Bakalárska práca

České
vysoké
učení technické
v Praze

F4

Fakulta elektrotechnická
Katedra počítačů

Aplikácia pre hráčov stolných hier

Robert Popelka

Školiteľ: doc. Ing. Ivan Jelínek, CSc.
Odbor: Otevřená informatika
Zameranie: Software
Máj 2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Popelka** Jméno: **Robert** Osobní číslo: **507311**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Specializace: **Software**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Aplikace pro organizaci událostí pro hráče deskových her

Název bakalářské práce anglicky:

An event management app for board game players

Pokyny pro vypracování:

Navrhněte a implementujte mobilní aplikaci pro hráče deskových her. Aplikace bude fungovat na principu sociální sítě. Aplikace umožní:
vytvoření události pro hraní vybrané hry na místě definovaném gps souřadnicemi vybraném na interaktivní mapě, ve vybraném čase a s vybraným počtem volných míst,
přihlášení na vytvořenou událost zobrazenou na interaktivní mapě, kde se hráči střetnou (offline) v čase specifikovaném tvůrcem události,
filtrování zobrazení událostí v okolí na základě plánovaných her událostí zvolených tvůrcem dané události, nebo počtem volných míst dané události,
komunikaci mezi uživateli prostřednictvím chatu,
hodnocení kvality her dle určených kritérií, tzv. rating
Proveďte rešerši již existujících řešení. Specifikujte a upřesněte požadavky na aplikaci. Navrhněte vhodné implementační prostředí. Navrhněte strukturu aplikace. Aplikaci implementujte. Navrhněte způsob testování aplikace, aplikaci otestujte a test vyhodnoďte.

Seznam doporučené literatury:

[1] Antonio Leiva. Kotlin for Android Developers (20. 6. 2017)
[2] Android Developer Guides - https://developer.android.com/guide/
[3] Maps SDK for Android (Google Maps API Documentation)
https://developers.google.com/maps/documentation/android-sdk

Jméno a pracoviště vedoucí(ho) bakalářské práce:

doc. Ing. Ivan Jelínek, CSc. kabinet výuky informatiky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **07.02.2024** Termín odevzdání bakalářské práce: **24.05.2024**

Platnost zadání bakalářské práce: **21.09.2025**

doc. Ing. Ivan Jelínek, CSc. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

 © ČVUT v Praze, Design: ČVUT v Praze, VIC

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____ _____

Datum převzetí zadání Podpis studenta

 © ČVUT v Praze, Design: ČVUT v Praze, VIC

Poďakovanie

Ďakujem pánovi doc. Ing. Ivanovi Jelínkovi, CSc. za odborné vedenie, ktoré mi pomohlo túto prácu vypracovať.

Prehlásenie

Prehlasujem, že som predloženú prácu vypracoval samostatne a uviedol som všetku použitú literatúru.

V Prahe, 22. mája 2024

Abstrakt

Cieľom tejto práce je návrh a implementácia aplikácie pre hráčov stolných hier. Aplikácia sa skladá z klienskej strany vo forme mobilnej aplikácie a serverovej strany. Aplikácia dovoľuje užívateľom vytvárať udalosti v ich okolí, ktoré sú zobrazené na interaktívnej mape v mobilnej aplikácii a poskytuje vzájomú komunikáciu. Vytvorené udalosti sú odosielané na serverovú stranu a ukladané v databáze. Súčasťou tejto práce je analýza problému, rešerš existujúcich riešení a trhu, špecifikácia požiadaviek, návrh riešenia, voľba vhodných technológií na implementáciu, implementácia riešenia a testovanie spomínaného riešenia.

Kľúčové slová: mobilná aplikácia, .NET MAUI, Java, Spring Boot, stolné hry

Školiteľ: doc. Ing. Ivan Jelínek, CSc.
ČVUT FEL,
Technická 2,
16000 Praha 6

Abstract

The goal of this thesis is the design and implementation of an application for board game players. The application consists of a client side in the form of a mobile application and a server side. The application allows users to create events in their area, which are displayed on an interactive map in the mobile application, and facilitates communication between users. Created events are sent to the server side and stored in the database. Part of this work is the analysis of the problem, research of existing solutions and the market, specification of requirements, design of the solution, choice of suitable technologies for implementation, implementation of the solution and testing of said solution.

Keywords: mobile application, .NET MAUI, Java, Spring Boot, board games

Title translation: Application for boardgame players

Obsah

Úvod 1

1 Analýza problému 2
1.1 Definícia cieľovej skupiny a modelového užívateľa . . . 2
1.2 Biznisové ciele . . . 2
1.3 Analýza trhu . . . 3
1.3.1 Match n Game . . . 3
1.3.2 Boardgame Arena . . . 3
1.3.3 Tabletopia . . . 3
1.3.4 Eventbrite . . . 3
1.3.5 Meetup . . . 4
1.3.6 Board Games Companion . . . 4
1.4 Kľúčové funkcie . . . 4
1.5 Voľba platformy . . . 5
1.6 Analýza požiadaviek . . . 6
1.6.1 Biznisové požiadavky . . . 6
1.6.2 Funkčné požiadavky . . . 7
1.6.3 Nefunkčné požiadavky . . . 10
1.6.4 Systémové požiadavky . . . 10

2 Návrh riešenia 11
2.1 Biznisový doménový model . . . 11
2.2 Dizajn užívateľského rozhrania . 12
2.2.1 Wireframes . . . 12
2.2.2 Scenáre užitia . . . 14
2.3 Návrh architektúry . . . 17
2.3.1 Klientská strana . . . 17
2.3.2 Serverová strana . . . 18
2.4 Návrh API . . . 19

3 Výber technológií 21
3.1 Technológie pre klientskú stranu 21
3.1.1 Prototyp užívateľského rozhrania . . . 21
3.2 Technológie pre serverovú stranu 24
3.3 Vývojové prostredie . . . 24

4 Implementácia 25
4.1 Serverová strana . . . 25
4.1.1 Perzistentná vrstva . . . 26
4.1.2 Servisná vrstva . . . 27
4.1.3 Prezentačná vrstva . . . 28
4.1.4 Nasadenie . . . 32
4.2 Klientská strana . . . 33
4.2.1 Užívateľské rozhranie . . . 33
4.2.2 Model . . . 36

5 Testovanie 40
5.1 Automatizované testy . . . 40
5.1.1 Unit testy . . . 40
5.2 Manuálne testy . . . 41
5.3 Užívateľské testy . . . 42
5.3.1 Testy priechodu nového užívateľa . . . 42
5.3.2 Výsledky testov . . . 43
5.3.3 Ukazatele výkonnosti . . . 44
5.3.4 Zhrnutie užívateľského testovania . . . 46

Záver 47

Literatúra 49

Obrázky

1.1 Dotazník pre zistenie dopytu 5
1.2 UML diagram biznisových požiadaviek 7
1.3 UML diagram funkčných požiadaviek k biznisovému cieľu - interakcia medzi užívateľmi 8
1.4 UML diagram funkčných požiadaviek k biznisovému cieľu - vytvorenie udalosti 9

2.1 UML diagram biznisového doménového modelu 12
2.2 Wireframes prihlasovacej a hlavnej stránky 13
2.3 Wireframes profilovej stránky, četu a stránky hry 13
2.4 UML diagram komponentov klientskej strany 18
2.5 UML diagram komponentov serverovej strany 19

3.1 Snímky obrazovky PoC - prihlasovacia a hlavná stránka 23
3.2 Snímky obrazovky PoC - vytvorenie udalosti a prihlásenie sa na udalosť 23

4.1 ER diagram generovaný IntelliJ IDEA 27
4.2 Swagger UI interaktívna dokumentácia 31
4.3 Snímky obrazovky - Prihlasovacia a hlavná stránka 34
4.4 Snímky obrazovky - Profilová stránka, čet a stránka hry 34
4.5 Sekvenčný diagram - aktualizácia udalostí 39

5.1 Výsledky dotazníka - záujem o stolné hry 43
5.2 Výsledky dotazníka - hodnotenie funkcionality vytvorenia udalosti .. 44
5.3 Miera zlyhania požiadaviek na server 45
5.4 Priemerné doby odozvy požiadaviek na server 46

Tabuľky

1.1 Biznisové ciele pokryté existujúcimi riešeniami 4

Úvod

Táto práca sa zameriava na vývoj mobilnej aplikácie určenej pre jednoduchú organizáciu udalostí na hranie stolných hier.

V dnešnej dobe internetu sa môže zdať, že tradičné formy zábavy a voľnočasových aktivít sa pomaly vytrácajú a dávajú prednosť digitálnemu priestoru. Tento trend je možné pozorovať vo viacerých médiách, napríklad divadlo, film, noviny, atď [16].

Obrovský nárast popularity digitálnych hier zapadá do tohto trendu a preto by bol úpadok stolných hier očakávaným vývojom. Avšak opak môže byť pravdou. Prieskum popularity stolných hier poukazuje na kontinuálny nárast trhu so stolnými hrami z roka na rok [19].

Z hľadiska trhu stolných hier môžeme hovoriť o "zlatej ére" [19]. A vzniká dopyt nielen po stolných hrách ako takých, ale aj rôznych udalostiach a nástrojoch pre nadšencov stolných hier. Organizácia udalostí na hranie stolných hier môže byť náročná, hlavne na malej škále.

Riešenie opísané v tejto práci sa teda zameriava na vytvorenie nástroja pre organizáciu týchto udalostí a spájanie nadšencov stolných hier.

V úvodnej kapitole je popísaná analýza problému, prieskum trhu a existujúcich riešení a analýza biznisových cieľov a požiadaviek. Následne je popísaný návrh riešenia zahŕňajúci vypracovanie biznisového modelu, dizajnu a návrh architektúry. Na implementáciu vytvoreného návrhu boli zvolené vhodné technológie popísané v tretej kapitole. V ďalšej kapitole bola popísaná implementácia navrhnutého riešenia a v poslednej kapitole je popísané testovanie vypracovaného riešenia.

Kapitola 1
Analýza problému

Hlavným cieľom tohto projektu je navrhnúť nástroj pre nadšencov stolných hier ktorý im uľahčí stretávanie sa s inými hráčmi a organizáciu udalostí. Hlavným problémom je teda určenie polohy udalosti z pohľadu organizátora a následné zdieľanie polohy pre záujemcov. Rovnako dôležitým problémom je aj efektívna komunikácia medzi užívateľmi. Riešenie by teda malo adresovať tieto problémy.

1.1 Definícia cieľovej skupiny a modelového užívateľa

Ako už bolo spomenuté, modelový užívateľ je nadšenec spoločenských hier, ktorý chce stretávať iných ľudí, ktorí zdieľajú toto hobby. Cieľová skupina je relatívne široká, nakoľko zúčastňovať sa udalostí pre hranie stolných hier môže ktokoľvek bez ohľadu na vek a pohlavie. Avšak nástroje na stretávanie sa a komunikáciu medzi užívateľmi by mali byť vekovo zdola ohraničené v záujme ochrany maloletých. Väčšina hráčov stolných hier sa pohybuje vo vekovom rozmedzí 18-55, čo bude primárnou cieľovou skupinou [9].

1.2 Biznisové ciele

Na základe analýzy problému a definície cieľovej skupiny boli definované základné biznisové ciele navrhovaného riešenia.

Hlavným zámerom je vytvoriť nástroj na organizáciu stretnutí a udalostí pre hranie spoločenských hier. Užívatelia budú môcť definovať miesto na vstavanej mape, zobrazovať udalosti v okolí na základe aktuálnej polohy, stanoviť čas udalosti, pridať hry z databázy a definovať minimálny a maximálny počet hráčov.

Užívatelia budú mať svoj účet, čo pridá aplikácii sociálny element a bude slúžiť ako jednoduchá sociálna sieť. Pri vytváraní účtu bude užívateľ zadávať meno a heslo, a po úspešnom vytvorení účtu bude môcť nahrať svoj profilový obrázok. V rámci tejto siete budú môcť užívatelia vidieť udalosti vytvorené inými užívateľmi a prihlásiť sa na zvolené udalosti. Užívatelia budú takisto môcť komunikovať so spoluhráčmi zo zvolenej udalosti.

Databáza hier bude umožňovať pridanie nových hier s informáciami ako názov, počet hráčov, vekové obmedzenie a obrázok. Užívatelia budú môcť hodnotiť existujúce hry a získavať tak lepší prehľad o ich kvalite.

Sekundárnym biznisovým cieľom by mohol byť aj výmenný bazár na ktorý by mohli užívatelia pridávať inzercie hier z druhej ruky.

1.3 Analýza trhu

Pri analýze trhu bolo nájdených viacero existujúcich riešení na pôvodne zadané biznis ciele. Žiadna z týchto aplikácii však nespája všetky zadané ciele. Taktiež nebolo nájdené riešenie ktoré by sa sústredilo špecificky na inzerciu, alebo výmenu stolných hier. Avšak existuje mnoho inzerčných platform, ako je napríklad Vinted, a skupín na inzerciu na veľkých sociálnych sieťach, ako napríklad Facebook Marketplace, ktoré tvoria silnú konkurenciu a preto je implementácia inzercie hier v tomto projekte zbytočná.

Nasleduje popis existujúcich riešení a tabuľka 1.1 popisujúca pokryté biznisové ciele.

1.3.1 Match n Game

Match n Game je mobilná aplikácia, ktorá sa sústredí na trh spoločenských hier a jej primárna funkcionalita je ako sociálna sieť na stretávanie iných hráčov. Nedisponuje však žiadnou funkcionalitou na vytváranie udalostí, či databázou hier [17].

1.3.2 Boardgame Arena

Board Game Arena je multiplatformová aplikácia s prvkami sociálnej siete ktorá sa sústredí na trh stolných hier. Aplikácia umožňuje užívateľom vytvárať virtuálne miestnosti a hrať množstvo hier z databázy. Avšak je plne online a nedisponuje žiadnou funkcionalitou, ktorá by pomohla užívateľom s organizáciou offline udalostí [2].

1.3.3 Tabletopia

Tabletopia, je rovnako ako Boardgame Arena multiplatformová aplikácia s prvkami sociálnej siete ktorá umožňuje užívateľom hrať stolné hry online. Rozdiely medzi nimi sú hlavne v užívateľskom prostredí a databáze hier [25].

1.3.4 Eventbrite

Eventbrite je multiplatformová aplikácia na vytváranie udalostí rôzneho typu. Z hľadiska biznis cieľov sa asi najviac zhoduje s navrhovaným riešením. Avšak nie je cielená na trh stolných hier a nedisponuje vstavanou interaktívnou mapou [6].

1.3.5 Meetup

Meetup je rovnako ako Eventbrite multiplatformová aplikácia na vytváranie udalostí. Taktiež nie je cielená na trh stolných hier, avšak obsahuje sekciu stolné hry, ktorá má 1,798,065 užívateľov a 2,091 skupín po celom svete [10].

1.3.6 Board Games Companion

Board Games Companion je mobilná aplikácia, ktorá umožňuje užívateľovi hodnotiť stolné hry z databázy, rovnako ako zaznamenávať stav hier, alebo vytvorenie vlastného katalógu. Aplikácia však nefunguje ako sociálna sieť a neumožňuje užívateľom vytvárať udalosti [5].

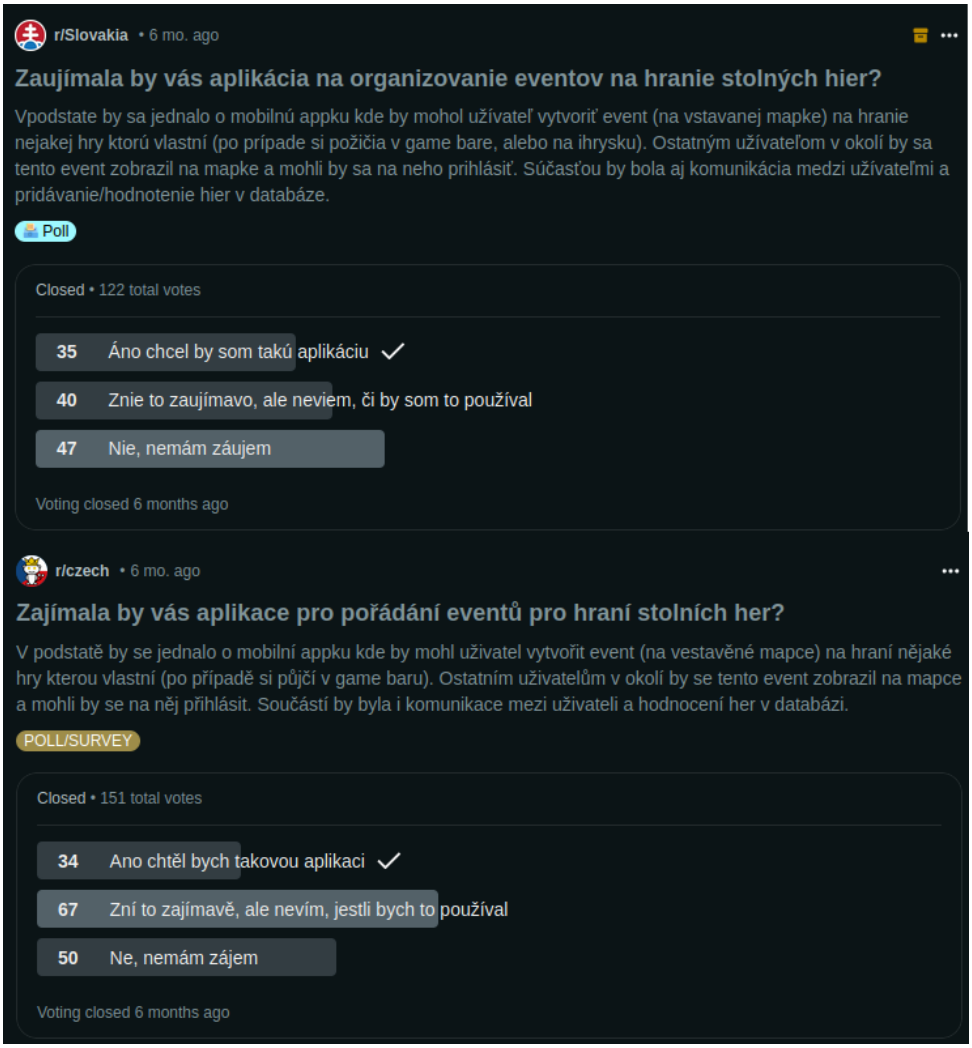
	Match n Game	Boardgame Arena	Tabletopia	Eventbrite	Meetup	Board Games Companion
Sociálna sieť	✓	✓	✓	✓	✓	X
Vytváranie udalostí	X	X	X	✓	✓	X
Interaktívna mapa	X	X	X	X	X	X
Databáza hier	X	✓	✓	X	X	✓
Webová aplikácia	X	✓	✓	✓	✓	X
Mobilná aplikácia	✓	✓	✓	✓	✓	✓

Tabuľka 1.1: Biznisové ciele pokryté existujúcimi riešeniami

1.4 Kľúčové funkcie

Interaktívna mapa na zobrazovanie a vytváranie udalostí je kľúčová funkcionalita navrhovaného riešenia. Avšak to, že žiadne z existujúcich riešení na trhu nedisponuje touto funkcionalitou by mohlo znamenať, že pre ňu nie je dopyt. Pre získanie prehľadu bol vytvorený krátky anonymný dotazník popisujúci danú funkcionalitu.

Dokopy na tento dotazník odpovedalo 268 respondentov, z toho 69 (25,7%) vyjadrilo záujem o dané riešenie, 104 (38,8%) vyjadrilo mierny záujem a (35,4%) vyjadrilo nezáujem. Samozrejme tento dotazník nie je validný štatistický údaj a nie je plne vierohodný ukazateľ záujmu na reálnom trhu. Avšak aspoň orientačne validuje dopyt pre dané riešenie.



Obrázok 1.1: Dotazník pre zistenie dopytu

1.5 Voľba platformy

Ideálne by bolo vyvinúť mobilnú aplikáciu pre Android aj IOS, keďže spolu pokrývajú vyše 99% trhu mobilných operačných systémov. Avšak MVP (minimal viable product) by mal byť navrhnutý primárne pre jednu platformu pre urýchlenie procesu. Preto je vhodnou voľbou Android, ktorý pokrýva takmer 70% celosvetového trhu [23].

Avšak pre budúce rozširovanie by bolo ideálne zvoliť na implementáciu frontendu framework, ktorý zjednoduší cross-platform vývoj aj pre IOS. Taktiež budú následne zvolené vhodné backendové a databázové technológie.

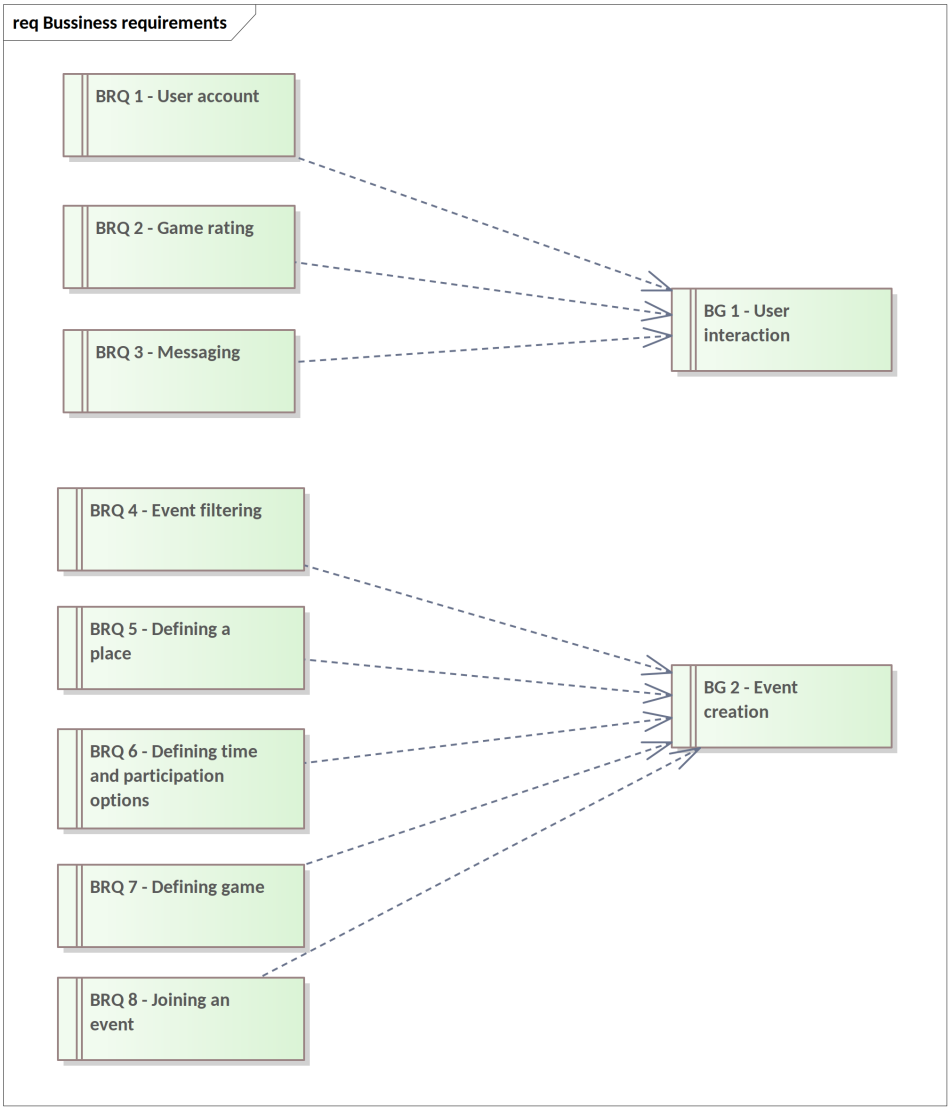
1.6 Analýza požiadaviek

Biznisové ciele a rešerš trhu a existujúcich riešení načrtli kľúčové biznis ciele a požiadavky, ktoré je potrebné dôkladne definovať na vytvorenie kvalitného návrhu riešenia. Z biznisových cieľov boli vytvorené biznisové požiadavky, ktoré špecifikujú biznisové riešenie a načrtnú funkčné a systémové požiadavky, ktoré budú špecifikovať softvérové riešenie vo fáze návrhu.

1.6.1 Biznisové požiadavky

Po rešerši a zohľadnení pôvodných biznisových cieľov a požiadaviek bol vyradený výmenný bazár, nakoľko to nie je kľúčová funkcionalita a nie je po nej veľký dopyt. Takisto bolo pridané vytvorenie účtu užívateľa ako podkategória hlavného biznisového cieľa - interakcia užívateľov (BG 1 - User interaction). Druhým hlavným biznisovým cieľom je vytváranie udalostí (BG 2 - Event creation).

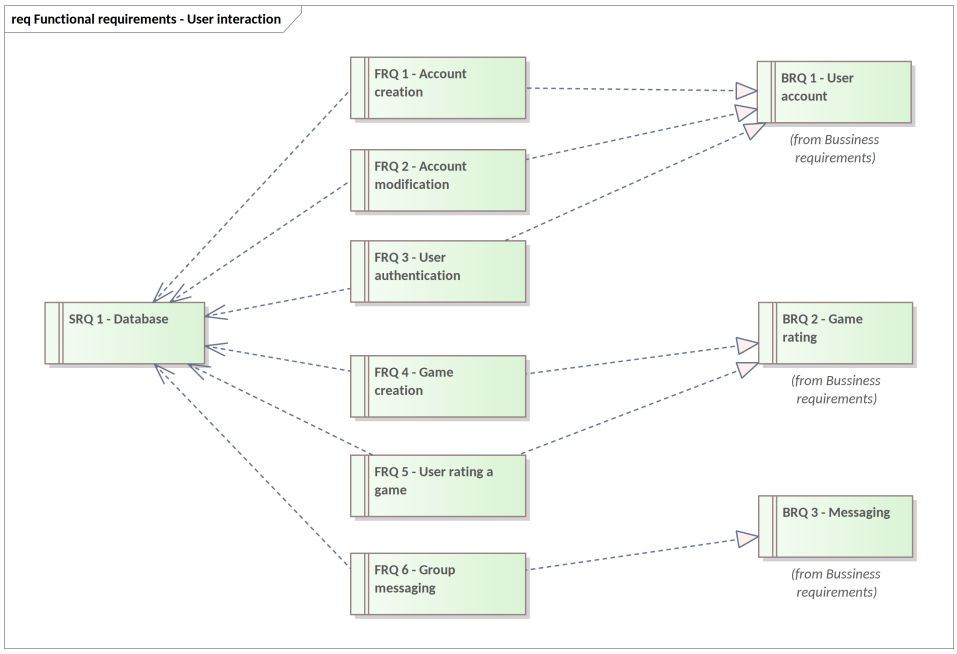
Dva hlavné biznisové ciele sa štiepia na 8 biznisových požiadaviek: vytvorenie a správa užívateľského účtu, hodnotenie hier v databáze, správy medzi užívateľmi, filtrovanie udalostí, definovanie polohy udalosti zakladateľom, definovanie času a možností udalosti (počet hráčov, vek, atď.), definovanie hry/hier na udalosť a pripojenie sa na už existujúcu udalosť.



Obrázok 1.2: UML diagram biznisových požiadaviek

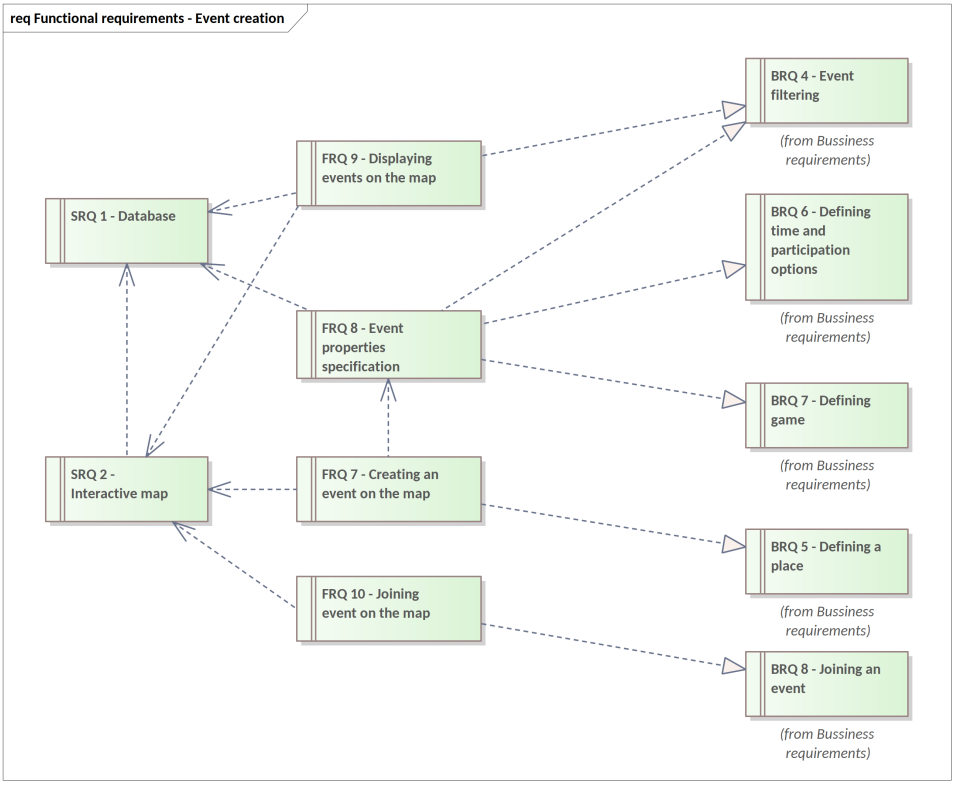
1.6.2 Funkčné požiadavky

Z biznisových požiadaviek bolo následne odvodených niekoľko funkčných požiadaviek. Funkčné požiadavky boli rozdelené na dve podkategórie podľa toho ku ktorému biznisovému cieľu patria. - 1.3 a 1.4.



Obrázok 1.3: UML diagram funkčných požiadaviek k biznisovému cieľu - interakcia medzi užívateľmi

- FRQ 1 - Vytvorenie účtu: Registrácia a zadanie osobných údajov, uloženie do databáze.
- FRQ 2 - Upravovanie účtu: Možnosti upravovania informácií v existujúcom užívateľskom účte, ako je napríklad profilový obrázok.
- FRQ 3 - Autentizácia užívateľov: Pri prihlásení je potrebné autentizovať prihlasovacie údaje užívateľa a povoliť mu vstup do aplikácie.
- FRQ 4 - Vytvorenie hry: Možnosť pridávania nových hier do databázy od užívateľov.
- FRQ 5 - Hodnotenie hier užívateľmi: Pozitívne alebo negatívne hodnotenie hier v databáze, ktoré je verejne viditeľné.
- FRQ 6 - Skupinové správy: Implementácia možnosti odosielania a prijímania skupinových správ medzi užívateľmi.



Obrázok 1.4: UML diagram funkčných požiadaviek k biznisovému cieľu - vytvorenie udalosti

- FRQ 7 - Vytvorenie udalosti na mape: Funkcionalita, ktorá umožní užívateľom vytvárať nové udalosti priamo na interaktívnej mape, čo zjednoduší lokalizáciu a organizáciu udalostí.
- FRQ 8 - Špecifikácia možností udalosti: Zahrnutie detailnej špecifikácie možností pre jednotlivé udalosti, vrátane popisu hry, počtu hráčov, trvania a iných relevantných informácií.
- FRQ 9 - Zobrazenie udalostí na mape: Funkcionalita ktorá bude zobrazovať všetky udalosti vytvorené užívateľmi na interaktívnej mape, ktorá sa bude aktualizovať v reálnom čase.
- FRQ 10 - Prihlásenie užívateľa na udalosť: Zabezpečenie možnosti, aby sa užívatelia mohli prihlasovať na existujúce udalosti, zobrazené na interaktívnej mape.

1.6.3 Nefunkčné požiadavky

Užívateľské rozhranie bude dostupné v anglickom jazyku a dizajn aplikácie bude minimalistický s hlavným sústredením na interaktívnu mapu, ktorá bude zaberať väčšinu obrazovky. Na vytváranie a prihlásenie sa na udalosti bude slúžiť kontextové menu. Na prihlásenie, zobrazenie profilu, hry, skupiny, alebo četu budú slúžiť samostatné stránky v rámci aplikácie.

1.6.4 Systémové požiadavky

Systémové požiadavky boli vytvorené na realizáciu funkčných požiadaviek. Pri návrhu aplikácie a bližšej špecifikácii sa však môžu ešte rozšíriť, či upraviť.

- SRQ 1 - Databáza: Vytvorenie a správa databázy užívateľov, hier a udalostí kde sa uchovávajú informácie o registrovaných užívateľoch, ich údaje, správy, vytvorené udalosti, jednotlivé hry a ich hodnotenie.
- SRQ 2 - Interaktívna mapa: Využitie interaktívnej mapy, ktorá umožní užívateľom preskúmať dostupné udalosti v okolí.

Kapitola 2
Návrh riešenia

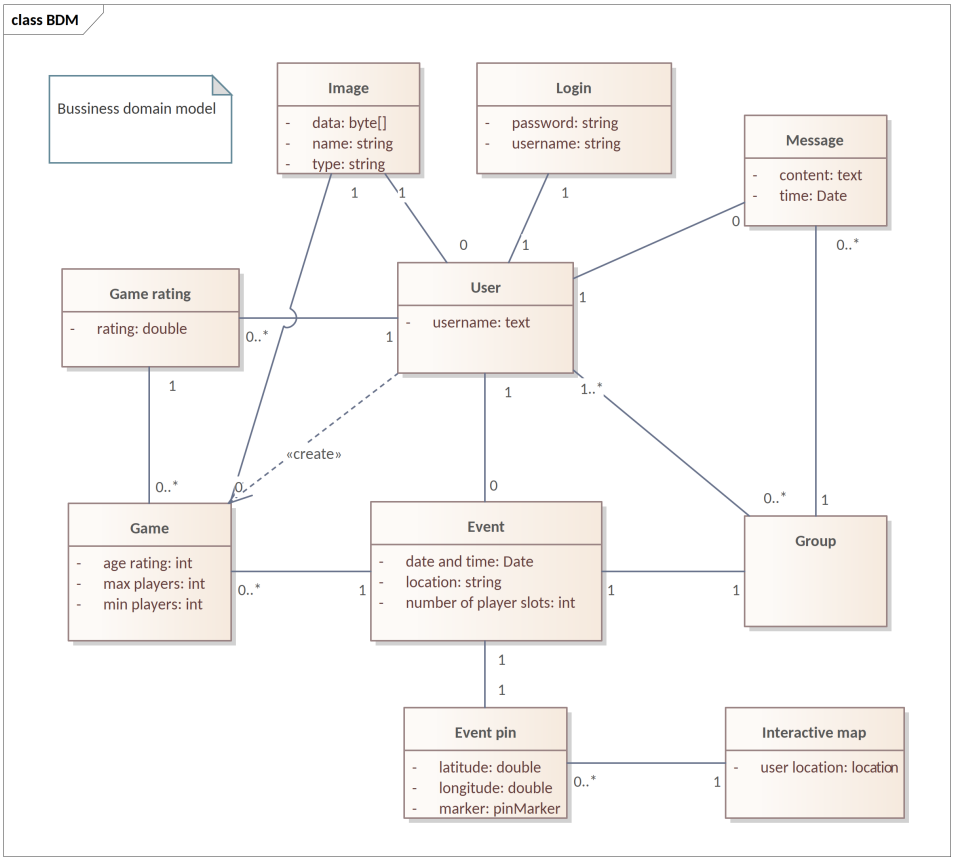
Návrh riešenia je tvorený priamo zo zadaných biznisových, funkčných, nefunkčných a systémových požiadaviek, bližšie špecifikuje softvérové riešenie a ponúka podklad pre výber technológií a následnú implementáciu riešenia.

2.1 Biznisový doménový model

Na základe biznisových požiadaviek bol navrhnutý BDM (biznisový doménový model), ktorý popisuje biznis objekty a vzťahy medzi nimi. - 2.1

Hlavné tri objekty sú User (užívateľ), Event (udalosť) a Interactive map (interaktívna mapa). Každý užívateľ má užívateľské meno, profilový obrázok a login pomocou ktorého sa registroval. Game rating predstavuje individuálne hodnotenie hry užívateľmi. Užívatelia môžu vytvoriť udalosť, alebo sa prihlásiť na udalosť. Tvorca udalosti musí špecifikovať potrebné informácie pri vytvorení udalosti.

Udalosť má dátum a čas konania, rovnako ako maximálny počet hráčov a počet prihlásených hráčov. Dôležitou súčasťou udalosti je jej poloha, ktorá je definovaná GPS súradnicami a adresou. Ku každej udalosti je automaticky vytvorený aj objekt Group ktorý uchováva informácie o skupine užívateľov, ktorí sú prihlásení na danú udalosť, vrátane skupinových správ. Rovnako je automaticky vytvorený aj objekt Event pin ktorý predstavuje špendlík zobrazený na interaktívnej mape. Interactive map predstavuje samotnú mapu na ktorej môžu užívatelia interagovať s udalosťami. Objekt Game predstavuje hry uložené v databáze, ktoré môžu byť pridávané a hodnotené užívateľmi. Image predstavuje objekt ktorý v sebe drží dáta obrázku v databáze.



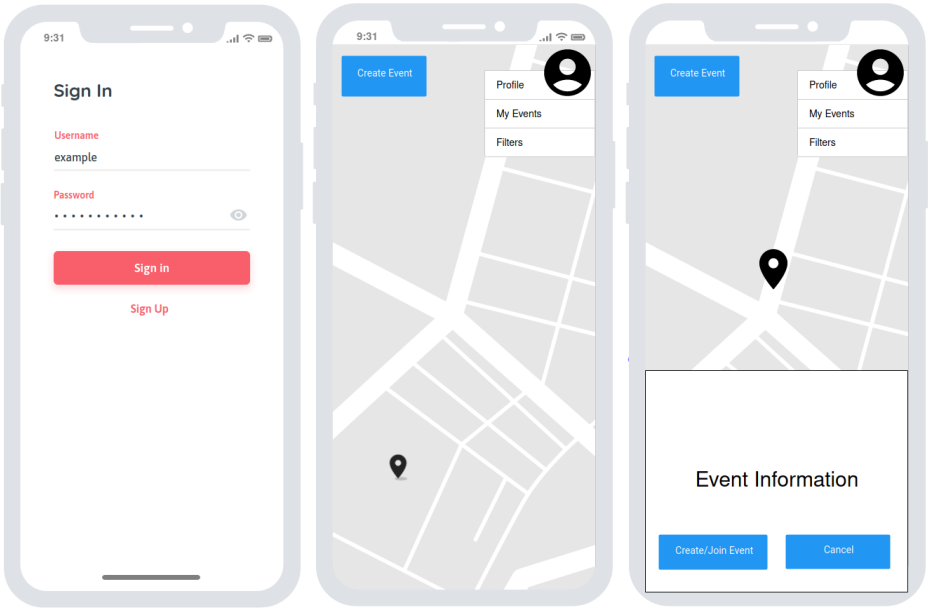
Obrázok 2.1: UML diagram biznisového doménového modelu

2.2 Dizajn užívateľského rozhrania

Kľúčovou funkciou aplikácie je interaktívna mapa, ktorá umožňuje užívateľovi vybrať udalosť na mape a prihlásiť sa na ňu. Užívateľ môže taktiež vytvoriť novú udalosť stlačením tlačidla „Create event“ v hornom ľavom rohu, následne určiť polohu klepnutím na mapu a vyplniť formulár na špecifikáciu všetkých potrebných informácii o danej udalosti.

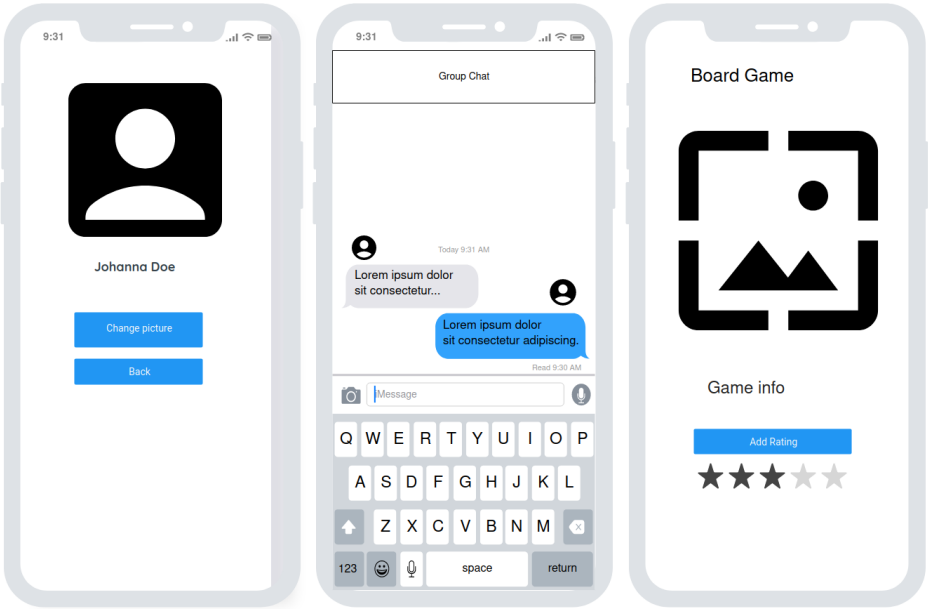
2.2.1 Wireframes

Na predstavu o dizajne frontendu slúži 6 wireframes, ktoré ilustrujú ako by malo vyzerať GUI. - 2.2 a 2.3. Prvý wireframe zobrazuje bežnú prihlasovaciu stránku a ďalšie dva hlavnú stránku aplikácie s mapou, ktorá pokrýva väčšinu obrazovky. V ľavom hornom rohu sa nachádza tlačidlo na vytvorenie novej udalosti a v pravom hornom rohu bude zobrazený profilový obrázok užívateľa, ktorý po stlačení spustí menu s možnosťami a navigačnými tlačidlami na prechod na ďalšie stránky. Po ťuknutí na špendlík označujúci udalosť na mape, alebo pri vytvorení novej udalosti sa zobrazí formulár s informáciami o udalosti a tlačidlami na vytvorenie, alebo prihlásenie na udalosť.



Obrázok 2.2: Wireframes prihlasovacej a hlavnej stránky

Ďalšie wireframes zobrazujú stránku užívateľského profilu, kde si môže užívateľ nastaviť profilový obrázok, skupinové správy, kde môžu užívatelia, ktorí sú prihlásení na konkrétnu udalosť komunikovať a stránku hry v databáze, kde sa nachádzajú informácie o danej hre, obrázok a pole na pridanie hodnotenia.



Obrázok 2.3: Wireframes profilovej stránky, četu a stránky hry

2.2.2 Scenáre užitia

Scenáre užitia (use case scenarios), sú imaginárne situácie, ktoré popisujú ako by mala prebiehať interakcia medzi užívateľom a systémom. Tieto scenáre slúžia na lepšie pochopenie funkcionality systému alebo aplikácie a pomáhajú pri návrhu a testovaní. V prípade softvérového vývoja poskytujú scenáre užitia konkrétny pohľad na to, ako by mal systém reagovať na rôzne akcie užívateľa. Sú užitočným nástrojom pri definovaní a overovaní požiadaviek a biznis procesov [18].

Prihlásenie/registrácia užívateľa

Tento scenár popisuje prihlásenie a autentizáciu užívateľa, alebo registráciu nového užívateľa.

1. Užívateľ zvolí prihlásenie, alebo registráciu prepínacím tlačidlom.
2. Systém zobrazí prihlasovaciu, alebo registračnú stránku.
3. Užívateľ vyplní meno a heslo a stlačí tlačidlo „Log in”/„Register”.
4. Systém validuje zadané údaje a potvrdí prihlásenie/registráciu (PREJSŤ NA KROK 5), alebo zobrazí chybové hlásenie (PREJSŤ NA KROK 3).
5. Systém zobrazí hlavnú stránku aplikácie.

Zmena profilového obrázku

Tento scenár popisuje proces zmeny profilového obrázku užívateľa.

1. Užívateľ stlačí tlačidlo so svojim profilovým obrázkom.
2. Systém zobrazí dropdown menu s možnosťami.
3. Užívateľ vyberie možnosť „Profile”.
4. Systém zobrazí stránku profilu užívateľa s jeho informáciami a profilovým obrázkom.
5. Užívateľ ťukne na tlačidlo „Change picture”.
6. Systém zobrazí štandardný výber obrázku zo zariadenia užívateľa.
7. Užívateľ vyberie obrázok (PREJSŤ NA KROK 8), alebo zruší výber stlačením návratového tlačidla (PREJSŤ NA KROK 4).
8. Systém uloží vybraný obrázok a aktualizuje profilový obrázok užívateľa (PREJSŤ NA KROK 4).

Vytváranie udalostí

Tento scenár popisuje kľúčovú funkcionalitu navrhovaného riešenia, ktorá poskytuje užívateľovi možnosť vytvoriť vlastnú udalosť na interaktívnej mape.

1. Užívateľ stlačí tlačidlo „Create Event“.
2. Systém zmení farbu tlačidla na indikáciu stlačenia a zobrazí správu, ktorá napovie užívateľovi ako umiestniť špendlík.
3. Užívateľ klepne na mapu, aby určil umiestnenie.
4. Systém umiestni špendlík na špecifikovanú polohu a otvorí okno vytvárania udalosti.
5. Užívateľ vyplní údaje o udalosti a stlačí tlačidlo „Create“ (PREJSŤ NA KROK 7) alebo tlačidlo „Cancel“ (PREJSŤ NA KROK 6).
6. Systém zatvorí menu vytvárania udalosti a odstráni špendlík, ktorý bol predtým vytvorený (PREJSŤ NA KROK 1).
7. Systém zatvorí menu vytvárania udalosti a pridá udalosť medzi udalosti užívateľa.

Pripojenie sa k udalosti

Tento scenár popisuje proces pri ktorom sa užívateľ pripojí na už existujúcu udalosť na mape.

1. Užívateľ stlačí špendlík na mape.
2. Systém vycentruje mapu na špendlík a otvorí okno s informáciami o udalosti.
3. Užívateľ stlačí tlačidlo „Join Event“ v okne s informáciami (PREJSŤ NA KROK 5) alebo klepne na mapu (PREJSŤ NA KROK 4).
4. Systém zatvorí okno s informáciami o udalosti (PREJSŤ NA KROK 1).
5. Systém zmení tlačidlo „Join Event“ na tlačidlo „Quit Event“, zvýši počet účastníkov udalosti a pridá udalosť medzi udalosti užívateľa.

Komunikácia v skupinovom čete udalosti

Tento scenár popisuje proces odoslania správy do skupinového četu udalosti na ktorú je užívateľ prihlásený.

1. Užívateľ stlačí tlačidlo so svojim profilovým obrázkom.
2. Systém zobrazí dropdown menu s možnosťami.
3. Užívateľ vyberie možnosť „My Events”.

4. Systém zobrazí stránku ktorá zobrazuje zoznam udalostí na ktoré je užívateľ prihlásený.
5. Užívateľ stlačí udalosť ktorú chce otvoriť.
6. Systém zobrazí okno s informáciami o udalosti a tlačidlami na zobrazenie hry, špendlíka, alebo četu danej udalosti.
7. Užívateľ stlačí tlačidlo „Group chat".
8. Systém zobrazí stránku so skupinovým četom príslušným danej udalosti.
9. Užívateľ vyplní textové pole správy a stlačí tlačidlo odoslania.
10. Systém uloží správu a aktualizuje zobrazené správy.

Hodnotenie hry

Tento scenár popisuje proces ohodnotenia hry užívateľom.

1. Kroky 1 - 6 zo scenáru *Komunikácia v skupinovom čete udalosti*.
2. Užívateľ stlačí tlačidlo s názvom hry danej udalosti.
3. Systém zobrazí stránku s informáciami o danej hre a tlačidlom na pridanie hodnotenia.
4. Užívateľ stlačí tlačidlo „Add Rating" na pridanie hodnotenia.
5. Systém zobrazí element na pridanie hodnotenia 1-5.
6. Užívateľ stlačí tlačidlo „Add Rating" (PREJSŤ NA KROK 3), alebo zvolí jednu z možností na pridanie hodnotenia (PREJSŤ NA KROK 7).
7. Systém aktualizuje hodnotenie danej hry.

Odhlásenie sa z udalosti

Tento scenár popisuje odhlásenie užívateľa z jednej z jeho udalostí.

1. Užívateľ stlačí špendlík na mape.
2. Systém zobrazí informácie o zvolenej udalosti a tlačidlo „Quit Event".
3. Užívateľ stlačí tlačidlo „Quit Event".
4. Systém odhlási užívateľa z danej udalosti, zníži počet prihlásených hráčov a zmení tlačidlo „Quit Event" na „Join Event". Ak je na danej udalosti 0 hráčov, udalosť sa vymaže.

Filtrovanie zobrazených udalostí

Tento scenár popisuje proces nastavenia filtrov na zobrazenie udalostí na interaktívnej mape.

1. Užívateľ stlačí tlačidlo so svojim profilovým obrázkom.
2. Systém zobrazí dropdown menu s možnosťami.
3. Užívateľ vyberie možnosť „Filters".
4. Systém zobrazí okno s možnosťami na filtrovanie udalostí podľa mena vlastníka, hry a voľných miest.
5. Užívateľ vyplní možnosti filtrovania a stlačí tlačidlo „Apply Filters". Akékoľvek prázdne polia sú automaticky nastavené ako bez filtra.
6. Systém aktualizuje udalosti zobrazené na mape podľa nastavených filtrov.
7. Užívateľ stlačí tlačidlo „Refresh Filters".
8. Systém aktualizuje udalosti zobrazené na mape bez filtrov.

2.3 Návrh architektúry

Keďže ide o mobilnú aplikáciu s komunikáciou cez internet a databázou, je vhodné trojvrstvové rozdelenie aplikácie: Frontend, backend a databáza. Na serverovej strane (backend) aplikácie bude prebiehať biznis logika a spravovanie dát v databáze cez CRUD (create read update delete) operácie. Klientská strana aplikácie (frontend) bude komunikovať so serverovou stranou cez HTTP požiadavky.

2.3.1 Klientská strana

Ako návrhový vzor klientskej strany aplikácie bol zvolený MVVM (Model-View-ViewModel). Tento vzor pomáha štrukturovať aplikáciu a rozdeliť ju do troch hlavných komponentov: Model, View (Pohľad) a ViewModel. Hlavnou myšlienkou MVVM vzoru, je zachovať výhody MVC (Model-View-Controller) vzoru, ktorý rozdeľuje logiku aplikácie od zobrazenia a zároveň získať výhody ktoré ponúka data binding (viazanie dát). Data binding dovoľuje zobrazovaciemu kódu (View) pracovať priamo s dátovými objektami z modelu bez toho aby sa staral o biznis logiku [27].

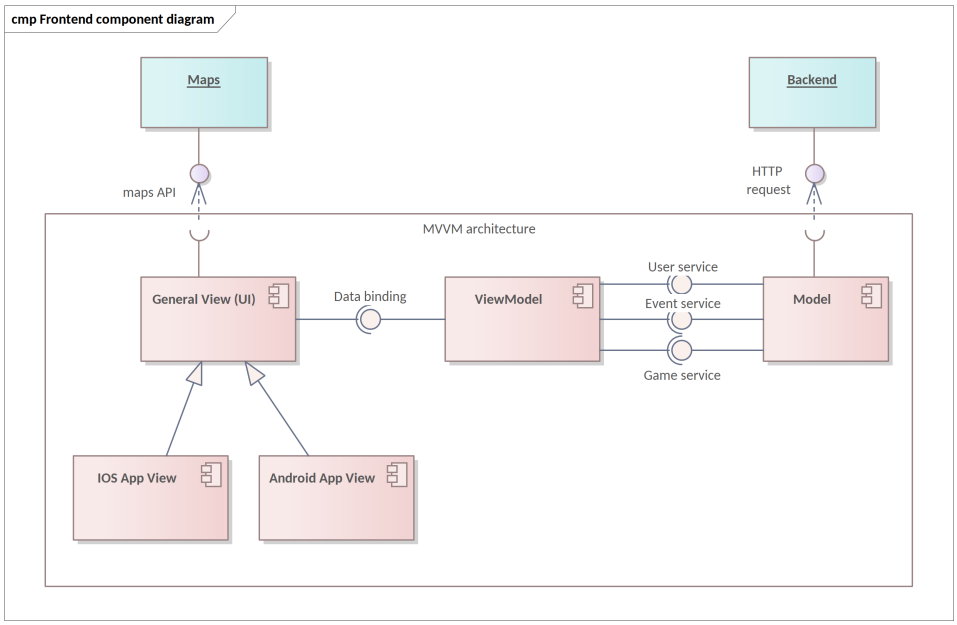
Model reprezentuje dátové objekty aplikácie. Zodpovedá za získavanie dát a komunikáciu so serverovou stranou aplikácie. Je nezávislý od používateľského rozhrania a zobrazovacieho kódu. Predstavuje teda adaptér na backend ktorý pracuje s databázou. Komunikácia medzi modelom a backendom bude prebiehať cez HTTP požiadavky.

View predstavuje užívateľské rozhranie a spôsob, akým sú dáta prezentované užívateľovi. Neobsahuje logiku aplikácie. V našom prípade bude obsahovať

všetky stránky aplikácie ktoré budú zobrazené užívateľovi. Takisto bude používať API na mapu ktorá bude zobrazená na hlavnej stránke aplikácie a bude viazaná na dáta vo viewmodele.

ViewModel spravuje užívateľský vstup a reaguje na udalosti vyvolané užívateľom. Aktualizuje model na základe akcií vykonaných v pohľade. Chová sa ako konvertor medzi modelom a pohľadom. Interakcia s modelom bude prebiehať cez servisy, ktoré budú posielať HTTP požiadavky na backend.

Vzor MVVM pomáha oddeliť logiku aplikácie od jej prezentácie, čo uľahčuje údržbu, testovanie a rozšírenie softvéru. Model zabezpečuje konzistentnosť dát, pohľad zodpovedá za vizuálnu reprezentáciu a interakciu s užívateľom a viewmodel spravuje logiku za interakciou s užívateľom.



Obrázok 2.4: UML diagram komponentov klientskej strany

2.3.2 Serverová strana

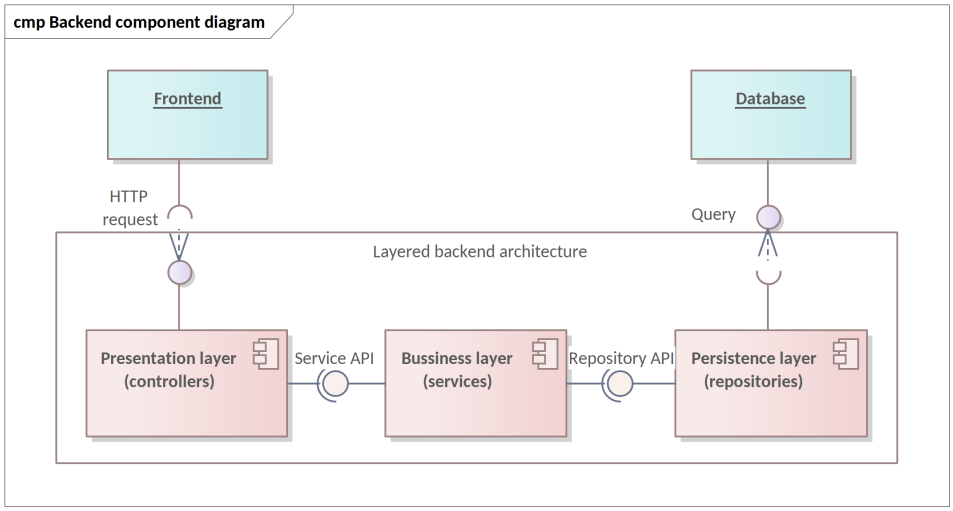
Na návrh serverovej strany aplikácie bola použitá viacvrstvová architektúra bežne používaná v enterprise aplikáciách. Výhodami viacvrstvovej architektúry sú prepoužiteľnosť a flexibilita, nakoľko ak chceme zmeniť, alebo pridať funkcionalitu do už existujúcej viacvrstvovej aplikácie, nie je potrebné prerobiť celú aplikáciu, ale len upraviť, alebo pridať jednotlivé vrstvy [4].

Serverová strana aplikácie je rozdelená do troch vrstiev: prezentačná vrstva, biznisová vrstva a perzistentná vrstva.

Prezentačná vrstva pozostáva z kontrolerov, ktoré budú zodpovedné za prijímanie a odpovedanie na HTTP požiadavky od klientskej strany aplikácie. Na tejto vrstve bude prebiehať autorizácia klientov.

Biznisová vrstva pozostáva zo servisov, ktoré budú zodpovedné za biznis logiku aplikácie. Tu budú prebiehať všetky výpočty, validácie a biznisové pravidlá.

Perzistentná vrstva sa skladá z repositories, ktoré budú zodpovedné za CRUD operácie v databáze a biznisových objektov, ktoré budú mapované na entity v databáze.



Obrázok 2.5: UML diagram komponentov serverovej strany

2.4 Návrh API

Na komunikáciu medzi klientskou a serverovou stranou je potrebné definovať vhodné rozhranie API. Keďže komunikácia bude prebiehať cez internet, môžeme hovoriť o web API. Dobre navrhnuté webové API by malo byť platformovo nezávislé a nezávisle rozšíriteľné, teda by nemalo záležať na platforme klienta a implementácia nových funkcionalít by nemala závisieť na implementácii klienta.

Najznámejší architektonický prístup na návrh webových rozhraní pre klient-server architektúry je REST (Representational State Transfer). REST je architektonický štýl pre budovanie distribuovaných systémov, nezávislý od konkrétneho komunikačného protokolu. Väčšina bežných implementácií REST API však používa HTTP ako aplikačný protokol. Výhodou REST je, že používa otvorené štandardy a neviaže implementáciu API na špecifickú implementáciu klienta. Napríklad REST webové rozhranie môže byť napísané v ASP.NET, alebo v Jave a klientske aplikácie môžu používať akýkoľvek jazyk alebo sadu nástrojov, ktoré dokážu generovať požiadavky a spracovávať odpovede HTTP.

RESTful API rozhrania sú navrhnuté okolo zdrojov, ktoré môžu byť akýkoľvek objekt, dáta, alebo služba. Jednotlivé zdroje majú unikátne identifikátory URI, ktoré klienti používajú na prístup k danému zdroju. Pre prácu s danými zdrojmi definuje REST štyri základné CRUD operácie . V prípade RESTful API s použitím HTTP protokolu sa jedná o operácie GET (read), POST (create), PUT (update/create), PATCH (update) a DELETE (delete) [15].

Ako bolo spomenuté na komunikáciu medzi klientom a serverom bude slúžiť HTTP protokol. Na zahájenie komunikácie pošle klient HTTP požia-

davku so špecifikovanou HTTP metódou, URI a môže obsahovať aj hlavičky a telo. Server potom môže odpovedať HTTP odpoveďou so stavovým kódom, hlavičkami a telom odpovede s reprezentáciou daného zdroja v mediálnom formáte špecifikovanom v hlavičke. Napríklad na prístup k zdroju `/user` na serveri `example.cz` a získanie užívateľa s identifikátorom 1 pošle klient nasledujúcu HTTP GET požiadavku: `http://example.cz/user/1`. Po spracovaní a schválení požiadavky môže odpoveď od serveru vyzerať nasledovne:

```
HTTP/1.1 200 OK
Content-Type: application/json
{
    "id": 1,
    "name": "Robert Popelka"
}
```

`HTTP/1.1` značí typ protokolu, `200 OK` je stavový kód značiaci úspešné spracovanie požiadavky a `Content-Type: application/json` je hlavička ktorá špecifikuje mediálny typ tela odpovedi. Nasleduje telo odpovedi vo formáte JSON, ktoré reprezentuje požadovaný zdroj. Stavové kódy sa delia na 5 základných typov [26]:

- 1xx informačná odpoveď – požiadavka bola prijatá, proces pokračuje
- 2xx úspech – požiadavka bola úspešne prijatá, pochopená a zodpovedaná
- 3xx presmerovanie – na dokončenie požiadavky je potrebné vykonať ďalšie kroky
- 4xx chyba klienta – požiadavka obsahuje zlú syntax alebo ju nemožno splniť
- 5xx chyba servera – server nesplnil zjavne platnú požiadavku

Prezentačná vrstva serverovej strany aplikácie bude implementovať RESTful API na komunikáciu s klientom pomocou HTTP požiadaviek. Na mapovanie požiadaviek na konkrétne zdroje budú slúžiť kontrolery, ktoré budú definovať koncové body daných požiadaviek.

Kapitola 3
Výber technológií

Technológií na vývoj mobilných aplikácií je mnoho a výber často nie je jasný. Bolo potrebné vybrať technológie pre užívateľské rozhranie, serverovú stranu s databázou a taktiež vhodné vývojové prostredie pre zvolené technológie. V prvom rade bol zvolený framework pre užívateľské rozhranie a následne boli zvolené vhodné backendové technológie.

3.1 Technológie pre klientskú stranu

Pri výbere vhodného frameworku na cross-platform vývoj mobilných aplikácií bolo zohľadnených viacero faktorov. Jedným z hlavných faktorov je plná podpora pre Android, nakoľko MVP tohto projektu bude implementovaný pre Android. Menej dôležitá je cross-platform podpora pre iOS.

Popularita frameworku je ďalším dôležitým kritériom, keďže populárne frameworky často disponujú širšou komunitou a podporou. Taktiež dôležitým faktorom sú aj osobné skúsenosti vývojára s daným jazykom.

Medzi najpopulárnejšie frameworky pre cross-platform vývoj mobilných aplikácií patria Flutter s programovacím jazykom Dart, React Native s použitím JavaScriptu, Kotlin Multiplatform Mobile s Kotlinom, Ionic využívajúci JavaScript, a .NET Xamarin, ktorého evolúciou je MAUI (Multi-plaform App UI), s programovacím jazykom C# a XAML [8]. Vzhľadom na absenciu skúseností s jazykmi JavaScript, TypeScript a Dart, bol zvolený .NET MAUI, alebo KMM. Na rozhodnutie medzi týmito dvoma frameworkami bolo potrebné ich hlbšie zhodnotiť.

3.1.1 Prototyp užívateľského rozhrania

Na hlbšiu evaluáciu MAUI bola vytvorená proof of concept aplikácia ktorá modeluje základné užívateľské rozhranie tohto projektu. Proof of concept je realizácia určitej myšlienky s cieľom preukázať jej hodnotu alebo demonštrácia princípu a overiť, že má praktický potenciál. Realizácia je zvyčajne malá a môže alebo nemusí byť úplná. V kontexte výberu frameworku sa dá hovoriť aj o proof of technology [28].

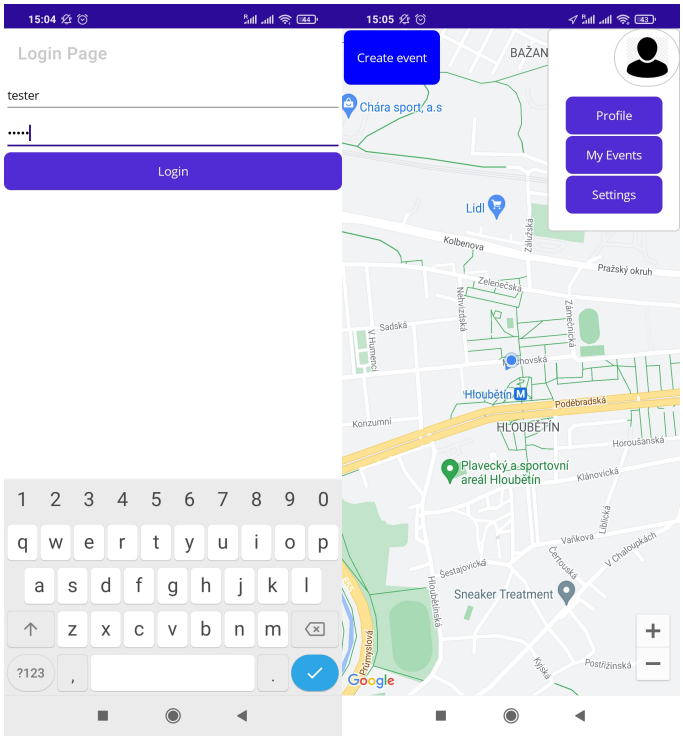
Ako proof of concept bola vytvorená jednoduchá Android aplikácia v MAUI. Použité vývojové prostredie bolo Visual Studio a mobilný telefón

s Androidom na testovanie počas vývoja. Táto aplikácia pokrýva hlavný biznisový cieľ vytvorenia udalostí 1.2 na užívateľskej vrstve a implementáciu základných scenárov užitia vytvorenia udalosti a prihlásenia užívateľa na udalosť 2.2.2.

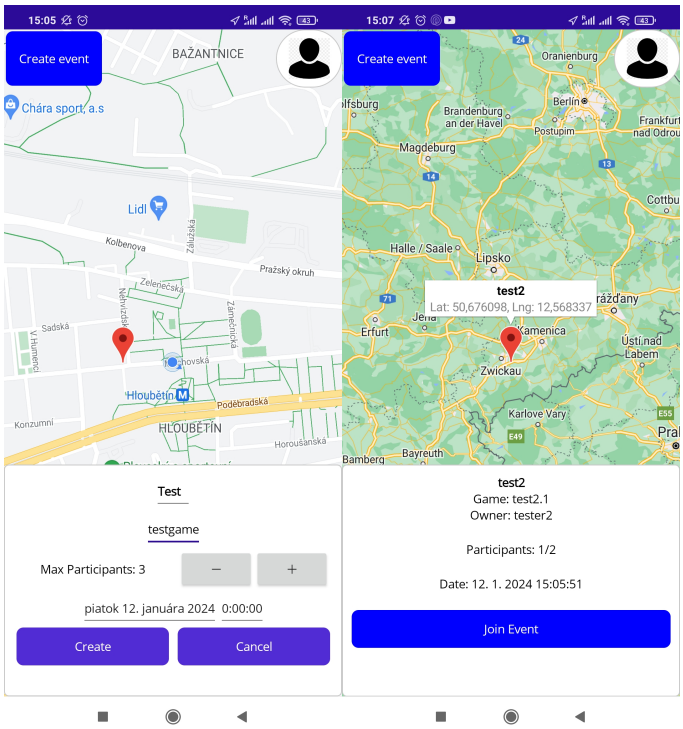
Implementácia obsahuje dve stránky - prihlasovaciu a hlavnú stránku s interaktívnou mapou - obrázok 3.1. Na implementáciu interaktívnej mapy bol použitý balík Microsoft.Maui.Controls.Maps, ktorý obsahuje rozhranie na použitie natívnych máp pre konkrétnu platformu. V prípade Androidu sú natívne mapy Google Maps a bolo potrebné prepojiť aplikáciu s Google Maps API [13]. V rámci tohto balíčku bol použitý objekt Pin, ktorý reprezentuje vytvorenú udalosť, ktorá je zobrazená na mape. Na mape je možné vytvoriť udalosť, alebo sa pridať k už existujúcej udalosti - 3.2.

Oficiálna dokumentácia k MAUI je celkom prehľadná a obšírna [14]. Práca s MAUI bola zo začiatku pomerne náročná, ale vďaka kvalitnej dokumentácii išla náročnosť s postupom času prudko dole. MAUI taktiež obsahuje plnú podporu pre implementáciu MVVM návrhového vzoru a je teda sľubnou voľbou pre frontend.

Rovnako bola začatá aj implementácia proof of concept v KMM frameworku, no už pri hľadaní knižníc na prácu z Google Maps API, alebo Bing Maps API nastali viaceré komplikácie a od ďalšej implementácie bolo upustené.



Obrázok 3.1: Snímky obrazovky PoC - prihlasovacia a hlavná stránka



Obrázok 3.2: Snímky obrazovky PoC - vytvorenie udalosti a prihlásenie sa na udalosť

3.2 Technológie pre serverovú stranu

Pri výbere backendového riešenia bolo zohľadnených viacero kritérií. Jedným z týchto kritérií je popularita, pretože populárne technológie často disponujú rozsiahlou komunitou a podporou.

Pri výbere backendového riešenia bolo zvážených viacero možností ako ASP.NET Core a C# od microsoftu ktorý ponúka prehľadnú dokumentáciu a návody na implementáciu full-stack aplikácie s použitím MAUI a ASP.NET, alebo Ktor, ktorý zas ponúka možnosť implementácie full-stack aplikácie s použitím KMM. Ktor a Kotlin sú technológie navrhnuté priamo od tvorcov KMM [1]. Ďalšími dobrými možnosťami by mohli byť Spring Boot s Javou, alebo Node.js.

Keďže návrh špecifikuje komunikáciu s frontendom pomocou HTTP požiadaviek, veľmi nezáleží na kompatibilite medzi frontendovým frameworkom a backendom, nakoľko všetky vymenované technológie ponúkajú možnosť komunikácie cez REST API volania. Preto bol zvolený na implementáciu backendu Spring Boot, s ktorým mám najlepšie osobné skúsenosti.

Databázová technológia bola vyberaná z najpopulárnejších technológii: PostgreSQL, MySQL, Microsoft SQL Server, SQLite [20]. Spring Boot dokáže pracovať s ktoroukoľvek z týchto technológií, takže ako databázová technológia bol zvolený Microsoft SQL Server, kvôli jednoduchosti konfigurácie a nasadenia.

3.3 Vývojové prostredie

Výber vývojového prostredia je pomerne jasný. Pre Javu bola zvolená IntelliJ IDEA, pretože ponúka obrovské množstvo funkcionalít vrátane menežovania databáze a plnú podporu pre Spring Boot.

Pre .NET MAUI je vhodným vývojovým prostredím Visual Studio, nakoľko obe technológie boli vytvorené firmou Microsoft a Visual Studio poskytuje plnú podporu pre MAUI [14].

Ďalším užitočným nástrojom pre vývoj je Insomnia. Insomnia je nástroj na testovanie a vývoj REST API, ktorý umožňuje vývojárom testovať a ladiť HTTP požiadavky. Jeho hlavným účelom je poskytovať užívateľsky prívetivé rozhranie na vytváranie, odosielanie a sledovanie REST API požiadaviek a odpovedí [7].

Kapitola 4
Implementácia

V tejto časti sa pozrieme na implementáciu navrhovaného riešenia. Kompletná implementácia nebude preberaná dopodrobna, nakoľko veľká časť implementácie je bežná a nie veľmi zaujímavá. Bude vysvetlená architektúra projektu, použité technológie a niektoré konkrétne implementačné riešenia.

4.1 Serverová strana

Ako bolo spomenuté, serverová strana aplikácie bola implementovaná v jazyku Java s použitím Spring Boot frameworku. Cieľom Spring Bootu je minimalizovať potrebnú konfiguráciu a zjednodušiť tým vývoj aplikácií v Jave.

Spring ponúka viaceré závislosti na implementáciu serverovej strany aplikácie, ako Spring Web, Spring Data JPA (Java persistence API) a Spring Security. Spring Web ponúka nástroje potrebné na spracovávanie HTTP požiadaviek a implementáciu RESTful API. Spring Data JPA ponúka prístup k dátam z SQL dabáze cez objektovo relačné mapovanie a rozhranie JpaRepository, ktoré automaticky generuje SQL dotazy. Spring Security ponúka nástroje na autentizáciu, autorizáciu, šifrovanie hesiel a mnoho ďalších [22].

Na zostavovanie projektu bol použitý nástroj Maven a všetky závislosti boli definované v pom.xml súbore. Okrem vyššie spomenutých Spring závislostí boli použité ešte Lombok, na automatické generovanie get a set metód, Mapstruct na mapovanie Java Beans, Springfox Swagger na automatické vytvorenie dokumentácie REST API, Microsoft JDBC Driver na pripojenie k Microsoft Server SQL databáze a JUnit na testovanie.

Počas vývoja bola na testovanie používaná lokálna H2 databáza, pre ktorú má Spring Boot vstavanú podporu. Pri nasadení aplikácie bol použitý Microsoft SQL server nasadený na cloudovej platforme Microsoft Azure. Samotné relačné tabuľky nebolo potrebné manuálne vytvárať, nakoľko Spring Data JPA vytvorí potrebné tabuľky automaticky pri štarte aplikácie z modelu definovaného v perzistentnej vrstve aplikácie.

Zdrojový kód serverovej strany aplikácie je rozdelený na balíky podľa jednotlivých vrstiev aplikácie 2.5. Okrem definovaných vrstiev z návrhu obsahuje aj pomocný balík util, konfiguračný balík config a hlavnú triedu StartApplication ktorá je konfiguračnou triedou celej Spring Boot aplikácie.

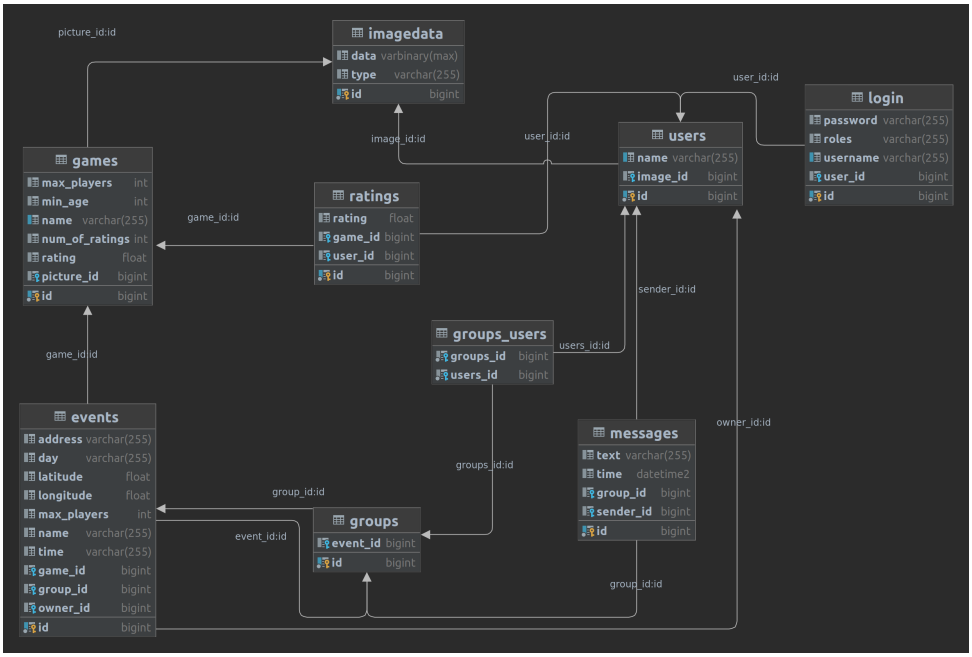
4.1.1 Perzistentná vrstva

Prvým krokom pri implementácii serverovej strany aplikácie bola perzistentná vrstva, nakoľko je zodpovedná za správu databázy a objektovo relačné mapovanie ORM. V zdrojovom kóde je rozdelená na dva balíky: entity a repository, pričom balík entity obsahuje model biznisových objektov a repository obsahuje rozšírenia rozhrania JpaRepository. Model bol implementovaný podľa návrhu BDM 2.1 a všetky biznisové objekty, ktorých dáta je potrebné uchovávať v databáze boli implementované ako JPA `@Entity` a reprezentujú tabuľky v databáze. Príklad triedy User, ktorá predstavuje užívateľa:

```
@Entity
@Table(name = "users")
@Getter
@Setter
public class User extends AbstractEntity {
    @Column(unique = true)
    private String name;
    @OneToOne
    @JoinColumn(name = "image_id")
    private ImageData profilePicture;
    @ManyToMany(mappedBy = "groupUsers", fetch = FetchType.EAGER)
    private List<Group> groups;

    public User(String name) {
        this.name = name;
        groups = new ArrayList<>();
    }

    public User() {
    }
}
```

Anotácia `@Table(name = "users")` definuje názov tabuľky ktorá bude vygenerovaná pre ukladanie dát o užívateľoch. `@Column(unique = true)` definuje nasledujúci atribút ako stĺpec v tabuľke a nastavuje naň obmedzenie ktoré garantuje, že sa hodnoty v danom stĺpci nebudú opakovať. `@OneToOne` definuje reláciu medzi dvoma entitami a `@JoinColumn(name = "image_id")` definuje názov stĺpca ktorý obsahuje id pripojeného objektu. `@ManyToMany` definuje reláciu medzi dvoma entitami, kde v tomto prípade každá entita User môže byť prepojená s viacerými entitami Group a naopak. Kvôli tomu JPA automaticky vytvorí novú tabuľku, ktorá bude obsahovať všetky kombinácie spojení medzi entitami Group a User.

Spring Data JPA pri spustení aplikácie zo všetkých entít a vzťahov medzi nimi vygeneruje relačnú databázu na pripojenom databázovom serveri. Vzniknutý ER model (entity-relational model) je zobrazený na nasledujúcom obrázku.



Obrázok 4.1: ER diagram generovaný IntelliJ IDEA

Repository rozhrania, ktoré rozširujú rozhranie JpaRepository sú veľmi jednoduché a obsahujú len deklarácie metód na posielanie SQL dotazov do databázy. Samotná implementácia repository je automaticky generovaná Springom a SQL dotazy sú vytvorené z názvov deklarovaných metód. Napríklad metóda:

```
Optional<User> findByName(String name);
```

vygeneruje nasledujúci SQL dotaz:

```
select u from User u where u.name = ?1
```

Základné CRUD operácie cez primárne kľúče entít, ako napríklad findById sú automaticky generované bez potreby deklarácie. Je možné deklarovať aj vlastné zložitejšie SQL dotazy s použitím anotácie `@Query`, ale v prípade tohto projektu to nebolo nutné [21].

4.1.2 Servisná vrstva

Ako už bolo spomenuté, na servisnej vrstve prebieha väčšina biznisovej logiky aplikácie a v zdrojovom kóde sa nachádza v balíčku service. Skladá sa z tried service pre jednotlivé entity, ktoré ponúkajú metódy kombinujúce viacero CRUD operácií a rovnako zachytávajú potenciálne nulové hodnoty a vyhadzujú vlastné výnimky. Jednotlivé triedy service sú označené Spring anotáciou `@Service`, ktorá deklaruje, že daná trieda je komponent danej Spring aplikácie a vytvorí dependency injection kdekoľvek je daná trieda deklarovaná s anotáciou `@Autowired`. Takisto zaručuje, že daná trieda sa

bude za behu aplikácie chovať ako singleton, teda bude existovať len jedna inštancia danej triedy. Príklad metódy z triedy EventServiceImpl:

```
public boolean joinEvent(Long eventId, Long userId) {
    if (eventId == null) {
     throw new FieldMissingException("Event ID is missing.");
    }
    if (userId == null) {
     throw new FieldMissingException("User ID is missing.");
    }
    Event event = eventRepository.findById(eventId)
        .orElseThrow(() -> new NotFoundException("Event not
        found with ID:" + eventId));
    User user = userRepository.findById(userId)
        .orElseThrow(() -> new NotFoundException("User not
        found with ID: " + userId));
    Group group = event.getGroup();
    group.getGroupUsers().add(user);
    groupRepository.save(group);
    eventRepository.save(event);
    userRepository.save(user);
    return true;
}
```

V príklade je implementácia metódy joinEvent, ktorá vykoná 5 CRUD operácií: vyhľadá entity User a Event z databázy podľa ich primárnych kľúčov a po tom ako pridá užívateľa do skupiny danej udalosti uloží všetky tri upravené entity. Ak entity so zadanými primárnymi kľúčami nie sú v danej databáze, metóda vyhodí výnimku `NotFoundException`.

4.1.3 Prezentačná vrstva

V prípade serverovej strany aplikácie sú užívateľským rozhraním HTTP požiadavky a odpovede od serveru. V prezentačnej vrstve sú teda kľúčovými komponentmi kontrolery, ktoré definujú REST API koncové body, na ktoré môžu klienti posielať požiadavky.

Kontrolery

Jednotlivé kontrolery sa nachádzajú v balíku controller a sú označené Spring anotáciou `@RestController`. Jednotlivé koncové body a ich funkcionalita sú definované anotáciami pre jednotlivé REST operácie. Príklad metódy getGameByName v triede GameController:

```
@RestController
@RequestMapping("/game")
public class GameController {
private final GameService gameService;
private final ImageDataService imageDataService;
private final DTOMapper dtoMapper;

@Autowired
public GameController(GameService gameService, ImageDataService
            imageDataService, DTOMapper dtoMapper) {
     this.gameService = gameService;
     this.imageDataService = imageDataService;
     this.dtoMapper = dtoMapper;
}

@GetMapping("/{name}")
public ResponseEntity<GameDTO> getGameByName(@PathVariable
        String name) {
     return ResponseEntity.ok(dtoMapper.gameToDto(gameService
            .findByName(name)));
}
```

Anotácia `@RequestMapping("/game")` definuje URI na prístup k zdrojom daného kontrolera. V prípade, že klient chce zavolať metódu getGameByName so vstupným argumentom MenoHry, teda chce dostať reprezentáciu daného dátového objektu, pošle HTTP požiadavku na URI:
`http://example.server/game/MenoHry`.

Metóda getGameByName následne zavolá metódu findByName zo servisnej vrstvy a vrátený dátový objekt mapuje na jeho DTO. DTO, alebo data transfer object je reprezentácia entity z modelu 4.1, ktorá je odosielaná cez HTTP protokol.

DTO sa používa, aby boli cez HTTP protokol odosielané len potrebné dáta z entít a zamedzilo sa zacykleniu. Zacyklenie by mohlo vzniknúť napríklad pri entitách User a Group. Keďže User a Group majú medzi sebou ManyToMany reláciu, objekt User obsahuje zoznam skupín a Group obsahuje zoznam užívateľov. Keby sme teda chceli odoslať konkrétnu inštanciu User, tak by sme museli odoslať aj všetky inštancie Group ktoré má v zozname. A pre každú inštanciu Group by sme museli odoslať aj všetky inštancie User, ktoré majú v zozname, čo vedie k nekonečnému cyklu. Preto na odoslanie informácií o entite User použijeme triedu UserDTO, ktorá má namiesto zoznamu objektov Group zoznam primárnych kľúčov daných skupín. Všetky DTO objekty a DTOMapper sa nachádzajú v balíku dto.

Ak nastane výnimka pri volaní servisných metód, je potrebné ju zachytiť a vrátiť chybový kód ako odpoveď na danú HTTP požiadavku. Na to slúži trieda RestResponseEntityExceptionHandler ktorá je označená anotáciou `@ControllerAdvice` a zachytáva výnimky, ktoré nastanú v triedach kontrolerov. Pri zachytení výnimky vrátia odpovedajúci chybový kód a správu.

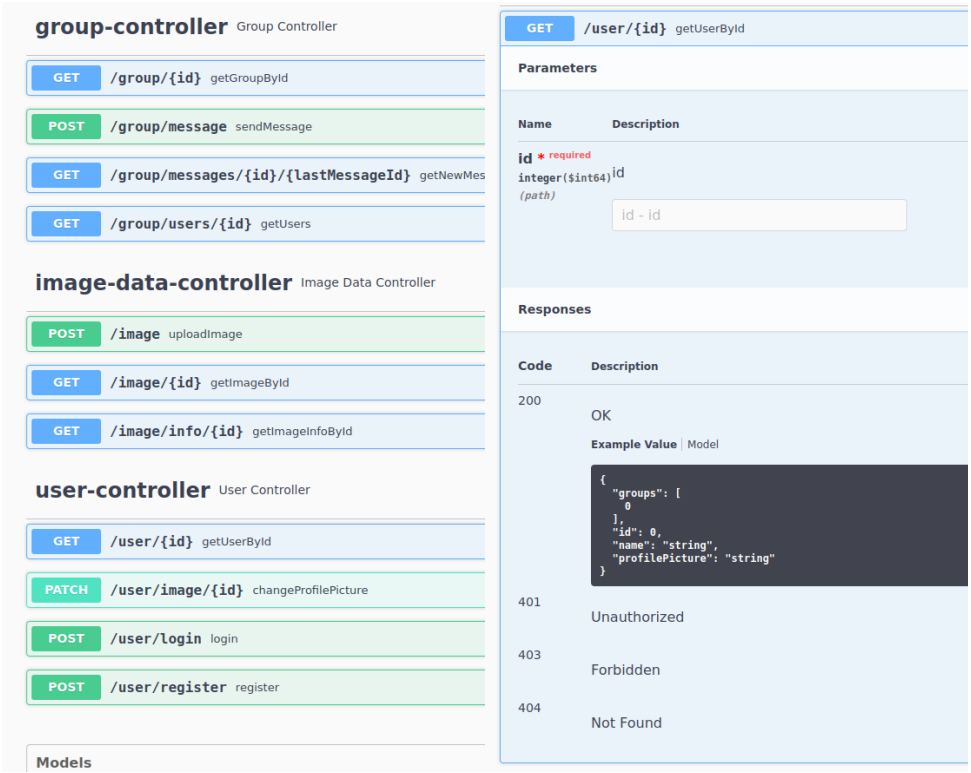
Príklad zachytenia výnimky NotFoundException:

```
@ControllerAdvice
public class RestResponseEntityExceptionHandler extends
    ResponseEntityExceptionHandler {

    public static final String FIELD_MISSING = "FIELD_MISSING";
    public static final String FIELD_INVALID = "FIELD_INVALID";

    @ExceptionHandler(value = {NotFoundException.class})
    protected ResponseEntity<Object> handleNotFoundException(
            NotFoundException ex) {
        return ResponseEntity.status(HttpStatus.NOT_FOUND).body(
            ex.getErrorCode());
    }
```

Swagger

Swagger je zbierka open-source nástrojov postavených na špecifikácii OpenAPI, ktoré pomáhajú vývojárom navrhnúť, zostaviť, zdokumentovať a používať REST API [24]. V tomto projekte bol Swagger použitý primárne na vytvorenie interaktívnej dokumentácie REST API.

Na automatizované vytvorenie Swagger dokumentácie bol použitý SpringFox a konfiguračná trieda SpringFoxConfig v balíku config. Vygenerovanú dokumentáciu je možné zobraziť vo webovom prehliadači na URI `http://example.server/swagger-ui/` počas behu aplikácie. V rámci interaktívnej dokumentácie sú zobrazené všetky koncové body a informácie o ich vstupných parametroch a výstupných hodnotách. Taktiež je možné zasielať HTTP požiadavky na dané koncové body pomocou vzorov priamo v rámci Swagger UI.



Obrázok 4.2: Swagger UI interaktívna dokumentácia

Autentizácia a autorizácia

Autentizácia a autorizácia sú dôležité mechanizmy na zabezpečenie zdrojov web API. Autentizácia je proces overenia identity užívateľa a deje sa väčšinou pred autorizáciou. Autorizácia je proces overenia povolení užívateľa a určuje aké zdroje môže daný užívateľ používať [3].

V našom prípade chceme aby mali prístup k zdrojom len prihlásení užívatelia. Autentizácia teda v našom prípade prebieha na základe overenia užívateľského mena a hesla, ktoré zašle klient cez HTTP požiadavku, aplikácia overí prihlasovacie údaje s prihlasovacími údajmi z databázy a pri úspešnom prihlásení pošle klientovi späť automaticky vygenerovaný autorizačný token, ktorý si klient uloží a posiela ho v hlavičke každej ďalšej požiadavky. Na autorizáciu je teda použitý token, ktorý je automaticky generovaný a časovo obmedzený. Konfigurácia zachytávačov je nastavená v triede InterceptorAppConfig v balíku config. Metóda na generovanie autorizačných tokenov v triede TokenGenerator vyzerá nasledovne:

```
public String generateToken() {
     byte[] randomBytes = new byte[24];
     secureRandom.nextBytes(randomBytes);
     String token = base64Encoder.encodeToString(randomBytes);
     tokens.entrySet().removeIf(entry -> entry.getValue()
            .isBefore(LocalDateTime.now()));
     tokens.put(token, LocalDateTime.now().plusHours(1));
     return token;
}
```

4.1.4 Nasadenie

Serverovú stranu aplikácie bolo treba nasadiť na webový server, na ktorý majú prístup všetci užívatelia s klientskou mobilnou aplikáciou. Keďže na nasadenie databázy bol použitý cloudový server od Microsoft Azure 4.1, rovnaká technológia bola použitá aj na nasadenie serverovej strany aplikácie. Azure ponúka rýchle a jednoduché nasadenie SQL databáz a webových aplikácií ako PaaS (Platform as a Service). Taktiež ponúka množstvo analytických nástrojov na monitorovanie nasadených aplikácii a databáz [11].

Na nasadenie Spring Boot aplikácie na Azure stačí skompilovať danú aplikáciu do JAR formátu a nahrať ju na pripravený Azure server.

4.2 Klientská strana

Klientská strana aplikácie bola implementovaná vo frameworku MAUI s použitím programovacích jazykov C# a XAML. Hlavnou výhodou MAUI je zdieľaný zdrojový kód medzi platformami [14]. Väčšina zdrojového kódu je v tomto projekte zdieľaná a jediný platformovo špecifický kód je v súbore AndroidManifest.xml, kde sú deklarované povolenia Android aplikácie ako napríklad prístup k gps polohe zariadenia a takisto je tu definovaný API kľúč na prístup k Google Maps API.

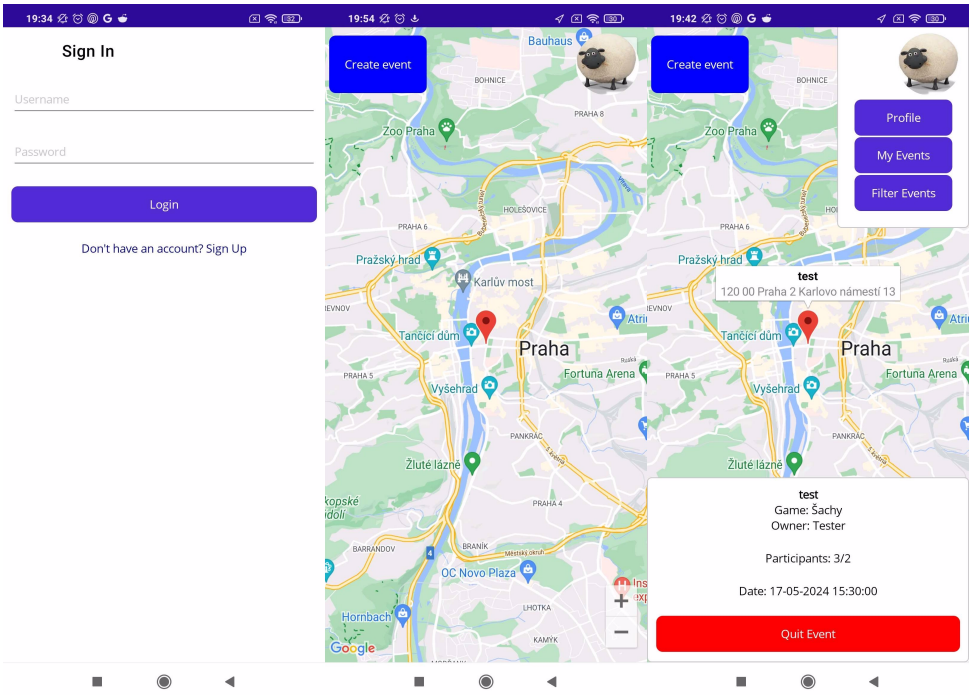
MAUI projekt bol implementovaný podľa návrhového vzoru MVVM 2.4 a ako základ bol použitý MAUI template na multiplatformovú aplikáciu. Zdrojový kód je rozdelený na balíky podľa návrhu s pridaním balíkov Service, ktorý patrí do modelu, Converters a Utils, ktoré obsahujú pomocné triedy. Hlavná konfigurácia sa nachádza v triede MauiProgram, ktorá spúšťa hlavnú triedu App. Na zostavenie projektu je použitý súbor BoardGameFinder.csproj a zostavuje ho MSBuild zabudovaný vo Visual Studiu.

4.2.1 Užívateľské rozhranie

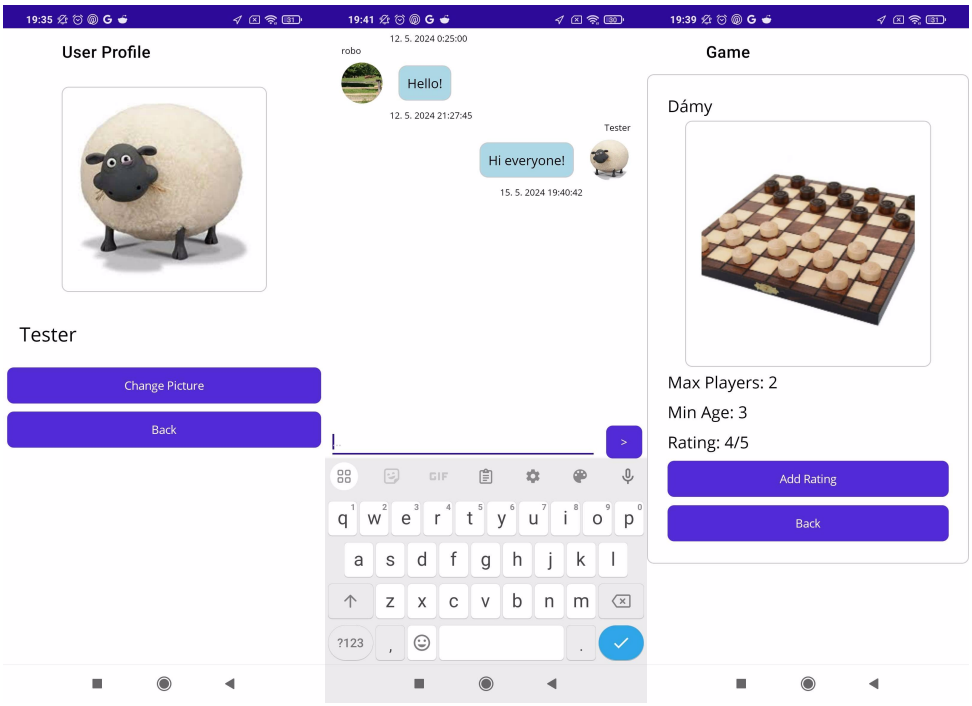
Prvým krokom v implementácii klientskej strany aplikácie bolo užívateľské rozhranie. Logika užívateľského rozhrania v MVVM návrhovom vzore je implementovaná v komponentoch View a ViewModel. V balíku View sú implementované jednotlivé stránky aplikácie a v balíku ViewModel sú jednotlivé viewmodely viazané na konkrétne stránky.

Stránky

Celá aplikácia pozostáva z piatich stránok, LoginPage, ProfilePage, GroupPage, GamePage a MainPage, implementovaných podľa návrhu wireframes 2.2, ktorých funkcionalita bola implementovaná v ich príslušných viewmodeloch podľa scenárov užitia 2.2.2.



Obrázok 4.3: Snímky obrazovky - Prihlasovacia a hlavná stránka



Obrázok 4.4: Snímky obrazovky - Profilová stránka, čet a stránka hry

Každá stránka implementuje rozhranie ContentPage a skladá za so XAML súboru s deklaráciami obsahu užívateľského rozhrania a C# súboru, ktorý obsahuje tzv. code behind. V code behind súbore môže byť implementovaná logika danej stránky, ale keďže bol použitý návrhový vzor MVVM, logika v code behind je minimálna a väčšina je v naviazanej viewmodel triede. V nasledujúcom príklade je inicializácia prihlasovacej stránky LoginPage.

```
public partial class LoginPage : ContentPage
{
    public LoginPage(LoginViewModel loginVM,UserServiceuserService)
    {
        InitializeComponent();
        BindingContext = loginVM;
```

Metóda InitializeComponent inicializuje XAML formulár prihlasovacej stránky a nastavenie vlastnosti BindingContext na objekt triedy LoginViewModel vytvorí dátovú väzbu na daný objekt. Dátová väzba potom dovoľuje stránke LoginPage pristupovať k vlastnostiam a metódam z triedy LoginViewModel a používať ich v ovládacích prvkoch deklarovaných v XAML súbore. Príklad viazania vlastnosti Username z triedy LoginViewModel na ovládací prvok Entry z LoginPage:

```
<Entry Grid.Row="1" Placeholder="Username" Text="{Binding Username,
    Mode=TwoWay}" Margin="10" />
```

Vlastnosť Username je v triede LoginViewModel deklarovaná nasledovne:

```
private string _username;
public string Username { get => _username; set { _username = value;
    OnPropertyChanged(nameof(Username)); } }
```

Pri nastavení, alebo zmene hodnoty Username sa spustí udalosť OnPropertyChanged, ktorá upozorní všetky ovládacie prvky ktoré danú vlastnosť používajú.

Dependency injection

Vytváranie inštancií stránok nie je implementované priamo, ale pomocou navigácie medzi stránkami s použitím metódy Shell.Current.GoToAsync. V triede AppShell sú registrované trasy jednotlivých stránok, na ktoré je možné prejsť z akejkoľvek inej stránky.

```
public partial class AppShell : Shell
{
public AppShell()
{
     InitializeComponent();
     Routing.RegisterRoute(nameof(LoginPage), typeof(LoginPage));
     Routing.RegisterRoute(nameof(ProfilePage), typeof(ProfilePage));
     Routing.RegisterRoute(nameof(GroupPage), typeof(GroupPage));
     Routing.RegisterRoute(nameof(GamePage), typeof(GamePage));
}
}
```

Na inštanciovanie jednotlivých stránok počas navigácie a riešenie závislostí v ich konštruktoroch bol použitý dependency injection. V triede MauiProgram sú deklarované triedy ktoré môžu byť vložené ako typ singleton, alebo transient.

```
builder.Services.AddTransient<LoginPage>();
builder.Services.AddTransient<LoginViewModel>();
```

LoginPage a LoginViewModel sú deklarované ako transient, čo znamená, že keď je volaná metóda `Shell.Current.GoToAsync(nameof(LoginPage))`, MAUI Builder vytvorí novú inštanciu LoginViewModel a následne LoginPage, kam vloží LoginViewModel ako vstupný argument do konštruktoru [12].

4.2.2 Model

Model klientskej strany pozostáva z dvoch balíkov Model a Service. V balíku Model sa nachádzajú jednotlivé dátové objekty reprezentujúce biznisové objekty aplikácie a v balíku Service sa nachádzajú servisné triedy, ktoré slúžia na komunikáciu so serverovou stranou aplikácie.

Balík Service

Úlohov servisných tried je odosielanie HTTP požiadaviek serverovej strane aplikácie a vytváranie dátových objektov z balíku Model. Príklad metódy RequestUserAsync z triedy UserService:

```
public async Task<User> RequestUserAsync(long id)
{
User? ret = null;
Uri uri = new Uri(Constants.RestUrl + "user/" + id);
try
{
     HttpResponseMessage response = await _client.GetAsync(uri);
     if (response.IsSuccessStatusCode)
     {
         string content = await response.Content
                .ReadAsStringAsync();
         ret = JsonSerializer.Deserialize<User>(content,
                _serializerOptions);
     }
}
catch (Exception ex)
{
     Debug.WriteLine(@"\tERROR {0}", ex.Message);
}

return ret ?? MainUser;
}
```

Táto metóda pošle HTTP GET požiadavku na koncový bod definovaný premennou `uri`. `Constants.RestUrl` je URI servera na ktorom je nasadená serverová strana a je uložená ako reťazec v súbore Constants.cs. Premenná `_client` je objekt triedy HttpClient a jej metódy slúžia na posielanie HTTP požiadaviek. Ak má odpoveď od serveru status značiaci úspešnú požiadavku, telo odpovede je deserializované z formátu JSON do objektu triedy User z balíku Model.

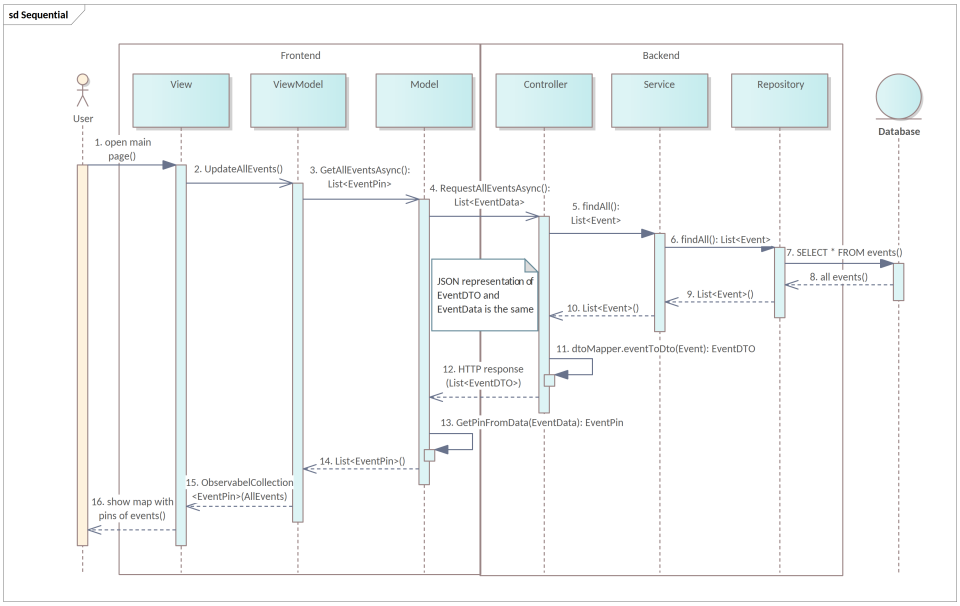
Balík Model

Všetky triedy v balíku Model sú jednoduché dátové triedy, okrem triedy EventPin, ktorá rozširuje triedu Pin z balíku Microsoft.Maui.Controls.Maps. Okrem vlastností nadradenej triedy Pin, obsahuje navyše informácie o udalosti ktorú reprezentuje. Tieto vlastnosti sú potom zobrazené na hlavnej stránke pri stlačení daného špendlíka na mape.

```
<maps:Map x:Name="map"
   Grid.Column="0"
   Grid.ColumnSpan="4"
   Grid.Row="0"
   Grid.RowSpan="4"
   IsShowingUser="True"
   MapClicked="OnMapClicked"
   ItemsSource="{Binding AllEvents}">
<maps:Map.ItemTemplate>
<DataTemplate>
     <model:EventPin
         EventId="{Binding EventId}"
         Location="{Binding Location}"
         Address="{Binding Address}"
         Label="{Binding Label}"
         Type="{Binding Type}"
         Game="{Binding Game}"
         Participants="{Binding Participants}"
         MaxParticipants="{Binding MaxParticipants}"
         Owner="{Binding Owner}"
         Joined="{Binding Joined}"
         Day="{Binding Day}"
         Time="{Binding Time}"
         Group="{Binding Group}"
         MarkerClicked="Pin_MarkerClicked"/>
</DataTemplate>
</maps:Map.ItemTemplate>
</maps:Map>
```

V príklade je deklarácia ovládacieho prvku mapy a špendlíkov triedy EventPin v súbore MainPage.xaml. Špendlíky sú viazané na kolekciu `AllEvents` typu ObservableCollection v triede MainViewModel. Kolekcia `AllEvents` je aktualizovaná volaním metódy GetAllEventsAsync z triedy EventService, ktorá pošle HTTP GET požiadavku na koncový bod `/event/all` serverovej strany aplikácie.

V nasledujúcom sekvenčnom diagrame je zobrazený celý proces aktualizácie udalostí na interaktívnej mape.



Obrázok 4.5: Sekvenčný diagram - aktualizácia udalostí

Autorizácia

Na autorizáciu HTTP požiadaviek je používaný autorizačný token vygenerovaný serverovou stranou aplikácie. Pri úspešnom prihlásení, alebo registrácii obdrží LoginService v tele odpovedi autorizačný token a id užívateľa. Autorizačný token je následne nastavený ako predvolená hlavička ktorá bude súčasťou každej požiadavky.

```
_client.DefaultRequestHeaders.Authorization =
    new AuthenticationHeaderValue("Bearer", token);
```

A id užívateľa je odoslané triede MainPage pomocou query vlastnosti.

```
await Shell.Current.GoToAsync($"..?user_id={userId}");
```

Nasadenie a distribúcia

Po dokončení implementácie, bola klientská strana zostavená pre platformu Android vo vývojovom prostredí Visual Studio. Po zostavení bol vygenerovaný digitálne podpísaný súbor vo formáte .apk, ktorý je možné použiť na inštaláciu aplikácie na zariadení s operačným systémom Android.

Kapitola 5
Testovanie

Testovanie je neoddeliteľnou súčasťou softvérového vývoja. Proces testovania umožňuje overiť správnosť a spoľahlivosť implementovaných riešení, identifikovať potenciálne chyby a zvýšiť kvalitu vyvíjaného produktu.

5.1 Automatizované testy

Automatizované testy sú vhodné predovšetkým na odhalenie chýb v kóde a okrajových prípadov, ktoré by mohli narušiť správne fungovanie aplikácie.

5.1.1 Unit testy

Základné automatizované testy sú jednotkové testy (unit testy). Jednotkové testy sa venujú testovaniu najmenších nezávislých jednotiek softvéru, teda jednotlivých metód. Overujú správne fungovanie samostatných častí kódu bez závislosti na ostatných častiach aplikácie. Pomáhajú odhaliť chyby vo funkcionalite jednotlivých jednotiek. Jednotkové testy by mali byť rýchle, opakovateľné a nemali by závisieť od vonkajších faktorov alebo databáz.

Keďže ide o overenie funkčnosti jednotlivých metód, nemá zmysel testovať primitívne metódy, ktoré obsahujú minimum biznisovej logiky. Preto boli jednotkové testy použité na biznisovej vrstve serverovej strany aplikácie, kde sa nachádza väčšina biznisovej logiky. Konkrétne boli testované metódy jednotlivých servisov, ktoré sú zodpovedné za vytváranie a správu biznisových objektov.

V zložke so zdrojovými kódmi serverovej strany src sa nachádzajú dva adresáre main a test. V adresáre test sa nachádzajú jednotlivé testovacie triedy, ktoré korešpondujú s triedami testovaných servisov. Každá z týchto tried obsahuje niekoľko jednotkových testov, ktoré testujú metódy príslušného servisu. Na implementáciu testov boli použité knižnice Spring Boot Starter Test a JUnit. Príklad testu metódy updateRating z triedy GameService je zobrazený v nasledujúcej ukážke. Na začiatku testu sú vytvorení dvaja užívatelia a jedna hra. Obaja užívatelia pridajú hodnotenie danej hre a jeden z nich ho ešte zmení. Test následne overí, či bolo hodnotenie danej hry správne aktualizované a či má správny počet hodnotení.

```
@Test
public void updateRating_ValidParameters_RatingUpdatedSuccessfully() {
    Game game = new Game();
    game.setName("Test Game");
    gameRepository.save(game);
    User user = new User("John Doe");
    userRepository.save(user);
    User user1 = new User("Joahnna Doe");
    userRepository.save(user1);

    gameService.updateRating(game, 3.5, user.getId());
    gameService.updateRating(game, 3.0, user.getId());
    Game updatedGame = gameService.updateRating(game, 4.5,
           user1.getId());

    assertNotNull(updatedGame);
    assertEquals((4.5 + 3.0) / 2, updatedGame.getRating());
    assertEquals(2, updatedGame.getNumOfRatings());
}
```

Výstupom testov je porovnanie očakávaných a získaných hodnôt a správy o tom, či test prebehol správne a nedošlo k žiadnym výnimkám.

Celkovo bolo implementovaných okolo 50 jednotkových testov, ktoré je možné v IntelliJ IDEA spustiť naraz a overiť tým správne fungovanie väčšiny metód v biznisovej vrstve. Správne vyhodnotenie všetkých testov však neznamená bezchybnosť testovaných metód, nakoľko dosiahnuť 100 percentné pokrytie testov je v praxi nemožné a často redundantné.

5.2 Manuálne testy

Počas vývoja boli rôzne časti aplikácie testované manuálne. Išlo o tzv. white box testing, pri ktorom je testerom vývojár a popri testovaní kontroluje správnosť zdrojového kódu a odstraňuje prípadné nájdené nedostatky. Ako testovacie scenáre boli použité scenáre užitia 2.2.2, alebo len ich časti [16].

Klientská a serverová strana boli spočiatku testované samostatne. Serverová strana aplikácie bola spustená vo vývojovom prostredí IntelliJ IDEA na lokálnej sieti s použitím lokálnej H2 databáze. Na testovanie jednotlivých HTTP požiadaviek bol použitý nástroj Insomnia v ktorom boli vytvárané REST požiadavky podľa ponúkaných vzorov.

Klientská strana aplikácie bola testovaná vo vývojovom prostredí Visual Studio na pripojenom lokálnom Android zariadení. Pri prvotnom testovaní komunikácie medzi klientskou a serverovou stranou bola použitá lokálna sieť Wi-Fi.

5.3 Užívateľské testy

Užívateľské testovanie (User Testing), je proces, pri ktorom užívatelia interagujú s produktom, aby sa zhromaždili informácie o tom, ako dobre produkt spĺňa ich potreby a očakávania. Cieľom testovania užívateľov je zabezpečiť, že produkt je intuitívny, jednoduchý na používanie a efektívny pre koncových užívateľov. Pomáha identifikovať potenciálne problémy s použiteľnosťou, navigáciou a celkovým užívateľským zážitkom.

5.3.1 Testy priechodu nového užívateľa

Testy priechodu sa zameriavajú hlavne na jednoduchosť a intuitívnosť používania aplikácie, ale aj na celkový dojem a funkcionalitu. Dôležitou súčasťou je, aby testeri vopred nemali žiadnu skúsenosť s testovanou aplikáciou. Testerom bolo zadaných niekoľko úloh, ktoré sa pokúsili splniť a následne vyplnili dotazník v ktorom hodnotili náročnosť daných úloh.

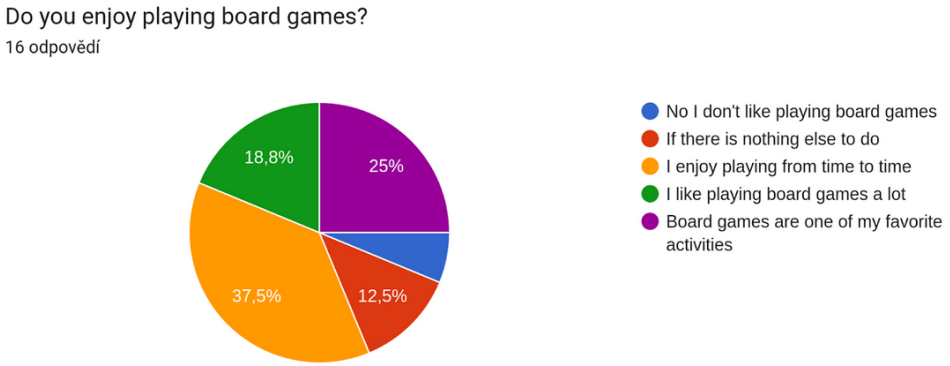
Úlohy ktoré testeri plnili boli definované podľa scenárov užitia 2.2.2 a znejú nasledovne:

1. Zaregistrujte sa
2. Zmeňte si profilový obrázok
3. Prihláste sa na udalosť na mape
4. Nájdite udalosti na ktoré ste prihlásené a napíšte správu do skupinového četu
5. Otvorte hru v jednej z vašich udalostí a dajte jej hodnotenie
6. Vytvorte novú udalosť
7. Nastavte filtrovanie udalostí podľa hry, alebo majiteľa
8. Opustite udalosť na ktorú ste prihlásení.

Užívateľské testy sa konali vzdialene pomocou formulára Google Forms v ktorom sa nachádza odkaz na stiahnutie .apk súboru na inštaláciu klientskej aplikácie. Náročnosť každej úlohy mohli testeri hodnotiť na 5 stupňovej škále od veľmi náročné po veľmi jednoduché. Ku každej úlohe bolo priradené aj pole na pripomienky od testerov, alebo prípadné nájdené chyby v aplikácii. Na konci formulára mali testeri ohodnotiť ako radi hrávajú stolné hry na 5 stupňovej škále od nerád hrávam stolné hry, po hranie stolných hier je jedna z mojich obľúbených aktivít. Taktiež mali uviesť, či by mali záujem o testovanú aplikáciu, keby bola oficiálne publikovaná.

5.3.2 Výsledky testov

Na konci testovania bol vyhodnotený testovací formulár a zozbierané pripomienky a návrhy na rozšírenie funkcionality. Testovania sa zúčastnilo 16 participantov, pričom 7 participantov uviedlo mierny, alebo vysoký záujem o hranie stolných hier a 3 uviedli mierny, alebo úplny nezáujem, 6 participantov sa vyjadrilo neutrálne.



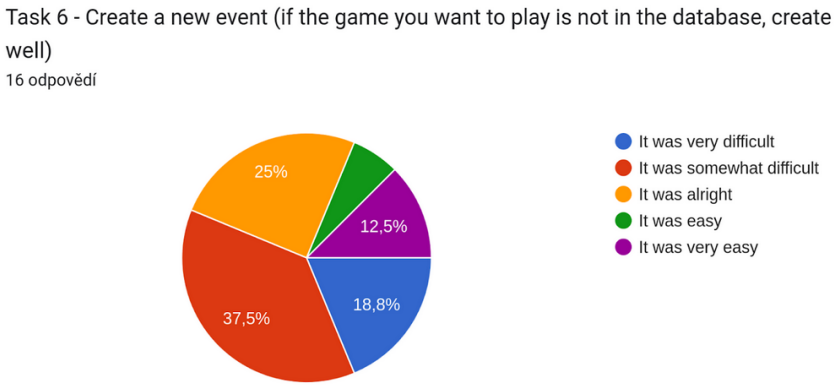
Obrázok 5.1: Výsledky dotazníka - záujem o stolné hry

Po vykonaní všetkých úloh nahlásilo mierny, alebo silný záujem o plnú verziu testovanej aplikácie 7 participantov, 7 participantov sa vyjadrilo neutrálne a 2 vyjadrili nezáujem. Z jednotlivých úloh 5.3.1 boli ako najjednoduchšie hodnotené úlohy 1,2,3 a 8. Ktoré ohodnotila polovica a viac participantov ako veľmi jednoduché. Iba jediný participant ohodnotil tieto úlohy ako mierne náročné.

Úlohy 4 a 5 ohodnotili len dvaja participanti ako mierne náročné, alebo veľmi náročné, pričom úloha 4 bola hodnotená prevažne pozitívne a úloha 5 prevažne neutrálne.

Úloha 7 bola taktiež hodnotená prevažne neutrálne, no až štyria participanti ju označili za mierne náročnú, alebo náročnú.

Najhoršie hodnotená bola úloha 6, ktorú až 9 participantov ohodnotilo ako mierne náročnú, alebo náročnú.



Obrázok 5.2: Výsledky dotazníka - hodnotenie funkcionality vytvorenia udalosti

Odhalené chyby a pripomienky testerov

Celkovo bolo zozbieraných približne 40 pripomienok. Najbežnejším problémom bolo vytváranie udalosti. Potreba vyhľadať a prípadne vytvoriť hru, pri nastavovaní informácií o udalosti sa ukázala ako neintuitívna a užívateľsky nepríjemná. Taktiež testerom chýbala spätná väzba pri nesprávnom vytvorení hry. Tieto nedostatky by mohli byť v budúcnosti odstránené pridaním hlásenia chýb a vyskakovacieho okna na pridanie hry, namiesto textového poľa s vyhľadávaním.

Druhým najčastejšie hláseným problémom bolo zobrazenie bieleho textu na bielom pozadí. Tento problém sa nikdy nevyskytol pri manuálnom testovaní užívateľského rozhrania a vyskytol sa len u pár testeroch. Na testovacom zariadení sa ho nepodarilo replikovať. Pôjde teda pravdepodobne o chybu na špecifických zariadeniach, alebo verziách Android. V budúcnosti by mohli byť zozbierané dáta o zariadeniach a systémoch testerov, ktorý natrafili na daný problém, aby bolo možné ho opraviť.

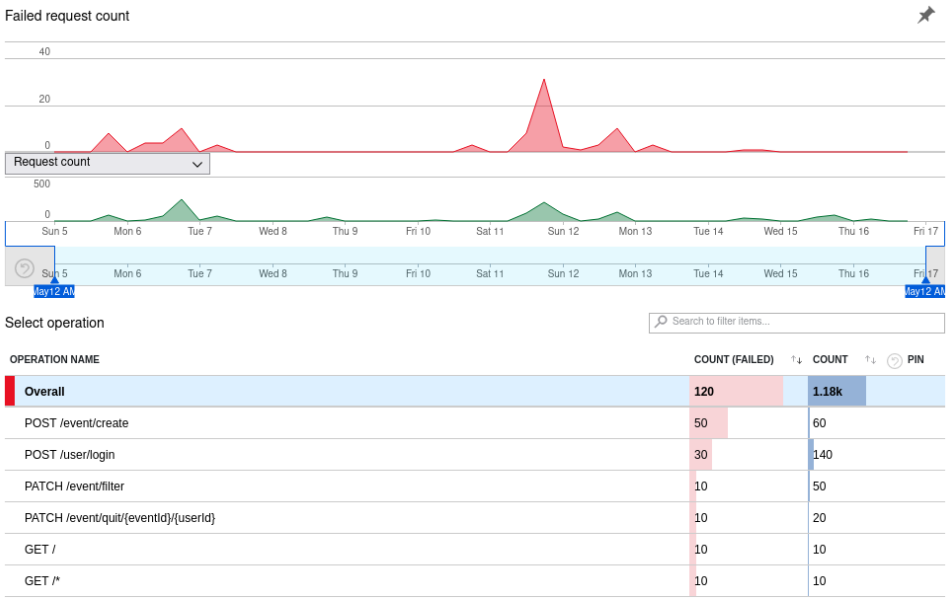
Chýbajúca odozva bola nahlásená aj pri úlohách 3, 4, a 5. Vo všetkých týchto prípadoch by v budúcnosti bolo vhodné pridať modálne okná, alebo hlásenia ktoré by upozorňovali užívateľa na vykonané zmeny, alebo chybové hlášky.

Viacero participantov požadovalo viac možností pri voľbe obrázku, ako otáčanie zvoleného obrázku, orezávanie, použitie fotoaparátu, alebo zvolenie predvolených avatarov.

5.3.3 Ukazatele výkonnosti

Počas konania užívateľských testov boli na platforme Azure zbierané dáta o výkonnosti serverovej strany aplikácie. Okrem údajov o zaťažení hardware boli zozbierané aj údaje o počte registrovaných požiadaviek, priemernej rýchlosti odozvy a počte zlyhaných požiadaviek.

V nasledujúcej infografike je zobrazený počet zlyhaných požiadaviek, ako aj celkový počet požiadaviek, zaokrúhlený na desiatky, za časový úsek 12 dní, počas ktorého prebiehalo užívateľské testovanie aplikácie.



Obrázok 5.3: Miera zlyhania požiadaviek na server

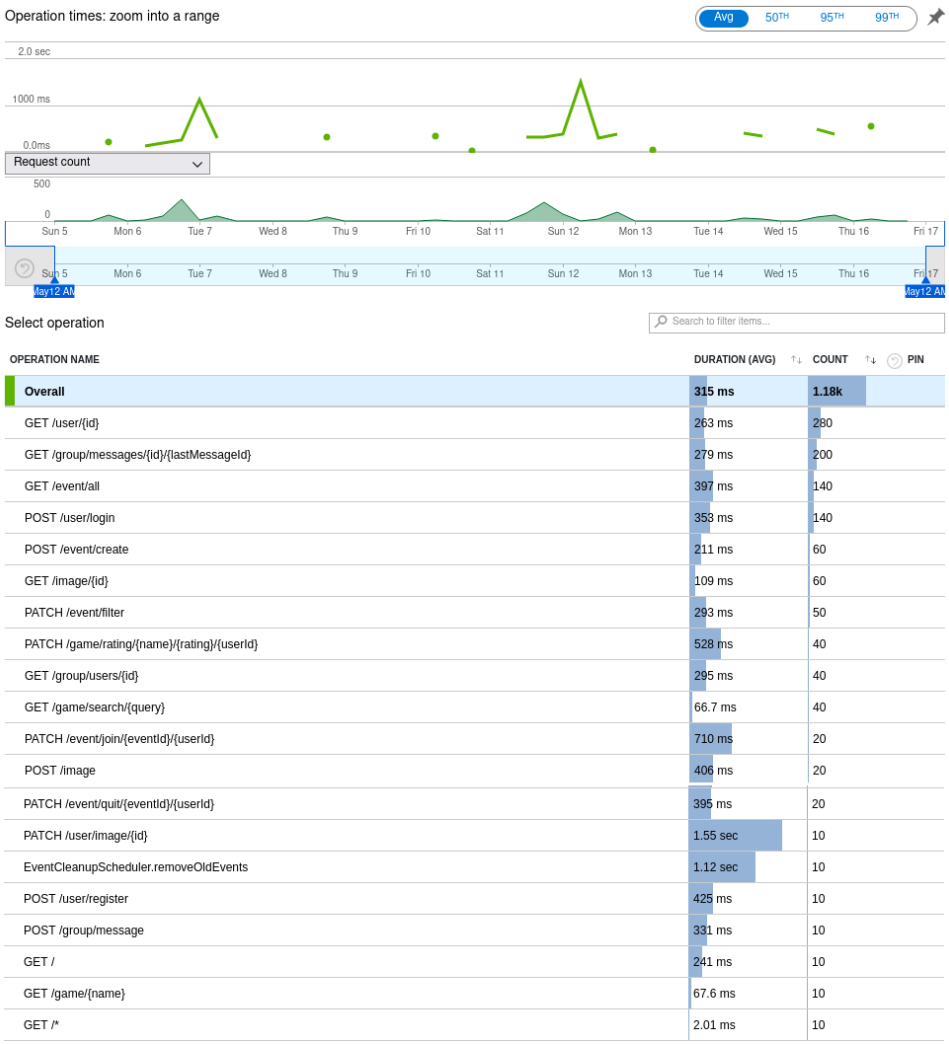
Počas testovacieho obdobia zaznamenala serverová strana testovanej aplikácie celkovo 1180 požiadaviek, z ktorých 120 bolo chybových. Požiadavky `GET/` a `GET/*` vznikli pri prístupe na URI servera cez webový prehliadač, takže dáta o nich nie sú relevantné ako ukazatele výkonnosti aplikácie. Z relevantných požiadaviek bolo teda 100 chybových, čo tvorí približne 8,6% všetkých zaregistrovaných požiadaviek.

Najväčšiu chybovosť mala požiadavka POST/event/create, ktorá koreš-ponduje s úlohou 6 z užívateľského testu - vytvorenie udalosti. Tento údaj potvrdzuje negatívnu spätnú väzbu od testerov a poukazuje na potrebu prepracovania danej funkcionality.

Celková priemerná doba odozvy na požiadavky bola 315ms, ale pohybovala sa niekedy až na 1500ms. Nárast priemernej doby odozvy však nebol spôsobený zaťažením, ale naopak malým množstvom požiadaviek s vysokou dĺžkou odozvy.

Požiadavka s najvyššou priemernou dĺžkou odozvy bola `PATCH/user/image/{id}`, ktorá slúži na nahranie profilového obrázku užívateľa do databázy. To je pravdepodobne spôsobené prenosom veľkého množstva dát, kompresiou a dekompresiou obrázku na serverovej strane. Na zrýchlenie odozvy danej požiadavky by bolo vhodné prerobiť danú metódu na serverovej strane, aby namiesto posielania uloženého obrázku späť klientovi v tele odpovede poslala len potvrdzujúcu správu.

Naopak najkratšiu odozvu mala požiadavka `GET/game/search/{query}`, ktorá slúži na vyhľadávanie hier v databáze na základe párovania reťazcov. To je vhodné, nakoľko rýchlosť odozvy pri danej funkcionalite výrazne zvyšuje užívateľskú spokojnosť.



Obrázok 5.4: Priemerné doby odozvy požiadaviek na server

5.3.4 Zhrnutie užívateľského testovania

Užívateľské testovanie prebehlo úspešne. V priebehu 12 dní sa testovania zúčastnilo 16 testerov, ktorí bez predošlých skúseností s testovanou aplikáciou prešli scenármi užitia aplikácie a overili jej funkcionalitu a intuitívnosť. Testeri odhalili viacero nedostatkov aplikácie, hlavne v spätnej väzbe aplikácie na niektoré akcie vykonané užívateľom. Úloha vytvorenia udalosti sa ukázala ako pomerne náročná a danú funkcionalitu by bolo vhodné upraviť aby bola užívateľsky prívetivejšia. Na zvyšok úloh a aj celkovo aplikáciu bola však prevažne pozitívna spätná väzba. Počas celej doby testovania nedošlo k pádu ani žiadnej zásadnej chybe na serverovej strane. Rovnako žiaden z testerov nezaznamenal pád mobilnej aplikácie.

Záver

Cieľom práce bolo spraviť analýzu, návrh, implementovať softvérové riešenie a nakoniec otestovať vytvorené riešenie aplikácie pre hráčov stolných hier. Na tento účel bola vykonaná podrobná analýza problému a definované hlavné biznisové ciele riešenia.

So zámerom špecifikácie požiadaviek bola vykonaná analýza trhu, existujúcich riešení a potenciálnych užívateľov.

Následne boli dané požiadavky spracované do návrhu technického riešenia a architektúry aplikácie. Rovnako bol vytvorený prvotný návrh dizajnu a scenáre užitia popisujúce prácu užívateľa s aplikáciou.

Na základe návrhu boli zvolené vhodné technológie na implementáciu a bola vytvorená proof of concept mobilná aplikácia na demonštráciu vytvorených scenárov užitia a overenie vhodnosti zvolenej technológie.

V rámci implementácie bola vytvorená serverová strana v jazyku Java s použitím Spring Boot frameworku, ktorá bola nasadená na cloud server od Microsoft Azure a spravuje Microsoft SQL databázu. Následne bola implementovaná klientská strana vo forme Android aplikácie s použitím .NET MAUI frameworku.

Nakoniec bola vytvorená aplikácia testovaná v priebehu 12 dní skupinou 16 testerov a boli zozbierané hodnotenia jednotlivých funkcionalít aplikácie z užívateľského pohľadu. Na chyby odhalené testermi boli navrhnuté riešenia, ktoré môžu byť v budúcnosti implementované. Aplikácia bola navrhnutá a implementovaná podľa moderných návrhových vzorov a je rozdelená do vrstevnej architektúry, takže budúce úpravy by mali byť jednoduché. Celkovo 43.8% testerov vyjadrilo záujem o testovanú aplikáciu a len 12.5% vyjadrilo nezáujem, takže sa dá hovoriť o úspešnom softvérovom riešení.

V budúcnosti by samozrejme bolo možné aplikáciu vylepšiť a rozšíriť. Okrem zlepšenia užívateľskej prívetivosti funkcionalít s ktorými mali testeri problémy a opravy odhalených chýb, by bolo vhodné napríklad navrhnúť originálnu grafickú identitu aplikácie, pridať systém priateľstiev medzi užívateľmi, priame správy medzi priateľmi, alebo blokovanie, či nahlasovanie užívateľov ktorí by sa správali nevhodne.

Všetky zadané ciele bakalárskej práce boli splnené a aplikácia vyvinutá v rámci implementačnej časti práce je vhodným nástrojom na komunikáciu a vytváranie udalostí pre hráčov stolných hier. Aplikácia umožňuje užívateľom

vytvorenie udalosti pre hranie vybranej hry na mieste definovanom gps súradnicami vybranom na interaktívnej mape. Tvorca udalosti môže vybrať čas konania, názov udalosti a počet voľných miest. Užívatelia môžu vidieť udalosti vytvorené inými užívateľmi na interaktívnej mape a prihlásiť sa na ne. Užívatelia môžu filtrovať udalosti zobrazené na mape podľa majiteľa, zvolenej hry, alebo počtu voľných miest. Po prihlásení na udalosť môže užívateľ komunikovať s ostatnými užívateľmi prihlásenými na danú udalosť prostredníctvom skupinového četu danej udalosti. Hry uložené v databáze môžu užívatelia hodnotiť.

Okrem zadaných cieľov bola funkcionalita aplikácie rozšírená o autentizáciu prostredníctvom prihlasovacieho mena a hesla a autorizáciu prostredníctvom časovo obmedzených autorizačných tokenov náhodne generovaných pri úspešnom prihlásení.

Literatúra

[1] Leiva Antonio. *Kotlin for Android Developers: Learn Kotlin the Easy Way While Developing an Android App*. Leanpub, 2018. ISBN: 1530075610.

[2] Board Game Arena. Board game arena. `https://boardgamearena.com/`. [Online; cit. 2.11.2023].

[3] Okta Auth0. Authentication vs authorization. `https://auth0.com/docs/get-started/identity-fundamentals/authentication-and-authorization`. [Online; cit. 13.5.2024].

[4] Šebek Jiří Cerny Tomas. Návrh sw - perzistentní vrstva. `https://moodle.fel.cvut.cz/pluginfile.php/399049/course/section/68931/2.Introduction%20to%20Software%20Design.pdf`. [Online; cit. 9.5.2024].

[5] Board Games Companion. Board games companion. `https://play.google.com/store/apps/details?id=com.progrunning.boardgamescompanion&hl=en&gl=US&pli=1`. [Online; cit. 2.11.2023].

[6] Eventbrite. Eventbrite. `https://www.eventbrite.com/`. [Online; cit. 2.11.2023].

[7] Insomnia. Insomnia. `https://insomnia.rest/`. [Online; cit. 9.5.2024].

[8] Kotlin. The six most popular cross-platform app development frameworks. `https://kotlinlang.org/docs/cross-platform-frameworks.html`. [Online; cit. 3.11.2023].

[9] Marketsplash. 65+ board game statistics: Industry, demographics sales. `https://marketsplash.com/board-game-statistics/`. [Online; cit. 10.1.2024].

[10] Meetup. Meetup. `https://www.meetup.com/`. [Online; cit. 2.11.2023].

[11] Microsoft. Azure app service. `https://azure.microsoft.com/en-us/products/app-service/`. [Online; cit. 14.5.2024].

[12] Microsoft. Dependency injection. https://learn.microsoft.com/en-us/dotnet/maui/fundamentals/dependency-injection?view=net-maui-8.0. [Online; cit. 15.5.2024].

[13] Microsoft. .net maui map. https://learn.microsoft.com/en-us/dotnet/maui/user-interface/controls/map?view=net-maui-8.0. [Online; cit. 12.1.2024].

[14] Microsoft. .net multi-platform app ui documentation. https://learn.microsoft.com/en-us/dotnet/maui/?view=net-maui-8.0&WT.mc_id=dotnet-35129-website. [Online; cit. 12.1.2024].

[15] Microsoft. Restful web api design. https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design. [Online; cit. 9.5.2024].

[16] Microsoft. White box testing. https://www.javatpoint.com/white-box-testing. [Online; cit. 15.5.2024].

[17] Match n Game. Matchngame. https://matchngame.com/. [Online; cit. 2.11.2023].

[18] Visual Paradigm. Demystifying use cases, scenarios, flow of events, and templates. https://guides.visual-paradigm.com/demystifying-use-cases-scenarios-flow-of-events-and-templates/#:~:text=Use%20case%20scenarios%20provide%20a,that%20may%20arise%20during%20execution. [Online; cit. 8.5.2024].

[19] Konieczny Piotr. Golden age of tabletop gaming: Creation of the social capital and rise of third spaces for tabletop gaming in the 21st century. *Polish Sociological Review*, 206:199–216, 2019. DOI: 10.26412/psr206.05.

[20] Positivise. 6 most trending databases used in 2022 for application development [developer's survey]. https://positiwise.com/blog/6-most-trending-databases-used-in-2022-for-application-development. [Online; cit. 3.11.2023].

[21] Spring. Jpa query methods. https://docs.spring.io/spring-data/jpa/reference/jpa/query-methods.html. [Online; cit. 11.5.2024].

[22] Spring. Spring boot. https://spring.io/projects/spring-boot. [Online; cit. 11.5.2024].

[23] Statcounter. Mobile operating system market share worldwide. https://gs.statcounter.com/os-market-share/mobile/worldwide. [Online; cit. 3.11.2023].

[24] Swagger. Open api guide. https://swagger.io/docs/specification/about/. [Online; cit. 12.5.2024].

[25] Tabletopia. Tabletopia. https://tabletopia.com/. [Online; cit. 2.11.2023].

[26] Wikipedia. List of http status codes. https://en.wikipedia.org/wiki/List_of_HTTP_status_codes. [Online; cit. 10.5.2024].

[27] Wikipedia. Model-view-viewmodel. https://en.wikipedia.org/wiki/ModelâĂŞviewâĂŞviewmodel. [Online; cit. 8.5.2024].

[28] Wikipedia. Proof of concept. https://en.wikipedia.org/wiki/Proof_of_concept. [Online; cit. 11.1.2024].