .....	79
Obr. 5.16 – Tabulka KA_PSNEException .....	79
Obr. 5.17- Ukázka ověření přihlášeného uživatele na začátku procedury .....	80
Obr. 5.18 – Ukázka ověření práv uživatele pro provedení požadované operace .....	81
Obr. 5.19 – Ukázka validace vstupního parametru procedury .....	81
Obr. 5.20 – Ukázka založení nového záznamu do tabulky KA_Building .....	81
Obr. 5.21 – Ukázka prodloužení platnosti loginu po provedení požadované operace v uložené proceduře .....	82
Obr. 5.22 – Ukázka definice vstupních parametrů procedury KA_Borrow_All pro zobrazení seznamu zápůjček .....	82
Obr. 5.23 – Ukázka SQL příkazu v proceduře KA_Borrow_All pro zobrazení seznamu zápůjček na základě dodaných vstupních parametrů .....	83
Obr. 5.24 - Ukázka definice vstupních parametrů procedury KA_Borrow_Detail pro zobrazení detailu zápůjčky .....	83
Obr. 5.25 - Ukázka SQL příkazu v proceduře KA_Borrow_Detail pro zobrazení detailu zápůjčky na základě dodaných vstupních parametrů .....	84
Obr. 5.26 - Ukázka definice vstupních parametrů procedury KA_Borrow_New pro vytvoření nové zápůjčky .....	84
Obr. 5.27 - Ukázka SQL příkazu v proceduře KA_Borrow_New pro vytvoření nové zápůjčky .....	84
Obr. 5.28 - Ukázka definice vstupních parametrů procedury KA_Borrow_Edit pro aktualizaci zápůjčky .....	85
Obr. 5.29 - Ukázka SQL příkazu v proceduře KA_Borrow_Edit pro aktualizaci zápůjčky .....	85
Obr. 5.30 - Ukázka definice vstupních parametrů procedury KA_Borrow_Del pro smazání zápůjčky .....	85
Obr. 5.31 - Ukázka SQL příkazu v proceduře KA_Borrow_Del pro smazání zápůjčky .....	86
Obr. 5.32 - Ukázka metody webové služby pro založení nového klíče .....	89
Obr. 5.33 - Ukázka metody webové služby pro odstranění klíče .....	90
Obr. 5.34 - Ukázka metody webové služby pro aktualizaci klíče .....	90
Obr. 5.35 - Ukázka metody webové služby pro zobrazení detailu klíče .....	91

Obr. 5.36 - Ukázka metody webové služby pro zobrazení seznamu vybraných klíčů.....	92
Obr. 5.37 – Deklarace třídy Borrow.cs .....	93
Obr. 5.38 – Deklarace třídy Building.cs .....	93
Obr. 5.39 – Deklarace třídy Floor.cs.....	94
Obr. 5.40 – Deklarace třídy Key.cs.....	94
Obr. 5.41 – Deklarace třídy Person.cs .....	95
Obr. 5.42 - Deklarace třídy PSNException.cs .....	95
Obr. 5.43 – Deklarace třídy PSNRole.cs .....	96
Obr. 5.44 – Deklarace třídy Role.cs.....	96
Obr. 5.45 – Deklarace třídy Room.cs.....	97
Obr. 5.46 – Deklarace třídy User.cs.....	97
Obr. 5.47 – Definice ovládacího prvku GridView pro zobrazení místností s napojeným prvkem ObjectDataSource .....	99
Obr. 5.48 – Definice prvku ObjectDataSource pro zobrazení místností .....	99
Obr. 5.49 – Dodání hodnot parametrů datovému zdroji pro zobrazení místností .....	100
Obr. 5.50 – Definice metody RoomAll ze třídy DBRoom pro zobrazení místností.....	100
Obr. 5.51 – Zobrazení místností ve webové aplikaci.....	100
Obr. 5.52 – Část definice ovládacího prvku FormView pro zobrazení detailu místnosti .....	101
Obr. 5.53 – Definice prvku ObjectDataSource pro zobrazení detailu místnosti .....	101
Obr. 5.54 - Definice metody RoomDetail ze třídy DBRoom pro zobrazení detailu místnosti .....	102
Obr. 5.55 - Zobrazení detailu místnosti ve webové aplikaci .....	102
Obr. 5.56 – Formulář pro zadání údajů nové místnosti .....	103
Obr. 5.57 – Část definice ovládacího prvku FormView pro založení nové místnosti.....	103
Obr. 5.58 - Definice prvku ObjectDataSource pro založení nové místnosti .....	104
Obr. 5.59 - Definice metody AddRoom ze třídy DBRoom pro založení nové místnosti .....	104
Obr. 5.60 - Formulář pro zadání údajů aktualizované místnosti.....	105
Obr. 5.61 - Část definice ovládacího prvku FormView pro aktualizaci místnosti .....	105
Obr. 5.62 - Definice prvku ObjectDataSource pro aktualizaci místnosti .....	106
Obr. 5.63 - Definice metody EditRoom ze třídy DBRoom pro aktualizaci místnosti .....	106
Obr. 5.64 - Definice metody DeleteRoom ze třídy DBRoom pro odstranění místnosti ....	107

Obr. 5.65 – Definice třídy pro práci se zápůjčkami .....	107
Obr. 5.66 - Definice třídy pro práci s budovami .....	108
Obr. 5.67 - Definice třídy pro práci s patry .....	108
Obr. 5.68 – Definice třídy pro práci s klíči .....	109
Obr. 5.69 – Definice třídy pro práci se stavy klíčů a zápůjček .....	109
Obr. 5.70 – Definice třídy pro práci s místnostmi vlastníků.....	110
Obr. 5.71 – Definice třídy pro práci s osobami.....	110
Obr. 5.72 – Definice třídy pro práci s výjimkami .....	111
Obr. 5.73 – Definice třídy pro práci s rolemi osob .....	111
Obr. 5.74 – Definice třídy pro práci s uživatelskými rolemi .....	112
Obr. 5.75 – Definice třídy pro práci s místnostmi .....	112
Obr. 5.76 – Definice třídy pro práci s uživateli .....	113

**SEZNAM TABULEK**

Tab. 4.1 - Použité technologie ..... 71