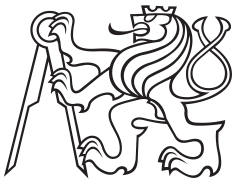
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická

Přechod aplikace FELSight na mikroslužbovou architekturu

Ladislav Svoboda

Vedoucí: Ing. Miroslav Balík, Ph.D.
Studijní program: Otevřená Informatika
Specializace: Software
Květen 2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Svoboda** Jméno: **Ladislav** Osobní číslo: **491936**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Specializace: **Software**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Přechod aplikace FELSight na mikroslužbovou architekturu

Název bakalářské práce anglicky:

Migration of FELSight application to microservice architecture

Pokyny pro vypracování:

Cílem práce je přesunout část funkcionalit aplikace FELSight, za použití vhodných technologii a postupů softwarového inženýrství, na architekturu více vhodnou pro tento projekt.
Cíle práce jsou následující:
1) Prostudujte práci Kohouta [1] a vyberte jaké funkcionality se budou migrovat.
2) Rozdělte funkcionality do vhodných mikroslužeb, vytvořte jejich návrh a implementujte alespoň 2.
3) Vytvořte základ služby pro integraci do fakultní platformy.
4) Nahraďte části monolitu nově vzniklými službami.
5) Pokryjte služby integračními testy.
6) Vyhodnoťte nové služby, možné problémy a přínosy.

Seznam doporučené literatury:

1. Kohout, A. (2022). Transformace architektury aplikace FelSight [Diplomová práce]. České vysoké učení technické v Praze.
2. Carnell, J., & Sánchez, I. H. (2021). Spring microservices in action. Simon and Schuster.
3. Nadareishvili, I., Mitra, R., McLarty, M., & Amundsen, M. (2016). Microservice architecture: aligning principles, practices, and culture. " O'Reilly Media, Inc.".

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Miroslav Balík, Ph.D. katedra teoretické informatiky FIT

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

Datum zadání bakalářské práce: **09.02.2023** Termín odevzdání bakalářské práce: **26.05.2023**

Platnost zadání bakalářské práce: **22.09.2024**

Ing. Miroslav Balík, 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 hlavně Ing. Miroslavu Balíkovi, Ph.D. za jeho odborné vedení, rady a námitky v průběhu psaní této práce. Dále bych chtěl poděkovat své rodině a přítelkyni za podporu po celou dobu studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškerou použitou literaturu.

V Praze, 7. května 2023

Ladislav Svoboda

Abstrakt

FELsight je aplikace, která poskytuje studentům různé funkcionality, které jim mají ulehčovat studium. Například zobrazování rozvrhů nebo jídelníčky menz. Náplní této práce bylo vybrat dvě funkcionality poskytované portálem pro studenty FELSight a vytvořit pro ně nové mikroslužby. Práce také přiblíží jaké funkce FELSight poskytuje, jak vypadá jeho současná architektura a jaké problémy to přináší. V rámci práce byly mikroslužby pokryty testy a byly připraveny na nasazení na vývojářské prostředí a integraci do stávajícího monolitu. Nakonec je také představen základ pro integraci funkcionalit, kterými se tato práce zabývá, do fakultní platformy HUB.FEL. Ta má za úkol sjednotit všechny aplikace vyvíjené CZM na jedno místo.

Klíčová slova: aplikace FELSight, Java

Vedoucí: Ing. Miroslav Balík, Ph.D.

Abstract

FELsight is an application that provides students with various functionalities to help them with their studies. For example, viewing timetables or menus for school buffets.The goal of this thesis was to select two functionalities provided by the student portal FELSight and create new microservices for them. The thesis also outlined all of the functionalities provided by FELSight, what its current architecture looks like and what problems it poses. As part of the thesis, the microservices were covered by tests and were prepared to be deployed to the development environment and integrated into the monolith. Finally, the basis for integrating the functionalities addressed in this work into the faculty platform HUB.FEL is also presented. HUB.FEL is intended to unify all applications developed by CZM in one place.

Keywords: application FELSight, Java

Obsah

Obrázky

Tabulky

Kapitola 1
Úvod

V úvodní kapitole se budeme zabývat úvodem do problematiky a strukturou práce.

1.1 Téma práce

Tato práce realizuje migraci části logiky aplikace FELSight na architekturu mikroslužbovou. Práce navazuje na diplomovou práci *The Architecture Transformation of FELSight Faculty Application*[1] Adama Kohouta. FELSight je aplikace vyvíjená Centrem znalostního managementu Fakulty elektrotechnické, poskytovaná studentům a učitelům fakulty, jimž nabízí podpůrné funkce při studiu. V rámci práce se budeme věnovat hlavně migraci funkcionality zobrazení a plánování rozvrhu, dále pak integraci již existující Room Service do reálného chodu aplikace. Room Service je služba, která vznikla jako ukázka pro *The Architecture Transformation of FELSight Faculty Application*[1] Adama Kohouta.

Motivací pro změnu architektury je stárnoucí podoba Java EE aplikace, u které dochází k navyšování technického dluhu a složitosti údržby. Také kvůli využité technologii JSF pro implementaci frontendu je aplikace oproti moderním řešením nesrovnatelně pomalejší. Pro efektivní přechod na moderní frontendové řešení je nejprve třeba celou aplikaci zmigrovat do mikroslužbové architektury. Výstupem bude nová verze monolitu, napojená na dvě nové mikroslužby. První poskytuje funkcionalitu zobrazování rozvhu uživatele, jeho plánování do dalšího semestru a export do kalendáře; druhá poskytuje data o místnostech na fakultě.

1.2 Obsahový rámec práce

V první části práce přiblížíme aplikaci FELSight, její funkce a její současnou architekturu. Dále popíšeme, jaké problematiky přináší právě současná architektura.

V druhé částí popíšeme přínosy migrace a jaké bude obsahovat kroky. Také blíže popíšeme funkcionalitu částí, které budeme migrovat. Poté vysvětlíme, jaké nové služby budou vznikat, jak budou fungovat, a jak bude vypadat jejich

architektura. V rámci této části práce se také věnujeme vysvětlení fakultní IT platformy HUB.FEL.

V poslední části práce popíšeme, jak bude vypadat uvedení nově vzniklých služeb do chodu na vývojářském prostředí. Také v rámci toho představíme změny, které musí na prostředí proběhnout, aby nasazení nových služeb bylo možné, a jak budou vypadat testy pro tyto služby. Dále je zde zmíněno, jaké již existující služby využijeme pro naše prostředí.

Kapitola 2
Současný stav

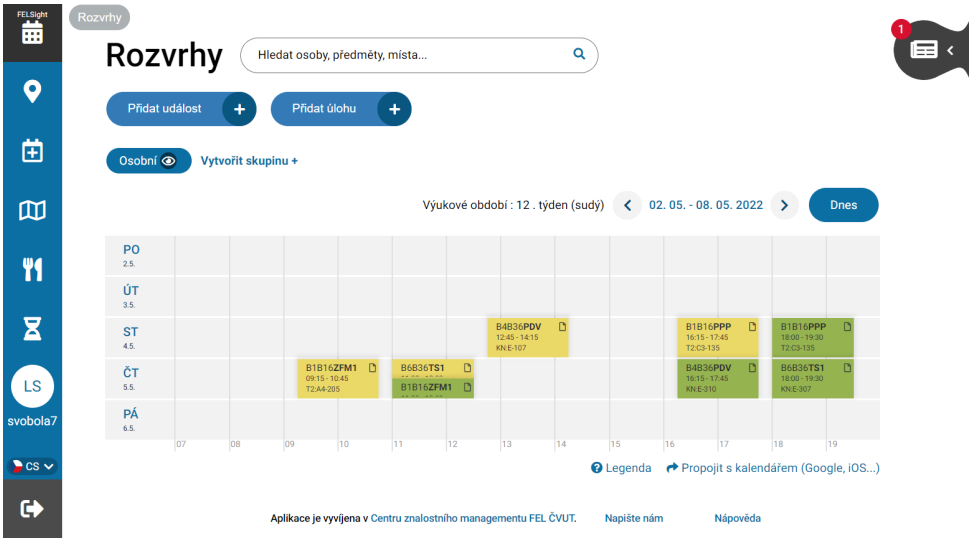
V této kapitole se budeme věnovat současné architektuře a popisu uživatelské části aplikace FELSight. V závěru probereme problémy, které současné řešení přináší.

2.1 Funkce aplikace FELSight

Studentský portál FELSight nabízí studentům funkce, které jim mají ulehčit studentský život. Portál je rozdělený do sekcí nabízejících různé služby.

2.1.1 Úvodní sekce - Rozvrhy

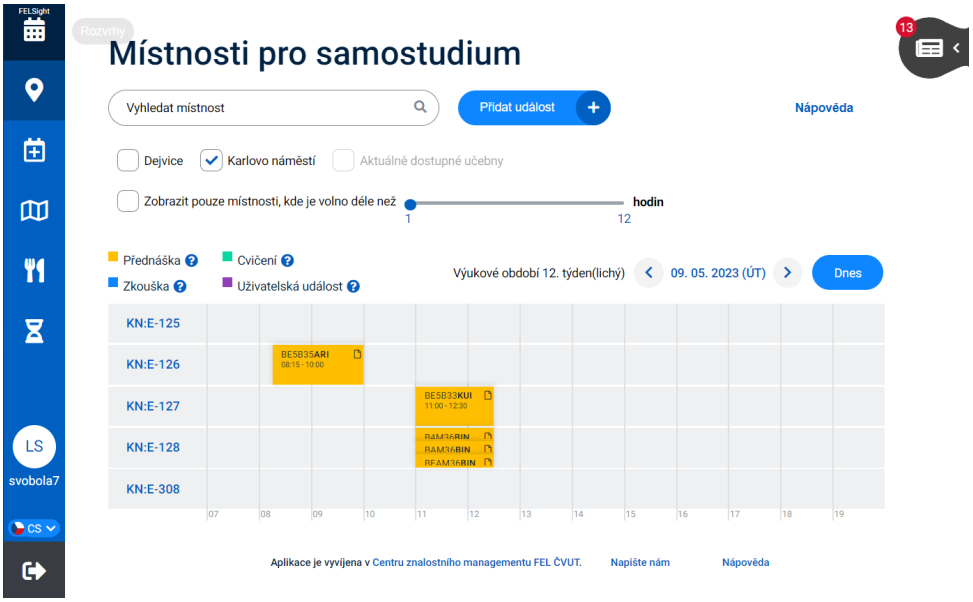
Tato sekce dovoluje uživateli zobrazit svůj rozvrh nebo pomocí vyhledávání v horní liště přidat do zobrazeného rozvrhu jiného uživatele, skupinu, místnost či kurz. Zobrazený rozvrh lze po týdnech posouvat. Uživateli je také zobrazena informace o tom, zda-li se jedná o týden výukový, zkouškový nebo o prázdniny. Dále je uživateli nabízena možnost si zkopírovat odkaz, který může následně vložit do svého kalendáře a svůj rozvrh do něj exportovat. Na obrázku 2.1 je zobrazen uživatelův osobní rozvrh.



Obrázek 2.1: Úvodní stránka zobrazující uživatelův rozvrh.

2.1.2 Místnosti pro samostudium

Tato sekce dovoluje zobrazit uživateli místnosti pro samostudium a jejich rozvrh ve zvoleném týdnu. Dále je možné vyfiltrovat prázdné místnosti a tím dovolit studentovi najít místnost a zadat do ní událost; tu uvidí ostatní studenti. Místnosti je také možné filtrovat podle toho, zda se nachází v Dejvicích nebo na Karlově náměstí. Ukázku filtrování podle místa můžeme vidět na obrázku 2.2.

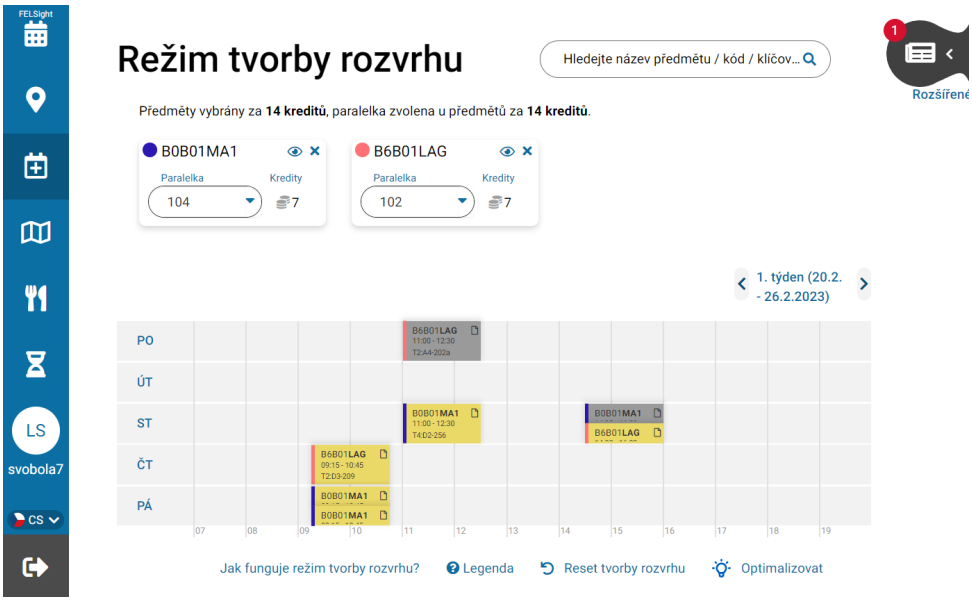


Obrázek 2.2: Sekce Rozvrhy pro samostudium.

2.1.3 Plánování rozvrhu

Tato sekce dovoluje uživateli naplánovat si rozvrh na nadcházející semestr. Uživatel má možnost zobrazit si svůj rozvrh podle vybraných laboratoří a cvičení. Na obrázku 2.3 můžeme vidět příklad rozvrhu s vybranými paralelkami. Také je automaticky upozorněn na přítomnost předmětů s nepravidelným rozvrhem. Předmět s nepravidelným rozvrhem může mít buď to rozvrh, který se střídá v sudém a lichém týdnu, nebo je úplně nepravidelný a v různých týdnech se rozvrh nepravidelně mění. Pokud nemá žádné nepravidelné předměty, je rozvrh zobrazen pouze pro jeden týden - všechny týdny jsou stejné.

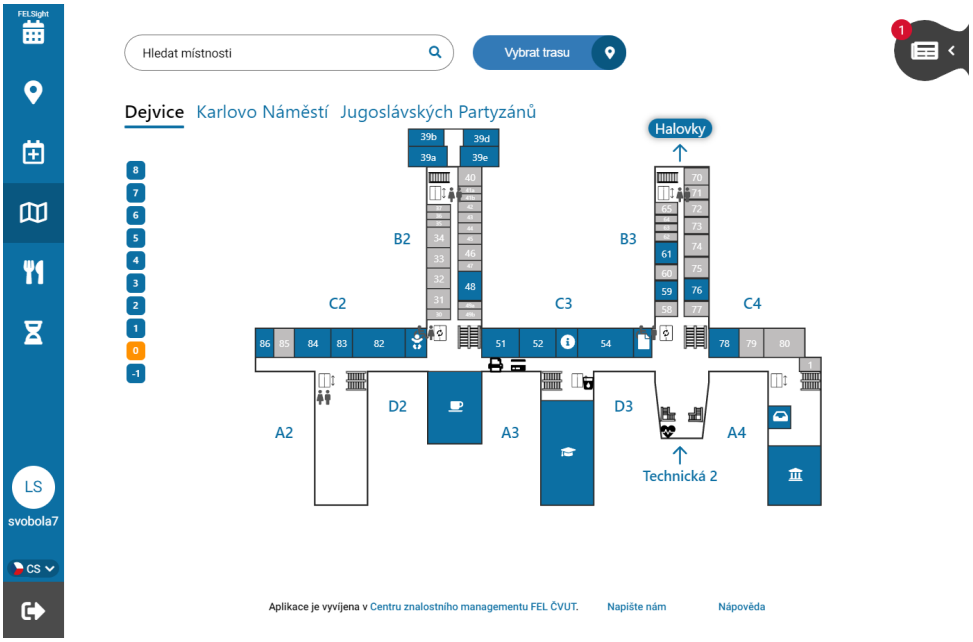
Při přítomnosti předmětu, jehož rozvrh se liší v sudých a lichých týdnech, dostane uživatel možnost přepínat mezi tím, jestli se zobrazuje rozvrh pro sudý nebo lichý týden. Po přidání nepravidelného předmětu se uživateli zobrazí výběr týdnů, kde si může zobrazit svůj rozvrh pro daný týden semestru. Týdny, ve kterých je rozvrh nepravidelný, jsou označeny.



Obrázek 2.3: Sekce plánování rozvrhu.

2.1.4 Plánek budov

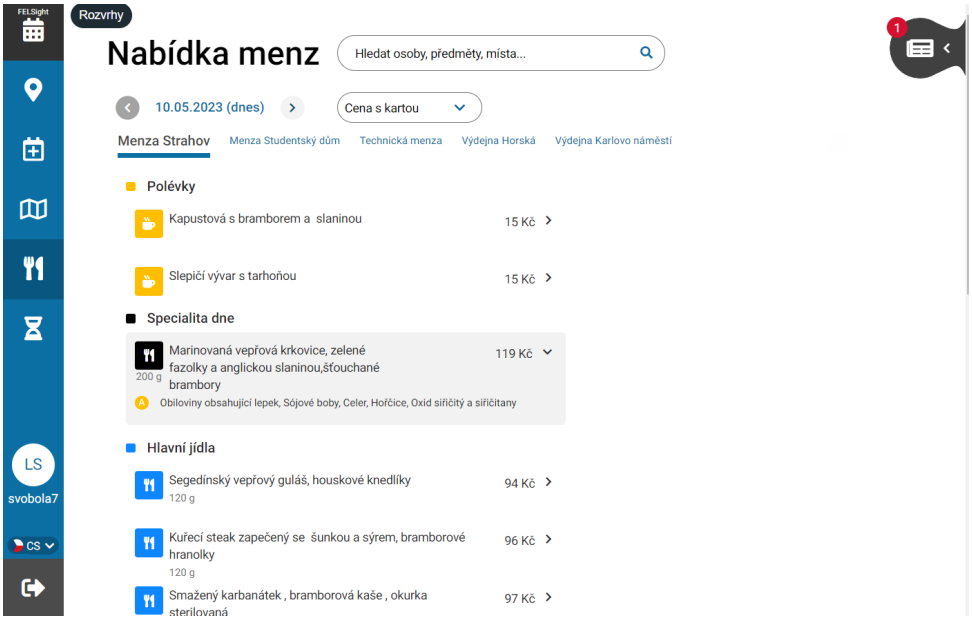
Tato sekce, zobrazená na obrázku 2.4, umožňuje uživateli navigaci po budovách FEL ČVUT. Místnost je možné najít pomocí vyhledávače v horní liště nebo rozkliknutím detailu místnosti v rozvrhu a následným kliknutím na políčko „zobrazit na mapě“. Po kliknutí je vybraná místnost označena za cíl a jako start je automaticky označen vchod budovy. Pokud si uživatel přeje zvolit jiný start než ten základní, stačí kliknout na políčko začátek a vyhledat konkrétní místnost.



Obrázek 2.4: Sekce Plánek budov.

2.1.5 Jídelníčky

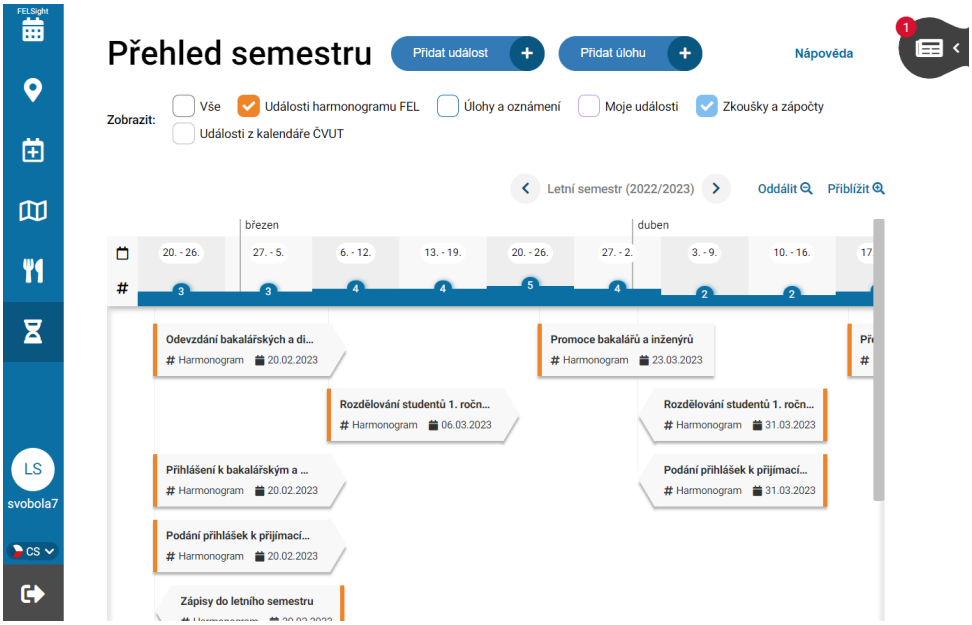
Sekce "Jídelníčky" umožňuje zobrazit si jídelníček zvolené menzy pro současný den nebo pro dny v budoucnosti, pro které jsou již jídelníčky doplněny. Pokud menza na zvolený den nemá žádné menu, není potom zobrazena ve výběrové liště. Uživatel si také může vybrat, zda-li chce zobrazit cenu se slevou pro studenty nebo cenu bez slevy. Jídla jsou rozdělena do kategorií, které udává Menza. Uživateli se po zobrazení detailu pokrmu zobrazí alergeny, které jídlo obsahuje. Sekci s otevřeným detailem můžeme vidět na obrázku 2.5.



Obrázek 2.5: Sekce Jídelníčky menz.

2.1.6 Přehled semestru

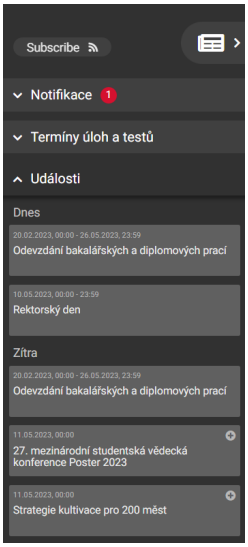
V této sekci si uživatel může zobrazit důležité události semestru pomocí časové osy. Události se dají filtrovat podle toho, zda se jedná o uživatelovy vlastní události, události z kalendáře ČVUT, události související s harmonogramem ČVUT FEL nebo jestli jsou to termíny zápočtových testů a zkoušek. Na obrázku 2.6 můžeme vidět harmonogram s nastaveným filtrem pro události harmonogramu ČVUT.



Obrázek 2.6: Sekce Přehled semestru.

2.1.7 Notifikační lišta

Na pravé straně obrazovky se nachází vyjíždějící lišta, ta obsahuje tři sekce. V sekci “Notifikace” se uživateli zobrazí upozornění - například o ohodnocení odevzdaného úkolu nebo o změně skupiny. Dále je zde sekce „Termíny úloh a testů“, ve které se zobrazují termíny odevzdání úkolů a testy, které se dnes, a v příštím týdnu, konají. Nakonec je zde sekce „Události“, kterou můžeme vidět na obrázku 2.7. V sekci „Události“ se zobrazují vlastní, harmonogramové a všeobecné události z ČVUT.



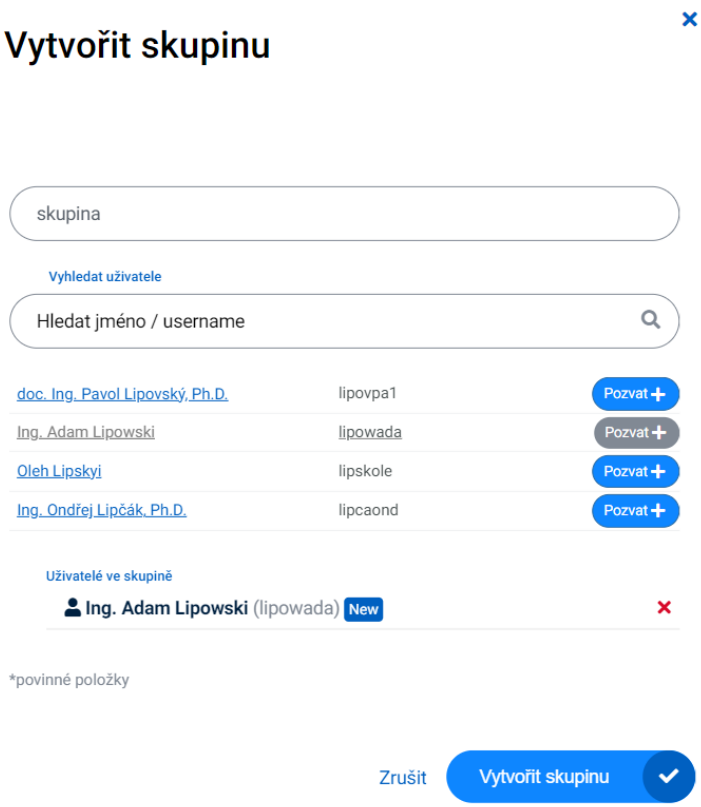
Obrázek 2.7: Notifikační lišta.

2.1.8 Moje...

V sekci „Moje...“ si uživatel může v tabulkovém pohledu prohlédnou všechny své události, známky Moodle, notifikace a skupiny, jejichž je členem.

2.1.9 Skupiny

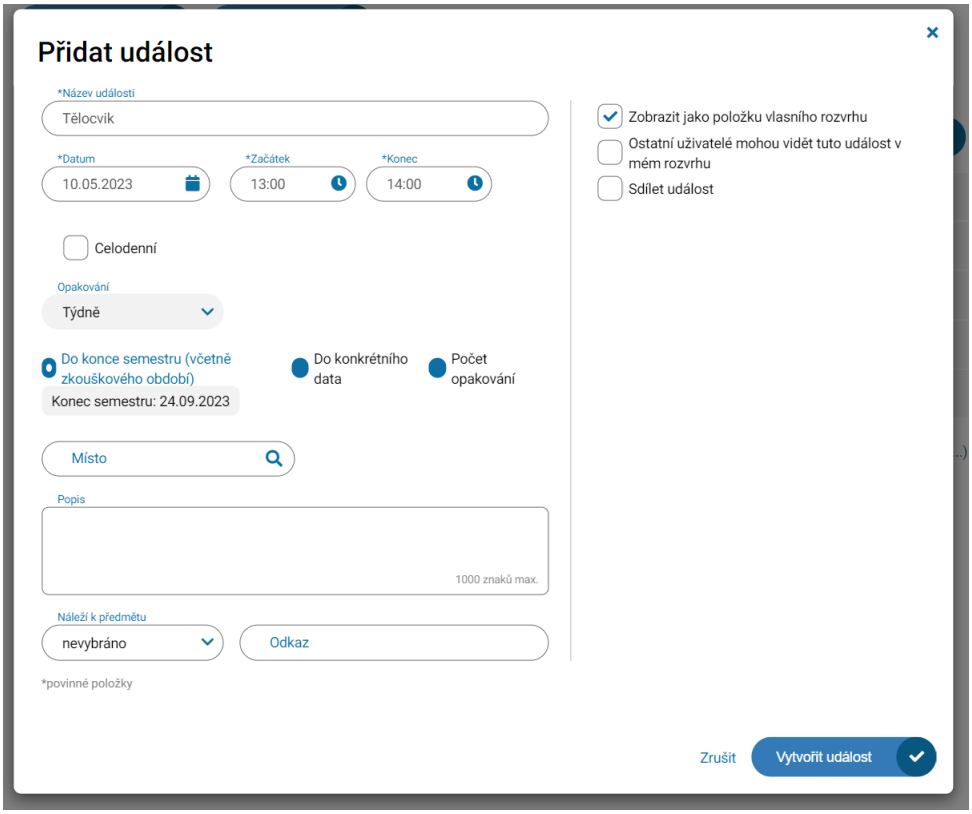
Aplikace umožňuje vytvořit skupinu a přizvat do ni ostatní uživatele za pomocí dialogu zobrazeném v obrázku 2.8. Po přijmutí pozvánky je pak možné si zobrazit rozvrh všech uživatelů skupiny nebo vytvářet události, do kterých jsou ostatní uživatelé automaticky pozváni.



Obrázek 2.8: Dialog vytvoření nové skupiny uživatelů.

2.1.10 Vlastní události

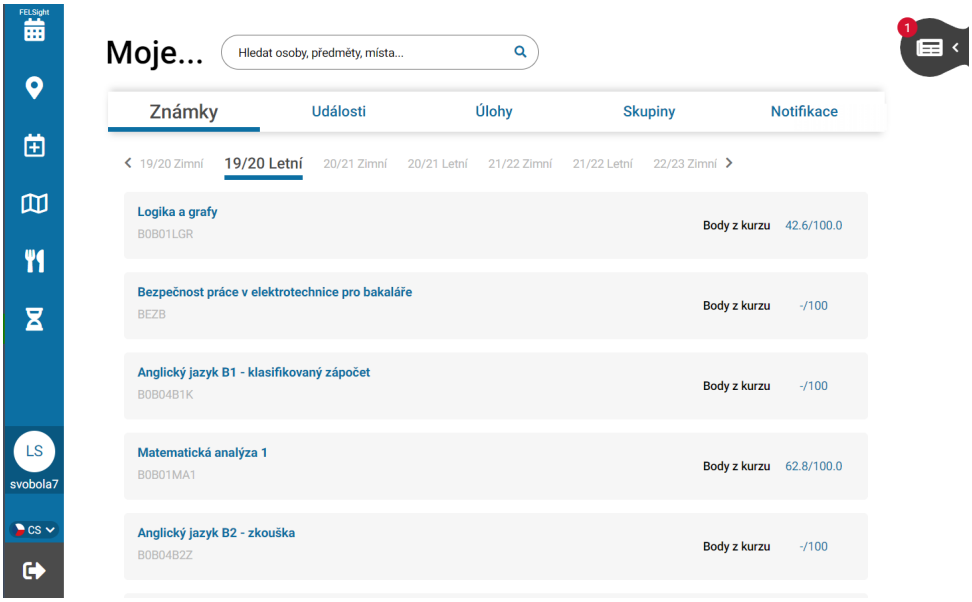
Uživatel má možnost do svého rozvrhu přidat sebou vytvořenou událost. U té může za pomocí dialogu na obrázku 2.9 specifikovat místo, předmět či přidat její popis. Lze také pozvat ostatní uživatele. Zároveň je možné nastavit, aby byla událost opakována. Počet opakování události se dá nastavit buď podle data posledního výskytu události, a nebo podle počtu týdnů, kdy se má událost opakovat. Stejně tak je možné vytvořit událost pro skupinu.



Obrázek 2.9: Dialog pro vytvoření nové vlastní události.

2.1.11 Moodle známky

Zde si uživatel může zobrazit své současné a předešlé známky zadané vyučujícími do fakultního Moodle[2]. Moodle[3] je open-source platformou pro vzdělávání. Na FEL ČVUT je využívána hlavně ke sdílení studijních materiálů, zadávání domácích úkolů a zadávání výsledků testů, úkolů a zkoušek. Tuto sekci můžeme vidět na obrázku 2.1.11



Obrázek 2.10: Sekce „Moje...“ zobrazující záložku Moodle známky.

2.2 Technická podoba aplikace FELSight

V této sekci přiblížíme stávající architekturu aplikace, její technologie a problémy, kterou stávající řešení způsobuje.

2.2.1 Technologie

FELSight je monolitickou aplikací, napsanou v platformě Java EE[4]. Monolitickou aplikací je aplikace, ve které je uživatelské rozhraní, byznysová pravidla a kód pro získání dat spojen do jednoho spustitelného programu, který se nasazuje na jedno místo[5].

FeLSight využívá PostgreSQL[6] databáze a je nasazován na aplikační server Payara[7].

Java EE (novější verze jsou známé pod názvem Jakarta EE[8]) je součást platformy Java, zaměřená na tvorbu podnikových aplikací. Obsahuje specifikace a rozhraní pro knihovny, jako jsou například JavaServer Faces a Java Enterprise Beans. Specifikace udávají chování, které musí knihovny třetích stran splňovat a uživatel tedy ví jaká funkčnost mu bude knihovnou poskytnuta.

2.2.2 Moduly

FELSight je rozdělený do tří Java modulů, jimiž jsou Web, EJB a EAR.

2.2.2.1 Web

Web obsahuje uživatelské rozhraní a logiku pro jeho interakci se servisní vrstvou. Pro vývoj uživatelského rozhraní, je použita specifikace JSF(JavaServer Faces)[9], která je poskytována právě Javou EE. Struktura rozhraní je specifikována v .xhtml souborech, ve kterých jsou také specifikovány akce pro uživatelovu interakci s rozhraním. Data v rozhraní jsou podávána dynamicky při vykreslování stránky na serveru. Využívají se zde odkazy na registrované komponenty projektu zvané Java Enterprise Beans nebo se zde specifikují volání Javascriptových funkcí.

Jazyk JavaScript je využíván pro operace s již vykreslenou stránkou pro animace a dynamické změny obsahu webu - bez potřeby stránku překreslit pomocí serveru. Příkladem toho je skrývání položek podle současně rozkliknutého okna.

2.2.2.2 EJB

Modul EJB obsahuje servisní vrstvu, model a logiku pro komunikaci s externími zdroji. Všechny registrované komponenty využívají vkládání závislostí, které nám ulehčuje správu životního cyklu komponent. Jsou zde také specifikované třídy, které reprezentují databázový model. Dále se zde nachází logika pro pravidelné aktualizace, ty se pouštějí každý den ráno. FELSight se dotazuje na data KOSapi[10] a Sirius API[11]; tato data si poté ukládá k sobě do databáze pro rychlejší přístup.

2.2.2.3 EAR

Modul EAR slouží pouze k sestavení finálního souboru, který poté může být nasazen. Neobsahuje žádný kód, pouze definice Gradle[12] úkolů, které slouží k sestavení aplikace.

2.2.3 Existující mikroslužby

Vedle monolitu ještě běží tři malé, na sobě nezávislé aplikace, kde každá zajišťuje odpovědnost za vymezený celek logiky. Jsou to služby pro jídelníčky, navigaci a známky Moodle. Navigace je napsána v React.js, zbylé 2 aplikace jsou vytvořeny pomocí Spring Boot[13]. Ten je blíže popsán v kapitole 3.7.1.

2.3 Problémy se stávající architekturou

Hlavním problémem spojeným se současnou architekturou je dlouhá doba načítání stránky. Kvůli tomu, že JSF jsou vytvořeny na serveru, a až potom odeslány uživateli, vzniká problém jak s délkou čekání na dokončení celého požadavku, tak s objemem posílaných dat. Právě kvůli vykreslování na serveru se musí vždy posílat všechny vyskakovací dialogy a další okna, která si uživatel může ze současné stránky zobrazit. Také se musí při každé změně stránky

posílat znovu. Kvůli tomu aplikaci dlouho trvá úvodní načtení stránky. Při každé změně sekce se tento problém opakuje.

Dále se za dlouhá léta vývoje aplikace a přidávání funkcionalit rozrostla velikost kódu tak, že již neexistuje nikdo, kdo by byl seznámen s kódem celé aplikace. To také znamená, že noví stážisté, kteří většinou právě na FELSight začínají, jsou hned ze začátku vystaveni složitému a objemnému celku. Pochopení částí logiky je složitější kvůli velké provázanosti kódu, které nedovoluje se s logikou seznamovat po malých kusech.

Kapitola 3

Migrace aplikace

V této kapitole se budeme blíže věnovat přínosům migrace, jejím postupem a nakonec výběrem a popisem funkcionalit, které budeme migrovat. Dále tato část vysvětluje architekturu a požadavky na nové služby - jak obecné, tak specifické.

3.1 Přínos migrace

Pokud bychom chtěli namísto knihovny JSF využít samostatnou aplikaci pro uživatelské rozhraní, v našem případě je tím zamýšlena aplikace využívající React.Js[14], tak by muselo dojít k nahrazení všech komponent pro komunikaci s JSF rozhraními. Nová komponenta by musela poskytovat REST rozhraní. FELSight v současné době obsahuje pouze REST rozhraní pro získávání kalendářů. Změnou rozhraní by se ale vyřešil pouze problém s uživatelským rozhraním.

Adam Kohout ve své práci[1] říká, že i když teoreticky díky rozdělením do modulů dochází k částečnému rozdělení polí působností, prakticky tomu tak není z důvodu shromáždění většiny byznys logiky do EAR modulu. Z toho důvodu chceme monolit postupně rozdělovat do dílčích mikroslužeb. Při správném rozdělení tak dojde k rozdělení působností a rozdělení logiky do menších dílčích celků. Dále rozdělení ulehčí zaučování nových programátorů díky zmenšení částí kódu, se kterými se musí v jednu chvíli seznamovat. V poslední řadě nám také mikroslužby dovolí zprovoznit vybrané funkcionality FELSight na fakultní platformě HUB.FEL. Tu si blíže popíšeme v kapitole 3.6.1.

3.2 Postup migrace

Postup migrace bude rozdělený do následujících kroků:

1. Výběr funkcionalit k migraci,
2. implementace nových mikroslužeb,
3. napojení nových mikroslužeb k monolitu,

4. nasazení mikroslužeb na testovací prostředí,
5. předprodukční testování,
6. nasazení na produkční prostředí.

3.3 Výběr funkcionalit

Pro migraci v rámci této práce jsme za pomoci anýzy od Adama Kohouta vybrali dvě funkcionality, které budeme migrovat[1]. K tomu ještě budeme převádět POC od Adama Kohouta do reálné byznys aplikace. POC přesouvá sekci "Místnosti pro samostudium". Další dvě sekce, které jsme vybrali, jsou úvodní sekce. První slouží k zobrazování rozvrhů a druhá k plánování rozvrhu na příští semestr. Tyto sekce byly vybrány z toho důvodu, že se jedná o stěžejní funkce aplikace, jež jsou nejvíce využívány uživateli.

3.4 Detailní popis migrovaných funkcionalit

Sekce „Rozvrhy" slouží zároveň jako úvodní stránka aplikace, která se zobrazuje uživateli automaticky ihned po přihlášení. Prvotně se uživateli zobrazí jeho rozvrh s možností pohybovat se v něm dle zvoleného týdne. Uživatel si také může dole zkopírovat link, který po vložení do kalendářové aplikace (např. Google Calendar[15] nebo Microsoft Outlook[16]) poskytuje kalendáři data o uživatelově rozvrhu. Uživatel si také může zobrazit rozvrh jiného studenta, učitele, místnosti či kurzu. Rozvrhy se získávají z aplikace Sirius API, která rozvrhy překládá z dat poskytnutých komponentou studia univerzitního systému zvanou KOS[17].

3.4.1 Plánování rozvrhu

Sekce „Plánování rozvrhu" umožňuje uživateli naplánovat si rozvrh na příští semestr. Tento rozvrh si aplikace ukládá do své databáze, uživatel si tak nemusí rozvrh pokaždé znovu předělávat. Zobrazený rozvrh se mění podle toho, zda má uživatel přidaný nějaký předmět, kterému se liší rozvrh v sudém a lichém týdnu a nebo má rozvrh uplně nepravidelný. V případě sudého/lichého týdne se uživateli zobrazí možnost přepnout mezi zobrazením rozvrhu sudého/lichého týdne. Pokud je přítomný předmět s plně nepravidelným rozvrhem, zobrazí se okno, ve kterém může uživatel vybrat, jaký konkrétní týden si chce zobrazit.

Týdny, ve kterých má předmět nepravidelný rozvrh, jsou barevně označeny pro větší přehlednost. Uživatel má také možnost vyhledávat předměty, které se vyučují v prázdných oknech jeho rozvrhu. Po stisknutí prázdného pole v rozvrhu se zobrazí modální dialog vyhledávání s předvyplněným časem začátku a konce zvoleného prázdného intervalu. Student si potom může vybrat nějaký předmět vyučovaný v tomto intervalu, a příjemněji si tak naplánovat rozvrh podle svých potřeb.

Semestr pro rozvrh se mění automaticky podle toho, co je nastavené v aplikaci KOS. Stejně tak se z KOS získávají data o budoucích rozvrzích.

Pokud si uživatel změní své předměty v KOS, ale již má uložený rozvrh ve FELSight, tak to bude aplikací rozpoznáno. Uživateli se zobrazí jeho seznam předmětů tak, jak jsou zapsané v KOS. Jeho vybrané paralelky jsou pak také smazány a rozvrh je restartován do stejné podoby, jako v KOS. Aplikace dále zobrazuje naplněnost paralelek, KOS tuto informaci ale zveřejňuje jen jednou denně.

3.4.2 Místnosti pro samostudium

Tato sekce zobrazuje seznam místností označených jako místnosti pro samostudium. Ty mohou být studenty využity v případě, že jsou prázdné, pro studium - například při čekání na další vyučovací hodinu. Sekce umožňuje místnosti filtrovat podle jejich umístění (Karlovo náměstí nebo Dejvice) a také podle toho, jak dlouho jsou místnosti neobsazené. Poté je možné zde vytvořit událost, kterou ostatní studenti uvidí. Díky tomu budou vědět, že tato místnost je v danou dobu využívána někým jiným. Data o volnosti místností jsou opět získávána z aplikace KOS.

3.5 Rozdělení do mikroslužeb

Jelikož jak sekce “Rozvrhy”, tak sekce “Plánování rozvrhů” pracují s objekty pro rozvrh, budou přesunuty do společné služby. Její název bude Timetable Service a je detailně popsána v sekci 3.9.1. Místnosti pro samostudium budou přesunuty do služby, ve které budou také uchovány záznamy o všech místnostech a jejich detailech. Základ této služby již je vytvořen v rámci práce Adama Kohouta a její název je Room Service[1]. Její popis se nachází v kapitole 3.9.2.

3.5.1 Funkční požadavky

Tato část popisuje, jaké funkční požadavky musí naše služby splňovat, aby mohly vykonávat funkci migrovaných částí aplikace.

3.5.1.1 Funkční požadavky pro funkcionalitu Rozvrhy

- Uživateli se vždy zobrazí jeho rozvrh pro současný týden.
- Uživatel si může přidat do zobrazeného rozvrhu libovolnou místnost, uživatele nebo kurz.
- Uživatel si může zobrazit rozvrh všech přidaných objektů pro libovolné existující datum.
- Aplikace umožní uživateli vygenerovat odkaz na vložení osobního rozvrhu do kalendáře.

3.5.1.2 Funkční požadavky pro funkcionalitu Plánování rozvrhu

- Aplikace zobrazí uživateli předměty zapsané na příští semestr.
- Uživatel si může vybrat specifickou paralelku (pro cvičení, laboratoře a přednášky).
- Uživatel si může přidat libovolný předmět z fakulty.
- Uživatel si může odstranit předmět, který má v rozvrhu.
- Aplikace ukládá uživatelem zvolený rozvrh.
- Uživatel může rozvrh restartovat do původního stavu podle aplikace KOS.
- Aplikace upozorní uživatele na to, že je nějaký předmět nepravidelný.
- Aplikace upozorní uživatele na to, že má nějaký předmět jiný rozvrh v sudém/lichém týdnu.
- Uživatel si v nepravidelném rozvrhu může zobrazit rozvrh pro libovolný týden semestru.
- Uživatel si v rozvrhu se sudým/lichým týdnem může zobrazit jak sudou, tak lichou variantu rozvrhu.
- Aplikace navrátí rozvrh do původního stavu, pokud se změní předměty zapsané v KOS.

3.5.1.3 Funkční požadavky pro Místnosti pro samostudium

- Uživatel si může zobrazit místnosti, které jsou v danou chvíli volné.
- Uživatel může filtrovat místnosti podle umístění.
- Uživatel může filtrovat místnosti podle toho, po jak dlouhou dobu jsou volné.
- Uživatel může v prázdné místnosti vytvořit událost.
- Uživatel si může zobrazit rozvrh libovolného předmětu.
- Uživatel může exportovat svůj rozvrh do kalendářového formátu iCal[18].

3.5.2 Nefunkční požadavky

V této sekci specifikujeme jaké nefunkční požadavky mají naše služby splňovat, z důvodů zajištění plynulého chodu, sledování metrik aplikace a začlenění do již běžících prostředí s jinými aplikacemi.

3.5.2.1 Sdílené nefunkční požadavky

- Aplikace budou vystavovat RESTové rozhraní pro komunikaci s monolitem.
- Aplikace budou vystavovat GraphQL[19] rozhraní.
- Aplikace se bude umět registrovat do Netflix Eureka[20].
- Aplikace poběží jako kontejnery v Dockeru[21].
- Aplikace budou cachovat dotazy, u kterých se nemění podoba dat, v intervalu dvanácti hodin.
- Třídy pro převod z KOSapi a Sirius API budou pokryté unit testy.

3.5.2.2 Nefunkční požadavky Room Service

- Logika pro rezervaci místností bude pokryta integračními testy.

3.5.2.3 Nefunkční požadavky Timetable Service

- Logika pro tvoření rozvrhu na příští semestr bude pokryta integračními testy.

3.6 Podpůrná služba pro zapojení do HUB.FEL

V této sekci se budeme zabývat službou určenou k podpoře integrace migrovaných funkcionalit do fakultní IT platformy HUB.FEL. Tato služba se jmenuje Core Service.

3.6.1 HUB.FEL

HUB.FEL je fakultní IT platforma vytvořená pro sjednocení všech fakultních aplikací vyvíjených Centrem Znalostního Managementu na jedno přehledné místo.

Samotné části aplikace jsou rozděleny do tzv. agend. V současné době platforma obsahuje tyto dvě agendy:

- EProcesy - poskytuje studentům možnost podat digitální požadavek na studijní oddělení.
- Hodnocení zaměstnanců - sloužící ke každoročnímu procesu hodnocení zaměstnanců fakulty.

Uživatelské rozhraní této aplikace je napsané v React.Js. Backendové služby poskytují data frontendu za pomocí rozhraní GraphQL.

3.6.2 Zdůvodnění vytvoření služby

Po vytvoření nových mikroslužeb v rámci této práce bude potřeba ještě pro služby vytvořit uživatelské rozhraní. Pro kompletní funkčnost budeme potřebovat data ze samotných mikroslužeb obohatit o data, která jsou sdílená mezi více službami. V monolitu je toto vyřešeno jednoduše sdíleným přístupem k datům všude, kvůli jednotné databázi. Pro samostatnou funkčnost zmigrovaných funkcionalit v platformě HUB.FEL budeme muset monolit nahradit novou službou, která se bude starat o data potřebná více službami. Tato služba bude v první iteraci poskytovat data o semestrech a kurzech.

3.6.3 Nefunkční požadavky

- Aplikace bude vystavovat GraphQL rozhraní.
- Aplikace poběží jako kontejner v aplikaci Docker.
- Aplikace se bude umět registrovat do Eureky.
- Aplikace bude aktualizovat svá data jednou za den.
- Aplikace bude cachovat dotazy na data, která se mění při aktualizaci (interval jeden den).
- Aplikace pošle mail s upozorněním, pokud selže průběžná denní aktualizace.
- Třídy pro převod z KOSapi budou pokryté unit testy.

3.7 Postupy a technologie mikroslužeb

V posledních letech dochází k migraci vývoje nových funkcionalit FELSightu z monolitu do mikroservisního modelu. Podle Carnella by mikroslužby měly splňovat následující vlastnosti[22]:

- nezávislost
  - Každá služba by měla být nezávislá. Měla by mít svou databázi, kterou by s dalšími službami neměla sdílet.
- jasně definované odpovědnosti
  - Mikroslužby by měly mít dobře definovaný kontext, ve kterém operují. To znamená, že by měly mít jasný účel a zodpovídat za specifickou vlastnost celku.
- nezávisle nasaditelná
  - Každá služba v mikroslužbové aplikaci může být zkompilována a nasazena nezávisle na ostatních službách.

- odolnost
  - Služby by měli být schopné zpracovat selhání a úmyslně se ukončit.
- škálovatelnost
  - Můžeme zvýšit rychlost či množství požadavků, který dokáže naše aplikace zpracovat, přidáním dalších instancí služby.
- sledovatelnost
  - Služby by měly vystavovat metriky a informace, které dovolují průběžné sledování a dohledání a vyřešení problémů.

3.7.1 Spring Boot

Podle statistiky Jetbrains z roku 2021[23] je Spring Boot zdaleka nejpoužívanějším ze všech frameworků pro Javu. Také ho využívá většina služeb v rámci CZM. Je tedy ideální volbou pro základ služeb vytvářených v rámci této práce.

Některé knihovny také poskytují mnohem kódově úspornější řešení než v Java EE. V dalších kapitolách si popíšeme některé z těchto knihoven blíže.

3.7.1.1 Spring Boot Actuator

Actuator[24] nám poskytuje řadu již implementovaných endpointů pro zjišťování a monitorování stavu aplikace. V našich službách využíváme endpoint /health, který jednoduše vrací stav aplikace. Můžeme pak za použití nenáročného volání zjistit, zda aplikace běží. Také používáme endpoint /prometheus, který vrací metriky, se kterými poté může pracovat aplikace pro sběr dat Prometheus a další služby pro vizualizaci těchto dat(například Grafana[25]). Knihovna může být využita v případě potřeby na více věcí, mezi nimi například zobrazení seznamu všech existujících endpointů nebo zaregistrovaných bean.

3.7.1.2 Spring Batch

Spring Batch nám umožňuje rozdělit naše aktualizace do nakonfigurovaných kroků a automaticky ukládá jejich průběh a detaily do databáze. Umožňuje nám tak jednodušší hledání závady v případě selhání, a to díky uložení získané chyby. Kromě toho se ukládá do jakého stavu krok doběhl, kdy začal a jak dlouho trval.

Spring Batch rozděluje kroky konkrétní operace do objektů typu Job a Step. Job v našem případě představuje jeden obecný typ dat, např. “Předměty”. Step je poté dílčí kus Jobu, například paralelky kurzů.

Knihovna nám umožňuje data zpracovávat po menších kusech, zvaných Chunk. Díky tomu mohou aktualizace, které často obsahují tisíce záznamů pro jeden Step, tato data zpracovávat paralelně.

Zpracování samotných dat se poté dá zařídit dvěma způsoby. První způsob je implementace rozhraní Batchlet, které nepodporuje zpracování Chunků, ale celý Step je proveden jako jedna akce, nad jejíž implementací má uživatel absolutní kontrolu. Tento přístup se hodí při vytváření kroků aktualizace, které nejsou datově náročné. Například kontrola toho, zda všechny ostatní aktualizace proběhly úspěšně nebo odeslání mailu v případu selhání. Druhou možností je využití třech rozhraní, které Spring Batch poskytuje pro zpracování Chunků. Těmi jsou rozhraní ItemReader, ItemProcessor a ItemWritter. Rozhraní ItemWriter se volá jako první při zpracovávání Chunku, slouží k přečtení dat ze zdroje a následnému dávkování těchto dat ItemProcessoru. Ten slouží k transformaci dat do námi využitelné podoby a následně data předává ItemWriteru. Ten data zapíše námi specifikovaným způsobem, v našem případě uložením do databáze.

3.7.1.3 Spring Data JPA

Využitím Spring Data JPA[26] můžeme prakticky odstranit všechnu logiku pro získávání dat z databáze, kterou v monolitu musíme manuálně specifikovat. Za pomocí klíčových slov v názvech metody[27] totiž můžeme nechat funkce pro základní operace nad entitami, jako například vyhledávání entit pomocí nějakého jejich atributu, vygenerovat přímo Spring Boot z názvu metody. Jelikož je toto většina operací, které kromě ukládání provádíme, výrazně tak zredukujeme složitost logiky, která se nachází v našich repository třídách.

3.7.2 Databáze

Pro naše služby využijeme dvou databází, jednou je NoSql Databáze MongoDB a druhou je SQL databáze PostgreSQL. Obě využijeme v různých kontextech a v následujících dvou kapitolách si blíže popíšeme jejich specifika a případy užití.

3.7.2.1 MongoDB

MongoDB[28] není relační databází, ale NoSQL. Podle práce Li[29] víme, že právě proto není vhodná pro ukládání provázaných dat, která se nachází ve více tabulkách. Dále nabývá ostatních vlastností, které jsou typické pro NoSql databáze. Využívá flexibilních dokumentů, které se ukládají ve formátu BSON[30] a nemá pevnou podobu dat, která mohou být ukládána. Díky její jednoduchosti nabízí vhodné řešení pro ukládání dat bez potřeby relací. Podle Li[29] jsou obecně NoSql databáze rychlejší než tradiční SQL databáze v případě krátkých dotazů a malého množství dat. Tuto databázi tedy využijeme k ukládání našich vlastních objektů, které se nezískávaji z externích zdrojů a zároveň nepotřebují mít relace na další objekty.

3.7.2.2 PostgreSQL

PostgreSQL je open source objektově relační SQL databáze. Poskytuje nástroje pro kontrolu integrity dat, jako jsou cizí klíče a různá integrační omezení. Dále splňuje standart ACID. ACID je zkratkou pro následující vlastnosti databáze:

- atomicita (atomicity)
  - Každá operace v transakci je považována za jednu jednotku. Buď se provede celá jednotka nebo žádná. Zabraňuje ztrátě dat při výpadku zdroje nebo jiného selhání uprostřed zpracování dat.
- konzistence (consistency)
  - Změny v tabulkách se dějí pouze předem definovanými způsoby. Zabraňuje ztrátě integrity dat.
- izolace (isolation)
  - Zaručuje, že se mezi sebou nebudou narušovat paralelní operace nad jednou tabulkou.
- trvanlivost (durability)
  - Pokud je transakce dokončena úspěšně, jsou její změny zachovány i po selhání systému.

Informace o standartu ACID jsou získány z dokumentace IBM[31]. Díky těmto vlastnostem je PostgreSQL vhodný pro ukládání našich provázaných entit.

3.7.3 Docker

Docker nám umožňuje spustit naše aplikace v kontejneru. To se dá přirovnat k běhu uvnitř virtuálního operačního systému. Tím se zajistí, že aplikace bude běžet všude stejně, a nebude docházet k rozdílům závislých na prostředí, ve kterém je aplikace spuštěna. Při vytváření konfiguračního souboru pro sestavení kontejneru, tzv. Dockerfile, si specifikujeme jaké naše soubory chceme do kontejneru vložit. Dále také jaký operační systém zvolíme jako základ našeho kontejneru a jaké nadstandardní knihovny pro tento systém potřebujeme.

3.7.4 Eureka

Eureka je klient pro objevování instancí služeb běžících na serveru. Slouží k load balancingu a správě stavu služeb. Podle Bourkeho[32] load balancing znamená distribuování požadavků na síti do více instancí jedné služby, pro zajištění hladkého a stabilního provozu. Po úspěšném nastartování služba zavolá Eureku a zaregistruje se do ní jako běžící. Pokud by nějaká komponenta

chtěla tuto službu zavolat bez Eureky, musela by zavolat přímo na její pevnou adresu. To by mohlo působit problémy, například v případě několika instancí této služby. V tomto případě zavolá Eureku s názvem služby, které se chce dovolat. Pokud je tato služba zaregistrovaná, vrátí Eureka pravou adresu této služby. Eureka nám tedy dává možnost load balancingu v našem systému.

3.7.5 REST API

REST API je specifikace přístupu k budování architektury aplikací založené na komunikaci pomocí HTTP protokolu. Využívá čtyr volání, které implementují CRUD matici operací. Tato volání jsou GET, POST, PUT a DELETE. Jako principy přístupu REST ke tvorbě aplikací je udáváno těchto 5 zásad[33]:

- jednotné rozhraní
  - Rozhraní by mělo identifikovat všechna data použitá v interakci mezi klientem a serverem. Data by měla mít jednotné reprezentace jak na serveru, tak u klienta. Data by měla přenášet dost informací pro zpracování požadavku.
- klient-server architektura
  - Zajišťuje dodržení rozdělení zodpovědnosti. Rozhraní má být rozděleno na část zodpovědnou za zdroj dat (server) a na část zodpovědnou za uživatelské rozhraní (klient).
- bezstavovost
  - Každý dotaz klienta na serveru by měl obsahovat všechny informace potřebné k pochopení a uskutečnění požadavku.
- možnost cachování
  - Udává, že by se odpoveď měla identifikovat jako cachovatelná nebo necachovatelná.
- vrstvení
  - Architektura by měla být postavena do hierarchických vrstev, které omezují chování komponent.

3.7.6 GraphQL

GraphQL je dotazovací jazyk, který nám umožňuje specifikovat, jaké objekty a atributy přesně chceme vrátit. Vrácený objekt má tedy přesně ty atributy, o které jsme si požádali při posílání dotazu. Narozdíl od RESTful rozhraní nám umožňuje dělat kruhové závislosti, což se nám hodí při zasílání databázových objektů s oboustrannými relacemi (např. Student a jeho zapsané paralelky a paralelka a její zapsaní studenti). GraphQL nemá sloužit jako nahrazení RESTful rozhraní, ale jako alternativa, nebo mohou fungovat společně vedle

sebe[19]. Rozlišuje pouze dva typy dotazu, kterými jsou "query", který je obdobou GET a "mutation", které fungují jako obecný druh dotazu, měnící konkrétní data (obdoba DELETE, POST, PUT a PATCH).

3.7.7 GraphQL vs REST

Díky studii Eizingera[34] víme, že nám GraphQL při komunikaci s FE v porovnání s tradičním REST rozhraním dává dvě velké výhody. První z nich je zamezení tzv. overfetchingu. To znamená, že se vždy přenášejí i ta data, která v současné době nepotřebujeme. Zvětšuje se tak objem odeslaných dat.

Druhou je vyvarování se potřeby uskutečnit několik dotazů pro získání všech dat, které potřebujeme. To GraphQL řeší například možností využít cyklické závislosti. V RESTu bychom pro získání cyklické závislosti museli uskutečnit další dotaz.

3.7.8 Naše využití obou možností rozhraní

Jelikož aplikace FELSight a její současná podoba řešení uživatelského rozhraní nepodporují GraphQL, definujeme službám, které jsou určené pro komunikaci s monolitem, i RESTové rozhraní. Týká se to tedy Room Service a Timetable Service. Obě tyto služby obsahují i GraphQL rozhraní pro budoucí implementaci do platformy HUB.FEL.

3.7.9 KOSapi

KOSapi poskytuje RESTové rozhraní pro komunikaci s komponentou studia univerzitního systému zvanou KOS. My ho využíváme pouze ke čtení dat, například informace o kurzech, semestrech, paralelkách a katedrách. Data poskytuje ve formátu Atom-XML[35]. S tímto formátem se pracuje složitěji než s formátem JSON. To je z toho důvodu, že bychom museli vytvářet další logiku pro zpracování Atom-XML formátu, zatímco JSON formát používáme při komunikaci s našimi ostatními službami. Využíváme proto vlastní službu, která slouží jako prostředník mezi ostatními službami a KOSapi, a data mapuje do formátu JSON. O naší službě KOSapi Service je blíže psáno v kapitole 4.1.3.

3.7.10 Sirius API

Sirius API nám dává údaje o rozvrhových událostech předmětů, místností a uživatelů. Dále obsahuje semestry a jejich týdny. Nakonec nám umožňuje získat údaje o události ve formátu zpracovatelného kalendáři iCal. Narozdíl od KOSapi data posílá ve formátu JSON rovnou.

3.7.11 CI/CD

CI je zkratka pro průběžnou integraci. CD je zkratkou s dvěma významy, prvním je průběžné nasazování, druhým průběžné doručení. Jedná se o procesy

a praktiky, které ulehčují a zrychlují testování a vyvíjení softwaru. My budeme využívat hlavně nástroj Gitlab Pipelines[36]. Právě v nich můžeme specifikovat různé skripty, které využíváme k uskutečnění níže popsaných praktik.

Průběžná integrace slouží pro sestavení a testování aplikace předem specifikovanými skripty, které se spouští vždy automaticky pro specifikované scénáře (např. po každém provedení Git Push nebo při vytvoření nového merge requestu). Můžeme tak za pomocí testů zachytit nové chyby v aplikaci ještě předtím, než se vůbec dostane do fáze uživatelského testování. Můžeme také například za pomocí nástroje SonarQube[37] vynutit určité pokrytí kódu testy nebo dodržení určité úrovně kvality nového kódu. Pokud vývojářův nový kód nelze sestavit, selže v testech nebo nedodržuje požadovanou kvalitu, jeho Pipeline je označena jako selhaná a uživatel je na to upozorněn.

Průběžné nasazování/doručení je krokem nad rámec průběžné integrace a umožňuje nám automatizovat nasazování naší aplikace do provozu.

Rozdíl mezi průběžným nasazováním a doručením je, že akce, které jsou provedeny při průběžným doručení, jsou spuštěny manuálně příkazem uživatele, zatímco průběžné nasazení je spuštěno automaticky. Příkladem může být nasazení na vývojářské prostředí nebo nahrání artefaktu do Gitlab Repository[38].

3.8 Architektura

Všechny servicy se drží přístupu specification first při vytváření rozhraní. To znamená, že je nejdříve specifikován soubor, ve kterém jsou popsány objekty, se kterými rozhraní pracuje a podoba rozhraní samotného. V případě REST je to .yaml dokument dodržující specifikaci openapi 3.0[39] a u GraphQL je to soubor schema.graphql. Tyto specifikační soubory mají další využití, například se jejich obsah dá využít k vizualizaci rozhraní aplikace za pomocí editoru Swagger[40]. Služby také automaticky za použití knihovny Springfox[41] vystavují endpoint, po jehož navštívení přes prohlížeč je zobrazena specifikace aplikace v uživatelsky přívětivé podobě.

Ovladače rozhraní a objekty jsou potom vygenerovány pomocí openapi generátoru pro Maven[42] a DGS generátoru pro GraphQL[43]. To nám šetří potřebu specifikovat DTO třídy a GraphQL typy manuálně.

Fowler popisuje DTO jako jednoduchý objekt určený pro zmenšení objemu přenesených dat. Místo objektů, na které má vazby, často obsahuje pouze klíče těchto objektů určených k jejich dohledání v případě potřeby[44].

Schopnost GraphQL Federace služeb nám umožňuje dát dohromady několik specifikací různých mikroslužeb do jednoho prostředníka. Díky tomu nemusíme řešit komunikaci mezi službami pro to, aby se na jednom místě mohla dát sloučit všechna data, která potřebuje klient. Náš prostředník vytvoří jedno velké GraphQL schéma z těch poskytlých našemi službami. Klient tak má možnost za pomocí jednoho dotazu získat data ze všech služeb, které potřebuje. Minimalizujeme tak počet volání z uživatelského rozhraní HUB.FEL.

Spring Boot automaticky podporuje vygenerované ovladače pro REST rozhraní. Stejnou funkčnost pro GraphQL dosáhneme využitím knihovny

Netflix DGS[45].

Byznys logika je rozdělena do vrstev tak, aby byl dodržen princip REST. Těmi jsou:

- Doména, která spravuje a uchovává data, tou je v našem případě PostgreSQL nebo Mongo databáze.
- Servisní vrstva, ve které se nachází byznysová logika pro manipulaci s daty z domény.
- Vrstva ovladačů, ve které se nachází třídy pro ovládání vystaveného rozhraní.

Room Service a Core Service využívají PostgreSQL databázi, do které ukládají data z KOSApi pro rychlejší dohledání. Tato data jsou aktualizována jednou denně a v případě selhání aktualizace se odešle mail uživatelům, kteří jsou nastaveni v databázi jako administrátoři. V současné implementaci je zpracování dat z KOSapi zařizováno knihovnou Java Batch API[46]. V nové službě pro tuto funkcionalitu využijeme knihovny Spring Batch díky její dostupnosti ve Spring Boot a také snížené velikosti kódu nutného pro implementaci stejné funkcionality.

Po provedení Git Push jsou v každé service automaticky spuštěny CI/CD skripty (blíže popsáno v sekci 3.7.11), které jsou specifikované v souboru .gitlab-ci.yml[47]. V případě selhání jakéhokoliv kroku dojde k přerušení vykonávání Pipeline a žádný další krok již není proveden. Soubor .gitlab-ci.yml v našem případě obsahuje specifikace, jak vypadají kroky pro sestavení a nasazení aplikace na server.

Soubory .gitlab-ci.yml nejsou úplně totožné pro obě služby, ale obsahují ty stejné kroky, pouze s drobnými změnami:

1. build
   - Tento krok zkompiluje kód a vytvoří z něj spustitelný .jar soubor.
2. test
   - Spustí testy v projektu, aby zajistil, že při vývoji nedošlo k rozbití již existující logiky.
3. image build
   - Sestaví Docker Image z .jar souboru a nahraje ho do Gitlab Repository.
4. client build
   - Vygeneruje ze specifikačního souboru artefakt klienta. Tento artefakt budou následně používat ostatní služby pro komunikaci s touto službou. Může se jednat o REST klienta či o GraphQL klienta.

3.9 Konkrétní služby

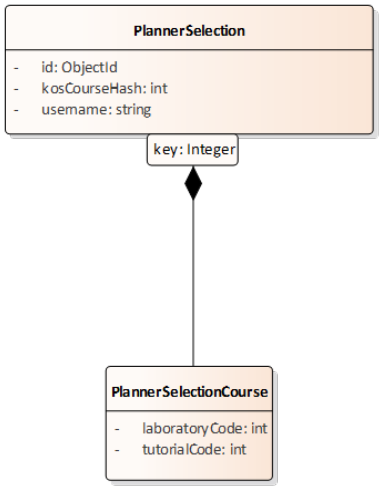
V této sekci konkrétně popíšeme jednotlivé služby.

3.9.1 Timetable Service

Timetable Service je službou sloužící pro manipulaci s rozvrhovými objekty. Jejím zdrojem dat jsou externí aplikace Sirius API a KOSapi. Ze Sirius API aplikace čerpá aktuální data o rozvrzích osob, místností a předmětů. Tato data se cachují dvanáct hodin, protože se v průběhu dne nemění. Jinak se tato data nijak neukládají do databáze služby z důvodu poskytnutí co nejaktuálnějších dat.

Dále služba získává data z KOSapi. Z KOSapi bere data o uživatelem zapsaných předmětech v KOS, které porovnává s těmi, co má uživatel v databázi. Pokud zjistí, že předměty jsou odlišné, uživatelův rozvrh je obnoven do podoby podle KOS. Detailní popis tohoto procesu je zobrazen v sekvenčním diagramu 3.2. Dále se z KOSapi získávají data o tom, jaký semestr je v současné době v KOS označený jako semestr pro rozvrh. Pro tento semestr jsou pak získávána data o tom, jak budou vypadat paralelky uživatelem zvolených předmětů. Uživatelem zvolené předměty a paralelky se ukládájí do MongoDB databáze ve formě klíč-hodnota, kde klíčem je kód předmětu a hodnotou pár zvolená laboratoř a zvolená paralelka. Tato entita je zobrazena na diagramu 3.1. MongoDB bylo pro tento případ zvoleno kvůli druhu ukládaných dat. Jedná se o malé krátké mapy, které nemají žádné relace na jiné tabulky - není proto zapotřebí využít velkou relační PostgreSQL databázi tak, jako na monolitu.

Služba poskytuje rozhraní pro získávání a ukládání výše zmíněných uživatelem navolených rozvrhů. Dále poskytuje rozhraní pro získání současného rozvrhu místností, lidí a kurzů. K tomu ještě obsahuje nový endpoint, který přijímá seznam klíčů objektů, kterým má vrátit rozvrh (místnosti, rozvrhy a uživatelé). Těmto objektům získá rozvrh a poté všechna data pošle v jednom objektu nazpět. Nakonec také poskytuje endpoint pro získávání rozvrhu předmětů na příští semestr. Ukázka tohoto rozhrání se nachází v přílohách D.1, D.2 a D.3.



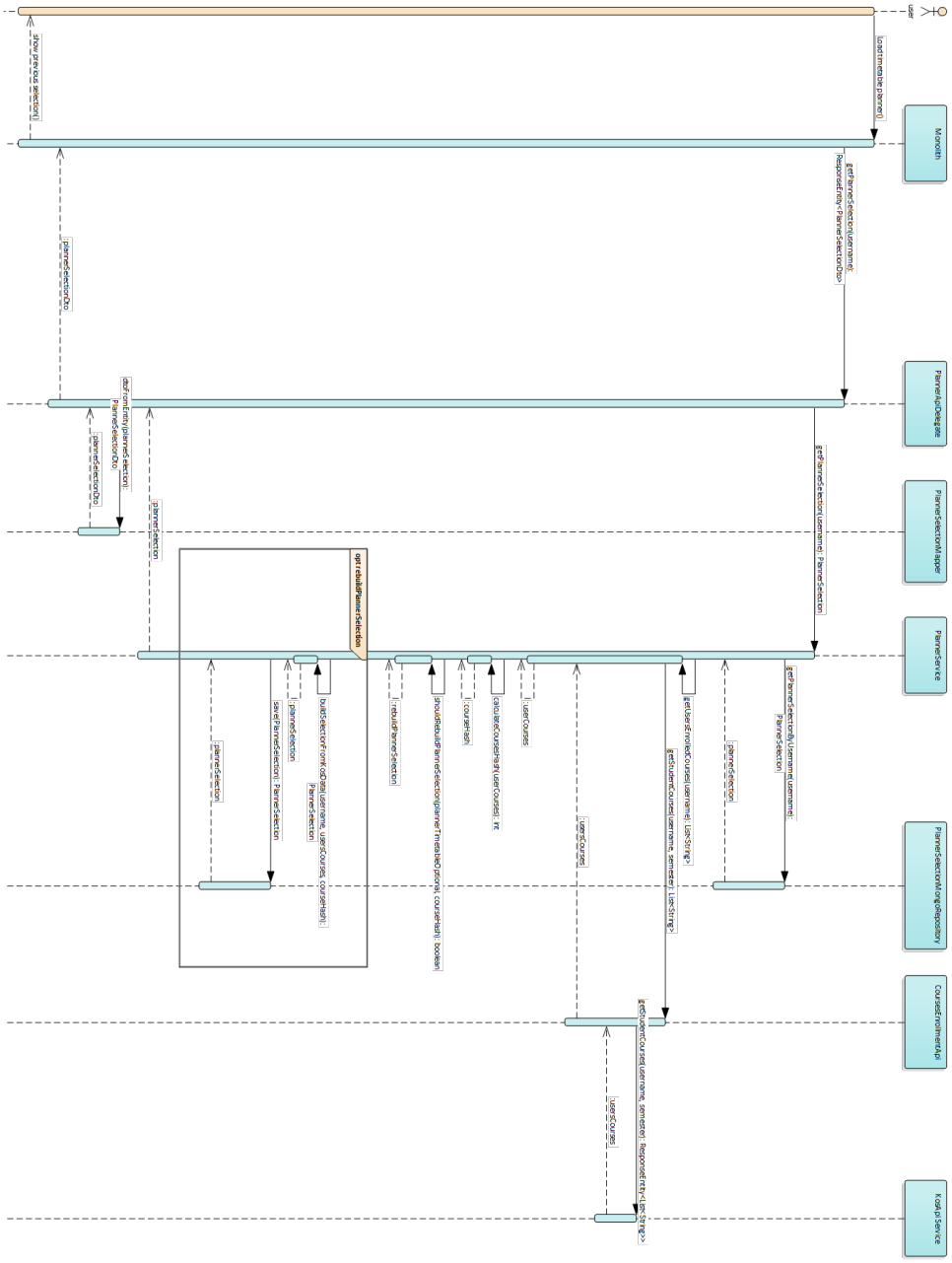
Obrázek 3.1: UML diagram tříd Timetable Service.

Sekvenční diagram 3.2 popisuje volání z monolitu pro získání uživatelova navoleného rozvrhu. Monolit nejdříve zavolá ovladač pro Plánování rozvrhu. Ten slouží jako ovladač rozhraní pro získávání rozvrhu. Z ovladače se dále zavolá servisní vrstva, ve které se volá databáze. Z databáze se vrací Optional objekt, který využijeme později při kontrole, jestli je potřeba vytvořit nový rozvrh.

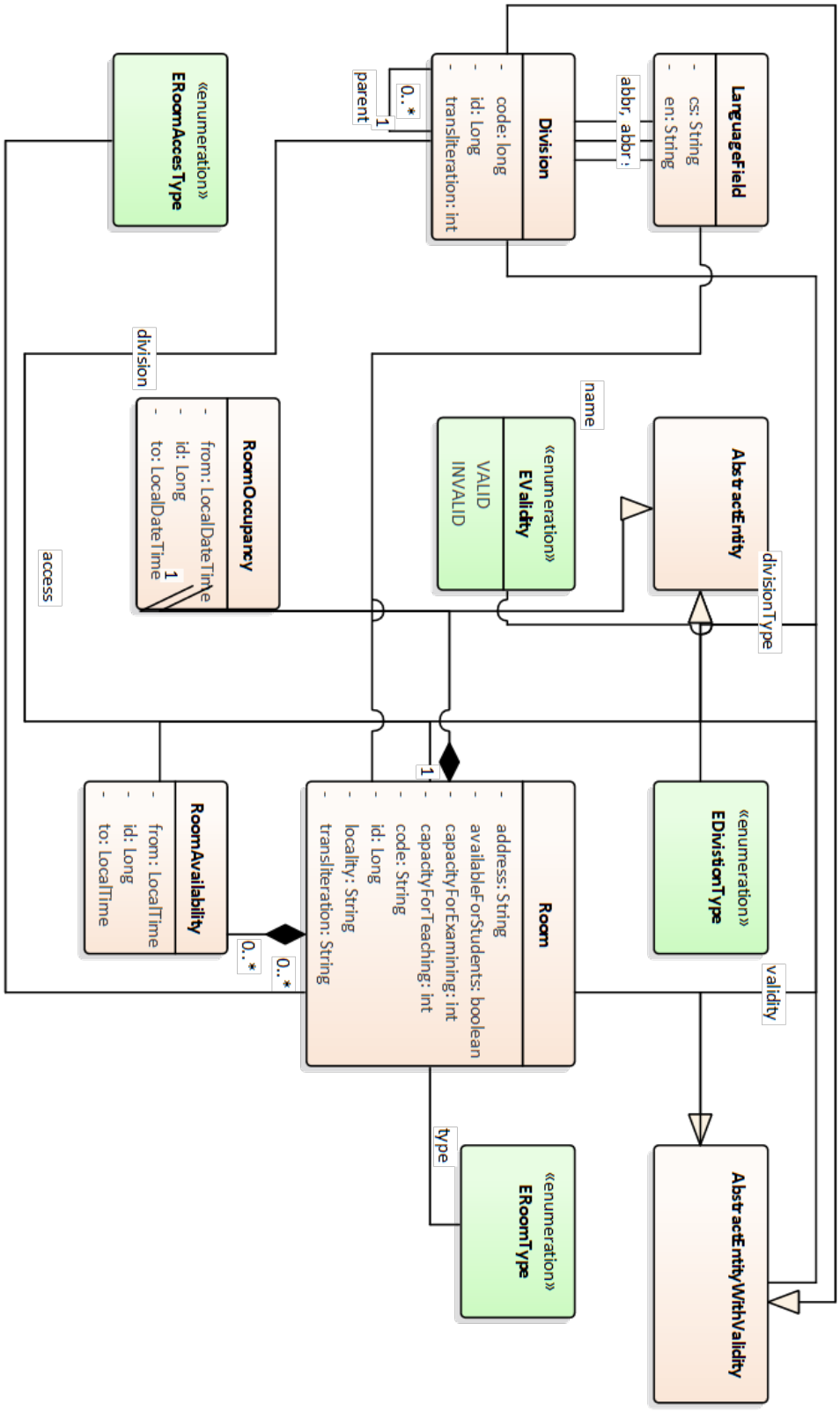
Po dotazu do databáze se volá klient pro KOSapi Service, aby se získaly studentovy zapsané kurzy v KOS. Následně probíhá kontrola, jestli se má vytvořit nový rozvrh. Ta probíhá na základě dvou parametrů. Prvním je to, zda-li vůbec existuje studentův rozvrh v databázi. Pokud ano, zkontroluje se CourseHash atribut databázového rozvrhu oproti kurzům, které byly získány z KOS. Pokud jsou totožné, je uživateli vrácen rozvrh z databáze. Pokud se kurzy změnily, nebo rozvrh vůbec neexistuje, je z kurzů z KOS vytvořen nový rozvrh, který je poslán do ovladače. V ovladači proběhne konverze z databázové entity na DTO objekt (toto probíhá v ovladači kvůli tomu, že máme jak REST ovladače, tak GraphQL ovladače, které vracejí jiné třídy specifické pro GraphQL). DTO je zasláno zpět do monolitu a rozvrh je zobrazen uživateli.

3.9.2 Room Service

Room Service vystavuje endpoint pro získání všech místností s možností vyfiltrování pouze místností pro samostudium. Také vystavuje endpoint pro získání jedné specifické místnosti. Místnosti, které jsou označené jako pro samostudium, si v databázi uchovávají informace o tom, od kdy do kdy jsou volné a kdy se v nich konají jiné události. Tato informace se potom posílá i do uživatelského rozhraní pro zobrazení v tabulce. Ukázku rozhrání Room Service můžeme vidět v příloze F.1 a na obrázku 3.3 je zobrazen diagram entit.



Obrázek 3.2: Sekvenčí diagram dotazu na získání rozvrhu pro sekci Plánování rozvrhu.



Obrázek 3.3: UML diagram tříd Room Service.

3.9.3 Core Service

Core Service je vytvořena jako jádro pro funkcionality FELSight v rámci fakultní platformy HUB.FEL, nemusí tedy vystavovat REST rozhraní, jako je tomu u služeb předchozích.

V současné podobě slouží hlavně jako POC pro pozdější zapojení do fakultní platformy. Obsahuje tedy zatím pouze databázi s kurzy a endpointy, pro vrácení jednoho specifického předmětu nebo všech předmětů. Ukázku jeho rozhraní můžeme najít v přiloze E.1.

3.9.4 Vyhodnocení nové architektury

V tabulce 3.1 popisujeme přínosy a nevýhody nové architektury oproti původnímu řešení.

	Monolit	**Mikroslužby**
Škálovatelnost	Nelze vytvářet další instance.	Dají se vytvořit další instance služeb.
Množství kódu	Standardní.	Nižší díky využití modernějších knihoven.
Složitost celku	Standardní architektura celku.	Kvůli rozdělení do služeb a přítomnosti podpůrných služeb může být celek všech služeb složitější na úvodní seznámení.
Složitost jednotlivých částí	Stejná jako u celku, kvůli tomu že se jedná o monolit	Jednoduchá, díky jasně definované odpovědnosti služeb.
Změna uživatelského rozhraní	Kvůli starší technologii je nemožné využít moderní řešení uživatelského rozhraní.	Mikroslužby dovolují jednodušší změnu díky tomu že nejsou závislé na tom, komu data poskytují.

Tabulka 3.1: Porovnání vlastností mikroslužbového řešení a původního monolitu.

Kapitola 4

Zapojení do provozu

V této kapitole přiblížíme průběh připojení vytvořených služeb k monolitu a jejich nasazení do provozu na serveru. Kapitola dále obsahuje popis testů, které zjednoduší dlouhodobou spolehlivost služeb.

4.1 Zapojení k monolitu

Aby monolit mohl používat třídy a rozhraní našich služeb, bude potřebovat jejich klienta. Toho vytvoříme způsobem, který popisuje Adam Kohout ve své práci[1], jen místo nástroje Gradle využijeme Maven.

1. Za pomoci Maven[48] kroku build client vytvoříme klienta a nahrajeme ho do Gitlab Repository. Tohoto klienta přidáme jako Maven Dependency do projektu monolitu. Ukázku Gitlab Repository, které obsahuje klienta najdeme v příloze H.1.

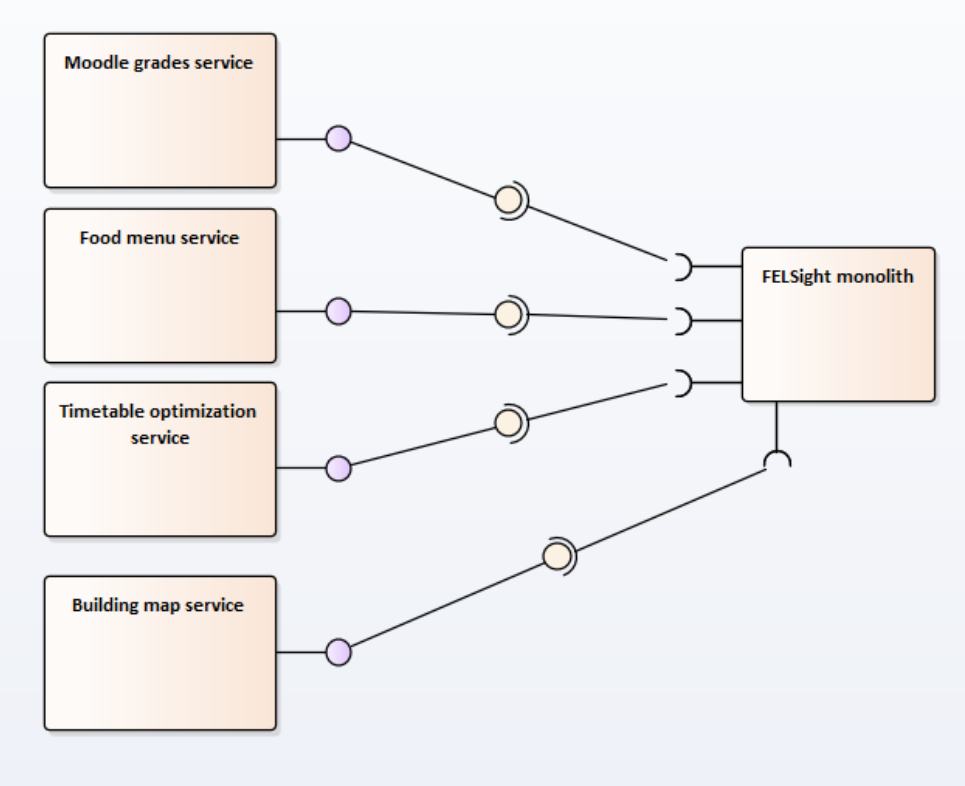
2. Doděláme logiku potřebnou pro komunikaci se službou (posílání tokenu, adresa služby).

3. Implementujeme mapování DTO objektů do objektů, které potřebuje monolit.

4. Nahradíme třídu, která v současné době komunikuje s databází a dodává z ní data, klientem. Ten bude komunikovat s novou službou. V příloze H.2 můžeme vidět jak takový klient vypadá.

4.1.1 Současná podoba serveru

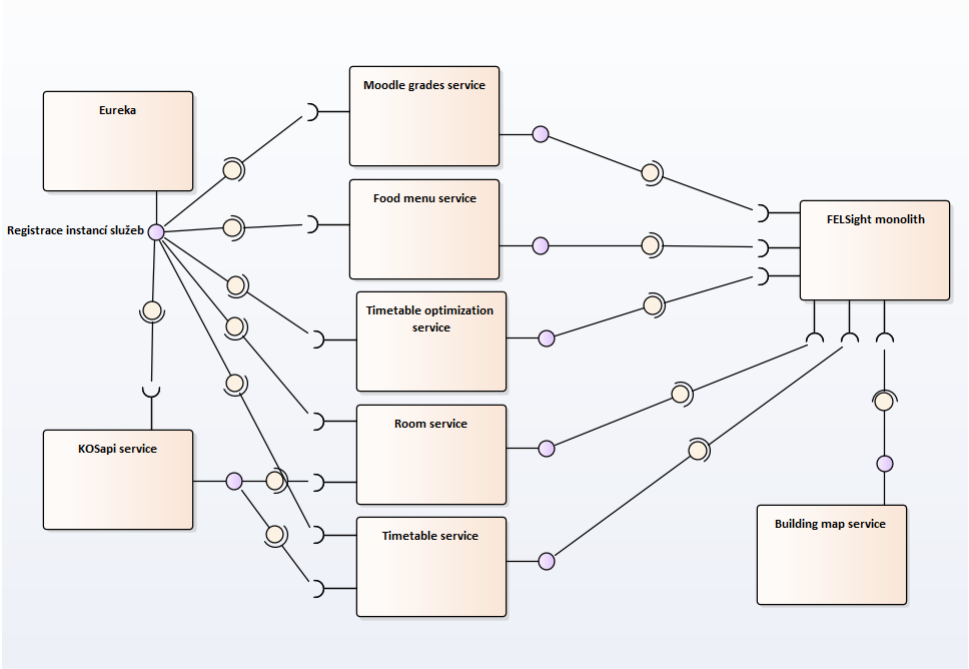
V současné době jsou předprodukční a produkční prostředí stejná, co se týče jejich architektury. Na každém z nich běží aplikační server Payara, do kterého je nasazován artefakt FELSight. Dále vedle nich běží Spring Boot služby pro jídelníčky, známky moodle a Node.js mapa ČVUT FEL. Na ilustračním diagramu 4.1 můžeme vidět současný stav.



Obrázek 4.1: Diagram služeb běžících na serveru při současné implementaci.

4.1.2 Budoucí podoba

Společně s migrací funkcionalit z monolitu do mikroslužeb byla také změněna struktura vývojového prostředí, na kterém tyto nové služby běží. Na prostředí byla přidána Eureka, jejíž účel je popsán v kapitole 3.7.4. Dále se zde nachází instance aplikace Prometheus, která slouží k monitoringu našich aplikací za pomoci vystavených endpointů, popsaných v kapitole 3.7.1.1, ze kterých sbírá data. Ilustrační diagram serveru s novými službami, můžeme vidět na obrázku 4.2.



Obrázek 4.2: Diagram služeb běžících v novém stavu.

4.1.3 KOSapi Service

Tato služba slouží k obalení volání do KOSapi a jejich převedení na jednotný JSON formát, který mohou ostatní služby využívat bez potřeby duplikace kódu pro překládání z XML formátu. Zatímco monolit využíval komunikace přímo s KOSapi, naše nové služby s ním komunikují již přes KOSapi Service.

4.2 Testy

V této sekci budeme probírat integrační a jednotkové testy našich služeb. Jaké pokrývají části a u integračních testů blíže popíšeme jejich scénáře.

4.2.1 Unit testy

Beizer[49] popisuje unit testing jako postup, používaný k otestování funkčnosti jednotlivých jednotek kódu. V našem kontextu objektově orientovaného programování je touto jednotkou metoda. Unit testy chceme pokrýt naší servisní vrstvu, abychom zaručili její současnou i budoucí funkčnost.

4.2.1.1 Pokrytí

Jak bylo zmíněno v kapitole 3.5.2, unit testy jsou pokryty všechny třídy, které zajištují mapování. A to jak mapování z externích zdrojů do databázových entit, tak mapování z entit do DTO objektů, které posílá rozhraní.

Dále jsou unit testy pokryty metody v třídě PlannerService z důvodu zajištění funkčnosti jedné ze stěžejních funkcionalit aplikace.

4.2.2 Integrační testy

Podle Beizera[49] je integračním testováním to testování, ve kterém jsou jednotlivé komponenty dány dohromady a testovány jako celek za účelem odhalení chyb v jejich vzájemné interakci. V této práci využijeme integrační testy pro kontrolu funkčnosti složitějších funkcionalit.

Pro integrační testy si v obou službách vytvoříme novou speciální vnořenou databázi (narozdíl od externí databáze, ke které se připojujeme po síti v normálním provozu). Pro MongoDB využijeme knihovnu Flapdoodle Embed Mongo[50], která je určená k využití vnořené MongoDB, které normálně Spring Boot neposkytuje. Při testování funkcionalit využívajících PostreSQL namísto toho využijeme databázi H2[51]. Ta již základně běží vnořená v aplikaci a pro naše účely postačí k simulaci komunikace s naší běžnou databází. Ukázku nastavení třídy pro integrační testy a scénáře najdeme v přílohách G.1 a G.2.

4.2.2.1 Scénáře pro funkcionalitu Plánování rozvrhu

V této sekci za pomocí tabulek 4.1 a 4.2 popíšeme scénáře integračních testů pro Plánování rozvrhu.

Cíl scénáře	Funkčnost generování nového rozvrhu a ukládání změn.
Popis scénáře	Uživateli se vrátí nový rozvrh podle předmětů z KOS. Uživatel přidá do jeho PlannerSelection nový kurz a zvolí pro něj paralelky. PlannerSelection se uloží do databáze. Otestuje se, že proběhly změny které chceme a žádné jiné.
Očekávaný výsledek	Uživateli se nejdříve vrátí rozvrh bez zvolených paralelek, poté se uloží rozvrh se zvolenými paralelkami a přidaným předmětem.

Tabulka 4.1: První testovací scénář Timetable Service.

4.2.2.2 Scénáře pro funkcionalitu Místnosti pro samostudium

V této sekci popíšeme za pomocí tabulky 4.3 scénář pro testování Místností pro samostudium.

Cíl scénáře	Funkčnost odebírání předmětů a obnovení rozvrhu do původní podoby.
Popis scénáře	Uživateli se vrátí nový rozvrh podle předmětů z KOS. Z PlannerSelection je odebrán jeden předmět. Rozvrh je následně uložen do databáze. Otestuje se že proběhly změny, které chceme a žádné jiné. Rozvrh je obnoven do původní podoby. Je otestováno, že je rozvrh v původní podobě.
Očekávaný výsledek	Uživateli se z databáze smažou odebrané předměty. Uživateli se vrátí rozvrh do základního stavu.

Tabulka 4.2: Druhý testovací scénář Timetable Service.

Cíl scénáře	Funkčnost ukládání a zobrazování rezervací místností.
Popis scénáře	V místnosti se nachází rezervace, která je v jiné datum než uživatelovo zvolené. Otestuje se že v uživatelův zvolený čas je místnost prázdná. Poté se uloží uživatelova rezervace a otestuje se že nyní již místnost v uživatelův čas prázdná není.
Očekávaný výsledek	Uživatel si zobrazí události v místnosti, místnost je v zvolenou dobu volná a on si do ní vytvoří rezervaci. Poté se zobrazí rezervace.

Tabulka 4.3: První testovací scénář Room Service.

4.3 Zapojení do fakultní platformy HUB.FEL

Kvůli vzniku této platformy, jakožto jednotného uživatelského rozhraní pro fakultní aplikace, je potřeba připravit i námi migrované funkcionality na jejich integraci. Pro uskutečnění jsme vytvořili službu Core Service, která je popsána v kapitolách 3.6 a 3.9.3.

4.3.1 Architektura HUB.FEL

HUB.FEL je sdružením několika logicky oddělených částí. Každá z agend zmíněných v kapitole 3.6.1 má na sobě nezávislé služby, které sjednocují centrální část aplikace.

Nachází se zde služba pro správu notifikací, která ostatním službám udává jednotný formát objektu notifikace, které mají posílat. Dále je přítomná služba starající se o uživatelská práva napříč platformou. Té jednotlivé agendy poskytují data o tom, jaká má uživatel práva. Také zde běží Eureka server, který spravuje instance všech služeb zapojených do platformy. Služby poskytují své rozhraní frontendu výhradně přes GraphQL, kde má každá agenda svou federační službu, která zajišťuje sjednocení schémat všech mikroslužeb jedné agendy a přísun dat pro uživatelské rozhraní.

Za účelem zapojení funkcionalit FELSight, jakožto nové agendy, stačí

spustit naše tři nové služby a vytvořit jim federační službu. S podporou již existujících centrálních aplikací HUB.FEL bude možné ihned po vytvoření nového uživatelského rozhraní uvést tyto funkcionality do provozu.

Kapitola 5

Závěr

V rámci této práce jsme představili jaké funkcionality poskytuje studijní portál FELSight. Popsali jsme architekturu jeho stávajícího řešení a jaké problémy tento přístup přináší. Vybrali jsme tři funkcionality a vytvořili jejich funkční požadavky, které zajistí stejnou funkčnost. Vytvořili jsme dvě služby, které tyto požadavky splňují. Těmi jsou Room Service a Timetable Service. K tomu jsme ještě vytvořili službu Core Service, která funguje jako podpůrná pro budoucí možnost zakomponování zmigrovaných funkcionalit FELSight do platformy HUB.FEL. Odkazy na GitLab repozitáře těchto služeb jsou k dohledání v příloze C. Přiblížili jsme architekturu a použité technologie těchto tří nově vzniklých služeb a porovnali výhody a nevýhody různých možností řešení. Popsali jsme jak bude vypadat zapojení Room Service a Timetable Service do stávajícího prostředí monolitu. Vysvětlili jsme jaké části budou pokryté unit a integračními testy. Ukázali jsme scénáře integračních testů a jaké funkcionality mají pokrývat. Poté jsme popsali co je to platforma HUB.FEL, jak vypadá její současné prostředí a jak do ní zakomponovat Room Service, Timetable Service a Core Service.

5.1 Budoucí rozvoj

Tato práce nabízí dvě možnosti budoucího rozvoje.

Tou první je navázání ve směru dalšího rozdělování monolitu na služby. Ve FELSight jsou ještě zbylé funkcionality jako notifikace, uživatelské události a skupiny, které mají možnost být přesunuty do samostatných služeb a znovu začleněny do monolitu.

Druhou možností je vývoj uživatelského rozhraní pro platformu HUB.FEL, které je hlavním chybějící článkem zapojení služeb vytvořených v rámci této práce. Samozřejmě i další služby vzniklé v rámci rozdělení FELSight mají možnost být zakomponovány do platformy HUB.FEL, poté co vznikne jejich uživatelské rozhraní. Těchto služeb je méně, z toho důvodu, že platforma už sama o sobě některé tyto funckionality obsahuje (např. notifikace a události) a šlo by spíše o úpravu či rozšíření stávajících služeb HUB.FEL.

Příloha A

Bibliografie

[1] A. Kohout, "The Architecture Transformation of FelSight Faculty Application", angl. URL: https://dspace.cvut.cz/bitstream/handle/10467/101696/F3-DP-2022-Kohout-Adam-The_Architecture_Transformation_of_FelSight_Faculty_Application.pdf?sequence=-1&isAllowed=y.

[2] CZM ČVUT. "Moodle FEL ČVUT". (2023), URL: https://moodle.fel.cvut.cz/ (cit. 07. 04. 2023).

[3] Moodle Pty Ltd. "Moodle.org". (2023), URL: https://moodle.org/ (cit. 07. 04. 2023).

[4] Oracle. "Java EE". (2023), URL: https://www.oracle.com/java/technologies/java-ee-glance.html (cit. 28. 04. 2023).

[5] ITS North Carolina. "Monolithic". (2005), URL: https://web.archive.org/web/20070902151937/http://www.its.state.nc.us/Information/Glossary/Glossm.asp (cit. 28. 03. 2023).

[6] The PostgreSQL Global Development Group. "PostgreSQL". (2023), URL: https://www.postgresql.org/about/ (cit. 06. 01. 2023).

[7] Payara Services Ltd. "Payara Glasssfish". (2023), URL: https://www.payara.fish/ (cit. 28. 03. 2023).

[8] Eclipse Foundation. "Jakarta® EE". (2023), URL: https://jakarta.ee/ (cit. 28. 03. 2023).

[9] Oracle. "JavaServer Faces". (2023), URL: https://www.oracle.com/java/technologies/javaserverfaces.html (cit. 28. 03. 2023).

[10] J. Jirůtka. "KOSapi dokumentace". čeština. (2023), URL: https://kosapi.fit.cvut.cz/projects/kosapi/wiki/Resources (cit. 19. 11. 2022).

[11] J. Jirůtka. "Sirius API dokumentace". čeština. (2023), URL: http://cvut.github.io/sirius/docs/api-v1.html (cit. 19. 11. 2022).

[12] Gradle Inc. "Gradle Build Tool". (2023), URL: https://gradle.org/ (cit. 13. 01. 2023).

[13] VMware, Inc. "Spring Boot". angl. (2023), URL: https://spring.io/projects/spring-boot (cit. 19. 11. 2022).

[14] Meta Platforms, Inc. "React". (2023), URL: https://react.dev/ (cit. 28. 03. 2023).

[15] Google LLC. "Google Calendar". (2023), URL: https://calendar.google.com/ (cit. 16. 01. 2023).

[16] Microsoft. "Microsoft Outlook". (2023), URL: https://outlook.live.com/ (cit. 21. 03. 2023).

[17] VIC ČVUT. "KOS". (2023), URL: https://www.kos.cvut.cz/ (cit. 03. 01. 2023).

[18] Z Content. "iCalendar.org". (2023), URL: https://icalendar.org/ (cit. 15. 04. 2023).

[19] The GraphQL Foundation. "GraphQL". (2023), URL: https://graphql.org/ (cit. 20. 03. 2023).

[20] Netflix, Inc. "Netflix Eureka". (2023), URL: https://github.com/Netflix/eureka (cit. 26. 03. 2023).

[21] Docker Inc. "Docker". (2023), URL: https://www.docker.com/ (cit. 26. 03. 2023).

[22] J. Carnell a K. Patel, *Spring Microservices in Action*. Simon a Schuster, 11. čvn. 2017.

[23] JetBrains s.r.o. "The State of Developer Ecosystem in 2021 Infographic". (2021), URL: https://www.jetbrains.com/lp/devecosystem-2021/java/#Java_what-web-frameworks-do-you-use-if-any (cit. 21. 04. 2023).

[24] VMware, Inc. "Actuator". (2023), URL: https://docs.spring.io/spring-boot/docs/current/reference/html/actuator.html (cit. 23. 03. 2023).

[25] Grafana Labs. "Grafana". (2023), URL: https://grafana.com/ (cit. 28. 03. 2023).

[26] VMware, Inc. "Spring Data JPA". (2023), URL: https://docs.spring.io/spring-data/jpa/docs/current/reference/html/ (cit. 28. 03. 2023).

[27] VMware, Inc. "Spring Data JPA keywords". (2023), URL: https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#repository-query-keywords (cit. 28. 03. 2023).

[28] MongoDB, Inc. "MongoDB". (2023), URL: https://www.mongodb.com/ (cit. 06. 01. 2023).

[29] Y. I. Li a S. M. Manoharan. "A performance comparison of SQL and NoSQL databases". (1. srp. 2013), URL: https://ieeexplore.ieee.org/abstract/document/6625441?casa_token=DEVmfy9l6dcAAAAA:xzXFY2h1G8NRBI5R3JMjKRB2ND8HOsrsArZTK61BB73jePr9878mnvkDLo7ZuYLLGPtY-ZREUQ.

[30] MongoDB, Inc. “BSON”. (2023), URL: https://www.mongodb.com/basics/bson#:~:text=BSON%20is%20a%20binary%20encoded,it%20supports%20fewer%20data%20types (cit. 28. 04. 2023).

[31] IBM Corporation. “IBM Documentation”. (2023), URL: https://www.ibm.com/docs/en/cics-ts/5.4?topic=processing-acid-properties-transactions (cit. 28. 03. 2023).

[32] T. Bourke, *Server Load Balancing*. "O'Reilly Media, Inc.", 1. led. 2001.

[33] L. Gupta. “What is REST”. (2023), URL: https://restfulapi.net/ (cit. 28. 04. 2023).

[34] T. Eizinger, “API design in distributed systems: a comparison between GraphQL and REST”, *University of Applied Sciences Technikum Wien-Degree Program Software Engineering*, 2017.

[35] W3C qa-dev group. “AtomXML”. (2023), URL: https://validator.w3.org/feed/docs/atom.html (cit. 20. 04. 2023).

[36] GitLab B.V. “CI/CD pipelines | GitLab”. (2023), URL: https://docs.gitlab.com/ee/ci/pipelines/ (cit. 28. 03. 2023).

[37] SonarSource S.A. “SonarQube”. (2023), URL: https://www.sonarsource.com/products/sonarqube/ (cit. 28. 03. 2023).

[38] GitLab B.V. “Repository | GitLab”. (2023), URL: https://docs.gitlab.com/ee/user/project/repository/ (cit. 28. 03. 2023).

[39] OpenApi Initiative. “OpenAPI Specification v3.1.0 | Introduction, Definitions, More”. (2023), URL: https://spec.openapis.org/oas/v3.1.0 (cit. 28. 04. 2023).

[40] SmartBear Software. “Swagger”. (2023), URL: https://swagger.io/ (cit. 28. 04. 2023).

[41] M. Pitt, D. Krishnan a A. Kelly. “SpringFox”. (2023), URL: https://springfox.github.io/springfox/ (cit. 28. 04. 2023).

[42] OpenAPITools.org. “openapi generator”. (2023), URL: https://github.com/OpenAPITools/openapi-generator/tree/master/modules/openapi-generator-maven-plugin (cit. 28. 04. 2023).

[43] Netflix, Inc. “Code Generation - DGS Framework”. (2023), URL: https://netflix.github.io/dgs/generating-code-from-schema/ (cit. 28. 04. 2023).

[44] M. G. Fowler, *Patterns of Enterprise Application Architecture*. 5. lis. 2002.

[45] Netflix, Inc. “DGS Framework”. (19. dub. 2023), URL: https://netflix.github.io/dgs/ (cit. 28. 04. 2023).

[46] Oracle. “Introduction to Batch Processing”. (2023), URL: https://docs.oracle.com/javaee/7/tutorial/batch-processing001.htm (cit. 17. 04. 2023).

[47] GitLab B.V. "The '.gitlab-ci.yml' file". (2023), URL: https://docs.gitlab.com/ee/ci/yaml/gitlab_ci_yaml.html (cit. 23. 04. 2023).

[48] The Apache Software Foundation. "Apache Maven". (2023), URL: https://maven.apache.org/ (cit. 16. 01. 2023).

[49] B. Beizer, *Software Testing Techniques*. Auerbach Publications, 30. lis. 1998. DOI: 10.1201/9781420048131.axh.

[50] Flapdoodle OSS. "Flapdoodle Embed Mongo". (2023), URL: https://github.com/flapdoodle-oss/de.flapdoodle.embed.mongo (cit. 22. 04. 2023).

[51] H2. "H2 Database Engine". (2023), URL: https://www.h2database.com/html/main.html (cit. 28. 04. 2023).

Příloha B

Terminologie

CD Průběžné doručení

CI Průběžná integrace

CZM Centrum Znalostního Managementu

DTO Data Transfer Object

FEL.HUB Fakultní platforma pro sjednocení aplikací CZM

FELSight Portál s funkcemi mířenými na studenty

JSF JavaServer Faces

POC Proof of concept

REST Representational State Transfer

JAVA EE Java Enterprise Edition

Příloha C

Git repozitáře projektů na fakultním GitLab

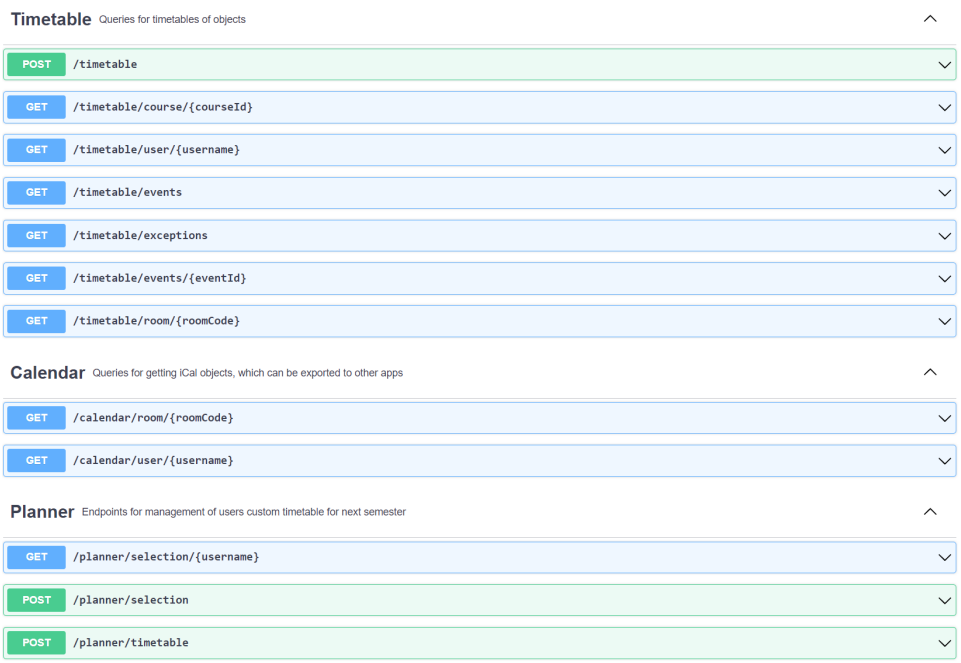
- Repozitář Timetable Service: `https://gitlab.fel.cvut.cz/projects/24704`
- Repozitář Room Service: `https://gitlab.fel.cvut.cz/projects/25953`
- Repozitář Core Service: `https://gitlab.fel.cvut.cz/projects/26251`
- Repozitář FELSight: `https://gitlab.fel.cvut.cz/czm/felsight-group/felsight/-/tree/feature_timetable_service_integration`

Příloha D

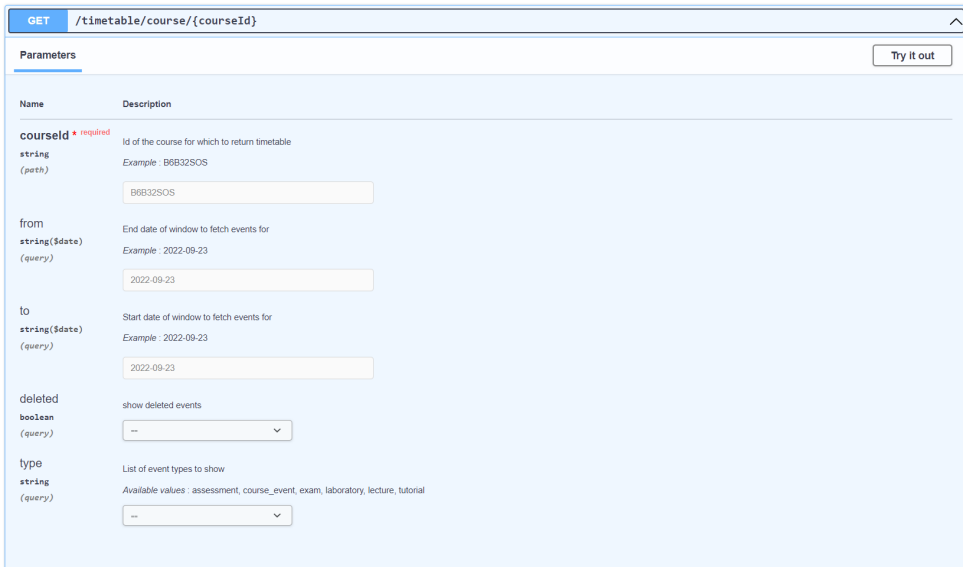
Ukázka rozhraní Timetable Service

```yaml
/timetable/course/{courseId}:
  get:
    tags:
      - Timetable
    operationId: getCourseTimetable
    parameters:
      - in: path
        name: courseId
        schema:
          type: string
        required: true
        example: B6B32SOS
        description: Id of the course for which to return timetable
      - $ref: '#/components/parameters/fromDateParam'
      - $ref: '#/components/parameters/toDateParam'
      - $ref: '#/components/parameters/deletedEventsParam'
      - $ref: '#/components/parameters/eventTypeParam'
    responses:
      '200':
        description: OK
        content:
          application/json:
            schema:
              type: array
              items:
                $ref: '#/components/schemas/TimetableSlot'
      '400':
        description: Bad request.
      '401':
        description: Authorization information is missing or invalid.
      '404':
        description: Course could not be found.
      '500':
        description: internal error.
```

Obrázek D.1: Ukázka textové verze schématu rozhraní Timetable Service.



Obrázek D.2: Pohled na grafickou verzi rozhraní Timetable Service.



Obrázek D.3: Ukázka detailu požadavku v grafickém rozhraní.

Příloha E

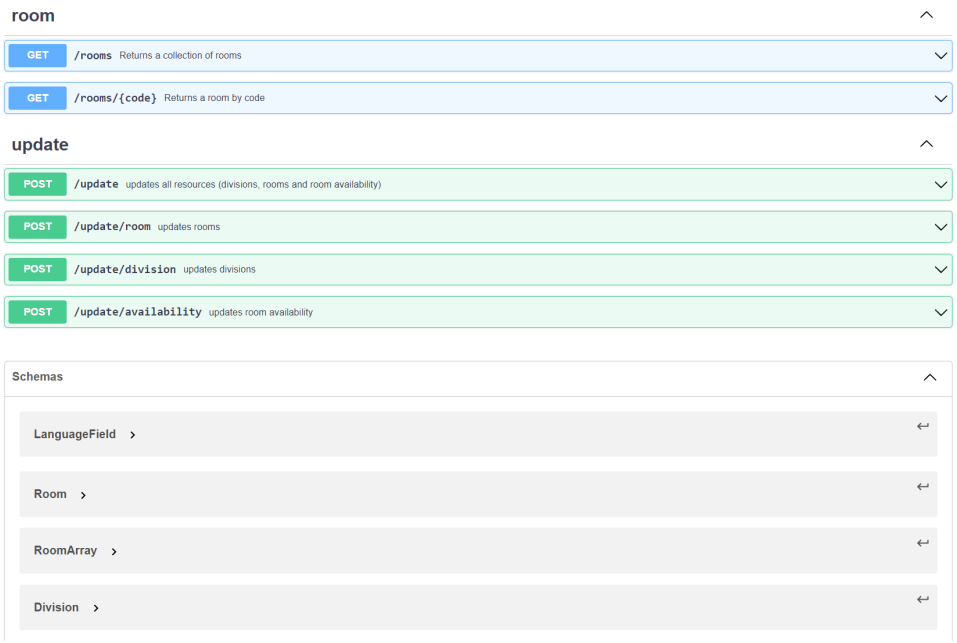
Ukázka rozhraní Core Service

```
type Query{
    courseMany(language: LanguageCode): [CourseGQL]
    courseOne(language: LanguageCode): CourseGQL
    courseManyByCode(code: String, language: LanguageCode): CourseGQL
}
scalar Date

type Mutation{
    runUpdatesAll: Boolean
    runUpdate(update: UpdateName):Boolean
}

##########################################

type CourseGQL {
    externalKosId: String
    code: String!
    name: LanguageFieldGQL!
    completion: String
    credits: Int
    departmentCode: String
    range: String
    season: SemesterPeriodTypeGQL
    state: String
    studyForm: String
}
```

Obrázek E.1: GraphQL schéma Core Service.

Příloha F

Ukázka rozhraní Room Service



Obrázek F.1: Grafické zobrazení rozrhaní Room Service.

Příloha G

Ukázka integračních testů Timetable Service

```java
@DataMongoTest
@Import(MongoConfig.class)
@ContextConfiguration(
    classes = {
      PlannerService.class,
      PlannerTimetableSlotMapperImpl.class,
      PlannerSelectionMapperImpl.class
    })
public class PlannerServiceIT {

  1 usage
  @MockBean private CoursesEnrollmentApi coursesEnrollmentApi;

  1 usage
  @Autowired private PlannerSelectionMongoRepository plannerSelectionMongoRepository;

  6 usages
  @Autowired private PlannerService plannerService;
  6 usages
  private List<String> coursesInitial;

  no usages  Ladislav svoboda
  @BeforeEach
  public void setup() {
    plannerSelectionMongoRepository.deleteAll();
    coursesInitial = List.of("B0B01LAG", "B0B01LGR", "B0B35APO", "B0B36DBS");

    Mockito.when(coursesEnrollmentApi.getStudentCourses(Mockito.any(), Mockito.any()))
        .thenReturn(ResponseEntity.ok(coursesInitial));
  }
```

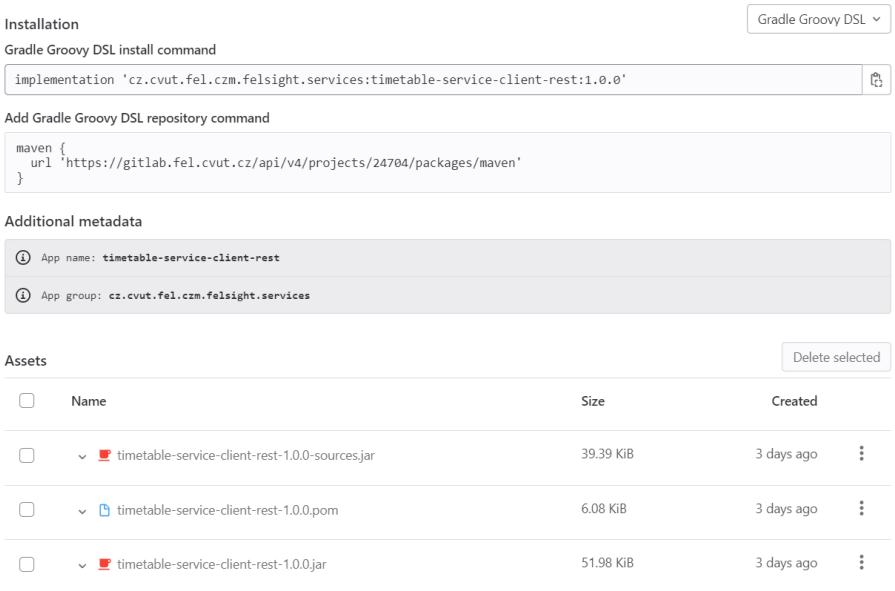
Obrázek G.1: Inicializace testovací třídy.

```java
@Test
public void addCourseWithSelectionNoSelectionInDb() {
    PlannerSelection selection = plannerService.getPlannerSelection( username: "svobola7");
    Assertions.assertTrue(
        selection.getPlannerTimetableCourses().keySet().containsAll(coursesInitial));

    String newCourse = "B00TEST";
    PlannerSelectionCourse entry = new PlannerSelectionCourse( tutorialCode: 122,  laboratoryCode: 125);
    selection.getPlannerTimetableCourses().put(newCourse, entry);

    PlannerSelectionCourse apoChanged = selection.getPlannerTimetableCourses().get("B0B35APO");
    apoChanged.setLaboratoryCode(500);
    apoChanged.setTutorialCode(100);
    selection.getPlannerTimetableCourses().put("B0B35APO", apoChanged);

    selection = plannerService.saveUsersPlannerSelection(selection);
    Assertions.assertTrue(
        selection.getPlannerTimetableCourses().keySet().containsAll(coursesInitial));
    Assertions.assertTrue(selection.getPlannerTimetableCourses().containsKey(newCourse));
    Assertions.assertEquals(selection.getPlannerTimetableCourses().get(newCourse), entry);
    Assertions.assertEquals(selection.getPlannerTimetableCourses().get("B0B35APO"), apoChanged);
}
```

Obrázek G.2: Ukázka implementace scénáře.

Příloha H

Ukázak generace klienta



Obrázek H.1: Ukázka repozitáře obsahujícího klienta pro monolit.

```java
@Named
@Stateless
public class TimetableApiImpl extends TimetableApi implements TimetableClient, Serializable {
  1 usage
  private static final Logger logger = Logger.getLogger(TimetableApiImpl.class.getName());
  1 usage
  private static final boolean SHOW_DELETED = false;

  1 usage
  @Inject private TimetableSlotMapper timetableSlotMapper;

  2 usages
  @Resource(lookup = "resource/timetableapi", type = Properties.class)
  private Properties jndiProperties;

  no usages  Ladislav svoboda
  @PostConstruct
  private void init() {
    getApiClient()
        .setBasePath(jndiProperties.getProperty("path"))
        .addDefaultHeader("authorization", jndiProperties.getProperty("token"));
  }
```

Obrázek H.2: Ukázka inicializace klienta uvnitř projektu monolitu.