FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Frontend webové aplikace podporující proces přijímacího řízení pro zahraniční uchazeče na ČVUT FIT
Student:	Bc. Xuan Tam Trinh
Vedoucí:	Ing. David Pešek
Studijní program:	Informatika
Obor / specializace:	Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2023/2024

Pokyny pro vypracování

Cílem práce je navrhnout a implementovat frontend webové aplikace podporující proces přijímacího řízení pro zahraniční uchazeče na ČVUT FIT. Aplikace je navrhována a vyvíjena v rámci dvou souběžných diplomových prací – tato práce se zaměří na návrh a implementaci klientské části.

Postupujte v následujících krocích:

1. Analyzujte proces přijímacího řízení a popište požadavky cílových uživatelů.
2. Na základě požadavků navrhněte klientskou část systému.
3. Konzultujte vypracovaný návrh s autorem serverové části.
4. Zvolte vhodné technologie pro realizaci projektu.
5. Implementujte funkční prototyp klientské aplikace pro webové prohlížeče.
6. Prototyp řádně uživatelsky otestujte.
7. Proveďte napojení systému na serverovou část aplikace.
8. Zhodnoťte výsledek a navrhněte další možný rozvoj.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 21. října 2022 v Praze.

Diplomová práce

FRONTEND WEBOVÉ APLIKACE PODPORUJÍCÍ PROCES PŘIJÍMACÍHO ŘÍZENÍ PRO ZAHRANIČNÍ UCHAZEČE NA ČVUT FIT

Bc. Xuan Tam Trinh

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. David Pešek
3. května 2023

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Chtěl bych upřímně poděkovat především panu inženýru Davidu Peškovi za jeho užitečné rady a vedení této diplomové práce. Jeho zkušenosti a odbornost byly klíčové pro úspěšné dokončení tohoto projektu. Také bych rád vyjádřil svou vděčnost všem členům akademické obce ČVUT, kteří svými konzultacemi přispěli k realizaci této práce. Velké poděkování patří také Tomáši Kováříkovi za jeho trpělivost a ochotu při spolupráci na backendové části vyvíjené aplikace.
Rád bych také vyjádřil své upřímné poděkování mé rodině, přítelkyni a přátelům za jejich neustálou podporu, pomoc a povzbuzení. Bez jejich podpory by vypracování této práce nebylo možné.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. Dále prohlašuji, že jsem s Českým vysokým učením technickým v Praze uzavřel licenční smlouvu o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona. Tato skutečnost nemá vliv na ust. § 47b zákona č. 111/1998 Sb., o vysokých školách, ve znění pozdějších předpisů.

V Praze dne 3. května 2023 .....

Abstrakt

Tato diplomová práce se zaměřuje na analýzu, návrh a implementaci frontendové části webové aplikace pro podporu procesu přijímacího řízení anglických studijních programů na ČVUT FIT v Praze. Aplikace sestává z uživatelské části pro uchazeče a administrátorské části pro organizátory přijímacího řízení. Práce obsahuje obchodní analýzu přijímacího procesu, technickou analýzu vhodných technologií, komplexní návrh architektury aplikace a uživatelského rozhraní, popis implementace frontendové části včetně ověření její správnosti. Výsledkem práce je funkční prototyp, který je napojený na backendovou část.

Klíčová slova webová aplikace, UI, UX, frontend, React, přijímací řízení, automatizace

Abstract

This thesis focuses on the analysis, design and implementation of the frontend part of a web application for supporting the admission process for English study programmes at CTU FIT in Prague. The application consists of a user part for applicants and an administrator part for admissions organizers. The thesis contains a business analysis of the admission process, technical analysis of suitable technologies, a comprehensive design of the application architecture and user interface, description of the implementation of the frontend part including verification of its correctness. The result of the work is a functional prototype that is integrated with the backend part.

Keywords web application, UI, UX, frontend, React, admission procedure, automation

Seznam zkratek

API	Application Programming Interface
CD	Continuous delivery
CI	Continuous integration
CSS	Cascading Style Sheets
CSV	Comma-separated values
DOM	Document Object Model
DoD	Definition of done
FIT	Fakulta informačních technologií
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
Hi-Fi	High Fidelity
IDE	Integrated development environment
JSX	JavaScript XML
LCD	Liquid crystal display
Lo-FI	Low Fidelity
REST	Representational state transfer
SPA	Single page application
UCP	Use Case Points
UI	User interface
UML	Unified Modeling Language
UX	User experience
VIC	Výpočetní a informační centrum
XML	Extensible Markup Language
XSS	Cross-Site Scripting
ČVUT	České vysoké učení technické

Úvod

V dnešní moderní společnosti je patrný narůstající zájem o vyšší vzdělání, stejně jako o studium v zahraničí. Tento trend je zaznamenaný i na ČVUT FIT, kde každý rok přichází více mezinárodních studentů. Organizační proces přijímacího řízení pro anglické uchazeče se však výrazně liší od běžného přijímacího řízení, zejména kvůli své distanční formě, což činí organizaci náročnější a zdlouhavější. V době psaní této práce na fakultě neexistuje systém pro podporu tohoto procesu, a většina organizace probíhá manuálně. Tento proces je náročný jak pro uchazeče, tak pro studijní oddělení. Každou iteraci přijímacího řízení pro anglické studijní programy se účastní desítky uchazečů, v budoucnosti se však odhaduje, že by jich mohlo být řádově stovky. Tato skutečnost zdůrazňuje potřebu zavedení jednotného systému pro podporu a automatizaci tohoto procesu, a to je zároveň hlavním cílem této práce.

Tato práce se zaměřuje na analýzu, návrh a implementaci frontendové části webové aplikace pro podporu procesů přijímacího řízení anglických studijních programů na ČVUT FIT. Aplikace je vyvíjena ve spolupráci s Tomášem Kováříkem, který navrhuje backendovou část v rámci své diplomové práce. Aplikace se skládá z uživatelské části pro uchazeče a administrátorské části pro organizátory přijímacího řízení. Uchazeči mohou ve sjednoceném prostředí spravovat své přihlášky, přihlašovat se na různé události související s přijímacím řízením či komunikovat s organizátory. Zároveň aplikace umožňuje administrátorům efektivní správu a organizaci aktivit spojených s přijímacím řízením. Hlavním cílem aplikace je snížit administrativní zátěž a zjednodušit proces pro všechny zúčastněné strany.

Tato práce je strukturována do sedmi kapitol. V první kapitole jsou uvedeny hlavní i dílčí cíle práce. Druhá kapitola se věnuje obchodní analýze, která zahrnuje představení obchodních procesů přijímacího řízení a zkoumání obdobného systému Přiříz. Na základě této analýzy jsou identifikovány a zdokumentovány veškeré požadavky na aplikaci. Třetí kapitola obsahuje komparativní analýzu populárních technologií pro vývoj webových klientů, z nichž je vybrána ta nejvhodnější pro vývoj aplikace. Ve čtvrté kapitole je prezentován komplexní návrh architektury aplikace a na základě analýzy požadavků jsou vytvořeny Lo-Fi a Hi-Fi prototypy, které jsou validovány s autorem backendové části. Pátá kapitola se zabývá technickým popisem kritických částí implementace a jejím následným napojením na backendovou část. Šestá kapitola pojednává o testovacích metodách využitých během vývoje, včetně finálního uživatelského testování, které ověřuje celkovou funkčnost systému. V závěrečné kapitole je zhodnocen výsledný stav aplikace a nastíněny další možnosti pro další budoucí rozvoj.

Kapitola 1
Cíle

Hlavním cílem této diplomové práce je navrhnout a implementovat frontendovou část webové aplikace pro podporu přijímacího řízení anglických studijních programů. Aplikace bude určena pro uchazeče i organizátory přijímacího řízení.

Pro dosažení hlavního cíle je nezbytné stanovit a splnit několik dílčích úkolů. Nejprve je nezbytné se seznámit s interními procesy přijímacího řízení a všemi formálními náležitostmi jak ze strany organizátorů, tak ze strany uchazečů. Následně je vhodné analyzovat existující řešení, což umožní identifikovat silné a slabé stránky současných systémů, z nichž lze čerpat inspiraci nebo se vyhnout chybám. Po získání přehledu o procesu přijímacího řízení a jeho nedostatcích je třeba komunikovat s klíčovými osobami a vytvořit detailní specifikaci požadavků, na základě které bude založen celý následující vývoj aplikace. Dále je důležité vybrat vhodnou technologii pro vývoj a detailně ji technicky prostudovat. Poté je třeba navázat spolupráci s autorem backendové části, vytvořit společný návrh aplikace a definovat společné API, což umožní paralelní vývoj. Posledním dílčím cílem je realizovat celou aplikaci na základě návrhu a řádně ji otestovat, což zaručí, že navržená aplikace je kvalitní a splňuje očekávání všech zainteresovaných stran.

Kapitola 2
Obchodní analýza

Žádný projekt nemůže dosáhnout úspěchu bez řádné identifikace a dokumentace potřeb zúčastněných stran. Proto je nezbytné nejprve provést podrobnou obchodní analýzu, která zkoumá problémovou doménu, seznamuje se s interními procesy a identifikuje klíčové osoby. Tato kapitola je zaměřena na analýzu oblasti přijímacího řízení pro anglické studijní programy. Následně je provedena detailní analýza podobného systému, během které jsou zdůrazněny jeho silné stránky a nedostatky. To poskytne jasnější představu o navrhovaném systému a pomůže předejít případným chybám v rané fázi vývoje. Nakonec je kapitola uzavřena detailní analýzou požadavků, včetně modelování případů užití a odhadů pracnosti.

2.1 Analýza domény

Na ČVUT FIT se každoročně, kromě běžného přijímací řízení pro české studijní programy, koná také speciální přijímací řízení pro anglické studijní programy, které kvůli své distanční podobě obnáší odlišné procesy. Tato práce se zaměřuje právě na přijímací řízení pro uchazeče o anglické studijní programy.

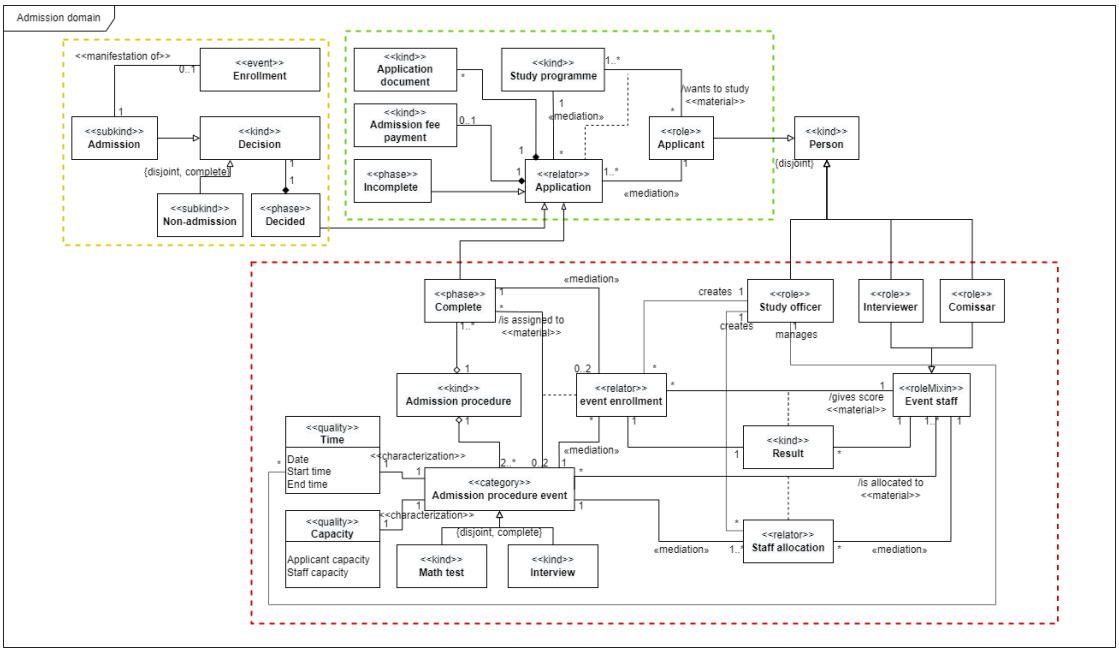
Uchazeči, kteří mají zájem o anglický studijní program nejprve vyplňují přihlášku na univerzitním portálu `prihlaska.cvut.cz`, který zaeviduje přihlášku do systému. Následně jsou povinni zaplatit příslušný poplatek za přijímací řízení a připojit k přihlášce veškeré povinné dokumenty, včetně dokladu o předchozím vzdělání, kopie pasu a důkazů o znalosti anglického jazyka. Jakmile jsou splněny všechny požadavky, musí být přihláška manuálně ověřena studijním oddělením v systému iKOS, čímž se zajistí splnění všech formálních náležitostí, a následně je zařazena do samotného přijímacího řízení. Po uplynutí uzávěrky pro podávání přihlášek obdrží výpočetní a informační centrum ČVUT e-mailové oznámení se seznamem všech způsobilých uchazečů, kterým jsou následně poskytnuty přístupové údaje do systému Moodle. Poté jsou uchazeči vyzváni k absolvování zkušebního testu, aby se seznámili s testovacím prostředím, ve kterém se bude konat test z matematiky.

Nejdůležitější fází přijímacího řízení je absolvování testu z matematiky a případně následný online pohovor k ověření znalostí anglického jazyka. Celý průběh testu z matematiky je realizován distanční formou prostřednictvím systému Moodle pod dohledem komisaře, který je umístěn do síťové multimediální laboratoře SAGElab, kde může sledovat aktivitu všech uchazečů během testu na LCD stěně složené ze 16 monitorů. Uchazeči musí dosáhnout minimální hranice 60 bodů, aby úspěšně složili test z matematiky. Po úspěšném absolvování testu jsou uchazeči v určeném termínu pozváni k ústnímu pohovoru, během kterého musí tazatelům prokázat základní komunikační dovednosti v anglickém jazyce. Obě části jsou organizovány studijním oddělením, které rozvrhují jak uchazeče, tak komisaře i tazatele do naplánovaných termínů na základě individuální dohody.

Po zohlednění kapacity studijních programů, výsledků testů z matematiky a pohovorů, obdrží uchazeči rozhodnutí o přijetí nebo odmítnutí ke studiu. Přijatí uchazeči poté musí zaplatit poplatek spojený se studiem za první semestr a osobně se dostavit na studijní oddělení, aby se zapsali do svého vybraného studijního programu.

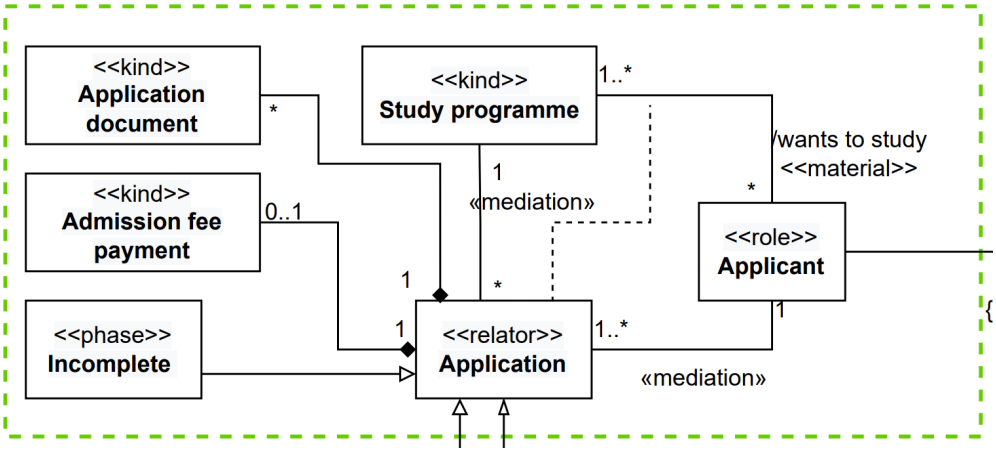
2.1.1 Analytický doménový model

K vizualizaci strukturálních aspektů problémové domény je zvolen konceptuální modelovací jazyk OntoUML, který umožňuje vytváření sémanticky bohatých konceptuálních modelů. Je zvláště užitečný pro modelování komplexních domén s mnoha entitami, vztahy a závislostmi. OntoUML umožňuje vytváření různých perspektiv na danou doménu, což zajišťuje, že jsou zachyceny různé aspekty a stanoviska zainteresovaných stran. V kontrastu s OntoUML je klasický modelovací jazyk UML spíše vhodný pro modelování technických návrhů a zaměřuje se na modelování chování a struktury softwarových systémů. OntoUML naproti tomu pomáhá zajistit, že konceptuální modely jsou konzistentní, bez nejednoznačností nebo protichůdností, což může být obtížnější dosáhnout se standardním UML. Jeho schopnost podpory různých perspektiv a zajištění konzistence modelů činí OntoUML cenným nástrojem pro různé oblasti, které se zabývají složitými a heterogenními doménami. [1]



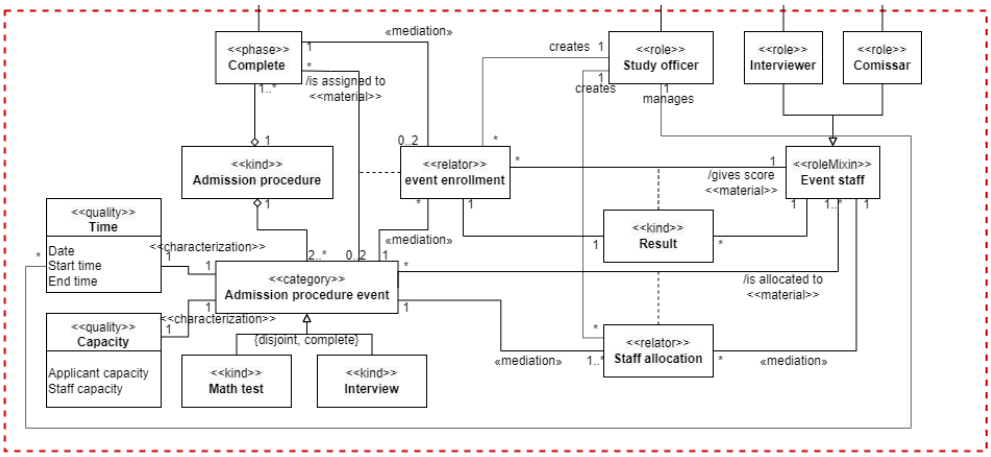
Obrázek 2.1 Doménový model přijímacího řízení v OntoUML

Diagram doménového modelu, zobrazený na obrázku 2.1, ilustruje strukturu přijímacího procesu pro uchazeče o anglické studijní programy. Tento proces lze rozdělit do tří subdomén, které zahrnují různé entity a vztahy mezi nimi. Každá subdoména představuje jednu větší fázi v procesu přijímacího řízení, a proto je vhodné každou zkoumat zvlášť. Zelená subdoména znázorňuje fázi podávání přihlášek, v níž mají uchazeči možnost projevit zájem o daný studijní program. Červená subdoména zobrazuje bodovací fázi, jež sestává z matematického testu a ústního pohovoru. Poslední žlutá subdoména představuje finální fázi přijímacího řízení, kde probíhá rozhodnutí o přijetí nebo nepřijetí uchazeče. Při analýze je důležité se zaměřit na vztahy a závislosti mezi jednotlivými entitami, které nemusí být na první pohled zřejmé ze samotného popisu domény.



Obrázek 2.2 Model subdomény podání přihlášek

Z diagramu subdomény podání přihlášek je patrné, že jeden uchazeč může mít zájem o více studijních oborů, a proto může podat více přihlášek, což je zachyceno vícenásobnou kardinalitou mezi entitami `Applicant` a `Application`. Dále může každá přihláška nabývat jednoho ze tří stavů — nekompletní `Incomplete`, kompletní `Complete` nebo rozhodnutá `Decided`. V subdoméně podání přihlášek jsou zobrazeny pouze přihlášky ve stavu nekompletní, což jsou právě nezaplacené a neúplné přihlášky, které neobsahují všechny povinné dokumenty.

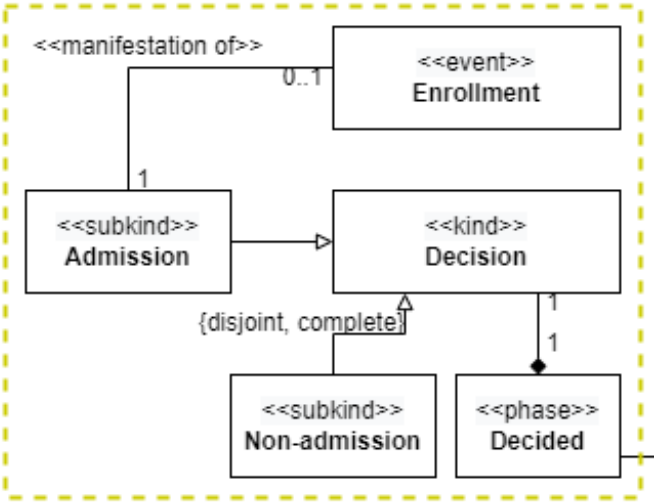


Obrázek 2.3 Model subdomény bodovací části přijímacího řízení

Jakmile jsou splněny formální požadavky, přihláška je považována za kompletní a je zařazena do přijímacího řízení, jak lze vidět na obrázku 2.3 ve vazbě mezi entitami `Complete` a `Admission procedure`. Z diagramu lze hned na první pohled vidět, že tato přijímacího řízení je ze všech nejobsáhlejší a z praktického hlediska také organizačně nejnáročnější. Bodovací fáze je tvořena dvěma typy událostí — z matematického testu `Math test` a ústního pohovoru `Interview`. Ačkoli je nutné naplánovat minimálně jeden test a pohovor, může být v závislosti na počtu přihlášek naplánováno více dálostí, což znázorňuje vazba přijímacího řízení na entitu události `Admission procedure event`. Každá událost má předem stanovené datum konání a kapacitu pro uchazeče a supervizora[1]. K jednotlivým událostem se může přihlásit daný počet uchazečů, zatímco ke

[1]Supervizor události je v kontextu přijímacího řízení tazatel či komisař.

každé události musí být přiřazen minimálně jeden supervizor. Plánování data konání, rozdělení uchazečů na konkrétní události a přiřazení supervizorů ke každé události je zodpovědností studijního referenta, což je zachyceno v diagramu vazbami na entitu `Study officer`. Z diagramu je rovněž patrné, že bodování jednotlivých uchazečů může provádět i jiný supervizor než ten, který se zúčastní dané události. Pokud uchazeč nesplní požadavky na bodování, není opravný termín poskytnut, což lze vidět v maximální kardinalitě 2 ve vazbě přihlášky `Complete` na událost `Admission procedure event`.



Obrázek 2.4 subdoména rozhodnutí o přijetí

Po absolvování testu, ústního pohovoru, nebo v případě neúspěšného vyhodnocení jakékoliv události v přijímacím řízení, přechází přihláška do stavu rozhodnutá `Decided`. V této fázi se na přihlášky váže rozhodnutí `Admission`, které může znamenat přijetí nebo nepřijetí uchazeče ke studiu. V případě přijetí dochází k ukončení přijímacího řízení, kdy je uchazeč zapsán ke studiu. Naopak, v případě nepřijetí uchazeče končí jeho účast v přijímacím řízení a uchazeč musí zvážit další možnosti, jak dosáhnout svých studijních cílů.

2.1.2 Zhodnocení

Po analýze procesů a doménového diagramu bylo zjištěno, že určitá část přijímacího řízení, konkrétně proces podání přihlášek a přijetí uchazečů, je již z velké části automatizována prostřednictvím systému iKOS. Avšak, stále zde existuje mezera ve správě a organizaci samotného přijímacího řízení, což je patrné zejména při zkoumání doménového diagramu v obrázku 2.1. Studijní referent musí na základě individuální domluvy s uchazeči plánovat termíny a následně k nim přiřazovat uchazeče a supervizory, což představuje značnou administrativní zátěž. Navíc, nízká informovanost uchazečů a vysoký počet emailových dotazů na studijní oddělení přispívají k další neefektivitě. Ačkoliv v současné době je počet uchazečů v řádu desítek, odhady naznačují, že v budoucnu může tento počet vzrůst až na řády stovek. S aktuálními procesy by zvládnutí takového množství žadatelů bylo obtížné.

Řešením těchto problémů je navrhnout a implementovat systém, který umožní delegovat odpovědnost za přihlašování na termíny na samotné uchazeče a zaměstnance. Tímto způsobem lze snížit administrativní zátěž spojenou s domlouváním časových možností mezi uchazeči a zaměstnanci studijního oddělení. Systém by měl také poskytovat relevantní informace o aktuálním průběhu přijímacího řízení a dokonce sám uchazeče navigovat k dalším krokům. To povede ke snížení počtu dotazů na studijní oddělení a zvýší se tím informovanost uchazečů.

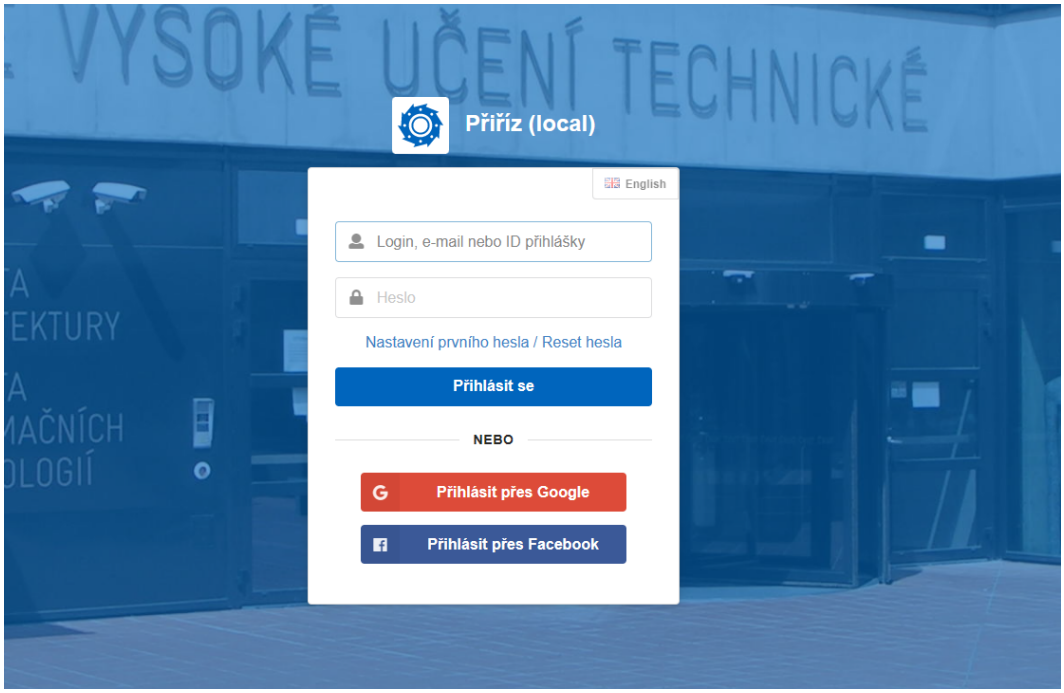
2.2 Analýza systému Přiríz

V rámci obchodní analýzy je vhodné provést důkladnou analýzu podobných řešení. Tato analýza poskytuje cenné poznatky o vlastnostech, silných a slabých stránkách stávajících systémů a pomáhá identifikovat příležitosti pro zlepšení a inovaci. Analýzou podobných řešení lze získat hlubší porozumění, potřebám a preferencím svých cílových uživatelů a vytvořit nový systém, který tyto potřeby lépe splňuje. Analýza obdobných řešení může také pomoci identifikovat potenciální rizika a výzvy a usnadnit rozhodnutí o klíčových prvcích návrhu, jako je architektura, uživatelské rozhraní nebo funkčnost. [2]

V kontextu standardního přijímacího řízení na ČVUT FIT plní aplikace Přiříz roli podpůrného systému, což může posloužit jako inspirace pro navrhovaný systém. Tento systém je využíván výhradně pro české přijímací řízení a je rozdělen do dvou částí — administrační a uživatelskou. I když se Přiříz ukázal jako cenný nástroj pro správu přijímacího řízení pro české studijní programy, z důvodů rozdílných procesů jej nelze uplatit ani rozšířit pro anglické přijímací řízení. Tato podkapitola má za cíl provést analýzu systému Přiříz, aby byly identifikovány vlastnosti a funkcionality, které by mohly být inspirací pro začlenění do navrhovaného systému.

2.2.1 Autentizace

Při vstupu do aplikace Přiříz se uživatelům jako první zobrazí rozhraní s přihlašovacím formulářem, který slouží k ověření jejich identity. Tento formulář vyžaduje, aby uživatelé zadali svoji e-mailovou adresu nebo identifikátor přihlášky spolu s heslem. Tento prvnotní krok zajišťuje, že pouze oprávnění uživatelé mají přístup k aplikaci. Jakmile uživatelé poskytnou potřebné přihlašovací údaje, jsou automaticky přesměrováni do uživatelské nebo administrátorské sekce, v závislosti na jejich roli a oprávnění. Přihlášení do uživatelské sekce je možné na základě identifikátoru podané přihlášky nebo e-mailové adresy, zatímco administrátorská sekce vyžaduje přihlášení pouze na základě e-mailové adresy.

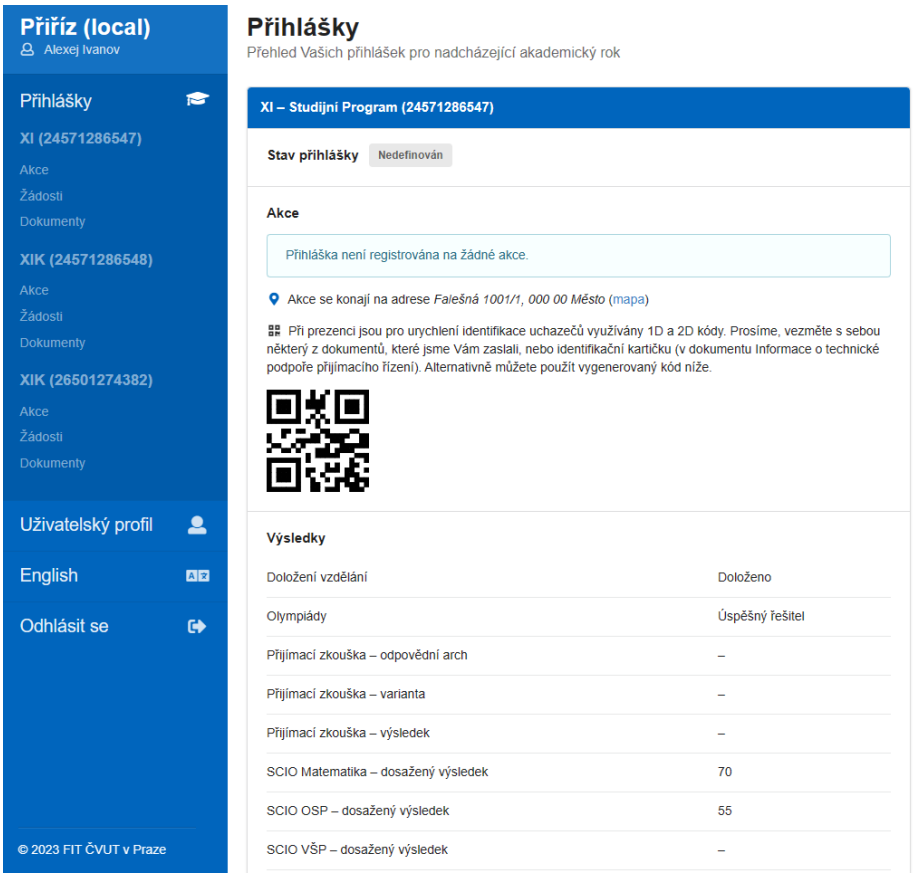


Obrázek 2.5 Přihlašovací formulář v aplikaci Přiříz

Aplikace také nabízí alternativní způsob přihlášení, který využívá třetí strany, konkrétně Google a Facebook, jako poskytovatele identity. Avšak vzhledem k bezpečnostním rizikům spojeným s náchylností k odcizením účtů či hackerským útokům a možným obavám uživatelů ohledně ochrany soukromí, se doporučuje v aplikacích tento způsob přihlášení nepoužívat. Využití třetích stran pro autentizaci může být také problematické z hlediska závislosti na externích službách a komplikací při případných změnách v API nebo politikách. [3]

K prvnímu vstupu do uživatelské sekce je nutné nejprve provést reset hesla, aby se uživatelé mohli přihlásit. Pro tento účel je nejprve potřeba navštívit specializovanou stránku pro reset hesla, na které uživatelé zadávají svůj e-mail. Poté je na jejich e-mailovou adresu odeslán odkaz, který umožňuje časově omezené jednorázové přihlášení do účtu. Tímto způsobem uživatelé mají možnost nastavit si nové heslo a získat tím přístup do účtu.

2.2.2 Uživatelská část



Obrázek 2.6 Uživatelská sekce v aplikaci Přiříz

Uživatelské rozhraní aplikace Přiříz je navrženo s důrazem na jednoduchý a minimalistický design, který zaručuje přehlednost a minimalizuje rozptýlení uživatelů během práce s aplikací. Tento styl designu je prospěšný z několika důvodů. Především umožňuje uživatelům soustředit se na základní úkoly, čímž zlepšuje celkový uživatelský zážitek a efektivitu práce. Rozvržení všech stránek je jednotné, s postranním menu zobrazeným na levé straně obrazovky a obsahem stránky na pravé straně. Jednotné rozvržení zlepšuje orientaci v aplikaci a pomáhá uživatelům rychle se seznámit s rozhraním, což snižuje čas strávený hledáním potřebných funkcí a informací.

Po přihlášení se uživatelům zobrazí v aplikaci aktuality spojené s přijímacím řízením, díky čemuž jsou informováni o důležitých změnách a aktualitách. Navíc, jak bylo vysvětleno v popisu domény na obrázku 2.6, uchazeči mohou podávat přihlášky do více studijních programů — z tohoto důvodu, bez ohledu na přihlášku, kterou uživatel použil pro přihlášení, ukáže aplikace v bočním menu všechny zaznamenané přihlášky a možné akce, které lze s nimi provést. Tato funkce zajišťuje, že uživatelé mají snadný přístup ke všem svým přihláškám a mohou s nimi efektivně pracovat.

Obrázek 2.7 Uživatelské akce v aplikaci Přiříz

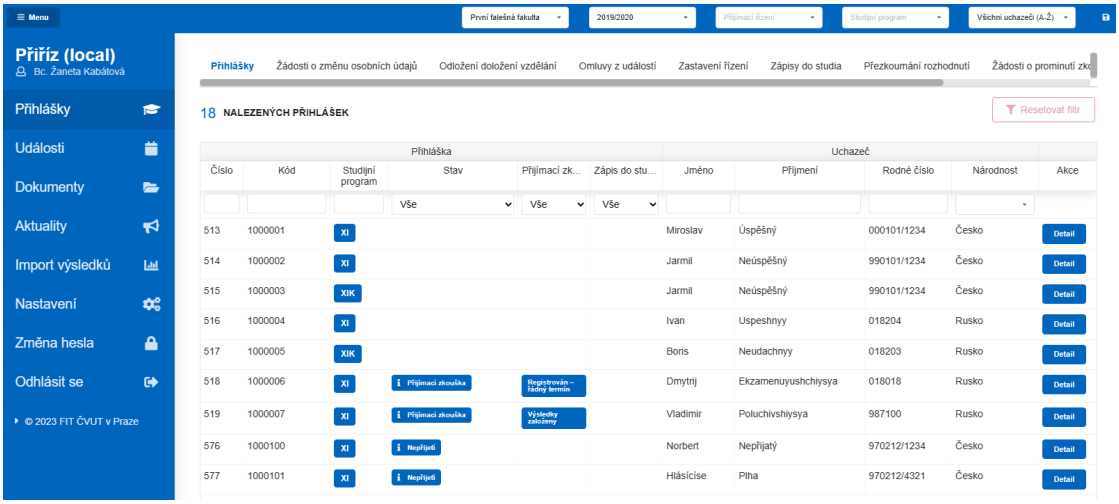
V rámci jedné přihlášky se uživatelé mohou přihlašovat na akce související s přijímacím řízením, jako jsou přijímací zkoušky nebo zápis. Na rozdíl od přijímacího řízení pro anglické studijní programy se zde nachází pouze písemný test a pohovor není součástí. Přihlašování k jednotlivým termínům lze provádět prostřednictvím přehledové tabulky, zobrazené na obrázku 2.7. Tabulka poskytuje relevantní informace o termínech, jako jsou čas, datum otevření a uzavření termínu, kapacita, místo a případné poznámky. Pokud termín volný a otevřený pro zápis, ve sloupci akcí se zobrazí tlačítka pro přihlášení nebo případně odhlášení pro daný termín. Toto uspořádání informací zjednodušuje uživatelům rozhodování při výběru preferovaných termínů a usnadňuje plánování.

Kromě přihlašování na akce přijímacího řízení mohou uživatelé vytvářet žádosti na studijní oddělení pro různé účely. V době psaní této práce aplikace Přiříz umožňuje uživatelům žádat o zastavení řízení, přezkoumání rozhodnutí, prominutí přijímací zkoušky nebo zápis do studia. Tato funkce poskytuje centralizovanou platformu pro správu žádostí uživatelů, čímž zjednodušuje komunikaci se studijním oddělením. Při podání žádosti je uživateli zobrazen formulář, který vyplní, a po potvrzení aplikace vygeneruje dokument, který uchazeč podepíše a předá osobně studijnímu oddělení.

Další funkcí, která je dostupná pro individuální přihlášky, je možnost stahování relevantních dokumentů spojených s přijímacím řízením ve formátu PDF. Poskytování snadného přístupu k základním dokumentům zajišťuje, že uchazeči mají k dispozici veškeré potřebné informace pro hladký průběh přijímacího řízení.

Nakonec si uživatelé mohou prohlédnout svůj uživatelský profil, kde si mohou zkontrolovat své osobní a kontaktní údaje a v případě potřeby požádat o jejich změnu, pokud jsou některé údaje nesprávné. Uživatelé si také mohou v profilové sekci změnit heslo, v případě, že jim stávající nevyhovuje nebo ztratili své původní. Díky možnosti editovat uživatelský profil aplikace umožňuje uživatelům udržovat své informace aktuální, což je klíčové pro organizátory přijímacího řízení.

2.2.3 Administrátorská část



Obrázek 2.8 Ukázka administrátorského rozhraní aplikace Přiříz

Aplikace Přiříz obsahuje jednotné administrační rozhraní pro efektivní řízení a organizaci celého procesu přijímacího řízení. Po přihlášení se administrátorovi zobrazí stránka s grafickým rozložením podobným uživatelskému rozhraní — na levé straně obrazovky je boční menu a na pravé straně obsah stránky. V horní části obrazovky je navíc panel, který umožňuje filtrovat obsah podle roku přijímacího řízení, studijních programů nebo jména uchazečů, což usnadňuje hledání konkrétních informací. Administrační rozhraní je rozsáhlé a nabízí širokou škálu podpůrných nástrojů pro správu funkcionalit vystavené pro uchazeče.

Obrázek 2.8 zobrazuje tabulku přihlášek, která administrátorům umožňuje sledovat všechny přihlášky evidované v rámci přijímacího řízení. Tabulka poskytuje detailní informace o přihláškách, včetně jejich stavu v kontextu přijímacího řízení. Po přechodu do detailu přihlášky lze zobrazit veškeré žádosti, události či dokumenty spojené s přihláškou, což umožňuje rychle získat informace o jednotlivých uchazečích. Díky tomu mohou administrátoři snadno získat přehled o všech evidovaných přihláškách a případných potřebách uchazečů. Data o přihláškách nevznikají lokálně v systému, ale jsou synchronizována vůči systému KOS z bezpečnostních důvodů.

Aplikace rovněž umožňuje správu událostí, což je klíčová funkce pro koordinaci a plánování různých částí přijímacího procesu. K dispozici je přehledová tabulka všech termínů, ze které lze přejít do detailu konkrétního termínu, kde jsou k dispozici informace o průběhu, kapacitách a seznamu všech přihlášených. Uvnitř detailu události je možné sledovat docházku a generovat dokumenty, které budou uchazečům zobrazeny v uživatelské sekci. Ze stejných důvodů jako u přihlášek, jsou všechny události jsou importovány ze systému KOS a poté již nelze provádět jejich modifikace. Co se týče výsledků uchazečů z jednotlivých událostí, jsou hromadně načteny do aplikace ze souborů ve speciálním formátu, díky čemuž není nutné výsledky zapisovat manuálně.

Dále lze v aplikaci vytvářet a spravovat aktuality, což je užitečná funkce pro informování uchazečů o důležitých novinkách, změnách nebo aktualizacích. Tyto aktuality vznikají přímo v systému Přiříz a nejsou importovány z žádného externího datového zdroje. Při vytváření aktuality je možné zvolit pro který studijní program se má aktualita zobrazit a nastavit časový interval, kdy bude aktualita viditelná. To umožňuje cíleně informovat různé skupiny uchazečů. Pro účely lokalizace lze zadat titulek a popisek aktuality v češtině a případně i v angličtině.

Jak bylo nastíněno v popisu administrátorské části, data týkající se přihlášek, studijních programů a událostí jsou z bezpečnostních důvodů pravidelně importována ze systému KOS. Jistým nedostatkem je však absence grafického rozhraní pro realizaci tohoto importu v aplikaci. Namísto toho se data načítají prostřednictvím specializovaných bash skriptů, které se provádějí mimo administrační rozhraní aplikace. Pokud jde o export dat zpět do systému KOS, aplikace umožňuje generování souborů ve formátu CSV, jež obsahují data ve struktuře zpracovatelné systémem KOS. Tímto způsobem je zajištěna kompatibilita a přenositelnost dat mezi aplikací Přiříz a systémem KOS.

2.2.4 Zhodnocení

Jak bylo zjištěno v analýze domény v sekci 2.1, hlavními nedostatky v procesu přijímacího řízení pro anglické studijní programy jsou především nízká informovanost uchazečů a zároveň obrovská administrativní zátěž pro studijní oddělení. Přestože se proces přijímacího řízení pro české studijní programy liší od procesu přijímacího řízení pro anglické studijní programy, aplikace Přiříz nabízí řadu funkcí, které by mohly být použity k automatizaci procesů a výslednému zjednodušení přijímacího řízení pro anglické studijní programy.

Autentizační mechanismus je adekvátně navržen, jednotné přihlašování usnadňuje přístup pro uchazeče i administrátory, přičemž přihlašování prostřednictvím třetích stran by však z bezpečnostních důvodů nemělo být implementováno a mělo by být nahrazeno bezpečnější metodou.

Modul pro správu přihlášek nabízí jednoduchý způsob, jak získat přehled o přihláškách jak pro uchazeče, tak pro administrátory. Nicméně, chybí zde celková navigace uchazečů v kontextu celého přijímacího řízení. V navrhovaném řešení by mohly být například zavedeny grafické prvky pro zobrazení stavu přijímacího řízení, které by uchazečům ukazovaly aktuální pokrok jejich přihlášky v rámci přijímacího procesu a navigoval je k dalším krokům. Z pohledu administrátorů je modul pro správu přihlášek velmi slušně implementovaný, avšak import přihlášek by mohl být také součástí systému, nikoliv řešený pouze pomocí bash skriptů.

Dále modul pro generování žádostí umožňuje automatizaci generování žádostí ve formátu PDF a snižuje počet dotazů na studijní oddělení. Vzhledem k distanční povaze anglického přijímacího řízení, kdy se žádosti řeší převážně prostřednictvím emailové komunikace, lze generování žádostí v navrhované aplikaci zobecnit a zavést systém posílání zpráv mezi uchazeči a administrátory přímo v aplikaci. Tímto způsobem by byla komunikace mezi oběma stranami zjednodušena a zrychlena.

Sekce pro přihlášení na události je pro uchazeče velmi dobře řešena, neboť jim poskytuje rychlý přehled o důležitých termínech a akcích v rámci přijímacího řízení. Z pohledu administrátorů by tento modul mohl být rozšířen o evidenci a alokaci supervizorů na konkrétní termíny, což by umožnilo delegovat zodpovědnost za přihlašování a organizaci událostí přímo na zaměstnance. Tím by se snížila administrativní zátěž studijního oddělení spojenou s plánováním termínů a následným rozdělováním supervizorů na jednotlivé termíny.

Pro zvýšení informovanosti uchazečů lze také využít systém aktualit, který je již implementován v systému Přiříz. Jako další podpůrné nástroje lze zavést systém notifikací a automatického zasílání emailových zpráv, které by informovaly uchazeče o důležitých změnách, termínech či událostech v rámci přijímacího řízení. Tímto způsobem by byla informovanost uchazečů značně zlepšena a zároveň by se snížila zátěž studijního oddělení, které by nemuselo řešit tolik dotazů od uchazečů.

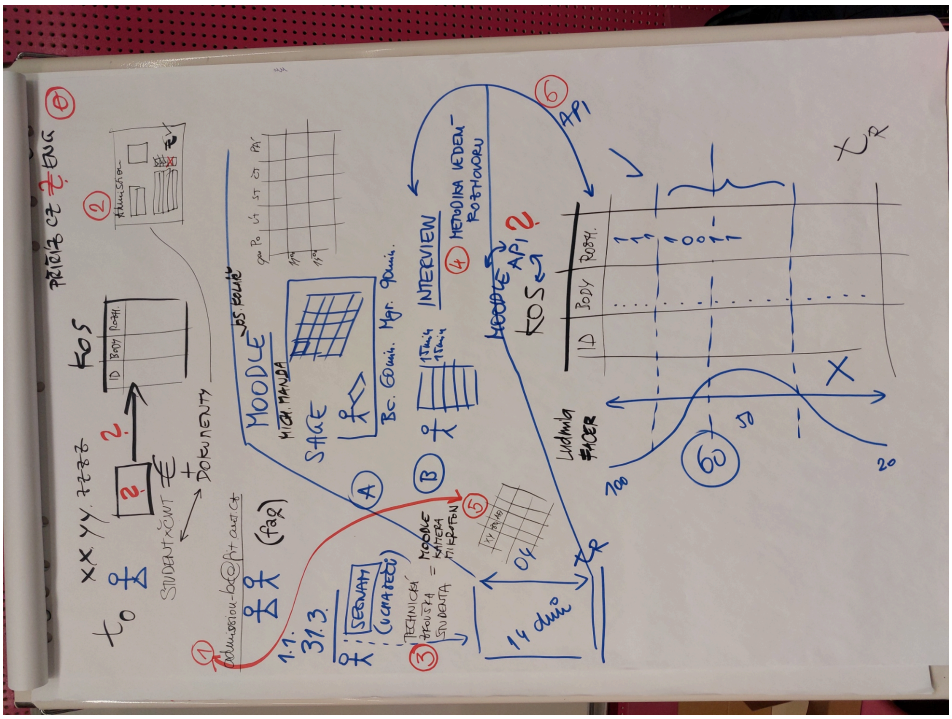
Celkový design aplikace Přiříz je minimalistický a snadno srozumitelný, s mnoha funkcionalitami, textovými nápovědami a vysvětlivkami, které umožňují uživatelům rychle se v systému zorientovat a efektivně s ním pracovat. Dále, velká část výše uvedených modulů může být využita i pro vlastní řešení. Pro minimalizaci dotazů směřujících na studijní oddělení by měl být také návrh řešení zaměřen na snadnost použití a měl by uživatele vést k akcím, které jsou potřebné pro postup v přijímacím řízení.

2.3 Analýza požadavků

Po důkladném seznámení se s problémovou doménou a detailní analýze obdobné aplikace Přiříz je nutné identifikovat a jasně definovat požadavky a očekávání zainteresovaných stran v rámci analýzy požadavků. Účelem analýzy požadavků je zajistit, aby konečný produkt vyhovoval potřebám všech koncových uživatelů.

2.3.1 Sběr a validace požadavků

Pro systematický sběr a validaci požadavků je nezbytné stanovit metody a postupy pro jejich shromažďování. To je důležité, protože bez jasně definovaných postupů a metod by celý proces sběru požadavků mohl být neefektivní a chaotický. Z toho důvodu byly se studijním oddělením dohodnuty pravidelné osobní schůzky, které umožnily postupně budovat konkrétní představu o specifikách navrhovaného systému prostřednictvím skupinových diskuzí a brainstormingu.



Obrázek 2.9 Výsledek brainstormingového setkání na jedné z informačních schůzek

Na úvodní schůzce proběhlo seznámení s klíčovými osobami, což vedlo k zlepšení pochopení interních procesů přijímacího řízení pro anglicky mluvící uchazeče. Byly také představeny externí systémy, které je nutné integrovat do navrhovaného systému, jmenovitě KOS, Moodle a mailové servery. Dále byly zjištěny také zodpovědné osoby, které dané systémy spravují. Poté následovaly schůzky, na kterých se probíraly technické detaily externích systémů a možnosti jejich integrace do vlastního řešení. Na základě těchto schůzek byla vytvořena a validována specifikace požadavků ze strany zaměstnanců studijního oddělení. Během celého průběhu sběru a validace požadavků bylo využito různých metod, jako jsou brainstormingová setkání, analýza dokumentů, skupinové diskuze, analýza rozhraní, rozhovory či prototypování. Tyto techniky byly nezbytné pro pochopení potřeb uživatelů, detailů stávajících systémů a nezbytných funkcí pro nový systém.

Brainstormingová setkání poskytla účastníkům prostor pro svobodné sdílení nápadů a návrhů, což přispělo k inovativním řešením a hlubšímu porozumění navrhovaného systému. Analýza dokumentů odhalila cenné informace o současném systému, identifikovala nedostatky a oblasti vhodné pro zlepšení. Skupinové diskuze, které zahrnovaly reprezentativní uživatele a zúčastněné strany, poskytly zpětnou vazbu týkající se jejich potřeb, možností a problémů, což usnadnilo identifikaci a specifikaci požadavků. Analýza rozhraní systémů Přiríz, iKOS a Moodle zaručila správné zahrnutí veškerých požadavků na integraci s externími systémy. Nakonec, rozhovory se zúčastněnými stranami a uživateli byly zásadní pro pochopení jejich cílů a očekávání, což mělo významný dopad na konečnou představu o výsledném systému.

2.3.2 Specifikace požadavků

Kromě řádného pochopení a identifikace požadavků je nutné se zabývat i jejich samotnou dokumentací, což lze následně využít k vytváření odhadů, budoucím validacím či zaučení nových účastníků vývoje. Dobře strukturovaná dokumentace požadavků navíc usnadňuje efektivní komunikaci mezi zúčastněnými stranami a minimalizuje riziko nedorozumění, která by mohla vést k nákladným změnám nebo zpoždění v průběhu vývoje. Pro vytvoření přehledné a organizované dokumentace požadavků v rámci této práce byly uplatněny tři široce uznávané metodologické frameworky — SMART, FURPS a MoSCoW.

Prvním z aplikovaných frameworků je SMART, který představuje sadu kritérií, jež napomáhají zajištění správného definování každého požadavku. Tento framework specifikuje vlastnosti, podle kterých lze validovat jednotlivé požadavky z hlediska korektnosti. Dle SMART musí být správně definované požadavky následující:

- `Specifické:` Požadavek je jasně a konkrétně formulován.
- `Měřitelné:` Požadavek lze kvantifikovat a ověřit jeho naplnění.
- `Dosažitelné:` Požadavek je uskutečnitelný s využitím dostupných zdrojů a technologií.
- `Realistické:` Požadavky musejí být realistické vzhledem k známým omezením systému a projektovým zdrojům.
- `Sledovatelné:` Požadavky musí být sledovatelné v průběhu celého procesu vývoje, od návrhu přes implementaci až po testování a úpravy.[4]

Model FURPS představuje uznávaný framework pro definování a kategorizaci softwarových požadavků. Jeho hlavním úkolem je zajistit, že ve specifikaci požadavků jsou náležitě zohledněny všechny klíčové aspekty softwarového projektu. Díky jemnější kategorizaci požadavků, kterou FURPS poskytuje, lze ověřit, které aspekty vyvíjeného systému jsou zahrnuty. FURPS je akronym, který umožňuje rozdělit požadavky do následujících kategorií:

- `Funkčnost:` Popisuje, co systém provádí, jeho funkcionalitu a vlastnosti.
- `Použitelnost:` Odkazuje na jednoduchost použití a dostupnost systému pro uživatele.
- `Spolehlivost:` Hodnotí schopnost systému fungovat bez chyb a odolávat selhání.
- `Výkon:` Zahrnuje rychlost, efektivitu a odezvu systému.
- `Podpora:` Odpovídá na otázky týkající se údržby, rozšiřitelnosti a škálovatelnosti systému [5].

Metoda MoSCoW je technika prioritizace požadavků, která umožňuje řadit požadavky od nejdůležitějších po méně důležitých a umožňuje efektivně alokovat zdroje na základě priority jednotlivých požadavků. Díky tomu je možné zajistit, že klíčové prvky projektu budou přednostně realizovány a zdroje budou správně směřovány na základě důležitosti. Tato metoda pomáhá lépe rozhodnout, které prvky mají být implementovány jako první a které mohou být odloženy nebo zcela vynechány, pokud je to nezbytné. MoSCoW je akronym, který definuje následující priority:

- `Must have:` Kritické požadavky, které musí být splněny pro úspěch projektu.
- `Should have:` Důležité požadavky, které by měly být splněny, pokud je to možné.
- `Could have:` Požadavky, které by bylo hezké splnit, ale jejich absence neohrozí úspěch projektu.
- `Won't have:` Požadavky, které nebudou splněny v aktuálním projektu, ale mohou být zváženy pro budoucí vývoj. [6]

Aplikací modelů SMART, FURPS a MoSCoW při tvorbě specifikace požadavků umožňuje pečlivě zvážit všechny aspekty projektu, od jeho funkčnosti až po jeho důležitosti. Všechny požadavky, které byly zachyceny, jsou dokumentovány ve strukturovaném formátu, který zahrnuje:

- `Unikátní identifikátor:` Kód požadavku pro snazší odkazování.
- `Název:` Krátký vystihující název požadavku.
- `Kategorii:` Kategorizace požadavku dle FURPS.
- `Prioritu:` Prioritizace požadavku dle MoSCoW.
- `Popis:` Detailní popis požadavku.
- `Kritéria splnění (definition of done):` Podmínka, kdy lze požadavek označit jako splněný.

Tento formát poskytuje několik výhod, jako je zlepšená srozumitelnost, sledovatelnost, prioritizace a jednoduché ověření požadavků. Použitím standardizovaného formátu mohou zainteresované strany snáze pochopit a interpretovat požadavky, sledovat jejich stav a postup, zaměřit svou pozornost na nejdůležitější požadavky a ověřit, že jsou implementovány správně a fungují podle zamýšleného způsobu. Z důvodu velkého množství požadavků, které celkově tvoří 47 funkčních požadavků a 7 nefunkčních požadavků, lze kompletní specifikaci požadavků nalézt v příloze A. Pro ukázku, je níže uveden jeden z funkčních požadavků, specifikovaný ve výše uvedené struktuře.

FP1 - Uživatelské role
Priorita: Must have
FURPS: Funkčnost
Popis: Systém umožní přihlášeným uživatelům přístup k obsahu a funkcím, které odpovídají jejich uživatelské roli, a zamezí přístup k obsahu a funkcím, na které nemají oprávnění.
DoD: Aplikace úspěšně rozlišuje mezi uchazeči a administrátory, přičemž uchazeči mají přístup pouze do uživatelské sekce.

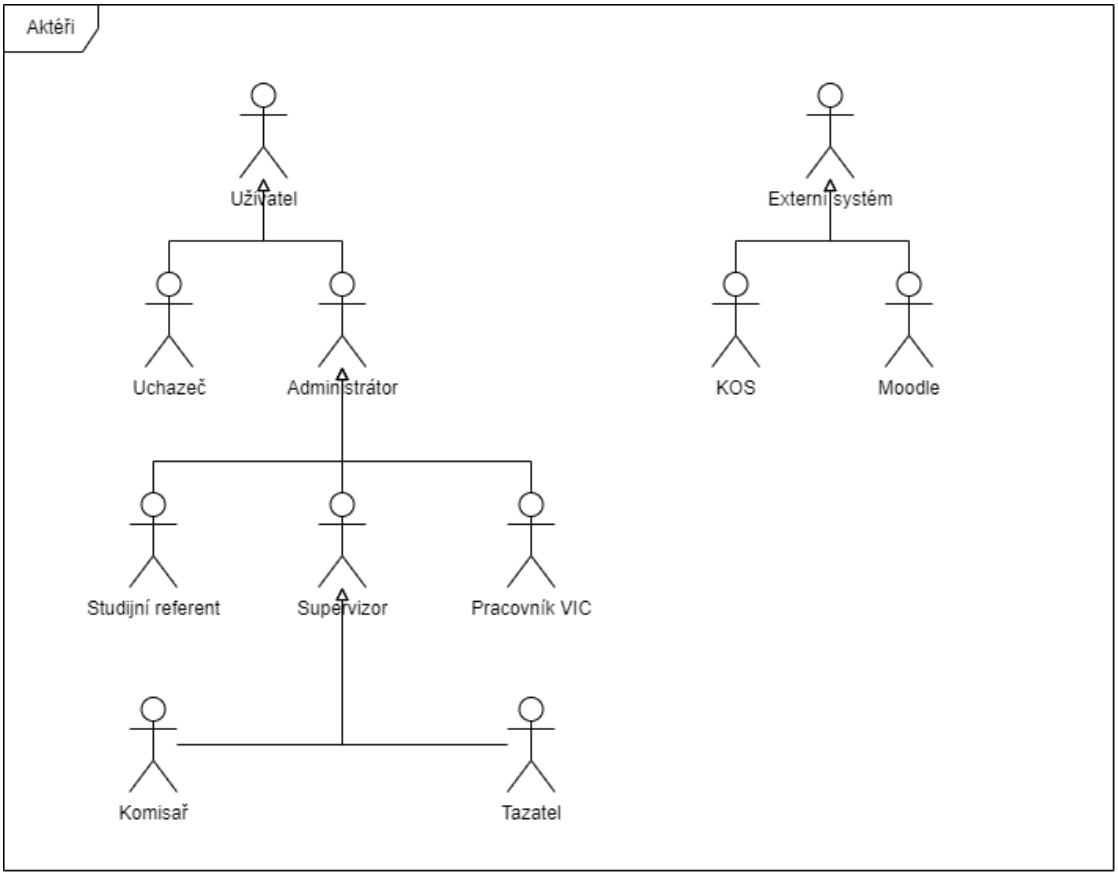
Obrázek 2.10 Ukázka strukturování specifikace požadavků

2.4 Analýza případů užití

Analýza případů užití představuje užitečnou metodu, která umožňuje vývojářům a zúčastněným stranám zachycovat, definovat a zkoumat funkční požadavky systému prostřednictvím organizovaného a strukturovaného přístupu. Tento přístup se primárně zaměřuje na zobrazování interakcí mezi systémem a jeho uživateli, označovanými jako aktéři, a na identifikaci cílů a scénářů, které systém musí splňovat. Hlavní přínos případů užití spočívá v možnosti znázornit, jak bude systém využíván z hlediska různých aktérů v různých situacích, což samotná specifikace požadavků nedokáže plně zohlednit. Tímto způsobem lze získat další nástroj pro validaci požadavků se zúčastněnými stranami [7].

2.4.1 Aktéři

V analýze případů užití jsou aktéři jednotlivci či entity, které interagují se vyvíjeným systémem za účelem dosažení konkrétního cíle. Mohou to být například uživatelé, administrátoři nebo jiné systémy, které jsou zapojeny do interakce se systémem. Identifikace aktérů v analýze případů užití je klíčová, protože umožňuje lepší pochopení zapojujících se stran a jejich vztahu k systému. Identifikace všech aktérů je prvotní krok pro návrh scénářů, které přesně reflektují potřeby a požadavky daných aktérů, a tím zajišťují, že systém bude efektivně plnit specifikované požadavky.[7] Z provedené analýzy domény přijímacího řízení 2.1 lze identifikovat řadu aktérů a to jsou uchazeči, studijní referenti, supervizoři, pracovníci z VIC a také externí systémy jako jsou KOS či Moodle.



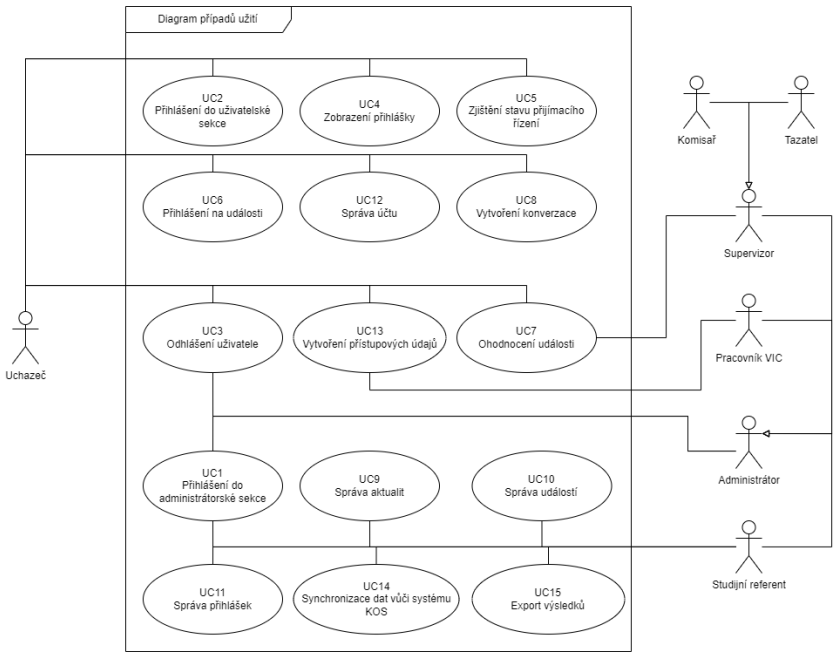
Obrázek 2.11 Přehled aktérů v doméně přijímacího řízení

2.4.2 Případy užití

K zachycení případu užití byla použita následující struktura scénářů:

1. `Identifikátor:` Kód scénáře pro účely jednoduššího odkazování.
2. `Název:` Krátký popisný název, který přesně odráží účel scénáře použití.
3. `Funkční požadavky:` identifikátory souvisejících funkčních požadavků.
4. `Aktéři:` Seznam primárních a sekundárních aktérů zapojených do scénáře.
5. `Předpoklady:` Podmínky, které musí být splněny před provedením případu použití.
6. `Kontext:` Událost nebo akce, která spouští scénář případu použití.
7. `Hlavní scénář:` Posloupnost interakcí mezi aktéry a systémem.
8. `Alternativní scénář:` Popis všech alternativních cest nebo scénářů, které se mohou v případě použití vyskytnout, obvykle kvůli výjimkám nebo odchylkám v hlavním scénáři.
9. `Výsledky:` Očekávané výsledky případu použití, včetně jakýchkoli změn stavu systému.
10. `Omezení:` Relevantní pravidla, zásady nebo omezení.

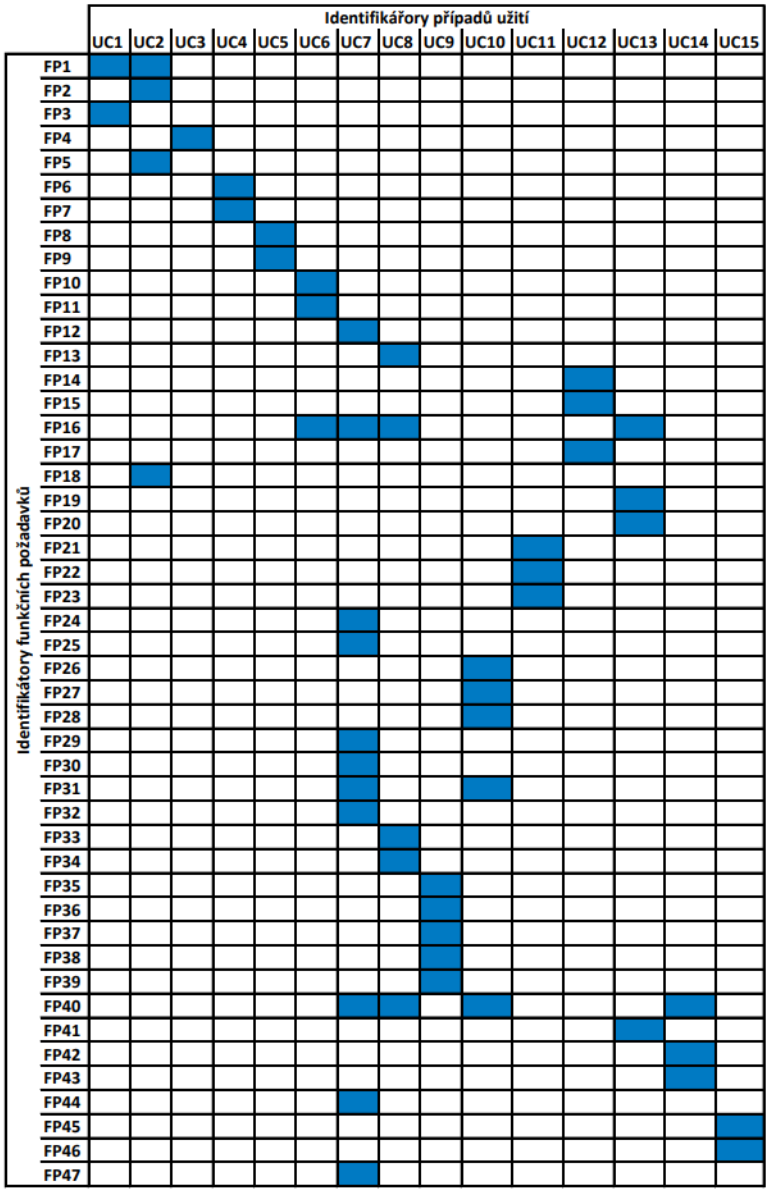
Textové scénáře případů užití nabízejí mnoho výhod oproti samotnému diagramu. Zlepšují srozumitelnost a porozumění tím, že poskytují podrobný popis interakcí a situací během případu užití, který nelze jednoduše vyčíst z pouhého diagramu případu užití. Scénáře rovněž napomáhají k validaci a testování systému, protože slouží jako základ pro tvorbu testovacích případů a zajišťují, že systém bude splňovat požadavky v reálných situacích. Podrobný textový popis všech scénářů případů užití lze najít v příloze B. Dále lze také na obrázku 2.12 vidět diagram případů užití, který byl vytvořen jako doplněk k textovému popisu, poskytující souhrnný pohled na celou oblast analýzy případů užití. [7]



Obrázek 2.12 Diagram případů užití

2.4.3 Pokrytí požadavků

V kontextu návrhu a analýzy systému je nezbytné pečlivě zohlednit všechny klíčové funkční požadavky, které mohou mít vliv na výsledné řešení. Při provádění analýzy případů užití však může být snadné nechtěně opomenout některé z těchto důležitých požadavků vzhledem k jejich množství. Aby bylo zajištěno, že všechny funkční požadavky jsou řádně zahrnuty a analyzovány, byla vytvořena speciální tabulka 2.13, která slouží k přesnému sledování pokrytí všech funkčních požadavků prostřednictvím případů užití. Tato tabulka je uspořádána tak, že každý řádek reprezentuje jeden konkrétní funkční požadavek a sloupce jsou jednotlivé případy užití. V tabulce lze vidět, že všechny funkční požadavky byly pokryty minimálně jedním případem užití.



Obrázek 2.13 Tabulka pokrytí požadavků pomocí případů užití

2.4.4 Odhad pracnosti

Při vývoji jakéhokoliv projektu je dobré vytvořit časový plán, který napomáhá dodržet slíbené lhůty, minimalizovat rizika či snižovat pravděpodobnost zpoždění projektu nebo jeho úplného selhání. Aby bylo toto možné, je nutné vytvořit odhad pracnosti vyvíjeného systému. Pro odhad pracnosti lze použít metodu *Use Case Points*, která se zakládá na počtu a složitosti případů užití a také na různých technických a environmentálních faktorech. Metoda UCP je založena na předpokladu, že úsilí potřebné k vývoji softwarového systému je přímo úměrné velikosti a složitosti systému, což lze odhadnout na základě scénářů případů užití. Výpočet se skládá ze tří částí — výpočet hrubé složitosti případů užití a aktérů, výpočet dopadu technických faktorů a na závěr výpočet dopadu faktorů prostředí [8].

Jako první krok je nutné spočítat složitosti aktérů, a proto jsou aktéři nejprve rozčleněni do tří skupin podle následujících vlastností:

1. `Jednoduchý:` Aktéři, jako jsou systémy s definovaným API, mají velmi jednoduché požadavky a zvyšují složitost případu použití pouze mírně.

2. `Průměrný:` Aktéři, jako jsou systémy komunikující prostřednictvím daných protokolů, mají více potřeb a zvyšují složitost případu použití do určité míry.

3. `Složitý:` Aktéři, jako jsou uživatelé komunikující prostřednictvím GUI, kteří mají výrazný dopad na složitost případu použití.

Na základě výše vysvětleného rozdělení do kategoriií, lze každému aktéru přiřadit určitou váhu. Jednoduchému aktérovi je přidělena váha 1, průměrnému váha 2 a složitému váha 3. Hrubou složitost aktérů *uaw* lze získat součtem vážených hodnot z jednotlivých kategorií. Výsledná hodnota hrubé složitost aktérů *uaw* činí 18, což je vypočteno v tabulce 2.1.

Tabulka 2.1 Výpočet hrubé složitosti aktérů

Typ aktéra	Složitost	Váha
Uchazeč	Složitý	3
Pracovník VIC	Složitý	3
Komisař	Složitý	3
Tazatel	Složitý	3
Studijní referent	Složitý	3
KOS	Jednoduchý	1
Moodle	Průměrný	2
Celkem		**18**

Obdobným způsobem je poté vypočítána hrubá složitost případů užití *uucw*, kde jsou váhy závislé na počtu transakcí, což v kontextu případů užití znamená počet vykonaných akcí v rámci daného scénáře. Stejně jako u aktérů jsou případy užití rozděleny do následujících kategorií:

1. `Jednoduchý:` 4 a méně transakcí, mají menší vliv na celkovou složitost projektu.

2. `Průměrný:` 4 - 7 transakcí, mají střední vliv na složitost projektu.

3. `Složitý:` více než 7 transakcí, mají největší vliv na celkovou složitost projektu.

Přidělení vah jsou pro případy odlišné než byly u aktérů — jednoduché případy užití mají váhu 5, průměrné váhu 10 a složité váhu 15. Hrubou složitost případů užití lze opět získat součtem vážených hodnot z jednotlivých případů užití. Výsledná hodnota hrubé složitost případů užití *uucw* činí 125, což lze vidět v tabulce 2.2.

Tabulka 2.2 Výpočet hrubé složitosti případů užití

Případ užití	Počet transakcí	Složitost	Váha
UC1	6	Průměrný	10
UC2	4	Jednoduchý	5
UC3	2	Jednoduchý	5
UC4	2	Jednoduchý	5
UC5	2	Jednoduchý	5
UC6	4	Průměrný	10
UC7	10	Složitý	15
UC8	9	Složitý	15
UC9	6	Průměrný	10
UC10	6	Průměrný	10
UC11	4	Jednoduchý	5
UC12	4	Jednoduchý	5
UC13	9	Složitý	15
UC14	4	Jednoduchý	5
UC15	4	Jednoduchý	5
Total			**125**

Po vypočítání hrubých složitostí je nezbytné stanovit technický faktor $tFactor$, který reprezentuje míru dopadu různých technických faktorů na celkovou složitost projektu. Úkolem je posoudit metodou UCP určených 13 faktorů spojených s technickými aspekty vyvíjeného systému. Hodnotící stupnice zahrnuje hodnoty od 0, což značí bezvýznamný vliv, až do hodnoty 5, která představuje velmi důležitý vliv faktoru. Výsledný technický faktor je roven sumě součinů váhy a důležitosti pro každý z faktorů. Po ohodnocení jednotlivých faktorů v tabulce 2.3, je celkových technický faktor $tFactor$ roven hodnotě 54.5.

Tabulka 2.3 Výpočet hodnoty technického faktoru

Popis	Váha	Důležitost	Faktor
Distribuovaný systém	2	3	6
Doba reakce nebo požadovaná rychlost zpracování	2	3	6
Efektivita koncového uživatele	1	4	4
Složitost vnitřního zpracování	1	5	5
Znovupoužitelnost kódu	1	4	4
Jednoduchost instalace	0.5	2	1
Jednoduchost užití	0.5	5	2.5
Přenositelnost	2	4	8
Snadnost změny	1	5	5
Souběžnost	1	5	5
Bezpečnost	1	5	5
Přístup pro třetí strany	1	3	3
Požadavek speciálního tréninku	1	0	0
Celkem			**54.5**

Vliv na odhad doby realizace projektu mají rovněž zkušenosti a dovednosti účastníků vývoje. Tyto vlivy jsou zahrnuty a reflektovány prostřednictvím faktoru prostředí $eFactor$. Postup jeho hodnocení je analogický postupu u technického faktoru — nejprve je posouzeno všech 8 faktorů souvisejících s prostředím, ve kterém vyvíjený systém vzniká, s použitím hodnot stupnice od 0 do 5. Po zhodnocení jednotlivých faktorů v tabulce 2.4 je faktor prostředí $eFactor$ roven 22,5.

Tabulka 2.4 Výpočet hodnoty faktoru prostředí

Popis	Váha	Důležitost	Faktor
Znalost vývojového procesu	1.5	3	4.5
Zkušnost s aplikacemi	0.5	4	2
Zkušenost s objektovou orientací	1	3	3
Dovednosti vedoucího analytika	0.5	4	2
Motivace	1	5	5
Vyváženost požadavků	2	3	6
Zaměstnanci na částečný úvazek	-1	0	0
Složitost programovacího jazyka	-1	0	0
Celkem			**22.5**

Na základě výše uvedených výpočtů lze získat konečný počet bodů případů užití *ucp* podle následujícího vzorce:

$$uucp = uaw + uucw$$

$$tf = 0,6 + (0,01 * tFactor)$$

$$ef = 1,4 + (-0,03 * eFactor)$$

$$ucp = uucp * tf * ef$$

Finální hodnota bodů, která dosahuje 118,7, lze využít pro výpočet odhadu pracovních hodin. Běžně se uvádí, že jeden bod případu užití je ekvivalentní 20 pracovním hodinám, tudíž lze získat odhad, který činí $118,7 * 20 = 2374$ pracovních hodin. Tato hodnota přibližně odpovídá 99 pracovním dnům, pokud se bere v úvahu standardní osmihodinová pracovní doba.[8].

2.5 Shrnutí

Pomocí obchodní analýzy byla důkladně zmapována doména a interní procesy přijímacího řízení pro anglické studijní programy. Na základě schůzek se studijním oddělením a analýzou obdobného řešení Přiříz byla validována a zdokumentována specifikace všech funkčních a nefunkčních požadavků, která byla navíc doplněna scénáři případů užití pro jednodušší pochopení. Na základě analýzy případů užití bylo ověřeno, že jsou ve funkčních požadavcích reflektovány všechny potřeby cílových skupin a vytvořen časový odhad realizace projektu, který pomůže definovat rozsah projektu a důležité termíny pro další vývoj.

Kapitola 3
Rešerše technologií

Výběr vhodné technologie představuje je kritický pro budoucí implementaci navrhovaného systému, neboť jednotlivé technologie disponují rozdílnými vlastnostmi, silnými a slabými stránkami. Tyto aspekty mohou výrazně ovlivnit jak kvalitu finálního řešení, tak i průběh celého vývojového procesu. Tato kapitola je zaměřena na komparativní analýzu populárních technologií pro vývoj webových klientů. Následně je z těchto technologií vybrána jedna, na kterou je provedena podrobná technická analýza. Tato analýza poskytne pevný základ pro implementační fázi vyvíjené aplikace.

3.1 Srovnání technologií pro vývoj webových klientů

V současném rychle se rozvíjejícím digitálním prostředí je k dispozici mnoho technologií pro frontendový vývoj, přičemž každá z nich má své vlastní přednosti a slabiny. V této podkapitole budou zkoumány různé technologické frameworky s cílem vybrat ten nejvhodnější pro vývoj navrhované aplikace. Vzhledem k obrovskému množství dostupných frameworků byly pro srovnání vybrány tři populární technologie v oblasti vývoje webových klientů, a to Vue, Angular a React, na základě jejich oblíbenosti a širokého uplatnění v praxi. [9]

3.1.1 Vznik a vývoj

Přestože historie nebo datum vzniku technologie nejsou vždy přímými ukazateli její vyspělosti, lze na základě jejího historického vývoje identifikovat opakující se vzory ve vydávání verzí, a lze také posoudit, jak často jsou zpětně nekompatibilní změny zaváděny či jak rychle jsou vydávány opravy nalezených chyb. Tyto faktory poskytují dobrý přehled o tom, jak se technologie vyvíjela a jaké má šance udržet se na trhu v blízké budoucnosti. Oba aspekty jsou důležité pro projekty, které usilují o dlouhodobě udržitelný vývoj aplikací s využitím dané technologie.

Angular, nejstarší z porovnávaných technologií, byl uveden na trh v říjnu 2010 společností Google. Pozoruhodná je však skutečnost, že od roku 2014 se začalo pracovat na jeho kompletním přepracování, což vedlo k vzniku Angular 2 v roce 2016. V době psaní této práce je Angular dostupný ve verzi 15.2.4, přičemž nové verze vycházejí zhruba každých šest měsíců s téměř týdenními aktualizacemi. Hlavní verze jsou podporovány přibližně 18 měsíců [10].

React, vyvinutý společností Facebook, byl zaveden na technologický trh v roce 2013. Rychle si získal popularitu díky své architektuře založené na komponentách a jednoduchosti ve vytváření komplexních uživatelských rozhraní. Společnost Facebook ho intenzivně využívá k vývoji svých vlastních produktů, jako jsou Facebook a Instagram. Stejně jako Angular prošel React během svého vývoje významnými změnami. V době psaní tohoto textu je React ve verzi 18.2.0 [11].

Vue je nejmladší z porovnávaných technologií pro vývoj webových klientů. Byl vyvinut v roce 2014 Evanem You, bývalým zaměstnancem společnosti Google. Tento framework si za poslední roky získal značnou popularitu, i když není podporován žádnou velkou společností. Nejnovější verze Vue je 3.2.47, přičemž nemá pevný plán pro vydávání verzí — opravné verze jsou publikovány podle potřeby, minor verze jsou obvykle vydávány s odstupem 3 až 6 měsíců a major verze jsou vydávány nepravidelně [12].

Přes své podobnosti byly tyto tři frameworky vyvinuty z různých důvodů. Je důležité znát tyto rozdíly, neboť na základě těchto informací lze odhadnout, jakým směrem se bude daná technologie v budoucnosti vyvíjet. Angular byl vytvořen s cílem pomáhat týmům a organizacím řešit problémy při tvorbě jednostránkových aplikací a poskytovat řešení široké škály problémů. React byl navržen pro tvorbu rychlých, jednoduchých a škálovatelných uživatelských rozhraní komplexních webových aplikací. Vue byl vytvořen jako lehká, progresivní alternativa k Angularu, která umožňuje rychle vytvořit vyvíjený projekt a zároveň poskytnout prostředky pro udržení škálovatelnosti s rostoucí složitostí aplikace. Všechny tři frameworky mají několikaletou historii a vyvinuly se do takové míry, že jsou dnes spolehlivé i pro větší podnikové aplikace [13].

3.1.2 Komunita a ekosystém

Popularita je důležitým ukazatelem zralosti zkoumané technologie, protože může napovědět, jak dobře se daná technologie osvědčila v praxi. Když je technologie populární, je pravděpodobné, že již byla úspěšně použita v reálném světě v produkčních aplikacích a je považována za efektivní řešení, které splňuje požadavky organizací.

Angular Websites

Countries	
United States	529,558
Germany	48,701
Brazil	36,996
United Kingdom	30,094
Russia	27,242
France	24,078
Italy	19,159
India	17,504
Netherlands	16,965
Canada	14,369

React Websites

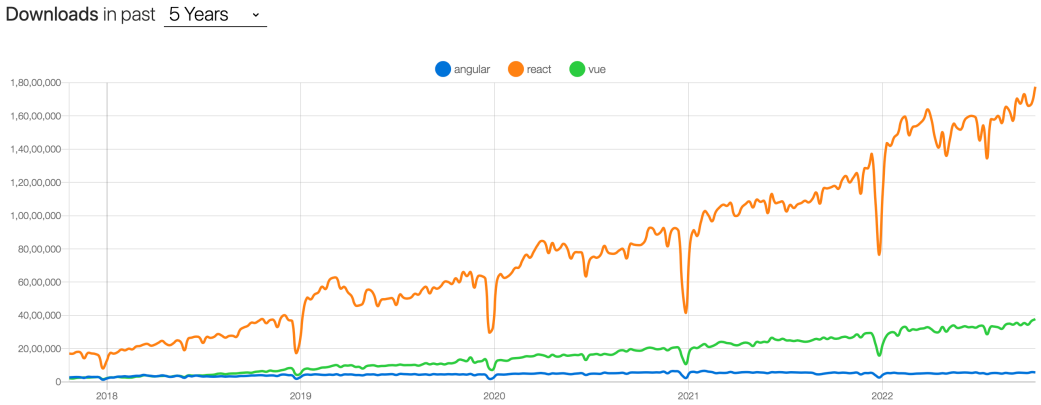
Countries	
United States	2,175,019
United Kingdom	563,184
Australia	263,980
Brazil	249,163
Canada	237,377
Germany	220,361
France	181,265
Russia	157,259
Japan	142,928
Italy	96,345

Vue Websites

Countries	
United States	1,107,006
Germany	59,064
Russia	57,561
United Kingdom	50,486
Brazil	49,897
China	42,337
Canada	32,686
Netherlands	28,347
Australia	28,139
France	27,099

Obrázek 3.1 Počet webových aplikací v závislosti na použitém frameworku a lokality [14]

Podle statistických dat z portálu BuiltWith, který analyzuje webové aplikace pomocí specializovaných algoritmů schopných rozpoznat použité technologie, je React z porovnávaných frameworků nejpoužívanější, následovaný Vue a s Angular na posledním místě, což lze pozorovat na obrázku 3.1. Tento trend je potvrzen i z jiné perspektivy grafem na obrázku 3.2, který zobrazuje počet stažení v průběhu času pomocí správce balíčků npm. Z grafu je patrné, že React má dlouhodobě největší počet stažení v komunitě vývojářů. Ačkoli jsou všechny tři knihovny poměrně populární, je React považován za nejpopulárnější a nejlépe přijatý technickou komunitou. Je důležité však poznamenat, že popularita webových frameworků se může lišit v závislosti na kontextu a konkrétních požadavcích projektu, jelikož různé frameworky mohou nabízet různé výhody a nevýhody. Je tedy potřeba zvážit konkrétní potřeby a kontext projektu před výběrem frameworku. [14]



Obrázek 3.2 Graf zobrazující počet stažení webových frameworků v závislosti v čase [14]

Popularita je nesporně důležitým aspektem, avšak může nastat situace, kdy široce využívaná technologie disponuje malým či se zmenšujícím ekosystémem. Tento stav je obecně vnímán jako indikace toho, že vývoj dané technologie směřuje nesprávným směrem, a přesto je stále používána pro vývoj aplikací. Tato situace může naznačovat, že technologie ztrácí podporu a v budoucnosti se pravděpodobně nebude tolik rozšiřovat o nové inovace. Ilustrativním příkladem této situace je knihovna jQuery, jež byla dříve široce využívána při vývoji webových aplikací, avšak postupně je nahrazována modernějšími technologiemi [15]. Ačkoliv může být obtížné určit velikost komunity okolo konkrétní technologie s definitivní jistotou, existuje řada indikátorů, které mohou poskytnout vhled do její velikosti a aktivity. Jeden z ukazatelů, který svědčí o úrovni podpory a šíři ekosystému dané technologie, je množství dostupných knihoven a npm balíčků. Tyto informace lze zaznamenat v tabulce 3.1. [1]

Tabulka 3.1 Hrubý počet dostupných knihoven

Framework	**hrubý počet npm knihoven**
Angular	66634
React	233801
Vue	76035

Dalším důležitým aspektem, který je třeba zvážit, je velikost core týmu, který daný framework udržuje a rozvíjí. Rozsáhlý core tým může signalizovat, že o technologii existuje značný zájem, což vedlo k vytvoření týmu, jenž se podílí na podpoře a pomoci komunitě. Větší core tým rovněž naznačuje, že framework bude pravděpodobně lépe udržován, aktualizován a že bude k dispozici více zdrojů pro vývojáře. V tomto ohledu je React na prvním místě s core týmem o 25 členech [16]. Těsně za ním následuje Angular se 24 členy, což je také velmi pozitivní ukazatel zájmu a podpory pro tento framework [17]. Na třetím místě se nachází Vue, jehož core tým sestává z 18 členů, což je sice menší počet než u ostatních technologií, avšak stále dostačující pro udržení a další rozvoj [18]. Na základě výše uvedených faktorů a informací lze usoudit, že všechny tři frameworky disponují silnými ekosystémy a základem podpory, přičemž React má výrazný náskok v počtu dostupných knihoven a velikosti core týmu. Angular zaujímá také silnou pozici, zatímco Vue, ač s menším základním týmem, si udržuje svoji pozici na trhu díky rychlému a snadnému vývoji aplikací.

[1] Hodnoty v tabulce byly získány na základě vlastního vyhledávání v online registru npm.

3.1.3 Klíčové koncepty

I když jsou všechny tři porovnávané frameworky založeny na jazyce JavaScript, každý z nich je postaven na odlišných konceptech a stylu programování. Pro rozhodnutí, který framework se nejvíce hodí pro vývoj navrhovaného systému z technického hlediska, je třeba nejprve porozumět klíčovým konceptům a posoudit, jaké jsou jejich silné a slabé stránky. Informace uvedené v této části byly získány z oficiálních dokumentací jednotlivých technologií. [19, 20, 21]

3.1.3.1 Angular

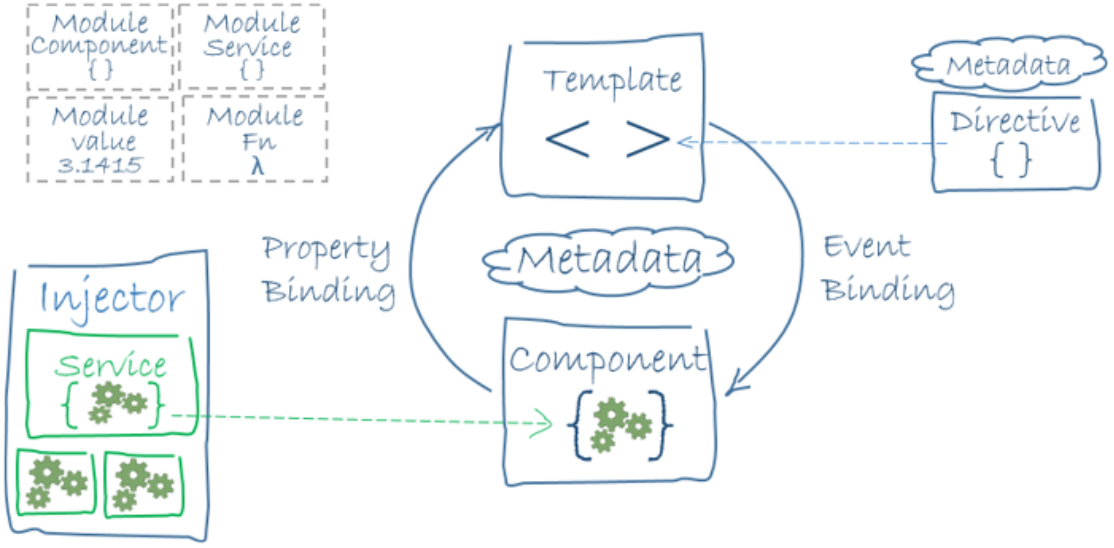
Angular je založen na jazycích HTML a TypeScript, který představuje rozšíření JavaScriptu o statické typování vyvinuté společností Microsoft a musí být zpětně přeložen do JavaScriptu, aby byl spustitelný v prohlížečích. Uživatelská rozhraní v Angularu jsou definována prostřednictvím šablon napsaných ve specializované HTML syntaxi, která přijímá téměř všechny HTML značky s několika výjimkami, například značka `script` je z bezpečnostních důvodů vyloučena, aby se zabránilo útokům typu *script injection*. Angular rozšiřuje běžné HTML tím, že umožňuje použití některých funkcí JavaScriptu přímo v HTML, například pomocí zápisu do dvojitých složených závorek.

Výpis kódu 3.1 Ukázka šablony ve frameworku Angular k vykreslení nákupního košíku

```
<div class="cart">
  <h2>Shopping Cart</h2>
  <div *ngIf="cart.length === 0">
    Your cart is empty.
  </div>
  <div *ngIf="cart.length > 0">
    <table>
      <thead>
        <tr>
          <th>Product</th>
          <th>Quantity</th>
          <th>Price</th>
        </tr>
      </thead>
      <tbody>
        <tr *ngFor="let item of cart">
          <td>{{item.name}}</td>
          <td>{{item.quantity}}</td>
          <td>{{item.price}}</td>
        </tr>
      </tbody>
    </table>
    <p>Total: {{getTotal()}}</p>
    <button (click)="checkout()">Checkout</button>
  </div>
</div>
```

Funkcionalita aplikací vytvořených v Angularu je organizována do modulů, které zapouzdřují kód určený pro specifickou aplikační doménu nebo úzce související sadu funkcionalit. Typicky má každý projekt alespoň jeden kořenový modul, obsahující logiku pro spuštění a nastavení aplikace, a poté aplikační moduly, které definují chování aplikace. Tento systém modulů umožňuje vytvářet znovu použitelné funkční jednotky a skládat je do komplexních struktur. Moduly pak komunikují mezi sebou tak, že každý modul může definovat sadu funkcí, které importuje pro vlastní užití, a sadu, které exportuje pro ostatní moduly.

Základním stavebním prvkem všech Angular aplikací jsou komponenty, které představují objektové třídy spojené s dříve popsanými šablonami a případnými CSS styly. Tímto způsobem lze části uživatelského rozhraní rozložit do samostatných komponent, které lze libovolně znovu použít. Pro definování aplikační logiky pro danou komponentu je možné vytvořit `service` třídy, které jsou s komponentou propojeny prostřednictvím vestavěného mechanismu *Dependency Injection*. Souhrn těchto konceptů lze vidět na obrázku 3.3.



Obrázek 3.3 Ilustrace zobrazující Angular komponentu a jejích dílčích částí [19]

Na obrázku je zobrazeno, že komponenta předává šabloně data prostřednictvím atributů, které šablona vykreslí v HTML pomocí specializovaných direktiv. Šablona odesílá zprávy zpět do komponenty, pokud nastane uživatelská událost, jako je například stisknutí tlačítka nebo vyplnění formuláře. Pokud je k obsluze požadavku potřebná další aplikační logika, komponenta deleguje požadavek `service` třídě, která obsahuje logiku pro obsluhu požadavku. Celý tento systém probíhá uvnitř izolovaného modulu a může případně využívat služby jiných modulů nebo poskytovat vlastní služby ostatním.

3.1.3.2 React

React je ze všech uvedených technologií nejjednodušší a zaměřuje se výhradně na tvorbu grafických komponent. Unikátní vlastnost knihovny React spočívá ve využívání speciální JSX syntaxe, která umožňuje zápis HTML kódu přímo do JavaScript kódu a tím vytváření šablon podobně jako v Angularu, označovaných jako komponenty. Tyto komponenty představují JavaScriptové struktury, vykreslující jeden konkrétní grafický prvek uživatelského rozhraní. Tvorba komplexního grafického rozhraní je kompozicí jednotlivých komponent do hiearchické stromové struktury. Definice vlastních komponent je velmi jednoduchá a přímočará, neboť komponenta se skládá pouze z JSX definice, atributů zvaných `props`, umožňujících předávání dat z vnějšku, a vnitřního stavu `state`, který komponenta uchovává uvnitř sebe. Data mohou být předávána z rodičovských komponent dětským komponentám, přičemž data jsou neměnná a nelze je předávat směrem nahoru v hierarchii stromu komponent. Stav je možné měnit, avšak při změně stavu se komponenta i její potomci znovu překreslí.

Výpis kódu 3.2 Ukázka komponenty ve frameworku React k vykreslení nákupního košíku

```
import React from "react";

function ShoppingCart({ cart, checkout }) {
  const getTotal = () =>
    cart.reduce((acc, item) => acc + item.price * item.quantity, 0);
  return (
    <div className="cart">
      <h2>Shopping Cart</h2>
      {cart.length === 0 && <div>Your cart is empty.</div>}
      {cart.length > 0 && (
        <div>
          <table>
            <thead>
              <tr>
                <th>Product</th>
                <th>Quantity</th>
                <th>Price</th>
              </tr>
            </thead>
            <tbody>
              {cart.map((item) => (
                <tr key={item.id}>
                  <td>{item.name}</td>
                  <td>{item.quantity}</td>
                  <td>{item.price}</td>
                </tr>
              ))}
            </tbody>
          </table>
          <p>Total: {getTotal()}</p>
          <button onClick={checkout}>Checkout</button>
        </div>
      )}
    </div>
  );
}
```

3.1.3.3 Vue

Podobně jako Angular, technologie Vue využívá HTML šablonovou syntaxi a data pocházející z JavaScriptové části, která mohou být předána do HTML prostřednictvím dvojitých složených závorek. Každá Vue aplikace se skládá z komponent představujících základní stavební prvky aplikace, které obsahují tři části — šablonu v HTML, CSS stylování a skript, v němž je řízen způsob práce s daty a logika chování komponenty. Aplikace jsou běžně organizovány do stromu zanořených komponent rovněž vytvářejících hierarchickou stromovou strukturu. Pro využití komponent, které nejsou definované lokálně, je nutné komponenty před použitím registrovat, přičemž komponenty lze registrovat jak globálně, tak lokálně. Globální registrace komponenty umožňuje přístup ke komponentě veškerým subkomponentám.

Data aplikace jsou uložena ve funkci komponenty `data`, která slouží k perzistenci stavu komponenty. Data mohou být předána přes atribut `prop` z rodičovských komponent dětským komponentám. Pokud je hodnota předána jako atribut `prop`, stává se datovým atributem dané instance komponenty, tudíž k `prop` lze přistupovat v komponentě stejně jako k vlastnímu datovému atributu pomocí klíčového slova `this`.

Výpis kódu 3.3 Ukázka šablony ve frameworku Vue k vykreslení nákupního košíku

```
<template>
  <div class="cart">
    <h2>Shopping Cart</h2>
    <div v-if="cart.length === 0">Your cart is empty.</div>
    <div v-if="cart.length > 0">
      <table>
        <thead>
          <tr>
            <th>Product</th>
            <th>Quantity</th>
            <th>Price</th>
          </tr>
        </thead>
        <tbody>
          <tr v-for="(item, index) in cart" :key="index">
            <td>{{item.name}}</td>
            <td>{{item.quantity}}</td>
            <td>{{item.price)}}</td>
          </tr>
        </tbody>
      </table>
      <p>Total: {{getTotal()}}</p>
      <button @click="checkout()">Checkout</button>
    </div>
  </div>
</template>

<script>
export default {
  data() {
    return {
      cart: [], // Cart array data
    }
  },
  methods: {
    getTotal() {
      // Method to calculate the total here
    },
    checkout() {
      // Method to handle the checkout button click here
    }
  }
}
</script>
```

3.1.4 Výkon

Výkon se považuje za jednu z nejdůležitějších oblastí při vývoji webových klientů, ať už je zvolen Angular, React, Vue nebo jakákoliv jiná technologie, neboť klade omezení na implementaci a má přímý vliv na použitelnost aplikace. V případě webových aplikací je výkon přímo spojen s *Document Object Model*, nebo zkáceně DOM, který reprezentuje webovou stránku v prohlížeči i v kódu. Díky DOM lze manipulovat s rozhraním webové aplikace a při jakýchkoli aktualizacích vykreslit případné změny.

Vue i React používají koncept nazývající se virtuální DOM, který je založen na principu kopie skutečné podoby DOM, na níž jsou prováděny veškeré změny bez přímého vlivu na originální podobu. Nastane-li v uživatelské rozhraní změna, postupuje se v následujících krocích:

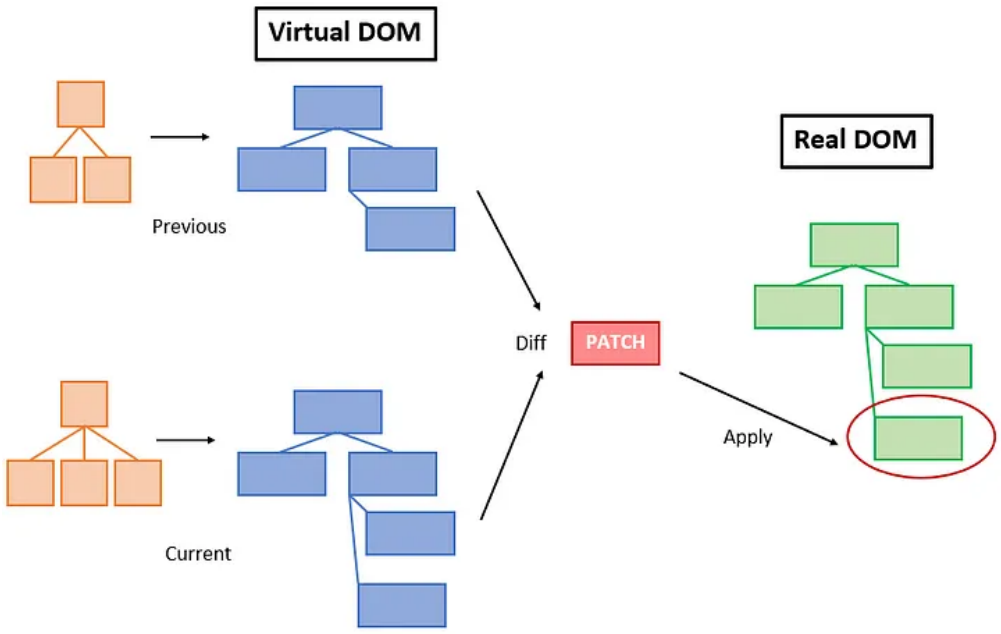
1. Po provedení změn v aplikaci se vygeneruje nový virtuální strom DOM, který představuje aktualizovaný stav aplikace.

2. Nový strom virtuálního DOM se pak porovná s předchozím stromem virtuálního DOM, aby se zjistily rozdíly nebo potřebné aktualizace.

3. Po zjištění rozdílů se vypočítá minimální sada operací DOM pro aktualizaci skutečného DOM, která se nazývá rekonciliace.

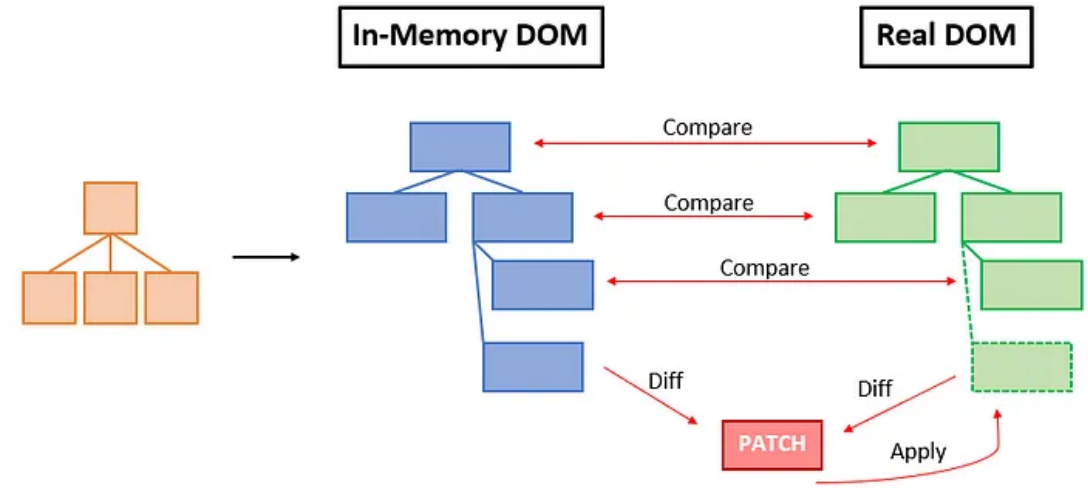
4. Nakonec se skutečný DOM aktualizuje pomocí změn vypočtených v předchozím kroku.

Tento způsob eliminuje potřebu překreslování celého grafického rozhraní aplikace při každé malé změně, což šetří výpočetní čas. Jako důsledek je však nutné počítat s tím, že virtuální DOM vykazuje vysokou spotřebu paměti, jelikož je nezbytné udržovat v paměti několik kopií skutečného DOM pro srovnávání a zjišťování změn oproti předchozí iteraci [22].



Obrázek 3.4 Znázornění překreslení grafického rozhrazní s pomocí virtuálního DOM [22]

Na druhou stranu je v Angularu používán framework s inkrementálním DOM. Na rozdíl od výše popsaného virtuálního DOM se zde nevytváří žádné mezilehlé stromové uspořádání a skutečný strom je upravován přímo na místě. Místo vytváření virtuální reprezentace celého uživatelského rozhraní, inkrementální DOM identifikuje konkrétní části rozhraní, které se skutečně změnily, vytváří virtuální reprezentaci těchto částí a následně aktualizuje skutečný DOM tak, aby odrážel nové změny. U inkrementálního DOM není vyžadován tak velký prostor v paměti, protože je paměť přidělována pouze pro detekci změn. Nevýhodou je však horší výpočetní výkon, jelikož inkrementální DOM musí provádět jemnější aktualizace ve srovnání s virtuálním DOM, který aktualizuje DOM ve větších dávkách [22].



Obrázek 3.5 Znázornění překreslení grafického rozhrazní s pomocí inkrementálního DOM [22]

3.1.5 Zhodnocení

Během analýzy byly prodiskutovány možnosti technologií Angular, React a Vue, které byly frameworky mezi sebou srovnány z různých perspektiv, a lze říci, že všechny technologie jsou velmi kvalitní z hlediska jednoduchosti vývoje, velikost komunity a výkonnosti. Avšak React se jeví jako nejvhodnější volba pro vývoj vlastního řešení z několika důvodů, které již byly zmíněny v provedené analýze:

- React je nejpopulárnější z porovnávaných frameworků, což naznačuje, že již byl úspěšně použit v mnoha produkčních aplikacích a je široce přijímán technickou komunitou. Popularita také znamená, že je větší pravděpodobnost nalezení řešení na případné problémy a otázky, které se mohou vyskytnout během vývoje.

- React má největší ekosystém z těchto tří frameworků, což zahrnuje velký počet knihoven a balíčků npm, které mohou usnadnit a urychlit vývoj. Silný ekosystém také znamená, že framework bude pravděpodobně dobře podporován v budoucnosti.

- React má největší core tým, což naznačuje, že je framework dobře udržován a aktualizován. Větší core tým také znamená, že budou k dispozici více zdrojů pro vývojáře

- React používá speciální JSX syntaxi, která umožňuje snadné a přímočaré vytváření komponent. Toto zjednodušuje vývoj a zvyšuje čitelnost kódu.

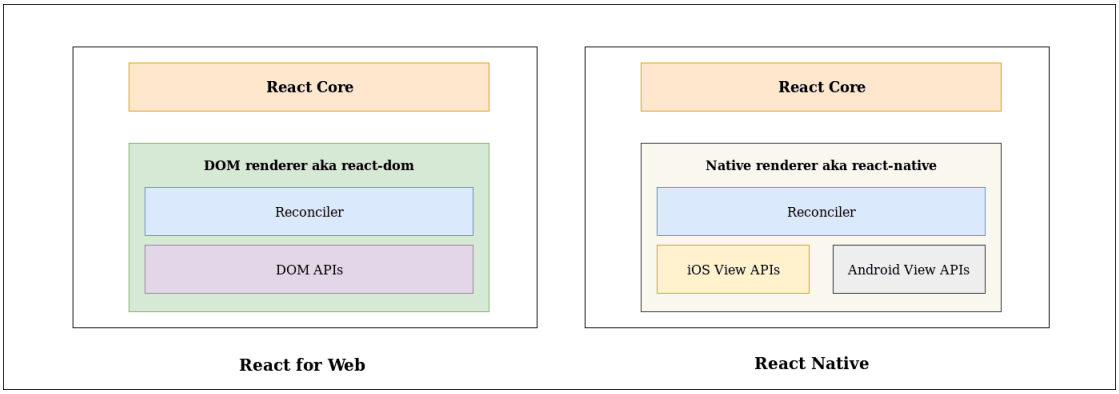
- React je vhodný pro širokou škálu projektů a může být snadno integrován do stávajících aplikací. Jeho flexibilita a modulární přístup umožňují vývojářům snadno experimentovat s různými architekturami.

3.2 Technická analýza knihovny React

Po provedení komparativní analýzy technologií pro vývoj webových klientů byla knihovna React vybrána jako nejvhodnější pro vývoj. Poté, kdy byl React identifikován jako ideální technologie, je důležité získat důkladné porozumění jeho technickým detailům, neboť tato znalost bude užitečná během implementační fáze. Tato podkapitola se bude věnovat porozumění technickým aspektům knihovny React.

3.2.1 Interní architektura

Modulární interní architektura knihovny React umožňuje implementovat podporu pro různé platformy i mimo oblast webových rozhraní. Dobrým příkladem je například varianta React Native, která je zejména cílená na vývoj mobilních aplikací. Základem každé platformě specifické implementace React jsou moduly React Core, Renderer a Reconciler [23].



Obrázek 3.6 Struktura implementace knihovny React pro různé platformy [23]

3.2.1.1 Core API

React core je soubor funkcí a tříd, kterými lze interagovat s knihovnou React. Core API neobsahuje žádný platformě specifický kód a je společný pro všechny varianty Reactu, ať už jde o variantu pro webové prohlížeči či mobilní vývoj. V React core API lze nalézt důležité funkce jako jsou například:

- `React.createElement`: Tato funkce je zodpovědná za vytváření instancí React elementů, které reprezentují komponenty nebo DOM elementy. Funkce přijímá typ elementu, vlastnosti `props` a potomky jako parametry.
- `React.createRef`: Tato funkce umožňuje vytvořit referenci ke konkrétní React komponentě a programaticky je modifikovat či přistupovat k jejím datům.
- `React.createClass`: Tato metoda je použita pro vytváření třídních komponent v Reactu ve starších verzích knihovny.
- `React.Component`: Je základní třídou pro všechny třídní komponenty v Reactu. Definuje životní cyklus komponenty a základní strukturu pro práci s vlastnostmi `props` a stavem `state`.
- `React.Children`: Je objekt, který poskytuje metody pro práci s potomky komponenty. To zahrnuje metody pro procházení potomků, mapování a počítání. [20]

3.2.1.2 Renderer

Jednou z nejpokročilejších funkcí Reactu je možnost psát vlastní moduly zvané vykreslovače *renderers* pro různá prostředí. Každý vykreslovač sestává ze dvou částí:

- `hostConfig`: HostConfig je rozhraní metod, které definují způsob interakce vykreslovače s cílovou platformou či prostředím. Poskytuje funkce, jak vytvářet, aktualizovat i odstraňovat instance prvků z uživatelského rozhraní, spravovat hierarchii a zpracovávat události.
- `reconciler`: Reconciler je základní algoritmus knihovny React, který se stará o proces porovnávání a aktualizace grafických prvků. Je zodpovědný za určení, které aktualizace je třeba provést v uživatelském rozhraní na základě změn ve stromu komponent. Reconciler je platformově nezávislý, což znamená, že může pracovat s jakýmkoli vykreslovacím nástrojem, který má kompatibilní HostConfig. [23]

V příkladu 3.4 je prezentována vlastní implementace vykreslovače React, která demonstruje základní strukturu a funkcionalitu vykreslovače. Tento vykreslovač se omezuje pouze na výpis datového typu komponenty a nezobrazuje žádné vizuální prvky. Objekt `HostConfig` obsahuje různé metody související s vykreslovacím procesem, avšak v tomto příkladu je demonstrováno využití pouze metody `createInstance`. Tato metoda je volána při vytváření instance vložené komponenty. V příkladu je funkcí `createInstance` vypsán typ prvku a je vrácen objekt obsahující `type` a `props`. Reconciler je vytvořen voláním funkce `Reconciler` s objektem `HostConfig`. Objekt `CustomRenderer` disponuje metodou `render`, která přijímá tři argumenty: `element`, `renderDom` a `callback`. `Element` představuje nejvyšší React komponentu určenou k vykreslení, `renderDom` je kontejner, do kterého bude umístěn vykreslený výstup, a `callback` je volitelná funkce, jež je volána po dokončení vykreslovacího procesu.

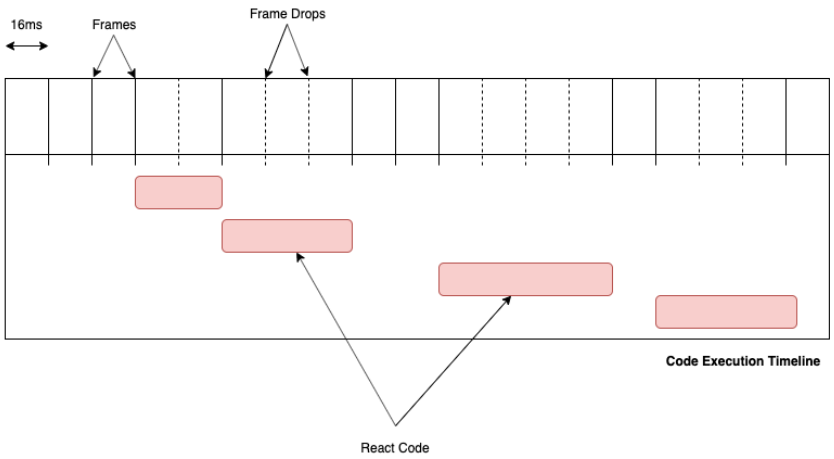
Výpis kódu 3.4 Ukázka vlastní implementace vykreslovače

```
const HostConfig = {
  createInstance(type, props, rootContainerInstance, hostContext) {
    console.log(`Element type: ${type}`);
    return { type, props };
  },
  ...
};
const reconciler = Reconciler(HostConfig);
const CustomRenderer = {
  render(element, renderDom, callback) {
    const isAsync = false;
    const container = reconciler.createContainer(renderDom, isAsync);
    const parentComponent = null;
    reconciler.updateContainer(
      element,
      container,
      parentComponent,
      callback
    );
  },
};
function App() {
  return (
    <div>
      <h1>Hello, Custom Renderer!</h1>
    </div>
  );
}
CustomRenderer.render(<App />, {});
```

3.2.1.3 Reconciler

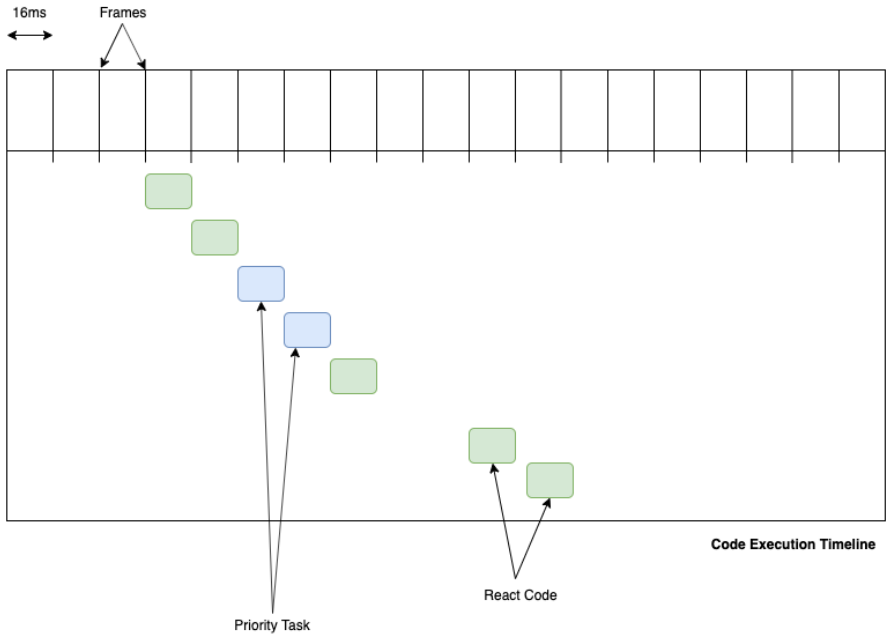
Jak bylo zmíněno v sekci 3.1.4 o porovnání výkonnosti technologií, React využívá pro optimalizaci vykreslování koncept virtuálního DOM. Udržuje dva stromy obsahující React elementy a zjišťuje změny mezi nimi, které se mají vykreslit. Pro porovnání stromů se používá *reconciler*, který je zodpovědný za implementaci porovnávacího algoritmu. Tato část knihovny může být sdílena napříč různými implementacemi React. V době psaní této práce existují dva základní typy reconcilerů - *stack reconciler* a *fiber reconciler*.

Stack reconciler, který pohání React verze 15 a starší, představuje první a původní implementaci porovnávacího algoritmu. V případě každé změny komponenty je vytvořena nová kopie stromové struktury komponent prostřednictvím rekurzivního průchodu aktuálním virtuálním stromem od vrcholu až po listy, kdy je pro každou komponentu volána její metoda `render`. Následně je pomocí prohledávání do hloubky procházen jak aktuální virtuální strom, tak i ten nově vytvořený, přičemž při detekci změny oproti původnímu stromu je současně aktualizován skutečný strom vykreslených komponent. Tento algoritmus, ačkoli jednoduchý, má jako hlavní nevýhodu přístup založený na synchronním procházení stromu. Může tak způsobit zpomalení či zastavení procesu vykreslování na několik vteřin u komponent s náročnými výpočetními operacemi, což vede k neresponzivnímu uživatelskému rozhraní, jak ilustruje obrázek 3.7. [24]



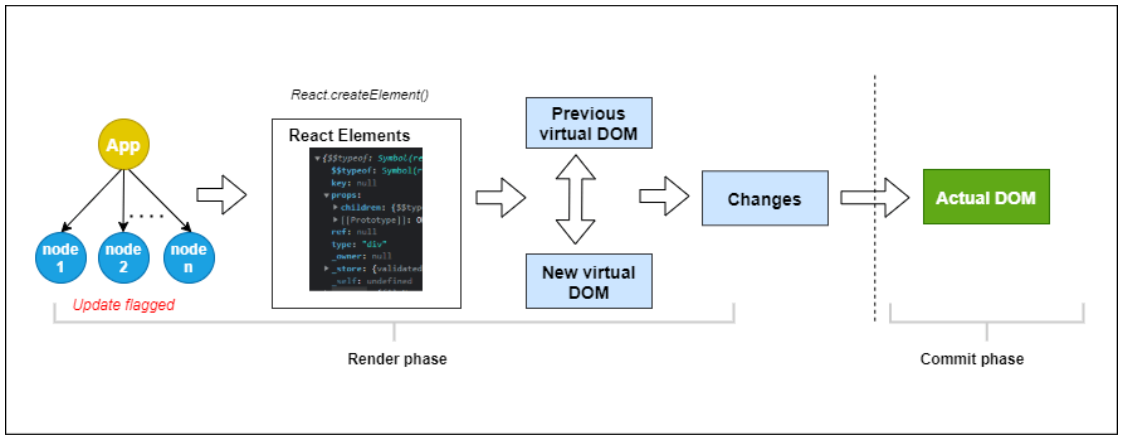
Obrázek 3.7 Ilustrace rychlosti vykreslování pomocí *stack reconciler* [25]

Fiber reconciler, poprvé uvedený v React verzi 16, přináší zásadní změny v porovnávacím algoritmu a zavádí novou úroveň kontroly nad prioritou aktualizací komponent. Hlavním cílem je zlepšení reaktivity aplikace díky rozdělení procesu aktualizace komponent na menší jednotky práce, jež mohou být plánovány a prováděny postupně. *Fiber reconciler* využívá speciální datovou strukturu nazvanou *Fiber*, která reprezentuje jednotku práce v procesu aktualizace komponent. Každá komponenta v aplikaci má svůj vlastní *Fiber*, jenž uchovává informace o aktuálním stavu komponenty, jejích potomcích, právě probíhajících změnách a dalších metadatech. *Fiber reconciler* pracuje s páry stromů *Fiber*, kde jeden strom zobrazuje aktuální stav aplikace a druhý strom nový, aktualizovaný stav. *Fiber reconciler* porovnává oba stromy a identifikuje rozdíly mezi nimi, což umožňuje provést pouze nezbytné aktualizace skutečného DOM. Každé jednotce práce je přiřazena priorita, která určuje důležitost dané aktualizace. Vyšší priorita znamená, že aktualizace by měla být provedena co nejdříve. React může plánovat a provádět jednotky práce asynchronně, což zajišťuje, že uživatelské rozhraní zůstává reaktivní i při náročných operacích. Během procházení stromu komponent může *Fiber reconciler* přerušit práci na méně důležitých aktualizacích, pokud se objeví aktualizace s vyšší prioritou. Tímto způsobem je umožněno rychleji reagovat na akce uživatele a zajistit plynulý běh aplikace. [24]



Obrázek 3.8 Ilustrace rychlosti vykreslování pomocí *fiber reconciler* [25]

Celý proces vykreslování lze shrnout obrázkem 3.9, na kterém jde vidět, že celý proces vykreslování je rozdělen na *render* a *commit* fázi. V *render* fázi jsou při změně stavu jakékoliv komponenty je zavolána funkce `createElement`, která vytváří nový virtuální strom komponent. Poté je zavolán *reconciler*, který porovnává nově vytvořený strom s aktuálním stromem a identifikuje změny, které je třeba provést. Jak bylo zmíněno, *Fiber reconciler* rozděluje tuto práci na menší jednotky s různými prioritami a plánuje je asynchronně. V *commit* fázi se provádí skutečné aktualizace DOM na základě změn identifikovaných reconcilerem. Tyto aktualizace zahrnují přidávání, aktualizaci nebo odstraňování uzlů DOM a aplikaci změn stavu a vlastností komponent. Po dokončení *commit* fáze je uživatelské rozhraní aktualizováno a reaguje na změny stavu či interakci uživatele. [26]



Obrázek 3.9 Schéma procesu vykreslování v React [26]

3.2.2 Javascript XML

JSX, což je zkratka pro Javascript XML, představuje rozšíření standardu ECMAScript 6, jež umožňuje používat v kódu JavaScriptu syntaxi velmi podobnou HTML. Toto rozšíření je pouze syntaktickou zkratkou, která napomáhá psaní jednoduššího a přehlednějšího kódu. Vzhledem k tomu, že současné prohlížeče podporují pouze čistý JavaScript, je nezbytné přeložit XML zápis pomocí transpilátorů. Transpilátory jsou specializované překladače, které překládají zdrojový kód z jednoho jazyka do jiného, přičemž zachovávají stejnou úroveň abstrakce. Pro tento účel se v praxi často využívají například Babel či TypeScript. Celý proces transpilace probíhá během vývoje nebo sestavení aplikace, což znamená, že prohlížeč není vědom o použití JSX a spouští pouze kód obsahující objekty popsané pomocí React API. [20]

Následující jednoduchá komponenta vykreslí v prohlížeči tlačítko a navrací objekt, jenž vypadá jako HTML element, kterým však ve skutečnosti není:

Výpis kódu 3.5 Kód před přeložením JSX

```
import React from 'react'

function ButtonComponent() {
  return (
    <Button color="blue" shadowSize={2}>
      Click Me
    </Button>
  );
}

export default ButtonComponent;
```

Tato jednoduchá komponenta je pomocí transpilátoru přeložena na kód, který volá JavaScriptovou funkci `React.createElement`:

Výpis kódu 3.6 Kód po přeložení JSX

```
import React from 'react'

function ButtonComponent() {
  return <React.createElement(
      Button,
      {color: 'blue', shadowSize: 2},
      'Click Me'
    );
}

export default ButtonComponent;
```

Funkce `createElement` přijímá tři parametry, které jsou datový typ vykresleného elementu, předané atributy a jeho případné potomky. Po zavolání funkce se navrací JavaScriptový objekt popisující grafický element ve virtuálním DOM, jež se využívá pro optimalizaci vykreslování. Teoreticky lze psát kód v React bez jakéhokoliv použití JSX, avšak použití JSX značně urychluje proces vývoje a ulehčuje ladění chyb.

3.2.3 Komponenty

Během vývoje webových aplikací s využitím frameworku React je prvním krokem rozdělení uživatelského rozhraní na menší izolované grafické prvky, pro které je následně v Reactu implementována vlastní komponenta. Oproti vytváření jednoho rozsáhlého grafického prvku nabízí tento modulární přístup řadu výhod, jako jsou jednoduchost, recyklovatelnost, škálovatelnost a udržitelnost. Jak již bylo zmíněno při analýze klíčových konceptů webových frameworků, základním stavebním kamenem Reactu jsou právě komponenty, které lze implementovat dvěma způsoby — pomocí třídních či funkčních komponent.

3.2.3.1 Třídní komponenty

Třídní komponenty představují jeden ze dvou způsobů definování znovupoužitelných grafických prvků v technologii React. Tyto komponenty byly zavedeny v raných verzích Reactu jako jediná metoda pro vytváření komponent. Avšak postupem času byly značně nahrazeny funkčními komponentami. Přesto třídní komponenty stále zůstávají důležitou součástí ekosystému React a jejich pochopení je nezbytné.

Třídní komponenty jsou definovány rozšiřováním třídy `React.Component`, která je poskytována přímo technologií React. Pro tento účel je využito dědičnosti ze standardu ES6, což umožňuje třídní komponentě přistupovat k metodám a vlastnostem třídy `React.Component`. Nejdůležitější metodou je `render`, která je zodpovědná za vrácení React elementu pro reprezentaci prvku v DOM. Metoda `render` je automaticky volána knihovnou React, kdykoli se změní stav nebo vlastnosti komponenty, což v důsledku způsobí její překreslení.

Správa stavu je další důležitou funkcí třídních komponent. Stav komponenty je objekt, který uchovává dynamická data specifická pro danou komponentu. Inicializuje se v konstruktoru komponenty pomocí atributu `this.state`. Pro úpravu stavu se používá metoda `this.setState`, jež zajistí, že React je informován o změně a může komponentu aktualizovat.

Pro třídní komponenty jsou také důležité takzvané `props`, které se používají k předávání dat z nadřazené komponenty do podřízené komponenty. Jsou přijímány jako objekt v podřízené komponentě a jsou přístupné prostřednictvím třídního atributu `this.props`. `Props` jsou pouze pro čtení a neměly by být přímo upravovány podřízenou komponentou.

Užitečné jsou také metody životního cyklu, které umožňují spouštět kód v různých bodech životního cyklu komponenty. Některé z nejčastěji používaných metod životního cyklu jsou:

- `componentDidMount`: Voláno poté, co byla komponenta vložena do DOM. Často se používá k načítání dat nebo k provádění jiných nastavovacích operací.
- `componentDidUpdate`: Voláno vždy, když je aktualizován stav nebo `props` komponenty. Tato metoda může být použita k provádění vedlejších efektů, jako je aktualizace DOM nebo provádění síťových požadavků, na základě změn ve stavu.
- `componentWillUnmount`: Voláno bezprostředně před odstraněním komponenty z DOM. Používá se k uvolnění jakýchkoli prostředků nebo posluchačů, které byly vytvořeny během životního cyklu komponenty.

Kód 3.7 demonstruje všechny výše představené koncepty. Jedná se o třídní komponentu zobrazující počítadlo, které překresluje každou sekundu textovou hodnotu uplynulého času. Kód demonstruje využití specializovaných metod životního cyklu komponenty pro nastavení a uvolnění časovače — v metodě `componentDidMount` je nastaven časovač, který každou vteřinu volá funkci `tick`, jež pomocí `setState` aktualizuje stav komponenty. Metoda `componentDidUpdate` při každé změně stavu zapisuje do vývojové konzole textovou hodnotu uplynulého času a metodou `componentWillUnmount` se uvolní objekt časovače při odstranění komponenty.

Výpis kódu 3.7 Ukázka třídní React komponenty

```
import React from 'react';

class TimerComponent extends React.Component {

  constructor(props) {
    super(props);
    this.state = {
      seconds: 0,
    };
  }

  componentDidMount() {
    this.timerID = setInterval(() => this.tick(), 1000);
  }

  componentDidUpdate(prevProps, prevState) {
    if (prevState.seconds !== this.state.seconds) {
      console.log('Seconds updated:', this.state.seconds);
    }
  }

  componentWillUnmount() {
    clearInterval(this.timerID);
  }

  tick() {
    this.setState((prevState) => ({
      seconds: prevState.seconds + 1,
    }));
  }

  render() {
    return (
      <div>
        <h1>Elapsed time: {this.state.seconds} seconds</h1>
      </div>
    );
  }

}

export default TimerComponent;
```

3.2.3.2 Funkční komponenty

Funkční komponenta v React je jednoduchá JavaScriptová funkce, která vrací JSX. Ve verzi React 16.8, před vydáním funkce *React Hooks*, jež bude vysvětleno později v této sekci, se funkční komponenty často označovaly jako bezstavové komponenty, jelikož pouze přijímaly a vracely data, která se vykreslovala do DOM bez zpracování jakéhokoli stavu.

Funkční komponenty se staly populární díky své jednoduchosti a čitelnosti. Jsou obvykle kratší a snáze udržitelné než třídní komponenty. Navíc díky zavedení *React Hooks* se funkční komponenty staly kompletním řešením pro vytváření komponent v React, protože mohou také spravovat stav a zároveň pracovat také s vedlejšími efekty.

Výpis kódu 3.8 Ukázka funkční React komponenty

```
import React from 'react';

function NameComponent(props) {
  return (
    <div>
      <h1>Hello, {props.name}!</h1>
    </div>
  );
}

export default GreetingComponent;
```

Výše uvedený Kód 3.8 ukazuje jednoduchý příklad funkční komponenty, která zobrazuje uvítací zprávu se jménem uživatele. Komponenta přijímá `props` jako argument a vrací JSX, který má být vykreslen do DOM.

React Hooks jsou funkcionalita představená v Reactu 16.8, která umožňuje funkčním komponentám spravovat stav a provádět efekty, které byly dříve možné pouze v třídních komponentách. Hooks jsou speciální funkce, které se volají uvnitř funkčních komponent a mohou pracovat s jejich životním cyklem a vnitřními stavy.

Nejběžnějšími hooks knihovny React jsou:

- `useState`: Tento hook umožňuje funkční komponentě spravovat svůj vlastní stav. Vrací dvojici hodnot — aktuální stav a funkci pro aktualizaci stavu.
- `useEffect`: Tento hook umožňuje provádět efekty jako například načítání dat nebo aktualizace DOM v závislosti na změnách stavu nebo `props`. Nahrazuje metody životního cyklu `componentDidMount`, `componentDidUpdate` a `componentWillUnmount` v třídních komponentách.

Výpis kódu 3.9 Ukázka funkční React komponenty s Hooks

```
import React, { useState, useEffect } from 'react';

function TimerComponent() {
  const [seconds, setSeconds] = useState(0);

  useEffect(() => {
    const timerID = setInterval(() => {
      setSeconds(prevSeconds => prevSeconds + 1);
    }, 1000);

    return () => {
      clearInterval(timerID);
    };
  }, []);

  return (
    <div>
      <h1>Elapsed time: {seconds} seconds</h1>
    </div>
  );
}

export default TimerComponent;
```

Kód 3.9 ukazuje příklad funkční komponenty v Reactu, která používá hooky. Tento příklad představuje přepis dřívějšího příkladu s třídní komponentou a zobrazuje časovač s uplynulým časem. V kódu je použit hook `useState` pro správu stavu a `useEffect` hook pro nastavení a uvolnění časovače. Stav `seconds` je inicializován na hodnotu 0 a každou vteřinu aktualizován prostřednictvím funkce `setSeconds`. JavaScriptová metoda `setInterval` umožňuje spuštění časovače, který periodicky volá funkci aktualizuje hodnotu času pomocí `setSeconds` každou vteřinu. Prvotní inicializace časovače probíhá uvnitř hooku `useEffect`, zatímco uvolnění zdrojů je zajištěno navrácenou funkcí, která se zavolá při odstranění komponenty z DOM. Vracená funkce z hooku `useEffect` zruší časovač pomocí `clearInterval` před odstraněním komponenty.

Funkční komponenty společně s *React Hooks* poskytují silný, flexibilní a jednoduchý způsob, jak vytvářet a spravovat komponenty v React. Jejich jednoduchost a čitelnost zvyšují udržitelnost kódu a zjednodušují komplexní aplikace. V současné době se doporučuje používat pouze funkční komponenty a hooks pro většinu nových projektů, avšak je důležité poznamenat, že třídní komponenty mají své místo v ekosystému React a jejich znalost je stále užitečná.

3.2.4 Správa stavu

Správa stavu je klíčovým aspektem vývoje webových aplikací v Reactu, protože umožňuje udržet a manipulovat s daty v komponentách a mezi nimi. V Reactu lze spravovat stav na několik různých způsobů, z nichž některé jsou přirozeně součástí samotného Reactu, jako je použití třídních komponent a hooks, zatímco jiné jsou externí knihovny, které poskytují pokročilé řešení pro správu stavu, jako je například knihovna Redux.

3.2.4.1 Lokální stav

Správa lokálního stavu se zabývá uchováváním a aktualizací dat, která ovlivňují pouze určitou komponentu. Jak bylo popsáno v předchozí části 3.2.3 o komponentách, stav je v třídních komponentách uložen v objektu `this.state` a aktualizace jsou prováděny prostřednictvím metody `this.setState`. Ve funkčních komponentách je pro správu lokálního stavu využíván `useState` hook. Tyto mechanismy jsou nezbytné pro uchovávání lokálních dat, jelikož nejsou standardně perzistentní a při každém překreslení se komponenta celá znovu vytvoří, což má za důsledek ztrátu neuložených dat. Lokální stav je nutné řešit pro komponenty, které potřebují ukládat a aktualizovat svá vlastní data, jako jsou například hodnoty ve formulářových polích. Pokud je třeba sdílet data mezi více komponentami, lze využít Context API nebo knihovny třetích stran, jako je Redux.

3.2.4.2 Context API

Často je nutné předávat lokální stav komponenty i jejím potomkům — názorným příkladem může být komponenta zajišťující autentizaci uživatelů, která předává informace o uživateli jejím potomkům. Pro tyto účely poskytuje knihovna React funkci Context API pro sdílení dat mezi více komponentami bez nutnosti manuálního předávání dat pomocí `props`. To usnadňuje práci se sdílenými daty v aplikacích, které mají hlubokou hierarchii komponent.

Pro vytvoření kontextu se využívá funkce `React.createContext`, která vrací objekt kontextu s dvěma speciálními komponentami — `Provider` a `Consumer`. `Provider` komponenta umožňuje definovat hodnotu kontextu, která bude dostupná pro všechny komponenty v jeho podstromě. `Consumer` komponenta slouží k přístupu k hodnotě kontextu. Ve funkčních komponentách je možné pro přístup k hodnotě kontextu použít `useContext` hook místo komponenty `Consumer`. Tento hook přijímá kontext jako argument a vrací aktuální hodnotu kontextu.

Výpis kódu 3.10 Ukázka předávání stavů pomocí Context API

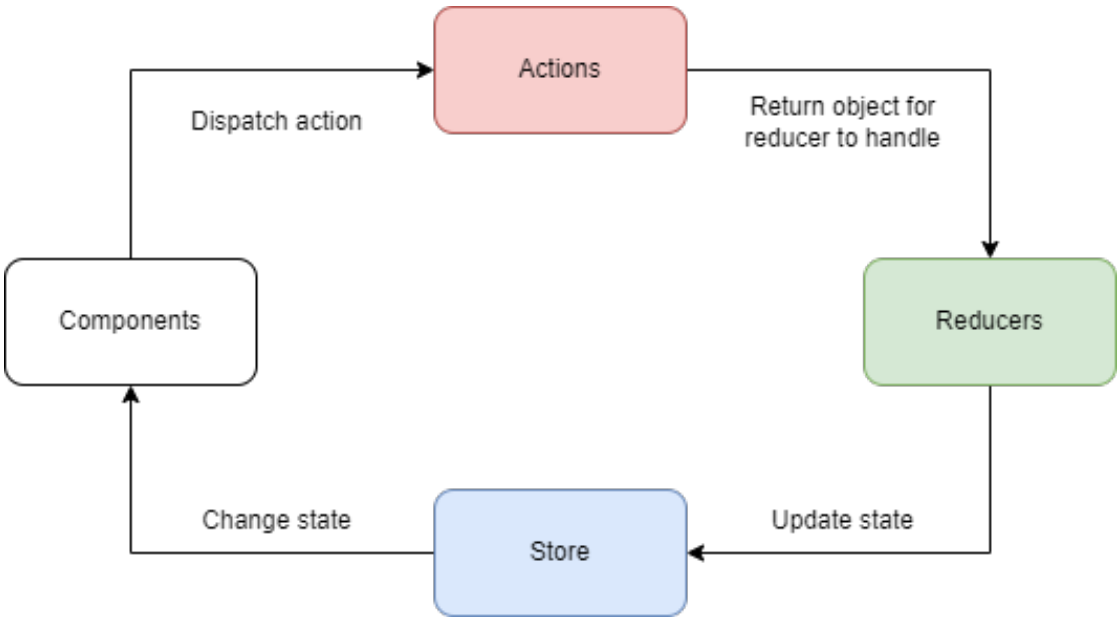
```
// Create authentication context
const AuthenticationContext = React.createContext();

// Create authentication provider component
const AuthenticationProvider = ({ children }) => {
  const [authenticated, setAuthenticated] = useState(false);
  const toggleAuthentication = () => {
    setAuthenticated(!authenticated);
  };

  return (
    <AuthenticationContext.Provider
      value={{ authenticated, toggleAuthentication }}
    >
      {children}
    </AuthenticationContext.Provider>
  );
};

// Create a sample component that uses the authentication context
const SampleComponent = () => {
  const { authenticated, toggleAuthentication } =
    useContext(AuthenticationContext);

  return (
    <div>
      <h1>{authenticated ? 'Authenticated' : 'Not Authenticated'}</h1>
      <button onClick={toggleAuthentication}>
        Toggle Authentication
      </button>
    </div>
  );
};

const App = () => {
  return (
    <AuthenticationProvider>
      <SampleComponent />
    </AuthenticationProvider>
  );
};
```

V ukázce kódu 3.10 byl vytvořen kontext autentizace `AuthenticationContext` pomocí funkce `React.createContext`. Následně byla definována komponenta `AuthenticationProvider`, ve které je obsažen lokální stav `authenticated` a funkce `toggleAuthentication` pro změnu stavu autentizace. Tato komponenta používá speciální komponentu `Provider`, která pomocí kontextu poskytuje objekt obsahující lokální stav včetně funkce pro její modifikaci. Poté byla pro demonstraci vytvořena komponenta `SampleComponent`, která pomocí `useContext` získává přístup k hodnotě kontextu a k funkci pro změnu stavu autentizace. Tato komponenta zobrazuje aktuální stav autentizace a tlačítko pro jeho změnu. V komponentě `App` byla nakonec použita komponenta `AuthenticationProvider` jako obalová komponenta pro `SampleComponent`, díky čemuž komponenta i její další potomci získá kontext autentizace. Tento příklad demonstruje, jak lze využít Context API pro sdílení a aktualizaci stavu mezi více komponentami bez nutnosti manuálního předávání dat pomocí `props`.

3.2.4.3 Globální stav

Správa globálního stavu se týká ukládání a aktualizace dat, která mají být přístupná napříč celou aplikací a různými komponentami. Větší aplikace často vyžadují složitější a více centralizovanou správu stavu, která zahrnuje koordinaci mezi více komponentami a jejich stavy. Pro tyto účely lze využít externí knihovny, jako je Redux, nebo v některých případech samotné Context API. Jedním z příkladů globálního stavu může být nákupní košík v e-commerce aplikaci, který by měl být přístupný z různých částí aplikace, jako je hlavní stránka, stránka s produkty, detail produktu a samotná stránka nákupního košíku.

Pro zajištění dostupnosti těchto dat napříč celou aplikací může být použit Redux, což je knihovna pro správu stavu aplikace, která usnadňuje práci s globálním stavem a umožňuje centralizovanou správu dat. Redux je nezávislý na použité knihovně či frameworku, ale je velmi oblíbený v kombinaci s Reactem. Redux se skládá ze tří částí: `store`, `actions` a `reducers`. Jejich vzájemnou komunikaci a interakci lze pozorovat na diagramu v obrázku 3.10.



Obrázek 3.10 Diagram zobrazující komunikaci částí Redux

`Store` je ústřední část Reduxu, která udržuje celý globální stav aplikace. Může být chápán jako jediný objekt, který uchovává stav všech komponent. `Store` umožňuje přístup k datům pomocí metody `getState`, aktualizaci stavu prostřednictvím metody `dispatch` a registraci posluchačů pomocí metody `subscribe`.

`Actions` jsou prosté JavaScriptové objekty, které popisují, jaký typ změny se má provést na stavu aplikace. `Actions` obsahují typ, který je jednoznačný řetězec popisující, co se má stát, a `payload`, který obsahuje data, která mají být použita při aktualizaci stavu. `Actions` jsou vytvořeny pomocí *action creators*, což jsou funkce, které vracejí `action` objekty.

`Reducers` jsou JavaScriptové funkce, které přijímají aktuální stav a `action` jako argumenty a vrací nový stav. `Reducers` jsou zodpovědné za aktualizaci stavu v reakci na přijatou `action`. V Redux lze mít více reducerů, které řeší různé části stavu. Tyto reducery lze poté kombinovat do jednoho hlavního reduceru pomocí funkce `combineReducers`.

Výpis kódu 3.11 Ukázka využití Redux pro implementaci nákupního košíku

```
// actions
export const addToCart = (product) => ({
  type: "ADD_TO_CART",
  payload: product,
});
export const removeFromCart = (productId) => ({
  type: "REMOVE_FROM_CART",
  payload: productId,
});

const cartReducer = (state = [], action) => {
  switch (action.type) {
    case "ADD_TO_CART":
      return [...state, action.payload];
    case "REMOVE_FROM_CART":
      return state.filter((item) => item.id !== action.payload);
    default:
      return state;
  }
};

const App = () => {
  const cart = useSelector((state) => state.cart);
  const dispatch = useDispatch();
  const handleAddToCart = () => {
    const product = { id: 1, name: "Product 1", price: 100 };
    dispatch(addToCart(product));
  };
  const handleRemoveFromCart = () => {
    const productId = 1;
    dispatch(removeFromCart(productId));
  };
  return (
    <div>
      <button onClick={handleAddToCart}>Add to cart</button>
      <button onClick={handleRemoveFromCart}>Remove from cart</button>
      <ul>
        {cart.map((item) => (
          <li key={item.id}>{item.name}</li>
        ))}
      </ul>
    </div>
  );
};

const rootReducer = combineReducers({
  cart: cartReducer,
});
const store = createStore(rootReducer);
ReactDOM.render(
  <Provider store={store}>
    <App />
  </Provider>,
  document.getElementById('root')
);
```

V ukázce kódu 3.11 je představena jednoduchá aplikace React, která implementuje nákupní košík s využitím Reduxu pro správu stavu. Reducer `cartReducer` zpracovává akce `ADD_TO_CART` a `REMOVE_FROM_CART`, které jsou exportovány jako funkce `addToCart` a `removeFromCart`. Tyto akce slouží k přidání a odebrání produktu z košíku. V komponentě `App` je použit hook `useSelector` k získání aktuálního stavu košíku a hook `useDispatch` pro zpětné odesílání akcí `addToCart` a `removeFromCart`. Reducer `cartReducer` je funkce, která na základě zpracovávané akce vrátí nový stav košíku. Pro přidání produktu do košíku je použita akce `ADD_TO_CART`, která do stavu přidává nový produkt. Opačně, akce `REMOVE_FROM_CART` odebere produkt z košíku podle jeho ID. Funkce `combineReducers` slouží ke kombinování více reducerů do jednoho hlavního reduceru. V tomto případě je použit pouze jeden reducer, `cartReducer`, který je kombinován do `rootReducer`. V hlavní komponentě `App` jsou dále definovány dvě funkce `handleAddToCart` a `handleRemoveFromCart`, které ovládají přidávání a odebírání produktů z košíku. Tyto funkce jsou přiřazeny tlačítkům, která spouštějí akce přidání a odebrání produktů. Seznam produktů v košíku je zobrazen pomocí mapování stavu `cart` na jednotlivé položky seznamu. Pro každý produkt v košíku je vytvořen seznamový prvek s názvem produktu.

3.3 Shrnutí

Na základě komparativní analýzy, která se zaměřila na srovnání technologií Angular, React a Vue, bylo rozhodnuto o výběru nejvhodnější technologie pro vývoj vlastního řešení. Při hodnocení byly zohledněny různé aspekty, jako je historie, popularita, velikost komunity a jednoduchost použití. Výsledkem analýzy bylo rozhodnutí pro knihovnu React, která byla díky svým vlastnostem vybrána jako optimální technologie pro účely vývoje vlastní aplikace. Po výběru technologie React následovala její technická analýza, která zahrnovala prozkoumání interních mechanizmů, možností implementace a řešení problémů různé škály. Důkladné zkoumání této knihovny umožnilo získat podrobnější pochopení jejích funkcí a schopností.

Kapitola 4
Návrh řešení

Návrh důležitý pro vytvoření aplikace, která splňuje všechna očekávání uživatelů a zároveň je jednoduchá na údržby či má nižší provozní náklady. Tato kapitola je zaměřena na definování aplikační struktury frontendové části a následném návrhu infrastruktury celého systému. Poté je zbytek kapitoly zaměřen na detailní návrh uživatelského rozhraní.

4.1 Aplikační architektura

Vývoj a údržba softwaru bez dobře definované architektury může snadno přerůst v nezvládnutelný proces. Toto může způsobit zvýšení nákladů, prodloužení doby potřebné k vývoji a potíže při implementaci nových funkcí či úpravách stávajících. Dobrá architektura umožňuje software lépe přizpůsobit se měnícím požadavkům a technologiím, a také usnadňuje testování a nasazení.

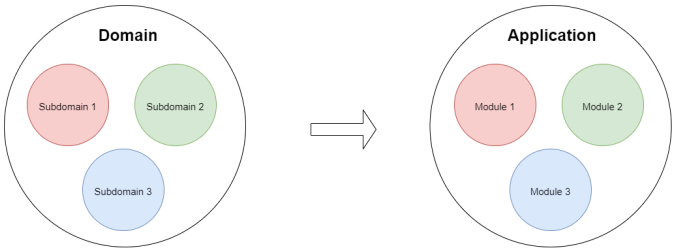
4.1.1 Klíčové vlastnosti

I když neexistuje univerzální definice, která by přesně určovala, jak má správná softwarová architektura vypadat, konkrétní podoba architektury je velmi často přizpůsobená technologii, zvyklostem týmu nebo požadavkům projektu. V této práci je při návrhu architektury kladen důraz na následující vlastnosti:

- `Rozšiřitelnost`: Schopnost softwarové architektury zvládat přidávání nových funkcí a komponent se nazývá rozšiřitelnost. Tato vlastnost umožňuje systému snadno se rozvíjet a růst společně s přibývajícími potřebami uživatelů a změnami v technologickém prostředí.
- `Udržitelnost`: Udržitelnost znamená, že softwarová architektura je navržena tak, aby byla snadno upravitelná, aktualizovatelná a rozšiřitelná.
- `Jednoduchost`: Softwarová architektura by měla být co nejjednodušší a intuitivní, aby ji bylo možné snadno pochopit a používat. Ideálně by měla být tak zřejmá, že ji dokáže každý odvodit pouhým čtením kódu. Jednoduchost v architektuře zvyšuje produktivitu a minimalizuje možnost chyb způsobených nepochopením kódu.
- `Nezávislost`: Struktura projektu by měla být dekomponovaná tak, aby obsahovala co nejméně závislostí mezi jednotlivými částmi. Nezávislost také zvyšuje modularitu a snižuje riziko, že změna v jedné části softwaru negativně ovlivní jinou část.
- `Flexibilita`: Architektura by měla být navržená tak, aby bylo možné jednoduše provádět změny v průběhu životního cyklu softwaru. Flexibilita umožňuje rychle reagovat na nové požadavky nebo změny v technologiích. [27]

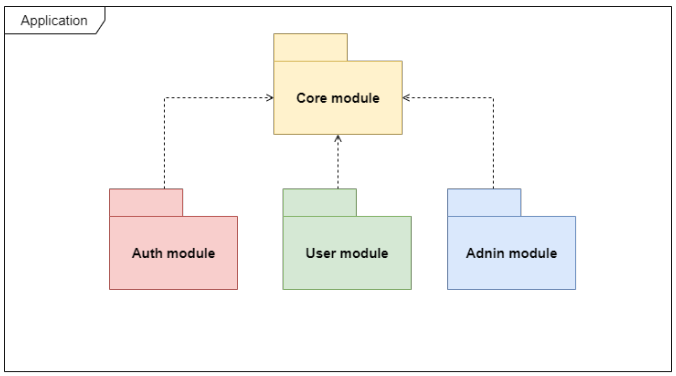
4.1.2 Dekompozice podle domény

Pro dosažení nezávislosti a zvýšení udržitelnosti aplikace je vhodné rozdělit ji do nezávislých funkčních modulů. Jeden z možných způsobů, jak dekomponovat vyvíjený systém, je postupovat podle obchodní domény. [28]



Obrázek 4.1 Modularizace aplikace na základě analýzy domény

Během sběru a analýzy požadavků, které byly provedeny v sekci 2.3, byly identifikovány dva nadřazené typy aktérů — uchazeč a administrátor. Tyto dva typy aktérů poskytují základ pro rozdělení aplikačních modulů a na základě této myšlenky lze aplikaci rozdělit na moduly pro autentizaci, uživatelskou část a administrátorskou část. Rozdělení aplikace na tyto moduly umožňuje lepší oddělení funkcí a zjednodušuje jejich správu a rozšíření. Vzhledem k tomu, že moduly mají být od sebe izolované, je nutné vytvořit hlavní modul, který slouží jako spojovací článek mezi jednotlivými funkčními moduly. Tento hlavní modul lze nazvat core modul, který koordinuje ostatní moduly a zajišťuje jejich správnou komunikaci a integraci.



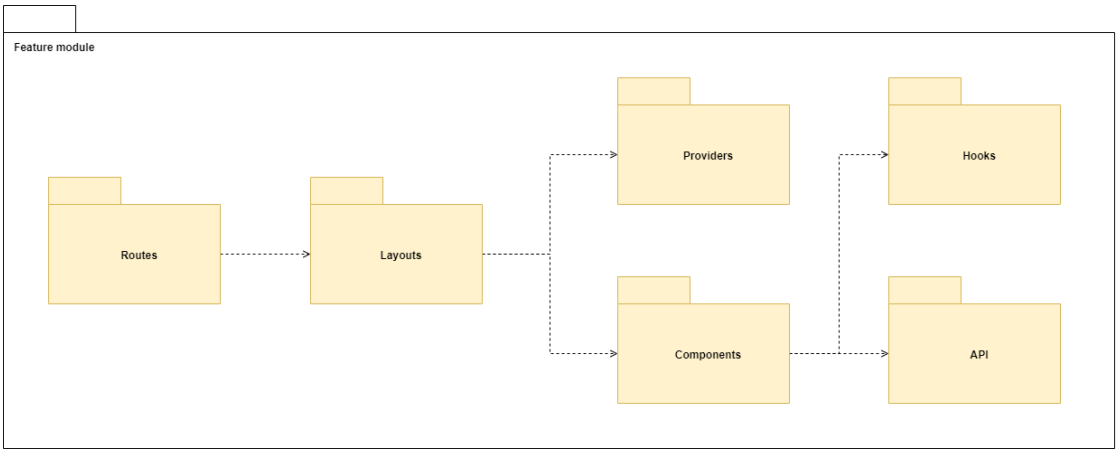
Obrázek 4.2 Výsledná modularizace aplikace

Na obrázku 4.2 lze vidět diagram balíčků výsledné dekompozice aplikace. Šipky, které lze pozorovat v diagramu, naznačují závislosti balíčků, které směřují pouze z funkčních modulů do core modulu. Funkční moduly jsou tak na sobě nezávislé a obsahují aplikační logiku pro danou obchodní subdoménu. Core modul naopak zapouzdřuje veškerou sdílenou obchodní logiku pro ostatní funkční moduly, a proto by se v tomto modulu měly nacházet veškeré funkcionality, které se budou nejméně měnit. Díky tomu je zajištěna centrální správa a koordinace mezi jednotlivými moduly.

Dekompozice podle domény přináší několik výhod. Jednou z nich je snazší řízení a udržování jednotlivých částí systému, jelikož každý modul je zaměřen na specifickou oblast a je oddělen od ostatních. Kromě toho je taková architektura škálovatelnější, neboť umožňuje přidávání nových modulů bez nutnosti zásadních změn v ostatních částech aplikace.

4.1.3 Struktura funkčních modulů

Dekompozice aplikace do samostatných funkčních modulů je důležitým krokem pro dosažení nezávislosti a zvýšení udržitelnosti aplikace. Avšak samotná dekompozice nestačí, a proto je nutné zajistit, že i uvnitř samotných modulů bude dodržována určitá struktura. Tato struktura umožňuje lepší organizaci kódu, zjednodušuje navigaci v projektu, a také usnadňuje spolupráci mezi vývojáři. Na obrázku 4.3 lze vidět diagram balíčků, který znázorňuje navrženou adresářovou strukturu uvnitř funkčních modulů.



Obrázek 4.3 Vnitřní struktura funkčního modulu

V diagramu lze pozorovat adresáře, které obsahují spolu související implementační soubory. Význam jednotlivých adresářů je následující:

- `Routes`: Tento adresář obsahuje definice cest a směrování uvnitř modulu. Zde jsou umístěny soubory, které určují, jaké komponenty se zobrazí při přístupu k určitým cestám či URL.
- `Layouts`: Adresář obsahující šablony pro strukturu stránek modulu. Tyto šablony definují základní rozvržení stránek, jako jsou hlavička, navigace nebo patička, a umožňují jednotný vzhled napříč celým modulem.
- `Providers`: Zde se nachází zdroje dat pro komponenty modulu. Providers zajišťují načítání i ukládání dat a jejich poskytování ostatním komponentám modulu, což umožňuje efektivnější správu a manipulaci s daty.
- `Components`: Adresář obsahující jednotlivé komponenty modulu. Komponenty jsou reprezentací stavebních bloků uživatelského rozhraní, které tvoří vizuální a funkční části aplikace.
- `Hooks`: Tento adresář obsahuje implementace vlastních React Hooks, které umožňují znovupoužitelnou logiku mezi komponentami. Hooks zjednodušují správu stavu a životního cyklu komponent v modulu.
- `API`: Adresář pro API volání a komunikaci s backendem. Zde jsou umístěny soubory a funkce, které umožňují získávání a odesílání dat ze serveru prostřednictvím API volání

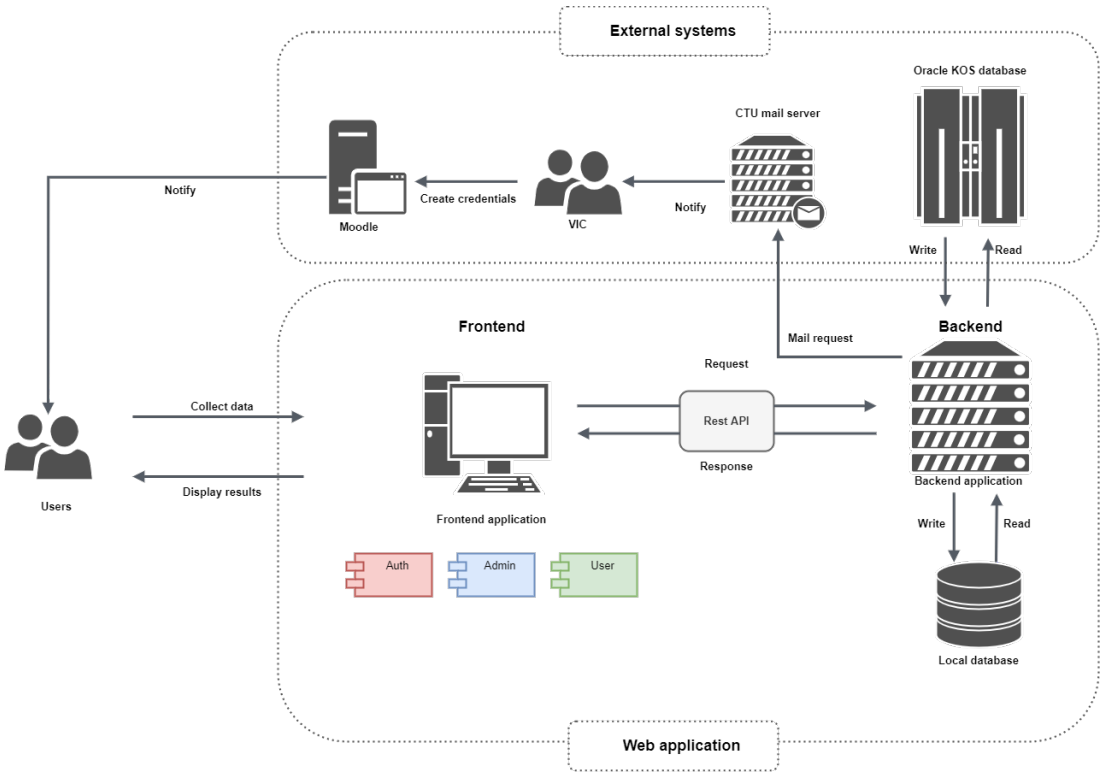
Kromě toho, že seskupení implementačních souborů zvyšuje přehlednost a zesnadňuje orientaci v projektu, jsou závislosti mezi adresáři navržené tak, aby při provedení změny v jakémkoliv adresáři, byl ovlivněn pouze jeho přímý soused, což zvyšuje predikovatelnost dopadu změn, a tím lze celkově snížit riziko regresí během vývoje.

4.2 Systémová infrastruktura

Poté, co byla definována struktura frontendové části, došlo k domluvení schůzky s autorem backendové části za účelem navržení infrastruktury celého systému. Tento návrh je založen na důkladné analýze domény, sběru požadavků a diskuzí s autorem backendové části. Zjednodušené schéma systémové infrastruktury je zobrazeno v obrázku 4.4.

Na tomto obrázku lze rozpoznat frontendovou část složené ze tří modulů, které byly navrženy v předchozí části této práce. Tyto moduly komunikují s backendovou částí prostřednictvím REST API, což zajišťuje propojení obou částí systému. V infrastruktuře lze také vidět dva datové zdroje — lokální databázi a Oracle databázi systému KOS. V těchto databázích jsou uložena data o přihláškách, studijních programech a důležitých termínech, která se pravidelně synchronizují mezi lokální a externí databází. Volba backendové architektury a databázové technologie je ponechána na uvážení autora backendové části, což je v schématu zobrazeno pouze zjednodušeně.

Infrastruktura systému rovněž zahrnuje univerzitní mailový server, který je určen pro případné zasílání emailových zpráv v rámci systému. Specifickým příkladem použití mailového serveru je situace, kdy je nutné zaslat emailovou zprávu na VIC se seznamem uchazečů, pro které je třeba vytvořit přístupové údaje do systému Moodle. Jakmile jsou údaje v Moodle vytvořeny, uchazeči automaticky obdrží emailovou zprávu, která je informuje o tom, že byl pro ně vytvořen účet a mohou se následně přihlásit do systému.



Obrázek 4.4 Zjednodušené schéma infrastruktury navržené aplikace

Během schůzky s autorem backendové části byly také specifikovány konkrétní požadavky na definici REST API. Vzhledem k tomu, že těchto požadavků je značné množství a mohou být obtížné udržet v přehledu, bylo navrženo vytvoření prototypu uživatelského rozhraní. Tento prototyp bude sloužit jako nástroj, který usnadní udržování přehledu o požadavcích na rozhraní a zároveň bude pomáhat při koordinaci vývoje frontendu a backendu systému. Vytvoření prototypu uživatelského rozhraní bude hlavním tématem následující podkapitoly této práce.

4.3 Uživatelské rozhraní

Dobře navržené rozhraní může udělat produkt intuitivnější, uživatelsky přívětivější a efektivnější, což vede ke zlepšení uživatelské zkušenosti a zvýšení spokojenosti. Na druhou stranu špatně navržené rozhraní může být matoucí, frustrující a vést k uživatelským chybám, což může poškodit přijetí produktu, udržení uživatelů a celkový úspěch. Tato podkapitola se bude věnovat detailnímu návrhu uživatelského rozhraní pro vyvíjenou aplikaci.

4.3.1 Použitelnost

Použitelnost, někdy také označována jako uživatelská přívětivost, je míra, do jaké je produkt nebo systém schopen splnit požadavky a potřeby svých uživatelů. Použitelnost je významným faktorem pro úspěch jakéhokoli produktu nebo systému, protože ovlivňuje, jak snadno a efektivně mohou uživatelé dosáhnout svých cílů při použití daného produktu nebo systému. Jakob Nielsen, přední odborník na použitelnost, definoval použitelnost prostřednictvím pěti hlavních složek kvality:

- `Pochopitelnost`: Týká se toho, jak snadno a rychle se uživatelé mohou naučit, jak navigovat a interagovat se systémem nebo produktem. Použitelný systém by měl uživatelům umožnit dosáhnout svých cílů s minimálním úsilím a školením.
- `Efektivita`: Úroveň produktivity, kterou uživatelé mohou dosáhnout poté, co se naučili používat systém. Použitelný systém by měl uživatelům umožnit rychle a s minimálním úsilím dokončit úkoly.
- `Zapamatovatelnost:` Zapamatovatelnost se vztahuje na schopnost uživatelů si pamatovat, jak používat systém nebo produkt po určité době, kdy jej nepoužívali. Použitelný systém by měl být snadno zapamatovatelný, aby si uživatelé mohli znovu osvojit rozhraní bez nutnosti jej znovu učit.
- `Chyby:` Tato složka se týká počtu chyb, kterých se uživatelé dopouštějí při používání systému, závažnosti těchto chyb a jak snadno se z nich mohou zotavit. Použitelný systém by měl minimalizovat výskyt chyb a poskytnout jasnou orientaci, jak uživatelům pomoci, když se chyby stávají.
- `Spokojenost`: Týká se subjektivních pocitů, které uživatelé mají při interakci se systémem nebo produktem. Použitelný systém by měl být příjemný a příjemný k použití, čímž zanechává uživatelům pozitivní zkušenost.

Při návrhu uživatelského rozhraní pro aplikaci je důležité zohlednit všechny složky použitelnosti a zajistit, aby byly splněny. Důvod, proč na použitelnosti záleží, je spojen s tím, že uživatelé mají omezené zdroje, jako je čas, pozornost a trpělivost, a systémy s vysokou použitelností tyto zdroje mohou značně šetřit. [29]

4.3.2 Prototypování

Prototypování je proces vytváření zjednodušených a funkčních verzí produktu nebo systému, které slouží k validaci konceptů, návrhu a funkcí před vývojem finální verze. Prototypy mohou mít různé úrovně detailu a komplexity, od papírových náčrtků až po plně interaktivní digitální modely. Prototypování je důležitým krokem v procesu návrhu uživatelského rozhraní, protože umožňuje rychle a efektivně získat zpětnou vazbu od uživatelů a iterativně zdokonalovat návrh na základě této zpětné vazby. [30]

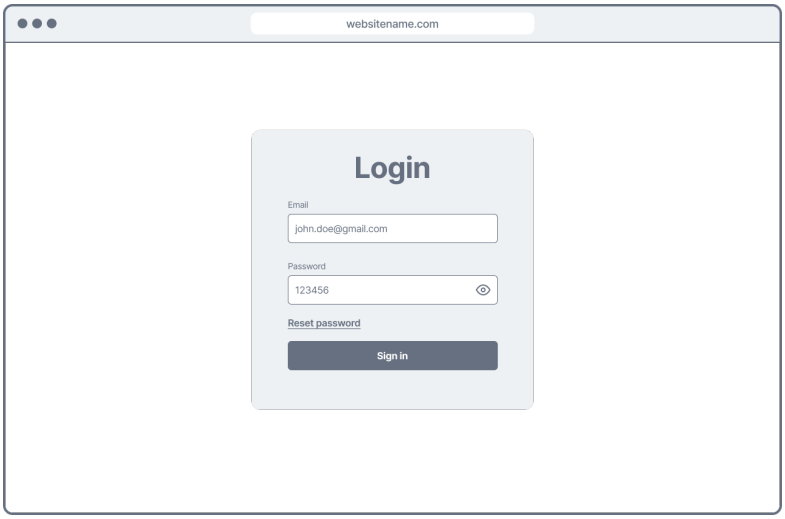
4.3.3 Lo-fi prototyp

Lo-fi prototypování, neboli prototypování s nízkou úrovní detailu, je metoda vytváření zjednodušených verzí návrhu uživatelského rozhraní s omezenou funkcionalitou a grafickým zpracováním. Myšlenkou lo-fi prototypování je rychle a levně vyvinout základní verzi výsledného produktu, který umožňuje testovat klíčové koncepty a funkce s cílovými uživateli. Lo-fi prototypování nabízí možnost soustředit se na klíčové aspekty návrhu, jako jsou funkčnost a interakce, místo na vizuální detaily a estetiku. [30] Prototypování typu lo-fi lze realizovat v různých podobách, avšak v této práci je zvolena metoda vizualizace pomocí wireframů. Wireframes jsou základní vizuální reprezentace struktury a rozložení uživatelského rozhraní, které slouží jako nástroj pro rychlé a jednoduché prototypování. Tento druh prototypování se zaměřuje na uspořádání prvků na obrazovce, hierarchii informací a základní funkce. Wireframes mohou být vytvořeny tradičním způsobem, jako je ruční kreslení na papíře, nebo digitálně pomocí specializovaných nástrojů a software. Kompletní návrh wireframů lze najít v příloze C, z důvodů příliš velkého rozsahu jsou v této části rozebrány jen nejdůležitější funkcionality. Návrh byl vytvořen za pomocí aplikace přímo specializovanou pro návrh uživatelských rozhraní Figma a jsou v něm zohledněny výsledky obchodní analýzy a rešerše systému Přiříz.

4.3.3.1 Autentizace

Autentizace v aplikaci je navržena tak, aby zajistila jednoduché a bezpečné přihlašování pro administrátory i uchazeče. Po spuštění webové aplikace je uživateli zobrazen přihlašovací formulář, kam zadá své přístupové údaje, které zahrnují uživatelské jméno a heslo. Potvrzení přihlášení probíhá stisknutím tlačítka pod formulářem. Přihlašovací stránka je společná jak pro uchazeče, tak pro administrátory.

Pro zjednodušení přihlašování a snížení pravděpodobnosti zadání nesprávných přístupových údajů lze heslo odkrýt stisknutím tlačítka s ikonou oka. Pokud jsou zadané přístupové údaje platné, aplikace ověří roli uživatele a přesměruje jej buď do uživatelské části nebo do administrátorské části, v závislosti na jeho roli. V případě, že uživatel zadá nesprávné přístupové údaje nebo nemá adekvátní přístupová práva, aplikace přístup odmítne a zobrazí konkrétní chybovou hlášku pod formulářem. Pro situace, kdy uživatel zapomene své přístupové údaje, je ve formuláři umístěno tlačítko pro resetování hesla. Po kliknutí na toto tlačítko je uživatel přesměrován do formuláře pro obnovu hesla. Návrh celé přihlašovací obrazovky je zobrazen v obrázku 4.5.

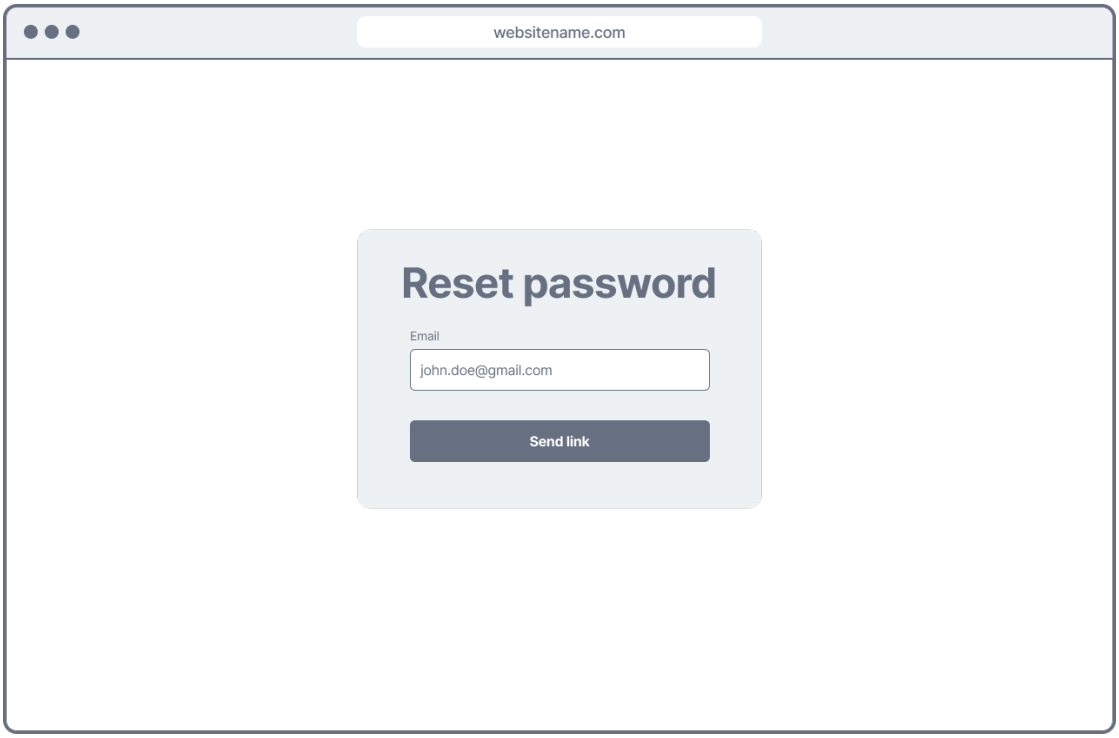


Obrázek 4.5 Wireframe přihlašovací obrazovky

V případě přesměrování uživatele na obrazovku pro reset hesla je aplikací vykreslen formulář, který přijímá pouze e-mailovou adresu daného účtu. Po vyplnění a potvrzení formuláře uživatelem je zobrazena zpráva o úspěšném odeslání instrukcí pro obnovu hesla na zadaný e-mail, pokud existuje. Pokud e-mailová adresa neexistuje, je uživateli zobrazena chybová hláška o neexistenci účtu. Při úspěšném odeslání instrukcí je tlačítko pro potvrzení ve formuláři uzamčeno na 5 minut a uživatel je vyzván ke kontrole své e-mailové schránky, zda mu zpráva dorazila. Tato opatření zajišťují dostatek času pro nalezení e-mailu s instrukcemi a současně brání přehlcení e-mailového serveru, který zprávy rozesílá. Pokud uživateli zpráva v daném časovém intervalu nedorazí, aplikace mu umožní poslat novou zprávu s instrukcemi. Klíčovým prvkem zaslané zprávy je odkaz s unikátním tokenem, který po rozkliknutí uživatele přesměruje na obrazovku pro nastavení nového hesla bez nutnosti znalosti původního hesla.

Vzhledem k tomu, že systém zobrazuje citlivá data, jsou na hesla uživatelů kladeny zvýšené nároky a musí splňovat následující vlastnosti:

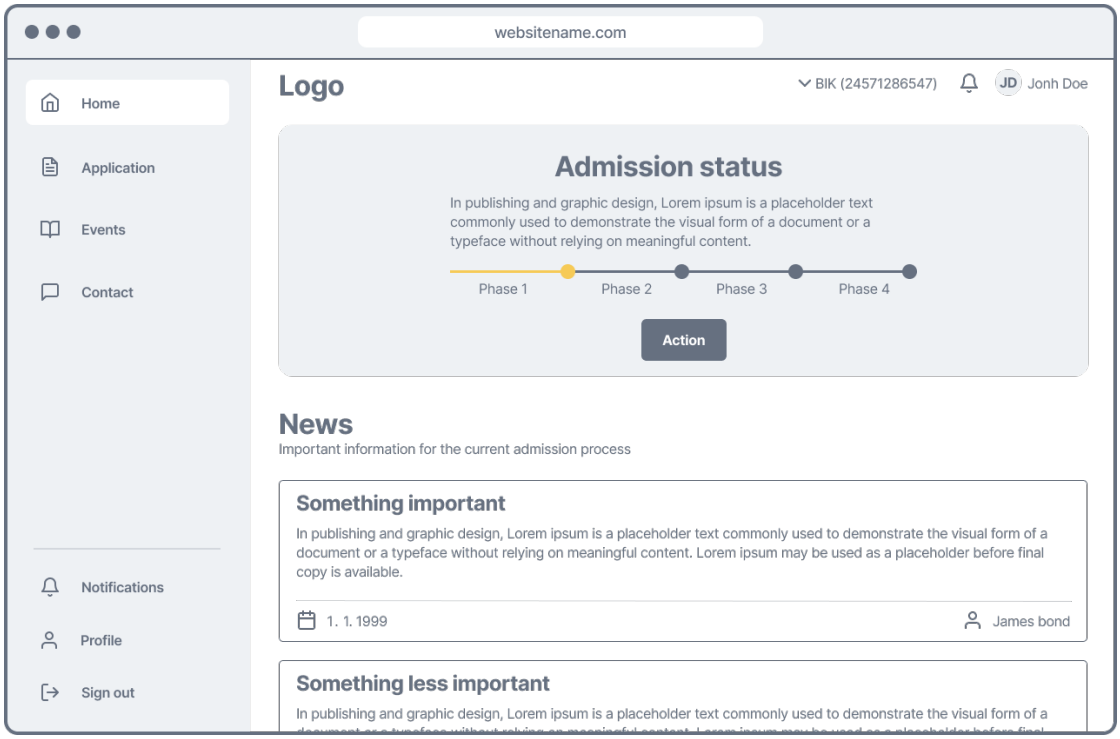
- `Délka:` Hesla by měla mít minimálně 8 znaků, ideálně více. Delší heslo je obtížnější prolomit.
- `Složitost:` Hesla by měla obsahovat kombinaci alespoň jednoho velkého a malého písmene, jednoho čísla a jednoho speciálního znaku.
- `Nepředvídatelnost:` Hesla by neměla být založena na snadno odhadnutelných informacích, jako jsou jméno nebo datum narození uživatele.



Obrázek 4.6 Wireframe obrazovky pro obnovu hesla

4.3.3.2 Uživatelská část

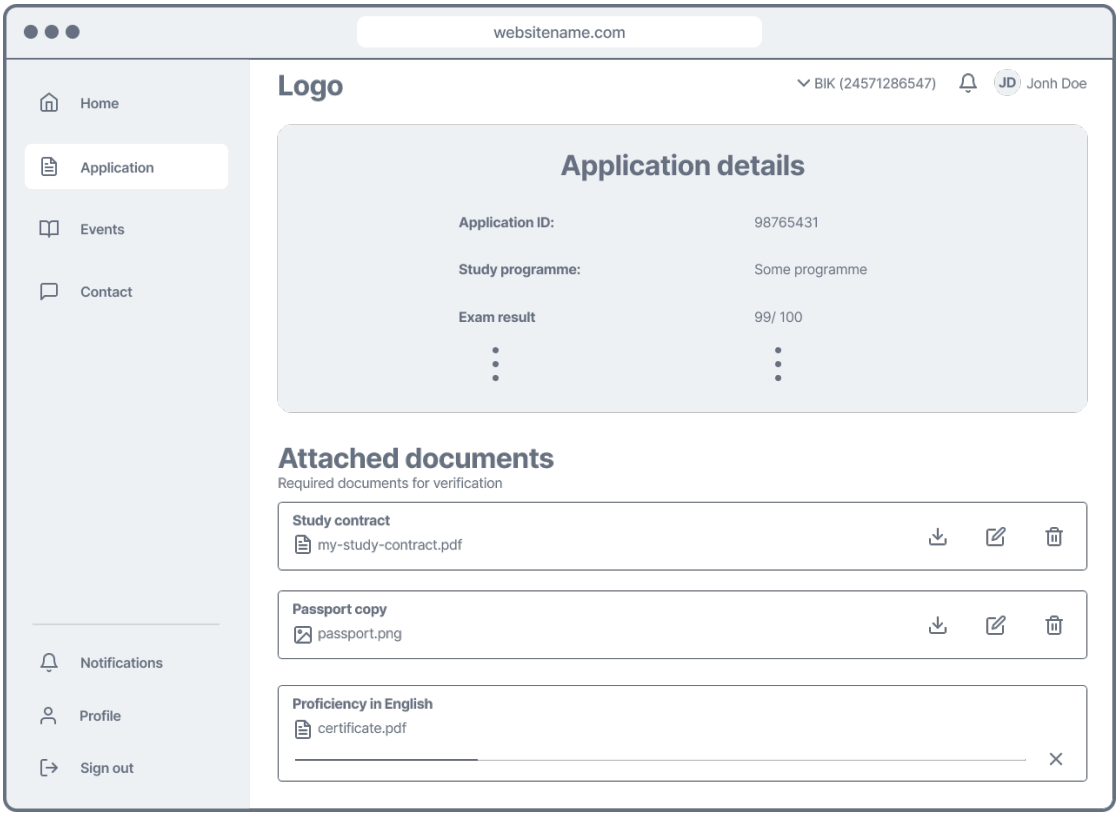
Pokud je uživatel na základě své role přesměrován do uživatelské části aplikace, zobrazí se mu domovská stránka, ilustrovaná v obrázku 4.7. Celkové rozložení stránek je inspirováno systémem Přiříz, což znamená, že v pravé části obrazovky se nachází dynamický obsah stránky, zatímco v levé části obrazovky je umístěno postranní menu s tlačítky pro přesměrování do různých sekcí aplikace. V horní části obrazovky je navíc umístěna lišta, která zobrazuje jméno aktuálně přihlášeného uživatele, ikonu notifikace s počtem nových upozornění a speciální filtr, umožňující uchazečům přepínat mezi jednotlivými přihláškami. Tento filtr globálně omezuje obsah na základě zvolené přihlášky, což výrazně zjednodušuje správu více přihlášek současně ve srovnání se systémem Přiříz.



Obrázek 4.7 Wireframe domovské obrazovky v uživatelské sekci

Domovská stránka uchazečů obsahuje informativní progress bar a seznam aktualit. Progress bar uživatelům přehledně zobrazuje veškeré stavy a data z aktuálně zvolené přihlášky ve vizuálně atraktivní formě přímo na domovské obrazovce. Progress bar zobrazuje všechny možné stavy v kontextu přijímacího řízení, vizualizuje již splněné akce a zároveň i následující akci, kterou uchazeč musí splnit. K tomu progress bar poskytuje tlačítko, které po rozkliknutí uchazeče aplikace sama navede ke části systému, ve které je nutné provést požadovanou akci. Všechny stavy a jejich náležitosti pro splnění jsou v progress baru podrobně popsány, což uživatelům usnadňuje orientaci. Pod progress barem jsou zobrazeny všechny aktuality spojené se studijním programem zvolené přihlášky. Aktuality představují užitečný zdroj informací pro uchazeče, který jim pomáhá zůstat v obraze a udržet si přehled o důležitých událostech a změnách v rámci přijímacího řízení. Každá novinka se skládá z názvu, detailního popisu, data publikace a jména autora. Díky těmto informacím mohou uchazeči rychle identifikovat, zda je daná novinka relevantní pro jejich situaci, a případně se seznámit s podrobnostmi.

Pokud uživatel chce získat podrobné informace o své aktuálně zvolené přihlášce, může v postranním menu vybrat tlačítko, které ho přesměruje na stránku s detailem přihlášky, jak je znázorněno na obrázku 4.8. Na této stránce aplikace zobrazuje všechny informace spojené s přihláškou, jako je stav rozhodnutí, identifikátor přihlášky a typ studijního programu. Díky tomu mají uchazeči snadný přístup k nejdůležitějším informacím o své přihlášce a mohou tak lépe sledovat její postup v přijímacím řízení. V detailu přihlášky uživatel také nalezne svoje přístupové údaje do systému Moodle a záznamy o všech událostech, kterých se zúčastnil, a jejich případné ohodnocení. Navíc lze v detailu přihlášky najít veškeré původně přiložené dokumenty, které uchazeč může upravovat nebo přidávat nové. Tato funkce umožňuje uchazeči snadno aktualizovat svou přihlášku a zajistit, že studijní oddělení má vždy k dispozici nejnovější a nejrelevantnější informace.

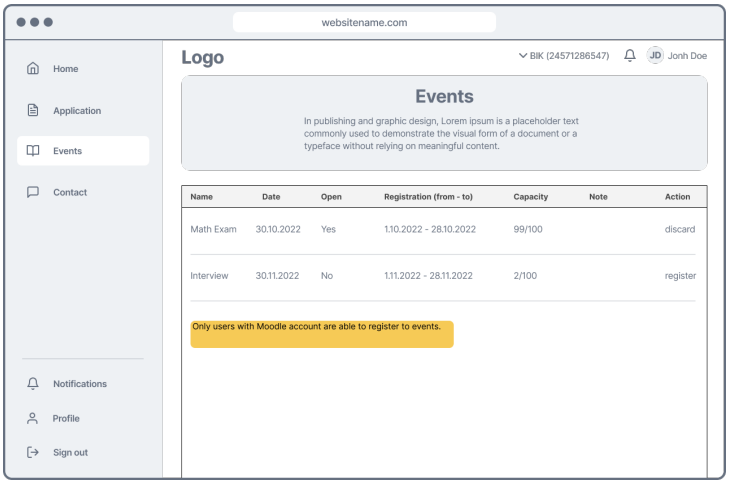


Obrázek 4.8 Wireframe detailu přihlášky v uživatelské sekci

Jestliže má uživatel všechny informace v pořádku, může přejít do sekce událostí, kterou lze vidět na obrázku 4.9. V této části aplikace se uživatelům zobrazuje přehledová tabulka se všemi událostmi spojenými s jejich studijním programem. Tento nástroj pomáhá uchazečům lépe se zorientovat a zúčastnit se důležitých akcí souvisejících s přijímacím řízením. Každá událost v tabulce obsahuje informace jako název, datum konání, informace o tom, zda je zápis otevřený, uzávěrky zápisu, kapacitu a případné poznámky. V posledním sloupci tabulky jsou zobrazena tlačítka pro přihlášení a odhlášení. Pro účast na událostech platí některá pravidla, která musí být splněna. Tato pravidla vycházejí zejména z analýzy domény a aplikace je automaticky ověřuje. Zároveň všechny nutné podmínky lze ověřit v domovské obrazovce pomocí progress baru.

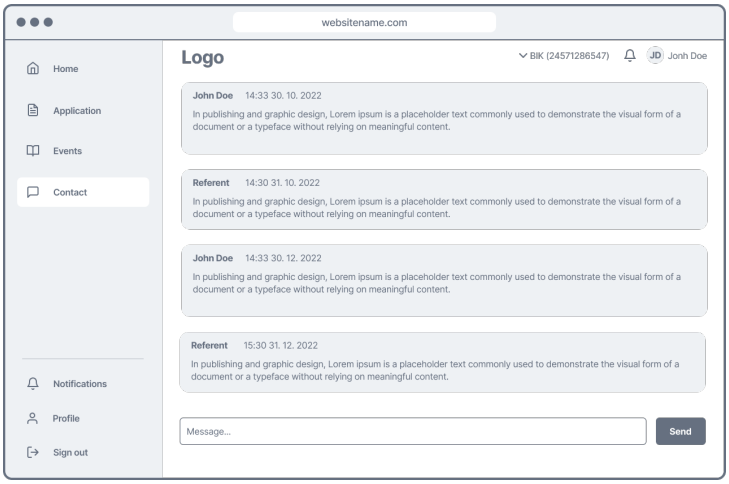
Podmínky, které musí být nutně splněna pro přihlášení na události, jsou následující:

1. Pro přihlášení je nutné, aby byl termín otevřený pro zápis a měl dostatečnou kapacitu.
2. Žádnou událost uživatel nesmí absolvovat vícekrát či být zapsaný na více událostech souběžně.
3. Přihlašuje-li se uživatel na test z matematiky, je nutné aby měl vytvořené přístupy do Moodle.
4. Přihlašuje se uživatel na pohovor, musí mít úspěšně absolvovaný test z matematiky.



Obrázek 4.9 Wireframe událostí v uživatelské sekci

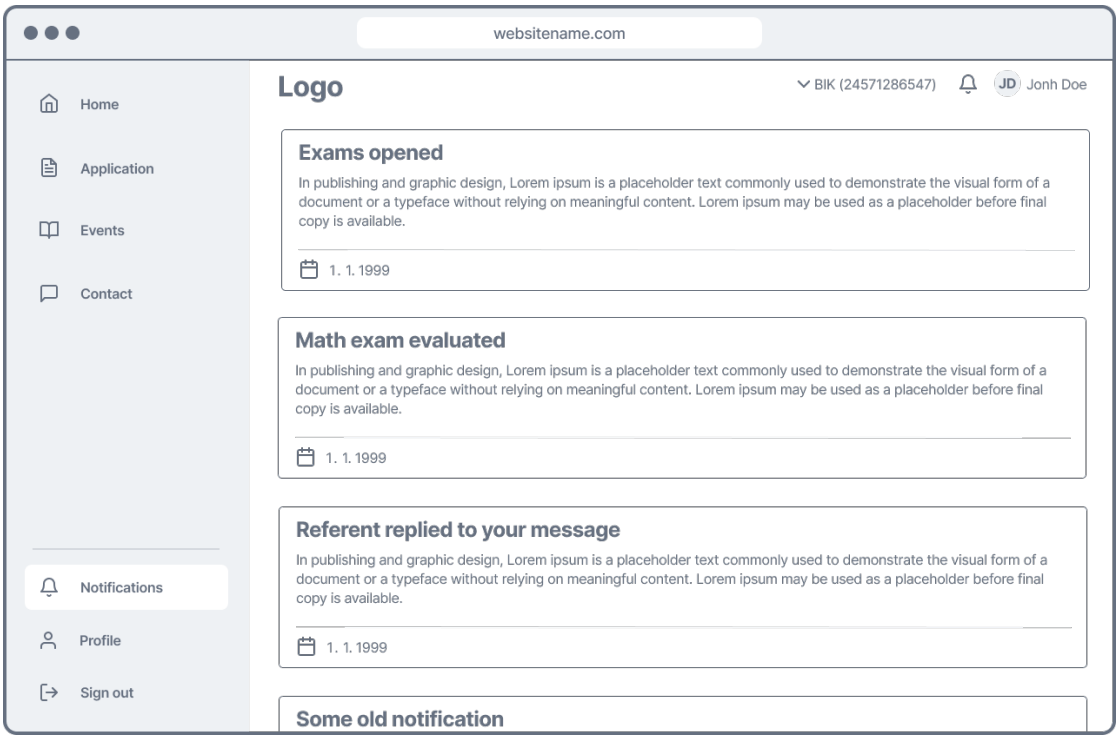
Nastane-li během přijímacího řízení jakýkoliv problém, uživatel má možnost vytvořit konverzaci s administrátory, což je znázorněno v obrázku 4.10. Detail konverzace umožňuje uživatelům odesílat zprávy administrátorům, přičemž zprávu si může zobrazit a na ni odpovědět jakýkoliv administrátor. Díky tomu je administrativní zátěž snížena, protože korespondence s uchazeči se může rozložit mezi více administrátorů. V detailu konverzace lze vidět celou historii vyměněných zpráv, což uživatelům poskytuje přehled a usnadňuje sledování komunikace s administrátory.



Obrázek 4.10 Wireframe konverzací v uživatelské sekci

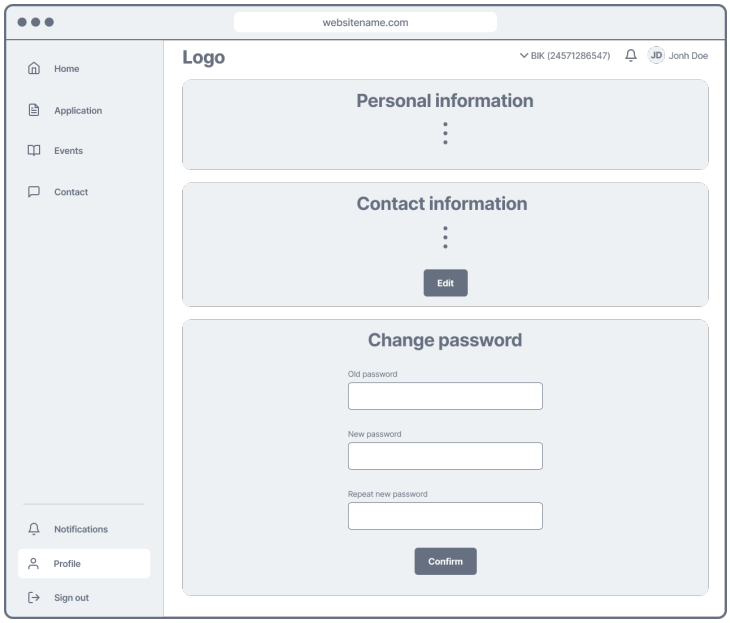
Aplikace také informuje uživatele o důležitých nebo nadcházejících událostech prostřednictvím systému notifikací, který je zobrazen v horní liště na každé stránce. Ikona zvonku v horní liště slouží k zobrazení rychlého přehledu nových notifikací po jejím rozkliknutí. Z tohoto přehledu se uživatelé mohou dostat do kompletní historie notifikací, která je znázorněna v obrázku 4.11. V historii notifikací uživatelé uvidí seznam nových i již zobrazených notifikací. Notifikace, které nebyly ještě zobrazeny, jsou prioritně řazeny na začátek a graficky zvýrazněny. Po kliknutí na notifikaci se označí jako zobrazená a již se nebude uživatelům zobrazovat v horní liště. Notifikace se generují v následujících situacích:

1. Při úspěšném přihlášení nebo odhlášení z události.
2. Před blížící se událostí, na kterou je uživatel přihlášen.
3. Při vytvoření přístupových údajů do systému Moodle.
4. Při obdržení nové zprávy v konverzaci.
5. Při získání ohodnocení z události.
6. Při změně stavu přihlášky.



Obrázek 4.11 Wireframe notifikací v uživatelské sekci

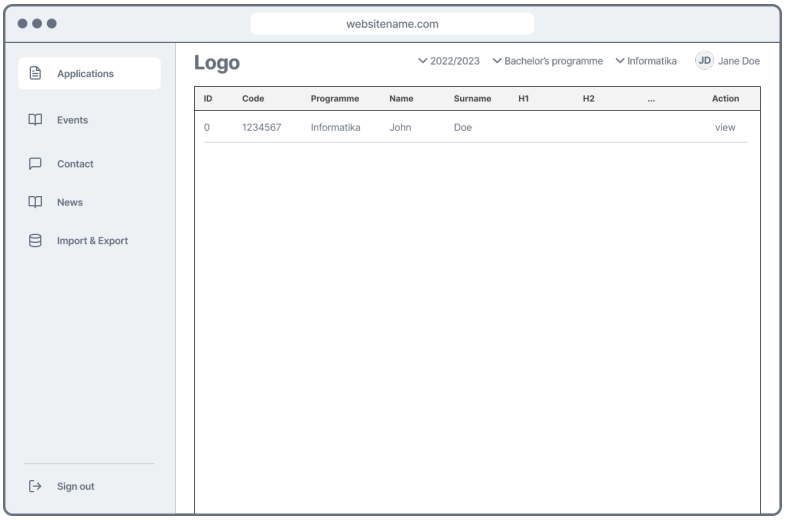
Poslední důležitou komponentou uživatelské části je profil uživatele poskytující uživatelům přehled o informacích souvisejících s jejich účtem, které jsou sdíleny mezi všemi přihláškami. V uživatelském profilu mají uživatelé možnost zobrazit a aktualizovat své kontaktní údaje, jako jsou například telefonní číslo a trvalá či kontaktní adresa. Kromě toho mohou uživatelé také změnit své heslo účtu. Znázornění uživatelského profilu lze vidět na obrázku 4.12.



Obrázek 4.12 Wireframe uživatelského profilu v uživatelské sekci

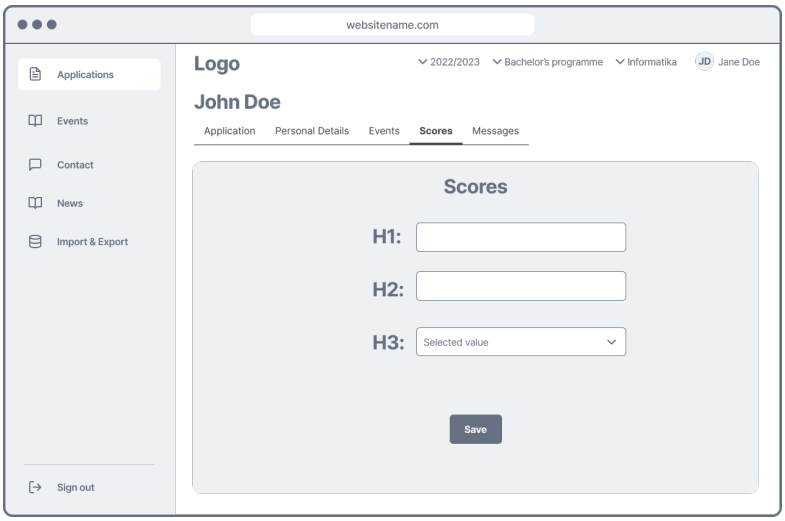
4.3.3.3 Administrátorská část

Administrátorská část aplikace nabízí stejné rozvržení stránek jako uživatelská — v levé části stránky se nachází navigace, v pravé části je pak dynamický obsah relevantní pro danou sekci. V horní části stránky je umístěna lišta pro globální filtrování dat, která v administrátorské sekci umožňuje filtrovat obsah na základě akademického roku a typu studijního programu. Při vstupu do administrátorské části jsou defaultně zobrazeny administrátorům přehled všech přihlášek, který lze vidět na obrázku 4.13.



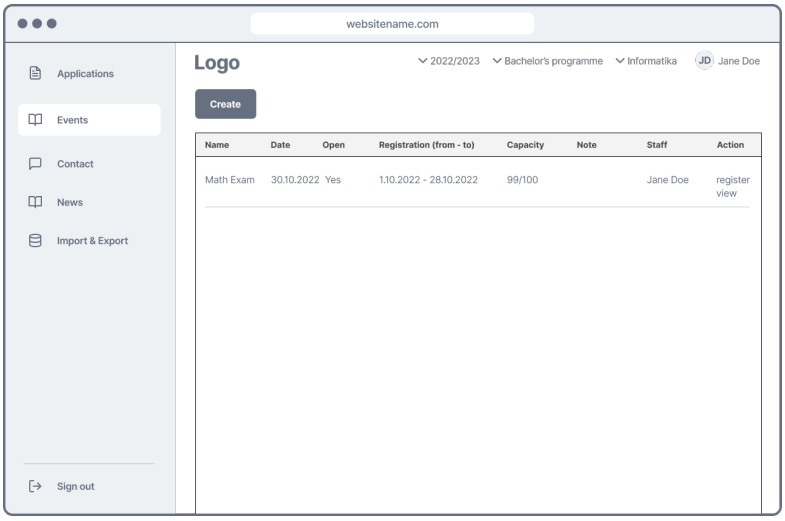
Obrázek 4.13 Wireframe přehledu přihlášek v administrátorské sekci

Přehled přihlášek obsahuje tabulku, v níž každý řádek reprezentuje jednu přihlášku v aktuálním přijímacím řízení. Každý řádek obsahuje interní identifikátor přihlášky, dále kód přihlášky, pod kterým je přihláška evidovaná v přijímacím řízení, studijní program, název uchazeče a sloupce H1 až H10 představující hodnoty pro bodování přihlášky. Na konci každého řádku je tlačítko pro vstup do detailu přihlášky zobrazeném v obrázku 4.14, ve kterém lze spravovat osobní údaje uchazeče, udělovat hodnocení či vytvořit s uchazečem konverzaci.



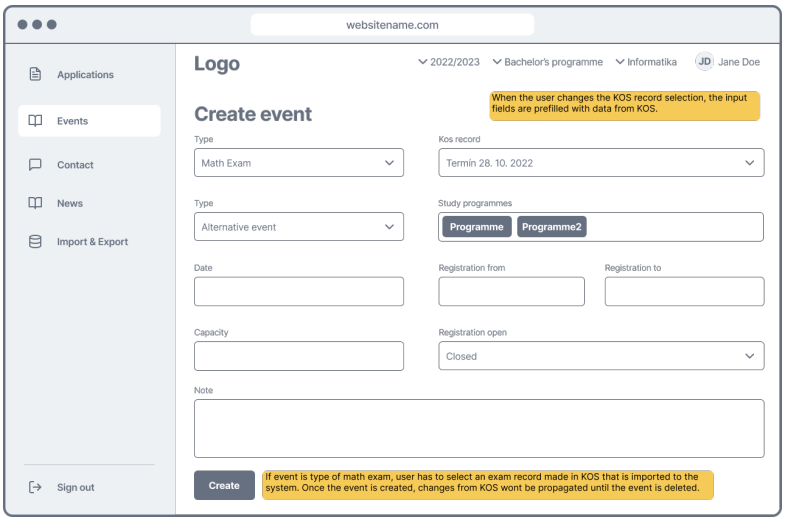
Obrázek 4.14 Wireframe detailu přihlášky sekce hodnocení v administrátorské sekci

Administrátorská část aplikace dále poskytuje možnost spravovat a přihlašovat se na události. V sekci pro přehled událostí mají administrátoři k dispozici tabulku všech událostí evidovaných v systému. Administrátoři se mohou přihlašovat na události jako supervizoři, čímž je zodpovědnost za organizaci událostí delegována na samotné komisaře a tazatele. Tato funkcionalita snižuje administrativní zátěž studijních referentů, protože se nemusí zabývat zjišťováním časových možností komisařů či tazatelů a následnou jejich alokací na vypsané termíny událostí.



Obrázek 4.15 Wireframe přehledu událostí v administrátorské sekci

V přehledu událostí mají administrátoři možnost přejít do formuláře pro vytvoření nové události, který jim do aplikace umožňuje snadno přidávat nové termíny přijímacího řízení. Důležitým aspektem správy událostí, který vyplývá z analýzy požadavků, je nutnost nejprve vytvořit záznam o události v systému KOS a následně jej importovat do lokální databáze navrhované aplikace. Tento postup zajišťuje, že všechny události jsou správně zaznamenány a synchronizovány mezi oběma systémy. Všechny importované události z KOSu se poté zobrazují ve formuláři v dropdown poli, jak lze vidět na obrázku 4.16. Po vybrání požadovaného KOS záznamu se v lokální databázi systému vytvoří nová událost a spáruje se s vybraným KOS záznamem. Tento krok je důležitý, protože zajišťuje, že vytváření nových událostí je kontrolované a změny v KOS záznamech nezpůsobí žádné vedlejší nekontrolované efekty. Jinými slovy, pokud dojde ke změnám v KOS záznamu, nepropíšou se automaticky do lokální události.

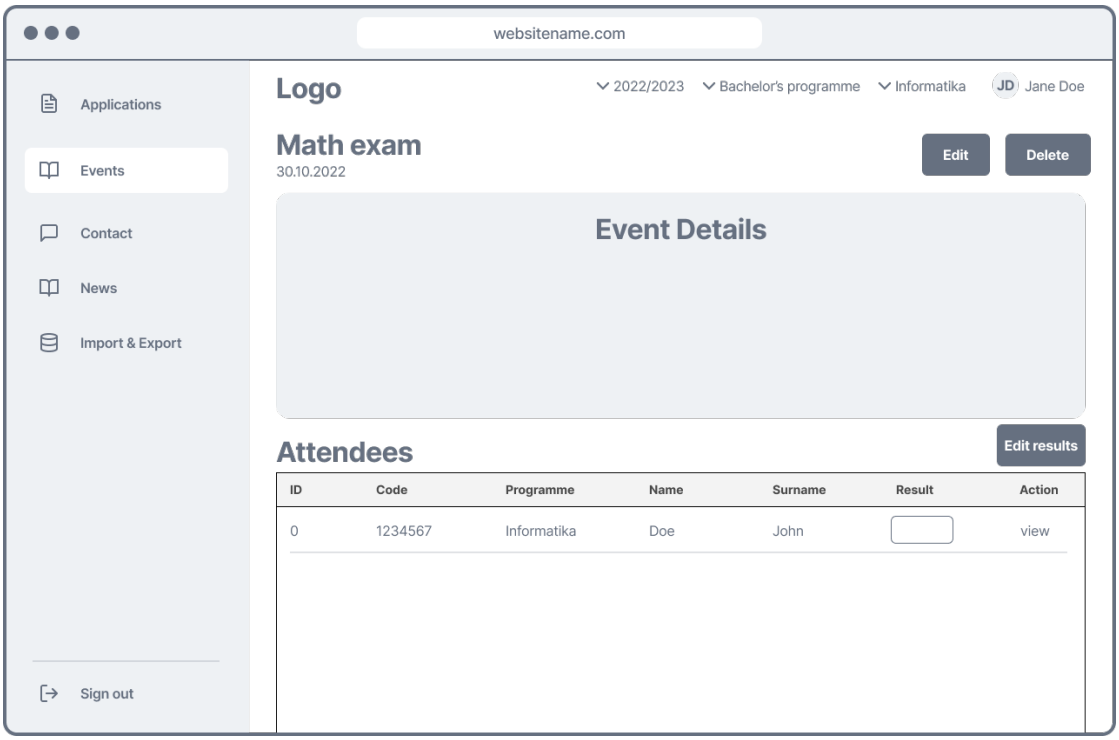


Obrázek 4.16 Wireframe vytvoření událost v administrátorské sekci

Pokaždé, když administrátoři ve formuláři zvolí adekvátní KOS záznam, všechny hodnoty formuláře se automaticky předvyplní daty z vybraného KOS záznamu. Tento proces usnadňuje a urychluje vyplnění formuláře, což přispívá k efektivnější správě událostí. Tato funkce také minimalizuje pravděpodobnost chyb při zadávání informací do formuláře, protože většina údajů je předvyplněna z ověřeného zdroje. Ve formuláři je důležité vyplnit následující pole:

- `KOS záznam:` Učuje mapování mezi lokální událostí na záznam v KOS. Po vybrání dojde k předvyplnění formuláře daty z KOS a záznam již nebude spárovatelný s dalšími událostmi.
- `Typ události:` Specifikuje typ události, který může být test z matematiky či pohovor.
- `Studijní program:` Výběr studijních programů, pro které je daný termín určen.
- `Čas konání:` Datum a čas, kdy se daná událost koná.
- `Otevření zápisu:` Datum, kdy bude možné se na událost přihlásit.
- `Uzavření zápisu:` Datum, po kterém se již nelze z události odhlásit.
- `Kapacitu:` Číslo udávající maximální počet uchazečů přihlášených na danou událost.
- `Status zápisu:` Příznak, zda-li je událost otevřená pro zápis. V opačném případě není možné se na událost přihlásit i přesto, že je již po datu otevření zápisu.
- `Poznámky:` Případné poznámky, které se zobrazí uchazečům v tabulce.

Po úspěšném vytvoření události se v přehledové tabulce objeví nový řádek reprezentující nově vytvořenou událost. Z tabulky událostí je možné se, kromě přihlášení či odhlášení jako supervizor, dostat do detailu události, který je znázorněn v obrázku 4.17. V detailu události lze vidět kromě relevantních informací, také tlačítka pro editaci a smazání události. Při zakliknutí tlačítka pro smazání se uživateli zobrazí na obrazovce dialogové okno, které po potvrzení vymaže událost z lokální databáze. Událost může být smazána pouze v případě, že termín ještě neproběhl. Po smazání události se smaže také vazba na záznam v KOS, což umožňuje následně na daný KOS záznam opět vytvářet nové události. Po kliknutí na tlačítko pro editaci se uživateli zobrazí formulář, který je téměř totožný jako formulář pro vytváření událostí. Hlavní rozdíl spočívá v omezení změn některých parametrů během modifikace. Při editaci události již nelze měnit typ události, vazbu na KOS záznam a určené studijní programy. Toto omezení zajišťuje konzistenci a stabilitu dat v průběhu celého procesu správy událostí.



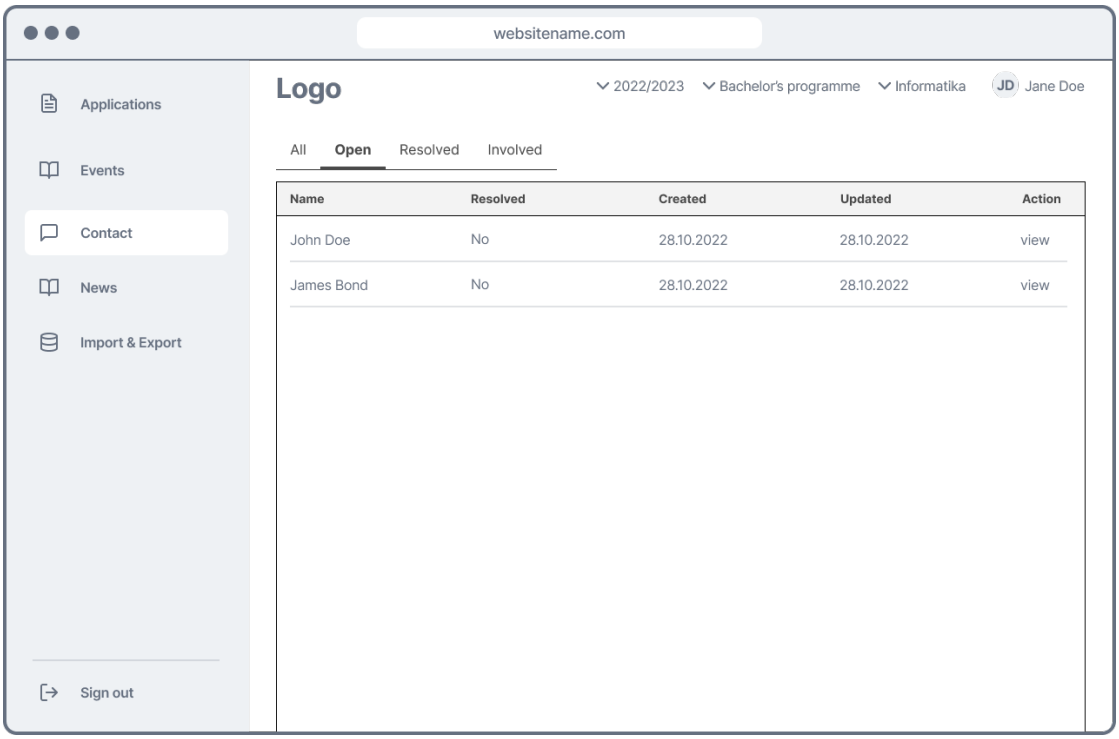
Obrázek 4.17 Wireframe detailu události v administrátorské sekci

Detail události obsahuje také seznam všech uchazečů, kteří se přihlásili na danou událost. Seznam uchazečů zobrazuje kód přihlášky, studijní program, název uchazeče a výsledek z události, který lze editovat. V závislosti na typu události se v seznamu objeví buď textové pole, které přijímá číselný vstup od 0 do 100 v případě testu z matematiky, nebo v případě pohovoru dropdown menu, kde lze zvolit, zda uchazeč prošel, a textové pole pro případné komentáře. Po vyplnění výsledků se nad tabulkou zobrazí tlačítko pro uložení, které výsledky hromadně uloží do databáze. Tato funkce umožňuje administrátorům hromadně udělovat hodnocení. Aplikace pak po uložení výsledků automaticky vyhodnotí stav přihlášek — pokud uchazeč splnil úspěšně jak test z matematiky, tak ústní pohovor, mění se stav přijetí na přijatý. V opačném případě, kdy daný uchazeč neprošel alespoň z jedné události, mění se automaticky stav jeho přihlášky na nepřijatý.

Jak již bylo zmíněno v části o popisu konverzací v uživatelské sekci, na zprávy zasílané uchazeči mohou odpovídat více administrátorů. V přehledu konverzací má každá konverzace status, který může být buďto otevřený či uzavřený, přičemž otevřené konverzace indikují nové konverzace nebo konverzace, které ještě nebyly vyřešeny. Poté, co administrátor vstoupí do detailu konverzace, může tento status změnit a označit konverzaci jako uzavřenou.

Konverzací může být poměrně velké množství, a proto jsou do přehledu konverzací zahrnuty speciální filtry, které na základě různých parametrů mohou vyfiltrovat konverzace. Konverzace je možné třídit na základě tří filtrů:

- `Otevřené:` Vyfiltruje všechny konverzace, které nemají status otevřený. Díky tomuto filtru mohou administrátoři si zobrazovat relevantní konverzace.
- `Uzavřené:` Vyfiltruje všechny konverzace, které nemají status uzavřený. Díky tomuto filtru si lze udržovat historii vyřešených konverzací.
- `Zúčastněné:` Vrátí konverzace, kterých se aktuálně přihlášený administrátor účastnil, tedy konverzace, kde poslal minimálně jednu zprávu.

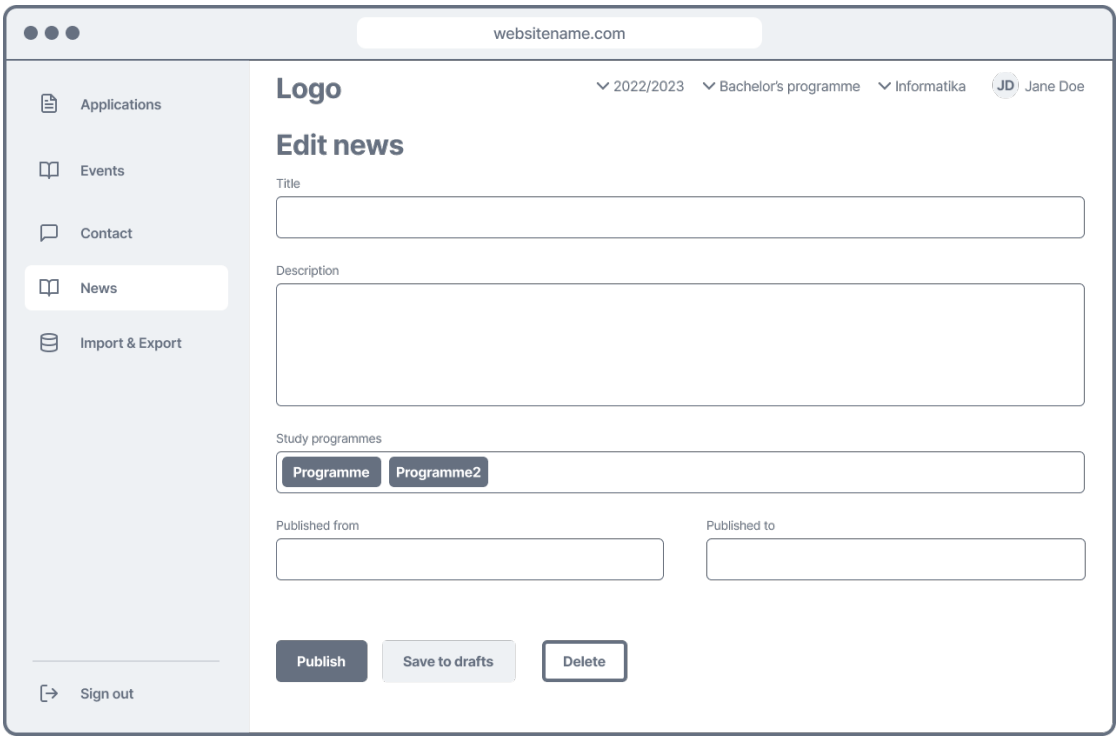


Obrázek 4.18 Wireframe přehledu konverzací v administrátorské sekci

Pro zúčastněné konverzace také platí, že kdykoliv od uchazeče je zaslána nová zpráva, zúčastnění administrátoři obdrží notifikaci o nové zprávě, což platí stejně naopak i pro uchazeče. Tímto způsobem je zajištěno, že i přes velké množství konverzací je za pomocí různých stavů a filtrů možné udržovat si přehled o nových a již vyřešených konverzacích.

V administrátorské části je pro správu aktualit k dispozici přehledová tabulka s obdobným rozložením jako u událostí, avšak u aktualit je možné provádět filtrování na základě publikovaných a rozpracovaných článků. Publikované aktuality jsou takové, které mají definované atributy pro čas začátku i konce publikace a aktuální datum se nachází mezi těmito daty.

Nad tabulkou je opět umístěno tlačítko pro vytvoření nové aktuality, které zobrazí administrátorovi formulář pro přidání nového článku. Po přístupu do detailu aktuality se administrátorovi zobrazí totožný formulář s tím, který je použit pro vytváření událostí. Podobu formuláře a jeho vstupní data lze nalézt na obrázku 4.19.

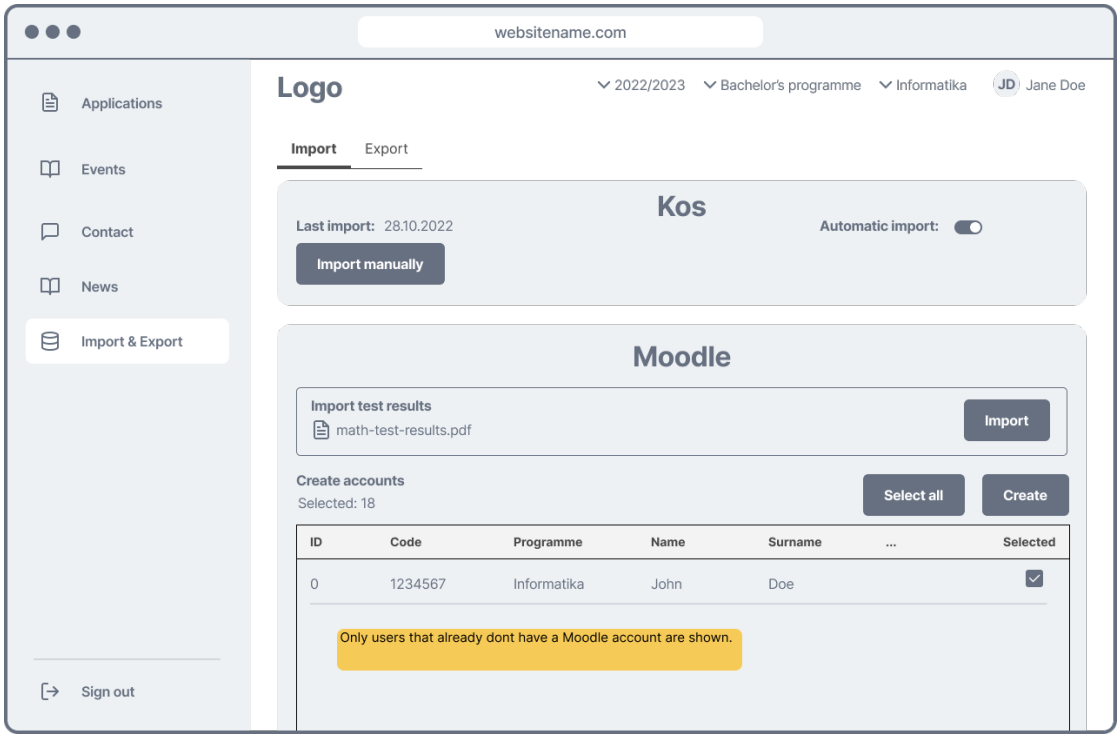


Obrázek 4.19 Wireframe detailu aktuality v administrátorské sekci

V dolní části formuláře se nachází, kromě tlačítek pro smazání a uložení, tlačítko pro uložení do rozpracovaných. Toto umožňuje uložit zprávu do přechodného stavu, kdy bude v systému neviditelná, dokud nebude řádně publikována. Tím je administrátorům poskytnut prostor pro promyšlení během psaní aktuality. Vzhledem k tomu, že aplikace je určená pro uchazeče anglických studijních programů, není zde nutné řešit lokalizaci jako u systému Přiřiz. Při vytváření či editaci aktuality je nutné vyplnit následující položky:

- `Název:` Vystihující název aktuality.
- `Popis:` Detailní popis aktuality.
- `Studijní programy:` Vybrané studijní programy, pro které se má aktualita zobrazit.
- `Začátek publikace:` Datum, od kdy bude aktualita viditelná.
- `Konec publikace:` Datum, po kterém aktualita není viditelná.

Poslední důležitou částí systému je jak funkcionalita pro import dat z externích systémů, tak export lokálních dat pro další zpracování.



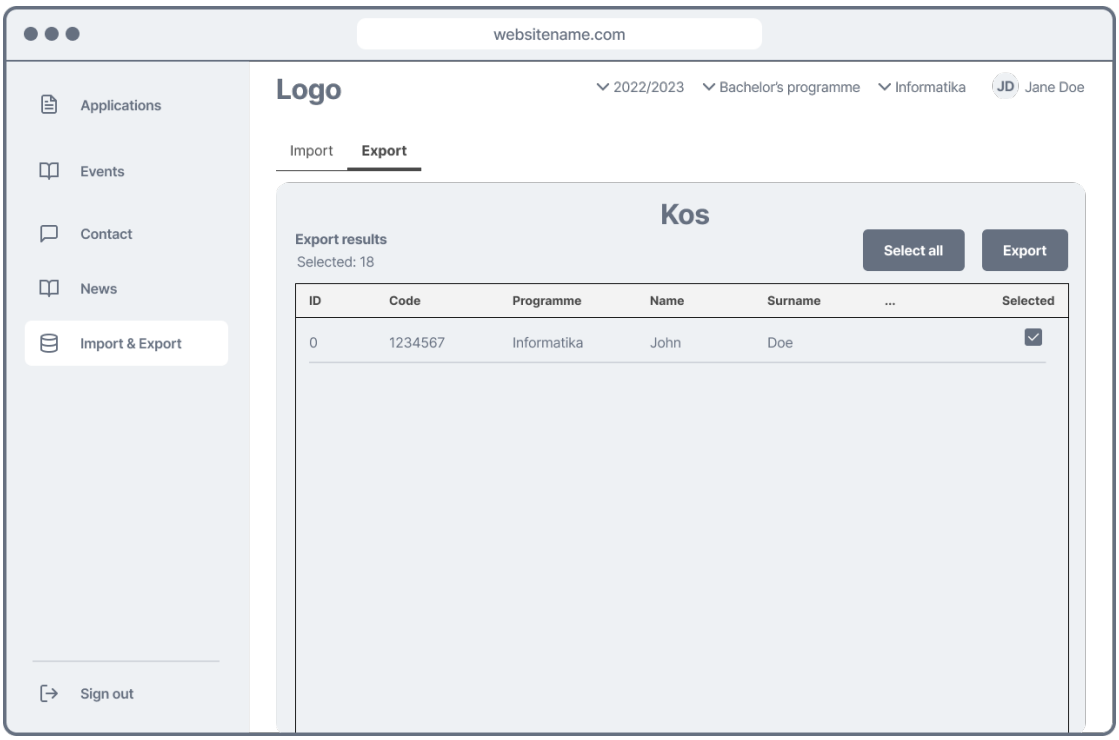
Obrázek 4.20 Wireframe import sekce v administrátorské sekci

Na základě analýzy požadavků bylo zjištěno, že citlivá data o přihláškách, studijních programech a uchazečích musí zůstat v databázi systému KOS z důvodu bezpečnosti a ochrany osobních údajů. Aby bylo možné tato data v aplikaci efektivně využít, je nutné zajistit jejich synchronizaci s lokální databází. Pro tento účel mohou administrátoři využít speciální funkcionalitu, která na stisknutí tlačítka stáhne z KOS databáze požadovaná data a následně je synchronizuje s lokální databází. Import dat z KOS bude probíhat automaticky v denním intervalu, což zajišťuje pravidelnou aktualizaci informací a udržuje je v souladu se zdrojovými daty. Administrátoři mají možnost automatický import vypnout či zapnout podle potřeby.

Zároveň v rámci importu je potřeba řešit integraci se systémem Moodle v kontextu vytváření přístupových údajů a stahování výsledků. Vzhledem k tomu, že pro Moodle není vystavené žádné veřejné API, je vytváření přístupových údajů vyřešené alternativním způsobem. Na základě výběru přihlášek z tabulky, se vygeneruje emailová zpráva se seznamem všech z tabulky vybraných přihlášek, kterým se má založit přístupové údaje. Pracovník z VIC poté na emailovou zprávu reaguje tím, že účty založí, a tím se automaticky na emailové schránky uchazečů rozešlou zprávy s přístupovými údaji.

I přestože je možné hodnocení uchazečů zapsaných na danou událost zapisovat manuálně, aplikace nabízí efektivnější způsob zpracování výsledků z matematického testu prostřednictvím integrace zpracování exportů ze systému Moodle. Moodle poskytuje export do souboru, který obsahuje uživatelská jména uživatelů, čas trvání testu a jejich bodové ohodnocení. Tento soubor lze poté v aplikaci využít pro import výsledků, což ušetří čas a úsilí spojené s manuálním vyplňováním hodnocení a minimalizuje chybovost spojenou s přepisem výsledků.

Veškerá data související s přijímacím řízením, včetně změn v rozhodnutích o přijetí, jsou uložena v lokální databázi. Pro oficiální přijetí uchazeče je však nezbytné tyto lokální informace přenést do databáze KOS. Z tohoto důvodu aplikace poskytuje funkci exportu do formátu CSV, která umožňuje manuální výběr přihlášek z tabulky, jejichž data se mají vyexportovat. Po potvrzení výběru se administrátorovi zobrazí modální okno, ve kterém musí akci potvrdit, neboť se jedná o nevratný krok. Data vybraných přihlášek se poté exportují do CSV souboru kompatibilního se systémem KOS. Po úspěšném exportu se jednotlivé přihlášky uzamknou a již nad nimi nelze provádět žádné zápisové operace. Jinými slovy, po exportu jsou přihlášky dostupné pouze pro čtení.



Obrázek 4.21 Wireframe export sekce v administrátorské sekci

4.3.3.4 Validace návrhu

Celý návrh aplikace, jak byl podrobně popsán v předchozích částech této podkapitoly, byl nejprve prezentován a důkladně zkontrolován vedoucím práce, který ověřil jeho kvalitu a relevanci. Následně proběhlo několik schůzek s autorem backendové části, jehož úkolem bylo vytvořit návrh API na základě poskytnutých wireframů. Tyto schůzky byly zaměřeny na důkladné probrání návrhu a jeho souvislostí s backendovou částí projektu. Během těchto setkání se uskutečnily také workshopy, kde byl autorovi backendové části představen celý koncept projektu včetně případů užití. Díky tomu mohl autor backendové části získat ucelený přehled o celém návrhu, což mu umožnilo lépe pochopit jeho fungování a cíle. Tento přístup měl za cíl zajistit, že obě části projektu, frontendová i backendová, budou vzájemně kompatibilní a efektivně spolupracovat. Součástí těchto setkání byly i brainstormové sezení, během kterých autor backendové části předložil nápady a návrhy, které byly následně zapracovány do výsledného hi-fi prototypu. Díky aktivnímu zapojení a spolupráci obou autorů byly dosaženy úpravy a inovace původního návrhu, které vedly k jeho zlepšení. Veškeré změny oproti původnímu návrhu budou popsány v následující části o návrhu hi-fi prototypu, kde budou prezentovány v podobě detailních mockupů.

4.3.4 Hi-fi prototyp

Hi-fi prototypování, neboli prototypování s vysokou úrovní detailu, je metoda vytváření pokročilých verzí návrhu uživatelského rozhraní s důrazem na grafické zpracování, interakce a plnou funkcionalitu. Cílem hi-fi prototypování je přiblížit se co nejvíce vzhledu a chování finální aplikace, aby bylo možné provést důkladné testování s cílovými uživateli a získat zpětnou vazbu na vizuální aspekty, funkčnost a celkovou použitelnost. [30] Hi-fi prototypy jsou vytvářeny pomocí specializovaných nástrojů a software, jako jsou Adobe XD, Sketch nebo Figma, které umožňují detailní práci s grafickými prvky a simulaci interakcí mezi jednotlivými obrazovkami.

Vývoj hi-fi prototypu v této práci navazuje na předchozí fázi lo-fi prototypování, kde byla vytvořena základní struktura uživatelského rozhraní a byly identifikovány klíčové funkcionality. Hi-fi prototyp se zaměřuje na vizuální a interakční detaily, jako jsou barevné schéma, typografie, ikony, animace a přechody mezi obrazovkami, což umožňuje realistické testování a hodnocení celkového zážitku z používání aplikace. Návrh hi-fi prototypu byl vytvořen opět s využitím aplikace Figma, která nabízí širokou škálu nástrojů a funkcí pro grafické zpracování a simulaci interakcí. Výsledný hi-fi prototyp je detailnější a obsahuje kompletní sadu obrazovek, které zahrnují všechny klíčové funkce identifikované v předchozí fázi lo-fi prototypování. Kompletní návrh hi-fi prototypu lze najít v příloze D. Jelikož hi-fi prototyp vychází z velké části z lo-fi prototypu, jsou v této části rozebrány pouze zásadní rozdíly oproti původnímu návrhu.

4.3.4.1 Minimalistický design

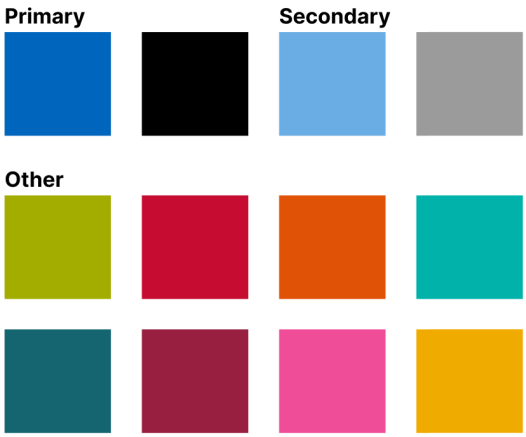
Při návrhu hi-fi prototypu je kladen velký důraz na minimalismus, což je filozofie usilující o vyřazení všech zbytečných prvků z designu. Tato metoda grafického designu klade důraz na poskytování prostoru pro pouze klíčové prvky, které mají být na první pohled viditelné. Cílem je upoutat pozornost uživatele, snížit neintuitivní prvky a zároveň dosáhnout elegantního vzhledu. V grafickém designu se minimalismus stává čím dál tím oblíbenějším přístupem. Tato strategie se snaží odstranit veškerý nepotřebný obsah z webových stránek či aplikací s cílem zjednodušit přístup k nejdůležitějším komponentám a poskytnout uživatelům lepší celkový zážitek. Ačkoliv se určité grafické aspekty mohou stávat či ztrácet na popularitě, dodržování zásad minimalismu při návrhu uživatelsky přívětivých digitálních produktů je naprosto zásadní. Uživatelé tak snadněji najdou, co hledají, rychlost načítání webových stránek je díky minimalistickému UX designu vyšší a kognitivní přetížení je sníženo. Při návrhu hi-fi prototypu jsou proto zohledněny následující vlastnosti:

- `Vizuální vodítka:` Formální složky návrhu, jako jsou písmo, barva, rozvržení a obrázky, by měly být více než jen esteticky přitažlivé. Měly by plnit funkci, jako je upoutání pozornosti uživatele nebo komunikace zamýšleného významu.
- `Rovnováha designu:` Použití velkého množství grafiky, animací nebo videí může způsobit zpomalení výkonu a přehlcujícímu uživatelskému zážitku. Rychlost aplikace lze zvýšit nalezením rovnováhy mezi designem a funkčností každé u komponenty, vzoru či interakce.
- `Využití prostoru:` Prostor je jedním z nejdůležitějších prvků minimalistického designu. Prostor, který se nachází mezi jednotlivými prvky na obrazovce, se označuje jako negativní prostor. Tento prostor mezi grafickými komponenty je důležitý, protože vytváří rovnováhu a harmonii, zlepšuje čitelnost a srozumitelnost a vede pozornost uživatele k nejdůležitějšímu obsahu nebo akcím na obrazovce. Celkově pomáhá vytvářet čisté, uspořádané a snadno použitelné rozhraní pro uživatele.
- `Jednoduchost volby:` Lidé mají tendenci cítit se zahlceni, když jim je najednou předloženo příliš mnoho možností a informací. Snížením počtu možností výběru se uživatelé mohou vyhnout pocitu zahlcení dostupnými možnostmi a požadované akce se stávají snáze pochopitelnými. [31]

4.3.4.2 Branding

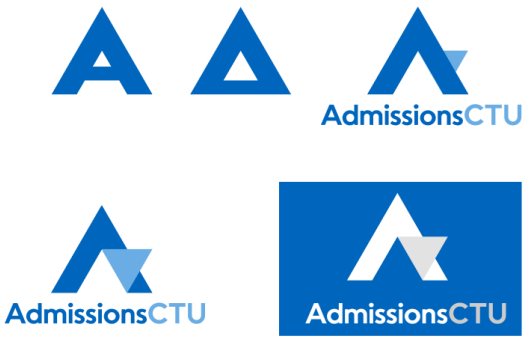
Branding hraje důležitou roli v návrhu hi-fi prototypu, protože vytváří jednotný vizuální styl, který odráží identitu univerzity a zajišťuje soudržnost mezi komponenty uživatelského rozhraní. Identita by měla být snadno rozpoznatelná a měla by reprezentovat hodnoty, které se aplikace snaží zprostředkovat uživatelům. Důvěra je dalším klíčovým faktorem pro úspěch jakéhokoli digitálního produktu, protože ovlivňuje, jak ochotni jsou uživatelé aplikaci používat a sdílet ji s ostatními. Dobře navržený a soudržný branding může právě významně přispět k budování důvěry uživatelů. V této práci je branding založen na grafickém manuálu ČVUT [32], který stanovuje pokyny pro použití loga, typografie, barev a dalších vizuálních prvků, které mají být použity při tvorbě marketingových a komunikačních materiálů pro univerzitu.

Barevné schéma, které lze vidět na obrázku 4.22, je založeno na oficiálních barvách ČVUT, které zahrnují hlavní modrou barvu a černou, doplňkovou světlejší modrou a neutrální šedou. Tyto barvy jsou použity v různých kombinacích a intenzitách v celém uživatelském rozhraní aplikace, aby bylo dosaženo soudržnosti a vizuálního kontrastu mezi jednotlivými prvky.



Obrázek 4.22 Barevné schéma podle grafického manuálu ČVUT [32]

Vývoj loga aplikace, znázorněný na obrázku 4.23, začal s analýzou existujícího loga ČVUT, které se skládá z symbolu lva a textového názvu univerzity. Cílem bylo vytvořit originální logo, které by bylo vizuálně kompatibilní s logem ČVUT a zároveň reprezentovalo aplikaci a její hlavní funkce. Proces návrhu zahrnoval různé koncepty a experimenty s formami, barvami a typografií, které vedly k výslednému logu, který se skládá z jednoduchého symbolu trojúhelníku reflektující zvolený název aplikace *AdmissionsCTU*, doplněného názvem aplikace v modré barvě.



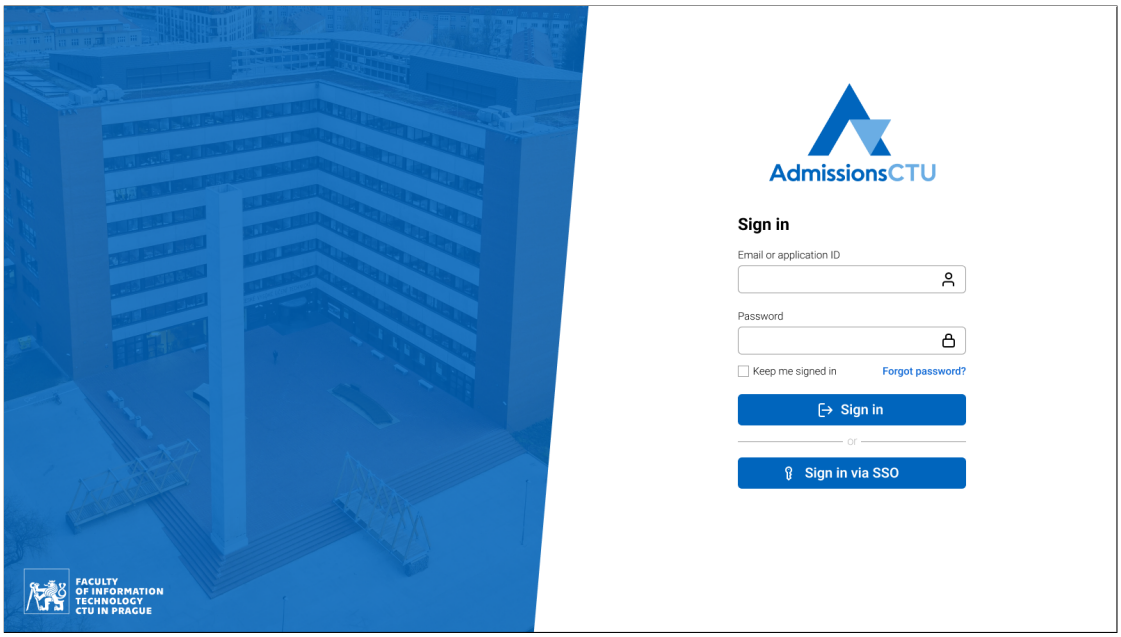
Obrázek 4.23 Proces návrhu loga aplikace

4.3.4.3 Autentizace

Rozvržení obrazovek spojené s autentizací, zobrazené na obrázku 4.24, bylo navrženo s důrazem na jednoduchost a efektivní využití prostoru, což usnadňuje celkové použití a zároveň zlepšuje vizuální dojem. V levé části obrazovky je umístěn obrázek budovy fakulty spolu s oficiálním logem ČVUT, což přispívá k vyšší důvěryhodnosti aplikace na první pohled, protože uživatelé vnímají propojení s univerzitou. V pravé části obrazovky je umístěn přihlašovací formulář, který se skládá ze dvou částí — lokálního přihlášení a tlačítka pro SSO přihlášení. Tyto dvě části byly separovány oproti původnímu návrhu, aby bylo pro administrátory snazší a intuitivnější rozpoznat, jakými údaji se mají přihlásit.

Lokální přihlášení zůstalo z velké části beze změny, avšak bylo doplněno o možnost odškrtnutí políčka, které umožňuje uživatelům zůstat dlouhodobě přihlášeni do aplikace. Díky této funkcionalitě se uživatelé po odhlášení nemusí znovu přihlašovat, což šetří čas a zvyšuje pohodlí používání aplikace.

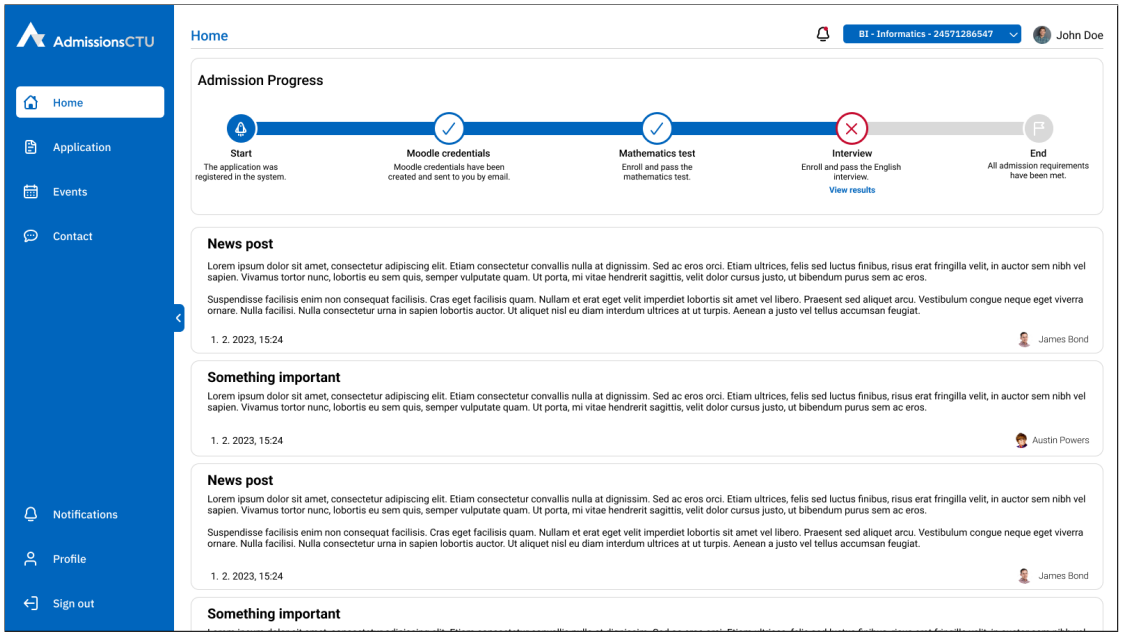
Pro administrátory bylo přidáno tlačítko pro SSO přihlášení, které je přesměruje na oficiální školní autorizační servery. Tam se administrátoři přihlásí a získají přístup do aplikace. Tento způsob přihlášení je daleko více intuitivnější a působí mnohem důvěryhodněji, jelikož probíhá prostřednictvím oficiálních univerzitních stránek, zároveň díky čemuž je zajištěna větší bezpečnost a ochrana citlivých údajů.



Obrázek 4.24 Mockup přihlašovací obrazovky

4.3.4.4 Uživatelská část

Hi-fi návrh uživatelské části, který byl vyvinut na základě původního lo-fi prototypu, prošel pouze drobnými kosmetickými úpravami, které však přispívají ke zlepšení celkového vizuálního dojmu a funkcionality aplikace. Celkové rozložení uživatelských obrazovek je zobrazeno na obrázku 4.25. Do horní části postranního menu bylo přidáno logo aplikace, které zvyšuje vizuální jednotnost a snadno identifikuje danou aplikaci. Na bok postranního menu bylo navíc přidáno tlačítko pro schování a zpětné zobrazení menu. Tato funkce umožňuje uživatelům účinněji využít obrazovku a ušetřit místo, pokud postranní menu momentálně nepotřebují.

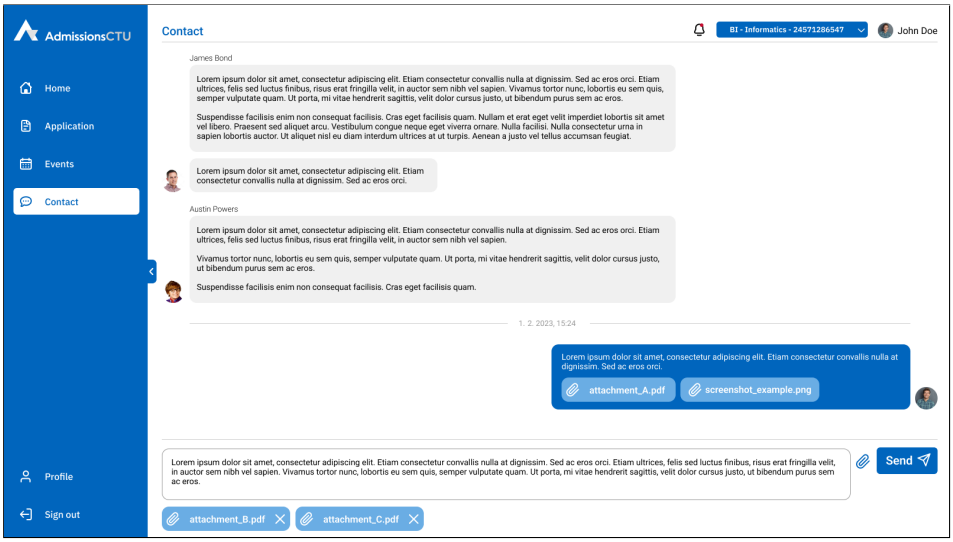


Obrázek 4.25 Mockup domovské obrazovky v uživatelské sekci

V návrhu domovské stránky byly navíc doplněny konkrétní stavy v progress baru, které přehledně znázorňují průběh přijímacího řízení pro uchazeče, jak lze vidět v obrázku 4.25. V případech, kdy uchazeč neprojde úspěšně jakoukoliv událostí, progress bar ukáže tento stav červeně, což signalizuje neúspěch. Významy jednotlivých stavů v progress baru jsou následující:

- `Start:` Přihláška byla zaevidována do systému *AdmissionsCTU*
- `Moodle credentials:` Byly vytvořeny přístupové údaje do systému Moodle
- `Mathematics test:` Uchazeč je přihlášen na termín z testu z matematiky.
- `Interview:` Uchazeč je přihlášen na termín z pohovoru.
- `End:` Uchazeč splnil všechny formální náležitosti přijímacího řízení a brzy dostane vyjádření od univerzity.

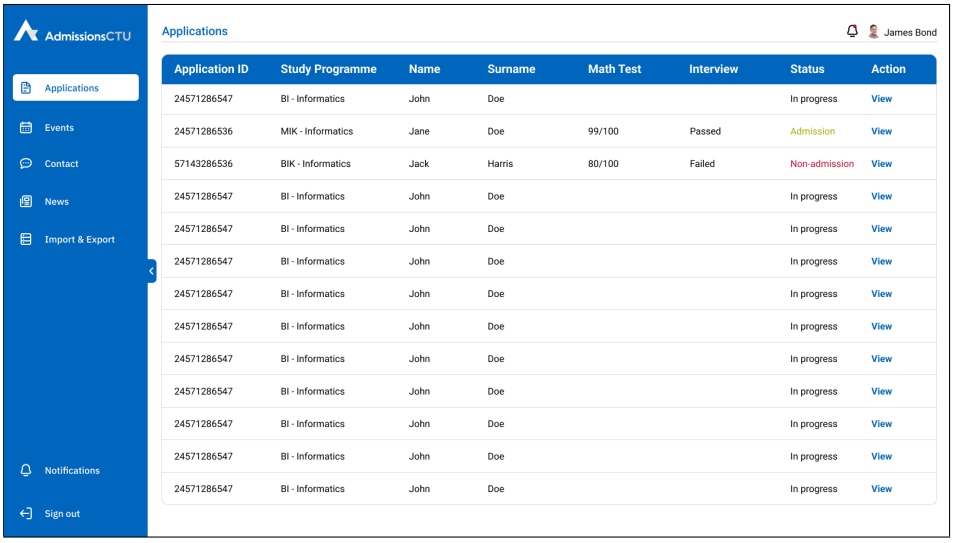
Další významnou změnou v uživatelském rozhraní je zavedení funkce nahrávání příloh, která umožňuje uchazečům přikládat důležité dokumenty, obrázky a podobné soubory do konverzací. Tato možnost přidává více flexibility a interakce mezi uchazeči a univerzitou, neboť umožňuje snadnější sdílení informací a materiálů souvisejících s přijímacím řízením. Kromě toho byla do uživatelského rozhraní zavedena možnost nahrávání profilových obrázků uživatelů. Tato funkce umožňuje uživatelům přidat osobní fotku, která se zobrazuje v jejich profilu a v konverzacích. Profilové obrázky zvyšují vizuální atraktivitu aplikace a zároveň podporují sociální interakci mezi uživateli. Přítomnost profilových obrázků také usnadňuje rozpoznání jednotlivých účastníků konverzace, což zlepšuje komunikaci a zvyšuje pocit sounáležitosti se skupinou či univerzitou.



Obrázek 4.26 Mockup konverzací v uživatelské sekci

4.3.4.5 Administrátorská část

Jedinou zásadní změnou v administrátorské části aplikace je odstranění globálního filtru z horní lišty pro filtrování z důvodu příliš veliké komplexity z hlediska vývoje. Tento krok byl učiněn na základě diskuze s autorem backendové části, protože implementace tohoto filtru by vyžadovala značné úsilí a zdroje ze strany vývojářů. Jelikož se nejednalo o požadovanou funkcionalitu, ale pouze o prvek čistě pro vylepšení uživatelského zážitku, bylo rozhodnuto, že je možné tento požadavek bez následků vynechat. Tímto rozhodnutím se projekt zaměřil na základní funkcionality a požadavky, které byly považovány za klíčové pro úspěšné splnění cílů projektu. Výsledné rozložení obrazovek administrátorské části lze pozorovat na obrázku 4.27.



Obrázek 4.27 Mockup přehledu přihlášek v administrátorské sekci

4.4 Shrnutí

Nejprve byl v návrhu kladen důraz na definici aplikační architektury frontendové části. Frontendová aplikace byla rozdělena do funkčních modulů, které vycházely z identifikovaných subdomén, což umožnilo lepší organizaci a efektivnější vývoj. Následně bylo věnováno úsilí strukturování samotných funkčních modulů s cílem zvýšit udržitelnost a rozšiřitelnost aplikace, což je důležité pro dlouhodobý vývoj.

Poté byla domluvena schůzka s autorem backendové části, na základě které byla navržena infrastruktura celého systému. Během této schůzky byly identifikovány externí systémy a jejich role, které pomohly k lepšímu porozumění celkového fungování aplikace a jejího začlenění do existujících technologií a procesů.

Pro podporu vývoje a validaci požadavků byl nejprve vytvořen lo-fi prototyp za pomocí wireframů. Wireframy zobrazovaly klíčové funkcionality a procesy, díky čemuž bylo možné rychle identifikovat potencionální problémy a zlepšit komunikaci. Na základě těchto zvalidovaných wireframů byl následně vytvořen hi-fi prototyp, který byl navržen s ohledem na analýzu grafického manuálu ČVUT. Při návrhu hi-fi prototypu byl kladen důraz na jednoduchost použití a funkčnost, aby byla aplikace přehledná a snadno ovladatelná pro uživatele.

Díky tomuto návrhu byly poskytnuty detailní podklady, na základě kterých se bude stavit implementační část. Tyto podklady umožní zapojeným stranám lépe porozumět požadavkům na aplikaci a zjednoduší koordinaci mezi frontendovým a backendovým vývojem.

Kapitola 5
Implementace

Po důkladném zkoumání všech specifik domény přijímacího řízení pro anglické studijní programy a následném komplexním návrhu aplikace, byly úspěšně splněny veškeré požadavky umožňující její implementaci. V úvodu této kapitoly jsou popsány klíčové části implementace klientské části aplikace. Následně je představeno propojení se serverovou částí a na závěr jsou diskutovány aspekty týkající se nasazení aplikace na univerzitní servery.

5.1 Implementace frontendové části

V této podkapitole budou probrány zásadní metody a principy, které byly využity pro implementace klientské části aplikace. Kompletní zdrojový kód frontendové části lze nalézt v přiloženém médiu této práce.

5.1.1 Sestavovací nástroj

V současnosti je pro vývojáře k dispozici široká škála nástrojů určených pro vývoj webových aplikací. Jedním z nejdůležitějších rozhodnutí, která musí být vývojářem učiněna, je volba nástroje pro sestavení aplikace. Mezi dvěma nejoblíbenějšími nástroji pro sestavování patří Create React App a Vite, oba jsou výkonné nástroje, jež lze využít pro vytváření efektivních, optimalizovaných a škálovatelných webových aplikací. Vite a Create React App mají své silné stránky a omezení, jež je třeba zohlednit při rozhodování, který z nich použít. Create React App je oficiálním nástrojem pro vytváření nových projektů v Reactu, poskytujícím dobře promyšlenou a zkušenostmi ověřenou konfiguraci. Na druhé straně, Vite je modernější a rychlejší nástroj, který nabízí vysokou úroveň konfigurovatelnosti a optimalizace. [33]

Na základě vysoké konfigurovatelnosti, popularity a jednoduchosti použití byl pro vývoj aplikace vybrán nástroj Vite. Tento nástroj je založen na technologii ES modules, která umožňuje rychlejší vývoj a menší velikost výsledných souborů. Jeho jednoduchost a flexibilita poskytují vývojářům možnost snadno přizpůsobit konfiguraci podle potřeb projektu. Navíc, díky své rostoucí popularitě a aktivní komunitě je Vite podporován širokou škálou modulů a integrací, což usnadňuje práci s různými knihovnami a nástroji.

Vite poskytuje několik konkrétních příkazů, které usnadňují práci při vývoji a nasazení webových aplikací. Tyto příkazy jsou dostupné prostřednictvím příkazové řádky a zahrnují:

- `vite`: Tento příkaz spustí vývojový server, který automaticky kompiluje a načítá změny v kódu během vývoje. Vývojový server poskytuje také funkci *hot module replacement*, která umožňuje rychlé aktualizace aplikace při jakékoliv změně v kódu bez nutnosti jejího obnovení.

- `vite build`: Tento příkaz slouží pro sestavení produkční verze aplikace. Výstupem je optimalizovaný balíček, který je připraven pro nasazení na server. Během sestavování aplikace Vite provádí optimalizace, jako je minifikace, třídění CSS a odstranění nepoužitého kódu, což zajišťuje co nejmenší velikost výsledných souborů.

- `vite preview`: Tento příkaz umožňuje vývojářům zobrazit produkční verzi aplikace na lokálním serveru. Tato funkce je užitečná pro kontrolu aplikace před jejím nasazením na produkční prostředí.

Výpis kódu 5.1 Použitá konfigurace pro nástroj Vite

```
// vite.config.js

import react from '@vitejs/plugin-react';
import { defineConfig } from 'vite';

// https://vitejs.dev/config/
export default defineConfig({
  base: '/app/',
  plugins: [react()],
  build: {
    outDir: './build',
  },
  server: {
    watch: {
      usePolling: true,
    },
    host: true,
    strictPort: true,
    port: 3000,
  },
});
```

Vite umožňuje vývojářům upravit konfiguraci prostřednictvím souboru `vite.config.js`. Tento soubor umožňuje nastavit různé parametry a možnosti, které ovlivňují chování Vite během vývoje a sestavování aplikace. Výše uvedený kód 5.1 popisuje konfiguraci Vite použitou v při vývoji aplikace, kde význam nastavených parametrů je následující:

- `plugins`: Tato volba definuje moduly, jež budou použity Vite během vývoje a sestavování aplikace. V tomto případě je použit modul `@vitejs/plugin-react`, umožňující použití Reactu v aplikaci.

- `build`: Tato část konfigurace zahrnuje nastavení pro sestavování produkční verze aplikace. Zde je výstupní adresář `outDir` nastaven na `./build`, což znamená, že sestavená aplikace bude uložena v adresáři `build`.

- `server`: Tato konfigurace zahrnuje nastavení pro vývojový server Vite. Nastavení `watch` s možností `usePolling` zajišťuje, že jakékoliv změny provedené během vývoje budou do aplikace automaticky zapracovány bez nutnosti jejího znovuspuštění. Konfigurace `host` umožňuje přístup k vývojovému serveru z externích zařízení, což je užitečné pro kontejnerizaci. Možnost `strictPort` zajišťuje, že pokud je port již obsazen, server se nezapne na jiném portu. Navíc `port` s hodnotou 3000 nastavuje port, na kterém bude vývojový server spuštěn.

5.1.2 Statické typování

Statické typování je koncept, který spočívá v předem známém určení typů proměnných, funkcí a dalších entit v kódu před jeho spuštěním. Tento přístup přináší několik výhod, které zvyšují efektivitu vývoje, spolehlivost aplikace a udržitelnost kódu. Jednou z hlavních výhod statického typování je detekce chyb. Statické typování umožňuje kompilátorům a nástrojům pro statickou analýzu kódu identifikovat a předcházet potenciálním chybám typů ještě před spuštěním aplikace. Díky tomu se snižuje pravděpodobnost chyb v produkčním prostředí, což zvyšuje spolehlivost celé aplikace. Další výhodou statického typování je zlepšení čitelnosti a udržitelnosti kódu. Typy proměnných a funkcí jsou přímo definovány v kódu, což usnadňuje orientaci v kódu a jeho údržbu. Tato vlastnost je zvláště důležitá v projektech většího rozsahu, kde může být obtížnější udržet přehled nad funkcemi a proměnnými. Ačkoli jazyk JavaScript je dynamicky typovaný, existují způsoby, jak do něj zavést statické typování. Jedním z takových způsobů je použití jazyka TypeScript, který je nadmnožinou JavaScriptu a přidává možnost definovat statické typy. [34]

Výpis kódu 5.2 Použitá konfigurace pro nastavení TypeScript

```
// tsconfig.json
{
  "compilerOptions": {
    "outDir": "build/dist",
    "target": "es5",
    "lib": [
      "dom",
      "dom.iterable",
      "esnext"
    ],
    "allowJs": true,
    "skipLibCheck": true,
    "esModuleInterop": true,
    "allowSyntheticDefaultImports": true,
    "strict": true,
    "forceConsistentCasingInFileNames": true,
    "noFallthroughCasesInSwitch": true,
    "module": "esnext",
    "moduleResolution": "node",
    "resolveJsonModule": true,
    "isolatedModules": true,
    "noEmit": true,
    "jsx": "react-jsx"
  },
  "include": [
    "src",
    "index.d.ts"
  ],
  "exclude": [
    "node_modules"
  ],
  "baseUrl": ".",
  "paths": {
    "@/*": [
      "./src/*"
    ]
  }
}
```

V kódu 5.2 je uvedená vlastní konfigurace TypeScript použitou během vývoje. Díky této konfiguraci lze ovládat chování, kompatibilitu, rozsah kontroly a další vlastnosti. Mezi nejdůležitější nastavení této konfigurace patří:

- `outDir`: Tento atribut specifikuje výstupní adresář pro zkompilované TypeScript soubory. V tomto projektu je nastaven na `build/dist`.
- `target`: Tento atribut určuje cílovou verzi ECMAScript, do které se má TypeScript kód kompilovat. Je nastaven na es5 pro širší kompatibilitu s webovými prohlížeči, aby bylo zajištěno, že aplikace bude fungovat správně i na starších prohlížečích a platformách.
- `lib`: Tento atribut zahrnuje seznam knihoven, které mají být zahrnuty do kompilace. V tomto případě jsou zahrnuty knihovny `dom`, `dom.iterable` a `esnext` pro poskytnutí širokého spektra funkcí. `Dom` poskytuje definice typů a API pro interakci s `DOM`, `dom.iterable` poskyutje nástroje pro iteraci přes kolekce DOM prvků, a `esnext` poskytuje nejnovější funkcionalitu ECMAScript.
- `strict`: Tato možnost povoluje přísnou kontrolu typů, která vynucuje silnější bezpečnost typů a pomáhá předcházet potenciálním chybám souvisejícím s typy. Když je nastaveno na `true`, kompilátor TypeScript zkontroluje kód s přísnějšími pravidly pro typy a zamezí použití implicitních typů `any`, což zvyšuje spolehlivost a bezpečnost kódu.
- `module`: Tento atribut nastavuje systém modulů použitý pro výstupní kód, který je nastaven na `esnext`. `Esnext` umožňuje použití nejnovějších funkcí modulů ECMAScript, jako jsou například dynamické importy, což zlepšuje modularitu a škálovatelnost kódu.
- `jsx`: Tento atribut definuje režim transformace JSX, který je nastaven na `react-jsx` pro kompatibilitu s React frameworkem. Díky tomuto nastavení je možné používat JSX syntaxi přímo v TypeScript souborech, což zjednodušuje vývoj React komponent.

Použitím TypeScriptu v projektu lze definovat vlastní typy. V ukázce 5.3 lze vidět příklad použití TypeScriptu pro definici modelové třídy uživatele, kterou aplikaci využívá k ukládání uživatelských dat.

Výpis kódu 5.3 Ukázka definice vlastní typů

```
export class User {
  id: number;
  username: string;
  fullName: string;
  firstName: string;
  lastName: string;

  constructor(
    id: number,
    username: string,
    fullName: string,
    firstName: string,
    lastName: string
  ) {
    this.id = id;
    this.username = username;
    this.fullName = fullName;
    this.firstName = firstName;
    this.lastName = lastName;
  }
}
```

5.1.3 Definice komponent

Pro definici grafických prvků v aplikaci byla ve většině situací použity funkční komponenty a react hooks, jejichž přednosti byly podrobně popsány v podkapitole o technické analýze knihovny react. Navíc bylo důležité zahrnout aspekty typové bezpečnosti a funkcionalit, které poskytuje jazyk TypeScript.

V následujícím příkladě 5.4 je prezentován styl implementace komponent, který je předveden na definici specializovaného tlačítka s integrovanou funkcí načítací animace. Tato komponenta reaguje na výpočet prováděný asynchronní funkcí a zobrazuje animaci, dokud výpočet není ukončen. Při interakci s tlačítkem, konkrétně stisknutím, je spouštěna interně definovaná funkce `handleClick`. Tato funkce modifikuje stav komponenty a tím ovlivňuje její vizuální prezentaci. Veškeré komponenty aplikace využívají TypeScript alespoň pro definování vstupních parametrů, což v kódu napomáhá udržet přehled o atributech jednotlivých komponent a identifikovat povinné parametry. V uvedeném příkladu jde vidět, že komponenta definuje typ `LoadingButtonProps`, který zahrnuje povinný parametr `onClick`, specifikující akci, jež má komponenta vykonat při stisknutí. Interní stavy komponent jsou uchován pomocí hooku `useState`, který v tomto případě definuje `isProcessing` pro uložení stavu načítání a `setIsProcessing` pro modifikaci tohoto stavu.

Výpis kódu 5.4 Ukázka definice vlastní komponenty

```
import { Button, ButtonProps } from '@chakra-ui/react';
import React, { MouseEventHandler, useState } from 'react';

type LoadingButtonProps = ButtonProps & {
  onClick: (event: MouseEventHandler) => any;
};

export function LoadingButton({
  children,
  onClick,
  ...props
}: LoadingButtonProps) {
  const [isProcessing, setIsProcessing] = useState(false);

  const handleClick = async (event: MouseEvent) => {
    if (onClick === undefined) {
      return;
    }

    setIsProcessing(true);
    const result: any = onClick(event);
    if (result instanceof Promise) {
      await result;
    }

    setIsProcessing(false);
  };

  return (
    <Button isLoading={isProcessing} onClick={handleClick} {...props}>
      {children}
    </Button>
  );
}
```

5.1.4 Vzhled a stylování

Pro stylování aplikace a jejích komponent byla použita knihovna Chakra UI. Chakra UI je populární, moderní a vysoce konfigurovatelná knihovna pro React, která se zaměřuje na snadné a rychlé vytváření responzivních a vysoce použitelných aplikací. Jedním z hlavních přínosů knihovny je poskytnutí sady předdefinovaných komponent s přednastavenými styly, které mohou být jednoduše použity a rozšířeny podle potřeb projektu. Díky tomu se lze soustředit na vytváření funkcionality aplikace bez nutnosti věnovat příliš mnoho času vlastnímu designu. Chakra UI se také zaměřuje na komponentový přístup, což umožňuje snadno přizpůsobit vzhled a chování jednotlivých komponent nezávisle na ostatních. Navíc knihovna nabízí širokou paletu užitečných a intuitivních podpůrných funkcí, které usnadňují práci s responsivním designem, přístupností a dalšími aspekty moderního vývoje webových aplikací. Jednou z výhod Chakra UI je, že nepotřebuje žádné dodatečné CSS pro stylování. Chakra UI používá *inline styling*, což znamená, že styly jsou aplikovány přímo na jednotlivé komponenty, a to pomocí JavaScriptu. Toto usnadňuje správu stylů a eliminuje nutnost používat externí stylovací knihovny nebo vytvářet samostatné CSS soubory, které by redudantními soubory pro každou komponentu znehodnocovaly přehlednost projektu. Díky Chakra UI lze dosáhnout požadovaného vzhledu aplikace s menším množstvím kódu a snadnější údržbou. [35]

Výpis kódu 5.5 Ukázka použití knihovny ChakraUI pro definování komponenty notifikace

```
type NotificationProps = {
  notification: UserNotification;
  onClick: (id: number) => void;
};

function Notification({ onClick, notification }: NotificationProps) {
  return (
    <Box
      alignItems="start"
      p={3}
      bg={colorPalette.cardSecondary}
      rounded="lg"
      w="full"
      {...(!notification.isSeen && {
        cursor: 'pointer',
        onClick: () => onClick(notification.id),
      })}
    >
      <HStack alignItems="end" justifyContent="space-between" w="full">
        <HStack>
          {!notification.isSeen &&
            <Circle size={3} bg={colorPalette.alertRed} />
        }
          <Text fontWeight="bold" fontSize="xl">
            {notification.title}
          </Text>
        </HStack>
        <Text size="sm" color={colorPalette.border}>
          {notification.createdAt.toLocaleString()}
        </Text>
      </HStack>
      <Text>{notification.description}</Text>
    </Box>
  );
}
```

V ukázce kódu 5.5 je demonstrováno použití knihovny Chakra UI pro definování komponenty notifikace. Tato komponenta je vytvořena skládáním různých dílčích komponent poskytovaných knihovnou Chakra UI, jako jsou `Box`, `HStack` a `Text`. V kódu je vidět, že jednotlivé atributy, které by normálně byly nastavovány pomocí CSS, jsou nyní nastaveny přímo v rámci jednotlivých komponent. Například pro komponentu `Box` je definováno odsazení `p`, barva pozadí `bg`, zaoblení rohů `rounded`, šířka `w` a další. Díky tomu umožňuje Chakra UI snadno a rychle vytvářet komponenty s požadovanými styly bez nutnosti psaní externího CSS.

ChakraUI také umožňuje konfigurovat knihovní komponenty a nastavit jim výchozí vzhled či hodnoty atributů. Tato funkcionalita umožňuje globálně přizpůsobit vzhled aplikace podle vlastních potřeb a preferencí. Konfigurace stylů a vlastností komponent knihovny ChakraUI může být realizována pomocí metody `extendTheme`. Část definice vlastního stylu pro ChakraUI použitou v aplikaci je zobrazena v ukázce 5.6. V ukázce jsou definovány barvy pomocí objektu `colorScheme`, který obsahuje hodnoty pro primární barvu komponent, pozadí a další barvy. Dále je definován objekt `componentTheme`, který slouží k nastavení vlastností jednotlivých komponent, jako je například velikost nebo barva. Toto nastavení lze pak uplatnit pomocí speciálního knihovnou poskytovaného provideru `CharkraProvider` přijímající atribut `theme`, do kterého je předán objekt `themeConfig` obsahující výše popsané nastavení.

Výpis kódu 5.6 Konfigurace výchozího vzhledu komponent knihovny ChakraUI

```
export const themeConfig = extendTheme({
  colors: { ...colorScheme },
  components: { ...componentTheme },
});

export const colorScheme = {
  primary: colorPalette.primary,
  bgColor: colorPalette.textPrimary,
  ...
};

export const componentTheme = {
  Spinner: {
    defaultProps: {
      size: 'xl',
    },
    baseStyle: {
      color: colorPalette.primary,
      borderWidth: '5px',
    },
  },
  ...
};

export function AppProvider() {
  return (
    <Provider store={store}>
      <AuthProvider>
        <ChakraProvider theme={themeConfig}>
          <RouterProvider router={appRouter} />
        </ChakraProvider>
      </AuthProvider>
    </Provider>
  );
}
```

5.1.5 Routování

Single Page Applications představují koncept, kdy je celá webová aplikace spuštěna v prohlížeči jako jedna stránka, a její obsah se dynamicky mění pomocí přímé modifikace DOM pomocí JavaScriptu na základě interakcí uživatele s aplikací. Při přechodu mezi různými obrazovkami nevyžaduje stránku celou znovu načíst, což vede k plynulému chování a rychlé odezvě aplikace. Pro implementaci SPA byla vybrána knihovna React Router, která se používá společně s Reactem pro implementaci routování mezi více obrazovkami v jednostránkových aplikacích.

Výpis kódu 5.7 Ukázka použití knihovny React Router pro implementaci routování

```
export const appRouter = createBrowserRouter(
  [
    {
      path: '/',
      element: <Navigate to={`/${appPaths.authentication}`} />,
    },
    {
      path: `/${appPaths.authentication}`,
      children: <LoginPage />,
    },

    ...

  ],
  {
    basename: `/${routerConfig.routePrefix}`,
  }
);

...

export function AppProvider() {
  return (
    <Provider store={store}>
      <AuthProvider>
        <ChakraProvider theme={themeConfig}>
          <RouterProvider router={appRouter} />
        </ChakraProvider>
      </AuthProvider>
    </Provider>
  );
}
```

V ukázce kódu 5.7 je demonstrováno použití knihovny React Router pro implementaci routování. Nejprve je vytvořen router objekt pomocí metody `createBrowserRouter`. Tato metoda přijímá pole objektů, které reprezentují jednotlivé cesty a jejich příslušné komponenty, a jako druhý parametr objekt pro nastavení chování routování. V aplikaci je nastaven parametr `basename`, který slouží pro nastavení základní URL cesty, jež se globálně připne ke všem cestám. V příkladu lze vidět dvě cesty první cesta je definována jako hlavní stránka aplikace a při jejím načtení je uživatel přesměrován na autentizační stránku pomocí komponenty `Navigate`. Druhá cesta, definovaná v konstantě `appPaths.authentication`, obsahuje komponentu reprezentující přihlašovací obrazovku. Router objekt je následně předán do komponenty `RouterProvider`, která slouží jako vstupní bod pro routování v aplikaci a inicializuje mechanismus navigace. Komponenta `RouterProvider` je umístěna uvnitř komponenty `AppProvider`, která slouží jako kořenová komponenta aplikace.

5.1.6 Kontejnerizace

Kontejnerizace je proces zabalení aplikace spolu se všemi jejími závislostmi a konfiguracemi do jednotného balíčku, který se nazývá kontejner. Kontejnery umožňují spouštět aplikace v izolovaném prostředí a zajišťují konzistentní chování napříč různými vývojovými, testovacími a produkčními prostředími. Pro zajištění platformové nezávislosti aplikace byly použity nástroje Docker a Docker Compose, které umožňují snadné vytváření a správu kontejnerů pro aplikaci.

5.1.6.1 Lokální vývoj

Lokální vývojové prostředí poskytuje možnost pracovat na aplikaci v izolovaném prostředí, jež se podobá produkčnímu prostředí, avšak umožňuje rychlejší testování a ladění aplikace během vývoje. V ukázce kódu 5.8 je prezentován dockerfile určený pro lokální vývoj. Tento soubor obsahuje pokyny pro vytvoření docker kontejneru založeného na obrazu `node:alpine3.17`, který zahrnuje JavaScriptové běhové prostředí NodeJs. Pracovní adresář `app` je nastaven jako aktuální pracovní adresář kontejneru. Následně jsou zkopírovány soubory `package.json` a `package-lock.json` obsahující informace o závislostech aplikace. Tyto závislosti jsou pak nainstalovány pomocí příkazu `npm install`. Nakonec jsou do kontejneru přeneseny zdrojové kódy aplikace a spuštěn vývojový server prostřednictvím příkazu `npm run dev`, což umožňuje rychlé testování a ladění aplikace.

Výpis kódu 5.8 Definice dockerfile pro lokální vývoj

```
FROM node:alpine3.17
WORKDIR /app
COPY package.json .
COPY package-lock.json .
RUN npm install
COPY . .
CMD ["npm", "run", "dev"]
```

V ukázce 5.9 lze dále vidět Docker Compose určený pro lokální vývoj. Zde je definována služba `client`, která vychází z výše zmíněné dockerfile definice. Služba nastavuje sdílené svazky pro synchronizaci zdrojových kódů aplikace a knihovny `node_modules`. Sdílení zdrojových kódů mezi hostitelským systémem a kontejnerem umožňuje využití funkce *Hot Module Replacement*, díky níž jsou změny v kódu aplikovány do kontejneru bez nutnosti nového sestavení. Port 3000 je zpřístupněn na hostitelském systému, čímž je zajištěn přístup k aplikaci běžící v kontejneru.

Výpis kódu 5.9 Definice dokcer compose pro lokální vývoj

```
version: "3.8"
services:
  client:
    image: admissions_ctu_frontend_dev
    build:
      context: ../..
      dockerfile: deploy/dev/Dockerfile.dev
    volumes:
      - ../../:/app
      - ../../node_modules:/app/node_modules/
    ports:
      - "3000:3000"
    environment:
      - NODE_ENV=development
volumes:
  node_modules:
```

5.1.6.2 Produkční spuštění

Produkční prostředí je optimalizováno pro výkon a bezpečnost aplikace, která je připravena k nasazení. V produkčním prostředí je aplikace zkompilována a spuštěna s využitím webového serveru Nginx, který je optimalizován pro doručování statických souborů a zajišťuje rychlejší odezvu aplikace. V níže uvedeném kódu 5.10 je definována konfigurace Nginx pro produkční spuštění. Server naslouchá na portu 80 a očekává požadavky na adrese `localhost`. Konfigurace serveru zajišťuje, že všechny požadavky na URL cestě `app` jsou směrovány na aplikaci ke zpracování.

Výpis kódu 5.10 Definice nginx pro produkční spuštění

```
server {
    listen       80;
    server_name  localhost;

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

    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
        root   /usr/share/nginx/html;
    }
}
```

Výpis kódu 5.11 zobrazuje dockerfile pro produkční spuštění. Tento soubor je založen na stejném obrazu jako jeho lokální verze. Pracovní adresář `app` je opět nastaven jako aktuální pracovní adresář kontejneru. Poté jsou zkopírovány soubory `package.json` a `package-lock.json` a nainstalovány závislosti pomocí příkazu `npm ci`. Následně jsou do kontejneru zkopírovány zdrojové kódy aplikace a spuštěn příkaz `npm run build`, který zkompiluje aplikaci pro produkční prostředí. V další části dockerfile je použit obraz `nginx:1.21-alpine` jako obraz pro webový server. Zkompilovaná aplikace je zkopírována do adresáře `/usr/share/nginx/html/app`, což je umístění, kde nginx očekává statické soubory aplikace. Konfigurační soubor `nginx.conf` je zkopírován do `/etc/nginx/conf.d/default.conf`, čímž se upraví konfigurace webového serveru tak, aby správně odkazoval na webovou aplikaci. Nakonec je webový server spuštěn na portu 80, což je standardní port pro HTTP komunikaci.

Výpis kódu 5.11 Definice dockerfile pro produkční spuštění

```
FROM node:17-alpine
WORKDIR /app
COPY package*.json ./
RUN npm ci
COPY . .
RUN npm run build
FROM nginx:1.21-alpine
COPY --from=0 /app/build /usr/share/nginx/html/app
COPY deploy/prod/nginx.conf /etc/nginx/conf.d/default.conf
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Poslední součástí produkčního spuštění je definice docker compose uvedená v ukázce 5.12. V této definici je opět vytvořena služba `client`, která využívá dockerfile z výše uvedeného příkladu 5.11. V docker compose je port 8080 na hostitelském systému propojen s portem 80 kontejneru. Tímto propojením se zajistí, že aplikace běžící v kontejneru na portu 80 bude dostupná na hostitelském systému na portu 8080.

Výpis kódu 5.12 Definice dokcer compose pro produkční spuštění

```
version: "3.8"

services:
  client:
    image: admissions_ctu_frontend
    build:
      context: ../..
      dockerfile: deploy/prod/Dockerfile.prod
    ports:
      - "8080:80"
    environment:
      - NODE_ENV=production
```

5.2 Napojení na backendovou část

Během návrhu vyvíjené aplikace bylo definováno API mezi frontendovou a backendovou částí. Díky tomuto předdefinovanému rozhraní mohly obě části aplikace vznikat paralelně, což urychlilo celkový vývoj. Cílem této podkapitoly je popsat důležité části implementace, které byly důležité pro sjednocení obou řešení do výsledné aplikace.

5.2.1 Popis backendové části

Backendové řešení je založeno na architektuře mikroslužeb a využívá širokou škálu technologií, jako jsou Spring Boot, Apache Kafka, RSQL a PostgreSQL. Návrh a zvolené technologie se zaměřují na vysokou granularitu dekompozice služeb, s cílem vytvořit vysoce škálovatelný a rozšiřitelný backend, který se snadno přizpůsobí různým potřebám a požadavkům.

Jednotlivé serverové služby jsou implementovány jako samostatné aplikace v technologii Spring Boot, které spolu navzájem komunikují prostřednictvím systému Apache Kafka. Tento systém, distribuovaná platforma pro zpracování proudů událostí a výměnu zpráv, umožňuje rychlou a efektivní komunikaci mezi službami.

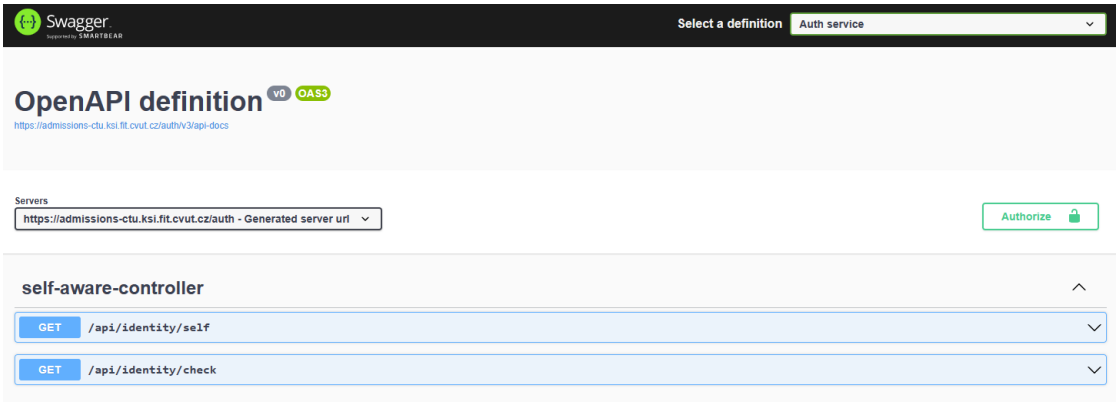
Lokální data aplikace jsou uchovávána v databázi PostgreSQL, která je synchronizována s databází KOS prostřednictvím dedikované synchronizační služby. Synchronizace je automaticky spouštěna pomocí cron jobů, což zajišťuje pravidelné a důsledné aktualizace dat.

Vzhledem k tomu, že aplikace obsahuje velké množství funkcí spojených s filtrováním dat, byla zodpovědnost za implementaci filtrování přenechána frontendové části. Integrace dotazového jazyka RSQL usnadňuje tento úkol, neboť umožňuje frontendové části snadno a efektivně pracovat s daty a filtrovat je na základě různých parametrů.

Pro podporu frontendového vývoje byla vystavena speciální služba, na které běží Swagger. Tento nástroj sjednocuje RESTové API definice všech služeb a poskytuje frontendu jasný a ucelený přehled o dostupných API. Tímto způsobem je práce na frontendové části při integraci s backendovými službami usnadněna.

5.2.2 Komunikace s backendovou částí

Pro komunikaci mezi frontendovou a backendovou částí aplikace je nezbytné provádět API volání. Během vývoje frontendové části byla pro implementaci těchto volání vybrána knihovna Axios. Axios je široce používaná JavaScriptová knihovna, která umožňuje snadnou komunikaci s RESTovými API prostřednictvím HTTP požadavků.



Obrázek 5.1 Dokumentace API poskytovaná backendovou částí pro snadnější integraci a ladění

Na základě API definice, kterou autor backendové části poskytl prostřednictvím služby Swagger, byly pro každý endpoint ve frontendové části vytvořeny funkce pro jejich volání. API dokumentace sjednocuje endpointy ve všech mikroslužbách, což umožňuje komplexní přehled o dostupných funkcích ve všech službách. Ve Swagger dokumentaci lze nalézt příklady požadavků a odpovědí, které umožňují ve frontendové aplikaci definovat datové typy na základě schémat ve Swaggeru díky použití TypeScriptu. Swagger je plně interaktivní a umožňuje posílat požadavky a získávat odpovědi přímo uvnitř nástroje. Tato funkce usnadňuje ladění a testování API volání ještě před implementací kódu ve frontendové části.

Výpis kódu 5.13 Ukázka volání API pomocí knihovny Axios

```
export async function getUserDetails(id: number): Promise<User> {
  const accountResponse = await axios.get(
    evaluateParams(backendPaths.userDetail, { id: id })
  );

  const accountData = accountResponse.data;

  return new User(
    accountData.id,
    accountData.username,
    accountData.fullName,
    accountData.firstName,
    accountData.lastName
  );
}
```

Výše uvedený kód 5.13 demonstruje příklad funkce `getUserDetails`, která komunikuje s backendovým endpointem za účelem získání detailů uživatele. Tato funkce přijímá `id` uživatele jako parametr a vrátí `User` objekt obsahující výsledná uživatelská data. Axios provádí HTTP GET požadavek na URL, která je vyhodnocena pomocí funkce `evaluateParams` a následně předána jako parametr. Jakmile je odpověď získána, extrahují se data a vytvoří se nový objekt User s příslušnými atributy. Kromě dat navracených backendem, obsahuje odpověď navracená knihovnou různé metadata o požadavku a odpovědi, jako jsou návratový kód, časová razítka a další informace užitečné pro ladění a řešení problémů. Pokud v rámci API volání backend navrátí chybný stavový kód, knihovna axios vyhodí vyjímku, kterou lze ošetřit v komponentě na frontendu. Tímto způsobem je možné efektivně reagovat na chybové stavy a informovat uživatele o případných problémech při komunikaci s backendem.

Poté, co jsou definovány funkce pro komunikaci s backendovou částí, je možné je libovolně volat v komponentách a vykreslit navrácená data do uživatelského rozhraní. Ukázkovým příkladem je komponenta pro načítání konverzace v kódu 5.14. Pro načítání konverzací se pomocí vlastního hooku `useApplication`, který stahuje data o aktuální přihlášce uživatele, získá identifikátor konverzace a předá se funkci `getConversationDetail`. Jelikož je komunikace s backendem asynchronní operace, je vhodné během čekání na zpracování požadavku zobrazit načítací animaci. Tímto způsobem dostává uživatel zpětnou vazbu, že aplikace zpracovává jeho požadavek, což snižuje frustraci uživatele spojenou s čekáním.

Výpis kódu 5.14 Načtení dat konverzací a jejich vykreslení v komponentě

```
export function Contact() {
  const { selected } = useApplication();
  const conversationId = selected.conversationId;
  const conversationLoader = () => getConversationDetail(conversationId);
  return (
    <DataLoader loadingFunction={conversationLoader}>
      {(data) => <MessageList conversationId={data.id} />}
    </DataLoader>
  );
}
```

Definice komponenty, zobrazující načítací animaci, lze vidět v níže uvedeném kódu 5.15. Tato komponenta si udržuje vnitřní stav dat, která je při první inicializaci nedefinována. Dokud jsou data nedefinována, zobrazí se načítací animace. Po doběhnutí zpracování požadavku se zavolá funkce `setData`, která nastaví hodnotu dat a vykreslí svého potomka. Tento postup umožňuje snadno integrovat načítání dat do komponent a zajistit plynulý přechod mezi stavy aplikace.

Výpis kódu 5.15 Komponenta zobrazující načítací animaci

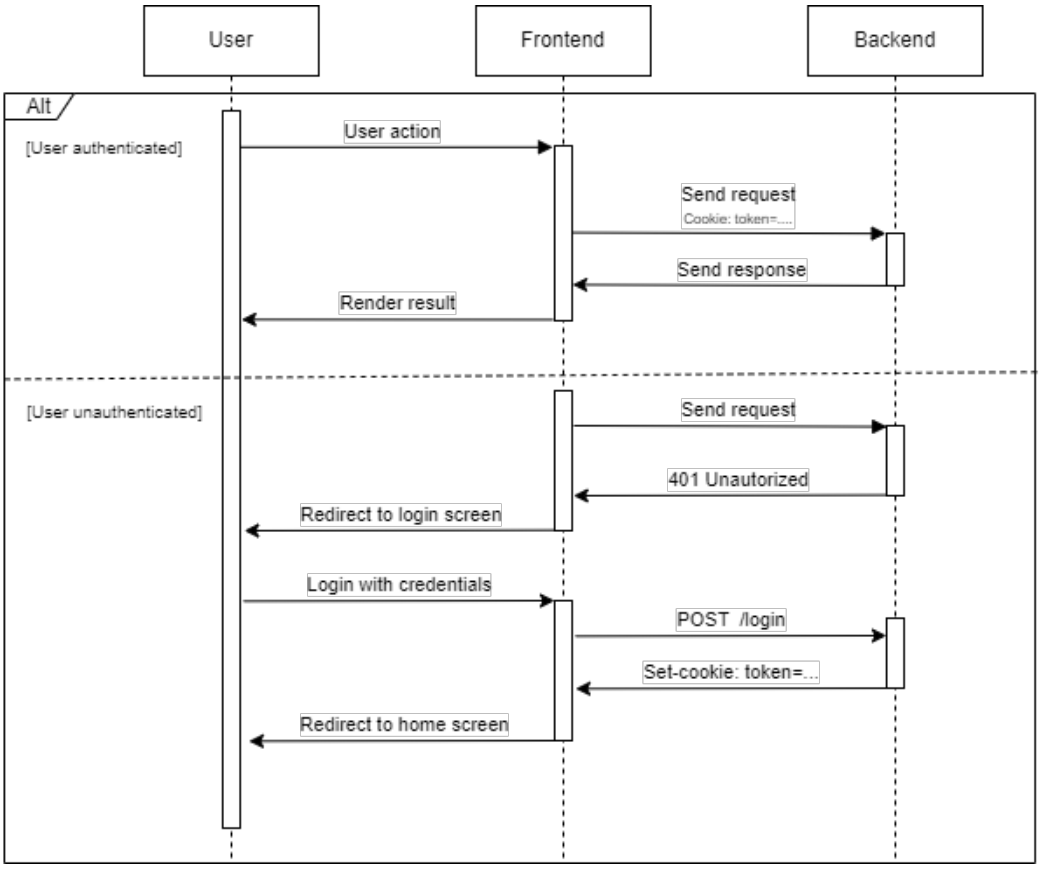
```
type DataLoaderProps<T> = {
  loadingFunction: () => Promise<T>;
  loadingElement?: ReactNode;
  children: (data: T, reload: () => Promise<void>) => ReactNode;
};

export function DataLoader<T>(
  { children, loadingElement, loadingFunction }: DataLoaderProps<T>
) {
  const [data, setData] = useState<T | undefined>(undefined);
  useEffect(() => {
    loadingFunction().then(setData);
  }, [loadingFunction]);

  if (!data) {
    if (loadingElement) {
      return <>{loadingElement}</>;
    } else {
      return (
        <Box display="flex" alignItems="center" justifyContent="center">
          <Spinner />
        </Box>
      );
    }
  } else {
    return <>{children(data, () => loadingFunction().then(setData))}</>;
  }
}
```

5.2.3 Autentizace

Po konzultaci s autorem backendové části byla zvolena strategie autentizace založená na cookies. Tato metoda řízení uživatelské autentizace pro webové aplikace spočívá v tom, že na frontendovou část je vytvořen cookie s hodnotou autentizačního tokenu. Tento token je následně zasílán s každým požadavkem na backend, který ověřuje jeho platnost a přístupová práva uživatele.

5.2.3.1 Autentizace pomocí cookies

Sekvenční diagram 5.2 demonstruje princip zabezpečení endpointů prostřednictvím autentizace založené na cookies. V diagramu jsou prezentovány dva scénáře — první představuje situaci, kdy je uživatel přihlášen, a druhý, kdy přihlášen není. V případě, že uživatel není přihlášen, přístup je backendem odmítnut a je vrácen statusový kód 401. Následně frontend reaguje na tuto situaci tím, že přesměruje uživatele na přihlašovací obrazovku. Na přihlašovací obrazovce zadává uživatel své přístupové údaje a odesláním formuláře je z frontendu na backend odeslán autentizační požadavek. Jsou-li přístupové údaje platné, backend odpovídá nastavením cookies s hodnotou autentizačního tokenu. Tato cookie obsahuje příznak `HttpOnly`, který zajišťuje ochranu hodnoty cookies před neoprávněným přepsáním skrze XSS útoky. Poté jsou s každým následujícím požadavkem z frontendu odesílány i autentizační tokeny v podobě cookies, čímž je ověřována identita uživatelů s každým požadavkem. Díky tomuto způsobu autentizace je zabezpečení aplikace zvýšeno, a uživatelé mají přístup pouze k těm částem aplikace, ke kterým disponují oprávněním.



Obrázek 5.2 Sekvenčí diagram zobrazující autentizaci

5.2.3.2 Axios Interceptors

Axios interceptors představují důležitou součást knihovny axios a hrají klíčovou roli při implementaci globálního ošetření neoprávněného přístupu a následného přesměrování uživatele na přihlašovací obrazovku. Interceptory zjednodušují sledování i logování chyb, manipulaci s hlavičkami požadavků a cachování odpovědí, což je činí ideálním nástrojem pro správu akcí spojených s autentizací a zabezpečením aplikace.

V rámci aplikace axios interceptors jednoduše řeší situace, kdy backend vrátí stavový kód 401, signalizující neoprávněný přístup. Aplikace může v takovém případě automaticky odmítnout požadavek a přesměrovat uživatele na přihlašovací stránku. Ukázka kódu 5.16 ilustruje implementaci řešení s využitím axios interceptors. Jakmile je obdržena odpověď s hodnotou 401, aplikace je přesměrována na stránku určenou pro obsluhu neoprávněného přístupu. V ostatních situacích aplikace navrátí standardní chybu a přenechá zodpovědnost jejího zpracování aplikaci.

Výpis kódu 5.16 Definice middleware kontrolující oprávnění uživatele

```
export function setAxiosInterceptors() {

  axios.interceptors.response.use(
    (response) => response,
    (error) => {
      if (error.response.status === 401) {
        window.location.href = `/${appPaths.unauthenticated}`;
      }
      return error;
    }
  );

}
```

5.2.3.3 Poskytování uživatelských dat

Jak bylo vysvětleno v předchozí části, autentizace se primárně zabývá ověřováním identity uživatele. Kromě toho je důležité věnovat pozornost autorizaci, která určuje, jaká oprávnění má uživatel na základě své role. Za tímto účelem backend poskytuje speciální endpoint pro získání identity uživatele a jeho rolí. Aby bylo možné implementovat autorizační mechanismus, je nejprve nutné, aby frontendová aplikace předávala uživatelská data a oprávnění v rámci celé aplikace.

V ukázce kódu 5.17 je prezentována definice vlastního provideru, který globálně zpřístupňuje uživatelská data a oprávnění v aplikaci. Tento provider vytváří kontextový objekt `AuthContext`, který uchovává aktuálně přihlášený účet a funkci `setAccount` pro aktualizaci účtu. Komponenta `AuthProvider` následně využívá tento kontext pro získání a poskytnutí dat o přihlášeném uživateli. Iniciální hodnota uživatelských providerů je nastavena na `undefined`, což symbolizuje stav nepřihlášeného uživatele.

Uživatelská data jsou nejprve získána prostřednictvím metody pro API volání `getSelf`, která konzumuje endpoint určený pro získání informací o uživatelském účtu, jako jsou `username` a `authorities`, které jsou uloženy jako pole řetězců. Následně je volána metoda `getUserDetails`, která načítá osobní údaje uživatele, jako jsou jméno, příjmení či celé jméno, na základě identifikátoru uživatele. Oba tyto kroky se provádějí v rámci funkce `fetchUserData`, která je spuštěna pomocí hooku `useEffect` při prvním spuštění aplikace.

Stav `isAuthChecked` uchovává informaci o tom, zda již byla kontrola autentizace provedena. Pokud `isAuthChecked` není nastaven na `true`, provider vrátí prázdnou komponentu a aplikace se bude načítat. Jakmile je autentizace zkontrolována, komponenta `AuthProvider` poskytuje kontextový objekt `AuthContext`, který obsahuje data o přihlášeném uživateli a funkci pro aktualizaci těchto dat. Tím jsou v celé aplikaci zpřístupněna uživatelská data a role pro další zpracování.

Výpis kódu 5.17 Definice provideru poskytující data o přihlášeném uživateli

```
export const AuthContext = createContext<AuthContextType>({
  account: undefined,
  setAccount: () => {},
});

type AuthProviderProps = {
  children: ReactNode;
};

export function AuthProvider({ children }: AuthProviderProps) {
  const [isAuthChecked, setIsAuthChecked] = useState(false);
  const [account, setAccount] = useState<Account | undefined>(undefined);

  const fetchUserData = useCallback(async () => {
    if (await isAuthenticated()) {
      const self = await getSelf();
      const user = await getUserDetails(self.id);
      setAccount(Account.fromSelfAndUser(self, user));
    }
    setIsAuthChecked(true);
  }, []);

  useEffect(() => {
    fetchUserData();
  }, [fetchUserData]);

  if (!isAuthChecked) {
    return <></>;
  }

  return <AuthContext.Provider value={{ account, setAccount }}>
           {children}
         </AuthContext.Provider>;
}
```

Pro následné získání uživatelských dat uvnitř komponent byl definován vlastní hook, který lze vidět v kódu 5.18. Tento hook extrahuje data z provideru a vrací objekt, ve kterém jsou vy-extrahovány uživatelská data a definovány další pomocné metody pro práci s přihlášeným účtem.

Výpis kódu 5.18 Definice hook pro získání uživatelských dat

```
export const useAuth = () => {
  const authContext = useContext(AuthContext);

  return {
    account: authContext.account,
    refreshData: authContext.fetchUserData,
    signIn: (account: Account) => authContext.setAccount(account),
    signOut: () => authContext.setAccount(undefined),
  };
};
```

5.2.4 Autorizace

I přesto, že má backend zabezpečené endpointy, je nutné i ve frontendové části řešit přístupy a uživatele pouštět pouze do částí systému, ke kterým mají skutečně oprávnění. Toto je jeden z hlavních požadavků pro správné fungování aplikace, protože zajišťuje bezpečnost a správné řízení přístupů.

Ve frontendové části aplikace jsou v rámci routování implementovány dva typy cest — veřejné a soukromé. Veřejné cesty představují části systému, které nevyžadují žádnou formu autentizace a jsou přístupné i nepřihlášeným uživatelům. Příkladem takové cesty může být přihlašovací obrazovka, která musí být dostupná všem uživatelům. Na druhou stranu, soukromé cesty vyžadují autentizovaného uživatele a případně další speciální role, které omezují přístup pouze na oprávněné osoby. Tato implementace zlepšuje zabezpečení a umožňuje flexibilní řízení přístupů v rámci aplikace, které lze globálně nastavit pro více cest. Definici soukromé cesty lze vidět ve výpisu kódu 5.19. Soukromé cesty jsou implementovány jako obyčejné komponenty, které svým potomkům přidávají autorizační logiku a lze jim specifikovat seznam požadovaných rolí, které musí uživatel mít. Komponenta využívá vlastní hook `useAuth`, který poskytuje načtená uživatelská data. Pokud uživatel nesplňuje požadavky, je přesměrován na stránku s neoprávněným přístupem. V případě, kdy uživatel má dostatečná práva, komponenta vykreslí požadovanou obrazovku.

Výpis kódu 5.19 Definice komponenty pro řízení přístupu

```
PrivateRouteProps = { requiredRoles: string[]; children: ReactNode; };

export function PrivateRoute(
    { children, requiredRoles }: PrivateRouteProps
  ) {
  const auth = useAuth();
  if (auth.account &&
    requiredRoles.every((r) => auth.account!.authorities.includes(r))
  ) {
    return <>{children}</>;
  }
  return <Navigate to={`/${appPaths.unauthorized}`} replace={true} />;
}
```

Privátní cestu lze definovat obalením jednotlivých komponent definovanou komponentou `PrivateRoute` a předáním vyžadovaných rolí. Tímto způsobem lze snadno kontrolovat přístup k různým částem aplikace na základě rolí uživatelů. V ukázce 5.20 lze vidět použití komponenty `PrivateRoute` pro globální řízení přístupu do administrátorské sekce, kde uživatelé musí mít alespoň roli `SOURCE-SSO`. Tento přímočarý způsob řízení přístupu umožňuje snadnou správu a zabezpečení citlivých částí aplikace, které by měly být dostupné pouze pro oprávněné uživatele.

Výpis kódu 5.20 Řízení přístupu do administrační části

```
export const adminRoutes: RouteObject[] = [
  {
    element: <PrivateRoute requiredRoles={['SOURCE-SSO', ...]} />,
    children: [
      {
        path: adminPaths.applications,
        element: <Applications />
        ),
      },
      ...
    ],
  },
];
```

5.2.5 Formulářová data

Při práci s formuláři je nezbytné zajistit adekvátní validaci dat předtím, než jsou odeslána na server. Pro implementaci formulářů v aplikaci byla využita knihovna `react-hook-form` ve spolupráci s knihovnou `JOI` pro validaci dat. Knihovna `react-hook-form` usnadňuje proces zpracování dat z formulářů a jejich validace tím, že poskytuje sadu hooků pro manipulaci s těmito daty. Knihovna `JOI` umožňuje validaci dat v JavaScriptu a TypeScriptu prostřednictvím schémat validace pro různé typy dat. Díky těmto schématům lze jednoduše definovat požadavky na data vstupující do aplikace a zabezpečit, že neplatná data nebudou odeslána na server.

V kódu 5.21 je znázorněna validace nové zprávy odesílané v rámci konverzace. V příkladu je použit hook `useForm` z knihovny `react-hook-form`, který je konfigurován s validátorem `JOI`. Validátor `JOI` je založen na schématu validace, které definuje požadavky na data. V tomto případě spočívá validace v ověření, že položka `content` obsahuje neprázdný řetězec, čímž je zabráněno, že se na backend budou posílat prázdné zprávy v rámci konverzace. Při odesílání formuláře je zavolána funkce `handleSubmit`, která předá data z formuláře do funkce `onSubmit` k případnému dalšímu zpracování. Pokud jsou data validní, budou odeslána na server. V opačném případě odeslání neproběhne a uživatel bude upozorněn na chybu.

Výpis kódu 5.21 Ukázka validace formulářových dat

```
export function MessageList({ conversationId }: MessageListProps) {
  const { handleSubmit, register, reset } = useForm({
    resolver: joiResolver(
      Joi.object({ content: Joi.string().required() })
    ),
  });
  ...
  return (
    <DataLoader loadingFunction={conversationLoader}>
      {(data, reload) => {
        ...
        const onSubmit = async (values: FieldValues) => {
          await postConversationMessage({
            conversationId: conversationId,
            content: values.content,
          });
          reload();
          reset();
        };
        return (
          <Flex h="full" flexDirection="column">
            ...
            <form onSubmit={handleSubmit(onSubmit)}>
              <HStack align="start">
                <Textarea {...register('content')} />
                <Button type="submit" leftIcon={<UilMessage />}>
                  Send
                </Button>
              </HStack>
            </form>
            ...
          </Flex>
        );
      }}
    </DataLoader>
  );
}
```

5.2.6 Export do KOS

Během vypracování této práce bylo zjištěno na základě diskuzí s autorem backendové části, že přímý export dat do systému KOS není možný z důvodu časových omezení spojených s vyřizováním udělení přístupů ze strany univerzity. Vzhledem k této situaci byl zvolen alternativní přístup k exportu dat, který spočívá v exportu do souboru ve formátu CSV. Tento soubor lze následně využít pro nahrání dat o přihláškách do systému KOS. Pro generování souboru ve formátu CSV byla využita knihovna `React CSV`, jež umožňuje mapovat datové atributy objektů do sloupců ve formátu CSV. V ukázce kódu 5.22 je znázorněna konfigurace sloupců pro generování souboru CSV. Názvy sloupců korespondují se schématem tabulky `TPRIHL` z databáze KOS, ve které jsou uloženy data o přihláškách, včetně sloupců pro zápis výsledků z událostí a rozhodnutí o přijetí.

Výpis kódu 5.22 Konfigurace sloupců pro generování CSV

```
export const kosExportHeaders: KosExportHeader[] = [
  { label: 'KODPRIHL', key: 'applicationId' },
  { label: 'PRIJMENI', key: 'lastName' },
  { label: 'JMENO', key: 'firstName' },
  { label: 'EMAIL', key: 'email' },
  { label: 'TELEFON', key: 'phoneNumber' },
  { label: 'FORMA_STUDIA', key: 'studyProgrammeFullName' },
  { label: 'ULICE', key: 'street' },
  { label: 'MISTO', key: 'city' },
  { label: 'PSC', key: 'postalCode' },
  { label: 'ZEME', key: 'country' },
  { label: 'HODN1', key: 'mathTestPoints' },
  { label: 'HODN2', key: 'interviewResults' },
  { label: 'ROZHODNUTI', key: 'admissionStateName' },
  ...
];
```

Pro generování finálního souboru ve formátu CSV je využita komponenta `CSVLink`, kterou poskytuje knihovna `React CSV`. Tato komponenta vyžaduje tři povinné parametry — konfiguraci hlaviček `kosExportHeaders`, název generovaného souboru a seznam přihlášek `applicationList`, z nichž se má vytvořit CSV. Proměnná `applicationList` obsahuje data, která jsou získána z backendu prostřednictvím endpointu pro načítání seznamu přihlášek. Tyto přihlášky zahrnují všechna nezbytná data pro opětovný zápis do systému KOS. Kód 5.23 ilustruje použití komponenty `CSVLink`. Tato knihovna nabízí možnost mapování názvů atributů jednotlivých objektů do příslušných sloupců dle konfigurace z kódu 5.22. Po exportu konkrétních přihlášek frontend navíc odesílá na backend dodatečný požadavek o označení těchto přihlášek jako exportovaných. Tímto způsobem jsou přihlášky uzamčeny a v nich již nelze provádět další zápisové operace. Takové opatření zajišťuje konzistenci dat a zabraňuje potenciálním konfliktům či chybám při následných manipulacích s daty přihlášek v systému KOS.

Výpis kódu 5.23 Komponenta pro generování CSV

```
<CSVLink
  headers={kosExportHeaders}
  data={applicationList}
  filename={`export-${new Date().toISOString()}.csv`}
>
  {button}
</CSVLink>
```

5.3 Nasazení

Po implementaci frontendové části a její napojení na backend je možné aplikaci nasadit na virtuální servery univerzity pro účely testování a ověření funkčnosti. Jak frontendová, tak backendová část byly verzovány pomocí systému Gitlab, který poskytuje širokou škálu nástrojů pro podporu a automatizaci nasazení. V této podkapitole budou rozebrány použité nástroje a jejich konfigurace, které byly využity pro účely nasazení aplikace.

5.3.1 Continuous Integration & Delivery

Continuous Integration a *Continuous Delivery* jsou dvě klíčové komponenty softwarového vývoje, které umožňují snadnější správu verzí a rychlejší nasazení změn. *Continuous Integration* je proces, který zajišťuje pravidelné a časté sloučení změn do hlavní větve projektu. Při každém odeslání změn spouštějí automatizované úkoly, které ověřují správnou funkčnost kódu a jeho kompatibilitu s ostatními částmi aplikace. CI pomáhá rychle identifikovat a opravit problémy, což vede k menšímu počtu chyb ve finálním produktu a urychluje vývoj. Naproti tomu *Continuous Delivery* je další krok v procesu, který se zaměřuje na přípravu pro nasazení změn do produkčního prostředí. [36]

K implementaci CI/CD byl využit modul `Gitlab pipelines`, který poskytuje sadu automatizovaných úkolů. `Gitlab pipelines` umožňuje jednoduché nastavení a konfiguraci těchto úkolů pomocí souboru `.gitlab-ci.yml`. Níže jsou uvedeny jednotlivé úkoly použité v procesu CI/CD:

- `check-version`: Ověřuje, zda je aktuální verze aplikace správně nastavena v konfiguračním souboru `package.json` a zda nebyla již dříve použita. Tento úkol se spouští pouze při vytváření merge requestů, což zajišťuje, aby se aplikace řádně verzovala a zabránilo kolizím mezi verzemi.
- `increment-version`: Manuální úkol, který umožňuje v rozhraní `Gitlab pipelines` nastavit novou verzi aplikace. Při spuštění tohoto úkolu Gitlab automaticky přepíše verzi aplikace v `package.json` a nahraje novou verzi jako další commit, což zjednodušuje řízení verzování.
- `lint`: Kontroluje kvalitu kódu pomocí nástroje ESLint, který analyzuje kód a identifikuje potenciální problémy nebo nesrovnalosti ve stylu kódu. Tímto způsobem je zajištěna konzistentnost a čitelnost kódu napříč projektem.
- `test`: Spouští automatizované testy aplikace, které ověřují správnou funkcionalitu jednotlivých komponent a služeb. Testy pomáhají detekovat chyby a nedostatky v kódu a zvyšují důvěru v kvalitu aplikace.
- `build`: Sestavuje aplikaci do produkční verze, která je optimalizována pro výkon a minimalizaci velikosti. Sestavená aplikace je následně nahrána do `Gitlab container repository`, odkud si lze produkční verzi stáhnout pro nasazení na server.
- `create-tag`: Vytváří Git tag pro aktuální verzi aplikace na základě verze v `package.json`, což umožňuje snadné sledování historie verzí a rychlý přístup k jednotlivým verzím. Tímto způsobem je zajištěno efektivní sledování změn a rychlá navigace mezi verzemi aplikace.

5.3.2 Continuous Deployment

Continuous Deployment je proces, který jde ještě dále než *Continuous Delivery*. Zatímco *Continuous Delivery* zajišťuje, že aplikace je vždy připravena k nasazení, *Continuous Deployment* automatizuje samotné nasazení aplikace na produkční servery bez nutnosti manuálního schválení. Tímto způsobem každá úspěšně otestovaná a schválená změna je automaticky nasazena do

produkčního prostředí, což umožňuje ještě rychlejší a efektivnější uvedení nových funkcí a oprav do produkce. [36]

Pro automatizaci nasazení aplikace byl ve spolupráci s autorem backendové části vytvořen nový Gitlab repozitář. Tento repozitář zahrnuje jak frontend, tak backend části aplikace a obsahuje konfigurace pro společné nasazení na univerzitní server. Repozitář disponuje vlastní konfigurací CI/CD procesu a vlastní docker-compose konfigurací, která zahrnuje klientskou část a všechny serverové mikroslužby.

Navíc byl na školních serverech zřízen dočasný virtuální server pro demonstrační účely. Jedná se o linuxový virtuální stroj s konfigurací 2 CPU, 8 GB RAM a 60 GB kapacitou disku. Tato konfigurace také podporuje virtualizaci, což umožňuje spouštět Docker kontejnery nahrané na `Gitlab container repository`. Při nasazení aplikace na tento virtuální server jsou kontejnery automaticky spuštěny a aplikace je připravena k použití. Pro implementaci automatického nasazení je opět využit modul `Gitlab pipelines`. Konfigurace probíhá v následujících krocích:

1. Pipeline je nutné manuálně spustit a specifikovat verze frontendové a backendové části, které se mají nahrát.

2. Během procesu se vygenerují bash skripty, které stahují specifikované verze kontejnerů pomocí příkazů `docker pull`.

3. Poté se vytvoří SSH spojení s virtuálním serverem, a pomocí nástroje SCP se vygenerované skripty přenesou na server.

4. Na serveru se tyto skripty spustí, což zajistí stažení požadovaných verzí kontejnerů z Gitlab container repository.

5. Po nahrání požadovaných verzí kontejnerů se spustí docker-compose, který aplikaci vystaví na portu 8080, čímž je aplikace připravena k použití.

5.3.3 Výsledná aplikace

Výsledná aplikace byla úspěšně napojena na backendovou část, a většina funkcionalit byla implementována, s výjimkou několika málo prioritních požadavků. Z časových a technických důvodů nebyly následující funkční požadavky začleněny do aplikace:

- `FP15 - Nice to have:` Nahrávání profilových obrázků a příloh
- `FP43 - Nice to have:`Nastavení automatického importu z frontendové části
- `FP44 - Nice to have:` Import výsledků ze systému Moodle
- `FP46 - Should have:` Přímý export do systému KOS

Aplikace byla během psaní této práce nasazena na univerzitních serverech a byla přístupná všem členům akademické obce ČVUT. Nasazená aplikace bude sloužit pro další uživatelské testování a prezentace zúčastněným stranám. Video demonstrace funkčního prototypu aplikace, který je napojen na backend, lze najít v přiloženém médiu této práce. Tato video demonstrace poskytuje ucelený přehled o funkčnosti aplikace a jejích možnostech.

5.4 Shrnutí

V této kapitole byly představeny klíčové aspekty implementace klientské části, které vycházely z návrhu. Byly diskutovány koncepty od konfigurace sestavovacího nástroje přes definice a stylování komponent až po kontejnerizaci pro lokální a produkční vývoj.

Následně byly se autorem backendové části domluveny schůzky, během kterých byl autor této práce seznámen s technickými záležitostmi backendové části. Poté proběhlo samotné napojení na backendovou část, kde bylo nutné implementovat například mechanismus komunikace s backendovou částí, autentizaci či autorizaci.

V závěru byla řešena automatizace vydávání a nasazení aplikace na univerzitní virtuální servery s využitím Gitlab pipelines. Výsledná aplikace byla implementována s výjimkou několika méně prioritních požadavků.

Kapitola 6
Testování

Testování je důležitým aspektem softwarového inženýrství, který pomáhá identifikovat a opravovat chyby, zajišťovat funkčnost a budovat důvěru mezi zúčastněnými stranami. První část této kapitoly se zabývá různými metodami statické analýzy kódu. Poté jsou popsány automatizované testy a v závěru kapitoly bude probránu uživatelské testování.

6.1 Statická analýza kódu

Statická analýza zdrojového kódu představuje proces prozkoumávání kódu programu bez jeho spuštění za účelem identifikace potenciálních problémů, chyb či nedodržení doporučených postupů. Tento proces zahrnuje kontrolu syntaxe, stylu, detekci nebezpečných konstrukcí a vyhledávání známých zranitelností. Statická analýza kódu může významně zlepšit jeho kvalitu, zjednodušit údržbu a snížit náklady na vývoj tím, že umožňuje odhalit a opravit chyby v již raných fázích vývojového cyklu. [37]

6.1.1 Formatter

Formátovač kódu představuje nástroj, který automaticky upravuje zdrojový kód dle předem stanovených pravidel a konvencí s cílem dosáhnout čitelnějšího a konzistentního kódu. Jeho účelem je zvýšit srozumitelnost kódu a usnadnit spolupráci mezi vývojáři. V rámci této práce byl použit formátovač kódu Prettier, jenž umožňuje nastavení stylu formátování kódu prostřednictvím konfiguračního souboru `.prettierrc`.

Výpis kódu 6.1 Konfigurace formátovače kódu pro vlastní projekt

```
// .prettierrc.json

{
  "semi": true,
  "tabWidth": 2,
  "printWidth": 100,
  "singleQuote": true,
  "trailingComma": "all",
  "jsxSingleQuote": true,
  "bracketSpacing": true
}
```

Nastavení uvedené v ukázce 6.1 obsahuje několik definicí, jež mají následující význam:

- `semi`: Toto pravidlo zajistí, že každý příkaz bude ukončen středníkem.
- `tabWidth`: Nastavením šířky zarážky na 2 mezery zajišťuje, že odsazení bude konzistentní a kód bude kompaktnější, což usnadňuje jeho čtení.
- `printWidth`: Toto pravidlo omezuje maximální šířku řádku na 100 znaků, což zabraňuje příliš dlouhým řádkům a zlepšuje čitelnost kódu.
- `singleQuote`: Nastavením použití jednoduchých uvozovek místo dvojitých uvozovek zvyšuje konzistenci a zjednodušuje zápis řetězců.
- `trailingComma`: Toto pravidlo vyžaduje, aby všechny položky v seznamu nebo objektu byly ukončeny čárkou, což usnadňuje přidávání nových položek a zlepšuje čitelnost kódu.
- `jsxSingleQuote`: Stejně jako pravidlo `singleQuote`, ale specificky pro JSX syntaxi, což zajišťuje konzistenci mezi běžným JavaScriptem a JSX.
- `bracketSpacing`: Toto pravidlo zajišťuje, že mezi závorkami objektů bude mezera, což zlepšuje čitelnost a vizuálně odděluje prvky v objektu.

Použití formátovače kódu Prettier ve spojení s výše uvedenou konfigurací zajišťuje, že kód je dobře strukturovaný, čitelný a konzistentní napříč celým projektem, což snižuje čas strávený manuálním formátováním kódu nebo opravováním stylu při code review. Pro vývoj aplikace autor práce použil IDE WebStorm od společnosti Jetbrains, do které lze integrovat knihovnu Prettier. Díky tomu byl formátovač kódu integrován přímo do vývojového prostředí, díky čemuž byla kvalita kódu kontinuálně ověřována po celou dobu vývoje při psaní kódu.

6.1.2 Linter

Lintování a formátování jsou dva odlišné procesy zaměřené na zlepšení kvality a konzistence kódu, ale každý s jiným účelem. Lintování se soustředí na prohledávání kódu, aby odhalilo možné problémy, chyby nebo nesrovnalosti vzhledem k doporučeným postupům a konvencím. Jeho hlavním cílem je pomoci vývojářům identifikovat a řešit problémy v kódu, ještě než se stanou vážnými komplikacemi. Naopak formátování se věnuje upravování zdrojového kódu podle stanovených pravidel a konvencí, což zvyšuje jeho čitelnost a jednotnost kódu napříč celým projektem. V projektech se často spolupracuje s oběma typy nástrojů, aby se dosáhlo optimálního výsledku a zajištěno, že kód je jak kvalitní, tak konzistentní. V tomto projektu byl použit linter ESLint, který je široce používaný a vysoce konfigurovatelný pro JavaScript a TypeScript projekty. ESLint umožňuje nastavit pravidla, která nejlépe vyhovují konkrétním potřebám projektu a dodržování doporučených postupů vývoje. Výhodou ESLintu je jeho kompatibilita s konfigurací formátovače kódu Prettier. Tato integrace umožňuje kombinaci obou nástrojů pro analýzu a úpravu kódu, což zajišťuje maximální efektivitu a kvalitu výsledného kódu. Jak již bylo vysvětleno v sekci 5.3.1 o nasazení, linter se také spouští v rámci CI/CD procesu při integraci nových změn do kódu.

Ve výpisu kódu 6.2 je zobrazena konfigurace lintování kódu pomocí ESLint, která je navržena tak, aby byla kompatibilní s formátováním kódu prostřednictvím Prettieru. ESLint konfigurace se skládá z několika klíčových částí, které slouží k zajištění správného a efektivního lintování. V části `extends` je uveden `plugin:prettier/recommended`, což znamená, že ESLint bude používat doporučená pravidla knihovny Prettier při lintování kódu. Tímto způsobem se zajistí, že Prettier a ESLint budou pracovat společně a nebudou si navzájem odporovat. Dále je v části `plugins` uveden prettier, což znamená, že ESLint bude používat Prettier jako plugin pro lintování. V části `rules` je pak definována řada pravidel, která se použijí při lintování a formátování kódu. Tato pravidla zahrnují například nastavení stylu zalomení řádků, vypnutí některých kontrol pro React, nastavení řazení importů, nebo specifická pravidla pro TypeScript. Zároveň je zde uvedeno pravidlo `prettier/prettier` s konfigurací pro Prettier, které říká, že se má použít lokální konfigurace `.prettierc`.

Výpis kódu 6.2 Konfigurace lintování kódu pro vlastní projekt

```
// .eslintrc.json
{
  ...
    "extends": [
    ...
    "plugin:prettier/recommended",
    ...
  ],
  "plugins": [
    ...
    "prettier",
    ...
  ],
  "rules": {
    "linebreak-style": ["error", "unix"],
    "react/prop-types": "off",
    "import/order": [
      "error",
      {
        "groups": [ "builtin", "external", "internal",
          "parent", "sibling", "index", "object" ],
        "newlines-between": "always",
        "alphabetize": { "order": "asc", "caseInsensitive": true }
      }
    ],
    "import/default": "off",
    "import/no-named-as-default-member": "off",
    "import/no-named-as-default": "off",
    "react/react-in-jsx-scope": "off",
    "jsx-a11y/anchor-is-valid": "off",
    "@typescript-eslint/no-unused-vars": ["error"],
    "@typescript-eslint/explicit-function-return-type": ["off"],
    "@typescript-eslint/explicit-module-boundary-types": ["off"],
    "@typescript-eslint/no-empty-function": ["off"],
    "@typescript-eslint/no-explicit-any": ["off"],
    "@typescript-eslint/no-non-null-assertion": ["off"],
    "prettier/prettier": [ "error", {},
      { "usePrettierrc": true, "endOfLine:": "auto" } ]
  },
  ...
}
```

6.2 Automatizované testy

Automatizované testování je zásadní nástroj ve vývoji softwaru, který umožňuje vytvářet testovací sady pro odhalení chyb a zajištění, že nové úpravy nezpůsobí regrese. Opakovatelnost těchto testů zaručuje konzistentní ověřování správného chování softwaru. V rámci tohoto projektu byly automatizované testy zavedeny hlavně ve formě unit testů, zkoumajících jednotlivé funkční části aplikační logiky. Testování celkové komunikace mezi aplikacemi a většími částmi systému je ponecháno na následujících uživatelských testech.

Jest, testovací framework pro JavaScript aplikace včetně Reactu, byl použit pro implementaci unit testů. Jest nabízí snadno použitelnou a efektivní testovací platformu, umožňující vytváření široké škály testů s podporou funkcí jako *mocking*, analýza pokrytí kódu a *snapshot testing*. Díky tomu je Jest ideální volbou pro testování React komponent a aplikací. Vysoce konfigurovatelný Jest umožňuje přizpůsobení testovacího prostředí dle specifických požadavků a nabízí možnost automatizované testy v budoucnu rozšířit i na integrační a end-to-end testy. V rámci unit testů byly otestovány veškeré pomocné funkce, implementace vlastních hooks a klíčové komponenty. Unit testy jsou dále strukturované podle vzoru AAA, který definuje následující strukturu:

1. `Arrange:` Před samotným testem se nastaví všechny potřebné proměnné, objekty a závislosti. Tento krok zahrnuje inicializaci testovacího prostředí, vytvoření mocků a nastavení počátečních hodnot.
2. `Act:` V tomto kroku se provede testovaná akce nebo funkce, což může zahrnovat volání funkce nebo komponenty, kterou chceme otestovat.
3. `Assert:` Nakonec je ověřeno, zda testovaná funkce nebo komponenta splňuje očekávané výsledky. To zahrnuje porovnání výstupu s očekávanou hodnotou nebo kontrolu, zda byly volány správné metody či zda došlo ke správným změnám v DOM. [38]

V níže uvedené ukázce 6.3 lze vidět implementaci unit testu pro ověření korektnosti vlastního hooku `useInterval`. Hook `useInterval` je navržen pro opakované volání zadané callback funkce v pravidelných časových intervalech. Hlavním úkolem tohoto testu je ověřit, že callback funkce je skutečně volána v požadovaném intervalu. V části Arrange je vytvořena simulovaná callback funkce pomocí metody `jest.fn` a definována proměnná perioda s hodnotou 1000 milisekund. V části Act je hook `useInterval` renderován s použitím funkce `renderHook` a s předáním simulované callback funkce a periody. Poté je čas v Jest simulovaně posunut o 5 násobek periody pomocí metody `jest.advanceTimersByTime`. V části Assert je pomocí metody `toHaveBeenCalledTimes` ověřeno, že simulovaná callback funkce byla zavolána přesně 5 krát, což odpovídá počtu očekávaných volání v průběhu 5 násobku periody.

Výpis kódu 6.3 Ukázka testování vlastních hooks

```
describe('useInterval', () => {
  ...
  it('should call the callback at the specified interval', () => {
    // Arrange
    const callback = jest.fn();
    const period = 1000;
    // Act
    renderHook(() => useInterval(callback, period));
    jest.advanceTimersByTime(period * 5);
    // Assert
    expect(callback).toHaveBeenCalledTimes(5);
  });
  ...
});
}
```

Dalším využitím unit testů je pro validace správného vykreslení komponent. V příkladu 6.4 je ukázka testování `PrivateRoute komponenty`, která byla použita pro implementaci autorizace a podmíněně vykresluje předaný obsah na základě uživatelské role. Tento test je zaměřen na ověření, že pokud uživatel má požadované oprávnění, komponenta vykreslí své potomky. V této části testu je vytvářena hodnota kontextu autentizace, která obsahuje mock uživatelského účtu s požadovanými oprávněními. Dále je definována komponenta `TestComponent`, která slouží jako zástupný potomek komponenty `PrivateRoute`. Pro vykreslení komponenty `PrivateRoute` je použita funkce `render`. Komponenta je obalena `MemoryRouter`, který slouží k simulaci chování React Routeru, a `AuthContext.Provider`, který zajišťuje předání hodnoty kontextu autentizace do komponenty `PrivateRoute`. V poslední části testu je ověřeno, že `TestComponent` byla skutečně vykreslena. Pro ověření přítomnosti `TestComponent` v DOM se využívá funkce `screen.getByText`, která vyhledává element podle zadaného textu. V závěrečné fázi se očekává, že `TestComponent` bude přítomna v dokumentu, což znamená, že komponenta `PrivateRoute` funguje správně a vykresluje své potomky v případě splnění požadovaných rolí.

Výpis kódu 6.4 Ukázka testování komponent

```
const TestComponent = () => <div>Test Component</div>;

describe('PrivateRoute component', () => {

  ...

  test('renders children if user has required roles', () => {
    // Arrange
    const authContextValue = { account: userAccountMock };

    // Act
    render(
      <MemoryRouter>
        <AuthContext.Provider value={authContextValue}>
          <PrivateRoute requiredRoles={userAccountMock.authorities}>
            <TestComponent />
          </PrivateRoute>
        </AuthContext.Provider>
      </MemoryRouter>,
    );

    // Assert
    expect(screen.getByText('Test Component')).toBeInTheDocument();
  });

  ...

});
```

Důvodem, proč nebyly použity integrační nebo end-to-end testy, je zvážení nákladů a času potřebného pro jejich implementaci a údržbu. Práce se zaměřuje na rychlý vývoj a spoléhá na jednotkové testy pro ověření správného chování jednotlivých částí aplikace. Integrační a end-to-end testy by přinesly větší komplexnost do testovací sady a zvýšily nároky na prostředky potřebné pro jejich provádění. Navíc, integrační a end-to-end testy často vyžadují více úsilí na údržbu, protože mohou být citlivé na změny v aplikaci a vyžadují pravidelné aktualizace testovacích scénářů. [39] Práce se místo toho soustředila na uživatelské testování, které poskytuje důležitá zjištění o celkové funkčnosti aplikace a její použitelnosti. Uživatelské testování také umožňuje získat zpětnou vazbu od skutečných uživatelů, což je cenný zdroj informací pro identifikaci problémů a možností pro zlepšení.

6.3 Uživatelské testování

Uživatelské testování je klíčovou součástí vývoje softwaru, která se zaměřuje na získání zpětné vazby od skutečných uživatelů a odhalení možných problémů, které by mohly narušit jejich zkušenosti s aplikací. Při uživatelských testech se uživatelům představuje aplikace a sleduje se, jak ji používají, což umožňuje identifikovat nedostatky v návrhu, funkčnosti nebo použitelnosti aplikace.

6.3.1 Screening

Screening, neboli výběr účastníků, je důležitou součástí procesu uživatelského testování. Jeho úspěch závisí na kvalitě a reprezentativnosti účastníků. Je nezbytné získat skupinu účastníků, která odpovídá cílové skupině aplikace, aby se zajistilo, že získaná zpětná vazba bude relevantní a použitelná pro vývoj. Při výběru účastníků je důležité najít rovnováhu mezi různorodostí a homogenitou, aby se zohlednily různé perspektivy a zároveň se zajistila soudržnost skupiny. [40] V kontextu přijímacího řízení by měla být cílová skupina rozdělena do dvou kategorií — uchazeči a administrátoři.

Kandidáti pro testování uživatelského rozhraní by měli být budoucí či aktuální studenti vysokých škol, kteří mají základní zkušenosti s používáním webových aplikací. Měli by být schopni efektivně komunikovat své potřeby a názory a mít základní znalosti o procesu přijímacího řízení. Tito kandidáti by měli být schopni adaptovat se na nové technologie a zároveň mít schopnost vyjadřovat své myšlenky a připomínky.

Naproti tomu kandidáti pro testování administrátorského rozhraní by měli být pracovníci v oblasti administrativy, znalost domény přijímacího řízení je výhodou, však není nutnou podmínkou. Měli by mít zkušenosti s práci v administrativních rolích, být schopni efektivně řešit problémy, které se mohou vyskytnout během procesu, a být schopni komunikovat s ostatními účastníky procesu. Administrátoři by měli mít silné organizační dovednosti, aby dokázali spravovat a sledovat celý proces.

Aby byl zajištěn vhodný výběr kandidátů, byl vytvořen screeningový pohovor, který obsahuje následující strukturu:

1. `Obecné otázky:` Cílem těchto otázek je získat demografické informace o účastnících, jako je věk, pohlaví, vzdělání a zkušenosti s podobnými aplikacemi nebo procesy, aby byl zajištěn různorodý a reprezentativní vzorek účastníků.

2. `Profese:` Tyto otázky pomáhají identifikovat současné zaměstnání nebo roli účastníka, což může poskytnout náhled na jeho potenciální znalost domény a příslušných úkolů v rámci přijímacího procesu.

3. `Znalost práce s počítačem:` Tyto otázky hodnotí pohodlí a zkušenosti účastníka s používáním webových aplikací, online platforem nebo jiných relevantních technologií, což zajišťuje, že se účastník dokáže v aplikaci efektivně pohybovat a pracovat s ní.

4. `Znalost domény:` Tyto otázky hodnotí, zda je účastník obeznámen s typickými kroky přijímacího řízení na vysoké školy nebo s podobnými procesy. Tyto znalosti jsou důležité pro pochopení a poskytnutí smysluplné zpětné vazby k funkcím webové aplikace.

Na základě výše uvedené struktury byly připraveny screeningové otázky, které pomohly při výběru kandidátů. Kompletní výpis otázek lze najít v příloze E.1. Na základě screeningového pohovoru bylo zvoleno celkem 5 účastníků — 2 administrátoři a 3 uchazeči. Popis jednotlivých kandidátů lze najít v příloze E.2, přičemž jména účastníků byly z důvodů ochrany identity anonymizována.

6.3.2 Prostředí

Prostředí, ve kterém se uživatelské testování provádí, má významný vliv na kvalitu získaných výsledků. Testovací prostředí by mělo být co nejvíce přizpůsobeno reálným podmínkám, ve kterých budou uživatelé aplikaci používat. To zahrnuje fyzické prostředí, zařízení, operační systémy a prohlížeče. Při testování je důležité zajistit, aby se účastníci cítili pohodlně a nebyli omezováni vnějšími faktory, jako je hluk, nedostatek času nebo stres. Tím se dosáhne přirozenějšího chování účastníků a získají se relevantnější výsledky. [40]

Pro účely uživatelské testování bylo zvoleno klidné a pohodlné prostředí, které simulovalo reálné podmínky používání aplikace. Testování se uskutečnilo v zasedací místnosti, která byla vybavena stoly a židlemi pro účastníky a moderátora, a dostatečně osvětlená pro pohodlnou práci. Každý účastník měl k dispozici notebook s předem nainstalovanou webovou aplikací a přístupem k internetu.

Během uživatelského testování byl přítomen moderátor, který se staral o nastavení prostředí pro každého účastníka zvlášť. Účastníkům byly poskytnuty veškeré potřebné informace o testování v podobě testovacích scénářů. Dále byl účastníkům podrobně vysvětlen celý přijímací proces, který aplikace pokrývá, aby měli účastníci plně pochopení kontextu a mohli tak lépe hodnotit jednotlivé funkce aplikace. Během testování byl moderátor přítomen v místnosti a byl připraven poskytnout pomoc nebo objasnit jakékoliv nejasnosti, pokud bylo třeba. Moderátor také sledoval účastníky, pozoroval jejich interakci s aplikací a zaznamenával poznámky o jejich chování, obtížích či úspěších při plnění zadaných úkolů.

Testování bylo navrženo tak, aby bylo co nejvíce formální a profesionální, a zároveň aby poskytovalo dostatek prostoru pro uvolněnou a přirozenou komunikaci mezi účastníky a moderátorem. Celkově bylo prostředí a průběh testování navržen tak, aby podporoval efektivní sběr relevantních dat a zajišťoval kvalitní zpětnou vazbu pro zlepšení webové aplikace.

6.3.3 Testovací scénáře

Testovací scénáře představují klíčovou součást uživatelského testování, jelikož definují konkrétní úkoly, které mají účastníci splnit, a pomáhají tak identifikovat problémy v aplikaci. Scénáře by měly být realistické, smysluplné a přiměřeně náročné, aby účastníci používali aplikaci tak, jak by to dělali v reálných podmínkách. Navíc by scénáře měly pokrýt všechny klíčové funkce aplikace, aby byla zajištěna jejich důkladná validace. Pro účely tohoto uživatelského testování byly vytvořeny scénáře, které pokrývají hlavní oblasti funkcí aplikace. Scénáře byly rozděleny do dvou skupin — scénáře pro uchazeče a scénáře pro administrátory. Každý scénář byl navržen tak, aby účastník provedl jednu nebo více klíčových funkcí aplikace a poskytl zpětnou vazbu na jejich použitelnost, efektivnost a spolehlivost. [41]

Testovací scénáře vycházejí z velké části ze scénářů případu užití, které byly vytvořeny v rámci analýzy analýzy požadavků v sekci 2.4.2. Kompletní seznam všech testovacích scénářů lze nalézt v příloze E.3. Struktura uvedených scénářů je následující:

1. `Identifikátor:` Jedinečný identifikační kód nebo číslo, který slouží k jednoznačnému označení konkrétního scénáře. Tento identifikátor usnadňuje komunikaci a organizaci scénářů během testování a při analýze výsledků.

2. `Název:` Stručný a výstižný název scénáře, který popisuje hlavní úkol nebo situaci, kterou účastník bude řešit. Název by měl být srozumitelný a informativní pro účastníky i pro moderátora.

3. `Předpoklady:` Výčet podmínek, které musí být splněny před zahájením testovacího scénáře. Předpoklady mohou zahrnovat například předchozí akce, které účastník musel provést, nebo specifické nastavení aplikace, které je nutné pro úspěšné dokončení scénáře.

4. `Kontext:` Krátký popis situace nebo scénáře, ve kterém se účastník ocitá. Kontext poskytuje účastníkovi informace o tom, jaký je účel úkolu a jakým způsobem by měl být úkol splněn. Kontext také může obsahovat informace o očekávaných výsledcích nebo chování aplikace.

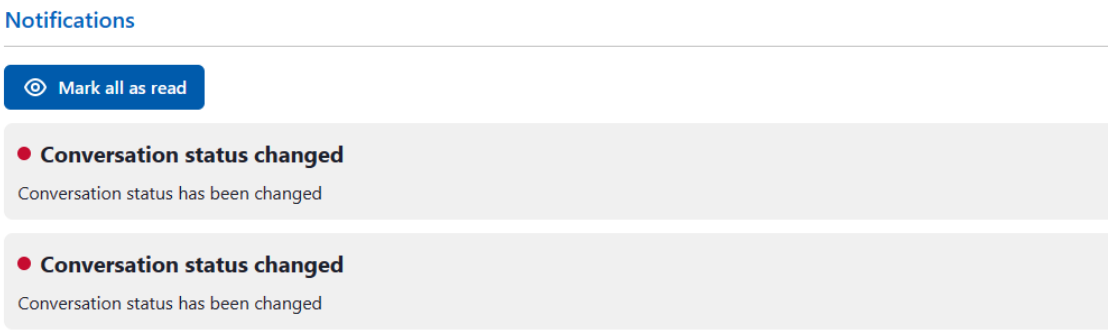
5. `Úspěch:` Kritéria, která musí být splněna, aby byl scénář úspěšně dokončen. Kritéria úspěchu mohou zahrnovat správné vykonání akcí, dosažení očekávaných výsledků nebo splnění jiných požadavků definovaných v scénáři.

6. `Postup:` Sekvenční seznam kroků, které účastník musí provést, aby splnil úkol a dosáhl úspěchu ve scénáři. Kroky by měly být jasně popsány a jednoznačné, aby účastník přesně věděl, co se od něj očekává. Postup by měl být navržen tak, aby účastník prošel všemi klíčovými funkcemi aplikace relevantními pro daný scénář.

6.3.4 Výsledky testování

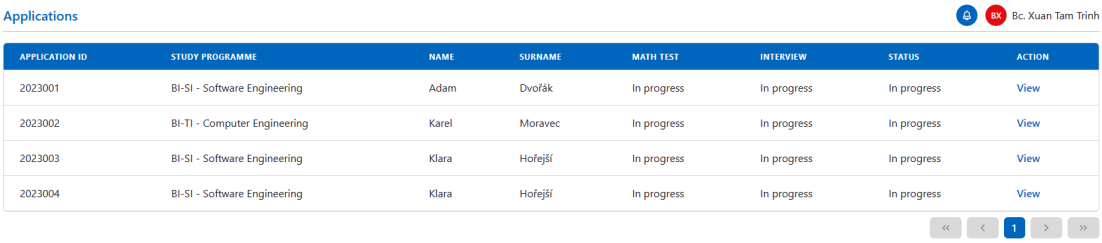
Uživatelské testování bylo provedeno ve dvou iteracích — nejprve byla testována uživatelská část a následně administrátorská část v odlišných termínech. Testovací proces byl pečlivě naplánován, začínal úvodním seznámením účastníků s aplikací a doménou přijímacího řízení. Díky tomu účastníci získali dostatečnou představu o účelu a funkcích aplikace, což zvýšilo hodnotu jejich zpětné vazby. Poté následovalo samotné testování na základě předem připravených testovacích scénářů, které byly představeny v předchozí části. Po dokončení testování se konala společná diskuze se všemi účastníky, kde byly prezentovány výsledky testování, probrán průběh a diskutovány možnosti zlepšení aplikace. Tento přístup umožnil získat cennou zpětnou vazbu a identifikovat potenciální problémy či nedostatky.

Během testování použitelnosti se účastníci bez větších potíží zorientovali v systému a navigaci, což svědčí o intuitivním rozhraní aplikace. Uživatelská část byla testována bez významných problémů, a obecně byla zpětná vazba pozitivní. Jediným společným nedostatkem, který účastníci identifikovali, byla nutnost potvrzovat notifikace jednotlivě v případě jejich většího množství. Tento nedostatek mohl vést k frustraci uživatelů, a proto na základě této zpětné vazby byla do aplikace implementována funkce pro hromadné označení notifikací jako přečtené, jak je zobrazeno na obrázku 6.1.



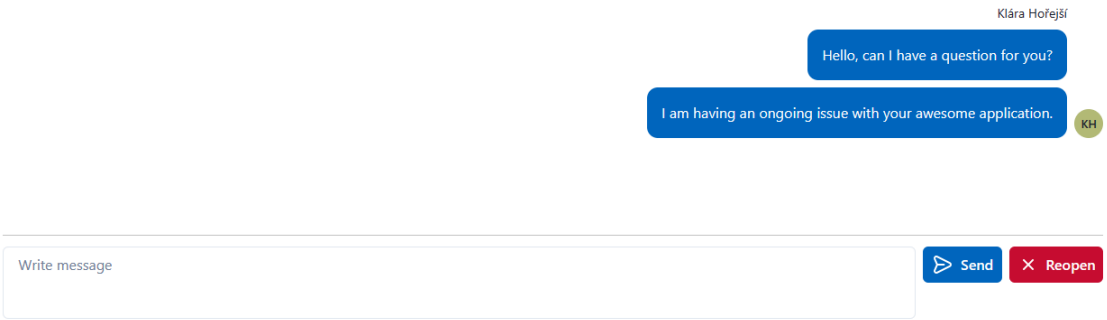
Obrázek 6.1 Přidání hromadného označení notifikací jako přečtené po uživatelském testování

Při testování administrátorské části bylo největší obtíží najít správný záznam v přehledech přihlášek, událostí či aktualit, pokud bylo větší množství dat. Tato zpětná vazba poukázala na potřebu zlepšit efektivitu práce s daty v aplikaci. Na základě této zpětné vazby bylo v aplikaci u všech přehledových tabulek implementováno stránkování, které zobrazí v aplikaci jen určitý počet záznamů, což lze vidět v obrázku 6.2. Stránkování usnadňuje práci s velkým množstvím dat a zvyšuje přehlednost přehledových tabulek, což vede ke zlepšení uživatelské zkušenosti pro administrátory. Dále bylo účastníky navrženo zavést možnost komplexnější filtrace na základě všech údajů v tabulkách, což by umožnilo rychlejší a přesnější vyhledávání potřebných informací. Avšak z časových důvodů nebylo toto vylepšení implementováno, což zůstává otevřenou možností pro budoucí vývoj aplikace.



Obrázek 6.2 Přidání funkcionality stránkování po uživatelském testování

U jednoho účastníka nastal problém, že omylem označil konverzaci jako uzavřenou a následně ji nemohl navrátit do původního stavu. Tato situace poukázala na potřebu zvýšit flexibilitu při práci s konverzacemi a umožnit jejich snadnou úpravu. Z toho důvodu byla do aplikace zavedena možnost znovu otevřít uzavřené konverzace, což lze vidět v obrázku 6.3.



Obrázek 6.3 Přidání možnosti znovuotevření konverzace po uživatelském testování

Po testování a implementaci výše zmíněných úprav již v aplikaci byly provedeny pouze drobné kosmetické úpravy, které zde nemá smysl všechny zmiňovat. Tyto úpravy zahrnovaly například úpravy barev, fontů nebo uspořádání prvků na stránkách, což přispělo ke zlepšení celkového vizuálního dojmu a uživatelského rozhraní aplikace.

Celkově lze říci, že uživatelské testování bylo úspěšné, neboť odhalilo několik oblastí, ve kterých byla aplikace schopna zlepšit svou uživatelskou přívětivost a efektivitu práce. Výsledkem testování je tak aplikace, která lépe vyhovuje potřebám svých uživatelů a nabízí vyšší úroveň spokojenosti při jejím používání. Navíc, díky zpětné vazbě od účastníků testování, bylo možné identifikovat další potenciální oblasti zlepšení, které mohou být v budoucnu implementovány, čímž se aplikace bude nadále rozvíjet a zlepšovat.

6.4 Shrnutí

V průběhu vývoje a testování aplikace bylo použito několik různých metod a technik, které pomohly zajistit stabilitu, bezpečnost a uživatelskou přívětivost aplikace. Během psaní kódu byly průběžně využívány nástroje pro statickou analýzu kódu, jako jsou formátovač a linter. Tyto nástroje umožňovaly identifikovat a opravit chyby v kódu bez jeho spuštění kontinuálně během vývoje.

Další z metod použitých během vývoje bylo psaní a provádění unit testů. Tyto testy při integraci nových změn a funkcí pomohly zabránit regresím, což znamená, že se zabrání opětovnému objevení již opravených chyb nebo způsobení nových problémů v důsledku těchto změn.

Kromě výše zmíněných technik bylo pro otestování systému jako celku provedeno komplexní uživatelské testování. Toto testování bylo prováděno na menším vzorku reprezentativních uživatelů, kteří simulovali skutečné uživatele aplikace. Cílem uživatelského testování bylo odhalit problémy, které by nemusely být zřejmé během jednotkových testů nebo statické analýzy kódu, a získat cennou zpětnou vazbu o tom, jak aplikace funguje z pohledu koncových uživatelů. Uživatelské testování přineslo důležité informace o chování a názorech uživatelů na aplikaci, což umožnilo vývojářům provést potřebné úpravy a zlepšení. Díky tomu bylo zajištěno, že finální verze aplikace bude co nejvíce vyhovovat potřebám a očekáváním jejích uživatelů a zároveň splňovat vysoké standardy kvality.

Kapitola 7

Zhodnocení a možná rozšíření

Tato práce představuje úspěšný vývoj frontendové části aplikace, který byl dosažen díky pečlivé analýze domény, výběru vhodných technologií a promyšlenému návrhu. Během vývoje byly pokryty většina funkčních i nefunkčních požadavků, s výjimkou několika málo prioritních požadavků, které z časových důvodů nebyly zahrnuty, jak již bylo vysvětleno v kapitole o implementaci. Aplikace byla účinně napojena na backendovou část a podrobena důkladnému testování, včetně statické analýzy kódu, automatizovaných jednotkových testů a finálního uživatelského testování, které potvrdilo její celkovou funkčnost a spolehlivost.

V současné době je aplikace funkční prototyp nasazený na univerzitní servery, který je přístupný širokému spektru uživatelů pro testování a ověřování jeho funkčnosti. Prototyp však není přímo napojen na ostrou databázi KOS, ale používá simulovaná data v backendové části aplikace pro efektivní testování a validaci různých funkcí bez zásahu do produkčního prostředí ČVUT. Pro uvedení aplikace do produkce je třeba provést napojení na ostrou databázi ČVUT a řádně aplikaci právně ošetřit. Kromě toho existují další možná rozšíření frontendové části, která by mohla vylepšit celkovou aplikaci.

Prvním zřejmým zlepšením frontendové části je dokončení vynechaných málo prioritních požadavků. Jedná se například o přidání možnosti nahrávání souborů, jako jsou přílohy v konverzacích nebo profilové obrázky uživatelů, které byly vynechány z technických důvodů.

Další možné rozšíření zahrnuje zlepšení responzivity aplikace a rozšíření podpory prohlížečů. Aktuálně je webová aplikace podporována pouze pro desktopové verze prohlížeče Chrome. Zvýšení kompatibility s dalšími prohlížeči a platformami by značně zlepšilo uživatelský zážitek. Toto zlepšení může být v podobě přizpůsobení aplikace pro použití na mobilních zařízeních. To by umožnilo uživatelům snadněji přistupovat k aplikaci a využívat její funkce na cestách nebo mimo klasické pracovní prostředí.

Jedním z dalších možných vylepšení je implementace websocketů. V současné době aplikace periodicky stahuje data o přihláškách a uživatelích pro případné změny stavů v reálném čase. Zavedení websocketů by umožnilo efektivnější a rychlejší aktualizaci dat, což by zlepšilo uživatelský zážitek. Toto vylepšení by vyžadovalo spolupráci s backendovou částí.

Závěr

Cílem této diplomové práce bylo analyzovat, navrhnout a implementovat frontendovou část webové aplikace, která by podporovala procesy přijímacího řízení anglických studijních programů na ČVUT FIT, což by usnadnilo a zefektivnilo práci jak pro uchazeče, tak i pro organizátory.

V první fázi práce proběhla podrobná obchodní analýza interních procesů přijímacího řízení z pohledu uchazečů i organizátorů, která umožnila identifikovat hlavní nedostatky a slabé stránky současného procesu. Následně byla provedena analýza existujícího systému Přiříz, který se používá pro přijímací řízení českých studijních programů. Prozkoumání jeho funkcionalit umožnilo vytvoření základní koncepce výsledného systému pro anglické studijní programy. Na základě zjištěných informací byla vytvořena kompletní specifikace funkčních a nefunkčních požadavků na novou aplikaci, která byla podpořena analýzou případů užití. S využitím případů užití byl vytvořen časový odhad, který posloužil pro řízení vývoje a stanovení časového plánu. Poté následovala komparativní analýza populárních technologií pro vývoj webových klientů, konkrétně Angular, Vue a React, která zahrnovala srovnání různých aspektů těchto technologií. Na základě této analýzy byla knihovna React zvolena jako nejvhodnější pro vývoj frontendové části aplikace. Po technické analýze interní architektury a základních principů knihovny React, které poskytly pevný základ znalostí pro implementaci, proběhlo několik schůzek s autorem backendové části. Během těchto schůzek byla definována systémová architektura aplikace a následně bylo navrženo uživatelské rozhraní v podobě wireframů a následně mockupů, které posloužily jako podklad pro návrh API. Celý návrh byl poté realizován, v práci jsou představeny klíčové části implementace frontendové části, její napojení na backendovou část a nasazení na univerzitní servery. Implementace byla podrobena různým testům, jako je statická analýza, unit testy a uživatelské testy. Tyto testy pomohly ověřit funkčnost a spolehlivost výsledné aplikace. Na základě uživatelských testů byly identifikovány drobné nedostatky v uživatelském rozhraní, které byly následně zahrnuty do finálního řešení.

Na závěr lze prohlásit, že všechny definované hlavní i dílčí cíle této práce byly úspěšně splněny. Tato diplomová práce představuje první krok pro zefektivnění a zjednodušení přijímacího řízení anglických studijních programů na ČVUT FIT.

Příloha A

Specifikace požadavků

A.1 Funkční požadavky

A.1.1 Autentizační část

FP1 - Uživatelské role
Priorita: Must have
FURPS: Funkčnost
Popis: Systém umožní přihlášeným uživatelům přístup k obsahu a funkcím, které odpovídají jejich uživatelské roli, a zamezí přístup k obsahu a funkcím, na které nemají oprávnění.
DoD: Aplikace úspěšně rozlišuje mezi uchazeči a administrátory, přičemž uchazeči mají přístup pouze do uživatelské sekce.

FP2 - Lokální přihlášení
Priorita: Must have
FURPS: Funkčnost
Popis: Uživatelé se budou moci přihlásit pomocí e-mailové adresy zadané v přihlášce a hesla nebo identifikátoru přihlášky. V případě přihlášení pomocí identifikátoru budou přihlášeni do stejného účtu, i když mají více přihlášek.
DoD: Uživatelé úspěšně přihlášeni pomocí e-mailové adresy a hesla.

FP3 - SSO přihlášení
Priorita: Must have
FURPS: Funkčnost
Popis: Uživatelé se budou moci přihlásit prostřednictvím jednotného přihlašovacího systému (SSO) s využitím svých stávajících údajů.
DoD: Uživatelé úspěšně přihlášeni pomocí SSO.

FP4 - Odhlášení
Priorita: Must have
FURPS: Funkčnost
Popis: Uživatelé budou mít možnost odhlásit se ze svého účtu.
DoD: Uživatelé se mohou odhlásit a po obnovení stránky musí být opětovně přihlášeni pro přístup do chráněných částí aplikace.

FP5 - Trvalé přihlášení
Priorita: Nice to have
FURPS: Funkčnost, Použitelnost
Popis: Uživatelé budou moci zvolit možnost trvalého přihlášení, díky které zůstanou přihlášení i po uzavření prohlížeče.
DoD: Pokud se uživatel neodhlásí a zavře prohlížeč, zůstane přihlášený, což zvyšuje použitelnost aplikace.

A.1.2 Uživatelská část

FP6 - Přepínání přihlášek
Priorita: Could have
FURPS: Funkčnost, Použitelnost
Popis: V případě, že mají uživatelé více přihlášek, eviduje se v systému vždy jeden účet společný pro všechny přihlášky. Ke zvýšení komfortu uživatelů, se lze přepínat globálně mezi přihláškami, kdy se uživatelům poté zobrazí obsah specifický ke zvolené přihlášce.
DoD: Uživatel má možnost globálně nastavit, jaké přihlášky se obsah aplikace týká a aplikace na základě toho zobrazí správný obsah.

FP7 - Detail přihlášky
Priorita: Must have
FURPS: Funkčnost
Popis: Uživatel si může zobrazit detailní informace o své přihlášce.
DoD: Aplikace uživateli zobrazí detail přihlášky, kde lze nalézt rozhodnutí o přijetí, informace o událostech, na kterých byl uživatel přihlášen.

FP8 - Zobrazení aktualit
Priorita: Must have
FURPS: Funkčnost
Popis: Uživatel si může procházet aktuality a novinky týkající se přijímacího řízení.
DoD: Aplikace uživateli zobrazí seznam aktualit, které se týkají jeho studijního programu přihlášky.

FP9 - Zobrazení stavu přijímacího řízení
Priorita: Nice to have
FURPS: Funkčnost, použitelnost
Popis: Aplikace uživatele naviguje v celém procesu přijímacího řízení a vidí v jakém stavu se aktuálně nachází. Aplikace sama uživateli nabízí další akční kroky, které musí provést, aby se posunul dál v přijímacím řízení.
DoD: V aplikaci uživatel má k dispozici progress bar, ve kterém jsou zobrazeny všechny splněné i budoucí akce v kontextu přijímacího řízení.

FP10 - Přihlášení na události
Priorita: Must have
FURPS: Funkčnost
Popis: Uživatel se může přihlásit na události spojené s přijímacím řízením.
DoD: Uživateli se zobrazí seznam všech akcí spojené s přijímacím řízením, na které je možné se přihlásit. Uživateli je umožněno přihlásit se pouze na události, které jsou pro jeho studijní program a zároveň jsou otevřené pro zápis. Přihlašování na události musí respektovat proces přijímacího řízení, například na pohovor je možné se pouze hlásit až po absolvování testu z matematiky.

FP11 - Odhlášení z události
Priorita: Must have
FURPS: Funkčnost
Popis: Uživatel se může přihlásit na události spojené s přijímacím řízením.
DoD: Uživateli se může odhlásit z událostí, které ještě neabsolvoval a zároveň ještě mají otevřený zápis.

FP12 - Zobrazení výsledků
Priorita: Must have
FURPS: Funkčnost
Popis: Uživatel si může zobrazit ohodnocení z událostí, kterých se zúčastnil.
DoD: Uživatel vidí počet bodů v případě testu z matematiky a komentáře v případě pohovoru.

FP13 - Konverzace
Priorita: Must have
FURPS: Funkčnost
Popis: Má-li uživatel jakýkoliv dotaz či potřebuje s čímkoliv pomoct, má možnost přímo v aplikaci zaslat administrátorům zprávu.
DoD: Uživatel má možnost zasílat libovolný počet zpráv administrátorům. Po zaslání zprávy se konverzace dostává do stavu otevřená. Uživatel má možnost konverzaci uzavřít, nemá-li další dotazy.

FP14 - Zobrazení uživatelského profilu
Priorita: Must have
FURPS: Funkčnost
Popis: Uživatel si může zobrazit svůj profil, kde nalezne informace ohledně svého účtu.
DoD: Uživateli se zobrazí stránka, kde vidí minimálně své osobní a kontaktní informace.

FP15 - Nahrání profilového obrázku
Priorita: Nice to have
FURPS: Funkčnost, Použitelnost
Popis: Aby aplikace působila osobněji, je u uživatelských jmen zobrazen profilový obrázek.
DoD: Uživatel je schopen nahrát, smazat a zobrazit svůj profilový obrázek a vidět profilové obrázky ostatních uživatelů.

FP16 - Uživatelské notifikace
Priorita: Must have
FURPS: Funkčnost
Popis: Aplikace informuje uživatele o důležitých akcích v podobě notifikací. Uživatel si může zobrazit historii všech původních notifikací a označovat notifikace jako přečtené.
DoD: Uživateli je zobrazen přehled všech notifikací a má možnost označit notifikace jako přečtené.

FP17 - Změna hesla
Priorita: Must have
FURPS: Funkčnost
Popis: V případě-li uživateli nevyhovuje aktuální heslo, může si ho změnit.
DoD: Uživateli je zobrazen formulář, do kterého může alespoň nahrát a ulož své nové heslo.

FP18 - Reset hesla
Priorita: Must have
FURPS: Funkčnost
Popis: V případě-li uživateli zapomněl své heslo, může si ho vyžádat obnovení hesla.
DoD: Uživateli je zobrazen formulář, do kterého zadá svoji emailovou adresu, na kterou je zaslán email s instrukcema ohledně obnovy hesla. Uživatel musí být schopen zjistit své původní heslo či nastavit nové.

FP19 - Získání přístupových údajů do systému Moodle
Priorita: Nice to have
FURPS: Funkčnost
Popis: Po vytvoření přístupových údajů do systému Moodle je uživatel vyzván k vyzkoušení demo testu v systému Moodle, aby si vyzkoušel pracovat v testovacím prostředí.
DoD: Uživateli je poslána emailová zpráva či zobrazena notifikace, s informacemi a výzvě k vyzkoušení demo testu.

FP20 - Výzva k demo testu
Priorita: Nice to have
FURPS: Funkčnost
Popis: Po vytvoření přístupových údajů do systému Moodle je uživatel vyzván k vyzkoušení demo testu v systému Moodle, aby si vyzkoušel pracovat v testovacím prostředí.
DoD: Uživateli je poslána emailová zpráva či zobrazena notifikace, s informacemi a výzvě k vyzkoušení demo testu.

A.1.3 Administrátorská část

FP21 - Přehled přihlášek
Priorita: Must have
FURPS: Funkčnost
Popis: Administrátor vidí přehledovou tabulku, díky které má přehled o stavech všech přihlášek.
DoD: Administrátor vidí tabulku, ve které je minimálně vidět identifikátor přihlášky, studijní program a rozhodnutí o přijetí. Dále se administrátor může prokliknout z tabulky do detailu přihlášky.

FP22 - Detail přihlášky
Priorita: Must have
FURPS: Funkčnost
Popis: Administrátor má možnost si zobrazit detail přihlášky, díky které získá informace spojené s konkrétní přihláškou.
DoD: V detailu přihlášky administrátor vidí, veškeré informace spojené s přihláškou, uživatelským profilem a ohodnocení z událostí.

FP23 - Modifikace přihlášky
Priorita: Must have
FURPS: Funkčnost
Popis: Administrátor může editovat data o přihlášce či z uživatelského profilu.
DoD: Administrátorovi se zobrazí formulář, ve kterém může minimálně měnit ohodnocení z událostí.

FP24 - Přehled událostí
Priorita:
FURPS: Funkčnost
Popis: Administrátor vidí přehledovou tabulku, díky které má přehled všech událostí.
DoD: Administrátor vidí tabulku, ve které je minimálně vidět název události, typ studijního programu, čas konání a kapacity uchazečů. Dále se administrátor může prokliknout z tabulky do detailu události.

FP25 - Detail události
Priorita: Must have
FURPS: Funkčnost
Popis: Administrátor má možnost si zobrazit detail události, díky které získá informace spojené s konkrétní událostí.
DoD: V detailu události administrátor vidí, veškeré informace o názvu události, kapacitách, typu události, časy konání a zápisu a zúčastněné supervizory. Dále může administrátor filtrovat události na základě data otevření zápisu, podle toho jestli je administrátor supervizor dané události a události, které obsahují neobodované uchazeče.

FP26 - Vytvoření události
Priorita: Must have
FURPS: Funkčnost
Popis: Administrátor má možnost vytvořit novou událost. K vytvoření události je nutné zvolit záznam události ze systému KOS a spárovat lokální událost s událostí ze systému KOS.
DoD: Administrátorovi je zobrazen formulář, ve kterém zvolí záznam události ze systému KOS, se kterým se lokální událost páruje.

FP27 - Modifikace události
Priorita: Must have
FURPS: Funkčnost
Popis: Po vytvoření události má administrátor možnost událost modifikovat.
DoD: Administrátorovi vidí stejný formulář jako pro vytvoření události. Při modifikaci již však nelze měnit typ události, mapování na KOS záznam a studijní programy události.

FP28 - Smazání události
Priorita: Must have
FURPS: Funkčnost
Popis: Administrátor může událost po vytvoření smazat.
DoD: Administrátorovi se zobrazí tlačítko pro smazání, dokud událost nemá žádné ohodnocení uchazeč. Po smazání události lze mapovat novou událost na původně zabraný záznam z KOS.

FP29 - Přihlášení na události
Priorita: Must have
FURPS: Funkčnost
Popis: Administrátor se může přihlásit na událost jako supervizor.
DoD: Administrátorovi se zobrazí v přehledu tlačítko pro přihlášení na událost jako supervizor. Administrátor může být přihlášený na více událostí najednou.

FP30 - Odhlášení z události
Priorita: Must have
FURPS: Funkčnost
Popis: Administrátor se může odhlásit z události jako supervizor.
DoD: Administrátorovi je na přihlášených událostech zobrazeno tlačítko pro odhlášení.

FP31 - Upozornění na chybějící supervizi
Priorita: Nice to have
FURPS: Funkčnost
Popis: V případě-li je událost není plná z hlediska kapacity supervizorů, může administrátor požádat ostatní administrátory o přihlášení na událost.
DoD: Administrátorovi je v detailu události zobrazeno tlačítko pro notifikaci administrátorů o nenaplněné kapacitě události. Po kliknutí na tlačítko se administrátorům pošle notifikace či emailová zpráva s výzvou o přihlášení na daný termín.

FP32 - Udělení hodnocení
Priorita: Must have
FURPS: Funkčnost
Popis: Administrátor může udělit hodnocení všem uchazečům, kteří jsou zúčastnění na dané události.
DoD: Administrátor vidí tabulku, ve které může hromadně udělit uchazečům hodnocení za danou událost. V případě testu z matematiky uděluje body a v případě pohovoru zaškrtává políčko, zda-li úspěšně prošli pohovorem a políčko pro případné komentáře.

FP33 - Přehled konverzací
Priorita: Must have
FURPS: Funkčnost
Popis: Administrátor vidí přehledovou tabulku, díky které má přehled o všech konverzacích.
DoD: Administrátor vidí tabulku všech konverzací, ve které lze vidět minimálně jméno uchazeče a status konverzace (otevřená x uzavřená). Administrátor může také filtrovat na základě statusu konverzace či zda-li je sám účastníkem.

FP34 - Detail konverzace
Priorita: Must have
FURPS: Funkčnost
Popis: Administrátor si může zobrazit detail konverzace a odpovědět na zprávy uchazečů.
DoD: Administrátor může vstoupit či začít konverzace a případně měnit status konverzace.

FP35 - Přehled aktualit
Priorita: Must have
FURPS: Funkčnost
Popis: Administrátor vidí přehledovou tabulku, díky které má přehled o všech aktualitách.
DoD: Administrátor vidí tabulku všech aktuality, ve které lze vidět minimálně jméno aktuality, studijní programy, autor a čas, kde je aktualita aktivní. Administrátor může také filtrovat neaktivní aktuality.

FP36 - Modifikace aktuality
Priorita: Must have
FURPS: Funkčnost
Popis: Administrátor si může měnit data aktuality.
DoD: Administrátor vidí všechny informace o vybrané aktualitě.

FP37 - Vytvoření aktuality
Priorita: Must have
FURPS: Funkčnost
Popis: Administrátor má možnost vytvořit novou aktualitu, která bude zobrazena na webu.
DoD: Administrátorovi je zobrazen formulář, ve kterém zadá název aktuality, obsah, datum platnosti a případně přiřadí studijní programy, pro které je aktualita relevantní.

FP38 - Modifikace aktuality
Priorita: Must have
FURPS: Funkčnost
Popis: Administrátor má možnost upravit již existující aktualitu.
DoD: Administrátorovi je zobrazen formulář s aktuálními informacemi o aktualitě, které může upravit. Po uložení změn se aktualita aktualizuje.

FP39 - Smazání aktuality
Priorita: Must have
FURPS: Funkčnost
Popis: Administrátor má možnost smazat aktualitu.
DoD: Administrátorovi se zobrazí tlačítko pro smazání aktuality. Po potvrzení smazání je aktualita odstraněna z webu.

FP40 - Administrátorské notifikace
Priorita: Must have
FURPS: Funkčnost
Popis: Administrátor dostává notifikace o důležitých událostech v systému.
DoD: Administrátorovi se zobrazují notifikace o nových konverzacích, žádostech o supervizi událostí a dalších relevantních událostech.

FP41 - Vytvoření přístupových údajů do systému Moodle
Priorita: Must have
FURPS: Funkčnost
Popis: Administrátor má možnost vytvořit přístupové údaje pro uživatele, aby mohli přistupovat do systému Moodle.
DoD: Administrátorovi je zobrazen formulář, ve kterém zadá informace o uživateli, pro kterého chce vytvořit přístupové údaje. Po uložení jsou přístupové údaje odeslány uživateli emailem.

FP42 - Manuální import dat ze systému KOS
Priorita: Must have
FURPS: Funkčnost
Popis: Administráto může importovat data ze systému KOS do aplikace, a tak aktualizovat data o přihláškách, událostech a studijních programech.
DoD: Administrátor po kliknutí na tlačítko pro manuální import stáhne data z KOS databáze a aktualizuje lokální data aplikace.

FP43 - Automatický import dat ze systému KOS
Priorita: Nice to have
FURPS: Funkčnost
Popis: Implementovat automatický import dat ze systému KOS prostřednictvím API.
DoD: Aplikace periodicky importuje data ze systému KOS pomocí API.

FP44 - Import výsledků ze systému Moodle
Priorita: Nice to have
FURPS: Funkčnost
Popis: Umožnit administrátorům importovat výsledky studentů ze systému Moodle do aplikace.
DoD: Uživatel úspěšně importuje výsledky studentů ze systému Moodle ze souboru.

FP45 - Export do CSV
Priorita: Must have
FURPS: Funkčnost
Popis: Umožnit administrátorům exportovat data do CSV souboru pro další zpracování pro systém KOS.
DoD: Uživatel může vybrat data o přihláškách a exportovat je do CSV souboru.

FP46 - Export do systému KOS
Priorita: Should have
FURPS: Funkčnost
Popis: Implementovat export dat z aplikace zpět do systému KOS prostřednictvím API.
DoD: Uživatel může exportovat data z aplikace do systému KOS pomocí API.

FP47 - Rozhodnutí o ne/přijetí
Priorita: Must have
FURPS: Funkčnost
Popis: Aplikace automaticky rozhoduje o přijetí či nepřijetí studentů na základě zadaných kritérií.
DoD: Aplikace automaticky rozhoduje o přijetí či nepřijetí studentů na základě zadaných kritérií. Přihláška přejde automaticky do stavu nepřijetí, pokud uchazeč nesplní test z matematiky či pohovor. Při úspěšném splnění obou částí, přechází přihláška do stavu přijetí

A.2 Nefunkční požadavky

NP1 - Podpora na prohlížečích Chrome
Priorita: Must have
FURPS: Podpora
Popis: Aplikace lze spustit na prohlížečích Chrome
DoD: Aplikace je podporovaná na prohlížečích Chrome alespoň ve verzi 111.0

NP2 - Rozšiřitelnost a udržitelnost
Priorita: Should have
FURPS: Podpora
Popis: Aplikace bude připravena na další možné rozšíření a je vyvíjena s ohledem na snadnou údržbu.
DoD: Aplikace je navržena s architekturální dekompozicí alespoň do modulů rozdělených na základě funcionalit.

NP3 - Snadnost použití
Priorita: Should have
FURPS: Použitelnost
Popis: Aplikace je navržena s důrazem na snadnost použtití, aby systém byl intuitivní a zároveň nerozptyloval uživatele.
DoD: Rozhraní aplikace je navržena s použitím minimalismu a design je reprezentativní v bar-

vách univerzity.

NP4 - Výkon aplikace
Priorita: Must have
FURPS: Výkon
Popis: Aplikace musí být dostatečně rychlá a odezivní, aby uživatelé nemuseli čekat příliš dlouho na načítání stránek a interakci s aplikací.
DoD: Načítání stránek a interakce s aplikací nezabere více než 3 sekundy, a to i při použití pomalejší sítě.

NP5 - Bezpečnost aplikace
Priorita: Must have
FURPS: Spolehlivost
Popis: Aplikace musí být navržena tak, aby byla bezpečná proti útokům zvenčí a zneužití uživatelských dat.
DoD: Aplikace je chráněna proti SQL injection, CSRF a XSS útokům a všechny uživatelské údaje jsou šifrovány při přenosu přes internet.

NP6 - Přenositelnost aplikace
Priorita: Should have
FURPS: Podpora
Popis: Aplikace musí být navržena tak, aby byla snadno přenositelná na jiné platformy než na kterých je původně vyvíjena.
DoD: Aplikace může být spuštěna na jiných platformách než na kterých byla původně vyvíjena, např. na jiném operačním systému nebo na cloudové platformě.

NP7 - Testování aplikace
Priorita: Should have
FURPS: Spolehlivost
Popis: Aplikace musí být otestována tak, aby byla zajištěna vysoká kvalita a spolehlivost aplikace.
DoD: Aplikace je otestována pomocí automatických testů a manuálních testů, které zahrnují všechny funkce a scénáře uživatelů. Testy jsou součástí vývojového cyklu a jsou pravidelně aktualizovány.

NP8 - Uložení citlivých dat
Priorita: Must have
FURPS: Spolehlivost
Popis: Z bezpečnostních důvodů jsou citlivá data o přihláškách, událostech a studijních programech uložena na straně databáze KOS.
DoD: Data o přihláškách, událostech a studijních programech jsou uložená na v KOS databázi.

Příloha B

Scénáře případů užití

UC1		**Přihlášení do administrátorské sekce**
Související požadavky:		FP1, FP3
Aktéři:		Administrátor
Předpoklady:		Administrátor je odhlášen
		Administrátor má ČVUT identitu
Kontext:		Administrátor se chce přihlásit do svého účtu
Hlavní scénář:	1.	Administrátor spustí aplikaci
	2.	Systém zobrazí administrátorovi přihlašovací formulář
	3.	Administrátor stikne tlačítko přihlásit pomocí SSO
	4.	Systém přesměruje administrátora na SSO přihlašování provozované ČVUT
	5.	Administrátor zadá svoje přihlašovací údaje a potvrdí přihlášení
	6.	Systém přesměruje administrátora do administrátorské sekce
Alternativní scénář:	6a.	Systém zobrazí administrátorovi chybovou hlášku, jelikož zadal chybné údaje
Výsledky:		Administrátor je přihlášený v systému
Omezení:		

UC2		**Přihlášení do uživatelské sekce**
Související požadavky:		FP1, FP2, FP5, FP18
Aktéři:		Uchazeč
Předpoklady:		Uchazeč je odhlášen
		Uchazeč má v systému evidovanou přihlášku a účet
Kontext:		Uchazeč se chce přihlásit do svého účtu
Hlavní scénář:	1.	Uchazeč spustí aplikaci
	2.	Systém zobrazí uchazeči přihlašovací formulář
	3.	Uchazeč vyplní své přihlašovací údaje a potvrdí formulář
	4.	Systém přesměruje uchazeče do uživatelské sekce
Alternativní scénář:	3a.	Uchazeč vyplní přihlašovací údaje a zaškrtne políčko pro trvalé přihlášení
	3b.	
	3b1.	Uchazeč stiskne tlačítko pro obnovu hesla
	3b2.	Systém přesměruje uživatele na formulář pro obnovení hesla
	3b3.	Uchazeč vyplní svoji emailovou adresu a potvrdí formulář
	3b4.	Systém pošle uchazeči email s odkazem pro reset hesla
	3b5.	Uchazeč rozklikne odkaz, zadá nové heslo a potvrdí změny
	4a.	Systém zobrazí uchazeči chybovou hlášku, jelikož zadal chybné údaje
Výsledky:		Uchazeč je přihlášený v systému a vidí domovskou stránku
Omezení:		

UC3		**Odhlášení uživatele**
Související požadavky:		FP4
Aktéři:		Uživatel
Předpoklady:		Uživatel je přihlášen
Kontext:		Uživatel již dokončil svoji práci a chce se odhlásit ze svého účtu
Hlavní scénář:	1.	Uživatel klikne na tlačítko pro odhlášení
	2.	Systém odhlásí uživatele a přesměruje ho následně na přihlašovací obrazovku
Alternativní scénář:		
Výsledky:		Uživatel je odhlášen z aplikace a nachází se v přihlašovací obrazovce
Omezení:		

UC4		**Zobrazení přihlášky**
Související požadavky:		FP6, FP7
Aktéři:		Uchazeč
Předpoklady:		Uchazeč je přihlášen
		Uchazeč má v systému evidovanou nejméně jednu nebo více přihlášek
Kontext:		Uchazeč si chce zobrazit a zkontrolovat údaje o své přihlášce
Hlavní scénář:	1.	Uživatel klikne na detail přihlášky
	2.	Systém přesměruje uchazeče na stránku s informacemi o přihlášce a výsledcích ze všech událostí
Alternativní scénář:	1a.	Má-li uchazeč více přihlášek, zvolí nejprve v nastavení přihlášku o kterou má zájem. Poté uživatel klikne na detail přihlášky
Výsledky:		Uchazeč se nachází v detailu přihlášky a vidí informace a výsledky spojené se svojí přihláškou
Omezení:		

UC5		**Zjištění stavu přijímacího řízení**
Související požadavky:		FP8, FP9
Aktéři:		Uchazeč
Předpoklady:		Uchazeč je přihlášen
Kontext:		Uchazeč chce zjistit relevantní informace o přijímacím řízení
Hlavní scénář:	1.	Uchazeč klikne na tlačítko pro přesměrování do domovské stránky
	2.	Systém zobrazí uchazeči domovskou stránku s grafickým prvek, který ukazuje celkový postupu přihlášky v přijímacím řízení a pod ním všechny aktuality spojené se studijním programem přihlášky uchazeče
Alternativní scénář:		
Výsledky:		Uchazeč se nachází v domovské stránce a vidí na stránce stav přihlášky v přijímacím řízení a případné aktuality
Omezení:		

UC6			**Přihlášení na události**
Související požadavky:			FP10, FP11, FP16
Aktéři:			Uchazeč
Předpoklady:			Uchazeč je přihlášen Stav rozhodnutí o přijetí přihlášky uchazeče není rozhodnutý
Kontext:			Uchazeč se chce zúčstatnit testu z matematiky či pohovoru
Hlavní scénář:	1.		Uchazeč klikne na tlačítko pro zobrazení seznamu událostí
	2.		Systém přesměruje uchazeče na seznam všech událostí spojené s jeho studijním programem
	3.		Uchazeč si zvolí a přihlásí se na jeden termín
	4.		Systém uchazeče přihlásí na termín a pošle uchazeči notifikaci o úspěšném přihlášení na událost
Alternativní scénář:	3a.		
		3a1.	Uchazeč si zvolí omylem špatný termín, a proto po přihlášení, klikne na tlačítko pro odhlášení z události
		3a2.	Systém uchazeče odhlásí z události a pošle uchazeči notifikaci o úspěšném odhlášení
Výsledky:			Uchazeč je přihlášen na zvoleném termínu
Omezení:			Uchazeč nesmí být paralelně na dvě stejné akce zároveň Uchazeč nesmí být paralelně na dvě stejné akce zároveň Uchazeč se může hlásit na pohovor, pouze když úspěšně absolvoval test z matematiky Uchazeč se může hlásit pouze na události otevřené pro zápis

UC7		**Ohodnocení události**
Související požadavky:		FP12, FP16, FP24, FP25, FP29, FP30, FP31, FP32, FP40, FP44, FP47
Aktéři:		Administrátor, uchazeč
Předpoklady:		Uchazeč je přihlášen na událost Administrátor je přihlášen
Kontext:		Po skončení události chce administrátor udělit uchazečům hodnocení
Hlavní scénář:	1.	Administrátor klikne na tlačítko pro zobrazení seznamu událostí
	2.	Systém přesměruje Administrátora na seznam všech událostí
	3.	Administrátor si zvolí a přihlásí se na jeden termín jako supervizor
	4.	Systém Administrátora přihlásí na termín a pošle mu notifikaci o úspěšném přihlášení na událost
	5.	Po proběhnutí události administrátor přejde do seznamu událostí a rozklikne detail události, které se zúčastnil
	6.	Systém Administrátora systém zobrazí uživateli detail události, kde lze vidět seznam všech účastníků. V každém řádku seznamu účastníků je možnost vyplnit výsledek události.
	7.	Administrátor vyplní výsledky pro každého účastníka události a uloží hromadně změny
	8.	Systém uloží změny a pošle všem uchazečům notifikaci o novém hodnocení. Zároveň systém automaticky mění stav o přijetí přihlášky na základě výsledků z událostí.
	9.	Uchazeč na základě notifikace přejde do detail přihlášky, k zobrazení výsledku události
	10.	Systém Zobrazí uchazeči detail přihlášky, kde vidí své hodnocení z události
Alternativní scénář:	3a.	
	3a1.	Administrátor si zvolí omylem špatný termín, a proto po přihlášení, klikne na tlačítko pro odhlášení z události
	3a2.	Systém administrátora odhlásí z události a pošle uchazeči notifikaci o úspěšném odhlášení
Výsledky:		Administrátor je přihlášen na zvoleném termínu jako supervizor Uchazeč vidí své hodnocení v detailu přihlášky Systém automaticky změnil stavy přihlášek na základě výsledků z událostí
Omezení:		

UC8		**Vytvoření konverzace**
Související požadavky:		FP13, FP16, FP33, FP34, FP40
Aktéři:		Administrátor, uchazeč
Předpoklady:		Uchazeč je přihlášen
		Administrátor je přihlášen
Kontext:		Uchazeč potřebuje pomoc od studijního oddělení
Hlavní scénář:	1.	Uchazeč klikne na tlačítko pro vstup do konverzace
	2.	Systém přesměruje uchazeče do detailu konverzace
	3.	Uchazeč sepíše dotaz na studijní oddělení a potvrdí k odeslání
	4.	Systém notifikuje administrátora o novém dotazu
	5.	Administrátor přejde do přehledu konverzací a vybere konverzaci s daným uchazečem
	6.	Systém přesměruje administrátora do detailu konverzace
	7.	Administrátor sepíše odpověď na dotaz uchazeče a potvrdí k odeslání
	8.	Systém notifikuje uchazeče o nové zprávě
	9.	Uchazeč je spokojen s odpovědí a označí konverzaci jako uzavřenou
Alternativní scénář:		
Výsledky:		Byla vytvořena nová konverzace
		Konverzace je ve stavu uzavřená
Omezení:		Každá přihláška má vlastní konverzace, čili při přepnutí přihlášky, dojde i k přepnutí konverzace

UC9			**Správa aktualit**
Související požadavky:			FP35, FP36, FP37, FP38, FP39
Aktéři:			Administrátor
Předpoklady:			Administrátor je přihlášen
Kontext:			Administrátor chce publikovat novou aktualitu ohledně přijímacího řízení
Hlavní scénář:	1.		Administrátor klikne na tlačítko pro vstup do přehledu aktualit
	2.		Systém přesměruje administrátora do přehledu aktualit
	3.		Administrátor klikne na tlačítko pro vytvoření nové aktuality
	4.		Systém zobrazí administrátorovi formulář pro vytvoření aktuality
	5.		Administrátor vyplní všechny povinné políčka a potvrdí vytvoření
	6.		Systém uloží aktualitu a zobrazí ji všem uchazečům, kterých se týká
Alternativní scénář:	6a.		
		6a1.	Systém uloží aktualitu a zobrazí ji všem uchazečům, kterých se týká
		6a2.	Administrátor vyplnil některá políčka špatně, proto se vrátí do detailu aktuality a stiskne tlačítko k modifikaci aktuality
		6a3.	Systém zobrazí administrátorovi formulář pro modifikaci aktuality
		6a4.	Administrátor vyplní všechny povinné políčka a potvrdí změny
		6a5.	Systém uloží změny v aktualitě a zobrazí je všem uchazečům, kterých se týká
	6b.		
		6b1.	Systém uloží aktualitu a zobrazí ji všem uchazečům, kterých se týká
		6b2.	Administrátor zjistil, že aktualita není relevantní a proto se vrátí do detailu aktuality a stiskne tlačítko pro smazání
		6b3.	Systém smaže aktualitu z databáze a uchazečům již nebude aktualitu zobrazovat
Výsledky:			V aplikaci byla vytvořena nová aktualita Aktualitu vidí pouze uchazeči s vybraným studijním programem
Omezení:			Aktualita je viditelná pouze ve zvoleném časovém intervalu

UC10			**Správa událostí**
Související požadavky:			FP26, FP27, FP28, FP31, FP40
Aktéři:			Administrátor
Předpoklady:			Administrátor je přihlášen
Kontext:			Administrátor chce vytvořit novou událost přijímacího řízení
Hlavní scénář:	1.		Administrátor klikne na tlačítko pro vstup do přehledu událostí
	2.		Systém přesměruje administrátora do přehledu událostí
	3.		Administrátor klikne na tlačítko pro vytvoření nové události
	4.		Systém zobrazí administrátorovi formulář pro vytvoření události
	5.		Administrátor vyplní všechny povinné políčka a potvrdí vytvoření
	6.		Systém uloží událost a nabídne ji všem uchazečům, kterých se týká
Alternativní scénář:	6a.		
		6a1.	Systém uloží událost a nabídne ji všem uchazečům, kterých se týká
		6a2.	Administrátor vyplnil některá políčka špatně, proto se vrátí do detailu události a stiskne tlačítko k modifikaci
		6a3.	Systém zobrazí administrátorovi formulář pro modifikaci události
		6a4.	Administrátor vyplní všechny povinné políčka a potvrdí změny
		6a5.	Systém uloží změny v události a zobrazí je všem uchazečům, kterých se týká
	6b.		
		6b1.	Systém uloží událost a nabídne ji všem uchazečům, kterých se týká
		6b2.	Administrátor zjistil, že událost není relevantní a proto se vrátí do detailu události a stiskne tlačítko pro smazání
		6b3.	Systém smaže událost z databáze a uchazečům již nebude událost zobrazovat
Výsledky:			V aplikaci byla vytvořena nová událost
			Události vidí pouze uchazeči s vybraným studijním programem
Omezení:			Událost lze smazat pouze do té doby, kdy nejsou na události evidované žádné hodnocení

UC11		**Správa přihlášek**
Související požadavky:		FP21, FP22, FP23
Aktéři:		Administrátor
Předpoklady:		Administrátor je přihlášen
Kontext:		Administrátor chce zjistit informace spojené s konkrétní přihláškou
Hlavní scénář:	1.	Administrátor klikne na tlačítko pro vstup do přehledu přihlášek
	2.	Systém přesměruje administrátora do přehledu přihlášek
	3.	Administrátor klikne na tlačítko pro vstup do detailu vybrané přihlášky
	4.	Systém přesměruje administrátora do detailu přihlášky, kde vidí veškeré informace spojené s vybranou přihláškou
Alternativní scénář:	4a.	
	4a1.	Systém přesměruje administrátora do detailu přihlášky, kde vidí veškeré informace spojené s vybranou přihláškou
	4a2.	Na základě požadavku od uchazeče administrátor chce změnit některé informace přihlášky, a proto klikne na tlačítko pro modifikaci přihlášky
	4a3.	Systém zobrazí administrátorovi formulář pro modifikaci přihlášky
	4a4.	Administrátor vyplní všechny povinné políčka a potvrdí změny
	4a5.	Systém uloží změny v přihlášce
Výsledky:		Administrátor vidí informace vybrané přihlášky
Omezení:		

UC12		**Správa účtu**
Související požadavky:		FP14, FP15, FP17
Aktéři:		Uchazeč
Předpoklady:		Uchazeč je přihlášen
Kontext:		Uchazeč chce zobrazit upravit nastavení svého účtu
Hlavní scénář:	1.	Uchazeč klikne na tlačítko pro vstup do uživatelského profilu
	2.	Systém přesměruje administrátora do uživatelského profilu a zobrazí mu formulář pro změnu nastavení účtu
	3.	Uchazeč vidí informace o svém účtu, vyplní políčka, kde chce aby došlo ke změně nastavení a potvrdí změny
	4.	Systém uloží provedené změny
Alternativní scénář:		
Výsledky:		Systém uložil provedené změny v nastavení účtu uchazeče
Omezení:		

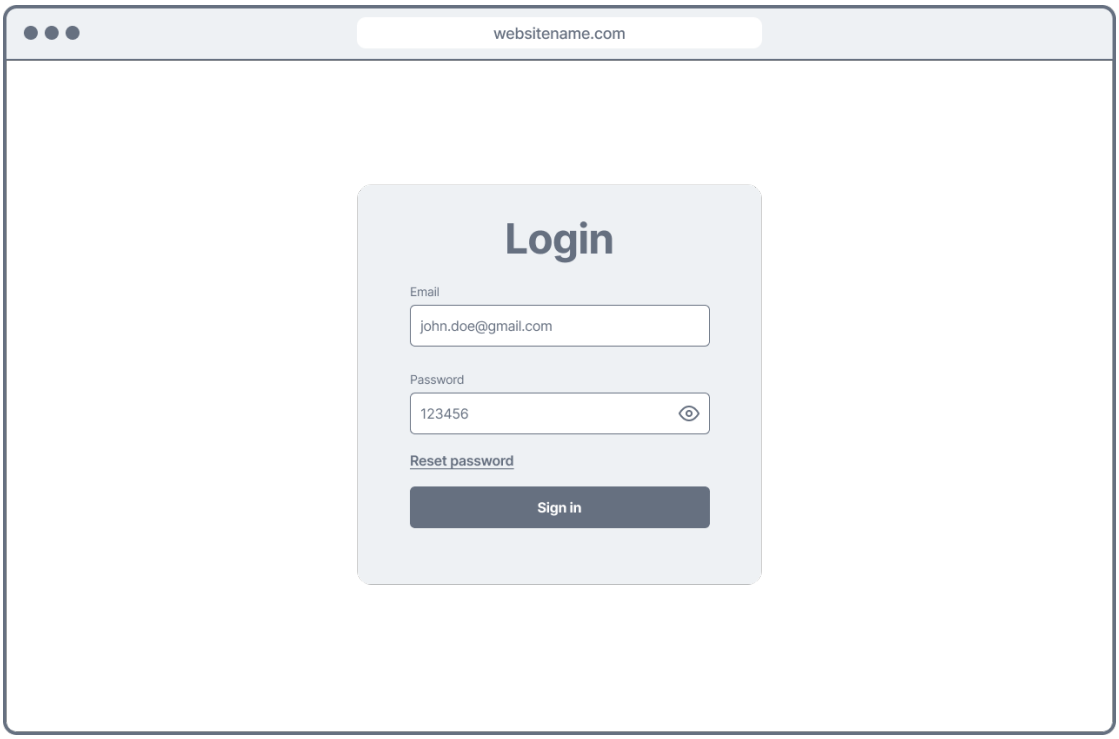
UC13		**Vytvoření přístupových údajů systému Moodle**
Související požadavky:		FP16, FP19, FP20, FP41
Aktéři:		Pracovník VIC, uchazeč
Předpoklady:		Uchazeč je přihlášen Pracovník VIC má ČVUT identitu
Kontext:		Uchazeč chce získat své přístupové údaje do systému Moodle
Hlavní scénář:	1.	Systém periodicky zasílá požadavek na VIC o založení přístupových údajů pro nové uchazeče
	2.	Pracovník VIC reaguje na žádost, přihlásí se do aplikace a přesune se do sekce pro vytvoření přístupových údajů
	3.	Systém zobrazí pracovníku seznam nových uchazečů, kde může každému uchazeči vyplnit přístupové údaje
	4.	Pracovník VIC vyplní daným uchazečům přístupové údaje a potvrdí uložení
	5.	Systém uloží přístupové údaje a notifikuje uchazeče o nových přístupových údajích
	6.	Uchazeč se přihlásí do svého účtu a přejde do detailu přihlášky
	7.	Systém zobrazí uchazeči přístupové údaje, kde heslo zůstane skryté s možností odkrytí
	8.	Uchazeč klikne na ikonu pro odkrytí hesla
	9.	Systém zobrazí odkryje heslo
Alternativní scénář:		
Výsledky:		Uchazeči byly vytvořeny přístupové údaje do systému Moodle Uchazeč vidí přístupové údaje v detailu přihlášky
Omezení:		

UC14		**Synchronizace dat vůči systému KOS**
Související požadavky:		FP40, FP42, FP43
Aktéři:		Administrátor
Předpoklady:		Administrátor je přihlášen
Kontext:		Administrátor chce aktualizovat data vůči databázi KOS
Hlavní scénář:	1.	Administrátor klikne na tlačítko pro vstup do sekce pro import dat
	2.	Systém přesměruje administrátora do sekce pro import dat
	3.	Administrátor vidí poslední datum importu a status, zda-li je automatický import spuštěný. Administrátor klikne na tlačítko pro nový import
	4.	Systém příjme požadavek k importu dat a notifikuje administrátora, až import bude zrealizován
Alternativní scénář:	3a.	Administrátor vidí poslední datum importu a status, zda-li je automatický import spuštěný. Administrátor změní nastavení automatického importu a klikne na tlačítko pro nový manuální import
hline Výsledky:		Data o přihláškách, událostech, a studijních programech jsou aktualizovaná vůči databáze KOS
		Uchazeč vidí přístupové údaje v detailu přihlášky
Omezení:		

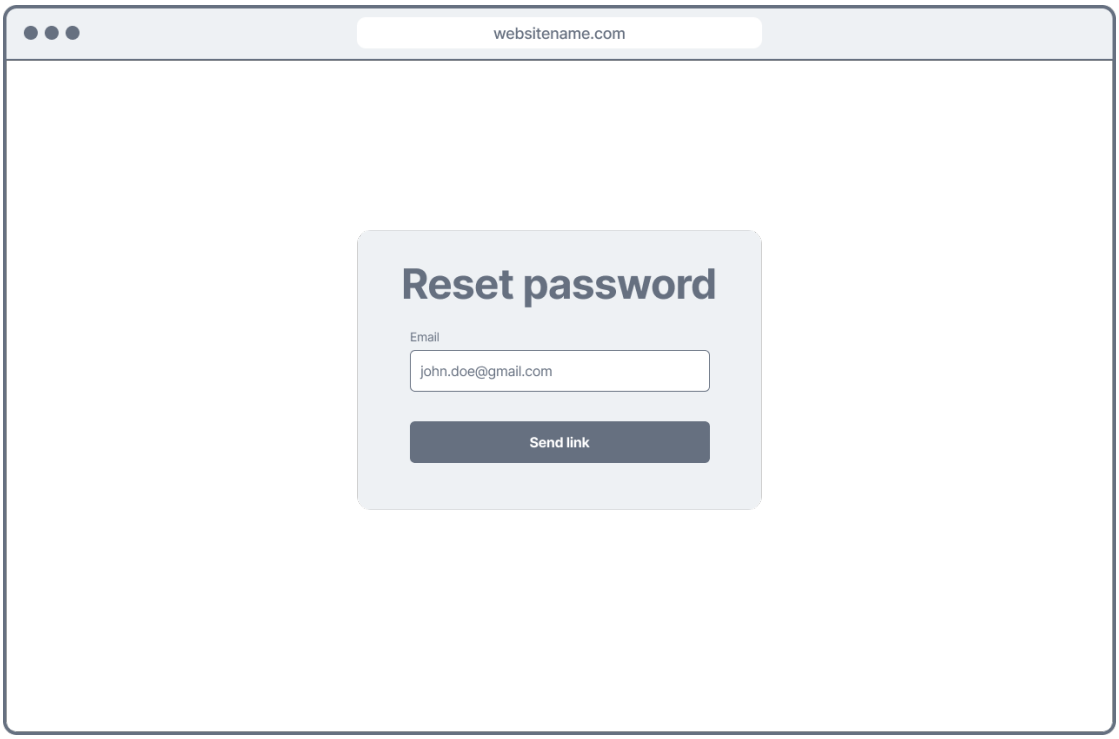
UC15		**Export výsledků**
Související požadavky:		FP45, FP46
Aktéři:		Administrátor
Předpoklady:		Administrátor je přihlášen
Kontext:		Administrátor chce exportovat data z aplikace do systému KOS
Hlavní scénář:	1.	Administrátor klikne na tlačítko pro vstup do sekce pro export dat
	2.	Systém přesměruje administrátora do sekce pro export dat
	3.	Administrátor vidí poslední datum exportu. Administrátor klikne na tlačítko pro export
	4.	Systém příjme požadavek k exportu dat a notifikuje administrátora, až budou data exportována do KOS databáze
Alternativní scénář:	3a.	Administrátor zvolí možnost exportu do CSV souboru, které jsou ve formátu rozpoznatelný systémem iKOS. Administrátor manuálně importuje pomocí CSV souboru data do databáze KOS
Výsledky:		Lokální data byly zapsána do databáze KOS
Omezení:		

Příloha C

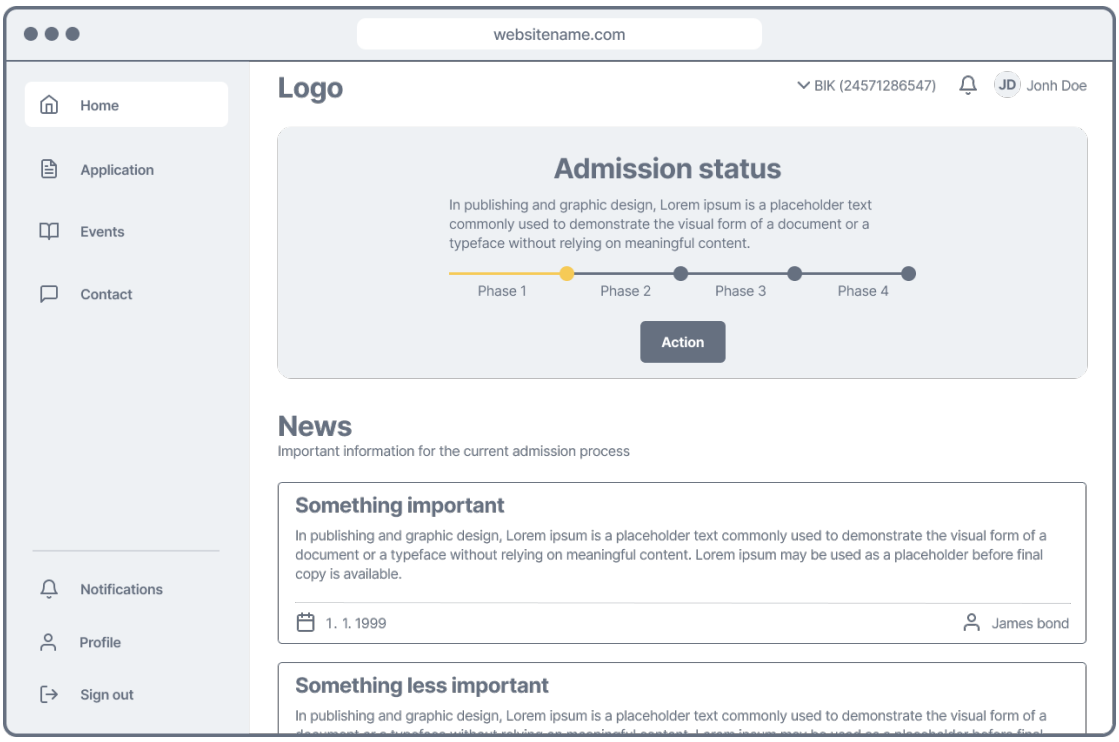
Lo-Fi prototyp



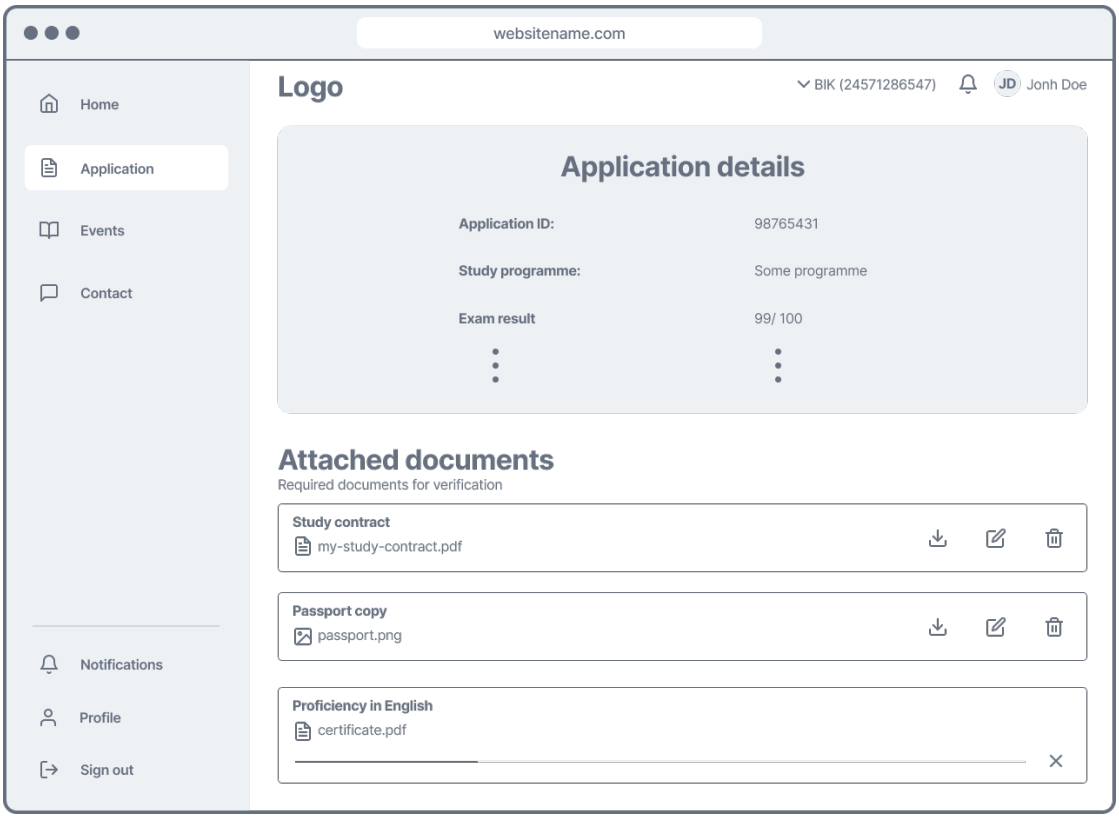
Obrázek C.1 Wireframe přihlašovací obrazovky



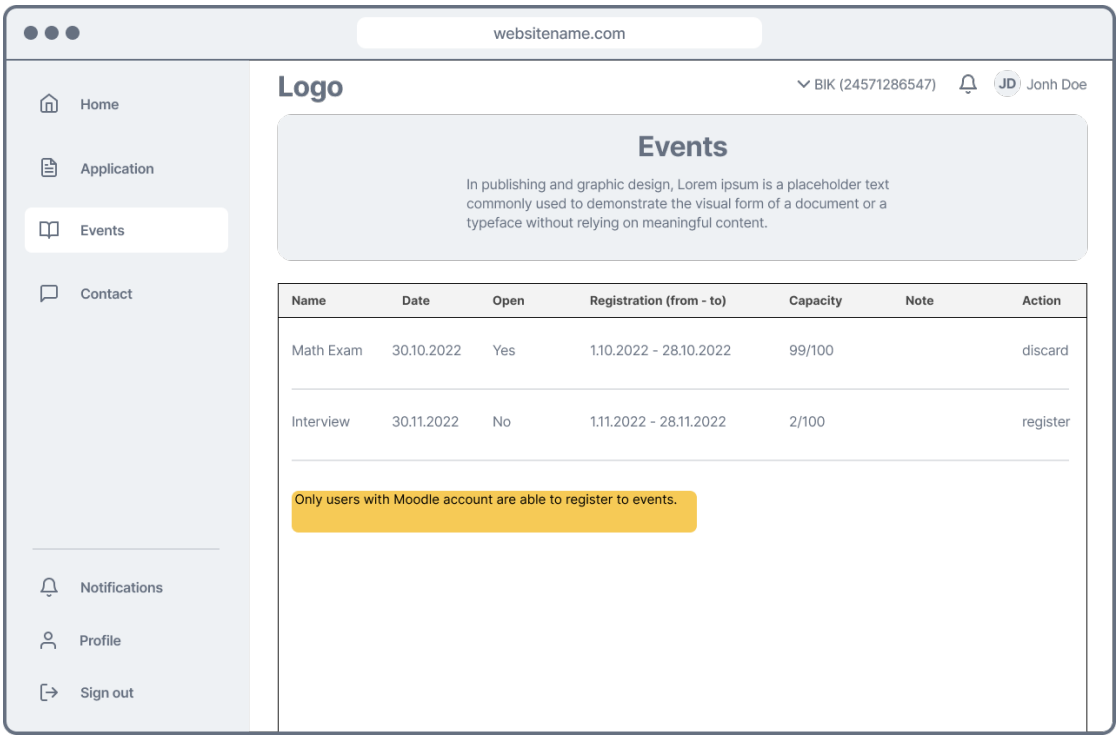
Obrázek C.2 Wireframe obrazovky pro obnovu hesla



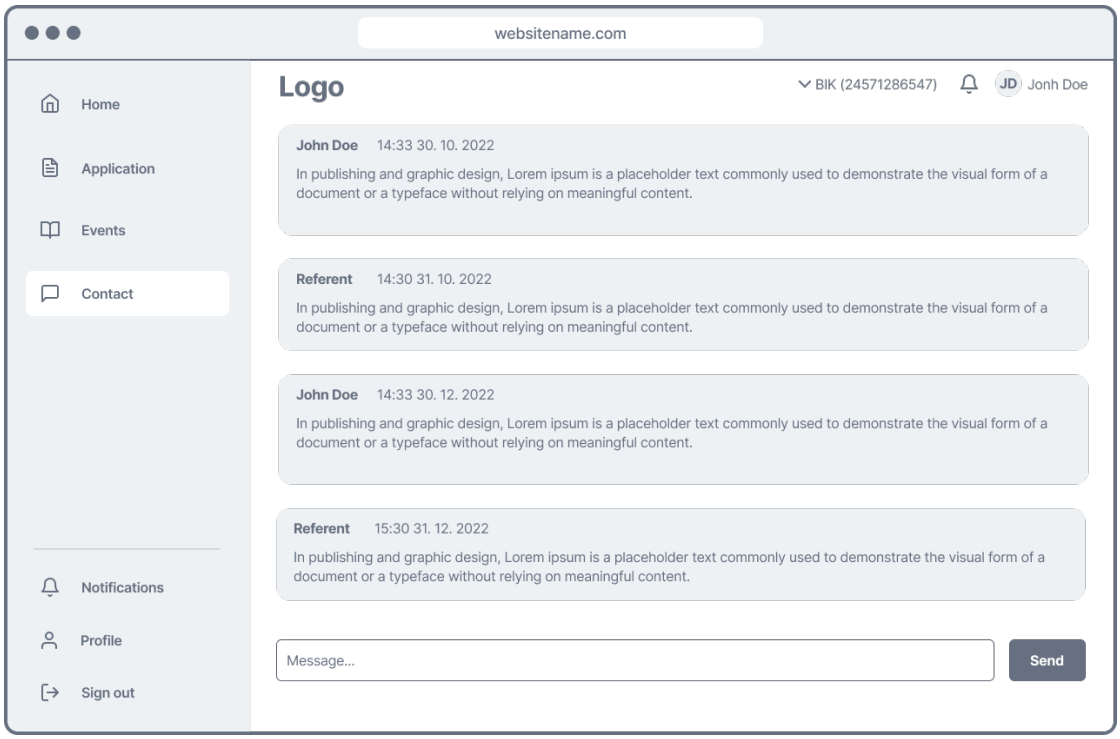
Obrázek C.3 Wireframe domovské obrazovky v uživatelské sekci



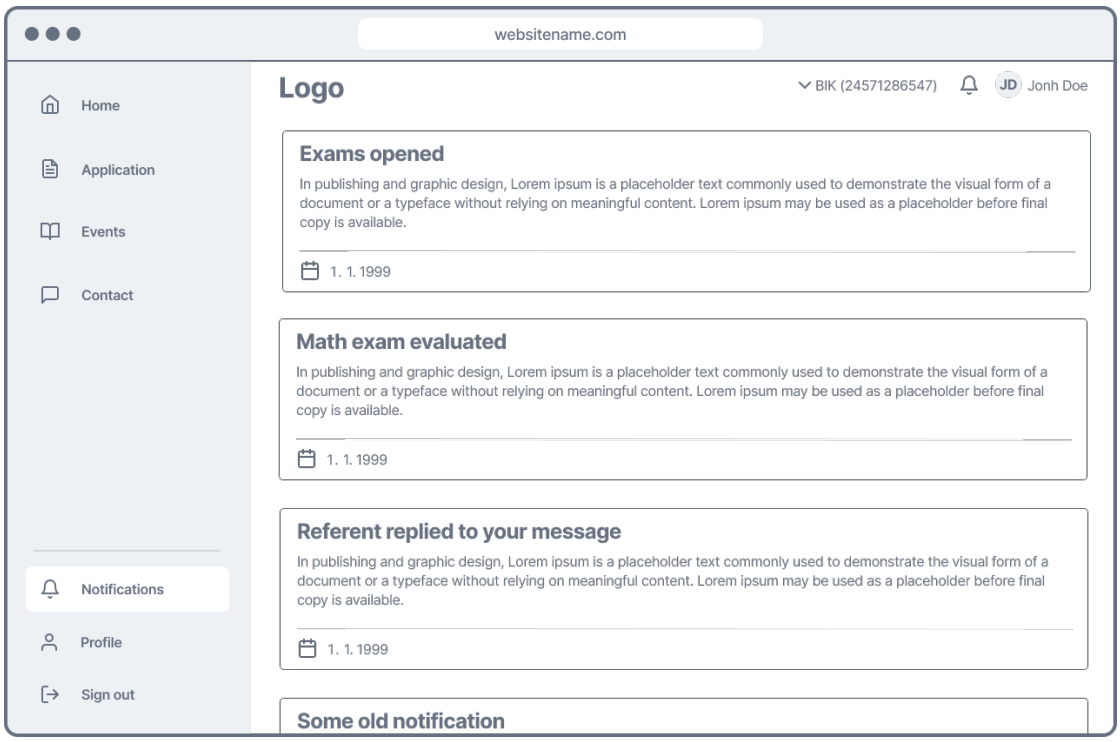
Obrázek C.4 Wireframe detailu přihlášky v uživatelské sekci



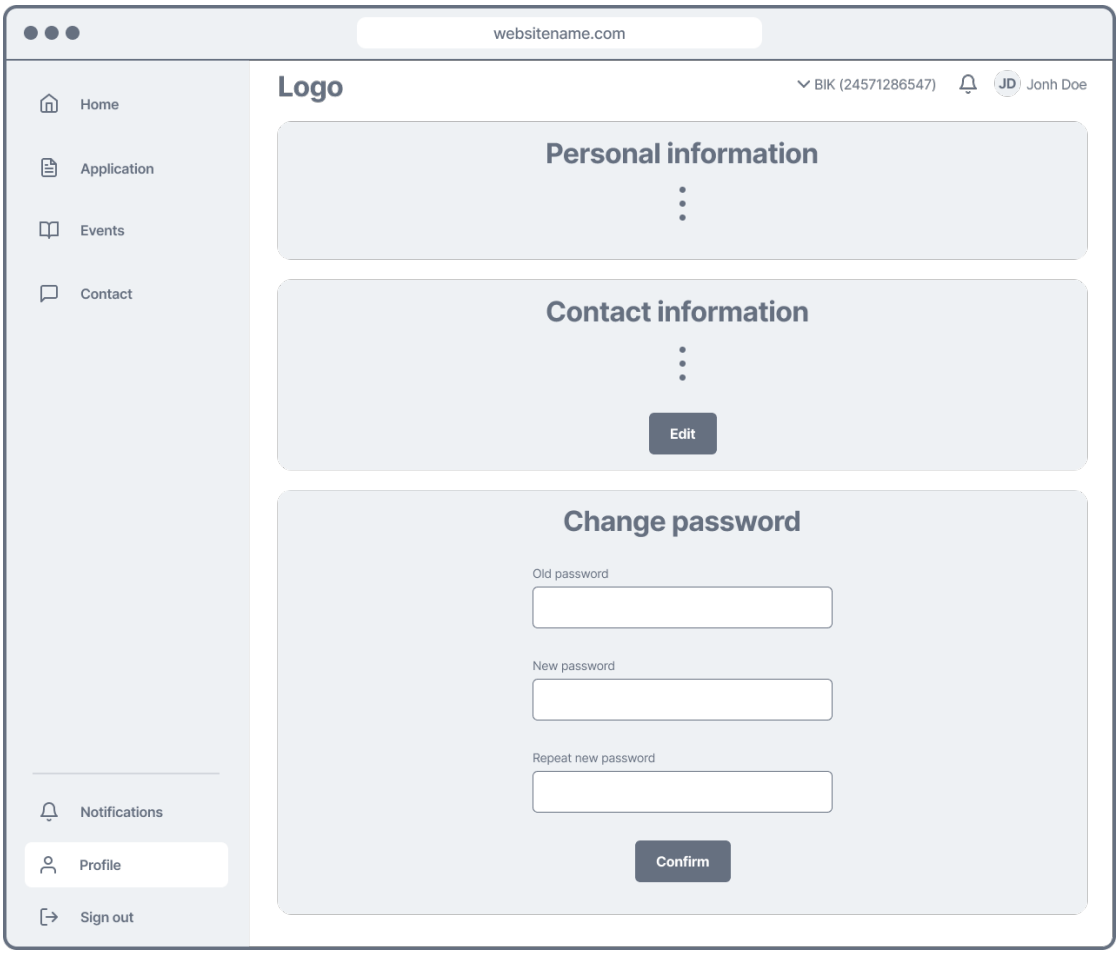
Obrázek C.5 Wireframe událostí v uživatelské sekci



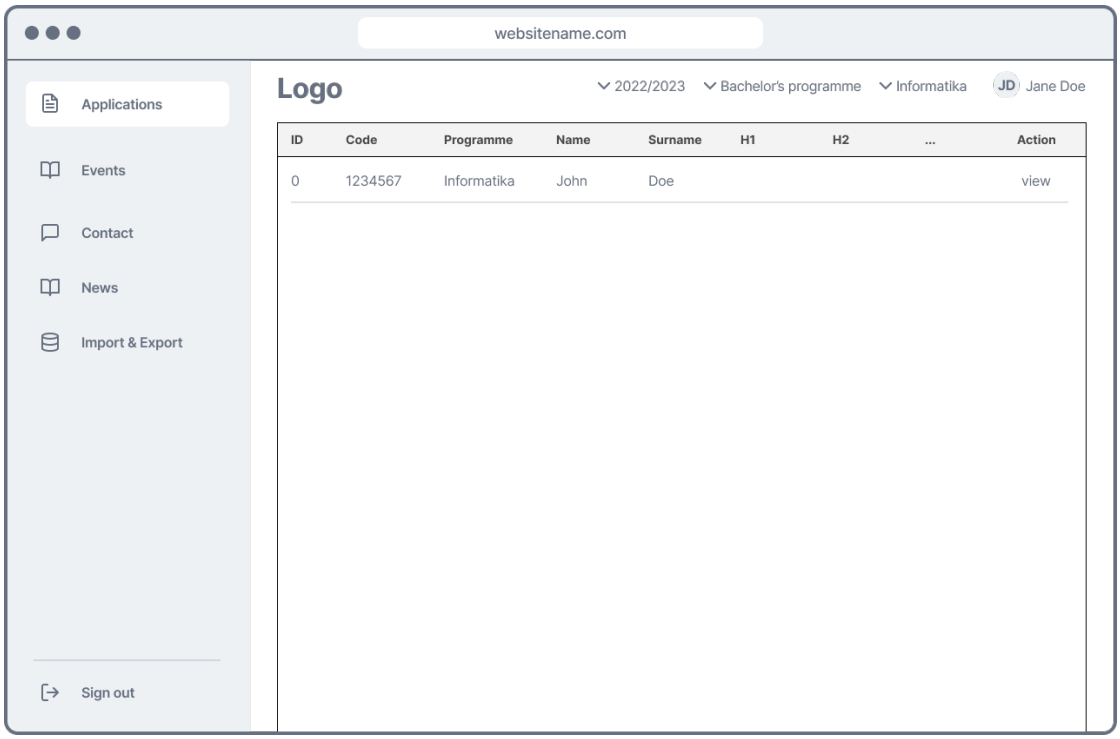
Obrázek C.6 Wireframe konverzací v uživatelské sekci



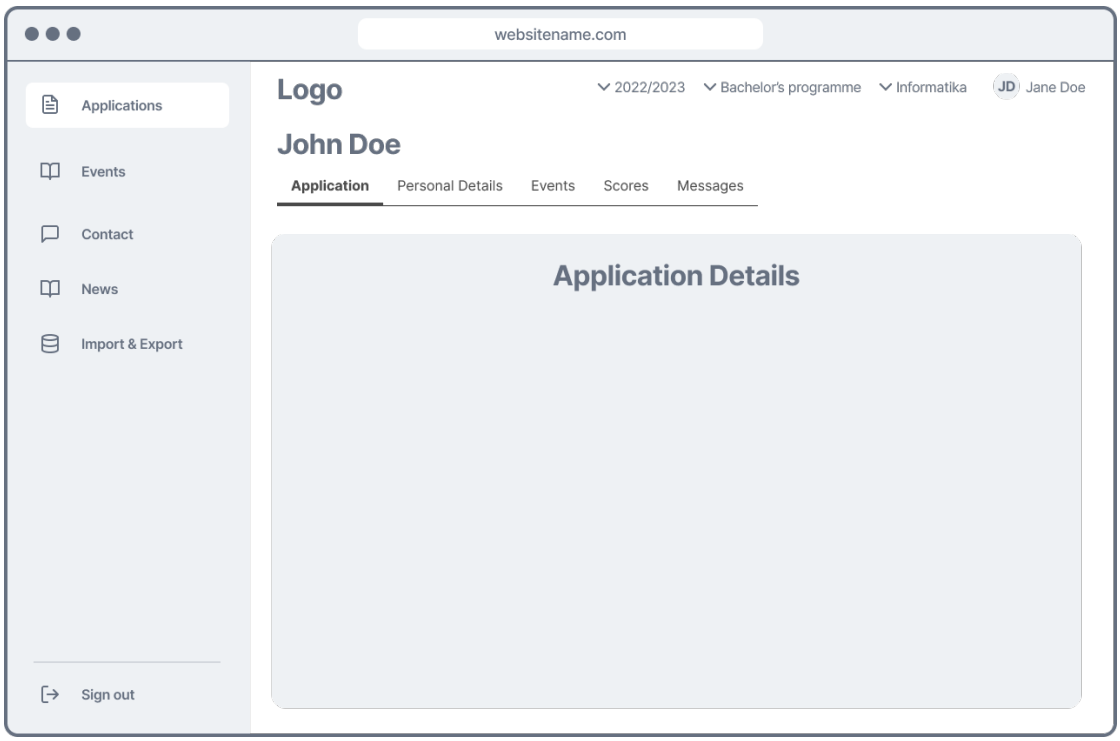
Obrázek C.7 Wireframe notifikací v uživatelské sekci



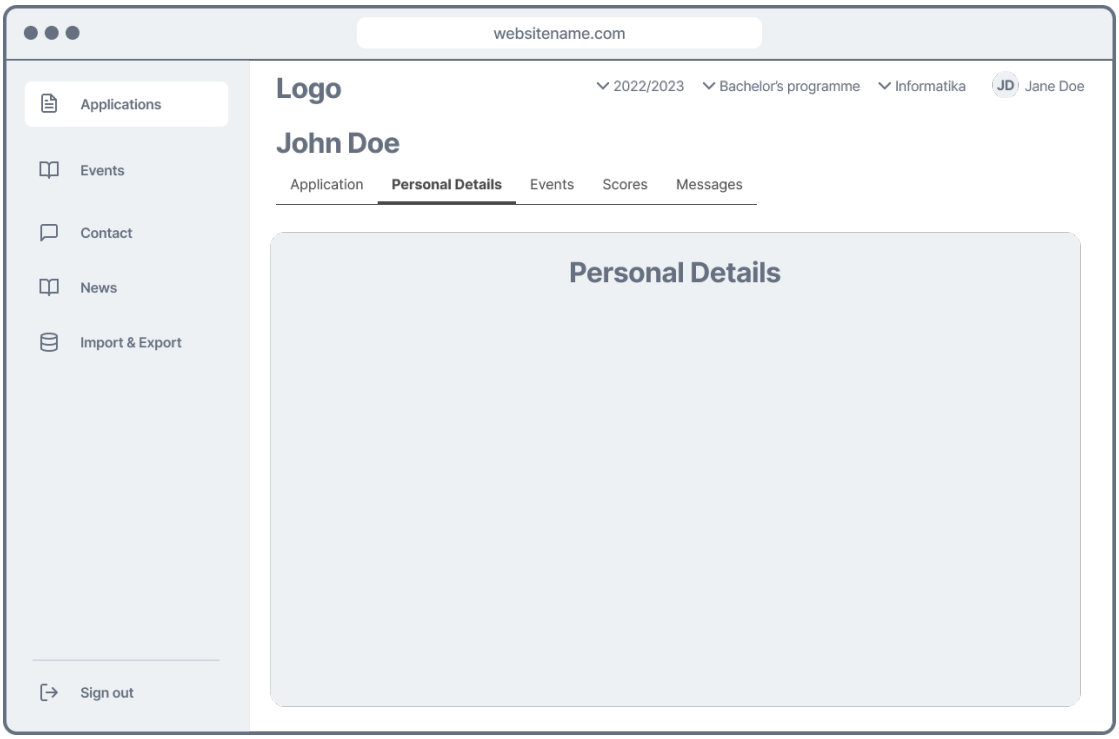
Obrázek C.8 Wireframe uživatelského profilu v uživatelské sekci



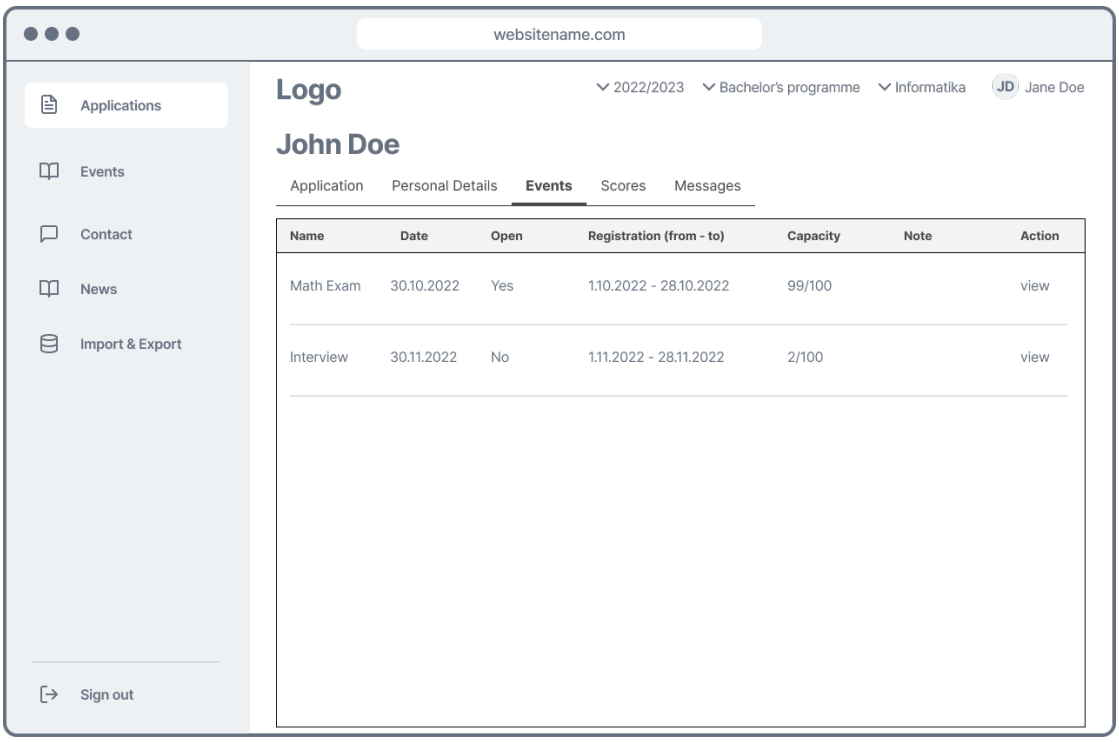
Obrázek C.9 Wireframe přehledu přihlášek v administrátorské sekci



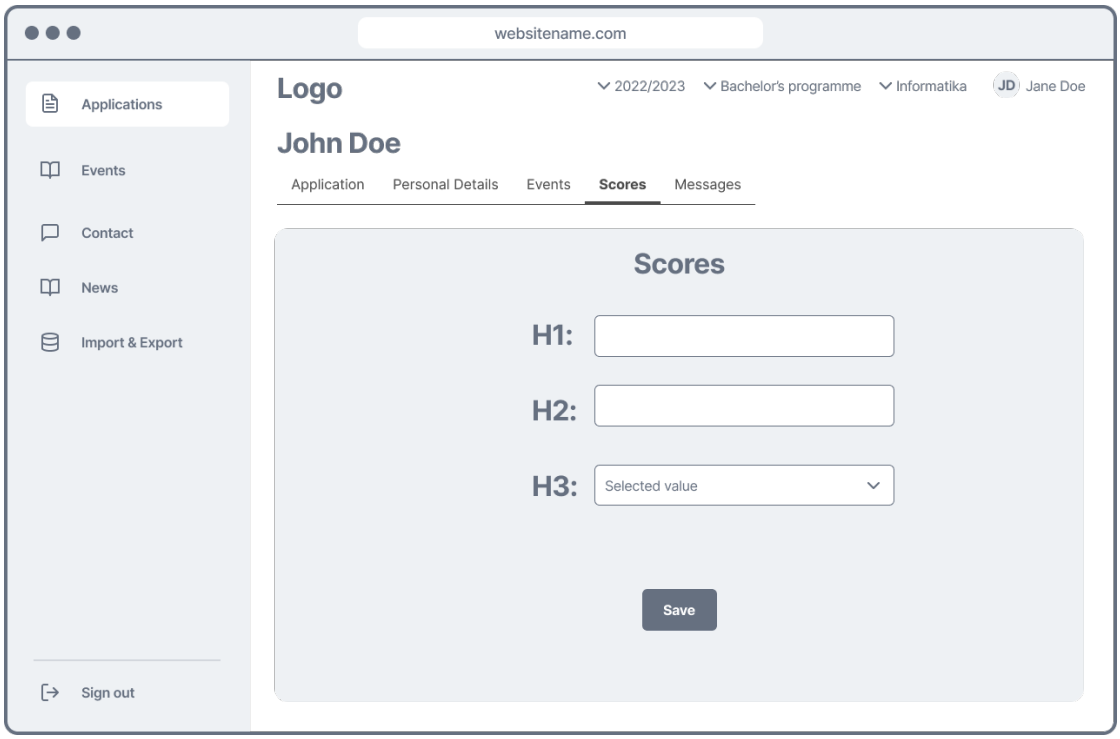
Obrázek C.10 Wireframe detailu přihlášky v administrátorské sekci



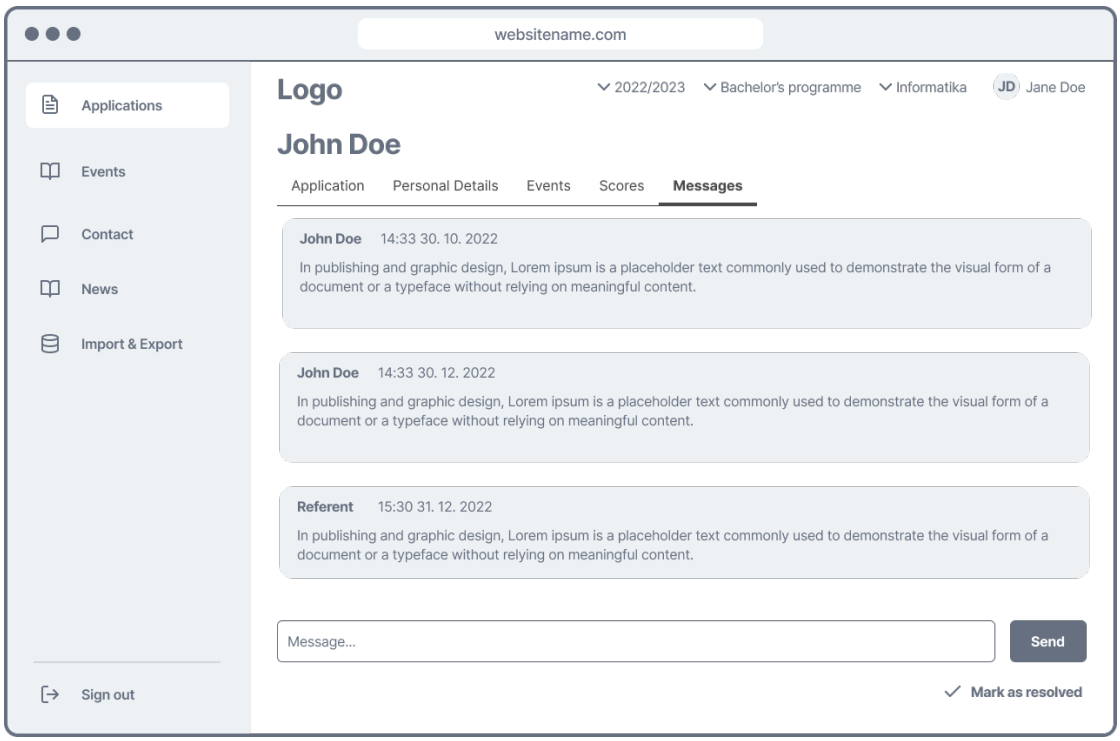
Obrázek C.11 Wireframe detailu přihlášky sekce osobní údaje v administrátorské sekci



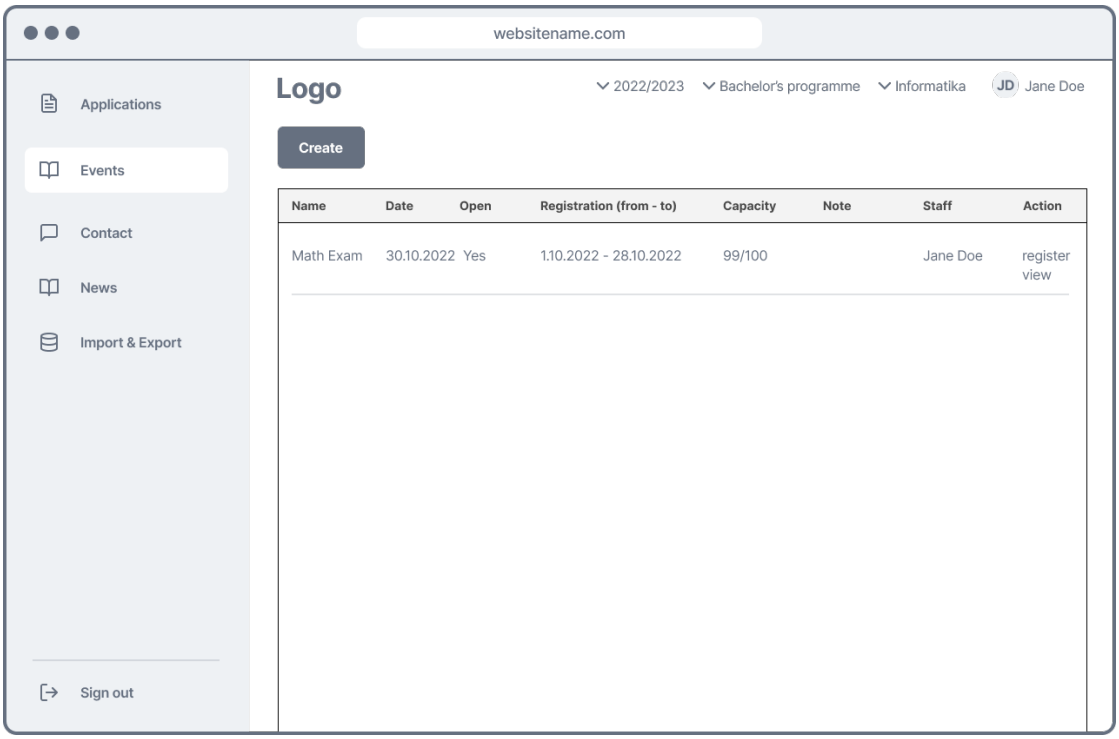
Obrázek C.12 Wireframe detailu přihlášky sekce událostí v administrátorské sekci



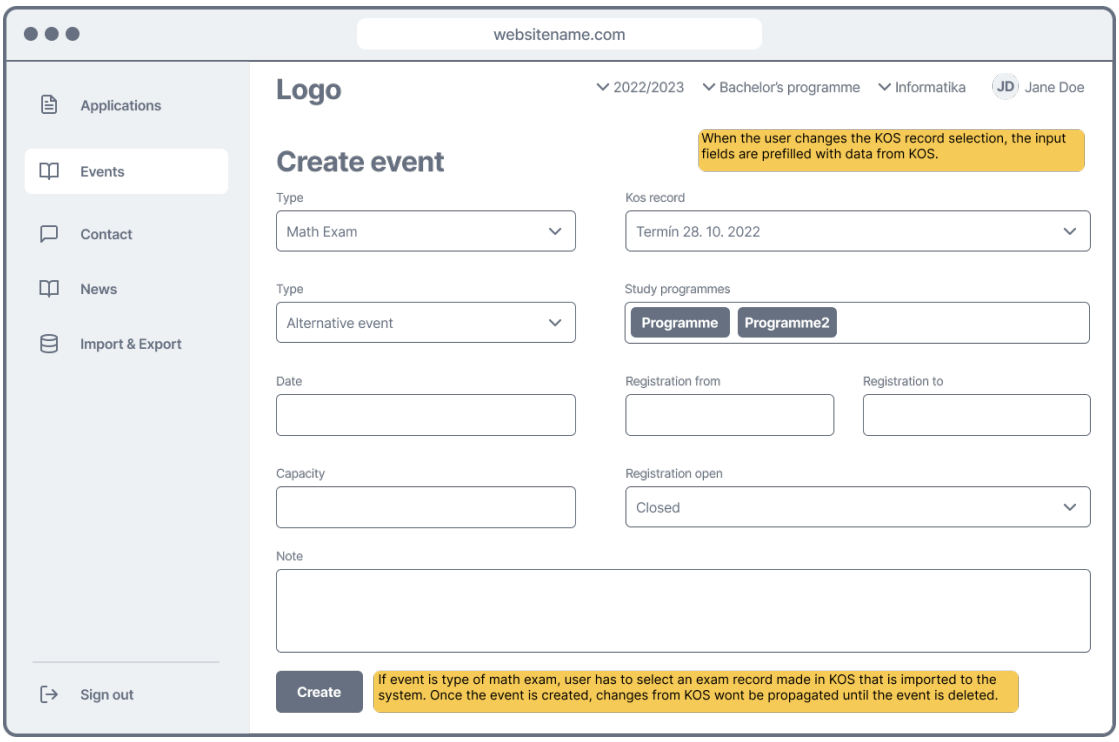
Obrázek C.13 Wireframe detailu přihlášky sekce hodnocení v administrátorské sekci



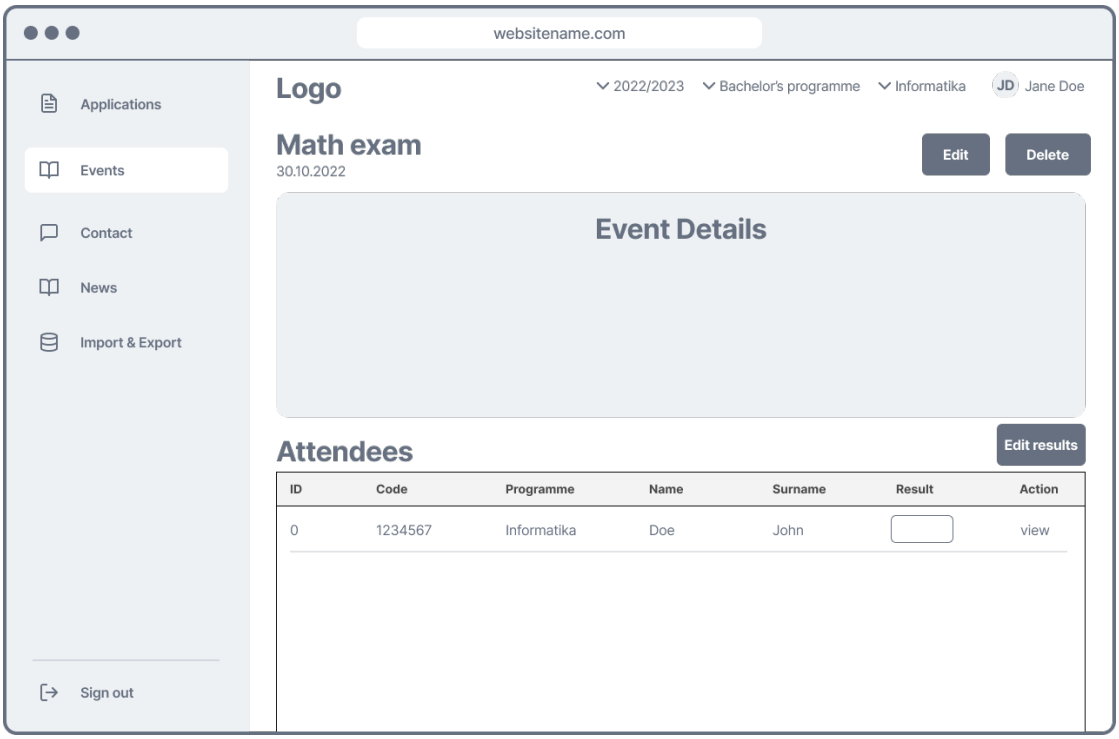
Obrázek C.14 Wireframe detailu přihlášky sekce konverzací v administrátorské sekci



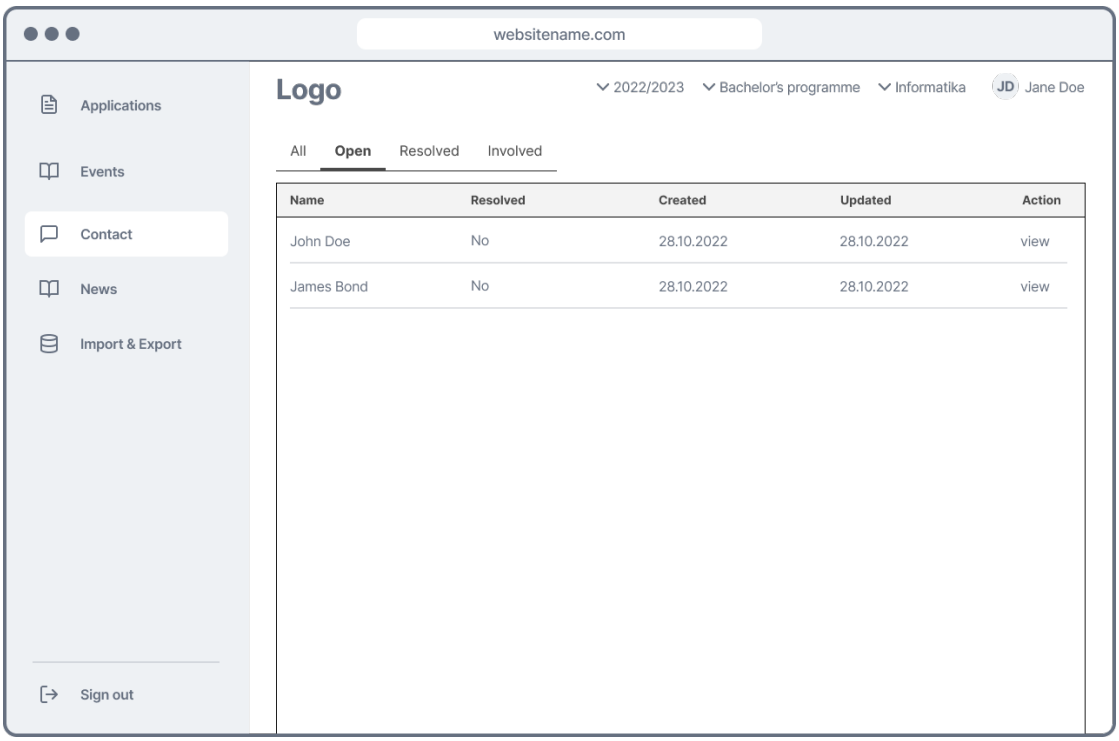
Obrázek C.15 Wireframe přehledu událostí v administrátorské sekci



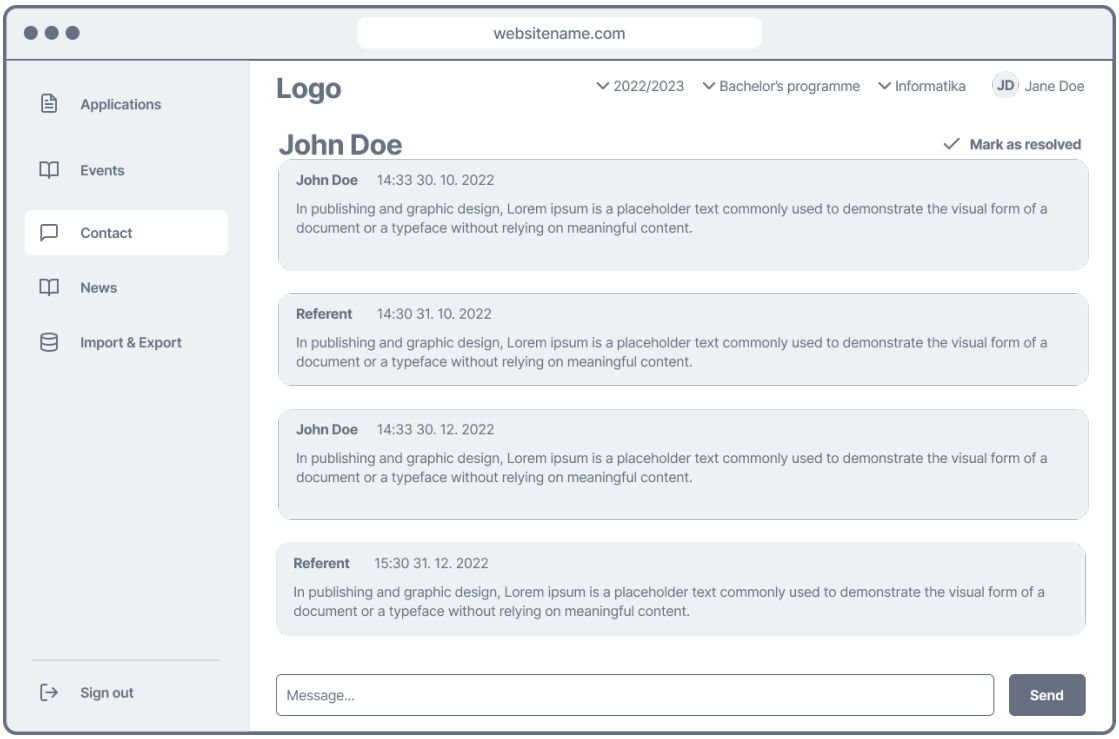
Obrázek C.16 Wireframe vytvoření událost v administrátorské sekci



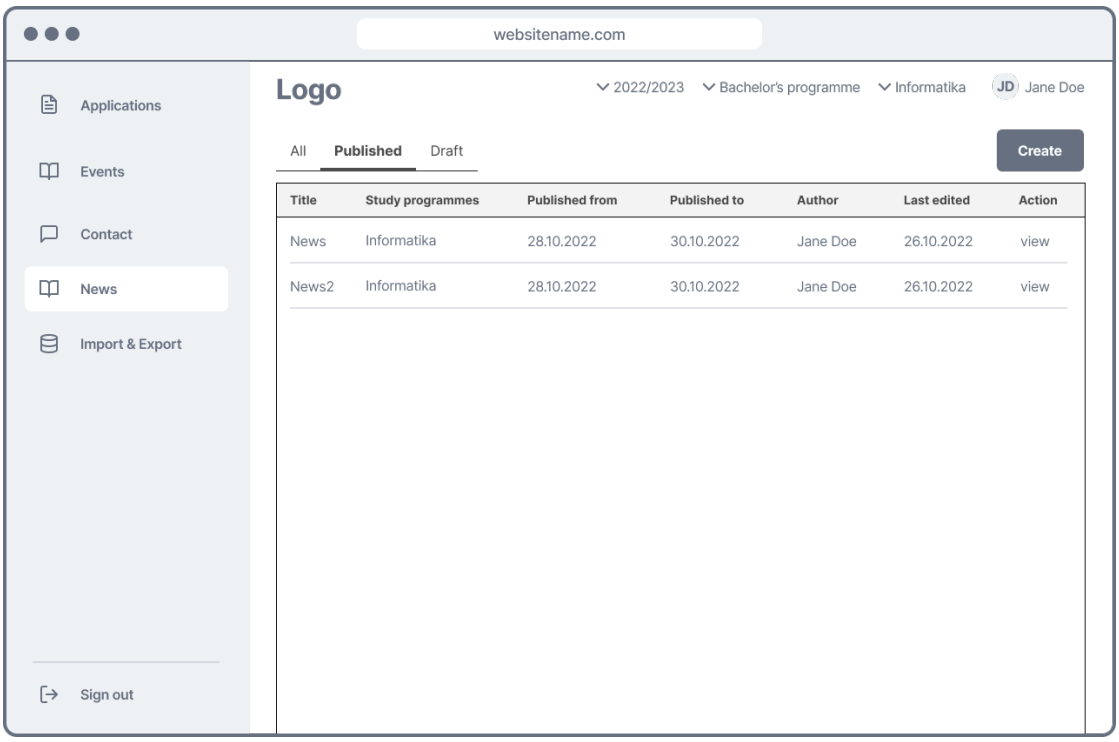
Obrázek C.17 Wireframe detailu události v administrátorské sekci



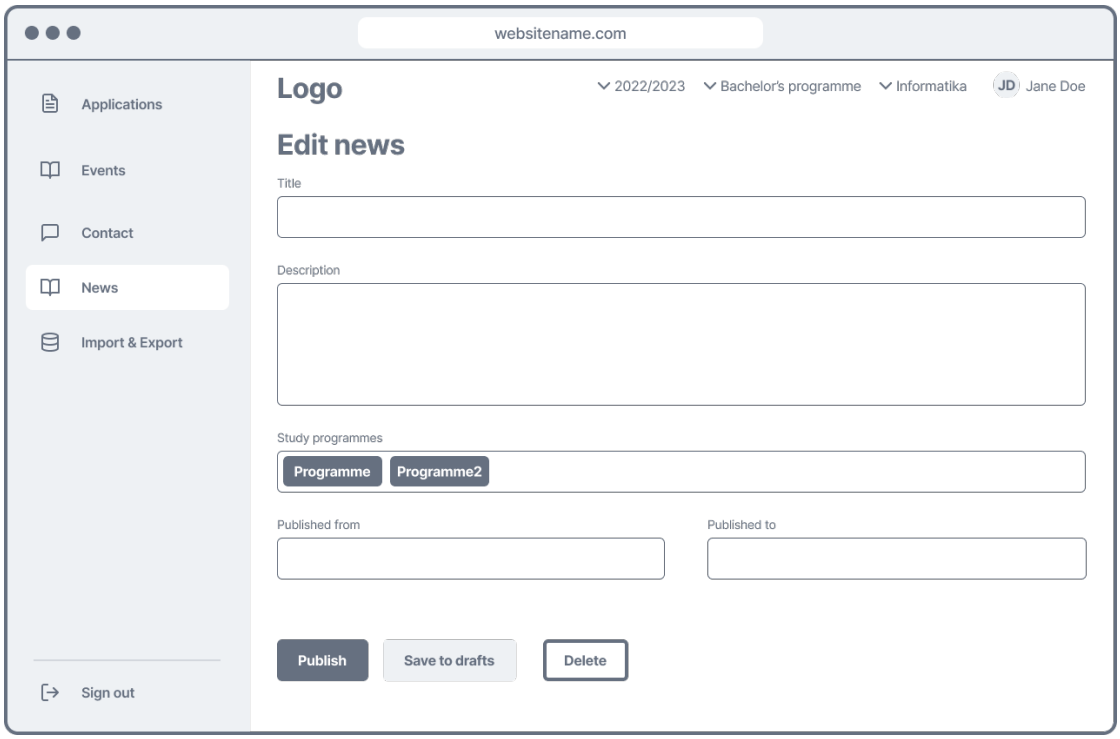
Obrázek C.18 Wireframe přehledu konverzací v administrátorské sekci



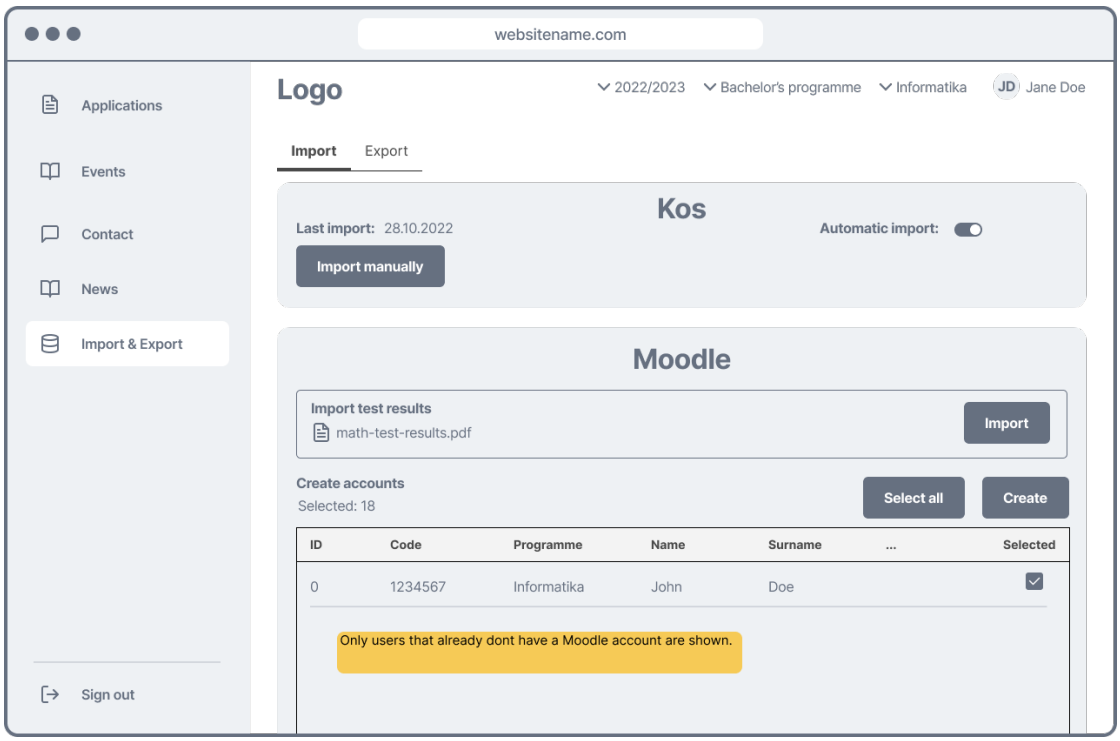
Obrázek C.19 Wireframe detailu konverzace v administrátorské sekci



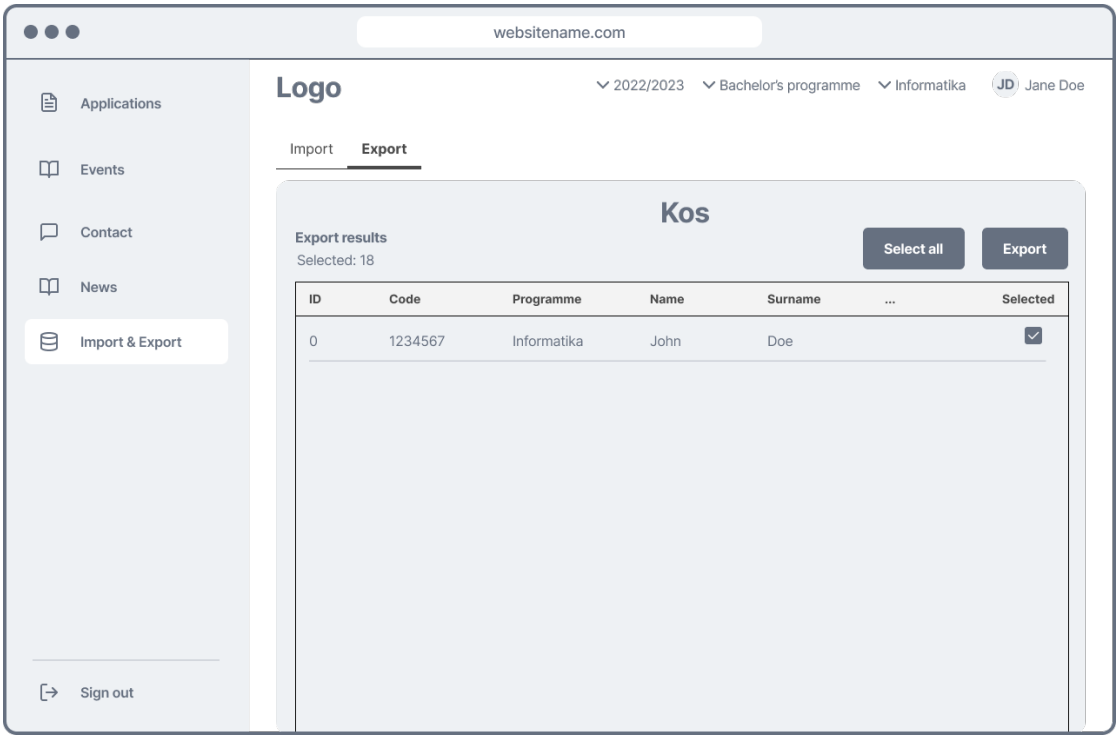
Obrázek C.20 Wireframe přehledu aktualit v administrátorské sekci



Obrázek C.21 Wireframe detailu aktuality v administrátorské sekci



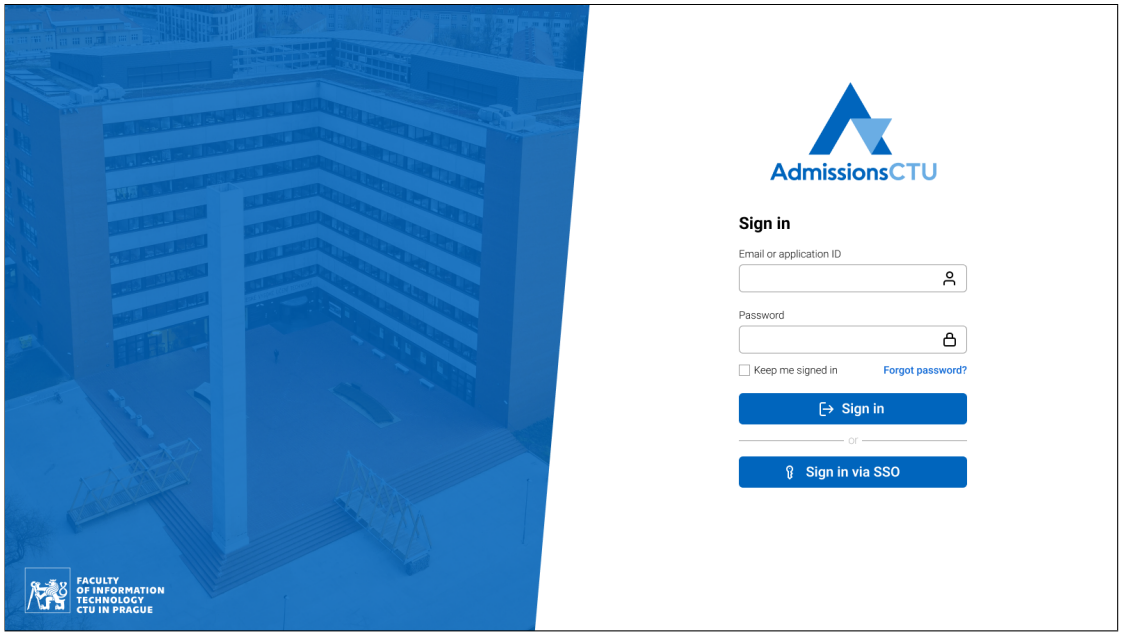
Obrázek C.22 Wireframe import sekce v administrátorské sekci



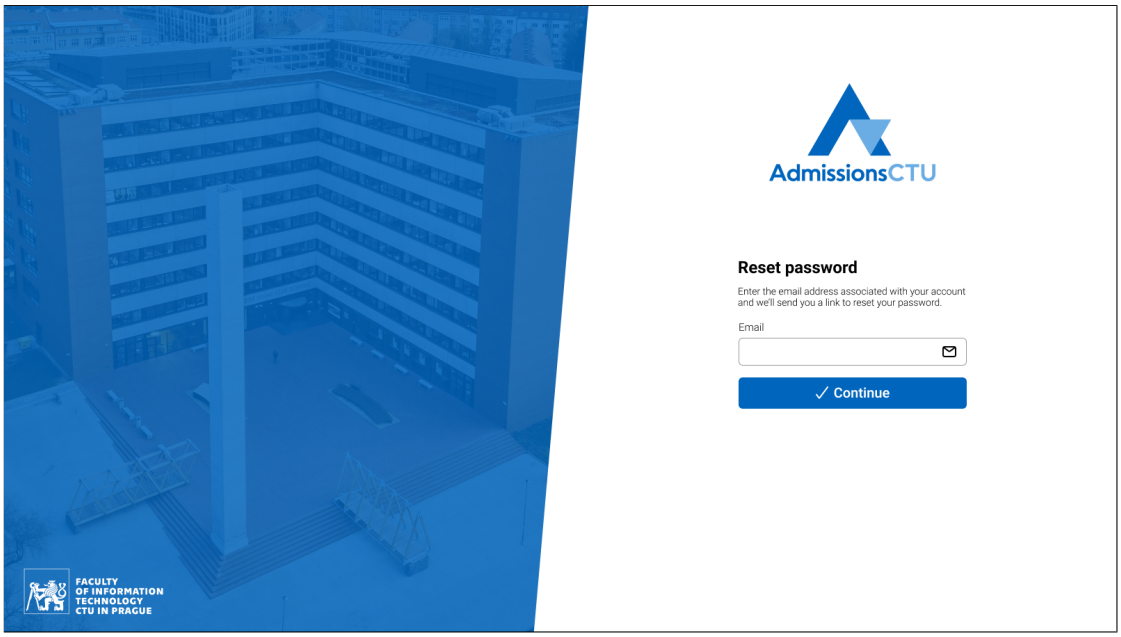
Obrázek C.23 Wireframe export sekce v administrátorské sekci

Příloha D

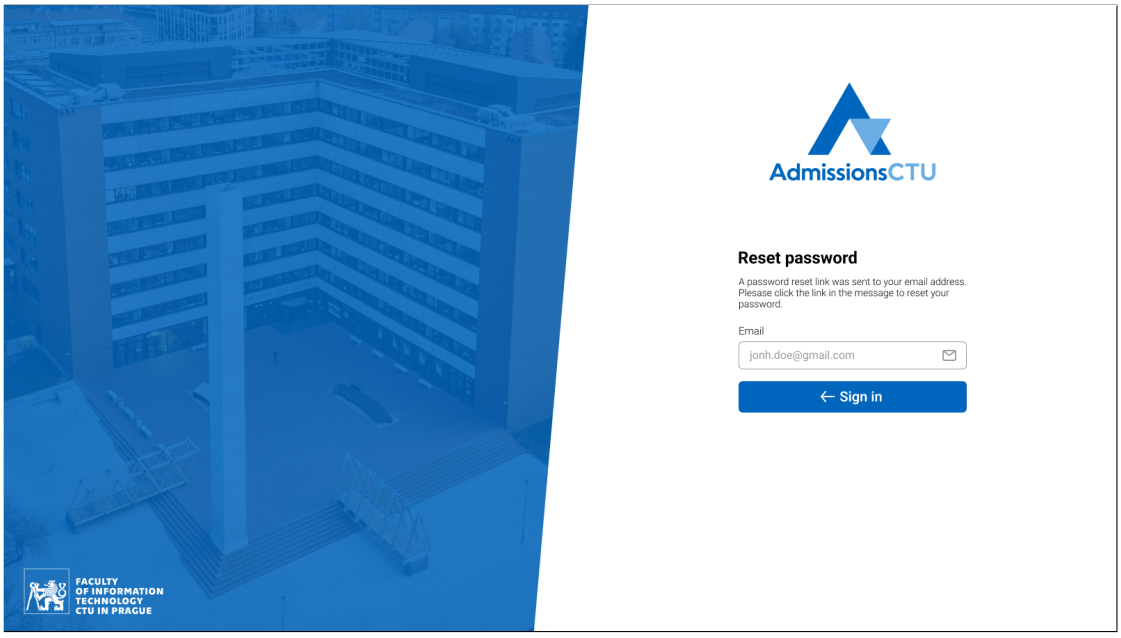
Hi-Fi prototyp



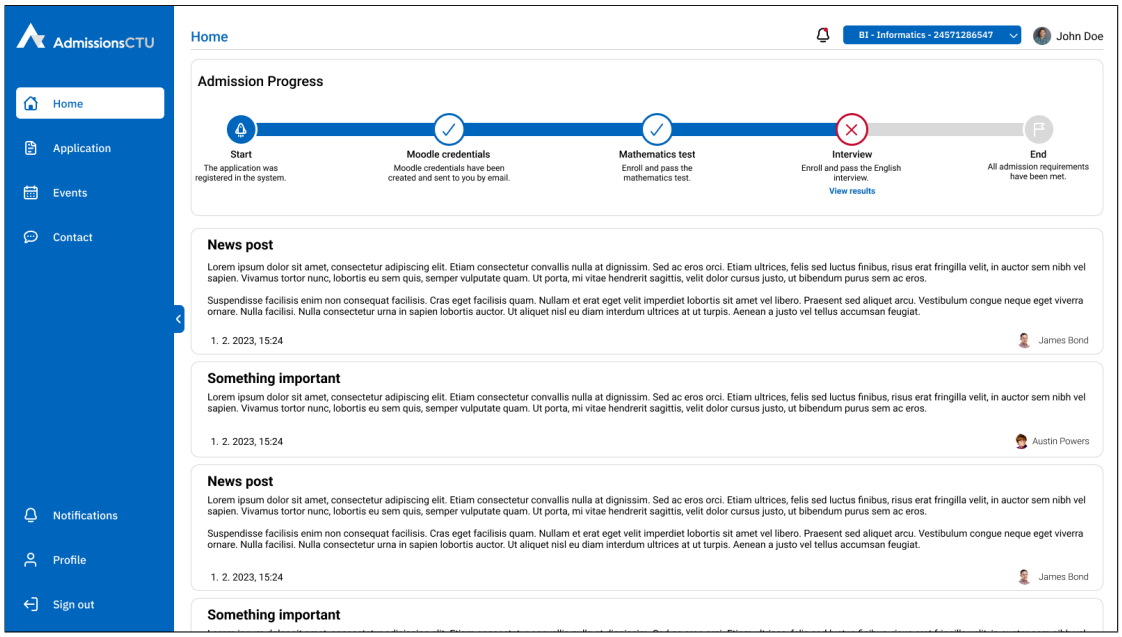
Obrázek D.1 Mockup přihlašovací obrazovky



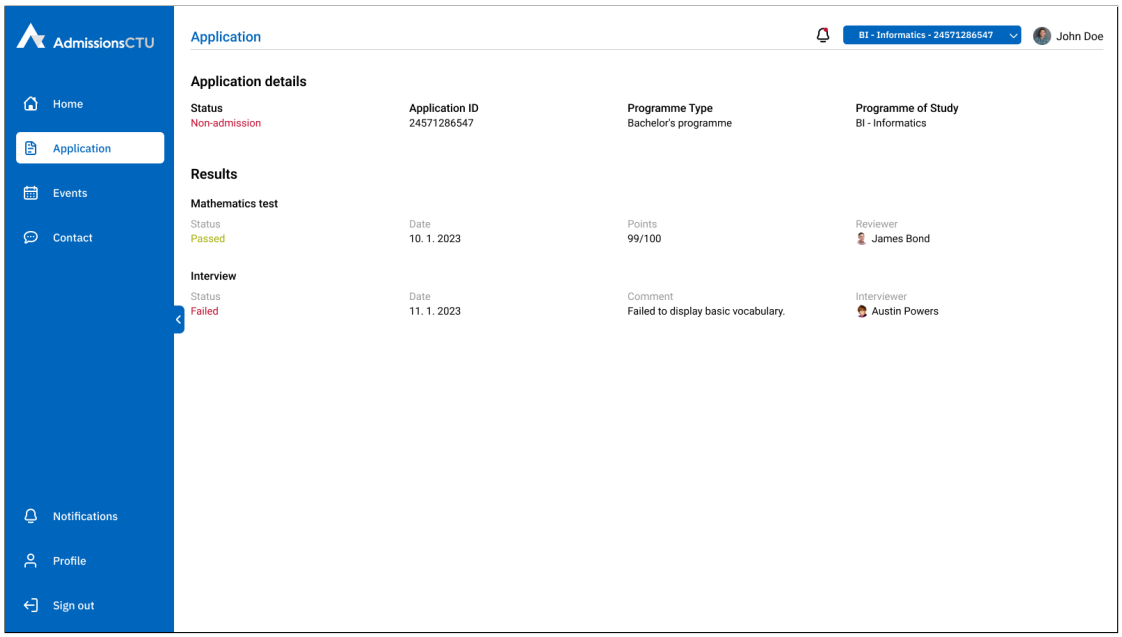
Obrázek D.2 Mockup obrazovky pro obnovu hesla



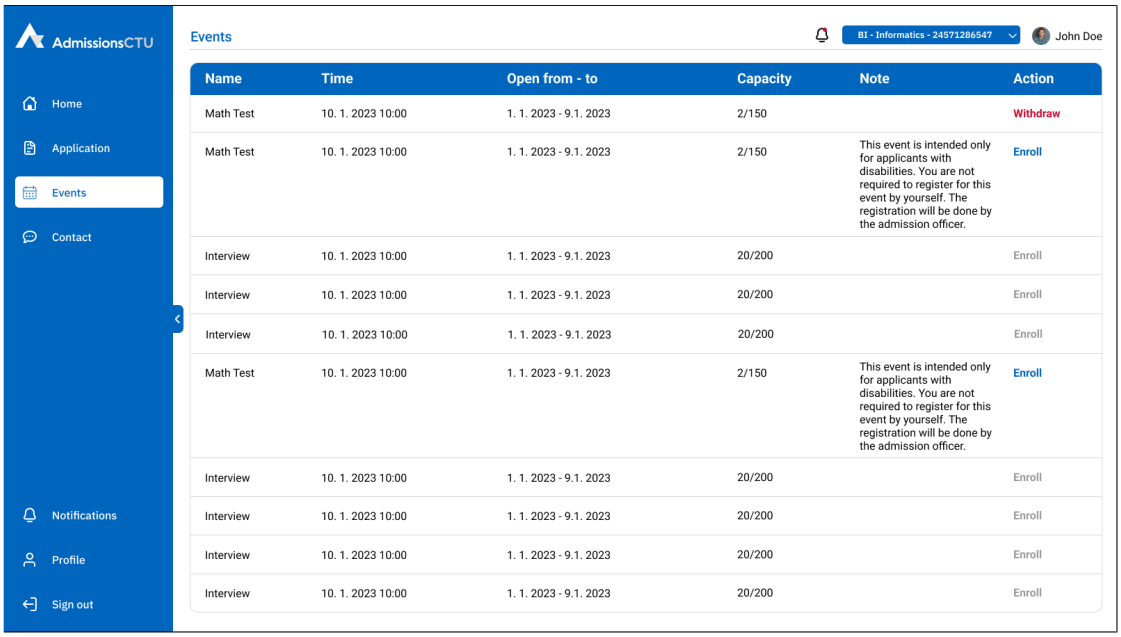
Obrázek D.3 Mockup potvrzení obnovy hesla



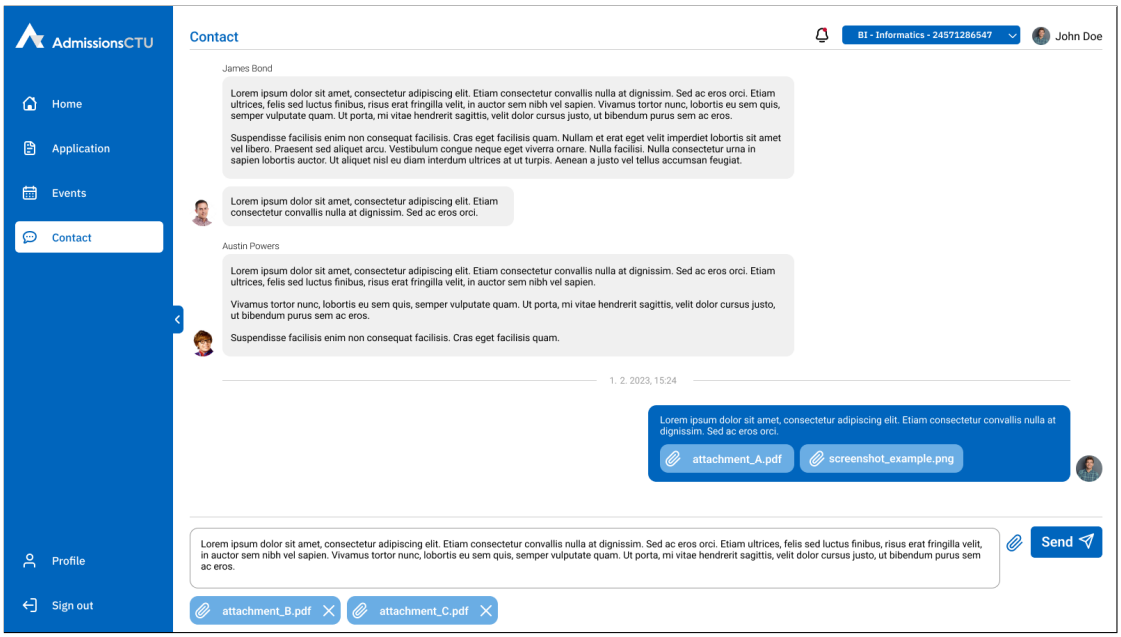
Obrázek D.4 Mockup domovské obrazovky v uživatelské sekci



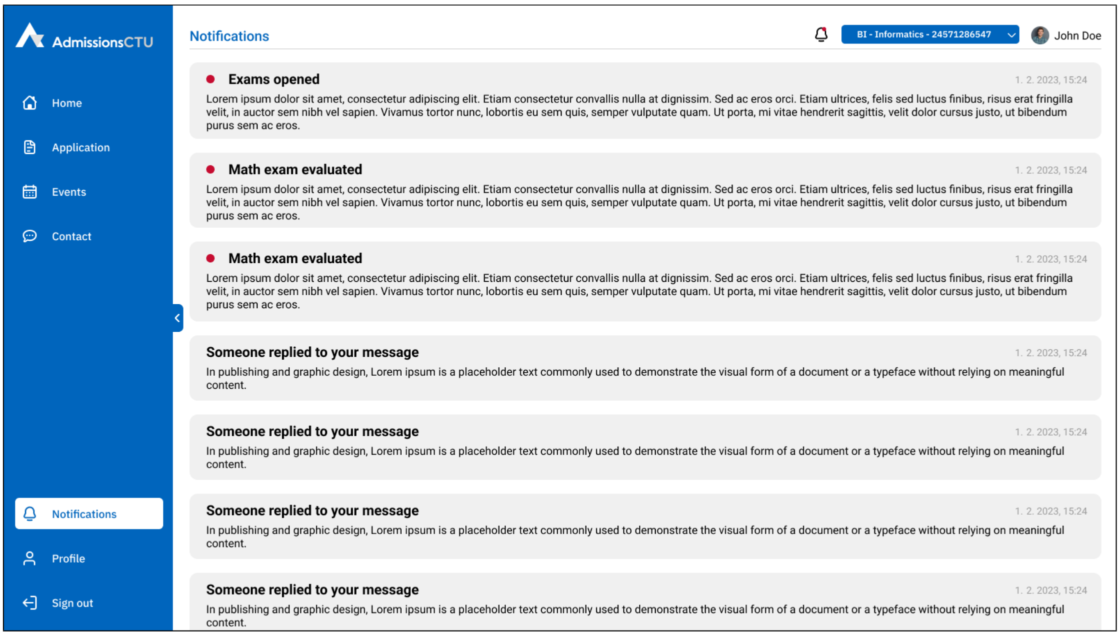
Obrázek D.5 Mockup detailu přihlášky v uživatelské sekci



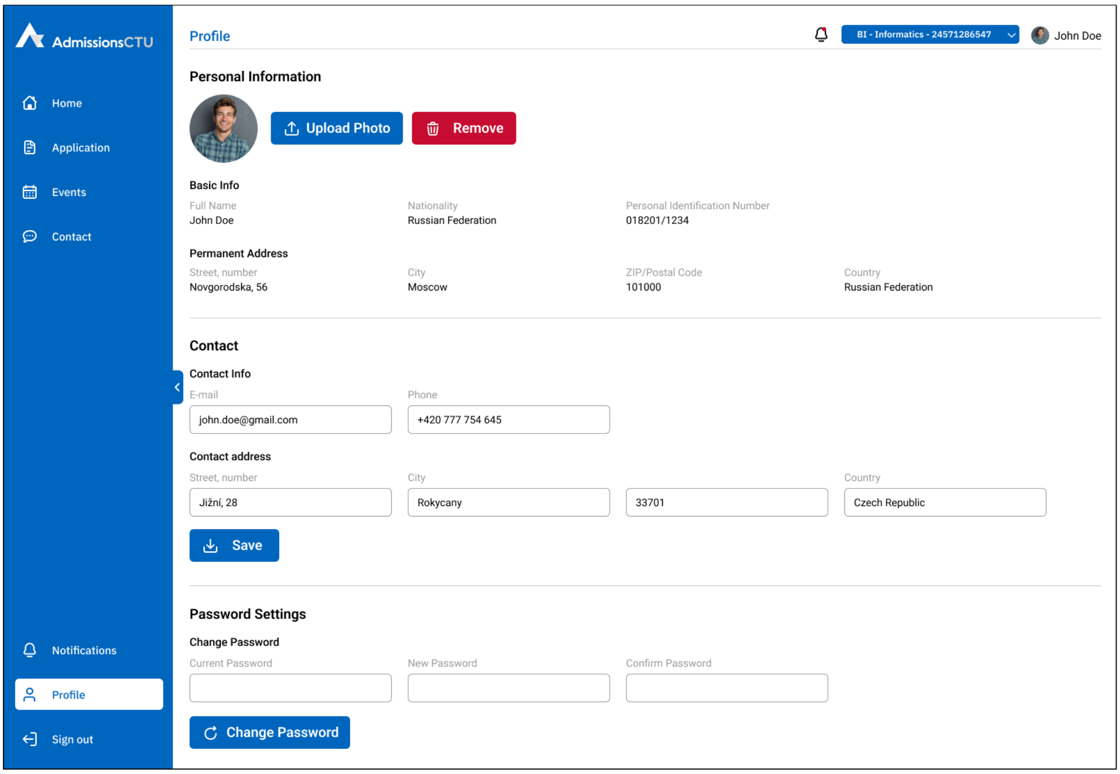
Obrázek D.6 Mockup událostí v uživatelské sekci



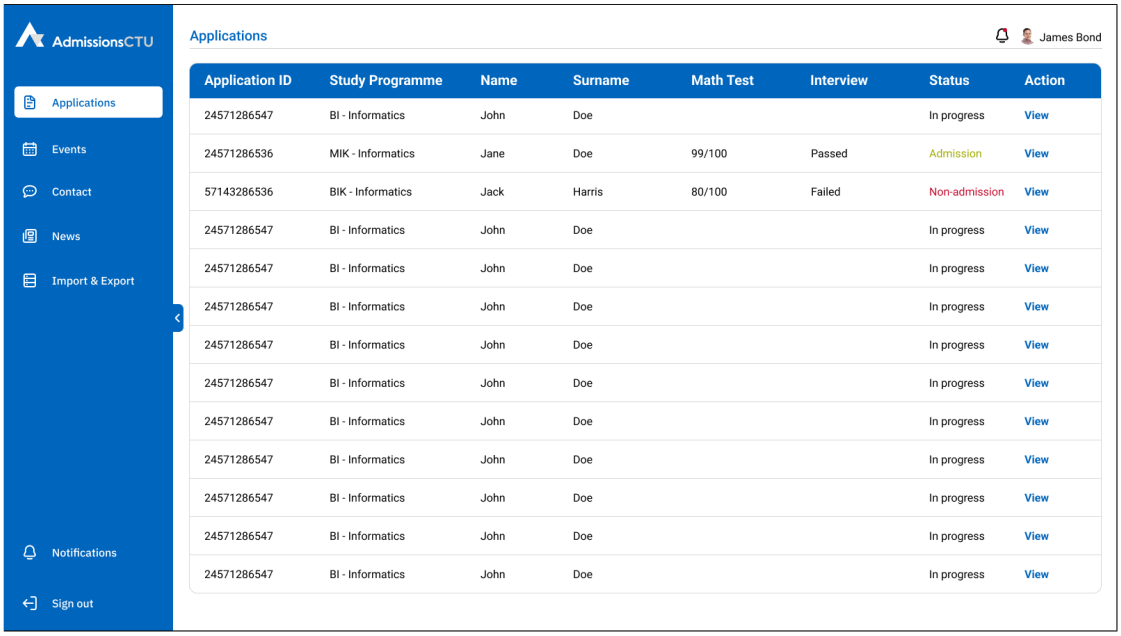
Obrázek D.7 Mockup konverzací v uživatelské sekci



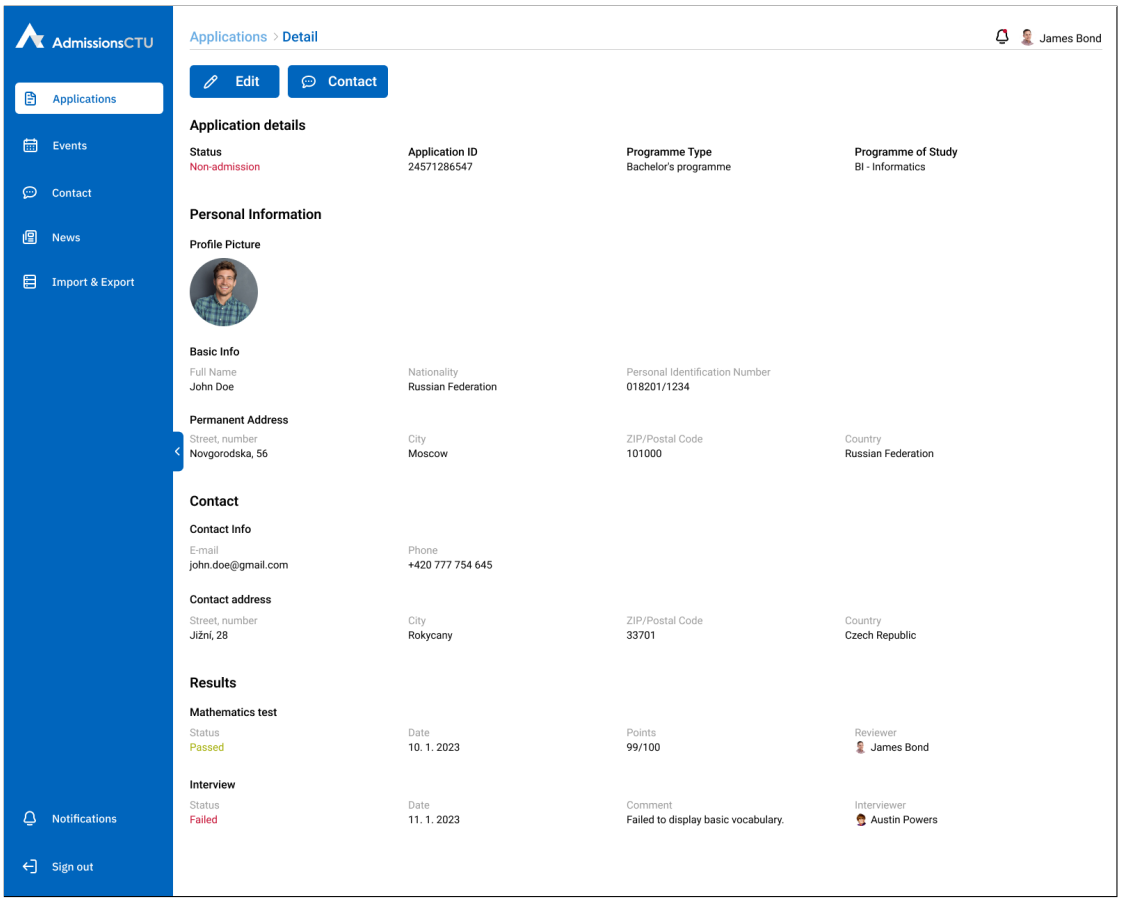
■ **Obrázek D.8** Mockup notifikací v uživatelské sekci



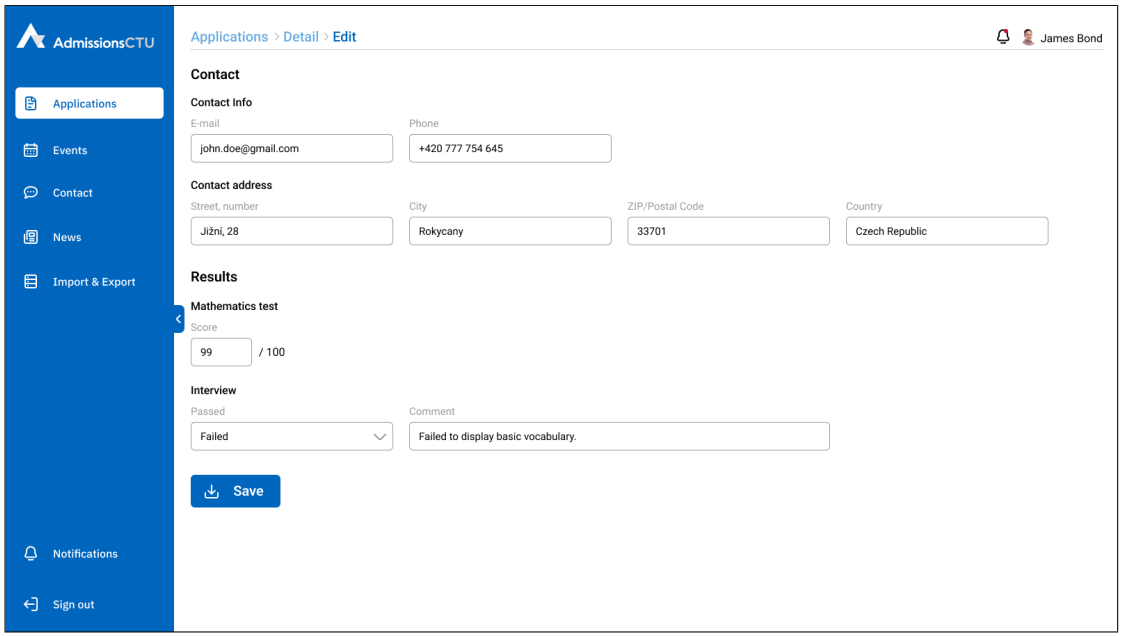
■ **Obrázek D.9** Mockup uživatelského profilu v uživatelské sekci



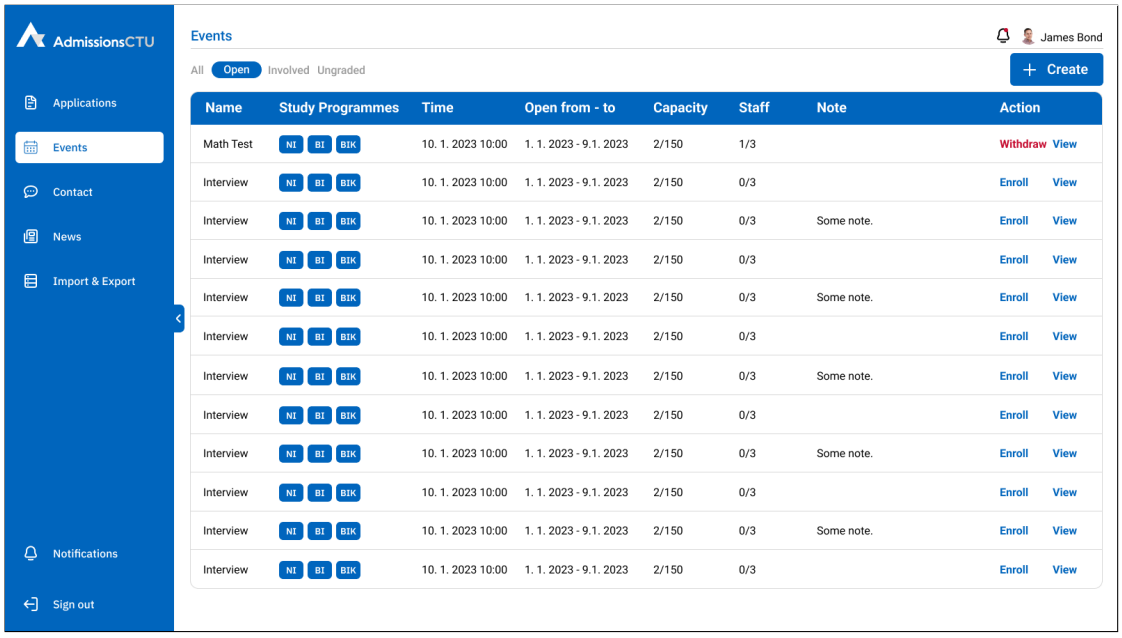
Obrázek D.10 Mockup přehledu přihlášek v administrátorské sekci



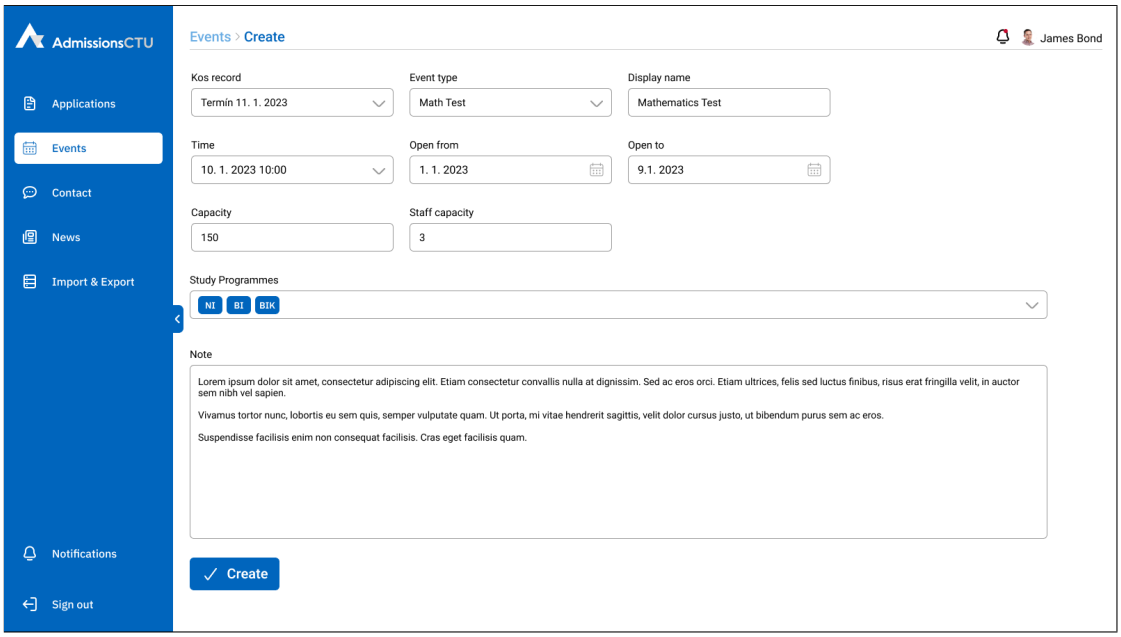
Obrázek D.11 Mockup detailu přihlášky v administrátorské sekci



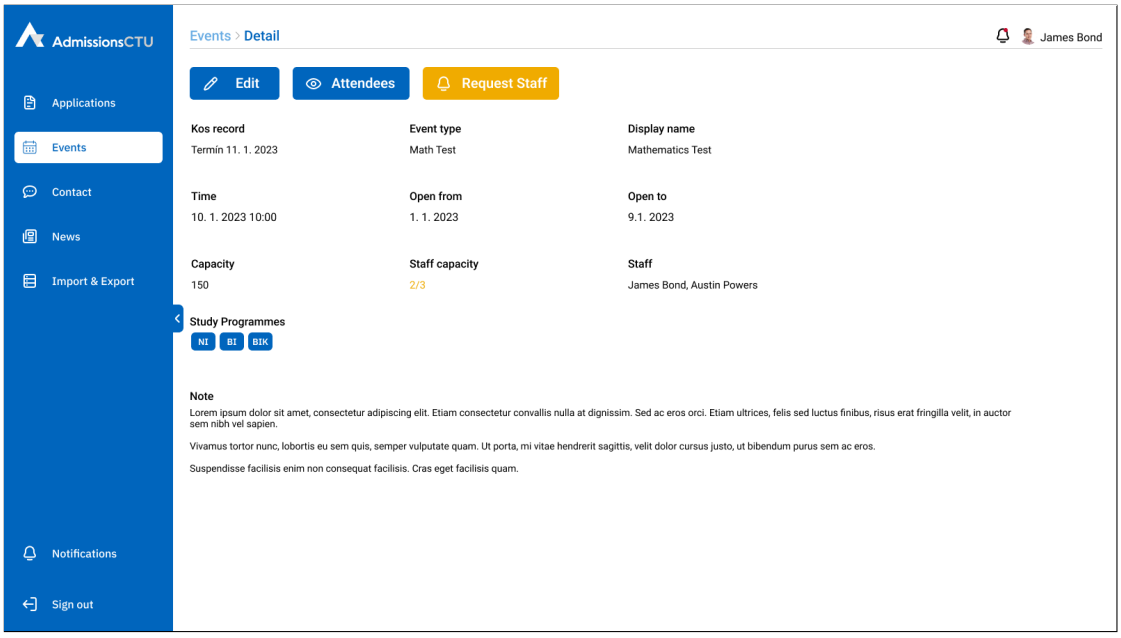
Obrázek D.12 Mockup editace přihlášky v administrátorské sekci



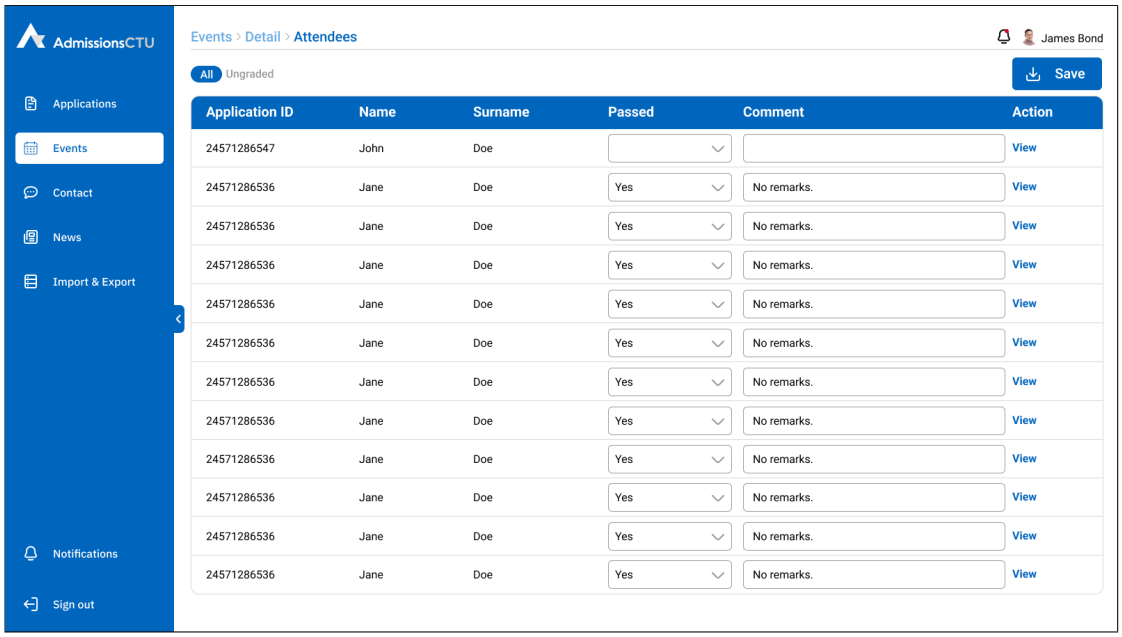
Obrázek D.13 Mockup přehledu událostí v administrátorské sekci



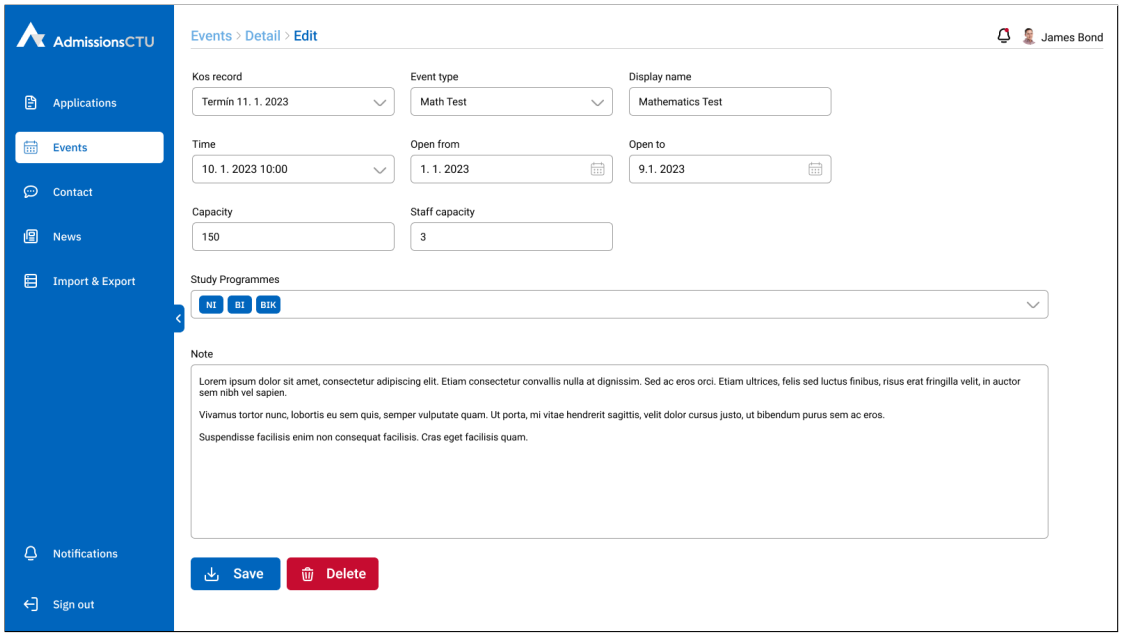
Obrázek D.14 Mockup vytvoření události v administrátorské sekci



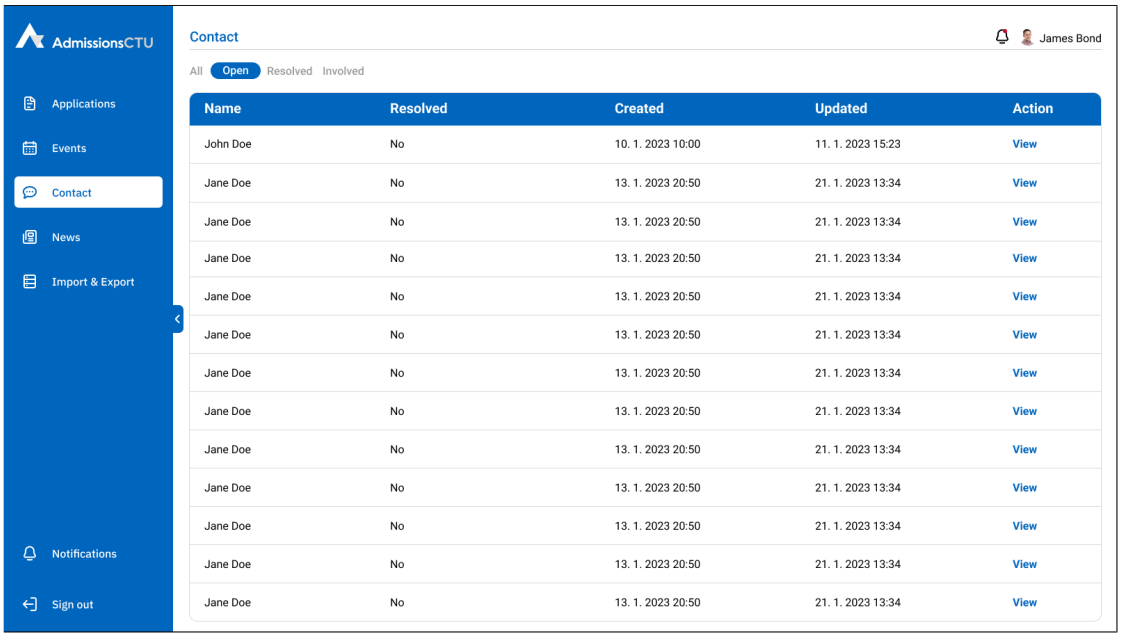
Obrázek D.15 Mockup detailu události v administrátorské sekci



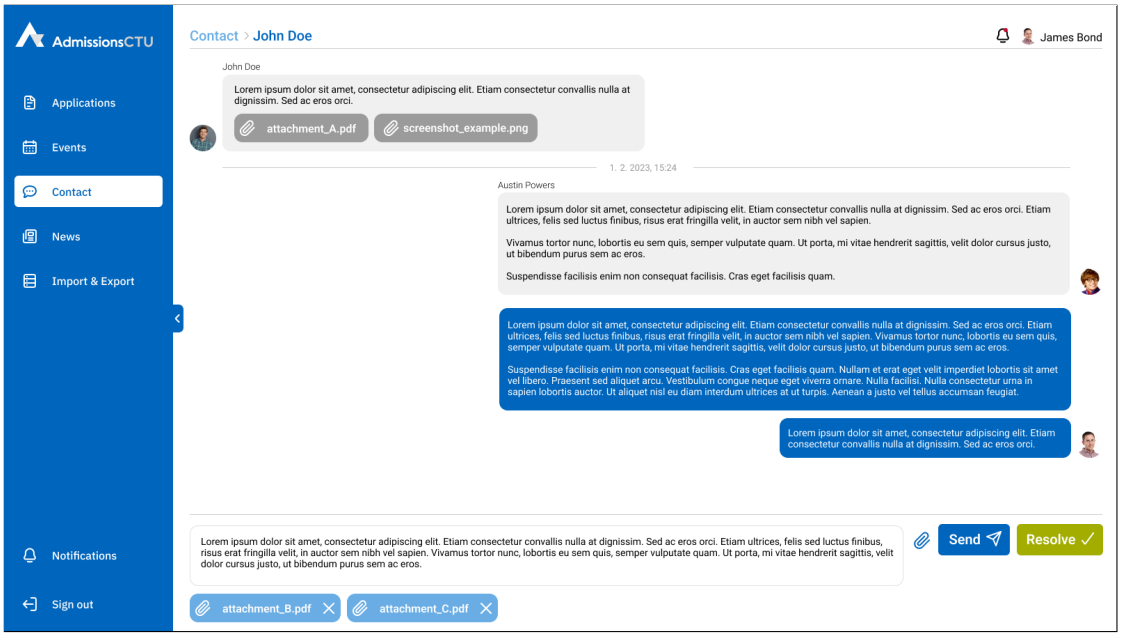
Obrázek D.16 Mockup detailu události sekce bodování v administrátorské sekci



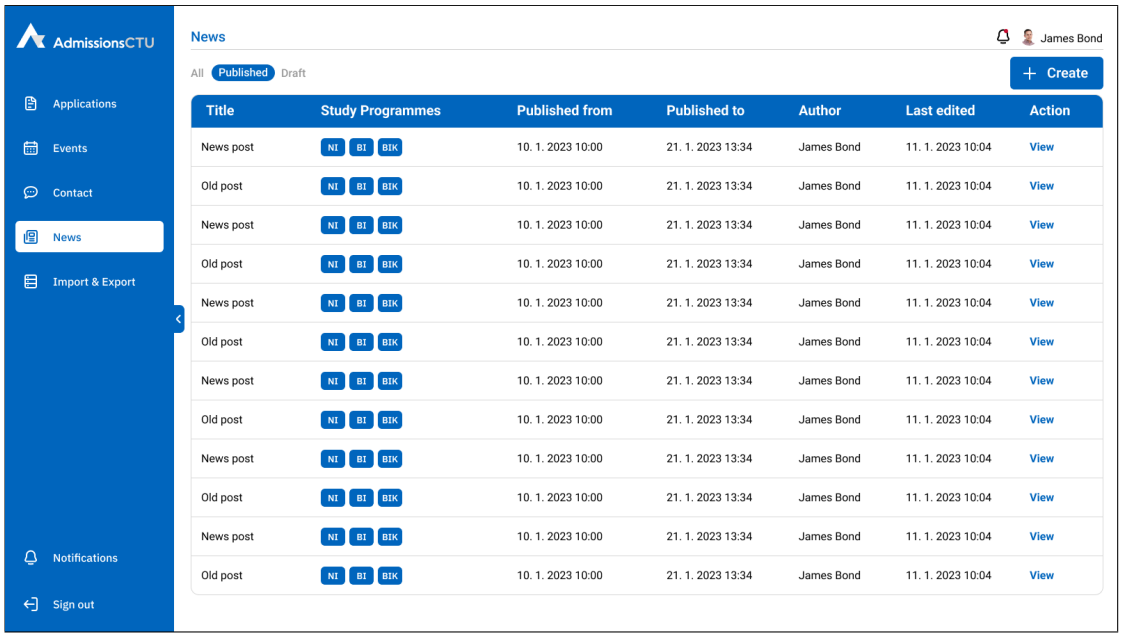
Obrázek D.17 Mockup editace události v administrátorské sekci



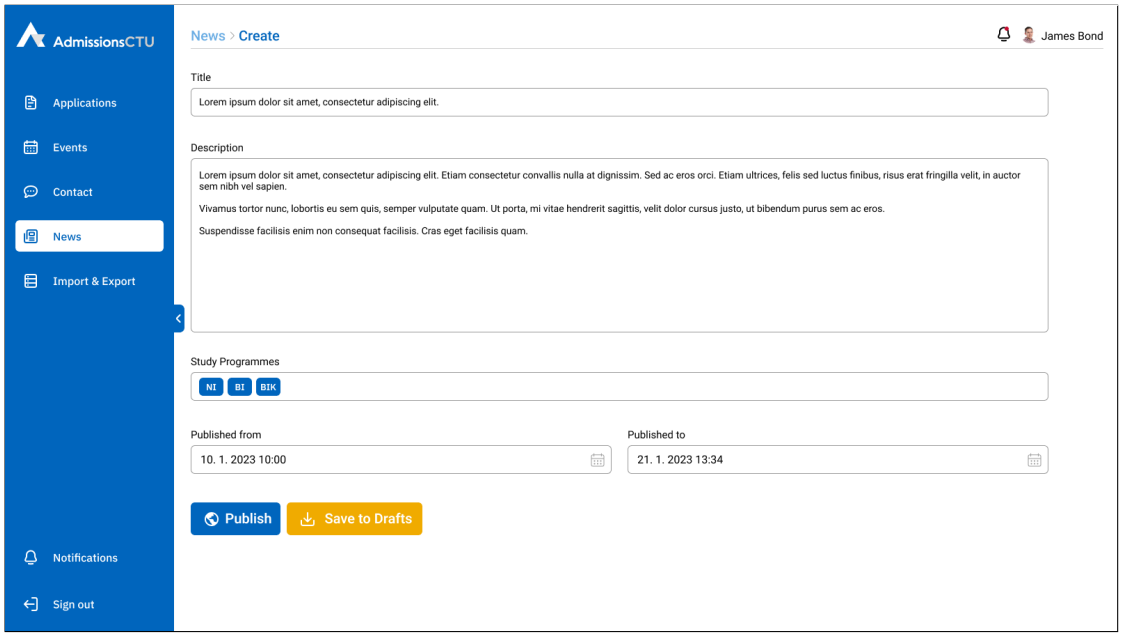
Obrázek D.18 Mockup přehledu konverzací v administrátorské sekci



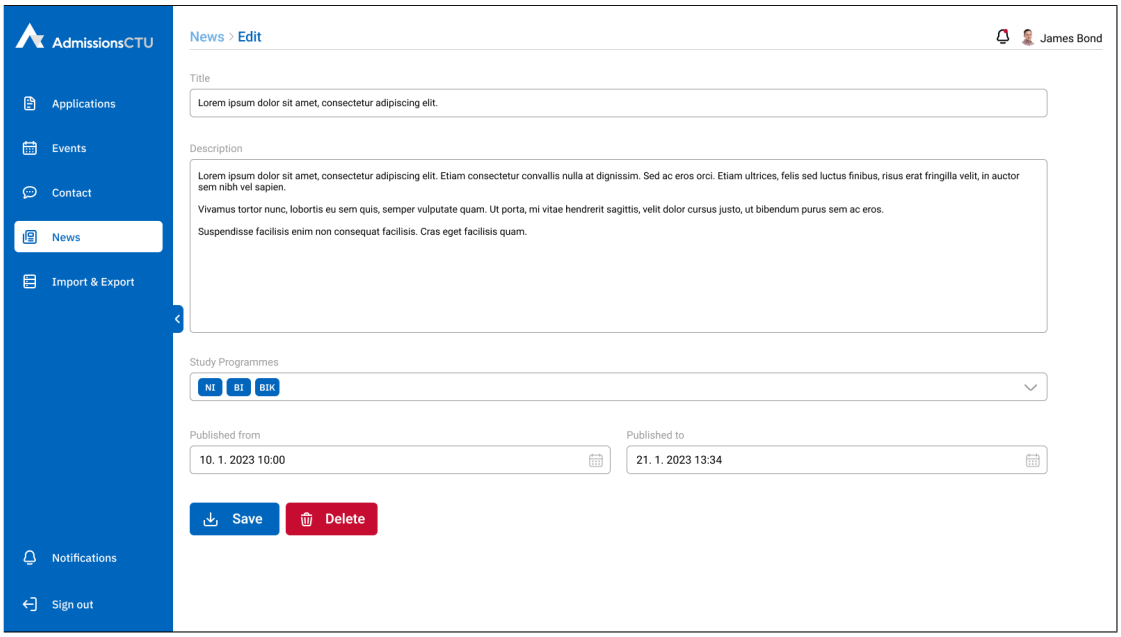
Obrázek D.19 Mockup detailu konverzace v administrátorské sekci



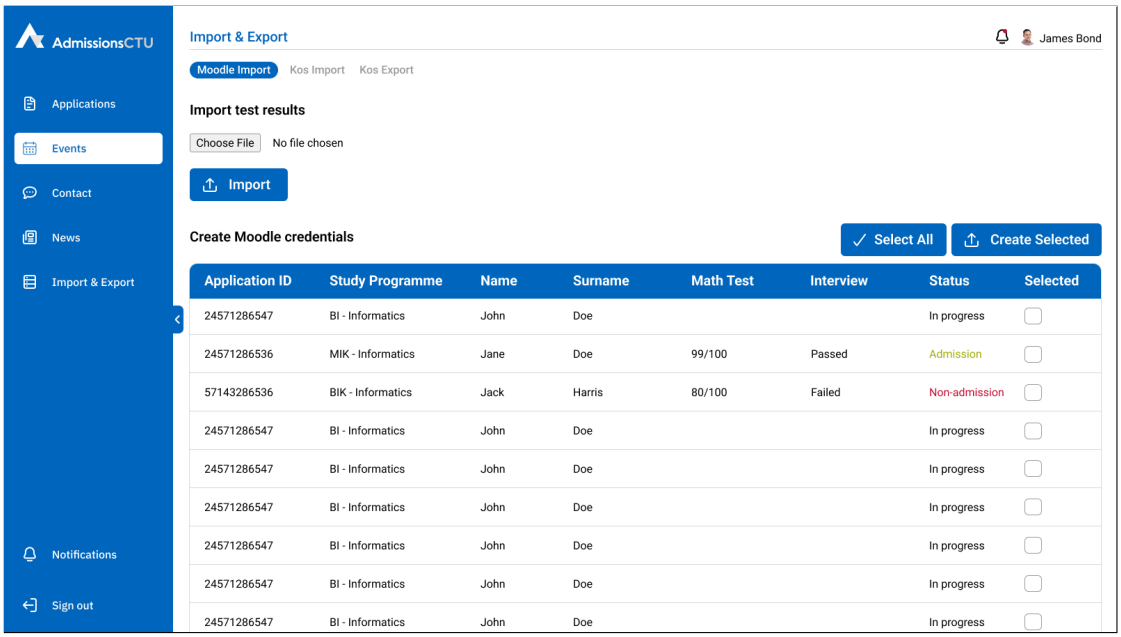
Obrázek D.20 Mockup přehledu aktualit v administrátorské sekci



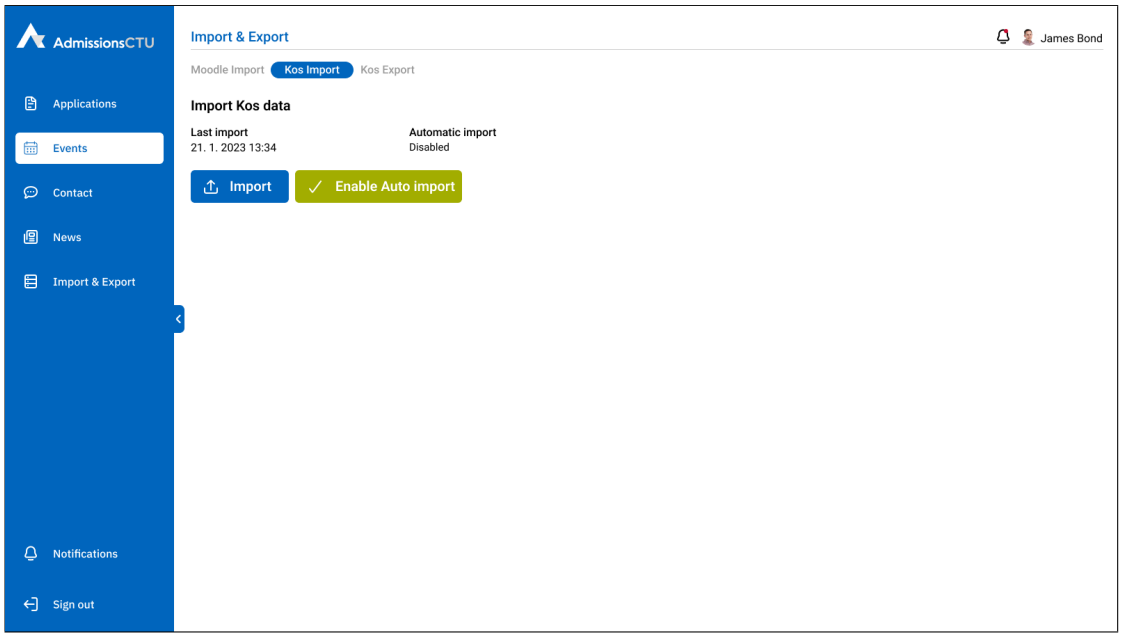
Obrázek D.21 Mockup vytvoření aktualit v administrátorské sekci



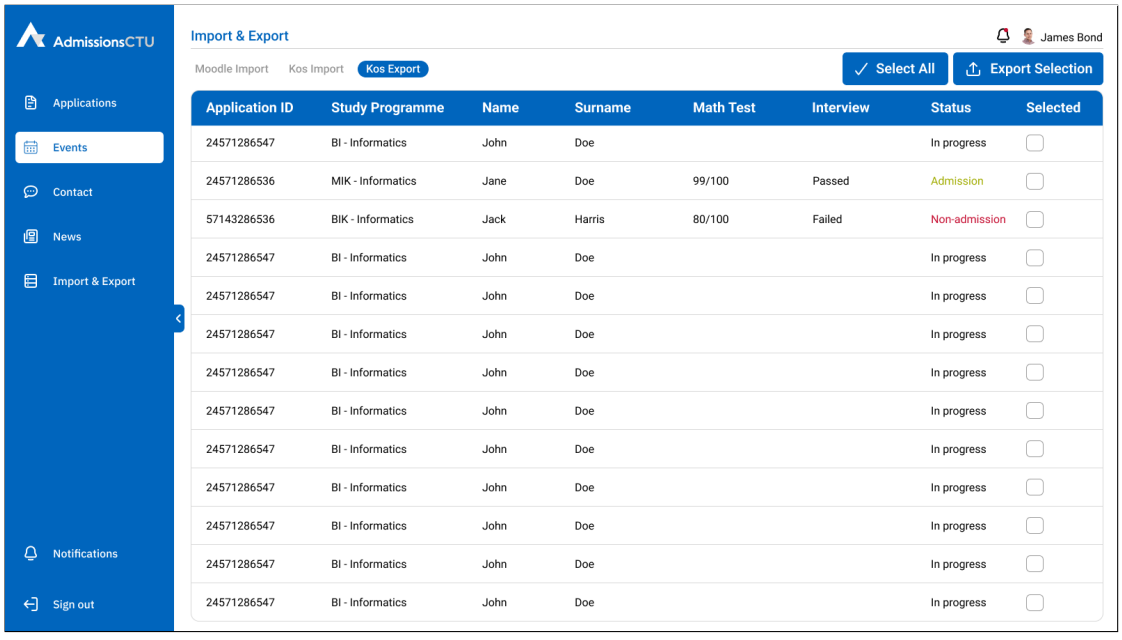
Obrázek D.22 Mockup detailu aktuality v administrátorské sekci



Obrázek D.23 Mockup import výsledků z Moodlu v administrátorské sekci



Obrázek D.24 Mockup import dat z KOS v administrátorské sekci



Obrázek D.25 Mockup export do KOS v administrátorské sekci

Příloha E

Podklady k uživatelskému testování

E.1 Screeningové otázky

1. Obecné otázky
   a. Jaký je váš věk?
   b. Jaké je vaše nejvyšší dosažené vzdělání?
   c. Máte jakékoliv zkušenosti s používáním webových aplikací nebo online systémů pro přijímacím řízení?
   d. Jaká je vaše znalost anglického jazyka?
2. Profese
   a. Jaká je vaše současná profese nebo pracovní role? (I student je považováno za profesi)
   b. Jak dlouho jste pracoval/a v této roli?
   c. Máte jakoukoliv zkušenosti s administrativní prací?
3. Znalost práce s počítačem
   a. Jak často používáte webové aplikace nebo online platformy ve své práci nebo osobním životě?
   b. Jak hodnotíte své dovednosti v práci s počítačem a webovými aplikacemi (např. začátečník, mírně pokročilý, pokročilý)?
   c. Jaké webové aplikace nebo online platformy používáte nejčastěji?
   d. Máte zkušenosti s používáním aplikací pro správu času, úkolů nebo projektů?
   e. Jaké typy zařízení používáte pro přístup k webovým aplikacím (např. počítač, tablet, smartphone)?
4. Znalost domény
   a. Jak dobře rozumíte procesu přijímacího řízení na vysokých školách?
   b. Máte zkušenosti s konkrétními kroky přijímacího řízení, jako je podání přihlášky, výběr oboru, přijímací zkoušky nebo rozhovory?
   c. Dokážete si představit jak probíhá přijímací řízení?

E.2 Popis účastníků

1. Uchazeči

   a. **Jméno:** JXXXX SXXXXXX
   Věk: 18 let
   Pohlaví: Muž
   Vzdělání: Středoškolské s maturitou
   Profese: Budoucí student, plánuje podat přihlášku na vysokou školu
   Zkušenosti s webovými aplikacemi: Používá sociální sítě, hledá informace o vysokých školách a přihlašovacích podmínkách online.

   b. **Jméno:** KXXXX VXXXXXX
   Věk: 20 let
   Pohlaví: Žena
   Vzdělání: Aktuálně studuje na vysoké škole, první rok bakalářského studia
   Profese: Studentka
   Zkušenosti s webovými aplikacemi: Používá univerzitní portál, e-learningové nástroje a e-mail.

   c. **Jméno:** TXXXX HXXXX
   Věk: 23 let
   Pohlaví: Muž
   Vzdělání: Bakalářský titul v oboru informatiky
   Profese: Absolvent vysoké školy, plánuje podat přihlášku na magisterské studium
   Zkušenosti s webovými aplikacemi: Aktivně používá různé online platformy pro vzdělávání a komunikaci, má základní znalosti o přijímacím řízení a požadavcích.

2. Administrátoři

   a. **Jméno:** MXXXXX NXXXX
   Věk: 42 let
   Pohlaví: Muž
   Vzdělání: Magisterský titul v oboru pedagogiky
   Profese: Administrátor ve společnosti poskytující vzdělávací služby
   Zkušenosti s webovými aplikacemi: Používá interní systém pro správu firemních procesů.

   b. **Jméno:** EXX NXXXXXX
   Věk: 29 let
   Pohlaví: Žena
   Vzdělání: Bakalářský titul v oboru marketingu
   Profese: Projektová manažerka v marketingové agentuře
   Zkušenosti s webovými aplikacemi: Projektová manažerka v marketingové agentuře s 7 lety zkušeností Zkušenosti s webovými aplikacemi: Používá aplikace pro správu projektů, komunikaci se zákazníky a kolegy, sledování marketingových kampaní a analýzu dat.

E.3 Testovací scénáře

UTS1	Zobrazení informací o přihlášce
Předpoklady:	Moderátor přihlásí účastníka na účet uchazeče se dvěma přihláškami.
Kontext:	Jste uchazeč, který podal na ČVUT FIT dvě přihlášky. Jedna přihláška je na prezenční studium a druhá na kombinované studium. Chcete zjistit status přihlášky kombinovaného studia.
Úspěch:	Zjištění statusu přihlášky kombinovaného studia.
Postup:	1. V aktuální chvíli se nacházíte v domovské stránce aplikace, přejděte do detailu přihlášky. 2. Pokud vidíte, že se zobrazily informace spojené s přihláškou na prezenční studium, přepněte se na přihlášku na kombinované studium. 3. Přečtěte si údaje o přihlášce na kombinované studium.

UTS2	**Přihlášení na termín z Matematiky**
Předpoklady:	Moderátor přihlásí účastníka na účet uchazeče, který není přihlášen na žádnou událost.
Kontext:	Jste v počáteční fázi přijímacího řízení a vaším prvním krokem je absolvování testu z matematiky. Proto se potřebujete přihlásit na termín.
Úspěch:	Účet uchazeče je přihlášen na správný termín.
Postup:	1. V aktuální chvíli se nacházíte v domovské stránce aplikace, přejděte do nabídky událostí. 2. Najděte termín testu z matematiky s datem 23. 6. 2023 a přihlaste se na něj. 3. Zjistili jste, že toto datum Vám nevyhovuje, a proto se chcete z něj odhlásit. 4. Místo toho se přihlaste na termín testu z matematiky s datem 25. 6. 2023.

UTS3	**Označení notifikací jako přečtené**
Předpoklady:	Moderátor přihlásí účastníka na účet uchazeče, který obsahuje přečtené i nepřečtené notifikace.
Kontext:	Po delší době jste se přihlásil na Váš účet a chcete se podívat na historii notifikací, abyste měl přehled o událostech, které nastaly.
Úspěch:	Všechny notifikace jsou označené jako přečtené.
Postup:	1. V aktuální chvíli se nacházíte v domovské stránce aplikace, nalezněte historii notifikací. 2. Zobrazte si přehled nových notifikací. 3. Najděte notifikaci o nové zprávě v konverzaci a označte ji jako přečtenou. 4. Chcete si zobrazit také kompletní historii notifikací, kde jsou vidět i již přečtené notifikace.

UTS4	Zobrazení výsledku z pohovoru
Předpoklady:	Moderátor přihlásí účastníka na účet uchazeče, který má udělenou známku z pohovoru.
Kontext:	Nedávno jste se v rámci přijímacího řízení účastnili pohovoru a právě Vám přišla notifikace o udělení známky. Chcete zjistit svoji známku z pohovoru.
Úspěch:	Zjištění hodnocení z pohovoru včetně komentářů tazatele.
Postup:	1. V aktuální chvíli se nacházíte v domovské stránce aplikace, nalezněte sekci sekci s hodnocením. 2. Přečtěte, jaký výsledek z pohovoru jste dostali. 3. Zjistěte jaké měl tazatel komentáře k pohovoru.

UTS5	Poslání dotazu na studijní oddělení
Předpoklady:	Moderátor přihlásí účastníka na účet uchazeče.
Kontext:	Uprostřed přijímacího řízení jste zjistili, že máte chybu v údajích ve Vaší přihlášce. Proto chcete kontaktovat studijní oddělení a požádat o opravu.
Úspěch:	V konverzaci je zaslána nová zpráva.
Postup:	1. V aktuální chvíli se nacházíte v domovské stránce aplikace, přejděte do detailu přihlášky. 2. Zkontrolujte, že jste přihlášen na studijní program BIK-Informatika. 3. Pokud je studijní program v přihlášce odlišný, přejděte do detailu konverzace. 4. Napište jakoukoliv žádost o opravu a zprávu pošlete.

UTS6	**Zobrazení uživatelského profilu**
Předpoklady:	Moderátor přihlásí účastníka na účet uchazeče a připraví přístup k emailové adrese účtu.
Kontext:	Jako uchazeč si chcete zkontrolovat údaje o Vašem účtu a provést případné změny.
Úspěch:	Zjištění kontaktních údajů a úspěšná změna hesla.
Postup:	1. V aktuální chvíli se nacházíte v domovské stránce aplikace, nalezněte Váš uživatelský profil. 2. V uživatelském profilu najděte svoji trvalou i přechodnou adresu a přečtěte ji. 3. Najednou jste si uvědomil, že se Vám nevybavuje aktuální heslo k tomuto účtu, a tak chcete provést z uživatelského profilu obnovu hesla. 4. Vyplňte emailovou adresu, kterou Vám sdělí moderátor a potvrďte formulář. 5. Přihlaste se nyní na emailovou adresu, přečtěte si instrukce a rozklikněte odkaz v přijatém mailu o obnově hesla. 6. Zvolte libovolné heslo a potvrďte změnu.

ATS1	**Zobrazení informací o přihlášce konkrétního uchazeče**
Předpoklady:	Moderátor přihlásí účastníka na účet administrátora.
Kontext:	Jako administrátor si chcete zjistit informace přihlášky uchazeče.
Úspěch:	Zjištění informací o přihlášce a kontaktních údajů uchazeče Jana Nováka.
Postup:	1. V aktuální chvíli se nacházíte v přehledu přihlášek v administrátorské sekci. Nalezněte v přehledu přihlášek přihlášku Jana Nováka a přečtěte informace o přihlášce. 2. Pokud v přehledu přihlášek nevidíte kontaktní údaje, přejděte do detailu přihlášky Jana Nováka. 3. Přečtěte všechny údaje spojené s přihláškou uchazeče, včetně hodnocení z událostí. 4. Najděte kontaktní adresu uchazeče a ověřte, zda-li je stejná jako trvalá adresa.

ATS2	**Import dat ze systému KOS**
Předpoklady:	Moderátor přihlásí účastníka na účet administrátora.
Kontext:	Aplikace synchronizuje data o přihláškách, událostech a studijní programech vůči univerzitní databázi. Zároveň dlouho neproběhla žádná aktualizace, a proto chcete aktualizovat data.
Úspěch:	Aplikace přijala požadavek na import ze systému KOS.
Postup:	1. V aktuální chvíli se nacházíte v přehledu přihlášek v administrátorské sekci. V seznamu přihlášek se podívejte na přihlášky a zapamatujte si kolik jich bylo. 2. Nalezněte v administrátorské části sekci pro import dat ze systému KOS. 3. Pokud jste nalezl sekci pro import, klikněte na tlačítko importovat. 4. Zkontrolujte zda-li Vám přišla notifikace o dokončeném importu. 5. Vraťte se do přehledu přihlášek a porovnejte, zda-li přibyly nové přihlášky.

ATS3	**Vytvoření události**
Předpoklady:	Moderátor přihlásí účastníka na účet administrátora.
Kontext:	Jako administrátor chcete v rámci přijímacího řízení naplánovat nový termín pro test z matematiky.
Úspěch:	V aplikaci je zaevidován nový termín z matematiky.
Postup:	1. V aktuální chvíli se nacházíte v přehledu přihlášek v administrátorské sekci. Přejděte do přehledu událostí a prohlédněte si termíny testů z matematiky. 2. Nalezněte formulář pro vytvoření nové události. 3. Pokud jste nalezl sekci pro import, klikněte na tlačítko importovat. 4. Tato událost se musí spárovat s KOS záznamem, který má identifikátor s číslem 2, a musí být typu test z matematiky. Dále nastavte událost pro studijní programu BIK-Informatica a BI-Informatica. Ostatní údaje můžete vyplnit libovolně. 5. Potvrďte vytvoření události a zkontrolujte v přehledu událostí, zda-li byla vytvořena.

ATS4	**Ohodnocení události**
Předpoklady:	Moderátor přihlásí účastníka na účet administrátora.
Kontext:	Jako administrátor chcete po proběhnutí pohovoru udělit uchazečům hondocení.
Úspěch:	Všichni uchazeči zvoleného termínu obdrželi hodnocení.
Postup:	1. V aktuální chvíli se nacházíte v přehledu přihlášek v administrátorské sekci. Přejděte do přehledu událostí a nalezněte termín pohovoru s datem 27. 5. 2023 2. Přejděte do detailu události a najděte seznam všech uchazečů. 3. V seznamu uchazečů jsou evidovány tři uchazeči. Uchazeči Jan Novák udělte úspěšný absolvování pohovoru a do komentáře napište ”Výborný výkon”. Ostatním uchazečům udělte neúspěch, komentáře nemusíte psát. 4. Uložte změny a zkontrolujte v detailu přihlášky uchazečů, zda-li se změny propsaly.

ATS5	**Vyřešení konverzace**
Předpoklady:	Moderátor přihlásí účastníka na účet administrátora.
Kontext:	Jako administrátor chcete odpovědět na dotazy uchazečů a pomoct jim v případě problému spojených s přijímacím řízením.
Úspěch:	Všechny konverzace jsou ve stavu uzavřené.
Postup:	1. V aktuální chvíli se nacházíte v přehledu přihlášek v administrátorské sekci. Přejděte do přehledu konverzací a zobrazte si všechny otevřené konverzace. 2. Vejděte do každé konverzace a zodpovězte na otázky uchazečů. Odpovědi byste měli všechny za pomocí procházení aplikace. 3. Po odpovězení otázek každou konverzaci uzavřete. 4. Zkontrolujte, že v aplikaci se již nenachází otevřené konverzace.

ATS6	**Vytvoření aktuality**
Předpoklady:	Moderátor přihlásí účastníka na účet administrátora.
Kontext:	Jako administrátor chcete globálně oznámit uchazečům novinky spojené s přijímacím řízením.
Úspěch:	V aplikaci je zavevidována nová aktualita se správnými údaji.
Postup:	1. V aktuální chvíli se nacházíte v přehledu přihlášek v administrátorské sekci. Přejděte do přehledu aktualit a najděte formulář pro vytvoření aktuality. 2. Libovolně vyplntě název a popisek aktulity, studijní program vyberte BIK-Informatica a nastavte viditelnost aktuality na celý příští týden. 3. Potvrďte vytvoření aktuality. 4. Zkontrolujte, zda se aktualita v systému vytvořila a zkontrolujte detail aktuality, zda-li jsou všechny údaje v pořadku.

ATS7	**Export dat**
Předpoklady:	Moderátor přihlásí účastníka na účet administrátora.
Kontext:	Jako administrátor chcete vyexportovat data a výsledky o přihláškách pro další zpracování.
Úspěch:	Ze systému byly vyexportovány správné přihlášky a byl vygenerován CSV soubor se správnými daty.
Postup:	1. V aktuální chvíli se nacházíte v přehledu přihlášek v administrátorské sekci. Nalezněte v aplikaci sekci pro export dat. 2. Ve výběru přihlášek zvolte přihlášku Jana Nováka a Jana Nohy. 3. Potvrďte výběr a otevřete vygenerované CSV. 4. Prohlédněte si soubor CSV a zkontrolujte, že všechny data jsou stejná jako v aplikaci.

Bibliografie

1. GUIZZARDI, Giancarlo. *Ontological Foundations for Structural Conceptual Models*. Enschede, The Netherlands: Telematica Instituut, 2005. Telematica Institute Fundamental Research Series, č. 15. Dostupné také z: http://www.researchgate.net/publication/215697579_Ontological_Foundations_for_Structural_Conceptual_Models.
2. FRONTCZAK, Agata; KOLUSZKOWSKA, Magda. *Competitive Analysis: How to Do the Market Research in Software Development Projects* [online]. 2021-12. [cit. 2023-03-28]. Dostupné z: https://fivedottwelve.com/blog/competitive-analysis-how-to-do-the-market-research-in-software-development-projects/.
3. MORROW, Susan. *Third-party authentication (OAUTH): Good or bad for security?* [online]. 2022-06. [cit. 2023-03-28]. Dostupné z: https://resources.infosecinstitute.com/topic/third-party-authentication-oauth-good-or-bad-for-security/.
4. MANNION, Mike; KEEPENCE, Barry. SMART requirements. *ACM SIGSOFT Software Engineering Notes*. 1995, roč. 20, č. 2, s. 42–47. ISSN 0163-5948. Dostupné z DOI: 10.1145/224155.224157.
5. MLADENOVA, Tsvetelina. Software Quality Metrics – Research, Analysis and Recommendation. In: *2020 International Conference Automatics and Informatics (ICAI)*. 2020, s. 1–5. Dostupné z DOI: 10.1109/ICAI50593.2020.9311361.
6. HUDAIB, Amjad; MASADEH, Raja; QASEM, Mais; ALZAQEBAH, Abdullah. Requirements Prioritization Techniques Comparison. *Modern Applied Science*. 2018, roč. 12. Dostupné z DOI: 10.5539/mas.v12n2p62.
7. INFLECTRA. *Use cases & scenarios: What they are & more* [online]. 2023-01. [cit. 2023-03-28]. Dostupné z: https://www.inflectra.com/Ideas/Topic/Use-Cases.aspx.
8. OCHODEK, Mirosław; NAWROCKI, Jerzy; KWARCIAK, K. Simplifying effort estimation based on Use Case Points. *Information & Software Technology*. 2011, roč. 53, s. 200–213. Dostupné z DOI: 10.1016/j.infsof.2010.10.005.
9. DEV, Logan. *The most demanded frontend frameworks in 2022* [online]. 2023-01. [cit. 2023-04-01]. Dostupné z: https://www.devjobsscanner.com/blog/the-most-demanded-frontend-frameworks-in-2022/.
10. ANGULAR. *Angular Versioning and Releases* [online]. 2021-08. [cit. 2023-04-02]. Dostupné z: https://angular.io/guide/releases.
11. FACEBOOK. *React Releases* [online]. 2022-06. [cit. 2023-04-02]. Dostupné z: https://github.com/facebook/react/releases.
12. VUE.JS. *Releases - Vue.js* [online]. 2023-02. [cit. 2023-04-02]. Dostupné z: https://vuejs.org/about/releases.html.

13. SHAPEL, Maryia. *Angular vs React vs Vue: The Final Verdict* [online]. SaM Solutions, 2023-03 [cit. 2023-04-04]. Dostupné z: https://www.sam-solutions.com/blog/angular-vs-react-vs-vue/.
14. SUDHAN, Harihara. *Angular or react or vue? how to make the right choice for your business in 2023?* [online]. F22 Labs, 2022-10 [cit. 2023-04-08]. Dostupné z: https://www.f22labs.com/blogs/angular-vs-react-vs-vue-in-2023.
15. RAMAKRISHNAN, Apoorva. *(why) is Jquery Dead?* [online]. Medium, 2021-08 [cit. 2023-04-10]. Dostupné z: https://medium.com/@apoorva72900/why-is-jquery-dead-b41dba56491e.
16. FACEBOOK. *Meet the team* [online]. React, 2023 [cit. 2023-04-11]. Dostupné z: https://react.dev/community/team.
17. ANGULAR. *Angular contributors* [online]. Google, 2023 [cit. 2023-04-11]. Dostupné z: https://angular.io/about?group=Angular.
18. VUE.JS. *Meet the team* [online]. Vue.js, 2023 [cit. 2023-04-11]. Dostupné z: https://vuejs.org/about/team.html.
19. ANGULAR. *Angular docs* [online]. Google, 2023 [cit. 2023-04-18]. Dostupné z: https://angular.io/docs.
20. FACEBOOK. *React docs* [online]. React, 2023 [cit. 2023-04-18]. Dostupné z: https://react.dev/learn.
21. VUE.JS. *Vue.js docs* [online]. Vue.js, 2023 [cit. 2023-04-13]. Dostupné z: https://vuejs.org/guide/introduction.html.
22. DULANGA, Chameera. *Incremental vs virtual dom* [online]. Bits a Pieces, 2020-12 [cit. 2023-04-15]. Dostupné z: https://blog.bitsrc.io/incremental-vs-virtual-dom-eb7157e43dca.
23. RAMACHANDRAN, Atul. *Guide to Custom React Renderers. How to build your own renderer from scratch?* [online]. 2021-10. [cit. 2023-04-18]. Dostupné z: https://blog.atulr.com/react-custom-renderer-1/.
24. KUMAR, Tejas. *Fluent react: Build fast, performant, and intuitive web applications*. O'REILLY MEDIA, 2024. ISBN 9781098138714.
25. SHARMA, Tushar. *React fiber architecture - an overview* [online]. [cit. 2023-04-18]. Dostupné z: https://tusharf5.com/posts/react-fiber-overview/.
26. IMOH, Ifeoma. *Understand how rendering works in react* [online]. Telerik, 2023-01 [cit. 2023-04-18]. Dostupné z: https://www.telerik.com/blogs/understand-how-rendering-works-react.
27. KAMASALA, Viswanath. *The eleven defining characteristics of modern software architecture* [online]. HackerNoon, 2020-10 [cit. 2023-04-20]. Dostupné z: https://hackernoon.com/the-eleven-defining-characteristics-of-modern-software-architecture-o8113ehc.
28. PASQUALE, Dario Di. *Introduction to domain-driven design (DDD)* [online]. Medium, 2023-01 [cit. 2023-04-21]. Dostupné z: https://dariodip.medium.com/introduction-to-domain-driven-design-ddd-24a62cb6472d.
29. NIELSEN, Jakob. *Usability engineering*. San Francisco, Calif.: Morgan Kaufmann Publishers, 1994. ISBN 9780125184069.
30. RETTIG, Marc. Prototyping for Tiny Fingers. *Commun. ACM*. 1994, roč. 37, č. 4, s. 21–27. ISSN 0001-0782. Dostupné z DOI: 10.1145/175276.175288.

31. RAJPUT, Asha. *Minimalism in UI/UX Design: All you need to know* [online]. 300Mind, 2022-11 [cit. 2023-04-23]. Dostupné z: https://300mind.studio/blog/minimalist-ui-ux-design/.

32. CHALUPSKÁ, Ilona. *Logo A Grafický Manuál* [online]. ČVUT v Praze [cit. 2023-04-24]. Dostupné z: https://www.cvut.cz/logo-a-graficky-manual.

33. BISIAKOWSKI, Dominik. *Why we use vite instead of create react app* [online]. 2023-04. [cit. 2023-04-26]. Dostupné z: https://makimo.pl/blog/why-we-use-vite-instead-of-create-react-app/.

34. BOYLE, Eamonn. *Static types vs dynamic types: Stop fighting and make my life easier already* [online]. The Startup, 2020-12 [cit. 2023-04-25]. Dostupné z: https://medium.com/swlh/static-types-vs-dynamic-types-stop-fighting-and-make-my-life-easier-already-73f58bfe7d0.

35. BLOOM, Yotam. *Why we chose chakra-ui for our design system* [online]. Melio's R&D blog, 2023-03 [cit. 2023-04-26]. Dostupné z: https://medium.com/meliopayments/why-we-chose-chakra-ui-for-our-design-system-part-1-a9f988127dab.

36. SHAHIN, Mojtaba; BABAR, Muhammad Ali; ZHU, Liming. Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices. *IEEE access*. 2017, roč. 5, s. 3909–3943.

37. GOMES, Ivo; MORGADO, Pedro; GOMES, Tiago; MOREIRA, Rodrigo. An overview on the static code analysis approach in software development. *Faculdade de Engenharia da Universidade do Porto, Portugal*. 2009.

38. KHORIKOV, Vladimir. *Unit Testing Principles, Practices, and Patterns*. Simon a Schuster, 2020. ISBN 9781617296277.

39. WACKER, Mike. *Just say no to more end-to-end tests* [online]. 2015-04. [cit. 2023-04-30]. Dostupné z: https://testing.googleblog.com/2015/04/just-say-no-to-more-end-to-end-tests.html.

40. LEE, Sung Heum. Usability testing for developing effective interactive multimedia software: Concepts, dimensions, and procedures. *Journal of Educational Technology & Society*. 1999, roč. 2, č. 2.

41. QUICKMARK. *Creating task scenarios that improves usability test results* [online]. Medium, 2016-12 [cit. 2023-04-30]. Dostupné z: https://weekdayhq.medium.com/creating-task-scenarios-that-improves-usability-test-results-eece56959d19.

Obsah přiloženého média

- `readme.txt` — stručný popis obsahu média
- `demo` — adresář s ukázkami aplikace
  - `auth.mp4` — video ukázka autentizační části
  - `user.mp4` — video ukázka uživatelské části
  - `admin.mp4` — video ukázka administrátorské části
- `src`
  - `impl` — zdrojové kódy implementace
  - `thesis` — zdrojová forma práce ve formátu LaTeX
  - `ui`
    - `lo-fi` — zdrojové soubory lo-fi návrhu
    - `hi-fi` — zdrojové soubory hi-fi návrhu
- `text`
  - `thesis.pdf` — text práce ve formátu PDF