FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Webová aplikace pro tipovací soutěže
Student:	Daniel Blažek
Vedoucí:	Ing. Jan Blizničenko
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

Tato bakalářská práce se bude zabývat analýzou, návrhem a implementací webové aplikace, která umožňuje bezplatné tipování výsledků různých událostí. Smyslem práce je vytvořit platformu, na které bude moci skupina lidí předpovídat výsledek událostí, které si sami nadefinují. U těchto událostí mají možnost nastavení bodování a každá skupina má svůj bodovací žebříček.

Instrukce:
- Proveďte rešerši již existujících řešení
- Proveďte rešerši relevantních technologií, nástrojů, frameworků a knihoven
- Vytvořte návrh jednotlivých částí aplikace
- Proveďte implementaci aplikace
- Otestujte a vytvořte dokumentaci ke svému řešení

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 18. ledna 2023 v Praze.

Bakalářská práce

WEBOVÁ APLIKACE PRO TIPOVACÍ SOUTĚŽE

Daniel Blažek

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Jan Blizničenko
9. května 2023

Obsah

Seznam obrázků

Seznam výpisů kódu

Chtěl bych poděkovat svému vedoucímu Ing. Janu Blizničenko za vedení a pomoc při tvorbě bakalářské práce. Svým rodičům za podporu v průběhu celého studia. Sestře za cenné rady a oporu. Na závěr svým kamarádům, bez kterých by studium na této škole nebylo takové, jaké je.

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 9. května 2023 .....

Abstrakt

Tato bakalářská práce se zabývá analýzou, návrhem a implementací webové aplikace umožňující pořádání tipovacích soutěží. Smyslem práce je vytvořit platformu, na které bude moci skupina osob předpovídat výsledky vlastních událostí a porovnat své umění tipovat. U událostí je možné nastavit bodování a každá skupina má dostupný bodovací žebříček. Práce začíná rešerší existujících řešení tohoto problému, ze které vyplynulo, že se na trhu nenachází aplikace splňující tuto funkčnost. Na tuto rešerši reaguje tvorba aplikace rozdělené na klientskou a serverovou část, které spolu komunikují pomocí REST API zabezpečeného využitím autorizačního serveru. Tato implementovaná aplikace dovoluje vytvořit místnosti, ve kterých je uživatelům umožněno předpovídat výsledky vlastních událostí. Výsledná aplikace byla otestována různými druhy testů a detailně zdokumentována.

Klíčová slova webová aplikace, tipovací soutěž, frontend, backend, autorizační server, Angular, Spring, Keycloak

Abstract

This bachelor thesis deals with the analysis, design and implementation of a web application that enables the organization of guessing competitions. The purpose of the thesis is to create a platform on which a group of people can predict the results of their own events and compare their guessing skills. Scoring can be set for each event and each group has an available leaderboard. The work started with a search of existing solutions to this problem, which showed that there is no application on the market that meets this functionality. The search was followed by the creation of an application divided into client and server parts, which communicate with each other using a REST API secured by the use of an authorization server. This implemented application allows the creation of rooms in which users are allowed to predict the results of their own events. The resulting application has been tested by various types of tests and documented in detail.

Keywords web app, guessing game, frontend, backend, authorization server, Angular, Spring, Keycloak

Seznam zkratek

API	Application Programming Interface
CORS	Cross-Origin Resource Sharing
CRUD	Create, Read, Update, Delete
CSS	Cascading Style Sheets
DTO	Data Transfer Object
HMAC	Hash-based Message Authentication Code
HTML	Hypertext Markup Language
IAM	Identity and Access Management
IDE	Integrated Development Enviroment
JDBC	Java Database Connectivity
JPA	Java Persistence API
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
JWT	JSON Web Token
MPA	Multi Page Application
OAuth 2.0	Open Authorization 2.0
PHP	Hypertext Preprocessor
REST	Representational State Transfer
RSA	Rivest–Shamir–Adleman
SAML 2.0	Security Assertion Markup Language 2.0
SOAP	Simple Object Access Protocol
SPA	Single Page Application
SSO	Single sign-on
VCS	Version Control System
XHTML	Extensible Hypertext Markup Language
XML	Extensible Markup Language

Úvod

Předpověď sportovních či jiných výsledků je zpestření sportovní či jiné události. Článek od firmy SportBusiness týkající se zapojení fanoušků do události [1] říká, že člověk, který si něco vsadil na výsledek, je do události více investován a zážitek mu připadá intenzivnější. Ať už vsadil peníze či pouze svůj respekt u kamarádů. Na trhu jsou desítky aplikací, které umožní uživateli předpovědět výsledky takovýchto událostí a porovnat své umění s přáteli. Bohužel nástroje nabízí pouze velké a důležité události a uživatel, který by si chtěl tuto soutěž uspořádat na menší či neoficiální události, které tyto platformy nenabízejí (například školní turnaj v šachu), nemá nástroj, který by mu usnadnil práci a nemá jinou možnost než body počítat ručně. Proto je záměrem této práce vyvinout aplikaci, která to uživateli umožní. Je očekáváno, že tato aplikace bude oceněna mnoha lidmi a bude dlouhodobě využívána.

Cílem této bakalářské práce je návrh a implementace webové aplikace, která umožní pořádání tipovacích soutěží s vlastními událostmi a bodováním. Nedílnou součástí práce bude vytvoření uživatelsky přívětivého rozhraní s jednoduchým ovládáním.

Část práce, která se týká rešerše, bude zaměřena na seznámení s již existujícími řešeními tohoto problému včetně jejich pozitiv a negativ a také na analýzu dostupných technologií a nástrojů pro vývoj takovéto webové aplikace.

Druhá část se bude zabývat návrhem všech nutných komponent a samotné implementace aplikace. Součástí této sekce bude také testování a dokumentace projektu.

Kapitola 1

Analýza existujících řešení

První část práce je věnována analýze již existujících řešení. Tedy platformám, které umožňují předpovídat výsledky událostí s možností vytvoření skupin uživatelů a porovnání výstupu těchto předpovědí. Součástí této sekce budou nejen webové aplikace, ale i mobilní. Nezáleží na tématu, kterému se daná aplikace věnuje. Tedy nebude bráno v potaz, zda se zabývají fotbalem, esportem či zda mají široké spektrum zaměření.

1.1 Tipování v počítačových hrách

Jako první budou představeny herní studia, která umožnila hráčům svých her předpovídat výsledky různých událostí. Většinou se jedná o velké turnaje v daných hrách. Tyto aplikace jsou analyzovány, aby bylo zjištěno jak jejich tvůrci řešili problémy, které se při vytváření takovéhoto nástroje naskytnou, například způsob odemykání vyřazovací fáze či oddělení jednotlivých částí turnaje.

1.1.1 League of Legends Pick'em

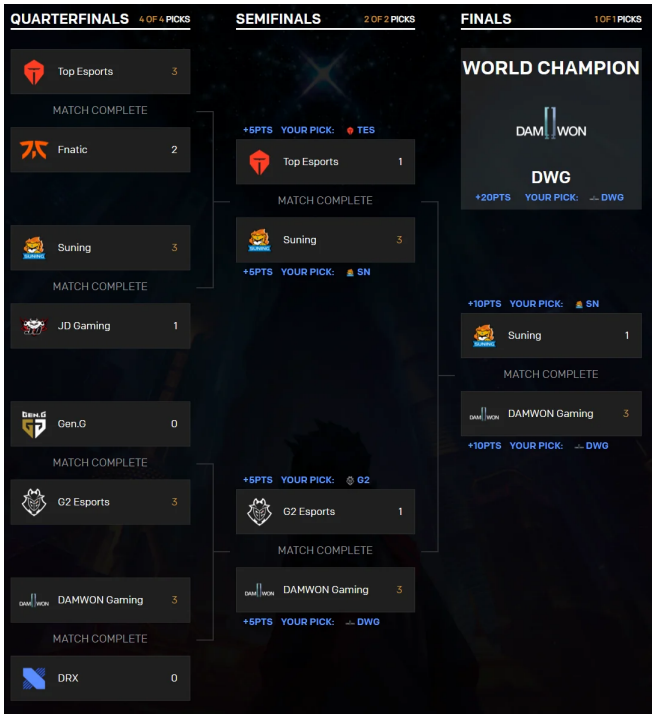
Jedná se o aplikaci, která tvůrci této práce vnukla prvotní nápad pro vytvoření platformy s tímto cílem. League of Legends Pick'em (dále jen Pick'em) je webová aplikace, která umožňuje jednou za rok každému hráči League of Legends tipovat výsledky mistrovství světa v této hře za herní odměny. Je to pouze část rozsáhlého projektu LolEsports, který se věnuje oficiálnímu elektronickému sportu této hry.

Základ této aplikace vznikl již v roce 2014. Bylo možné zde hlasovat pouze o umístění ve skupinové a následné vyřazovací fázi tohoto turnaje. Za poslední roky byla přidána možnost tipování statistik celého turnaje za různý počet bodů např. hráč s nejvíce zásahy nebo nejvíce hraná postava či takzvaná mince, která náhodně předpoví za uživatele pokud by zapomněl předpovědět sám. [2] 1.1

Tato část je inspirací pro přidání možnosti vytvoření podobných statistik do tohoto projektu. Tipování ve vyřazovací fázi bylo vyřešeno pomocí postupného odemykání, například se odehrají všechny semifinále a následně se uživatelům umožní tipování na finále s korektními týmy. Je zde také možnost vytvoření skupin s ostatními uživateli a zobrazení žebříčku. [4]

1.1.2 Counter Strike: Global Offensive Pick'em

Tato platforma se nachází přímo ve hře, na níž je možnost se dostat jedním kliknutím z hlavního menu. Na rozdíl od předchozí hry není tipování zdarma. Původně se tipovalo nálepkami, které



Obrázek 1.1 Ukázka League of Legends Pick'em vyřazovací fáze [3]

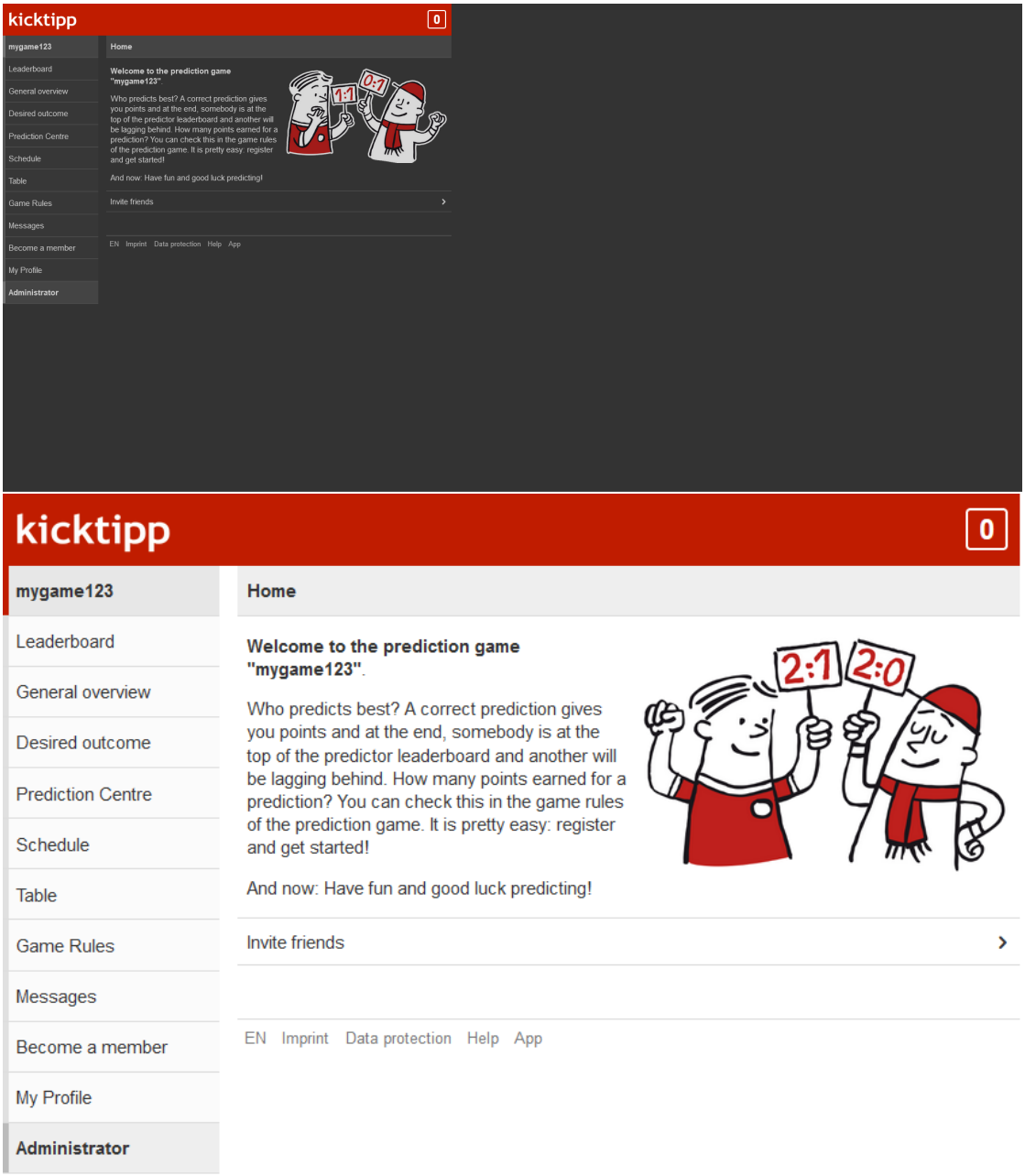
bylo možné nalepit na svou virtuální zbraň, nyní pouze pomocí divácké vstupenky, kterou si člověk zakoupí za reálné peníze. Odměnou byly mince různé úrovně, které hráč obdržel pokud získal různý počet bodů. Tuto výhru si následně mohl vystavit na svůj profil. Předpovídal se celý turnaj na jednou. Tato myšlenka dodala inspiraci implementovat do aplikace více stylů předpovědí např. všechny najednou nebo naopak postupně. Daná aplikace nemá možnost vytvoření oddělené skupiny, ale pouze srovnání s celým seznamem přátel na platformě Steam. [5]

1.2 Kicktipp

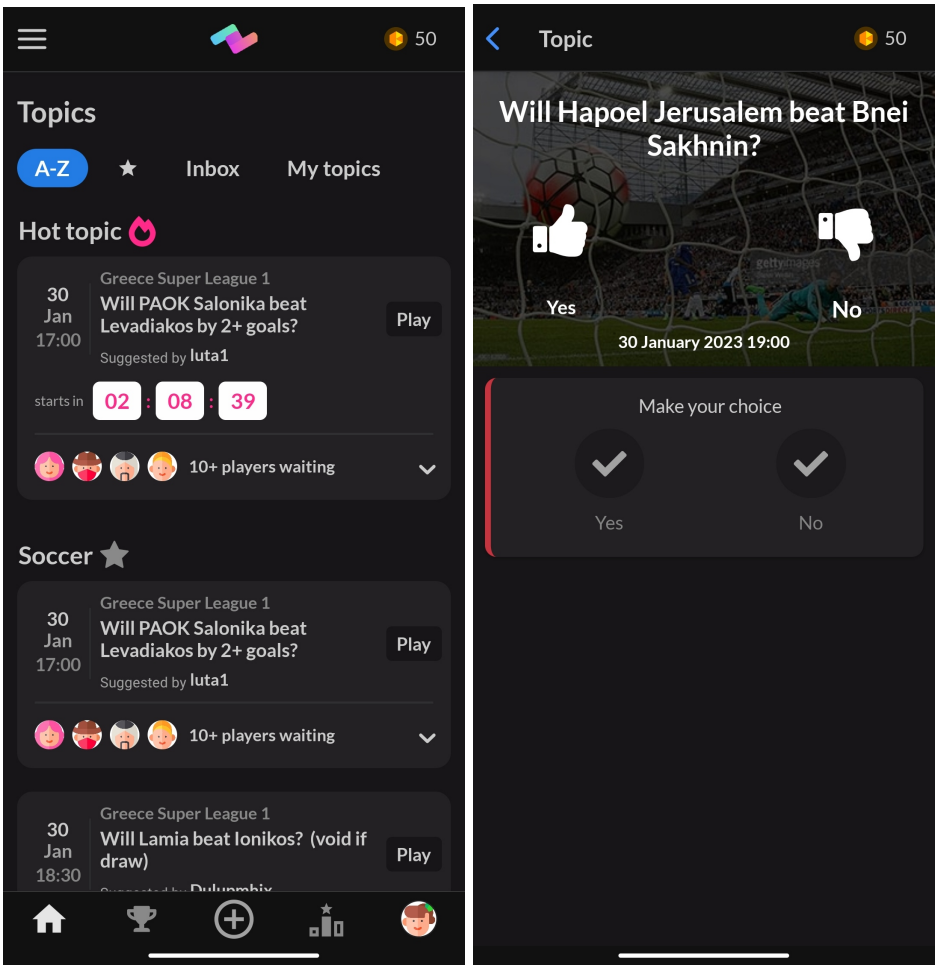
Kicktipp je webová aplikace, která má nejpodobnější vlastnosti jako platforma, která by měla být výsledkem této práce.

Jedná se o rozsáhlý projekt, který na první pohled funguje spolehlivě. Avšak není moc uživatelsky přívětivý, jak lze vypozorovat z obrázků 1.2. Je zde široká nabídka sekcí v nichž se nachází další podstránky. Samotná webová aplikace také využívá pouze čtvrtinu stránky, jak lze vidět na obrázku 1.2.

Aplikace podporuje vytvoření vlastních skupin, do kterých si uživatel, který je zvolen jako administrátor, smí přidat události. Není zde možnost vytvoření vlastních událostí a tedy jediné události, které smí administrátor přidat jsou již před připravené od stránky, které se týkají pouze velkých světových turnajů či závodů. U stránky se je možnost rozdělit skupinu na týmy, či možnost nastavení denního hodnocení (každý den se vyhodnotí žebříček a vybere se denní vítěz). Obsahuje také možnost přidání statistik celého turnaje, které jsou také před připravené. [6]



Obrázek 1.2 Ukázka aplikace Kicktipp [6]



■ **Obrázek 1.3** Ukázka aplikace PeerBet [7]

1.3 PeerBet

PeerBet je jediná mobilní aplikace, která se v této sekci nachází. Jde o velice jednoduchou aplikaci, která umožňuje vytvoření jednotlivých témat, u nichž si uživatelé vybírají ano či ne viz první obrázek 1.3. Témata se buď zveřejňují pro všechny uživatele aplikace po schválení administrátorem nebo si uživatel smí vytvořit téma jen pro své přátele i bez schválení. Není zde bodování, pouze počet špatných a správných odpovědí bez žebříčku. Nemá možnost vytvoření skupiny. Na druhém obrázku 1.3 se nachází ukázka seznamu všech témat. Tato aplikace byla v průběhu tvorby práce ukončena. [7]

1.4 Ostatní

Existuje množství dalších řešení, ale většina platforem, které implementují podobnou funkcionalitu, jsou pouze alternativy sázkových kanceláří, takové aby si uživatelé mohli vyzkoušet sázet i bez rizika ztráty peněz. Aplikace jsou většinou neudržované a pouze pár jich podporuje možnost vytvoření skupiny uživatelů. Žádná nemá možnost vytvoření vlastní události či turnaje.

1.5 Závěr

Po analýze existujících řešení bylo rozhodnuto pro vytvoření aplikace, který má poměrně široké spektrum zaměření a volnost ve tvorbě jednotlivých událostí, čímž umožní uživatelům vytvořit soutěž podle svých představ. Jedná by se o model, který je synergií aplikací PeerBet a Kicktipp. Aplikace bude mít možnost vytvoření místnosti, ve které budou moci nadále administrátoři vytvářet události. Tyto události budou mít určitý počet bodů, které uživatel obdrží při správné odpovědi a datum uzavření předpovědí (v praxi se jedná o začátek dané události). Navíc se k těmto událostem bude moci přidat libovolné množství odpovědí. Tento model by měl uživatelům umožnit reprezentovat jakýkoli formát.

Například pokud by chtěl uživatel reprezentovat automobilový závod s předpovědí na první tři umístění, tak by vytvořil tři události daného závodu, naplnil možnosti všemi závodníky a nastavil by uzamknutí událostí na začátek závodu. Tím by umožnil hráčům vybrat u každého umístění řidiče. Po skončení závodu by nahrál výsledek do této aplikace a ta by sama vypočítala body a zobrazila tabulku pořadí hráčů.

Pokud by uživatel chtěl reprezentovat vyřazovací systém s postupným odemykáním. V jednotlivých fázích by vytvářel události a přidal do každé dvě možnosti (soupeři daného zápasu). Poté by postupně zadával výsledky. Také by měl možnost nastavit váhu jednotlivých fází pomocí bodů.

Kapitola 2
Specifikace

Tato kapitola představuje o jakou aplikaci by se mělo jednat. Toto je ujasněno pomocí definice funkčních a nefunkčních požadavků a jejich krátkých popisů. Na závěr jsou také představeny případy užití a jejich diagramy.

2.1 Funkční požadavky

Jedná se o seznam funkcionalit, které má systém poskytovat. Funkční požadavky nespecifikují postup ani důvod implementace požadavků pouze funkčnost, kterou musí splňovat. [8]

- **(F1) Přihlášení a registrace uživatele**
  Uživatel se bude moci přihlásit přes zabezpečený formulář. Pokud již nemá účet, tak si ho bude moci vytvořit.
- **(F2) Přidání uživatele do seznamu přátel**
  Uživatel bude mít možnost si přidat další uživatele do svého seznamu přátel.
- **(F3) Vytvoření místnosti**
  Uživateli bude umožněno vytvořit místnost, ve které se budou odehrávat budoucí tipovací soutěže. U této místnosti bude možnost nastavení ikonky, jména, popisu a další konfigurace.
- **(F4) Nastavení uživatele jako administrátora**
  Tvůrce skupiny či administrátor bude moci přiřadit uživateli, který je členem dané místnosti, roli administrátora, čímž mu umožní pracovat se skupinou (vytvářet události, vyhodnocovat události atd.).
- **(F5) Vytvoření události**
  Administrátor místnosti bude schopný vytvořit tipovací událost v dané místnosti. Součástí založení takovéto události bude také nastavení bodování a přidání všech možností výběru.
- **(F6) Přidání uživatele do místnosti**
  Administrátor bude moci přidat uživatele do tipovací místnosti, pokud již v místnosti není. Přidání bude probíhat buď pomocí výběru ze seznamu přátel či pomocí vytvoření odkazu, přes který se bude moci další uživatel připojit od místnosti i bez nutnosti, aby se nacházel v seznamu přátel.
- **(F7) Umožnění tipování**
  Uživateli bude umožněno tipování výsledků událostí v místnostech, ve kterých se nachází

- **(F8) Zadání výsledků a vyhodnocení tipů**
  Uživateli, který bude mít v místnosti roli administrátora, bude umožněno vyplňovaní výsledků událostí. Vyhodnocení předpovědí a výpočet bodování se bude provádět automaticky po každém zadání výsledků jednotlivých událostí.

- **(F9) Zobrazení tabulky**
  Všichni uživatelé dané místnosti si budou moci zobrazit pořadí či tabulku, v níž se budou nacházet jednotlivé tipy a pořadí uživatelů seřazených dle bodů.

2.2 Nefunkční požadavky

Nefunkční požadavky popisují nároky na kvalitu, které jsou určovány pro celou aplikaci. Jsou méně detailní než funkční požadavky. Nepopisují, co by měl systém dělat, ale jak by měl systém fungovat. Zabývá se o otázky ohledně výkonu, údržby, bezpečnosti a spolehlivosti. [9]

- **(N1) Webová aplikace**
  Bude se jednat o webovou aplikaci, která bude veřejně dostupná a dohledatelná. Webová aplikace je přístupná přes prohlížeč, čímž umožňuje spuštění na různých platformách.

- **(N2) Podpora moderních prohlížečů**
  Systém bude funkční na většině moderních prohlížečů.

- **(N3) Zabezpečení**
  Aplikace bude bezpečně pracovat s hesly a jejich šiframi.

- **(N4) Responzivita**
  Projekt je primárně určen pro počítačové prohlížeče, ale responzivita umožní uživateli využívat tuto aplikaci i na zařízeních s jiným poměrem stran například na mobilních telefonech či tabletech.

- **(N5) Rozšiřitelnost**
  Systém bude naimplementován tak, aby přidání jakéhokoli dalšího rozšíření bylo co nejjednodušší.

2.3 Případy užití

Tato sekce popisuje případy užití aplikace a jejich aktéry. Případ užití je seznam funkčností, které by měl systém poskytovat danému aktérovi činnosti a zároveň zachycuje funkční požadavky. [10]

2.3.1 Aktéři

Celkově se ve dvou diagramech případů užití nachází čtyři aktéři, kteří jsou rozdělení po dvojicích. Toto rozdělení má svůj důvod, jelikož je požadováno separovat celkovou funkčnost aplikace od funkčnosti v místnosti. Aktéři jsou následující:

- **Nepřihlášený uživatel**

  Většina funkcí aplikace je pro nepřihlášeného uživatele nepřístupná. Jediné, co smí tento uživatel udělat, je přihlášení či registrace, čímž se úspěšně stane přihlášeným uživatelem.

- **Přihlášený uživatel**

  Přihlášený uživatel má přístup ke všem funkčnostem aplikace. Nepřihlášeným se může stát pomocí odhlášení.

- **Hráč místnosti**

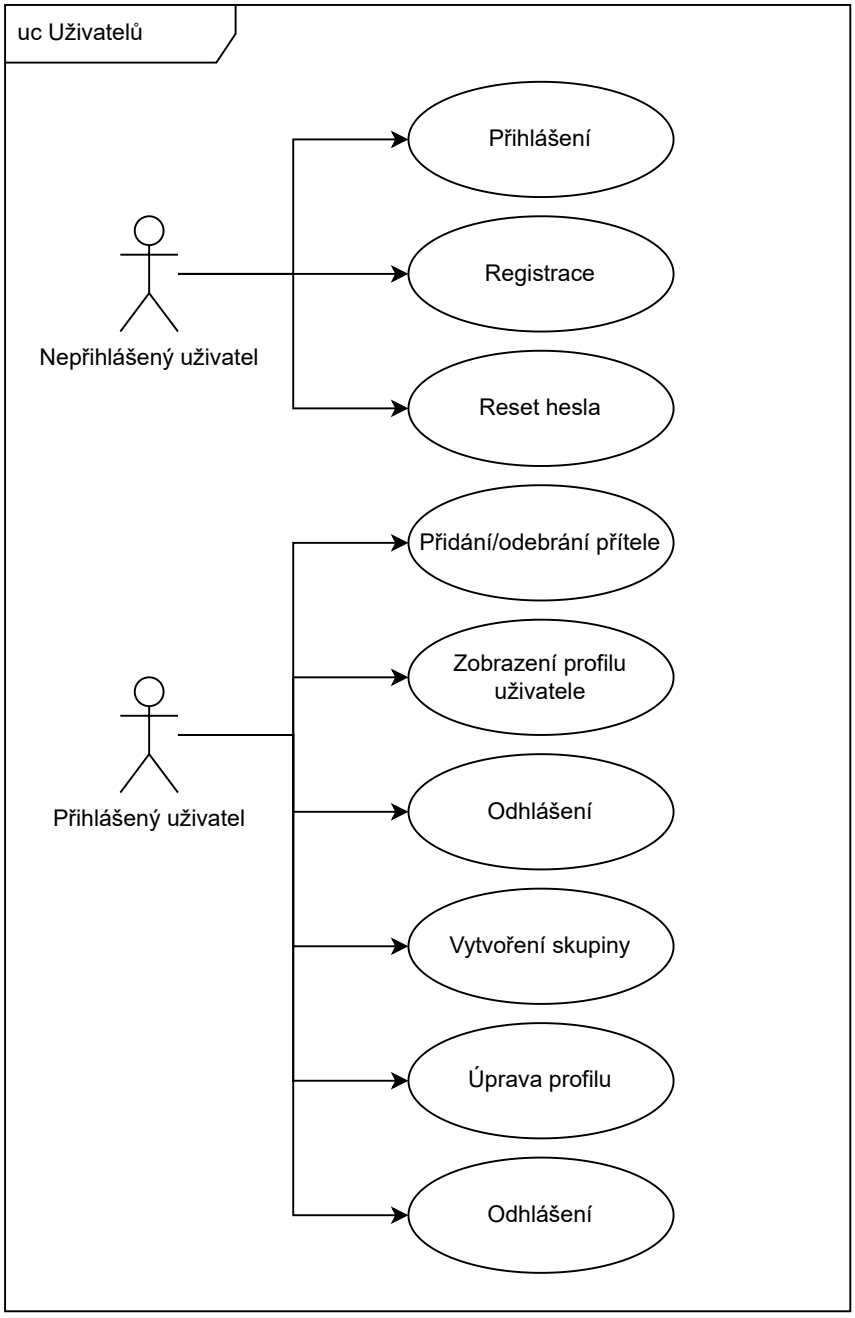
  Hráč místnosti je uživatel, který je vždy přihlášený a zároveň byl administrátorem přidán do dané místnosti.

- **Administrátor místnosti**

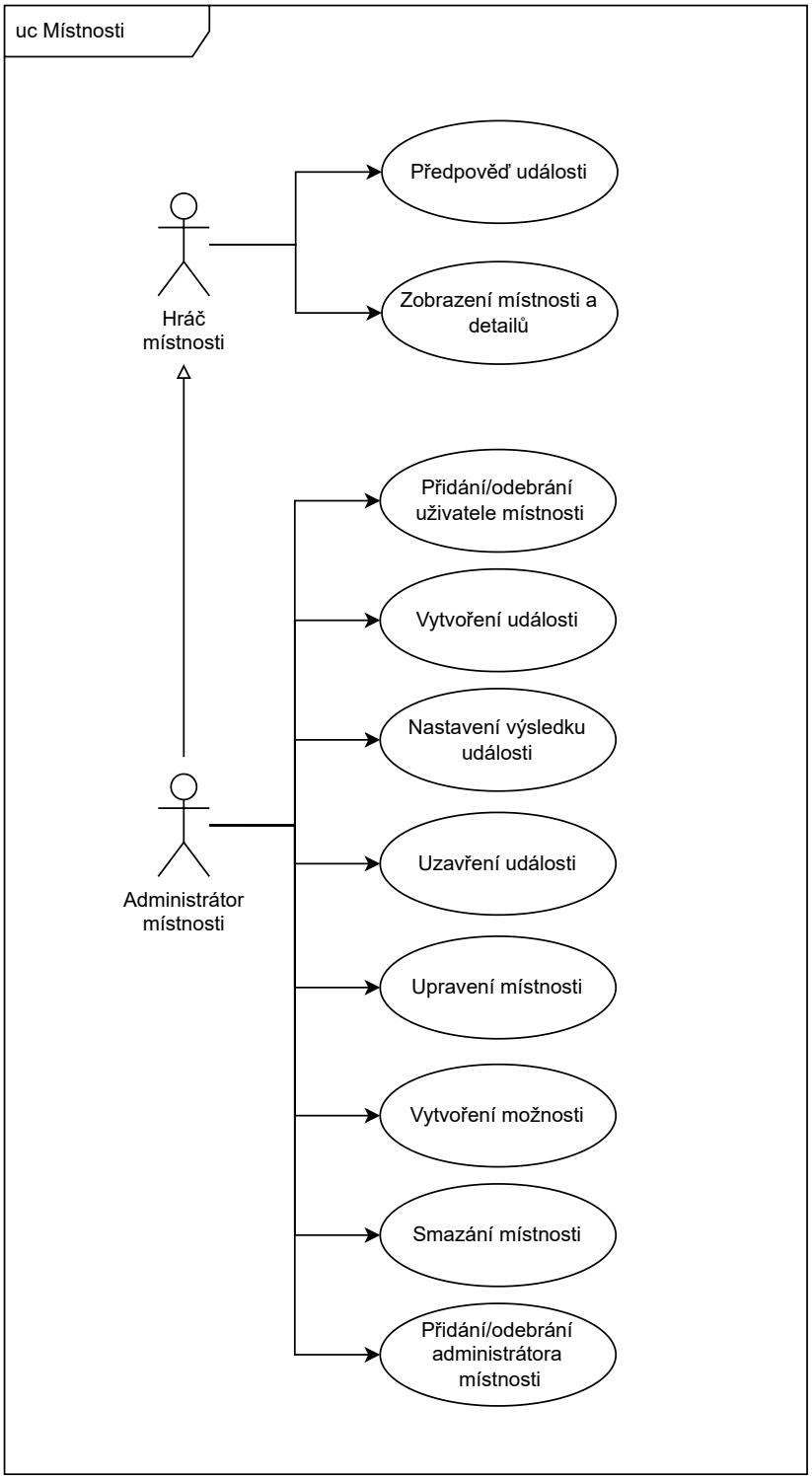
  Administrátor je přihlášený uživatel, který se administrátorem stal pokud vytvořil danou místnost či v ní byl nastaven jako administrátor jiným administrátorem. Zároveň je vždy hráč, tudíž má přístup ke všem funkčnostem jako hráč.

2.3.2 Samotné případy užití

Na obrázku 2.1 se nachází případy užití dle přihlášení uživatele. Obrázek 2.2 zobrazuje případy užití dle práv uživatele v dané místnosti. Popis jednotlivých případů užití není potřeba poskytovat, jelikož lze jejich význam jednoduše pochopit z popisu v diagramu.



Obrázek 2.1 Obecný use case diagram



Obrázek 2.2 Use case diagram místnosti

Kapitola 3
Analýza technologií

Tato kapitola se soustředí na rešerši a analýzu většiny dostupných technologií, architektur a možností výběru pro vývoj aplikace s touto specifikací. První část definuje základní technologie a přístupy. Další dvě části budou diskutovat nad jednotlivými hlavními částmi systému a na závěr bude představena autentizace.

3.1 Základy

Tato sekce je zaměřena na základní architekturu webových aplikací a specifický výběr pro případ tohoto projektu.

3.1.1 Třívrstvá architektura

Třívrstvá architektura označuje jeden ze způsobů návrhu aplikací. Projekt je rozdělen do tří oddělených vrstev, kde každá část plní svůj jasný úkol a komunikuje pouze s předchozí či následující. Obsahuje tyto vrstvy:

- **Prezentační vrstva**
  Tato vrstva je jako jediná viditelná pro uživatele a provádí s ním veškerou interakci. Podporuje minimální funkčnost, tedy nevykonává žádné výpočty či práci s daty. V tomto projektu se jedná o webovou aplikaci s logikou, která komunikuje s aplikační vrstvou.

- **Aplikační vrstva**
  Prostřední vrstva obsahuje hlavní logiku celé aplikace. Komunikuje jak s prezentační vrstvou tak s datovou částí a aktivně zpracovává a reaguje na požadavky z prezentační vrstvy. Provádí také potřebné výpočty. V tomto projektu bude tuto vrstvy zastupovat zdrojový server.

- **Datová vrstva**
  Poslední vrstva tohoto modelu je datová. Tato část má za úkol práci s daty a jejich uložení. Bezpečně zajišťuje všechny potřebné datové operace. V případě tohoto projektu se bude jednat o databázový server.

Hlavní výhodou této architektury je právě rozdělení, jenž přináší možnost jednoduchého rozšíření, přechod na modernější platformu či hardwarové zařízení nebo malé požadavky na výkon zařízení uživatele (všechny složité výpočty a práce se provádí na aplikační vrstvě). V měřítku této práce bude prezentační vrstva nazývaná frontend a aplikační vrstva s datovou backend. Tyto názvy jsou dlouhodobě používané v praxi kvůli přehlednosti a výstižnosti. [11]

3.1.2 API

API je rozhraní, pomocí kterého komunikují a předávají si data dvě oddělené platformy [12]. V tomto případě budou toto rozhraní využívat části frontend a backend. Existuje mnoho možností výběru, pomocí kterého se spolu dorozumívají. Tato sekce se bude věnovat třem nejpoužívanějším dle zdroje [12] a to jsou SOAP, GraphQL a REST.

- **SOAP** je nejstarší ze všech rozhraní, který pro volání využívá pouze značkovací jazyk XML. Jedná se o vysoce standardizovaný protokol s velice obsáhlými zprávami i za předpokladu, že nesou poměrně málo informací. Podporuje jak stavové tak bezstavové volání.

  Pro práci s SOAP protokolem je nutnost znalostí a pochopení všech zahrnutých protokolů, čímž se stává poměrné složitým na naučení. V historii byl používán nejvíce, ale dnes ho nahradily nové, modernější protokoly, avšak stále je používán v praxi např. v bankovních společnostech. Na rozdíl od dvou nadcházející alternativ se jedná o procedurální rozhraní, tedy rozhraní, které spouští vzdálené procedury. [13]

- **REST** je zdaleka nejpoužívanější. Jedná se o datové rozhraní, které na rozdíl používá model klient-server (klient zadává požadavky a server odpovídá). Pro komunikaci používá základní CRUD operace a to přesněji Create, Retrieve, Update, Delete, které jsou implementovány prostřednictvím protokolu HTTP, který mu zároveň umožňuje zasílat návratové stavy. Zprávy mohou být zasílány v jednom z mnoha formátu jako například JSON, HTML, PHP či prostý text, z nichž je formát JSON nejpoužívanější. [14]

- **GraphQL** je na rozdíl od SOAP nejnovější. Jedná se o open source datové rozhraní, kterému se přezdívá následník REST [12]. Bylo navrženo jako rychlé, flexibilní a přívětivé pro vývojáře. Dokáže získat data z více zdrojů v jediném volání. Ovšem tím se stává náročnější pro vývoj backendového serveru. Funguje na principu schémat, kdy se zpráva porovná se schématem a až po úspěšném potvrzení probíhá volání serveru. Tím se stává náročnější na výkon a tvorba schémat je také pracná. [15]

Výslednou volbou pro tento projekt je rozhraní REST API z důvodu jednoduchosti, flexibility, škálovatelnosti a pozitivních osobních zkušeností autora práce.

3.2 Backend

Tato sekce se zabývá technologiemi backendové části aplikace, která se stará o většinu logiky celého projektu a také o komunikaci s databází. Na trhu se nachází nespočet jazyků a frameworků, které umožňují vytvořit backend aplikace jako například jazyky Python, GO, Javascript nebo frameworky Spring Boot či Ruby on Rails. V této sekci je představena volba pro tento projekt a také jsou zde uvedeny další technologie, které se využijí pro vývoj.

3.2.1 Java

Java je high level multiplatformní objektově orientovaný programovací jazyk využívaný na miliardách zařízení. V dnešní době má širokou škálu využití a na jeho základech staví další programovací jazyky jako například Kotlin.

Jeho hlavní výhodou je to, že na rozdíl od konvenčních jazyků jako například C nebo C++ kompilátor nepřekládá ihned do strojového kódu, ale takzvaného byte kódu (.class), který je následně přeložen pomocí JVM (Java Virtual Machine) do strojového kódu daného stroje. JVM je softwarový nástroj, který je dostupný pro většinu softwarových a hardwarových platforem, tudíž pro spuštění kódu na jiném zařízení stačí přesunout byte kód. [16]

3.2.2 Spring Boot

Spring je open-source prostředí pro vývoj webových aplikací, které je založeno na jazyce Java. Bylo vytvořen v roce 2003 za účelem usnadnění tvorby aplikací. Obsahuje množství modulů, mezi než patří například Spring Data, Spring Cloud a Spring Security nebo právě Spring Boot. [17]

Spring Boot je rozšíření prostředí Spring. Specializuje se na rychlé a snadné vytvoření samostatných webových aplikací Spring bez konfigurace, která je pro Spring aplikaci potřebná. Tím vývojář obětuje vysokou konfigurovatelnost za zrychlení tvorby, čímž se prostředí stává ideální pro tvorbu REST API rozhraní. [17]

Mezi hlavní alternativy pro výběr architektury backend serveru patří jazyk Python a framework Ruby on Rails.

Dle [18] je Python interpretovaný, objektově orientovaný programovací jazyk. Je vysoko úrovňový a obsahuje velké množství knihoven, které ulehčují práci. Jeho jednoduchá syntaxe umožňuje rychlé naučení a rapidní vývoj aplikací. Má širokou škálu využití, mezi něž právě patří vývoj backend serverů s pomocí microframeworku Flask. Ten umožňuje pomocí jazyka Python rychlou a lehce rozšiřitelnou implementaci. [18], [19], [20]

Ruby on Rails je platforma pro vývoj webových aplikací založená na jazyku Ruby. Je zaměřena na co nejrychlejší implementaci aplikací pomocí co nejmenšího množství kódu. Dle zdroje [21] je Ruby on Rails založena na principu Convention Over Configuration, což znamená, že vychází z předpokladu, že existuje nejlepší způsob, jak věci implementovat (v některých případech dokonce dokáže odradit od alternativ). [21]

Závěrečnou volbou pro tento projekt se stal framework Spring Boot. Mezi hlavní důvody patří vysoká škálovatelnost, která umožní vývoj většího projektu do budoucna (Flask a Rails jsou preferovány pro vývoj menších aplikací), integrovaná podpora databázových systémů, vysoká efektivita díky použití JVM a rozsáhlá komunita.

3.2.3 Gradle

Gradle je flexibilní nástroj, který se zaměřuje na automatizaci sestavování jakéhokoli programu. Zajišťuje požadované závislosti knihoven a připravuje kód pro kompilaci. Běží na JVM a je velmi konfigurovatelný. [22]

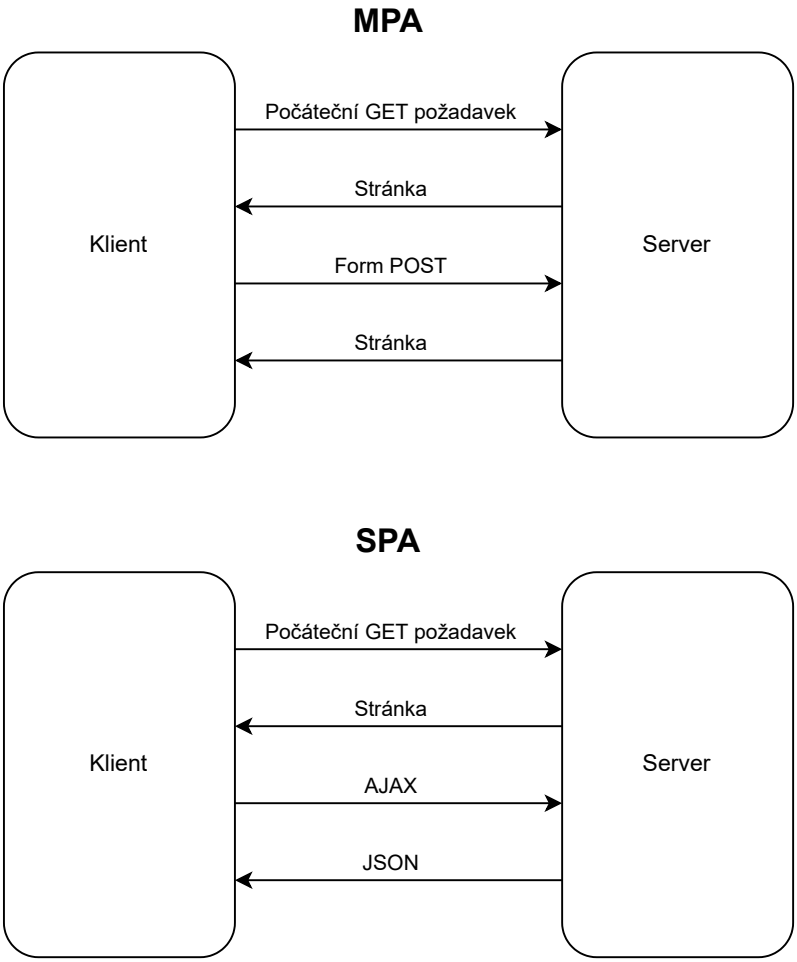
3.2.4 PostgreSQL

Jedná se o open source, relační databázový systém využíván k bezpečnému uložení dat pro širokou škálu aplikací. Je zaměřen objektově, čímž umožňuje rozšířit existující datové typy a vytvořit vlastní. Má skvělou pověst pro svou spolehlivost a bezpečnost. [23], [24]

Pro účely projektu bylo voleno mezi MySQL a právě PostgreSQL, jasným vítězem se stal systém PostgreSQL z důvodu širší nabídky datových typů, rychlejších read-write operací a open source kódem.

3.3 Frontend

Tato sekce rozebere technologie pro tvorbu frontendové části aplikace. V dnešní době se webové stránky vytvářejí pomocí trojice jazyků a to HTML, CSS a JavaScript. Na rozdíl od backendu zde není rozsáhlý výběr jazyků a frameworků, jelikož pro strukturu a grafickou stránku jsou využívány pouze jazyky HTML a CSS. Většina platforem pracující s logikou těchto stránek stojí na základech jazyku JavaScript, čímž se tento jazyk stává naprosto dominantní na tomto trhu.



Obrázek 3.1 Porovnání SPA a MPA. Inspirováno obrázkem [26].

3.3.1 SPA a MPA

U každé webové aplikace je na výběr zda je požadováno, aby byla typu MPA či SPA. Obrázek 3.1 zobrazuje rozdílný přístup těchto dvou typů.

- **MPA** je typ webových aplikací, které se skládají z více HTML souborů, mezi kterými se klient pohybuje. Princip fungování je takový, že při každém požadavku server zašle celou stránku, tedy šablonu, kaskádové styly i skripty. [25]

- **SPA** je typ webových aplikací, které na rozdíl od MPA pracují pouze s jedinou HTML stránkou, u níž pouze mění její obsah bez jakéhokoli znovu zasílání stránek. V praxi fungují tak, že jako reakci na požadavek server zašle danou stránku se všemi náležitostmi. Pokud uživatel interaguje se stránkou tak, že zažádá o nová data, server mu pošle pouze balíček (většinou ve formátu JSON), který skripty zpracují a upraví obsah dané stránky. [25]

Pro tento projekt byl vybrán typ SPA, jelikož se jedná o rychlejší, méně náročný a dnes nejvyužívanější typ.

3.3.2 HTML

HTML je zkratka pro značkovací jazyk, který se používá pro definici struktury webové stránky. Funguje na principů jednotlivých bloků s různými typy neboli tagy. V historii se používal také pro formátování, ale na dnešním trhu ho nahradily kaskádové styly. [27]

3.3.3 CSS

CSS neboli kaskádové styly je jazyk, který umožňuje formátování webových stránek psaných v jazycích HTML, XHTML a XML. Jeho hlavní výhodou je oddělení struktury od vzhledu stránky, čímž zlepšuje přehlednost a ulehčuje práci. Styly mohou být definovány přímo do kódu dané stránky, ovšem v dnešní době se většinou definují v externím souboru. [28]

3.3.4 Bootstrap

Bootstrap je open source framework usnadňující proces vývoje responzivních webových stránek. Je zaměřen na co nejrychlejší tvorbu pomocí skriptů založených na HTML, CSS a JavaScriptu. [29]

V tomto projektu bylo rozhodováno mezi využití nástroje Boostrap a manuální implementací vzhledu aplikace. Nakonec bylo rozhodnuto pro manuální vývoj z důvodu větší přizpůsobitelnosti vzhledu.

3.3.5 JavaScript

JavaScript je objektově orientovaný skriptovací jazyk s dynamickým typováním. Využívá se pro tvorbu dynamických prvků webových stránek. Kód jazyka JavaScript byl uzpůsoben, pro spouštění na zařízení klienta např. přímo v prohlížeči uživatele. O JavaScriptu lze říct, že se jedná o základní stavební kámen dynamických webových stránek, jelikož celý trh stojí přímo na něm. [30]

3.3.6 TypeScript

TypeScript je objektově orientovaný programovací jazyk založen na jazyku JavaScript, oproti kterému je avšak silně typovaný. Jeho vývoj zapříčinilo to, že se jazyk JavasScript začal využívat nejen jako client-side, ale také jako jazyk na serverech. Tím se však rozšiřoval, čímž se stával nepřehledným a také mu chyběly funkce, které by v něm ulehčily vývoj webových aplikací. Proto TypeScript přinesl funkce jako silné typování, kontrolu kódu při kompilaci či silnější objektovou orientaci, které z něj udělaly plnohodnotný server-side jazyk, který se ovšem stále může využít jako client-side. [31]

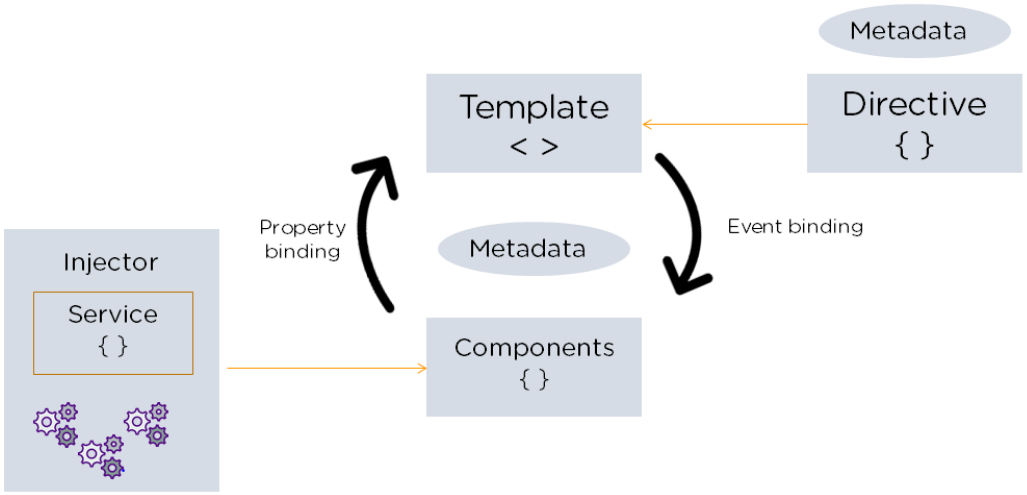
Dnes je brán jako vývojářsky přívětivá alternativa JavaScriptu.

3.3.7 Angular

Angular je open source platforma umožňující vývoj single page aplikací využívající jazyky HTML a TypeScript. Funguje na architektuře založené na samostatných znovupoužitelných komponentách, které se následně skládají do sebe, díky čemuž usnadňuje vývoj a odděluje vzhled od funkčnosti. [32]

Mezi hlavní výhody patří rychlost, množství knihoven či podpora firmy Google [33].

Architektura Angularu, znázorněná na obrázku 3.2, ukazuje, že každá komponenta se skládá z TypeScript třídy (`Component`) definující logiku, HTML šablony (Template) udávající strukturu komponenty a stylů. Dále využívá služby (`Service`), které umožňují přenos dat či logiky mezi



Obrázek 3.2 Angular architektura [34]

jednotlivýni komponentami a `Injector` využívající dependency injection, která umožňuje automatické vkládání závislostí. Mezi `Component` a `Template` existují dvě vazby a to `Event binding` umožňující reagovat na vstup uživatele a `Property binding` umožňující uživatelům interpolovat hodnoty, které jsou vypočteny z dat aplikace, do HTML. [34]

V rámci tohoto projektu bylo dále rozhodováno o alternativách jako knihovna React či framework Vue.js.

React je open-source JavaScript knihovna pro tvorbu uživatelského rozhraní podporovaná společností Meta. Stejně jako Angular využívá znovupoužitelné komponenty, avšak využívá jazyk JavaScript. [35]

Stejně jako Angular a React patří Vue.js mezi takzvaně component-based modely (modely využívající znovupoužitelné komponenty). Jedná se však o progresivní framework, což znamená, že umožňuje webovou aplikaci rozdělit na několik různých částí a ty následně vyvíjet nezávisle na sobě. [36]

Pro tento projekt byl vybrán framework Angular kvůli podpoře jazyka TypeScript, silné dependency injection, nativní podpoře základních funkcí jako routeru či formulářů a v neposlední řadě také pozitivních osobních zkušeností autora práce.

3.4 Autentizace

Jeden z požadavků a logické potřeby tohoto typu projektu je autentizace uživatele. V jakékoli interakci uživatele s aplikací nás zajímá, jaký uživatel tyto akce provedl a zda má na ně práva. Tato funkčnost může být implementována ručně, ale v současné době existují nástroje, které tuto funkčnost nabízejí. Tato část představí celý koncept autentizace včetně nástrojů, které k tomu byly využity.

3.4.1 OAuth 2.0

OAuth 2.0 neboli Open Authorization 2.0 je autorizační protokol, který byl vytvořen, aby umožnil aplikacím přístup ke zdroji dat pomocí autorizace uživatele. Jak napovídá název jedná se o druhou verzi tohoto protokolu, kterou v roce 2012 nahradil původní. V dnešní době se podle zdroje [37] jedná o průmyslový standard pro autorizaci. Primárně byl vytvořen pro web, ale v současné době se využívá i v jiných odvětvích jako například v mobilních či desktop aplikacích. [37]

3.4.1.1 Tokeny

K autorizaci jsou využívány tokeny. Jedná se o datovou strukturu, která reprezentuje oprávnění přístupu k datům s určitou platností. Struktura tokenu není pevně určená, ovšem pro tento projekt byl zvolen token JWT neboli JSON Web Token. Tento token je popsán o sekci níže. OAuth 2.0 smí poskytovat více typů tokenů. Hlavní token je takzvaný Access token, který dává oprávnění přístupu k datům. Authorization code je token, který lze později vyměnit za Access token. Server může také zaslat Refresh token, který má oproti předchozím delší platnost a po jeho prokázání bude poskytnut Access token. [37]

3.4.1.2 Subjekty

Proces autorizace je složitý proces, při kterém spolu komunikuje více samostatný subjektů či systémů. Mezi tyto subjekty patří:

- **Vlastník dat**
  Jedná se o vlastníka dat, který dává oprávnění k přístupu k datům. V kontextu této aplikace se bude jednat o uživatele.
- **Klient**
  Klient je aplikace, která se snažím k daným datům přistoupit.
- **Autorizační server**
  Jedná se o aplikaci, která přijímá požadavky o tokeny. Pokud se klient dokáže autorizovat a vlastník dat schválí právo k přístupu, klientovi bude přidělen token, kterým se bude prokazovat při každém přístupu k datům.

  Většinou tato aplikace nabízí dva koncové body. Takzvaný uživatelský, který zobrazuje rozhraní pro přihlášení uživatele a následně strojový, pomocí kterého si aplikace předají token.
- **Zdrojový server pracující s daty**
  Server, který po úspěšné autorizaci daným tokenem umožní klientovi přístup k datům.

[37]

3.4.1.3 Proces

Tento proces, který končí obdržením dat, může mít více podob, z nichž je níže představen základní takzvaný implicit, ve kterém je autorizačním serverem poskytnut pouze samotný Access token. Existují také složitější procesy, které ovšem nebudou pro tuto práci potřeba.

Tuto posloupnost kroků autorizace, která je reprezentována na obrázku 3.3, nazýváme OAuth 2.0 workflow. Funkce po sobě jdoucích kroků jsou následující:

1. Uživatel na klientovi inicializuje přihlášení například tlačítkem přihlásit.
2. Klient přesměruje uživatele na autorizační server s žádostí o token.
3. Autorizační server ověří správnost požadavku.
4. Autorizační server zobrazí klientovi přihlašovací formulář (stále na stránce serveru).
5. Klient zadá své osobní údaje a přihlásí se.
6. Autorizační server zkontroluje osobní údaje uživatele, tedy nalezne v databázi daného uživatele a porovná hashe hesel.
7. Pokud je uživatel korektně autorizován, tak autorizační server zašle klientovi access token, který ho uloží na bezpečné a dostupné místo (paměť prohlížeče).

8. Klient zašle požadavek o data na zdrojový server společně s tokenem získaným v minulém kroku.
9. Zdrojový server ověří pomocí autorizačního serveru token, tedy zašle požadavek o kontrolu společně s tokenem a získá informaci, zda je token validní.
10. V případě, že je zaslaný token platný, zašle zdrojový server data.
11. Nyní má klient k dispozici požadované data, se kterými smí pracovat.
12. Pokud někdy v budoucnu propadne platnost klientova tokenu a bude mít zájem o další data, bude nucen zaslat požadavek o obnovení tokenu.

[38]

3.4.2 JWT

JWT nebo JSON Web Token je otevřený standard pro bezpečný přenos informací mezi dvěmi stranami. Data přenáší ve JSON formátu, čímž zajistí malou velikost, která umožní tyto tokeny zasílat přes internet a ukládat je do malé paměti prohlížeče klienta. Skládá se ze tří částí a to hlavička, datová sekce a ověřený podpis. V hlavičce jsou uloženy informace týkající se algoritmu, kterým byl token podepsán např. HMAC či RSA a typu tokenu např. JWT. Datová sekce neboli payload se skládá se samotného JSON souboru, který v případě tohoto projektu obsahuje informace o uživateli, který se autorizoval. V podpisové části je uložen řetězec, který se používá k ověření datové sekce. [39], [40]

3.4.3 OpenID Connect

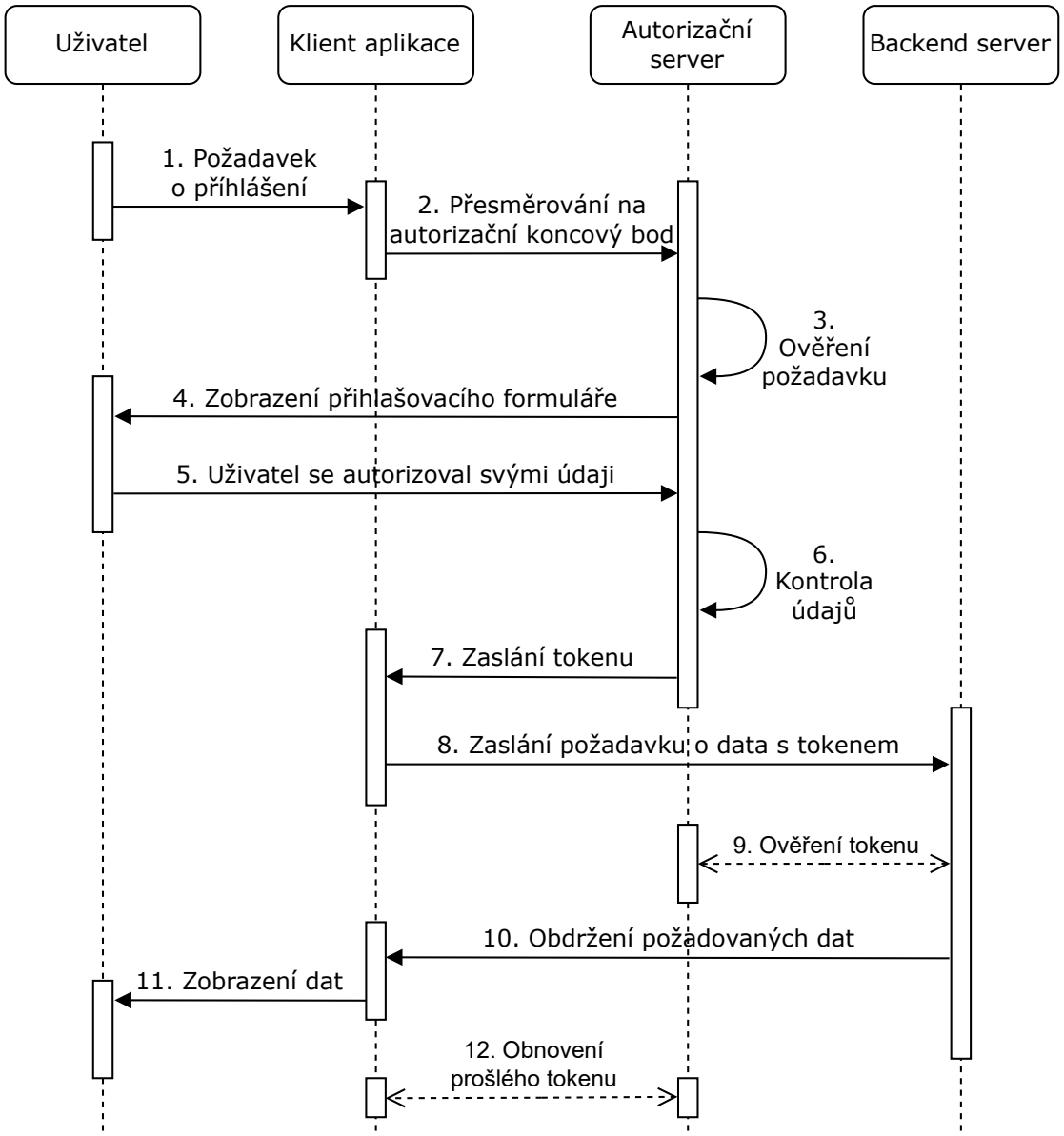
OpenID Connect je rozšíření protokolu OAuth 2.0, který používá dodatečný JWT token tzv. ID token. Tento token standardizuje oblasti, které jinak nechává OAuth 2.0 na volbě vývojáře. Hlavní výhodou použití tohoto tokenu je to, že udržuje dodatečné informace o autorizovaném uživateli. Tím umožní rozšíření těchto dat mezi všechny samostatné části projektu. [41]

3.4.4 Keycloak

Dle stránky [42] se jedná o open-source nástroj s licencí „Apache License 2.0“ pro správu identit a přístupu (IAM), čímž zjednodušuje proces ověřování aplikací. Za jeho tvorbou a udržováním stojí firma Red Hat. První verze byla publikována roku 2014. [42]

Podporuje tři autentizační protokoly a to OpenID Connect, OAuth 2.0 a SAML 2.0. Obsahuje vlastní SSO stránku, která je velice přizpůsobitelná a to proto, aby si jí každý vývojář mohl upravit podle vzhledu svého klienta. Nabízí administrátorskou konzoli, která umožní programátorovi konfigurovat Keycloak v grafickém prostředí. Obsahuje také složitější funkce, které pracují s identitou uživatele, které ovšem nejsou v této práci využity a t tohoto důvodu nebudou v práci představeny. Avšak za zmínku stojí možnost autorizovat uživatele pomocí takzvaných poskytovatelů sociální identity. Jedná se o přihlášení uživatele pomocí účtu na jiné platformě (např. pomocí Google či Stack Overflow účtu). [42]

V tomto projektu bude Keycloak zastupovat pozici autorizačního serveru s OpenID Connect autorizací včetně přizpůsobené přihlašovací SSO stránky, na kterou bude uživatel přesunut, pokud podá podnět k přihlášení.



Obrázek 3.3 OAuth 2.0 workflow. Inspirováno obrázkem [38]

Kapitola 4
Návrh

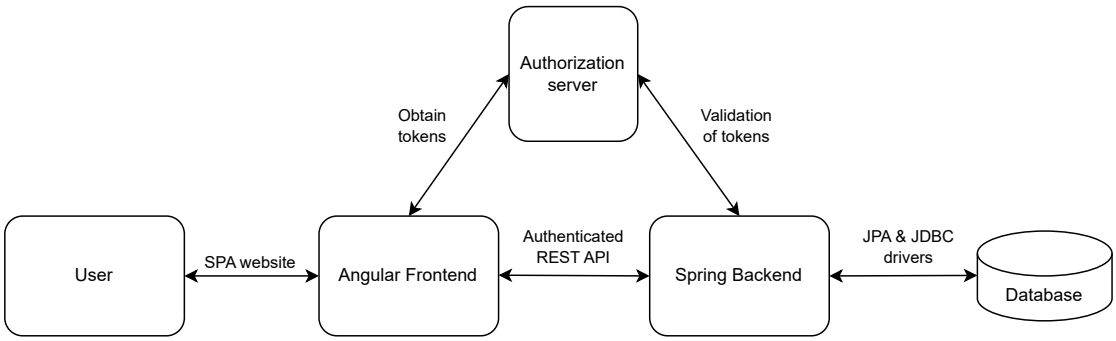
Tato kapitola se věnuje návrhu aplikace. V první sekci je představen celkový návrh aplikace. Další sekce sekce se zaměřují na návrh jednotlivých komponent projektu jako databáze, API a celkového vzhledu aplikace.

4.1 Vysokoúrovňové schéma projektu

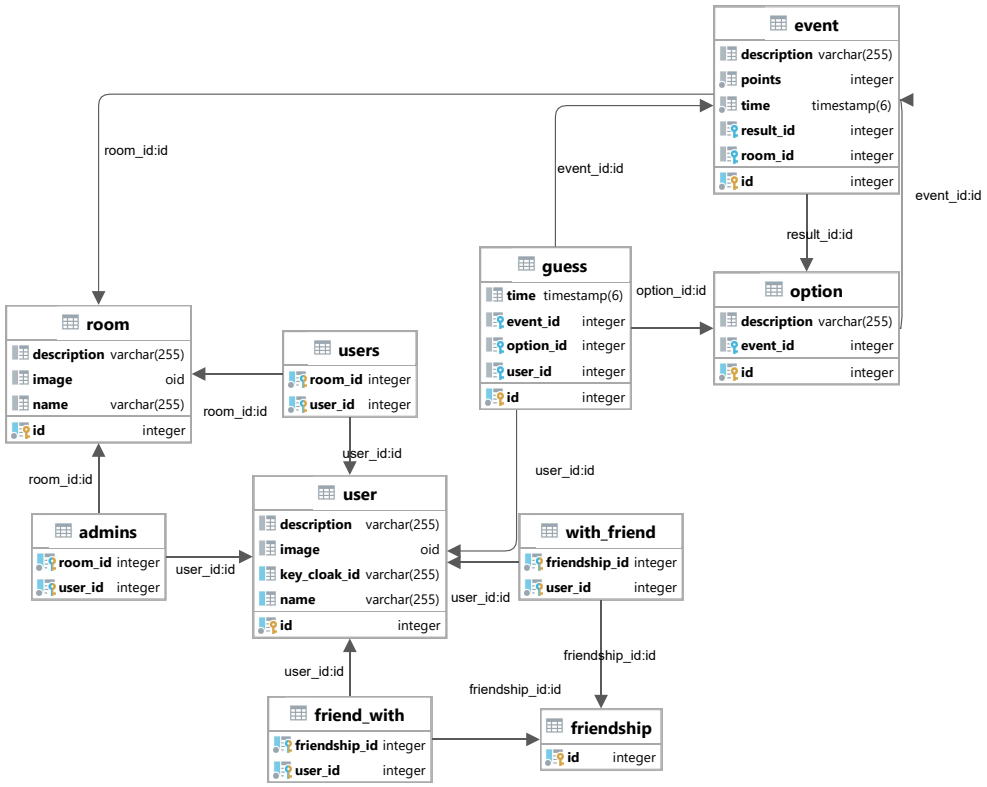
Jelikož byly výše představeny technologie, které se k stavbě takovéhoto projektu využívají, je umožněno vytvořit celkový návrh aplikace od uživatele až po koncovou databázi.

Na obrázku 4.1 je představen celý model projektu. Model bude představen od koncového uživatele. Tedy uživatel přistoupí na náš frontend server stojící na jazyku Angular, který s koncovým uživatelem komunikuje pomocí webové prohlížeče, ve kterém je využívána technologie Single Page Application. Zmíněný frontend server musí odněkud získat data. Z toho důvodu komunikuje s backend serverem, který stojí na Frameworku Spring, pomocí REST API požadavků. Z důvodu bezpečného uložení a rychlou prací s objemnými daty je tento Spring server napojen na databázi pomocí JPA a JDBC ovladačů.

Takováto architektura je již plně funkční avšak jí chybí jakékoliv zabezpečení. Proto je v modelu zakomponován autorizační server, jehož úkolem je zabezpečit komunikaci mezi frontend a backend serverem pomocí využití autorizačních tokenů. Tento server zpracovává požadavky od frontend serveru, ve kterých žádá o tokeny a požadavky od backend serveru, pomocí kterých kontroluje zda jsou tokeny, pomocí kterých se daný klient autorizoval, validní.



Obrázek 4.1 Architektura projektu



■ **Obrázek 4.2** Diragram databáze

4.2 Návrh databáze

Návrh databáze je důležitý proces, jelikož správně vytvořená databáze umožňuje rychlý a snadný přístup k uloženým datům. Pro návrh byly využity databázové diagramy. Tato schémata graficky znázorňují jednotlivé objekty a vztahy, z nichž lze vyčíst a pochopit strukturu celé databáze. Pro vygenerování databázové diagramu byl využit nástroj vizualizace databáze ve vývojovém prostředí IntelliJ IDEA, které je popsáno v sekci 5.1.1. [43], [44]

Model na obrázku 4.2 znázorňuje strukturu databáze. Entita `user` značí uživatele, který smí mít dva různé M:N vztahy k místnosti (`room`) a to administrátor (`admins`) či uživatel/hráč (`users`). Místnost obsahuje různé události (`event`), které mají různé možnosti výsledku (`option`). Uživatelem může být vytvořena předpověď (`guess`) která se vztahuje k danému uživateli, události a možnosti, kterou vybral. Dále je zde definována entita `friendship`, která označuje přátelství. Aby bylo přátelství oboustranné, byly vytvořeny dva M:N vztahy (`with_friend` a `friend_with`).

4.3 Návrh API

Pro komunikaci backend a frontend serveru je využíváno RESTové rozhraní zasílající zprávy ve formátu JavaScript Object Notation zkráceně JSON. Tyto koncové body nacházející se na backend části projektu jsou volány frontend částí projektu. Existují čtyři různé typy volání a to `GET`, který se využívá pro získání informací, `POST`, který se používá pro vytvoření záznamu, `PUT`, který upravuje záznam a `DELETE`, který dává požadavek o smazání záznamu. Seznam návrhů endpointů je rozsáhlý, proto je níže představeno pouze pár důležitých koncových bodů o různých typech.

- **GET /event/{id}**
  Jedná se o koncový bod, který dostane v adrese identifikátor události, pomocí kterého se pokusí danou událost nalézt a při úspěchu o ní vrátí informace. Pokud nenalezne událost, vrátí prázdnou odpověď.

- **GET /guess/{roomId}/user/{userId}**
  Koncový bod, který vrátí všechny předpovědi, které uživatel s identifikátorem userId provedl ve skupině s identifikátorem roomId. Pokud nenalezne žádné předpovědi, vrátí prázdnou odpověď.

- **GET /room/{id}/ranking**
  Tento koncový bod obsluhuje žádosti o bodovací tabulku místnosti s identifikátorem id. Server vypočte danou tabulku a zašle jí ve formátu výpisu 4.1. Tento formát zjednoduší práci na frontend serveru, kde stačí danou tabulku pouze seřadit dle dostupných pravidel a bude připravena k zobrazení.

Výpis kódu 4.1 Formát odpovědi požadavku o tabulku

```
[
   {
      "user": <information about user 1>,
      "points": <points of user 1>
   },
   {
      "user": <information about user 2>,
      "points": <points of user 2>
   },
   //more potencial ranks
]
```

- **POST /room**
  Koncový bod obsluhující tvorbu nové místnosti. Tělo požadavku by mělo obsahovat data v JSON formátu reprezentující danou místnost. Server se pokusí vytvořit daný záznam a do místnosti přidá jako hráče a administrátora uživatele, který požadavek zaslal (informace o uživateli zjistí z autorizačního tokenu). Při úspěchu vrátí data nově existující události. V případě, že se tvorba dané události nezdařila, server vrátí chybu.

- **PUT /event/{id}**
  Jedná se o koncový bod implementující úpravu události. V těle požadavku jsou požadována JSON data obsahující nové hodnoty, na které se má záznam s identifikátorem id upravit. Využívá se při nastavení správného výsledku události a přijímá požadavky pouze od administrátorů místnosti, ve které se událost nachází. Pokud v úpravě nastala chyba (např. nenalezení události dle identifikátoru či uživatel pokoušející se o úpravu nemá dostatečná práva) bude vrácen chybový kód.

- **DELETE /friendship/{id}**
  Koncový bod, který se pokusí smazat přátelství s identifikátorem id.

4.4 Návrh stránek

Vzhled a struktura stránek je důležitá komponenta aplikace. Je to část, se kterou má uživatel největší kontakt, a proto je požadováno, aby byla co uživatelsky nejpřívětivější a přehledná.

4.4.1 Styl

Aplikace byla navrhnuta v minimalistickém stylu. Dle článku [45] od studia Octa se jedná o jednoduchou, přehlednou aplikaci, která neobsahuje rušivé elementy. Bílá barva hraje v tomto stylu klíčovou roli. Pomáhá stránku provzdušnit a vytvořit takzvaný negativní prostor. Dále bývá doplněna občasnými barevnými sekcemi. Mezi hlavní výhody tohoto stylu patří uživatelská přívětivost, lepší soustředěnost uživatele na obsah a lehčí navigace mezi jednotlivými stránkami. [45]

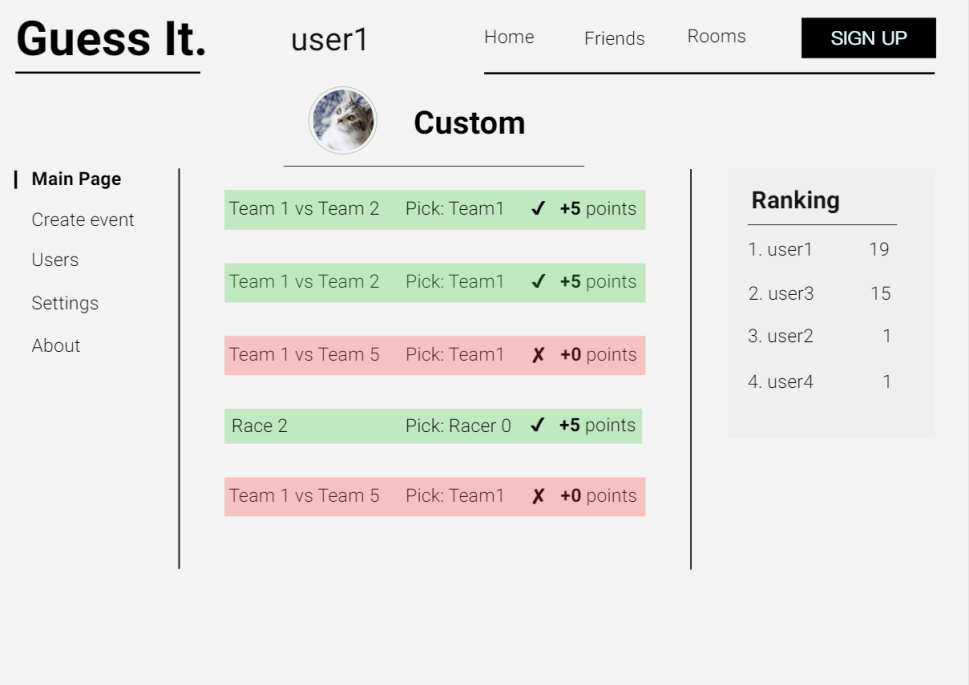
4.4.2 Vzhled a struktura - Wireframe

K vytvoření a postupném vývoji návrhů jednotlivých stránek byly použity wireframes. Wireframe je schéma, které se využívá pro specifikaci vzhledu aplikace a usnadnění komunikace jednotlivých vývojářů. Umožňuje vytvořit návrh prostředí ještě před jakoukoliv implementací. Jedná se o kostru vzhledu a rozložení aplikace. Mezi hlavní výhody patří lehké provádění úprav a vizualizace hned z počátku projektu. [46]

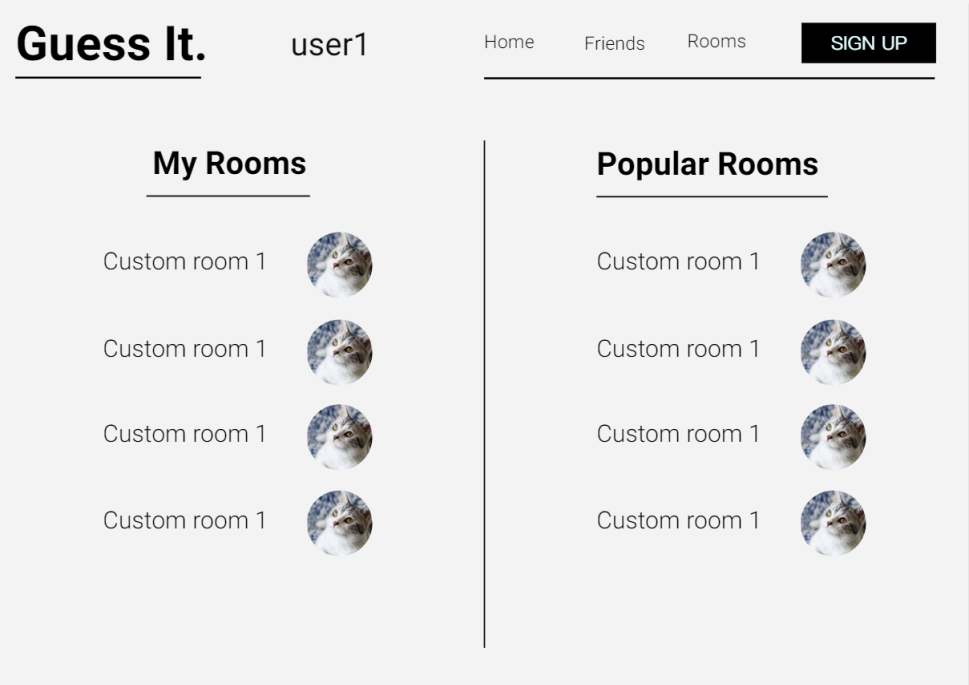
Existuje mnoho platforem, které poskytují možnost vytvoření wireframe schémat, kde mezi nejpoužívanější nástroje patří aplikace Mockitt [47]. Mockitt je jednoduchý avšak velice výkonný nástroj s velikou škálou využití. Je ideální, jak pro tvorbu webových, tak ostatních druhů aplikací. [48]

Obrázek 4.3 zobrazuje grafický návrh stránky místnosti. Na horním okraji se nachází záhlaví, které je zobrazeno na každé stránce aplikace. Pomocí něj se může uživatel dostat na tři hlavní stránky či se přihlásit. Je zde také název aplikace a jméno aktuálně přihlášeného uživatele, pomocí kterého se dostane na profil. Zbytek stránky reprezentuje aktuálně otevřenou místnost. Na levé straně se nachází menu, které umožňuje přepínat prostřední sekci mezi různými podstránkami např. hlavní stránka obsahující seznam událostí nebo stránka, pomocí které může administrátor vytvořit novou událost. Nabídka podstránek se bude měnit dle práv uživatele. V poslední (pravé) části se nachází žebříček reprezentující pořadí a bodové hodnocení hráčů v dané místnosti.

Obrázek 4.4 znázorňuje stránku místností. Je rozdělena na dvě sekce a to místnosti, ve kterých se uživatel aktuálně nachází a druhá část zobrazuje nejnovější místnosti, do kterých smí uživatel nahlédnout (ať z důvodu inspirace či zvědavosti).



Obrázek 4.3 Wireframe návrhu místnosti



Obrázek 4.4 Wireframe návrhu seznamu místností

Kapitola 5
Implementace

Tato kapitola se týká samotné implementace projektu. První sekce představí nástroje, které byly při implementaci využity. Následuje seznámení s implementací autorizace. Poslední dvě sekce se týkají realizace dvou hlavních částí projektu a to backend a frontend serveru.

5.1 Použité nástroje

Pro ulehčení vývoje se v praxi používá mnoho nástrojů pro mnoho věcí. Samozřejmě se dá takováto aplikace naprogramovat i se základními nástroji jako je textový editor, ale využití komplexnějších platforem a rozšíření tuto práci velice zjednoduší. Tato sekce představuje všechny nástroje, které byly na projektu využity.

5.1.1 IntelliJ IDEA

V současné době se většina projektů implementuje v takzvaném vývojovém prostředí či IDE. Jedná se o platformu, která vývojářům značně ulehčí práci s kódem. Pro vývoj jazyku Java a specificky programů ve frameworku Spring se využívá prostředí od společnosti JetBrains IntelliJ IDEA [49]. Tato platforma nabízí velké množství rozšíření neboli pluginů, čímž umožní prostředí dále rozšiřovat a více programátorovi ulehčit práci.

V kontextu toho projektu byly v tomto IDE naprogramovány obě části projektu. Také byly využity další rozšíření tohoto nástroje jako generátory, které umožňují vytvořit počáteční kostru programu, či přehledné grafické zobrazení databáze .

5.1.2 Git

Uložení projektu v počítači daného vývojáře není dostatečně bezpečné z hlediska integrity dat. Proto se využívá systém zprávy verzí neboli VCS, který pomáhá při zaznamenávání změn provedených v sledovaných souborech. Tím zaručuje, že je projekt bezpečně uložen mimo zařízení vývojáře a lze pracovat s jednotlivými verzemi.

Autorem práce bylo rozhodnuto využít verzovací systém Git [50] se serverem gitlab.fit.cvut.cz, který poskytuje Fakulta informačních technologií Českého vysokého učení technického v Praze. Jsou zde vytvořeny tři takzvané repozitáře a to pro uložení zdrojových kódů backend, frontend a Keycloak serveru. Strategie je taková, že při jakékoli implementaci nové funkčnosti bude vývoj probíhat na oddělené větvi a po úspěšném naprogramování a otestování proběhne spojení těchto dvou větví pomocí Merge Requestu, čímž bude zajištěna bezpečná a přehledná práce s kódem.

V backend repozitáři je nastaven skript v souboru **.gitlab-ci.yml**, který je spuštěn v kontejneru vytvořeným softwarem Docker. Tento skript daný projekt sestaví a spustí testy, čímž zajistí, že odevzdaný kód je vždy funkční a bez chyb.

5.1.3 RESTer

Pro vývoj backend serveru bylo použito rozšíření do prohlížeče Firefox s názvem RESTer [51]. Tato jednoduchá aplikace umožní vývojáři zasílat REST API požadavky. Mezi hlavní důvody pro využití specificky tohoto nástroje je možnost výběru předpřipravených hlaviček, historie zaslaných požadavků či kontrola správnosti těla vůči JSON formátu.

V tomto případě byl nástroj využit při testování jednotlivých koncových bodů backend serveru nacházejících se na lokální adrese (localhost). Pro použití toho nástroje musela být vypnuta autentizační ochrana backend serveru, jelikož tento jednoduchý nástroj nenabízí možnost autentizace.

5.2 Implementace autorizace uživatele

Jedním z prvních kroků implementace projektu byla instalace a konfigurace Keycloak serveru, který zastupuje roli autorizačního serveru. Dále došlo k napojením tohoto serveru na další dvě výše zmíněné části projektu.

5.2.1 Keycloak

Keycloak nástroj může být instalován pomocí více způsobů a to jako server, Docker image a operator. V tomto projektu bylo zvažováno použití Docker image, ale nakonec byl využit první z těchto způsobů. Byl stáhnut instalační soubor z oficiálních stránek nástroje a spuštěn. Poté stačilo v Keycloak adresáři zadat do konzole příkaz `bin/kc.bat start-dev`, který tuto nově nainstalovanou aplikaci spustí ve vývojářském režimu. Keycloak sám aktivuje svůj WildFly server, který, pokud není nakonfigurován jinak, poslouchá na adrese **localhost:8080**. Následně byly na této adrese nastaveny přihlašovací údaje admin účtu a server byl připraven ke konfiguraci.

Po přihlášení do administrátorského účty byla vytvořena oblast neboli realm, která zajišťuje správu uživatelů, skupin a rolí. V této oblasti byl následně definován klient s názvem guessit, který bude využit pro přihlašování do této aplikace. Nakonec zbývalo upravit nastavení klienta, aby správně pracoval s ostatními částmi projektu. Byly zde definovány adresy, ze kterých smí Keycloak přijímat požadavky a adresy, na které může vracet uživatele po úspěšném přihlášení či odhlášení.

Nyní bylo potřeba upravit formulářey tak aby graficky ladily ke zbytku aplikace. Postup byl následující. V složce **theme**, nacházející se v kořenovém adresáři, byla vytvořena složka guessit, která definuje nový vlastní styl. Uvnitř této složky byl vytvořen adresář **login**, který definuje, že je potřeba graficky upravit přihlašovací formulář Keycloak serveru (mezi další možnosti patří například administrátorská konzole). Byly vytvořeny další dva adresáře a to **resources** a uvnitř něj **css**. V **resources** byl založen soubor **theme.properties**, který definuje konfiguraci stylů. Jsou v něm nastaveny hodnoty parametrů parent, který udává rodiče, jehož styl bude upraven (aby nebylo nutno začínat na zelené louce) a import, který definuje jeho existující templaty a styly. Dále je zde nastaven parametr styles, který definuje soubory obsahující kaskádové styly, mezi nimiž je definován i nově vytvořený soubor **styles.css**. V tomto souboru byly upraveny prvky jako například pozadí, písmo, barvy jednotlivých částí. Tedy struktura adresáře themes vypadá následovně:

```
themes
└── guessit
    └── login
        └── resources
            ├── theme.properties
            └── css
                └── styles.css
```

Nyní stačí nový styl nastavit v administrátorské konzoli nově vytvořeného realmu a vznikl graficky upravený login formulář (obsahující stránky přihlášení, registrace a vrácení hesla).

5.2.2 Backend

Konfigurace backend serveru tak, aby komunikoval a prověřoval požadavky pomocí autorizační Keycloak serveru byla následující. Do Spring projektu byla přidána závislost `spring-boot-starter-oauth2-resource-server`, která definuje, že server bude využíván jako zdrojový server. Dále byla definována adresa autorizačního serveru v souboru application.properties. Nyní stačilo vytvořit třídu `JWTSecurityConfig`, která obsahuje jedinou funkci `filterChain`, jejíž úkolem je filtrovat každý požadavek a zkontrolovat, zda je klient korektně autorizován. Tato třída byla nastavena tak, aby každý autorizovaný klient mohl zasílat požadavek na jakýkoli koncový bod a až následně si samy koncové body zkontrolují, zda má uživatel pravomoc pro danou akci. Pro příklad koncový bod obstarávající vytvoření události musí ověřit, zda uživatel, který se snaží o vznik nové události, je veden jako administrátor dané místnosti. Toto provede pomocí nalezení uživatele v databázi dle Keycloak identifikátoru, který se nachází v tokenu, jímž se klient autorizoval a jednoduše zjistí, zda se nachází mezi administrátory místnosti.

Společně s OAuth 2.0 resource server závislostí byla přidána její pod závislost Spring Security, díky níž bylo umožněno vytvoření `CORSFilter` třídy, která se stará o CORS filtrování všech požadavků, které na server přicházejí.

5.2.3 Frontend

Do seznamu importů byl přidán modul `KeycloakAngularModule` poskytující třídu `KeycloakService`, která ulehčuje práci s tokeny a komunikací mezi klientem a jednotlivými částmi projektu. Dále byla v tomto souboru definováná funkce `initializeKeycloak`, která je zavolána při každém spuštění Angular aplikace. V této třídě je volán initializér Keycloak služby s konfigurací, ve které jsou specifikovány všechny potřebné údaje (adresa autorizačního serveru, oblast, klíč klienta atd.).

Nyní je tato služba připravena na žádost o přihlášení uživatele. Tento požadavek zašle uživatel pomocí komponenty záhlaví, ve které se nachází tlačítko login. Toto tlačítko zavolá funkci přihlášení třídy `KeycloakService`. Funkce automaticky přesměruje uživatele na přihlašovací stránku a po úspěšném přihlášení uložení tokeny. Tyto uložené tokeny nadále vkládá do všech požadavků odcházejících z daného klienta. Navíc se po úspěšném přihlášení změní tlačítko login na logout, které umožní odhlášení uživatele.

Při každém spuštění klienta si projekt zkontroluje, zda je přihlášen uživatel, zda má již zaevidován účet v databázi a získá si o něm požadované informace. V případě, že je uživatel nově registrován, zašle frontend požadavek na backend server o vytvoření uživatele v databázi. Tento požadavek backend zpracuje a uloží uživatele do databáze společně s klíčem, pod kterým je uložen na autorizačním serveru, jenž získá z tokenu, kterým se uživatel autorizoval. Nyní má frontend aplikace údaje o uživateli, které uloží do instance třídy `SharedVariables`. Tato třída umožňuje jakékoli další třídě přístup k datům, které obsahuje. Dále byla tato třída využita, pokud nějaká komponenta požadovala informaci o aktuálně přihlášeném uživateli.

5.3 Implementace backendu

Tato sekce se věnuje základním schopnostem frameworku Spring, které byly využity při implementaci backend části projektu. Také ujasní strukturu a celkový postup při implementaci.

5.3.1 Schopnosti Springu

Spring je rozsáhlá platforma, která programátorovi ulehčí vývoj backend serveru. Mezi nejvetší výhodu Springu patří, že podporuje Dependency Injection, Inversion of Control vytváří samostatné komponenty nazývané Beans a používá mechanismus Injection.

- **Dependency Injection**
  Jedná se o techniku, která umožňuje předávání závislostí mezi objekty v aplikaci bez využití přímých referencí při sestavování. Pomáhá snížit křížovou závislost a zvyšuje čitelnost kódu.
- **Inversion of Control**
  Návrhový vzor využívající kontejner, který přenáší odpovědnost za řízení toku programu na knihovnu třetí strany (v tomto projektu Spring), která pak volá vytvořený kód.
- **Bean**
  Spring Bean je objekt, který je vytvářen a spravován Spring IoC kontejnerem. Tyto objekty jsou zpravidla definovány jako singletony, které se vytvoří jednou a následně se používají v různých částech aplikace. Existuje více možností jak definovat třídu jako Bean, avšak v tomto projektu je využita metoda pomocí anotací (například `@Repository`, `@Component`, `@Controller`).
- **Injection**
  Jedná se o mechanismus, který umožňuje aplikaci využívat služby nebo objekty, které jsou definovány jinými komponentami aplikace. Spring automaticky vkládá závislosti do Bean. Může se jednat o další Beany nebo jiné objekty či hodnoty.

Spring aplikace využívají anotace (označení začínající na @). Jedná se o formu metadat, která poskytují data o programu. Anotace se používají k poskytování doplňkových informací a tudíž nemají přímý vliv na fungování kódu, který anotují. Příklady použití jsou ukázány v následujících sekcích. V projektu bylo použito mnoho anotací, které jsou z různých balíčků definovaných v souboru **build.gradle**. [52]

5.3.2 Granulace projektu

Logické rozdělení projektu na jednotlivé části, které se využívá i v praxi, zaručí společně se schopnostmi Springu takzvaný low coupling a high cohesion. To znamená, že jsou třídy na sobě co nejméně závislé a zároveň každá třída implementuje pouze jednu funkčnost, čímž zajistí přehledný a lehce rozšiřitelný kód. Struktura je následující:

```
bc_be_blazeda6 ..... kořenový adresář
├── gradle ..... gradle adresář
├── src ..... adresář obsahující zdrojový kód
│   ├── main
│   │   ├── java
│   │   │   └── com.bc_be_blazeda6
│   │   │       ├── config ..... konfigurační třídy
│   │   │       ├── controllers ..... controller třídy
│   │   │       ├── converters ..... converter třídy
│   │   │       ├── domain ..... doménové modely
│   │   │       ├── dtos ..... DTO modely
│   │   │       ├── exceptions ..... výjimky
│   │   │       ├── repos ..... repository třídy
│   │   │       ├── services ..... service třídy
│   │   │       └── BcBeBlazeda6Application
│   │   └── resources ..... zdroje
│   │       └── application.properies
│   └── test ..... testy
├── docs ..... dokumentace projektu
├── gitlab-ci.yml
├── build.gradle
└── settings.gradle
```

5.3.2.1 Domain

V tomto projektu bylo využito Spring JPA. Jedná se o rozšíření Springu, které umožňuje správu relačních dat a práci s relačními databázemi. Část serveru, která pracuje s tímto rozšířením je uložená v podsložce **domain**.

Jsou zde uložené jednotlivé třídy s `@Entity` anotací, která značí třídu reprezentující datovou strukturu jedné relace. `@Id` je označen hlavní číslovací sloupec a navíc je v tomto projektu pro každou třídu vytvořen generátor, který využívá databázovou sekvenci odkud Spring zjišťuje následující index.

Jednotlivé relace jsou na sebe navázány vztahy označenými anotacemi `@OneToOne`, `@ManyToOne`, `@OneToMany` a `@ManyToMany`, které definují vzájemnou násobnost. Tyto relace si Spring sám zpracuje a pokud je požadováno, tak vytvoří celou strukturu databáze.

Výpis kódu 5.1 Ukázka *domain* třídy `Event` (bez komentářů)

```
@Entity
public class Event {

    @Id
    @Column(nullable = false, updatable = false)
    @SequenceGenerator(
            name = "event_sequence",
            sequenceName = "event_sequence",
            allocationSize = 1,
            initialValue = 1
    )
    @GeneratedValue(
            strategy = GenerationType.SEQUENCE,
            generator = "event_sequence"
    )
    private Integer id;
```

```
@Column
private String description;

@Column(nullable = false)
private LocalDateTime time;

@Column(nullable = false)
private Integer points;

@ManyToOne(fetch = FetchType.EAGER)
@JoinColumn(name = "room_id")
private Room room;

@OneToMany(mappedBy = "event")
private Set<Guess> eventGuesss;

@OneToMany(mappedBy = "event")
private Set<Option> eventOptions;

@OneToOne
@JoinColumn(name = "result_id")
private Option result;
```

V ukázce 5.1 se nachází *domain* třída události. Je označená anotací `@Entity`, která předává informaci o tom, že třída označuje jednu tabulku v databázi. Třída obsahuje hlavní proměnnou `id` s anotacemi `@Id`, `@Column`, `@SequenceGenerator` a `@GeneratedValue`. `@Id` značí identifikátor tabulky, `@Column` značí, že tato proměnná třídy reprezentuje sloupec tabulky, `@SequenceGenerator` vytváří vlastní generátor posloupnosti dle dodaných parametrů a `@GeneratedValue` říká této proměnné, že hodnota se má automaticky brát z generátoru. Dále tato třída obsahuje sloupce popis, body, čas uzavření a také vazby na další třídy definované anotacemi `@ManyToOne`, `@OneToMany` atd.

5.3.2.2 DTO (Data Transfer Object)

Tyto třídy označují strukturu jakou by měly mít JSON požadavky a odpovědi. Mají tvar velice podobný *domain* třídám, avšak nenesou žádnou logiku, pouze slouží jako úložiště dat. Spring dokáže automaticky přeložit JSON soubor na instanci dané třídy a zpět. Tyto třídy byly využity v *controllerech*, které se starají o příjem a odeslání JSON hodnot. Jsou v nich také využity anotace `@JsonIgnore` a `@JsonIgnoreProperties`, které informují serializér a deserializér o tom, že má při převodu ignorovat dané proměnné či podproměnné třídy.

Za zmínku v této sekci stojí *DTO* s názvem `RankingDTO`, které reprezentuje bodovací tabulku. Kolekce objektů této třídy je zasílána jako odpověď na požadavek o tabulku místnosti, ve které se uživatel aktuálně nachází.

5.3.2.3 Converters

Každá třída implementuje čtyři funkce, které převádí, jak samostatné, tak kolekce *domain* modelů na *DTO* modely a zpět. Tento převod je důležitý kvůli tomu, že *controller* třídy získávají a jako návratovou hodnotu navracejí *DTO* objekty, zatímco *service* a *repository* třídy pracují s *domain* objekty. Tyto jednotlivé funkce mají své největší využití v *controllerech*. Pro převod využívají knihovnu ModelMapper, která implementuje automatické namapování proměnných mezi různými třídami.

5.3.2.4 Controllers

Třídy označené anotací `@RestController` definují jednotlivé koncové body, reagují a odpovídají na požadavky. Jejich logika by měla být pouze v zavolaní dané funkce v *service* třídě, avšak některé řeší i práva uživatele. Pro získání parametrů požadavku byly využity anotace `@PathVariable`, která získá hodnotu z adresy, na kterou by požadavek zaslán, a `@RequestBody`, která informuje Spring kontejner, aby se automaticky pokusil vytvořit daný objekt z těla požadavku. Tento objekt je vždy instance *DTO* třídy, čímž odděluje vnější strukturu koncových bodů od vnitřní struktury doménového modelu. Toto oddělení se využívá kvůli zvýšení bezpečnosti a flexibility aplikace.

Výpis kódu 5.2 Ukázka *controller* třídy `RoomController` (bez komentářů)

```
@RestController
public class RoomController {

    @Autowired
    private RoomConverter roomConverter;

    private final RoomService roomService;

    private final UserService userService;

    RoomController(RoomService roomService, UserService userService) {
        this.roomService = roomService;
        this.userService = userService;
    }

    @GetMapping("/room")
    public Collection<RoomDTO> lastTenRooms() {
        return roomConverter.fromModelMany(roomService.findTop10ByOrderIdDesc());
    }

    @GetMapping("/room/{name}")
    public Collection<RoomDTO> allRoomsByUserName(@PathVariable String name) {
        return roomConverter.fromModelMany(roomService.readAllByUserName(name));
    }

    @GetMapping("/room/id/{id}")
    public RoomDTO allRoomsById(@PathVariable Integer id) {
        return roomConverter.fromModel(roomService.readAllById(id).get());
    }

    @GetMapping("/room/{id}/ranking")
    public List<RankingDTO> getRoomRanking(@PathVariable Integer id) {
        return roomService.getRanking(id);
    }

    @PostMapping("/room")
    public RoomDTO newRoom(@RequestBody RoomDTO newRoom) throws EntityStateException {
        Room roomModel = roomConverter.toModel(newRoom);

        roomModel.setId(0);
        this.roomService.create(roomModel);

        roomModel = this.roomService.findTopByOrderByIdDesc();
```

```
        return roomConverter.fromModel(roomModel);
    }
}
```

Ukázka 5.2 obsahuje třídu `RoomController`. Tato třída implementuje `@RestController` orientující se na místnosti. Každá funkce označuje jeden koncový bod, který leží na relativní adrese zadefinované v anotaci `@<metoda>Mapping`. Například lze v této třídě nalézt koncové body implementující zaslání dat ohledně místností uživatele dle jeho jména, tabulky místnosti či vytvoření nové místnosti.

5.3.2.5 Services

Jedná se o třídy, které definují hlavní logiku aplikace. Jsou označeny anotací `@Component`, díky níž z ní Spring vytvoří Bean a zapojí do ní všechny potřebné závislosti. Takovýto Bean je připraven pro injeknutí do další Beany. V daném případě se injektuje do daného *controlleru*.

5.3.2.6 Repository

Soubory s anotací `@Repository` jsou třídy, které definují veškerou práci s databází. Rozšiřují typovou třídu `JpaRepository`, která již definuje základní operace s databázemi. Další operace byly přidány pomocí anotace `@Query` či pomocí definice jednoduchých operací dle jména funkce, které si Spring JPA sám převede na databázové skripty.

Výpis kódu 5.3 Ukázka *repository* třídy `GuessRepository` (bez komentářů)

```
@Repository
public interface GuessRepository extends JpaRepository<Guess, Integer> {

    @Query(nativeQuery = true, value = "SELECT g.* FROM guess g JOIN event e ON (e.id
        = g.event_id) JOIN users u ON (u.user_id = g.user_id) where u.room_id=:roomId
        and u.user_id=:userId and e.room_id=:roomId")
    Collection<Guess> readAllByRoomIdAndUserId(Integer roomId, Integer userId);

    Guess findByUserIdAndEventId(Integer userId, Integer eventId);

    Guess findTopByOrderByIdDesc();

    Guess readAllById(Integer guessId);
}
```

Ukázka 5.3 obsahuje třídu `GuessRepository`, která pracuje s částí databáze týkající se předpovědí. Funkce reprezentují jednotlivé dotazy. První funkce s anotací `@Query` v ní přímo definuje manuálně napsaný dotaz na databázi, zatímco zbylé tři funkce tvoří dotaz ze struktury názvů funkce využitím Spring JPA.

5.3.2.7 Config

V podsložce **config** se nachází soubory, které definují konfiguraci autentizace, bezpečnosti a celkového nastavení Spring frameworku. Nachází se zde tři soubory. **DomainConfig** je soubor, který obstarává a definuje jednotlivé *domain* a *repository* třídy společně s řízením transakčních operací. **CORSFilter** obsahující funkci `doFilter`, kterou prochází každý požadavek, definuje pravidla podle kterých rozhoduje, který požadavek přijmout a který zamítnout. `JWTSecurityConfig` je třída, která definuje jak musí být uživatel autorizován, aby byly jeho požadavky vyřízeny.

5.3.2.8 Ostatní soubory

- Soubor **BcBeBlazeda6Application** je hlavní soubor celého projektu, který spouští Spring aplikaci.
- V **application.properties** se definují hodnoty, které využívá Spring nebo sám vývojář. Nachází se zde všechny konstanty projektu z důvodu zjednodušení práce s nimi. Navíc jsou v tomto souboru definovány hodnoty jako přístupové údaje k databázi, Spring JPA nastavení či adresa autentizačního serveru aplikace.
- V **build.gradle** je sestavovací skript, ve kterém je nakonfigurováno nastavení nástroje Gradle. Je zde také možné nalézt všechny použité závislosti projektu.
- **settings.gradle** je soubor, ve kterém se nachází důležité konstanty pro sestavování projektu například název či autor.
- Podsložka **exceptions** obsahuje všechny výjimkové třídy, které byly vytvořeny z důvodu, že neexistovala výjimka, která by správně reprezentovala danou chybu aplikace.

5.3.3 Postup implementace

Základ projektu byl vygenerován využitím generátoru Spring Boot aplikací, který se nachází ve vývojovém prostředí IntelliJ IDEA. Vývoj začal vytvořením jednotlivých datových struktur přesněji *domain* a *DTO* tříd a kontrolou, zda se korektně vytváří databáze. Následně byly ke každé *domain* třídě vytvořeny *service*, *repository* a *controller* třídy. V každém *controlleru* byly vytvořeny dva koncové body a to GET pro získání všech daných objektů v databázi a POST pro vytvoření nového. Proběhlo napojení na *service* a *repository* třídy. Tato základní kostra byla otestována pomocí nástroje RESTer, který umožnil ručně zasílat požadavky a kontrolovat, zda mají správný výstup. Po otestování a opravě chyb byla aplikace připravená pro připojení k autorizačnímu serveru, který popisuje sekce 5.2.2.

Po tomto kroku započal vývoj přední části projektu. Postupným rozšiřováním této části projektu vznikaly nové typy požadavků, které musel být backend schopen zpracovat. Tyto koncové body byly tedy implementovány a vždy otestovány buď přímo v klientovi či pomocí dočasného vypnutí autorizace a zasláním daného požadavku pomocí RESTeru.

5.4 Implementace frontendu

Tato sekce ujasňuje, jak byla přední strana projektu implementována pomocí definice struktury s ukázkami kódů. Také se v této sekci nachází popis postupného vývoje této webové aplikace.

5.4.1 Granulace projektu

Stejně jako v backend části projektu je velice důležité rozdělení na jednotlivé logické celky tak, aby byl projekt přehledný a co nejjednodušeji rozšiřitelný. Toto zaručí programátorovi rychlou a co nejbezproblémovější implementaci aplikace bez zbytečných zásek a nutnosti složitého znovu pochopení projektu při návratu k němu. Struktura projektu je následující:

- bc_fe_blazeda6 — kořenový adresář
  - documentation — dokumentace projektu
  - src — adresář obsahující zdrojový kód
    - assets — adresář obsahující dodatečné assety
    - app
      - components — komponenty
      - models — modely
      - services — *service* třídy
      - app.component.css
      - app.component.html
      - app.component.ts
      - app.const.ts
      - app.module.ts
      - sharedVariables.ts
    - favicon.ico
    - index.html
    - main.ts
    - styles.css
  - angular.json
  - package.json
  - package-lock.json
  - tsconfig.app.json
  - tsconfig.json
  - tsconfig.spec.json

5.4.1.1 Components

Tato složka obsahuje všechny komponenty využívané v naší aplikaci. Každý komponent má svou vlastní podsložku, ve které se nachází tři soubory s koncovkami **.ts** **.html** a **.css**, jejichž význam byl představen v sekci 3.3.7. Aby mohla být daná komponenta použita, musí být deklarována v hlavním souboru aplikace **app.module.ts**. Většina těchto komponent implementuje třídu `OnInit`, což značí, že při inicializaci dané komponenty bude vždy zavolána funkce `ngOnInit`. V této funkci většinou komponenty vykonávají funkce potřebné k zobrazení komponenty. Mezi něž patří např. získání informací o místnostech daného uživatele z backend serveru či přizpůsobení vzhledu stránky dle autorizace uživatele.

Výpis kódu 5.4 Ukázka **.ts** souboru komponenty zobrazující tabulku (bez komentářů)

```
@Component({
  selector: 'app-ranking',
  templateUrl: './ranking.component.html',
  styleUrls: ['./ranking.component.css']
})
export class RankingComponent implements OnInit {

  @Input() roomId: number

  ranking: Ranking[] = []

  errorMessage = ''

  constructor(private roomsService: RoomsService,
              private router: Router) {
  }
```

```
  ngOnInit() {
    this.roomsService.getRanking(this.roomId).subscribe({
      next: data => {
        this.ranking = data

        this.ranking.sort((a, b) => {
          if (a.points < b.points) {
            return 1
          }
          return -1
        })
      },
      error: error => {
        this.errorMessage = 'Error getting ranking. Try again later...'
        console.log('Error getting ranking.', error)
      }
    })
  }

  navigateToUser(userName: string) {
    this.router.navigate(['/user/' + userName]);
  }

  showImage(image: File): string {
    return '' + image;
  }
}
```

Kód 5.4 implementuje TypeScript třídu komponenty zobrazující tabulku hráčů. Třída implementuje rozhraní `OnInit`, které bylo zmíněno o odstavec výše. Obsahuje tři proměnné. První proměnná s `@Input`, která označuje, že hodnota této proměnné se má získat z rodiče, drží informaci o tom, ve které skupině se tabulka nachází. Druhá proměnná `ranking` obsahuje tabulku k zobrazení a třetí proměnná `errorMessage` udržuje chybovou zprávu. Při inicializaci této komponenty se automaticky zavolá funkce `ngOnInit`, jejíchž úkolem je získat tabulku pomocí *service* třídy a seřadit jí. Pokud tento proces proběhne úspěšně, tak se seřazená tabulka zobrazí uživateli na stránce. V opačném případě se zobrazí chybová zpráva. Funkčnost funkce `navigateToUser` je jednoduchá a to pouhé přesměrování na profil uživatele dle obdrženého jména. Poslední funkce zvaná `showImage` přiloží souboru hlavičku, čímž ho umožní zobrazit uživateli.

5.4.1.2 Models

Složka s názvem **models** obsahuje třídy definující modely aplikace. Tyto modely se používají při komunikaci s backend serverem a přeložení požadavku ve formátu JSON na objekt dané třídy a zpět. Mají identickou strukturu jako *DTO* třídy na backend serveru a také plní stejný úkol (úschova dat). Navíc má každý model ve svém souboru nadefinován takzvaný adaptér, který značně ulehčí vytvoření objektu a zajistí jednoduchou úpravu kódu, pokud by v budoucnu došlo k úpravě modelu.

Výpis kódu 5.5 Ukázka modelové třídy `Event` a jejího adaptéru `EventAdapter` (bez komentářů)

```
export class Event {
  id: number
  description: string
  time: Date
  points: number
  room: Room
```

```
  guesses: Guess[]
  options: Option[]
  result: Option

  constructor(id: number, description: string, time: Date, points: number, room: Room,
      guesses: Guess[], options: Option[], result: Option) {
    this.id = id;
    this.description = description;
    this.time = time;
    this.points = points;
    this.room = room;
    this.guesses = guesses;
    this.options = options;
    this.result = result;
  }
}

@Injectable({
  providedIn: "root",
})
export class EventAdapter implements Adapter<Event> {
  adapt(item: any): Event {
    return new Event(item.id,item.description, item.time , item.points, item.room,
        item.guesses,item.options, item.result);
  }
}
```

Na ukázce 5.5 se nachází třída Event, která definuje model události. Jejím jediným úkolem je poskytovat strukturu, do které se namapuje odpověď z API. Obsahuje úplný konstruktor, který naplní všechny její proměnné zadanými hodnotami. Navíc je v této třídě definován adaptér, který ulehčuje vytvoření objektu po získání dat z API.

5.4.1.3 Services

Třídy nacházející se v složce **service** definují hlavní logiku aplikace. Jsou volány z jednotlivých komponent, čímž zjednodušují jejich logiku a zvyšují přehlednost práce s funkcionalitami, které tyto třídy implementují. Mezi jejich úkoly patří například zasílání požadavku na backend server, výpočty či kontroly stavu.

Výpis kódu 5.6 Ukázka servisní třídy GuessService (bez komentářů)

```
@Injectable({
  providedIn: 'root'
})
export class GuessService {

  constructor(private http: HttpClient) {
  }

  getGuessesByUserIdAndRoomId(userId: number, roomId: number) {
    return this.http.get<Guess[]>(Const.url + 'guess/' + roomId + '/user/' + userId)
  }

  setOption(userId: number, eventId: number, optionId: number) {

    const httpOptions = {
      headers: new HttpHeaders({'Content-Type': 'application/json'})
```

```
    };

    const user = {id: userId}
    const event = {id: eventId}
    const option = {id: optionId}

    const data = {
      user: user,
      event: event,
      option: option,
      time: new Date()
    };

    return this.http.put<Guess>(Const.url + 'guess', data,
      httpOptions)
  }
}
```

Výpis 5.6 znázorňuje třídu `GuessService`, která se stará o složitější operace týkající se předpovědí. V jejím konstruktoru je inicializován `HttpClient`, pomocí kterého se ve funkcích zasílají požadavky na backend server. Tato *service* třída implementuje dvě funkce. První se týká získání předpovědí dle identifikátoru uživatele a místnosti. Druhá nastavuje předpověď na možnost s identifikátorem `optionId` na událost s identifikátorem `eventId` uživatelem s identifikátorem `userId`. Obě tyto funkce vrací hodnoty ve tvaru `Observable`, což znamená, že funkce se provede až po zavolání metody `subscribe`, která uvnitř svého těla definuje, co se má stát po obdržení dat.

5.4.1.4 Ostatní soubory

Soubor **app.module.ts** se dá nazvat jako hlavní soubor aplikace. Jsou v něm definovány všechny komponenty projektu, importované moduly, exportované moduly, funkce inicializující připojení k autorizačnímu serveru a další důležité informace.

Tři soubory **app.component** s koncovkami **.ts**, **.html** a **.css** definují hlavní komponentu, která se zobrazí při otevření webové aplikace. V souboru obsahujícím kostru aplikace jsou definovány komponenty `app-header`, což je záhlaví definované ve složce components a `router-outlet`, který rozhoduje jaké má místo sebe dosadit komponenty dle dané adresy. Pravidla, podle kterých zobrazuje komponenty, se nachází v souboru **app.module.ts** v proměnné routes.

Třída `SharedVariables` nacházející se v souboru **sharedVariables.ts** slouží jako dynamické úložiště důležitých informací v aplikaci jako je identifikátor či jméno aktuálně přihlášeného uživatele. Funguje tak, že dané hodnoty jsou ihned po jejich získání do této třídy uloženy. K daným hodnotám smí ostatní části aplikace přistupovat, čímž se zaručí, že budou tyto informace zpropagovány do všech částí aplikace. Alternativou této třídy je soubor **app.const.ts**, který slouží jako úložiště statických konstant např. adresy backend serveru.

Soubory v kořenovém adresáři končící koncovkou **.json** jsou soubory poskytující konfiguraci kompletního prostředí projektu. Tuto konfiguraci využívá přímo Angular při sestavování a spouštění aplikace.

Adresář **assets** obsahuje položky zobrazující se v jednotlivých komponentech jako například výchozí obrázek skupiny či uživatele.

Obrázek **favicon.ico** definuje logo stránky. Soubor **index.html** je hlavní stránka Angular aplikace, na které je zobrazen `app-root`. V souboru **styles.css** jsou definovány její styly a **main.ts** je Typesript třída, která inicializuje spuštění celé aplikace.

5.4.2 Postup implementace

Základ projektu byl vygenerován pomocí generátoru Angular aplikací nacházejícím se v rozšíření „Angular and AngularJS“ vývojového prostředí IntelliJ IDEA, které v něm ulehčuje vývoj Angular aplikací. Navíc byla k tvorbě jednotlivých komponent využita schémata nacházející se v tomto rozšíření. [53]

Vývoj začal vytvořením komponenty záhlaví, která obsahuje hlavní naváděcí menu a inicializaci přihlášení uživatele, a komponenty domovské stránky. Následně byla v souboru **app.component.html** definována komponenta záhlaví a `router-outlet`, čímž byla vytvořena webová stránka, která má vždy zobrazené záhlaví a obsah, který se dynamicky mění dle adresy. Právě `Router-outlet` se stará o výběr komponenty dle takzvaných routes, které jsou definovány v hlavním souboru **app.module.ts**. Následně byl projekt připojen k již existujícímu autorizačnímu serveru a backend serveru dle sekce 5.2.3.

Po úspěšném napojení byly tvořeny nové komponenty, které byly přidávány do `routes` a postupně rozšiřovaly aplikaci až po první funkční prototyp. Následně došlo k refaktorizaci kódu, opravení chyb a upravení aplikace, které mělo za cíl zlepšit uživatelskou přívětivost.

Posledním krokem bylo nastavení vzhledu jednotlivých komponent. Toto bylo provedeno pomocí manuální implementace kaskádových stylů s přepínáním dle šířky prohlížeče.

Kapitola 6
Testování a dokumentace

Tato kapitola představí dvě důležité součásti vývoje softwaru. První se týká testování, ve které jsou představeny různé druhy testů, které projekt podstoupil. Druhá ujasní jak a čím byla aplikace zdokumentována.

6.1 Testování

Tato sekce se zabývá testováním projektu. Na backend serveru byly naimplementovány automatizované testy různých typů, zatímco na frontend serveru byly provedeny uživatelské testy.

6.1.1 Backend

Na backend serveru byly napsány automatizované testy. Jedná se o typ testů, které se definují předem a následně ověřují kód automaticky bez zásahu vývojáře. Jejich hlavním cílem je ověření správnosti funkcionality aplikace. K testování je používán framework JUnit a závislost Spring Boot Starter Test. Testy byly spuštěny při každém sestavení aplikace a nahrání na server. [54]

Konkrétněji byly napsány testy typu unit. Jedná se o testy, které prověřují funkčnost samostatných tříd a funkcí nezávisle na zbytku projektu. Jelikož není při tomto testování načten celý kontext Spring aplikace, je nutno volané funkce nacházející se v jiných částech projektu simulovat (anglicky mock) a nastavit jim předem výstup. [54].

Tento typ testů byl napsán pro všechny části kódu, které provádějí složitější operace, které se dají otestovat (například nemá smysl testovat, zda *DTO* třída funguje korektně, pokud obsahuje pouze třídní proměnné a get/set funkce). V rámci tohoto projektu byly otestovány *service* a *controller* třídy.

Celkově bylo napsáno 66 testů pokrývající každou funkci *service* a *controller* tříd. Tyto testy se nachází na relativní cestě **src/test** v kořenovém adresáři backend části projektu.

6.1.2 Frontend

Na frontend části projektu byly provedeny uživatelské testy rozhraní aplikace s potenciálními uživateli a osobami se zkušenostmi z tohoto oboru. Jedná se o typ testování, které umožní pomocí interakce uživatele s aplikací zjistit slabé stránky řešení. [55]

6.1.2.1 Scénář

Aby byla korektně otestována celá aplikace je vhodné vytvořit scénář, dle kterého bude tester provádět operace. Scénář vypadá následovně:

- Registrace pomocí libovolných údajů
- Úprava profilu (změna popisku a obrázku)
- Přidání existujícího uživatele do přátel
- Vytvoření místnosti
- Přidání uživatele z přátel a dle jména do místnosti (jeden z přátel, jeden dle jména)
- Nastavení jednoho z uživatelů jako administrátora
- Upravení informací o skupině (změna jména, popisku a obrázku)
- Vytvoření událostí dle volby testera
- Přidání libovolných možností do událostí
- Výběr možností u událostí
- Uzavření událostí (pomocí tlačítka a časově)
- Nastavení správných možností událostí
- Kontrola bodovací tabulky
- Odhlášení z aplikace

Tester prováděl tyto kroky bez jakéhokoli předchozího poskytnutí informací o navigaci v aplikaci, čímž se umožnilo zjištění intuitivnosti řešení. V průběhu provádění těchto kroků byl tester sledován a byly zaznamenávány informace o jeho interakci. Následně byly tyto informace zpracovány a byl proveden rozhovor s testerem, čímž vznikl závěr testování.

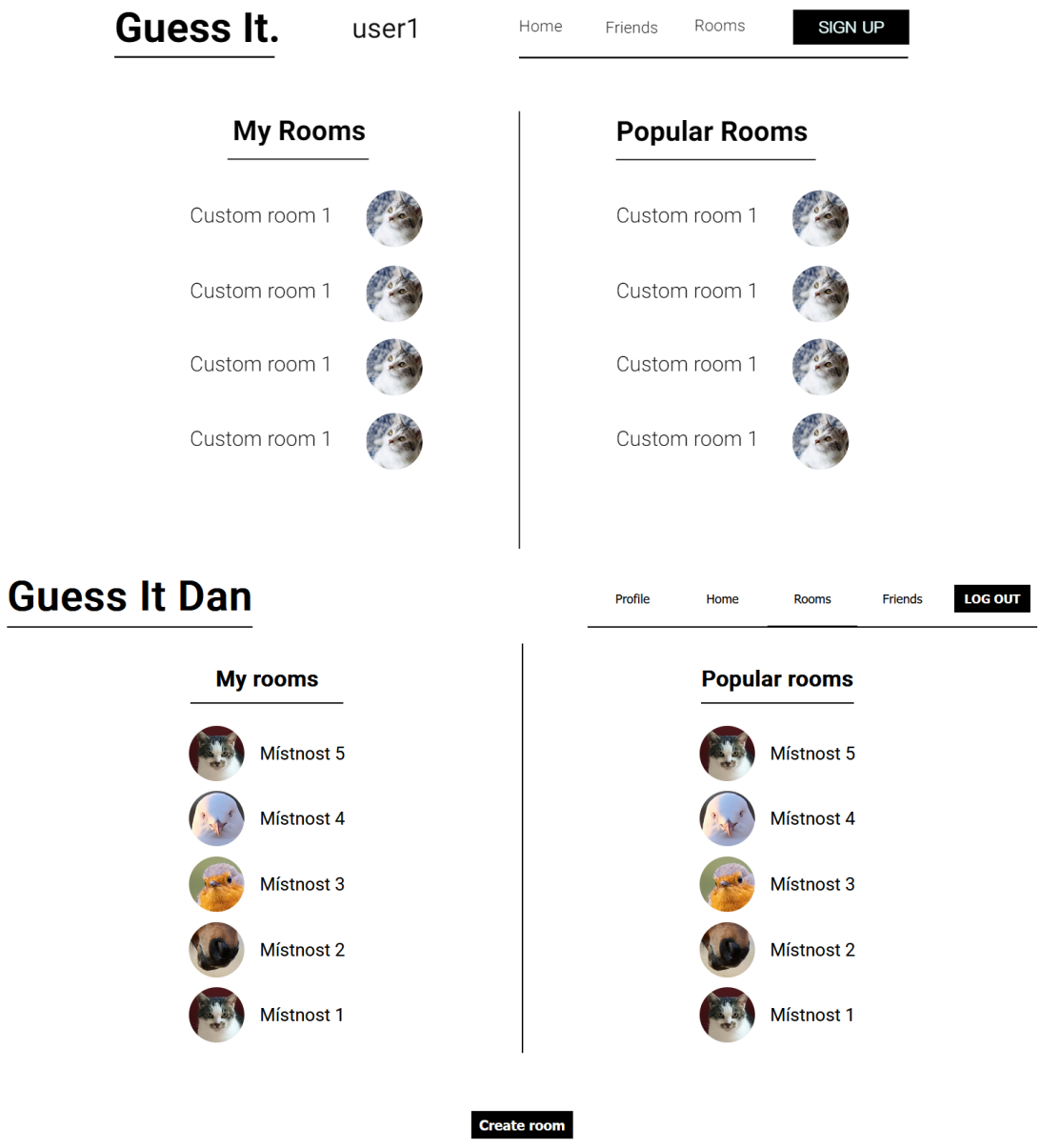
6.1.2.2 Zhodnocení uživatelského testování

Aplikace byla otestována dvěma osobami z oboru se zkušenostmi s návrhem uživatelského rozhraní a dvěma potencionálními uživateli. Toto testování bylo velice přínosné, jelikož ze zpětné vazby každého z testerů vzniklo více než deset úprav.

Největší nedostatky byly ve znázornění, že na nějakou komponentu stránky lze kliknout, ve velikosti polí pro zadání textu, malé informovanosti uživatele (např. administrátor nedostal informaci o tom, že byla úspěšně založena událost, což zapříčinilo, že na toto tlačítko kliknul vícekrát a tím vytvořil více stejných událostí). Dále byl problém v zobrazení tlačítek, které administrátor aktuálně nemůže použít (např. nastavení správné možnosti, pokud je událost ještě otevřená) a celkovém seřazení dat (např. pokud mají dvě události naprosto stejná data, tak je možné, že se jejich posloupnost změní podle toho, v jakém pořadí přijdou ze zdrojového serveru).

Díky tomuto testování došlo poměrně k hodně úpravám na frontendu aplikace. Jednalo se o změny, které se týkaly témat od vizuální úpravy jednotlivých komponent až po části logiky klienta.

Mezi hlavní změny, které stojí za zmínku, patří například přidání tlačítka profilu do záhlaví a následném přesunutí jména aktuálně přihlášeného uživatele k názvu aplikace (tato změna je zobrazena na obrázku 6.1 a navíc si lze na tomto porovnání všimnout upravené hustoty zaplnění samotné stránky), nastavení hover efektu tlačítkům navigace, zobrazení tlačítek a sekcí, které má uživatel právo použít či zobrazit, upravení vzhledu stránky profilu, logika odebrání uživatele z místnosti a další úpravy.



Obrázek 6.1 Srovnání návrhu klienta s funkční aplikací po uživatelském testování

6.2 Dokumentace

Dokumentováním kódu se myslí proces vytváření písemných popisů, pokynů a vysvětlení doprovázejících software. Tento popis poskytuje stručné a výstižné vysvětlení toho, jak program funguje či co jeho části dělají a jak by se měly korektně používat. Je to nedílná součást vývoje, která usnadňuje údržbu, zvyšuje přehlednost a napomáhá lepšímu porozumění kódu mezi vývojáři. [56]

Jedním z hlavních důvodů, proč je dokumentace kódu klíčová, je to, že umožňuje účinnou a efektivní údržbu softwaru. Když vývojáři pracují na složitých programech, často se musí vracet ke kódu, který oni nebo jiní napsali dříve, a upravovat jej. V takových situacích může být dokumentace neocenitelným zdrojem, který vývojářům pomáhá pochopit stávající kód a jeho fungování.

6.2.1 Dokumentace architektury

Dokumentace architektury je postup jehož výsledkem je dokument v libovolném formátu, který souhrnně popisuje a vysvětluje, jak funguje projekt jako celek. Tedy jaké mají samostatné části úkoly a jak spolu komunikují. Tento typ dokumentace většinou vychází z návrhu projektu (v této aplikaci se jedná o sekci 4.1). Jeho hlavním úkolem je ujasnit, jak byla daná aplikace navrhnuta a jaké technologie byly při její implementaci využity. Tato dokumentace se nachází v adresáři této práce ve složce doc v souboru **architecture_doc.md**. [57]

6.2.2 Dokumentace API

Důležitou součástí dokumentace projektu je dokumentace API rozhraní, kterou by měl každý backend server poskytovat. Přehledně popisuje jednotlivé koncové body společně s jejich parametry a stručným popisem co dělají, což umožňuje vyvíjet klienta či frontend aplikaci bez přístupu ke kódu backend serveru. [58]

V tomto projektu je tento typ dokumentace dostupný na backend serveru. Byl vytvořen s použitím nástrojů OpenAPI a Swagger, které automatizují její vytvoření. Dokument má formát webové stránky, která, pokud je server spuštěn, se nachází na relativní adrese „/swagger-ui/index.html“. Jedná se také o jediný koncový bod, který po klientovi nepožaduje autorizaci.

6.2.3 Dokumentace backend serveru

Tato dokumentace popisuje jednotlivé struktury kódu backend serveru [57]. Každá třída, balíček, funkce a proměnná má komentář, který popisuje co reprezentuje, či co a jak provádí. Byla automaticky vygenerována pomocí nástroje Javadoc a nachází se v kořenovém adresáři ve složce **/doc**. Aktualizace dokumentace (znovu vygenerování) je možná pomocí spuštění úkolu `javadoc` v nástroji Gradle. Má formát webové stránky, která přehledně zobrazuje strukturu kódu společně s komentáři, které byly napsány. Za zmínku stojí části komentářů s hodnotami `@return`, `@param` a `@throws`, které si nástroj sám zpracuje a postupně definuje jako výstup, parametry a výjimky, které daná funkce vyhazuje.

6.2.4 Dokumentace frontend serveru

Stejně jako backendendová část projektu musí mít i klient řádně zdokumentovaný kód. Dokumentace byla vygenerována využitím nástroje Compodoc, což je alternativa Javadoc k frameworku Angular. Komentáře jednotlivých funkcí, tříd atd. mají stejnou strukturu jako komentáře psané v jazyce Java. Dokumentace je dostupná v adresáři frontend části projektu ve složce **/docu-**

mentation. Lze také znovu vygenerovat zadáním příkazu `npx compodoc -p tsconfig.json` do příkazového řádku. Obsahuje přehled veškerého kódu projektu s vysvětlením.

6.2.5 Programátorská příručka

Programátorská příručka je stručný návod, který usnadňuje zapojení nového vývojáře do projektu. Obsahuje informace jako stručný popis aplikace, přehled potřebných technologií k vývoji, styl psaní kódu atd. Důležitou částí je také návod na zprovoznění projektu. Tato příručka se nachází v adresáři této práce ve složce doc v souboru **developers_guide.md**.

Kapitola 7
Závěr

7.1 Vyhodnocení požadavků

Hlavním cílem této bakalářské práce bylo vytvořit prototyp webové aplikace umožňující pořádání tipovacích soutěží. Součástí mělo být vytvoření vlastních událostí s vlastním bodováním a tvorba bodovacího žebříčku pro každou místnost.

Výstupem této práce je otestovaný a zdokumentovaný prototyp webové aplikace splňující požadavky výše. Aplikace se skládá ze tří částí - backend server postavený na frameworku Spring, frontend server pracující na frameworku Angular a autorizační server využívající projekt Keycloak. Backend a frontend server komunikují prostřednictvím REST API architektury. Klient aplikace má minimalistický vzhled a je uživatelsky přívětivý. Zabezpečení bylo provedeno pomocí autorizačního serveru a jeho poskytovanými tokeny. Backend server korektně filtruje požadavky dle práv uživatele a k uložení dat využívá databázový systém PostgreSQL.

Práce byla rozdělena do logicky oddělených kapitol, podle kterých vznikal projekt. První kapitola se týkala analýzy již existujících řešení tohoto problému. Byly zde představeny platformy, které různými způsoby umožnily předpovídat výsledky. V závěru bylo zjištěno, že platforma, která by splňovala tato kritéria, zatím neexistuje. Druhá kapitola se zabývala specifikací aplikace. Představila, co znamenají pojmy funkční a nefunkční požadavky a tyto požadavky dále definovala pro tuto aplikaci. Součástí bylo představení případů užití aplikace. Následovala analýza technologií, ve které byly stručně popsány všechny technologie využité k tvorbě a implementaci projektu. Technologie byly rozděleny podle částí aplikace, ve které byly využity. V kapitole týkající se návrhu bylo představeno vysokoúrovňové schéma. Dále zde byl definován návrh menších, ale důležitých částí projektu. Následovala implementace aplikace. V první sekci byly popsány nástroje, které byly využity při tvorbě a následovala samotná implementace, která byla rozdělena dle částí projektu. V části týkající se autorizace byl představen postup, jakým byla implementována ve všech samostatných částech. Zbylé dvě části týkající se backend a frontend serveru, představily strukturu, popisy ukázek kódu a postupy implementace těchto částí. Šestá kapitola se týkala testování a dokumentování aplikace. V první sekci se nachází popis toho, jak byla aplikace otestována a o čem nás testy informovaly. Druhá sekce ujasnila všechny důležité informace ohledně dokumentace aplikace.

Z toho vyplývá, že všechny požadavky zadání byly úspěšně splněny.

7.2 Budoucí práce

Aplikace tvoří kvalitní základ, který je možné dále rozšiřovat a přidávat nové funkce. Je možné přidat mnoho funkcionalit či úprav, které by se mohly do aplikace implementovat. Mezi tyto úpravy patří například:

- **Porovnání předpovědí s jiným uživatelem skupiny**
  Umožnit uživateli porovnat své předpovědi s druhým uživatelem ve stejné místnosti.
- **Omezení hodnotící tabulky podle časů předpovědí**
  Umožnit uživateli zobrazit tabulku, ve které budou započítány pouze předpovědi z daného časového úseku.
- **Rozšíření o takzvanou minci, která náhodně předpoví za uživatele, pokud zapomene vybrat možnost**
  Jedná se o funkci, která náhodně vybere možnost za uživatele, pokud nevybere žádnou možnost v časovém limitu. Avšak pokud mince předpoví korektně, hráč obdrží malou penalizaci.
- **Odznaky, které by si uživatel mohl vystavit na profil**
  Umožnit uživateli získat odměny za různé úspěchy např. dosažení vysokého počtu správných předpovědí, dlouhá doba aktivity. Tyto odměny by si dále mohl vystavit na profilu.
- **Rozšíření o chat**
  Přidat chat do místnosti, a tím umožnit hráčům interakci s ostatními či přidat soukromý chat mezi dvěma uživateli.

Bibliografie

1. SPORTBUSINESS. *How social betting will reshape fan engagement* [online]. 2021. [cit. 2023-01-25]. Dostupné z: https://www.sportbusiness.com/2021/09/how-social-betting-will-reshape-fan-engagement/.
2. RIOT GAMES. *League of Legends Pick'em* [online]. [B.r.]. [cit. 2023-01-16]. Dostupné z: https://pickem.lolesports.com/pickem.
3. REDDIT. *LoL Pick'Em - So close! : leagueoflegends* [online]. 2020. [cit. 2023-01-30]. Dostupné z: https://www.reddit.com/r/leagueoflegends/comments/jlkoc4/lol_pickem_so_close/.
4. ESPORTS.NET. *Worlds Pick'em 2022 Predictions, Information and Tournament Rewards* [online]. 2022. [cit. 2023-01-16]. Dostupné z: https://www.esports.net/news/worlds-pickem-predictions/.
5. FANDOM GAMES COMMUNITY. *Pick'Em Challenge* [online]. 2022. [cit. 2023-01-16]. Dostupné z: https://counterstrike.fandom.com/wiki/Pick'Em_Challenge.
6. KICKTIPP GMBH. *Predictor games for the Premier League, Euro, World Cup and more* [online]. 2022. [cit. 2023-01-16]. Dostupné z: https://www.kicktipp.com/.
7. PEERBET. *PeerBet* [soft.]. 2022. [cit. 2023-01-16]. Dostupné z: https://peerbet.io/.
8. GURU99. *What is a Functional Requirement in Software Engineering?* [Online]. 2022. [cit. 2023-03-29]. Dostupné z: https://www.guru99.com/functional-requirement-specification-example.html.
9. HOOGENRAAD, Wim. *Typy požadavků ve výběru SaaS - IT strategie* [online]. 2017. [cit. 2023-01-24]. Dostupné z: https://www.itpedia.nl/cs/2017/01/04/sisp-2-2-soorten-requirements/.
10. RYDVAL, Slávek. *OCUP.CZ: Případy užití (Use Cases)* [online]. 2010. [cit. 2023-05-02]. Dostupné z: http://ocup.ocup.cz/2010/07/pripady-uziti-use-cases.html.
11. MANAGEMENTMANIA. *Třívrstvá architektura (Three-tier architecture)* [online]. 2015. [cit. 2023-01-26]. Dostupné z: https://managementmania.com/cs/trivrstva-architektura-three-tier-architecture.
12. KOĎOUSKOVÁ, Barbora. *Co je to API a jaké jsou možnosti jeho využití?* [Online]. 2021. [cit. 2023-01-26]. Dostupné z: https://www.rascasone.com/cs/blog/co-je-api.
13. ALTEXSOFT. *Comparing API Architectural Styles: SOAP vs REST vs GraphQL vs RPC* [online]. 2020. [cit. 2023-01-26]. Dostupné z: https://www.altexsoft.com/blog/soap-vs-rest-vs-graphql-vs-rpc/.
14. RED HAT, INC. *What is a REST API?* [Online]. 2020. [cit. 2023-01-26]. Dostupné z: https://www.redhat.com/en/topics/api/what-is-a-rest-api.

15. REDHAT. *What is GraphQL?* [Online]. 2019. [cit. 2023-01-26]. Dostupné z: https://www.redhat.com/en/topics/api/what-is-graphql.
16. AZURE MICROSOFT. *Co je Java? – Příručka pro začátečníky v jazyce Java | Microsoft Azure* [online]. [B.r.]. [cit. 2023-01-27]. Dostupné z: https://azure.microsoft.com/cs-cz/resources/cloud-computing-dictionary/what-is-java-programming-language/.
17. AZURE MICROSOFT. *Co je Java Spring Boot?* [Online]. 2023. [cit. 2023-01-27]. Dostupné z: https://azure.microsoft.com/cs-cz/resources/cloud-computing-dictionary/what-is-java-spring-boot/.
18. THE PYTHON SOFTWARE FOUNDATION. *What is Python? Executive Summary | Python.org* [online]. 2001. [cit. 2023-05-06]. Dostupné z: https://www.python.org/doc/essays/blurb/.
19. COURSERA. *What Is Python Used For? A Beginner's Guide | Coursera* [online]. 2023. [cit. 2023-05-06]. Dostupné z: https://www.coursera.org/articles/what-is-python-used-for-a-beginners-guide-to-using-python.
20. HTTPS://PYTHONBASICS.ORG. *What is Flask Python - Python Tutorial* [online]. 2021. [cit. 2023-05-06]. Dostupné z: https://pythonbasics.org/what-is-flask-python/.
21. THE RAILS FOUNDATION. *Getting Started with Rails — Ruby on Rails Guides* [online]. 2021. [cit. 2023-05-06]. Dostupné z: https://guides.rubyonrails.org/getting_started.html.
22. GRADLE. *What is Gradle?* [Online]. 2022. [cit. 2023-01-27]. Dostupné z: https://docs.gradle.org/current/userguide/what_is_gradle.html.
23. KINSTA. *What Is PostgreSQL?* [Online]. 2022. [cit. 2023-01-28]. Dostupné z: https://kinsta.com/knowledgebase/what-is-postgresql/.
24. POSTGRES.CZ. *PostgreSQL* [online]. 2023. [cit. 2023-01-28]. Dostupné z: https://postgres.cz/wiki/PostgreSQL.
25. SIMICART. *SPA vs. MPA: Pros, Cons & How To Make Final Choice* [online]. 2022. [cit. 2023-01-29]. Dostupné z: https://www.simicart.com/blog/spa-vs-mpa/.
26. EXCELLENT WEBWORLD. *What Is a Single Page Application? Meaning, Pitfalls & Benefits* [online]. 2020. [cit. 2023-01-29]. Dostupné z: https://www.excellentwebworld.com/what-is-a-single-page-application/.
27. ŠTRÁFELDA, Jan. *Co je HTML* [online]. [B.r.]. [cit. 2023-01-28]. Dostupné z: https://www.strafelda.cz/html.
28. IDEALAB. *Co je to CCS? | Definice | Význam | Marketingový slovník | idealab.cz* [online]. 2009. [cit. 2023-01-28]. Dostupné z: https://idealab.cz/slovnik/css-cascading-style-sheets/.
29. HOSTINGER. *What is Bootstrap — Everything You Need to Know* [online]. 2022. [cit. 2023-01-28]. Dostupné z: https://www.hostinger.com/tutorials/what-is-bootstrap/.
30. MOZILLA. *What is JavaScript? - Learn web development | MDN* [online]. 1998. [cit. 2023-01-29]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript.
31. THE NEW STACK. *What Is TypeScript?* [Online]. 2022. [cit. 2023-01-29]. Dostupné z: https://thenewstack.io/what-is-typescript/.
32. KNOWLEDGEHUT. *Angular Introduction | Angular Tutorial* [online]. 2011. [cit. 2023-01-29]. Dostupné z: https://www.knowledgehut.com/tutorials/angular/angular.
33. KOĎOUSKOVÁ, Barbora. *Co je Angular, v čem je jiný než AngularJS a proč ho použít?* [Online]. 2021. [cit. 2023-01-29]. Dostupné z: https://www.rascasone.com/cs/blog/co-je-angular-angularjs.

34. SIMPLILEARN SOLUTIONS. *What is Angular?: Architecture, Features, and Advantages [2022 Edition]* [online]. 2023. [cit. 2023-01-30]. Dostupné z: https://www.simplilearn.com/tutorials/angular-tutorial/what-is-angular.
35. META. *Describing the UI – React* [online]. 2023. [cit. 2023-05-06]. Dostupné z: https://react.dev/learn/describing-the-ui.
36. RASCASONE. *Vue JS: výhody, nevýhody a možnosti využití* [online]. 2023. [cit. 2023-05-06]. Dostupné z: https://www.rascasone.com/cs/blog/co-je-framework-vuejs.
37. OKTA. *What is OAuth 2.0 and what does it do for you? - Auth0* [online]. 2023. [cit. 2023-02-27]. Dostupné z: https://auth0.com/intro-to-iam/what-is-oauth-2.
38. C# CORNER. *Understanding Workflow Of OAuth2.0 Authorization Grant Types* [online]. 2020. [cit. 2023-02-27]. Dostupné z: https://www.c-sharpcorner.com/article/understanding-workflow-of-oauth2-0-authorization-grant-types/.
39. OKTA. *JSON Web Tokens* [online]. 2023. [cit. 2023-02-28]. Dostupné z: https://auth0.com/docs/secure/tokens/json-web-tokens.
40. SUPERTOKENS. *What is a JWT? Understanding JSON Web Tokens* [online]. 2022. [cit. 2023-02-28]. Dostupné z: https://supertokens.com/blog/what-is-jwt.
41. OKTA. *What's the Difference Between OAuth, OpenID Connect, and SAML? | Okta* [online]. 2023. [cit. 2023-02-27]. Dostupné z: https://www.okta.com/identity-101/whats-the-difference-between-oauth-openid-connect-and-saml/.
42. SENNOVATE. *Keycloak: What, when and why to choose?* [Online]. 2022. [cit. 2023-02-27]. Dostupné z: https://sennovate.com/the-mssp-guide-to-keycloak/.
43. JETBRAINS S.R.O. *Database diagrams* [online]. 2022. [cit. 2023-01-24]. Dostupné z: https://www.jetbrains.com/help/idea/creating-diagrams.html.
44. MICROSOFT. *Základy návrhu databáze - Podpora Microsoftu* [online]. 2023. [cit. 2023-01-24]. Dostupné z: https://support.microsoft.com/cs-cz/office/z%5C%C3%5C%A1klady-n%5C%C3%5C%A1vrhu-datab%5C%C3%5C%A1ze-eb2159cf-1e30-401a-8084-bd4f9c9ca1f5.
45. OCTET DESIGN STUDIO. *Minimalist Design: Meaning, Benefits and Process* [online]. 2021. [cit. 2023-03-17]. Dostupné z: https://octet.design/minimalist-design-meaning-benefits-process/.
46. GUILIZZONI, Peldi. *What Are Wireframes? | Wireframing Academy | Balsamiq* [online]. 2008. [cit. 2023-01-24]. Dostupné z: https://balsamiq.com/learn/articles/what-are-wireframes/.
47. WONDERSHARE. *Mockitt* [soft.]. 2023. [cit. 2023-01-20]. Dostupné z: https://mockitt.wondershare.com/download.html.
48. UX PLANET. *An in-depth review of a new UI design tool-Wondershare Mockitt | by ux-planet.org | UX Planet* [online]. 2021. [cit. 2023-03-29]. Dostupné z: https://uxplanet.org/an-in-depth-review-of-a-new-ui-design-tool-wondershare-mockitt-4ba1be1077c9.
49. JETBRAINS. *IntelliJ IDEA* [soft.]. 2000. [cit. 2023-01-20]. Dostupné z: https://www.jetbrains.com/idea/.
50. TORVALDS, Linus. *Git* [soft.]. [B.r.]. [cit. 2023-01-20]. Dostupné z: https://git-scm.com/.
51. JAN. *RESTer* [soft.]. 2022. [cit. 2023-01-20]. Dostupné z: https://addons.mozilla.org/en-US/firefox/addon/rester/.

52. LEDVINKA, Martin. *Spring Framework – Handout* [online]. 2015. [cit. 2023-02-12]. Dostupné z: https://cw.fel.cvut.cz/old/_media/courses/a7b39wpa/spring-handout.pdf.

53. JETBRAINS. *Angular and AngularJS - IntelliJ IDEs Plugin | Marketplace* [online]. 2000. [cit. 2023-03-09]. Dostupné z: https://plugins.jetbrains.com/plugin/6971-angular-and-angularjs.

54. PITTET, Sten. *The different types of testing in software | Atlassian* [online]. 2023. [cit. 2023-04-09]. Dostupné z: https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing.

55. MINDSEA. *User Testing: What It Is, How To Do It and Why It Matters - MindSea* [online]. [B.r.]. [cit. 2023-04-09]. Dostupné z: https://mindsea.com/user-testing/.

56. AYOMIDE, Sarafadeen Ibrahim. *How To Write Code Documentation* [online]. 2022. [cit. 2023-04-13]. Dostupné z: https://www.madcapsoftware.com/blog/write-code-documentation/.

57. SWIMM. *Code Documentation: Benefits, Challenges, & Tips for Success* [online]. 2023. [cit. 2023-04-13]. Dostupné z: https://swimm.io/learn/code-documentation/code-documentation-benefits-challenges-and-tips-for-success/.

58. VASUDEVAN, Keshav. *What is API Documentation? | Swagger Blog* [online]. 2017. [cit. 2023-04-13]. Dostupné z: https://swagger.io/blog/api-documentation/what-is-api-documentation-and-why-it-matters/.

Obsah přiloženého archivu

- `readme.txt` ..... stručný popis obsahu média
- `doc` ..... adresář s dokumentací
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF