Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Reengineering školního systému techniků a pilotů dle Part 66

Bc. Petr Erben

Diplomová práce

2013

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2012/2013**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Petr Erben**
Osobní číslo:	**I10380**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Reengineering školního systému techniků a pilotů dle Part 66**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování:

Hlavním tématem práce je analýza vlastnoručně vytvořené webové aplikace za účelem vytvoření větší úrovně abstrakce. Kromě dodání samotného řešení aplikace tedy bude nutné provést zpětnou analýzu zaměřenou na diagramy struktury a chování a zpracování požadavků. Na konec bude potřeba dokázat, že daný systém dodržuje standardy a ustanovení pro implementaci informačního systému v letecké dopravě.

Rozsah grafických prací:

Rozsah pracovní zprávy:

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

Seznam odborné literatury:

Nash, Tray. C 2010. Rychlý průvodce novinkami a nejlepšími postupy. Brno: Comuter Press, 2010. 624 s. 978-80-251-3034-6.
ŘEPA, Václav. Podnikové procesy: Procesní řízení a modelování . 2., aktualizované a rozšířené vydání. Praha 7 : Grada Publishing, 2007. 281 s. ISBN
DAVENPORT, Thomas H. Process innovation: reengineering work through information technology. Boston: Harvard Business School Press, 1996. 323 s. 978-80-247-2252-8.

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

Datum zadání diplomové práce: **31. října 2012**
Termín odevzdání diplomové práce: **17. května 2013**

Univerzita Pardubice Česká republika

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

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

Prohlašuji:

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

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

v Pardubicích dne 27. 4. 2011

Petr Erben

Poděkování:

Touto cestou bych chtěl poděkovat Ing. Karlovi Šimerdovi za velmi cenné rady a připomínky k mé diplomové práci a za čas věnovaný mým konzultacím.

ANOTACE

Práce je věnována reinženýrství vlastnoručně vytvořeného školního systému techniků a pilotů dle Part 66 pro Ústav letecké dopravy spadající pod Fakultu dopravní ČVUT, popisu reinženýrství softwarových projektů a vlastního řešení reinženýrství školního systému techniků a pilotů dle Part 66 v podobě analytických diagramů. Teoretická část práce se zabývá popisu reinženýrství softwarových projektů a popisu reinženýrství školního systému techniků a pilotů dle Part 66. Praktická část se zabývá analýzou školního systému techniků a pilotů dle Part 66. V rámci práce bylo vytvořeno celkem 206 analytických diagramů popisujících architekturu a chování školního systému techniků a pilotů dle Part 66.

KLÍČOVÁ SLOVA

reinženýrství, analýza, software, model, diagram

TITLE

Reengineering of academic system for engineers and pilots according to Part 66

ANNOTATION

The thesis is dedicated to reengineering of self-made academic system for technicians and pilots according to Part 66 for the institute of aviation that comes under Faculty of transportation CTU, description of Reengineering of software projects and own solution of reengineering of academic system for technicians and pilots according to Part 66 in analytic diagrams. The theoretical part deals with the description of reengineering of software projects and the description of reengineering of academic system for technicians and pilots according to Part 66. The practical part deals with the analysis of academic system for technicians and pilots according to Part 66. There were created total of 260 analytic diagrams describing architecture and behaviour of this system in this thesis.

KEYWORDS

rengineering, analysis, software, model, iagram

Obsah

Seznam tabulek

Seznam obrázků

Seznam použitých zkratek

ERP	Enterprise Resource Planning
OOP	Objektově Orientované Programování
MVC	Model View Controller
DI	Dependency Injection
UML	Unified Modelling Language
CASE	Computer-Aided Software Engineering
SQL	Structured Query Language

Úvod

V dnešní době je vyvíjeno stále více softwarových řešení pro nejrůznější účely. Při tvorbě těchto systémů se programátoři, softwarový architekti, případně návrháři systému snaží, aby pro vývoj zvolili takový postup, aby byl z hlediska rozvoje systému co nejefektivnější.

Jedním ze základních cílů softwarového inženýrství je efektivní vytváření softwarových produktů. Důležitým faktorem při vytváření informačního systému je *metodika vývoje softwaru*. Jedná se o volbu správného souboru postupů, pravidel a nástrojů používaných pro návrh, plánování a řízení vývoje informačního systému.

Dalším velmi užitečným prvkem při vývoji softwaru je použití *návrhových vzorů*. Návrhový vzor představuje obecné řešení problému, které se využívá při návrhu programů. Použití návrhových vzorů nám umožňuje napsat kód, ve kterém je pro jiného člověka snazší se orientovat. Jelikož jsou to ověřené způsoby, je menší šance, že uděláme chybu.

Bohužel však ani ta nejlepší metodika, návrhový vzor nebo způsob psaní kódu s postupem času a při neustálém vývoji v oblasti informačních technologií nezabrání tomu, že se kód, ve kterém je informační systém napsaný, stane dříve nebo později zastaralý a pro další rozvoj systému dále neudržitelný.

Účelem práce je představit způsob, který tuto problematiku řeší. Tento způsob se nazývá *softwarové reinženýrství*. Jednotlivé informační systémy se liší charakterem a množstvím požadavků nebo prostředím, ve kterém jsou nasazené. Proto nelze jednoznačně vytvořit obecné řešení, které by se dalo použít pro obecnou množinu systémů. Existují však určité techniky a pravidla, které celý tento proces zpřehlední. Postupně si popíšeme jednotlivé fáze a charakteristiky reinženýrství. Důležitou součástí při tom bude nalézt chyby, kterých jsme se dopustili během vývoje systému. Chyby snadno odhalíme detailní analýzou celého systému, která nám umožní popsat daný systém vzhledem ke zvolené měřené veličině. Analýze softwaru bude proto také věnována jedna z kapitol.

Cílem praktické části je vytvoření procesu reinženýrství na základě detailní analýzy konkrétního softwarového projektu. Jedná se o webovou aplikaci školního systému pilotů a techniků dle Part 66, popsanou ve více než 250 diagramech chování a struktur, o které budu více pojednávat v jedné z kapitol. Při tvorbě aplikace bylo bohužel návrhu celého systému věnováno minimum času a při postupných, rostoucích požadavcích na systém se některé operace stávaly časově více náročné a kód méně přehledný. Tyto nedostatky mě vedly k hledání řešení, které detailně popíše celý systém a následně odhalí jeho chyby a nedostatky.

1. Softwarové reinženýrství

V této kapitole si vysvětlíme podstatu a důležitost softwarového reinženýrství. Následně si představíme cíle reinženýrství a chyby, kterých se můžeme dopustit během vývoje softwarového sytému, a na které se tato disciplína zaměruje. Nedílnou součástí bude i seznámení s životním cyklem softwarového reinženýrství, ve kterém si vysvětlíme, jak se má při tomto procesu postupovat. Důležité přitom bude poznat techniky, které můžeme při tomto procesu používat. Nakonec si ukážeme, jaké máme možnosti při realizaci nového řešení, vytvořeném v procesu softwarového reinženýrství.

1.1. Podstata

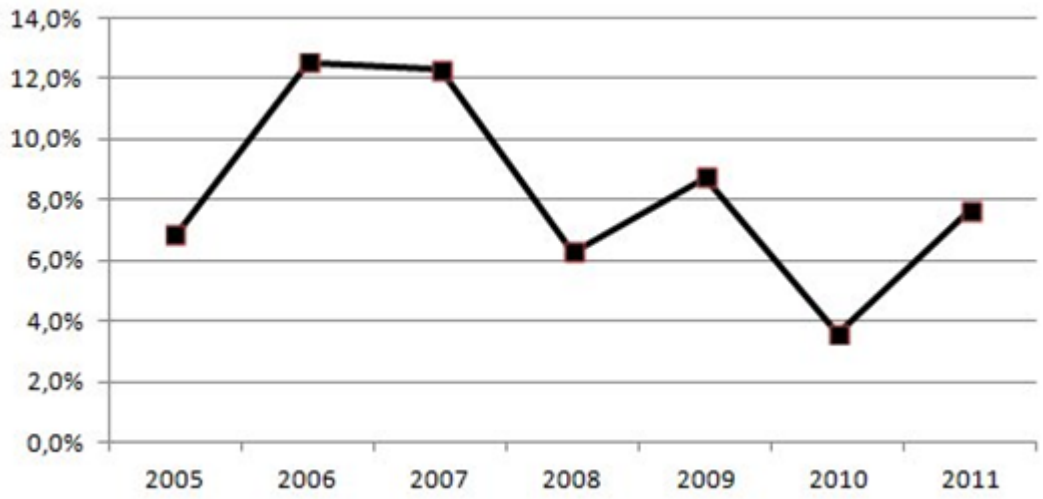
Tvorba softwarových systémů je komplikovaný a časově náročný proces v závislosti na velikosti vytvářeného systému. Během tvorby softwarového systému musí vývojáři dbát na celou řadu vnitřních, ale i vnějších faktorů. Mezi vnitřní faktory patří například:

a) Dovednost vývojářů vytvářet komplexní softwarové systémy, mnohdy tvořené přímo na míru.
b) Schopnosti a softwarové vybavení, které vývojáře omezuje na volbu konkrétního způsobu a postupu při vývoji systému a tím i cílovou platformu pro nasazení.

Mezi vnější faktory patří:

a) Hardwarové a softwarové vybavení pro cílové nasazení, ať už přímo u zákazníka nebo u hostingové společnosti.
b) Prostředky, kterými je zákazník schopen pokrýt vývoj daného softwarového systému.
c) Schopnost zákazníka ovládat daný systém a tím i jeho náklady spojené např. se školením zaměstnanců.

Podle výše uvedených faktorů je zřejmé, že vývoj softwarových systémů není snadná záležitost. Nelze se tedy divit, že v současné době se nevyvíjí tolik softwarových systémů, jako v předešlých letech. Následující jev je znázorněn na Obrázku 1.1.



Obr. 1. 1 - Vývoj českého ERP, převzato z [17]

Z grafu je zřejmé, že nových systémů spíše ubývá. Naopak přibývá více takových systémů, které fungují řadu let. Některé tyto systému vykazují řadu vad:

a) Nedostatečná dokumentace
b) Zastaralý kód
c) Chyby v návrhu systému

Jejich majitelé, jsou časem postaveni před nelehkou situaci, kdy se musí rozhodnout, zda daný systém nechat tak jak je, zda ho přepsat nebo zda ho úplně zahodit a vytvořit nový. Je třeba důkladně zvážit, která varianta bude pro majitele nejméně bolestivá.

a) Majitel nechá systém kompletně nezměněn. Náklady na inovaci systému jsou nulové. Některé nové funkcionality z rostoucích požadavků na systém jsou však velmi nákladné, některé vůbec nelze implementovat. Navíc s přibývajícími funkcionalitami se může zhoršovat dosavadní výkon celého systému.
b) Majitel se rozhodne starý systém zahodit a vytvořit úplně nový. Vývoj nového systému je velmi nákladný. Dokumentace současného systému je často zastaralá nebo nekonzistentní. Mnohdy také bývalí vývojáři již nejsou dostupní. Je tedy velmi pravděpodobné, že v nové verzi systému bude více chyb, než v té původní nebo budou chybět některé funkcionality.
c) Majitel se rozhodne pro přepsání (upravení) současného systému. Stejně jako v předešlém případě, kde se jednalo a vytvoření nové verze systému, i tento postup je pro majitele nákladný. Postupem času by se mu ale jeho investice měla vrátit v podobě většího a stabilního výkonu systému, menší chybovosti a menším nákladům na údržbu a inovaci jednotlivých funkcionalit.

Obecně platí, že každý softwarový systém má svou životnost od okamžiku, kdy byl vytvořen. Úkolem vývojářů, kteří daný systém vyvíjejí je, aby s použitím moderních, ale zároveň osvědčených postupů, tuto životnost co nejvíce prodloužili. Dříve či později se totiž každý vytvořený softwarový systém stane zastaralý a pro další rozvoj systému dále neudržitelný. V tu dobu přichází na řadu softwarové reinženýrství.

1.2. Cíle

Softwarový systém podstupující proces změny v podobě softwarové reinženýrství, by měl mít jeden z následujících cílů:

a) Zvětšení výkonnosti
b) Zvětšení stability
c) Zmenšení nákladů na provoz
d) Otevření nových možností

Nejčastější příčiny vedoucí ke ztrátě výkonu softwarového systému jsou:

a) Hardware – Disciplína softwarového reinženýrství se nezabývá změnou hardwaru, na němž je spuštěn daný softwarový sytém. Je však třeba jej zmínit, protože je to nedílná součást ovlivňující výkon celého sytému.

b) Architektura – Často je návrhu systému věnováno nedostatek času. Společnosti, které jsem během studie navštívil, nevěnovali návrhu při vývoji systému příliš mnoho času, protože to pro ně bylo nákladné. Bohužel si ale neuvědomují rizika, k nímž může dojít během dalšího vývoje systému. V případě, že se změní požadavky nebo přibydou nové, je třeba se vrátit na začátek návrhu dané části systému a neřešit problém lokálně, tzv. „záplatou“.

c) Kód - Velmi četným problémem jsou mnohdy úplně zbytečně komplikované a náročné dotazy do databáze pro získání konkrétních dat. Nemohu neuvést příklad, se kterým jsem se také setkal – Programátor, procházel kolekci. Při každém průchodu kolekce prováděl dotaz do databáze, se kterým dále pracoval. Nejen, že dotaz zahrnoval mnohem více dat, než ve skutečnosti programátor potřeboval, ale každý takovýto dotaz byl volán úplně zbytečně. Jaké se nabízí řešení pro zlepšení výkonu systému? Provést jen jeden dotaz do databáze, vybrat pouze ty data, které skutečně potřebuji a celý tento výsledek uložit do nějaké jiné kolekce. Při každém průchodu původní kolekce, tak ušetřím jeden dotaz do databáze, který navíc obsahuje pouze data, které skutečně potřebuji. V systémech, které obsahují tisíce, statisíce nebo milióny řádků kódu, není snadné se orientovat. Obzvláště pro programátora, který není autorem daného systému. V takovém případě je potřebná dostatečná dokumentace a model systému, díky kterým se programátor zorientuje. Bez těchto podkladů není

vyloučeno, že se například uvnitř jedné metody bude zbytečně volat více stejných metod, které navíc plní stejnou funkci.

V dnešní době, jsou uživatelé užívající softwarové systémy náročnější než dříve. Např. před 10 lety, když uživatel navštívil webovou stránku, byl ochoten pro načtení dané stránky čekat i několik sekund. Dnes, pokud se stránka nenačte maximálně do 2 sekund, uživatel ztrácí zájem a odchází. S tím je spojená i stabilita softwarového systému. Pokud se v softwarovém systému, který uživatel využívá, vyskytne chyba, kterou uživatel nezavinil, tak opět ztrácí motivaci k užívání daného systému.

Důležitým cílem může být i snížení nákladů na provoz softwarového sytému. Pokud například systém nespravují původní programátoři, může být pro nové programátory velmi nesnadné se v systému orientovat a o to více nesnadné provádět změny, které nezasáhnou funkčnost jiných částí systému. Takovéto zásahy do systému jsou pro majitele velmi nákladné a je třeba zvážit, zda se investice do přepsání systému v podobě softwarového reinženýrství během několika měsíců, případně let nevrátí.

Jednou z posledních motivací pro softwarové reinženýrství je otevření možnosti pro přidání nových funkcí. Ve starých softwarových systémech by bylo nemožné nebo při nejmenším velmi nákladné do systému implementovat novou funkcionalitu. Tato funkcionalita při tom může být již v základu implementována při použití jiného řešení.

1.3. Chyby

Pojďme se podrobněji zaměřit na příčinu softwarového reinženýrství a tou jsou mimo stáří softwarového systému také chyby, vyplívající z jeho vývoje.

1.3.1. Architektura

Modelování a návrhu architektury softwarového systému je potřeba věnovat nejvíce času. Je to nejdůležitější část vývoje softwarového systému, kde se převádějí funkční a nefunkční požadavky zákazníka na jednotlivé funkcionality celého sytému. Při softwarovém reinženýrství je potřeba všem těmto funkcionalitám porozumět, abychom je mohli bezpečně přepsat. Během přepisování jednotlivých částí systému můžeme narazit na následující nedostatky:

a) Špatný návrh – Nedostatečné porozumění všech požadavků zákazníka, nesprávné převedení požadavků zákazníka na jednotlivé funkcionality systému nebo nesprávné rozšiřování systému. Přesně tak může dojít k chybné architektuře systému, která bude mít za následek softwarové reinženýrství.

b) Nesprávná volba architektonického vzoru – Nejužívanějším architektonickým vzorem v dnešní době při vývoji softwarových systémů, je vzor MVC. Jedná se o softwarovou architekturu, která rozděluje systém na tři části – datový model, uživatelské rozhraní a řídící logiku. Některé ze systémů podstupující softwarové reinženýrství mají všechny tyto části systému promíchané. Je potom velmi pracné od sebe jednotlivé části oddělit, aby bylo možné přejít na jiné řešení.

1.3.2. Kód

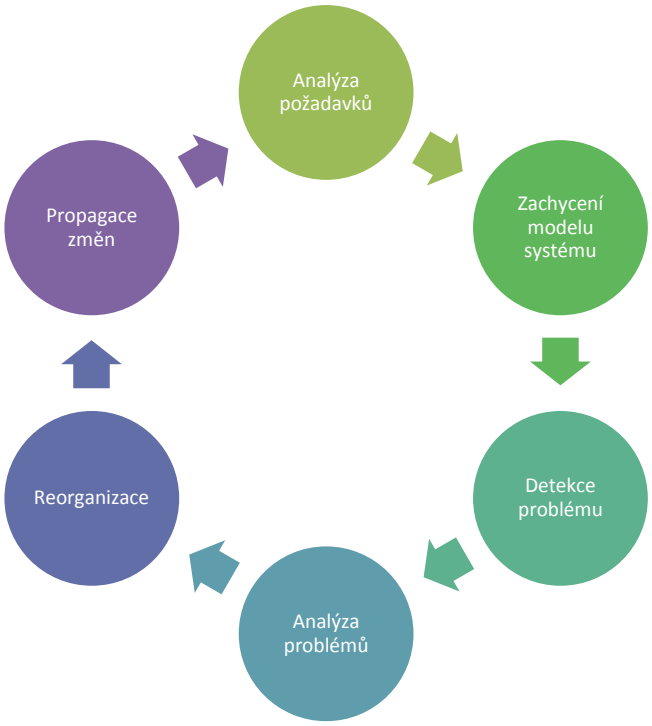
Při vývoji softwarového systému bychom měli dodržovat řadu osvědčených návrhových vzorů a postupů, abychom se vyvarovali nesrozumitelnému, neefektivnímu nebo dokonce chybovému kódu. Jedna z nejužívanějších vývojových metodik dnešní doby je Objektově orientované programování. OOP je metodika vývoje softwaru, která se dnes při vývoji softwarových systémů, pokud to dovoluje daný programovací jazyk. Stále se však lze setkat se softwarovými systémy, u který tato metodika nebyla použita. Takovéto systémy pak mohou vykazovat řadu vad. Velmi častou vadou je nemožnost zapouzdření tříd vykazující se mnohočetnou provázaností mezi třídami samotnými. Tento problém řeší velmi populární návrhový vzor nazvaný „Dependecy Injection“ (DI). Hlavním úkolem DI je odstranění závislostí mezi jednotlivými třídami a zavedení rozhraní definující společné chování tříd, které jej implementují. Výsledkem je nezávislost jednotlivých tříd při provádění požadovaných operací. Tyto třídy je pak možné rozdělit do skupin, modulů nebo podsystémů, které je mnohem snadnější přesunout nebo přepsat během procesu softwarového reinženýrství.

1.3.3. Dokumentace

Mnoho systému podstupujících softwarové reinženýrství čelí podobnému problému a nekonzistentní, nesrozumitelné, nedostatečné nebo dokonce neexistující dokumentaci. Bez těchto dokumentů jsou noví vývojáři v procesu softwarového reinženýrství ve velmi složité situaci a je malá šance, že přepsaný systém nebude obsahovat chyby.

1.4. Životní cyklus

Účelem procesu softwarového reinženýrství je nahradit celý systém nebo jeho jednotlivé části. Na obrázku 1.2 lze vidět postup, který je vhodné při tomto procesu dodržovat, neboť při jeho následování můžeme eliminovat značné riziko výskytu chyb.



Obr. 1. 2 - Životní cyklus softwarového reinženýrství, převzato z [6]

1.4.1. Analýza požadavků

Účel požadavků je vytvoření vyšší specifikace toho, co má systém dělat. Při specifikaci požadavků je nezbytná komunikace s majitelem systému. Ta bývá problematická hned z několika hledisek. Většinou totiž zákazník nemá jasnou a ucelenou představu o výsledném softwarovém systému. Softwarový inženýr je sice odborník ve svém oboru, ale většinou nemá žádné nebo velmi omezené znalosti v oboru, pro který je daný informační systém vyvíjen.

1.4.2. Vytvoření modelu

Při formulaci požadavků bývá velmi často používán přirozený jazyk, neboť je přirozeným komunikačním prostředkem mezi lidmi. Bohužel však v sobě zahrnuje určitou nejednoznačnost. Požadavky mohou být rozporné a mohou se měnit. Proto je třeba vyměnit tento komunikační prostředek za něco, čemu bude rozumět jak zákazník, tak programátor. Pro tyto případy byl vytvořen jazyk, který se nazývá Unified Modelling Language (UML). UML je univerzální jazyk pro vizuální modelování systémů. Pro každou část zkoumaného systému je potřeba detailně zanalyzovat požadavky a vytvořit několik pohledů (diagramů) z různých úhlů. Ne vždy však máme k dispozici dokumentaci nebo kompletní souhrn požadavků a je proto nezbytné vytvořit návrhové pohledy přímo ze zdrojového kódu.

1.4.3. Detekce problémů

Detekce problému bývá prvotním impulsem pro softwarové reinženýrství. Problémy nejčastěji spadají do kategorie výkonnost nebo stabilita. Pro zjištění příčiny těchto problémů můžeme využít model systému, který nám poodhalí výskyt těchto chyb. Další možnost je využití speciálních nástrojů pro měření výkonnosti softwarového systému. Takový nástroj je pak schopný simulovat běh zkoumaného softwarového systému a mnohokrát zopakovat celý jeho průběh. Následnou analýzou výsledků se dá velmi přesně zjistit, jaká jsou slabá místa systému a je už pak na programátorovi, jak se s nimi vypořádá.

1.4.4. Analýza problémů

Než pro každý nalezený problém navrhneme řešení, je třeba jej důkladně zanalyzovat a zjistit jeho podstatu. Mohlo by pak dojít k situaci, kdy následné řešení pouze zhorší chování celého systému nebo problematické části.

1.4.5. Návrh řešení

V této části procesu softwarového reinženýrství navrhneme řešení nalezeného problému. K tomuto účelů můžeme využít dobře známé a osvědčené návrhové vzory.

1.4.6. Aplikace změn

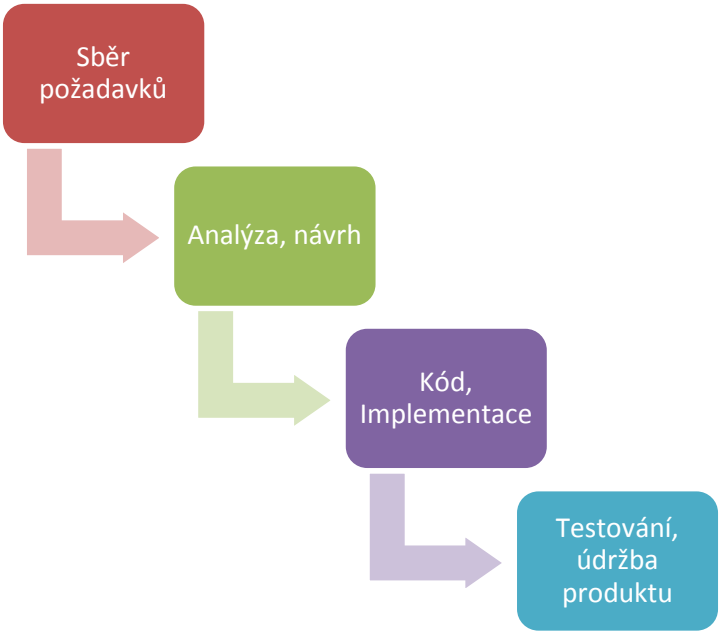
Poslední fáze představuje proces implementace navrhnutých řešení pro jednotlivé detekované problémy softwarového systému.

1.5. Techniky

Způsobů jak řešit proces softwarového reinženýrství určitého softwarového systému je mnoho. V následující části si vysvětlíme nejzákladnější techniky používané při tomto procesu.

1.5.1. Přímé inženýrství

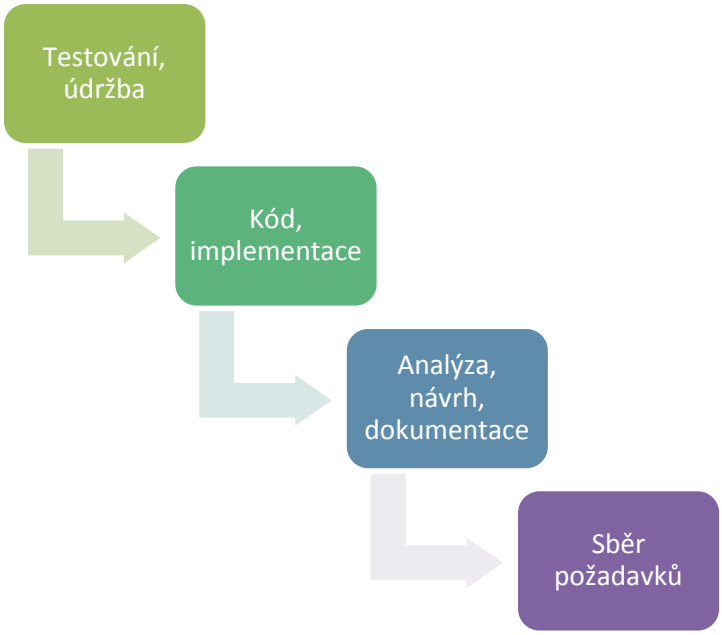
Proces přímého „klasického“ softwarového inženýrství představuje postup vývoje softwarového systému od požadavků, návrhu, implementace až k nasazení nového systému u zákazník a jeho následnou údržbu, jak je vidět na Obrázku 1.3.



Obr. 1. 3 - Technika přímého inženýrství, převzato z [6]

1.5.2. Zpětné inženýrství

Cílem zpětného inženýrství je zanalyzovat a zdokumentovat princip fungovaní zkoumaného systému za účelem vyšší úrovně abstrakce. Při nedostatečné dokumentaci zkoumaného systému postaveného před proces softwarového reinženýrství je tato technika nezbytná pro pochopení všech částí systému.



Obr. 1. 4 - Technika zpětného inženýrství, převzato z [6]

Na Obrázku 1.4 lze vidět, že v případě zpětného inženýrství se postupuje opačně, než jak je tomu u přímého inženýrství. Zkoumaný systém tedy popisujeme od nejnižší vrstvy abstrakce až po nejvyšší. Úspěch tohoto procesu tedy záleží především na programátorovi, který analyzuje zdrojový kód a postupně rekonstruuje principy fungování jednotlivých částí systému.

Mezi jednotlivé cíle zpětného inženýrství patří:

a) Inspekce systému
b) Identifikace jednotlivých částí systému a jejich vzájemné interakce
c) Vytvoření vyšší úrovně abstrakce
d) Vytvoření dokumentace

1.5.3. Restrukturalizace

Restrukturalizace jako technika používaná při procesu softwarového reinženýrství se zabývá úpravou současného kódu softwarového systému do přehlednější, srozumitelnější a implementačně výhodnější podoby. K dosažení toho efektu můžeme využít řadu nástrojů, které programátorovi při psaní kódu usnadní mnoho práce.

1.5.4. Refaktorizace

Hlavním úkolem refaktorizace je znovupoužitelnost a snadnější údržba kódu zkoumaného softwarového systému. Jedná se o reorganizaci objektů do více komplexnějších částí. Cílem tedy je modularizovat jednotlivé části kódu do samostatných transformací pro jejich budoucí znovupoužitelnost a snadnější údržbu. Ideálním řešením při této části procesu softwarového reinženýrství je použití návrhových vzorů.

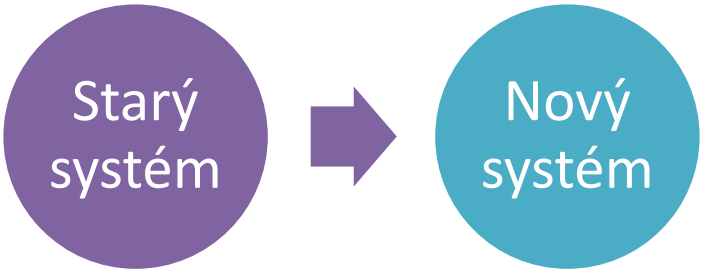
1.6. Způsoby realizace

Jakmile máme připravené vše potřebné, je na čase se pustit do realizace. V následující části si představíme tři základní způsoby realizace softwarového reinženýrství.

1.6.1. Metoda velkého třesku

Metoda velkého třesku, taktéž označována jako „Big Bang“. Při této realizaci bývá původní systém nahrazen novým v jediném kroku, jak je vidět na Obrázku 1.5.

Ne vždy může být tato metoda optimální, např. u rozsáhlých projektů může

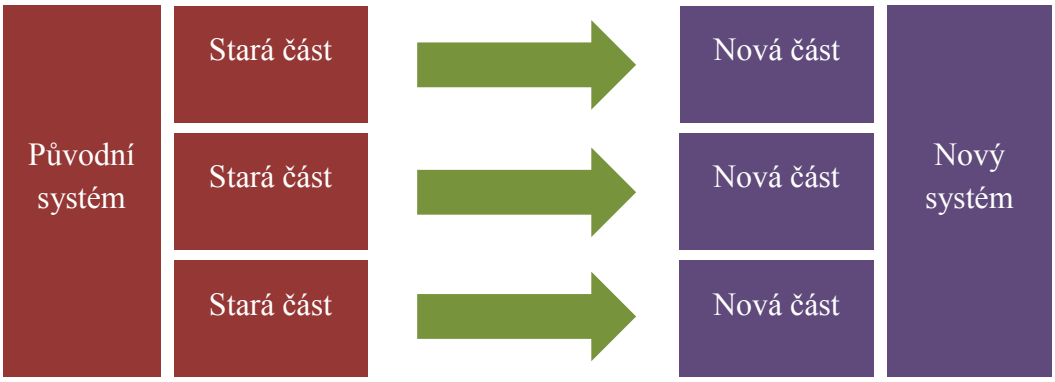


Obr. 1. 5 - Realizace metodou velkého třesku, převzato z [9]

tato metoda představovat až příliš velké riziko spojené s vynaložením značného množství času a nákladů. Podstatnější problém představuje údržba systému. Nelze vyloučit, že v během vývoje nového systému proběhnou v původním systému změny, která je nutné přesunout také do nového systému, nebo alespoň do částí, kterých se dotýká proces softwarového reinženýrství. Výhodou však zůstává to, že původní systém je nahrazen v jediném kroku a pro nový systém není třeba udržovat alternativní prostředí pro běh starých částí systému.

1.6.2. Inkrementální přístup

V případě inkrementálního přístupu se jedná výměnu systému po částech, jak je znázorněno na Obrázku 1.6. Při této realizaci se postupně původní části systému nahrazují novými, tedy nové části jsou do systému přidávány inkrementálně spolu s tím, jak vznikají nové požadavky.



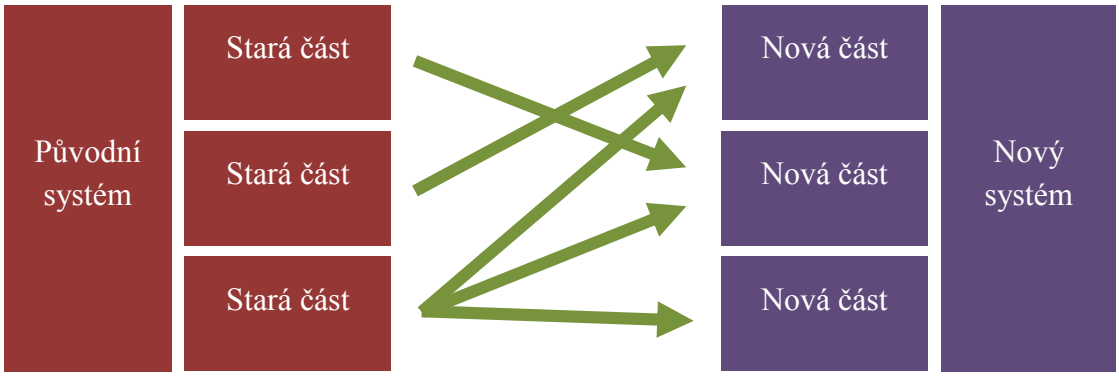
Obr. 1. 6 - Realizace s pomocí inkrementálního přístupu, převzato z [9]

Na rozdíl od metody „Velkého Třesku“ nemůže být změněna celková struktura systému. Změněna tedy může být pouze struktura jednotlivých částí systému, které jsou právě nahrazovány. Přepisované části jsou od systému izolované. Tím pádem změny v částech, které nejsou nahrazovány, se nijak nedotknou částí, které nahrazovány jsou.

Na realizaci procesu softwarového reinženýrství pomocí inkrementálního přístupu je potřeba větší množství času, neboť je potřeba evidovat nové a původní části systému. Z toho důvodu představuje tento přístup menší riziko. Díky tomu, že jsou jednotlivé části systému monitorovány, lze snadno odhalit případný výskyt chyb.

1.6.3. Evoluční přístup

Stejně jako u realizace inkrementálního přístupu jsou u realizace evolučního přístupu jednotlivé části systému postupně nahrazovány novými částmi. Rozdíl je v tom, že výběr nahrazovaných částí je prováděn s ohledem na jejich funkcionalitu, nikoli na jejich výskyt v původním systému. Na Obrázku 1.7 je vidět funkcionální provázanost původních částí systému do nových částí.



Obr. 1. 7 - Realizace pomocí evolučního přístupu, převzato z [9]

Z obrázku je zřejmé, že kvůli provázanosti jednotlivých funkcí, musí být tyto funkce identifikovány v rámci celého systému, což může představovat větší časovou náročnost. Naproti tomu jsou jednotlivé části systému funkčně rozděleny do několika komponent, díky kterým se sníží náklady spojené s údržbou a rozvojem systému.

2. Modelování softwarových systému

V této kapitole si ukážeme, k čemu nám při slouží jazyk UML a proč je tak důležitý při tvorbě modelů softwarového sytému. Dále si představíme základní stavební kameny těchto modelů, kterými jsou diagramy, a ukážeme si všechny druhy těchto diagramů, které byly vytvořeny do verze jazyka UML 2.0.

2.1. UML

Jazyk UML (Unified Modeling Language, unifikovaný modelovací jazyk) je univerzální jazyk pro vizuální modelování systémů [3]. UML podporuje objektově orientovaný přístup k analýze, návrhu a popisu procesů a systémů. Účelem jazyka UML je spojit současné existující postupy modelovacích technik a softwarového inženýrství.

UML neobsahuje, žádný způsob, postup ani metodiku vývoje softwarových systémů, poskytuje pouze vizuální syntaxi, kterou můžeme využít při sestavování našich modelů.

V dnešní době se při tvorbě rozsáhlejších a sofistikovanějších softwarových systémů neobejdeme bez CASE nástrojů, jenž nám umožňují detailní popis a modelování celého softwarového systému. Velikou výhodou jazyka UML při tom je, že je navržen tak, aby jej všechny tyto CASE nástroje mohly implementovat. Jazyk UML je tedy srozumitelný nejen pro lidi, ale i pro programy CASE. V softwarovém inženýrství této výhody využívá hlavně analytik, který je schopný transformovat požadavky klienta pomocí CASE nástroje do jazyka UML, kterému zase rozumí programátor.

Jedna z metodik využívající CASE nástroje při tvorbě softwarových systémů se jmenuje Unified Process. Tato metodika nám sděluje, jaké postupy máme využít, jaké pracovníky zaměstnat, jaké činnosti vykonat, jaké produkty vyrobit, abychom sestavili funkční model softwarového systému [3].

Jazyk UML je tedy velmi cenným prostředníkem, schopným přenášet myšlenky analytika, programátora ale i zákazníka do přehledné a srozumitelné formy v podobě diagramů různých druhů, závislých na způsobu pohledu na softwarový systém.

2.2. Diagramy

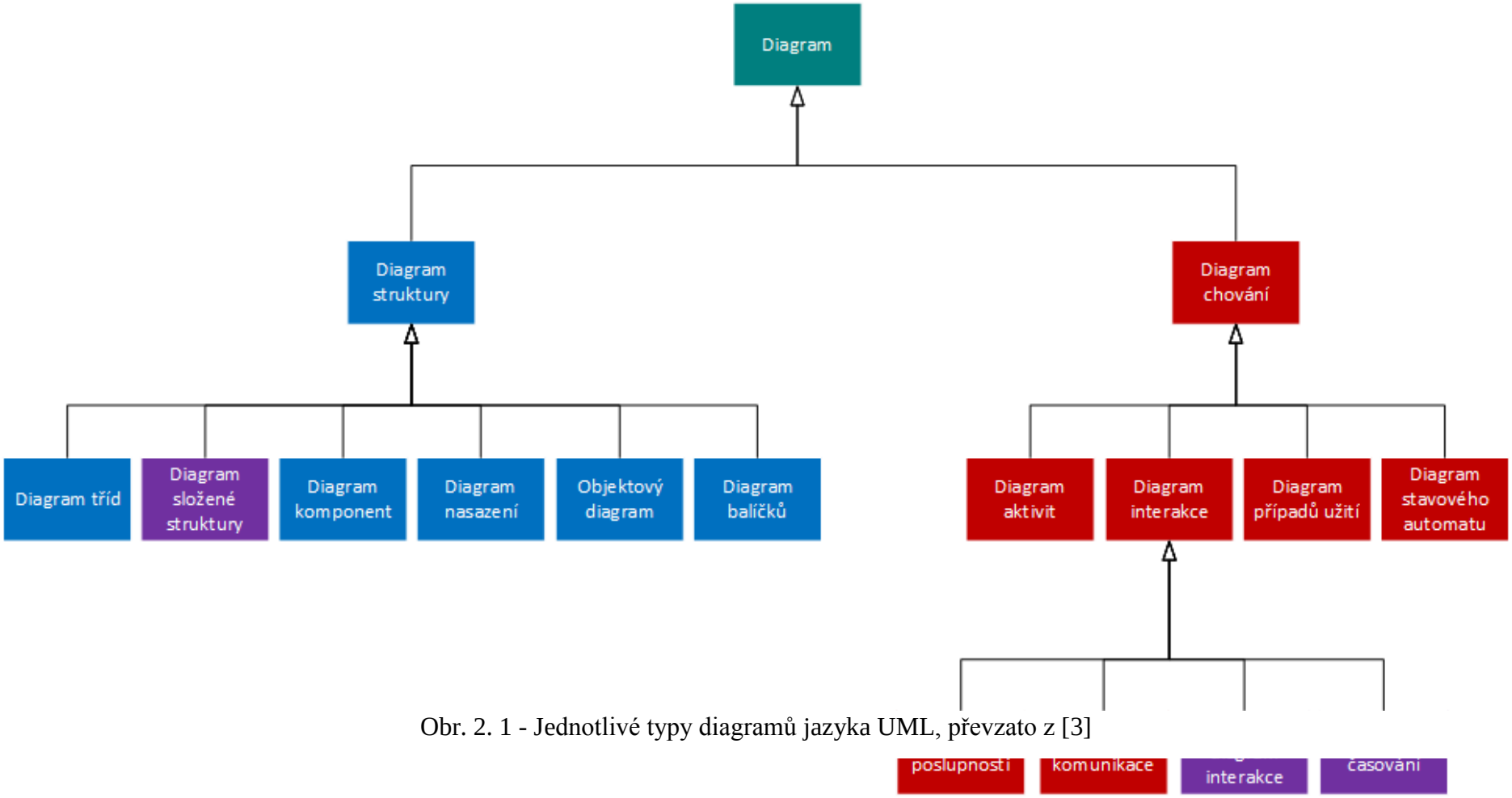
Diagramy jsou nejznámější a nejpoužívanější částí standardu. Představují pohledy na různé části sémantického základu navrhovaného systému. Sémantický základ je souhrn specifikací systému, který vymezuje hranice, v němž se můžeme při návrhu pohybovat. Každý nově vytvořený prvek nebo relace jsou přidávány do

modelu. Model je pak souhrnem všech těchto prvků a relací mezi nimi, které slouží k tomu, aby popisovali chování softwarového systému.

2.3. Typy diagramů

Na Obrázku 1.8 můžete vidět, že jazyk UML definuje třináct základních typů diagramů, rozdělených do dvou kategorií:

a) Diagramy chování - definují statickou strukturu systému
b) Diagramy struktur - definují dynamickou strukturu systému



Obr. 2. 1 - Jednotlivé typy diagramů jazyka UML, převzato z [3]

2.3.1. Diagramy struktur

Diagramy struktur definují statickou strukturu systému. Účelem těchto diagramů je zachycení předmětů a strukturní asociace mezi těmito předměty. Mezi diagramy struktur patří:

a) Diagram tříd – Diagram tříd popisuje základní stavební kameny každého objektově orientovaného systému. Diagram především ukazuje statický pohled na model nebo část modelu a jeho úkolem je znázornit typy objektů a jejich vztahy. Vzhledem k tomu, že tento diagram zachycuje pravidla modelovaného systému, je nepostradatelným podkladem při přímém i zpětném inženýrství.

   Praktická ukázka diagramu tříd je znázorněna v **Příloze C: Diagramy tříd**.

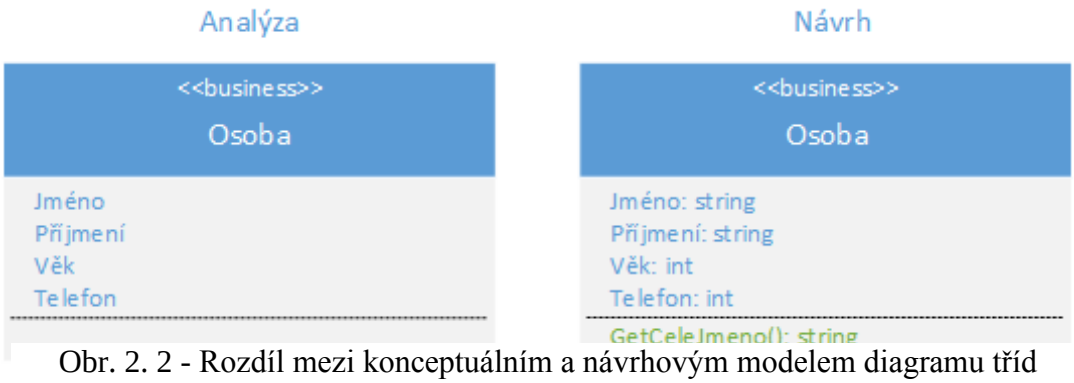
   U diagramu tříd rozlišujeme 3 základní typy:

   I. Konceptuální model – Jeho hlavním úkolem je analýza požadavků. Tento model obsahuje pouze tzv. byznys třídy (business classes), u kterých uvádíme pouze názvy klíčových atributů a některých klíčových metod.

   II. Návrhový model – Také znám jako tzv. Design model, který rozšiřuje původní analytické třídy o viditelnost atributů, datové typy, metody, původní třída se může rozdělit na několik dalších, apod.

   III. Implementační model – obsahuje veškeré implementační charakteristiky potřebné pro případné vygenerování zdrojového kódu v daném programovacím jazyce.

b) Diagram složené struktury – Diagram složených struktur je jednou z novinek, které přináší UML 2. Jedná se o diagram, který zobrazuje vnitřní strukturu interního prvku, včetně jeho interakce s ostatními částmi systému. Diagram složených struktur popisuje způsob, jakým mohou být třídy zobrazeny jako kompozitní prvky odhalující rozhraní a obsahující porty a části. Součástí toho diagramu mohou být také části, kolaborace, rozhraní, porty, delegáty, atd.



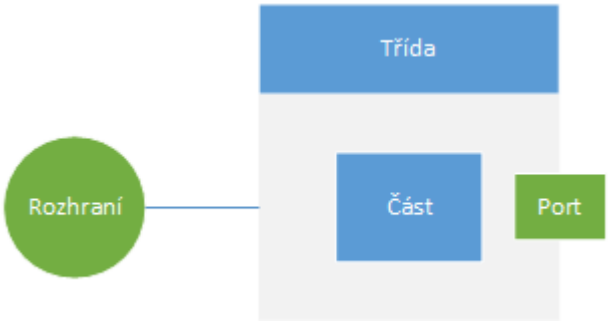
Obr. 2. 2 - Rozdíl mezi konceptuálním a návrhovým modelem diagramu tříd

I. Části – Reprezentují jednu nebo více instancí, které jsou součástí konkrétního klasifikátoru.

II. Rozhraní – Rozhraní je podobné třídě, ale s řadou omezení. Všechny operace rozhraní jsou veřejné a abstraktní a neposkytují žádnou výchozí implementaci a také všechny atributy rozhraní musí být konstanty. Třída může dědit pouze z jedné třídy, ale může implementovat více rozhraní. Třída a rozhraní, které daná třída implementuje, mají mezi sebou uzavřenou „dohodu“, kde rozhraní třídě říká: „Implementuj moje metody a já tě za to budu zastupovat“.

III. Porty – Pomocí portů probíhá interakce daného klasifikátoru s okolím.

IV. Kolaborace – Kolaborace zastupují nějakou funkcionalitu systému a jsou s ostatními prvky diagramu propojeny pomocí konektorů nebo asociací.

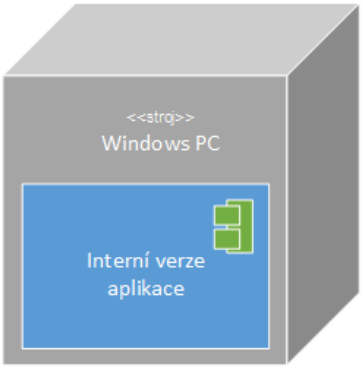


Obr. 2. 3 - Diagram složené struktury

c) Diagram komponent – Diagram komponent znázorňuje komponenty a zobrazuje vztahy mezi nimi a jednotlivými částmi systému. Komponenty jsou reprezentovány třídami, mohou obsahovat atributy i metody a na venek komunikují pomocí rozhraní.

Obr. 2. 4 - Diagram komponent

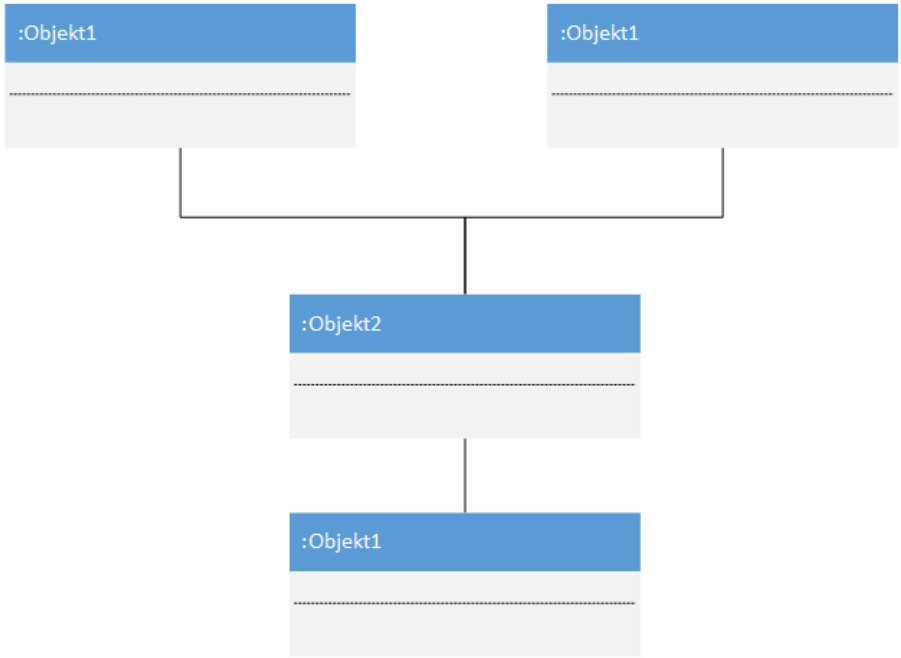
d) Diagram nasazení – Diagram nasazení modeluje run-time architekturu. Tento diagram ukazuje konfiguraci jednotlivých prvků (uzlů) hardwaru a také ukazuje, jak jsou na tyto uzly napojeny jednotlivé části systému.



Obr. 2. 5 - Diagram nasazení

e) Objektový diagram – Jedná se o zvláštní případ diagramu tříd. Objektový diagram používá podmnožinu prvků diagramu střídů s cílem zdůraznit vztah mezi instancemi tříd v určitém okamžiku v čase. Používá se hlavně pro znázornění konfigurace vzájemně propojených objektů ve zvláštních situacích, kde diagram tříd nestačí.

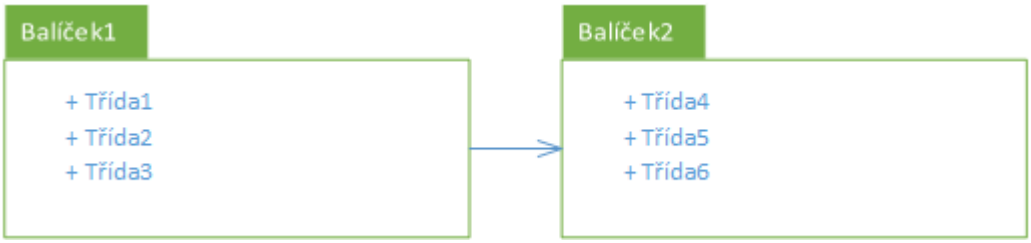
   Praktická ukázka objektového diagramu je znázorněna v **Příloze D: Objektové diagramy**.



Obr. 2. 6 - Objektový diagram

f) Diagram balíčků – Diagram balíčků je používán tak, aby odrážel organizaci balíčků a jejich prvků. Tento diagram sdružuje jednotlivé elementy do skupin, které představují balíčky, a znázorňuje mezi nimi závislosti.

   Praktická ukázka diagramu balíčků je znázorněna v **Příloze B: Diagram balíčků**.



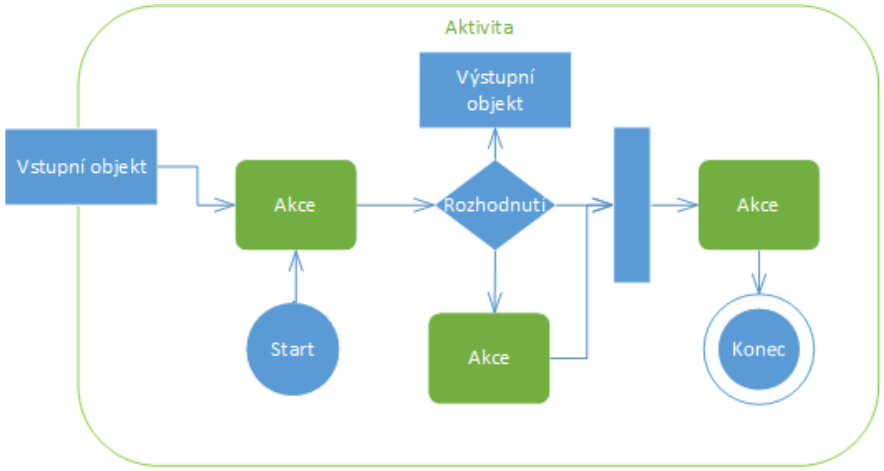
Obr. 2. 7 - Diagram balíčků

2.3.2. Diagramy chování

Diagramy chování definují dynamickou strukturu systému. Hlavním úkolem těchto diagramů je zachycení způsobu, jímž na sebe jednotlivé předměty působí tak, aby bylo dosaženo chování softwarového systému. Mezi jednotlivé diagramy chování patří:

a) Diagram aktivit – Diagram aktivit slouží k zobrazení posloupnosti činností. Diagram znázorňuje postup z počátečního bodu do koncového, při čemž postupuje přes mnoho rozhodovacích cest, které jsou popsány procedurální logikou.

   Praktická ukázka diagramu aktivit je znázorněna v **Příloze E: Diagramy aktivit**.



Obr. 2. 8 - Diagram aktivit

b) Diagram interakce – Jedná se o speciální případ diagramu aktivit, kde jednotlivé uzly představují diagramy interakcí. Diagramy interakce mohou zahrnovat sekvence, komunikace, a časování. Hlavním účelem diagramu interakce je zachytit tok v rámci řízení procesu nebo systému.

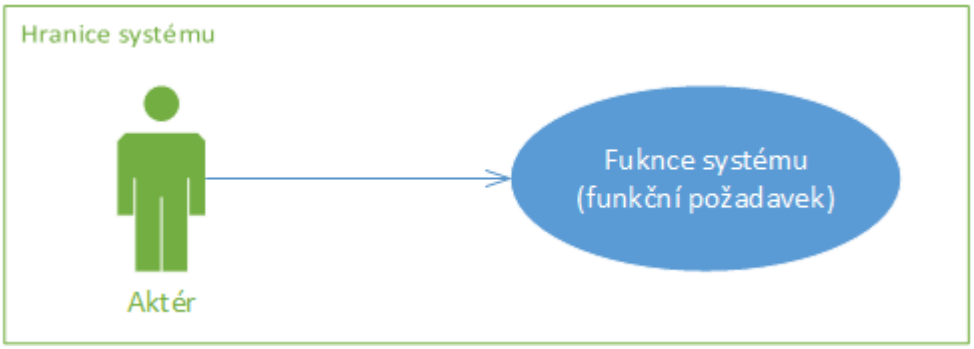
c) Diagram případů užití – Diagram případů užití zachycuje požadavky na systém. Případy užití jsou prostředkem komunikace všech účastníků systému s jeho jednotlivými funkčnostmi. Předpokladem toho diagramu je tedy zachycení všech požadavků na systém. Požadavky rozeznáváme dvojího druhu a to funkční a nefunkční. Diagram případů užití zachycuje pouze funkční

požadavky, neboť jen ty zahrnují jednotlivé funkcionality systému. Naproti tomu nefunkční požadavky nám sdělují omezení jednotlivých funkcionalit.

Praktická ukázka diagramu případů užití je znázorněna v **Příloze D: Diagramy případů užití**.

Při tvorbě diagramu případů užití je vhodné dodržovat následující postup:

I. Nalezení hranic systému
II. Vyhledání aktérů
III. Nalezení případů užití
    i. Specifikace případů užití
    ii. Určení alternativních scénářů
IV. Tento postup je potřeba opakovat tak dlouho, dokud se případy užití, aktéři a hranice systému neustálí.



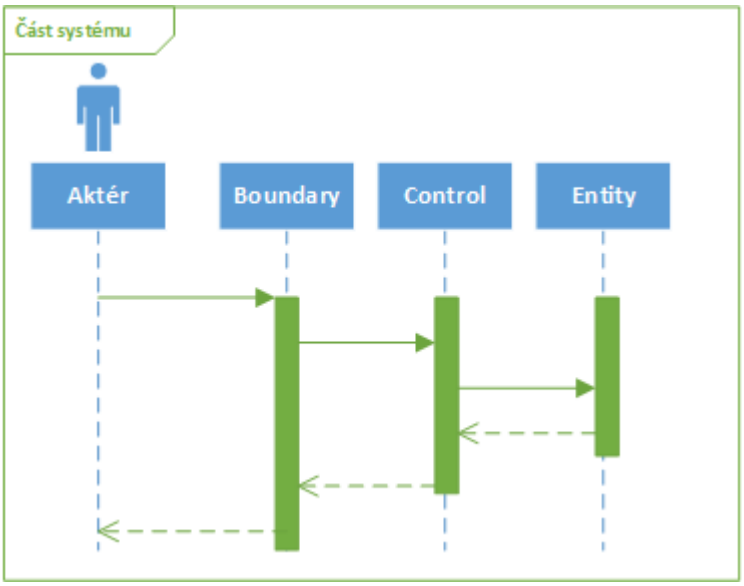
Obr. 2. 9 - Diagram případů užití

d) Diagram stavového automatu – Diagram stavového automatu zachycuje chování jednoho objektu v jeho životním cyklu a také reakci toho objektu na různé události.

Obr. 2. 10 - Stavový diagram

e) Diagram posloupnosti – Sekvenční diagram nebo diagram posloupnosti je jedním z nejčastěji používaným diagramem mezi diagramy interakcí. Sekvenční diagramy nejsou určeny pro zobrazování komplexní logiky procesu, ale skvěle se hodí pro monitorování objektů, které komunikují s ostatními objekty a zprávami, které tyto komunikace spouštějí.

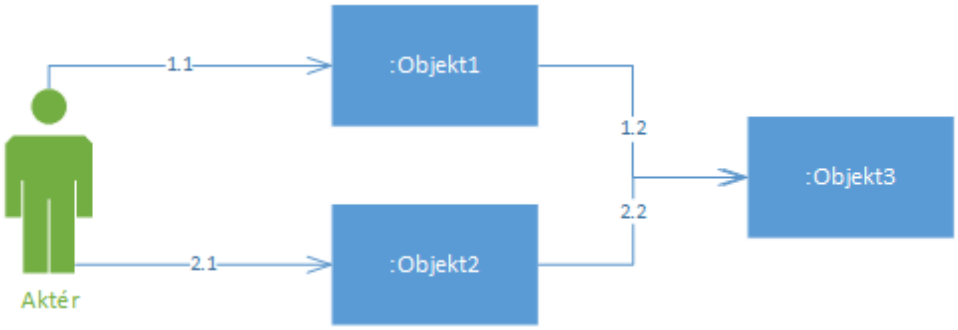
Praktická ukázka diagramu posloupnosti je znázorněna v **Příloze J: Sekvenční diagramy**.



Obr. 2. 11 - Diagram posloupností

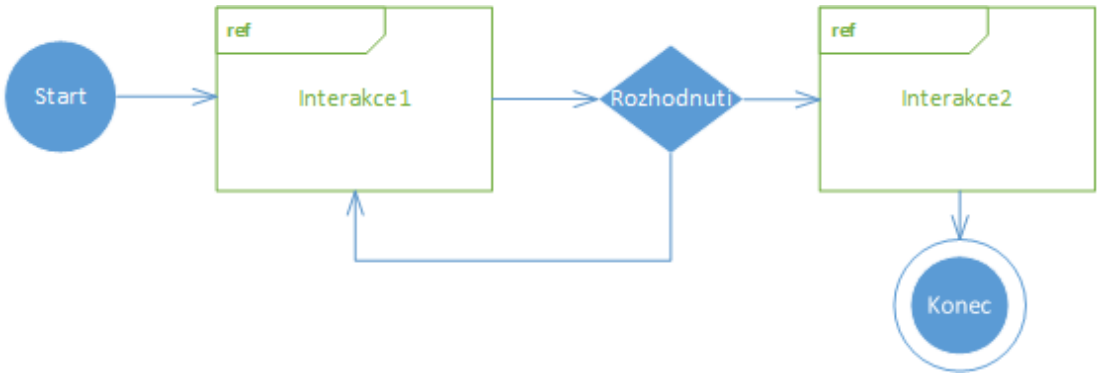
f) Diagram komunikace – Komunikační diagram je jedním z diagramů interakcí. Podobně jako sekvenční diagram, komunikační diagram také monitoruje komunikaci objektů zasahujících do systémů, ale více se zaměřuje na vztahy mezi objekty. Diagram komunikace umožňuje libovolné rozložení objektů, spojuje je a pro znázornění pořadí jejich komunikace používá číslování.

Praktická ukázka diagramu komunikace je znázorněna v **Příloze I: Komunikační diagramy**.



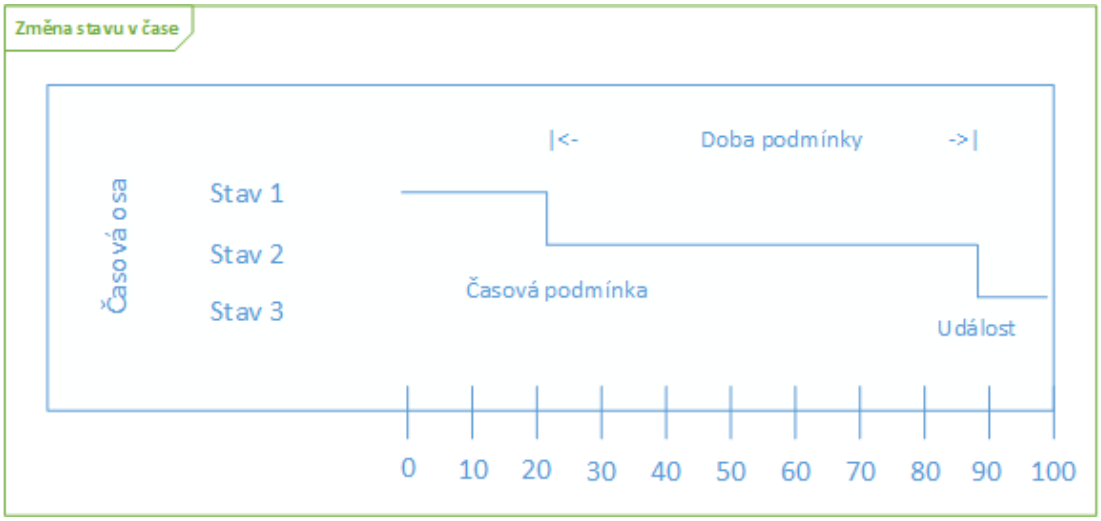
Obr. 2. 12 - Komunikační diagram

g) Stručný diagram interakce – Stručný diagram interakce je novým diagramem od UML 2. Jedná se o speciální případ diagramu aktivit. Tento diagram zachycuje průběh jednotlivých funkcí systému. Každý uzel diagramu může být další interakce.



Obr. 2. 13 - Diagram interakce

h) Diagram časování – Diagram časování patří do skupiny diagramů interakcí. Používá při zobrazení změny stavu nebo hodnoty jednoho nebo více objektů v průběhu času.



Obr. 2. 14 - Diagram časování

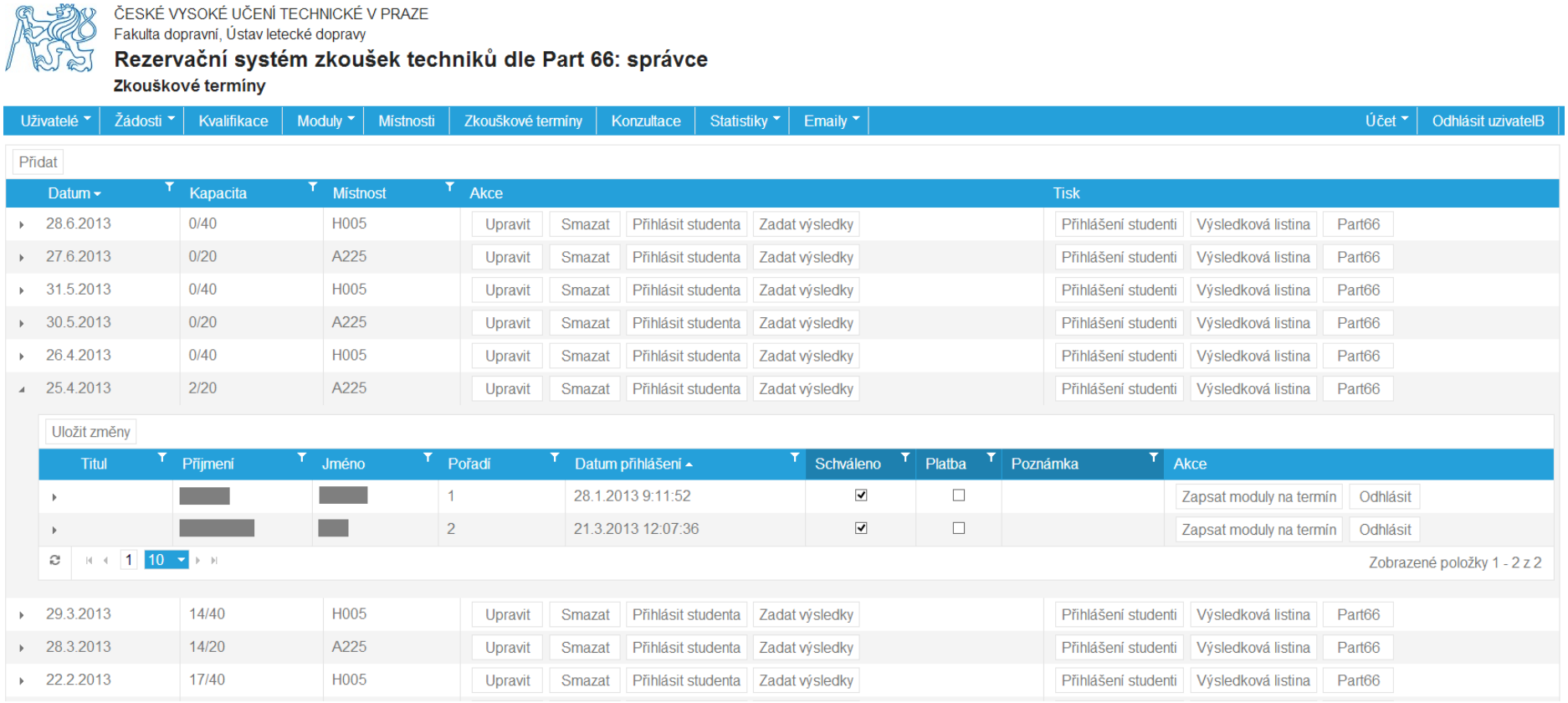
3. Reengineering školního systému techniků a pilotů podle Part66

V této kapitole si podíváme praktickou část diplomové práce a to softwarové reinženýrství školního systému techniků a pilotů podle Part66. V první části kapitoly si představíme, k čemu aplikace slouží a jaké nástroje a technologie byly použity na její zhotovení. V další části poukážeme slabá místa aplikace a vysvětlíme si, proč nebyly odhaleny během testování. Důležité bude pro tyto nedostatky navrhnout efektivní řešení a v poslední části si pak ukážeme implementaci navržených změn.

3.1. Představení

Samotná aplikace slouží kantorům na Ústavu letectví spadající pod Fakultu Dopravní ČVUT pro vytváření nejrůznější modulů a zkoušek. Technici a piloti se na tyto zkoušky přihlašují a aplikace na základě jejich výsledků vytváří různé statistiky a zprávy, které je možné tisknout pro inspekci ministerstva dopravy. Aplikace je podle způsobu oprávnění rozdělena na 3 části pro studenta, administrátora a auditora.

Na obrázku 3.1 můžete vidět ukázku aplikace (jména studentů nebyla zveřejněna). V této části aplikace administrátor vypisuje zkoušky, na které se přihlašují jednotliví studenti. V této části může administrátor přidávat, upravovat a mazat jednotlivé zkouškové dny, může také ručně přihlásit nebo odhlásit některého studenta na zkoušku, zamítnout jeho účast na zkoušce, může zadat výsledky testu a tisknout nejrůznější dokumenty související s daným zkouškovým dnem, přičemž jeden z těchto dokumentů se jmenuje Part66.



Obr. 3. 1 - Ukázka aplikace, převzato z [18]

3.1.1. Technická specifikace

Pro zhotovení aplikace byly použity programovací, popisné, skriptovací a stylovací jazyky a frameworky:

a) .NET Framework 4.0
b) ASP.NET MVC 3.0
c) Entity Framework 4.0
d) HTML5
e) CSS 3
f) jQuery 1.7.1
g) Telerik Tools for ASP.NET MVC, Autofac, NLog

Jedním z cílů při tvorbě aplikace bylo použít nejaktuálnější a zároveň nejosvědčenější jazyky a frameworky, které byly dostupné v roce 2012. Důraz byl kladen na rozšiřitelnost celého systému a prodloužení jeho životního cyklu a zároveň tak oddálení procesu softwarového reinženýrství.

Při samotném programování aplikace byly použity následující nástroje:

a) MS Visual Studio 2012
b) JetBrains Resharper, dotTrace
c) Stylecop
d) Git
e) YouTrack

Uvedené nástroje pomáhají programátorovi, tvořícímu aplikaci pro .NET Framework, udržet kód přehledný, odpovídající standardům a popsaný dokumentačními komentáři. Jako verzovací systém byl použit Git, který vedle zálohování umožnuje také spravovat různé verze systému.

Systémové požadavky:

a) MS IIS 7.5
b) MS SQL Server 2008 R2

3.2. Analýza požadavků

Po 2 měsících provozu systému se ozval majitel aplikace s dvěma požadavky na systém:

a) Vytvoření dokumentace jako součást přípravy pro rozšíření systému kvůli jeho budoucímu napojení na jiný informační systém generující obsah zkoušek.
b) Zlepšit výkon systému při přihlašování studenta na zkoušku.

Hlavní problém byl v tom, že během vývoje aplikace nebyla vytvořena žádná dokumentace a vzhledem k velikosti systému bylo velmi riskantní pouštět se do jakýchkoli rozsáhlejších zásahů do systému i přes to, že autorem jsem byl já sám. Jen pro informaci dodám, že vzhledem k množství nefunkčních požadavků na část systému pro přihlašování studenta na zkoušku má služba obsluhující tuto část přes 1200 řádků kódu.

Pro budoucí rozšiřitelnost a orientaci v systému tedy bylo potřeba vytvořit detailní dokumentaci celého systému na jejíž základě bude pro programátora mnohem snazší provádět změny bez obavy narušení některé ze stávajících funkcionalit. Bylo tedy nutné vyhledat inženýrskou disciplínu, která řeší tento problém a tou je softwarové reinženýrství.

3.3. Vytvoření modelu

Hlavním úkolem této části bylo vytvořit detailní pohledy na systém, které usnadní programátorovi pohyb v celém systému vytvořením mnohem větší úrovně abstrakce.

Celou aplikaci bylo nutné koncepčně rozdělit na části podle oprávnění uživatele a podle funkcionalit jak je znázorněno v Tabulce 3.1:

Auditor	Správce	Student
Emaily	Emaily	Úvod
Emaily šablony	Emaily šablony	Manuál
Úvod	Úvod	Stav
Konzultace	Konzultace	Zkoušky
Kvalifikace	Kvalifikace	Účet
Moduly	Moduly	Žádosti
Moduly podmínky	Moduly podmínky	
Místnosti	Místnosti	
Statistiky	Statistiky	
Uživatelé	Uživatelé	
Výsledky	Výsledky	
Zkoušky	Zkoušky	
Účet	Účet	
Žádosti	Žádosti	

Tabulka 1 - Jednotlivé části systému

Z tabulky je patrné, že uživatel Auditor a uživatel Správce pohlížejí na stejné části systému. Rozdíl mezi těmito typy uživatelů je v tom, že Správce může navíc v systému provádět změny, zatímco Auditor pouze sleduje, co se v systému odehrává.

Před vypsáním samotných diagramů, které byly vytvořeny v rámci celé aplikace, by bylo vhodné zavést jakousi pomyslnou stupnici, která znázorňuje důležitost nebo nepostradatelnost daného diagramu v procesu softwarového reinženýrství. Pro následující diagramy tedy byly vytvořený tyto stupně hodnocení:

a) 1 – Diagram doplňuje dokumentaci.
b) 2 – Diagram zpřehledňuje jednotlivé části systému.
c) 3 – Je velmi vysoká pravděpodobnost, že bez tohoto typu diagramu dojde k chybě.

Pro každou část aplikace byly vytvořeny následující diagramy:

a) Diagramy tříd – Díky namodelování diagramů tříd lze nyní na systém pohlížet s dostatečnou a přehlednou úrovní abstrakce. Budoucí programátoři nebo návrháři systému mohou nyní daleko pružněji a bezpečněji manipulovat s architekturou systému.

   Stupeň hodnocení: 3

b) Objektové diagramy – Objektové diagramy napomáhají odhalit objekty a jejich vztahy. Chyba odhalená pomocí toho typu diagramu je znázorněná na Obr. 3. 3 - Objektový diagram pro část zkoušky.

   Stupeň hodnocení: 3

c) Diagramy aktivit – Díky diagramů aktivit lze na jednotlivé části systému pohlížet jako na sekvence jednoduchých logických kroků, které jsou třeba k dosažení požadovaného chování systému.

   Stupeň hodnocení: 2

d) Databázový diagram – Diagram databáze je velmi důležitým architektonickým prvkem celé dokumentace. Za pomocí toho diagramu jsem odhalil způsobenou cirkulární referencí mezi tabulkami. Chyba odhalená pomocí toho typu diagramu je znázorněná na Obr. 3. 2 - Databázový diagram aplikace.

   Stupeň hodnocení: 3

e) Diagramy požadavků – Diagramy požadavků byly velmi důležitou součástí procesu softwarového reinženýrství. Především se jedná o nefunkční požadavky, které by bez modelu diagramů požadavků byly ze zdrojového kódu velmi těžko čitelné.

   Pokud se například nějaký programátor rozhodne kompletně předělat již existující systém bez dokumentace požadavků, s určitou jistotou se dá říct, že z hlediska funkčních požadavků, čili jednotlivých funkcí, bude systém fungovat jako dřív. Velký problém však nastane, když majitel systému zjistí, že systém už neobsahuje všechna omezení plynoucí z nefunkčních požadavků, které v dřívější verzi systému byla.

Model diagramů požadavků je tak jednou z nejdůležitějších částí dokumentace, obzvlášť v případě softwarového reinženýrství.

Stupeň hodnocení: 3

f) Diagramy případů užití – Pomocí diagramů případů užití došlo k zadokumentování všech funkcí a jejich vzájemné interakce se všemi aktéry systému.

Stupeň hodnocení: 2

g) Diagramy sledování požadavků – Diagram sledování požadavků spojuje všechny funkční požadavky s jednotlivými případy užití a tím dokazuje, že systém obsahuje všechny potřebné funkce.

Stupeň hodnocení: 1

h) Komunikační diagramy – Díky komunikačním digramům došlu ke zpřehlednění volání metod v jednotlivých částech systému.

Stupeň hodnocení: 2

i) Sekvenční diagramy – Sekvenční digramy zachytily veškerou interakci mezi jednotlivými objekty a pomohly odhalit další chybu v systému. Chyba odhalená pomocí toho typu diagramu je znázorněná na Obr. 3. 4 - Sekvenční diagram pro část emaily.

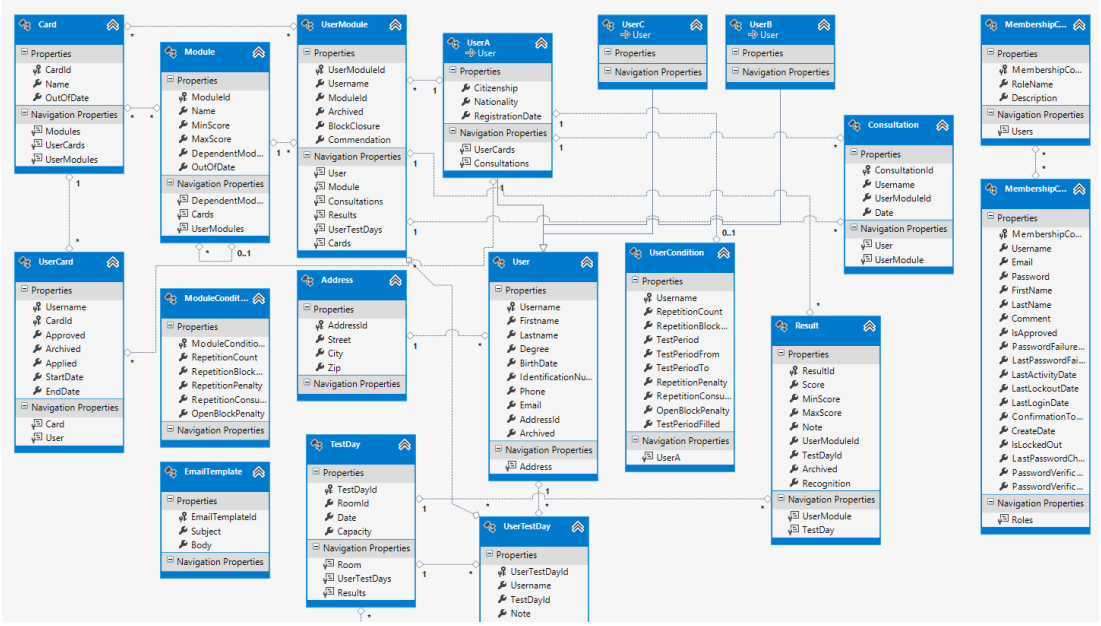
Stupeň hodnocení: 3

Všechny uvedené diagramy byly vytvořeny manuálně ze zdrojového kódu technikou reverzního inženýrství popsanou v kapitole 0. Pro podrobný přehled všech vytvořených diagramů se můžete podívat do přílohy.

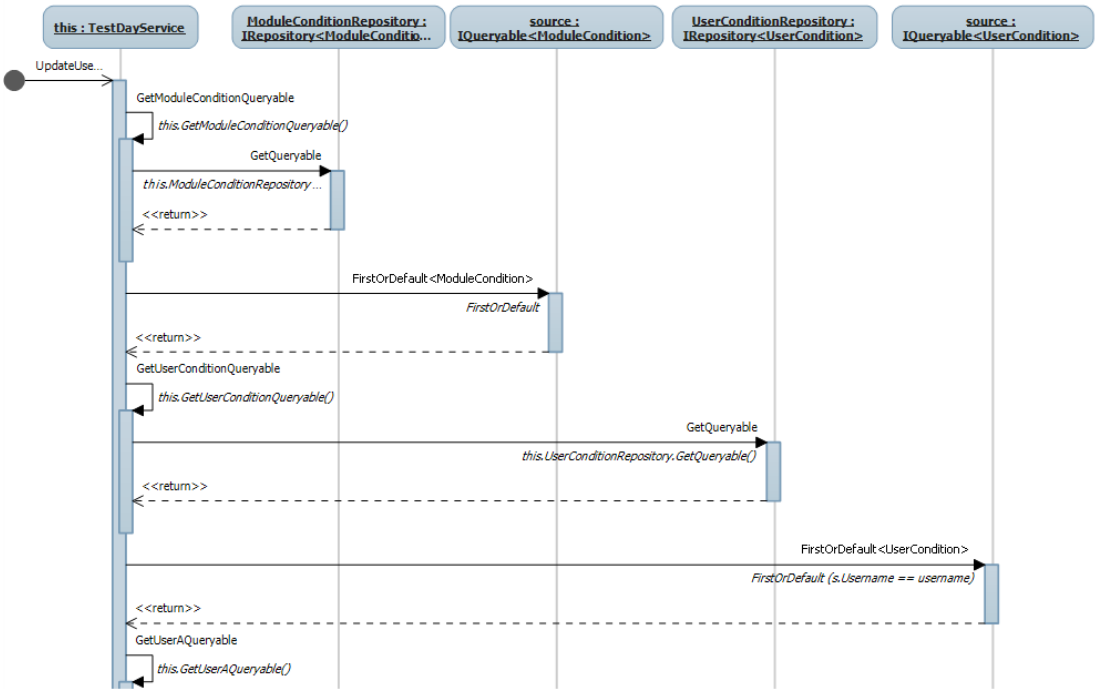
Chytřejší vývojová prostředí jsou navíc schopná vygenerovat některé typy těchto diagramů automaticky přímo ze zdrojového kódu. Mezi takové diagramy například patří:

a) Diagram tříd

b) Databázový diagram



c) Sekvenční diagram



3.4. Detekce problémů

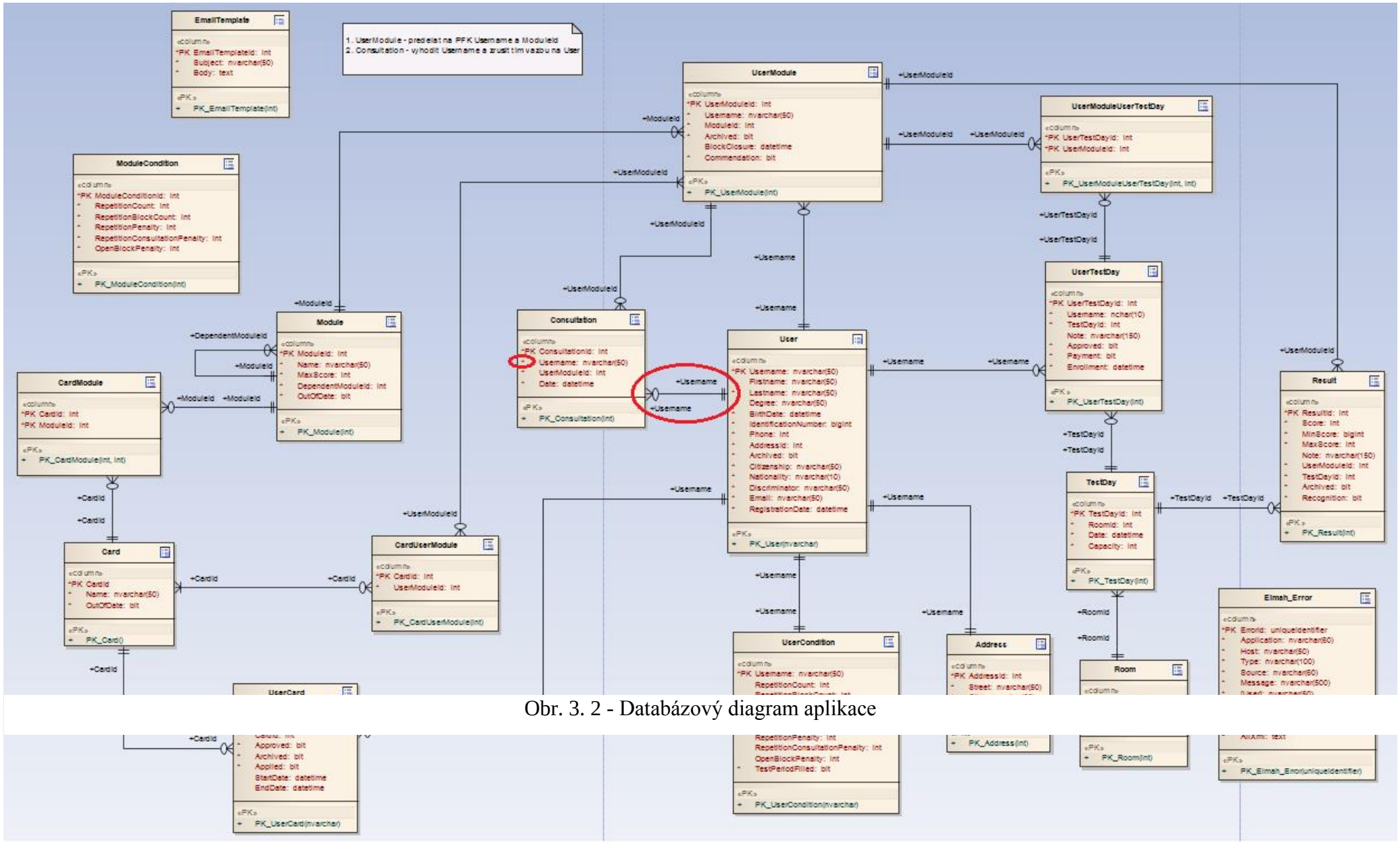
V této části procesu softwarového reinženýrství bylo potřeba odhalit chyby, ke kterým mohlo dojít během vývoje systému.

3.4.1. Cirkulární reference

První chyba byla zjištěna při tvorbě databázového diagramu. Na Obrázku 3.2 je vidět zbytečné spojení tabulky `Consultation` s tabulkou `User`.

K cirkulární referenci v databázi dochází v případě, kdy nevhodně zacházíme s vazbami, tj. s cizími klíči mezi tabulkami. Jedná se o situaci, kdy na sebe vzájemně odkazují dvě nebo více tabulek. Při manipulaci s daty právě z těchto tabulek může pak lehce dojít k tomu, že se jejich obsah stane nekonzistentní.

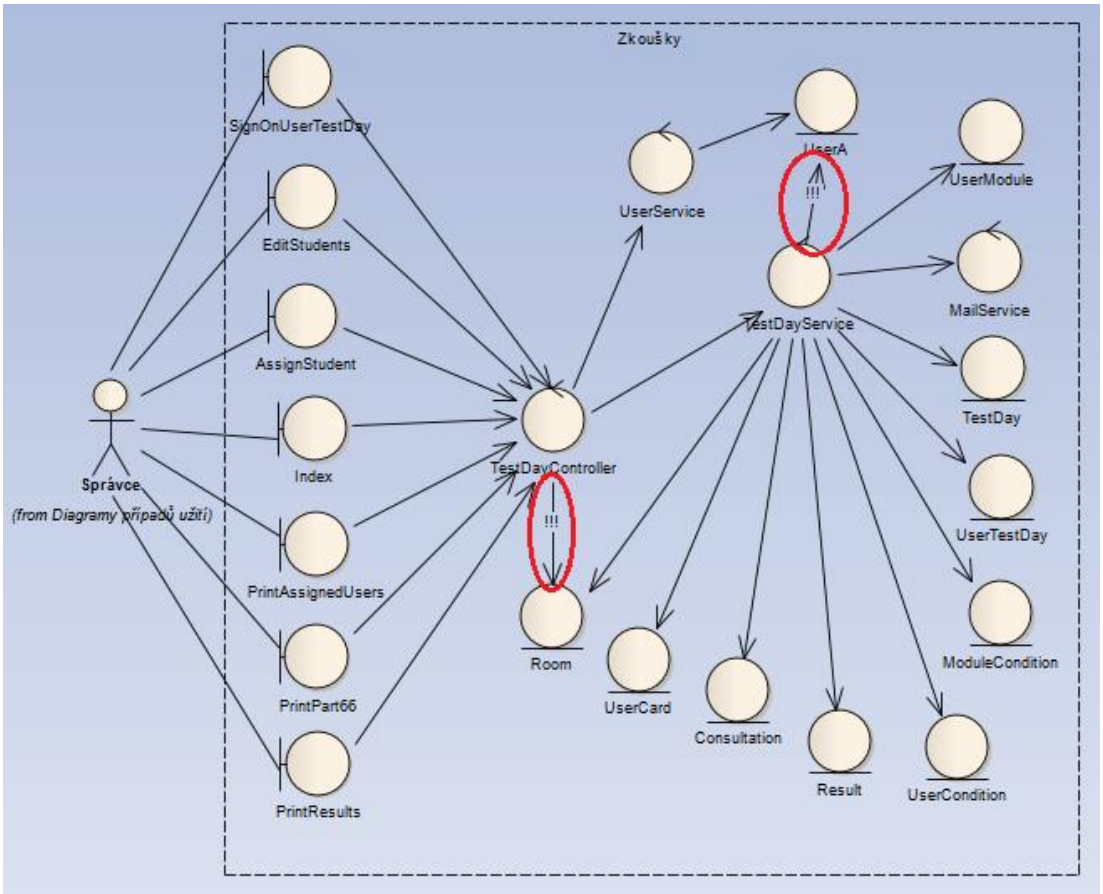
Každý student má po neúspěšném absolvování svého zapsaného modulu právo na konzultaci, díky které se mu sníží penalizovaná doba do opakování. Bohatě tedy stačí, pokud bude tabulka `Consultation` spojená s tabulkou `UserModule`, která sama obsahuje cizí klíč na tabulku `User`.



Obr. 3. 2 - Databázový diagram aplikace

3.4.2. Duplicitní registrace metod

Další chyba byla nalezena při zkoumání vytvořeného diagramu tříd. Na Obrázku 3.3 je znázorněn diagram tříd pro část zkoušky.

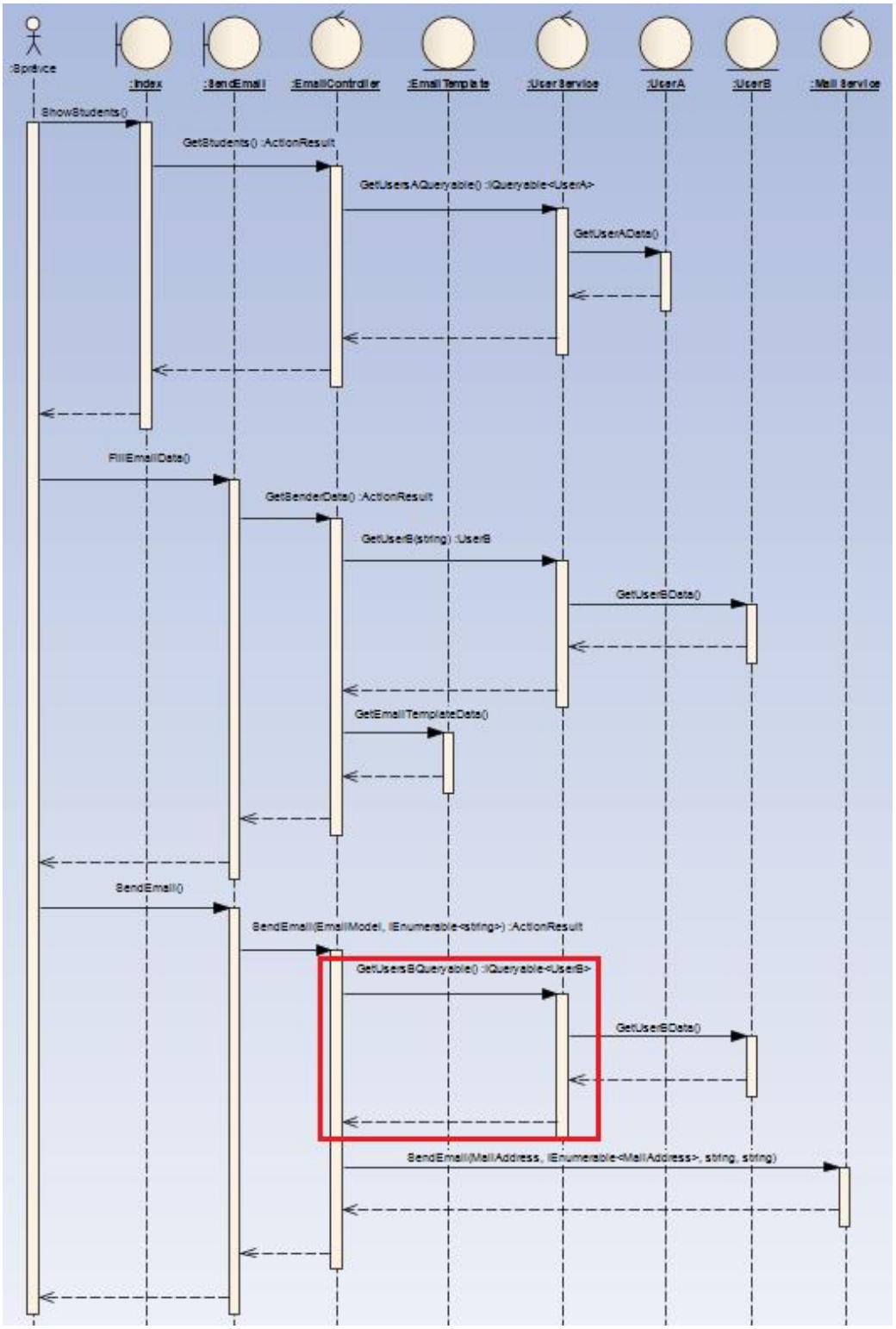


Obr. 3. 3 - Objektový diagram pro část zkoušky

Řídící třída `TestDayController` obsahuje zbytečnou referenci na datovou třídu `Room`, která je součástí služby `TestDayService`. Stejně tak, služba `TestDayService` obsahuje zbytečnou referenci na datovou třídu `UserA`, která je součástí služby `UserService`. Jak lze vidět, bez patřičné dokumentace dochází k dílčím chybám již při vývoji systému, nikoli až za běhu.

3.4.3. Nevhodná skladba metod

Poslední typ chyby, který se podařilo odhalit, bylo zbytečné volání metod datové třídy `UserB`, namísto toho, aby se využilo získaných dat z předchozího volání. Na Obrázku 3.4 je vidět zvýrazněná chyba zachycená v sekvenčním diagramu v části aplikace zvané emaily.



Obr. 3. 4 - Sekvenční diagram pro část emaily

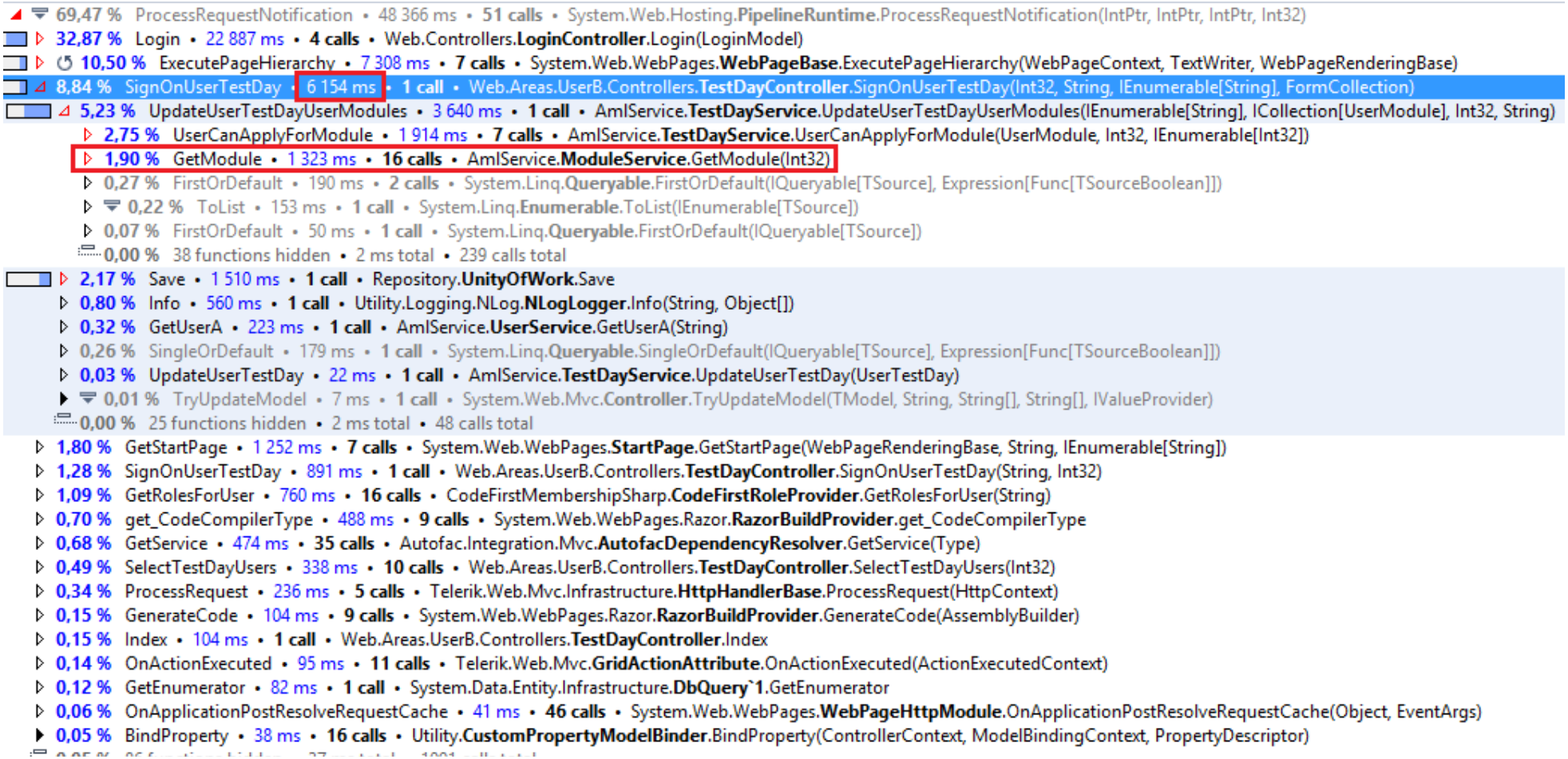
3.4.4. Nevhodné zacházení s SQL

V druhé části detekce chyb bylo třeba se zaměřit na hledání míst, kde aplikace dosahuje slabých výkonů v části zkoušky, kde dochází k přihlašování studenta na zkoušku.

Vhodným řešením se ukázalo použití profilovacích nástrojů, které přesně změří výkon dané části aplikace. Profilovací nástroj po správném nastavení opakovaně provádí požadovanou akci a měří čas jednotlivých úseků kódu.

Pomocí této metody se mi podařilo odhalit kritický úsek problematické části systému. Metoda `UpdateUserTestDayModule` zpracovává přihlášení studenta na zkoušku, konkrétně přihlášení jednotlivých jeho modulů. Pro každý tento modul je v rámci jeho přihlášení prováděna série kontrol z mnoha na něj kladených omezujících podmínek. V rámci jedné z této kontrol byl nalezen nevhodný SQL dotaz, který způsoboval velkou časovou prodlevu.

Na Obrázku 3.5 můžete vidět nalezený kritický úsek při přihlašování studenta na zkoušku.



Obr. 3. 5 - Kritický úsek - profilování

Problém tedy spočíval v nevhodném zacházení s SQL dotazem a následnou manipulací s daty. Výsledná kolekce dotazu nebyla zkrácená o podmínku, která podstatnou měrou zvětšovala její velikost. Dále k dotazu nebyla připojena potřebná tabulka a při každém procházení výsledné kolekce se tak zbytečně volal jeden SQL dotaz do databáze, aby vrátil požadovaný záznam z řádků nepřipojené tabulky.

Postiženou část neefektivního kódu můžete vidět na Obrázku 3.6.

```
var notFinishedUserModules = this.GetUserModuleQueryable()
    .Include(s => s.Results).Where(s => s.Username == username).ToList();
foreach (var notFinishedUserModule in notFinishedUserModules)
{
    var module = this.ModuleService.GetModule(notFinishedUserModule.ModuleId);
    notFinishedUserModule.Module = module;
    if (currentSelectedModuleIds.Contains(notFinishedUserModule.Module.ModuleId))
    {
        // modul se zapisuje
        if (!originalSelectedUserModuleIds.Contains(notFinishedUserModule.Module.ModuleId))
        {
            // kontrola, zda je mozne zapsat modul na zkousku
            if (this.UserCanApplyForModule(notFinishedUserModule, testDayId, currentSelectedModuleIds))
            {
                userModules.Add(notFinishedUserModule);
            }
            else
            {
                successApplication = false;
            }
        }
    }
    else
    {
        // modul se odepisuje
        if (originalSelectedUserModuleIds.Contains(notFinishedUserModule.Module.ModuleId))
        {
            if (!this.ModuleHasDependentModule(notFinishedUserModule, currentSelectedModuleIds, false, removedUserModuleIds))
            {
                successApplication = false;
            }
            else
            {
                var result = this.ResultRepository.GetQueryable()
                    .SingleOrDefault(s => s.TestDayId == testDayId && s.UserModuleId == notFinishedUserModule.UserModuleId);
                if (result != null && result.Score == 0)
                {
                    this.ResultRepository.Delete(result);
                }

                if (result == null || result.Score == 0)
                {
                    userModules.Remove(notFinishedUserModule);
                }
            }
        }
    }
}
```

Obr. 3. 6 - Kritický úsek - kód

3.5. Analýza problémů

Nezbytnou součástí celého procesu softwarového reinženýrství po nalezení problematické části bylo zanalyzovat, proč došlo k daným problémům a jak se jich vyvarovat do budoucna.

3.5.1. Neexistující dokumentace

Problémy spojené s cirkulární referencí byly způsobené četnými změnami požadavků od původního zadání. Vzhledem k tomu, že během vývoje nebyla vytvářena žádná dokumentace, neexistoval téměř žádný opěrný nebo kontrolní bod, který by pomohl tento problém odvrátit.

3.5.2. Špatný pracovní postup

Problém, který jsem pozoroval u sebe nebo u ostatních kolegů a to jak v práci, tak ve škole, je to, že si svou práci nosíme domů. Danou problematikou se tak zabýváme skoro celý den. Dalším problém představuje pracovní postup, nad kterým se nikdo moc nezabývá, protože v něm nevidí smysl.

Většinou tak programátor přijde do práce a s výjimkou oběda pracuje např. 8 hodin bez přestávky. Není se potom čemu divit, když pak během takové práce nevědomě udělá spoustu chyb, které by jinak neudělal.

3.5.3. Nedostatek testovacích dat

Poslední typ problému, se kterým jsem se setkal při softwarovém reinženýrství školního systému techniků a pilotů byl nedostatek dat pro testování.

Po té, co byla aplikace hotová, začalo testování všech funkcionalit, které aplikace nabízí. Všechny drobné problémy a nedostatky, které byly během testování nalezeny, byly odstraněny. Proto pro mě bylo překvapení, když se klient po zhruba dvou měsících ozval, zda by nešlo urychlit metodu pro přihlašování studentů na zkoušku.

Profilovací nástroj skutečně odhalil kritický úsek, jak bylo popsáno v kapitole 3.4.4. Pro mě bylo záhadou, proč jsem tento nedostatek neodhalil už během testování. Celým problém spočíval v tom, že jsem měl k dispozici velmi málo testovacích dat, na kterých se neefektivní kód pracující s SQL dotazem neprojevil.

3.6. Návrh řešení

Po nalezení všech nedostatků aplikace tedy bylo potřeba navrhnout efektivní řešení, které vyřeší dané problém v rámci celé aplikace a ne jen pro daný kritický úsek.

V příštím projektu tedy rozhodně nedoporučuji provádět změny existujícího řešení, dokud není k dispozici alespoň částečná dokumentace pokrývající problematickou část.

Během práce je třeba stanovit a dodržovat osvědčený pracovní postup, ve kterém budou vymezeny krátké a účelové přestávky. Odpovědí na otázku jak zlepšit kvalitu a efektivitu práce je tedy zvolit nějaký vhodný pracovní postup. Pro mě i mé

kolegy byl velkým přínosem řešící tuto problematiku pracovní postup, který se jmenuje Pomodoro. S touto technikou jsem se seznámil na přednášce Jiřího Knesla, který je považován za jednoho z nejlepších programátorů v České republice.

Po odhalení chyby spojené s neefektivní prací s SQL dotazem bylo potřeba přijít na to, proč tento neefektivní kód nebyl odhalen už během testování. Celý problém spočíval v tom, že jsem měl k dispozici jen velmi málo testovacích dat, na kterých se tento nedostatek nijak výrazně neprojevil.

3.7. Aplikace změn

Pro databázový diagram byl z tabulky `Consultation` odstraněn cizí klíč `Username` a tím i vazba na tabulku `User`. Pro získání potřebných dat bude tabulka `Consultation` používat spojení s tabulkou `UserModule`, která je sama spojena s tabulkou `User` přes cizí klíč `Username`.

Pro diagram tříd část zkoušky byla odstraněna vazba z řídící třídy `TestDayController` na datovou třídu `Room`. Nyní řídící třída získává potřebná data ze služby `TestDayService`. Stejně tak služba `TestDayService` již nadále neobsahuje přímo vazbu na datovou třídu `UserA`, ale využívá k tomu službu `UserService`.

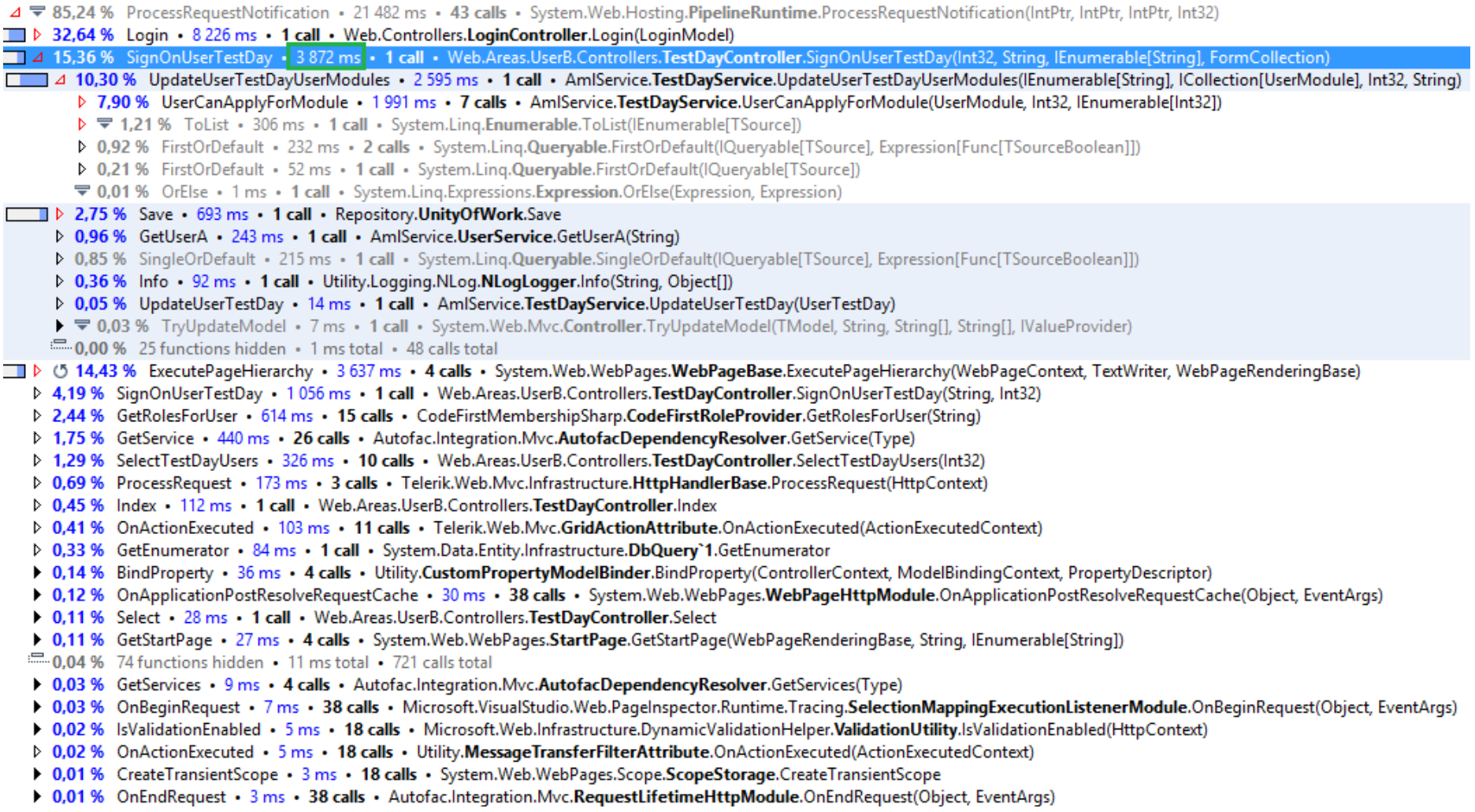
Díky sekvenčnímu diagramu pro část emaily došlo k odhalení zbytečného volání metod datové třídy `UserB`. Toto volání bylo odstraněno a řídící třída `EmailController` využívá těchto dat z předchozího volání, čímž se ušetří jeden SQL dotaz do databáze.

Řešení kritického úseku způsobující časové prodlevy během přihlašování studenta na zkoušku spočívalo v doplnění podmínky pro zkrácení výsledné kolekce určené k procházení a připojení požadované tabulky k SQL dotazu. Na Obrázku 3.7 můžete vidět výsledné řešení a na Obrázku 3.8 výsledný efekt.

```
var notFinishedUserModules = this.GetUserModuleQueryable()
    .Include(m => m.Module)
    .Include(s => s.Results)
    .Where(s => s.Username == username && (!s.Results.Any() || s.Results.All(t => t.Score < t.MinScore))).ToList();
foreach (var notFinishedUserModule in notFinishedUserModules)
{
    if (currentSelectedModuleIds.Contains(notFinishedUserModule.ModuleId))
    {
        // modul se zapisuje
        if (!originalSelectedUserModuleIds.Contains(notFinishedUserModule.ModuleId))
        {
            // kontrola, zda je mozne zapsat modul na zkousku
            if (this.UserCanApplyForModule(notFinishedUserModule, testDayId, currentSelectedModuleIds))
            {
                userModules.Add(notFinishedUserModule);
            }
            else
            {
                successApplication = false;
            }
        }
    }
    else
    {
        // modul se odepisuje
        if (originalSelectedUserModuleIds.Contains(notFinishedUserModule.ModuleId))
        {
            if (!this.ModuleHasDependentModule(notFinishedUserModule, currentSelectedModuleIds, false, removedUserModuleIds))
            {
                successApplication = false;
            }
            else
            {
                var result = this.ResultRepository.GetQueryable()
                    .SingleOrDefault(s => s.TestDayId == testDayId && s.UserModuleId == notFinishedUserModule.UserModuleId);
                if (result != null && result.Score == 0)
                {
                    this.ResultRepository.Delete(result);
                }

                if (result == null || result.Score == 0)
                {
                    userModules.Remove(notFinishedUserModule);
                }
            }
        }
    }
}
```

Obr. 3. 7 - Kritický usek - oprava - kód



Obr. 3. 8 - Kritický usek - oprava - profilování

3.8. Implementace softwarového systému v letecké dopravě

Vzhledem k velkému rozvoji softwarových systémů v letecké dopravě začátkem 80. let, bylo potřeba nějakého dokumentu pro určení letové způsobilosti. Vzhledem k tomuto faktu byl vytvořen dokument DO-178B, Programové vybavení leteckých palubních systémů a výstroje z pohledu osvědčování jejich způsobilosti, Tento dokument byl později rozšířen o dokument ED-109, specifikující také pozemní softwarové systémy. Podle této normy se pokusím dokázat, že Školní systém techniků a pilotů dle part66 splňuje standardy a ustanovení kladeného na softwarový systém v letecké dopravě v rámci procesu softwarového reinženýrství. Vzhledem k licenčním podmínkám nebyla nejnovější forma normy DO-178C/ED-12C k dispozici.

3.8.1. Cíle

Školní systém techniků a pilotů dle Part66 není softwarový systém přímo implementovaný na palubě letadla, ale díky němu se proškolují technici a piloti, kteří pak své znalosti aplikují v letecké dopravě. Pokud by byl tento softwarový systém chybový, mohl by tak nepřímo zapříčinit leteckou nehodu.

Hlavním úkolem této části tedy bude poskytnout dostatečnou úroveň jistoty v otázce bezpečnosti, která bude v souladu s požadavky v letové způsobilosti.

3.8.2. Poruchové stavy

Kompletní přehled všech kategorií poruchových stavů lze nalézt v příslušných předpisech a poradních materiálech, poradním oběžníku FAA AC 25.1309-1A nebo v poradním materiálu AMJ 25-1309 ve znění případných změn. Uvedené kategorie poruchových vztahů jsou zde uvedeny z těchto poradních materiálů. Poruchové stavy se dělí do těchto kategorií:

a) Katastrofické – Poruchové stavy bránící v pokračování bezpečného letu a přistání.

b) Rizikové/Velmi závažné – Poruchové stavy, které by snížili způsobilost letadla nebo schopnost posádky zvládnout nepříznivé provozní podmínky.

c) Závažné – Poruchové stavy vedoucí ke snížení zásob bezpečnosti nebo funkční způsobilosti nebo k podmínkám zhoršujícím výkonnost posádky.

d) Nezávažné – Poruchové stavy, které by výrazně nesnížili bezpečnost letadla. Např. běžná změna letového plánu

e) Bez vlivu – Poruchové stavy neovlivňující provozní způsobilost letadla nebo schopnost posádky.

3.8.3. Úrovně kritičnosti

Důležitou součástí bezpečnosti softwarových systémů v letecké dopravě jsou definice úrovní kritičnosti. Úrovně kritičnosti jsou založeny na tom, jakou měrou může software přispět k potencionálním poruchovým vztahům. Každá úroveň kritičnosti definuje, jak se mění úroveň úsilí vynaloženého na prokázání souladu s požadavky pro osvědčení způsobilosti s kategorií poruchových vztahů. Rozeznáváme následující úrovně kritičnosti

a) Úroveň A – Software, jehož anomální chování by mělo za následek katastrofický poruchový stav.
b) Úroveň B – Software, jehož anomální chování by mělo za následek rizikový/velmi vážný poruchový stav.
c) Úroveň C – Software, jehož anomální chování by mělo za následek závažný poruchový stav.
d) Úroveň D – Software, jehož anomální chování by mělo za následek nezávažný poruchový stav.
e) Úroveň E – Software, jehož anomální chování by nemělo vliv na provozní způsobilost letadla nebo pracovní zátěž pilotů.

3.8.4. Splnění cílů procesů v závislosti na úrovni kritičnosti softwaru v rámci softwarového reinženýrství

Pro kompletní přehled všech tabulek odkazující na cíle jednotlivých procesů životního cyklu softwaru v závislosti na jejich úrovně kritičnosti a v rámci procesu softwarového reinženýrství se podívejte do **Přílohy A: Splnění cílů procesů v závislosti na úrovni kritičnosti softwaru v rámci softwarového reinženýrství.**

Závěr

Hlavním úkolem mé diplomové práce bylo představení procesu softwarového reinženýrství a ukázat jeho praktické využití na reálném projektu. Nedílnou součástí přestavby každého softwarového systému představuje dokumentace, bez které se programátor postavený před tuto problematiku neobejde.

Proces softwarového reinženýrství byl úspěšně aplikován na školní systém techniků a pilotů dle part66. V rámci dokumentace bylo technikou reverzního inženýrství vytvořeno celkem 260 diagramů chování a struktur poskytujících detailní pohled na systém a vytvářejících dostatečnou úroveň abstrakce pro snazší orientaci v rámci budoucího rozšíření systému. Zároveň došlo k opravám chyb v návrhu systému, které byly odhaleny postupným vytvářením dokumentace softwarového systému.

Dále podle požadavků majitele byla opravena část aplikace, která slouží k přihlášení studenta na zkoušku. Výkon aplikace se v dané části zlepšil o 2,3 sekundy, což představuje 37% zlepšení oproti původnímu návrhu.

V dnešní době, kdy více přibývá systémů, které jsou potřeba upravit nebo předělat, než těch, které jsou zcela nové, je potřeba softwarovému reinženýrství věnovat nemalou pozornost.

V rámci procesu softwarového reinženýrství byla prokázána způsobilost školního systému techniků a pilotů dle part66 v letecké dopravě.

Softwarovému reinženýrství se nevyhne téměř žádný softwarový systém. Lze jej pouze výrazně oddálit použitím moderních, ale zároveň osvědčených návrhových vzorů, metodik, postupů, technik a také přehlednou a dostatečnou dokumentací. Je třeba myslet i na to, že se o softwarový systém bude v budoucnu starat někdo jiný.

Seznam příloh

Použitá literatura

[1] Simon Robinson, K. Scott Allen, Ollie Cornes, Jay Glynn, Zach Greenvoss, Burton Harvey, Christian Nagel, Morgan Skinner, Karli Watson: C# programujeme profesionálně, Computer Press, a.s., Brno, 2003

[2] Bieman, J. a Kang, B.: Cohesion and Reuse in an Object-Oriented System, Proceedings of the ACM Symposium on Software Reusability, April 1995. 3.1.3

[3] Jim Arlow, Ila Neustadt: UML2 a unifikovaný proces vývoje aplikací, Objektově orientovaná analýza a návrh prakticky, Computer Press, a.s., Brno, 2011

[4] Marco Bellinaso: ASP.NET 2.0, Problém, návrh, řešení, Computer Press, a.s. Brno, 2007

[5] Bill Evjen, Scott Hanselman, Farhan Muhammad, Srinivasa Sivakumar, Devin Rader: ASP.NET 2.0 Programujeme profesionálně, Computer Press, a.s., Brno, 2006

[6] Chikofsky, E. a Cross, J.: , II. Reverse engineering and design recovery: A taxonomy, IEEE Software, January 1990. 2.5.1, 3.1.4

[7] Deursen, A. a Klint, P. a Verhoef, C: , Research Issues in the Renovation of Legacy Systems, Springer-Verlag, 1999. 2.1

[8] Demeyer, S. a Ducasse, S. a Nierstrasz, O:, Finding Refactorings via ChangeMetrics, Working paper, April 1999. 3.2.4

[9] Ducasse, S. a Demeyer, S.:, The FAMOOS Object-Oriented Reengineering Handbook, http://www.iam.unibe.ch/_famoos/handbook/, 1999. 1

[10] Galdiera, G. a Basili, V.:, Reusing existing software, Technical report CS-TR-2116, University of Maryland, College park, 1988. 2.5.2

[11] Hitz, M. a Montazeri, B.: , Measure Coupling and Cohesion in Object-Oriented Systems, Proceedings of International Symposium on Applied Corporate Computing (ISAAC'95), October 1995. 3.1.2, 3.1.3

[12] Hitz, M. a Montazeri, B.:, A Measurement Tudory Perspective, IEEE Transactions on Software Engineering vol. 22, April 1996. 3.1.2,3.1.3

[13] Humphrey, W.:, Introduction to the Personal Software Process, SEI Series in Software Engineering. Addison Wesley, 1997. 3

[14] Riel, A.:, Object-Oriented Design Heuristics, Addison-Wesley, May 1996. 3.2.2

[15] Tegarden, T. a Sheetz, S. a Monarchi, D.: , A Software Complexity Model of Object-Ori ented Systems, Decision Support Systems vol. 13,1995. 3.1.4

[16] Consideration in Airborne Systems and Equipment Certification, 1140 Connecticut Avenue, N.W. Suite 1020 Washington, D.C. 20036, 1992

[17] CVIS [online] 2013 [cit. 2013-05-6]. Vývoj přírůstku českého ERP trhu v letech 2005-2011. Dostupné z WWW: <http://www.cvis.cz/hlavni.php?stranka=novinky/clanek.php&id=1312>

PŘÍLOHA A: Splnění cílů procesů v závislosti na úrovni kritičnosti softwaru v rámci softwarového reinženýrství

Legenda:

- Tento cíl má být splněn nezávislými subjekty

X - Tento cíl je splněn

- Splnění tohoto cíle je ponecháno na úsudku žadatele

S - Tento cíl je splněn v rámci softwarového reinženýrství

N - Tento cíl nelze splnit v rámci softwarového reinženýrství

Tabulka 2 - Proces plánování softwaru

	Cíl	Závaznost dle úrovně softwaru	Splněno v rámci softwarového reinženýrství	Poznámka
1	Činnosti vývojových a přidružených procesů jsou definovány.	X	S	Vytvoření modelu systému.
2	Přechodová kritéria, vzájemné vztahy a řazení mezi procesy jsou definovány.	X	S	Vytvoření modelu systému.
3	Prostředí životního cyklu softwaru je definováno.	X	S	Vytvoření modelu systému.
4	Doplňkové úvahy jsou vzaty v potaz.	X	S	Vytvoření modelu systému.
5	Normy pro vývoj softwaru jsou definovány.		N	Software je již vytvořen.
6	Softwarové plány jsou definovány.		N	Software je již vytvořen.
7	Softwarové plány jsou koordinovány.		N	Software je již vytvořen.

Tabulka 3 - Vývojové procesy softwaru

	Cíl	Závaznost dle úrovně softwaru	Splněno v rámci softwarového reinženýrství	Poznámka
1	Vysokoúrovňové požadavky jsou vypracovány.	X	S	Vytvoření diagramů aktivit.
2	Odvozené vysokoúrovňové požadavky jsou definovány.	X	S	Vytvoření diagramů aktivit.
3	Architektura softwaru je vytvořena.	X	S	Vytvoření diagramů analytických tříd.
4	Nízkoúrovňové požadavky jsou vypracovány.	X	S	Vytvoření diagramů požadavků.
5	Odvozené nízkoúrovňové požadavky jsou vypracovány.	X	S	Vytvoření diagramů požadavků.
6	Zdrojový kód je vytvořen.	X	S	Přepsání části kódu pro přihlašování studenta na zkoušku.
7	Spustitelný cílový kód je vytvořen a integrován v cílovém počítači.	X	S	Aplikace již běží na předem vybraném hostingu.

Tabulka 4 - Ověření výstupů procesu tvorby požadavků na software

	Cíl	Závaznost dle úrovně softwaru	Splněno v rámci softwarového reinženýrství	Poznámka
1	Vysokoúrovňové požadavky na SW vyhovují požadavkům na systém.	X	S	Vytvoření diagramů sledování požadavků.
2	Vysokoúrovňové požadavky jsou přesně konzistentní.	X	S	Vytvoření diagramů aktivit.
3	Vysokoúrovňové požadavky jsou slučitelné s cílovým počítačem.	X	S	Aplikace již běží na předem vybraném hostingu.
4	Vysokoúrovňové požadavky jsou ověřitelné.	X	S	Vytvoření diagramů sledování požadavků.
5	Vysokoúrovňové požadavky odpovídají normám.		N	Software je již vytvořen.
6	Vysokoúrovňové požadavky jsou trasovatelné k požadavkům na systém.	X	S	Vytvoření diagramů sledování požadavků.
7	Algoritmy jsou přesné.		N	Není součástí softwarového reinženýrství.

Tabulka 5 - Ověření výstupů procesu návrhu softwaru

	Cíl	Závaznost dle úrovně softwaru	Splněno v rámci softwarového reinženýrství	Poznámka
1	Nízkoúrovňové požadavky vyhovují vysokoúrovňovým požadavkům.	X	S	Vytvoření diagramů sledování požadavků.
2	Nízkoúrovňové požadavky jsou přesné a konzistentní.	X	S	Vytvoření diagramů požadavků.
3	Nízkoúrovňové požadavky jsou slučitelné s cílovým počítačem.	X	S	Aplikace již běží na předem vybraném hostingu.
4	Nízkoúrovňové požadavky jsou ověřitelné.	X	S	Vytvoření diagramů sledování požadavků.
5	Nízkoúrovňové požadavky odpovídají normám.		N	Není součástí softwarového reinženýrství.
6	Nízkoúrovňové požadavky jsou trasovatelné k vysokoúrovňovým požadavkům.		S	Vytvoření diagramů sledování požadavků.
7	Algoritmy jsou přesné.		N	Není součástí softwarového reinženýrství.
8	Architektura SW je slučitelná s vysokoúrovňovými požadavky.	X	S	Aplikace již běží na předem vybraném hostingu.
9	Architektura softwaru je konzistentní.		S	Vytvoření modelu systému.
10	Architektura softwaru je slučitelná s cílovým počítačem.	X	S	Aplikace již běží na předem vybraném hostingu.
11	Architektura softwaru je ověřitelná.	X	S	Vytvoření modelu systému podle zdroj. kódu.
12	Architektura softwaru odpovídá normám.		N	Není součástí softwarového reinženýrství.
13	Neporušenost hranic rozkladu je potvrzena.	X	S	Použit objektový přístup.

Tabulka 6 - Ověření výstupů procesů kódování a integrace softwaru

	Cíl	Závaznost dle úrovně softwaru	Splněno v rámci softwarového reinženýrství	Poznámka
1	Zdrojový kód vyhovuje nízkoúrovňovým požadavkům.	X	S	Vytvoření modelu systému.
2	Zdrojový kód vyhovuje architektuře softwaru.	X	S	Vytvoření modelu systému.
3	Zdrojový kód je ověřitelný.	X	S	Vytvoření modelu systému ze zdroj. kódu.
4	Zdrojový kód odpovídá normám.		N	Není součástí softwarového reinženýrství.
5	Zdrojový kód je trasovatelný k nízkoúrovňovým požadavkům.	X	S	Vytvoření diagramů požadavků.
6	Zdrojový kód je přesný a konzistentní.	X	S	Během programování byly použiti nástroje pro úpravu kódu.
7	Výstup procesu integrace SW je kompletní a správný.	X	S	Vytvoření modelu systému.

Tabulka 7 - Zkoušení výstupů integračního procesu

	Cíl	Závaznost dle úrovně softwaru	Splněno v rámci softwarového reinženýrství	Poznámka
1	Spustitelný cílový kód vyhovuje vysokoúrovňovým požadavkům.	X	S	Vytvoření modelu systému.
2	Spustitelný kód je odolný vůči nestandardním vysokoúrovňovým požadavkům.	X	S	Vytvoření modelu systému.
3	Spustitelný cílový kód vyhovuje nízkoúrovňovým požadavkům.	X	S	Vytvoření modelu systému.
4	Spustitelný cílový kód je odolný vůči nestandardním nízkoúrovňovým požadavkům.	X	S	Vytvoření modelu systému.
5	Spustitelný cílový kód je slučitelný s cílovým počítačem.	X	S	Aplikace již běží na předem vybraném hostingu.

Tabulka 8 - Ověřování výsledků procesu ověřování

	Cíl	Závaznost dle úrovně softwaru	Splněno v rámci softwarového reinženýrství	Poznámka
1	Zkušební postupy jsou správné.		N	Není součástí softwarového reinženýrství.
2	Výsledky zkoušek jsou správné a neshody vysvětlitelné.		N	Není součástí softwarového reinženýrství.
3	Pokrytí vysokoúrovňových požadavků zkouškami je dosaženo.		N	Není součástí softwarového reinženýrství.
4	Pokrytí nízkoúrovňových požadavků zkouškami je dosaženo.		N	Není součástí softwarového reinženýrství.
5	Pokrytí struktury softwaru zkouškami (upravené pokrytí podmínek/rozhodnutí) je dosaženo.		N	Není součástí softwarového reinženýrství.
6	Pokrytí struktury softwaru zkouškami (pokrytí rozhodnutí) je dosaženo.		N	Není součástí softwarového reinženýrství.
7	Pokrytí struktury softwaru zkouškami (pokrytí příkazů) je dosaženo.		N	Není součástí softwarového reinženýrství.
8	Pokrytí struktury softwaru zkouškami (vazby v datech a v řízení) je dosaženo.		N	Není součástí softwarového reinženýrství.

Tabulka 9 - Proces řízení konfigurace softwaru

	Cíl	Závaznost dle úrovně softwaru	Splněno v rámci softwarového reinženýrství	Poznámka
1	Konfigurační položky jsou identifikovány	X	S	Vytvoření modul systému.
2	Bázové konfigurace jsou ustanoveny, trasovatelnost je zajištěna.	X	S	Vytvoření modul systému.
3	Hlášení nedostatků, změnové řízení, přezkoumání změn a evidence stavu konfigurace jsou řízeny.	X	S	Použití nástroje YouTrack.
4	Archivování, vyhledávání a vydávání jsou vyřešeny.	X	S	Použití nástroje GIT.
5	Kontrola zavádění softwaru je ustanovena.	X	S	Vytvoření modul systému.
6	Řízení prostředí životního cyklu softwaru je řízena.	X	S	Vytvoření modul systému.

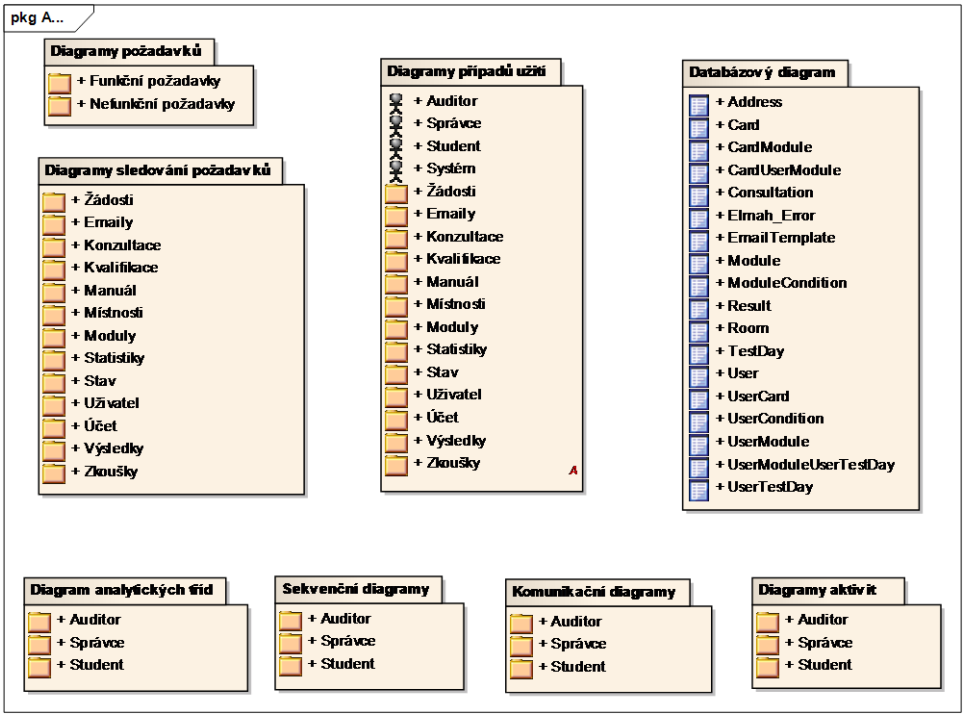
Tabulka 10 - Proces zabezpečování jakosti softwaru

	Cíl	Závaznost dle úrovně softwaru	Splněno v rámci softwarového reinženýrství	Poznámka
1	Je dosaženo jistoty, že vývojové a přidružené procesy vyhovují schváleným softwarovým plánům a normám.	#	N	Není součástí softwarového reinženýrství.
2	Je dosaženo jistoty, že přechodová kritéria pro procesy životního cyklu SW jsou splněna.		N	Není součástí softwarového reinženýrství.
3	Přezkoumání kompletnosti a připravenosti SW je provedeno.	#	N	Není součástí softwarového reinženýrství.

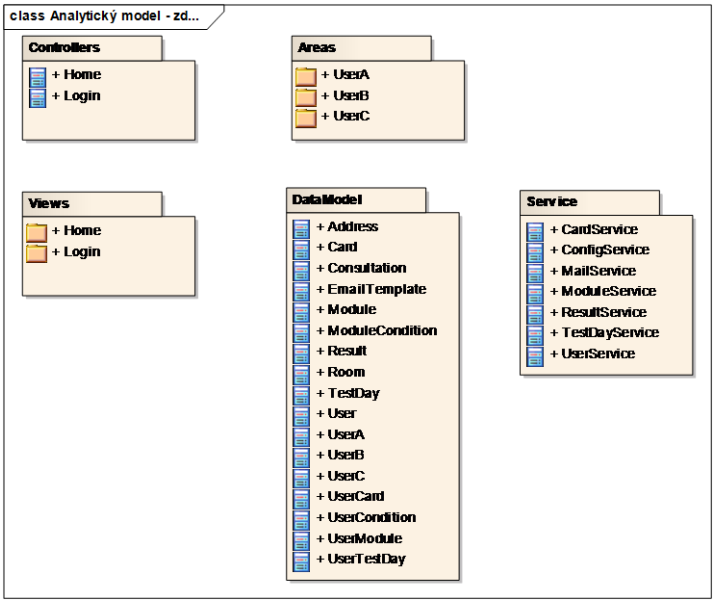
Tabulka 11 - Proces styku s certifikačním orgánem

	Cíl	Závaznost dle úrovně softwaru	Splněno v rámci softwarového reinženýrství	Poznámka
1	Komunikace a součinnosti mezi žadatelem a certifikačním orgánem jsou dohodnuty.		N	Není součástí softwarového reinženýrství.
2	Způsob splnění požadavků je navržen a získán souhlas pro Plán softwarových aspektů osvědčování způsobilosti.		N	Není součástí softwarového reinženýrství.
3	Splnění požadavků je doloženo.	X	S	Vytvoření modelu systému.

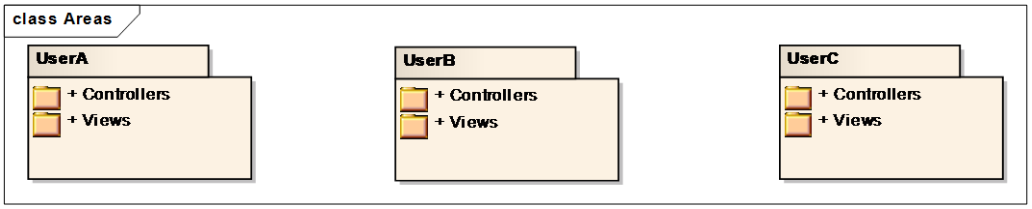
PŘÍLOHA B: Diagram balíčků



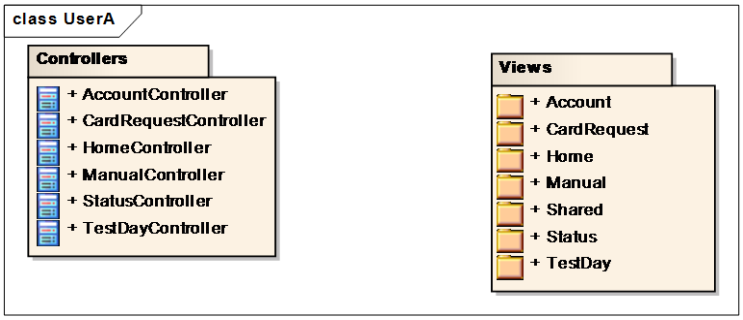
PŘÍLOHA C: Diagramy tříd



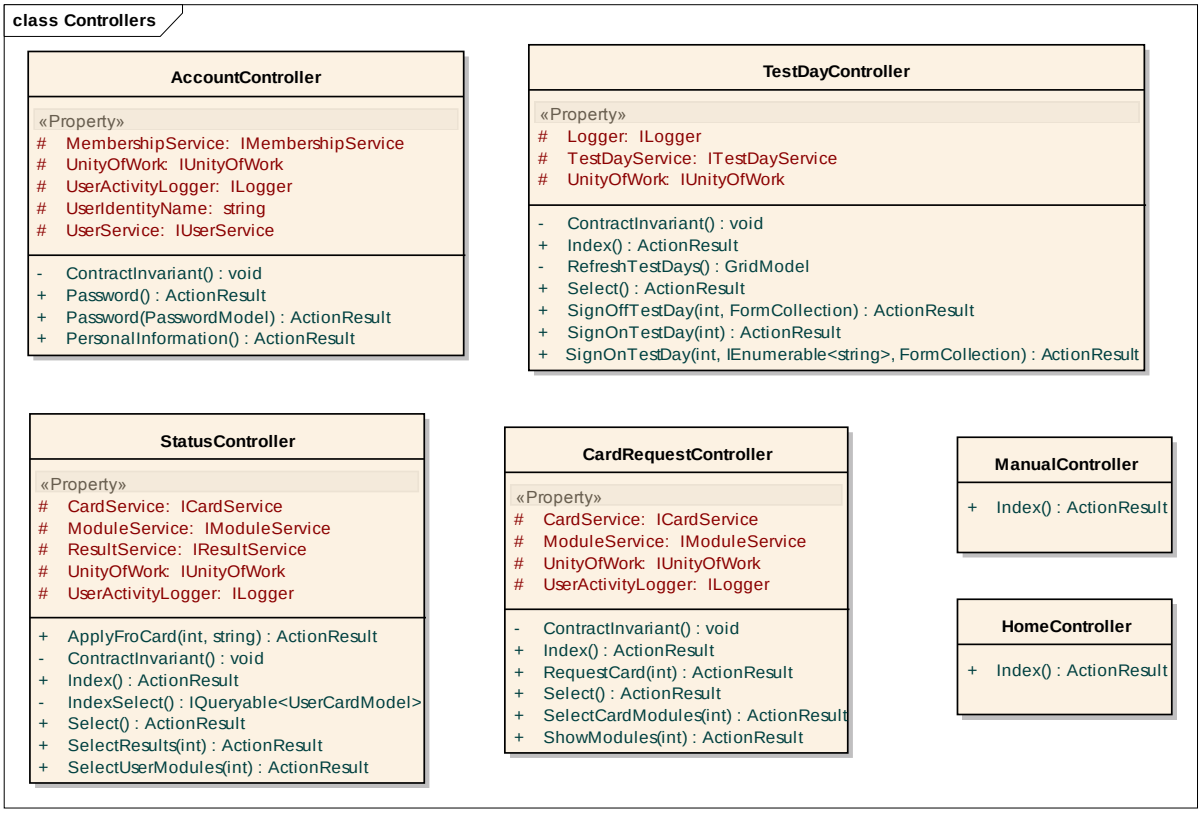
Areas



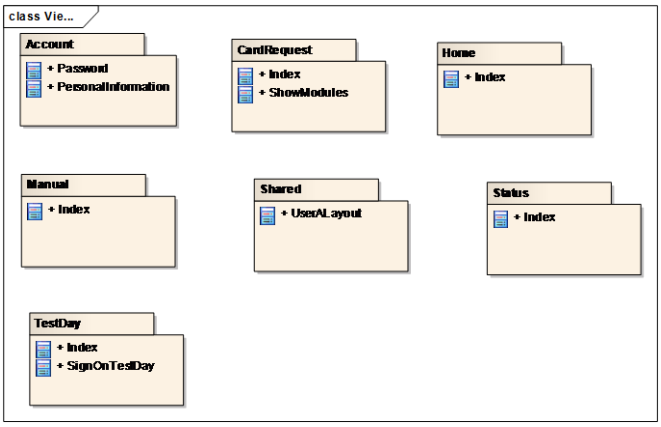
UserA



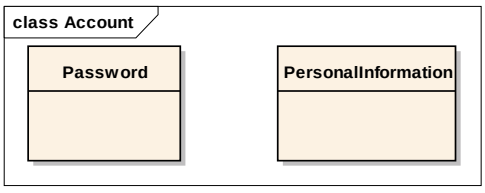
Controllers



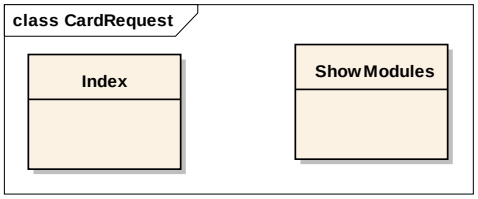
Views



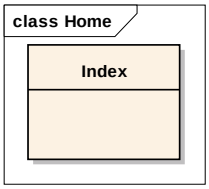
Account



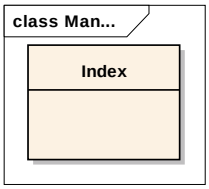
CardRequest



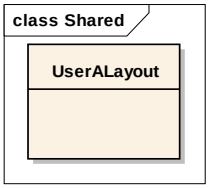
Home



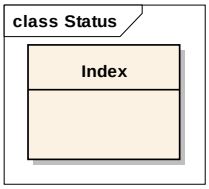
Manual



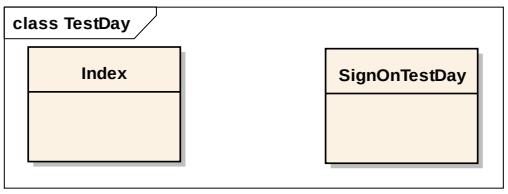
Shared



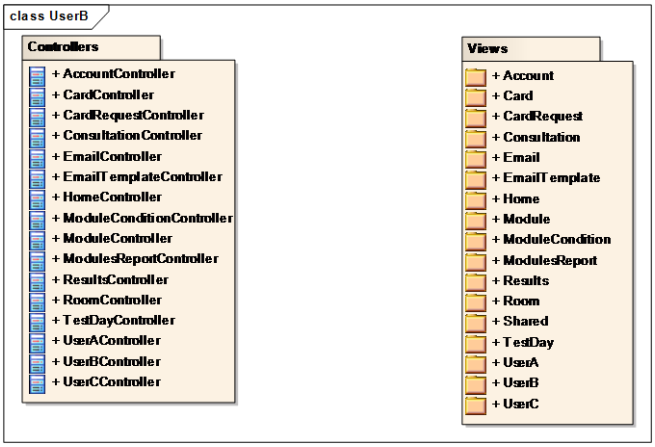
Status



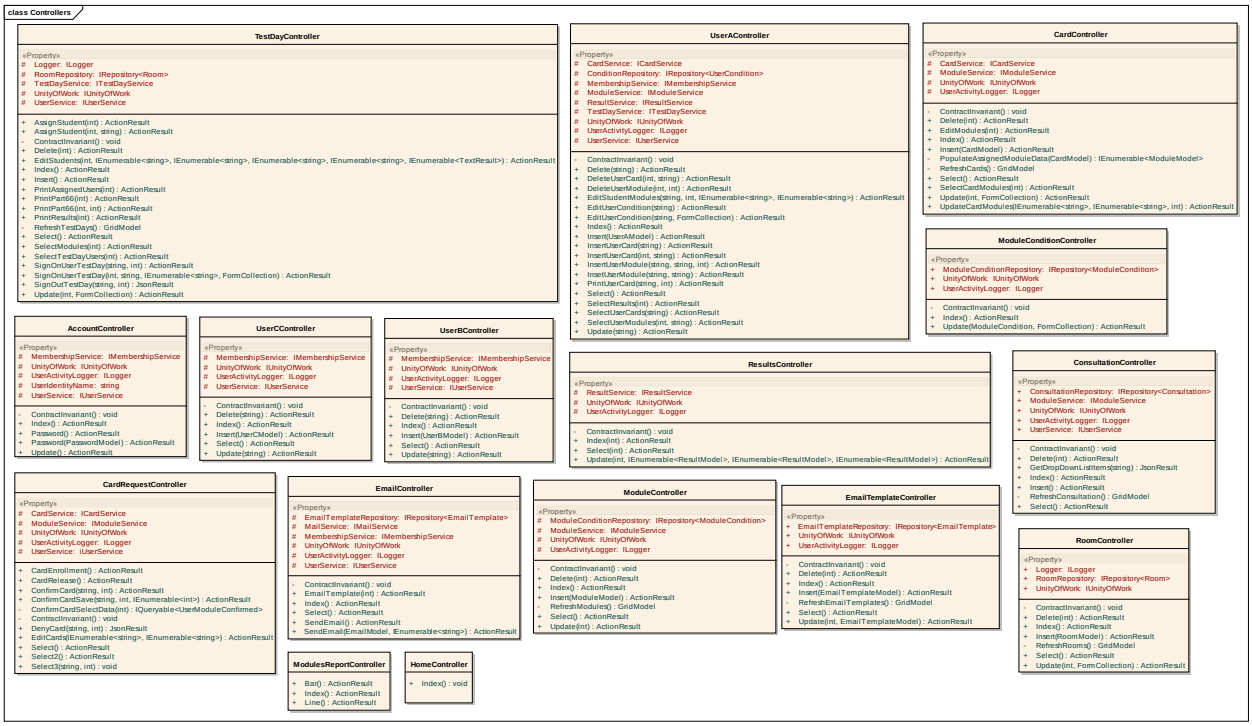
TestDay



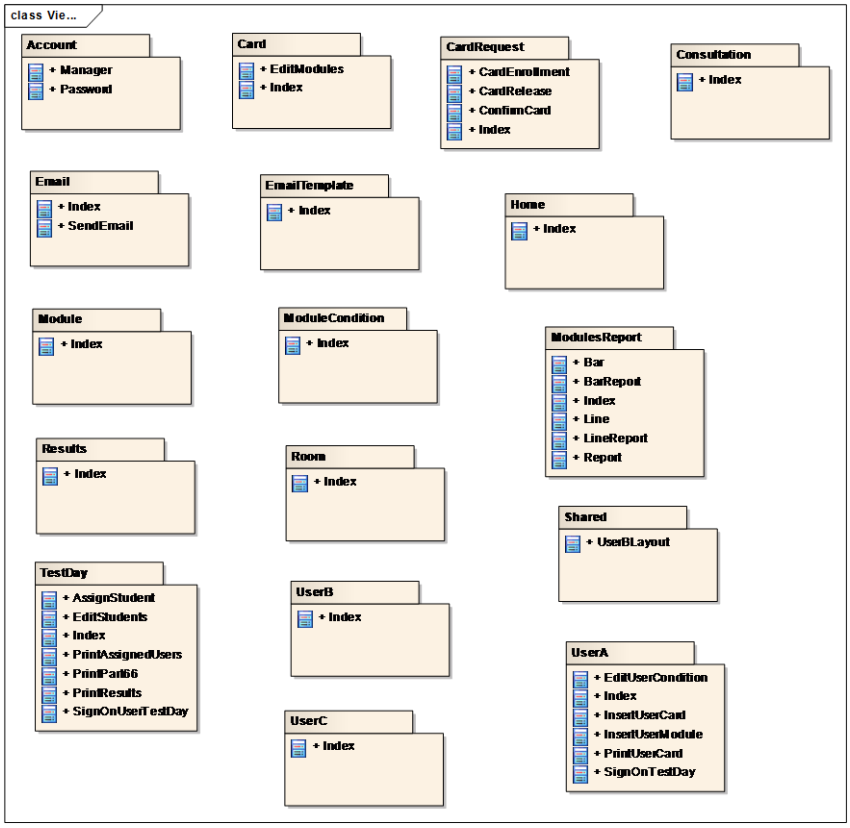
UserB



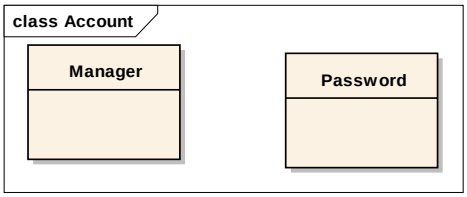
Controllers



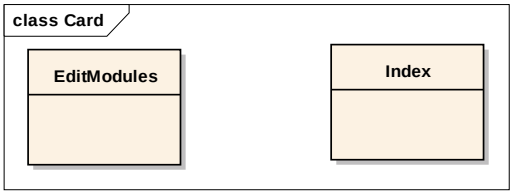
Views



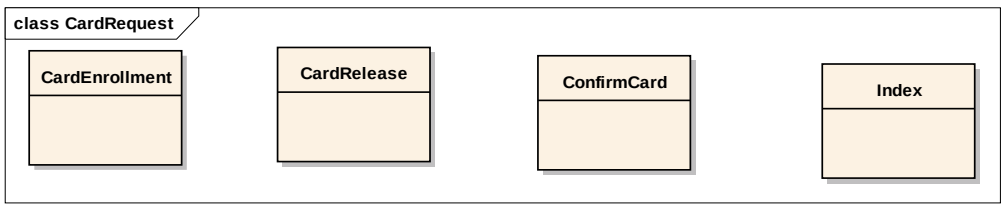
Account



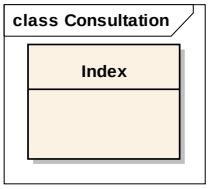
Card



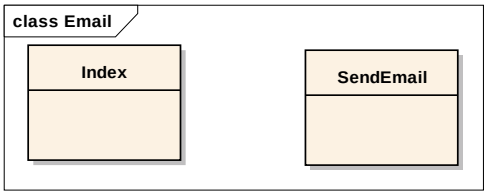
CardRequest



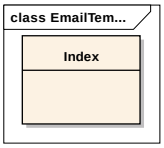
Consultation



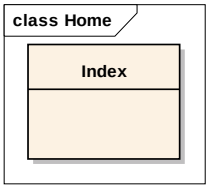
Email



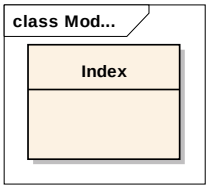
EmailTemplate



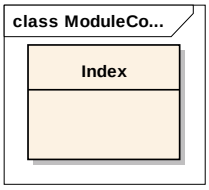
Home



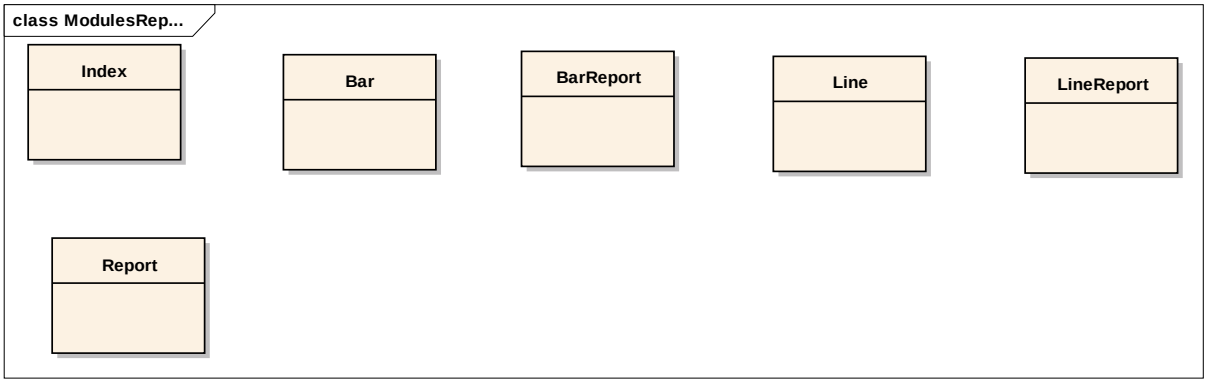
Module



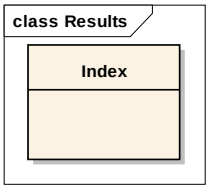
ModuleCondition



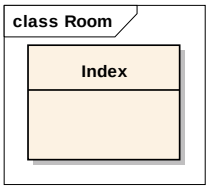
ModulesReport



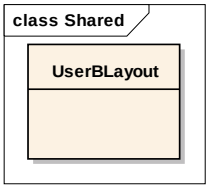
Results



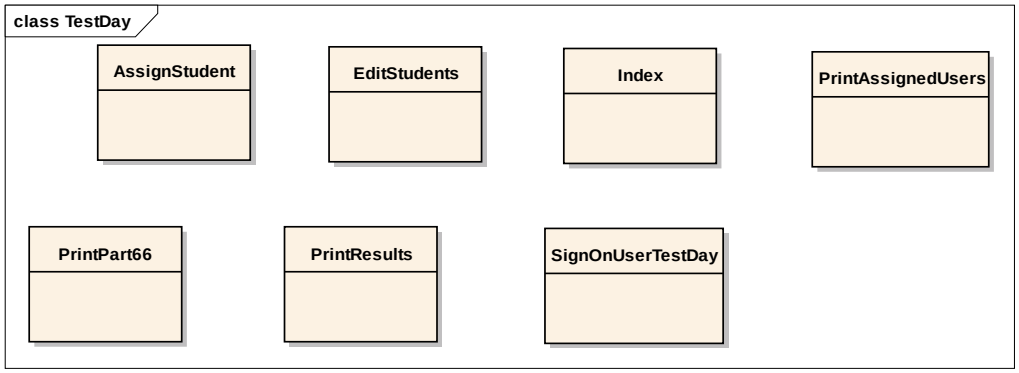
Room



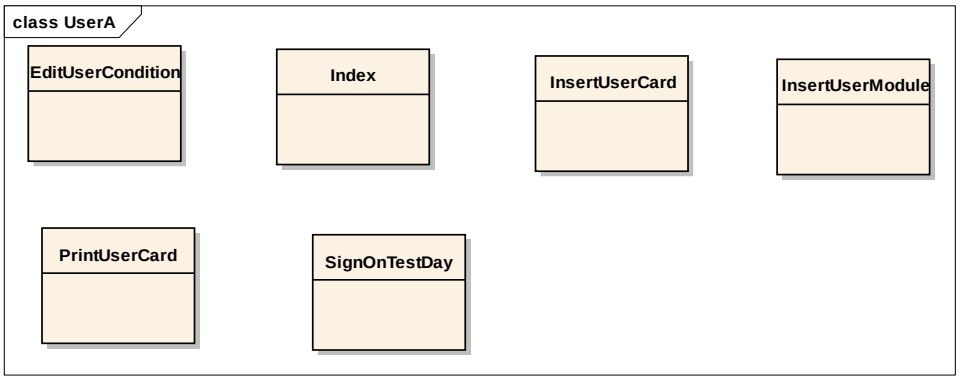
Shared



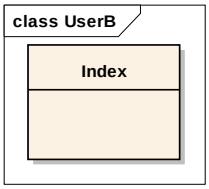
TestDay



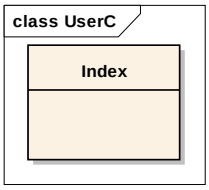
UserA



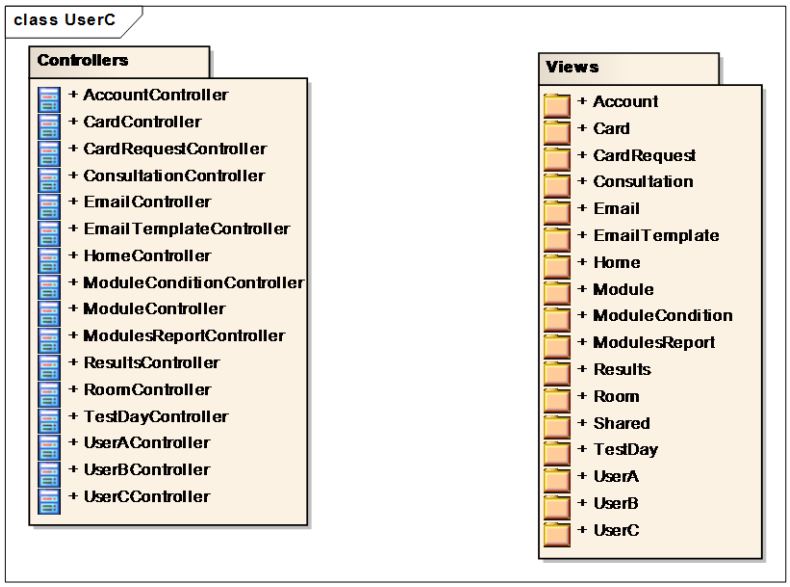
UserB



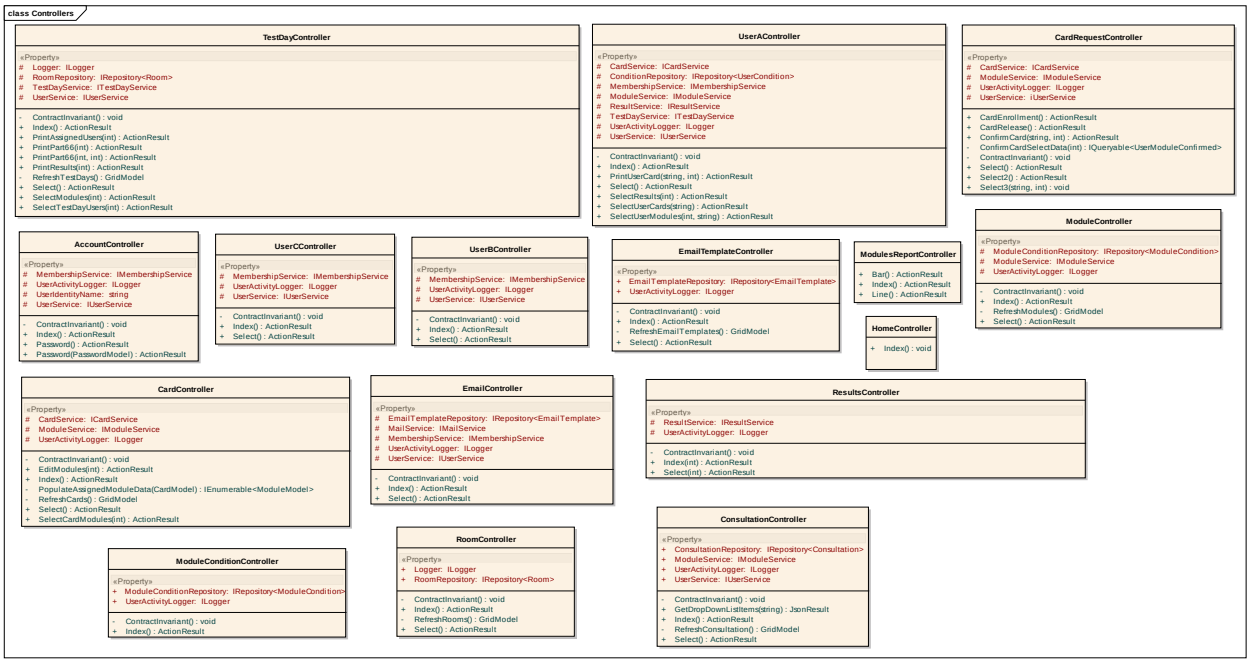
UserC



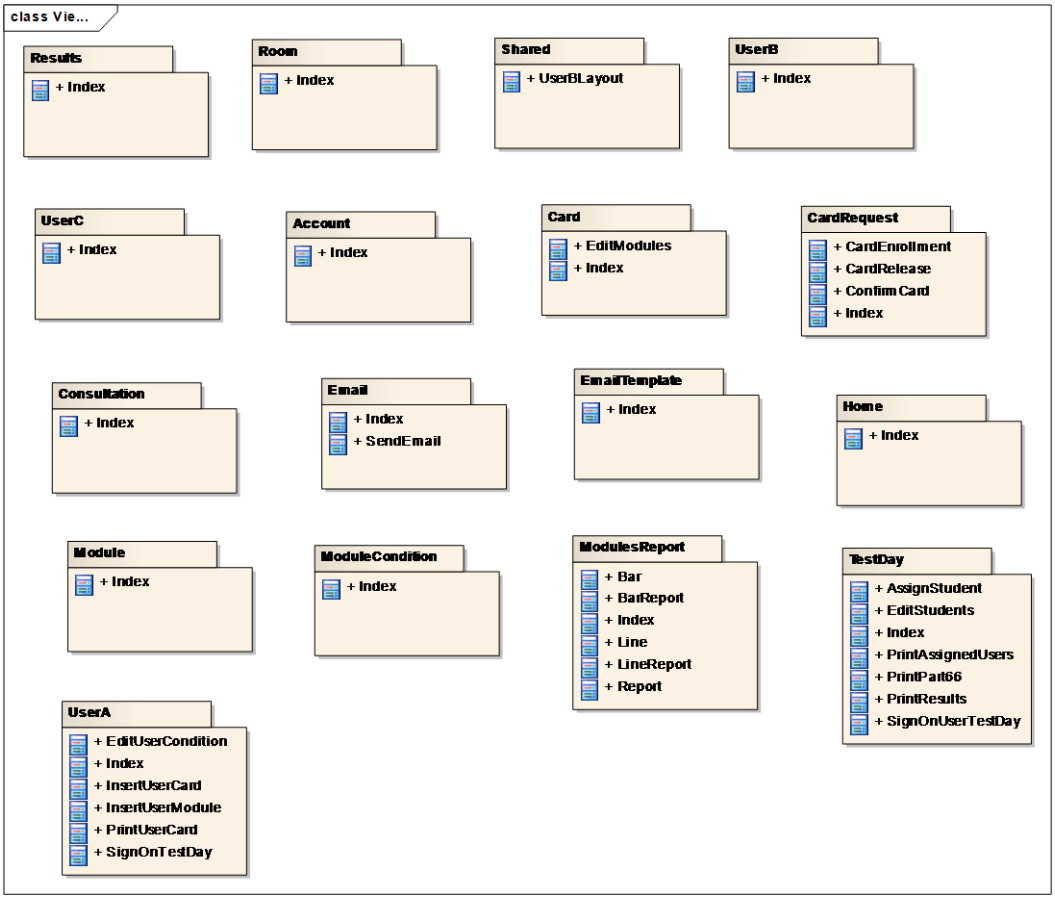
UserC



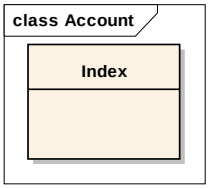
Controllers



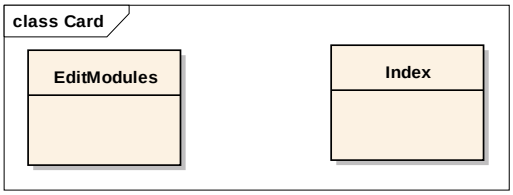
Views



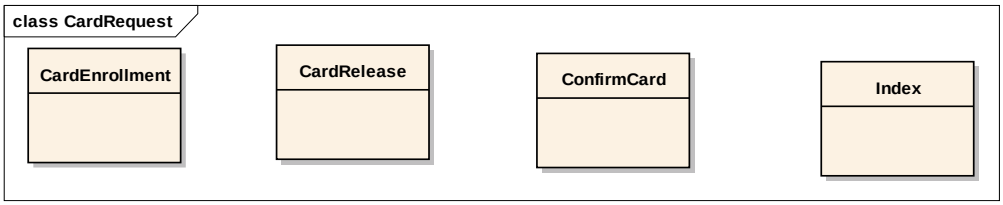
Account



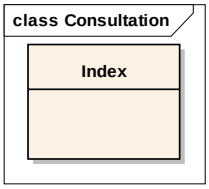
Card



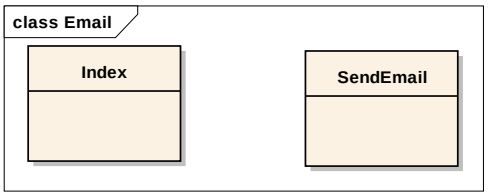
CardRequest



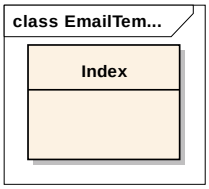
Consultation



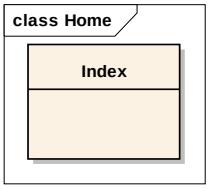
Email



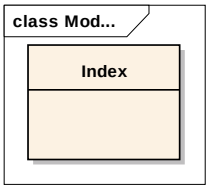
EmailTemplate



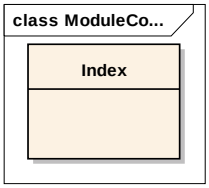
Home



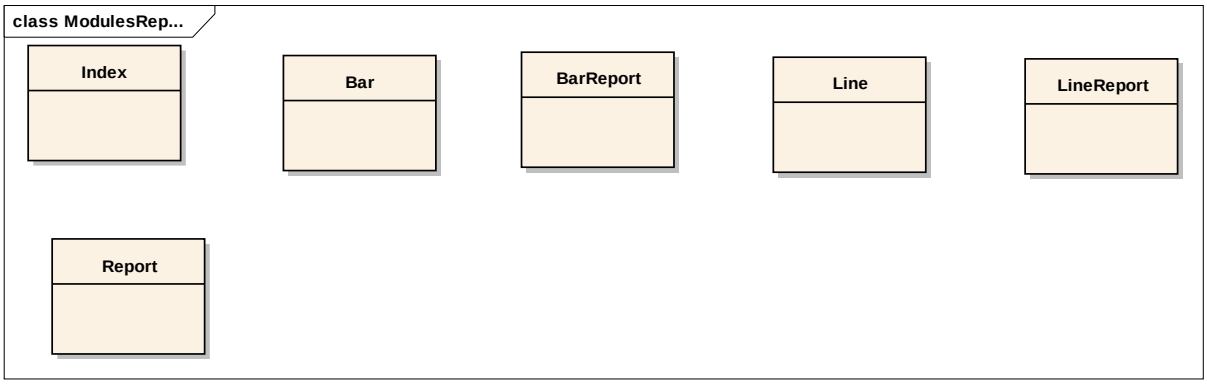
Module



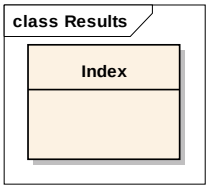
ModuleCondition



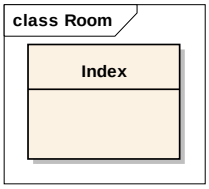
ModulesReport



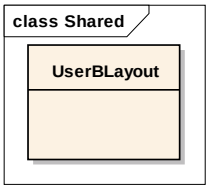
Results



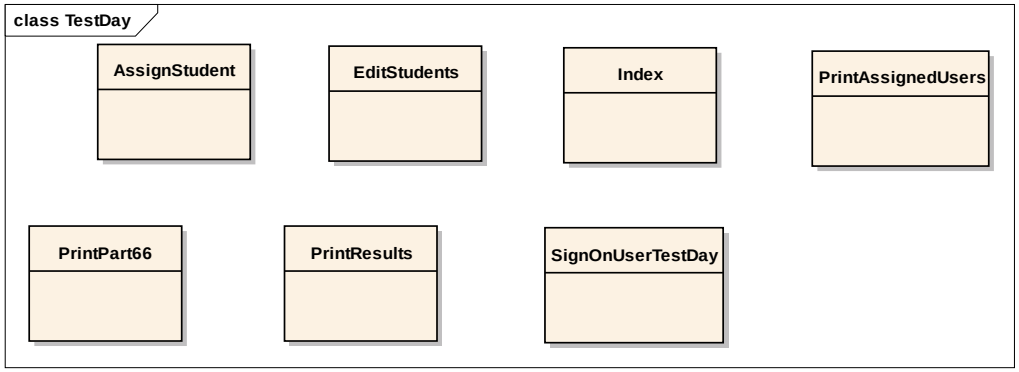
Room



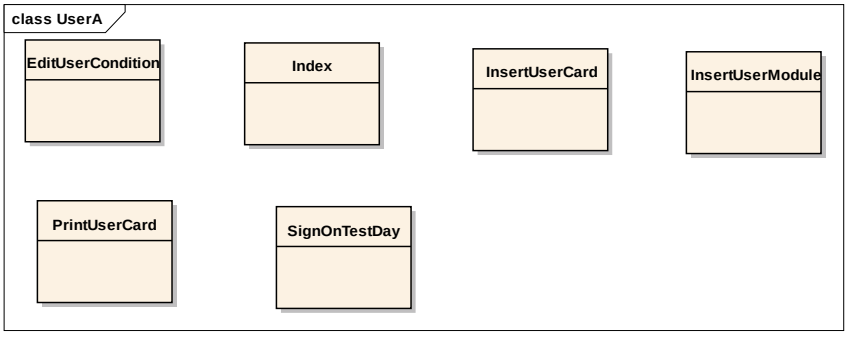
Shared



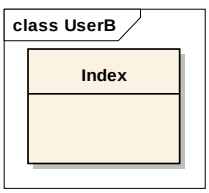
TestDay



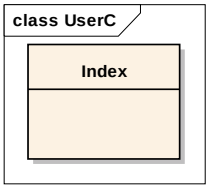
UserA



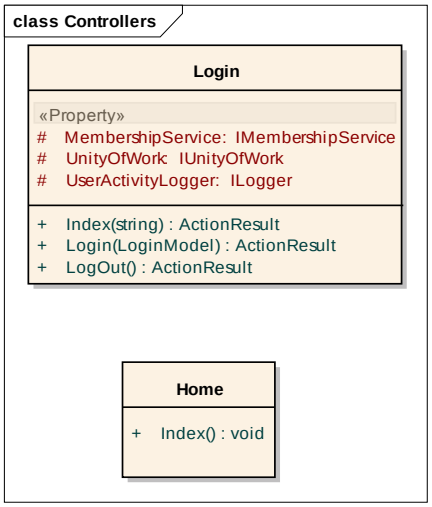
UserB



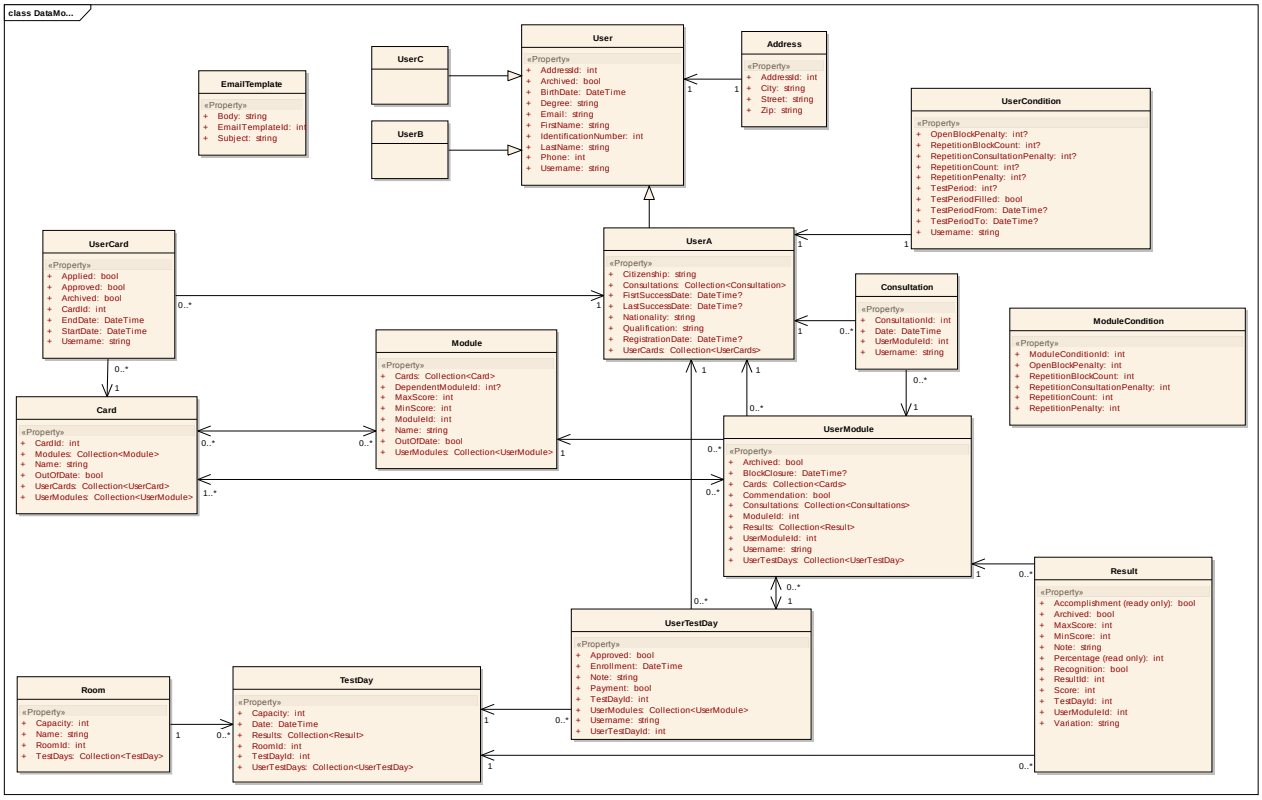
UserC



Controllers

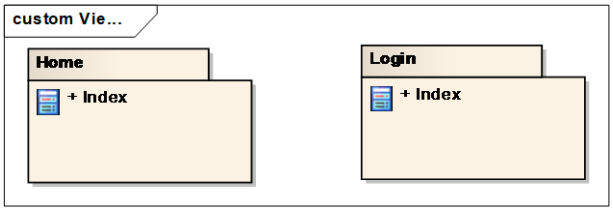


DataModel

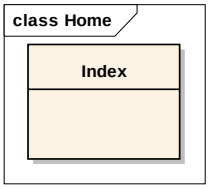


Service

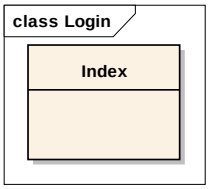
Views



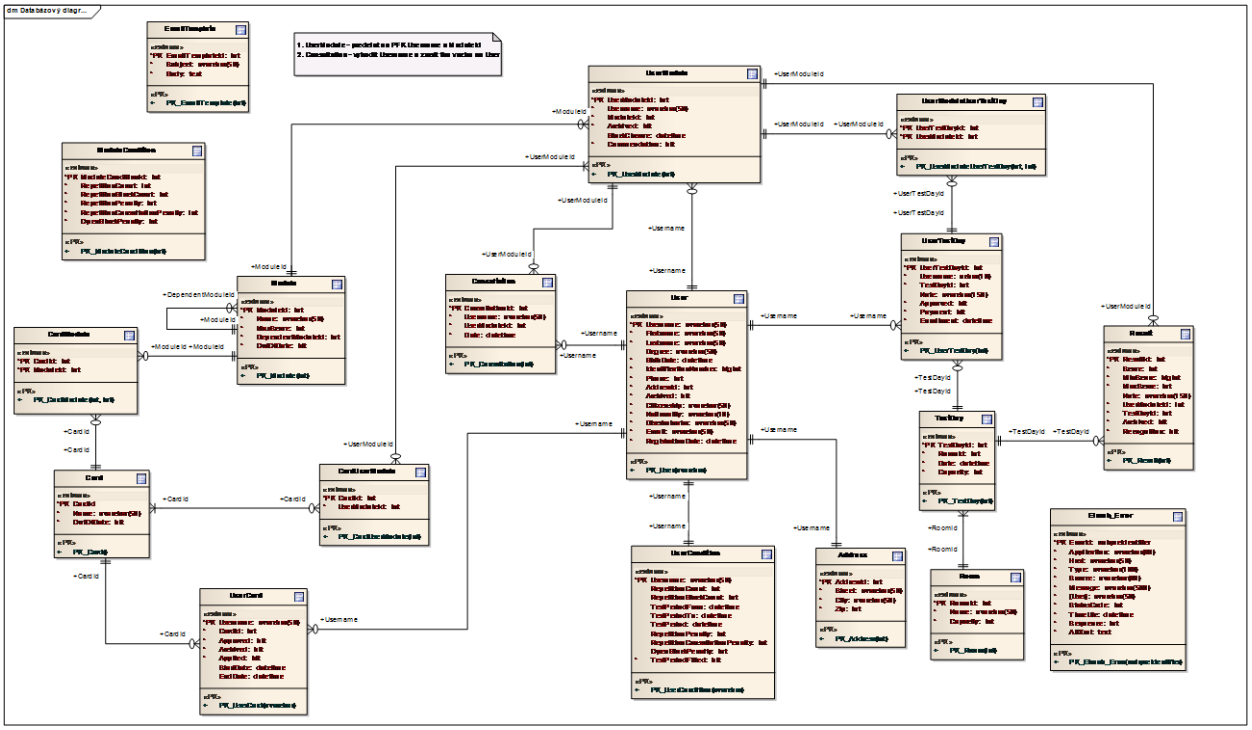
Home



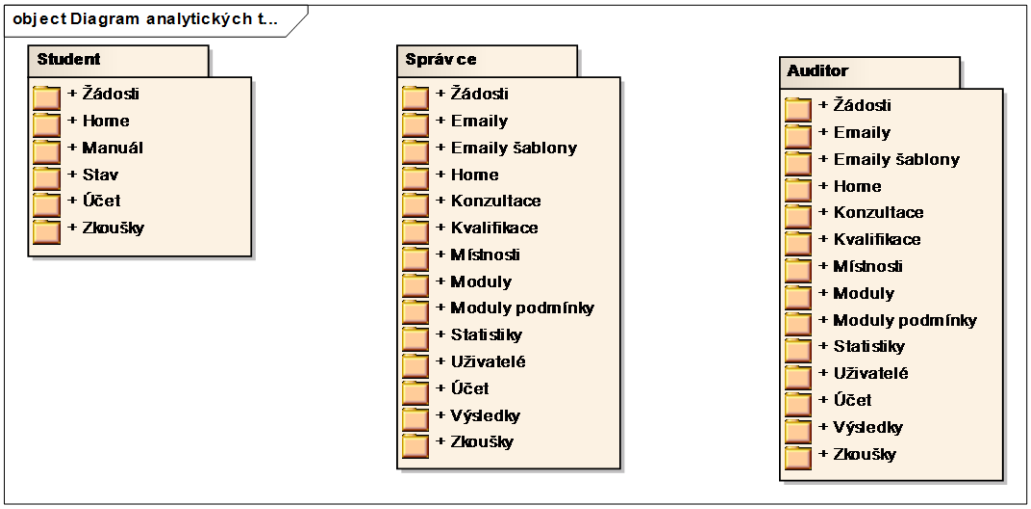
Login



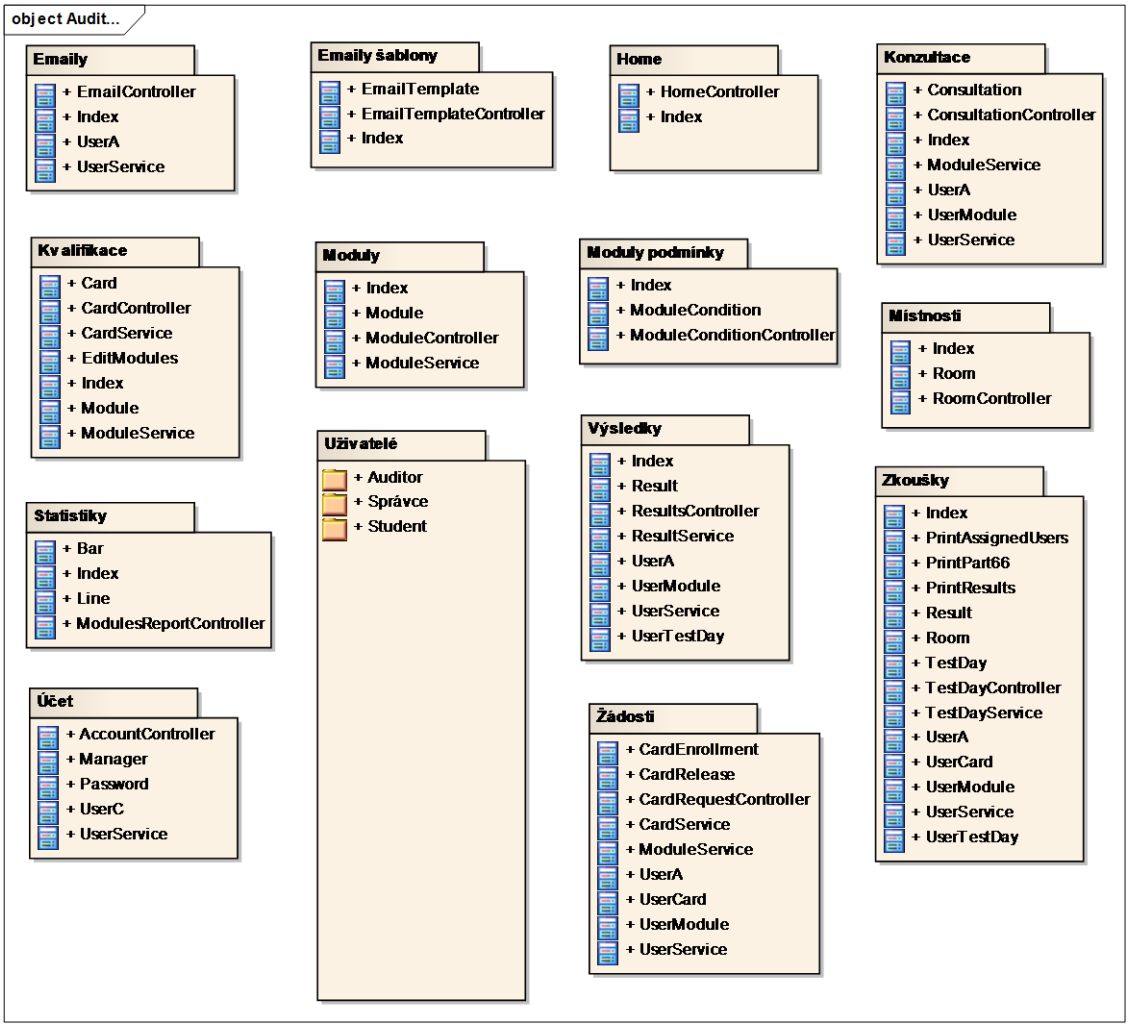
Databázový diagram



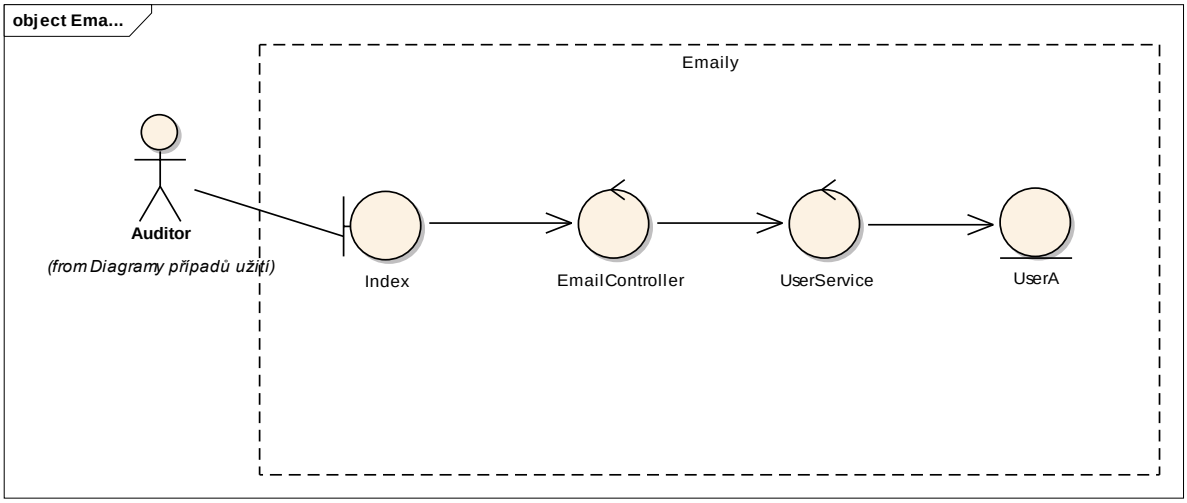
PŘÍLOHA D: Objektové diagramy



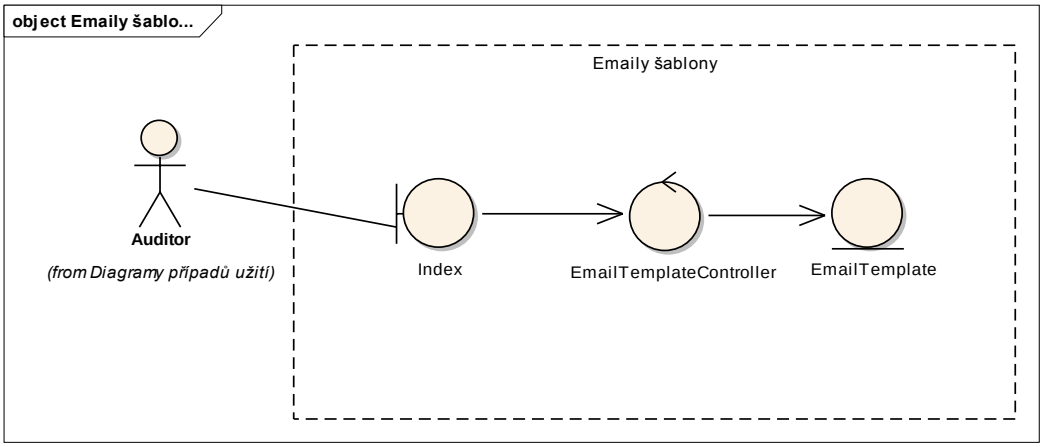
Auditor



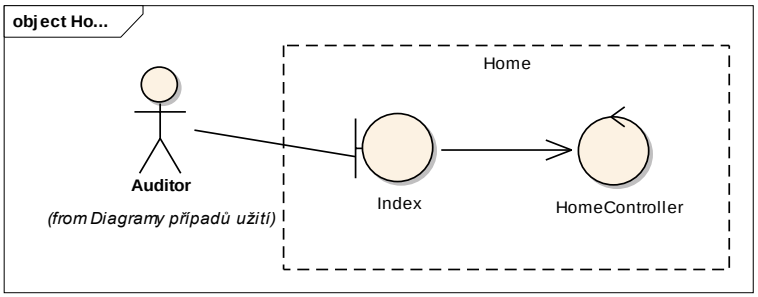
Emaily



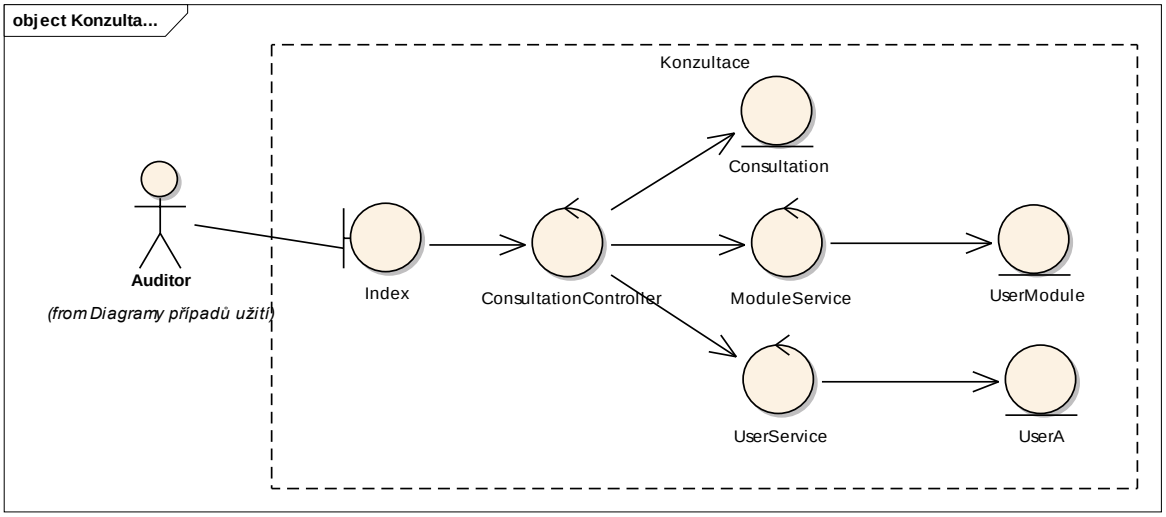
Emaily šablony



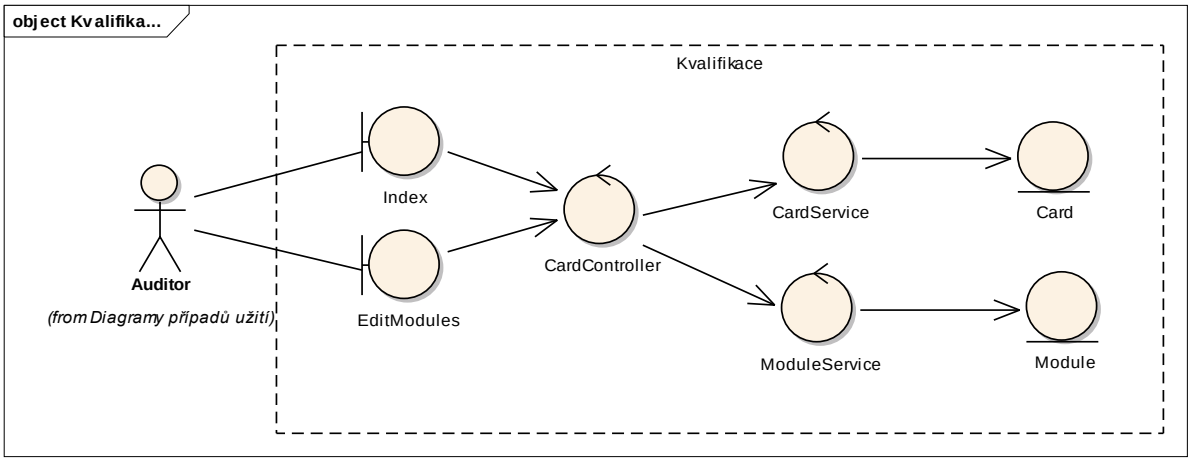
Home



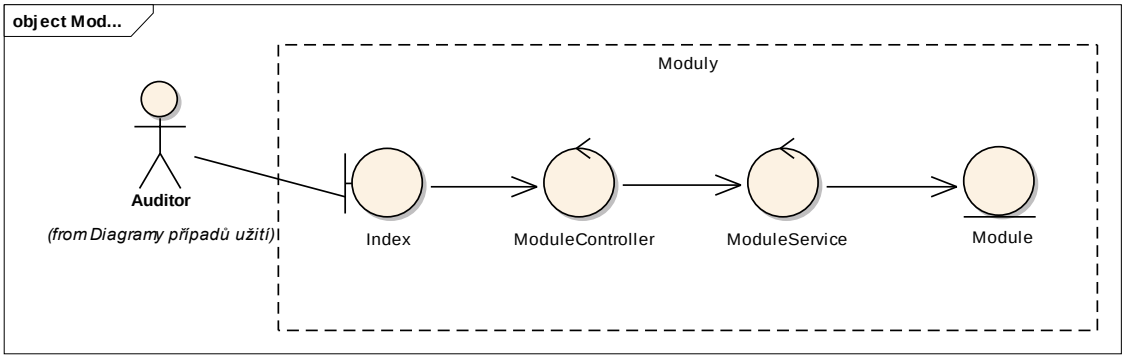
Konzultace



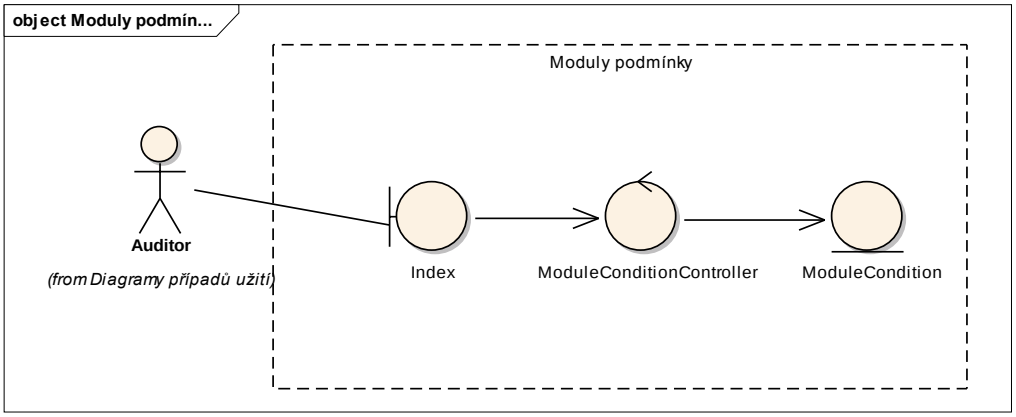
Kvalifikace



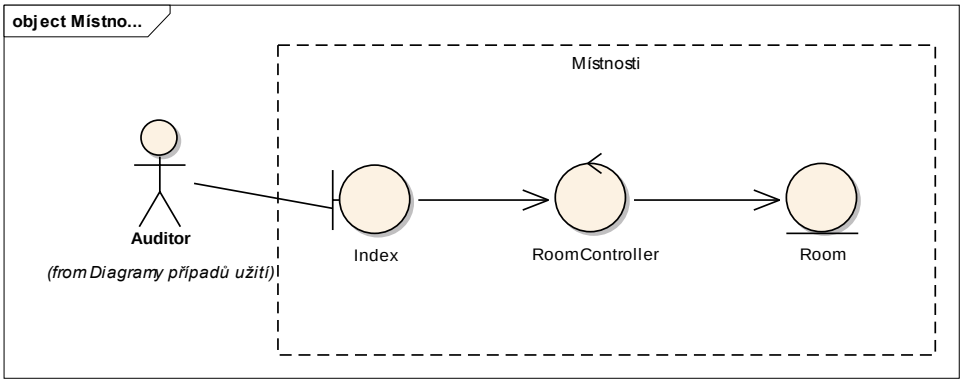
Moduly



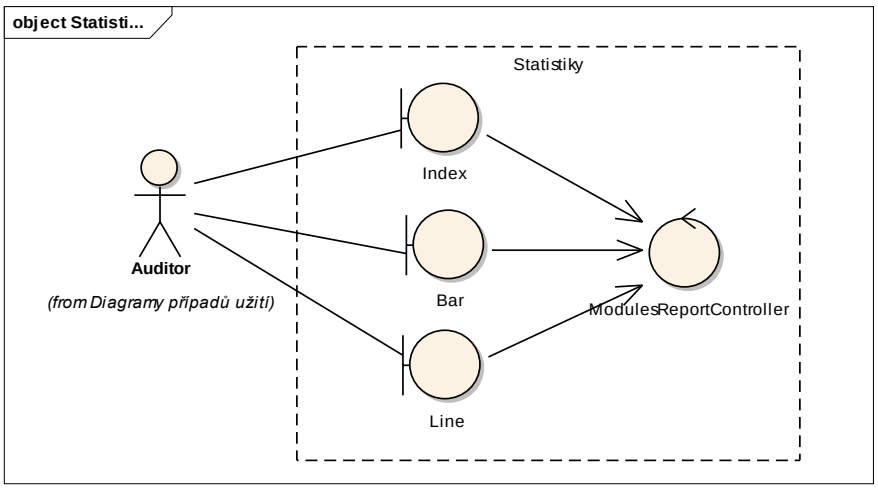
Moduly podmínky



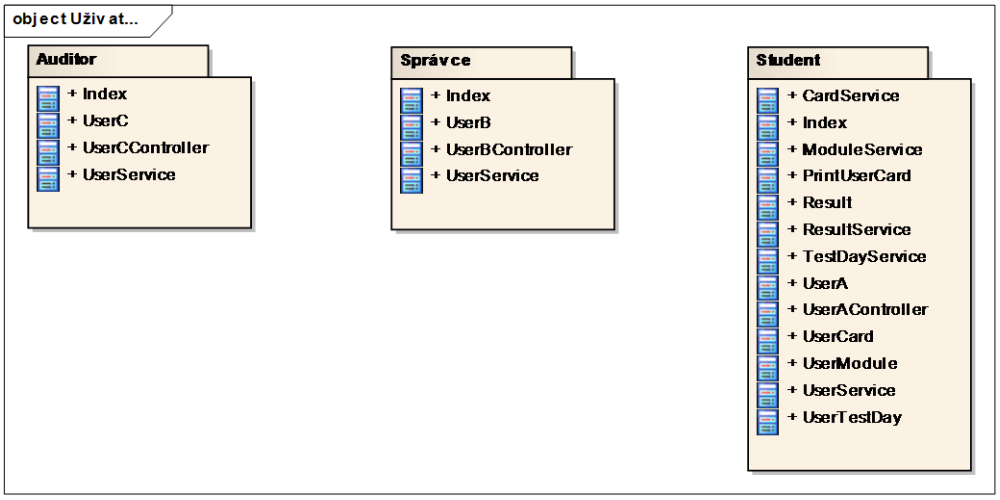
Místnosti



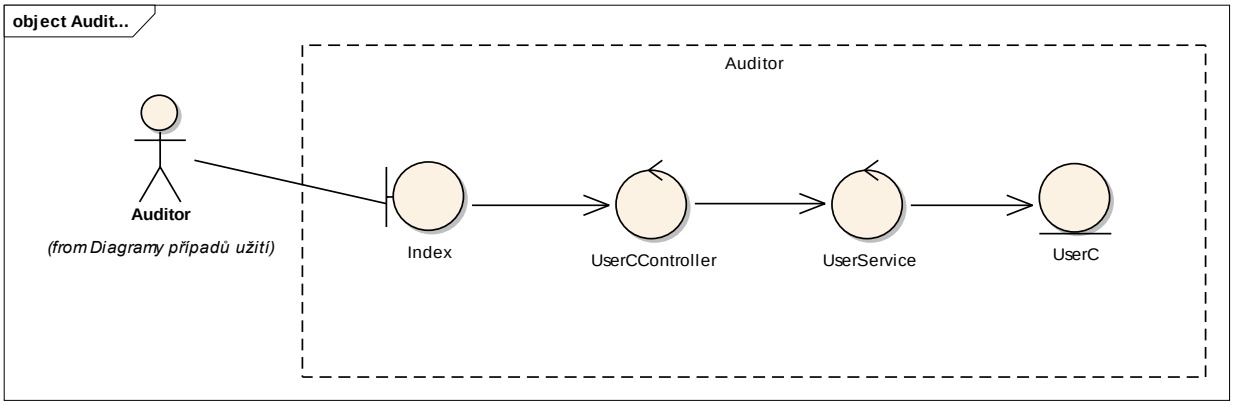
Statistiky



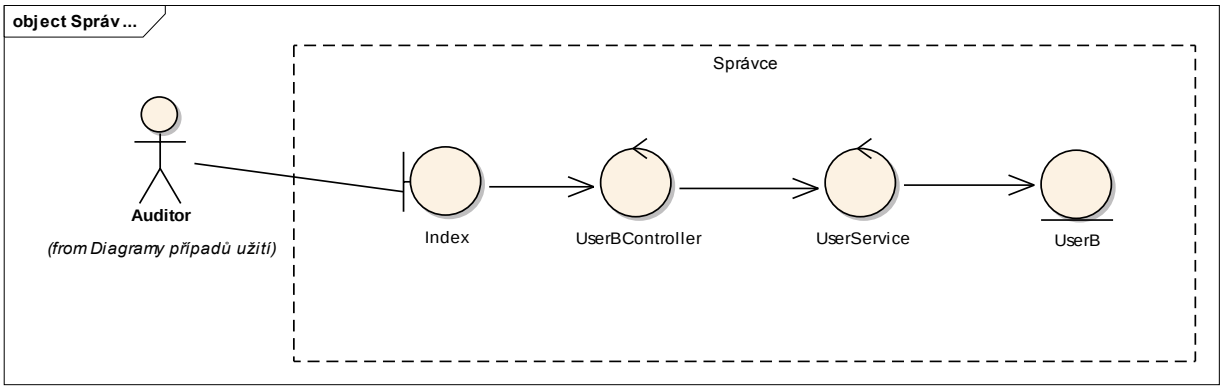
Uživatelé



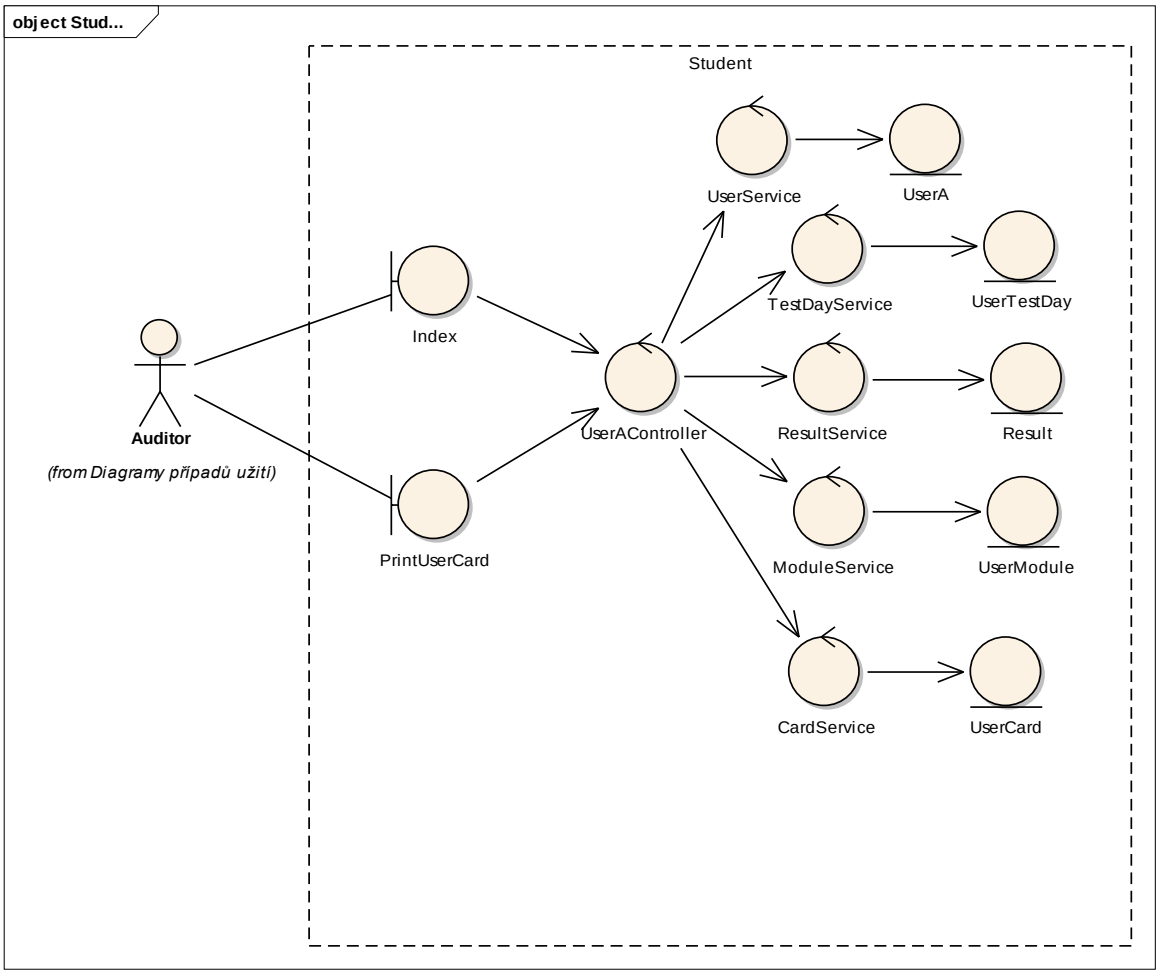
Auditor



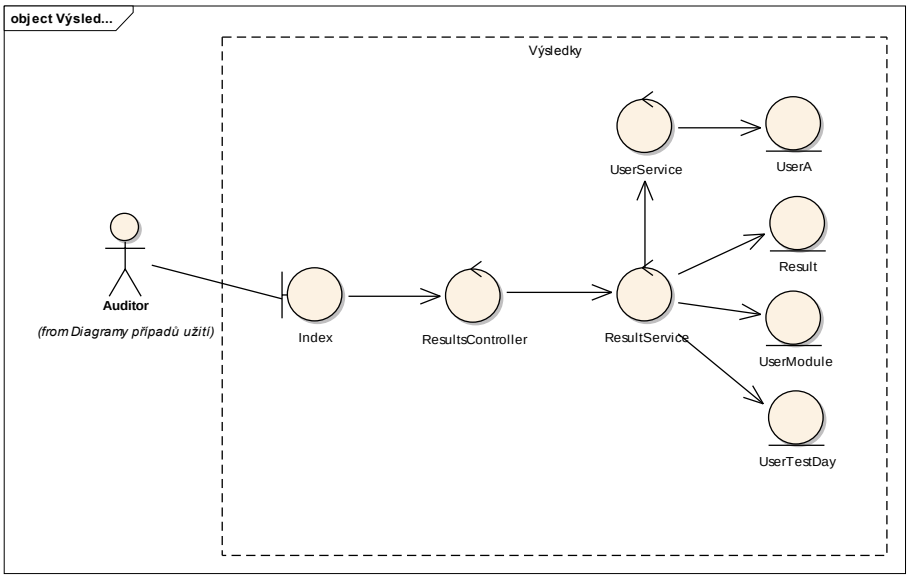
Správce



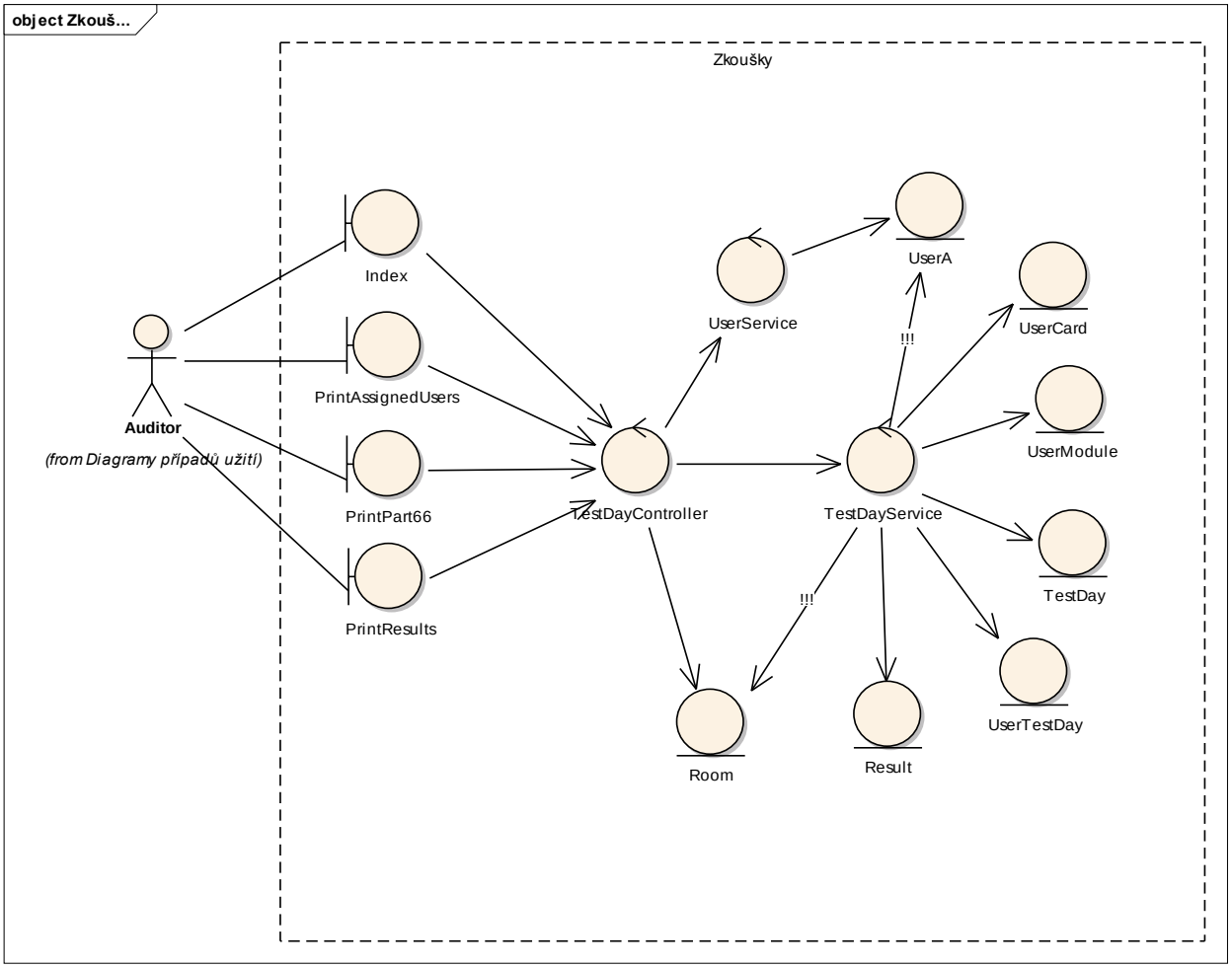
Student



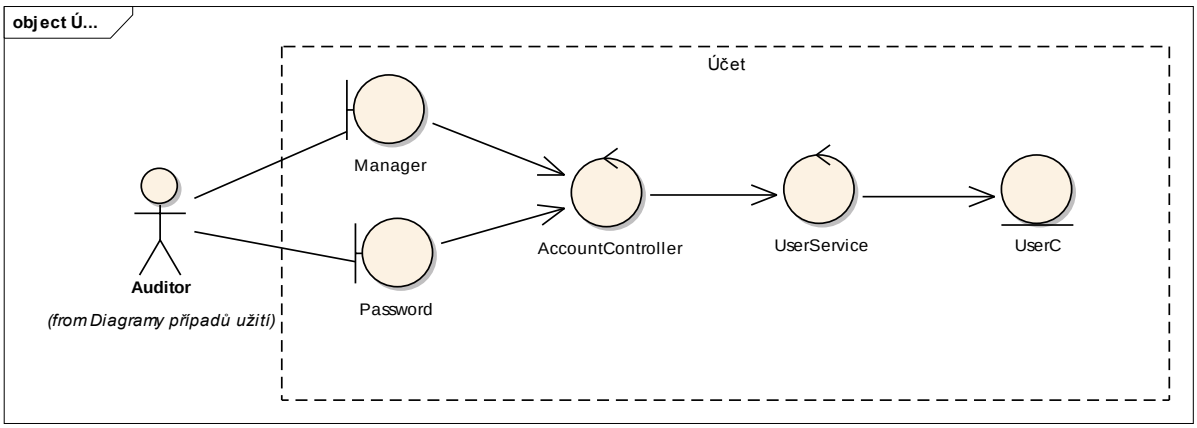
Výsledky



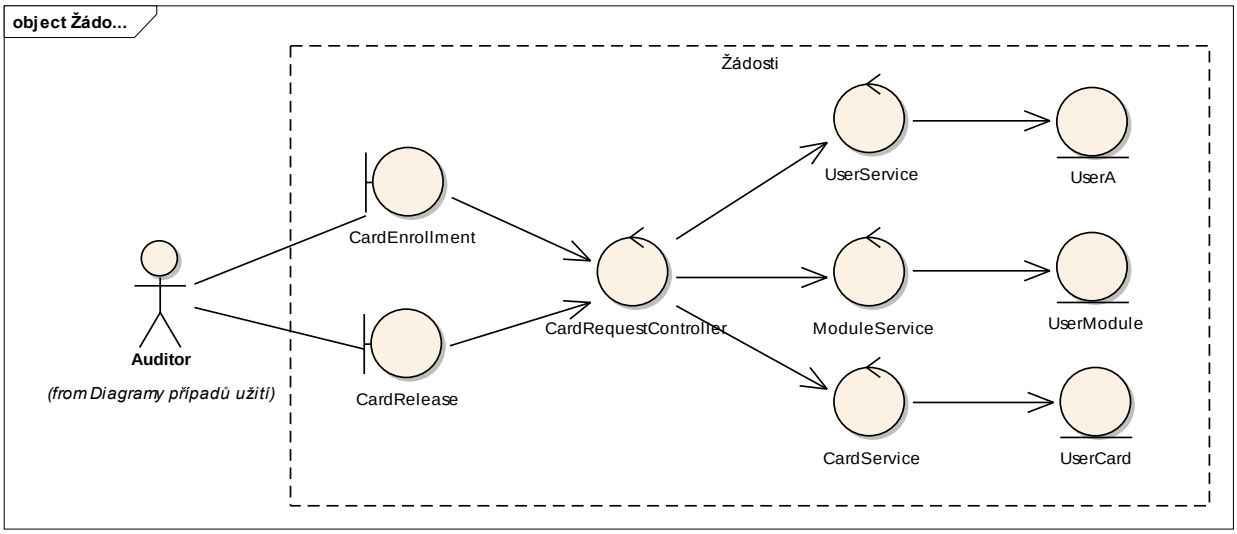
Zkoušky



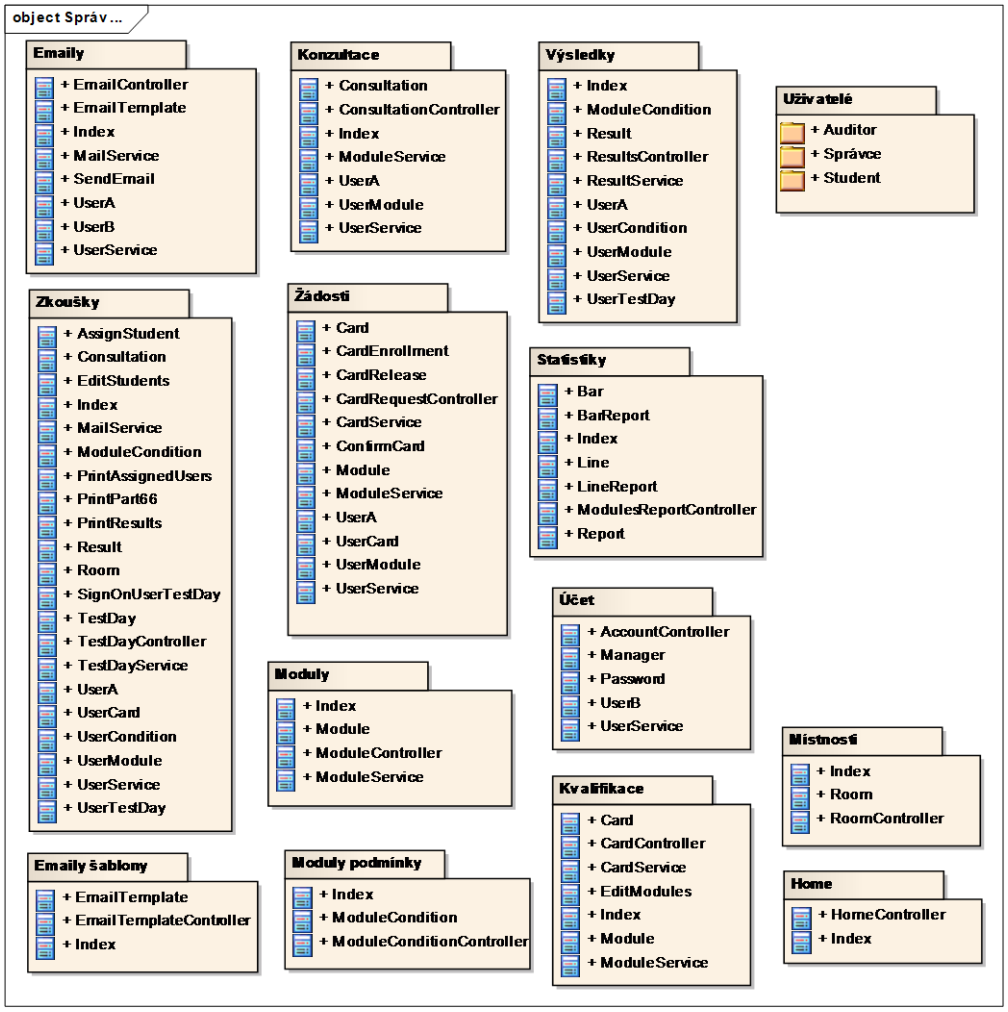
Účet



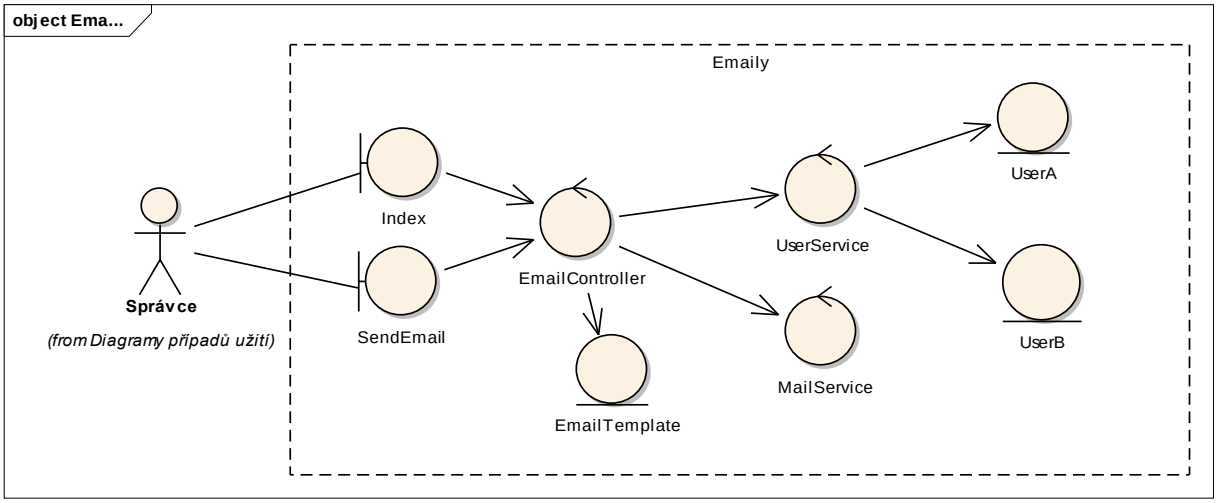
Žádosti



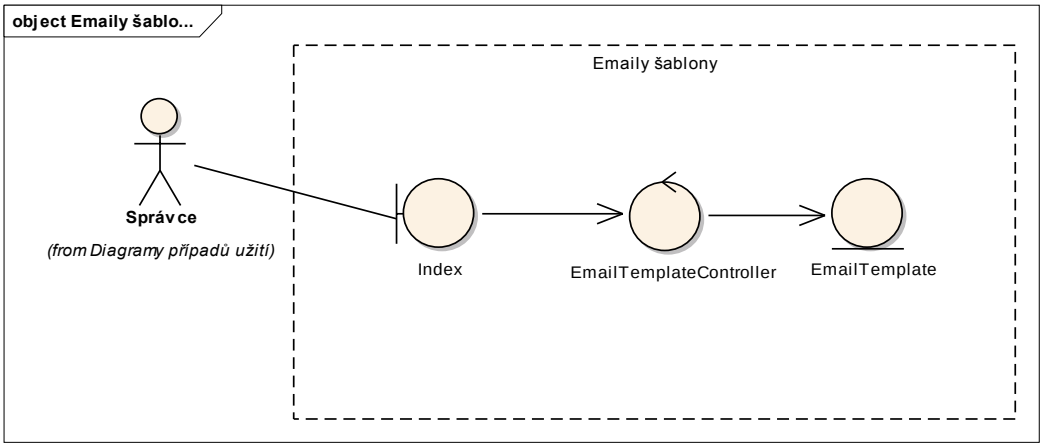
Správce



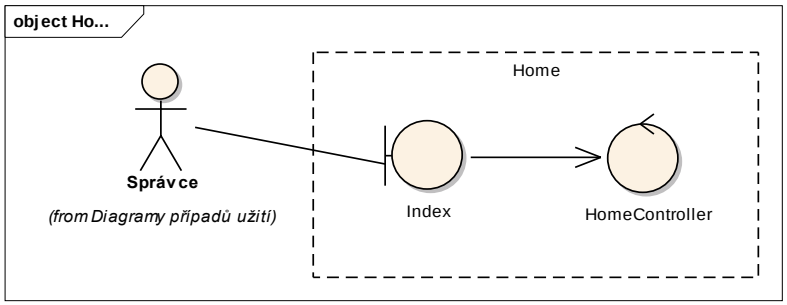
Emaily



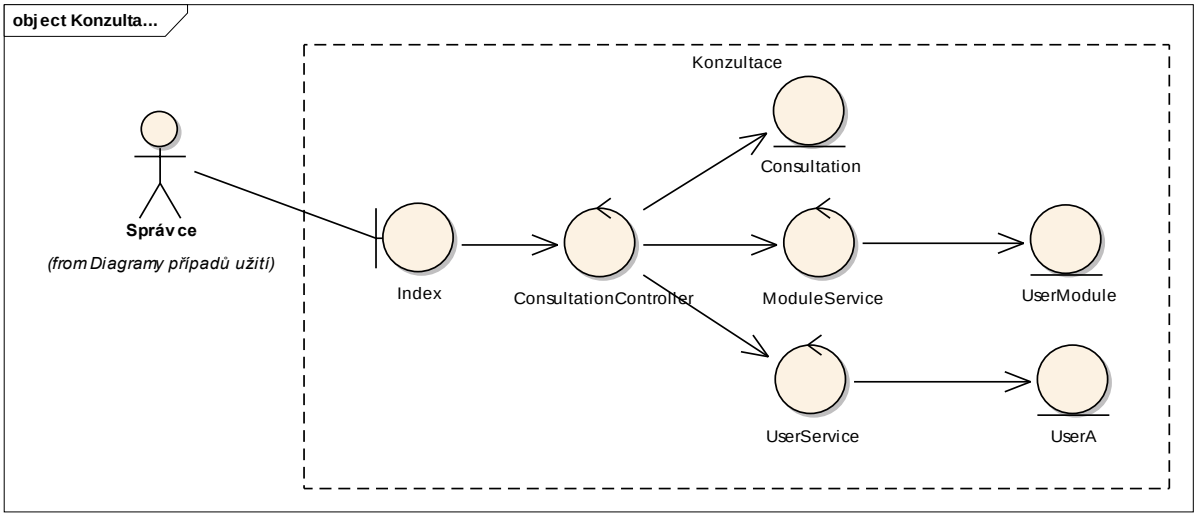
Emaily šablony



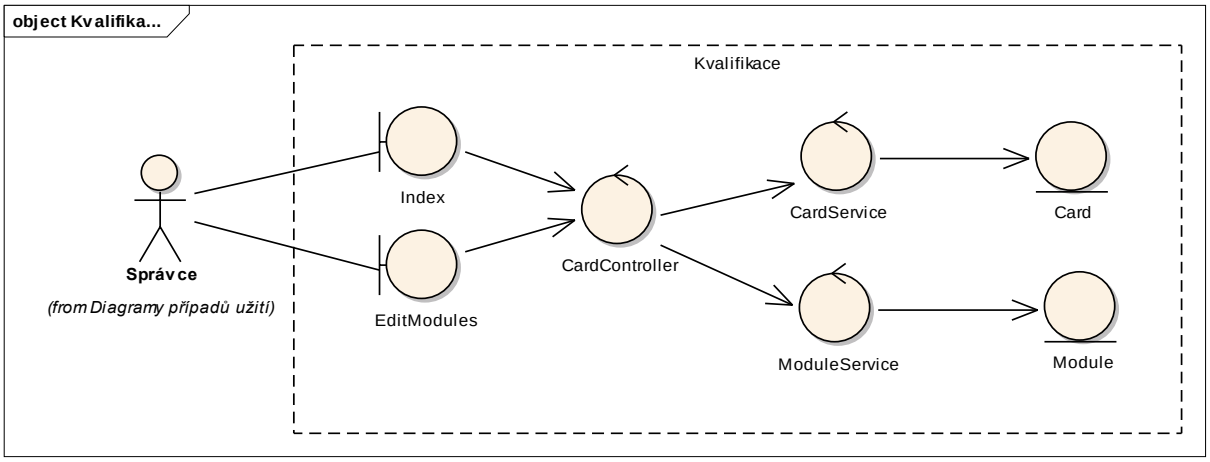
Home



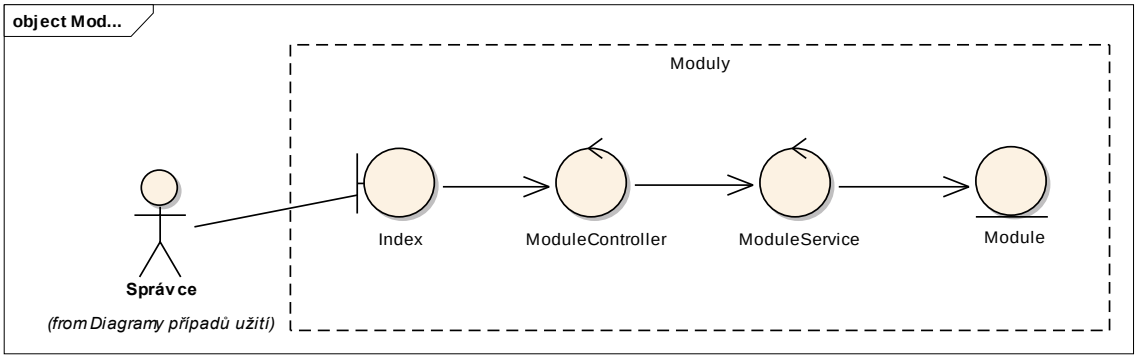
Konzultace



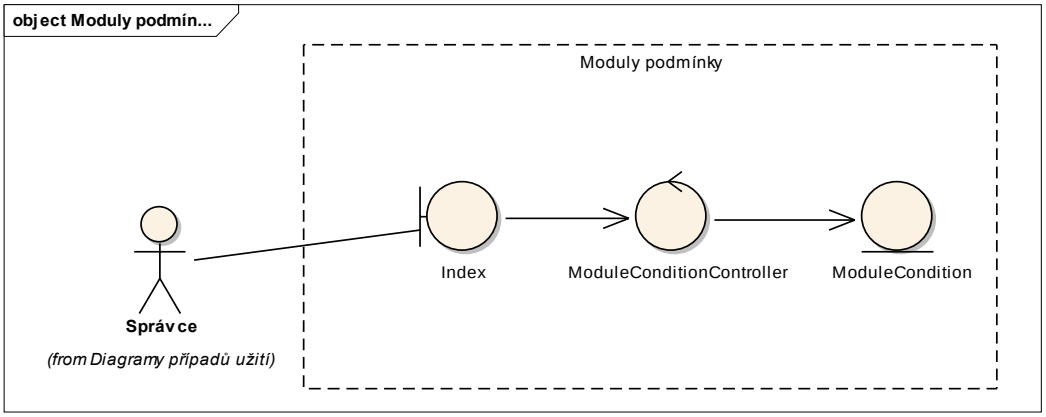
Kvalifikace



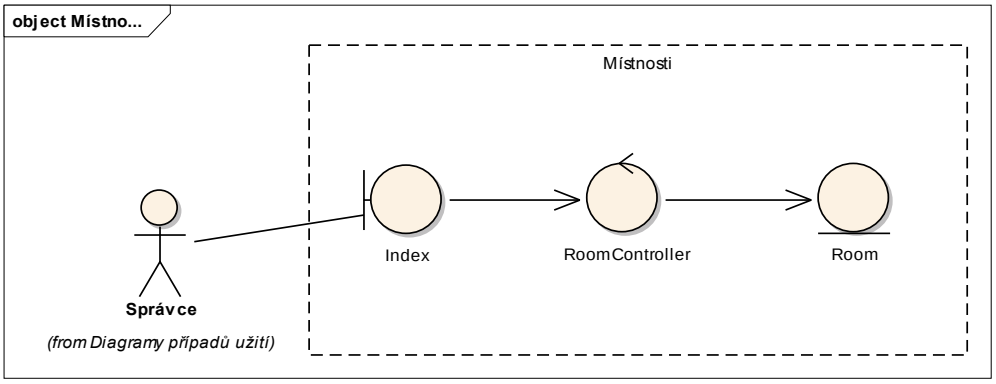
Moduly



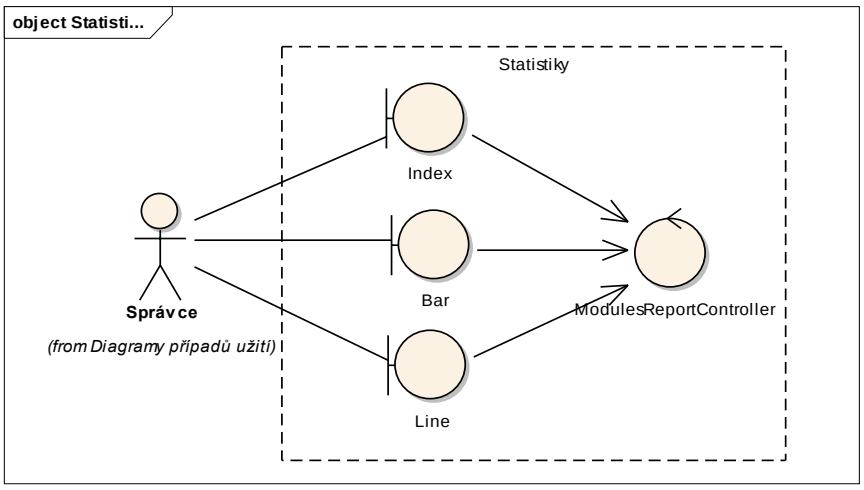
Moduly podmínky



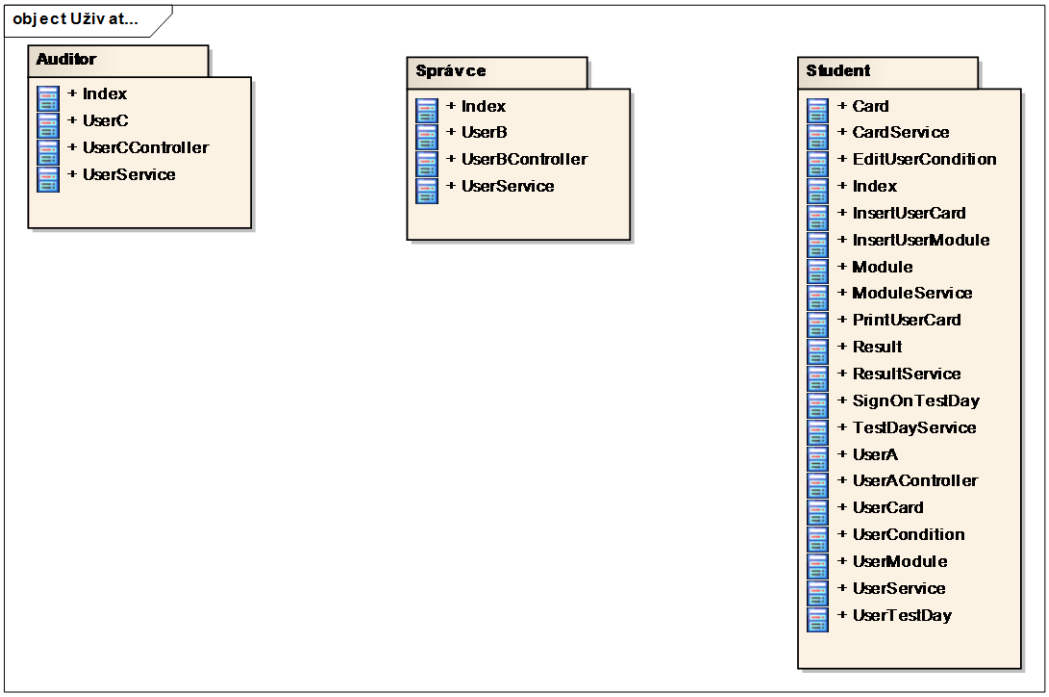
Místnosti



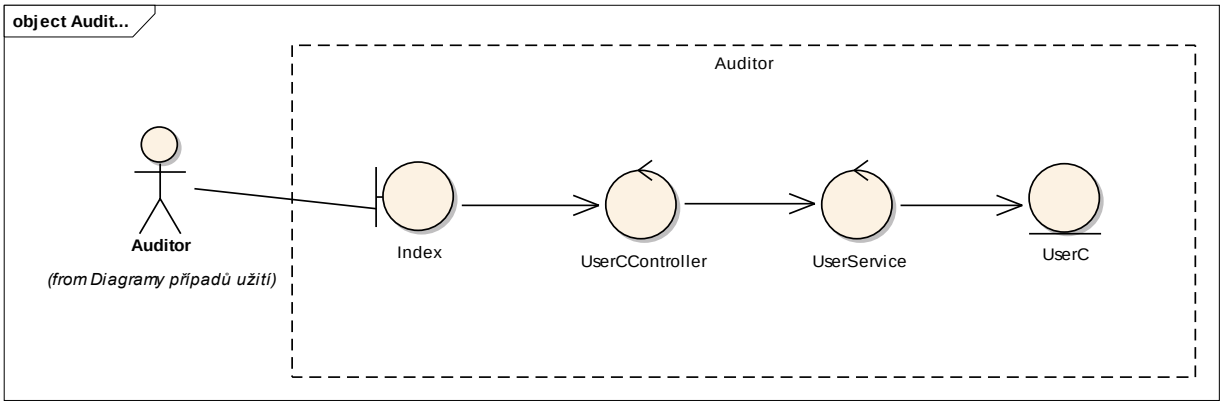
Statistiky



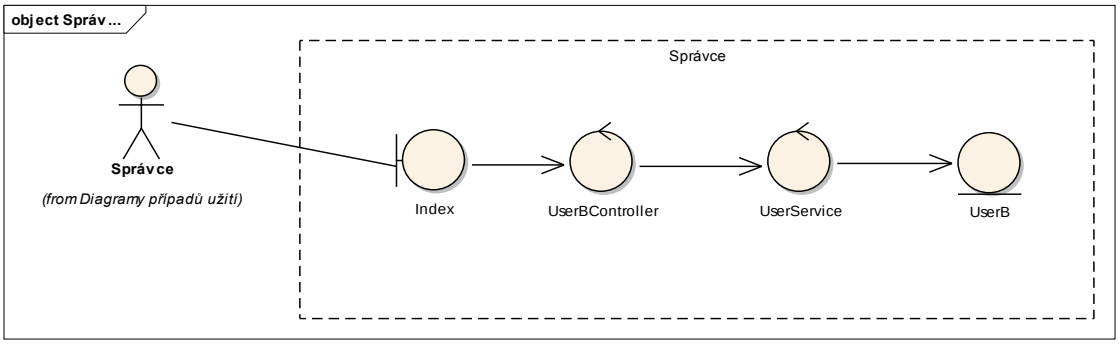
Uživatelé



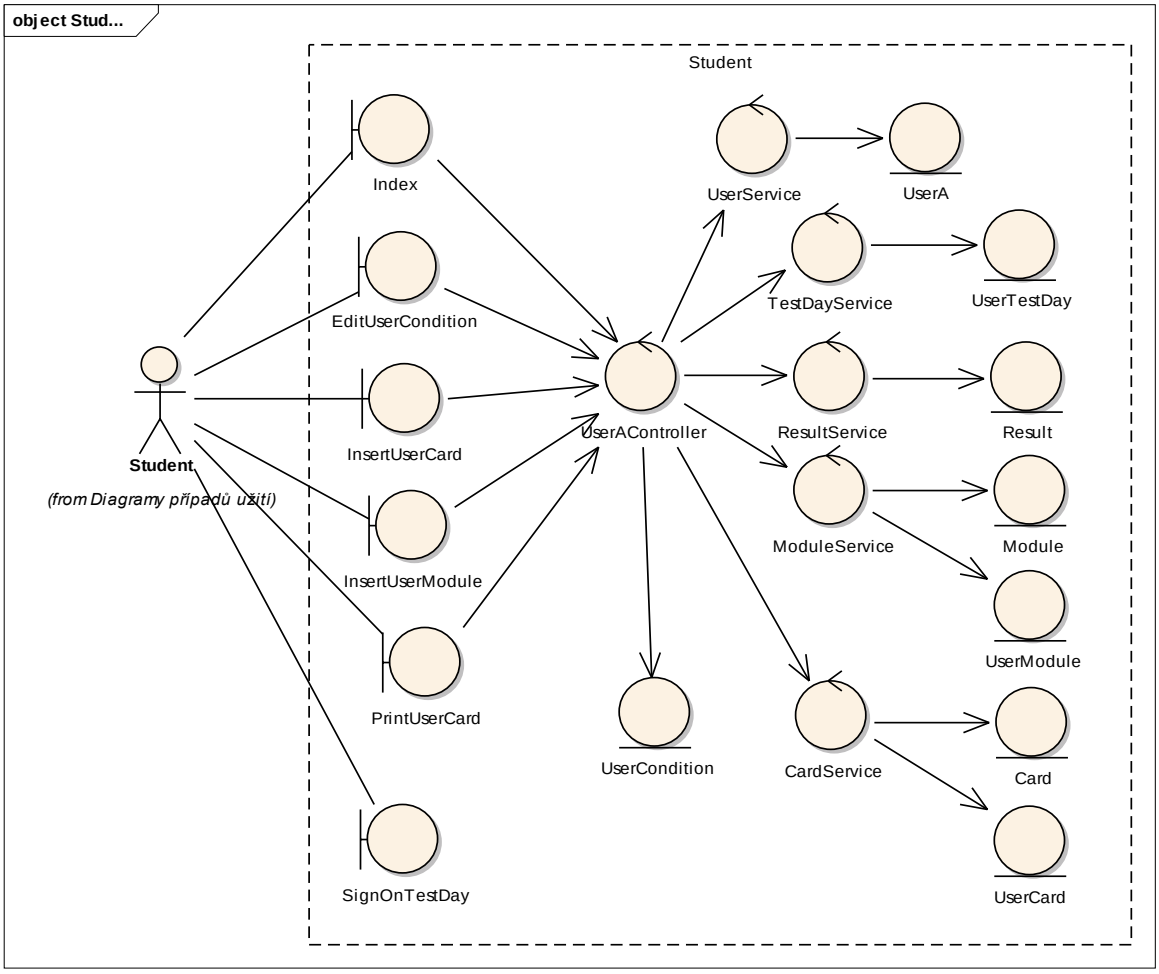
Auditor



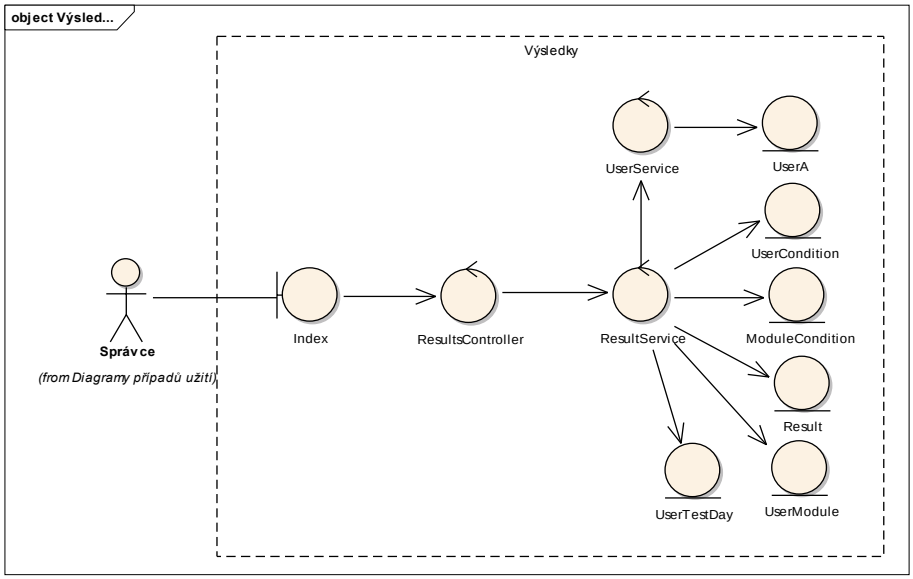
Správce



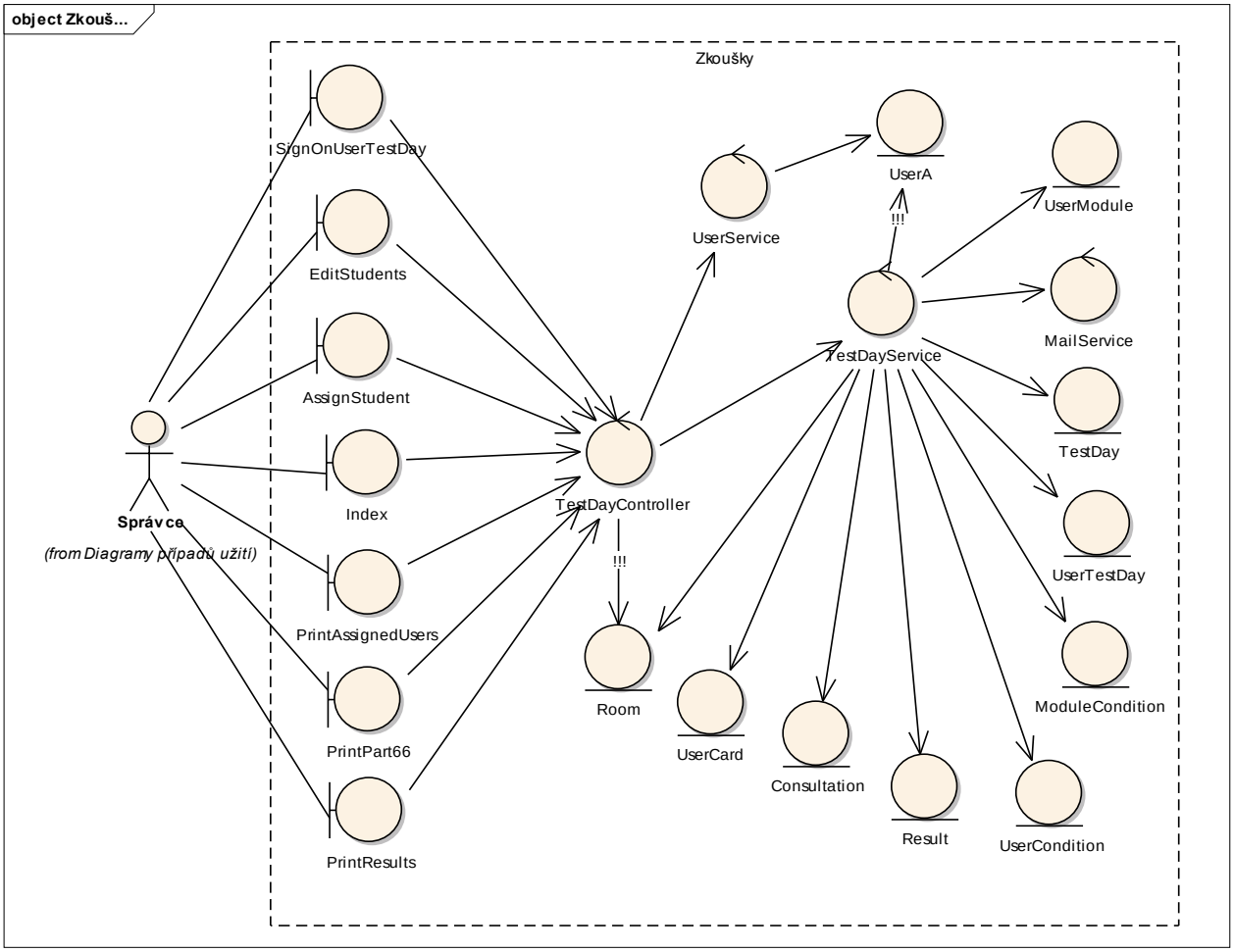
Student



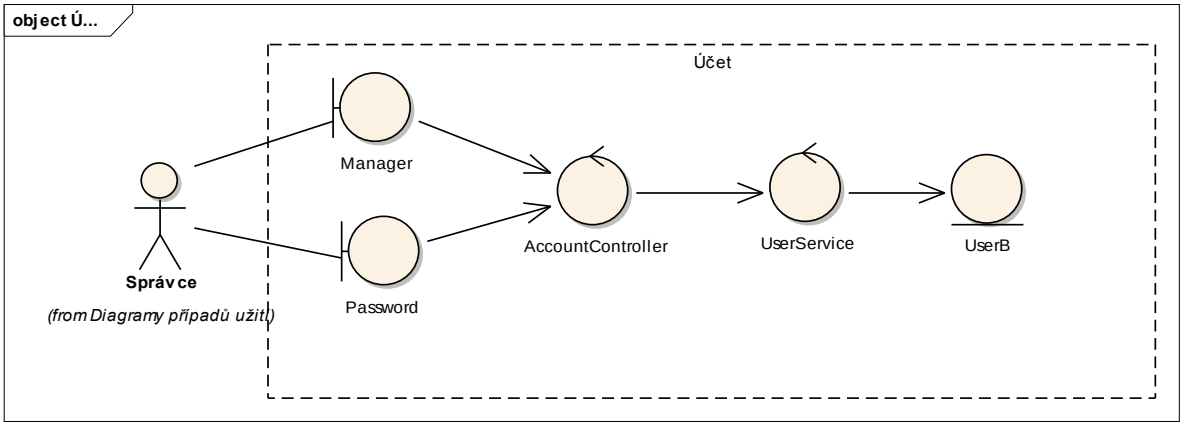
Výsledky



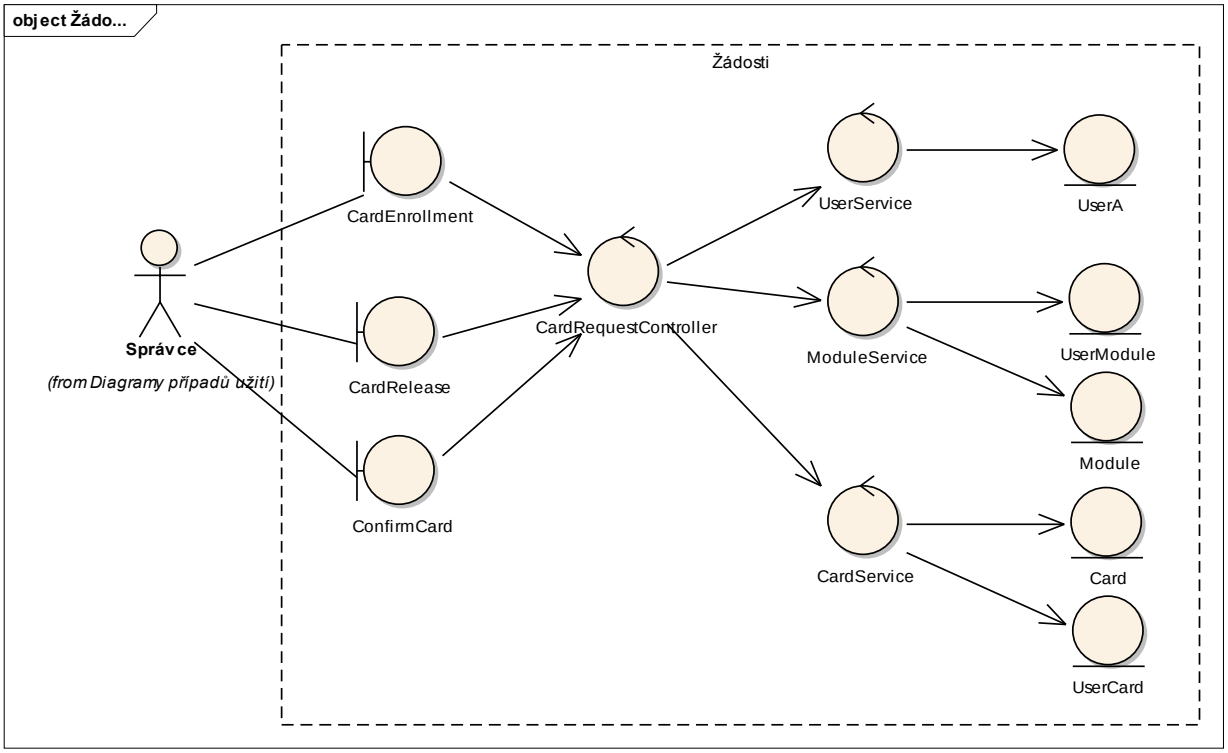
Zkoušky



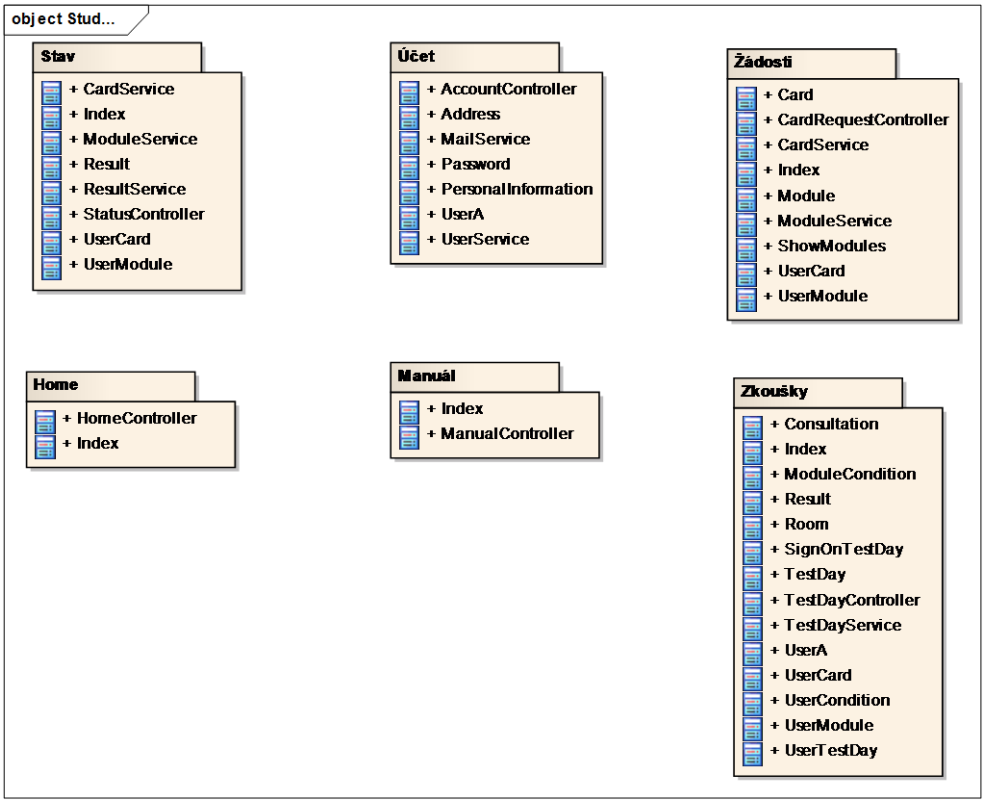
Účet



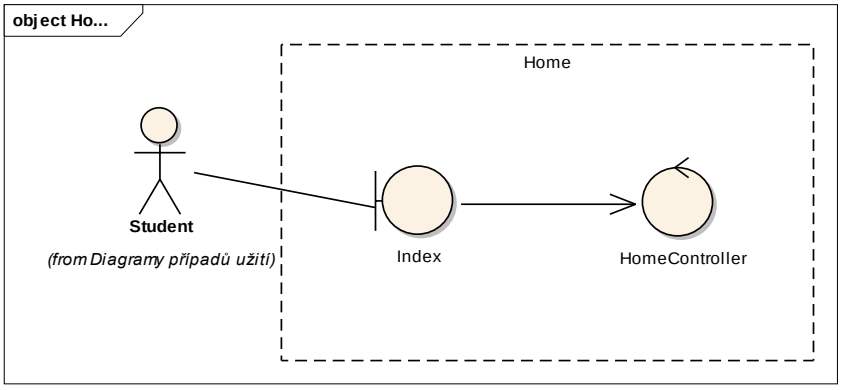
Žádosti



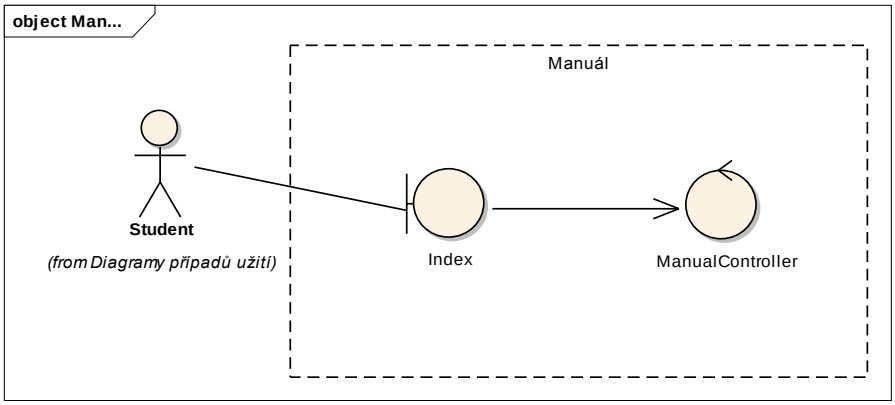
Student



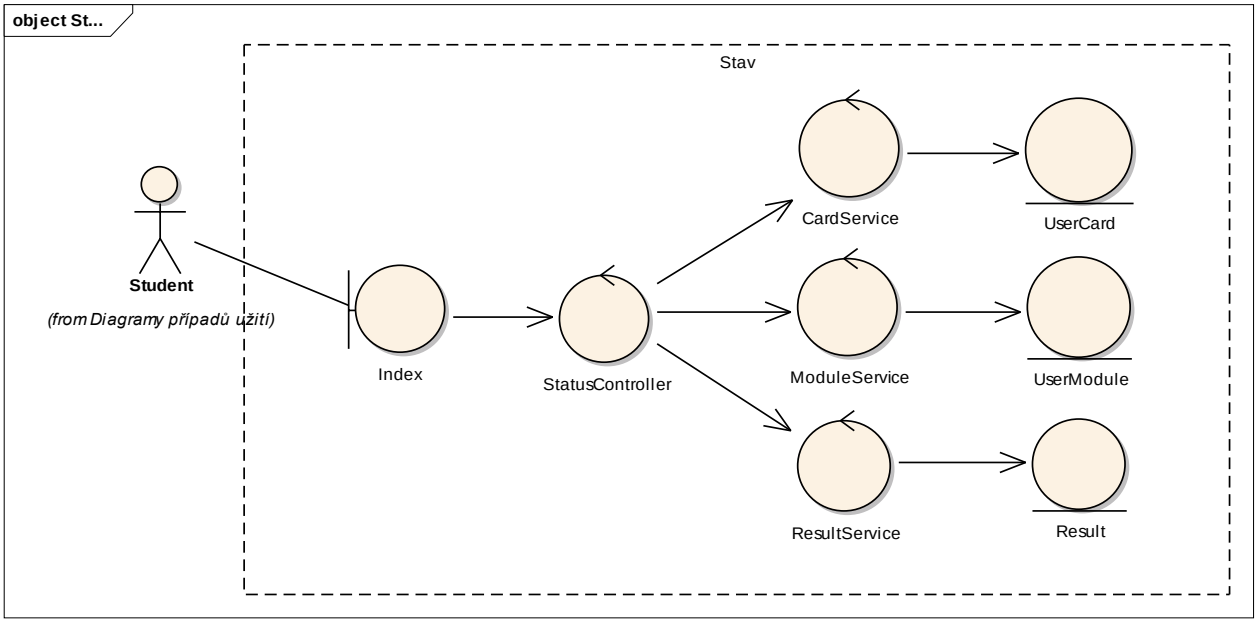
Home



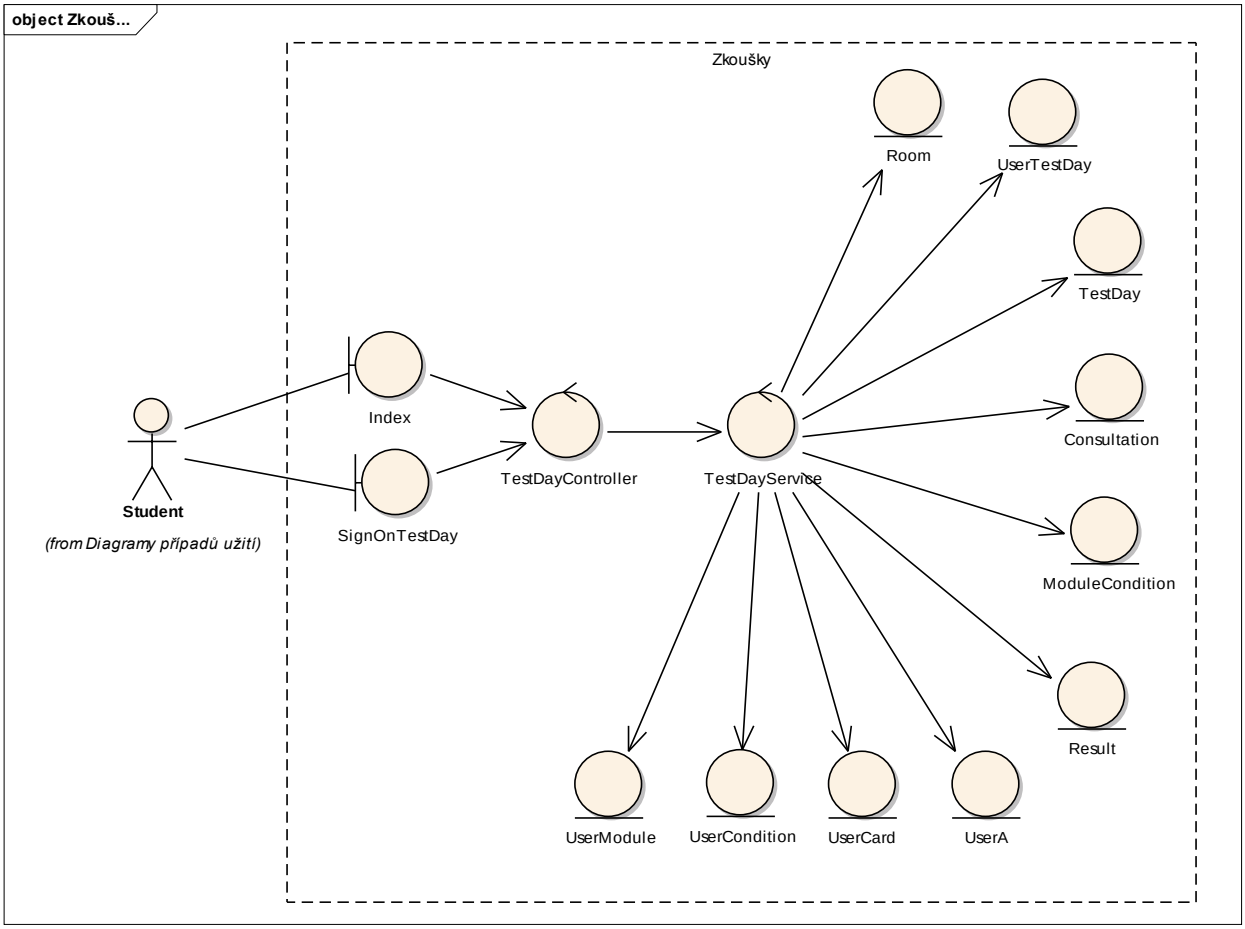
Manuál



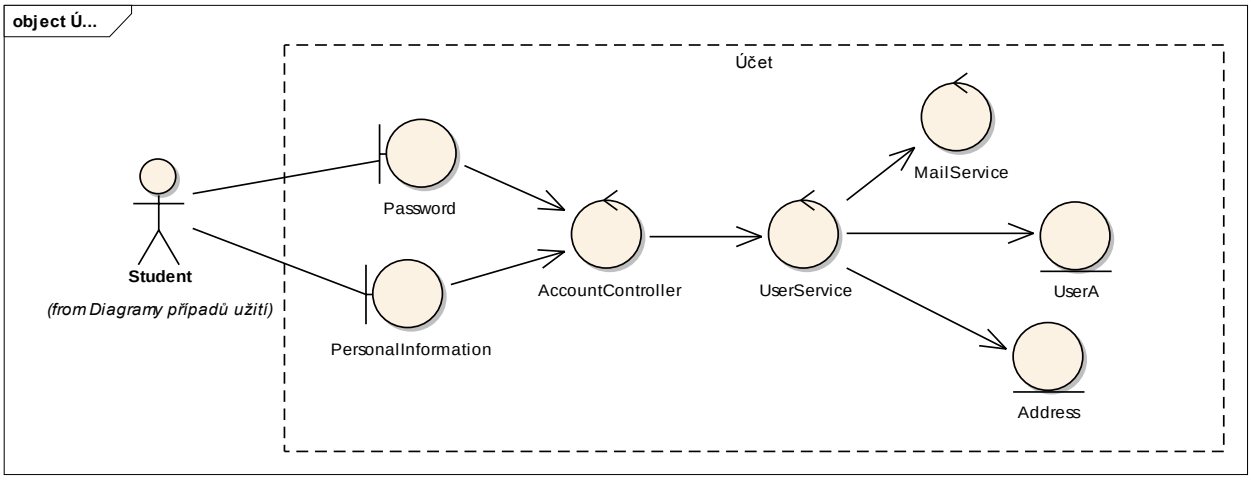
Stav



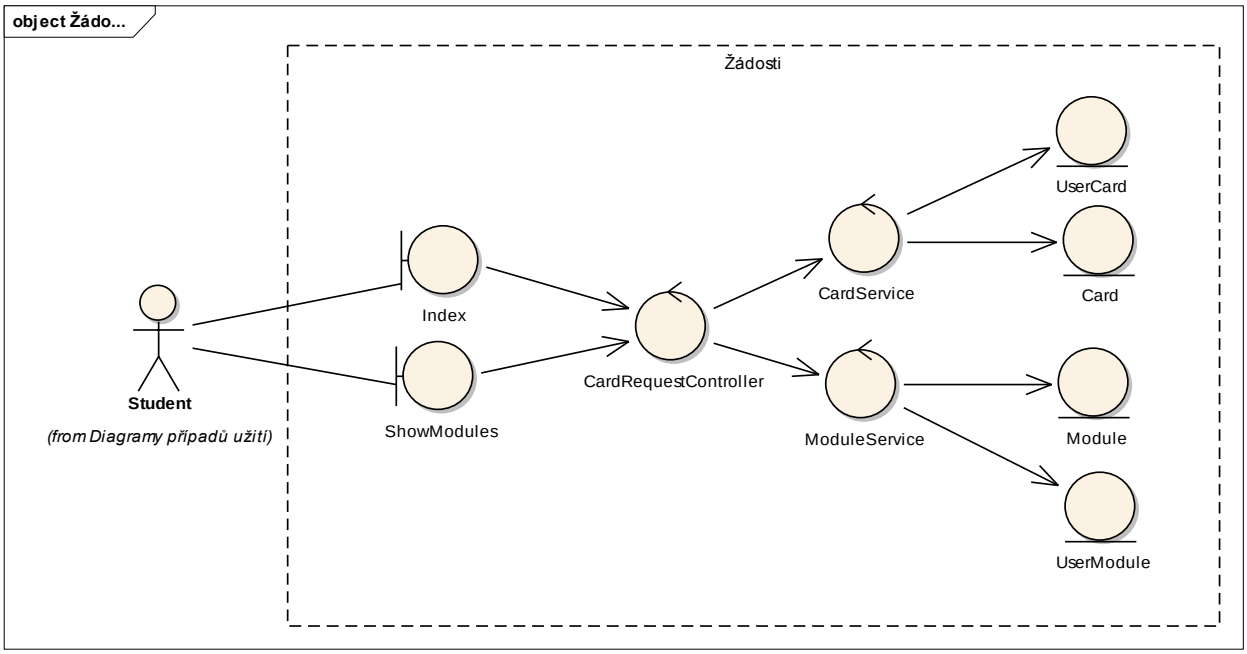
Zkoušky



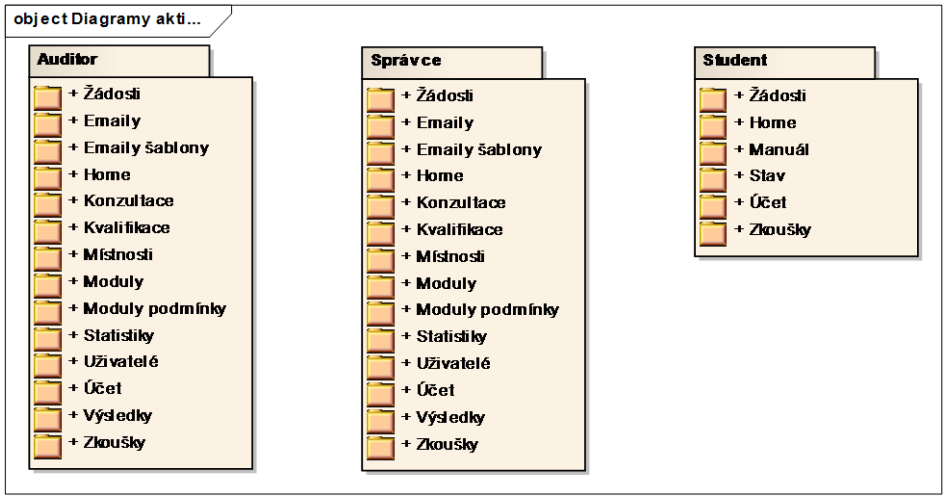
Účet



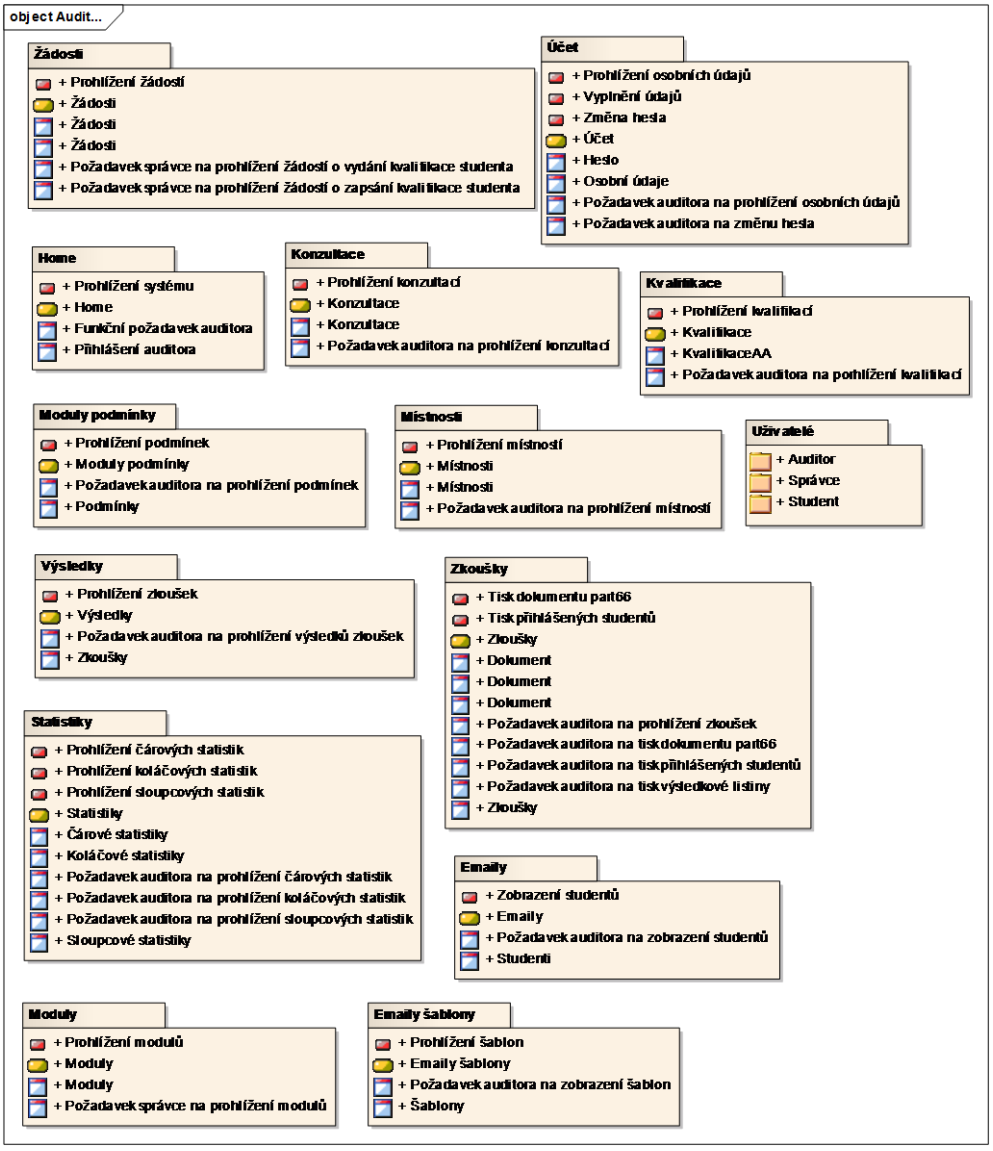
Žádosti


object Žádo...
Žádosti
UserCard
Card
CardService
Index
CardRequestController
Student
(from Diagramy případů užití)
ShowModules
ModuleService
Module
UserModule

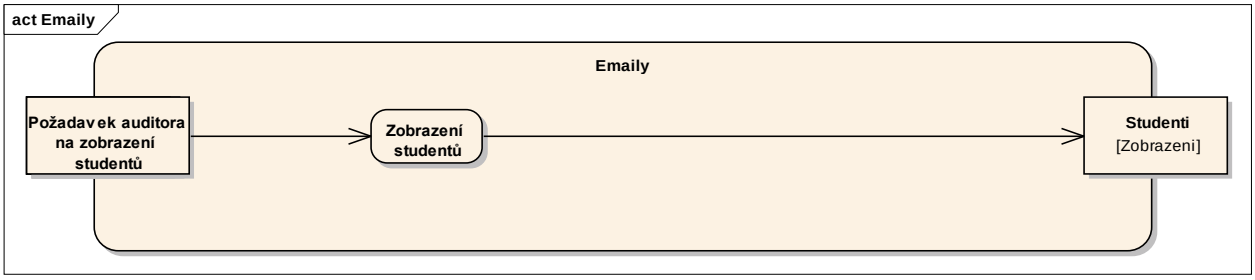
PŘÍLOHA E: Diagramy aktivit



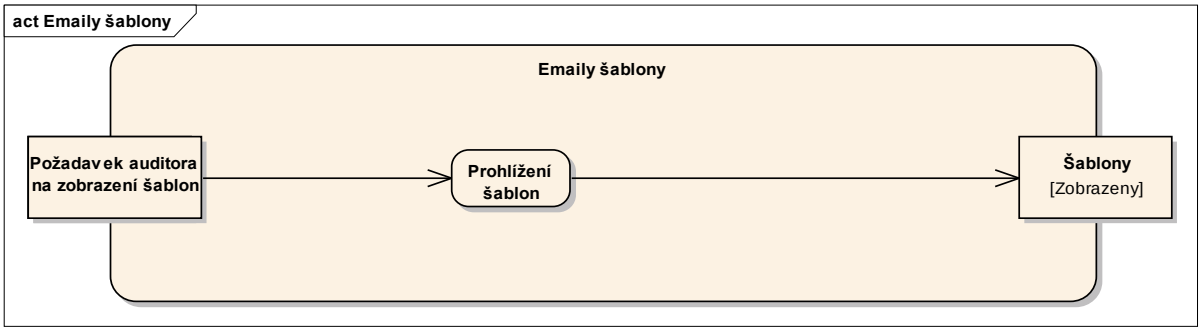
Auditor



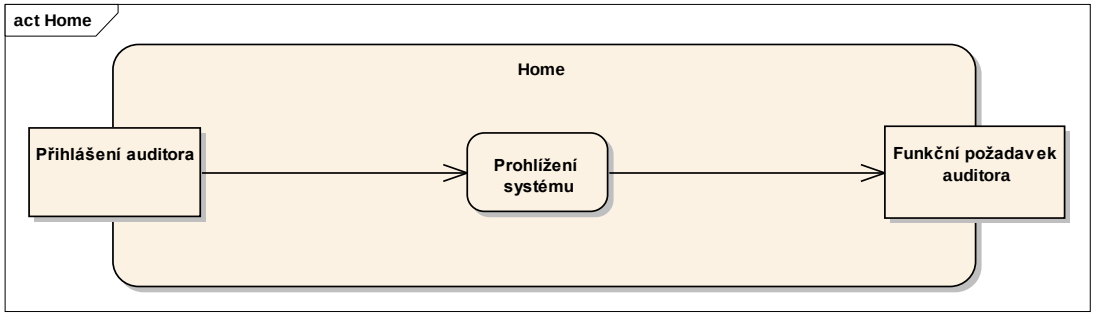
Emaily



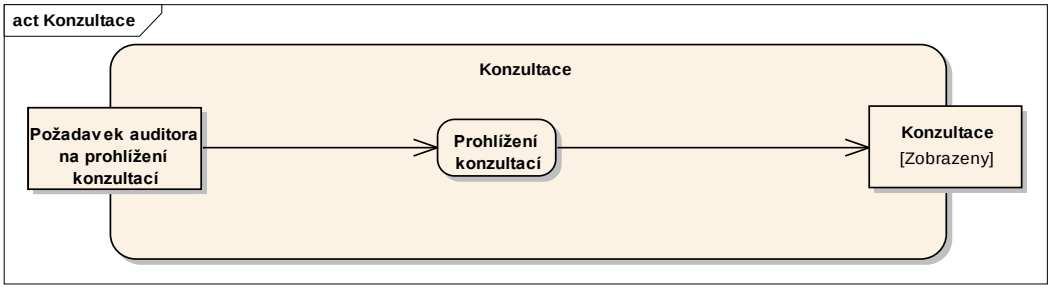
Emaily šablony



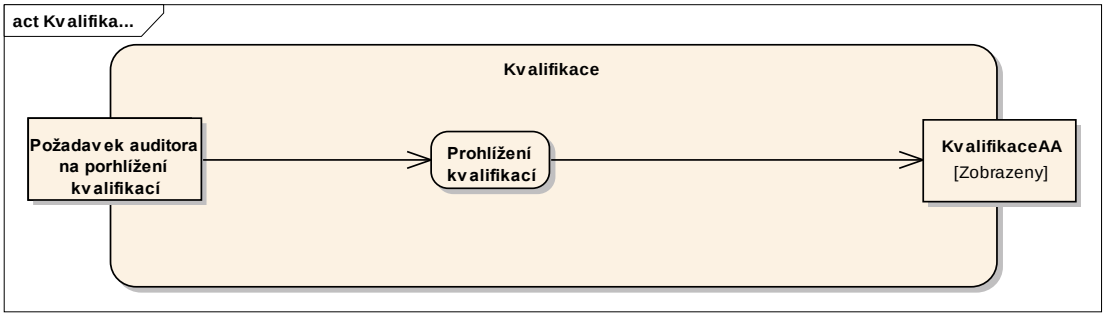
Home



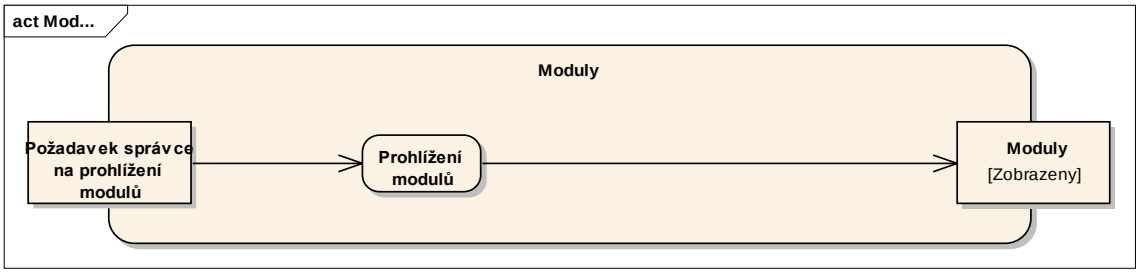
Konzultace



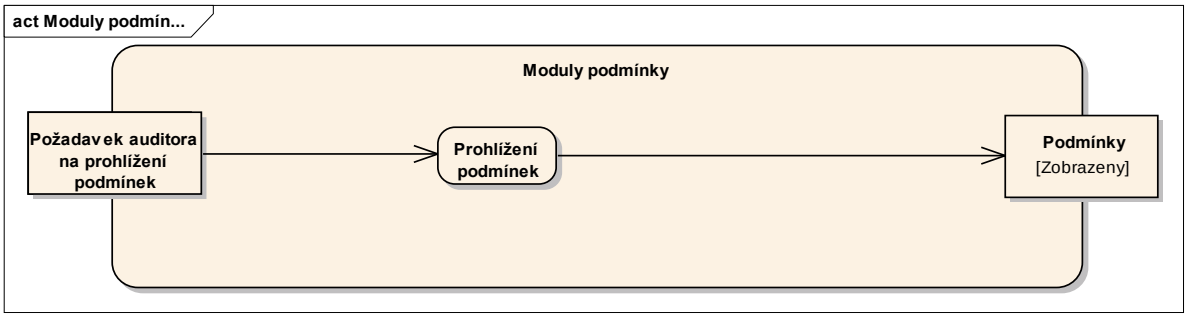
Kvalifikace



Moduly

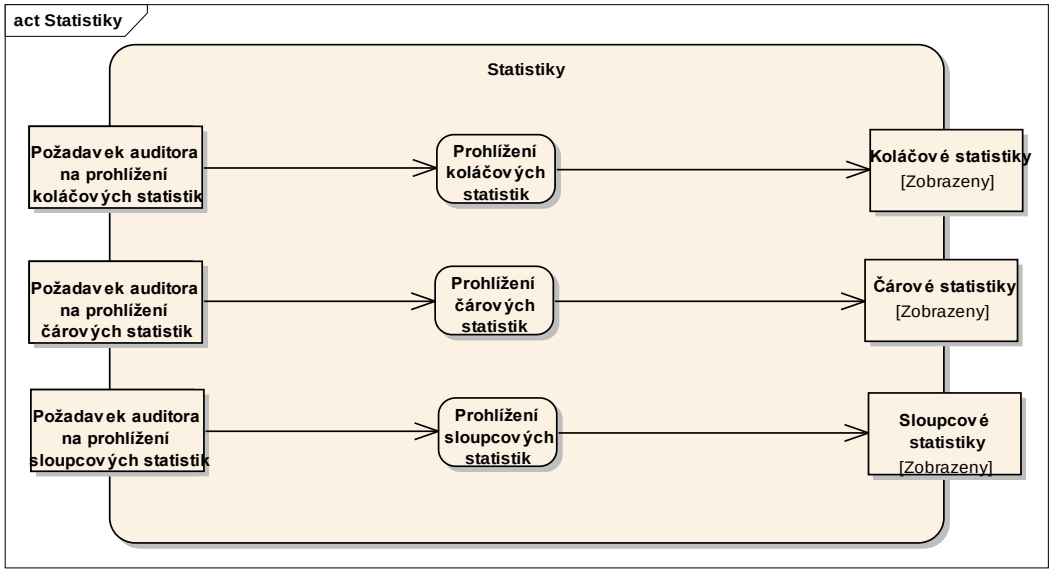


Moduly podmínky

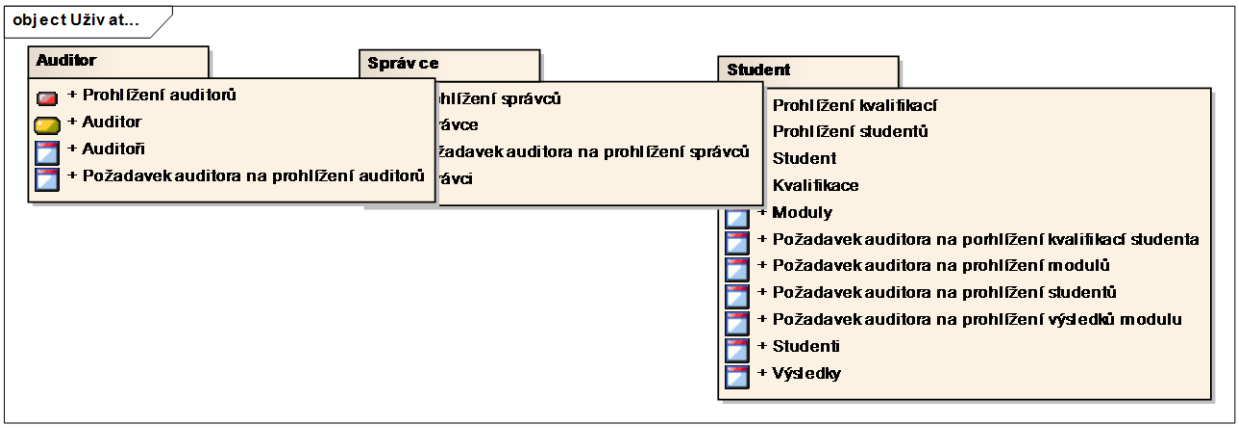


Místnosti

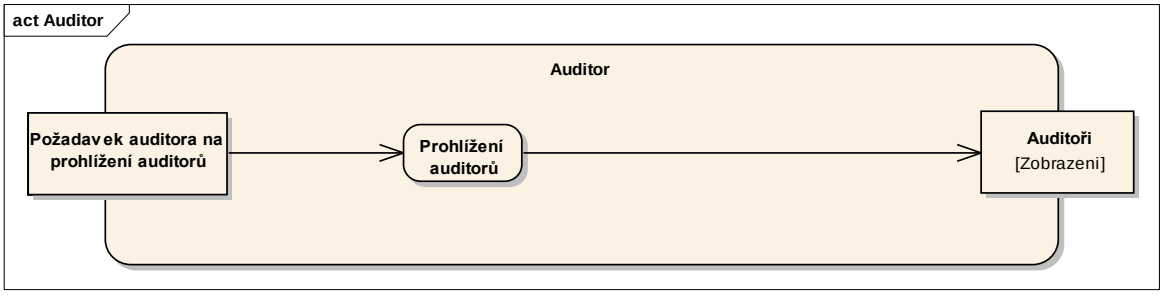
Statistiky



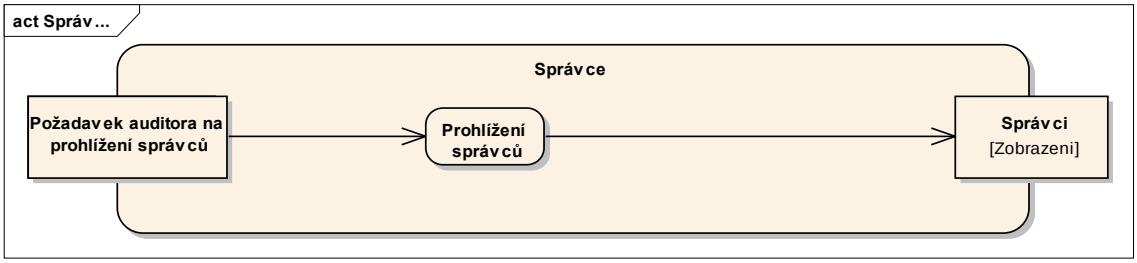
Uživatelé



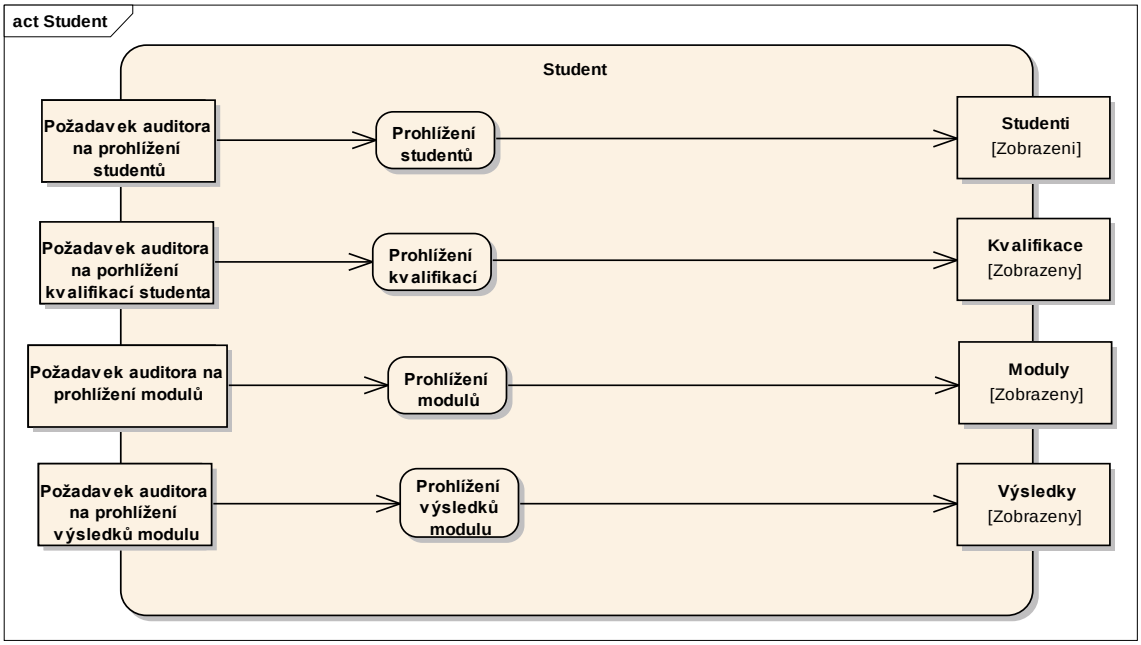
Auditor



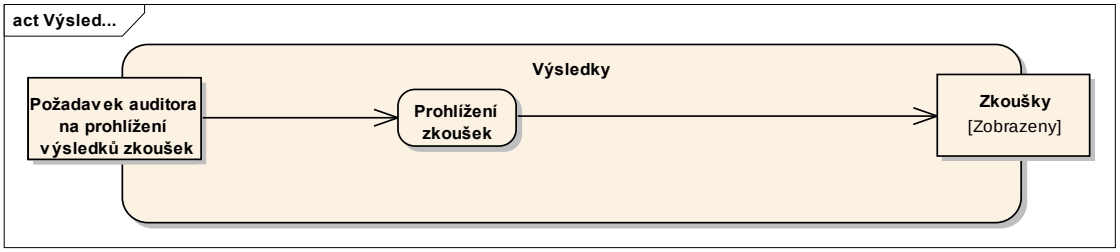
Správce



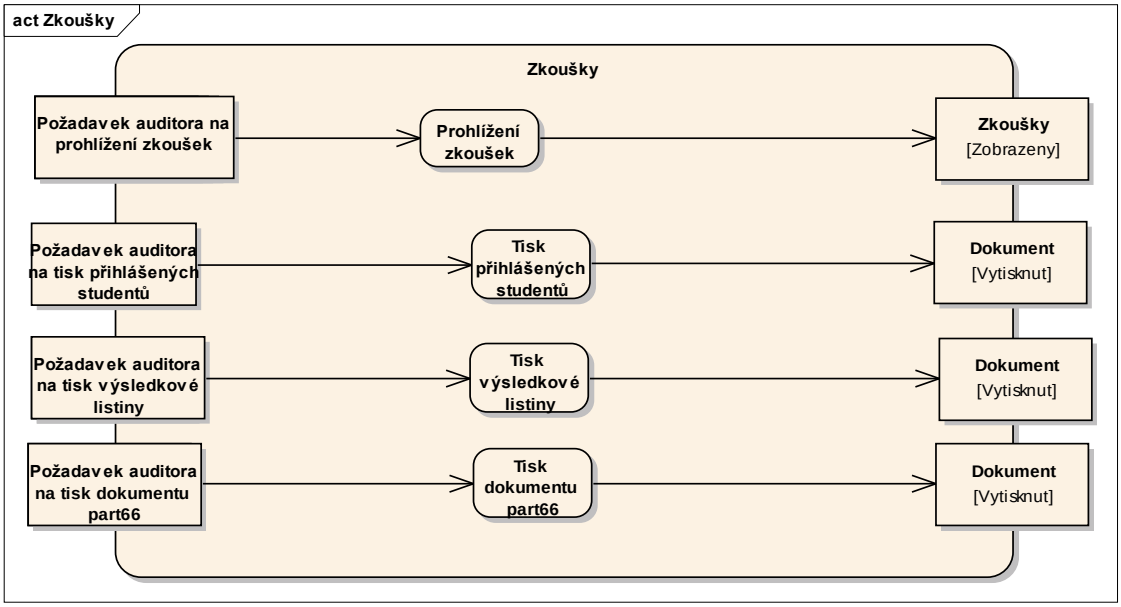
Student



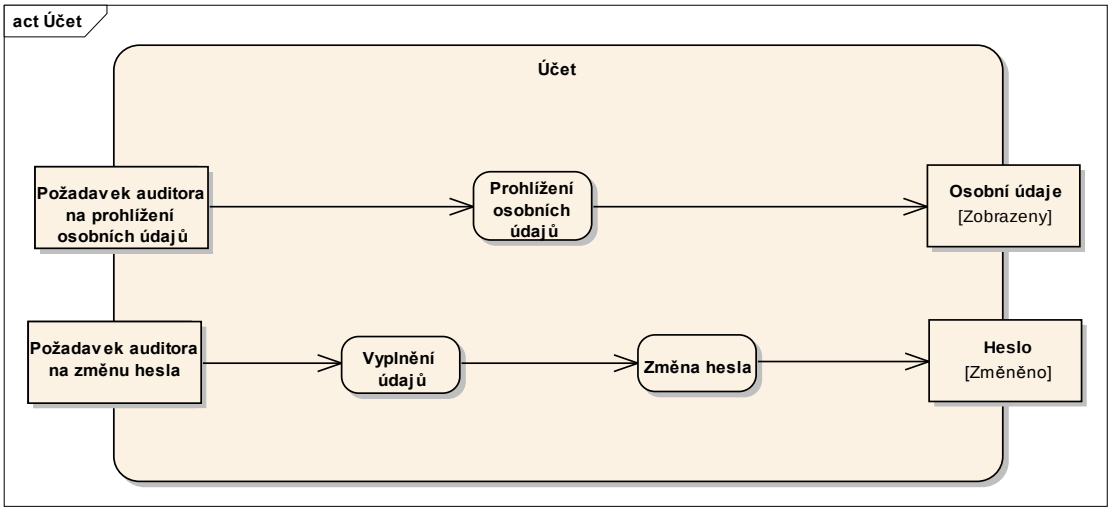
Výsledky



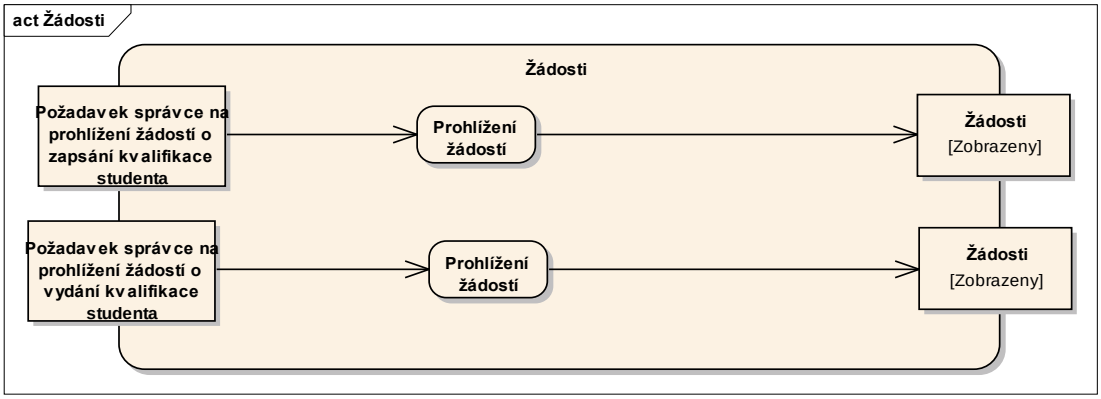
Zkoušky



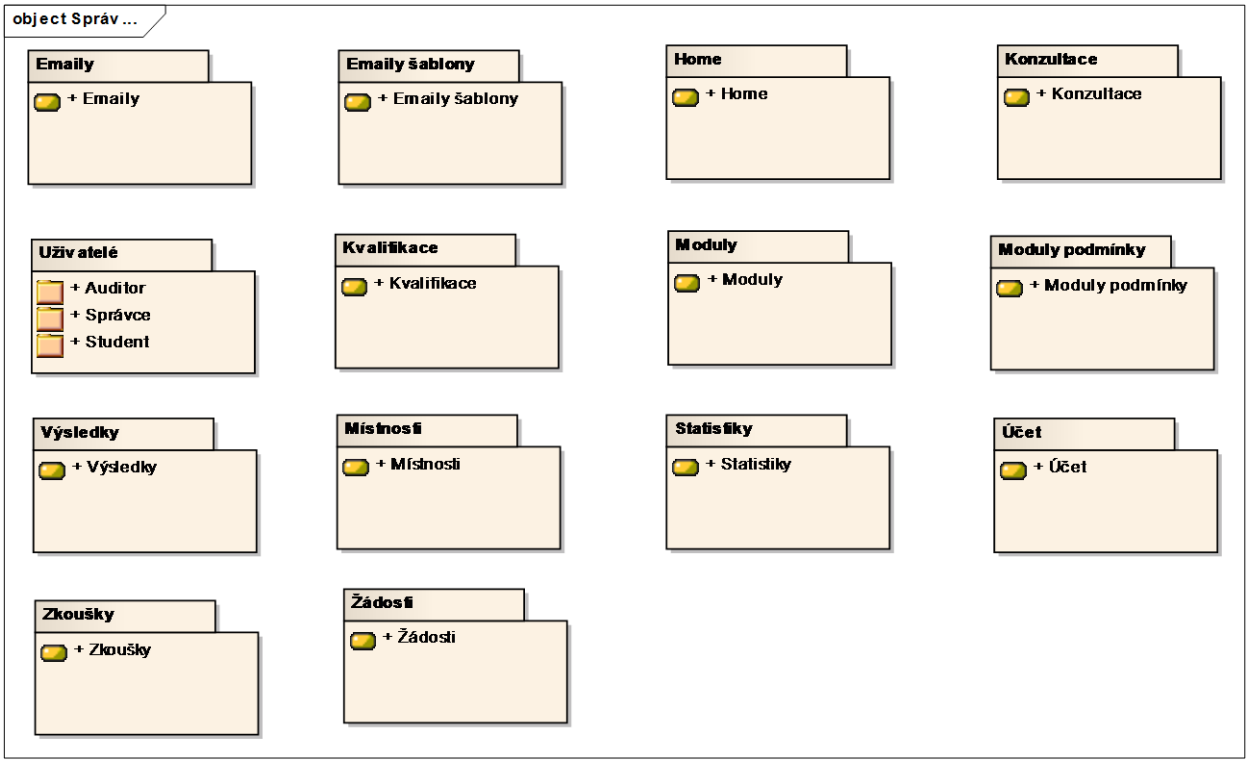
Účet



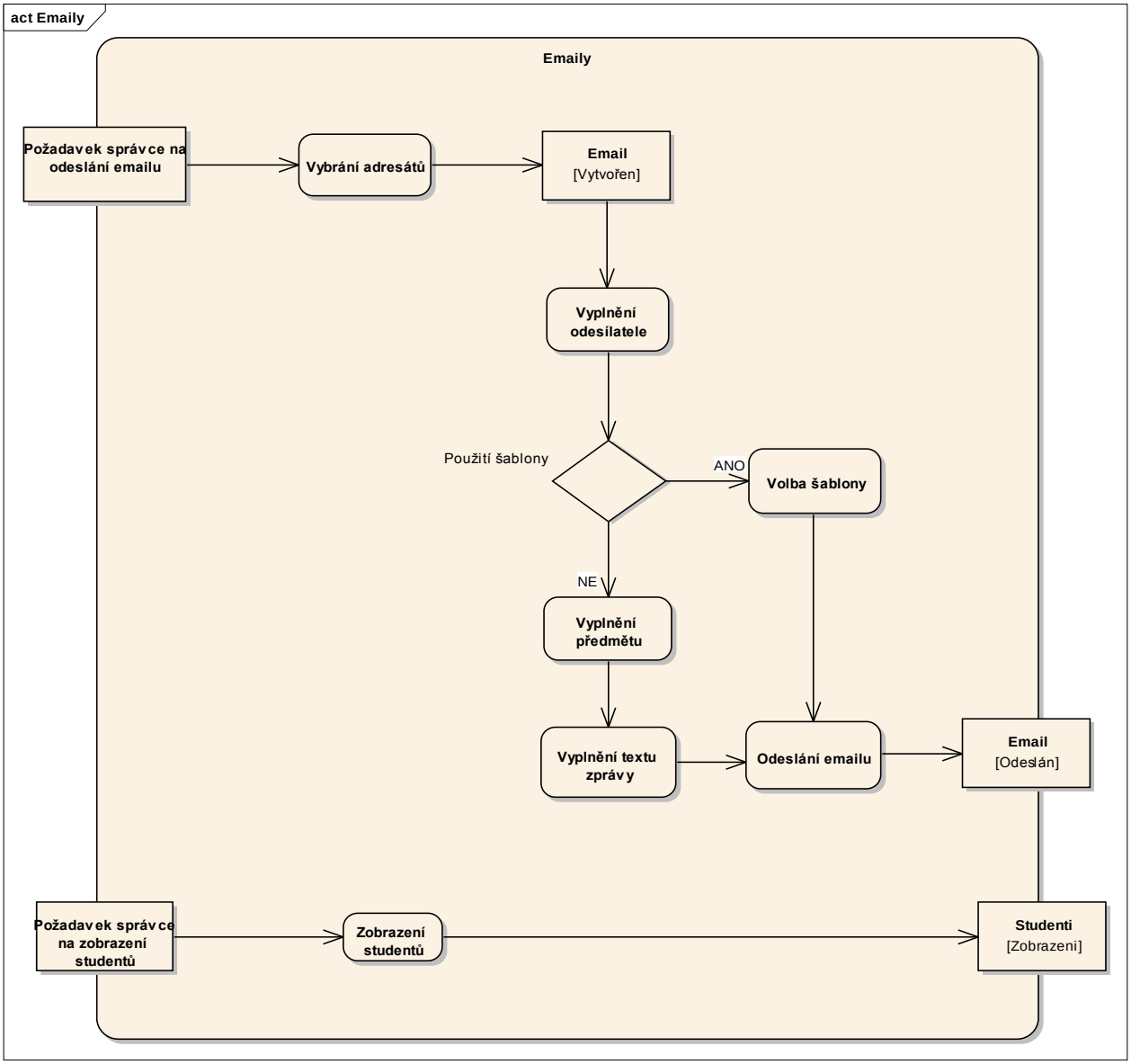
Žádosti



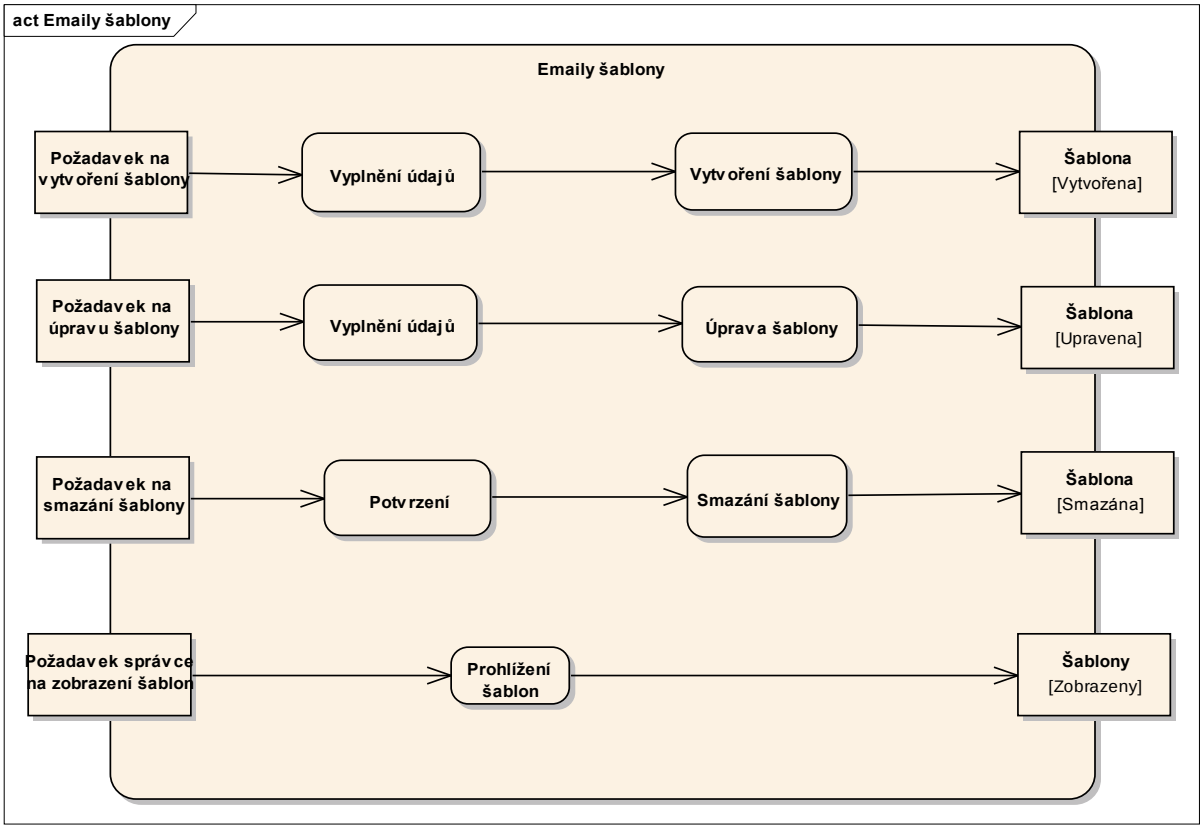
Správce



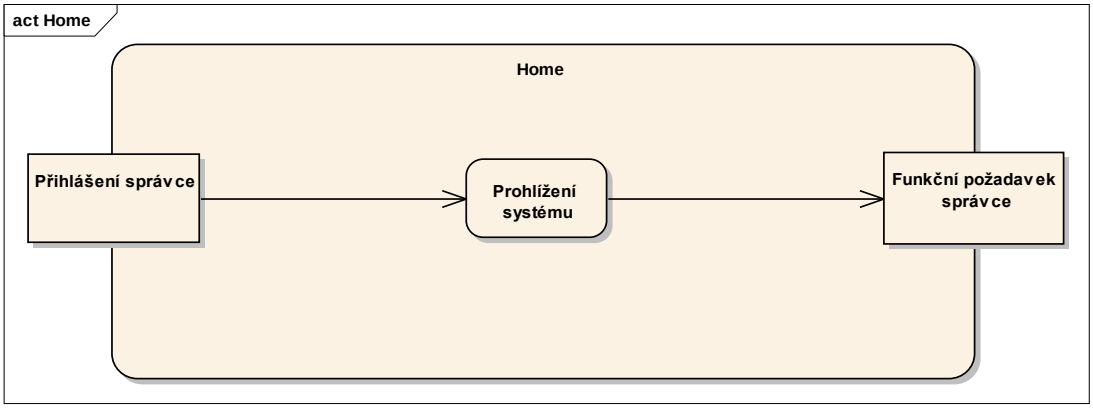
Emaily


act Emaily
Emaily
Požadavek správce na odeslání emailu
Vybrání adresátů
Email
[Vytvořen]
Vyplnění odesílatele
Použití šablony
ANO
Volba šablony
NE
Vyplnění předmětu
Vyplnění textu zprávy
Odeslání emailu
Email
[Odeslán]
Požadavek správce na zobrazení studentů
Zobrazení studentů
Studenti
[Zobrazeni]

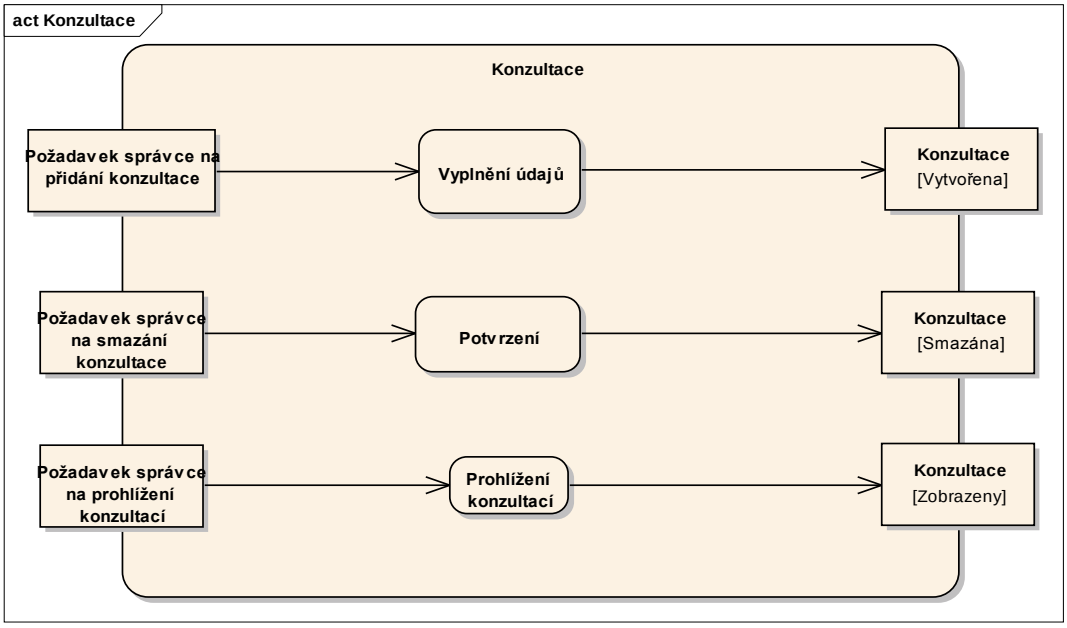
Emaily šablony



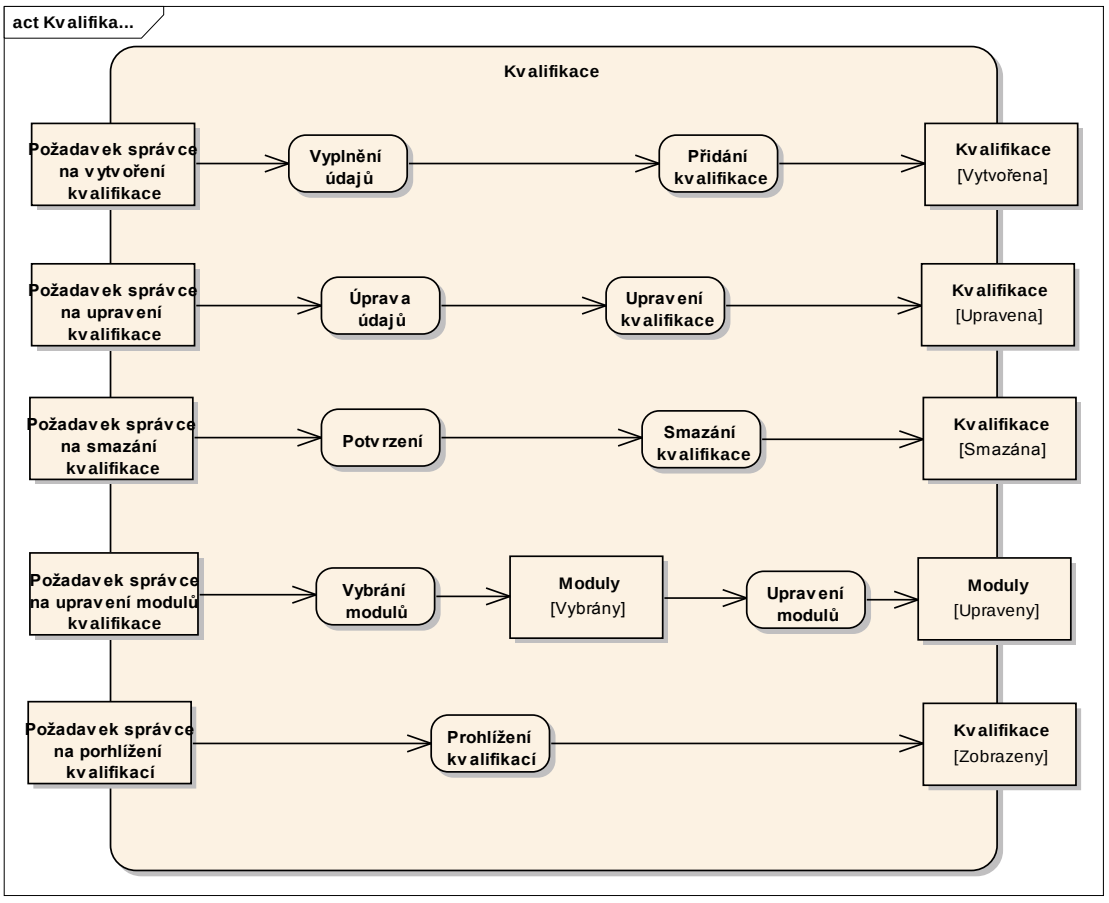
Home



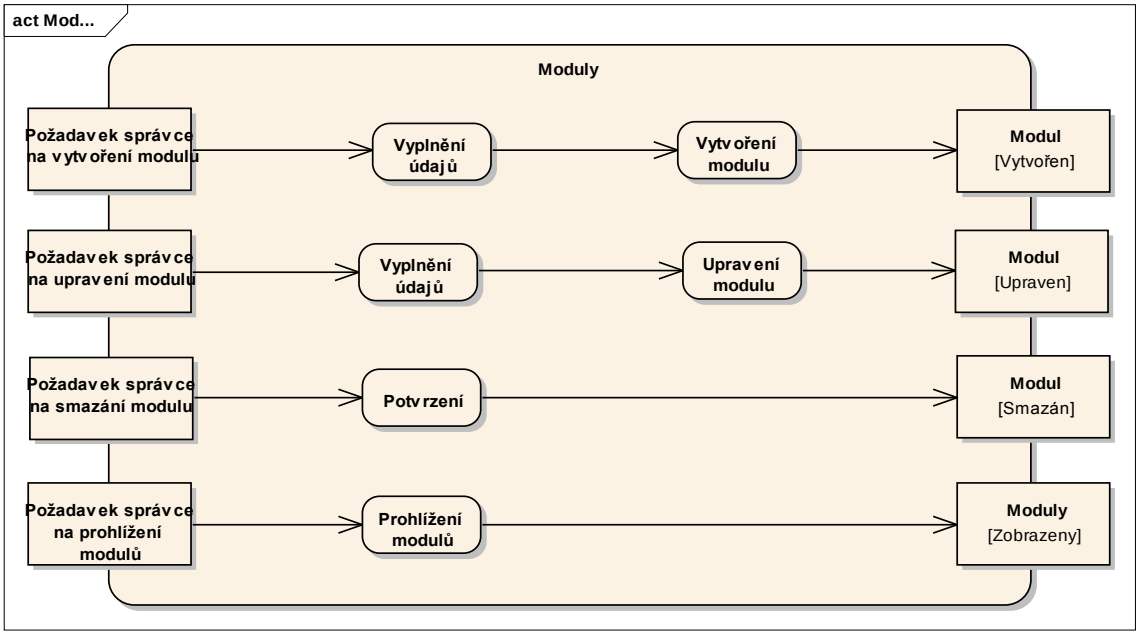
Konzultace



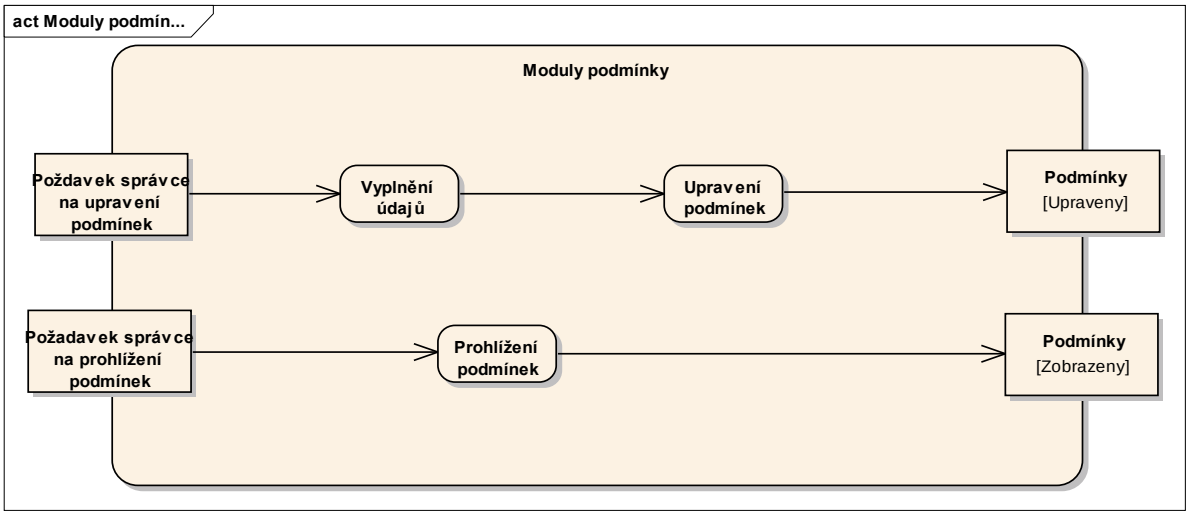
Kvalifikace



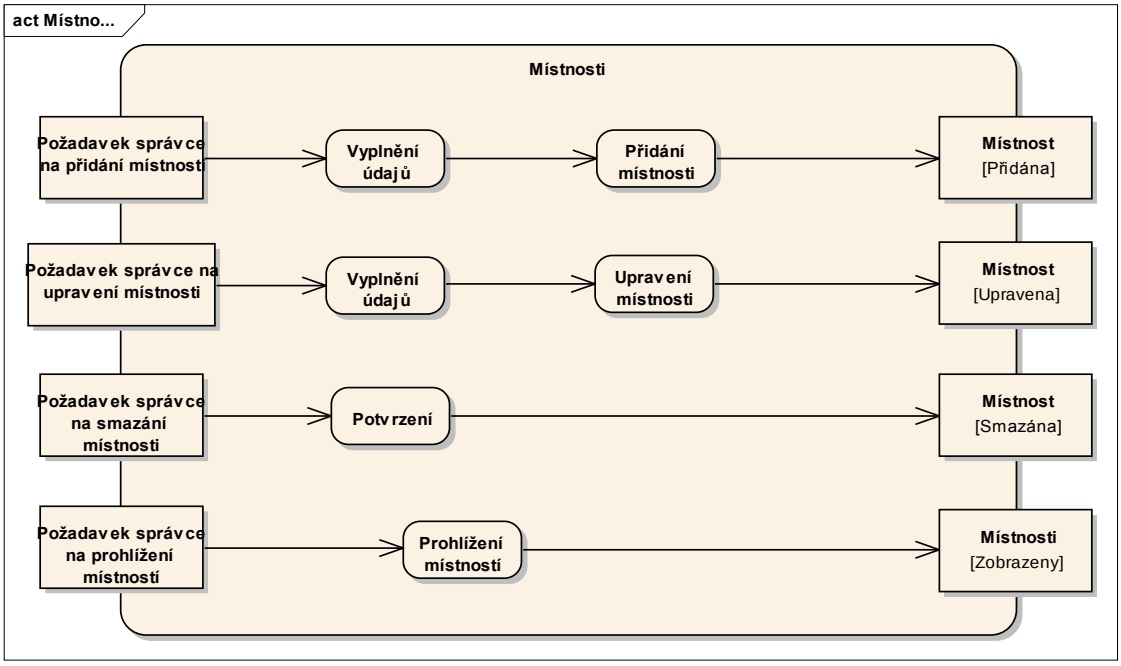
Moduly



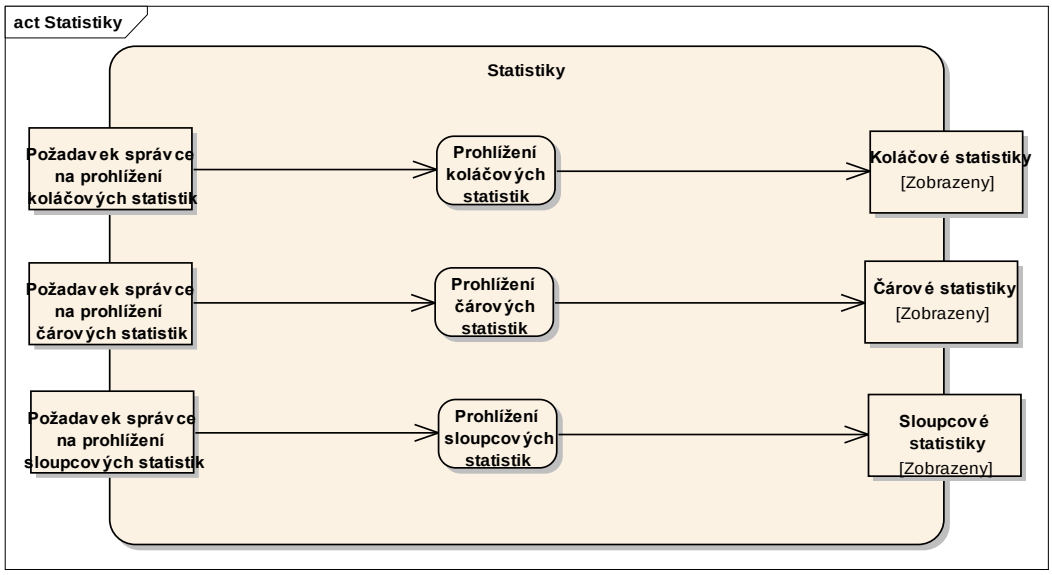
Moduly podmínky



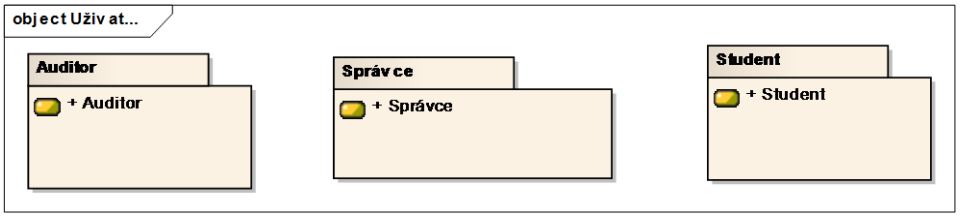
Místnosti



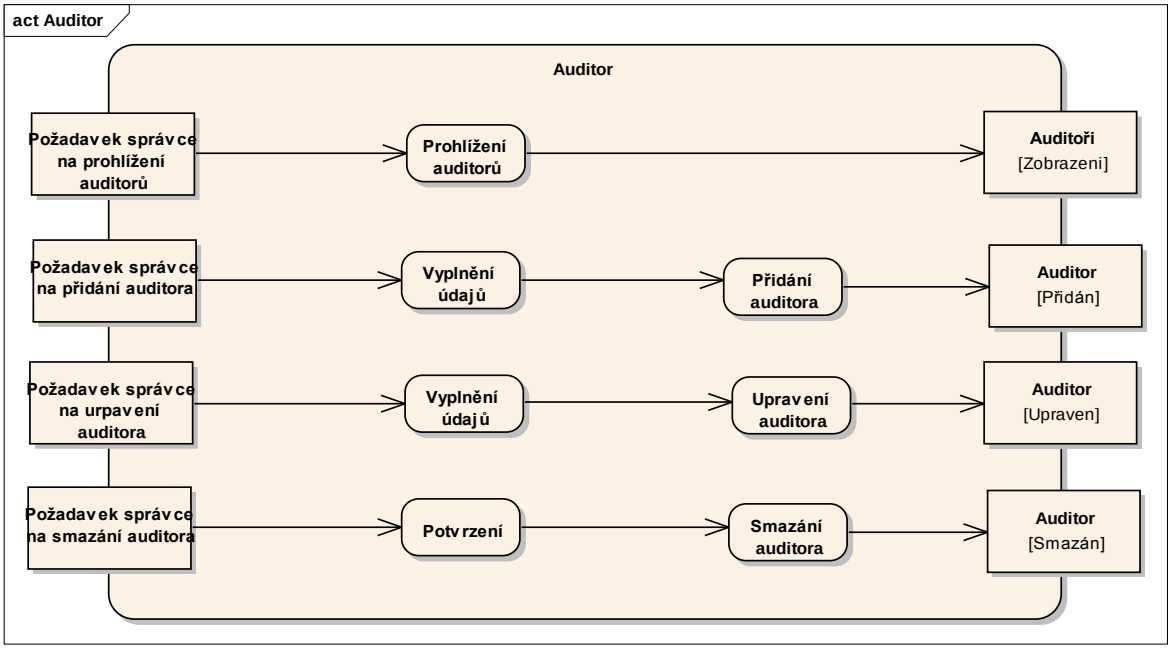
Statistiky



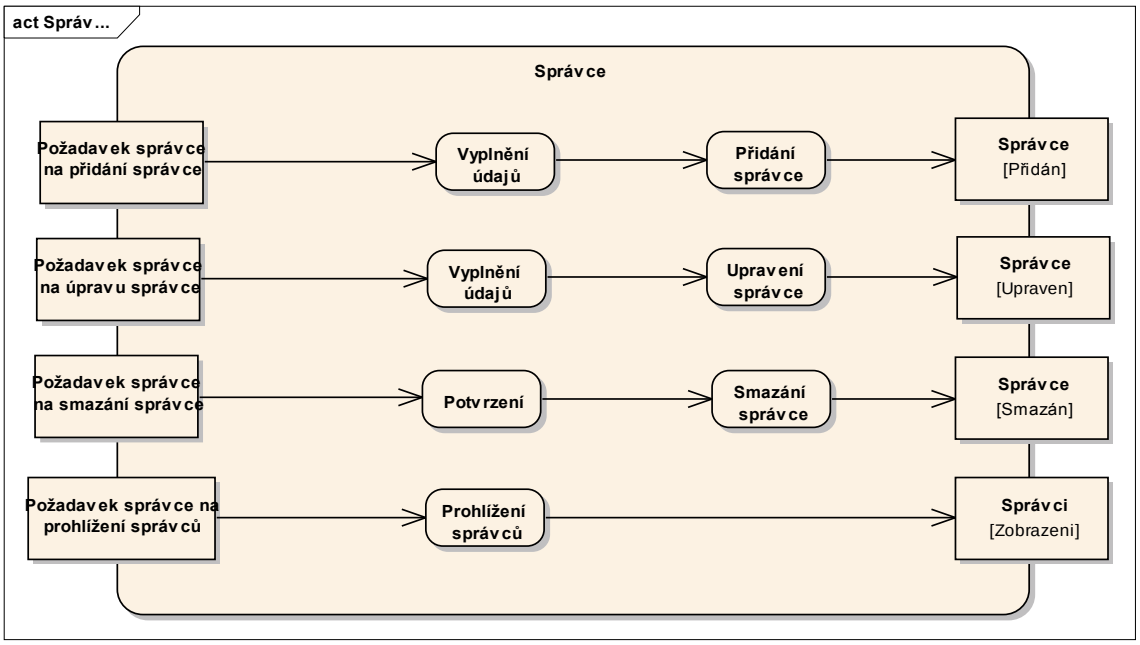
Uživatelé



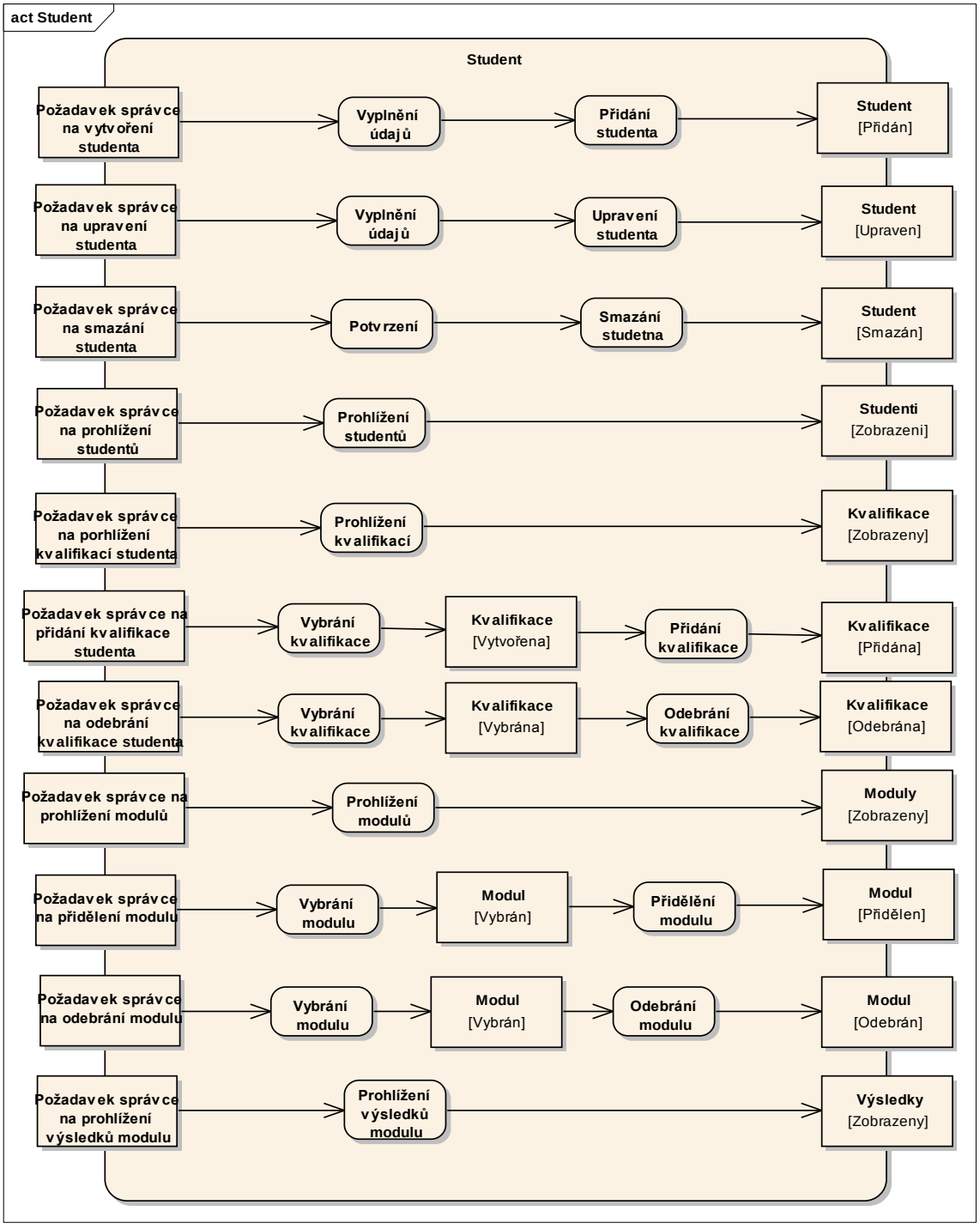
Auditor



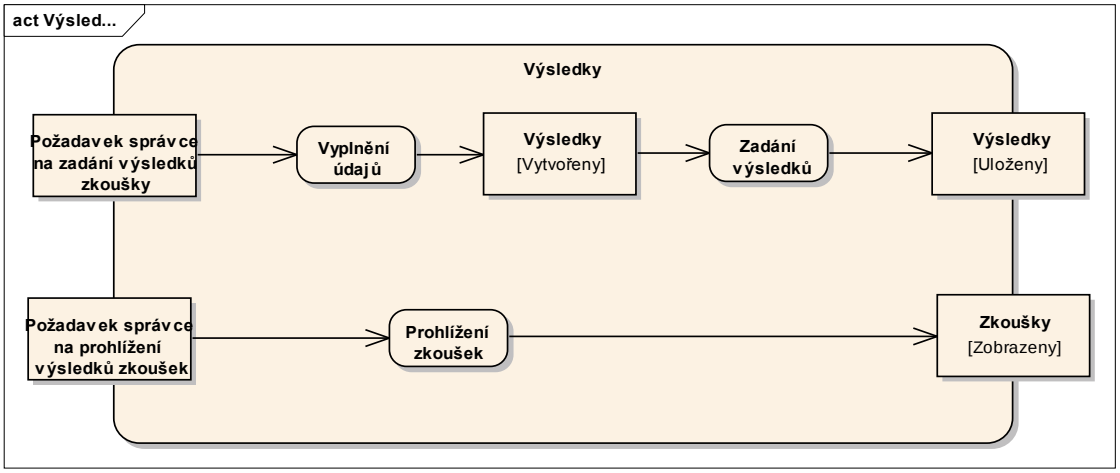
Správce



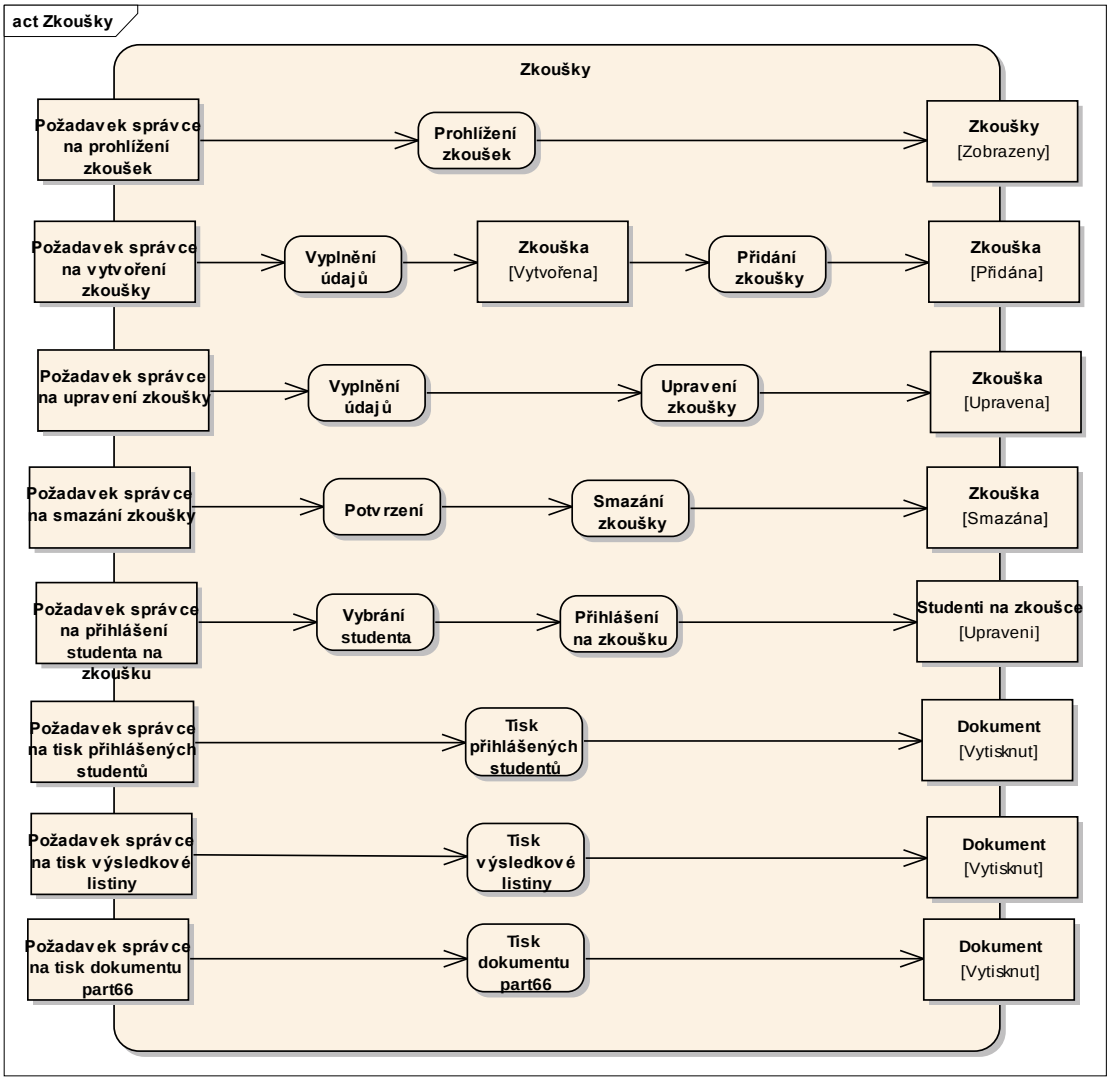
Student


act Student
Student
Požadavek správce na vytvoření studenta
Vyplnění údajů
Přidání studenta
Student [Přidán]
Požadavek správce na upravení studenta
Vyplnění údajů
Upravení studenta
Student [Upraven]
Požadavek správce na smazání studenta
Potvrzení
Smazání studetna
Student [Smazán]
Požadavek správce na prohlížení studentů
Prohlížení studentů
Studenti [Zobrazeni]
Požadavek správce na porhlížení kvalifikací studenta
Prohlížení kvalifikací
Kvalifikace [Zobrazeny]
Požadavek správce na přidání kvalifikace studenta
Vybrání kvalifikace
Kvalifikace [Vytvořena]
Přidání kvalifikace
Kvalifikace [Přidána]
Požadavek správce na odebrání kvalifikace studenta
Vybrání kvalifikace
Kvalifikace [Vybrána]
Odebrání kvalifikace
Kvalifikace [Odebrána]
Požadavek správce na prohlížení modulů
Prohlížení modulů
Moduly [Zobrazeny]
Požadavek správce na přidělení modulu
Vybrání modulu
Modul [Vybrán]
Přidělění modulu
Modul [Přidělen]
Požadavek správce na odebrání modulu
Vybrání modulu
Modul [Vybrán]
Odebrání modulu
Modul [Odebrán]
Požadavek správce na prohlížení výsledků modulu
Prohlížení výsledků modulu
Výsledky [Zobrazeny]

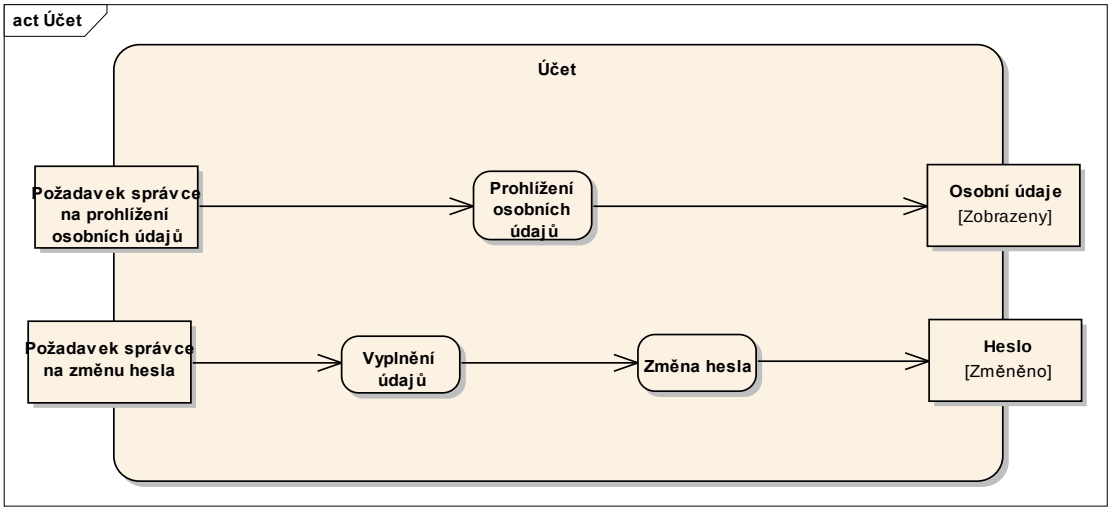
Výsledky



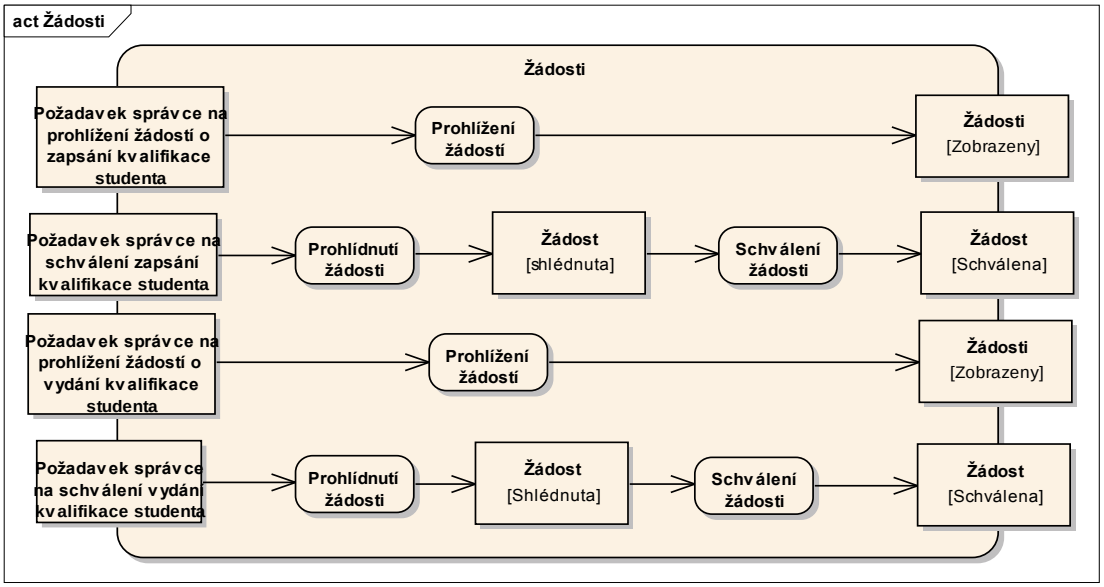
Zkoušky



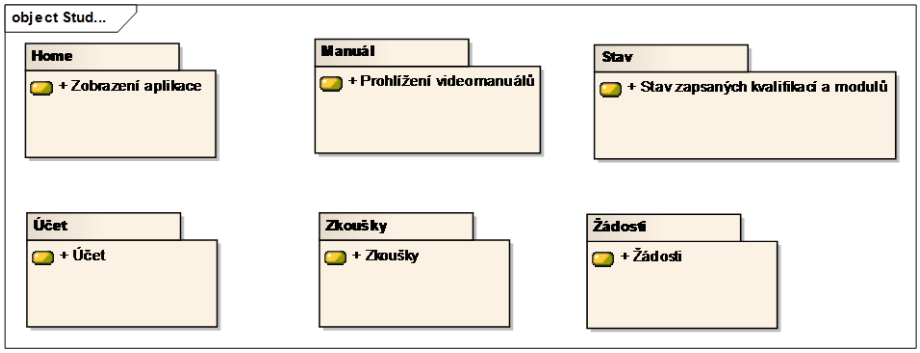
Účet



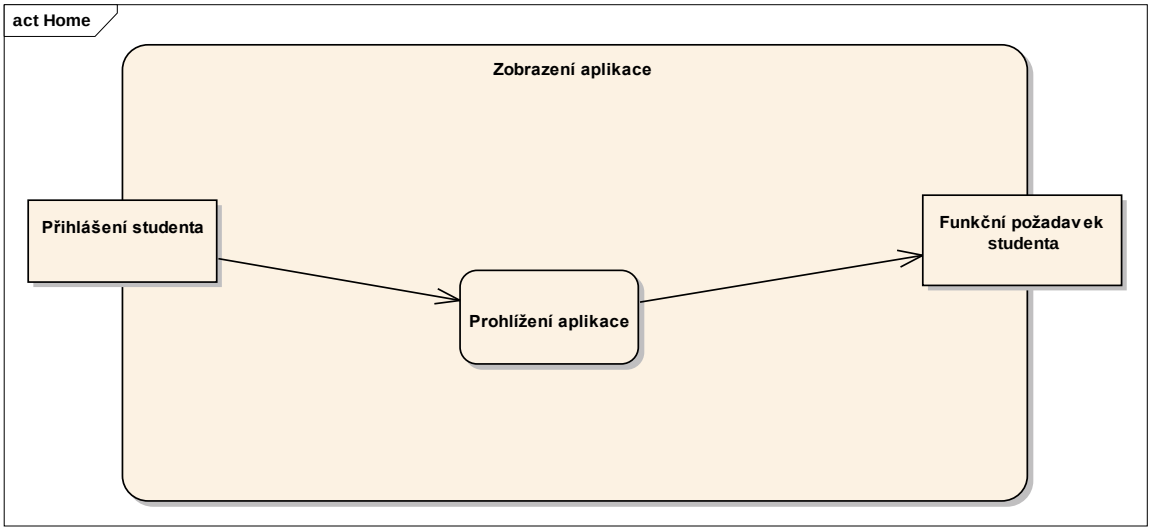
Žádosti



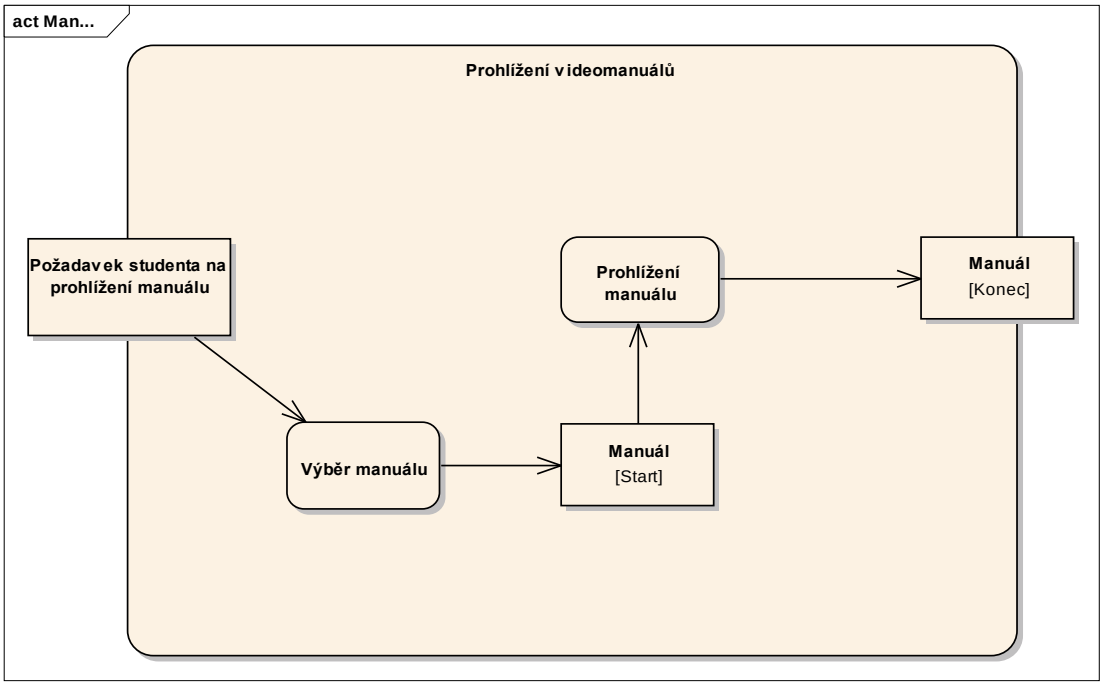
Student



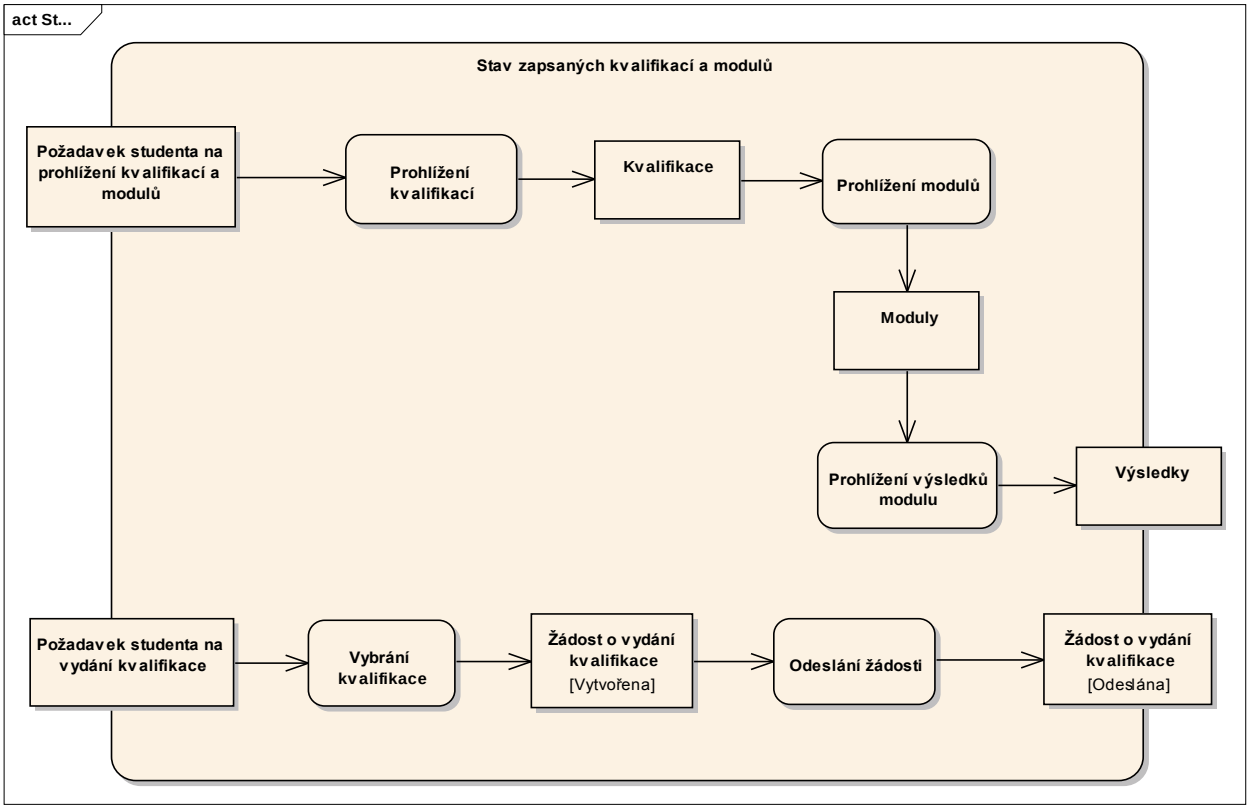
Home


act Home
Zobrazení aplikace
Přihlášení studenta
Prohlížení aplikace
Funkční požadavek studenta

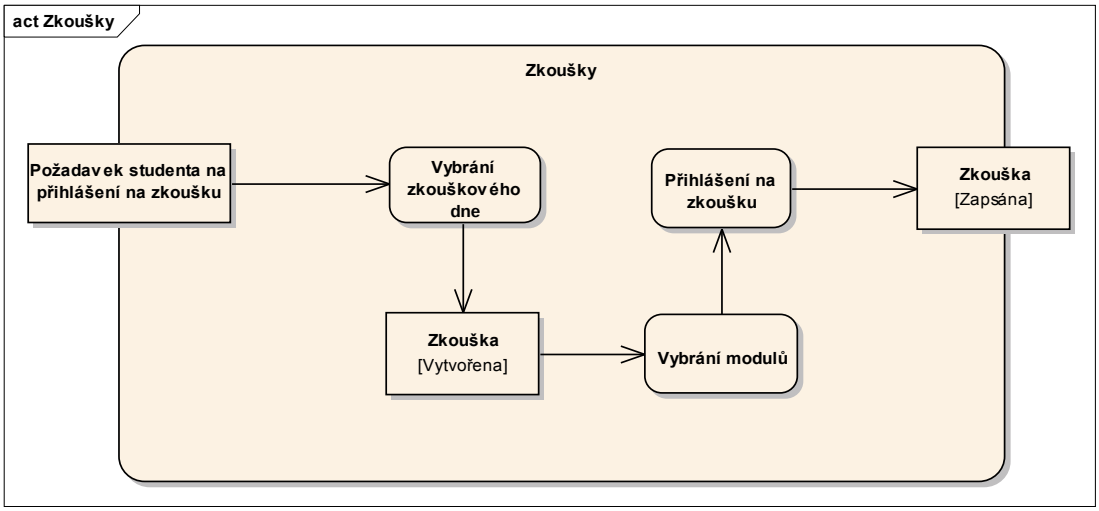
Manuál


act Man...
Prohlížení videomanuálů
Požadavek studenta na prohlížení manuálu
Výběr manuálu
Manuál
[Start]
Prohlížení manuálu
Manuál
[Konec]

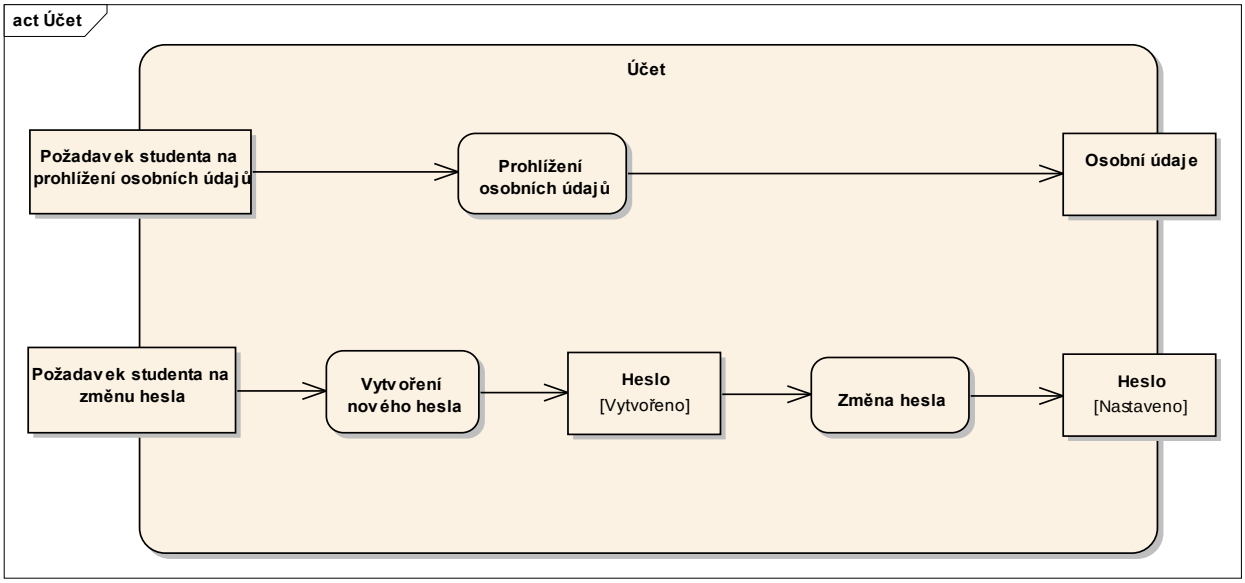
Stav



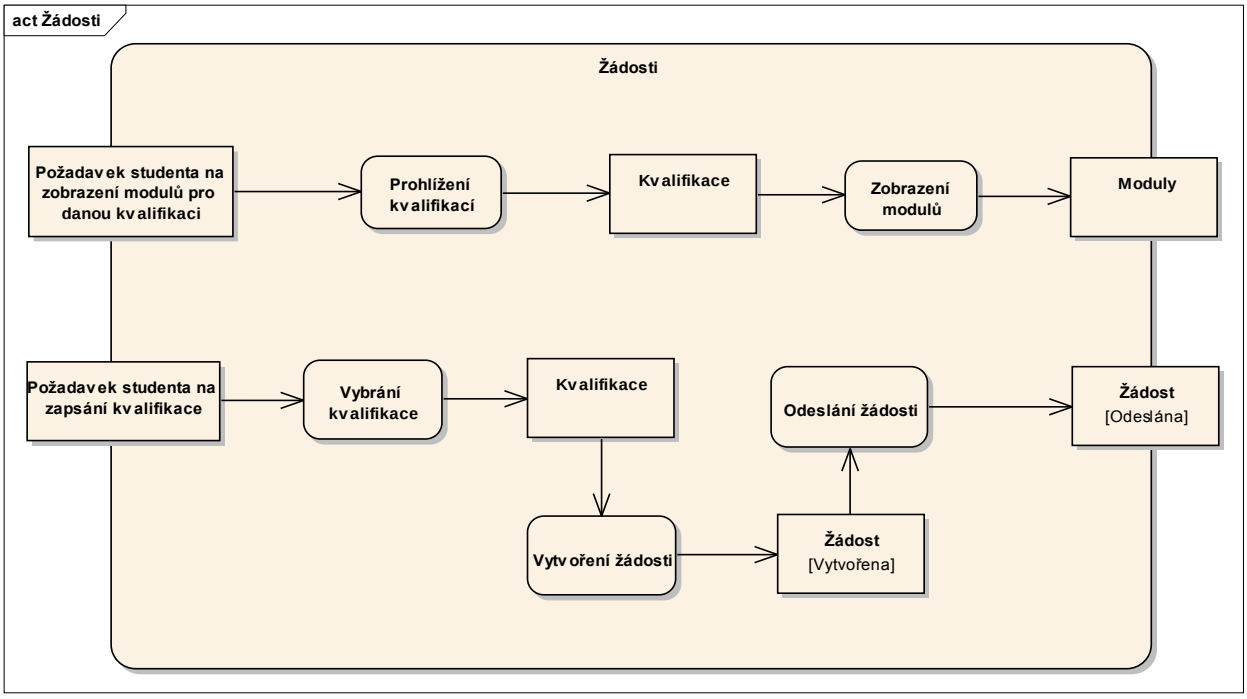
Zkoušky



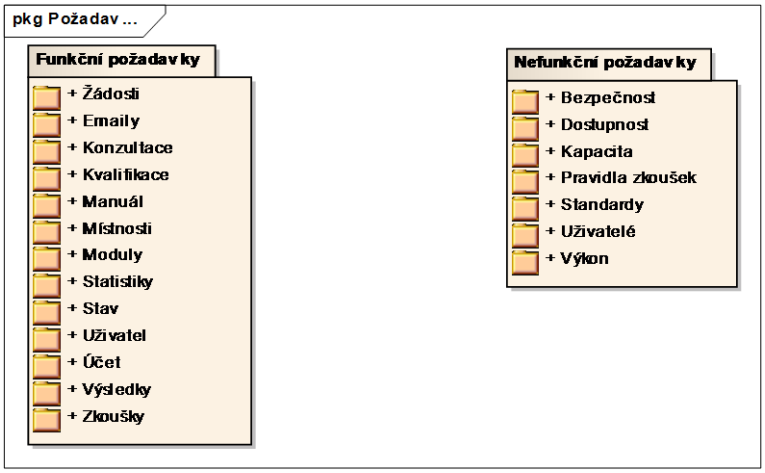
Účet



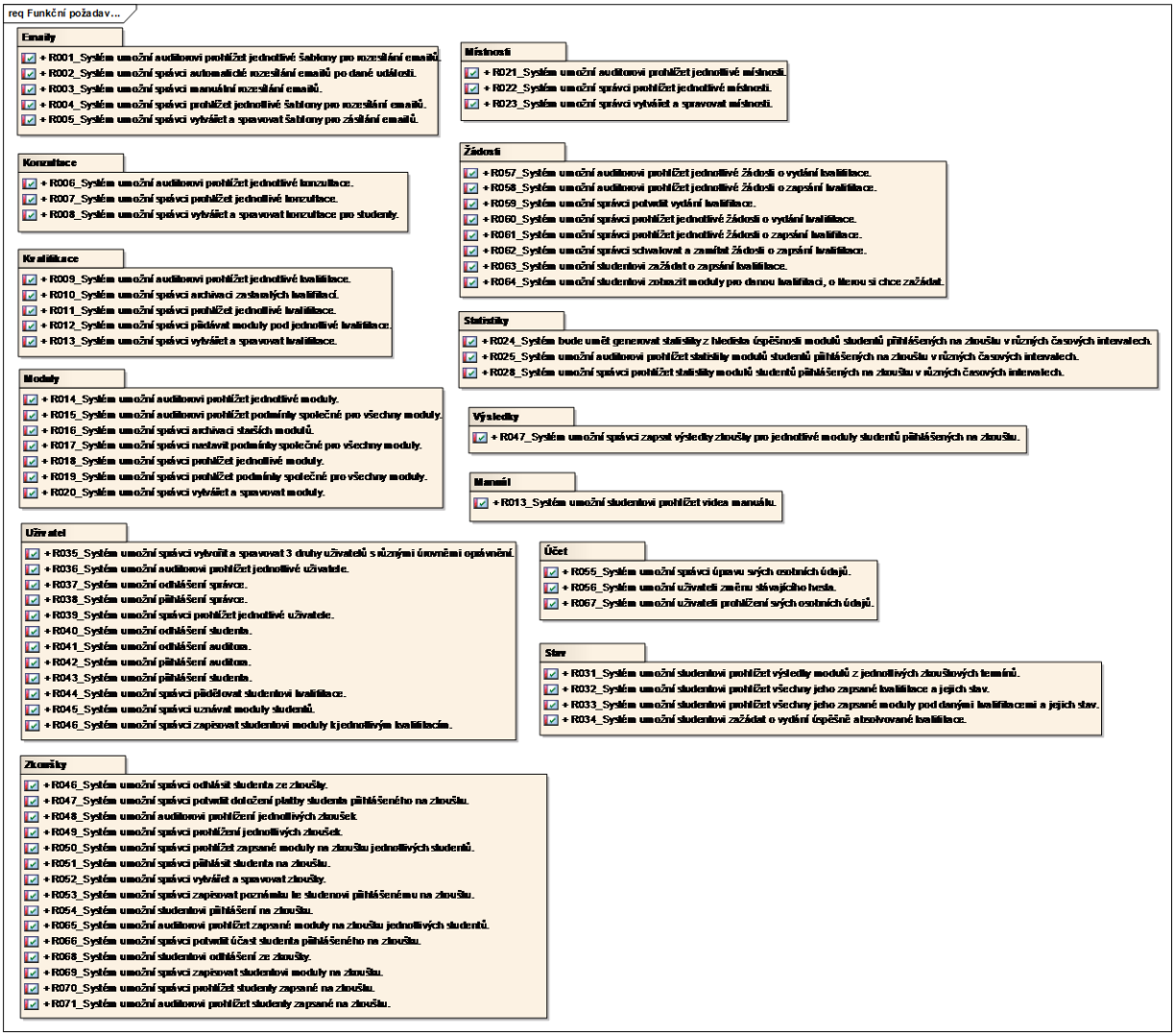
Žádosti



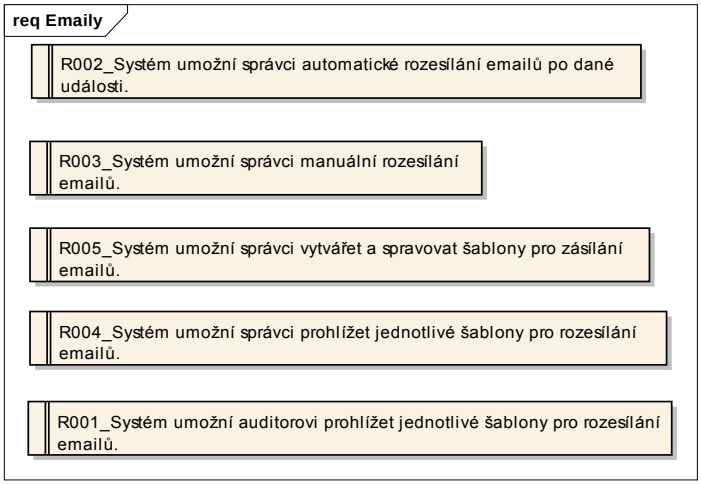
PŘÍLOHA F: Diagramy požadavků



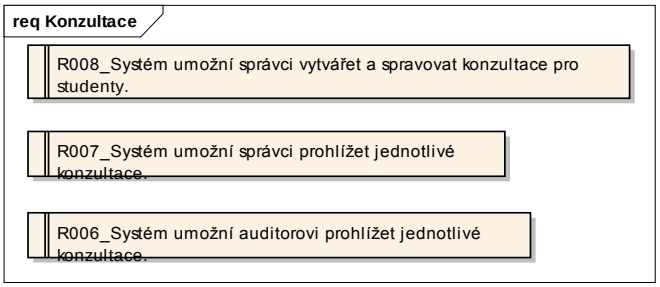
Funkční požadavky



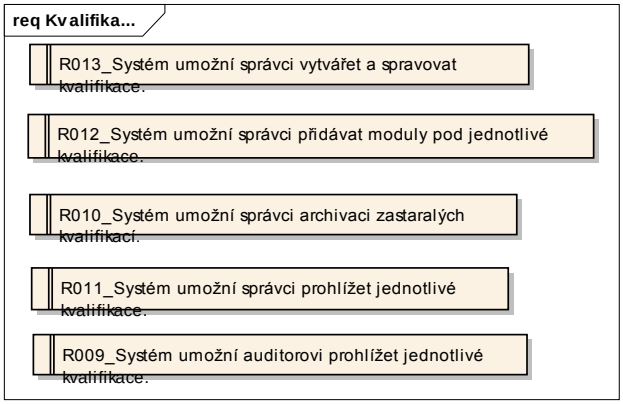
Emaily



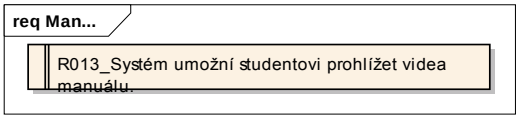
Konzultace



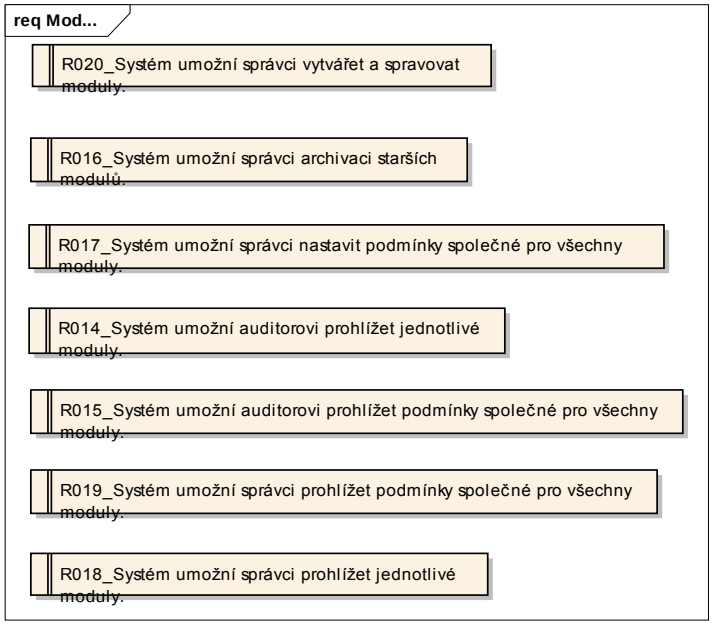
Kvalifikace



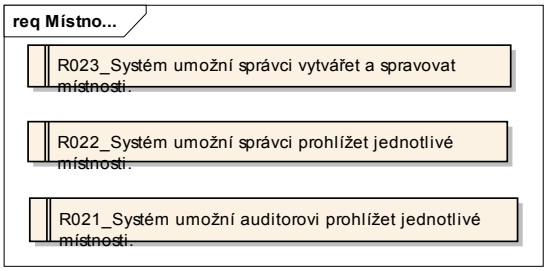
Manuál



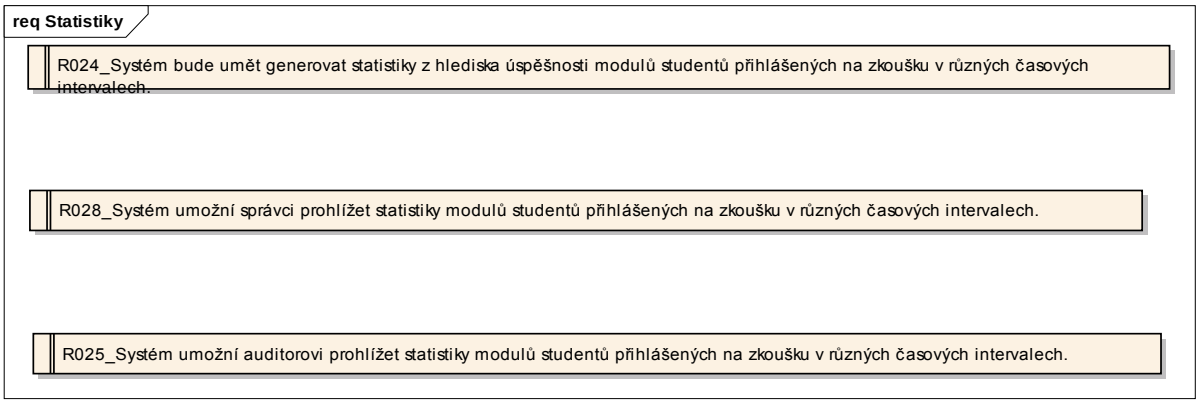
Moduly



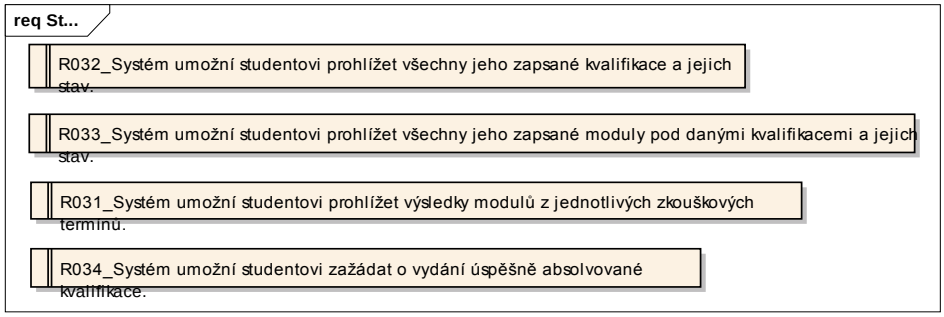
Místnosti



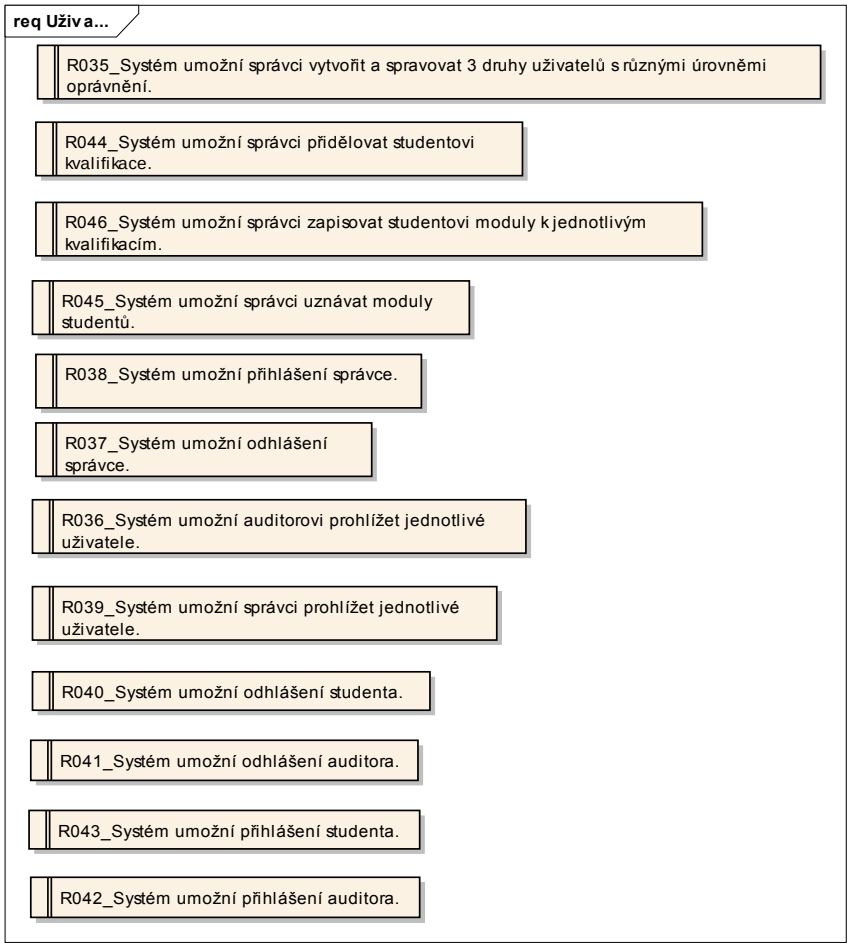
Statistiky



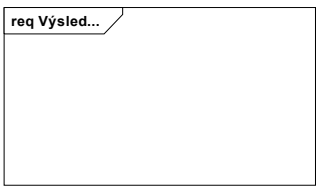
Stav



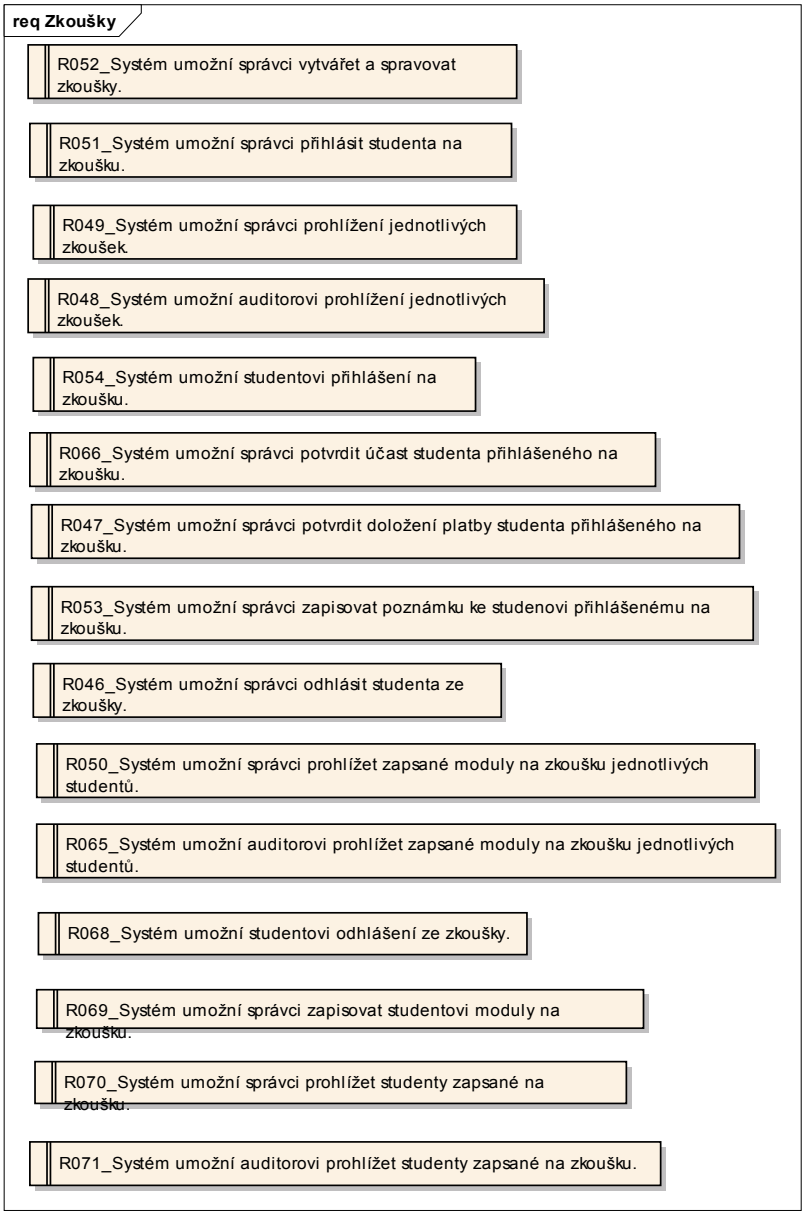
Uživatel



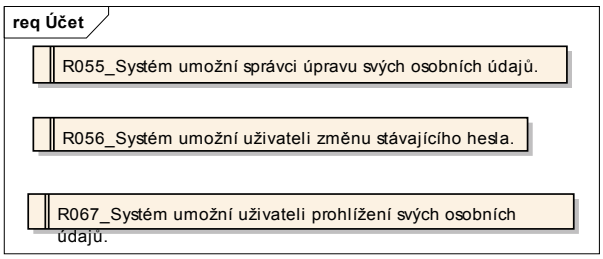
Výsledky



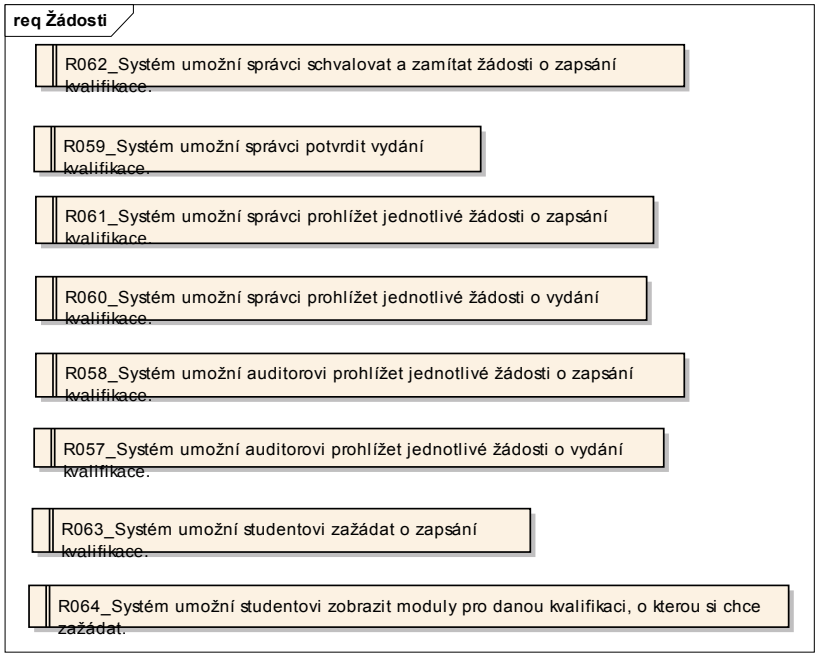
Zkoušky



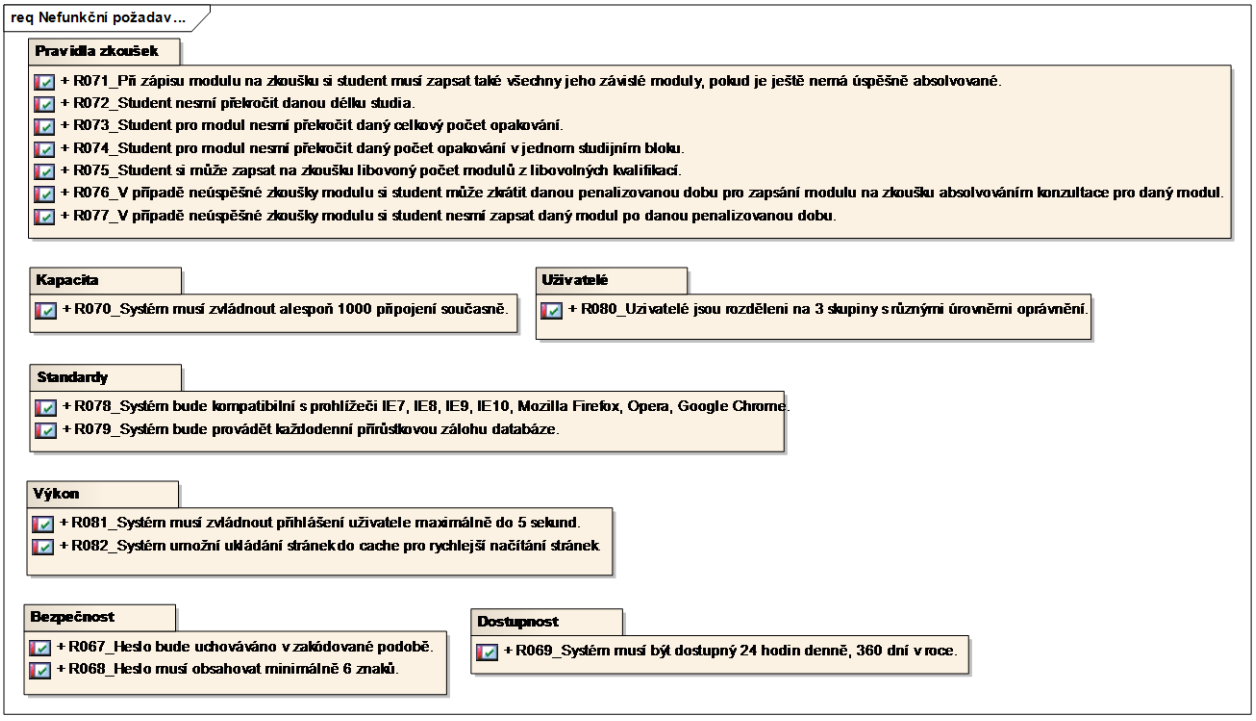
Účet



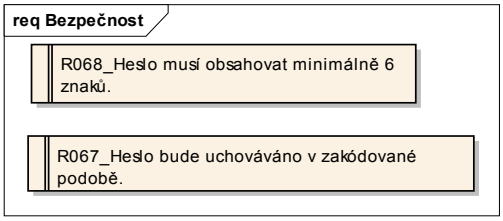
Žádosti



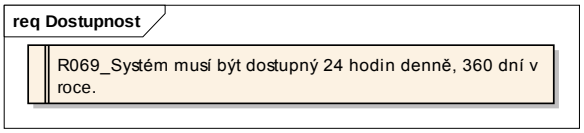
Nefunkční požadavky



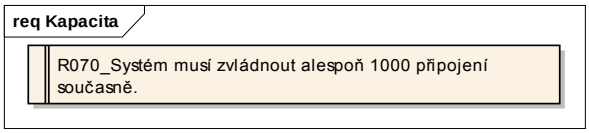
Bezpečnost



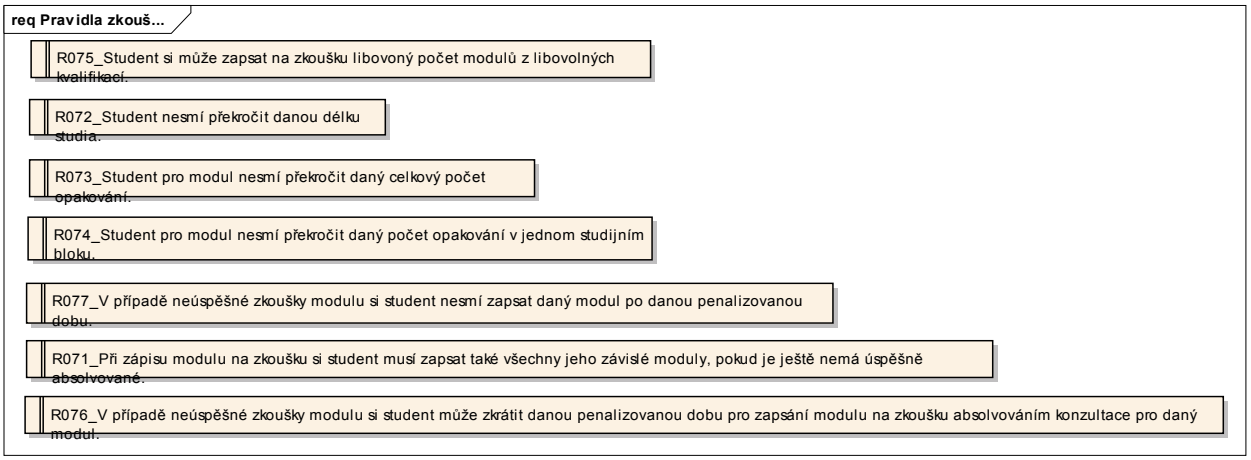
Dostupnost



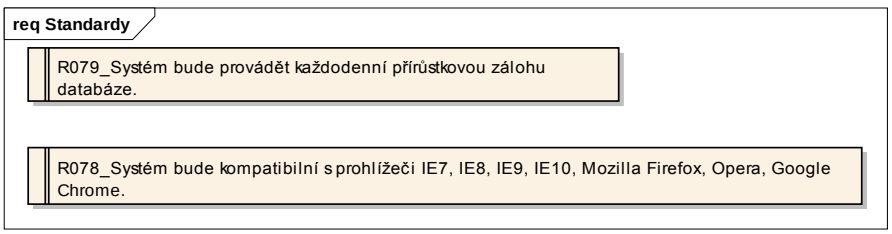
Kapacita



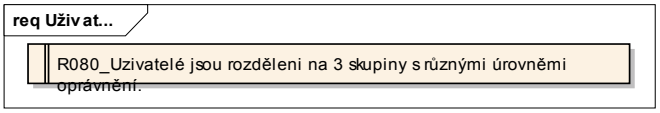
Pravidla zkoušek



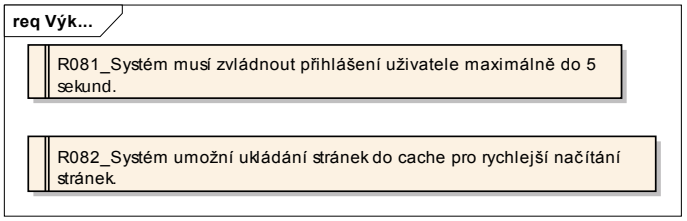
Standardy



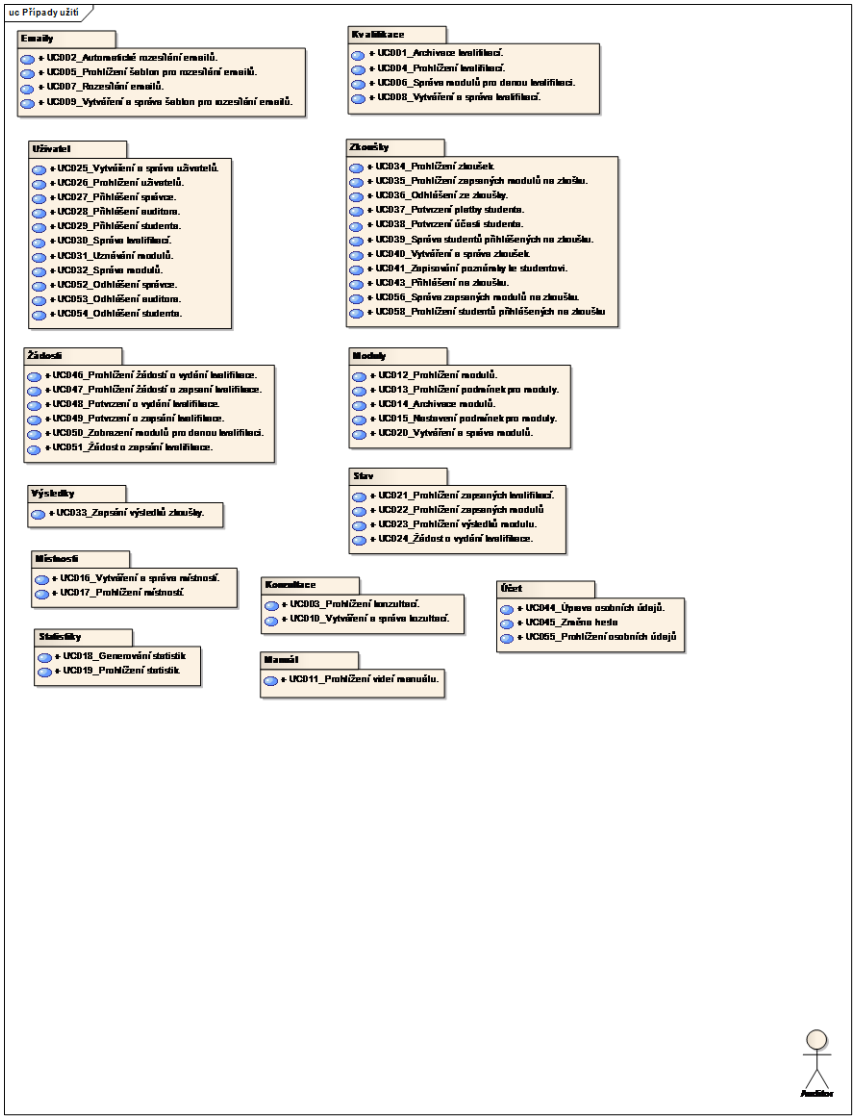
Uživatelé



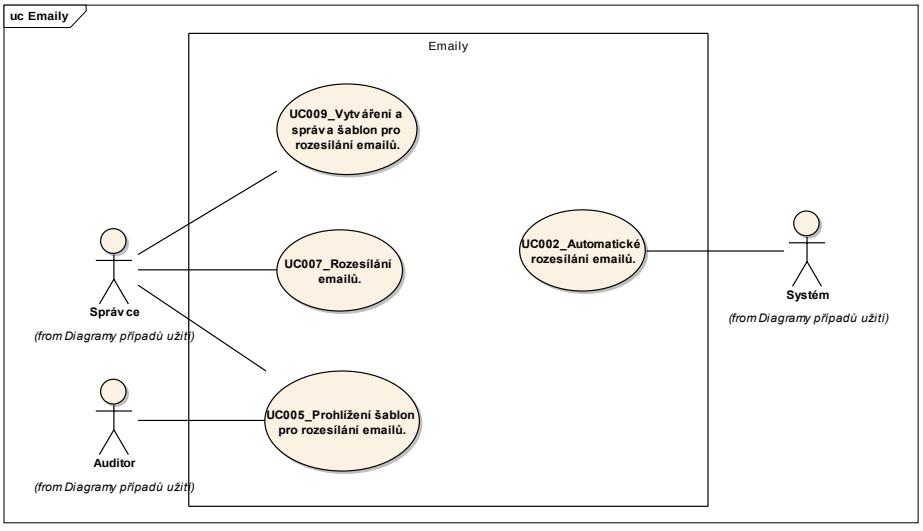
Výkon



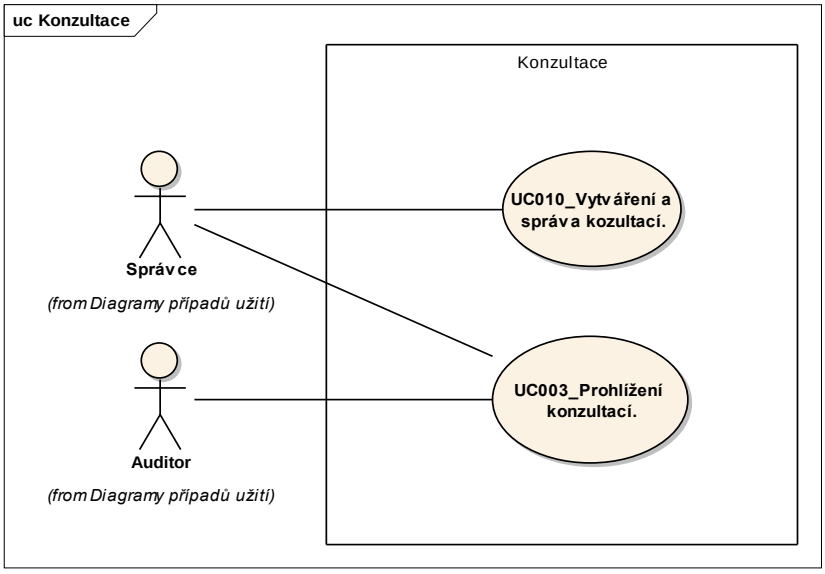
PŘÍLOHA F: Diagramy případů užití



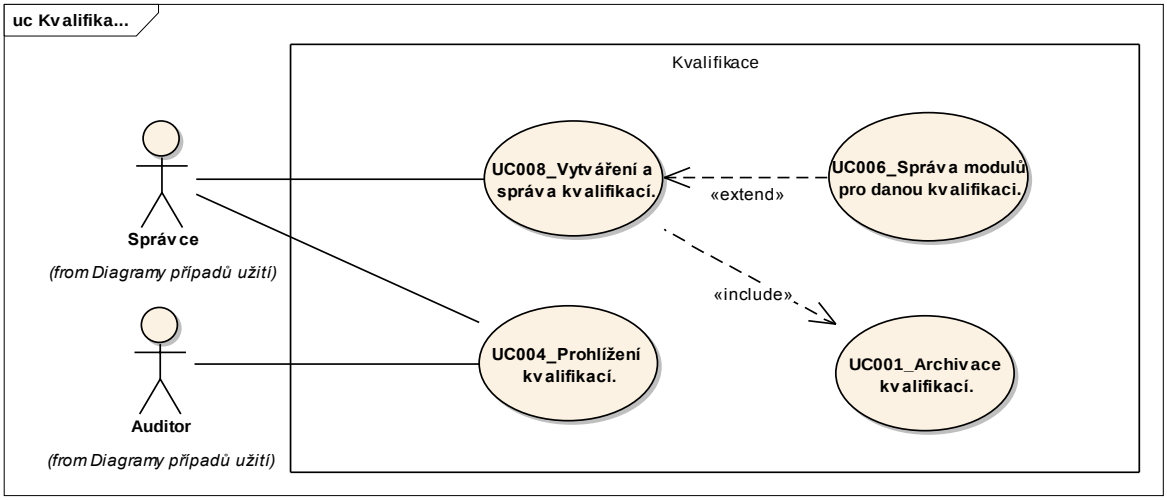
Emaily



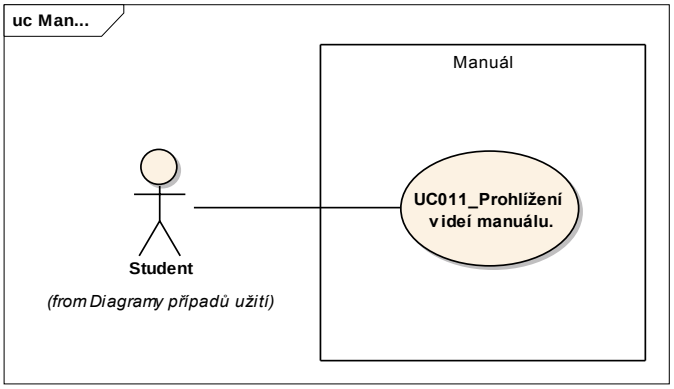
Konzultace



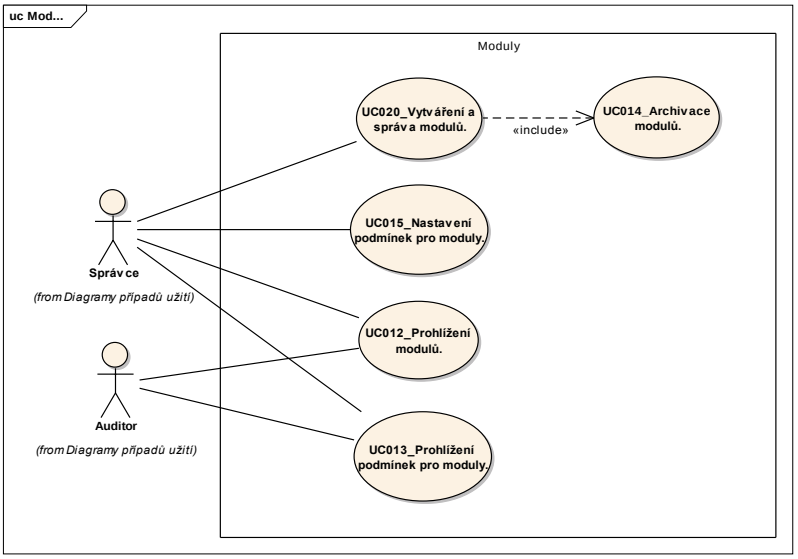
Kvalifikace



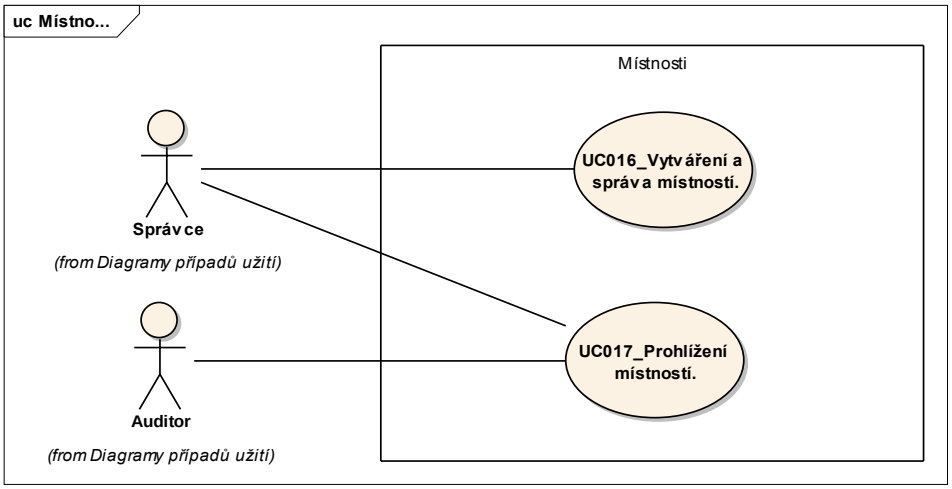
Manuál



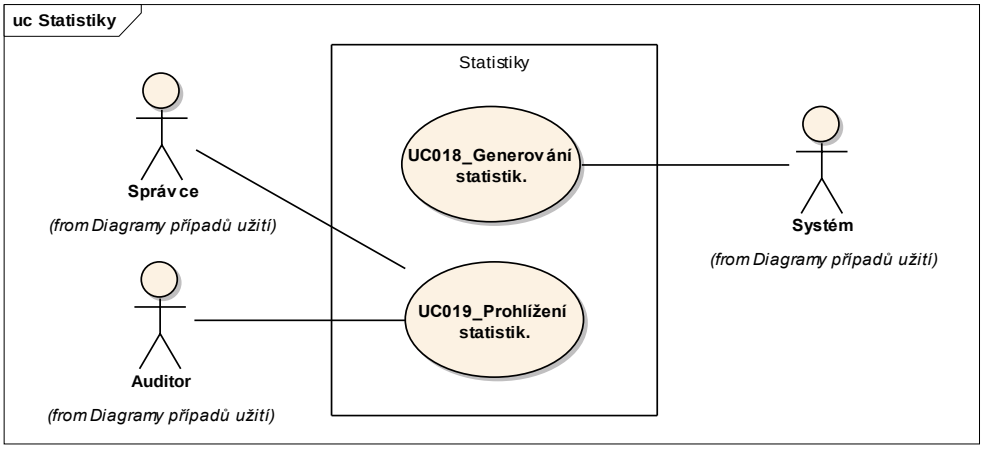
Moduly



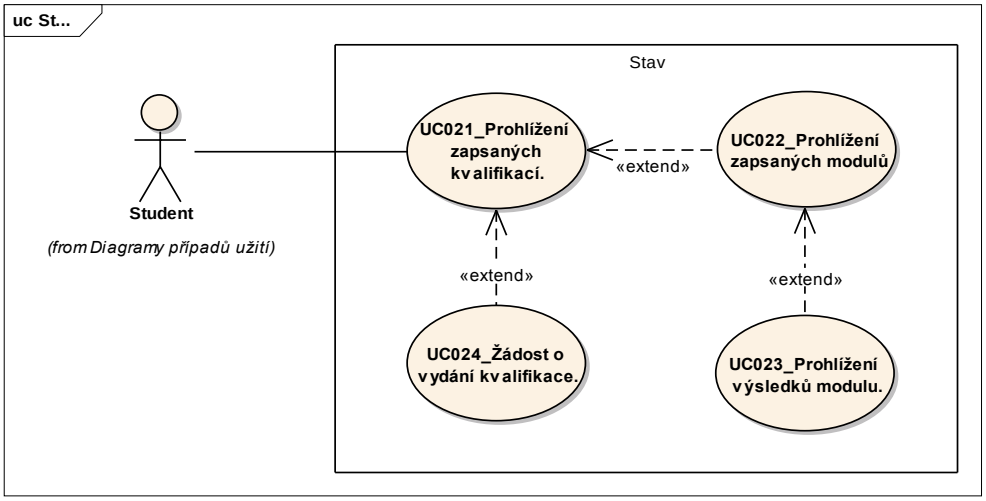
Místnosti



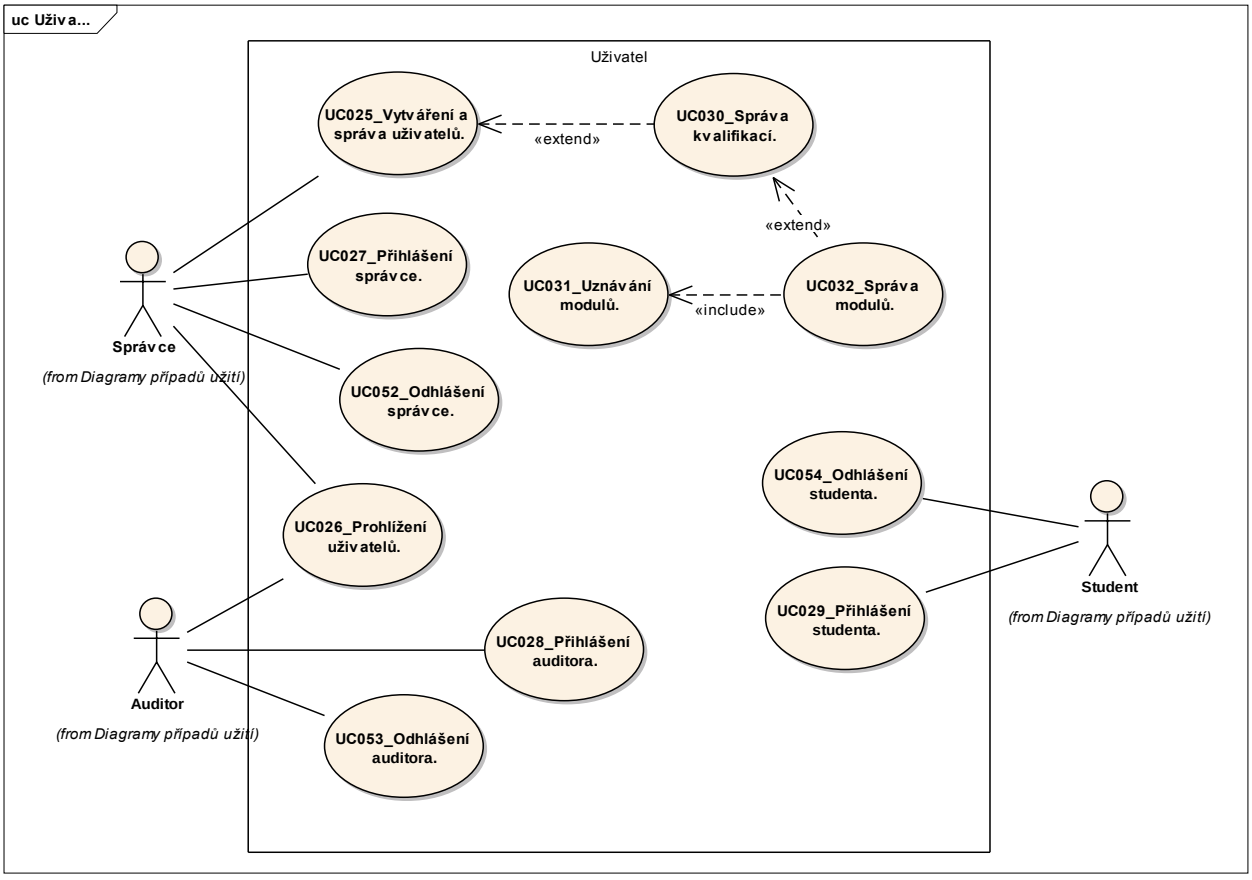
Statistiky



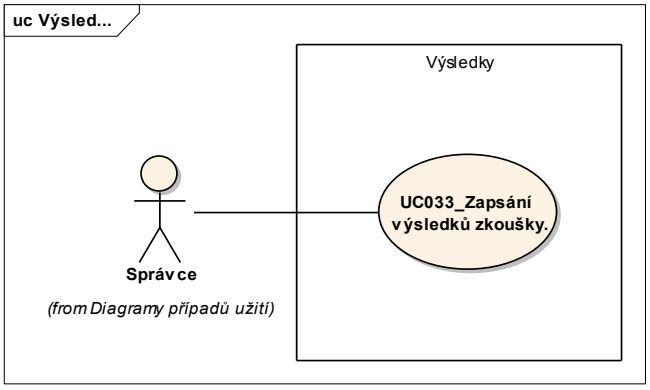
Stav



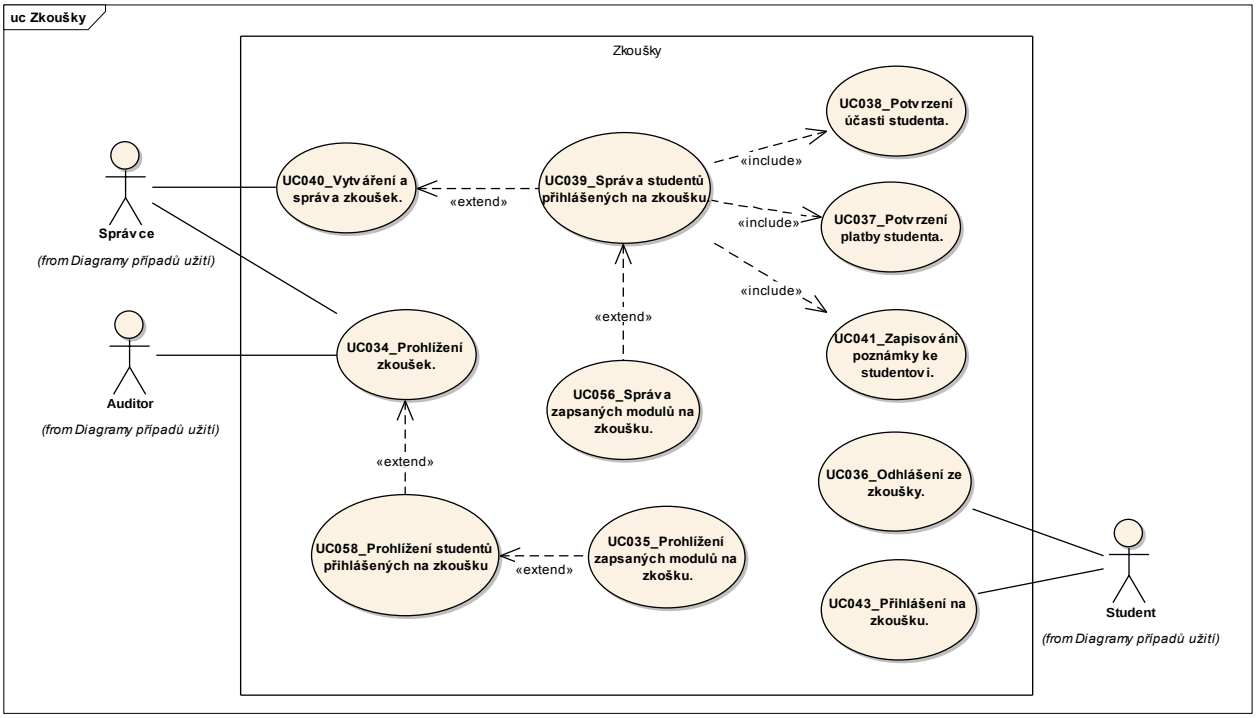
Uživatel



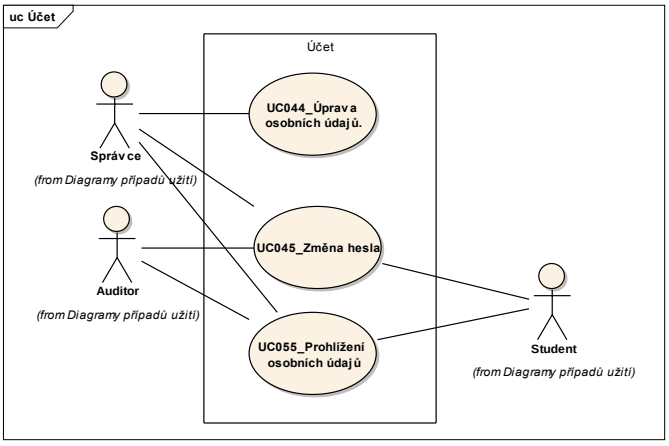
Výsledky



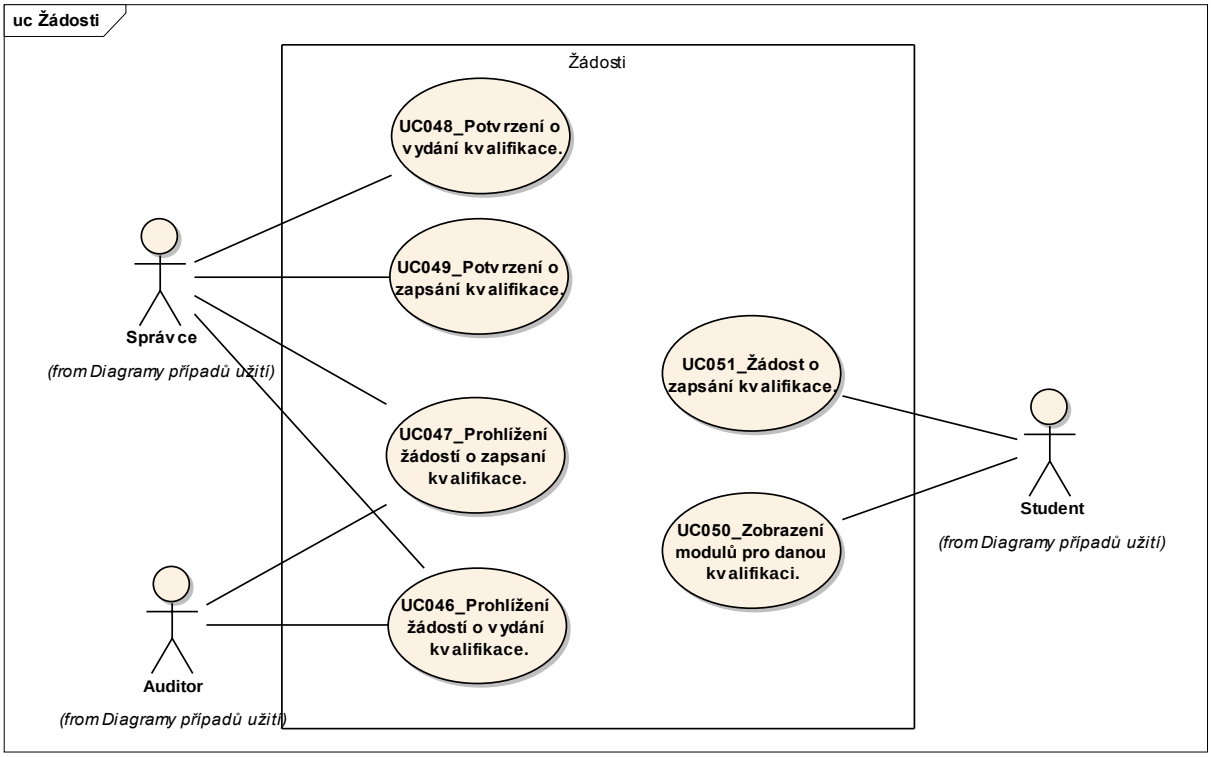
Zkoušky



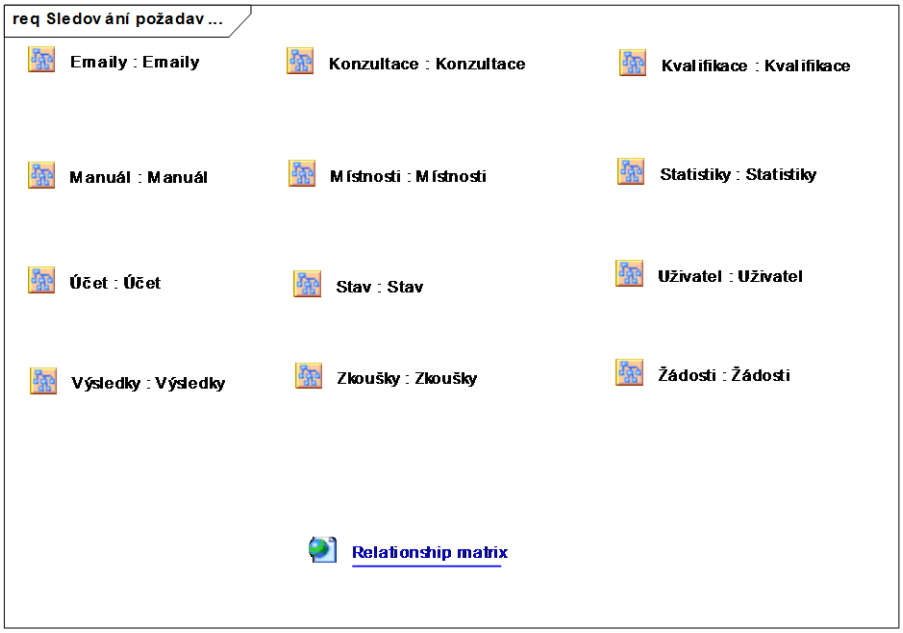
Účet



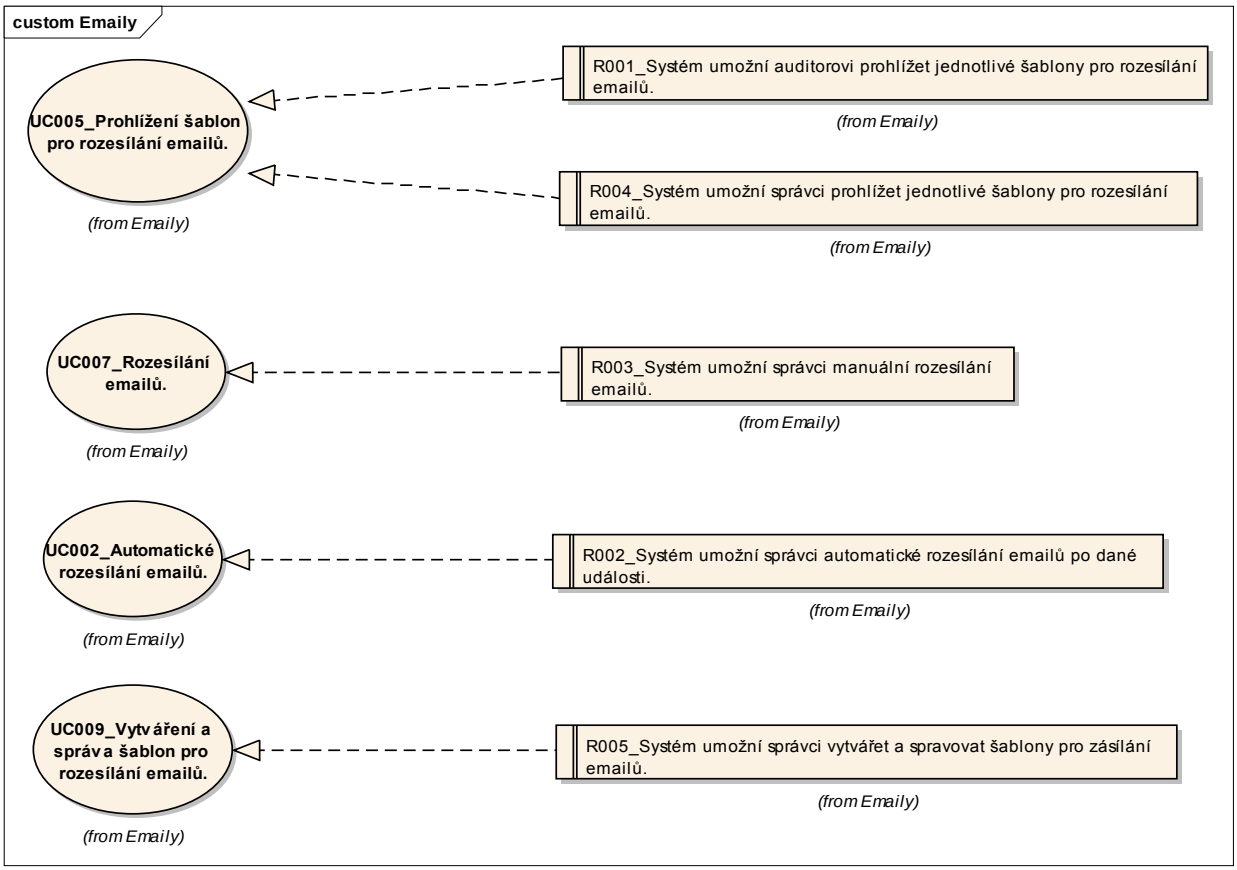
Žádosti


uc Žádosti
Žádosti
UC048_Potvrzení o vydání kvalifikace.
UC049_Potvrzení o zapsání kvalifikace.
UC047_Prohlížení žádostí o zapsaní kvalifikace.
UC046_Prohlížení žádostí o vydání kvalifikace.
UC051_Žádost o zapsání kvalifikace.
UC050_Zobrazení modulů pro danou kvalifikaci.
Správce
(from Diagramy případů užití)
Auditor
(from Diagramy případů užití)
Student
(from Diagramy případů užití)

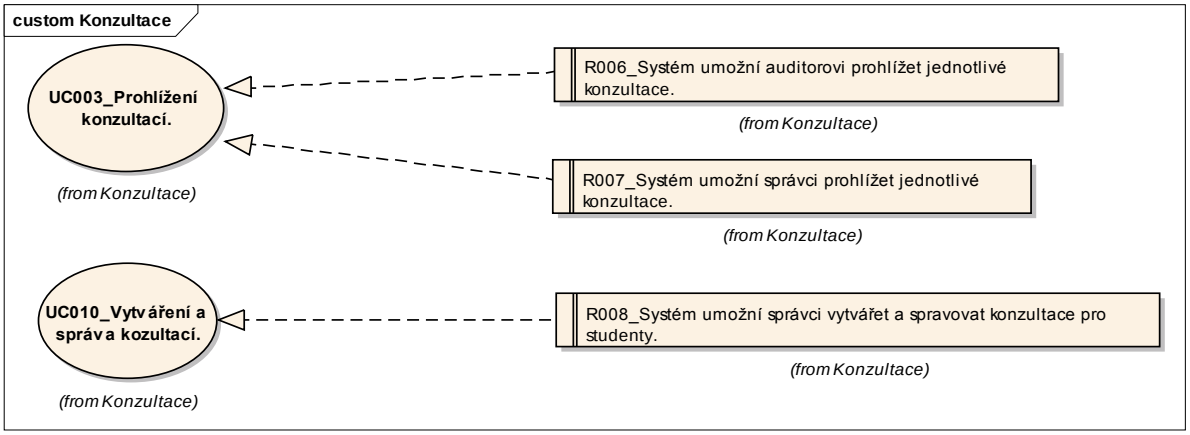
PŘÍLOHA G: Diagramy sledování požadavků



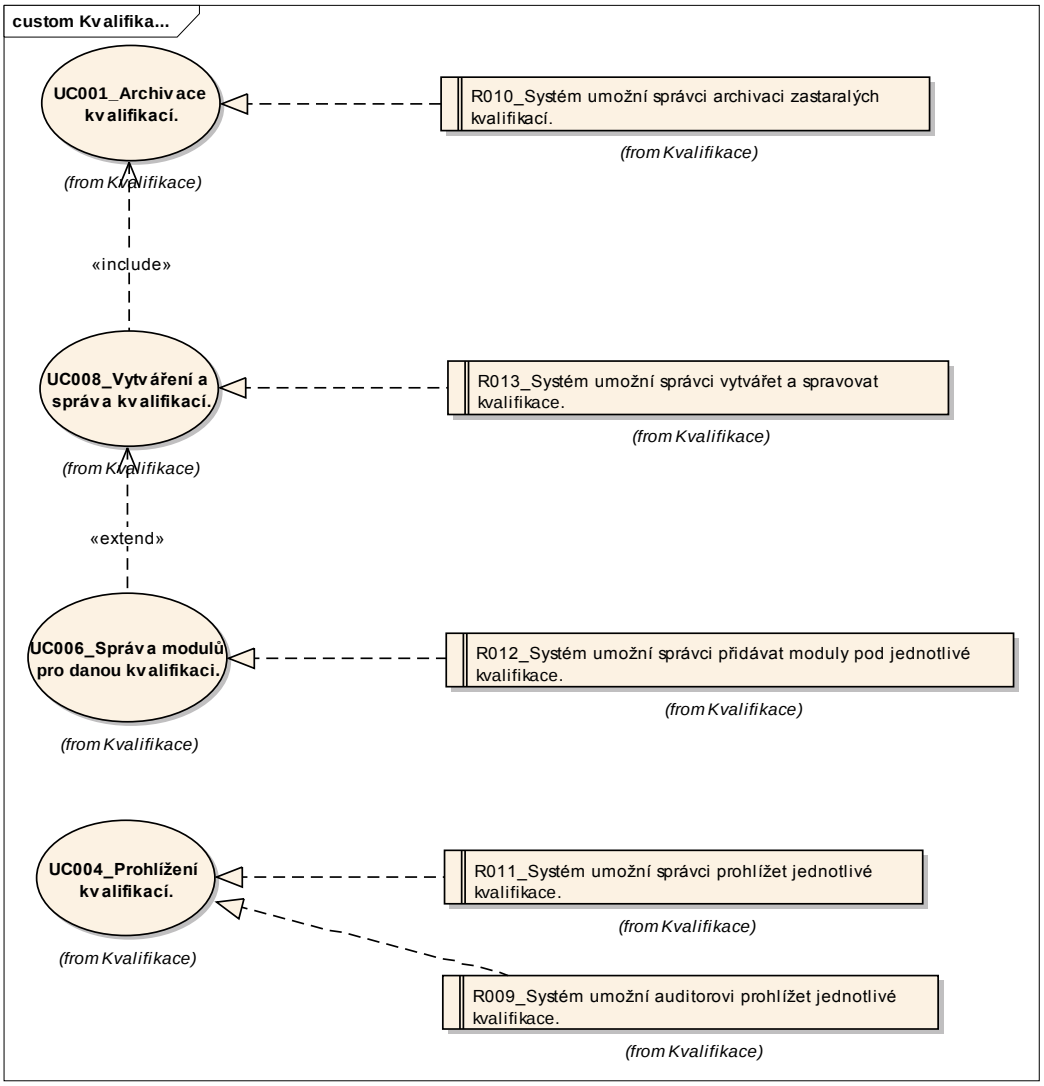
Emaily



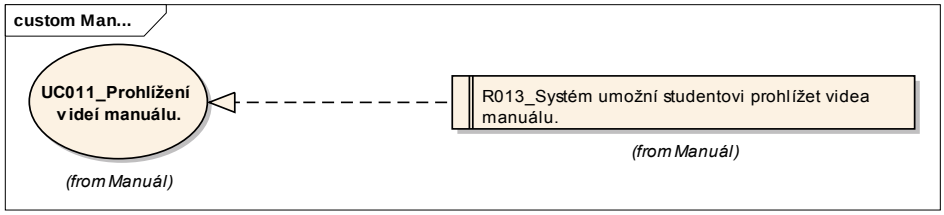
Konzultace



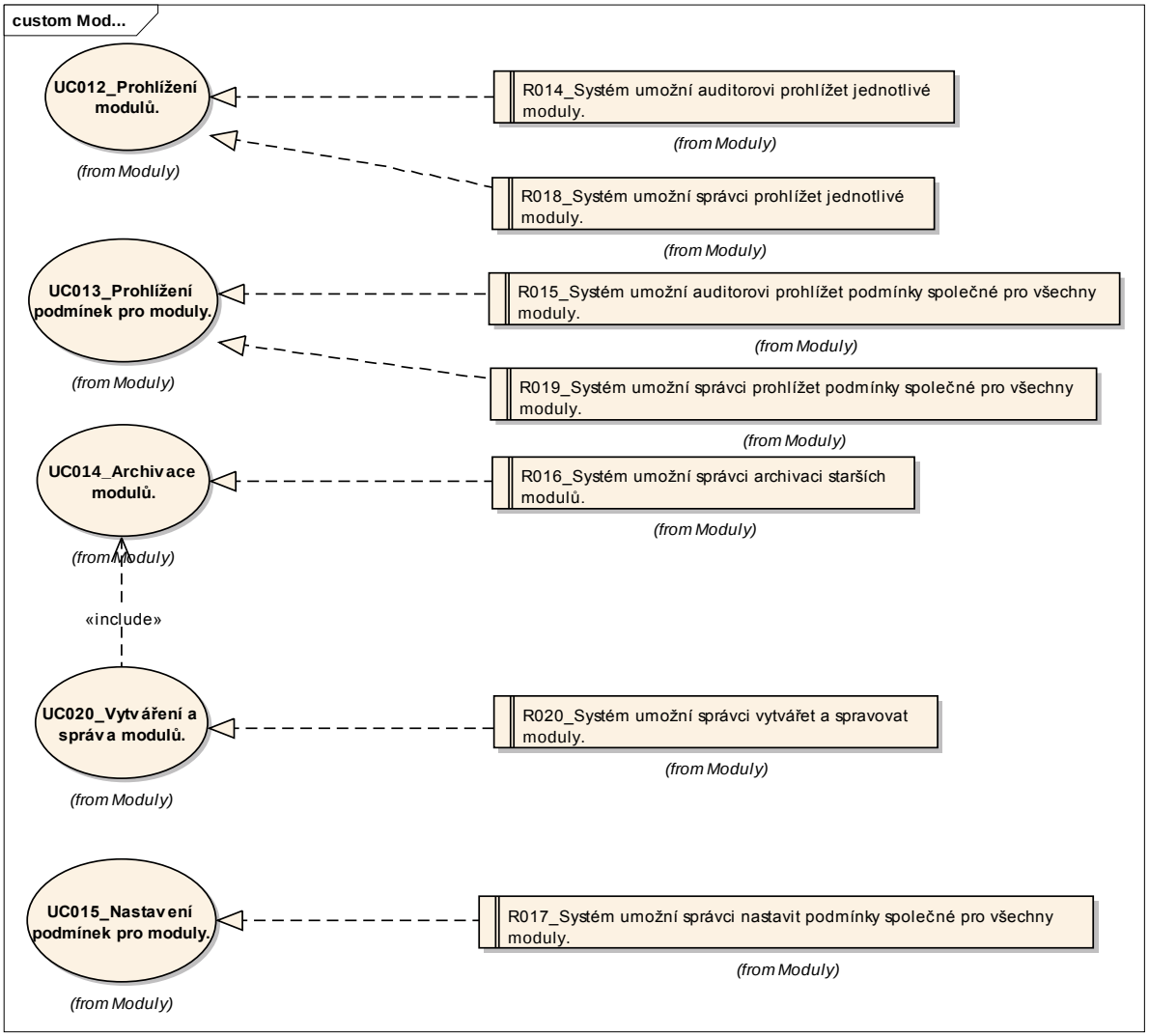
Kvalifikace



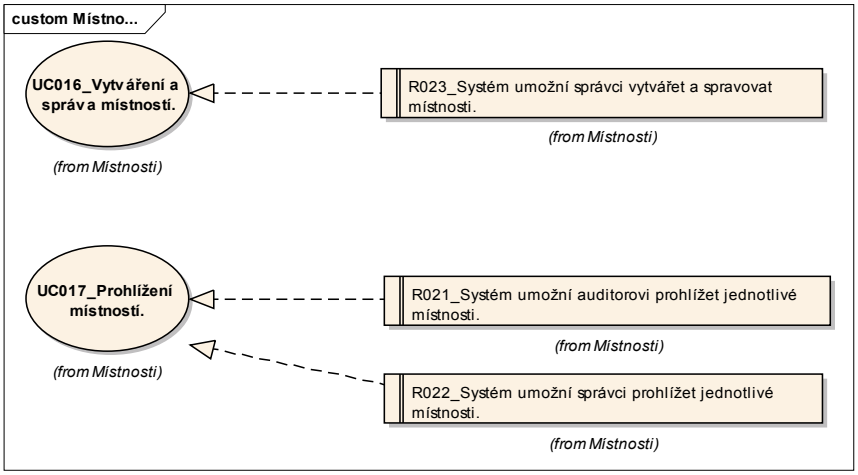
Manuál



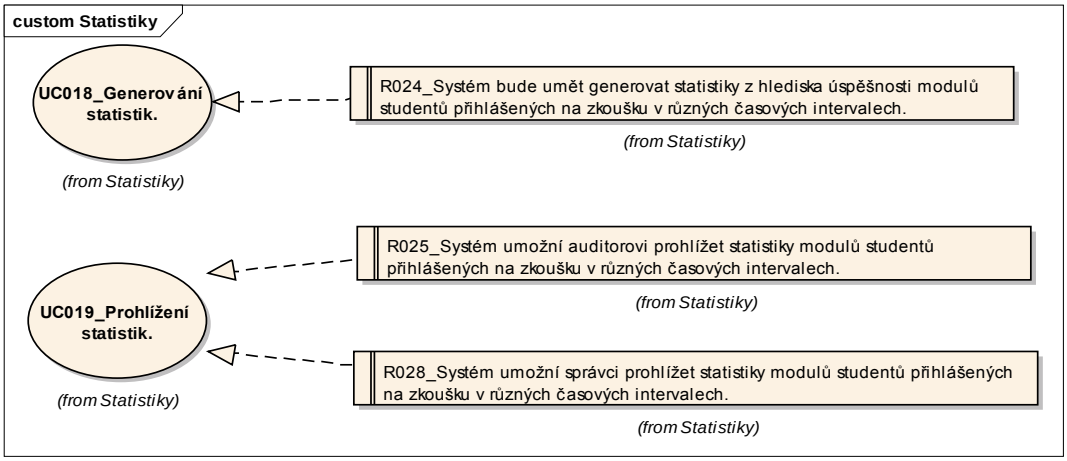
Moduly



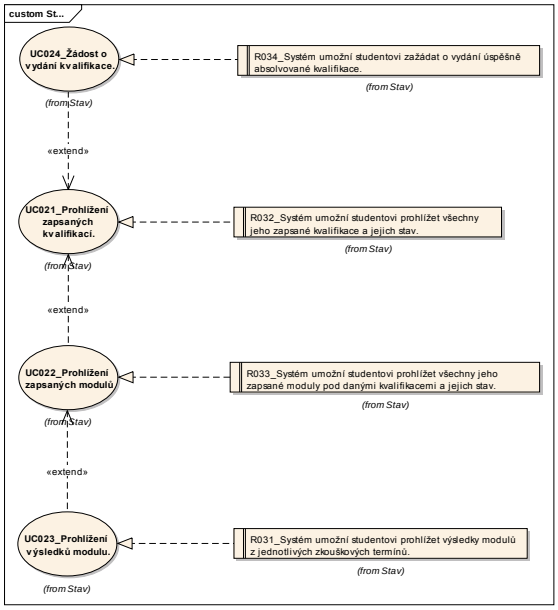
Místnosti



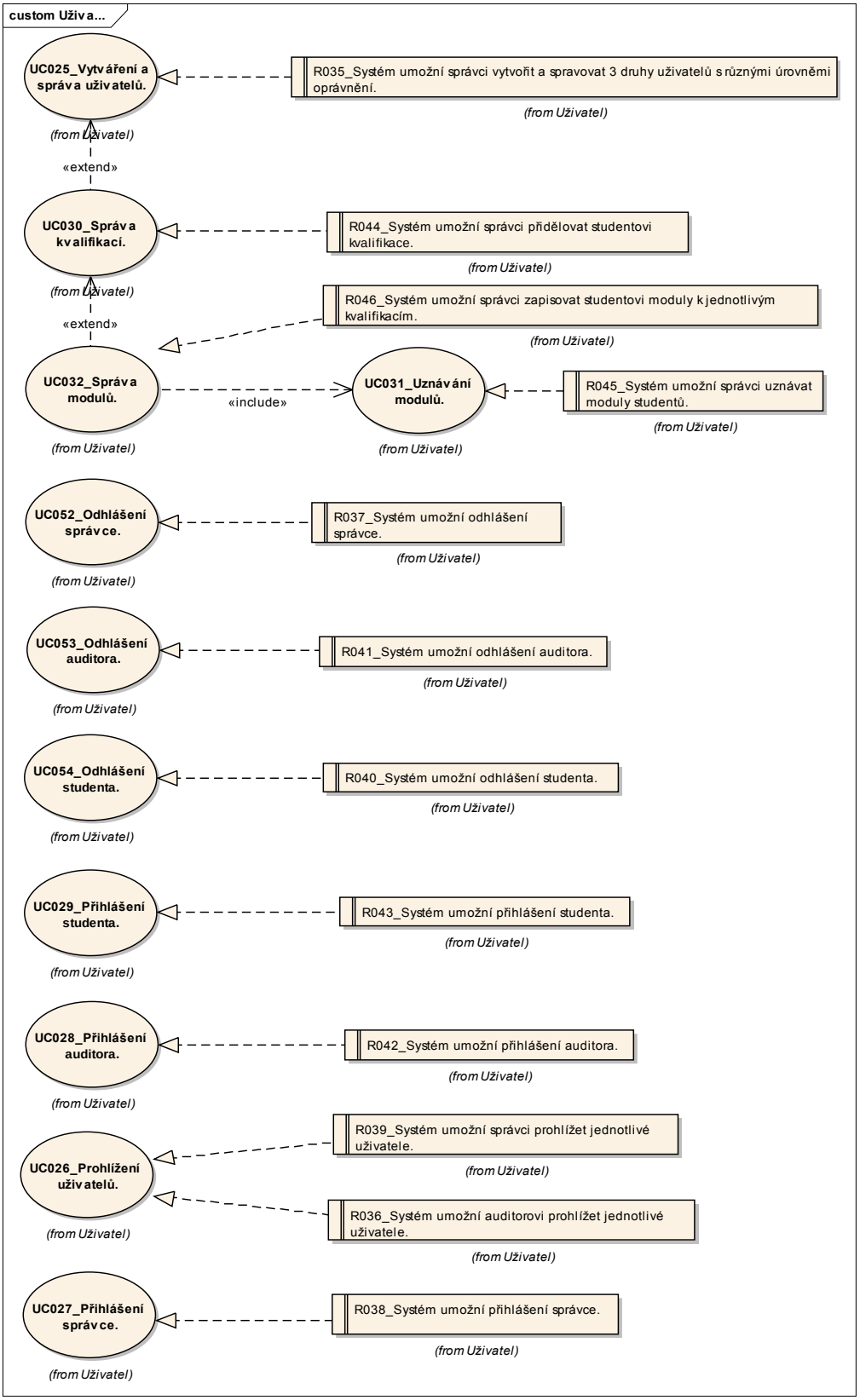
Statistiky



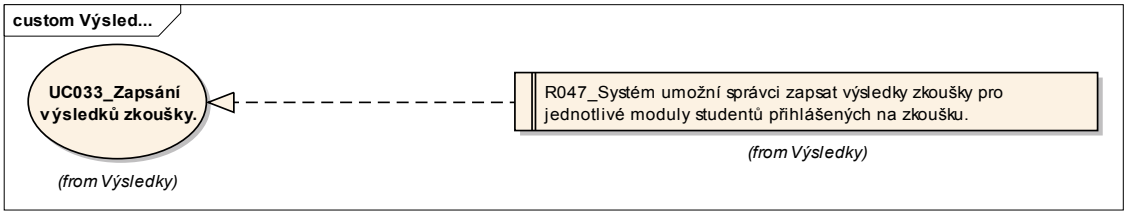
Stav



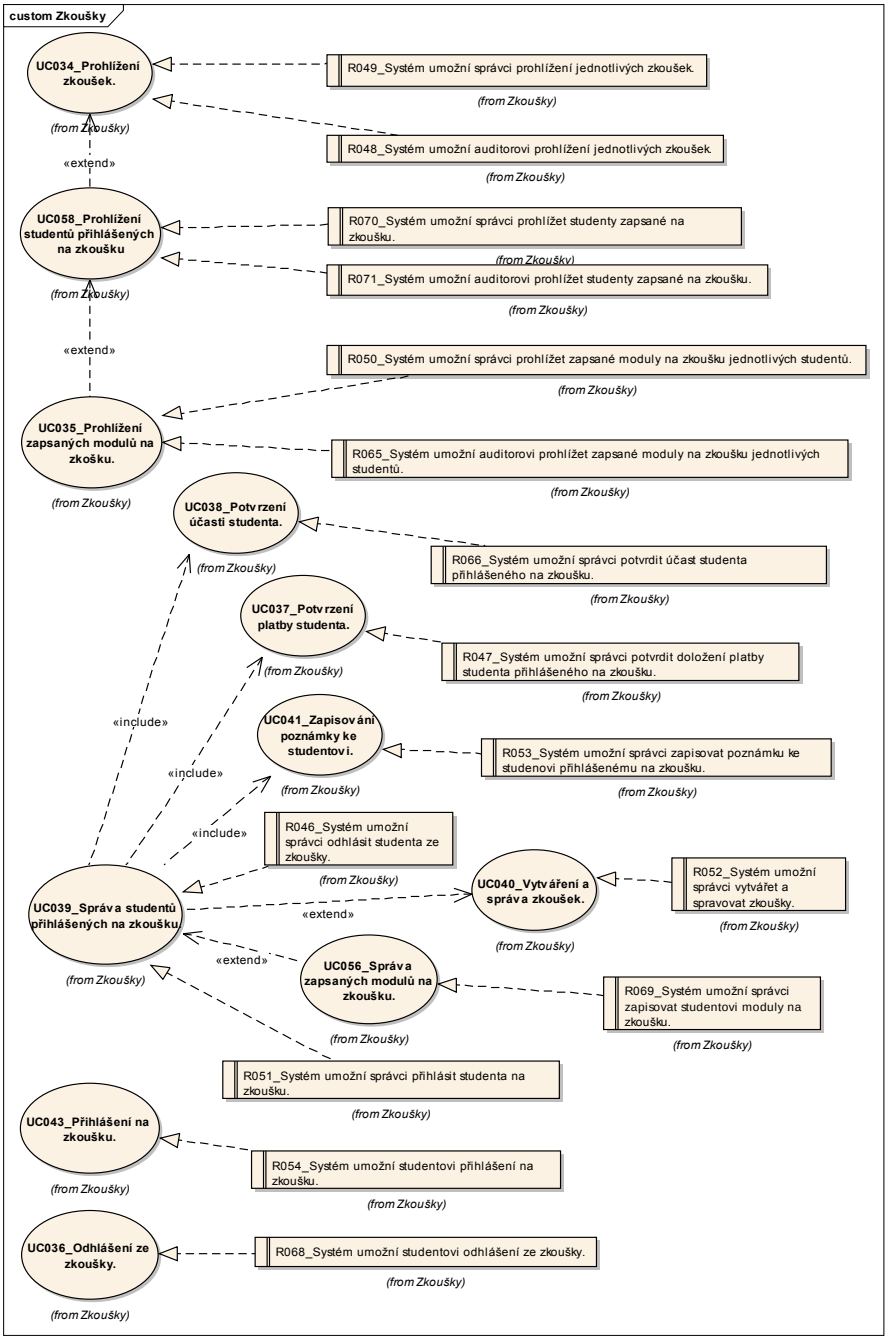
Uživatel


custom Uživa...
UC025_Vytváření a správa uživatelů.
(from Uživatel)
R035_Systém umožní správci vytvořit a spravovat 3 druhy uživatelů s různými úrovněmi oprávnění.
(from Uživatel)
«extend»
UC030_Správa kvalifikací.
(from Uživatel)
R044_Systém umožní správci přidělovat studentovi kvalifikace.
(from Uživatel)
«extend»
R046_Systém umožní správci zapisovat studentovi moduly k jednotlivým kvalifikacím.
(from Uživatel)
UC032_Správa modulů.
(from Uživatel)
«include»
UC031_Uznávání modulů.
(from Uživatel)
R045_Systém umožní správci uznávat moduly studentů.
(from Uživatel)
UC052_Odhlášení správce.
(from Uživatel)
R037_Systém umožní odhlášení správce.
(from Uživatel)
UC053_Odhlášení auditora.
(from Uživatel)
R041_Systém umožní odhlášení auditora.
(from Uživatel)
UC054_Odhlášení studenta.
(from Uživatel)
R040_Systém umožní odhlášení studenta.
(from Uživatel)
UC029_Přihlášení studenta.
(from Uživatel)
R043_Systém umožní přihlášení studenta.
(from Uživatel)
UC028_Přihlášení auditora.
(from Uživatel)
R042_Systém umožní přihlášení auditora.
(from Uživatel)
UC026_Prohlížení uživatelů.
(from Uživatel)
R039_Systém umožní správci prohlížet jednotlivé uživatele.
(from Uživatel)
R036_Systém umožní auditorovi prohlížet jednotlivé uživatele.
(from Uživatel)
UC027_Přihlášení správce.
(from Uživatel)
R038_Systém umožní přihlášení správce.
(from Uživatel)

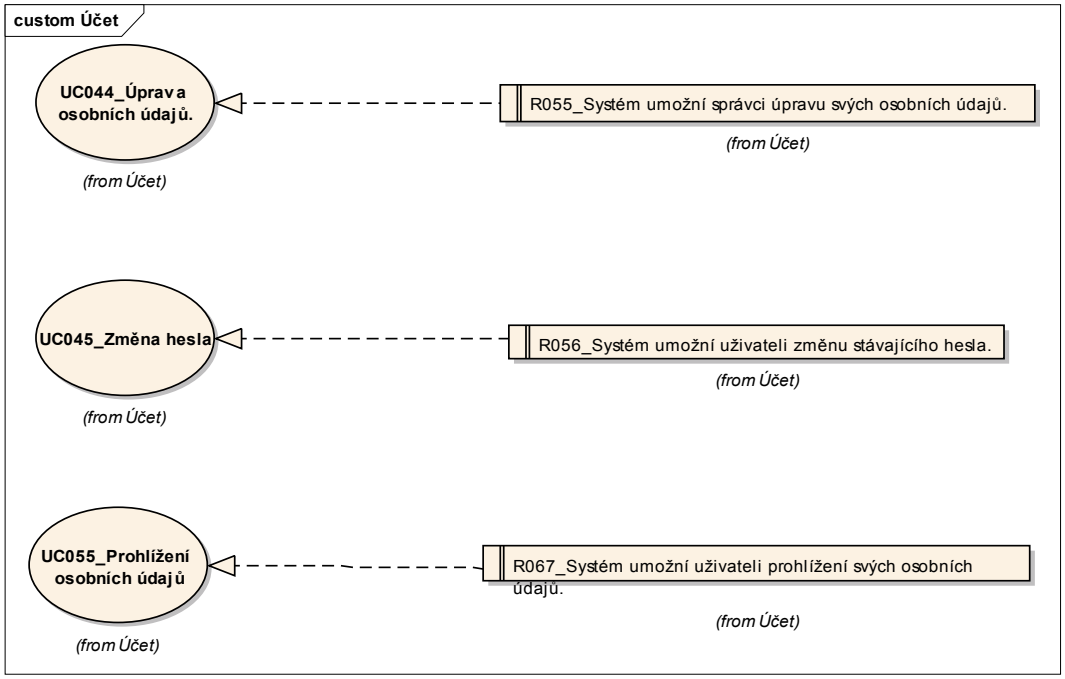
Výsledky



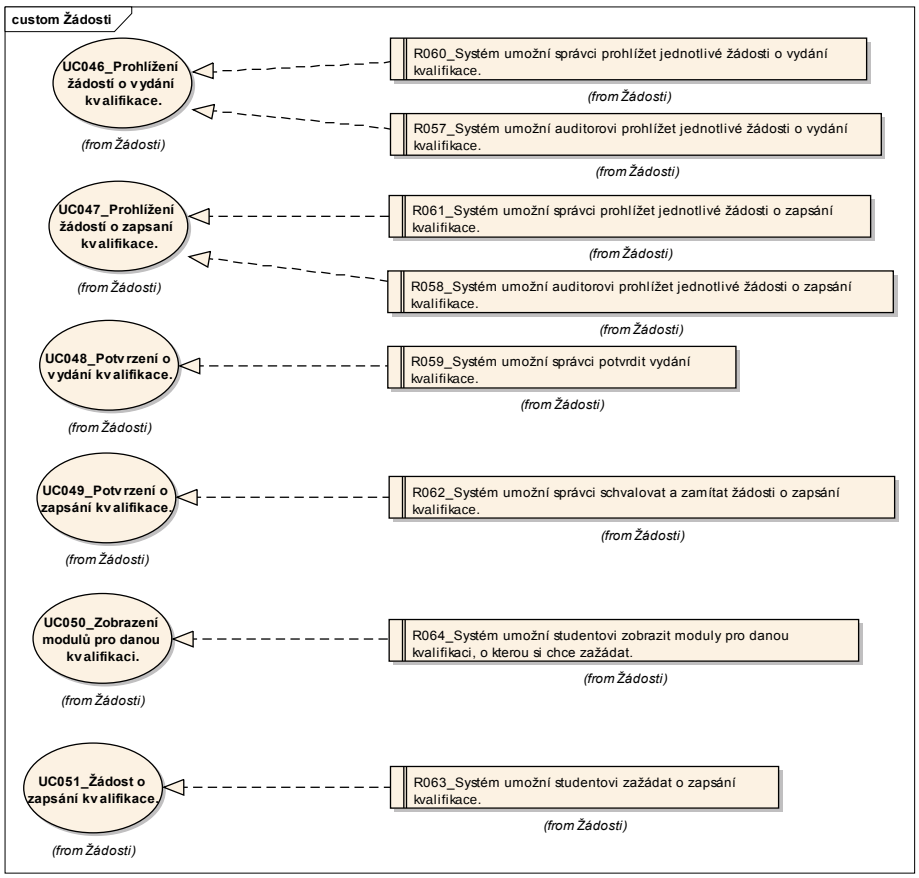
Zkoušky



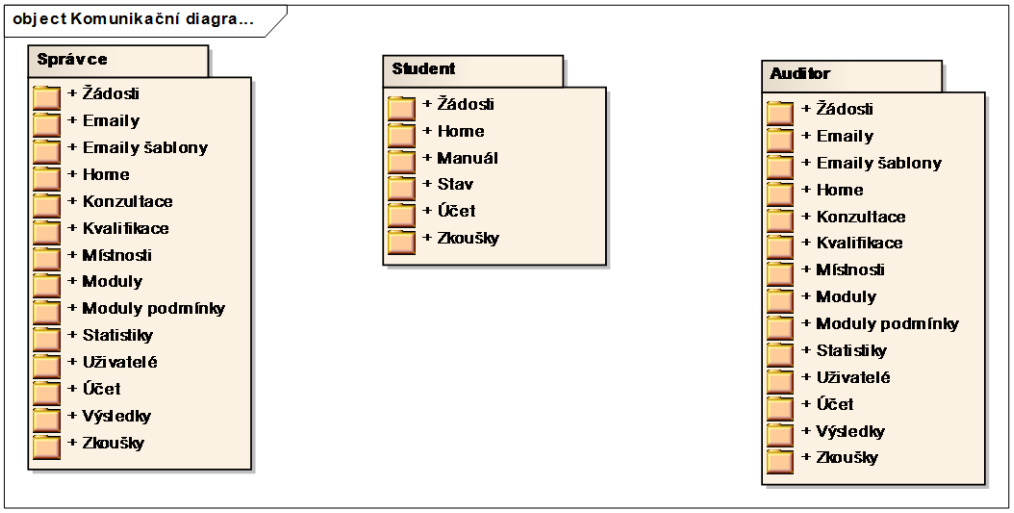
Účet



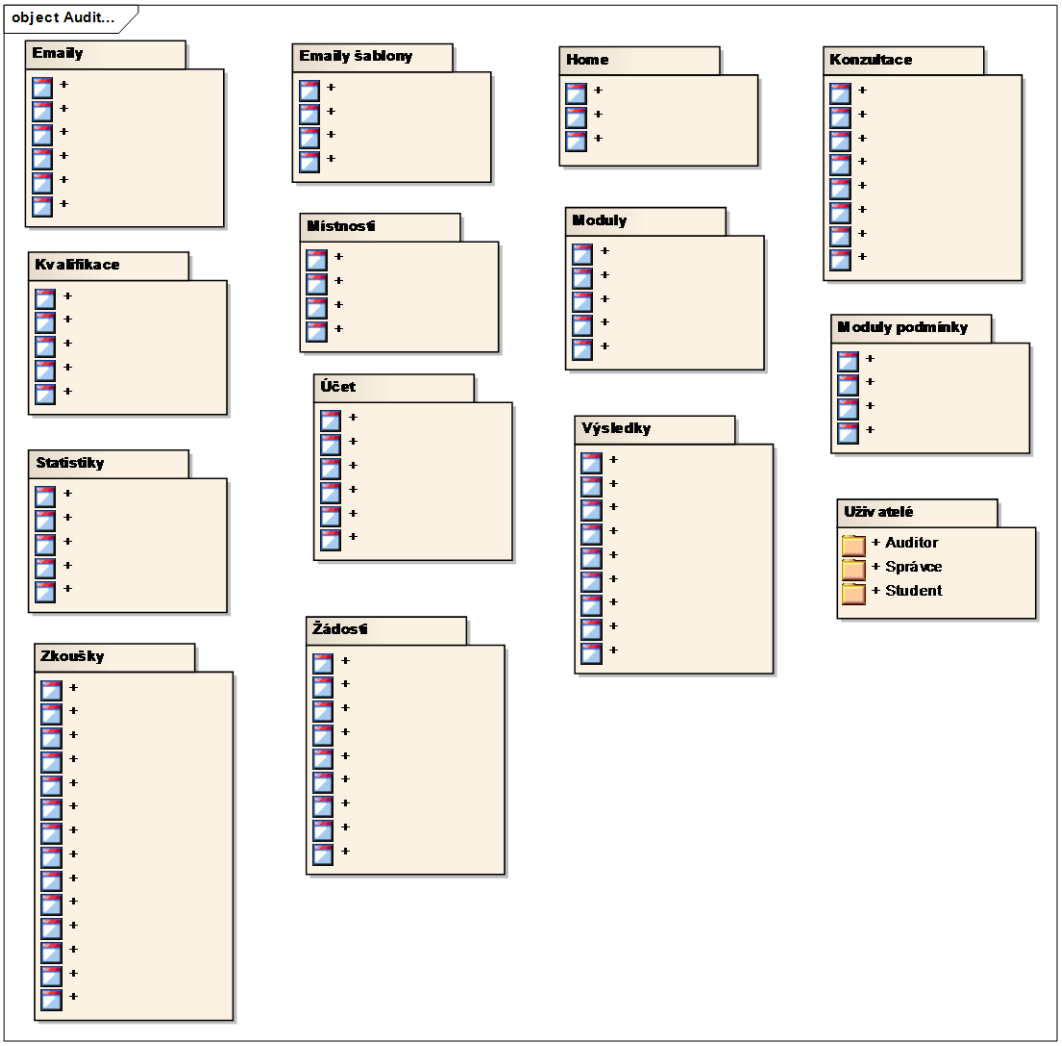
Žádosti



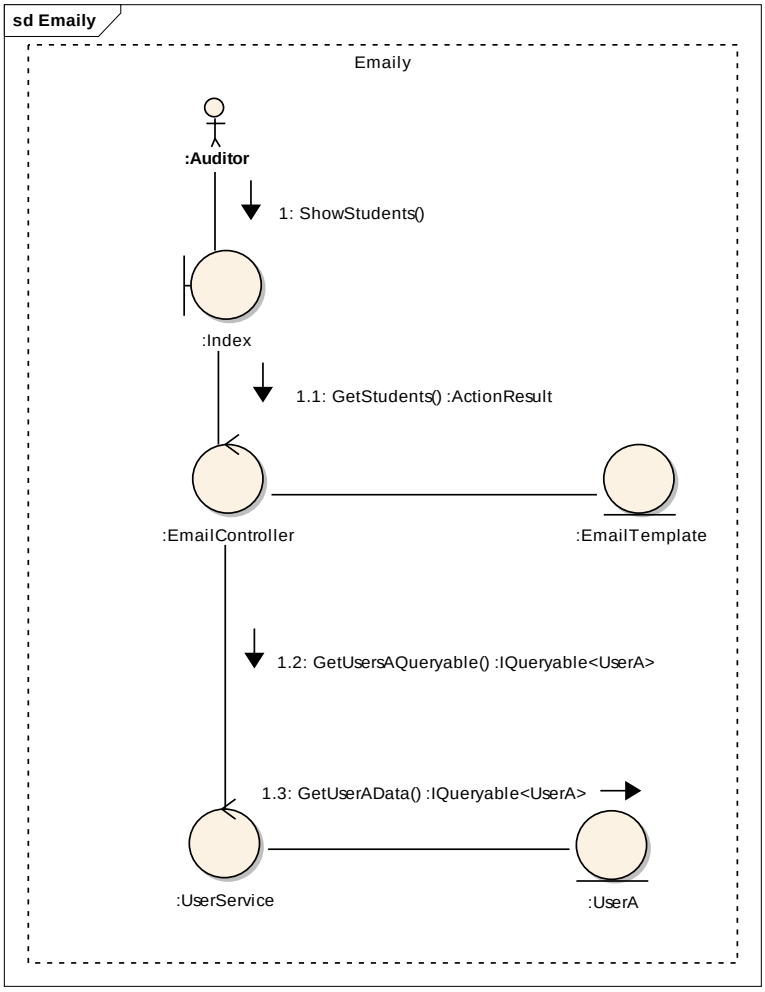
PŘÍLOHA I: Komunikační diagramy



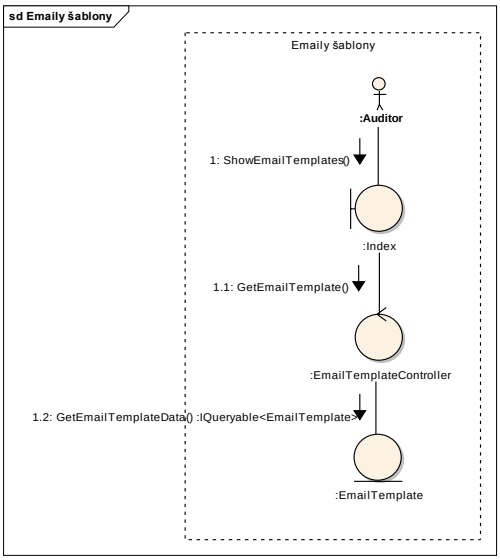
Auditor



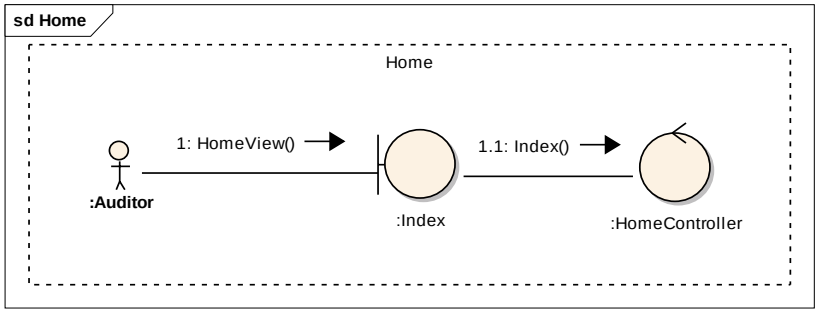
Emaily



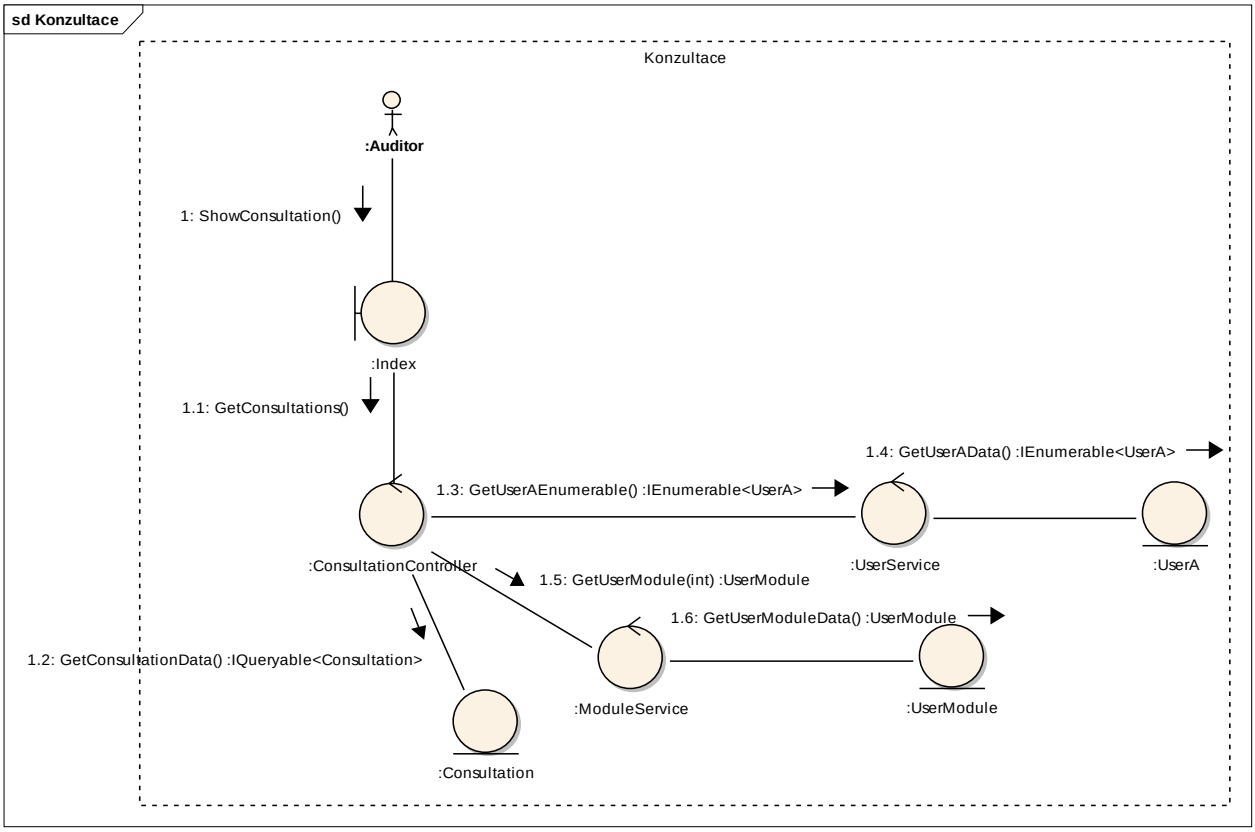
Emaily šablony



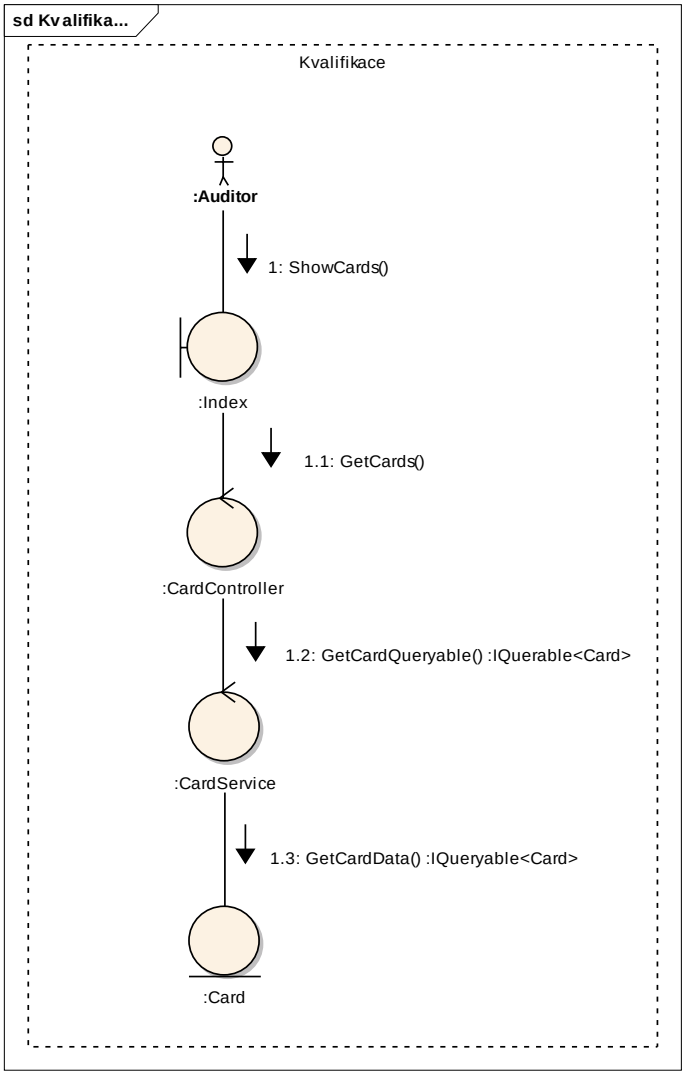
Home



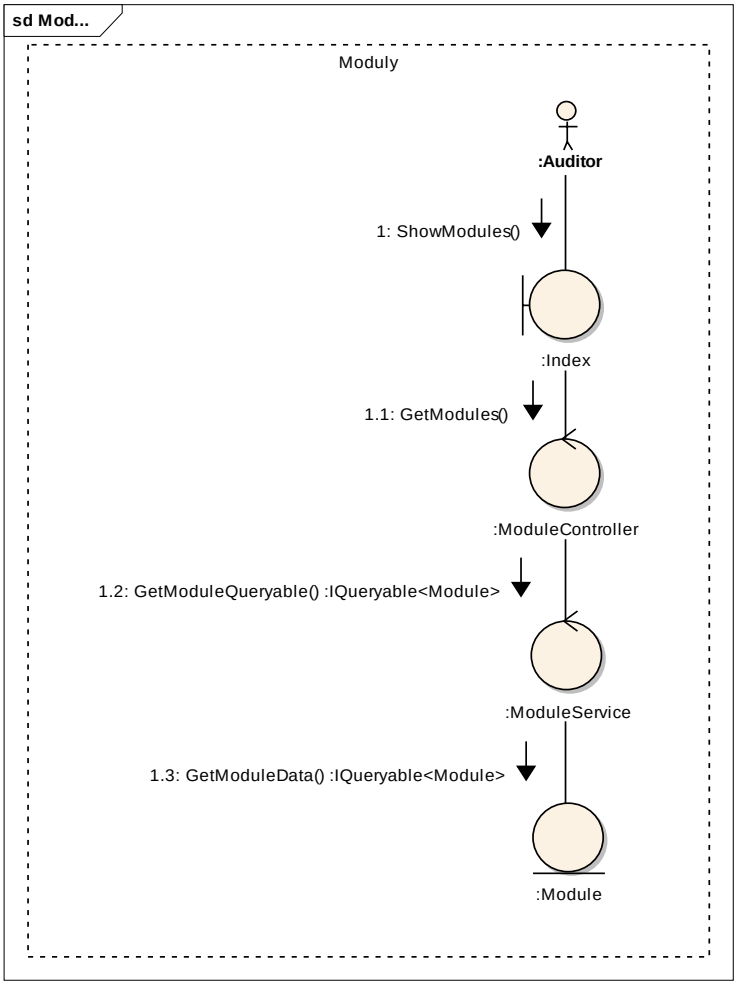
Konzultace



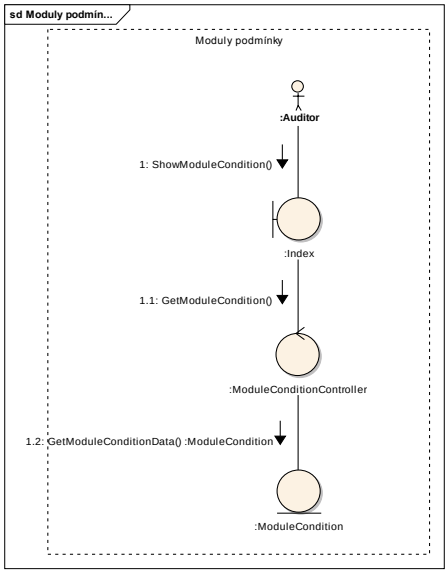
Kvalifikace



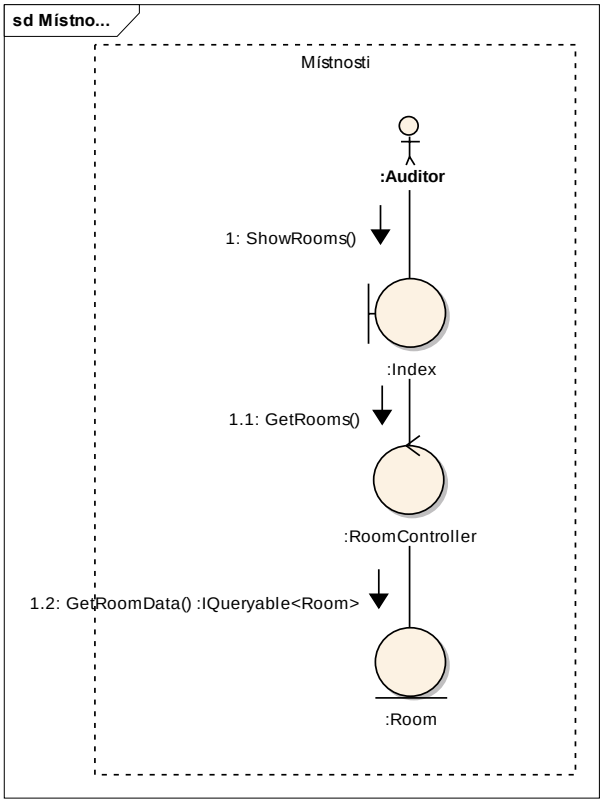
Moduly



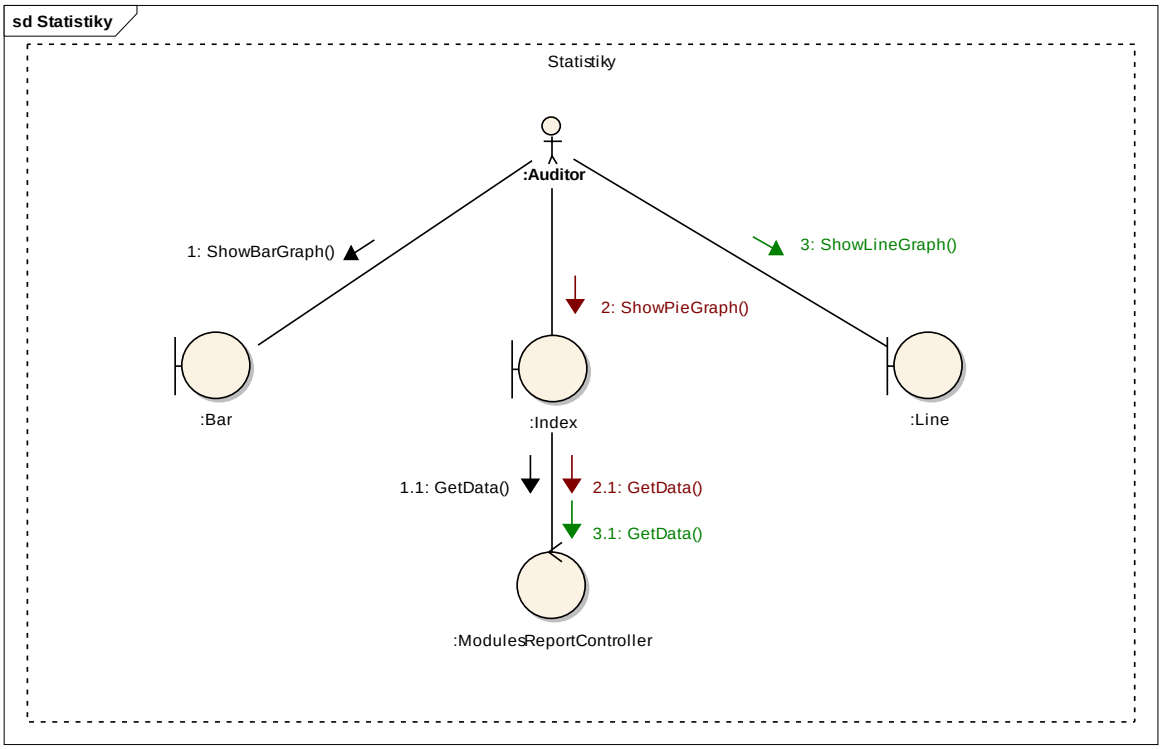
Moduly podmínky



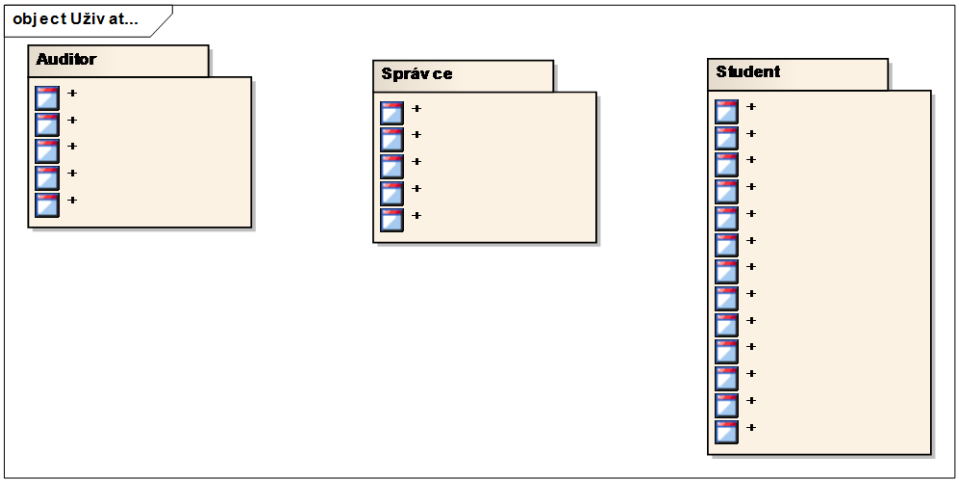
Místnosti



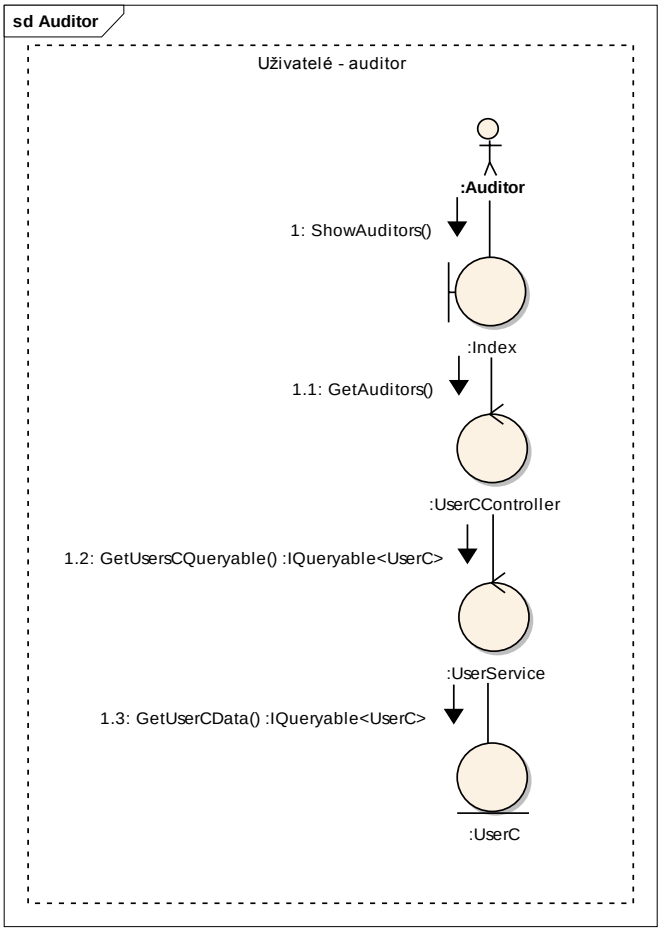
Statistiky



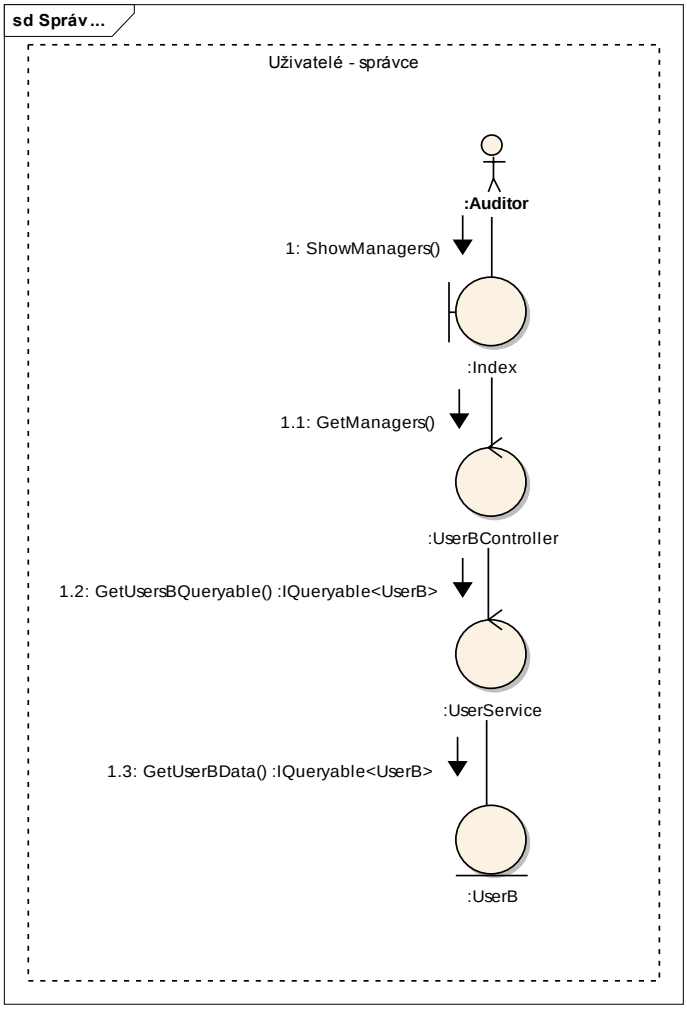
Uživatelé



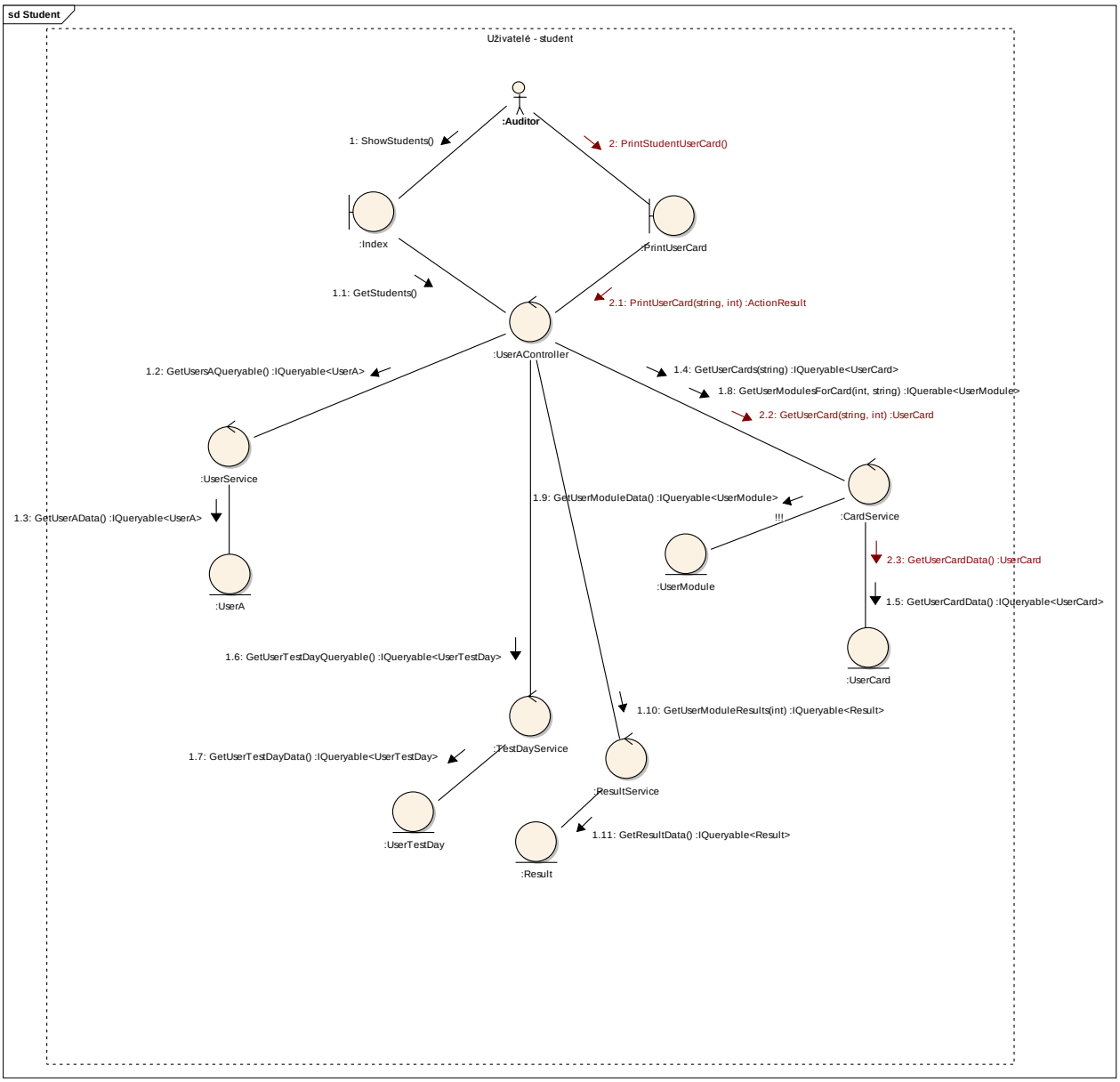
Auditor



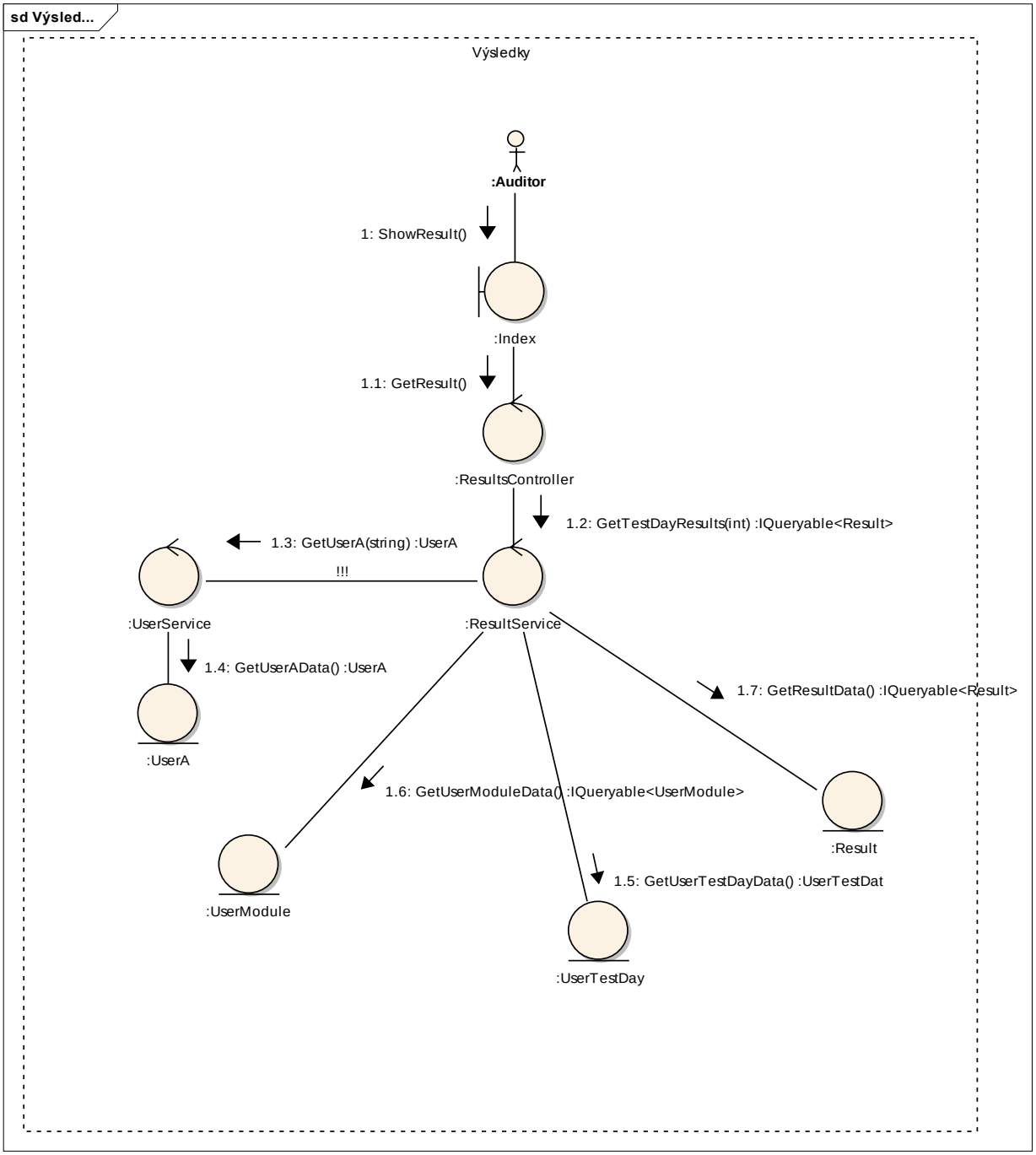
Správce


sd Správ...
Uživatelé - správce
:Auditor
1: ShowManagers()
:Index
1.1: GetManagers()
:UserBController
1.2: GetUsersBQueryable() :IQueryable<UserB>
:UserService
1.3: GetUserBData() :IQueryable<UserB>
:UserB

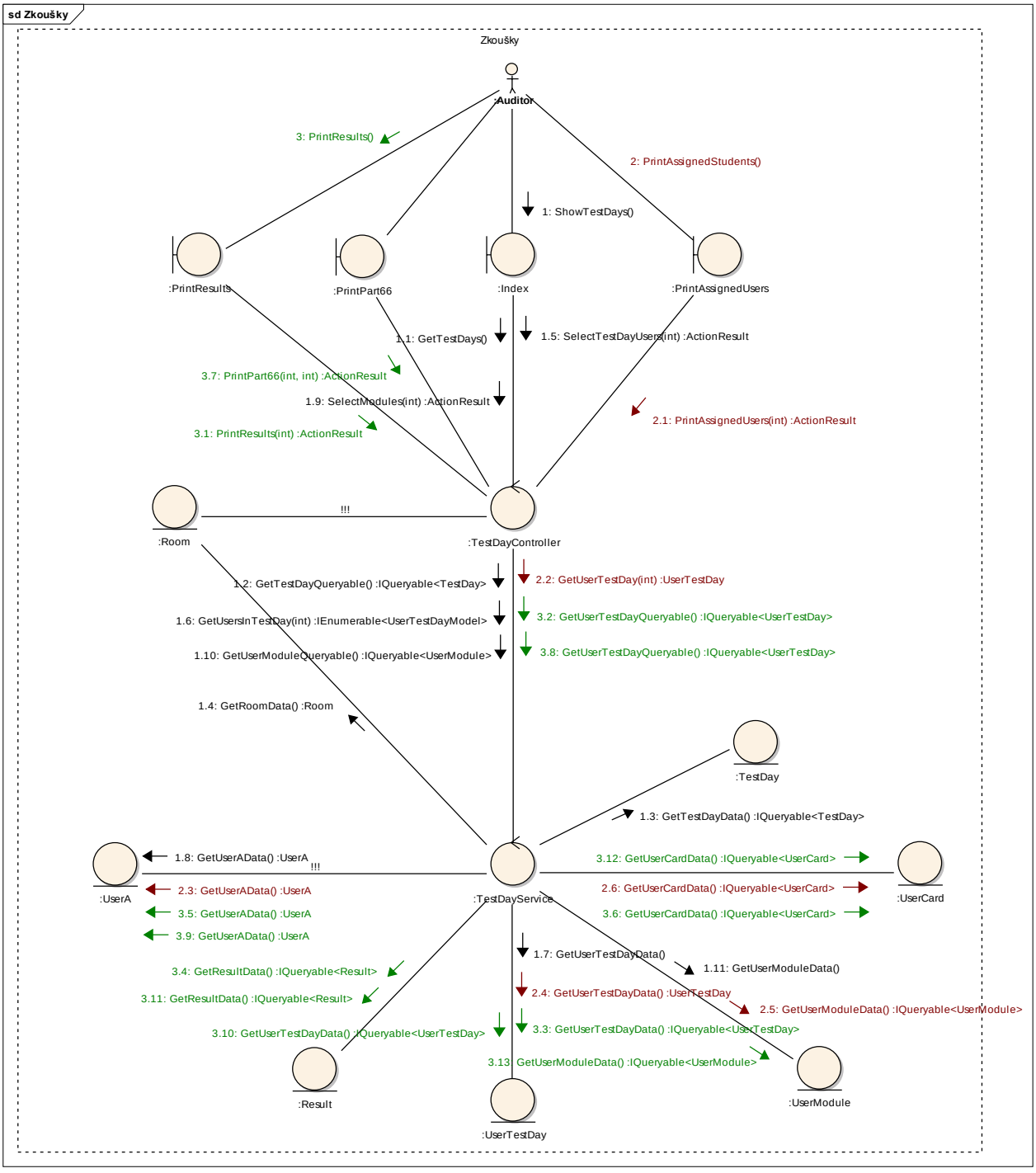
Student


sd Student
Uživatelé - student
:Auditor
1: ShowStudents()
2: PrintStudentUserCard()
:Index
:PrintUserCard
1.1: GetStudents()
2.1: PrintUserCard(string, int) :ActionResult
:UserAController
1.2: GetUsersAQueryable() :IQueryable<UserA>
1.4: GetUserCards(string) :IQueryable<UserCard>
1.8: GetUserModulesForCard(int, string) :IQuerable<UserModule>
2.2: GetUserCard(string, int) :UserCard
:UserService
1.3: GetUserAData() :IQueryable<UserA>
1.9: GetUserModuleData() :IQueryable<UserModule>
!!!
:CardService
:UserModule
2.3: GetUserCardData() :UserCard
1.5: GetUserCardData() :IQueryable<UserCard>
:UserA
:UserCard
1.6: GetUserTestDayQueryable() :IQueryable<UserTestDay>
1.10: GetUserModuleResults(int) :IQueryable<Result>
:TestDayService
1.7: GetUserTestDayData() :IQueryable<UserTestDay>
:ResultService
:UserTestDay
1.11: GetResultData() :IQueryable<Result>
:Result

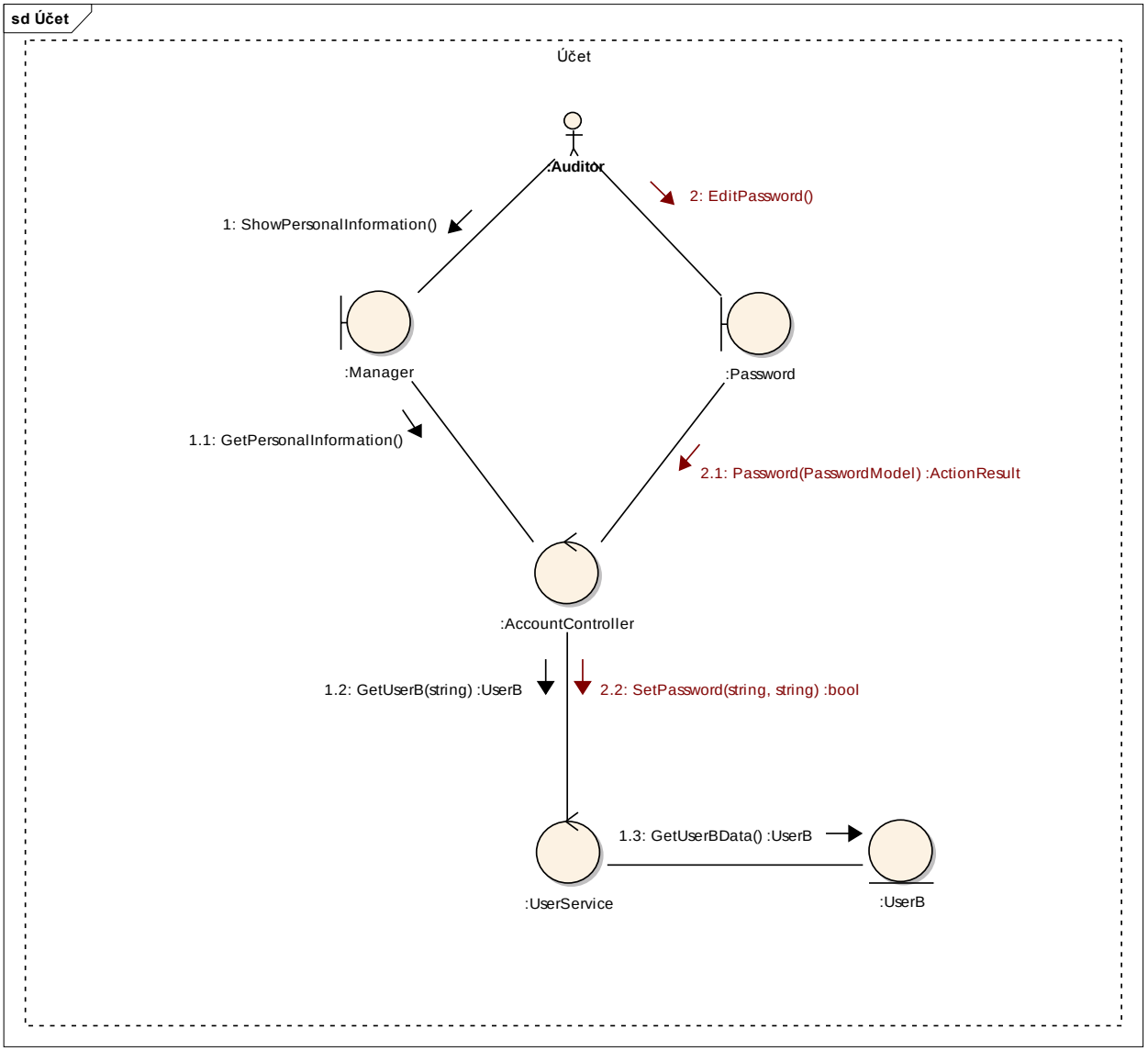
Výsledky


sd Výsled...
Výsledky
:Auditor
1: ShowResult()
:Index
1.1: GetResult()
:ResultsController
1.2: GetTestDayResults(int) :IQueryable<Result>
1.3: GetUserA(string) :UserA
!!!
:UserService
:ResultService
1.4: GetUserAData() :UserA
:UserA
1.7: GetResultData() :IQueryable<Result>
1.6: GetUserModuleData() :IQueryable<UserModule>
:Result
:UserModule
1.5: GetUserTestDayData() :UserTestDat
:UserTestDay

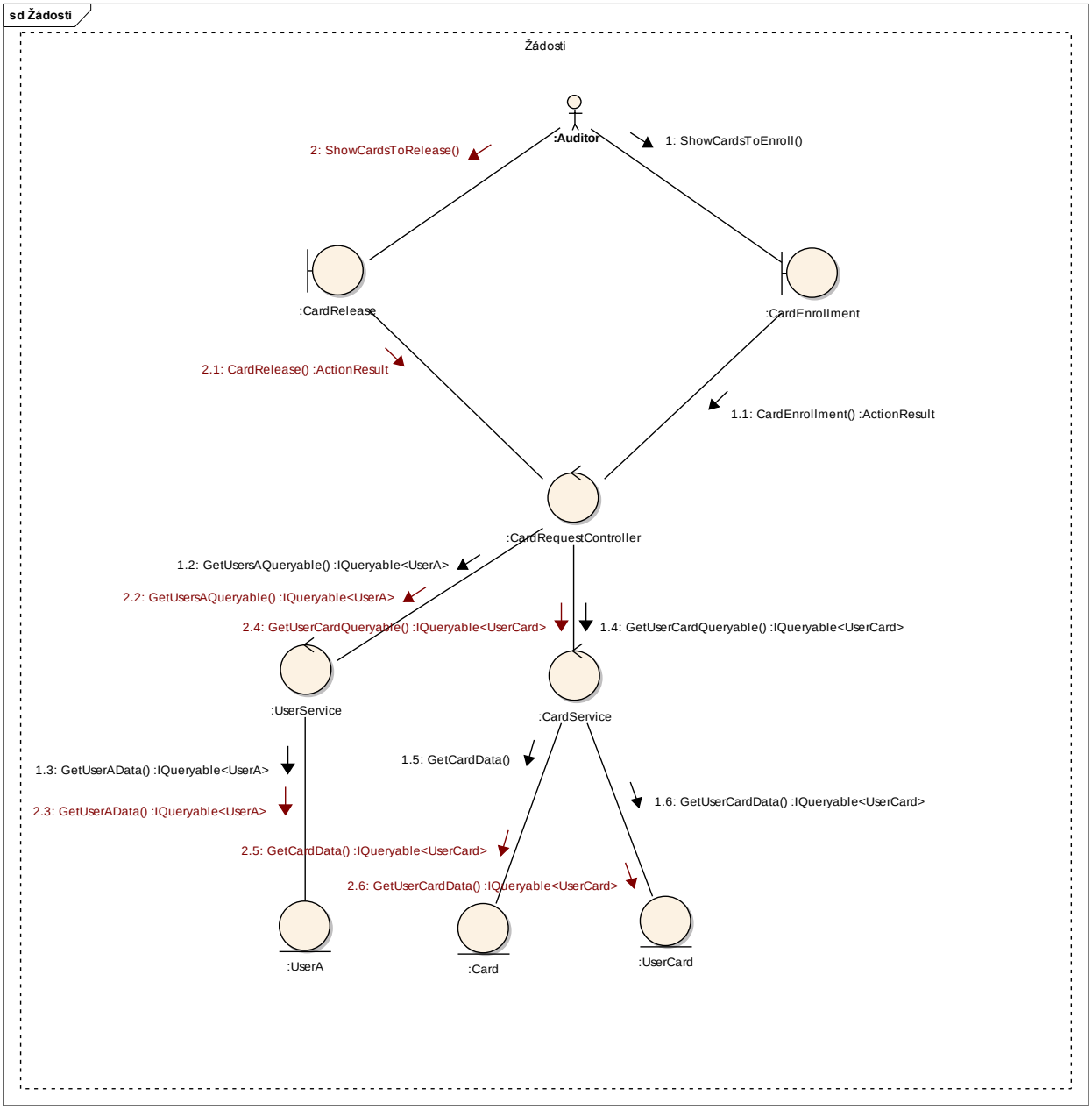
Zkoušky


sd Zkoušky
Zkoušky
:Auditor
3: PrintResults()
2: PrintAssignedStudents()
1: ShowTestDays()
:PrintResults
:PrintPart66
:Index
:PrintAssignedUsers
1.1: GetTestDays()
1.5: SelectTestDayUsers(int) :ActionResult
3.7: PrintPart66(int, int) :ActionResult
1.9: SelectModules(int) :ActionResult
2.1: PrintAssignedUsers(int) :ActionResult
3.1: PrintResults(int) :ActionResult
:Room
!!!
:TestDayController
1.2: GetTestDayQueryable() :IQueryable<TestDay>
2.2: GetUserTestDay(int) :UserTestDay
1.6: GetUsersInTestDay(int) :IEnumerable<UserTestDayModel>
3.2: GetUserTestDayQueryable() :IQueryable<UserTestDay>
1.10: GetUserModuleQueryable() :IQueryable<UserModule>
3.8: GetUserTestDayQueryable() :IQueryable<UserTestDay>
1.4: GetRoomData() :Room
:TestDay
1.3: GetTestDayData() :IQueryable<TestDay>
1.8: GetUserAData() :UserA
!!!
3.12: GetUserCardData() :IQueryable<UserCard>
:UserA
2.3: GetUserAData() :UserA
:TestDayService
2.6: GetUserCardData() :IQueryable<UserCard>
:UserCard
3.5: GetUserAData() :UserA
3.6: GetUserCardData() :IQueryable<UserCard>
3.9: GetUserAData() :UserA
1.7: GetUserTestDayData()
1.11: GetUserModuleData()
3.4: GetResultData() :IQueryable<Result>
2.4: GetUserTestDayData() :UserTestDay
2.5: GetUserModuleData() :IQueryable<UserModule>
3.11: GetResultData() :IQueryable<Result>
3.10: GetUserTestDayData() :IQueryable<UserTestDay>
3.3: GetUserTestDayData() :IQueryable<UserTestDay>
3.13: GetUserModuleData() :IQueryable<UserModule>
:Result
:UserTestDay
:UserModule

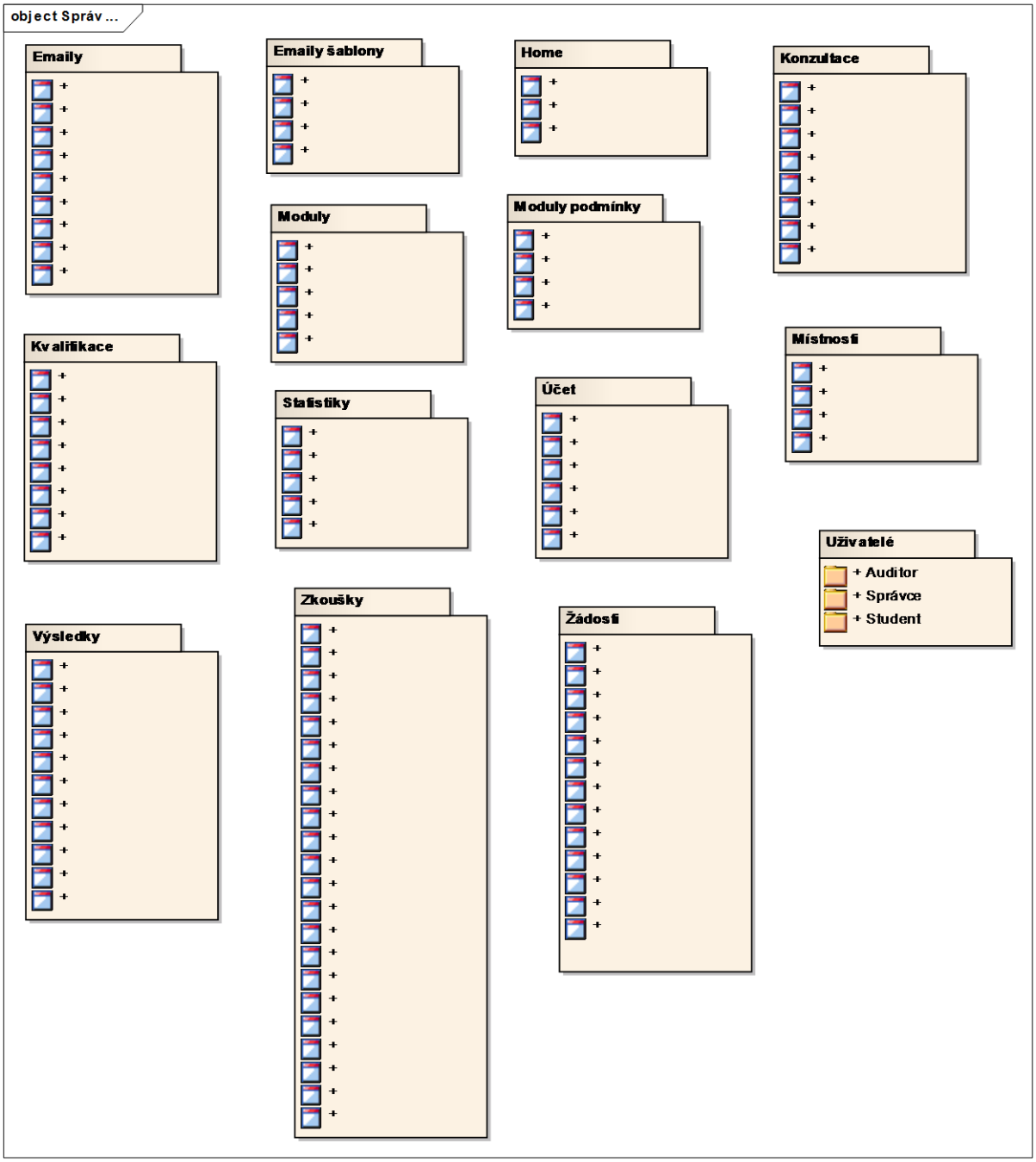
Účet


sd Účet
Účet
:Auditor
1: ShowPersonalInformation()
2: EditPassword()
:Manager
:Password
1.1: GetPersonalInformation()
2.1: Password(PasswordModel) :ActionResult
:AccountController
1.2: GetUserB(string) :UserB
2.2: SetPassword(string, string) :bool
1.3: GetUserBData() :UserB
:UserService
:UserB

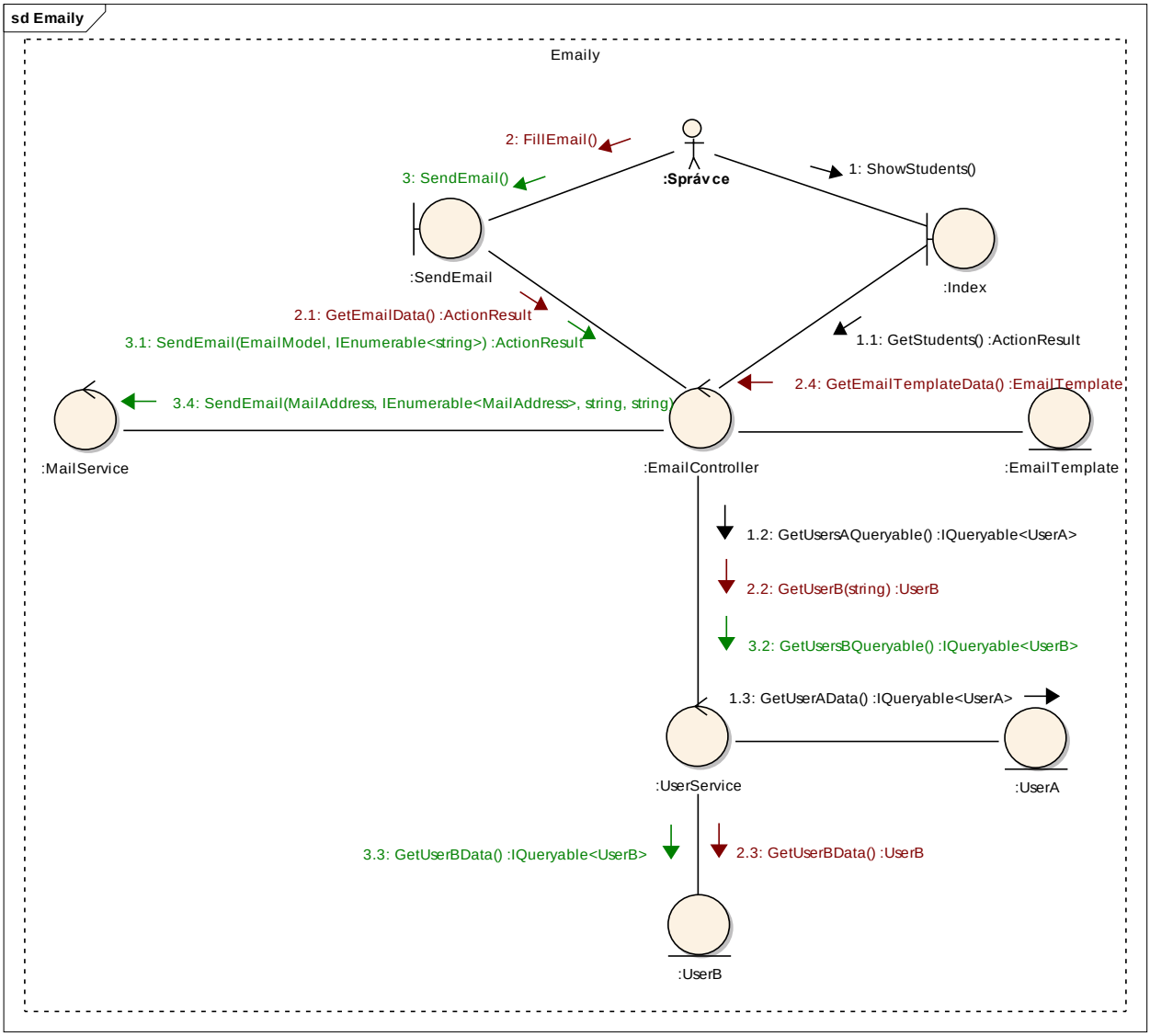
Žádosti


sd Žádosti
Žádosti
:Auditor
1: ShowCardsToEnroll()
2: ShowCardsToRelease()
:CardRelease
:CardEnrollment
2.1: CardRelease() :ActionResult
1.1: CardEnrollment() :ActionResult
:CardRequestController
1.2: GetUsersAQueryable() :IQueryable<UserA>
2.2: GetUsersAQueryable() :IQueryable<UserA>
2.4: GetUserCardQueryable() :IQueryable<UserCard>
1.4: GetUserCardQueryable() :IQueryable<UserCard>
:UserService
:CardService
1.3: GetUserAData() :IQueryable<UserA>
2.3: GetUserAData() :IQueryable<UserA>
1.5: GetCardData()
1.6: GetUserCardData() :IQueryable<UserCard>
2.5: GetCardData() :IQueryable<UserCard>
2.6: GetUserCardData() :IQueryable<UserCard>
:UserA
:Card
:UserCard

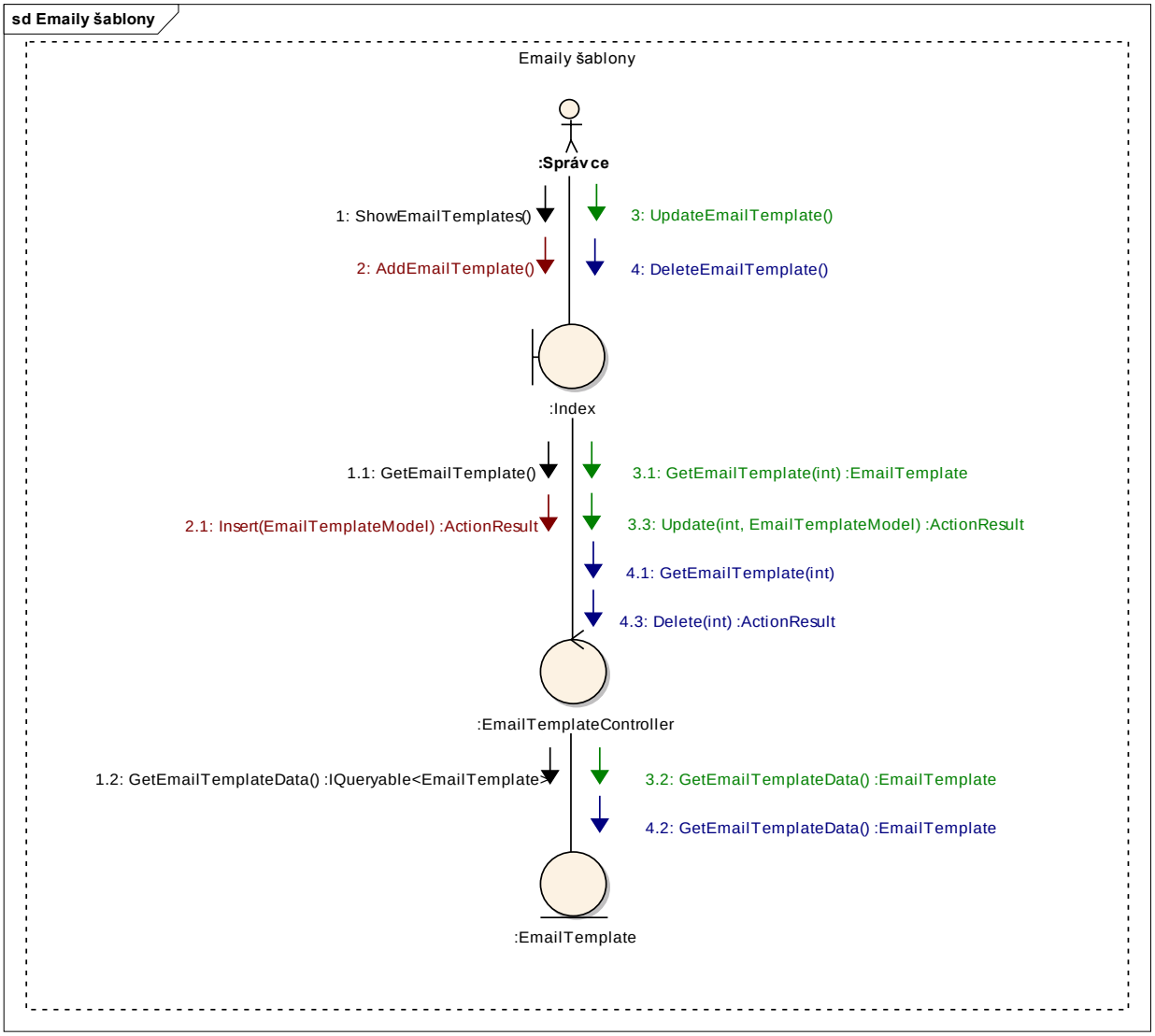
Správce


object Správ...
Emaily
Emaily šablony
Home
Konzultace
Moduly
Moduly podmínky
Kvalifikace
Místnosti
Statistiky
Účet
Uživatelé
+ Auditor
+ Správce
+ Student
Zkoušky
Výsledky
Žádosti

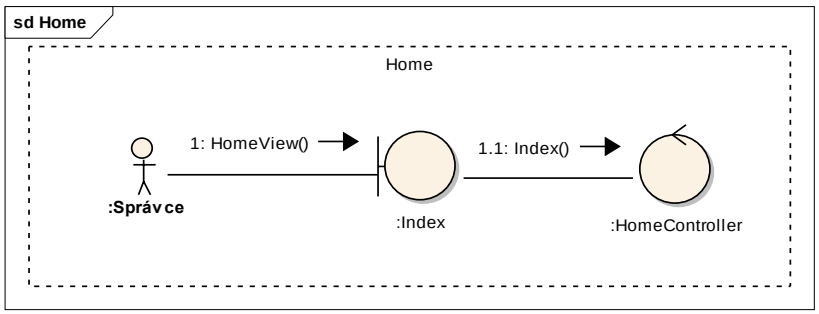
Emaily


sd Emaily
Emaily
2: FillEmail()
3: SendEmail()
:Správce
1: ShowStudents()
:SendEmail
:Index
2.1: GetEmailData() :ActionResult
3.1: SendEmail(EmailModel, IEnumerable<string>) :ActionResult
1.1: GetStudents() :ActionResult
2.4: GetEmailTemplateData() :EmailTemplate
3.4: SendEmail(MailAddress, IEnumerable<MailAddress>, string, string)
:MailService
:EmailController
:EmailTemplate
1.2: GetUsersAQueryable() :IQueryable<UserA>
2.2: GetUserB(string) :UserB
3.2: GetUsersBQueryable() :IQueryable<UserB>
1.3: GetUserAData() :IQueryable<UserA>
:UserService
:UserA
3.3: GetUserBData() :IQueryable<UserB>
2.3: GetUserBData() :UserB
:UserB

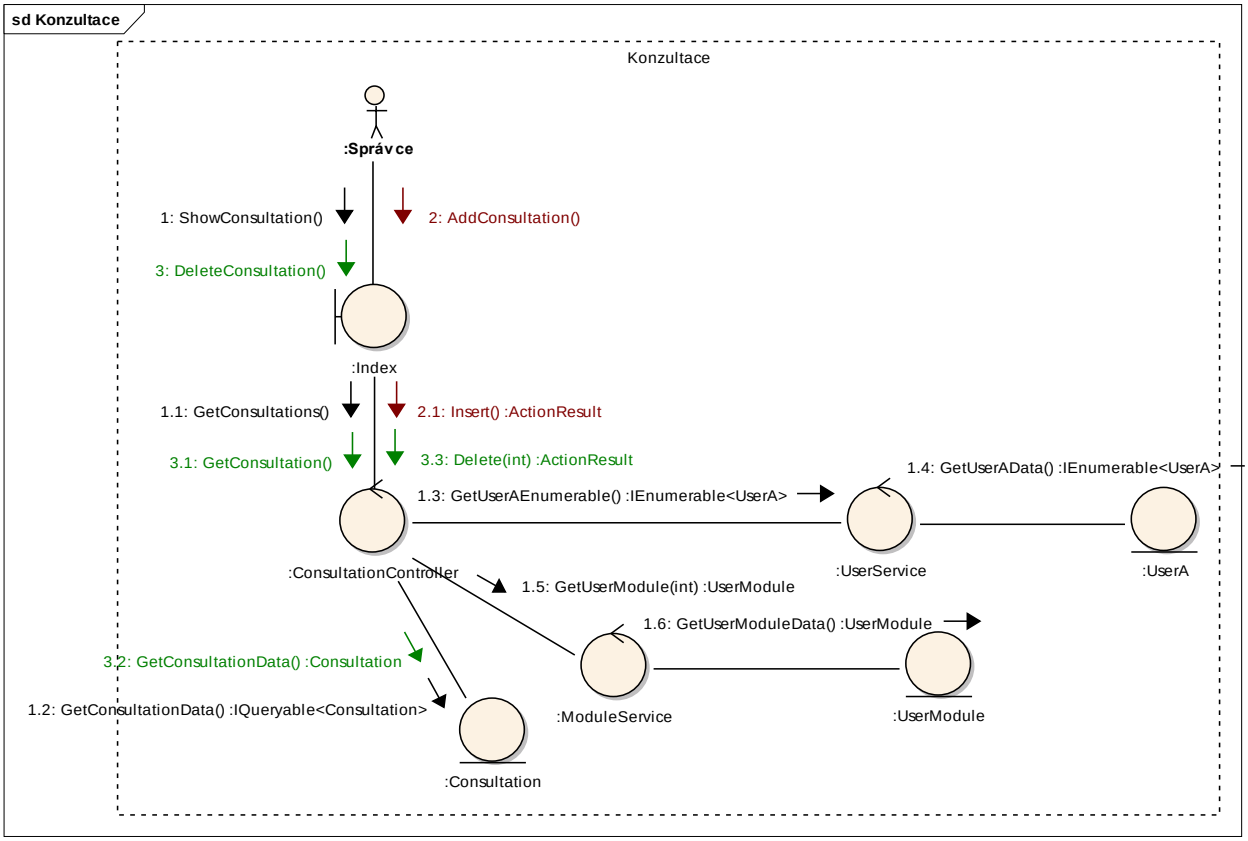
Emaily šablony



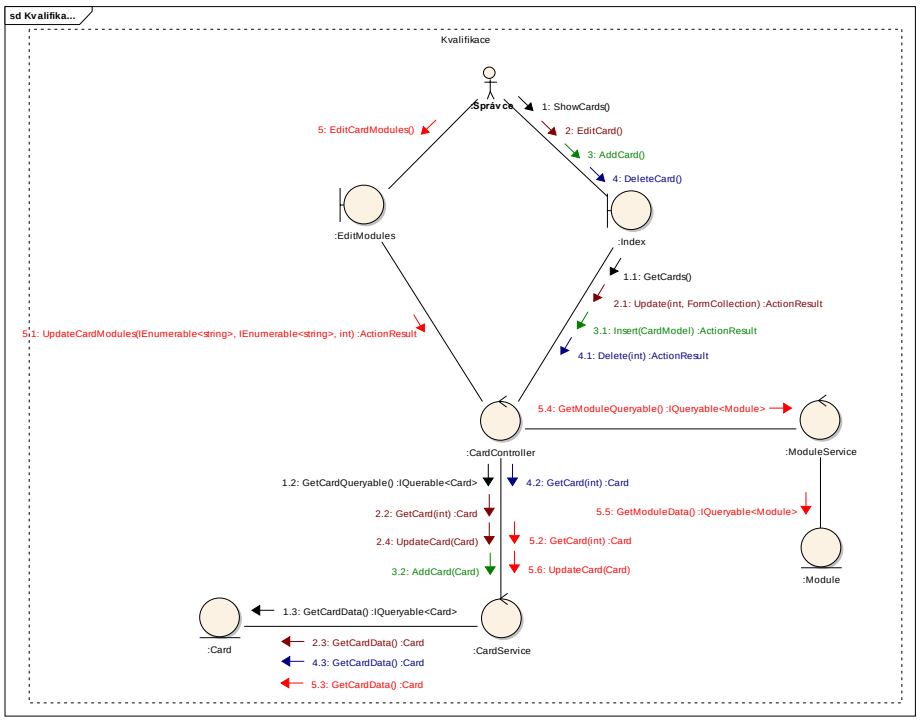
Home



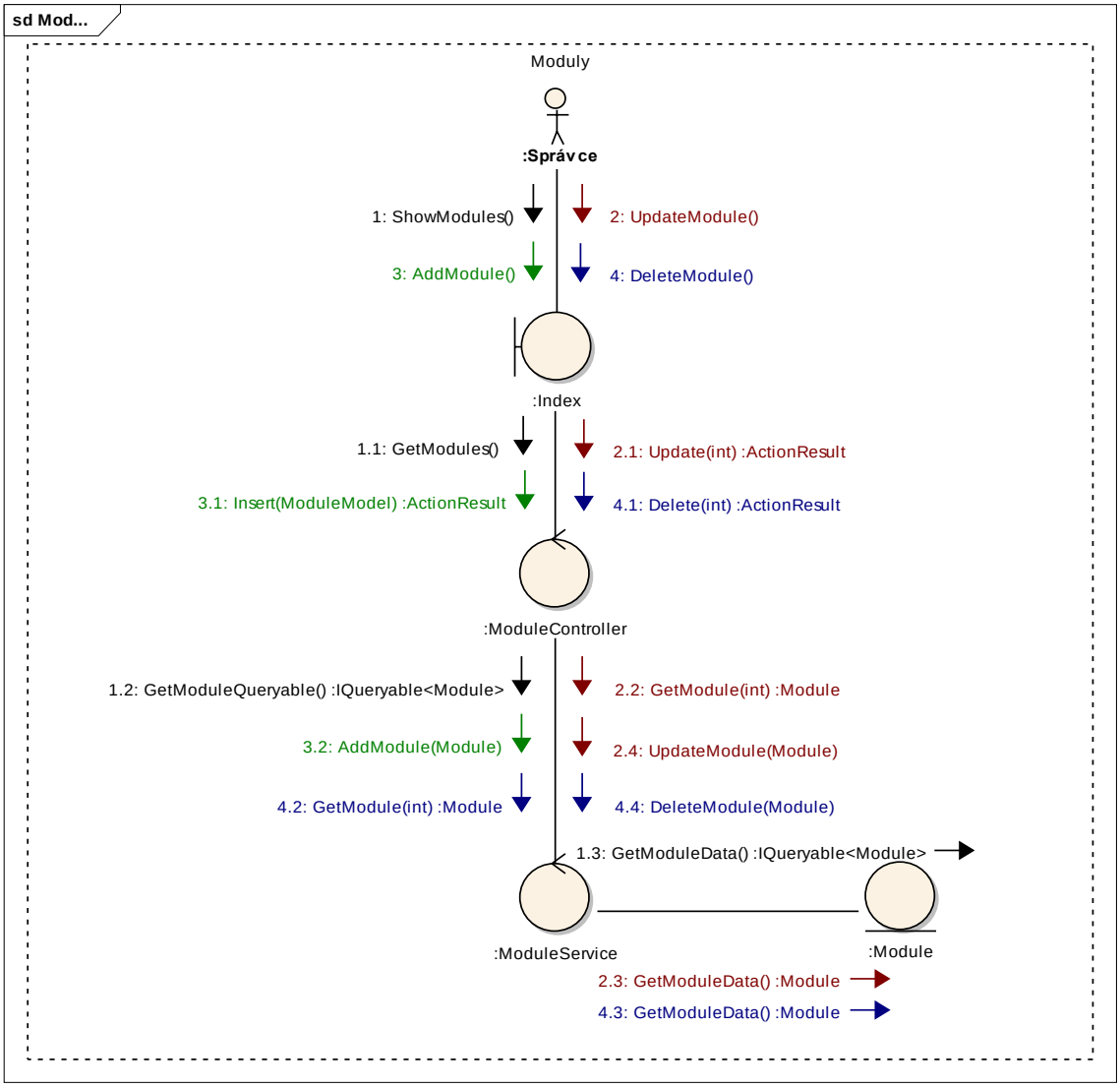
Konzultace



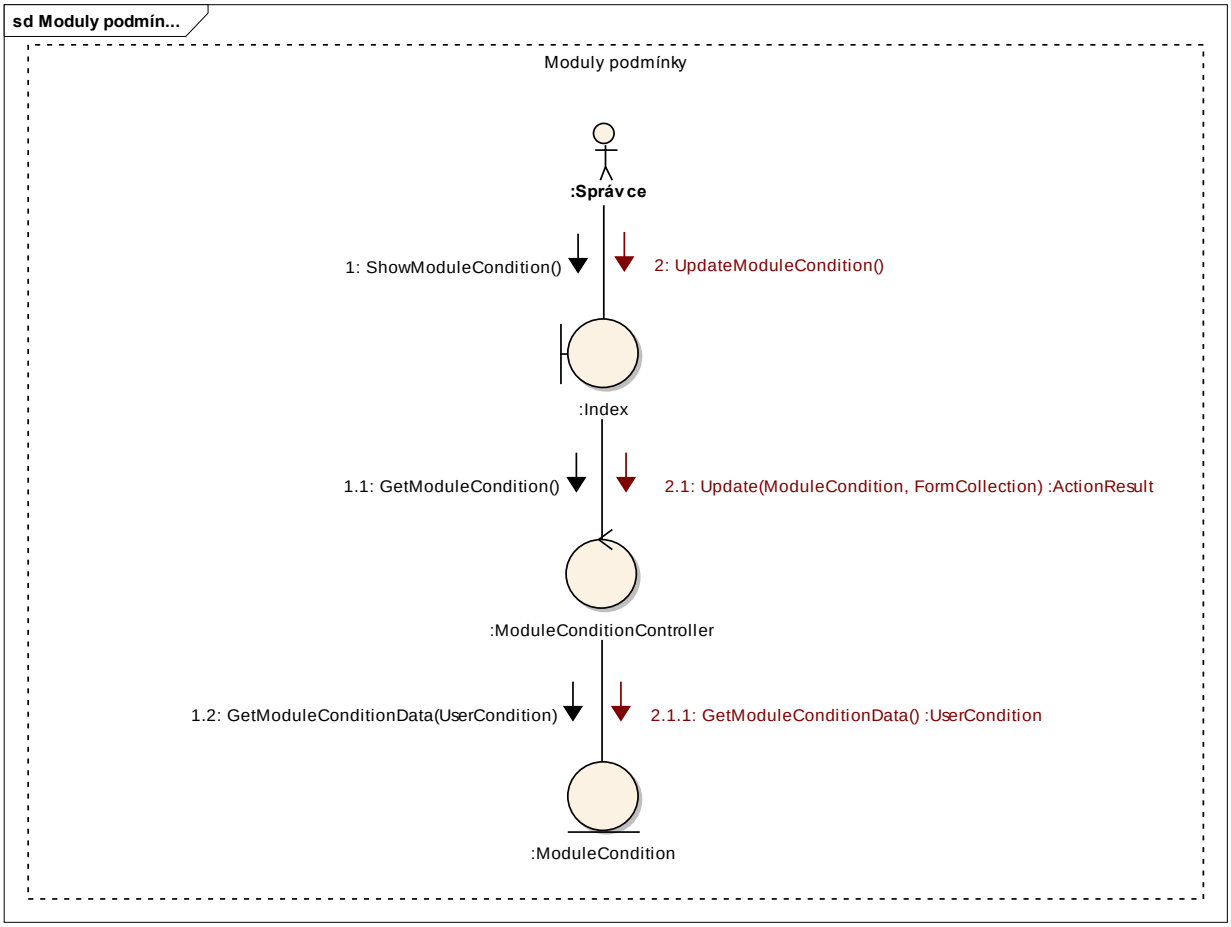
Kvalifikace



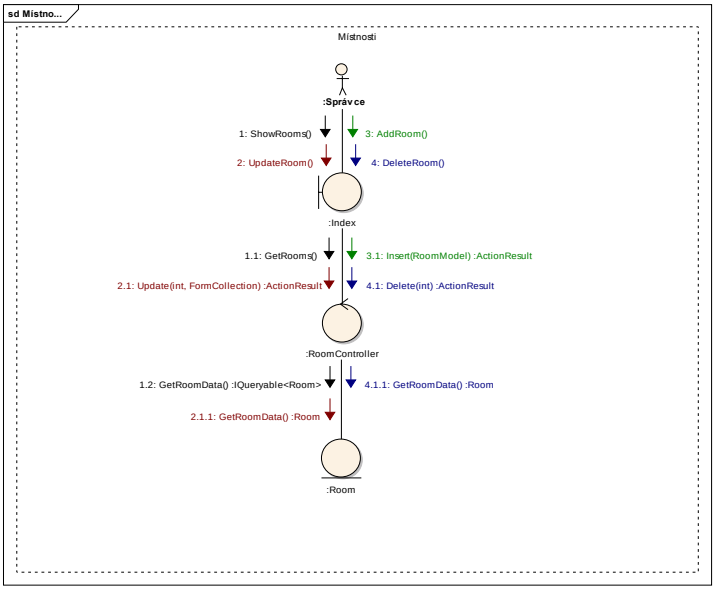
Moduly


sd Mod...
Moduly
:Správce
1: ShowModules()
2: UpdateModule()
3: AddModule()
4: DeleteModule()
:Index
1.1: GetModules()
2.1: Update(int) :ActionResult
3.1: Insert(ModuleModel) :ActionResult
4.1: Delete(int) :ActionResult
:ModuleController
1.2: GetModuleQueryable() :IQueryable<Module>
2.2: GetModule(int) :Module
3.2: AddModule(Module)
2.4: UpdateModule(Module)
4.2: GetModule(int) :Module
4.4: DeleteModule(Module)
1.3: GetModuleData() :IQueryable<Module>
:ModuleService
:Module
2.3: GetModuleData() :Module
4.3: GetModuleData() :Module

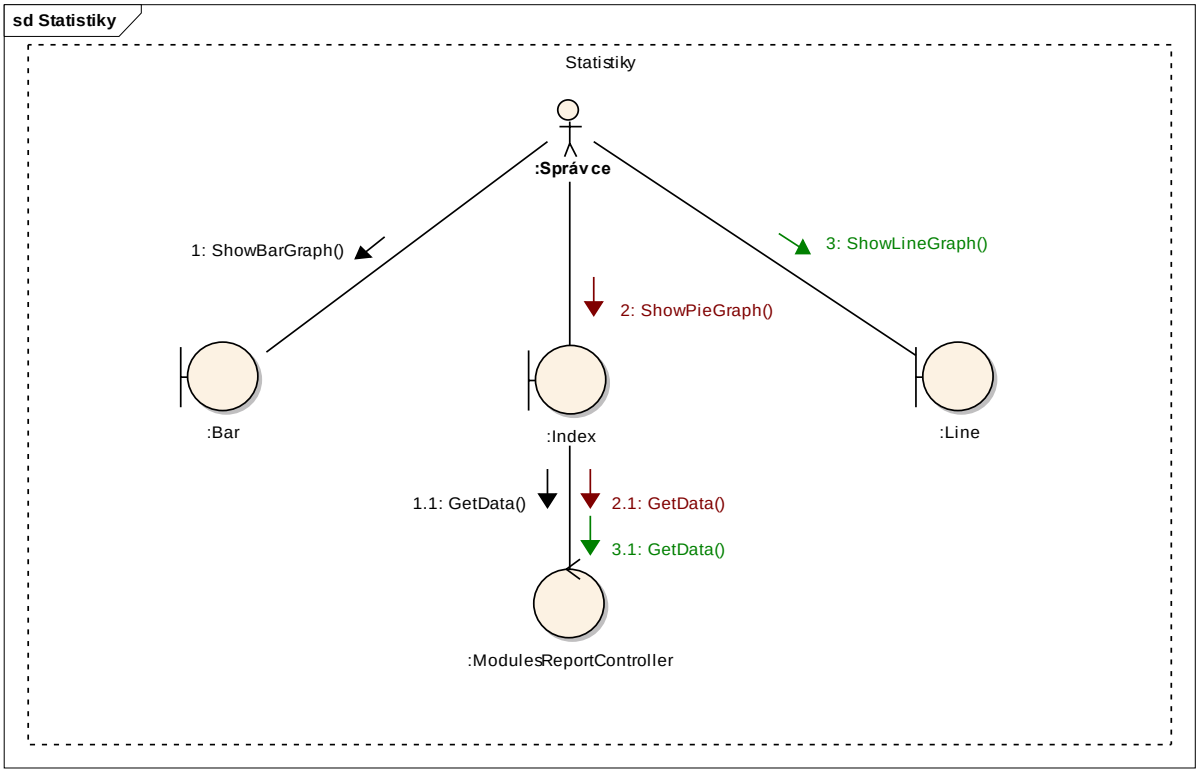
Moduly podmínky



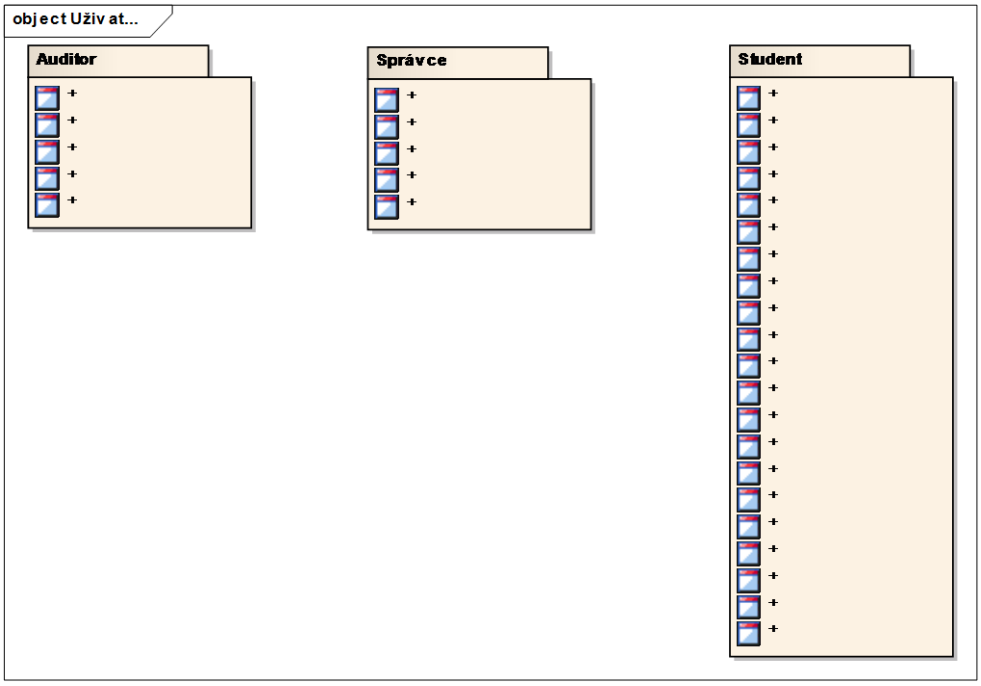
Místnosti



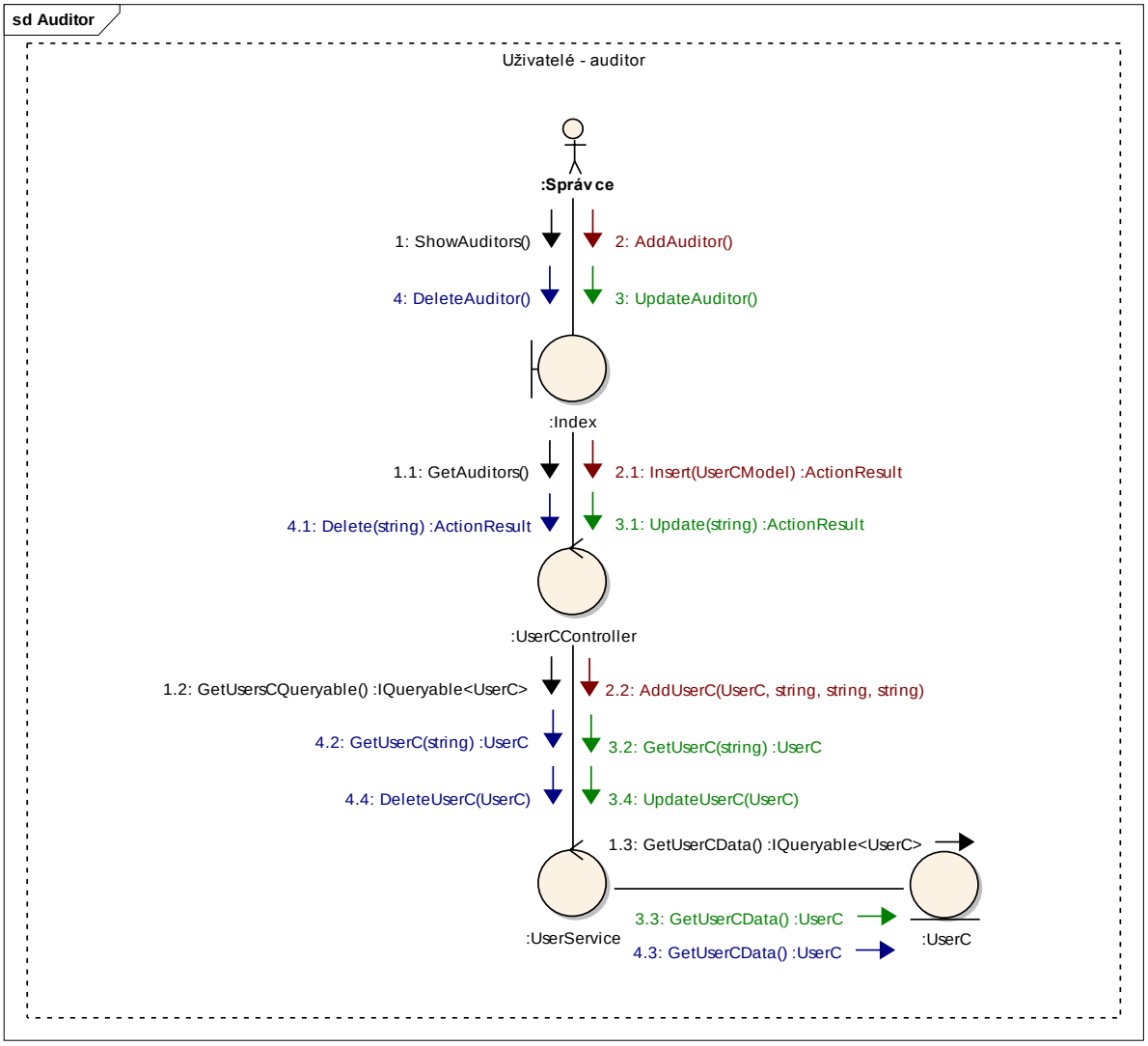
Statistiky



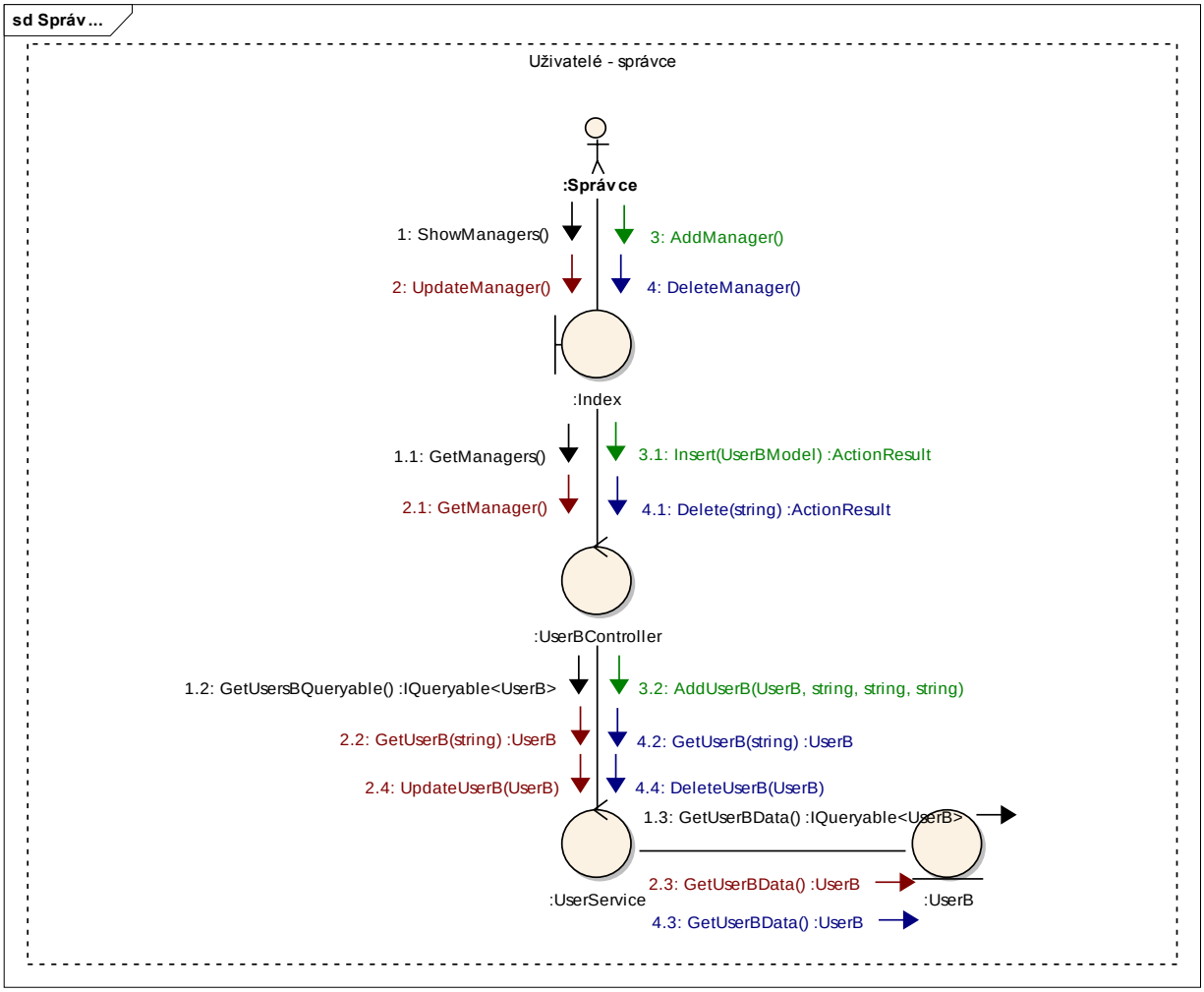
Uživatelé



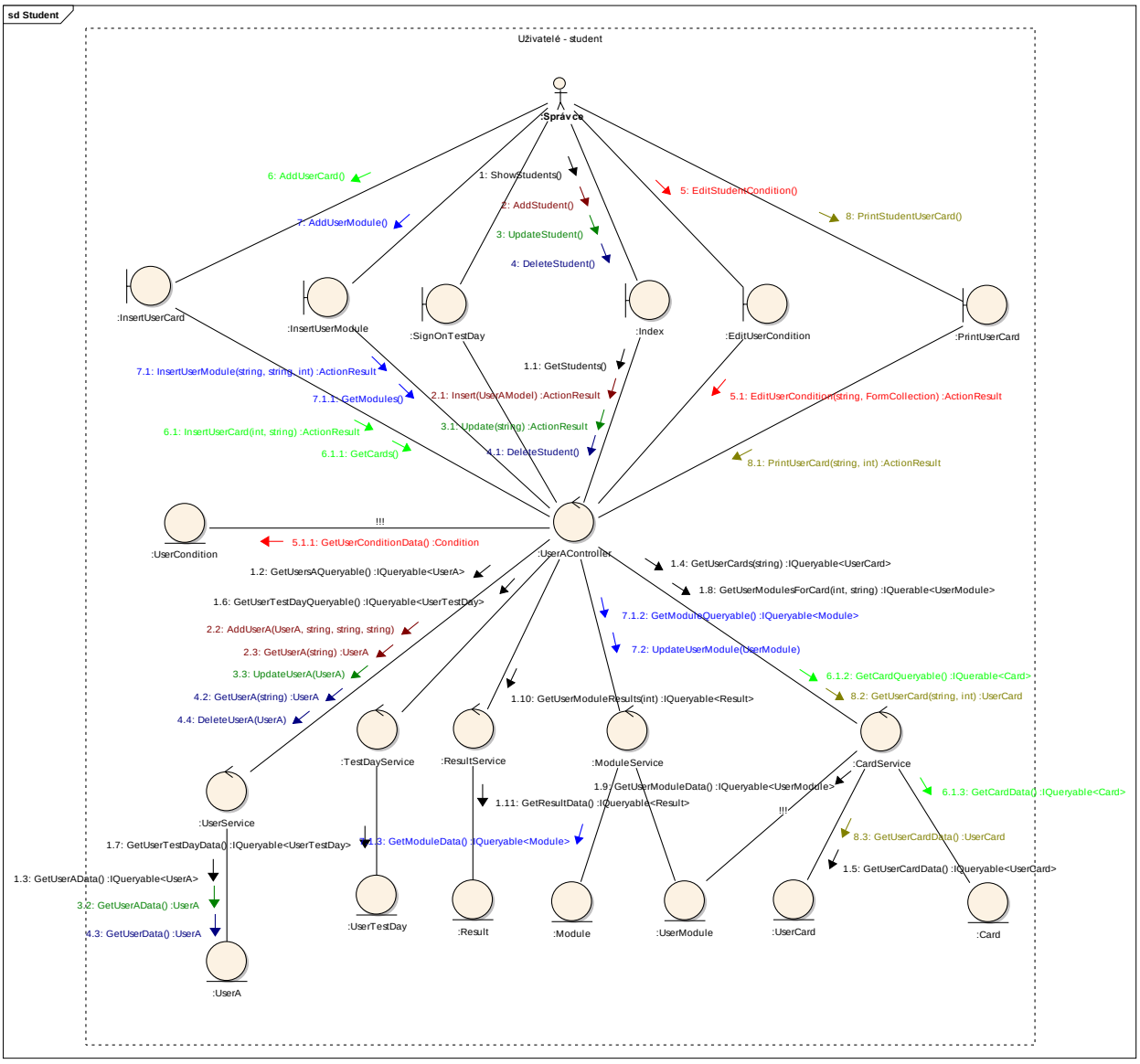
Auditor


sd Auditor
Uživatelé - auditor
:Správce
1: ShowAuditors()
2: AddAuditor()
4: DeleteAuditor()
3: UpdateAuditor()
:Index
1.1: GetAuditors()
2.1: Insert(UserCModel) :ActionResult
4.1: Delete(string) :ActionResult
3.1: Update(string) :ActionResult
:UserCController
1.2: GetUsersCQueryable() :IQueryable<UserC>
2.2: AddUserC(UserC, string, string, string)
4.2: GetUserC(string) :UserC
3.2: GetUserC(string) :UserC
4.4: DeleteUserC(UserC)
3.4: UpdateUserC(UserC)
1.3: GetUserCData() :IQueryable<UserC>
3.3: GetUserCData() :UserC
4.3: GetUserCData() :UserC
:UserService
:UserC

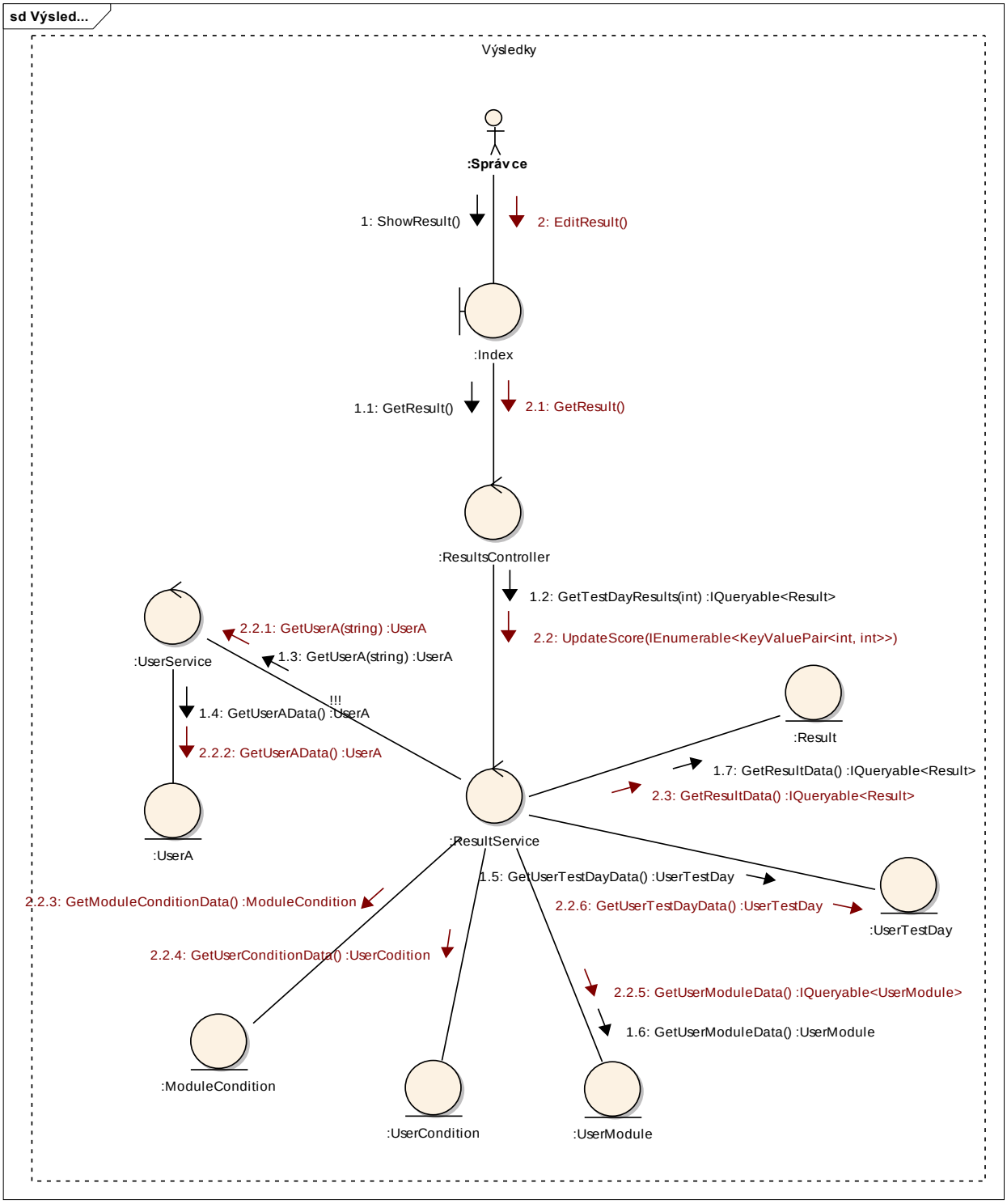
Správce


sd Správ...
Uživatelé - správce
:Správce
1: ShowManagers()
3: AddManager()
2: UpdateManager()
4: DeleteManager()
:Index
1.1: GetManagers()
3.1: Insert(UserBModel) :ActionResult
2.1: GetManager()
4.1: Delete(string) :ActionResult
:UserBController
1.2: GetUsersBQueryable() :IQueryable<UserB>
3.2: AddUserB(UserB, string, string, string)
2.2: GetUserB(string) :UserB
4.2: GetUserB(string) :UserB
2.4: UpdateUserB(UserB)
4.4: DeleteUserB(UserB)
1.3: GetUserBData() :IQueryable<UserB>
2.3: GetUserBData() :UserB
:UserService
:UserB
4.3: GetUserBData() :UserB

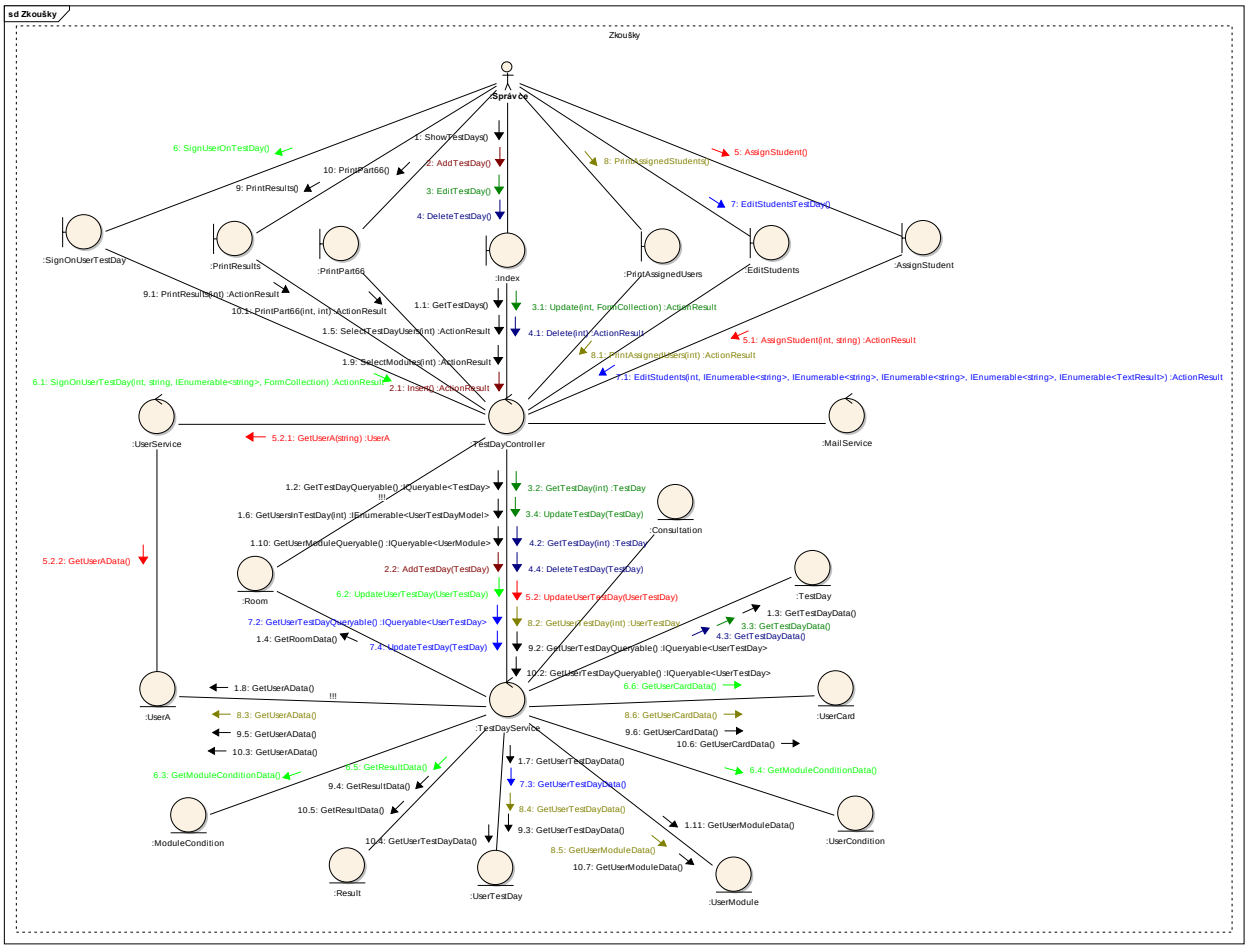
Student


sd Student
Uživatelé - student
:Správce
6: AddUserCard()
1: ShowStudents()
5: EditStudentCondition()
8: PrintStudentUserCard()
7: AddUserModule()
2: AddStudent()
3: UpdateStudent()
4: DeleteStudent()
:InsertUserCard
:InsertUserModule
:SignOnTestDay
:Index
:EditUserCondition
:PrintUserCard
1.1: GetStudents()
7.1: InsertUserModule(string, string, int) :ActionResult
7.1.1: GetModules()
2.1: Insert(UserAModel) :ActionResult
5.1: EditUserCondition(string, FormCollection) :ActionResult
3.1: Update(string) :ActionResult
6.1: InsertUserCard(int, string) :ActionResult
6.1.1: GetCards()
4.1: DeleteStudent()
8.1: PrintUserCard(string, int) :ActionResult
:UserCondition
!!!
5.1.1: GetUserConditionData() :Condition
:UserAController
1.4: GetUserCards(string) :IQueryable<UserCard>
1.2: GetUsersAQueryable() :IQueryable<UserA>
1.8: GetUserModulesForCard(int, string) :IQuerable<UserModule>
1.6: GetUserTestDayQueryable() :IQueryable<UserTestDay>
7.1.2: GetModuleQueryable() :IQueryable<Module>
2.2: AddUserA(UserA, string, string, string)
7.2: UpdateUserModule(UserModule)
2.3: GetUserA(string) :UserA
6.1.2: GetCardQueryable() :IQuerable<Card>
3.3: UpdateUserA(UserA)
8.2: GetUserCard(string, int) :UserCard
4.2: GetUserA(string) :UserA
1.10: GetUserModuleResults(int) :IQueryable<Result>
4.4: DeleteUserA(UserA)
:TestDayService
:ResultService
:ModuleService
:CardService
:UserService
1.9: GetUserModuleData() :IQueryable<UserModule>
6.1.3: GetCardData() :IQueryable<Card>
1.11: GetResultData() :IQueryable<Result>
!!!
1.7: GetUserTestDayData() :IQueryable<UserTestDay>
1.13: GetModuleData() :IQueryable<Module>
8.3: GetUserCardData() :UserCard
1.5: GetUserCardData() :IQueryable<UserCard>
1.3: GetUserAData() :IQueryable<UserA>
3.2: GetUserAData() :UserA
4.3: GetUserData() :UserA
:UserTestDay
:Result
:Module
:UserModule
:UserCard
:Card
:UserA

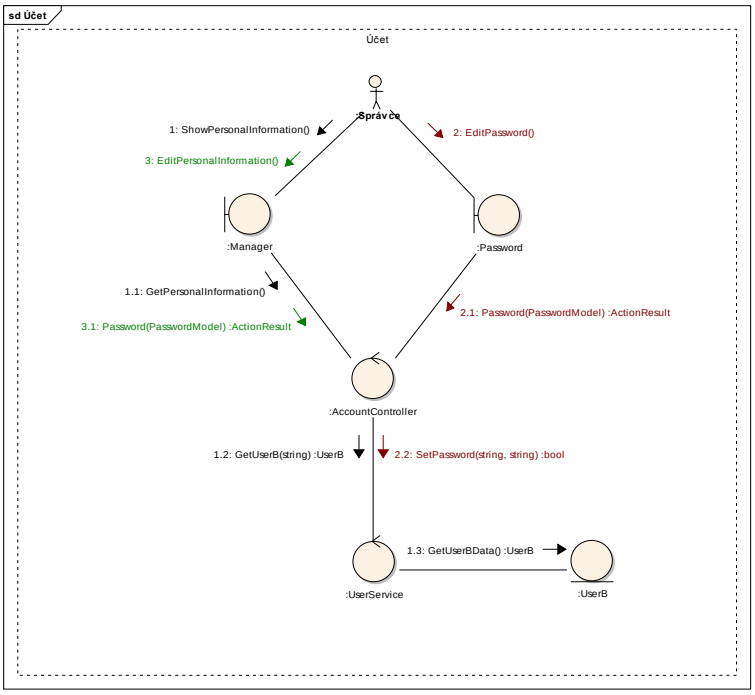
Výsledky


sd Výsled...
Výsledky
:Správce
1: ShowResult()
2: EditResult()
:Index
1.1: GetResult()
2.1: GetResult()
:ResultsController
1.2: GetTestDayResults(int) :IQueryable<Result>
2.2: UpdateScore(IEnumerable<KeyValuePair<int, int>>)
:UserService
2.2.1: GetUserA(string) :UserA
1.3: GetUserA(string) :UserA
!!!
1.4: GetUserAData() :UserA
2.2.2: GetUserAData() :UserA
:UserA
:Result
1.7: GetResultData() :IQueryable<Result>
2.3: GetResultData() :IQueryable<Result>
:ResultService
1.5: GetUserTestDayData() :UserTestDay
2.2.6: GetUserTestDayData() :UserTestDay
:UserTestDay
2.2.3: GetModuleConditionData() :ModuleCondition
2.2.4: GetUserConditionData() :UserCodition
2.2.5: GetUserModuleData() :IQueryable<UserModule>
1.6: GetUserModuleData() :UserModule
:ModuleCondition
:UserCondition
:UserModule

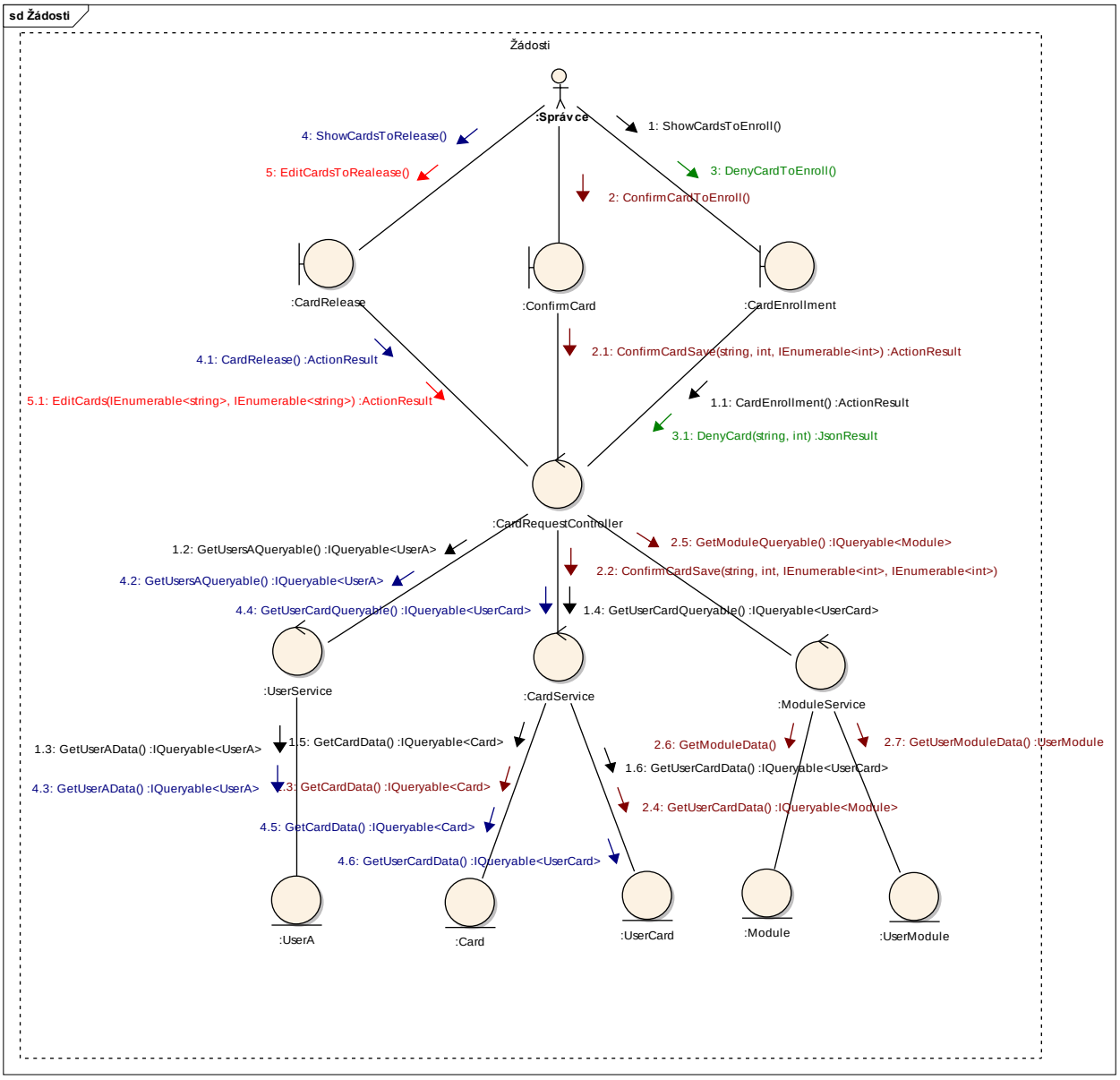
Zkoušky



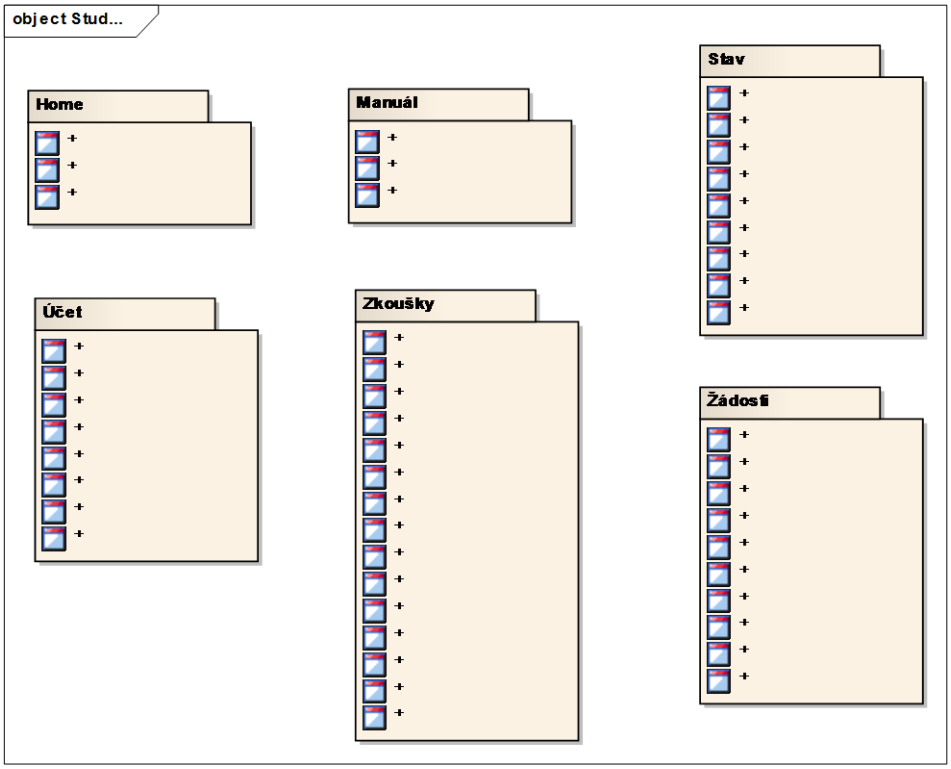
Účet



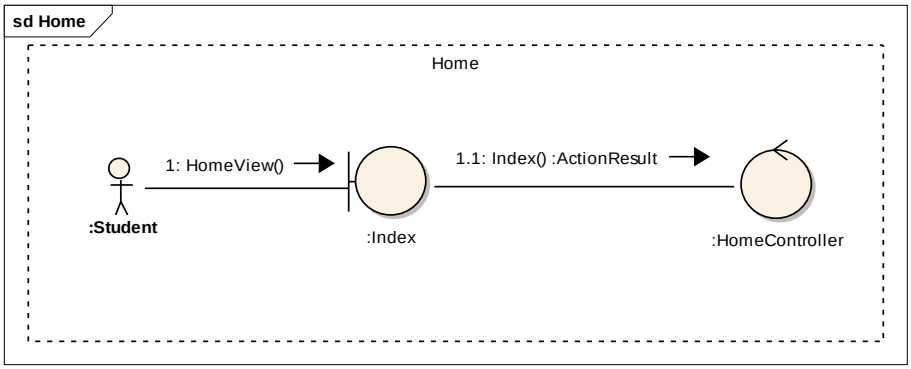
Žádosti


sd Žádosti
Žádosti
:Správce
1: ShowCardsToEnroll()
2: ConfirmCardToEnroll()
3: DenyCardToEnroll()
4: ShowCardsToRelease()
5: EditCardsToRealease()
:CardRelease
:ConfirmCard
:CardEnrollment
2.1: ConfirmCardSave(string, int, IEnumerable<int>) :ActionResult
4.1: CardRelease() :ActionResult
5.1: EditCards(IEnumerable<string>, IEnumerable<string>) :ActionResult
1.1: CardEnrollment() :ActionResult
3.1: DenyCard(string, int) :JsonResult
:CardRequestController
1.2: GetUsersAQueryable() :IQueryable<UserA>
2.5: GetModuleQueryable() :IQueryable<Module>
2.2: ConfirmCardSave(string, int, IEnumerable<int>, IEnumerable<int>)
4.2: GetUsersAQueryable() :IQueryable<UserA>
4.4: GetUserCardQueryable() :IQueryable<UserCard>
1.4: GetUserCardQueryable() :IQueryable<UserCard>
:UserService
:CardService
:ModuleService
1.3: GetUserAData() :IQueryable<UserA>
1.5: GetCardData() :IQueryable<Card>
2.6: GetModuleData()
2.7: GetUserModuleData() :UserModule
1.6: GetUserCardData() :IQueryable<UserCard>
4.3: GetUserAData() :IQueryable<UserA>
2.3: GetCardData() :IQueryable<Card>
2.4: GetUserCardData() :IQueryable<Module>
4.5: GetCardData() :IQueryable<Card>
4.6: GetUserCardData() :IQueryable<UserCard>
:UserA
:Card
:UserCard
:Module
:UserModule

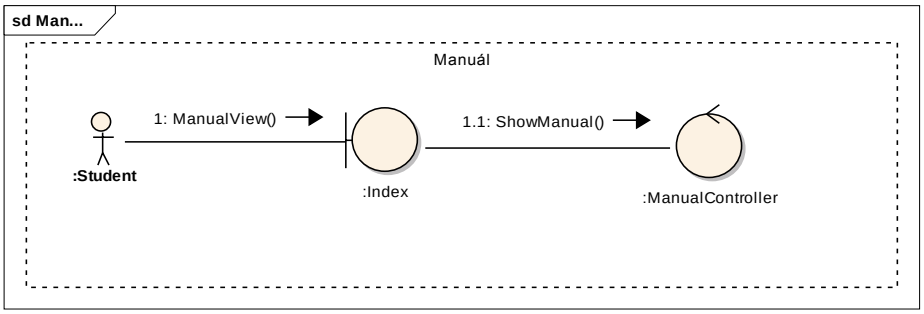
Student



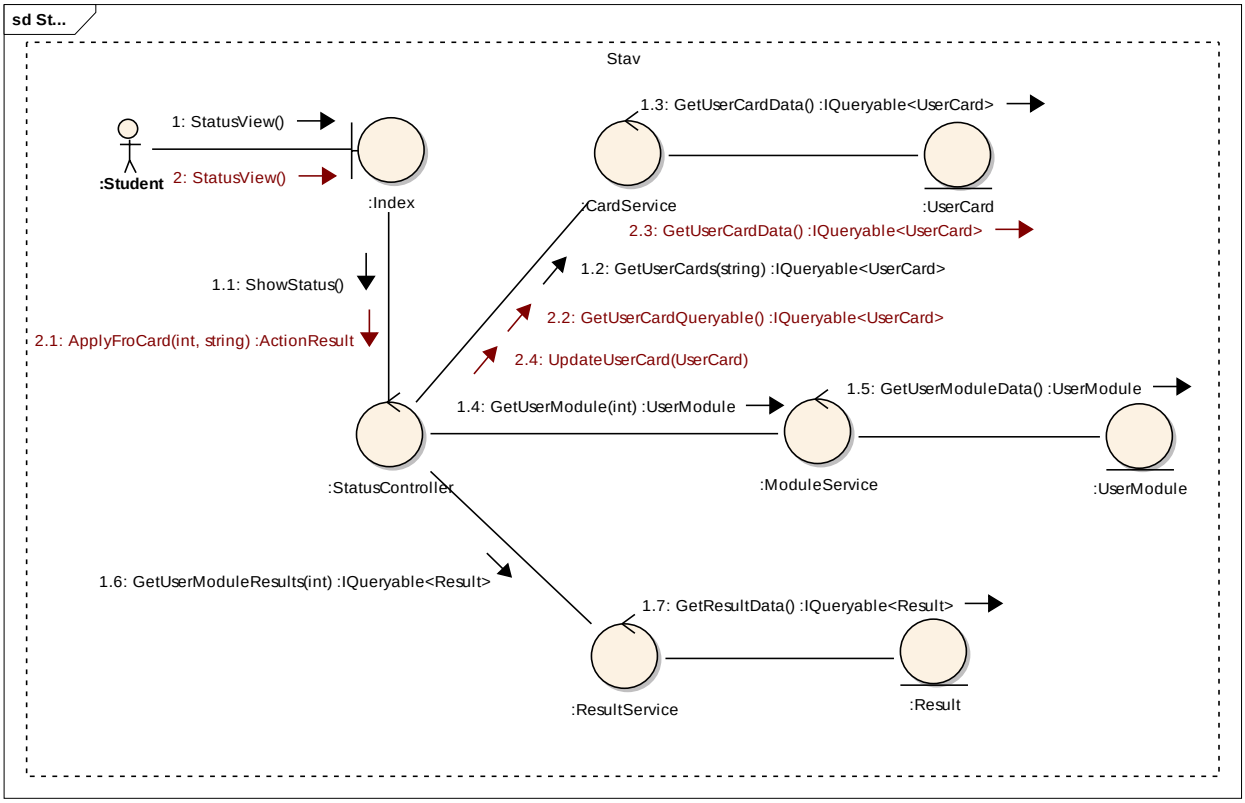
Home



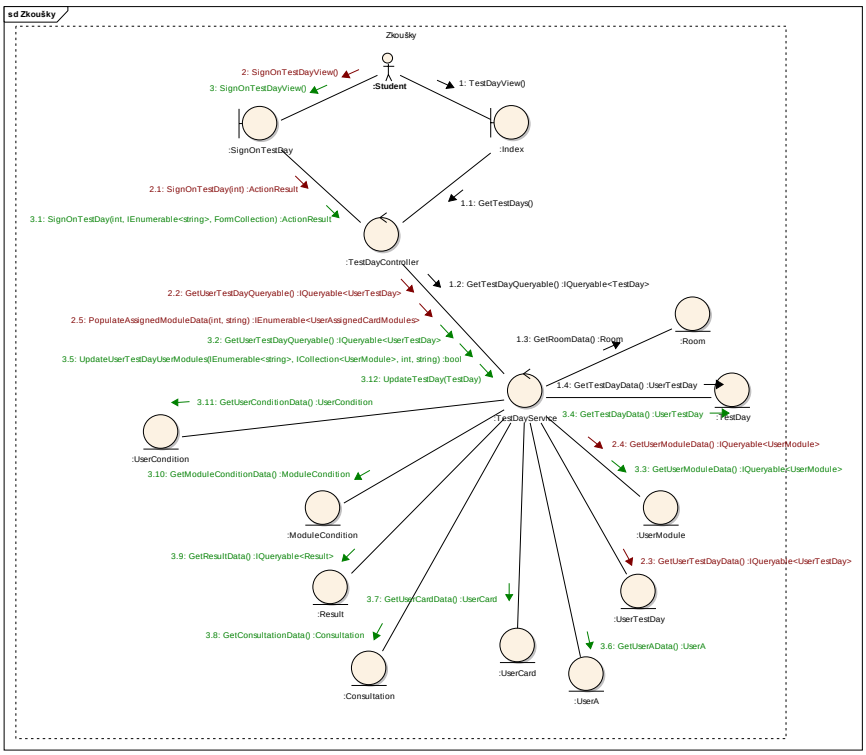
Manuál



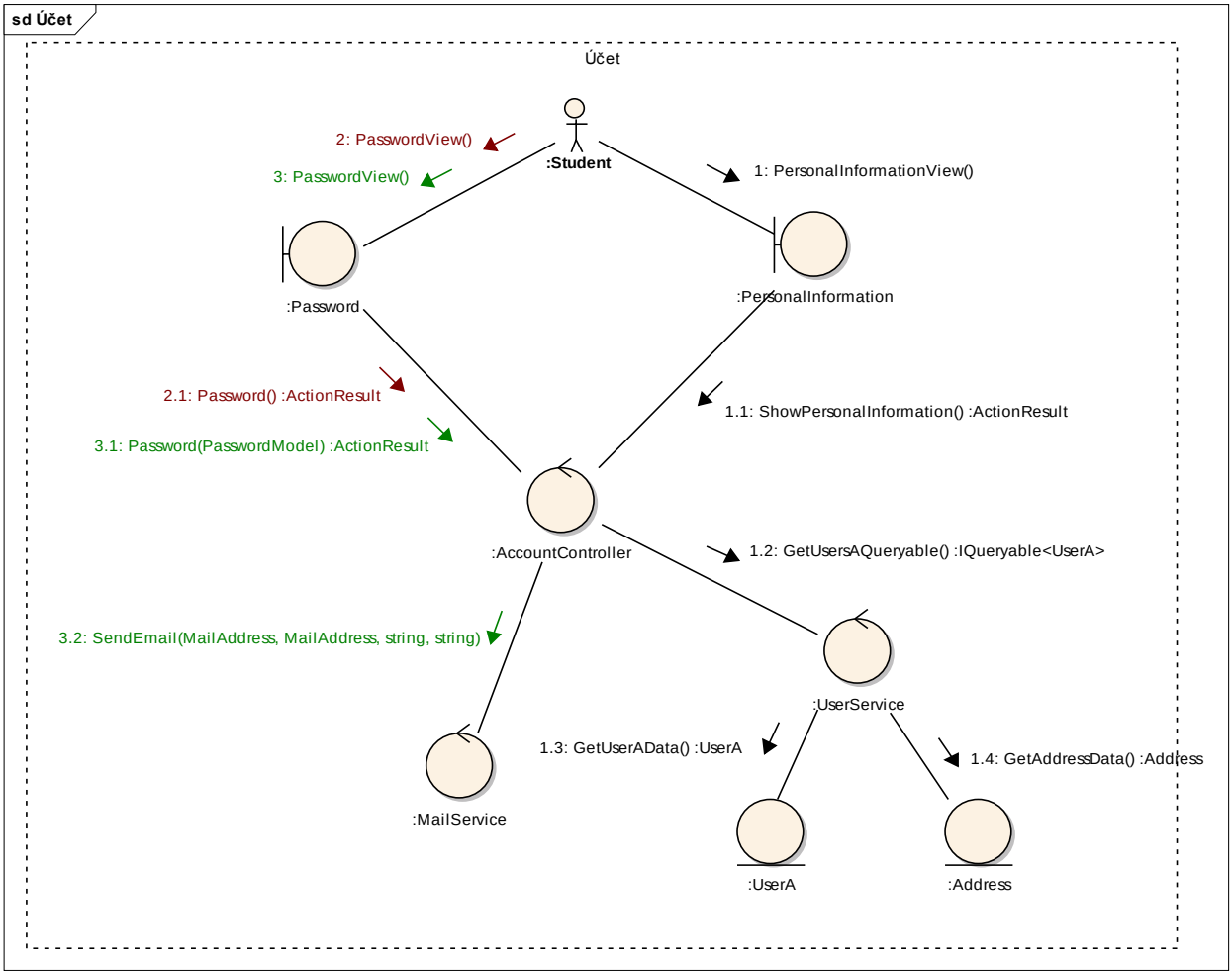
Stav



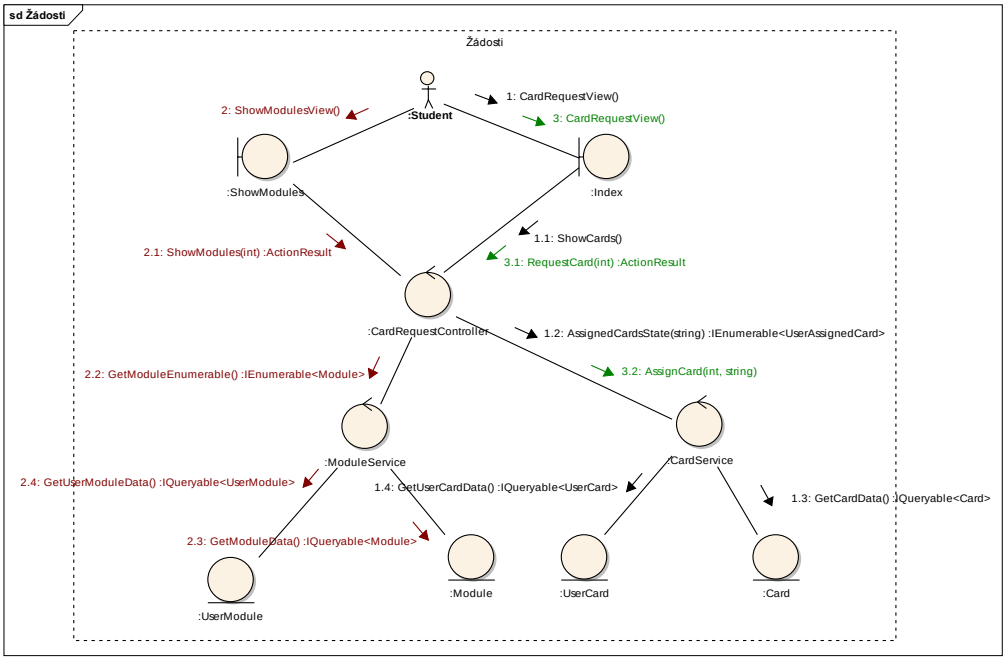
Zkoušky



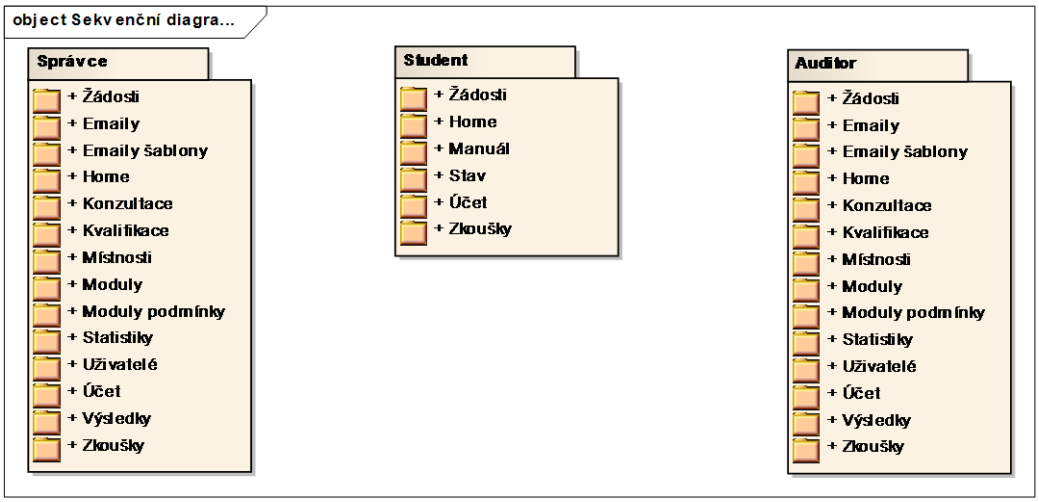
Účet



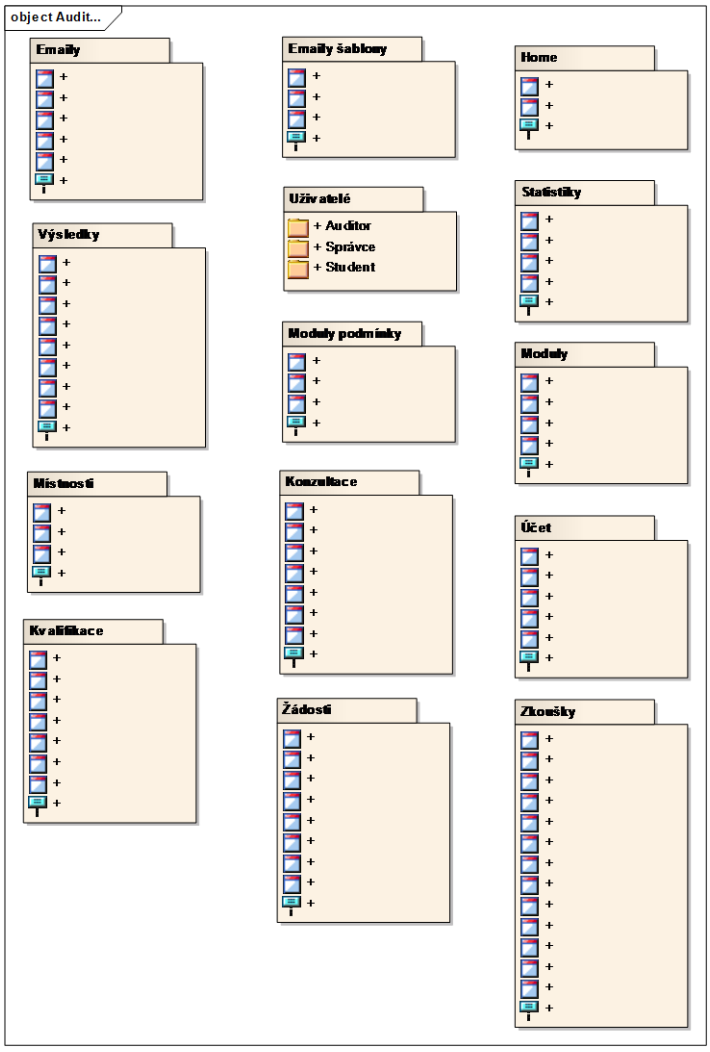
Žádosti



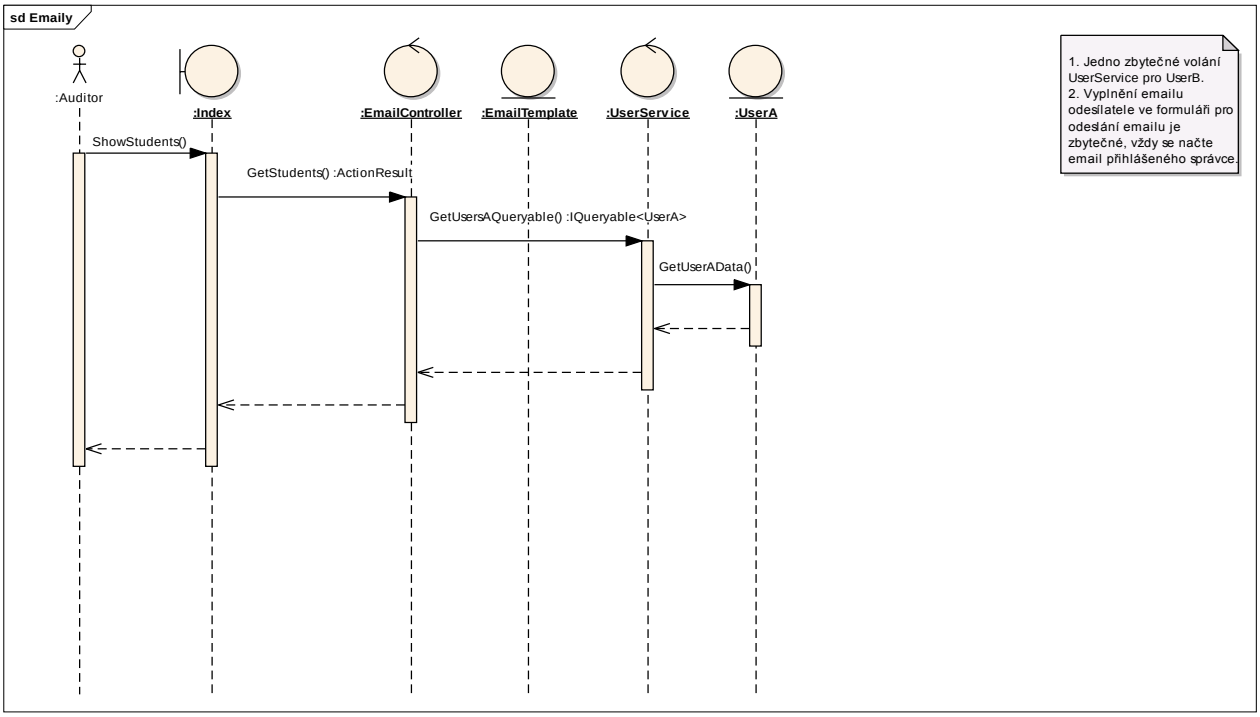
PŘÍLOHA J: Sekvenční diagramy



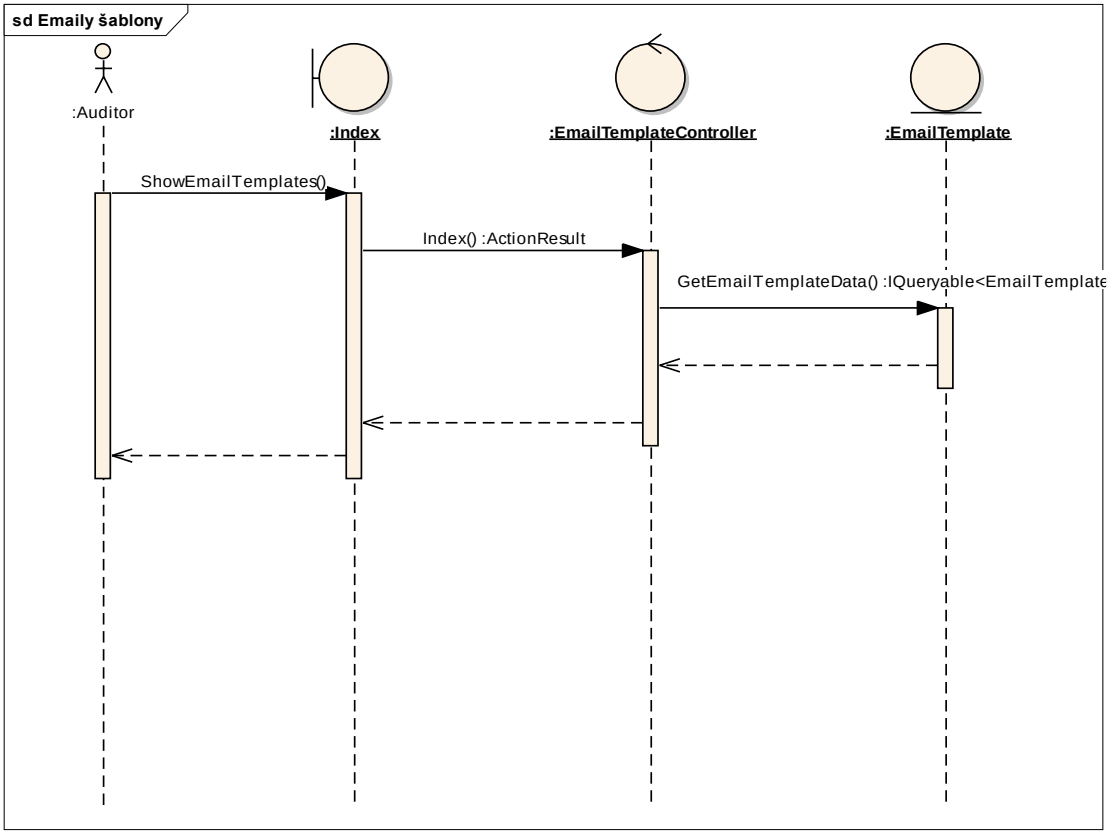
Auditor



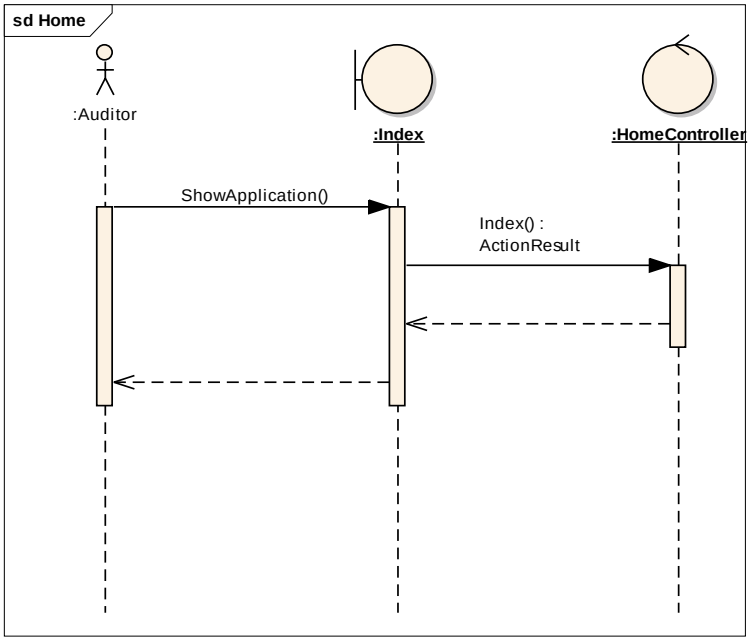
Emaily



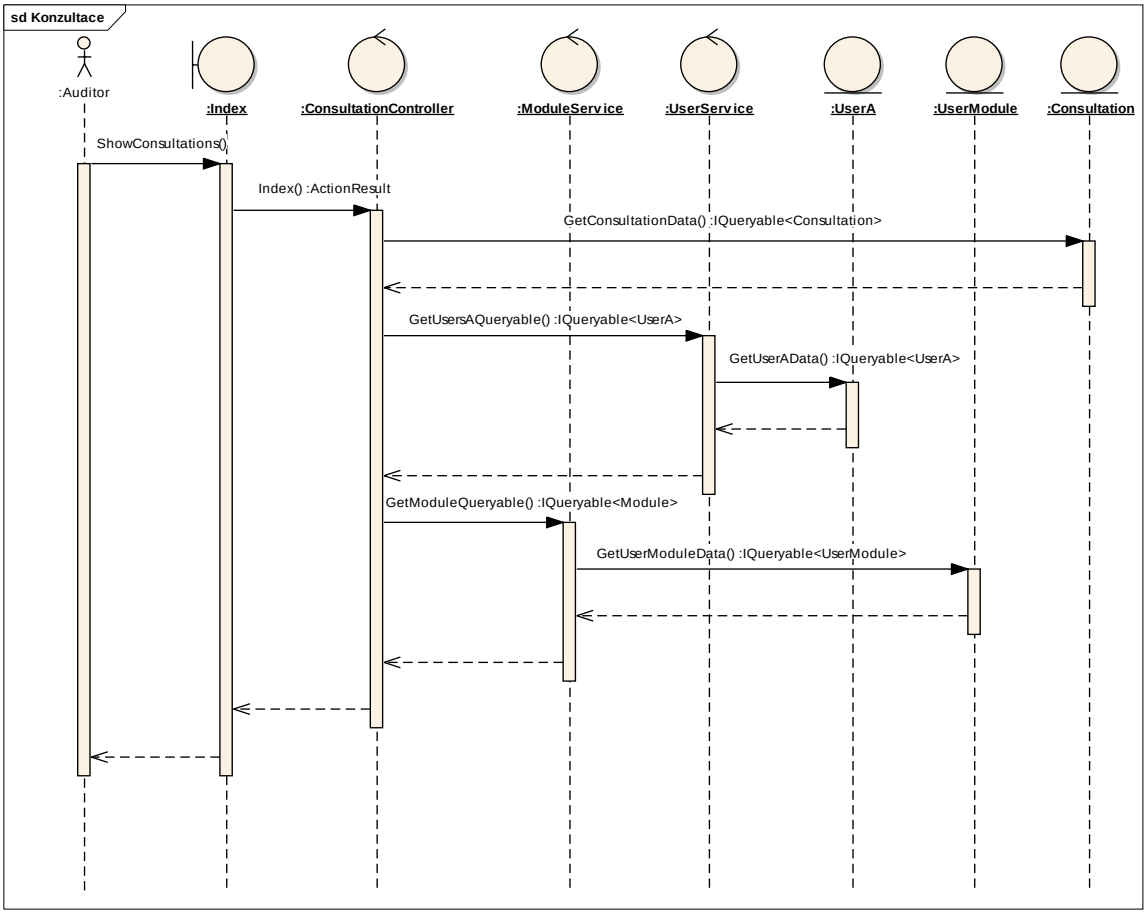
Emaily šablony



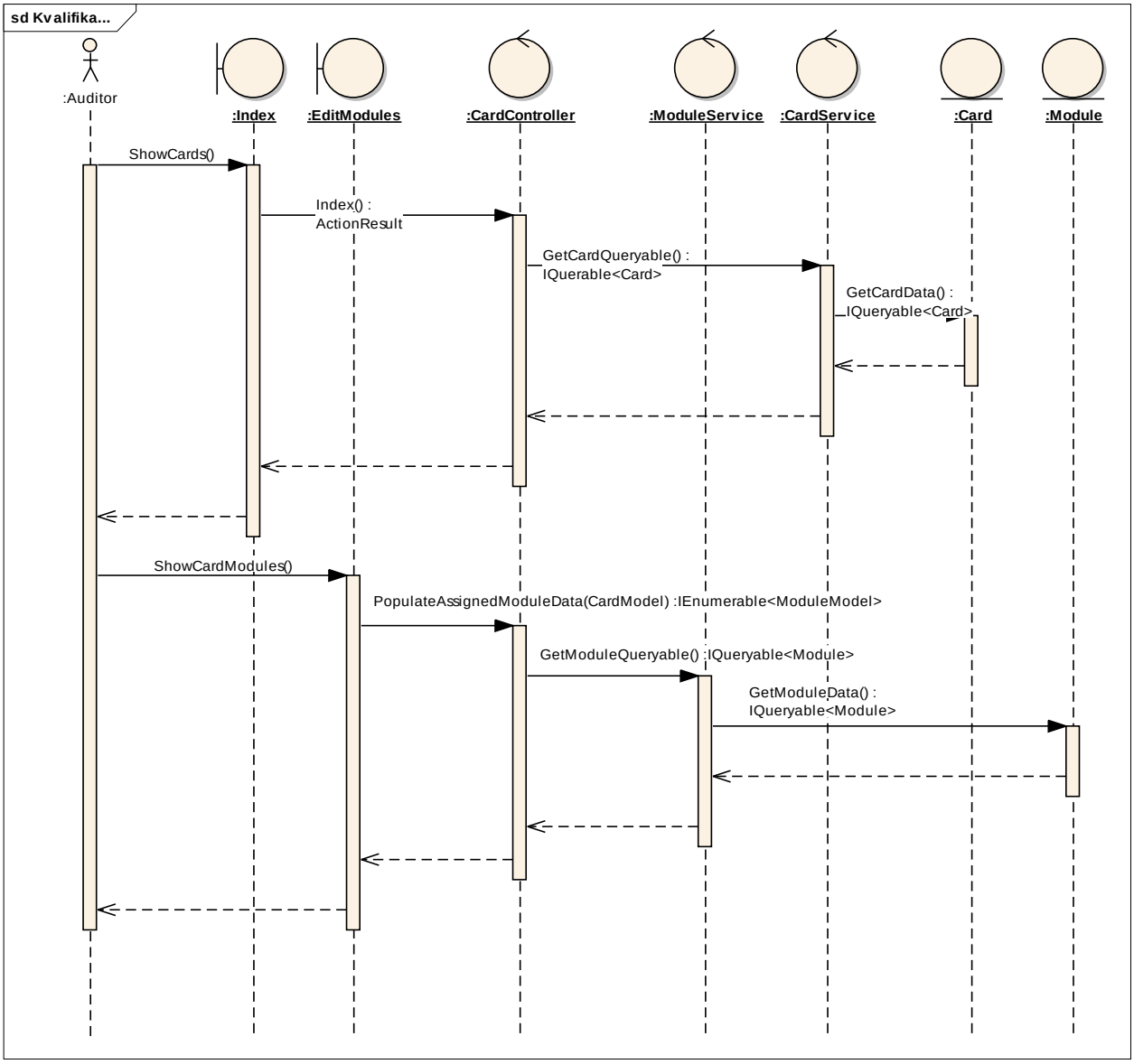
Home



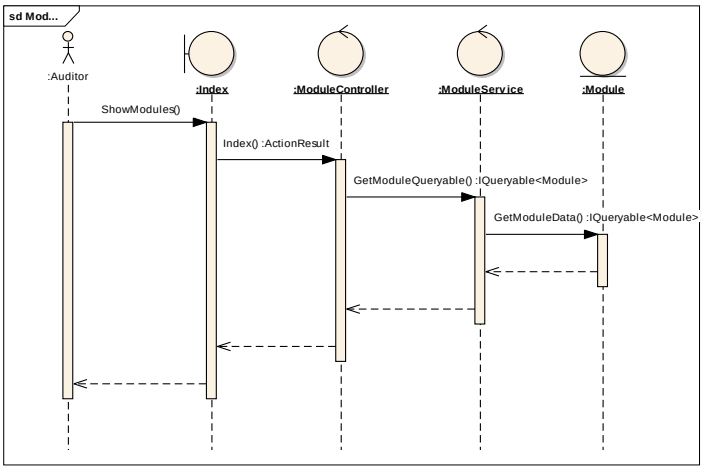
Konzultace



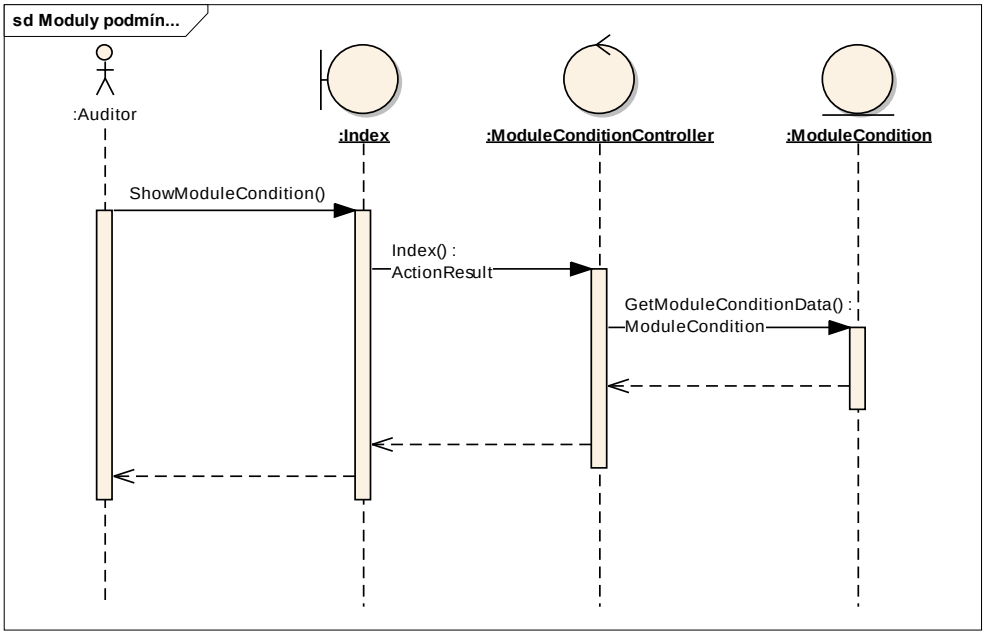
Kvalifikace



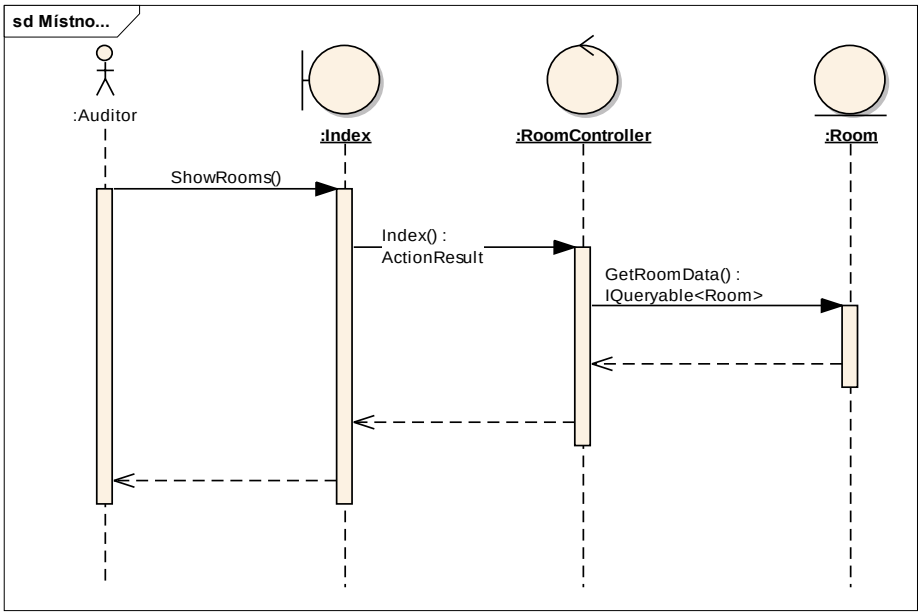
Moduly



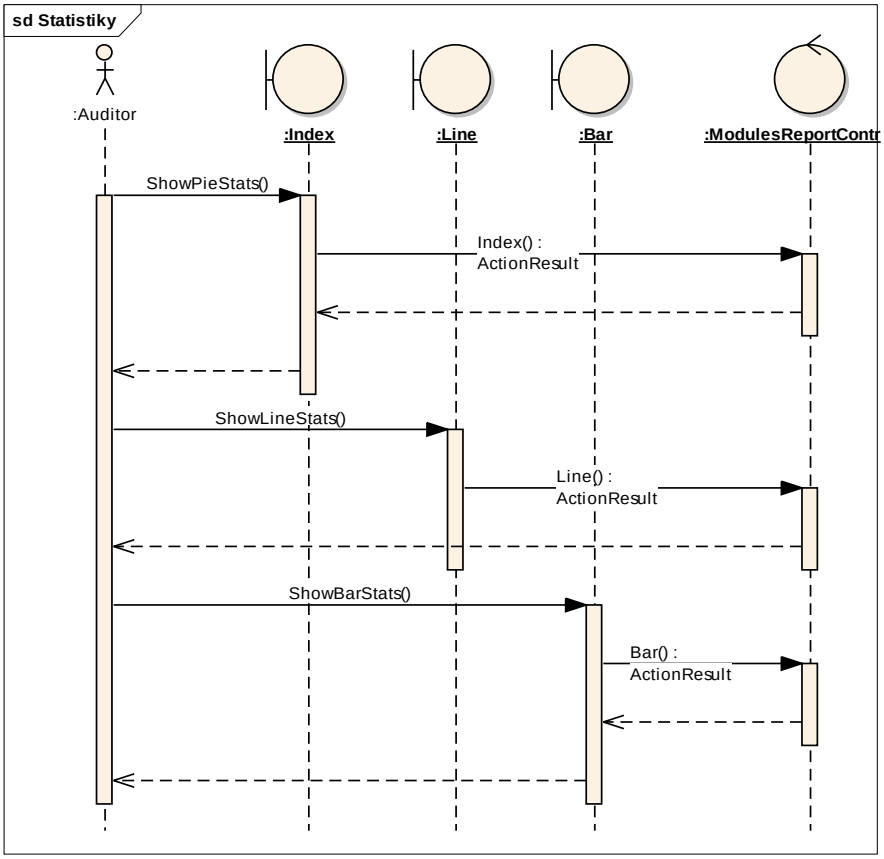
Moduly podmínky



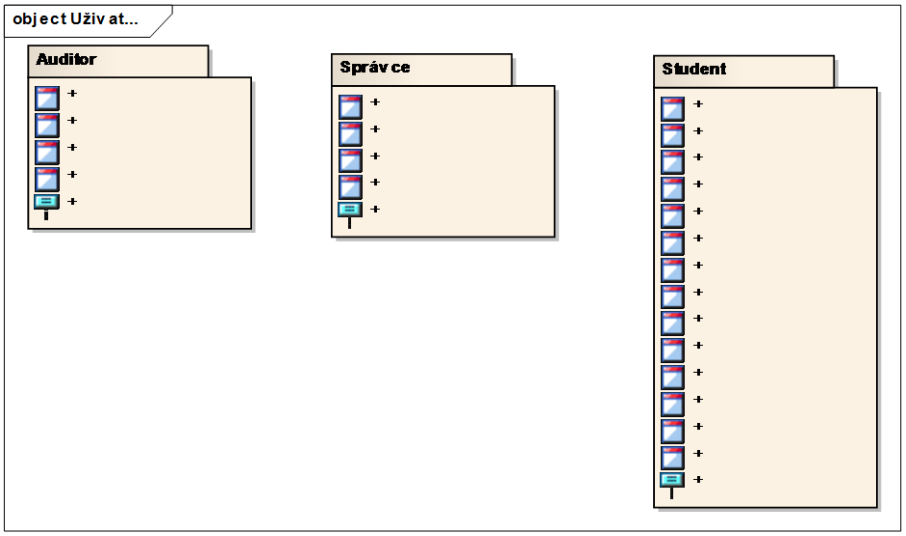
Místnosti



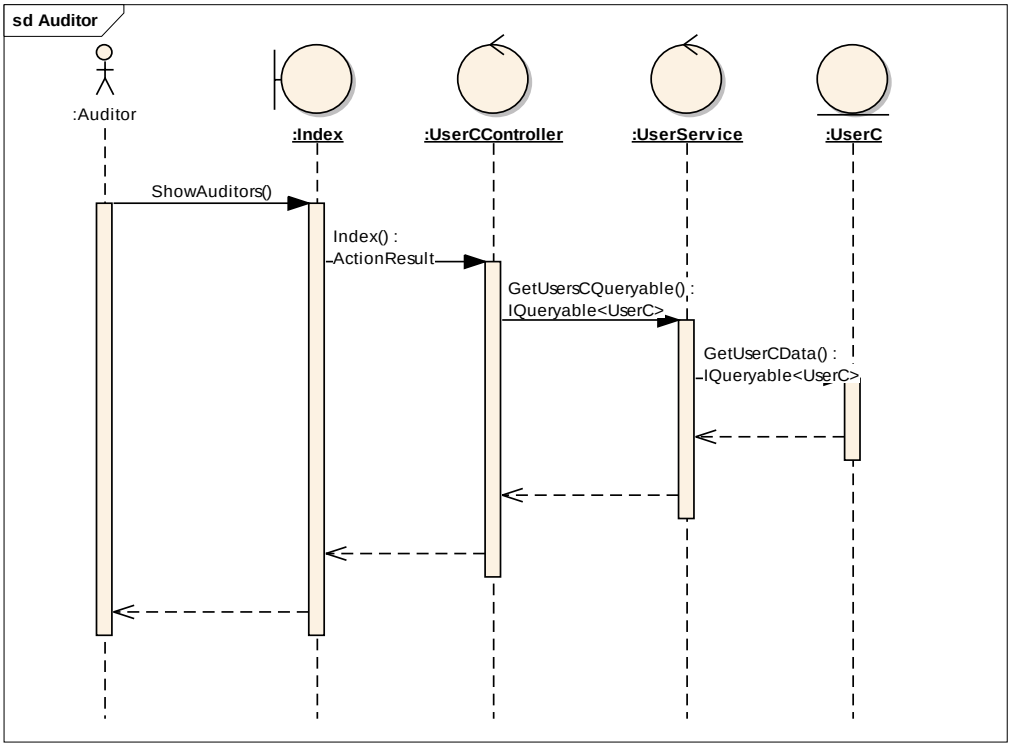
Statistiky



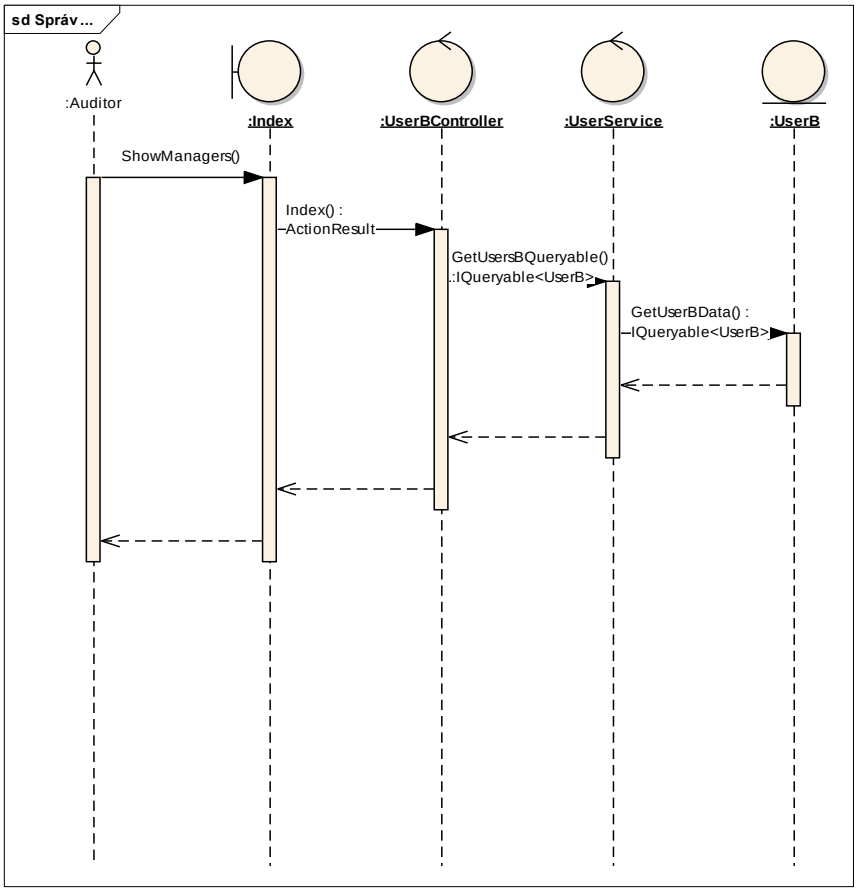
Uživatelé



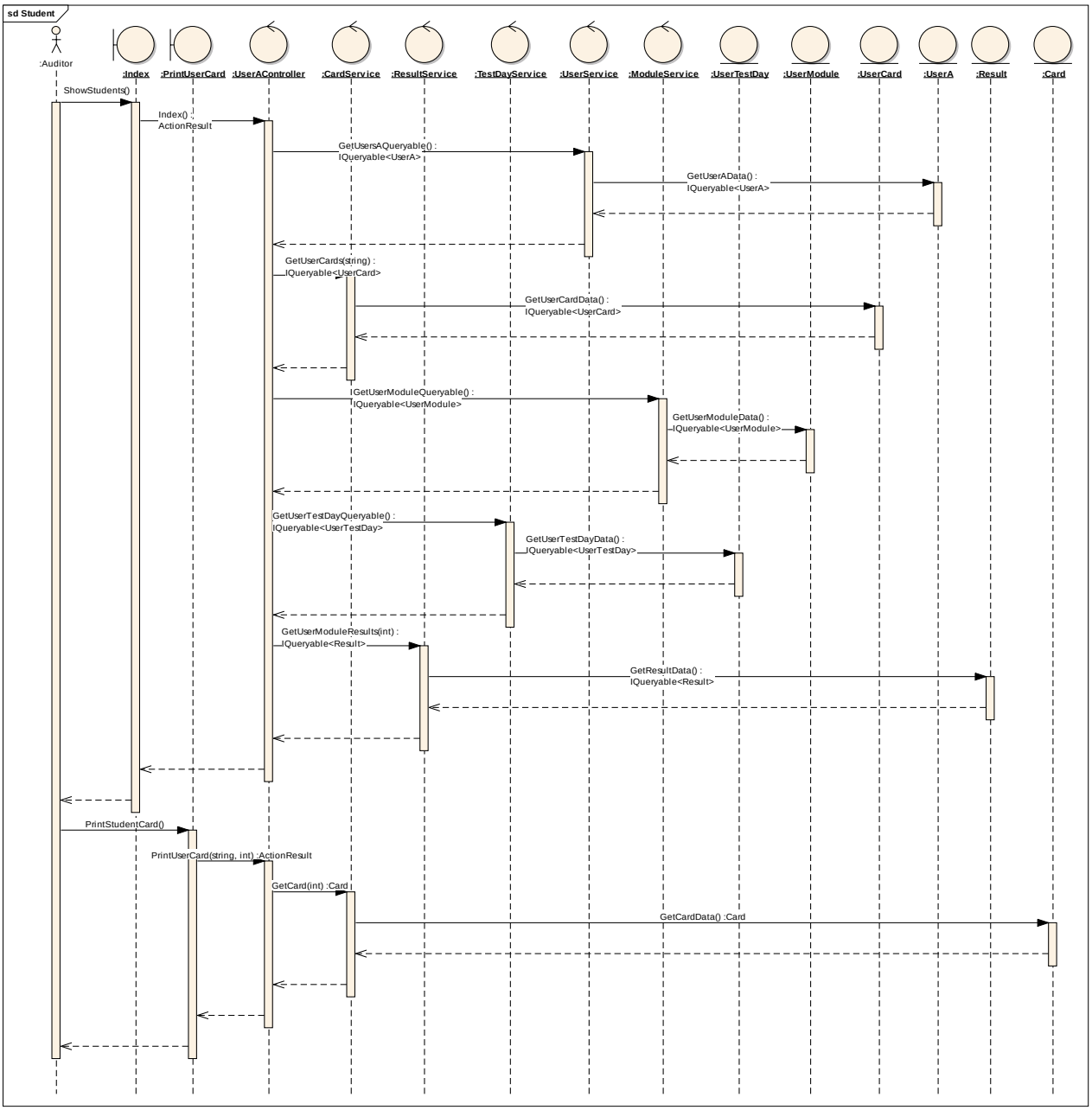
Auditor



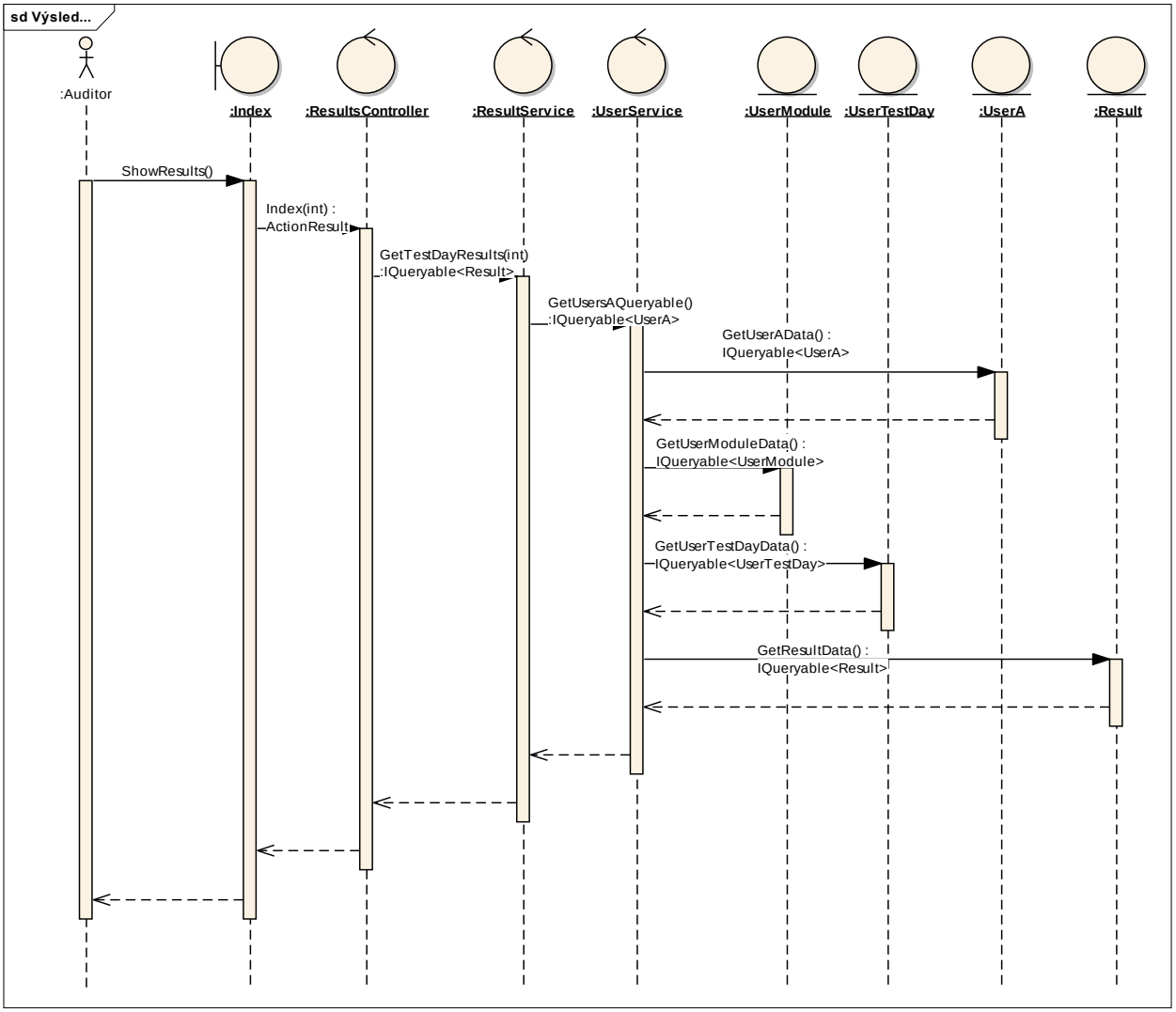
Správce



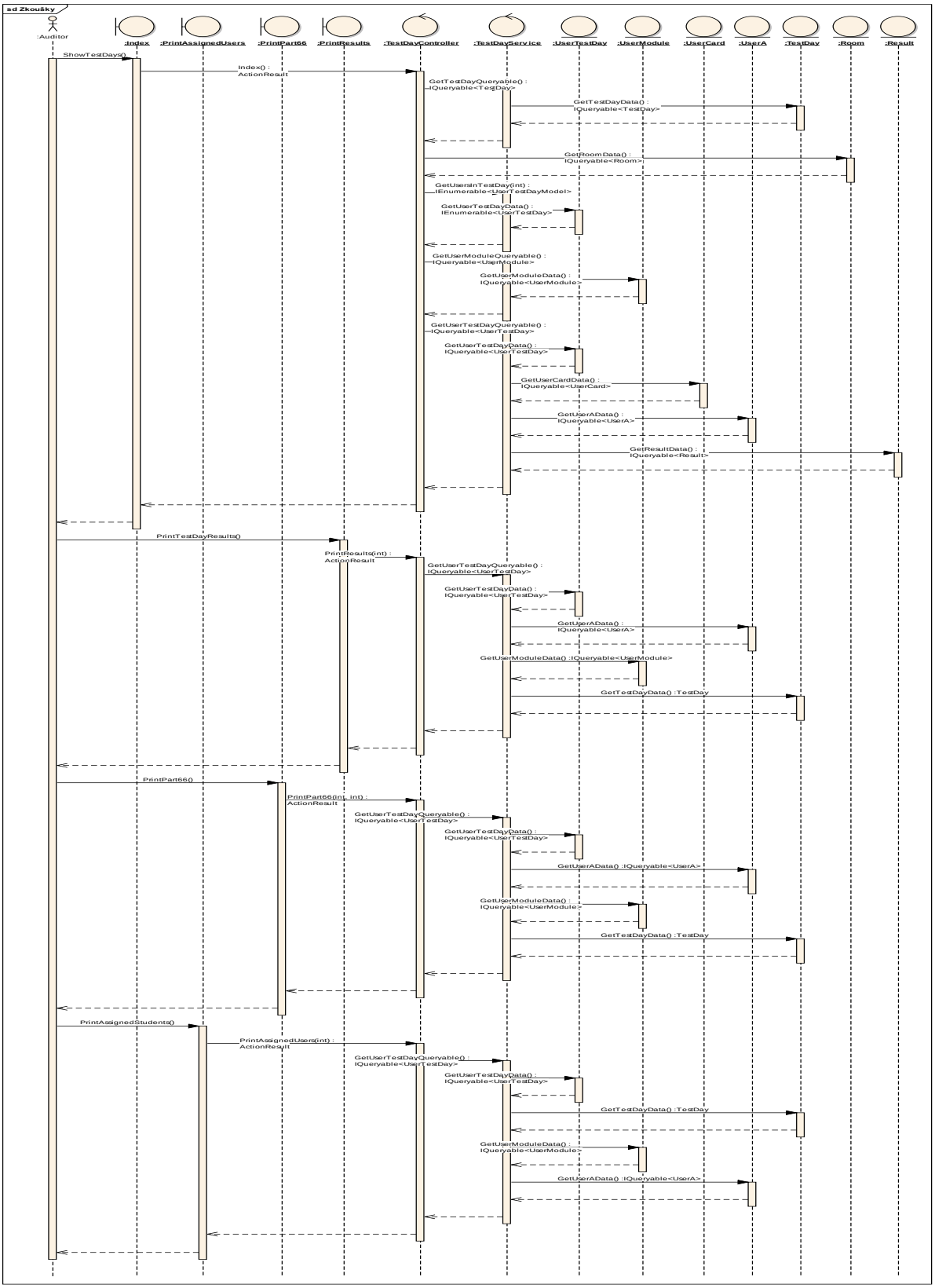
Student


sd Student
:Auditor
:Index
:PrintUserCard
:UserAController
:CardService
:ResultService
:TestDayService
:UserService
:ModuleService
:UserTestDay
:UserModule
:UserCard
:UserA
:Result
:Card
ShowStudents()
Index() : ActionResult
GetUsersAQueryable() : IQueryable<UserA>
GetUserAData() : IQueryable<UserA>
GetUserCards(string) : IQueryable<UserCard>
GetUserCardData() : IQueryable<UserCard>
GetUserModuleQueryable() : IQueryable<UserModule>
GetUserModuleData() : IQueryable<UserModule>
GetUserTestDayQueryable() : IQueryable<UserTestDay>
GetUserTestDayData() : IQueryable<UserTestDay>
GetUserModuleResults(int) : IQueryable<Result>
GetResultData() : IQueryable<Result>
PrintStudentCard()
PrintUserCard(string, int) :ActionResult
GetCard(int) :Card
GetCardData() :Card

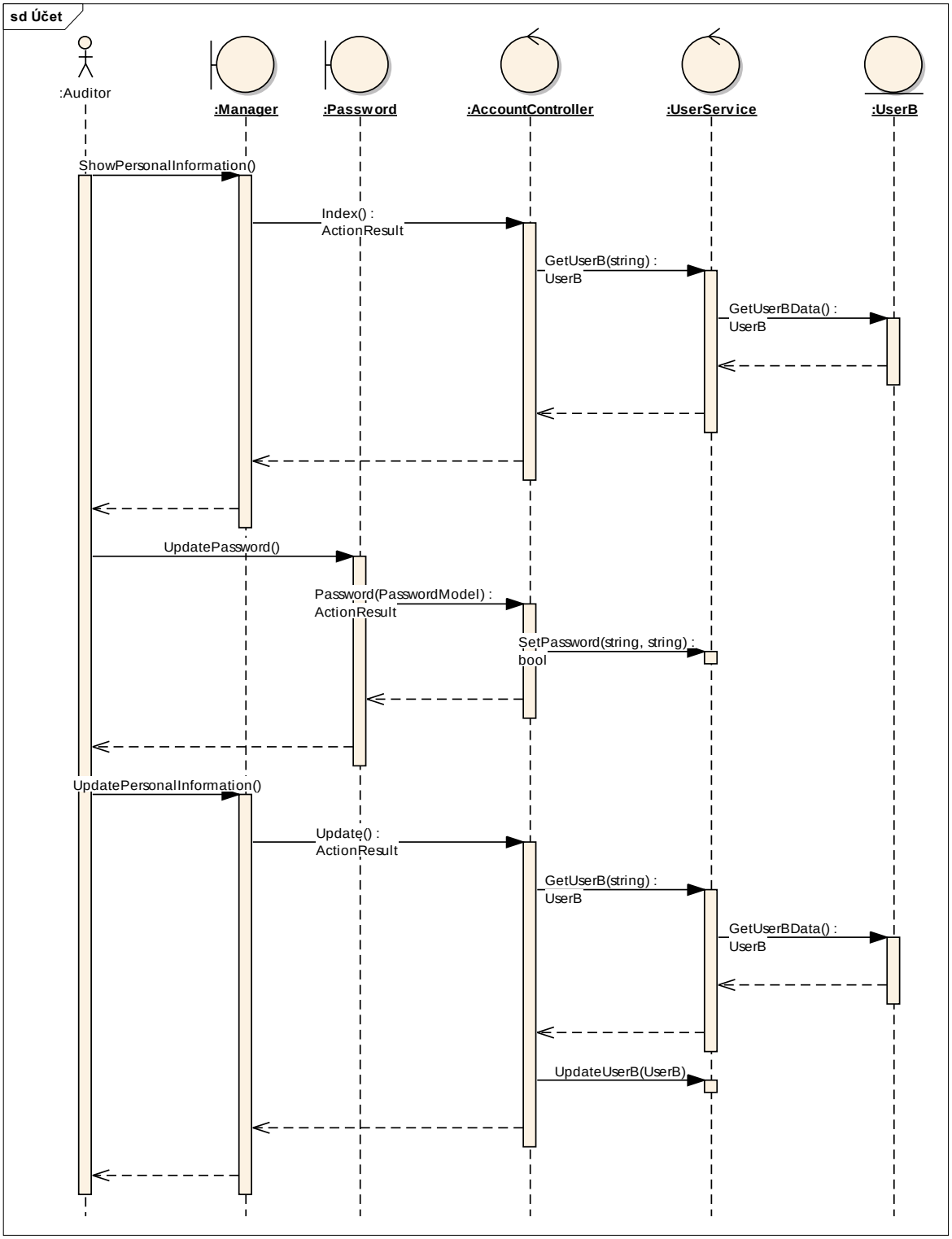
Výsledky


sd Výsled...
:Auditor
:Index
:ResultsController
:ResultService
:UserService
:UserModule
:UserTestDay
:UserA
:Result
ShowResults()
Index(int) :
ActionResult
GetTestDayResults(int)
:IQueryable<Result>
GetUsersAQueryable()
:IQueryable<UserA>
GetUserAData() :
IQueryable<UserA>
GetUserModuleData() :
IQueryable<UserModule>
GetUserTestDayData() :
IQueryable<UserTestDay>
GetResultData() :
IQueryable<Result>

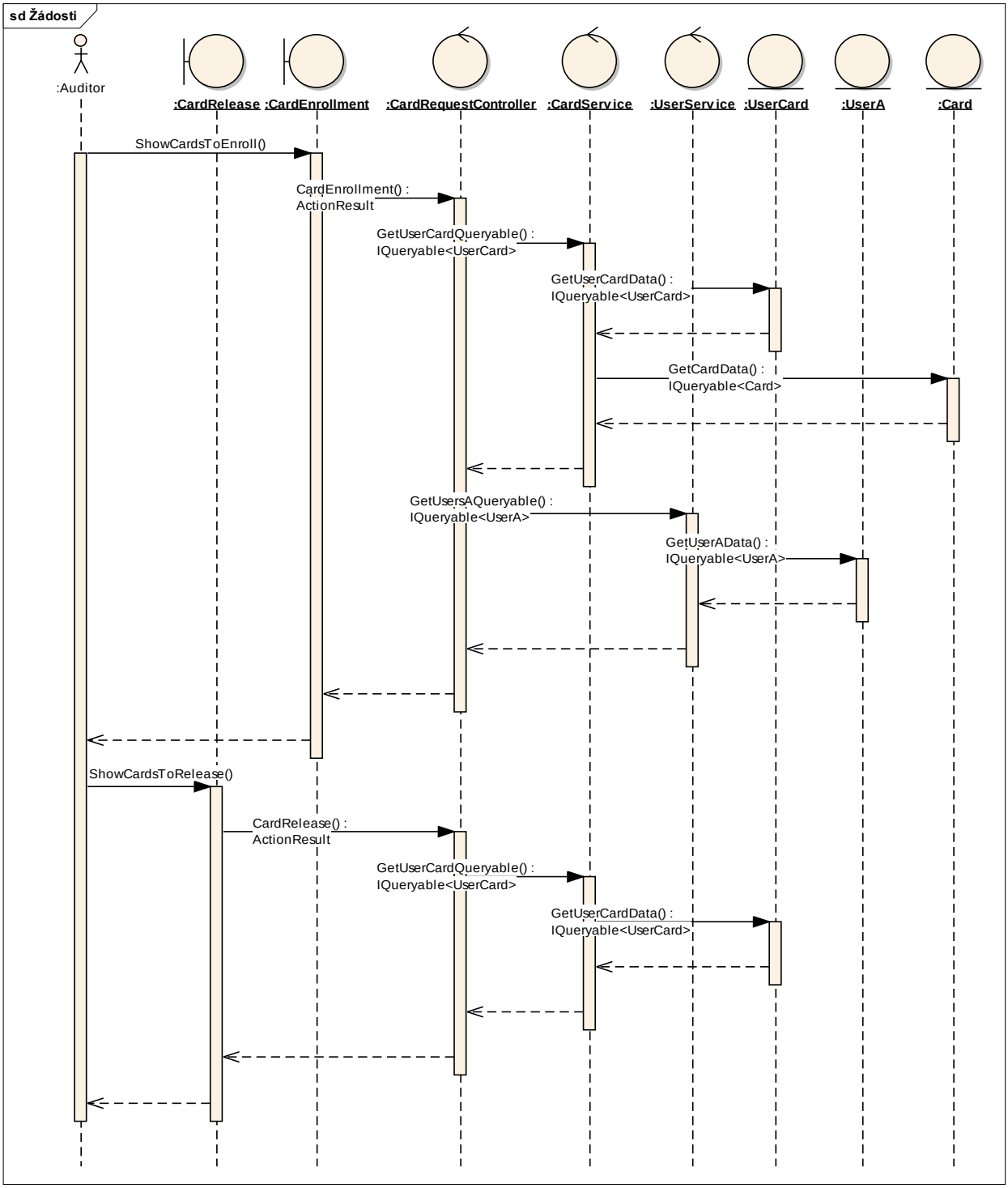
Zkoušky


sd Zkoušky
:Auditor
:Index
:PrintAssignedUsers
:PrintPart66
:PrintResults
:TestDayController
:TestDayService
:UserTestDay
:UserModule
:UserCard
:UserA
:TestDay
:Room
:Result
ShowTestDays()
Index() : ActionResult
GetTestDayQueryable() : IQueryable<TestDay>
GetTestDayData() : IQueryable<TestDay>
GetRoomData() : IQueryable<Room>
GetUsersInTestDay(int) : IEnumerable<UserTestDayModel>
GetUserTestDayData() : IEnumerable<UserTestDay>
GetUserModuleQueryable() : IQueryable<UserModule>
GetUserModuleData() : IQueryable<UserModule>
GetUserTestDayQueryable() : IQueryable<UserTestDay>
GetUserTestDayData() : IQueryable<UserTestDay>
GetUserCardData() : IQueryable<UserCard>
GetUserAData() : IQueryable<UserA>
GetResultData() : IQueryable<Result>
PrintTestDayResults()
PrintResults(int) : ActionResult
GetUserTestDayQueryable() : IQueryable<UserTestDay>
GetUserTestDayData() : IQueryable<UserTestDay>
GetUserAData() : IQueryable<UserA>
GetUserModuleData() :IQueryable<UserModule>
GetTestDayData() :TestDay
PrintPart66()
PrintPart66(int, int) : ActionResult
GetUserTestDayQueryable() : IQueryable<UserTestDay>
GetUserTestDayData() : IQueryable<UserTestDay>
GetUserAData() :IQueryable<UserA>
GetUserModuleData() : IQueryable<UserModule>
GetTestDayData() :TestDay
PrintAssignedStudents()
PrintAssignedUsers(int) : ActionResult
GetUserTestDayQueryable() : IQueryable<UserTestDay>
GetUserTestDayData() : IQueryable<UserTestDay>
GetTestDayData() :TestDay
GetUserModuleData() : IQueryable<UserModule>
GetUserAData() :IQueryable<UserA>

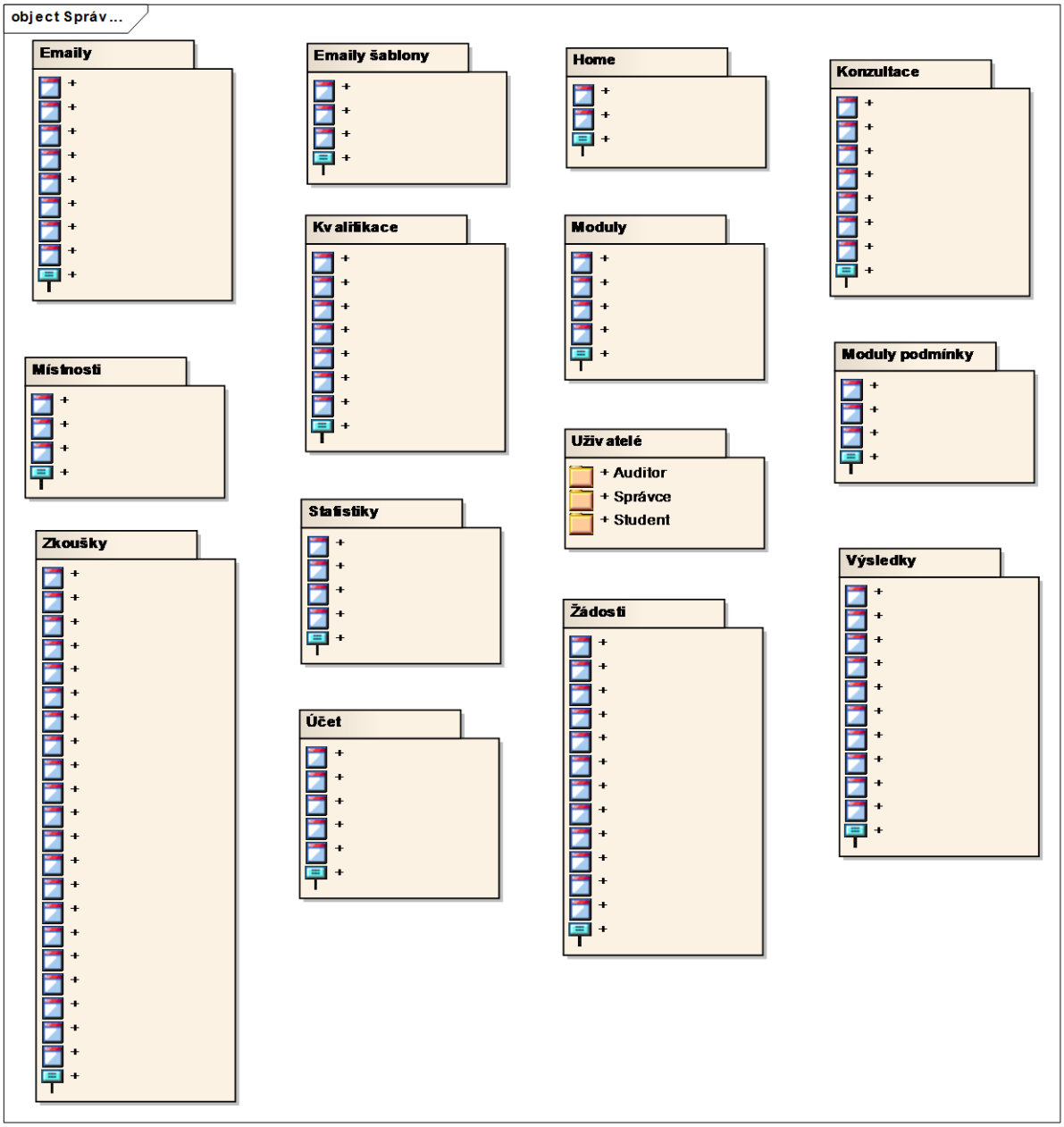
Účet


sd Účet
:Auditor
:Manager
:Password
:AccountController
:UserService
:UserB
ShowPersonalInformation()
Index() :
ActionResult
GetUserB(string) :
UserB
GetUserBData() :
UserB
UpdatePassword()
Password(PasswordModel) :
ActionResult
SetPassword(string, string) :
bool
UpdatePersonalInformation()
Update() :
ActionResult
GetUserB(string) :
UserB
GetUserBData() :
UserB
UpdateUserB(UserB)

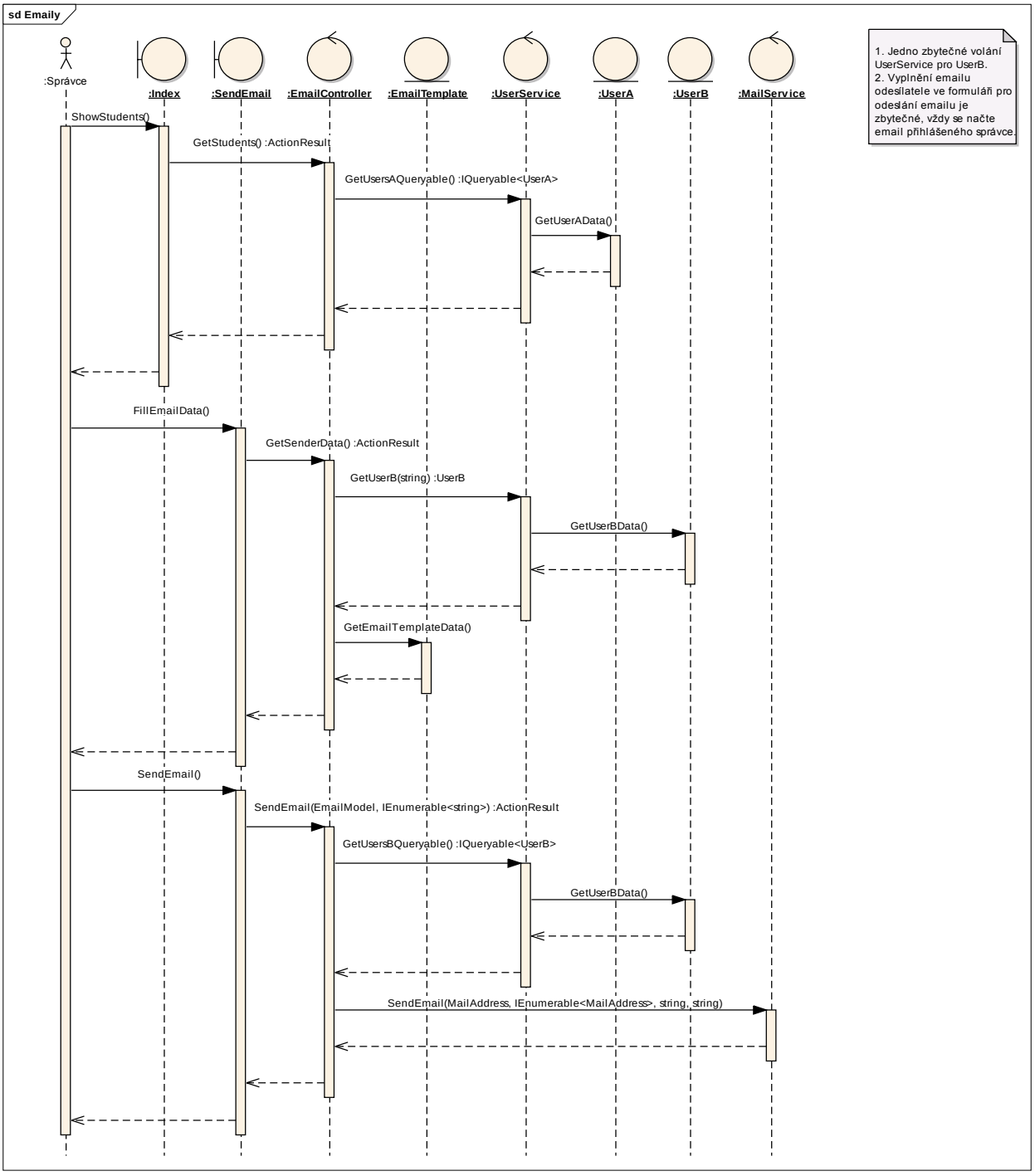
Žádosti


sd Žádosti
:Auditor
:CardRelease
:CardEnrollment
:CardRequestController
:CardService
:UserService
:UserCard
:UserA
:Card
ShowCardsToEnroll()
CardEnrollment() :
ActionResult
GetUserCardQueryable() :
IQueryable<UserCard>
GetUserCardData() :
IQueryable<UserCard>
GetCardData() :
IQueryable<Card>
GetUsersAQueryable() :
IQueryable<UserA>
GetUserAData() :
IQueryable<UserA>
ShowCardsToRelease()
CardRelease() :
ActionResult
GetUserCardQueryable() :
IQueryable<UserCard>
GetUserCardData() :
IQueryable<UserCard>

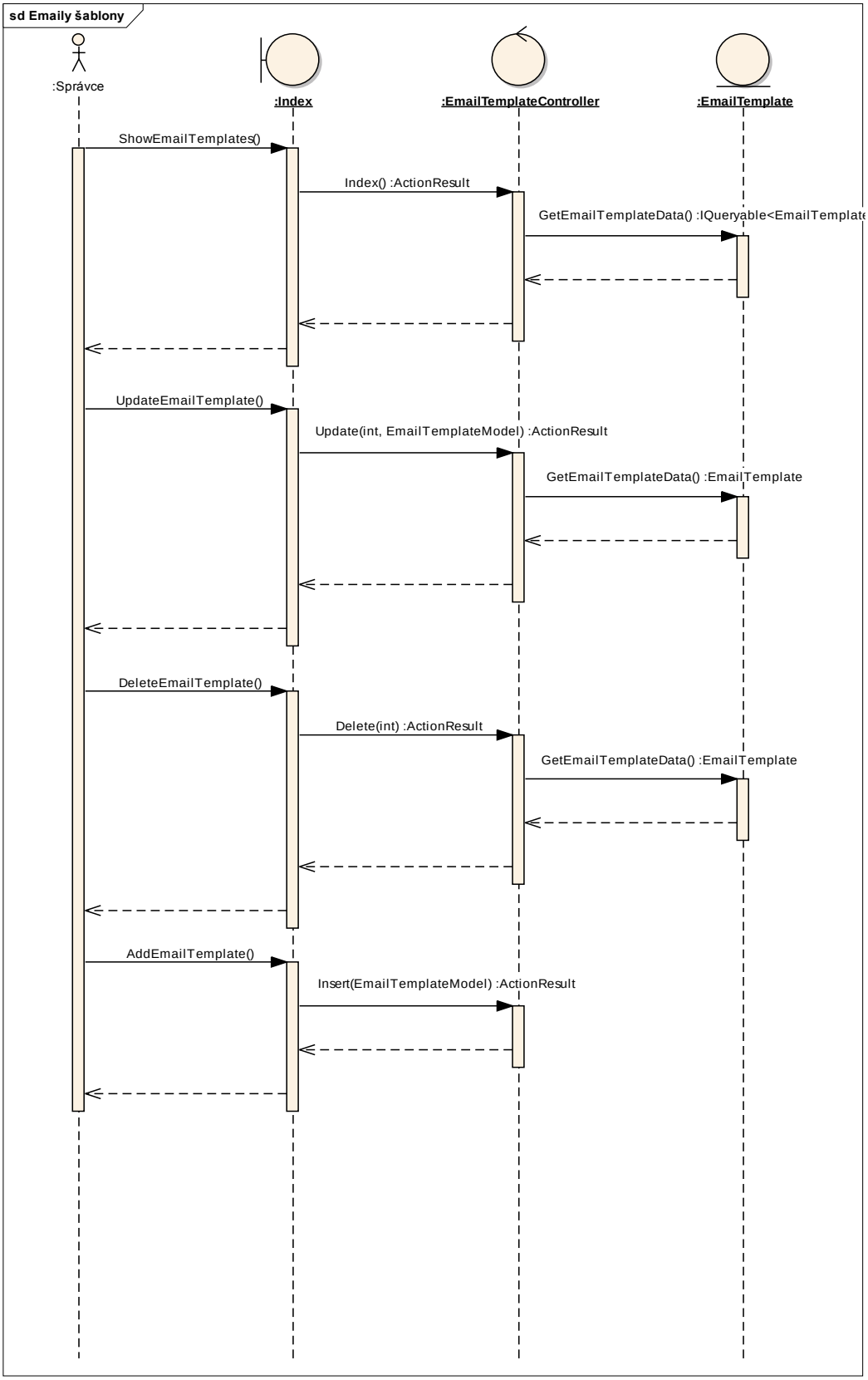
Správce


object Správ...
Emaily
Emaily šablony
Home
Konzultace
Kvalifikace
Moduly
Místnosti
Moduly podmínky
Uživatelé
+ Auditor
+ Správce
+ Student
Statistiky
Zkoušky
Výsledky
Účet
Žádosti

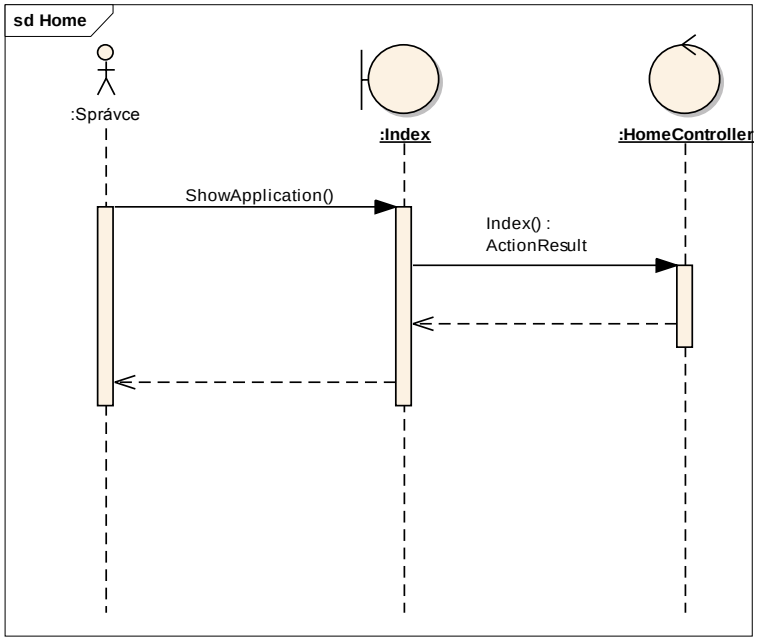
Emaily


sd Emaily
:Správce
:Index
:SendEmail
:EmailController
:EmailTemplate
:UserService
:UserA
:UserB
:MailService
1. Jedno zbytečné volání UserService pro UserB.
2. Vyplnění emailu odesílatele ve formuláři pro odeslání emailu je zbytečné, vždy se načte email přihlášeného správce.
ShowStudents()
GetStudents() :ActionResult
GetUsersAQueryable() :IQueryable<UserA>
GetUserAData()
FillEmailData()
GetSenderData() :ActionResult
GetUserB(string) :UserB
GetUserBData()
GetEmailTemplateData()
SendEmail()
SendEmail(EmailModel, IEnumerable<string>) :ActionResult
GetUsersBQueryable() :IQueryable<UserB>
GetUserBData()
SendEmail(MailAddress, IEnumerable<MailAddress>, string, string)

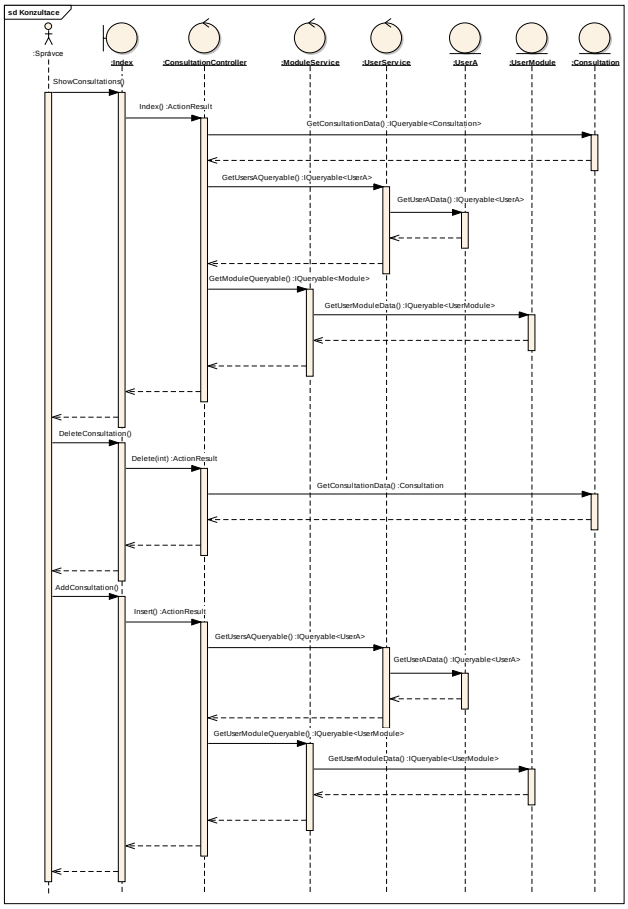
Emaily šablony



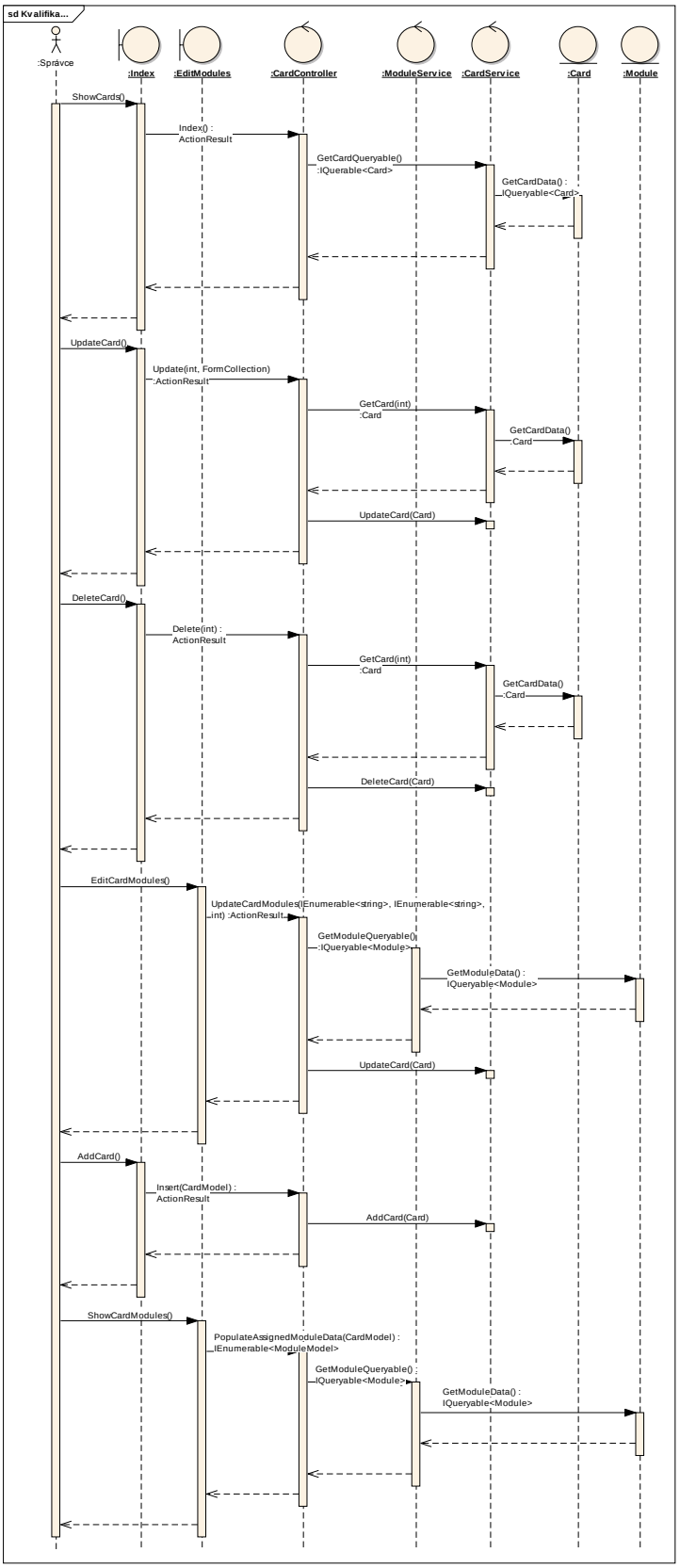
Home



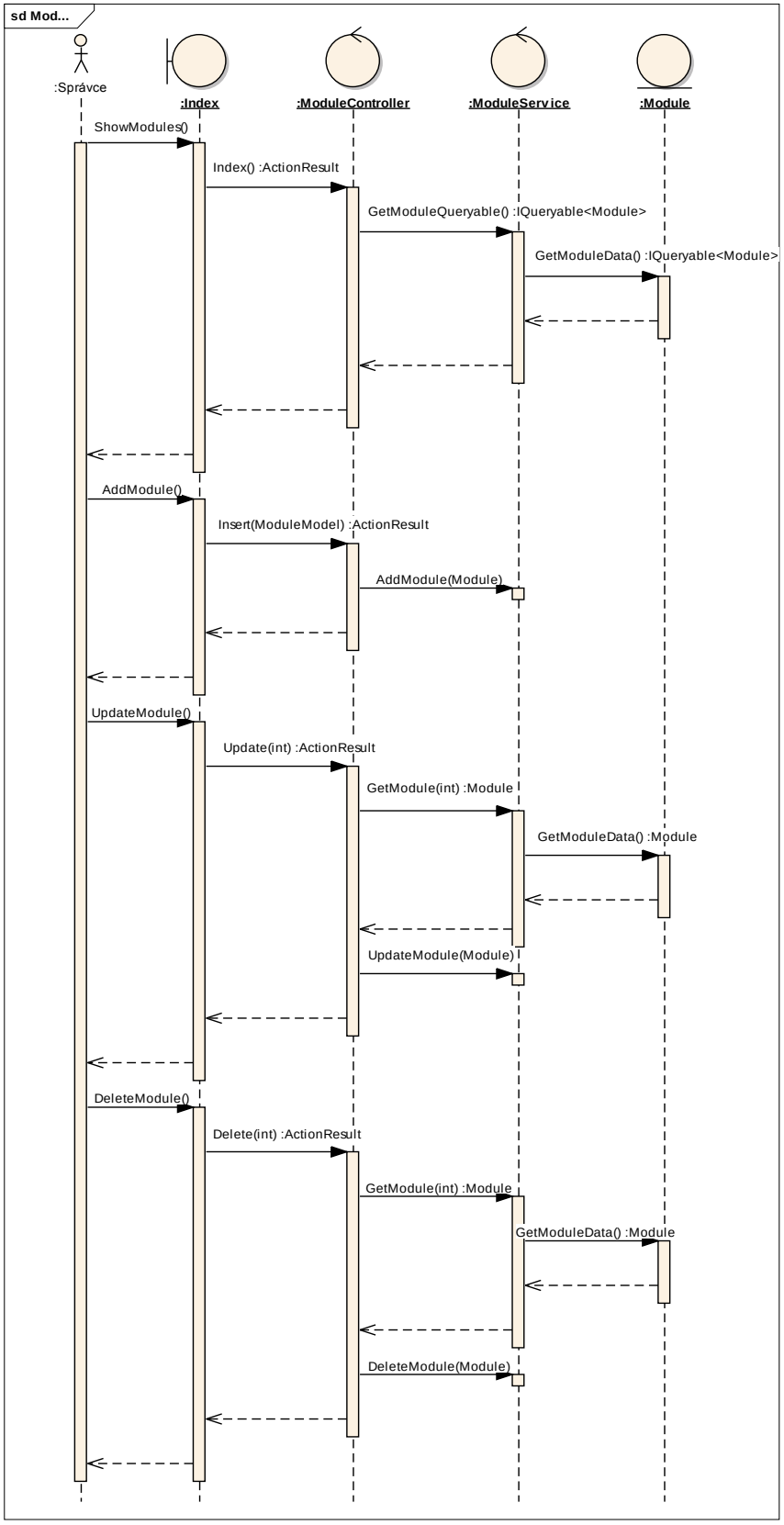
Konzultace



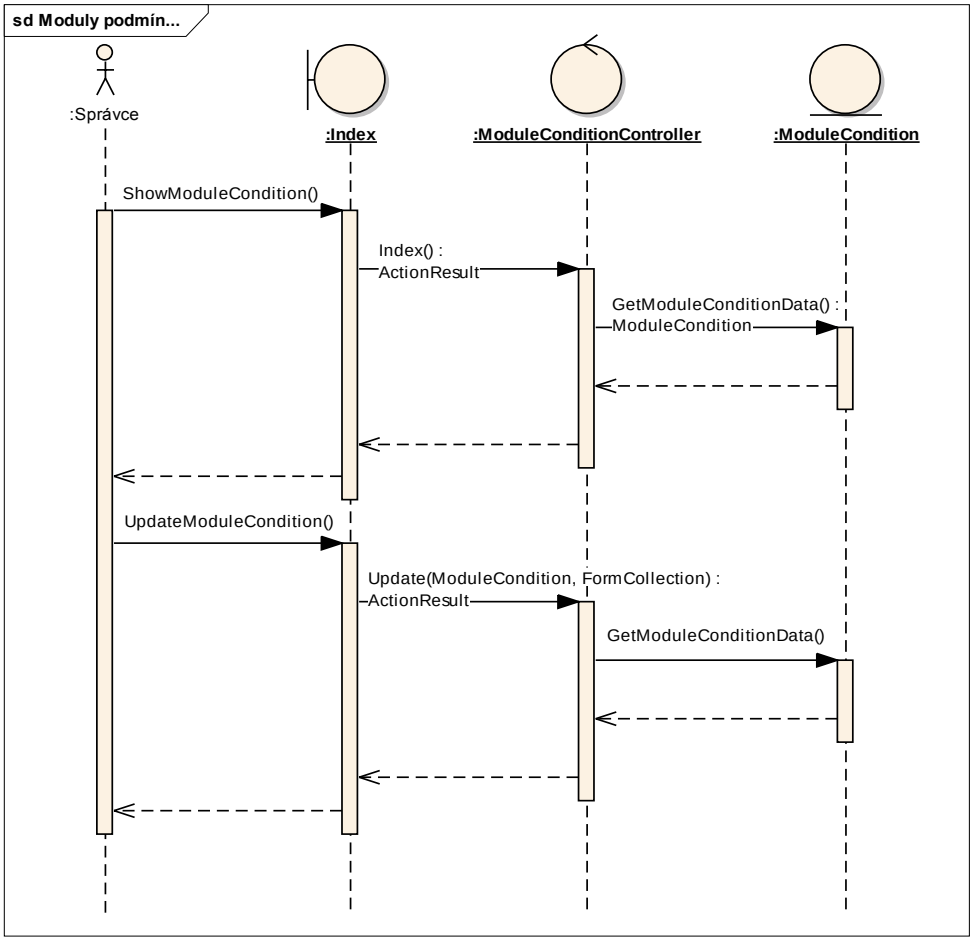
Kvalifikace



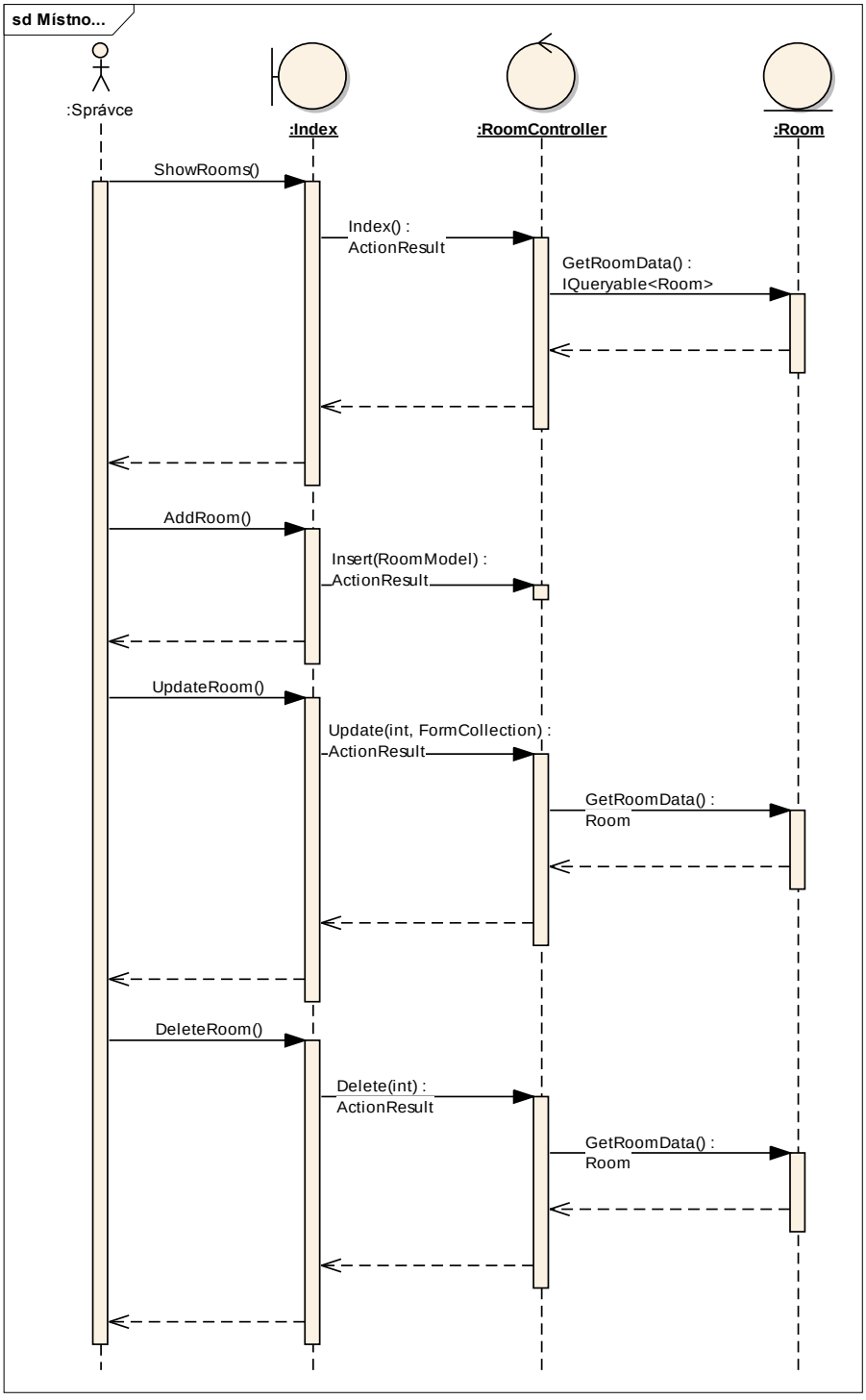
Moduly


sd Mod...
:Správce
:Index
:ModuleController
:ModuleService
:Module
ShowModules()
Index() :ActionResult
GetModuleQueryable() :IQueryable<Module>
GetModuleData() :IQueryable<Module>
AddModule()
Insert(ModuleModel) :ActionResult
AddModule(Module)
UpdateModule()
Update(int) :ActionResult
GetModule(int) :Module
GetModuleData() :Module
UpdateModule(Module)
DeleteModule()
Delete(int) :ActionResult
GetModule(int) :Module
GetModuleData() :Module
DeleteModule(Module)

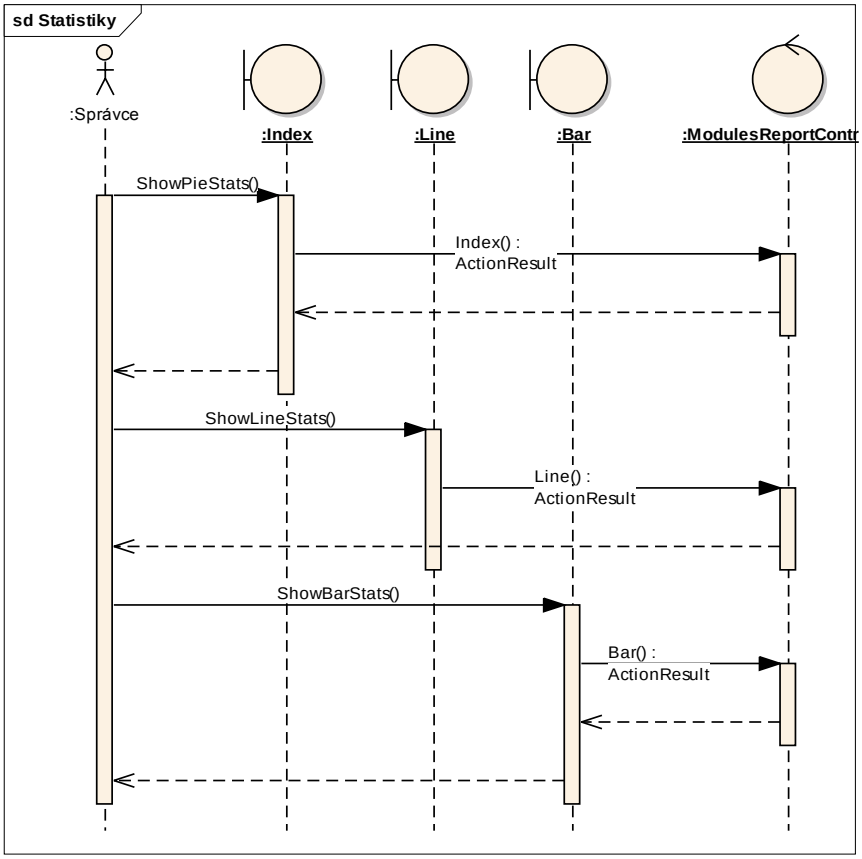
Moduly podmínky



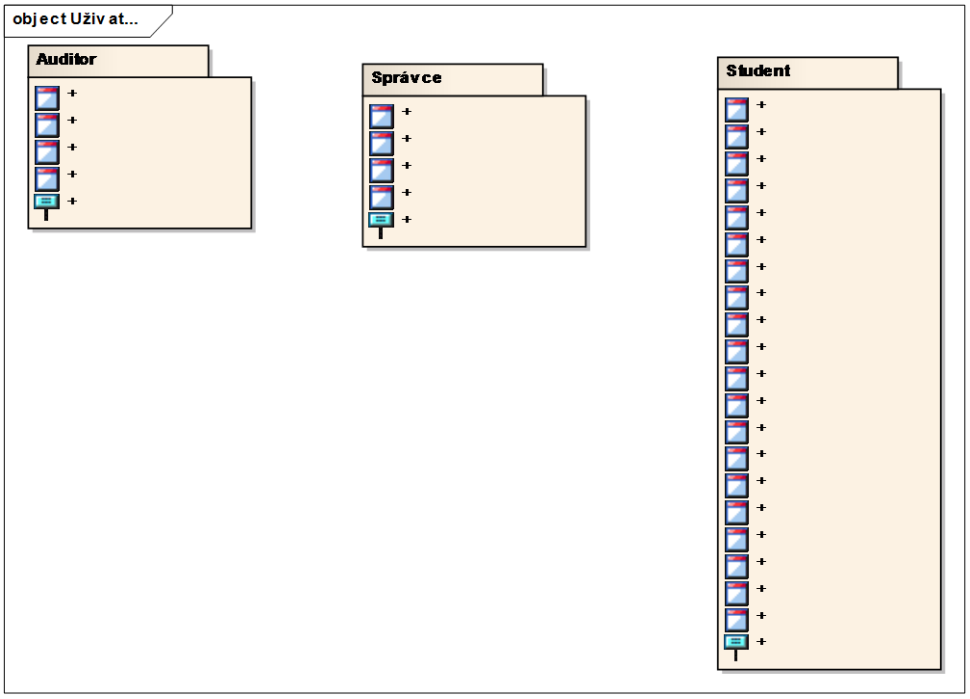
Místnosti


sd Místno...
:Správce
:Index
:RoomController
:Room
ShowRooms()
Index() :
ActionResult
GetRoomData() :
IQueryable<Room>
AddRoom()
Insert(RoomModel) :
ActionResult
UpdateRoom()
Update(int, FormCollection) :
ActionResult
GetRoomData() :
Room
DeleteRoom()
Delete(int) :
ActionResult
GetRoomData() :
Room

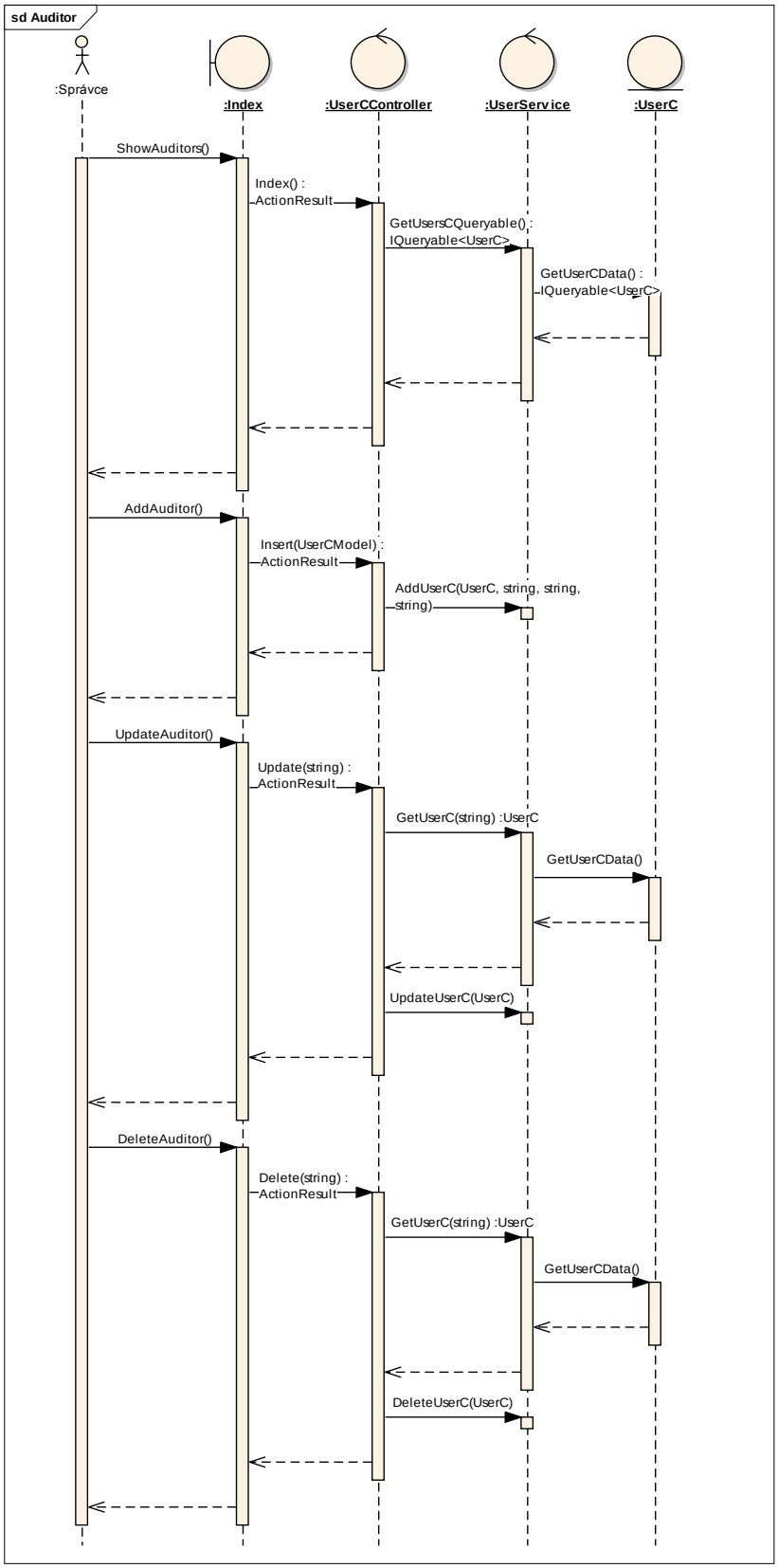
Statistiky



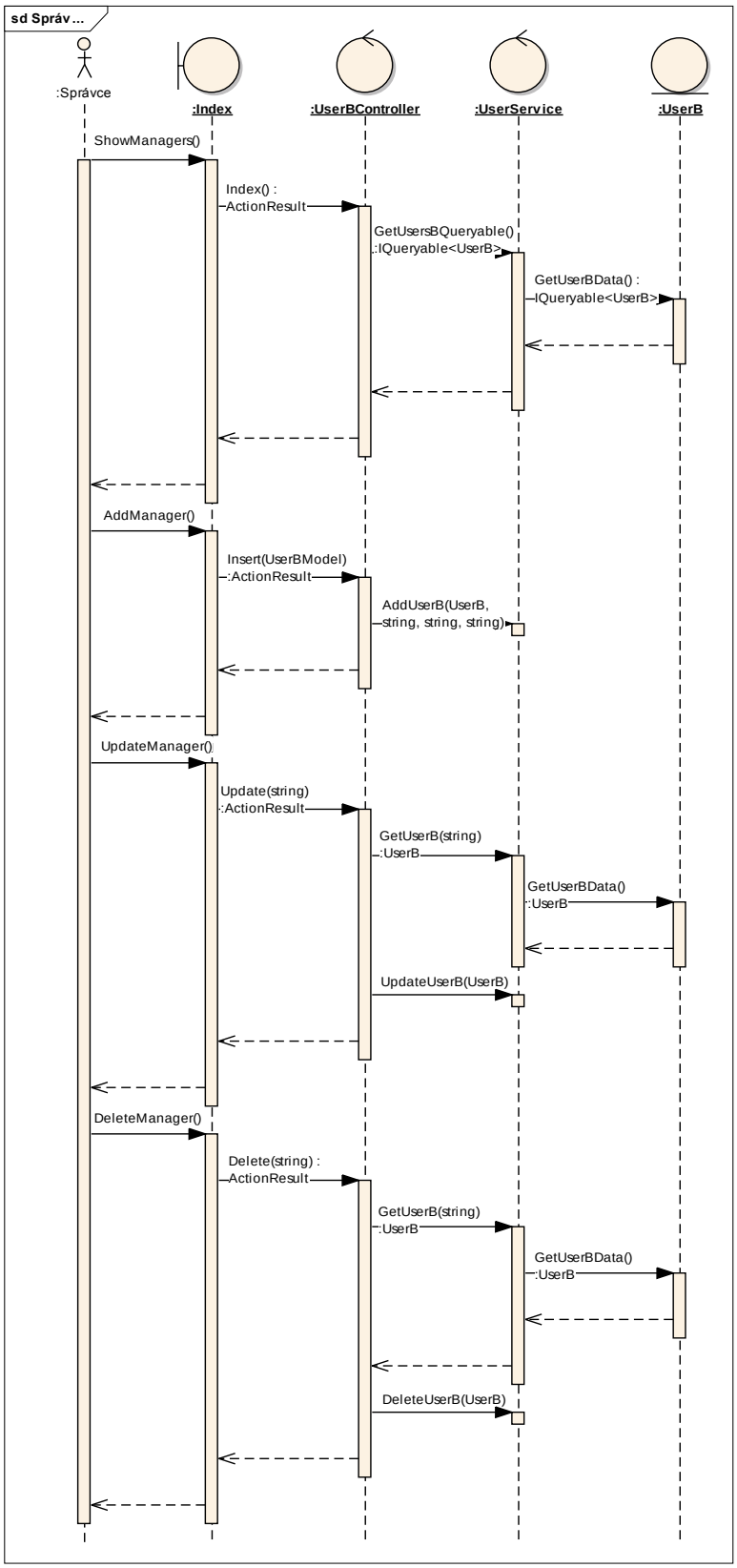
Uživatelé



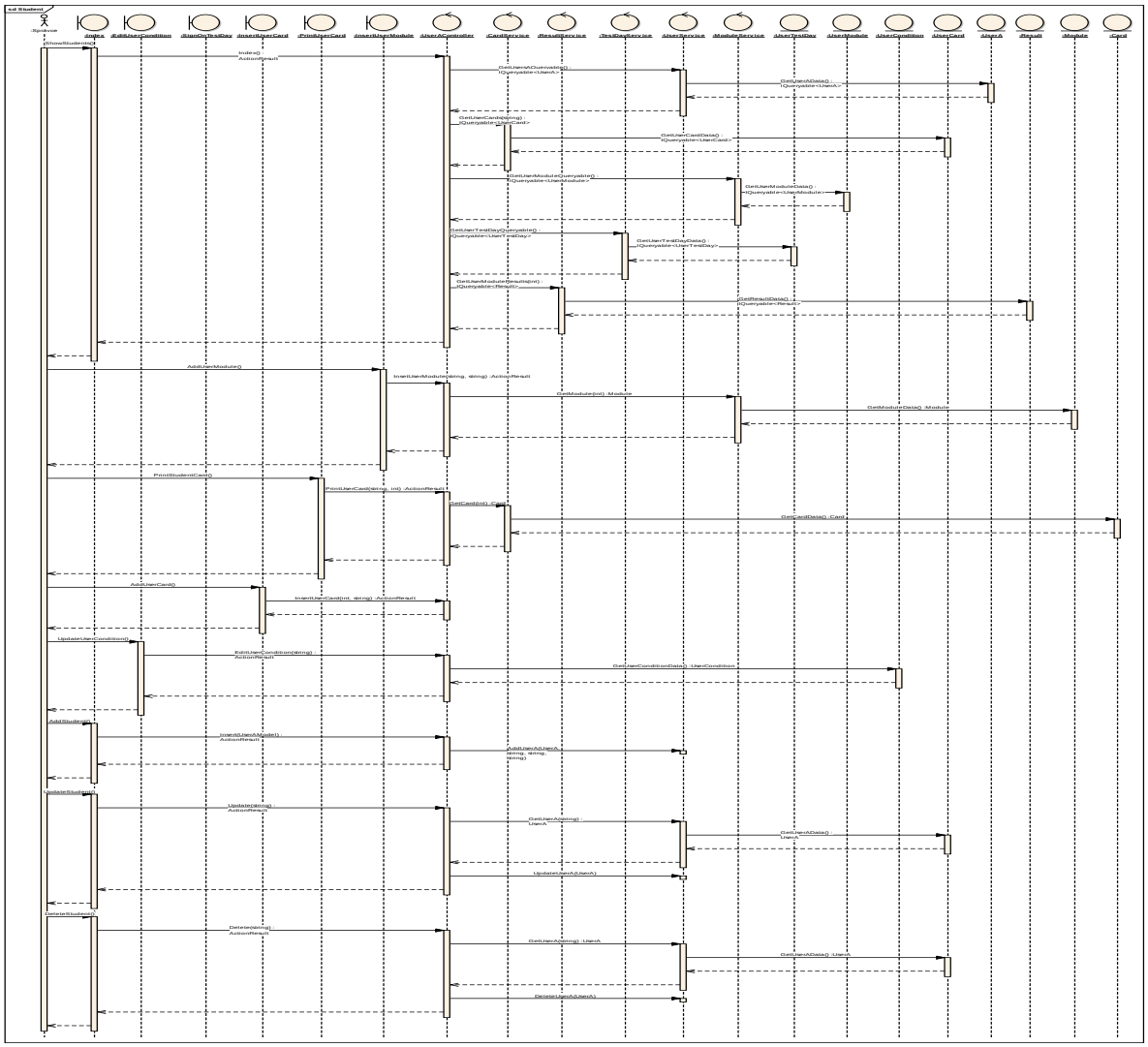
Auditor


sd Auditor
:Správce
:Index
:UserCController
:UserService
:UserC
ShowAuditors()
Index() : ActionResult
GetUsersCQueryable() : IQueryable<UserC>
GetUserCData() : IQueryable<UserC>
AddAuditor()
Insert(UserCModel) : ActionResult
AddUserC(UserC, string, string, string)
UpdateAuditor()
Update(string) : ActionResult
GetUserC(string) :UserC
GetUserCData()
UpdateUserC(UserC)
DeleteAuditor()
Delete(string) : ActionResult
GetUserC(string) :UserC
GetUserCData()
DeleteUserC(UserC)

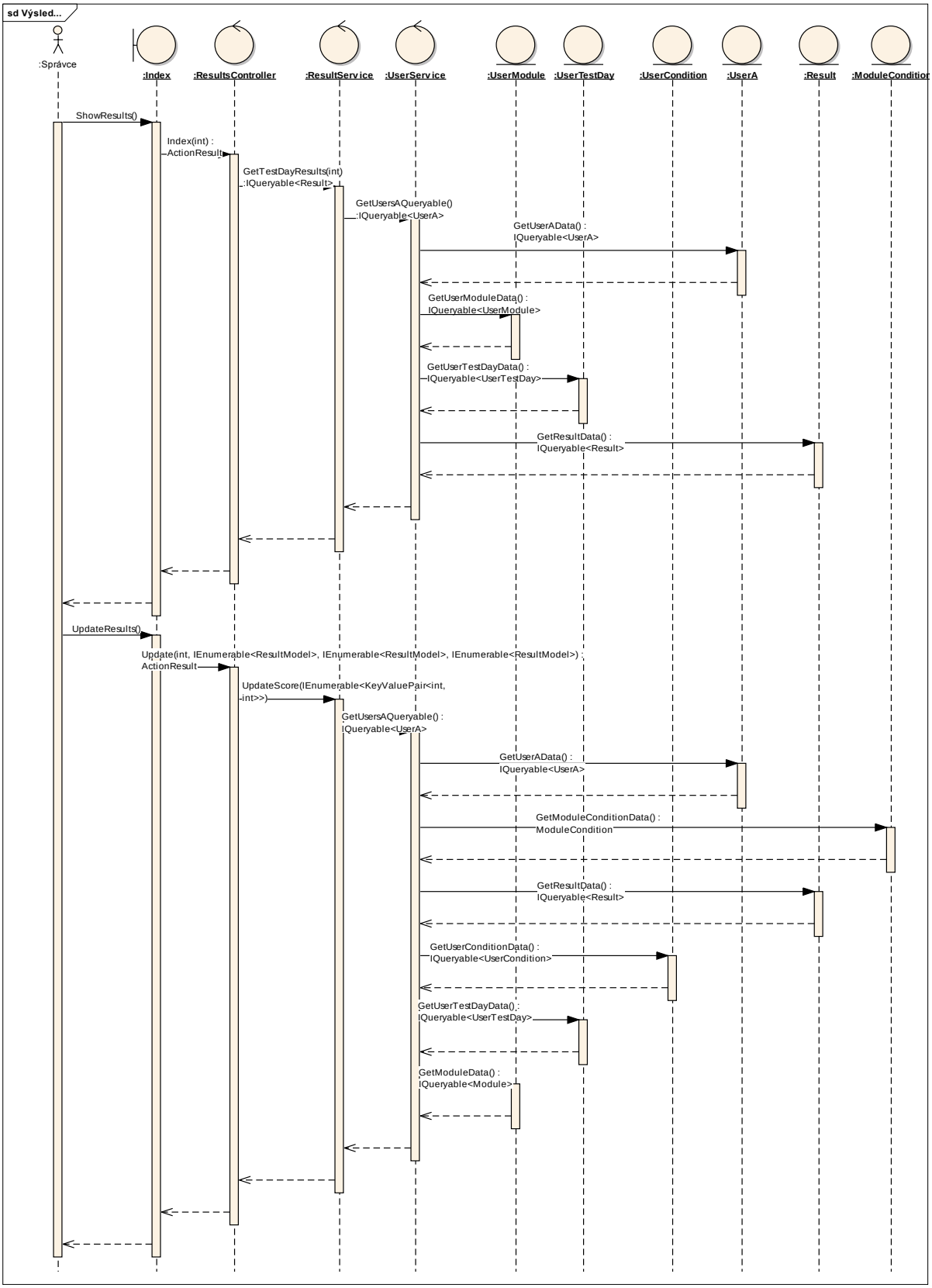
Správce


sd Správ...
:Správce
:Index
:UserBController
:UserService
:UserB
ShowManagers()
Index() :
ActionResult
GetUsersBQueryable()
:IQueryable<UserB>
GetUserBData() :
IQueryable<UserB>
AddManager()
Insert(UserBModel)
:ActionResult
AddUserB(UserB,
string, string, string)
UpdateManager()
Update(string)
:ActionResult
GetUserB(string)
:UserB
GetUserBData()
:UserB
UpdateUserB(UserB)
DeleteManager()
Delete(string) :
ActionResult
GetUserB(string)
:UserB
GetUserBData()
:UserB
DeleteUserB(UserB)

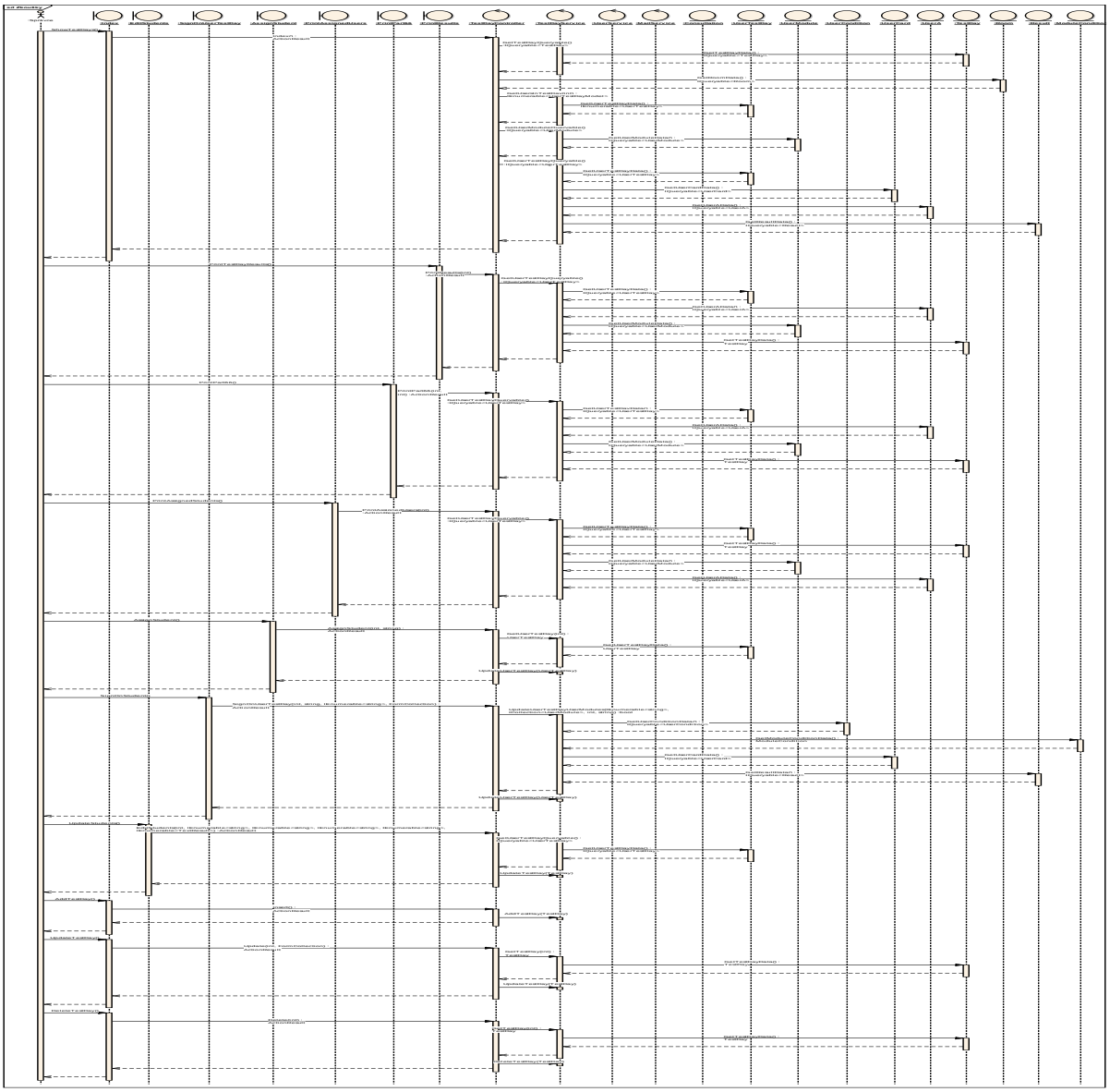
Student


sd Student
:Spvisce
ShowStudents()
Index()
ActionResult
GetUserA(Queryable)
IQueryable<UserA>
GetUserA(Data)
IQueryable<UserA>
GetUserCard(String)
IQueryable<UserCard>
GetUserCard(Data)
IQueryable<UserCard>
GetUserModule(Queryable)
IQueryable<UserModule>
GetUserModule(Data)
IQueryable<UserModule>
GetUserTestDay(Queryable)
IQueryable<UserTestDay>
GetUserTestDay(Data)
IQueryable<UserTestDay>
GetUserModule(ResultInt)
IQueryable<Result>
GetResult(Data)
IQueryable<Result>
AddUserModule()
InserUserModule(string, string) :ActionResult
GetModule(int) :Module
GetModule() :Module
PrintStudentCard()
PrintUserCard(string, int) :ActionResult
GetCard(int) :Card
GetCard(Data) :Card
AddUserCard()
InserUserCard(int, string) :ActionResult
UpdateUserCondition()
EditUserCondition(String)
ActionResult
GetUserCondition(Data) :UserCondition
AddStudent()
InserUserA(Model)
ActionResult
AddUserA(string, string, string, string)
UpdateStudent()
Update(String)
ActionResult
GetUserA(string) :UserA
GetUserA(Data) :UserA
UpdateUserA(UserA)
DeleteStudent()
Delete(String)
ActionResult
GetUserA(string) :UserA
GetUserA(Data) :UserA
DeleteUserA(UserA)

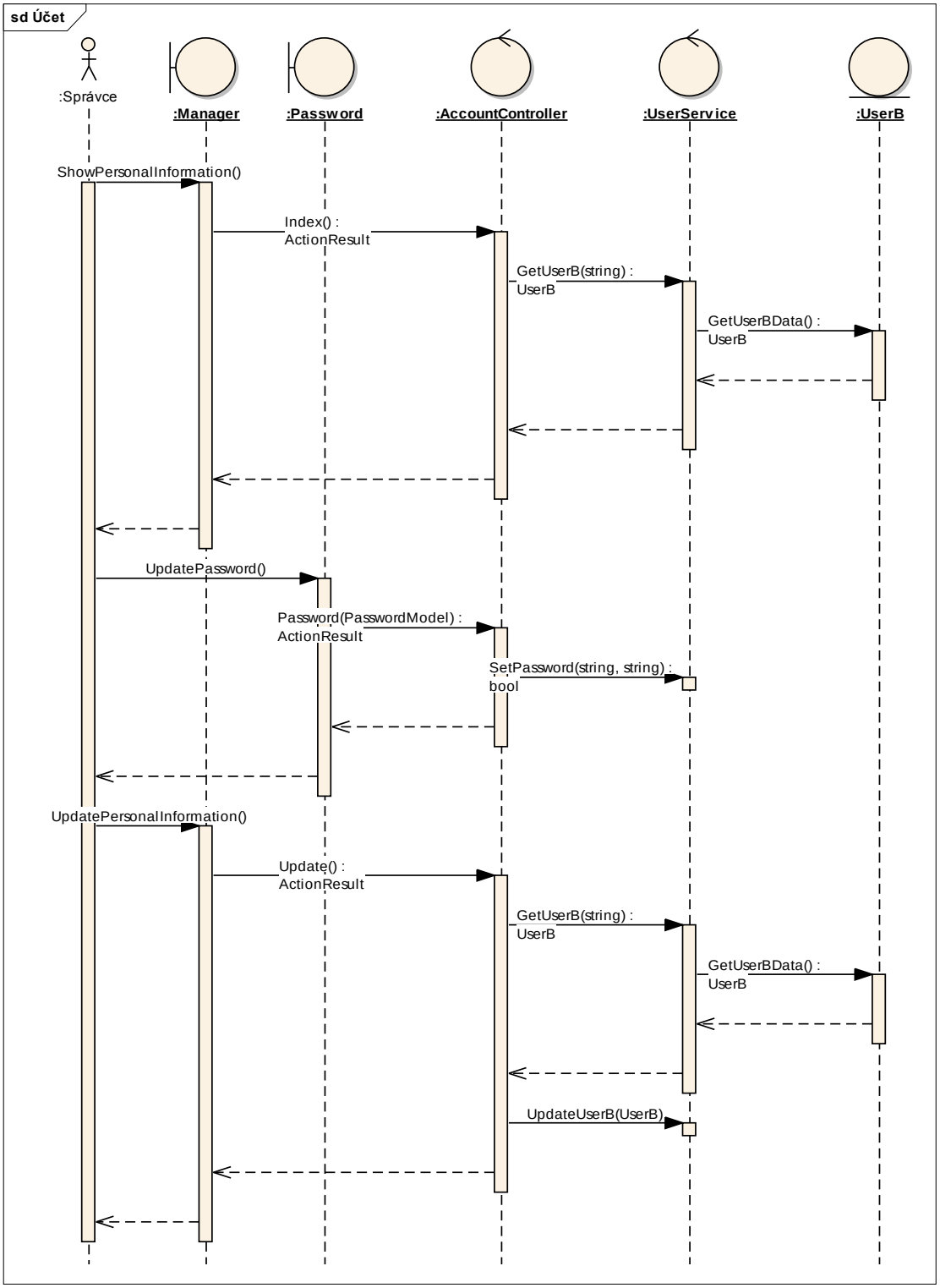
Výsledky


sd Výsled...
:Správce
:Index
:ResultsController
:ResultService
:UserService
:UserModule
:UserTestDay
:UserCondition
:UserA
:Result
:ModuleCondition
ShowResults()
Index(int) : ActionResult
GetTestDayResults(int) :IQueryable<Result>
GetUsersAQueryable() :IQueryable<UserA>
GetUserAData() : IQueryable<UserA>
GetUserModuleData() : IQueryable<UserModule>
GetUserTestDayData() : IQueryable<UserTestDay>
GetResultData() : IQueryable<Result>
UpdateResults()
Update(int, IEnumerable<ResultModel>, IEnumerable<ResultModel>, IEnumerable<ResultModel>) : ActionResult
UpdateScore(IEnumerable<KeyValuePair<int, int>>)
GetUsersAQueryable() : IQueryable<UserA>
GetUserAData() : IQueryable<UserA>
GetModuleConditionData() : ModuleCondition
GetResultData() : IQueryable<Result>
GetUserConditionData() : IQueryable<UserCondition>
GetUserTestDayData() : IQueryable<UserTestDay>
GetModuleData() : IQueryable<Module>

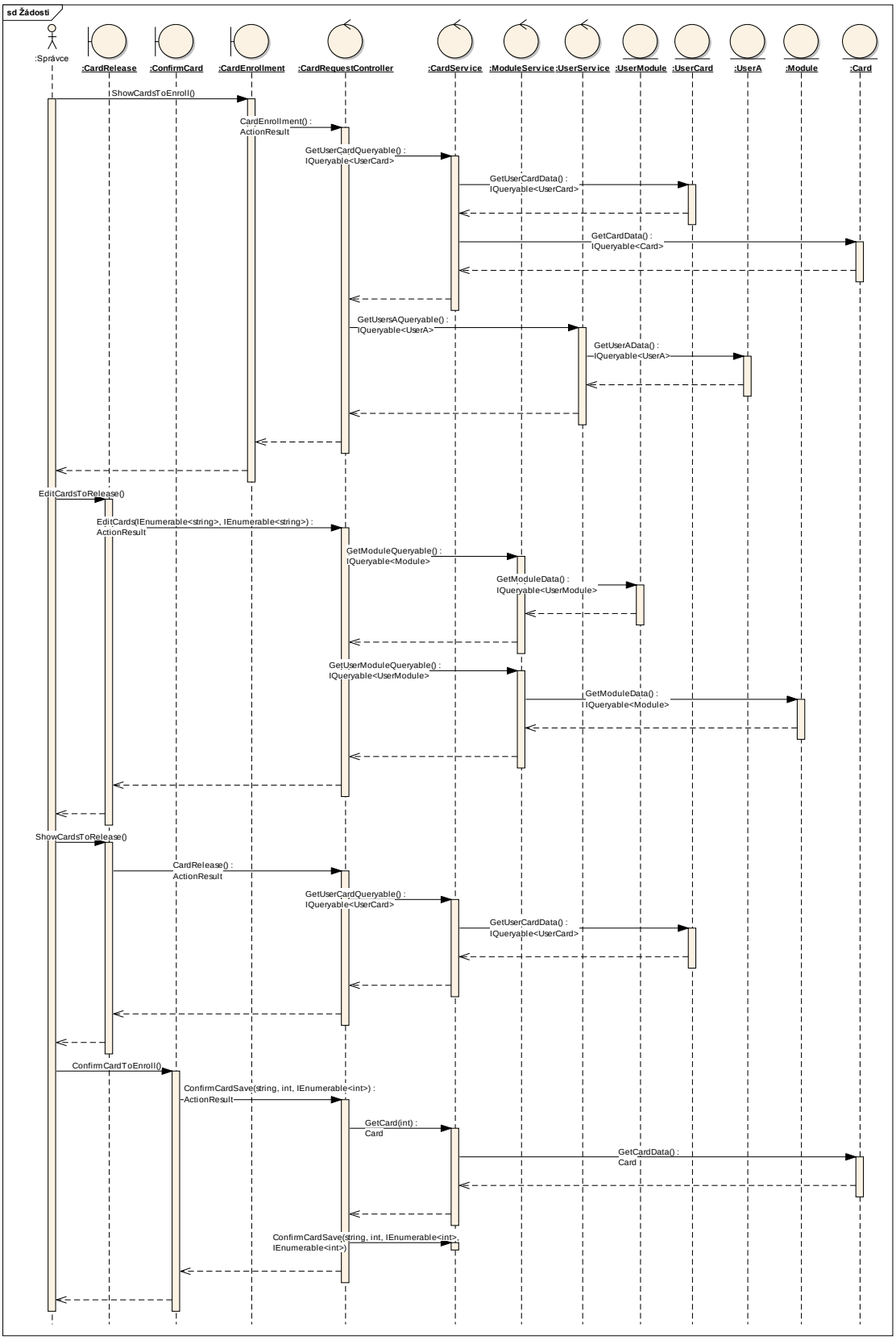
Zkoušky



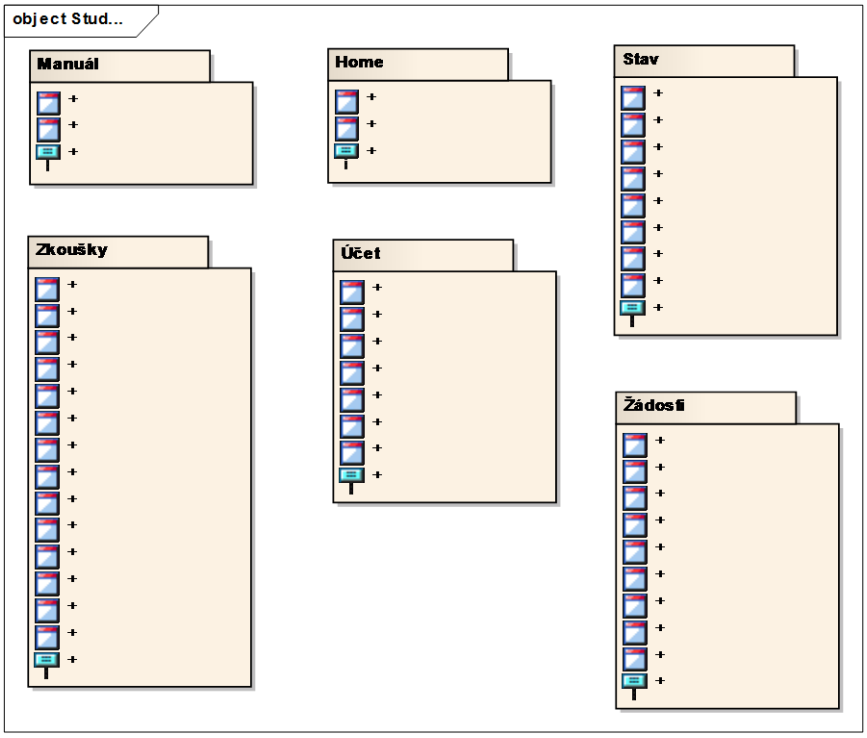
Účet


sd Účet
:Správce
:Manager
:Password
:AccountController
:UserService
:UserB
ShowPersonalInformation()
Index() :
ActionResult
GetUserB(string) :
UserB
GetUserBData() :
UserB
UpdatePassword()
Password(PasswordModel) :
ActionResult
SetPassword(string, string) :
bool
UpdatePersonalInformation()
Update() :
ActionResult
GetUserB(string) :
UserB
GetUserBData() :
UserB
UpdateUserB(UserB)

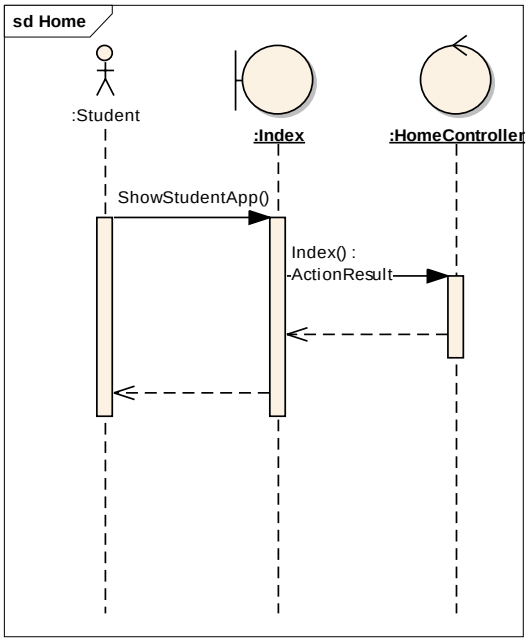
Žádosti


sd Žádosti
:Správce
:CardRelease
:ConfirmCard
:CardEnrollment
:CardRequestController
:CardService
:ModuleService
:UserService
:UserModule
:UserCard
:UserA
:Module
:Card
ShowCardsToEnroll()
CardEnrollment() :
ActionResult
GetUserCardQueryable() :
IQueryable<UserCard>
GetUserCardData() :
IQueryable<UserCard>
GetCardData() :
IQueryable<Card>
GetUsersAQueryable() :
IQueryable<UserA>
GetUserAData() :
IQueryable<UserA>
EditCardsToRelease()
EditCards(IEnumerable<string>, IEnumerable<string>) :
ActionResult
GetModuleQueryable() :
IQueryable<Module>
GetModuleData() :
IQueryable<UserModule>
GetUserModuleQueryable() :
IQueryable<UserModule>
GetModuleData() :
IQueryable<Module>
ShowCardsToRelease()
CardRelease() :
ActionResult
GetUserCardQueryable() :
IQueryable<UserCard>
GetUserCardData() :
IQueryable<UserCard>
ConfirmCardToEnroll()
ConfirmCardSave(string, int, IEnumerable<int>) :
ActionResult
GetCard(int) :
Card
GetCardData() :
Card
ConfirmCardSave(string, int, IEnumerable<int>
IEnumerable<int>)

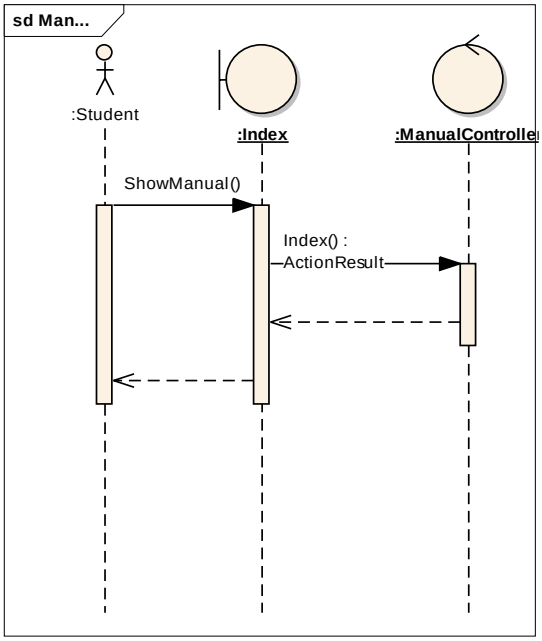
Student



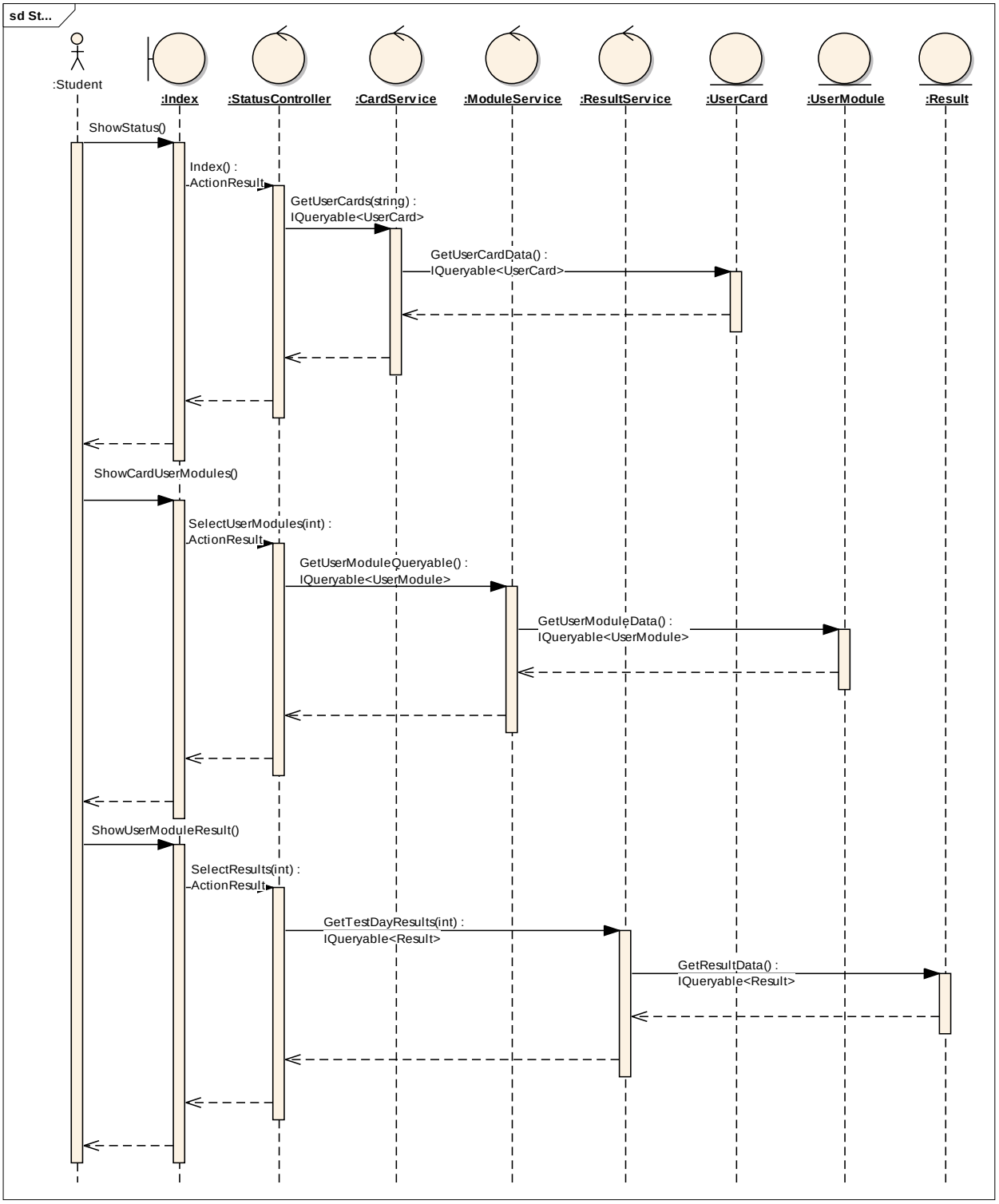
Home



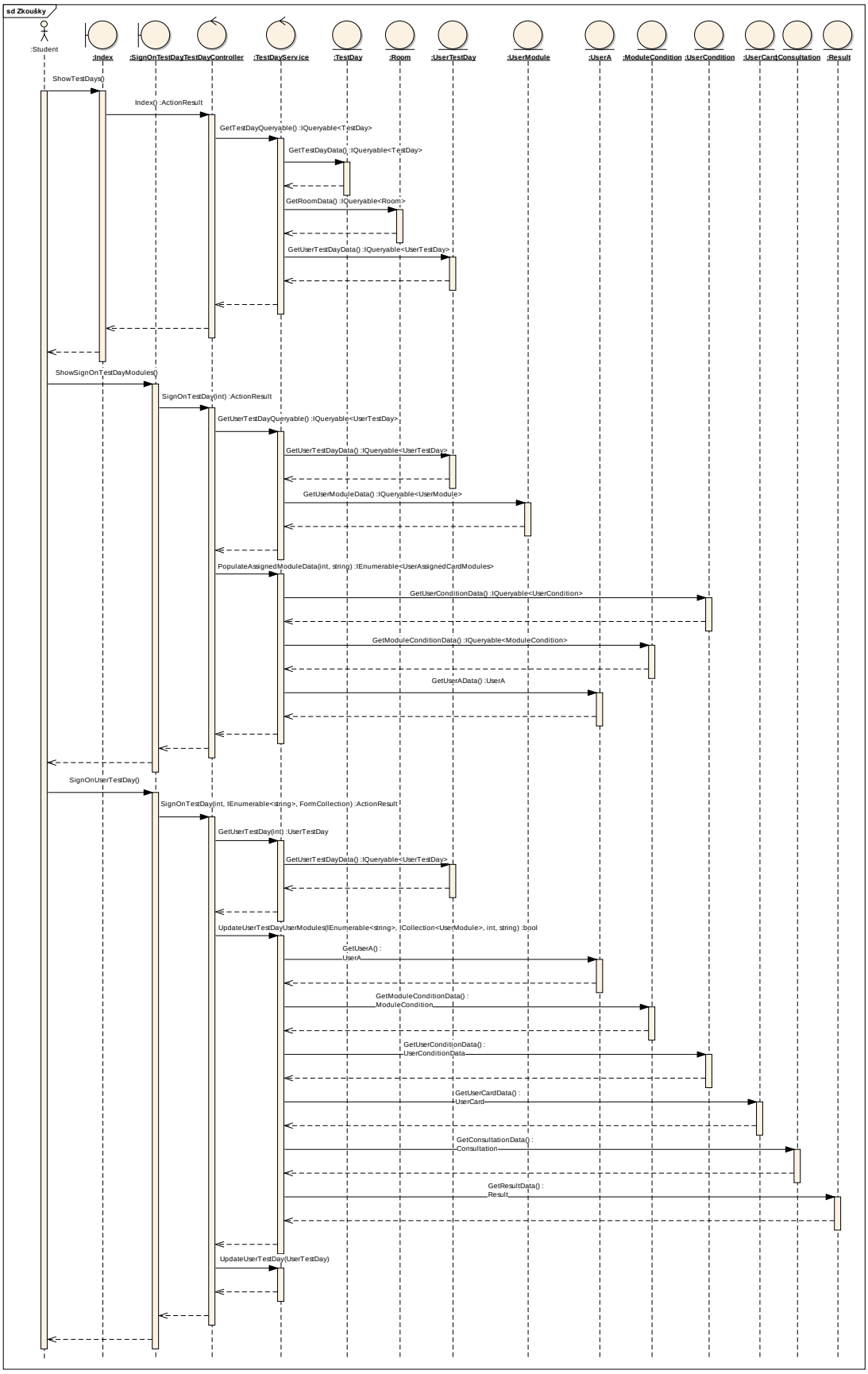
Manuál


sd Man...
:Student
:Index
:ManualControlle
ShowManual()
Index() :
ActionResult

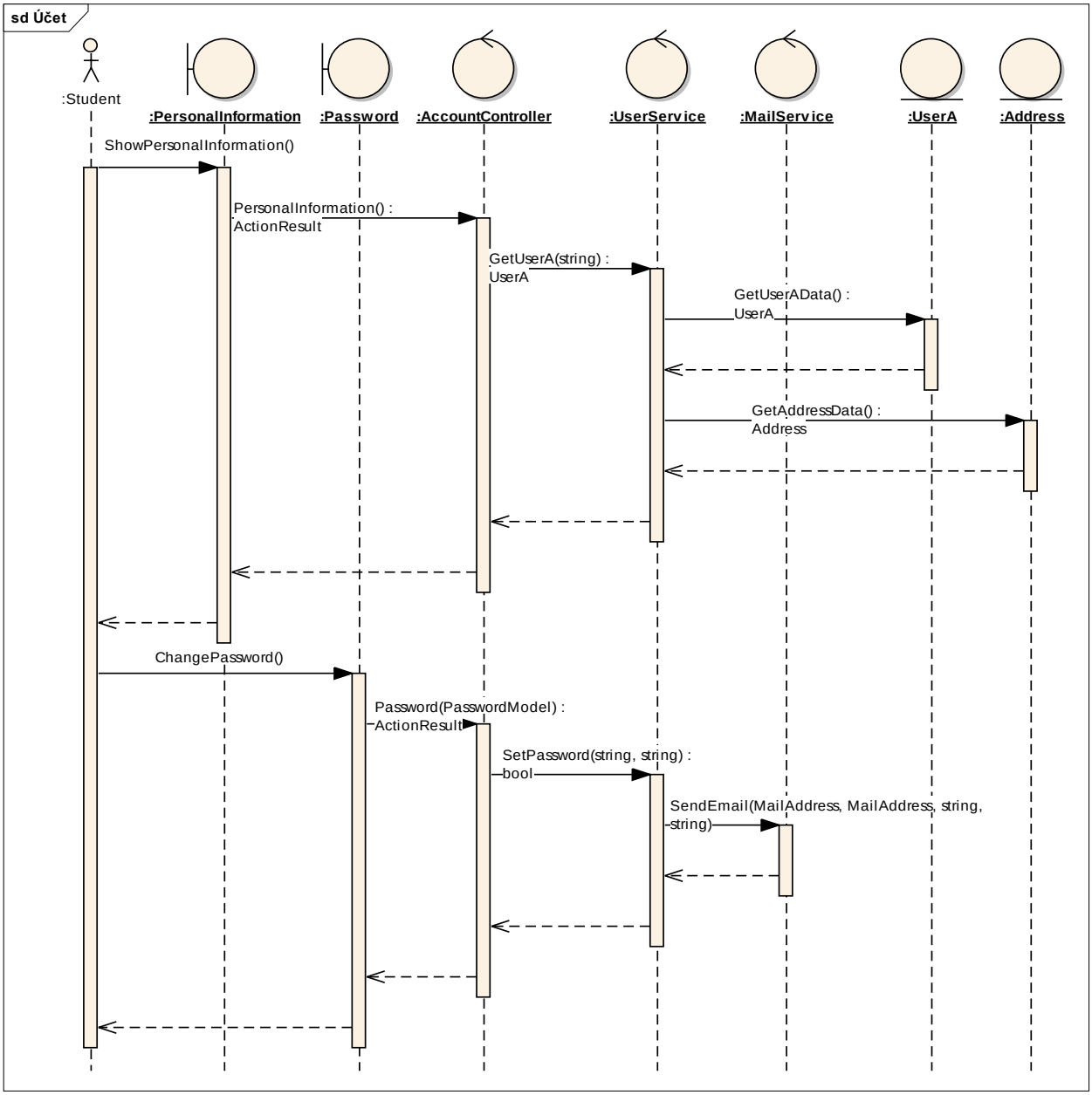
Stav


sd St...
:Student
:Index
:StatusController
:CardService
:ModuleService
:ResultService
:UserCard
:UserModule
:Result
ShowStatus()
Index() :
ActionResult
GetUserCards(string) :
IQueryable<UserCard>
GetUserCardData() :
IQueryable<UserCard>
ShowCardUserModules()
SelectUserModules(int) :
ActionResult
GetUserModuleQueryable() :
IQueryable<UserModule>
GetUserModuleData() :
IQueryable<UserModule>
ShowUserModuleResult()
SelectResults(int) :
ActionResult
GetTestDayResults(int) :
IQueryable<Result>
GetResultData() :
IQueryable<Result>

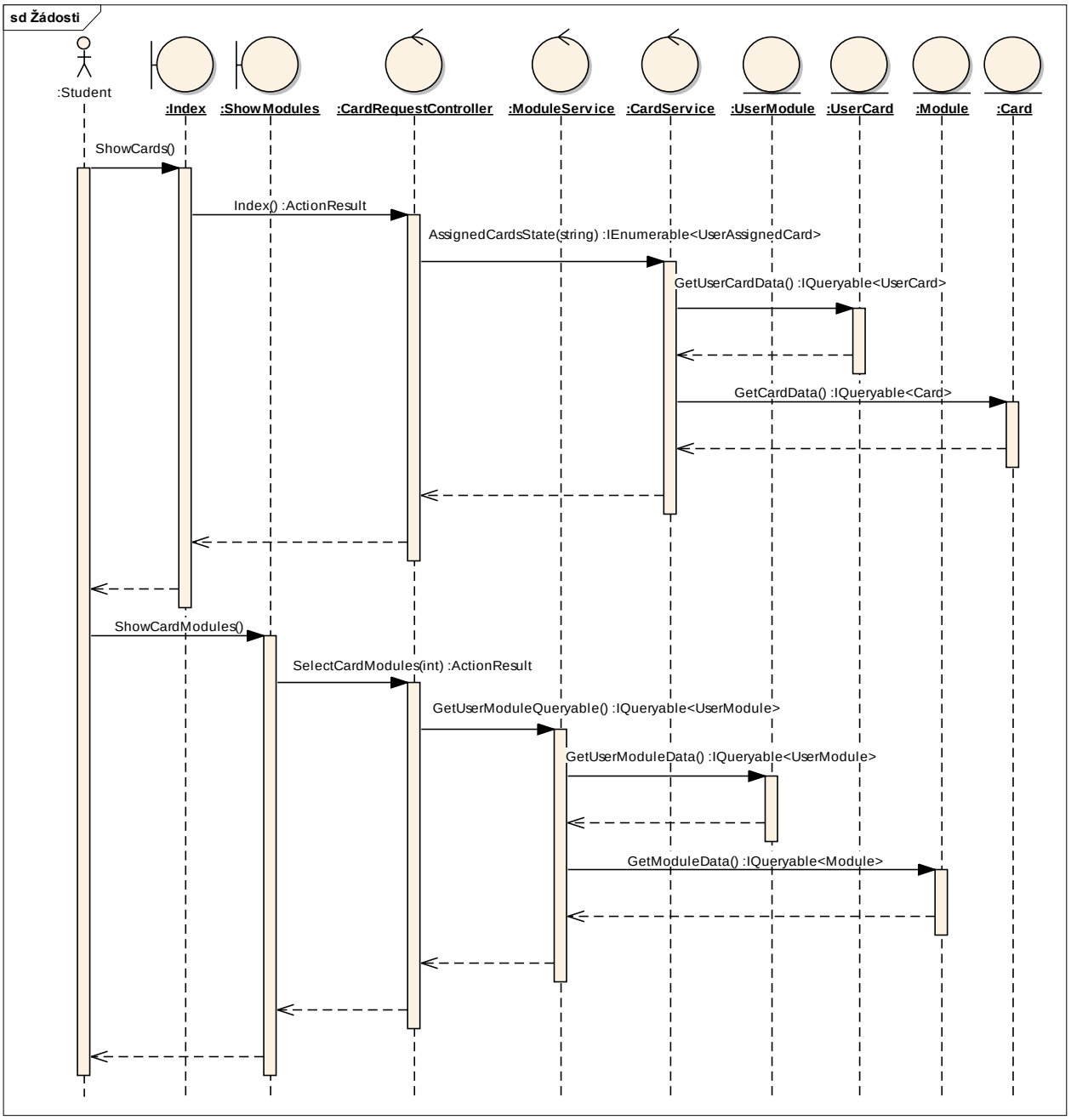
Zkoušky


sd Zkoušky
:Student
:Index
:SignOnTestDayTestDayController
:TestDayService
:TestDay
:Room
:UserTestDay
:UserModule
:UserA
:ModuleCondition
:UserCondition
:UserCardConsultation
:Result
ShowTestDays()
Index() :ActionResult
GetTestDayQueryable() :IQueryable<TestDay>
GetTestDayData() :IQueryable<TestDay>
GetRoomData() :IQueryable<Room>
GetUserTestDayData() :IQueryable<UserTestDay>
ShowSignOnTestDayModules()
SignOnTestDay(int) :ActionResult
GetUserTestDayQueryable() :IQueryable<UserTestDay>
GetUserTestDayData() :IQueryable<UserTestDay>
GetUserModuleData() :IQueryable<UserModule>
PopulateAssignedModuleData(int, string) :IEnumerable<UserAssignedCardModules>
GetUserConditionData() :IQueryable<UserCondition>
GetModuleConditionData() :IQueryable<ModuleCondition>
GetUserAData() :UserA
SignOnUserTestDay()
SignOnTestDay(int, IEnumerable<string>, FormCollection) :ActionResult
GetUserTestDay(int) :UserTestDay
GetUserTestDayData() :IQueryable<UserTestDay>
UpdateUserTestDayUserModules(IEnumerable<string>, ICollection<UserModule>, int, string) :bool
GetUserA() :
UserA
GetModuleConditionData() :
ModuleCondition
GetUserConditionData() :
UserConditionData
GetUserCardData() :
UserCard
GetConsultationData() :
Consultation
GetResultData() :
Result
UpdateUserTestDay(UserTestDay)

Účet


sd Účet
:Student
:PersonalInformation
:Password
:AccountController
:UserService
:MailService
:UserA
:Address
ShowPersonalInformation()
PersonalInformation() :
ActionResult
GetUserA(string) :
UserA
GetUserAData() :
UserA
GetAddressData() :
Address
ChangePassword()
Password(PasswordModel) :
ActionResult
SetPassword(string, string) :
bool
SendEmail(MailAddress, MailAddress, string,
string)

Žádosti


sd Žádosti
:Student
:Index
:ShowModules
:CardRequestController
:ModuleService
:CardService
:UserModule
:UserCard
:Module
:Card
ShowCards()
Index() :ActionResult
AssignedCardsState(string) :IEnumerable<UserAssignedCard>
GetUserCardData() :IQueryable<UserCard>
GetCardData() :IQueryable<Card>
ShowCardModules()
SelectCardModules(int) :ActionResult
GetUserModuleQueryable() :IQueryable<UserModule>
GetUserModuleData() :IQueryable<UserModule>
GetModuleData() :IQueryable<Module>