FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Mobilní aplikace pro matchmaking hráčů badmintonu
Student:	Patrik Benk
Vedoucí:	Ing. Marek Suchánek
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2023/2024

Pokyny pro vypracování

V prostředí online her je pojem matchmaking užíván pro mechanismus vybírající soupeřící hráče či týmy na základě jejich různých metrik a ratingu. Podobné mechanismy mohou být využity i v tradičních sportech, ve kterých se proti sobě utkávají dvě či více stran. Cílem této práce je vyvinout mobilní aplikaci pro zařízení Android a iOS, která bude umožňovat matchmaking hráčů badmintonu.

- Popište problematiku matchmakingu a ratingu hráčů. Rovněž stručně popište varianty a průběh hry badmintonu i parametry/metriky hráčů, které mohou být použity pro rating.
- Proveďte rešerši existujících řešení pro matchmaking v badmintonu a případně dalších sportech.
- Sestavte katalog požadavků a navrhněte vlastní řešení formou mobilní aplikace, které umožní registrovaným hráčům badmintonu sledovat žebříček a hledat vhodné protihráče. Aplikaci navrhněte tak, aby bylo snadné ji dále rozvíjet a případně použít pro jiné sporty.
- Implementujte aplikaci dle návrhu. Výběr technologií zdůvodněte a výslednou implementaci zdokumentujte a otestujte.
- Zhodnoťte přínosy aplikace a shrňte možnosti dalšího rozvoje.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 17. listopadu 2022 v Praze.

Bakalářská práce

MOBILNÍ APLIKACE PRO MATCHMAKING HRÁČŮ BADMINTONU

Patrik Benk

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Marek Suchánek
29. června 2023

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Rád bych poděkoval především vedoucímu bakalářské práce, Ing. Markovi Suchánkovi, za jeho trpělivost, podporu a cenné rady poskytnuté nejen k bakalářské práci, ale i samotnému vývoji softwaru. Poděkování také patří mé rodině, která mi po celou dobu studia poskytovala potřebné prostředky a podporu.

Prohlášení

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

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 29. června 2023 .....

Abstrakt

Tato práce se zabývá vývojem mobilní aplikace pro matchmaking hráčů badmintonu. Vývoj je řízen klasickým vodopádovým modelem, přičemž i struktura práce odpovídá jeho rozdělení na jednotlivé vývojové fáze. Nejdříve je jasně určen cíl, kterého by se mělo dosáhnout a navrhnuto řešení jeho realizace, vyplívající z rozboru již existujících částečných řešení. Následně je důkladně zanalyzována daná problémová doména a tím i zpřesněn způsob realizace cíle. V návrhu jsou rozebrány a následně určeny jednotlivé algoritmy, technologie a architektury systému. Posléze přichází na řadu samotná implementace dvou serverových částí v běhovém prostředí Node.js a mobilního klienta, jak pro iOS, tak i Android, za pomocí knihovny React Native. Implementace je poté otestována. Na konci práce jsou navhrnuta další možná vylepšení, následováno konečným zhodnocením dosažení vytyčených cílů.

Klíčová slova vývoj mobilní aplikace, hodnocení hráčů, matchmaking, Node.js, React Native, PostgreSQL, REST, WebSocket

Abstract

This thesis deals with the development of a complete mobile application for matchmaking badminton players. The development is guided by the classical waterfall model, and the structure of the work partly follows its division into different development phases. First, the goal to be achieved is clearly identified and a solution for its realization is proposed, resulting from the analysis of already existing partial solutions. Subsequently, the problem domain is thoroughly analysed and the way of realising the goal is refined. In the proposal, the individual algorithms, technologies and system architectures are analyzed and subsequently determined. Then comes the actual implementation of the 2 server parts in the Node.js runtime environment and the mobile client, both for iOS and Android, using the React Native library. The implementation is then tested. At the end of the work, further possible improvements are outlined, followed by a final evaluation of the achievement of the set goals.

Keywords Mobile application development, Player rating, Matchmaking, Node.js, React Native, PostgreSQL, REST, WebSocket

Seznam zkratek

API	Application Programming Interface
FIDE	Fédération Internationale Des Échecs
iOS	iPhone Operating System
JSON	JavaScript Object Notation
JWT	JSON Web Token
REST	Representational State Transfer
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
UI	User Interface

Kapitola 1
Úvod

Sportovní aktivity jsou důležitým aspektem lidského života, jelikož nám nejen pomáhají udržovat si fyzickou zdatnost a disciplínu, ale hlavně nás spojují. Vytvořením zdravého soutěživého prostředí můžeme docílit ještě větší motivace ke zlepšování se a v rámci skupinových sportů i utužování vztahů. V každém sportu, kde existuje nějaká měřitelná jednotka, vypovídající o výkonu, lze vytvořit soutěživé prostředí, jejím srovnáváním. Touto jednotkou může být například čas, vzdálenost, váha nebo, jak to bývá u nejvíce sportů, body odvozené z vlastních pravidel. Možnost měření našeho výkonu vytváří ideální prostředí pro zlepšování se, jelikož nám to dává lepší představu o našich schopnostech a díky možnosti srovnání si můžeme také vybírat protihráče podobné úrovně. Můžeme tak lépe optimalizovat náš plán tak, abychom vždy vykonávali daný sport na hraně našich sil a nejen se zlepšovali co nejrychleji, ale aby i zápasy byly co nejzábavnější.

Sporty, ve kterých se hráči hodnotí a srovnávají, jdou dále rozdělit na sporty s absolutním a relativním hodnocením. Sporty s absolutním hodnocením jsou takové, kde lze dovednost účastníka měřit objektivně jeho výkonem, který není závislý na výkonech ostatních. Příkladem sportu s absolutním hodnocením může být gymnastika, plavání, vzpírání a další. Jedná se však o zlomek všech sportů, jelikož v naprosté většině sportů stojí hráči proti sobě a jejich výkony jsou tedy na sobě závislé. Celkové hodnocení a následné srovnávání účastníků ve sportech s absolutním hodnocením bývá velmi jednoduché, jelikož často stačí pouze sečíst dílčí hodnocení ze všech událostí, či pouze srovnávat nejvyšší dosažené skóre. To ale nejde říct o sportech s relativním hodnocením, na které je potřeba použít některý z komplexnějších hodnotících a srovnávacích systémů, který při určování celkového hodnocení účastníka z výsledku zápasu bere v úvahu i nerovné podmínky, způsobené odlišnými výkony hráčů.

Jedním z těchto sportů je i badminton, na který se budu soustředit v této práci, jelikož je velice rozšířený a mám s ním i osobní zkušenosti. Badminton je nejrychlejším raketovým sportem a jeho pravidla jsou relativně jednoduchá. Hra se hraje buď ve formě dvojhry nebo čtyřhry, vždy na dva vítězné sety. Set dostane ta strana, která jako první dosáhne 21 bodů. V případě, že nastane nerozhodný stav 20:20, strana, co jako první dosáhne dvoubodového náskoku, set vyhrává. Body se získávají po jednom z každé vyhrané výměny. Strana, která míčkem zasáhne protivníkovo území, aniž by soupeř dokázal míček odrazit, vyhrává výměnu. [1]

Jako sport s relativním hodnocením potřebuje k objektivnímu určení úrovně hráče co nejvíce dat. V ideálním prostředí by tato data zahrnovala nejenom výsledek samotného zápasu, ale i podrobné statistiky hráče jako např. obratnost, rychlost úderu, přesnost či herní styl. Na skutečnou sílu hráče mají tyto vlastnosti totiž velký vliv a jejich zanedbáním ztrácíme potřebnou přesnost při odhadu úrovně. V reálném světě je však velmi obtížné tyto metriky přesně měřit a proto se v rámci badmintonu musíme spokojit pouze s výsledným skórem zápasu.

Samotný matchmaking, nebo–li párování hráčů podle úrovní, v tomto prostředí existuje pouze ve formě dopředného plánování turnajů. Párování hráčů v reálném čase, na které jsme zvyklí

z kompetitivních počítačových her, zde vůbec neexistuje. Oproti prostředí počítačových her má sice nevýhodu v menším počtu uživatelů, ale na druhou stranu jeho implementace bude o dost snazší, jelikož se při párování nemusí zohledňovat odezva připojení jednotlivých hráčů. [2]

Složitější hodnotící systémy jsou taktéž kvůli své náročnosti na správu, využívány pouze na profesionálních soutěžích, kde jsou pro tento účel přímo vyhrazeny potřebné prostředky. Problémem je však použití takového systému v klubu, který si takové náklady nemůže dovolit a musí tak spoléhat pouze na vlastní nedokonalé řešení. V klubu, který navštěvuji, není tento problém řešen vůbec a párování probíhá víceméně náhodně nebo podle vlastního odhadu schopností hráčů, právě kvůli své složitosti a náročnosti na obsluhu. Sice existují aplikace, které dokáží určovat úrovně hráčů podle výsledků zápasů, ale všechny musejí být obsluhovány pouze jedním člověkem, na kterého jsou pak kladeny moc velké nároky. Dosavadní aplikace taktéž neřeší párování hráčů na základě jejich úrovní a tak přidávají další práci vedoucím klubu.

Kapitola 2
Cíl práce

Cílem práce je vyvinout mobilní matchmakingovou aplikaci, která bude nejenom vypočítávat úrovně hráčů na základě výsledků zápasů, ale také je následně na konkrétních událostech párovat mezi sebou. Aby se eliminovaly vysoké nároky na jedince spravující daný klub, jsou tyto nároky rozprostřeny mezi všechny uživatele aplikace, tak že hráči si skóre po zápase budou zadávat do aplikace sami a následně se budou i sami párovat.

Postupovat se bude podle klasického vodopádového modelu, podle kterého budou taktéž rozděleny kapitoly na analýzu, návrh, implementaci a testování. Nejdříve budou určeny jasné požadavky na řešení, plynoucí z rešerše konkurenčních aplikací, a důkladně zanalyzovaná problémová doména matchmakingu hráčů badmintonu pomocí mobilní aplikace. Poté budou na základě předchozí analýzy určeny vhodné technologie a architektura systému. Ve zvolených technologiích bude následně implementováno samotné řešení, které bude taktéž otestováno. Nakonec se navrhnou případná vylepšení a výsledky práce se zhodnotí.

Kapitola 3
Analýza

Analýza je důležitým procesem softwarového vývoje, který se zaměřuje na upřesnění řešeného problému a stanovení jasného cíle. Nejprve bude provedena rešerše již existujících řešení a na jejím základě určeny požadavky na novou aplikaci. Poté bude na diagramech zachycena doména samotná, čímž získáme další informace potřebné k úplnému pochopení problematiky.

3.1 Konkurenční aplikace

Ze všech existujících řešení se o žádném nedá přímo říct, že by byla konkurenční, jelikož žádná z nich nenabízí všechny potřebné funkcionality. Za konkurenční budou tedy považovány i ty, které problém řeší částečně. Párování hráčů je možné nejen pomocí matchmakingu, ale i turnaje, proto budou uvažovány i aplikace sloužící k vytváření turnajů.

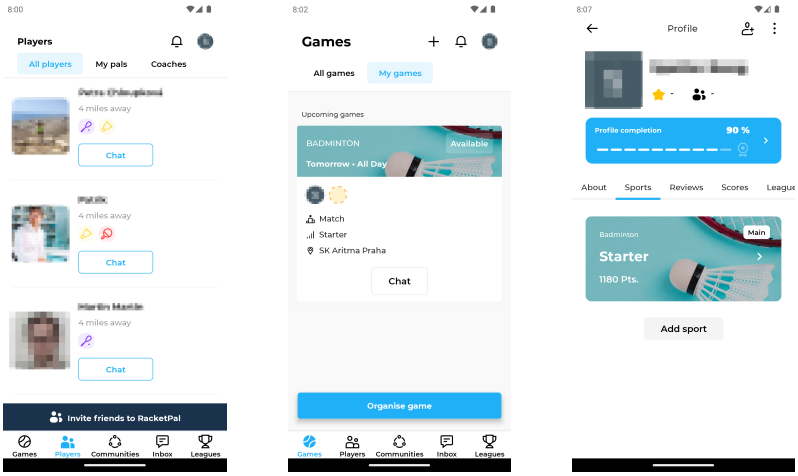
3.1.1 RacketPal

Jde o jednu z nejpoužívanějších mobilních aplikací pro hledání vhodných spoluhráčů nejen v badmintonu, ale i dalších raketových sportech. Z pohledu uživatele je samotný matchmaking neprůhledný a ne moc zřejmý, jelikož vypadá spíše jako seznamovací platforma, ve které se vybírá ze seznamu doporučených lidí seřazených především podle vzdálenosti. Domnívám se však, že doporučování bere v úvahu taktéž úroveň hráče, jelikož je v aplikaci zaznamenávána. Každý hráč může vytvořit zápas, spoluhráče pak může buď do zápasu vlastnoručně přidat nebo získat na základě doporučení. Po ukončení zápasu je potřeba výsledné skóre zadat do aplikace, tak aby mohlo být Elo hráčů přepočítáno. Aplikace taktéž umožňuje zobrazení žebříčku nejlepších hráčů a jejich historii zápasů.

Její největší nevýhodou je však, že matchmaking je pojat jako dopředné párování hráčů před událostí a pro párování hráčů přímo na místě akce je tedy nepoužitelná. Turnaje také chybí.

Shrnutí:

+ Výpočet úrovně hráče na základě výsledku zápasu.
+ Matchmaking (nejspíše) zohledňuje úroveň hráče.
+ Žebříček nejlepších hráčů a historie zápasů.
– Nejde použít pro lokální párování hráčů.
– Nelze zde vygenerovat turnaj zohledňující úroveň hráčů.



(a) Hledání protivníků **(b)** Vytvořené zápasy **(c)** Statistiky hráče

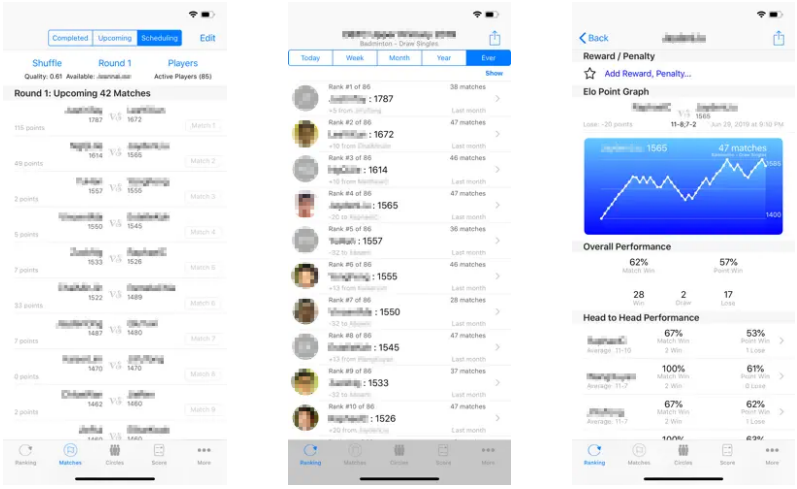
Obrázek 3.1 Snímky obrazovky aplikace RacketPal [3]

3.1.2 Elo Challenge

Velmi jednoduchá aplikace sloužící pouze pro zaznamenávání Elo ratingu a jeho přepočtu. Obsahuje žebříček nejlepších hráčů i historii zápasů. Naopak matchmaking i generování turnajů chybí úplně.

Shrnutí:

+ Výpočet úrovně hráče na základě výsledku zápasu.
+ Žebříček nejlepších hráčů a historie zápasů.
– Nemá matchmaking ani turnaj zohledňující úroveň hráčů.



(a) Seznam nadcházejících zápasů **(b)** Žebříček nejlepších hráčů **(c)** Statistiky hráče

Obrázek 3.2 Snímky obrazovky aplikace Elo Challenge [4]

3.1.3 Winner – Tournament Maker App

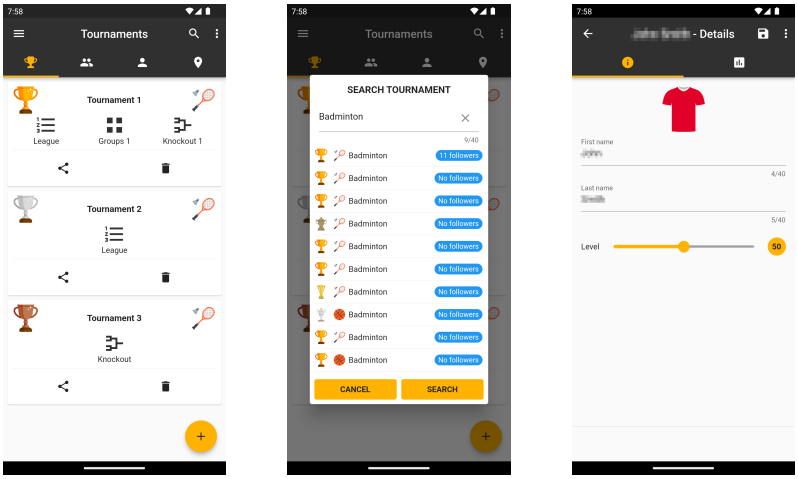
Aplikace určená pouze pro vytváření turnajů. Neobsahuje matchmaking jako takový, ani přepočet úrovně na základě výsledku zápasu. Úroveň se každému hráči vlastnoručně zadá v aplikaci a dále se již nemění. Dle úrovně pak dokáže vytvořit turnaj, který je optimalizován tak, aby součet rozdílů úrovní v jednotlivých zápasech byl co nejmenší.

Shrnutí:

\+ Turnaj zohledňující úroveň hráčů.

– Chybí klasický matchmaking.

– Úroveň je pevně zadávána a nepřepočítává se.



(a) Vyhledávání turnajů ke sledování

(b) Seznam vytvořených turnajů

(c) Statistiky hráče

Obrázek 3.3 Snímky obrazovky aplikace Winner – Tournament Maker App [5]

3.1.4 Shrnutí

I přes to, že každá z výše zmíněných aplikací řeší pouze část problému, složením vybraných funkcionalit z každé aplikace je možné dosáhnout plnohodnotného řešení. Na základě nejen toho, ale i vlastních nápadů jsem tedy vytvořil následující požadavky.

3.2 Funkční požadavky

Požadavky určující funkčnost aplikace. Říkají nám co má aplikace umožňovat a jakým způsobem. Pro znázornění priorit požadavků je použita metoda MoSCoW [6], kde požadavky nutné ke splnění projektu budou označeny písmenem M, požadavky důležité, ale ne nutné písmenem S, požadavky volitelné písmenem C a požadavky, u kterých je dopředu známo, že se v rámci projektu plnit nebudou, písmenem W.

3.2.1 Registrace, přihlášení a správa dat uživatele (M)

Systém by měl umožnit uživateli založení účtu a jeho následné spravování. K založení účtu bude potřeba přezdívka, email a heslo. Systém si ke každému účtu taktéž bude pamatovat kluby, které vede a události, ke kterým je uživatel přihlášen. Dále bude spravovat jeho úrovně v různých typech her a historii zúčastněných zápasů.

3.2.2 Registrace přes služby třetích stran (C)

K autentizaci bude taktéž možné použít účty jiných služeb.

3.2.3 Používání aplikace bez registrace (W)

Uživatel bude moci aplikaci používat i bez registrace, vytvořením tzv. dočasného účtu. V tomto případě mu však nebude ukládána, jak úroveň, tak ani zápasy. Navíc všechny zápasy, kterých se bude nepřihlášený uživatel účastnit budou označeny jako nehodnocené.

3.2.4 Vytvoření a správa události správcem (S)

Uživatelům může být přidělena role správce, která jim zpřístupní vytváření a spravování událostí v daném klubu.

3.2.5 Přihlášení uživatele na událost (M)

Uživatelé se budou moci přihlašovat na vytvořené události. Ke vstupu na událost je potřeba aby daný uživatel byl schválen některým ze správců.

3.2.6 Přidávání dalších hráčů do skupiny (S)

Vedoucí bude moci do své skupiny pozvat další hráče, kteří se v případě přijmutí stanou členy dané skupiny. Připojit se do matchmakingu či rozpustit skupinu může také pouze vedoucí.

3.2.7 Přidávání offline hráčů do skupiny (C)

Hráčům, kteří se chtějí zúčastnit matchmakingu, ale zároveň nejsou uživateli, bude umožněno se připojit skrze jiného uživatele jako tzv. offline hráč. Pro přidání offline hráče musí vedoucí skupiny zadat jeho přezdívku a odhadovanou úroveň. Jinak pro ně platí to stejné, co pro nepřihlášené uživatele.

3.2.8 Vytvoření a správa turnaje správcem (W)

V rámci události bude možné vytvořit turnaj o libovolném počtu hráčů. Správa turnaje zahrnuje jak jeho spuštění po připojovací fázi a tím i vytvoření plánu turnaje, tak i jeho ukončení. Plán turnaje by měl stejně jako matchmaking zohledňovat úrovně hráčů tak, aby rozdíly mezi hráči byly co nejmenší.

3.2.9 Přihlášení hráče na turnaj (W)

Každý hráč se může v jeden moment účastnit maximálně jednoho turnaje. Stejně jako u matchmakingu se do turnaje mohou připojovat i skupiny hráčů.

3.2.10 Přiřazování hráčů pomocí matchmakingu (M)

Matchmaking bude spojovat hráče s nejpodobnější úrovní. Do matchmakingu se budou moci připojovat i skupiny hráčů.

3.2.11 Zadání skóre zápasu (M)

Po konci zápasu bude vedoucím jednotlivých skupin umožněno zadání skóre.

3.2.12 Kontrola shodného skóre (S)

V případě neshody skóre bude zvoleno takové, za kterým bude stát nejvíce účastníků. Pokud bude na výběr více možností, bude z nich vybráno náhodně.

3.2.13 Automatický přepočet úrovně hráče (M)

Systém bude automaticky přepočítávat úroveň všech hráčů na základě výsledků hodnocených zápasů.

3.2.14 Zobrazení statistik hráče (S)

Uživatelé si budou moci zobrazit své statistiky jako například úroveň nebo historii zápasů.

3.2.15 Zobrazení žebříčku nejlepších hráčů (M)

Na hlavní stránce aplikace bude dostupný žebříček nejlepších hráčů.

3.3 Nefunkční požadavky

Požadavky nesouvisející s funkčností aplikace, ale s jejím naplněním.

3.3.1 Rozšiřitelnost pro další sporty

Při vývoji bude kladen důraz na to, aby byla aplikace lehce rozšiřitelná pro další sporty.

3.3.2 Podpora více jazyků

Aplikace bude vyvíjena tak, aby následné přidání překladu pro nový jazyk bylo co nejsnazší.

3.3.3 Ochrana osobních údajů

Bude dbáno na dodržování základních bezpečnostních principů nutných k ochraně dat uživatelů jako např. ukládání zahashovaného hesla nebo omezení přístupu k datům neautorizovaným osobám.

3.4 Model případů užití

Model případů užití slouží k přesnější specifikaci funkčních požadavků. Je složen ze dvou částí: diagram případů užití a jeho textový popis. Diagram slouží pouze k přiřazení aktérů[1] k případům užití, ve kterých vystupují. Textová část zase obsahuje detailní specifikaci procesu, který daný aktér v případu užití vykonává. V rámci jednoho případu užití může existovat i více scénářů, přičemž jeden je definovaný jako hlavní. Každý scénář může mít navíc určené pre–conditions, což je seznam podmínek, které musí platit před započetím konkrétního scénáře.

3.4.1 Autentizace

Každý uživatel se musí autentizovat a to buď pomocí trvalého nebo dočasného účtu. Dočasný účet je pro ty, kteří chtějí aplikaci pouze vyzkoušet a nechtějí se registrovat. Nevýhodou pak ale je neukládání úrovně a jiných statistik uživatele. Alternativou pro autentizaci je taktéž použití služeb třetích stran. Pokud se uživatel sám neodhlásí, bude si aplikace autentizaci pamatovat a nebude tedy nutné se znovu přihlašovat.

Aktér: Uživatel

Hlavní scénář: Autentizace pomocí trvalého účtu

1. Uživatel se rozhodne autentizovat pomocí trvalého účtu.
2. Systém uživateli zobrazí přihlašovací stránku. Pokud uživatel ještě není registrován pokračuje scénář dále. V opačném případě se rovnou přejde na 7. krok.
3. Uživatel přejde na registrační stránku.
4. Systém uživateli zobrazí stránku pro registraci, kde po něm bude požadována přezdívka, email a heslo.
5. Uživatel zadá všechny tyto informace a odešle je.
6. Systém uživatele znovu přesune na přihlašovací stránku.
7. Uživatel zadá své přihlašovací údaje, tedy email a heslo.
8. Systém uživatele vpustí do aplikace s odpovídajícími právy a daty.

Vedlejší scénář: Autentizace pomocí služeb třetích stran

1. Uživatel se rozhodne autentizovat pomocí služeb třetích stran.
2. Systém uživateli zobrazí přihlašovací stránku. Pokud uživatel ještě není registrován pokračuje scénář dále. V opačném případě se rovnou přejde na 7. krok.
3. Uživatel přejde na registrační stránku.
4. Systém uživateli zobrazí stránku pro registraci, kde bude možné zvolit registraci pomocí služeb třetích stran.
5. Uživatel si jednu z nich vybere a povolí propojení s danou službou.
6. Systém uživatele znovu přesune na přihlašovací stránku, kde bude možné zvolit přihlášení pomocí služeb třetích stran.
7. Uživatel si jednu z nich vybere a povolí propojení s danou službou.
8. Systém uživatele vpustí do aplikace s odpovídajícími právy a daty.

Vedlejší scénář: Autentizace pomocí dočasného účtu

[1] Role účastnící se daného procesu

1. Uživatel se rozhodne autentizovat pomocí dočasného účtu.
2. Systém uživateli zobrazí přihlašovací stránku pro dočasné účty, kde bude vyžadována pouze přezdívka.
3. Uživatel si zvolí a následně potvrdí přezdívku.
4. Systém uživatele vpustí do aplikace s pouze základními právy a bez ukládání dat.

3.4.2 Přihlášení se na událost

Správce samotný je automaticky přijat na události, které spravuje a nemůže z nich být vyhozen. Všichni ostatní musí být nejdříve přijati, aby mohli vstoupit.

Aktér: Uživatel

Pre–condition:

- Je vypsána alespoň jedna událost.

Hlavní scénář: Přihlášení se na událost

1. Uživatel se rozhodne přihlásit se na událost.
2. Systém uživateli zobrazí všechny dostupné kluby.
3. Uživatel si jednu z nich zvolí.
4. Systém uživateli zobrazí všechny události v něm vypsané. Události budou seřazeny vzestupně podle data a času konání.
5. Uživatel si zvolí jednu z nich.
6. Systém ho vpustí do události pouze v případě byl–li již přijat. Pokud zatím přijat nebyl a nečeká se na vyřízení jiné žádosti, nabídne mu poslání žádosti k připojení.
7. V případě, že se uživatel rozhodne žádost neodeslat, scénář končí. Jinak scénář pokračuje.
8. Systém pošle všem správcům klubu, ve které byla událost vytvořena, tuto žádost na rozhodnutí a přesune uživatele zpátky na stránku s událostmi.

3.4.3 Zúčastnění se zápasu

Zápasů se uživatelé účastní vždy jako skupina s alespoň jedním členem. Každý má implicitně vlastní skupinu, ve které je uživatel samotný v roli vedoucího. Vedoucí může do skupiny přidávat další členy[2] a používat přiřazovací systémy. Naopak členové skupiny nemohou dělat nic, protože vše řídí vedoucí. Tedy stačí uvažovat scénář pouze z pohledu vedoucího. Vedoucí může sestavit i tým protivníků, ale pak může zápasy hledat pouze pomocí matchmakingu. V případě, že jsou dokonce oba týmy plné, hledání se přeskakuje a dovolí vedoucímu zápas rovnou započít.

Aktér: Uživatel

Pre–condition:

- Uživatel je přihlášen na nějakou událost.
- Události se aktivně účastní i jiní uživatelé.
- Událost je aktivně spravována alespoň jedním správcem.
- V události existuje dosud nespuštěný turnaj.

[2] Včetně offline hráčů

Post–condition:

- Úroveň na hráčském profilu účastníka je přepočítána.
- Detaily odehraného zápasu jsou uloženy.

Hlavní scénář: Zúčastnění se zápasu pomocí matchmakingu

1. Uživatel se rozhodne zúčastnit se zápasu.
2. Systém uživateli zobrazí jeho skupinu a nabídne mu přidání dalších hráčů či odebrání stávajících.
3. Až se rozhodne vyhledat samotný zápas, klikne na tlačítko zápasu a zvolí si matchmaking a formát hry, pro které má odpovídající počet hráčů.
4. Systém přidá skupinu do přiřazovacího systému a pokusí se ho spárovat s ostatními skupinami tak, aby společně tvořili dva celé týmy a úrovně hráčů byly co nejpodobnější.
5. Až systém najde nějaké vhodné řešení, pošle vedoucímu skupiny návrh na potvrzení.
6. Uživatel potvrdí návrh. Pokud vedoucí, kterékoliv z ostatních skupin odmítne, vrátí se scénář na předešlý krok.
7. Systém vytvoří zápas pro skupinu uživatele a umožní mu zadání skóre.
8. Uživatel zadá skóre zápasu.
9. Pokud systém detekuje neshodu skóre mezi vedoucími, vyzve je znovu k zadání a vrátí se na předešlý krok.
10. V případě, že se jednalo o hodnocený zápas, systém přepočítá úrovně všech účastníků a uloží detaily zápasu.

Vedlejší scénář: Zúčastnění se zápasu pomocí turnaje

1. Tento scénář se s tím hlavním začne rozcházet až ve kroku 3, ve kterém uživatel místo matchmakingu zvolí turnaj.
2. Systém skupinu přidá do čekací místnosti turnaje.
3. Až některý ze správců turnaj spustí, systém vygeneruje hrací plán a podle něj bude postupně vytvářet zápasy. Dále scénář pokračuje krokem 7 z hlavního scénáře.

3.4.4 Zobrazení statistik hráče

Všem uživatelům se ukládá nejen aktuální úroveň, ale i historická úroveň a odehrané zápasy k jejich konkrétním hráčským profilům. Hráčské profily se liší nejen sportem, ale i formou hry. Tyto statistiky může zobrazit pouze uživatel pod který dané hráčské profily spadají.

Aktér: Uživatel

Pre–condition:

- Uživatel má odehraný alespoň jeden zápas.

Hlavní scénář: Zobrazení statistik hráče

1. Uživatel se rozhodne zobrazit jeho statistiky.
2. Systém uživateli zobrazí všechny jeho hráčské profily, ve kterých odehrál alespoň jeden zápas.
3. Uživatel si zvolí jeden z nich.
4. Systém mu zobrazí aktuální úroveň a počet odehraných zápasů daného hráčského profilu. Dále mu umožní si zobrazit i graf historie jeho úrovně, či konkrétní odehrané zápasy s informacemi o času, účastnících a skóre.

3.4.5 Zobrazení žebříčku nejlepších hráčů

Jelikož každý hráčský profil má svoji úroveň, je možné je mezi sebou srovnávat. Uživatel si tedy může zobrazit žebříček a zjistit jak je na tom v porovnání s ostatními.

Aktér: Uživatel

Pre–condition:

- Uživatel má odehraný alespoň jeden zápas.

Hlavní scénář: Zobrazení žebříčku nejlepších hráčů

1. Uživatel se rozhodne zobrazit žebříček nejlepších hráčů.
2. Systém uživateli nabídne volbu mezi různými sporty a formami hry.
3. Uživatel si jednu z nich vybere.
4. Systém uživateli zobrazí všechny hráče daného sportu a formy ze všech klubů, kteří odehráli alespoň jeden zápas a seřadí je podle úrovně sestupně. Kromě úrovně mu zobrazí i počet odehraných zápasů a jeho samotného v tabulce zvýrazní.

3.4.6 Správa události

Jakýkoliv správce klubu může spravovat události v něm. Každá událost musí být aktivně spravována alespoň jedním správcem, aby událost mohla přecházet mezi fázemi svého životního cyklu.

Aktér: Správce

Pre–condition:

- Správce spravuje nějaký klub.

Hlavní scénář: Správa události

1. Správce se rozhodne vytvořit událost.
2. Systém správci nabídne přidání detailů k události.
3. Správce zvolí název, datum a případně i čas, či místo události.
4. Systém správce vpustí do události.
5. V den události umožní systém správci událost spustit.
6. Po spuštění může správce v této události spravovat turnaje.
7. Správce může kdykoliv událost ukončit.

3.4.7 Rozhodnutí žádosti uživatele k připojení se na událost

Jakýkoliv správce klubu může rozhodovat o žádostech k připojení se na libovolnou událost. Pokud se správce rozhodne uživatele odmítnout, bude uživateli nabídnuto poslání nové žádosti. Uživatel může do události vstoupit pouze tehdy, až je přijat některým ze správců.

Aktér: Správce

Pre–condition:

- Správce spravuje nějaký klub

- V klubu je vytvořena událost.
- Na událost podal žádost k připojení se alespoň jeden uživatel.

Hlavní scénář: Rozhodnutí žádosti uživatele k připojení se na událost

1. Správce se rozhodne vyřídit žádost k připojení na konkrétní událost.
2. Systém správci zobrazí všechny žádosti k dané události.
3. Správce zvolí jednu nebo více těchto žádostí a zvolí, zda je chce přijmout nebo odmítnout.
4. Systém uživatele, kteří byli přijati, vpustí do události. Naopak těm, kterým byla žádost zamítnuta, ji umožní znovu podat.

3.4.8 Správa turnaje

Jakýkoliv správce, který může spravovat událost, v ní může spravovat i turnaje. Turnaje mají podobné fáze jako události a je tedy nutné aby ji nějaký správce vytvořil, spustil a následně ukončil.

Aktér: Správce

Pre–condition:

- Správce spravuje nějakou spuštěnou událost.

Post–condition:

- Detaily o dokončeném turnaji jsou uloženy.

Hlavní scénář: Správa turnaje

1. Správce se rozhodne vytvořit turnaj.
2. Systém správci nabídne zvolení si formátu hry.
3. Správce si zvolí formát.
4. Systém správci zobrazí právě připojené hráče.
5. Až bude správce spokojený s počtem připojených hráčů, rozhodne se turnaj spustit.
6. Systém vygeneruje plán turnaje a správci ho zobrazí.
7. Správce může kdykoliv turnaj ukončit.

3.4.9 Jmenování správce

Vlastníci klubu mohou jmenovat libovolného uživatele správcem, který pak za ně může spravovat události. Uživatel však toto jmenování musí potvrdit. Vlastník sám je automaticky správcem.

Aktér: Vlastník

Pre–condition:

- Existuje alespoň jeden další uživatel.
- Vlastník má přiřazen nějaký klub.

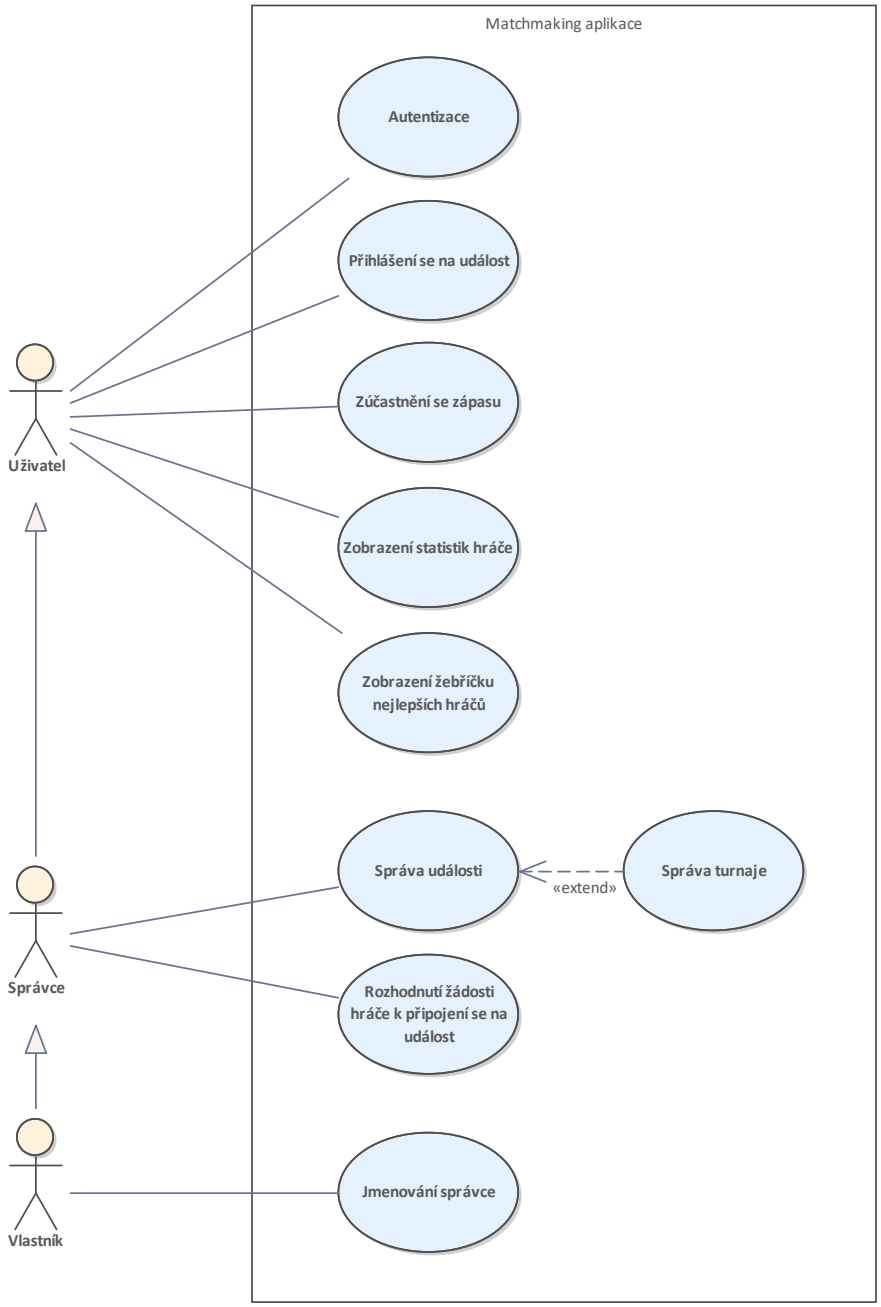
Post–condition:

- Zvolený uživatel se stal správcem klubu.

Hlavní scénář: Jmenování správce

1. Vlastník se rozhodne jmenovat konkrétního uživatele správcem nějakého jeho klubu.
2. Systém vlastníkovi zobrazí všechny uživatele aplikace.
3. Vlastník si vybere onoho uživatele a dotyčnému bude poslána žádost o přijetí.
4. Uživatel se rozhodne o přijetí. Pokud odmítne, začíná scénář od začátku. Pokud přijme je scénář u konce.

Obrázek 3.4 Diagram případů užití



3.5 Namapování funkčních požadavků na případy užití

Jelikož případy užití jsou detailní specifikací funkčních požadavků, měli by je všechny pokrývat. Následující tabulka slouží pro ověření právě této skutečnosti.

Tabulka 3.1 Tabulka namapování funkčních požadavků na případy užití

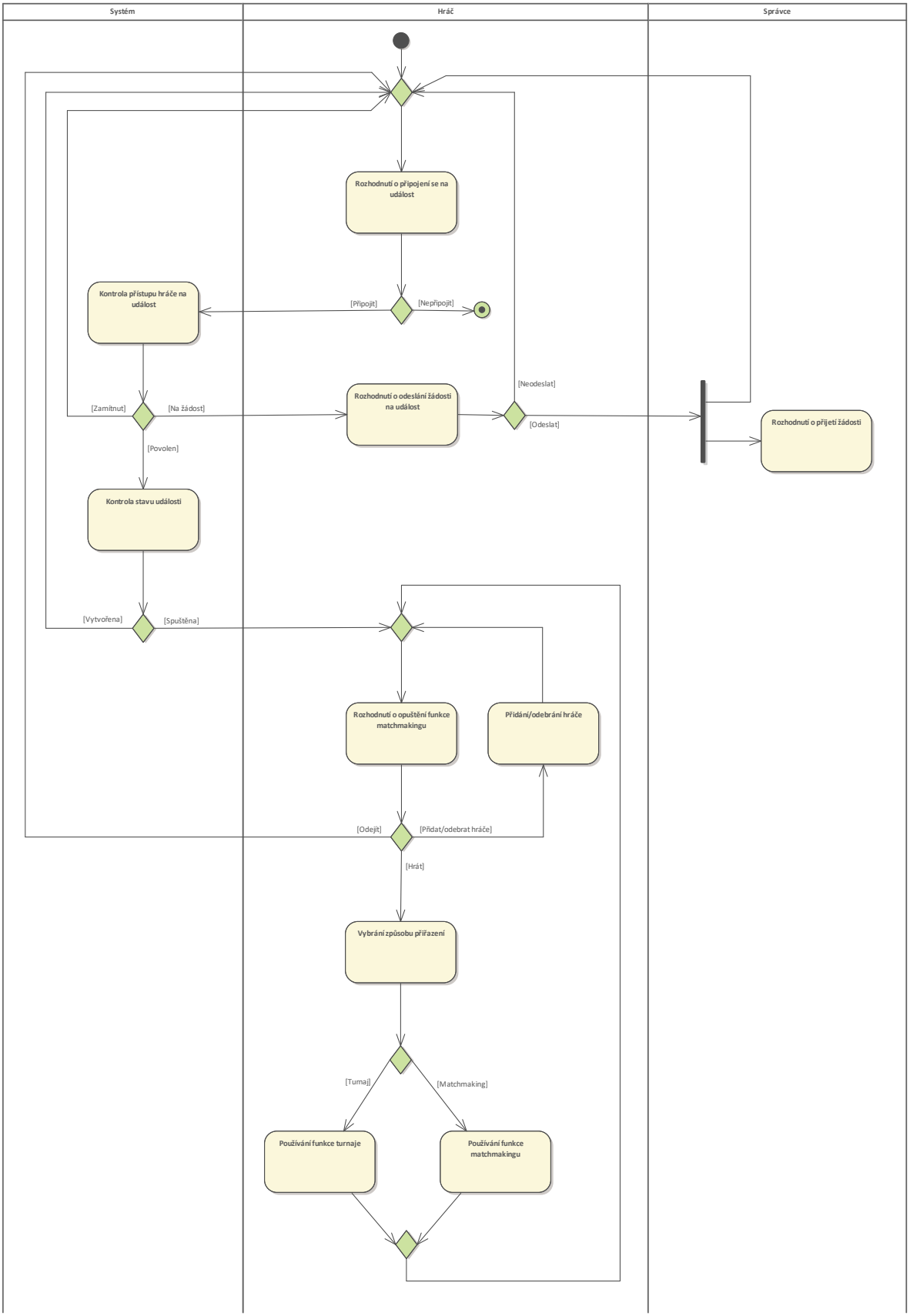
Funkční požadavky \ Případy užití	3.4.1	3.4.2	3.4.3	3.4.4	3.4.5	3.4.6	3.4.7	3.4.8	3.4.9
3.2.1	+								
3.2.2	+								
3.2.3	+								
3.2.4						+			
3.2.5		+							
3.2.6			+						
3.2.7			+						
3.2.8								+	
3.2.9			+						
3.2.10			+						
3.2.11			+						
3.2.12			+						
3.2.13			+						
3.2.14				+					
3.2.15					+				

3.6 Diagramy aktivit

Diagramy aktivit graficky znázorňují jednotlivé procesy probíhající uvnitř dané domény. Z tohoto diagramu lze snadno vyčíst strukturu konkrétního procesu, tedy jaké akce jsou v rámci něj vykonávány, v jakém pořadí a jak se větví. Diagram 3.5 znázorňuje proces použití aplikace hráčem[3] vzhledem k jedné konkrétní události. V případě, že se jedná o hráče, který skupinu nevede ale pouze příjme pozvání do skupiny, vše za něj od té chvíle řeší vedoucí. Z diagramu je patrné, že pokud se hráč chce zúčastnit matchmakingu nebo turnaje, musí nejdříve podat žádost o připojení se na konkrétní událost a být následně schválen některým ze správců.

[3] Konkrétně vedoucím skupiny

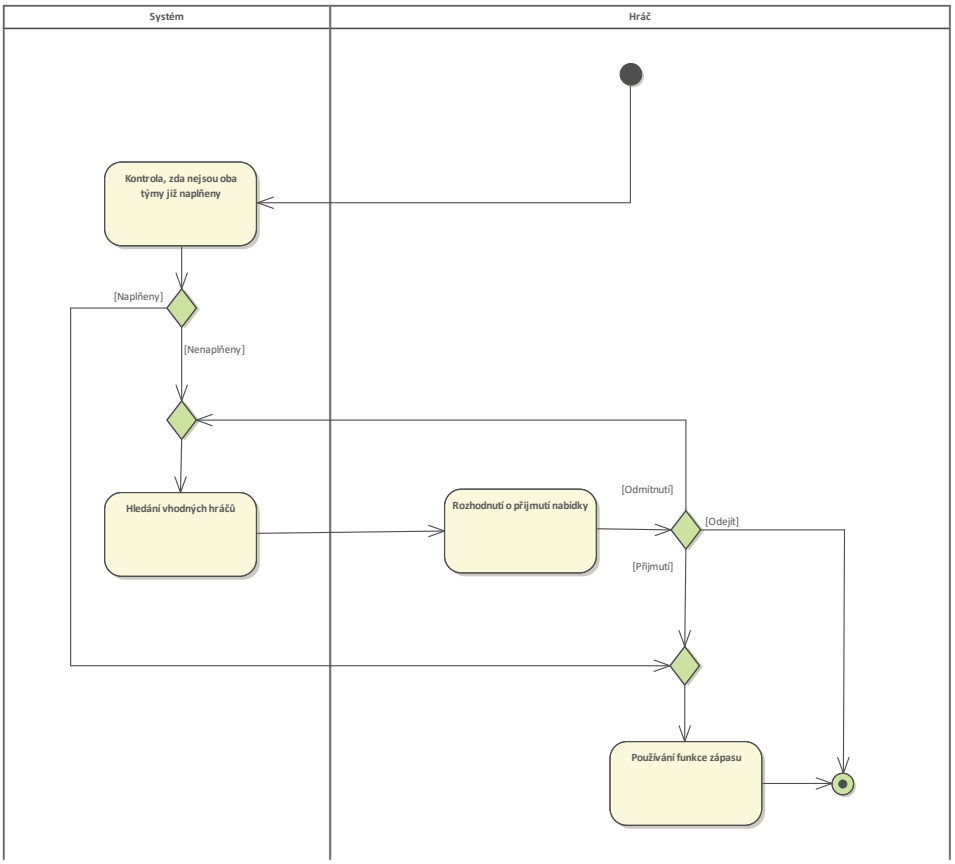
Obrázek 3.5 Diagram aktivit použití aplikace



3.6.1 Matchmaking

Následující diagram 3.6 detailně zachycuje proces používání funkce matchmakingu. Stejně jako předchozí diagram je proces vnímán z pohledu vedoucího skupiny a předpokládá se že konkrétní událost je po celou dobu spuštěná. Skupina je nejdříve zkontrolována, zda již není plná a v případě, že ano, je jí hned spuštěn zápas. V opačném případě je zařazena do samotného matchmakingu a při nalezení vhodného zápasu je vedoucímu skupiny poslána nabídka na přijetí.

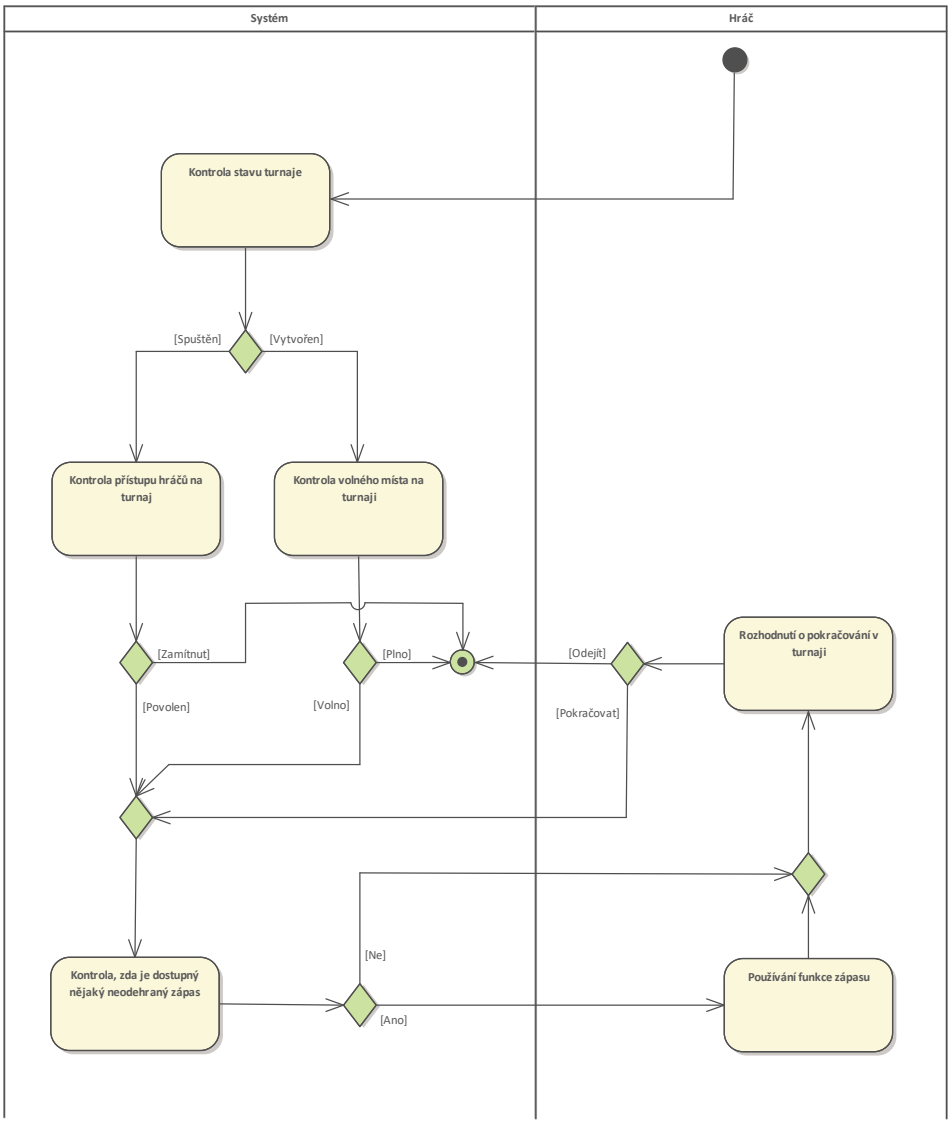
Obrázek 3.6 Diagram aktivit matchmakingu



3.6.2 Turnaj

Funkce turnaje, taktéž z pohledu vedoucího skupiny, je zase znázorněna na diagramu 3.7. Předpokladem je nejen spuštěná událost, ale i na něm vytvořený turnaj. V rámci turnaje je, stejně jak u události, kontrolován přístup dané skupiny a to různými způsoby v závislosti na tom v jaké fázi turnaj je. V případě, že je pouze vytvořený, spočívá kontrola pouze ve zjištění volného místa. Po spuštění už k turnaji ale mají přístup pouze ti, co se do něj připojili již v předchozí fázi. Ve spuštěném turnaji pak skupiny odehrávají pouze zápasy, které jim byli vygenerovány při spuštění.

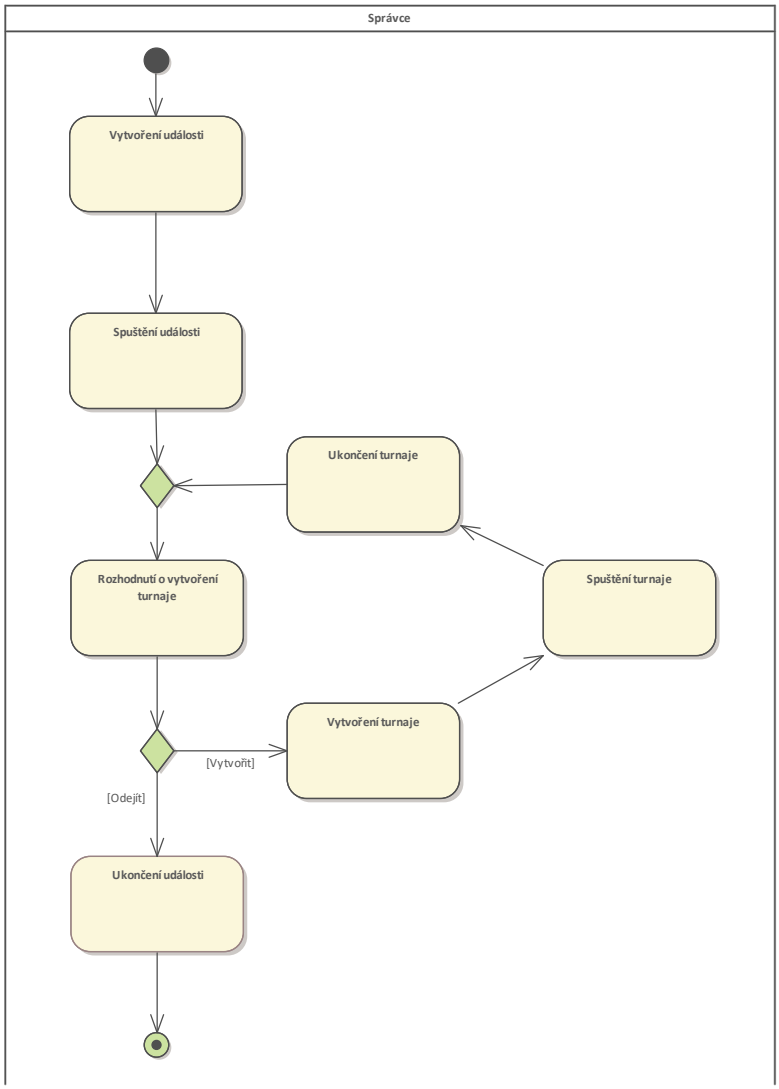
Obrázek 3.7 Diagram aktivit turnaje



3.6.3 Správce

Správa nejen událostí, ale i turnajů v ní vytvořených je zachycena na diagramu 3.8. Turnajů může být na jedné události vytvořeno v jeden moment více, ale každý musí být jiného formátu. Na zmíněném diagramu se uvažují turnaje pouze stejného formátu. Jak vyplývá z diagramu správce je zodpovědný za přechod mezi jednotlivými fázemi událostí a turnajů. Bez správce tedy nemůže událost ani turnaj projít celým svým životním cyklem.

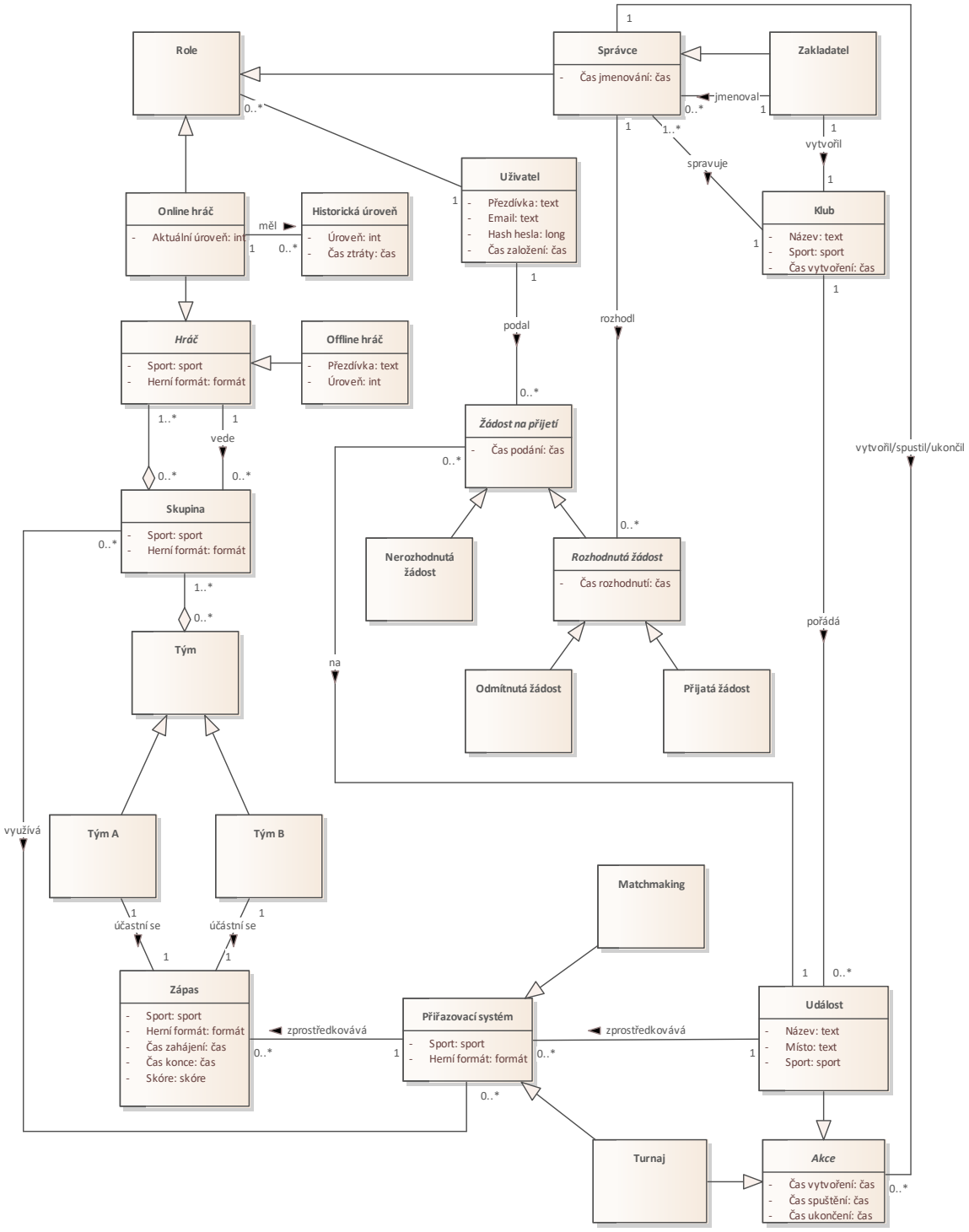
Obrázek 3.8 Diagram aktivit správce



3.7 Doménový konceptuální model

Doménový konceptuální model 3.9 znázorňuje datové struktury celého systému a vztahy mezi nimi. Tento diagram by nám měl pomoct nejen se samotnou analýzou problémové domény, ale i s výběrem vhodné databáze a její následnou tvorbou. Z modelu jasně vyplývá, že s uživatelem může být spojeno více rolí nejen správce, ale i hráče a proto je nutné je odlišit. Každá role hráče bude spojena s jedním sportem a jejím herním formátem. Hráči budou moci být součástí skupin, které mohou využívat přiřazovací systémy a ze kterých se budou následně skládat jednotlivé týmy v konkrétních přiřazených zápasech.

Obrázek 3.9 Doménový konceptuální model



Kapitola 4
Návrh

Po důsledném porozumění problematice můžeme přejít na návrh samotného řešení. Navrhnout je potřeba nejenom vhodné technologie se kterými se bude pracovat, ale i architekturu programu a některé specifické algoritmy v něm použité.

4.1 Algoritmy

Hlavní funkcí aplikace je hodnocení hráčů a jejich následné párování. Tyto problémy však nemají jednoznačné řešení a pro optimální výsledky je nutné tyto algoritmy optimalizovat. V této práci se však budu zabývat pouze návrhem samotných algoritmů, nikoliv jejich optimalizací, jelikož toto téma by možná samo o sobě obsáhlo vlastní akademickou práci.

4.1.1 Určování úrovně hráče

Tímto problémem se zabývalo již mnoho autorů a mohu si tedy vybrat z již existujících řešení a pouze poupravit pro moje účely.

4.1.1.1 Elo rating

Nejstarší a nejpoužívanější ze všech hodnotících systémů, především kvůli své relativní jednoduchosti. Tento systém využívá teorie pravděpodobnosti pro určení šancí každého hráče na výhru v závislosti na jejich úrovních. Tato šance se pak odráží na velikosti změny úrovně. Čím nižší je šance hráče na výhru, tím více se jeho úroveň zvýší v případě výhry. Naopak v případě prohry s vyšší šancí na výhru hráč více ztrácí. Nevýhodou je, že tento systém počítá pouze s formou zápasů 1v1. [7]

4.1.1.2 Glicko

Je založen na normálním rozdělení a dá se považovat za zobecnění Elo ratingu. Díky tomu, že součástí hodnocení je i rozptyl vyjadřující neurčitost výkonu, lze pomocí něho lépe zachytit skutečnou úroveň hráče. Existuje i vylepšená varianta Glicko 2, která dokonce počítá s časem jako dalším zdrojem neurčitosti, což vede k lepšímu odhadu aktuální síly hráče a napomáhá lepšímu určení změny úrovně všech účastníků zápasu. Glicko taktéž ve své originální podobě počítá pouze se zápasy typu 1v1 a vzhledem k jeho složitosti by jeho úprava byla mnohem náročnější. Na druhou stranu se dost možná jedná o mnohem přesnější model. [8][9]

4.1.1.3 Trueskill

Systém hodnocení založený na systému Glicko. Byl vyvinutý společností Microsoft pro matchmaking v počítačových hrách. Počítačové hry dovolují ze zápasu získávat mnohem více informací, jako styl hraní, pohyby, atd. Se všemi těmito informacemi Trueskill dokáže pracovat a díky tomu je tento systém daleko přesnější. Systém taktéž podporuje hodnocení hráčů v rámci skupin. I když by se hodilo mít hodnocení přesnější, u reálných zápasu to nebude moc významné, jelikož v tomto prostředí je daleko těžší získat ze zápasů všechny data nutná k vyhodnocení herních stylů a výkonnosti hráčů. V rámci klubu je reálné maximálně zpracování výsledného skóre. Systém je taktéž proprietární a není tedy jasné jak funguje a za použití si Microsoft účtuje poplatky. [10]

4.1.1.4 Rozhodnutí

I přesto, že Glicko systém je v mnoha ohledech lepší jak Elo rating, není kvůli své složitosti vhodný pro projekt této velikosti. Nestačí jenom naprogramovat samotný algoritmus a přizpůsobit ho pro kolektivní sporty, ale navíc je potřeba i složitější optimalizace pro dosažení optimálního výsledku. Elo rating je z hlediska optimalizace nejjednodušší, jelikož obsahuje nejméně parametrů a jde do něj lépe vidět. Tento systém bude však nutné upravit aby mohl hodnotit hráče i na základě skupinových zápasů. Stačí však jen skupinu považovat jako jednoho hráče s úrovní rovnou součtu úrovní jejích členů a posléze při přepočtu úrovně odrazit podíl člena na úrovni skupiny ve velikosti změny jeho hodnocení.

Pro výpočet nové úrovně R'_a se používá následující vzorec:

$$R'_a = R_a + K\left(S_a - E_a\right),$$

kde R_a je původní úroveň hráče, K nebo–li K–faktor je konstanta, která určuje jak moc daný výsledek ovlivní hráčovo skóre. Čím je K–faktor větší, tím má poslední výsledek zápasu větší vliv na skóre hráče. Pokud tedy bude zvolen K–faktor příliš veliký, bude hráčova úroveň vypovídat především o výkonu v poslední hře. Pokud bude naopak příliš malý, bude skóre hráčů málo responzivní. S_a vyjadřuje výsledek zápasu, kde 1 odpovídá výhře, 0 prohře a $0,5$ remíze. Pravděpodobnost hráče na výhru E_a se zase vypočítává následujícím způsobem:

$$E_a = \frac{1}{1 + 10^{\frac{R_b - R_a}{400}}},$$

ve kterém R_b vyjadřuje původní úroveň protivníka. Konstanta 400 byla vybrána z toho důvodu, aby hráč s úrovní o 200 větší než soupeř měl šanci na výhru přesně 75%. Změněním této konstanty mohu tedy docílit jiné váhy úrovní, což však není potřeba. [7]

Určení K–faktoru a počátečního skóre hráčů je naopak velmi důležité a proto tyto konstanty převezmu z jedné z nejznámějších implementací tohoto systému, čímž je hodnotící systém FIDE používaný v prostředí šachů. Dle návodu pro pořadatele šachových turnajů FIDE, je hráči nejdříve přiřazen K–faktor 40 a až po dosažení 30 zápasů se mu sníží na 20, případně pokud dosáhne úrovně vyšší než 2400 je mu K–faktor snížen až na 10. Dále je stanoveno, že pokud rozdíl úrovní hráčů překročí hodnotu 400, bude se jako rozdíl uvažovat právě toto maximum. Počáteční úroveň sice nikde stanovená není, ale jelikož se průměrná úroveň hráčů pohybuje kolem 1500, myslím si, že právě tato hodnota bude vhodnou volbou. [11]

Tato část bude taktéž vyvíjena s ohledem na možné vylepšení či úplné nahrazení Elo ratingu přesnějším systémem, jako například Glicko. Jedním z možných vylepšení by mohlo být zohledňování i samotného skóre zápasu při přepočtu úrovně hráčů.

4.1.2 Matchmaking

Na rozdíl od předchozího problému, se tímto moc autorů nezabývalo a ti, kteří ano, tak se až moc zaměřují na specifické prostředí počítačových her a pro tento projekt jsou tak nepoužitelné.

Řešení budu tedy muset vymyslet sám a nebudu tak brát velký zřetel na efektivnost, či optimálnost z hlediska uživatelské spokojenosti.

Po algoritmu je požadováno aby skupiny hráčů propojoval mezi sebou tak, aby vytvořili celý tým schopný zápasu a zároveň aby jeho nabídky byly co nejoptimálnější z hlediska vhodnosti přiřazení. Vhodnost přiřazení bude vnímána především jako rozdíl celkových úrovní týmů, kde úroveň týmu bude zohledňovat nejenom úrovně jejich členů, ale i vhodnost spoluhráčů, taktéž interpretovanou jako průměr rozdílů jejich úrovní. Vhodnost spoluhráčů se může odrazit na jejich spolupráci a dynamice a zvýšit tak nejen celkovou sílu daného týmu, ale i požitek ze hry. Bude tedy nutné stanovit vztah mezi těmito vlastnostmi, jak určením podílu vhodnosti spoluhráčů na úrovni týmu, tak i na celkové vhodnosti přiřazení. Algoritmus bude po určitých časových intervalech vybírat takovou kombinaci připojených skupin, která bude průměrem přes všechny zápasy co nejvhodnější. Následně všem vedoucím skupin nabídne vygenerované zápasy a při přijmutí zápasu všemi zúčastněnými skupinami dotyčné odpojí z matchmakingu a vytvoří pro ně zápas.

Tento algoritmus by šlo dále vylepšit určením vztahu mezi vhodností týmů a jejich prioritou v rámci čekání, tak aby se dostalo na každého a hráči s hodně odlišnou úrovní nebyli úplně ignorováni. Vývoj bude počítat s možným rozšířením či úplným nahrazením algoritmu, pro účely této práce bude však implementována pouze jeho základní varianta.

4.2 Technologie

I když se k popisu architektury dostanu až v další kapitole, je důležité již nyní poukázat na fakt, že se jedná o mobilní aplikaci shromažďující data a propojující více uživatelů. Bude tedy nutné použít architekturu typu klient-server, z čehož i plyne nutnost stanovení si technologií jak pro klienta, tak i server a s ním související databázi.

4.2.1 Klient

Mobilní aplikace mohou být vyvíjeny buď nativně, tedy přímo jazykem podporovaným danou platformou, nebo multiplatformně pro více platforem naráz. Výhodou multiplatformního vývoje je rychlejší dodání aplikace na obě zařízení a její konzistence mezi různými platformami. Debugovaní na dvou platformách však konzistentní už není a je třeba testovat na obou zařízeních. Většina multiplatformních nástrojů je taktéž pozadu od těch nativních a neposkytují veškeré nejnovější funkce dané platformy. Na druhou stranu vývoj pro každou platformu zvlášť by byl nejen časově náročný, ale vyžadoval by taktéž více znalostí, protože Android a iOS mají různé nativní programovací jazyky [12]. Z multiplatformních prostředí pro vývoj mobilních aplikací přichází v úvahu pouze Flutter, React Native a Xamarin. Zvolil jsem si React Native, pro jeho velkou komunitu a tedy i spoustu dokumentů a návodů [13]. Navíc jazyk, který React Native používá mi je už do určité míry známý. [14]

React Native framework využívá pro tvorbu aplikací JavaScript. Nevýhodou JavaScriptu je, že se jedná o dynamicky typovaný jazyk a tedy při kompilaci nevyhodí chyby týkající se špatně vybraného typu, což stěžuje odchycení těchto chyb. Tohle však lze obejít používáním nadstavby JavaScriptu, která je React Nativem taktéž podporovaná, nazývající se TypeScript. TypeScript se od JavaScriptu liší hlavně v tom, že je staticky typovaným jazykem a je tedy třeba typ proměnné definovat už při deklaraci. [15]

4.2.2 Server

I přes to, že zde je na výběr spoustu serverových prostředí, které by pro můj účel byly vhodné, rozhodl jsem se pro Node.js a to hlavně z toho důvodu, že Node.js taktéž jako React Native

používá programovací jazyk JavaScript[1] a některé části kódu budou podobné ne–li úplně stejné. Navíc, jelikož na projektu pracuji sám, nebude mě zatěžovat přecházení z jednoho jazyka na druhý. [16]

4.2.3 Databáze

Databáze lze rozdělit na dva typy: SQL nebo–li relační databáze a NoSQL taktéž nazývány nerelační databáze. Dále podrobně popíši tyto dva typy databází a zvolím si tu nejlepší pro vyvíjenou aplikaci.

4.2.3.1 SQL databáze

SQL databáze jsou tu poměrně dlouho a právě díky tomu jsou ověřeným řešením mnoha projektů. Pro dotazování se používá SQL dotazovací jazyk, který je, až na menší odlišnosti, pro všechny tyto databáze stejný. Jako své hlavní struktury používá tabulky, do kterých se jako řádky ukládají záznamy nebo–li vstupy uživatele a které jsou strukturované na jednotlivé atomické informace pomocí sloupců tabulky. Rozdělení záznamů tabulky na sloupce jasně definuje jejich strukturu, která je pro všechny záznamy tabulky stejná a nelze ji porušit. Navíc podporují transakce, které díky dodržování vlastností ACID poskytují ochranu vůči různým chybám a zaručují validitu dat. Relační se jim říká proto, že tabulky mohou být mezi sebou propojeny relacemi pomocí tzv. primárních a cizích klíčů. Jsou proto vhodné pro datové modely, kde jsou informace mezi sebou hodně propojené a často je nutné tyto informace spojovat dohromady. [17]

4.2.3.2 NoSQL databáze

NoSQL databáze naopak nepoužívají dotazovací jazyk SQL a nejsou založené na relačním principu. Jsou celkem čtyři základní typy NoSQL databází.

Nejjednodušším typem jsou databáze typu klíč-hodnota, které se od těch relačních liší hlavně tím, že každý záznam má vždy pouze dva sloupce, tedy klíč a hodnotu a pro náš problém je tedy nevhodná. Dalším typem jsou sloupcově orientované databáze používané především v analytice, kvůli svému způsobu zapisování do sloupců, které zajišťuje snazší analýzu malého počtu sloupců dané databáze. Grafové databáze jsou zase určeny pro lepší práci s daty, ve kterých existuje spoustu vztahů a používá se tedy například pro zachycení znalostních grafů. Pro nás jsou však relevantní pouze grafy dokumentové. [18]

Dokumentové databáze se vyznačují vyšší rychlostí jednoduchých operací na úkor paměťové náročnosti a konzistence dat. Na rozdíl od SQL databází nejsou používány tabulky s pevně danou strukturou, ale tzv. kolekce, které každý vstup ukládají do dokumentů. Dokumenty sice strukturované jsou, ale v rámci stejné kolekce se mohou dokumenty strukturou lišit. Toto je výhodné především pro data, která nemají pevně danou formu a nemohou tedy vyhovět pevně dané struktuře tabulek u SQL databází. Díky této jednoduché implementaci, jsou v některých oblastech rychlejší než databáze relační [19]. Rychlost lze taktéž zvýšit vnořováním dokumentů do sebe a vytvořit tak strukturu, která bude obsahovat vše potřebné, bez nutnosti používání další kolekce. To však povede k obtížnějšímu udržování konzistence dat, způsobené jejich duplikací a tedy větší paměťovou náročností. Její jednoduchá struktura a atomičnost na úrovni jednotlivých záznamů/dokumentů přispívá také k její jednoduché horizontální škálovatelnosti, tedy rozprostření dat v rámci jedné databáze mezi více serverů. [17]

Výhody nad relačními databázemi jsou vyšší rychlost u jednoduchých operací na datech s málo relacemi, lepší škálovatelnost a flexibilní schéma.

[1] Případně i TypeScript

4.2.3.3 Rozhodnutí

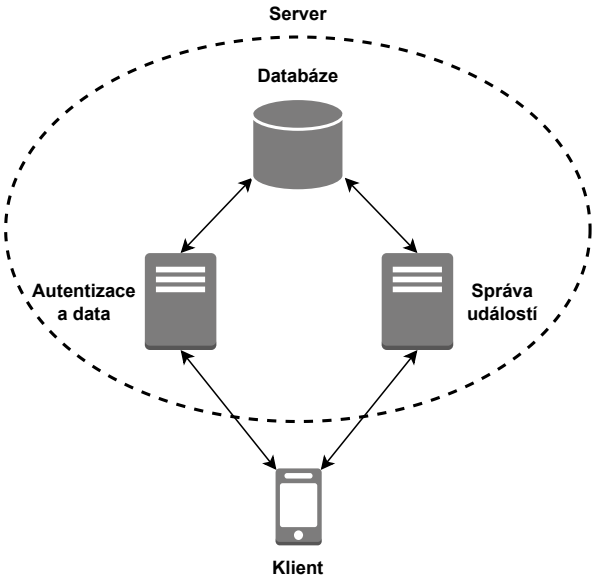
Dle diagramu tříd mé domény se domnívám, že při využití vnořování dokumentů bych u dokumentových databází mohl docílit vyšší rychlosti. Přidaná paměťová náročnost by nebyla tak výrazná a díky tomu, že úprava, či mazání dat by měly být ojedinělé operace, tak i horší konzistence dat je akceptovatelná. Snadnější škálovatelnost je taktéž pro můj projekt vítanou vlastností. Flexibilní schéma pro mě však nepředstavuje velkou přidanou hodnotu, jelikož jediné využití by našlo pouze ve sjednocení zápasů různých forem, čehož lze však docílit také v relačních databázích vynecháním některých hodnot. Rozhodující pro mě byl nakonec fakt, že se stále jedná o poměrně novou technologii a některé zkušenosti napovídaly, že může docházet ke ztrátě dat [20]. To ostatně potvrzuje i fakt, že tyto databáze dlouhou dobu ani nepodporovali transakce, které by zaručovali konzistenci dat a základní pojištění před různými chybami. Duplikace a nekonzistence dat by v pozdějších fázích mohla také vést k nepřehlednosti a potenciálně zhoršené analýze těchto dat nebo migraci do jiné databáze. Dokumentové databáze se tedy spíše hodí pro rychlou práci s jednoduchými daty, jejichž možná ztráta není nijak závažná. Například tedy pro ukládání logů, zpráv atd. Pro můj projekt je tedy nevhodná. Z těchto důvodů jsem si zvolil databáze relační. Výběr konkrétního databázového systému spočíval především v rozhodnutí se mezi MySQL a PostgreSQL. MySQL je sice jednodušší, ale zase nenabízí tolik funkcí a mám s ním méně zkušeností, proto jsem dal přednost PostgreSQL. [21]

4.3 Architektura

Jak již bylo řečeno, bude se jednat o architekturu typu klient-server. Tato informace nám však pouze říká, že systém bude složen ze dvou částí: server, který bude centrálně spravovat data a obsluhovat připojená zařízení a klient, jež se bude na server připojovat a zobrazovat data uživateli. Je však důležité si taktéž přesně zadefinovat tyto dvě části, určit přesný způsob komunikace mezi nimi a zvolit architekturu i pro ně samotné.

Klientů se sice může připojit více, ale vždy se bude jednat o jeden a ten samý systém. Server je naopak nutné rozdělit na dva podsystémy, jelikož jeho zodpovědností je nejenom autentizace a spravování dat, ale také správa událostí, spočívající ve spojování uživatelů v reálném čase. Jeden podsystém se bude tedy starat pouze o autentizaci a správu dat a nebudou na něj tedy kladeny nároky na vysokou rychlost komunikace, či průtok dat. Naopak pro druhý podsystém bude rychlost komunikace zásadní, jelikož správa událostí zahrnuje mimo jiné i správu herních skupin a samotný algoritmus pro matchmaking, který je hlavní funkcí celého systému a plynulost je zde velmi důležitá. Oba podsystémy budou komunikovat s databází a klientem zvlášť, bez vzájemného propojení.

Obrázek 4.1 Celková architektura systému



4.3.1 API

API je soubor pravidel a protokolů určující rozhraní, pomocí kterého bude moci jeden systém komunikovat s druhým. Při výběru vhodného API je nutné zvážit nejenom požadovanou rychlost a bezpečnost komunikace, ale i například složitost či časovou náročnost implementace.

4.3.1.1 SOAP

SOAP je API, které je definováno velmi přísnými pravidly. Jeho implementace je tedy náročná a ne vždy uzpůsobovatel všem požadavkům. Výhodou je však zase vysoká bezpečnost a stabilita komunikace, díky čemuž je toto řešení oblíbené především v korporátní sféře.

4.3.1.2 REST

REST je naopak ve specifikaci svých pravidel mnohem volnější a jedná se tak spíš o seznam několika doporučení a pravidel. To má za následek jednoduchost, rychlost a univerzálnost celého řešení. V tomto kontextu by se dalo tedy vnímat jako jasný opak SOAP. Jedním z hlavních znaků REST API je, že musí být beze stavový a nesmí tedy ukládat žádné informace o stavu klienta. Klient by tedy měl v každém požadavku poskytnout všechny data nutná k jejímu vyřízení. Toto pravidlo zajišťuje jednoduchost a uniformnost celého řešení. Právě proto je REST považován za standard při vývoji serveru mobilních a webových aplikací. Serverová část je však rozdělena na dva podsystémy, přičemž znaky klasického serveru pro mobilní aplikace naplňuje pouze podsystém starající se o správu dat a autentizaci. Z tohoto důvodu bude REST použit právě pouze na tomto podsystému. [22]

4.3.1.3 WebSocket

WebSocket na rozdíl od předchozích dvou řešení podporuje obousměrnou komunikaci. Je vhodný tedy v případech, kdy nestačí pouhé vyřizování požadavků klienta serverem, ale je nutné taktéž posílat data klientovi nezávisle na jeho požadavcích. I když obousměrnosti komunikace jde

dosáhnout i v REST, za pomocí takzvaného short nebo long pollingu, nejedná se o konkurenci WebSocket. U WebSocket totiž probíhá navázání komunikace pouze jednou a to hned na začátku. Poté mohou být všechna data přenášena v obou směrech až do ukončení komunikace bez nadbytečných dat ve formě hlaviček. Jedná se tedy o ideální volbu pro serverovou část starající se o správu událostí, kde je právě rychlost prioritní. [23]

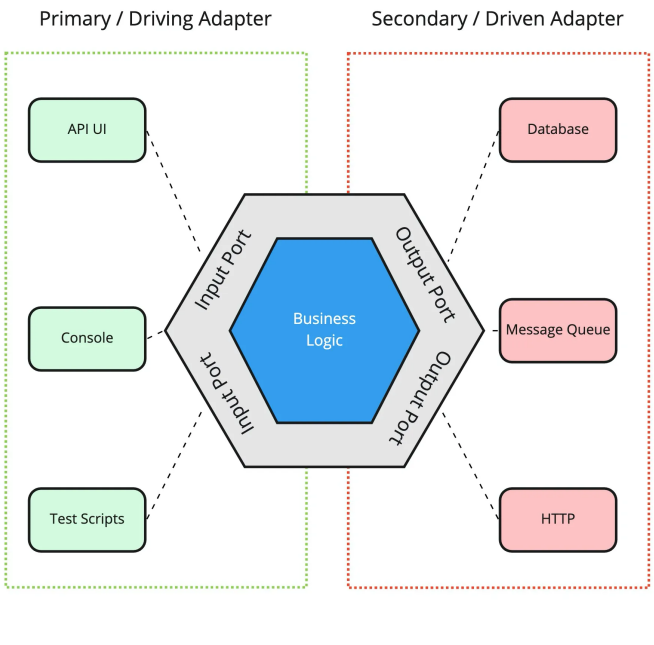
4.3.2 Oddělení zodpovědností a závislostí

V rámci architektury je taktéž důležité si určit jakým způsobem bude zdrojový kód strukturován. Vhodné oddělení zodpovědností a závislostí nám totiž nejenom dokáže zpřehlednit kód, ale i zjednodušit testování, či případné přidávání dalších funkcionalit. Části kódu, kterým je přiřazena jasná zodpovědnost a které nejsou závislé na implementaci jiných částí, lze totiž velmi snadno nahradit.

4.3.2.1 Hexagonální architektura

Přesně k takovému účelu byla vytvořena architektura hexagonální. Ta rozděluje kód na samotné jádro aplikace obsahující byznys logiku a adaptéry, které se dále dělí na primární a sekundární, zprostředkovávající komunikaci s vnějšími systémy. Primární adaptéry se starají o komunikaci se systémy, které aplikaci využívají. Může se tedy jednat například o klienta nebo nějakého administračního nástroje. Sekundární zase zajišťují komunikaci se systémy využívanými aplikací. Tedy například databáze nebo služby třetích stran. Oddělení byznys logiky od vnějších systému je zajištěno právě adaptéry, které s jádrem komunikují přes jasně definované porty.

Obrázek 4.2 Hexagonální architektura [24]



Na hexagonální architekturu navazuje i takzvaná onion a clean architektura. V zásadě se však liší pouze vyšší granularitou vrstev, kterou tento systém nevyžaduje. [25]

Kapitola 5

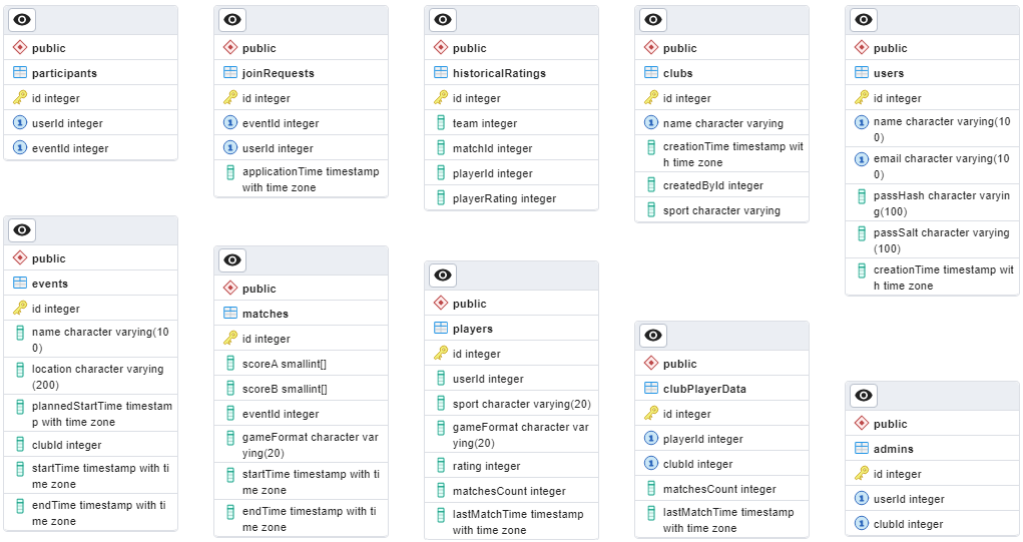
Implementace

V této kapitole budou popsány veškeré nástroje a postupy použité při realizaci navrženého systému.

5.1 Databáze

Doménový konceptuální model 3.9 vytvořený v rámci analýzy výrazně pomohl při tvorbě databáze, jelikož zachycuje hlavní struktury v doméně a jejich vztahy. Nicméně některé struktury byly spíše abstraktního charakteru a jejich existence měla pouze zachytit různé vztahy v doméně, není databáze přesnou kopií konceptuálního modelu. Pro realizaci některých vztahů vyjádřených v konceptuálním modelu, bylo dokonce nutné vytvořit struktury nové. Databáze se tedy s konceptuálním modelem shoduje pouze částečně. Navíc některé struktury nemohly být implementovány v rámci databáze a byly místo toho pojaty jako vlastní struktury správce událostí.

Obrázek 5.1 Schéma databáze



5.2 Sdílené knihovny

Díky tomu, že všechny části systému jsou psány v jazyce TypeScript, bylo možné sdílet určité části kódu. Existuje několik způsobů, jak provést sdílení, avšak většina z nich značně komplikuje vývojový proces a zpomaluje práci. Proto jsem se rozhodl pro metodu pomocí nástroje npm link, která využívá symbolických odkazů pro sdílení společného kódu. Při použití TypeScriptu však vznikají určité problémy s touto metodou, protože knihovna musí být po každé změně znovu sestavena do podoby transpilovaného JavaScriptu, aby mohla být použita v jiných projektech. Přesto se však tato obtíž nevyrovná obrovskému přínosu sdílených knihoven, který spočívá nejen v úspoře psaní a zpřehlednění kódu, ale také v kontrole typů dat přenášených mezi klientem a serverem v případě TypeScriptu.

5.2.1 Sports

Knihovna, která se zabývá definicí sportu, zjednodušuje jejich využití v dalších projektech. Obsahuje kompletní definice všech sportů, což umožňuje projektům, které využívají tuto knihovnu, přistupovat ke sportům obecně a snadno tak rozšiřovat systém o nové sporty pomocí úprav této knihovny. Kromě seznamu sportů obsahuje knihovna tedy také jejich definice, které zahrnují například všechny její možné herní formáty, maximální počet hráčů, definici platného skóre nebo vítěze zápasu.

```
const sports = new Map<string, SportData>([
[
"badminton",
{
        gameFormats: new Map([
        ["singles", { teamSize: 1, maxMatchTime: 3600000 }],
        ["doubles", { teamSize: 2, maxMatchTime: 3600000 }],
        ]),
        scoreIsValid: BadmintonScore.isValid,
        isWinner: BadmintonScore.isWinner,
},
],
]);
```

Výpis kódu 1 Definice sportů

Tento výpis kódu zobrazuje právě seznam všech sportů. Přidání sportu do tohoto seznamu přímo odpovídá rozšíření o podporovaný sport v celém systému.

```
export function isWinner(
score: Array<number>,
opponentScore: Array<number>
): boolean {
        let wonSets = 0;
        let lostSets = 0;
        for (let round = 0; round < score.length; round++) {
                if (score[round] > opponentScore[round]) wonSets++;
                else lostSets++;
        }

        return wonSets > lostSets;
}
```

Výpis kódu 2 Funkce na ověření výhry

Zde je zase funkce použitá v předešlém seznamu, sloužící pro zjištění vítěze zápasu v závislosti na skóre.

5.2.2 TransferData

Data předávaná mezi klientem a serverem jsou definována právě v této knihovně. Její použití výrazně usnadňuje kontrolu přijímaných dat na obou stranách. Nicméně, jelikož TypeScript umožňuje kontrolu pouze primitivních typů a ne objektů, bylo nutné definovat tyto funkce pro každý datový typ samostatně. Tímto způsobem jsem zajistil, že přenášená data jsou validní a odpovídají očekávaným formátům i přes tato omezení TypeScriptu.

```
export interface PlayerData {
        userName: string;
        rating: number;
}
export function isPlayerData(obj: any) {
        return (
        obj && typeof obj.userName === "string" &&
        typeof obj.rating === "number"
        );
}
```

Výpis kódu 3 Definice struktury PlayerData

Pod každou definicí objektu následovala funkce pro její kontrolu. Ta při úspěchu zároveň přiřadí kontrolovanému objektu daný typ díky User-Defined Type Guards.

```
export interface LobbyData {
        userId: number;
        leaderId: number;
        sublobbyA: Array<LobbyMemberData>;
        sublobbyB: Array<LobbyMemberData>;
}
export function isLobbyData(obj: any): obj is LobbyData {
        return (
        obj &&
        typeof obj.userId === "number" &&
        typeof obj.leaderId === "number" &&
        isArray(obj.sublobbyA, "object", isLobbyMemberData) &&
        isArray(obj.sublobbyB, "object", isLobbyMemberData)
        );
}
```

Výpis kódu 4 Definice struktury LobbyData

V případech kdy objekt obsahoval list, bylo nutné zkontrolovat i každou jeho položku a kontrola se takhle mohla i několikrát vnořit.

5.2.3 Repositories

Poslední z těchto knihoven slouží pro komunikaci s databází PostgreSQL, byla však navržena pro podporu jakéhokoliv zdroje dat bez nutnosti zásahu do projektů ji využívajících. Jako jediný repozitář nebyl sdílený mezi všemi, ale pouze mezi serverovými částmi a dokonce i zde se využití překrývalo jen částečně, takže oddělení do vlastní knihovny nebylo nutné. Na druhou stranu to kód u obou mírně zpřehlednilo a oddělilo zodpovědnosti.

```
public async add(
        clubId: number,
        eventName: string,
        location: string | null,
        plannedStartTime: Date | null
): Promise<void> {
        const query = {
                name: "events.add",
                text: `
                INSERT
                INTO events("clubId", name, location, "plannedStartTime")
                VALUES ($1, $2, $3, $4)`,
                values: [clubId, eventName, location, plannedStartTime],
        };
        await this.pool.query(query);
}
```

Výpis kódu 5 Přidání události do databáze

Jelikož pro komunikaci s databází byla využívána knihovna node-postgres byly operace jako přidání události velmi jednoduché a při použití parametrizovaného query nebylo ani nutné

ošetřovat vstupy proti SQL injection, jelikož to dělala knihovna sama. Právě zde jde i vidět, co jsem zmiňoval v části databáze a to že PostgreSQL má problém s rozlišováním velkých a malých písmen. Problémem se to stává právě když pro své proměnné používám camelCase, který je pro typescript doporučován. Naštěstí jde tento problém obejít bez toho aniž bych musel používat dvě různé konvence a to uvozováním názvů obsahující velká písmena v query.

```
public async add(
userId: number,
clubName: string,
sport: string,
creationTime: Date
): Promise<void> {
        const client = await this.pool.connect();
        try {
                await client.query("BEGIN");
                const clubQuery = {
                        name: "clubs.add",
                        text: `
                        INSERT
                        INTO clubs(name, sport, "creationTime", "createdById")
                        VALUES ($1, $2, $3, $4)
                        RETURNING id`,
                        values: [clubName, sport, creationTime, userId],
                };
                const clubId: number =
                        (await client.query(clubQuery)).rows[0].id;
                const adminQuery = {
                        name: "admins.add",
                        text: `
                        INSERT
                        INTO admins("userId", "clubId")
                        VALUES ($1, $2)`,
                        values: [userId, clubId],
                };
                await client.query(adminQuery);
                await client.query("COMMIT");
        } catch (err) {
                await client.query("ROLLBACK");
                throw err;
        } finally {
                client.release();
        }
}
```

Výpis kódu 6 Přidání klubu do databáze

V situacích, kdy na sobě upravované tabulky vzájemně závisely a nebylo by vhodné upravit jednu bez druhé, bylo nutné využít transakce. Tato knihovna implementuje mechanismus transakcí pomocí try-catch bloků, které slouží k zachycení potenciálních chyb a vrácení změn zpět[1]

[1] rollback

v případě potřeby. V případě úspěšného průběhu transakce jsou zase změny potvrzeny[2], čímž se trvale zapíší do databáze. Tímto způsobem knihovna zajistí, že všechny změny provedené v rámci transakce budou provedeny úspěšně a bezpečně.

5.3 Server

Na obou serverových částí bylo dodržováno principů hexagonální architektury a jelikož obě komunikují pouze s klientem a databází, bylo možné kód rozdělit na dvě části. Oba obsahovaly controllers, starající se o komunikaci a zpracování dat do požadovaného stavu, které následně zpracuje část vykonávající byznys logiku. V případě serveru pro správu dat a autentizaci to byly services, které z většiny obsahují pouze kontrolu smysluplnosti příchozího dotazu a případné použití databáze přes knihovnu Repositories. Naopak správce událostí k těmto services obsahuje ještě vlastní struktury, které zachycují stav dané události a taktéž vykonávají pouze byznys logiku.

5.3.1 Správce dat a autentizace

Jak již bylo řečeno tato část slouží jako prostředník mezi klientem a databází, přičemž hlídá zda jsou prováděny pouze povolené operace. Při vývoji byl využit framework Express.js, který výrazně zjednodušuje vytvoření a nastavení REST API. Na této části je taktéž implementována autentikace, která je dále používaná v celém systému. Autentikace je zajištěna pomocí JWT tokenů, které se generují při každém úspěšném přihlášení a jsou poslány klientovy, který se jimi posléze ověřuje bez nutnosti opakovaného zadávání hesla. JWT tokeny se po získání ukládají do bezpečného úložiště, kde zůstávají i po vypnutí aplikace a není tak nutné se znovu přihlašovat. V prvotním přihlášení je však nutné heslo poslat. Z důvodu bezpečnosti se tak posílá již zašifrované, přičemž na serveru se šifruje znovu a to se solí tak, aby mohlo být bezpečně uloženo v databázi.

```
public add = async (req: Request, res: Response) => {
        try {
                const userId: number = res.locals.userId;
                if (!Client.isAddParticipantData(req.body))
                throw new WrongDataFormatError();
                await this.participantService.add(
                userId,
                req.body.targetUserId,
                req.body.eventId
                );
                const msg: string = "Participant role created";
                res.status(StatusCodes.CREATED).json(msg);
        } catch (err) {
                errorHandler(err, res);
        }
};
```

Výpis kódu 7 Controller pro přidání účastníka

Tato část kódu je konkrétní metodou participant controlleru, která kontroluje zda žádost o přidání účastníka na událost obsahuje všechna potřebná data. Knihovna TransferData nám zde velmi pomáhá při kontrole daného typu.

[2] commit

```
  public async add(
userId: number,
targetUserId: number,
eventId: number
): Promise<void> {
        const clubId = await this.rep.eventRepository.getClubId(eventId);
        if (!clubId) throw new WrongDataError();
        if (!(await this.rep.adminRepository.exists(userId, clubId)))
        throw new Unauthorized();
        await this.rep.participantRepository.add(targetUserId, eventId);
}
```

Výpis kódu 8 Service pro přidání účastníka

V případě že žádost úspěšně projde předchozím controllerem, pokračuje právě touto service. Ta kontroluje, zda odesílatel žádosti má potřebná oprávnění a pokud ano, provede operaci na databázi.

5.3.2 Správce událostí

Serverová část starající se o správu událostí. Pro rychlý přístup k datům a jednoduchou manipulaci s nimi, ukládá jejich část do vlastních struktur, kde je společně se services prováděna byznys logika. V těchto strukturách je prováděn i samotný matchmaking, pro který je nutné si udržovat mimo jiné i aktuální seznam připojených skupin nebo nabídnutých zápasů. Uživatelé mají taktéž svoji vlastní strukturu, ve které jsou na serveru udržováni a která obsahuje i jejich vlastní WebSocket, tak aby jim server mohl kdykoliv poslat zprávu. Uživatelé jsou taktéž povinně členy skupin, ve kterých se pak mohou účastnit matchmakingu. Struktura skupiny zase umožňuje snadné adresování všech uživatelů v ní připojených. Matchmaking se provádí periodicky podle stanovené konstanty a po nabídnutí zápasu mají vedoucí skupin čas do dalšího matchmakingu na přijmutí. K zahájení zápasu dochází pouze tehdy, když je navržený zápas přijmutý všemi skupinami[3]. Jelikož správce událostí používá pro komunikaci s klientem WebSocket, autentikace pomocí JWT tokenů je nutná pouze při připojení. Připojení se na socket se provádí vždy se specifikovaným ID události a odpovídá tak tedy i připojení se na samotnou událost. Stejně tak odpojení se od WebSocket je ekvivalentní opuštění události.

[3] respektive jejími vedoucími

```
public invite(user: User, targetUser: User, sublobby: "A" | "B") {
        if (!this.isLeader(user)) throw new NotLeaderOfLobbyError();
        if (this.isMember(targetUser)) throw new AlreadyMemberOfLobbyError();
        if (this.matchmakingGroup) throw new InMatchmakingError();

        if (sublobby === "A") this.invitedA.add(targetUser);
        else if (sublobby === "B") this.invitedB.add(targetUser);
        const data: Server.LobbyInvitationData = {
                leaderId: user.getUserId(),
                leaderName: user.getUserName(),
                sublobby,
        };
        targetUser.send(Server.LOBBY_INVITATION_RESPONSE, data);
}
```

Výpis kódu 9 Pozvání do skupiny

Tato funkce kontroluje zda je uživatel, který zve, oprávněn tuto akci provést a následně posílá pozvánku uživateli, který je zván.

I přes to, že funkce matchmakingu je na této části velmi významná, je pro svou délku a složitost nevhodná na vypsání. Zmínit lze však část funkce na hodnocení, která je taktéž důležitou součástí celé aplikace.

```
export default (
sport: string,
teamRatingA: number,
teamRatingB: number,
scoreA: Array<number>,
scoreB: Array<number>
): { teamRawRatingDifferenceA: number; teamRawRatingDifferenceB: number } => {
        const probOfWinningA =
        1 / (1 + Math.pow(10, (teamRatingA - teamRatingB) / 400));
        const probOfWinningB = 1 - probOfWinningA;
        const simpleResultA = Sports.isWinner(sport, scoreA, scoreB) ? 1 : 0;
        const simpleResultB = 1 - simpleResultA;

        const teamRawRatingDifferenceA = simpleResultA - probOfWinningA;
        const teamRawRatingDifferenceB = simpleResultB - probOfWinningB;
        return { teamRawRatingDifferenceA, teamRawRatingDifferenceB };
};
```

Výpis kódu 10 Částečný výpočet nové úrovně

Zde je uplatněna část vzorce na výpočet nového hodnocení. K-faktor je zde vynechán, aby mohl být později uplatněn hráčův vlastní K-rating na jeho nové hodnocení a nikoliv skupinový.

5.4 Klient

Klient používá UI knihovnu React Native Paper, která značně usnadňuje design celé aplikace a knihovnu React Navigation používanou k přechodu mezi různými obrazovkami.

Kapitola 6
Uživatelské testování

Aplikace proběhla uživatelským testováním, kterého se zúčastnili čtyři lidé. Každý z těchto lidí prošel třemi testovacími scénáři a následně podal zpětnou vazbu k tomu jak se scénáře podařilo projít.

6.1 Testovací scénáře

6.1.1 Testovací scénář 1

1. Zaregistruj si nový účet
2. Přihlas se na tento účet
3. Přejdi do vytvořeného klubu pro badminton
4. Požádej o přijmutí na některou z událostí v tomto klubu
5. Po přijmutí vejdi do události
6. V události přidej hráče do svého týmu
7. Spusť zápas
8. Zadej skóre
9. Přejdi do svého profilu a zkontroluj, zda je zde zápas již zapsaný

6.1.2 Testovací scénář 2

1. Vytvoř klub a v něm událost
2. Počkej až někdo podá žádost o připojení se
3. Potvrď jeho žádost
4. Odstartuj událost
5. Připoj se do ní
6. Připoj se do matchmakingu a počkej až se ti nabídne nějaký návrh na zápas

7. Odmítni ho a odejdi z matchmakingu
8. Ukonči událost
9. Aktualizuj seznam událostí a zkontroluj, že smazaná událost chybí

6.1.3 Testovací scénář 3

1. Podej žádost o připojení se do probíhající události
2. Počkej až tvoji žádost někdo potvrdí
3. Připoj se do události a čekej ve své skupině dokud tě někdo nepozve do své vlastní
4. Nejprve odmítni nabídku
5. Poté ji přijmi
6. Nech vedoucího skupiny zahájit matchmaking
7. Počkej než se zápas najde, spustí a nakonec ukončí
8. Odejdi z události a zkontroluj zda se zápas propsal do tvé úrovně

6.2 Zpětná vazba ke scénářům

Z testování vyplynuly následující nedostatky aplikace. Prvním z problémů bylo nepřehledné zadávání skóre výsledků zápasů, u kterého nebylo jasné k jaké straně patří jaké skóre. Další výtkou byl případ, kdy došlo k neshodě zadaného skóre oběma hráči a nebylo na to upozorněno. Posledním poznatkem bylo, že vytvořený klub nebo událost bylo velmi snadné bez potvrzení smazat.

Kapitola 7
Návrhy na zlepšení

V této kapitole představuji návrhy na zlepšení, které buď přímo vyplynuly z předchozích kapitol nebo jsou výsledkem aktivního hledání nových přístupů.

7.1 Návrhy vyplývající z uživatelského testování

Z výsledků testování jsem dospěl k potřebě upravit uživatelské rozhraní především při zadávání skóre zápasu. Dále pak nutnost potvrzení při důležité operaci, jako je například mazání klubu nebo opuštění matchmakingu. Neupozornění na neshodné skóre naopak nevidím jak veliký problém, především proto, že tato situace stejně nejde nijak ovlivnit. Možným řešením by však bylo penalizování hráčů u kterých opakovaně dochází k podobným neshodám.

7.2 Návrhy vyplývající z nesplněných požadavků

Funkční požadavky, které nakonec implementovány nebyly představují jasný předmět dalšího vývoje. Jelikož se ale jednalo o požadavky s nízkou prioritou neznamená to nutně, že by při dalším vývoji měli být upřednostňovány. Jedním z těchto požadavků je registrace uživatelského účtu přes služby třetích stran. Vzhledem k tomu, že knihovny zprostředkovávající tuto funkci jsou, stejně jako současný autentizační systém, založené na tokenech, neměla by implementace představovat sebemenší problém. Používání aplikace bez registrace je dalším z nesplněných požadavků, u kterého si však již nejsem jistý zda by pro aplikaci představoval jakýkoliv užitek. To stejné však neplatí o možnosti přidání offline hráče do skupiny. Dle mého názoru by užitek takové funkce výrazně převýšil náročnost její implementace. Aktuálně totiž neexistuje možnost účasti osoby, která není uživatelem, což snižuje přívětivost celého systému. Další z důležitých požadavků je vytváření a správa turnajů. Turnaj je taktéž jistým typem matchmakingu, který je v klubech velmi oblíbený. V rámci práce ale nebyl implementován hlavně z důvodu velké náročnosti. Proto by tato funkcionality byla nejspíš hlavním bodem dalšího vývoje.

7.3 Další návrhy

Tyto návrhy lze rozdělit na tři kategorie: nové funkcionality, zdokonalení těch současných a zlepšení infrastruktury, spočívající především ve zvýšení bezpečnosti a škálovatelnosti.

7.3.1 Nové funkcionality

Ověřování emailu, změna hesla, kontrola maximální délky vstupu nebo filtry pro snazší vyhledávání v seznamech jsou funkce, které jsou velmi užitečné a neměly by tedy chybět ani v této aplikaci. Žádoucí je taktéž možnost přerušení kteréhokoliv načítání bez nutnosti restartovat aplikaci, přidání možnosti zobrazení statistik ostatních hráčů, graf s historií elo ratingu nebo možnost filtrovat oblíbené kluby či přihlášené události.

7.3.2 Zdokonalení současných funkcionalit

Zdokonalit lze především samotné algoritmy matchmakingu a hodnocení hráčů. Vylepšení by mohlo spočívat buď pouze v úpravě již existujících algoritmů a jejich optimalizaci nebo dokonce jejich úplným nahrazením. V současných algoritmech je mimo jiné velký prostor pro zlepšení ve zvolení vhodnějších konstant pro K-faktor a počáteční úroveň. Zohledněni skóre při výpočtu nové úrovně hráče by taktéž mohla výrazně pomoci přesnosti hodnocení.

7.3.3 Zlepšení infrastruktury

V rámci serveru by bylo vhodné omezit maximální počet požadavků, které může jeden klient poslat za určitý časový úsek, tak aby nedošlo k zahlcení v případě útoku. Pro bezpečnost je také nutné ověřovací JWT tokeny periodicky zneplatňovat a generovat nové, aby v případě odcizení tokenu získal útočník kontrolu nad účtem pouze dočasně. S debugováním by zase mohlo pomoci zaznamenávání veškerých operací provedených na serveru pomocí logů. Za zvážení také stojí použití některé z in-memory databází, jako například Redis, pro lepší škálovatelnost.

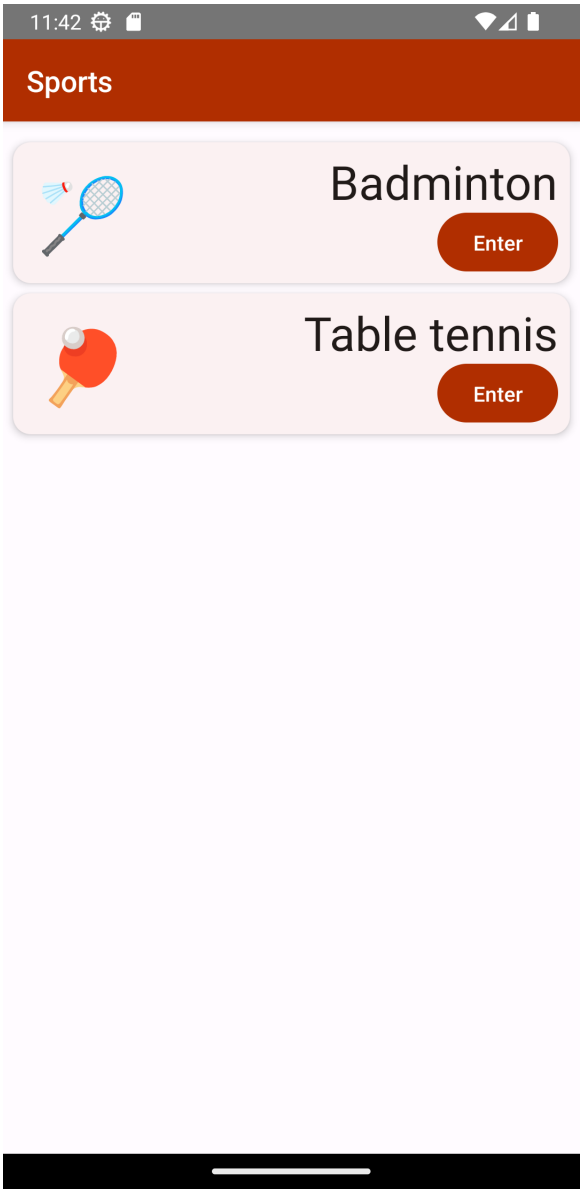
Kapitola 8
Závěr

V této práci jsem úspěšně vyvinul mobilní aplikaci pro hodnocení a matchmaking hráčů badmintonu. Nejdříve jsem sestavil seznam konkrétních požadavků na základě rešerše konkurenčních aplikací. Tyto požadavky jsem následně pomocí modelu případů užití jasně specifikoval a s využitím diagramů tříd a aktivit důkladně zanalyzoval problémovou doménu matchmakingu hráčů badmintonu pomocí mobilní aplikace. Po této analýze jsem se přesunul na samotný návrh řešení, kde jsem systém rozdělil na klienta a dvě rozdílné serverové části. První část spravující především autentizaci a přístup k datům a komunikující s klientem přes REST API a druhou využívající WebSocket API pro rychlou odezvu při vytváření skupin a matchmakingu na samotných událostech. Pro obě tyto části byla zvolena databáze PostgreSQL, architektura MVC a běhové prostředí Node.js. Pro klienta byla zase vybrána knihovna React Native, podporující multiplatformní vývoj. Navržené řešení jsem následně implementoval a řádně otestoval. Ze stanovených požadavků na aplikaci se mi podařily implementovat nejen všechny nutné, ale i některé volitelné. Příkladem můžou být tedy nejen hlavní funkce jako samotný matchmaking, automatický přepočet úrovně nebo zobrazení žebříčku nejlepších hráčů, ale i zobrazení statistik hráče a jeho historii zápasů nebo možnost účastnit se matchmakingu jako skupina. Navíc díky důslednému naplnění nefunkčních požadavků bude další vývoj v podobě rozšíření o nový sport nebo překlad otázkou pouze několika minut.

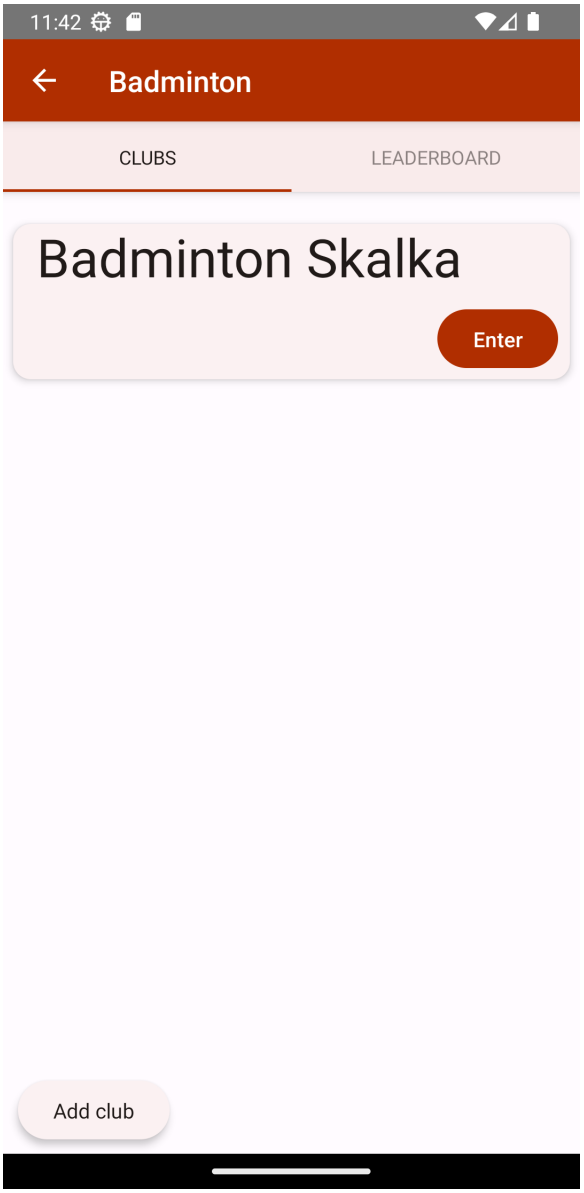
Příloha A

Snímky obrazovky aplikace

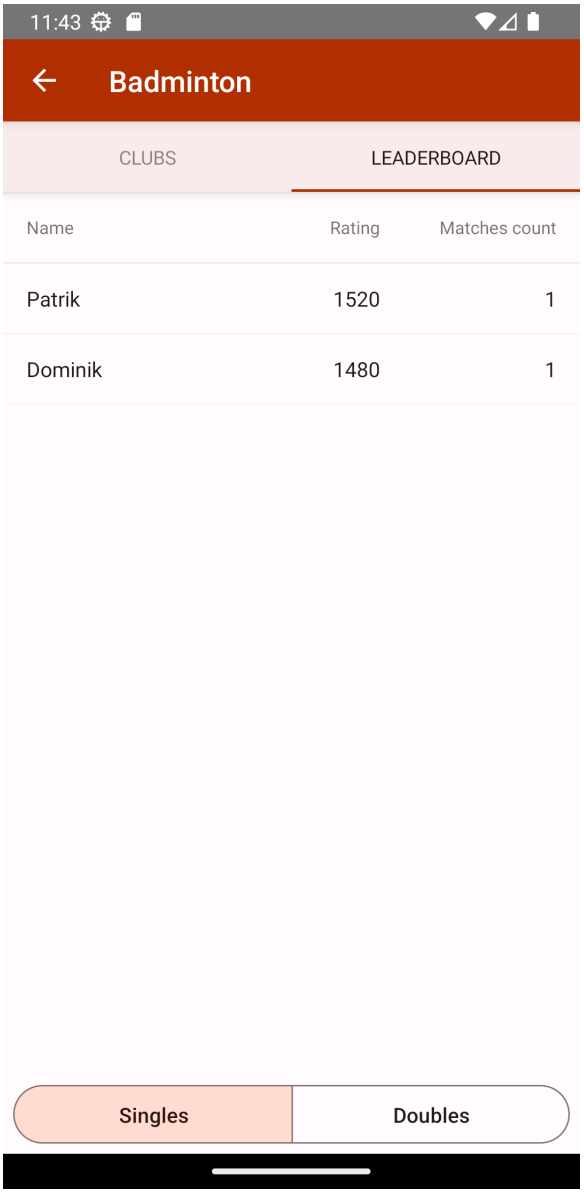
Obrázek A.1 Obrazovka sportů



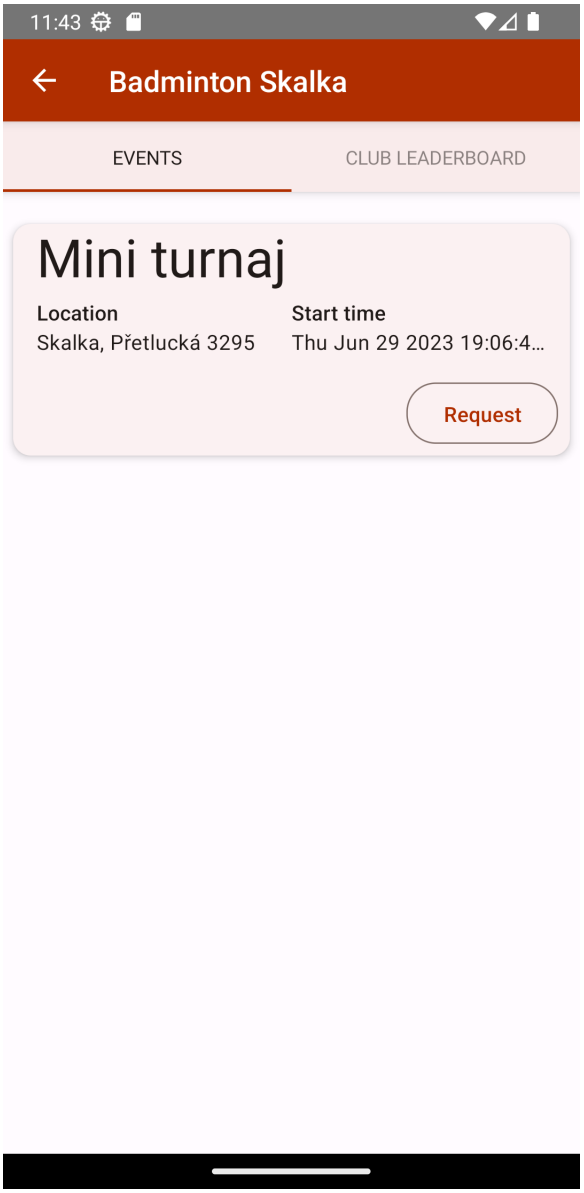
Obrázek A.2 Obrazovka klubů



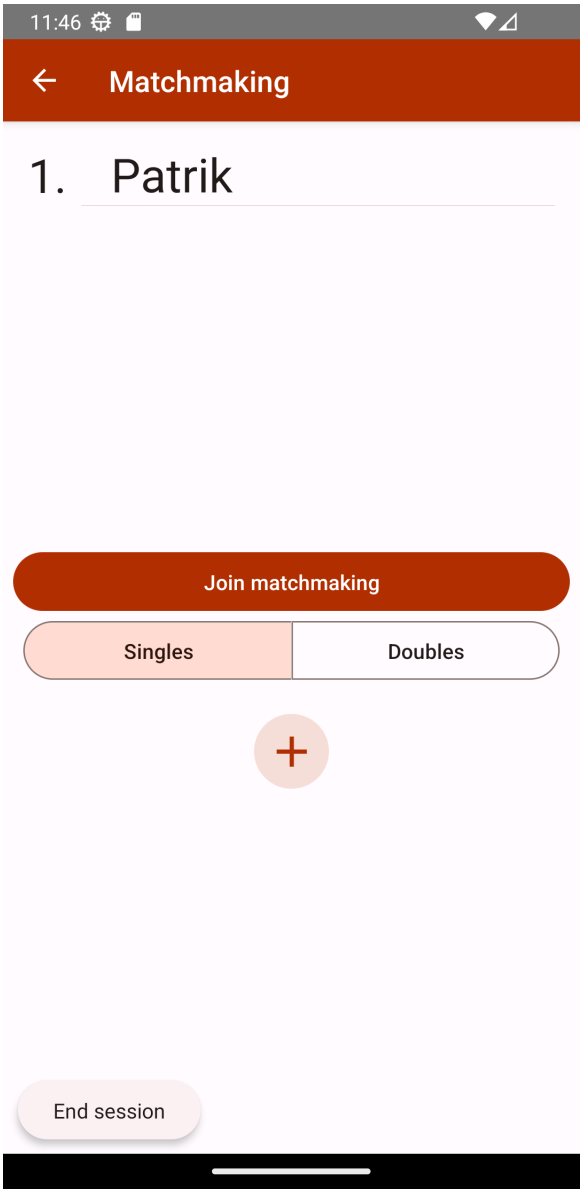
Obrázek A.3 Obrazovka žebříčku



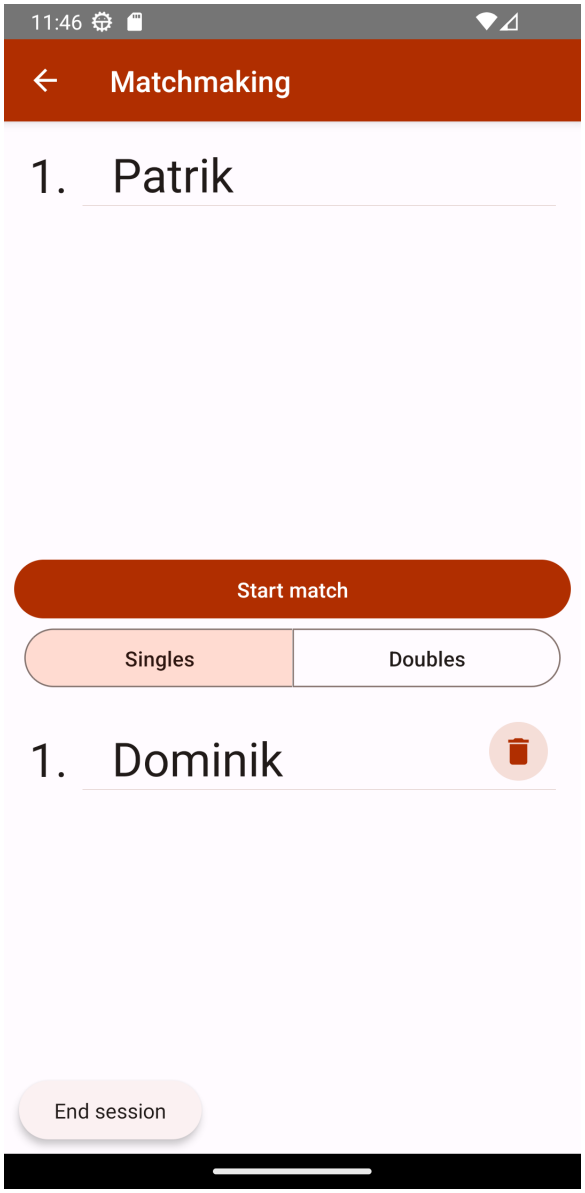
Obrázek A.4 Obrazovka událostí



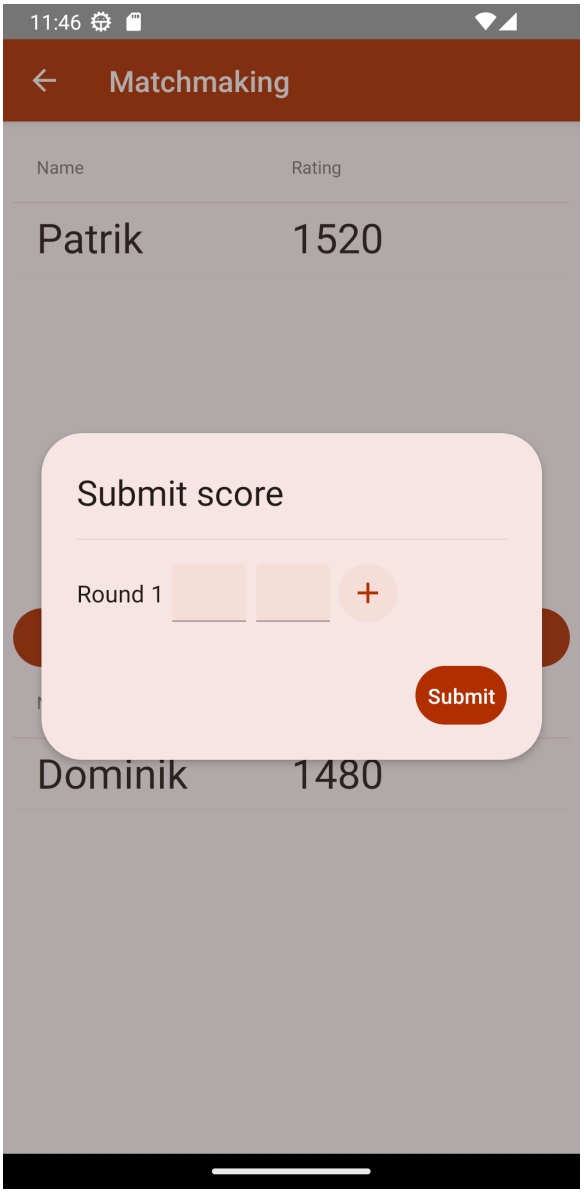
Obrázek A.5 Obrazovka skupiny



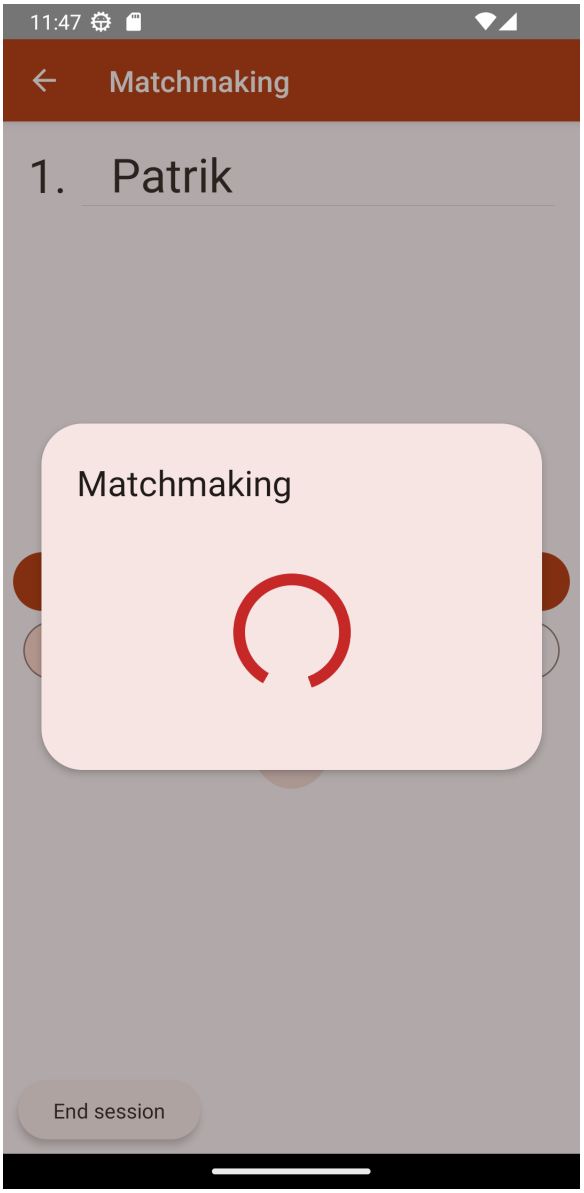
Obrázek A.6 Obrazovka skupiny se spoluhráčem



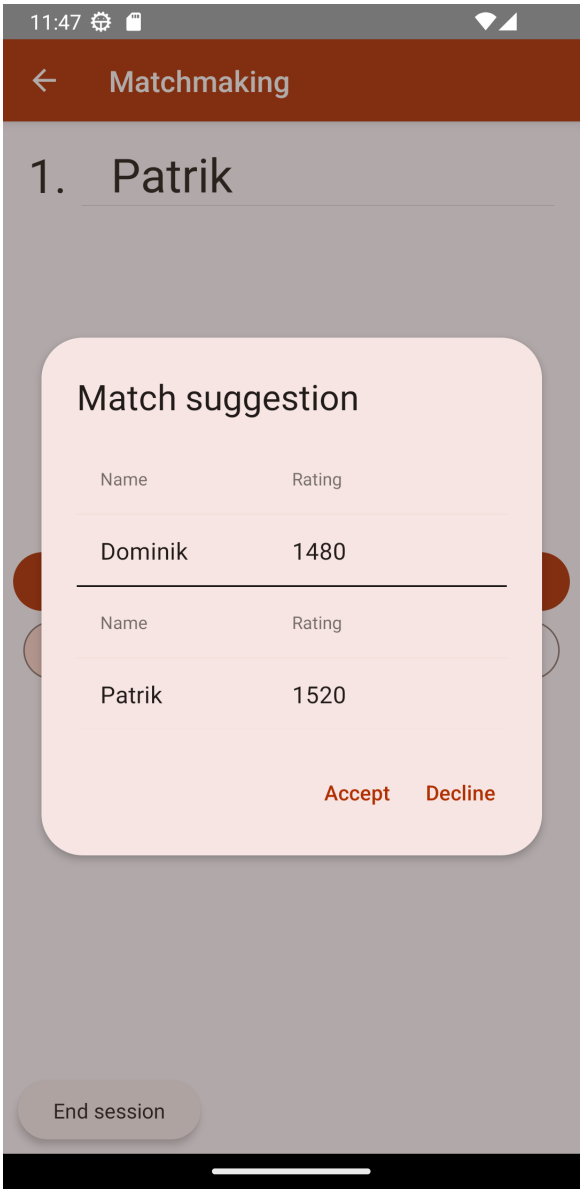
Obrázek A.7 Obrazovka zadávání skóre



Obrázek A.8 Obrazovka matchmakingu



Obrázek A.9 Obrazovka doporučeného zápasu



Bibliografie

1. *The Laws of Badminton* [online]. Badminton BC. [cit. 2023-03-12]. Dostupné z: https://www.badmintonbc.com/page/2888/The-Laws-of-Badminton.
2. WANG, Ning; WU, Jie. Latency Minimization Through Optimal User Matchmaking in Multi-Party Online Applications. In: *2018 IEEE 19th International Symposium on "A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*. 2018, s. 1–10. Dostupné z DOI: 10.1109/WoWMoM.2018.8449817.
3. *RacketPal* [online]. Racketpal Ltd. Dostupné také z: https://racketpal.co.uk.
4. *Elo Challenge* [online]. Dostupné také z: https://elosportschallenge.wordpress.com/category/elo-challenge-app.
5. *Winner - Tournament Maker App* [online]. Talent Apps. Dostupné také z: https://play.google.com/store/apps/details?id=il.talent.winner&hl=en_US.
6. *MoSCoW Prioritisation* [online]. Agile Business Consortium. [cit. 2023-03-12]. Dostupné z: https://www.agilebusiness.org/dsdm-project-framework/moscow-prioririsation.html.
7. VEISDAL, Jørgen. *The Mathematics of Elo Ratings* [online]. [cit. 2023-03-12]. Dostupné z: https://www.cantorsparadise.com/the-mathematics-of-elo-ratings-b6bfc9ca1dba.
8. GLICKMAN, Mark E. *The Glicko system* [online]. [cit. 2023-03-12]. Dostupné z: http://www.glicko.net/glicko/glicko.pdf.
9. GLICKMAN, Mark E. *Example of the Glicko-2 system* [online]. [cit. 2023-03-12]. Dostupné z: http://www.glicko.net/glicko/glicko2.pdf.
10. *TrueSkill™ Ranking System* [online]. Microsoft Corporation. [cit. 2023-03-12]. Dostupné z: https://www.microsoft.com/en-us/research/project/trueskill-ranking-system.
11. *Arbiters' manual 2022* [online]. The Arbiters' Commission. [cit. 2023-03-12]. Dostupné z: https://arbiters.fide.com/wp-content/uploads/Publications/Manual/ARBManual2022.pdf.
12. SHEPEL, Artur. *Advantages and Disadvantages of Multi-Platform App Development* [online]. [cit. 2023-03-12]. Dostupné z: https://deventor.io/blog/the-pros-and-cons-of-selecting-cross-platform-development.
13. EDDIE, James. *Flutter VS Xamarin VS React Native: Which One to Choose in 2022?* [Online]. [cit. 2023-03-12]. Dostupné z: https://blog.devgenius.io/flutter-vs-xamarin-vs-react-native-which-one-to-choose-in-2022-48368604be8.
14. *React Native* [online]. Dostupné také z: https://reactnative.dev.
15. *TypeScript* [online]. Dostupné také z: https://www.typescriptlang.org.

16. *Node.js* [online]. Dostupné také z: https://nodejs.org.
17. ARULRAJ, Sangeeth. *SQL vs NoSQL* [online]. Dostupné také z: https://medium.com/nerd-for-tech/sql-vs-nosql-faef10e3852d.
18. *Understanding the Different Types of NoSQL Databases* [online]. MongoDB. [cit. 2023-03-12]. Dostupné z: https://www.mongodb.com/scale/types-of-nosql-databases.
19. LI, Yishan; MANOHARAN, Sathiamoorthy. A performance comparison of SQL and NoSQL databases. In: *2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*. 2013, s. 15–19. Dostupné z DOI: 10.1109/PACRIM.2013.6625441.
20. *MongoDB 4.2.6* [online]. Jepsen. [cit. 2023-03-12]. Dostupné z: https://jepsen.io/analyses/mongodb-4.2.6.
21. *PostgreSQL* [online]. Dostupné také z: https://www.postgresql.org.
22. *REST vs. SOAP* [online]. Red Hat. [cit. 2023-03-12]. Dostupné z: https://www.redhat.com/en/topics/integration/whats-the-difference-between-soap-rest.
23. AGGARWAL, Anuradha. *Short Polling vs Long Polling vs Web Sockets* [online]. [cit. 2023-03-12]. Dostupné z: https://anuradha.hashnode.dev/short-polling-vs-long-polling-vs-web-sockets.
24. DOGANTEKIN, Tugce Konuklar. *Hexagonal Architecture* [online]. [cit. 2023-03-12]. Dostupné z: https://medium.com/idealo-tech-blog/hexagonal-ports-adapters-architecture-e3617bcf00a0.
25. DAMTOFT, Eric. *Onion vs Clean vs Hexagonal Architecture* [online]. Dostupné také z: https://medium.com/@edamtoft/onion-vs-clean-vs-hexagonal-architecture-9ad94a27da91.

Obsah přiloženého média