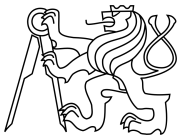
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Card-Manager: aplikace pro správu RFID karet
Student:	Martin Vondrák
Vedoucí:	Ing. Tomáš Kadlec
Studijní program:	Informatika
Studijní obor:	Web a multimédia
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2017/18

Pokyny pro vypracování

Cílem této práce je vytvořit webovou aplikaci card-manager, která bude poskytovat uživatelské rozhraní pro správu čipových karet přidělených identitám. Využita bude s aplikací acs-controller (vyvinuto OICT FIT ČVUT).

* Eviduje se vztah identita-karta. Jedna identita může mít přiřazeno více karet.
* Aplikace bude poskytovat RESTful API pro identifikaci osoby na základě čísla karty.
* V aplikaci bude možné zadat oprávněné uživatele, kteří provádějí evidenci a umožní nastavení zástupu.

Aplikace bude fungovat v režimu

(1) samostatném s lokálním úložištěm sdíleném s aplikací acs-manager,

(2) propojeném s IdM (Identity Management), identity budou uloženy pouze v IdM, evidované karty se přes IdM propisují do LDAP.

Zjistěte požadavky na aplikaci, proveďte jejich analýzu. Navrhněte webovou aplikaci a její API. Implementujte aplikaci a otestujte (jednotkové, funkční a integrační testy - směrem k IdM). Práce se řídí dokumentem Pravidla a zásady projektů FIT v aktuální verzi.

Seznam odborné literatury

* Pravidla a zásady projektů FIT - https://goo.gl/j7nYj0
* Další literaturu dodá vedoucí práce v průběhu dle potřeby.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 6. února 2017

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Bakalářská práce

Card-Manager: aplikace pro správu RFID karet

Martin Vondrák

Vedoucí práce: Ing. Tomáš Kadlec

16. května 2017

Poděkování

Děkuji vedoucímu práce Ing. Tomáši Kadlecovi a oponentovi práce Ing. Jiřímu Špačkovi za pravidelné schůzky a cenné rady. Také děkuji mé rodině a všem, kteří mě v průběhu studia podporovali. A Lubošovi.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 16. května 2017 .....

Odkaz na tuto práci

Vondrák, Martin. *Card-Manager: aplikace pro správu RFID karet.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Cílem této bakalářské práce je implementace systému pro evidenci a přidělení čipových karet. V rešeršní části se zabývám převážně analýzou požadavků kladených na systém a možnostmi integrace s dalšími systémy. V praktické části navazuji návrhem a implementací systému, který tyto požadavky splní a bude možné jej integrovat s dalšími systémy. Systém je řešen formou webové aplikace, která jako zdroj dat používá lokální databázi nebo jiný externí systém. Podařilo se vytvořit systém, který nově umožňuje centrálně evidovat čipové karty a přidělovat je osobám. Výsledek této práce je užitečný zejména zástupcům oddělení a kateder na Fakultě informačních technologií ČVUT v Praze, kteří mají v kompetenci přidělování karet.

Klíčová slova informační systém, webová aplikace, implementace, správa čipových karet, Symfony, PHP

Abstract

Goal of this bachelor's thesis is implementation of system for managing chip cards. I analyze the system requirements and possibilities of integration with other systems in research part. In following practical part, I design and develop the system, that will meet all the reqirements and it will be possible to integrate it with other systems. System is developed as a web application, that uses local database or other external system as data source. System for managing records of chip cards and for assigning those cards to persons was developed. Outcome of this thesis is useful mainly for representatives of departments at Faculty of Information Technology, CTU in Prague, who are responsible for assigning chip cards.

Keywords information system, web application, implementation, management of chip cards, Symfony, PHP

Obsah

Seznam obrázků

Seznam tabulek

Úvod

V současné době se v řadě institucí, jako jsou vysoké školy nebo střední a velké podniky, používají čipové karty k různým účelům. Mezi tyto účely patří například řízení přístupu do místností, evidence docházky nebo placení v místní jídelně. V souvislosti s tím je nutné uspokojivě vyřešit evidenci a přidělování těchto karet jednotlivým osobám.

V rámci realizace projektu acs-controller[1] na Fakultě informačních technologií ČVUT v Praze, jehož cílem bylo řešení přístupu do respirií a některých speciálních místností, vznikla potřeba přidělovat přenosné čipové karty osobám existujícím pouze v rámci fakultního IdM.

Práce bude prospěšná především zástupcům jednotlivých oddělení a kateder na Fakultě informačních technologií ČVUT v Praze, kteří mají ve svých kompetencích přidělování přenosných čipových karet. Při splnění určitých předpokladů bude řešení přenositelné i do prostředí jiných, nejen vzdělávacích, institucí.

V práci se zabývám analýzou, návrhem a implementací webové aplikace Card-Manager, kterou bude možné provozovat ve dvou režimech. Jeden z režimů využívá jako úložiště lokální databázi. Druhý režim využívá místo lokálního úložiště externí systém.

Celá práce se řídí dokumentem Pravidla a zásady projektů FIT[2], dále jen „Pravidla“. V některých fázích však nemusí být Pravidla dodržena v plném rozsahu. V takovém případě jsou odchylky zdokumentovány a odůvodněny.

V analytické fázi práce se zabývám sběrem hypotéz a následným sestavením požadavků na aplikaci. V návrhové fázi navrhuji koncepty řešení dílčích částí aplikace. Tyto koncepty vychází z poznatků získaných v předchozí fázi. V implementační fázi je popsán samotný průběh implementace výše zmíněných konceptů.

Tato práce svou business doménou úzce souvisí s paralelně vznikající prací studenta Michala Pěcha s názvem ACS-Manager: aplikace pro správu pravidel přístupu do místností. Dále je tato práce úzce napojena na projekt oddělení

ICT FIT ČVUT s názvem Identity, který poskytuje aplikaci Card-Manager osoby organizace.

Kapitola 1

Cíl práce

Cílem řešeršní části práce je analýza samotné problematiky a získání určitého nadhledu, aby bylo možné problém zasadit do kontextu ostatních systémů fungujících na FIT ČVUT v Praze. Dále je nutné seznámit se s problematikou z hlediska uživatelských potřeb, aby byl maximalizován přínos pro uživatele systému.

Cílem praktické části práce je návrh implementace webové aplikace, kterou bude možné provozovat ve dvou režimech. První režim je provoz s externími systémy fakulty, které slouží jako úložiště. V druhém režimu má aplikace Card-Manager vlastní lokální úložiště. Dále je cílem následná realizace tohoto návrhu, včetně pokrytí automatizovanými testy.

Kapitola 2

Analytická fáze

V analytické fázi tvorby aplikace Card-Manager zkoumám současný stav a existující řešení, dále provádím uživatelský průzkum. Na závěr této fáze stanovuji požadavky na aplikaci. Celá fáze se řídí Pravidly. Některé části neprovádím v plném rozsahu, ale v takovém případě naleznete v příslušné části odůvodnění.

2.1 Průzkum existujících řešení a jeho vyhodnocení

V této podkapitole představím současný stav problematiky evidence a přidělování čipových karet v kontextu Fakulty informačních technologií ČVUT v Praze, protože zadavatelem práce na aplikaci Card-Manager je oddělení ICT fakulty. Jelikož potřeby fakulty jsou specifické, nebudu provádět průzkum existujících řešení třetích stran.

2.1.1 Vymezení problematiky v rámci FIT ČVUT v Praze

Na ČVUT v Praze existuje několik typů čipových karet[3] uvedených níže. Oficiální terminologií se nazývají průkazy, protože plní mnoho dalších funkcí. Tyto funkce nejsou v kontextu této práce relevantní, proto se dále budeme držet termínu čipová karta.

- Průkaz studenta ČVUT[4]
- Průkaz ISIC se znakem ČVUT[4]
- Průkaz typu OSOBNÍ (s kontaktním čipem)[5]
- Průkaz typu PŘENOSNÝ[6]

Vydání karty některého z prvních tří typů je podmíněno existující identitou v rámci ČVUT v Praze, se kterou je karta svázaná. Tyto karty vydává jednotlivým koncovým uživatelům přímo Vydavatelství průkazů ČVUT. Vydání přenosné čipové karty v tomto směru podmíněno není. Karta je vázaná na organizační jednotku, které byla na její žádost vydána Vydavatelstvím průkazů ČVUT. Jednotlivé organizační jednotky si samy stanoví pravidla pro přidělování karet a jejich užívání.[7]

2.1.2 Současný stav na FIT ČVUT v Praze

Z přechozí podkapitoly vyplývá, že organizační jednotka je zodpovědná za evidenci svých karet. V současné době na FIT ČVUT v Praze neexistuje systém, který by centrálně evidoval, komu organizační jednotka kartu přidělila k použití. Každá jednotka tak řeší evidenci svépomocí různými prostředky. Existují však minimálně dva důvody, které ukazují, že je žádoucí na FIT ČVUT v Praze takovou evidenci mít.

Prvním důvodem je, že karta může být přidělena identitě, existující pouze v rámci Fakulty informačních technologií ČVUT v Praze. Díky tomu může karta fungovat i v jiných fakultních systémech. Konkrétně se může jednat o aplikace ACS-Manager pro správu přístupu do místností nebo studentské systémy jako ProgTest[8]. Druhým důvodem je možnost zajistit přehled o distribuci přenosných karet konkrétním osobám.

2.2 Uživatelský průzkum

V této podkapitole dle Pravidel popisuji části kvalitativní a kvantitativní průzkum, které byly sjednoceny. Sjednocení proběhlo po dohodě s vedoucím práce, že kvůli nízkému počtu potenciálních uživatelů nebudu provádět kvantitativní průzkum.

Kvalitativní průzkum probíhal primárně formou rozhovorů s vedoucím oddělení ICT Ing. Tomášem Kadlecem a se správcem fakultního IdM Ing. Jiřím Špačkem. V neposlední řadě se schůzek účastnil Ing. Martin Bílý, který spravuje karty a identity v rámci systému K4. Na základě těchto rozhovorů byl vytvořen seznam hypotéz, které shrnují všechny poznatky získané o aplikaci Card-Manager.

- Základními subjekty aplikace jsou karty a osoby.
- Mezi osobou a kartou lze nastavit vazbu, která říká, že osoba je držitelem karty.
  - Vazba obsahuje časovou platnost vazby od-do,
  - osobu, která vazbu vytvořila (typicky zástupce organizační jednotky),

  - datum vzniku.

- Organizační jednotky slouží k hierarchickému rozčlenění a nastavení zodpovědností.
- Organizační jednotce lze stanovit manažera (či manažery) z množiny osob.
- Každá karta patří jedné nebo žádné organizační jednotce.
- Aplikace musí umožňovat provoz ve dvou režimech.
  - Lokální režim se samostatným úložištěm dat. V tomto režimu aplikace sdílí část úložiště s aplikací ACS-Manager.
  - Provoz se závislostí na IdM, které bude přes API fungovat jako úložiště dat.
- Aplikace musí logovat operace, které provedla do centrálního logu.
- Aplikace musí poskytovat webové rozhraní pro správu držitelů karet.
- Aplikace musí poskytovat RESTful API pro převedení čísla karty na osobu.
- Manažer organizační jednotky může za sebe zvolit zástup.
- Hlavním správcem aplikace je osoba s rolí administrátora.

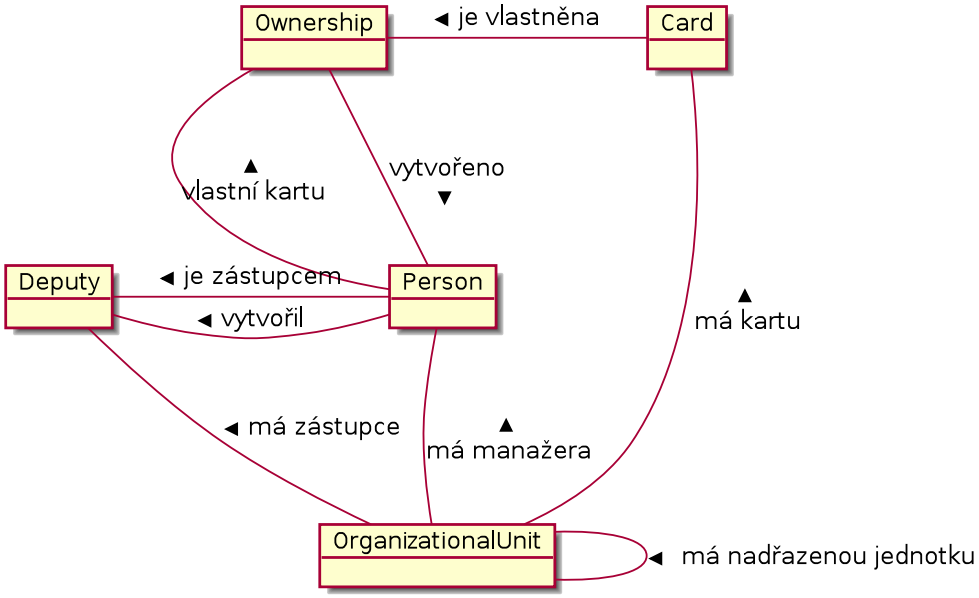
Závislosti mezi jednotlivými daty jsou pro zjednodušení a lepší představivost zachyceny diagramem 2.1. U subjektů Person a Card je zřejmé, co reprezentují, dále OrganizationalUnit reprezentuje organizační jednotku. Organizační jednotka může mít volitelně nadřazenou jednotku. Ownership reprezentuje vazbu, která říká, že osoba je držitelem karty. Deputy reprezentuje informace o zástupci manažera organizační jednotky.

2.3 Požadavky na kvality

V této podkapitole dokumentuji odchylky od požadavků na kvality, kladené na aplikaci Card-Manager. Požadavky na kvality, které zde nejsou uvedeny, jsou zcela v souladu s Pravidly. Struktura kapitoly odpovídá struktuře Pravidel.

2.3.1 Architektura a integrace do infrastruktury

V této části se zaměřím na odchylky od požadavků ze stejnojmenné kapitoly v dokumentu Pravidla.



Obrázek 2.1: Diagram vztahů mezi datovými subjekty

První odchylkou je správa uživatelů. Dle Pravidel je vyžadováno, aby byla správa uživatelů zajištěna fakultním IdM. To není ze zřejmých důvodů v lokálním režimu možné. Naopak v IdM režimu provozu není možné ze stejně zřejmých důvodů, aby aplikace Card-Manager byla provozuschopná i v případě výpadku IdM.

Dále není dodržen požadavek na zařazení aplikace do katalogu služeb FIT od počátku práce na projektu. V první fázi také aplikace nevyužívá mezipaměti pro urychlení obsluhy požadavků. S využitím mezipaměti se však počítá v budoucnu a bylo na to myšleno ve všech fázích projektu.

2.3.2 Webová přístupnost

V této části opět popisuji odchylky od požadavků na kvality ze stejnojmenné kapitoly z dokumentu Pravidla.

Odchylkou této části je nedodržení některých požadavků WCAG 2.0 AA. Zejména jde o část progressive enhancement, kterou nebylo možné dodržet při procesu načítání karty. Obecně lze říct, že aplikace pro svůj běh vyžaduje JavaScript.

2.3.3 Verzování

V části Verzování, která opět koresponduje se stejnojmennou kapitolou v Pravidlech, popisuji odchylky od požadavků na verzování zdrojového kódu apli-

kace.

První odchylkou je nepoužití Git Flow branching modelu. Místo toho jsem použil GitLab Flow model, které je na rozdíl od Git Flow méně komplexní a v některých případech usnadní práci.[9] Aplikace také není dostupná ve třech verzích (vývojová, testovací a produkční), jak vyžadují Pravidla.

2.4 Požadavky na funkce

V této části naleznete požadavky na funkce, které vyplynuly z hypotéz získaných během kvalitativního průzkumu. Veškeré níže uvedené požadavky je nutné implementovat.

1. Karty
    a) Přidělení karty osobě
    b) Úprava přidělení karty
    c) Odebrání karty osobě
    d) Rozhraní pro převedení ID karty na osobu (v lokálním režimu)
    e) Registrace nové karty (v lokální režimu)
    f) Odstranění karty (v lokálním režimu)
2. Organizační jednotky
    a) Přiřazení karty do organizační jednotky
    b) Odebrání karty organizační jednotce
    c) Odebrání karty osobě
    d) Vytvoření nové organizační jednotky (v lokálním režimu)
    e) Úprava organizační jednotky (v lokální režimu)
    f) Odstranění organizační jednotky (v lokálním režimu)
3. Osoby
    a) Stanovení zástupce manažera organizační jednotky
    b) Stanovení manažera organizační jednotky (v lokálním režimu)
    c) Stanovení administrátora aplikace (v lokálním režimu)

Požadavky, které mají v závorce uvedeno „v lokálním režimu“, jsou nezbytné pouze při provozu s lokálním úložištěm. V IdM režimu provozu jsou takto označené požadavky implementovány v jiných informačních systémech. Naopak pokus o využití takových funkcí v IdM režimu způsobí chybu.

Kapitola 3

Návrhová fáze

V návrhové fázi tvorby aplikace Card-Manager se zabývám konceptuálním návrhem implementace aplikace, což je odchylka od Pravidel. Z Pravidel je zde realizována pouze část případů užití a návrh Lo-Fi prototypů UI. Zbytek byl po domluvě s vedoucím práce vynechán.

3.1 Případy užití

V této části bakalářské práce představím všechny případy užití aplikace Card-Manager a jejich aktéry. Případy jsou rozdělené do dvou skupin podle režimu provozu aplikace.

3.1.1 Aktéři

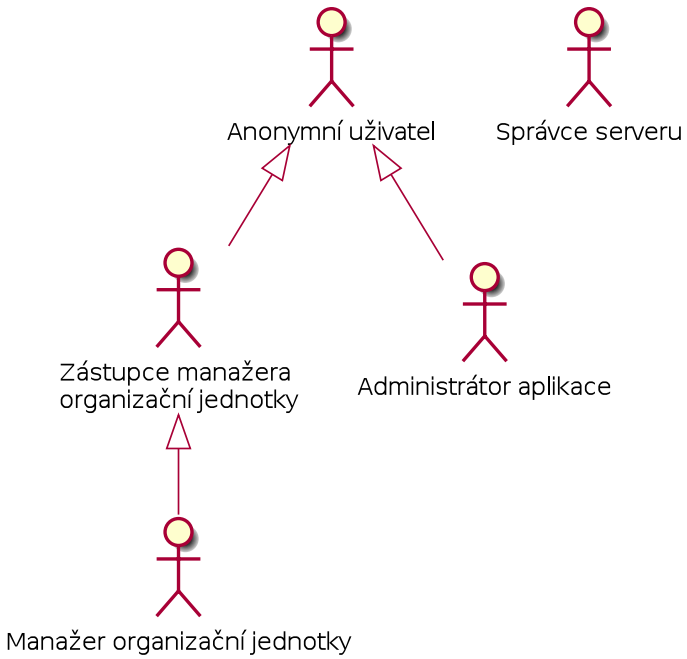
V této části popisuji jednotlivé aktéry a jejich vztahy, kteří vystupují v případech užití aplikace Card-Manager. Pro snažší představu je také uveden diagram 3.1 se všemi aktéry.

3.1.1.1 Anonymní uživatel

Anonymní uživatel nemá v aplikaci Card-Manager oprávnění na žádnou operaci kromě přihlášení. Pokud se mu podaří úspěšně přihlásit, stává se z něj uživatel s právy minimálně v rozsahu zástupce manažera organizační jednotky.

3.1.1.2 Správce serveru

Správce serveru je velmi speciální případ aktéra. Svoje využití najde pouze v případě provozu v lokálním režimu. Při nasazení je totiž nutné vygenerovat pomocí příkazu určité údaje, které jsou v druhém režimu získávány z IdM.



Obrázek 3.1: Aktéři vystupující v aplikaci Card-Manager

3.1.1.3 Zástupce manažera organizační jednotky

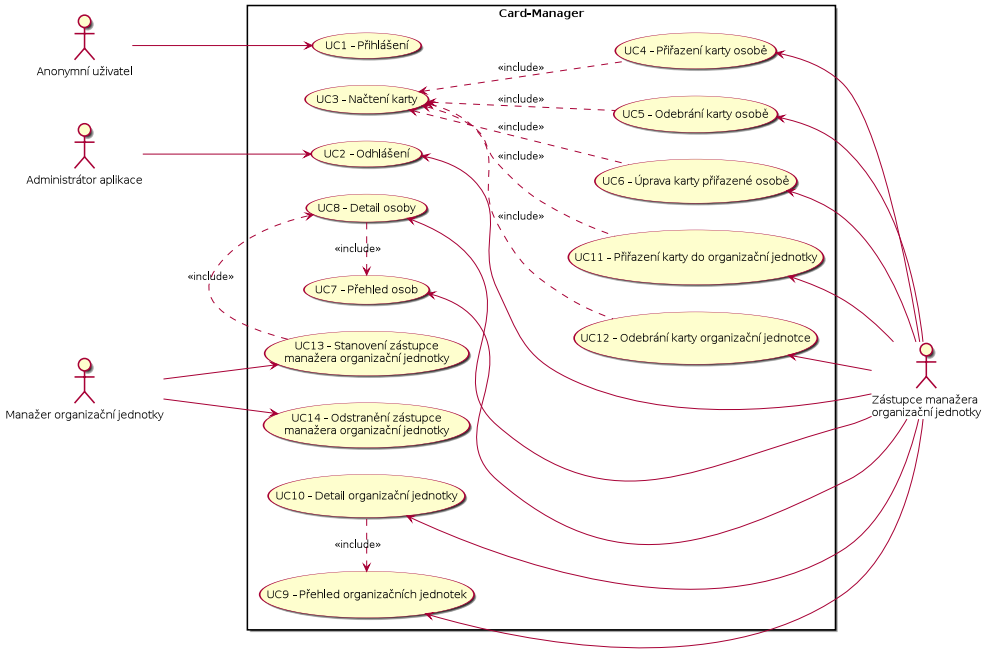
Zástupce manažera organizační jednotky je zodpovědný za nakládání s kartami jeho organizační jednotky. Například přidělení nebo odebrání karty osobě. Tato práva získal na základě časově omezeného pověření od manažera organizační jednotky.

3.1.1.4 Manažer organizační jednotky

Manažer organizační jednotky má stejná práva jako zástupce manažera organizační jednotky, ale nejsou nějak časově omezena. Navíc může ještě zvolit své zástupce. Svoje práva získal od administrátora v případě lokálního režimu nebo na základě speciální role v IdM.

3.1.1.5 Administrátor aplikace

Administrátor aplikace je aktér, který má nejvyšší oprávnění v rámci aplikace Card-Manager a je zodpovědný za její údržbu. Například vytváření osoby nebo organizační jednotky při používání v lokálním řežimu. Svoje oprávnění získává od správce serveru při nasazování aplikace.



Obrázek 3.2: Případy užití v obou režimech provozu aplikace

3.1.2 Specifikace případů užití

V této části podrobně rozepisuji jednotlivé případy užití popsány případy užití, které můžeme rozdělit do dvou základních skupin. První skupinou jsou takové případy, které dávají smysl bez ohledu na to, v jakém režimu je aplikace provozována. Druhou skupinou jsou případy, které vznikají pouze v lokálním režimu, kdy je potřeba nahradit některé funkce IdM.

3.1.2.1 Společné případy užití

V této podkapitole popisuji případy užití, které jsou společné pro oba režimy využívání aplikace. Jsou to případy, které souvisí zejména s naplněním hlavního cíle aplikace, tedy přiřazení čipových karet do organizačních jednotek a následně jednotlivým osobám. Pro přehlednost jsou případy užití zobrazené v kontextu aktérů, kteří do těchto případů vstupují, v diagramu 3.2.

UC1 – Přihlášení Cílem tohoto případu užití je autentizovat uživatele pro další práci s aplikací, jelikož z bezpečnostních důvodů není umožněna práce s aplikací anonymním uživatelům.

Aktér: Anonymní uživatel
Předpoklad: Uživatel zná své přístupové údaje.
Scénář:

1. Systém zobrazí formulář pro přihlášení.
2. Uživatel vyplní uživatelské jméno a heslo a klikne na ovládací prvek pro přihlášení.
3. Systém požadavek zpracuje a zobrazí úvodní obrazovku.

UC2 – Odhlášení V tomto případu užití je cílem ukončení sezení autentizovaného uživatele. **Aktér:** Zástupce manažera organizační jednotky
Scénář:

1. Uživatel klikne na tlačítko „Odhlásit se“.
2. Systém odhlásí uživatele.

UC3 – Načtení karty Cílem tohoto případu užití je načíst čipovou kartu pomocí čtecího zařízení. Načtení karty je předpokladem pro úspěšné provedení dalších vybraných operací s kartou.
Aktér: Zástupce manažera organizační jednotky
Předpoklad: Čtecí zařízení je připojené k PC.
Scénář:

1. Uživatel klikne na tlačítko „Načíst kartu“.
2. Uživatel přiloží kartu ke čtecímu zařízení.
3. Systém zobrazí základní informace o kartě a ovládací prvky pro operace s kartou dle role uživatele.

UC4 – Přidělení karty osobě V tomto případu užití je cílem uživatele kartu přidělit existující osobě v rámci aplikace.
Aktér: Zástupce manažera organizační jednotky
Předpoklad: Čtecí zařízení je připojené k PC.
Scénář:

1. Scénář začíná případem užití UC3 – Načtení karty.
2. Uživatel klikne na tlačítko „Přidělit kartu“.
3. Systém zobrazí formulář pro přidělení karty.
4. Uživatel ve formuláři vyplní:
   a) osobu, které chce kartu přidělit,

b) časový interval, po který je přidělení platné.

5. Uživatel odešle formulář.

6. Systém zpracuje formulář a zobrazí výsledek operace.

UC5 – Odebrání karty osobě Cílem tohoto případu užití je odebrat právě platné přidělení karty.
Aktér: Zástupce manažera organizační jednotky
Předpoklad: Načtená karta je někomu přidělena.
Scénář:

1. Scénář začíná případem užití UC3 – Načtení karty.

2. Uživatel klikne na tlačítko „Odebrat kartu“.

3. Systém zpracuje požadavek a zobrazí výsledek operace.

UC6 – Úprava karty přidělené osobě V tomto případu užití je cílem upravit právě platné přidělení karty.
Aktér: Zástupce manažera organizační jednotky
Předpoklad: Čtecí zařízení je připojené k PC.
Scénář:

1. Scénář začíná případem užití UC3 – Načtení karty.

2. Uživatel klikne na tlačítko „Upravit přidělení“.

3. Systém zobrazí formulář pro upravení přidělení karty.

4. Uživatel ve formuláři vyplní časový interval, po který je přidělení platné.

5. Uživatel odešle formulář.

6. Systém zpracuje formulář a zobrazí výsledek operace.

UC7 – Přehled osob Cílem tohoto případu užití je zobrazit si seznam všech osob registrovaných v systému.
Aktér: Zástupce manažera organizační jednotky
Scénář:

1. Uživatel klikne na položku „osoby“ v menu.

2. Systém zobrazí stránkovaný seznam osob s filtrací.

UC8 – Detail osoby V tomto případě užití je cílem zobrazit podrobné informace o osobě a ovládací prvky přidružených operací s osobou.
Aktér: Zástupce manažera organizační jednotky
Scénář:

1. Scénář začíná případem užití UC7 – Přehled osob.
2. Uživatel klikne na konkrétní osobu.
3. Systém zobrazí detailní informace o osobě.

UC9 – Přehled organizačních jednotek Cílem tohoto případu užití je zobrazit si seznam všech organizačních jednotek registrovaných v systému.
Aktér: Zástupce manažera organizační jednotky
Scénář:

1. Uživatel klikne na položku „Organizační jednotky“ v menu.
2. Systém zobrazí stránkovaný seznam organizačních jednotek s filtrací.

UC10 – Detail organizační jednotky V tomto případě užití je cílem zobrazit podrobné informace o organizační jednotce a ovládací prvky přidružených operací s jednotkou.
Aktér: Zástupce manažera organizační jednotky
Scénář:

1. Scénář začíná případem užití UC9 – Přehled organizačních jednotek.
2. Uživatel klikne na konkrétní organizační jednotku.
3. Systém zobrazí detailní informace o organizační jednotce.

UC11 – Přiřazení karty do organizační jednotky Cílem tohoto případu užití je přiřadit kartu do organizační jednotky registrované v systému, aby byla stanovena zodpovědnost za kartu.
Aktér: Zástupce manažera organizační jednotky
Scénář:

1. Scénář začíná případem užití UC3 – Načtení karty.
2. Uživatel klikne na tlačítko „Přiřadit kartu do OJ“.
3. Systém zpracuje požadavek a zobrazí výsledek operace.

UC12 – Odebrání karty organizační jednotce V tomto případu užití je cílem odebrat kartu přiřazenou organizační jednotce.
Aktér: Zástupce manažera organizační jednotky
Scénář:

1. Scénář začíná případem užití UC3 – Načtení karty.
2. Uživatel klikne na tlačítko „Odebrat kartu OJ“.
3. Systém zpracuje požadavek a zobrazí výsledek operace.

UC13 – Stanovení zástupce manažera organizační jednotky Cílem tohoto případu užití je stanovit zástupce organizační jednotky. To je nutné pro případy, kdy bude manažer organizační jednotky delší dobu mimo pracoviště.
Aktér: Manažer organizační jednotky
Scénář:

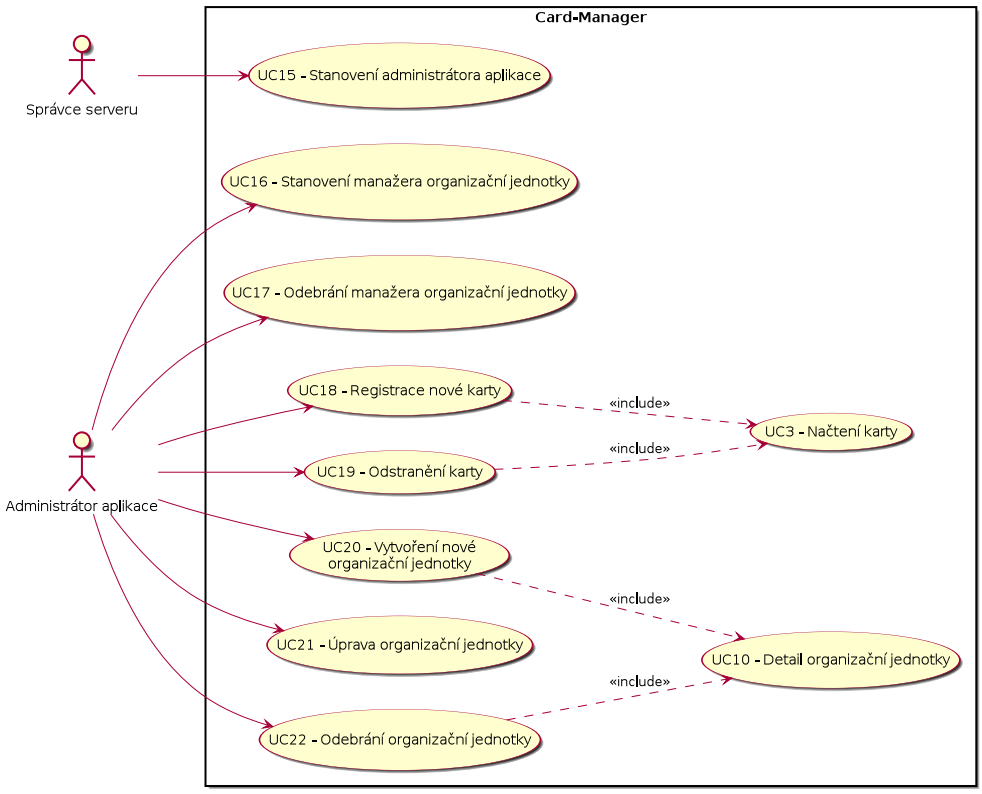
1. Scénář začíná případem užití UC8 – Detail osoby.
2. Uživatel klikne na tlačítko „Jmenovat zástupcem manažera OJ“.
3. Systém zobrazí formulář pro jmenování zástupce manažera OJ.
4. Uživatel ve formuláři vyplní časový interval platnosti a odešle jej.
5. Systém zpracuje formulář a zobrazí výsledek operace.

UC14 – Odstranění zástupce manažera organizační jednotky V tomto případu užití je cílem odstranit již nepotřebného zástupce manažera organizační jednotky.
Aktér: Manažer organizační jednotky
Scénář:

1. Uživatel klikne na tlačítko „Seznam mých zástupců“.
2. Systém zobrazí seznam zástupců pro právě přihlášeného uživatele.
3. Uživatel vybere zástupce, kterého chce odstranit a klikne u něj na tlačítko „Odstranit zástupce“.
4. Systém zpracuje požadavek a zobrazí výsledek operace.

3.1.2.2 Případy užití v lokálním režimu

V této kapitole jsou popsány případy užití, které najdou svoje uplatnění pouze v lokálním režimu provozu aplikace. A to zejména z toho důvodu, že v tomto režimu není k dispozici IdM jako datový zdroj a aplikaci je potřeba naplnit daty jiným způsobem. Stejně jako v předchozí podkapitole je pro přehlednost uveden diagram 3.3 případů užití v kontextu s aktéry.



Obrázek 3.3: Případy užití v lokálním režimu provozu aplikace

UC15 – Stanovení administrátora aplikace V tomto případu užití je cílem stanovení administrátora aplikace pro lokální režim provozu aplikace. V režimu provozu s IdM je administrátor stanoven speciální technickou rolí `T-ACSPRUKAZY-18000-ADMINISTRATOR`, přidělenou v IdM.
Aktér: Správce serveru
Předpoklad: Přístup k CLI při nasazování aplikace.
Scénář:

1. Uživatel si otevře CLI v adresáři, kde je nasazena aplikace.
2. Uživatel spustí připravený příkaz s příslušnými parametry.
3. Systém provede příkaz a vypíše výsledek operace.

UC16 – Stanovení manažera organizační jednotky Cílem tohoto případu užití je stanovení manažera organizační jednotky v lokálním režimu provozu aplikace Card-Manager. Pokud je aplikace provozována v IdM režimu,

manažer OJ je registrován automaticky na základě technické role v IdM, například `T-ACSPRUKAZY-18301-VEDOUCI`.
Aktér: Administrátor aplikace
Scénář:

1. Scénář začíná případem užití UC10 – Detail organizační jednotky.
2. Uživatel klikne na tlačítko „Přidat manažera“.
3. Uživatel ve formuláři vybere osobu, která se má stát manažerem. Následně odešle formulář.
4. Systém zpracuje formulář a zobrazí výsledek operace.

UC17 – Odebrání manažera organizační jednotky V tomto případu užití je cílem uživatele odebrat manažera organizační jednotky v lokálním režimu provozu aplikace. V IdM režimu provozu je tento UC řešen odbráním business role. Viz UC16 – Stanovení manažera organizační jednotky.
Aktér: Administrátor aplikace
Scénář:

1. Scénář začíná případem užití UC10 – Detail organizační jednotky.
2. Uživatel klikne na tlačítko „Seznam manažerů“.
3. Uživatel u manažera, kterého chce odebrat, klikne na tlačítko „Odebrat manažera“.
4. Systém zpracuje požadavek a zobrazí výsledek operace.

UC18 – Registrace karty Cílem tohoto případu užití je zaregistrovat do aplikace Card-Manager novou čipovou kartu. V režimu provozu s IdM je stanoven jiný proces pro registraci karty, který je mimo aplikaci Card-Manager.
Aktér: Administrátor aplikace
Scénář:

1. Scénář začíná případem užití UC3 – Načtení karty.
2. Uživatel klikne na tlačítko „Registrovat kartu“.
3. Systém zpracuje požadavek a zobrazí výsledek operace.

UC19 – Odstranění karty V tomto případu užití je cílem odstranit čipovou kartu zaregistrovanou do aplikace Card-Manager. V režimu provozu s IdM je stanoven jiný proces pro odstranění registrované čipové karty, který je mimo aplikaci Card-Manager.
Aktér: Administrátor aplikace
Scénář:

1. Scénář začíná případem užití UC3 – Načtení karty.
2. Uživatel klikne na tlačítko „Odstranit kartu“.
3. Systém zpracuje požadavek a zobrazí výsledek operace.

Alternativní scénář:

1. Uživatel klikne na tlačítko „Seznam karet“.
2. Systém zobrazí stránkovaný seznam karet s filtrací.
3. Uživatel klikne na konkrétní kartu.
4. Systém zobrazí detail karty.
5. Uživatel klikne na tlačítko „Odstranit kartu“.
6. Systém zpracuje požadavek a zobrazí výsledek operace.

UC20 – Vytvoření nové organizační jednotky Cílem tohoto případu užití je vytvořit v rámci aplikace Card-Manager novou organizační jednotku. V režimu provozu s IdM je stanoven jiný proces na vytvoření OJ, který je mimo aplikaci Card-Manager.
Aktér: Administrátor aplikace
Scénář:

1. Uživatel klikne na tlačítko „Vytvořit organizační jednotku“.
2. Systém zobrazí formulář pro vytvoření nové organizační jednotky.
3. Uživatel vyplní požadované informace a odešle formulář.
4. Systém zpracuje formulář a zobrazí výsledek operace.

UC21 – Úprava organizační jednotky V tomto případu užití je cílem upravit existující organizační jednotku v rámci aplikace Card-Manager. V režimu provozu s IdM je stanoven jiný proces na úpravu OJ, který je mimo aplikaci Card-Manager.
Aktér: Administrátor aplikace
Scénář:

1. Scénář začíná případem užití UC10 – Detail organizační jednotky.
2. Uživatel klikne na tlačítko „Upravit organizační jednotku“.
3. Systém zobrazí formulář pro úpravu organizační jednotky.
4. Uživatel změní požadované informace a odešle formulář.
5. Systém zpracuje formulář a zobrazí výsledek operace.

UC22 – Odebrání organizační jednotky Cílem tohoto případu užití je smazání existující organizační jednotky v aplikaci Card-Manager. V režimu provozu s IdM je stanoven jiný proces na smazání OJ, který je mimo aplikaci Card-Manager.
Aktér: Administrátor aplikace
Scénář:

1. Scénář začíná případem užití UC10 – Detail organizační jednotky.
2. Uživatel klikne na tlačítko „Odebrat organizační jednotku“.
3. Systém zpracuje požadavek a zobrazí výsledek operace.

3.2 Režimy provozu aplikace

Protože v průběhu analytické fáze vznikla potřeba provozovat aplikaci ve dvou režimech, popisuji v této části tyto režimy a také koncept konfigurace aplikace pro provoz obou režimů.

Prvním z režimů provozu aplikace je tzv. lokální režim. V tomto režimu aplikace využívá vlastní databázi pro perzistenci dat. V tomto režimu je možné aplikaci provozovat bez dalších externích systémů.

Druhý režim provozu je připravený speciálně pro potřeby FIT ČVUT v Praze. Jedná se variantu, kdy se jako úložiště dat použije fakultní IdM a pro získání dat se využije služba, která bude mít unifikované rozhraní s implementací odpovědnou za získání dat z databáze. Tato služba bude rozhraní implementovat pomocí komunikace přes API, které bude vystavené fakultním IdM.

Celá aplikace se pak při nasazení zkonfiguruje pomocí parametrů a požadovaná implementace pro manipulaci s daty se vybere použitím návrhového vzoru Strategy[10]. Využije se přístup do lokální databáze nebo komunikace přes API s IdM.

3.3 Datová vrstva

V této části popisuji, s jakou strukturou dat pracuje aplikace Card-Manager. V lokálním režimu provozu bude tato struktura přímo implementována databázovým schématem. V IdM režimu provozu bude struktura perzistentně uložených dat vycházet ze schématu IdM API. Ze strany IdM je nezbytné takové API poskytnout. Díky němu budou dostupná všechna potřebná data.

Část datového modelu, se kterým aplikace Card-Manager pracuje, je stejná jako v aplikaci ACS-Manager. Proto paralelně s aplikacemi Card-Manager a ACS-Manager vzniká aplikace Identity, která tato data a základní operace s nimi sdružuje a dále je poskytuje výše jménovaným aplikacím.

Obecně API poskytované IdM vrací reprezentaci dat ve formátu JSON. Proto je také nutné nalézt vhodný způsob transformace dat z formátu, který poskytuje IdM do vnitřní reprezetace aplikace Card-Manager. V analytické části jsem uvedl, že v první fázi nebude implementováno využití cache. Nicméně v IdM režimu bude nutné toto implementovat v dalších fázích projektu, aby se omezila komunkace po síti a zrychlila se tím aplikace Card-Manager.

Diagram 3.4 zobrazuje vztah mezi jednotlivými entitami. Níže je každá entita podrobněji popsána, včetně atributů z IdM, které odpovídají atributům vnitřní reprezentace dat. Také byla na základě rozhovoru s Ing. Jiřím Špačkem stanovena maximální doba platnosti entity uložené v mezipaměti.

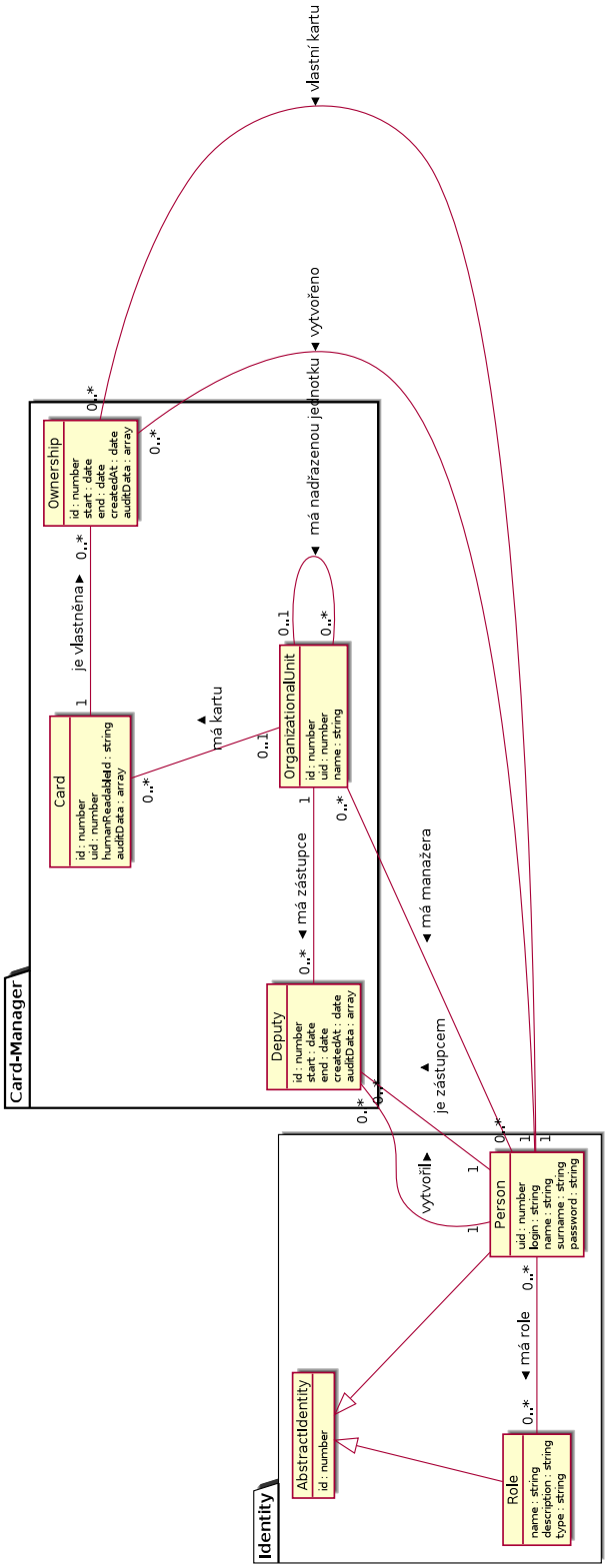
3.3.1 Card

Entita Card reprezentuje čipovou kartu. Každá čipová karta je identifikována svým unikátním identifikátorem. Dále v IdM režimu provozu existuje u karty atribut `inRegistryOf`, který reprezentuje vazbu přiřazení karty do organizační jednotky. Karta může patřit v jednom momentě maximálně jedné organizační jednotce. Tato entita můža být uložena v mezipaměti po 24 hodin. Více informací k entitě Card se nachází v tabulce 3.1.

3.3.2 Deputy

Entita Deputy reprezentuje informace o zastoupení organizační jednotky jinou osobou než manažerem. Může se chápat jako spojovací entita mezi osobou a jednotkou o časový interval, po který je toto zastoupení platné. Každý manažer může těchto zastoupení vytvořit neomezené množství s různými osobami a jedna osoba může zastupovat několik jednotek. Tabulka 3.2 obsahuje další informace o entitě Deputy.

V IdM je zastoupení organizační jednotky realizováno speciální technickou rolí, například `T-ACSPRUKAZY-18301-VEDOUCI-ZASTUPCE`, tudíž neexistují konkrétní atributy, které by bylo třeba transformovat. Tato entita má platnost uložení v mezipaměti 24 hodin.



Obrázek 3.4: Reprezentace dat v aplikaci Card-Manager

Lokální jméno	IdM jméno	Datový typ	Popis
id	_id	integer	unikátní číslo karty v rámci organizace
uid	chipCode	integer	unikátní číslo čipu
humanReadableId	jen lokálně	string	unikátní lidsky čitelný identifikátor karty
auditData	jen lokálně	array	slouží k uložení dat při změně karty (například původní organizační jednotka po jejím smazání)

Tabulka 3.1: Reprezentace čipové karty

Lokální jméno	Datový typ	Popis
id	integer	automaticky generovaný unikátní identifikátor
start	DateTime	začátek platnosti zastoupení
end	DateTime	konec platnosti zastoupení
createdAt	DateTime	datum vytvoření zastoupení
auditData	array	slouží k uložení dat při změně zastoupení (například původní osoba, která zastoupení vytvořila)

Tabulka 3.2: Reprezentace zástupce organizační jednotky

3.3.3 OrganizationalUnit

Entita OrganizationalUnit reprezentuje organizační jednotku. Jednotka může mít přiřazené karty, viz atribut inRegistryOf výše. Organizační jednotka může mít své manažery a také může mít nadřazenou organizační jednotku. Manažeři z rodičovské organizační jednotky mohou spravovat karty příslušející všem potomkům. Platnost uložení entity v mezipaměti je 24 hodin. Další informace k entitě OrganizationalUnit jsou k dispozici v tabulce 3.3.

3.3.4 Ownership

Entita Ownership reprezentuje vlastnictví karty. Je to spojovací entita, která má dvě povinné vazby na kartu a osobu, která danou kartu vlastní. Další vazbou na jinou entitu je osoba, která vlastnictví vytvořila, na rozdíl od předchozích dvou je tato vazba nepovinná. Entita může být v mezipaměti uložena 24 hodin. Více informací k entitě Ownership se nachází v tabulce 3.4.

Lokální jméno	IdM jméno	Datový typ	Popis
id	_id	integer	automaticky generovaný unikátní identifikátor
uid	_id	integer	uživatelem daný unikátní identifikátor, v IdM režimu se převezme generovaný identifikátor
name	displayName	string	jméno jednotky

Tabulka 3.3: Reprezentace organizační jednotky

Lokální jméno	IdM jméno	Datový typ	Popis
id	_id	integer	automaticky generovaný unikátní identifikátor
start	validSince	DateTime	začátek platnosti
end	validUntil	DateTime	konec platnosti
createdAt	jen lokálně	DateTime	datum vytvoření

Tabulka 3.4: Reprezentace vlastnictví karty

3.4 Business logika

V této části popisuji komplikovanější hlavní procesy aplikace Card-Manager. Jednotlivé procesy mohou vykonávat uživatelé s určitým oprávněním. Průběh procesů z uživatelského hlediska jsem popsal v části případy užití. Tato podkapitola se procesy zabývá z hlediska omezení, která jsou na ně kladena.

3.4.0.1 Přidělení karty osobě

Proces přidělení karty osobě je klíčový proces, který se v aplikaci Card-Manager vykonává. Zástupce nebo manažer organizační jednotky může přidělit pouze takovou kartu, která patří do jednotek, které spravuje nebo do jednotek, které hierarchicky spadají pod výše zmíněné jednotky spravované uživatelem.

Také je možné přidělit kartu, která není momentálně přiřazena žádné organizační jednotce. V takovém případě se karta před přidělením osobě automaticky přiřadí první nalezené organizační jednotce podle uživatele, který celý proces zahájil.

Jednotka zmíněná v předchozím odstavci se hledá následujícím způsobem. Nejprve se vyhledají všechny jednotky, kde je uživatel, který proces inicioval manažerem a použije se první nalezená jednotka. Pokud se žádná taková jednotka nenajde, začnou se vyhledávat jednotky, kde je uživatel zástupcem a použije se první nalezená jednotka. Nalezení jednotky je vždy garantováno,

protože pouze zástupce nebo manažer organizační jednotky může vykonat tento proces.

Při přidělování karty se také kontroluje jestli je takové přidělení možné provést z hlediska času. Konkrétně datum začátku platnosti přidělení nesmí být větší než konce platnosti. Dále se kontroluje jestli takové přidělení časově nekoliduje s jiným přidělením stejné karty. Za kolizi se považuje i přidělení stejné osobě s překryvem časových intervalů.

3.4.0.2 Odebrání karty osobě

Proces odebrání karty osobě se v kontextu aplikace Card-Manager dá rozdělit na dva případy. Prvním je vypršení časové platnosti přidělení a druhým případem je předčasné ukončení. Nicméně oba případy se řeší pouze pomocí data konce platnosti přidělení.

V prvním případě je zřejmé, že se s datem konce platnosti nic nestane. Při případném ověření přidělení bude aktuální datum větší než datum konce platnosti a přidělení je tak považováno za neplatné.

V druhém případě se nastaví datum konce platnosti přidělení na aktuální datum, pokud je přidělení momentálně platné, tzn. není naplánováno do budoucna a tedy začátek platnosti je menší než aktuální datum. V opačném případě se nastaví datum konce platnosti stejné jako datum začátku platnosti.

3.4.0.3 Stanovení zástupce organizační jednotky

Proces stanovení zástupce organizační jednotky vykonává manažer jednotky. Manažer může stanovit zástupce přímo pro jednotky, kterých je manažerem, ale také pro jednotky, které hierarchicky patří pod výše zmíněné jednotky.

Zastoupení je platné jen po určitý časový interval. Stejně jako v případě přidělení karty je potřeba ověřit jestli je daný interval platný, tedy zda-li začátek platnosti není pozdější datum než konec platnosti.

V lokálním režimu je osobě, která zastoupení vykonává, přidělena aplikační role `ROLE_CM_DEPUTY`. V IdM režimu je pak IdM zaslán požedavek na přidělení technické role `T-ACSPRUKAZY-18301-VEDOUCI-ZASTUPCE`, kde číselné označení identifikuje organizační jednotku. Z této technické role se při přihlášení dané osoby získají všechny potřebné informace pro správné vypočítání zástupu.

3.4.0.4 Odebrání zástupce organizační jednotky

Proces odebrání zástupce organizační jednotky se v kontextu aplikace Card-Manager dá rozdělit na dva případy, stejně jako v procesu odebrání karty osobě. Oba případy jsou řešeny identicky, jako ve výše zmíněném procesu.

Při odebrání zástupce organizační jednotky je však potřeba vyřešit ještě jeden problém a tím je odebrání aplikační role `ROLE_CM_DEPUTY` osobě zástupce. Při standardním vypršení platnosti není možné z pricipu fungování webové aplikace roli ihned odebrat. Řešením je v pravidelných intervalech

spouštět na serveru příkaz, například pomocí démonu Cron[11], který projde všechny osoby s touto rolí a pokud k nim nenajde platné pravidlo zastoupení, tuto roli odebere.

Pro zachování konzistence se tento příkaz použije i v případě předčasného odebrání zástupce organizační jednotky. Výše zmíněný příkaz je užitečný pouze v lokálním režimu provozu. V IdM režimu provozu je zodpovědnost za odstranění adekvátní technické role přenechána IdM.

3.4.0.5 Stanovení manažera organizační jednotky

Proces stanovení manažera organizační jednotky vykonává administrátor aplikace Card-Manager a pouze v lokálním režimu. Jako administrátor může stanovit manažery pro všechny jednotky registrované v aplikaci.

Podobně jako při stanovení zástupce organizační jednotky, je stanovené osobě přidělena aplikační role, v tomto případě jde o roli `ROLE_CM_MANAGER`. V IdM režimu je adekvátní technická role `T-ACSPRUKAZY-18301-VEDOUCI`, ze které se získají informace pro přiřazení aplikační role. Za přidělení technické role v IdM režimu provozu nezodpovídá aplikace Card-Manager, takže IdM nelze zaslat takový požadavek.

3.4.0.6 Odebrání manažera organizační jednotky

Tento proces může realizovat pouze administrátor aplikace. V návaznosti na přechozí proces, je i tento podporován pouze v lokálním režimu. Odebrání aplikační role je, jako v případě procesu odebrání zástupce organizační jednotky, provedeno příkazem, který se spouští v pravidelných intervalech.

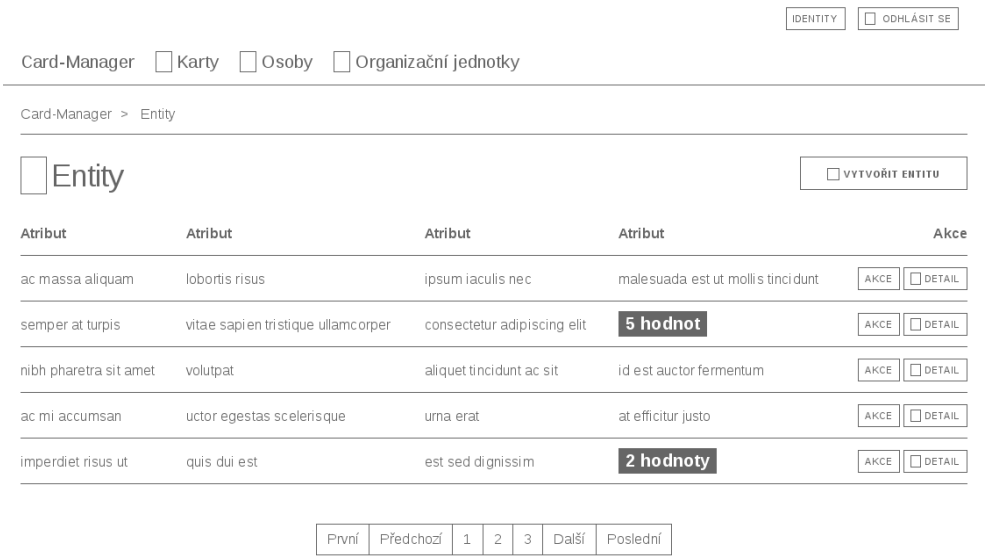
3.5 Rozhraní

V této podkapitole se zabývám popisem rozhraní, které aplikace Card-Manager poskytuje. Rozhraním je myšleno nejen uživatelské rozhraní, ale také RESTful API, které musí aplikace poskytovat, jak vyplynulo z analytické části.

3.5.1 Uživatelské rozhraní

V této části popisuji uživatelské rozhraní aplikace Card-Manager. Jelikoz bakalářská práce, stejně jako jiné SW projekty, má omezenou časovou dotaci, bylo nutné některé požadavky Pravidel nebo běžných postupů přizpůsobit či dokonce zanedbat.

Cílem je mít uživatelské rozhraní jednotné s dalšími aplikacemi používanými v rámci oddělení ICT. Zejména pak s aplikacemi ACS-Manager a Identity, které svou business doménou velmi úzce souvisí s aplikací Card-Manager a předpokládá se, že tyto aplikace budou používané společně.

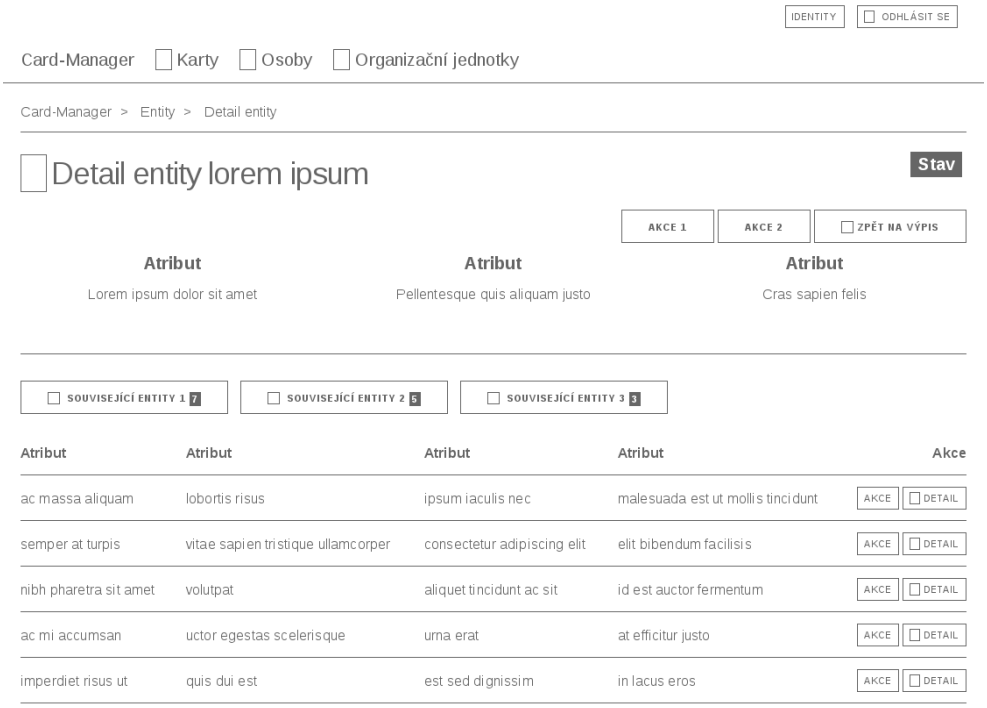


Obrázek 3.5: Drátěný model přehledu entit

Obecně aplikace poskytuje dvě základní obrazovky, a to přehled entit a detail konkrétní entity s akcemi, které uživatel s danou entitou může provádět. Tyto obrazovky vychází z drátěných modelů 3.5 a 3.6. Pro každou entitu je pak rozepsáno, které konkrétní údaje jsou na obrazovce uvedeny.

K přehledu entit můžeme obecně vztáhnout následující informace. Text „Entity“ z drátěného modelu se nahradí jménem entity, se kterou pracujeme v množném čísle, například „Karty“. Obdobně se přizpůsobí tlačítko „Vytvořit entitu“, které se zobrazuje podle dostupnosti této operace. V případě, že v tabulce entit má atribut více hodnot, jsou sdruženy do štítku, kde je pouze informace o tom, kolik hodnot tento atribut má. Prázdné obdélníky vedle nadpisu, položek v menu apod. značí piktogramy, které budou použité pro lepší orientaci uživatelů.

Podobně jako v případě přehledu entit, byla pro detail entity stanovena obecně platná pravidla. Text „Entity“ z drátěného modelu je opět nahrazen názvem entity v odekvátním tvaru. Prázdné obdélníky vedle nadpisu, položek v menu apod. opět reprezentují piktogramy. Štítek „Stav“ v pravém horním rohu ukazuje stav ve kterém se entita nachází. Ne všechny entity však mají vnitřní stavy. Tlačítka s textem „Související entity“ slouží jako záložky k přepínání tabulek s přehledem souvisejících entit pod nimi.



Obrázek 3.6: Drátěný model detailu entity

3.5.1.1 Card

Entita Card reprezentující čipovou kartu nemá ve svém přehledu tlačítko pro, vytvoření nové karty. Tabulka s kartami pak obsahuje následující atributy:

- číslo karty,
- název (v lokálním režimu),
- organizační jednotka,
- v akcích je pouze odkaz na detail karty.

V detailu karty je štítek se stavem, který indikuje, jestli je karta přidělená, či nikoliv. Atributy se shodují s přehledem karet. V lokálním režimu jsou pro administrátora aplikace k dispozici akce pro smazání a editaci karty. Dále detail karty obsahuje tři záložky souvisejících entit. Koknrétně aktuální přidělení, nadcházející přidělení a přechozí přidělení. Každé přidělení z tabulky pak obsahuje atributy:

- stav,

- držitel,
- začátek platnosti,
- konce platnosti,
- přidělil,
- v akcích je pouze odkaz na detail přidělení.

3.5.1.2 Person

Entita Person reprezentuje osobu organizace. Protože za tato data je zodpovědná aplikace Identity, přehled osob je pouze pro čtení a nemá tlačítko pro vytvoření nové osoby. Tabulka s osobami obsahuje následující atributy:

- osobní číslo,
- uživatelské jméno,
- jméno,
- příjmení,
- v akcích je pouze odkaz na detail karty.

V detailu osoby se nachází atributy jméno a příjmení, osobní číslo a uživatelské jméno. Také je zde podle oprávnění uživatele tlačítko pro stanovení zástupce organizační jednotky. Dále se zde nachází pět záložek souvisejících entit. Koknrétně přidělil/a karty, to jsou karty, které daná osoba přidělila jiným osobám. Dále přidělené karty, které této osobě byly přiděleny. Následuje záložka zastupuje, kde je seznam jednotek, které tato osoba zastupuje a jmenovaní zástupci, kde je seznam osob, které tato osoba pověřila správou organizačních jednotek. Poslední záložka manažer obsahuje seznam organizačních jednotek, kde je daná osoba vedena jako manažer.

Seznam karet v prvních dvou záložkách má atributy shodné jako v záložkách u detailu karty, pouze je vypuštěn držitel respektive osoba, která kartu přidělila. Další dvě záložky, týkající se zastoupení, mají níže uvedené atributy, pouze v záložce jmenovaní zástupci je navíc atribut zástupce:

- stav,
- organizační jednotka,
- začátek platnosti,
- konce platnosti,
- v akcích je pouze odkaz na detail zastoupení.

Seznam organizačních jednotek v záložce manažer se svými atributy shoduje s přehledem organizačních jednotek, viz následující část s jedním rozdílem. Ze zřejmých důvodů je vynechán atribut manažeři jednotky.

3.5.1.3 OrganizationalUnit

Entita OrganizationalUnit reprezentuje organizační jednotku. V přehledu jednotek je tlačítko pro vytvoření nové jednotky. Tabulka jednotek obsahuje následující atributy:

- UID,
- název,
- nadřazená jednotka,
- manažeři jednotky,
- v akcích je pouze odkaz na detail karty.

V detailu jednotky se nachází atributy UID, název a nadřazená jednotka. Podle režimu a role uživatele jsou dostupné akce smazání a editace jednotky, dále úprava manažerů jednotky a zjištění inventáře jednotky. Detail jednotky obsahuje tři záložky souvisejících entit. Koknrétně manažeři, zástupci a karty. Záložka manažeři obsahuje stejné atributy jako přehled osob, viz výše, navíc je přidána akce odebrat manažera. Záložka karty obsahuje pouze číslo karty a odkaz na detail karty. V záložce zástupci pak najdeme tyto atributy:

- stav,
- zástupce,
- začátek platnosti,
- konce platnosti,
- v akcích je zrušení zastoupení a detail zastoupení.

3.5.1.4 Ownership

Entita Ownership reprezentuje přiřazení karty osobě. Přehled je realizován vždy v detailu souvisejících entit pomocí výše zmíněných záložek. V detailu přidělení se pak nachází dle oprávnění uživatele tlačítko pro úpravu přidělení a odebrání karty a také štítek, který značí platnost přidělení. Dále jsou uvedeny následující atributy přidělení:

- začátek platnosti,
- konce platnosti,

- držitel,
- karta,
- přiděleno (datum přidělení),
- přidělil.

3.5.1.5 Deputy

Entita Deputy reprezentuje zastoupení organizační jednotky další osobou. Přehled je realizován vždy v detailu souvisejících entit pomocí výše zmíněných záložek. V detailu zastoupení se nachází dle oprávnění uživatele tlačítko pro zrušení zastoupení a také štítek, který značí platnost zastoupení. Uvedeny jsou také následující atributy zastoupení:

- začátek platnosti,
- konce platnosti,
- zástupce,
- organizační jednotka,
- vytvořeno (datum jmenování),
- vytvořil/a.

3.5.2 RESTful API

V této části popisuji požadavky kladené na aplikaci Card-Manager ve smyslu poskytnutí RESTful API

V lokálním režimu provozu bude aplikace Card-Manager poskytovat RESTful API pro převedení čísla čipu na login uživatele. Bližší specifikací tohoto API se bude zabývat implementační návrh. Všechny požadavky budou ověřeny pomocí OAuth2.0.

Kapitola 4

Implementační fáze

V této kapitole představuji zvolené technologie pro implementaci aplikace Card-Manager. Dále zde popisuji architekturu aplikace a také popisuji některé implementačně zajímavé detaily.

4.1 Zvolené technologie

V této části představuji technologie využité při implementaci aplikace. Technologie byly zvolené v souladu s Pravidly. Při výběru technologií byla také snaha držet technologie konzistetní s ostatními projekty.

4.1.1 Symfony

Symfony je sada znovu použitelných komponent, které dohromady tvoří framework[12] pro programovací jazyk PHP[13]. Obecně má využití nástroje jako je framework Symfony několik výhod. Mezi hlavní výhody patří pomoc se členěním kódu a architekturou aplikace. Dále pak samotný fakt, že framework implementuje běžné funkce společné pro všechny aplikace a některé koncepty softwarového inženýrství. A v neposlední řadě fakt, že v přechozí větě zmíněná implementace je otestována.

Velmi důležitá vlastnost Symfony je dependency injection kontejner, který se sestaví na začátku běhu aplikace. Dependency injection je obecně technika pro řešení závislostí v objektivně orientovaném programování. V DI kontejneru se pak nachází veškeré služby včetně jejich závislostí. Tyto služby se registrují v konfiguračních souborech aplikace.

4.1.2 Rozšíření pro Symfony

Protože Symfony framework je poměrně rozšířený, existuje pro něj spousta rozšíření v podobě bundles třetích stran, která usnadní implementaci někte-

rých částí, jako například RESTful API. V této části pak popisuji rozšíření, která byla použita při vývoji aplikace Card-Manager.

4.1.2.1 FOSRestBundle

FOSRestBundle[14] je vyvíjen skupinou vývojářů, která si říká FriendsOfSymfony. Unsadňuje implementaci RESTful API. Zejména napomáhá s udržováním konzistence URI zdrojů a použitím správných HTTP metod. Dále má pak speciální vrstvu View, která využívá dostupné možnosti serializace objektů a usnadňuje tak práci vývojářům aplikace.

4.1.2.2 FOSJsRoutingBundle

FOSJsRoutingBundle[15] je vyvíjen stejnou skupinou vývojářů jako výše zmíněný FOSRestBundle. Tento bundle umožňuje částečné využití komponenty Symfony Routing v programovacím jazyce JavaScript. Také umožňuje využití zaregistrovaných URI aplikace v jazyce JavaScript.

4.1.2.3 KnpMenuBundle

KnpMenuBundle[16] integruje PHP knihovnu KnpMenu, která usnadňuje vytváření veškerých druhů navigací v aplikaci. Jak bundle, tak samotná knihovna je od uskupení vývojářů KNP Labs.

4.1.2.4 JsTranslationBundle

JsTranslationBundle, někdy také nazývaný BazingaJsTranslationBundle[17], má podobnou funkci jako FOSJsRoutingBundle. Poskytuje část funkcí komponenty Symfony Translation a překlady aplikace pro použití v jazyce JavaScript.

4.1.3 Twig

Twig[18] je šablonovací systém od stejné skupiny jako Symfony framework. Tudíž je součástí standardní distribuce Symfony a pro zachování konzistence s ostatními projekty v oddělení ICT byl v tomto projektu také využit. Mezi hlavní přednosti šablonovacího systému patří bezpečnost, protože Twig umožňuje vypsat vstup uživatele v uzavřeném prostředí, takže nedojde k vykonání potenciálně škodlivého kódu. Využití šablonovacího systému také podporuje a usnadňuje oddělení prezentační vrstvy od zbytku aplikace.

4.1.4 Doctrine

Doctrine je projekt, který sdružuje několik PHP knihoven primárně pro práci s relačními databázemi[19]. Podporuje několik databázových strojů a díky

abstrakci se s nimi na aplikační úrovni pracuje stejně. Mezi hlavní části, které se využívají v aplikaci Card-Manager, patří Doctrine ORM a Doctrine DBAL.

Doctrine ORM je PHP knihovna pro objektově relační mapování. Na základě metadat u entitních tříd vytvoří pro tato data databázové schéma. Také poskytuje rozhraní pro manipulaci s daty. Doctrine DBAL je abstraktní databázová vrstva, přes kterou se komunikuje s konkrétním databázovým strojem, který byl využit.

4.1.5 Guzzle

Guzzle je HTTP klient pro PHP, který velmi unsnadňuje posílání HTTP požadavků[20]. Využití v aplikaci Card-Manager najde zejména při komunikaci s IdM.

4.2 Architektura aplikace

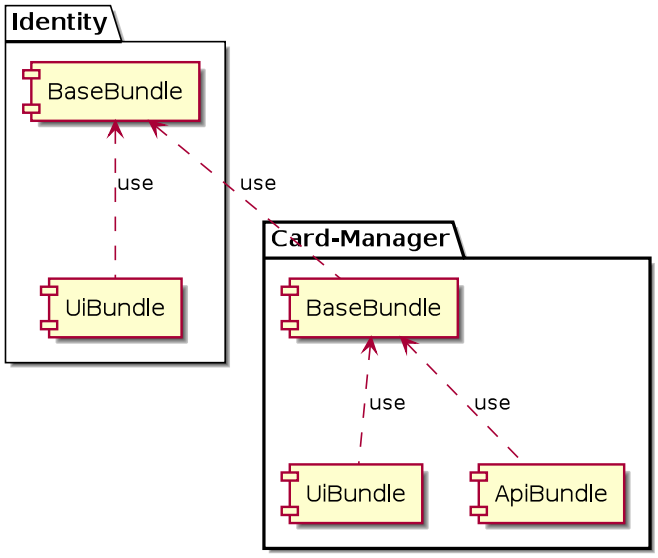
V této kapitole popisuji architekturu aplikace Card-Manager, okrajově také architekturu aplikace Identity, která je nezbytná pro provoz aplikace Card-Manager bez ohledu na režim.

Během analytické fáze byla identifikována potřeba sdílet část dat a příslušné operace s aplikací ACS-Manager. Konkrétně jde o část pro práci s osobami a rolemi. Z tohoto důvodu bylo rozhodnuto, že každá z aplikací Identity, Card-Manager a ACS-Manager bude balíkem Symfony bundles. Závislost a struktura jednotlivých částí je zachycena diagramem 4.1.

Jednotlivé bundles aplikace Card-Manager a jejich struktura bude popsána v následujích částech.

4.2.1 BaseBundle

Obecně se dá říci, že BaseBundle obsahuje implementaci datového modelu, včetně Repository tříd odpovědných za získání dat z úložiště. Dále obsahuje implementaci business logiky a bezpečnotsních a konfiguračních mechanismů. Celkově tedy BaseBundle obsahuje implementaci business procesů aplikace Card-Manager.



Obrázek 4.1: Architektura aplikace Card-Manager

```
BaseBundle/
├── DataFixtures/
│   └── ORM/ ..... třídy s ukázkovými daty pro Doctrine ORM
├── DependencyInjection/ ..... třídy ovlivňující konfiguraci aplikace
│   └── Compiler/ ..... třídy ovlivňující sestavení kontejneru služeb
├── Entity/ ..... třídy datového modelu pro Doctrine ORM
├── Event/ ..... třídy reprezentující události nastalé v aplikaci
├── EventSubsciber/ ..... třídy zodpovědné za obsluhu událostí
├── Exception/ ..... vlastní výjimky pro signalizaci chyby
├── Repository/ ..... třídy pro manipulaci s entitami v Doctrine ORM
├── Resources/
│   ├── config/ ..... konfigurační soubory ve formátu YAML
│   └── doc/ ..... části technické dokumentace
├── Security/ ..... třídy za autorizaci uživatelů
├── Service/ ..... třídy implementující business logiku aplikace
│   ├── Idm/ ..... třídy implementující komunikaci s IdM
│   └── Serializer/ .... třídy implementující serializaci a deserializaci dat
├── Tests/ ..... třídy s jednotkovými a integračními testy
├── Validator/ ..... třídy pro validaci dat
└── CvutFitIctCardManagerBaseBundle.php .. třída pro registraci bundle
```

4.2.2 UiBundle

UiBundle obecně poskytuje uživatelské rozhraní pro práci s aplikací. Protože je Card-Manager webová aplikace, obsahuje UiBundle převážně HTML šablony pro šablonovací systém Twig. Dále pak Controller třídy, které jsou zodpovědné za zpracování uživatelského vstupu. V neposlední řadě se v tomto bundle nachází i soubory s překlady.

```
UiBundle/
├── Controller/ ..... třídy obsluhující vstup uživatele
├── Form/ ..... třídy s definicí formulářů
├── Menu/ ..... třídy s definicí menu
├── Resources/
│   ├── config/ ..... konfigurační soubory ve formátu YAML
│   ├── translations/ ..... soubory s překlady ve formátu YAML
│   └── views/ ..... HTML šablony systému Twig
├── Twig/ ..... rozšíření do šablonovacího systému Twig
└── CvutFitIctCardManagerUiBundle.php .... třída pro registraci bundle
```

4.2.3 ApiBundle

ApiBundle je pak obecně svým zaměřením podobný UiBundle, ale jeho rozhraní je RESTful API, které dává daná aplikace k dispozici externím systémům. Využívá se zde výše zmíněný FOSRestBundle.

```
ApiBundle/
├── Controller/ .... třídy poskytující reprezentace zdroje v RESTful API
└── CvutFitIctCardManagerApiBundle.php ... třída pro registraci bundle
```

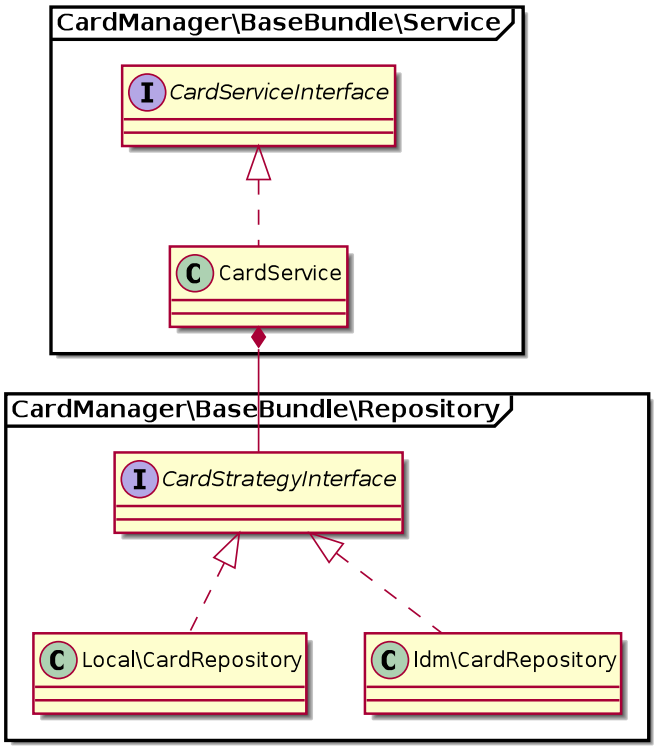
4.3 Implementace dílčích částí

V této podkapitole popisuji implementační detaily, které jsou nad rámec běžných CRUD aplikací nebo vlastnosti specifické pro aplikaci Card-Manager. Jedná se zejména o záležitosti v souvislosti s IdM režimem provozu.

4.3.1 Režimy provozu aplikace

V návrhové fázi jsem stanovil, že aplikace se při nasazení do provozu parametrem zkonfiguruje a za pomocí návrhového vzoru Strategy se použije příslušná implementace. Toto využití implementace je znázorněno diagramem 4.2, kde jsem si jako příklad zvolil entitu Card. Pro všechny ostatní entity je implemetace řešena stejně.

Výše zmíněný parametr při nasazení se jmenuje `identity.strategy` a může nabývat hodnot `local` nebo `idm`. Jednotlivé služby využívající implementaci strategie se pak při registraci v konfiguračním souboru služeb označí speciální značkou.



Obrázek 4.2: Diagram strategie režimu provozu

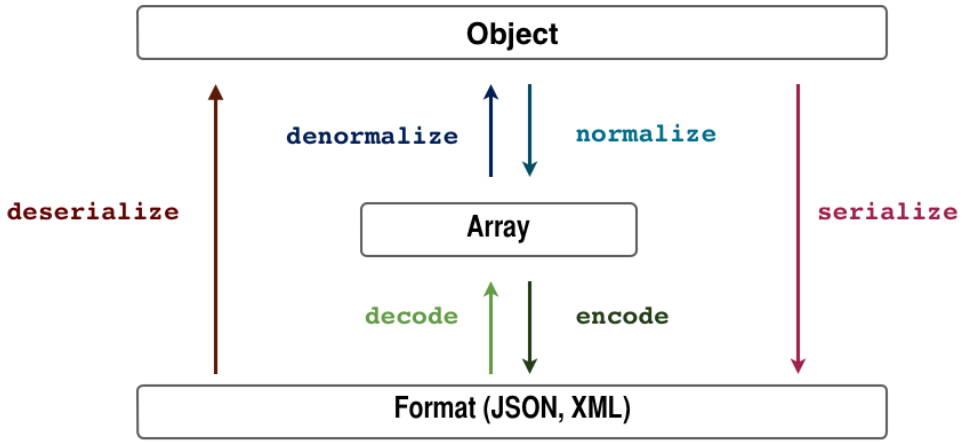
```
services:
    cvut_fit_ict_card_manager_base.service.card:
        class: Cvut\Fit\Ict\CardManager\BaseBundle\Service\
            CardService
        tags:
            - { name: cvut_fit_ict_card_manager_base.
                strategy_context }
```

Službám takto označeným se pak ve speciální třídě `StrategyContextPass`, která ovlivňuje sestavení DI kontejneru, injektuje podle hodnoty parametru `identity.strategy` příslušná třída Repository.

4.3.2 Transformace dat

Již v návrhové fázi jsem zmínil, že bude potřeba transformovat data z IdM, které je poskytuje ve formátu JSON. K tomu jsem využil komponentu Symfony Serializer. Tato komponenta implementuje transformaci dat z formátů jako JSON nebo XML na objekty a obráceně[21]. Celý proces serializace a deserializace je znázorněn v diagramu 4.3.

Komponenta Serializer má obecnou třídu pro normalizaci a denormalizaci



Obrázek 4.3: Proces serializace a deserializace dat

[22]

objektů `ObjectNormalizer`. Tento obecný způsob však není dostačující pro aplikaci Card-Manager. Z tohoto důvodu je nutné implementovat vlastní normalizéry pro každou entitu. Normalizéry se podobně jako služby z předchozí částí speciálně označí a díky tomu budou injektovány do služby `serializer`. Tato služba se pak používá v třídách `Idm\Repository` pro transformaci dat.

4.3.3 Události

V aplikaci Card-Manager je několik systémových událostí, na které je potřeba reagovat. Jedná se o smazání karty, smazání organizační jednotky a smazání osoby. Prvni dvě události jsou vyvolané aplikací Card-Manager, třetí událost je pak vyvolána aplikací Identity a v aplikaci Card-Manager je potřebné ji obsloužit. Nicméně toto platí jen pro lokální režim provozu, v IdM režimu provozu je konzistence dat v kompetenci IdM.

Při smazání entity Card je potřeba smazat navázané přidělení karty, entity Ownership. V případě smazání entity OrganizationalUnit je třeba smazat entity zastoupení OJ pro tuto jednotku. Dále je pak potřeba odebrat karty z právě mazané organizační jednotky a uložit tuto informaci do auditních dat karty.

Při mazaní entity Person je potřeba smazat entity přiřazení Ownership, kde je smazaná osoba držitelem karty. Dále se musí smazat entity zastoupení Deputy, kde je tato osoba zástupcem. Také je potřeba tuto osobu odebrat z entit Ownership a Deputy, kde je vedena jako osoba zodpovědná za vytvoření a tuto informaci uložit do auditních dat příslušné entity.

Pro implementaci vyvolání událostí a jejich obsloužení se využívá komponenta Symfony EventDispatcher. Tato komponenta má tři základní části.

Třídy, které reprezentují události, dále službu `event_dispatcher`, která slouží k vyvolání událostí. Třetí částí jsou tzv. Listener nebo Subscriber třídy, které se zaregistrují v konfiguraci služeb a následně obsluhují události, pro které jsou určené[23].

V aplikaci Card-Manager jsou pak výše zmínené události reprezentovány třídami `CardDeletedEvent` a `OrganizationalUnitDeletedEvent`. Tyto události obsluhují příslušné Subscriber třídy, které se jmenují `CardSubscriber`, resp. `OrganizationalUnitSubscriber` a obsahují metody `onCardDelete()`, resp. `onOrganizationalUnitDelete()`. Aplikace Identity pak má speciální událost `PersonDeletedEvent`, na kterou v aplikaci Card-Manager reaguje třída `AbstractIdentitySubscriber` metodou `onPersonDelete()`.

4.3.4 Práce s uživateli

Pro autentizaci a autorizaci uživatelů se využívá komponeneta Symfony Security, která poskytuje kompletní bezpečnostní mechanismus[24]. Základní autorizace probíhá na úrovni aplikačních rolí, které vycházejí z části aktéři v návrhové fázi.

Zástupce manažera organizační jednotky je reprezentován aplikační rolí `ROLE_CM_DEPUTY`, dále samotný manažer organizační jednotky je reprezentován rolí `ROLE_CM_MANAGER`. Administrátorovi aplikace pak odpovídá role `ROLE_CM_ADMIN`. Tyto role je nutné při nasazení aplikace v lokálním režimu provozu inicializovat v databázi.

K tomu byla využita komponenta Symfony Console, která umožňuje vytvářet konzolové příkazy.[25] Konkrétně příkaz `card-manager:init-roles` realizovaný třídou `CardManagerInitRolesCommand` vytvoří v databázi tři výše zmíněné role, pokud ještě neexistují.

Stejná technologie byla využita i pro stanovení administrátora v lokálním režimu. To se dá učinit příkazem `card-manager:init-admin [<login>]`. Dále byly touto metodou vytvořeny také příkazy pro odebrání neplatných rolí zástupce organizační jednotky, `card-manager:remove-deputy-role` a také manažera jednotky, `card-manager:remove-manager-role`. Tyto příkazy budou spouštěny automaticky v pravidelných intervalech démonem Cron.

4.3.5 Controller

Controller třídy jsou zodpovědné za zpracování vstupu uživatele, to je řešeno tzv. akcemi. Akce jsou speciální metody, které se dle konfigurace volají při HTTP požadavku na daný zdroj. Třídy jsou rozděleny podle entit, například `CardController`, ale v rámci CRUD operací jsem stanovil jednotný způsob implementace napříč entitami. Jde o konvence pojmenovaní některých akcí, jejich URI a HTTP metody, kterou akceptují. Tyto konvence jsou popsané v tabulce 4.1.

Akce	URI	HTTP metoda	popis
`indexAction`	/	GET	zobrazí přehled
`showAction`	/{entityId}	GET	zobrazí detail
`newAction`	/new	GET	zobrazí formulář pro vytvoření
`createAction`	/new	POST	zpracuje formulář pro vytvoření
`editAction`	/{entityId}/edit	GET	zobrazí formulář pro úpravení
`updateAction`	/{entityId}/edit	POST	zpracuje formulář pro úpravení
`deleteAction`	/{entityId}/delete	GET	smazání

Tabulka 4.1: CRUD akce v třídách Controller

Každý Controller obsahuje prefix URI zaznamenaný formou anotace. Tento prefix slouží k tomu, aby nedošlo ke kolizi, jak by se z tabulky mohlo zdát. Dále `{entityId}` je parametr, který se nahradí identifikátorem příslušné entity.

```
/**
 * Class CardController
 *
 * @package Cvut\Fit\Ict\CardManager\UiBundle\Controller
 *
 * @Route("/cards")
 */
class CardController extends Controller
{
...
}
```

Kapitola 5

Vyhodnocení

V této kapitole provádím vyhodnocení vyvinuté aplikace Card-Manager. Nejprve v části testování popisuji způsob a výsledek testování aplikace. Potom v podkapitole Další rozvoj popisuji budoucí práce, které budou provedeny na aplikaci.

5.1 Testování

V této podkapitole popisuji proces testování aplikace Card-Manager a jeho výsledek. Při testování aplikace bylo využito konceptu automatického testování, důraz byl kladen zejména na jednotkové testy. Dále byla provedena statická analýza kódu.

5.1.1 Jednotkové a integrační testy

V této části představím, které části aplikace Card-Manager byly podrobeny jednotkovým a integračním testům, strukturu těchto testů a výsledek testování.

Jednotkové testy umožňují otestovat samostatnou třídu. Testovány byly entitní a Repository třídy. Částečně byly otestovány také některé třídy business logiky ze složky `Service`. Pro testování byl použit nástroj PHPUnit[26].

Testy se nachází separátně v příslušných Symfony bundles ve složce `Tests`. Tato složka svou hierarchií pak kopíruje příslušný bundle. Pojmenování testů se drží konvence, kdy se třída testu pojmenuje jako testovaná třída s příponou „Test“, například `CardRepository` a `CardRepositoryTest`. Metoda testu se pak pojmenuje jako testovaná metoda s předponou „test“, například `findById` a `testFindById`.

Testují se hlavně návratové hodnoty volaných metod, pokud je to možné testují se také návratové typy. Dále se testuje chování volané metody s chybným vstupem. Zejména, jestli je tento chybný vstup detekován a je vyhozena správná výjimka. Výstup nástroje PHPUnit se nachází níže.

```
$ vendor/bin/phpunit -c phpunit.no-coverage.xml
PHPUnit 5.7.19 by Sebastian Bergmann and contributors.

.........................SSSSS.................................
     65 / 75 ( 86%)
........S.
                                                                 75 /
    75 (100%)

Time: 674 ms, Memory: 20.00MB

OK, but incomplete, skipped, or risky tests!
Tests: 75, Assertions: 180, Skipped: 6.
```

Testy které jsou označené písmenem `S`, jsou integrační testy, které byly vynechány. Důvody vynechání těchto testů budou vysvětleny v podkapitole Další rozvoj.

5.1.2 Statická analýza kódu

V této části popisuji proč je dobré provádět statickou analýzu kódu, jaký nástroj byl použit při analyzování aplikace Card-Manager a v závěru se nachází výsledky analýzy.

Statická analýza hledá chyby v kódu, aniž by bylo nutné daný kód spustit. Přibližuje tak interpretovaný jazyk, jako je PHP, kompilovaným jazykům. Logicky tak předchází chybám při spuštění aplikace a napomáhá s udržováním čistého kódu.

Pro statickou analýzu aplikace Card-Manager byl použit nástroj PHPStan[27], který nabízí pět úrovní analýzy. Výstup analýzy naleznete níže.

```
$ vendor/bin/phpstan analyse src/Cvut/Fit/Ict/CardManager/
 107/107 [----------------------------] 100%


 [OK] No errors


 ! [NOTE] PHPStan is performing only the most basic checks. You
     can pass a higher rule level through the --level option
 !        (the default and current level is 0) to analyse code
     more thoroughly.
```

5.2 Další rozvoj

V této podkapitole popisuji další rozvoj aplikace Card-Manager. Další rozvoj souvisí zejména s IdM režimem provozu, implementací nových procesů a rozvojem uživatelského rozhraní.

V průběhu implementační fáze se ukázalo, že problém komunikace přes IdM API je složitější, než se předpokládalo v analytické fázi a byly tak kladeny

větší nároky na IdM. Z tohoto důvodu se nepodařilo aplikaci připravit pro IdM režim provozu. Také API IdM není zcela připraveno. V době dokončení této práce poskytuje zdroje pro získání potřebných dat, chybí však některé možnosti filtrace.

Cílem je tedy připravit aplikaci pro IdM režim, provést integrační testy vůči IdM a nasadit aplikaci do ostrého provozu v tomto režimu pro využití na Fakultě informačních technologií ČVUT v Praze.

Další prostor pro rozvoj je implementace nového procesu v souvislosti s přidělováním karty osobě. Při průchodu aplikací se zjistilo, že v zadání byla opomenuta záležitost fyzického vydání karty osobě, které byla přidělena. Aplikace Card-Manager eviduje pouze časový interval platnosti přidělení karty a držitele karty. Počátek platnosti a konec platnosti se však nemusí shodovat s fyzickým vydáním a vrácením karty a nelze to tedy z těchto údajů vyčíst. Zástupce organizační jednotky pak bohužel neví, zda-li mu byla karta vrácena či nikoliv. Na tento nedostatek se bohužel nepřišlo v analytické ani návrhové fázi.

Pro rešení problému z předchozího odstavce již vznikl implementační návrh. Entita Card se rozšíří o příznak, jestli je karta dostupná nebo nedostupná a datum poslední změny. Tyto dva nové atributy se budou udržovat pomocí událostí z komponenty Symfony EventDispatcher podobně, jako se modifikují údaje u entit Ownership a Deputy. Více implementačních detailů popisuji v části 4.3.3 Události.

Některé části návrhu uživatelského rozhraní byly zanedbány, viz příslušná kapitola. To mělo za následek, že při průchodu aplikací na schůzce, které se účastnili Ing. Tomáš Kadlec, Ing. Jiří Špaček, Ing. Martin Bílý, Michal Pěch a Martin Vondrák, bylo vzneseno několik připomínek k uživatelskému rozhraní.

Mezi nejzásadnější patří připomínka k UC3 – Načtení karty. V novém provedení bude přeskočen první krok a vstup pro načtení karty bude ihned viditelný. Po načtení karty bude uživatel přesměrován na detail karty, kde budou nově dostupné operace, kterým musí přecházet načtení karty. V případě, že karta nebude v aplikaci nalezena, bude v lokálním režimu provozu uživatel přesměrován na formulář pro registraci karty s notifikací, že karta nebyla nalezena a může ji nyní vytvořit.

V neposlední řadě by bylo užitečné aplikaci podrobit výkonostním testům a to hlavně v IdM režimu provozu, až na něj aplikace bude připravena. Následně provést případnou optimalizaci.

Realizace výše zmíněného rozvoje byla se zadavatelem předběžně naplánována na období července až září 2017.

Závěr

Cílem práce byla analýza, návrh a implementace, včetně automatizovaných testů, aplikace pro přidělování karet osobám. Tuto aplikaci musí být možné provozovat ve dvou režimech. Jeden režim s lokálním úložištěm a druhý režim, kde jako úložiště slouží externí systém, IdM Fakulty informačních technologií ČVUT v Praze.

V práci jsem provedl analýzu současného stavu na FIT ČVUT v Praze a na základě pravidelných rozhovorů se zadavatelem Ing. Tomášem Kadlecem a správcem fakultního IdM Ing. Jiřím Špačkem vznikl seznam hypotéz. Na základě těchto hypotéz byly stanoveny požadavky kladené na aplikaci.

Na základě poznatků zjištěných v analytické fázi jsem navrhl aplikaci, kterou je možné provozovat v obou výše zmíněných režimech. Aplikace primárně umožňuje evidovat čipové karty organizační jednotky a přidělovat osobám. Dále aplikace umožňuje uživateli, který má oprávnění karty takto spravovat, dočasně předat svoje oprávnění další osobě.

V průběhu implementace se ukázalo, že problém komunikace přes IdM API je složitější než se předpokládalo v analytické části. Byly tak kladeny větší nároky na IdM. Z tohoto důvodu aplikace plně nepodporuje provoz v IdM režimu a také API IdM není zcela připraveno. V době dokončení této práce poskytuje zdroje pro získání potřebných dat, chybí však některé možnosti filtrace.

Aplikace je spíše evidenčního charakteru, takže nezahrnuje některé business procesy, například vydání karty osobě. V budoucnosti by tedy bylo možné tyto procesy analyzovat a následně implementovat do aplikace. Práce se také příliš nezabývá návrhem uživatelského rozhraní, proto by bylo možné současné rozhraní otestovat a následně navrhnout a realizovat změny.

Literatura

[1] Kadlec, T.: acs-controller. [software], říjen 2016, [cit. 2017-05-06]. Dostupné z: `https://ict.fit.cvut.cz/gitlab/mar/acs_controller`

[2] Fakulta informačních technologií, České vysoké učení technické v Praze: *Pravidla a zásady projektů FIT*. duben 2017, [cit. 2017-04-11]. Dostupné z: `https://docs.google.com/document/d/1umkLCuvYY1EYMat8jLnYfUj5WyccC3s-30thPvePot4`

[3] ČVUT v Praze – Vydavatelství průkazů. [online], prosinec 2016, [cit. 2016-12-10]. Dostupné z: `http://intranet.cvut.cz/informace-pro-zamestnance/prukazy`

[4] ČVUT v Praze – Průkaz typu STUDENT. [online], červenec 2013, [cit. 2016-12-10]. Dostupné z: `http://intranet.cvut.cz/informace-pro-studenty/prukazy/student`

[5] ČVUT v Praze – Průkaz typu OSOBNÍ. [online], leden 2015, [cit. 2016-12-10]. Dostupné z: `http://intranet.cvut.cz/informace-pro-zamestnance/prukazy/osobni`

[6] ČVUT v Praze – Průkaz typu PŘENOSNÝ. [online], září 2012, [cit. 2016-12-10]. Dostupné z: `http://intranet.cvut.cz/informace-pro-zamestnance/prukazy/prenosny`

[7] České vysoké učení technické v Praze: *Podmínky pro vydávání a používání průkazů typu přenosný ČVUT*. červen 2013, [cit. 2016-12-10]. Dostupné z: `http://intranet.cvut.cz/informace-pro-zamestnance/prukazy/resolveuid/21694ec07e3f43a3c6f729831f1868d3`

[8] ProgTest. [software], [cit. 2017-04-15]. Dostupné z: `https://progtest.fit.cvut.cz`

[9] Sijbrandij, S.: GitLab Flow | GitLab. [online], září 2014, [cit. 2017-05-06]. Dostupné z: https://about.gitlab.com/2014/09/29/gitlab-flow/

[10] tutorialspoint.com: Design Patterns Strategy Pattern. [online], [cit. 2017-05-10]. Dostupné z: https://www.tutorialspoint.com/design_pattern/strategy_pattern.htm

[11] Vixie, P.; Mašláňová, M.: cron(8): daemon to execute scheduled commands - Linux man page. [online], [cit. 2017-05-10]. Dostupné z: https://linux.die.net/man/8/cron

[12] SensioLabs: What is Symfony. [online], [cit. 2017-05-12]. Dostupné z: http://symfony.com/what-is-symfony

[13] PHP Group: PHP. [software], 2001-2017, [cit. 2017-05-12]. Dostupné z: http://php.net/

[14] SensioLabs: Getting Started With FOSRestBundle (The Symfony Bundles Documentation). [online], [cit. 2017-05-13]. Dostupné z: http://symfony.com/doc/current/bundles/FOSRestBundle/index.html

[15] SensioLabs: FOSJsRoutingBundle (The Symfony Bundles Documentation). [online], [cit. 2017-05-13]. Dostupné z: http://symfony.com/doc/current/bundles/FOSJsRoutingBundle/index.html

[16] SensioLabs: Using KnpMenuBundle (The Symfony Bundles Documentation). [online], [cit. 2017-05-13]. Dostupné z: http://symfony.com/doc/current/bundles/KnpMenuBundle/index.html

[17] Durand, W.: BazingaJsTranslationBundle/index.md at master · willdurand/BazingaJsTranslationBundle. [online], [cit. 2017-05-13]. Dostupné z: https://github.com/willdurand/BazingaJsTranslationBundle/blob/master/Resources/doc/index.md

[18] SensioLabs: Home - Twig - The flexible, fast, and secure PHP template engine. [online], [cit. 2017-05-12]. Dostupné z: https://twig.sensiolabs.org/

[19] About – Doctrine Project. [online], [cit. 2017-05-13]. Dostupné z: http://www.doctrine-project.org/about.html

[20] Guzzle, PHP HTTP client – Guzzle Documentation. [online], [cit. 2017-05-13]. Dostupné z: http://docs.guzzlephp.org/en/latest/

[21] SensioLabs: The Serializer Component (The Symfony Components). [online], [cit. 2017-05-14]. Dostupné z: http://symfony.com/doc/current/components/serializer.html

[22] SensioLabs: Serializer Workflow. [online], [cit. 2017-05-14]. Dostupné z: http://symfony.com/doc/current/_images/serializer_workflow.png

[23] SensioLabs: The EventDispatcher Component (The Symfony Components). [online], [cit. 2017-05-14]. Dostupné z: http://symfony.com/doc/current/components/event_dispatcher.html

[24] SensioLabs: The Security Component (The Symfony Components). [online], [cit. 2017-05-14]. Dostupné z: http://symfony.com/doc/current/components/security.html

[25] SensioLabs: The Console Component (The Symfony Components). [online], [cit. 2017-05-14]. Dostupné z: http://symfony.com/doc/current/components/console.html

[26] Bergmann, S.: PHPUnit. [software], 2001-2017, [cit. 2017-05-16]. Dostupné z: https://phpunit.de/

[27] Mirtes, O.: PHPStan. [software], 2016-2017, [cit. 2017-05-16]. Dostupné z: https://github.com/phpstan/phpstan

[28] Twitter: Bower. [software], 2012-2017, [cit. 2017-05-16]. Dostupné z: https://bower.io/

[29] Adermann, N.; Boggiano, J.: Composer. [software], 2012-2017, [cit. 2017-05-16]. Dostupné z: https://getcomposer.org/

Příloha A

Pravidla a zásady projektů FIT

FIT ČVUT, 2017-02-21

Pravidla a zásady projektů FIT

Tento dokument popisuje preferovaný způsob řešení projektů FIT. Pokud některá část tohoto dokumentu není vůči konkrétnímu projektu efektivní nebo na ní není dostatek prostředků, je jí snížena úroveň nezbytnosti[1] nebo je od ní zcela odstoupeno. O takovém rozhodnutí musí existovat záznam.

Tento dokument je nedílnou součástí projektové dokumentace jako příloha ke kapitole Požadavky na kvalitu. Veškeré oblasti tohoto dokumentu, které uvedená kapitola nezmíní, se předpokládají a požadují. Výchozí úrovní nezbytnosti požadavků (není-li uvedeno jinak) je [MUST].

Obsah dokumentu

Dokumenty

Ke každému projektu vzniká sada samostatných dokumentů v čele s harmonogramem. Dokumenty jsou (až na výjimky) psané česky. Výchozím dokumentem je Harmonogram, který podléhá schvalování. Není-li uvedeno jinak, o projektu rozhoduje vedení fakulty (grémium děkana).

[1] Key words for use in RFCs to Indicate Requirement Levels

Harmonogram

Stěžejním rozcestníkem projektu je iterativní harmonogram. V každém běhu se předpokládá realizace jen takových funkcí a vlastností aplikace, které náleží do daného běhu. Každý běh (iterace) obsahuje všechny uvedené fáze harmonogramu. Pro každou fázi je stanoven termín předpokládaného dokončení. Každá fáze podléhá schvalování vedením písemnou formou. Obsah harmonogramu se s postupem času upřesňuje.

1. **Analytická fáze**
    - Průzkum existujících řešení (dále jen SOTA[2]).
    - Vytvoření hypotéz jako kvalitativním průzkum, typicky formou rozhovorů.
        i. Uživatelské skupiny
        ii. Potřeby uživatelů
    - Ověření hypotéz jako kvantitativní průzkum, typicky dotazníkem v souladu s explicitně zmíněnou metodikou[3].
    - Sestavení požadavků na kvality (vycházející z tohoto dokumentu).
    - Sestavení požadavků na funkce (prioritní seznam rozdělený dle úrovní nezbytnosti).
    - Posouzení SOTA vůči požadavkům na funkce.
2. **Návrhová fáze**
    - Procesy (back-end, front-end)
    - Prototypování (paralelní).
    - Diagramy (procesní, aktivit).
    - Scénáře průchodu, user-stories, případové studie
        i. pro všechny vznikající funkce
        ii. testování (např. formou storyboarding, inspekce).
    - Hi-fi prototypy (grafický návrh, …).
3. **Implementační fáze**
    - Koncepty řešení dílčích požadavků (funkční / nefunkční).
    - Technická specifikace (API, parametry, manuál) jako příloha k projektové dokumentaci nebo přímo u projektu v repozitáři.
    - Testování (inspekce, heuristika).
4. **Vyhodnocení**
    - Testování, logování.
    - Vyhodnocení (feedback, statistiky, logy).

Ostatní dokumenty projektu

V rámci vývoje vznikají další typy dokumentů s níže uvedenými náležitostmi pro různé skupiny čtenářů. Obsah každého dokumentu je cílený na příslušnou skupinu uživatelů a zohledňuje jejich schopnosti a možnosti.

- **Projektová dokumentace**
    - je množina samostatných dokumentů vznikajících pro jednotlivé iterace agilního vývoje aplikace[4],
    - má strukturu podle fází harmonogramu,
    - odkazuje na související legislativní úpravu problémové domény aplikace,
    - je určená pro zadavatele, návrháře a vývojáře projektu a případně pro uživatele.

[2] State of the Art
[3] 6 kroků, jak vytvořit dotazník
[4] Agile software development

- **Instalační a provozní příručka**
  - obsahuje kompletní postup pro sestavení (build) a nasazení (deployment) aplikace a nových verzí,
  - popisuje dostupná prostředí (staging/produkční verze) v návaznosti na kap. Údržba a rozvoj,
  - popisuje provozuschopnost v případě nedostupnosti souvisejících služeb,
  - popisuje proces obnovení provozu v případě výpadku.
- **Uživatelská dokumentace**
  - je průběžně udržovaný samostatný dokument,
  - obsahuje informaci, k čemu a komu aplikace souží,
  - je určená koncovým uživatelům frontendové aplikace[5], resp. aplikačního rozhraní (RESTful API)[6],
  - je dostupná z webu FIT (stačí odkazem)[7],
  - zahrnuje CHANGELOG (viz kap. Verzování).
- **Dokumentace vnitřního API**
  - je sada dokumentů generovaná z kódu průběžně udržovaná společně s kódem aplikace,
  - je psaná anglicky v příslušné standardizované syntaxi[8], přičemž dokumentace veřejných entit zahrnuje minimálně:
    - souhrnný popis dokumentované entity (funkce, třídy, metody, proměnné, …),
    - souhrnný popis parametrů (funkce/metody) nebo typových proměnných (generické typy),
    - popis vyhazovaných výjimek (které výjimky a kdy vznikají),
    - popis návratové hodnoty (a její význam).

Architektura a integrace do infrastruktury

Projekt je webovou aplikací, která efektivně využívá existující technologie a služby FIT. Aplikace je členěná na nezávislé části, které je možné vyměnit a je provozuschopná i v případě výpadků souvisejících služeb.

- Projekt je webovou aplikací s
  - uživatelským rozhraním (UI) pro webový prohlížeč, nebo
  - RESTful API s upřesněním standardu vč. formátu[9].
- Architektura aplikace
  - striktně odděluje frontend a backend,
  - správa uživatelů (user-management) je zajištěna fakultním IDM,
  - využívá maximum dostupných služeb (např. notifikace),
  - [SHOULD] podporuje použití pro více fakult na jediné instanci.
- Aplikace
  - je součástí katalogu služeb FIT[10] od počátku práce na projektu (stav „připravuje se“),
  - využívá mezipaměť pro urychlení obsluhy požadavků,

[5] 10 Examples of Great End User Documentation
[6] Doporučené nástroje pro dokumentaci RESTful API: RAML, Swagger / OpenAPI
[7] Návod ke psaní dokumentace ICT FIT
[8] Např. JavaDoc nebo DoxyGen
[9] Doporučujeme vycházet ze standardu JSON API
[10] Katalog služeb ICT FIT

    - je provozuschopná i v případě nedostupnosti (zpomalení) souvisejících služeb[11].
- [MAY] Aplikace je implementovaná na platformě/jazyku:
    - Ruby,
    - JavaScript, resp. izomorfní JavaScript[12],
    - Groovy/Java na Spring Frameworku,
    - Python,
    - PHP na frameworku Symfony.

Webová přístupnost

Aplikace je přístupná pro uživatele bez ohledu na jejich omezení a zařízení, kterým k aplikaci přistupují.

- Aplikace respektuje požadavky WCAG 2.0 AA[13], zejména
    - sémantické značkování výstupu HTML,
    - jednoznačné perzistentní URL jednotlivých stránek[14],
    - podpora tisku,
    - *progressive enhancement*[15].
- Výstup aplikace (HTML) je v souladu s principem *mobile-first*[16], *media-first*[17], zejména
    - přizpůsobivé uživatelské rozhraní[18],
    - použitelnost ovládacích prvků pro manipulaci prsty,
    - nenáročnost s ohledem na výkon CPU a spotřebu baterie,
    - minimalizace přenesených dat.
- [MAY] Aplikace je odolná vůči výpadkům připojení a funkčnost bez připojení k Internetu.[19]
- [MAY] Aplikace podporuje *Web App Manifest*[20] a integraci do operačního systému[21] zahrnující
    - podporu push notifikací[22],
    - synchronizaci na pozadí přes *Service Workers*[23].

[11] Např. bez datového, resp. autentifikačního, zdroje, informace z mezipaměti, resp. zobrazí jen veřejné informace (s příslušným upozorněním).
[12] Viz Isomorphic JavaScript.
[13] Web Content Accessibility Guidelines (WCAG) 2.0
[14] Viz Cool URIs a Why JavaScript web applications should embrace traditional URLs.
[15] Poskytnout klientovi úplnou funkcionalitu i v případě, že nepodporuje dynamické technologie; viz článek na Gov.UK.
[16] Viz kniha Mobile First (Luke Wroblewski).
[17] Společná definice zobrazení od sémantického obsahu pro čtečky a textové interprety, přes tisk a malé obrazovky až po velké obrazovky.
[18] Viz Responsive Web Design.
[19] Tzv. Offline-First.
[20] Viz Web App Manifest.
[21] Viz články Progressive Web Apps: Escaping Tabs Without Losing Our Soul a Getting started with Progressive Web Apps.
[22] Viz Push Notifications on the Open Web.
[23] Viz Introduction to Service Worker.

Kód aplikace

Veškerý kód je psaný kompletně v angličtině s prioritou udržitelnosti a čitelnosti. Vývoj kódu přehledně odděluje provozní větev od vývojových. Před nasazením prochází každý nově vzniklý kód kontrolou kvality na několika úrovních.

- Aplikace respektuje *best practices* pro psaní udržitelného a čitelného kódu[24]; zejména
  - logické členění kódu do modulů podle funkcionality,
  - specifikace konvencí používaných technologií (např. CSS[25], JavaScript[26], Java[27]),
  - dodržování stylu autora při editaci cizího kódu,
  - minimalizace importů[28] (import, include, using, atd.),
  - používání existujících knihoven[29], kdykoli je to efektivní a smysluplné,
  - používání návrhových vzorů[30],
  - komentování potenciálně nejasných částí.
- Veškerý kód je psaný
  - v UTF-8 s unixovým koncem řádek (řídící znak LF / 0x0A),
  - anglicky (názvy funkcí a proměnných, komentáře, systémová a jiná hlášení).
- Veškeré výstupy (texty pro uživatele) podporují lokalizaci a internacionalizaci.
- Pro aplikaci existuje kompletní česká lokalizace.

Verzování

Vývoj kódu je organizovaný s přehledným oddělením provozní a vývojové větve. Umožňuje operativní opravy kritických chyb (hotfix) a nezávislý vývoj nových funkcí. Podporuje bezpečný model nasazování nových verzí[31] pro účely testování a ladění (akceptační testy).

- Kód aplikace je vyvíjen na revizním systému Git[32] využívající
  - repozitář na službě GitLab provozované fakultou[33] nebo oddělením ICT[34],
  - standardní branching model Git Flow[35]
    (nástroje OMGF[36] nebo Git-Flow Cheatsheet[37]) a
  - sémantické verzování[38].
- Používání revizního systému se řídí pravidly *commitování*, zejména
  - *commit* každé dílčí změny funkcionality,
  - zachování funkcionality celku přes jednotlivé *commity*,

[24] Viz Best Practices a kniha The Pragmatic Programmer.
[25] Konkrétně konvenci SUIT CSS.
[26] JavaScript Quality Guidelines and Recommendations
[27] Code Conventions for the Java Programming Language
[28] Např. neimportovat celý balíček, když z něj bude použita jen malá část.
[29] Pod svobodnými nebo open-source softwarovými licencemi a respektovat podmínky těchto licencí.
[30] Gang of Four Design Patterns či Design Patterns na Wiki
[31] Deployment environment
[32] Jak na Git
[33] GitLab FIT ČVUT
[34] GitLab ICT
[35] Git Flow
[36] OMGF
[37] Git-Flow Cheatsheet
[38] Semantic Versioning 2.0.0

    - používání rozkazovacího tvaru v přítomném čase[39].
- Součástí vývoje je udržování aktuálního souboru CHANGELOG[40] dostupného z webu na úrovni
    - nových funkcí (či inovací) vždy při jejich začlenění do vývojové větve,
    - nových MINOR verzí vždy při začlenění vývojové větve do provozní.
- Na společných (sdílených) větvích není povoleno přepisování historie.
- Veškeré texty verzování jsou anglicky.

Kontrola kvality (QA)

Veškerý kód se před nasazením patřičně kontroluje na úrovni automatizovaných nástrojů a dílčích (jednotkových a dalších) testů. Součástí kontroly kódu je (jednoduchý) schvalovací proces. Alternativně se kód vyhodnocuje, zda splňuje stanovené kvalitativní metriky.

- Vývoj kódu se opírá o kontrolní nástroje jako zejména
    - příslušný *linter*[41].
- Veškeré nasazování změn kódu (merge) procházejí kontrolním procesem[42] s následujícími pravidly.
    - Veškeré merge jsou prováděny formou požadavků na začlenění[43] (dále PR).
    - Veškeré PR (bez ohledu na svou podstatu a závažnost) budou potvrzované minimálně druhým členem týmu – programátorem, alternativně nadřízeným.
    - PR kritického požadavku si může jeho řešitel sám akceptovat. O takovém úkonu neprodleně vyrozumí členy týmu. Povinnost potvrzení podle předchozího bodu zůstává, však může být učiněno dodatečně (bez zbytečného prodlení).
    - Součástí kontrolního procesu nasazování je continuous integration[44] na GitLabu[45].
- Kód obsahuje automatické testy, mezi které patří zejména
    - jednotkové testy,
    - integrační testy (API, resp. automatizované průchody).
- Klíčová funkcionalita je ošetřena testy, které jsou specifikované v dokumentaci s maximální mírou automatizace.
- Vývoj kódu je řízený testy[46] [SHOULD].
- Minimální požadované hodnoty metrik[47] pomocí fakultní služby Sonar[48] jsou stanoveny následujícím způsobem: [SHOULD]
    - Method Total Length (< 30 lines)
    - Class Total Length (< 300 lines)
    - Unit Tests Line coverage (> 70 %)
    - Unit Tests Branch coverage (> 70 %)
    - Density of duplicated lines (< 5 %)
    - Lack of cohesion of methods (< 3)
    - Average complexity by method (< 5)

[39] How to Write a Git Commit Message
[40] Keep a Changelog
[41] Platí zejm. pro dynamické a značkovací jazyky; viz např. doporučení pro JavaScript a CSSLint.
[42] Best Practices for Code Review a Code reviews v praxi
[43] Code Review Via GitLab Merge Requests
[44] Continuous integration
[45] GitLab Continuous Integration
[46] Test-Driven Development
[47] Sonar Metric Definitions
[48] Sonar FIT ČVUT

    - Rules compliance index (žádné závady úrovně „blocker“ ani „critical“)

Provoz, údržba a rozvoj aplikace a podpora uživatelů

Veškerá (nově vznikající) funkcionalita je uživatelům dostupná přehledně a jednoduše. Aplikace (nová verze) se nasazuje do provozu po splnění akceptačních testů. Součástí údržby a dlouhodobého rozvoje aplikace je sběr informací o používání a jejich pravidelné vyhodnocování následované patřičným zapracováním do aplikace.

- Vzhled uživatelského rozhraní (UI) aplikace je moderní, přehledný, vzdušný a tvořený obsahem.[49]
- Sada akceptačních testů je specifikovaná pro účely testování všech dostupných a nově vznikajících funkcí.
- Sběr dat se provádí na základě
    - zpětné vazby uživatelů prostřednictvím
        - funkce issue tracking v rámci GitLabu a
        - e-mailu na helpdesk,
    - logování[50] a integrace s monitorovacími službami na úrovni
        - chyb (fatal, warning),
        - informačních zpráv o používání[51] (používanost funkcí, doby trvání, přístupy) a
        - systémových zpráv a dalších výstupů.
- Proces vyhodnocování dat zahrnuje
    - podporu uživatelů a
    - opravy chyb včetně klasifikace jejich závažnosti.
- [SHOULD] Proces rozvoje od návrhu po realizaci za účelem
    - vylepšování stávajících funkcí (optimalizace chodu a procesů),
    - přidávání nových funkcí.
- [SHOULD] Proces testování nových verzí aplikace zahrnuje
    - provoz nezávislé (beta) verze,
    - podporu AB testování,
    - provádění inspekcí a heuristik,
    - pozorování.

Bezpečnost a ochrana osobních údajů

Aplikace je standardně zabezpečená; zejména nepracuje s hesly uživatelů a veškerá komunikace probíhá přes šifrovaný protokol. Aplikace také respektuje nařízení rektora o ochraně osobních údajů.

- Aplikace respektuje principy bezpečných webových aplikací[52], jmenovitě
    - veškerá komunikace (S2S, S2C) probíhá přes HTTPS,
    - jako API poskytuje různé úrovně oprávnění pomocí *scopes*[53],

[49] UXMyths: You don’t need the content to design a website.
[50] Logging Best Practices
[51] Google Analytics s využitím událostí (events)
[52] Viz principy OWASP.
[53] Securing Access with OAuth2: How to deal with OAuth Scopes

    - nepracuje s hesly uživatelů; autentizace, resp. autorizace, probíhá přes Shibboleth (není-li potřeba autorizace), resp. autorizační server FIT (protokol OAuth 2.0)[54].
- Aplikace respektuje nařízení rektora o ochraně osobních údajů[55], zejména dokumentace definuje,
    - jaké informace jsou citlivé/osobní,
    - jaká data jsou dostupná kterým uživatelům v závislosti na autorizaci uživatele (např. anonymní uživatel, přihlášený uživatel, student, vyučující, administrátor),
    - které citlivé/osobní informace jsou přístupné v rozporu s nařízením.

Další koncepty webových aplikací (W2.0)

Aplikace explicitně zohledňuje možnosti využití níže uvedených webových konceptů a případně dalších. Vzhledem k omezení jednotlivých projektů mohou být využití konceptů pouze součástí projektové dokumentace – byť jen jako potenciální rozšíření funkcionality s uvedenými přínosy a konkrétními příklady.

- Dokumentace (např. v příloze) popisuje možnosti a úroveň nezbytnosti využití všech následujících konceptů:
    - RSS,
    - personalizace,
    - customizace,
    - folksonomie (tagování),
    - social networking,
    - real-time web[56],
    - crowdsourcing[57],
    - kolaborace,
    - průvodce (wizardy),
    - konfigurátory (rozšířených dotazů vyhledávání, parametrů služby),
    - gamifikace[58],
    - mikrodata[59].

[54] Viz Autorizační server FIT (OAuth 2.0).
[55] Příkaz rektora č. 5/2015 Ochrana osobních údajů na ČVUT v Praze
[56] Real-time web (Wiki)
[57] Crowdsourcing (Wiki)
[58] Gamification (Wiki)
[59] Microdata (Wiki)

Příloha B

Instalační příručka

V této příloze popisuji, jak lokálně zprovoznit aplikaci Card-Manager. Následující postup je použitelný pro Unix-like operační systémy při splnění dvou požadavků:

- PHP >= 7.0
- MySQL >= 5.5
- Bower[28].

1. Nejprve zkopírujte obsah ve složce `impl/src` z média a přejděte do složky, kam jste obsah nakopírovali.
2. Následně je třeba nainstalovat nástroj pro instalaci back-end závislotí, který se jmenuje Composer[29] a nainstalovat back-end závislosti. Během instalace budete dotázáni na několik parametrů, ty prosím vyplňte.

```
$ sudo ./getcomposer.sh
$ composer install
```

3. Volitelně můžete ověřit, že aplikace splňuje veškeré požadavky Symfony.

```
$ bin/console symfony_requirements
```

4. Následně je třeba vytvořit databázi a databázové schéma podle parametrů, na které jste byli dotazováni v prvním kroku.

```
$ bin/console doctrine:database:create
$ bin/console doctrine:schema:create
```

5. Dále je třeba databázi naplnit testovacími daty.

```
$ bin/console doctrine:fixtures:load
```

6. Poté je nutné do databáze nahrát aplikační role a stanovit administrátora aplikace.

```
$ bin/console card-manager:init-roles
$ bin/console card-manager:init-admin vondrm12
```

7. Následně nainstaluje front-end závislosti pomocí nástroje Bower a zkompilujte je.

```
$ bower install
$ bin/console assets:install --symlink web
$ bin/console assetic:dump
```

8. V předposledním kroku je nutné spustit webový server.

```
$ bin/console server:run localhost:8000
```

9. Nyní se na adrese `http://localhost:8000/login` můžete přihlásit uživatelským jménem `vondrm12`. Heslo je stejné.

Příloha C

Seznam použitých zkratek

ACS Access Control System

ACS-C ACS-Controller

ACS-M ACS-Manager

API Aplication Programming Interface

CRUD Create, Retrieve, Update, Delete

C-M Card-Manager

DI Dependency Injection

HTML HyperText Markup Language

HTTP Hypertext Transfer Protocol

IdM Identity Manager

JSON JavaScript Object Notation

K4 systém pro správu přístupu v budovách

Lo-Fi Low-Fidelity

OJ organizační jednotka

REST Representational State Transfer

UC Use Case

UI User Interface

YAML YAML Ain't Markup Language

Příloha D

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu média
- `impl`
  - `install.txt` ..... instalační příručka
  - `src` ..... zdrojové kódy implementace
- `text`
  - `BP_Vondrak_Martin_2017.pdf` ..... text práce ve formátu PDF
  - `src` ..... text práce ve ve formátu LaTeX