FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Anketa ČVUT - refaktoring backend
Student:	Oleksandr Chmel
Vedoucí:	Ing. Michal Valenta, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2024/2025

Pokyny pro vypracování

Cílem práce je revize a refaktoring backend části aplikace Anketa ČVUT. V rámci toho dojde ještě k promítnutí změn na databázové úrovni v části aplikace, která zobrazuje výsledky anket.

Postupujte v těchto krocích:

1. Seznamte se s aktuálním stavem bakend a popište ho.
2. Navhrněte změny, které podpoří udržitelnost projektu, diskutujte je s vedoucím práce a následně aplikujte.
3. Kvalitu nového kódu zkontrolujte pomocí vhodného nástroje (např. Sonar Qube).
4. Zaveďte systematický přístup hlašení chyb z backend.
5. V součinnosti s vedoucím práce změňte backend tak, aby vyhodnocení anket nevytvářelo další materializované pohledy.
6. Navhněte a po domluvě s vedoucím práce implementujte vhodné testy backend tak, aby mohly být spouštěny v rámci CI/CD.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 8. června 2023 v Praze.

Diplomová práce

ANKETA ČVUT - REFAKTORING BACKEND

Bc. Oleksandr Chmel

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Michal Valenta, Ph.D.
11. ledna 2024

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Chtěl bych poděkovat především svému vedoucímu panu Ing. Michalu Valentovi PhD., za jeho čas, trpělivost, podporu a neocenitelné rady během procesu vypracování této diplomové práce. Rád bych poděkoval také své drahé přítelkyni Berenice a svým kamarádům, zejména Filipovi, kteří mě při vytváření této práce podpořili a bez jejich pomoci by nebylo možné práci dokončit.

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, software) uvedené v bibliografii nebo textu této diplomové práce.

V Praze dne 11. ledna 2024 .....

Abstrakt

České vysoké učení technické využívá digitální řešení Anketa k průzkumu spokojenosti studentů s vyučovanými předměty. Ovšem poslední úprava základu této aplikace byla implementována několik let zpátky, a je tedy vůči velmi rychlému technologickému pokroku zastaralá. Zároveň obsahuje některá bezpečnostní rizika. Tato práce se věnuje inovativnímu návrhu a refactoringu backend komponenty pro vyplňování anketních lístků. To má za cíl odstranit potenciální bezpečnostní rizika a zlepšit škálovatelnost softwaru.

Klíčová slova Anketa ČVUT, Kotlin, Spring WebFlux, Gradle, kvalita kódu, udržitelnost

Abstract

The Czech Technical University uses the Anket digital solution to survey student's satisfaction factor with taught subjects. However, last modification of the application core was implemented several years ago, and is therefore outdated compared to rapid technological progress. At the same time, it contains some security risks. This work is dedicated to the innovative solution design and refactoring of the backend component for filling surveys. This should eliminate potential security risks and improve software scalability.

Keywords Anketa CTU, Kotlin, Spring WebFlux, Gradle, code quality, maintainability

Seznam zkratek

CI/CD Continuous integration and continuous delivery

CSS Cascading style sheets

CSV Comma-separated values

ČVUT České vysoké učení technické

DOM Document object model

DSL Domain specific language

DSN Data source name

IoC Inversion of control

JAR Java archive

JDBC Java database connectivity

JS JavaScript

JSON JavaScript object notation

JVM Java virtual machine

JWT Json web token

LDAP Lightweight directory access

MVC Model-View-Controller

ORM Object-relation mapping

POM Project object model

PU Případ užití

RAM Random access memmory

SAML Security Assertion Markup Language

SEO Search engine optimalization

SSO Single sign-on

SQL Structured query language

SPA Single-page aplikace

XML Extensible markup language

XSS Cross-site scripting

WAR Web archive

Úvod

Vysoké školy v dnešní době kladou velký důraz na kvalitu výuky a studijních materiálů, a také v neposlední řadě spokojenosti studentů. Metody jako papírové dotazníky nebo ústní rozhovory jsou kvůli vysokému počtu studentů na Českém vysokém učení technickém náročné na zpracování a vyhodnocení. Proto se hledají digitální řešení, která kromě usnadnění sběru dat umožní rychlejší a efektivnější analýzu výsledků.

Na ČVUT se takovým digitálním řešením stala aplikace Anketa, která zde funguje již od roku 2002. V roce 2018 došlo k velkému refactoringu jak serverové části, tak i vzhledu. Této problematice se věnovala řada závěrečných práci na Fakultě informačních technologií. Aplikace umožňuje administrátorovi vytvářet ankety různých typů na úrovni jednotlivých fakult. Studenti pak mají možnost hodnotit nejen konkrétní předmět, ale také vybrané vyučující a samotnou fakultu, což ve výsledku poskytuje neocenitelnou zpětnou vazbu vedení fakulty, vyučujícím a studentům nižších ročníků.

Motivace a cíle práce

Motivací ke psaní této diplomové práce byl skutečný praktický přínos pro univerzitu, respektive fakultu. Volba daného tématu umožnila volnost ve výběru technologií, možnost prozkoumat a integrovat inovativní řešení a také stanovit nový technologický směr vývoje aplikace Anketa.

Hlavním cílem této závěrečné práce je refactoring backendu pro aplikaci Anketa, který bude zahrnovat revizi používaných technologií, aktualizaci závislostí a nastavení nových projektových standardů. Pro splnění stanoveného cíle je potřeba důkladně analyzovat a popsat stávající řešení pro stanovení kroků provedení rafactoringu. Navrhnout změny ve stávající aplikaci a zvolit technologie pro jejich realizaci. Implementovat změny do zdrojového kódu aplikace a zavést nástroje, které podpoří udržitelnost projektu do budoucna. Zprovoznit aktualizovanou verzi aplikace na vývojovém, případně produkčním prostředí, čímž se prokáže funkčnost nového řešení. Zdokumentovat provedený postup za účelem rozvoje aplikace Anketa v následujících letech.

Text závěrečná práce je rozdělen do dvou hlavních částí: teoretické a implementační, přičemž každá z nich obsahuje několik kapitol věnovaných jednotlivým aspektům vybraného tématu. Teoretická část nejdříve popisuje problémovou doménu a seznamuje čtenáře s hlavními pojmy. Dále sleduje popis existujícího řešení a jeho nedostatků. Následně jsou uvedeny požadavky vedoucího závěrečné práce, jejich analýza a osobní úvaha nad volbou jednotlivých technologií a postupů.

Praktická část této diplomové práce je zaměřena na postup refactoringu backendu aplikace Anketa. Ta obsahuje popis postupu vývoje nového řešení a také praktické ukázky kódu, které mají za cíl zlepšit odezvu a udržitelnost aplikace. Dále sleduje popis implementace testů na projektu a jejich nastavení v rámci CI/CD.

Související závěrečné práce

Vývoji aplikace Anketa ČVUT a analýze požadavků na systém se již věnovalo kolem deseti závěrečných prací. Zde jsou uvedeny některé z nich:

1. Vojtěch Štecha: *Anketa ČVUT - verze 3.0 - vyplňování anketních lístků.* Tato diplomová práce popisuje vývoj nové aplikace Anketa ČVUT. Analyzuje dostupné požadavky, uvádí a porovnává možné návrhy realizace. [1]

2. Alice Kopalová: *Automatizace administrátorských procesů databáze aplikace Anketa ČVUT.* Náplní této práce je analýza a dokumentace administrátorských procesů v aplikaci Anketa ČVUT, návrh řešení, jak administrátorské procesy spouštět z backendu aplikace, a jeho implementace. [2]

3. Nam Nguyen Hai: *Anketa ČVUT - redesign vyplňovacího modulu.* Táto práce se věnuje redesignu komponenty vyplňování anketních lístků aplikace Anketa ČVUT, která slouží pro sběr zpětné vazby od studentů na vysoké škole. [3]

4. Jakub Jun: *Anketa ČVUT verze 3 - modul pro správu anket - uživatelské rozhraní.* Cílem práce je návrh, implementace a nasazení uživatelského rozhraní (frontend) modulu pro správu anket v rámci projektu Anketa ČVUT verze 3. [4]

5. Ilya Parakhnich: *Anketa ČVUT v 4.0 - Datové schéma a byznys vrstva v databázi.* Tato diplomová práce se zabývá kompletním refactoringem databázové vrstvy aplikace Anketa ČVUT a také návrhem systematické správy jejích zdrojových kódů. [5]

Kapitola 1

Popis problémové domény

Tato kapitola je zaměřena na popis problémové domény, hlavních pojmů a vydefinování rolí v aplikaci. To má za cíl pomoct čtenáři lépe pochopit fungování Ankety jako celku a procesy, které probíhají uvnitř aplikace.

1.1 Základní pojmy

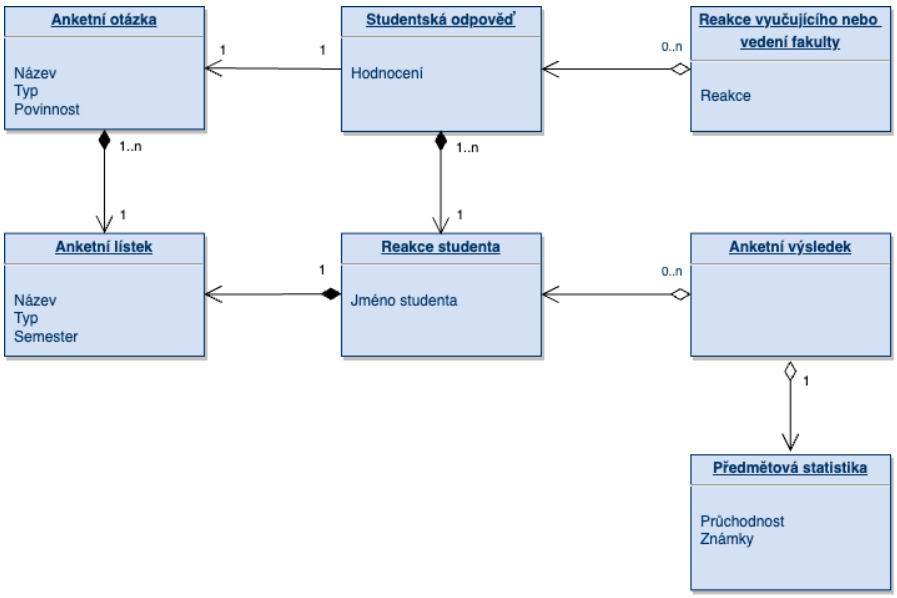
Webová aplikace Anketa pomáhá Českému vysokému učení technickému získávat zpětnou vazbu od studentů k jednotlivým předmětům, vyučujícím nebo fakultám. Jedná se o komplexní systém, který se skládá z několika samostatných komponent: komponenty vyplňování anketních lístků, komponenty administrace pro správce ankety, komponenty generátoru reportů anketních výsledků. Vzhledem k tomu, že tato diplomová práce je zaměřená na refactoring backendu pro vyplňování anketních lístků, následující odstavce budou popisovat pouze základní pojmy, které jsou relevantní pro zvolené téma.

V kontextu vyplňování anketních lístků, lze vydefinovat následující základní pojmy:

- **Anketní otázka:** Je to formulovaný dotaz, který se prezentuje studentům s cílem získat jejich názor a hodnocení spokojenosti tykající se různých aspektů výuky v semestru a obecně fakultního prostředí. Otázky jsou dvou typů: škálové, kde student může zvolit hodnocení od 1 do 5 a otevřené, kde student má možnost do komentáře vyjádřit svůj pohled na jeden z aspektů studia.
- **Anketní lístek:** Jedná se o formulář obsahující seznam anketních otázek, který vedení fakulty připravuje ke konci vyučovacího semestru. Typ anketního lístku se může lišit, přičemž může nabývat formy buď *předmětového* a *nepředmětového* charakteru. Student má každý semestr možnost vyplnit jeden nepředmětový anketní lístek a více předmětových v závislosti na počtu absolvovaných předmětů.
- **Reakce studenta:** Jde o sadu odpovědí poskytnutých studentem na otázky v anketním lístku, která zahrnuje jak škálové hodnocení na stupnici od 1 do 5, tak i zanechanou textovou zpětnou vazbu. Tyto odpovědi dohromady poskytují pohled očima studenta na průběh výuky předmětu a akademického roku na fakultě.
- **Reakce vyučujícího / vedení fakulty:** Je to textový komentář zanechaný vyučujícím předmětu nebo vedením fakulty u studentské reakce na anketní otázku. Může se jednat o poděkování za poskytnutou zpětnou vazbu nebo o reakci na studentské připomínky či návrhy.

- **Anketní výsledek:** Jde o souhrn všech studentských odpovědí a dat získaných z anketních lístků. Tento výsledek poskytuje přehledné informace, které reflektují názory studentů na průběh předmětů během semestru, přístup jednotlivých vyučujících k provedení výuky a postoj vedení fakulty k problémům, které respondenti zaznamenali. Kromě toho je výsledek předmětové ankety obohacen o statistické údaje, které umožňují lepší interpretaci zobrazovaných dat.

Na základě výše uvedených základních pojmů lze vytvořit zjednodušený diagram doménového modelu[6], viz. obrázek 1.1, který by měl sloužit k lepšímu pochopení kontextu aplikace a vazeb mezi doménovými objekty.



Obrázek 1.1 Zjednodušený diagram doménového modelu aplikace Anketa ČVUT

1.2 Uživatelské role

S ohledem na již představené základní pojmy lze vydefinovat tři uživatelské role v aplikaci. Ty slouží k rozdělení zón zodpovědnosti a plnění speciálních případů užití.

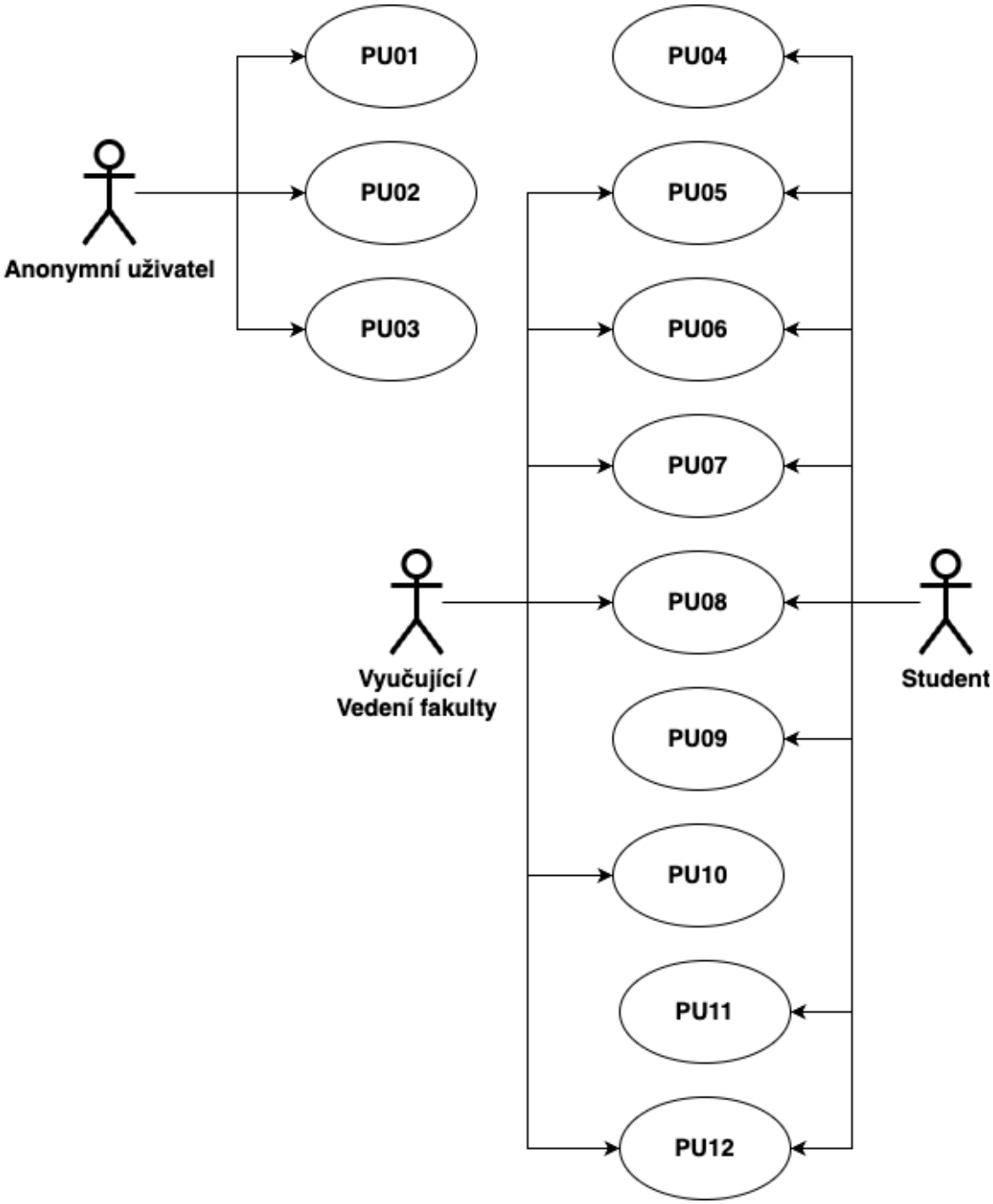
- **Student** - vyplňuje hodnocení a komentuje průběh absolvovaných předmětů, zanechává zpětnou vazbu vyučujícím a vedení fakulty.
- **Vyučující** - reaguje na studenské komentáře.
- **Vedení fakulty** - reaguje na studenské komentáře, odstraňuje nevhodné nebo irelevantní reakce k předmětu.

1.3 Případy užití

Případ užití (PU) (z angl. Use Case)[7] je textový popis funkcionality systému z pohledu jeho koncového uživatele nebo zákazníka. Tento pojem se často používá v kontextu softwarového inženýrství, protože případy užití jsou klíčovým faktorem pro pochopení toho, jak má výsledný systém fungovat a jaký návrh použit. Tato sekce shrnuje případy užití pro komponentu vyhodnocení anketních lístků aplikace Anketa, které jsou uvedeny v tabulce 1.1. Pro snadnější vizualizaci je tabulka s případy užití doprovázena use-case diagramem na obrázku 1.2, který znázorňuje vztah mezi aktéry (uživatelskými rolemi) a případy užití.

Tabulka 1.1 Případy užití

	Popis
PU01	Autentizace prostřednictvím Single Sign-on: Anonymní uživatel by měl mít možnost se přihlásit do aplikace prostřednictvím univerzitního poskytovatele identity Shibboleth.
PU02	Autentizace prostřednictvím LDAP: Anonymní uživatel by měl mít možnost se přihlásit do aplikace prostřednictvím univerzitního Lightweight directory access (LDAP) protokolu.
PU03	Interní autentizace: Anonymní uživatel by měl mít možnost se přihlásit do aplikace pomocí přiděleného interního identifikátoru a zvoleného hesla.
PU04	Zobrazení anketních lístků: Přihlášený uživatel s rolí studenta by měl mít možnost se dostat na stránku se seznamem anketních lístku pro aktuální semestr a zobrazit si detail vybraného předmětového nebo nepředmětového anketního lístku.
PU05	Zobrazení výsledků ankety: Přihlášený uživatel by měl mít možnost se dostat na stránku výsledků anketních lístků z minulých semestrů.
PU06	Detail vyučujícího: Přihlášený uživatel by měl mít možnost si zobrazit anketní statistiky k vybranému vyučujícímu a prohlédnout si studentskou zpětnou vazbu v podobě textových komentářů.
PU07	Detail předmětu: Přihlášený uživatel by měl mít možnost si zobrazit anketní statistiky ke zvolenému předmětu a prohlédnout si studentskou zpětnou vazbu v podobě textových komentářů. Kromě toho detail předmětu zobrazí vyjádřeni jednotlivých vyučujících předmětu k jeho průběhu v semestru.
PU08	Export výsledků: Přihlášený uživatel by měl mít možnost exportovat výsledky předmětové ankety vybraného semestru do souboru ve formátu Comma-separated values (CSV). Soubor by měl obsahovat anketní statistiky předmětů vyučovaných na fakultě pro vybraný semestr.
PU09	Vyplňování anketních lístků: Přihlášený uživatel s rolí studenta by měl mít možnost vyplnit předmětové a nepředmětové anketní lístky pro aktuální vysokoškolský semestr.
PU10	Reakce vyučujícího / vedení fakulty: Přihlášený uživatel s rolí vyučujícího nebo vedení fakulty by měl mít možnost odpovědět textovým komentářem na studentskou zpětnou vazbu.
PU11	Hodnocení reakce: Přihlášený uživatel s rolí student by měl mít možnost hodnotit komentáře ostatních studentů prostřednictvím pozitivní nebo negativní reakce.
PU12	Zpětná vazba pro aplikaci: Přihlášený uživatel by měl mít možnost zanechávat zpětnou vazbu vývojovému týmu aplikace Anketa nebo nahlašovat chybné chování systému.



Obrázek 1.2 Diagram případů užití komponenty vyplňování anketních lístků aplikace Anketa

1.4 Časový průběh

Tato sekce se věnuje popisu časového průběhu vyplňování anketních lístků. Na konci zimního a letního semestru je studentům poskytnuta příležitost vyjádřit svůj názor a podělit se o zkušenosti s vybranými předměty. Dále studenti mohou zmínit svou spokojenost/nespokojenost s vedením fakulty a diskutovat přístupy jednotlivých vyučujících. Odpovědi v anketních lístcích jsou ve výchozím nastavení anonymní a studenti mohou hodnotit pouze absolvované předměty.

Po dokončení sběru informací od studentů správce systému uzavře možnost vyplnit anketu a proběhne zpracování lístků, trvající obvykle několik týdnů. V tomto období vyučující mohou odpovídat na studentské komentáře a také skrývat ty nevhodné nebo irelevantní. Následně proběhne publikace výsledků anketních lístků včetně statistik známek a průchodnosti předmětů. V tomto stavu je povoleno pouze nahlížení výsledků bez možnosti přidání reakce. Detailní časový průběh Ankety je popsán v metodickém pokynu provozu systému[8].

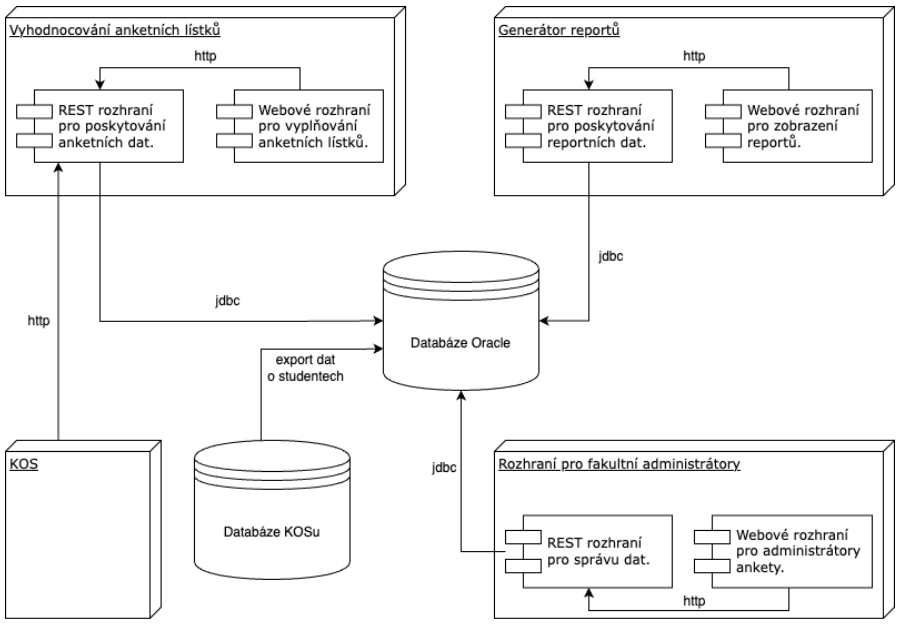
Výsledky Ankety hrají důležitou roli ve fungování ČVUT. Studenti tak mohou zanechat zpětnou vazbu vedení fakulty nebo jednotlivým vyučujícím a tím ovlivnit průběh předmětů v následujících semestrech. Zároveň hodnocení z předmětových anket může sloužit jako cenné rady pro ostatní studenty, například při výběru volitelných předmětů.

Kapitola 2
Popis existujícího řešení

Tato kapitola je zaměřena na popis existujícího řešení z pohledu softwarové implementace. Nejdříve je uveden celkový pohled na aplikaci Anketa a potom následuje popis jednotlivých modulů komponenty vyplňování anketních lístků, což je kontextem této práce. To má za cíl pomoci čtenáři si lépe představit architekturu aplikace a seznámit ho s použitými technologiemi.

2.1 Celkový pohled

Aplikace Anketa, jak již bylo zmíněno v předchozí kapitole, se skládá z několika samostatných komponent, kde každá z nich je zodpovědná za určitou sadu případů užití, a k ní příslušných uživatelských rolí. Zjednodušený koncept aplikace Anketa je zobrazen na obrázku 2.1. Z obrázku je vidět, že „zdrojem pravdy“ pro tři uvedené moduly je centrální Oracle databáze, která zajišťuje komunikaci mezi nimi. Detailněji se jednotlivým částem komponenty vyplňování anketních lístků budou věnovat následující odstavce této kapitoly.



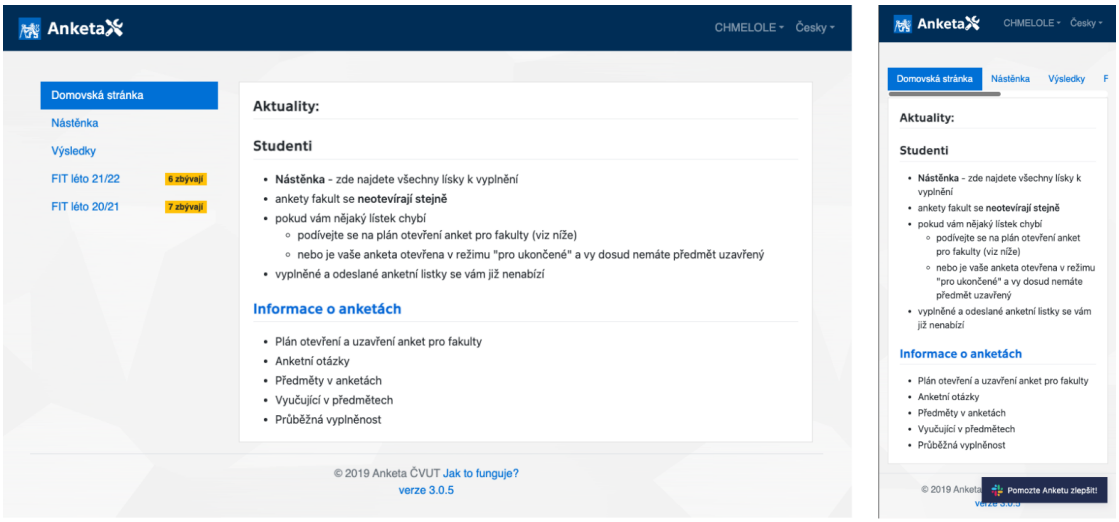
Obrázek 2.1 Diagram komponent aplikace Anketa ČVUT

2.2 Aplikační frontend

Stávající řešení, které vzniklo v rámci diplomové práce Vojtěcha Štechy a bakalářské práce Jakuba Juna, je momentálně živě dostupné na *https://anketa.is.cvut.cz/html/anketa/*. Pro nepřihlášeného uživatele je dostupná pouze domovská stránka webové aplikace se statickým obsahem. Ostatní sekce webu vyžadují autentizaci prostřednictvím univerzitního SSO nebo pomocí uživatelského jména a hesla pro externí vyučující. Po úspěšné autentizaci se uživateli vytvoří unikátní identifikátor *JSON Web token*, který server používá pro kontrolu přístupu ke zdrojům. Detailněji se procesu autentizace a autorizace věnuje následující odstavce.

Přihlášený uživatel má k dispozici jednu ze tří uživatelských rolí popsaných v předchozí kapitole. Všechny role mají přístup k sekci Výsledky, která nabízí tabulkové zobrazení a filtrování výsledků anket podle semestru a názvu. Dále si přihlášený uživatel může zobrazit detail anketního lístku podle fakulty, předmětu nebo učitele. Fakultní nebo nepředmětový anketní výsledek poskytuje celkové hodnocení průběhu semestru na fakultě a také obsahuje studentské zpětné vazby a postřehy ke zlepšení. V detailu předmětové ankety jsou studentské recenze k průběhu jednotlivých předmětů v semestru, jejich časové náročnosti a obtížnosti zkoušek. Navíc jsou zde zaznamenány statistiky známek a grafy průchodnosti k jednotlivým předmětům na základě výsledků z minulých let. To slouží, zejména ostatním studentům, při výběru volitelných předmětů. Hodnocení přístupu k výuce jednotlivých vyučujících lze nalézt v detailu učitele. Každý detail předmětového anketního lístku má u sebe rozcestník na jednotlivé vyučující. Studenti vidí, zda si učitel komentáře všimnul a také mohou přidávat „palec nahoru“ k užitečným zpětným vazbám. Vyplňování anketních lístků je dostupné pouze uživatelům s rolí *student* ve specifickém období.

Z technologického pohledu se webová stránka tváří jako single-page aplikace, která se dokáže přizpůsobit zobrazení na různých velikostech obrazovky, viz. obrázek 2.2. Takovému webu se říká responzivní[9]. Zdrojové kódy k aplikaci a proces nasazení se spravují prostřednictvím školního GitLabu. Za tyto kódy je zodpovědný pan Ing. Michal Valenta PhD (vedoucí této diplomové práce). Při bližším seznámení s projektem se ukázalo, že aplikace je napsána v jazyce JS a využívá frameworky React a Axios. Jde o atraktivní kombinaci technologií pro vývoj moderních webových platforem. Ovšem s ohledem na datum poslední aktualizace souboru *package.json*, jsou některé verze projektových závislosti dost zastaralé a vyžadují aktualizaci. Detailněji se technologiím aplikačního frontendu věnují následující podkapitoly.



Obrázek 2.2 Domovská stránka aplikace Anketa ČVUT na různých zařízeních

2.2.1 Single-page aplikace

Single-page aplikace nebo také SPA jsou webové aplikace nebo webové stránky, které interagují s uživatelem dynamickým přepisováním obsahu aktuální stránky v prohlížeči, místo načtení nového webového dokumentu ze serveru. V takovém případě se většina obsahu a logiky načte při prvotním spuštění aplikace v prohlížeči a následující uživatelské interakce vedou k asynchronnímu volání serverového API a aktualizacím Document object model (DOM)[10] na straně klienta.

Motivaci k vývoji a zavedení nového moderního standardu SPA se stal postupný přesun služeb do internetu a úsilí o zlepšení uživatelského zážitku při prohlížení webových stránek. To bylo možné díky zvyšujícímu se výkonu osobních počítačů a mobilních telefonů, stejně jako pokroku ve vývoji technologie JS. Dříve nejpoužívanější architektura Model-View-Controller (MVC)[11] pro webové aplikace se potýkala s několika nedostatky. Hlavním z nich byla dlouhá doba načtení stránky do prohlížeče při každé interakci s uživatelem. Koncept SPA odstranil nebo výrazně zredukoval tento problém. Kromě toho, nový koncept přinesl do světa vývoje řadu technických benefitů:

- **Snížení zátěže na server:** S příchodem nového konceptu zpracování uživatelských událostí a vykreslování stránky uživateli začalo probíhat na straně prohlížeče, čímž se snížila zátěž na server a služba tak byla schopna obsloužit větší počet návštěvníků.
- **Lepší rozdělení zodpovědnosti:** SPA odděluje logiku zobrazení dat uživateli od logiky načtení dat z databáze. Tím pádem serverová část aplikace je zaměřená pouze na zpracování uživatelských dotazů a čtení odpovídajících záznamů z úložiště. Oddělení dvou konceptů od sebe umožňuje dobré rozdělení rolí a sestavení vývojového týmu.
- **Jednodušší integrace:** Kvůli již zmíněnému oddělení logiky zobrazení dat a načtení dat do dvou aplikací se vývojářům otevírají nové možnosti. Jednou z nich je taková, že může existovat několik různých webových klientů (například odlišný vzhled aplikace podle fakulty), které využívají sdílený API server. Druhou možností je vývoj konceptuálně odlišné aplikace, například pro mobilní telefony nebo desktop, která by byla napojená na poskytované backendem rozhraní.
- **Snadnější implementace offline:** Díky schopnosti moderních prohlížečů cachovat obsah mohou SPA poskytnout funkčnost uživateli i v režimu offline. Data se po zpětném napojení na internet synchronizují se serverem.

I přes zmíněné výhody mají SPA některé nedostatky. Jedním z potenciálních problému je Search engine optimalization (SEO)[12]. Načítání většiny obsahu pomocí jazyku JavaScript (JS) komplikuje proces indexace obsahu stránek ve vyhledávačích. Dalším z problémů může být pomalejší načtení aplikace do prohlížeče při první interakci, což je způsobené stahováním většího množství zdrojů. Navíc SPA jsou také náchylnější k některým druhům bezpečnostních rizik, jako je například Cross-site scripting (XSS)[13]. Jde o typ bezpečnostního útoku na webové aplikace, při kterém se útočník pokouší odcizit citlivá data uživatele prostřednictvím vkládání škodlivého kódu do obsahu stránky.

V kontextu aplikace pro vyplňování anketních lístků lze zanedbat komplikaci procesu indexace stránek, protože většina obsahu je znepřístupněna anonymnímu uživateli. S ohledem na velikost aplikace a schopnost moderních prohlížečů cachovat obsah webových stránek lze zanedbat čas načtení do browseru. Při testech v prohlížeči Google Chrome sestava DOM stránky trvala 300ms. Náchylnost k některým bezpečnostním rizikům, jako je XSS, je spíše záležitost volby technologie implementace, než obecně konceptu SPA. O tom projednává následující podkapitola, která se věnuje popisu zvolené knihovny *React*.

S ohledem na uvedené výhody a nevýhody, koncept SPA se v kontextu aplikace Anketa ČVUT jeví jako perspektivní a moderní volba, podporující udržitelnost projektu do budoucna.

2.2.2 React

React je knihovnou v jazyce JS, která usnadňuje proces tvorby mobilních a webových aplikací. Jeho největší výhodou je schopnost rychle a efektivně zpracovávat uživatelské události ze vstupního rozhraní a propisovat změny zpět. Knihovna zastupuje roli View z pohledu architektonického návrhu MVC. Pro vytvoření plnohodnotné webové aplikace je často zapotřebí zapojení externích knihoven, jako je například *Redux* nebo *Axios*.

Samotný React nepracuje přímo s DOM prohlížeče, ale vytváří vlastní virtuální strom komponent uživatelského rozhraní neboli VDOM. O vykreslování specifických HTML prvků se stará *ReactDOM*. V podstatě jde o oddělení procesu vytváření stromu uživatelského rozhraní od procesu vykreslování samotných komponent na obrazovku. Tento koncept se dá použít i ve vývoji mobilních aplikací, kde React je zodpovědný za zprávu uživatelského rozhraní, a *React Native* za komunikaci s operačním systémem mobilního telefonu a zobrazení UI prvků na obrazovce.

Základními stavebními bloky pro vytváření uživatelského rozhraní v knihovně React jsou komponenty. Jedná se izolované a znovupoužitelné kusy kódu v rámci projektu. Navíc komponenty mohou být vnořovány do sebe, což umožňuje stavbu složitějších prvků uživatelského rozhraní. Deklarace komponenty v Reactu může být provedená vytvořením třídy nebo funkce, viz. výpis kódu 1. Většina vývojářů preferuje funkční způsob deklarace komponenty, pro správu stavu a životního cyklu komponenty používají speciální funkce, tzv. „hooks“[14]. Mezi ty nejvíce používané patří například *useState* a *useEffect*.

```
1   // Deklarace třídní komponenty
2   class HelloWorld extends React.Component {
3       render() {
4           return <h1>Hello, {this.props.name}</h1>;
5       }
6   }
7   // Deklarace funkční komponenty
8   function HelloWorld(props) {
9       return <h1>Hello, {props.name}</h1>;
10  }
```

Výpis kódu 1 Ukázka deklarace komponenty v Reactu

2.2.3 Bootstrap

Nejčastějším problémem ve vývoji webových aplikací je responzivní návrh uživatelského rozhraní. V moderní době, s rostoucí popularitou mobilních zařízení, je důležité, aby webové stránky vypadaly atraktivně a fungovaly správně i na menších obrazovkách. Proto se v posledních letech aktivně vyvíjí projekty s otevřeným zdrojovým kódem zaměřené specificky na design a rozložení prvků na webové stránce. Mezi takové projekty patří například *Material Design* od Googlu nebo *Bootstrap* od Twittru. Volba knihovny je vždy na osobním uvážení vývojáře. [15]

Knihovna *Bootstrap* byla navržena tak, aby se dala snadno integrovat do nového nebo již existujícího projektu. Knihovna nabízí komplexní sadu nástrojů pro vývoj webového uživatelského rozhraní. Ta disponuje velkou kolekcí Cascading style sheets (CSS)[16] stylů pro vytváření responzivního designu. Sadu dynamických komponent v jazyce JS pro jednodušší sestavení webových prvků jako je vyklápěcí menu, modální okno, obrázkový karusel nebo jiné interaktivní komponenty. *Bootstrap* přispívá k urychlení procesu implementace běžných komponent uživatelského rozhraní a eliminuje potřebu vytvářet duplicitní kód, což v důsledku vede k lepší udržitelnosti projektu do budoucna.

2.2.4 Axios

Jedná se o populární JavaScriptovou knihovnu, která se používá ve webových aplikacích pro HTTP komunikaci na stráně klienta (často prohlížeče). Axios podporuje běžné HTTP metody jako jsou GET, POST, PUT, DELETE. Velkou výhodou knihovny je automatický překlad aplikačních objektů do JavaScript object notation (JSON) formátu a zpátky při odesílání dat, resp. přijímání odpovědí. Při zpracování velkého množství dat Axios nabízí API pro sledování průběhu procesu nahrání dat na server. Také knihovna poskytuje jednoduchý a efektivní způsob ošetření chyb v HTTP požadavcích. Navíc Axios poskytuje možnost přidání kódu upravujícího obsah HTTP požadavků (angl. interceptor)[17], který je zvlášť užitečný pro případy jako je autentizace nebo zaznamenávání dat. Ukázka volání backendu pomocí Axios je znázorněna ve výpisu kódu 2, kde dochází k ověření uživatelského přihlášení.

```
1   Axios.post('/login/create', {
2       token: queryString.parse(this.props.location.search).token,
3       password: this.state.password,
4   }).then(response => {
5       this.setState({
6           success: true,
7       });
8   }).catch(error => {
9       console.error('Došlo k chybě:', error);
10  });
```

Výpis kódu 2 Ukázka volání backendu pomocí Axios

2.3 Aplikační backend

Stávající řešení aplikačního backendu, podobně jako frontendová část, vzniklo v rámci diplomové práce Vojtěcha Štechy a také za působení Duc Thang Nguyena. Potom se některé části aplikace doplňovaly vývojářem Šimonem Valentou. Serverová implementace je postavená na technologiích *Java*, *Maven*, *Hibernate* a *Spring Boot*, o kterých pojednávají následující podkapitoly. Aplikace poskytuje Rest rozhraní, kterého využívá frontendová část pro přijímání a odesílání dat. Většina endpointů je chráněna použitím technologie JWT, která zajišťuje autorizaci uživatelských požadavků. Aplikace je napojena na Oracle databázi prostřednictvím Java database connectivity (JDBC) rozhraní, které poskytuje standardizovaný protokol přístupu k datovému úložišti.

Projekt je spravován a udržován ve školním GitLabu a jeho obsah je rozdělen do několika balíčků (resp. package), které sjednocují třídy se stejnou „zónou zodpovědností“. Pro zjednodušení procesu nasazení do vývojového a produkčního prostředí bylo využito CI/CD integrovaného do GitLabu. Kód v aplikaci lze rozdělit na dvě funkční vrstvy a datové nebo podpůrné třídy:

- **Rest vrstva:** Jsou to třídy se společným sufixem *Controller*, které definují Rest operace pro přístup k serverovým zdrojům.
- **Servisní vrstva:** Jsou to třídy se společným sufixem *Service*, které implementují sestavení SQL dotazů a provádí mapování dat do uživatelsky přijatelné podoby.
- **Datové třídy:** Jsou to třídy se sufixy *Entity*, *View*, *Dto* nebo obsah balíčku *util*. Slouží primárně jako datové obálky obsahu z databáze nebo pro generování uživatelského autorizačního tokenu.

S ohledem na datum poslední aktualizace projektových zavilostí lze usoudit, že projekt není udržován pravidelně. Verze použitého frameworku Spring Bootu *1.5.6* je ke dnešnímu dni 5 let stará a již od dubna roku 2023 není aktivně podporována. Použitá verze aplikačního frameworku, stejně tak i verze řady použitých nástrojů, obsahují řadu bezpečnostních zranitelností, které jsou opraveny v následujících verzích.

I přes poměrně jednoduché rozdělení kódu v projektu do dvou vrstev, je projekt v jeho aktuální podobě dost nepřehledný a analýza jednotlivých části aplikace časově náročná i pro zkušenějšího vývojáře. Skoro nikde není použit princip „code against interfaces, not implementation“[18], který značné zjednodušuje testování a údržbu vytvořeného kódu. Často se porušuje princip *single responsibility*[19], a také princip *immutability*, kde se obsah vstupního parametru metody mění v rámci její implementace. V projektu existuje řada tříd s cyklickými závislostmi a nesprávným použitím injektáže. Podle statistik z nástroje *SonarQube* kód obsahuje několik stovek překlepů nebo „code smells“[20], duplicity ve funkcích a není pokrytý testy. K projektu neexistuje dokumentace. Výše uvedené nedostatky způsobují špatnou udržitelnost aplikace do budoucna, a přidání nových funkcionalit bude časově náročné bez hlubší znalosti projektu. Proto bylo rozhodnuto o provedení rozsáhlého refactoringu aplikačních zdrojových kódů a změně systémové architektury, včetně náhrady některých použitých technologií.

2.3.1 Java

Java je v dnešní době populární objektově orientovaný programovací jazyk, který byl vyvinut společností Sun Microsystems (později součástí korporace Oracle). Obrovskou výhodou daného jazyka je platformní nezávislost. Zdrojový kód v jazyce Java je kompilován do *bytecode*, který následně může být spouštěn pomocí Java virtual machine (JVM) na libovolné platformě. Java nabízí přístup k široké sadě standardních knihoven, které zjednodušují práci se systémovými soubory, grafickým uživatelským rozhraním nebo síťovou komunikací. JVM má vestavěnou technologii správy pamětí, která automatický alokuje a uvolňuje systémovou paměť podle potřeb aplikace. Kromě toho Java podporuje vícevláknové programování. Vstupním bodem pro spuštění programu v jazyce Java je metoda *main*, viz. výpis kódu 3.

Na trhu Java působí již od začátku 21. století a celkově bylo vydáno 20 verzí tohoto programovacího jazyka, pokud nebudeme počítat minoritní fix verze. Novější verze přinesli do Javy různá rozšíření a nové koncepty. Do vydání Java 11, byl programovací jazyk dostupný pro komerční i nekomerční použití. Později korporace Oracle zavedla nový licenční model, který vyžadoval komerční podporu. Ten byl zaměřen na podnikové zákazníky, kteří hledali dlouholetou a stabilní podporu. Se vznikem komerční Javy se na trhu objevila i její bezplatná varianta *OpenJDK*. Jde o komunitně řízený projekt s otevřeným zdrojovým kódem. Komerční verze Javy, oproti té zdarma, nabízí rychlejší vydání bezpečnostních záplat a také optimalizace některých pokročilých funkcí.

Serverová část aplikace Anketa využívá verzi jazyka Java 1.8, která byla vydána v roce 2014, což je přibližně 10 let dozadu. I přesto, že oficiální podpora OpenJDK uvedené verze končí až v roce 2026, je doporučené přecházet na novější aktualizace programovacího jazyka. Ty přinášejí vylepšené bezpečnostní funkce, výkonnostní optimalizace kompilátoru a JVM, a také nová API rozšíření.

```
1  public class Main {
2      public static void main(String[] args) {
3          System.out.println("Hello world");
4      }
5  }
```

Výpis kódu 3 Ukázka programu v jazyce Java

2.3.2 Maven

Apache Maven je nástroj pro automatizaci procesu sestavení projektů, zejména napsaných v programovacím jazyce Java. Ten se dá snadno integrovat do vývojového prostředí, jako je například IntelliJ IDEA, což zrychluje proces implementace aplikace. Nástroj podporuje širokou škálu pluginů, které usnadňují generování dokumentace ke zdrojovým kódům, testování projektových funkcí a zpracování konfigurací. Také Maven umožňuje paralelní běh procesu sestavení, což je benefitem pro velké projekty s mnoha moduly. Ovšem před použitím této funkce se doporučuje zjistit zda projekt a integrované pluginy podporují paralelní zpracování.

Nástroj Maven používá Project object model (POM) ve formátu XML pro popis projektu, externích závislostí, procesu sestavení a pluginů. Proto základem Maven projektu je soubor *pom.xml*, který musí být uložen do kořenové složky projektu a dodržovat předepsánou strukturu. Pokud se jedná o projekt s více moduly, potom každý z nich musí obsahovat soubor *pom.xml* definující závislosti mezi projektovými komponentami. Životní cyklus projektu Maven se skládá z několika fází, které definují jednotlivé kroky sestavení. První z nich je *validate*, během které dochází k ověření závislostí a struktury projektu. Pak následuje fáze *compile*, kdy je zdrojový kód přeložen kompilátorem. Pak následuje fáze *test*, která spustí automatizované jednotkové a integrační testy pro ověření funkčnosti nové verze. Následující fáze *package* vytváří soubor formátu Java archive (JAR) nebo Web archive (WAR). Finální fáze *deploy* kopíruje sestavenou verzi aplikace do lokálního nebo vzdáleného repozitáře. Přidáním parametru *skipTests* lze přeskočit spuštění testů v procesu sestavení. Ovšem Maven neposkytuje přímou možnost jak vynechat libovolnou jinou fázi, protože nástroj byl navržen tak, aby jednotlivé kroky sestavení byly sekvenčně závislé na sobě.

Maven je dost spolehlivý nástroj, ale má řadu omezení. Prvním z nich je flexibilita, kde předem definovaný životní cyklus projektu může být omezujícím faktorem pro některé typy nefunkčních požadavků. Druhým z nich je striktně daná struktura konfiguračního souboru ve formátu XML, která zamezuje integraci logiky do procesu sestavení. Třetím omezením je nabídka pluginů, která je dost omezená. Pluginy by v kontextu aplikace Anketa podpořily udržitelnost projektu do budoucna. Proto v rámci refactoringu backendu dojde k nahrazení nástroje Maven za více flexibilní software.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0">
3      <modelVersion>4.0.0</modelVersion>
4      <groupId>cz.cvut.fit.anketa</groupId>
5      <artifactId>backend</artifactId>
6      <version>0.0.1-SNAPSHOT</version>
7      <dependencies>
8          <dependency>
9              <groupId>org.projectlombok</groupId>
10             <artifactId>lombok</artifactId>
11             <version>1.18.12</version>
12         </dependency>
13     <dependencies>
14 </project>
```

Výpis kódu 4 Ukázka konfigurace Maven projektu

2.3.3 Hibernate

Hibernate je nástroj pro Object-relation mapping (ORM) pro jazyk Java. Jeho hlavní úlohou je mapování Java tříd na databázové tabulky a naopak. Tento framework usnadňuje proces vývoje databázových aplikací poskytnutím API pro základní operace s tabulkami, čímž odstraňuje potřebu integrace SQL dotazů do aplikačního kódu. Díky Hibernate je umožněná abstrakce nad volbou konkrétního databázového zdroje, což zjednodušuje proces migrace mezi různými databázovými systémy.

Konfigurace mapování mezi Java třídou a databázovou tabulkou se dříve prováděla pomocí XML souboru. V dnešní době je preferovaným způsobem použití Java anotací, viz. výpis kódu 5. Uvedený příklad ukazuje deklaraci Java třídy *StudentEntity*, jejíž data se budou načítat a ukládat do tabulky *V_STUDENT* v databázi. Vazby mezi tabulkami lze implementovat použitím anotací *OneToOne*, *OneToMany* nebo *ManyToMany*. Hibernate podporuje odložené načítaní (z angl. lazy loading)[21], což znamená, že data jsou načteny z databáze až ve chvíli, kdy jsou vyžádány aplikací.

```
1  @Entity @Table(name = "V_STUDENT")
2  public class StudentEntity {
3      @Id @Column(name = "ID_STUDENT")
4      private Long studentId;
5      ...
6  }
```

Výpis kódu 5 Ukázka deklarace Java třídy pro Hibernate

I přes uvedené výhody a jednoduchý způsob konfigurace, je Hibernate poměrně složitý framework a jeho správné použití vyžaduje znalost interního fungování systému. Kromě toho, v některých případech nástroj generuje neefektivní SQL příkazy a „N+1 SELECT“ dotazy, které mohou vést k výkonnostním problémům softwaru. V kontextu aplikace pro vyplňování anketních lístků použití Hibernate frameworku vede k vytváření nadbytečných kusů kódu a konfigurací. Navíc nástroj využívá systémové prostředky pro správu interního kontextu. Pro uvedené případy užití aplikace, kde převažují operace čtení ze spojených pohledů nad jednoduchými dotazy, se použití Hibernate frameworku jeví jako nevýhoda. Ta může mít dopad na udržitelnost projektu, proto v rámci refactoringu backendu bude tento nástroj odstraněn z aplikace.

2.3.4 Spring Boot

Spring Boot široce používaný framework s otevřeným zdrojovým kódem, který je obzvlášť oblíbenou volbou pro vývoj mikrošlužeb a webových aplikací v ekosystému Java. Hlavní výhodou je jednoduchý proces vytvoření spustitelné aplikace s použitím minimálního množství konfigurací, které jsou potřebné pro rychlé dosažení funkčního prototypu. Spring Boot podporuje širokou škálu rozšíření, která usnadňují napojení aplikace na databázi, zabezpečení Rest rozhraní nebo sledování provozního stavu v produkčním prostředí. Navíc framework má rozsáhlou dokumentaci, velkou komunitu a tisíce vláken na internetových fórech. To usnadňuje řešení případných problémů s konfigurací nebo konzultaci osvědčených postupů.

V kontextu vývoje aplikace pro vyplňování anketních lístků použití Spring Boot frameworku přináší řadu výhod a urychluje proces implementace. Ovšem je potřeba zvážit všechny technologické aspekty projektu, jako jsou požadavky na výkon, bezpečnost, škálovatelnost a také v neposlední řadě zájem a zkušenost vývojářů na fakultě. Spring Boot je komplexní a složitý framework, kde řešení některých typů problémů vyžaduje hlubší znalost platformy a konceptů ekosystému Spring.

2.4 Autentizace a Autorizace

Proces autentizace a autorizace jsou dva odlišné pojmy v kontextu bezpečnosti v informačních technologiích. Autentizace je proces ověření identity uživatele, během kterého dotyčná osoba potvrdí, že je tím, kým tvrdí. Pro potvrzení identity osoba použije informaci, kterou disponuje pouze ona. Typicky se jedná o kombinaci uživatelského jména nebo emailu a hesla. Jinými moderními způsoby ověření identity je použití certifikátu nebo biometrického údaje, jako je například otisk prstu. Autentizace typicky probíhá na začátku interakce uživatele s chráněným systémem nebo zdrojem dat a slouží jako vstupní brána. Výsledkem procesu ověření identity je získaní „bezpečnostní vstupenky“ (z angl. ticket), která slouží pro autorizaci uživatelských požadavků během následující interakce. V kontextu webových aplikací jsou nejčastějším typem „bezpečnostní vstupenky“ *session cookie* a *token*. Jsou to buď náhodné unikátní textové řetězce nebo šifrované objekty v textové reprezentaci. Každá z technik reprezentace „bezpečnostní vstupenky“ má své výhody a nevýhody a musí se volit podle požadavků na webovou aplikaci.

Následujícím zmíněným pojmem byla autorizace. Jedná se o proces během kterého systém rozhoduje, zda ověřený uživatel má přístup k určitému zdroji nebo funkcím aplikace. Typicky se jedná o operace čtení, editace, vytváření nebo mazání dat. V mnoha systémech je proces autorizace založený na uživatelských rolích (například administrátor nebo uživatel), ke kterým jsou přiřazená specifická oprávnění.

V kontextu aplikace Anketa je příkladem autentizace přihlášení uživatele prostřednictvím *Single sign-on* a získaní JWT tokenu. Následný proces autorizace přiřazuje přihlášenému uživateli roli studenta nebo vyučujícího na základě poskytnuté hodnoty tokenu. Pokud uživatelský JWT token není validní, požadavek musí skončit s HTTP návratovým kódem *401 Unauthorized*. Pokud Rest API operace očekává odlišnou roli než má přihlášený uživatel, požadavek musí skončit s HTTP návratovým kódem *403 Forbidden*.

2.4.1 Single sign-on

Single sign-on (SSO) je koncept autentizačního mechanismu ve kterém aplikace ve společném ekosystému spoléhají na potvrzení identity uživatele jinou důvěryhodnou aplikací. Motivací k zavedení nového mechanismu autentizace se stala snaha zlepšit pohodlí uživatele, kde se eliminuje potřeba zakládat nový účet pro novou službu. Také při změně hesla stačí zadat nové údaje u poskytovatele účtu pouze jednou. V dnešní době společnosti jako Google, Facebook či Github poskytují SSO přihlášení pro tisíce webových aplikací v internetovém prostoru. Nicméně, použití SSO přináší bezpečnostní riziko, kde kompromitace přihlašovacích údajů může vést k přístupu neoprávněné osoby k řadě propojených aplikací nebo služeb.

Koncept SSO je široce používán v podnikovém a vysokoškolském prostředí. České vysoké učení technické využívá softwarové řešení s otevřeným zdrojovým kódem *Shibboleth*, které poskytuje podporu SSO v rámci univerzitních systémů. Samotný nástroj je postaven na standardu Security Assertion Markup Language (SAML) - protokolu pro komunikaci prostřednictvím zpráv ve formátu XML. V *Shibboleth* existují dvě klíčové role: poskytovatel identity (IdP) a poskytovatel služby (SP). Poskytovatelem identity je samotná univerzita a poskytovatelem služby je libovolná důvěryhodná zaregistrovaná aplikace, například studentský informační systém KOS nebo aplikace pro správu rozvrhu Timetable. Pro přihlášení prostřednictvím SSO je student nebo univerzitní pracovník přesměrován na stránku IdP *https://idp2.civ.cvut.cz*, kde dotyčný uvede své autentizační údaje. Po dokončení tohoto kroku je uživatel automaticky navrácen do aplikace. Poskytovatel služby od poskytovatele identity obdrží pouze nezbytné informace potřebné pro autorizaci osoby. V kontextu ČVUT se jedná o uživatelské jméno, které je unikátní napříč vysokou školou s vysokoškolskými systémy.

2.4.2 JSON Web token

Json web token (JWT) představuje kompaktní a bezpečný způsob pro výměnu informací mezi dvěma stranami. Ve webových aplikacích se JWT často používá pro autorizaci uživatelských požadavků. Dokument RFC 7519 [22] specifikuje strukturu a způsob použití zmíněné technologie.

Struktura JWT se skládá ze tří částí: hlavičky, obsahu a podpisu. Hlavička obsahuje metadata o tokenu - typicky se jedná o typ tokenu a použitý hašovací algoritmus. Existují dva typy tokenu: *JSON Web Signature* a *JSON Web Encryption*[22]. Rozdíl mezi nimi je v případech použití. *JSON Web Signature* se použije v případech, kdy je potřeba zajistit integritu a autentičnost dat v tokenu, zatímco *JSON Web Encryption* klade důraz na ochranu citlivých údajů uložených v tokenu. Podle volby typu tokenu lze zvolit správný kryptografický algoritmus nebo hašovací funkci dle požadavků na rychlost a zabezpečení. Další částí tokenu je jeho obsah. Obvykle se jedná o uživatelská data reprezentovaná textovým řetězcem. V programovacím jazyce Java lze použít již vestavěnou možnost serializace datových tříd pro konverzi uživatelských dat do textové podoby. Poslední částí tokenu je digitální podpis, který slouží pro ověření, zda data nebyla změněna třetí stranou (viz. man-in-the-middle attack)[23]. V případě použití kryptografického algoritmu je celý token zašifrován, a tak nelze přečíst jeho obsah bez příslušného dešifrovacího klíče. V HTTP komunikaci se používá *Base64* reprezentace tokenu pro jednodušší přenos speciálních textových znaků.

Velkou výhodou JWT je samostatnost, protože token obsahuje všechny potřebné informace pro autentizaci uživatele. Mezi nevýhody patří nemožnost zrušit vydaný token. Také je potřeba dbát na volbu silného hašovacího hesla, které má zabranit úniku citlivých informací uživatele.

2.5 Popis databáze

Na projektu se používá relační databázový systém Oracle Database od společnosti Oracle Corporation. Jedná se o komerční produkt s rozsáhlou podporou a velkou vývojářskou komunitou. První verze databáze vznikla v 70. letech minulého století. Následující verze přinášely nové možnosti a optimalizace stávajících funkcionalit. Databáze podporuje širokou škálu datových typů, včetně nestrukturovaných dat jako je XML nebo JSON. Systém lze provozovat na virtuálním stroji, v cloudu nebo hybridně. Výběr vhodného způsobu se odvíjí od požadavků na dostupnost databázového systému.

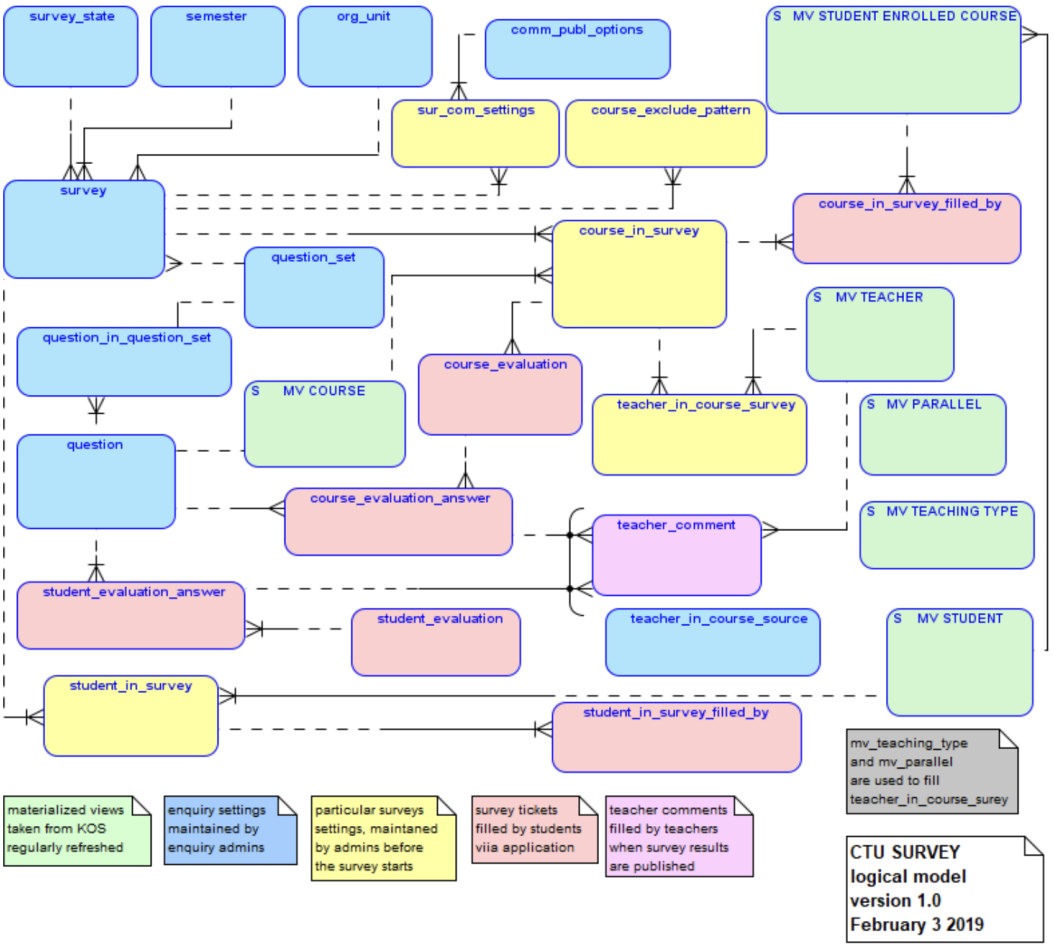
Databáze Oracle hraje důležitou roli v ekosystému Ankety. Jedná se o technologii, která propojuje tři aplikační komponenty a pomáhá strukturovaně ukládat velké množství dat. V kontextu vyplňování anketních lístků se využívá tří základních úložných prvků databázového systému Oracle pro organizaci uživatelských dat a řízení administrátorských procesů:

- **Tabulka:** Jde o základní konstrukci libovolného relačního databázového systému. Tabulka se skládá ze sloupců a sloupce mají specifikovaný datový typ a formát, například povolení *NULL* hodnoty. Specifickým sloupcem pro databázi Oracle je tzv. *ROWID*, který reprezentuje fyzickou adresu řádku v bloku dat a používá se pro urychlení procesu manipulace s uloženými daty a optimalizaci SQL dotazů[24]. Ovšem nejedná se o stabilní a dlouhodobý identifikátor, který by nahradil primární klíče v databázi.

- **Pohled:** Je to „virtuální“ tabulka, která byla vytvořená na základě SQL dotazu. Pohledy se často používají pro zjednodušení zobrazení komplexních dotazů nebo jako bezpečnostní omezení pro přístup k určitým datům. Ze své podstaty jsou pohledy neměnné, což znamená že v nich uložené informace nelze modifikovat. Zobrazená data jsou generovány v reálném čase z tabulek uvedených v dotazu. Tím pádem změna dat v jedné z dotazovaných tabulek ovlivní výsledek z pohledu.

- **Materializovaný pohled:** Jedná se o podobný koncept jako již zmíněné pohledy, s tím rozdílem, že data ze zdrojového SQL dotazu jsou skutečně uloženy do pevného disku databáze. Data v materializovaném pohledu nejsou aktualizovány v reálném čase, ale v pravidelných intervalech nebo na základě databázových událostí. Takovou událostí může být modifikace tabulky pohledového SQL dotazu. Tento koncept může výrazně urychlit časté uživatelské požadavky obsahující složité výpočty nebo spojení dat z více tabulek.

Zjednodušené databázové schéma pro aplikaci Anketa je zobrazeno na obrázku 2.3. Pro generování primárních klíčů v některých tabulkách se využívá tzv. *sekvencí*, které jsou atomické a postupně inkrementují čitatel na základě požadavku o nový identifikátor. Pro import dat ze studentského informačního systému a řízení administrátorských procesů se využívá databázových procedur.



Obrázek 2.3 Databázové schéma aplikace Anketa. Převzato z [2]

Kapitola 3

Analýza požadavků a volba technologií

Tato kapitola je zaměřena na rozbor zadání a analýzu změn v aplikaci Anketa, které vytvořil vedoucí této diplomové práce za účelem zlepšení udržitelnosti projektu. V následujících odstavcích jsou uvedeny jednotlivé požadavky na změny v backendové části vyplňování anketních lístků aplikace Anketa a analýza možného postupu řešení. Jsou zde také popsány existující technologie, které budou následně použité v rámci implementace změn. To má za cíl seznámit čtenáře s postupem refactoringu backendu a nastínit inovativní řešení. Uvedené požadavky byly nejdříve diskutovány s vedoucím diplomové práce pro stanovení postupu nasazení jednotlivých změn.

3.1 Shrnutí požadavků

Cílem práce je revize a refaktoring backendové komponenty vyplňování anketních lístků aplikace Anketa ČVUT. V rámci které dojde k automatizaci procesu kontroly kódu a promítnutí změn na aplikační a databázové úrovni ve zdrojových kódech. Spolu s vedoucím této diplomové práce byly stanoveny následující požadavky, jejichž účelem je integrace inovativních řešení problémů kontroly kvality kódu, automatického testování a zajištění dlouhodobé udržitelnosti projektu.

1. Návrh, diskuze a aplikace změn, které podpoří udržitelnost projektu.
2. Kontrola kvality nového kódu pomocí vhodného nástroje, například Sonar Qube.
3. Zavedení systematického přístup automatického hlášení chyb z backendu.
4. Změna backendové implementace v součinnosti s vedoucím práce tak, aby vyhodnocení anket nevytvářelo další materializované pohledy.
5. Návrh a implementace vhodných jednotkových a integračních testů, spustitelných v rámci CI/CD.
6. Návrh a implementace postupu anonymizace identifikátoru některých vyučujících, jejichž interní identifikátor odpovídá rodnému číslu.

Následující text detailně popisuje důležitost zmíněných problémů, postup jejich řešení a sadu pluginů, které pomáhají optimalizovat související procesy. Poslední požadavek je pouze implementačního charakteru, proto je jeho realizace zmíněna až v kapitole Implementace.

3.2 Podpora udržitelnosti projektu

Nejdříve je potřeba definovat pojem udržitelnost v kontextu vývoje softwaru. Udržitelný software je takový, který můžeme rozšiřovat nebo vylepšovat v průběhu času bez potřeby zásadních implementačních změn, zatímco zachovává svoji funkčnost a efektivitu. Mezi hlavní kvality udržitelného softwaru patří schopnost adaptace a rozšiřitelnost, bezpečnost, modularita, vysoká kvalita kódu a testovatelnost. Dodržováním výše uvedených principů lze vyvíjet software, který se může rychle adaptovat potřebám uživatelů.

Udržitelný software by měl být navržen a implementován tak, aby umožnil přidávání nových funkcionalit, bez potřeby většího refactoringu základu aplikace. To záleží především na správné volbě architektury a také použitých technologiích při implementaci aplikace. Také je potřeba dbát na prostředí ve kterém se software vyvíjí a bude udržován, protože špatná volba technologického základu povede k rychlé degradaci softwarového řešení.

Volba architektury projektu a technologií je závislá na mnoha faktorech. Je to často kompromis mezi schopnostmi programátora, a mezi tím, jaký výsledek očekává zadavatel. Pro tuto diplomovou práci byly klíčovými faktory nefunkční požadavky, případy užití, a také časové omezení. K nefunkčním požadavkům patří volba databáze Oracle, která omezuje programátora na technologie umožnujících integraci Oracle adaptéru. Z analýzy případů užití lze usoudit, že v aplikaci dominují operace čtení z databáze ve srovnání s operacemi zápisu. Refactoring backendu musí být proveden v časovém rozmezí několika měsíců. Také z pohledu rozvoje projektu do budoucna, není vhodné volit technologie, kterým se nevěnuje výuka na Fakultě informačních technologií.

Na základě diskuze s vedoucím závěrečné práce bylo rozhodnuto, že se projekt bude držet Java technologií. Zároveň bude zaveden nový programovací jazyk Kotlin, pro sestavu projektu se použije Gradle a aplikačním frameworkem zůstane Spring. Vybranou kombinací technologického základu lze zanechat atraktivitu projektu pro její budoucí rozvoj na fakultě a ušetřit čas při zaškolení nových vývojářů. O výše zmíněných technologiích projednávají následující podkapitoly.

3.2.1 Kotlin

Kotlin je programovací jazyk, který byl vyvinut a vydán společností JetBrains v roce 2011 a později v roce 2017 přijat společností Google jako oficiální jazyk pro vývoj Android aplikací. Dnes je Kotlin moderní, široce podporovaný a efektivní jazyk, vhodný pro vývoj jak serverových, tak i mobilních aplikací.

Hlavním důvodem vzniku Kotlinu je zastaralost jazyku Java, který není dost flexibilní, efektivní a udržitelný. Oproti Javě, Kotlin do světa programovaní přináší řadu výhod:

- **Java kompatibilita:** Kotlin je plně kompatibilní s programovacím jazykem Java, což ve výsledku umožňuje jeho snadnou integraci do již existujících Java projektů. Zároveň přepis aplikace do Kotlinu nemá dopad na výkon, protože stejně jako Java je kompilován do bytecodu spustitelného na JVM.
- **Null safety:** Jedná se o jednu z největších výhod Kotlinu, kterou přináší do Java světa bezpečnostní mechanismy proti nulovým odkazům. Motivací k implementaci v programovacím jazyku daného bezpečnostního prvku se stala událost „chyba za miliardu dolarů“ (z angl. The billion dollar mistake)[25]. Jedním z takových mechanismů je zavedení systému typů, který rozlišuje mezi referencemi, zda mohou být nulové nebo nikoliv. Kontrola nullability referencí probíhá během procesu kompilace kódu.

- **Produktivita:** Menší množství zdrojového kódu dělá aplikace v Kotlinu přehlednější a časově méně náročné na údržbu do budoucna. Zároveň díky rozšířeným funkcím (z angl. extension function)[26], datovým třídám, sealed hierarchii a širokému výběru metod ze standardních knihoven je proces vývoje pohodlnější a výsledný kód kompaktnější.
- **Podpora coroutin:** Velkou výhodou jazyka Kotlin je nativní podpora coroutin. Jedná se o efektivní nástroj pro zavedení asynchronních operací na principu lehkých vláken. Na rozdíl od tradičních Java vláken, coroutiny vyžadují menší množství systémových zdrojů pro jejich vytváření a správu. Díky coroutinám lze v Kotlinu provádět „pomalé operace“, jako je volání http služby nebo dotazování databázové služby, bez blokování hlavního aplikačního vlákna. Navíc coroutiny v Kotlinu poskytují jednoduchou a čitelnou syntaxi, která umožňuje snadné zapouzdření asynchronního kódu do aplikace. [27]
- **Komunita:** Kolem Kotlinu vznikla rapidně rostoucí komunita vývojářů, která se aktivně podílí na zavedení nových funkcí a rozšíření pomocných knihoven. K nejpopulárnějším knihovnám pro Kotlin patří například Ktor (serverový framework), Kotest (testovací framework), Exposed (ORM framework) či Arrow (knihovna pro funkcionální programování). Některé z uvedených nástrojů budou použity při refactoringu vyplňování anketních lístků aplikace Anketa.

Vzhledem v výhodám, které byly popsány výše, se zavedení Kotlinu v aplikaci Anketa jeví jako ideální řešení pro implementaci změn a podporu udržitelnosti projektu.

3.2.2 Gradle

Gradle je nástroj s otevřeným zdrojovým kódem pro zprávu projektových závislostí a automatické sestavování projektů. Je známý díky své flexibilitě, široké sadě pomocných pluginů a možností konfigurace pomocí jazyka Groovy nebo Kotlin Domain specific language (DSL). Gradle kombinuje nejlepší vlastnosti svých předchůdců Apache Ant a Apache Maven. Odstranění Maven a zavedení Gradle do projektu by mohlo přinést řadu výhod:

- **Rychlejší sestavení:** Gradle využívá inkrementální kompilace, a také inkrementálního sestavení, což znamená, že sleduje změny na projektu a sestavuje pouze ty změněné části a další na nich závislé. Tím lze dosáhnout výrazného zrychlení procesu sestavení projektu, což zvyšuje produktivitu vývojáře. Navíc Gradle umí chytře ukládat projektové závislosti do své cache, čímž se zamezí opakovanému stahování knihoven třetích stran z externích repozitářů.
- **Flexibilita:** Díky podpoře jazyků Groovy a Kotlin DSL, poskytuje Gradle větší flexibilitu a dynamiku pro konfiguraci složitějších projektů. Navíc uvedené jazyky umožňují integraci logických procesu do životního cyklu projektu, což je vhodné pro složité a atypické scénáře sestavení. Bohužel ale v nástroji Maven se stále používají konfigurační XML soubory, které nepřinášejí potřebnou flexibilitu.
- **Podpora pro vícejazyčné projekty:** Díky rozsáhlému systému pluginů je Gradle schopen podporovat velké množství programovacích jazyků. Mezi ty nejpopulárnější patří Java, Kotlin, Groovy, Scala a C/C++. Uvedené jazyky lze kombinovat v rámci jednoho projektu, což přináší vývojářům větší flexibilitu během procesu implementace. Například v kontextu vývoje aplikace pro vyplňování anketních lístků lze souběžně vyvíjet v jazycích Java a Kotlin bez omezení.

Výše uvedené výhody dělají Gradle ideálním nástrojem na sestavení projektu Anketa. Navíc konfigurace projektu je výrazně jednodušší než u Maven. Základní jednotkou řízení procesu sestavení je úkol (z ang. task) a základním konfiguračním souborem je *build.gradle*, který je umístěn do kořenové složky projektu a jednotlivých modulů. V něm se definují projektové závislosti, externí pluginy a proces sestavení, viz. výpis kódu 6.

```
1   group 'cz.cvut.fit'
2   version '1.0.0'
3   name 'anketa-backend'
4
5   buildscript {
6       repositories {
7           gradlePluginPortal()
8           ...
9       }
10      dependencies {
11          classpath "org.openapitools:openapi-generator-gradle-plugin:7.0.1"
12          ...
13      }
14  }
```

Výpis kódu 6 Ukázka souboru build.gradle

Sestavení projektu se podřizuje životnímu cyklu Gradlu. Jsou to tři fáze, kterými projde projekt od začátku spuštění procesu sestavení až do jeho konce:

- **Inicializace:** Zde Gradle detekuje, načte a vyhodnotí konfigurační soubory, aby zjistil, které části projektu jsou součástí procesu sestavení. Potom se pro každou část vytvoří instance *Project* (datová obálka pro metadata části projektu).
- **Konfigurace:** Pro vybrané projektové části z předchozí fáze proběhne detekce úkolů a jejich zařazení do plánu vykonání (z angl. task graph)[28].
- **Vykonání:** V této fázi jsou jednotlivé úkoly spuštěny podle sestaveného plánu. Některé z nich mohou být vykonány paralelně pro urychlení procesu sestavení projektu.

Gradle nabízí možnost implementace vlastních úkolů, což přináší do projektu flexibilitu a schopnost se uzpůsobit specifickým potřebám.

3.2.3 Spring a Spring WebFlux

V dnešním technologickém světě existuje velké množství nástrojů a frameworků pro vývoj HTTP služeb. Díky plné kompatibilitě Kotlinu s Javou lze volit v podstatě libovolný Java framework pro tvorbu a správu webových služeb. Ovšem existují i některé frameworky, vytvořené přímo v Kotlinu, které dokážou poskytovat přístup ke specifickým funkcím jako couroutiny. Mezi nejpopulárnější volby patří Ktor, Kweb, Spring Boot, Micronaut, Javalin či Spark. Klíčovými faktory, které ovlivnily volbu webového frameworku staly:

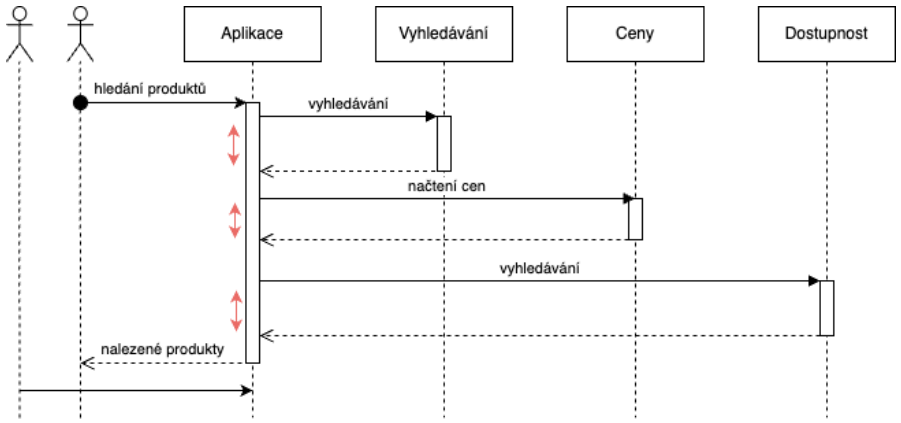
1. dobrá kompatibilita s Kotlinem a případně i s Javou
2. možnost napojení na Oracle databázi
3. jednoduchá integrace autorizace a autentizace
4. rozsáhlá dokumentace
5. atraktivita pro budoucí vývojáře

S ohledem na uvedené faktory se použití webového frameworku Spring WebFlux jeví jako optimální řešení. Je potřeba rozlišovat dvě odlišné technologie: Spring a Spring WebFlux. První z uvedených je kontejnerová technologie využívající IoC principu. IoC zde znamená inverzní ovládání (z angl. Inversion of control (IoC))[29], což je princip v softwarovém inženýrství, který se používá k přeneseni zodpovědnosti za řízení závislostí z vlastního kódu do externího kontejneru. To umožňuje programátorovi se zaměřit na logiku aplikace. Ve Springu se k dodržení IoC principu používá vkládání závislostí (z angl. deependency injection)[30], což znamená, že kontejner přidává externí závislosti do komponenty místo toho, aby to komponenta řešila sama. Tím se snižuje počet instancí komponent v kontextu aplikace a také zvyšuje testovatelnost systému. Princip vkládání závislostí se stal možný díky zavedení anotací, reflexe[31] a implementaci návrhového vzoru proxy v Javě.

Základní jednotkou ve Springu je „bean“, nebo jinými slovy instance komponenty, kterou framework vytvořil a spravuje. Ve výchozím nastavení jsou beany singleton, což znamená, že se v kontextu aplikace vytvoří pouze jedna instance komponenty za celý životní cyklus kontejneru. Proces vytváření nových bean si může programátor nadefinovat sám podle potřeb projektu.

Druhá zmíněná technologie je Spring WebFlux. Jedná se o nadstavbu v rámci Springu 5, která poskytuje podporu pro tvorbu reaktivních aplikací. Spring WebFlux je postaven na projektu Reactor, který mu umožňuje efektivnější zpracování paralelních požadavků s menším množstvím hardwarových zdrojů díky asynchronním a neblokujícím I/O operacím.

Výhodu výše uvedeného principu lze popsat na následujícím příkladu. Představme si typickou webovou aplikaci ve frameworku Spring Boot, která zpracovává uživatelské HTTP požadavky a typickým dotazem je vyhledávání produktů podle názvu. Ten může kombinovat data z full-textového vyhledávače, aktuální ceny z databáze a dostupnosti produktů z externího zdroje. Tím pádem pro zobrazení dat uživateli aplikace provede a zpracuje tři požadavky do externích systémů. Ve Spring Boot, který je postaven na *thread-per-request*[32] modelu, jsou tyto interakce mezi aplikací a externími službami blokující operace. Což v praxi znamená, že vlákno většinu času čeká na načtení dat z externích systémů a nemůže jej využit další uživatel webové aplikace. Taková situace je zobrazená na sekvenčním diagramu volání, viz. obrázek 3.1. Každé aplikační vlákno zpracovává pouze jeden uživatelský požadavek, což je velmi neefektivní.

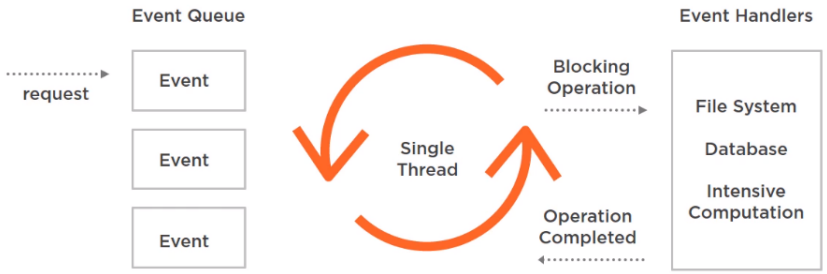


Obrázek 3.1 Sekvenční diagram volání webové služby s blokujícím zpracováním požadavků

S rostoucím počtem uživatelů webové platformy můžeme škálovat aplikaci do šířky přidáváním nových instancí. Ovšem to neřeší problém čekání vláken, které neustále konzumují systémové prostředky. Vytváření nových vláken pro souběžné provádění blokujících operací také není efektivní řešení, protože nativní Java vlákna jsou nákladné pro spouštění a ukončení. Proto tradiční model souběžného zpracování požadavků postupně ustupuje novému konceptu reaktivního (nebo asynchronního) programování[33].

Klíčovou myšlenkou nového principu je implementace neblokujících operací, které mohou být spouštěny nezávisle na sobě. Zde jedno fyzické vlákno může obsluhovat více požadavků téměř současně, což v důsledku může snížit spotřebu systémových zdrojů a zlepšit škálovatelnost aplikace. Místo psaní kódu, který se provádí krok za krokem, programátor definuje atomické reakce na změny v datovém toku nebo v řízení logiky aplikace. Jsou to jednotlivé signály/události, pomocí kterých části aplikace asynchronně komunikují mezi sebou. Pokud zpracování jednoho signálu vyžaduje určitou čekací dobu, vlákno nezůstane zablokované a pustí se do dokončení jiné akce. Samozřejmě přepínaní kontextu mezi jednotlivými operacemi nebude zadarmo a vyžaduje procesorový čas a systémové prostředky, ale takto vlákno nebude čekat bez využití.

Spring WebFlux využívá modelu *event loop*[33], kde hlavní smyčka neustále čeká a reaguje na jednotlivé události, viz. obrázek 3.2. Zde je vidět, že místo vytvoření nového vlákna na požadavek, server ho zaregistruje a zařadí do fronty podle priority. Smyčka události sleduje všechny požadavky a postupně je odbavuje pomocí zpětného volání (z angl. callback). Blokující operace nezastavují smyčku a jsou odbavovány asynchronně. Tento model lze jednoduše a efektivně škálovat přidáním dalších vláken.



Obrázek 3.2 Model neblokujícího zpracování požadavků

Hlavními případy užití aplikace Anketa je čtení dat z databáze, kde tyto read operace mají blokující a netransakční povahu. Během zkouškového období aplikace čelí větší návštěvnosti ze strany studentů, což často ovlivňuje čas odezvy systému. Implementace nového a efektivnějšího principu pro zpracovaní požadavků by mohla mít pozitivní dopad na běh Ankety v novém semestru.

3.3 Kontrola kvality kódu

Kontrola kvality kódu je z části objektivní, a z části subjektivní posudek softwarového kódu. Jeho cílem je ověření plnění stanovených standardů z ohledem na moderní přístup a nejlepší praktiky. Je to nezbytný proces, který musí být zařazen do vývoje libovolného softwarového produktu z několika zásadních důvodů:

- **Odhalení chyb:** Revize kódu z několika úhlů pohledu pomáhá identifikovat potenciální problémy a vyhnout se jim v produkčním prostředí, kde oprava může stát dost finančních prostředků.
- **Podpora udržitelnosti:** Kvalitní kód je snadnější pro pochopení, údržbu a opravu, což hraje klíčovou roli v kontextu dlouhodobého rozvoje softwarového produktu.
- **Zabezpečení a výkon:** Kontrolou kódu lze odhalit jeho slabá místa a tak předcházet možným útokům v budoucnu nebo neefektivnímu použití hardwarových zdrojů.
- **Vzdělávání:** Méně zkušení vývojáři mohou během procesu kontroly kvality kódu získávat znalosti od zkušenějších členů vývojového týmu, což jim umožňuje osvojit si nové postupy a praktiky.

Kvalitní kód nelze jednoznačně definovat, protože se pohled dvou programátorů může v řadě ohledů odlišovat. Ale v rámci revize kódu musí tito dva lide dojít ke shodě a také opravit nalezené nedostatky. Pro zvýšení efektivity a urychlení procesu kontroly kódu lze využít několika nástrojů pro automatizaci formátování, testování nebo statickou analýzu zdrojových souborů. Tyto nástroje pomáhají odhalit chyby nebo slabá místa v kódu, která nejsou vidět na první pohled. Během refactoringu backendu aplikace vyplňování anketních lístků došlo k integraci několika pomůcek, které přispívají k dodržování kvality kódu.

3.3.1 Kotlinter

Tento nástroj se především zaměřuje na kódový styl a formátovaní zdrojových souborů podle standardů Kotlin Coding Conventions[34] a detekci nepoužívaných částí. Kotlinter podporuje automatickou opravu některých běžných problémů. Nástroj lze jednoduše integrovat do existujícího projektu díky Gradle pluginu, viz. výpis kódu 7. Díky podpoře paralelního běhu kontrola kvality kódu může být spouštěna souběžně na několika souborech, což výrazně zrychluje proces sestavení projektu. Kotlinter je cenným nástrojem pro vývojáře, kteří chtějí zajistit čistotu, dobrou strukturu a dodržení osvědčených postupů psaní kódu v Kotlinu.

```
1  plugins {
2      id("org.jmailen.kotlinter") version "4.0.0"
3  }
4  kotlinter {
5      reporters = ["checkstyle", "html", "json", "plain"]
6  }
```

Výpis kódu 7 Ukázka konfigurace nástroje Kotlinter v souboru build.gradle

3.3.2 Detekt

Nástroj Detekt, pro statickou analýzu zdrojových kódů, se hodně podobá již zmíněnému nástroji Kotlinter. Kromě kontroly dodržení Kotlin Coding Conventions Detekt nabízí řadu funkcí navíc:

- **Kontrola komplexnosti:** Detekt prochází kód a identifikuje jeho části, které jsou příliš náročné pro pochopení. Většinou se jedná o moc dlouhé metody nebo nadměrné používání vnořených struktur, smyček a větvení, což zvyšuje cyklomatickou složitost (z angl. cyclomatic complexity)[35] aplikačního kódu. Tento pojem vyjadřuje složitost programu počtem nezávislých cest skrz zdrojový kód.
- **Dodržovaní stylu kódu:** Nástroj provádí analýzu zdrojových souborů, zda splňují stanovený styl kódu. Jedná se například o počet vstupních argumentů v konstruktoru třídy, počet metod ve třídě, délka metody nebo dokonce počet symbolů na jednom řádku.
- **Konfigurovatelnost:** Detekt používá soubory ve formátu YAML nebo JSON pro konfiguraci pravidel validace kódu, výjimek, procesu sestavení projektu a reportů. Rozsah nastavení nástroje je mnohem větší a flexibilnější než u Kotlinteru.
- **Rozšiřitelnost:** Detekt umožňuje rozšíření již existujících pravidel o vlastní implementaci, a proto nabízí odpovídající Kotlin rozhraní. Nové pravidlo musí dědit rozhraní od předchůdce *Rule*, implementovat logiku pro analýzu zdrojového kódu a poskytovat informaci o nalezených nedostatcích.

Integrace Detektu na projektu je dost jednoduchá díky existujícímu Gradle pluginu, viz. výpis kódu 8. V rámci refactoringu backendu aplikace Anketa bylo využito integrace Detektu přímo do kompilátoru, čímž se snížil čas sestavení projektu.

```
1  apply plugin: "io.github.detekt.gradle.compiler-plugin"
2  detekt {
3      buildUponDefaultConfig = true
4      config.setFrom("$rootDir/config/detekt/detekt.yml")
5      enableCompilerPlugin = true
6      debug = true
7  }
```

Výpis kódu 8 Ukázka konfigurace Detektu v souboru build.gradle

3.3.3 OWASP Dependency-Check

Často webové aplikace využívají integraci knihoven třetích stran nebo dokonce i celé frameworky pro urychlení procesu vývoje a delegaci zodpovědnosti za specifické funkcionality. Přestože tento přístup může výrazně šetřit čas během implementace projektu, nese sebou velkou nevýhodu, a to akceptaci potenciálních bezpečnostních rizik. Kontrola zranitelnosti u samotné aplikace a dalších aplikačních závislostí je zásadní z důvodu ochrany osobních dat, finančních dopadů a podpory důvěryhodnosti i reputace společnosti.

Riziková místa v aplikacích se dají odhalit několika způsoby, mezi které patří penetrační testování, kvalitní revize zdrojového kódu nebo použití pomocných nástrojů jako OWASP Dependency-Check. Jde o plugin do Gradlu, který skenuje projektové závislosti a detekuje jejich možné zranitelnosti metodou porovnání informací z otevřených zdrojů. Existuje řada veřejně známých databází poskytujících detailní informace o nalezených bezpečnostních chybách.

- **CVEDetails:** Je to webová služba, která poskytuje informace o nalezených a veřejně známých zranitelnostech a expozicích. Tento web je zdrojem užitečných informací o historii a specifice softwarové slabiny. Každá bezpečnostní chyba je zde označena unikátním identifikátorem CVE-yyyy-xxxxxx, kde CVE je zkratkou pro Common Vulnerabilities and Exposures. Jedná se o standard poskytování informací o zranitelnostech, a také jejich hodnocení.
- **NVD:** Jedná se o zkratku pro National Vulnerability Database, vládní databázi ve Spojených státech obsahující standardizované informace o známých zranitelnostech softwaru. Tento zdroj poskytuje CVSS (Common Vulnerability Scoring System) standard pro kvalifikaci a hodnocení závažnosti zranitelnosti.
- **Exploit Database:** Je to veřejně známá databáze škodlivých kusů kódů nebo postupů, kterými lze ověřit zabezpečení vlastního softwaru. Ovšem neoprávněné použití těchto exploitů vůči cizímu softwaru může nést těžké právní následky.

Kromě samotného pluginu společnost OWASP poskytuje i tzv. „checklist“. Jde o sadu doporučení a nejlepších praktik pro zabezpečení webové aplikace. Ty mají za cíl naučit vývojáře, jak předcházet a identifikovat bezpečnostní slabiny aplikace. [36]

Pro integraci pluginu do projektu je zapotřebí provést několik jednoduchých úprav v souboru *build.gradle* (viz. výpis kódu 9) a potom spustit úlohu *dependencyCheckAnalyze* při sestavení aplikace.

```
1  buildscript {
2     dependencies {
3        classpath("org.owasp:dependency-check-gradle:8.4.2")
4     }
5  }
6  plugin: "org.owasp.dependencycheck"
```

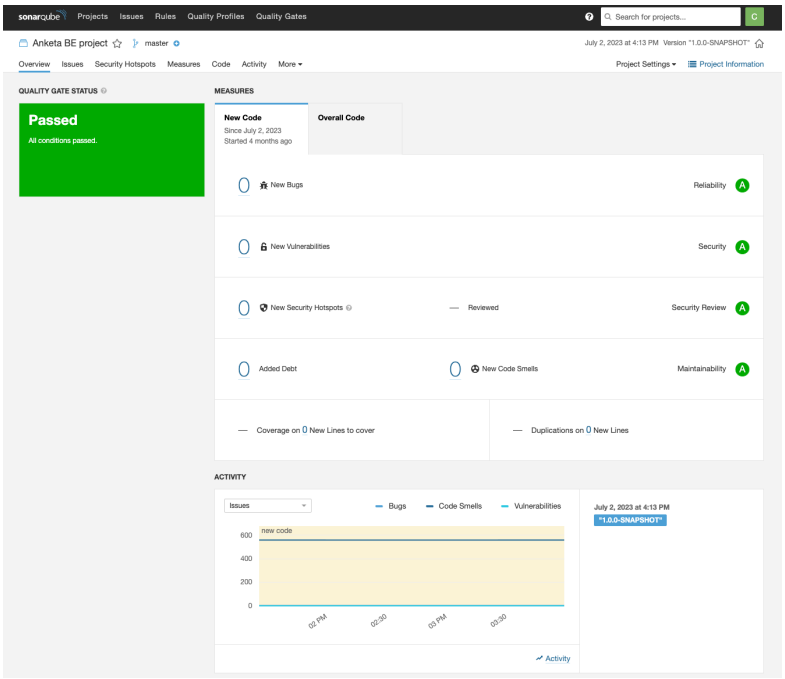
Výpis kódu 9 Ukázka konfigurace OWASP Dependency-Check v souboru build.gradle

3.3.4 SonarQube

SonarQube je platforma s otevřeným zdrojovým kódem pro statickou analýzu zdrojového kódu aplikace. Hlavním cílem integrace takového nástroje je zajištění dlouhodobé udržitelnosti projektu, a také nastavení standardů pro bezpečný vývoj. Platforma poskytuje detailní informace o potencionálních chybách, duplicitách v kódu, testovém pokrytí a zranitelnostech v závislostech. I přesto, že podobné funkcionality na projektu nabízí již zmíněné pluginy, SonarQube přináší řadu dalších užitečných benefitů:

- **Přehledný dashboard:** SonarQube hned na úvodní stránce nabízí přehledný dashboard s projektovými metrikami a grafem aktivit. Vývojář zde může vidět, zda nový kód splňuje nastavené standardy kvality, a také si může prohlédnout celkové hodnocení projektu. Pro jednoduchost se používá škála písmen hodnocení metrik od A do F. V záložce *Quality Gates* lze nastavit vlastní kritéria pro posouzení kvality aplikačního kódu, například počet duplicit nebo testové pokrytí. Dohromady taková kritéria tvoří sadu pravidel, jež musí kód splňovat, aby byl považován za kvalitní a mohl být nasazen do produkčního prostředí.
- **Sledování verzí projektu:** Jedná se o zásadní funkcionalitu v kontextu dlouhodobé údržby aplikace, díky níž je vývojář schopen sledovat změny projektových metrik v dlouhodobém horizontu, viz. graf aktivit na obrázku 3.3. Kromě toho v záložce *Aktivity* lze najít detailní přehled projektových změn podle verze, data přidání nebo typu událostí.

- **Podpora mnoha jazyků:** Tento nástroj, na rozdíl od již zmíněných, je zaměřený na širokou škálu programovacích jazyků včetně skriptovacích jako Python nebo značkovacích jako XML. Tím pádem se vývojář nemusí omezovat pouze na jeden v rámci projektu a využít ten nejvhodnější pro specifický případ užití.
- **Integrace do CI/CD:** SonarQube lze integrovat do projektových CI/CD pipeline, což umožní kontinuální sledování kvality kódu v procesu vývoje. Spouštění analýzy před každým *commitem* není časově efektivní, protože může trvat několik desítek minut. Proto v kontextu aplikace Anketa lze přidat splnění kriterií ze SonaQubu jako podmínku akceptace *merge requestu*. Samotná analýza by probíhala v rámci pipeline, tím pádem by tento proces nezdržoval vývoj.



Obrázek 3.3 Domovská stránka projektu v SonarQube

3.3.5 Shrnutí

Na základě předchozího textu je možné dojít k mylnému závěru, že uvedené nástroje nabízí stejnou nebo podobnou sadu funkcionalit. Ovšem není tomu tak, protože každá ze zmíněných technologií má vlastní vyhrazenou zónu zodpovědnosti a jejich kombinace by měla výrazně zlepšit kvalitu kódu do budoucna.

1. **Kotlinter -** formátování kódu a dodržení Kotlin Coding Conventions před *commitem*.
2. **Detekt -** analýza komplexnosti a dodržení stanoveného stylu kódu před *commitem*.
3. **OWASP Dependency-Check -** analýza zranitelností projektových závislostí v rámci CI/CD pipeline.
4. **SonarQube -** analýza testového pokrytí, duplicit a potencionálních chyb v rámci CI/CD pipeline před akceptací změn ve zdrojovém kódu.

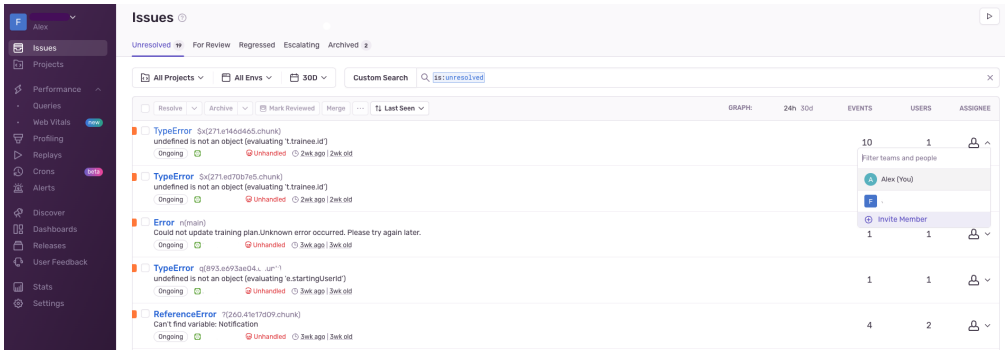
3.4 Automatické hlášení chyb systému

Automatické hlášení chyb je důležitou součástí v moderním procesu vývoje a údržby softwaru v produkčním prostředí, které přispívá k jeho efektivnější správě a monitoringu. V kontextu aplikace Anketa je automatické hlášení chyb zásadní funkcionalitou, protože projekt nemá stálého maintainera, který by pravidelně kontroloval běh aplikace a opravoval nalezené nedostatky. Proto by integrace nástroje poskytujícího uvedenou funkcionalitu přinesla projektu několik výhod.

- **Automatická identifikace problémů:** Chyby v aplikaci jsou identifikovány a automaticky reportovány do externího systému podle nastavené konfigurace. Takový moderní přístup umožňuje okamžitou reakci na problém. Zároveň aplikační logy pak nesou pouze informační hodnotu a nejsou používány jako primární zdroj informace o stavu aplikace.
- **Detailnější přehled chyb:** Nástroje pro automatické hlášení chyb umožňují řazení a třídění identifikovaných problémů podle široké škály parametrů, čímž usnadňují proces nastavení priority jednotlivých požadavků.
- **Snadnější napodobení:** Častým problémem v opravě chyb softwaru je jejich napodobení vývojářem ve vlastním prostředí, které se často může rapidně odlišovat od uživatelského. Proto nástroje pro automatické hlášení chyb, kromě samotného problému, reportují informace o uživatelském rozhraní, stavu aplikace a konfiguraci operačního systému, což je velice nápomocné při diagnostice a opravě chyb.
- **Automatizace managementu:** Nástroje pro automatickou detekci chyb umožňují nastavení emailových notifikací o problémech na konkrétní vývojový tým podle typu události, což pomáhá směřovat informace správnou cestou. Zároveň umí vytvářet úlohy do externích nástrojů pro správu a sledování projektů, například Jira, což ve výsledku redukuje práci projektovému managementu.
- **Zlepšení uživatelské zpětné vazby:** Rychlá reakce na problém může vést ke zlepšení spokojenosti uživatelů aplikace.

3.4.1 Sentry

Existuje řada řešení, která se nabízí různými metodami, platforma jako služba (PaaS) nebo software jako služba (SaaS) v závislosti na konkretních požadavcích zákazníka. Jedním z populárních řešení je platforma Sentry, viz. obrázek 3.4, která se buď dá provozovat na vlastních serverech nebo používat jako cloudová služba. V druhém případě lze volit mezi neplacenou a placenou licencí, kde si za příplatek zákazník může dokoupit funkcionality navíc.



Obrázek 3.4 Domovská stránka projektu v Sentry

V kontextu této diplomové práce a refactoringu backendu aplikace Anketa lze použít neplacenou licenci služby Sentry, ale pro provoz v produkčním prostředí může fakulta využít možnosti instalace vlastní instance na virtuálním stroji. V případě použití Sentry je možné chyby ze všech tří komponent Ankety ukládat do centrálního úložiště, díky možnosti rozlišování typu prostředí. Navíc pomocí speciálního JS modulu lze odchytávat chyby ve frontendových aplikacích. Kromě automatického reportu identifikovaných problémů, neplacená licence Sentry nabízí monitoring pomalých http požadavků, sledování průběhu SQL dotazů a metriky kvalitativního hodnocení webových aplikací. Také se v beta verzi softwaru nabízí možnost sledování naplánovaných úloh, kde je zásadním parametrem nejen průběh a výsledek, ale také celková doba trvání operace. Pro integraci Sentry do projektu je potřeba přidat dvě aplikační závislosti, viz. výpis kódu 10, a také nastavit Data source name (DSN) v konfiguračním souboru aplikace, viz. výpis kódu 11.

```
1  dependencies {
2     implementation("io.sentry:sentry-logback:6.33.1")
3     implementation("io.sentry:sentry-spring-boot-starter-jakarta:6.33.1")
4  }
```

Výpis kódu 10 Přidání Sentry závislosti v souboru build.gradle

```
1  sentry:
2     dsn: https://xxx.ingest.sentry.io/xxx
3     traces-sample-rate: 1.0
4     environment: production
```

Výpis kódu 11 Ukázka konfigurace Sentry DSN

3.5 Odstranění materializovaných pohledů

Popisu materializovaných pohledů a jejich roli v provozu aplikace Anketa se věnuje část kapitoly Existující řešení. S každým novým během univerzitní ankety postupně přibývají nové pohledy a tím se postupně zahlcuje databázový stroj. Ovšem to není jediný nedostatek takového přístupu, těch problémů je více a některé z nich mají kritický vliv na běh aplikace.

- **Bezpečnost dat:** Stávající přístup k implementaci SQL dotazů může mít kritický vliv na bezpečnost aplikačních dat, protože může být ohrožený *SQL injekcí*[37], viz. složený Java dotaz ve výpisu kódu 12. Zde je kritickým bodem přímé vkládaní hodnot parametrů *surveyId* a *semesterCode* do SQL dotazu. Tyto parametry nejsou nijak ošetřeny, protože jsou nastavovány přímo z http požadavku při volání aplikačního endpointu. Útočník by tak mohl do svého požadavku vložit škodlivý kód, který by mohl změnit chování SQL dotazu a vést k neoprávněnému přístupu k uživatelským údajům. Z ohledem na zkušenosti univerzity s kybernetickými útoky či úniky dat, musí být tento přístup nahrazen za spolehlivý a bezpečný.
- **Správa schématu:** Vytváření nových tabulek nebo pohledů při každém běhu Ankety rychle vede k nespravovatelnému počtu objektů v databázi, což komplikuje správu schématu a udržování vazeb mezi tabulkami. Nyní databáze obsahuje několik stovek materializovaných pohledů. Každý pohled obsahuje „malé množství dat“ - jednotky až stovky záznamů. Takové rozptýlení dat mezi pohledy porušuje princip normalizace a navíc vede ke zpomalení operací zálohování nebo obnovy databázového stroje.
- **Údržba aplikace:** Implementace nových případů užití může být velice obtížná, pokud jsou data rozložena do mnoha databázových objektů. Navíc to může vést k problémům se škálováním databáze při zvyšování počtu aktivních uživatelů a objemu uložených dat. Při správném návrhu by aplikace neměla dynamicky přistupovat nebo vytvářet tabulky, což často vede k nekonzistenčnímu stavu databázového stroje v průběhu transakce. Údržba takového zdrojového kódu je programátorsky a ekonomicky náročná.

```
1  Query query = em.createNativeQuery(
2    "SELECT * FROM RES_CUR_COM CC\n" +
3    "JOIN RES_TEACH_SUM_" + surveyId + "_" + semesterCode + " TEACHER\n" +
4    "ON TEACHER.ID_TEACHER = CC.ID_TEACHER\n" +
5    "WHERE CC.SEMESTER_CODE = ? AND CC.ID_TEACHER = ? AND CC.SURVEY_KEY = ?"
6  );
```

Výpis kódu 12 Příklad vytváření SQL dotazu ve stávajícím řešení

Nejjednodušším a časově nejefektivnějším řešením pro odstranění materializovaných pohledů je zavedení dvou nových textových sloupců *SURVEY_KEY* a *SEMESTER_CODE* ve společné tabulce pro data ze skupiny pohledů. Údaje z materializovaných pohledů se společným názvem *RES_CUR_EVAL_K_CCCC*, kde *K* značí anketní klíč a *CCCC* číslo semestru, se dají importovat do jedné tabulky *RES_CUR_EVAL*. Struktura sloupců společných pohledů se v průběhu několika let nezměnila, což umožní následující datovou transformaci. Pro efektivnější vyhledávání ve větším množství dat mohou dva nové sloupce *SURVEY_KEY* a *SEMESTER_CODE* tvořit kompozitní index.

Popsaným postupem lze odstranit bezpečnostní riziko *SQL injekce* v aplikaci a dosáhnout lepší udržitelnosti zdrojových kódů a databázového schématu. V rámci implementace je potřeba provést refactoring všech existujících SQL dotazů nad materializovanými pohledy a posunout dříve zmíněné parametry *surveyId* a *semesterCode* do *WHERE* části databázových požadavků.

3.6 Automatické testování v rámci CI/CD

Testování vytvořeného softwaru je ověření, zda aplikace nebo systém splňuje uživatelské a technické požadavky, které byly stanoveny v rámci zadání. Hlavním cílem testování je zajištění kvality, zjištění míry uživatelské spokojenosti, identifikace případných chyb a prevence kritických rizik provozu aplikace v produkčním prostředí. Existuje mnoho různých typů testů, kde každý má své specifické zaměření a účel. V kontextu této diplomové práce budou zmíněny tři druhy testů:

- **Jednotkové testy:** Jde o testy, které slouží k ověření „malého kousku“ aplikačního kódu, typicky jde o metodu nebo třídu. Jednotkové testy by neměly obsahovat externí závislosti a komponenty mimo kontext testu jsou obvykle napodobeny (mocked).
- **Integrační testy:** Tyto testy jsou zaměřeny na ověření chování několika komponent aplikace propojených dohromady. Tento druh testování obvykle vyžaduje složitější přípravu, instanci databázového úložiště a předem připravená data.
- **Smoke testy:** Tento druh testů obvykle slouží ke kontrole aplikace před nasazením do produkčního prostředí. Jde o souhrn jak jednotkových, tak i integračních testů, které lze pouštět v rámci CI/CD během sestavení aplikace.

Výše uvedené druhy testů lze začlenit do procesu vývoje aplikace prostřednictvím tvorby testovacích tříd, a tak ověřovat funkčnost jednotlivých vrstev aplikace, jako je Rest API či servisní nebo databázová vrstva. Automatické testování v rámci CI/CD je umožněno díky Gradle úloze *test*, kterou lze pouštět v procesu sestavení projektu v rámci pipeline.

3.6.1 GitLab

GitLab je platforma s otevřeným zdrojovým kódem pro vývoj a provoz softwarových produktů, která se nabízí různými metodami, platforma jako služba (PaaS) nebo software jako služba (SaaS). Na Fakultě informačních technologií se GitLab používá pro správu studentských a univerzitních projektů. Integrované funkce navíc umožňují provádět revize kódu, spravovat požadavky, automatizovat proces sestavení a vydání aplikace. V kontextu automatizace testování jde o začlenění běhu testů do samostatného kroku nebo do kroku sestavení projektu pipeliny. To lze udělat zavedením nové konfigurace v souboru *.gitlab-ci.yml*, viz. výpis kódu 13. Uvedený kód deklaruje novou úlohu v rámci CI/CD pro automatické spouštění testů, kde neúspěšný průběh testování blokuje akceptaci změn ve zdrojovém kódu.

```
1  stages:
2    - test
3  test_regular_job:
4    stage: test
5    script:
6      - ./gradlew test --refresh-dependencies -Dspring.profiles.active=test
7    only:
8      - merge_requests
```

Výpis kódu 13 Příklad konfigurace testovací úlohy v GitLabu

Kapitola 4
Implementace

Tato kapitola je zaměřena na popis praktického provedení refactoringu backendu komponenty vyplňovaní anketních lístků. Následující text detailně popisuje provedené změny, aplikované technologie a techniky. Kapitola poskytuje čtenáři ucelený pohled na klíčové aspekty v kontextu nové implementace.

4.1 Volba architektury

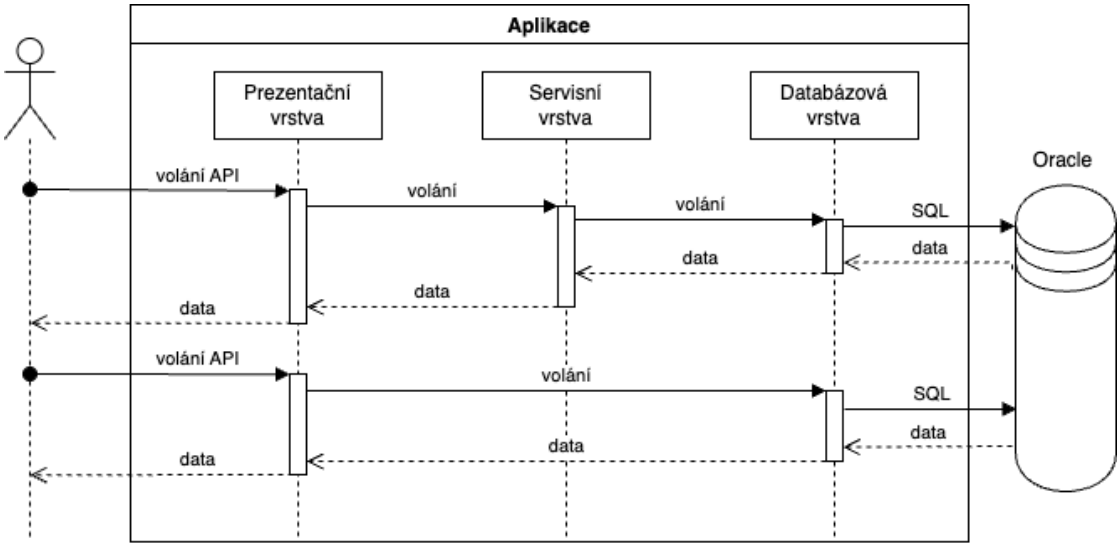
Správná volba architektury hraje klíčovou roli v procesu návrhu softwaru z několika důvodů. Za prvé, dobrý architektonický návrh aplikace podporuje lepší výkon a škálovatelnost, což je zásadní pro platformy s postupně rostoucím počtem uživatelů. Za druhé, zajištění rozšiřitelnosti a flexibility softwaru umožní jeho snadnou adaptaci pro nové požadavky. Za třetí, volba jednoduché architektury snižuje dlouhodobé náklady na údržbu softwarového řešení a také čas potřebný na zaškolení nových členů vývojového týmu. Dále dobře zvolená architektura pomáhá v prevenci chyb, a také minimalizaci času na jejich odstranění.

Z výše uvedených důvodů lze vytknout několik klíčových parametrů ovlivňujících volbu architektury: škálovatelnost, rozšiřitelnost, jednoduchou údržbu a čas potřebný na realizaci. V oblasti softwarového inženýrství existuje mnoho různých architektur, kde každá z nich je uzpůsobena specifickým typům projektů. Proto je důležité nejdříve určit typ projektu na základě uvedených případů užití, o kterých hovoří první kapitola. Aplikace Anketa využívá jeden centrální datový zdroj a většina operací je čtecího charakteru. Operace zápisu nebo modifikace dat jsou transakčního charakteru a používají se pouze v případech vytvoření studentského hodnocení nebo reakce vyučujícího. Hlavním úkolem servisní logiky je mapování dat z databázových záznamů do objektů prezentace. Proto v kontextu aplikace pro vyplňování anketních lístku se nejvíce hodí vrstevná klient-server architektura, a to z několika důvodů:

- **Existující klient:** Aplikace již disponuje hotovou implementací webového frontendu, který je oddělenou komponentou a zastupuje roli klienta.
- **Jednoduchost:** Rozdělení kódu na vrstvy usnadňuje údržbu softwarového produktu do budoucna a zvyšuje čitelnost zdrojového kódu aplikace. Navíc každá vrstva může být testována samostatně, nezávisle na celé aplikaci.
- **Oddělení zájmů:** Vrstvové dělení podporuje důležitý koncept ve vývoji softwaru, *Separation of Concerns*[38], který je základním stavebním kamenem kvalitního kódu. Každá vrstva se zaměřuje na konkrétní aspekt aplikace, což ve výsledku usnadňuje správu kódu a integraci nových technologií.

- **Škálovatelnost:** Serverová část aplikace se dá jednoduše škálovat, jak vertikálně, přidáním dalším serverových prostředků, tak i horizontálně, přidáním další aplikační instance. Komunikace mezi klientem a serverem je *stateless*[39] (není vázaná na konkrétní sezení), tím pádem není potřeba směřovat uživatele v rámci sezení na jednu instanci nebo synchronizovat stav dvou a více serverových aplikací.

Serverová část aplikace je rozdělena do tří vrstev: prezentační, servisní a databázové. Takové rozdělení je jednoduché a logické z ohledem na případy užití aplikace. Databázová vrstva pouze přistupuje k datům uloženým v databázi Oracle a jejím výstupem jsou nezpracované výsledky SQL dotazů. Tak například databázová reprezentace boolean hodnoty je daná písmeny „Y“ nebo „N“ pro pravdu, respektive nepravdu. Převod textové reprezentace boolean hodnoty se pak provádí v aplikační logice. Servisní vrstva poskytuje rozhraní pro provedení komplexnějších operací, jako je například uložení anketního lístku do databáze, kde je potřeba postupně zpracovat odpovědi na jednotlivé otázky. Prezentační vrstva komunikuje se servisní a databázovou vrstvou a poskytuje uživateli informace o aktuálním stavu aplikace, http návratový kód a obsah odpovědi. Příklad datového toku mezi vrstvami je uveden na obrázku 4.1.



Obrázek 4.1 Příklad datového toku mezi aplikačními vrstvami aplikace Anketa

4.2 Struktura projektu

Projekt je rozdělen do několika Gradle modulů pro lepší organizaci zdrojového kódu aplikace a flexibilnější správu verzí. Navíc díky rozdělení kódu do modulů lze separovat projektové závislosti, a tím snížit riziko konfliktu. Moduly mohou být znovu použité na jiných projektech, čímž se eliminuje potřeba vytvářet duplicitní kód. Rozdělení zdrojového kódu do modulů má vliv na čas sestavení projektu, protože změny v jednom nezávislém modulu nevyžadují znovusestavení celého projektu.

- `anketa-backend` ..... kořenový modul celého projektu
  - `anketa-api` ..... definice Restového API
  - `anketa-core-reactive` ..... definice aplikační logiky
  - `anketa-plugins` ..... definice gradle pluginů

4.3 Generování Rest API

Generování Restového API pomocí OpenApi Gradle pluginu umožňuje vytváření standardizovaného, srozumitelného a dokumentovaného rozhraní pro webovou aplikací. Zde je využito přístupu *design-first*[40], kde se nejdříve navrhuje restová API specifikace, a poté se generuje aplikační kód. Tento koncept pomáhá zajistit, že vytvořené rozhraní splňuje požadavky ještě před jeho implementací. OpenApi Gragle plugin může generovat jak serverové, tak i klientské balíčky zdrojového kódu pro širokou řadu programovacích jazyků a frameworků, což výrazně urychluje proces vývoje a redukuje množství projektového kódu. Dalším benefitem technologie OpenApi je možnost jednoduše spravovat verze uživatelského rozhraní, což je obrovskou výhodou pro projekty, kde je kladen velký důraz na zpětnou kompatibilitu. OpenApi poskytuje jednoznačný popis restového rozhraní, které je nezávislé na volbě programovacího jazyka.

V kontextu aplikace pro vyplňování anketních lístků za generování restového rozhraní je zodpovědný Gradle modul *anketa-api*. Ten obsahuje konfiguraci procesu generování, a také samotný soubor *anketa-api-1.0.0.yml* s definicí všech projektových endpointů, viz. výpis kódu 14. V příkladu uvedený endpoint slouží pro vytvoření autorizačního tokenu pro externího uživatele, jehož přihlašovací údaje aplikace získá z těla požadavku. Aplikace odpoví návratovým http kódem 404 v případě, že uživatel neexistuje.

```
1   openapi: 3.0.1
2   info:
3     title: Survey API
4   tags:
5     - name: auth
6       description: The application authorization API
7   paths:
8   /auth/external:
9     post:
10      tags:
11        - auth
12      operationId: authorizeExternalUser
13      summary: Creates external user login token
14        requestBody:
15        content:
16          application/json:
17            schema:
18              $ref: '#/components/schemas/AuthPostRequest'
19        required: true
20      responses:
21        '200':
22          description: Ok.
23          content:
24            application/json:
25              schema:
26                $ref: '#/components/schemas/AuthPostResponse'
27        ...
28        '404':
29          description: Requested resource was not found.
30          ...
```

Výpis kódu 14 Příklad generovaného endpointu prostřednictvím OpenApi pluginu.

4.4 Odstranění materializovaných pohledů

Popisu role materializovaných pohledů v kontextu aplikace pro vyplňování anketních lístků a spojených s nimi problémům se věnují samostatné podkapitoly Popisu existujícího řešení a Analýzy požadavků. Pro odstranění zmíněných nedostatků bylo zapotřebí nahradit celou logiku volání databázového zdroje novým modernějším přístupem. Příklad dřívějšího přístupu k vytváření SQL dotazů v aplikačním kódu je uveden ve výpisu kódu 12. Ten byl zcela nahrazen poskytnutou Spring funkcionalitou, známou jako *Spring Data JPA*, která podporuje deklarativní definici databázových dotazů, asynchronní implementaci operací a mapování výsledků SQL dotazů do Kotlin tříd. Jinými slovy lze vytvořit rozhraní ve kterém volání metody zastupuje proces vytvoření SQL dotazu, volání databázového zdroje a transformaci dat do aplikačních struktur. V uvedeném příkladu metoda *getCourseSurveyResultQuestions* přijímá tři povinné parametry, které jsou bezpečně nastaveny do databázového požadavku prostřednictvím parametrů. To ve výsledku má odstranit bezpečnostní nedostatek předchozí verze aplikace, která nebyla odolná vůči *SQL Injection* útokům.

```
1  @Repository
2  interface QuestionRepository : ReadOnlyQueryRepository {
3     @Query(
4         """
5         SELECT
6             QUESTION.QUESTION_KEY,
7             QUESTION.SHORT_CZ,
8             QUESTION.SHORT_EN
9         FROM
10            V_SURVEY_QUESTIONS QUESTION
11        JOIN
12            RES_CUR_SUM SUM ON SUM.SEMESTER_CODE = QUESTION.SEMESTER_CODE
13            AND SUM.SURVEY_KEY = QUESTION.SURVEY_KEY
14        WHERE
15            SUM.ID_COURSE = :courseId
16            AND QUESTION.SEMESTER_CODE = :semesterCode
17            AND QUESTION.SURVEY_KEY = :surveyId
18            AND QUESTION.RELATETED_TO_TEACHER = 'N'
19            AND QUESTION.IS_COURSE_SURVEY = 'Y'
20            AND QUESTION.HAS_TEXT_ANSWER = 'Y'
21    """,
22    )
23    suspend fun getCourseSurveyResultQuestions(
24        @Param("courseId") courseId: Long,
25        @Param("semesterCode") semesterCode: String,
26        @Param("surveyId") surveyId: Long,
27    ): List<CourseQuestionRecord>
28 }
```

Výpis kódu 15 Příklad volání databáze použitím Spring komponenty

Zavedení nového způsobu čtení a zápisu aplikačních dat do databáze vyžadovalo některé změny ve schématu databáze. Jedná se o odstranění materializovaných pohledů určených k uložení anketních výsledků a reakcí vyučujících po vyhodnocení předmětových a nepředmětových lístků. Příkladem takového pohledu je *RES_CUR_SUM_1_B182*, který slouží pro uložení předmětových statistik pro anketní lístek s ID 1 a během v semestru B182. Samotná tabulka obsahuje ne-

celých 200 záznamů. S každým novým během ankety vzniká nový materializovaný pohled s podobným počtem záznamů a odlišným sufixem semestru. Takto neefektivní použití databázového stroje má dopad na výkon aplikace a časem může být provoz a údržba softwarového řešení dost problematická. Proto pro vyřešení daného problému byly data z materializovaných pohledů *RES_CUR_SUM_* přeneseny do jedné tabulky *RES_CUR_SUM*, která bude mít navíc dva sloupce *SURVEY_KEY* a *SEMESTER_CODE* tvořící index. Daná kombinace sloupců může tvořit primární klíč spolu s nějakým identifikátorem původního pohledu pro lepší dotazování nad záznamy. V aplikačním kódu došlo k nahrazení vytvoření názvu tabulky spojením parametrů z požadavku za složitější *JOIN* operaci, viz. řádek 12 výpisu kódu 15. Seznam nahrazených materializovaných pohledů je uveden v tabulce 4.1.

Tabulka 4.1 Odstraněné materializované pohledy.

Prefix materializovaných pohledů	Nová tabulka
RES_CUR_ANSWER_*	RES_CUR_ANSWER
RES_CUR_EVAL_*	RES_CUR_EVAL
RES_CUR_ANSWER_*	RES_CUR_ANSWER
RES_CUR_SUM_*	RES_CUR_SUM
RES_FAC_ANSWER_*	RES_FAC_ANSWER
RES_FAC_EVAL_*	RES_FAC_EVAL
RES_FAC_SUM_*	RES_FAC_SUM
RES_FAC_QUEST_*	RES_FAC_QUEST
RES_STUDENT_COURSE_*	RES_STUDENT_COURSE
RES_TEACH_ANSWER_*	RES_TEACH_ANSWER
RES_TEACH_CUR_*	RES_TEACH_CUR
RES_TEACH_SUM_*	RES_TEACH_SUM

4.5 Optimalizace SQL dotazů

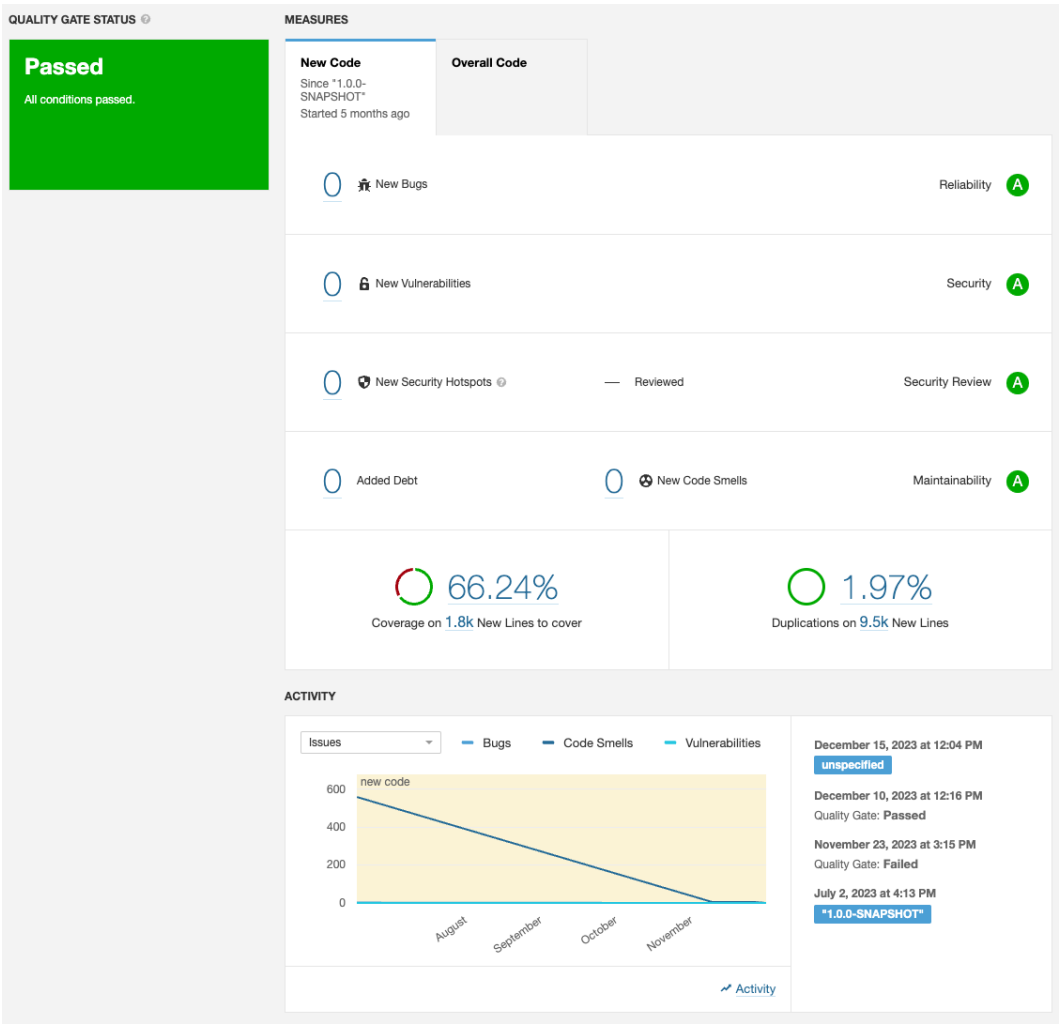
Během procesu refactoringu backendu v aplikaci Anketa došlo k optimalizaci některých databázových dotazů metodou odstranění nadbytečných částí nebo spojením SQL požadavků dohromady. Dále jsou uvedeny tří typické případy provedení úprav ve zdrojovém kódu.

- **Oprava návratové hodnoty:** V některých případech, kdy bylo potřeba ověřit boolean příznak určitého záznamu, docházelo k tomu, že se celý záznam načetl z databáze a následně se provedla jeho verifikace v aplikačním kódu. Tento postup byl nahrazen tak, že nyní SQL dotaz přímo vrací boolean hodnotu jako svůj výsledek. Příkladem je ověření přístupu uživatele, viz. třída *AllowedUserRepository*.
- **Neefektivní aktualizace záznamů:** V některých případech, v kontextu ukládání dat do databáze, docházelo k postupné aktualizaci záznamů po jednom namísto aktualizace celé dávky. Tento postup byl nahrazen aktualizací všech záznamů v dávce jedním dotazem. Příkladem je aktualizace reakcí vyučujících, viz. *ReactionRepository*.
- **Neefektivní načítání záznamů:** V některých případech, kde bylo potřeba načíst data z několika různých tabulek, docházelo k vytvoření dvou jednodušších SQL dotazů a postupnému volání databázového zdroje. Tento přístup byl nahrazen spojením dvou dotazů dohromady prostřednictvím *JOIN* operace a příslušných klíčů. Příkladem je načtení předmětových anketních výsledků, viz. *ResultRepository*.

4.6 Kontrola kvality kódu

Pro provedení statické analýzy kódu bylo využito několika nástrojů, které již byly popsány v kapitole Analýza požadavků a volba technologií. Cílem použití zmíněných nástrojů je zvýšení udržitelnosti projektu do budoucna a minimalizace pravděpodobnosti chybného chování aplikace v produkčním prostředí. Byly implementovány dvě úrovně ochrany, které by měly zabránit vzniku nekvalitního kódu v projektu. První z nich je lokální statická analýza změněných souborů s aplikačním kódem, která probíhá na počítači vývojáře. Ta je spuštěna prostřednictvím projektového Git hook[14]. Tento mechanismus by měl zabránit vzniku „špatného kódu“ v centrálním Git repozitáři. Ovšem tento bezpečnostní mechanismus lze obejít potlačením Git hook. Proto druhou úrovní ochrany je nastavení centrálního Git repozitáře, který by měl speciální konfigurací zabránit aktualizaci produkční větve *master*, pokud změněný kód neprošel kontrolou v rámci *merge requestu* a nesplnil všechny podmínky v nástroji *SonarQube*.

Pro statickou analýzu zdrojového kódu aplikace pro vyplňování anketních lístků bylo využito fakultní instance nástroje SonarQube, dostupného na adrese *https://dev.fit.cvut.cz/sonarqube/*. Během procesu refactoringu backendu se podařilo odstranit všechny nedostatky zdrojového kódu, duplicitní řádky, a také některá bezpečnostní rizika.



Obrázek 4.2 Statická analýza zdrojového kódu Ankety z nástroje SonarQube

4.7 Anonymizace ID vyučujících

Jedná se o požadavek, který vznikl během refactoringu backendu, a proto není součástí originálního zadání diplomové práce. Jde o problém spojený se starými identifikátory vyučujících ze studentského informačního systému, kde některé číselné hodnoty jsou totožné s rodnými čísly vyučujících. V kontextu aplikace pro vyplňování anketních lístků se tento identifikátor používá pro zobrazení detailního hodnocení vybraného vyučujícího. V důsledku toho může dojít k nechtěnému prozrazení zmíněného osobního údaje, což představuje bezpečnostní problém. Tato situace vyžaduje implementaci změn v aplikaci, jelikož nesprávné používání rodných čísel jako identifikátorů může vést k porušení zásad ochrany osobních údajů.

V rámci refactoringu backendu byla implementována funkcionalita mapování skutečného identifikátoru vyučujícího za náhodný údaj přiřazený správcem aplikace a naopak. Základem funkce anonymizace je tabulka *MV_ANONYMIZED_TEACHER_ID* se dvěma sloupci: skutečným identifikátorem a falešným identifikátorem vyučujícího. Tím pádem se uživateli ve výpisu výsledku anketního lístku zobrazí falešný identifikátor vyučujícího, a tak osobní údaj nebude prozrazen. Kontrola toho, že přiřazený identifikátor nemá jiného majitele probíhá pomocí databázového kontrolního skriptu, viz. výpis kódu 16.

```
1   CREATE OR REPLACE TRIGGER CHECK_REFERENCED_ID_UNIQUENESS
2   BEFORE INSERT OR UPDATE ON MV_ANONYMIZED_TEACHER_ID
3   FOR EACH ROW
4   DECLARE
5       v_count NUMBER;
6   BEGIN
7       SELECT COUNT(*)
8       INTO v_count
9       FROM MV_TEACHER
10      WHERE ID_TEACHER = :new.TEACHER_REFERENCED_ID;
11
12      IF v_count > 0 THEN
13          RAISE_APPLICATION_ERROR(-20001, 'Hodnota existuje...');
14      END IF;
15  END;
```

Výpis kódu 16 Ukázka databázového kontroléru unikátnosti identifikátoru vyučujícího

4.8 Dokumentace a doporučený postup údržby

Dokumentace je klíčová v oblasti vývoje softwaru, protože poskytuje nezbytný základ pro efektivní správu, údržbu, zaškolení nových členů týmu a implementaci nových funkcionalit. Kvalitní dokumentace umožňuje rychle pochopit strukturu a funkci zdrojového kódu, což je zásadní v případech diagnostiky a řešení problémů nebo integrace nových technologií. Kvalitní dokumentace pomáhá snížit riziko vzniku chyb, zvyšuje kvalitu poskytnutého řešení z hlediska procesu dlouhodobé údržby.

Předchozí verze zdrojových kódů komponenty vyplňování anketních lístků neměla skoro žádný Javadoc ani doporučený postup implementace nových funkcionalit, což vedlo k použití odlišných vývojových stylů, zanedbání správy verzí souborů a celkově vzniku nepřehledných struktur. To mělo za následek vznik velkého množství duplicitních částí kódu (kolem 20 procent v původní implementaci), nečitelných metod a zbytečných komentářů. V důsledku čeho byl proces refactoringu v některých situacích časově náročný.

Pro nápravu situace byly do projektu přidány soubory *README.md* a *CHANGELOG.md*, které by měly usnadnit novému vývojáři proces seznámení s projektem a doporučeným způsobem implementace nových funkcionalit. Mezi hlavní doporučení patří pravidelná kontrola nově vznikajících zdrojových kódů prostřednictvím code review a statická analýza v nástroji SonarQube. Tyto kroky by měly zajistit konzistenci provedení změn na projektu, a také podpořit jeho dlouhodobou údržbu do budoucna.

Kapitola 5
Testování

Tato kapitola je zaměřena na vysvětlení postupu testování vytvořeného kódu. Následující odstavce popisují použité techniky testování, typy testů a do projektu integrované technologie, které pomáhají zvýšit čitelnost vytvořených testů a podporují udržitelnost zdrojového kódu. To má za cíl seznámit čtenáře s metodami verifikace funkčnosti softwaru v kontextu aplikace pro vyplňování anketních lístků.

Testování softwaru je klíčovým prvkem procesu vývoje i přes značnou časovou obtížnost. Jedná se o postupy a metody, které zajišťují kvalitu, spolehlivost a bezpečnost vytvořené aplikace. Testování pomáhá identifikovat potencionální problémy, které by mohly výrazně ovlivnit provoz aplikace v produkčním prostředí. Existuje mnoho různých typů testů a testovacích metod. V kontextu této diplomové práce budou především použité jednotkové a integrační testy.

5.1 Jednotkové testy

Jednotkové testy slouží primárně k ověření malé části aplikačního kódu. Ty jsou základním stavebním kamenem kvalitního softwaru a při správné implementaci zajišťují minimalizaci chybného chování aplikace. Další velkou výhodou jednotkových testů je schopnost odhalit nevhodné použití externích závislostí ve zdrojovém kódu. Kód, který nesprávně využívá tyto závislosti, se obtížně testuje. To plyne z nutnosti simulovat funkcionality externích závislostí přímo v testu. Nelze-li takovou simulaci uskutečnit, pak je zřejmé, že externí závislost není použita korektně. To lze znázornit ve výpisu kódu 17, kde třída uvnitř vlastní implementace vytváří databázového klienta. Uvedená část aplikace se potýká s obtížným ověřením chování metody *getUser* prostřednictvím jednotkových testů bez napojení na skutečnou databázi. Izolací použitého databázového klienta lze dosáhnout lepší reprodukovatelnosti stavů testované aplikace.

```
1   class UserService {
2       private val db: DbClient = DbClient()
3
4       fun getUser(id: Int): User =
5           db.findUser(id)
6   }
```

Výpis kódu 17 Příklad špatného použití externí závislosti

V rámci refactoringu backendu aplikace Anketa bylo využito integrace frameworku Kotest pro testování v jazyce Kotlin. Ten podporuje různé styly psaní testů a je také navržen speciálně pro funkcionální využití. V následujících bodech jsou uvedeny důvody volby testovacího frameworku Kotest a jeho výhody oproti podobným platformám.

- **Intuitivní DSL:** Kotest nabízí intuitivní a flexibilní DSL, které umožňuje programátorům psát testy s využitím konstrukcí a klíčových slov blízkých přirozenému jazyku. Což znamená, že čtení zdrojového kódu jednotlivých testů připomíná čtení vět v anglickém jazyce. To zvyšuje srozumitelnost a ulehčuje údržbu projektu.

- **Integrace s Gradle a IDE:** Díky existujícímu externímu pluginu lze Kotest jednoduše integrovat s nástrojem Gradle, použitým na sestavení aplikace, což ve výsledku usnadňuje automatizaci procesu testování během sestavení projektu a vede ke zvýšení kvality kódu. Také Kotest lze integrovat do použitého pro vývoj IDE IntelliJ a tím zvýšit efektivitu práce.

- **Podpora asynchronního testování:** Jde o důležitou vlastnost testovacího frameworku, protože v základu aplikace je využito asynchronního řízení dat prostřednictvím Spring WebFlux. Kotest poskytuje jednoduché a flexibilní DSL rozhraní pro testování asynchronního kódu a podporuje řízení životního cyklu coroutin.

- **Bohatá standardní sada matcherů:** Kotest ve svém základním balíčku obsahuje širokou sadu matcherů pro pohodlné ověření chování testovaného kódu. Díky svému expresivnímu pojmenování, jako je například *shouldBe* či *shouldHave*, poskytují matcher funkce čitelný a srozumitelný způsob vyhodnocení testovacích tvrzení. Pro ověření očekávané hodnoty velikosti proměnné *result*, která byla získána z testované funkce, lze použít následující zápis: *result shouldHaveSize 1*.

V kontextu aplikace pro vyplňování anketních lístků jsou jednotkové testy použité pro ověření implementace restového API. Jedna testová třída testuje pouze jeden konkrétní endpoint a podle názvu samotné třídy lze odvodit testovanou funkcionalitu. Příklad takové třídy je uveden ve výpisu kódu 18. Tento výpis obsahuje pouze jeden test kontroly přístupu anonymním uživatelem. Celkově aplikační balíček *cz.cvut.fit.anketa.api* obsahuje 23 testových tříd a 85 testových metod.

```
1  @WebFluxTest(SurveyController::class)
2  class PostSurveyApiTest(...) : ShouldSpec({
3      should("fail with unauthorized, because user is not allowed") {
4          coEvery { securityService.getAuthorizedUser(TEST_TOKEN) }
5              returns LOGGED_TEST_USER
6
7          webTestClient.post()
8              .uri("/surveys")
9              .testAuthorization()
10             .contentJson(content)
11             .exchange()
12             .expectStatus().isForbidden
13     }
14 }
```

Výpis kódu 18 Příklad jednotkového testu s použitím Kotestu

5.2 Integrační testy

Integrační testy, na rozdíl od již zmíněných jednotkových testů, slouží pro ověření komunikae jednotlivých vrstev nebo komponent softwaru mezi sebou. Tento typ testování může odhalit problémy, které nejsou odhaleny jednotkovými testy, a to kvůli konceptu izolace. Nejčastějšími problémy, které integrační testování odhaluje, jsou chyby v integraci komponent třetích stran nebo kompatibilita datových formátů. Obvykle automatizace procesu běhu integračních testů vyžaduje konfiguraci databázového úložiště a simulaci externích systémů. Tuto možnost přináší použitý testovací framework Kotest, který použitím externích pluginů umožňuje napojení testové databáze nebo konfiguraci externích http služeb.

V kontextu aplikace pro vyplňování anketních lístků bylo využito nativní podpory technologie Spring pro vývoj integračních testů. Ten poskytuje možnost spouštět tzv. „lehký“ kontejner, který bude obsahovat pouze komponenty potřebné pro daný integrační test. Řízení životního cyklu, stavu a obsahu kontejneru probíhá prostřednictvím anotací *DataR2dbcTest*, *ContextConfiguration*, *EnableR2dbcRepositories* a dalších. Příklad projektového integračního testu lze najít ve výpisu kódu 19. Uvedený test kontroluje, zda rozhraní pro vytváření uživatelských účtů správně komunikuje s databázovým systémem a správně ukládá data. V uvedeném kódu není použita simulace komponent prostřednictvím anotace *MockkBean* a proto testovaná servisní třída komunikuje s použitou paměťovou databází H2. Pro urychlení běhu samotného testu se do kontejneru načítají pouze potřebné komponenty, což se nastavuje prostřednictvím anotací v hlavičce třídy.

```
1   @DataR2dbcTest
2   @EnableAutoConfiguration
3   @EnableR2dbcRepositories
8   @ContextConfiguration(classes = [IStoreUserImpl::class])
9   internal class UserServiceTestCase(
10        private val service: IStoreUser,
11  ) : ShouldSpec({
12      should("create user") {
13          val result = service.createUser(
14              UserPostRequest("test-token", "password")
15          )
16          result shouldBe Either.Right(Unit)
17      }
18
19      should("fail because token was not found") {
20          val result = service.createUser(
21              UserPostRequest("test-token-invalid", "password")
22          )
23          result shouldBe Either.Left(StoreUserProblem.TokenNotFound)
24      }
25  })
```

Výpis kódu 19 Příklad integračního testu s použitím Kotestu.

5.2.1 Databáze H2

Databáze H2 je lehký, v paměti běžící relační databázový systém napsaný v programovacím jazyce Java. Je známý díky své rychlosti, flexibilitě a jednoduchosti integrace do aplikačního kódu. V kontextu aplikace pro vyplňování anketních lístků byly přidány dvě externí závislosti, viz. výpis kódu 20. Webový framework Spring se postaral o propojení aplikace s databázovým systémem. Proto je v dnešní době databáze H2 populární volbou pro implementaci integračních testů. Zde je uvedeno několik důvodů zapojení technologie do projektu:

- **Rychlost:** H2 je paměťovou databází, což znamená, že veškerý obsah databáze se drží v RAM paměti počítače během procesu testování. Proto všechny dotazové operace jsou rychlejší ve srovnání s tradičními databázemi přistupujícími k datům uloženým na pevném disku. Rychlá odezva je skvělým benefitem v situacích, kdy je potřeba frekventovaně resetovat stav datového úložiště. Ovšem popsaný přístup k uložení a správě dat má nevýhodu: velikost jedné databázové instance je omezená velikostí RAM paměti testovacího prostředí. To může mít vliv na paralelní spouštění integračních testů.
- **Jednoduchá konfigurace:** Webový framework Spring Webflux umožňuje jednoduchou konfiguraci pro připojení k databázi H2 nastavením parametrů v souboru ve formátu YAML, viz. výpis kódu 21. Kromě toho počáteční stav databáze lze inicializovat ze souboru ve formátu SQL nebo ze speciálního databázového souboru. V kontextu aplikace pro vyplňování anketních lístků je využito standardního souboru *schema.sql* pro inicializaci stavu databáze, vytvoření schéma, tabulek a vložení dat. Navíc integrační testy s použitím H2 lze snadně spouštět prostřednictvím CI/CD, což zvyšuje udržitelnost projektu.
- **Izolace a transakce:** Databáze H2 podporuje transakce a jejich řízení, což umožňuje testovat kód závislý na transakční logice. Kromě toho použitím paměťové databáze lze zajistit dobrou izolaci mezi testy, což znamená, že výsledek jednoho testu neovlivní stav následujícího. V testovém prostředí tak lze spouštět několik databázových instancí na různých systémových portech, a tím výrazně zrychlit celý proces testování.

Ovšem i přes uvedené výhody, databáze H2 není plně kompatibilní s databází Oracle použitou v produkčním prostředí. Některé specifické funkce se dají napodobit vytvořením speciálních metod v kontextu H2, ale pro plnou kompatibilitu je zapotřebí úprava datového modelu v produkčních tabulkách a transformace některých SQL dotazů.

```
1  testRuntimeOnly("com.h2database:h2")
2  testRuntimeOnly("io.r2dbc:r2dbc-h2")
```

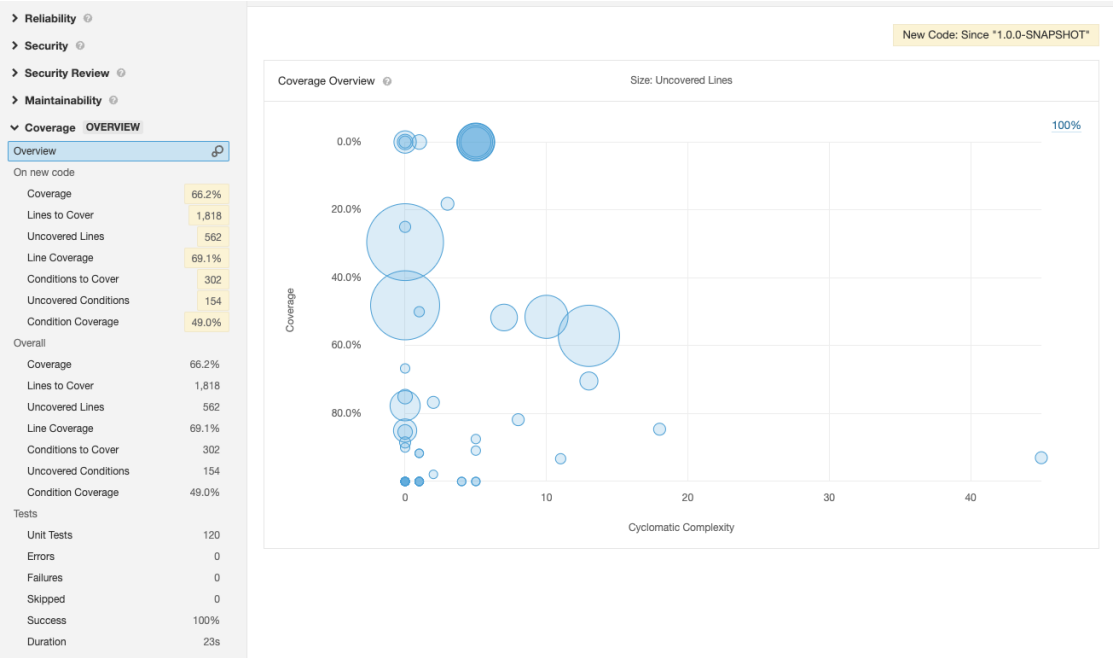
Výpis kódu 20 Gradle závislosti pro paměťovou databázi H2

```
1  spring:
2    r2dbc:
3      url: r2dbc:h2:mem:///anketa;MODE=Oracle;SCHEMA=PUBLIC;
4      username: sa
5      password:
```

Výpis kódu 21 Konfigurace paměťové databáze H2

5.3 Výsledky testování

Během procesu testování bylo do aplikace vyplňování anketních lístků přidáno 34 testových tříd, které obsahují dohromady 120 jednotkových a integračních testů, viz. obrázek 5.1. Podle analýzy z nástroje SonarQube bylo dosaženo testového pokrytí kódu z **66 procent**, což je více než nastavené minimum **50 procent**. Došlo k výraznému pokroku oproti stavu před začátkem refactoringu backendu, kdy neexistoval žádný aplikační test. Ale stále některé části kódu, zejména projektové konfigurace nebo Rest rozhraní, nejsou pokryté testy kvůli své špatné testovatelnosti a závislosti na stavu aplikace. Příkladem takové situace může být rozhraní pro přihlášení uživatele *cz.cvut.fit.anketa.controller.AuthController#authorizeUser()*, které je závislé na http kontextu aplikace a čerpá informace z atributů http požadavku. K dosažení ještě lepšího testového pokrytí kódu lze uvažovat o metodě automatického testování vůči instanci aplikace v kontejneru.



Obrázek 5.1 Analýza testového pokrytí aplikačního kódu

Kapitola 6
Závěr

Diplomová práce se zabývá refactoringem backendu v kontextu komponenty vyplňování anketních lístků univerzitní aplikace Anketa. Motivací k provedení změn bylo především zvýšení efektivity aplikace a zlepšení její udržitelnosti do budoucna. Také se zde otevřel prostor pro implementaci moderních postupů, konceptů a zapojení inovativních řešení. Hlavním cílem provedení refactoringu bylo zavedení konzistentních vývojových standardů, což by usnadnilo provoz a údržbu aplikace v následujících letech.

Na začátku byla provedena analýza problémové domény popisující hlavní pojmy, uživatelské role, případy užití a časový průběh vyplňování anketních lístků. První kapitola uvádí čtenáře do problematiky a poskytuje přehled o komplexitě stávajícího řešení i požadavcích na systém.

Dále se práce zaměřuje na popis existujícího řešení z pohledu softwarové implementace. Zde jsou popsány použité technologie, postupy a koncepty. Text se zaměřuje na výhody a nevýhody zvolených metod a hledání možných alternativ. Analýza existujícího řešení vyžadovala značné teoretické znalosti a zároveň také praktické dovednosti s technologiemi integrovanými do projektu.

Při analýze vstupních požadavků bylo vycházeno primárně ze zkušeností z praxe s projekty založenými na technologiích Java a Spring Boot. Text třetí kapitoly diskutuje různé cesty k dosažení stanovených cílů metodou integrace pomocných pluginů nebo využitím externích platforem.

Praktická část diplomové zahrnovala provedení změn ve zdrojovém kódu aplikace a implementaci automatizovaných testovacích metod. Všechny stanovené cíle a požadavky se podařilo splnit. V následujících bodech jsou uvedeny jednotlivé technologické aktualizace v projektu.

- **Migrace z Java do Kotlin:** Došlo ke změně primárního programovacího jazyka projektu, kdy Kotlin nahradil původně používanou Javu. Nový jazyk nabízí větší flexibilitu a modernější přístup ke psání zdrojového kódu, což má za následek lepší produktivitu vývojáře.
- **Migrace z Maven do Gradle:** Došlo k nahrazení dříve použitého nástroje na sestavení Maven za Gradle. Ten nabízí širší možnosti konfigurace a větší nabídku pomocných pluginů.
- **Migrace do Spring WebFlux:** Původně použitý aplikační framework Spring Boot byl nahrazen jiným aplikačním frameworkem Spring WebFlux, který je svojí architekturou a použitými koncepty vhodnější volbou pro implementaci projektu typu Anketa.
- **Zavedení statické analýzy kódu:** Do projektu byly integrovány pluginy Kotlinter, Detekt a OWASP Dependency-Check, které provádí statickou analýzu kódu a kontrolu zranitelností aplikačních závislostí, čímž se podpoří udržitelnost projektu do budoucna. Dalším užitečným nástrojem je zavedení kontroly kódu v SonarQubu, kde lze sledovat historické změny v aplikaci a vyhodnocovat řadu dalších užitečných parametrů.

- **Integrace Sentry:** Integrace Sentry do aplikačního backendu umožnila automatické hlášení chyb ze všech tří prostředí do centrálního úložiště. Kromě automatického reportu identifikovaných problémů neplacená licence Sentry umožňuje také monitoring pomalých http požadavků a sledování průběhu SQL dotazů.
- **Odstranění materializovaných pohledů:** Jedná se o zásadní změnu ve zdrojovém kódu aplikace pro vyplňování anketních lístků, kde došlo k opravám v databázovém schématu a rozsáhlé implementaci nových struktur a tříd. Samotná změna přinesla projektu odstranění bezpečnostního rizika, lepší správu databázového schématu a snadnější udržitelnost.
- **Integrace frameworku Kotest:** Integrace testovacího frameworku Kotest do projektu přinesla řadu výhod, kde primárními z nich je podpora asynchronního programování a intuitivní DSL, které zajišťuje čitelnost kódu testových metod.

Stará implementace aplikační logiky byla zcela zahozena. Původní kód v jazyce Java byl zcela nahrazen novou implementací v jazyce Kotlin, proto došlo k redukci počtu řádků zdrojového kódu aplikační logiky z 5 tisíc na 3 tisíce. Také bylo vytvořeno 120 jednotkových a integračních testů, které pokrývají 66 procent nového kódu. Duplicitní kód se objevuje pouze v třídách repozitářích poskytujících propojení s databázovým zdrojem aplikace.

Díky provedeným změnám byly odstraněny bezpečnostní rizika SQL injekce a prozrazení rodných čísel některých vyučujících. Také byly optimalizovány některé databázové dotazy, což vede k lepší výkonnosti a škálovatelnosti aplikace.

6.1 Budoucí rozvoj aplikace

I přes rozsáhlé změny ve zdrojovém kódu aplikace, je zde stále prostor pro zlepšení. Dalším krokem v rozvoji aplikace by mohl být refactoring restového rozhraní a zavedení nové flexibilnější verze. Stávající restové rozhraní definuje logiku načtení dat, jejich postupným skládáním, místo poskytnutí přístupu k datům jako zdroji. Díky integraci pluginu OpenApi, je možné implementovat novou verzi restového rozhraní se zachováním již existující verze.

Co se týče databáze, bylo by možné zavést nástroj pro správu jednotlivých verzí. To by umožnilo mít přehled o aktualizacích v databázovém schématu a podporu bezpečného nasazení změn do produkčního prostředí.

Pro usnadnění procesu integračního testování a zajištění lepší kompatibility databázových zdrojů by bylo možné implementovat testovací kontejner. Tyto změny mohou být například implementovány v následujících závěrečných pracích věnovaných problematice aplikace Anketa.

Bibliografie

1. ŠTECHA, Vojtěch. Anketa ČVUT - verze 3.0 - vyplňování anketních lístků [online]. 2019 [cit. 2023-12-17]. Dostupné z: https://dspace.cvut.cz/handle/10467/82698. Accepted: 2019-06-11T14:51:23Z Publisher: České vysoké učení technické v Praze. Vypočetní a informační centrum.
2. KOPALOVÁ, Alice. Automatizace administrátorských procesů databáze aplikace Anketa ČVUT [online]. 2023 [cit. 2023-12-04]. Dostupné z: https://dspace.cvut.cz/handle/10467/109372. Accepted: 2023-06-15T22:51:38Z Publisher: České vysoké učení technické v Praze. Vypočetní a informační centrum.
3. HAI, Nam Nguyen. Anketa ČVUT - redesign vyplňovacího modulu [online]. 2021 [cit. 2023-12-17]. Dostupné z: https://dspace.cvut.cz/handle/10467/96819. Accepted: 2021-08-26T22:53:08Z Publisher: České vysoké učení technické v Praze. Vypočetní a informační centrum.
4. JUN, Jakub. Anketa ČVUT verze 3 - modul pro správu anket - uživatelské rozhraní [online]. 2020 [cit. 2023-12-17]. Dostupné z: https://dspace.cvut.cz/handle/10467/90395. Accepted: 2020-09-04T14:01:30Z Publisher: České vysoké učení technické v Praze. Vypočetní a informační centrum.
5. ILYA, Parakhnich. Anketa ČVUT v 4.0 - Datové schéma a byznys vrstva v databázi [online]. 2017 [cit. 2023-12-17]. Dostupné z: https://dspace.cvut.cz/handle/10467/69130. Accepted: 2017-06-07T14:53:44Z Publisher: České vysoké učení technické v Praze. Vypočetní a informační centrum.
6. *Domain Modeling: What you need to know before coding* [Thoughtworks] [online]. [cit. 2023-12-17]. Dostupné z: https://www.thoughtworks.com/insights/blog/agile-project-management/domain-modeling-what-you-need-to-know-before-coding.
7. Use case. In: *Wikipedia* [online]. 2023 [cit. 2023-12-17]. Dostupné z: https://en.wikipedia.org/w/index.php?title=Use_case&oldid=1180881181. Page Version ID: 1180881181.
8. TECHNICKÉ, České vysoké učení. *Anketa ČVUT: dokumentace a metodika provozu systému* [online]. České vysoké učení technické, 2022 [cit. 2023-04-23]. Dostupné z: https://www.cvut.cz/sites/default/files/content/d1dc93cd-5894-4521-b799-c7e715d3c59e/cs/20221007-metodicky-pokyn-c-32022.pdf.
9. *Responsive Web Design: What Is RWD? - 2023* [MasterClass] [online]. [cit. 2023-12-17]. Dostupné z: https://www.masterclass.com/articles/responsive-web-design.
10. CONSORTIUM, World Wide Web et al. Document object model (DOM) level 3 core specification. 2004. Dostupné také z: https://travesia.mcu.es/server/api/core/bitstreams/6abd537a-f388-472a-a577-2457aca21055/content.

11. BUCANEK, James (ed.). Model-View-Controller Pattern. In: *Learn Objective-C for Java Developers* [online]. Berkeley, CA: Apress, 2009, s. 353–402 [cit. 2023-12-17]. ISBN 978-1-4302-2370-2. Dostupné z DOI: 10.1007/978-1-4302-2370-2_20.
12. Search engine optimization. In: *Wikipedia* [online]. 2023 [cit. 2023-12-17]. Dostupné z: https://en.wikipedia.org/w/index.php?title=Search_engine_optimization&oldid=1187292867. Page Version ID: 1187292867.
13. *Cross Site Scripting (XSS) — OWASP Foundation* [online]. [cit. 2023-12-17]. Dostupné z: https://owasp.org/www-community/attacks/xss/.
14. *Git - Git Hooks* [online]. [cit. 2023-12-17]. Dostupné z: https://git-scm.com/book/en/v2/Customizing-Git-Git-Hooks.
15. DHADUK, Hiren. *Bootstrap vs. Material - A Detailed Comparison [2023]* [Simform - Product Engineering Company] [online]. 2021-04-30. [cit. 2024-01-11]. Dostupné z: https://www.simform.com/blog/bootstrap-vs-material/.
16. *CSS: Cascading Style Sheets — MDN* [online]. 2023-07-22. [cit. 2024-01-11]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS.
17. *Interceptors — Axios Docs* [online]. [cit. 2023-12-17]. Dostupné z: https://axios-http.com/docs/interceptors.
18. HODGES, Nick. *Code Against Interfaces, Not Implementations* [Medium] [online]. 2020-02-17. [cit. 2023-12-17]. Dostupné z: https://betterprogramming.pub/code-against-interfaces-not-implementations-37b30e7ab992.
19. *SOLID Definition – the SOLID Principles of Object-Oriented Design Explained* [freeCodeCamp.org] [online]. 2022-04-26. [cit. 2023-12-17]. Dostupné z: https://www.freecodecamp.org/news/solid-principles-single-responsibility-principle-explained/.
20. TUFANO, Michele; PALOMBA, Fabio; BAVOTA, Gabriele; OLIVETO, Rocco; DI PENTA, Massimiliano; DE LUCIA, Andrea; POSHYVANYK, Denys. When and Why Your Code Starts to Smell Bad. In: *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering* [online]. 2015, sv. 1, s. 403–414 [cit. 2023-12-17]. Dostupné z DOI: 10.1109/ICSE.2015.59. ISSN: 1558-1225.
21. GUPTA, Lokesh. *Guide to Lazy Loading in Hibernate* [HowToDoInJava] [online]. 2014-09-26. [cit. 2023-12-17]. Dostupné z: https://howtodoinjava.com/hibernate/lazy-loading-in-hibernate/.
22. JONES, Michael B.; BRADLEY, John; SAKIMURA, Nat. *JSON Web Token (JWT)* [online]. 2015-05. [cit. 2023-11-30]. Request for Comments, RFC 7519. Internet Engineering Task Force. Dostupné z DOI: 10.17487/RFC7519. Num Pages: 30.
23. Man-in-the-middle attack. In: *Wikipedia* [online]. 2023 [cit. 2023-12-01]. Dostupné z: https://en.wikipedia.org/w/index.php?title=Man-in-the-middle_attack&oldid=1187622284. Page Version ID: 1187622284.
24. ROESER, Mary Beth; ADAMS, Drew; ASHDOWN, Lance; ARORA, Vikas; BABY, Thomas; BAER, Hermann; BASKAN, Yasin; BAYLISS, Nigel; BELDEN, Eric; BHATIYA, Rajesh; CHAUDHRY, Atif; CORCORAN, Nelson; DAS, Dinesh; DILMAN, Mark; DURAISWAMY, Sudha; FAN, Yanfei; GLEESON, Mike; GOMEZ, Laura Solis; GOPAL, Naveen; HAMMERSCHMIDT, Beda; HUEY, Patricia; KNAGGS, Peter; KRISHNAMURTHY, Sriram; KRUGLIKOV, Andre; KUMAR, Praveen; LLEWELLYN, Bryn; LI, Yunrui; MACNICOL, Roger; MANIYANI, Darshan; MCDERMID, David; MICHELS, Jan; MIR, Rahil; MULAGUND, Gopal; NEALL, Ian; POTINENI, Padmaja; QIAN, Hanlin; RAVIPATI, Giridhar; SHANTAVEERAPPA, Prashanth; SMITH, Wayne; TOMAR, Samarjeet; VYAS, Nirav; WILLIAMS, Alan; WITKOWSKI, Andy; WOLICKI, Sergiusz; YU, Tsae-Feng; ZHANG, Weiran; KRISHNAMURTHY, Usha. *ROWID Pseudocolumn* [Oracle Help Center]

[online]. [cit. 2023-12-17]. Dostupné z: https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/ROWID-Pseudocolumn.html. Publisher: November2023.

25. *NULL: The Billion Dollar Mistake — HackerNoon* [online]. [cit. 2023-11-05]. Dostupné z: https://hackernoon.com/null-the-billion-dollar-mistake-8t5z32d6.
26. *Extensions — Kotlin* [Kotlin Help] [online]. [cit. 2023-12-17]. Dostupné z: https://kotlinlang.org/docs/extensions.html.
27. ELIZAROV, Roman; BELYAEV, Mikhail; AKHIN, Marat; USMANOV, Ilmir. Kotlin coroutines: design and implementation. In: *Proceedings of the 2021 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software* [online]. New York, NY, USA: Association for Computing Machinery, 2021, s. 68–84 [cit. 2023-12-17]. Onward! 2021. ISBN 978-1-4503-9110-8. Dostupné z DOI: 10.1145/3486607.3486751.
28. GREGORY, Tom. *All about the Gradle task graph* [online]. 2021-03-22. [cit. 2023-12-17]. Dostupné z: https://gradlehero.com/all-about-the-gradle-task-graph/. Section: posts.
29. MAK, Gary. Inversion of Control and Containers. In: MAK, Gary (ed.). *Spring Recipes: A Problem-Solution Approach* [online]. Berkeley, CA: Apress, 2008, s. 3–20 [cit. 2023-12-17]. ISBN 978-1-4302-0623-1. Dostupné z DOI: 10.1007/978-1-4302-0623-1_1.
30. Dependency injection. In: *Wikipedia* [online]. 2023 [cit. 2023-12-17]. Dostupné z: https://en.wikipedia.org/w/index.php?title=Dependency_injection&oldid=1188955281. Page Version ID: 1188955281.
31. *Using Java Reflection* [online]. [cit. 2023-11-06]. Dostupné z: https://www.oracle.com/technical-resources/articles/java/javareflection.html.
32. *Learning Network Programming with Java* [online]. [cit. 2023-12-17]. Dostupné z: https://subscription.packtpub.com/book/cloud-and-networking/9781785885471/7/ch07lvl1sec53/the-thread-per-request-approach.
33. KANADE, Aditya. Chapter Seven - Event-Based Concurrency: Applications, Abstractions, and Analyses. In: MEMON, Atif M. (ed.). *Advances in Computers* [online]. Elsevier, 2019, sv. 112, s. 379–412 [cit. 2023-12-17]. Dostupné z DOI: 10.1016/bs.adcom.2017.12.006.
34. *Coding conventions — Kotlin* [Kotlin Help] [online]. [cit. 2023-12-17]. Dostupné z: https://kotlinlang.org/docs/coding-conventions.html.
35. *IBM Documentation* [online]. 2021-03-08. [cit. 2023-12-17]. Dostupné z: https://www.ibm.com/docs/en/raa/6.1?topic=metrics-cyclomatic-complexity.
36. *OWASP Web Security Testing Guide — OWASP Foundation* [online]. [cit. 2023-12-17]. Dostupné z: https://owasp.org/www-project-web-security-testing-guide/.
37. *What is SQL Injection — SQLI Attack Example & Prevention Methods — Imperva* [Learning Center] [online]. [cit. 2023-12-17]. Dostupné z: https://www.imperva.com/learn/application-security/sql-injection-sqli/.
38. *Separation of Concerns - ABAP Keyword Documentation* [online]. [cit. 2023-12-17]. Dostupné z: https://help.sap.com/doc/abapdocu_753_index_htm/7.53/en-US/abenseperation_concerns_guidl.htm.
39. Service statelessness principle. In: *Wikipedia* [online]. 2021 [cit. 2023-12-17]. Dostupné z: https://en.wikipedia.org/w/index.php?title=Service_statelessness_principle&oldid=1014842633. Page Version ID: 1014842633.
40. *Code-First vs. Design-First: Eliminate Friction with API Exploration* [SmartBear.com] [online]. [cit. 2023-12-17]. Dostupné z: https://swagger.io/blog/code-first-vs-design-first-api/.

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu média
- `exe` ..... adresář se spustitelnou formou implementace
- `src`
  - `kotlin` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF