UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

BAKALÁŘSKÁ PRÁCE

2024 Matyáš Čížek

Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Mobilní aplikace pro zlepšení organizace a komunikace ve sportovních týmech

Bakalářská práce

2024 Matyáš Čížek

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2023/2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Matyáš Čížek**
Osobní číslo:	**I22244**
Studijní program:	**B0688A140009 Informační technologie**
Téma práce:	**Mobilní aplikace pro zlepšení organizace a komunikace ve sportovních týmech**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování

Cílem bakalářské práce je vytvoření mobilní aplikace pro platformu Android zajišťující lepší organizaci a komunikaci ve sportovních týmech. Hlavními funkce aplikace budou plánování sportovních událostí, jako například tréninky nebo zápasy, zaznamenávání účastí a absencí hráčů na událostech a komunikaci v týmu pomocí chatu. Teoretická část se bude věnovat obecnému vývoji aplikací pro Android a možnostem využití platformy Firebase pro tvorbu aplikací. Součástí bude také analýza již existujících řešení. Praktická část bude věnována návrhu a implementaci konkrétní aplikace v jazyce Kotlin.

Rozsah pracovní zprávy: **min. 30 stran**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

LACKO, Ľuboslav. Mistrovství Android. Přeložil Martin HERODEK. Mistrovství. Brno: Computer Press, 2017. ISBN 978-80-251-4875-4.
SPÄTH, Peter. Pro Android with Kotlin: Developing Modern Mobile Apps. Berlin: Springer, 2018. ISBN 978-1484238196.

Vedoucí bakalářské práce: **Ing. Jan Panuš, Ph.D.**
Katedra informačních technologií

Datum zadání bakalářské práce: **15. prosince 2023**
Termín odevzdání bakalářské práce: **10. května 2024**

L.S.

Ing. Zdeněk Němec, Ph.D. v.r.
děkan

Ing. Jan Panuš, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 28. února 2024

Prohlašuji:

Práci s názvem Mobilní aplikace pro zlepšení organizace a komunikace ve sportovních týmech jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 7/2019 Pravidla pro odevzdávání, zveřejňování a formální úpravu závěrečných prací, ve znění pozdějších dodatků, bude práce zveřejněna prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 19. 12. 2024

Matyáš Čížek v. r.

PODĚKOVÁNÍ

Chtěl bych poděkovat vedoucímu bakalářské práce panu Ing. Janu Panušovi, Ph.D. za rady během psaní této práce. Také bych chtěl poděkovat své rodině za podporu během celého studia.

ANOTACE

Cílem bakalářské práce je vytvoření mobilní aplikace pro platformu Android zajišťující lepší organizaci a komunikaci ve sportovních týmech. Hlavní funkcí aplikace bude plánování sportovních událostí, jako například tréninky nebo zápasy, zaznamenávání účastí a absencí hráčů na událostech a komunikace v týmu pomocí chatu. Teoretická část se bude věnovat obecnému vývoji aplikací pro Android a možnostem využití platformy Firebase pro tvorbu aplikací. Součástí bude také analýza již existujících řešení. Praktická část bude věnována návrhu a implementaci konkrétní aplikace v jazyce Kotlin.

KLÍČOVÁ SLOVA

Android, mobilní aplikace, Firebase, Kotlin, organizace týmu

TITLE

Mobile application for better organisation and communication in sports teams

ANNOTATION

The aim of this bachelor thesis is to create a mobile application for the Android platform to ensure better organisation and communication in sports teams. The main features of this application will be scheduling of sporting events such as practices or matches, recording players participation and absences, and team communication using the chat. The theoretical part will be dedicated to Android app development and possibilities of using the Firebase platform for application development. Analysis of existing solutions will be also included. The practical part will be dedicated to the design and implementation of a specific application in Kotlin.

KEYWORDS

Android, mobile application, Firebase, Kotlin, team organization

OBSAH

SEZNAM ILUSTRACÍ A TABULEK

SEZNAM ZDROJOVÝCH KÓDŮ

SEZNAM ZKRATEK A ZNAČEK

UI - User Interface

API - application programming interface

XML - Extensible Markup Language

SSOT - Single source of truth

SDK - Software development kit

CLI - Command Line Interface

JSON - JavaScript Object Notation

URL - Uniform Resource Locator

BaaS - Backend as a service

UDF - Unidirectional Data Flow

ÚVOD

Bakalářská práce se věnuje návrhu a vývoji nativní aplikace pro Android pro zlepšení komunikace a organizace ve sportovních týmem. Komunikace ve sportovních týmech může být občas velmi složitá, zejména během zjišťování účasti hráčů na společných týmových událostech, jako jsou tréninky nebo zápasy. Členové týmu se mohou domlouvat pomocí chatovacích aplikací, což ale nemusí být zrovna přehledné. Například v chatu bývá obtížné dohledávat starší informace a zjišťovat počet účastníků na trénincích. Každý člen může také preferovat jinou aplikaci nebo nemusí používat žádnou. Tyto problémy by měla tato aplikace řešit. Pro přehlednější organizaci týmových událostí bude poskytovat seznam událostí, do kterého bude moci správce týmu přidávat různé akce, ať už to jsou tréninky, zápasy, nebo jiné události. Každý člen týmu bude potom moci jednoduše prohlížet důležité informace o událostech, vyhledat místo konání, začátek události nebo čas srazu, a také vyjádřit svou účast. Pro další komunikaci v týmu bude sloužit chat. Uživatel se bude moci přihlásit do více týmů, jednoduše mezi nimi v uživatelském rozhraní přepínat a prohlížet potřebné informace.

Úvodní kapitola teoretické části se věnuje obecnému vývoji moderních aplikací pro platformu Android. Bude se zabývat vývojovým prostředím Android Studio, frameworkem pro vývoj nativního uživatelského rozhraní Jetpack Compose a programovacím jazykem Kotlin. Druhá kapitola bude věnována základům Android aplikace. Mezi základní komponenty aplikace patří aktivity, služby, broadcast receivery a poskytovatelé obsahu. Detailněji se bude zabývat doporučenou architekturou pro vývoj aplikací, která se skládá z datové, doménové a UI (User Interface) vrstvy. Závěrečná kapitola teoretické části se bude věnovat platformě Firebase a možnostem jejího využití pro vývoj Android aplikací. Detailněji budou popsány její služby Firebase Authentication, Cloud Firestore, Cloud Functions a Firebase Security Rules.

Praktická část se bude v úvodu zabývat analýzou již dostupných mobilních aplikací pro organizaci sportovních týmů. Další část bude věnována návrhu samotné aplikace. Součástí bude analýza požadavků, digram případů použití a datový model. Závěrečná kapitola bude věnována popisu vývoje samotné aplikace, ukázce uživatelského rozhraní, popisu způsobu propojení s Firebase, implementaci základních funkcionalit aplikace a jejich testování.

1 TEORETICKÁ ČÁST

1.1 Vývoj pro platformu Android

Před začátkem vývoje Android aplikace je důležité zvolit, jestli je výhodnější vytvořit nativní aplikaci, nebo multiplatformní. Mobilní aplikace založené na nativním kódu jsou obvykle rychlejší a mohou lépe využívat vlastností operačního systému, i přesto je vývoj multiplatformních aplikací čím dál více oblíbenější. Je to mimo jiné díky rychlejšímu a levnějšímu vývoji. Pro vývoj těchto aplikací se často využívá sada Flutter od společnosti Google. Umožňuje vývoj aplikací pro mobily, web a desktop v jednom zdrojovém kódu. Pro psaní kódu se používá programovací jazyk Dart. [6]

Nativní vývoj znamená vývoj pro jednu konkrétní platformu, jako je například iOS nebo Android. Zdrojový kód musí být psán pro každou platformu zvlášť. A také se pro každou platformu používají jiné technologie a programovací jazyky. Aplikace pro platformu iOs se píší v jazycích Swift a Objective-C. Preferovanými jazyky, pro vývoj aplikací pro Android jsou Kotlin, Java a C++. V roce 2019 bylo na konferenci Google I/O oznámeno, že preferovaným jazykem pro vývoj Android aplikací bude Kotlin. [1] Jak již bylo zmíněno, výhodou nativních aplikací je jejich výkon. Dále je to také lepší bezpečnost díky využívání vestavěných funkcí zařízení, jako je například šifrování souborů, biometrických systémů nebo kamer. Tyto aplikace se také jednodušeji testují. Vývojové prostředí Android Studio, například nabízí pro tyto účely emulátory. [6]

1.1.1 Operační systém Android

Operační systém Android je vyvíjen organizací Open Handset Alliance. Mezi její součásti patří nejvýznamnější firmy v mobilní branži, jako například Google, HTC, Intel, NVIDIA, Qualcomm a Samsung. Jedná se o systém, který podporuje více platforem, takže může běžet na zařízeních odlišných značek. Tato vlastnost ovšem přináší jednu nevýhodu, a to chybějící optimalizaci systému pro konkrétní platformu. Na druhou stranu Android umožňuje na rozdílných platformách přizpůsobení a vytvoření nadstaveb. Největší výhodou a zároveň nevýhodou této operačního systému Android je jeho otevřenost a možnost úprav od výrobců nebo uživatelů.

Pro Android je možné nainstalovat velké množství aplikací, ne u každé musí být kvalita příliš vysoká, jelikož proces jejich schvalování není tak přísný jako u aplikací pro iOS. Mobilní telefony a tablety s operačním systémem Android vyrábí mnoho různých výrobců, což může přinášet problém, protože aplikace musí být přizpůsobena, aby běžela na přístrojích s různým

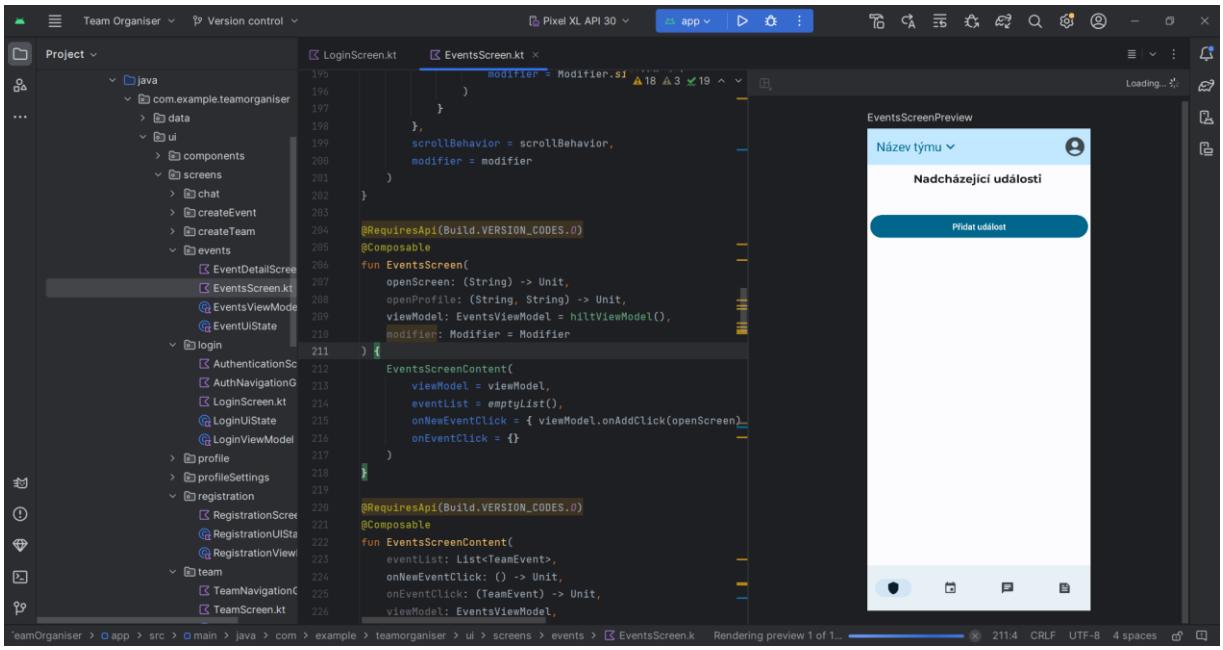
rozlišením displeje, výkonem procesoru a grafiky. Aktualizace nejsou, na rozdíl od ostatních platforem, vydávány centrálně, ale od různých výrobců zařízení, takže se můžeme u různých zařízeních setkat s různými verzemi aplikací. [1]

1.1.2 Android Studio

Oficiálním nástrojem pro vývoj Android aplikací je Android Studio. Je dostupné pro platformy Windows, Linux a macOS. Vývojové prostředí poskytuje řadu funkcí, jako například správu programových zdrojů pro programovací jazyky Kotlin, Javu nebo C++ a také testování aplikací v emulátorech nebo pomocí připojených reálných zařízení. Umožňuje rovněž inspekci kódu. [5]

Emulátory slouží k otestování aplikace na přístrojích s různými velikostmi obrazovek. Díky tomu není potřeba k účelům testování kupovat žádný hardware navíc. Avšak doporučuje se testovat aplikaci alespoň na jednom fyzickém zařízení, protože se v emulátorech může lišit ovládání a výkon aplikace. Pro správu virtuálních zařízení je určen Android Virtual Device Manager. V jeho prostředí je možné vytvořit virtuální mobil, tablet, televizi nebo chytré hodinky, a potom také vybrat jeho úroveň API. [5]

Další důležitou složkou pro vývoj Android aplikací je SDK. Jedná se o soubor nástrojů, které jsou využívány Android Studiem. [5] V Android Studiu je možné spravovat jednotlivé SDK pomocí Android SDK manageru, který obsahuje seznam všech hlavních verzí Androidu s úrovněmi API. Kromě implicitně nainstalovaných novějších verzí je doporučeno nainstalovat také starší SDK, aby aplikace mohla fungovat na starších zařízeních. Při výběru SDK je důležité pozorovat, jaké je procentuální zastoupení verze Androidu mezi používanými zařízeními, pro které je SDK určeno. [1]



Obrázek 1: Prostředí Android Studia [zdroj: vlastní]

1.1.3 Kotlin

Kotlin je staticky typovaný programovací jazyk vyvíjený společností JetBrains. Využívá se pro vývoj aplikací pro platformu Android, web, desktop nebo serverový vývoj. Díky technologii Kotlin Multiplatform umožňuje psát a udržovat stejný kód pro více platforem, což snižuje čas vývoje a počet výskytů chyb. [7]

Jedním z důvodů, proč je jazyk Kotlin doporučován pro vývoj Android aplikací je jeho expresivnost a stručnost, díky tomu se snižuje množství opakovaně používaného kódu. Další výhodou je jeho bezpečnost, obsahuje totiž vlastnosti, které pomáhají zabránit častým programátorským chybám, jako je například NullPointerExption. Tento jazyk je zároveň interoperabilní s Javou, takže kód napsaný v Javě může být zavolán z kódu v Kotlinu a naopak. Dále usnadňuje práci s asynchronním programováním pomocí návrhového vzoru coroutines. Coroutines výrazně zjednodušují správu dlouho běžících úloh na pozadí, které by blokovali hlavní vlákno a aplikace by tak pomalu reagovala. I když je v současné době Java stále podporovaným jazykem pro vývoj Android aplikací, tak ale již neumožňuje používání všech knihoven a nástrojů. Jedním z nich je například Jetpack Compose. Tato sada nástrojů pro vývoj nativního UI je podporována pouze v jazyce Kotlin. [2], [11]

1.1.4 Jetpack Compose

Jetpack Compose je framework pro vytvoření grafického uživatelského rozhraní. Jedná se pouze o část sady knihoven Jetpack, která také například obsahuje knihovnu Appcombat nebo knihovnu Fragment. [12]

Framework poskytuje sadu již připravených komponent, které lze využívat pro vývoj grafického uživatelského rozhraní. Kromě toho umožňuje také vytvoření vlastních komponent. Před zavedením Jetpack Compose se pro tvorbu UI aplikací pro Android používaly XML soubory, ve kterých se definoval vzhled aplikace. Compose umožňuje vytvořit uživatelské rozhraní bez použití XML souborů, k vytvoření komponent se totiž používají funkce. Mezi výhody této nové knihovny je psaní kratšího kódu, pomocí deklarativního kódu, použití architektury single source of truth (SSOT) a jednodušší tvorba vlastních komponent, protože jsou implementovány jako funkce a není tak potřeba je dědit. [12]

Základními prvky jsou funkce, které jsou označeny anotací Composable. Tyto funkce definují určitou část uživatelského rozhraní. Jedna Composable funkce může obsahovat více UI prvků. Ty potom mohou být například seskupeny do sloupce, pokud jsou vloženy do funkce Column, nebo mohou být uvnitř funkce Row, aby se řadily od řádku. Ke změně velikosti, umístění nebo vzhledu komponent se používá parametr modifier. [13]

1.2 Základy Android aplikace

1.2.1 Komponenty aplikace

Komponenty aplikace jsou základními stavebními kameny Android aplikace. Každá z komponent je bodem, přes který může systém nebo uživatel vstoupit do aplikace. Některé komponenty mohou záviset na ostatních. Existují čtyři základní typy: aktivity, služby, broadcast receivery a poskytovatele obsahu. Každý z těchto typů má daný účel a také životní cyklus, který určuje, jak bude komponenta vytvořena a zničena. [3]

Aktivity

Aktivita je hlavní třída, která se zobrazí po spuštění aplikace. Aplikace se může skládat z více aktivit, které si mezi sebou posílají údaje. Uživatelům umožňují přes grafické uživatelské rozhraní přijímat informace od aplikace a ovládat ji. Přes aktivitu se nejčastěji implementuje nějaká částečná úloha, kterou má uživatel realizovat. Příkladem může být vyplnění formuláře, nastavení parametrů nebo výběr položky ze seznamu. Na větších obrazovkách je potom možné realizovat komplexnější úlohy. Například může u úlohy lépe navigovat uživatele a

zobrazit mu lepší uživatelské rozhraní. Aktivity jsou uspořádány hierarchicky, takže po spuštění aplikace se jako první zobrazí hlavní spouštěcí aktivita, ze které se potom mohou spouštět ostatní aktivity. [1]

Služby

Služby slouží k realizaci déle trvajících operací na pozadí. Dále také umožňují spolupráci se vzdálenými procesy. Služby běží na pozadí, tím pádem nepotřebují uživatelské rozhraní. Umožňují asynchronní provádění operací, které mohou trvat déle. Také jsou schopny požádat procesy o provedení operace a sdílení údajů. [1]

Příkladem služby může být přehrávání hudby na pozadí, zatímco uživatel používá jinou aplikaci nebo může také načítat data přes síť bez toho, aby blokovala práci s aktivitou. [3]

Broadcast receivers

Jedná se o objekty, které slouží k vysílání a přijímání. Poslouchají na pozadí události, které jsou vyvolávány zařízením a reagují na ně. Tyto události jsou implementovány jako objekty typu Intent (záměr). Záměry vytváří vydavatelé a poté je směřují přes broadcast do vysílání. Následně jsou zachytávány přijímači, které mají záměry objednané nebo registrované. [1]

Příkladem fungování broadcast receiveru může být aplikace, která umožňuje nastavit upozornění pro uživatele na nadcházející události. Díky tomu, že je upozornění přeneseno k broadcast receiveru, tak není potřeba, aby aplikace stále běžela, dokud není upozornění vypnuto. Přijímače sice nezobrazují uživatelské rozhraní, ale umožňují vytvořit notifikaci, která upozorní uživatele na výskyt události. Avšak nejčastěji je broadcast receiver pouze branou k dalším komponentám a měl by vykonávat co nejméně práce. [3]

Poskytovatelé obsahu

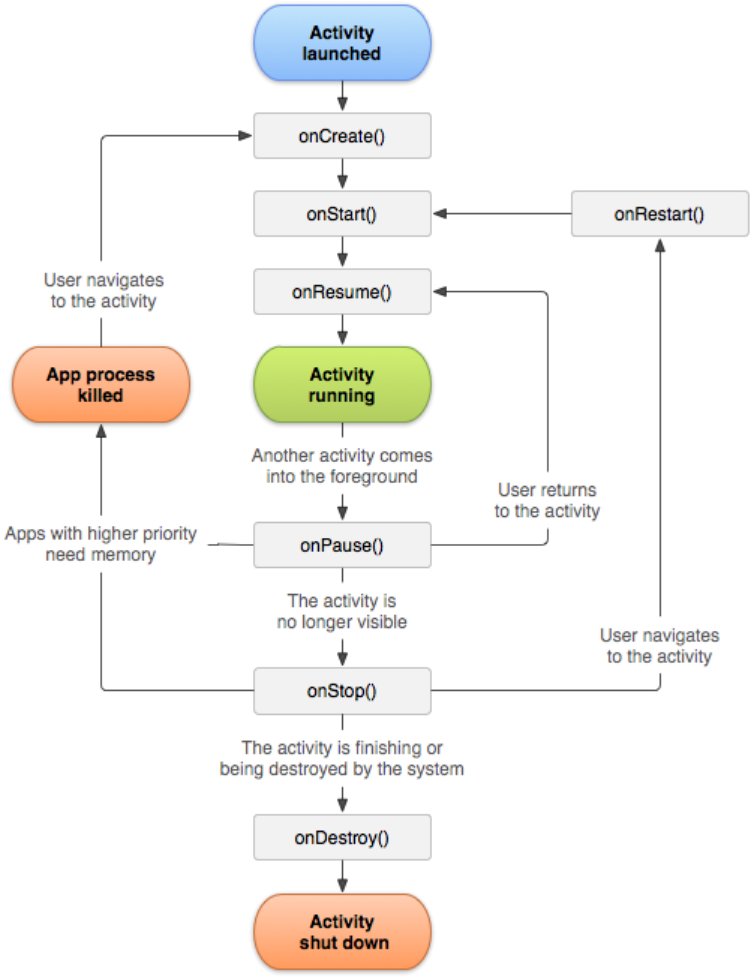
Poskytovatelé obsahu (content providers) slouží k ukládání a sdílení dat mezi aplikacemi a procesy. Díky tomu může aplikace přistupovat k datům z ostatních aplikací, které jsou v tomto případě poskytovateli obsahu. [1]

Systém Android například nabízí poskytovatele obsahu, který spravuje informace o kontaktech uživatele. Jakákoliv aplikace s patřičnými právy se může dotázat poskytovatele obsahu. Například použitím ContactsContract.Data se mohou číst nebo zapisovat informace o určité osobě. [3]

1.2.2 Životní cyklus aktivity

Instance aktivit přecházejí během používání aplikace mezi různými stavy svého životního cyklu. Třída Activity poskytuje několik volání, které upozorňují na změnu stavu nebo na vytvoření, zastavení a spuštění aktivity systémem. [30]

- onCreate() – vykoná se, když systém poprvé vytvoří aktivitu. Stará se o základní logiku při spuštění aplikace.
- onStart() – po zavolání může uživatel vidět aktivitu. Aplikace se připravuje na to, že se aktivita dostává do popředí a stává se interaktivní. Poté, co toto volání skončí, aktivita přechází do stavu Resumed a volá se onResume()
- onResume() – aktivita se dostává do popředí a je interaktivní. Ve stavu Resume je aktivita do té doby než se dostane na pozadí. Například tím, že uživateli někdo zavolá na mobil nebo se vypne obrazovka.
- onPause() – tato metoda je zavolána, když uživatel opustí aktivitu. Aktivita se nezničí, pouze se dostane do pozadí. Může být také viditelná, pokud je obrazovka v režimu s více obrazovkami.
- onStop() – aktivita přechází do stavu Stopped, když ji uživatel nevidí na obrazovce. K této situaci může dojít ve chvíli, kdy nově spuštěná aplikace překryje celou obrazovku, nebo když aktivita přestane běžet a blíží se její ukončení.
- onDestroy() – metoda je zavolána, předtím než je aktivita zničena. Do tohoto stavu se aktivita může dostat buď před svým ukončením, nebo při změně v konfigurace, jako je například překlopení obrazovky.



Obrázek 2: Diagram životního cyklu aktivity [3]

1.2.3 Architektura aplikace

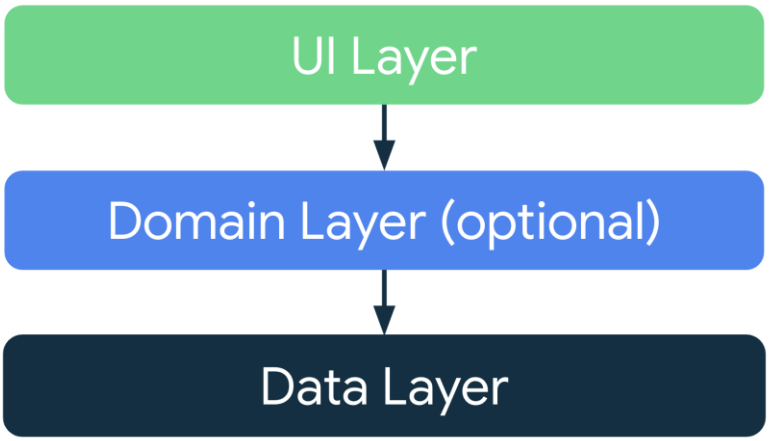
Android aplikace obsahuje typicky několik komponent, jak již bylo zmíněno v předchozí kapitole. V prostředí operačního systému Android je možné, že komponenty budou spuštěny samostatně a mimo pořadí, operační systém je tak může kdykoliv zničit. Z tohoto důvodu není doporučováno ukládat data aplikace v paměti nebo stav v komponentách. Komponenty by zároveň neměly být na sobě závislé.

Architektura aplikace definuje hranice každé části aplikace a přiděluje každé z nich odpovědnosti. Pro návrh architektury existuje několik principů. Nejdůležitějším z nich je oddělení zodpovědností. Třídy, které obsahují uživatelské rozhraní, by měly manipulovat pouze s UI a komunikovat s operačním systémem. Pokud se tento princip bude dodržovat, tak bude zamezeno problémům spojených se životním cyklem aktivit a zjednoduší se tím testování aplikace.

Dalším z principů je oddělení UI od datových modelů. Datové modely by měly být nezávislé na prvcích uživatelského rozhraní. Měly by reprezentovat pouze data aplikace. Perzistentní modely jsou preferovány například díky tomu, že když operační systém zničí aplikaci, tak uživatel nepřijde o svoje data.

Dalším principem je používání „Single Source of Truth (SSOT). SSOT je vlastníkem dat a může je jako jediný modifikovat. Data jsou navenek vystavována jako neměnitelné typy a jediným způsobem, jak je měnit, je pomocí funkcí nebo událostí. Typickým příkladem SSOT je databáze. Tento princip může být využit společně s návrhovým vzorem „Unidirectional Data Flow“ (UDF). V UDF je stav posílán jedním směrem a události, které modifikují data opačným směrem. Tento návrhový vzor zajišťuje konzistenci dat, snižuje náchylnost k chybám, je lehčí odstraňovat v něm chyby a přináší všechny výhody vzoru SSOT.

Pokud uplatníme výše zmíněné principy můžeme rozdělit aplikaci na dvě až tři vrstvy. [3]



Obrázek 3: Diagram typické architektury aplikace [3]

UI vrstva

Tato vrstva slouží k zobrazení dat aplikace na obrazovce. Pokud například uživatel stiskne tlačítko, a dojde tak ke změně dat, uživatelské rozhraní by mělo zobrazit tyto změny. Vrstva UI se skládá ze dvou složek. První jsou prvky uživatelského rozhraní, které překreslují data na obrazovku. Při vývoji Android aplikací k tomu mohou být použity Views nebo funkce z knihovny Jetpack Compose. Dalším prvkem jsou state holders, které udržují data, vystavují se uživatelskému rozhraní a zajišťují logiku. Může je například reprezentovat třída ViewModel. [3]

Doménová vrstva

Doménová vrstva není povinná. Je umístěna mezi UI a datovou vrstvou. Doménová vrstva se může starat o zapouzdření komplexní business logiky nebo té jednoduché, která je použita ve více ViewModelech. Třídám z této vrstvy se obvykle říká use casy nebo interaktory. Každý use case by měl mít na starost pouze jednu funkcionalitu. [3]

Datová vrstva

Datová vrstva obsahuje business logiku. Business logika definuje pravidla, jak má aplikace vytvářet, ukládat a měnit data. Vrstva je obvykle tvořena repozitáři, které obsahují zdroje dat. Je doporučené vytvořit repozitář pro každý typ dat, se kterým aplikace pracuje. Třídy s repozitáři zveřejňují data zbytku aplikace, centralizují je, řeší konflikty mezi jinými zdroji dat, oddělují zdroje dat od zbytku aplikace a obsahují business logiku aplikace. [3]

1.3 Firebase

Firebase je platforma od společnosti Google, která poskytuje službu Backend as a Service (BaaS). Jedná se cloudový model pro vývoj aplikací. Tato služba zajišťuje outsourcing backendu aplikace, zatímco vývojář se zaměřuje na vývoj frontendu aplikace. Služba nabízí infrastrukturu, aplikace a skripty pro zajištění backendových úkolů, jako je například hosting, autentizace nebo práce s databází. Mezi její výhody patří rychlejší vývoj, snížení ceny vývoje a také odpadá potřeba řídit serverovou infrastrukturu. Používání však přináší také některé nevýhody, jako například nižší flexibilitu nebo méně možností pro přizpůsobení. Mezi obvyklé případy použití Firebase patří vývoj backendu, hosting webové aplikace, testing nebo monitorování aplikace. [8], [10]

Co se týče ceny za tuto službu, uživatelé Firebase si mohou vybrat mezi dvěma plány. Spark Plan, který je zdarma, a Blaze Plan, který používá cenový model pay-as-you-go. Plán zdarma nabízí například hosting více domén, real-time databázi, správu uživatelů, 10 GB úložného prostoru a další funkcionality. Pro ostatní funkce, jako jsou například cloudové funkce, hosting aplikace, uložiště Cloud Storage, je třeba aktivovat Blaze Plan. [8], [9]

1.3.1 Firebase Authentication

Firebase Authentication nabízí backendové služby, SDK a předpřipravené UI knihovny pro autentizaci uživatele. Mezi hlavní funkcionality patří autentizační řešení FirebaseUI Auth. Toto řešení se stará o přihlášení uživatelů přes uživatelské rozhraní a implementuje doporučené postupy pro autentizaci na mobilních zařízeních a webových stránkách. Řeší také takové případy, jako je obnova účtu. Umožňuje následující způsoby autentizace: kombinace

emailu a hesla, pomocí poskytovatele identity (Google, Apple, Facebook, Twitter, GitHub), posíláním SMS zpráv, propojení vlastního autentizačního systému nebo anonymní autentizace.

Služba pracuje tak, že nejprve získáme v aplikaci přihlašovací údaje od uživatele ve formě e-mailu a hesla nebo tokenu OAuth od poskytovatele identity. Poté se tyto údaje pošlou do Firebase Authentication SDK. Backendové služby ověří správnost údajů a vrací odpověď klientovy. Po úspěšném přihlášení můžeme pracovat s informacemi o uživatelském profilu a ověřovat přístup k datům.

Objekt uživatele ve Firebase reprezentuje přihlášeného uživatele v aplikaci. Obsahuje základní data o účtu: ID, e-mail, jméno a URL fotografie. Pokud v aplikaci budou potřeba další údaje o uživateli, tak se musí uložit v databázi, například v Cloud Firestore. Po přihlášení se uživatel stává aktuálním uživatelem v instanci Auth. Instance udržuje stav uživatele, takže když se obnoví stránka, aplikace neztratí informace o uživateli. Reference na objekt uživatele je udržována v instanci do té doby, než se uživatel odhlásí. [31]

1.3.2 Cloud Firestore

Cloud Firestore je cloudová, NoSQL databáze pro vývoj aplikací pro mobilní platformy, web a servery. Dokáže udržovat data synchronizovaná napříč všemi klientskými aplikacemi a nabízí offline podporu pro aplikace. Databáze Firestore může být také snadno integrována s dalšími službami, jako jsou například Cloud Functions.

Mezi další vlastnosti patří podpora flexibilních hierarchických datových struktur. Data jsou ukládána do dokumentů, které obsahují pole pro zápis hodnot. Dokumenty jsou seskupeny do kolekcí, které slouží k organizaci dat a sestavování dotazů. Podporují různé datové typy, ať už to jsou jednoduché datové typy jako string nebo number, nebo komplexní zapouzdřené objekty. V rámci dokumentů je také možné vytvářet sub-kolekce a díky tomu sestavovat hierarchické datové struktury.

Pomocí dotazů je možné získat specifické nebo všechny dokumenty v kolekci. Je také umožněno vytvářet mělké dotazy, to znamená, že se získávají data z jakékoliv úrovně hierarchie bez toho, aby se zároveň načítaly nepotřebné celé kolekce a vnořené sub-kolekce. Dotazy mohou obsahovat více filtrů a také mohou kombinovat filtrování a řazení. Díky tomu, že jsou dokumenty indexovány, je výkon dotazu úměrný velikosti sady výsledků místo velikosti sady dat. Uživatel může používat aplikaci, i když je zařízení bez připojení k síti. Po připojení k síti potom Cloud Firestore synchronizuje všechny změny, které uživatel provedl

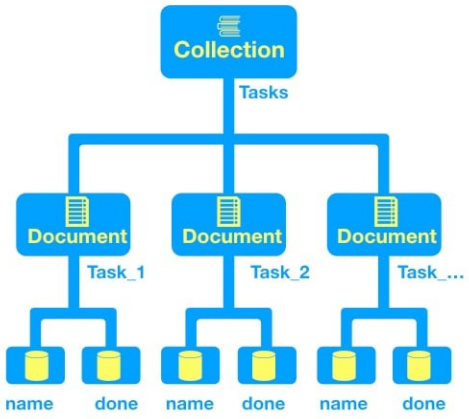
offline. Další výhodou je možnost použití posluchačů, které aplikaci upozorňují na změnu v datech tím, že posílá datový snapshot. Díky této funkcionalitě se z databáze načtou poslední změny bez toho, aby bylo potřeba načítat všechna data z databáze. [32]

Datový model

Jak již bylo zmíněno základní jednotkou uložiště je dokument. Obsahuje pole s přiřazenými hodnotami. Každý dokument je identifikován jménem. Dokumenty se uchovávají v kolekcích, které jsou bez schématu. Můžeme tedy do dokumentů vkládat jakékoliv pole s hodnotami s libovolným datovým typem. Doporučuje se ovšem napříč více dokumenty stejného typu používat stejná pole a datové typy. Kolekce může obsahovat pouze dokumenty, nemůže tedy obsahovat jinou kolekci. Jméno dokumentu je v rámci kolekce unikátní. Tato jména generuje Cloud Firestore automaticky, ale je také možné zadat vlastní. Kolekce existuje jen do té doby, dokud obsahuje dokumenty.

Každý dokument v databázi je identifikován podle svého umístění v databázi. Pro vytvoření odkazu na umístění se v kódu používají reference. Reference je jednoduchý objekt, který ukazuje na místo v databázi. Kromě dokumentů je také možné vytvořit referenci na kolekci.

Kolekce může také obsahovat sub-kolekci. Sub-kolekce je kolekce, která je propojena s jinou kolekcí. Použitím lze vytvořit hierarchické struktury, ve kterých se snadněji přistupuje k datům. Je možné vytvořit maximálně 100 úrovní hierarchie. Je potřeba dávat si pozor na odstraňování sub-kolekcí. Pokud dojde k odstranění dokumentu, který obsahuje sub-kolekci, dojde ke smazání pouze dokumentu, sub-kolekce se automaticky nesmaže. Je potřeba ji odstranit manuálně. [33]



Obrázek 4: Příklad datového modelu v Cloud Firestore [28]

1.3.3 Firebase Security Rules

Slouží k zabezpečení dat uložených v databázích Cloud Firestore a Firebase Realtime Database nebo uložišti Cloud Storage. Využívají flexibilní konfigurační jazyky, kterými definují, k jakým datům budou mít uživatelé přístup. Bezpečnostní pravidla pro Realtime Database používají formát JSON, zatímco pravidla pro Cloud Firestore a Cloud Storage využívají vlastní unikátní jazyky. Pravidla jsou flexibilní, mohou se přizpůsobit, aby byly vhodné pro specifické chování a strukturu aplikace. Podle potřeb mohou být obecné nebo konkrétní. Protože pravidla nejsou definována uvnitř aplikace a klienti je tak nemohou ovlivnit, chyby v aplikaci nenarušují data a jsou vždy zabezpečena.

Firebase Security Rules fungují tak, že se nejdříve hledá shodný vzor s databázovými cestami. A potom se používají vlastní podmínky k povolení přístupu k datům v daných cestách. Je povinné definovat pravidla pro každý Firebase produkt, který je použit v aplikaci. Pravidla jsou uplatňována jako výrazy OR. Pokud tedy příliš obecné pravidlo uděluje přístup k datům, konkrétní pravidlo nemůže přístup omezit. Proto je důležité, aby se pravidla příliš nepřekrývala. [34]

1.3.4 Cloud Functions

Cloud Functions je serverless framework, který umožňuje automaticky spouštět backendový kód jako reakci na vyvolané události. Mohou být napsány v jazycích JavaScript, TypeScript a Python. Kód je uložen v infrastruktuře Google Cloud a běží ve spravovaném prostředí. Pro jejich funkci není potřeba spravovat vlastní servery. Funkce se zavádí jednoduše jedním příkazem z příkazové řádky a potom už není potřeba se starat o přístupové údaje, konfiguraci serveru ani poskytování nových serverů. Cloudové funkce jsou izolovány od uživatele, takže je zajištěno, že jsou soukromé a pracují tak, jak je potřeba.

Životní cyklus funkce probíhá následovně. Nejdříve je potřeba napsat novou funkci, vybrat poskytovatele událostí a definovat podmínky, pod kterými budou spouštěny. Po zavedení funkce, Firebase CLI (Command Line Interface) vytvoří archiv ve formátu zip s kódem funkce a nahraje ji do bucketu Cloud Storage předtím než Cloud Functions vytvoří repozitář Artifact Registry v našem projektu. Cloud Build potom vrátí kód funkce a sestaví její zdroj. Image kontejneru pro sestavený kód funkcí je nahrán do soukromého repozitáře Artifact Registry v našem projektu a nová funkce je zavedena. Když poskytovatel události vygeneruje událost, která odpovídá podmínkám funkce, tak kód je spuštěn. Pokud funkce obsluhuje

mnoho událostí najednou, Google Cloud vytvoří více instancí, aby pracovala rychleji. Po změně funkce se vyčistí staré verze instancí a jsou nahrazeny novými. Po smazání funkce jsou smazány všechny její instance i archivy zip. Připojení mezi funkcí a poskytovatelem připojení je také odstraněno. [35]

2 PRAKTICKÁ ČÁST

2.1 Analýza existujících aplikací

V obchodu Google Play je pro uživatele Androidu k dispozici mnoho aplikací, které jsou zaměřeny na organizaci ve sportovních týmech. V této části budu porovnávat silné a slabé stránky těch, které patří mezi ty nejpopulárnější.

2.1.1 Týmuj

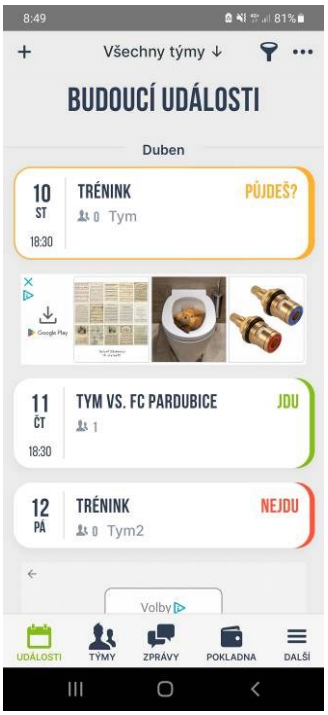
Jedná se o českou aplikaci, která je kromě systémů Android a iOS, dostupná také pro web. Mobilní aplikace je dostupná od roku 2018. V současné době má 350 tisíc registrovaných členů. Měsíčně ji aktivně využívá přes 100 tisíc sportovců. Mezi hlavní funkce patří plánování událostí tréninků a zápasů. Poskytuje přehled o docházce a umožňuje snadnou komunikaci a informovanost v týmu. Aplikace nabízí základní a prémiovou verzi. Týmuj Premium umožňuje navíc oproti základní verzi neomezený počet členů v týmu, rozšířené notifikace, vytvoření podskupiny v docházce, neomezený počet událostí nebo podpora nastavení událostí. Cena této verze je 119 Kč za 1 měsíc a 599 Kč za 6 měsíců. [14], [15]

Výhody

- Česká lokalizace
- K dispozici je přehledný návod
- Webová aplikace
- Přehledné a jednoduché uživatelské rozhraní
- Umožňuje vybírat platby od členů týmu
- K dispozici je chat, jak pro celý tým, tak k jednotlivým událostem

Nevýhody

- Základní varianta omezuje počet členů v týmu nebo událostí
- Základní verze neumožňuje vytvářet opakované události



Obrázek 5: Aplikace Týmuj [14]

2.1.2 PlayerPlus

Další oblíbenou aplikací je PlayerPlus. Je možné ji stáhnout na systémy Android a iOS, nebo používat webovou aplikaci. Využívá ji přes 250 000 týmů a přibližně 4 miliony hráčů. Aktuálně má aplikace pro Android přes milion stažení. Dostupná je ve více než deseti jazycích. Aplikace nabízí široký výběr funkcionalit. Kromě obvyklých funkcí pro organizaci týmu, umožňuje také pozorovat komplexní statistiky o hráčích, spravovat seznam pokut, vytvářet sestavy nebo plánovat spolujízdy v autě na zápasy. Také lze propojit data z aplikací Google Fit nebo Apple Health. Uživatelé mohou používat verzi zdarma nebo Premium. Cena prémiové verze je pro jeden tým je 5,99 euro měsíčně, s každým dalším přidaným týmem se potom cena zvyšuje o 4,50 euro. [16]

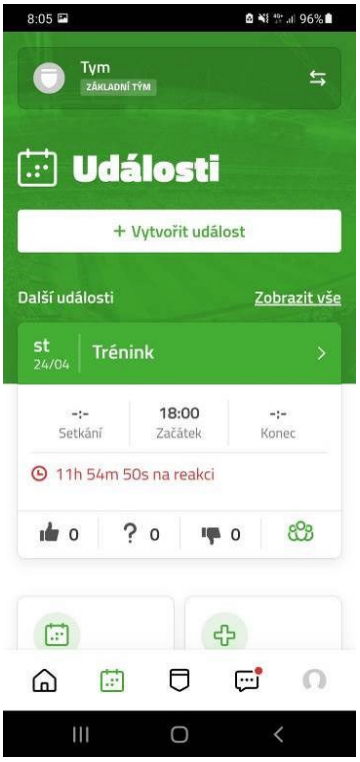
Výhody

- Česká lokalizace
- Poskytuje větší množství funkcí oproti ostatním aplikacím
- Pro členy týmu jsou připraveny různé role s různými pravomocemi. Například trenér, pokladník, pomocník nebo hráč
- Umožňuje vytvořit ankety

- TeamCloud – cloudové uložiště pro členy týmy, v základní verzi má kapacitu 15 MB, v prémiové potom 1 GB

Nevýhody

- Reklamy v základní verzi
- Základní verze omezuje maximální počet členů týmu na 30
- Vyšší cena prémiové verze



Obrázek 6: Aplikace PlayerPlus [16]

2.1.3 Heja

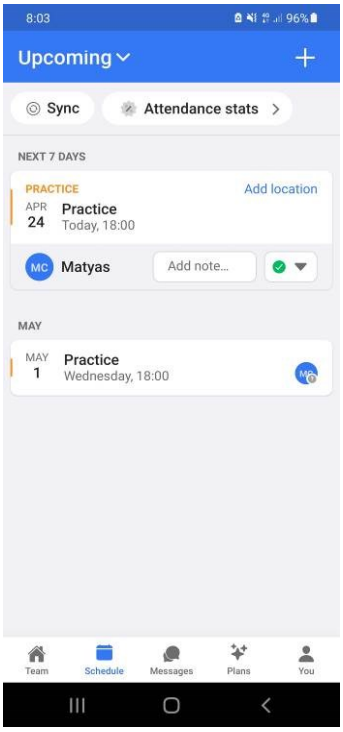
Další nástrojem pro komunikaci v týmu je Heja. Dostupná je pro platformy Android, iOS a web. Tuto aplikaci používá přes 350 000 týmů. Cílem aplikace je umožnit trenérům, rodinám a hráčům být součástí dobře řízeného sportovního týmu. Mezi základní funkce patří organizace týmových aktivit a komunikace. Využívají se k tomu dva chaty, jeden je pro celý tým a druhý je jen pro trenéry a správce. Rozšiřující funkcí je například nastavení úkolů, které má tým splnit. Trenér může nahrát video nebo sdílet odkaz, ve kterém úkol vysvětlí, a poslat ho hráčům. Aplikace je ke stažení zdarma, ale poskytuje také verzi Pro. Cena verze Pro je 13,99 euro za měsíc. [17]

Výhody

- Přehledné uživatelské rozhraní
- V prémiové verzi je možné mít neomezený počet správců v jednom týmu
- V základní verzi není omezen počet členů týmu

Nevýhody

- Chybí česká lokalizace
- Webová aplikace je dostupná pouze v prémiové verzi
- Statistiky o účastech hráčů jsou zpřístupněny pouze ve verzi Pro



Obrázek 7: Aplikace Heja [17]

2.1.4 Stack Team App

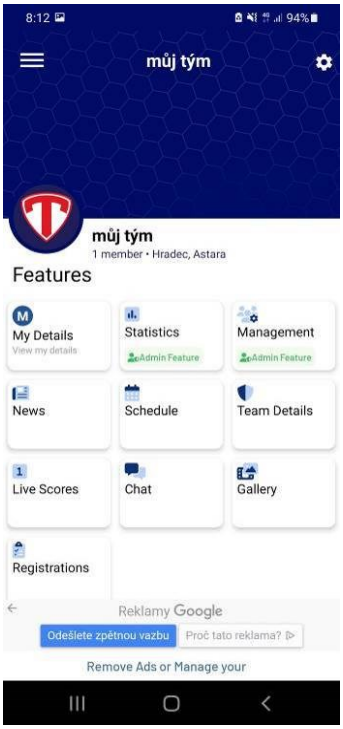
Stack Team App je platforma, která umožňuje týmům a dalším skupinám zlepšit komunikaci pomocí aplikace, kterou si mohou sami přizpůsobit. Je dostupná pro Android, iOS nebo přes web. Uživatelé si mohou do aplikace nahrát vlastní logo, zvolit barvy rozhraní nebo vybrat tlačítka, která budou dostupná v aplikaci. Členové týmu si tedy mohou vybrat, jaké funkce by aplikace měla mít. Mezi dostupné funkce patří posílání upozornění, vytváření událostí, chat, zobrazení živých výsledků, týmový obchod a mnoho dalších funkcí. Všechny funkce aplikace jsou zdarma. [18]

Výhody

- Na výběr je velké množství funkcí, které mohou být použity bez omezení
- Možnost přizpůsobení aplikace

Nevýhody

- Chybí česká lokalizace
- Uživatelské rozhraní není příliš přehledné
- Obsahuje reklamy



Obrázek 8: Aplikace Stack Team App [18]

2.2 Návrh aplikace

Aplikace bude kompatibilní pouze se zařízeními s operačním systémem Android. Pro běh aplikace bude potřeba minimálně SDK 26. Při podílu používaných zařízení z května 2024 to znamená, že aplikace bude kompatibilní s 95,4% zařízeními se systémem Android. [19] Bude nabízet základní, nezbytné funkcionality pro členy sportovních týmu tak, aby uživatele neomezovala.

Uživatelé přihlášení do týmu mohou mít jednu ze dvou rolí: správce týmu a uživatel.

2.2.1 Požadavky

Funkční požadavky

- Uživatel se může přihlásit pomocí emailu a hesla
- Uživatel může upravit a smazat svůj profil
- Uživatel může vytvořit svůj vlastní tým
- Uživatel, který vytvořil tým, se stává jeho správcem
- Člen týmu může vyjádřit svoji účast na událostech
- Člen týmu může prohlížet seznam účastníků na událostí
- Člen týmu může psát a mazat své zprávy v chatu
- Člen může prohlížet všechny události týmů
- Správce může předat svou roli jinému členovi týmu
- Správce může posílat pozvánku do týmu
- Správce může odebírat členy týmu
- Správce může přidávat, upravovat a odstraňovat členy týmu
- Správce může vytvořit, editovat, mazat a zrušit události
- Správce může vytvořit opakující se událost

Nefunkční požadavky

- Aplikace bude dostupná v českém jazyce
- Aplikace bude využívat pro autentizaci a ukládaní dat platformu Firebase
- Aplikace bude nabízet přehledné uživatelské rozhraní
- Aplikace bude podporovat tmavý režim

Obrázek 9: Diagram případů užití [zdroj: vlastní]

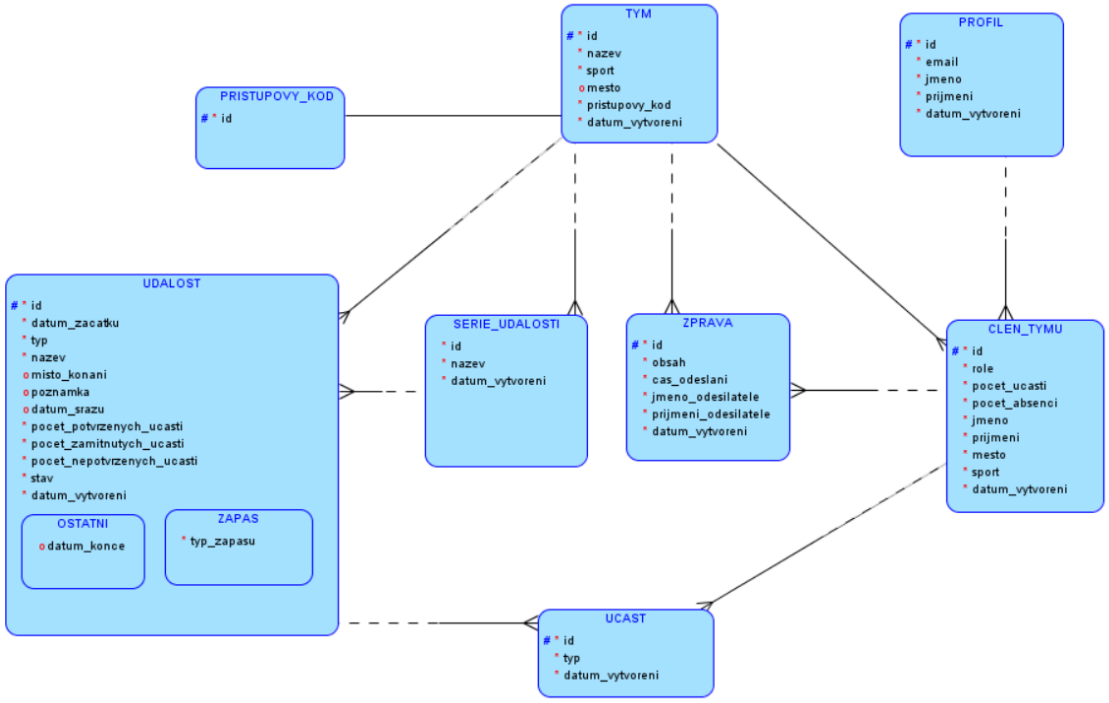
2.2.2 Datový model

Jednotlivé dokumenty jsou v databázi Cloud Firestore organizovány do hierarchické struktury s kolekcemi a sub-kolekcemi [33]. Kořenovými kolekcemi jsou profily, týmy a přístupové kódy k týmu. Dokument s profily obsahuje informace o uživateli. Dokument s týmem, kromě polí s daty o týmu, obsahuje také sub-kolekce s událostmi, sériemi událostí, zprávami a členy týmu.

Sub-kolekce jsou použity z důvodu snadnějšího získávání dat. Například pokud budu znát ID dokumentu s týmem a ID události, mohu se jednoduše dostat k referenci na událost bez toho, abych použil složitější dotazy. Problém by mohl nastat ve chvíli, kdy bych potřeboval získat události ve všech týmech. Tato situace dá se vyřešit pomocí skupinových dotazů nad kolekcemi, které poskytuje Cloud Firestore.

Dokument se členem týmu je identifikován stejným ID jako profil, se kterým je propojen. Díky tomu se snadněji mohou získávat data, pokud známe identifikátor uživatele, a také to zajistí, že se jeden uživatel nemůže stát vícekrát členem jednoho týmu. Protože je možné vytvořit opakované událostí, jsou také vytvářeny dokumenty identifikující sérii událostí, jejich ID potom obsahují dané dokumenty s událostmi. Každý dokument s událostí může obsahovat sub-kolekci s účastmi členů. Ty se uchovávají z toho důvodu, aby bylo možné zaznamenávat a zobrazovat, kteří členové týmu potvrdili, nepotvrdili nebo zamítli účast na události.

V některých entitách jsou uvedeny duplicitní informace, jako například jméno a příjmení v dokumentu s profily a členy týmu. Je to takto vyřešené z toho důvodu, aby byly dotazy pro získání dat jednodušší. S tímto přístupem mohu získat potřebná data z dokumentu člena týmu jedním dotazem. Tento přístup přináší nevýhodu v tom, že když se změní hodnota v jednom dokumentu, tak je potřeba, aby se změna promítla také v ostatních dokumentech. Tento problém se dá vyřešit jednou ze služeb od platformy Firebase – Cloud Functions. [35]



Obrázek 10: ER digram aplikace [zdroj: vlastní]

2.3 Implementace aplikace

Aplikace je napsána v jazyce Kotlin. Pro vytvoření bylo využito vývojové prostředí Android Studio. Uživatelské rozhraní bylo vytvořeno pomocí sady nástrojů Jetpack Compose. Pro vyřešení navigace jsem použil knihovnu Jetpack Compose Navigation. Implementované komponenty uživatelského rozhraní, jako jsou například tlačítka, navigační lišty a textová pole, vycházejí z designového systému Material Design 3.

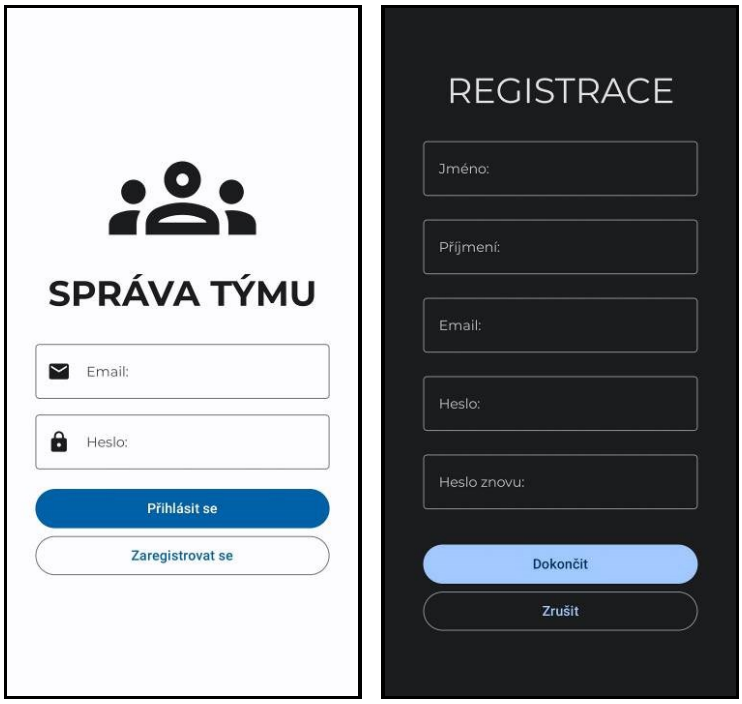
2.3.1 Uživatelské rozhraní

UI aplikace je rozděleno na tři hlavní části. První část má na starost autentizaci uživatele. Patří do ní obrazovky s přihlašovacím a registračním formulářem. Po přihlášení je uživatel přesměrován do části správy týmu. Zde může spravovat události týmu, používat chat, zobrazovat účasti ostatních členů a přizpůsobit si nastavení týmu. V části nastavení profilu může uživatel spravovat profil, pod kterým je aktuálně přihlášen.

Přihlašovací obrazovka

Po spuštění aplikace se uživateli jako první zobrazí přihlašovací obrazovka. Autentizace je zajištěna pomocí e-mailu a hesla. Pokud uživatel zadá nesprávné údaje, objeví se chybová hláška v podobě Snackbar komponenty v dolní části obrazovky. Uživatel nemůže používat aplikaci anonymně, musí se zaregistrovat. Při registraci uživatel poskytuje jméno, příjmení a

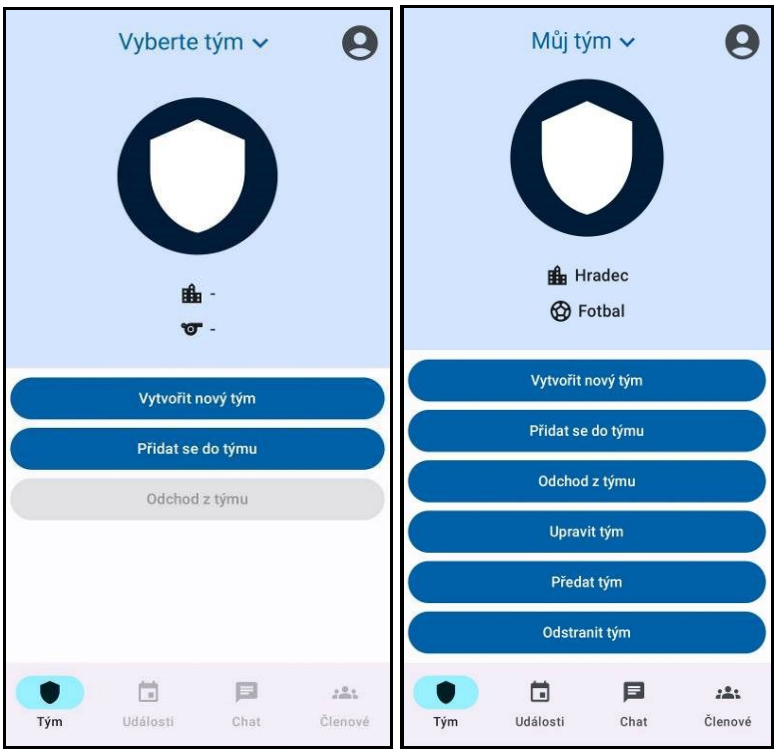
e-mail. Formulář je opatřen validací zadaných údajů. Kontroluje se vyplnění všech údajů, správná délka, korektní emailová adresa a také zda adresa již není použita.



Obrázek 11: Obrazovka přihlášení a registrace [zdroj: vlastní]

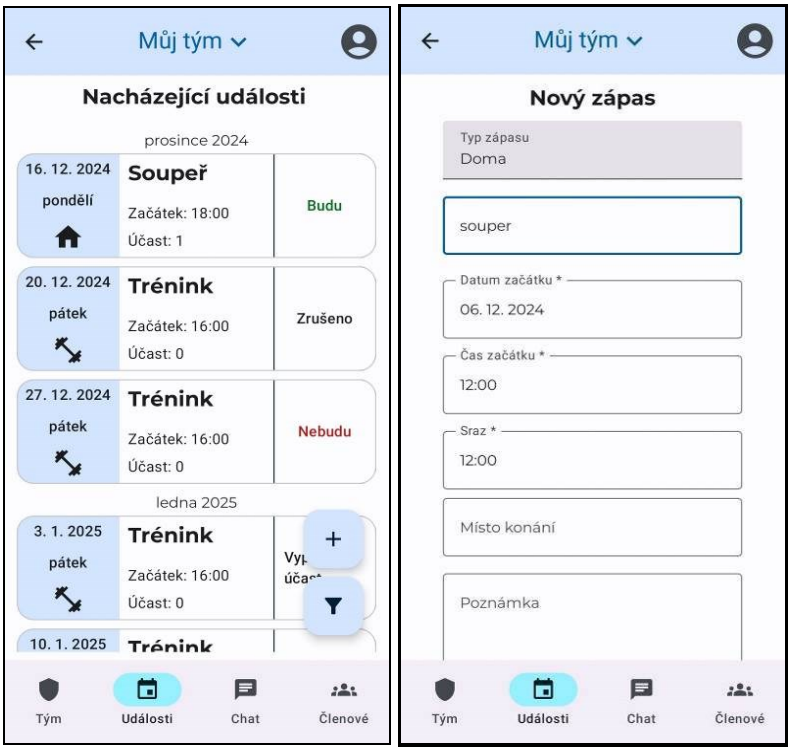
Obrazovky týmu

Obrazovka s nastavením týmu se zobrazí po přihlášení uživatele. V této části může uživatel spravovat svůj tým. Obrazovky v této části disponují dolní lištou pro snadnou navigaci mezi obrazovkami s nastavením týmu, událostmi, chatem a seznamem spoluhráčů. V horní liště může uživatel pomocí tlačítka zobrazit seznam týmů, ve kterých je členem, a mezi nimi vybrat, který by chtěl zobrazit. Pokud není členem žádného týmu, tak v horní liště uvidí pouze text „Vyberte tým“. V této chvíli má dvě možnosti, buď může vytvořit nový tým, nebo se přidat do již vytvořeného týmu. Tým se vytváří přes formulář, kde stačí zadat pouze název týmu, město a sport, který budou členové provozovat. Jestli uživatel obdržel unikátní přístupový kód od jiného člena, tak může zvolit možnost „Přidat se do týmu“. Kód zadá do textového pole v dialogovém okně. Po vstupu do týmu se mu zobrazí v horní liště název týmu a v hlavním menu základní informace o týmu.



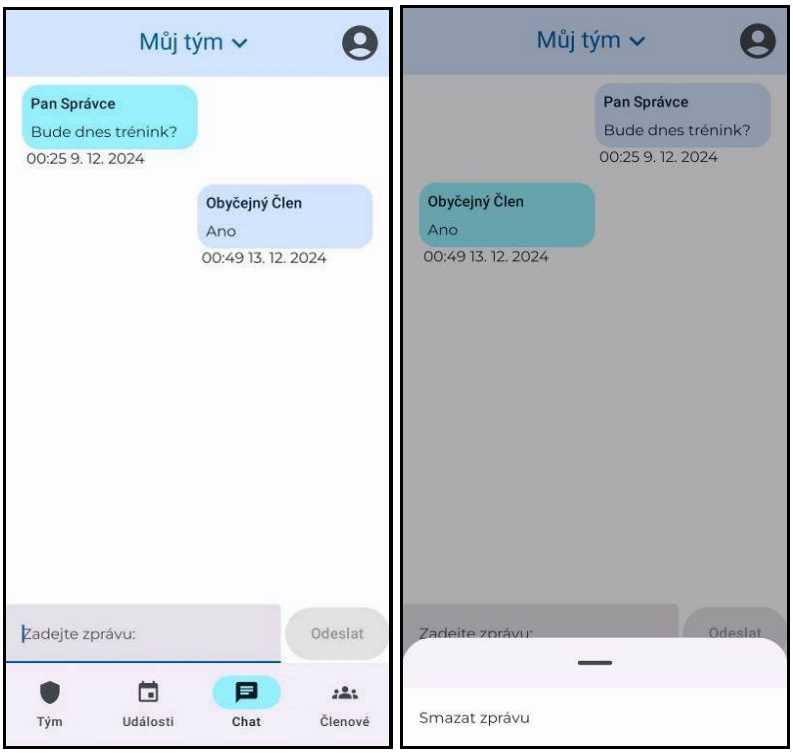
Obrázek 12: Nastavení týmu bez vybraného týmu a s vybraným týmem [zdroj: vlastní]

Po stisknutí položky Události v dolní liště se uživateli zobrazí seznam týmových událostí. Jednotlivé položky seznamu zobrazují základní informace o událostech a o tom, jestli se jich uživatel zúčastní. Různé typy událostí jsou odlišeny ikonami. Seznam může uživatel také filtrovat, jestli chce zobrazit nadcházející nebo dokončené události. Dále obrazovka obsahuje tlačítko pro vytvoření nové události, které může používat pouze správce týmu. Po rozkliknutí libovolné události v seznamu se zobrazí obrazovka s detaily této události. V této části může člen týmu vyjádřit svou účast na akci. Správce zde může navíc měnit údaje o události, zrušit ji nebo ji smazat.



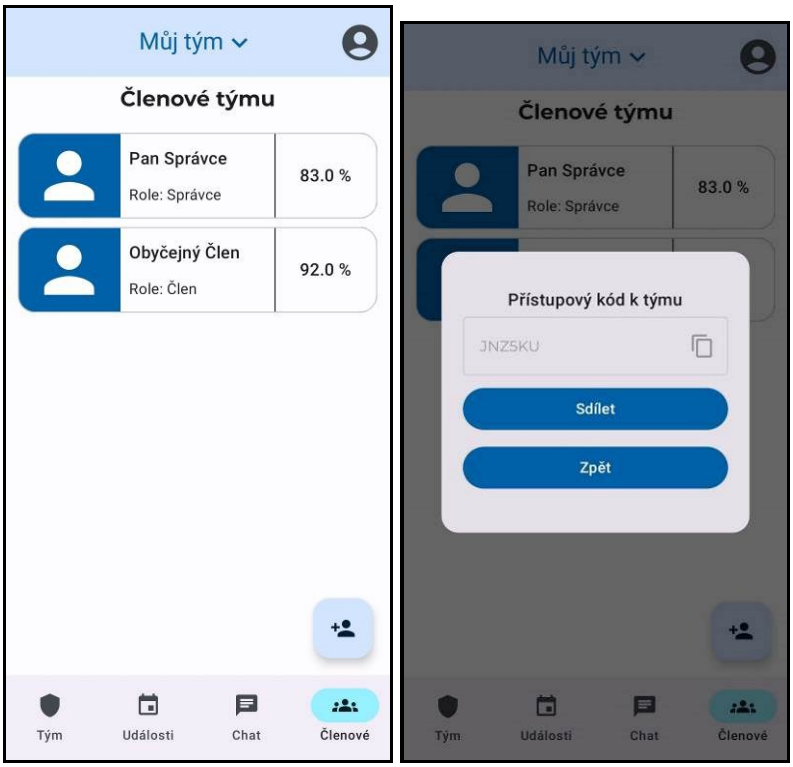
Obrázek 13: Obrazovky pro správu událostí [zdroj: vlastní]

Další funkcí pro členy týmu je chat. Obsahuje klasický seznam zpráv, kde jsou zprávy přihlášeného člena a ostatních členů od sebe odlišeny jak barevně, tak umístěním. V dolní části obrazovky se nachází textové pole pro zadání zpráv. Po dlouhém stisknutí vlastní zprávy se v dolní části zobrazí menu s možností zprávu odebrat.



Obrázek 14: Obrazovka chatu [zdroj: vlastní]

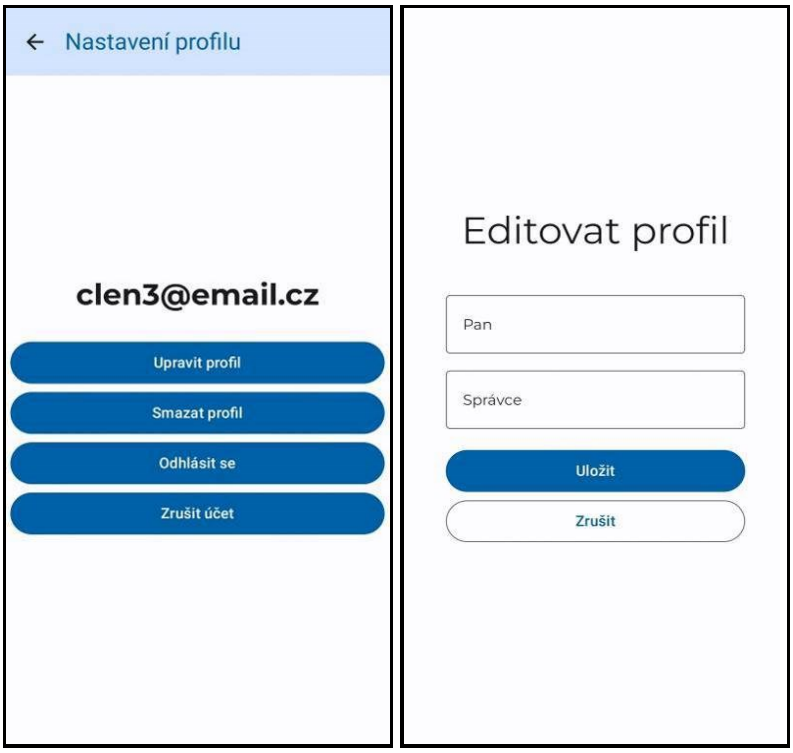
Poslední položkou v menu je přehled účastí členů týmů. Po rozkliknutí uvidí uživatel seznam členů týmu, kde každá položka obsahuje jméno, roli a procentuální účast člena na událostech. Správce týmu na obrazovce vidí tlačítko pro přidání člena týmu. Po jeho stisknutí se objeví dialogové okno s přístupovým kódem k týmu, který může buď zkopírovat do schránky, a nebo ho sdílet přes oblíbenou aplikaci (např. Messenger). Správce týmu může také členy odebrat z týmu, nebo jim předat svou roli správce. Tyto možnosti uvidí po rozkliknutí položky člena týmu ze seznamu.



Obrázek 15: Obrazovky správy členů [zdroj: vlastní]

Nastavení profilu

V této sekci se může uživatel odhlásit, upravit nebo smazat svůj profil. Obrazovka se skládá z jednoduchého menu s emailem přihlášeného profilu a tlačítky. Uživatel má možnost editovat své jméno a příjmení. Pokud stiskne a potvrdí odhlášení nebo smazání profilu, tak je odkázán na úvodní registrační obrazovku.



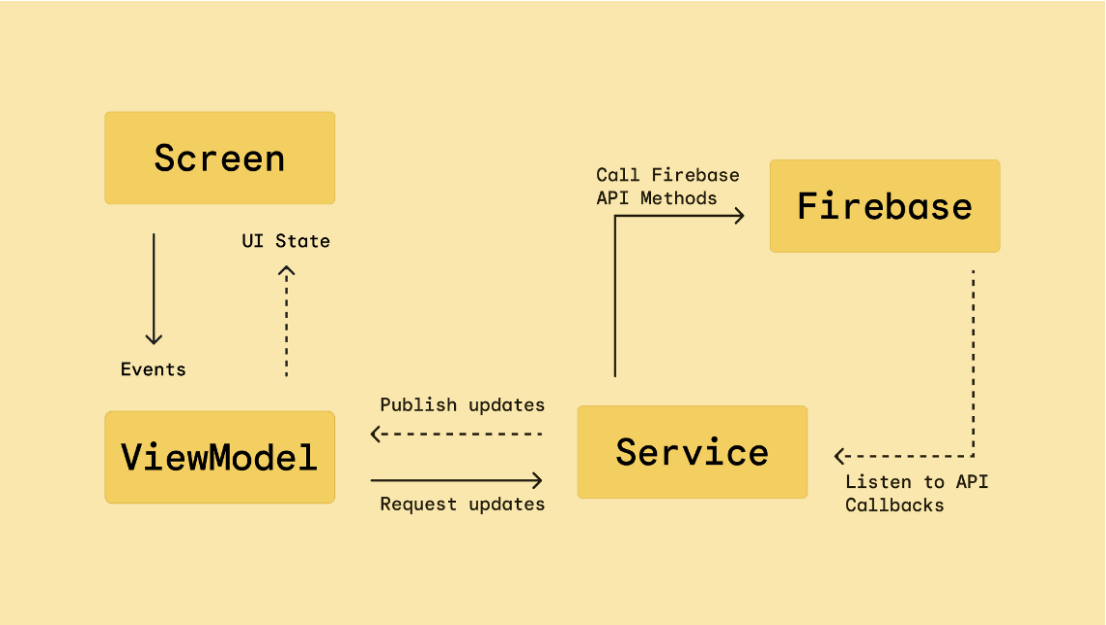
Obrázek 16: Obrazovky nastavení profilu [zdroj: vlastní]

2.3.2 Struktura projektu

Zdrojové soubory projektu se nacházejí ve složce app/com/example/teamorganiser. V tomto adresáři se nachází třída TeamOrganiserActivity, která je vstupním bodem aplikace a nastavuje obsah na hlavní komponentu TeamOrganiserApp.

Architektura aplikace je navržena podle doporučených postupů a je tedy rozdělena na dvě vrstvy: datovou a UI. Ve složce ui jsou uloženy jednotlivé komponenty uživatelského rozhraní. Pro každou obrazovku je vytvořena komponenta s uživatelským rozhraním vytvořeném pomocí JetpackCompose funkcí, ViewModelem obrazovky a pomocnou datovou třídou se stavem uživatelského rozhraní. ViewModel obsahuje logiku příslušné obrazovky a udržuje její stav. Jeho velkou výhodou je to, že zůstává v paměti, dokud v ní je uložen ViewModelStoreOwner, díky tomu je schopen udržovat data, i když proběhnou změny v konfiguraci, například při překlopení obrazovky. [20]

Datová část se nachází ve složce data. Skládá se z datových tříd, které reprezentují entity, se kterými se pracuje v databázi, a služeb. Služby poskytují funkce pro komunikaci s databází a autentizaci. Do ViewModelů jsou vkládány pomocí dependency injection.



Obrázek 17: Diagram komunikace mezi vrstvami aplikace [29]

2.3.3 Propojeni s Firebase

Aplikace je zaregistrována jako součást projektu vytvořeném pomocí Firebase konzole. V kořenovém adresáři aplikace je uložen konfigurační soubor google-services.json, který obsahuje unikátní identifikátory projektu. Nezbytné knihovny pro využívání Firebase jsou naimportovány do projektu pomocí dependencies v souboru Gradle, včetně služeb Cloud Firestore, Firebase Auth a Cloud Functions. Pro každou z těchto služeb je vytvořen uvnitř objektu FirebaseModule objekt singleton. Díky tomu je zajištěno, že bude vytvořena jen jedna instance objektu, která může být použita napříč aplikací. [21]

Pro využití funkcí Firebase Auth je vytvořen interface AccountService. Do jeho implementace je pomocí dependency injection vložen objekt typu FirebaseAuth. Poskytuje funkce pro správu profilu přihlášeného uživatele a také například vlastnost, která vrací ID přihlášeného uživatele. Pro každou entitu v databázi je vytvořena speciální služba. Například pro operace s týmy je vytvořeno rozhraní TeamService. To obsahuje funkce pro vytváření, mazání, editaci a načítání týmů z databáze. Třída implementující rozhraní využívá jak objekt FirebaseFirestore, tak také FirebaseFunctions, aby se mohly volat cloudové funkce.

Zavedené Cloudové funkce jsou uloženy v souboru index.js ve složce functions. Obsahují funkce pro aktualizaci dat týmu a profilu napříč různými kolekcemi, mazání dokumentů a kolekcí a každodenní aktualizaci počtu účastí na událostech. Důvodem, proč využívám cloudovou funkci pro mazání, je ten, že v aplikaci není možné volat funkci, která smaže kolekce a smazáním dokumentu nedojde ke smazání jeho sub-kolekcí. [36]

```
exports.recursiveDelete = onRequest(
    {timeoutSeconds: 540, memory: "2GB"}, async (req, res) => {
      const path = req.body.data.path;

      try {
        await firebaseTools.firestore
            .delete(path, {
              project: "team-organiser-c6945",
              recursive: true,
              force: true,
              token: process.env.FB_TOKEN,
            });
      } catch (error) {
        console.error("Při mazání se stala chyba:", error);
      }
      return {
        path: path,
      };
    });
```

Zdrojový kód 1: Cloud funkce pro mazání dokumentů

2.3.4 Přepínání obrazovek

V knihovně Compose se využívá pro navigaci komponenta Navigation. Tato komponenta definuje 4 důležité typy: NavHost, NavGraph, NavController a NavDestination. NavHost je komponenta UI, která obsahuje aktuální obrazovku. NavGraph je datová struktura, která definuje všechny možné destinace a vzájemně je propojuje. NavDestination je položka v NavGraphu. Navigaci mezi obrazovkami řídí NavController. [22]

V samotné aplikaci se využívá NavController v kořenové komponentě aplikace TeamOrganiserApp. Inicializuje kořenový NavHost, ve kterém jsou definovány cesty k jednotlivým částem aplikace. Navigační graf je ještě rozdělen na podgrafy pro autentizaci, správu týmu a správu profilu. Pro změnu aktuální obrazovky se používají dva typy funkcí – navigate a navigateAndPopUp. Funkce navigate přesune na vrchol zásobníku novou obrazovku, přitom se žádná jiná obrazovka ze zásobníku nevyjme, takže se pomocí tlačítka zpět může uživatel vrátit na předchozí obrazovky. Funkce navigateAndPopUp odebere ze zásobníku vybranou obrazovku a na její místo vloží novou. Tento přístup se například používá při přechodu z přihlašovací obrazovky na obrazovku týmu, protože potom by se po stisknutí zpět měla aplikace vypnout. [22]

```
NavHost(
    navController = appState.navController,
    startDestination = AUTH_SCREEN,
    modifier = Modifier.padding(paddingValues)
) {
    navigation(
        route = AUTH_SCREEN,
        startDestination = LOGIN_SCREEN
    ) {
        composable(LOGIN_SCREEN) {
            LoginScreen(
                showSnackbar = { message, length ->
                    appState.showSnackbarMessage(message, length)
                },
                openAndPopUp = { route, popUp ->
                    appState.navigateAndPopUp(route, popUp)
                },
                openRegistrationScreen = { route ->
                    appState.navigate(route)
                }
            )
        }
```

Zdrojový kód 2: Část navigačního grafu

2.3.5 Autentizace uživatele

Pro autentizaci uživatele byla využita služba Firebase Authentication. Z nabízených možností autentizace uživatele využívám přihlášení pomocí e-mailu a hesla. Jak již bylo zmíněno výše pro využívání služeb Firebase Authentication se využívá rozhraní AccountService.

Toto rozhraní se například využívá ve třídě RegistrationViewModel. Po stisknutí tlačítka Dokončit registraci se zavolá metoda createUserWithEmailAndPassword, která vytvoří nového uživatele a automaticky ho a přihlásí. Protože služba FirebaseAuthentication umožňuje uložit pouze identifikátor (e-mail nebo telefon) a heslo uživatele, tak ostatní údaje musí být uloženy jiným způsobem. [31] Z tohoto důvodu jsem v databázi Cloud Firestore vytvořil kolekci Profiles, ve které jsou uloženy dokumenty s dalšími nezbytnými údaji o uživatelích, jako je například jméno a příjmení. Každý dokument v této kolekci je identifikován stejným ID jako příslušný účet ve Firebase Authentication.

```
val userId = accountService.signUp(email, password)
if(userId != null){
    val profile = Profile(
        id= userId,
        firstName = firstName,
        lastName = lastName,
        email = email
    )
    profileService.saveProfile(profile)
}
```

Zdrojový kód 3: Registrace uživatele

2.3.6 Vytvoření události

Uživatel má možnost vytvořit tři typy událostí: zápas, trénink a jiná událost. Pokud uživatel zvolí typ události zápas, tak se do databáze Cloud Firestore přidá jeden dokument. Tréninky a další události, je ale možné na rozdíl od zápasu opakovat a může být vytvořeno více dokumentů najednou. Proto formulář k jejich vytvoření obsahuje navíc možnost zadat datum posledního termínu a typ opakování. Událost může být opakována každý den, každý týden, každých 14 dní, každý měsíc nebo každý rok. Pokud je tedy vybrána možnost opakovat událost, tak se do databáze přidá takový počet událostí, jaký se s vybranou frekvencí časově vejde mezi datum začátku a konce. Ještě před vytvořením těchto souvisejících událostí se vytvoří nový dokument reprezentující sérii událostí. Toto je důležité například ve chvíli, když chce správce týmu odstranit ne jednu, ale rovnou celou sérii událostí. V databázi se potom vyhledají všechny události s ID série a poté se smažou.

```
if (newEvent is OtherEvent && repetitionType != RepetitionType.NEVER) {
    val setId = eventService.saveEventSet(
        eventSet = EventSet(name = name),
        teamId = currentTeamId
    )
    newEvent = newEvent.copy(setId = setId)
    val events = getEventList(
        event = newEvent,
        firstDate = startDate,
        lastDate = lastDate,
        startTime = startTime,
        endTime = endTime,
        repetition = repetitionType
    )
    eventService.saveEvents(events, currentTeamId)
} else {
    eventService.saveEvent(newEvent)
}
```

Zdrojový kód 4: Vytvoření série událostí

2.3.7 Potvrzení účasti na události

Počet účastníků na události mohou členové vidět buď na obrazovce se seznamem účastí, a nebo na detailu události. Účast potvrzuje člen týmu na detailu události. Na výběr má tři možnosti: „Budu“, „Nebudu“ a „Nevím“. Tlačítko, na které klikne, zůstává aktivní, dokud nestiskne jiné. Po potvrzení účasti dojde ke změně počtu účastníku na dané události. O to se stará metoda updateParticipationCount. Protože operace změny počtu účastí by měly být atomické, v metodě je použita funkce z knihovny Cloud Firestore runTransaction. Veškerá manipulace s poli confirmCount, denyCount a unsureCount je prováděna uvnitř této funkce, díky tomu je zabráněno vzniku nekonzistencí v těchto datech. [23]

```
firestore.runTransaction { transaction ->
    val event = transaction.get(eventRef)
    val confirmCount = event.getLong("confirmedCount") ?: 0
    val denyCount = event.getLong("deniedCount") ?: 0
    val unsureCount = event.getLong("unsureCount") ?: 0

    val incrementedField = when(newValue){
        ParticipationType.CONFIRM ->
            "confirmedCount" to confirmCount + 1
        ParticipationType.DENY ->
            "deniedCount" to denyCount + 1
        ParticipationType.UNSURE ->
            "unsureCount" to unsureCount + 1
    }

    val decrementedField = when(oldValue){
        ParticipationType.CONFIRM ->
            "confirmedCount" to confirmCount - 1
        ParticipationType.DENY ->
            "deniedCount" to denyCount - 1
        ParticipationType.UNSURE ->
            "unsureCount" to unsureCount - 1
        null -> null
    }
    if (decrementedField != null && decrementedField.second > 0) {
        transaction.update(eventRef, mapOf(incrementedField,
decrementedField))
    }else{
        transaction.update(eventRef, mapOf(incrementedField))
    }
```

Zdrojový kód 5: Počítání účasti na události

Kromě účastí na událostech je potřeba také sledovat celkovou účast jednotlivých členů. K tomu se používají Firebase Cloud Functions, konkrétně funkce jejíž spuštění jde naplánovat na určitý čas. Tyto funkce používají ke spuštění Cloud Scheduler. [37] Aktualizace počtu účastí je spouštěna každý den o půlnoci. Probíhá tak, že se načtou všechny události ze všech týmů, které mají stav „Naplánovány“, takže ještě nebyly ukončeny, ani zrušeny. U každé z těchto událostí se načtou dokumenty s účastmi členů a podle typu účasti se aktualizuje počet účastí členů. Zároveň se událost nastaví, jako ukončena.

2.3.8 Komunikace členů

Komunikace členů je zajištěna přes jeden týmový chat. Týmové zprávy se načítají z databáze ve formě flow. Flow je proud dat, který může být vypočítáván asynchronně. Je to typ, který dokáže vysílat více hodnot sekvenčně. Hodnoty ve flow musí být stejného typu. [24] Aby se změny v databázi projevily okamžitě na UI, hodnoty jsou předány jako parametr funkci z knihovny Compose collectAsStateWithLifecycle(). Pokud tedy dojde ke změně dat v databázi, obrazovka s chatem se znovu vykreslí a uživatel uvidí ty nejnovější zprávy. Stejný

přístup je uplatněn také při načítání dalších entit, jako jsou týmy, členové týmu nebo události. [38]

```
@OptIn(ExperimentalCoroutinesApi::class)
override val chatMessages: Flow<List<ChatMessage>>
    get() = profileService.getCurrentTeamId().flatMapLatest { teamId ->
        if (teamId == null) {
            flowOf(emptyList())
        } else {
            firestore
                .collection(TEAMS_COLLECTION)
                .document(teamId)
                .collection(MESSAGES_COLLECTION)
                .orderBy(CREATED_AT_FIELD)
                .snapshots(MetadataChanges.INCLUDE)
                .map {
                    it.toObjects<ChatMessage>()
                }
        }
    }
```

Zdrojový kód 6: Načítání zpráv z chatu

2.3.9 Přihlášení do týmu

Uživatel se může stát členem dvěma způsoby. Buď si vytvoří vlastní tým, ve kterém se zároveň stane správcem, a nebo obdrží přístupový kód k týmu. Kód je týmu vygenerován po jeho vytvoření, jedná se o unikátní šestimístný řetězec. Oprávnění na zobrazení přístupového kódu má pouze správce týmu. Může se k němu dostat na obrazovce se seznamem členů. Po kliknutí na tlačítko Přidat se mu objeví dialog s kódem, který uživatel buď může zkopírovat do schránky, nebo ho může sdílet.

Ke sdílení je použit Intent, což je objekt, který žádá akci od nějaké jiné aplikace. Vytváří se nová instance Activity, do které se vloží Intent, který popisuje aktivitu, která se má spustit. Po stisknutí tlačítka sdílet se tak správci zobrazí menu s aplikacemi, pomocí kterých může být kód sdílen. [25]

```
fun shareAccessCode(context: Context, code: String){
    val intent = Intent(Intent.ACTION_SEND).apply {
        type = "text/plain"
        putExtra(Intent.EXTRA_SUBJECT,
context.getString(R.string.teamAccessCode_subject))
        putExtra(Intent.EXTRA_TEXT, code)
    }

    context.startActivity(
        Intent.createChooser(
            intent,
            "Sdílet kód k týmu"
        )
    )
}
```

Zdrojový kód 7: Sdílení přístupového kódu

2.3.10 Správa členských rolí

Pokud si správce týmu přeje předat tým, může tak učinit v menu týmu. Kde se po stisknutí tlačítka „Předat tým“ objeví výběrové pole se seznamem členů týmu. Po potvrzení je správcova role změněna na obyčejného člena. Předat roli je také možné přes obrazovku se členy týmu. Stačí se dostat, pomocí kliknutí na jednoho z členů týmu ze seznamu, na jeho detail a stisknout tlačítko „Předat roli“.

Na straně frontendu je omezení funkcí podle členských rolí vyřešeno skrytím komponent, které může používat pouze jen určitý typ uživatele (např. tlačítko pro vytvoření události může stisknout pouze správce). Na straně backendu jsou pro zabezpečení využity Firebase Rules. Ty omezují čtení, zápis, editaci a mazání entit podle toho, jestli je uživatel přihlášen nebo je členem týmu s patřičnou rolí. Jsou zapsány v souboru firebase.rules, mohou být ale také zobrazeny a editovány pomocí Firebase konzole.

```
match /messages/{messageId} {
      allow read: if request.auth != null;

      allow create: if request.auth != null
          && request.resource.data.senderId == request.auth.uid
          && isTeamMember(teamId, request.auth.uid);
      allow delete: if request.auth != null
          && resource.data.senderId == request.auth.uid
          && isTeamMember(teamId, request.auth.uid);
      }
}

function isTeamMember(teamId, userId) {
  return
exists(/databases/$(database)/documents/teams/$(teamId)/members/$(userId));
}
```

Zdrojový kód 8: Příklad pravidel Firebase Rules

2.3.11 Testování

Aplikace pro Android se mohou testovat pomocí dvou typů testů: instrumentované a lokální. Instrumentované testy musí běžet na fyzickém zařízení nebo emulátoru. Tyto testy nejčastěji kontrolují uživatelské rozhraní a interakci s aplikací. Lokální testy se spouští ve vývojovém prostředí nebo na serveru. Věnují se obvykle nějaké malé části aplikace a většinou jsou rychlé. [27]

Projekt obsahuje adresář test pro lokální unit testy a androidTest pro instrumentované testy. V instrumentovaných testech ověřuji správné zobrazení komponent jednotlivých obrazovek. Například kontroluji správnost obsahu textového pole po zadání textu nebo správné vykreslení obrazovek pro členy týmů s různými rolemi. Unit testy kontrolují správnou funkci ViewModelů. Tyto testy by měly běžet v izolaci, neměly by potřebovat závislost na repozitáři nebo databázi, aby mohli fungovat. Tento problém se nejčastěji řeší vytvořením testovacího dvojníka. Což je objekt, který se chová a vypadá jako komponenta aplikace, ale je upraven tak, aby odpovídal účelům testování. [26]

K vytvoření testovacích dvojníků využívám knihovnu MockK. Například k otestování funkce ViewModelu pro potvrzení editace profilu nepotřebuji závislost na službách AccountService a ProfileService. Vytvořím si tedy jejich dvojníky pomocí funkce mockk() a při korektním vyplnění údajů pouze ověřím, jestli se funkce pro editaci profilu v databázi zavolala právě jednou. V tuto chvíli nepotřebuji testovat, jestli se údaje do databáze skutečně uložily.

```
@OptIn(ExperimentalCoroutinesApi::class)
@Before
fun setUp(){
    Dispatchers.setMain(testDispatcher)

    accountService = mockk(relaxed = true)
    profileService = mockk(relaxed = true)

    viewModel = EditProfileViewModel(
        accountService,
        profileService
    )
}

@OptIn(ExperimentalCoroutinesApi::class)
@After
fun tearDown() {
    Dispatchers.resetMain()
}

@Test
fun editProfileViewModel_onSaveClickSuccess() = runTest {
    val navigate: (String, String) -> Unit = mockk(relaxed = true)
    val showSnackbar: (String, SnackbarDuration) -> Unit = mockk(relaxed =
true)

    viewModel.onFirstNameChange("Jmeno Prijmeni")
    viewModel.onLastNameChange("Doe")
    viewModel.profile.value = viewModel.profile.value.copy(id = "123")

    viewModel.onSaveClick(navigate, showSnackbar)

    testDispatcher.scheduler.advanceUntilIdle()
    coVerify(exactly = 1) {
profileService.updateProfile(viewModel.profile.value) }
    coVerify(exactly = 1) {
profileService.updateProfileData(viewModel.profile.value) }

    verify { showSnackbar("Profil byl úspěšně upraven",
SnackbarDuration.Short) }
    verify { navigate(PROFILE_SETTINGS_SCREEN, EDIT_PROFILE_SCREEN) }
}
```

Zdrojový kód 9: Test editace profilu

ZÁVĚR

Cílem bakalářské práce bylo vytvořit mobilní aplikaci pro zlepšení organizace a komunikace ve sportovních týmech s použitím platformy Firebase. Tento cíl byl splněn, aplikace poskytuje základní funkcionality pro správu týmu. Během vývoje jsem si osvojil práci s frameworkem Jetpack Compose, který mi umožnil vytvořit přehledné uživatelské rozhraní i bez větších zkušeností z návrhem UI. Zároveň jsem prohloubil své znalosti platformy Firebase, zejména ohledně služby Cloud Functions. Také jsem se seznámil s knihovnami pro různé typy testování Android aplikací.

V budoucnu by se do aplikace mohly přidat další vlastnosti, jako například správa hřišť a soupeřů nebo rozšíření chatu tak, aby byl dostupný pro jednotlivé události. Co se týče dalších služeb Firebase, tak by se dalo využít uložiště Cloud Storage, například pro ukládání profilových fotek uživatelů a znaků týmů. Nebo služba Firebase Cloud Messaging pro zasílání notifikací uživatelům. Aby aplikaci mohl využívat větší počet uživatelů, mohla se převést na multiplatformní aplikaci. K tomu by mohla být využita technologie Kotlin Multiplatform. Mohl bych tak vytvořit webovou aplikaci, kterou často poskytují podobné platformy pro správu týmu.

POUŽITÁ LITERATURA

[1] LACKO, Ľuboslav. Mistrovství - Android. Brno: Computer Press, 2017, 648 s. ISBN 978-80-251-4875-4.

[2] Android's Kotlin-first approach. *Android Developers* [online]. 2024, 2024-01-30 [cit. 2024-03-10]. Dostupné z: https://developer.android.com/kotlin/first.

[3] Application fundamentals. *Android Developers* [online]. 2023 [cit. 2024-03-19]. Dostupné z: https://developer.android.com/guide/components/fundamentals/.

[4] Android Developers. *App architecture* [online]. 2023 [cit. 2024-03-19]. Dostupné z: https://developer.android.com/topic/architecture/intro.

[5] SPÄTH, Peter. *Pro Android with Kotlin: Developing Modern Mobile Apps*. Berlin: Springer, 2018. ISBN 978-1484238196.

[6] KLUZ, Konrad. Flutter vs Native – What To Choose? Comparison of Flutter vs Android and iOS. *CrustLab* [online]. 2024 [cit. 2024-11-24]. Dostupné z: https://crustlab.com/blog/flutter-ios-android-comparison-overview-kotlin-swift-dart/.

[7] Kotlin: A modern and expressive programming language for everyone. *Constructor University* [online]. 2023 [cit. 2024-11-24]. Dostupné také z: https://constructor.university/blog/kotlin-programming-language.

[8] An Overview of Firebase BaaS. *Back4app* [online]. 2022 [cit. 2024-12-01]. Dostupné z: https://blog.back4app.com/backend-as-a-service-firebase/.

[9] Firebase Pricing. *Firebase* [online]. 2024 [cit. 2024-11-30]. Dostupné z: https://firebase.google.com/pricing.

[10] Firebase Documentation. *Firebase* [online]. 2024 [cit. 2024-02-19]. Dostupné z: https://firebase.google.com/docs.

[11] Kotlin coroutines on Android. *Android Developers* [online]. 2024 [cit. 2024-12-05]. Dostupné z: https://developer.android.com/kotlin/coroutines.

[12] URBAŃCZYK, Marek. Lekce 1 - Úvod do Jetpack Compose. *Itnetwork.cz* [online]. 2024 [cit. 2024-12-01]. Dostupné z: https://www.itnetwork.cz/kotlin/compose/uvod-do-jetpack-compose.

[13] Compose layout basics. *Android Developers* [online]. 2024 [cit. 2024-12-02]. Dostupné z: https://developer.android.com/develop/ui/compose/layouts/basics.

[14] *Týmuj* [online]. 2024 [cit. 2024-11-12]. Dostupné z: https://tymuj.cz/.

[15] Přes Týmuj se k zápasům svolávají tisíce sportovních týmů. Česká aplikace teď získává miliony. *CzechCrunch* [online]. 2023 [cit. 2024-11-12]. Dostupné z: https://cc.cz/pres-tymuj-se-k-zapasum-svolavaji-tisice-sportovnich-tymu-ceska-aplikace-ted-ziskava-miliony/.

[16] *PlayerPlus* [online]. 2024 [cit. 2024-11-12]. Dostupné z: https://player.plus/.

[17] *Heja* [online]. 2024 [cit. 2024-11-12]. Dostupné z: https://heja.io/.

[18] *Team App* [online]. 2024 [cit. 2024-11-12]. Dostupné z: https://www.teamapp.com/.

[19] Android Distribution Chart. *Composables* [online]. 2024 [cit. 2024-11-16]. Dostupné z: https://composables.com/android-distribution-chart.

[20] ViewModel overview. *Android Developers* [online]. 2024 [cit. 2024-12-06]. Dostupné z: https://developer.android.com/topic/libraries/architecture/viewmodel.

[21] Singleton. *Algoritmy.net* [online]. 2016 [cit. 2024-12-06]. Dostupné z: https://algoritmy.net/article/1326/Singleton.

[22] Navigation with Compose. *Android Developers* [online]. 2024 [cit. 2024-12-07]. Dostupné z: https://developer.android.com/develop/ui/compose/navigation.

[23] Transactions and batched writes. *Firebase* [online]. 2024 [cit. 2024-12-05]. Dostupné z: https://firebase.google.com/docs/firestore/manage-data/transactions#transactions.

[24] Kotlin flows on Android. *Android Developers* [online]. 2024 [cit. 2024-12-02]. Dostupné z: https://developer.android.com/kotlin/flow.

[25] Intents and intent filters. *Android Developers* [online]. 2024 [cit. 2024-11-29]. Dostupné z: https://developer.android.com/guide/components/intents-filters.

[26] Use test doubles in Android. *Android Developers* [online]. 2024 [cit. 2024-11-30]. Dostupné z: https://developer.android.com/training/testing/fundamentals/test-doubles.

[27] Fundamentals of testing Android apps. *Android Developers* [online]. 2024 [cit. 2024-11-30]. Dostupné z: https://developer.android.com/training/testing/fundamentals.

[28] Getting Started With Cloud Firestore for iOS. *Envato Tuts+* [online]. 2018 [cit. 2024-12-08]. Dostupné z: https://code.tutsplus.com/getting-started-with-cloud-firestore-for-ios--cms-30910t.

[29] COELHO, Marina. Building an Android app with Jetpack Compose and Firebase. The *Firebase Blog* [online]. 2022 [cit. 2024-12-05]. Dostupné z: https://firebase.blog/posts/2022/04/building-an-app-android-jetpack-compose-firebase.

[30] The activity lifecycle. *Android Developers* [online]. 2024 [cit. 2024-03-19]. Dostupné z: https://developer.android.com/guide/components/activities/activity-lifecycle.

[31] Firebase Authentication. *Firebase* [online]. 2024 [cit. 2024-11-30]. Dostupné z: https://firebase.google.com/docs/auth.

[32] Cloud Firestore. *Firebase* [online]. 2024 [cit. 2024-12-01]. Dostupné z: https://firebase.google.com/docs/firestore.

[33] Cloud Firestore Data model. *Firebase* [online]. 2024 [cit. 2024-12-01]. Dostupné z: https://firebase.google.com/docs/firestore/data-model.

[34] Firebase Security Rules. *Firebase* [online]. 2024 [cit. 2024-12-01]. Dostupné z: https://firebase.google.com/docs/rules.

[35] Cloud Functions for Firebase. *Firebase* [online]. 2024 [cit. 2024-12-01]. Dostupné z: https://firebase.google.com/docs/functions.

[36] Delete data with a Callable Cloud Function. *Firebase* [online]. 2024 [cit. 2024-12-05]. Dostupné z: https://firebase.google.com/docs/firestore/solutions/delete-collections.

[37] Schedule functions. *Firebase* [online]. 2024 [cit. 2024-12-05]. Dostupné z: https://firebase.google.com/docs/functions/schedule-functions?gen=2nd.

[38] State and Jetpack Compose. *Android Developers* [online]. 2024 [cit. 2024-12-05]. Dostupné z: https://developer.android.com/develop/ui/compose/state.

SEZNAM PŘÍLOH

PŘÍLOHA A*:* Mobilní aplikace

Obsahuje projekt mobilní aplikace pro organizaci týmu. Projekt je uložen v archivu s názvem CizekM_MobilniAplikace _JP_2024.zip.