ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA EKONOMICKÁ

Bakalářská práce

Webové aplikace

Web applications

Radek Nolč

Plzeň 2024

Čestné prohlášení

Prohlašuji, že jsem bakalářskou práci na téma

„Webové aplikace“

vypracoval samostatně pod odborným dohledem vedoucího bakalářské práce za použití pramenů uvedených v přiložené bibliografii.

Plzeň dne 22. 4. 2024 v.r. *Radek Nolč*

Zásady pro vypracování práce

1. Vymezte terminologii spojenou s webovými aplikacemi a jejich vývojem.
2. Představte technologie spojené s vývojem webových aplikací.
3. Popište životní cyklus produktu.
4. Vytvořte návrh, naprogramujte a vytvořte dokumentaci vlastní webové aplikace.
5. Formulujte závěr.

Studijní program
Informační management

Poděkování

Tímto bych rád poděkoval vedoucímu bakalářské práce, Ing. Janu Brčákovi, za jeho vedení, cenné připomínky a osobní přístup, který mi při zpracování této práce věnoval. I přes to, že se jedná o téma mimo jeho hlavní odbornost, zvládl tuto výzvu profesionálně.

Obsah

Úvod

Webové aplikace jsou v současnosti neodmyslitelnou součástí našeho digitálního světa. Od e-commerce přes vzdělávací platformy až po sociální sítě. Webové aplikace umožňují uživatelům interagovat s obrovským množstvím informací a služeb skrze pouhé kliknutí myši. Díky své přístupnosti a širokému rozsahu funkcí tak představují klíčový prvek moderního internetového ekosystému.

Rozvoj technologií a programovacích jazyků v posledních letech dramaticky změnil způsob, jakým jsou webové aplikace navrhovány a vyvíjeny. Vznik nových aplikačních rámců, vylepšené možnosti vývoje klientské a serverové části, a nové přístupy k ukládání a zpracování dat. Všechny tyto faktory ovlivňují efektivitu a bezpečnost webových aplikací.

Hlavním cílem této bakalářské práce je vytvoření webové aplikace, která integrací moderních technologií a metodik umožní studentům univerzity využívající IS/STAG automatizovat zápisy na zkouškové termíny. Vývoj webové aplikace bude organizován do následujících segmentů:

- Specifikace softwaru - tento segment se bude zabývat detailním popsáním požadavků na webovou aplikaci.
- Návrh webové aplikace - v této fázi bude vytvořen detailní návrh architektury aplikace.
- Implementace serverové části aplikace - bude popisovat vývoj serverové strany aplikace.
- Implementace klientské části aplikace - tato část se bude věnovat vývoji klientské části aplikace.
- Validace aplikace - bude obsahovat aspekty testování aplikace.
- Vypuštění aplikace do internetu - bude popisovat detaily nasazení aplikace na produkční server, včetně konfigurace serverového prostředí.
- Návrhy na další rozšíření - poslední část bude diskutovat možnosti pro budoucí vylepšení a rozšíření aplikace.

Bakalářská práce je strukturována do šesti hlavních kapitol, které postupně pokrývají celé spektrum vývoje webových aplikací. Po úvodní kapitole, která představuje základní pojmy jako webový server a webová aplikace, následuje podrobný přehled technologií používaných v klientských a serverových částech, včetně HTML, CSS, JavaScript, PHP a Java. Dále se práce věnuje metodám uchovávání dat a komunikačním technikám mezi klientem a serverem. Představeny jsou i klíčové nástroje pro vývoj webových aplikací a diskutovány jsou softwarové procesy a přístupy k vývoji softwaru. Před realizací webové aplikace je také věnována kapitola problematice architektury aplikace.

1. Základní pojmy z oblasti webu

Pro dobré porozumění tématu je nutné nejdříve představit klíčové pojmy spjaté s tím, z čeho se skládá internet.

1.1 Webový server

Webový server lze popsat jako spojení softwaru a hardwaru, které odpovídá na požadavky od uživatelů internetu. Hlavním úkolem je zobrazování, zpracování a doručování obsahu. (Gillis, 2020)

Hardware webového serveru je veřejně připojen k internetu tak, aby mohl přijímat požadavky uživatelů. Tyto požadavky dále zpracuje software webového serveru a pošle uživateli zpět obsah webové stránky. (Gillis, 2020)

Při výběru vhodného softwaru je důležité brát zřetel zejména na kompatibilitu se stávajícím operačním systémem a možnosti zabezpečení. (Gillis, 2020)

Gillis (2020) uvádí jako často používané softwary webových serverů například:

- **Apache HTTP Server** Vyvíjen Apache Software Foundation. Jedná se o bezplatný software pro řadu operačních systémů.
- **Microsoft Internet Information Services (IIS)** Vyvíjen společností Microsoft pro produkty Microsoftu, jedná se o velmi rozšířené řešení.
- **Nginx** Jedno z nejpopulárnějších řešení pro administrátory díky velkým možnostem využití a škálovatelnosti. Díky architektuře dokáže zpracovat více požadavků najednou.

1.2 Webová stránka

Webová stránka lze popsat jako kolekce několika souborů, zpravidla psaných ve značkovacím jazyce HTML (HyperText Markup Language), používaných pro různé účely. Jednotlivé části webové stránky jsou propojeny pomocí tzv. hypertextových odkazů. Hypertextový odkaz si lze představit jako podtržený, do modra zbarvený útržek textu. Webové stránky mohou obsahovat i méně typické odkazy, s takovými odkazy se lze setkat například v hlavičce v podobě navigační lišty.

Existují dva druhy zpracovávání webových stránek na základě práce webového serveru, a to statické a dynamické. (Gillis, 2020)

Statické webové stránky jsou vráceny z webového serveru na základě požadavku uživatele v podobě původně uložených souborů na serveru. Neprobíhá zde žádné zpracování požadavku na pozadí, takže se všem uživatelům posílá stejný obsah. (Gillis, 2020)

U dynamických webových stránek se neposílají původně uložené soubory, ale veškeré zpracování požadavku uživatele probíhá až za chodu. (Gillis, 2020) Při tvorbě dynamických webových stránek se používá různých programovacích jazyků, velmi často kombinace PHP a JavaScriptu. Jedním z nejlépe představitelných příkladů dynamické webové stránky může být internetový obchod.

1.3 Webová aplikace

Webová aplikace je software fungující ve webovém prohlížeči, kdy webový prohlížeč funguje jako grafické rozhraní webové aplikace. Firmy používají webové aplikace k výměně informací a poskytování služeb na dálku. Běžně se můžeme setkat s webovými aplikacemi například v podobě internetových obchodů, sociálních sítí nebo zpravodajských stránek. Webové aplikace tedy umožňují přístup ke složitým funkcím bez potřeby instalace nebo konfigurace softwaru na zařízení koncového uživatele. (Shields, 2023)

Na internetu je mnoho úspěšných webových aplikací, mezi nejznámější patří například Google Docs, Netflix, Microsoft Office, Facebook nebo Spotify. (Shields, 2023)

1.3.1 Výhody webových aplikací

Přístupnost Webové aplikace jsou přístupné z různých webových prohlížečů, což znamená, že uživatelé mohou k aplikacím přistupovat z jakéhokoli zařízení. (Shields, 2023) Jedinou podmínkou pro používání webové aplikace je funkčnost webového prohlížeče na daném zařízení.

Efektivní vývoj Vývoj webových aplikací je považován za relativně jednoduchý a nákladově efektivní. Díky dostupnosti webových aplikací přes webové prohlížeče programátoři nemusí upravovat kód aplikace pro více platforem, kód stačí upravit pouze jednou. (Shields, 2023)

Jednoduchost úprav Úpravy zdrojového kódu webové aplikace mohou být vypuštěny přímo na server a projeví se ihned pro všechny uživatele. (Shields, 2023)

Rychlejší uvedení na trh Webové aplikace jsou v porovnání například s mobilními aplikacemi rychlejší na vývoj. Zároveň není potřeba dalších postupů při zveřejnění aplikace, jako je tomu u mobilních aplikací s Google Play u operačního systému Android nebo App Store u systému iOS. (Shields, 2023)

Dohledatelnost Webové aplikace se mohou díky indexaci vyhledávačů ukazovat ve výsledcích vyhledávání. (Shields, 2023)

1.3.2 Nevýhody webových aplikací

Nedostupnost offline Webové aplikace jsou zcela závislé na internetovém připojení, což znamená, že uživatelé nemohou v případě omezeného připojení k internetu aplikaci využívat. (Shields, 2023)

Nekonzistence Variabilita webových prohlížečů může přinést problémy - jednotlivé prohlížeče mohou prezentovat stránky odlišně, což může způsobit nekonzistentní vizuální podobu. (Shields, 2023)

Omezené využití nativních funkcí Webové aplikace mají omezenou schopnost přistupovat k nativním funkcím daného zařízení. Mezi nativní funkce

zařízení může patřit například fotoaparát, GPS nebo NFC čtečka. (Shields, 2023)

1.3.3 Fungování webových aplikací

Webové aplikace fungují na základě architektury client-server. Kód je rozdělen na dvě, úzce spolupracující, části. První částí je klientská (uživatelská), účelem klientské části je interakce s uživatelem pomocí rozhraní, například tlačítek a formulářů. Serverová část se stará o zpracování dat a následné odpovědi se zpracovaným požadavkem zpět do klientské části, kde dojde ke zobrazení výsledku. (Popovych, 2023)

1.3.4 Aplikační rámec

Aplikační rámec, známý také jako „Framework“ v angličtině, představuje strukturu, kterou lze využít jako fundament pro vývoj libovolné aplikace. Tím pádem programátor není nucen znovu vytvářet základní stavební kameny. Každý aplikační rámec je obvykle spojen s nějakým programovacím jazykem a je navržen pro specifický účel či způsob využití. (Olawanle, 2022)

Používání aplikačních rámců šetří čas a snižuje riziko chyb programátorů aplikací – není potřeba psát vše odzačátku. Aplikační rámce jsou již otestovány, což snižuje obavy z chyb. Další výhody zahrnují možnost vytvářet bezpečnější kód, jednodušší testování a ladění, eliminaci duplicitního kódu a udržování čistého, snadno přizpůsobitelného, kódu. (Olawanle, 2022)

1.3.5 API

Aplikační programové rozhraní (API) určuje způsob a pravidla, jakým si klientská část vyměňuje data se serverovou částí. Tyto pravidla lze přirovnat ke smlouvě typu: „Pokud mi pošleš data A, já vždycky odpovím s daty B. Když mi pošleš data C, odpovídám D.“ (Cocca, 2023)

S pomocí tohoto souboru pravidel klientská část přesně ví, co musí požadovat k dokončení určité úlohy od serverové části. Serverová část zase přesně ví, co se požaduje, když je třeba provést určitou akci. (Cocca, 2023)

2. Technologie spojené s webovými aplikacemi

Kód webových aplikací je rozdělen na dvě části, v této kapitole budou představeny technologie pro serverovou a klientskou část. Po představení těchto částí budou představeny způsoby uchovávání dat a způsoby komunikace mezi klientskou a serverovou částí.

2.1 Technologie serverové části

Při programování na straně serveru je klíčovým úkolem programátora zajistit, aby server zpracovával a posílal pouze relevantní data. Programovací jazyky navržené pro serverovou část jsou klíčovým nástrojem ke splnění této zásadní úlohy. Tyto jazyky například umožňují vytvářet dynamické webové stránky a aplikace, provádět plánování a těžbu dat, automatizovat procesy a rozesílat e-maily. (Tran, 2021)

2.1.1 PHP

Jedná se o velmi populární programovací jazyk, který byl vytvořen v roce 1995. S ohledem na jeho flexibilitu a rychlost vývoje zůstává oblíbeným i v současnosti, a díky němu funguje více než 20 milionů webových stránek. (Tran, 2021)

Mezi klíčové vlastnosti patří již implementované funkce zrychlující počáteční vývoj, podpora mnoha databázových služeb, zahrnutý kód pro základní bezpečnostní hrozby a široká podpora napříč poskytovateli cloudových služeb. (Harish, 2023)

2.1.2 Node.js

Node.js patří k převládajícím programovacím jazykům v oblasti webu díky jeho výkonu a škálovatelnosti. Poskytuje v základu mnoho podpůrných knihoven, které mohou být pro mnoho programátorů zprvu nepřehledné. (Tran, 2021)

2.1.3 Python

Python je jazyk pro všeobecné použití. Díky jeho rychlosti a jednoduchosti vývoje se často používá také v programování webových aplikací. Python má mnoho knihoven a aplikačních rámců usnadňující a zrychlující vývoj. Obecně je Python vnímán jako programovací jazyk, který je lehký na naučení díky jeho struktuře kódu. (Tran, 2021)

Mezi klíčové vlastnosti patří jednoduchá struktura kódu, nativní knihovny, nezávislost vůči platformě, na které kód funguje, a dynamické typování. (Harish, 2023)

2.1.4 Java

Java je programovací jazyk, který vyniká svou schopností poskytovat platformní nezávislost. Tato vlastnost je dosažena díky konceptu „Write Once, Run Anywhere“ (WORA), což znamená, že kód napsaný v Javě by měl být schopen běžet na různých operačních systémech bez potřeby úprav. (Tran, 2021)

Java byla navržena s důrazem na minimalizaci závislostí na konkrétních knihovnách a platformách. Tato vlastnost dává vývojářům větší flexibilitu při psaní kódu, protože nemusí řešit specifické vlastnosti různých operačních systémů. To zjednodušuje vývoj a údržbu aplikací, což je zvláště významné v prostředích s rozmanitými platformami. (Tran, 2021)

Další výhodou Javy je, že je objektově orientovaným programovacím jazykem. (Tran, 2021) Objektově orientovaný přístup usnadňuje organizaci a strukturování kódu, což přispívá k lepší čitelnosti a udržitelnosti. (Gillis, 2021) Java se také vyznačuje bezpečností a odolností vůči chybám, což je důležité zejména při vývoji rozsáhlých a spolehlivých softwarových systémů. (Tran, 2021)

2.2 Technologie klientské části

Při programování klientské části je doporučeno oddělení souborů na základě způsobu použití. Zpravidla se rozděluje na HTML soubory, které se využívají jako nositelé obsahu, CSS soubory, sloužící pro definování vzhledu stránky a JavaScriptu, ten definuje chování webové aplikace. Při správném rozdělení těchto souborů lze dosáhnout dobré konzistence – úprava vzhledu stránky nezmění chování aplikace. (Microsoft, 2023)

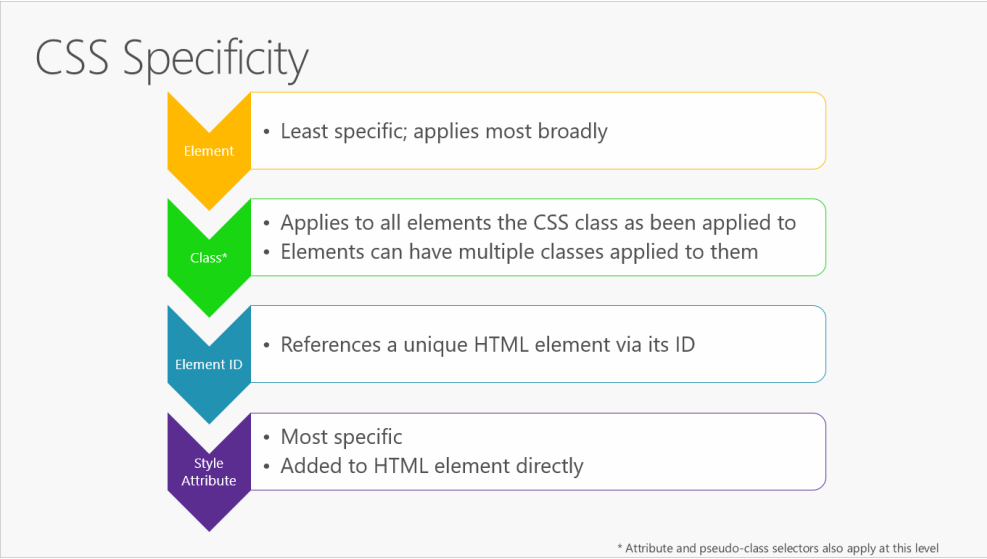
2.2.1 HTML

Hypertext Markup Language (HTML) je jednoduchý značkovací jazyk, který se používá k vytváření webových dokumentů. Tyto webové dokumenty jsou soubory uložené v textových dokumentech, takže je lze prohlížet na jakémkoli počítači. (Stejskal, 2004) Značky v HTML reprezentují jednotlivé prvky na stránce. Příklady těchto značek zahrnují formátovaný text, obrázky, formuláře, bloky a další. (Microsoft, 2023)

2.2.2 Kaskádové styly

Kaskádové styly (CSS) se používají pro definování vzhledu, tyto styly se mohou zapisovat přímo do HTML souboru, nebo do separátního souboru. Definované styly se mohou navzájem přepisovat – může být definovaný globální vzhled nějakého prvku webového dokumentu a definicí dalšího stylu, stejného prvku, se tento globální vzhled nahradí. To, zda dojde k přepsání nějakého stylu určuje, jak podrobně je definováno, pro který prvek styl platí. Tato pravidla jsou znázorněna na Obr. 1. (Microsoft, 2023)

Obrázek 1: Určení míry podrobnosti definovaného stylu



Zdroj: Microsoft (2023)

2.2.3 JavaScript

JavaScript je programovací jazyk určen převážně pro webové prostředí. Stejně jako je tomu u CSS, i JavaScript může být vepsán přímo do HTML souboru. Převážně se, pokud to podmínky dovolí, doporučuje JavaScript co nejvíce oddělit od HTML souboru do vlastního souboru. (Microsoft, 2023)

Pokud programátor pracuje s JavaScriptem, nevyhne se těm nejběžnějším úkolům, mezi které lze zařadit například vybrání nějakého HTML prvku a zjištění jeho hodnoty, provádění validace dat nebo volání serverové části přes nějaké rozhraní. (Microsoft, 2023)

2.3 Technologie uchování dat

Existuje široká škála způsobů, jak uchovávat data webových aplikací, a každý z nich má své vlastní výhody a nevýhody. Jedním z tradičních přístupů je využívání relačních databází, často také používaným řešením je uchovávání a členění dat do souborů.

Tyto způsoby uchovávání dat se mohou vzájemně doplňovat. Nejlepším příkladem může být správa uživatelského účtu. Uživatel může mít například jméno, heslo, e-mailovou adresu a profilový obrázek. Data se jménem, heslem a e-mailovou adresou budou uloženy v databázi, profilový obrázek uživatele v souborovém systému webové aplikace.

V této sekci se seznámíme s metodou ukládání dat do databáze.

2.3.1 Databáze

„Databáze je soubor dat používaný k modelování některých typů organizačních struktur nebo organizačních procesů.“ (Hernandez, 2006, s. 42) Při procesu shromažďování dat nezáleží na tom, zda programátor používá papír, nebo nějaký

počítačový program. O databázi se jedná do doby, dokud se shromažďují data organizovaně. (Hernandez, 2006)

V databázovém světě se mluví o dvou typech databází, o operačních a analytických. (Hernandez, 2006)

Operační databáze se používají v případech, kdy je potřeba, aby data byla dynamická (neustále se měnící). Tato data v operačních databázích reflektují aktuální stav informací. Skvělým příkladem organizace používající operační databáze může být nemocnice, protože se její data neustále mění. (Hernandez, 2006)

Analytické databáze jsou používány v případech, kdy je potřeba ukládat a vyhledávat časově závislá data. Tento typ databáze ukládá data, které nejsou určeny k pozdějším změnám, mění se jen velmi zřídka. Příkladem organizace používající tento typ databáze může být společnost provádějící marketingové analýzy. (Hernandez, 2006)

2.4 Technologie komunikace klient-server

K dosažení efektivní komunikace mezi klientskou a serverovou částí je nezbytné pečlivě vybrat vhodný způsob komunikace. Tento krok ovlivňuje celkový výkon a škálovatelnost webových aplikací.

2.4.1 SOAP

SOAP (Simple Object Access Protocol) je komunikační protokol z roku 1999, který využívá XML formát k definování komunikace. Tento bezestavový a jednosměrný protokol vyžaduje, aby veškerá komunikace byla obsažena v rámci SOAP dokumentu; jakékoli informace mimo tento rámec jsou ignorovány. Tato specifikace pomáhá zajistit přesné a jednoznačné definování požadavků. (Alonso et al., 2003)

2.4.2 REST

Architektonický styl Representational State Transfer (REST) z roku 2000 nachází časté uplatnění při tvorbě webových služeb a rozhraní pro programování (API). REST API jsou koncipována s ohledem na lehkost, škálovatelnost a flexibilitu. Tento typ komunikace se často využívá nejen ve webových aplikacích, ale také v mobilních aplikacích a při vývoji Internetu věcí (IoT). (Cocca, 2023)

2.4.3 WebSockety

WebSockety umožňují rychlou a bezpečnou obousměrnou komunikaci mezi klientem a serverem na webu prostřednictvím HTTP(S) s podporou textových i binárních zpráv. Jedná se o způsob komunikace v reálném čase, jsou tedy ideální pro hry pro více hráčů, okamžité notifikace (například na sociálních sítích), aktuální zobrazení informací o akciích a další podobné aplikace. (Microsoft, 2022a)

2.4.4 GraphQL

GraphQL je dotazovacím jazykem z roku 2012, který poskytuje efektivní a flexibilní možnosti načítání dat ze serveru. V porovnání s REST je složitější, nabízí ale výhody jako silně typovaná data nebo jedno koncové místo volání. Tento způsob komunikace je vhodný pro tvorbu aplikací s komplexními požadavky na data a tam, kde je důležité efektivní načítání dat. (Cocca, 2023)

3. Nástroje pro vývoj webových aplikací

Tato kapitola se zaměřuje na představení vývojových prostředí používaných při vývoji webových aplikací a několika užitečných nástrojů.

3.1 Integrované vývojové prostředí

Red Hat (2019) uvádí, že integrované vývojové prostředí (IDE) je software pro tvorbu aplikací, který spojuje běžné nástroje vývojáře do jednoho grafického uživatelského rozhraní a obvykle zahrnuje:

- **Editor zdrojového kódu** Textový editor, který pomáhá při psaní softwarového kódu pomocí funkcí jako je zvýrazňování syntaxe s vizuálními nápovědami, poskytování automatického dokončování kódu a kontrola chyb při psaní kódu.
- **Automatizace lokálního sestavení** Nástroje, které automatizují jednoduché, opakovatelné úkoly jako součást vytváření lokálního sestavení softwaru pro použití vývojářem, jako je kompilace zdrojového kódu na binární kód, balení binárního kódu a spouštění automatizovaných testů.
- **Debugger** Program pro testování vyvíjeného softwaru, který dokáže graficky zobrazit umístění chyby v původním kódu.

IDE umožňuje vývojářům rychle začít programovat aplikace, protože není nutné manuálně konfigurovat a integrovat více nástrojů jako součást procesu nastavení. Vývojáři také nemusí trávit hodiny individuálním učením se, jak používat různé nástroje. Většina funkcí IDE je navržena tak, aby šetřily čas, příkladem funkce s velkou úsporou času může být inteligentní dokončování kódu a automatická generace kódu, což odstraňuje potřebu psát celé znakové sekvence ručně. (Red Hat, 2019)

I bez použití IDE je možné vyvíjet aplikace. Stačí, aby si vývojář v podstatě vytvořil své vlastní IDE ruční integrací různých nástrojů s textovým editorem, jako například Vim nebo Emacs. Pro některé vývojáře přináší tato metoda výhodu ve velkém přizpůsobení a kontrole. Nicméně, v podnikovém kontextu jsou úspory času, standardizace prostředí a automatizační funkce moderních IDE obvykle preferovanější. (Red Hat, 2019)

3.1.1 Visual Studio Code

Visual Studio Code je bezplatný, lehký, ale výkonný editor zdrojového kódu. Software je k dispozici pro Windows, macOS, Linux a Raspberry Pi OS. Obsahuje vestavěnou podporu pro JavaScript, TypeScript a Node.js a disponuje bohatým ekosystémem rozšíření pro další programovací jazyky (například C++, C#, Java, Python, PHP a Go). (Heller, 2022)

Vývojáři oceňují lehký charakter editoru a zároveň jeho schopnost kontroly syntaxe, automatického doplňování kódu, refaktorování, ladění a možnost provádět kontrolu v repozitáři. (Heller, 2022)

3.1.2 IntelliJ IDEA

IntelliJ IDEA je vysoce oblíbené integrované vývojové prostředí vyvinuté společností JetBrains. Toto vývojové prostředí je primárně navrženo pro práci s jazykem Java, ale podporuje také širokou škálu dalších programovacích jazyků, jako je Kotlin, Groovy, Scala a další. (Simplilearn, 2023)

IntelliJ IDEA například poskytuje inteligentní podporu pro psaní kódu včetně chytrého automatického dokončování, analýzy a návrhů, což zrychluje proces programování a minimalizuje chyby. Díky pokročilým nástrojům pro refaktoring, jako je přejmenování, extrakce metod a optimalizace importů, mohou vývojáři snadno a s jistotou upravovat svůj kód. IDE průběžně provádí inspekci kódu na potenciální problémy a nabízí rychlá řešení a doporučení pro udržení kvality kódu a minimalizaci chyb. (Simplilearn, 2023)

3.2 Užitečné nástroje

Při vývoji softwaru jsou nezbytné nástroje, které značně usnadňují proces programování. Tyto nástroje se mohou starat například o správu verzí, posílání předdefinovaných API požadavků nebo virtualizaci a kontejnerizaci služeb.

3.2.1 Git

Git je distribuovaný systém správy verzí, což znamená, že lokální klon projektu je kompletním repozitářem pro správu verzí. Tyto plně funkční lokální repozitáře usnadňují práci offline nebo na dálku. Vývojáři tzv. commitují svou práci lokálně a poté synchronizují svůj klon repozitáře s kopií na serveru. (Microsoft, 2022b)

Microsoft (2022b) dále uvádí například tyto výhody používání Gitu:

- **Simultánní vývoj** Každý vývojář má svoji vlastní lokální kopii kódu a může pracovat současně na svých vlastních úkolech nezávisle na ostatních vývojářích. Díky tomu, že je téměř každá operace lokální, funguje Git také offline.
- **Integrace** Díky své oblíbenosti se Git integruje do většiny nástrojů a produktů. Většina integrovaných vývojových prostředí má vestavěnou podporu právě pro tento systém správy verzí, a mnoho dalších nástrojů, běžně používaných při vývoji, podporuje integraci právě s tímto systémem. Může se jednat o nástroje například pro kontinuální integraci, kontinuální nasazení, automatizované testování a další.
- **Podpora komunity** Git je open-source a hodně vývojářů ho považuje standardem pro správu verzí. Velikost komunitní podpory ve srovnání s ostatními systémy správy verzí usnadňuje získání pomoci v případě potřeby.
- **Efektivita týmu vývojářů** Použití Gitu pro správu zdrojového kódu zvyšuje produktivitu týmu tím, že podporuje spolupráci, automatizuje procesy a zlepšuje viditelnost a stopovatelnost práce.

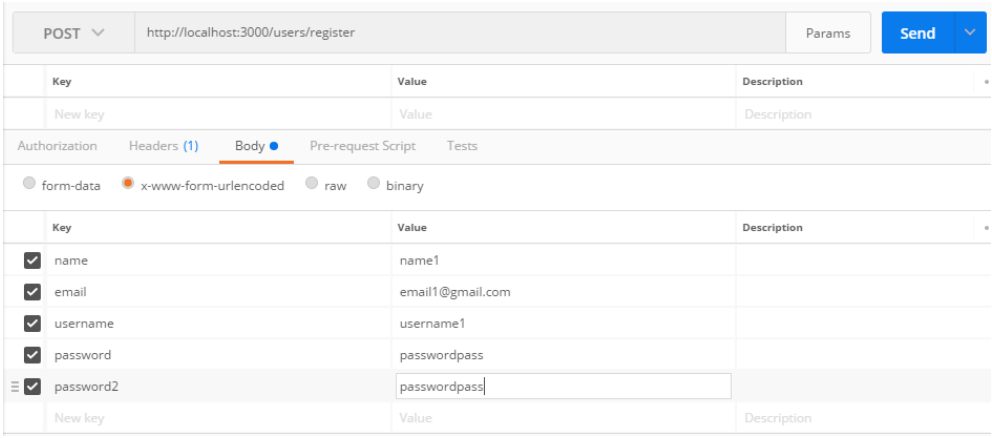
3.2.2 Postman

Postman je nástroj na podporu vývoje API, který velmi usnadňuje proces tvorby, testování a úprav na straně API. Tento nástroj zahrnuje téměř veškerou funk-

cionalitu, která může být potřebná pro každého vývojáře. Postman umožňuje provádět různé typy HTTP požadavků, ukládat prostředí pro pozdější použití a převádět API do kódu pro různé programovací jazyky. (Hooda, 2022)

Obr. 2 ukazuje použití nástroje, v ukázce se posílá *POST* HTTP požadavek na URL adresu *http://localhost:3000/users/register* s klíči *name*, *email*, *username*, *password* a *password2* a jejich hodnotami.

Obrázek 2: Ukázka použití aplikace Postman



Zdroj: Hooda (2022)

3.2.3 Docker

Docker poskytuje schopnost zabalit a spustit aplikaci ve volně izolovaném prostředí nazvaném kontejner. Izolace a bezpečnost umožňují spouštět mnoho kontejnerů současně na daném hostitelském serveru. Kontejnery jsou lehké a obsahují všechny potřebné nástroje a služby pro spuštění aplikace, takže není potřeba se spoléhat na to, co je nainstalováno na hostitelském serveru. (Docker, 2023)

4. Softwarové procesy

Softwarový proces představuje strukturu aktivit, které jsou uplatňovány při vývoji softwarového produktu. Existuje několik modelů pro tyto procesy, každý model rozdílně popisuje přístupy k různým úkolům nebo aktivitám, které probíhají během tohoto procesu. (Rossman, 2010)

Každý model musí obsahovat čtyři klíčové aktivity – specifikace (definování funkcí a omezení), návrh a implementace (splnění specifikací), validace (testování funkcí) a evoluce softwaru (vývoj nových a úprava starých funkcí). (Sommerville, 2013)

Přestože neexistuje žádný ideální proces, lze aktuálně používané procesy zdokonalit, jelikož se mohou skládat ze zastaralých technik, nebo nemusí využívat optimálních postupů. (Sommerville, 2013)

4.1 Modely softwarových procesů

Tato sekce se zabývá modely softwarových procesů, které představují rozdílné metody a přístupy používané ve vývoji softwaru. Tyto modely nejenže definují kroky, kterými projde software od nápadu po jeho realizaci, ale také ovlivňují způsob práce týmu a rozdělení úkolů v průběhu vývoje. Sekce se podrobněji zaměřuje na dva klíčové modely: Vodopádový model a Spirálový model.

4.1.1 Vodopádový model

Jedním z modelů popisující tento vývojový proces je Vodopádový model (Waterfall Model v angličtině), který přirovnává pokrok vývoje jako stále plynoucí přes fáze koncepce, zahájení, analýzy, návrhu, konstrukce, testování a údržby. Tento stále plynoucí proces tedy připomíná vodopád. (Rossman, 2010)

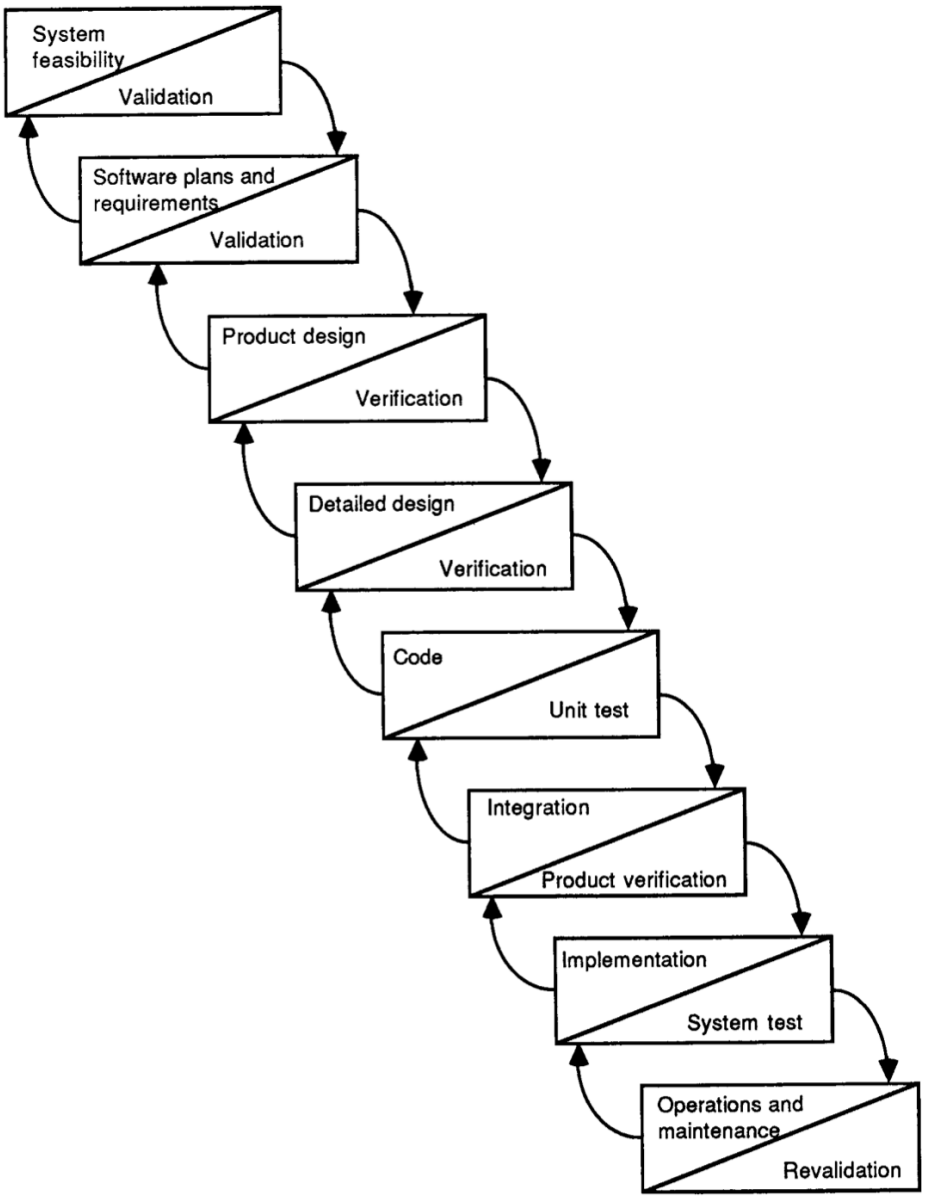
Definice modelu tvrdí, že by se mělo na další fázi přejít až po dokončení a zdokonalení aktuální fáze. Nicméně existuje mnoho dalších modelů inspirovaných tímto modelem, které se od původního modelu liší. (Rossman, 2010)

První formální popis tohoto modelu pochází z článku autora Winstona W. Royce z roku 1970. (Rossman, 2010) Vodopádový model se stal prvním definovaným modelem na světě. (Sommerville, 2013)

Sommerville (2013) uvádí, že hlavní fáze vodopádového modelu přesně reflektují základní vývojové aktivity:

1. analýza a definice požadavků (určení systémových služeb a cílů),
2. návrh systému a softwaru (určení celkové systémové architektury na základě analýzy požadavků),
3. implementace a testování jednotek (realizace návrhu systému jako sady jednotek a následné testování těchto jednotek),
4. integrace a testování systému (spojení jednotek do komplexního systému, následné otestování systému jako celku),
5. a provoz a údržba (instalace systému, oprava neodhalených chyb, zlepšování stávajících a vytváření nových funkcí na základě požadavků).

Obrázek 3: Vodopádový model



Zdroj: Boehm (1988)

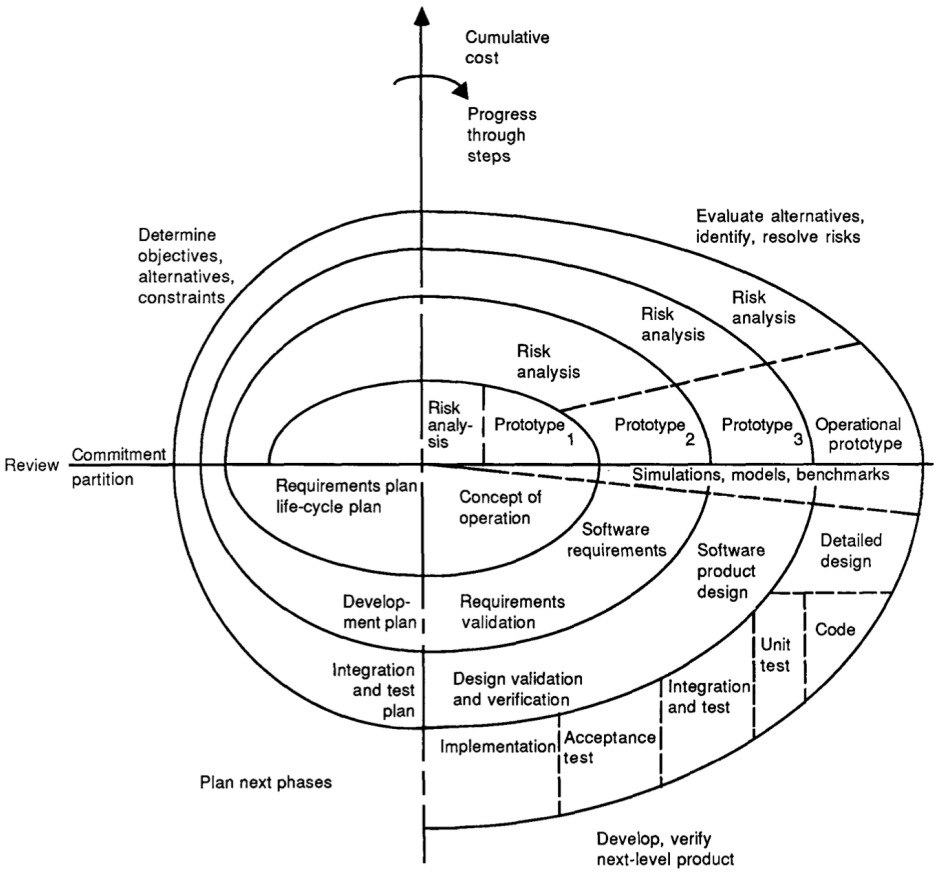
Vodopádový model zpracovává jednotlivé fáze za sebou v sekvenčním pořadí. Jakmile se fáze označí za hotovou a potvrzenou, všichni na tuto fázi nahlíží jako na „vytesanou do kamene“. Jakmile je předešlá fáze dokončena, přichází na řadu další fáze. (Rossman, 2010)

4.1.2 Spirálový model

Druhým ze zmíněných modelů je Spirálový model (Spiral Model v angličtině), který představuje softwarový vývoj jako spirálový proces, kde se jeho čtyři fáze - požadavky, návrh, implementace a testování neustále opakují. (Sommerville, 2013)

Spirálový model vychází z tzv. vydání, které označují aktuální verzi softwaru. Po dokončení poslední fáze (testování) začíná nové vydání, kdy se na počátku vydání opět definují požadavky. (Sommerville, 2013)

Obrázek 4: Spirálový model



Zdroj: Boehm (1988)

4.2 Aktivity softwarových procesů

Existují čtyři základní aktivity softwarových procesů: specifikace, návrh a implementace, validace a evoluce. (Sommerville, 2013) Tyto aktivity budou představeny v této sekci, společně s další dodatečnou aktivitou, kterou je podpora softwaru.

4.2.1 Specifikace softwaru

Specifikace je jednou z klíčových aktivit ve vývojovém procesu. Vzhledem k tomu, že zadavatelé požadavků na software často mají velmi abstraktní představu o tom, co přesně od softwaru potřebují, je během specifikace nezbytné identifikovat klíčové funkcionality softwaru na základě analýzy požadavků. (Rossman, 2010)

Analýza požadavků má tři druhy aktivity. Prvním druhem aktivity je sběr požadavků, kdy dochází ke komunikaci s poptávajícím softwaru a potenciálními

uživateli o tom, co od dané aplikace očekávají. Další aktivitou je analyzování, zda jsou požadavky dobře definované, případně zda se nevylučují. Po zjištění nesouvislostí se opět komunikuje, aby došlo k dodefinování požadavků. Poslední aktivitou je zaznamenání požadavků do různých forem – dokumentů, uživatelských scénářů nebo specifikací. (Rossman, 2010)

Některé požadavky na funkcionality mohou být po sepsání následně vyřazeny například z důvodu ceny funkce nebo nejasné funkcionality na začátku vývojového procesu. (Rossman, 2010)

4.2.2 Návrh a implementace softwaru

Fáze návrhu a implementace převádí specifikace systému do funkční formy. (Sommerville, 2013)

Aktivity procesu návrhu se mohou lišit v závislosti na vyvíjeném systému - návrh systému nereagujícího v reálném čase se bude zpravidla lišit od návrhu systému, kde funkčnost v reálném čase je stěžejní. (Sommerville, 2013)

Sommerville (2013) uvádí čtyři aktivity, které mohou být součástí procesu návrhu softwaru:

1. návrh architektury (definování celkové struktury systému a jeho základní komponenty),
2. návrh rozhraní (definování, jak lze komponentu používat),
3. návrh komponent (popsání jednotlivých komponent systému a jejich funkčnosti),
4. a návrh databáze (návrh datových struktur a způsobů jejich reprezentace v databázi).

Sommerville (2013) dále uvádí, že tyto aktivity vedou k sadě výstupů, které se liší svou podrobností a vzhledem s ohledem na komplexitu systému.

4.2.3 Validace softwaru

Validace je důležitou částí softwarového vývoje, tato část vývoje zajišťuje, že všechny chyby způsobené při vývoji jsou odhaleny co nejdříve. (Rossman, 2010) Zároveň má tato fáze dokázat, že systém vyhovuje specifikaci a splňuje očekávání zadavatele. (Sommerville, 2013)

Systémy (kromě malých programů) by se neměly testovat jako jedna velká jednotka, ale odděleně. Proces testování se rozděluje na vývojové (testování jednotlivých komponent vývojáři systému), systémové (testování celého systému jako celku, tj. již integrovaných komponent) a předávací testování (testování s reálnými daty od zadavatele). (Sommerville, 2013)

Proces vývojového testování (testování komponent) se většinou prokládá se samostatným vývojem. Programátoři při vývoji sestavují testovací data a následně kód testují již během vývoje, což je jeden z ekonomicky rozumných přístupů, protože programátoři perfektně znají jednotlivé komponenty, takže mohou generovat vhodné testovací případy. (Sommerville, 2013)

4.2.4 Evoluce softwaru

Údržba a vylepšování softwaru na základě objevených chyb nebo nových požadavků dokáže být mnohem časově náročnější, než původní vývoj softwaru. Může se stát, že bude potřeba přidat kus kódu, který vůbec nezapadá do původního návrhu programu. Horším scénářem je nutnost zpracovat požadavek na novou funkcionalitu, která způsobí přeorganizování architektury půlky programu. Pokud nastane, že cena údržby přesáhne 25 % původní ceny softwaru, lze označit kvalitu softwaru za špatnou a management by v tomto případě měl rozhodnout o předělání systému dříve, než se náklady za údržbu dostanou mimo kontrolu. (Rossman, 2010)

4.2.5 Podpora softwaru

Neméně důležitým procesem, který si mnoho programátorů neuvědomuje, je zaučení a podpora softwaru, pokud software nikdo nepoužívá (neumí ho používat), všechna snaha přijde vniveč. Proto je velice důležité zajistit budoucím uživatelům softwaru školení, které je naučí používat daný software. (Rossman, 2010)

5. Architektura

Návrh architektury pomáhá vysvětlit, jak bude systém uspořádán, a slouží k definování celkové struktury daného systému. Jedná se o první fázi procesu návrhu softwaru – identifikují se hlavní (strukturní) komponenty a vztahy mezi nimi. Výstupem je model architektury popisující organizaci komunikujících komponent. (Sommerville, 2013)

Architektura softwaru hraje klíčovou roli v určování výkonu, robustnosti, distribuovatelnosti a údržbě systému. Tato architektura je sestavena z jednotlivých komponentů, které mají za úkol implementovat funkční požadavky aplikace. (Sommerville, 2013)

Robustnost softwarové architektury znamená schopnost systému odolávat neočekávaným situacím a chybám. Distribuovatelnost se týká schopnosti šířit zátěž mezi různé části systému, což je důležité pro dosažení vyšší škálovatelnosti a odolnosti vůči selháním. S architekturou jsou také pevně spjaty možnosti údržby systému – čitelný a modulární kód usnadňuje údržbu a další rozšíření aplikace. (Sommerville, 2013)

5.1 Rozhodnutí při návrhu architektury

Při navrhování architektury softwaru čelí softwarový architekt klíčovým rozhodnutím, která zásadně ovlivňují vývojový proces. S ohledem na své znalosti a zkušenosti musí zodpovědět otázky týkající se volby architektonické koncepce, jako například rozhodnutí o použití existující generické architektury nebo vytvoření nového, specifického modelu. Dále musí zvážit, zda systém bude pracovat s více jádry procesoru, což může výrazně ovlivnit výkon a škálovatelnost. Rovněž je nutné definovat přístup k vývoji, zahrnující vývojové metody, nástroje a postupy. Samozřejmě softwarový architekt nesmí zapomenout ani na možnosti údržby, je potřeba vybrat architekturu z jemně rozdělených a samostatných komponent pro řešení kritických požadavků. (Sommerville, 2013)

5.2 Architektonické vzory

V dnešní době je běžně využíván koncept vzorů, které umožňují prezentovat, sdílet a opakovaně využívat informace v oblasti softwarových systémů. (Sommerville, 2013)

„Architektonický vzor si můžeme představit jako stylizovaný abstraktní popis optimálního postupu, který byl vyzkoušen a otestován v různých systémech a prostředcích.“ (Sommerville, 2013, s. 150)

5.2.1 MVC architektura

Začátkem byl architektonický vzor používaný při vývoji uživatelského rozhraní desktopových aplikací. Dnes je tento vzor především využíván ve webových a mobilních aplikacích. (Svirca, 2020)

Podstata architektonického vzoru MVC je v systematickém rozdělení aplikace do tří hlavních částí: model, pohled a řadič. (Svirca, 2020)

Model Model je označován jako nejnižší úroveň aplikace, což znamená, že je zodpovědný za práci s daty. S daty se pracuje například v podobě přímého napojení s databází, takže všechny operace s daty probíhají přímo v modelu. Model po vykonání operace vrací data zpět řadiči. (Svirca, 2020)

Pohled Pohled se stará o zobrazení dat, tedy vytváří uživatelské rozhraní pro uživatele. Pokud jde o webové aplikace, pohled je zpravidla v podobě HTML a CSS kódu s daty, která jsou shromážděna v modelu. Tato data nejsou přijímána přímo, ale prostřednictvím řadiče. (Svirca, 2020)

Řadič Řadič je hlavní částí architektonického vzoru, protože se stará o propojení mezi pohledy a modely. Řadič se nestará o zpracování dat, pouze zadává úkoly modelu. Po obdržení dat z modelu je zpracuje, vezme všechny informace a pošle je pohledu. (Svirca, 2020)

Tento architektonický vzor přináší několik výhod. Jeho architektura efektivně odděluje uživatelské rozhraní od logiky aplikace, což zajišťuje znovupoužitelnost komponent. Díky tomu je také snadná údržba a jednotlivé komponenty aplikace mohou být nezávisle nasazeny a udržovány. Tato architektura také usnadňuje nezávislé testování jednotlivých komponent. (Svirca, 2020)

Vzor má však i několik nevýhod, jako je vysoká složitost, což může být problematické pro menší aplikace, a také neefektivnost přístupu k datům přímo z pohledu. (Svirca, 2020)

5.2.2 Hexagonální architektura

Hexagonální architektura byla poprvé popsána Alistair Cockburnem. Tento vzor je známý také jako Architektura portů a adapterů. (Tariq, 2022)

V hexagonální architektuře jsou umístěny vstupy a výstupy na okraj návrhu. To umožňuje izolovat centrální logiku aplikace a díky tomu, že vstupy a výstupy jsou na okraji, lze měnit jejich implementaci, aniž by došlo k ovlivnění hlavního kódu. (Tariq, 2022)

Změna implementace může mít například podobu změny databázového systému, změny způsobu komunikace (přechod z REST na GraphQL) a další podobné změny. Pokud architektura byla aplikována správně, žádná z těchto změn neovlivní centrální aplikační logiku.

Hexagonální architektura má tři vrstvy – infrastrukturní, aplikační a doménovou. (Tariq, 2022)

Infrastrukturní vrstva Zajišťuje komunikaci s vnější infrastrukturou aplikace, například řadiči nebo databázemi externích systémů. (Tariq, 2022)

Aplikační vrstva Přijímá požadavky z klientské části (webu nebo API), zpracuje předané hodnoty a následně pošle dál ke zpracování, do doménové vrstvy. (Tariq, 2022)

Doménová vrstva Obsahuje jakoukoli klíčovou funkcionality aplikace. Pracuje pouze s doménovými objekty a nemá informaci o vnějších vrstvách. (Tariq, 2022)

Hexagonální architektura zjednodušuje údržbu aplikace prostřednictvím izolace kódu – změny v jedné komponentě nemají vliv na ostatní komponenty. Díky této izolaci kódu je také usnadněno testování – testy jsou prováděny v izolaci a nezávisle na implementačních detailech z vnějšku. Tato architektura umožňuje rozvoj vnitřního jádra před integrováním externích služeb, jelikož je aplikace nezávislá na vnějších oblastech. (Tariq, 2022)

Nicméně ani hexagonální architektura není ve všem dokonalá. Některé z nevýhod zmiňují problém s odpojením tříd, kdy třídy mezi vrstvami mohou ovlivnit celkový výkon aplikace. Další nevýhodou mohou být obtíže s laděním a pochopením adaptérů a komplexity – může být náročné rozhodnout, které prvky by měly být umístěny na konkrétních vrstvách. (Tariq, 2022)

6. Realizace webové aplikace

Tato část bude věnována realizaci webové aplikace, kdy při tomto procesu budou respektovány informace z předchozích kapitol.

Nejdříve dojde ke specifikaci aplikace, následně k návrhu - architektury, rozhraní i datové vrstvy. Později bude představena implementace serverové a klientské části a na konci kapitoly dojde k validaci, vypuštění aplikace do internetu a budou zde také diskutovány návrhy na další rozšíření.

6.1 Specifikace softwaru

Prvním krokem v etapě realizace webové aplikace bude specifikace softwaru, což představuje jednu z klíčových činností v procesu vývoje. Během tohoto procesu budou prezentovány hlavní funkcionality softwaru odvozené z cíle webové aplikace.

6.1.1 Cíl webové aplikace

Dojde k vytvoření webové aplikace určené pro všechny studenty, kteří jsou zapsáni na univerzitě používající informační systém STAG. Tato aplikace bude postavena na využití dostupných webových služeb poskytovaných univerzitním informačním systémem.

Během zkouškového období se skoro každý student potýká s naplněnou kapacitou zkoušek. Konkrétním příkladem může být předmět *KEM/STA* na Fakultě ekonomické na Západočeské univerzitě v Plzni. Kapacita tohoto předmětu může dosáhnout až 300 studentů za akademický rok, avšak jednotlivé zkouškové termíny jsou omezeny na 30 studentů. Toto omezení vede k okamžitému obsazení termínů zkoušek i týdny předem již v den otevření zápisu. Také nebývá ojedinělé, že pár studentů, pár dní před zkouškou, se z nějakého důvodu odhlásí z daného termínu. Tím zkouškový termín uvolní kapacitu, a protože studenti, kteří by uvolněné místo chtěli, nekontrolují dostatečně, zda se termín zkoušky uvolnil, nedojde ani k jeho opětovnému naplnění.

Řešení nastíněného problému spočívá v implementaci funkce, která umožní studentům nastavit si automatické zapisování na zkouškové termíny. Student si vybere preferovaný termín a systém jeho žádost posoudí a zařadí do fronty ke zpracování. Tímto způsobem může student efektivně řešit problém s plnými zkouškovými termíny bez nutnosti neustálého sledování změn.

Hodně problémů automatizace selže na tom, že pokud dojde k velké adopci webové aplikace, přestane být stěžejní logika aplikace účinná pro všechny studenty (i pro ty studenty, kteří nepoužívají aplikaci). Řešením tohoto možného úskalí je například zavedení nějaké virtuální měny, která formou nejvyšší nabídky určí, na jaké požadavky přijde řada (bude probíhat jakási burza o zařazení požadavku do aktuálního cyklu). Pokud zároveň dojde k omezení, že se zpracuje například pouze 60 % ze všech aktivních požadavků, dojde tímto k omezení přísunu požadavků od uživatelů. Virtuální měny bude ubývat tím, že uživatelé budou chtít jejich požadavky mít zpracovány, čímž ve finále dojde k celkovému poklesu požadavků od uživatelů.

V případě zavedení virtuální měny je důležité přemýšlet nad její distribucí. Vhodné je zvolit formu distribuce tak, aby umožnila přístup všem studentům a zároveň odměňovala aktivní studenty, kteří například pomohou s nějakou univerzitní akcí. Proto bude zaveden výchozí stav této virtuální měny pro každého studenta, který se registruje do aplikace. Zároveň zde bude možnost vystavení tzv. kupónu na další jednotky virtuální měny pro uživatele, kteří budou registrování jako administrátoři. Tito administrátoři mohou být zaměstnanci univerzity, kteří za výpomoc na univerzitních akcí mohou vystavit studentům kupón na tuto virtuální měnu.

Problém s výchozím nenulovým stavem virtuální měny spočívá ve zvýšeném riziku, že uživatelé mohou systém obcházet. Typickým příkladem obcházení systému může být postup, kdy se student zaregistruje do aplikace, vypotřebuje všechny výchozí jednotky virtuální měny, následně si účet vymaže a poté opět zaregistruje. Řešením může být evidence, jaká *Orion* konta se do aplikace historicky zaregistrovala a zakázat všechna již evidovaná.

Cílem je tedy poskytnout studentům snadný a efektivní přístup k automatizaci problémů zkouškového období, a to prostřednictvím uživatelsky přívětivé webové aplikace.

6.1.2 Funkční požadavky

Funkční požadavky pro nepřihlášené uživatele této webové aplikace jsou:

- registrace nového účtu pomocí uživatelského jména, zvoleného hesla a e-mailové adresy,
- obnovení zapomenutého hesla,
- a přihlášení ke stávajícímu účtu.

Funkční požadavky pro uživatele, studenty, této webové aplikace jsou:

- přihlášení a propojení uživatelského účtu k IS/STAG kontu pomocí webové služby univerzity,
- vytvoření nového automatického požadavku k zápisu zkouškového termínu,
- manuální zapsání na zkouškový termín,
- načtení kupónu na virtuální měnu,
- změna jazykové mutace aplikace,
- odhlášení z webové aplikace,
- zobrazení profilu uživatele,
- úprava profilu uživatele - změna jména,
- úprava nastavení účtu - změna hesla, nastavení e-mailové adresy,
- a deaktivace účtu.

Funkční požadavky pro administrátory univerzity jsou:

- změna jazykové mutace,
- odhlášení z webové aplikace,
- zobrazení profilu uživatele,
- úprava profilu uživatele,
- úprava nastavení účtu,
- deaktivace účtu,
- a vytvoření kupónu na virtuální měnu.

Funkční požadavky pro provozovatele aplikace jsou stejné jako pro administrátory univerzity, kromě vytváření kupónu na virtuální měnu a navíc:

- vytvoření administrátora univerzity.

6.1.3 Nefunkční požadavky

Webová aplikace bude:

- vypuštěna na serveru a přístupná uživatelům z webového prohlížeče,
- zabezpečena před možnými útoky, příkladem zabezpečení může být SSL certifikát, šifrování hesel a další techniky,
- snadno použitelná (intuitivní) pro všechny uživatele na všech zařízeních,
- rozšiřitelná o nové funkce,
- otestována,
- a funkční, spolehlivá a výkonově optimalizovaná.

6.2 Návrh webové aplikace

Další klíčovou fází v procesu softwarového vývoje je samotný návrh aplikace, který transformuje specifikace systému do lépe pochopitelných výstupů.

Webová aplikace bude vyvíjena v monolitickém stylu, kde všechny komponenty budou integrovány do jednoho projektu. Alternativním přístupem by mohla být mikroservisní architektura, která rozděluje komponenty do samostatných služeb. Webová aplikace bude rozdělena na dvě hlavní části – klientskou a serverovou (API).

Pro komunikaci mezi klientskou a serverovou částí je předpokládáno využití REST API, o kterém je zmíněno v sekci REST. V určitých případech bude využita technologie i WebSocketů, o které bylo zmíněno v sekci WebSockety.

V rámci této sekce bude podrobněji projednána architektura, rozhraní, komponenty a databáze.

6.2.1 Návrh architektury

Podle definice funkčních požadavků lze odvodit, že webová aplikace bude strukturována do několika komponent (domén).

Komunikace s webovou službou IS/STAG Bude se jednat o komponentu zajišťující správné fungování podstaty webové aplikace. Tato část bude specializována na vzájemnou výměnu informací mezi informačním systémem STAG a realizovanou webovou aplikací. V rámci této komponenty budou pečlivě definovány postupy a způsoby připojení k webové službě, zajišťující efektivní a spolehlivou interakci mezi oběma systémy.

Bezpečnost Bezpečnost představuje klíčovou a nezbytnou složku každé webové aplikace. Tato komponenta bude zodpovědná za implementaci kritických funkcionalit, včetně omezení přístupu na základě udělených oprávnění. Úkolem komponenty je zajistit, aby uživatelé měli pouze oprávněný přístup k funkcím a datům v rámci aplikace.

Správa práv uživatelů představuje klíčový aspekt fungování webové aplikace. Autor bude mít za úkol zajistit efektivní implementaci této funkcionality jak na straně klienta, tak zejména na straně serveru. Tím bude zabezpečeno, že student bude schopen provádět pouze operace, které mu jsou povoleny, a nedojde k neoprávněnému provedení akcí. Tohoto se docílí implementací JWT tokenu, což je zašifrovaný JSON, který nese libovolné informace. Může nést například identifikátor uživatele, dobu expirace nebo práva uživatele.

Uživatelé Komponenta pracující s uživateli bude jedna z důležitých částí webové aplikace. Tato komponenta bude mít jako hlavní úkol vytváření nových uživatelů - studentů i administrátorů univerzity a manipulaci s účtem přihlášeného uživatele.

Zápisy na zkoušku Tato komponenta bude odpovědná za logiku zápisu studentů na termíny zkoušek, student díky komponentě vytvoří nový automatický zápis. Následně aplikace rozhodne, kdy poprvé bude pokus o zápis spuštěn (na základě dat o zkouškovém termínu) a poté tento pokus (úkol) přidá do fronty ke spuštění, kterou si tato komponenta bude udržovat.

Virtuální měna Komponenta starající se o virtuální měnu bude nedílnou součástí aplikace. Tato komponenta bude zajišťovat všechny operace s virtuální měnou - kontrola, navýšení nebo odečtení zůstatku studenta, a zároveň se bude také starat o vystavování a uplatňování kupónů.

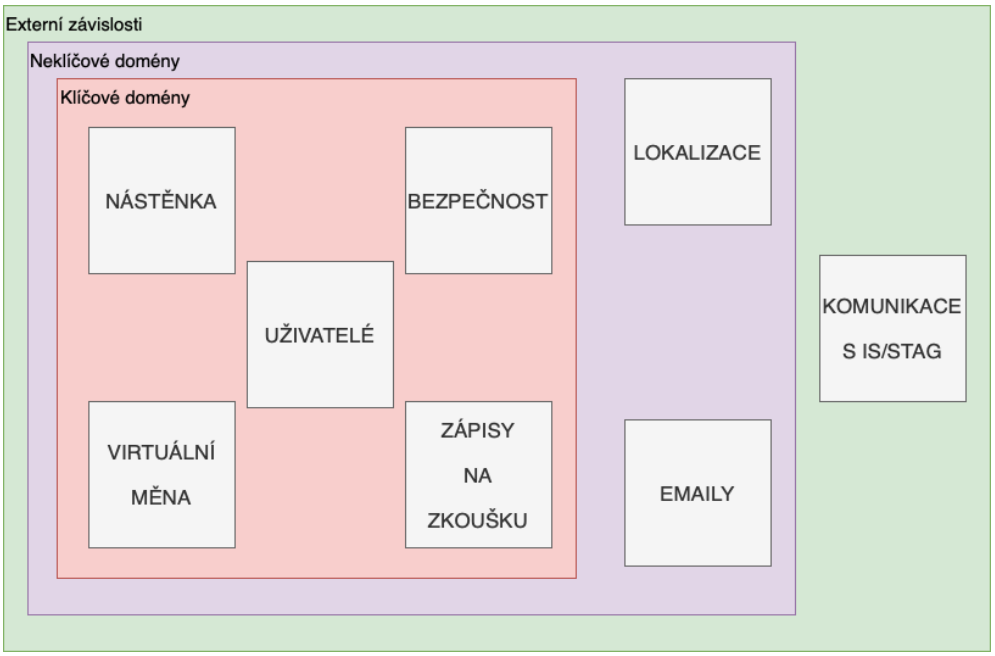
Nástěnka Pro dobrý přehled se zavede také komponenta nástěnky, která se bude starat o agregaci dat z ostatních komponent a následné předání agregovaných dat uživatelům.

Lokalizace Komponenta pro lokalizaci představuje efektivní metodu pro přidávání více jazyků do webové aplikace. Bude se zabývat nastavením zdrojů pro překlady, zajišťováním překladů v reálném čase během zpracování požadavků a dalšími funkcemi souvisejícími s lokalizací.

E-maily Tato komponenta bude zodpovědná za definici e-mailových zpráv, jejich odesílání prostřednictvím SMTP serveru a konstrukci samotných e-mailů.

Na Obr. 5 je zobrazena hierarchie vztahů mezi doménami v rámci webové aplikace. Klíčové aplikační funkcionality jsou reprezentovány doménami *Uživatelé*, *Zápisy na zkoušku*, *Virtuální měna*, *Bezpečnost* a *Nástěnka*. Mezi neklíčové funkcionality patří *Lokalizace* a *E-maily*. Externí závislostí je doména *Komunikace s IS/STAG*. Z perspektivy webové aplikace je prioritou správná funkčnost klíčových komponent. V případě nefunkčnosti neklíčových domén nebo domény *Komunikace s IS/STAG*, může uživatel pociťovat omezení z celkové perspektivy aplikace, přesto aplikace bude fungovat v omezeném rozsahu.

Obrázek 5: Rozdělení domén v souvislosti s aplikací



Zdroj: Vlastní zpracování, 2024

6.2.2 Návrh rozhraní

V této podsekci dojde k definování rozhraní mezi komponentami. Rozhraní lze chápat jako soubor s předdefinovanými funkcemi. Specifikace rozhraní by měla být precizní natolik, aby ji bylo možné využít nezávisle na konkrétním způsobu implementace dané komponenty.

Přístup k návrhu softwaru tak, aby veškerý kód spojený s jednou doménou byl uvnitř této domény se nazývá „Doménově řízený přístup“, zkratkou „DDD“ z anglického výrazu „Domain-Driven Design“. Z těchto principů budou vycházet navrhované datové typy. Proto tato rozhraní budou pojmenována jako „UseCase“.

K definování bude využit přímo datový typ *rozhraní* v jazyce Java.

Komunikace s IS/STAG

V ukázce kódu 1 jsou definovány metody pro manipulaci se zkouškami. Jedná se o velmi důležité rozhraní, které v aplikaci slouží jako vstupní bod do komponenty komunikace s IS/STAG.

Ukázka kódu 1: Rozhraní pro manipulaci se zkouškami

```java
public interface ExamManagementUseCase {
    List<Exam> findStudentExams(String userId);
    String findStudentNumberFromTerm(String userId, long termId);
    Response enrollStudentToExam(String userId, long termId);
}
```

Nepostradatelné je také rozhraní v ukázce kódu 2, to se stará o analýzu a převod relevantních informací z IS/STAG do datových objektů webové aplikace.

Ukázka kódu 2: Rozhraní pro analýzu informací o zkouškách

```java
public interface ExamParserUseCase {
    Exam parseExam(ExamInformation examInformation);
    List<Exam> parseExams(List<ExamInformation> examInformation);
    List<Exam> parseStudentExams(List<StudentExamInformation> examInformation);
}
```

Rozhraní pro informace o webové službě má pouze jednu metodu, *getLoginUrl()*. Tato metoda se stará o zjištění a sestavení přihlašovací URL adresy do webové služby IS/STAG tak, aby měla tato URL adresa všechny potřebné náležitosti, například cílovou adresu pro následné přesměrování zpět do aplikace.

Ukázka kódu 3: Rozhraní pro informace o webové službě

```java
public interface StagGeneralUseCase {
    String getLoginUrl();
}
```

Po přihlášení se do systému IS/STAG dochází k předání informací o studentovi z IS/STAG do webové aplikace formou textového řetězce v Base64 kódování. Aby mohla webová aplikace správně fungovat, je důležité analyzovat a dále zpracovat předané informace. O toto se stará rozhraní, které je v ukázce kódu 4.

Ukázka kódu 4: Rozhraní pro analýzu informací z předaných informací

```java
public interface StagUserInfoParserUseCase {
    List<String> parseStudentNumbers(String base64encoded);
    String parseFirstname(String base64encoded);
    String parseLastname(String base64encoded);
}
```

Poslední, neméně důležité *rozhraní* je zobrazeno v ukázce kódu 5. Tento *UseCase* se stará o propojení informací o autorizaci IS/STAG s uživateli webové aplikace.

Ukázka kódu 5: Rozhraní pro propojení IS/STAG s uživatelem aplikace

```java
public interface UserConnectionUseCase {
    boolean connectStagToUser(TokenInfo tokenInfo, String userId);
    boolean invalidateToken(String userId);
}
```

Bezpečnost

Jak již bylo zmíněno v podsekci Návrh architektury, tato komponenta se bude starat o bezpečnost aplikace. K tomuto účelu jsou v aplikaci čtyři rozhraní: *AuthenticationUseCase*, *TokenExtractionUseCase*, *TokenManagementUseCase* a *TokenValidationUseCase*.

Rozhraní pro přihlášení uživatele zobrazuje rozhraní, které se v jedné metodě stará o veškerou logiku přihlášení uživatele a v druhé o resetování hesla. Vstupní data první metody jsou přihlašovací údaje uživatele, výstupním nativní implementace rozhraní *Authentication*. Vstupními daty druhé, *forgotPassword()*, metody je objekt starající se o přesun dat.

Ukázka kódu 6: Rozhraní pro přihlášení uživatele

```
1 public interface AuthenticationUseCase {
2     Authentication authenticate(String username, String password);
3     void forgotPassword(ForgotPassword forgotPassword);
4 }
```

Dalším klíčovým rozhraní je v ukázce kódu 7. Toto rozhraní se stará o analýzu a převod HTTP požadavku do autorizačního tokenu aplikace.

Ukázka kódu 7: Rozhraní pro zjištění autorizačního tokenu z HTTP požadavku

```
1 public interface TokenExtractionUseCase {
2     String parse(HttpServletRequest httpServletRequest);
3 }
```

Další dvě rozhraní se starají o autorizační token. Rozhraní *TokenManagementUseCase* v ukázce kódu 8 nabízí základní operace s tokenem - vygenerování, zjištění expirace a zjištění identifikátoru uživatele.

Ukázka kódu 8: Rozhraní pro manipulaci s autorizačním tokenem

```
1 public interface TokenManagementUseCase {
2     String generate(String userId, Collection<String> authorities);
3     boolean isExpired(String token);
4     String getUserId(String token);
5 }
```

Druhé rozhraní, v ukázce kódu 9, se stará pouze o zjištění, zda je autorizační token správný.

Ukázka kódu 9: Rozhraní pro ověřování autorizačního tokenu

```
1 public interface TokenValidationUseCase {
2     boolean validate(String token, String userId);
3 }
```

Uživatelé

Rozhraní pro úpravu uživatele zobrazuje funkce starající se o úpravu uživatele. Mezi úpravy patří úprava e-mailové adresy, změna hesla a deaktivace účtu.

Ukázka kódu 10: Rozhraní pro úpravu uživatele

```
public interface UpdateUserUseCase {
    boolean updateUserEmailAddress(String userId, ChangeEmailAddress
        changeEmailAddress);
    boolean updateUserPassword(String userId, ChangePassword changePassword);
    boolean deactivateUser(String userId);
}
```

Rozhraní v ukázce kódu 11 zpracovává požadavky na vytvoření uživatele, definuje metodu pro vytvoření studenta a pro vytvoření univerzitního administrátora.

Ukázka kódu 11: Rozhraní pro vytváření uživatelů

```
public interface UserCreationUseCase {
    boolean registerStudent(CreateStudent createStudent);
    boolean registerUniversityAdmin(CreateUniversityAdmin
        createUniversityAdmin);
}
```

Vhodným pomocným rozhraním bude Rozhraní pro analýzu implementace rozhraní autentifikace, který převádí nativní instanci implementace *Authentication* do doménové entity *User*. Toto rozhraní eliminuje duplicitní kód, který by se musel při každém převodu provádět.

Ukázka kódu 12: Rozhraní pro analýzu implementace rozhraní autentifikace

```
public interface UserParserUseCase {
    User parseUser(Authentication authentication);
}
```

Zápisy na zkoušku

Jediným rozhraní v této komponentě je Rozhraní pro manipulaci s úkoly zápisů. Toto rozhraní poskytuje metody pro zjištění zápisového úkolů daného studenta a vytvoření/odstranění zápisového úkolu.

Ukázka kódu 13: Rozhraní pro manipulaci s úkoly zápisů

```
public interface ManageEnrollmentTasksUseCase {
    List<EnrollmentTask> getAllByUser(String userId);
    boolean createEnrollmentTask(CreateEnrollmentTask createEnrollmentTask,
        String executorId);
    boolean removeEnrollmentTask(RemoveEnrollmentTask registerExamTask, String
        executorId);
}
```

Virtuální měna

V této komponentě jsou tři rozhraní. První rozhraní, Rozhraní pro vytvoření kupónu, zajišťuje vytvoření kupónu k virtuální měně.

Ukázka kódu 14: Rozhraní pro vytvoření kupónu

```
public interface CouponCodeCreationUseCase {
```

```
    String createCouponCode(String issuerId, CreateCodeCoupon
        createCodeCoupon);
}
```

Rozhraní pro inkasování kupónu se stará o zajištění, že uživatel inkasuje platný kupón.

Ukázka kódu 15: Rozhraní pro inkasování kupónu

```
public interface RedeemCouponCodeUseCase {
    int redeem(String userId, RedeemCoupon redeemCoupon);
}
```

Posledním rozhraní je ukázáno v ukázce kódu 16, které se stará o manipulaci s virtuální měnou studenta - poskytuje metody pro kontrolu zůstatku a připsání/odečtení jednotek virtuální měny.

Ukázka kódu 16: Rozhraní pro manipulaci s virtuální měnou

```
public interface ManageUserCoinsUseCase {
    boolean hasEnoughCoins(Student student, int coins);
    boolean hasEnoughCoins(String id, int coins);
    boolean depositCoins(Student student, int coins);
    boolean depositCoins(String id, int coins);
    boolean withdrawCoins(Student student, int coins);
    boolean withdrawCoins(String id, int coins);
}
```

Nástěnka

Každý student bude mít možnost se podívat na přehlednou nástěnku, která bude agregovat nějaká data. O toto se postará Rozhraní nástěnky zápisových úkolů, které má dvě metody, první metodu pro zjištění naposledy spuštěných zápisových úkolů a druhou pro zjištění přehledu o zápisových úkolech.

Ukázka kódu 17: Rozhraní nástěnky zápisových úkolů

```
public interface TaskDashboardUseCase {
    List<EnrollmentTask> getRecentlyProcessedEnrollmentTasks(String userId,
        long count);
    TasksSummary getEnrollmentTasksSummary(String userId);
}
```

Lokalizace

Rozhraní v ukázce kódu 18 zobrazuje metody, které se v rámci aplikace budou používat pro lokalizaci jednotlivých zpráv. Tyto metody na základě vstupního řetězce *messageCode* a, pokud bude potřeba, dalších parametrů zjistí definovaný překlad.

Ukázka kódu 18: Rozhraní pro lokalizaci zpráv

```
public interface LocalizeMessageUseCase {
    String localize(String messageCode);
```

```
3     String localize(String messageCode, Locale locale);
4     String localize(String messageCode, Object[] params);
5     String localize(String messageCode, Object[] params, Locale locale);
6 }
```

E-maily

Rozhraní pro sestavení emailů zobrazuje dvě metody pro správné fungování e-mailových zpráv. První metoda, *buildMessage()*, se stará o sestavení zprávy jako celku. Druhá metoda, *loadMessageContent()*, načítá e-mailovou šablonu a dále s ní pracuje.

Ukázka kódu 19: Rozhraní pro sestavení emailů

```
1 public interface EmailMessageBuilderUseCase {
2     Email buildMessage(BasicMailMessage basicMailMessage);
3     String loadMessageContent(BasicMailMessage basicMailMessage) throws
          IOException;
4 }
```

Rozhraní pro posílání emailových zpráv zajišťuje jednou metodou zasílání e-mailových zpráv. Tato metoda se stane hlavním vstupním bodem pro zasílání e-mailových zpráv z ostatních komponent aplikace.

Ukázka kódu 20: Rozhraní pro posílání emailových zpráv

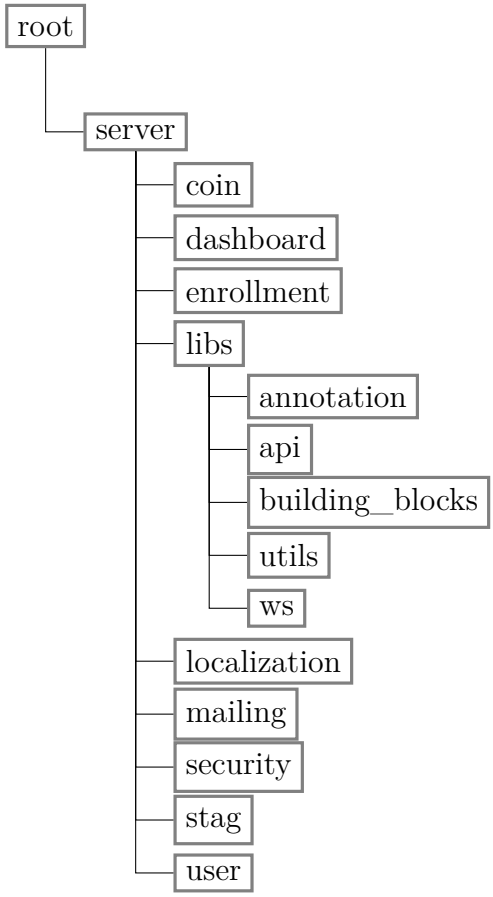
```
1 public interface MailSenderUseCase {
2     boolean send(BasicMailMessage mailMessage);
3 }
```

6.2.3 Návrh komponent

Při návrhu komponent je klíčové zohlednit architekturu webové aplikace. Realizace webové aplikace bude vycházet z inspirace ze stěžejních podnětů hexagonální architektury. Tato architektura byla představena v podsekci s názvem Hexagonální architektura.

Povšechná struktura projektu je prezentována na Obr. 6. Struktura je organizována do jednoho hlavního adresáře pojmenovaného *server*. Tento adresář obsahuje všechny komponenty (domény) zmíněné v Návrh architektury. Dále obsahuje složku *libs*, která slouží jako centrální úložiště pro sdílené knihovny a nástroje, které podporují přesah mezi jednotlivými komponentami systému, a bude obsahovat podadresáře, jako jsou *annotation*, *api*, *building_blocks*, *utils* a *ws*, které pomohou ke strukturovanému přístupu k rozdělení funkcionalit.

Obrázek 6: Povšechná struktura API části aplikace

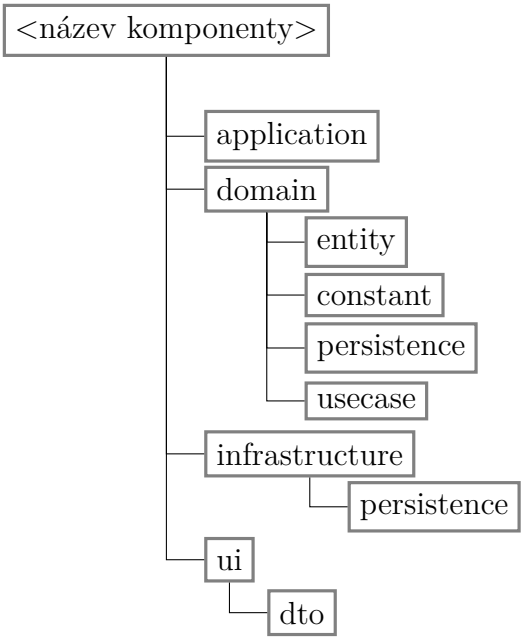


Zdroj: Vlastní zpracování, 2024

Jak bylo v podsekci o hexagonální architektuře zmíněno, rozděluje komponentu (doménu) minimálně do tří vrstev - infrastrukturní, aplikační a doménovou. Při realizaci aplikace bude dodána, pokud to doména bude vyžadovat, ještě jedna vrstva - rozhraní (UI).

Vzhledem k celkovému počtu vrstev v jedné komponentě, by mohla komponenta mít například strukturu zobrazenou na Obr. 7. Na tomto obrázku jsou zobrazeny 4 zmíněné vrstvy: *application* - aplikační (implementující „business“ logiku), *domain* - doménovou (včetně dalších typů souborů, které do doménové vrstvy patří), *infrastructure* - infrastrukturní (implementující logiku operací s databází a dalšími službami) a *ui* - rozhraní (implementující řadiče a přenosové datové třídy).

Obrázek 7: Příklad struktury komponenty (domény)



Zdroj: Vlastní zpracování, 2024

6.2.4 Rozhodnutí o datovém modelu

Tato podsekce se zaměří na výběr datového modelu. Vybraný datový model bude klíčový pro celkovou funkčnost a rozšiřitelnost aplikace, a protože žádný datový model není univerzálně nejlepší, je zásadní vybrat takový, který nejlépe vyhovuje specifikacím a požadavkům aplikace. Jak již bylo zmíněno, aplikace by měla být navržena s důrazem na flexibilitu, umožňující jednoduché přidávání nových komponent a modifikace existujících, aby mohla efektivně reagovat na měnící se požadavky a potřeby.

V objektově orientovaném programování v Javě se k atributům třídy přistupuje skrze koncept inkapsulace. Každý atribut třídy je typicky definován s určitým modifikátorem přístupu, nejčastěji jako private, což zabraňuje přímému přístupu k těmto proměnným z vnějšího kódu. Pro manipulaci s hodnotami těchto atributů jsou využívány *gettery* a *settery*, které poskytují kontrolovaný a bezpečný přístup k proměnným.

Model Entity-Attribute-Value (EAV) je přístup k modelování dat, který se odlišuje od tradičních přístupů tím, že používá tři základní koncepce: *entitu*, *atribut* a *hodnotu*. Tento model umožňuje reprezentaci dat s vysokou mírou flexibility, protože každý atribut je uložen jako záznam, nikoliv jako pevně definovaný sloupec v tabulce. EAV je vhodný pro situace, kde množství a typ atributů entity se může často měnit. Nicméně, s takovou flexibilitou přicházejí kompromisy, včetně složitosti dotazování a potenciálních problémů s výkonem při práci s velkými objemy dat.

S přihlédnutím k očekávanému objemu dat, rozsahu a požadavkům na budoucí rozšiřování funkcí se pro webovou aplikaci zvolí datový model Entity-Attribute-Value (EAV).

6.2.5 Návrh databáze

Vzhledem k datovému modelu a orientaci vyvíjené webové aplikace na práci spíše v reálném čase je vhodné zvolit databázové řešení zaměřené na rychlost a flexibilitu. Pro tyto potřeby se ideálně hodí NoSQL databáze jako MongoDB nebo Cassandra, případně in-memory databáze jako Redis. Tato řešení typicky poskytují rychlejší odezvy a lepší škálovatelnost v situacích, kde je požadována okamžitá reaktivita. Naopak tradiční relační databáze, jako je PostgreSQL nebo MySQL, ačkoli jsou široce používány, mohou za určitých okolností zaostávat ve výkonu a škálovatelnosti.

Po důkladném zvážení dostupných možností a s ohledem na požadavky webové aplikace, bude databázové řešení zprostředkováno NoSQL databází MongoDB. Kromě společných vlastností pro NoSQL databáze, má MongoDB také velkou, aktivní komunitu. To znamená, že jsou dostupné bohatší zdroje pro podporu a integraci, což usnadní budoucí rozšiřování aplikace.

6.3 Implementace serverové části aplikace

Následující část je zaměřena na konkrétní implementaci serverové části webové aplikace. V této části budou představeny kroky před samotným vývojem a následně se tato část věnuje ukázkám řešení vybraných problémů, které vznikly při vývoji.

6.3.1 Příprava před vývojem

Před samotným vývojem je potřeba si připravit všechny potřebné služby, které budou při vývoji použity.

Databáze

V sekci Návrh databáze je zmíněno, proč bylo rozhodnuto pro nerelační způsob uchovávání dat. Nejsnažší a ověřený způsob, jak pracovat s lokální databází, je pomocí služby Docker, zmíněné v sekci 3.2.3. V tomto případě bude využit nástroj nazývaný *Docker Compose* pro orchestraci kontejnerů.

Ukázka kódu 21: Soubor docker-compose.yml s nastavením MongoDB

```
1  version: "3"
2  services:
3    mongodb:
4      image: mongo
5      container_name: mongodb
6      ports:
7        - "27017:27017"
8      volumes:
9        - ./mongodb/data:/data/db
10     environment:
11       - MONGO_INITDB_ROOT_USERNAME=rootuser
12       - MONGO_INITDB_ROOT_PASSWORD=mysecretpassword
```

Díky nastavení v ukázce kódu 21 lze spustit Docker kontejner s MongoDB prostředím, ke kterému se lze přihlásit na adrese *localhost:27017* s přihlašovacím jménem *rootuser* a heslem *mysecretpassword*.

Java Spring Boot

Java Spring Boot představuje open-source nástroj, který zjednodušuje proces vytváření mikroslužeb a webových aplikací pomocí programovacího jazyka Java.

Spring Initializr je nástroj, který usnadňuje inicializaci a konfiguraci nových projektů postavených na Springu. Tato služba umožňuje vytvořit základní kostru webové aplikace pomocí jednoduchého webového rozhraní.

Při inicializaci projektu bude zvolen nástroj Gradle s programovacím jazykem Groovy. Z principu fungování lze očekávat, že bude rychlejší než konkurenční nástroj Maven. Bude specifikována verze Javy 21 a budou vybrány základní závislosti, například pro práci s databází, vytváření REST API nebo zabezpečení.

Spring Boot se následně nastavuje v souboru *application.properties*, nebo *application.yml*. Základní nastavení webové aplikace je zobrazeno na ukázce kódu 22.

Ukázka kódu 22: Základní nastavení Java Spring Bootu

```
1  spring:
2    application:
3      name: "Stagger – Server"
4    data:
5      mongodb:
6        authentication–database: admin
7        host: localhost
8        username: rootuser
9        password: mysecretpassword
10       database: springboot_mongo
11       port: 27017
```

6.3.2 Vybrané implementační problémy

V této podsekci budou zmíněny vybrané implementační problémy, se kterými se autor práce setkal v průběhu realizace serverové části webové aplikace.

Definice datové jednotky

Webová aplikace rozlišuje dvě datové jednotky. První se neukládá do databáze, existuje pouze v paměti aplikace a druhá se do databáze ukládá. V sekci Rozhodnutí o datovém modelu bylo určeno, že bude použit Entity-Attribute-Value (EAV) přístup k datovému modelu.

Byly zavedeny dvě výchozí třídy - *Model*, který nese neukládající se data, a *BDO* nesoucí data (implementace zobrazena na ukázce kódu 23), které jsou určeny pro uložení do databáze. Na základě této charakteristiky mají obě třídy své vlastní (výchozí) atributy, například *ID* nebo údaje o tom, zda data reprezentována touto třídou byla smazána.

Ukázka kódu 23: Implementace třídy BDO

```java
public abstract class BDO<T extends BDO<T>> {

    private String id;
    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    private Long deletedAt;
    public void setDeleted() {
        deletedAt = Instant.now().toEpochMilli();
    }
    public Long getDeletedAt() {
        return deletedAt;
    }

    public boolean isDeleted() {
        return deletedAt != null;
    }

    @Override
    public boolean equals(Object object) {
        if (this == object) return true;
        if (object == null || getClass() != object.getClass()) return false;

        BDO<?> bdo = (BDO<?>) object;

        if (!id.equals(bdo.id)) return false;
        return Objects.equals(deletedAt, bdo.deletedAt);
    }

    @Override
    public int hashCode() {
        int result = id.hashCode();
        result = 31 * result + (deletedAt != null ? deletedAt.hashCode() : 0);
        return result;
    }
}
```

K zavedení dynamické možnosti atributů tak, aby se splnila podstata modelu EAV bylo zapotřebí zavést rozhraní *WithAttributes* s jednou metodou *getAttributes()*. Po zavedení tohoto rozhraní se mohly definovat další dvě třídy, *ModelWithAttributes* a *BDOWithAttributes* (implementace v ukázce kódu 24), reprezentující stejnojmenné třídy bez přípony „WithAttributes“, tentokrát s vlastními atributy.

Ukázka kódu 24: Implementace třídy BDOWithAttributes

```java
public abstract class BDOWithAttributes<T extends BDOWithAttributes<T>>
    extends BDO<T> implements WithAttributes {

    private Map<String, Object> attributes;
    @Override
    public Map<String, Object> getAttributes() {
```

```
6           if (attributes == null) {
7               attributes = new HashMap<>();
8           }
9           return attributes;
10      }
11  }
```

Abychom mohli do jednotlivých modelů ukládat atributy, je podstatné také definovat třídu, která reprezentuje jednotlivé atributy. Implementace takové třídy je na ukázce kódu 25.

Ukázka kódu 25: Implementace třídy Attribute

```
1   public abstract class Attribute<TYPE, ATTRIBUTED extends WithAttributes> {
2
3       private final String name;
4
5       protected Attribute(String name) {
6           this.name = name;
7       }
8
9       public final Object getRawValueFrom(ATTRIBUTED source) {
10          return source.getAttributes().get(name);
11      }
12
13      public TYPE getValueFrom(ATTRIBUTED source) {
14          if (source == null) {
15              return getType().getDefaultValue();
16          }
17
18          Object rawObject = getRawValueFrom(source);
19          return rawObject == null ? getDefaultValue() :
                getType().convertFromRaw(rawObject);
20      }
21
22      public void setValueTo(ATTRIBUTED source, TYPE value) {
23          source.getAttributes().put(name, value);
24      }
25
26      public void setValueTo(ATTRIBUTED source, ATTRIBUTED from) {
27          setValueTo(source, getValueFrom(from));
28      }
29
30      public TYPE getDefaultValue() {
31          return getType().getDefaultValue();
32      }
33
34      public String getName() {
35          return name;
36      }
37
38      public abstract AttributeType<TYPE> getType();
39  }
```

Implementace třídy Attribute zobrazuje abstraktní třídu s typovými parametry *TYPE* a *ATTRIBUTED*. První parametr definuje, o jaký typ atributu se jedná, může se jednat například o textový, celočíselný nebo pravdivostní. Druhý

parametr definuje, k jaké třídě tento atribut patří.

Jako příklad implementace typu atributu může posloužit třída v Implementace textového typu atributu, která zavádí *Singleton* a implementuje dvě metody: *convertFromRaw()* a *getDefaultValue()*.

Ukázka kódu 26: Implementace textového typu atributu

```
public final class StringAttributeType extends AttributeType<String> {

    public static final StringAttributeType INSTANCE = new
        StringAttributeType();

    @Override
    public String convertFromRaw(Object object) {
        if (object == null) {
            return getDefaultValue();
        }
        return (String) object;
    }

    @Override
    public String getDefaultValue() {
        return "";
    }
}
```

Spojením klíčových myšlenek do jednoho celku získáme sofistikovaný mechanismus pro přístup k atributům. Jako příklad demonstrace implementace specifických atributů byly vybrány atributy pro kód kupónu virtuální měny. Příklad této implementace je možné nalézt v ukázce kódu 27.

Ukázka kódu 27: Implementace atributů kupónu na virtuální měnu

```
public abstract class CouponCodeAttribute<TYPE> extends Attribute<TYPE,
    CouponCode> {

    protected CouponCodeAttribute(String name) {
        super(name);
    }

    public static class StringAttribute extends CouponCodeAttribute<String> {

        protected StringAttribute(String name) {
            super(name);
        }

        @Override
        public AttributeType<String> getType() {
            return StringAttributeType.INSTANCE;
        }
    }

    public static class IntegerAttribute extends CouponCodeAttribute<Integer> {

        protected IntegerAttribute(String name) {
            super(name);
        }

```

```
25          @Override
26          public AttributeType<Integer> getType() {
27              return IntegerAttributeType.INSTANCE;
28          }
29      }
30
31      public static class StringListAttribute extends
            CouponCodeAttribute<List<String>> {
32
33          protected StringListAttribute(String name) {
34              super(name);
35          }
36
37          @Override
38          public AttributeType<List<String>> getType() {
39              return StringListAttributeType.INSTANCE;
40          }
41      }
42
43      private static class Holder {
44          private static final StringAttribute CODE = new
                StringAttribute("Code");
45          private static final IntegerAttribute MAX_USAGES = new
                IntegerAttribute("Max Usages") {
46              @Override
47              public Integer getDefaultValue() {
48                  return 1;
49              }
50          };
51          private static final IntegerAttribute VALUE = new
                IntegerAttribute("Value");
52          private static final StringListAttribute USED_BY = new
                StringListAttribute("Used By");
53          private static final StringAttribute ISSUED_BY = new
                StringAttribute("Issued By");
54      }
55
56      public static StringAttribute CODE() {
57          return Holder.CODE;
58      }
59
60      public static IntegerAttribute MAX_USAGES() {
61          return Holder.MAX_USAGES;
62      }
63      public static IntegerAttribute VALUE() {
64          return Holder.VALUE;
65      }
66
67      public static StringListAttribute USED_BY() {
68          return Holder.USED_BY;
69      }
70
71      public static StringAttribute ISSUED_BY() {
72          return Holder.ISSUED_BY;
73      }
74  }
```

Definice chování databázových objektů

Při práci s databázovými objekty může být vhodné mít obecný přístup k definování pravidel. Pro účely webové aplikace byla zavedena třída *BDOType* zobrazena v ukázce kódu 28.

Ukázka kódu 28: Implementace třídy BDOType

```
public abstract class BDOType<TYPE extends BDO<?>> implements
    BDOObservable<TYPE> {

    private final List<BDOObserver<TYPE>> observers = new ArrayList<>();
    private final String name;

    protected BDOType() {
        this.name = extractName();
    }

    @Override
    public void addObserver(BDOObserver<TYPE> observer) {
        observers.add(observer);
    }

    @Override
    public void removeObserver(BDOObserver<TYPE> observer) {
        observers.remove(observer);
    }

    @Override
    public void notifyBeforeInsert(TYPE bdo) {
        for (BDOObserver<TYPE> observer : observers) {
            observer.beforeInsert(bdo);
        }
    }

    @Override
    public void notifyAfterInsert(TYPE bdo) {
        for (BDOObserver<TYPE> observer : observers) {
            observer.afterInsert(bdo);
        }
    }

    @Override
    public void notifyBeforeSave(TYPE bdo) {
        for (BDOObserver<TYPE> observer : observers) {
            observer.beforeSave(bdo);
        }
    }

    @Override
    public void notifyAfterSave(TYPE bdo) {
        for (BDOObserver<TYPE> observer : observers) {
            observer.afterSave(bdo);
        }
    }

    @Override
    public void notifyBeforeDelete(TYPE bdo) {
        for (BDOObserver<TYPE> observer : observers) {
```

```
51              observer.beforeDelete(bdo);
52          }
53      }
54
55      @Override
56      public void notifyAfterDelete(TYPE bdo) {
57          for (BDOObserver<TYPE> observer : observers) {
58              observer.afterDelete(bdo);
59          }
60      }
61
62      protected boolean isSoftDelete() {
63          return false;
64      }
65
66      protected abstract Class<TYPE> getDocumentClass();
67
68      private String extractName() {
69          BDOTypeDefinition annotation =
                getClass().getAnnotation(BDOTypeDefinition.class);
70          if (annotation != null) {
71              return annotation.value();
72          } else {
73              throw new IllegalStateException("BDOTypeDefinition is missing on "
                    + getClass().getSimpleName());
74          }
75      }
76  }
```

Třída *BDOType* implementuje metodu *isSoftDelete()*, která určuje, zda by mělo dojít k trvalému odstranění potomka třídy *BDO* z databáze, nebo by měl být pouze označen časem odstranění pomocí atributu *deletedAt*. Dále tato třída zobrazuje implementaci návrhového vzoru *Observer*, který lze využít pro kaskádové reakce na vyvolané akce.

Logika rolí a práv

Práce s rolemi uživatelů je stěžejní pro fungování realizované aplikace. V sekci Specifikace softwaru byly definovány tři skupiny uživatelů - *Studenti*, *Administrátoři univerzity* a *Provozovatelé aplikace*.

S ohledem na charakteristiku aplikace byly definovány celkem tři třídy - *Authority*, *AuthorityGroup* a *AuthorityManager*. Dále dvě třídy, které drží konstanty textových řetězců - *AuthorityGroupName* a *AuthorityName*.

Ukázka kódu 29 zobrazuje implementaci třídy *Authority*. Tato třída v sobě nese pouze název autority a implementuje metody *equals()* a *hashCode()* pro funkčnost porovnávání jednotlivých instancí.

Ukázka kódu 29: Implementace jedné autority

```
1   public final class Authority implements GrantedAuthority {
2
3       private final String authority;
4
5       public Authority(String authority) {
6           this.authority = authority;
```

```
7       }
8
9       @Override
10      public String getAuthority() {
11          return authority;
12      }
13
14      @Override
15      public boolean equals(Object object) {
16          if (this == object) return true;
17          if (object == null || getClass() != object.getClass()) return false;
18
19          Authority authority1 = (Authority) object;
20
21          return authority.equals(authority1.authority);
22      }
23
24      @Override
25      public int hashCode() {
26          return authority.hashCode();
27      }
28  }
```

Nějaká množina autorit pak může být zahrnuta ve kterékoli *AuthorityGroup*, která definuje jednotlivé uživatelské skupiny. Implementace této třídy je na ukázce kódu 30.

Ukázka kódu 30: Implementace třídy pro skupiny autorit

```
1   public final class AuthorityGroup {
2
3       private final String name;
4       private final Set<Authority> authorities = new HashSet<>();
5
6       public AuthorityGroup(String name) {
7           this.name = name;
8       }
9
10      public String getName() {
11          return name;
12      }
13
14      public void addAuthority(Authority authority) {
15          authorities.add(authority);
16      }
17
18      public Set<Authority> getAuthorities() {
19          return authorities;
20      }
21  }
```

Protože by nebylo efektivní jednotlivé skupiny autorit definovat ve třídě *AuthorityGroup*, byla tato logika vyvedena do třídy *AuthorityManager*. Toto řešení umožní snadnou správu a rozšiřitelnost o další role, případně přidání autorit k jednotlivým rolím.

Ukázka kódu 31: Definice jednotlivých skupin

```
1  public final class AuthorityManager {
2
3      private static final Map<String, AuthorityGroup> groups = new HashMap<>();
4
5      static {
6          AuthorityGroup studentGroup = new
               AuthorityGroup(AuthorityGroupName.STUDENT);
7          studentGroup.addAuthority(new
               Authority(AuthorityName.REDEEM_COIN_COUPON));
8          studentGroup.addAuthority(new
               Authority(AuthorityName.ASSIGN_OWN_STAG_TOKEN));
9          studentGroup.addAuthority(new
               Authority(AuthorityName.INVALIDATE_OWN_STAG_TOKEN));
10         studentGroup.addAuthority(new
               Authority(AuthorityName.CREATE_OWN_ENROLLMENT_TASK));
11         studentGroup.addAuthority(new
               Authority(AuthorityName.REMOVE_OWN_ENROLLMENT_TASK));
12         studentGroup.addAuthority(new
               Authority(AuthorityName.MY_EXAM_ENROLLMENT_TASKS));
13         studentGroup.addAuthority(new Authority(AuthorityName.MY_EXAMS));
14         studentGroup.addAuthority(new
               Authority(AuthorityName.DO_MANUAL_ENROLLMENT));
15         studentGroup.addAuthority(new
               Authority(AuthorityName.MY_RECENTLY_PROCESSED_ENROLLMENT_TASKS));
16         studentGroup.addAuthority(new
               Authority(AuthorityName.MY_ENROLLMENT_TASKS_SUMMARY));
17
18         AuthorityGroup universityAdminGroup = new
               AuthorityGroup(AuthorityGroupName.UNIVERSITY_ADMIN);
19         universityAdminGroup.addAuthority(new
               Authority(AuthorityName.CREATE_COIN_COUPON));
20
21
22         AuthorityGroup productOwnerGroup = new
               AuthorityGroup(AuthorityGroupName.PRODUCT_OWNER);
23         productOwnerGroup.addAuthority(new
               Authority(AuthorityName.REGISTER_UNIVERSITY_ADMIN));
24
25         groups.put(AuthorityGroupName.STUDENT, studentGroup);
26         groups.put(AuthorityGroupName.UNIVERSITY_ADMIN, universityAdminGroup);
27         groups.put(AuthorityGroupName.PRODUCT_OWNER, productOwnerGroup);
28     }
29
30     public static AuthorityGroup getAuthorityGroup(String key) {
31         return groups.get(key);
32     }
33 }
```

Zasílání e-mailů

Pro dobrý uživatelský zážitek bylo nezbytné vytvořit nějakou logiku zasílání e-mailových zpráv. K tomuto účelu byly vytvořeny dvě výchozí třídy, *BasicMailMessage* a *UserMailMessage*. První zmíněná třída je obecnou třídou definující základní metody a atributy. Druhá třída, *UserMailMessage*, rozšiřuje první tím, že určí příjemce e-mailu nějakého uživatele.

Ukázka kódu 32 zobrazuje implementaci e-mailové zprávy o zapomenutém heslu. Tato třída v konstruktoru přebírá příjemce a nové, vygenerované, heslo. Dále implementuje abstraktní metody *getFrom()* - definování odesílatele, *getSubjectTranslationKey()* - definování překladového klíče pro předmět e-mailové zprávy, *getTemplateName()* - definice názvu HTML souboru pro šablonu zprávy a *addVariables()* - umožnění přidání do e-mailové zprávy proměnné v textu.

Ukázka kódu 32: Implementace e-mailové zprávy o zapomenutém heslu

```
public final class ForgotPasswordEmail extends UserMailMessage {

    private final String newPassword;

    public ForgotPasswordEmail(User recipient, String newPassword) {
        super(recipient);
        this.newPassword = newPassword;
    }

    @Override
    public InternetAddress getFrom() {
        return Address.TESTING;
    }

    @Override
    public String getSubjectTranslationKey() {
        return "mailing.user.forgot-password.subject";
    }

    @Override
    public String getTemplateName() {
        return "forgot-password";
    }

    @Override
    protected void addVariables(List<Variable> variables) {
        variables.add(new Variable("new_password", newPassword));
    }
}
```

Z Implementace e-mailové zprávy o zapomenutém heslu lze vyčíst, že zde nedochází k vytváření implementace e-mailové zprávy pro každou jazykovou mutaci zvlášť, toto rozdělení se zpracovává až v rámci *EmailMessageBuilderUseCase*. Implementace takové metody je na ukázce kódu 33.

Ukázka kódu 33: Implementace metody loadMessageContent() z EmailMessageBuilderUseCase

```
    public String loadMessageContent(BasicMailMessage basicMailMessage) throws
        IOException {
        String lang = basicMailMessage.getLocale() != null ?
            basicMailMessage.getLocale().getLanguage() : "";
        if (Tool.isEmptyString(lang)) {
            LOGGER.warn("Asserted default language to Mail Message since
                message's locale was not found");
            lang = AppLocales.DEFAULT.getLanguage();
        }

```

```
8           String htmlContent = Tool.readResource("mails/" + lang + "/" +
                basicMailMessage.getTemplateName() + ".html");
9           for (Variable variable : basicMailMessage.getVariables()) {
10              htmlContent = variable.apply(htmlContent);
11          }
12          return htmlContent;
13      }
```

Zpracování úkolů zapsání na zkoušku

Algoritmus pro zpracování zápisových úkolů bylo potřeba optimalizovat tak, aby nedocházelo k poklesům výkonu a byly volány správné postupy ve správný okamžik.

S ohledem na potenciálně sníženou přehlednost v případě ukázky algoritmu v jazyku Java, je v ukázce kódu 34 použit pseudokód, který algoritmus popisuje. Jedná se o ukázku metody *runRegistrations()*, která se provádí v určité časové okamžiky.

Ukázka kódu 34: Pseudokód pro registrace na zkouškové termíny

```
1   Metoda spustitRegistrace
2       Pokud je zamceno
3           Zapise do logu "Volani spustitRegistrace zruseno kvuli nevyresene
                inicializaci"
4           Vrati se z metody
5
6       Pokud je seznam ukolu registrace prazdny
7           Vrati se z metody
8
9       Nastav fromKey na prvni klic v seznamu ukolu registrace
10      Nastav toKey na aktualni cas
11
12      Pokud fromKey je po toKey
13          Vrati se z metody
14
15      Ziska podmapu navigovatelne mapy od fromKey do toKey a obrati ji // Vrati
            navigovatelnou mapu od aktualniho casu do starsich
16
17      Inicializuj seznam uspesnych ukolu
18      Inicializuj mapu ukolu k pridani po zpracovani
19      Inicializuj mapu informaci o plnosti zkousek
20
21      Pro kazde datetime a procesovatelnou frontu
22          Pokud je fronta prazdna nebo datetime je po aktualnim casu
23              Pokracuj s dalsim
24          Nastav maxSteps na vypocitany pocet kroku pro zpracovani fronty
25          Nastav currentStep na 1
26          Zapise do logu "Zpracovava se X ukolu, bude maximalne Y kroku." kde X
                je velikost fronty a Y je maxSteps
27          Dokud currentStep <= maxSteps a fronta neni prazdna
28              Vyjme ukol z fronty
29              Prida ukol do mapy ukolu k pridani po zpracovani
30
31              Zkontroluje, jestli je zkouska plna
32              Pokud neni plna
33                  Podle vysledku zpracovani ukolu
```

```
                    Pokud je zkouska plna
                        Oznaci zkousku jako plnou
                    Pokud uzivatel jiz byl zapsan
                        Odebere ukol z mapy ukolu k pridani po zpracovani
                    Pokud je vse v poradku
                        Prida ukol do seznamu uspesnych ukolu
                        Odebere ukol z mapy ukolu k pridani po zpracovani
                        Zvysi currentStep o 1

    Konec Pro kazde datetime a frontu v procesovatelne fronte

    Pro kazdou neuspesnou ulohu v mape k pridani
        Prida ukol zpet do fronty

    Pro kazdou uspesnou ulohu
        Odeslat e-mail o uspechu
Konec Metody
```

Neočekávaný způsob odpovědi z univerzitního systému

Během vývoje komunikace s webovou službou IS/STAG autor narazil na nečekané chování v odpovědích z API informačního systému univerzity na určité požadavky.

Při manuálním odesílání požadavků na zápis zkouškových termínů pomocí aplikace Postman, která je představena v sekci Užitečné nástroje, autor zjistil, že univerzitní API vrací HTTP status 200 (úspěch) i v případech, kdy zkouška nebyla zapsána. Původně plánoval autor kontrolovat jen statusy z kategorií úspěchu a chybného požadavku, ale jak se ukázalo, to v tomto případě není dostatečné.

Řešením problému je vytvoření „slovníku“ možných odpovědí, který bude následně porovnáván s reálnou odpovědí z webové služby univerzity. Výčtový typ zodpovědný za uchovávání těchto odpovědí byl pojmenován jako „Response“ a jeho implementace je prezentována v ukázce kódu 35. Tento výčtový typ byl použit také při samotném zpracování úkolů pro jeho jednoduchost při porovnávání výsledků.

Ukázka kódu 35: Výčtový typ pro uchování odpovědí (upraveno)

```
public enum Response {
    OK("OK", true),
    EXAM_FULL("Obsazeni terminu nesmi prekrocit limit terminu", false),
    ALREADY_ENROLLED("Tento termin mate jiz zapsany, provedte Reload
        prohlizece", false, "error.enrollment.already-enrolled"),
    ENROLLMENT_TIME_LIMIT("Na termin se jeste nelze prihlasit - existuje
        casovy limit", false),
    UNENROLLMENT_TIME_LIMIT("Z terminu se jiz nelze odhlasit - existuje casovy
        limit", false),

    // Processor messages
    VALIDATION_ERROR("Nastal problem pred spustenim ukolu", false),
    NOT_IN_RESERVE("Termin nelze zapsat - neni v casove reserve", false),
    UNKNOWN_ERROR("Unknown Error", false);

    private final String message;
    private final boolean successful;
    private final String translationKey;

```

```
17      private static final Logger LOGGER =
            LoggerFactory.getLogger(Response.class);
18
19      Response(String message, boolean successful, String translationKey) {
20          this.message = message;
21          this.successful = successful;
22          this.translationKey = translationKey;
23      }
24
25      Response(String message, boolean successful) {
26          this(message, successful, message);
27      }
28
29      public String getMessage() {
30          return message;
31      }
32
33      public String getMessageTranslationKey() {
34          return translationKey;
35      }
36
37      public boolean isSuccessful() {
38          return successful;
39      }
40
41      public static Response fromMessage(String message) {
42          for (Response response : Response.values()) {
43              if (response.getMessage().equals(message)) {
44                  return response;
45              }
46          }
47          LOGGER.warn("Some method tried to access Response with no existing
                message: " + message);
48          return UNKNOWN_ERROR;
49      }
50  }
```

Stěžejní funkce výčtového typu v ukázce kódu 35 je metoda *fromMessage()*, ta je vstupním bodem například pro kontrolu odpovědi z webové služby. Použití této metody je zobrazeno na ukázce kódu 36, tato metoda se stará o zápis studenta na konkrétní zkouškový termín.

Ukázka kódu 36: Implementace metody zápisu na zkoušku

```
1       public Response registerStudentToExam(String studentNumber, long termId,
            String stagToken) {
2           HttpEntity<?> requestEntity = new
                HttpEntity<>(getAuthorizedHttpHeaders(stagToken));
3           Map<String, String> params = new HashMap<>();
4           params.put("osCislo", studentNumber);
5           params.put("termIdno", String.valueOf(termId));
6
7           String finalUrl = universityApiUrl + PATH +
                "zapisStudentaNaTermin?osCislo={osCislo}&termIdno={termIdno}";
8           try {
9               ResponseEntity<String> response = restTemplate.exchange(finalUrl,
                    HttpMethod.GET, requestEntity, String.class,
                    params.get("osCislo"), params.get("termIdno"));
```

```
10              return Response.fromMessage(response.getBody());
11          } catch (HttpClientErrorException httpClientErrorException) {
12              LOGGER.error("Registering student to exam returned status code %d
                    and message
                    %s".formatted(httpClientErrorException.getStatusCode().value(),
                    httpClientErrorException.getMessage()));
13              return Response.fromMessage(httpClientErrorException.getMessage());
14          }
15      }
```

6.4 Implementace klientské části aplikace

Tato část se věnuje tvorbě klientské části webové aplikace. Na úvodu se věnuje přípravě před zahájením vývoje, dále se přejde k demonstraci konkrétně naprogramovaných pomocných React komponent určených pro aplikaci a nakonec se zaměří na vybrané implementační problémy.

6.4.1 Příprava před vývojem

Před zahájením vývoje klientské části webové aplikace je klíčové zvážit dva hlavní faktory: časovou náročnost spojenou s vývojem vlastního řešení od základu a finanční prostředky dostupné pro projekt. Tyto úvahy jsou nezbytné pro efektivní alokaci zdrojů a plánování projektu. V některých případech může být koupě již existující šablony mnohem ekonomičtější variantou, která umožní rychlejší vývoj a snížení nákladů na implementaci.

V kontextu realizované webové aplikace se autor rozhodl využít již existující šablonu, což umožnilo autorovi se soustředit více na specifika aplikace než na základní strukturu. Pro účely bakalářské práce byla zakoupena šablona „Metronic“ od vývojářů „KeenThemes“. Tato šablona přináší možnost výběru z různých technologií, přičemž autor dal přednost šabloně číslo 1 s využitím technologie React a TypeScript.

6.4.2 Implementace vlastních pomocných komponent

V této podsekci budou představeny dvě vybrané komponenty, které byly implementovány na základě požadavků na funkčnost webové aplikace.

Autorizovaná komponenta

Vzhledem k požadavku na rozlišení přístupu k jednotlivým stránkám a komponentám podle autorizace uživatele, bylo nezbytné zavést mechanismus, který by v případě nedostatečných oprávnění uživatele danou komponentu skryl. Tento přístup zajišťuje, že uživatelé vidí pouze obsah, ke kterému mají oprávnění. Implementace tohoto řešení je předvedena v přiložené ukázce kódu 37.

Ukázka kódu 37: Implementace autorizované komponenty

```
type Props = {
    children?: ReactNode;
    authorities: Authority[]
```

```
    type: 'SOME' | 'ALL'
}

const AuthorizedComponent = ({ children, authorities, type }: Props) => {
    const {hasAnyAuthority, hasAllAuthorities} = useAuth()

    if (type === 'SOME' && !hasAnyAuthority(authorities)) {
        return <></>
    }

    if (type === 'ALL' && !hasAllAuthorities(authorities)) {
        return <></>
    }

    return <>{children}</>;
};

export default AuthorizedComponent
```

Komponenta v Implementace autorizované komponenty přijímá tři parametry: *children*, což je potomek komponenty, který má být zobrazen, pokud jsou splněny podmínky autorizace. Druhým parametrem jsou *authorities*, seznam oprávnění potřebných k zobrazení potomku. Posledním parametrem je *type*, který určuje, zda je pro zobrazení potřeba splnit pouze některé, nebo všechna uvedená oprávnění. Pokud autorizace selže, komponenta vrací prázdný element.

Zobecněné tlačítko

Další komponentou, která byla implementována, je vlastní řešení tlačítka formou komponenty. Přestože se jedná pouze o tlačítko, zobecnění implementace dokázalo ušetřit mnoho času. Tato implementace je zobrazena v Obecná implementace komponenty tlačítka.

Ukázka kódu 38: Obecná implementace komponenty tlačítka

```
interface Props {
    className: string
    iconName?: string
    tooltip?: string
    text?: string
    onClick: () => void
    authority?: Authority
}

export const Button = (props: Props) => {
    const {hasAuthority} = useAuth()
    const intl = useIntl()

    return <button
        className={props.className}
        title={props.tooltip}
        onClick={props.authority && hasAuthority(props.authority) ?
            props.onClick : () => toast.error(intl.formatMessage({id:
                'GENERAL.FORBIDDEN'}))}
    >
```

```
20          {props.iconName && <KTIcon iconName={props.iconName}
                classNameName='fs-3'/>}
21          {props.text && props.text}
22      </button>
23  }
```

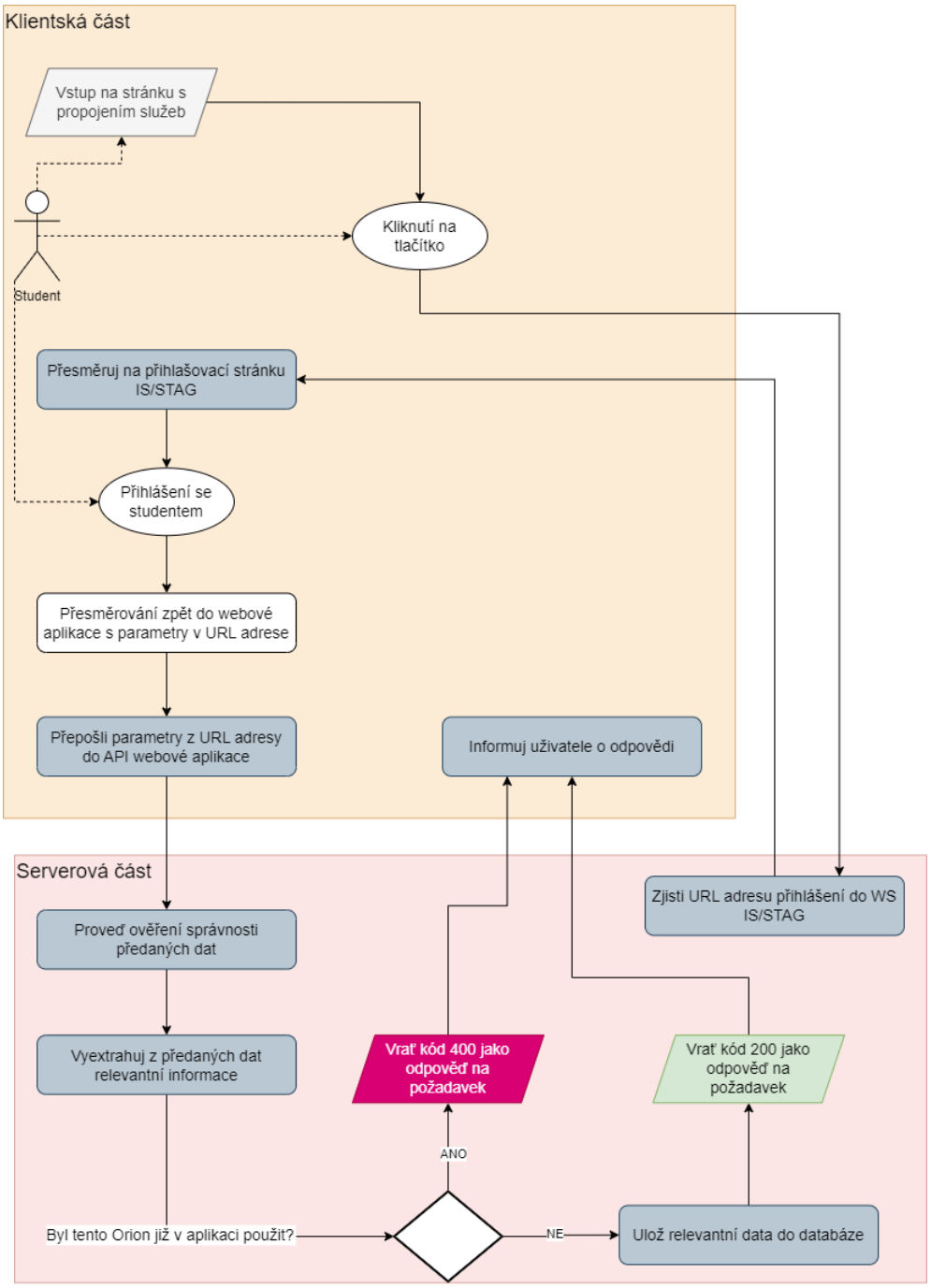
Komponenta zobrazena v ukázce kódu 38 je definovaná rozhraním *Props* a přijímá několik parametrů pro konfiguraci svého chování a vzhledu. Parametr *className* určuje CSS třídu, která bude použita pro stylizaci tlačítka, zatímco *iconName* je nepovinný parametr specifikující název ikony, která má být zobrazena uvnitř tlačítka. Další nepovinný parametr *tooltip* poskytuje text vyskakovacího popisku, který se zobrazí, když uživatel umístí kurzor nad tlačítko. Parametr *text* určuje text, který má být zobrazen na tlačítku. Povinný „callback“ *onClick* definuje funkci, která se vyvolá po kliknutí na tlačítko. Nepovinný parametr *authority* určuje oprávnění potřebná pro aktivaci tlačítka. Pokud uživatel má dané oprávnění, kliknutí na tlačítko vyvolá definovanou akci. Pokud uživatel oprávnění nemá, zobrazí se chybová zpráva o nedostatečném oprávnění.

6.4.3 Vybraný implementační problém

V této podsekci bude představen vybraný problém, a to propojení uživatelského účtu s IS/STAG. Tento problém byl vybrán pro jeho úzkou provázanost obou částí aplikace - klientské i serverové.

Obr. 8 zobrazuje řešení propojení v realizované webové aplikaci. Vstupním bodem je vyčleněná sekce v nastavení účtu. Po kliknutí na tlačítko určené pro propojení dochází ke zjištění přihlašovací adresy do webových služeb univerzitního systému, následně je uživatel přesměrován na tuto adresu, kde se přihlásí jako student. Poté je student odeslán zpět do klientské části aplikace prostřednictvím přihlašovací stránky univerzitních webových služeb s přidanými parametry v URL adrese. Klientská část aplikace detekuje existenci nezbytných parametrů a přeposílá je na server. Server poté prověří validitu těchto dat a extrahuje z nich relevantní informace. Následně se zkontroluje, zda již historicky existuje propojení mezi Orion kontem a nějakým uživatelským účtem. V případě, že takové propojení již existuje, server odpovídá chybovým hlášením, které je následně zobrazeno v klientské části. Jestliže propojení není nalezeno, informace se uloží do databáze a server odesílá klientské části pozitivní odpověď, což je uživateli prezentováno jako úspěch operace.

Obrázek 8: Propojení s IS/STAG



Zdroj: Vlastní zpracování, 2024

6.5 Validace aplikace

Aby se ověřila funkčnost vyvinuté webové aplikace, je nutné pečlivě otestovat všechny její části. Vzhledem k tomu, že aplikace obsahuje pouze jednu klientskou a jednu serverovou část, testování se omezuje na tyto dvě oblasti.

6.5.1 Validace serverové části

Vzhledem k náročnosti architektury serverové části aplikace byla zvolena strategie validace pomocí automatizovaných testů. Tato metodika umožňuje zajištění stálosti a spolehlivosti aplikace, i když dojde k budoucím úpravám či rozšířením funkcionalit.

Hlavním přínosem automatizovaných testů je efektivita a rychlost v ověřování funkčnosti aplikace. Pro potřeby realizované aplikace byl kladen důraz především na jednotkové testy, které jsou klíčové pro ověření funkčnosti jednotlivých komponent izolovaně. Tyto testy byly následně doplněny integračními testy zaměřenými na místa, kde byly předpokládány možné komplikace, aby se tak zvýšila celková robustnost a spolehlivost serverové strany aplikace.

6.5.2 Validace klientské části

Klientská strana aplikace prošla manuálním testováním během vývoje i v závěrečné fázi testování funkcionalit. Autor se soustředil na procházení a zkoumání jednotlivých vyvíjených komponent a stránek, přičemž věnoval značnou pozornost ověřování správnosti jejich chování.

Aby se potvrdila správnost implementace funkce automatických zápisů, bylo nezbytné vyčkat na vypsání předtermínů zkoušek pro studenty na konci studia. Tyto termíny pak autor využil k praktickému ověření očekávaného chování aplikace.

6.5.3 Potenciální úzká místa

Tato podsekce se zaměří na identifikaci a analýzu potenciálních úzkých míst, která by mohla ovlivnit výkon a efektivitu webové aplikace. Pro příklad budou uvedeny dva konkrétní aspekty, které vyžadují pozornost, a bude navržen možný postup pro jejich řešení.

Neefektivní algoritmus automatických zápisů

Potenciálním slabým místem webové aplikace může být algoritmus určený pro automatické zápisy. Neoptimalizovaný algoritmus by mohl vést k zaznamenání významných zpoždění mezi momentem, kdy byl zápis naplánován, a jeho skutečnou realizací. Toto by mohlo podkopat základní účel aplikace, kterým je automatické, efektivní a přesné zpracování zápisů.

Výhodou je, že samotná podstata algoritmu není poskytována třetí stranou, což umožňuje jeho další optimalizaci a hledání nejefektivnějšího řešení. Díky tomu lze algoritmus neustále vylepšovat a přizpůsobovat tak, aby účel aplikace byl splněn.

Pomalá odezva třetí strany

V kontextu produkčního nasazení může dojít k tomu, že rychlost reakce univerzitního systému se stane potenciálním omezením. Přestože by byl algoritmus pro automatické zápisy na zkoušku ve webové aplikaci navržen a optimalizován s maximální efektivitou, stále zůstává fakt, že konzistentní výkon externího, v tomto případě univerzitního systému, nelze plně garantovat.

K odstranění těchto obav by bylo vhodné uskutečnit uživatelské testování s účastí minimálně 100 studentů, během kterého by každý student provedl pokus o zápis na zkoušku za podmínek, které by simulovaly skutečné použití – odeslání požadavku do informačního systému IS/STAG všemi 100 studenty současně. Tento přístup by umožnil ověřit schopnost systému zvládnout nápor a poskytnout rychlou odpověď i v situaci maximální zátěže.

6.6 Vypuštění aplikace do internetu

Pro zprovoznění webové aplikace na internetu je nutné mít k dispozici hardware, software a spojení mezi oběma komponentami.

6.6.1 Rozhodnutí o správě verzí

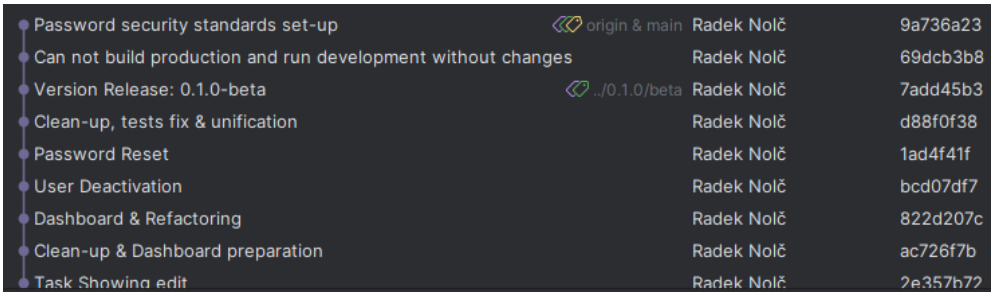
Při přípravě vypuštění webové aplikace bylo nezbytné určit způsob, jakým bude aplikace verzována a identifikována. Pro tento účel autor zvolil standardní schéma číslování verzí, které bylo rozšířeno o specifikaci, zda se jedná o betaverzi aplikace. Tento způsob verzování poskytuje jasný a strukturovaný přehled o vývoji a stavu aplikace v čase.

Konkrétně byla pro účely bakalářské práce vydána verze označená jako 0.1.0--beta. Tento název byl použit nejen pro označení samotné verze, ale i pro pojmenování větve v systému pro správu verzí Git. Takto pojmenovaná větev slouží jako výchozí bod pro další vývoj a přidávání nových funkcí.

V rámci této strategie, jakmile je vyvinuta nová funkce nebo je odstraněna chyba, příslušné změny jsou vloženy do hlavní větve systému Git. V případě, že je nutné vydat novou verzi, vytváří se z hlavní větve nová větev, která nese odpovídající verzi aplikace. Tento způsob verzování a správy kódu přispívá k lepší orientaci ve vývoji aplikace a umožňuje rychlé identifikování a řešení problémů spojených s konkrétními verzemi.

Obr. 9 zobrazuje praktickou ukázku strategie vkládání změn do hlavní větve verzovacího systému Git.

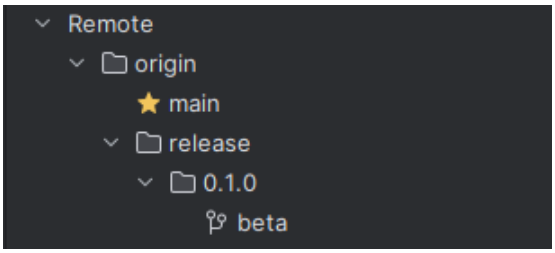
Obrázek 9: Výpis z hlavní větve verzovacího systému



Zdroj: Vlastní zpracování, 2024

V bodě „Version Release: 0.1.0-beta“ na Obr. 9 došlo k vytvoření nové větve umístěné v */release/0.1.0/beta*. Tato nová větev je zobrazena na Obr. 10.

Obrázek 10: Větve ve verzovacím systému



Zdroj: Vlastní zpracování, 2024

Po uvedení verze *0.1.0-beta* byly během závěrečné revize webové aplikace zjištěny některé nedostatky, což vedlo k vytvoření nové verze. Tato nově vydaná verze nese označení *0.1.1-beta* a v rámci této bakalářské práce je považována za finální.

6.6.2 Doména

Před vypuštěním webové aplikace autor zakoupil doménu „stagger.cz“, která bude sloužit jako vstupní bod pro uživatele. Toto řešení bylo vybráno pro možnost využití DNS, to „přesměruje“ uživatele na správnou IP adresu serveru. Ačkoli nákup domény není pro spuštění aplikace nezbytný, výrazně přispívá k lepší dostupnosti a přístupnosti aplikace.

6.6.3 Hardware

Hardware pro vypuštění webové aplikace zajišťuje dedikovaný server pronajatý autorem. Jedná se o server s operačním systémem Ubuntu, vybavený procesorem Intel Xeon a 64 GB operační paměti u společnosti Hetzner.

6.6.4 Software

Pro zajištění vhodného softwarového prostředí bylo rozhodnuto použít Docker pro virtualizaci a Nginx jako webový server.

Jak bylo zmíněno v podsekci 3.2.3, Docker umožňuje izolaci aplikace do kontejnerů, což zjednodušuje nasazení a správu aplikací ve srovnání s tradičními metodami.

Nginx, představen v sekci 1.1, je známý pro svou výkonnost, stabilitu a bohaté možnosti konfigurace, tyto vlastnosti ho činí v kontextu podmínek dedikovaného serveru a realizované aplikace ideální volbou.

Nastavení Nginx

Kód uvedený na ukázce kódu 39 demonstruje příklad nastavení webového serveru Nginx pro provoz na portu 80 (HTTP) na doméně *domena.cz*. Veškeré požadavky jsou přesměrovány do adresáře */usr/share/nginx/html/public/* a zpracovány souborem *index.html*, pokud není specifikován jiný soubor.

Ukázka kódu 39: Příklad nastavení Nginx

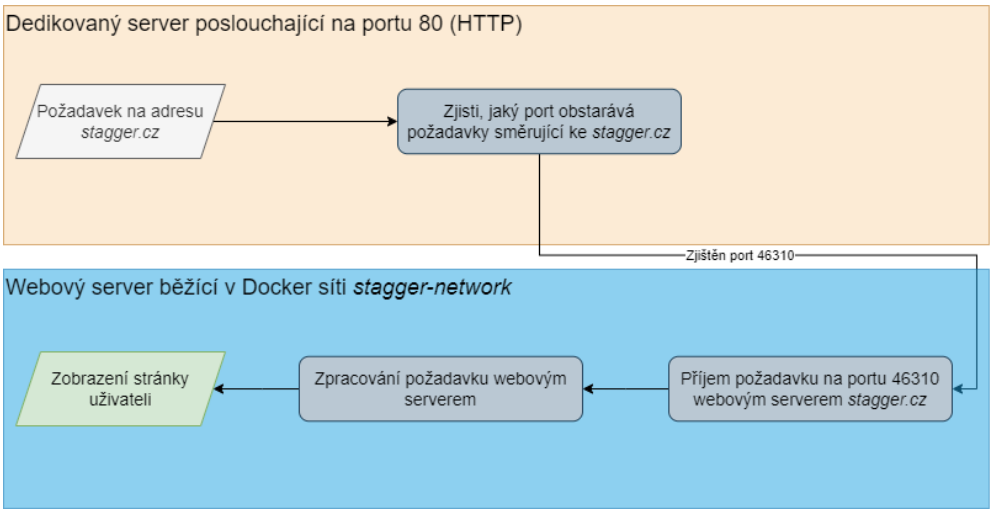
```
server {
       listen 80;
       listen [::]:80;

       server_name domena.cz;

       location / {
           root /usr/share/nginx/html/public/;
           try_files $uri /index.html;
       }
}
```

Vzhledem k charakteru autorova serveru, na kterém jsou provozovány různé aplikace v izolovaných Docker kontejnerech, je nutné upravit konfiguraci.

Kontejnery poskytují izolaci prostředí jako výhodu, avšak při správě webového serveru to může být nevýhoda, protože komplikuje přístup k jednotlivým kontejnerizovaným webovým serverům.

V prostředí Docker kontejnerů lze konfigurovat vnitřní a vnější přístupové porty. Nastavení vnějšího portu na HTTP port by bylo běžně považováno za standardní řešení. Avšak v situaci, kdy server hostuje více aplikací, by tento port mohl být již obsazen jinou aplikací, což vyžaduje úpravu tohoto nastavení. Zjednodušené zobrazení řešení tohoto problému je znázorněno na Obr. 11.

Obrázek 11: Zjednodušené zobrazení směrování do sítě Dockeru



Zdroj: Vlastní zpracování, 2024

Nastavení Dockeru

V kapitole Příprava před vývojem byl zobrazen způsob spuštění lokálního prostředí MongoDB přes kontejner. Abychom mohli webovou aplikaci spustit jako celek, bylo potřeba nastavit a spojit jednotlivé kontejnery. Pro tyto účely byl opět použit nástroj pro orchestraci, *Docker Compose*, pro jeho jednoduchost a přehlednost.

Ukázka kódu 40: Konečné nastavení docker-compose.yml (zkráceno)

```yaml
version: '3.8'
services:
  api:
    build:
      context: server
      dockerfile: Dockerfile
      network: host
    restart: unless-stopped
    environment:
      SPRING_PROFILES_ACTIVE: ${SPRING_PROFILES_ACTIVE}
      SPRING_APPLICATION_NAME: ${SPRING_APPLICATION_NAME}
      SPRING_DATA_MONGODB_AUTHENTICATION_DATABASE:
          ${SPRING_DATA_MONGODB_AUTHENTICATION_DATABASE}
      SPRING_DATA_MONGODB_HOST: mongodb
      SPRING_DATA_MONGODB_PORT: 27017
      [...]
      SETTINGS_CACHING_STUDENT_ACADEMIC_YEAR_GRADES_TTL:
          ${SETTINGS_CACHING_STUDENT_ACADEMIC_YEAR_GRADES_TTL}
    depends_on:
      - mongodb
    networks:
      - stagger-network

  mongodb:
    image: mongo
    restart: unless-stopped
    environment:
      MONGO_INITDB_ROOT_USERNAME:
          ${MONGO_INITDB_ROOT_USERNAME}
      MONGO_INITDB_ROOT_PASSWORD:
          ${MONGO_INITDB_ROOT_PASSWORD}
    networks:
      - stagger-network
    volumes:
      - mongo-data:/data/db

  client:
    build:
      context: client
      dockerfile: Dockerfile
      network: host
    restart: unless-stopped
    volumes:
      - web-data:/app/dist/
    depends_on:
      - api
    networks:
      - stagger-network

  web-server:
    image: nginx:1.25.1-alpine
    restart: unless-stopped
    ports:
      - 46310:80
    volumes:
      - ./nginx/conf/:/etc/nginx/conf.d/:ro
      - web-data:/usr/share/nginx/html/public/:ro
```

```
54      networks:
55        - stagger-network
56  networks:
57    stagger-network:
58
59  volumes:
60    mongo-data:
61    web-data:
```

Konečné nastavení docker-compose.yml (zkráceno) zobrazuje nastavení se čtyřmi službami: API, databází, klientem a webovým serverem.

Každá služba má nastavení *restart: unless-stopped*, které se mimo jiné stará o znovuspuštění v případě restartování dedikovaného serveru. Dalším aspektem, který zajišťuje, je pokus o znovu spuštění v případě pádu služby.

Všechny služby mají nastavení sítě *stagger-network*, které zajišťuje, že všechny kontejnery jsou propojeny do společné interní sítě. Toto uspořádání umožňuje kontejnerům vzájemně komunikovat. Další výhodou sítě je adresování ostatních služeb pomocí jejich jmen. Toto adresování lze vidět na ukázce kódu 40 v nastavení *api* služby na řádku s *SPRING_DATA_MONGODB_HOST*, kde není adresa databáze ve formě IP adresy, ale jako odkaz na službu *mongodb*.

Služby *api* a *mongodb* obsahují nastavení prostředí pomocí proměnných. Příkladem je nastavení *SPRING_PROFILES_ACTIVE*, které určuje aktivní profil serverové části. Hodnota tohoto nastavení se čte ze souboru *.env*, který by měl být umístěn ve stejné složce jako soubor *docker-compose.yml*.

Ukázka kódu 40 také zobrazuje nastínění řešení problému, který byl představen v podsekci Nastavení Nginx, a to problém se směrováním jednotlivých požadavků do Docker sítě a potenciálními konflikty portů. Na ukázce lze vidět nastavení portů služby *nginx*. Toto nastavení udává, že probíhá příjem mimo *stagger-network* síť na portu 46310 a tyto požadavky jsou dále uvnitř sítě brány jako port 80. Protože zde už ke konfliktům zmíněným v Nastavení Nginx nemůže dojít, lze použít nastavení Nginx podobné tomu z ukázky kódu 39.

6.6.5 Vystavení SSL certifikátu

V rámci implementace bezpečnostních opatření webové aplikace bylo potřeba zajistit komunikaci mezi serverem a uživateli prostřednictvím šifrovaného protokolu HTTPS. Pro tento účel bylo rozhodnuto využít bezplatné SSL certifikáty poskytované organizací Let's Encrypt. Pro automatizaci procesu vystavení a obnovy těchto certifikátů byl zvolen nástroj Certbot, který je oficiálně podporovaným klientem Let's Encrypt a byl již předtím instalován na autorově dedikovaném serveru.

Certbot je uznávaný pro svou jednoduchost a efektivitu v automatizaci procesů spojených s SSL certifikáty. Po instalaci nástroje na serveru bylo možné ihned přistoupit k procesu získání SSL certifikátu. Vzhledem k tomu, že na serveru byl provozován webový server Nginx, bylo využito integrace Certbotu s Nginx pro zjednodušení celého procesu.

Autor zahájil proces ověření vlastnictví domény prostřednictvím nástroje Certbot. Po úspěšném ověření domény byl SSL certifikát automaticky vystaven a začleněn do nastavení webového serveru Nginx. Tato automatizovaná metoda mi-

nimalizovala potřebu manuálního zásahu a zajistila chráněnou komunikaci mezi uživateli a serverem.

6.7 Návrhy na další rozšíření

Na začátku realizace byla webová aplikace navržena s cílem usnadnit studentům zapisování na již obsazené zkouškové termíny. Díky zvolené architektuře aplikace je možné naplánovat její budoucí rozvoj a přidání nových funkcí. Mezi potenciální rozšíření této webové aplikace patří:

Vylepšení oznámení Stávající webová aplikace využívá statický systém zasílání oznámení, přičemž všechna upozornění jsou automaticky posílána na e-mail uživatele. Avšak pro zvýšení uživatelského komfortu by bylo vhodné umožnit uživatelům přizpůsobit si oznámení dle vlastních preferencí. To by zahrnovalo nejen výběr specifických událostí, na které chtějí být upozorňováni, ale také volbu způsobu oznámení, ať už prostřednictvím e-mailu nebo SMS zpráv.

Burza zkouškových termínů Dalším častým problémem, který studenti řeší, je něco jako burza zkouškových termínů. Tato burza nastane například tím, že si studenti zapíší termín zkoušky dlouho dopředu a poté zjistí, že by se jim hodil nějaký jiný termín. Většinou to student zjistí, až když jsou všechny vyhovující termíny zkoušky obsazené, čímž je donucen jít na termín zkoušky, který se studentovi příliš nehodí, nebo shánět studenta, který si s ním tento termín vymění. Řešením tohoto může být sekce ve webové aplikaci, ve které uživatelé nabídnou svůj termín výměnou za nějaký jiný termín zkoušky. Aplikace by následně toto prohození sama provedla a studentům by přišlo pouze oznámení o provedené výměně.

Větší možnosti profilu S volbou oznámení souvisí také rozšíření profilových informací. Kromě přidání vlastního telefonního čísla by mohli uživatelé aplikace ocenit také například volbu profilové fotky.

Závěr

Tato bakalářská práce představila vývoj webové aplikace, která využívá kombinaci moderních webových technologií a inovativních přístupů k datovým modelům a ukládání dat. Cílem bylo vytvořit systém, který usnadní studentům zápisy na zkouškové termíny formou automatizovaných zápisů, a tím zefektivnit procesy související se studiem. Práce zkoumala aplikaci nepříliš standardního datového modelu Entity-Attribute-Value (EAV) a využití NoSQL databáze pro ukládání dat.

Díky použití NoSQL databáze MongoDB bylo možné dosáhnout vyšší rychlosti a lepší odezvy aplikace, což je klíčové pro aplikace vyžadující okamžitou odezvu. Model EAV byl implementován s cílem nabídnout dynamické schéma, které může efektivně reagovat na změny bez nutnosti přepracování celé databázové struktury.

Během návrhu architektury byla zvolena inspirace hexagonální architekturou. Tato architektura poskytuje výhody v oddělení závislostí mezi různými částmi aplikace, což vede k vyšší testovatelnosti a snadnější údržbě systému.

Během vývoje byl kladen důraz na testování a optimalizaci aplikace, což zahrnovalo jednotkové a integrační testy serverové části, aby bylo zajištěno, že všechny komponenty spolehlivě fungují jak samostatně, tak ve vzájemné interakci. Kromě toho byly prováděny manuální testy klientské části, které prováděl autor práce. Tyto testy zahrnovaly ověřování funkcionality systému s reálnými zkouškovými termíny pro končící studenty, tento přístup k testování pomohl identifikovat a opravit potenciální problémy před nasazením aplikace do produkčního prostředí.

Celkově tato práce demonstruje, jak lze teoretické koncepty a metodiky efektivně aplikovat při vývoji webové aplikace, která je připravena na další vývoj a rozšíření.

Seznam použitých zkratek

API Application Programming Interface

CSS Cascading Style Sheets

DDD Domain-Driven Design

DNS Domain Name System

EAV Entity-Attribute-Value

GPS Global Positioning System

HTML HyperText Markup Language

HTTP HyperText Transfer Protocol

HTTPS HyperText Transfer Protocol Secure

IDE Integrated Development Environment

IoT Internet of Things

IP Internet Protocol

IS/STAG Information System of the Study Agenda

JSON JavaScript Object Notation

JWT JSON Web Token

MVC Model-View-Controller

NFC Near Field Communication

NoSQL Not Only SQL

OS Operating System

PHP PHP: Hypertext Preprocessor

QL Query Language

REST Representational State Transfer

SOAP Simple Object Access Protocol

SQL Structured Query Language

SSL Secure Sockets Layer

URL Uniform Resource Locator

WS Web Service

WORA Write Once, Run Anywhere

XML eXtensible Markup Language

Literatura

Alonso, G., Casati, F., Kuno, H., & Machiraju, V. (2003). *Web services: Concepts, architectures and applications*. Springer Science & Business Media.

Boehm, B. W. (1988). A spiral model of software development and enhancement. *Computer*, 21(5),61–72.

Cocca, G. (2023). Different types of apis – soap vs rest vs graphql. `https://www.freecodecamp.org/news/rest-vs-graphql-apis/`. Dostupné online 18. 12. 2023.

Docker (2023). Docker overview. `https://docs.docker.com/get-started/overview/`. Dostupné online 20. 12. 2023.

Gillis, A. (2020). What is a web server and how does it work? `https://www.techtarget.com/whatis/definition/Web-server`. Dostupné online 29. 12. 2023.

Gillis, A. (2021). What is object-oriented programming (oop)? `https://www.techtarget.com/searchapparchitecture/definition/object-oriented-programming-OOP`. Dostupné online 18. 12. 2023.

Harish, M. (2023). Node.js vs. php vs. python: Which backend technology to choose in 2023? `https://infostride.com/node-js-vs-php/`. Dostupné online 18. 12. 2023.

Heller, M. (2022). What is visual studio code? microsoft's extensible code editor. `https://www.infoworld.com/article/3666488/what-is-visual-studio-code-microsofts-extensible-code-editor.html`. Dostupné online 20. 12. 2023.

Hernandez, M. J. (2006). *Návrh databází (2. vyd., J. Bouda, překl.)*. Grada Publishing.

Hooda, P. (2022). Introduction to postman for API development. `https://www.geeksforgeeks.org/introduction-postman-api-development/`. Dostupné online 20. 12. 2023.

Microsoft (2022a). WebSockets. `https://learn.microsoft.com/en-us/windows/uwp/networking/websockets`. Dostupné online 7. 4. 2024.

Microsoft (2022b). What is Git? `https://learn.microsoft.com/en-us/devops/develop/git/what-is-git`. Dostupné online 16. 1. 2024.

Microsoft (2023). Common client-side web technologies. `https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-client-side-web-technologies`. Dostupné online 12. 12. 2023.

Olawanle, J. (2022). What is a framework? software frameworks definition. `https://www.freecodecamp.org/news/what-is-a-framework-software-frameworks-definition/`. Dostupné online 14. 1. 2024.

Popovych, A. (2023). Web application architecture: How web apps work. https://clockwise.software/blog/web-application-architecture/. Dostupné online 14. 1. 2024.

Red Hat (2019). What is an IDE? https://www.redhat.com/en/topics/middleware/what-is-ide. Dostupné online 16. 1. 2024.

Rossman, B. (2010). *Application lifecycle management - Activities, methodologies, disciplines, tools, benefits, alm tools and products*. Emereo Publishing.

Shields, K. (2023). 9 great web application examples. https://designli.co/blog/5-great-web-application-examples/. Dostupné online 29. 11. 2023.

Simplilearn (2023). What is Intellij IDEA - a comprehensive guide. https://www.simplilearn.com/what-is-intellij-idea-article. Dostupné online 20. 12. 2023.

Sommerville, I. (2013). *Softwarové inženýrství (J. Goner, překl.)*. Computer Press.

Stejskal, J. (2004). *Vytváříme WWW stránky pomocí HTML, CSS a JavaScriptu*. Computer Press.

Svirca, Z. (2020). Everything you need to know about MVC architecture. https://towardsdatascience.com/everything-you-need-to-know-about-mvc-architecture-3c827930b4c1. Dostupné online 20. 12. 2023.

Tariq, A. (2022). Hexagonal architecture design pattern. https://www.mitrais.com/news-updates/hexagonal-architecture-design-pattern/. Dostupné online 20. 12. 2023.

Tran, T. (2021). Top 7 server-side scripting languages. https://www.orientsoftware.com/blog/server-side-scripting-languages/. Dostupné online 12. 12. 2023.

Seznam obrázků

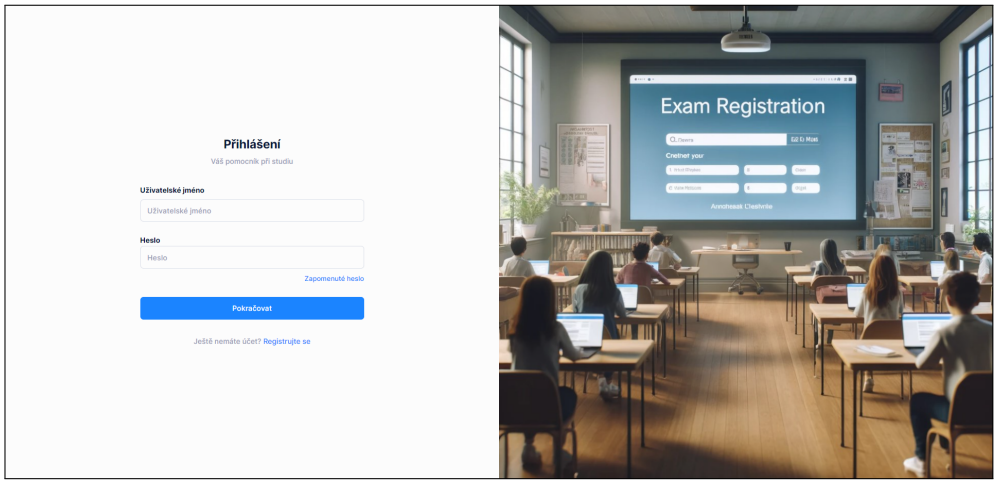
Seznam ukázek kódu

Seznam příloh

A Dokumentace realizované webové aplikace

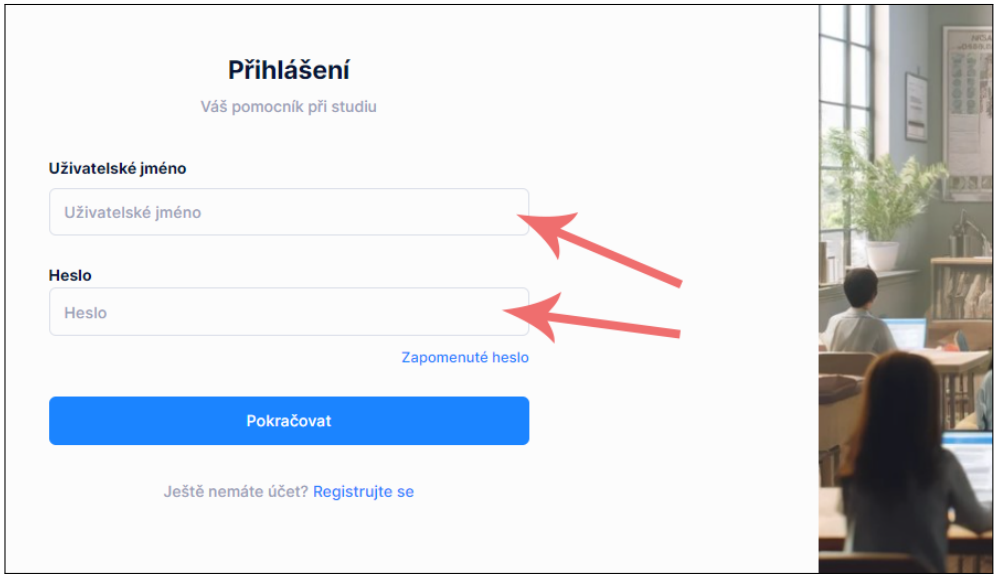
Společná dokumentace

Tento oddíl dokumentace je určen univerzálně pro všechny typy uživatelů a poskytuje základní informace o používání stránky pro autentifikaci. Stránka je navržena s běžně vyskytujícím se rozdělením, kdy je stránka vertikálně rozdělena na dvě části. V první polovině se nachází obsah stránky - může se jednat o přihlašovací formulář, formulář s obnovením hesla, nebo registrační formulář, zatímco druhá polovina obsahuje vizuální prvek.



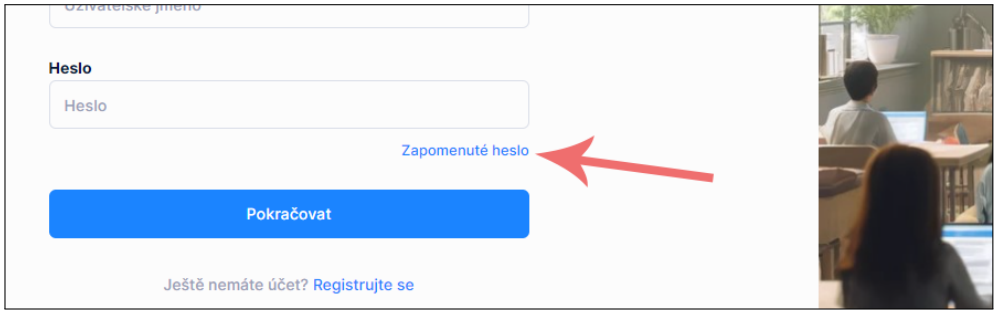
Přihlášení uživatele

Pro vstup je nutné zadat uživatelské jméno a heslo, které jste si určili během procesu registrace. Tyto údaje je po vyplnění potřeba potvrdit kliknutím na tlačítko „Pokračovat“.

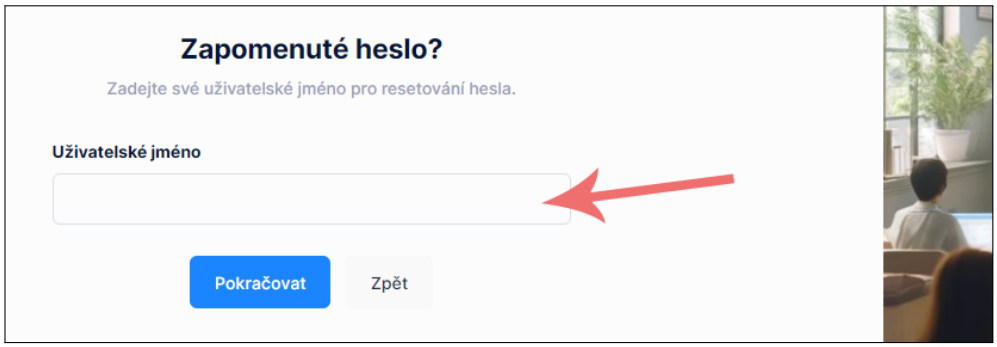


Zapomenuté heslo

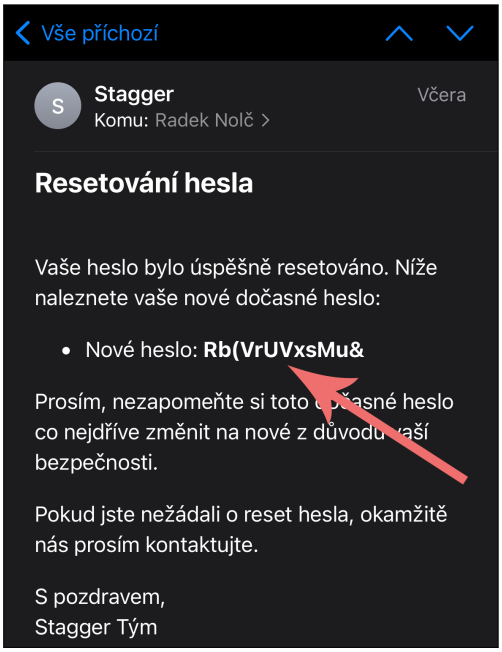
Pokud jste zapomněli heslo, můžete si ho snadno obnovit kliknutím na „Zapomenuté heslo“ na stránce s přihlášením.



Po kliknutí na zmíněný odkaz budete přesměrování na stránku s obnovením hesla, která Vás vyzve k vyplnění uživatelského jména.



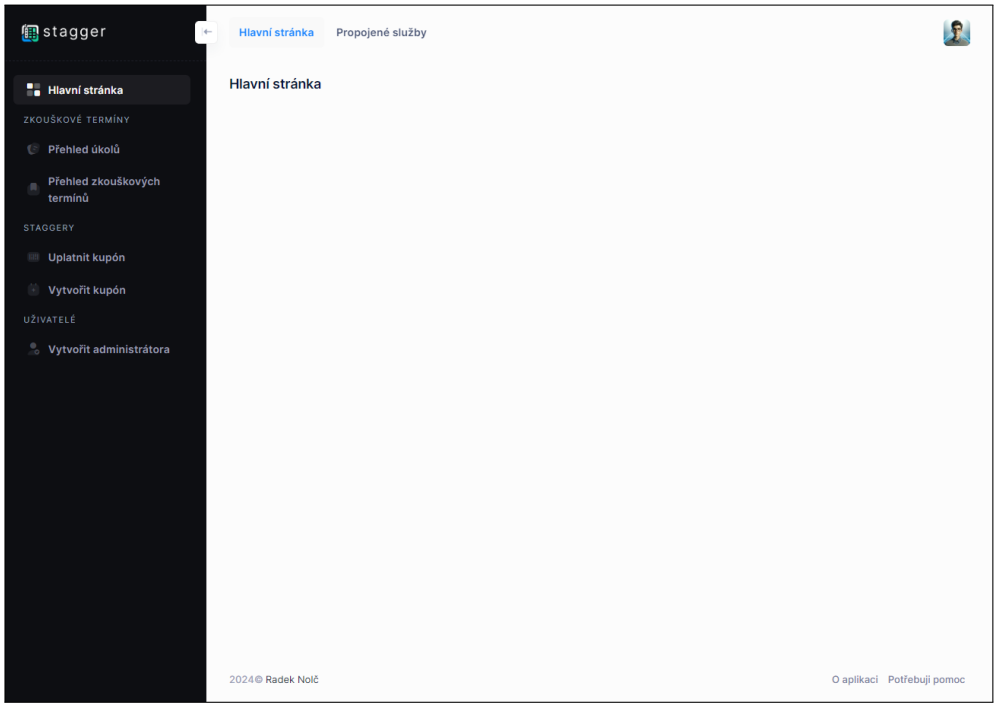
Vyplňte uživatelské jméno a stiskněte tlačítko „Pokračovat“. Po úspěšném odeslání formuláře přejděte do e-mailové schránky spojené se zadaným uživatelským jménem, měli byste zde nalézt e-mail od aplikace.



V e-mailové zprávě o resetování hesla Vám přijde náhodně vygenerované heslo, které nyní můžete použít k přihlášení. Nyní můžete přejít zpět do webové aplikace a přihlásit se. Pokud jste stále na stránce s obnovením hesla, stiskněte tlačítko „Zpět“.

Rozložení obsahu v přihlášené aplikaci

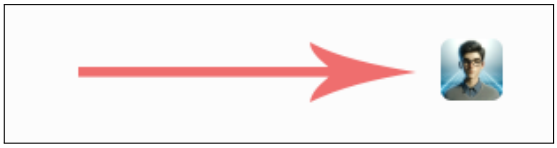
Po přihlášení nabízí webová aplikace typické rozvržení. Na levé straně se nachází navigační menu, kde jsou pro ilustraci zobrazeny všechny položky bez ohledu na uživatelská oprávnění. V horním panelu na levé straně je umístěno menu pro rychlý přístup, zatímco na pravé straně se nachází ikona uživatele, která otevírá další možnosti. Středová část obrazovky zaujímá obsah, v tomto případě se jedná o „Hlavní stránku“, která je pro ilustraci vyčištěna o komponenty.



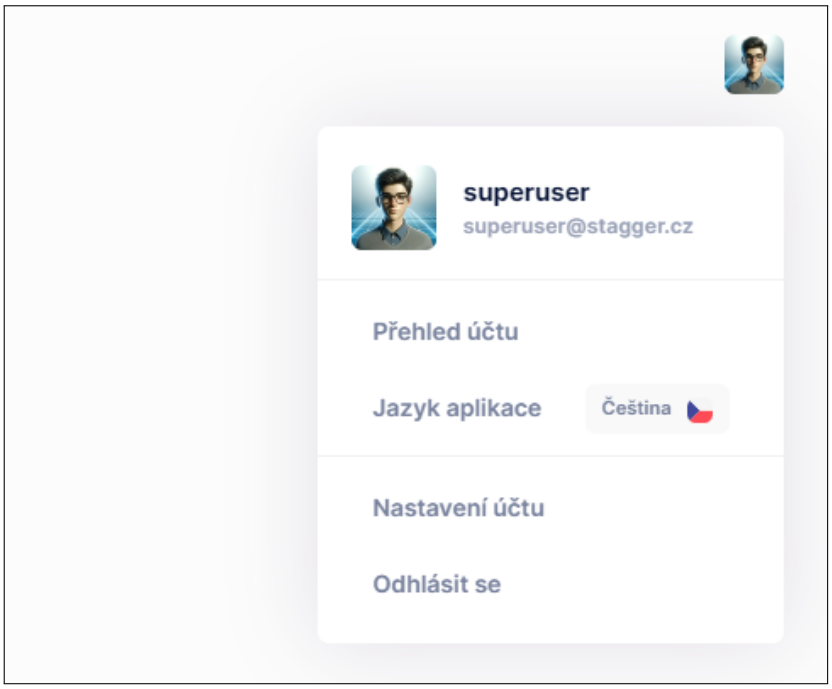
Běžnou součástí, se kterou se setkáte, je hlavička účtu. Pro názornost je tato komponenta zobrazena bez konkrétních dat, avšak na místech, kde jsou umístěné šipky, uvidíte různé informace. U první (horní) šipky mohou být zobrazena různá čísla, zatímco ve druhém případě šipka ukazuje na dynamicky vkládanou navigaci mezi podstránkami.

Uživatelské menu, přehled a nastavení účtu

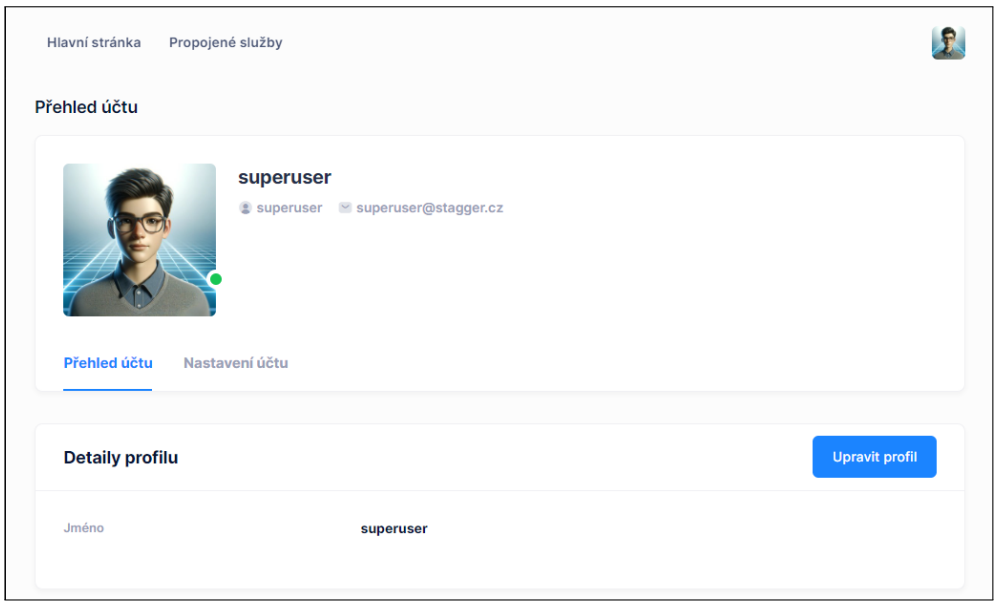
Pro otevření uživatelského menu klikněte na Váš profilový obrázek v pravém horním rohu.



Uživatelské menu, jak je vidět na přiloženém obrázku, poskytuje odkazy související s vaším účtem, konkrétně možnosti „Přehled účtu“ a „Nastavení účtu“, dále umožňuje nastavit jazyk aplikace a obsahuje tlačítko pro odhlášení.



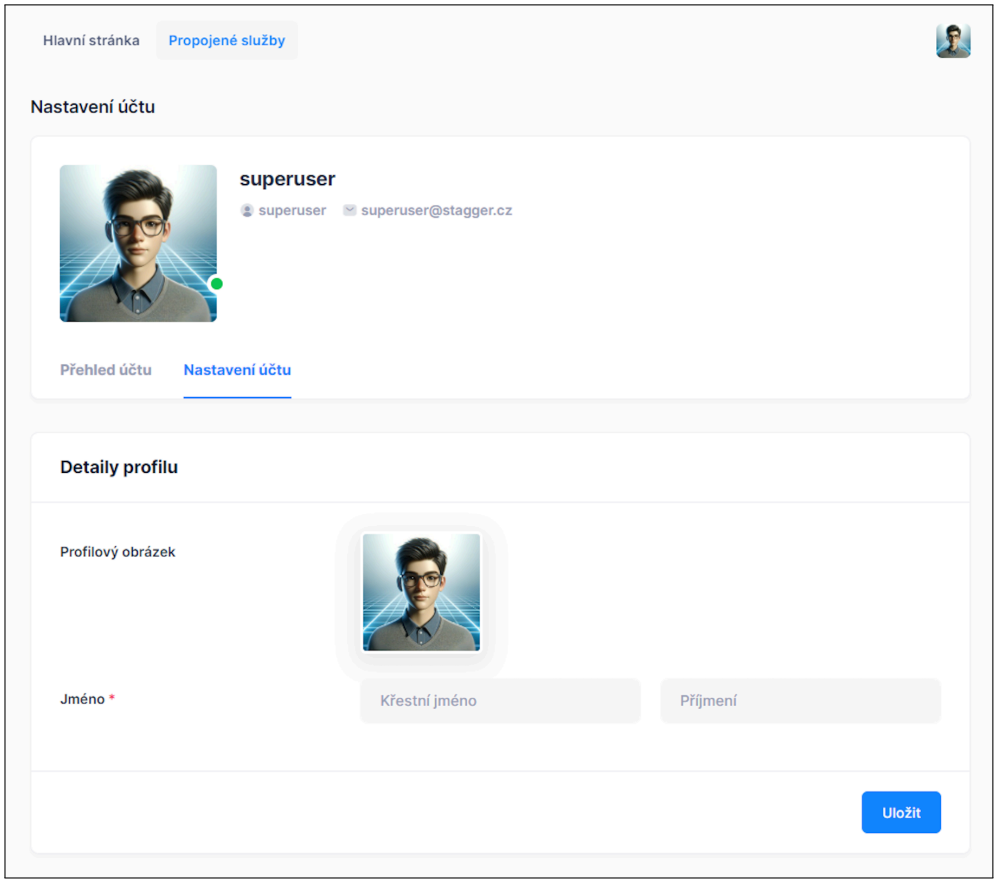
Pro přístup do přehledu účtu klikněte na tlačítko „Přehled účtu“ v uživatelském menu. Zobrazí se Vám přehled účtu s detaily profilu.



V aktuální verzi aplikace má jeden účet přiřazeno pouze jméno, které si můžete upravit kliknutím na „Upravit profil“, nebo přepnutí navigace hlavičky na

„Nastavení účtu“.

Na stránce pro nastavení účtu máte možnost upravit své jméno a příjmení v části „Detaily profilu“. Při dalším posunu dolů na stránce objevíte více sekcí, mezi které patří „Základní údaje“, „Propojené služby“ a „Deaktivace účtu“. V sekci „Základní údaje“ je možné změnit e-mailovou adresu a heslo pro přístup k účtu.



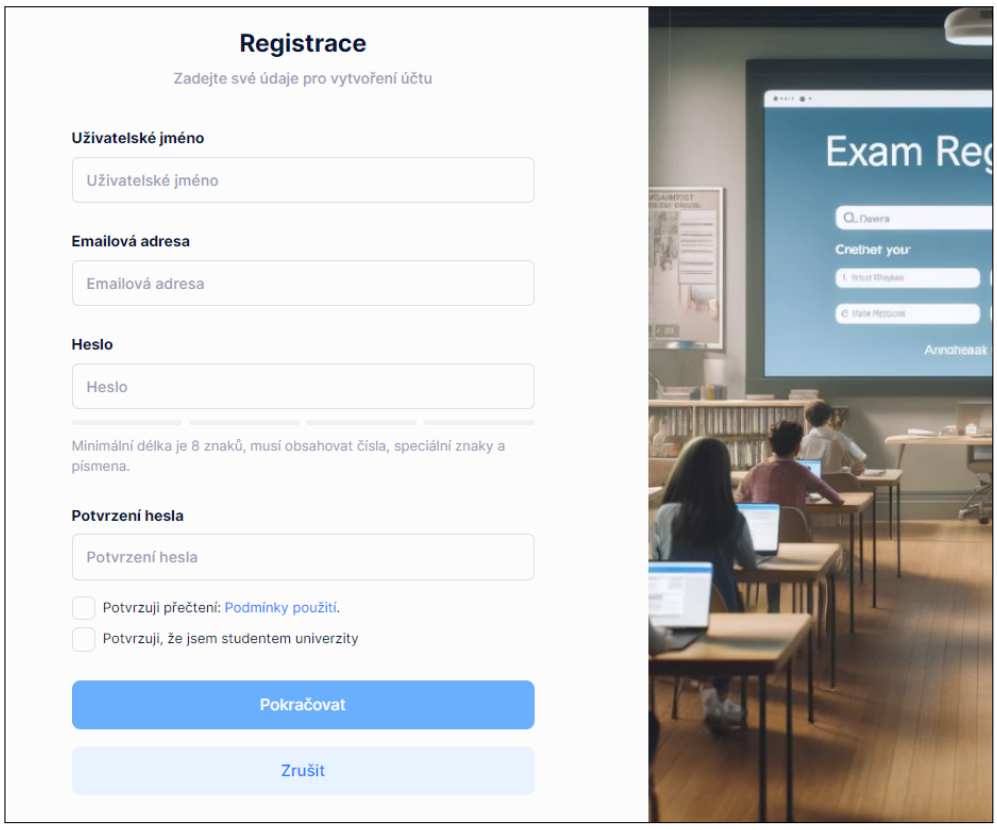
Dokumentace pro studenty

Tato část dokumentace je specificky zaměřena na registrované studenty a vysvětluje různé možnosti, které mají studenti při používání této aplikace k dispozici.

Registrace do aplikace

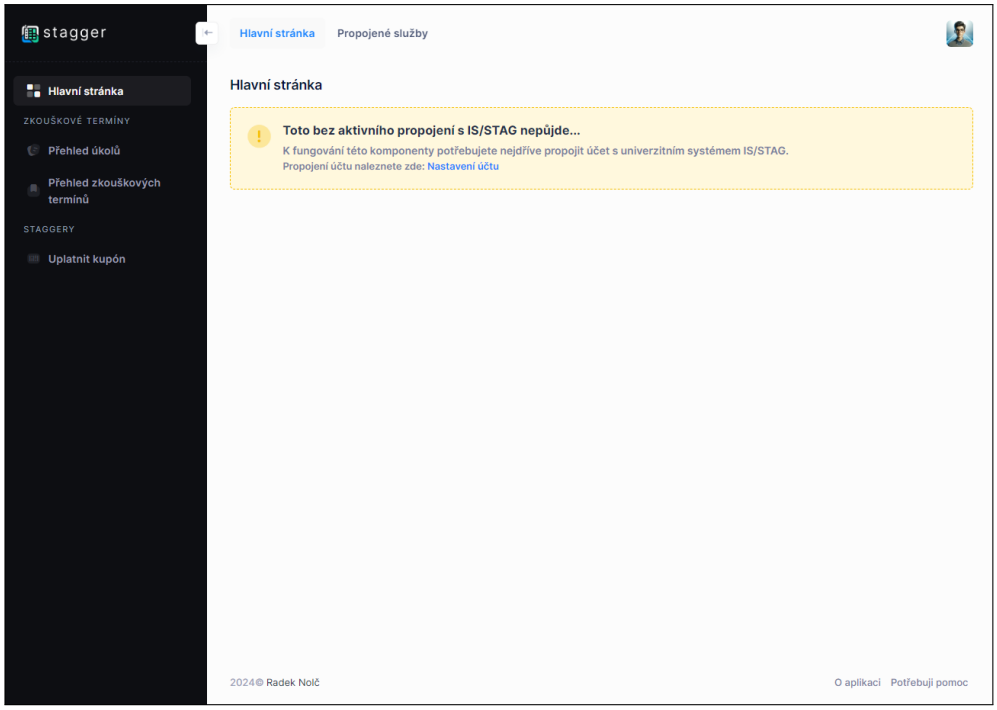
Pro přístup do aplikace se musíte nejdříve registrovat, to provedete kliknutím na odkaz „Registrujte se“ na přihlašovací stránce.

Tento odkaz Vám otevře registrační formulář, kde vyplňte potřebné údaje. Je potřeba vyplnit uživatelské jméno, které musí být v systému unikátní, Vaši e-mailovou adresu, která bude sloužit pro veškerou komunikaci včetně oznámení z aplikace, dostatečně silné heslo a potvrzení hesla. Poté potvrďte přečtení podmínek použití a to, že jste studentem.



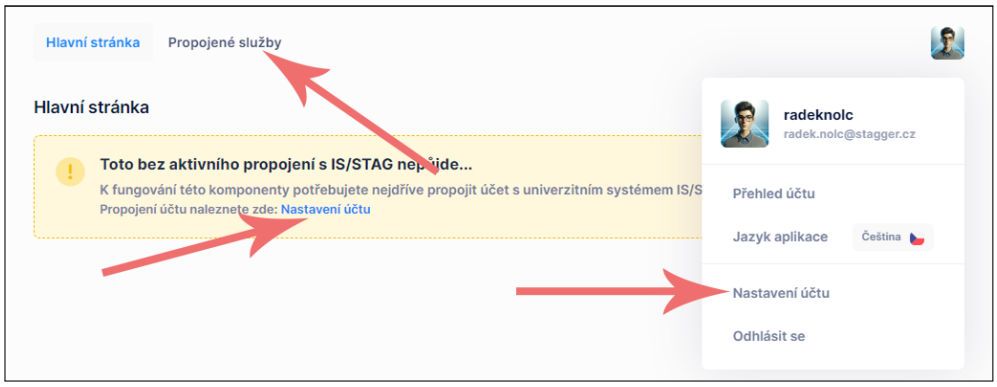
Hlavní stránka aplikace

Po Vašem prvním přihlášení do aplikace se vám na úvodní stránce zobrazí nástěnka s upozorněním, že nemáte propojený účet IS/STAG.

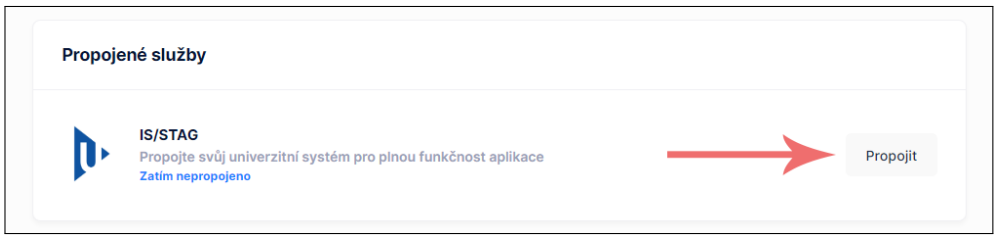


Propojení s IS/STAG

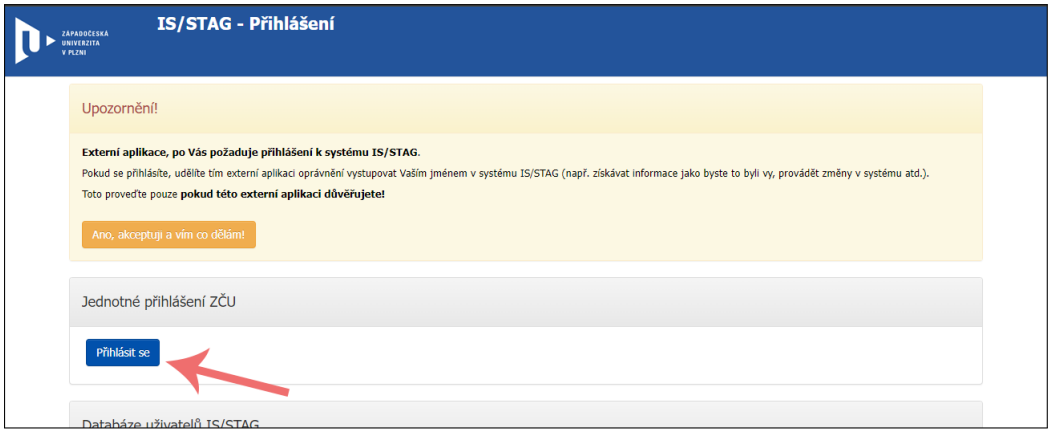
Aby nebyla omezena funkčnost aplikace, je potřeba propojit Váš účet s účtem IS/STAG. Přejděte do „Nastavení účtu“.



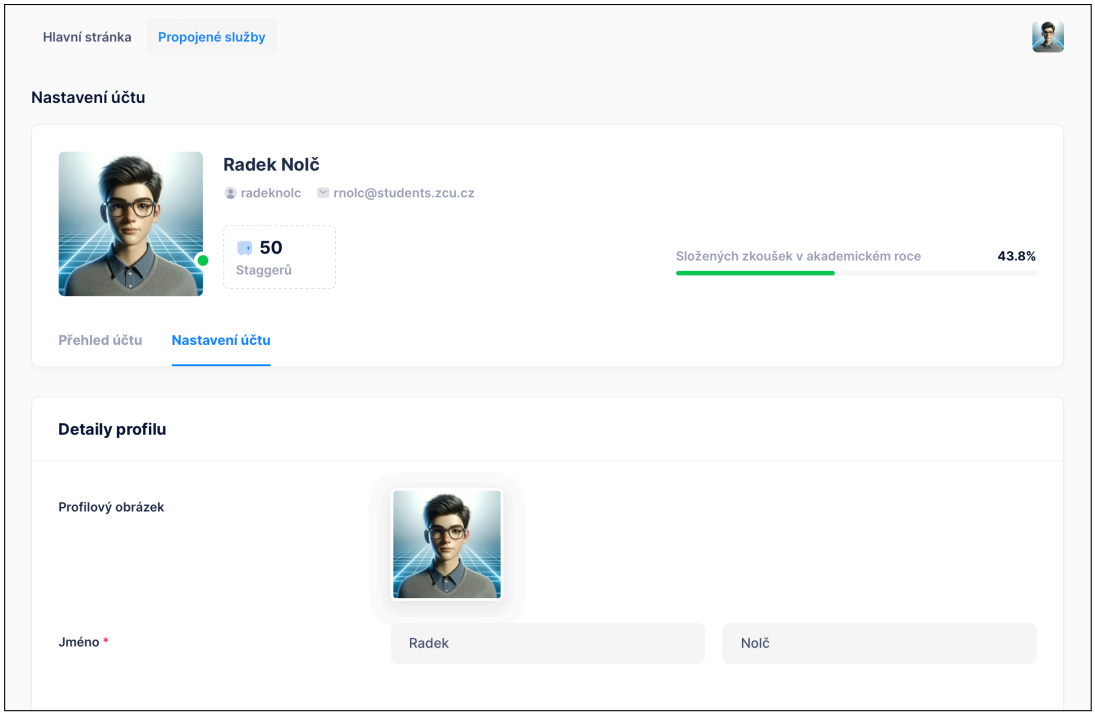
Po přesunutí se do nastavení, posuňte se k sekci „Propojené služby“ a klikněte na tlačítko „Propojit“.



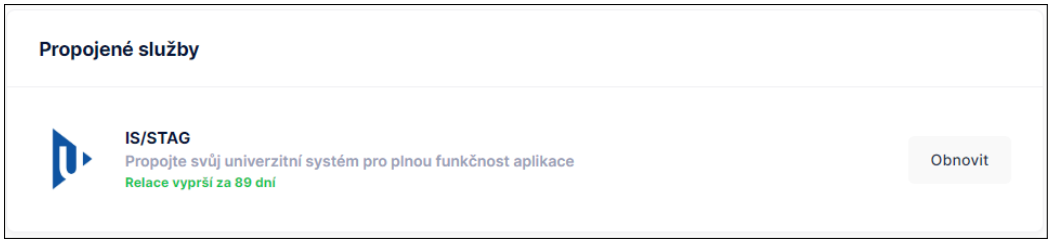
Budete přesměrováni na oficiální stránku přihlášení k univerzitnímu systému IS/STAG. Zde klikněte v sekci „Jednotné přihlášení ZČU“ na tlačítko „Přihlásit se“ a postupujte podle pokynů.



Po úspěšném přihlášení budete přesměrování zpět do webové aplikace. Pokud jste si již dříve nevyplnili Vaše jméno, automaticky se Vám nastavilo ze systému IS/STAG.

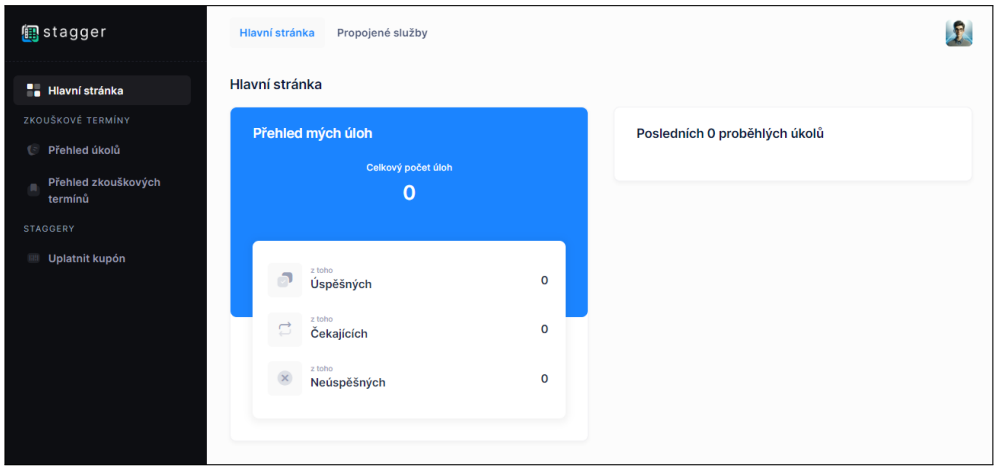


Pokud se nyní posunete k sekci „Propojené služby“, uvidíte aktualizovaný panel „IS/STAG“ s údaji o expiraci relace. V této sekci si můžete později relaci prodloužit kliknutím na tlačítko „Obnovit“.

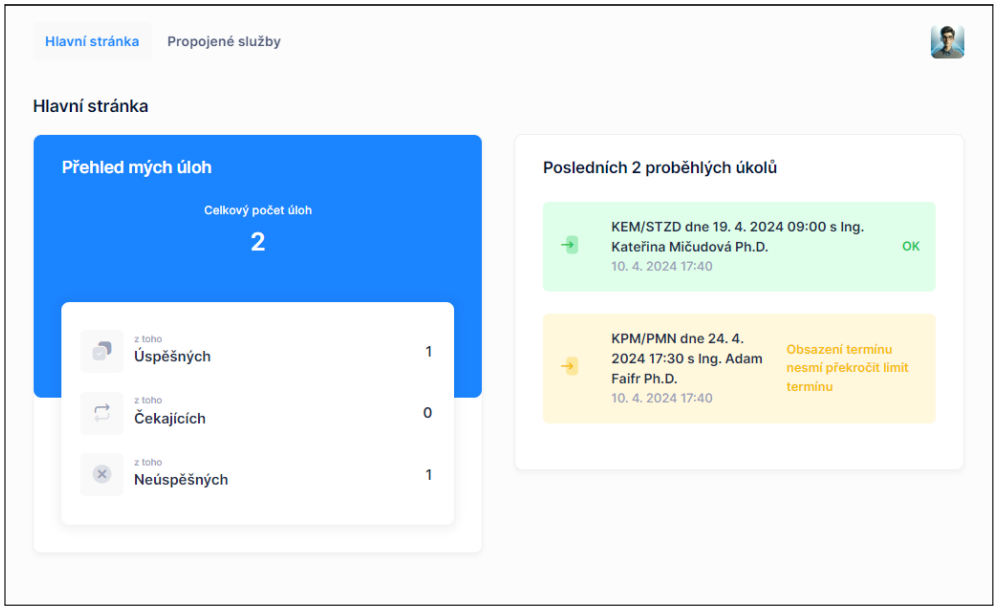


Hlavní stránka aplikace po propojení IS/STAG

Po úspěšném propojení s IS/STAG se na hlavní stránce nyní objeví přehled Vašich úkolů. Na hlavní stránce jsou dvě komponenty - „Přehled mých úkolů“ a „Posledních X proběhlých úkolů“.

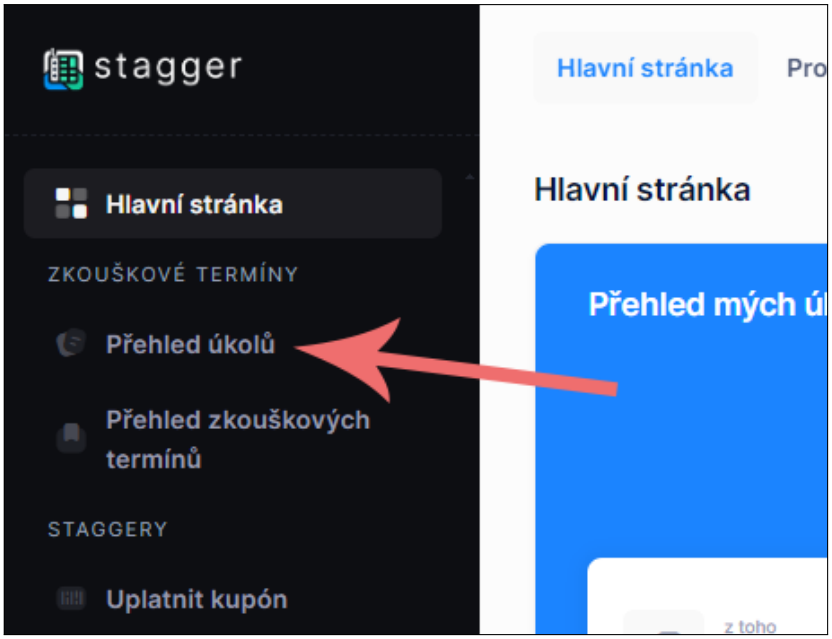


Po nějakém čase používání webové aplikace může Vaše hlavní stránka vypadat například tak, jak je zobrazeno níže.

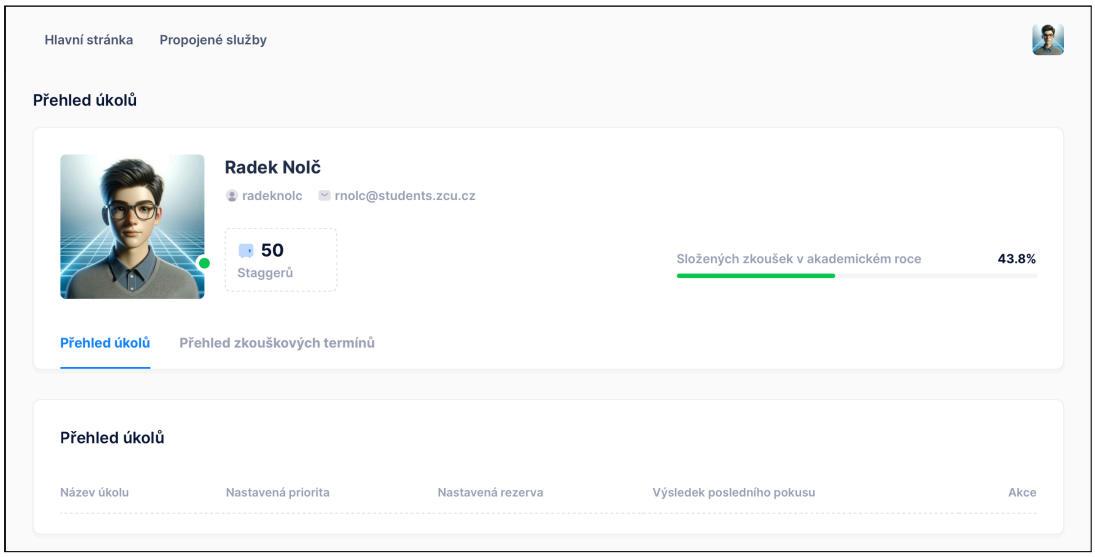


Přehled zápisových úkolů

V případě potřeby přehledu o zápisových úkolech, přejděte na stránku „Přehled úkolů“. Odkaz na tuto stránku naleznete v levém menu pod „Zkouškové termíny“.

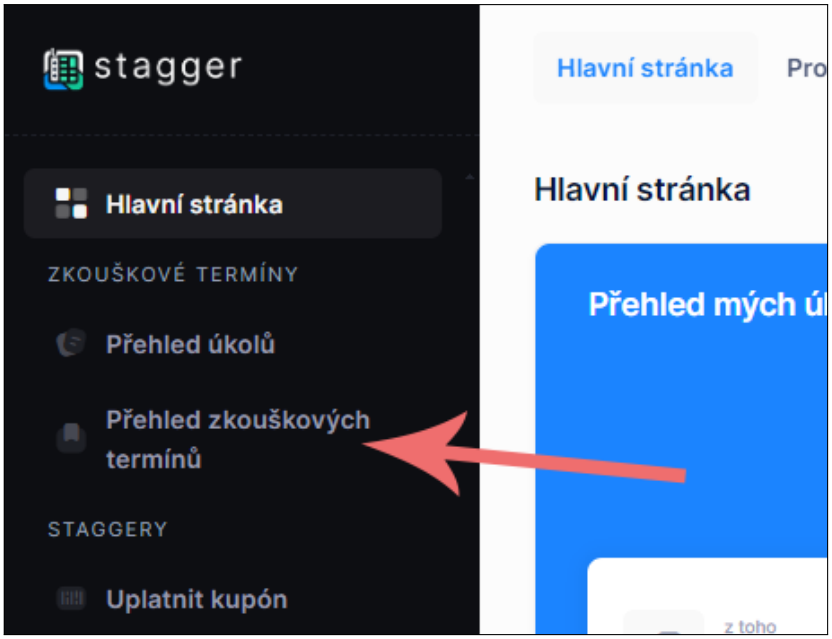


Kliknutím na tento odkaz se dostanete na stránku „Přehled úkolů“, kde později naleznete seznam všech úkolů. Nahoru se řadí aktivní úkoly, které ještě neproběhly, níže naleznete úkoly minimálně jednou proběhlé.

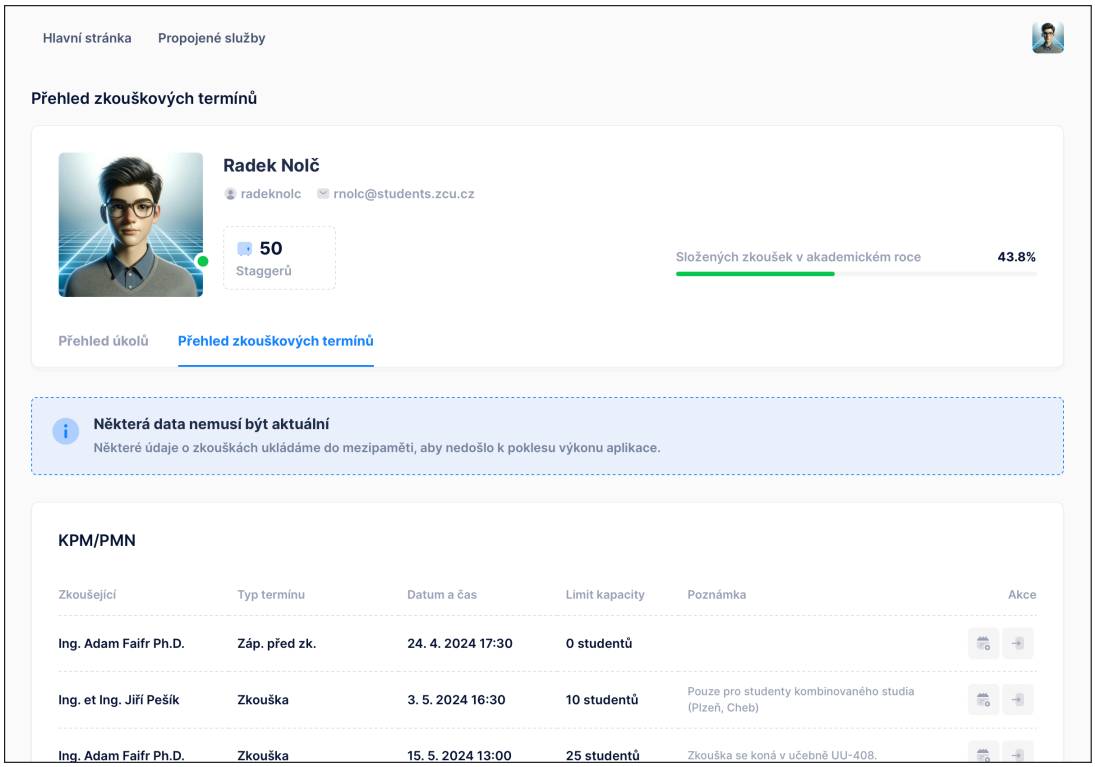


Přehled zkouškových termínů, vytvoření nového úkolu

Pokud chcete zobrazit Vaše zkouškové termíny nebo vytvořit nový zápis na zkoušku, potřebujete se přesunout na stránku „Přehled zkouškových termínů“, například přes postranní menu.

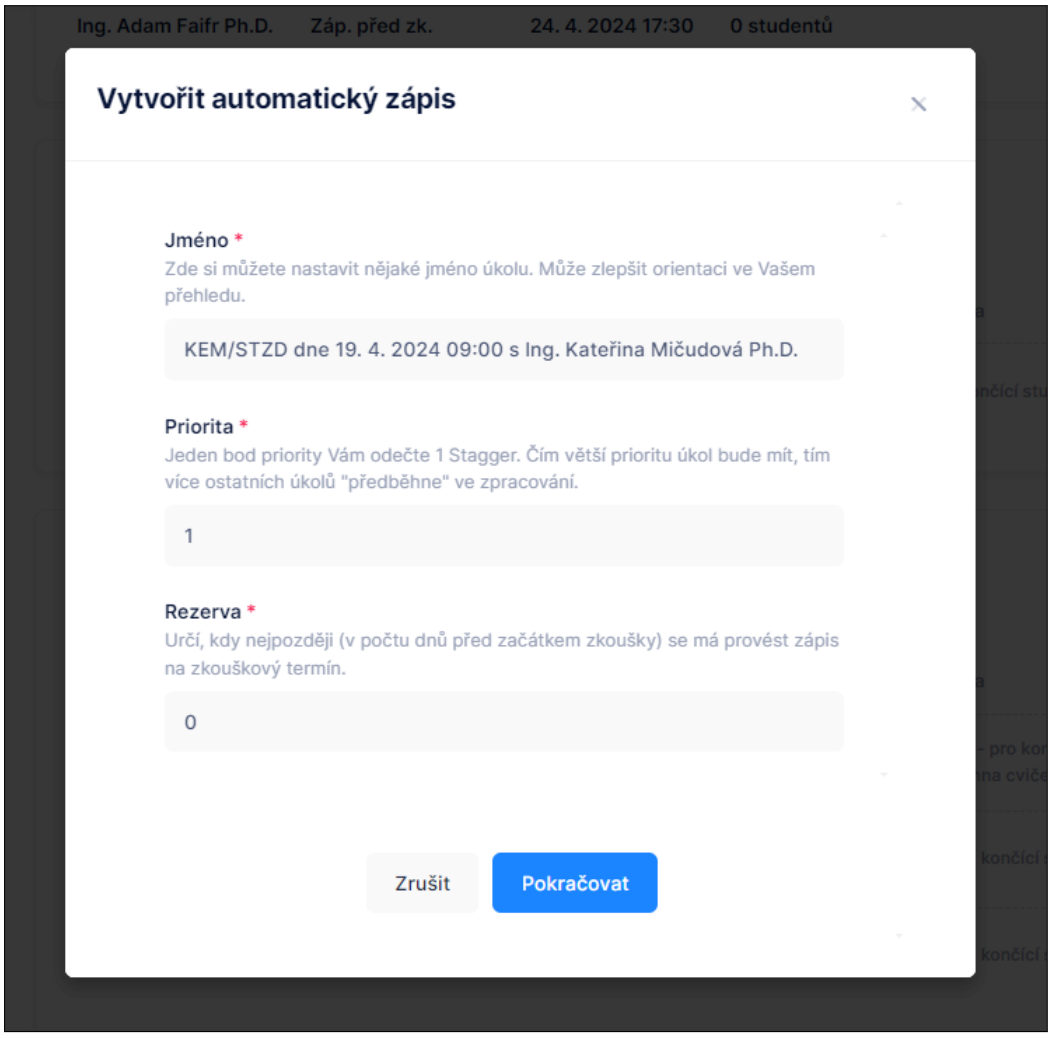


Pokud máte propojený účet se systémem IS/STAG, zobrazí se Vám seznam aktuálně vypsaných zkoušek, na které se můžete potenciálně zapsat. U každého záznamu jsou údaje o zkouškovém termínu a poté dvě akční tlačítka. První tlačítko vytváří nový automatický úkol, druhé tlačítko se Vás pokusí ručně zapsat ihned.



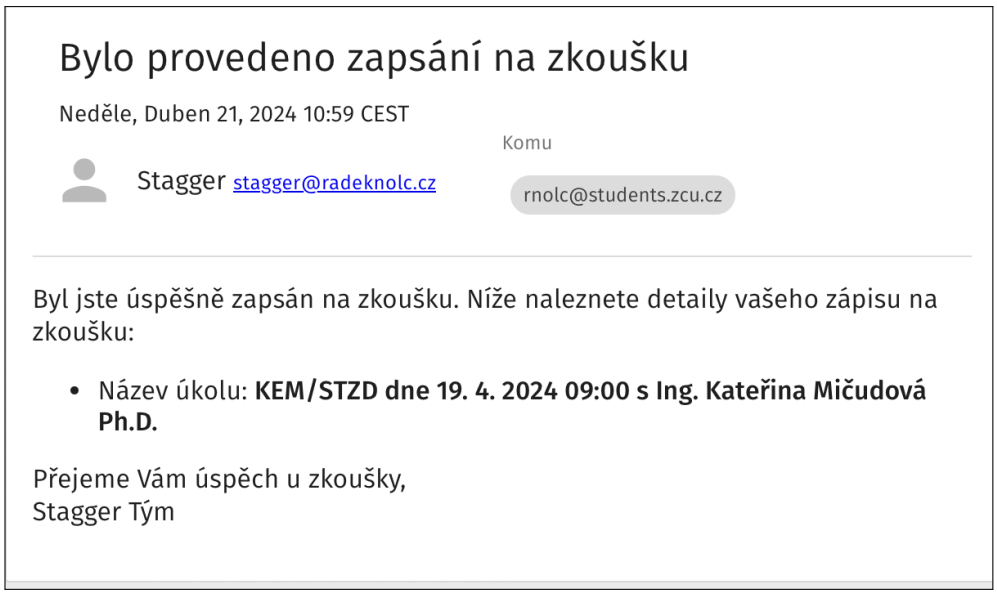
Pokud si přejete vytvořit nový automatický úkol, klikněte na první tlačítko u požadované zkoušky.

Po stisknutí tlačítka se Vám otevře okno s předvyplněnými daty. Je zde pole „Jméno“, které slouží k pojmenování vlastního úkolu, „Priorita“ definující prioritu zpracování úkolu a „Rezerva“ definující nejzazší zapsání na termín vůči datu konání zkoušky.



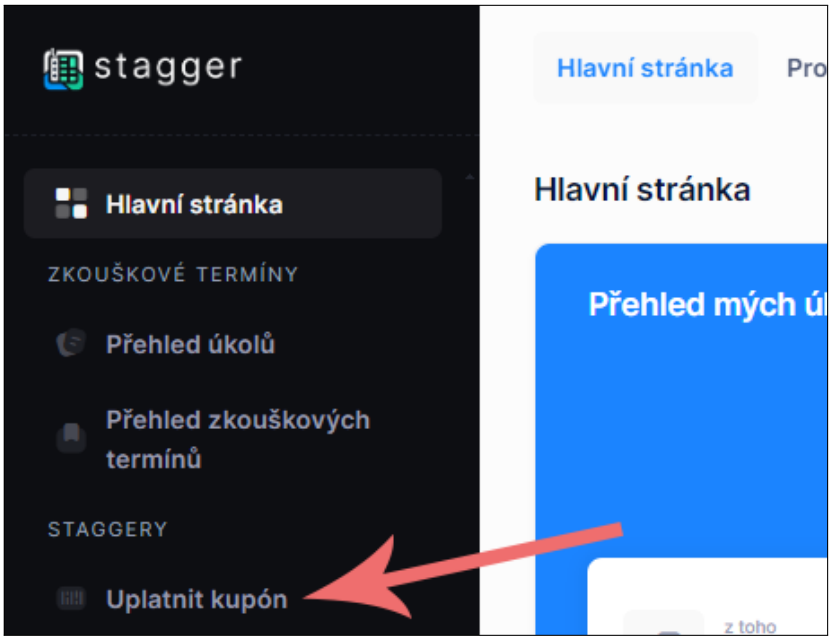
Upravte podle preference tato data a stiskněte „Pokračovat“. Tím vytvoříte automatický zápis.

Stav úkolu můžete sledovat například na stránce „Přehled úkolů“, nebo na hlavní stránce. Jakmile bude úkol úspěšně proveden, obdržíte také e-mail o úspěšném zápisu.

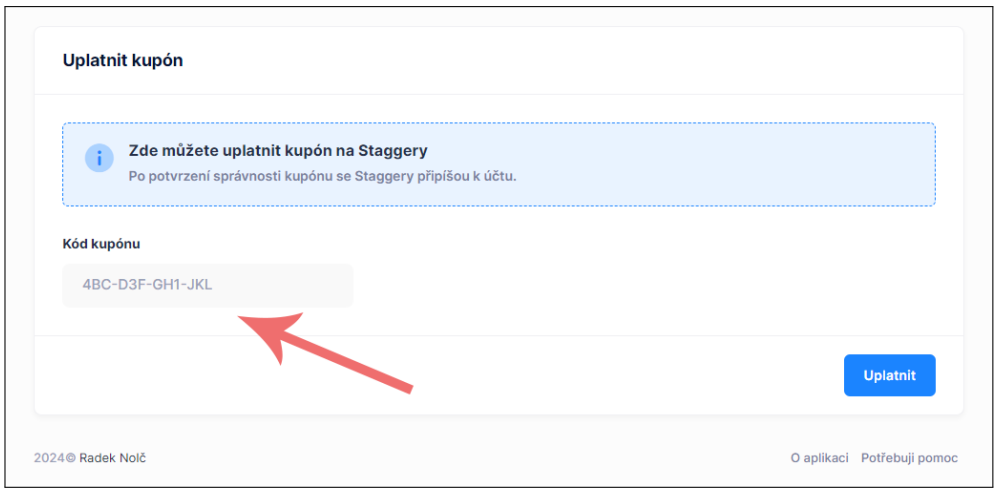


Dobití virtuální měny

V případě vyčerpání výchozího počtu virtuální měny je potřeba si pro další automatizované zápisy tuto měnu dobít. V aktuální verzi aplikace toto lze pouze pomocí vystaveného kupónu administrátorem univerzity. Pokud nějaký takový kupón máte, klikněte v menu na „Uplatnit kupón“.



Kliknutí na odkaz Vás přesměruje na stránku se sekcí „Uplatnit kupón“. Zde můžete vložit kód Vašeho kupónu na virtuální měnu a poté kliknout na tlačítko „Uplatnit“.



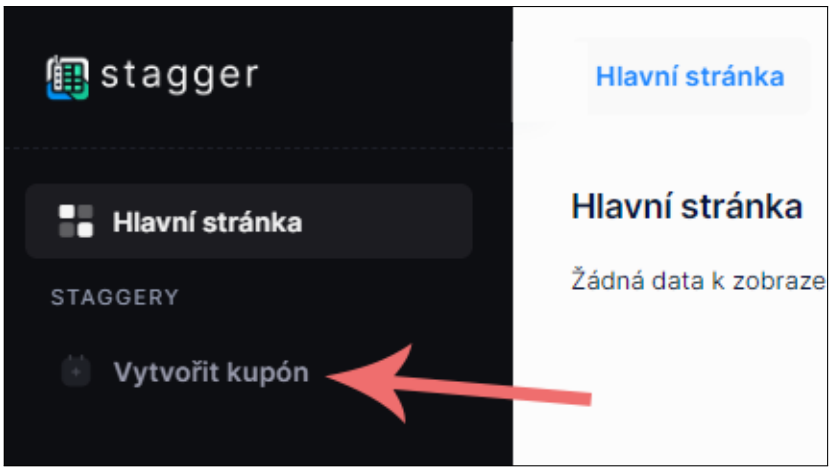
Po úspěšném uplatnění kupónu se Vám v hlavičce účtu aktualizuje stav virtuální měny.

Dokumentace pro administrátory univerzity

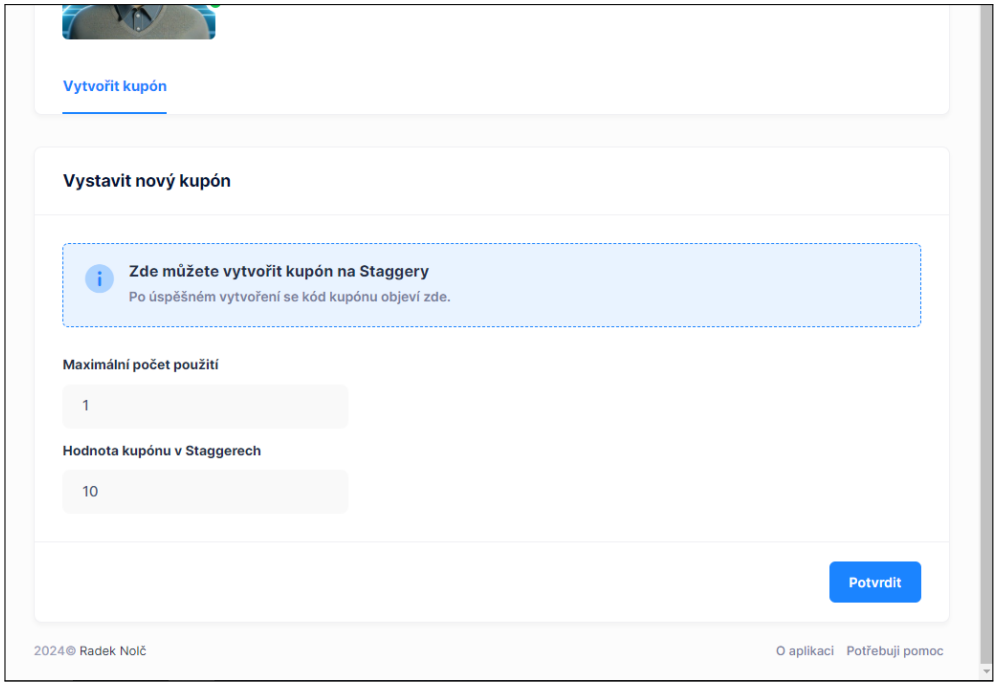
Tato sekce dokumentace představuje možnost použití webové aplikace administrátory univerzity.

Vytvoření kupónu na virtuální měnu

Pro vytvoření kupónu na virtuální měnu je potřeba se přesunout na stránku určenou pro tento účel. Klikněte v menu na „Vytvořit kupón“.



Po kliknutí na odkaz v menu budete přesměrováni na stránku se sekcí „Vytvořit kupón“. Zde vyplňte oba potřebné údaje a stiskněte „Potvdit“.



V případě úspěšně vytvořeného kupónu se kód kupónu objeví na stejném místě, jako byl umístěn formulář.

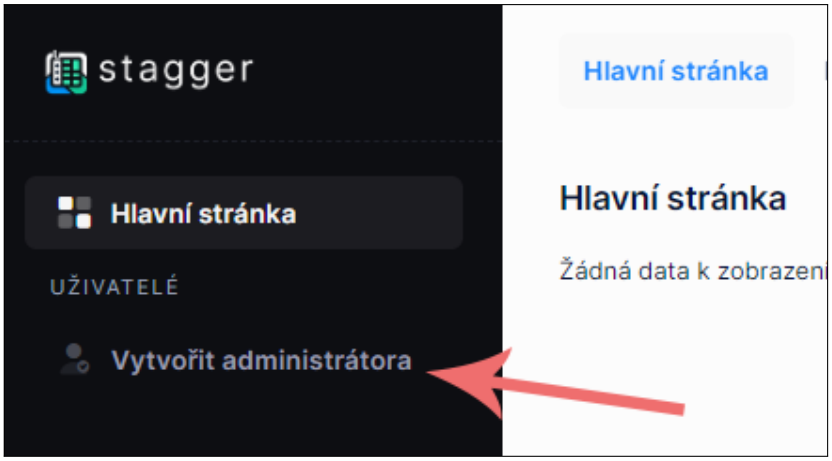
Tento kód kupónu si nyní můžete zkopírovat a rozdistribuovat podle uvážení. Pokud chcete vytvořit další kupón, stiskněte „Zpět“ a původní formulář se opět zobrazí.

Dokumentace pro provozovatele

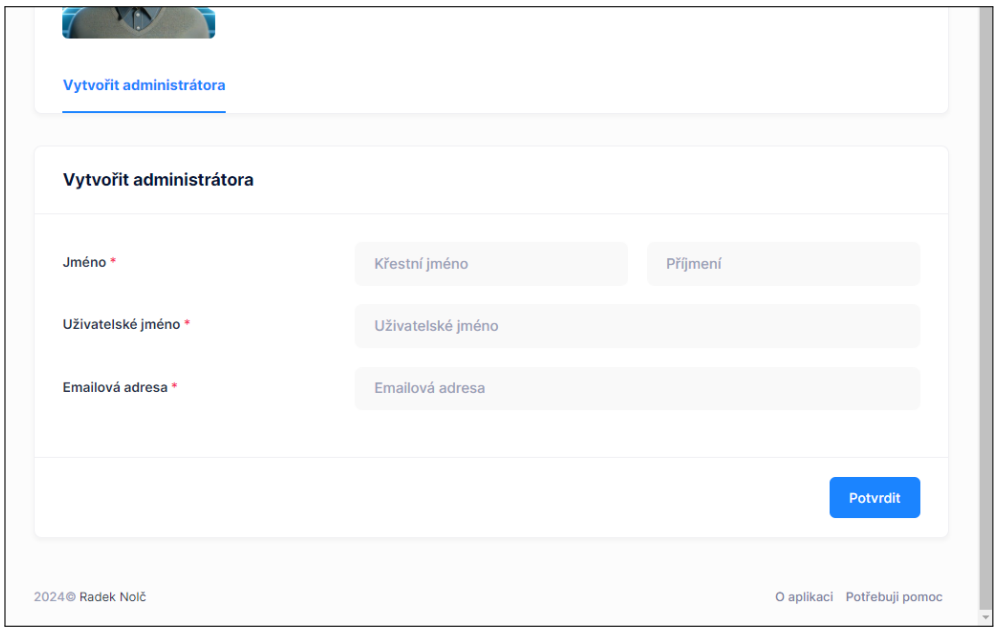
Poslední sekce dokumentace se zaměřuje na provozovatele aplikace.

Vytvoření univerzitního administrátora

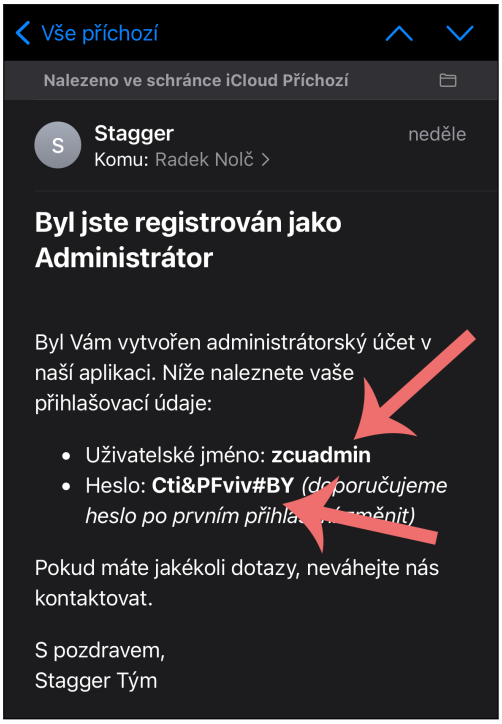
Pro vytvoření univerzitního administrátora přejděte v menu do „Vytvořit administrátora“.



Tato akce Vás přesune na stránku s formulářem. V tomto formuláři je povinné vyplnit jméno administrátora, uživatelské jméno a e-mailovou adresu. Po vyplnění stiskněte tlačítko „Potvrdit“.



V případě úspěchu bude administrátor přidán do systému a na zadaný e-mail bude zasláno zadané uživatelské jméno a systémem vygenerované heslo.


Vše příchozí
Nalezeno ve schránce iCloud Příchozí
S
Stagger
Komu: Radek Nolč >
neděle
Byl jste registrován jako Administrátor
Byl Vám vytvořen administrátorský účet v naší aplikaci. Níže naleznete vaše přihlašovací údaje:
Uživatelské jméno: zcuadmin
Heslo: Cti&PFviv#BY (doporučujeme heslo po prvním přihla
měnit)
Pokud máte jakékoli dotazy, neváhejte nás kontaktovat.
S pozdravem,
Stagger Tým

B Zdrojový kód realizované webové aplikace

Zdrojový kód realizované webové aplikace je uložen na přiloženém CD-R.

Abstrakt

Nolč, R. (2024). *Webové aplikace* [Bakalářská práce, Západočeská univerzita v Plzni].

Klíčová slova: webová aplikace; hexagonální architektura; NoSQL databáze; MongoDB; Spring Boot; React; TypeScript; Java; Docker

Tato bakalářská práce se zaměřuje na vývoj webové aplikace, která automatizuje proces přihlašování studentů na zkouškové termíny. Pro realizaci byly vybrány moderní technologie: na klientské části je použit TypeScript s frameworkem React, zatímco serverová část aplikace je postavena v Javě s frameworkem Spring Boot. Data jsou ukládána v NoSQL databázi MongoDB. Architektura aplikace je založena na principech hexagonální architektury, což přispívá k nezávislosti komponent, zlepšení modularity a testovatelnosti aplikace. Nasazení aplikace do produkčního prostředí proběhlo s využitím Dockeru. Výsledkem je plně funkční webová aplikace, která umožňuje studentům automatický zápis na zkouškové termíny. Webová aplikace je navržena tak, aby byla snadno rozšiřitelná a přizpůsobitelná budoucím požadavkům, což otevírá možnosti pro další rozvoj.

Abstract

Nolč, R. (2024). *Web applications* [Bachelor Thesis, University of West Bohemia].

Key words: web application; hexagonal architecture; NoSQL database; MongoDB; Spring Boot; React; TypeScript; Java; Docker

This bachelor thesis focuses on the development of a web application that automates the process of registering students for exams. Modern technologies were selected for implementation: TypeScript with the React framework is used on the client side, while the server side of the application is built in Java with the Spring Boot framework. Data are stored in the NoSQL database MongoDB. The architecture of the application is based on the principles of hexagonal architecture, which contributes to the independence of components, improved modularity, and testability of the application. The deployment of the application into the production environment was carried out using Docker. The result is a fully functional web application that allows students to automatically register for exams. The web application is designed to be easily expandable and adaptable to future requirements, which opens up possibilities for further development.