VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

INFORMAČNÍ SYSTÉM PRO SPRÁVU STUDENTSKÝCH SPOLKŮ

INFORMATION SYSTEM FOR THE ADMINISTRATION OF STUDENT GUILD

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **MICHAL HALABICA**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. JAROSLAV DYTRYCH, Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Halabica Michal**
Program: Informační technologie
Název: **Informační systém pro správu studentských spolků**
Information system for the administration of student guild
Kategorie: Informační systémy

Zadání:

1. Seznamte se s jazykem C# a s vhodnými technologiemi pro tvorbu webových uživatelských rozhraní.
2. Prostudujte současné systémy pro správu členů, majetku, financí a pořádání akcí využívané Studentskou unií FIT a možnosti jejich aplikace pro jiné studentské spolky.
3. Navrhněte nový informační systém pro správu studentských spolků, který umožní správu členů a jejich rozdělení do pracovních skupin, evidenci majetku, správu financí a pořádání akcí. Systém umožní integraci vybraných služeb z G Suite a správu oprávnění na Discord serveru.
4. Implementujte navržené řešení.
5. Zhodnoťte dosažené výsledky a vytvořte stručný plakát prezentující výsledky práce.

Literatura:

- Dle doporučení vedoucího

Pro udělení zápočtu za první semestr je požadováno:

- Body 1, 2 a 3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Dytrych Jaroslav, Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 30. října 2020

Abstrakt

Tato práce se zabývá vytvořením centrálního informačního systému pro studentský spolek. Cílem této práce je vytvořit takový systém, který ulehčí každodenní práci ve studentském spolku. Vytvořené řešení vychází z požadavků Studentské Unie FIT VUT v Brně, které byly přizpůsobeny pro možnost využití dalšími studentskými spolky. Informační systém je implementován jako webová aplikace za pomoci platformy .NET s jazykem C# a Angular s jazykem TypeScript. Výsledné řešení bylo testováno členy již zmíněné studentské unie.

Abstract

This thesis deals with the creation of a central information system for student associations. The goal of this thesis is to create a system, that will ease everyday work in the student association. The created solution is based on requirements of the FIT BUT Student Union, which were adapted for the possibility of use by other student associations. The information system was implemented as web application using .NET technologies with C# language and Angular with TypeScript language. The final version was tested by members of the already mentioned student union.

Klíčová slova

studentský spolek, informační systém, .NET, C#, ASP.NET Core, Angular, TypeScript, PostgreSQL, JSON, WebSocket, JWT, Docker, Discord, Google Workspace

Keywords

student association, information system, .NET, C#, ASP.NET Core, Angular, TypeScript, PostgreSQL, JSON, WebSocket, JWT, Docker, Discord, Google Workspace

Citace

HALABICA, Michal. *Informační systém pro správu studentských spolků*. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jaroslav Dytrych, Ph.D.

Informační systém pro správu studentských spolků

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jaroslava Dytrycha, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Michal Halabica
16. května 2021

Poděkování

Tímto bych rád poděkoval Ing. Jaroslavu Dytrychovi za ochotné vedení, odbornou pomoc a rady při řešení této práce. Dále bych rád poděkoval členům Studentské Unie FIT VUT v Brně za poskytnutí dat při sběru požadavků.

Obsah

Kapitola 1

Úvod

Studentský spolek je sdružení studentů, které si klade za cíl nějakým způsobem vést kontakt se studenty na vysoké škole. Tato práce bude ve většině případů zmiňovat Studentskou Unii FIT VUT v Brně, jakožto spolek, ze kterého pochází myšlenka na vytvoření tohoto systému. Studentská Unie FIT VUT v Brně (dále jen Studentská Unie FIT, nebo SU FIT) je spolek studentů, kteří pomáhají ostatním studentům i fakultě. Zastupují a chrání zájmy studentů v orgánech fakulty, ale řeší také různé aktivity na FIT VUT jako provoz studentského klubu, organizování volnočasových i vzdělávacích akcí apod.

Tento systém si klade za cíl vytvořit centrální systém umožňující jednodušší fungování spolku. Mezi prioritní cíle systému patří například správa členů a jejich členění do jednotlivých skupin (týmů), evidence majetku, evidence financí a jejich pohybů apod.

Stávající řešení vedení Studentské Unie FIT vede ke komplikované dostupnosti informací jak pro stávající, tak i pro nové členy spolku. Aby člen mohl vyhledat potřebnou informaci, tak musel kontaktovat vedení spolku, aby mu někdo sdělil, kde vůbec informaci nalezne. Aktuálně totiž mohou členové nalézt data na discích Google (osobních i unijních), nástěnkách Trello, nebo v osobních konverzacích (někdy i soukromých).

Práce je členěna do úvodu, 5ti kapitol, kde každá kapitola obsahuje upřesňující podkapitoly, a závěru práce. Kapitola 2 se věnuje analýze požadavků od jednotlivých členů SU FIT. Na tuto kapitolu navazuje kapitola 3, která se věnuje návrhu informačního systému primárně pomocí diagramů případů užití a ER (Entity-Relationship) diagramů. Kapitola 4 se zaměřuje na implementaci systému vycházející z popsaného návrhu. Kapitoly 6 a 5 popisují postupy nasazení systému a testování systému. Celou práci pak uzavírá závěrečná kapitola shrnující hlavní myšlenky práce a zhodnocení dosažených výsledků.

Kapitola 2

Analýza požadavků

Před vytvořením samotného návrhu byla provedena analýza požadavků. Sběr a analýza požadavků probíhala se členy SU FIT a byla rozdělena do dvou fází. První fáze probíhala konzultacemi (elektronicky i osobním setkáním) s vedením spolku. Z těchto konzultací byl zjištěn aktuální stav a požadavky na novou aplikaci. Druhá fáze probíhala elektronickou formou a byly do ní zapojeni všichni členové spolku. Elektronická forma sběru požadavků probíhala kombinací sdíleného dokumentu Google a konverzace v kanálu na platformě Discord.

Z tohoto sběru bylo vytvořeno 7 nejnutnějších funkcí celého systému, které jsou popsány v následujících podkapitolách. Současně z těchto sběrů byl zjištěn aktuální stav, který se jevil poměrně nedostatečný a chaotický. SU FIT se strukturuje na předsedu, místopředsedu a jednotlivé skupiny (týmy). Současný stav ukazuje, že každý tým si udržuje vlastní systém uchovávání znalostí, zkušeností a celého interního spravování týmu. Jedná se o sdílené dokumenty Google, mnohdy uložené na osobních účtech, nástěnky Trello, nebo soukromé konverzace.

2.1 Správa členů a týmů

Spolek jako takový potřebuje centrální evidenci svých členů, jejich rozdělení do týmů a možnost správy oprávněnými osobami.

Ke každému členovi by se měly evidovat základní informace, které člen poskytl v přihlášce do spolku nebo byly získány od člena po jeho přijetí do spolku a ke kterým současně udělil souhlas se zpracováním osobních údajů. Současně by měl systém evidovat také postavení člena ve spolku. Tyto informace by měly být evidované i po odchodu člena ze spolku, a to na dobu neurčitou, nebo dokud člen spolku nepožádá o výmaz svých údajů z databáze členů spolku.

Ke každému týmu by se mělo evidovat, o jaký tým se jedná, co daný tým dělá, kdo je vedoucí týmu a kdo jsou členové týmu. Vedoucí týmu by měl být schopen přidávat členy spolku do svého týmu, předseda (nebo jiná jím pověřená osoba) by měl být schopen přidat člena do jakéhokoliv týmu. Současně se také může stát, že nějaký tým nebude mít vedoucího. Nového vedoucího by měl určovat předseda, nebo jiná oprávněná osoba určená předsedou.

Přístup k informacím o členech spolku a týmech by měl být ke čtení dostupný všem členům spolku. Člen spolku by pak měl mít možnost měnit většinu údajů sám sobě, pokud by měl vyšší oprávnění pro správu členů a týmů, tak by mohl měnit všechny údaje. Člen by

si bez vyššího oprávnění neměl být schopen měnit datum přijetí do spolku, datum odchodu ze spolku, postavení, datum konce zkušební doby a oprávnění.

Ve spolku je také třeba evidovat pohovory, kterými člen spolku prošel. Mělo by ze systému být patrné, kdy a s kým takovým pohovorem člen prošel. Ten, kdo pohovor se členem spolku vedl, by měl být schopen zadat k pohovoru nějakou poznámku. Člen spolku může za svoji dobu působení projít několika pohovory, ale také žádným. Přístupnost takových informací by měla být pouze předsedovi a jím pověřeným osobám. Běžný člen by se k těmto informacím o pohovorech (ani svých, ani ostatních) neměl nikdy dostat.

Každý člen, který se zapíše do spolku, jako je Studentská Unie FIT, musí projít tzv. zkušební dobou. Během této zkušební doby by si měl nový člen spolku „osahat“ fungování spolku a vybrat si tým, případně týmy, kde bude chtít působit. V porovnání s řádným členem nemá člen ve zkušební době žádné výraznější omezení. Jediné omezení, které mu plyne ze statutu zkušební doby je, že se nemůže zapojit do hlasování na členské schůzi (zasedání). Systém by měl alespoň formou jednoduché informace o datu a času evidovat, kdy členovi spolku končí zkušební doba, na chod systému a oprávnění by neměla tato hodnota mít vliv.

2.2 Evidence schůzí

Jako další potřebná část systému byla identifikována evidence schůzí, nebo zasedání spolku. Ke každé schůzi by mělo být evidováno, o jakou schůzi se jedná, kdy má schůze být, kdy by měla končit, jaký je její program (v bodech) a jaká je očekávána docházka. Při schůzi by měla být evidována docházka členů a zpráva ze schůze. Očekávaná docházka by se měla dělit na tři úrovně. Povinná účast všech členů spolku, povinná účast pouze pro vybrané členy spolku (typické například pro schůze týmů) a dobrovolná účast.

Přístup k informacím o schůzích by měl být ke čtení dostupný všem členům spolku a k editaci přístupný pouze oprávněným osobám. Pokud by byla očekávána přítomnost člena a člen by se nemohl dostavit, tak by se měl být schopen omluvit ze schůze. Ve stanovách SU FIT se nachází, že povinností člena je předem se omluvit. Omluvy předem pak umožňují efektivnější zpětnou kontrolu docházky. Systém by měl toto stanovisko respektovat.

Z požadavků také vyplynulo, že se může stát, že se schůze bude nějakou dobu plánovat, než dojde k jejímu oficiálnímu vyhlášení. Tudíž by zde měla být možnost nějakého rozpracovaného stavu schůze.

2.3 Správa financí

Správa financí spolku je dost citlivé téma a mělo by být řádně evidováno. Systém by měl být schopen evidovat jednotlivé finanční pohyby (transakce) ve spolku. Mezi důležité informace u jednotlivého finančního pohybu patří kdo transakci založil, naposledy upravil, o jakou transakci se jednalo, kdy byla provedena, jestli se jednalo o příchozí, nebo odchozí transakci, nad jakým účtem byla transakce provedena a o jakou částku se v transakci jednalo. Pokud se jednalo o odchozí transakci jako například nákup, tak by mělo být možné zaevidovat dodavatele. Datum a čas provedení transakce a datum a čas zapsání transakce se mohou lišit. Tudíž je třeba tyto informace evidovat jako dvě nezávislé hodnoty. K transakci by mělo být možné v případě potřeby zapsat poznámku.

K transakci by také mělo být možno nahrát přílohu pro potřeby archivace faktury, daňového dokladu apod. Přílohu může nahrát pouze člen spolku, který transakci založil, případně oprávněná osoba.

Systém by měl být schopen vytvářet reporty pro jednodušší finanční správu. Mezi reporty by měly být informace o stavech na účtech, finanční report za jednotlivé dodavatele a podle kategorií.

Transakci by měl být schopen zadat jakýkoliv člen spolku. Nicméně pokud není člen oprávněnou osobou, nebo předseda, tak uvidí pouze transakce, které založil.

2.4 Evidence majetku

Systém by měl umět evidovat majetek, co spolek vlastní, a umožnit správu. V systému by měly být evidovány základní informace o majetkové položce, evidence výpůjček položky a inventarizace.

V případě potřeby by mělo být možné neumožnit vypůjčení dané položky. Vzhledem k evidenci výpůjček a inventarizaci by nemělo být možné ze systému provádět mazání položek, pouze evidovat informaci, že spolek už takovou položkou nedisponuje.

U výpůjčky by mělo být evidováno, kdo má položku vypůjčenou, od kdy a do kdy. U inventárního záznamu by mělo být evidováno, kdo inventuru provedl, kdy ji provedl a kolik kusů v době inventury spolek vlastnil.

Pro zjednodušení vyhledávání položek by mělo být možné položky kategorizovat. Dále by z důvodu zjednodušení práce mělo být možné zadávat inventuru hromadně na jednom pohledu bez nutnosti vyhledávat každý produkt zvlášť. Inventuru by mělo být možné zadat i zpětně a nemusí ji provádět ten, kdo ji bude zapisovat do systému. Typicky když inventuru bude zadávat vedoucí týmu, ale spočítáním položek pověří nějakého člena v jeho týmu.

2.5 Správa akcí

Studentský spolek jako například SU FIT organizuje různé akce pro studenty. Tudíž by potřeboval nějaké centrální místo pro evidenci dat o akci za účelem plánování. Systém by měl k akci umožnit zaevidovat, o jakou akci se jedná, od kdy do kdy probíhá, jaké jsou její očekávané a reálné počty zúčastněných popis a poznámku. Dále by měl systém umožnit vkládání komentářů k akcím.

Akce může založit jakýkoliv člen spolku. Nicméně by neměl být každý schopen upravovat informace o akci. Upravovat informace o akci může jen ten, kdo akci založil, případně oprávněná osoba.

2.6 Systém oprávnění a zabezpečení

Ze sběru požadavků dále vyplynulo, že by se do systému neměl dostat každý. Do systému by se měl být schopen přihlásit pouze člen spolku, který byl jinou oprávněnou osobou ve spolku do systému registrován.

Dále by každý člen spolku neměl mít všechny pravomoci, tudíž je požadováno, aby systém disponoval systémem oprávnění. Z požadavků vyplynulo, že je třeba vytvořit oprávnění, které umožní členovi neomezený pohyb po systému (takové pravomoci by připadly předsedovi), a dále vytvořit oprávnění podle jednotlivých sekcí. Pokud člen opustí spolek, neměl by mít v systému žádné oprávnění ani možnost přihlásit se do něj.

Systém oprávnění by měl dále umožňovat podle sekcí následující:

- **Správa členů a týmů**: Každý člen spolku by si mohl prohlížet informace o jiných členech spolku. Člen by mohl měnit informace pouze sám sobě, pokud by měl případné vyšší oprávnění, pak by mohl měnit informace i ostatním osobám.
- **Evidence schůzí** Zobrazit seznam schůzí, případně se ze schůze omluvit, by měl mít možnost každý, ale pouze osoby s vyššími právy mohou zakládat a spravovat schůze. Zde se jako osoba s vyššími právy chápe vedoucí týmu, předseda, případně další oprávněná osoba.
- **Správa financí** Člen spolku může prohlížet a měnit pouze své transakce. Účty a dodavatele pouze číst. Vytváření a správu všech transakcí, účtů, dodavatelů, ... by mohla provádět pouze osoba s vyššími právy a předseda.
- **Evidence majetku** Procházet seznam majetku a půjčovat si majetek by měl mít možnost každý člen. Vytváření a správu majetku by mohla provádět pouze oprávněná osoba.
- **Správa akcí** Každý člen spolku by měl mít možnost vytvořit a spravovat pouze takové akce, kterých je autorem. Osoba s vyšším oprávněním by pak měla mít možnost spravovat všechny akce.

2.7 Integrace se systémy třetích stran

Vzhledem k tomu, že spolky aktuálně používají různé aplikace třetích stran a jsou zvyklé na určité zažité standardy, by mělo být umožněno propojení systému s těmito aplikacemi. Zjistilo se, že v rámci SU FIT se nejvíce používají dvě služby, a to Google Workspace[1] a Discord[2]. Tudíž by mělo být možné se přihlásit do systému také prostřednictvím těchto služeb a přenášet data do těchto systémů.

Po registraci oprávněnou osobou by měl být nový člen spolku ověřen a na službě Discord obdržet příslušná oprávnění. Dále by mu měl být automaticky založen účet ve službě Google Workspace.

Další požadavek v oblasti integrace se systémy třetích stran je integrace kalendáře Google. To znamená, že při vytvoření schůze nebo akce by ji měl systém automaticky zapsat jako událost do kalendáře. Dále by měl systém odeslat oznámení o svolané nebo zrušené schůzi do příslušného kanálu, nebo soukromého chatu s členem spolku.

[1] Google Workspace – `https://workspace.google.com/intl/cs/`
[2] Discord – `https://discord.com/`

Kapitola 3

Návrh

Hlavním cílem aplikace je vytvořit systém, který umožní centrální správu spolku a poskytne centrální úložiště informací spolku. Cílová aplikace by měla běžet kdekoliv bez ohledu na operační systém. Proto bylo rozhodnuto, že se bude jednat o informační systém postavený na webových technologiích.

Systém by měl umožnit práci ve webovém rozhraní k tomu určeném, ale také na mobilních zařízeních, případně v aplikacích třetích stran. Celý systém by měl být tedy navržen na principu API First[1].

3.1 Technologie a návrh API

Tato podkapitola se bude věnovat výběru vhodných technologií pro implementaci serverové aplikace poskytující rozhraní pro klientské aplikace. Základní požadavek je ten, aby rozhraní splňovalo dnes používané standardy pro komunikaci se službami pomocí protokolů HTTP a TCP. Z toho důvodu se pro API využije protokol HTTP s použitím architektury REST a pro umožnění živé komunikace bez nutnosti vlastnictví veřejné IP adresy, nebo požadavku umístění serveru a klientů ve společné síti, se využije komunikační protokol WebSocket.

REST (Representational State Transfer) je architektura rozhraní sloužící pro distribuované systémy. Popisuje, jak jednoduše nakládat s daty na serveru (získávat, vytvářet, modifikovat a mazat). Správně navržené rozhraní REST by mělo přesně specifikovat, co poskytuje klientovi. K tomuto účelu dnes slouží různé specifikace. Mezi nejznámější patří například OpenApi. Dále by služba měla být bezsstavová. Jednotlivé požadavky by se mezi sebou neměly ovlivnit.

WebSocket je komunikační protokol poskytující obousměrnou komunikaci mezi klientem a serverem. WebSocket je protokol postavený na TCP, nicméně používá HTTP k navázání komunikace. V systému bude WebSocket používán pro komunikaci s boty v reálném čase. Do budoucna bude možné počítat i s integrací webové aplikace pro možnost notifikací, případně konverzací.

Výsledné řešení počítá s podporou rozhraní REST i WebSocket v jedné aplikaci, tudíž v kombinaci s jazykem C# se nabízí aplikační rámec ASP.NET Core.

ASP.NET Core je webový aplikační rámec s otevřeným zdrojovým kódem vyvíjený společností Microsoft a komunitou. Tento aplikační rámec umožňuje vytvářet webové aplikace v jazyce C#. Pro tuto práci bude této technologie využito jako poskytovatele rozhraní

[1] API First návrh je, že se nejdříve navrhuje rozhraní pro přístup klientů (API) a nad ním se implementují ostatní aplikace [15].

REST API a WebSocket. Vzhledem k tomu, že tyto typy rozhraní poskytuje nativně, je možné je implementovat do jedné aplikace. Protože je ASP.NET Core pouze sada knihoven, které společně slouží jako aplikační rámec, tak stále musí běžet na nějaké platformě .NET. K tomuto účelo byla vybrána platforma .NET 5, která byla oficiálně vydána koncem roku 2020 a tudíž je možné na této platformě práci postavit.

3.1.1 Návrh API

Tato kapitola se bude zabývat návrhem rozhraní REST API. Aplikační rámec ASP.NET Core je postaven na architektonickém vzoru MVC[2].

Rozdělení Model, Pohled (View) a Kontrolér (Controller) je v ASP.NET Core REST API zastoupen následovně:

- **Model** – Datové modely a uložení informací.
- **Pohled** – Samotné rozhraní (reprezentace v podobě dat ve formátu JSON).
- **Kontrolér** – Implementuje logiku, která nakládá s modelem a pohledem.

Pro potřeby systému je třeba vytvořit takové rozhraní REST API, které umožní přímočarou integraci do klientských aplikací. Z toho důvodu REST API musí obsahovat alespoň následující kontroléry a metody:

- **Správa členů a týmů** – Získání seznamu členů, získání detailu člena, získání profilového obrázku, vytvoření člena, modifikace člena, nahrání profilového obrázku, změna hesla, nastavení osobní poznámky.
- **Správa týmů** – Získání seznamu týmů, získání detailu týmu, získání seznamu členů v týmu, vytvoření týmu, modifikace týmu, smazání týmu, přidělení člena do týmu, vyřazení člena z týmu.
- **Pohovory** – Získání seznamu pohovorů, získání detailu pohovoru, vytvoření pohovoru, modifikace pohovoru a smazání pohovoru.
- **Přístupy** – Získání seznamu definic přístupů, získání detailu definice přístupu, získání seznam členů, kteří mají nastavenou konkrétní definici, vytvoření definice přístupu, modifikace definice přístupu, smazání definice přístupu a přidělení přístupu členovi.
- **Schůze** – Získání seznamu schůzí, získání detailu schůze, získání docházky na schůzi, získání reportu docházky ze všech schůzí, vytvoření schůze, úprava schůze, znovu svolání schůze, omluvení se ze schůze a aktualizace docházky na schůzi.
- **Finance – Účty** – Získání seznamu účtů, získání detailu účtu, vytvoření účtu, úprava účtu a smazání účtu.
- **Finance – Dodavatelé** – Získání seznamu dodavatelů, získání detailu dodavatele, vytvoření dodavatele, úprava dodavatele a smazání dodavatele.
- **Finance – Kategorie** – Získání seznamu kategorií, vytvoření kategorie, úprava kategorie a smazání kategorie.

[2]MVC je zkratka z anglického Model-View-Controller.

- **Finance – Transakce** – Získání seznamu transakcí, získání detailu transakce, vytvoření transakce, úprava transakce, smazání transakce, získání reportů (podle typu transakce, podle dodavatele a podle kategorie), získání, vytvoření a smazání přílohy u transakce.
- **Akce** – Získání seznamu akcí, získání detailu akce, vytvoření akce, úprava akce, smazání akce a vytvoření, úprava a smazání komentáře u akce.
- **Přihlášení** – Přihlášení formou zadání jména a hesla a podpůrné metody pro udržení přihlášení.
- **Přihlášení – OAuth2** – Metody pro vygenerování odkazů na přesměrování OAuth 2.0 bran externích systémů a další podpůrné metody.
- **Správa majetku** – Získání seznamu položek majetku, vytvoření položky, úprava položky, metody pro vypůjčení a vrácení položky a metody pro inventuru.
- **Správa majetku – Kategorie** – Získání seznamu kategorií, vytvoření kategorie, úprava kategorie a smazání kategorie.
- **Interní správa systému** – Odeslání upozornění externím systémům a správa aplikací přistupujících k systému (získání seznamu, vytvoření, modifikace a smazání)

3.2 Databáze

Tato podkapitola se bude věnovat návrhu databáze a výběru databázového systému, ve kterém budou data uchovávána.

Informační systém o takovém rozsahu vyžaduje ukládat velké množství dat a podle toho je třeba vybírat databázový systém. Databázi bude využívat serverová část systému poskytující API. Po prověření možností, které nabízí knihovna Entity Framework Core[3], se nabízely 3 kandidátní databázové systémy [1], které se současně nacházejí v seznamu nejpoužívanějších databázových systémů [6]: MySQL, PostgreSQL a Microsoft SQL Server.

Microsoft SQL Server (MS SQL) je proprietární řešení databázového systému vlastněné společností Microsoft. SQL server je poskytován v několika licenčních verzích. Edice Express a současně nejnižší nabízená licence je poskytována pro produkční použití zcela zdarma. Nicméně zde platí různá omezení jako například maximální velikost jedné databáze do 10 GB, nebo omezení využití paměti pro zefektivnění zpracování dotazů. Ostatní edice Microsoft SQL serveru vyžadují placenou licenci [14].

PostgreSQL server je svobodný a otevřený objektově-relační databázový systém. Oproti MS SQL serveru je dostupný zdarma a neobsahuje taková omezení jako edice Express. Dále podporuje pokročilejší možnosti práce s transakcemi a speciální datové typy, které by se u MS SQL řešily komplikovaně, až nerealizovatelně. Jedním z takových rozšíření je speciální datový typ JSON, který by se u MS SQL musel řešit další tabulkou.

MySQL je otevřený databázový systém vlastněný společností Oracle Corporation. Oproti MS SQL je sice stejně jako PostgreSQL k dispozici zdarma, nicméně oproti PostgreSQL neobsahuje pokročilé možnosti. Také jsou u MySQL známy implementační chyby, které mohou způsobit potíže s výkonností a stabilitou serveru.

Po porovnání výše uvedených možností bylo nakonec vybráno řešení používající databázový server PostgreSQL.

[3]Entity Framework Core – GitHub `https://github.com/dotnet/efcore`

Další podsekce se budou věnovat návrhu uložení dat v databázi (bez ohledu na vybraný databázový systém).

3.2.1 Správa členů a týmů

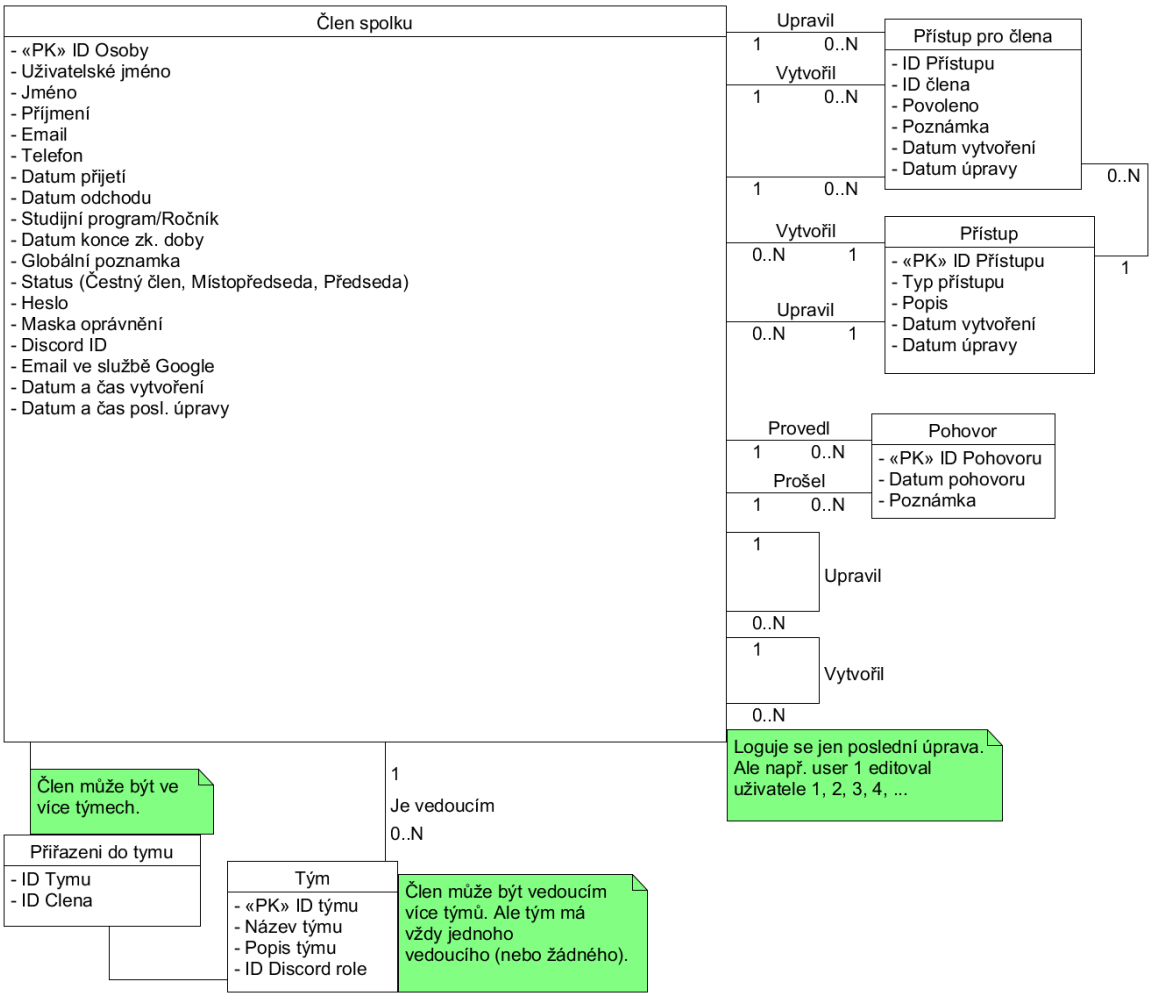
Základní a současně tou nejdůležitější entitou v celém systému je entita člena, která je vyobrazena jako součást ER diagramu v obrázku 3.1. V této entitě je zapotřebí evidovat potřebné osobní, provozní a interní informace o členovi spolku. Jako osobní informace se eviduje uživatelské jméno, jméno, příjmení, e-mailová adresa, telefonní číslo a studijní program (pokud je člen ještě studentem). Za provozní informace se považují údaje, které spolek potřebuje ke své agendě. Mezi tyto informace se řadí datum přijetí do spolku, datum odchodu ze spolku, datum konce zkušební doby, globální a interní poznámka a status člena. Za interní informace se považují údaje, které ve spolku slouží k čistě informativním účelům, nebo k účelům zajištění chodu systému. Mezi tyto informace se řadí identifikátory externích systémů (Discord a Google), datum a čas vytvoření člena, datum a čas poslední modifikace člena, heslo a maska oprávnění.

K entitě týmu je evidován název týmu, popis týmu a interní identifikátor a kdo je vedoucím týmu. Členství v týmu se eviduje pomocí vazební entity, která propojuje člena spolku a tým.

Na člena spolku se dále vážou pohovory, kterými člen prošel, nebo je vedl. K entitě pohovoru se eviduje datum a čas, kdy pohovor proběhl, kdo pohovor vedl, s kým byl pohovor veden a poznámka k pohovoru.

Evidence přístupu je složena ze dvou entit, kde jedna slouží k uložení popisu objektu přístupu. Objekt přístupu mohou být prostory, sekce na sociální síti Facebook, nebo jednotlivé služby z rodiny služeb Google. U objektu přístupu by kromě informace, o jaký přístup se vlastně jedná, měl být uložen popis, datum vytvoření a poslední úpravy tohoto objektu v systému. Další entita slouží k provázání člena spolku a objektu přístupu. U takového propojení by mělo být uloženo, kdo a kdy přístup členovi udělil, kdo a kdy přístup členovi naposledy upravil, poznámka a zda má přístup povolen, nebo zamítnut.

Obrázek 3.1 zachycuje kompletní ER diagram pro správu členů a týmů.



Obrázek 3.1: Část ER diagramu popisující člena spolku, týmy, pohovory a přístupy

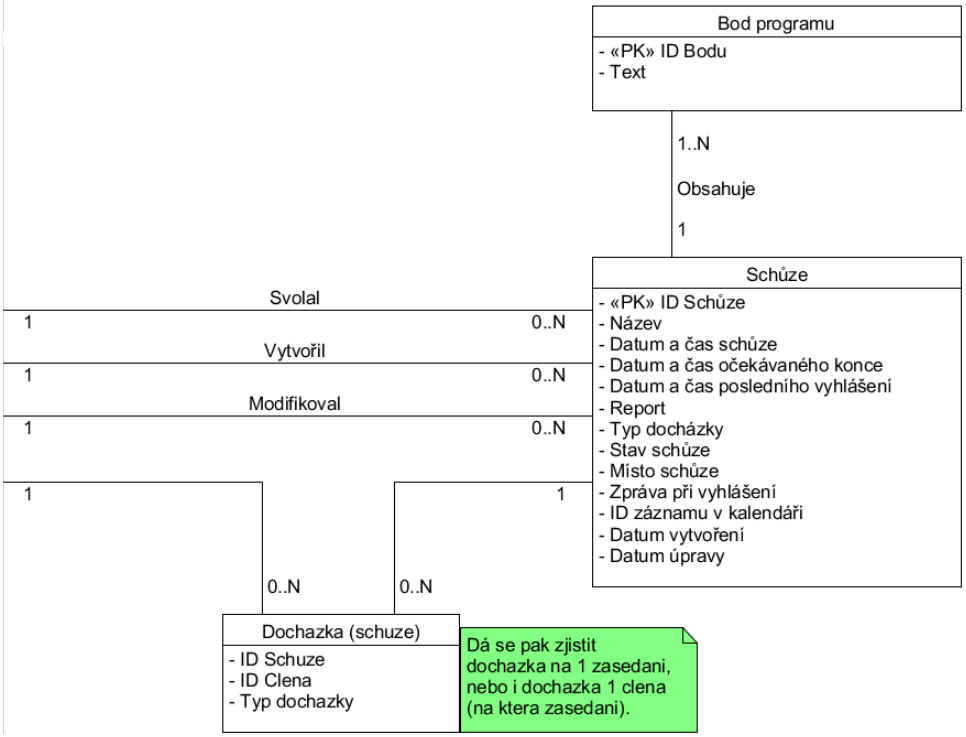
3.2.2 Evidence schůzí

Přihlášený člen, který má příslušné oprávnění, může vytvářet schůze. U schůze, jejíž ER diagram popisovaný v této kapitole se nachází na obrázku 3.2, se ukládají základní informace o schůzi jako jednoznačný identifikátor, název, datum a čas začátku schůze, datum a čas očekávaného konce, report, místo schůze, datum vytvoření a datum poslední modifikace. Dále se ke schůzi ukládají podpůrné informace k upřesnění schůze a propojení s externími systémy. Mezi podpůrné informace patří typ docházky, stav schůze, zpráva, která se odesílá členům při vyhlášení schůze, a datum a čas posledního vyhlášení schůze.

Typ docházky je rozdělení požadované účasti na schůzi. Účast je povinná, povinná pro pozvané a nepovinná. Povinná účast pro pozvané členy může být využitelná například při schůzi týmu. Stav schůze je vytvořena, svolána a zrušena.

Pro bod programu je definována samostatná entita obsahující unikátní identifikátor a text bodu.

Ke schůzi se eviduje také docházka členů. Každý člen může mít na jedné schůzi pouze jeden záznam docházky. K tomuto účelu je definována samostatná entita, která obsahuje identifikátor člena, schůze a stav docházky.



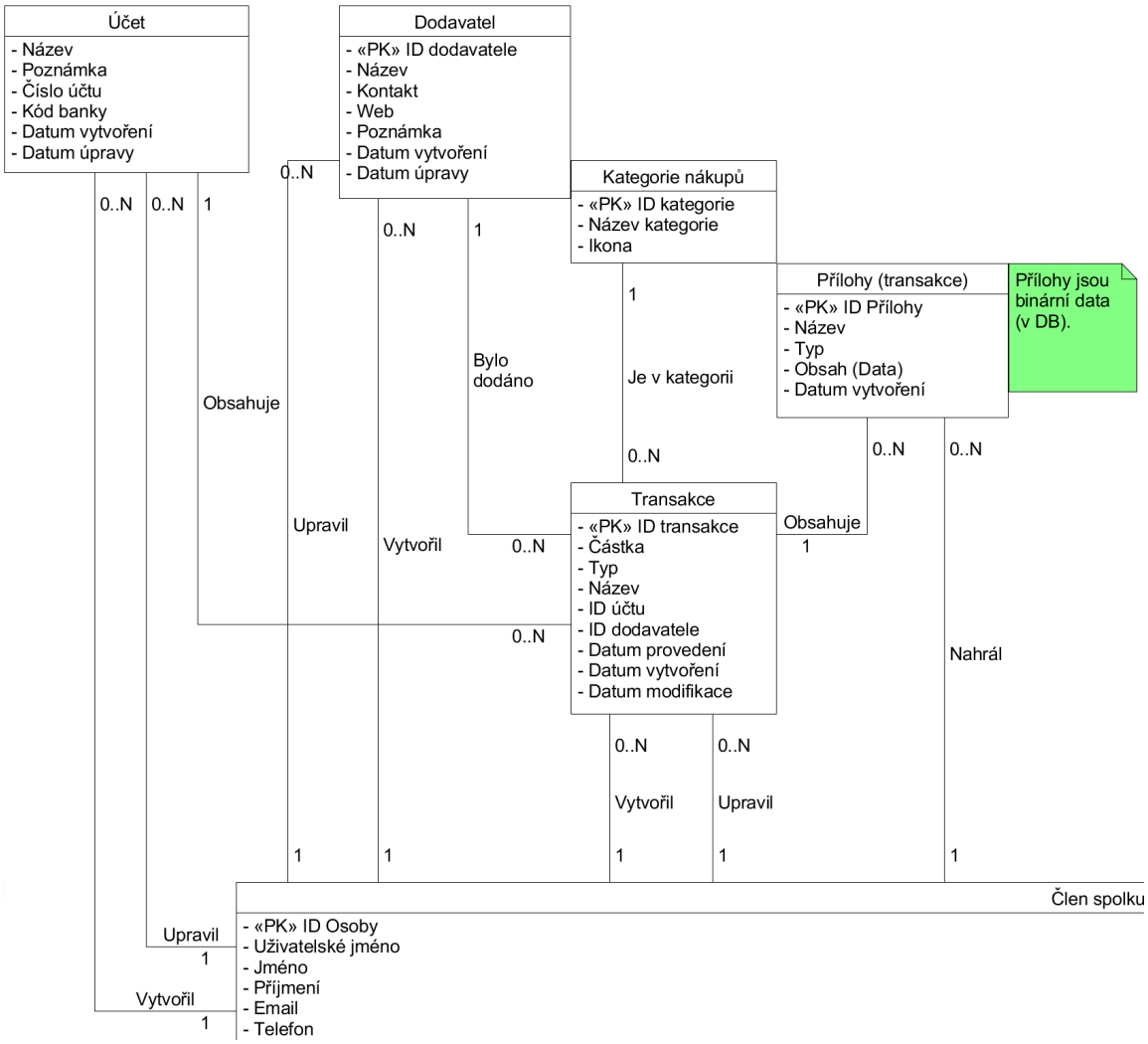
Obrázek 3.2: ER diagram popisující entity pro evidenci schůzí. Vazby směřující mimo obrázek navazují na entitu člena popsanou v kapitole 3.2.1.

3.2.3 Správa financí

Správa financí ve spolku se v systému bude ukládat jako transakce, podobné bankovním transakcím. Transakce finančního pohybu, která je popsána v ER diagramu na obrázku 3.3, je nejdůležitější část finanční správy spolku. Ke každé transakci bude uložen interní identifikátor, o jakou transakci se jednalo, částka, typ transakce (příchozí, nebo odchozí), nad jakým účtem byla transakce provedena, kdy byla provedena, vytvořena a naposledy upravena. Ke každé transakci existuje entita přílohy, kde bude možné nahrát soubor (pokud bude třeba).

U entity účtu bude uložen název účtu, poznámka, číslo účtu, kód banky, datum vytvoření a poslední úpravy. K entitě dodavatele se bude ukládat název, kontaktní údaje, poznámka, datum vytvoření a poslední modifikace.

Každou transakci bude možné kategorizovat. K tomu má pomoci entita kategorie nákupu.



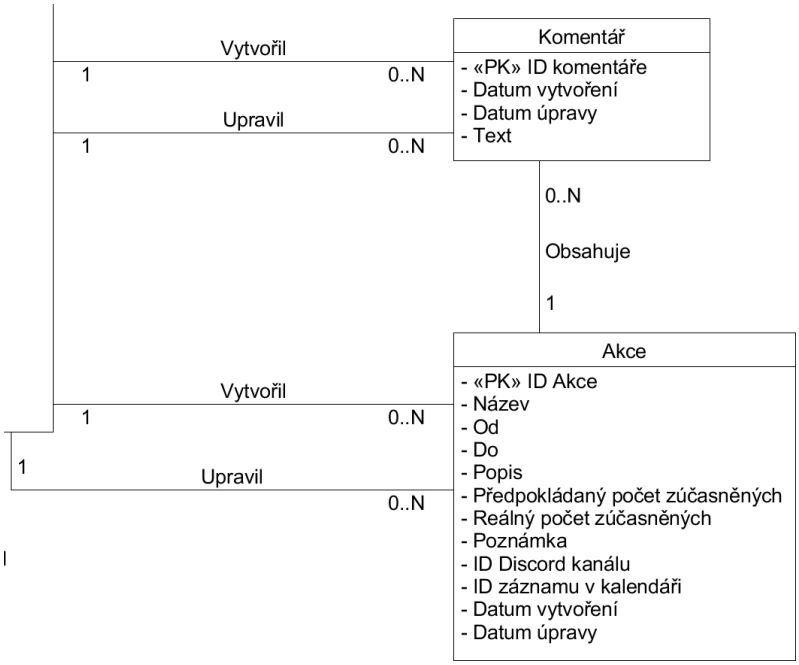
Obrázek 3.3: ER diagram popisující entity pro správu financí.

3.2.4 Správa akcí

Akce je v systému modelována pouze jako jedna entita, vyobrazená v ER diagramu na obrázku 3.4. V této entitě jsou uloženy základní údaje jako interní identifikátor akce, název, kdy akce začala, kdy skončila, popis, předpokládaný počet zúčastněných, reálný počet zúčastněných, poznámka, datum vytvoření a poslední úpravy akce.

Dále jsou v entitě akce uloženy podpůrné údaje potřebné pro provoz systému jako celku. Jedná se o identifikátor kanálu na Discordu, ve kterém může probíhat diskuze, a identifikátor události v Kalendáři Google.

Ke každé akci je možné vést diskuzi přímo v systému, a to za pomoci komentářů. Komentář reprezentuje samostatná entita a obsahuje datum vytvoření, datum poslední úpravy a text komentáře.



Obrázek 3.4: ER diagram popisující správu akcí. Vazby směřující mimo obrázek navazují na entitu člena spolku.

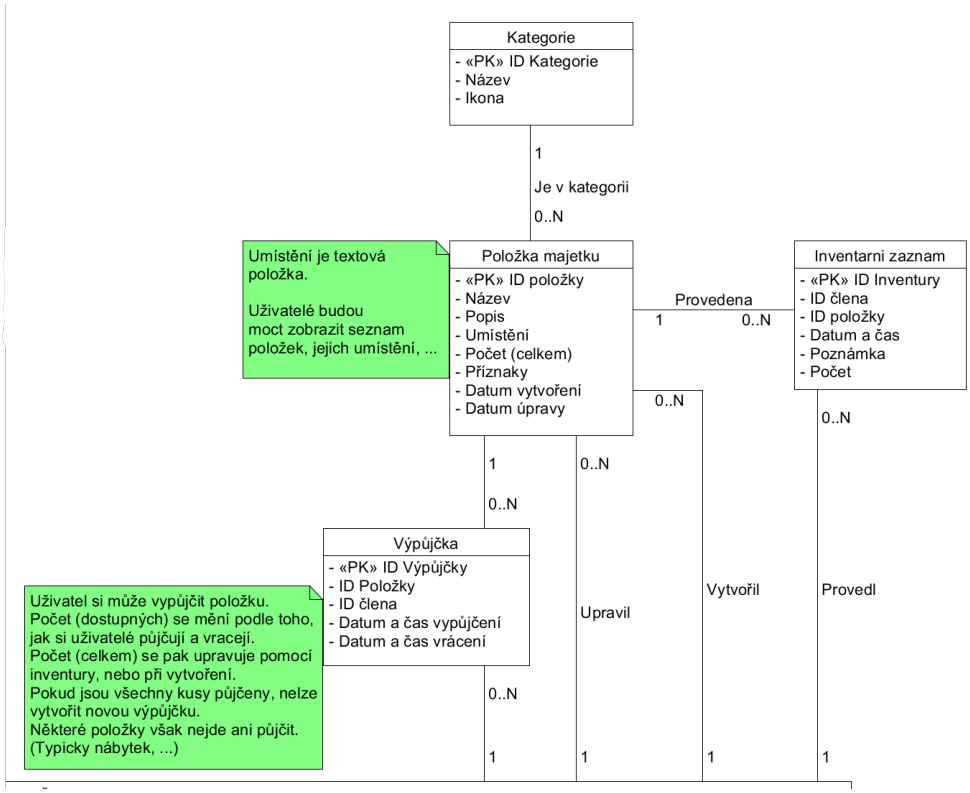
3.2.5 Evidence majetku

Základem evidence majetku je entita reprezentující položku majetku, která je vyobrazena v ER diagramu na obrázku 3.5. V této entitě je uložen identifikátor, název, popis, umístění, počet kusů (celkem), příznaky, datum vytvoření a datum úpravy.

Počet kusů se zadává při vytvoření položky v systému a aktualizuje se s každou inventurou. Atribut *příznaky* je bitová maska obsahující následující bity:

- Bit pro nastavení, že si žádný člen spolku nemůže danou položku vypůjčit. Může sloužit pro položky, které preventivně[4] nelze půjčit, nebo to reálně není možné.
- Bit pro nastavení, že položka je ze systému „Smazána“. Kvůli zachování historie inventárních záznamů a výpůjček není možné položku ze systému fyzicky smazat.

[4]Preventivní situace může být například tehdy, když položka byla zapsána do systému, ale reálně ještě není možné vyzvednutí, nebo se může jednat o zapůjčený majetek.



Obrázek 3.5: ER diagram popisující správu majetku. Vazby směřující obrázek navazují na entitu člena spolku.

3.2.6 Podpůrné části

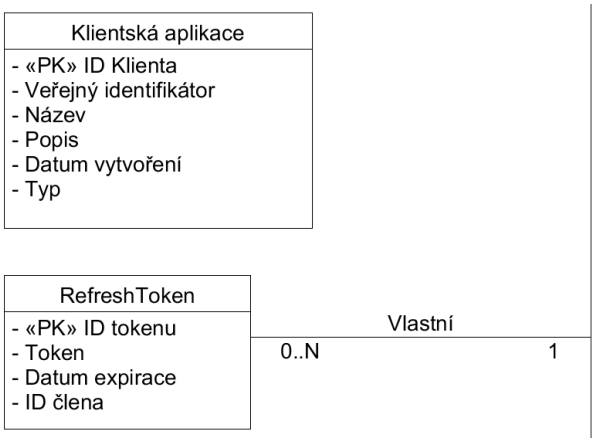
Kromě výše uvedených entit je vyžadováno, aby systém trvale ukládal provozní data, která musí vydržet restart systému.

Při přihlášení člena do systému mu bude vygenerován přístupový token s omezenou platností. Aby tento token mohl dále platit, tak je mu při přihlášení vygenerován také obnovovací token. Tento token si musí systém pamatovat a musí přežít restart systému. K tomu bude sloužit entita *`RefreshToken`*, která obsahuje identifikátor tokenu, token v textové podobě, datum expirace a vazbu na člena spolku, jehož entita je vyobrazena na ER diagramu v obrázku 3.1.

Je také třeba zajistit, aby se kromě členů do systému nedostala neoprávněná aplikace. K tomu slouží entita pro uchování klienta. U entity klienta je uložen jeho identifikátor, přístupový token, název, popis, datum vytvoření a typ aplikace.

Typ aplikace může být Discord bot, Google bot, oficiální webová aplikace, nebo obecně klient třetí strany.

ER diagram obsahující veškeré entity a vazby potřebné k zajištění provozních dat je vyobrazen na obrázku 3.6.



Obrázek 3.6: ER diagram popisující podpůrné části systému.

3.3 Volba technologií pro webového klienta

Tato kapitola se bude věnovat výběru vhodné technologie pro uživatelské rozhraní ve formě webové aplikace. Při výběru, jak bude webová aplikace implementována, se nabízelo několik možností. Jednou z možností bylo implementovat aplikaci úplně od základu s využitím jazyků HTML, CSS a JavaScript. Nicméně taková implementace by si vyžádala příliš velké úsilí, tudíž byla zařazena jako nerealizovatelná. Dále se nabízelo využít existujících aplikačních rámců pro implementaci webových uživatelských rozhraní.

Kandidáty pro výběr byly:

- **ASP.NET Core Razor Pages** – Jedná se o stejný aplikační rámec, který byl využit k implementaci API, popsaný v kapitole 3.1.
- **Angular** – Aplikační rámec s otevřeným zdrojovým kódem vyvíjený společností Google a komunitou vývojářů pod licencí MIT.
- **React** – Knihovna pro vývoj uživatelských rozhraní vyvíjená společností Facebook a komunitou vývojářů pod licencí MIT.

Po prozkoumání výše uvedených knihoven a aplikačních rámců bylo vyloučeno využití aplikačního rámce ASP.NET Core pro uživatelské rozhraní. Tento rámec slouží pouze ke generování statického obsahu bez dynamické funkčnosti (živé načítání dat, animace, ...). To by bylo třeba implementovat pomocí jazyka JavaScript.

Rozdíly mezi Angularem a Reactem byly zkoumány delší dobu, primárně z důvodu studia dokumentace obou možností. Mezi hlavní rozdíly patří například to, že React pracuje nad virtuálními elementy DOM[5], zatímco Angular pracuje nad reálnými elementy DOM v prohlížeči. Virtuální DOM používaný v knihovně React umožňuje rychlejší vykreslování elementů díky detekci změn a vyhodnocení toho, co se má vlastně vygenerovat [5]. Naopak výhodou Angularu oproti Reactu je jednodušší proces implementace. Zatímco React má logiku i šablonu implementovanou v jednom souboru, Angular obsahuje pokročilý systém šablon, který současně umožňuje oddělit komponentu na více souborů a tím dosáhnout například skladby logika (logika psaná v TypeScript), šablona v podobě HTML souboru

[5]DOM, z anglického Document Object Model je objektová reprezentace dokumentu.

a případně také soubory obsahující kaskádové styly. Díky tomuto přístupu může dojít ke zjednodušení komplikovanějších komponent. Vzhledem k potřebám informačního systému a zkušenostem byl vybrán Angular.

Další průzkum byl potřeba pro stanovení toho, jak by měla aplikace vlastně vypadat. Pro tento účel bylo rozhodnuto využít nějaké šablony s otevřeným zdrojovým kódem. Na výběr bylo použití šablony SB Admin[6], nebo AdminLTE[7]. Po porovnání těchto dvou šablon a výběru budoucích uživatelů byla zvolena šablona AdminLTE.

Šablona AdminLTE je webová šablona s otevřeným zdrojovým kódem pro vytváření administračních informačních systémů. Je postavena na knihovnách Bootstrap a jQuery. Vzhledem k použití aplikačního rámce Angular budou ze šablony využity pouze kaskádové styly.

Bootstrap

Bootstrap je knihovna s otevřeným zdrojovým kódem sloužící k jednodušší tvorbě webových stránek za pomoci sady připravených kaskádových stylů a komponent [11]. Knihovna Bootstrap je šířena pod licencí MIT [12].

3.4 Návrh webové aplikace

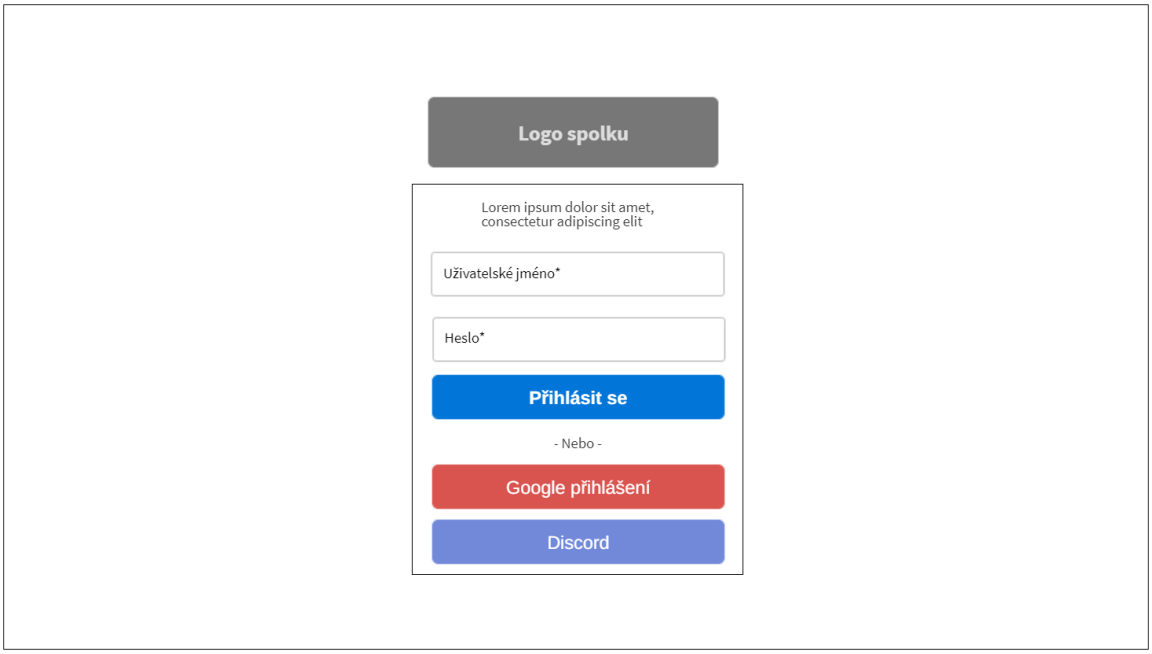
Tato kapitola se bude věnovat návrhu webové aplikace (resp. uživatelského rozhraní). Vzhledem k tomu, že bylo využito šablony AdminLTE, která byla popsána v kapitole 3.3, bylo zapotřebí, aby se výsledný návrh přizpůsobil této šabloně.

První návrhy webové aplikace byly vytvořeny ručně na papír. Některé byly následně překresleny pomocí nástroje MockFlow[8].

3.4.1 Návrh přihlašovací obrazovky

Jak již bylo zmíněno, do informačního systému by se neměla dostat neoprávněná osoba. Tudíž by uživatelské rozhraní mělo poskytnout odpovídající přihlášení, které se bude zobrazovat vždy, když uživatel nebude přihlášen. Přihlašovací obrazovka nacházející se na obrázku 3.7 by měla být jednoduchá a nezobrazovat položky, které se nabízí pouze přihlášeným uživatelům. Dále by měla přihlašovací obrazovka poskytovat možnost přihlášení se externími účty, pokud bude přihlášení externími účty dostupné.

[6] SB Admin – `https://startbootstrap.com/template/sb-admin`
[7] AdminLTE – `https://adminlte.io/`
[8] MockFlow – `https://mockflow.com/`



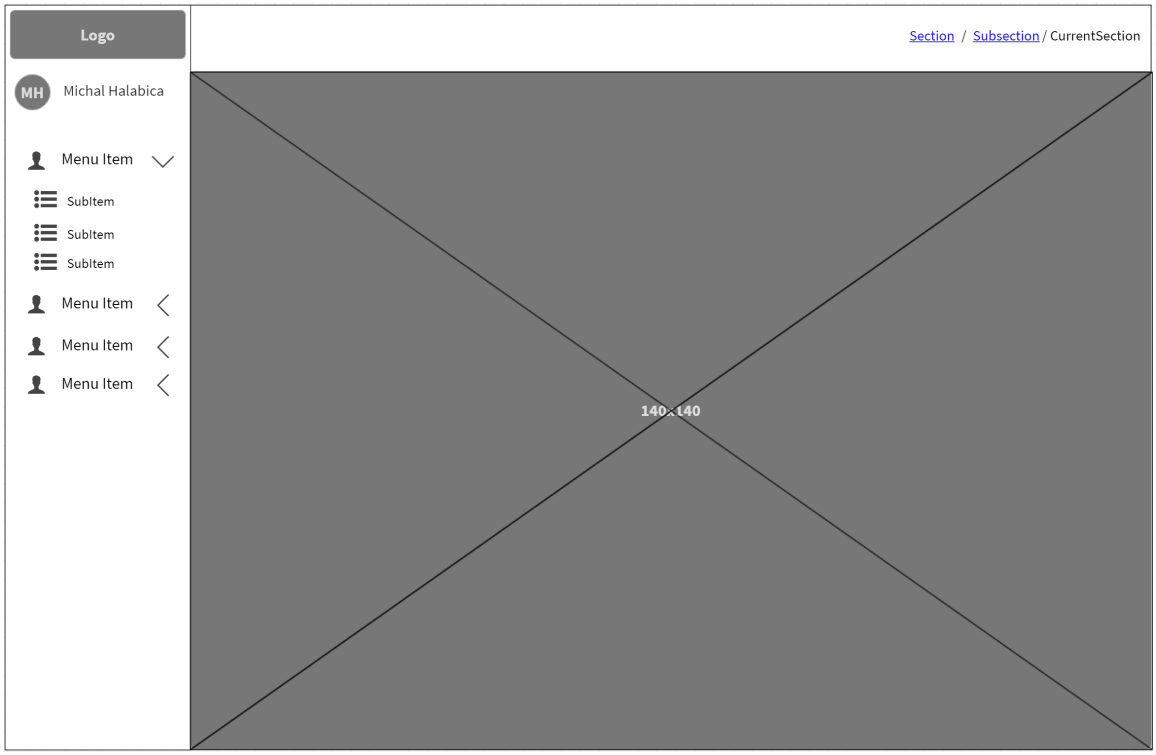
Obrázek 3.7: Návrh přihlašovací obrazovky

3.4.2 Návrh základního rozložení

Základním rozhraním se rozumí společné prvky, které se nachází na každém pohledu uživatelského rozhraní přihlášenému uživateli. Jedná se o menu, navigační lišty a plochu pro obsah jednotlivých podstránek.

Vzhledem k tomu, že se může systém dále rozšiřovat, je nutné ve webovém rozhraní zajistit, aby se do budoucna do menu daly přidávat další sekce a podsekce. Menu by se mělo skládat až ze dvou úrovní. První úroveň budou reprezentovat jednotlivé sekce v systému. Například: Správa členů, týmů, schůzí apod. Druhou úroveň budou reprezentovat jednotlivé podsekce. Například ve správě členů to může být seznam členů, přidání nového člena apod. Další prvek by měla být dynamická navigace. Ta by se přehledně měla nabízet uživateli, aby viděl, kde se zrovna nachází, a mohl se případně vrátit do nějaké podsekce zpět.

Celý návrh základního rozložení se nachází na obrázku 3.8 níže.



Obrázek 3.8: Návrh základního rozložení uživatelského rozhraní

3.4.3 Společný návrh podsekcí

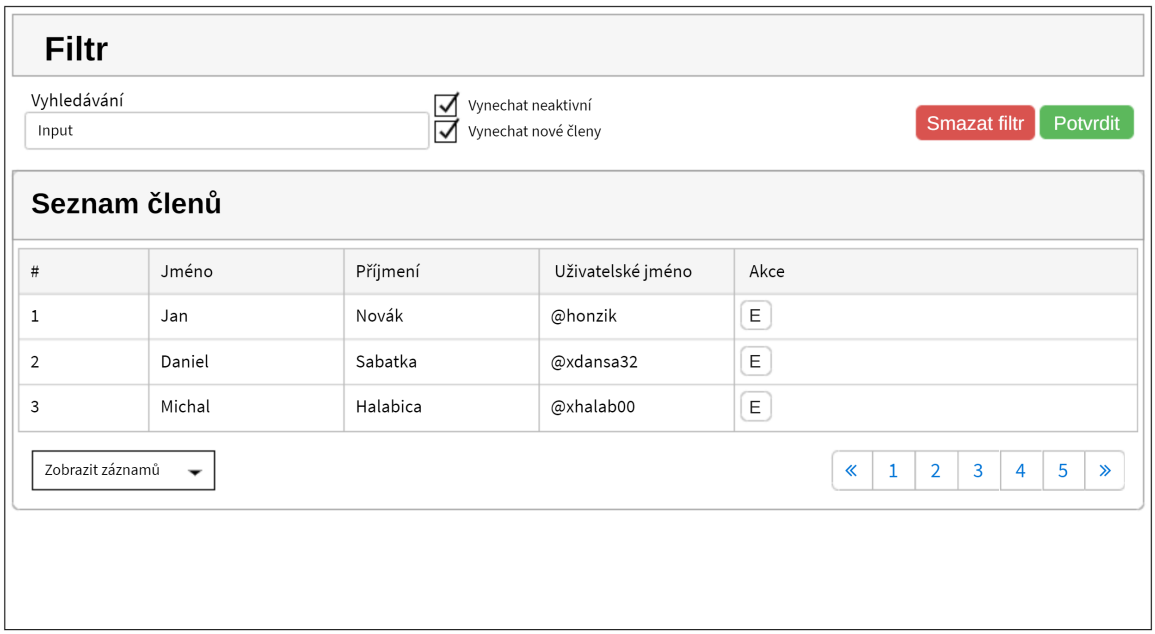
Jako další bylo potřeba vytvořit jednotný návrh podsekcí. Tyto podsekce se budou ve většině případů skládat z formulářů a tabulkových zobrazení. Dále bylo třeba vytvořit návrh pro podsekce obsahující grafy. Jako příklad tabulkových a formulářových zobrazení zde budou popsány podsekce správy členů nacházející se na obrázcích 3.9 a 3.11. Jako příklad podsekce obsahující grafy bude popsán finanční report. Ostatní sekce budou vzhledově vycházet ze stejných návrhů.

Tabulkové zobrazení

Tabulkové přehledy jsou v celém systému navrženy jako dva bloky, které mají svůj účel. První blok slouží jako formulář pro vyhledávací prvky. Ve správě uživatelů se může jednat například o vyhledávání ve jméně člena, u schůzí se zase může jednat o prvky rozsahu data a času apod. Součástí každého filtru je také možnost filtr smazat, respektive uvést do výchozího stavu. Vzhledem k tomu, že změna se nemá provést hned, je třeba do filtru zakomponovat odpovídající potvrzovací tlačítko.

Druhým blokem by měla být samotná reprezentace dat pomocí tabulky, nebo seznamu. Reprezentace všech dat na jedné stránce může být v případě velkého množství záznamů uživatelsky nepřívětivá, tudíž by do stránky měly být zakomponovány stránkovací prvky. Stránkovacími prvky se rozumí nastavit počet záznamů, který se má zobrazovat, a nastavení aktuální strany. Každá tabulka by měla obsahovat hlavičku, díky které bude známo, která data se v daném sloupci nachází. Poslední sloupec tabulky by měl být vyhrazen pouze pro tlačítka operací. Například u seznamu členů se zde zobrazí tlačítko pro přechod do editace

člena a znovu nastavení hesla. U schůzí se může jednat o proklik do editace schůze, omluvení se ze schůze apod.

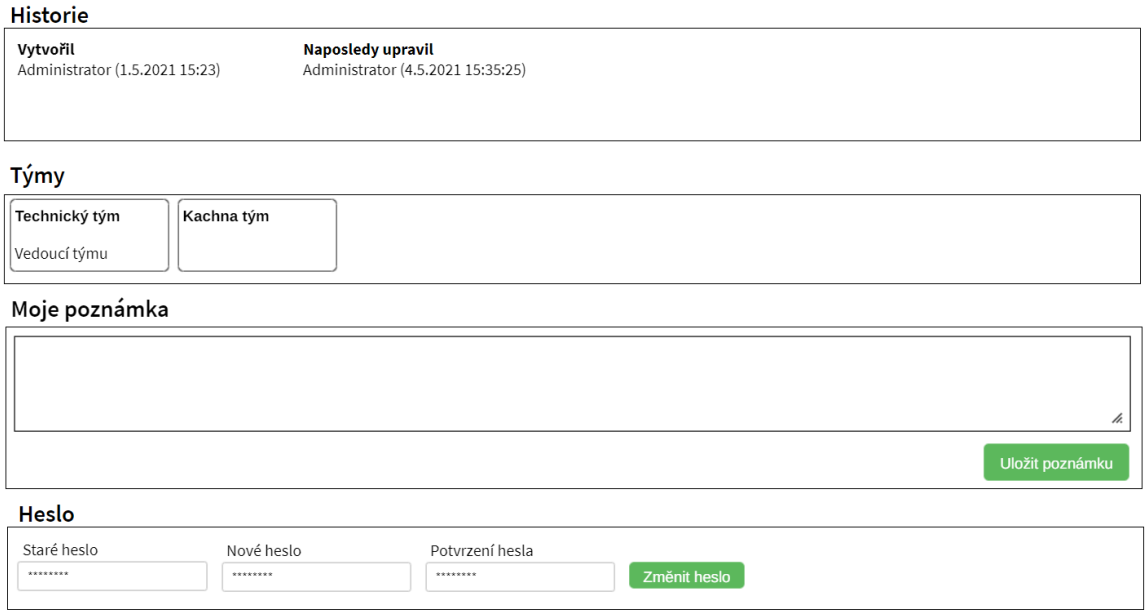


Obrázek 3.9: Návrh podsekce reprezentující tabulkové zobrazení v sekci seznam členů

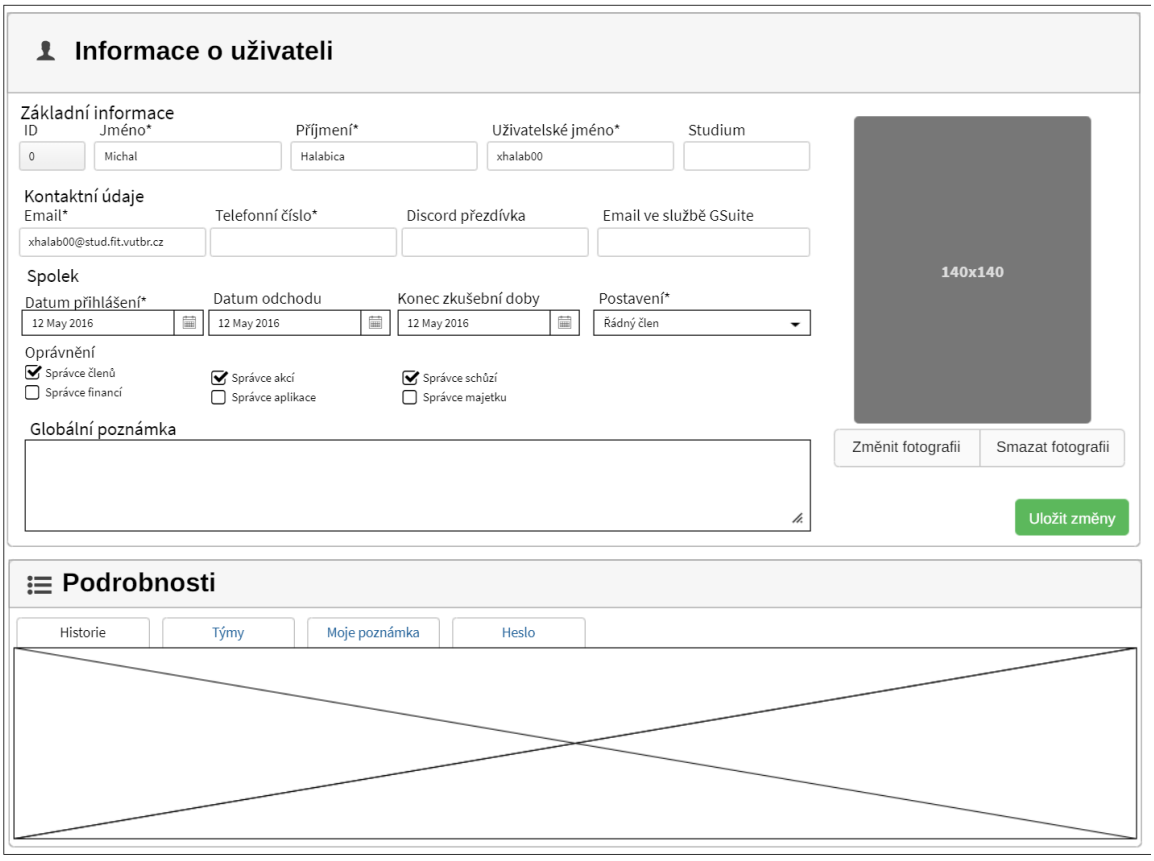
Formuláře

Formuláře mají v systému sloužit jakožto vstupní prvky pro vložení dat do systému. Většina formulářů v systému by se měla skládat ze dvou hlavních bloků. První blok by měl obsahovat základní údaje, které budou společné pro vytváření a editaci.

Druhý blok by měl obsahovat doplňující informace, případně další formuláře. Druhý blok by se měl zobrazovat v případě editace již existujícího záznamu. To z důvodu, že se zde mohou nacházet informace, které vyžadují vazbu na záznam, který v době vytváření neexistuje. Například v případě sekce editace člena spolku jsou doplňující informace umístěny na obrázku 3.10.



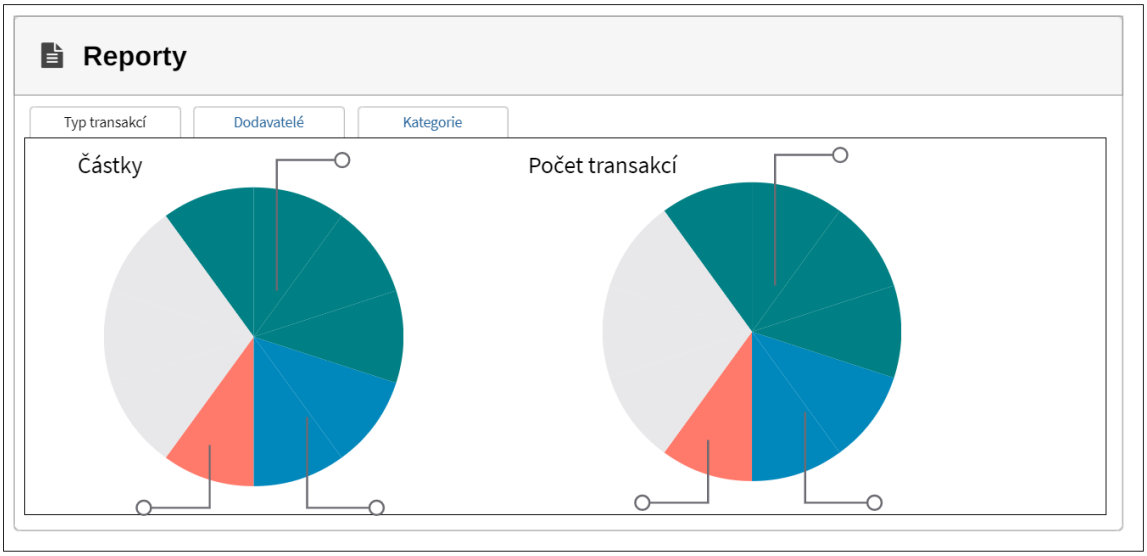
Obrázek 3.10: Návrh záložek podrobností u editace člena.



Obrázek 3.11: Návrh formuláře pro editaci člena spolku

Sekce s grafy

Pro přehlednější prezentaci seskupených dat je kromě tabulek možné využít také grafů. Grafy jakožto obrázky jsou jednodušší na reprezentaci dat a následné pochopení uživatelem. V tomto systému by mohly být grafy využity například ve finančních reportech, kde se dají pomocí grafů dobře prezentovat částky příchozích a odchozích transakcí, souhrn podle dodavatele apod. Sekce s grafem jsou na pohled poměrně jednoduché, jak tomu dokládá návrh na obrázku 3.12.



Obrázek 3.12: Návrh sekce obsahující grafy

3.5 Klient služby Discord (Discord bot)

Pro integraci služby Discord je zapotřebí vytvořit speciální aplikaci propojující službu Discord a informační systém. K tomu slouží tzv. „boti“. Každý spolek, který komunikuje pomocí platformy Discord, má založený tzv. Server. Na těchto serverech je možné pomocí hierarchie kanálů, rolí a pokročilé správy přístupu řídit provoz a obsah. Boti by měli tuto práci uživatelům usnadnit množinou automatizované logiky a příkazy.

Za účelem integrace služby Discord je třeba následující podpora:

- **Akce** – Propojení události a kanálu (případně jeho založení), kde má probíhat diskuze k akci.
- **Schůze** – Oznámení o svolání povinné schůze do společného kanálu, nebo pozvaným uživatelům do soukromé konverzace, případně oznámení o zrušení schůze.
- **Notifikace** – Odeslání hromadné zprávy do společného kanálu.
- **Správa týmů**
  - Založení role pro tým při vytvoření týmu.
  - Přiřazení role vedoucímu týmu.
  - Udržování konzistentního názvu role týmu s názevem týmu v informačním systému.

  - Smazání role při smazání týmu.

- **Správa členů** – Udržování konzistence rolí a přístupů s daty v informačním systému vč. periodické kontroly.

Boti se službou Discord komunikují za pomoci kombinace protokolu WebSocket a rozhraní REST API. Pomocí protokolu WebSocket od služby Discord dostávají události a pomocí REST API službě zasílají příkazy (Odeslání zprávy, smazání zprávy, přidání role, ...). Bot bude s informačním systémem komunikovat pouze pomocí rozhraní WebSocket. To znamená, že bude přes toto rozhraní dostávat i odesílat události.

3.6 Klient služby Google (GSuite bot)

Pro integraci služeb Google, resp. služeb poskytovaných v rámci Google Workspace, je zapotřebí speciální aplikace propojující služby Google a informační systém. Oproti službě Discord, služby Google k propojení nepoužívají protokol WebSocket, ale využívá se pouze rozhraní REST API.

Za účelem integrace služeb Google je třeba následující podpora:

- **Akce** – Zapisování a udržování akcí v kalendáři Google.
- **Schůze** – Zapisování a udržování schůzí v kalendáři Google.
- **Správa členů** – Vytváření a udržování členů. U každého člena se do systému propíše jméno a příjmení, globální poznámka, telefonní číslo a náhodné heslo pro přihlášení do služeb Google. Pro zajištění konzistence se budou periodicky provádět synchronizace všech členů.

Kapitola 4

Implementace

Tato kapitola navazuje na kapitolu 3, kde bylo popsáno, co by vlastně celý systém měl umět a v jakých technologiích by měl být implementován. Nyní se bude přecházet k samotnému popisu implementace.

Jak již bylo zmíněno, celý systém je rozdělen do několika projektů. Ve stručnosti se jedná o hlavní jádro celého systému poskytující rozhraní REST API a rozhraní na protokolu WebSocket, webovou aplikaci a další projekty, které budou popsány v dalších kapitolách.

Jak již bylo zmíněno v kapitole 3.3, webový klient je implementován za pomoci aplikačního rámce Angular, který využívá značkovacího jazyka HTML, kaskádových stylů a jazyka TypeScript, který se pro použití v prohlížečích překládá do jazyka JavaScript. Dále dle kapitoly 3.1 bylo rozhodnuto, že pro implementaci aplikace obsahující hlavní logiku celého systému, rozhraní REST API a rozhraní pomocí protokolu WebSocket bude využito aplikačního rámce ASP.NET Core pracujícího na platformě .NET 5 za pomocí jazyka C#. Veškeré další knihovny a aplikace (s výjimkou již zmíněného webového klienta) budou také implementovány v jazyce C# a poběží na platformě .NET 5.

4.1 Knihovny

Vzhledem k tomu, že celý systém obsahuje několik projektů bylo třeba vytvořit knihovny obsahující definice a logiku sdílenou napříč všemi projekty.

4.1.1 Základní knihovna s obecnými funkcemi

Knihovna pojmenovaná jako `AssocIS.Common` slouží ke sdílení obecné logiky, která by mohla být použitelná i v jiných projektech. Tato knihovna obsahuje logiku například pro práci s datem a časem, řetězci apod.

4.1.2 Knihovna pro práci s databází

Při implementaci práce s databází bylo využito přístupu tzv. Objektově-relačního mapování.

Objektově-relační mapování je technika, kdy jsou databázové tabulky mapovány na entity tříd programovacího jazyka. Díky tomu je uživateli umožněno pracovat nad databází bez nutnosti znalosti dotazovacího jazyka SQL[1] [9].

V projektu je využito implementace přístupu k databázi pomocí návrhového vzoru úložiště. Úložiště zapouzdřují logiku přistupující k datům a díky tomuto návrhovému vzoru

[1] SQL z anglického Structured Query Language

pak není třeba v další implementaci řešit, ze kterého zdroje data pochází. Stačí využívat rozhraní nabízené tímto úložištěm [13].

Kromě objektově-relačního mapování a návrhového vzoru úložiště je v knihovně využita ještě technika pro automatické generování databáze. Jedná se o přístup „nejprve kód, poté databáze“[2] a je využíván k vytváření a změnám databáze za pomoci migrací. Výhodou tohoto přístupu je, že není nutné pro práci s databází využívat různých nástrojů, nebo SQL dotazů přímo na databázový server. Díky tomu, že se o vše starají migrace (pokud jsou vytvářeny správným způsobem), je možné bezpečně zajistit změny databází bez nutnosti zásahu zkušené osoby [10].

Knihovna pojmenována jako *AssocIS.Database* slouží ke sdílení společné definice databázových entit, pomocných výčtů použitých v entitách, migrací a logiky implementující práci s databází.

4.1.3 Knihovna pro práci s daty

Hned po práci s databází je také potřeba zajistit společné třídy a výčty, které jsou používány napříč několika projekty. Pro tyto účely slouží knihovna pojmenovaná jako *AssocIS.Data*.

V této knihovně se nacházejí následující části:

- **Datové modely** – Jedná se o třídy, které obsahují definice dat, která se nacházejí na rozhraní REST API a na rozhraní protokolu WebSocket.
- **Výčty** – Definice výčtů, které se v aplikacích používají, ale nejsou použité v databázi.
- **Obecná logika** – Společná logika, která se používá k vyhodnocování v různých projektech nezávisle na jádru systému. Může se jednat například o výpočet dostupnosti položek majetku, algoritmus pro výpočet přítomnosti člena na prezenční listině apod.
- **Texty pro jazykovou mutaci** – Jednotná definice textů, které se používají v datových třídách na rozhraních. Nyní je podporován český a anglický jazyk. Jazykové mutace jsou v projektu zapsány pomocí tzv. prostředků. Prostředek jsou jakákoliv nespustitelná data (obrázky, řetězce apod.), která jsou při sestavení zabalena s aplikací [7].

4.1.4 Knihovny třetích stran

Kromě vlastních knihoven popsaných v kapitolách výše byly k implementaci využity také knihovny třetích stran. Knihovny třetích stran umožnily jednodušší a rychlejší vývoj celého systému.

Pro platformu .NET existuje balíčkovací systém NuGet vyvíjený a udržovaný společností Microsoft. K dnešnímu datu se odhaduje více než 100 000 knihoven („balíčků“) uložených v centrálním úložišti NuGet[3]. NuGet také umožňuje tyto balíčky spravovat v cloudových službách, nebo ve vlastních instancích. Jedná se tedy o největší platformu pro distribuci balíčků [8].

Při implementaci této práce byly z balíčkovacího systému NuGet použity tyto knihovny:

[2] V originálním znění se nazývá Code-First.

[3] Centrální úložiště NuGet – `https://www.nuget.org/`

- **Microsoft.AspNetCore**[4] – Nejdůležitější knihovna v celém informačním systému. Jedná se o tzv. meta-knihovnu[5], která v sobě udržuje veškeré knihovny pro aplikační rámec ASP.NET Core.
- **NSwag**[6] – Knihovna pro automatické vytváření specifikace OpenApi a poskytování prostředí Swagger UI pro testování API bez potřeby klientské aplikace.
- **BCrypt.Net-Next**[7] – Knihovna implementující hašovací funkci BCrypt.
- **Npgsql.EntityFrameworkCore.PostgreSQL**[8] – Knihovna „Entity Framework" pro objektově-relační mapování rozšířená o podporu databázového serveru PostgreSQL.
- **Microsoft.EntityFrameworkCore.Tools**[9] – Knihovna pro podporu databázových migrací.
- **Discord.NET**[10] – Knihovna implementující rozhraní služby Discord pro jednodušší implementaci bota.
- **WebSocket.Client**[11] – Knihovna implementující protokol WebSocket pro použití ze strany klienta.
- **Google.APIs**[12] – Sada knihoven pro integraci rozhraní REST API služeb Google. Jedná se o služby `Admin directory`, `Calendar` a `OAuth2`.
- **MailKit**[13] – Knihovna pro práci s elektronickou poštou.

4.2 Jádro systému

Jádrem systému se rozumí serverová aplikace poskytující rozhraní REST API a rozhraní na protokolu WebSocket. Tento projekt byl tudíž logicky pojmenován jako *AssocIS.Main*. Celé jádro, které běží na platformě .NET a aplikačním rámci ASP.NET Core, je postaveno do dvou hlavních pilířů aplikace. Tím jsou kontroléry a služby. Kontroléry poskytují implementaci rozhraní, tato implementace dále využívá služeb, které obsahují implementaci příslušných funkčních celků systému. Veškeré metody na rozhraní REST API mají adresu v následujícím formátu */api/v1/{název kontroléru}/...*, kde „v1" je verze rozhraní REST API. Vzhledem k tomu, že se jedná o první verzi aplikace, obsahuje konstantu 1.

[4] ASP.NET – `https://dotnet.microsoft.com/apps/aspnet`
[5] Meta-knihovna je knihovna bez zdrojového kódu obsahující pouze závislosti na ostatních knihovnách.
[6] NSwag – `https://github.com/RicoSuter/NSwag`
[7] BCrypt.Net-Next – `https://github.com/BcryptNet/bcrypt.net`
[8] Npgsql.EntityFrameworkCore.PostgreSQL – `https://github.com/npgsql/efcore.pg`
[9] EntityFrameworkCore.Tools (pro správnou funkčnost nástroje `dotnet ef`) – `https://www.nuget.org/packages/Microsoft.EntityFrameworkCore.Tools`
[10] Discord.NET – `https://github.com/discord-net/Discord.Net`
[11] WebSocket-Client – `https://github.com/Marfusios/websocket-client`
[12] Google.APIs – `https://github.com/googleapis/google-api-dotnet-client`
[13] MailKit – `http://www.mimekit.net/`

4.2.1 Přihlašování a oprávnění

Jak již bylo několikrát zmíněno, do systému by neměla mít přístup neoprávněná osoba. K tomu bylo zapotřebí implementovat ověření jak uživatelů, tak klientů, kteří k systému přistupují.

První fází, jak se do systému dostat, je získání přístupového klíče pro klientskou aplikaci, kterou se bude aplikace ověřovat. Tyto klíče jsou uloženy v databázi v tabulce *Clients* a ověřování probíhá, když se provádí přihlášení uživatele. Přístupový klíč je 32 znaků dlouhý řetězec ve formátu UUID[14].

Další fází je provedení přihlášení. Toto přihlášení se dá provést vícero způsoby. První způsob je přihlášení běžným způsobem. Tzn. pomocí uživatelského jména a hesla. V tomto případě klientská aplikace odesílá zadané údaje spolu s přístupovým klíčem. Systém poté ověří správnost klíče a pokračuje dále v procesu přihlášení.

Další způsob je přihlášení pomocí externí brány OAuth2. V tomto případě jsou klientské aplikaci vygenerovány přístupové odkazy na externí brány. Ale před tím, než se odkazy vygenerují, se provede ověření dostupnosti takové brány jednoduchým požadavkem protokolem HTTP a poté sestavením správného odkazu. Dalším krokem přihlášení pomocí OAuth2 je přesměrování klientského prohlížeče na stránky brány, kde uživatel provede přihlášení pomocí svých údajů. Poté, co se uživatel ověří na externí bráně, je přesměrován na adresu s metodou, která řeší získání údajů z externí brány a jejich následné zpracování. Poté, co jsou tyto údaje zpracovány a je ověřeno, že údaje poskytnuté bránou jsou pravé, je klient přesměrován na adresu, kterou specifikoval při volání na metodu, která generuje odkazy pro přechod na externí bránu. Současně je také v adrese zakódován identifikátor sezení pro daného uživatele. Poté je na klientské aplikaci, aby tento identifikátor sezení vyměnila za přístupový token pro daného uživatele a tím dokončila přihlášení. Při tomto posledním kroku se také kontroluje, že se do systému přihlašuje oprávněný uživatel, protože na externí bráně může mít účet také osoba, která nemá přístup do spolku. Tato metoda přihlášení je primární metodou, pokud je nějaká externí brána dostupná.

Po dokončení procesu přihlášení uživatele, ať už standardním způsobem, tak i pomocí externí brány OAuth2, je uživateli vygenerována sada přístupových tokenů. Tato sada obsahuje dva tokeny, a to přístupový token a obnovovací token. Přístupový token je ve formátu JWT[15], obnovovací token je náhodný řetězec. Řetězec tokenu ve formátu JWT se skládá ze tří částí, a to z hlavičky, těla a patičky. Hlavička obsahuje typ použité šifry a typ tokenu. V těle se nachází vlastní data, kde se nachází datum a čas výdeje a expirace tokenu, kdo token vydal, kdo má právo kontrolovat platnost tokenu a uživatelská data. Mezi uživatelská data patří jméno, příjmení, adresa elektronické pošty, uživatelské jméno, unikátní identifikátor a oprávnění v systému. Nakonec celý token obsahuje patičku, která obsahuje podpis vygenerovaný na základě použité šifry, privátního klíče a těla tokenu. Díky tomu je zajištěno, že pokud by se k přístupovému tokenu pokusila dostat neoprávněná osoba, modifikovat jej a opět s ním přistoupit k systému, tak systém zamezí přístupu na základě chybného podpisu [4].

Jak bylo zmíněno v předchozím odstavci, součástí tokenu jsou oprávnění uživatele. Tabulka 4.1 popisuje jednotlivá oprávnění.

[14]UUID z anglického Universally Unique Identifier
[15]JWT z anglického JSON Web Token.

Název oprávnění	Popis
`chairperson`	Oprávnění předsedy. Jedná se o nejvyšší oprávnění. Pokud má člen tohle oprávnění, tak se další oprávnění nenastavují.
`vice_chairperson`	Oprávnění místopředsedy.
`team_leader`	Oprávnění vedoucího týmu. Vedoucí týmu může spravovat svůj tým, vidět seznam týmů apod.
`user`	Oprávnění správce členů. Člen spolku s tímto oprávněním může spravovat ostatní členy, přístupy, pohovory, vytvářet týmy apod.
`event`	Oprávnění správce akcí. Člen spolku s tímto oprávněním může spravovat všechny akce a všechny komentáře.
`meeting`	Oprávnění správce schůzí. Člen spolku s tímto oprávněním může vytvářet a spravovat schůze.
`property`	Oprávnění správce majetku. Člen spolku s tímto oprávněním může vytvářet nové položky, zapisovat inventuru apod. Toto oprávnění není potřebné k vypůjčení a vrácení položky.
`finance`	Oprávnění správce financí. Člen spolku s tímto oprávněním může vytvářet a spravovat účty, dodavatele, spravovat všechny transakce apod. K vytvoření a správě vlastních transakcí není požadováno toto oprávnění.
`app`	Oprávnění správce aplikace. Neuděluje plná oprávnění jako předseda, jen umožňuje přístup k interním metodám správy systému.

Tabulka 4.1: Tabulka popisující oprávnění systému

Posledním způsobem přihlášení, který ale neslouží běžným uživatelům, nýbrž automatizovaným aplikacím je ověření pomocí rozhraní na protokolu WebSocket. Zde se očekává, že se aplikace připojí na rozhraní a do předem stanovené doby (ve výchozí konfiguraci 10 minut) provede své ověření. Ověření probíhá tak, že pomocí protokolu WebSocket pošle zprávu ve formátu JSON, která obsahuje přístupový klíč a celočíselnou hodnotu reprezentující bitovou masku položek reprezentujících zamýšlený účel klientské aplikace. Jedna aplikace může mít více účelů. Tyto účely jsou (v závorce je uvedena hodnota bitové masky):

- **Common** (`1`) – Účel získávání obecných událostí ze systému, jako například globální oznámení.
- **Meetings** (`2`) – Účel získávání událostí týkající se schůzí jako svolání schůze, zrušení schůze a propojení schůze se záznamem v kalendáři Google.
- **Users** (`4`) – Účel získávání událostí týkajících se vytvoření, modifikací, synchronizace členů a propojení nově vytvořeného účtu ve službě Google Workspace s členem spolku.
- **Teams** (`8`) – Účel získávání událostí týkajících se vytvoření, modifikací, smazání týmů a přiřazení identifikátoru role k týmu.
- **Events** (`16`) – Účel získávání událostí týkajících se vytvoření, modifikací, smazání akcí a vytvoření identifikátorů v externích systémech.

4.2.2 Lokalizace

Tato kapitola se bude zabývat implementací lokalizace na úrovni API. Veškerá hlášení, jako například chybová hlášení, upozornění apod., která opouští API, musí být připravena v různých jazykových mutacích. Podpora různých jazykových mutací je v aplikačním rámci ASP.NET Core obsažena v základu. Nicméně bylo zapotřebí rámci „říct“, že má s lokalizací počítat. Toho se docílilo registrací lokalizačních prostředků v metodě `ConfigureServices` nacházející se ve třídě `Startup` a v metodě `UseLocalization` implementované ve třídě `StartupExtensions`. Jak již bylo zmíněno, tak lokalizační balíčky se nachází v projektu `AssocIS.Data`.

Po registraci potřebných lokalizačních služeb a lokalizačních prostředků bylo zapotřebí nějak nastavit, jakou jazykovou mutaci vůbec zvolit. Navíc bylo zapotřebí, aby se vhodná jazyková mutace nastavila podle potřeby klienta nezávisle na jednotlivých požadavcích na rozhraní. Řešením nakonec byla implementace nacházející se ve třídě `CultureMiddleware`, která z požadavku zjistí potřebnou jazykovou mutaci a tu poté dále do datových struktur nastaví. Volba jazykové mutace klientem probíhá pomocí standardizované hlavičky HTTP `Accept-Encoding`.

Aktuálně je v rozhraní REST API podporován český a anglický jazyk.

4.2.3 Protokol WebSocket

Pro integraci klientských aplikací třetích stran i botů bylo zapotřebí využít protokolu umožňujícího komunikaci v reálném čase. K tomu nejlépe posloužil protokol WebSocket, jehož serverová část je podporována v aplikačním rámci ASP.NET Core. Tudíž bylo zapotřebí implementovat konkrétní komunikaci pomocí tohoto protokolu. Základním pilířem v této komunikaci jsou tzv. zprávy. V těchto zprávách se mohou nacházet instrukce pro ostatní aplikace, případně data. Součástí každé zprávy je vždy typ zprávy a nepovinný identifikátor zprávy. Tento identifikátor se používá v případě, že je zapotřebí provést obousměrnou komunikaci a poté tyto zprávy mezi sebou propojit. Pomocí protokolu WebSocket se zprávy odesílají jako řetězce ve formátu JSON[16], ve kterých jsou serializovaná data zpráv.

Jak již bylo zmíněno, každá zpráva obsahuje typ zprávy. V projektu jsou tyto typy definovány ve výčtu `WebSocketEventType`. Tyto typy jsou (v závorce je vždy uvedena celočíselná hodnota typu zprávy nacházející se ve zprávě):

- **Schůze** (`1`) – Jedná se o situaci, kdy jádro systému zasílá upozornění na svolanou, nebo zrušenou schůzi. Také pomocí této zprávy může znovu svolat již svolanou schůzi. V této zprávě se nachází datové struktury schůze, seznam očekávaných členů a identifikátor kalendáře Google (pokud existuje).
- **Ověření** (`2`) – Jedná se o jednorázový proces, který musí každý klient po připojení na rozhraní protokolu provést. Při tomto procesu je nutné zaslat svou identifikaci a účely popsané v posledním odstavci podkapitoly 4.2.1.
- **Vytvoření člena** (`3`) – Zpráva, která se odesílá na všechny klienty při vytvoření nového člena spolku v informačním systému. Součástí této zprávy jsou podrobné informace o členovi spolku.
- **Modifikace člena** (`4`) – Zpráva, která se odesílá na všechny klienty při úpravě člena spolku.

[16] JSON z anglického JavaScript Object Notation

- **Vytvoření týmu** (`5`) – Zpráva, která se odesílá na všechny klienty při vytvoření nového týmu.
- **Modifikace týmu** (`6`) – Zpráva, která se odesílá na všechny klienty při modifikaci stávajícího týmu.
- **Smazání týmu** (`7`) – Zpráva, která se odesílá na všechny klienty při smazání týmu.
- **Seznam rolí** (`8`) – Pomocná zpráva sloužící k získání seznamu rolí pomocí rozhraní REST API.
- **Vytvoření role** (`9`) – Pomocná zpráva, která obsahuje identifikátor role náležící k týmu.
- **Synchronizace členů** (`10`) – Zpráva, která se zasílá periodicky za účelem zajištění konzistence seznamu členů v informačním systému a v externích systémech.
- **Zpráva všem** (`11`) – Zpráva obsahující text, který má být doručen všem klientům.
- **Vytvoření externího identifikátoru** (`12`) – Pomocná zpráva sloužící k propojení identifikátoru z externího systému (například kalendáře Google) s identifikátorem záznamu v databázi.
- **Akce** (`13`) – Zpráva obsahující informace o vytvoření nové akce.

4.3 Webový klient

Po úspěšném dokončení jádra systému obsahujícího rozhraní REST API se mohlo přejít k implementaci webového klienta. Jak již bylo zmíněno v kapitole 3.3, pro implementaci webového klienta byl zvolen rámec Angular v kombinaci se šablonou AdminLTE.

Nejprve bylo zapotřebí založit projekt dle doporučené šablony pro projekty psané za pomocí rámce Angular. K tomu slouží nástroj zvaný Angular CLI[17], který slouží ke kompletní správě projektů vytvářených s využitím rámce Angular.

Po založení projektu bylo zapotřebí naimportovat šablonu do projektu pomocí balíkovacího systému NPM[18]. Po instalaci knihovny obsahující šablonu pak již stačilo do stávajících kaskádových stylů importovat kaskádové styly šablony.

Poté, co byla šablona úspěšně zapojena do aplikace, bylo možné začít implementovat webového klienta. Bylo třeba zajistit, aby jednotlivé části webové aplikace byly mezi sebou nezávislé a jednoduše rozšiřitelné. K tomuto účelu posloužily tzv. moduly. Modul v rámci Angular je sada komponent a služeb, které spolu souvisí. Tudíž každý logický celek systému obsahuje vlastní modul. Tyto moduly jsou implementovány tak, aby se výsledné sestavené skripty nestahovaly při startu aplikace, ale až při potřebě jejich použití.

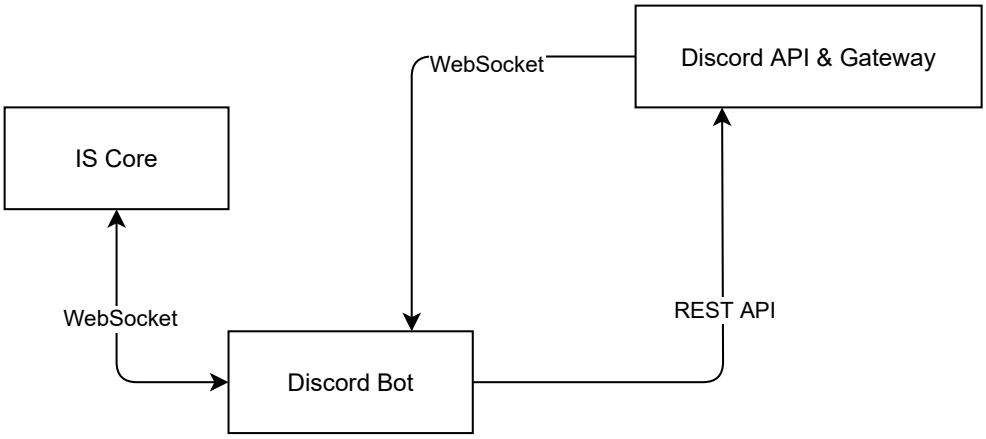
Výsledek implementace jednotlivých sekcí vycházející z návrhu v podkapitole 3.4 je k dispozici k nahlédnutí v příloze B.

[17] Angular CLI – `https://angular.io/cli`
[18] NPM z anglického Node Package Manager

4.4 Klient služby Discord (Discord bot)

Pro integraci služby Discord byla zvolena strategie, kdy není logika bota implementována jako součást jádra systému, ale jako samostatná aplikace. Díky tomu bylo dosaženo jednodušší a rozšiřitelné implementace a současně pojištění proti výpadkům externí služby. Díky tomuto rozdělení je také možné nasadit systém i do spolků, které nevyužívají těchto externích systémů. Pokud totiž nepojede bot z důvodu výpadku služby, tak nebude ovlivněn samotný systém.

Discord bot je implementován v projektu `AssocIS.Discord.Bot`. Jedná se o konzolovou aplikaci, která běží nepřetržitě. Základem celého bota je použití protokolu WebSocket pro propojení služeb Discord a jádra systému a rozhraní REST API pro zasílání požadavků směrem na službu Discord dle popisu na obrázku 4.1.



Obrázek 4.1: Diagram popisující komunikaci Discord bota

Zprávy

Jak již bylo zmíněno, komunikace botů s jádrem systému probíhá pomocí protokolu WebSocket. Díky tomu Discord bot reaguje na následující příchozí zprávy:

- **Schůze** (`1`) – Po vytvoření schůze v systému a přijetí této zprávy botem se provedou následující kroky:
  1. Zjistí se, zda je schůze povinná, povinná pro pozvané, nebo nepovinná.
     - Pokud je schůze povinná, pak se upozorňují všichni členové pomocí speciálního označení všech uživatelů.
     - Pokud je schůze nepovinná, nebo je povinná jen pro očekávané, pak se upozorňují do soukromé konverzace.
  2. Provede se sestavení zprávy na základě informací doručených ve zprávě.
  3. Pokud je tato vygenerovaná zpráva delší, než 2000 znaků, pak je třeba ji rozdělit do více zpráv. Nutnost rozdělit zprávu do více zpráv plyne z omezení služby Discord [3].
  4. Nakonec je zpráva odeslána do příslušného kanálu, nebo soukromé konverzace.

- **Vytvoření člena** (3) – Po vytvoření účtu novému členovi spolku je možné udělit přístup na server Discord, pokud se na serveru nachází. Toho se docílí přiřazením správné množiny rolí, která je definovaná v konfiguračním souboru bota.
- **Modifikace člena** (4) – Pokud někdo v systému provede úpravu člena, pak je třeba zajistit konzistentní stav i na straně služby Discord. Pokud například člen opustil spolek a tato informace byla do systému zapsána, pak by měl (již bývalý) člen ztratit přístup na server.
- **Vytvoření týmu** (5) – Po vytvoření nového týmu je zapotřebí zajistit konzistenci seznamu týmů i na straně služby Discord. Toho lze docílit vytvořením příslušné role pro tým. Pokud byl při vytvoření role určen vedoucí týmu, pak uživatel obdrží role reprezentující vedoucího týmu.
- **Modifikace týmu** (6) – Pokud dojde ke změně týmu (změna vedoucího, nebo změna názvu týmu), pak je třeba udržet konzistenci mezi daty v systému a nastavením Discord serveru. Pokud dojde ke změně názvu týmu, pak dojde k přejmenování příslušné role. Pokud dojde ke změně vedoucího týmu, pak se detekuje, zda je původní vedoucí vedoucím ještě nějakého týmu a podle toho jsou mu odebrány příslušné role.
- **Smazání týmu** (7) – Pokud dojde ke smazání týmu, tak se na straně služby Discord provede pouze smazání role. Discord poté takovou roli všem uživatelům (nositelům) odebere.
- **Seznam rolí** (8) – Příchozí požadavek z jádra systému, aby bot získal z Discord serveru seznam rolí a zaslal je zpět jádru, které následně prezentuje tyto role uživateli.
- **Synchronizace členů** (10) – Periodicky zasílaná zpráva se seznamem všech členů ve spolku. Slouží k tomu, aby se udržela konzistence všech členů i v případě výpadků služeb. Synchronizace členů probíhá následujícím postupem:
  1. Prochází se jednotliví uživatelé a detekuje se, zda je člen stále na serveru. Pokud není, pokračuje se k dalšímu.
  2. Provede se výpočet očekávaných rolí pro daného člena.
  3. Prochází se jednotlivé role uživatele a provádí se následující kroky:
     (a) Zjišťuje se, zda role, kterou má uživatel je mezi rolemi, které se mají sledovat. Pokud není, tak se role ignoruje.
     (b) Pokud je daná role mezi sledovanými, tak se zjišťuje, zda danou roli má vůbec uživatel mít. Pokud ano, tak se ignoruje.
     (c) Pokud není splněn žádný z předchozích bodů, je uživateli role odebrána.
     (d) Nakonec se zjišťuje, které role uživatel nemá a měl by mít. Takové role jsou mu přiřazeny.
- **Zpráva všem** (11) – Zpráva, ve které je obsažena zpráva adresována všem členům spolku. Zasílá se do společného kanálu spolku se speciálním označením odesílajícím upozornění všem členům na serveru.
- **Akce** (13) – Po založení nové akce je možné říct botovi, aby založil nový kanál. Tyto kanály by měly být určené pro živou diskuzi v reálném čase, včetně posílání příloh.

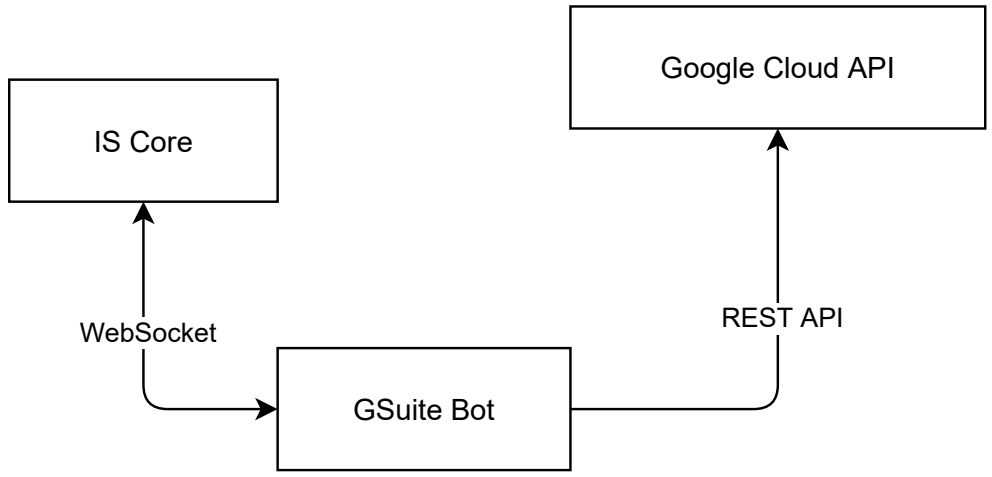
Zpracování zpráv z protokolu WebSocket je implementováno ve třídách popsaných v tabulce 4.2:

Typ zprávy	**Třída implementující zpracování zprávy**
Schůze	`MeetingManager`
Vytvoření a modifikace člena	`UserManager`
Vytvoření, modifikace a smazání týmu	`TeamManager`
Synchronizace členů	`UserManager`
Zpráva všem	`SystemManager`
Akce	`EventManager`

Tabulka 4.2: Tabulka popisující rozložení implementace jednotlivých tříd v závislosti na typu zprávy

4.5 Klient služby Google (GSuite bot)

Princip implementace GSuite bota je podobný, jako implementace Discord bota. Implementace je provedena jako samostatná aplikace a komunikace probíhá na protokolu WebSocket, ale narozdíl od Discord bota komunikace se službami Google neprobíhá pomocí protokolu WebSocket, ale pomocí rozhraní REST API, jak je tomu také popsáno na obrázku 4.2. Protokol WebSocket se používá pouze na propojení s jádrem systému. Implementace GSuite bota se nachází v projektu `AssocIS.Google.Bot`.



Obrázek 4.2: Diagram popisující komunikaci GSuite bota

Zprávy

Podobně jako Discord bot i GSuite bot přijímá zprávy z jádra systému pomocí protokolu WebSocket. Nicméně oproti Discord botu reaguje na zprávy uvedené níže jiným způsobem.

- **Schůze** (1) – Po vytvoření schůze se provede zapsání události do kalendáře Google patřícího spolku. Pokud byla schůze zrušena, pak se událost reprezentující tuto schůzi automaticky smaže.

- **Vytvoření člena** (`3`) – Po vytvoření člena v systému se provede jeho automatické propsání do adresářových služeb Google. Díky tomu člen získá okamžitý přístup do prostředí spolku v GSuite.
- **Modifikace člena** (`4`) – Po úpravě člena se do adresářových služeb Google propíšou jeho nové údaje. Pokud člen opustil spolek, pak bude jeho účet pozastaven.
- **Synchronizace členů** (`10`) – Periodicky zasílaná zpráva se seznamem všech členů. Stejně jako u Discord bota slouží k tomu, aby se udržela konzistence členů i v případě výpadků služeb. Princip synchronizace je oproti službě Discord jednodušší, stačí zaslat data na rozhraní REST API adresářových služeb Google.
- **Akce** (`13`) – Podobně jako u schůzí se vytvoření nové akce propisuje do kalendáře Google a opět se při smazání akce provede automatické smazání v kalendáři.

Po dokončení zpracování zpráv, které provádí vytváření (tzn. vytvoření schůze, člena a akce), se provede dodatečné odeslání zprávy typu `12` popsané v kapitole 4.2.3. Součástí této zprávy bude identifikátor ze služeb Google, aby bylo možné zpětně dohledat správnou událost v kalendáři, nebo adresářových službách.

Zpracování zpráv z protokolu WebSocket je implementováno ve třídách podle tabulky 4.3:

Typ zprávy	**Třída implementující zpracování zprávy**
Schůze	`MeetingManager`
Vytvoření, modifikace člena	`UserManager`
Akce	`EventManager`
Synchronizace členů	`UserManager`

Tabulka 4.3: Tabulka popisující rozložení implementace jednotlivých tříd v závislosti na typu zprávy v projektu GSuite bota.

Kapitola 5

Testování

Tato kapitola pojednává o další důležité části celého vývoje systému, a tím je testování. Během testování je kladeno za cíl co nejlépe systém vyzkoušet a případně jeho nedostatky zapracovat a opravit chyby. Toto testování proběhlo ve dvou fázích, podle toho, která část systému byla testována.

5.1 Testování při vývoji

Testování při vývoji je jedna z prvních fází, jak systém otestovat, aby fungoval tak, jak má. Tato fáze si bere jako hlavní cíl opravy chyb, vylepšení, nebo dořešení funkčních celků, které mohou být při implementacích přehlédnuty. Celý proces testování během vývoje byl rozdělen do pod-fází, které jdou společně s postupem implementace.

- **Databáze, rozhraní REST API a protokol WebSocket** – Pro otestování funkčnosti rozhraní REST API bylo využito specifikace OpenAPI a prostředí Swagger UI, které vytváří jednoduché prostředí pro tyto účely. Díky tomu bylo možné testovat rozhraní REST API bez nutnosti implementace jakékoliv klientské aplikace. Pro otestování správné funkčnosti rozhraní na protokolu WebSocket bylo využito ručně psaných skriptů spouštěných v prohlížeči.
- **Webový klient** – Během vývoje webového klienta bylo možné provést podrobnější testování rozhraní REST API, které webový klient využívá. Současně již také během vývoje bylo možné otestovat hotové části systému.
- **Discord a GSuite boti** – Podobně jako u webového klienta bylo díky botům možné podrobněji otestovat rozhraní na protokolu WebSocket. Bohužel vzhledem k povaze externích systémů (nestabilita, postupující vývoj apod.) není možné zaručit stabilní funkčnost těchto aplikací, tudíž je zapotřebí jejich neustálá údržba. Primárně je však nutné věnovat pozornost jednotlivým informacím o vydáních rozhraní k externím systémům a vyhodnocovat, zda je nutné provést úpravy.

Jak bylo zmíněno, tak během tohoto testování byly kladeny cíle zaměřené primárně na opravy chyb, vylepšení, nebo dořešení funkčních celků, které se díky implementacím dalších částí systém alespoň mohly lépe otestovat. Po dokončení implementací tak bylo možné systém poskytnout k testování uživatelům, jak je popsáno v následující podkapitole.

5.2 Testování na uživatelích

Jak bylo zmíněno, tak tato podkapitola se bude bude věnovat testování s budoucími uživateli. Jako tito uživatelé byli vybráni členové Studentské Unie FIT. Celý postup testování se skládal z následujících kroků:

1. Všichni členové byli obeznámeni o možnosti testování nově implementovaného systému včetně informací, kde je systém nasazen. Podmínkou pro zúčastnění tohto testování bylo sdělit svůj školní login (login ve formátu `xloginXX`) a svůj účet na serveru GitHub[1].

2. Každý člen, který projevil zájem o zapojení se do testování, pak obdržel dvojici přístupových údajů. První přístup byl vytvořen přímo pro daného člena a nesl přístupový login daného člena. Druhý přístup sloužil jako univerzální přístup na účet, který nesl oprávnění uživatele s nejvyššími oprávněními. To z důvodu, aby si každý člen mohl „osahat“, co vlastně systém umí.

3. Po obdržení údajů uživatel dostal také pozvánku do soukromého repozitáře na serveru GitHub, kam mohl psát svoji zpětnou vazbu z testování.

4. Tato zpětná vazba poté byla zpracována do výsledků, jak je tomu popsáno v další podkapitole.

Požadované údaje (jméno na serveru GitHub a školní login) byly použity výhradně k účelům ve výše popsaných bodech.

5.2.1 Výsledky z testování na uživatelích

Po provedení testování na uživatelích byly sumarizovány následující výsledky. Studentská Unie FIT má aktuálně přibližně 70 členů, z toho se do testování přihlásilo 15 členů. Z těchto patnácti členů systém nakonec nepoužili dva členové. Zbytek se do systému alespoň jednou přihlásil a celý systém si vyzkoušel. Někteří členové šli dokonce více do hloubky, než pouhá práce s uživatelským rozhraním. Ve výsledku systém obstál bez nějakých závažnějších problémů. Ale samozřejmě zde byly nějaké připomínky, které jsou popsány v bodech níže.

- Chyba – Vyřešeno – Podezřelý kód HTTP při neplatném požadavku po návratu z brány OAuth.
- Návrh – Do další verze – Některé prvky na stránce obsahují nevhodné texty. Za nevhodné texty jsou považovány takové texty, které v daném kontextu nedávají příliš smysl.
- Požadavek – Do další verze – Umožnit deaktivaci jednotlivých funkčních celků systému.
- Vylepšení – Do další verze – Provádět kontrolu rozsahu data a času (Od – Do) už na klientovi a ušetřit tak práci API.
- Chyba – Vyřešeno – Token pro sezení na bráně OAuth zůstává platný i po odhlášení.

[1] GitHub – `https://github.com/`

- Chyba – Do další verze – Pokud je aplikace nasazena jako kontejner Docker a je vytvořen nový kontejner, pak jsou zneplatněna všechna přihlášení. Webová aplikace by měla uživatele alespoň automaticky odhlásit.
- Chyba – Vyřešeno – Nemožnost měnit informace o uživatelích i když měl uživatel oprávnění
- Chyba – Vyřešeno – Ve finančních reportech je zaměněn text typů transakce.
- Požadavek – Vyřešeno – Zachytávání chyb ze serveru a jejich přehledná prezentace uživateli.
- Chyba – Vyřešeno – Nesprávné zvýraznění validace v přihlašovacím formuláři.
- Chyba – Vyřešeno – Pokud byl uživatel přesměrován na externí bránu OAuth2 Google, kde měl ale přihlášený jeden účet, který současně nepatřil do organizace, pak přihlášení skončilo s chybou na straně Google.

Připomínky, které jsou označeny jako „Vyřešeno“, byly implementovány během vytváření této práce. Připomínky, které jsou označeny, že budou zapracovány „Do další verze“ budou implementovány v budoucím vývoji celého systému.

Kapitola 6

Nasazení

Tato kapitola se bude zabývat nasazením výsledného systému a botů do produkčního prostředí.

Jak již bylo zmíněno, celý systém se dělí do několika aplikací, které jsou mezi sebou navzájem propojeny. Vzhledem k tomu, že API a webová aplikace jsou jádrem celého systému a webová aplikace slouží uživatelům k ovládání celého systému, by měly být nasazeny na stroji, který bude uživatelům přístupný buď po lokální síti, nebo pomocí veřejné adresy, nebo domény.

Ostatní aplikace jako boti nepotřebují veřejně přístupný bod a vystačí si s běžným připojením na Internet.

6.1 Konfigurace

Aby celý systém řádně fungoval, je zapotřebí provést prvotní konfiguraci. Nejdříve je nutné nakonfigurovat hlavní službu a poté lze konfigurovat a spouštět boty.

Jako první bod, aby systém mohl fungovat, je potřeba databázový server PostgreSQL. Instalaci databázového serveru je možné provést v prostředí Docker, nebo je možné jej nainstalovat přímo jako službu v operačním systému[1].

Pokud spolek plánuje využívat přihlašování pomocí externích systémů (Google nebo Discord), musí provést prvotní úkony pro registraci své aplikace v těchto službách.

6.1.1 Vytvoření aplikace ve službě Discord

Pro vytvoření aplikace ve službě Discord, která poté současně bude sloužit i jako aplikace pro bota, je třeba splnit několik následujících kroků:

1. Navštivte vývojářský portál Discord a přihlašte se pod účtem, který bude s aplikací spárován[2].
2. Otevřete seznam aplikací v levém horním menu.
3. V pravém horním menu naleznete tlačítko „Nová aplikace“ („New Application“), klikněte na něj.

[1] PostgreSQL – Download `https://www.postgresql.org/download/`

[2] Discord Developer Portal – `https://discord.com/developers/docs/intro`

4. Vyplňte název aplikace a potvrďte zadání[3].

5. *(Volitelný krok)* Pokud máte zájem, můžete aplikaci nastavit profilovou fotografii (nejedná se o profilovou fotografii bota) a popis.

6. V levém menu vyberte sekci „OAuth2“.

7. V této sekci se nachází seznam definic adres, které jsou povoleny pro přesměrování z brány OAuth2 („Redirects“).
   Zde vyplňte adresu ve formátu `http://{domena}[:port]/api/v1/oauth2/callback`. Pokud má vaše aplikace zabezpečené připojení pomocí protokolu HTTPS, pak nahraďte použitý protokol `http` protokolem `https`. Zadání portu v adrese je zapotřebí pouze pokud systém běží na jiném portu než 80, nebo 443. Poté proveďte uložení změn stisknutím tlačítka „Uložit změny“ („Save changes“).

8. Opište si klientské ID („Client ID“) a privátní klíč („Client Secret“). Budete je zadávat do konfigurace systému.

Tímto procesem byla založena aplikace ve službě Discord. Další kroky následují, pokud budete zakládat i bota pro Discord server spolku. Pro založení bota je nehledě na bránu OAuth2 je zapotřebí provést kroky 1 až 5.

9. Ve vývojářském portálu ve vaší aplikaci otevřete v levém menu sekci „Bot“.

10. Stiskněte tlačítko „Přidat bota“ („Add Bot“)[4].

11. *(Volitelný krok)* Pokud máte zájem, můžete botovi nastavit profilovou fotografii a novou přezdívku[5].

12. *(Volitelný krok)* Pokud si přejete zajistit, aby si někdo jiný nepřidal bota na svůj server, vypněte možnost veřejného bota („Public bot“) v podsekci kontroly přístupu („Authorization Flow“).

13. Pro správnou funkčnost bota je zapotřebí zapnout oprávnění, aby bot viděl uživatele. Tudíž v sekci záměry brány („Privileged Gateway Intents“) zapněte obě možnosti.

14. Běžte pomocí menu do sekce OAuth2 a vygenerujte si odkaz, díky kterému pozvete svého bota na server spolku. V poli oprávnění OAuth2 vyberte pouze možnost „bot“. Objeví se vám seznam zaškrtávacích polí pro oprávnění bota. Vyberte pouze oprávnění „Administrator“[6].

15. Vytvořený odkaz si zkopírujte a vložte do adresního řádku ve vašem prohlížeči. Měl by se vám zobrazit na výběr seznam serverů, na který můžete bota přidat. Pokud nevidíte server v seznamu serverů, pak nemáte oprávnění pro správu na daném serveru a musíte kontaktovat administrátora, aby vám příslušné oprávnění udělil, nebo provedl přidání

[3] Kliknutím na tlačítko „Vytvořit“ („Create“) souhlasíte s podmínkami použití služby Discord a jejich rozhraní.

[4] Vytvoření bota je nevratné. Pokud po této operaci budete chtít odebrat bota, budete muset odebrat celou aplikaci.

[5] Přezdívka se nemusí shodovat s názvem aplikace.

[6] Pokud si nejste jisti, že máte oprávnění správně nastavena, můžete provést kontrolu se snímkem v příloze A.

bota za vás. Vyberte svůj server a pokračujte, dále zkontrolujte, že bot vyžaduje oprávnění „Administrator“ a potvrďte přidání bota. V danou chvíli by se měl bot objevit na vašem serveru.

16. Vraťte se do sekce „Bot“ a zkopírujte si přístupový token, který budete zadávat do konfigurace bota.

6.1.2 Vytvoření aplikací na platformě Google Cloud

Před konfigurací přístupu na API služeb Google je zapotřebí, aby měl spolek platnou licenci v Google Workspace a nakonfigurovanou organizační jednotku[7]. Současně pro možnost konfigurace budete v organizaci potřebovat oprávnění SuperAdministrátor.

Aby bylo možné využít přihlášení pomocí brány OAuth2 od Google, je nutné provést několik následujících kroků:

1. Otevřete konzoli platformy Google Cloud[8] a přihlašte se pod účtem organizace, který chcete mít, nebo máte propojený s aplikací.
2. Vyberte projekt odpovídající vaší organizaci, kde chcete mít přístupové údaje.
3. V levém menu otevřete sekci „APIs & Services/Credentials“.
4. Klikněte na tlačítko „Create Credentials“ pro vytvoření přístupových údajů a vyberte možnost „OAuth client ID“.
5. Jako typ aplikace vyberte webovou aplikaci („Web Application“).
6. Vyplňte název aplikace a autorizované odkazy pro přesměrování. Pokud jste vytvářeli aplikaci ve službě Discord popsanou v podkapitole 6.1.1, odkaz pro přesměrování zadaný ve službě Discord je stejný, jako budete zadávat do „Authorised redirect URIs“.
7. Klikněte na tlačítko pro uložení změn.
8. Opište si klientské ID („Client ID“) a privátní klíč („Client secret“). Budete je zadávat do konfigurace systému.

Tyto kroky, jak již bylo zmíněno, dostačují k vytvoření možnosti přihlášení.

Zpřístupnění Kalendáře Google

Pro zpřístupnění funkcí pro práci s kalendářem, které nabízí bot, proveďte následující kroky:

1. Otevřete konzoli platformy Google Cloud a přihlašte se pod svým účtem organizace, který chcete mít, nebo máte propojený s aplikací.
2. Vyberte projekt odpovídající vaší organizaci, pomocí kterého budete přistupovat k vašemu kalendáři[9].
3. V levém menu otevřete sekci „APIs & Services/Dashboard“.

[7] Součástí této práce je pouze vytvoření přístupů do platformy Google Cloud pro organizaci.
[8] Google Cloud Platform – `https://console.cloud.google.com/`
[9] Pokud jste registrovali aplikaci pro přístup OAuth2, doporučuje se použít stejnou aplikaci.

4. Klikněte na tlačítko „Enable APIs and Services“, vyhledejte Google Calendar API a stiskněte tlačítko „Enable“.

Předchozím krokem došlo k připojení Google Calendar API k vaší aplikaci. Dále je zapotřebí vytvořit přístupové údaje.

5. V otevřené sekci Google Calendar API v levém menu navštivte sekci „Credentials“.
6. Klikněte na tlačítko „Create credentials“ a následně na možnost „Service account“[10].
7. Vyplňte název, který je povinný. Pokud budete chtít, můžete si k servisnímu účtu připojit popis a stiskněte tlačítko „Done“. Poté budete přesměrováni zpět na seznam přístupů.
8. Otevřete detail právě vytvořeného servisního účtu.
9. (a) Na kartě „detail“ se ve spodní části nachází „Show domain-wide delegation“. Otevřete jej, zaškrtněte „Enable G Suite Domain-wide Delegation“, vyplňte požadované údaje a stiskněte tlačítko „Save“.
   (b) Na kartě „keys“ klikněte na tlačítko „Add key“, vyberte možnost „Create new key“. V modálním okně ponechte zaškrtnutou možnost „JSON“ a stiskněte tlačítko „Create“. Po vytvoření nového klíče vám bude klíč stažen ve formátu JSON. Uložte si jej, údaje z tohoto souboru budete zadávat do konfigurace bota.

Zpřístupnění adresářových služeb

Adresářové služby ve službách Google nabízí možnost práce se členy v organizaci. Pro zpřístupnění funkcí pro práci s adresářovými službami jsou první tři kroky totožné s kroky popsanými v podkapitole 6.1.2, dále pokračujte následujícími kroky:

- Klikněte na tlačítko „Enable APIs and Services“, vyhledejte „Admin SDK Api“ a stiskněte tlačítko „Enable“.
- Přejděte na administrátorskou konzoli Google Workspace[11] a přihlašte se pod svým účtem organizace.
- V menu vyberte sekci „Zabezpečení“.
- Na konci stránky se nachází sekce „Ovládací prvky rozhraní API“. Otevřete ji.
- Na konci této podsekce se nachází možnost s názvem „Zmocnění přístupu k celé doméně“, otevřete ji.
- V této sekci přidejte novou aplikaci. Bude třeba vyplnit identifikátor servisního účtu, který byl vytvořen dle postupu popsaného v podkapitole 6.1.2[12]. Kromě identifikátoru bude třeba vyplnit také rozsahy OAuth. Pro správu členů bude zapotřebí nastavit rozsahy z rozhraní „Admin Directory API“[13].

[10] Service account vytváří speciální servisní účet pro aplikace, které nemají možnost uživatelské interakce, vč. možnosti zadávání přihlašovacích údajů.
[11] Google Admin – `https://admin.google.com`
[12] Identifikátor aplikace je k nalezení v detailu servisního účtu na kartě „Details“.
[13] Rozsahy OAuth2.0 – `https://developers.google.com/identity/protocols/oauth2/scopes`

6.2 Prostředí Docker

Docker je projekt s otevřeným zdrojovým kódem pro jednodušší správu a izolaci aplikací do kontejnerů, které je možné spustit lokálně, nebo také v cloudových řešeních. Docker je také společnost, která tuto technologii vyvíjí a propaguje. Pracují ve spolupráci s komunitou, včetně vývojářů ze společností jako Google nebo Microsoft. Cílem dockeru je tzv. „odlehčená virtualizace“, kde obraz a kontejner z něho běžící si s sebou nese pouze potřebné soubory k provozu, a tím se snižuje režie a zvyšuje bezpečnost [2]. Z důvodu nižší režie a vyšší bezpečnosti (díky izolaci) je nasazení pomocí kontejnerů Docker doporučeným nasazením.

Obraz pro Docker je sestavený balík s aplikací, ze kterého se vytváří kontejnery, ve kterých aplikace ve výsledku běží. Distribuce takových obrazů je poté jednoduchá díky distribuované síti registrů ukládající obrazy.

Po získání obrazu je možné kontejner spustit příkazem[14]:

```
docker run --name AssocIS -d -p 5000:5000 --env-file /env_file image_name
```

- `--name` je unikátní název vytvořeného kontejneru. Jedná se o volitelnou část příkazu, nicméně umožní jednodušší vyhledání kontejneru za účelem správy.
- `-d` řekne prostředí Docker, že se má kontejner spustit na pozadí. Pokud nebude parametr `-d` zadán, tak veškerý standardní (i chybový) výstup bude nasměrován do konzole a jediná možnost, jak se vrátit do konzole, bude zastavit kontejner.
- `-p 5000:5000` je otevření portu `5000` v hostitelském systému na port `5000` v kontejneru. Port v hostitelském systému bude dostupný na síti zavoláním na IP adresu počítače se zadaným portem. Pokud si přejete omezit port na danou IP adresu, tak zadejte `xxx.yyy.abc.def:5000:5000`, kde `xxx.yyy.abc.def` je IP adresa.
- `--env-file /env_file` je cesta (relativní, nebo absolutní) k souboru konfigurujícímu proměnné prostředí. Je požadován pro správnou konfiguraci systému.
- `image_name` je název obrazu.

Celý systém je rozdělen na následující obrazy:

- `AssocIS-Main` – Hlavní jádro informačního systému včetně klientské webové aplikace.
- `AssocIS-DC_Bot` – Discord bot pro integraci služby Discord.
- `AssocIS-Google-Bot` – Google bot pro integraci služeb Google.

6.2.1 Konfigurace informačního systému

Tato podkapitola se bude věnovat správnému nastavení jednotlivých konfiguračních souborů v souborech pro proměnné prostředí.

[14] V příkazu jsou uvedeny pouze nejdůležitější přepínače a parametry pro vytvoření a spuštění kontejneru. Pro více informací lze navštívit oficiální dokumentaci - `https://docs.docker.com/engine/reference/commandline/run/`

Informační systém

Konfigurační soubor pro správné nastavení informačního systému obsahuje následující položky. Pokud nebude určeno jinak, jedná se o povinnou položku.

- **Základní sekce**
  - `ConnectionStrings:Default` – Konfigurační řetězec pro připojení k databázi. Zadává se ve formátu `Host=xxx.yyy.aaa.bbb;Database=AssocIS;Username=postgres;Password=password`, kde `xxx.yyy.aaa.bbb` je IP adresa databázového serveru a `password` je heslo k databázovému serveru.
- **Nastavení hesla**
  - `ResetPassword:MinimalLength` – Minimální délka hesla, která může být členovi vygenerována. Jedná se o volitelnou položku. Výchozí hodnota této položky je nastavena na 10 znaků.
  - `ResetPassword:MaximalLength` – Maximální délka hesla, která může být členovi vygenerována. Jedná se o volitelnou položku. Výchozí hodnota této položky je nastavena na 20 znaků.
- **Elektronická pošta**
  - `Email:SenderAddress` – Adresa odesílatele při odesílání elektronické pošty[15]
  - `Email:SenderName` – Jméno odesílatele při odesílání elektronické pošty.
  - `Email:Smtp:Address` – Adresa serveru SMTP k odesílání elektronické pošty.
  - `Email:Smtp:Port` – Port serveru SMTP k odesílání elektronické pošty.
  - `Email:Smtp:Username` – Uživatelské jméno sloužící k ověření oprávněného uživatele na serveru SMTP[16].
  - `Email:Smtp:Password` – Přístupové heslo sloužící k ověření oprávněného uživatele na serveru SMTP.
- **Schůze**
  - `Meeting:WarningAbsentCount` – Minimální počet absencí, aby informační systém začal u člena spolku vydávat varování, že má člen vysokou absenci. Jedná se o volitelnou položku. Výchozí hodnota této položky je nastavena na 5 absencí.
- **OpenAPI**
  - `OpenApi:Servers:0` – Doménová adresa sloužící ke správnému fungování specifikace OpenAPI. V produkčním prostředí se jedná se o nepovinnou položku. Pokud nebude adresa zadána, tak pouze nebude možné testovat rozhraní REST API pomocí specifikace OpenApi, pokud se server nachází za reverzní proxy jiného webového serveru.

[15] Elektronická pošta se používá při svolání a zrušení schůze.
[16] Z důvodu bezpečnosti není možné se připojit k serveru SMTP s anonymním přístupem

- **OAuth2 – přihlašování pomocí služby Discord**. Pokud informační systém nebude využívat přihlašování pomocí služby Discord, pak není třeba tento konfigurační blok vyplňovat a stačí nastavit `OAuth:Discord:Disabled` na hodnotu `true`.
  - `OAuth:Discord:ClientId` – Klientské ID aplikace pro přístup ke službě Discord.
  - `OAuth:Discord:ClientSecret` – Privátní přístupový klíč ke službě Discord sloužící k ověření uživatele po návratu z přístupové brány.
  - `OAuth:Discord:RedirectUrl` – Adresa, na kterou má být uživatel přesměrován po dokončení přihlášení ve službě Discord. Je třeba ji vyplnit ve správném formátu (http://AdresaIS/**api/v1/auth/oauth2/callback**). Zvýrazněná část adresy se musí v adrese nacházet.
- **OAuth2 – přihlašování pomocí služeb Google**. Pokud informační systém nebude využívat přihlašování pomocí služeb Google, pak není třeba tento konfigurační blok vyplňovat a stačí nastavit `OAuth:Google:Disabled` na hodnotu `true`.
  - `OAuth:Google:ClientId` – Klientské ID aplikace pro přístup ke službám Google.
  - `OAuth:Google:ClientSecret` – Privátní přístupový klíč ke službám Google sloužící k ověření uživatele po návratu z přístupové brány.
  - `OAuth:Google:RedirectUrl` – Adresa, na kterou má být uživatel přesměrován po dokončení přihlášení ve službách Google. Je třeba ji vyplnit ve správném formátu (http://AdresaIS/**api/v1/auth/oauth2/callback**). Zvýrazněná část adresy se musí v adrese nacházet.

Discord bot

Konfigurační soubor pro správné natavení Discord bota obsahuje následující položky. Stejně, jako u informačního systému, pokud nebude určeno jinak, jedná se o povinnou položku.

- **Základní nastavení**
  - `Core:ClientId` – Jednoznačný identifikátor aplikace přidělený informačním systémem.
  - `Core:Uri` – Adresa rozhraní WebSocket informačního systému. Jedná se o volitelnou položku, pokud bot běží jinde než informační systém. Za jiné umístění se považuje jiný kontejner Docker, nebo jiný server. Výchozí hodnota této položky je `ws://localhost:5000/ws`.
  - `Events` – Možnost deaktivace jednotlivých vlastností bota. Jedná se o základní chod (nedporučuje se deaktivovat), schůze, týmy, správa členů a akce.
- **Konfigurace Discord serveru**
  - `Discord:Token` – Přístupový token pro připojení se na službu Discord. Tento token bylo možné získat z vývojářského portálu Discord popsaného v podkapitole 6.1.1.
  - `Discord:LogLevel` – Minimální úroveň logování, která se má na standardní výstup provádět. Jedná se o celočíselnou hodnotu, jejíž hodnoty reprezentují jednotlivé úrovně. 0 značí kritické chyby, 1 jsou méně kritické chyby, 2 jsou varování, 3 jsou informační zprávy, 4 a 5 reprezentují podrobné ladící výpisy. Jedná se

o volitelnou položku a výchozí hodnota této položky je 2, což znamená, že se na standardní výstup budou vypisovat až varování.

- `Discord:HomeGuildId` – Identifikátor serveru, ve kterém bot primárně běží[17].
- `Discord:NotificationChannelId` – Identifikátor kanálu, do kterého se posílají společná oznámení.
- `Discord:RoleGroups` – Konfigurace skupin rolí, které mají být dle připravených situací nastaveny členovi spolku. Mezi takové situace patří například přidání vedoucího týmu, nastavení předsedy, přechod člena ze zkušební doby do řádného členství apod.
- `Discord:TemporaryChannelsCategoryId` – Identifikátor kategorie kanálů, ve které může bot na základě příkazů z informačního systému vytvářet kanály (pro akce, schůze apod.).

Google bot

Konfigurační soubor pro správné nastavení Google bota je až na položky specifické pro služby Google stejný jako u Discord bota. Z toho důvodu budou níže popsány jen rozdílné položky.

- **Konfigurace pro propojení se službami Google**
  - **Přístupové údaje servisního účtu**
    - `Google:ServiceAccountConfig:Type` – Typ účtu.
      Zadejte řetězec `"service_account"`. Jiné hodnoty nejsou akceptovány.
    - `Google:ServiceAccountConfig:ProjectId` – Unikátní identifikátor aplikace.
    - `Google:ServiceAccountConfig:PrivateKeyId` – Identifikátor privátního klíče.
    - `Google:ServiceAccountConfig:PrivateKey` – Obsah certifikátu privátního klíče.
    - `Google:ServiceAccountConfig:ClientEmail` – Adresa elektronické pošty, na kterou je registrován servisní účet.
    - `Google:ServiceAccountConfig:ClientId` – Interní identifikátor servisního účtu.
    - `Google:ServiceAccountConfig:ClientX509CertUrl` – Adresa k veřejné části certifikátu.
  - `Google:User` – Adresa elektronické pošty, která je vázaná na servisní účet.
  - `Google:DefaultCalendarId` – Identifikátor Kalendáře Google, ve kterém má bot udržovat události. Identifikátor kalendáře lze získat v nastavení příslušného kalendáře.
  - `Google:Domain` – Organizační doména, ve které bot udržuje členy.

Veškeré konfigurace v konfigurační sekci „Přístupové údaje servisního účtu“ lze nalézt v konfiguračním souboru ve formátu JSON, který byl stažen při zakládání servisního účtu v konzoli Google. Postup vytvoření servisního účtu je popsán v kapitole 6.1.2.

[17] Identifikátory je možné získat zapnutím vývojářského režimu v klientovi služby Discord a následně využitím možnosti kopírování identifikátoru tlačítkem v kontextové nabídce.

6.2.2 Konfigurace databázového serveru v prostředí Docker

Pro spuštění databázového serveru v prostředí docker je třeba stáhnout obraz aplikace z registru DockerHub[18]. Toho lze dosáhnout příkazem `docker pull postgres`.

Následně je zapotřebí databázový server nakonfigurovat. K tomu slouží příkaz:

```
docker create --name Postgres -e 'POSTGRES_PASSWORD'='MAIN_PASSWORD' -p
    127.0.0.1:5432:5432 -v /path/to/data:/var/lib/postgresql/data postgres
```

- `--name Postgres` – Unikátní název kontejneru.
- `-e 'POSTGRES_PASSWORD'='MAIN_PASSWORD'` – Hlavní heslo pro přístup k databázovému serveru pod uživatelem „postgres“. Po přihlášení je možné založit nového uživatele.
- `-p 127.0.0.1:5432:5432` – Zpřístupnění výchozího portu pro databázi Postgres do hostitelského systému.
- `-v /path/to/data:/var/lib/postgresql/data` - Připojení adresáře s daty z hostitelského systému do kontejneru, aby při znovu vytvoření kontejneru nedošlo ke ztrátě databáze.

[18] DockerHub – PostgreSQL – `https://hub.docker.com/_/postgres`

Kapitola 7

Závěr

Cílem této práce bylo vytvořit centrální informační systém, který by usnadnil každodenní práci ve studentském spolku.

Na počátku celého vývoje proběhl sběr požadavků od členů Studentské Unie FIT formou osobních schůzí a elektronického dotazníku. Tento sběr požadavků umožnil obeznámení se stávající situací a požadavky na systém. Na základě těchto požadavků byl vytvořen návrh systému. Tento návrh se skládal z vytvoření návrhu databáze formou ER diagramu, funkcionality systému formou diagramu případů užití a analýzy dostupných technologií, na základě níž bylo rozhodnuto, že k implementaci bude využit programovací jazyk C#, který poběží na platformě .NET 5, a aplikační rámce ASP.NET Core a Angular. V této práci byla také provedena integrace externích systémů Google a Discord, které SU FIT využívá.

V samotné implementaci byly implementovány 4 aplikace. Jednalo se o hlavní jádro celého systému, které přistupuje k databázi a poskytuje rozhraní REST API a WebSocket, o webového klienta poskytujícího uživatelské rozhraní pro práci se systémem a nakonec o specializované aplikace (tzv. boty) integrující externí systémy Discord a Google Workspace.

V závěru této práce bylo také provedeno uživatelské testování, do nějž se připojilo 15 budoucích uživatelů. Na základě výsledků z provedeného testování bylo možné odhalit a vyřešit nejzávaznější chyby v systému, které vznikly při vývoji. Dále díky testování byly zjištěny nové návrhy a požadavky, které se dají implementovat v rámci dalšího vývoje.

Jak bylo uvedeno v předchozím odstavci, je očekáván další vývoj celého systému jako projekt s otevřeným zdrojovým kódem, do kterého budou moct přispět i další vývojáři. Mezi další vylepšení systému je plánováno například rozšíření lokalizace i do webového klienta, přidání dalších jazyků (například slovenského jazyka), nebo implementace nových sekcí podle budoucích požadavků.

Literatura

[1] Ajcvickers a olprod. *Database providers – EF Core* [online]. Dec 2019 [cit. 2020-09-04]. Dostupné z: https://docs.microsoft.com/cs-cz/ef/core/providers.

[2] Anil, N. a oldprod. *Co je Docker?* [online]. Aug 2020 [cit. 2021-03-01]. Dostupné z: https://docs.microsoft.com/cs-cz/dotnet/architecture/containerized-lifecycle/what-is-docker.

[3] *API Docs for Bots and Developers / Create message* [online]. Discord Inc. [cit. 2021-04-15]. Dostupné z: https://discord.com/developers/docs/resources/channel#create-message.

[4] auth0.com. *JSON Web Tokens Introduction* [online]. [cit. 2021-04-15]. Dostupné z: https://jwt.io/introduction.

[5] D, M. a G, K. *Angular vs. react: Which is better and why?* [online]. Mar 2020 [cit. 2020-10-01]. Dostupné z: https://www.cleveroad.com/blog/angular-vs-react.

[6] *DB Engines Ranking* [online]. DB-Engines.com, 2021 [cit. 2020-09-04]. Dostupné z: https://db-engines.com/en/ranking.

[7] De George, A. a oldprod. *Prostředky v aplikacích .NET* [online]. Jul 2018 [cit. 2020-10-03]. Dostupné z: https://docs.microsoft.com/cs-cz/dotnet/framework/resources/.

[8] Douglas, J. a oldprod. *Co je NuGet a co dělá?* [online]. May 2019 [cit. 2021-04-04]. Dostupné z: https://docs.microsoft.com/cs-cz/nuget/what-is-nuget.

[9] Juneau, J. Object Relational Mapping and JPA. In:. Leden 2013, s. 55–72. DOI: 10.1007/978-1-4302-5849-0_4. ISBN 978-1-4302-5848-3.

[10] Magnan, J. et al. *Entity Framework EF Code First* [online]. 2012 [cit. 2020-09-25]. Dostupné z: https://entityframework.net/ef-code-first.

[11] Mark Otto, J. T. *Introduction* [online]. May 2020 [cit. 2021-01-04]. Dostupné z: https://getbootstrap.com/docs/5.0/getting-started/introduction/.

[12] Mark Otto, J. T. *License FAQs* [online]. May 2020 [cit. 2021-01-04]. Dostupné z: https://getbootstrap.com/docs/5.0/about/license/.

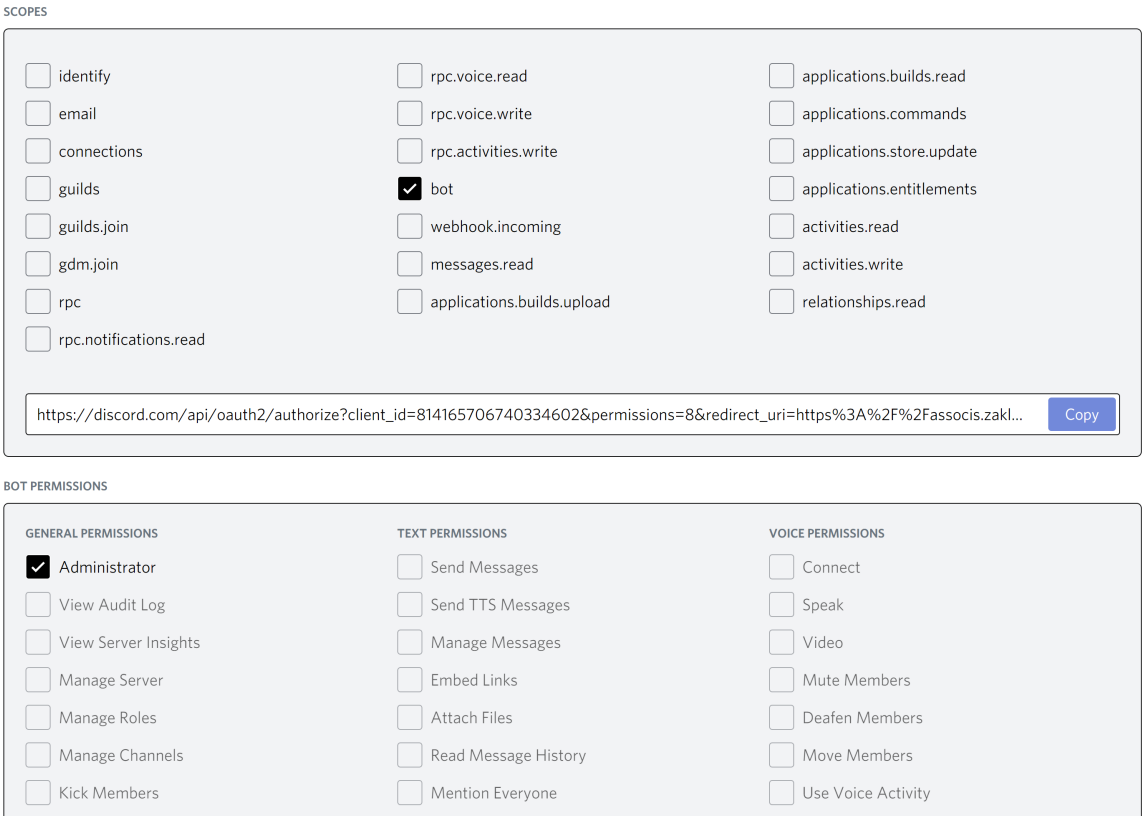
[13] Nish, A., Youssef, V., Cédric, M., Parente, J. a Wenzel, M. *Designing the infrastructure persistence layer* [online]. Aug 2018 [cit. 2020-09-25]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-design.

[14] Ray, M., Wells, J., Agarwal, S., Steen, H., Coulter, D. et al. *Editions and supported features - SQL Server 2017* [online]. Dec 2019 [cit. 2020-09-04]. Dostupné z: https://docs.microsoft.com/en-us/sql/sql-server/editions-and-components-of-sql-server-2017?view=sql-server-ver15.

[15] Wagner, J. *Understanding and the API-First Approach to Building Products* [online]. 2018 [cit. 2020-09-03]. Dostupné z: https://swagger.io/resources/articles/adopting-an-api-first-approach/.

Příloha A

Přehled oprávnění při přidání Discord bota na Discord server

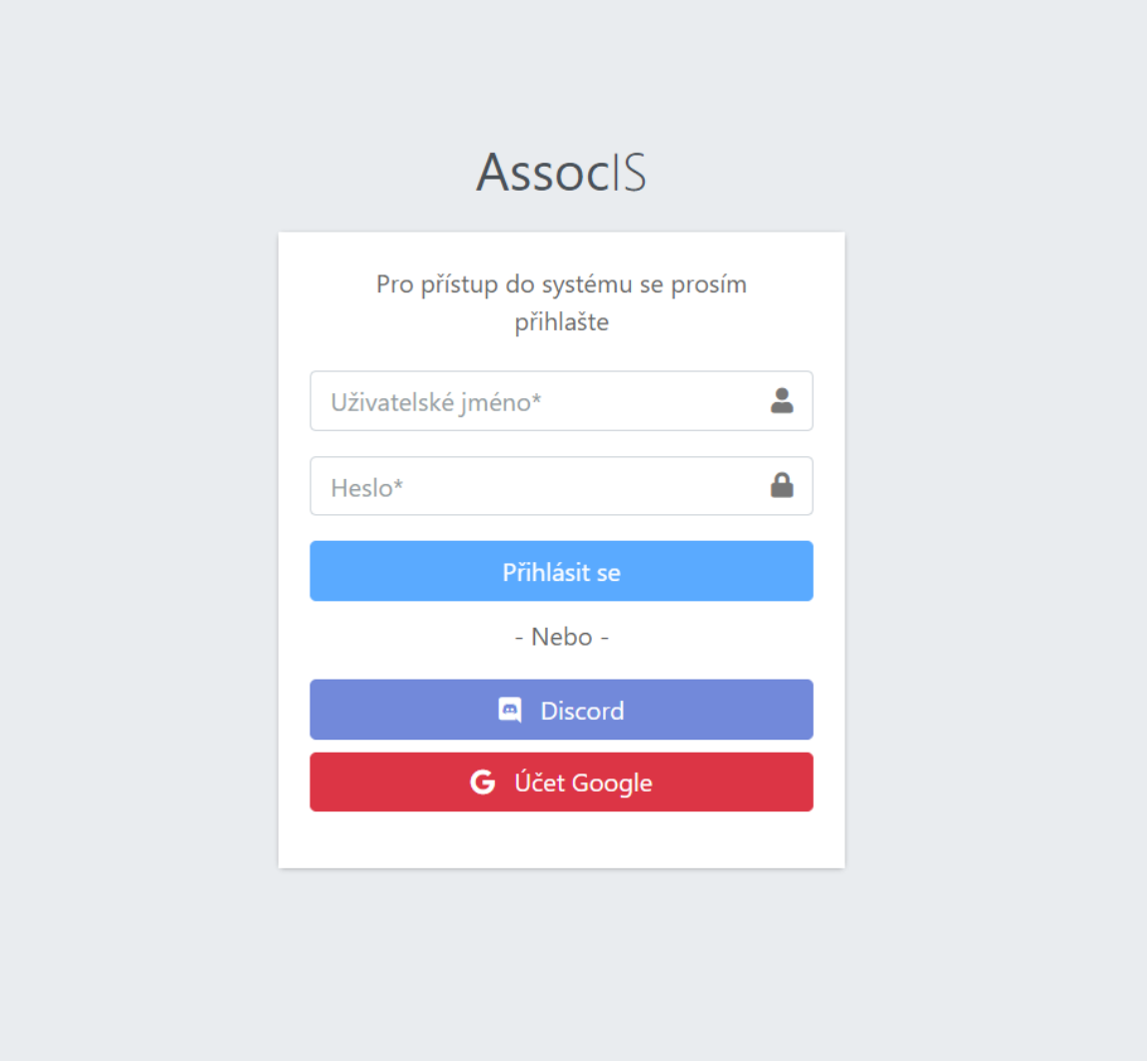


Obrázek A.1: Výběr oprávnění ve vývojářském portálu

Příloha B

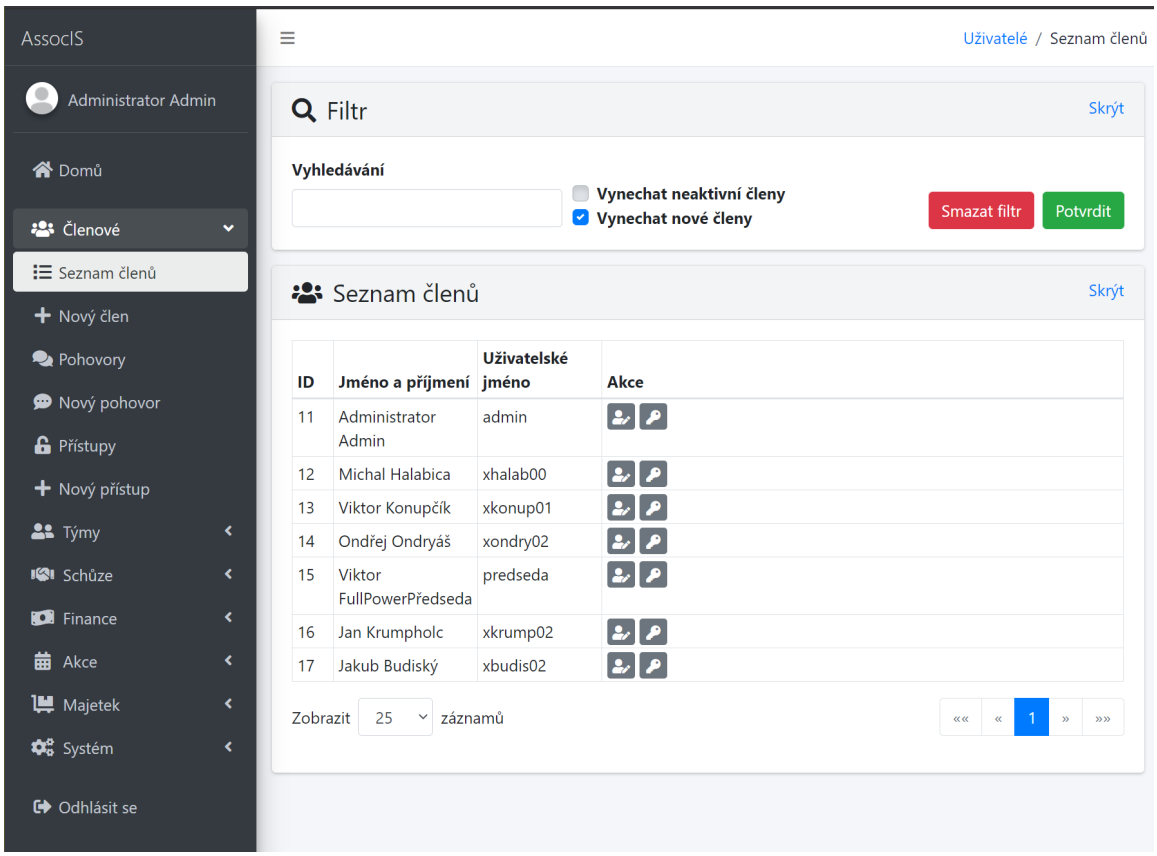
Vzhled uživatelského rozhraní

B.1 Přihlášení



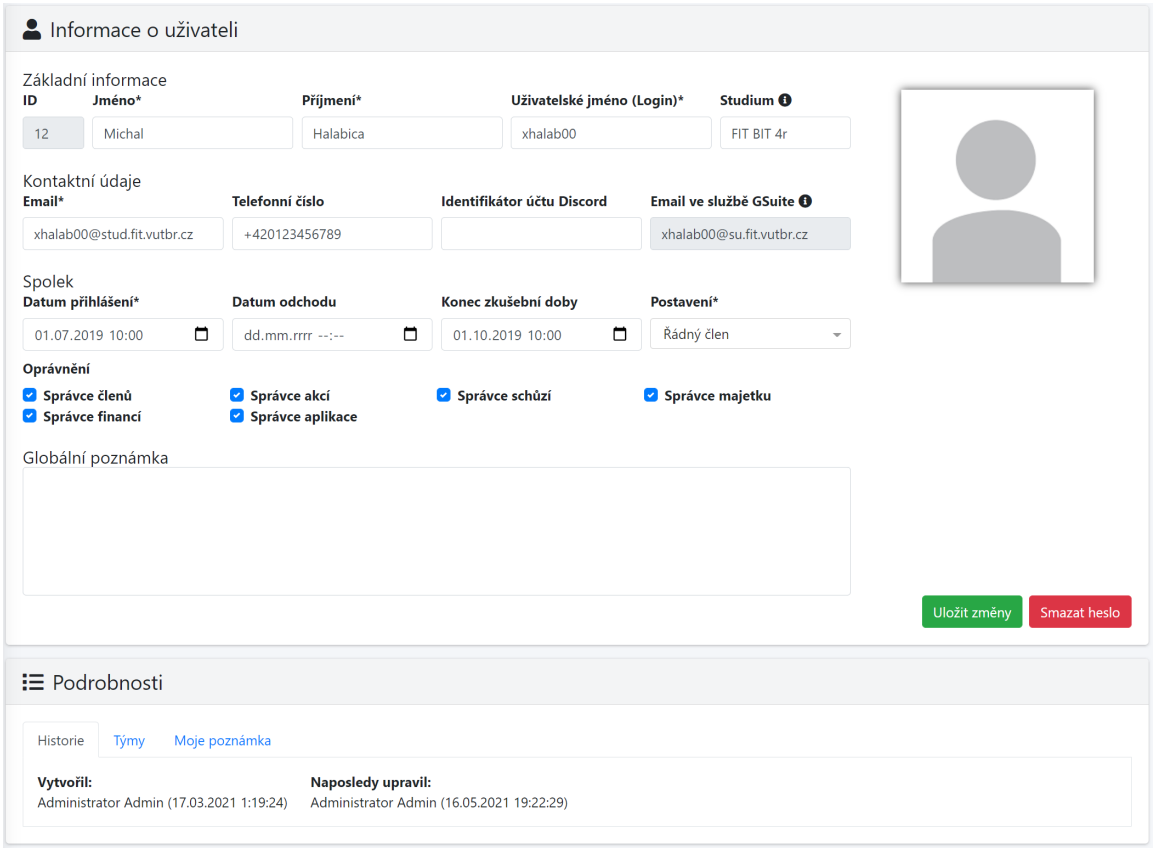
Obrázek B.1: Scéna pro nepřihlášeného uživatele s možnostmi přihlášení

B.2 Správa členů – Tabulkový pohled



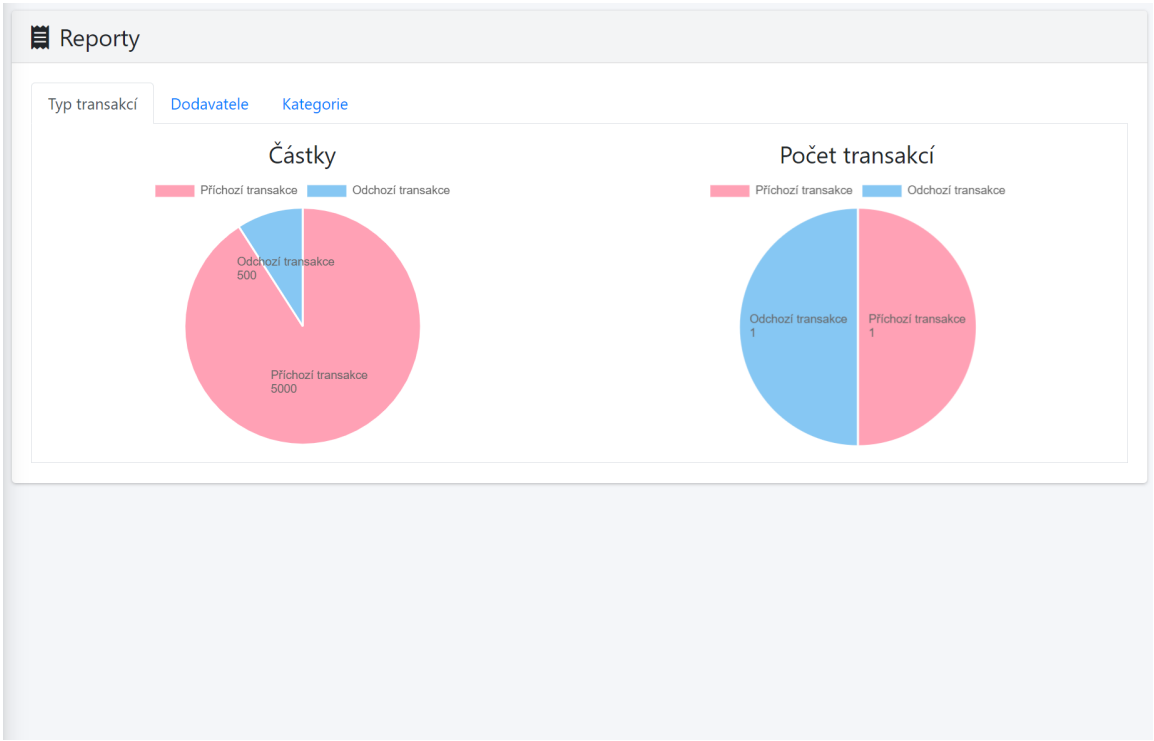
Obrázek B.2: Scéna pro přihlášeného uživatele se seznamem členů

B.3 Správa členů – Formuláře



Obrázek B.3: Podsekce editace člena spolku

B.4 Finanční reporty – Grafy



Obrázek B.4: Podsekce reprezentující finanční reporty pomocí grafů