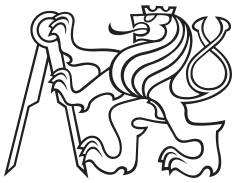
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

Integrace KOS na sociální media a další digitální kanály

David Stražovan

Vedoucí: Ing. David Kadleček, Ph.D.
Květen 2020

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Stražovan** Jméno: **David** Osobní číslo: **457796**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Integrace KOS na sociální media a další digitální kanály

Název bakalářské práce anglicky:

KOS integration to social media and other digital channels

Pokyny pro vypracování:

Vytvořte aplikaci pro sledování relevantních změn v systému KOS a jejich notifikování za použití digitálních kanálů. Postupujte následovně:
1. Seznamte se s rozhraním KOSApi.
2. Udělejte rešerši existujících řešení.
3. Navrhněte datový model aplikace a funkcionality.
4. Implementujte serverovou část aplikace.
5. Otestujte integraci na KOSApi a funkcionality vůči testovacím datům.
6. Implementujte klientskou část aplikace.
7. Otestujte integraci klientské a serverové části.

Seznam doporučené literatury:

[1] Martin Fowler – Patterns of enterprise application architecture. ISBN: 0-321-12742-0
[2] Brian Goetz – Java Concurrency in Practice. ISBN: 0-321-34960-1
[3] Adam Freeman - Essential TypeScript: From Beginner to Pro. ISBN: 978-1-4842-4978-9

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. David Kadleček, Ph.D., Centrum znalostního managementu FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **14.02.2020** Termín odevzdání bakalářské práce: **22.05.2020**

Platnost zadání bakalářské práce: **30.09.2021**

Ing. David Kadleček, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

.	
Datum převzetí zadání	Podpis studenta

Poděkování

Chtěl bych poděkovat svému vedoucímu bakalářské práce Ing. Davidu Kadlečkovi, Ph.D. za cenné rady, připomínky a vstřícnost při zpracování této práce.

Prohlášení

Prohlašuji, že jsem predloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje.

V Praze, 20. 5. 2020

Abstrakt

Tato práce se zabývá integrací školního systému KOS na různé komunikační a sociální kanály. Cílem je analýza a implementace systému, který studentům umožní jednododuché přihlášení školním účtem, nastavení upozornění na změny v KOS a jejich zasílání pomocí různých kanálů. V rámci této práce vznikne funkční serverová strana aplikace, která stahuje data z KOSApi, porovnává je s předchozími daty a v případě změny notifikuje uživatele. Dále vznikne klientská část aplikace umožňující přihlášení přes školní SSO a konfiguraci notifkací a odběrů změn dat. Cílem je také to, aby aplikace byla flexibilní, jednoduše rozšiřitelná a bylo možné ji použít i na jiné systémy, tj. je nezávislá na KOSApi a jeho modelu.

Klíčová slova: KOS, KOSApi, Java, Spring, React, TypeScript, integrace, notifikace

Vedoucí: Ing. David Kadleček, Ph.D.
Technická 2 ČVUT FEL
Praha 6, 166 27

Abstract

This work deals with the integration of the school system KOS into various communication and social channels. The goal is analysis and implementation of a system that allows students to easily log in with their school account, set up notification of changes in KOS and notifying via various channels. As part of this work, functional server side of the application, which downloads data from KOSApi, compares it with previous data and in case of change it notifies the user, will be implemented. Furthermore, the client part of the application will be created, enabling login via the school SSO and configuring notifications and subscriptions to data changes. The goal is also to make the application flexible, easily extensible so it can be used on other systems as well, ie it is independent of KOSApi and its model.

Keywords: KOS, KOSApi, Java, Spring, React, TypeScript, integration, notifications

Title translation: KOS integration to social media and other digital channels

Obsah

Obrázky

Tabulky

Kapitola 1
Úvod

První myšlenka na tento projekt se zrodila hned před prvním zkouškovým obdobím. Ráno přijde studentovi e-mail o otevření nových termínů zkoušky a když se podívá, tak jsou již zaplněné, jelikož k jejich otevření došlo ve skutečnosti již o den dříve. Nejedná se ovšem o jediný případ, kdy pozdní notifikace může způsobit studentům problémy. Jsou zde i další změny a události, pro které notifikace ze systému KOS ani neexistují. Příkladem může být otevření nové paralelky pro zapsaný předmět, jednorázové akce nebo změna kapacity parelelky předmětu.

Dalším tématem k řešení je kanál, po kterém notifikace přicházejí – e-mail. I častější notifikace přes e-mail by nemusely dosáhnout zamýšleného výsledku, jelikož ne každý kontroluje e-mail tak často a nebo nemá zaplé notifikace. Naopak v dnešní době studenti používají jiné komunikační kanály, které jsou více "real-time". Zde může být příkladem Slack, Discord nebo Facebook (a Facebookové skupiny). Možností může být i využití notifikační platformy jako je třeba Pushover[1].

Vyvíjená aplikace může nejen studentům, ale i učitelům, umožnit rychleji reagovat na změny ve studijním systému, ať už se jedná o přihlašování na zkoušky, do paralelek předmětů či na jejich zaplnění nebo uvolnění místa. Pro učitele nemusí být důležitá rychlost notifikace, ale vůbec její existence a že nemusí kontrolovat stavy sami.

[1] https://pushover.net/

Kapitola 2
Zadání

Zadáním práce je návrh a implementace systému umožňující sledování změn v systému KOS pomocí KOSApi a následnou notifikace uživatelů. Uživatelé budou mít možnost vybrat si o jakých událostech a jak chtějí být notifikování. Systém bude navržen tak, aby nebyl závislý na KOSApi a na jeho modelu a měl tak možnost napojení na další systémy bez nutnosti změny v datovém modelu. V rámci bakalářské práce bude realizována serverová strana aplikace, napojení na KOSApi a notifikace pomocí e-mailu a jednoduchá notifkace přes Slack. Dále bude implementována klientská strana aplikace, do které bude možné se přihlásit za použití školního SSO. Klient bude umožňovat konfiguraci notifikací a odběrů.

2.1 Cíle

Cílem je, v rámci bakalářské práce, implementace následujících funkcionalit:

- Přihlášení přes školní SSO.
- Uživatelská konfigurace notifikačních kanálů.
- Stažení dat a notifikace relevantních změn v systému KOS:
  1. Studijní výsledky studenta.
  2. Počet zkoušek a informace o jednotlivých zkouškách v předmětu.
  3. Počet a informace o jednorázových akcích předmětu.
  4. Počet a informace o jednotlivých pararelkách předmětu.
  5. Informace o jednotlivých předmětech.
- Administrační rozhraní.

Kapitola 3

Analýza

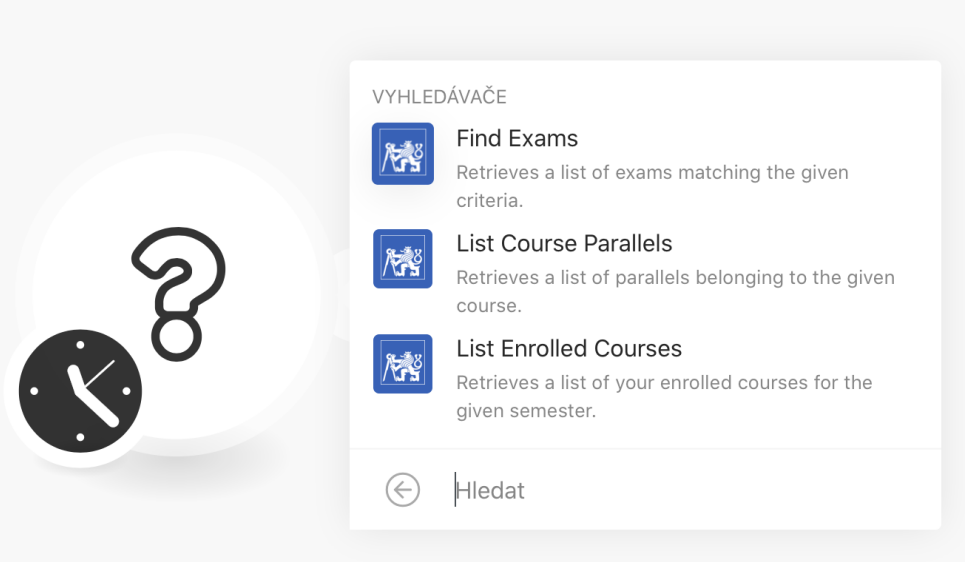
3.1 Existující řešení

Služeb pro integraci na jiné aplikace a následné provádění různých akcí (nejen notifikací) je dostupných mnoho. Příkladem mohou být Integromat[1], automate.io[2], Zapier[3] nebo Microsoft Power Automate[4]. Nevýhodou předchozích, vyjma Microsoft Power Automate, je absence out-of-the-box účtů pro studenty. Jediné řešení, které v kapitole bude více probrané je Integromat, jelikož v něm, jako jediném, již jiný autor připravil jednoduchou intergaci na KOSApi.

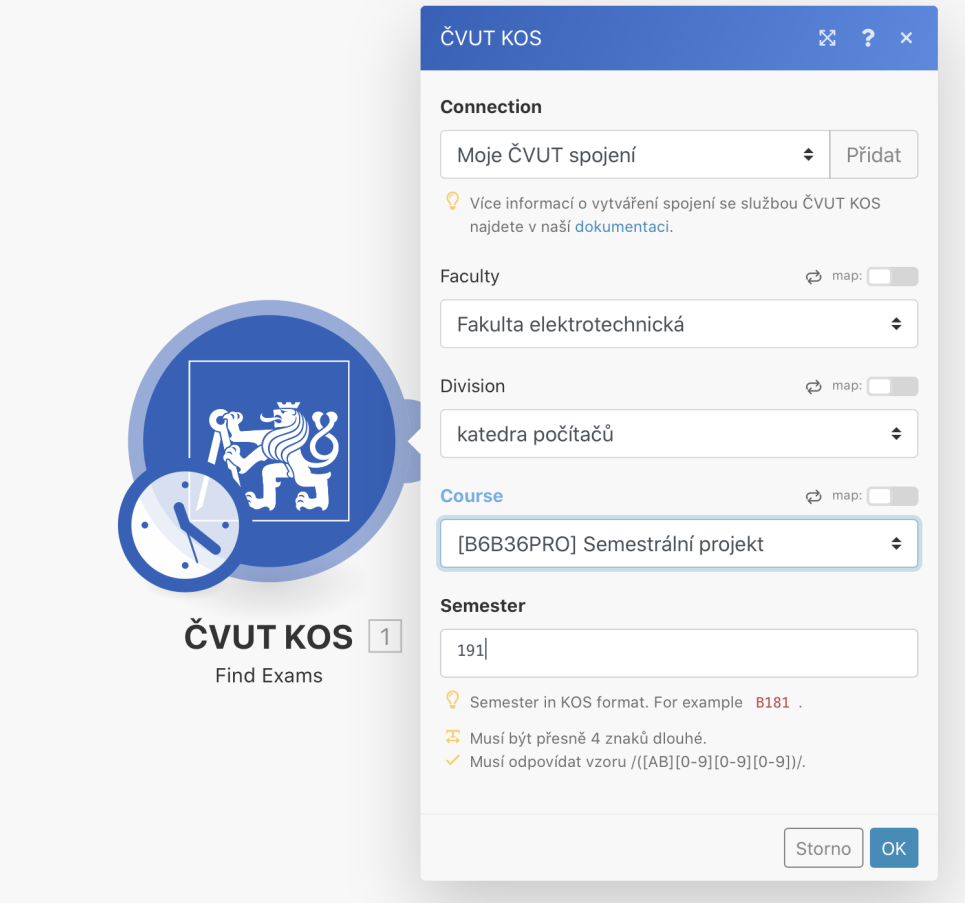
3.1.1 Integromat

Velkou výhodou služby Integromat je její jednoduchost. Jedná se v podstatě o WYSIWYG[5] tvorbu integračních scénářů. Služba je do jisté míry zdarma s omezeními na počet operací za měsíc, minimální interval atd. Nevýhodou je, že existující integrace na KOSApi poskytuje omezené množství operací (viz obrázek 3.1) a ty posílají vždy všechna data, ne jen změny. Na obrázku 3.2 je ukázka nastavení operace pro stažení seznamu zkoušek pro jeden předmět. Následující obrázek 3.3 ukazuje možnost nastevení odeslání stažených dat přes Gmail a konfiguraci intervalu spuštění scénáře.

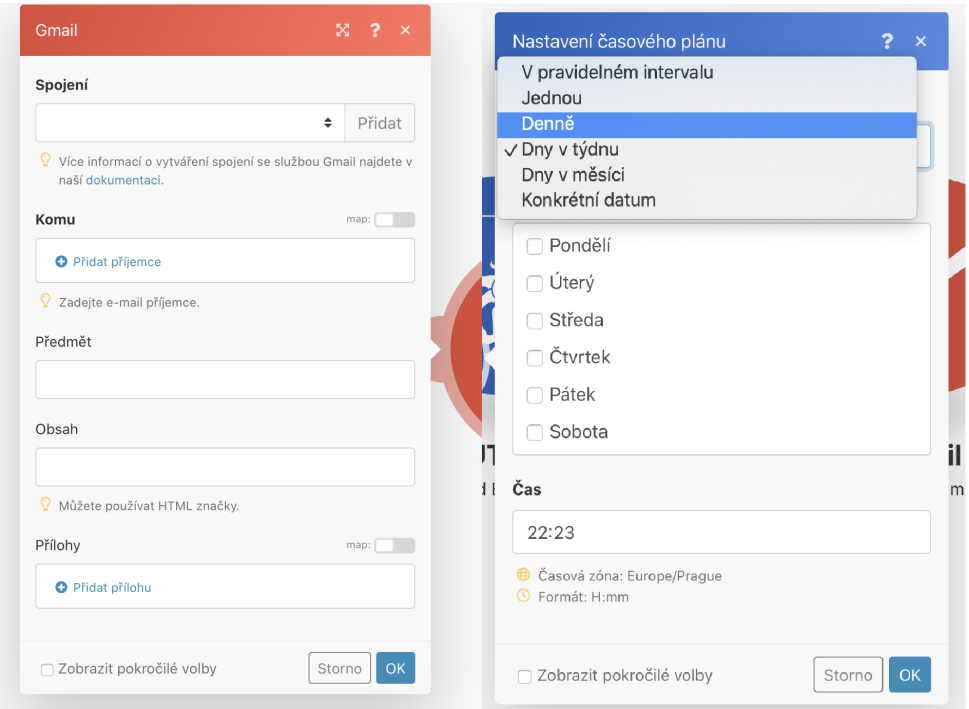
[1] https://integromat.com/
[2] https://automate.io/
[3] https://zapier.com/
[4] https://emea.flow.microsoft.com/
[5] What You See Is What You Get



Obrázek 3.1: Operace pro KOSAPi implementované na Integromatu



Obrázek 3.2: Konfigurace operace stažení seznamu zkoušek předmětu



Obrázek 3.3: Konfigurace notifikací a odběru

Kapitola 4
Návrh

Hlavní myšlenkou při návrhu systému byla flexibilita a rozšiřitelnost. Z těchto důvodů se model nesnaží napodobovat entity KOSApi, ale snaží se být obecnějším úložištěm dat. Zároveň zde byla snaha navrhnout systém tak, aby uživatel měl možnost si nastavit jemu vyhovující notifikace a administrátor mohl za běhu systému přidávat, upravovat a odstraňovat datové zdroje a struktury pro ukládání dat (včetně jejich textové reprezentace pro uživatele) bez potřeby restartu systému a snížení dostupnosti.

Aby uživatelům nebyly zasílány duplicitní informace a oni nebyli notifikováni i když se data nezměnila, je systém navržený tak, aby porovnával nová data s aktuálními a nová uložil (a následně notifikoval) jen pokud se liší.

4.1 Způsob detekce změny dat

Jedno z rozhodnutí, které bylo při návrhu potřeba udělat je způsob získání informace o tom, že jsou nějaká data aktualizována a je potřeba o nich notifikovat. Jednou z možností bylo použití událostí.

Po aktualizaci dat z datového zdroje vznikne v systému událost, která vyvolá notifikaci. Tento přístup, ač by přidal flexibilitě (na událost by mohly být i jiné reakce), nebyl úplně ideální. Důvodem je možnost uživatele nastavit si vlastní periodu pro notifikování. Mohla by nastat situace, že dostaneme notifikaci o aktualizaci dat, ale uživatel o ní notifikaci ještě dostat nechce (nebo ještě hůře, někteří uživatelé ji chtějí a jiní ještě ne). V takovém případě by bylo možné reagovat dvěma způsoby:

1. událost ignorovat, nebo
2. uložit si událost k pozdějšímu zpracování.

Ani jeden ze způsobů není úplně správný. Při prvním ztratíme informaci o tom, že jsme měli někoho notifikovat. Druhý způsob tímto problémem sice netrpí, ale také není ideální. Bylo by nutné mít události uložené persisteně tak, aby přežily restart aplikace. Dále v případě, že by ve frontě bylo více událostí odkazující na stejná data, bylo by potřeba zajistit použití té nejnovější a ostatní ignorovat. A posledním problémem je, že pokud bychom notifikovali jen při události a dlouho by se žádná data neaktualizovala, mohl by uživatel

dostat notifikaci pozdě. To by bylo možné vyřešit periodickým probouzením a kontrolou fronty, ale tím se v podstatě dostáváme k druhému řešení.

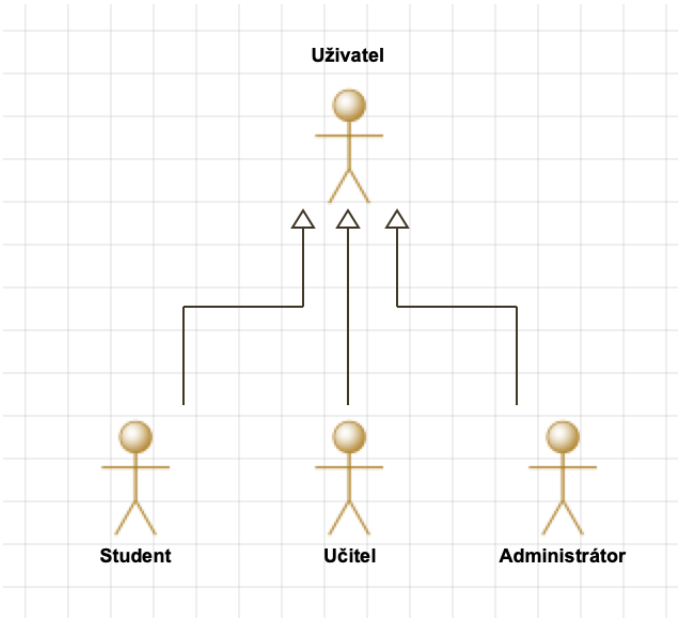
Druhým řešením, které bylo nakonec zvolené, je periodická kontrola aktualizací dat. I když se může stát, že občas zjistíme, že se nic nezměnilo, tak je řešení jednodušší a nemá žádný z předchozích problémů.

4.2 Případy užití

V následující kapitole jsou diagramy a popisy identifikovaných případů užití.

V diagramech s případy užití jsou často použity výrazy jako "Počet jednorázových akcí"místo "Odběr změn počtu jednorázových akcí". Důvodem je omezený prostor pro diagramy, celý popis je pak v seznamu pod diagramy.

4.2.1 Aktéři



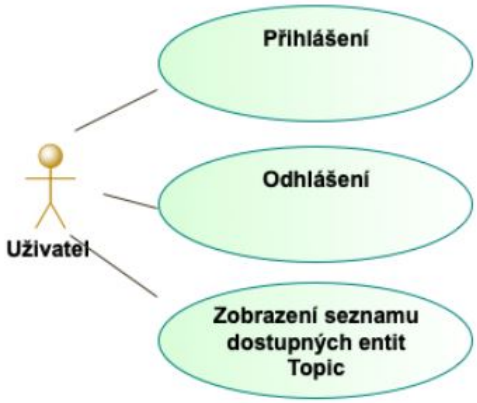
Obrázek 4.1: Aktéři případů užití

V systému existují následující aktéři:

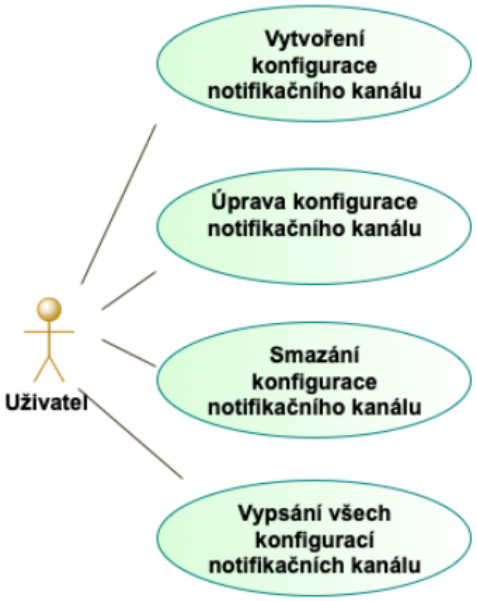
- *Uživatel*: Obecný uživatel systému. Jedná se o předka všech ostatních aktérů.
- *Student*: Student používající systém.
- *Učitel*: Učitel používající systém.
- *Administrátor*: Administrátor, který spravuje systém.

4.2.2 Případy užití uživatele

Případy užití uživatele, které jsem identifikoval jsou zachyceny na diagramech 4.2 a 4.3.



Obrázek 4.2: Případy užití uživatele



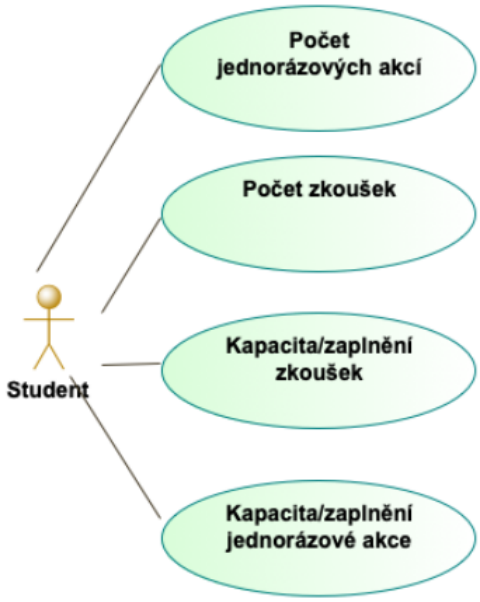
Obrázek 4.3: Případy užití uživatele

- *Přihlášení.* Každý uživatel musí být pro práci v systému přihlášený. Uživatelé se mohou přihlásit buďto jménem a heslem (např. administrátor) nebo pomocí školního SSO (studenté a učitelé).
- *Odhlášení.* Uživatel má možnost se ze systému odhlásit.

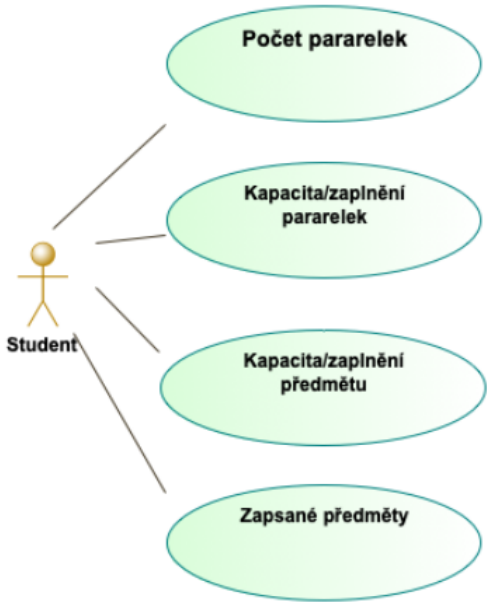
- *Vytvoření konfigurace notifikačního kanálu.* Uživatel má možnost vytvořit si novou konfiguraci notifikačního kanálu, který může použít při odběru jako notifikační kanál.
- *Vypsání všech konfigurací notifikačních kanálů.* Uživatel má možnost zobrazit si seznam všech svých vytvořených konfigurací notifikačních kanálů.
- *Úprava konfigurace notifikačního kanálu.* Uživatel má možnost upravit existující konfiguraci notifikačního kanálu.
- *Smazání konfigurace notifikačního kanálu.* Uživatel má možnost smazat existující konfiguraci notifikačního kanálu.
- *Zobrazení seznamu dostupných entit Topic.* Uživatel má možnost zobrazit si seznam všech jemu dostupných entit.

4.2.3 Případy užití studenta

Případy užití pro studenta jsou zachyceny na diagramu 4.4 a 4.5. V rámci každého případu užití je možné odběr vytvořit, upravit či zrušit a vybrat notifikační kanál použitý pro upozornění.



Obrázek 4.4: Případy užití studenta

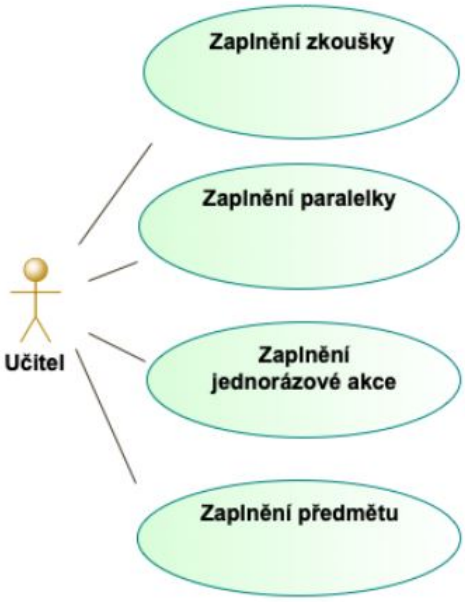


Obrázek 4.5: Případy užití studenta

- *Počet jednorázových akcí*. Student má možnost odebírat změny počtu jednorázových akcí předmětu.
- *Počet zkoušek*. Student má možnost odebírat změny počtu zkoušek předmětu.
- *Kapacita/zaplnění zkoušek*. Student má možnost odebírat změny kapacity a zaplnění zkoušek.
- *Kapacita/zaplnění jednorázové akce*. Student má možnost odebírat změny kapacity a zaplnění jednorázových akcí.
- *Počet paralelek*. Student má možnost odebírat změny počtu paralelek předmětu.
- *Kapacita/zaplnění paralelek*. Student má možnost odebírat změny kapacity a zaplnění paralelek.
- *Kapacita/zaplnění předmětu*. Student má možnost odebírat změny kapacity a zaplnění předmětu.
- *Zapsané předěmty*. Student má možnost odebírat změny v zapsaných předmětech (např. zápočty a známky).

4.2.4 Případy užití učitele

Případy užití pro učitele jsou zachyceny na diagramu 4.6. V rámci každého případu užití je možné odběr vytvořit, upravit či zrušit a vybrat notifikační kanál použitý pro upozornění.

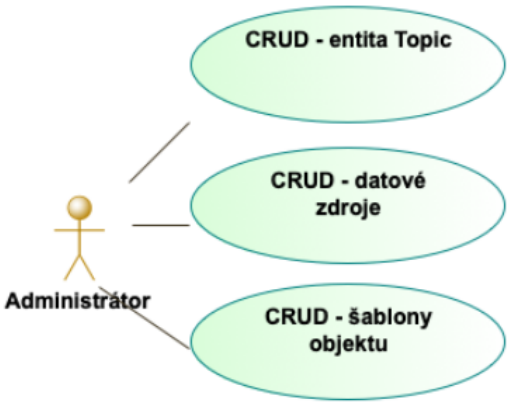


Obrázek 4.6: Případy užití učitele

- *Zaplnění zkoušky.* Učitel má možnost odebírat změny v zaplnění zkoušek.
- *Zaplnění paralelky.* Učitel má možnost odebírat změny v zaplnění paralelek.
- *Zaplnění jednorázové akce.* Učitel má možnost odebírat změny v zaplnění jednorázových akcí.
- *Zaplnění předmětu.* Učitel má možnost odebírat změny v zaplnění předmětů.

4.2.5 Případy užití administrátora

Případy užití administrátora, které jsem identifikoval jsou zachyceny na diagramu 4.7.



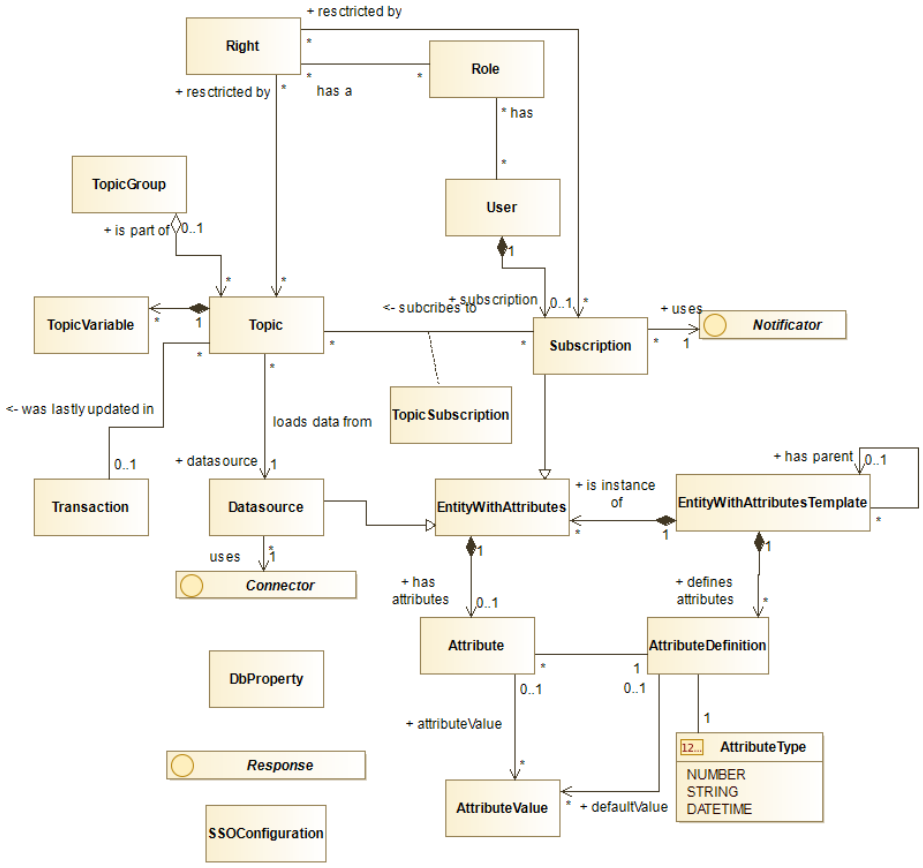
Obrázek 4.7: Případy užití administrátora

- *CRUD - entita Topic*. Administrátor má možnost provádět CRUD operace nad entitami Topic.
- *CRUD - datové zdroje*. Administrátor má možnost provádět CRUD operace nad datovými zdroji.
- *CRUD - šablony objektu*. Administrátor má možnost provádět CRUD operace nad šablonami objektů.

4.3 Diagram tříd

4.3.1 Datový model

Následující sekce popisuje třídy sloužící pro ukládání dat a konfigurace systému. Model je zachycený na diagramu 4.8.



Obrázek 4.8: Diagram tříd datového modelu

Role

Reprezentuje roli uživatele. Role určuje, jaká práva uživatel má.

Right

Reprezentuje oprávnění. Oprávněním se dají omezit operace v systému. Omezuje viditelnost entit Topic a jejich dat tak, aby byla viditelná jen oprávněným uživatelům. Dále jsou použity pro omezení práv na viditelnost a manipulaci s konfiguračními kanály. Zde spolupracuje s konceptem vlastníctví notifikačního kanálu, kde práva umožňují sdílení notifikačního kanálu s jiným uživatelem.

Tuto možnost není v rámci práce možné využít z klienské aplikace, ale může se jednat o další rozvoj.

User

Reprezentuje uživatele. Entita reprezentuje jak uživatele přihlašující se přes SSO, tak systémové uživatele.

AttributeDefinition

Reprezentuje definici typu. Obsahuje název atributu a příznak, zda je atribut povinný.

AttributeType

Reprezentuje typ atributu. Může být použito pokud je potřeba:

- zkontrolovat, že hodnota odpovídá typu
- provést převedění na odpovídající typ v kódu.

Attribute

Reprezentuje instanci definovaného atributu. Může mít jednu či více hodnot. Hodnoty jsou aktuálně reprezentovány jako text (zapouzdřené v entitě AttributeValue) pro jednoduché přidávání nových typů. Tyto hodnoty mohou být následně převedeny na odpovídající typy v kódu za pomocí AttributeType. Druhou možností bylo mít ve třídě více atributů, kde by byl jeden za každý typ a vždy by byl nejvýše jeden přítomný. Tato implementace mi přišla méně flexibilní a hůře udržovatelná pro větší množství typů.

AttributeValue

Reprezentuje hodnotu atributu. Může být použita buď pro hodnotu instance atributu nebo jak výchozí hodnota pro jeho definici.

EntityWithAttributesTemplate

Reprezentuje šablonu pro entity s atributy. Specifikuje, jaké atributy by daná entita měla mít a umožňuje tak kontrolu konfigurace. Hlavní myšlenkou třídy je možnost vytvářet a konfigurvat entity bez nutnosti úpravy kódu. Šablona může mít nadřazenou entitu. V rámci bakalářské práce se jedná spíše jen o značkování spíše než dědičnost. V rámci dalšího rozvoje by mohlo být určitě upraveno na dědičnost.

EntityWithAttribute

Reprezentuje instanci entity s atributy. Jedná se o abstraktní třidu.

Transaction

Třída reprezentující logickou transakci, která má dva hlavní významy:

1. Logický čas použitý jako časová značka. Takto se transakce používá pro uchování informace o tom, kdy byla data aktualizována a jaká poslední data byla přečtena odběrem.

2. Reprezentaci jedné operace. Aktualizace dat probíhá celá v jedné logické transakci a dá se na ní hledět jako na jeden celek (entitu), na kterou je možné navázav například logy nebo události (není implementováno v rámci bakalářské práce, může být implementováno v rámci dalšího rozvoje).

Topic

Reprezentuje skupinu souvisejících dat. Můžeme se na něj dívat jako na strukturu a na TopicVariable jako na její data. Zároveň je možné definovat, jak má vypadat její textová reprezentace (použitá pro notifikaci). Šablona pro textovou reprezentaci může využívat proměnné topicu. Pro rychlé vyhledávání a jednoznačnou nečíselnou identifikaci je Topic opatřen nepovinným atributem key. Atribut isTemplate určuje, zda se jedná jen o šablonu pro další instance či o funkční instanci. Instance entity Topic, které jsou šablonami nejsou brané v potaz v rámci aktualizace dat.

TopicVariable

Reprezentuje jednu proměnou topicu a slouží pro uchování dat. Zároveň specifikuje dotaz, který je použit pro získání konkrétních dat z odpovědi konektoru.

Datasource

Reprezentuje datový zdroj. Třída obsahuje konfiguraci pro periodu stahování dat, a také atributy, které slouží jako konfigurace použitého konektoru.

Subscription

Reprezentuje odběr. Jedná se o konfiguraci, která umožňuje nastavit si odběr změn v datech entit Topic a upozornění za použití nakonfigurovaného notifikačního kanálu. Její atributy jsou použity jako konfigurace pro notifikátor.

TopicSubscription

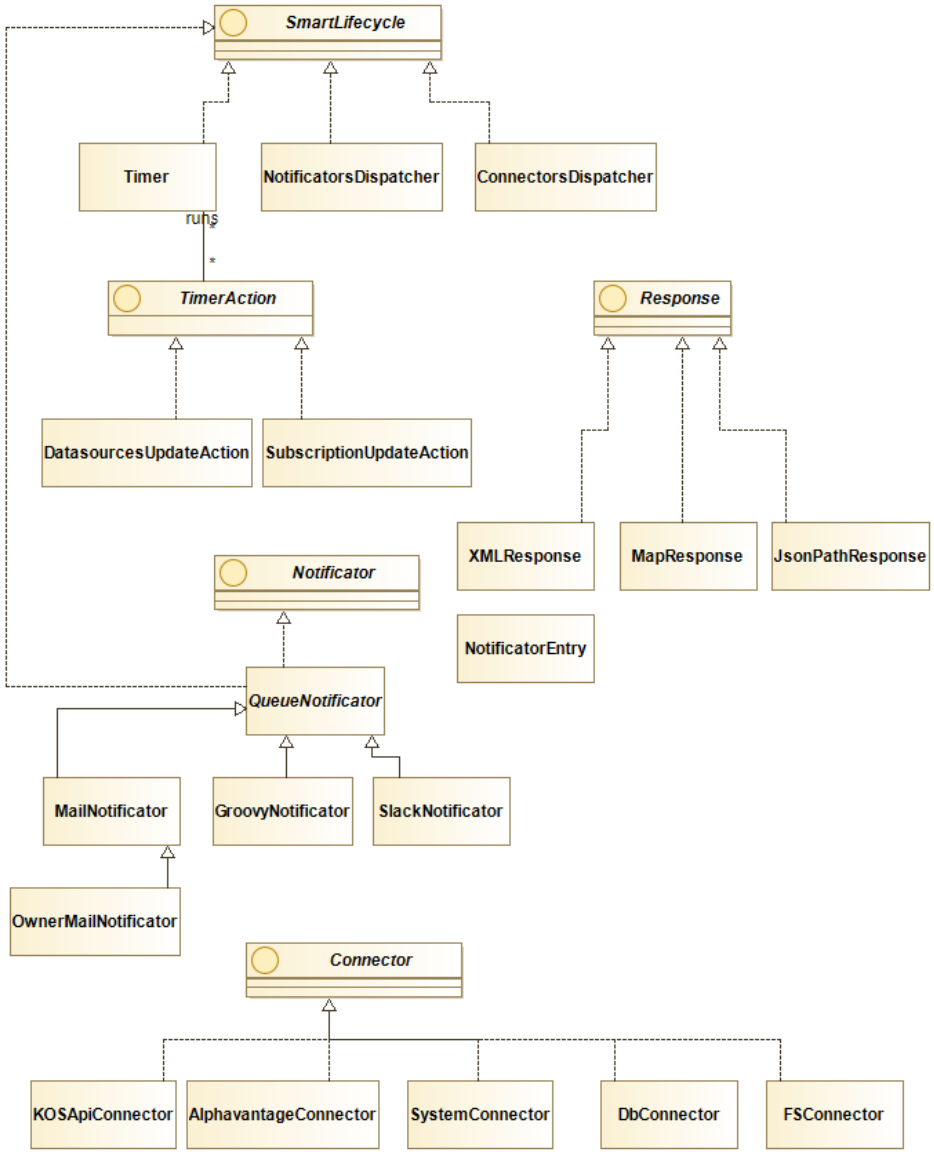
Dodatečné info o odběru konkrétního topicu. Umožňuje uživateli zvolit si, zda chce být notifikován ihned po změně dat nebo v jim nastaveném periodě (např. jednou za hodinu).

SSOConfiguration

Reprezentuje konfiguraci pro jednotlivá SSO. Odpovídá struktuře třídy ClientRegistration[2] obohacené o různé skripty ovlivňující proces přihlášní přes SSO a skript, který se má spustit po prvním přihlášení uživatele.

4.3.2 Model výkonných tříd aplikace

Následující sekce popisuje třídy, které vykonávají funkce systému za použití dat z datového modelu. Model je zachycený na diagramu 4.9



Obrázek 4.9: Diagram tříd funkčního modelu

TimerAction

Rozhraní pro periodicky spouštěné akce. Akce sama si může nastavit, jak často chce být spouštěna. Tyto akce jsou spouštěny komponentou Timer.

Timer

Komponenta sloužící ke spouštění periodických akcí. Při svém startu (tj. start aplikace) si načte z aplikačního kontextu všechny komponenty implementující rozhraní TimerAction. Ty poté spouští v jimi specifikovaných intervalech.

DatasourcesUpdateAction

Periodická akce sloužící ke stažení nových dat z datových zdrojů. Při svém spuštění zkontroluje, zda existují datové zdroje, které je potřeba aktualizovat a pokud ano, tak je pomocí konektoru aktualizuje. Odpověď vrácenou z datového zdroje použije k aktualizaci dat v entitách Topic, které daný zdroj používají.

SubscriptionUpdateAction

Periodická akce sloužící k notifikaci uživatelů systému o změnách v datech. Při svém spuštění zkontroluje, zda se některá data od poslední notifikace změnila a zda je možné o změně notifikovat. Pokud to tak je, použije notifikátor k odeslání notifikace.

Notificator

Rozhraní pro notifikátory. Předepisuje metodu pro odesílání notifikací.

NotificatorEntry

Notifikační data. Obsahují id konfigurace notifikačního kanálu (pro možné dodatečné čtení jejich atributů) a textovou reprezentaci aktualizovaných dat.

QueueNotificator

Asynchronní implementace notifikátoru. Veškerá notifikační data se vkládají do fronty, která je v jiném vlákně zpracovávána. Výhodou této implementace je, že vlákno, které chce notifikovat nemusí čekat na dokončení notifikace v případě, že by se jednalo o dlouhotrvající operaci.

MailNotificator

Implementace asynchorního notifikátoru pro odesílání e-mailů. Načítá svou konfiguraci z databáze a z odběru si čte atribut "receiver".

OwnerMailNotificator

Implementace asynchorního notifikátoru pro odesílání e-mailů uživateli, kterému patří konfigurace notifikačního kanálu. Nemá žádné další atributy a je jednoduchým řešením pro potřebu notifikace uživatele, aniž by mohl zadat libovolný email.

SlackNotificator

Implementace asynchorního notifikátoru pro odesílání notifikací na Slack. Implementace zatím podporuje jen jeden pracovní prostor. Jako jediný atribut má token, který uživatel získá použitím příkazu "/token"v daném pracovním prostoru. Tokeny jsou vázané na pracovní prostor a kanál. Zamezí se tak zbytečnému generování tokenů pro stejný kanál (pokud již token pro daný kanál existuje, tak je vrácen).

GroovyNotificator

Implementace asynchroního notifikátoru, který spustí skript uložený v konfiguraci notifikačního kanálu. Je to interní mechanismus, který využívá notifikací pro interní úkony. Příkladem použití je kontrola a úprava (vytvoření či smazání) entit Topic pro jednotlivé zkoušky předmětu, pokud se v entitě Topic obsahující data o počtu zkoušek změní hodnota.

Response

Rozhraní pro odpovědi konektorů. Jedná se o abstrakci, která umožňuje použít dotaz pro získání dat z odpovědi. Syntaxe a zpracování dotazu je na implementaci rozhraní. Administrátor, který aplikaci konfiguruje by měl vědět, jakou syntaxi použít.

XMLResponse

Implementace odpovědi ve formátu XML (použita pro KOSApi). Syntaxe pro získání dat je XPath.

MapResponse

Implementace odpovědi ve formě mapy. Dotaz pro získání hodnoty je název klíče, pod kterým se hodnota v mapě nachází.

JsonPathResponse

Implementace odpovědi ve formátu JSON. Syntaxí pro získání dat je JSONPath[3]. Tato implementace umožní snazší integeraci na moderní API používající formát JSON, ale zachovává možnost jednoduchého dotazování nad daty bez potřeby jejich zpracování.

Connector

Rozhraní pro konektor. Jeho implementacemi jsou komponenty, které můžeme použít pro připojení k datovým zdrojům.

KOSApiConnector

Implementace konektoru pro KOSApi. Z datového zdroje si čte atribut "path", který určuje url adresu ze které má stáhnout data. Konektor se stará o stažení dat a o získání a obnovu autentizačního tokenu.

SystemConnetor

Komponenta v aplikaci, která se chová jako konektor. Při dotazu na data vytvoří MapResponse která obsahuje aktuální údaje o systému, na kterém aplikace běží (např. celková / volná RAM, celková / volná Java Heap a zatížení procesoru). Tyto data jsou pak ukládána do entity Topic a jsou k dispozici v aplikaci. Ve spojitosti s GroovyNotificatorem by bylo například možné při změně pomocí skriptu zkontrolovat hodnoty a při docházející paměti udělat např. heap dump.

DbConnector

Implementace konektoru, která se připojí k databázi a spustí dotaz specifikovaný v atributu "query". Aktuální implementace je omezená čtením jen prvního řádku z výsledku dotazu, který poté uloží do MapResponse (kde klíčem je název sloupce a hodnotou hodnota). Tato implementace dovoluje vytváření různých analytických dotazů a jejich ukládání do entit Topic a případnou notifikaci. Tímto způsobem se dají jednoduše vytvářet různé indikátory dat (např. můžeme mít dotaz, který hlídá datové zdroje, u kterých se nepovedla aktualizace déle než 24 hodin) za běhu systému bez potřeby psaní kódu a snížení dostupnosti kvůli nasazení nové verze.

AplhavantageConnector

Implementace konektoru, která umožňuje stažení dat z Aplha Vantage[1]. Konektor z datového zdroje čte atributy *endpoint*, *apikey*, *function* a *params*, které používá ke složení URL adresy pro požadavek na data. Vrácená data jsou ve formátu JSON a proto vrací JsonPathResponse.

FSConnector

Implementace konektoru, která umožňuje číst data z file systému. Aktuální implementace podporuje jen XML soubory a vrací *XMLResponse*. Z datového zdroje čte atribut *path*.

NotificatorDispatcher

Komponenta která umožňuje předání notifikačních dat notifikátoru. Komponenta při startu nalezne všechny implementace rozhraní Notificator a předává jim data na základě jejich jména.

[1] https://www.alphavantage.co/

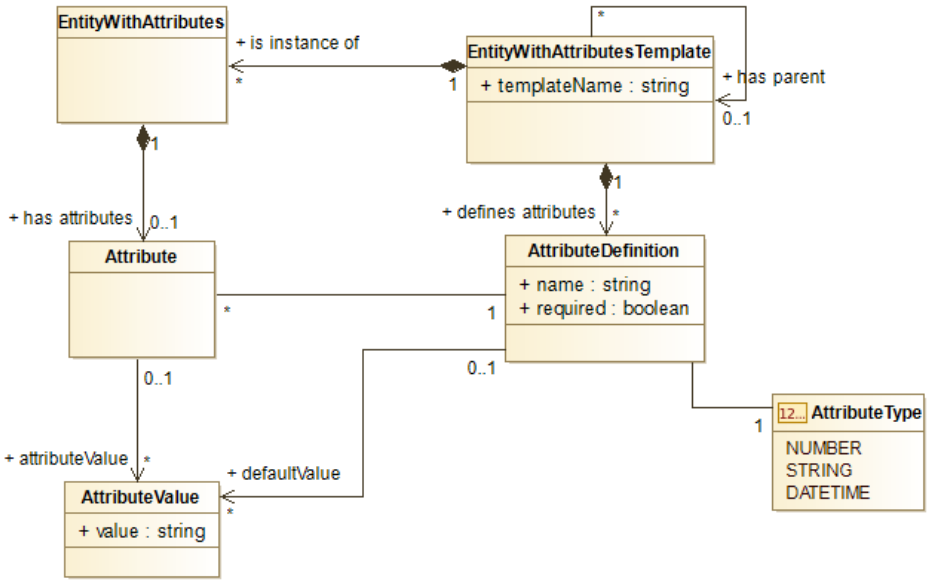
ConnectorsDispatcher

Komponenta, která umožňuje používat konektory na základě jejich jména. Při startu nalezne všechny implementace rozhraní Connector a odesílá přes ně požadavky na data.

4.4 Použité návrhové vzory

V této kapitole popíši některé návrhové vzory použité v datovém a funčkním modelu a jejich využití.

4.4.1 Dynamic Object Model



Obrázek 4.10: Použití návrhového vzoru DOM

Na diagramu 4.10 je realizace návrhového vzoru Dynamic Object Model[8]. Jedná se o návrhový vzor složený z dalších návrhových vzorů, kterými jsou:

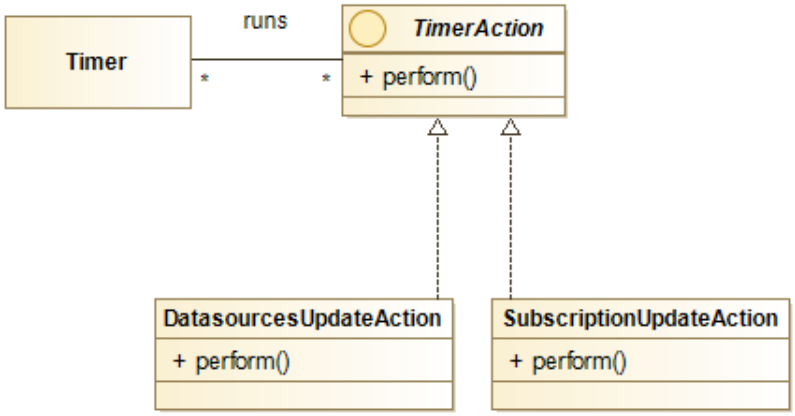
1. Type Object[7] - objekt reprezentující typ objektu. Reprezentuje jeho strukturu a hodnoty, které mohou být použity. V našem případě se jedná o *AttributeDefinition* a *EntityWithAttributeTemplate*. První z nich slouží jako definice atributů entit a druhý jako definice instancí entit.

2. Property List[6] - tento vzor nám umožňuje dynamicky přidávat a odebírat atributy objektů a dotazovat se na ně pomocí jejich jména. V našem případě se jedná o vazbu mezi *EntityWithAttributes* a *Attribute*.

3. Value Holder[1] - Slouží pro zapouzdření hodnoty atributu. V našem případě se jedná o *AttributeValue*

Návrhový vzor zvyšuje flexibilitu a dostupnost systému a zároveň snižuje realný počet tříd v kódu, protože je možné je definovat v konfiguraci. Cenou za to je zvýšená komplexita modelu (z pohledu pochopení) a práce s ním.

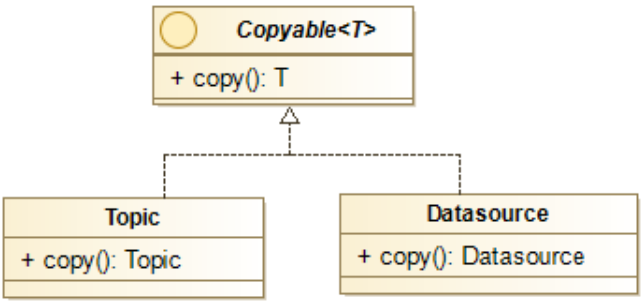
4.4.2 Command



Obrázek 4.11: Použití návrhového vzoru Command

Na diagramu 4.11 je realizace návrhového vzoru Command[5]. Realizacemi příkazů jsou zde *DatasourcesUpdateAction* a *SubscriptionUpdateAction*. Díky využití návrhového vzoru spolu s využitím možností kontejneru frameworku Spring je velice jednoduché přidávat nové akce. Díky zapozdření akcí je pak také velice jednoduché realizovat i složitější akce, které by například zahrnovaly komunikaci s jinými systémy.

4.4.3 Prototype



Obrázek 4.12: Použití návrhového vzoru Prototype

Na diagramu 4.12 je realizace návrhového vzoru Prototype[5]. Konkrétními prototypy jsou zde *Topic* a *Datasource*. Tento návrhový vzor umožňuje jednoduše vytvářet nové entity Topic a Datasource například pro obsazenost zkoušek nebo studijní výsledky studenta. Využití tohoto vzoru ušetřilo spoustu práce, kde by se jinak musely entity vytvářet ručně ve skriptech.

4.4.4 DTO

Tyto třídy nejsou na žádném z diagramů. Z pohledu softwarového návrhu nejsou zajímavé a slouží pro technické zajištění přenosu struktur modelu mezi serverem a klientem. Použití návrhového vzoru DTO[4] umožňuje například odesílat na serverovou stranu (i opačným směrem - při čtení) entity Topic i s jejich proměnnými (TopicVariable) nebo datové zdroje či konfigurace notifikačních kanálů přímo i s hodnoty atributů. Není tak potřeba dělat více volání a komunikace mezi serverovou a klientskou stranou. Nevýhodou je ovšem potřeba vytvářet více tříd a transformací z entit na DTO a opačným směrem pro každou třídu, která je takto posílána.

Velkou výhodou tohoto návrhového vzoru je zapouzdření. Uživatel API je odstíněn od vnitřní struktury tříd a implementace. Pokud by se implementace změnila, API pro uživatele zůstane stejná. Stejně tak je poté jednodušší API verzovat. Pokud se změní verze, mohou se naopak změnit zase jen DTO, ale implementace za tím může zůstat beze změny.

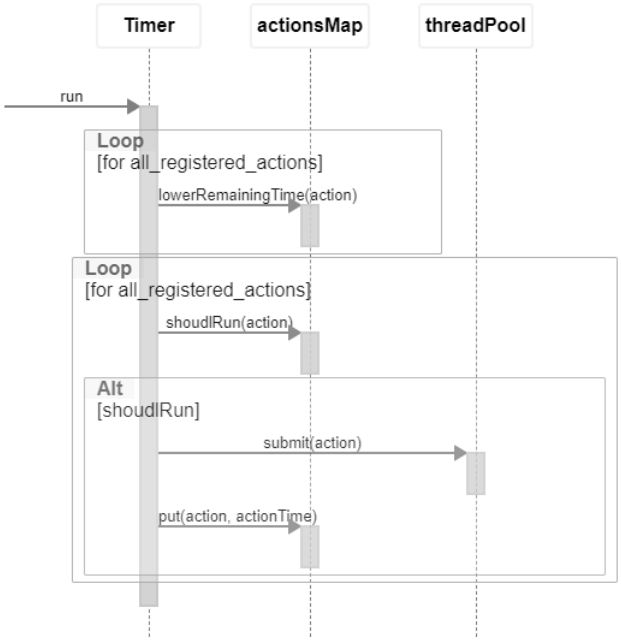
4.5 Sekvenční diagramy

4.5.1 Spouštění periodických akcí

Spouštění periodických akcí je popsáno sekvenčním diagramem 4.13. Komponenta nejdříve projde seznam všech zaregistrovaných periodických akcí a sníží čas zbývající do spuštění o jednu vteřinu. Poté projde seznam znovu, spustí asynchronně všechny akce, pro které je zbývající čas 0 a pak je znovu zaregistruje s jejich periodou.

Prostor pro potencionální rozšíření a úpravy

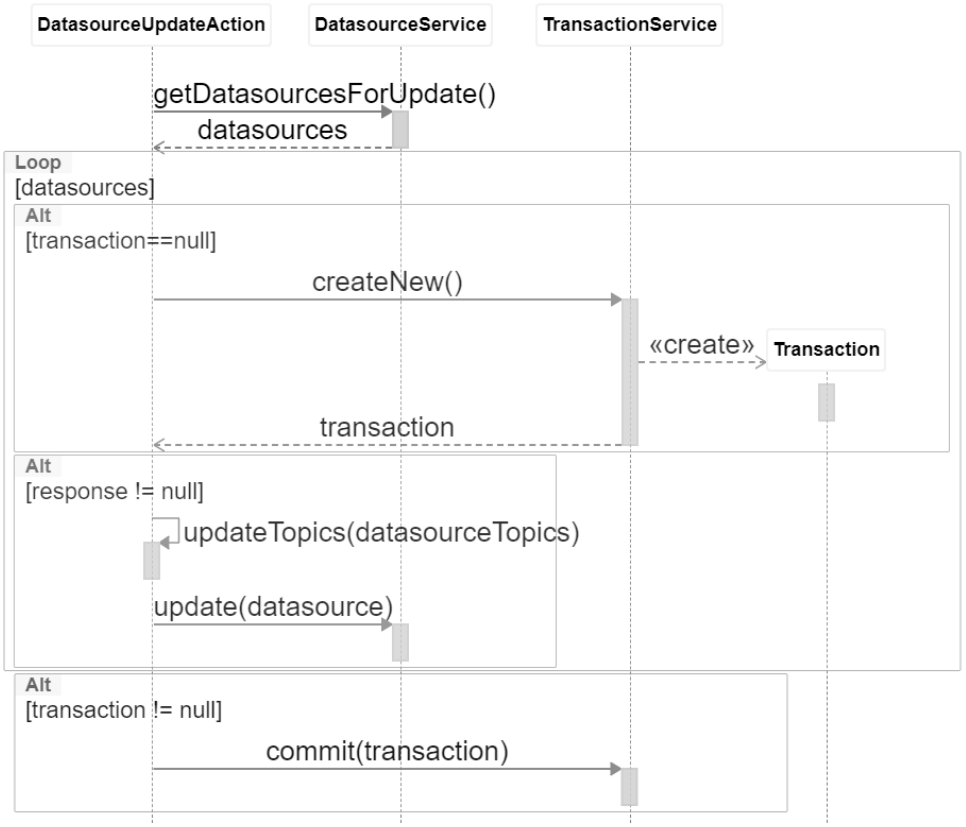
Aktuální řešení, ač funkční, není kompletní. V současné implementaci se akce spustí a hned zafrontuje znovu, nečeká se na její dokončení. Za zvážení by stála implementace módu, kdy se akce znovu zafrontuje až po jejím dokončení. Druhá úprava by byla použití *ScheduledExecutorService*, kde by se spuštění akcí plánovalo přímo místo odpočítávání.



Obrázek 4.13: Sekvenční diagram periodických akcí

4.5.2 Aktualizace datového zdroje

Aktualizace datového zdroje je popsána sekvenčním diagramem 4.14. Akce je spouštěna komponentou Timer. Nejdříve si z *DatasourceService* načte seznam datových zdrojů, které je potřeba aktualizovat. V případě, že neexistuje transakce pro aktuální proces aktualizace, vytvoří novou. Poté pro každý takový datový zdroj použije komponentu *ConnectorDispatcher* a pomocí ní stáhne data. Dalším krokem je Aktualizace dat v entitě Topic, pro všechny topicy, které používají daný datový zdroj. Konečnými kroky je aktualizace času posledního stažení dat, posledního úspěšného stažení dat (jen v případě úspěšného stažení) a commit transakce (myšleno logické, ne databázové).



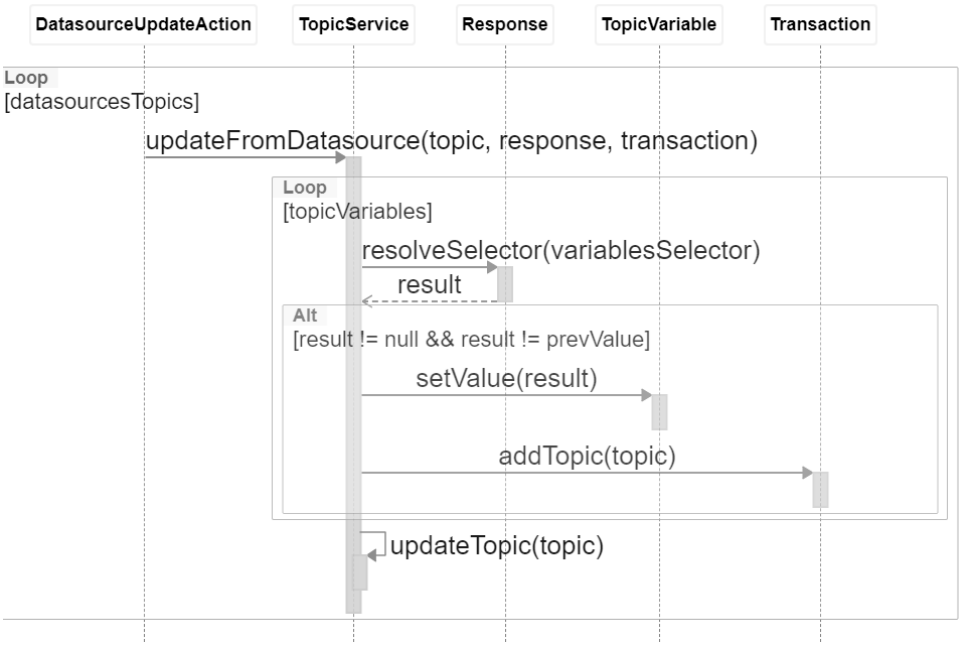
Obrázek 4.14: Sekvenční diagram aktualizace datového zdroje

Aktualizace dat v entitě Topic

Aktualizace dat v entitě Topic je popsána sekvenčním diagramem 4.15. Akce pro každou proměnnou v entitě Topic udělá následující kroky:

1. použije její dotaz pro získání dat z datového zdroje;
2. zkontroluje, že jsou požadovaná data přítomna a zda se liší od aktuálních dat;
3. pokud se liší, nastaví nová data jako aktuální hodnotu proměnné a přidá; entitu [2] do množiny entit změněných v rámci aktuální transakce.

Nakonec provede aktualizaci dat entity v úložišti.

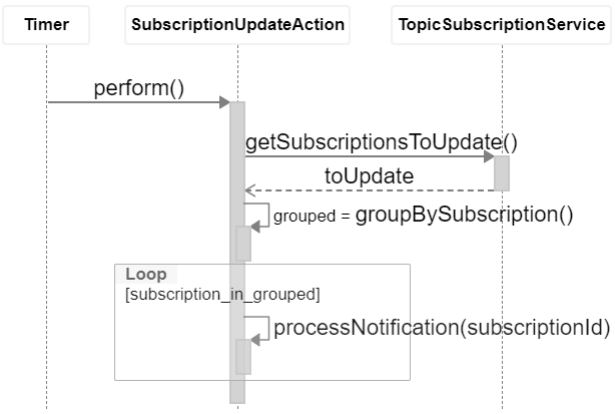


Obrázek 4.15: Sekvenční diagram aktualizace dat v topicu

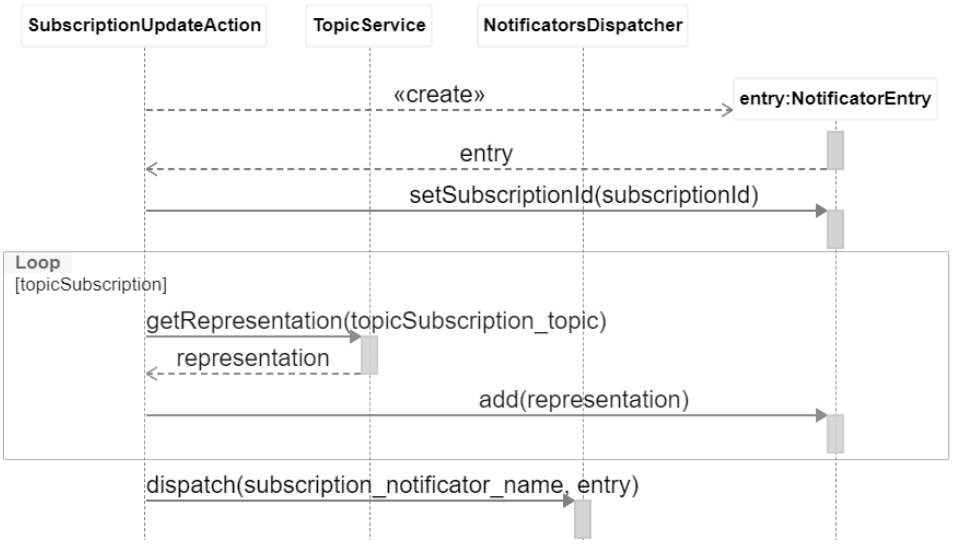
[2] Stačí změna jedné proměnné v entitě Topic, aby byl stav entity aktualizovaný

4.5.3 Notifikace uživatelů

Aktualizace dat topicu je popsána sekvenčním diagramem 4.16. Akce je spouštěna komponentou Timer. Nejdříve si z *TopicSubscriptionService* načte seznam odběrů entit Topic, u kterých je potřeba uživatele notifikovat. Tyto potom seskupí dle uživatelů. Poté tyto skupiny postupně projde a pro všechny entity Topic získá jejich textovou reprezentaci. Nakonec textové reprezantace pro celou skupinu předá komponentě *NotificatorsDispatcher* pro odeslání uživateli. Zpracování jednotlivých odběrů je popsáno sekvenčním diagramem 4.17.



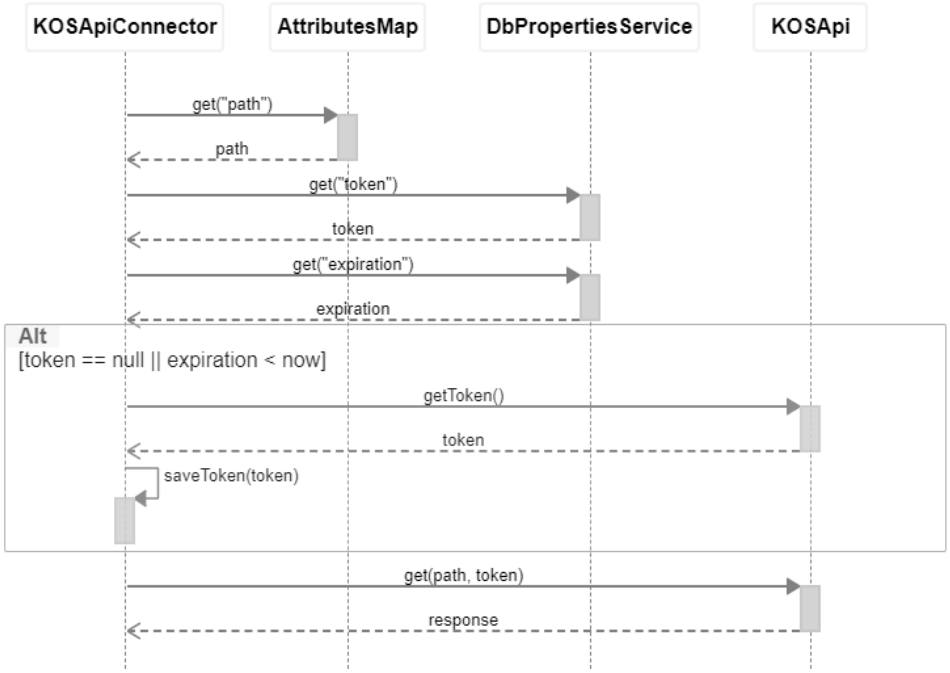
Obrázek 4.16: Sekvenční diagram notifikace uživatelů



Obrázek 4.17: Sekvenční diagram zpracování odběru

4.5.4 Stažení dat z KOSApi

Stažení dat z KOSApi je popsané sekvenčním diagramem 4.18. Konektor si načte URL adresu, ze které má data stáhnout. Poté z úložiště načte uložený token a čas jeho expirace. Zkontroluje přítomnost a platnost tokenu a v případě, že je token neplatný zajistí si z KOSApi nový a ten uloží. Následně stáhne data z požadované URL.

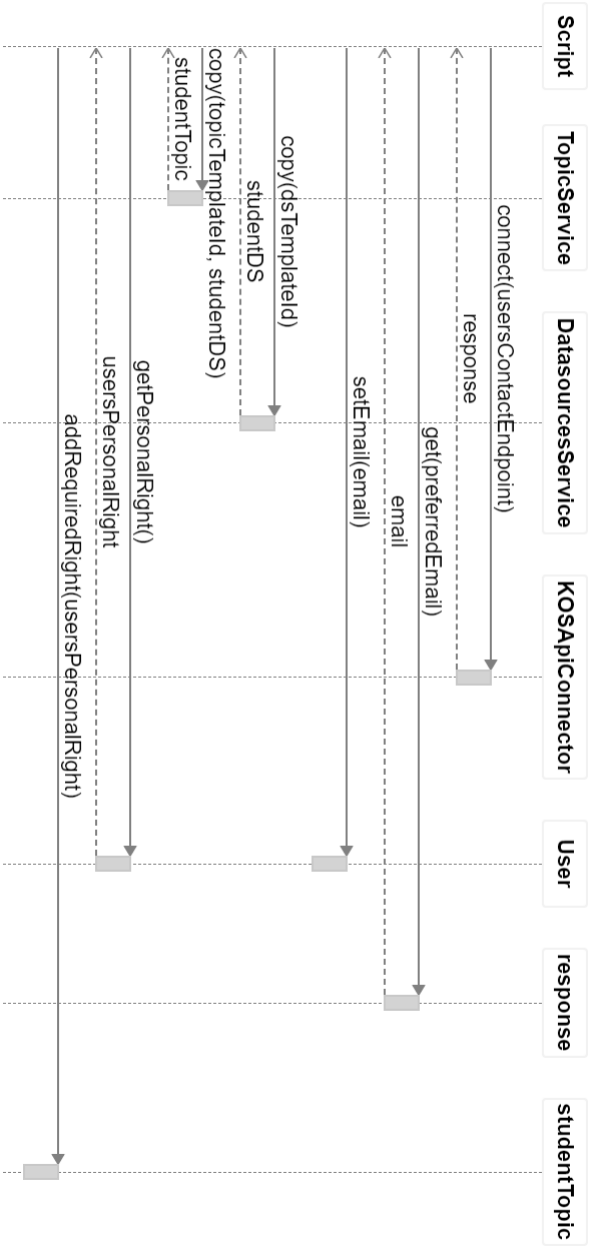


Obrázek 4.18: Sekvenční diagram připojení na KOSApi

4.5.5 První přihlášení uživatele přes ČVUT SSO

Kroky prováděné po prvním přihlášení uživatele přes ČVUT SSO jsou popsány na sekvenčním diagramu 4.19. Tento skript je spuštěný poté, co zjistíme, že přihlášeného uživatele ještě nemáme v databázi. Nejdříve je potřeba získat kontaktní informace uživatele (email) z KOSApi. Jakmile máme kontaktní údaje uložené, vytváříme datový zdroj pro studijní výsledky studenta a entitu Topic, do které budou informace uloženy. Nakonec omezíme viditelnost entity osobním právem uživatele[3] tak, aby pro něj bylo viditelné a mohl začít odebírat jeho změny.

[3]Koncept, kdy pro každého uživatele existuje jedno právo, které je bráno jako jeho "osobní".



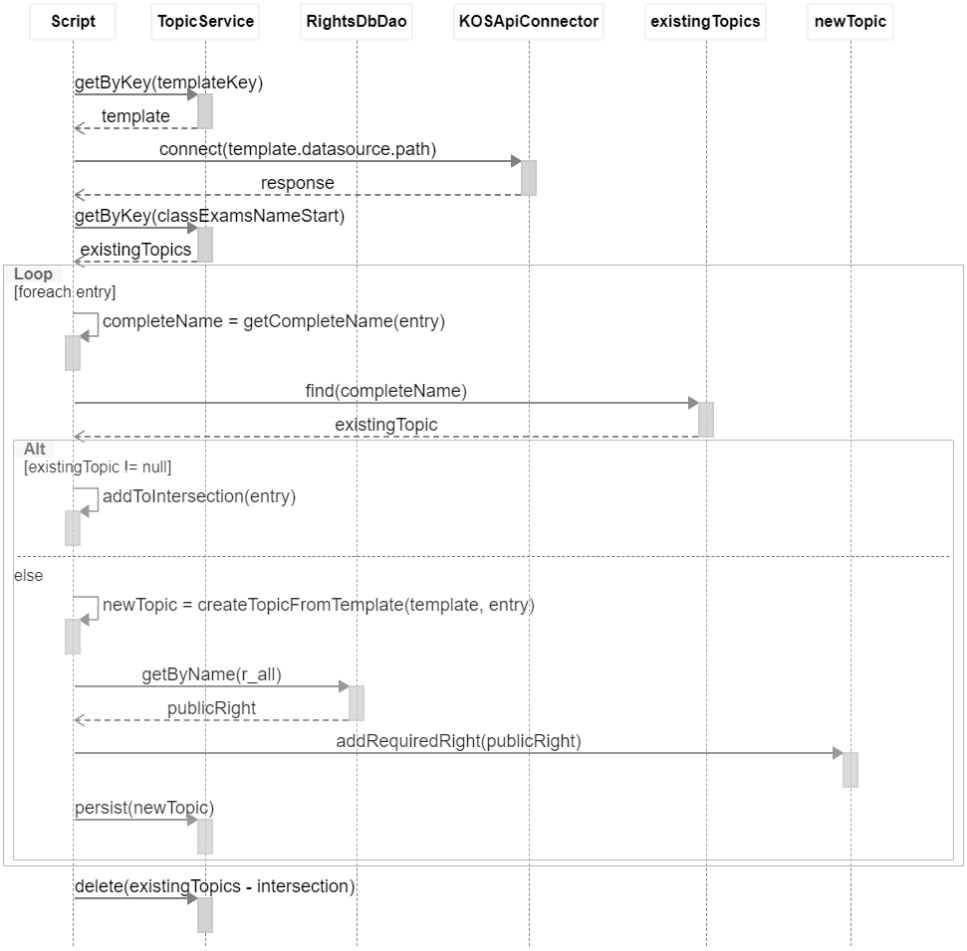
Obrázek 4.19: Sekvenční diagram prvního přihlášení uživatel přes ČVUT SSO

Možná vylepšení

Aktuálně je stažení kontaktních informací i vytváření entit Topic a datového zdroje synchronní a prováděn v rámci procesu přihlášení. Řešení je dostačující a z časových důvodů jsem se rozdl pro tento postup spíše než pro složitější mechanismy. V budoucnu by bylo lepší nechat jako synchronní akci jen stažení kontaktních informací a další akce mít jako asynchronní. Možnou realizací je využití konektoru *DBConnector* s dotazem na ID záznamů uživatelů, pro které je tyto akce potřeba provést a ukládat je do interní entity Topic. Pro tu by poté existoval odběr používající *GroovyConnector* se skriptem, který by další akce (např. vytvoření datového zdroje a entity Topic pro studijní výsledky) prováděl.

4.5.6 Automatická správa entit Topic pro jednotlivé zkoušky

Automatická správa entit Topic pro jednotlivé zkoušky je popsána sekvenčním diagramem 4.20. Jedná se o průběh skriptu, který se spustí ve chvíli, kdy se změní počet zkoušek pro daný předmět (reprezentovaný entitou Topic a odebíraný *GroovyScripNotificator* s potřebným skritpem). Stejným způsobem je možné realizovat automatizaci pro jednorázové akce. Ve skriptu se načte entita Topic použitá jako šablona pro entity Topic jednotlivých zkoušek a existující entity Topic pro jednotlivé zkoušky. Poté použije *KOSApiConnector* pro stažení dat o všech zkouškách pro daný předmět a začne jimi iterovat. Pokud pro danou zkoušku již entita Topic existuje, přidá ji do průniku stažených a existujích. V případě, že entita Topic ještě neexistuje, vytvoří ji podle šablony a přiřadí odpovídající práva tak, aby byla dostupná pro všechny a výslednou entitu uloží. Nakonec odstraní všechny entity Topic pro zkoušky, které již v systému KOS nejsou.

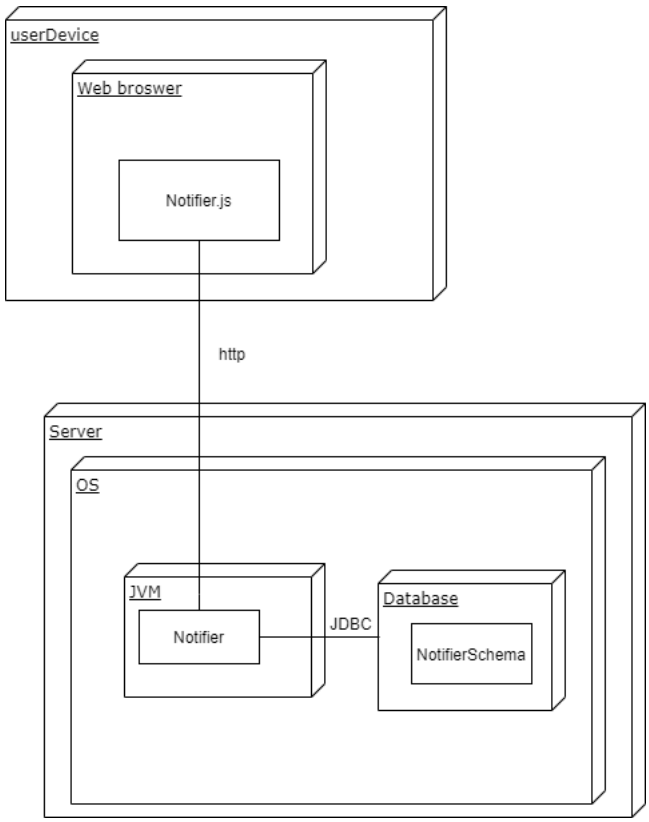


Obrázek 4.20: Sekvenční diagram automatické správy entit Topic pro jednotlivé zkoušky

4.6 Diagram nasazení

Aplikace je navržená tak, aby byla nasazená v prostředí, které odpovídá diagramu 4.21. Aplikace k fungování potřebuje:

1. JRE[4] verze alespoň 13.
2. Databázi vytvořenou na PostgreSQL databázovém serveru (ten může být fyzicky i na jiném zařízení).
3. Na straně uživatele prohlížeč se zapnutou podporou JavaScriptu.



Obrázek 4.21: Diagram nasazení aplikace

[4] Java Runtime Environment

Kapitola 5
Implementace

V této kapitole popisuji implementaci serverové a klientské strany aplikace. Popisuji návrh, použité technologie a důvody, proč jsem se pro ně rozhodl.

5.1 Architektura

V následující sekci popisuji architekturu zvolenou pro serverovou část aplikace.

5.1.1 Vrstvená architektura

Pro implementaci jsem zvolil vrstvenou architekturu. Tato architektura byla pro implementaci dostatečná. Jednou z výhod vrstvené architektury je jednoduché testování, kdy se dají jednotlivé vrstvy nahradit vrstvou testovací. Další výhodou je jednoduchý vývoj. Většina operací v aplikaci je synchronní a díky tomu je vývoj snažší a lépe se ladí.

Vrstvená architektura má i své nevýhody, mezi které patří například nižší flexibilita či horší škálovatelnost. Popis zvýšení flexibility řešení je popsán v Rozvoj architektury. Škálovatelnost aplikace se dá vyřešit správnou konfigurací. Pokud se spustí více instancí aplikace (zde by bylo samozřejmě nutné rozšířit datový model o instance), kde každý bude mít na starost vlastní množinu datových zdrojů a tyto množiny budou zároveň disjunktní, zvýší se tím výkon aplikace.

5.2 Použité technologie

5.2.1 Platforma Java

Jako platformu a jazyk implementace jsem zvolil jazyk Java. Zvolil jsem ji z následujících důvodů:

1. Multiplatformnost. Aplikaci je možné psát a spouštět na libovolné platformě.
2. Rozšířenost a komunita. Java je hodně rozšířená a pokud se při vývoji narazí na problém, není těžké najít odpověď.

3. Bohatá standardní knihovna.

4. Existence velkého množství knihoven, které usnadňují vývoj aplikací.

5.2.2 SpringBoot

Jako framework pro implementaci jsem zvolil SpringBoot. Spring je velice rozšířený a známy framework s velkou komunitou vývojářů. SprinBoot oproti samotnému Springu přidává následující:

- Nastavení, které je potřeba u Springu dělat ručně fungují ve SpringBootu out-of-the-box.
- Jednoduchá intergrace s technologiemi jako je JPA.
- Aplikaci lze spustit s embeded aplikačním serverem. Toto má výhody jak pro vývoj, tak pro jednoduché nasazení. Má to i své nevýhody, jako například nemožnost použít connection pool aplikačního serveru, ovšem v případě potřeby je možné i takovou aplikaci upravit a na aplikační server nasadit. Vývoj je tak rychlý a při problémech je možné rychle nalézt řešení v komunitě, internetových fórech či knihách věnujících se Springu a na něm postavených knihovnách.

5.2.3 PostgreSQL

Jako databázi jsem si vybral PostgreSQL. PostgreSQL je open-source relační databáze a pro účely zamýšleného použití je zdarma. Relační databáze se zde hodila lépe, jelikož data v aplikaci jsou strukturovaná a struktura se nemění (cílem modelu v této aplikaci je možnost mít různá data uložená ve stejných strukturách). Navíc je pro aplikaci velice důležité mít jistotu, že zápis dat bude atomický a databaze bude v konzistentním stavu. V opačném případě by aplikace mohla posílat nesmyslné notifikace, žádné notifikace nebo notifikace navíc.

5.2.4 JPA

Pro implementaci persistentní vrstvy jsem zvolil JPA. Výhodou této technologie je, že při vývoji datového modelu zároveň vyvýjím i databázový. Další výhodou je nezávislost na zvolené relační databázi. Pokud bych se později rozhodl přejít z aktuálně používaného PostgreSQL na MySQL či Oracle, stačí jen změnit databázový ovladač a aplikace může být bez dalších úprav v kódu používaná.

Při použití JPA je ale také potřeba být obezřetný. Někdy může jen zapomenutá reference na jedné ze stran vazby způsobit, že máme více tabulek než potřebujeme.

5.2.5 React.js, Blueprint.js, Redux, Redux-thunk

Pro implementaci klientské části aplikace byly zvoleny následující knihovny:

1. React.js - knihovna pro tvorbu uživatelských rozhraní v jazyce Javascript;
2. Blueprint.js - knihovna komponent založených na knihovně React;
3. Redux - knihovna pro správu stavu aplikace;
4. Redux-thunk - knihovna umožňující použití asynchronních funkcí v rámci knihovny Redux (například pro komunikaci se serverovou stranou).

Před implementací bylo zvažováno použití Vue.js na místo React.js. React.js je populárnější než Vue.js, a je tak snazší řešit problémy, které mohou nastat při vývoji. Zároveň pro React.js existuje více komponentových knihoven, a tak nebyl problém najít takovou, která vyhovuje potřebám aplikace. Z těchto důvodů byl zvolen React.js a knihovny, které jsou s ním spojené.

5.2.6 TypeScript

Krom knihovny pro tvorbu klientské části jsem se také rozhodl využít jazyk TypeScript. Jedná se o jazyk postavený nad JavaScriptem (kód v JavaScriptu je také validní v TypeScriptu). Přináší s sebou možnost použít typy v JavaScriptu. Tím zlepšuje kvalitu kódu, přidává jisté záruky a kontroly v času kompilace a zároveň zvýšenou podporu nápovědy ze strany editorů/IDE. Všechny knihovny použité pro vývoj klientské části aplikace mají podporu pro TypeScript.

Dalším kandidátem při výběru technologií byl ReasonML. ReasonML staví na již existujících principech funkcionálního programování a na jazyce OCaml. Přináší tak výhody funkcionálního programování a navíc je to jazyk ve kterém byl původně napsán React.js. Problémem je ale jeho rozšířenost, která není moc velká a chybějící definice typů pro existující JS knihovny. Ač by implementace v jazyce ReasonML byla zajímavá zkušenost, z praktických důvodů byl nakonec zvolen TypeScript.

Kapitola 6

Testování

6.1 Ověření flexibility modelu

V rámci práce bylo cílem mít dostatečně flexibilní model, který by mohl být použít i pro jiná data než data z KOSApi. Pro ověření jsem vytvořil několik entit Topic pro jiná data. Tyto entity slouží pro ukládání následujích dat:

1. Využití RAM a CPU (používající *SystemConnector*).
2. Počet uživatelů bez práv (používající *DBConnector*)
3. Údaje o stavu akcií společnosti Appple (používající *AplhavantageConnector*)
4. Údaje o stavu akcií společnosti IBM (používající *AplhavantageConnector*)

Uložení do struktur v aplikaci nebylo složité realizovat a návrh nebyl nijak omezující.

6.2 Testování v samotné aplikaci

Pro otestování zpracování dat z např. KOSApi je možné použít aplikaci samotnou. Krom zpracování je následující metodou otestováná kompletní funkčnost od detekce potřeby aktualizovat datový zdroj, přes zpracování dat až po detekci případné změny dat a notifikace přes notifikátor.

Postup:

1. Manuální stažení dat z KOSApi a jejich uložení na lokální úložiště.
2. Vytvoření nového datového zdroje používající *FSConnector* s atributem *path*, ve kterém je uvedená cesta k souboru s daty.
3. Vytvoření nové entity Topic do které mají být data uložena.
4. Vytvoření *GroovyNotificator* se skriptem, který načte data z entity Topic a porovná je s očekávanými výsledky.

Skript pak může nakonec výsledek buďto uložit a nebo například notifikovat při nesprávných datech.

6.3 Testovací scénáře

V této kapitole popisuji testovací scénáře aplikace z pohledu uživatele. Jedná se o průchody uživatelským rozhraním a ověření funkčnosti aplikace.

6.3.1 TC1: První přihlášení pomocí ČVUT SSO

Testovací scénář je popsán v tabulce 6.1

ID testu	**TC1_SSO_login**
Název testu	První přihlášení pomocí ČVUT SSO.
Popis testu	Ověřuje možnost přihlášení přes školní SSO.
Vstupní podmínky	Uživatel má aktivní školní účet.
Testovací data	
Kroky testu	
1.	Uživatel použije pro přihlášení možnost *Log in using ČVUT SSO*.
Očekávaný výsledek	Uživatel bude přesměrován na přihlašovací portál SSO.
2.	Uživatel vyplní své přihlašovací údaje a přihlasí se.
Očekávaný výsledek	Uživatel bude přesměrován do aplikace na úvodní stránku.

Tabulka 6.1: TC1: První přihlášení pomocí ČVUT SSO

6.3.2 TC2: Vytvoření konfigurace notifikačního kanálu

Testovací scénář je popsán v tabulce 6.2

ID testu	**TC2_CHANNEL_CONFIG**
Název testu	Vytvoření konfigurace notifikačního kanálu.
Popis testu	Ověřuje možnost vytvořit novou konfiguraci notifikačního kanálu a její uložení.
Vstupní podmínky	Uživatel je přihlášený.
Testovací data	
Kroky testu	
1.	Uživatel přejde na stránku *Notification channels*.
Očekávaný výsledek	Uživatel je na stránce *Notificatrion channels*.
2.	Uživatel v právém horním rohu použije tlačítko *+*.
Očekávaný výsledek	Aplikace zobrazí dialog pro vytvoření nové konfigurace.
3.	Uživatel vybere šablonu *OwnerMailNotificator*, vyplní název a použije tlačítko *Save*.
Očekávaný výsledek	Dialog se zavře a uživatel v seznamu konfigurací uvidí vytvořenou konfiguraci.

Tabulka 6.2: TC2: Vytvoření konfigurace notifikačního kanálu

6.3.3 TC3: Vytvoření odběru

Testovací scénář je popsán v tabulce 6.3

ID testu	**TC3_SUBSCRIPTION**
Název testu	Vytvoření odběru.
Popis testu	Ověřuje možnost odebírat změny entit Topic pomocí notifikačních kanálů.
Vstupní podmínky	Uživatel je přihlášený, má alespoň jednu vytvořenou konfiguraci notifikačního kanálu a dostatečná práva pro alespoň jednu entitu Topic.
Testovací data	
Kroky testu	
1.	Uživatel přejde na stránku *Topics*.
Očekávaný výsledek	Uživatel je na stránce *Topics*.
2.	Uživatel vybere jeden z jemu dostupných entit Topic a na ten klikne.
Očekávaný výsledek	Aplikace zobrazí dialog pro odběr změn.
3.	Uživatel klikne na tlačítko *+* a ze seznamu dostupných konfigurací jednu vybere a klikne na *Add*.
Očekávaný výsledek	V seznamu odběrů se objeví nově přidaná konfigurace.

Tabulka 6.3: TC3: Vytvoření odběru

6.3.4 TC4: Odstranění konfigurace notifikačního kanálu

Testovací scénář je popsán v tabulce 6.4

ID testu	**TC4_CHANNEL_DETELE**
Název testu	Odstranění konfigurace notifikačního kanálu.
Popis testu	Ověřuje možnost odstranit konfiguraci notifikačního kanálu a zároveň zrušení všech odběrů, které ji používají.
Vstupní podmínky	Uživatel je přihlášený a má nakonfigurovaý alespoň 1 odběr.
Testovací data	
Kroky testu	
1.	Uživatel přejde na stránku *Notification channels*.
Očekávaný výsledek	Uživatel je na stránce *Notification channels*.
2.	Uživatel vybere jednu z konfigurací, která je použita pro alespoň 1 odběr a klikne na tlačítko s ikonou koše.
Očekávaný výsledek	Konfigurace bude odstraněna v dialogu u entity Topic, pro kterou byl použit již nebude konfigrace přítomna.

Tabulka 6.4: TC4: Odstranění konfigurace notifikačního kanálu

Kapitola 7

Závěr

Cílem bakalářské práce bylo realizovat systém, který bude sledovat změny v systému KOS prostřednictvím KOSApi a následně notifikovat uživatele. Důležitou funkcionalitou bylo přihlášení přes školní SSO, aby stundeti měli systém dostupný out-of-the-box. Sledované změny se týkaly zkoušek, jednorázových akcí, paralelkách a předmětů samotných. Aby byl systém dále udržitelný, bylo požadavkem i administrační rozhraní.

Začal jsem analýzou, kde jsem identifikoval případy užití, navrhl model aplikace a procesů v ní. Poté jsem vybral technologie tak, aby systém byl dlouhodobě udržitelný a bez placených komponent. Následně jsem se seznámil s rozhraním KOSApi, školním SSO a způsoby integrace s ním.

V rámci realizace jsem implementoval serverovou a klientskou část aplikace a tu následně testoval. Jelikož je systém navržen pro obecné použití, bylo pro potřeby splnění cílů provést konfiguraci. První konfigurací byly šablony pro datové zdroje a entity pro uložení studijních výsledků studenta. Následovala konfigurace integrace na školní SSO, pro kterou byla existence předchozích šablon prerekvizitou. Následovala konfigurace šablon entit pro jednotlivé zkoušky a datového zdroje s entitou pro počet zkoušek předmětu. Na změny v počtu zkoušek reaguje systém skriptem, který upravuje entity pro jednotlivé zkoušky. Konfigurace pro jednorázové akce, paralelky a jednotlivé předměty je pak už jen duplikace a drobná úprava předchozích. Touto konfigurací jsou splněny všechny cíle související se stažením dat a notifikací relevantních změn v systému KOS.

Díky obecnému návrhu lze aplikaci použít pro integraci na jiné systémy než jen KOS. Přidání nových datových zdrojů je snadno realizovatelné za použití existujících konektorů a v případě potřeby nového se jedná jen o implementaci jednoho rozhraní. Pro zpracování nejčastějších formátů odpovědí (JSON a XML) jsou k dispozici již hotové mechanismy a entity pro uložení jsou dostatečně flexibilní, aby do nich bylo možné různorodá data uložit. Toto bylo ověřeno vytvořením konektoru na Alpha Vantage[1] a ukládáním informací o cenách akcií.

Klientská část aplikace umožňuje uživatelům využít školní SSO pro přihlášení, konfiguraci notifikačních kanálů a nastavení odběrů. Klientská a serverová část jsou separátní aplikace a v případě potřeby jiného rozhraní

[1] https://www.alphavantage.co

pro uživatele (například mobilní aplikace) je možné využít RESTové rozhraní serverové části.

Administrační rozhraní je realizované jako RESTové rozhraní. Tvorba UI je patří k dalšímu rozvoji a je uvedena v kapitole Další rozvoj s rozvojem architektury.

Aplikace je dostupná na webové adrese https://www.notificator.strazovan.eu/, kde je funkční instance s možností přihlášení přes školní SSO a s vytvořenými testovacími daty. V budoucnu bych rád domluvil nasazení, konfiguraci a možnost dalšího rozvoje v rámci FEL, tak aby aplikace byla plně dostupná všem studentům a bez omezení mými prostředky.

Kapitola 8
Další rozvoj

8.1 Rozvoj architektury

Jak bylo již zmíněno v sekci Architektura, zvolená vrstvená architektura není moc flexibilní. Navíc, pokud bychom chtěli přidávat nové notifikátory či konektory, nebo dělat jejich aktualizaci s opravami, bylo by potřeba sestavovat a nasadit celou aplikaci znovu. V rámci dalšího rozvoje by bylo lepší systém rozdělit na jádro, které by se staralo o data, aktualizace dat, notifikace a poskytovalo by RESTové rozhraní pro klientské aplikace. Dále by existovaly samostatně další služby, které by fungovaly jako jednotlivé konektory a notifikátory. Tyto by byly s jádrem aplikace spojeny pomocí RabbitMQ[1].

8.2 Administrační rozhraní

Veškeré konfigurace datových zdrojů, šablon a entit Topic byly v rámci bakalářské práce vytvářeny a modifikovány pomocí přímé komunikace s RESTovým rohzraním přes aplikaci Postman. Jako další rozvoj se nabízí tvorba administrátoského rozhraní, které tyto operace dovolí pohodlně provádět a zároveň umožní rychlý přístup k monitorování aplikace.

[1] https://www.rabbitmq.com/

Literatura

[1] FOOTE, Brian a Joseph YODER, 1998. *Metadata and Active Object-Models* [online]. Department of Computer Science University of Illinois [cit. 2020-05-20]. Dostupné z: https://joeyoder.com/PDFs/metadata.pdf. Department of Computer Science University of Illinois.

[2] ALEX, Ben, Luke TAYLOR, Rob WINCH, et al., 2020. *Spring Security Reference* [online]. [cit. 2020-05-20]. Dostupné z: https://docs.spring.io/spring-security/site/docs/5.3.2.RELEASE/reference/html5/

[3] GOESSNER, Stefan. *JSONPath - XPath for JSON* [online]. 2007 [cit. 2020-05-20]. Dostupné z: https://goessner.net/articles/JsonPath/

[4] FOWLER, Martin, 2003. *Patterns of enterprise application architecture*. Boston: Addison-Wesley. Addison-Wesley signature series. ISBN 03-211-2742-0.

[5] GAMMA, Erich, c1995. *Design patterns: elements of reusable object-oriented software*. Boston: Addison-Wesley. Addison-Wesley professional computing series. ISBN 02-016-3361-2.

[6] RIEHLE, Dirk, 1997. *A Role-Based Design Pattern Catalog of Atomic and Composite Patterns Structured by Pattern Purpose* [online]. [cit. 2020-05-20]. Dostupné z: https://www.ubilab.org/publications/print_versions/pdf/ubilab-tr-97-1-1.pdf

[7] MARTIN, Robert C., Dirk RIEHLE a Frank BUSCHMAN, 1994. *Pattern languages of program design 3*. Upper Saddle River: Addison-Wesley. ISBN 0201310112.

[8] MONOLESCU, Dragos, Markus VOELTER a James NOBLE, 2006. *Pattern languages of program design 5*. Upper Saddle River: Addison-Wesley. ISBN 03-213-2194-4.