České vysoké učení technické v Praze

Fakulta elektrotechnická

Katedra počítačů

Katederní knihovna s možností samoobslužného provozu

———

Diplomová práce

Autor:
Bc. Jan Skála

Vedoucí projektu:
Ing. Petr Novák, Ph.D.

Magisterský program:
Otevřená informatika

Obor:

Softwarové inženýrství

2021

Čestné prohlášení

Prohlašuji, že svou diplomovou práci na téma Katederní knihovna s možností samoobslužného provozu jsem vypracoval samostatně pod vedením vedoucího diplomové práce, s použitím odborné literatury a dalších informačních zdrojů citovaných v práci a uvedených v seznamu literatury na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Praze dne ....

Bc. Jan Skála

Poděkování

Tímto bych chtěl poděkovat Ing. Petru Novákovi Ph.D. za možnosti podílet se na zajímavém praktickém projektu. Zároveň také za vedení a cenné rady během formování konceptu celé aplikace. Dále Anastasii Surikové a Dominiku Brychovi za cenné informace při opravě chyby ve vykreslování navigačního menu projevující se jen na prohlížeči Safari. Na závěr Janu Hájkovi za pomoc při řešení nefungující autentizace v prostředí cloudu Microsoft Azure.

Abstrakt

Tato práce se zabývá tvorbou informačního systému umožňujícího správu inventáře knihovny na vybrané katedře ČVUT. Kromě základních funkcí bude poskytovat i možnost samoobslužného provozu. Výstupem práce jsou celkem 4 aplikace: Webová aplikace zobrazuje data a umožňuje přihlášení pomocí uživatelského jména a hesla ČVUT. Desktopová aplikace na kiosku zobrazující webovou aplikaci ve vestavěném prohlížeči poskytuje integraci s připojeným hardwarem. REST API služba komunikuje s kioskem. Služba na pozadí připomíná uživatelům datum vrácení zapůjčené publikace.

Klíčová slova Knihovna, kniha, informační systém, administrace, samoobslužný provoz, web, .NET 5, autentizace, autorizace, integrace, API, REST, Swagger, Blazor, přístupová karta, NFC, QR kód

Abstract

This work's goal is to create an information system allowing management of libraries provided by university departments. Besides core functionality the system also provides self-service features. The output of this thesis are 4 applications: A web application displaying data and authenticating users using existing CTU credentials. Desktop application (kiosk) showing the web app in an embedded browser and integrating with connected hardware. REST API service communicating with kiosk. And finally, a background service reminding users to return publications they borrowed.

Key words Library, book, information system, administration, self-service, web, .NET 5, authentication, authorization, integration, API, REST, Swagger, Blazor, access card, NFC, QR code

České vysoké učení technické v Praze
Fakulta elektrotechnická
© 2021 Jan Skála. Všechna práva vyhrazena.
Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě elektrotechnické. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí, je nezbytný souhlas autora.

Odkaz na tuto práci Skála, Jan. Katederní knihovna s možností samoobslužného provozu. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta elektrotechnická, 2021.

OBSAH

1 Úvod

Pod pojmem „Katederní knihovna“ si člověk představí většinou nějakou jednoduchou aplikaci pro správu menšího množství knížek. Tedy v podstatě nic složitého. Přihlédneme-li rovněž k umístění na elektrotechnické fakultě, tak se realizace takovéhoto projektu zdá více než samozřejmostí.

Ovšem skutečnost je poněkud jiná. Nejmenší katedry nemají dokonce žádnou vlastní evidenci knížek. Několik, zejména těch menších kateder, pro správu inventáře knihovny používá dokonce stále ještě papírové kartičky řazené v šuplících. Ty obsahují údaje, kdy a komu byla kniha půjčena. Větší katedry spravující až tisíce knih mají většinou nějaké lokální aplikace vytvořené v rámci bakalářských nebo diplomových prací.

I na menších katedrách může s postupem času začít přibývat zaměstnanců a tím současně i knih. Díky tomu se potřeby uživatelů knihovny rozšiřují, což vede k postupné obměně požadavků na činnost používaného informačního systému. Dříve si knížky převážně pořizovali pouze konkrétní lidé a ti je měli po celou dobu jejich působení na katedře u sebe. V současné době se však trend poněkud mění. Pořizují se knížky, jež se sdílejí, půjčují se studentům na bakalářské a diplomové práce, případně i externím pracovníkům na grantech. Proměnou prochází i doba, po kterou je kniha zapůjčena. Dnes si návštěvníci knihovny často z knihy chtějí prostudovat jen malou část, potřebují si ji tedy vypůjčit v podstatě okamžitě a za krátký čas ji snadno opět vrátit. Obvykle se tedy nestává, aby si zájemce knihu půjčil a snažil se ji přečíst celou, ale spíš vykonává cílenou rešerši.

Naprostá většina knihoven je spravována tzv. knihovníkem představující osobu v podstatě administrativního charakteru, tedy vydávající a vracející knížky do fyzické knihovny. Takováto knihovna má často podobu zamčené skříně umístěné někde na chodbě v prostorách katedry na ČVUT. Toto velmi znepříjemňuje a v podstatě znemožňuje rychlé půjčení a vrácení knihy. Zejména prohlídku několika knížek a výběru té potřebné. Avšak některé knihovny, nebo aspoň jejich části, i dnes podporují určitý stupeň samo-obslužného provozu. Ten je však vhodný pouze pokud má uživatel možnost si knihu při zájmu legálně z knihovny okamžitě vypůjčit a kdykoli ji do knihovny zase legálně vrátit. Toto jinými slovy znamená, vytvořit snadno a okamžitě záznam o zapůjčení knihy a později tento záznam při vrácení zrušit. S časem a vzrůstajícím počtem knih se tedy stále více zvyšuje potřeba na nějaký stupeň (aspoň částečně) samo-obslužného provozu takovéto katederní knihovny.

Na katedře kybernetiky je v knihovně evidováno přibližně 8 tisíc publikací (knihy, sborníky, manuály, a další) pro zhruba 100 (místních) uživatelů. Vydávání a vracení knížek pouze knihovníkem se již v tomto počtu knížek a lidí stává poněkud nepraktické. Zájemce obvykle musí požádat o zapůjčení pomocí emailu a následně se s knihovníkem dohodnout na místě a termínu předání. Z tohoto důvodu je potřeba vytvořit systém pro správu knihovny obsahující vhodný mechanismus pro aspoň částečný samo-obslužný provoz. Podmínkou je, aby si každý mohl knihy kdykoli prohlížet a zapůjčení i vrácení bylo snadné a uživatelsky přívětivé. Nadměrná složitost systému by mohla uživatele odradit a přimět je si knihu půjčit (odnést), aniž by tuto skutečnost kamkoli zaznamenali. Systém by pak chybně zobrazoval některé knihy jako dostupné, díky čemuž by se celý informační systém stal tak trochu zbytečným, neboť by již neposkytoval pravdivé údaje.

Cílem práce je navrhnout takovýto vhodný informační systém pro činnost menší katederní knihovny umožňující rovněž i určitý stupeň samoobslužného provozu. To znamená s minimálním

vynaložením lidských zdrojů pro každodenní činnost knihovny. Umožnit tedy nejen základní činnosti jako je správa knih, prohlížení obsahu knihovny komukoliv, ale současně poskytnout možnost přímého zapůjčení a vrácení publikace nejlépe přes webové rozhraní nebo prostřednictvím nějakého kiosku umístěného v místě fyzické knihovny.

Stále více se setkáváme s využitím moderních technologií, jako jsou NFC štítky nebo QR kódy používané u skutečně velkých (často automatizovaných) knihovních systémů pro statisíce knih. Tyto technologie však nejsou v podstatě nijak využívány u malých knihoven, kde by rovněž mohly velmi zjednodušit jejich celkovou činnost.

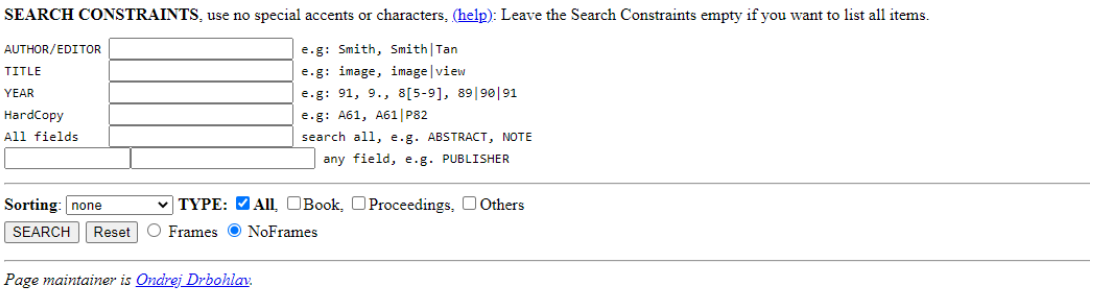
Studenti nebo zaměstnanci ČVUT, kteří budou mít zájem se do knihovny přihlásit se, nebudou muset nikterak registrovat. Přihlásit se budou moci pomocí již existujících přihlašovacích údajů ČVUT nebo s použitím (identifikační) osobní karty. Systém následně získá všechny potřebné informace o uživateli pomocí služby **usermap.cvut.cz**.

2 ANALÝZA

Zde vytvářené řešení je primárně určeno pro katedru kybernetiky. Ta je z historických důvodů rozdělena na dvě logické části. Tímto je i správa knihovny katedry v současné době rozdělena na dvě samostatné aplikace.

První (poněkud větší) část využívá již asi 20 let starou webovou aplikaci, do které má administrativní přístup pouze knihovník. Kdokoli na katedře může obsah této knihovny jen prohlížet přes webové rozhraní, avšak pro zapůjčení nebo vrácení knihy musí kontaktovat knihovníka. Bohužel některé údaje knih (týkající se například elektronických publikací) již není možné ani vložit, funkce vyhledávání je velmi omezená a o nějakém (i v menší míře provedeném) samo-obslužném provozu nelze uvažovat. Nalezení knihy a vytvoření záznamu o jejím vypůjčení je poměrně složité a neexistuje způsob zpětné verifikace, což brání uživateli, aby tak učinil sám.

Druhá (poněkud menší) část využívá rovněž velmi starou aplikaci napsanou v jazyce Python. Datum vzniku nebo poslední úpravy se mi nepodařilo přesně stanovit. Aplikace pracuje zcela odlišně a poskytuje skutečně minimalistické webové rozhraní s velmi omezeným hledáním (Obrázek 1: Webové rozhraní jedné části současné knihovny).



Obrázek 1: Webové rozhraní jedné části současné knihovny

Jde v podstatě o jednoúčelový skript spouštěný z příkazového řádku. Knihovnu (obsah databáze) mohou prohlížet pouze uživatelé mající účet na příslušném serveru, kde je umístěn zmíněný skript. Nikdo jiný (z katedry) se do této oddělené knihovny jen tak nepodívá. V této části knihovny naopak samo-obslužný provoz na bázi jednoduchého principu pracuje vcelku uspokojivě. Zájemce si nejprve fyzickou knihu prohlédne, a pokud si ji chce půjčit, přihlásí se na svém počítači výše zmíněným účtem. Následně spustí proceduru definovanou ve zmíněném skriptu a té předá identifikátor knihy jako parametr. Vrácení probíhá obdobným způsobem.

Tento stav na katedře vznikl v době, kdy se v inventáři knihovny vyskytovalo odhadem jen několik stovek knih, a katedra čítala jen pár desítek zaměstnanců. Tehdy se knihy téměř vzájemně nepůjčovaly. V těch krajních případech, kdy k tomu docházelo, to bylo jen formou ústní dohody. V současnosti při stovce zaměstnanců a několika tisících knihách je nutné tento stav poněkud změnit.

Zbytek kapitoly se věnuje analýze problematiky. Na základě způsobu používání dvou zmíněných aplikací a ve spolupráci s vedoucím práce jsou stanoveny požadavky na činnost nového knihovního informačního systému. Poté jsou rozděleny uživatelské role a oprávnění v aplikaci. Pomocí zpětné analýzy systémových požadavků definuji jednotlivé entity nacházející se v systému. Současně s nimi identifikuji klíčové uživatelské procesy. Další část popisuje způsoby autentizace v rámci instituce ČVUT. Předposlední část se snaží namapovat existující požadavky na již existující open-source

řešení. A závěrečná část analýzy obsahuje rozbor integrovaných systému potřebných k činnosti celého řešení. V práci je zavedeno několik pojmů, na něž se později v textu odkazuje:

Uživatel v rámci ČVUT – Student nebo zaměstnanec ČVUT disponující přihlašovacím jménem (login) a heslem ČVUT.

Přístupová karta – Libovolná NFC karta, může, ale nemusí to být přístupová karta studenta nebo zaměstnance ČVUT.

Kiosk – Malý vestavěný počítač v prostorách knihovny vybavený NFC[1] čtečkou a kamerou pro snímání QR [2]kódů.

Částečně samoobslužný provoz – Knihovna není volně přístupná, často se nachází za dveřmi na přístupovou kartu. Knihy si uživatelé smí půjčit pouze na kiosku.

Plně samoobslužný provoz – Knihovna je volně přístupná a uživatelé si smí půjčovat knihu i ze svého zařízení (mobilní telefon, počítač, …).

NFC štítek – Malý plastový štítek, jehož informace lze číst pomocí NFC čtečky.

QR kód – Strojově čitelný obrázek s bodovou maticí obsahující zakódovaná data.

Webová aplikace – Webová stránka zobrazená na zařízení uživatelů nebo v aplikaci na kiosku v prostorách knihovny. Obsahuje hlavní funkcionalitu zde vytvářeného řešení.

REST [3]API [4]služba – Aplikace vystavující REST rozhraní sloužící ke komunikaci s kioskem.

Služba na pozadí – Aplikace na serveru provádějící úkony na pozadí.

Aplikace na kiosku – Desktopová aplikace na kiosku v prostorách knihovny. Disponuje vestavěným webovým prohlížečem. Zobrazuje výše zmíněnou webovou aplikaci a komunikuje s NFC čtečkou a kamerou.

Knihovna – Instituce sloužící primárně ke zpřístupnění publikací (nejen knih).

Programová knihovna – Souhrn procedur a funkcí implementující nějakou funkcionalitu.

2.1 POŽADAVKY NA ČINNOST SYSTÉMU

V této podkapitole jsou definovány požadavky na činnost systému. Každý podnadpis reprezentuje jeden z požadavků a je následován krátkým slovním popisem.

Prohlížení obsahu

Knihy v knihovně (jejich záznamy) jsou volně přístupné pro prohlížení komukoliv. Knihy lze filtrovat podle názvu, autora, klíčových slov nebo typu publikace. Současně je umožněno full-text vyhledávání poskytující vyhledávání ve všech polích současně napříč všemi typy publikací.

Přihlášení přes identitu ČVUT

Zaměstnanci a studenti ČVUT se budou moci do knihovny přihlásit svými již existujícími přihlašovacími účty.

---

[1] Near field communication (NFC) je modulární technologie bezdrátové rádiové komunikace na velmi krátkou vzdálenost (do 4 cm)

[2] Strojově čitelný obrázek s bodovou maticí obsahující zakódovaná data.

[3] Representational State Transfer – Implementuje základní operace (CRUD) pomocí metod HTTP protokolu (GET, POST, PUT, DELETE).

[4] Application Programming interface – Rozhraní sloužící pro komunikaci. Zapouzdřuje složitější funkcionality za jednodušší metody a procedury, typicky CREATE, READ, UPDATE, DELETE.

Registrace karty

Umožní přihlášeným uživatelům registrovat si přístupovou (ideálně ČVUT) kartu pro pozdější použití k rychlé autentizaci při výpůjčce nebo vrácení knih na kiosku. Uživatelé mohou volitelně zabezpečit přihlášení pomocí své karty i pinem.

Správa hostů

Host je v samoobslužné knihovně každý uživatel nedisponující uživatelským účtem ČVUT. Knihovník může do systému hosty přidat nebo odebrat.

Zapůjčení knihy

Knihovník může zapůjčit libovolnou dostupnou knihu každému uživateli nebo každému hostovi. Zároveň může knihu za každého uživatele nebo hosta vrátit.

Evidence výpůjček

Návštěvník si může zobrazit, jaké knihy má právě zapůjčeny i jaké knihy si půjčil v minulosti. Knihovník si může zobrazit všechny výpůjčky.

Hodnocení od čtenářů

Čtenáři smí ohodnotit vypůjčenou knihu. Uživatelé knihovny poté vidí hodnocení pod každou knihou.

Integrace s Google Books (hledání ISBN)

Nalezení knihy podle ISBN pro předvyplnění údajů knihy při jejím vkládání do knihovny. Knihovník bude mít možnost nechat si před-vyplnit údaje o knize na základě zadaného ISBN/ISSN.

Upomínky

Systém bude automaticky připomínat uživatelům datum vrácení knihy, pokud se termín blíží nebo již uplynul.

Newsletter

Při vložení nové knihy do knihovny může knihovník odeslat newsletter informující zaregistrované návštěvníky knihovny o možnosti zapůjčení nové knihy.

Hlídač dostupnosti knih

Umožní uživatelům nastavit si upozornění na knihu, pokud je momentálně někomu zapůjčena, a není tedy dostupná. Budou tedy notifikování, jakmile bude kniha vrácena.

Informace o knize

Některé knihy budou disponovat QR kódem nebo NFC štítkem. Po naskenování štítku nebo kódu se zobrazí detailní informace o knize.

Samoobslužný provoz

Pro vypůjčení a vrácení knihy postačí uživatelům k autentizaci přihlášení přes webové rozhraní nebo jejich přístupová karta (na kiosku). Knihovník může některým uživatelům umožnit půjčovat

si knihy svévolně prostřednictvím kiosku nebo dokonce prostřednictvím webové aplikace (z jakéhokoli místa). V případě kiosku se zrychluje zapůjčení knihy jejím přiložením ke čtecímu zařízení na kiosku. Stejným způsobem bude probíhat i vrácení knihy.

Správa knih

Knihovník smí spravovat databázi knih. Kromě údajů o knize jako název, autor a klíčová slova, může nastavit i její stav. Stav knihy udává, zda se má uživatelům v aplikaci vůbec zobrazit a zda si ji lze půjčit.

Import dat

Knihovník může naimportovat údaje o knihách prostřednictvím nahrání CSV souboru. Pokud záznam o knize neexistuje, bude vytvořen, pokud však importovaná kniha v systému již je, budou její data aktualizována. Tato operace nesmí nijak ovlivnit existující záznamy o výpůjčkách.

Export dat

Knihovník může požádat o export všech dat z aplikace, buď pro účely zálohy, migrace nebo hromadné editaci knih. Knihy zahrnuté do exportu lze vybrat pomocí stejného filtru, jaký slouží i pro jejich vyhledávání.

Vymazání dat

Knihovník může vymazat databázi knih. Tato operace odstraní z databáze knih všechny knihy, jež nejsou aktuálně zapůjčeny. Vymazané knihy je možné poté znovu importovat. Stejně jako v předchozím případě ani tato operace nesmí nijak ovlivnit existující záznamy o výpůjčkách.

Životopis knihy

Aplikace zobrazí všechny aktivity týkající se vybrané knihy. Zobrazí tedy protokol aktivit vypůjčitelů spolu s datem vypůjčení a případného vrácení. Informace bude také obsahovat způsob, jakým se kniha vypůjčila a jakým způsobem byla vrácena.

Dostupnost na mobilních a vestavěných zařízeních

Aplikace musí být přístupná na mobilním zařízení nebo na vestavěném zařízení v místě knihovny (kiosek). Musí být schopna odbavit všechny požadavky pro běžné uživatele.

2.2 UŽIVATELSKÁ OPRÁVNĚNÍ

V této podkapitole jsou nejprve definovány systémové role. Každý uživatel může být členem jedné nebo více rolí. Každá role obsahuje slovní popis kompetencí případně diagram případu užití. Ten zároveň zachycuje prerekvizity pro některé akce. Ke konci kapitoly jsou stanovena autentizační schémata.

2.2.1 Role v systému

1. **Anonymní uživatel (Anonymous)**

Jedná se o uživatele bez jakékoli autentizace. Bude mu umožněno pouze prohlížet obsah knihovny, to znamená, jaké knihy jsou v knihovně k dispozici a základní informace o nich. Tuto roli mají implicitně všichni uživatelé.

2. Návštěvník (Visitor)

Smí prohlížet obsah knihovny stejně jako anonymní uživatel. Všichni přihlášení uživatelé jsou implicitně členy této role. Návštěvník po přihlášení do systému prostřednictvím identity ČVUT, si smí rovněž zobrazit komu je kniha právě zapůjčena. Kromě toho si může prohlédnout jaké knihy má aktuálně zapůjčeny sám i jaké knihy si půjčil již v minulosti. Po vrácení může knihu také ohodnotit. Pokud je požadovaná kniha aktuálně zapůjčena, má možnost požádat systém o hlídání její dostupnosti. Systém uživatele automaticky notifikuje, jakmile je kniha vrácena a stane se tedy dostupnou. Smí si registrovat přístupovou kartu pro pozdější rychlé přihlášení na kiosku v knihovně (Diagram užití 1: Návštěvník).

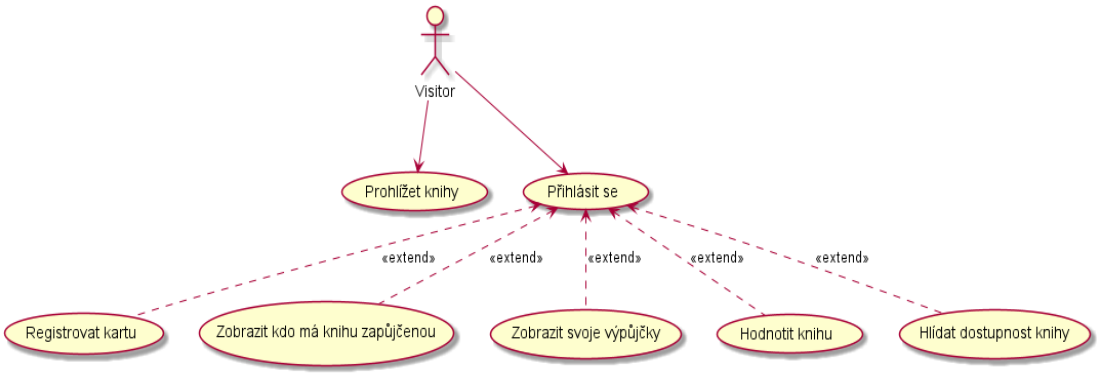


Diagram užití 1: Návštěvník a jeho možnosti v navrhované knihovně

3. Uživatel kiosku (Kiosk User)

V této roli smí uživatel uskutečnit všechny úkony jako návštěvník. Jakmile se však přihlásí na kiosku buď pomocí své zaměstnanecké karty, po předchozí registraci, nebo pomocí identity ČVUT, smí si půjčit knihu z knihovny nebo ji takto vrátit. Do této role musí být uživatel explicitně přidán knihovníkem.

4. Samoobslužný uživatel (Self Service User)

Role samoobslužného uživatele poskytuje uživateli stejná privilegia jako předchozí role. Kromě toho je mu umožněno si knihu půjčit a vrátit nejenom na kiosku, ale i mimo něj (kdekoli pomocí webové aplikace). Stačí tedy, aby se odkudkoli do aplikace přihlásil prostřednictvím identity ČVUT. Pouze knihovník může přidat uživatele do této role.

5. Knihovník (Librarian)

Pouze administrátor smí přiřadit uživatele do role knihovník. Jedině uživatel disponující uživatelským jménem ČVUT se může stát knihovníkem. Knihovník smí samozřejmě vykonávat všechny úkony jako předchozí role. Až po přihlášení může prohlížet evidované výpůjčky, spravovat databázi knih, měnit stavy knih, editovat všechny informace o knize, importovat nebo exportovat seznam knih, vymazat databázi knih, spravovat hosty a udělovat oprávnění k samoobslužné činnosti ostatním uživatelům (Diagram užití 2: Knihovník).

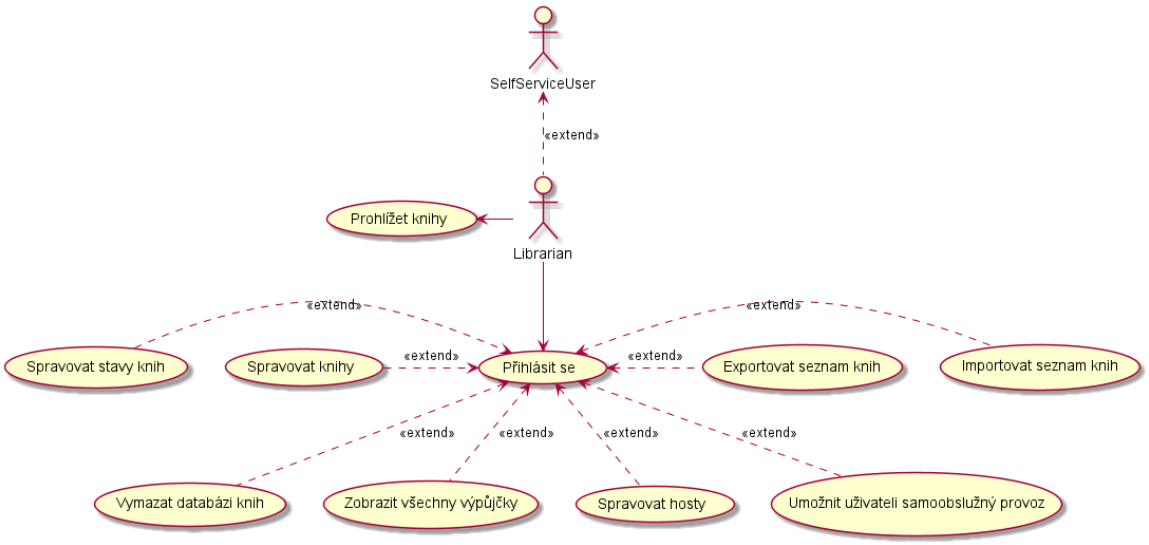


Diagram užití 2: Knihovník a jeho možnosti v navrhované knihovně

6. Správce (Admin)

Do role správce patří ti uživatelé se jmény ČVUT uvedenými v konfiguraci aplikace. Tato role je nadřazena všem ostatním rolím, nicméně v kompetencích administrátora nejsou kompetence knihovníka. Správci je dovoleno nastavovat kdo patří do uživatelské role knihovník. Administrátor však může sám sebe do role knihovníka jmenovat, což je vhodné zejména pro knihovny s méně početným personálem. Administrátor implicitně nedisponuje žádnou jinou rolí, kromě role návštěvníka

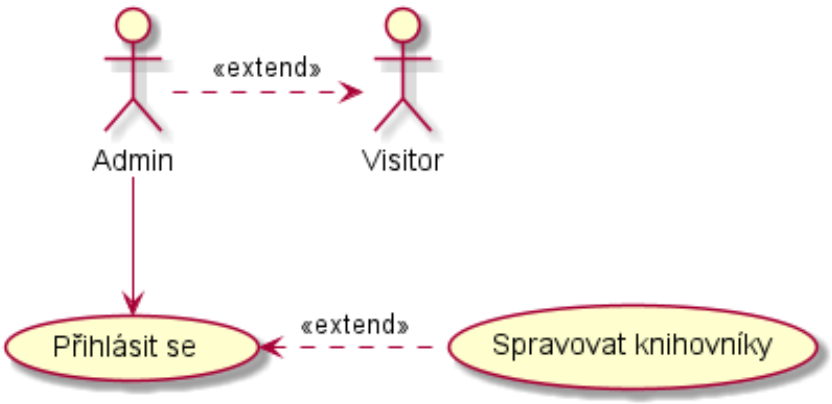


Diagram užití 3: Správce a jeho možnosti v navrhované knihovně

Na základě výše uskutečněné analýzy případů užití jsem definoval diskrétní kontrolu přístupu pro jednotlivé role a znázornil jejich kompetence do tabulky (Tabulka 1 Schéma oprávnění). Aby směl uživatel vykonat nějakou akci, musí patřit alespoň do jedné z rolí, v níž je tato akce umožněna.

Role\Akce	Anonymní	Návštěvník	Uživatel kiosku	Samoobslužný uživatel	Knihovník	Administrátor
Prohlížet obsah	ANO	ANO	ANO	ANO	ANO	ANO
Vidí, komu zapůjčeno	NE	ANO	ANO	ANO	ANO	NE
Půjčit si / vrátit na kiosku[5]	NE	NE	ANO	ANO	ANO	NE
Půjčit si / vrátit online	NE	NE	NE	ANO	ANO	NE
Půjčit / vrátit někomu	NE	NE	NE	NE	ANO	NE
Editovat / přidávat / mazat záznamy	NE	NE	NE	NE	ANO	NE
Umožnit uživateli samoobslužný provoz	NE	NE	NE	NE	ANO	NE
Spravovat knihovníky	NE	NE	NE	NE	NE	ANO

Tabulka 1 Schéma oprávnění pro jednotlivé definované role

Nicméně tento model není pro splnění požadavků dostatečný. Uvažujeme-li akci „půjčit si / vrátit na kiosku“ tak příslušnost k roli „uživatel kiosku“ evidentně nestačí. Systém kromě toho musí ověřit, zda uživatel přistupuje k informačnímu systému ze svého libovolného zařízení nebo skutečně z kiosku v knihovně. Model je tedy nutné rozšířit o autentizační schémata jednoznačné určující, jakým způsobem se uživatel vůči systému autentizoval.

2.2.2 Autentizační schémata

Běžné přihlášení

Uživatel se přihlásil ze svého zařízení (počítače / mobilního telefonu) prostřednictvím webového rozhraní pomocí svého existujícího ČVUT účtu.

[5] Půjčit a vrátit na kiosku, lze jen tehdy, pokud systém detekuje přístup přes kiosk.

Přihlášení na kiosku

Uživatel se přihlásil na kiosku v místě knihovny pomocí svého existujícího ČVUT účtu.

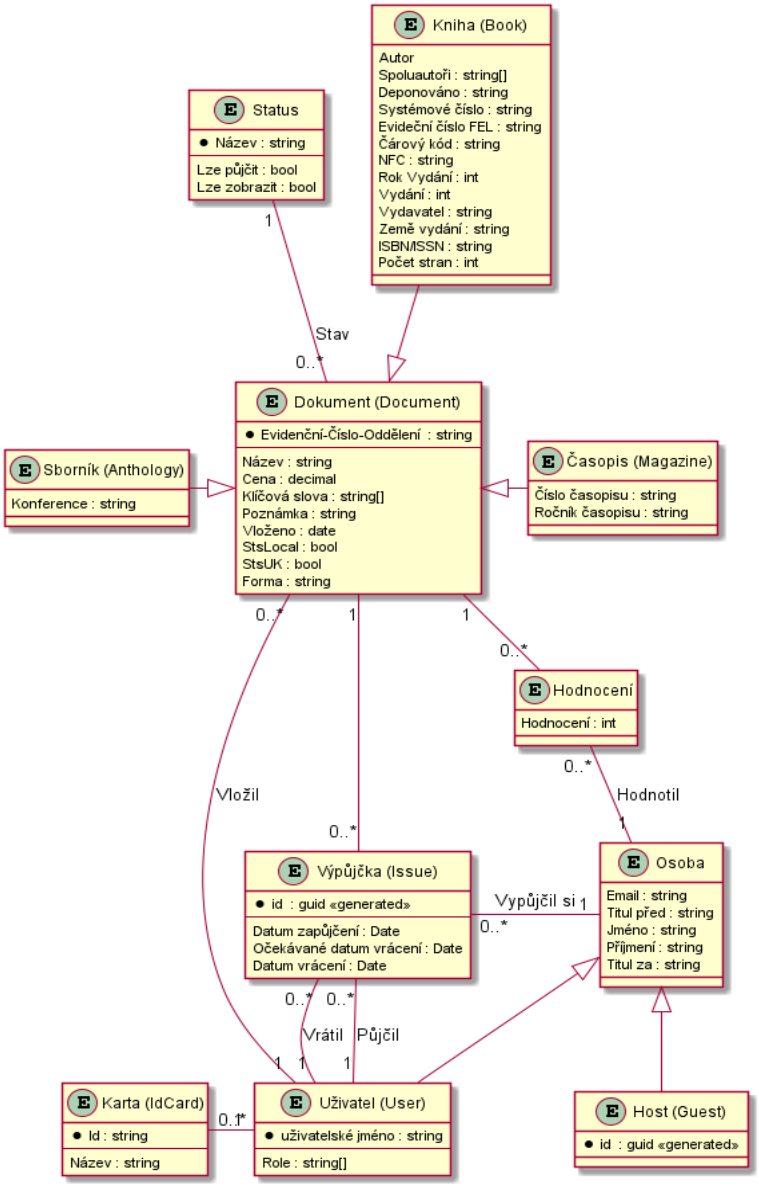
Přihlášení pomocí přístupové karty

Uživatel se přihlásil na kiosku v knihovně pomocí své identifikační karty a případně i nastaveného pinu.

Nyní systému stačí před každým pokusem o zapůjčení nebo vrácení knihy ověřit použité autentizační schéma a roli uživatele. Pokud uživatel disponuje rolí „uživatel kiosku“, ale při přihlášení jej nepoužil, systém pokus o zapůjčení nebo vrácení zamítne.

2.3 SYSTÉMOVÉ ENTITY V KNIHOVNÍM SYSTÉMU

Tato podkapitola mapuje jednotlivé systémové entity. Všechny jsou znázorněny na diagramu (Vývojový diagram 1: Navržené entity). Za ním se nachází popis každé z nich včetně seznamu atributů a jejich významu.



Vývojový diagram 1: Navržené entity v knihovním systému

Dokument (Document)

Entita dokument zahrnuje společné informace pro všechny typy položek v knihovně:

- Evidenční číslo oddělení – označení publikace na katedře, unikátní atribut jednoznačně identifikující každý záznam
- Název – povinné pole
- Cena – cena v korunách
- Klíčová slova – seznam klíčových slov
- Poznámka – libovolný delší formátovaný text
- Vloženo – datum kdy byl záznam do systému vložen
- StsLocal – příznak, zda je kniha evidována pouze v lokální knihovně katedry
- StsUK – příznak, zda je kniha evidována i v centrální knihovně (UK ČVUT)
- Forma – fyzická podoba publikace (CD/DVD, výtisk)
- Stav – stav dokumentu
- Vložil – uživatel, který záznam do systému vložil

Kniha (Book)

Speciální případ entity „Dokument“. Zdaleka nejpočetnější typ publikace vyskytující se v knihovně:

- Autor – první / hlavní autor
- Spoluautoři – seznam spoluautorů
- Deponováno – kde je kniha umístěna (CMP, GL)
- Systémové číslo – označení typu knihy centrální knihovnou
- Evidenční číslo FEL – označení publikace centrální knihovnou
- Čárový kód – číslo z čárového kódu v knize
- NFC – sériové číslo NFC štítku (pokud bude použit)
- Rok vydání – rok vydání (2020, 2021, …)
- Vydání – číslo vydání (1, 2, 3, …)
- Vydavatel – název vydavatele
- Země vydání
- ISBN/ISSN – Číselně zapsané ISBN/ISSN
- Počet stran

Časopis

Speciální případ entity „Dokument“. Kromě sdílených polí obsahuje navíc ještě:

- Číslo časopisu
- Ročník časopisu

Sborník

Speciální případ entity „Dokument“. Kromě sdílených polí obsahuje navíc pole konference pro název konference.

Hodnocení (BookReview)

Každá kniha může být ohodnocena. Hodnotit mohou jen uživatelé disponující přihlašovacími údaji ČVUT a zároveň si knihu alespoň jednou půjčili. Parametrem hodnocení je celé číslo od jedné (nedoporučuji) do deseti (doporučuji) včetně.

Stav knihy

Každá kniha může být označena stavem určující, zda je kniha v systému pro běžné uživatele viditelná a zda si ji lze půjčit.

Osoba

V této entitě jsou uchovány informace o osobě, pokud si půjčila jeden nebo více dokumentů. Osobou může být uživatel disponující přihlašovacími údaji ČVUT (Uživatel). Případně host manuálně registrovaný knihovníkem (Host) pomocí následujících údajů:

- Titul před
- Jméno
- Příjmení
- Titul za
- Email

Uživatel (User)

Speciální případ entity „Osoba“. Kromě společných polí obsahuje pole „uživatelské jméno“ shodující se s neměnným uživatelským jménem ČVUT (tzv. “login") právě přihlášeného uživatele. Entita v sobě zároveň uchovává informaci o tom, do jakých aplikačních rolí uživatel spadá. Údaje o uživatelích systém neuchovává, ale získává je pomocí uživatelského jména ze systému Usermap.

Host (Guest)

Speciální případ entity „Osoba“. Kromě společných polí obsahuje automaticky vygenerovaný unikátní identifikátor „id“. Hosty může do aplikace přidat nebo odebrat pouze knihovník. Entita slouží v případech, kdy je zapotřebí půjčit knihu někomu, kdo se nachází mimo organizaci ČVUT nebo nedisponuje přihlašovacími údaji. Na rozdíl od entity „Uživatel“ jsou informace o hostech uloženy v knihovním systému.

Karta (IdCard)

Entita uchovává údaje o (například) zaměstnanecké kartě studenta nebo zaměstnance ČVUT. V poli „id“ je uložen pouze NFC identifikátor (sériové výrobní číslo NFC karty). Současně je uloženo i pojmenování karty definované uživatelem. Není významné pro autentizaci, ale zlepšuje přehlednost a usnadňuje práci s kartami, zejména pokud si jich uživatel registruje víc.

Výpůjčka

Eviduje informace o zapůjčených dokumentech:

- Datum zapůjčení – kdy byl dokument zapůjčen
- Očekávané datum vrácení
- Datum vrácení – prázdné, pokud dokument ještě nebyl vrácen
- Dokument – vypůjčený dokument

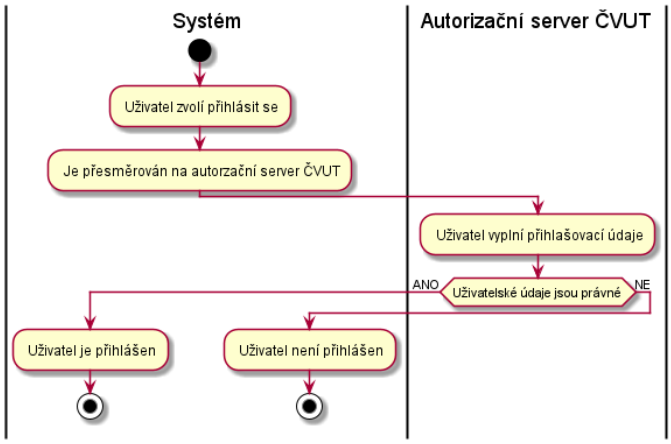
- Vypůjčil si – uživatel nebo host, jemuž je dokument zapůjčen
- Půjčil – kým byl dokument v knihovně zapůjčen, pouze knihovník smí půjčovat dokumenty ostatním, v případě samoobslužného provozu jsou záznamy „Půjčil“ a „Vypůjčil si“ shodné
- Vrátil – kým byl dokument v knihovně vrácen, pouze knihovník smí vracet dokumenty, které si půjčili ostatní, v případě samoobslužného provozu jsou záznamy „Vrátil“ a „Vypůjčil si“ shodné

2.4 SYSTÉMOVÉ PROCESY V KNIHOVNÍM SYSTÉMU

V této podkapitole jsou zachyceny nejdůležitější systémové procesy. Každý je reprezentován slovním popisem a diagramem aktivit. Je zde vysvětlen rozdíl mezi způsoby přihlášení uživatele a zapůjčením knihy.

2.4.1 Přihlášení pomocí identity ČVUT

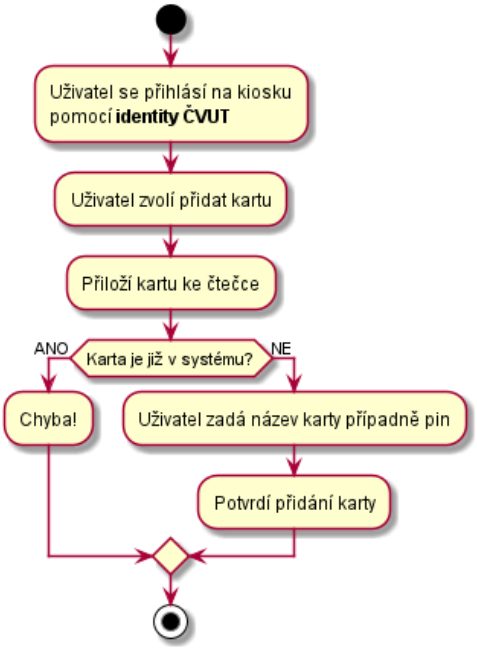
Každý student nebo zaměstnanec ČVUT disponuje přihlašovacími údaji (do některých informačních systémů zaměstnavatele). Systém musí být schopen ověřit každého uživatele na základě těchto údajů. Tyto údaje nesmí být samozřejmě po uživateli přímo požadovány. Namísto toho je uživatel přesměrován na autorizační server ČVUT pro vykonání požadovaného ověření (Vývojový diagram 2: Princip přihlášení pomocí identity ČVUT).



Vývojový diagram 2: Princip přihlášení pomocí identity ČVUT

2.4.2 Registrace přístupové karty uživatele

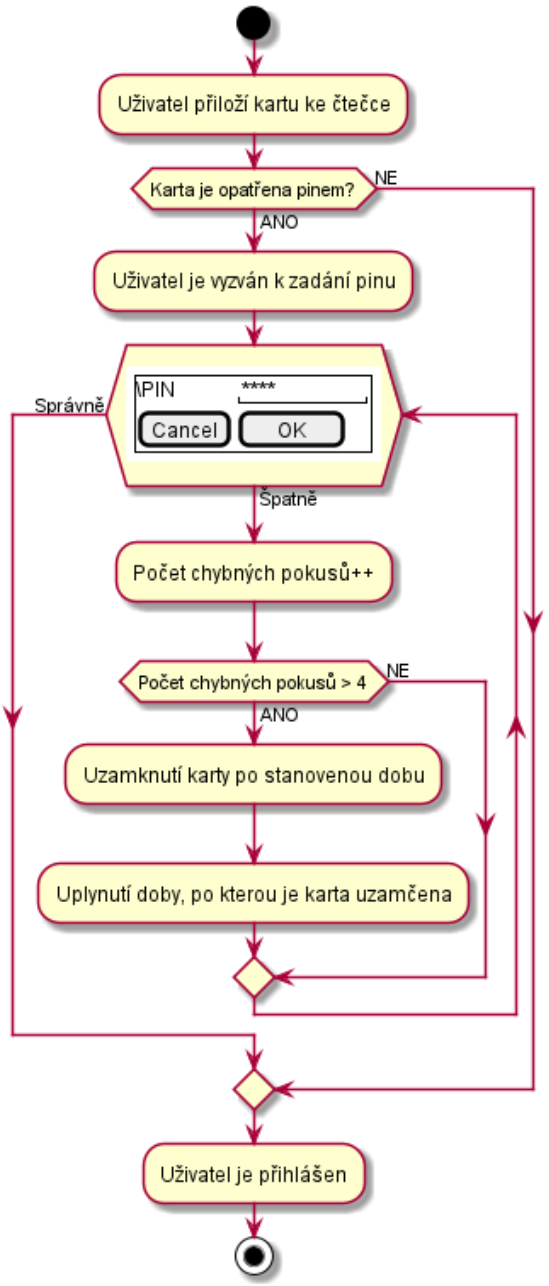
Přístupovou kartu bude číst výhradně čtečka na kiosku v knihovně a je tedy žádoucí, aby byla stejná čtečka využita i k její registraci (zejména z důvodu správné činnosti). Uživatel se nejprve na kiosku přihlásí pomocí identity ČVUT. Následně je mu umožněno přidat přístupovou kartu. Pokud je karta již v systému přidána, tak skončí proces chybou. Pokud nikoli, je uživateli umožněno zabezpečit svou kartu pinem, ten bude vyžadován při každém pokusu o přihlášení pomocí této karty (Vývojový diagram 3: Postup přidání / registrace vlastní přístupové).



Vývojový diagram 3: Postup přidání / registrace vlastní přístupové karty a případně i pinu

2.4.3 Přihlášení pomocí přístupové karty

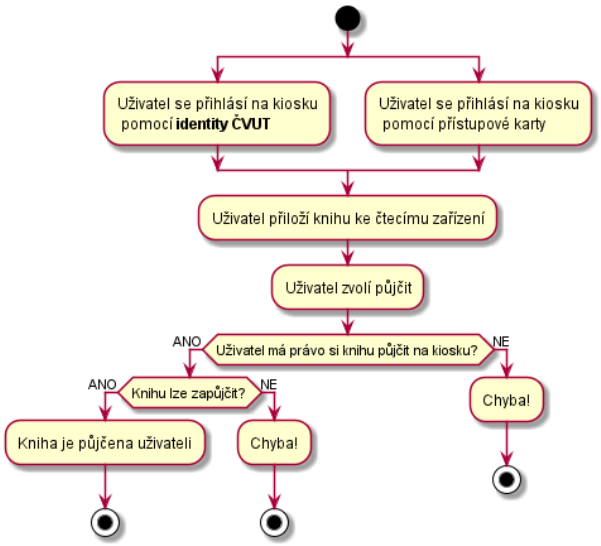
Pokud si uživatel v minulosti přidal přístupovou kartu (Vývojový diagram 3: Postup přidání / registrace vlastní přístupové), je mu umožněno ji použít k rychlému přihlášení na kiosku (Vývojový diagram 4: Postup přihlášení na kiosku pomocí registrované přístupové). PIN bývá obvykle o délce 4 číslice, což není dostatečně velký prostor klíčů, aby byl odolný vůči útokům hrubou silou. Aby potenciální útočník nemohl svévolně zkoušet všechny kombinace je nutné u každé karty hlídat počet chybných pokusů a při překročení stanoveného limitu kartu na nějaký čas uzamknout. Zároveň ne všechny registrované karty musí pinem disponovat (záleží na požadavku uživatele), proto je nutné, aby funkcionalita přihlášení přístupovou kartou byla dostupná pouze na kiosku. Útočníkovi je tak dostatečně zabráněno, aby útokem hrubou silou, kdy náhodně generovat identifikační čísla přístupových karet, objevil v systému nějakou přístupovou kartu nechráněnou pinem.



Vývojový diagram 4: Postup přihlášení na kiosku pomocí registrované přístupové karty

2.4.4 Zapůjčení knihy přímo na kiosku

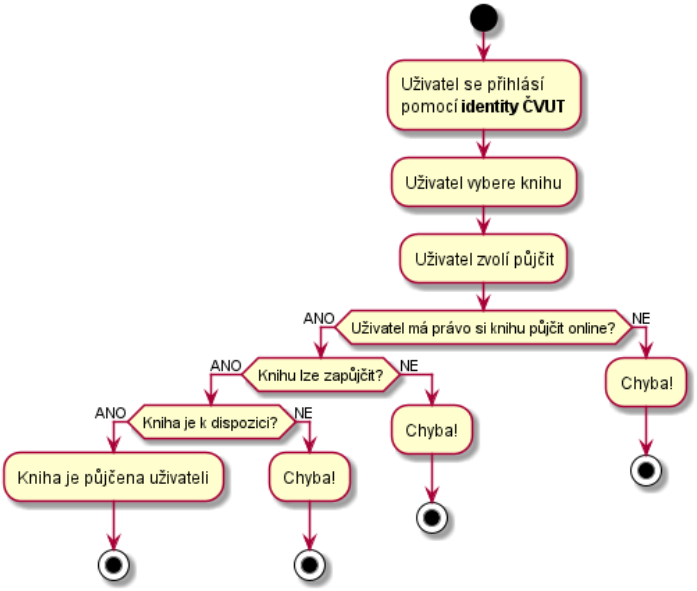
Uživatel se nejprve na kiosku musí přihlásit. Může použít registrovanou přístupovou kartu nebo webové rozhraní s ověřením pomocí identity ČVUT. Při pokusu o zapůjčení knihy systém ověří, zda je uživatel členem role s povolením si svévolně na kiosku půjčit knihu. Pokud ne, autorizace skončí s chybou (Vývojový diagram 5: Postup zapůjčení publikace na kiosku).



Vývojový diagram 5: Postup zapůjčení publikace na kiosku v místě knihovny

2.4.5 Zapůjčení knihy online odkudkoli

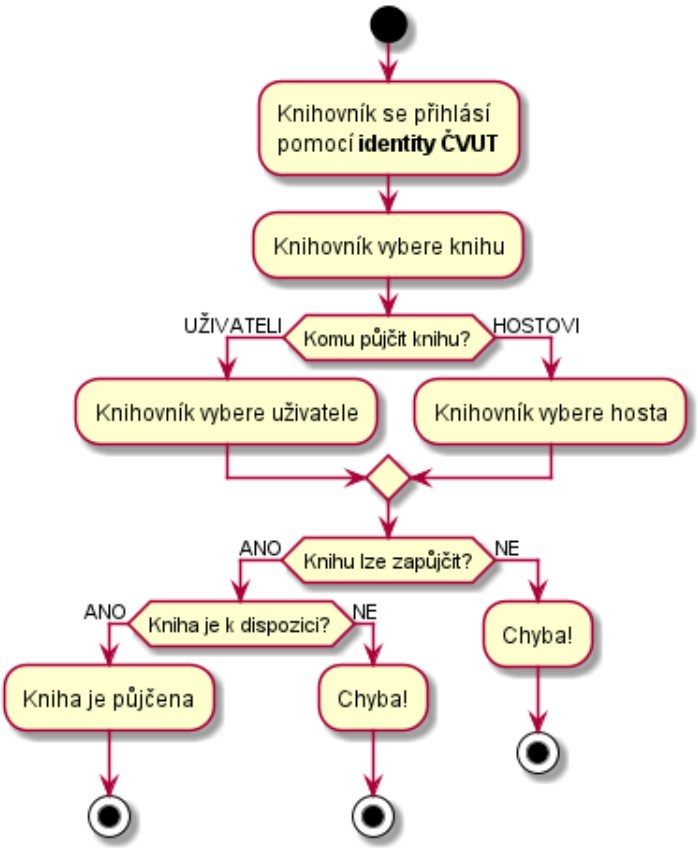
Uživatel se nejprve přihlásí do systému pomocí identity ČVUT. Při pokusu o zapůjčení knihy systém ověří, zda je uživatel členem role s povolením si svévolně půjčit knihu online. Pokud ne, autorizace skončí s chybou (Vývojový diagram 6: Průběh zapůjčení publikace online). Rozdíl oproti předchozímu procesu je v relaxaci podmínky nutnosti použití kiosku.



Vývojový diagram 6: Průběh zapůjčení publikace online z libovolného místa

2.4.6 Zapůjčení knihy knihovníkem

Nejkomplikovanější druh zapůjčení je zapůjčení knihovníkem. Ten je samozřejmě oprávněn půjčit knihu komukoli, a to nejen uživatelům s platným ČVUT účtem, ale i tzv. hostům nemající vůbec žádný účet a nemohou se tedy do knihovny nikterak přihlásit. Knihovník nejprve vybere knihu, a teprve následně vybere, zda ji chce půjčit uživateli nebo hostovi. Poté systém vykoná validaci požadavku stejně jako v předchozích dvou případech, nicméně tentokrát neověřuje práva osoby, jíž je kniha zapůjčena.



Vývojový diagram 7: Zapůjčení publikace přímo knihovníkem

2.5 Autentizace a autorizace v prostředí ČVUT

Existují dva způsoby, jak autentizovat webovou aplikaci (a tedy jejího uživatele), pokud chceme uživatelům umožnit využívat jejich existující účet ČVUT. Tím prvním je Shibboleth. Jedná se o projekt, často využívaný na vysokých školách nebo jiných státních institucích umožňující jednotné přihlášení. Hlavními prvky jsou:

1. Webový prohlížeč – představuje uživatele v rámci procesu SSO[6]
2. Zdrojová data – chráněný obsah, ke kterému chce uživatel přistoupit
3. Poskytovatel ověření (IdP) – ověřuje uživatele (ČVUT)
4. Poskytovatel služby (SP) – poskytuje zdrojová data (katederní knihovna)

Komunikace a výměna dat mezi poskytovatelem ověření a poskytovatelem služby probíhá pomocí zpráv v jazyce SAML. Údaje o uživatelích jsou předávány ve formě metadat prostřednictvím XML souboru. Nutnou podmínkou ke komunikaci je využití SSL/TLS šifrování a zároveň musí metadata obsahovat veřejný klíč (1).

Postup při autentizaci

1. Uživatel snažící se přistoupit k chráněnému zdroji je automaticky přesměrován na SP.
2. SP vydá autentizační žádost a přesměruje uživatele na IdP.

[6] Zkratka pro Single Sign On – mechanismus jednotného přihlášení pro vícero služeb

3. Pokud má uživatel existující relaci, pokračuje dalším krokem, pokud ne IdP jej ověří pomocí jména a hesla.
4. IdP po identifikaci uživatele odešle autentizační odpověď ve formě SAML a přesměruje uživatele zpět na SP.
5. SP zkontroluje autentizační odpověď a vytvoří relaci pro uživatele. Na základě atributů uživatele je mu následně udělen přístup ke chráněnému zdroji.
6. Uživatel má přístup k chráněnému zdroji.

Alternativou je využít autorizační protokol OAuth 2.0 běžně používaný pro zabezpečení REST API. Jeho výhoda spočívá v poskytnutí přístupu klientské aplikaci k datům na serveru, aniž by klientská aplikace znala heslo uživatele. Velkou výhodou oproti SAML je snadnější implementace a dostupnost na běžně používaných platformách. Hlavními prvky jsou:

- Uživatel – vlastní chráněný zdroj
- Služba (server s daty) – poskytuje chráněný zdroj
- Klientská aplikace – přistupuje ke chráněnému zdroji
- Autorizační server – vydá přístupový token klientovi, v případě úspěšné autentizace, může a nemusí být oddělený od služby

ČVUT disponuje vlastním autorizačním serverem implementující protokol OAuth 2.0. Kromě možnosti autentizovat někoho z ČVUT, jej lze také využít k delegování přístupu do systému Usermap. Aplikaci je nutno nejprve registrovat pomocí AppsManageru, tedy webového portálu, kde lze registrovat dva druhy aplikací: serverovou a klientskou. V obou případech je zapotřebí definovat oprávnění poskytnutá prostřednictvím přístupového tokenu. Klientská aplikace se autorizačnímu serveru prokáže pomocí dvojice přístupových údajů: client_id a client_secret. (2)

Přístupový token má obvykle omezenou dobu platnosti (nejčastěji 1 hodinu). Abychom nemuseli od uživatele po vypršení platnosti požadovat opětovné přihlášení, často bývá s přístupovým tokenem odeslán i token obnovovací. Ten má obvykle mnohem delší dobu životnosti. Při vypršení nebo blížícím se konci životnosti přístupového tokenu si klientská aplikace požádá o nový přístupový token prostřednictví obnovovacího tokenu.

2.5.1 Zhodnocení možností autentizace

Výhoda OAuth2 oproti Shibboleth spočívá v jeho jednoduchosti. Pro využívající platformu stačí, aby disponovala implementací HTTP protokolu. Nevyžaduje žádné speciální nastavení webového serveru. Při startu webové aplikace stačí jen načíst dvojici přístupových údajů: client_id a client_secret. Právě z těchto důvodu jsem dal OAuth 2.0 přednost před Shibboleth v mé implementaci.

2.6 EXISTUJÍCÍ ŘEŠENÍ KNIHOVNÍCH SYSTÉMŮ

Tato podkapitola se věnuje srovnání existujících open-source řešení. Srovnávacími kritérii jsou právě dříve zmíněné požadavky na činnost knihovny. Hodnotí se, zda dané řešení požadavek přímo splňuje nebo zda umožňuje integraci s aplikací třetích stran pro možné řešení tohoto požadavku. Navíc se přihlíží i k vizuální podobě, dostupnosti na mobilní platformy a způsobu jakým se řešení uvede do provozu. Jsou srovnána 3 existující řešení.

2.6.1 Invenio

Moderní integrovaný systém pro knihovny, vybudovaný nad Invenio Digital Repository web frameworkem[7]. Serverová část byla vyvinuta pomocí technologií Python a Flask. Pro uchování dat jsou použity relační databáze PostgreSQL a MySQL. Podporu pro full-text vyhledávání zajišťuje služba Elasticsearch. Vykreslování uživatelského rozhraní zajišťuje React (3).

Výhody

Disponuje definovatelným schématem pro položky knihovny a lze tedy přidat další typ entity v systému zatím neobsažené. Moderní vzhled, vhodný i pro mobilní aplikace. Rozšiřitelnost a snadná integrace pomocí REST API. Podpora pro full-text vyhledávání pomocí služby Elasticsearch. Díky podpoře pro Docker kontejnery lze celek snadno spustit na jakémkoli serveru, kde je Docker nainstalován, případně spolu s ním spustit i další služby rozšiřující původní funckionalitu.

Nevýhody

Dokumentace se nikde nezmiňuje o samoobslužném provozu. Dle dokumentace je projekt stále ve vývoji. Nelze se autentizovat ani autorizovat prostřednictví třetích stran. Podpora pro identifikační karty není zdokumentována.

2.6.2 BiblioteQ

Jedná se o profesionální nástroj sloužící k archivaci, katalogizaci a správě knihovny. Uživatelské rozhraní se vykresluje pomocí Qt. K uchování dat jsou použity relační databáze PostgreSQL a SQLite. Získávání dat o knihách a časopisech zajišťují protokoly The Open Library, SRU, a Z39.50. Software je dostupný pro všechny systémy podporující rozhraní Qt a je distribuován pod BSD licencí (4).

Výhody

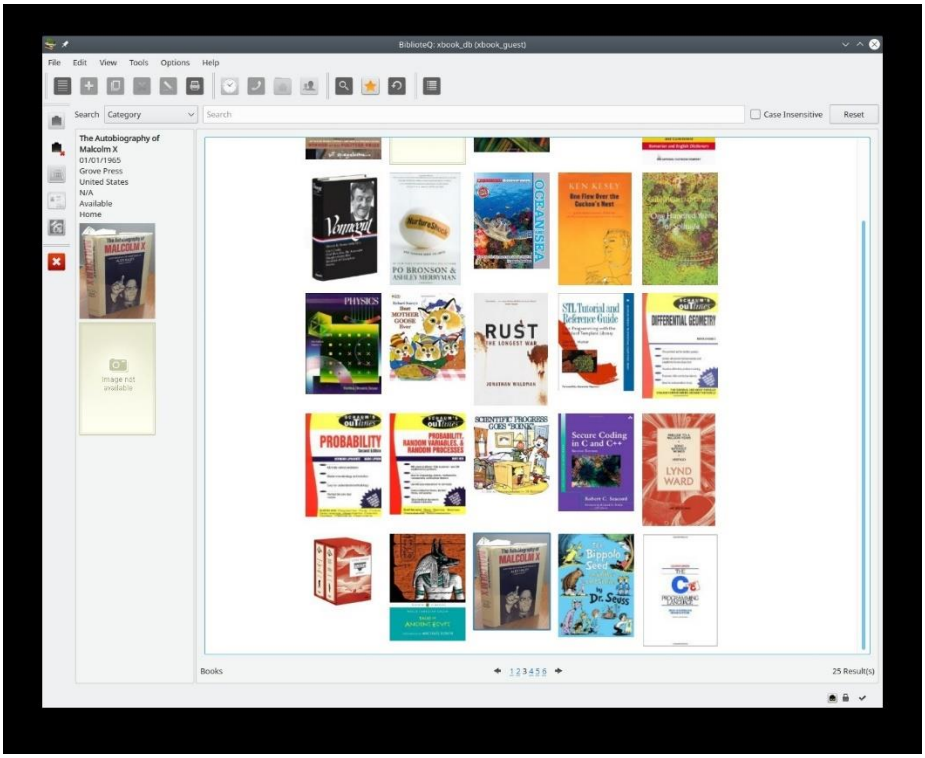
Podpora pro vyhledávání včetně vlastních SQL dotazů. Možnost zobrazovat některé publikace přímo ve webovém prohlížeči. Automatické stažení obálek pro knihy a časopisy pomocí služeb Amazon a Open Library.

Nevýhody

Aplikace disponuje starším uživatelským rozhraním, a tedy ne příliš vhodným pro mobilní zařízení (Obrázek 2: Příklad obrazovky knihovního systému BiblioteQ). Rovněž neposkytuje samoobslužný provoz. Nepodporuje všechny typy entit zmíněných v kapitole Systémové entity. Například u entit

[7] Softwarový balík usnadňující programování. Má větší rozsah než programová knihovna. Typicky zapouzdřuje a řeší komplexní problém.

„Časopis“ a „Sborník“ chybí potřebná pole. Další nevýhodou je nutnost manuální instalace na cílový stroj.



Obrázek 2: Příklad obrazovky knihovního systému BiblioteQ

2.6.3 Evergreen

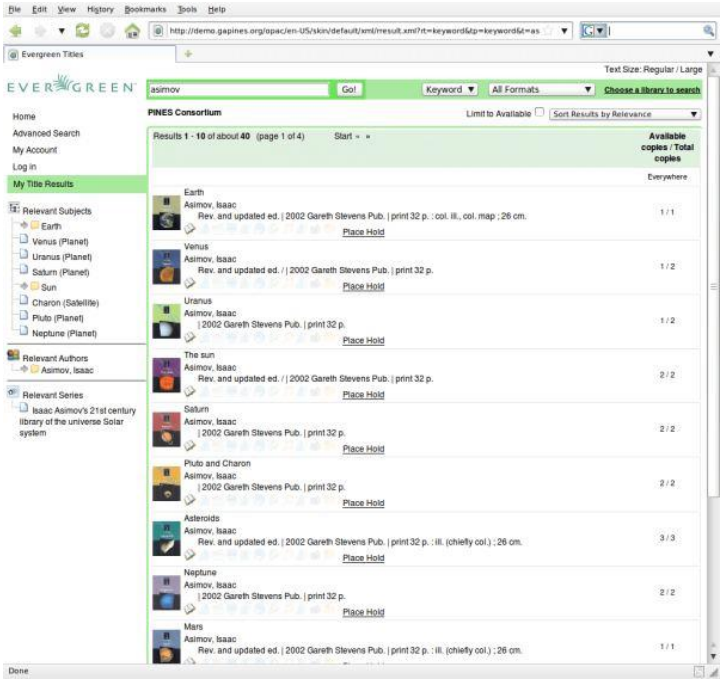
Evergreen je licencován pod licencí GNU (General Public License). Má aktivní uživatelskou a vývojářskou komunitu. Zároveň několik soukromých firem nabízí migraci z existujícího řešení, podporu a rozšíření. Systém byl poprvé spuštěn v roce 2006 a od té doby jej používá více než 544 knihoven různých typů. Patří sem veřejné, akademické i domácí instalace / využití (5).

Výhody

Aplikace dokáže vyřešit všechny základní požadavky na práci s knihovnou. Dokonce disponuje i automatickým načítání čárkových kódů pro usnadnění procesu zapůjčení a vrácení. Lze vystavovat i karty pro návštěvníky pro jejich pozdější autentizaci, nicméně pouze s čárkovým kódem. Disponuje i částečným samoobslužným provozem, avšak ne v takové míře, jak jej definuje náš funkční požadavek (Samoobslužný provoz).

Nevýhody

Aplikace má poněkud zastaralý design, a tedy ne zrovna vhodný pro mobilní zařízení (Obrázek 3: Příklad obrazovky knihovního systému Evergreen). Zároveň nemá žádné rozhraní umožňující integraci s poskytovatelem identity nebo s nějakou další službou. Nelze používat přístupové karty s technologií NFC. Pro uvedení do provozu je potřeba nastavit ručně několik aplikací, a i klientské aplikace pro knihovníky je třeba instalovat.



Obrázek 3: Příklad obrazovky knihovního systému Evergreen

2.6.4 Zhodnocení existujících řešení

Ze všech zmíněných tří existujících řešení je nejvíce perspektivní Invenio. Sice nedisponuje samoobslužným provozem, ale díky své serverové části obsahující REST API, jej lze napojit na jiné části / aplikace a tím docílit potřebných rozšíření. Zároveň díky kontejnerizaci všech částí jej lze bez problému spustit spolu i s další službou umožňující například samoobslužný provoz knihovny. Invenio by tak v tomto scénáři figurovalo jako databáze knih, zajišťovalo by full-text vyhledávání a katalogizaci. Nicméně rozhodl jsem se jej nepoužít zejména z důvodů stále probíhajícího vývoje a nutnosti existence dvou přihlašovacích údajů pro knihovníky. Existujícím účtem ČVUT a přihlašovacími údaji do aplikace Invenio.

2.7 DALŠÍ SYSTÉMY INTEGROVANÉ DO ZDE VYTVÁŘENÉHO ŘEŠENÍ

Podle analýzy požadavků na činnost systému bude zde vytvářený informační systém uchovávat jenom některá data. Ostatní data musí získat prostřednictvím integrace s dalšími systémy. Je tomu tak v případě autentizace uživatelů pomocí jejich existujících ČVUT účtů, dohledání kontaktních údajů jednotlivých uživatelů, získávání dat o knize pomocí zadaného ISBN/ISSN nebo prohledávání katalogu Ústřední knihovny ČVUT. Další nutnou integrací je napojení na emailový server pro možnost odesílat emaily z domény **cvut.cz** nebo z vhodných sub-domén.

2.7.1 Zuul – Autorizační server pro instituci ČVUT

ČVUT vyvíjí vlastní OAuth 2.0 autorizační server (dále jen OAAS) pod názvem Zuul OAAS. Je open-source a repositář se zdrojovými kódy se nachází na adrese **github.com/cvut/zuul-oaas**. Přestože je umístěn na webové adrese **auth.fit.cvut.cz** (adresa obsahuje subdoménu Fakulty informačních technologií ČVUT) může ho využít kdokoli z akademické obce ČVUT (tedy nejen FIT).

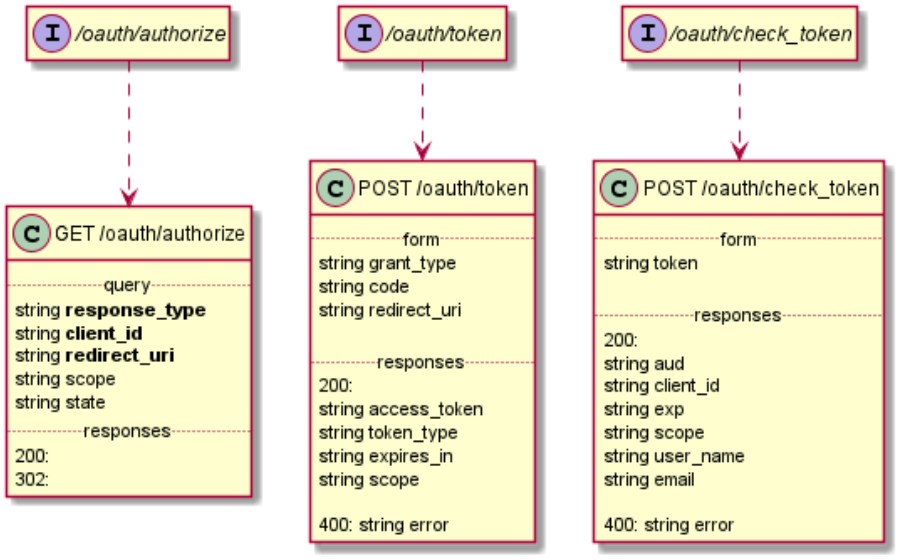
Jsou podporovány celkem tři autorizační granty z protokolu OAuth 2.0: authorization code, refresh token a client credentials. Aby je bylo možné využít v rámci vlastní aplikace, musí vývojář nejprve

svoji aplikaci registrovat pomocí registračního portálu s názvem AppsManager, nacházejícího se na adrese **auth.fit.cvut.cz/manager**. Nejdříve je nutné zaregistrovat projekt, následně zvolit oprávnění, k jednotlivým službám a až poté lze registrovat dva typy aplikací. Po registraci získáme dvojici **client_id** a **client_secret** (zjednodušeně jméno a heslo aplikace).

Prvním typem je „webová aplikace“, což v terminologii protokolu OAuth 2.0 odpovídá autorizačním grantům **authorization code** a **refresh token**. Používá se v případech nutnosti interaktivního přihlášení uživatele. U registrace je nutné uvést URL adresu k přesměrování. Tato adresa je použita k přesměrování po úspěšném přihlášení uživatele na webové stránce autorizačního serveru. Při pokusu o přihlášení je nejprve uživatel přesměrován na webovou stránku autorizačního serveru. Následně vyplní svoje přihlašovací údaje. Pokud jsou správné, autorizační server přesměruje uživatele zpět na URL adresu uvedenou při registraci aplikace. Do URL adresy je navíc přidán i autorizační kód. Pomocí autorizačního kód a dvojice **client_id** a **client_secret** je nyní aplikace schopna získat autentizační a obnovovací token uživatele.

Druhým typem aplikace je „servisní účet“, což v terminologii protokolu OAuth 2.0 odpovídá autorizačnímu grantu **client credentials**. V tomto případě v autentizačním procesu uživatel nijak nefiguruje. Aplikace získá přístupový token pouze za pomocí dvojice **client_id** a **client_secret**. Token má omezenou platnost a aplikace musí před jeho vypršením požádat o nový.

Rozhraní obsahuje tři koncové body (Obrázek 4: Tři koncové body v rozhraní Zuul). První koncový bod na adrese „/oauth/authorize“ slouží k přesměrování uživatele na webovou stránku autorizačního serveru při použití autorizačního grantu **authorization code**. Druhý koncový bod k získání tokenu se nachází na adrese „/oauth/token“. Některé autorizační servery ukládají veškeré informace o tokenu přímo do jeho těla a opatří jej digitálním podpisem. Token tak lze snadno ověřit, aniž by bylo nutné autorizační server opět volat. V tomto případě tomu však takto není. V odpovědi od OASS se nachází pouze identifikátor tokenu. Aktuální stav tokenu lze zjistit zavoláním třetího koncového bodu na adrese „/oauth/check_token“.



Obrázek 4: Tři koncové body v rozhraní Zuul

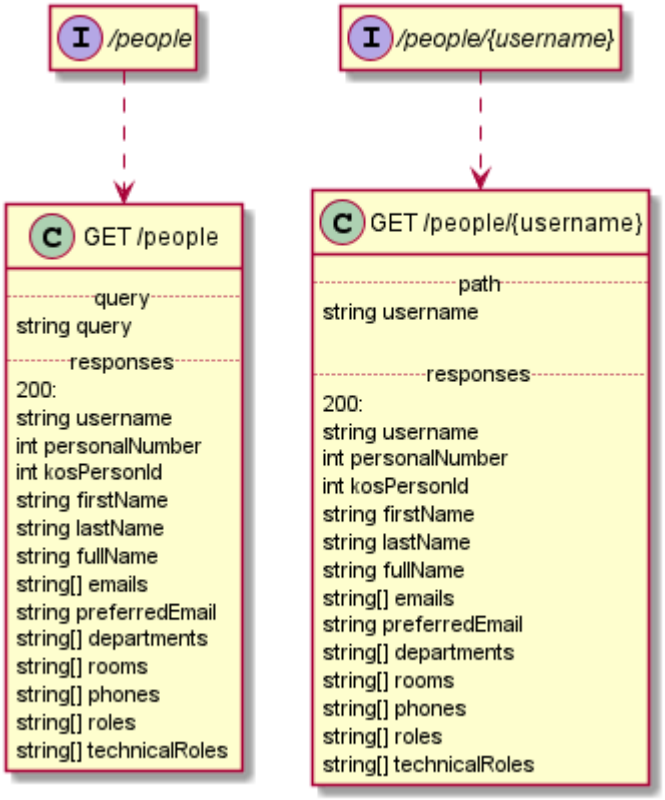
V rozhraní nicméně chybí koncový bod sloužící k odhlášení uživatele. Situace se navíc značně zhoršuje tím, jak si OASS pamatuje jeho předchozí relaci. Děje-li se tak na vlastních zařízeních uživatele, není to takový problém. Nicméně například u kiosku v knihovně, by mohlo dojít k situaci,

kdy se uživatel z kiosku odhlásí, avšak zůstane mu aktivní relace na autorizačním serveru. Uživatel, který použije kiosk bezprostředně po něm, by se tak mohl jeho jménem opět přihlásit. Tomu může kiosk zabránit například kompletním resetováním stavu webového prohlížeče použitého původně k přihlášení.

2.7.2 Usermap – Systém pro evidenci a správu uživatelů v rámci ČVUT

Kontaktní údaje uživatelů OASS neposkytuje. K získání údajů jako je jméno, příjmení nebo email je zapotřebí napojit se na systém Usermap. Jedná se o aplikaci pro správu vybraných osobních údajů, hlavní kontaktní adresář na ČVUT a nástroj pro správu uživatelských rolí většiny systémů IS ČVUT. Je dostupný pro studenty, pracovníky, částečně partnery i veřejnost. Základní údaje poskytuje anonymně, ale pro plnou funkcionalitu je nutné se autentizovat pomocí uživatelského jména a hesla ČVUT. (6)

Kromě webové aplikace, existuje i REST API rozhraní, agregátor údajů o identitách lidí na ČVUT, s pracovním názvem Usermap API. Primárním zdrojem dat je Usermap (SSU), konkrétně Usermap LDAP, jeho relační přívěšek EXT_UMAP a KOS (pro předmětové role). (7)



Obrázek 5: Rozhraní správy osobních údajů USERMAP API

Poskytuje dva koncové body využitelné k vyhledávání osob (Obrázek 5: Rozhraní správy osobních údajů USERMAP API). Oba podporují dotazy v jazyce RSQL, což značně usnadňuje vyhledávání osob. Lze například zadat dotaz ve tvaru „query=name== příjmení jméno titul “. Zároveň lze informace o uživateli získat i na základě jeho uživatelského jména (loginu).

Aby směla aplikace k datům z API přistupovat, musí mít v AppsManager portálu povolená oprávnění typu **urn:ctu:oauth:umapi.read** a **cvut:umapi:read**. K datům lze přistupovat při použití libovolného autorizačního grantu (**authorization code, client credentials**).

Z analýzy těchto dvou systémů vyplývá nutnost vykonat pro získání úplných údajů o uživateli po jeho přihlášení celkem tři http volání. Z autorizačního serveru vyzvednout autentizační token (první) a získat data o tokenu a uživatelské jméno (druhé). Poté pomocí uživatelského jména dohledat informace o uživateli ze služby Usermap API (třetí). Pokud by aplikace tyto kroky vykonávala při každé akci uživatele, značně by ji to zpomalilo, nemluvě o zátěži pro výše zmíněné systémy. Při implementaci bude vhodné tato data držet v mezi paměti, jelikož k jejich změně nedochází tak často.

2.7.3 Google Books – Poskytování základních informací o knihách

Pole evidenční číslo oddělení, systémové číslo, nebo evidenční číslo FEL na entitě kniha jsou specifická pro procesy v katederní knihovně. Zatímco například název knihy, autor nebo i klíčová slova se dají snadno online dohledat. Google Books je vyhledávací služba od společnosti Google, zpřístupňující naskenované (digitalizované) knihy z různých knihoven. Kromě toho umožňuje knihy vyhledávat podle mnoha atributů. Jedním z nich je i ISBN, což je označení pro mezinárodní standardní číslo knihy.

Služba poskytuje REST API rozhraní. Lze se autentizovat pomocí protokolu OAuth 2.0, tentokrát ale ne vůči autorizačnímu serveru ČVUT, ale vůči autorizačnímu serveru společnosti Google. Další možností je využít statický API klíč načtený aplikací například z konfiguračního souboru. Nicméně požadavky na veřejná data žádnou autentizaci nevyžadují. Jedinou limitací je počet požadavků, které je služba ochotna za den odbavit. Při použití API klíče je horní hranice 1000 požadavků za den. Omezení pro anonymní použití uvedeno není, ale lze jej očekávat ještě nižší. Z tohoto důvod, stejně jako v případě uživatelských dat, se vyplatí data udržovat v mezi paměti.

Pro vyhledávání knihy pomocí ISBN stačí zavolat koncový bod na adrese „/books/v1/volumes“ a do parametru dotazu předat hodnotu ISBN. Komunikaci jsem záměrně nezdokumentoval pomocí diagramu, jako tomu bylo v předchozích případech, neboť služba neodpovídá na všechny dotazy stejně. U některých publikací nejsou některá pole vůbec přítomna. Naproti tomu název knihy, autor a klíčová slova se v odpovědi nachází téměř vždy. (8)

2.7.4 Ústřední knihovna ČVUT

Ne všechny knihy má samozřejmě (lokální) katederní knihovna k dispozici. Může se tedy při vyhledávání stát situace, kdy nebude uživatel úspěšný. V takovém případě by bylo užitečné pokusit se tentýž dotaz zopakovat, ale tentokrát oproti databázi Ústřední knihovny ČVUT. Katalog knihovny disponuje informacemi o knihách, časopisech, e-knihách, e-časopisech a vysokoškolských kvalifikačních pracích obhájených na ČVUT obsažených ve svém fondu. Na stránkách knihovny není žádná zmínka o rozhraní použitelném z jiné aplikace, nicméně pro tento případ užití by to ani nebylo vhodné. Jednak bude mít zde vytvářený informační systém jiný datový model, a navíc by si uživatel stejně knihu prostřednictvím systému nedokázal přímo půjčit. Kromě toho vyhledávač knihovny je poměrně komplexní a replikovat celou jeho funkcionalitu by bylo složité. Namísto toho je ale možnost, vložit vyhledávací dotaz do parametrů URL adresy. Jakmile uživatel stránku otevře v prohlížeči, zobrazí se mu výsledky. (9)

Vyhledávač Summon

Jde o vyhledávač nad všemi zdroji dostupnými na ČVUT. Hledání probíhá v: katalogu knihovny, odborných databázích, elektronických knihovnách, institucionálním repositáři ČVUT, volně

dostupných informačních zdrojích. Lze se dostat i k plným textům článků a knih, pokud pro ně má ČVUT předplacený přístup. Do vyhledávacího pole lze vložit klíčová slova i spolu s Booleovskými operátory. Vyhledávání lze rozšířit na několik polí nebo se specificky zaměřit jen na jedno z nich. Stejně jako v předchozím případě, i zde je možné vložit vyhledávací dotaz přímo do parametrů URL. (10)

2.7.5 Poštovní server ČVUT

Informační systém bude pro svůj chod potřebovat odesílat emaily z domény **cvut.cz**. K tomu je zapotřebí existence emailové schránky a přístupové údaje k poštovnímu serveru ČVUT. Naopak aplikace už nemusí emaily umět přijímat. Odesílání emailů se standardně uskuteční pomocí protokolu SMTP. Ten pracuje nad protokolem TCP a obvykle se komunikuje přes port 25. Vyskytují se zde dva aktéři: poštovní klient (C) a poštovní server (S). Příkladem poštovního klienta může být například aplikace Microsoft Outlook nebo Mozilla Thunderbird. V tomto případě je v roli poštovního klienta právě zde vytvářený informační systém. Celková zpráva obsahuje hlavičku a tělo zprávy případně volitelné přílohy s libovolným obsahem. (11)

Ukázka komunikace

HELO localhost
MAIL FROM:<system@example.com>
RCPT TO:<test@example.com>
DATA
From: [IES] <system@example.com>
To: <test@example.com>
Date: Fri, 10 Oct 2021 09:00:00 +0100
Subject: Předmět zprávy

Text emailu.
.
QUIT

Novější servery používají ESMTP a namísto příkazu HELO se odesílá příkaz EHLO. Nicméně většina dnešních jazyků poskytuje pro práci s emaily programové knihovny nebo existují programové knihovny třetích stran, a proto není nutné syntaxi protokolu detailně znát. Vůči poštovnímu serveru je však nutné se autentizovat. Oficiální dokumentace postupu v případě ČVUT se mi nepodařilo najít. Rozhodl jsem se tedy kontaktovat IT administrátora katedry kybernetiky Ing. Daniela Večerku. Ten mi umožnil přidání IP adresy serveru, kde se informační systém bude nacházet, do seznamu povolených IP adres. Následně se tedy stačí připojit na SMTP relay na adrese **relay.felk.cvut.cz**.

2.7.6 SendGrid – Cloudová služba marketingové komunikace

Služba SendGrid nebude při běžném provozu používána. Poskytuje cloudovou službu umožňující marketingovou komunikaci nejen prostřednictvím emailového kanálu. Význam této integrace spočívá v testování odesílání emailů. Při vývoji aplikace a ladění jednotlivých funkcionalit není vhodné využívat oficiální emailové adresa nebo servery ČVUT. Nebudou tak zbytečně zatěžovány, a navíc monotónním obsahem a repetitivním opakováním některých emailů by je některý SPAM filtr mohl omylem vyhodnotit jako SPAM. Po nasazení do produkčního prostředí by mohlo dojít k zablokování emailové zprávy pro některé uživatele.

I SendGrid umožňuje odesílat emaily přes SMTP, nicméně autoři služby namísto toho doporučují používat jejich REST API rozhraní (12). Služba pracuje asynchronně, tedy při jejím provolání nás informuje o obdržení požadavku, uloží si jej do fronty a poté se vše vykoná na pozadí. Aplikace tak nemusí čekat na dokončení odesílání emailu, ale může okamžitě pokračovat ve své činnosti. Pro

některé programovací jazyky existuje dokonce oficiální programová knihovna, není tedy nutné volat rozhraní na přímo, ale je možné tak učinit prostřednictvím abstrakce.

2.7.7 Aplikační rozhraní prodejců knih

Ne mnoho prodejců knih takováto rozhraní poskytuje. Vývoj a údržba takového rozhraní je nákladná a mnohdy by obojí často obstarávala externí firma. Dalším problémem je nejednotnost datového modelu. Každý prodejce by logicky vystavoval datový model, nevíce se blížící fyzické podobě dat v jeho databázi. Informační systém by tedy musel rozumět každému datovému modelu každého z prodejců. S postupem času se navíc rozhraní vyvíjí, některé koncové body jsou nejprve označeny za zastaralé nebo přestanou zcela pracovat. Bylo by tedy nutné nějakým způsobem konzistenci integrovaných rozhraní monitorovat a reagovat na případné změny.

Během hledání mezi existujícími prodejci knih jsem našel jednu službu poskytující REST API rozhraní. Nese název TrhKnih.cz a jedná se o antikvariát, kde si v podstatě čtenáři prodávají knihy navzájem. Rozhraní je dobře zdokumentováno na webové adrese **api.trhknih.cz/v1**. Je dostupné pouze na protokolu HTTPS a přístup k němu mají jen partneři portálu TrhKnih.cz po dohodě s provozovatelem. Je zabezpečeno protokolem OAuth 2.0 a pro získání přístupového tokenu je zapotřebí použít autorizační grant **client credentials**. Rozhraní umožňuje vyhledávat pomocí textového řetězce a případně i roku vydání. V odpovědi na vyhledávací dotaz se nachází název, podtitul a autoři.

V rámci zde vytvářeného řešení jsem se tedy rozhodl neintegrovat žádné rozhraní prodejců knih. Bylo by velmi náročné zachovávat kompatibilitu a vytvořenou integraci by musel někdo stále udržovat. Zároveň by to pro uživatele nemělo velký přínos. Typickým cílem při návštěvě knihovny je i okamžité zapůjčení knihy po nějakou dobu. Zřídkakdy má návštěvník knihovny zájem cokoliv (okamžitě) nakupovat, zatímco prodejci knih málokdy cokoli půjčí.

2.8 NFC ŠTÍTKY A QR KÓDY

Na vybrané publikace budou umístěny NFC štítky nebo obrázky s QR kódy. I přístupové karty pro uživatele budou využívat technologii NFC. Jsou k dispozici dva typy NFC štítků. První typ umožňuje pouze čtení: Hodnota na štítku je zapsána přímo od výrobce a nejde ji změnit. Jedná se o výrobní neboli sériové číslo o délce nejčastěji 7 bytů. V systému se pak musí toto číslo k publikaci přiřadit.

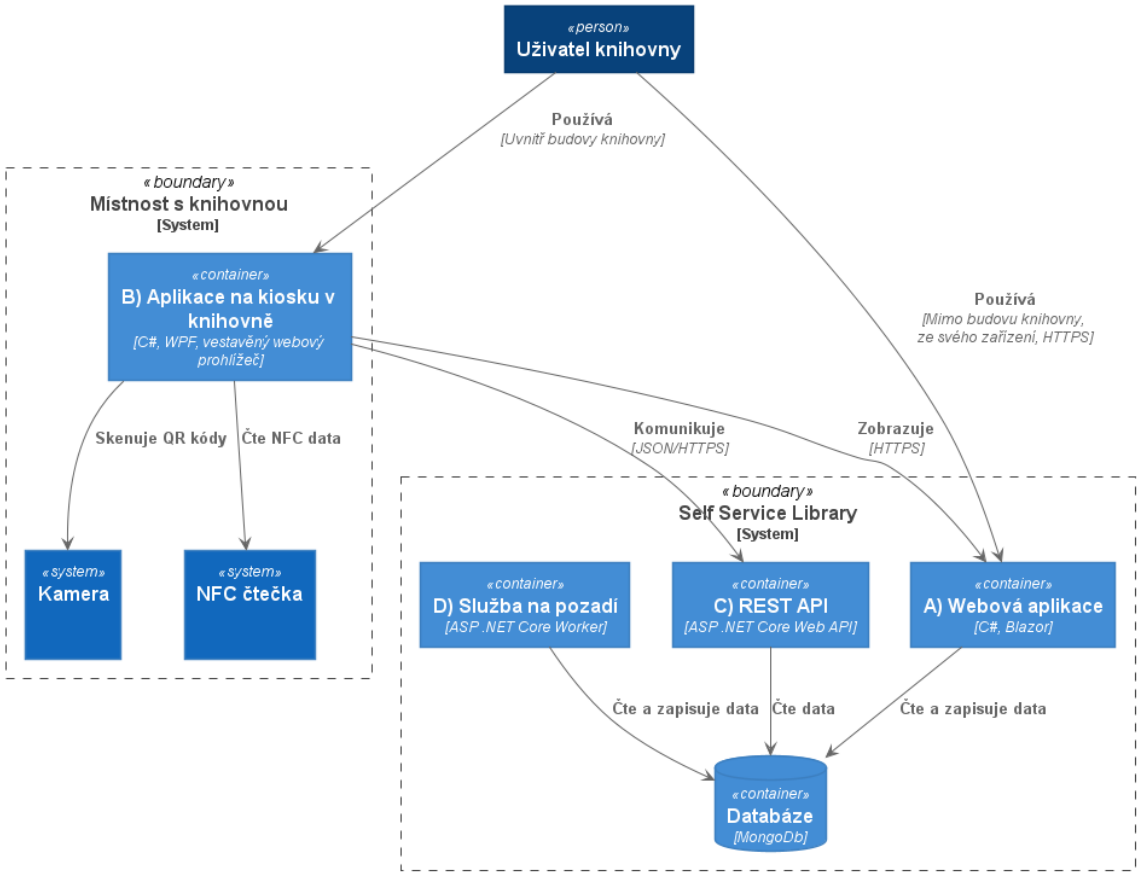
Na druhý typ štítku lze při použití vhodného hardwaru zapsat i vlastní data. Kromě výrobního čísla lze tedy zapsat i informace v tzv. NDEF formátu. Ta může být ve formě krátkého textu obsahujícího evidenční číslo oddělení publikace, nebo rovnou celou URL adresu odkazující na stránku s detailem knihy přímo v knihovním informačním systému. Nevýhodou je o něco vyšší cena a při velkém inventáři by se tak mohlo jejich používání poněkud prodražit.

Použití QR kódu je mnohem snazší. Do obrázku (složeného z množství vhodně uspořádaných bodů) lze rovněž snadno vložit celou URL adresu odkazující přímo na stránku s detailem knihy. K přečtení není potřeba žádný speciální hardware, uživatelům stačí pouze chytrý telefon s kamerou a vhodnou aplikací. Většina dnešních chytrých telefonů má aplikaci pro čtení QR kódů již často nainstalovanou. Nevýhodou QR obrázků může být snadná možnost jejich umazání či dokonce (mechanického) poškození.

3 NÁVRH ŘEŠENÍ

Celé řešení jsem rozdělil na 4 dílčí aplikace a znázornil na diagramu (Vývojový diagram 8: Celkový kontext vytvářeného knihovního systému):

- A) Webová aplikace – Hlavní část představující webové rozhraní. Zpracovává vstupy uživatele a poskytuje požadovaná data. Zobrazuje se na zařízeních uživatele (počítač, telefon, tablet), případně na kiosku v knihovně.
- B) Desktopová aplikace na kiosku – Zobrazuje webovou aplikaci (A) ve webovém prohlížeči. Komunikuje s připojenou NFC čtečkou a kamerou pro skenování QR kódů.
- C) REST API služba – Umožňuje kiosku vyhledávat knihy podle NFC štítků a autentizovat uživatele pomocí přístupové NFC karty.
- D) Služba na pozadí – Upozorňuje uživatele na vrácení knihy.



Vývojový diagram 8: Celkový kontext vytvářeného knihovního systému

Hlavní komponentou je webová aplikace (A) umístěná na serveru ve školní infrastruktuře. Uživatel k ní může přistoupit buď přímo ze svého zařízení (osobní počítač / mobilní telefon), nebo prostřednictvím kiosku (lokální počítač) v místě knihovny. Umožní prohlížení obsahu, jeho vyhledávání a knihovníkům dovolí editovat knihy nebo je půjčovat a samozřejmě vracet. Uživatelé s dostatečnými právy si budou moci prostřednictvím webu také knihu půjčit nebo vrátit sami.

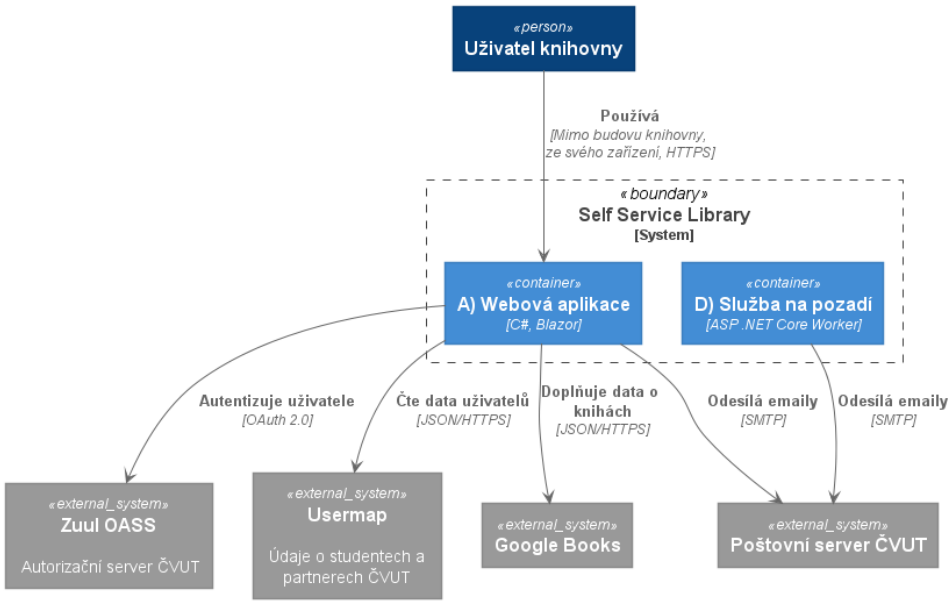
Desktopová aplikace (B) na kiosku v knihovně bude mít za úkol zobrazit tu samou webovou aplikaci ve svém vestavěném webovém prohlížeči. Kromě toho bude umět obsluhovat k ní připojenou NFC čtečku. Díky tomu mohou návštěvníci knihovny po přiložení knihy opatřené NFC štítkem, zobrazit webovou stránku na které se o knize dozví další informace. Pokud jsou přihlášení a mají potřebná

práva, mohou si ji dokonce okamžitě snadno zapůjčit. Lze se přihlásit standardně přes autorizační server ČVUT (jméno a heslo) nebo NFC kartou. Kartu je nutné nejprve registrovat ve webové aplikaci, případně si k ní i nastavit pin.

Ke své činnosti potřebuje mít knihovna možnost vyhledávat knihy podle NFC štítku a autentizovat uživatele pomocí NFC karty. K tomu slouží API služba (C), v podstatě samostatná webová aplikace, poskytující REST rozhraní. Nachází se na stejném serveru jako webová aplikace a je připojena ke stejné databázi. V komunikaci se chová jako mezivrstva mezi aplikací na kiosku a databází. Přímá komunikace mezi kioskem a databází není umožněna z bezpečnostních důvodů.

Systém potřebuje některé úkony vykonávat na pozadí v periodických intervalech. Tuto funkcionalitu zajišťuje služba na pozadí (D). Tato služba se nachází také na stejném serveru jako dvě předchozí komponenty a je rovněž připojena ke stejné databázi. Během své činnosti nalezne všechny výpůjčky, u nichž se blíží datum vrácení nebo již bylo překročeno a upozorní emailem příslušnou osobu.

S externími systémy, popsanými v kapitole Další systémy, komunikuje pouze webová aplikace a služba na pozadí. Integrace se Zuul OASS a Usermap je nezbytná k autentizaci uživatelů pomocí uživatelského jména a hesla ČVUT. Rozdělení komunikace mezi externími systémy a zde vytvářenými aplikacemi je znázorněno na diagramu (Vývojový diagram 9: Kontext integrace).



Vývojový diagram 9: Kontext integrace systému s externími komponenty

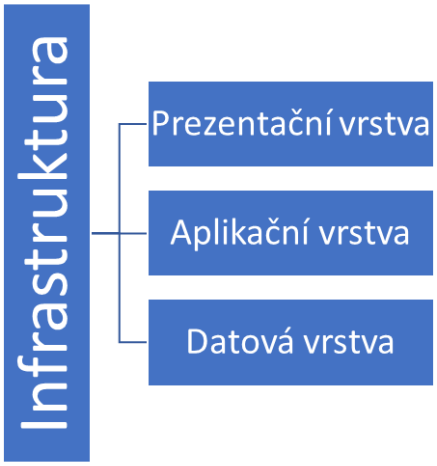
Jednotlivé aplikace jsem se rozhodl vytvořit za pomocí .NET a jazyka C#. Jedná se o bezplatný open-source vývojový framework od společnosti Microsoft. Lze vyvíjet nejrůznější druhy aplikací a rovněž cílit na několik platforem. V současnosti jsou podporované 3 verze. Verze s označením LTS mají dlouhodobější podporu (Tabulka 2 Přehled verzí .NET).

Verze frameworku .NET	Datum vydání	Datum konce podpory
.NET 5.0	2021-03-09	-
.NET Core 3.1 (LTS)	2021-03-09	2022-12-03
.NET Core 2.1	2021-03-09	2021-08-21

Tabulka 2 Přehled verzí .NET
převzato z https://versionsof.net

Pro vývoj jsem se rozhodl použít verzi .NET 5. Nedisponuje sice dlouhodobou podporu, nicméně ohlášená verze .NET 6 dlouhodobou podporu mít bude. Od verze 5 se navíc v technologii Blazor zkrátil čas renderování vynecháním prázdných uzlů v HTML dokumentu a optimalizací struktury RenderTreeFrame. Kromě toho lze s verzí 5 využívat nejnovější verzi jazyka C#, a tedy celkově těžit z přínosů této nové verze.

Při návrhu jsem se rozhodl dodržovat 3 vrstvou softwarovou architekturu. Jak název napovídá, rozděluje celou problematiku do tří vrstev. Každá z nich řeší určitou část problému a smí komunikovat výhradně se sousední vrstvou, a to prostřednictvím předem definovaných rozhraní (Obrázek 6: 3 vrstvá softwarová architektura).



Obrázek 6: 3 vrstvá softwarová architektura

Nejspodnější (datová) vrstva obstarává načítání a ukládání dat. Komunikuje přímo z databází a zajišťuje mapování z databázového modelu na systémové entity. Tím umožňuje aplikační vrstvě pracovat s daty bez znalosti jejich fyzické podoby.

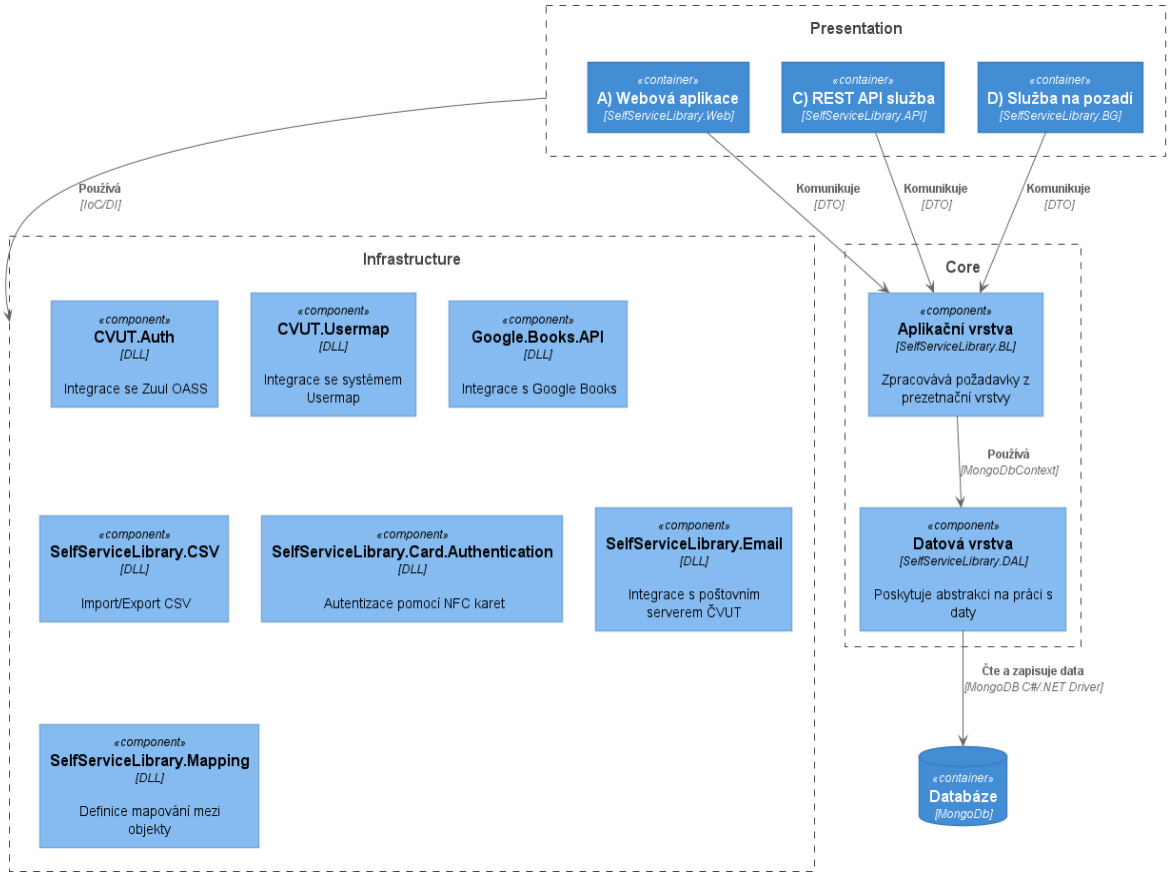
Aplikační vrstva obsahuje veškerou logiku, výpočty a procesy pro zpracování dat. Ke své činnosti využívá datovou vrstvu a prezentační vrstvě poskytuje množinu rozhraní. U každého požadavku uskuteční validaci autorizačních pravidel a při neúspěchu vyhodí výjimku. Neměla by být závislá na konkrétních třídách (implementacích) poskytnuté infrastrukturou Pokud ke své činnosti například potřebuje odeslat email, definuje rozhraní pro notifikaci uživatelů, a to je následně implementováno v jiné části kódu.

Prezentační vrstva zobrazuje informace uživateli, většinou formou grafického rozhraní nebo webové aplikace. Umožňuje uživateli zadávat vstupy a následně je validuje, ale neobsahuje ani nevykonává zpracování dat. Konzumuje rozhraní poskytované aplikační vrstvou.

Do infrastruktury patří vše, co přímo nesouvisí s aplikační logikou, avšak je nezbytné pro činnost celé aplikace. Aplikační vrstva ke své činnosti často potřebuje rozhraní, jež není sama schopna implementovat. Například rozhraní pro notifikaci uživatelů může být realizováno pomocí odesílání emailu a protokolu SMTP. Tento implementační detail je ponechán v infrastruktuře. Pokud se v budoucnu místo emailu začne používat například SMS, aplikační vrstvu není nutné nijak měnit. (12)

Během implementace jsem použil vývojové prostředí Microsoft Visual Studio Community (volně dostupné). V rozložení projektu jsem zohlednil výše zmíněnou architekturu a k lepší organizaci jsem

použil složkovou strukturu. Rozmístění a jednotlivé závislosti komponent jsou znázorněny na diagramu (Vývojový diagram 10: Diagram rozmístění a závislostí jednotlivých komponent). Nejdůležitější část kódu je umístěna ve složce **Core**, patří sem aplikační a datová vrstva. Ve složce **Presentation** se nachází prezentační vrstva. Tvoří ji 3 aplikace: REST API služba používaná kioskem, webová aplikace navštěvovaná uživateli a služba vykonávající akce na pozadí. V poslední složce **Infrastructure** se nachází zbytek kódu zajišťující integraci s externími systémy případně programovými knihovnami třetích stran, a tím poskytující implementaci některých rozhraní definovaných aplikační vrstvou. Složková struktura i názvy jednotlivých komponent jsou v anglickém jazyce. Stejně je tomu tak i v případě zdrojového kódu. Jedinou výjimkou jsou komentáře v datové vrstvě vysvětlující význam jednotlivých atributů entit a poskytující i jejich český překlad.



Vývojový diagram 10: Diagram rozmístění a závislostí jednotlivých komponent

Propojení definovaných rozhraní s jejich implementacemi je dosaženo pomocí návrhového vzoru s názvem Vkládání závislostí (anglicky známější Dependency injection). Jedná se o techniku v objektově orientovaném programování umožňující jednotlivým komponentám mezi sebou komunikovat, aniž by na sebe měly v době sestavení programu přímou referenci. Při startu aplikace je vytvořeno zobrazení z množiny rozhraní do množiny implementací. Parametrem zobrazení je i životnost. Pokud nějaká komponenta rozhraní konzumuje, nemusí se tak starat o konkrétní implementaci, ani ovlivňovat její životní cyklus.

3.1 DATABÁZE KNIHOVNÍHO SYSTÉMU

Každý informační systém potřebuje nějakým způsobem uchovávat data. Způsob uložení a forma závisí na jejich typu, objemu, nutnosti zachování silné konzistence, frekvenci zápisu a rovněž frekvenci čtení. Systém zajišťující funkci katederní knihovny bude evidentně při své činnosti odbavovat mnohem více čtení než zápisů. Uživatelé si prohlédnou mnohem více knih, než kolik si jich skutečně půjčí. Další poměrně častou operací bude full-text vyhledávání v databázi. Kromě toho je zapotřebí udržovat data konzistentní. Tedy není možné, aby se kniha prezentovala jako zapůjčená, ale při tom by sytém nevěděl, kdo si ji zapůjčil.

Z analýzy systémových entit je patrný stav knihovny, kde se nachází vícero typů položek nemající stejnou množinu atributů (Vývojový diagram 1: Navržené entity). Při uvedení do provozu se bude navíc databáze plnit již existujícími daty. Jejich zdrojem bude CSV soubor vzniklý exportováním všech záznamů z původního knihovního systému katedry. Při sběru a modelování požadavků na činnost systému mě vedoucí práce opakovaně upozorňoval na budoucí možnost změny datového schématu, v návaznosti na průběh exportování dat z právě používaného systém. Lze tedy očekávat možnost vkládání nových, a naopak rušení některých původních polí.

Při implementaci informačních systému bývá konvenčním řešením použití relačních databází. Data se ukládají do tabulek a ty jsou mezi sebou vhodně provázány. Dotazování pak probíhá pomocí jazyka SQL. Lze se dotazovat i napříč větším množstvím tabulek. Každá tabulka může odkazovat na řádek v jiné tabulce pomocí tzv. cizího klíče. Tato operace se nazývá JOIN a lze ji v rámci jednoho dotazu použít opakovaně. Nespornou výhodou je schopnost zajištění silné konzistence napříč daty. Toho je dosaženo pomocí transakcí, tedy série kroků vykonaných oproti databázi. Každá transakce je vykonána atomicky, to znamená nutnost úspěšného proběhnutí všech jejích kroků. V opačném případě se data dostanou do původního stavu, v jakém byla, než transakce započala. Naproti tomu nevýhodou je nutnost pevně definovaného schématu pro každou tabulku vynucené napříč všemi řádky v dané tabulce.

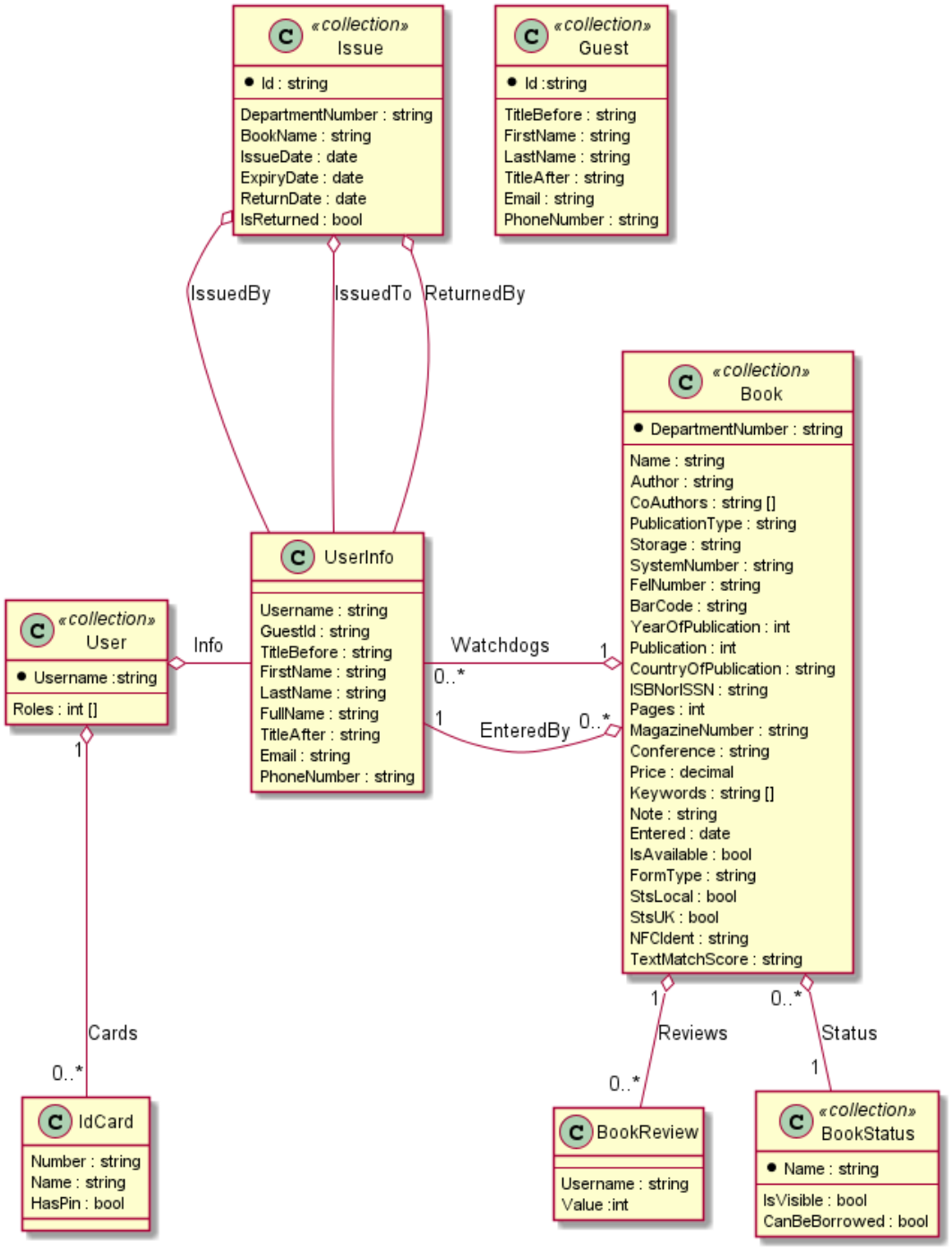
Alternativou jsou dokumentové databáze. Na rozdíl od předchozího typu databáze se data neukládají do tabulek, ale mají podobu strukturovaných dokumentů a jednotlivé položky dokumentu nemusí být skalární hodnoty. Kromě textu nebo čísla lze například uložit i pole hodnot nebo další dokument. Zástupcem dokumentové databáze je MongoDB. Dokumenty jsou ukládány do kolekcí a každý dokument může mít svoje vlastní schéma. V rámci jednoho dotazu nelze spojit data z více kolekcí. Sice existuje operátor **$lookup**, nicméně ten dokáže připojit pouze jednu kolekci, nelze jej tedy použít opakovaně. Nad jedním dokumentem lze vykonat několik různých operací najednou a všechny se provedou atomicky. Od verze 4.0 je dokonce možné provést atomickou operaci nad vícero dokumenty najednou.

Při implementaci jsem se tedy rozhodl použít technologii MongoDB umožňující snazší evoluci schématu dat. Jestliže se během implementace objeví potřeba přidat nová pole u některých položek, bude možné tak učinit, aniž by se předchozí data musela zásadně měnit. Zároveň dokáže zajistit konzistenci pro kritické procesy, jako jsou vypůjčení a vrácení knihy.

Při pokusu o zapůjčení knihy si aplikační vrstva nejprve v paměti vytvoří dokument výpůjčky. Následně definuje operaci úpravu dokumentu entity kniha takovým způsobem, aby byla kniha označena jako zapůjčená a byla k ní přiřazena právě vytvořená výpůjčka. Jelikož se jedná o dvě operace nad stejným dokumentem, budou provedeny atomicky. Součástí operace je i podmínka, jež

dovoluje takto upravit jen nezapůjčenou knihu. Databáze MongoDB vždy vrátí, kolik dokumentů bylo změněno. Pokud dojde k více pokusům o zapůjčení identické knihy ve stejnou chvíli, jen jeden z nich bude úspěšný. U ostatních bude počet změněných dokumentů roven nule. Při této skutečnosti aplikační vrstva vyhodí výjimku signalizující pokus o zapůjčení již zapůjčené knihy. Proces vrácení knihy probíhá obdobným způsobem.

Následující diagram popisuje datový model uvnitř MongoDB databáze (Vývojový diagram 11: Schéma dokumentové databáze). Každé schéma dokumentu je zastoupeno diagramem třídy. Dokumenty umístěné v samostatné kolekci jsou opatřeny stereotypem <<collection>>. Dokumenty vložené do jiných dokumentů jsou označeny pomocí relace agregace (čára zakončená kosočtvercem s bílým polem).



Vývojový diagram 11: Schéma dokumentové databáze MongoDB

Oproti prvotnímu návrhu podle původního diagramu entit, došlo k přejmenování hlavní entity z Document na Book (Vývojový diagram 1: Navržené entity). Hlavním důvodem pro přejmenování je, aby nedocházelo k záměně s dokumentem (formátem dat) v databázi MongoDB. Jednotlivé typy publikací nejsou reprezentovány jako samostatné entity, ale jsou rozlišeny pomocí atributu „Publication type“.

3.2 VYHLEDÁVÁNÍ V ULOŽENÝCH DATECH

Uživatelům musí být umožněno vyhledávat v datech. Vzhledem k požadavkům na činnost systému knihovny musí být aplikace schopna vyhledávat v atributech: jméno, autor, spoluautoři, vydavatel, země vydání, konference a klíčová slova. Naivní způsob implementace by mohl spočívat ve využití operátoru **$match**. Vyhledávací dotaz by mohl vypadat následovně: *"$match“: {„Author" : /Jan/is }*. Databáze by vrátila všechny výsledky obsahující v atributu autor řetězec „Jan“ (*/is* značí necitlivost na velká a malá písmena). Nicméně takový způsob vyhledávání není vhodný. Kdybychom aplikovali tuto podmínku na vícero atributů, ve výsledcích by již nebylo poznat, v kolika z nich byla nalezena shoda, a výsledky by tedy nebylo možné řadit od největší relevance.

Databáze MongoDB disponuje speciálním operátorem **$text** určeným pro textové vyhledávání. Kromě dotazu pro vyhledávání je možné specifikovat citlivost na velká a malá písmena nebo diakritiku. Při výchozím chování databáze výsledky nijak podle relevance neřadí. K vyhledávácímu dotazu lze přidat operátor **$meta** počítající relevantnost výsledku do námi vybraného atributu. Součástí výsledků tak bude další atribut obsahující desetinné číslo pro bezproblémové řazení výsledků od nejrelevantnějšího. Podmínkou pro použití operátoru je nutnost existence indexu typu text uvnitř kolekce, na níž bude databáze vyhledávání vykonávat. Lze indexovat libovolný textový atribut nebo pole textových atributů. Jedna kolekce smí mít pouze jeden index typu text, nicméně v rámci jednoho indexu je možné indexovat více atributů současně. Databáze díky tomu vždy vyhledává ve všech indexovaných atributech, nelze se tedy omezit jen na podmnožinu z nich. Pro některé jazyky má databáze definovaný seznam stop-slov. Jedná se o slova vyskytující se v daném jazyce často, ale neposkytující žádnou významovou informaci. Typickými zástupci jsou předložky, spojky nebo určité či neurčité členy. Mezi podporovanými jazyky bohužel chybí čeština, ale na vyhledávání to nemá kritický dopad, neboť pouze při výstavbě indexů jsou stop-slova přeskočena.

Alternativou je distribuovaný open-source vyhledávací a analytický engine s názvem Elasticsearch. Umožňuje textové / numerické nebo i zeměpisné vyhledávání. Je vybudován nad technologií Apache Lucene a byl poprvé představen v roce 2010. Poskytuje REST API pro vybudování nebo aktualizaci vyhledávacích indexů a zároveň umožňuje celé vyhledávání provádět. Oproti MongoDB disponuje větším množstvím podporovaných jazyků, dokonce existuje i slovník stop-slov pro češtinu. Jeho velkou nevýhodou je obrovská hardwarová náročnost. Dle dokumentace je minimální alokace operační paměti 8 GB. Doporučená (minimální) alokace paměti je dokonce 16 GB. (13)

Při implementaci jsem se rozhodl využít vyhledávání obsažené přímo v MongoDB. Sice nemá takovou podporu pro češtinu, ale má mnohem nižší požadavky na hardware. Zároveň nebude nutné komplikovat celkový návrh přidáním další služby. Vzhledem k datovému modelu uvnitř databáze (Vývojový diagram 11: Schéma dokumentové databáze) postačí vybudování indexů v kolekci „books“ a to na atributech: Name, Author, CoAuthors, Publisher, CountryOfPublication, Conference, Keywords.

3.3 DATOVÁ VRSTVA

Datová vrstva poskytuje přístup k MongoDB databázi a umožňuje pracovat s uloženými daty. K tomu využívá oficiální programovou knihovnu MongoDB Driver poskytující abstrakci v podobě rozhraní IMongoDatabase a IMongoCollection. S databází komunikuje v jazyce MongoDB Query Language (MQL), ale pro využití této programové knihovny nemusí vývojář syntaxi tohoto jazyka detailně ovládat. Jednotlivé typy dokumentů lze reprezentovat pomocí tříd v jazyce C# a jejich atributy pomocí vlastností (anglicky Properties). Při získávání dat z databáze jsou data automaticky serializována do příslušných objektů a při zápisu jsou naopak z objektů de-serializována. Dotazy vůči databázi lze stavět dvěma způsoby. (14)

Tím prvním je použití návrhového vzoru stavitel (anglicky Builder) vystavující množinu databází podporovaných operátorů prostřednictvím stejnojmenných metod. Stejný postup je rovněž využit při formování požadavků na vytvoření, úpravu nebo smazání dokumentu.

Druhým způsobem je využití abstrakce v podobě rozhraní System.Linq. IQueryable, tedy přímé součásti jazyka C# a používá se k formování dotazu nad libovolným datovým zdrojem. Na pozadí využívá System.Linq.Expressions. Expression místo vykonání příkazu napsaném v jazyce C# uloží jeho reprezentaci v podobě abstraktního syntaktického stromu. Programové knihovny třetích stran implementující toto rozhraní následně uskuteční analýzu vytvořeného stromu a zajistí převod gramatiky na gramatiku jazyka pro komunikaci s databází. Tímto způsobem se často řeší například objektově relační mapování. (14)

Během implementace jsem se při dotazování na data snažil maximálně využívat rozhraní IQueryable, avšak ne vždy to bylo možné. Například full-textové vyhledávání není zatím podporováno a byl jsem tak nucen použít výše zmíněný návrhový vzor stavitel. Naštěstí autoři programové knihovny MongoDB Driver jsou si těchto rozdílů vědomi a při formování dotazu lze oba způsoby vhodně zkombinovat. (14)

Databázové entity jsou reprezentovány pomocí tříd uchovávající pouze data, a tedy neobsahují žádnou vnitřní logiku. Jsou shodné s třídami znázorněnými na diagramu (Vývojový diagram 11: Schéma dokumentové databáze). Stejným způsobem je realizována i konfigurace. Obsahuje název databáze a řetězec potřebný k připojení. Třídy ve složce Queries obsahují několik metod, přičemž každá z nich odpovídá jednomu typu dotazu vůči databázi. Aplikační vrstva poskytuje třídu MongoDbContext pro práci s daty nad jednotlivými kolekcemi.

3.4 APLIKAČNÍ VRSTVA

Aplikační vrstva obsahuje nejpodstatnější část funkcionality. Přímo komunikuje s datovou vrstvu prostřednictvím třídy MongoDbContext. Definuje množinu rozhraní a část z nich i přímo implementuje. Slouží pro práci se systémovými entitami nebo k vykonání aplikačních procesů. Zbytek implementace, tykající se hlavně integrace s externími systémy, poskytuje infrastruktura. Přijímá požadavky z prezentační vrstvy, poskytuje jí data k zobrazení, případně data modifikuje na základě vstupu od uživatele.

3.4.1 Komunikace s prezentační vrstvou

Při komunikaci s prezentační vrstvou tam i zpět nejsou použity databázové entity. Ty totiž obsahují velké množství atributů, a není vždy na místě je všechny zobrazovat. Jiné zase není možné nastavit

a při použití databázových entit a ORM[8], by se aplikace stala zranitelná vůči útoku s názvem over-posting. Při tomto útoku útočník odešle na server více dat, než kolik jich je mu zobrazeno ve formuláři. Vzhledem k povaze funkčnosti ORM se data automaticky namapují a zapíšou do databáze. Tomu lze snadno předejít použitím DTO [9] objektu lišícího se od databázové entity. Tyto prosté objekty nemají žádnou vnitřní logiku a slouží pouze k přenášení dat. Aby byla ochrana před útokem účinná, musí se lišit DTO pro zápis od DTO pro čtení. (15)

3.4.2 Výjimky

Při pokusu o vykonání nevalidního požadavku nebo při selhání jednoho z autorizačních pravidel je běh programu zastaven vyhozením výjimky. Aplikační vrstva rozlišuje dva typy výjimek. Při nevalidním požadavku je vyhozena výjimka dědící ze třídy BusinessLayerException. Příkladem může být pokus o zapůjčení nedostupné knihy. Naproti tomu při neautorizovaném požadavku dojde k chybě typu AuthorizationException. V obou případech je k dispozici i slovní popis problému a prezentační vrstva může výjimku snadno odchytit a zobrazit uživateli zprávu o chybě.

3.4.3 Validace

Validace je nezbytnou součástí procesu zpracování vstupů od uživatele. Kromě ověření správnosti formátu dat musí být vynucena pravidla definovaná v aplikační logice. Nelze si například půjčit již zapůjčenou knihu nebo přidat knihu se stejným evidenčním číslem oddělení. K vynucení těchto pravidel je nutná komunikace s databází.

Při používání platformy .NET lze využít validaci pomocí tzv. atributů. Ty se při psaní kódu umísťují nad jednotlivé vlastnosti objektu. Pomocí nich lze například vynutit neprázdný textový řetězec, minimální délku řetězce, nebo splnění zadaného regulárního výrazu. Zároveň lze takto uskutečnit jen jednoduché validace, není například možné vykonávat komplexní logiku nebo spouštět asynchronní kód.

Alternativou je programová knihovna Fluent Validation. Množinu validačních pravidel udržuje v samostatné třídě. Pro každou vlastnost validovaného objektu lze přidat jedno nebo více pravidel. Jednotlivá pravidla jsou definována v konstruktoru třídy dědící z AbstractValidator<T>, kde T symbolizuje námi validovaný objekt. U každého pravidla lze definovat i chybovou hlášku zobrazující se při jeho nesplnění. Je možné přidat i vlastní pravidla, a dokonce během validace spouštět i asynchronní kód. Z těchto důvodů jsem se rozhodl upřednostnit tuto programovou knihovnu před anotacemi a definovat validační pravidla v aplikační vrstvě pomocí samostatných tříd.

3.4.4 Rozhraní

Při definování jednotlivých rozhraní jsem se držel návrhového vzoru fasáda. Ten si klade za cíl zjednodušit navenek komplexní funkcionalitu a celkově tak zlepšit čitelnost a porozumění kódu. Prezentační vrstva tedy nebude muset rozumět komplexním strukturám, ale namísto toho použije sadu jednoduchých metod v kombinaci s prvky uživatelského rozhraní. Implementace zbytku rozhraní je v kompetenci kódu umístěného v části infrastruktury.

---

[8] Objektově relační mapování Zajišťuje automatickou konverzi dat mezi databází a objekty z datové vrstvy.

[9] Zkratka pro Data Transfer Object

3.4.5 Služby

Některá rozhraní jsou v této vrstvě přímo implementována a poskytována formou služeb (anglicky services). Lze je chápat jako třídy s vnitřní logikou a množinou závislostí deklarovanými v konstruktoru formou parametrů. Služba ke své činnosti vždy potřebuje nějaké rozhraní, ale nikdy nezávisí na konkrétní implementaci. Díky tomu jsou dodrženy principy Barbary Liskov a inverze závislostí z paradigmatu SOLID.

Book Service

Poskytuje skupinu metod umožňující práci s publikacemi. Prezentační vrstvě umožňuje filtrovat knihy dle vícero kritérií a výsledná data stránkovat, a tak snížit množství načtených dat z databáze, což přispívá i k rychlejšímu překreslování uživatelského rozhraní. Kromě jednoduchých filtrů poskytuje i metodu pro full-textové vyhledávání. Dále lze knihu vyhledat pomocí NFC štítku nebo jednoznačným identifikátorem přestavující evidenční číslo oddělení. Běžným uživatelům po přečtení knih umožňuje přidat hodnocení. Zájemcům o knihu dovolí nastavit si upozornění na její dostupnost. Knihovníkům je dovoleno data o knihách editovat, exportovat nebo importovat pomocí CSV souboru a vyřazené položky rovnou mazat.

Book Status Service

Dovoluje pracovat s jednotlivými stavy knih. Ty lze vytvořit, upravit nebo odstranit. Při smazání stavu, se všechny dotčené knihy přesunou do výchozího stavu. Při importování knih z CSV souboru, je možné stav uvést v příslušném sloupečku.

Card Service

Každý uživatel si může registrovat několik přístupových karet s volitelným pinem. Tato služba pouze zajišťuje přiřazení karty k příslušnému uživateli. Ukládání pinu a o jeho následnou verifikaci obhospodařuje komponenta Self Service Library Card Authentication, poskytovaná infrastrukturou.

Guest Service

V aplikaci se kromě uživatelů s ČVUT účty vyskytují i hosté. Tato služba slouží k jejich správě. Dovoluje přidávat osoby, jejich kontaktní údaje a napovídat konkrétního hosta na základě textového řetězce. U každého hosta eviduje jméno, příjmení, tituly, kontaktní údaje a poznámku.

Issue Service

Kdykoli knihovník někomu něco půjčí nebo si uživatel knihu půjčí sám, vznikne záznam o výpůjčce. Tyto záznamy lze filtrovat podle nejrůznějších kritérií. Služba také poskytuje možnost zobrazit historii knihy s chronologicky seřazenými výpůjčkami. Zároveň poskytuje metodu k vrácení.

User Service

Obvykle se pro práci s uživateli a přidělování rolí používá na platformě .NET programová knihovna ASP.NET Identity. Protože data uživatelů i autentizaci poskytují externí systémy, rozhodl jsem se ji v tomto případě nepoužít. Namísto toho je však využita při ukládání a ověření přístupových karet. Třída UserService poskytuje metody pro přidání uživatele do role a ověření, zda se v roli vyskytuje.

3.5 INFRASTRUKTURA

Infrastruktura leží mimo všechny tři vrstvy. Implementuje zbytek rozhraní, bez kterých nemůže aplikace pracovat, ale jejichž implementace nebyla poskytnuta aplikační vrstvou. Drží povědomí o všech implementačních detailech souvisejících s programovými knihovnami třetích stran nebo externími systémy. Hlavní myšlenkou je namapovat rozhraní a objekty z aplikační vrstvy na rozhraní a objekty ležící mimo ni. K tomu je využíván návrhový vzor adaptér, jehož cílem je překonat nekompatibilitu mezi dvěma rozhraními. Analogie toho problému ze života běžného člověka je nekompatibilita různých zásuvek a elektrických koncovek.

Při integraci externího systému je mnohdy zapotřebí využívat HTTP protokol. To přináší komplikace v podobě nutnost starání se o životnost objektu HttpClient. Situaci lze spolehlivě řešit pomocí objektu HttpClientFactory v kombinaci se vkládáním závislostí. Ten sice vytváří více instancí objektu HttpClient, ale současně se snaží maximálně recyklovat instance objektu HttpMessageHandler, jež na pozadí využívá. Implementace využívá návrhový vzor Object pool. Namísto vytváření nových instancí jsou tedy znovupoužity dříve odložené instance. (16)

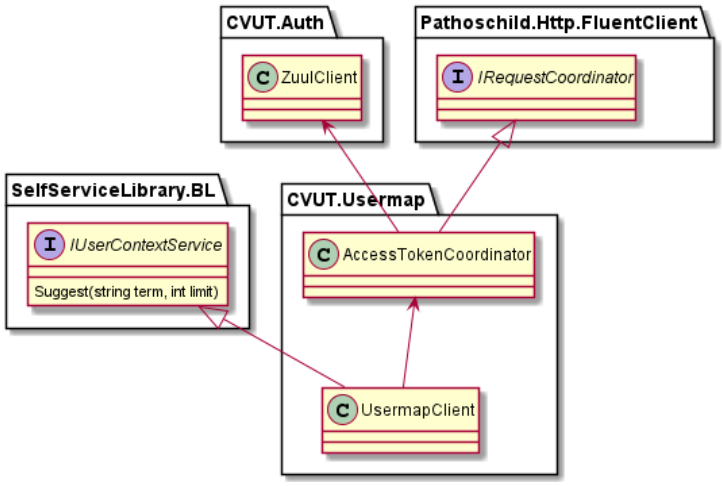
Pro usnadnění implementace při používání HTTP protokolu jsem se rozhodl použít programovou knihovnu s názvem Pathoschild.Http.FluentClient. Ta poskytuje abstrakci nad rozhraním třídy HttpClient. Odesílání a přijímání požadavků nebo serializaci a de-serializaci lze tak uskutečnit s mnohem kratší syntaxí. Kromě toho si může vývojář doprogramovat nejrůznější řídící prvky zajišťující například obnovu přístupového tokenu. Ve svém konstruktoru přijímá instanci třídy HttpClient, tím je možné jej použít i s výše popsaným návrhovým vzorem a předejít tak zmíněným problémům.

3.5.1 Integrace s autorizačním serverem Zuul OAAS

Neimplementuje žádné rozhraní z aplikační vrstvy, ale poskytuje pouze abstrakci nad autorizačním serverem Zuul OAAS. Prezentační vrstvě poskytuje metody k získání uživatelského tokenu nebo jeho obnově. Alternativně lze token i získat při použití dvojice client_id a client_secret. Jelikož je stav tokenu uložen na autorizačním serveru, obsahuje i metodu pro ověření platnosti tokenu a zjištění uživatele kterému patří.

3.5.2 Integrace se systémem Usermap

Namísto aplikační vrstvy implementuje rozhraní IUserContextService vyhledávající na základě textového řetězce osoby z ČVUT (Vývojový diagram 12: Struktura integrace s ČVUT Usermap).

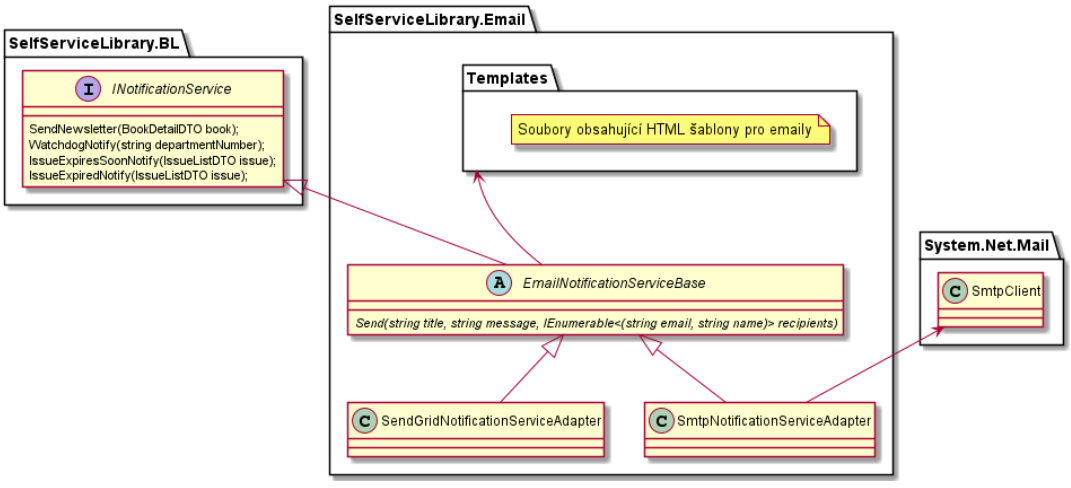


Vývojový diagram 12: Struktura integrace s ČVUT Usermap

Toho je dosaženo komunikací se systémem Usermap. Během komunikace je nutné zajistit stálou platnost používaného přístupového tokenu. Toto je úkolem třídy AccessTokenCoordinator kontrolující před každým požadavkem přítomnost a platnost tokenu. V případě nutnosti obstará nový token voláním metody ze třídy ZuulClient.

3.5.3 Odesílání emailů a tvorba šablon

Aplikační vrstva potřebuje na několika místech notifikovat uživatele prostřednictvím rozhraní INotificationService (Vývojový diagram 13: Implementace tvorby šablon a odesílání emailů). Prvním případem je rozeslání newsletteru knihovníkem upozorňující registrované návštěvníky knihovny na vložení nové knihy do knihovny. Služba na pozadí toto rozhraní využívá rovněž k připomenutí vrácení uživatelem zapůjčené knihy. Každý uživatel si může navíc nastavit hlídání dostupnosti právě zapůjčené knihy někým jiným. Systém jej pak upozorní, jakmile je kniha opět dostupná, tedy vrácena.



Vývojový diagram 13: Implementace tvorby šablon a odesílání emailů

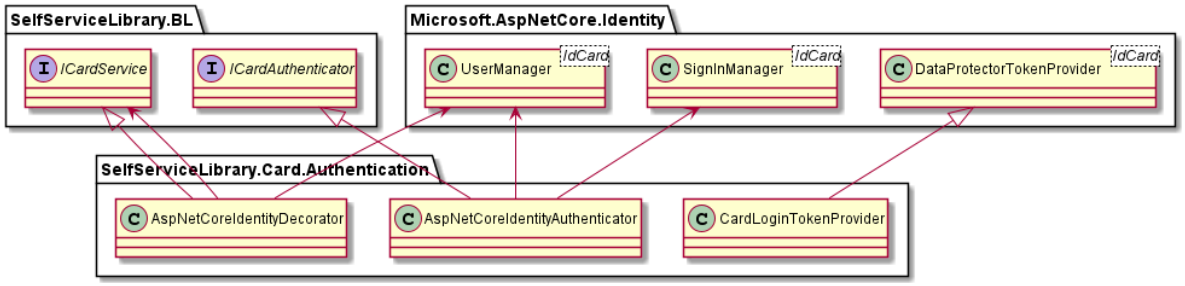
Rozhraní je implementováno abstraktní třídou EmailNotificationServiceBase sloužící současně jako bázová třída pro potenciální adaptéry emailových klientů. Adaptéry pak musí implementovat metodu Send poskytující odeslání emailu vícero příjemcům. Aby bylo možné emaily jednodušeji šablonovat, tak třída pří startu aplikace načte ze souborového systému HTML šablony pro jednotlivé

emaily. Ty obsahují textové řetězce ve složených závorkách, jež jsou před skutečným odesláním nahrazeny daty z objektu BookDetailDTO.

Výše zmíněná abstraktní třída je dále implementována dvěma emailovými adaptéry. Tím prvním je třída SmtpNotificationServiceAdapter odesílající emaily pomocí SMTP protokolu a je využita ke komunikaci s poštovním serverem ČVUT. Druhým je SendGridNotificationServiceAdapter odesílající emaily prostřednictvím REST API rozhraní cloudové služby SendGrid. Sloužící však pouze k odesílání emailů při ladění v testovacím provozu aplikace.

3.5.4 Ukládání a ověřování NFC karet

Aby bylo možné autentizovat uživatele pomocí NFC karet, musí si je uživatel nejprve přidat, tedy registrovat. Navíc ji může opatřit pinem, ten je však potřeba bezpečně uložit v databázi. Rozhodl jsem se použít existující programovou knihovnu AspNetCore.Identity.Mongo implementující rozhraní AspNetCore.Identity mocí MongoDB databáze (Vývojový diagram 14: Autentizace přístupovou kartou pomocí ASP.NET Core Identity).



Vývojový diagram 14: Autentizace přístupovou kartou pomocí ASP.NET Core Identity

Při implementaci je využíván návrhový vzor dekorátor pro dodržení open-close principu z paradigmatu SOLID. Třída AspNetCoreIdentityDecorator se nejprve pokusí číslo karty a pin uložit pomocí třídy UserManager. Pokud uspěje přepošle parametry volání do již existující implementace rozhraní ICardService.

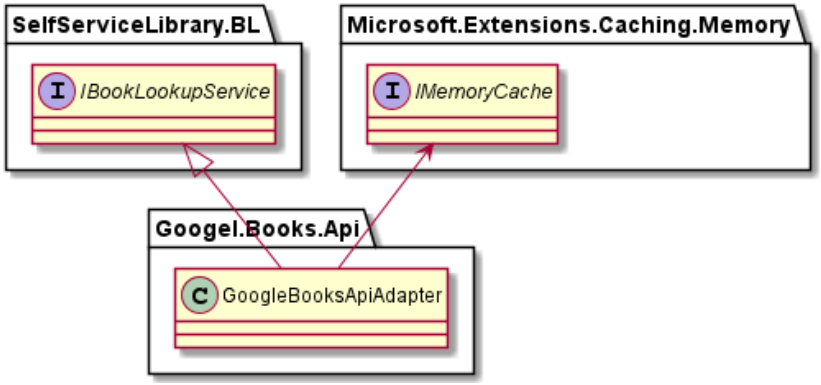
Autentizace uživatele probíhá ve třídě AspNetCoreIdentityAuthenticator. Pro ověření příchozí kombinace čísla karty a pinu se zavolá metoda ze třídy SignInManager. Ta se na pozadí stará i o překročení počtu neplatných pokusů při zadání pinu. Pokud k tomu dojde, uzamkne možnost přihlášení se po stanovenou dobu. Útočníkovi je tak znesnadněn útok hrubou silou, jenž by v případě pinů o délce 4 znaků byl velmi efektivní. (17)

Aby mohla aplikace na kiosku autentizovat uživatele pomocí přístupové karty a pinu, rozhraní ICardAuthenticator obsahuje metodu k vygenerování dočasného tokenu. Ten je získán po zadání platné kombinace čísla karty a pinu a může být později použit k autentizaci. Aplikace na kiosku tedy nejprve odešle na REST API službu zadané přístupové údaje, získá zpět dočasný token a přesměruje uživatele do webové aplikace, kde předá token do adresy URL. Webová aplikace se pokusí ověřit token a najít podle karty odpovídajícího uživatele a v případě úspěchu jej přihlásit.

Vygenerování a ověření tokenu zajišťuje třída CardLoginTokenProvider dědící ze třídy DataProtectorTokenProvider určené k těmto účelům. Musela být uskutečněna drobná modifikace, neboť ve výchozím nastavení jsou tokeny ukládány do paměti. Nicméně jelikož se jedná o dvě samostatné aplikace nesdílející operační paměť, musí být tokeny uloženy do databáze k příslušné kartě. Po úspěšném ověření, je token odebrán. (18)

3.5.5 Integrace se službou Google Books

Data o publikacích vznikala na různých místech bez nějaké jednotné validace. Díky tomu se v datech často vyskytují překlepy nebo některé informace úplně chybí. Aby byla knihovníkům usnadněna úprava existujících záznamů, využívá prezentační vrstva rozhraní IBookLookupService.

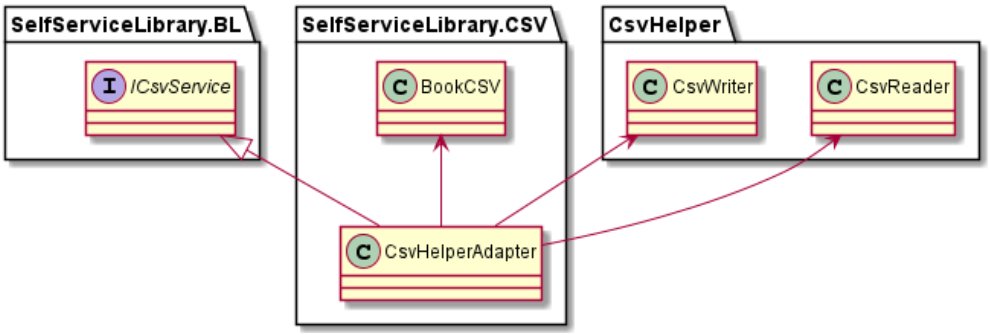


Vývojový diagram 15: Integrace na službu Google Books API a využití cache v operační paměti

To je implementováno třídou GoogleBooksApiAdapter získávající z detailu knihy ISBN a snažící se o knize dohledat zbytek informací pomocí služby Google Books. Data o knihách se příliš často nemění, a proto jsou odpovědi služby uchovány v operační paměti (Vývojový diagram 15: Integrace na službu Google Books API).

3.5.6 Práce s CSV soubory

Pro importování a exportování knih v podobě CSV souborů slouží rozhraní ICsvService implementované ve třídě CsvHelperAdapter. Ke své činnosti využívá programovou knihovnu CsvHelper. Struktura jednoho řádků CSV souboru je reprezentována uvnitř třídy BookCSV. Jelikož programová knihovna sama vykoná serializaci a de-serializaci, lze tak zajistit shodu schématu pro importování a exportování, (Vývojový diagram 16: Implementaci importu a exportu CSV pomocí programové knihovny).



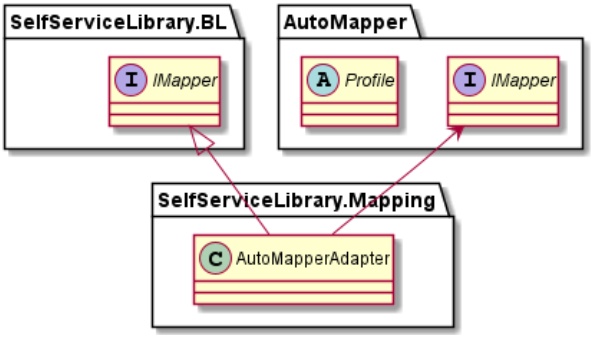
Vývojový diagram 16: Implementaci importu a exportu CSV pomocí programové knihovny

3.5.7 Mapování a přenos dat mezi komponentami

Mezi jednotlivými vrstvami dochází ke komunikaci pomocí objektů. Tyto vrstvy mají často jiná rozhraní, což vede k nutnosti mapování dat mezi nimi. Při tvorbě aplikace tak vzniká velmi monotónní kód, jenž se opakuje a bývá často rozsáhlý, díky čemuž je náchylný k chybám. Častou chybou bývá přidání atributů do systémové entity, ale následně je opomenuto mapování na příslušný atribut DTO. Problému se lze vyhnout, pokud definujeme mapování na jednom místě a zahrneme jej i do procesu testování.

K implementaci jsem se rozhodl využít programovou knihovnu Automapper. Kromě jednoho objektu dokáže mapovat i celé kolekce. Místo mapování kolekce v operační paměti, lze využít projekci sestavující abstraktní syntaktický strom pomocí rozhraní IQueryable. Kapitola Datová vrstva popisuje jakým způsobem je pak zajištěno, aby se mapování uskutečnilo přímo v databázi. Další velká výhoda spočívá v možnosti testování celého mapování. Zmíněná programová knihovna k tomu poskytuje metodu ujišťující neopomenutí žádných atributů použitých v definici. (19)

Jednotlivá mapovací pravidla se nachází v tzv. profilech. Tyto třídy dědí z abstraktní třídy poskytnuté programovou knihovnou a její název končí slovem Profile. Zajištění kompatibility mezi rozhraním aplikační vrstvy a programové knihovny Automapper je dosaženo pomocí třídy AutoMapperAdapter.



Vývojový diagram 17: Adaptace rozhraní IMapper na rozhraní knihovny AutoMapper

3.6 Prezentační vrstva

Při používání aplikace interaguje koncový uživatel právě s touto vrstvou. Zobrazuje data přes webové rozhraní a přijímá uživatelské vstupy a ty vždy validuje. Kromě webové aplikace obsahuje i REST API službu komunikující s aplikací na kiosku. Poslední částí je služba na pozadí upozorňující uživatele na datum vrácení knihy.

3.6.1 Webová služba (Self Service Library Web)

Pro implementaci webové aplikace jsem se rozhodl použít technologii Blazor umožňující vytvářet interaktivní webová rozhraní bez nutnosti psaní kódu v JavaScriptu. Lze tak snadno znovupoužít kód v C# vytvořený současně v aplikační vrstvě. Blazor dokáže pracovat ve dvou režimech. (20)

Klientský režim využívá technologii Web Assembly. Webový prohlížeč si při načtení stránky stáhne binární soubory a celou aplikaci spustí nativně. Nevýhoda spočívá ve složité interakci s DOM[10]podporující v aktuální verzi stále pouze Javascript. Dalším problémem je velké množství stahovaných dat při prvním spuštění aplikace a doba nutná k inicializaci běhového prostředí. Kromě toho je stále nedostatečně optimalizovaná automatická správa paměti. Situaci u jazyků jako je C# nebo Java komplikuje skutečnost přítomnosti jejich vlastní komponenty pro automatickou správu paměti. Prohlížeče se tak musí potýkat s další vrstvou abstrakce navíc. (21)

Druhým režimem je serverový režim využívající k práci oboustrannou komunikaci mezi klientem a serverem pomocí technologie SignalR. Ta se na pozadí pokouší maximálně využívat WebSocket, nicméně pokud není na daném zařízení podporován, dokáže simulovat jeho práci pomocí

[10] Objektový model dokumentu je objektově orientovaná reprezentace HTML stránky.

alternativních řešení. V tomto režimu běží veškerý námi napsaný kód na straně serveru. Veškerá uživatelská interakce je posílána na server vykonávající výpočty. Výsledky jsou odeslány zpět na klienta zajišťujícího překreslení uživatelské rozhraní. Nevýhodou tohoto přístupu je citlivost na slabší internetové připojení. Aplikace navíc nedokáže pracovat v off-line režimu. (22)

Při implementaci jsem se rozhodl použít serverový režim, neboť technologie Web Assembly je zatím stále ve vývoji, a ne všechna zařízení ji plně podporují. Pro zjednodušení tvorby uživatelského rozhraní jsem do aplikace přidal programovou knihovnu Blazorise. Jejím cíle je vytvořit abstrakci nad různými CSS frameworky, a tak umožnit vývojářům vyvíjet aplikaci za pomocí Blazor komponent namísto psaní HTML. Při instalaci jsem zvolil možnosti, aby byly jednotlivé komponenty implementovány pomocí frameworku Bootstrap. (23)

Validace

Jednotlivá validační pravidla jsou definována v aplikační vrstvě, nicméně validace je uskutečněna i v prezentační vrstvě. Uživatel díky tomu vidí, kde se dopustil chyby. Autor programové knihovny Blazorise zajistil snadnou integraci s validačními anotacemi, ale již neposkytnul integraci s programovou knihovnou Fluent Validation. Naštěstí tato implementace byla poskytnuta členem komunity ve formě NuGet balíčku s názvem Blazorise Fluent Validation. Při porušení jednoho z pravidel se zobrazí příslušná chybová hláška u prvku v uživatelském rozhraní obsahující chybná data.

Autentizace a autorizace

Autentizace je proces ověřující identitu uživatele. Autorizace je proces ověřující, zda má daný uživatel právo, uskutečnit nějakou specifickou akci. Při implementaci v obou případech napomáhají komponenty s názvem middleware aktivující se při každém příchozím požadavku. Jsou součástí frameworku ASP.NET Core použitého v prezentační vrstvě. Vývojář musí pouze definovat jedno nebo více autentizačních schémat a sadu pravidel vynucených v rámci autorizace. (24)

K autentizaci uživatele pomocí OASS je využíván OAuth 2.0 autentizační middleware. Ten předpokládá dodržení standardu protokolu na straně autentizačního serveru. Automaticky zajistí přesměrování uživatele na přihlašovací webovou stránku a po jeho úspěšném přihlášení a přesměrování zpět, vykoná definovanou proceduru. V kontextu zde vytvářeného řešení se jedná o získání uživatelského jména a dat o uživateli ze systému Usermap. Musel jsem uskutečnit drobnou modifikaci, neboť middleware odesílá dvojici **client_id** a **client_secret** v těle požadavku, zatímco server Zuul ji očekává v autorizační hlavičce. (25)

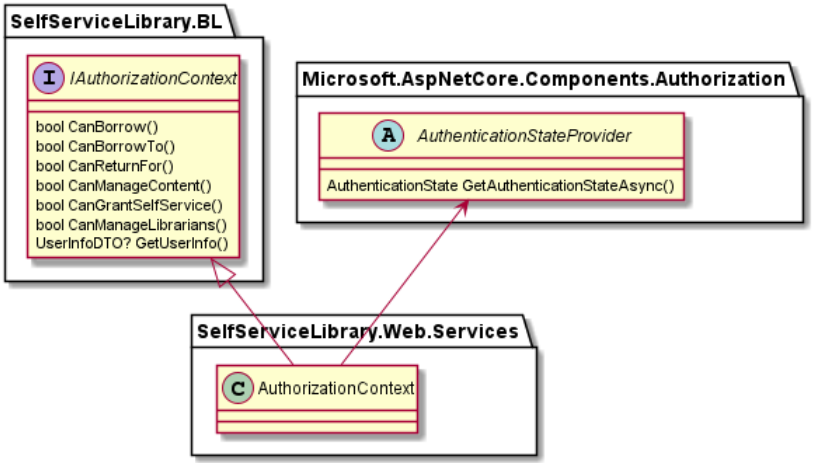
Při autentizaci uživatele pomocí přístupové karty se předpokládá získání dočasného tokenu aplikací na kiosku z REST API služby. Ten je předán jako parametr v adrese URL a následně webovou aplikací ověřen. Po úspěšném ověření je autentizován uživatel a ten je poté v systému uveden jako držitel přístupové karty. V tomto případě je uživatelská relace opatřena příznakem indikující použití kiosku v knihovně.

Při používání kiosku se uživatel samozřejmě může přihlásit i uživatelským jménem a heslem ČVUT. Aby to webová aplikace dokázala rozpoznat od předchozího případu, musí se kiosk autentizovat pomocí jednorázového hesla. To se vygeneruje pomocí programové knihovny Otp.NET a webová aplikace pak přes stejnou programovou knihovnu vykoná ověření. Na pozadí je používán TOTP algoritmus, jenž pomocí tajného klíče v kombinaci s aktuálním časem vygeneruje unikátní řetězec.

(26) Zmíněný tajný klíč nikdy není součástí komunikace a obě aplikace jej musí načíst z bezpečně uloženého konfiguračního souboru. Jakmile aplikace na kiosku přesměruje uživatele na přihlašovací stránku a do URL adresy uvede jednorázové heslo, bude se webová služba chovat jako v předchozím případě, ale tentokrát po úspěšném přihlášení opatří uživatelskou relaci příznakem indikujícím použití kiosku v knihovně.

Při autentizaci uživatele dochází k propojení informací z několika míst. Uživatelské jméno je získáno z autorizačního serveru Zuul, data o uživateli poskytuje systém Usermap a aplikační uživatelské role definuje zde vytvářený knihovní informační systém. Všechny tyto údaje jsou uloženy ve formě Cookies a odeslány do prohlížeče. Pro tento účel je využíván Cookie Middleware odpovídající za vystavení, šifrování a ověření integrity. Na pozadí jde o používání ASP.NET Core Data Protection API. Díky tomu nemusí webová aplikace kontaktovat externí služby při každém uživatelském požadavku, ale pouze si přečte data z Cookies. (27)

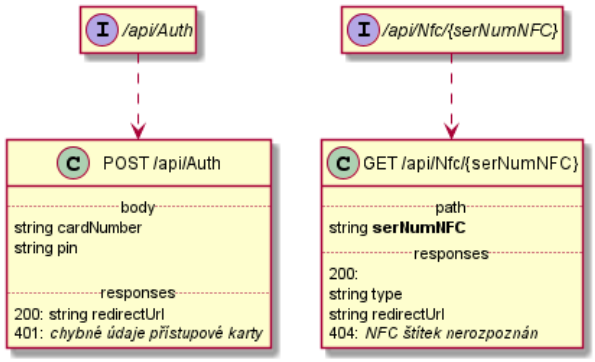
Pro aplikační vrstvu je zajištěna implementace rozhraní IAuthorizationContext pomocí třídy AuthenticationStateProvider. Ta je součástí Blazoru a poskytuje autentizační a autorizační kontext uživatele získaný přečtením a de-serializací Cookies. Webová aplikace některé prvky uživatelského rozhraní skryje, pokud se uživatel nenachází v roli knihovník. K tomu slouží vestavěná komponenta AuthorizeView využívající rovněž objekt AuthenticationState (Vývojový diagram 18: Autentizace ve webové aplikaci).



Vývojový diagram 18: Autentizace ve webové aplikaci

3.6.2 REST API služba (Self Service Library API)

Webové API implementované pomocí frameworku ASP.NET Core 5.0 poskytuje dva koncové body, se kterými může aplikace na kiosku interagovat (Vývojový diagram 19: REST API Rozhraní pro komunikaci s kioskem). První umožňuje získání dočasného tokenu pro autentizaci uživatele prostřednictvím přístupové karty a pinu. Druhý pomáhá kiosku rozpoznat přiložený NFC štítek pro identifikaci, zda se jedná o knihu či registrovanou přístupovou kartu.



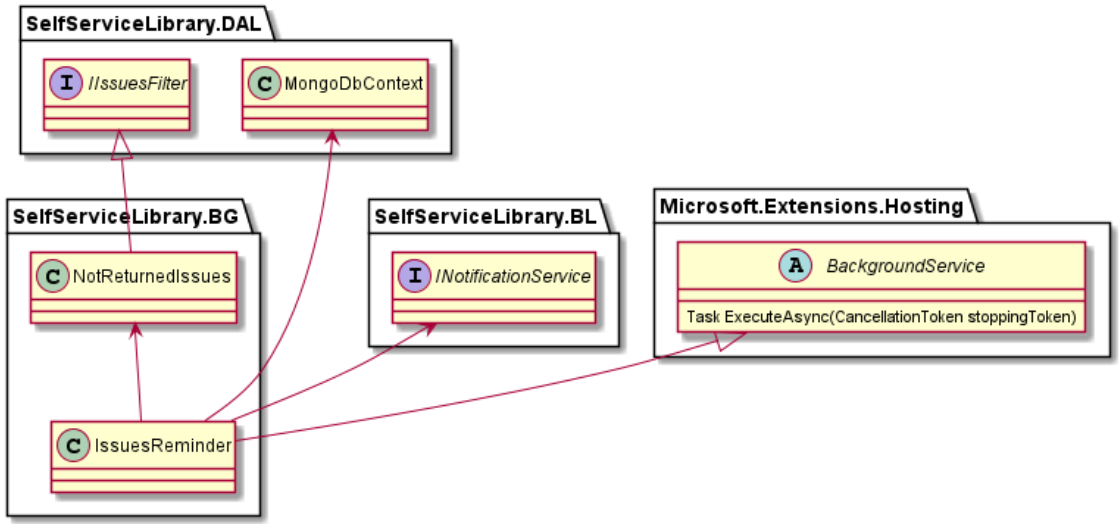
Vývojový diagram 19: REST API Rozhraní pro komunikaci s kioskem

Aby mohla desktopová aplikace na kiosku výše zmíněné koncové body používat, musí s každým HTTP požadavkem odeslat v hlavičce s názvem **X-Api-Key statický** klíč, jimž se vůči REST API služba autentizuje. To brání v používání REST API služby někým jiným.

Rozhraní je zdokumentováno pomocí Open API definice vygenerované ze zdrojového kódu při sestavování. Tato funkcionalita je zajištěna pomocí programové knihovny s názvem Swashbuckle.

3.6.3 Služba na pozadí (Self Service Library BG)

Služba na pozadí je implementována jako samostatná aplikace. Díky tomu ji lze nasazovat a aktualizovat nezávisle na webové službě. V současnosti obsahuje jen jeden proces na pozadí upozorňující uživatele na datum vrácení knihy.



Vývojový diagram 20: Implementace služby na pozadí notifikující uživatele

Další procesy lze snadno přidávat. Stačí vytvořit novou třídu dědící z abstraktní třídy BackgroundService. O zbytek se již postará Microsoft.NET.Sdk.Worker. SDK bohužel neposkytuje abstrakci na práci s časovači, takže pokud se má proces na pozadí spouštět periodicky, musí si tento mechanismus vývojář implementovat sám.

Třída IssuesReminder komunikuje s datovou vrstvou přímo (Vývojový diagram 20: Implementace služby na pozadí). Data poskytnutá třídou MongoDbContext lze totiž iterovat pomocí asynchronního iterátoru, díky čemuž není nutné alokovat paměť pro celou kolekci nevrácených výpůjček, ale lze je zpracovávat po jedné.

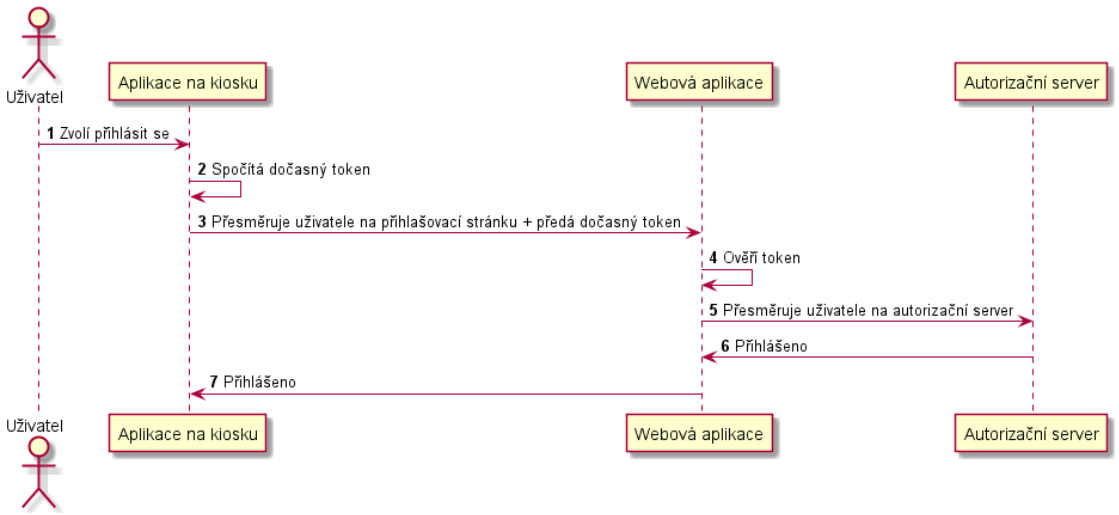
3.6.4 Aplikace na kiosku v knihovně

Limitace webových aplikací spočívá v jejich často velmi složité integraci s hardwarovými periferiemi. Nejen proto v rámci této práci vznikne desktopová aplikace obsahující sice vestavěný webový prohlížeč, ale díky svému nativnímu běhovému prostředí dokáže snadno komunikovat s potřebným hardwarem. Díky tomu se tedy budou uživatelé moci přihlásit svojí přístupovou kartou nebo si snadno půjčit knihu jejím přiložením ke čtečce. Aplikace byla ve spolupráci s vedoucím vyvinuta ve frameworku .NET Core 3.1 a pro definici uživatelského rozhraní je použit Windows Presentation Foundation (WPF).

Pro vykreslování webové aplikace využíváme komponentu Microsoft.Web.WebView2 pracující na pozadí s renderovacím enginem Microsoft Edge (Chromium). Kiosk se tímto řadí do kategorie hybridních aplikací, jejichž hlavní myšlenkou je právě kombinování výhod webových a nativních aplikací současně (28). Následuje soupis mnou navržených a implementovaných funkcionalit v aplikaci kiosku.

Standardní přihlášení

Aby dokázala webová aplikace při přihlášení odlišit kiosk od jiného zařízení (osobní počítač, mobilní telefon, …), je použit jiný koncový bod ve webové aplikaci. Kiosk se vůči webové aplikaci autentizuje pomocí dočasného tokenu vygenerovaného pomocí kombinace aktuálního času a tajného klíče (Vývojový diagram 21: Postup přihlášení bez použití karty). Stejně jako při ověřovaní, lze použít programovou knihovnu Otp.NET. Aplikace na kiosku poskytuje vlastní tlačítko na přihlášení a původní tlačítko na webové stránce je skryto pomocí CSS stylů. Pokud by se totiž uživatel přihlásil pomocí tlačítka přímo ve webové aplikaci, systém by nepoznal, zda přistupuje z kiosku.



Vývojový diagram 21: Postup přihlášení bez použití karty na kiosku

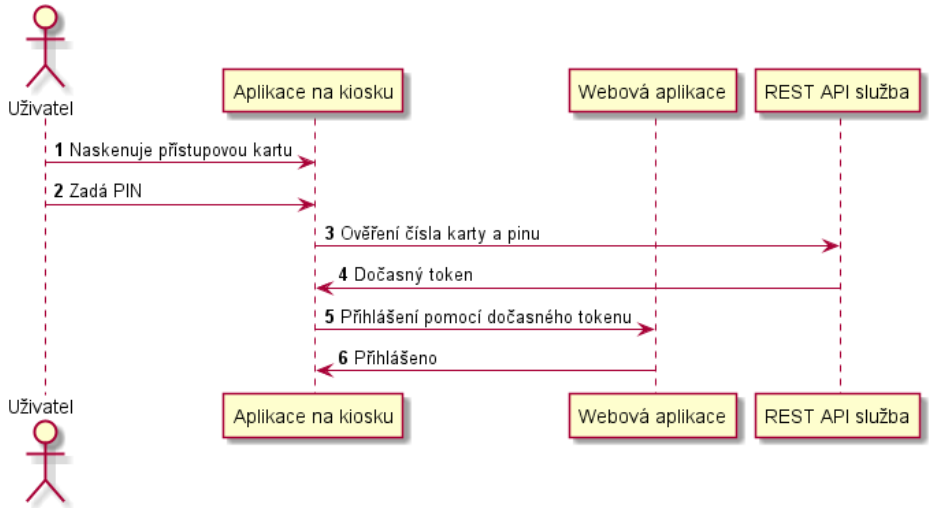
Registrace přístupové karty

Každý uživatel si může registrovat libovolnou NFC kartu a později ji použít ke snadnému přihlášení. Ne však každý vlastní NFC čtečku, a proto bude tato funkce dostupná pouze na kiosku umístěném v knihovně. Před registrací přístupové karty se musí uživatel nejprve přihlásit pomocí svého jména a hesla ČVUT. Poté při registraci přístupové karty je přesměrován do webové aplikace, kde přidání

karty dokončí. Protože klonování NFC karet je velmi snadné, lze si v dialogu pro registraci karty zvolit i 4-místný přístupový pin vyžadovaný poté při každém přihlášení.

Přihlášení pomocí registrované karty

Svou dříve přidanou (registrovanou) kartu mohou uživatelé později využít k přihlášení probíhajícího podle následujícího schématu (Vývojový diagram 22: Postup přihlášení pomocí přístupové karty na kiosku). Uživatel přiloží svou kartu a případně zadá pin. Aplikace na kiosku si poté vyzvedne dočasný token, s jehož pomocí následně autentizuje uživatele ve webové aplikaci. Token je předán jako parametr v URL adrese, avšak lze jej použít jen jednou. Je navíc unikátní pro každý pokus o přihlášení.



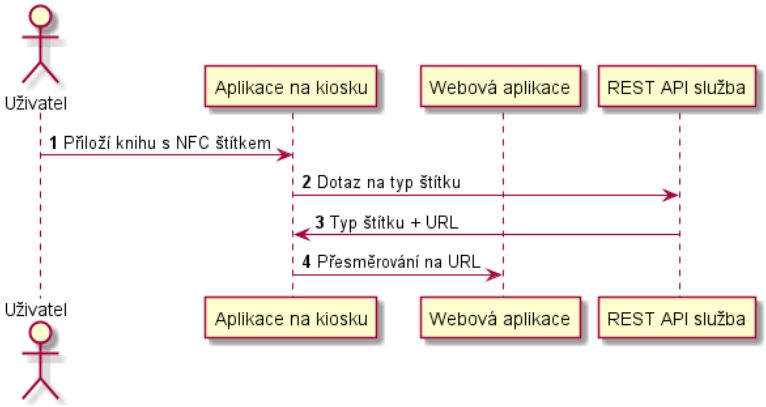
Vývojový diagram 22: Postup přihlášení pomocí přístupové karty na kiosku

Odhlášení

Při odhlášení je potřeba myslet na nedostatek autorizačního serveru Zuul neposkytujícího koncový bod pro odhlášení uživatele, což bylo popsáno v části (Další systémy). Aplikace na kiosku dokáže tento problém odstranit vyprázdněním všech Cookies prohlížeče při odhlášení uživatele. Stejně jako v předchozím případě musí být tlačítko pro odhlášení (na webové stránce) skryto a poskytnuto aplikací na kiosku.

Načtení NFC štítku knihy

Každý může ke čtečce na kiosku přiložit knihu a zobrazit si její detail ve webové aplikaci. Aplikace na kiosku nejprve identifikuje typ štítku pomocí REST API služby. V odpovědi je obsažena URL adresa s detailem (pokud ji NFC štítek neobsahuje přímo). Tu aplikace pak jednoduše zobrazí ve vloženém webovém prohlížeči (Vývojový diagram 23: Identifikace knihy dle detekovaného NFC).



Vývojový diagram 23: Identifikace knihy dle detekovaného NFC štítku

4 IMPLEMENTACE

Zde vytvářené řešení bude sice poprvé nasazeno pro katedru kybernetiky, avšak nic nebrání tomu, aby systém využívaly i další katedry. Ty si ovšem budou muset řešení umístit na svůj server, a to včetně databáze. Pro zjednodušení vývoje a následné instalace řešení jsem se rozhodl použít Docker. Jedná se o nástroj poskytující možnost kontejnerizovat aplikaci. Aplikace se v podstatě zabalí do jednoho logického celku spolu se všemi závislostmi potřebnými ke své činnosti. Mezi ty patří běhová prostředí, programové knihovny nebo systémové nástroje. Každou aplikaci tak lze zabalit formou Docker Image a jediné co se musí na server nainstalovat, je právě Docker. (29)

K propojení jednotlivých kontejnerů a definování vícero služeb jako jednoho celku, využívám nástroj Docker Compose. Celé řešení se dá spustit pomocí jediného příkazu v příkazové řádce. Jednotlivé služby a vazby mezi nimi jsou definovány v konfiguračním souboru „docker-compose.yml". I databáze je provozována uvnitř kontejneru, což přispívá k jednodušší přenositelnosti celého řešení. Docker Compose rovněž umístí kontejnery do vnitřní sítě, sice na sebe vidí vzájemně, ale z vnější sítě nejsou dostupné. To se dá explicitně změnit nastavením parametru ports. (30)

Způsob nasazení na infrastrukturu katedry kybernetiky jsem konzultoval s IT administrátorem Ing. Danielem Večerkou. Byl mi poskytnut virtuální server, kam jsem nainstaloval Docker a Docker Compose. Jedním ze síťových prvků na katedře je i reverzní proxy. Když vznikají nové webové projekty, obvykle se pro ně vytvoří nový podadresář. Každý požadavek na tento podadresář je pak obsloužen příslušným webovým serverem.

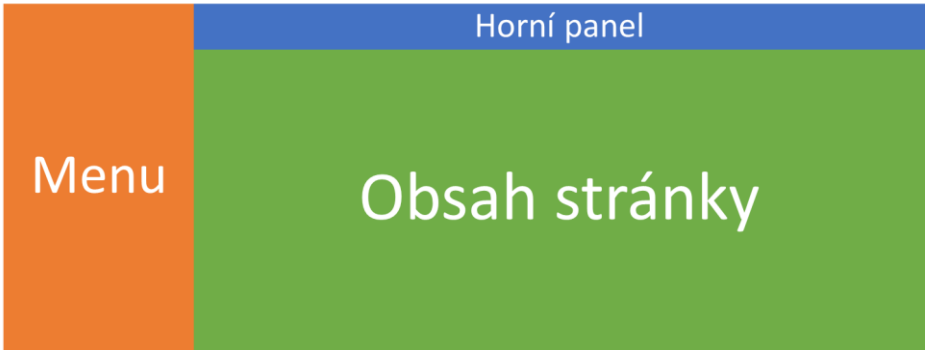
Protože na serveru se budou nacházet dvě služby komunikující přes protokol HTTP (webová aplikace a REST API služba), bude reverzní proxy komunikovat se směrovačem. K tomuto účelu použiji službu Traefik a přidám ji jako další kontejner do Docker Compose konfigurace. Ta dokáže pomocí jednoduchého nastavení směrovat požadavky na příslušnou službu. Všechny požadavky, jejichž URL začíná na „/api" budou směřovány na REST API službu a zbytek na webovou aplikaci. Dle dokumentace dokáže i sama od sebe řešit web sockety nutné pro správnou činnost Blazor. (31)

4.1 WEBOVÁ APLIKACE

Nejdůležitějším kontejnerem je webová aplikace, neboť právě s tou uživatel pracuje. Od uživatele přijímá vstupy a zobrazuje mu potřebná data. Při sestavování kontejneru se používají tzv multi-stage builds. Pro sestavení aplikace typicky potřebujeme SDK, nicméně pro následné spuštění už stačí jen běhové prostředí. Stejně jsem postupoval i při tvorbě Dockerfile (konfiguračního souboru pro Docker Image). Aplikaci sestavuji ve vrstvě obsahující .NET 5 SDK, nicméně poté zkopíruji jen potřebné binární soubory do vrstvy s běhovým prostředím. Díky tomu je výsledný Docker Image znatelně menší, což urychluje proces nasazení i pozdější aktualizace. (32)

Při návrhu uživatelského rozhraní jsem musel brát v potaz dostupnost na mobilní zařízení. Velkou oporou mi byl framework Blazorise poskytující rozhraní pro detekci velikosti displeje. V nastavení vzhledu komponent lze jednoduše oddělit vzhled pro zařízení s malou a velkou obrazovkou. Některé prvky se tak dají na mobilu skrýt, ale na desktopovém zařízení budou naopak zobrazeny.

Zvolil jsem obvyklé rozložení s responzivní navigací v levé části a horním panelem ukazující název stránky a případně jméno právě přihlášeného uživatele. Na mobilních zařízeních se navigace skryje, ale dá se vyvolat stisknutím tlačítka na horním panelu. Uprostřed se nachází prostor pro vykreslování jednotlivých podstránek (Obrázek 7: Rozložení webové aplikace).



Obrázek 7: Rozložení webové aplikace s responzivní navigací

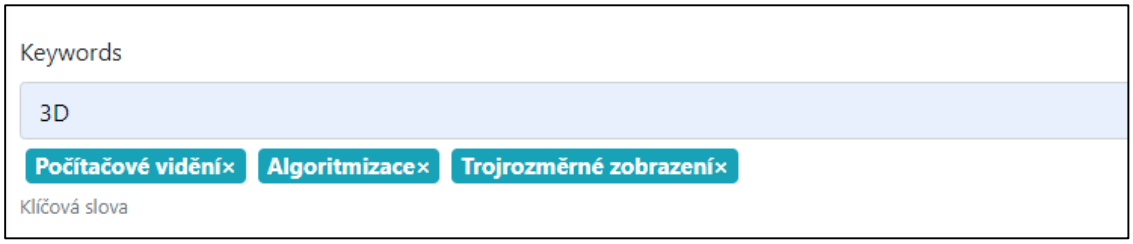
Celý web jsem rozdělil na několik samostatných stránek a komponent. Blazor se při interakci chová velmi podobně jako ostatní SPA[11] frameworky. Tedy ani při navigaci mezi stránkami nenačítá znova celé HTML, ale zpracuje pouze změny v důsledku rozdílu v datech. Navržené rozložení je implementováno v komponentě MainLayout zobrazující vždy namísto výrazu **@Body** právě navštívenou stránku. Kromě toho rovněž označuje příslušnou položku v navigačním menu.

Modální dialogy a jejich vnitřní logiku jsem implementoval jako samostatné komponenty, což usnadňuje jejich znuvupoužitelnost napříč celou aplikací. Pro celkově lepší čitelnost a udržitelnost kódu jsem dále implementoval sadu komplexních komponent. První z nich je vyhledávač dovolující knihovníkovi vybrat osobu z ČVUT nebo hosta (Obrázek 8: Komponenta Person picker). Obsahuje rovněž našeptávač napovídající na základě jména a příjmení. Při velkém počtu jmenovců lze osobu z ČVUT najít přímo pomocí uživatelského jména.

Obrázek 8: Komponenta Person picker poskytující nápovědu při hledání osob

[11] Single page aplikace, typ webové aplikace, který při interakci s uživatelem přepisuje navštívenou stránku namísto načtení nové.

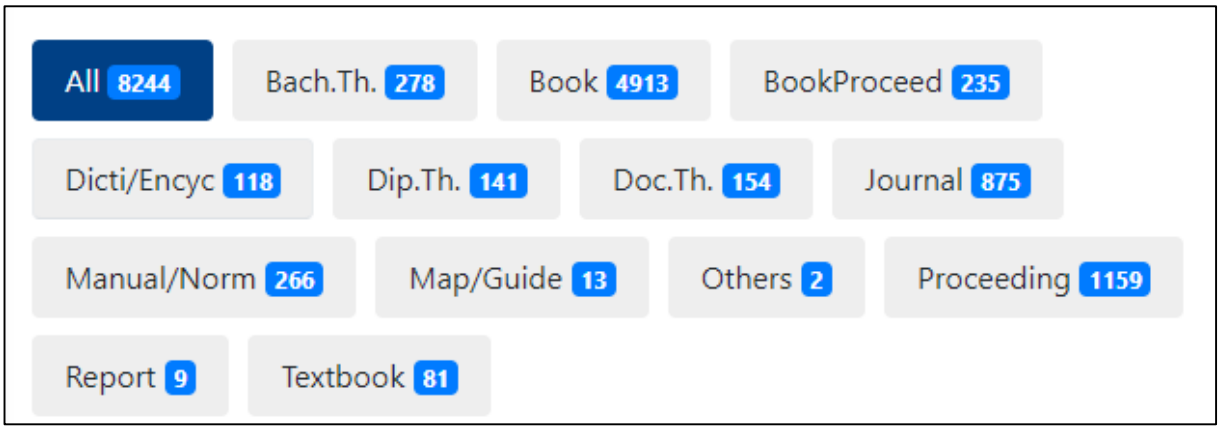
Při zadávání klíčových slov publikace nebo seznamu spoluautorů musí uživatel zadat několik oddělených textových hodnot. Ve frameworku Bootstrap pro to existuje komponenta s názvem Tags Input, ale Blazorise ji bohužel neposkytuje (Obrázek 9: Komponenta Tags Input).



Obrázek 9: Komponenta Tags Input využitá pro klíčová slova

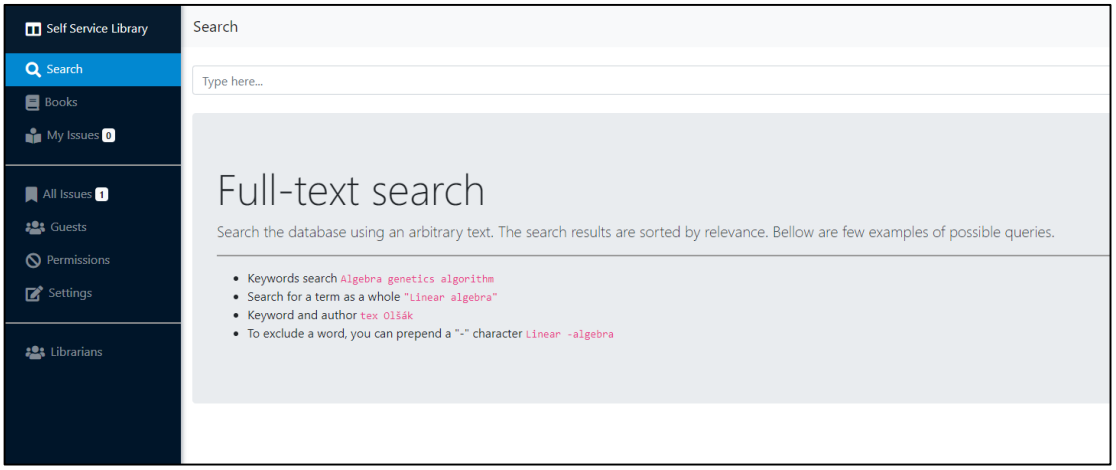
Sice by bylo možné použít přímo HTML a CSS styly z Bootstrapu, nicméně blok kódu by bylo nutné vždy celý okopírovat. Namísto toho jsem komponentu implementoval svépomocí a zachoval v podstatě stejné chování. Při stisknutí enteru se vstup z textového pole přidá jako čtvereček s křížkem. Pří kliknutí na křížek je hodnota ze seznamu naopak odebrána.

Při zobrazování položek z knihovny jsem chtěl umožnit uživateli jejich snadné filtrování a mít přehled o tom, v jakých typech publikací se vyskytují výsledky na zadaný vyhledávací dotaz. K tomu slouží komponenta přijímající slovník, kde klíče jsou názvy záložek a hodnoty představují počet výskytů. Výstupem je uživatelem zvolená záložka (Obrázek 10: Příklad zobrazení v knihovně pomocí komponenty Badged Tabs).



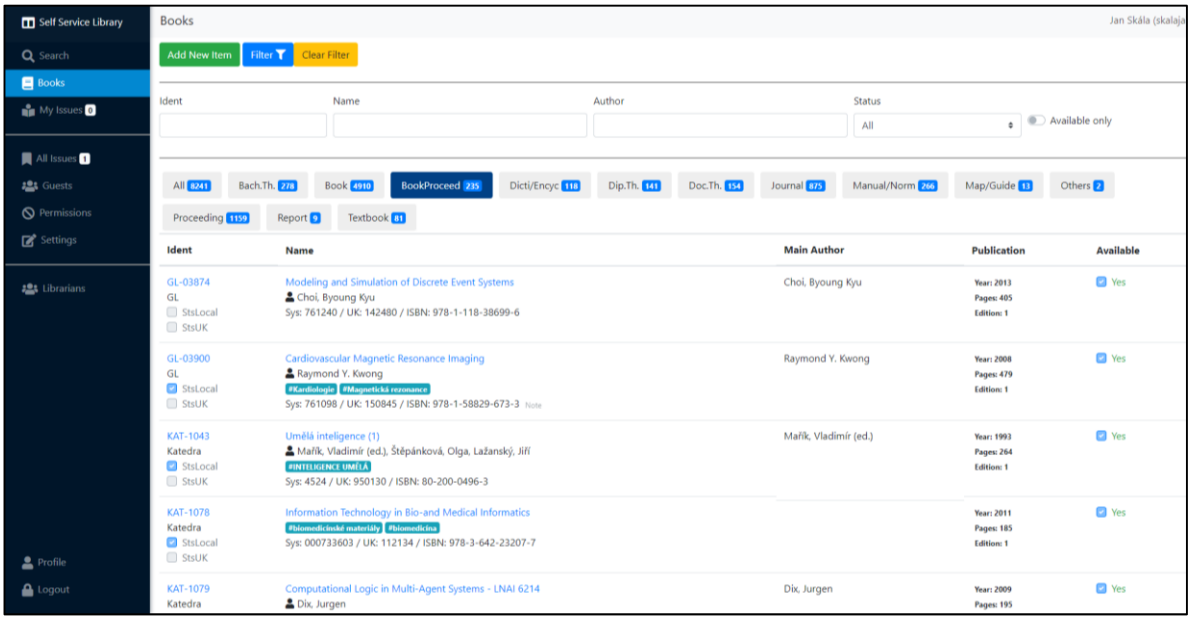
Obrázek 10: Příklad zobrazení v knihovně pomocí komponenty Badged Tabs

Komponenty se využívají na několika stránkách. První takovou stránkou je vyhledávání. Uživatel může vyhledávat mezi publikacemi podle textového řetězce. Výsledky se mu seřadí podle relevance a kategorie výsledků se seřadí sestupně od počtu nálezů. Aplikace hledá podle názvu, autora, spolu autorů, vydavatele, země, konference a klíčových slov. Stránka zároveň ukazuje nápovědu, jak vyhledávání používat (Obrázek 11: Stránka vyhledávání).



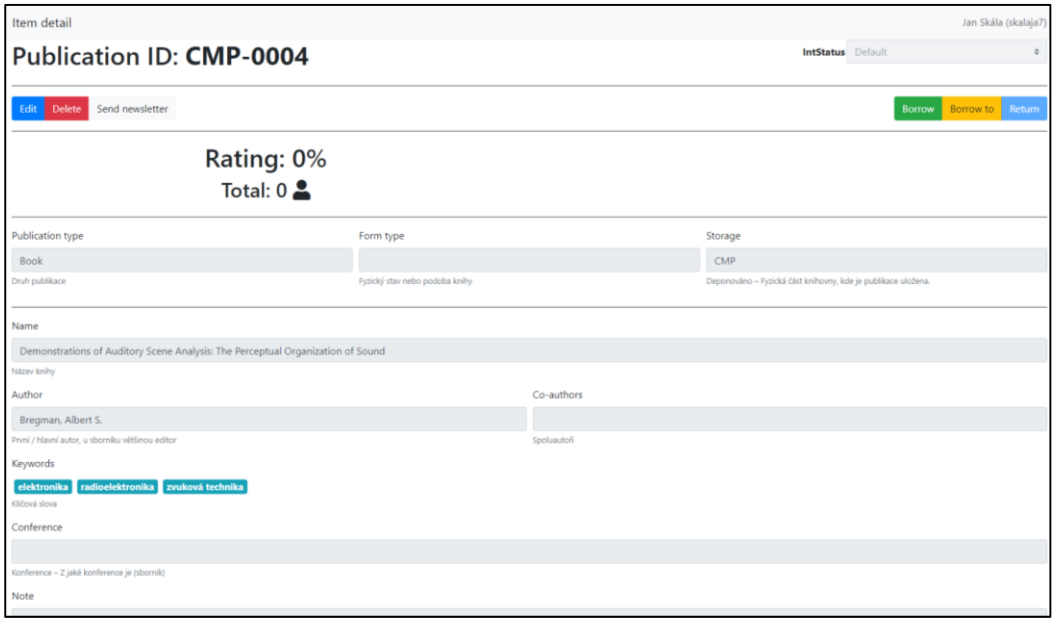
Obrázek 11: Stránka vyhledávání s poskytnutou nápovědou

Předchozí postup není vhodný, pokud hledáme specifickou knihu. V některých případech potřebujeme konkrétní evidenční číslo nebo konkrétního autora. V takové situaci dokáže pomoci stránka s přehledem knih. Nabízí sadu jednoduchých, zato však velmi užitečných, filtrů pomáhajících omezit vyhledávání na konkrétní atributy. Výsledky lze opět třídit podle typu publikace nebo si zobrazit jen dostupné knihy (Obrázek 12: Stránka s přehledem knih).



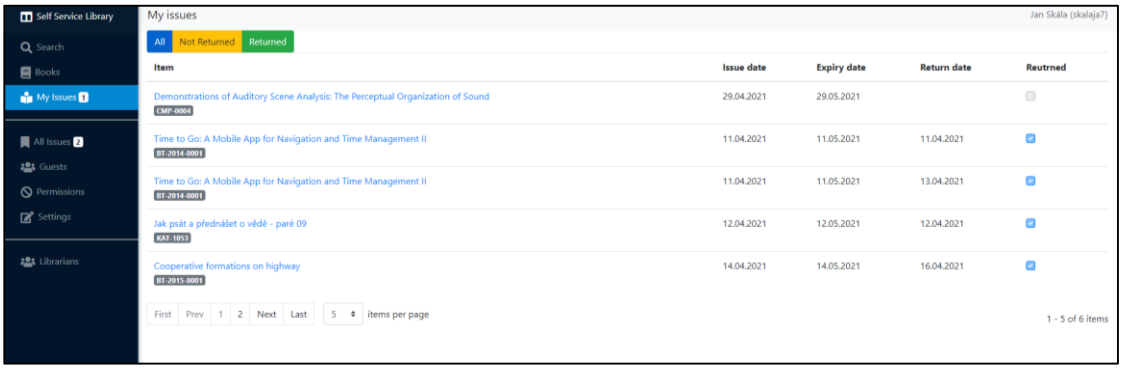
Obrázek 12: Stránka s přehledem knih a možnostmi různého filtrování

Vzhledem k velkému počtu atributů publikace je tabulka nezobrazuje všechny. Zbytek informací je dostupný na stránce s detailem knihy. Právě na té si může uživatel mající příslušná oprávnění, knihu sám půjčit, sám vrátit nebo nastavit hlídání dostupnosti. Knihovník zde naopak může knihu někomu půjčit, upravit data nebo odeslat newsletter informující uživatele o přítomnosti nové knihy v knihovně (Obrázek 12: Stránka s přehledem knih).



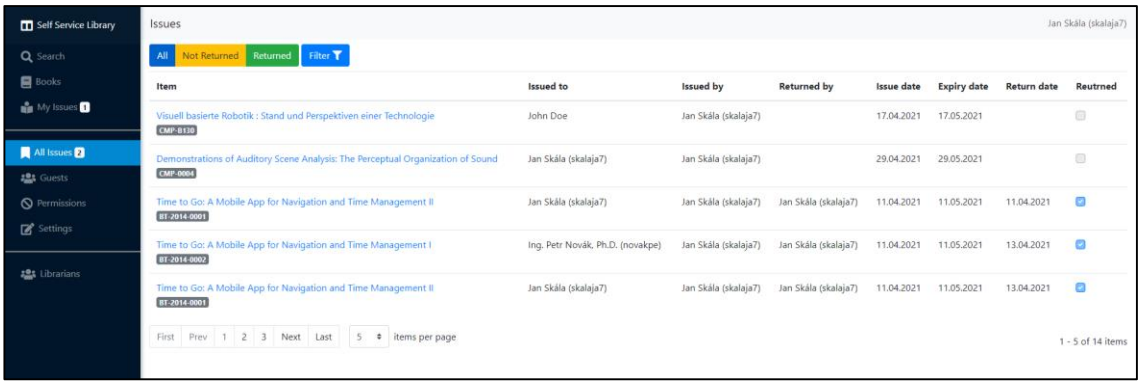
Obrázek 13: Stránka s detailem vybrané knihy

Pokud má uživatel něco vracet, vidí vlevo v menu u položky „My Issues“ počet jím aktuálně vypůjčených publikací. Po kliknutí se zobrazí podrobný přehled všech jeho výpůjček. Může filtrovat podle stavu nebo řadit podle data (Obrázek 14: Stránka **My Issues**).



*Obrázek 14: Stránka **My Issues** poskytující seznam výpůjček uživatele*

Podobným způsobem pracuje i stránka „All Issues“ určená pro knihovníka a obsahující všechny výpůjčky v knihovně. V menu navíc vidí počet nevrácených knih. Může použít stejné filtry jako na předchozí stránce. Kromě toho může filtrovat podle osoby, která knihu půjčila nebo u které se kniha nachází (Obrázek 15: Stránka **All Issues**).

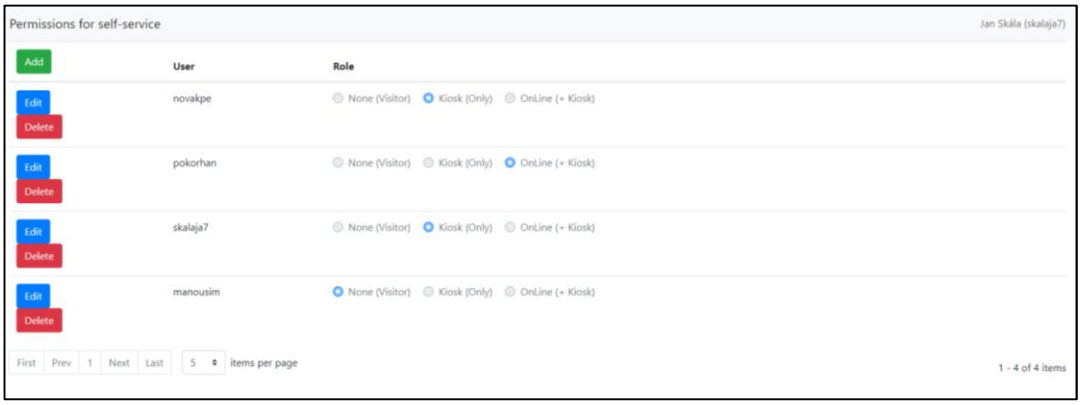


*Obrázek 15: Stránka **All Issues** poskytující seznam všech výpůjček*

K přidávání hostů (osob mimo ČUVT) slouží stránka s přístupem opět pouze pro knihovníka (Obrázek 16: Stránka **Správa hostů**). Při půjčování knihy externistům není nutné vyplňovat jejich údaje pokaždé znovu. Host se přidá pouze jednou a při půjčování knihy se již pouze vybere z vytvořeného seznamu.

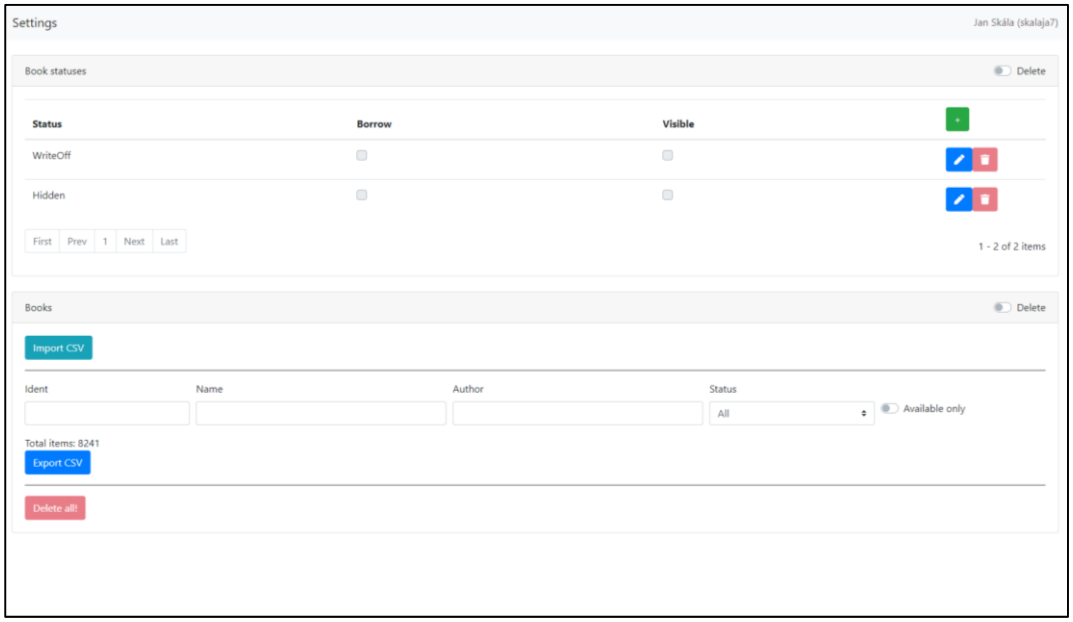
*Obrázek 16: Stránka **Správa hostů** dostupná pouze pro knihovníka*

Další stránka rovněž pouze pro knihovníka umožňuje spravovat oprávnění uživatelů knihovny (Obrázek 17: Stránka **Uživatelská oprávnění**). V tabulce se zobrazují pouze uživatelé, kteří knihovnu alespoň jednou navštívili. K editaci oprávnění stačí znát uživatelské jméno, poté je možné přidělit jednu z dostupných rolí. Pokud uživatel knihovnu zatím nenavštívil a chceme nastavit jeho roli, lze jej do seznamu i ručně přidat.



*Obrázek 17: Stránka **Uživatelská oprávnění** pro nastavení rolí uživatelům*

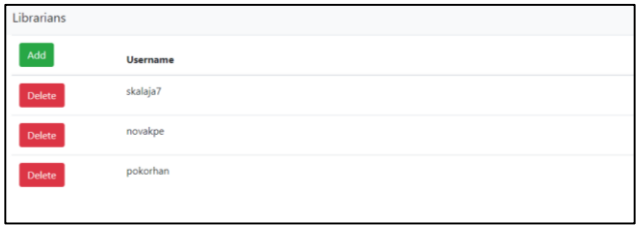
Zbytek úkonů určených pro knihovníka lze uskutečnit na stránce s nastavením. Mezi ty patří manipulace se stavy knih, importování a exportování pomocí CSV souborů a možnost vymazat databázi knih. K té by se měl knihovník uchýlit jen v krajním případě, neboť při importování pomocí CSV souborů dokáže systém aktualizovat stávající záznamy. Pro jistotu je nutné mazání stavů i knih nejprve odemknout pomocí přepínače (Obrázek 18: Stránka obsahující další nastavení).



Obrázek 18: Stránka obsahující další nastavení pro knihovníka

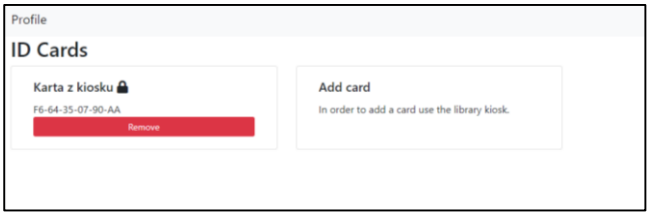
Ve frameworku Blazor není úplně přímočará cesta k tomu, jak stáhnout ze serveru soubor. Naštěstí je možné v rámci jedné webové aplikace kombinovat Blazor i Razor Components. Stránku pro stažení CSV souboru jsem tedy implementoval jako Razor komponentu, ta ověří po návštěvě adresy „/csv/books“, zda je uživatel knihovník a případně spustí stahování souboru.

Pouze administrátor může jmenovat uživatele knihovníkem. K tomu slouží stránka pro správu knihovníků s přístupem pouze pro administrátora. K přidání knihovníka stačí napsat jeho uživatelské jméno do tabulky (Obrázek 19: Stránka **Seznam knihovníků**).



*Obrázek 19: Stránka **Seznam knihovníků** spravovaný pouze administrátorem*

Poslední implementovanou stránkou je uživatelský profil. Každý přihlášený uživatel si zde může zobrazit seznam svých přístupových karet a zde je také případně odebrat. Tato stránka umožňuje i přidání nové karty, a to pouze ve spolupráci s aplikací na kiosku (Obrázek 20: Stránka **Uživatelský profil**).

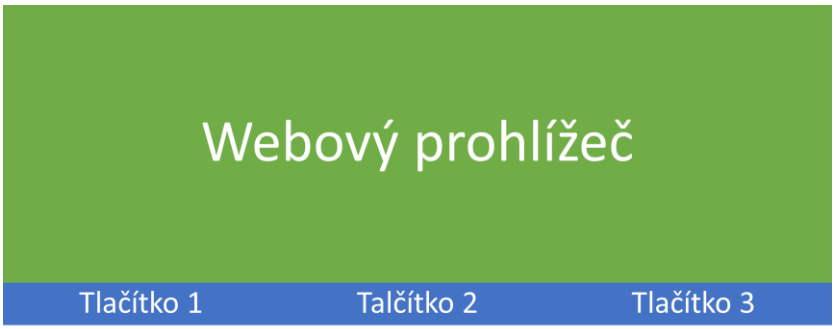


*Obrázek 20: Stránka **Uživatelský profil** obsahující registrované přístupové karty uživatele*

4.2 KIOSK – APLIKACE V MÍSTĚ KNIHOVNY

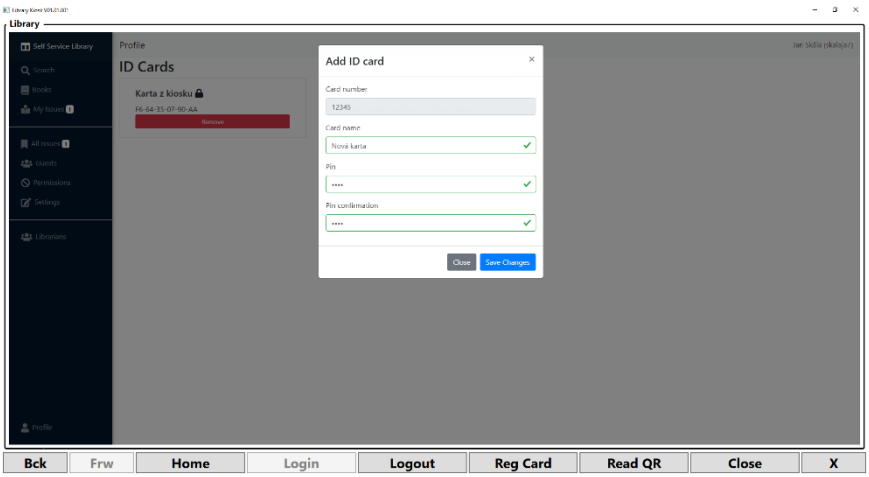
Aplikace na kiosk v knihovně byla tvořena společně s vedoucím práce. Ten poskytl základní části aplikace včetně napojení na NFC čtečku a čtení QR kódu. S hardwarem jsem tedy nemusel vůbec komunikovat přímo, ale využil jsem pouze poskytnuté abstrakce implementující návrhový vzor observer. Ten lze v jazyce C# snadno používat pomocí klíčového slova event.

V rámci této práce jsem implementoval uživatelské rozhraní aplikace a napojení na serverovou část. Do té patří REST API služba a webová aplikace. V horní části se nachází vestavěný webový prohlížeč zobrazující již zmíněnou webovou aplikaci. Všechny ovládací prvky prohlížeče včetně adresního řádku jsou skryty. Ve spodní části se nachází ovládací tlačítka pro kiosk a webovou stránku (Obrázek 21: Rozložení uživatelského rozhraní). Tím jsou myšlena tlačítka pro navigaci v prohlížeči (zpět, vpřed, domů) a tlačítka poskytující funkce kiosku (registrovat kartu, přihlásit, odhlásit, načíst QR kód).



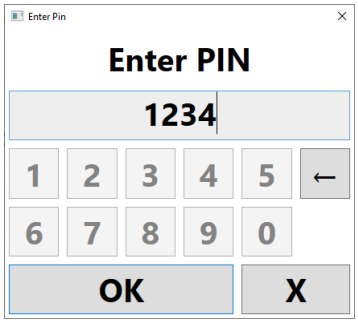
Obrázek 21: Rozložení uživatelského rozhraní

První implementovanou funkcí, pro kterou je vyžadována součinnost aplikace na kiosku, je přidání přístupové karty. Jakmile uživatel klikne na příslušné tlačítko, je z NFC čtečky získána hodnot identifikátoru karty. Následně je uživatel přesměrován na stránku se svým profilem, kde jen zadá pin a potvrdí přidání (Obrázek 22: Dialog pro přidání / registraci přístupové karty).



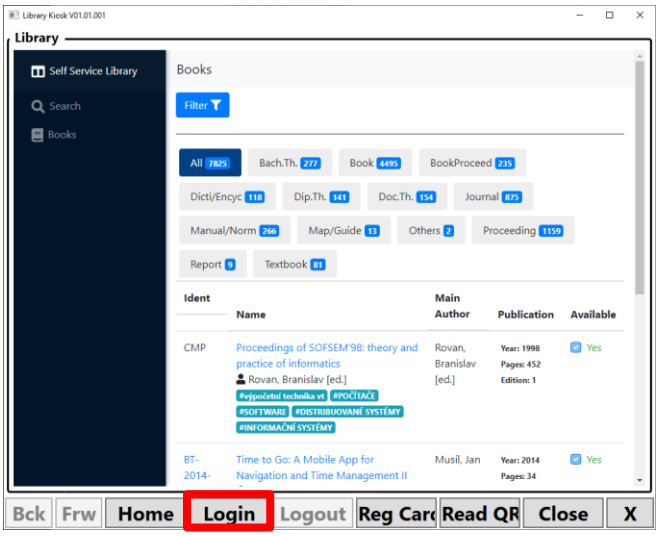
Obrázek 22: Dialog pro přidání / registraci přístupové karty uživatele

Poté jsem implementoval přihlášení, ke kterému lze použít právě dříve přidanou přístupovou kartu. Registrovanou kartu lze opatřit pinem. Byl tedy vytvořen dialog pro zadání tohoto pinu, tak aby byl vhodný i pro dotykové obrazovky (Obrázek 23: Dialog pro zadání pinu).



Obrázek 23: Dialog pro zadání pinu spojeného s registrovanou kartou

Uživatel samozřejmě k přihlášení přístupovou kartu použít nemusí. K vyvolání funkce kioskového přihlášení bylo vytvořeno nové tlačítko, zatímco původní tlačítko bylo ve webové aplikaci skryto. Bylo opatřeno CSS třídou **login-button** skrytou pomocí aplikace na kiosku vyvoláním skriptu ve vestavěném webovém prohlížeči (Obrázek 24: Tlačítko pro přihlášení uživatele v aplikaci na kiosku).

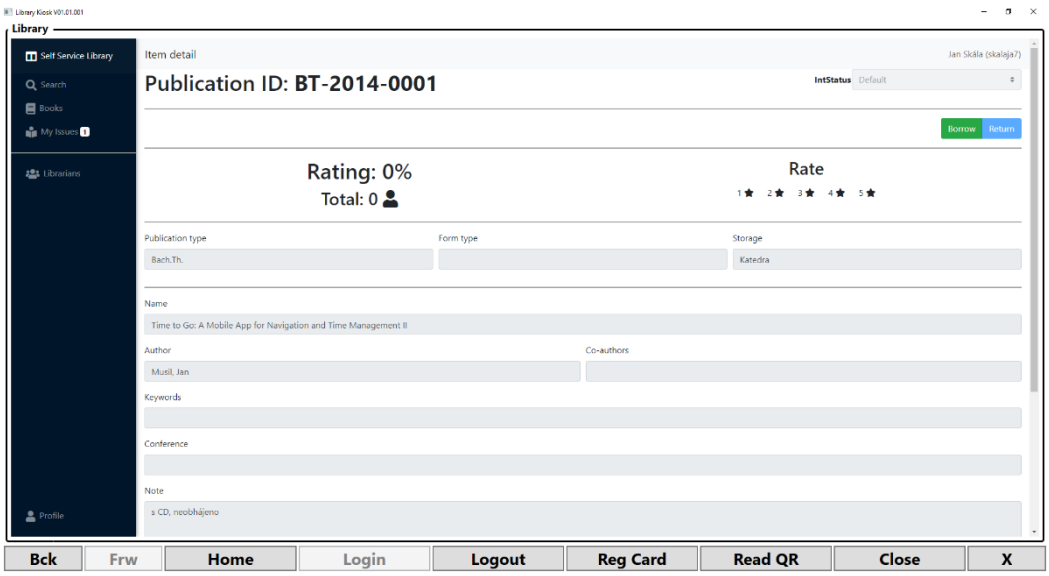


Obrázek 24: Tlačítko pro přihlášení uživatele v aplikaci na kiosku

Podobný způsobem je řešeno i odhlášení. Původní tlačítko ve webové aplikaci je opět skryto a aplikace na kiosku poskytuje k odhlášení vlastní tlačítko. Kromě přesměrování uživatele na stránku /logout pro odhlášení je potřeba ještě vymazat Cookies určené pro stránku autentizačního serveru ČVUT. Toho lze docílit využitím třídy CoreWebView2CookieManager poskytující na práci s Cookies jednoduché rozhraní.

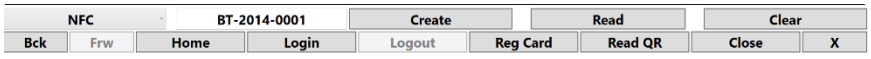
Aplikace na kiosku dokáže pracovat s jednou nebo dvěma NFC čtečkami. V režimu s jednou čtečkou musí uživatel po přihlášení přístupovou kartu odebrat, aby bylo možné načíst knihu. Odhlášení v tomto případě probíhá pouze manuálně, tedy až když uživatel klikne na tlačítko **Logout**. Při zapojení dvou čteček stačí nechat přístupovou kartu přiloženou na jedné z nich. Na druhou čtečku uživatel umísťuje knihy, o kterých má zájem si prohlédnout podrobnější informace nebo si je půjčit. Pro odhlášení uživatele poté stačí přístupovou kartu pouze odebrat.

Pokud uživatel přiloží ke čtečce nějakou knihu a NFC štítek neobsahuje přímo URL adresu pro detail knihy, tak se aplikace pokusí NFC štítek rozpoznat pomocí REST API služby. Při rozpoznání je uživatel přesměrován na stránku s detailem knihy. Pokud má příslušná práva, může si takto knihu rovněž i půjčit (Obrázek 25: Přímé zobrazení detailu knihy pomocí NFC). Místo NFC lze k načtení detailu knihy použít i vytvořený QR kód.



Obrázek 25: Přímé zobrazení detailu knihy pomocí NFC štítku umístěného na publikaci

Pro usnadnění manipulace s NFC štítky a QR kódy, poskytuje aplikace speciální rozšířené rozhraní pro knihovníka. To je odemčeno stisknutím Ctrl (klávesa) + Shift (klávesa) + Home (tlačítko aplikace). Nad tlačítky pro běžného uživatele se navíc zobrazí další ovládací prvky (Obrázek 26 Rozšířený panel).



Obrázek 26 Rozšířený panel aplikace pro správu NFC štítků v režimu knihovníka

Pokud knihovník pracuje s NFC štítkem umožňujícím zápis dat, může stisknutím tlačítka „Create“ provést zápis nejen evidenčního čísla oddělení dané knihy, ale přímo URL adresy pro zobrazení detailu publikace na přiložený NFC štítek. V případě NFC štítku bez možnosti zápisu dat je potřeba nejprve nalézt stránku s detailem knihy. Pokud je knihovník přihlášen, uvidí ve webové aplikaci možnost editace. V režimu editace, pak stačí na kiosku stisknout tlačítko „Read“ a data z NFC štítku (jeho sériové číslo) budou automaticky vepsána do webové stránky. Na závěr je nutné údaje uložit. Podle takto uloženého čísla v detailu publikace poté knihovna dokáže zobrazit kdykoli detail požadované publikace i v případě použití NFC štítku bez možnosti zápisu.

Pro snazší vytváření QR kódů obsahujících URL adresu publikace poskytuje aplikace na kiosku rovněž jednoduché rozhraní. Stačí mít otevřený detail některé knihy a na spodním panelu nastavit možnost „QRCode“ a stisknout tlačítko „Create“. Vytvořený obrázek je automaticky vložen do schránky a může být bez problémů vložen do libovolného obrázkového editoru a vytisknut. K publikaci se nijak neukládá, pouze na ni odkazuje (Obrázek 27: Příklad generování QR kódu). Pro přečtení QR kódu stačí stisknout tlačítko „Read QR“ a nasměrovat jej na kameru umístěnou na kiosku. Při úspěšném přečtení je uživatel přesměrován na stránku s detailem knihy, stejně jako je tomu v případě přiložení NFC štítku.

Obrázek 27: Příklad generování QR kódu pomocí aplikace na kiosku

Na kiosku byly použity následující technologie pro identifikaci knih a uživatelů (Obrázek 28: Použité technologie):

- USB čtečka NFC štítků/karet (vlevo nahoře)
- Nalepovací NFC štítky (vpravo nahoře)
- Plastová přístupová NFC karta (vlevo dole)
- Obrázek s QR kódem (vpravo dole)

Obrázek 28: Použité technologie usnadňující (samoobslužný) provoz knihovny

4.3 TESTOVÁNÍ

Ke každé softwarové implementaci neodmyslitelně patří testování. Pro tvorbu testů jsem využil framework xUnit. Ten umožňuje jednotlivé testy parametrizovat, což ušetří psaní kódu. Zároveň může vytvořit testovací prostředí pro jeden nebo více testů pomocí objektu s názvem fixture. (33)

V testech jsem také navíc použil programovou knihovnu s názvem Fluent Assertions. Její syntaxe je dobře čitelná a ze zdrojových kódu testů je tedy mnohem jasnější záměr. Kormě toho poskytuje

metody pro běžné testovací scénáře podporující porovnávání kolekci, struktury objektů nebo datumů. (34)

Některé prvky se nedají v testech použít přímo. Nelze například v rámci testování rozesílat emaily uživatelům. Činnost komponenty, na kterou testovaná část spoléhá, lze napodobit a implementovat jen metody použité při testu. Tato technika se nazývá mocking a realizoval jsem ji pomocí programové knihovny Moq. Ta dokáže implementovat podmnožinu metod nějakého rozhraní nebo abstraktní třídy a následně ověřit, zda nebo kolikrát byly zavolány.

4.3.1 Testy mapování

Pro mapování jsem použil framework Automapper. Při úpravě entit a přidávání nebo odebírání atributů, se může snadno opomenout nějaký atribut v DTO. Syntaxe zdrojového kódu je sice stále validní, ale při spuštění program skončí běhovou výjimkou. Mapování lze otestovat při startu aplikace nebo v testech. K tomu slouží metoda AssertConfigurationIsValid signalizující výjimku při neúspěchu obsahující popis chyby v mapování (Obrázek 29: Výpis Automapperu).

```
Unmapped members were found. Review the types and members below.
Add a custom mapping expression, ignore, add a custom resolver, or modify the source/destination type
For no matching constructor, add a no-arg ctor, add optional arguments, or map all of the constructor parameters
===============================================================================
Book -> BookDetailDTO (Destination member list)

Unmapped properties:
ReviewsAvg
```

Obrázek 29: Výpis Automapperu při detekci nesprávného mapování

4.3.2 Testy infrastruktury

Zde vytvářené řešení spoléhá na nemalé množství závislostí. Mezi ty patři programové knihovny třetích stran a externí systémy. Vytvořil jsem testy pro všechny komponenty infrastruktury, kromě autentizace pomocí přístupových karet. To je testováno společně se zbytkem aplikační vrstvy. Při testování komunikace s externími systémy, se testy snaží odhalit, zda systémy nezměnily své komunikační rozhraní nebo formát odpovědi. Například programová knihovna používaná pro import a export CSV souborů má rozhraní stále stejné. V tomto případě se porovnává hodnota ze souboru vygenerovaného testy s referenčními hodnotou. Integrace s emailovým serverem na pozadí využívá již hotového SMTP klienta není jej tedy potřeba testovat. Nicméně šablonování emailů testováno je.

4.3.3 Testy aplikační logiky

K otestování aplikační logiky je potřeba mít spuštěnou instanci databáze MongoDB. Stejně jako v případě nasazení do provozu jsem využil Docker. Aby bylo možné spouštět testy paralelně, musí být izolované. Nicméně tvořit nový kontejner s databází pro každý test, by bylo velmi nákladné. Seskupil jsem tedy testy vhodně tak, aby mezi sebou vždy sdílely instanci databáze. Nicméně testy v rámci skupiny si vzájemně svá data neovlivňují. Jsou otestovány všechny důležité procesy v aplikaci. Mezi ty patří půjčení a vrácení knihy, přidání přístupové karty a následné přihlášení.

4.3.4 Načítání konfigurace

Některé testy vyžadují načtení konfigurace. Ta může obsahovat přístupové údaje k externím systémům, a tudíž není vhodné, aby byly součástí zdrojových kódů. Všechny testy načítají konfiguraci z proměnných prostředí. Díky tomu se dají spustit lokálně nebo v pipeline při použití CI/CD.

5 ZÁVĚR

V této práci byla rozebrána problematika činnosti menší katederní knihovny umožňující rovněž i různé stupně samoobslužného provozu. Nejprve jsem uskutečnil pozorování současného stavu na katedře kybernetiky. Následně jsem společně s vedoucím práce formuloval seznam požadavků na činnost zde vytvářeného systému. Z těch vyplynuly uživatelské role v systému, diskrétní kontrola přístupu a autentizační schémata. Poté jsem definoval systémové entity nacházející se v systému. Nejdůležitější uživatelské úkony jsem zmapoval jako procesy.

V další části jsem zkoumal způsoby autentizace a autorizace v rámci instituce ČVUT. Byly srovnány dvě dostupné možnosti: Shibboleth a Zuul. Z těch jsem zvolil autorizační server Zuul implementující protokol Oauth 2.0. Byl vyvinut na katedře informatiky s dostupností pro celé ČVUT. K získávání dat uživatelů používám službu Usermap pro vyhledávání studentů nebo zaměstnanců ČVUT.

Poté se věnuji porovnání již existujících open-source řešení. Srovnal jsem tři systémy: Invenio, BiblioteQ a Evergreen. Ze všech byl nejvíce perspektivní systém Invenio, ale bohužel je stále ve značném stádiu vývoje a neumožňuje integraci s externím poskytovatelem identity. Proto jsem se rozhodl ve svém řešení žádný nepoužít.

Během analýzy jsem zjistil nutnost integrace s externími systémy. Již jenom kvůli autentizaci potřebuji komunikovat se systémy Zuul a Usermap. Mezi další integrované systémy patří Google Books pomáhající před-vyplnit informace o knize. Při vyhledávání knih umí vytvořený systém hledat i v Ústřední knihovně. Emaily notifikující uživatele lze odesílat přes poštovní server ČVUT. V závěru analýzy jsou srovnány typy NFC štítků a možnost použití QR kódu.

Poznatky z analýzy jsem použil k návrhu celkového řešení. Pro databázi jsem zvolil technologii MongoDB, zejména kvůli možnosti dynamického schématu dat. Stejně tomu bylo i v případě full-textového vyhledávání, kde jsem ji upřednostnil před službou Elasticsearch kvůli nižším hardwarovým nárokům.

Při formování aplikace jsem se držel 3 vrstvé softwarové architektury. S databází komunikuje datová vrstva díky třídě MongoDbContext, poskytující jednoduchou abstrakci na práci s daty. Veškerá důležitá logika, validace a potřebná rozhraní jsou definovány v aplikační vrstvě. Tu konzumuje vrstva prezentační zobrazující uživateli požadovaná data a přijímající jeho vstupy. Zbytek kódu leží v sekci infrastruktura. Ta zajišťuje napojení na externí systémy nebo adaptaci rozhraní knihoven třetích stran.

Výstupem práce jsou celkem 4 aplikace. První a nejdůležitější je webová aplikace vytvořená ve frameworku Blazor. Ta je zobrazena v koncových zařízení uživatelů a na kiosku v knihovně. Druhá aplikace instalovaná na kiosku je WPF aplikace obsahující webový prohlížeč a pomáhá při práci s externím hardwarem. Ta byla vytvořena ve spolupráci s vedoucím práce, který zajistil jádro celé aplikace, a to včetně komunikace s NFC čtečkami a kamerou pro načítání QR kódů. Kiosk také komunikuje s REST API službou, tedy malou webovou aplikací pomáhající s autentizací pomocí přístupových karet a hledáním knih s použitím NFC štítků. Poslední čtvrtou aplikací je služba na pozadí připomínající uživatelům termín vrácení knihy.

5.1 PROBLÉMY PŘI IMPLEMENTACI

Během implementace se objevily nečekané komplikace, jež nebyly patrné při návrhu. První je částečná nekompatibility rozhraní knihoven Automapper a MongoDB Driver. Automapper při projekci dat vrací rozhraní IQueryable, ale MongoDB Driver pro asynchronní komunikaci s databází vyžaduje rozhraní IMongoDbQueryable. Při mapování je tedy vývojář nucen komunikovat s databází synchronně. Během reverzního inženýrství obou knihoven jsem zjistil, skutečnost, kdy Automapper rozhraní nijak nepřetváří, ale pouze zakryje původní implementaci. Problém jsem tedy vyřešil vytvořením vlastní extension metody. Ta přetypuje vrácené rozhraní zpět na IMongoDbQueryable.

Při načítání dat z NFC čtečky a jejich zápisu do webové stránky zobrazované na kiosku v prohlížeči nedocházelo k odeslání uložených dat na server. Je to způsobeno tím, jak Blazor pracuje. Všechny změny stavu jsou detekovány, pouze pokud uživatel vykoná nějakou akci. Nicméně data ze čtečky jsou vložena do stránky pomocí kódu v Javascriptu. Řešením bylo zavolat stejnou událost volanou jako při akci uživatele a tím došlo k odeslání dat na server.

Řešení jsem průběžně nasazoval na server katedry kybernetiky. Na ten však neodkazoval žádný DNS záznam a na zprovoznění reverzní proxy se stále čekalo. Zpočátku to vůbec nevadilo, neboť jsme na webovou stránku mohli přistupovat zadáním IP adresy do prohlížeče. Tímto způsobem jsme však mohli používat jen protokol HTTP a nikoli HTTPS. Jakmile jsem však integroval autentizaci, tak prohlížeč začal zahazovat Cookies s nastavením SameSite=None, jež nepřišly přes HTTPS. Abych mohl s vedoucím sdílet aktuální verzi řešení, vytvořil jsem testovací instanci v prostředí cloudu Microsoft Azure. Všechny aplikace jsem vytvořil jako App Service a použil stejné kontejnery. Perzistentní uložiště ve formě MongoDB databáze, jsem se dočasně rozhodl provozovat ve službě MongoDB atlas, neboť ta je při malém využití bezplatná. Vedoucí tak mohl snáze poskytnout zpětnou vazbu a ukázat řešení i dalším lidem pro vznášení připomínek.

5.2 MOŽNÁ VYLEPŠENÍ

Při zavádění řešení na malé katedry není problém ručně přidat knihovníky nebo povolit uživatelům samoobslužný provoz. Nicméně na větších katedrách by to obnášelo spoustu práce. Řešením by bylo definování sady pravidel, jež na základě příslušnosti uživatele k roli v systému Usermap přiřadí uživatele do role v aplikaci. Šlo by například definovat, že zaměstnancům katedry bude automaticky povolen samoobslužný provoz.

Pokud si uživatel na stránce s knihami zvolí největší velikost stránky (250 knih), klientovi již trvá poněkud delší dobu vykreslit tabulku. Data ze serveru se sice vrátí rychle, ale klient zbytečně vykresluje i řádky nacházející se pod hranicí monitoru, i když je uživatel v danou chvíli nemůže v podstatě ani vidět. Ve frameworku Blazor lze klienta optimalizovat zabalením položek do komponenty Virtualize. Ta zajistí vykreslení jen viditelných komponent. Nicméně použitá programová knihovna Blazorise a komponenta DataGrid tuto optimalizaci neumí používat. Autoři plánují podporu přidat v dalších verzích.

6 SLOVNÍK POJMŮ A ZKRATEK

Pojem / zkratka	Vysvětlení
CRUD	Zkratka pro Create Read Update Delete
NFC	Near field communication (NFC) je modulární technologie bezdrátové rádiové komunikace na velmi krátkou vzdálenost (do 4 cm)
QR kód	Strojově čitelný obrázek s bodovou maticí obsahující zakódovaná data.
API	Application Programming interface Rozhraní sloužící pro komunikaci. Zapouzdřuje složitější funkcionality za jednodušší metody a procedury, typicky CREATE, READ, UPDATE, DELETE.
REST	Representational State Transfer Implementuje základní operace (CRUD) pomocí metod HTTP protokolu (GET, POST, PUT, DELETE).
Framework	Softwarový balík usnadňující programování. Má větší rozsah než programová knihovna. Typicky zapouzdřuje a řeší komplexní problém.
ORM	Objektově relační mapování Zajišťuje automatickou konverzi dat mezi (relační) databázi a objekty z aplikační domény.
DOM	Objektový model dokumentu je objektově orientovaná reprezentace HTML stránky.
DTO	Data transfer object – Přenáší data mezi procesy nebo vrstvami aplikace. Nemá žádnou vnitřní funkcionalitu.

7 SEZNAM OBRÁZKŮ

8 SEZNAM LITERATURY A POUŽITÝCH ZDROJŮ

1. Shibboleth Wiki. *Shibboleth.* [Online] [Citace: 4. 1 2021.] https://wiki.shibboleth.net/confluence/#all-updates.

2. OAuth 2.0. *Rozvoj FIT.* [Online] 4. 1 2021. https://rozvoj.fit.cvut.cz/Main/oauth2.

3. Invenio ILS Documentation. *Invenio ILS.* [Online] [Citace: 4. 1 2021.] https://invenioils.docs.cern.ch/.

4. BiblioteQ Dokumentace. *biblioteq.sourceforge.io/.* [Online] [Citace: 3. 1 2021.] https://biblioteq.sourceforge.io/.

5. Evergreen Documentation. *Evergreen ILS.* [Online] Evergreen. [Citace: 3. 1 2021.] https://docs.evergreen-ils.org/eg/docs/latest/index.html.

6. Usermap. *ČVUT Informační systémy a technologie.* [Online] [Citace: 5. 4 2021.] https://ist.cvut.cz/nase-sluzby/usermap/.

7. Jirůtka, Jakub. Usermap API. *Wiki FIT Čvut.* [Online] 5. 4 2021. https://rozvoj.fit.cvut.cz/Main/usermap-api.

8. Books APIs. *Google Developers.* [Online] [Citace: 14. 4 2021.] https://developers.google.com/books/docs/v1/using.

9. Katalogy a databáze. *České vysoké učení technické v Praze Ústřední knihovna.* [Online] [Citace: 17. 4 2021.] http://knihovna.cvut.cz/katalogy-a-databaze.

10. Vyhledávač SUMMON. *České vysoké učení technické v Praze Ústřední knihovna.* [Online] [Citace: 17. 4 2021.] http://knihovna.cvut.cz/vyhledavac-summon.

11. Simple Mail Transfer Protocol. *IETF Tools.* [Online] [Citace: 18. 4 2021.] https://tools.ietf.org/html/rfc5321.

12. Fowler, Martin. *Patterns of Enterprise Application Architecture.* 2012.

13. What is Elasticsearch. *Elastic.* [Online] [Citace: 19. 4 2021.] https://www.elastic.co/what-is/elasticsearch.

14. MongoDB C#/.NET Driver. *MongoDB Documentation.* [Online] [Citace: 20. 4 2021.] https://docs.mongodb.com/drivers/csharp/#mongodb-c--.net-driver.

15. Prevent over-posting. *Microsoft documentation.* [Online] [Citace: 22. 4 2021.] https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.identity.dataprotectortokenprovider-1?view=aspnetcore-5.0.

16. de la Torre, Cesar, Wagner, Bill a Rousos, Mike. *.NET Microservices: Architecture for Containerized .NET Applications.* místo neznámé : Microsoft Developer Division, .NET and Visual Studio product teams, 2021.

17. SignInManager<TUser> Class. *Microsoft documentation.* [Online] [Citace: 21. 4 2021.] https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.identity.signinmanager-1?view=aspnetcore-5.0.

18. DataProtectorTokenProvider<TUser>. *Microsoft documentation.* [Online] [Citace: 21. 4 2021.] https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.identity.dataprotectortokenprovider-1?view=aspnetcore-5.0.

19. Documentation. *Automapper.* [Online] [Citace: 23. 4 2021.] https://docs.automapper.org/en/stable/#.

20. Blazor. *.NET.* [Online] [Citace: 24. 4 2021.] https://dotnet.microsoft.com/apps/aspnet/web-apps/blazor.

21. *WebAssembly.* [Online] [Citace: 24. 4 2021.] https://webassembly.org.

22. Blazor Server. *Microsoft documentation.* [Online] [Citace: 24. 4 2021.] https://docs.microsoft.com/en-us/aspnet/core/blazor/hosting-models?view=aspnetcore-5.0#blazor-server.

23. What is Blazor. *Blazor.* [Online] [Citace: 24. 4 2021.] https://blazorise.com/docs/faq/#what-is-blazorise.

24. Overview of ASP.NET Core authentication. *Microsoft documentation.* [Online] [Citace: 25. 4 2021.] https://docs.microsoft.com/en-us/aspnet/core/security/authentication/?view=aspnetcore-5.0.

25. Authenticate with an OpenID Connect or OAuth 2.0 Identity provider. *Microsoft documentation.* [Online] [Citace: 25. 4 2021.] https://docs.microsoft.com/en-us/dotnet/architecture/microservices/secure-net-microservices-web-applications/#authenticate-with-an-openid-connect-or-oauth-20-identity-provider.

26. Otp.NET. *GitHub.* [Online] [Citace: 25. 4 2021.] https://github.com/kspearrin/Otp.NET.

27. Use cookie authentication without ASP.NET Core Identity. *Microsoft documentation.* [Online] [Citace: 25. 4 2021.] https://docs.microsoft.com/en-us/aspnet/core/security/authentication/cookie?view=aspnetcore-5.0.

28. Introduction to Microsoft Edge WebView2. *Microsoft documentation.* [Online] [Citace: 26. 4 2021.] https://docs.microsoft.com/en-us/microsoft-edge/webview2/.

29. What is a Container. *Docker.* [Online] [Citace: 28. 4 2021.] https://www.docker.com/resources/what-container.

30. Overview of Docker Compose. *Docker.* [Online] [Citace: 28. 4 2021.] https://docs.docker.com/compose/.

31. Docker-compose basic example. *Traefik.* [Online] [Citace: 28. 4 2021.] https://doc.traefik.io/traefik/master/user-guides/docker-compose/basic-example/.

32. Use multi-stage builds. *Docker.* [Online] [Citace: 29. 4 2021.] https://docs.docker.com/develop/develop-images/multistage-build/.

33. Documentation. *xUnit.* [Online] [Citace: 1. 5 2021.] https://xunit.net/#documentation.

34. Introduction. *Fluent Assertions.* [Online] [Citace: 1. 5 2021.] https://fluentassertions.com/introduction.

35. Swagger API. [Online] [Citace: 17. 1 2019.] http://swagger.io/.