UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

# BAKALÁŘSKÁ PRÁCE

2025

Andrii Streblychenko

Univerzita Pardubice

Fakulta elektrotechniky a informatiky

# Studentská komunitní platforma

Bakalářská práce

2025 Andrii Streblychenko

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2024/2025

# ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

| | |
|---|---|
| Jméno a příjmení: | **Andrii Strebłychenko** |
| Osobní číslo: | **I21235** |
| Studijní program: | **B0688A140009 Informační technologie** |
| Téma práce: | **Studentská komunitní platforma** |
| Zadávající katedra: | **Katedra informačních technologií** |

## Zásady pro vypracování

Cílem práce je vyvinout funkční responzivní webovou databázovou aplikaci jakožto online prostor, který bude sloužit studentům k vzájemné interakci, sdílení informací a spolupráci. Součástí platformy budou minimálně funkce pro správu studijních skupin například pro tvorbu projektů, sdílení materiálů a obecně možnosti pro navazování kontaktů s ostatními studenty. Samotné tvorbě platformy bude předcházet podrobná analýza, ze které vyplnynou požadavky na tvorbu systému. Součástí práce je také analýza a výběr vhodných nástrojů a technologií pro realizaci výstupní aplikace a odůvodnění jejich výběru. Aplikace bude vyvinuta tak, aby splňovala zásady bezpečné aplikace.

Rozsah pracovní zprávy: **min. 30 s.,dop. rozsah 40 s.**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná**

Seznam doporučené literatury:

ACKERMANN, Philip. *Full Stack Web Development: The Comprehensive Guide*. Rheinwerk Computing, 2023. ISBN 1493224379.
FISHER, Derek. *Application Security Program Handbook: A guide for software engineers and team leaders*. Manning, 2022. ISBN 978-1633439818.

Vedoucí bakalářské práce: **Ing. Monika Borkovcová, Ph.D.**
Katedra informačních technologií

Datum zadání bakalářské práce: **15. prosince 2024**
Termín odevzdání bakalářské práce: **16. května 2025**

L.S.

**prof. Ing. Petr Doležel, Ph.D.** v.r.
děkan

**Ing. Jan Panuš, Ph.D.** v.r.
vedoucí katedry

V Pardubicích dne 28. února 2025

Prohlašuji:

Práci s názvem «Studentská komunitní platforma» jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 7/2019 Pravidla pro odevzdávání, zveřejňování a formální úpravu závěrečných prací, ve znění pozdějších dodatků, bude práce zveřejněna prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 15. 05. 2025

Jméno a příjmení autora v.r.

# PODĚKOVÁNÍ

Rád bych poděkoval své matce za morální podporu a možnost studovat daleko od domova. Bez její podpory a úsilí, které do mě vložila, bych nikdy nebyl tam, kde jsem dnes. Dále bych chtěl poděkovat svým přátelům, kteří se během studia na univerzitě stali mými nejbližšími. Díky vám, vaší podpoře a společně stráveným okamžikům se mi podařilo úspěšně dokončit studium a zažít jedny z nejlepších chvil svého života. Na závěr chci vyjádřit zvláštní poděkování Ing. Monice Borkovcové, Ph.D., bez níž bych tuto práci nejen nedokončil včas, ale pravděpodobně bych ani nevěděl, jak ji začít a jak postupovat. Bez její pomoci by tato práce nikdy nevznikla.

# ANOTACE

Cílem této bakalářské práce je navrhnout a vyvinout responzivní webovou databázovou aplikaci, která bude sloužit jako online platforma pro usnadnění interakce, spolupráce a sdílení informací mezi studenty univerzity. Aplikace bude primárně zahrnovat funkce pro správu studijních skupin, tvorbu projektů, sdílení materiálů a obecně možnosti pro navazování kontaktů mezi studenty. Vývoj bude předcházet podrobná analýza, na jejímž základě budou definovány požadavky na systém. Součástí práce je také analýza a odůvodněný výběr vhodných technologií a nástrojů potřebných pro realizaci výstupní aplikace, přičemž důraz je kladen na bezpečnostní standardy a osvědčené postupy.

# KLÍČOVÁ SLOVA

studentská platforma, responzivní webová aplikace, databázový systém, spolupráce, studijní skupiny, webové technologie, bezpečnost aplikace

# TITLE

The Student Community Platform

# ANNOTATION

The aim of this bachelor thesis is to design and develop a responsive web-based database application intended as an online platform facilitating interaction, collaboration, and information sharing among university students. The application will primarily include functionalities for managing study groups, project creation, material sharing, and general networking opportunities between students. The development process begins with an in-depth analysis identifying the system requirements. Additionally, the thesis encompasses a thorough analysis and justified selection of appropriate technologies and tools necessary for implementing the final application, emphasizing security standards and best practices.

# KEYWORDS

student platform, responsive web application, database system, collaboration, study groups, web technologies, application security

# OBSAH

# SEZNAM OBRAZKU

# SEZNAM TABULEK

# SEZNAM ZKRATEK

HTTP – Hypertext Transfer Protocol

HTTPS – Hypertext Transfer Protocol Secure

AJAX – Asynchronous JavaScript and XML

RESTful API – Representational State Transfer Application Programming Interface

MVC – Model-View-Controller

MVP – Model-View-Presenter

MVVM – Model-View-ViewModel

MVVM-C – Model-View-ViewModel-Coordinator

MVI – Model-View-Intent

SQL – Structured Query Language

NoSQL – Not Only SQL

HTML – HyperText Markup Language

CSS – Cascading Style Sheets

JS – JavaScript (ECMAScript)

SPA – Single-Page Application

DOM – Document Object Model

CI/CD – Continuous Integration/Continuous Deployment

DevOps – Development and Operations

JWT – JSON Web Token

JSON – JavaScript Object Notation

JPA – Java Persistence API

ORM – Object-Relational Mapping

WS – WebSocket (pro dvoucestnou komunikaci v reálném čase)

STOMP –Streaming Text Oriented Messaging Protocol

DTO – Data Transfer Object

STAG – Studijní agenda (systém pro správu studijních dat)

# ÚVOD

V dnešní době hraje efektivní online komunikace a spolupráce klíčovou roli v životě studentů na vysokých školách. Existující sociální sítě a běžné komunikační nástroje však často nedokážou uspokojit specifické potřeby akademického prostředí. Studenti proto narážejí na řadu problémů, například při navazování nových kontaktů, sdílení studijních materiálů nebo integraci do života na univerzitě. Na základě těchto zkušeností vznikla myšlenka vytvořit specializovanou komunitní platformu určenou přímo pro studenty, která by sloužila jako jednotné místo pro komunikaci, výměnu informací a celkovou podporu při zapojení do akademické komunity.

Cílem této bakalářské práce je navrhnout, vytvořit a následně vyhodnotit webovou aplikaci, která bude reflektovat konkrétní potřeby studentů. Tato platforma by měla studentům nabídnout jednoduchý způsob, jak vytvářet a spravovat studijní skupiny, komunikovat mezi sebou pomocí chatu či diskusního fóra, a rovněž usnadnit ověření uživatelů díky propojení se studijním systémem, například STAG. Při vývoji aplikace byl kladen velký důraz na bezpečnost, snadnou použitelnost a také optimalizaci pro různá zařízení, včetně mobilních telefonů.

K výběru tohoto tématu mě vedla především osobní zkušenost s nedostatečnou podporou, kterou aktuálně dostupné platformy nabízejí studentům, zejména těm, kteří teprve začínají své studium nebo přicházejí ze zahraničí. Tito studenti často čelí výrazným informačním bariérám, které jim znesnadňují orientaci na univerzitě. Z toho důvodu bylo mým cílem navrhnout řešení, které by mohlo jim pomoc tyto překážky překonat a zároveň zjednodušilo jejich začlenění do studentského života.

Bakalářská práce je rozdělena do dvou hlavních částí. První část se zabývá popisem a porovnáním technologií a architektur, které se v současnosti využívají při vývoji webových aplikací. Druhá, praktická část obsahuje analýzu potřeb studentů, návrh samotné aplikace a její následnou realizaci. V závěru práce je pak provedeno celkové zhodnocení a naznačeny možnosti budoucího rozšíření platformy.

# 1. Webová aplikace

Webová databázová aplikace představuje softwarové řešení, které umožňuje uživatelům přistupovat k centralizované databázi prostřednictvím webového rozhraní. Tyto aplikace integrují správu dat a webovou prezentaci, usnadňuje aktualizaci, údržbu i zabezpečení informací. V současnosti jsou webové databázové aplikace klíčovým prvkem při správě informací ve vzdělávacích institucích i firmách, protože umožňují dynamické a interaktivní služby založené na standardních protokolech, jako jsou HTTP/HTTPS, a využívají technologie, například AJAX a RESTful API, pro efektivní komunikaci mezi klientem a serverem [1].

Webové aplikace fungují na principu modelu „klient-server“ a využívají standardní protokoly HTTP/HTTPS. Vývoj těchto aplikací je charakterizován přechodem od statických webových stránek k dynamickým systémům, které jsou postaveny na modulární architektuře se snadnou integraci nových funkcí a efektivní škálovatelnost. Technologie RESTful API standardizují výměnu dat mezi klientem a serverem a usnadňuje vývoj a údržbu aplikací. Historický vývoj webových technologií dokládá postupný přechod od Web 1.0 k interaktivním řešením Web 2.0 a dále k experimentálním přístupům Web 3.0, přičemž nejrozšířenější a nejspolehlivější jsou právě aplikace založené na modelu Web 2.0 [3]. Frameworky pro frontend, jako jsou Vue, React nebo Angular, v kombinaci s backendovými technologiemi, například Spring Boot nebo Laravel, poskytuje vytvářet flexibilní, rychlé a spolehlivé aplikace, které splňují požadavky současného uživatele [1].

## 1.1 Architektura řešení při vývoji webových aplikací

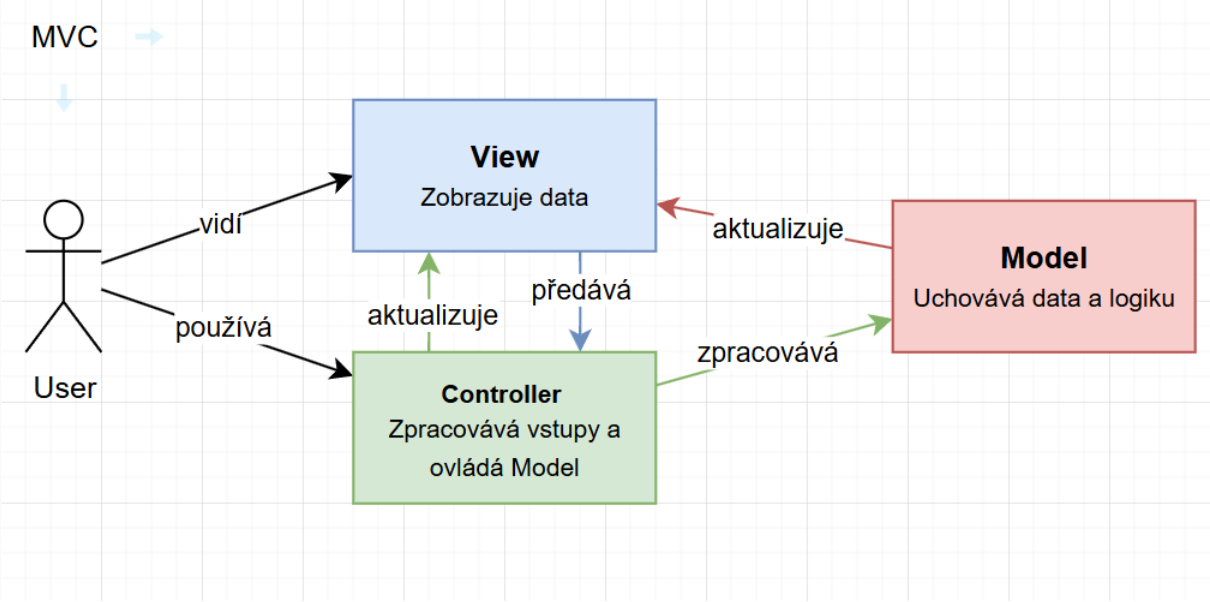
Architekturní řešení ve vývoji webových aplikací hrají klíčovou roli při formování celkové struktury a funkčnosti systému. Správná volba architektury určuje nejen výkon a škálovatelnost komponent v budoucnu. V dnešním světě, kde se software neustále vyvíjí, se architektonická rozhodnutí stávají základem pro rychlou reakci na nové požadavky a technologické výzvy. Je důležité si uvědomit, že architektura webové aplikace není jen souborem komponent, ale souhrnem podložených návrhových rozhodnutí, která ovlivňují celý životní cyklus produktu. Tento přístup snižuje pravděpodobnost vzniku chyb, minimalizovat náklady na údržbu a zajistit vysokou úroveň bezpečnosti dat, a to je obzvláště aktuální pro systémy využívané ve vzdělávacím i korporátním prostředí [1].

### 1.1.1 Architektonický vzor MVC

Model-View-Controller (MVC) je základní architektonický vzor, který se stal nedílnou součástí vývoje webových aplikací. Tento vzor rozděluje aplikaci do tří hlavních komponent – modelu,

view a controlleru – přičemž každá má jasně vymezenou roli. Model představuje datovou a business logiku; stará se o správu, validaci a ukládání dat, a zároveň definuje pravidla, která s těmito daty souvisejí. View, tedy prezentační vrstva, zodpovídá za vizuální zobrazení informací, které jsou získány z modelu, a to ve formě uživatelského rozhraní. Controller slouží jako prostředník mezi modelem a view; přijímá vstupy od uživatele, interpretuje je a na jejich základě volá odpovídající metody modelu, přičemž následně vyvolává aktualizace ve view.

Toto oddělení odpovědností přináší řadu výhod. Díky jasnému rozdělení funkcí, jednotlivé části aplikace lze vyvíjet a testovat nezávisle a výrazně usnadnovat údržbu a škálovatelnost systému. Například změny v uživatelském rozhraní (view) nevyžadují zásahy do business logiky (model), a naopak. Tento přístup také umožňuje týmům pracovat efektivněji, a soustředit pozornost na konkrétní komponenty bez nutnosti hlubokého porozumění celkové implementaci.

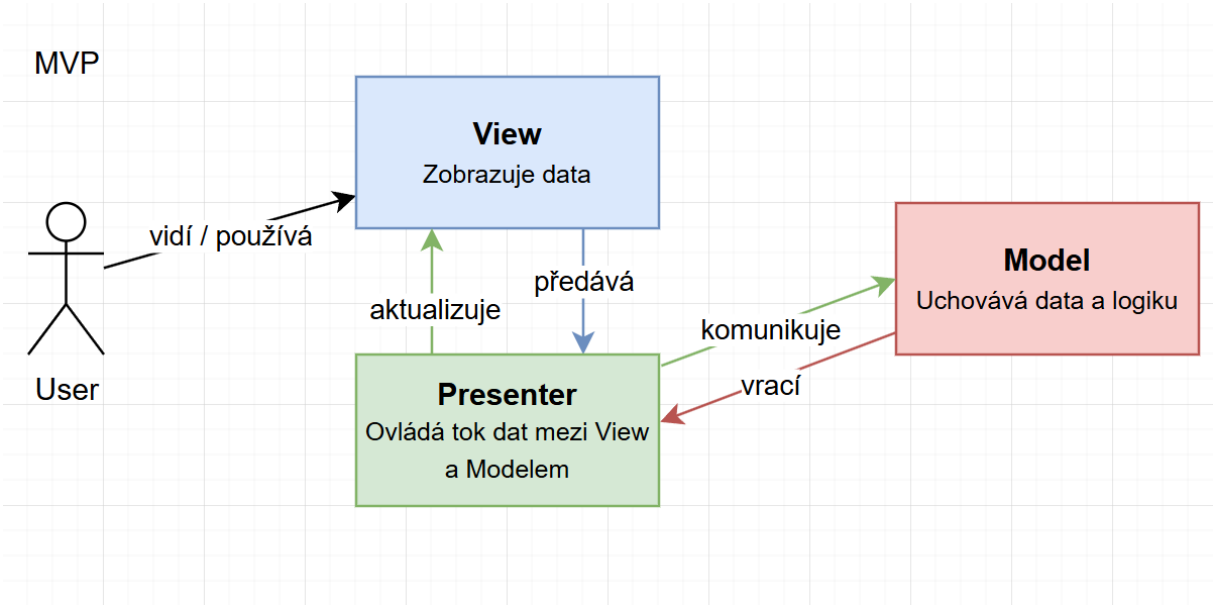



*Obrázek 1: Common model-view-controller pattern (vl zdroj)*

Výzkumy, jako například studie klasifikace MVC softwarových aplikací pomocí samoorganizujících map (Self-Organizing Maps, SOM) [4], ukazují, že správná implementace MVC vzoru výrazně ovlivňuje kvalitu softwaru. Analýza MVC aplikací pomocí metrik jako počet řádků kódu, duplicity, složitost a designová kvalita poskytuje cenné informace o tom, jak různé implementační strategie ovlivňují udržovatelnost a robustnost systému. Tato analýza umožňuje identifikovat potenciální problémy (tzv. "code smells") a optimalizovat architektonická rozhodnutí, zásadní milníky pro dlouhodobou úspěšnost a efektivitu softwarových řešení.

Z praktického hlediska je MVC vzor obzvláště cenný v prostředích, kde je potřeba zajistit vysokou míru interaktivity a dynamickou aktualizaci obsahu. Tyto požadavky jsou dnes typické pro webové aplikace. [4].

### 1.1.2 Model-View-Presenter **MVP**



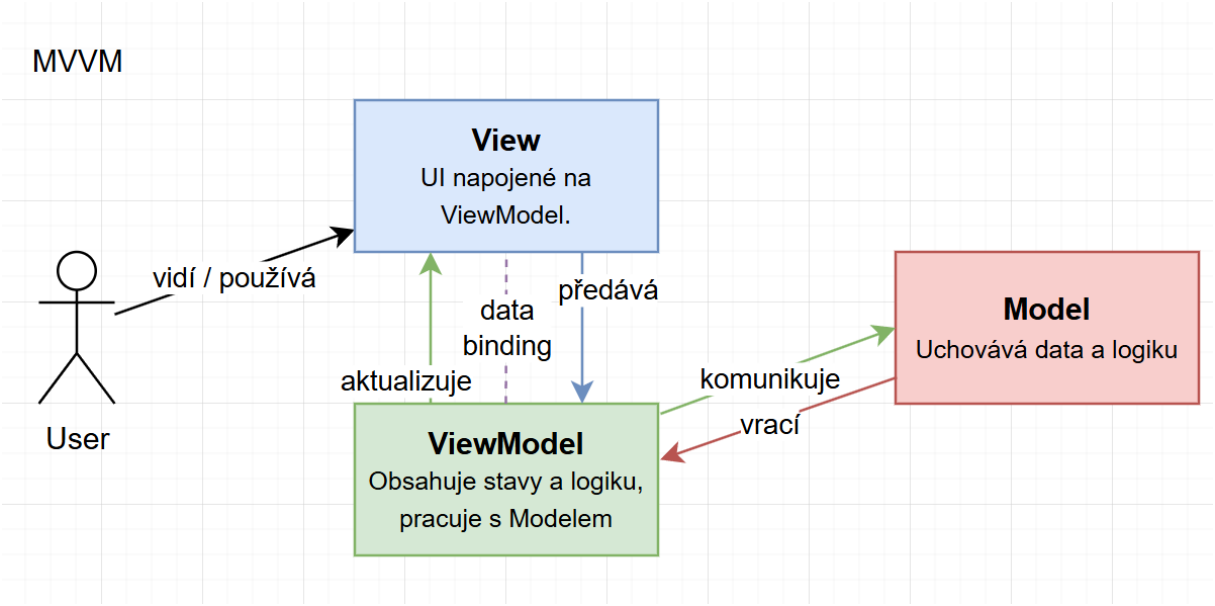

*Obrázek 2: Common model-view-presenter pattern (vl zdroj)*

Model-View-Presenter (MVP) je evoluční variantou klasického MVC, která vznikla s cílem snížit úzké propojení mezi prezentační vrstvou a business logikou. Ve vzoru MVP je View definováno jako pasivní komponenta, která nezpracovává žádnou prezentační logiku, ale pouze zobrazuje data a předává uživatelské akce do Presenter. Presenter funguje jako prostředník – přijímá vstupy z View, komunikuje s Modelem a následně aktualizuje View. Tento přístup umožňuje jasné oddělení odpovědností, lepší testovatelnost a snadnější údržbu aplikace.[5]

Presenter v tomto vzoru hraje klíčovou roli při zpracování uživatelských interakcí a aktualizaci UI. Na rozdíl od MVC, kde Controller často přímo manipuluje s View, v MVP Presenter nemá přímý přístup k View. Místo toho komunikuje prostřednictvím rozhraní. Tento přístup umožňuje snadnější testování a modularitu komponent.[5][8][33]

Empirické studie zaměřené na výkon nativních Android aplikací ukazují, že MVP může mít výhodu v oblasti správy zdrojů – například některé experimenty uvádějí, že architektura MVP vykazuje nižší spotřebu paměti ve srovnání s alternativními vzory. Mezi její hlavní přednosti patří jednoduchá implementace a dobrá testovatelnost [8]. V praxi ale může docházet ke zvýšení složitosti kódu kvůli soustředění business logiky do komponenty Presenter, a tím i ke ztíženému dalšímu rozvoji a údržbě systému [5].

### 1.1.3 **Model-View-ViewModel MVVM**




*Obrázek 3: Common model-view-viewmodel pattern (vl zdroj)*

Model-View-ViewModel (MVVM) je architektonický vzor, který se stal oblíbeným zejména díky podpoře obousměrné datové vazby a reaktivního programování. V MVVM se aplikace dělí na tři hlavní komponenty: Model, který spravuje data a business logiku; View, jež představuje uživatelské rozhraní; a ViewModel, který funguje jako prostředník mezi Modelem a View a transformuje data do formy vhodné pro zobrazení. Díky datové vazbě se změny v Modelu automaticky promítají do View a změny v uživatelském rozhraní jsou reflektovány ve ViewModelu bez nutnosti explicitní synchronizace.

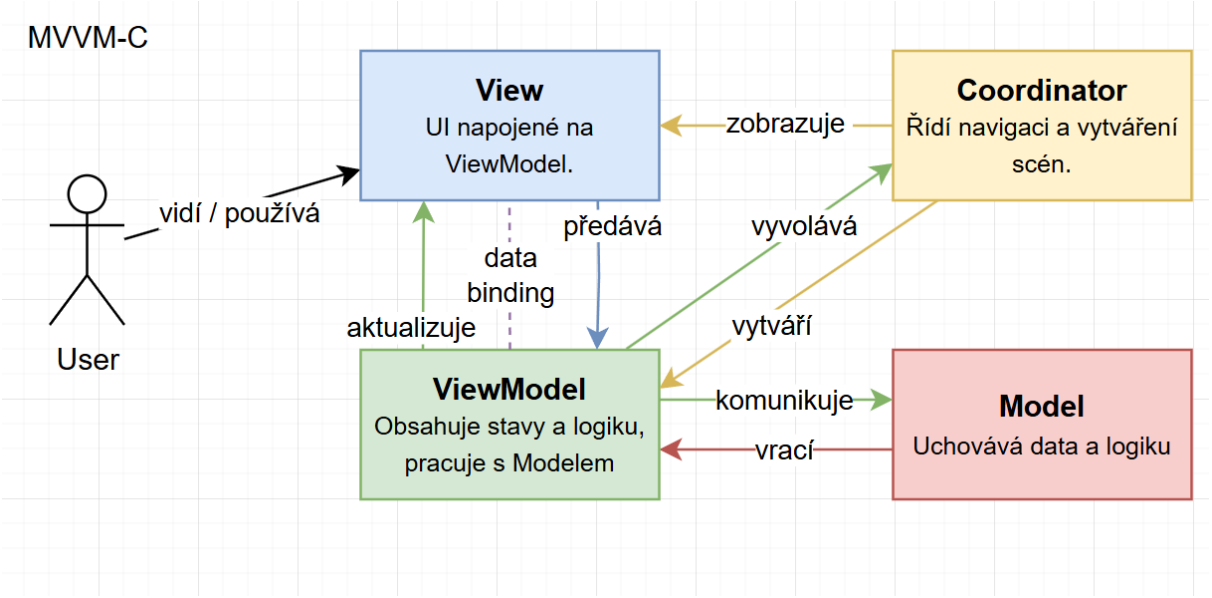
Rozsáhlé literární přehledy a studie, jako je například práce "Evaluating Android Architectural Patterns: A Deep Dive into MVP, MVVM, and MVI" [7], potvrzují, že MVVM zvyšuje testovatelnost a udržovatelnost aplikací. Použití reaktivní datové vazby přispívá k rychlejší odezvě a efektivnějšímu využití CPU, přičemž zároveň však může vést k mírně vyšší spotřebě paměti kvůli implementaci dodatečné knihovny pro datovou vazbu [8].

Ve srovnání s MVP, kde Presenter přímo komunikuje s View prostřednictvím rozhraní, MVVM nabízí volnější propojení mezi komponentami. ViewModel nemá přímou referenci na View, čím umožňuje lepší oddělení odpovědností a usnadňuje testování. MVVM podporuje opětovné použití ViewModelu v různých View, což zvyšuje flexibilitu a škálovatelnost aplikací [5][33].

MVVM se hojně využívá především v mobilních a desktopových aplikacích, kde je podpora datové vazby běžná, ale v oblasti webového vývoje může být MVVM součástí širší architektury

MVC, například jako součást front-endové architektury. Vue.js, Angular.js implementuje MVVM přístup prostřednictvím reaktivních komponent a dvoucestné datové vazby [25].

### 1.1.4 **Model-View-ViewModel-Coordinator MVVM-C**



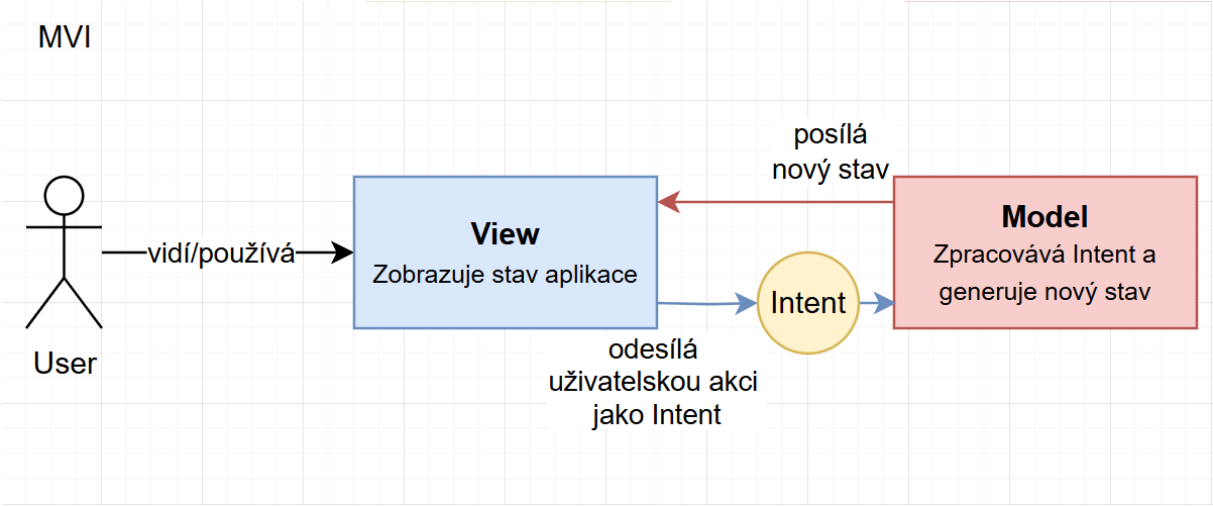

*Obrázek 4: Common model-view-viewmodel-coordinator pattern (vl zdroj)*

MVVM-C představuje rozšíření standardního MVVM vzoru o dodatečnou komponentu, a to koordinátora (Coordinator), která se zaměřuje na správu navigačních toků a řízení přechodů mezi obrazovkami. V této architektuře zůstávají základní principy MVVM zachovány, tzn. že Model spravuje data, View zobrazuje uživatelské rozhraní a ViewModel obsahuje prezentační logiku – avšak koordinátor zajišťuje centralizovanou správu navigace. Díky tomu může ViewModel zůstat čistý a zaměřený pouze na správu dat a logiku, aniž by se musel starat o implementaci přechodů mezi různými částmi uživatelského rozhraní. Koordinátor také odpovídá za vytváření závislostí, jako jsou instance View a ViewModel, a jejich správné propojení, čímž se snižuje duplicitní kód a minimalizuje riziko chyb při řízení toků mezi různými moduly. Tato architektura se ukazuje jako zvláště vhodná pro rozsáhlé projekty, kde je potřeba efektivně řídit interakce mezi komponentami a zároveň podporovat agilní vývoj a škálovatelnost [7] [34].

MVVM-C je často využíván při vývoji mobilních aplikací, zejména pro platformu iOS, kde koordinátor pomáhá udržet ViewController jednoduchý a zaměřený pouze na zobrazení uživatelského rozhraní. Tuto architekturu lze aplikovat i ve webovém vývoji. Například ve frameworku Vue.js může být Vue Router použit jako koordinátor, který spravuje navigaci mezi komponentami. V tomto kontextu router funguje jako samostatná entita, která vytváří a

propojuje komponenty View a ViewModel, čímž naplňuje principy MVVM-C i ve frontendovém vývoji [15].

### 1.1.5 Model-View-Intent MVI




*Obrázek 5: Common model-view-intent pattern (vl zdroj)*

MVI se skládá ze tří hlavních komponent: Model, View a Intent. View – pasivně zobrazuje aktuální stav a odesílá uživatelské akce jako Intenty. Intent – zachycuje uživatelské akce (např. kliknutí) a předává je Modelu k vyhodnocení. Model – každá změna vytváří nový stav, eliminuje nejednoznačnosti a zajišťuje konzistenci. Po zpracování v Modelu se stav automaticky promítne do View. Tímto způsobem data obíhají „v kruhu“, přičemž je zajištěn jednosměrný tok dat a neměnnost stavu aplikace. Tento vzor je obzvláště vhodný pro aplikace s komplexními interakcemi a asynchronními událostmi, kde je důležitá předvídatelnost a snadné ladění. Nicméně pro jednodušší aplikace může být implementace MVI zbytečně složitá a náročná na údržbu [5] [7].

Ačkoliv MVI vzniklo v kontextu mobilního vývoje, jeho principy jsou aplikovatelné i ve webovém prostředí, zejména při použití frameworku Vue.js. Vuex, oficiální knihovna pro správu stavu ve Vue.js, implementuje jednosměrný tok dat a centralizované řízení stavu. Tento přístup odpovídá principům MVI. Použití Vuex tak přináší výhody MVI do webového vývoje, zejména v projektech, kde je důležitá konzistence stavu a předvídatelnost chování aplikace [15].

### 1.1.6 **Monolitická architektura**

Monolitická architektura představuje tradiční přístup k vývoji webových aplikací, kdy je celá business logika, správa dat a prezentace uživatelského rozhraní implementována v jediné kódové bázi. V tomto přístupu nejsou aplikace rozděleny do nezávislých služeb – všechny

komponenty běží v rámci jednoho procesu a využívají společnou databázi. Mezi hlavní výhody monolitu patří jeho relativní jednoduchost při vývoji, testování, ladění a nasazení, protože není třeba implementovat složité mechanismy mezi-procesové komunikace. Blinowski a spol. [9] uvádí, že v jedno-serverových nasazeních může monolit vykazovat vyšší výkon díky minimálním režijním nákladům spojeným s vnitřním voláním metod. Nicméně s rostoucí velikostí a složitostí aplikace vznikají problémy se škálovatelností – změna v jednom modulu může ovlivnit celý systém a horizontální škálování často vyžaduje kompletní replikaci aplikace. Tento přístup vede k neefektivnímu využití zdrojů [9]. Se zvyšující se složitostí správy závislostí a udržení modulární struktury narůstá i obtížnost adaptace systému na dynamicky se měnící požadavky [1] [3].

### 1.1.7 **Mikroslužbová architektura**

Oproti monolitické architektuře je mikroslužbová koncepce založena na rozdělení aplikace do souboru autonomních, nezávisle nasaditelných služeb, z nichž každá plní přesně vymezenou funkční roli. Každá mikroslužba může mít vlastní obchodní logiku, vlastní databázi a je vyvíjena a nasazována nezávisle na ostatních službách. Tento přístup umožňuje flexibilní škálování – je možné zvýšit počet instancí pouze u těch služeb, které jsou zatíženy nejvyšším počtem požadavků, a právě tento model je vhodný zejména pro rozsáhlé systémy s vysokou mírou paralelních dotazů [9]. Mikroslužby také zvyšují odolnost systému, protože selhání jedné služby neznamená ztrátu funkčnosti celé aplikace. Distribuovaná povaha mikroslužeb však přináší další složitosti, zejména v oblasti komunikace mezi službami. Ta může zvýšit režii a zkomplikovat monitorování, především při vysokém zatížení [9]. Použití standardních protokolů, jako jsou HTTP/HTTPS a RESTful API, napomáhá standardizaci výměny dat mezi službami, zároveň však vyžaduje dodatečnou infrastrukturu pro orchestraci a správu služeb – například prostřednictvím API bran nebo systémů pro objevování služeb [2], [9].

### 1.1.8 **Zhodnocení architektur**

Výběr mezi monolitickou a mikroslužbovou architekturou závisí na mnoha faktorech, včetně velikosti a složitosti aplikace, požadavků na škálovatelnost a odolnost systému, a také na schopnosti týmu efektivně spravovat a vyvíjet systém. Jak uvádí [9], monolitická architektura může v jednoserverových prostředích vykazovat lepší výkon díky absenci nákladů na meziprocesovou komunikaci. Monolitické aplikace jsou navíc levnější na vývoj a údržbu. S rostoucím zatížením a narůstajícími požadavky však mohou být vhodnějším řešením mikroslužby. U rozsáhlých systémů s dynamicky se měnícím zatížením nabízí mikroslužbový přístup významné výhody v oblasti škálovatelnosti a nezávislého řízení jednotlivých částí

systému. Použití RESTful API a moderních technologií pro automatizaci a orchestraci napomáhá minimalizaci režijních nákladů spojených s distribuovanou povahou mikroslužeb. Hlavními nevýhodami mikroslužbové architektury jsou však vyšší nároky na systémové prostředky a potřeba profesionálních dovedností spolu se sehraným týmem pro vytvoření kvalitní aplikace [3], [9].

Na základě přehledu výhod a nevýhod jednotlivých architektonických přístupů vyplývá, že volba optimální architektury pro konkrétní projekt musí vycházet z jeho specifických požadavků, jako je velikost systému, očekávané zatížení a potřeba rychlých iterací. Pro projekt, který se zaměřuje na relativně malý až středně velký rozsah funkcí a nepočítá s masivním nárůstem uživatelské základny, se jako nejvhodnější volba jeví monolitická architektura. Díky své jednoduchosti, nižším režijním nákladům na interní volání a snadnější implementaci umožňuje rychlejší vývoj, testování a nasazení aplikace. To je klíčové pro dosažení stabilního a efektivního řešení. I když mikroslužby nabízejí výhody v oblasti škálovatelnosti a nezávislého nasazení jednotlivých komponent [3], [9].

# 2 Technologie používané při vývoji webových aplikací

Vývoj webových aplikací je komplexní proces, který vyžaduje integraci řady technologií, přičemž klíčovým prvkem je správná volba technologického stacku. Tento stack zahrnuje frontendové technologie, které zajišťují interaktivní a responzivní uživatelské rozhraní, a backendové technologie, kde je implementována business logika, správa dat a zabezpečení. Kromě toho hraje roli i integrace s databázemi a externími službami prostřednictvím RESTful API nebo jiných komunikačních protokolů. Taková řešení umožňují dynamickou výměnu informací mezi klientem a serverem a představují základ pro škálovatelné a udržitelné systémy [3] [9] [10].

## 2.1 Backendové technologie

Backendové technologie tvoří „motor“ webových aplikací, mezi nejčastěji používané patří Node.js, PHP (Laravel), Python (Django) a robustní enterprise platformy jako Spring Boot. Node.js, využívající JavaScript, je založený na asynchronní event-driven architektuře, což umožňuje efektivní zpracování mnoha souběžných požadavků [10]. PHP s frameworkem Laravel je oblíbený pro svou jednoduchost, rychlý vývoj a širokou komunitní podporu, avšak při vyšších nárocích může vykazovat omezení výkonu [10]. Python s frameworkem Django nabízí přehlednost a robustní strukturu, avšak může mít omezení při vysokém zatížení [10]. Enterprise platformy jako Spring Boot či MS.NET Core integrují pokročilé mechanismy pro správu databází, transakcí a RESTful API, čímž zajišťují vysokou úroveň bezpečnosti, výkonu a škálovatelnosti [12][13].

### 2.1.1 Srovnání technologií pro backend

*Tabulka 1: Srovnání backendových technologií. Zdroj:[1] [9] [10] [12] [13]*

| Kritéria | Express (Node.js) | Laravel (PHP) | Spring Boot (Java) | Django (Python) |
|---|---|---|---|---|
| Výkon a škálovatelnost | Vysoká efektivita při velkém množství paralelních požadavků díky asynchronní architektuře | Dobrá pro menší a středně velké projekty, škálovatelnost však klesá při extrémním zatížení | Výborná škálovatelnost a stabilní výkon i při vysokém zatížení díky pokročilým mechanismům správy transakcí | Vhodný pro menší až střední aplikace, škálovatelnost limitována při náročném zatížení |
| Vývojářská přívětivost a údržba | Podporuje modulární a flexibilní struktury, náročnější na znalosti asynchronního programování | Jednoduchá údržba, vhodný pro rychlý vývoj a prototypování, komunita usnadňuje vývoj | Pokročilá podpora a dlouhodobá údržba komplexních systémů, vyžaduje znalost Javy | Jednoduchá struktura a rychlá integrace, ideální pro prototypy, méně vhodné pro rozsáhlé projekty |
| Komplexnost a integrace | Ideální pro aplikace s vysokým počtem I/O operací, integrace vyžaduje manuální nastavení | Omezenější flexibilita při integraci složitějších systémů a více databází | Integrované pokročilé nástroje pro transakce, komunikaci mezi komponentami a API | Snadné základní integrace, složitější scénáře vyžadují doplňkové nástroje a knihovny |
| Náklady a dostupnost odborných znalostí | Zvyšující se dostupnost specialistů, nicméně vyžaduje hlubší odborné znalosti | Dobrá dostupnost odborníků díky široké komunitě a historii frameworku | Vyžaduje specializované znalosti, často vyšší náklady na vývoj, avšak výborná dlouhodobá návratnost | Velmi dobrá dostupnost odborníků díky silné komunitě, ale mohou nastat omezení při složitých projektech |

Výběr optimální backendové technologie závisí na konkrétních požadavcích projektu. Pro aplikace s vysokým počtem souběžných uživatelů a náročnými I/O operacemi je vhodné využít asynchronních platforem, jako je Express (Node.js), které zajišťují nízkou latenci a efektivní využití zdrojů, zatímco rozsáhlé podnikové systémy s přísnými požadavky na bezpečnost,

robustnost a integraci se nejlépe realizují pomocí enterprise řešení, například Spring Boot. Menší a jednodušší projekty pak mohou být efektivně vyvíjeny s využitím frameworku Laravel (PHP), který podporuje rychlý vývoj, nebo Django (Python), jenž je vhodný zejména pro prototypování a méně náročné aplikace [1] [9] [10] [12] [13].

## 2.2 Frontendové technologie

Frontendové technologie představují klíčovou složku vývoje webových aplikací, neboť zajišťují interaktivní a uživatelsky přívětivé rozhraní, které reaguje na dynamické změny v reálném čase. Hlavním cílem využívání frontendových frameworků je usnadnění vývoj**e.** Před rozšířením těchto technologií byly webové stránky vytvářeny pomocí staku HTML, CSS a JavaScriptu, který často ztěžoval práci. Dnes existuje dostatek různorodých frameworků, díky nimž si lze snadno vybrat takový, který nejlépe odpovídá konkrétním požadavkům. Mezi nejrozšířenější patří JavaScriptové frameworky, jako jsou React.js, Angular.js a Vue.js, které umožňují tvorbu komplexních jednostránkových aplikací (SPA). Tyto technologie využívají pokročilé techniky, například virtuální DOM pro efektivní aktualizaci uživatelského rozhraní, obousměrnou datovou vazbu zajišťující synchronizaci mezi modelem a zobrazením, a komponentový přístup podporující opětovné použití kódu. Díky těmto vlastnostem se frontendové technologie staly nepostradatelným nástrojem pro dosažení kvalitní uživatelské zkušenosti a rychlého načítání obsahu. Vzhledem k rostoucím nárokům uživatelů na rychlost a interaktivitu webových aplikací je tento aspekt stále důležitější [14].

### 2.2.1 Srovnání technologií pro frontend

*Tabulka 2: Srovnání frontendových technologií. Zdroj: [14] [15] [19]*

| Kritéria | Angular.js | React.js | Vue.js |
|---|---|---|---|
| Robustnost | Robustní a komplexní rámec vhodný pro rozsáhlé podnikové aplikace | Flexibilní řešení pro rychlý vývoj modulárního kódu | Lehké a přehledné řešení ideální pro menší až středně velké projekty |
| Učební křivka | Strmější učební křivka s vyššími nároky na konfiguraci | Nízká učební křivka, usnadňuje rychlý start | Velmi přívětivá pro začátečníky díky jednoduchosti a architektuře MVVM |
| Správa stavu | Podporuje komplexní správu stavu vhodnou pro náročnou logiku | Efektivní správa stavu umožňuje rychlou aktualizaci a údržbu kódu | Reaktivní datová vazba usnadňuje synchronizaci dat mezi komponentami |
| Komunita a podpora | Silná komunita a rozsáhlá podpora, ideální pro velké projekty | Velká a aktivní komunita | Rostoucí a aktivní komunita |
| Rychlost vývoje | Vyžaduje více konfigurace, může zpomalit vývoj, zejména u menších projektů | Velmi agilní, vhodné pro prototypování a rychlý vývoj | Rychlý vývoj díky nízké složitosti kódu a jednoduché konfiguraci |
| Dokumentace | Podrobná a komplexní dokumentace, usnadňuje pochopení složitých konceptů, ale může být náročná na studium | Dobrá dokumentace s rozsáhlým ekosystémem knihoven, avšak některé aspekty se spoléhají na komunitní zdroje | Výjimečně přehledná a srozumitelná dokumentace, usnadňuje učení a implementaci, |

Srovnání jednotlivých frontendových technologií odhaluje, že každý framework má své specifické přednosti i omezení, které ovlivňují volbu vhodného řešení pro daný projekt. Rozsáhlé a komplexní aplikace často těží z robustního rámce Angular.js, zatímco agilní vývoj modulárního kódu je efektivně řešen pomocí React.js. Vue.js pak kombinuje jednoduchost a přehlednost s moderními prvky, jako je reaktivní datová vazba a jej činí ideální volbou pro menší a středně velké projekty, kde je kladen důraz na rychlost vývoje a nízkou složitost kódu [14] [15] [19].

## 2.3 Relační databáze

Relační databáze představují základ správy dat, jejichž koncepty se ustálily již více než padesát let od zásadního příspěvku E.F. Codda, který v roce 1970 definoval relační model, založený na reprezentaci dat v řádcích a sloupcích. Tento model umožnil oddělit logickou strukturu dat od jejich fyzické reprezentace a otevřel cestu pro deklarativní dotazovací jazyk SQL, který si dodnes udržuje dominantní postavení díky své jednoduchosti a efektivitě při zpracování dotazů. Historický vývoj databázových systémů zaznamenal postupný přechod od hierarchických a síťových modelů, které byly omezeny svou flexibilitou, k relačnímu přístupu, jenž umožnil optimalizaci vyhledávání i manipulaci s daty pomocí sofistikovaných optimalizátorů, které překládají dotazy do efektivních exekučních plánů. Navzdory kritikám, například kvůli absenci úplné ortogonality či problémům s null hodnotami, se relační databáze staly osvědčeným a široce používaným řešením díky své robustnosti, konzistenci a možnosti správy velkých objemů dat ve vysoce konkurenčním prostředí. S nástupem nových požadavků na horizontální škálovatelnost a dynamickou správu dat s flexibilním schématem se na trhu objevují i systémy NoSQL, které částečně uvolňují některá omezení relačních databází, avšak díky dlouhodobému využívání a široké podpoře se relační databáze, jako MySQL, PostgreSQL, Microsoft SQL Server, SQLite či Oracle Database, stále řadí mezi nejpoužívanější systémy pro stěžejní aplikace [16].

### 2.3.1 PostgreSQL

PostgreSQL představuje open-source relační databázový systém, který vyniká svou stabilitou, důslednou podporou standardu SQL a širokou škálou pokročilých funkcí pro správu složitých datových struktur. Mezi jeho klíčové schopnosti patří například podpora rekurzivních dotazů, okenních funkcí a definice vlastních datových typů [20]. Ve srovnání s MySQL, přímým a zároveň jedním z nejpopulárnějších alternativních řešení, se PostgreSQL vyznačuje lepší schopností zvládat složitější datové struktury a náročné transakční operace. To potvrzují i srovnávací studie zaměřené na výkonnost a specifika jednotlivých databázových systémů [21]. Díky aktivní komunitě, rozsáhlé dokumentaci a pravidelným aktualizacím představuje PostgreSQL stabilní volbu pro aplikace, kde je prioritou zajištění konzistence a bezpečnosti dat.

## 2.4. Docker a Docker Compose

Docker je open-source platforma určená pro vývoj, nasazení a správu aplikací prostřednictvím kontejnerizační technologie. Kontejnery poskytují izolované a přenositelné prostředí, které zajišťuje konzistentní běh aplikací napříč různými systémy a infrastrukturami. Tento přístup

výrazně usnadňuje migraci a škálování softwaru, zefektivňuje vývojový cyklus a minimalizuje potřebu řešení komplikovaných konfigurací operačního systému [17].

Docker Compose pak rozšiřuje funkčnost Dockeru přes definice a koordinace vícekontejnerové aplikace prostřednictvím jediného konfiguračního souboru. Tento nástroj usnadňuje a automatizuje nasazení a správu závislostí mezi jednotlivými službami, jako jsou databáze, webové servery a další komponenty. Díky vzájemné komunikaci mezi kontejnery lze komplexní systémy spravovat mnohem efektivněji, aniž by bylo třeba každou službu nastavovat ručně [18].

Celkově Docker a Docker Compose představují moderní přístup k vývoji a správě softwaru, který podporuje agilní metody, kontinuální integraci a nasazení (CI/CD) pro DevOps flow s moznosti rychlou adaptaci na měnící se požadavky trhu. Platforma Docker dává jednoduchý přístup efektivní využití hardwarových zdrojů a zvyšuje spolehlivost a konzistenci běhu aplikací, zatímco Docker Compose koordinuje více kontejnerových služeb do jednotného, snadno spravovatelného celku [17] [18]. Tímto způsobem jsou tyto nástroje flexibilní a vysoce škálovatelné.

## 2.5 Java Spring Boot

Spring Boot je framework vyvinutý společností Pivotal, pro vývoj a nasazení aplikací založených na platformě Spring. Díky své auto-konfigurační povaze a přednastaveným inicializačním „starterům“ umožňuje rychle vytvářet samostatné, produkčně připravené aplikace s minimálním množstvím konfigurace.

### 2.5.1 Architektura a principy Spring Boot

Spring Boot staví na robustní architektuře tradičního Spring frameworku, avšak výrazně zjednodušuje konfiguraci díky mechanismu automatické konfigurace. Ten na základě dostupných knihoven a aktuálního kontextu prostředí volí vhodné nastavení, čímž zajišťuje optimální běh aplikace. Tato architektura podporuje vývoj mikroslužeb a modulárních systémů se snadnou správu a škálování. Auto-konfigurační moduly, tzv. „starters“, a vestavěný server (např. Tomcat) urychluje celý proces vývoje a nasazení aplikací, eliminují potřebu složitého manuálního nastavování [22].

### 2.5.2 Výhody a přínosy použití Spring Boot

Díky automatickým a přednastaveným konfiguracím je při vývoji možné soustředit se přímo na psaní aplikační logiky. Samostatné aplikace, které lze spustit jako běžné Java programy,

usnadňují nasazení v cloudovém prostředí. Spring Boot zároveň umožňuje snadnou integraci s databázemi, messaging, security a dalšími enterprise systémy. Dlouhodobou udržitelnost projektů podporuje rozsáhlá dokumentace, množství dostupných tutoriálů a aktivní komunita, která řešení Spring Boot pravidelně aktualizuje a rozvíjí [23].

### 2.5.3 Srovnání se standardním Spring frameworkem

Spring Boot představuje rozšíření standardního Spring frameworku, přičemž přidává vestavěnou automatickou konfiguraci a výrazně usnadňuje nasazení aplikací. Zatímco tradiční Spring aplikace vyžadují detailní konfiguraci pomocí XML souborů nebo anotací, Spring Boot tyto složitosti eliminuje díky svému auto-konfiguračnímu mechanismu. Tím dochází k rychlejšímu zahájení vývoje a zároveň se snižuje riziko chyb při nastavování aplikace. Kromě toho zjednodušuje jak samotný vývoj, tak i následnou údržbu. Díky vestavěnému serveru a předdefinovaným závislostem (tzv. *starters*) se není nutné zabývat ruční správou verzí a konfigurací externích knihoven [22], [24].

### 2.5.4 Omezení a perspektivy budoucího vývoje

Existují určité oblasti, kde může být přizpůsobení Spring Boot specifickým požadavkům náročnější. Například u velmi komplexních aplikací může být automatická konfigurace omezující, zejména v případech, kdy je zapotřebí detailní ladění nebo specifické optimalizace. Přestože Spring Boot výrazně zjednodušuje vývoj, zůstává přímým nástupcem rozsáhlého a komplexního Spring frameworku, jehož zvládnutí může být pro začínající vývojáře náročné. V případech, kdy projekt nevyžaduje pokročilé enterprise funkce, může být pro začátek vhodnější zvolit jednodušší backendová řešení, jako jsou Django, Laravel, Node.js nebo Flask. Pokud se však vývojář rozhodne využívat Spring Boot i přes jeho složitost, je pravděpodobné, že mnohé potenciální problémy a omezení bude možné v budoucnu překonat. Díky otevřenému zdrojovému kódu a aktivní komunitě dochází k pravidelným aktualizacím a rozšiřování funkcí. Výhled do budoucna je velmi slibný – Spring Boot totiž neustále posiluje svou integraci s cloudovými platformami a nástroji pro vývoj mikroslužeb, a zůstává tak i nadále klíčovou technologií pro enterprise vývoj [22], [24].

### 2.5.5 Z čeho se skládá Spring Boot projekt

Typická struktura projektu ve Spring Boot je navržena tak, aby podporovala logické rozdělení kódu a usnadňovala jeho správu. Hlavním vstupním bodem je třída obsahující metodu main, která se nachází v hlavním balíčku a inicializuje Spring kontext. Zdrojový kód se obvykle umisťuje do složky ***src/main/java***, zatímco konfigurační soubory, jako je

application.properties nebo application.yml, jsou umístěny v adresáři ***src/main/resources***. Součástí projektu jsou také složky pro statické zdroje a šablony, pokud aplikace využívá serverové renderování. Projekt je obvykle organizován do logických vrstev, kde jsou odděleny kontrolery, služby, repozitáře a modely. Testy se pak nacházejí v adresáři ***src/test/java***. Tato struktura je často generována pomocí nástroje Spring Initializr [32], který poskytuje správně nakonfigurovaný základ pro jejich projekty [22] [23] [24].

## 2.6 Vue.js

Vue představuje přístup k vývoji uživatelských rozhraní, který klade důraz na jednoduchost, modularitu a vysokou výkonnost díky svým inovativním principům a architektonickým řešením [19].

### 2.6.1 Architektura a principy Vue.js

Vue je postaveno na architektuře MVVM, která zajišťuje oddělení datové logiky od prezentační vrstvy prostřednictvím reaktivní datové vazby s automatickou aktualizaci uživatelského rozhraní při změnách v datech. Framework využívá virtuální DOM, který optimalizuje manipulaci se skutečným DOM a tím zvyšuje výkon aplikací. Dále je Vue založen na přístupu orientovaném na komponenty, kdy každá komponenta obsahuje šablonu, skript a styly v jednom souboru, podporuje modularitu a opakované použití kódu. Vue také přináší statickou kontrolu a robustnější vývoj podporou TypeScript, a prostřednictvím platformy Weex lze Vue využít i pro tvorbu mobilních aplikací [25].

### 2.6.2 Výhody a přínosy použití Vue.js

Vue nabízí rychlý start i pro začátečníky díky své značné flexibilitě, intuitivní syntaxi a přehledné dokumentaci, která výrazně urychluje vývoj. Reaktivní systém frameworku zajišťuje bezproblémový, efektivní a plynulý chod aplikace, díky němuž se jakákoli změna dat okamžitě projeví v uživatelském rozhraní. Velkou výhodou je rovněž možnost efektivně vyvíjet a spravovat aplikace, jak menší, tak i středně velké bez nárůstu složitosti kódu. Díky komponentové architektuře Vue lze snadno strukturovat i rozsáhlejší projekty a udržet jejich čitelnost a přehlednost [26].

### 2.6.3 Nedostatky Vue.js

I přes řadu výhod má Vue i své omezení. Ve srovnání s některými konkurenty může být ekosystém a komunita menší. To může vést k omezené dostupnosti specializovaných knihoven a rozšíření. Další výzvou je rychlý vývoj frameworku, vedoucí k častým aktualizacím a nutnosti přizpůsobovat stávající projekty novým verzím a může to komplikovat dlouhodobou údržbu.

Tyto faktory mohou představovat určité potíže zejména v komplexnějších enterprise projektech [26].

### 2.6.4 Z čeho se skládá Vue projekt

Typický Vue projekt je strukturován tak, aby podporoval modulární vývoj a efektivní správu kódu. Hlavním vstupním bodem je soubor ***main.js*** (nebo ***main.ts***, pokud je využit TypeScript), kde se inicializuje instance Vue a nastavují se globální konfigurace, včetně integrace routeru a správy stavu, například pomocí Vuex. Složka komponent obsahuje jednotlivé .vue soubory, kde každá komponenta zahrnuje šablonu, logiku a styly. Dále projekt obvykle obsahuje složku pro definici tras, která zajišťuje navigaci mezi komponentami, a složku s assety, kde jsou umístěny statické zdroje jako obrázky, styly a další soubory [25].

# PRAKTICKÁ ČÁST

## 3. Vybrané komunitní platformy a intranety

Intranet představuje vnitřní síť, určenou pro výměnu informací, spolupráci a automatizaci procesů uvnitř organizace nebo vzdělávací instituce. Díky omezenému přístupu k těmto systémům je zajištěna vysoká úroveň bezpečnosti a centralizovaného řízení dat. To je zvláště důležité pro vzdělávací instituce, kde má ochrana osobních údajů studentů a učitelů zásadní význam. Možnosti a výhody intranetových systémů jsou široce reflektovány v odborné literatuře a standardech [1]

### 3.1 Facebook

Tato sociální globální síť, známá pod označením Facebook od společnosti Meta využívají jednotlivci, rodiny, firmy, neziskové organizace i vládní instituce. Uživatelé si zde vytvářejí osobní profily, sdílejí fotografie, videa a příspěvky, komunikují prostřednictvím soukromých zpráv a mohou se zapojovat do skupin, které podporují komunitní interakci. Kromě toho Facebook nabízí Marketplace pro nákup a prodej zboží přímo na platformě, a řadu dalších nástrojů pro event management, reklamu a analytiku. Přestože platforma poskytuje rozsáhlé možnosti, pro některé uživatele může představovat omezení zejména z hlediska ochrany osobních údajů a moderace obsahu [27].

### 3.2 Reddit

Reddit – komunitně orientovaná diskusní platforma, provozovaná společností Reddit Inc., využívají studenti, nadšenci, profesionálové a experti z různých oborů. Uživatelé Redditu se zapojují do diskusí ve specializovaných fórech, tzv. subredditech, kde mohou publikovat příspěvky, hlasovat a komentovat obsah. Tato platforma umožňuje relativní anonymitu a podporuje živé a dynamické diskuse pro sdílení znalostí a názorů. Na druhou stranu však může být Reddit náchylný k problémům s moderací a občas nese riziko šíření kontroverzního nebo polarizujícího obsahu [28].

### 3.3 Telegram

Komunikační aplikace Telegram, kterou využívají jednotlivci, skupiny, firmy, a dokonce i vládní agentury, a je známá především svou vysokou úrovní zabezpečení a šifrování. Uživatelé Telegramu mohou využívat nejen standardní funkce pro posílání zpráv, ale také skupinové chaty, kanály pro vysílání informací a integrované miniaplikace (tg Mini Apps), které umožňují provádět například nákupy nebo rezervace přímo v rámci aplikace. Tato rozšířená funkčnost

podporuje inovativní využití platformy, ale může také čelit regulacím v určitých zemích a potenciálním bezpečnostním rizikům, zejména v případě veřejně přístupných kanálů [29].

## 3.4 LinkedIn

Profesionální sociální síť LinkedIn, vlastněná společností Microsoft, kterou používají pracovníci, personalisté, odborníci a firmy pro budování kariérních kontaktů a profesní komunikaci. Platforma umožňuje vytváření detailních profilů, sdílení odborných článků a informací o pracovních příležitostech, a také podporuje účast ve skupinách zaměřených na konkrétní odvětví. Mezi klíčové funkce patří možnost vyhledávání pracovních nabídek, navazování kontaktů a budování profesionálních sítí. Omezení LinkedInu spočívají zejména ve formálním prostředí, které může omezovat osobní interakce ve srovnání s volnějšími sociálními sítěmi [30].

## 3.5 Instagram

Instagram od Meta je vizuálně orientovaná sociální síť, kterou využívají jednotlivci, firmy a influenceři pro sdílení fotografií, videí a příběhů. Uživatelé zde vytvářejí atraktivní profily a prostřednictvím funkcí jako Stories, Reels nebo živých přenosů mohou efektivně komunikovat se svým publikem. Platforma nabízí také řadu nástrojů pro podniky, jako jsou analytika, placené reklamy a možnost správy kampaní. Ačkoliv Instagram efektivně podporuje budování vizuální identity a zapojené komunity, platforma zároveň čelí kritice v souvislosti s ochranou osobních údajů, nedostatečnou moderací obsahu a jistými omezeními pro uživatele i firmy [31].

# 4 Analýza

Během práce byla provedena analýza dotazů od studentů prostřednictvím dotazníku. Dotazník byl sestaven ve dvou jazycích – češtině a ruštině – pro porovnání požadavků mezi zahraničními a českými studenty prezenční formy. Byla také připravena verze dotazníku pro studenty Erasmu v angličtině, avšak z nedostatku dat nebude v této práci zohledněna. Dotazníky byly distribuovány prostřednictvím skupinových chatů na sociálních sítích, studentské pošty a tištěných letáků na fakultách. Na základě výsledků dotazníku budou formovány požadavky na aplikaci. Dále je důležité upřesnit, že u otázek s odpověďmi od 1 do 5, 1 znamená nejnižší hodnocení a 5 nejvyšší. Toto upřesnění je nutné kvůli rozdílům mezi českým a rusky mluvícím hodnotícím systémem (v rusky mluvících zemích 5 bodů odpovídá 1 bodu v Česku).

## 4.1 Uživatelské chování a zkušenosti

Grafy vztahující se k této části jsou uvedeny v příloze A.

Zaměřeno na to, jak studenti interagují s univerzitou, jaké mají zkušenosti a jaké problémy vnímají. Na základě analýzy dotazníku lze konstatovat, že studenti mají relativně nízký pocit začlenění do univerzitního života, průměrné hodnocení činí jen lehce nad hodnotou 2 z 5 možných. To koresponduje s nízkou četností jejich účasti na univerzitních akcích, většina respondentů navštěvuje tyto akce jen velmi zřídka. Výsledky naznačují, že hlavní příčinou není nedostatek informací, jelikož respondenti hodnotí dostupnost informací o událostech relativně kladně. Dominantním zdrojem informací jsou sociální sítě, zejména Instagram. Tato zjištění zdůrazňují relevanci navrhované univerzitní platformy, která může centralizovat informace o událostech a nabízet snadnější způsoby komunikace mezi studenty. Vysoká ochota studentů sdílet studijní materiály a identifikovaná potřeba komunitního propojení poukazují na potenciál plánované aplikace ke zvýšení jejich zapojení do univerzitního života. Jelikož studenti nejvíce oceňují kulturní a společenské akce, aplikace by měla klást důraz nejen na akademické aktivity, ale také na vytváření komunit a usnadnění sociální interakce. Navržená platforma tak může efektivně řešit problém nízkého začlenění studentů do univerzitního života prostřednictvím cíleného propojení komunikačních nástrojů, sdílení zdrojů a podpory neformální interakce mezi studenty.

## 4.2 Přednástupová fáze a informační bariéry

Grafy související s touto problematikou jsou uvedeny v příloze B.

Zaměřeno na problémy a potřeby studentů před nástupem na univerzitu. Výsledky analýzy odpovědí odhalují, že studenti před nástupem na univerzitu čelí značným potížím při získávání informací o životě na škole, ubytování, kroužcích a administrativních postupech. Průměrné hodnocení těchto potíží se pohybuje kolem 2,9 bodu ze 5, a tohle naznačuje, že bariéry nejsou zanedbatelné. Respondenti opakovaně uvádějí nedostatek podpůrné komunity a osobních kontaktů, které by jim usnadnily adaptaci v novém prostředí. Studentům schází spolehlivá informační podpora, která by umožnila rychlejší orientaci v prostředí univerzity.

Důležitou roli hraje očekávání přínosu online komunity, která by studentům poskytla potřebné informace ještě před nástupem na fakultu. Průměrné hodnocení přínosu takové platformy dosahuje hodnoty přibližně 3.7 bodu, to svědčí o pozitivním postoji vůči integraci informací do jednoho centrálního systému. Téměř všichni dotazovaní (99 z 118) potvrzují, že možnost konzultovat záležitosti s dalšími uchazeči prostřednictvím uzavřeného fóra by usnadnila adaptaci na univerzitní prostředí.

Tato zjištění podtrhují význam navrhované platformy, jejímž cílem je překonat informační bariéry a poskytnout studentům prostor pro přednástupovou komunikaci. Platforma by měla centralizovat informace o univerzitním životě, nabídnout přístup ke zkušenostem starších studentů a umožnit vzájemnou výměnu zkušeností mezi uchazeči. Navržené řešení tak usiluje o vytvoření prostředí, které podpoří rychlou a efektivní adaptaci nových studentů. Celkově výsledky analýzy potvrzují, že vytvoření jednotné online komunity může eliminovat nedostatek informací, posílit mezilidské vazby a výrazně přispět k lepší integraci studentů do akademického prostředí.

## 4.3 Požadavky na aplikaci

Grafy ilustrující požadavky studentů na funkcionalitu aplikace jsou uvedeny v příloze C.

Otázky, které přímo souvisejí s návrhem funkcí a vlastností navrhované platformy. Odpovědi na otázky zaměřené na požadavky na navrhovanou platformu ukazují, že studenti mají velký zájem o sjednocení různých funkcí do jednoho systému. Respondenti vyjadřují potřebu mít centralizovaný přehled událostí, diskusního fóra a tržiště, abych ta aplikace umožnila rychlou orientaci a snadnou komunikaci. Jednotlivci dávají přednost nástroji, který by jim nabídl možnost získat základní informace o ostatních studentech, přehled studijního rozvrhu a upozornění na nadcházející zkoušky či akce katedry. Odezva na myšlenku aktivního zveřejňování vlastních materiálů a příspěvků naznačuje, že studenti očekávají interaktivitu a ochotu podílet se na obsahu. Interaktivní vizualizace kampusu, například formou mapy budov, knihoven či učeben, bývá vnímána jako užitečný prvek, který pomůže novým i stávajícím studentům lépe se orientovat v prostředí fakulty.

Výsledky z dotazníku zdůrazňují zájem o online komunitu, jež nabídne přehled informací ještě před nástupem na fakultu. To by mohlo mít pozitivní dopad na adaptaci do nového prostředí, jelikož by takový systém usnadnil komunikaci mezi uchazeči a studenty, toto zjištění vyplývá z představ respondentu. Uživatelé preferují propojení aplikace s jejich studijním rozvrhem pro plánování a lepší organizaci času, a rovněž podporu sdílení studijních materiálů v rámci uzavřených či veřejných skupin. Zájem o tyto funkce svědčí o potřebě jednotného řešení, které zjednoduší přístup k důležitým informacím a podpoří komunitní vazby mezi studenty.

Cílem je odstranit roztříštěnost informací napříč různými systémy a nahradit ji jednotným, přehledným prostředím. Navrhované řešení by mělo nabídnout moderní prostředí, které usnadní interakci, posílí mezilidské vztahy a přispěje k celkové integraci studentů do akademického života. Tyto poznatky poskytují solidní základ pro další vývoj a optimalizaci univerzitní aplikace, která by splňovala reálné potřeby cílové skupiny.

## 4.4 Porovnání odpovědí mezi rusky a česky mluvícími studenty

Grafy použité v této části jsou součástí přílohy D.

Pro úplnost je vhodné upřesnit, že pod označením „rusky mluvící studenti“ nebo ru-studenty nejsou míněni pouze občané Ruské federace, ale širší skupina studentů hovořících rusky jako svým dorozumívacím jazykem ve skupině – tedy například studenti z Ukrajiny, Běloruska, Kazachstánu, Uzbekistánu a dalších států postsovětského prostoru. Tito studenti tvoří nezanedbatelnou část univerzitní komunity.

Mezi ruskými a českými studenty byly identifikovány určité významné rozdíly, které je nutné zohlednit při návrhu univerzitní online platformy. Nejvýraznějším zjištěním je vyšší ochota rusky mluvících studentů sdílet studijní materiály, kde průměrná hodnota dosáhla 4,06 oproti 3,68 u českých studentů. Tento rozdíl naznačuje, že platforma by měla být navržena s důrazem na nástroje pro jednoduché a bezpečné sdílení obsahu, přičemž u českých uživatelů bude nezbytné tuto funkcionalitu zároveň doprovodit edukací a podporou komunity, která výší důvěru v její používání.

V otázce integrace do univerzitního života se ruská skupina hodnotila mírně výše (2,31 bodu) než česká (2,07 bodu), ukazuje na obecnou potřebu zlepšit prostředí pro začlenění studentů. Tomu odpovídá také vyšší míra využívání osobních kontaktů a přátelských vazeb jako klíčového adaptačního nástroje. U ruských studentů to uvedlo 85,7 %, zatímco u českých 65,3 %. Pomoc od starších studentů zafungovala u 42,9 % ruskojazyčných respondentů, ale jen u 16,9 % českých. Z těchto dat vyplývá důležitost zavedení mentorského systému a podpory přirozené interakce mezi studenty napříč ročníky.

Rozdíly jsou patrné také ve zdrojích informací o univerzitních událostech. Zatímco Instagram dominuje u obou skupin, mezi ruskými studenty ho uvedlo 53,1 %, oproti 47,5 % u českých. Čeští studenti však více využívají oficiální web univerzity (18,6 %) než ruští (10,2 %), zatímco ruská skupina více čerpá z offline reklamy (letáky apod.). Tim, aplikace vyžaduje přizpůsobení formy komunikace cílovým zvyklostem jednotlivých skupin.

V preferencích typů událostí převládá u obou skupin zájem o zábavné akce, avšak u rusky mluvících je tato preference ještě silnější (67,3 %) oproti českým studentům (54,2 %). Kulturní a vědecké akce mají srovnatelnou váhu, a to potvrzuje, že univerzitní prostředí musí nabízet pestrou škálu aktivit nejen akademického, ale i komunitního charakteru, ideálně s podporou nástroje pro vytváření a správu komunitních skupin a kalendářů.

V otázce obtíží při nástupu je situace podobná. Významný rozdíl však tvoří jazyková bariéra – u ruských studentů ji uvedlo 40,8 %, u českých jen 3,4 % (a myslím, že to byli také cizí studenti). Pro platformu to znamená nutnost jazykové lokalizace a podpory vícejazyčné komunikace. Také absence známých či komunity byla častěji zmíněna mezi ruskými studenty (63,3 % vs. 48,3 % u českých), potvrzuje to nutnost komunitně orientovaného řešení s důrazem na prvky přednástupového propojení uchazečů a mentorů.

Přínos online platformy před nástupem byl pozitivně hodnocen v obou skupinách, ale ruská skupina častěji udělovala nejvyšší známky – celkem 38,8 % uvedlo hodnotu 5, stejně tak i 38,8 % hodnotu 4. U českých studentů byla větší rozptýlenost odpovědí, přičemž 41,5 % zvolilo 4 a 19,5 % nejvyšší hodnocení. Tato zpětná vazba potvrzuje potřebu přístupného, přehledného a centralizovaného nástroje pro komunikaci, sdílení informací a zapojení nově příchozích studentů ještě před samotným nástupem.

Ve všech ostatních oblastech byly rozdíly mezi skupinami marginální a pohybovaly se ve statisticky nevýznamném rozpětí, a proto nebyly v tomto rozboru podrobněji hodnoceny. Všechny výše uvedené rozdíly by však měly být zohledněny v architektuře, lokalizaci, návrhu rozhraní i komunitních funkcích univerzitní aplikace.

Na základě osobní zkušenosti lze navíc doplnit, že rusky mluvící komunita se často přirozeně odděluje od českého prostředí ve fázi navazování vztahů na univerzitě. Tato tendence není hodnocena ani pozitivně, ani negativně, je však zřejmé, že jedním z důvodů bývá komplikované začlenění na začátku studia, a tím i menší ochota navazovat kontakt mimo jazykovou skupinu. Z tohoto důvodu má navrhovaná platforma i sekundární – osobnější – cíl: propojit jazykově odlišné skupiny v rámci univerzitní komunity a podpořit integraci studentů se zahraniční zkušeností do širšího akademického prostředí. Platforma se tak může stát nejen praktickým nástrojem, ale i prostředím pro vzájemné porozumění a podporu napříč kulturami.

# 5. Požadavky

Na základě provedené analýzy dotazníků a diskusí se studenty vyšlo najevo, že hlavním problémem je roztříštěnost informací a nedostatek centralizace jednotlivých dat a informací, zejména pro nově přijaté a zahraniční studenty. Studenti projevují zájem o centrální platformu, která by sdružovala funkce pro komunikaci, plánování událostí a sdílení studijních materiálů. Klíčovou roli hraje také možnost získat základní informace o univerzitě ještě před nástupem, včetně přehledné prezentace rozvrhu a možností zapojení do zájmových skupin. Tato kapitola proto shrnuje požadavky na navrhovanou aplikaci, které vycházejí z uvedených zjištění. Nejprve jsou představeny funkční požadavky, jež popisují, co by systém měl umět, a poté nefunkční požadavky, zaměřené na aspekty bezpečnosti, dostupnosti či použitelnosti. Následně bude v dalších podkapitolách objasněna struktura databáze a popsáno vizuální modelování procesů prostřednictvím diagramů aktivit a případů užití.

## 5.1 Funkční požadavky

*Tabulka 3: Funkční požadavky webové aplikace (zdroj vlastní)*

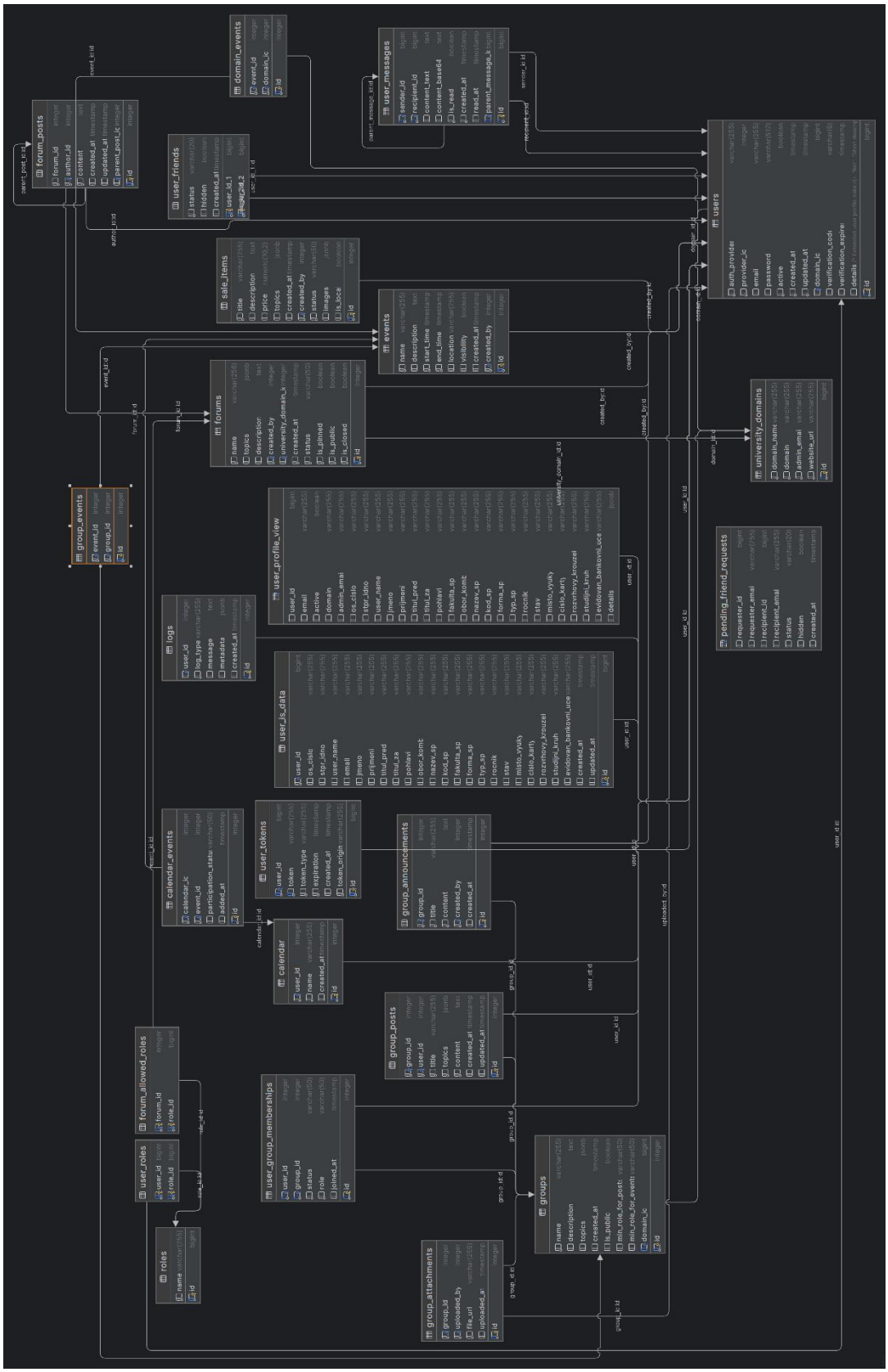
| | | |
|---|---|---|
| F1 | Registrace uživatelů | Aplikace musí umožňovat registraci uživatelů pomocí univerzitního e-mailu či ověřené identity (např. LDAP). |
| F2 | Přihlášení | Aplikace musí zprostředkovat bezpečné přihlášení (login/heslo, případně 2FA) do systému. |
| F3 | Správa profilů | Aplikace musí umožnit úpravu uživatelského profilu (jméno, foto, studijní skupina) a zobrazení profilu. |
| F4 | Fórum a diskuse | Aplikace musí umožňovat vytváření diskusních témat, odpovědí a zobrazování vláken ve fóru. |
| F5 | Komunitní skupiny | Aplikace musí umožňovat zakládat a řídit skupiny (přidávat a odebírat členy, řídit obsah). |
| F6 | Kalendář univerzitních akcí | Aplikace musí poskytovat modul pro plánování a sledování událostí (přednášky, setkání, akce) s možností registrace. |
| F7 | Integrace studijního rozvrhu | Aplikace musí podporovat načítání rozvrhu studenta (např. z univerzitního systému) pro zobrazení v kalendáři. |
| F8 | Bazar | Aplikace musí umožňovat vložení a správu inzerátů (prodej, výměna, hledání), a to výhradně přihlášenými uživateli. |
| F9 | Systém notifikací | Aplikace musí umět zasílat notifikace (např. o nových příspěvcích, odpovědích, akcích) buď e-mailem, nebo jinou formou (push, interní upozornění), dle preferencí uživatele. |
| F10 | Lokalizace | Aplikace musí mít překlady do českého a anglického jazyků |

## 5.2 Nefunkční požadavky

*Tabulka 4: Nefunkční požadavky webové aplikace (zdroj vlastní)*

| | | |
|---|---|---|
| N1 | Bezpečnost | Aplikace musí implementovat šifrovanou komunikaci (HTTPS), chránit se proti útokům (XSS, CSRF, SQL injection) a umožnit řízení přístupových práv (role). |
| N2 | Dostupnost | Aplikace by měla být dostupná 24/7 s definovanou maximální dobou výpadku. Je nutné zavést pravidelné zálohy databáze a případné hot-standby řešení. |
| N3 | Použitelnost (UX/UI) | Rozhraní musí být responzivní, přehledné a konzistentní. Orientace v aplikaci má být intuitivní pro studenty i správce, s ohledem na zásady UX designu. |
| N4 | Provozní prostředí | Aplikaci musí být možné spustit a nasadit na běžných serverech s OS Linux či Windows (případně kontejnerizovaně), s minimálními softwarovými závislostmi. |
| N5 | Lokalizace a jazyk | Aplikace by měla umožnit vícejazyčné rozhraní (primárně CZ, případně EN) podle nastavení uživatele. |
| N6 | API rozhraní | Aplikace by měla nabízet REST API pro mobilní klienty či jiné systémy (autentizovaný přístup k datům). |
| N7 | Rozšiřitelnost a modularita | Architektura aplikace (např. vícevrstvá MVC) má být navržena tak, aby usnadňovala přidávání nových modulů (např. integrace messengeru, rozhraní pro mobilní aplikace). |

# 6 Databáze



*Obrázek 6: Uml diagram databaze (vl.zdroj)*

Navržená relační databáze aplikace slouží k centralizaci a správě dat webové aplikace určené pro komunitní platformu. Databáze je navržena s ohledem na budoucí rozšiřitelnost a flexibilitu aplikace, přičemž některé její části jsou připraveny pro implementaci plánovaných funkcí.

## 6.1 Tabulky používané v aktuální verzi aplikace

***users*** – tabulka pro uchování základních údajů o uživatelích aplikace. Obsahuje informace o emailu, heslu (v hashované podobě), autentizačním poskytovateli (*auth_provider*) a podrobnosti profilu v JSON formátu pro flexibilní rozšiřování dat profilu.

***user_roles*** – propojuje uživatele s jejich rolemi a umožňuje snadné přiřazování oprávnění uživatelům prostřednictvím rolí.

***roles*** – obsahuje definici uživatelských rolí a jejich názvů. V současné verzi aplikace je tato tabulka ručně plněna předdefinovanými hodnotami na backendu.

***university_domains*** – uchovává informace o doménách jednotlivých univerzitních institucí. Data v této tabulce jsou rovněž manuálně zadávána a spravována z backendu aplikace.

***user_tokens*** – tabulka tokenů je vytvořena kvůli bezpečnosti a správě autentizačních tokenů. Oddělení tokenů od uživatelské tabulky zajišťuje bezpečnější a efektivnější správu, obnovu tokenů a lepší auditování přístupu.

***user_friends*** – spravuje přátelské vazby mezi uživateli, včetně stavů žádostí o přátelství (pending, approved).

***user_messages*** – uchovává zprávy v rámci soukromé komunikace mezi uživateli, včetně podpory obrázků v base64 a odpovědí na zprávy (referencování rodičovských zpráv).

***groups*** – umožňuje správu skupin, včetně názvů, popisů, témat, veřejnosti a minimální role pro přispívání.

***user_group_memberships*** – spravuje členství uživatelů ve skupinách, status členství a přiřazené role v rámci skupiny.

***forums*** – slouží k vytváření a správě diskusních fór, definic témat, příspěvků, stavu a přístupu na základě rolí.

***forum_posts*** – obsahuje jednotlivé příspěvky ve fórech, včetně hierarchických odpovědí.

***group_posts*** – slouží k ukládání příspěvků ve skupinách včetně názvu, obsahu a témat.

***user_is_data*** – obsahuje detailní informace o uživatelích získané integrací s IS/STAG, zejména data studentů, jako jsou čísla studijních průkazů, obory studia atd.

## 6.2 Pohledy (Views)

***user_profile_view*** – pohled spojuje data uživatelů (***users***) a rozšířené informace z tabulky ***user_is_data***. Používá se pro rychlé získání detailního uživatelského profilu a integraci s jinými částmi aplikace.

***pending_friend_requests*** – pohled poskytuje přehled čekajících žádostí o přátelství mezi uživateli. Umožňuje snadné zpracování žádostí přímo v aplikaci.

## 6.3 Tabulky aktuálně nevyužívané

Níže uvedené tabulky jsou připraveny jako základ pro plánované přidáni funkcionality kalendáře, událostí, souborových příloh a tržiště (bazar), ale zatím nejsou implementovány.

***events*** – navržena pro budoucí správu akcí a událostí pořádaných univerzitou nebo studenty.

***calendar*** a ***calendar_events*** – připravené k implementaci osobního nebo skupinového kalendáře s plánováním účasti na událostech.

***domain_events*** – umožní správu událostí spojených s konkrétní univerzitní doménou.

***group_events*** – připravena pro ukládání událostí pořádaných v rámci skupin.

***group_announcements*** – navržena pro oznámení uvnitř skupin mezi členů.

***group_attachments*** – tabulka plánovaná pro ukládání odkazů na soubory připojené k příspěvkům skupin, s možností uložení v externím NoSQL úložišti nebo cloudovém úložišti optimalizovaném pro větší objemy dat.

***sale_items*** – plánovaná tabulka tržiště (bazaru) pro prodej nebo výměnu předmětů mezi uživateli, včetně podpory obrázků a kategorií produktů.

V současné podobě aplikace tyto tabulky připraveny jako strukturovaný makety, které zajistí plynulé budoucí rozšíření funkcionalit bez zásadních změn v datovém schématu.

## 6.4 Realizace v aplikaci

Relační databáze aplikace je spuštěna pomocí technologie Docker Compose. Kontejner databáze je definován pomocí oficiálního obrazu PostgreSQL verze 17. Data jsou persistentní díky využití volumes. Pro inicializaci databázového schématu se používá předem připravený

DDL skript (***init.sql***). Díky této konfiguraci je zajištěno rychlé nasazení a konzistentní prostředí databáze v různých vývojových a produkčních scénářích.

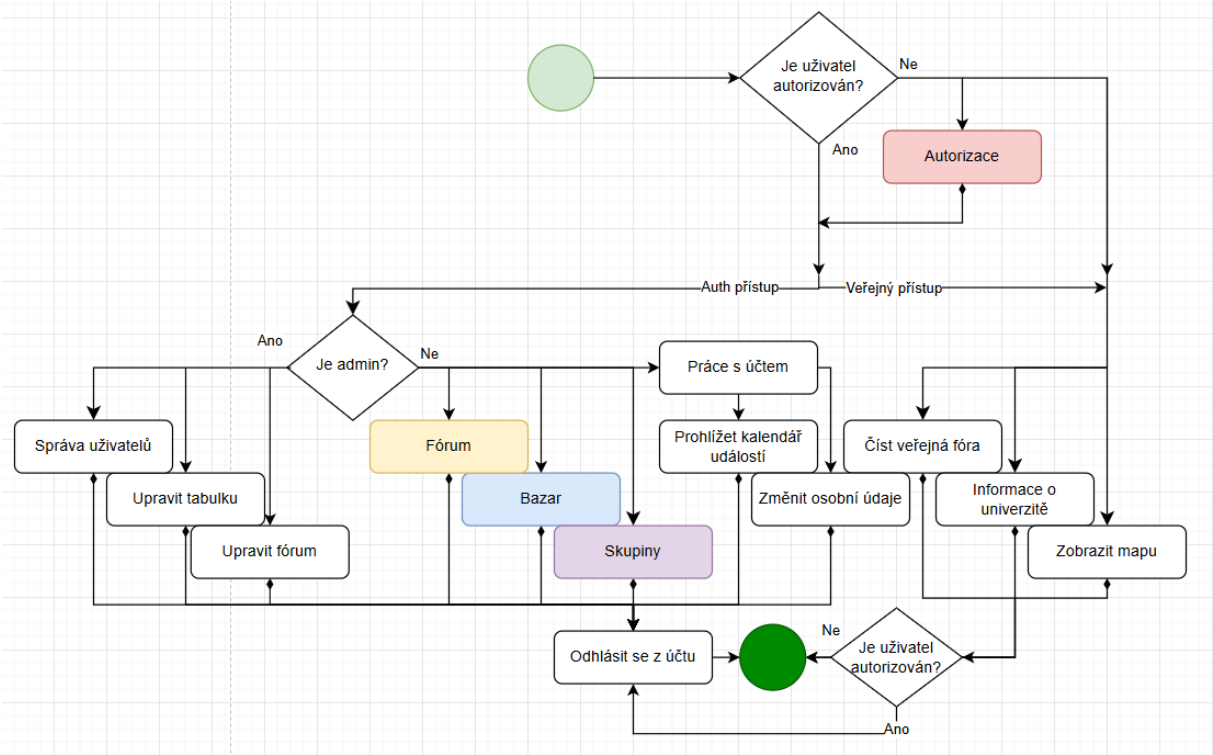
## 6.5 Shrnutí

Relační databáze je navržena modulárně s jasným oddělením aktuálně implementovaných a plánovaných částí aplikace. Použití samostatných tabulek, zejména pro tokeny uživatelů (*user_tokens*), zvyšuje bezpečnost a usnadňuje správu autentizace. Ručně plněné tabulky (*university_domains, roles*) poskytují kontrolu nad klíčovými aspekty fungování aplikace. Tabulky, které aktuálně nejsou využívány, jsou navrženy tak, aby tvořily stabilní základ pro plánované budoucí funkcionality, jako jsou kalendářové funkce, správa událostí, tržiště nebo integrace externího úložiště. Celkový návrh databáze umožňuje efektivní správu dat, snadnou údržbu a škálovatelnost celého systému.

# 7 Diagramy aktivit

V této části jsou představeny hlavní aktivity uživatele při práci s navrženou komunitní platformou, a to včetně autorizace, správy fóra, práce s bazarem a fungování skupin. Jednotlivé diagramy aktivit poskytují přehled klíčových kroků a rozhodovacích bodů, které uživatel nebo systém provádí v rámci konkrétních funkcí.

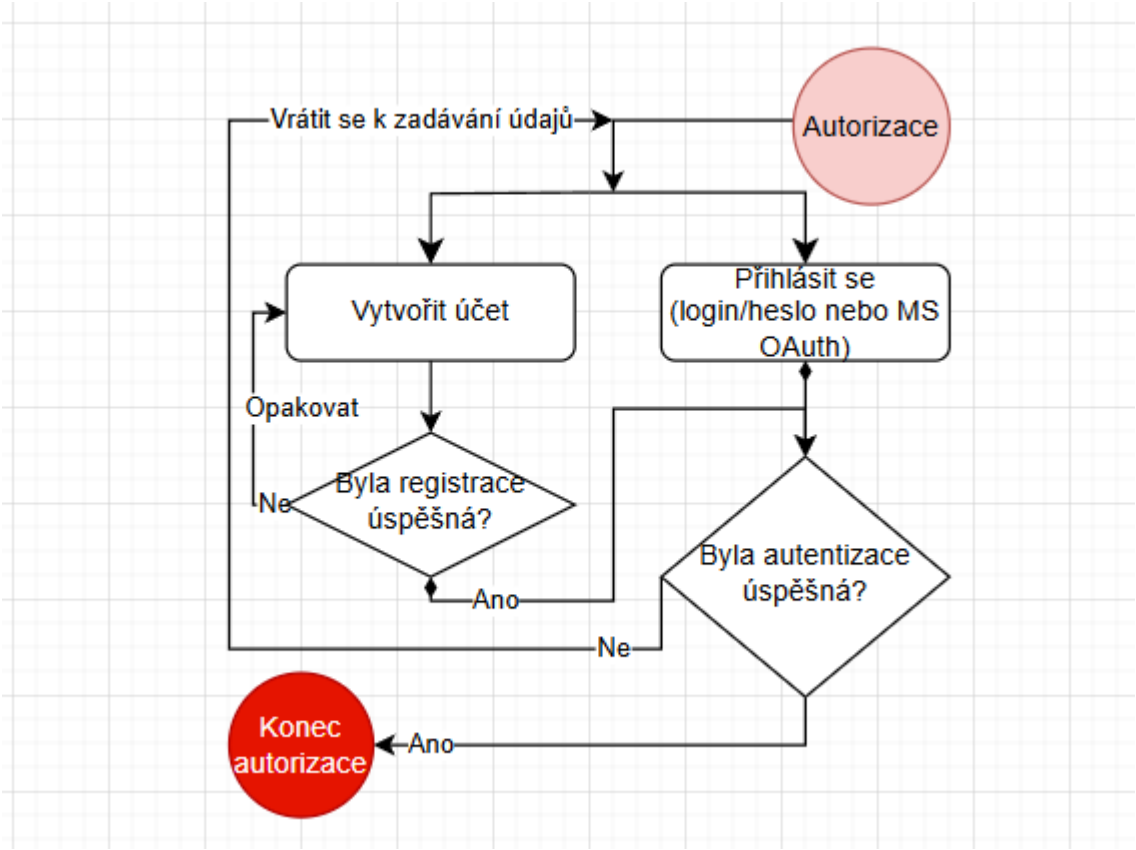
## 7.1 Hlavní diagram




*Obrázek 7: Hlavní diagram aktivit (vl. zdroj)*

Hlavní diagram aktivit znázorňuje obecný postup uživatele po spuštění aplikace. Po případné autorizaci (pokud je vyžadována) může uživatel přistupovat k jednotlivým funkcím, jako je fórum, bazar, skupiny, osobní údaje či správa obsahu. Rozhodovací body umožňují rozlišit, zda je uživatel autorizován, a podle toho mu nabídnout příslušné možnosti (například editaci profilu či vytváření nových příspěvků). Diagram tak poskytuje komplexní pohled na to, jak se uživatel v aplikaci pohybuje a jaké funkce jsou pro něj dostupné v závislosti na úrovni přístupu.

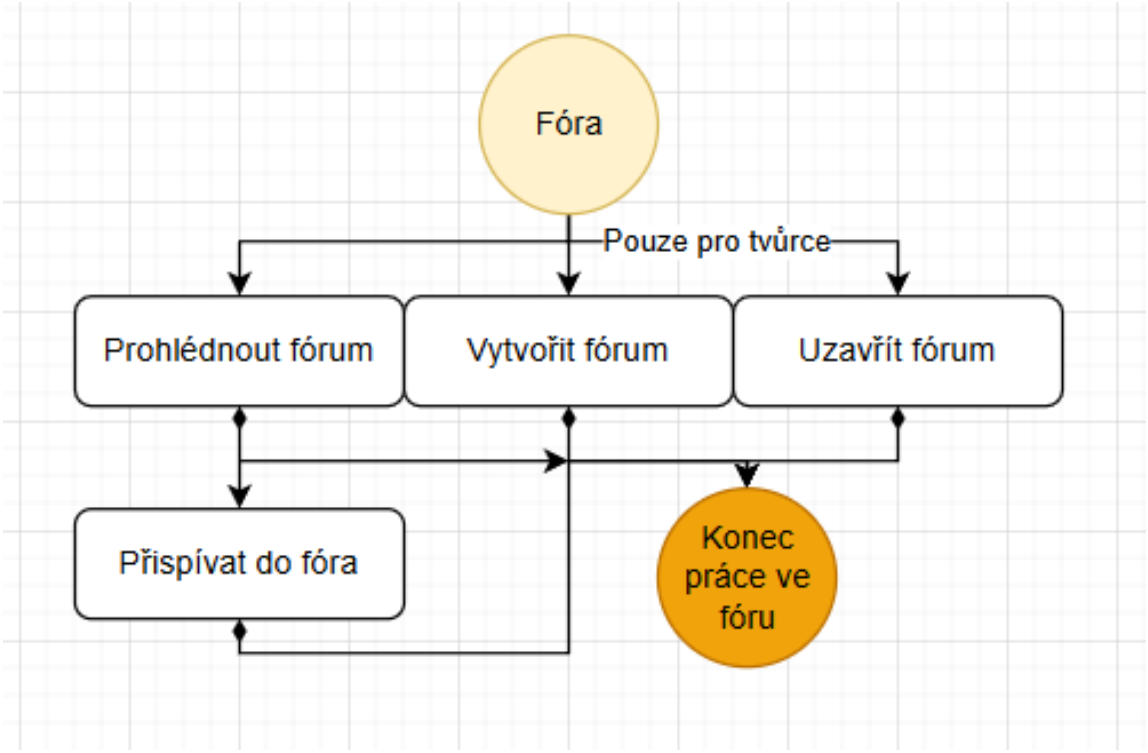
## 7.2 Diagram autorizace




*Obrázek 8: Diagram autorizace (vl. zdroj)*

Tento diagram se zaměřuje na proces vytváření uživatelského účtu a následné přihlášení. Nejprve uživatel rozhoduje, zda si chce založit nový účet, nebo se přihlásí pomocí stávajících přihlašovacích údajů či externí služby (např. OAuth). Po úspěšné registraci či autentizaci může pokračovat v plném využívání aplikace. Diagram zobrazuje rozhodovací uzly, kdy se ověřuje správnost zadaných údajů, a v případě neúspěchu umožňuje vrátit se k zadávání přihlašovacích či registračních dat.

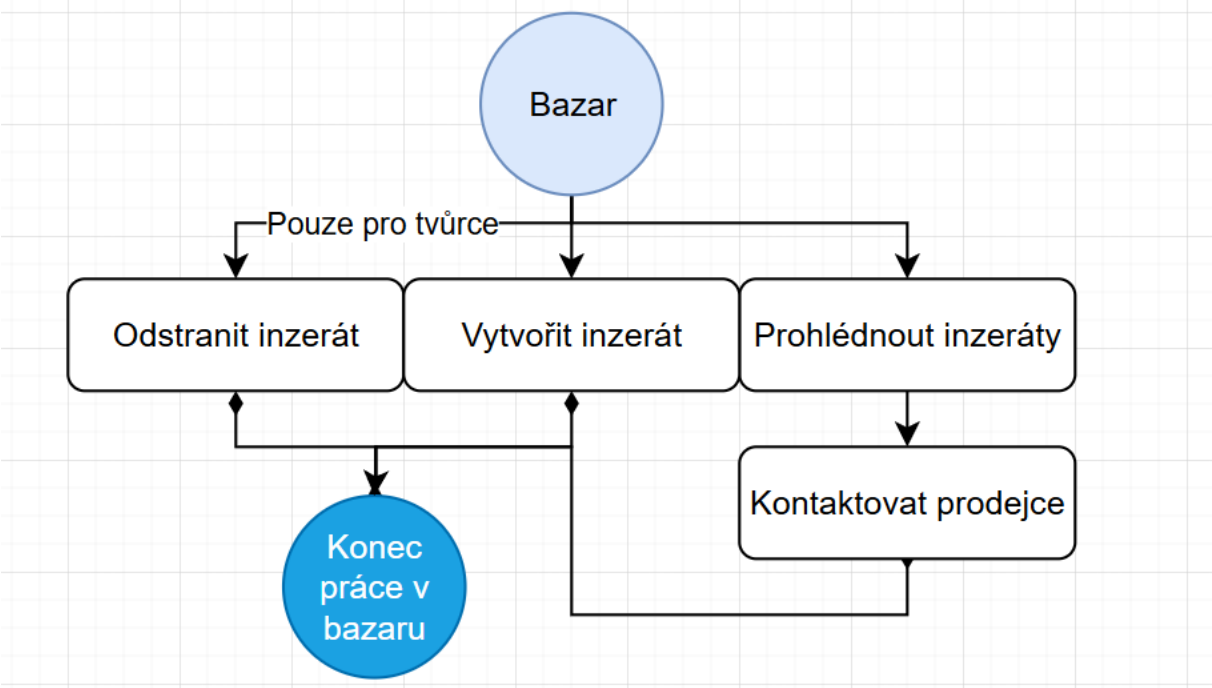
## 7.3 Diagram fóra




*Obrázek 9: Diagram fóra (vl. zdroj)*

V diagramu aktivit pro fórum jsou popsány možnosti, které má uživatel k dispozici v rámci diskusních vláken. Může si prohlédnout již existující fórum, vytvářet nová fóra nebo uzavírat stávající (typicky pouze s příslušnými oprávněními). Dále je zde popsána možnost přispívat do fóra, což zahrnuje vkládání příspěvků či komentářů k daným tématům. Výsledkem je ukončení práce ve fóru, ať už formou odhlášení se z diskuse, nebo přechodem k jiné funkci aplikace.

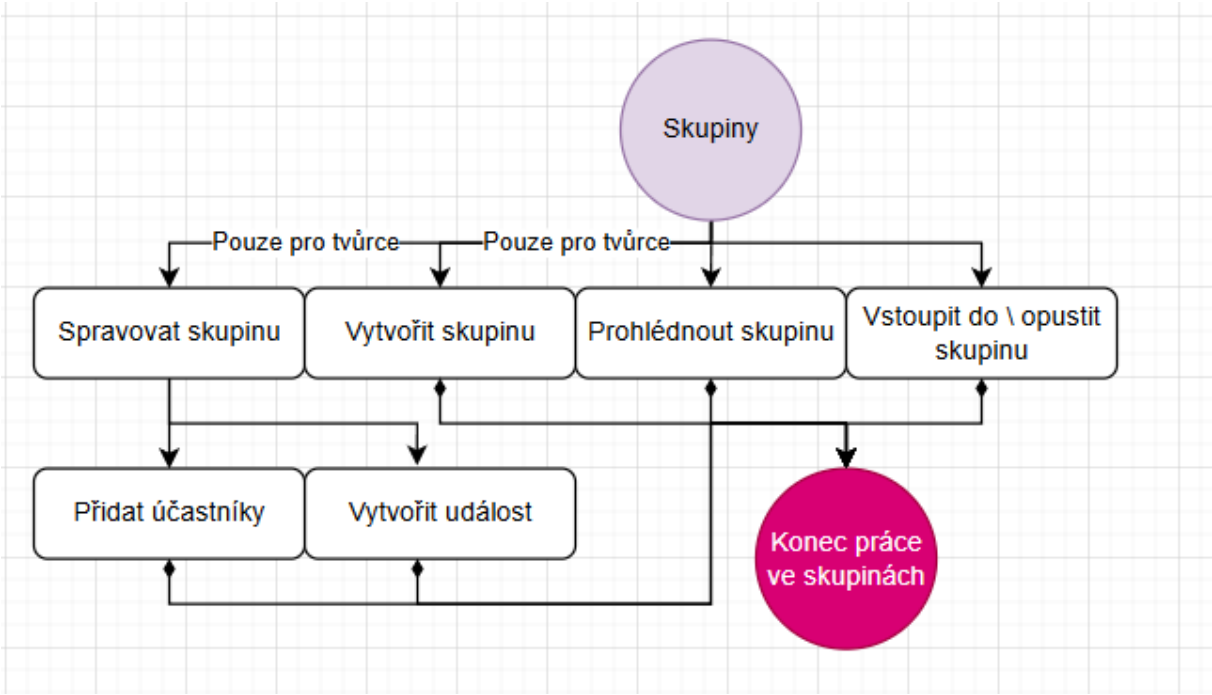
## 7.4 Diagram bazaru




*Obrázek 10: Diagram bazaru (vl. zdroj)*

Aktivity související s bazarem se zaměřují na práci s inzeráty. Uživatel s příslušnými oprávněními může inzeráty vytvářet nebo je naopak odstraňovat. Všichni uživatelé pak mají možnost prohlížet si aktuální inzeráty a kontaktovat prodávajícího, pokud je zaujme nabízené zboží či služba. Stejně jako u fóra je v závěru aktivity ukončení práce v bazaru, po němž se uživatel může vrátit k jiným sekcím aplikace nebo aplikaci opustit.

## 7.5 Diagram skupin



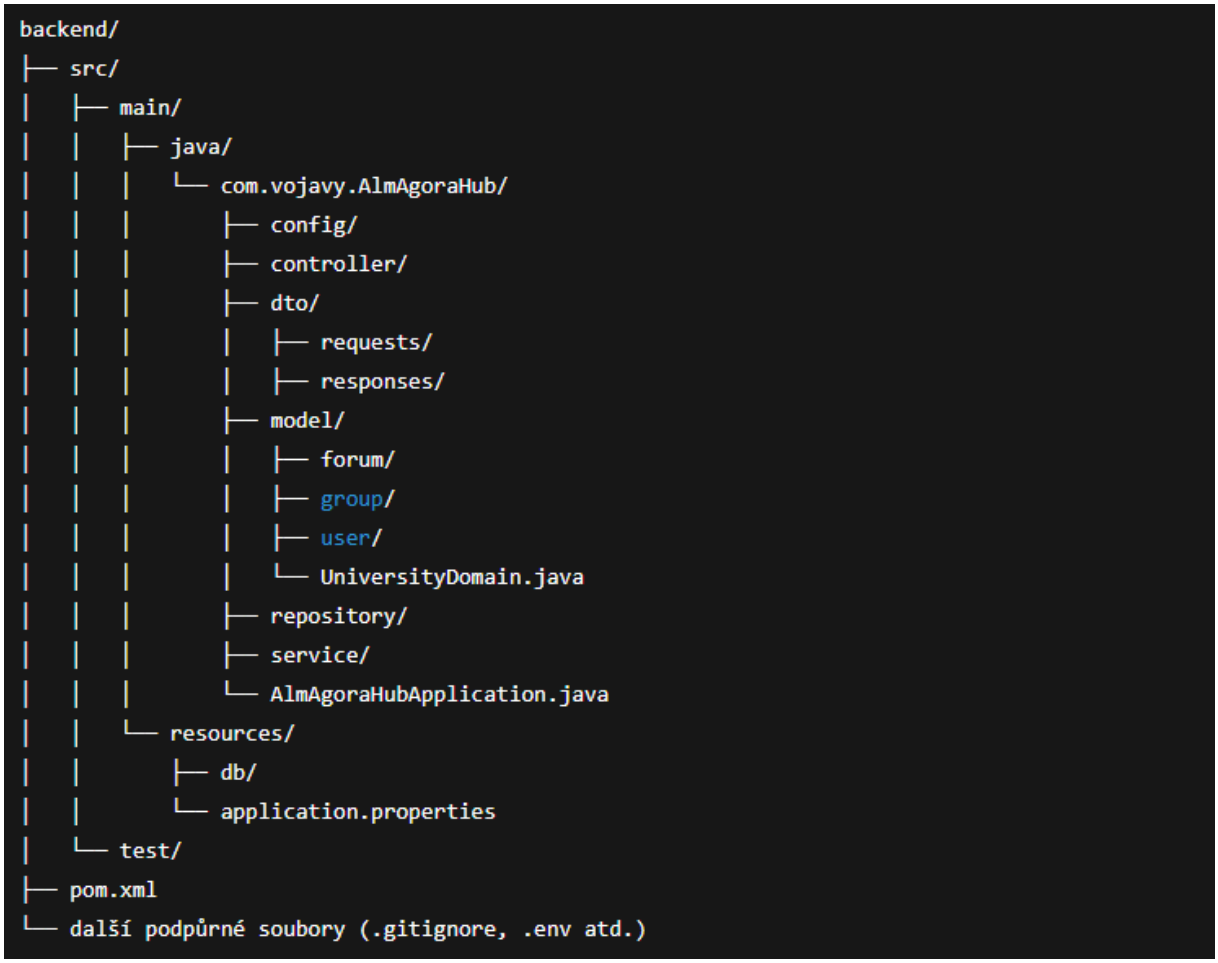

*Obrázek 11: Diagram skupin (vl. zdroj)*

Diagram aktivit pro skupiny zobrazuje proces správy a účasti ve skupinách. Uživatel může skupinu prohlédnout, vstoupit do ní či z ní odejít, vytvářet nové skupiny nebo v případě vyšších oprávnění skupinu spravovat (například přidávat další účastníky, plánovat události či měnit nastavení skupiny). Tato funkce je užitečná pro organizaci a komunikaci v rámci menších i větších komunit. Po ukončení práce ve skupinách se uživatel může přesunout do dalších sekcí aplikace nebo zcela ukončit svou činnost.

# 8 Java Spring Boot Backend

## 8.1 Struktura backendové části

Backendová část vyvíjena v jazyce Java 21 pomocí Spring Boot ve verzi 3.4.4. Projekt využívá monolitickou architekturu, vhodnou pro menší a středně velké projekty s důrazem na jednoduchost údržby a rychlý vývoj. Logická struktura projektu je rozdělena do modulárních balíčků podle standardních principů MVC a service-repository patternu, přičemž s důrazem na čitelnost, oddělení zodpovědností a rozšiřitelnost. Kromě klasického HTTP REST API tak platforma umožňuje i obousměrnou komunikaci mezi klientem a serverem přes STOMP protokol.




*Obrázek 12: Struktura backendu (vl. zdroj)*

***config/*** Obsahuje konfigurační třídy aplikace – nastavení zabezpečení, JWT filtry, inicializaci dat, emailové služby, WebSocket konfigurace, interceptor pro WebSocket komunikaci a konfiguraci STAG integrace.

***controller/*** Obsahuje REST a WebSocket kontrolery, které vystavují API endpointy pro správu autentizace, uživatelů, skupin, fóra, přátel, univerzitních domén, chatu a STAG integrace.

***dto/*** Obsahuje datové přenosové objekty (DTO) používané pro přenos dat mezi klientem a serverem. Struktura je rozdělena na požadavky (requests) a odpovědi (responses).

***model/*** Obsahuje JPA entity představující datové objekty aplikace – uživatele, skupiny, fóra a další.

***repository/*** Obsahuje rozhraní pro práci s databází, využívající Spring Data JPA pro snadnou manipulaci s entitami.

***service/*** Obsahuje implementace business logiky oddělenou od kontrolérů a modelů. Patří sem například autentizační služby, správa skupin a fóra.

***resources/*** Obsahuje konfigurační soubory aplikace, zejména databázové migrace a nastavení aplikace (například ***application.properties***).

## 8.2 Hlavní závislosti (pom.xml)

Projekt využívá sadu klíčových knihoven a nástrojů definovaných v souboru pom.xml, které zajišťují robustnost, bezpečnost, správu dat a efektivní komunikaci aplikace. Jednotlivé závislosti byly zvoleny s ohledem na požadovanou funkcionalitu a kompatibilitu se zvoleným technologickým stackem.

*(spring-boot-starter-web)*

Tento starter obsahuje základní komponenty pro vývoj webových aplikací založených na REST architektuře. Zahrnuje integrovaný server (embedded Tomcat), knihovny pro práci s HTTP požadavky a odpověďmi, a podporu JSON serializace a deserializace přes Jackson. Starter umožňuje rychlé vytvoření RESTful API bez nutnosti ruční konfigurace serveru.

*(spring-boot-starter-data-jpa)*
Starter pro snadnou integraci Java Persistence API (JPA) s podporou Spring Data. Umožňuje jednoduchou práci s databází prostřednictvím repozitářů, bez potřeby psaní složitých SQL dotazů. Tento přístup podporuje principy návrhového vzoru Repository a Unit of Work a výrazně zjednodušuje implementaci datové vrstvy.

*(spring-boot-starter-security)*
Zajišťuje implementaci bezpečnostních mechanismů v aplikaci, včetně autentizace, autorizace

a ochrany před běžnými zranitelnostmi (např. CSRF, CORS). V projektu je využíván zejména pro správu JWT autentizace a ochranu REST API endpointů.

*(spring-boot-starter-mail)*

Poskytuje podporu pro odesílání emailů přes SMTP protokol. Tento starter umožňuje implementaci funkcí jako je verifikace emailu při registraci, notifikace a další emailové služby integrované přímo do backendové aplikace.

*(spring-boot-starter-validation)*

Integruje specifikaci Bean Validation (Jakarta Validation) a umožňuje validaci vstupních dat v API požadavcích pomocí anotací, jako například @NotNull, @Email, @Size, čím zvyšuje spolehlivost a bezpečnost aplikace tím, že eliminuje nekorektní vstupy.

*(spring-boot-starter-websocket)*

Umožňuje implementaci WebSocket komunikace pro real-time přenos dat mezi klientem a serverem. Tento mechanismus je v projektu připraven pro budoucí rozšíření, například pro notifikace, chat nebo dynamické aktualizace bez potřeby obnovování stránky.

*io.jsonwebtoken (jjwt-api, jjwt-impl, jjwt-jackson)*

Knihovny z balíku io.jsonwebtoken slouží k bezpečné tvorbě, validaci a dekódování JWT tokenů. Používají se pro správu bezpečné komunikace mezi klientem a serverem bez nutnosti udržování serverové session. Díky nim je autentizace stateless a vhodná i pro škálovatelné distribuované systémy.

*(org.postgresql:postgresql)*

Oficiální JDBC driver pro připojení aplikace k databázi PostgreSQL. Tento ovladač umožňuje využívání všech funkcionalit PostgreSQL, jako jsou rozšířené typy, JSONB pole, transakční řízení a optimalizované dotazy.

*jakarta.servlet-api*

API pro podporu servletových operací, které umožňuje manipulaci s HTTP požadavky a odpověďmi. Tato knihovna je nezbytná pro správnou integraci s webovým serverem a poskytuje základní infrastrukturu pro práci s webovými požadavky (GET, POST atd.).

*(spring-boot-starter-websocket)*

Zajišťuje podporu pro WebSocket komunikaci nad protokolem STOMP. Umožňuje vytvářet kanály pro asynchronní přenos zpráv mezi serverem a klientem. Je základem pro implementaci

chatového systému a notifikačních funkcí. Ve spojení se SockJS a Spring Messaging poskytuje plnohodnotné prostředí pro real-time komunikaci.

## 8.3 Backendové architektonické principy

Backendová část projektu je navržena podle principů monolitické architektury, kdy všechny části systému běží v rámci jednoho procesu. Tento přístup byl zvolen s ohledem na potřeby projektu, zejména pro zajištění rychlého vývoje, jednoduchého nasazení a snadné údržby aplikace. Monolitická struktura je ideální volbou pro aplikaci střední velikosti, kde není potřeba extrémní horizontální škálovatelnosti ani složité orchestrace jednotlivých komponent.

Architektura je striktně vrstvená a jednotlivé vrstvy mají jasně definované odpovědnosti: Controller – třídy umístěné v balíčku *controller/* (např. ***UserController***, ***GroupController***) slouží jako vstupní body pro HTTP požadavky. Zpracovávají vstupy od uživatele, předávají je do servisní vrstvy a vrací odpovědi ve formě DTO objektů. Kontrolery jsou navrženy jako tenké vrstvy, které neobsahují žádnou business logiku. Service – třídy v balíčku *service/* (např. ***UserService***, ***GroupService***) obsahují hlavní business logiku aplikace. Servisní vrstva přijímá požadavky od kontrolerů, provádí aplikační operace, volá repozitáře a provádí validace nebo složitější transformační operace. Repository – rozhraní v balíčku *repository/* (např. ***UserRepository***, ***GroupRepository***) slouží k přímému přístupu do databáze pomocí Spring Data JPA. Repozitáře definují metody pro CRUD operace (vytvoření, načtení, aktualizaci a smazání dat). Model – třídy v balíčku *model/* (např. ***User***, ***Group***, ***ForumPost***) představují datové objekty odpovídající entitám uloženým v databázi. Tyto entity jsou mapovány na databázové tabulky pomocí JPA anotací (***@Entity***, ***@Table***, atd.). Takové oddělení vrstev zajišťuje vysokou míru modularity, lepší testovatelnost a čitelnost kódu, a umožňuje budoucí snadné rozšíření systému.

**Použití DTO (Data Transfer Object)**

Pro přenos dat mezi klientem a serverem jsou využívány DTO objekty, které jsou rozděleny do požadavků (*requests/*) a odpovědí (*responses/*), umístěné v balíčku *dto/*. Tento přístup zajišťuje ochranu interních datových struktur před přímým vystavením na veřejném API, možnost upravit nebo rozšířit strukturu odpovědí bez nutnosti měnit interní modely, zjednodušení validace vstupních dat (např. pomocí anotací @Valid v kontrolerech) a zvyšuje bezpečnost, flexibilitu a čistotu návrhu aplikace.

**JWT Autentizace**

Pro zabezpečení API aplikace je využíván autentizační mechanismus JWT (JSON Web Token). Při přihlášení uživatele je vygenerován JWT token, který je následně předáván v HTTP hlavičce ***Authorization*** při každém dalším požadavku na server. Backendová část implementuje vlastní JWT autentizační filtr (***JwtAuthFilter***), který ověřuje platnost tokenu, existence v databaze, extrahuje z něj informace o uživateli a nastavuje bezpečnostní kontext v rámci Spring Security. Tento přístup umožňuje bezstavovou autentizaci (*stateless authentication*). Server neukládá žádná uživatelská data mezi jednotlivými požadavky. Bezstavový model přispívá ke škálovatelnosti systému, protože umožňuje snadné paralelní nasazení více instancí backendové aplikace bez potřeby sdílení relací mezi servery. Autentizace je integrována pomocí *Spring Security* a spolupracuje s vlastními servisami ***AuthenticationService*** a ***UserService***, které načítají informace o uživateli z databáze.

**Informačni System Autentizace**

Připojení externích informačních systémů univerzit za účelem získávání dat je realizováno prostřednictvím samostatných kontrolerů, služeb, modelů a konfiguračních souborů. V případě integrace s IS/STAG probíhá připojení uživatelského účtu na straně frontendu, odkud je následně získán autentizační token. Tento token je předán na backend, kde je uložen do databáze a následně opakovaně využíván. Zvolený přístup umožňuje v budoucnu snadné rozšíření systému o integrace s dalšími informačními systémy jiných univerzit.

## 8.4 Základní struktura Spring Boot aplikace

Backendová část aplikace je postavena na technologii Spring Boot, která umožňuje rychlé sestavení robustních aplikací díky principu konvence nad konfigurací (Convention over Configuration). Tento přístup znamená, že pokud se drží zavedených konvencí (například struktury balíčků, pojmenování tříd a metod), systém automaticky zajistí správné nastavení komponent bez nutnosti ruční konfigurace. Výsledkem je výrazné snížení množství konfiguračního kódu, rychlejší vývoj a vyšší čitelnost projektu.

### 8.4.1 Vstupní bod aplikace

Vstupním bodem aplikace je třída ***AlmAgoraHubApplication***, která se nachází v balíčku *com.vojavy.AlmAgoraHub*:

```java
@SpringBootApplication  streblychenko
public class AlmAgoraHubApplication {

    public static void main(String[] args) {  streblychenko
        SpringApplication.run(AlmAgoraHubApplication.class, args);
    }

}
```

*Obrázek 13: AlmAgoraHubApplication,java vstupní bod Spring Boot (vl. zdroj)*

Tato třída obsahuje metodu *main*, která spouští celý Spring Boot kontejner a inicializuje veškeré kontexty aplikace. Anotace *@SpringBootApplication* sdružuje několik základních anotací *(@Configuration, @EnableAutoConfiguration, @ComponentScan*) a automaticky nakonfiguruje aplikaci na základě dostupných knihoven a tříd.

### 8.4.2 Konfigurační soubor application.properties

Chování aplikace je řízeno prostřednictvím souboru *application.properties*, který obsahuje základní nastavení databáze, autentizace, emailové služby, integrace se STAG systémem a další parametry

```
spring.application.name=AlmAgoraHub
# Připojení k databázi PostgreSQL
spring.datasource.url=${SPRING_DATASOURCE_URL}
spring.datasource.username=${SPRING_DATASOURCE_USERNAME}
spring.datasource.password=${SPRING_DATASOURCE_PASSWORD}
spring.datasource.driver-class-name=org.postgresql.Driver
# Nastavení JPA a Hibernate
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
spring.jpa.hibernate.ddl-auto=none
spring.jpa.show-sql=true
spring.flyway.enabled=true
spring.flyway.locations=classpath:db/migration
# Nastavení JWT
security.jwt.secret-key=${JWT_SECRET_KEY}
security.jwt.expiration-time=3600000
# Nastavení emailu
spring.mail.username=${VERIFICATION_EMAIL_USERNAME}
spring.mail.password=${VERIFICATION_EMAIL_PASSWORD}
# Docker Compose a externí konfigurační soubory
spring.docker.compose.enabled=false
spring.config.import=optional:file:.env[.properties]
# Adresy backendu a frontendu
hosting.address.backend=${BACKEND_URL}
hosting.address.frontend=${FRONTEND_URL}
# Integrace se STAG
stag.ws-base-url-first=https://stag-ws.
stag.ws-base-url-second=.cz/ws
```

*Obrázek 14: application.properties (vl: . zdroj)*

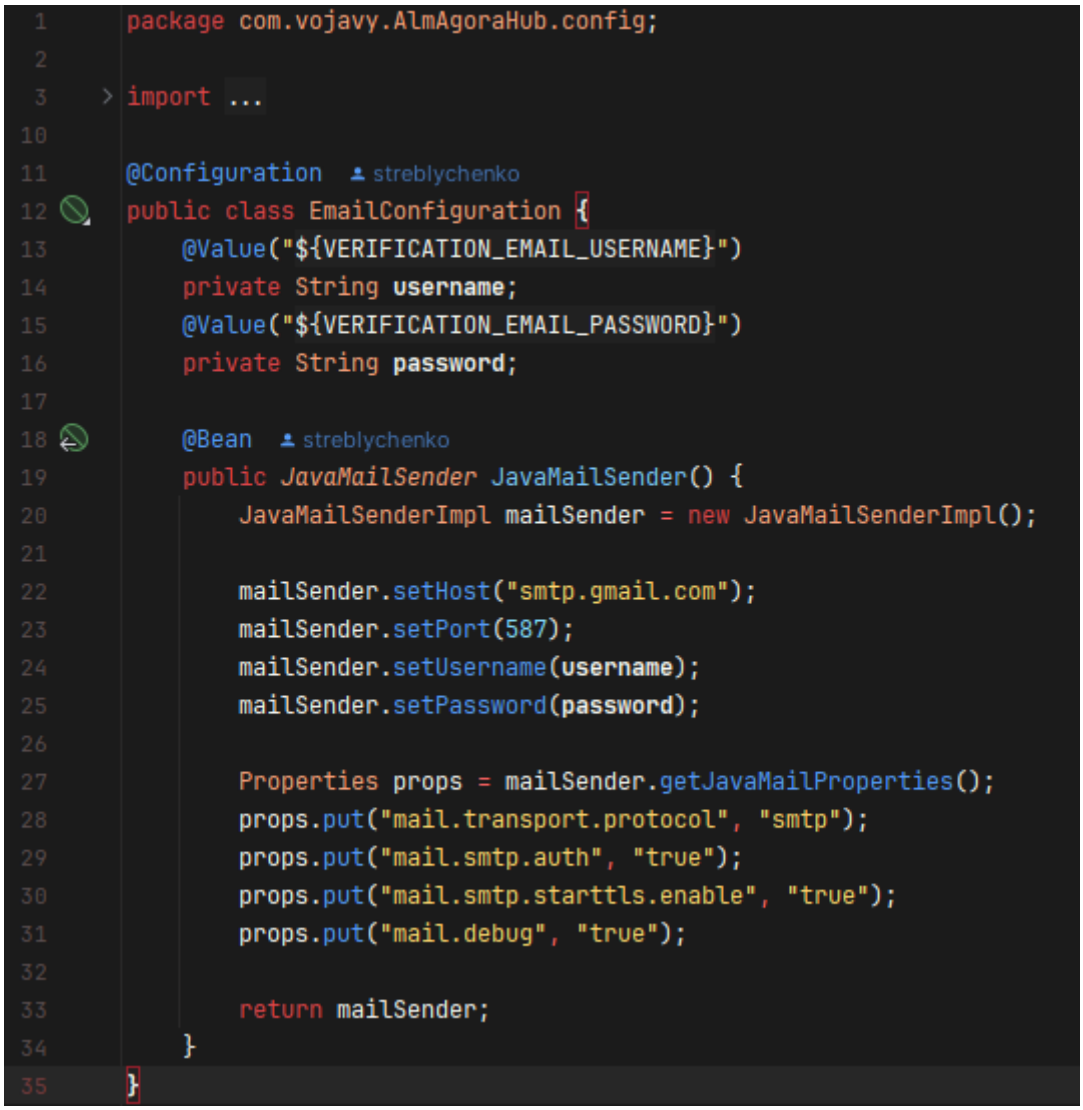
### 8.4.3 Konfigurační třídy aplikace

Aplikace obsahuje několik konfiguračních tříd, které definují chování různých částí systému

```java
package com.vojavy.AlmAgoraHub.config;

import ...

@Configuration  streblychenko
public class AppConfiguration {
    private final UserRepository userRepository;  2 usages

    public AppConfiguration(UserRepository userRepository) { this.userRepository = userRepository; }

    @Bean  streblychenko
    UserDetailsService userDetailsService() {
        return  String username → userRepository.findByEmail(username)
                .orElseThrow(() → new UsernameNotFoundException("User not found"));
    }

    @Bean  streblychenko
    BCryptPasswordEncoder bCryptPasswordEncoder() { return new BCryptPasswordEncoder(); }

    @Bean  streblychenko
    AuthenticationManager authenticationManager(AuthenticationConfiguration configuration) throws Exception {
        return configuration.getAuthenticationManager();
    }

    @Bean  streblychenko
    AuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();

        authProvider.setUserDetailsService(userDetailsService());
        authProvider.setPasswordEncoder(bCryptPasswordEncoder());

        return authProvider;
    }
}
```

*Obrázek 15: AppConfiguration.java (vl. zdroj)*

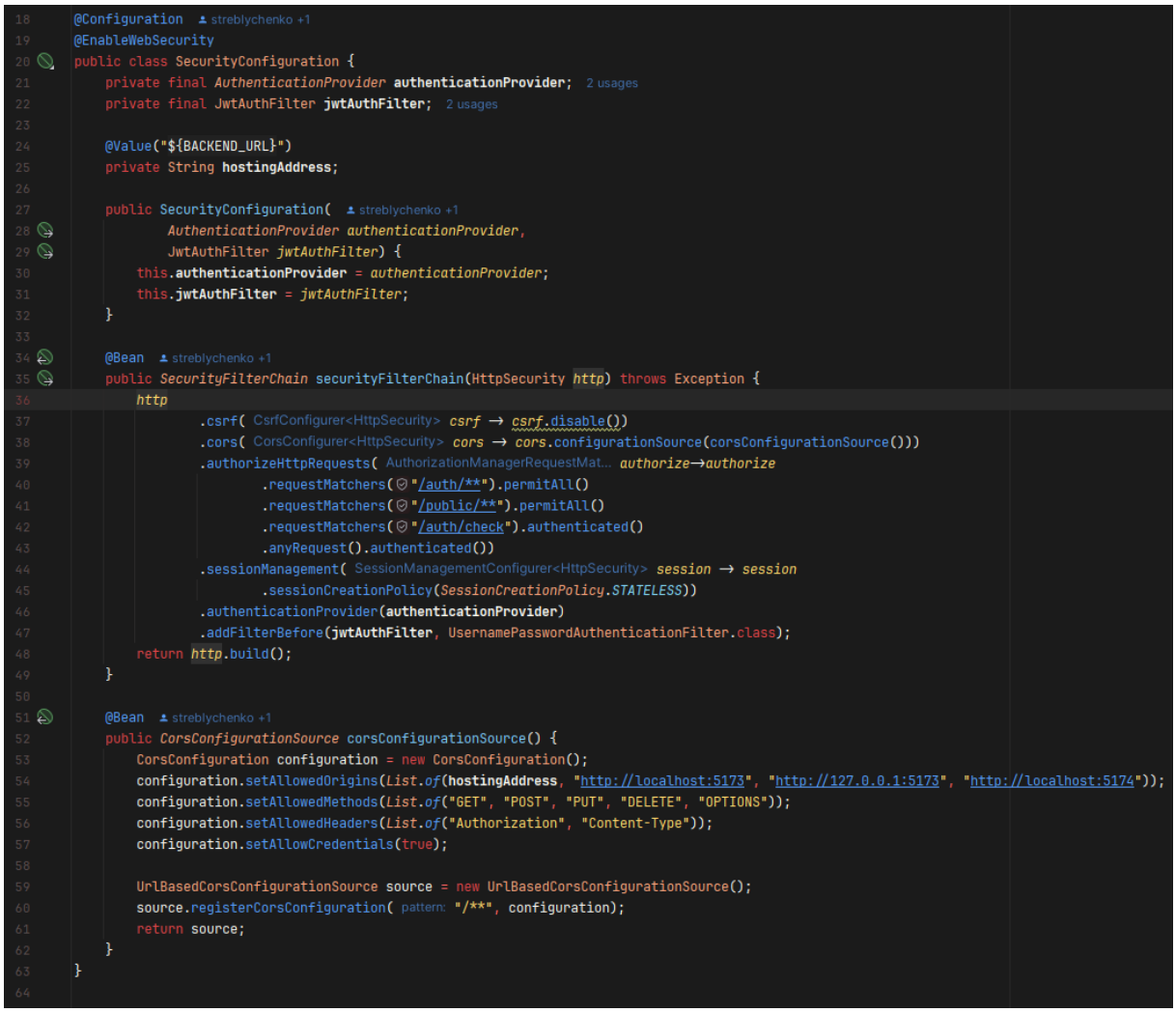
Třída ***AppConfiguration*** definuje základní bean komponenty pro zabezpečení aplikace. ***UserDetailsService*** – načítá uživatele z databáze podle emailu. ***BCryptPasswordEncoder*** – využíván pro bezpečné hashování hesel. ***AuthenticationManager*** – řídí autentizační procesy. ***AuthenticationProvider*** – poskytovatel autentizace založený na uživatelských detailech a šifrování hesla.



```java
package com.vojavy.AlmAgoraHub.config;

import ...

@Configuration  streblychenko
public class EmailConfiguration {
    @Value("${VERIFICATION_EMAIL_USERNAME}")
    private String username;
    @Value("${VERIFICATION_EMAIL_PASSWORD}")
    private String password;

    @Bean  streblychenko
    public JavaMailSender JavaMailSender() {
        JavaMailSenderImpl mailSender = new JavaMailSenderImpl();

        mailSender.setHost("smtp.gmail.com");
        mailSender.setPort(587);
        mailSender.setUsername(username);
        mailSender.setPassword(password);

        Properties props = mailSender.getJavaMailProperties();
        props.put("mail.transport.protocol", "smtp");
        props.put("mail.smtp.auth", "true");
        props.put("mail.smtp.starttls.enable", "true");
        props.put("mail.debug", "true");

        return mailSender;
    }
}
```

*Obrázek 16: EmailConfiguration.java (vl. zdroj)*

Třída ***EmailConfiguration*** definuje bean komponentu ***JavaMailSender*** pro odesílání emailů přes SMTP server Gmailu. Obsahuje nastavení hostitele, portu, autentizace a šifrované komunikace pomocí TLS



```java
@Configuration  streblychenko +1
@EnableWebSecurity
public class SecurityConfiguration {
    private final AuthenticationProvider authenticationProvider;  2 usages
    private final JwtAuthFilter jwtAuthFilter;  2 usages

    @Value("${BACKEND_URL}")
    private String hostingAddress;

    public SecurityConfiguration(  streblychenko +1
            AuthenticationProvider authenticationProvider,
            JwtAuthFilter jwtAuthFilter) {
        this.authenticationProvider = authenticationProvider;
        this.jwtAuthFilter = jwtAuthFilter;
    }

    @Bean  streblychenko +1
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
                .csrf( CsrfConfigurer<HttpSecurity> csrf → csrf.disable())
                .cors( CorsConfigurer<HttpSecurity> cors → cors.configurationSource(corsConfigurationSource()))
                .authorizeHttpRequests( AuthorizationManagerRequestMat... authorize→authorize
                        .requestMatchers("/auth/**").permitAll()
                        .requestMatchers("/public/**").permitAll()
                        .requestMatchers("/auth/check").authenticated()
                        .anyRequest().authenticated())
                .sessionManagement( SessionManagementConfigurer<HttpSecurity> session → session
                        .sessionCreationPolicy(SessionCreationPolicy.STATELESS))
                .authenticationProvider(authenticationProvider)
                .addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class);
        return http.build();
    }

    @Bean  streblychenko +1
    public CorsConfigurationSource corsConfigurationSource() {
        CorsConfiguration configuration = new CorsConfiguration();
        configuration.setAllowedOrigins(List.of(hostingAddress, "http://localhost:5173", "http://127.0.0.1:5173", "http://localhost:5174"));
        configuration.setAllowedMethods(List.of("GET", "POST", "PUT", "DELETE", "OPTIONS"));
        configuration.setAllowedHeaders(List.of("Authorization", "Content-Type"));
        configuration.setAllowCredentials(true);

        UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration( pattern: "/**", configuration);
        return source;
    }
}
```

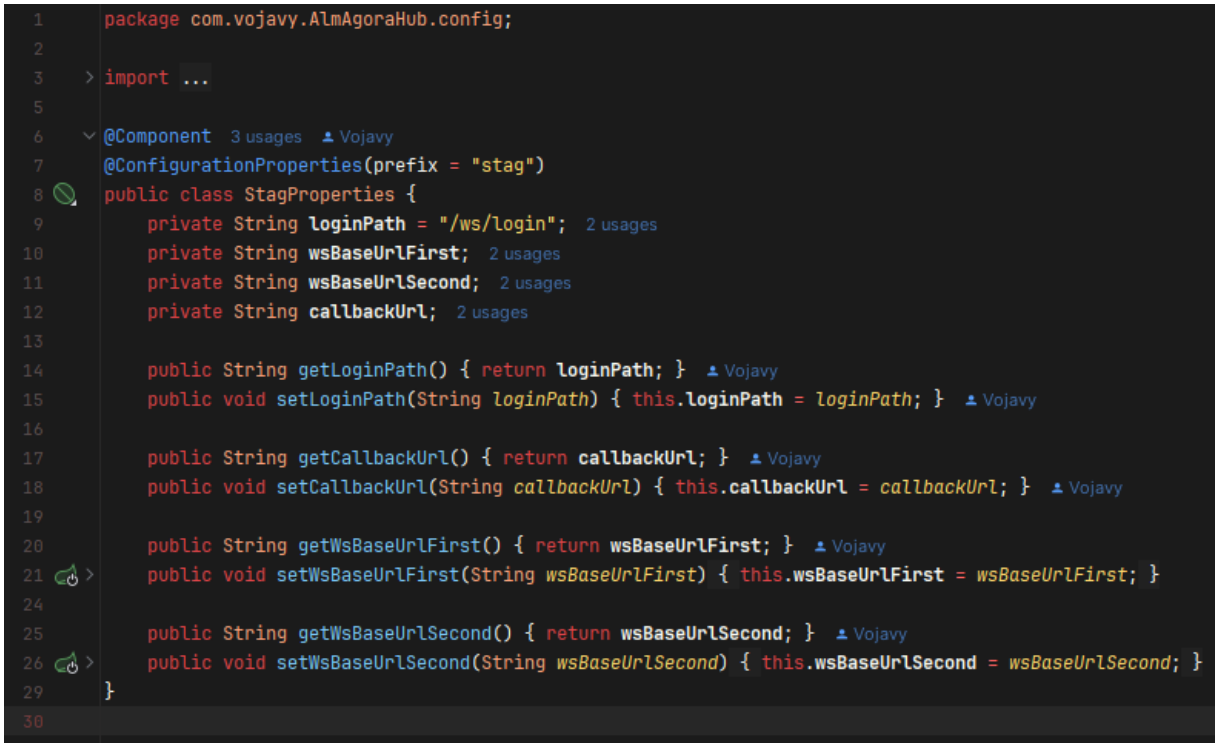
*Obrázek 17: SecurityConfiguration .java (vl. zdroj)*

Třída ***SecurityConfiguration*** nastavuje bezpečnostní politiku aplikace: Definuje pravidla autorizace pro jednotlivé endpointy (např. */auth/** a /public/**** veřejné, ostatní chráněné). Implementuje JWT autentizační filtr, který kontroluje platnost tokenů. Zajišťuje bezstavovou správu sezení (stateless session policy). Konfiguruje CORS pravidla pro prod a dev verze aplikace, aby umožnila bezpečné přístupy z frontendu.

```java
package com.vojavy.AlmAgoraHub.config;

import ...

@Configuration  Vojavy
public class StagConfiguration {

    @Bean  Vojavy
    public RestTemplate restTemplate(RestTemplateBuilder builder) {
        return builder.build();
    }
}
```

*Obrázek 18: StagConfiguration.java (vl. zdroj)*

Třída ***StagConfiguration*** definuje instanci **RestTemplate** – nástroje pro REST komunikaci s externími systémy (například IS/STAG API)



```java
package com.vojavy.AlmAgoraHub.config;

import ...

@Component  3 usages  Vojavy
@ConfigurationProperties(prefix = "stag")
public class StagProperties {
    private String loginPath = "/ws/login";  2 usages
    private String wsBaseUrlFirst;  2 usages
    private String wsBaseUrlSecond;  2 usages
    private String callbackUrl;  2 usages

    public String getLoginPath() { return loginPath; }  Vojavy
    public void setLoginPath(String loginPath) { this.loginPath = loginPath; }  Vojavy

    public String getCallbackUrl() { return callbackUrl; }  Vojavy
    public void setCallbackUrl(String callbackUrl) { this.callbackUrl = callbackUrl; }  Vojavy

    public String getWsBaseUrlFirst() { return wsBaseUrlFirst; }  Vojavy
    public void setWsBaseUrlFirst(String wsBaseUrlFirst) { this.wsBaseUrlFirst = wsBaseUrlFirst; }

    public String getWsBaseUrlSecond() { return wsBaseUrlSecond; }  Vojavy
    public void setWsBaseUrlSecond(String wsBaseUrlSecond) { this.wsBaseUrlSecond = wsBaseUrlSecond; }
}
```

*Obrázek 19: StagProperties .java (vl. zdroj)*

Třída ***StagProperties*** slouží pro načítání specifických parametrů integrace se STAG API ze souboru ***application.properties***: adresy STAG webových služeb, přihlašovací cesta, callback URL pro návrat po autentizaci

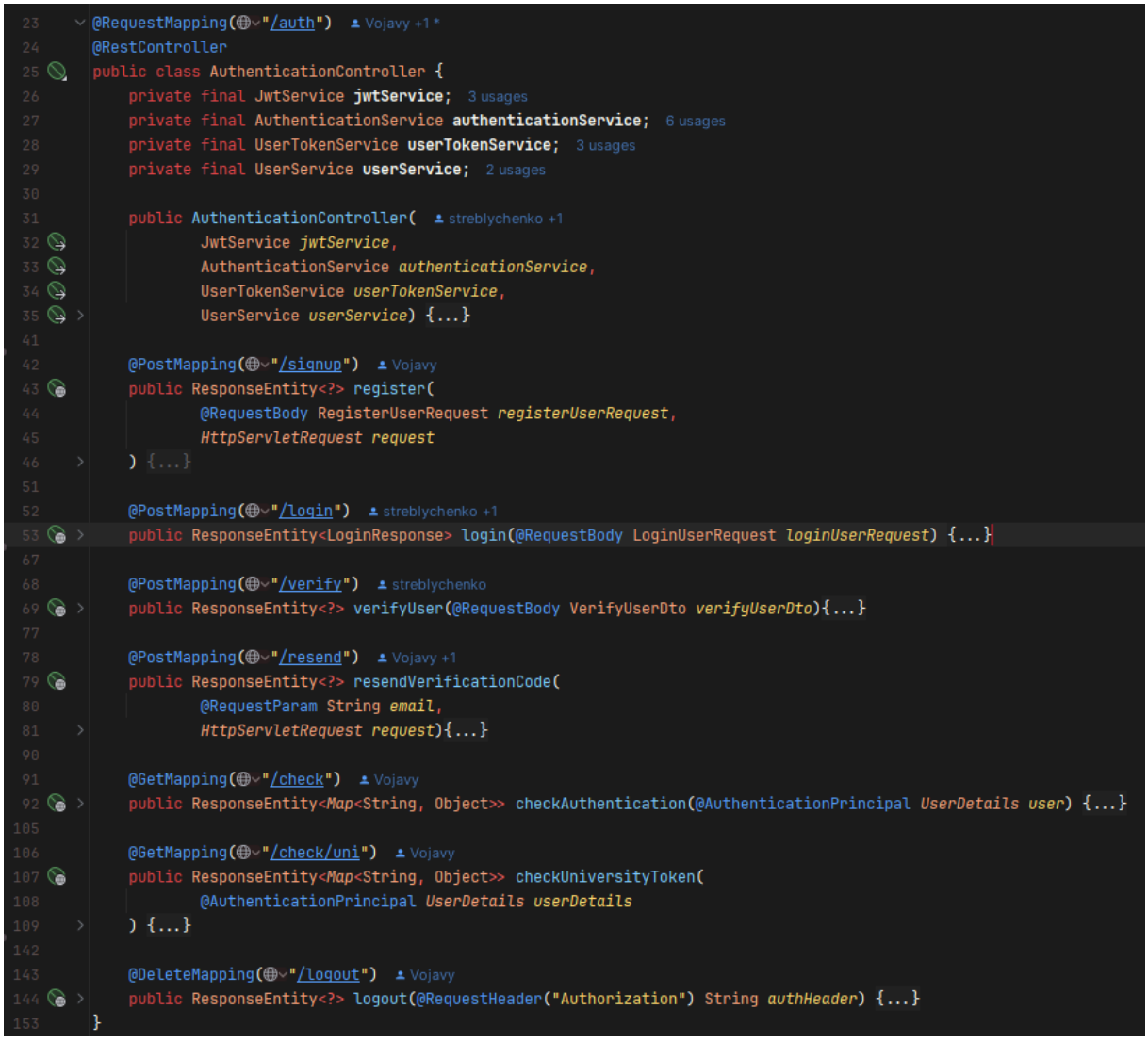
```java
@Component  Vojavy
public class DataInitializer implements CommandLineRunner {

    private final RoleRepository roleRepository;  3 usages
    private final UniversityDomainRepository domainRepository;  3 usages

    public DataInitializer(  Vojavy
            RoleRepository roleRepository,
            UniversityDomainRepository domainRepository
    ) {...}

    @Override  Vojavy
    public void run(String... args) {
        initializeRoles();
        initializeDomains();
    }

    private void initializeRoles() {  1 usage  Vojavy
        for (RoleType rt : RoleType.values()) {
            String name = rt.name();
            roleRepository.findByName(RoleType.fromString(name))
                    .orElseGet(() -> roleRepository.save(new Role(name)));
        }
    }

    private void initializeDomains() {  1 usage  Vojavy
        var defaults = List.of(
                new UniversityDomain(
                        domainName: "Univerzita Pardubice",
                        domain: "upce",
                        websiteUrl: "admin@upce.cz",
                        adminEmail: "https://portal.upce.cz/"
                ),
                new UniversityDomain(
                        domainName: "Západočeská univerzita v Plzni",
                        domain: "zcu",
                        websiteUrl: "admin@zcu.cz",
                        adminEmail: "https://portal.zcu.cz/"
                )
        );

        for (UniversityDomain ud : defaults) {
            domainRepository.findByDomain(ud.getDomain())
                    .orElseGet(() -> domainRepository.save(ud));
        }
    }
}
```

*Obrázek 20: DataInitializer.java (vl. zdroj)*

Třída ***DataInitializer*** je komponenta, která se spouští při startu aplikace (implements ***CommandLineRunner***) a inicializuje výchozí role uživatelů (podle ***RoleType*** enumu) a výchozí univerzitní domény (***UniversityDomain***), například Univerzita Pardubice a Západočeská univerzita v Plzni. Tím je zajištěno, že aplikace má při prvním spuštění základní funkční data.

## 8.5 Realizace JWT autorizace

Architektura JWT autentizace ve Spring Boot je v aplikaci navržena s cílem zajistit bezpečný, škálovatelný a bezstavový přístup k chráněným API zdrojům. Celý mechanismus je postaven na JSON Web Tokenech (JWT), které slouží jako digitální doklad o přihlášení uživatele. Tyto tokeny jsou generovány po úspěšné autentizaci a při každém požadavku jsou odesílány zpět na server v HTTP hlavičce. Server ověřuje platnost tokenu, identitu uživatele a případně jeho oprávnění. Tento přístup umožňuje odlehčení serveru od správy session a podporuje horizontální škálování systému.



```
@RequestMapping("/auth")  Vojavy +1 *
@RestController
public class AuthenticationController {
    private final JwtService jwtService;  3 usages
    private final AuthenticationService authenticationService;  6 usages
    private final UserTokenService userTokenService;  3 usages
    private final UserService userService;  2 usages

    public AuthenticationController(  streblychenko +1
            JwtService jwtService,
            AuthenticationService authenticationService,
            UserTokenService userTokenService,
            UserService userService) {...}

    @PostMapping("/signup")  Vojavy
    public ResponseEntity<?> register(
            @RequestBody RegisterUserRequest registerUserRequest,
            HttpServletRequest request
    ) {...}

    @PostMapping("/login")  streblychenko +1
    public ResponseEntity<LoginResponse> login(@RequestBody LoginUserRequest loginUserRequest) {...}

    @PostMapping("/verify")  streblychenko
    public ResponseEntity<?> verifyUser(@RequestBody VerifyUserDto verifyUserDto){...}

    @PostMapping("/resend")  Vojavy +1
    public ResponseEntity<?> resendVerificationCode(
            @RequestParam String email,
            HttpServletRequest request){...}

    @GetMapping("/check")  Vojavy
    public ResponseEntity<Map<String, Object>> checkAuthentication(@AuthenticationPrincipal UserDetails user) {...}

    @GetMapping("/check/uni")  Vojavy
    public ResponseEntity<Map<String, Object>> checkUniversityToken(
            @AuthenticationPrincipal UserDetails userDetails
    ) {...}

    @DeleteMapping("/logout")  Vojavy
    public ResponseEntity<?> logout(@RequestHeader("Authorization") String authHeader) {...}
}
```

*Obrázek 21: AuthenticationController.java (vl. zdroj)*

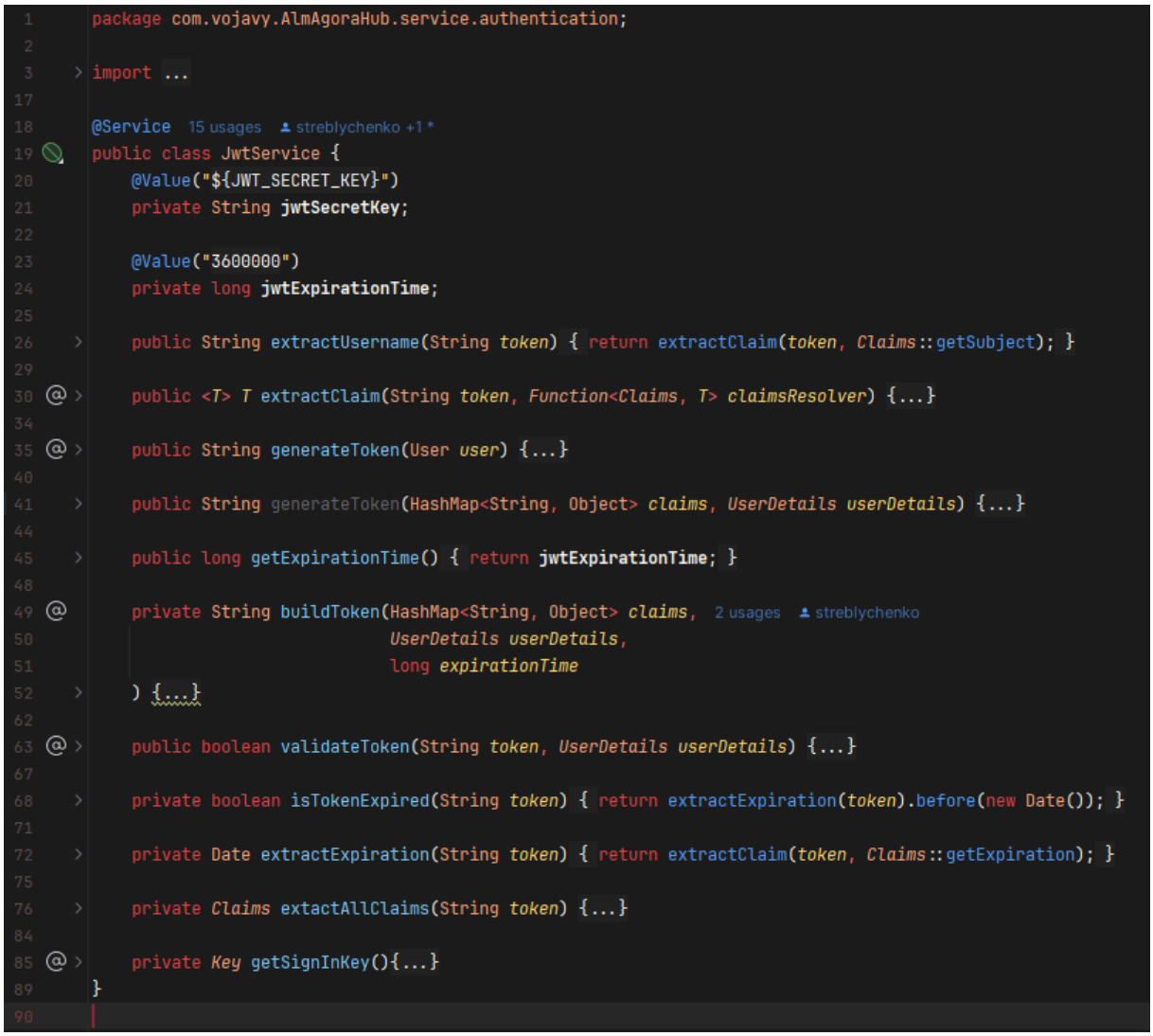
Třída ***AuthenticationController*** představuje hlavní vstupní bod pro autentizační požadavky. Zajišťuje registraci, přihlášení, ověření emailové adresy, kontrolu platnosti tokenu i odhlášení.

Každá z těchto operací komunikuje se službou ***AuthenticationService***, která obsahuje veškerou obchodní logiku související s autentizací.

```java
package com.vojavy.AlmAgoraHub.service.authentication;

import ...

@Service  3 usages  streblychenko +1 *
public class AuthenticationService {
    private final UserService userService;  7 usages
    private final PasswordEncoder passwordEncoder;  2 usages
    private final AuthenticationManager authenticationManager;  2 usages
    private final EmailService emailService;  2 usages
    private final UserTokenService userTokenService;  2 usages
    private final RoleService roleService;  3 usages

    public AuthenticationService(  Vojavy +1
            UserService userService,
            PasswordEncoder passwordEncoder,
            AuthenticationManager authenticationManager,
            EmailService emailService,
            UserTokenService userTokenService,
            RoleService roleService) {...}

    public User signupLocal(RegisterUserRequest input, String baseUrl) {...}

    public User authenticateLocal(LoginUserRequest input) {...}

    public void verifyUserLocal(VerifyUserDto input) {...}

    public void resendVerificationEmailLocal(String email, String baseUrl) {...}

    public void sendVerificationEmailLocal(User user, String baseUrl) {...}

    private String generateVerificationCode() {...}

    public void logoutByToken(String token) {...}
}
```

*Obrázek 22: AuthenticationService.java (vl. zdroj)*

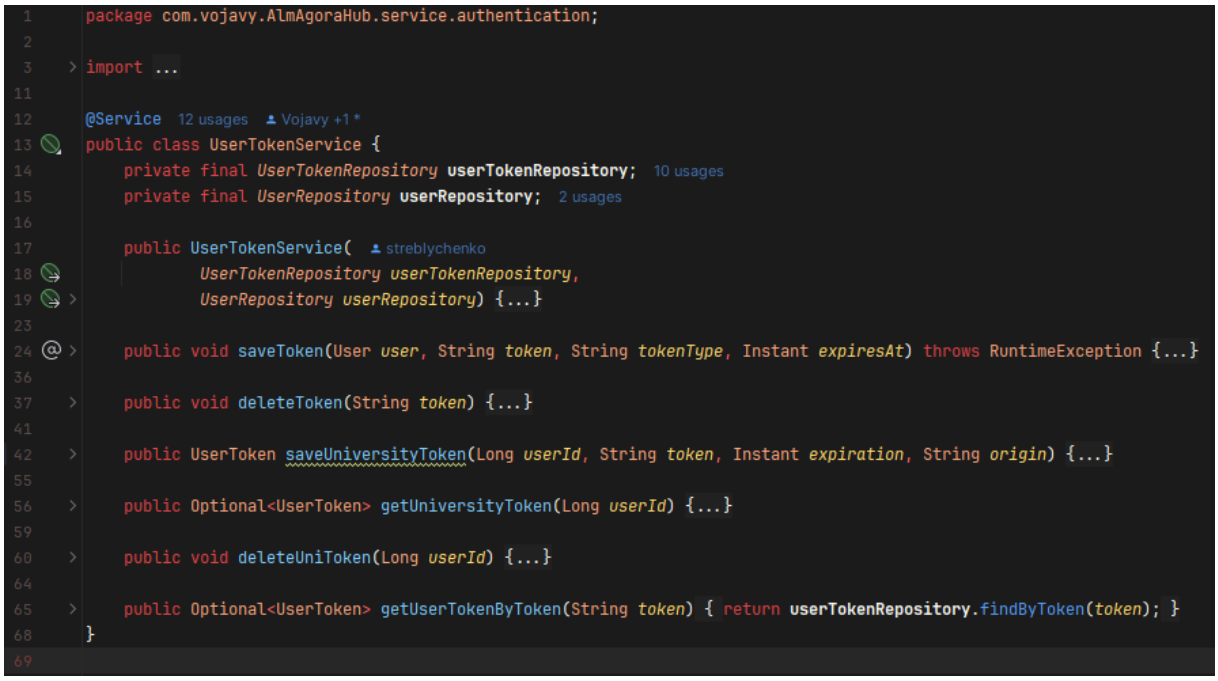
***AuthenticationService*** provádí kontrolu přihlašovacích údajů, inicializaci účtu a jeho uložení do databáze. V rámci registrace vygeneruje šestimístný verifikační kód, přiřadí uživateli výchozí roli „*ROLE_UNVERIFIED*“ a odešle potvrzovací email. Po úspěšném ověření účtu je uživateli přiřazena standardní role „*ROLE_USER*“. Tato služba také v případě přihlášení používá ***AuthenticationManager*** k ověření kombinace emailu a hesla a deleguje generování tokenu na ***JwtService***.



```java
package com.vojavy.AlmAgoraHub.service.authentication;

import ...

@Service  15 usages  streblychenko +1 *
public class JwtService {
    @Value("${JWT_SECRET_KEY}")
    private String jwtSecretKey;

    @Value("3600000")
    private long jwtExpirationTime;

    public String extractUsername(String token) { return extractClaim(token, Claims::getSubject); }

    public <T> T extractClaim(String token, Function<Claims, T> claimsResolver) {...}

    public String generateToken(User user) {...}

    public String generateToken(HashMap<String, Object> claims, UserDetails userDetails) {...}

    public long getExpirationTime() { return jwtExpirationTime; }

    private String buildToken(HashMap<String, Object> claims,  2 usages  streblychenko
                              UserDetails userDetails,
                              long expirationTime
    ) {...}

    public boolean validateToken(String token, UserDetails userDetails) {...}

    private boolean isTokenExpired(String token) { return extractExpiration(token).before(new Date()); }

    private Date extractExpiration(String token) { return extractClaim(token, Claims::getExpiration); }

    private Claims extactAllClaims(String token) {...}

    private Key getSignInKey(){...}
}
```

*Obrázek 23: JwtService.java (vl. zdroj)*

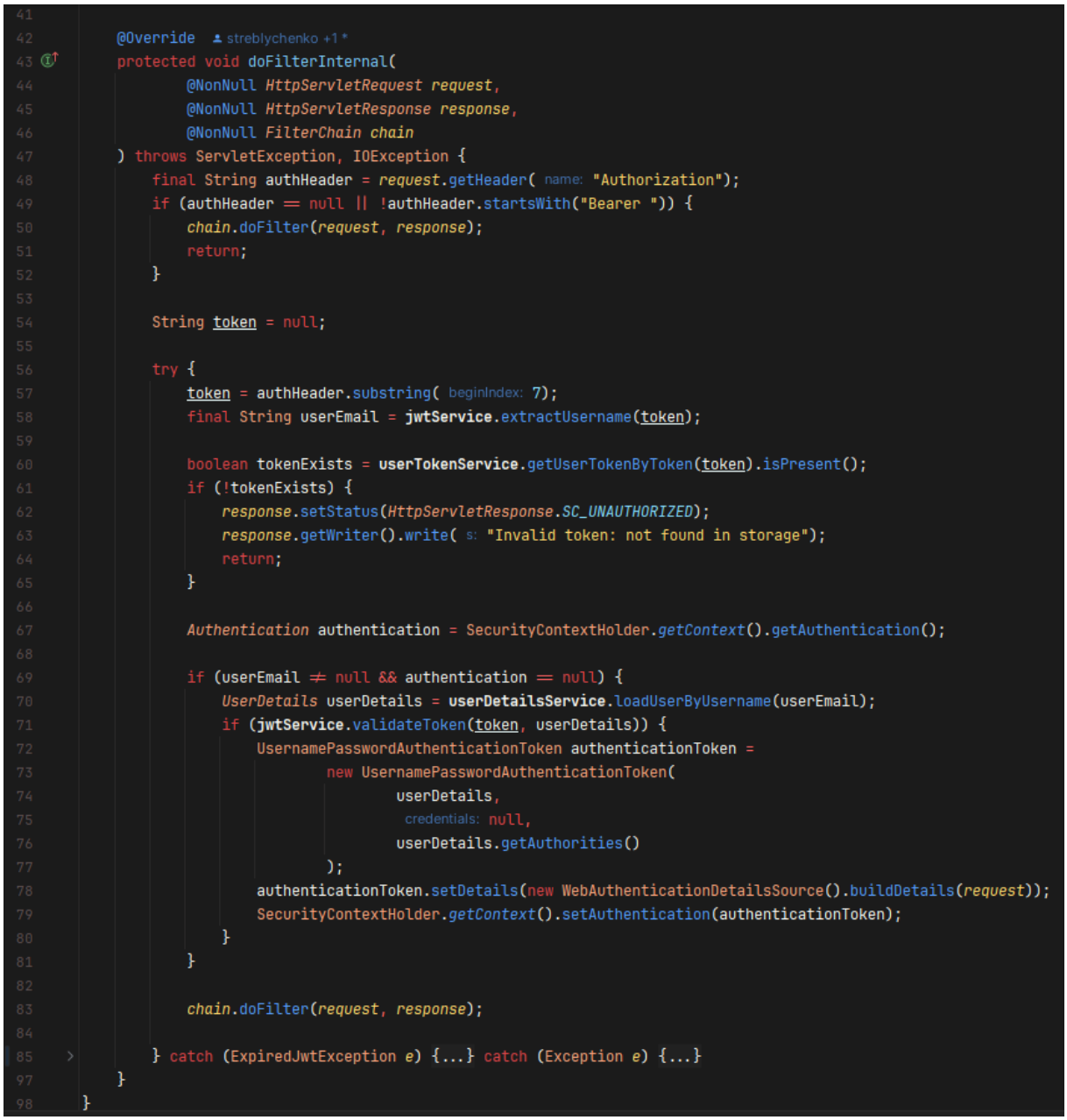
***JwtService*** je komponenta odpovědná za tvorbu, podepisování, extrahování informací z tokenu a jeho validaci. Při přihlášení služba vygeneruje token obsahující email uživatele jako subjekt a nastaví dobu platnosti podle konfigurace. Pro ověření tokenu používá uložený tajný klíč a kontroluje shodu s údaji uživatele.



```java
package com.vojavy.AlmAgoraHub.service.authentication;

import ...

@Service  12 usages  Vojavy +1 *
public class UserTokenService {
    private final UserTokenRepository userTokenRepository;  10 usages
    private final UserRepository userRepository;  2 usages

    public UserTokenService(  streblychenko
            UserTokenRepository userTokenRepository,
            UserRepository userRepository) {...}

    public void saveToken(User user, String token, String tokenType, Instant expiresAt) throws RuntimeException {...}

    public void deleteToken(String token) {...}

    public UserToken saveUniversityToken(Long userId, String token, Instant expiration, String origin) {...}

    public Optional<UserToken> getUniversityToken(Long userId) {...}

    public void deleteUniToken(Long userId) {...}

    public Optional<UserToken> getUserTokenByToken(String token) { return userTokenRepository.findByToken(token); }
}
```

*Obrázek 24: UserTokenService.java (vl. zdroj)*

***UserTokenService*** plní důležitou roli při správě tokenů na straně serveru. Každý token vydaný při přihlášení je uložen v databázi v entitě ***UserToken*** spolu s informací o typu tokenu (například „*auth*“ nebo „*uni*“), jeho platnosti a příslušnosti k uživateli. Tím je umožněna správa platnosti tokenů i z hlediska bezpečnosti — například v případě odhlášení nebo zneplatnění tokenu. Tato vrstva zajišťuje, že token je považován za platný pouze tehdy, pokud odpovídá záznamu v databázi.



```
41
42      @Override  streblychenko +1 *
43      protected void doFilterInternal(
44              @NonNull HttpServletRequest request,
45              @NonNull HttpServletResponse response,
46              @NonNull FilterChain chain
47      ) throws ServletException, IOException {
48          final String authHeader = request.getHeader( name: "Authorization");
49          if (authHeader == null || !authHeader.startsWith("Bearer ")) {
50              chain.doFilter(request, response);
51              return;
52          }
53
54          String token = null;
55
56          try {
57              token = authHeader.substring( beginIndex: 7);
58              final String userEmail = jwtService.extractUsername(token);
59
60              boolean tokenExists = userTokenService.getUserTokenByToken(token).isPresent();
61              if (!tokenExists) {
62                  response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
63                  response.getWriter().write( s: "Invalid token: not found in storage");
64                  return;
65              }
66
67              Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
68
69              if (userEmail != null && authentication == null) {
70                  UserDetails userDetails = userDetailsService.loadUserByUsername(userEmail);
71                  if (jwtService.validateToken(token, userDetails)) {
72                      UsernamePasswordAuthenticationToken authenticationToken =
73                              new UsernamePasswordAuthenticationToken(
74                                      userDetails,
75                                       credentials: null,
76                                      userDetails.getAuthorities()
77                              );
78                      authenticationToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
79                      SecurityContextHolder.getContext().setAuthentication(authenticationToken);
80                  }
81              }
82
83              chain.doFilter(request, response);
84
85          } catch (ExpiredJwtException e) {...} catch (Exception e) {...}
97      }
98  }
```

*Obrázek 25: JwtAuthFilter.java (vl. zdroj)*

***JwtAuthFilter*** je klíčovým bezpečnostním prvkem ve Spring Security. Tento filtr zachytává všechny příchozí HTTP požadavky a kontroluje přítomnost a validitu tokenu v hlavičce *Authorization*. Pokud hlavička není přítomna nebo nezačíná „*Bearer*“, požadavek je předán dále bez zásahu. Pokud však obsahuje token, filtr nejprve ověří jeho existenci v databázi pomocí ***UserTokenService***. Tato kontrola je zásadní pro zajištění, že token, který byl zneplatněn (například při odhlášení uživatele nebo zrušení přístupu u pozvaného účtu), nebude nadále akceptován — i když je technicky stále platný. Pokud token není nalezen v databázi, vrací se odpověď *401 Unauthorized* s chybovým hlášením. Pokud je token v pořádku a platný, dojde k jeho dekódování pomocí ***JwtService***, načtení uživatele přes ***UserDetailsService*** a nastavení

autentizačního kontextu do ***SecurityContextHolder***. Pokud dojde k chybě, jako je například vypršení platnosti tokenu (*ExpiredJwtException*), je tato skutečnost zaznamenána, token je z databáze odstraněn a požadavek je zamítnut.

Přihlášení uživatele probíhá tak, že uživatel odešle přihlašovací údaje ve formátu email + heslo na endpoint */auth/login*. ***AuthenticationService*** pomocí ***AuthenticationManager*** ověří správnost údajů a následně vygeneruje nový JWT token. Tento token je okamžitě uložen do databáze přes ***UserTokenService*** s přesně určenou dobou platnosti a odeslán klientovi spolu s dobou expirace. Klient jej následně používá k autorizovaným požadavkům tím, že jej přikládá do hlavičky *Authorization*. Při každém přístupu ke chráněným endpointům aplikace je token automaticky ověřován pomocí ***JwtAuthFilter***. Server tak může bezpečně určit identitu uživatele bez nutnosti správy session nebo ukládání stavových dat. To je výhodné zejména při škálování, protože jednotlivé instance backendu nemusí mezi sebou sdílet session storage. Díky JWT architektuře může být backend horizontálně škálován bez potřeby replikace session dat.

Součástí autentizační architektury je také e-mailová verifikace, která slouží k potvrzení identity uživatele při registraci. Uživatel obdrží email s kódem, který je následně ověřen v systému. Teprve po ověření je účet aktivní a získává plná oprávnění. Celý přístup je postaven na bezpečném uchovávání, ověřování a invalidaci tokenů, čímž je zajištěna vysoká úroveň kontroly a bezpečnosti aplikace. JWT architektura v kombinaci s Spring Security umožňuje efektivní řízení přístupu, bezproblémovou integraci s frontendem a připravenost na budoucí rozšíření o další autentizační mechanismy (např. *OAuth2* nebo univerzitní *Single Sign-On*).

## 8.6 Logika business vrstvy, kontrolerů, modelů, repozitářů a DTO

Backendová logika aplikace je navržena tak, aby podporovala čisté oddělení odpovědností, snadnou testovatelnost a čitelnou strukturu kódu. Architektura kombinuje klasický přístup vrstev MVC (model–repository–service–controller + externí Vue,js nebo jiny view) s několika návrhovými vzory a idiomy Spring Boot.

### Služby (Service layer)

```java
@Service  Vojavy +1 *
public class UserService {
    private final UserRepository userRepository;  10 usages
    private final JwtService jwtService;  2 usages

    public UserService(UserRepository userRepository, JwtService jwtService) {...}

    public List<User> findAll() {...}

    public Optional<User> findByEmail(String email) {  6 usages  Vojavy
        return userRepository.findByEmail(email);
    }

    public Optional<User> findById(Long id) {  Vojavy
        return userRepository.findById(id);
    }

    public User save(User user) {  Vojavy
        return userRepository.save(user);
    }

    public User update(User updatedUser) {  3 usages  Vojavy
        return userRepository.save(updatedUser);
    }

    public void updateUserWithPasswordCheck(  1 usage  Vojavy *
            Long userId,
            UpdateUserRequest request,
            BCryptPasswordEncoder encoder
    ) {...}

    public Long extractUserId(String tokenHeader){  36 usages  Vojavy
        if (tokenHeader == null) return null;
        String token = tokenHeader.replace( target: "Bearer ",  replacement: "");
        return jwtService.extractClaim(token,  Claims claims -> claims.get( claimName: "id", Long.class));
    }

    public void updateUserDetails(  1 usage  Vojavy *
            Long userId,
            UserDetailsExtended details
    ) {
        User user = userRepository.findById(userId).orElseThrow(() -> new IllegalArgumentException("User not found"));
        user.setDetails(details);
        userRepository.save(user);
    }
}
```

*Obrázek 26: služby. UserService.java (vl. zdroj)*

Služby (např. ***UserService***) obsahují hlavní obchodní logiku a slouží jako prostředník mezi kontrolery a datovou vrstvou. Ve výše uvedeném příkladu ***UserService*** zajišťuje operace jako vyhledání uživatele podle ID nebo emailu, aktualizaci údajů, validaci hesla nebo dekódování uživatelského ID z JWT tokenu. Tato vrstva využívá repozitáře (***UserRepository***) pro práci s

databází, ale zároveň poskytuje vyšší úroveň abstrakce: provádí logické kontroly (např. kontrolu aktuálního hesla) a řídí složitější aktualizace.

```java
package com.vojavy.AlmAgoraHub.service.forum;

import ...

@Service 2 usages  Vojavy *
@Transactional
class ForumPostService {

    private final ForumPostRepository postRepository; 4 usages

    protected ForumPostService(ForumPostRepository postRepository) { this.postRepository = postRepository; }

    protected ForumPost createPost( 1 usage  Vojavy
            Forum forum,
            com.vojavy.AlmAgoraHub.model.user.User author,
            String content,
            ForumPost parent
    ) {
        ForumPost p = new ForumPost();
        p.setForum(forum);
        p.setAuthor(author);
        p.setContent(content);
        p.setCreatedAt(Instant.now());
        p.setParentPost(parent);
        return postRepository.save(p);
    }

    protected ForumPost getPost(Forum forum, Integer postId) {  1 usage  Vojavy
        return postRepository.findById(postId)
                .filter( ForumPost p -> p.getForum().getId().equals(forum.getId()))
                .orElseThrow(() -> new IllegalArgumentException("Parent post not found"));
    }

    protected List<ForumPost> findAllByForum(Forum forum) {  1 usage  Vojavy *
        return postRepository.findByForumOrderByCreatedAtAsc(forum);
    }
}

```

*Obrázek 27: protected služby. ForumPostService.java (vl. zdroj)*

Některé služby v systému jsou navrženy jako public-facade vrstvy nad interními komponentami. Například ***ForumPostService*** může být *package-private*, ale přístup k jeho funkcím je poskytován přes veřejný ***ForumService***. Tento vzor usnadňuje zapouzdření nízkoúrovňové logiky a zachovává čistotu API.

**Modely (JPA entity)**

```java
public class User implements org.springframework.security.core.userdetails.UserDetails {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)  2 usages
    private String authProvider;

    @Column(name = "provider_id", nullable = false)  2 usages
    private Integer providerId;

    @Column(unique = true)  3 usages
    private String email;

    @Column(length = 512)  2 usages
    private String password;

    @Column(nullable = false)  3 usages
    private boolean active;

    @Column(name = "created_at", nullable = false)  3 usages
    private Instant createdAt;

    @Column(name = "updated_at", nullable = false)  3 usages
    private Instant updatedAt;

    @Column(name = "verification_code", length = 6)  2 usages
    private String verificationCode;

    @Column(name = "verification_expires")  2 usages
    private Instant verificationExpires;

    @Convert(converter = UserDetailsExtended.ConverterImpl.class)  2 usages
    @JdbcTypeCode(SqlTypes.JSON)
    @Column(columnDefinition = "jsonb")
    private UserDetailsExtended details;

    @ManyToOne  2 usages
    @JoinColumn(name = "domain_id")
    private UniversityDomain domain;

    @ManyToMany(fetch = FetchType.EAGER)  3 usages
    @JoinTable(
            name = "user_roles",
            joinColumns = @JoinColumn(name = "user_id"),
            inverseJoinColumns = @JoinColumn(name = "role_id")
    )
    private Set<Role> roles;
```

*Obrázek 28: modely. User.java (vl. zdroj)*

Modelové třídy (např. ***User***) reprezentují databázové entity. Jsou anotované pomocí JPA anotací jako *@Entity, @Table, @Id, @Column*, a definují strukturu odpovídající tabulkám v

relační databázi. Jednotlivé atributy jsou typicky mapovány na sloupce, přičemž některé jsou specifické — např. pole *details* ve třídě ***User*** je typu ***UserDetailsExtended***, které je serializováno do databázového typu *jsonb* pomocí vlastní implementace konvertoru (*@Convert + @JdbcTypeCode(SqlTypes.JSON)*). Tento přístup umožňuje snadné ukládání složených datových struktur jako JSON, což je výhodné pro dynamická pole nebo flexibilní rozšiřování bez nutnosti měnit databázové schéma.

**DTO (Data Transfer Objects)**

```java
package com.vojavy.AlmAgoraHub.dto.requests;

public class UpdateUserRequest {  4 usages  Vojavy
    private String email;  2 usages
    private String oldPassword;  2 usages
    private String newPassword;  2 usages
    private boolean active;  2 usages

    public String getEmail() { return email; }

    public void setEmail(String email) { this.email = email; }

    public String getOldPassword() { return oldPassword; }

    public void setOldPassword(String oldPassword) { this.oldPassword = oldPassword; }

    public String getNewPassword() { return newPassword; }

    public void setNewPassword(String newPassword) { this.newPassword = newPassword; }

    public boolean isActive() { return active; }

    public void setActive(boolean active) { this.active = active; }
}
```

*Obrázek 29: DTO. UpdateUserRequest.java (vl. zdroj)*

DTO objekty slouží k oddělení veřejného API (např. vstupní data z frontendu nebo výstupní odpovědi) od interních modelů databáze. Například ***UpdateUserRequest*** přijímá změnu e-mailu a hesla, ***VerifyUserDto*** slouží pro ověření účtu. DTO umožňují: validaci dat pomocí anotací *(@NotNull, @Email,* atd.), udržení konzistence API bez odhalení struktury interních entit, pohodlnou transformaci dat mezi modelem a API.

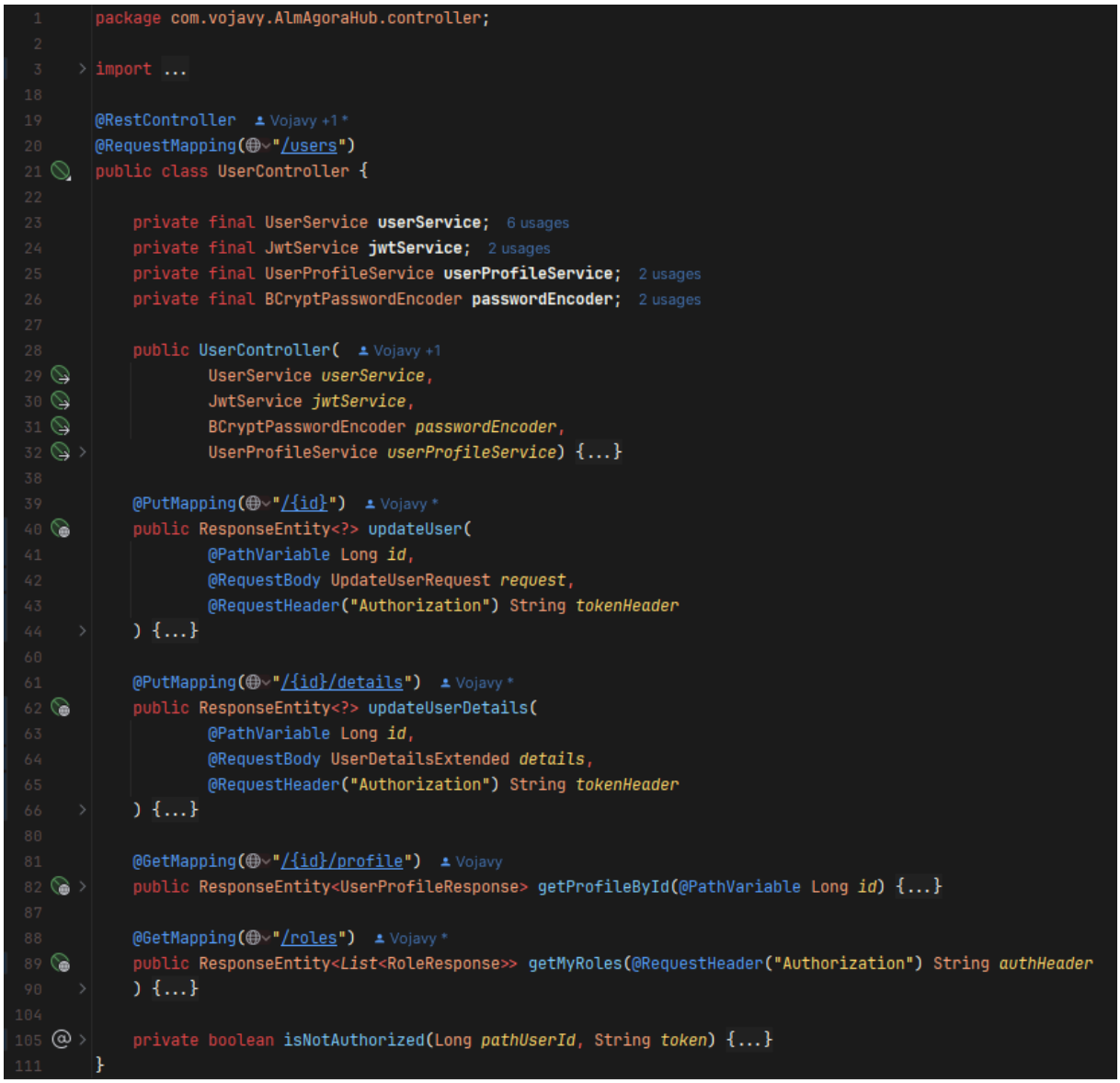
DTO jsou obvykle použity jako vstupy kontrolerů *(@RequestBody*) nebo jako návratové hodnoty (*ResponseEntity<UserProfileResponse>*).

### Repozitáře (Repository layer)


*Obrázek 30: repozitáře. UserRepository.java (vl. zdroj)*

Repozitáře jsou rozhraní (např. ***UserRepository***) rozšiřující ***JpaRepository***, poskytující automatickou implementaci běžných CRUD operací (*findById, save, delete*, atd.). Jako doplněk lze definovat i vlastní metody podle názvu (*findByEmail*). Repozitáře slouží jako primární nástroj pro interakci s databází, bez potřeby psaní SQL dotazů.

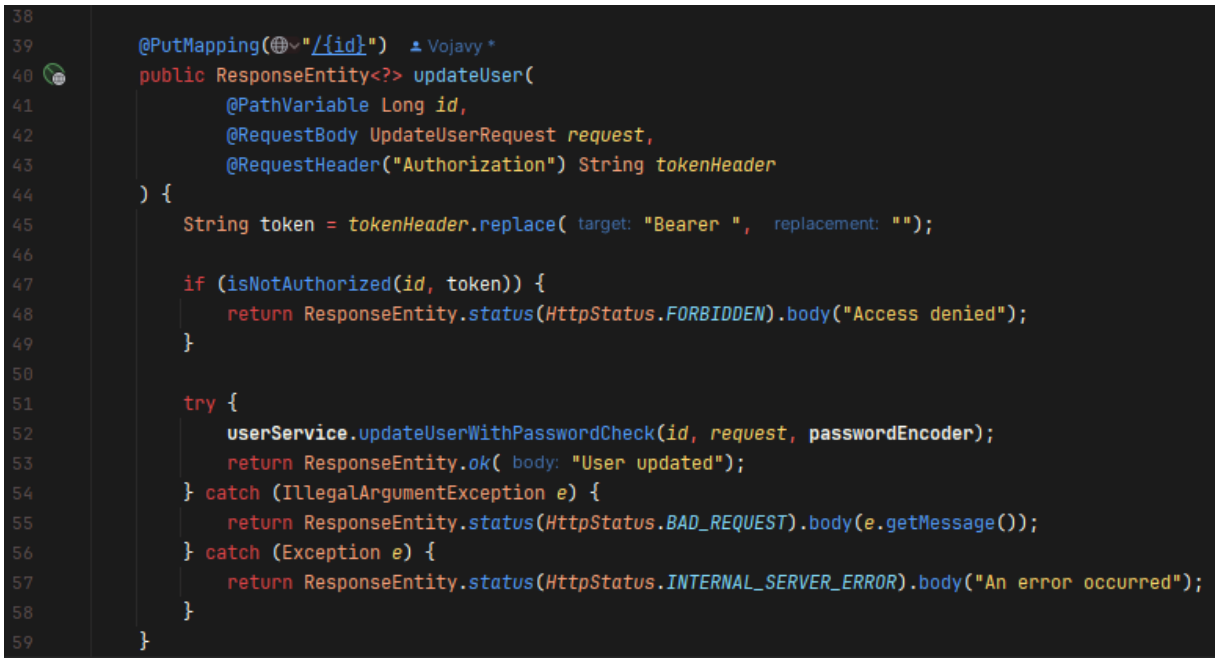
### Kontrolery (Controller layer)



```
package com.vojavy.AlmAgoraHub.controller;

import ...

@RestController  Vojavy +1 *
@RequestMapping("/users")
public class UserController {

    private final UserService userService;  6 usages
    private final JwtService jwtService;  2 usages
    private final UserProfileService userProfileService;  2 usages
    private final BCryptPasswordEncoder passwordEncoder;  2 usages

    public UserController(  Vojavy +1
            UserService userService,
            JwtService jwtService,
            BCryptPasswordEncoder passwordEncoder,
            UserProfileService userProfileService) {...}

    @PutMapping("/{id}")  Vojavy *
    public ResponseEntity<?> updateUser(
            @PathVariable Long id,
            @RequestBody UpdateUserRequest request,
            @RequestHeader("Authorization") String tokenHeader
    ) {...}

    @PutMapping("/{id}/details")  Vojavy *
    public ResponseEntity<?> updateUserDetails(
            @PathVariable Long id,
            @RequestBody UserDetailsExtended details,
            @RequestHeader("Authorization") String tokenHeader
    ) {...}

    @GetMapping("/{id}/profile")  Vojavy
    public ResponseEntity<UserProfileResponse> getProfileById(@PathVariable Long id) {...}

    @GetMapping("/roles")  Vojavy *
    public ResponseEntity<List<RoleResponse>> getMyRoles(@RequestHeader("Authorization") String authHeader
    ) {...}

    private boolean isNotAuthorized(Long pathUserId, String token) {...}
}
```

*Obrázek 31: kontrolery. UserController.java (vl. zdroj)*

Kontrolery (např. ***UserController***) slouží jako vstupní body API. Jsou označené anotacemi *@RestController a @RequestMapping*, které definují HTTP cesty a metody (*@GetMapping, @PutMapping, @PostMapping*, atd.). Pomocí anotace @RequestBody získávají vstupní data ve formátu JSON, *@PathVariable* umožňuje dynamické URL a *@RequestHeader* slouží pro získání autorizační hlavičky.



```
38
39      @PutMapping("/{id}")  Vojavy *
40      public ResponseEntity<?> updateUser(
41              @PathVariable Long id,
42              @RequestBody UpdateUserRequest request,
43              @RequestHeader("Authorization") String tokenHeader
44      ) {
45          String token = tokenHeader.replace( target: "Bearer ",  replacement: "");
46
47          if (isNotAuthorized(id, token)) {
48              return ResponseEntity.status(HttpStatus.FORBIDDEN).body("Access denied");
49          }
50
51          try {
52              userService.updateUserWithPasswordCheck(id, request, passwordEncoder);
53              return ResponseEntity.ok( body: "User updated");
54          } catch (IllegalArgumentException e) {
55              return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(e.getMessage());
56          } catch (Exception e) {
57              return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("An error occurred");
58          }
59      }
```

*Obrázek 32: kontrolery. UserController.java -> updateUser metoda (vl. zdroj)*

Kontrolery neobsahují business logiku — jejich úkolem je validace přístupových práv, přesměrování požadavků na příslušné služby a návrat odpovědi ve správném HTTP formátu. Například metoda ***updateUser*** ověřuje, zda se uživatel pokouší upravit svůj vlastní profil, a poté deleguje zpracování na ***userService.updateUserWithPasswordCheck***.

Pro zabezpečení přístupu je použit tokenový model: JWT tokeny jsou zasílány v hlavičce *Authorization* a validovány pomocí služeb. Kontroler například porovnává ID z tokenu s ID z cesty nebo kontroluje, zda má uživatel roli administrátora (***isNorAuthorized***). Tím je zajištěno, že pouze oprávněný uživatel může upravovat svůj profil.

## 8.7 Realtime komunikace: WebSocket, chat a notifikace

Platforma zahrnuje i podporu komunikace v reálném čase prostřednictvím WebSocket připojení. Tento mechanismus je zásadní pro zajištění okamžité zpětné vazby v modulech jako je chat a interní zprávy mezi uživateli.

WebSocket vrstva je implementována pomocí technologie Spring WebSocket a protokolu **STOMP**. Konfigurace samotné WebSocket infrastruktury je součástí třídy ***WebSocketConfig***, která aktivuje podporu **STOMP** brokeru, nastavuje příchozí kanály a definuje prefixy jako */app* pro odesílání zpráv a */queue, /user* pro jejich příjem. Příkladem použití je adresování zpráv na */user/{userId}/queue/messages*, které zajišťuje doručení pouze konkrétnímu uživateli.

Zabezpečení spojení je řešeno pomocí komponenty ***JwtChannelInterceptor***, která ověřuje JWT token obsažený v hlavičce každého **STOMP** připojení (*Authorization: Bearer ...*). Tento interceptor načte identitu uživatele ze zaslaného tokenu, ověří jeho platnost přes JwtService a zkontroluje existenci tokenu v databázi přes UserTokenService. Díky tomu je možné bezpečně omezit přístup k realtime kanálům pouze na oprávněné uživatele.

Ověřená identita je následně nastavena do Spring Security kontextu a zároveň do objektu ***StompHeaderAccessor***, čímž se zpřístupní jako ***Principal***. Tento přístup umožňuje adresovat uživatelské zprávy konkrétní identitě bez nutnosti další autentizace během samotné WebSocket relace.

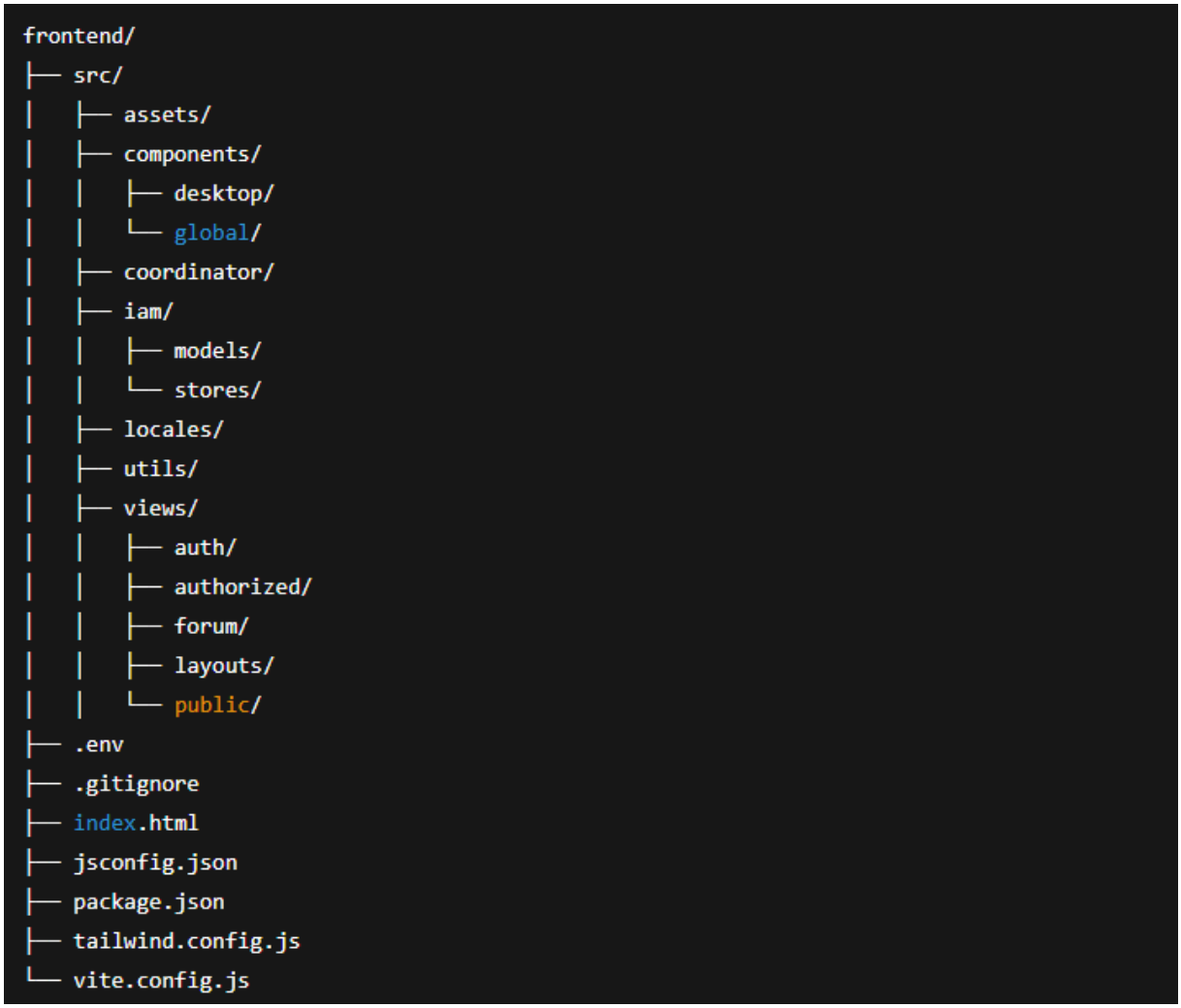
Komunikační architektura je navržena s podporou rozšiřitelnosti – například v budoucnu lze jednoduše přidat realtime upozornění na události ve skupinách, sledování online statusu uživatelů, push oznámení o nových zprávách, komentářích, žádostech atp.

# 9. Vue.js Frontend

## 9.1 Struktura frontendové části

Frontendová část aplikace postavena pomocí JavaScriptového frameworku **Vue 3** s využitím nástrojů *Vite*, *Tailwind CSS*, *Pinia*, *Vue Router* a *i18n* pro lokalizaci. Aplikace implementuje architekturu **MVI** (Model-View-Intent) v kombinaci s vlastním **Coordinator** mechanismem, který řídí navigaci mezi pohledy.

Frontend rozdělen do modulárních složek podle jednotlivých zodpovědností – komponenty rozhraní, správa stavů, modely dat, pohledy, nástroje a další pomocné prvky. Projekt zároveň podporuje real-time komunikaci se serverem pomocí protokolu WebSocket (**STOMP** + **SockJS**) pro chat.




*Obrázek 33: Struktura frontendu (vl. zdroj)*

*assets/* obsahuje statické zdroje používané napříč celou aplikací –SVG ikony a globální stylový soubor ***main.css*** obsahující základní Tailwind konfiguraci. Složka slouží k centralizované správě všech vizuálních podkladů aplikace.

*components/* obsahuje UI komponenty rozdělené tematicky podle kontextu použití – např. *desktop/* pro komponenty navigace na počítači nebo *global/* pro modální okna a univerzální rozhraní.

*coordinator/* vlastní implementace koordinační vrstvy, která řídí tok aplikace (navigaci, přesměrování, zobrazení chybových stavů apod.). Složka obsahuje ***coordinator.js*** jako hlavní správce chování a router.js s definicí všech aplikačních rout včetně parametrizovaných cest, ochrany JWT a rozdělení na veřejné a chráněné části.

*iam/* obsahuje logiku rozdělenou do složek. *models/* – modely pro přístup k API, které odpovídají jednotlivým doménám (uživatelé, skupiny, fórum, chat, STAG, atd.). Každý model definuje asynchronní metody pro načítání, ukládání nebo aktualizaci dat pomocí *Axios* klienta. *stores/* – stavové manažery implementované pomocí *Pinia*. Udržují a sdílejí reaktivní stav aplikace – například přihlášení uživatele, jazyková lokalizace nebo stav chatu.

*locales/* složka pro správu vícejazyčné lokalizace aplikace. Obsahuje jazykové soubory *cz.js*, *en.js* a *ru.js*, které definují překlady všech zobrazených textů ve struktuře odpovídající jednotlivým modulům aplikace. Lokalizace je řízena pomocí knihovny *vue-i18n.*

*utils/* obsahuje pomocné nástroje a utility, rozdělené do několika podsložek podle účelu

*utils/api/* – konfigurace *Axios* klienta pro HTTP komunikaci s backendem.

*utils/ws/* – konfigurace WebSocket komunikaci přes **STOMP** protokol.

*utils/device/* – detekce typu zařízení (mobil vs desktop).

*utils/groups/* – logika pro kontrolu oprávnění v rámci skupin.

*utils/jwt/* – dekódování JWT tokenu, získávání ID uživatele atd.

*views/* složka s jednotlivými stránkami (pohledy) aplikace, organizovanými podle typu přístupu. *auth/* – veřejné stránky pro registraci, přihlášení, ověření účtu a přístup při odmítnutí. *authorized/* – chráněné stránky pro přihlášené uživatele (např. uživatelské profily, nastavení, skupiny). *forum/* – fóry, členěné na veřejné a soukromé části, včetně správy příspěvků. *layouts/*

– základní rozvržení aplikace pro různé sekce (***HomeLayout***, ***PublicLayout***, ***UserLayout***). *public/* – úvodní a veřejné informační stránky, např. ***LandingView***.

## 9.2 Hlavní závislosti (package.json)

Frontendová část využívá několik klíčových knihoven, které zajišťují rychlý vývoj, správu dat, vícejazyčnou podporu a efektivní stylování uživatelského rozhraní.

*Vue 3 (vue)*

Hlavní framework pro tvorbu reaktivního uživatelského rozhraní založeného na komponentách. Vue 3 nabízí práci s reaktivními daty a tvorbě SPA (Single Page Applications). Díky využití Composition API (*ref, reactive, onMounted, provide, inject* atd) umožňuje lepší organizaci logiky uvnitř komponent, usnadňuje jejich opakované použití a podporuje budoucí rozšiřování projektu.

*Vue Router 4 (vue-router)*

Oficiální knihovna pro správu směrování v aplikaci Vue. Umožňuje deklarativní definici tras, dynamické načítání komponent a ochranu určitých cest pomocí ověřování uživatele. V projektu je využívána k rozdělení cest na veřejné a chráněné sekce a k řízení přechodů mezi jednotlivými pohledy.

*Axios (axios)*

Knihovna pro HTTP komunikaci s REST API. Axios zjednodušuje odesílání požadavků a zpracování odpovědí ze serveru, přičemž podporuje automatickou manipulaci s hlavičkami, zachytávání chyb a práci s promisy.

*STOMP.js a SockJS (@stomp/stompjs, sockjs-client)*

Dvojice knihoven zajišťující real-time komunikaci mezi klientem a serverem pomocí **STOMP** protokolu běžícího nad WebSocket nebo fallback transportech. @stomp/stompjs je oficiální klient pro **STOMP**, zatímco sockjs-client poskytuje kompatibilní připojení přes různé sítě, i v případě, že WebSocket není dostupný. Tento mechanismus je využíván pro implementaci osobního chatu mezi uživateli, doručování zpráv a případně v budoucnu i pro notifikační systém.

*Tailwind CSS (tailwindcss)*

Utility-first CSS framework, který poskytuje množství předdefinovaných tříd pro rychlé a konzistentní stylování komponent bez nutnosti psaní vlastních CSS souborů. Tailwind umožňuje efektivní tvorbu responzivního a estetického designu přímo v šablonách komponen.

*Vue-i18n (vue-i18n)*

Knihovna pro mezinárodní lokalizaci aplikace. Umožňuje spravovat překlady textů, dynamicky měnit jazyk a snadno rozšiřovat aplikaci o podporu nových jazykových mutací. V projektu jsou k dispozici lokalizace do českého, anglického a ruského jazyka.

*Dompurify (dompurify)*

Bezpečnostní knihovna pro čištění potenciálně nebezpečných HTML vstupů. Pomáhá ochránit aplikaci před útoky typu XSS (Cross-Site Scripting) tím, že sanitizuje uživatelsky zadaný obsah před jeho vykreslením na stránce. Použití této knihovny je zásadní zejména v prostředí s možností uživatelských příspěvků nebo komentářů. Použita v forech a skupinah.

*Vite (vite)*

Moderní nástroj pro vývoj a buildování frontendových aplikací. Nabízí extrémně rychlé spuštění serveru během vývoje díky využití ESBuild a hot module replacement (HMR) pro okamžité aktualizace změn bez potřeby obnovení stránky. V produkčním režimu poskytuje optimalizované a efektivní buildy.

*Pinia (pinia)*

Knihovna pro správu stavu v aplikacích Vue 3, považovaná za nástupce Vuex. Poskytuje jednoduché a intuitivní API pro definici stavových objektů (*store*), jejich akcí a getterů. V projektu je Pinia využívána k řízení aplikačního stavu pro autentizaci, uživatelské profily, skupiny, přátelství, fóra, nastavení jazyka atd.

## 9.3 Architektonické principy frontendové části

Frontendová část navržena podle principů **MVI-C** architektury (Model–View–Intent + Coordinator), která přináší jasnou separaci odpovědností, lepší udržovatelnost a snadnější rozšiřitelnost. Celý systém je postaven s využitím *Vue 3, Pinia, Vue Router* a *Tailwind CSS*.

### Model

Každá doménová oblast aplikace (např. autentizace, uživatelé, skupiny) má vlastní model, umístěný ve složce *src/iam/models/*. Modely jsou zodpovědné za přímou komunikaci s backendovým API pomocí **Axios** (***apiClient.js***) nebo **sockJs + STOMP** (*webSocket.js*) klienta a obsahují veškerou síťovou logiku. Vrací čistá data pro další zpracování ve storech. Příklad: ***authModel.js*** definuje metody jako *login(), register(), checkToken()* atd.

### View

View představuje vizuální vrstvu aplikace – tvořenou komponentami Vue umístěnými ve složce *views/*. Každá stránka aplikace (např. ***LoginView.vue***, ***UserProfileView.vue***, ***GroupView.vue***) odpovídá určitému uživatelskému rozhraní a slouží ke zobrazení dat získaných ze store a k vyvolávání interakcí (např. přihlášení, zobrazení příspěvků, změna údajů).

### Intent přes store (Pinia)

Na rozdíl od tradičního **MVI** modelu, kde jsou intenty samostatnými funkcemi, je zde jejich role převzata metodami ve storech. Každý store vytvořený pomocí knihovny **Pinia** obsahuje stav (state), odvozené hodnoty (getters) a akce (actions), které zastupují uživatelské záměry (intents) i jejich zpracování. Například ***authStore.js*** definuje metody jako **login, register, verify**, které reagují na akce uživatele, volají příslušný model a aktualizují stav. Tyto akce typicky nastaví loading stav, vyčistí chyby, provedou síťový požadavek přes model, aktualizují token nebo jiná data a při chybě nastaví příslušnou zprávu. Tento přístup poskytuje čisté, centralizované řízení dat a zároveň zachovává vysokou čitelnost.

### Coordinator

Koordinační vrstva (*src/coordinator/*) centralizuje logiku navigace pomocí Vue Routeru. Umožňuje oddělit řízení přechodů mezi stránkami od prezentační logiky komponent. Např. při úspěšném přihlášení metoda **login()** ve store zavolá **coordinator.navigateToHome()**.

**Autorizace a bezpečnost**

Při každém přechodu na chráněné stránky je ověřena platnost JWT tokenu uloženého v localStorage. Toto ověření probíhá přes metodu **checkToken()** modelu, která volá backendový endpoint */auth/check*. V případě selhání je token odstraněn, vyvolána událost jwt-expired a uživatel je přesměrován na veřejnou stránku s informací o vypršení relace.

**Rozšiřitelnost a udržovatelnost**

Díky použití Vue 3 s Composition API, Pinia pro správu stavu, oddělených modelů pro přístup k API a koordinátoru pro navigaci je celý frontend navržen s důrazem na modularitu. Komponenty lze snadno rozdělit, store rozšiřovat o nové akce a modely o nové API metody bez nutnosti zásadní změny architektury. Struktura tak podporuje vývoj nových funkcionalit, tak údržbu existujících částí a snadné přizpůsobení pro různé typy zařízení (desktop i mobilní rozhraní).

## 9.4 Základní struktura Vue.js aplikace

Frontendová část postavena na technologii **Vue 3** a nástroji **Vite**, který umožňuje velmi rychlý vývoj a build frontendových aplikací. Architektura aplikace je koncipována podle principu modularity a jasného oddělení odpovědností. Použitím principu „konvence nad konfigurací“ (Convention over Configuration) Vue automaticky zajišťuje správné napojení komponent, překladů a stavového řízení bez nutnosti přebytečné konfigurace. Díky tomu je aplikace jednodušší na údržbu, rozšiřitelná a připravená pro více jazyků i typů zařízení.

### 9.4.1 Vstupní bod aplikace

```
import { createApp } from 'vue'
import App from './App.vue'

import router from './coordinator/router'
import createCoordinator from './coordinator/coordinator'

import './assets/main.css'

import { createI18n } from 'vue-i18n'
import { createPinia } from "pinia";
import en from './locales/en'
import ru from './locales/ru'
import cz from './locales/cz'

const i18n : I18n<...>  = createI18n( options: {
    legacy: false,
    locale: 'cz',
    fallbackLocale: 'en',
    messages: {
        en,
        cz,
        ru
    }
})

const pinia : Pinia  = createPinia()
const app : App<Element>  = createApp(App)

app.use(pinia)
app.use(i18n)
app.use(router)
app.provide('coordinator', createCoordinator(router))
app.mount( rootContainer: '#app')
```

*Obrázek 34: App.js vstupní bod Vue.js (vl. zdroj)*

Vstupním bodem aplikace je soubor *main.js*, který se nachází ve složce *src/*. Tento soubor je zodpovědný za vytvoření a spuštění celé Vue aplikace. Pomocí **createApp(App)** se vytvoří instance aplikace Vue. router je importován z vlastní složky *coordinator/*. **createCoordinator()** - globální navigační vrstva a poskytován komponentám pomocí **app.provide**. Pinia slouží pro správu stavů jednotlivých domén aplikace. *vue-i18n* načítá lokalizační soubory ve třech

jazycích (cz, en, ru). Nakonec je aplikace připojena do DOMu přes **app.mount('#app')**, kde **#app** je id hlavního HTML elementu definovaného v *index.html*. Tato inicializace nastavuje veškerý globální kontext aplikace a zajišťuje jednotný přístup ke směrování, překladům a globálnímu stavu.

### 9.4.2 Konfigurační soubory aplikace

Frontend využívá konfigurační přístup založený na nástrojích **Vite**, **Tailwind CSS** a prostředí ***.env***, pro snadné přizpůsobení prostředí vývoje nebo deploy, přehledné definování proměnných a rozšíření základního nastavení bez potřeby nadbytečnou konfigurace.

**package.json**

Základní konfigurační soubor projektu, který definuje název, verzi, závislosti, skripty a další metadata aplikace. Pomocí sekce **scripts** lze spouštět vývojový server (*vite*), build (*vite build*) nebo náhled produkční verze (*vite preview*). V oddílech **dependencies** a **devDependencies** jsou uvedeny knihovny potřebné pro běh aplikace a vývojové prostředí.

**vite.config.js**

Konfigurační soubor nástroje Vite. Konfigurace obsahuje definici používaných pluginů (*vue, tailwindcss, vue-devtools*) a nastavení aliasu **@** pro zjednodušení importů komponent a souborů ze složky *src/*. Dále definuje globální proměnné a rozšíření výchozího prostředí pro lepší vývojářskou zkušenost. Tento přístup zajišťuje flexibilní konfiguraci bez zbytečné zátěže a zároveň zjednodušuje vývoj větších projektů.

**tailwind.config.js**

Soubor slouží ke konfiguraci utility-first frameworku **Tailwind CSS**. Určuje, ve kterých souborech se mají hledat použité CSS třídy (*content*), a rozšiřuje výchozí téma o vlastní barevné proměnné pomocí CSS proměnných definovaných v *:root*. Díky tomu je možné udržovat konzistentní vzhled napříč celou aplikací a snadno přepínat mezi světlým a tmavým režimem nebo přizpůsobovat design podle potřeb projektu.

**.env**

*.env* obsahuje prostředí specifické proměnné, které jsou dostupné v aplikaci pouze za předpokladu, že jejich název začíná prefixem **VITE_.** Tento prefix je požadovaný nástrojem Vite z bezpečnostních důvodů, aby bylo možné explicitně určit, které proměnné budou dostupné i ve frontendovém prostředí.

Typickým příkladem využití *.env* proměnné je adresa backendového API, uložená jako **VITE_BASE_API_URL**. Tato hodnota je následně použita při vytvoření instance klienta

```
import axios from 'axios'

const apiClient : AxiosInstance  = axios.create({
    baseURL: import.meta.env.VITE_BASE_API_URL,
    headers: {
        'Content-Type': 'application/json'
    }
})

apiClient.interceptors.request.use( onFulfilled: config => {
    const token : string  = localStorage.getItem( key: 'token')
    if (token) {
        config.headers.Authorization = `Bearer ${token}`
    }
    return config
})

apiClient.interceptors.response.use(
     onFulfilled: res => res,
     onRejected: async error => {
        return Promise.reject(error)
    }
)

export default apiClient

```

*Obrázek 35: apiClient.js, použity env proměnné (vl zdroj)*

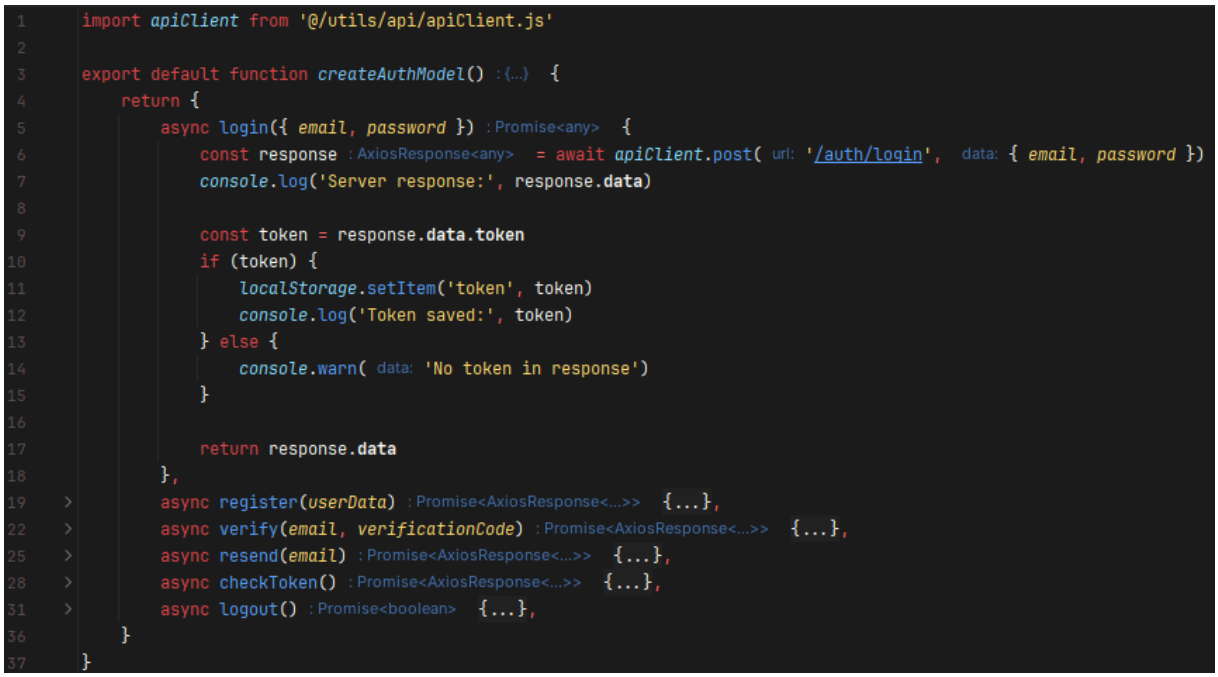
Použití **import.meta.env** je specifické pro Vite. Při buildu Vite automaticky nahradí výrazy **import.meta.env.*** odpovídajícími hodnotami z *.env* souboru podle dev nebo prod nastaveni.

Tento systém konfiguračních souborů zajišťuje vysokou flexibilitu a přehlednost celé aplikace. Umožňuje snadnou správu závislostí, rychlé přizpůsobení prostředí a podporu přenositelnosti bez pevného svazování s konkrétní konfigurací.

## 9.5 Architektura MVI-C

Frontend postaven podle **MVI-C** (Model–View–Intent + Coordinator). Tim zajišťuje oddělení zodpovědností a snadny tok dat mezi jednotlivými vrstvami uživatelského rozhraní. Architektura je plně přizpůsobena prostředí Vue 3, knihovně Pinia pro správu stavu a REST API / STOMP WS na backendu.

### 9.5.1 Model




*Obrázek 36: modely. authModel.js (vl. zdroj)*

Vrstva Model je umístěna ve složce *src/iam/models* a zajišťuje komunikaci s backendem pomocí knihovny Axios. Každý model odpovídá jedné doméně (např. ***authModel.js***, ***userModel.js***, ***groupModel.js***) a vrací metody, které provádějí specifické operace jako přihlášení, registraci, ověření uživatele, získání dat uživatele nebo správu skupin. Modely jsou implementovány jako funkce, které vracejí objekt s asynchronními metodami. Tyto metody využívají předkonfigurovaný klient (***apiClient***). Model je zcela oddělen od uživatelského rozhraní a slouží pouze pro přístup k datům a jejich transformaci z/na formát vhodný pro použití ve frontendové části aplikace.

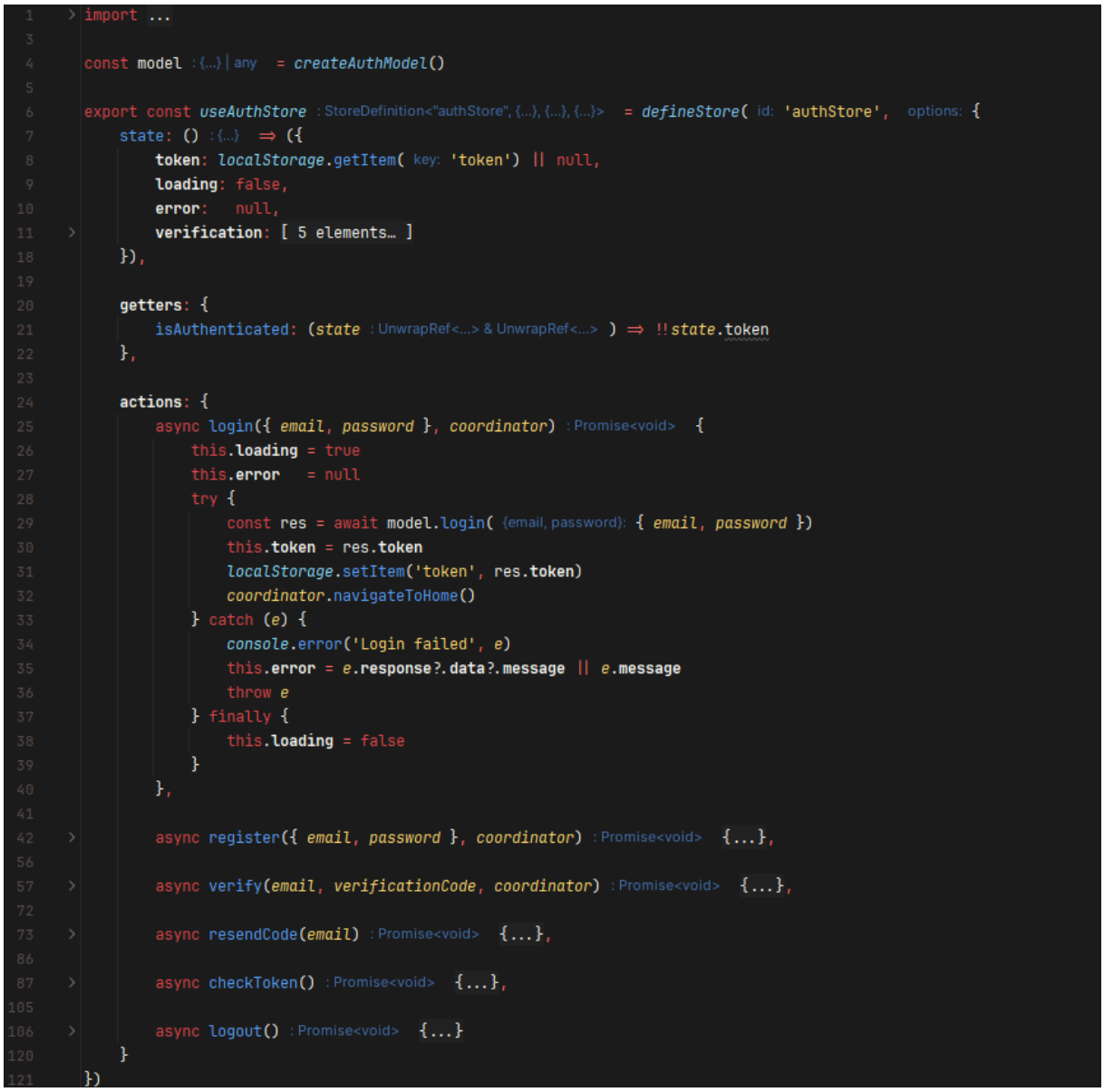
### 9.5.2 View

```vue
<template>
  <div class="min-h-full bg-primary text-text flex flex-col items-center justify-center p-6">
    <h1 class="text-2xl font-semibold mb-6">{{ t('login.title') }}</h1>

    <input
        v-model="email"
        type="email"
        :placeholder="t('login.email')"
        class="w-full max-w-sm px-4 py-2 border border-gray-300 rounded mb-2"
    />
    <input...>

    <button...>

    <p v-if="authStore.error" class="text-red-500 mt-4 text-sm text-center max-w-sm"...>
  </div>
</template>

<script setup>
import ...

const { t } = useI18n()
const coordinator = inject( key: 'coordinator')
const authStore = useAuthStore()

const email = ref( value: '')
const password = ref( value: '')

const isValidEmail = v => /^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(v)

const onLogin = async () => {
  authStore.error = ''
  if (!isValidEmail(email.value)) {
    authStore.error = t('errors.invalidEmail')
    return
  }
  if (!password.value) {
    authStore.error = t('errors.emptyPassword')
    return
  }
  await authStore.login( {email, password}: { email: email.value, password: password.value }, coordinator)
}
</script>
```

*Obrázek 37: views. LoginView.vue (vl. zdroj)*

Vrstva View představuje vizuální rozhraní aplikace a nachází se ve složce *src/views/*. Každý pohled (*.vue) odpovídá konkrétní stránce nebo části aplikace – například ***LoginView.vue***, ***UserSettingsView.vue***, ***GroupPostsView.vue***. Komponenty využívají možnosti Vue 3 včetně Composition API (*ref, inject, onMounted*, apod.) a jsou strukturovány jako jednoduché, znovupoužitelné a přehledné bloky uživatelského rozhraní. Ve View komponentách jsou definovány vstupní pole, tlačítka, validační logika a přímé vazby na stavové proměnné spravované prostřednictvím Pinia storů. Výsledkem je reaktivní a responzivní UI, které okamžitě reaguje na změny stavu. View komponenty neobsahují žádnou aplikační logiku ani síťovou komunikaci – pouze vizualizují data, validace vstupu a vyvolávají záměry uživatele (např. kliknutí na tlačítko přihlášení).

### 9.5.3 Intent (přes Store)




*Obrázek 38: stores. authStore.js (vl. zdroj)*

V této implementaci je Intent sloučen s logikou Actions a je realizován pomocí Pinia storů, které se nacházejí ve složce *src/iam/stores/*. Každý store (***authStore.js, userStore.js, groupStore.js***) definuje:

*state* – centrální stav pro danou doménu (např. *token, loading, error*)

*getters* – odvozené hodnoty (např. *isAuthenticated*)

*actions* – logiku reagující na záměry uživatele (např. *login, register, logout*)

Actions v tomto pojetí zastupují jak interpretaci intentu, tak jeho zpracování – jsou tedy zodpovědné za vykonání požadované operace (např. přihlášení uživatele), volání metod modelu a správné nastavení stavových proměnných.

Použitím Pinia jako jednotné vrstvy pro správu stavů i logiky je zajištěna vysoká čitelnost a přehlednost celé aplikace. Store může být snadno znovu použit a rozšířen bez nutnosti složitého propojování různých vrstev. Intenty v tomto přístupu nejsou reprezentovány jako samostatné objekty, ale jako pojmenované metody ve storu – např. *login({ email, password }, coordinator)*.

### 9.5.4 Coordinator (Koordinátor a router)

#### Coordinator

Součástí architektury MVI-C je Coordinator, který plní roli centrální vrstvy pro správu navigace. Jeho cílem je odstranit přímé závislosti komponent na routeru a soustředit logiku přechodů mezi stránkami do jednoho místa. Koordinátor tak slouží jako rozhraní pro směrování, které mohou jednotlivé vrstvy (např. store nebo komponenty) využívat bez přímého importu *vue-router*. Implementován koordinátor jako funkce *createCoordinator(router)*, která přijímá instanci routeru a vrací objekt s metodami jako *navigateToLogin(), navigateToHome(), navigateToUserSettings()* nebo *navigateToForum(id)*. Každá metoda definuje jednoznačnou navigační cestu, a zároveň poskytuje dodatečnou logiku – například kontrolu existence uživatelského ID, přesměrování na výchozí variantu, nebo otevření externího odkazu v nové záložce. Tento přístup zvyšuje čitelnost a rozšiřitelnost kódu, snižuje duplikaci a umožňuje snadno spravovat chování přechodů na jednom místě. Díky použití *provide('coordinator')* ve vstupním bodě aplikace je dostupný v celé aplikaci přes *inject('coordinator')*.

#### Router

Vue Router je základní knihovnou pro správu trasování v aplikaci. V projektu je nakonfigurován pomocí *createRouter()* s využitím historie (*createWebHistory*) a struktury nested routes, které odpovídají jednotlivým rozvržením (*layouts*) jako ***PublicLayout*** a ***HomeLayout***. Cesty jsou rozděleny na veřejné *(/, /login, /register, /forum, /user/search)*. Oni jsou dostupné bez přihlášení a chráněné *(/home, /app/groups, /user/:id, /chat/:id, /app/forum)* – vyžadují platný JWT token. Díky vlastnosti meta: *{ requiresAuth: true }* je možné jednoduše označit chráněné cesty a následně je kontrolovat v *router.beforeEach*. Tato globální navigační provádí kontrolu, zda má uživatel platný token, v případě absence tokenu přesměrování na veřejnou stránku. V případě chyby autorizace (např. 401 nebo 403) odhlášení a přesměrování.

Ochranu přístupu k *login, register, verify* pro již přihlášené uživatele. Zároveň zajišťuje volání metody *checkToken()* ve storu, která komunikuje s backendem a ověřuje platnost tokenu.

Celý směrovací systém podporuje rozdělení UI na různé přístupové vrstvy, umožňuje použití dynamických parametrů a je snadno rozšiřitelný o další cesty nebo podmínky přístupu. Díky zapojení koordinátoru je zároveň flexibilní a dobře udržovatelný.

# 10 Uživatelské rozhraní

Snímky aplikace jsou součásti přílohy „A“.

## 10.1 Autorizace a autentizace uživatelů v aplikaci

Autorizace a autentizace představují klíčovou část aplikace, která zajišťuje bezpečnost přístupu k jednotlivým funkcím platformy. Proces autorizace zahrnuje registraci, ověření e-mailové adresy a následné přihlášení uživatele.

### 10.1.1 Stránka registrace

Registrace představuje vstupní bod do aplikace. Uživatel zadává svůj e-mail a dvakrát heslo. Probíhá kontrola formátu e-mailu, povinné pole. Heslo musí splňovat kritéria složitosti, jako minimální délka (obvykle 8 znaků) a přítomnost různých znaků (malá a velká písmena, číslice, speciální znaky). Po úspěšné validaci je uživateli zaslán verifikační kód na zadaný e-mail, který je nutný k aktivaci účtu.

### 10.1.2 Stránka ověření e-mailové adresy (verifikace)

Na stránce verifikace zadává uživatel obdržený šestimístný verifikační kód. Je taky možnost opětovného zaslání verifikačního kódu prostřednictvím tlačítka „*Znovu odeslat ověřovací kód*“. Kromě ručního zadání kódu existuje možnost provést verifikaci přímo pomocí odkazu zaslaného e-mailem, který obsahuje verifikační kód jako URL parametr.

### 10.1.3 Ukázka e-mailu s verifikačním kódem

Uživatel obdrží e-mail obsahující verifikační kód a odkaz pro přímé ověření. E-mailova zprava v má šestimístný unikátní kód, platný po určitou době a odkaz pro přímou aktivaci účtu („Verify My Account“).

### 10.1.4 Stránka přihlášení (login)

Na stránce přihlášení uživatel zadává svůj e-mail a heslo použité při registraci. Probíhá zde kontrola existence zadané e-mailové adresy v databázi a ověření správnosti hesla vzhledem k uloženému hash heslu. Po úspěšné autentizaci je uživateli vystaven JWT token, který slouží k autorizaci v aplikaci.

### 10.1.5 Modální okna při odhlášení a vypršení platnosti tokenu

Pokud uživatel provede akci odhlášení nebo pokud platnost JWT tokenu skončí, objeví se modální okno informující uživatele o provedení odhlášení nebo vypršeni tokenu. Obsahuje text

informující uživatele a tlačítko přesměrování na domovskou stránku aplikace nebo na login stránku v případě vypršeni tokenu.

## 10.2 Správa skupin v aplikaci

Modul skupin v aplikaci umožňuje uživatelům vytvářet a spravovat tematické komunity. Tyto skupiny mohou být veřejné nebo soukromé, přirozené k univerzitní doméně nebo ne, čím určuje míru přístupu ostatních uživatelů k obsahu skupiny.

### 10.2.1 Formulář pro vytvoření skupiny

Formulář vytvoření skupiny obsahuje: výběr domény (volitelně) – uživatel může vybrat univerzitní doménu, ke které se skupina vztahuje, název skupiny – povinný údaj, je validován na délku a unikátnost, popis skupiny – krátký text s bližšími informacemi o skupině, témata – klíčová slova oddělená čárkou, která slouží ke snadnějšímu vyhledávání, viditelnost skupiny – pokud není zaškrtnuto, skupina bude soukromá a její obsah dostupný pouze členům.

### 10.2.2 Prohlížeč skupin (browser)

Stránka prohlížeče skupin umožňuje vyhledávat a filtrovat skupiny podle následujících kritérií: název skupiny, příslušnost k univerzitní doméně, viditelnost (veřejná/soukromá), témata skupiny, počet skupin zobrazených na stránku (stránkování). U každé skupiny je uvedena její viditelnost, název a stručný seznam témat.

### 10.2.3 Stránka uzavřené (soukromé) skupiny

Uživatelé, kteří nejsou členy soukromé skupiny, vidí pouze základní informace a mají možnost požádat o připojení pomocí tlačítka „Připojit se“. Po schválení administrátorem skupiny budou mít plný přístup ke všem částem skupiny.

### 10.2.4 Správa členů skupiny

Stránka členů skupiny umožňuje administrátorům a vlastníkovi skupiny spravovat členství uživatelů. Nabízí: seznam členů se zobrazením jména, statusu (čekající, aktivní apod.) a role, vyhledávání podle jména uživatele a možnost filtrování členů podle statusu a role. Pro administrátory jsou dostupná tlačítka pro schvalování nebo odmítání žádostí o vstup, tlačítka *kick, ban, unban* a *choicebox* pro nastaveni role clenu skupiny.

### 10.2.5 Formulář pro vytvoření nového příspěvku

Uživatelé s dostatečnou úrovní oprávnění (od role člena a výše, v závislosti na nastavení skupiny) mohou přidávat příspěvky do skupiny. Formulář obsahuje: název příspěvku (titulek), témata příspěvku (oddělená čárkou), obsah příspěvku s podporou bohatého formátování

(Markdown editor s podporou zvýraznění textu, nadpisů, vložení odkazů a obrázků, s satinaci pomoci *dompurify* proti XSS útoku).

### 10.2.6 Novostní kanál skupiny

Stránka příspěvků představuje hlavní prostor pro komunikaci v rámci skupiny. Je dostupná všem schváleným členům a umožňuje: Zobrazování všech příspěvků ve skupině chronologicky, možnost hledání podle názvu nebo tématu příspěvků, detailní zobrazení obsahu příspěvku, včetně obrázků a externích odkazů, editaci nebo mazání vlastních příspěvků (nebo příspěvků, na které má uživatel práva).

### 10.2.7 Obecné informace o skupině

Stránka obecných informací o skupině poskytuje přehled základních údajů, jako jsou: název a popis skupiny, datum založení skupiny, seznam témat spojených se skupinou, informace o vlastníkovi a členství včetně rolí jednotlivých uživatelů, stav skupiny (veřejná/soukromá). Na tyto stance je taky tlačítko «opustit skupinu» pro běžné členy a tlačítko «odstranit skupinu» dostupné pouze vlastníku.

## 10.3 Modul fóra v aplikaci

Fórum slouží k diskusi a výměně informací mezi studenty na různá témata spojená s univerzitním prostředím. Poskytuje uživatelům prostor pro sdílení zkušeností, otázek či informací.

### 10.3.1 Formulář vytvoření fóra

Při zakládání nového fóra uživatel vyplňuje několik povinných a nepovinných údajů: jasný a výstižný název tématu diskuse, detailnější popis a účel diskuse, klíčová slova, která umožňují snadnější vyhledávání a kategorizaci, uživatel může určit příslušnou doménu, pokud fórum souvisí s konkrétní univerzitou, veřejné – pokud není označeno, fórum zůstane soukromé. Pokud má uživatel administrátorské oprávnění, získává navíc možnost nastavit pokročilé parametry pro nastaveni minimální role uživatelů – určuje minimální úroveň oprávnění nutnou pro přidávání příspěvků, připnutí – připnutí fóra na začátek seznamu pro větší viditelnost a informační – označení diskuse jako informačního charakteru bez možnosti odpovědí.

### 10.3.2 Prohlížeč fór (browser)

Uživatelé mohou snadno vyhledávat existující fóra pomocí pokročilých filtrů: Hledání podle názvu, témat, domén nebo statusů (aktivní, uzavřené, archivované). Řazení výsledků podle data

vytvoření či aktivity. Každé fórum zobrazuje stručné informace: název, popis, témata, doménu a stav viditelnosti.

### 10.3.3 Detail fóra (stránka fóra)

Na stránce fóra uživatelé vidí kompletní obsah diskuse: Úvodní příspěvek se zadaným názvem a popisem. Seznam příspěvků s možností odpovědět na konkrétní zprávu („Reply“). Podpora vkládání zpráv s možností formátování (Markdown editor s podporou zvýraznění textu, nadpisů, vložení odkazů a obrázků, s satinaci pomoci *dompurify* proti XSS útoku).

### 10.3.4 Nastavení fóra

Pro administrátory a vlastníky fóra jsou k dispozici pokročilé nástroje pro správu a moderaci: možnost úpravy názvu, popisu a témat fóra a nastavení domény a změna viditelnosti (veřejné/soukromé) a speciální akce, z kterých: uzavření fóra – zamezí dalším odpovědím, archivace fóra – fórum je archivováno, zůstává čitelné, ale není možné přidávat nové zprávy, označení jako vyřešené – indikuje, že problém či téma diskuse je vyřešen, smazání fóra – kompletně odstraní fórum a veškerý obsah z aplikace.

## 10.4 Uživatelský profil v aplikaci

### 10.4.1 Osobní profil uživatele

Osobní profil obsahuje informace viditelné samotnému uživateli, který má možnost měnit údaje: Jméno, e-mail, status aktivity, osobní popisek. Uživatel může připojit účet IS/STAG, čímž se automaticky načtou informace o studiu (Osobní číslo, fakulta, studijní program, ročník). Možnost odebrat uživatele ze seznamu přátel. Odkaz na stránku nastavení profilu.

### 10.4.2 Veřejný profil uživatele

Veřejný profil zobrazuje omezené informace dostupné ostatním uživatelům: Základní informace (jméno, uživatelské jméno, status aktivity). Informace ze systému STAG (pokud jsou dostupné). Možnost odeslat žádost o přátelství nebo poslat zprávu. Seznam přátel uživatele (pokud jsou viditelní).

### 10.4.3 Propojení se STAGem (správa tokenu)

Uživatelé mají speciální stránku, kde mohou spravovat připojení k systému IS/STAG.

Tato stránka nabízí: výběr univerzity ze seznamu, Indikátor stavu platnosti tokenu STAG, možnost zahájit přihlášení do STAGu, možnost odebrat uložený token. Tato integrace umožňuje automaticky načítat studijní data uživatelů přímo z univerzitního systému.

### 10.4.4 Vyhledávání uživatelů

Uživatelé mají k dispozici pokročilé vyhledávání ostatních členů aplikace, včetně vyhledávání podle jména, příjmení, e-mailu, filtrování podle domény, fakulty, ročníku, studijního oboru nebo titulů. Výsledky jsou prezentovány ve formě karet s základními kontaktními informacemi a možností přejít na profil vybraného uživatele.

### 10.4.5 Stránka nastavení profilu

Uživatelé mohou ve svém profilu nastavovat detailní informace v rámci několika kategorií. Zabezpečení – změna hesla a správa přístupu. Kontakty – telefon, sociální sítě (Instagram, Facebook, Telegram atd.). Osobní údaje – datum narození, lokalita a další soukromé informace. O mně – volný text popisující uživatele, status, seznam zájmů, dovedností a osobní webové stránky. U každé položky je možnost editace či odstranění.

### 10.4.6 Modul chatu mezi uživateli

Chat poskytuje soukromou komunikaci mezi dvěma uživateli v reálném čase. Podporované funkce zahrnují posílání textových zpráv, možnost odpovídat na konkrétní zprávy a automatické posouvání historie zpráv při nových zprávách pro lepší orientaci uživatelů.

# 11 Budoucí vylepšení a rozšíření aplikace

Aplikace nabízí široký prostor pro budoucí rozvoj a rozšíření funkcionalit. Je zde možnost přidání nových způsobů autorizace, například pomocí OAuth 2.0, rozšíření o tržiště (bazar) pro výměnu a prodej předmětů, implementace osobních a skupinových kalendářů, zavedení přísného oddělení obsahu podle univerzitních domén, přidání systému notifikací a integrace message brokera pro efektivnější asynchronní komunikaci. Dále je možné připojit úložiště souborů a videí, rozšířit chat o skupinové konverzace či multimediální obsah, a vytvořit informační stránky pro jednotlivé univerzitní domény. Z hlediska technického rozvoje se nabízí prostor pro zlepšení designu, dekompozici frontendové části na komponenty, vytvoření mobilní aplikace, přidání veřejného API například pro propojení s Telegramem, zavedení podpory přepínání vzhledových témat (světlý/tmavý režim), vytvoření administrátorského rozhraní a přidání funkcí pro zaměstnance univerzit, které jim umožní moderovat obsah a správu platformy.

# ZÁVĚR

Tato bakalářská práce se zabývala návrhem a realizací studentské komunitní platformy, jejímž hlavním cílem bylo vytvořit centralizované prostředí usnadňující vzájemnou komunikaci, sdílení materiálů a organizaci studentských skupin na jedněm miste. Navržené řešení reaguje na reálné potřeby studentů, identifikované pomocí podrobné analýzy uživatelských požadavků, a přináší jim jednotný prostor pro interakci, sdílení informací i efektivnější integraci do akademického života.

V rámci práce byla úspěšně navržena a implementována aplikace, jejíž backend využívá framework Spring Boot, který zaručuje robustnost, bezpečnost a udržitelnost. Frontendová část byla vytvořena pomocí frameworku Vue.js s architektonickým vzorem MVI-C pro zajištěni vysokou flexibility, přehlednosti kódu a možnosti snadného dalšího rozvoje. Databázová vrstva využívá systém PostgreSQL, přičemž veškerá infrastruktura je nasazena v kontejnerizovaném prostředí s Docker Compose.

Realizovaná platforma splnila stanovené cíle – především nabídla studentům efektivní nástroje pro komunikaci, tvorbu a správu studijních skupin, soukromé i veřejné profily, integrovanou autentizaci s ověřováním přes e-mail a úspěšně implementovanou integraci se studijním systémem STAG.

Práce také identifikovala oblasti pro budoucí rozvoj a rozšíření. Doporučeno je zejména zavedení dalších metod autentizace, například OAuth2, implementace pokročilých funkcí, jako jsou kalendáře a notifikace, vytvoření informačních stránek fakult nebo administrátorského panelu. Velký potenciál má i vývoj mobilní aplikace či otevření platformy směrem k externím službám.

Věřím, že tato práce poskytuje pevný základ pro další rozvoj studentské komunitní platformy a přispěje ke zkvalitnění akademického života na univerzitě. Osobně považuji zkušenosti získané během realizace za mimořádně přínosné, neboť práce na tomto projektu přinesla cenné poznatky nejen v oblasti technologií, ale také v efektivním řešení reálných uživatelských potřeb a v oblasti návrhu a budování architektury aplikace.

# POUŽITÁ LITERATURA

[1] ACKERMANN, Philip. *Full Stack Web Development: The Comprehensive Guide.* Rheinwerk Computing, 2023.

[2] FISHER, Derek. *Application Security Program Handbook: A guide for software engineers and team leaders.* Manning, 2022.

[3] JACKSI, Karwan and Shakir M. Abass. "Development History Of The World Wide Web." *International Journal of Scientific & Technology Research*, 2019. Dostupné z: https://www.semanticscholar.org/paper/Development-History-Of-The-World-Wide-Web-Jacksi-Abass/a6b82f4a6caabd0010343793c834e9d5de6cc09c

[4] CLASSIFYING MODEL-VIEW-CONTROLLER SOFTWARE APPLICATIONS USING SELF-ORGANIZING MAPS, Guamán et al., 2021. Dostupné z: https://www.semanticscholar.org/paper/Classifying-Model-View-Controller-Software-Using-Guam%C3%A1n-Delgado/cf2875b64995fd753c2be54c3d33ef30f48236e0

[5] YODGORBEK Komilov. **Android Architecture Patterns: Comparing MVC, MVP, MVVM, and MVI** (2024)., Medium Dostupné z: https://medium.com/%40YodgorbekKomilo/android-architecture-patterns-comparing-mvc-mvp-mvvm-and-mvi-b282712cfb46

[6] **KURAPATI, Lakshmanarao.** (2024). Different Types of Architectures in Frontend Design and Development. *International Journal for Multidisciplinary Research*, IJFMR240528707, September–October 2024. Dostupné z: https://www.researchgate.net/publication/388486719_Different_Types_of_Architectures_in_Frontend_Design_and_Development

[7] **ASIMIYU, Zainab.** (2024, November). Evaluating Android Architectural Patterns: A Deep Dive into MVP, MVVM, and MVI. Dostupné z: https://www.researchgate.net/publication/385492745_Evaluating_Android_Architectural_Patterns_A_Deep_Dive_into_MVP_MVVM_and_MVI

[8] **Performance Comparison of Native Android Application on MVP and MVVM**, Bambang Wisnuadhi, Ghifari Munawar, & Ujang Wahyu. (2020). Dostupné z: https://www.researchgate.net/publication/348060919_Performance_Comparison_of_Native_Android_Application_on_MVP_and_MVVM

[9] Choma, D., Chwaleba, K., & Dzieńkowski, M. (2022). The Efficiency and Reliability of Backend Technologies: Express, Django, and Spring Boot. *International Journal of Advanced Computer Science and Applications*, 13(5). Dostupné z: https://www.researchgate.net/publication/376709512_THE_EFFICIENCY_AND_RELIABILITY_OF_BACKEND_TECHNOLOGIES_EXPRESS_DJANGO_AND_SPRING_BOOT

[10] Shaikh, A. (2025). Node.js vs Django vs Laravel: Which is the best back-end web framework? *Peerbits Blog*. Dostupné z: https://www.peerbits.com/blog/nodejs-vs-django-vs-laravel-back-end-web-framework.html

[11] Odeniran, Q. (2023). *Comparative Analysis of Fullstack Development Technologies: Frontend, Backend and Database*. Georgia Southern University. Dostupné z: https://digitalcommons.georgiasouthern.edu/etd/2663/

[12] Kramer, N. (2024). Backend Frameworks List: Choosing the Right One. *Daily.dev Blog*. Dostupné z: https://daily.dev/blog/backend-frameworks-list-choosing-the-right-one

[13] Framework Training. (2024). Spring Boot vs. Node.js vs. .NET Core vs. Django: a "celebrity face-off". *Framework Training News & Insights*. Dostupné z: https://www.frameworktraining.co.uk/news-insights/spring-boot-vs-node-js-versus-dot-net-core-vs-django/

[14] Odeniran, Qozeem. „Comparative Analysis of Fullstack Development Technologies: Frontend, Backend and Database." Georgia Southern University, 2023. Dostupné z: https://digitalcommons.georgiasouthern.edu/etd/2663

[15] Li, Nian a Bo Zhang. „The Research on Single Page Application Front-end development Based on Vue." Journal of Physics: Conference Series, 2021. Dostupné z: https://www.researchgate.net/publication/351145453_The_Research_on_Single_Page_Application_Front-end_development_Based_on_Vue

[16] CHAMBERLIN, Donald. "50 Years of Queries." Communications of the ACM, vol. 67, no. 8, August 2024, pp. 110–121. doi:10.1145/3649887. Dostupné z: https://dl.acm.org/doi/10.1145/3649887

[17] Docker, Inc. "Four Ways Docker Boosts Enterprise Software Development." Dostupné z: https://www.docker.com/blog/four-ways-docker-boosts-enterprise-software-development/#:~:text=For%20workloads%20targeting%20on,are%20still%20dominant

[18] BMC Software. "State of Containers." Dostupné z: https://www.bmc.com/blogs/state-of-containers/#:~:text=The%20primary%20reason%20for%20leveraging,%E2%80%9D

[19] Zero To Mastery. "Angular vs React vs Vue: Which Framework Should You Choose?" Dostupné z: https://zerotomastery.io/blog/angular-vs-react-vs-vue/#:~:text=Angular%20vs%20React%20vs%20Vue%3A,Detailed%20Documentation%3A%20Vue%27s

[20] PostgreSQL. Dostupné z: https://www.postgresql.org/

[21] Integrate.io. "PostgreSQL vs MySQL: Which One Is Better for Your Use Case?" Dostupné z: https://www.integrate.io/blog/postgresql-vs-mysql-which-one-is-better-for-your-use-case/#:~:text=Here%20are%20some%20of%20the,defining%20characteristics%20of%20PostgreSQL

[22] Spring Boot. Dostupné z https://spring.io/projects/spring-boot

[23] Spring. Dostupné z https://spring.io/

[24] IJFMR. (2024). Spring Boot, introduced by Pivotal, enhances productivity in enterprise application development. International Journal for Multidisciplinary Research, IJFMR28930, Dostupné z: https://www.ijfmr.com/papers/2024/5/28930.pdf

[25] Vue.js. Dostupné z: https://vuejs.org/

[26] Invedus. "Angular vs React vs Vue.js – A Comparative Study." Dostupné z: https://invedus.com/blog/angular-vs-react-vs-vue-js-a-comparative-study/

[27] Facebook. Dostupné z: https://www.facebook.com

[28] Reddit. Dostupné z: https://www.reddit.com

[29] Telegram. Dostupné z: https://telegram.org

[30] LinkedIn. Dostupné z: https://www.linkedin.com

[31] Instagram. Dostupné z: https://www.instagram.com

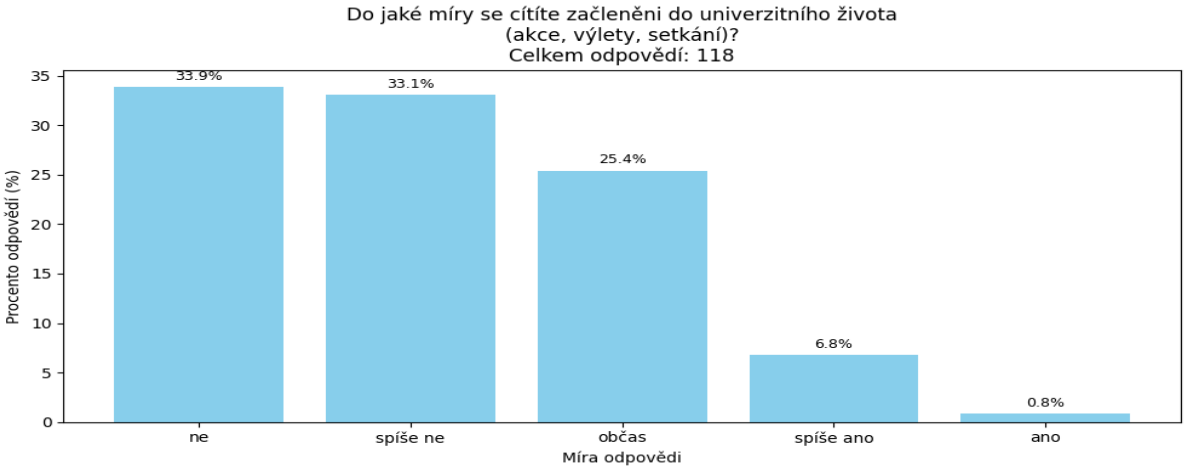
[32] Spring Initializr Dostupné z https://start.spring.io/

[33] TRĘTOWICZ M., ZALEWSKI A., TURKOWSKI J., MVC MVP and MVVM architecture pattern – Introduction, itCraft, Dostupné z https://itcraftapps.com/blog/mvc-mvp-and-mvvm-architecture-pattern-introduction/

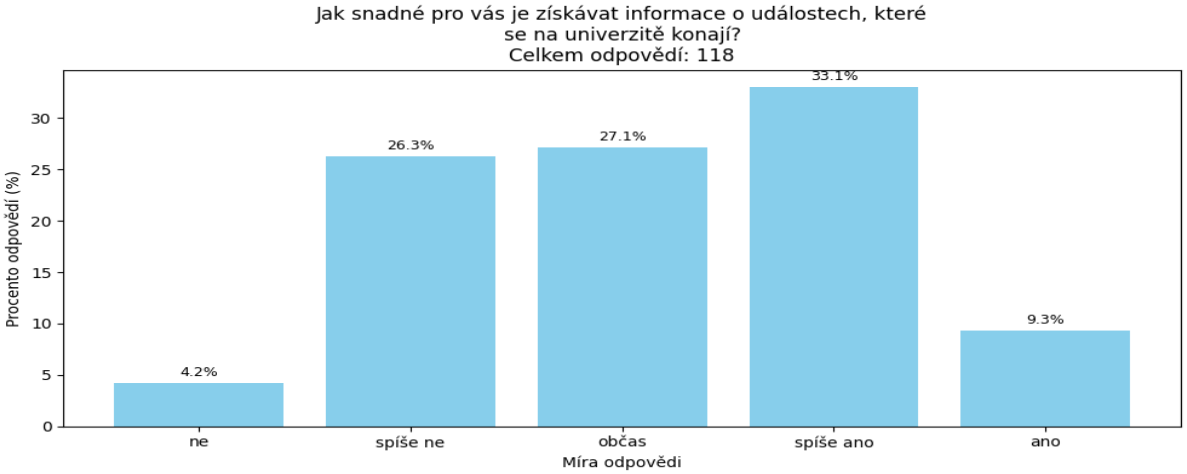
[34] BUDHABHOOSHAN Patil, Understanding MVVM-C, LinkedIn, Dostupné z: https://www.linkedin.com/pulse/understanding-mvvm-c-budhabhooshan-patil/

# SEZNAM PŘÍLOH

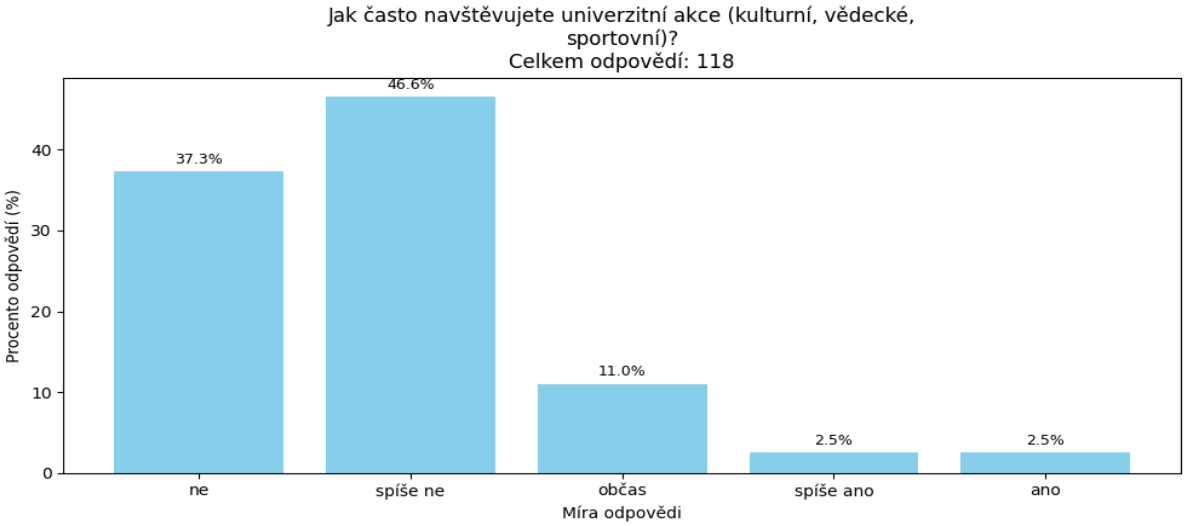
# PŘÍLOHA A: *Grafy z analýzy – Uživatelské chování a zkušenosti*



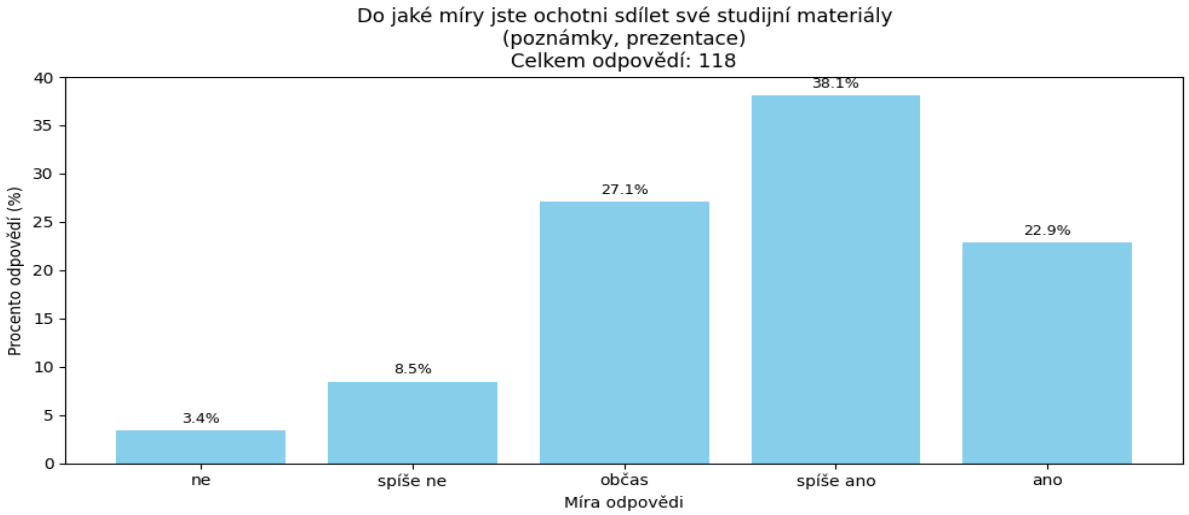

Obrázek *1: Do jaké míry se cítíte začleněni do univerzitního života? (*vl. zdroj)



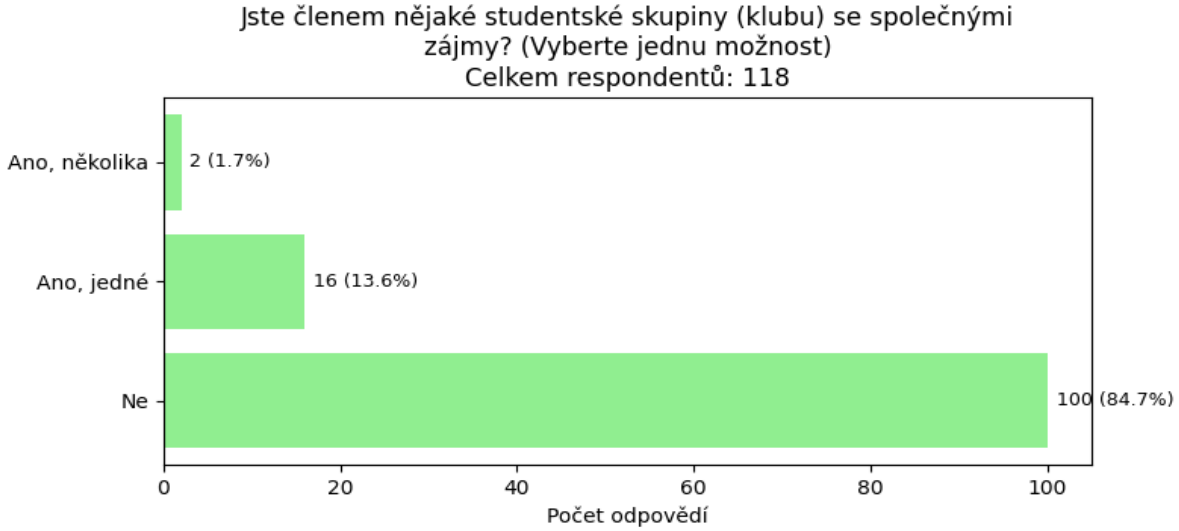

Obrázek *2: Jak snadné pro vás je získávat informace o událostech, které se na univerzitě konají?* (vl. zdroj)



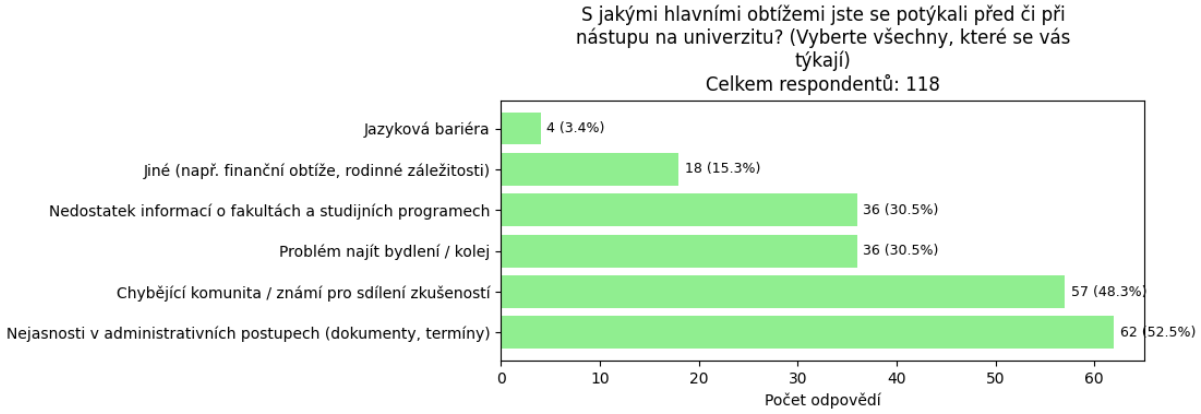

Obrázek *3: Jak často navštěvujete univerzitní akce (kulturní, vědecké, sportovní)?* (vl. zdroj)



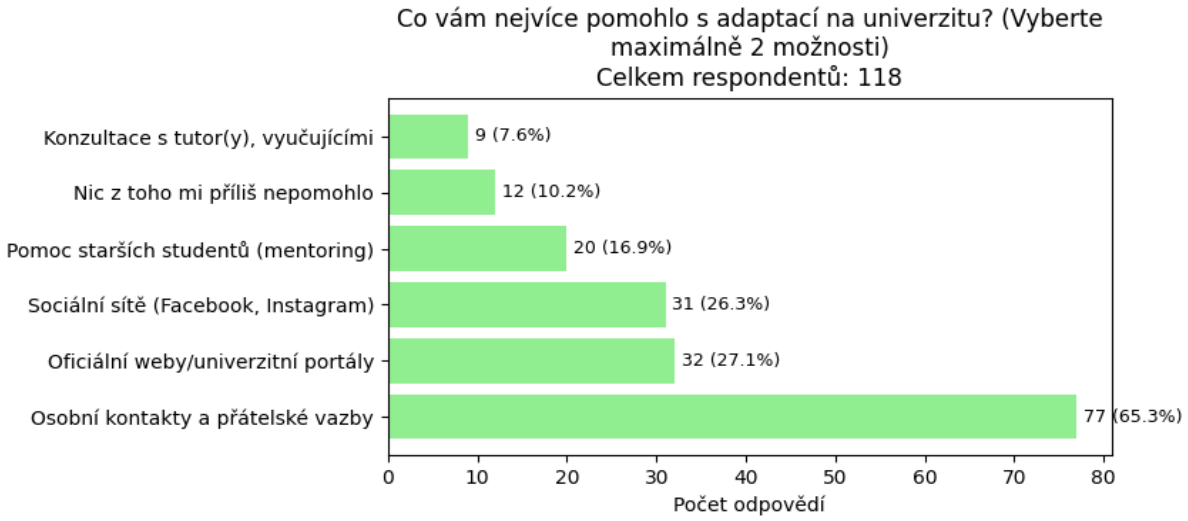

Obrázek 4*: Do jaké míry jste ochotni sdílet své studijní materiály* (vl. zdroj)



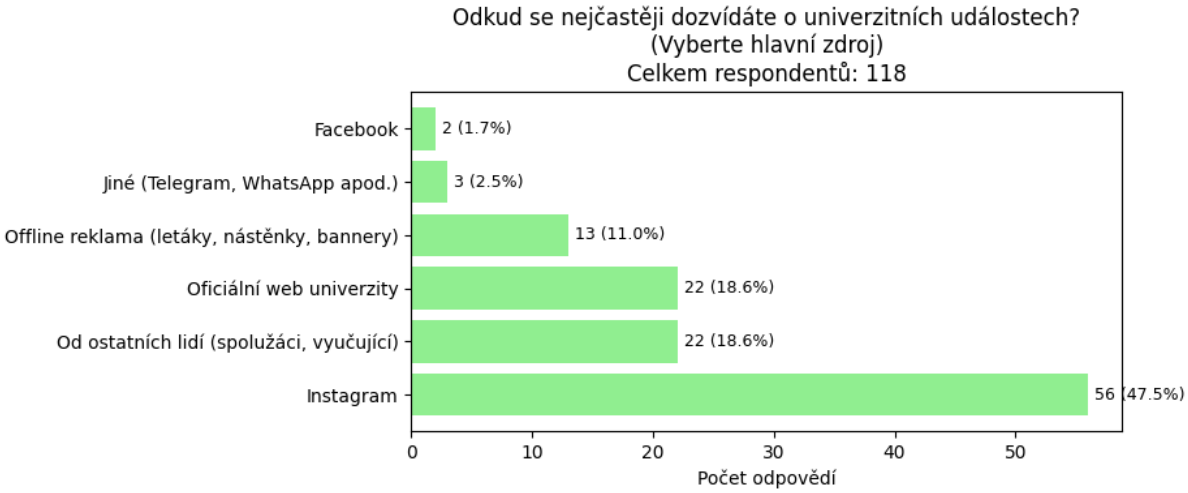

Obrázek 5*: Jste členem nějaké studentské skupiny (klubu) se společnými zájmy?* (vl. zdroj)



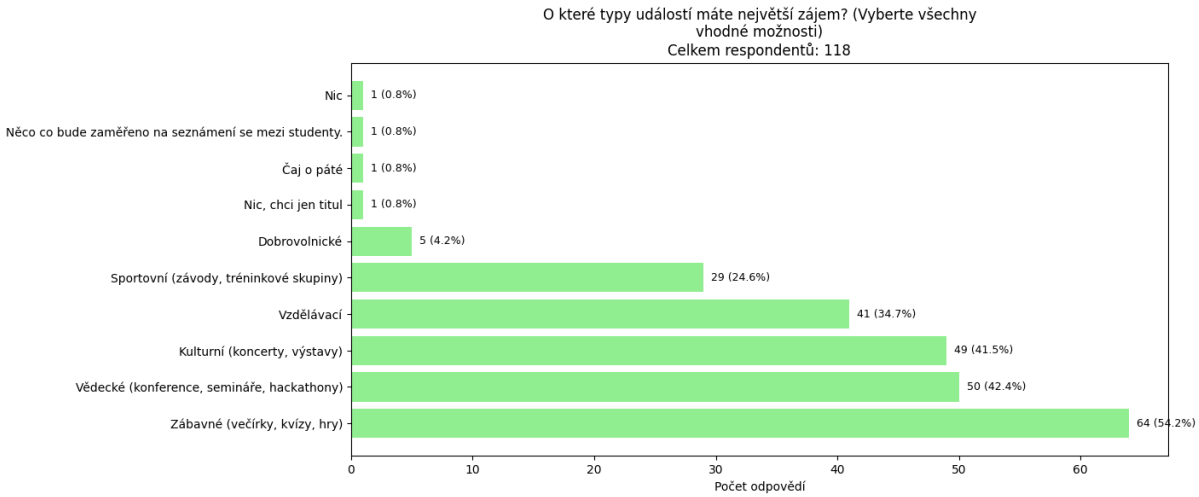

Obrázek 6*: S jakými hlavními obtížemi jste se potýkali před či při nástupu na univerzitu?* (vl. zdroj)




Obrázek *7: Co vám nejvíce pomohlo s adaptací na univerzitu?* (vl. zdroj)

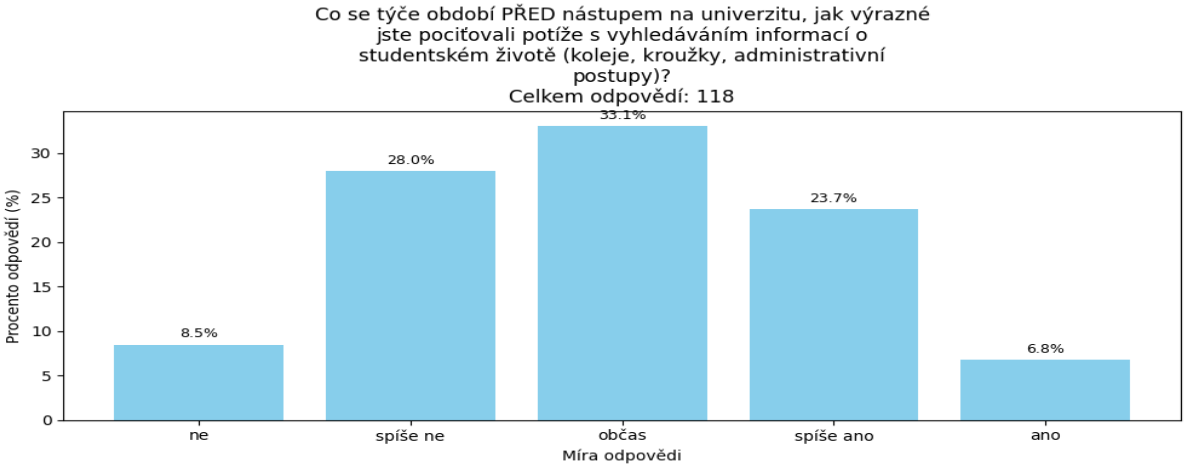



Obrázek *8: Odkud se nejčastěji dozvídáte o univerzitních událostech?* (vl. zdroj)

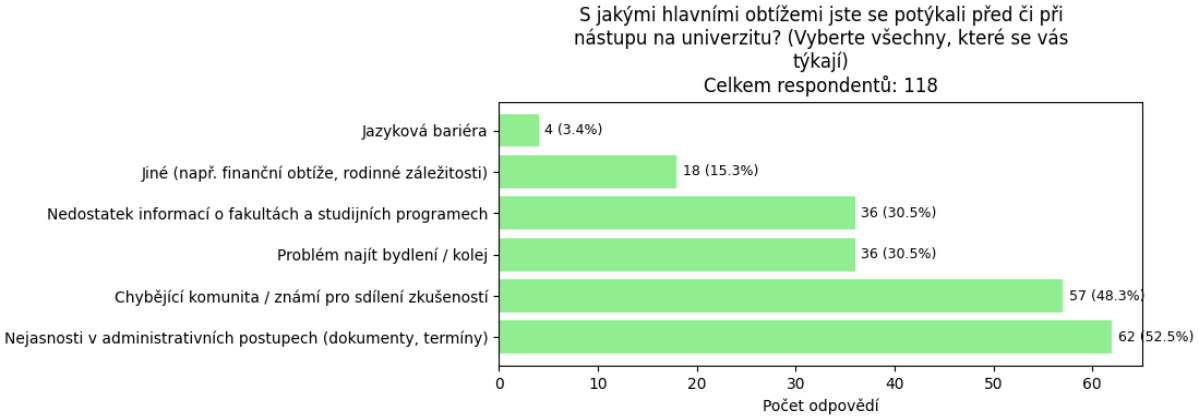



Obrázek *9: O které typy událostí máte největší zájem?* (vl. zdroj)

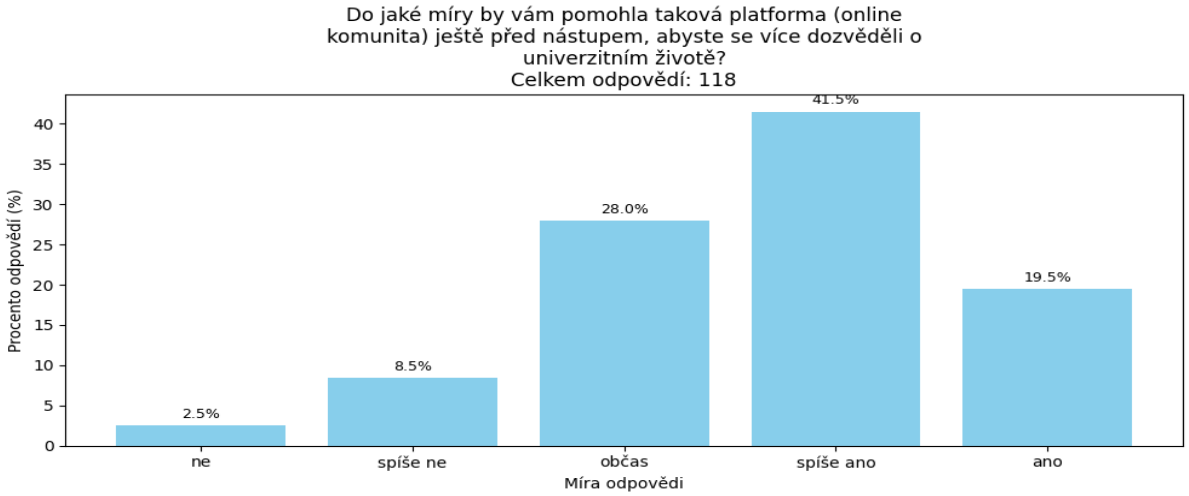
# PŘÍLOHA B: *Grafy z analýzy – Přednástupová fáze a informační bariéry*



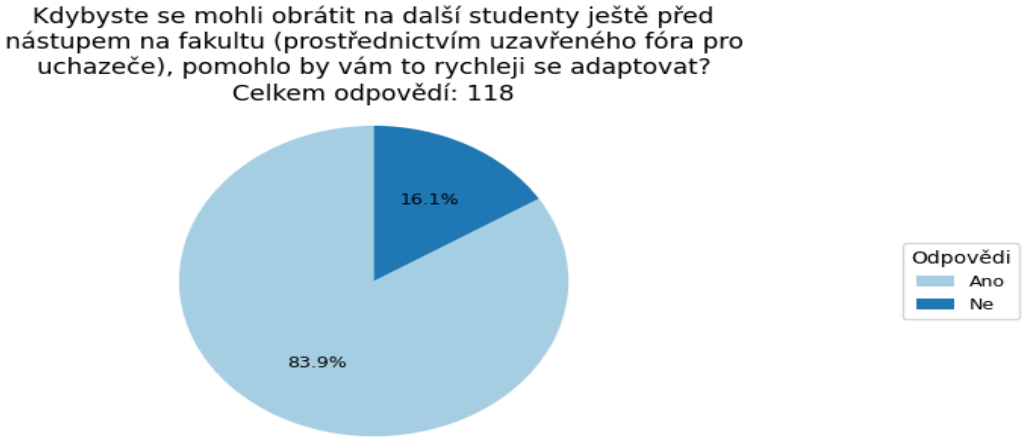

Obrázek *1: Co se týče období PŘED nástupem na univerzitu potíže s vyhledáváním informací?* (vl. zdroj)




Obrázek *2: S jakými hlavními obtížemi jste se potýkali před či při nástupu na univerzitu?* (vl. zdroj)

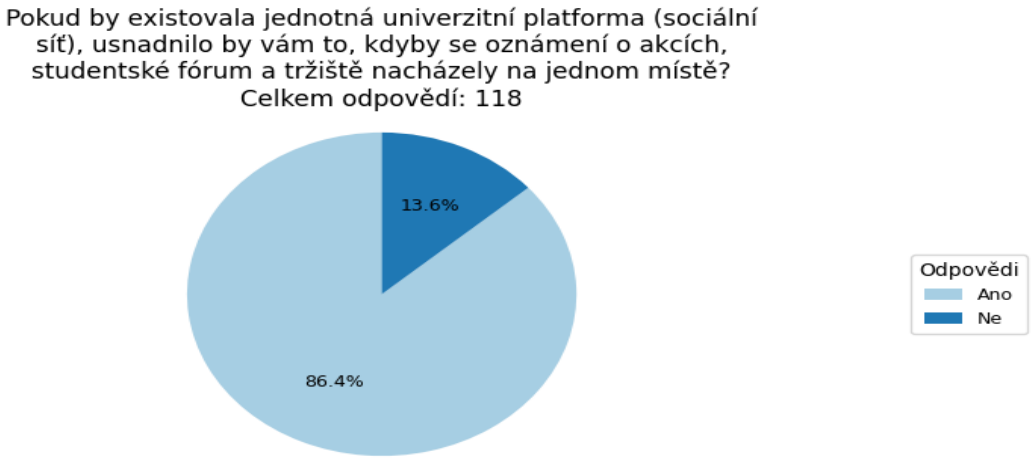



Obrázek *3: Do jaké míry by vám pomohla taková platforma ještě před nástupem?* (vl. zdroj)

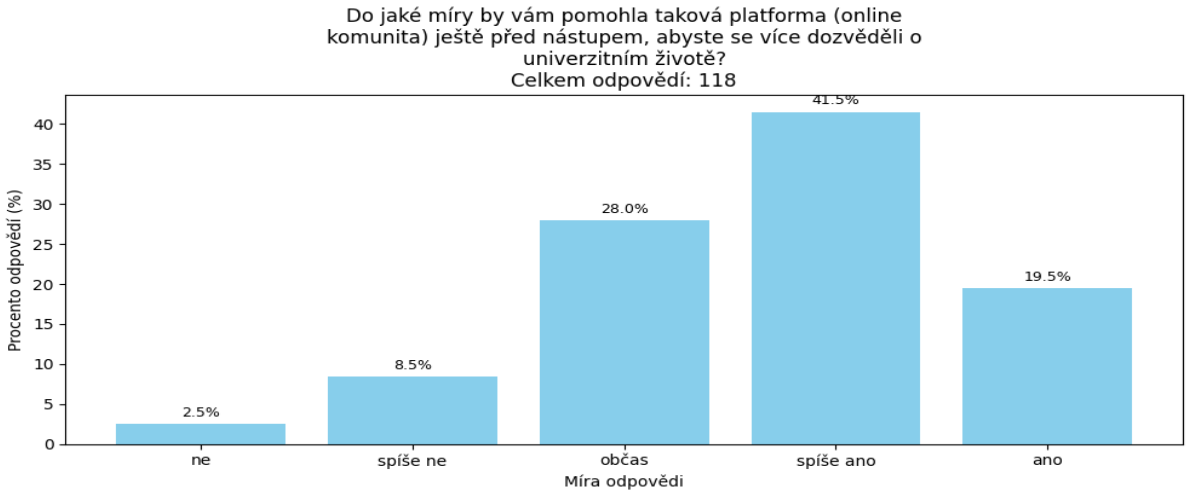



Obrázek *4: Kdybyste se mohli obrátit na další studenty ještě před nástupem...* (vl. zdroj)

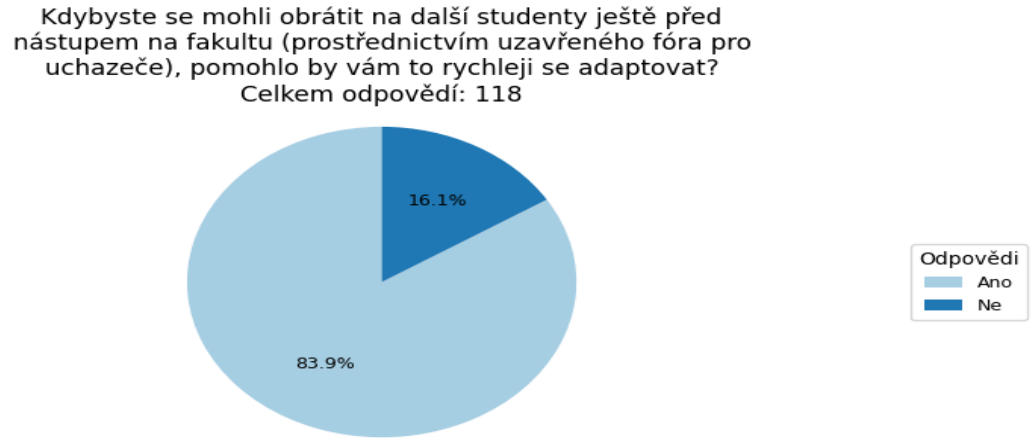
## PŘÍLOHA C*: Grafy z analýzy – Požadavky na aplikaci*



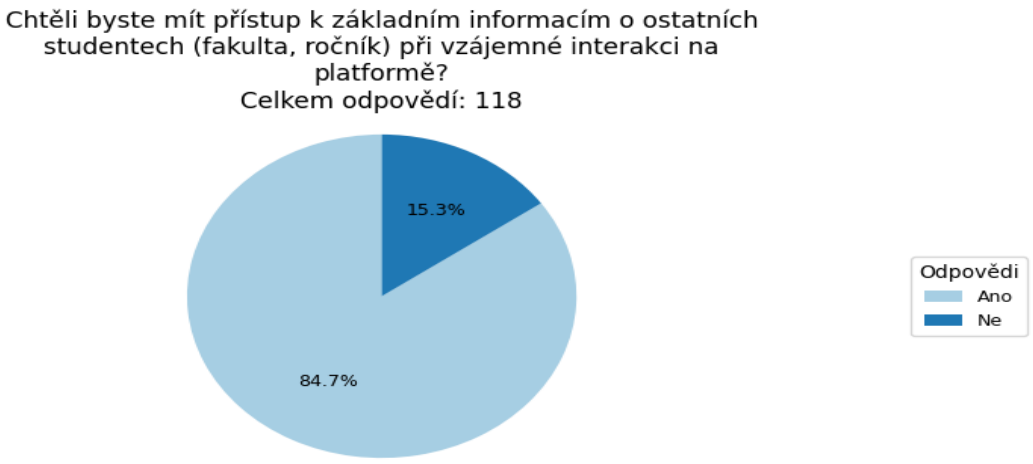

Obrázek *1: Pokud by existovala jednotná univerzitní platforma... usnadnilo by vám to...?* (vl. zdroj)



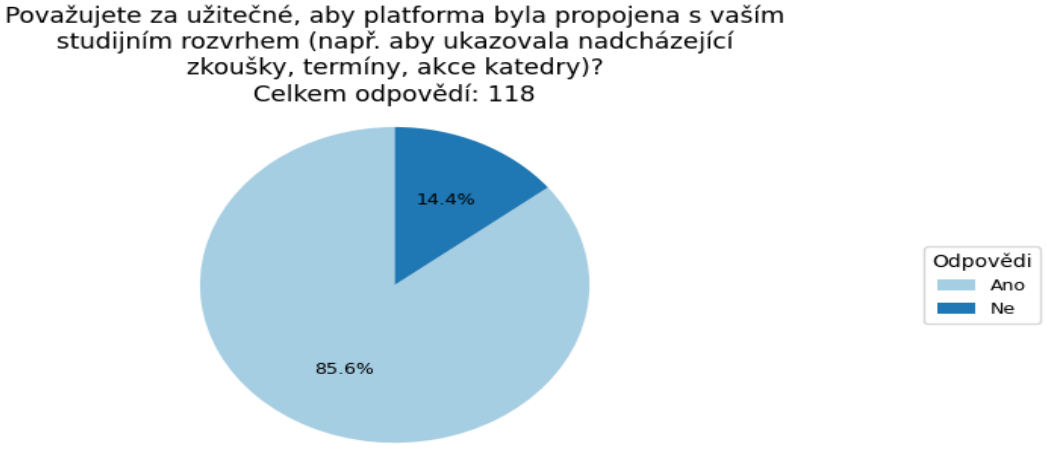

Obrázek *2: Do jaké míry by vám pomohla taková platforma ještě před nástupem?* (vl. zdroj)



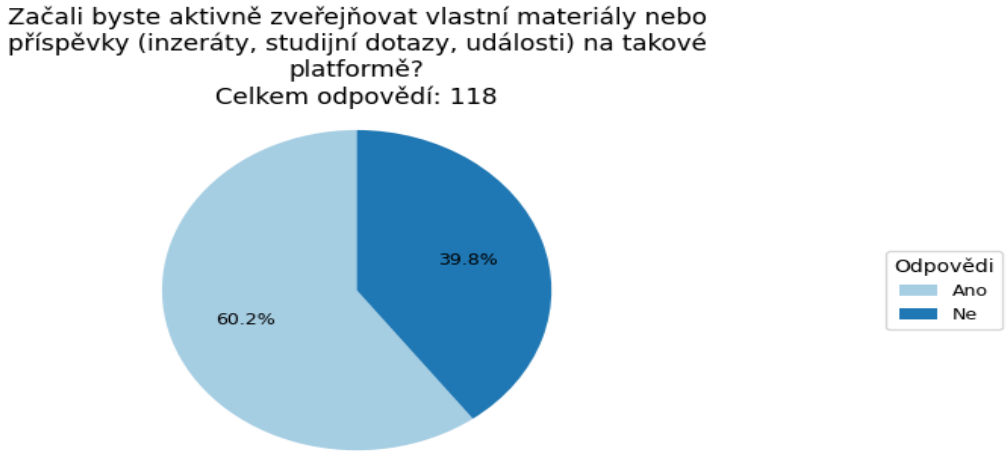

Obrázek *3: Kdybyste se mohli obrátit na další studenty ještě před nástupem...* (vl. zdroj)



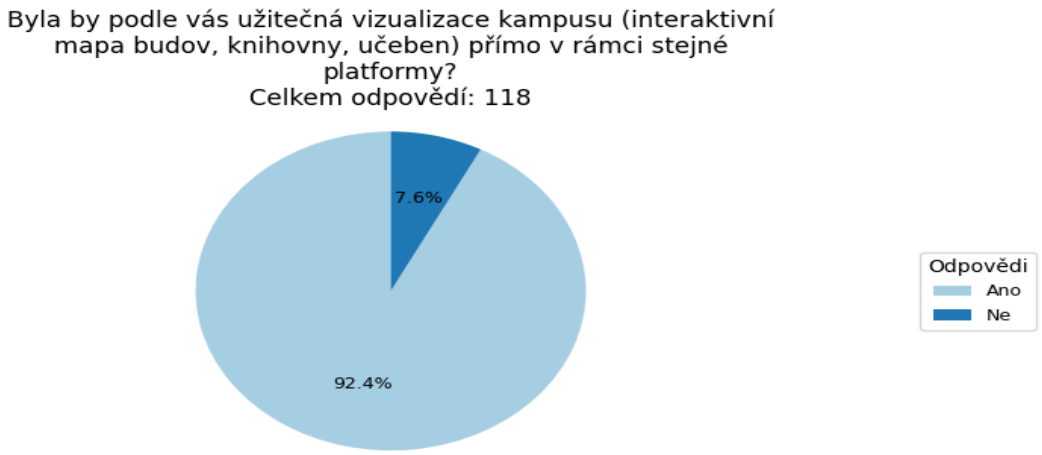

Obrázek 4: *Chtěli byste mít přístup k základním informacím o ostatních studentech...?* (vl. zdroj)



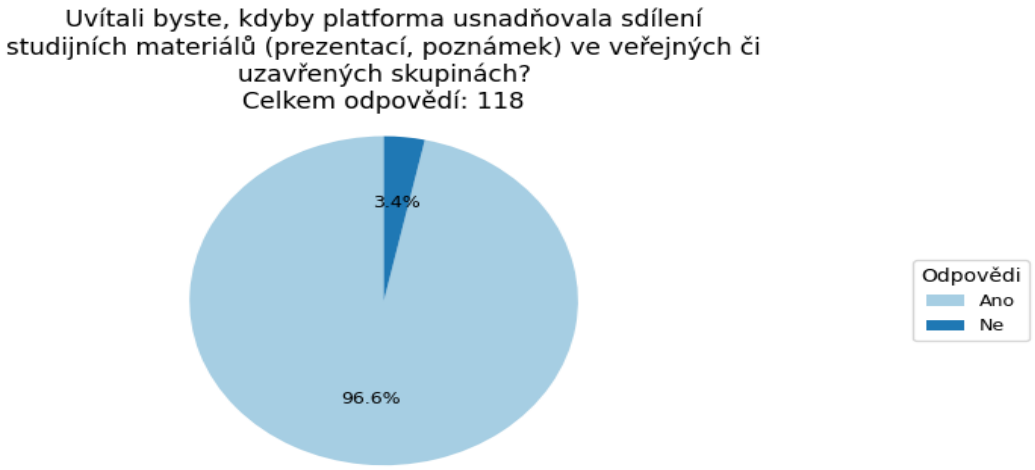

Obrázek 5: *Považujete za užitečné, aby platforma byla propojena s vaším studijním rozvrhem...?* (vl. zdroj)




Obrázek 6: *Začali byste aktivně zveřejňovat vlastní materiály nebo příspěvky...?* (vl. zdroj)

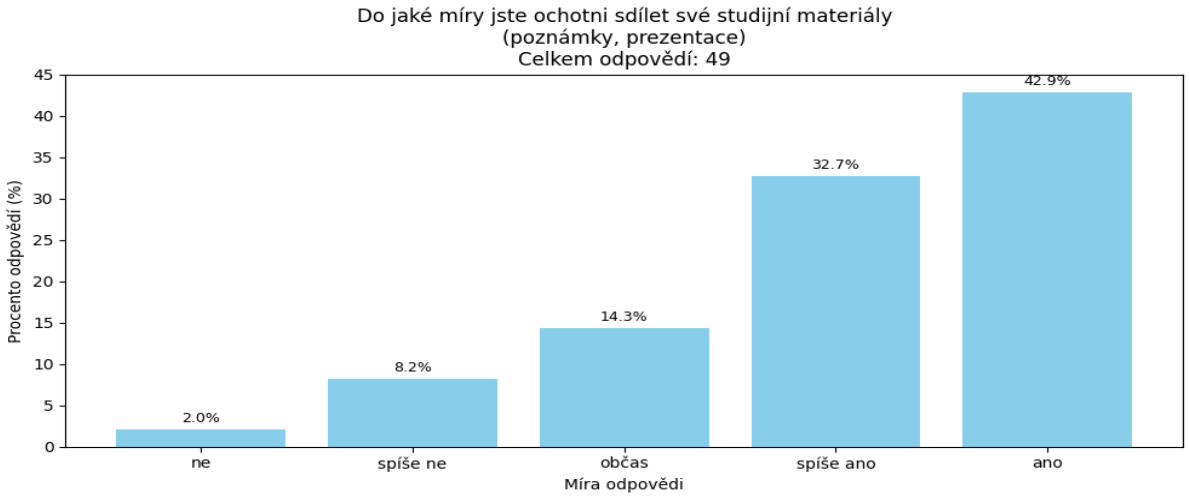



Obrázek 7: *Byla by podle vás užitečná vizualizace kampusu...?* (vl. zdroj)

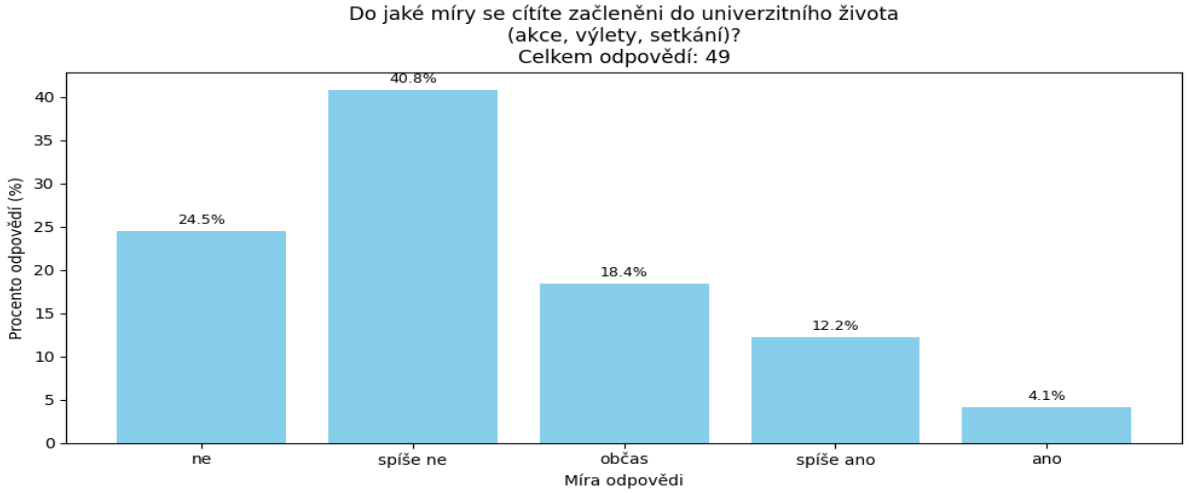



Obrázek 8: *Uvítali byste, kdyby platforma usnadňovala sdílení studijních materiálů...?* (vl. zdroj)

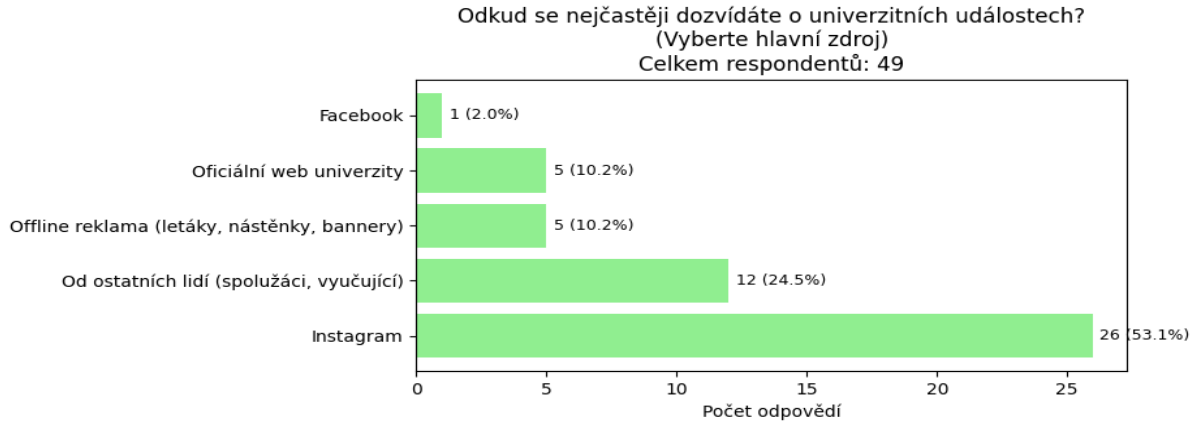
# PŘÍLOHA D. Grafy z analýzy – Odpovědi rusky mluvících studentů



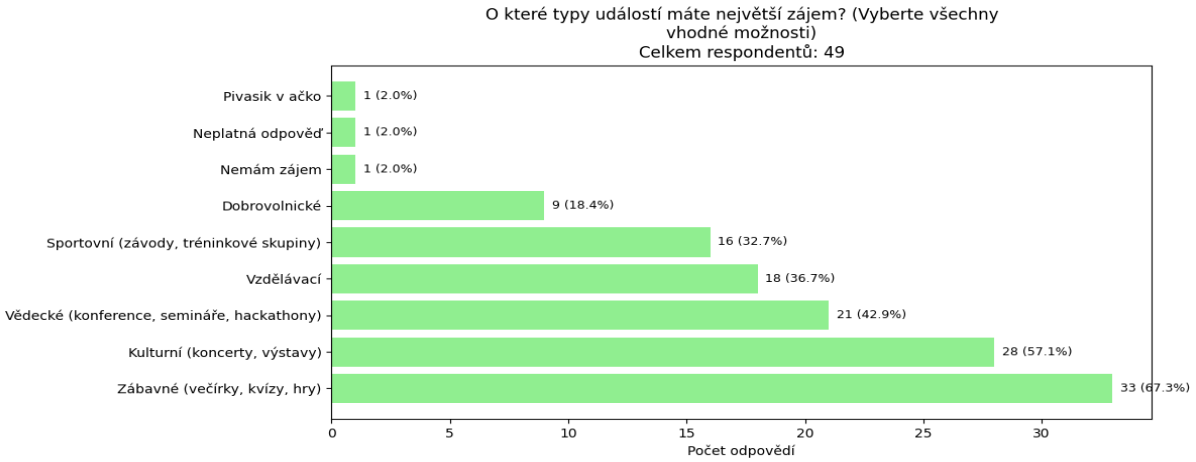

Obrázek *1*: *Do jaké míry jste ochotni sdílet své studijní materiály? ru-studenty* (vl. zdroj)



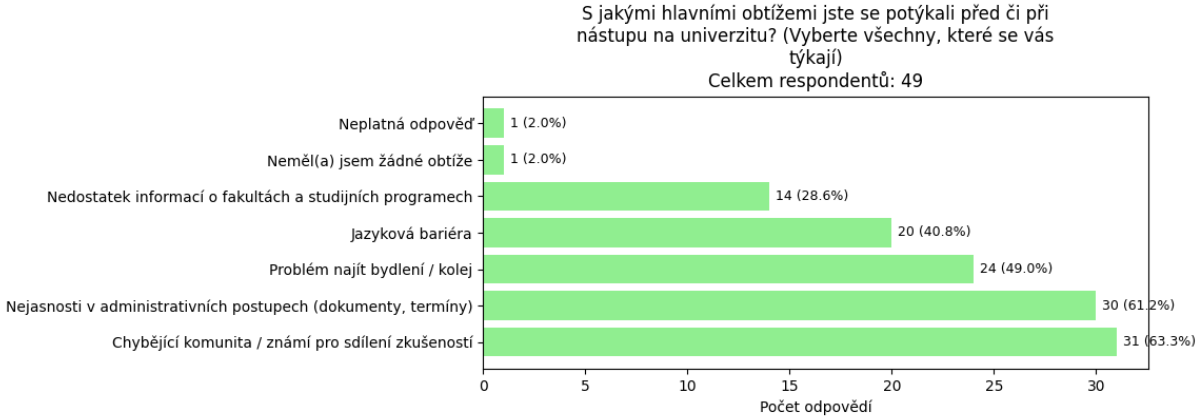

Obrázek *2*: *Do jaké míry se cítíte začleněni do univerzitního života? ru-studenty* (vl. zdroj)



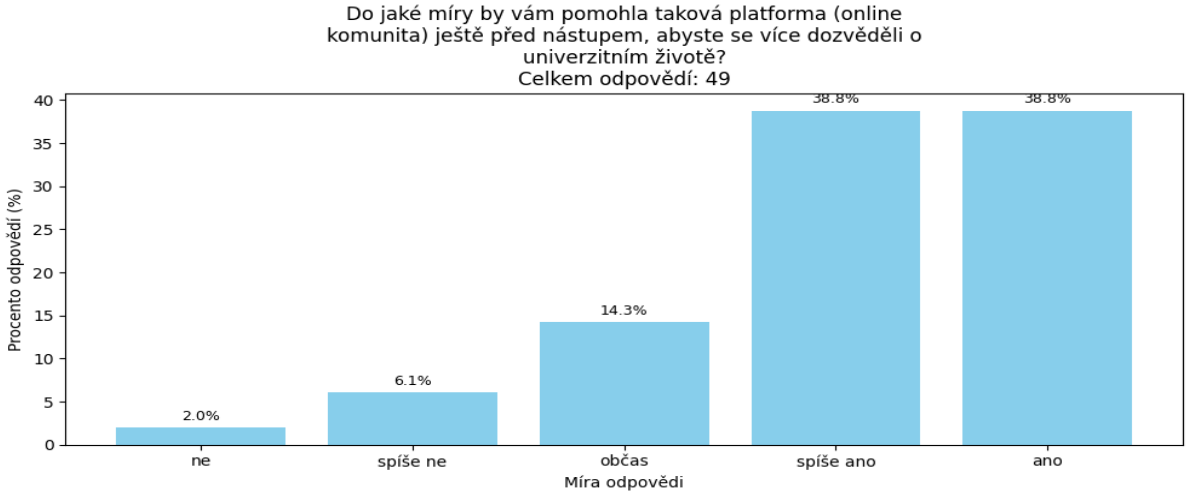

Obrázek *3*: *Odkud se nejčastěji dozvídáte o univerzitních událostech? ru-studenty* (vl. zdroj)




Obrázek 4: *O které typy událostí máte největší zájem? ru-studenty* (vl. zdroj)

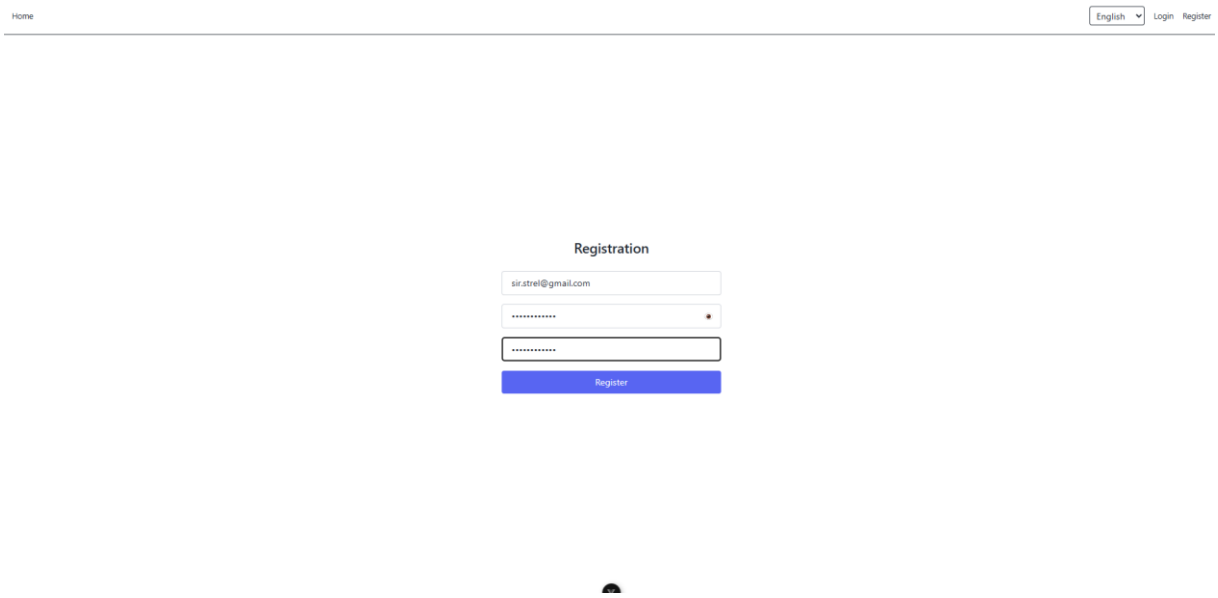



Obrázek 5: *S jakými hlavními obtížemi jste se potýkali před či při nástupu na univerzitu? ru-studenty* (vl. zdroj)

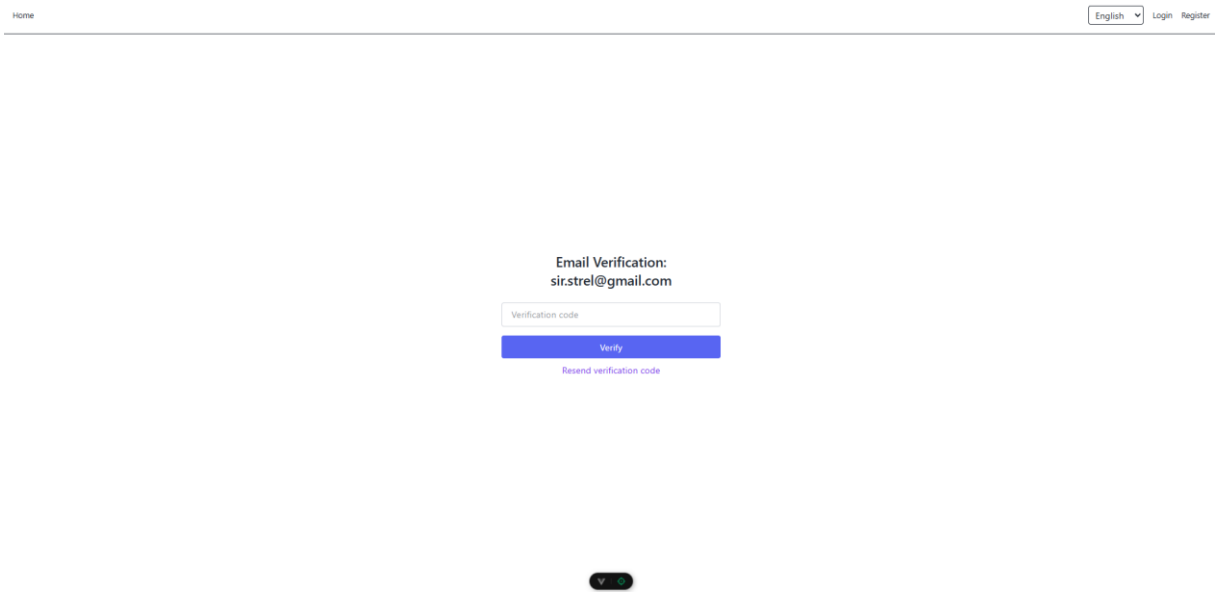



Obrázek 6: *Do jaké míry by vám pomohla taková platforma (online komunita) ještě před nástupem, abyste se více dozvěděli o univerzitním životě? ru-studenty* (vl. zdroj)

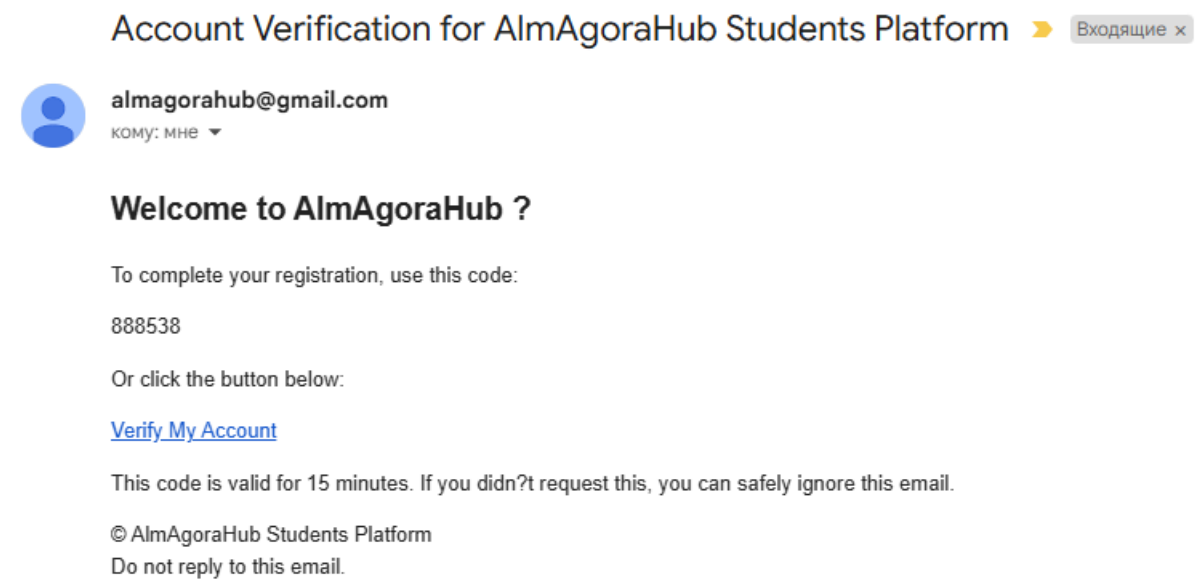
# PŘÍLOHA E: Snímky aplikace



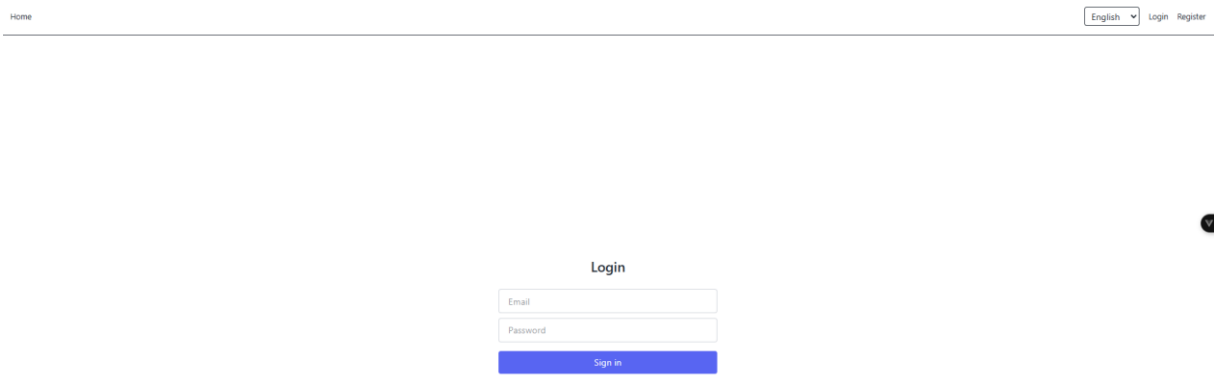

**Obrázek 1:** Stránka registrace nového uživatele (vl. zdroj)



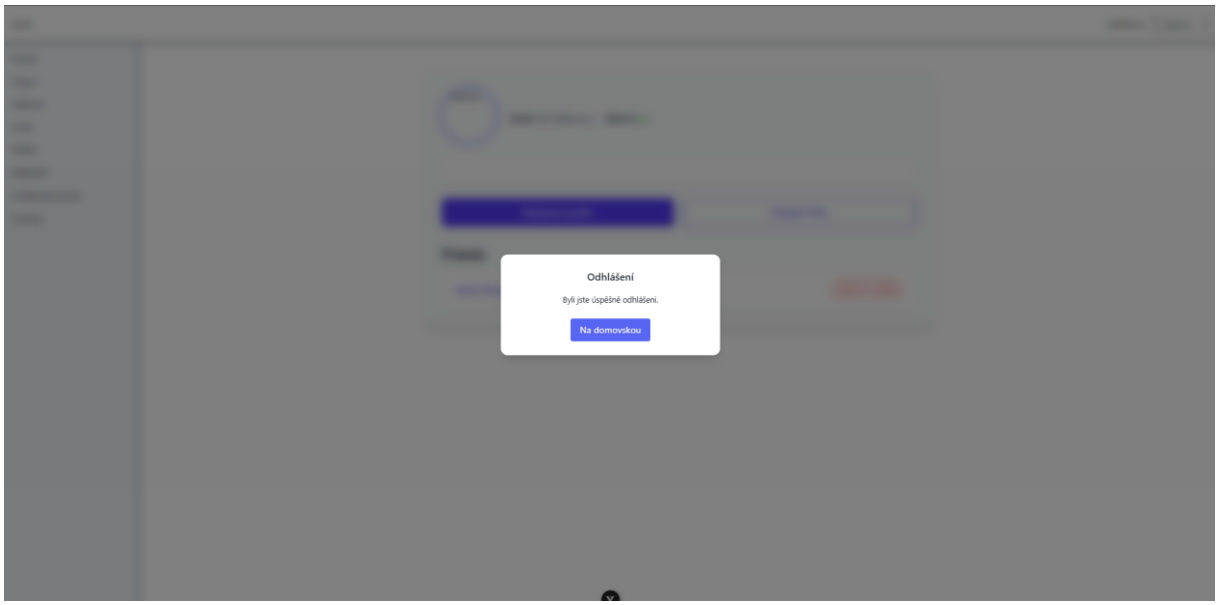

**Obrázek 2:** Stránka ověření e-mailové adresy po registraci (vl. zdroj)



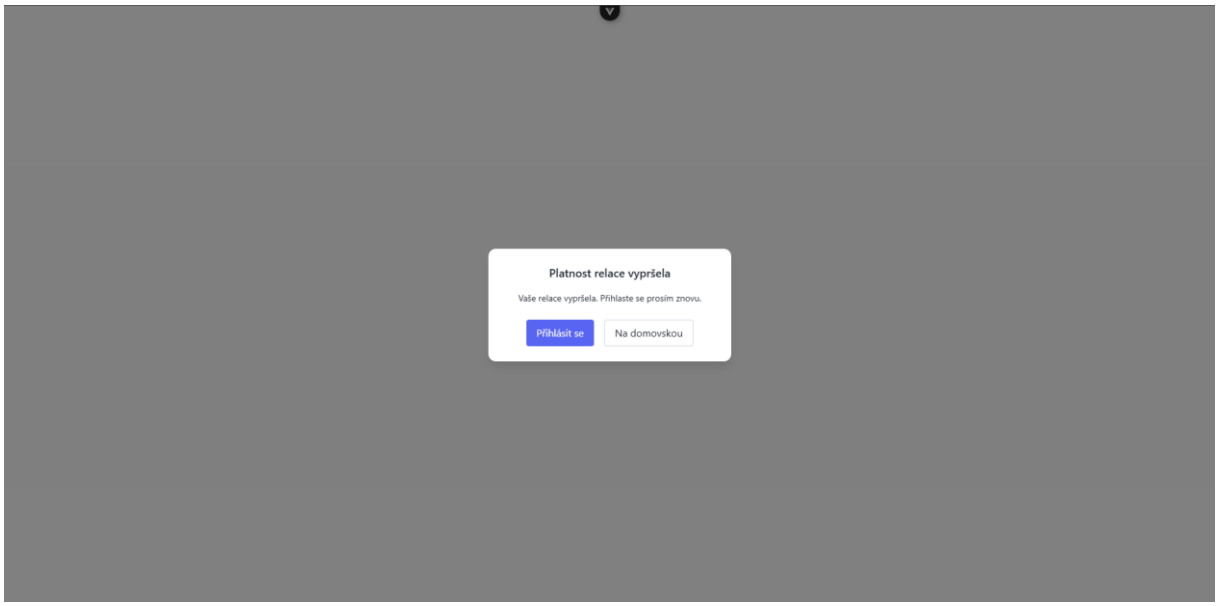

**Obrázek 3:** Ověřovací e-mail s kódem zaslaný uživateli (vl. zdroj)



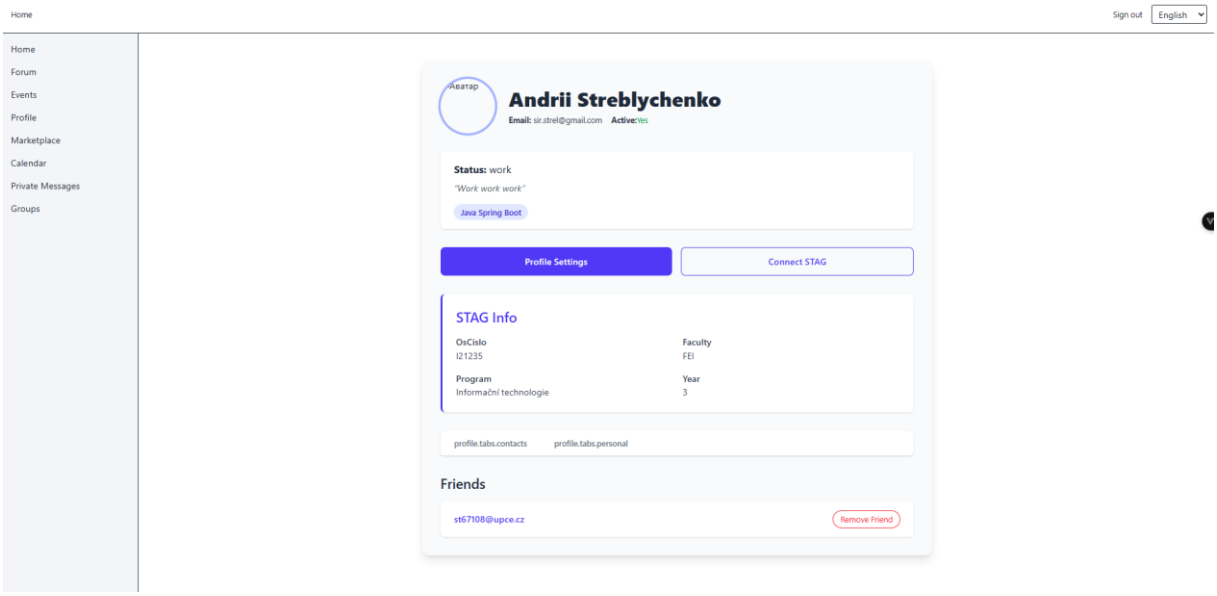

**Obrázek 4:** Stránka pro přihlášení uživatele (vl. zdroj)



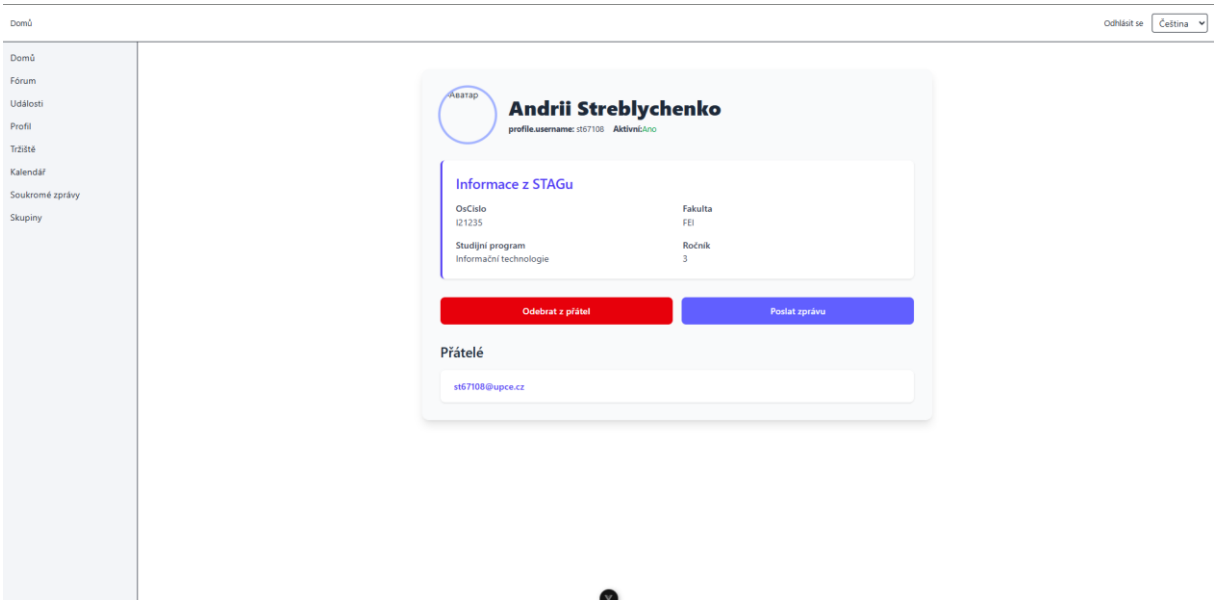

**Obrázek 5:** Modální okno úspěšného odhlášení z aplikace (vl. zdroj)



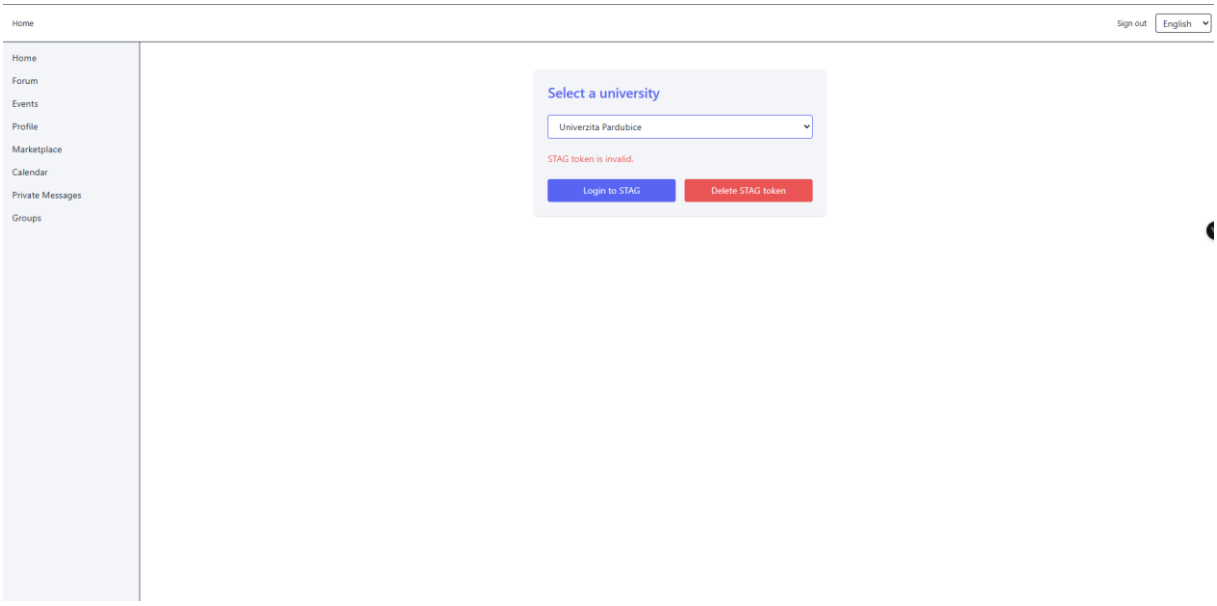

**Obrázek 6:** Modální okno s upozorněním na vypršení tokenu (vl. zdroj)



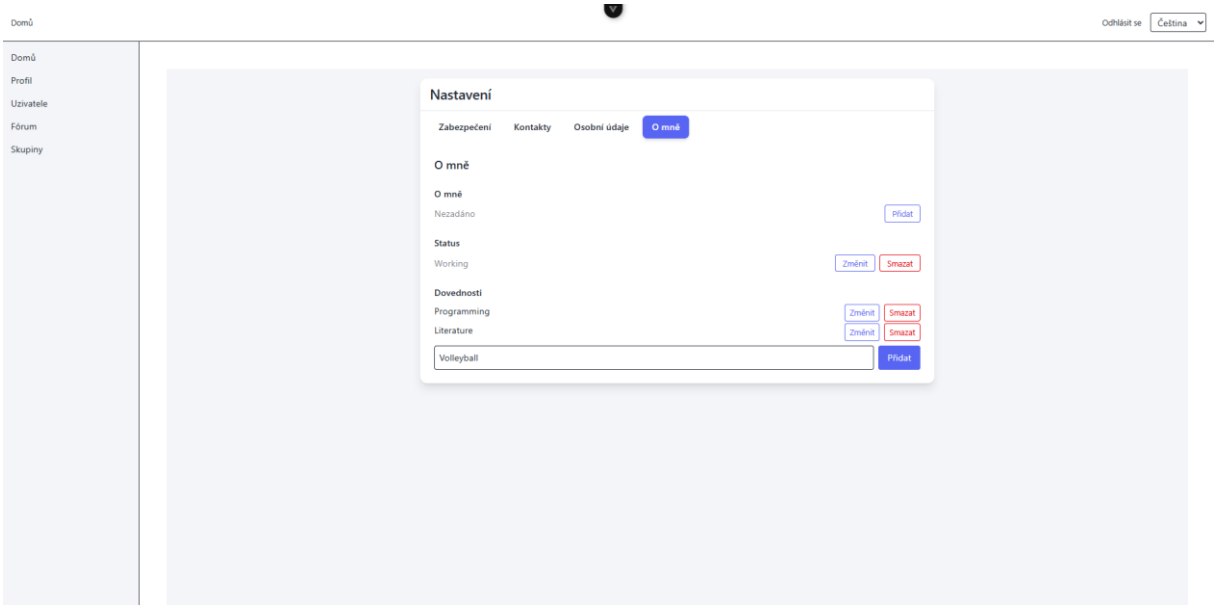

**Obrázek 7:** Osobní profil uživatele propojený se STAGem (vl. zdroj)



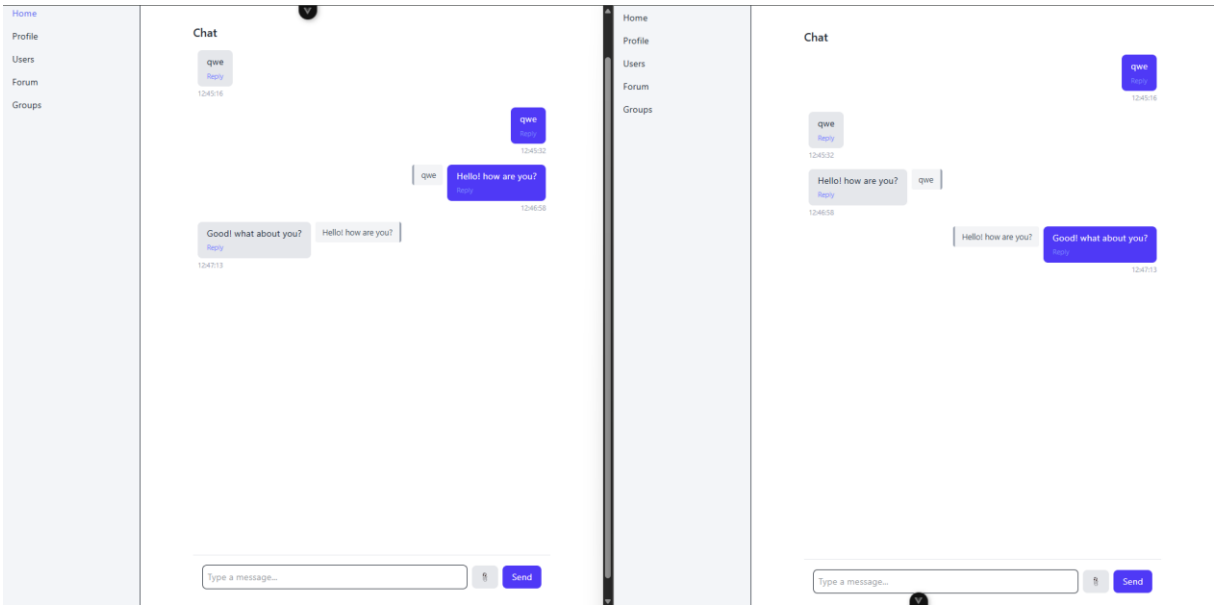

**Obrázek 8:** Veřejný profil jiného uživatele s možností přidání mezi přátele (vl. zdroj)



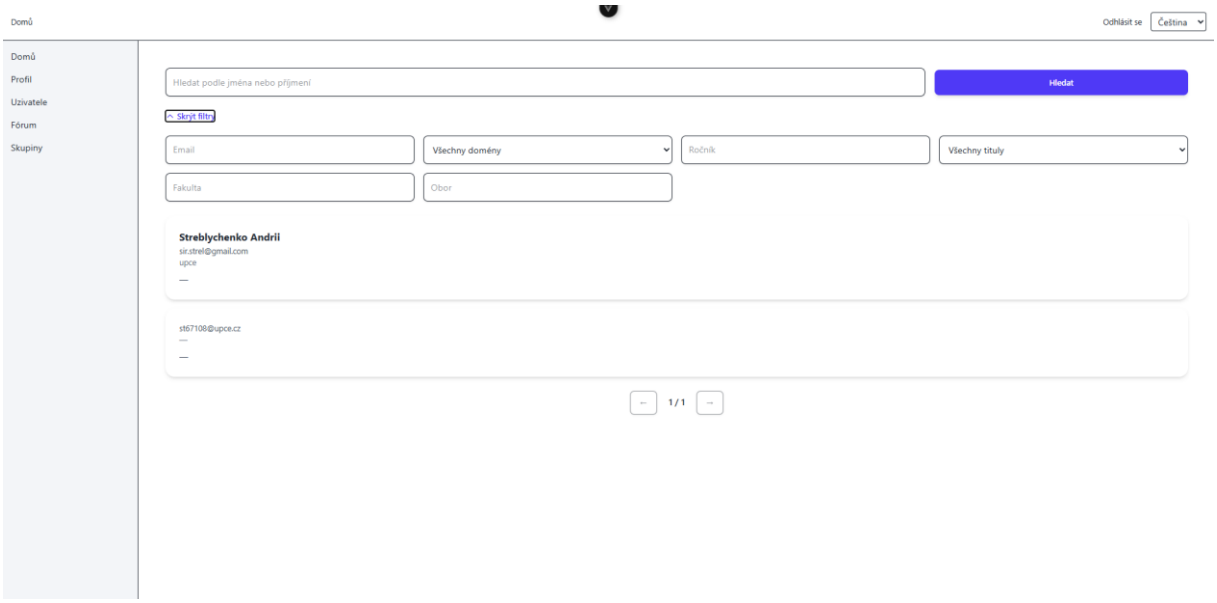

**Obrázek 9:** Připojení ke STAGu – výběr univerzity a ověření platnosti tokenu (vl. zdroj)



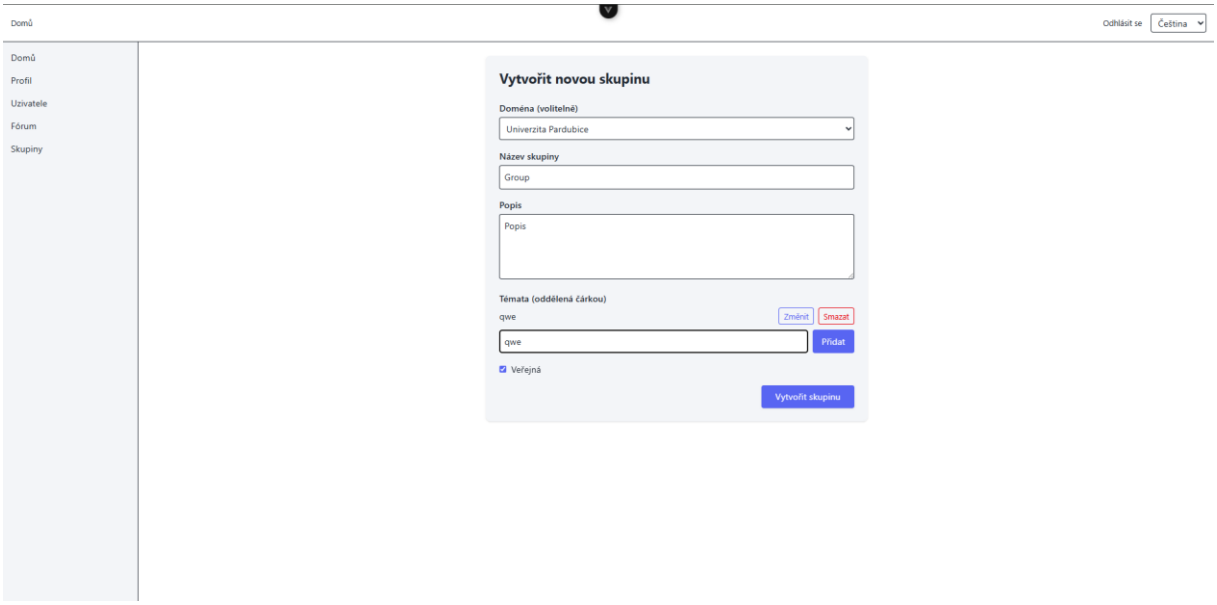

**Obrázek 10:** Stránka nastavení profilu – sekce „O mně“, status a dovednosti (vl. zdroj)



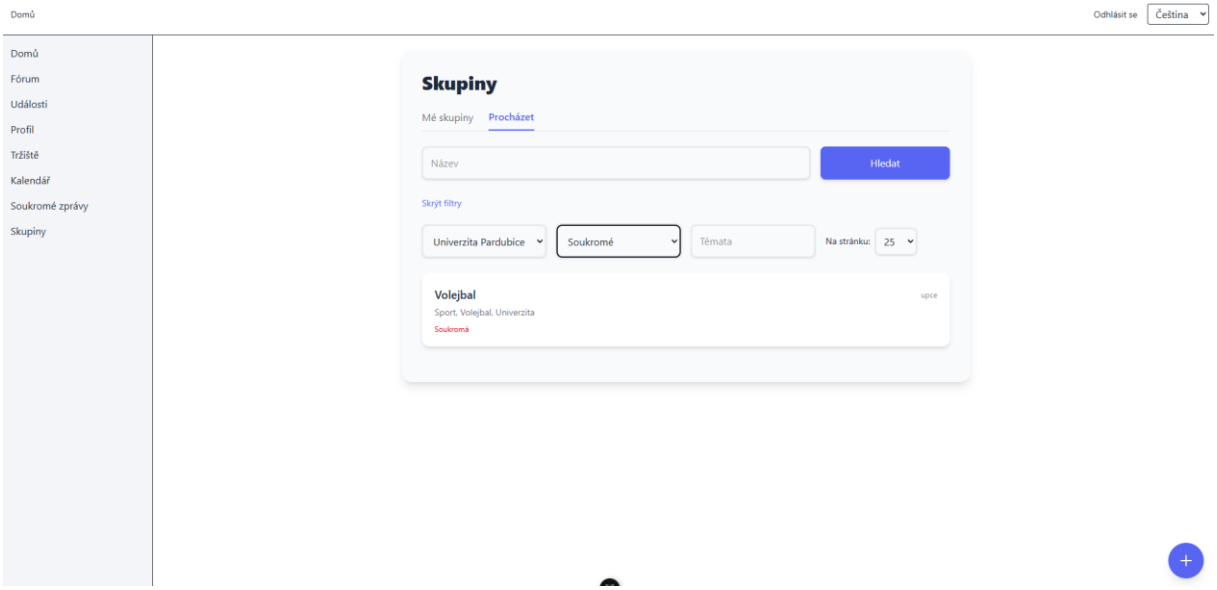

**Obrázek 11:** Modul soukromého chatu mezi uživateli (zobrazení dvou stran konverzace) (vl. zdroj)



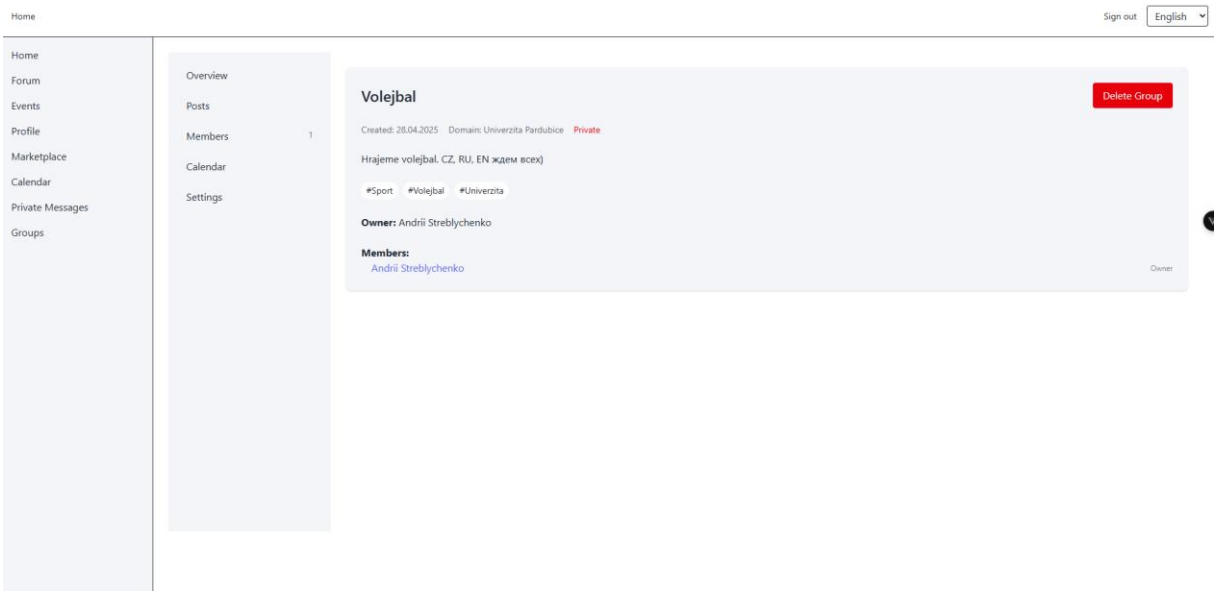

**Obrázek 12:** Vyhledávání uživatelů podle jména, e-mailu, fakulty nebo domény (vl. zdroj)



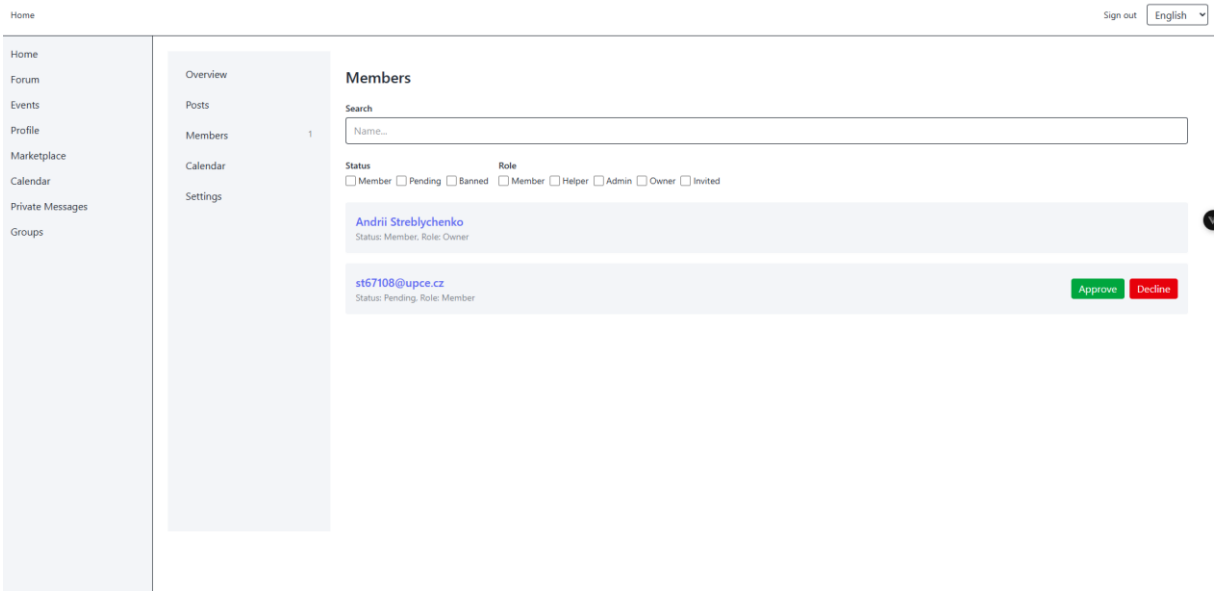

**Obrázek 13:** Formulář pro vytvoření nové skupiny (vl. zdroj)



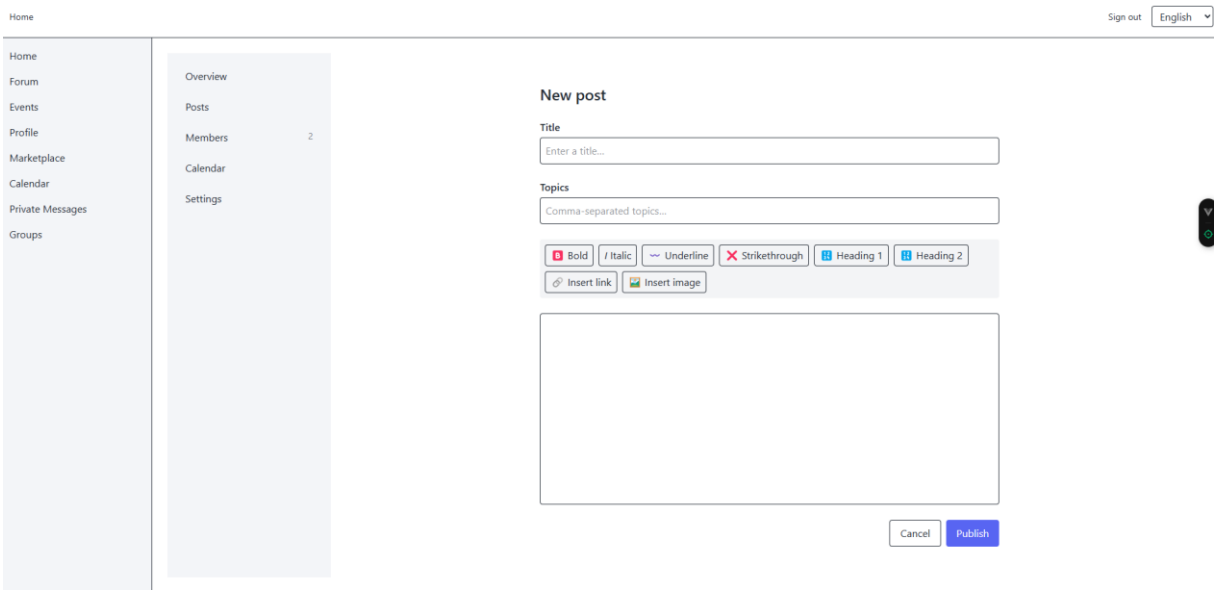

**Obrázek 14:** Prohlížeč skupin s možností filtrování podle domény, typu a tématu (vl. zdroj)



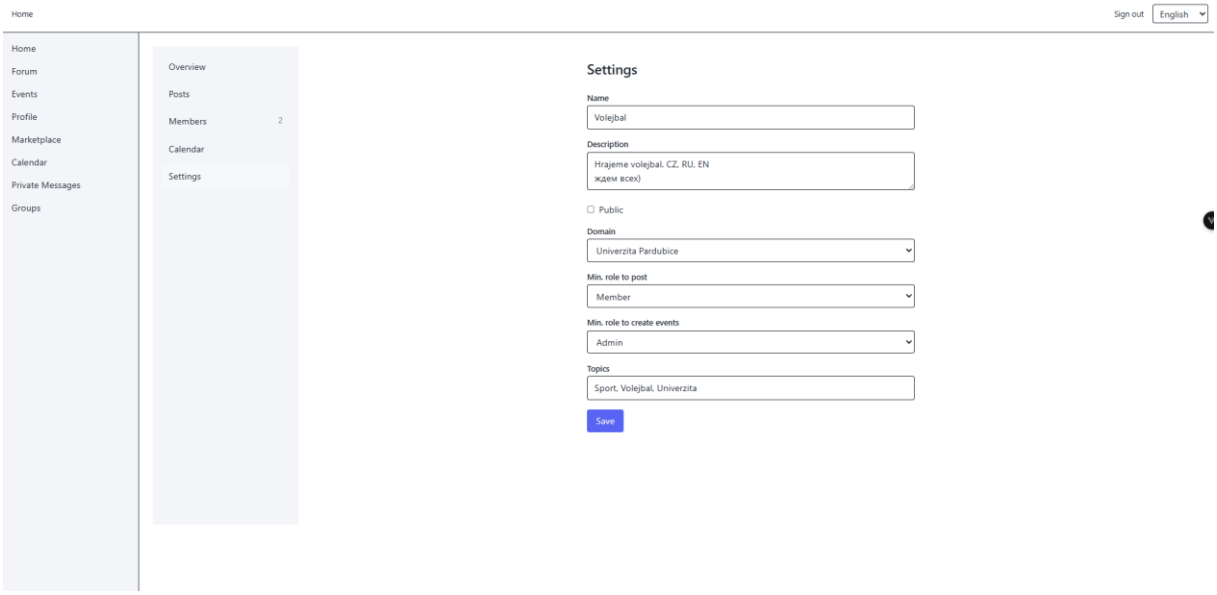

**Obrázek 15:** Přehled skupiny – informace o členství, popis, štítky a vlastník (vl. zdroj)



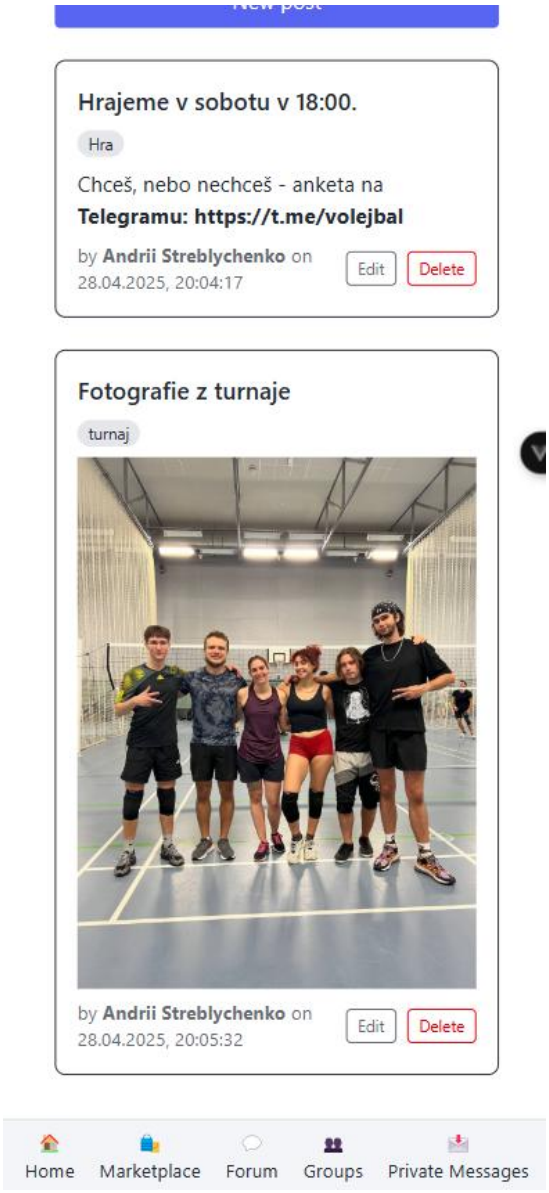

**Obrázek 16:** Správa členů skupiny – schvalování žádostí o vstup (vl. zdroj)



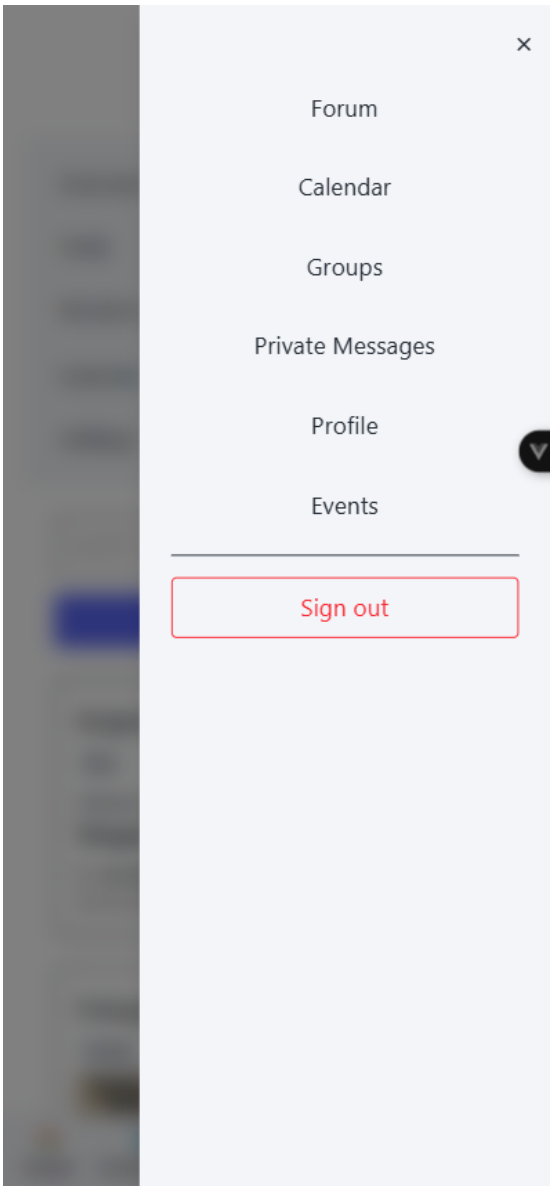

**Obrázek 17:** Vytvoření nového příspěvku ve skupině – editor s podporou formátování (vl. zdroj)



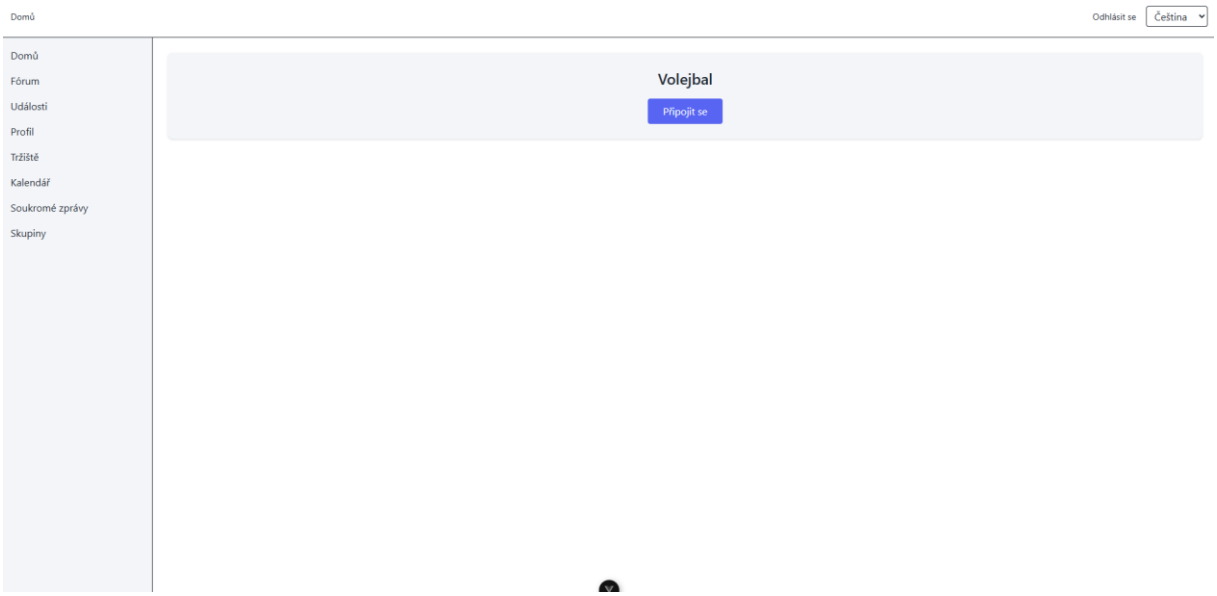

**Obrázek 18:** Nastavení skupiny – viditelnost, role, doména a témata (vl. zdroj)



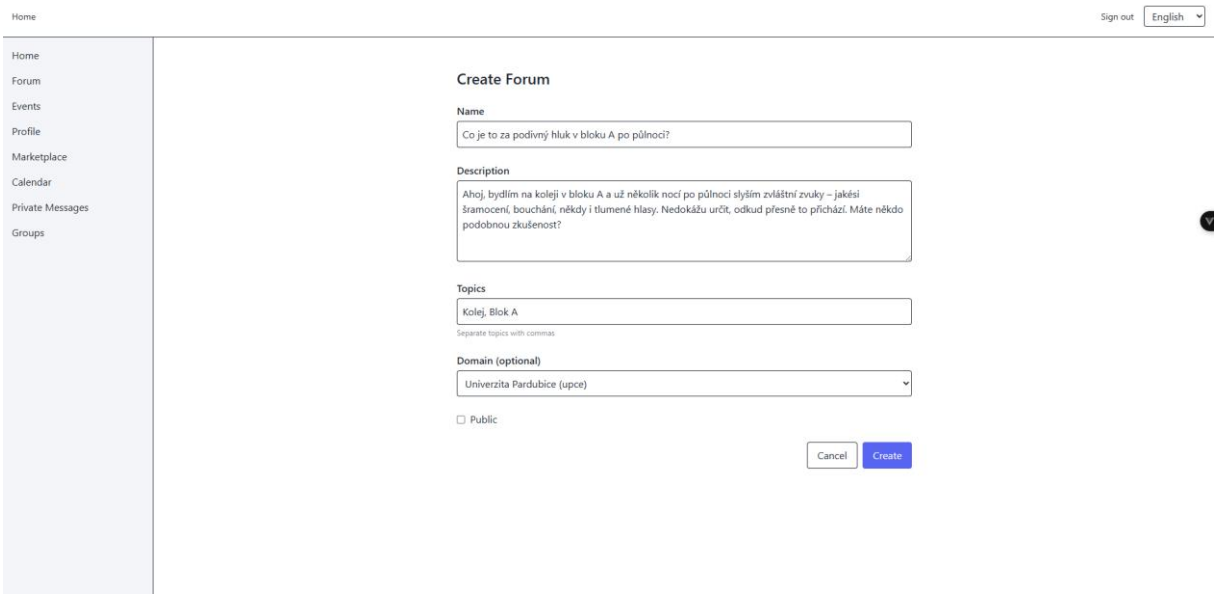

**Obrázek 19:** Mobil. Přehled příspěvků ve skupině – textový i obrazový obsah (vl. zdroj)



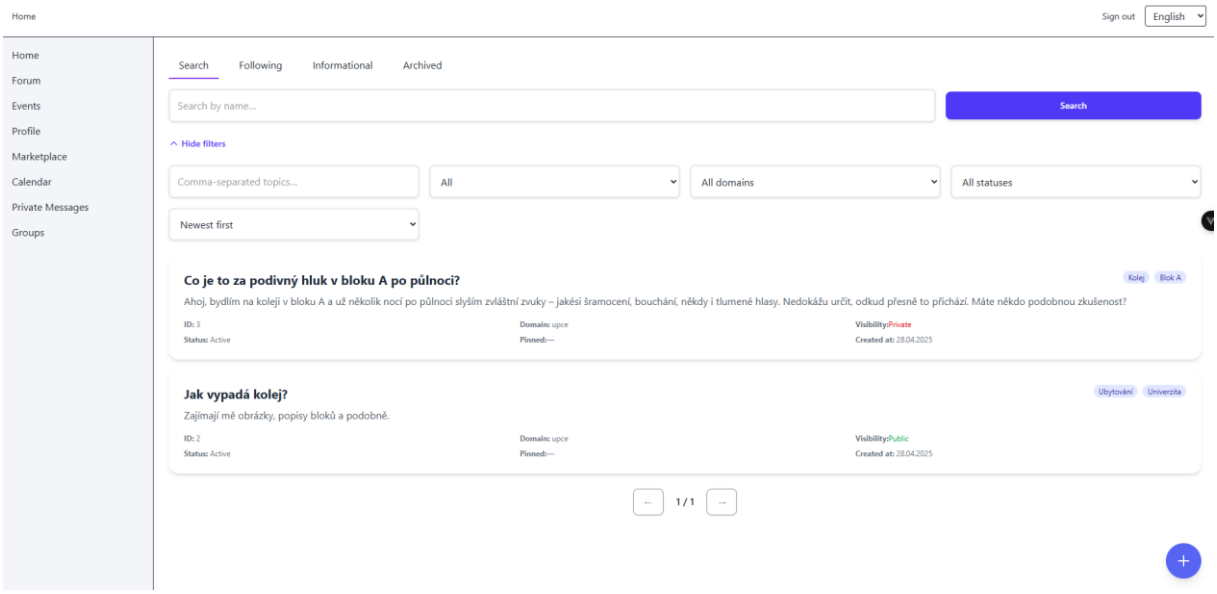

**Obrázek 20:** Mobilní boční menu – navigace mezi sekcemi aplikace (vl. zdroj)



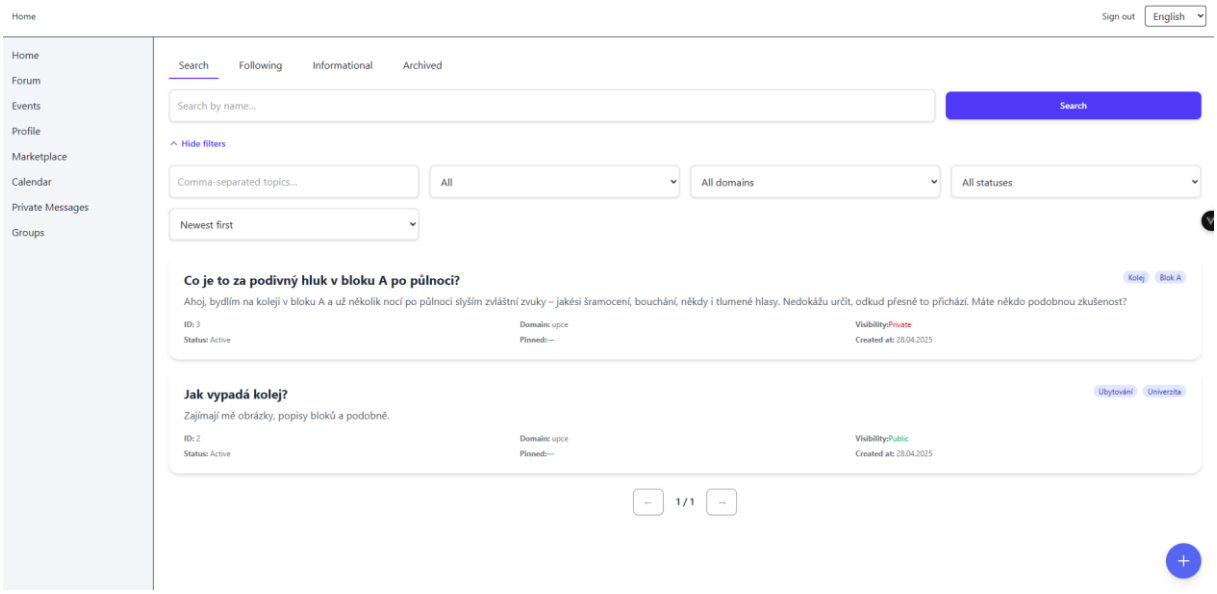

**Obrázek 21:** Detail soukromé skupiny – možnost připojení (vl. zdroj)



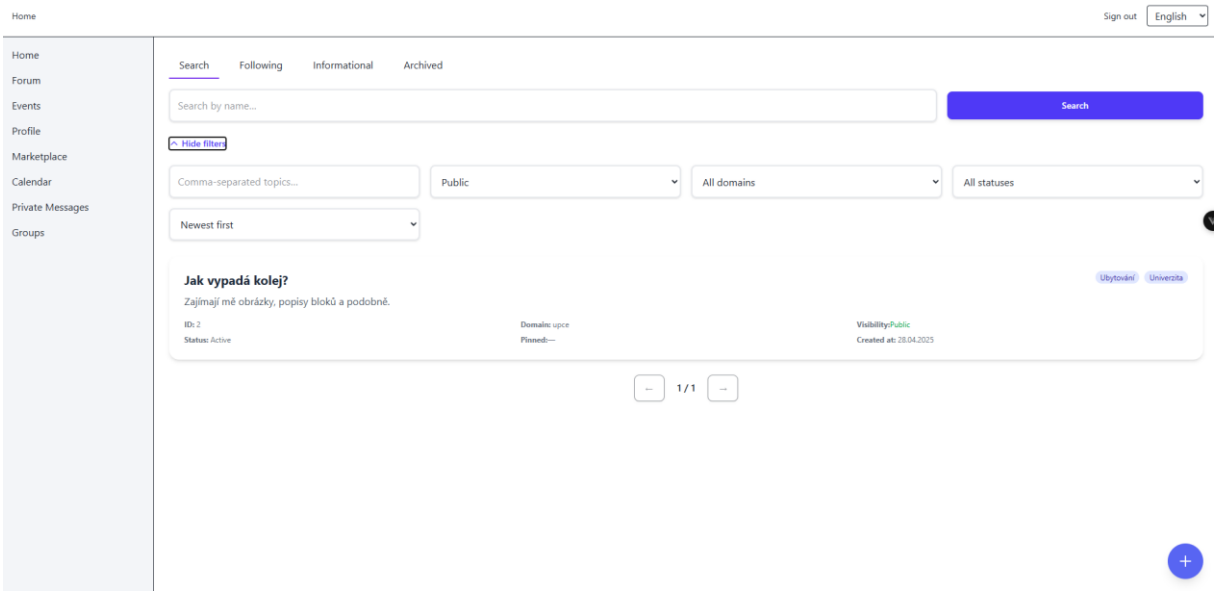

**Obrázek 22:** Formulář pro vytvoření nového fóra s tématem a doménou (vl. zdroj)




**Obrázek 23:** Vyhledávání v diskusním fóru – seznam aktuálních vláken (vl. zdroj)

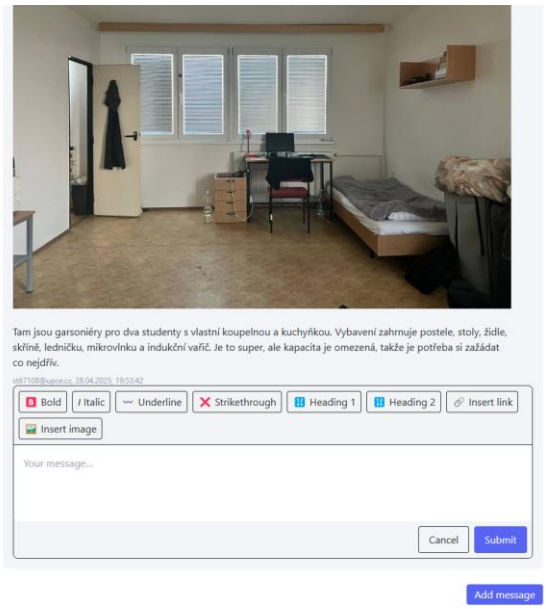



**Obrázek 24:** Vyhledávání ve fóru s aktivním filtrem pro veřejné příspěvky (vl. zdroj)

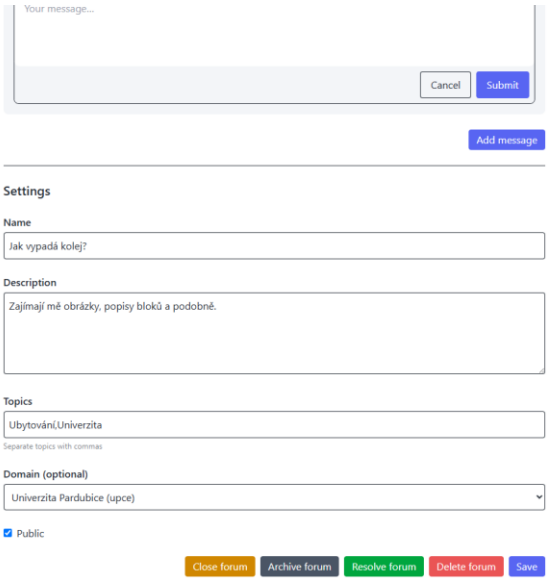



**Obrázek 25:** Vyhledávání veřejných vláken ve fóru podle domény a stavu (vl. zdroj)

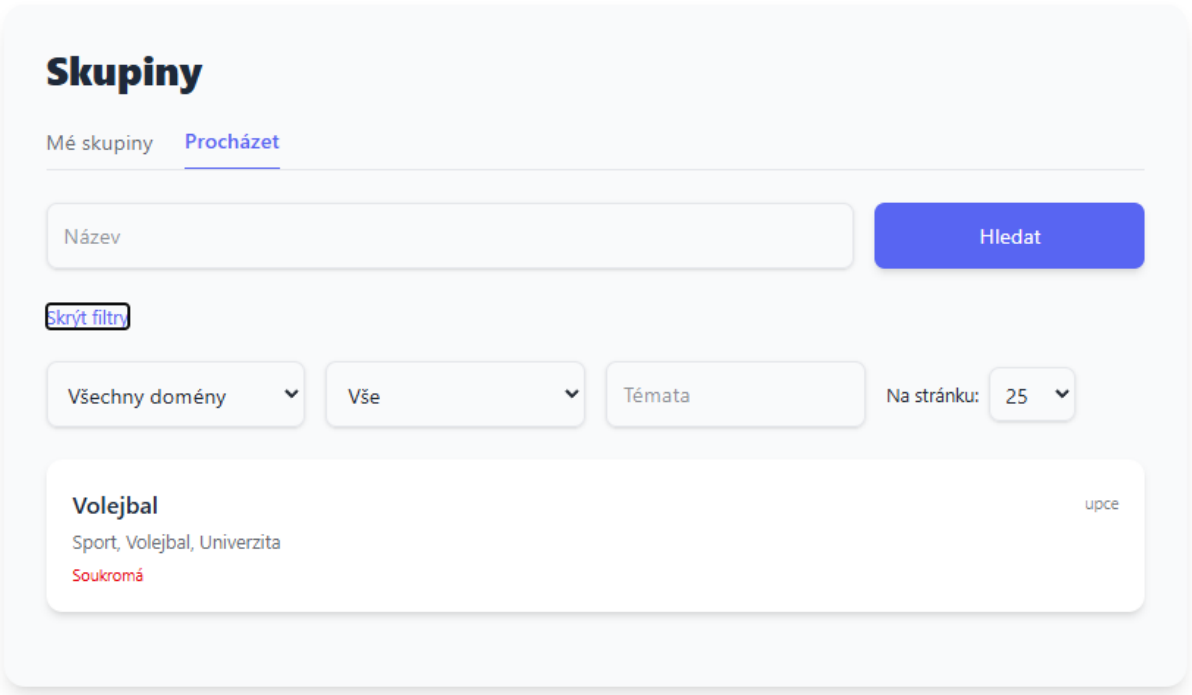

**Obrázek 26:** Detail diskusního vlákna – zobrazení odpovědí a interakcí mezi uživateli (vl. zdroj)



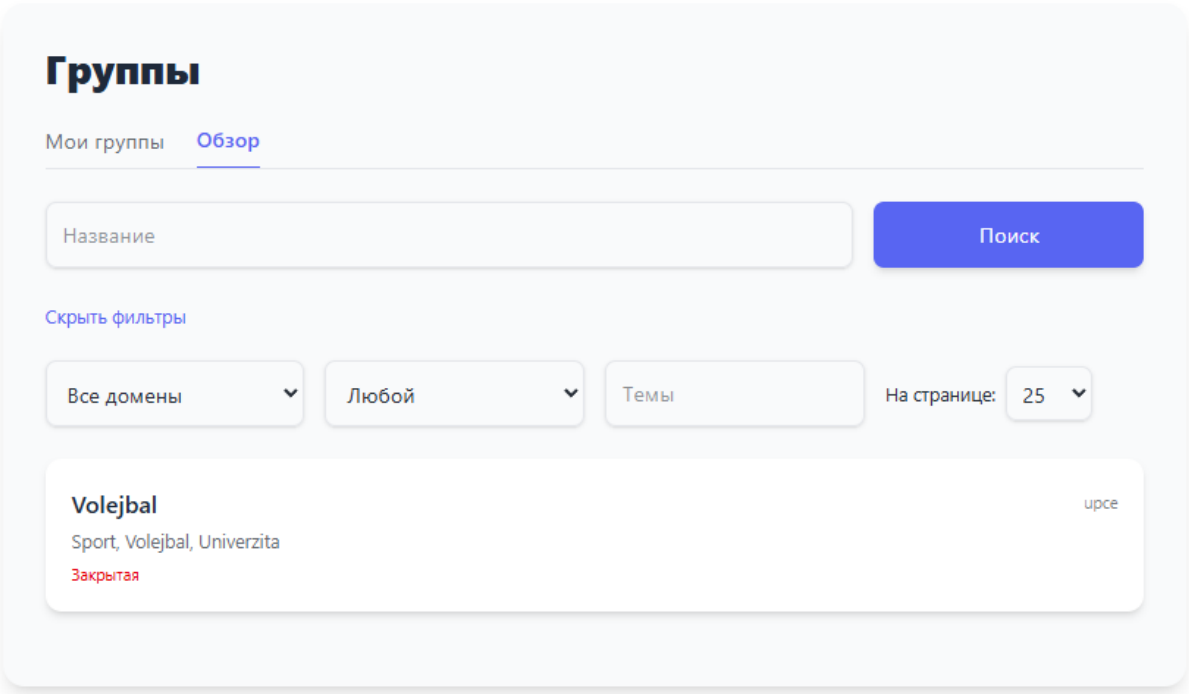

**Obrázek 27:** Odpověď ve fóru s vloženým obrázkem pokoje na koleji (vl. zdroj)



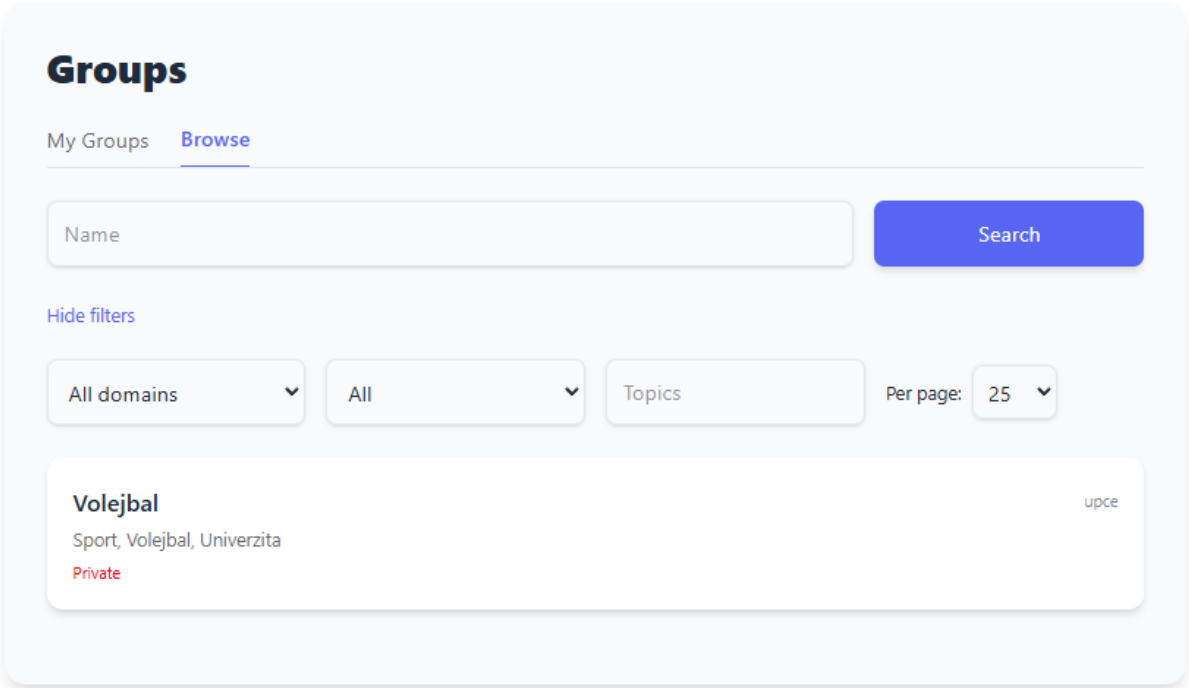

**Obrázek 28:** Nastavení fóra – úprava popisu, domény a viditelnosti (vl. zdroj)




**Obrázek 29:** Seznam skupin – česká lokalizace (vl. zdroj)



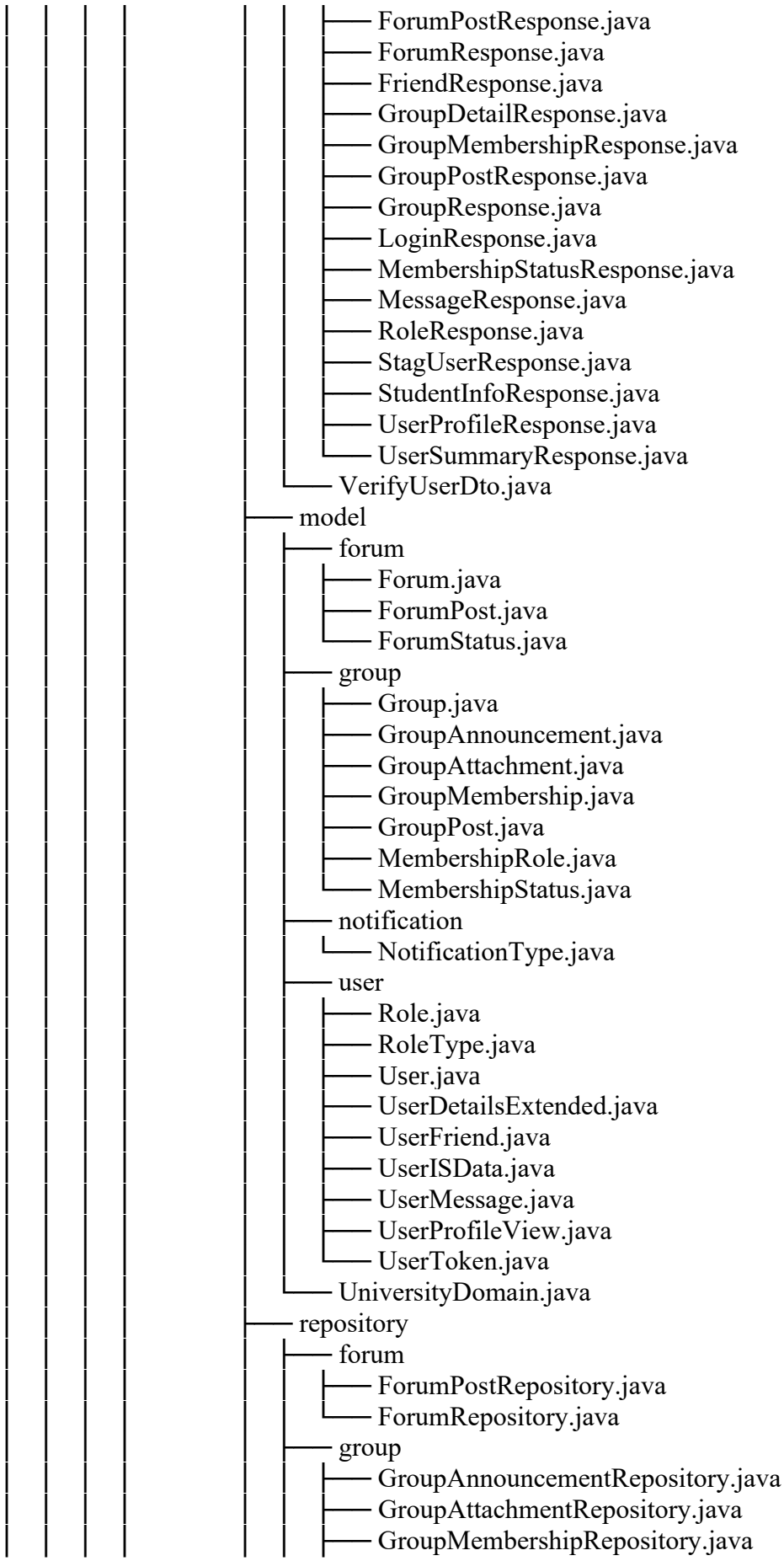

**Obrázek 30:** Seznam skupin – ruská lokalizace (vl. zdroj)

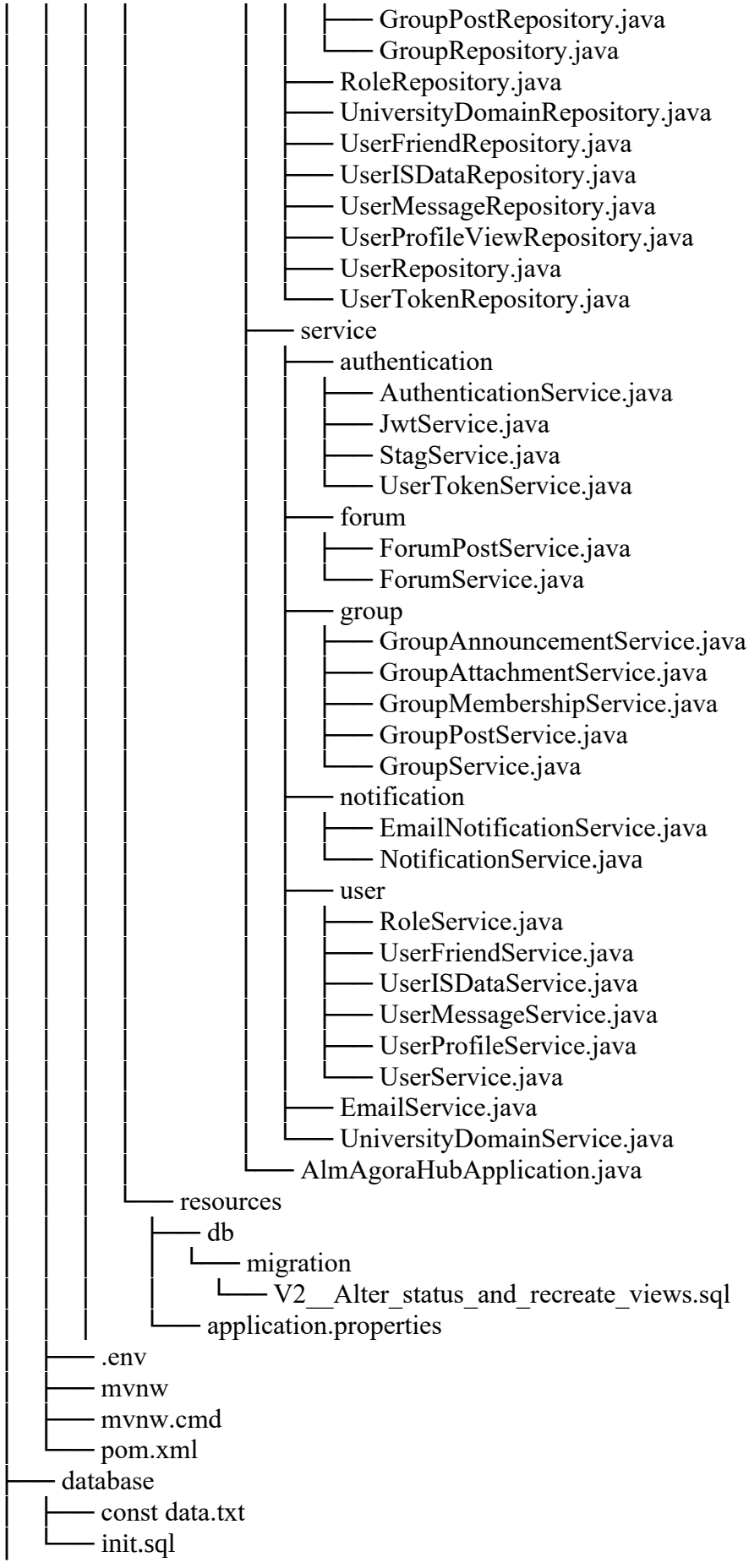



**Obrázek 31:** Seznam skupin – anglická lokalizace (vl. zdroj)

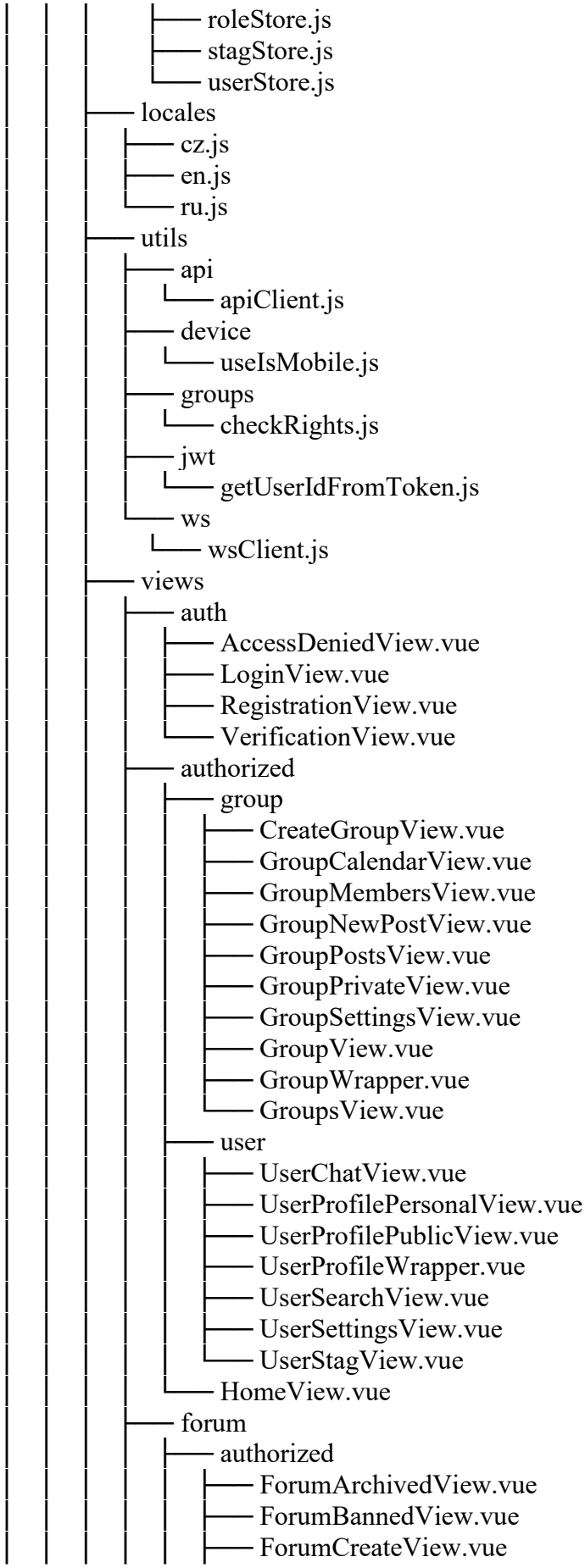
## PŘÍLOHA F*:* Zdroje aplikace

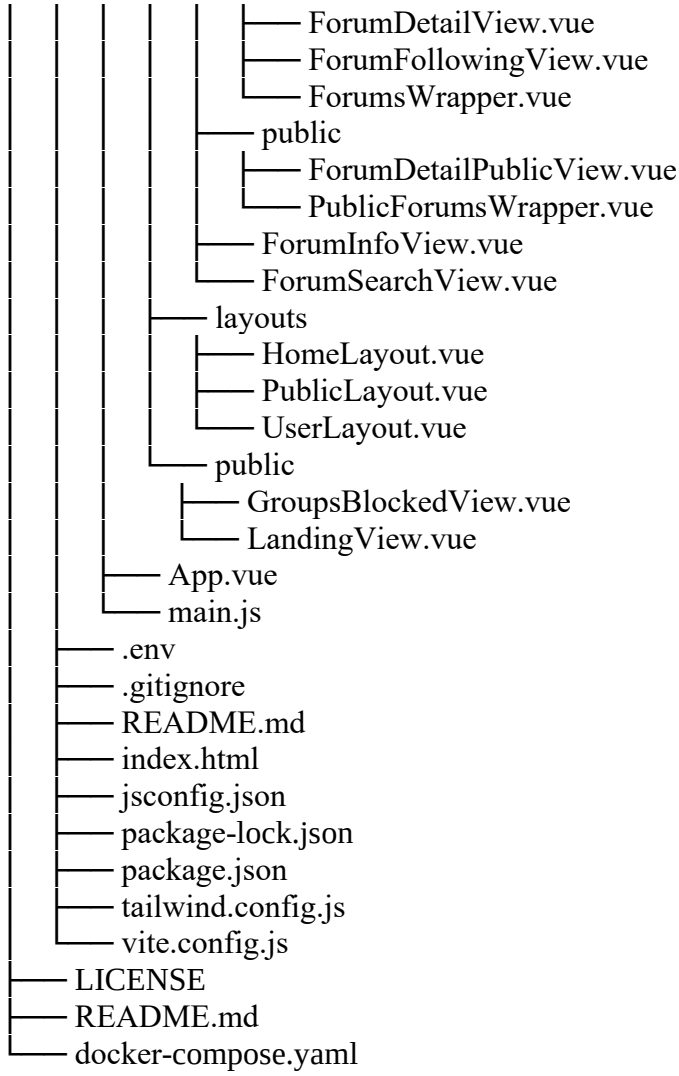
```
Student-Community-Platform/
├── analyse
│   ├── CZ (Odpovedy).xlsx
│   ├── RU (Odpovedy).xlsx
│   ├── graph-ru.py
│   └── graph.py
├── backend
│   ├── src
│   │   ├── main
│   │   │   ├── java
│   │   │   │   └── com
│   │   │   │       └── vojavy
│   │   │   │           └── AlmAgoraHub
│   │   │   │               ├── config
│   │   │   │               │   ├── AppConfiguration.java
│   │   │   │               │   ├── AsyncConfig.java
│   │   │   │               │   ├── DataInitializer.java
│   │   │   │               │   ├── EmailConfiguration.java
│   │   │   │               │   ├── JwtAuthFilter.java
│   │   │   │               │   ├── JwtChannelInterceptor.java
│   │   │   │               │   ├── SecurityConfiguration.java
│   │   │   │               │   ├── StagConfiguration.java
│   │   │   │               │   ├── StagProperties.java
│   │   │   │               │   └── WebSocketConfig.java
│   │   │   │               ├── controller
│   │   │   │               │   ├── AuthenticationController.java
│   │   │   │               │   ├── ChatController.java
│   │   │   │               │   ├── ForumController.java
│   │   │   │               │   ├── GroupController.java
│   │   │   │               │   ├── RolesController.java
│   │   │   │               │   ├── StagController.java
│   │   │   │               │   ├── UniversityDomainController.java
│   │   │   │               │   ├── UserController.java
│   │   │   │               │   └── UserFriendController.java
│   │   │   │               ├── dto
│   │   │   │               │   ├── requests
│   │   │   │               │   │   ├── CreateForumPostRequest.java
│   │   │   │               │   │   ├── CreateForumRequest.java
│   │   │   │               │   │   ├── CreateGroupRequest.java
│   │   │   │               │   │   ├── GroupPostRequest.java
│   │   │   │               │   │   ├── GroupSettingsRequest.java
│   │   │   │               │   │   ├── HistoryRequest.java
│   │   │   │               │   │   ├── LoginUserRequest.java
│   │   │   │               │   │   ├── RegisterUserRequest.java
│   │   │   │               │   │   ├── SendMessageRequest.java
│   │   │   │               │   │   └── UpdateUserRequest.java
│   │   │   │               │   └── responses
│   │   │   │               │       ├── ForumDetailResponse.java
```


ForumPostResponse.java
ForumResponse.java
FriendResponse.java
GroupDetailResponse.java
GroupMembershipResponse.java
GroupPostResponse.java
GroupResponse.java
LoginResponse.java
MembershipStatusResponse.java
MessageResponse.java
RoleResponse.java
StagUserResponse.java
StudentInfoResponse.java
UserProfileResponse.java
UserSummaryResponse.java
VerifyUserDto.java
model
forum
Forum.java
ForumPost.java
ForumStatus.java
group
Group.java
GroupAnnouncement.java
GroupAttachment.java
GroupMembership.java
GroupPost.java
MembershipRole.java
MembershipStatus.java
notification
NotificationType.java
user
Role.java
RoleType.java
User.java
UserDetailsExtended.java
UserFriend.java
UserISData.java
UserMessage.java
UserProfileView.java
UserToken.java
UniversityDomain.java
repository
forum
ForumPostRepository.java
ForumRepository.java
group
GroupAnnouncementRepository.java
GroupAttachmentRepository.java
GroupMembershipRepository.java


GroupPostRepository.java
GroupRepository.java
RoleRepository.java
UniversityDomainRepository.java
UserFriendRepository.java
UserISDataRepository.java
UserMessageRepository.java
UserProfileViewRepository.java
UserRepository.java
UserTokenRepository.java
service
authentication
AuthenticationService.java
JwtService.java
StagService.java
UserTokenService.java
forum
ForumPostService.java
ForumService.java
group
GroupAnnouncementService.java
GroupAttachmentService.java
GroupMembershipService.java
GroupPostService.java
GroupService.java
notification
EmailNotificationService.java
NotificationService.java
user
RoleService.java
UserFriendService.java
UserISDataService.java
UserMessageService.java
UserProfileService.java
UserService.java
EmailService.java
UniversityDomainService.java
AlmAgoraHubApplication.java
resources
db
migration
V2__Alter_status_and_recreate_views.sql
application.properties
.env
mvnw
mvnw.cmd
pom.xml
database
const data.txt
init.sql

```
├── frontend
│   ├── public
│   │   └── favicon.ico
│   ├── src
│   │   ├── assets
│   │   │   └── main.css
│   │   ├── components
│   │   │   ├── desktop
│   │   │   │   ├── ContainerCentralComponent.vue
│   │   │   │   └── NavBarComponent.vue
│   │   │   ├── forum
│   │   │   │   └── ReplyEditor.vue
│   │   │   ├── global
│   │   │   │   ├── JwtExpiredModal.vue
│   │   │   │   └── LogoutSuccessModal.vue
│   │   │   ├── group
│   │   │   │   └── SidebarMenu.vue
│   │   │   ├── mobile
│   │   │   │   ├── BottomNavbarComponent.vue
│   │   │   │   └── BurgerComponent.vue
│   │   │   └── settings
│   │   │       ├── UserSettingsAbout.vue
│   │   │       ├── UserSettingsContacts.vue
│   │   │       ├── UserSettingsPersonal.vue
│   │   │       └── UserSettingsSecurity.vue
│   │   ├── coordinator
│   │   │   ├── coordinator.js
│   │   │   └── router.js
│   │   ├── iam
│   │   │   ├── models
│   │   │   │   ├── group
│   │   │   │   │   ├── groupModel.js
│   │   │   │   │   └── groupPostModel.js
│   │   │   │   ├── authModel.js
│   │   │   │   ├── chatModel.js
│   │   │   │   ├── domainModel.js
│   │   │   │   ├── forumModel.js
│   │   │   │   ├── friendModel.js
│   │   │   │   ├── localeModel.js
│   │   │   │   ├── roleModel.js
│   │   │   │   ├── stagModel.js
│   │   │   │   └── userModel.js
│   │   │   └── stores
│   │   │       ├── authStore.js
│   │   │       ├── chatStore.js
│   │   │       ├── domainStore.js
│   │   │       ├── forumStore.js
│   │   │       ├── friendStore.js
│   │   │       ├── groupStore.js
│   │   │       └── localeStore.js
```


roleStore.js
stagStore.js
userStore.js
locales
cz.js
en.js
ru.js
utils
api
apiClient.js
device
useIsMobile.js
groups
checkRights.js
jwt
getUserIdFromToken.js
ws
wsClient.js
views
auth
AccessDeniedView.vue
LoginView.vue
RegistrationView.vue
VerificationView.vue
authorized
group
CreateGroupView.vue
GroupCalendarView.vue
GroupMembersView.vue
GroupNewPostView.vue
GroupPostsView.vue
GroupPrivateView.vue
GroupSettingsView.vue
GroupView.vue
GroupWrapper.vue
GroupsView.vue
user
UserChatView.vue
UserProfilePersonalView.vue
UserProfilePublicView.vue
UserProfileWrapper.vue
UserSearchView.vue
UserSettingsView.vue
UserStagView.vue
HomeView.vue
forum
authorized
ForumArchivedView.vue
ForumBannedView.vue
ForumCreateView.vue


ForumDetailView.vue
ForumFollowingView.vue
ForumsWrapper.vue
public
ForumDetailPublicView.vue
PublicForumsWrapper.vue
ForumInfoView.vue
ForumSearchView.vue
layouts
HomeLayout.vue
PublicLayout.vue
UserLayout.vue
public
GroupsBlockedView.vue
LandingView.vue
App.vue
main.js
.env
.gitignore
README.md
index.html
jsconfig.json
package-lock.json
package.json
tailwind.config.js
vite.config.js
LICENSE
README.md
docker-compose.yaml