VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

MOBILNÍ APLIKACE PRO VYTVÁŘENÍ VÝZEV A ÚČAST V NICH

MOBILE APPLICATION FOR CREATING CHALLENGES AND PARTICIPATING IN THEM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **TOMÁŠ PLACHÝ**
AUTHOR

VEDOUCÍ PRÁCE **prof. Ing. ADAM HEROUT, Ph.D.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Plachý Tomáš**
Program: Informační technologie
Název: **Mobilní aplikace pro vytváření výzev a účast v nich**
Mobile App for Making Challenges and Taking Part in Them
Kategorie: Uživatelská rozhraní

Zadání:

1. Vyhledejte a analyzujte existující mobilní aplikace pro vytváření výzev mezi uživateli.
2. Seznamte se s problematikou multiplatformního návrhu a vývoje mobilních aplikací.
3. Navrhněte aplikaci, která uživateli umožní vytvořit výzvu a sdílet ji se svými přáteli a sledovat její plnění.
4. Implementujte navrženou aplikaci.
5. Testujte aplikaci v provozu a na vhodné skupině uživatelů a iterativně vylepšujte její funkčnost a uživatelskou zkušenost.
6. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu; vytvořte plakátek a krátké video pro prezentování projektu.

Literatura:

- LEBENSOLD, Jonathan. React native cookbook: bringing the web to native platforms. Sebastopol: O'Reilly, 2018, ix, 162 stran. ISBN 978-1-491-99384-
- WOOD, L. E. User interface design. Boston: CRC Press, 1998, 312 s. ISBN 0-8493-3125-

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 4, značné rozpracování bodu 5.

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

Vedoucí práce: **Herout Adam, prof. Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 28. května 2020
Datum schválení: 1. listopadu 2019

Abstrakt

Tato práce se zabývá problematikou návrhu, implementace a testování mobilní aplikace pro operační systémy iOS a Android.

V práci jsem vyvinul aplikaci, která uživatelům umožňuje vytvořit výzvu, pozvat do ní přátele a sledovat jejich postup ve výzvě, což může mít za následek vyšší motivaci ke splnění výzvy pro daného uživatele.

Aplikace byla navržena a vyvíjena s důrazem na názory uživatelů. Nejdříve jsem vytvořil prototyp, který mi posloužil ke zjištění toho, jaké funkce jsou pro uživatele klíčové. Na základě těchto informaci jsem poté vytvořil oficiální verzi aplikace a tu iterativně vylepšoval s ohledem na zpětnou vazbu od testerů.

Abstract

This thesis deals with the problematics of designing, implementing and testing of mobile application for operationg systems iOS and Android.

The main work of this thesis is development of an application, which allows its users to create challenges and invite their friends to participate in them. Users can also watch the progress of other participants in a challenge, which can give themselve more motivation to complete the challenge.

The application was designed and developed with focus on opinions of its users. I have started by developing a prototype in order to find out, which functions do the users find the most crucial. Thereafter I created the first official version, which was subsequently iteratively improved with regard to feedback from testers.

Klíčová slova

mobilní aplikace, multiplatformní aplikace, vývoj, React Native, Firebase, iOS aplikace, Android aplikace, výzva, tvorba výzev, uživatelské testování, vývoj zaměřený na uživatele

Keywords

mobile application, multiplatform aplication, application development, React Native, Firebase, iOS application, Android application, challenge, create challenge, user testing, user-centered development

Citace

PLACHÝ, Tomáš. *Mobilní aplikace pro vytváření výzev a účast v nich.* Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Adam Herout, Ph.D.

Mobilní aplikace pro vytváření výzev a účast v nich

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana profesora Adama Herouta. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Tomáš Plachý
28. května 2020

Poděkování

Rád bych poděkoval panu profesoru Heroutovi za odborné vedení a konstruktivní kritiku této práce.

Obsah

Kapitola 1

Úvod

Cílem této práce je popsat postup výběru technologií, návrhu, vývoje a uživatelského testování při vytváření aplikace pro tvorbu výzev a účast v nich. V průběhu celého vývoje se budu řídit hlavně názory uživatelů a v závislosti na nich budu aplikaci vylepšovat a přepracovávat.

Ve zkratce jde v aplikaci především o to, aby se uživatel mohl motivovat k provádění nějaké činnosti tím, že z ní udělá skupinovou výzvu. Klíčovým prvkem je tedy možnost srovnat se s ostatními a nejlépe s lidmi, kteří jsou pro nás důležití – tedy s našimi přáteli. Proto je nutné, aby aplikace byla multiplatformní a uživatel mohl do výzvy pozvat opravdu naprostou většinu lidí, které zná, a ne jen ty, kteří s ním sdílejí operační systém na telefonu. Na konci výzvy nečeká žádná garantovaná odměna a je nesmyslné dělat z ní kontrolovanou soutěž. Výzvy tedy nebudou omezené jen na činnosti, které může chytrý telefon měřit a uživatel bude zadávat svůj postup do aplikace ručně. A jelikož je jeho hlavní odměnou pocit zadostiučinění po splnění výzvy, nemá důvod podvádět.

V následujících kapitolách nejdříve ověřím, o jaké funkce by měli potencionální uživatelé zájem a zda je pro takovou aplikaci na trhu místo. Poté vyberu technologie, které budu k vývoji aplikace používat a sestavím první návrh aplikace. Hlavní část práce se bude ale zabývat vývojem aplikace zaměřeným na uživatele. Sestavím tedy skupinu testerů, jejímž členům aplikaci co nejdříve zpřístupním. V závislosti na jejich zpětné vazbě budu potom své řešení přepracovávat. Nakonec popíšu nejdůležitější oblasti implementace a zhodnotím naplnění cílů práce.

Kapitola 2

Má smysl vyvinout aplikaci na tvorbu výzev a účast v nich?

V této kapitole shrnu vlastnosti aplikace pro tvorbu výzev a účast v nich (dále jen Challenge Master), které jsem sestavil nejen na základě vlastní představy o podobě aplikace, ale i na základě nápadů a názorů potenciálních uživatelů, se kterými jsem svůj plán vyvinout aplikaci konzultoval. Dále prozkoumám existující řešení, která tyto požadavky alespoň částečně splňují a nakonec vyvodím závěr o tom, jestli je pro naši aplikaci na trhu místo a jestli má naději na získání uživatelů.

2.1 Požadavky na aplikaci

Základním požadavkem na aplikaci Challenge Master je možnost vytvářet výzvy a zvát do nich ostatní uživatele. K tomu se pojí také potřeba reagovat na pozvánky od ostatních uživatelů a zobrazovat seznam výzev, kterých se uživatel účastní. Je také nutné uživatele co nejméně omezovat v tom, koho do výzvy může pozvat. Aplikace proto musí být multiplatformní (minimálně pro operační systémy Android a iOS) a dostupná zdarma. Klíčovou vlastností aplikace je také možnost uživatele motivovat sama sebe pomocí srovnávání se s ostatními účastníky ve výzvě. To s sebou nese požadavek na zobrazení detailu výzvy, ve kterém budou dostupné jak informace o cíli výzvy, tak informace o ostatních účastnících a jejich postupu ve výzvě. Požadavky vyplývající z tohoto odstavce lze shrnou v následujícím seznamu.

- Možnost tvořit nové výzvy
- Možnost zvát do výzev ostatní uživatele
- Možnost zjistit, jak si ve výzvě vedou ostatní účastníci
- Možnost sdílet vlastní postup ve výzvě
- Aplikace musí být dostupná jak pro iOS tak pro Android
- Uživatel nesmí být nijak omezen v tom, co může být předmětem výzvy
- Aplikace musí být zcela zdarma

2.1.1 Typy výzev

Ještě je nutné zamyslet se nad samotnými výzvami. Postup účastníků se bude u výzvy „Kdo dřív uběhne 50km“ skládat z jiných dat než například u výzvy „Dělej další 2 týdny 50 kliků denně“. Proto jsem definoval následující tři typy výzev a jako jednotku pro měření postupu zvolil „opakování“, pod níž si lze představit jakoukoliv činnost a je možné ji použít pro všechny typy výzev.

- **Periodická** výzva je taková, ve které se má nějaká činnost opakovat po určitých jednotkách času (například každý den).
- **Dosáhni cíle** je typ výzvy, ve kterém se soutěží o to, kdo jako první dosáhne určitého počtu opakování.
- **Nejvíce opakování** je typ výzvy, ve kterém jde o to splnit co nejvíce opakování před tím, než výzva skončí.

2.2 Existující řešení

V této kapitole popíšu aplikace, které alespoň z části splňují výše zmíněné požadavky. Naprostá absence podobných aplikací na trhu se může zdát jako příležitost pro náš produkt, ale nese to s sebou i riziko, že o takovou aplikaci není zájem a proto ji nikdo nevyvinul. Naopak pokud již existuje přehršel podobných aplikací, ztrácí vývoj dalšího duplicitního řešení smysl. Ideální by bylo dojít ke zjištění, že již existuje několik aplikací zabývající se výzvami, ale žádná z nich se na problém nedívá z naší perspektivy a nenabízí uživatelům možnosti, které považujeme za klíčové.

Pro operační systémy iOS a Android je dostupná řada aplikací zabývajících se výzvami. Ve většině případů ale aplikace obsahují jen předem definované výzvy a uživatel je nemůže sám vytvářet. Často se vyskytují i aplikace pro změnu životního stylu, které umožňují uživatelům vytvořit činnost, kterou chtějí provozovat a aplikace jim ji potom připomíná. V těchto aplikacích ale chybí možnost srovnat se s ostatními uživateli a sledovat, jak nový zvyk dodržují.

Naší představě jsou nejvíce podobné aplikace popsané v kapitolách 2.2.1 a 2.2.2, které umožňují jak tvorbu vlastních výzev, tak srovnávání se s ostatními uživateli.

2.2.1 Challenge Achieved – Motivation, Goals, Habits

Tato aplikace umožňuje uživatelům vytvářet nové výzvy a účastnit se výzev veřejných. Dokonce výzvy rozděluje na tři typy podobně, jak to dělám já v kapitole 2.1. Aplikace má navíc přesah v podobě prvků sociální sítě, jelikož v ní člověk může sdílet příspěvky a postup ve výzvách a řadí uživatele do globálního žebříčku. Celkově se jedná o velmi propracovanou aplikaci, která nabízí řadu předdefinovaných výzev a může sloužit i pro zaznamenávání splněných úkolů například v rámci skupinového projektu. Aplikaci si stáhlo přes deset tisíc uživatelů.

Požadavky pro naši aplikaci ale nesplňuje ve dvou klíčových bodech. Zaprvé není dostupná pro operační systém iOS a tudíž limituje uživatele v tom, s kým mohou výzvu sdílet. Zadruhé se vývojáři nesoustředili na intuitivní vyhledávání a zvaní ostatních uživatelů (tuto službu nabízí jen pomocí sdílení URL výzvy). Dalo by se jí také vytknout to, že je příliš komplikovaná a jejím středobodem nejsou výzvy daného uživatele, ale spíše sociální prvky aplikace.

2.2.2 Habitus

Co se služeb týče, nabízí tato aplikace stejné možnosti, jako ta v kapitole 2.2.1. Je však dostupná pro oba operační systémy, které nás zajímají a uživatelské rozhraní je velmi přehledné a intuitivní. Budí také dojem, že ústředním bodem aplikace je uživatel a jeho výzvy, i když nabízí i možnost sdílení příspěvků. Navíc umožňuje intuitivnější přidávání uživatelů do výzvy a dělí výzvy do různých kategorií.

Aplikace je relativně mladá (má mezi pěti a deseti tisíci stažení) a je zatím v bezplatném módu, který však platí jen pro prvních několik tisíc uživatelů a určité služby aplikace se časem zpoplatní. Tím pádem nesplňuje požadavek na to neomezovat uživatele ve sdílení aplikace, jelikož ne každý bude ochoten za podobnou aplikaci zaplatit.

2.3 Odůvodnění vývoje nové aplikace

Existence aplikací zabývajících se výzvami je dobrým znamením toho, že je o tento typ služby zájem. Zároveň jich však není na trhu tolik, aby nebylo místo pro naše řešení.

Některé z dostupných aplikací sice splňují větší část našich požadavků, ale žádná z nich je nesplňuje všechny. Aplikace Challenge Master proto nebude duplicitní a bude mít šanci najít si vlastní uživatelskou základnu.

Kapitola 3

Výběr vhodných technologií pro vývoj multiplatformní mobilní aplikace

V této kapitole se budu zabývat výběrem technologií, které posléze použiji k vývoji samotné aplikace Challenge Master. Nejdříve stručně popíši jednotlivé možnosti pro implementaci frontendu a backendu a na závěr odůvodním technologie, které jsem se rozhodl pro tento projekt použít. Tyto technologie se rapidně vyvíjejí (k čemuž mnohdy přispívá celá komunita programátorů) a dostupné tištěné zdroje jsou většinou zastaralé už v době vydání. Proto jsem v této kapitole čerpal i informace z internetových zdrojů.

3.1 Frameworky vhodné pro vývoj multiplatformních aplikací

Při řešení problému vývoje multiplatformní mobilní aplikace můžeme volit z několika přístupů (a potažmo z frameworků, které se těmito přístupy řídí). Rozdíly mezi nimi a jejich jednotlivé výhody a nevýhody rozeberu níže. Ve zkratce se ale jedná o to, jakou úroveň abstrakce nad nativním kódem zvolíme a jak se bude kód překládat.

U jednotlivých přístupů rozeberu některé technologie, které daný přístup využívají. Vzhledem ke kontextu práce budu vybírat jen technologie vhodné pro vývoj pro operační systémy Android a iOS s ohledem na aktuální trendy.

Definice pojmů

- **Framework** je podpůrná platforma pro vývoj softwarových projektů.
- **Nativní kód** [3] je kód zapsaný v určitém programovacím jazyce, který je prováděn specifickým procesorem. V kontextu této práce se jedná o kód překládaný do strojových instrukcí operačního jazyka iOS nebo Android a prováděný procesorem na mobilním zařízení.
- **Hybridně-nativní kód** [14] je kód v libovolném programovacím jazyce, který je nejprve alespoň z části přeložen do nativního kódu, a až pak přeložen do strojových instrukcí a vykonán.

- **Hybridně-webový kód** [14] je kód, který se nepřekládá do nativního kódu, ale běží ve webovém prostředí na určité platformě. V kontextu této práce se jedná o platformy iOS a Android.
- **Progresivní webová aplikace** [13] je taková, která se přizpůsobí zařízení, na kterém je prohlížena, ale běží ve webovém prohlížeči.

3.1.1 Nativní přístup

Vyvíjet aplikace nativně [3] znamená naprogramovat aplikaci zvlášť pro každou platformu v jejím nativním kódu. Tento přístup umožňuje nejrychlejší běh aplikace, jelikož neobsahuje žádnou nadbytečnou abstrakci nad nativním kódem. Aplikaci je tedy možné optimalizovat přímo pro daný operační systém. Nativní přístup nám také dává největší kontrolu nad vzhledem uživatelského prostředí a nad funkcemi hardwaru, na kterém operační systém běží.

Tyto možnosti s sebou však nesou také nevýhody. Jednu aplikaci je nutné naprogramovat dvakrát (zvlášť pro iOS a Android), což zvyšuje časovou náročnost projektu. Nativní programování také vyžaduje komplexnější znalost operačního systému a jeho možností.

Nativní přístup je tedy vhodný hlavně pro projekty citlivé na výkonost aplikace (jako jsou například mobilní hry) a pro aplikace vyvíjené organizovaným týmem programátorů, který obsahuje specialisty na každý operační systém.

Java/Kotlin a Swift/Objective-C

V našem případě máme s tímto přístupem jen jednu možnost a to zvolit jazyk Java nebo Kotlin a Android Studio jakožto IDE (Integrated Development Environment) pro vývoj pro Android a jazyk Swift nebo Objective-C a XCode IDE pro vývoj pro iOS.

3.1.2 Hybridně-nativní přístup

Hybridně nativní aplikace [14] je tvořena nativním kódem a kódem v jazyce JavaScript, který s tímto nativním kódem komunikuje. Vývojář pracuje prakticky výhradně v jazyce JavaScript a frameworky, které jsou zmíněné níže, se postarají o překlad na nativní komponenty a o komunikaci s nimi. Hlavní výhodou tohoto přístupu je možnost použít jeden předpis v jazyce JavaScript pro vytvoření aplikace jak pro iOS tak pro Android, což může vést k úspoře času potřebného na vývoj aplikace.

Přidaná režie pro vytváření a obsluhu nativních komponent má však za následek zpomalení aplikace. Vývojář je také limitován tím, do jaké míry mu dovolí daný framework upravovat komponenty, jelikož k nim přistupuje jen přes API (Application Programming Interface) a nevytváří je přímo nativně. Je také třeba řešit problém konzistence vzhledu aplikace, jelikož nativní komponenty vygenerované z jednoho předpisu mohou vypadat v různých operačních systémech různě.

Hybridně-nativní přístup je tedy vhodný pro projekty s důrazem na rychlý a levný vývoj. Je také vhodný pro vývojáře bez zkušeností s nativním vývojem nebo při vývoji jednoduchých a výkonnostně nenáročných aplikací.

React Native

React Native [16] je jedním z nejoblíbenějších frameworků pro vývoj pro Android a iOS. Mezi jeho hlavní přednosti patří zaměření na znovupoužitelnost komponent a možnost

naprogramování části aplikace nativně (čímž můžeme optimalizovat výkon). React Native také nabízí funkci Fast refresh, která umožňuje vývojářům vkládat nový kód do běžící aplikace bez nutnosti znovu přeložit celou aplikaci.

Vývoj tohoto frameworku zaštiťuje společnost Facebook, ale zároveň se jedná o open-source projekt (tzn. kdokoliv se může zapojit do jeho vývoje) s podporou rozsáhlé komunity.

Expo

Expo [15] je nadstavba nad frameworkem React Native, která usnadňuje vývoj a testování univerzálních aplikací z jednoho předpisu v jazyce JavaScript.

Hlavní výhodou frameworku Expo je možnost spustit aplikaci na fyzickém mobilním zařízení (podporuje jak operační systém Android, tak iOS) bez nutnosti vydání aplikace – stačí si jen stáhnout aplikaci Expo CLI a naskenovat QR kód. Expo také nabízí řadu hotových řešení pro práci s hardwarem telefonu nebo možnost aktualizovat již vydanou aplikaci automaticky po tom, co ji uživatel spustí (tedy bez nutnosti instalace nové verze z obchodu).

Expo však neumožňuje naprogramovat určité části aplikace nativně (programátor nemá k nativním souborům vůbec přístup) a hotová aplikace zabírá více místa, než by bylo potřeba, protože obsahuje veškerá hotová řešení z balíčku Expo Software Development Kit (i ta, která se v aplikaci nepoužívají).

NativeScript

NativeScript [2] je framework umožňující vývojářům naprogramovat aplikaci jako webovou stránku (například s využitím frameworků Angular, Vue nebo TypeScript) bez toho, aby se v telefonu musela zobrazovat ve webovém prostředí. Aplikace se tedy díky NativeScriptu bude chovat jako hybridně-nativní, ale při vývoji lze využít zkušeností z tvorby webových stránek. Ke spouštění a testování aplikace v průběhu vývoje lze použít NativeScript Sidekick, na kterém lze emulovat mobilní zařízení. Podobně jako ostatní frameworky je i NativeScript open-source. Nemá ale podporu tak rozsáhlé komunity jako třeba React Native.

Xamarin

Xamarin [11] je open-source framework pod záštitou společnosti Microsoft. Využívá jazyk C# pro vývoj aplikací pro platformy iOS, Android a Windows a tudíž nabízí silnou typovou kontrolu. Jelikož je Xamarin založen na .NET frameworku, má vývojář k dispozici mnoho užitečných funkcí jako například LINQ (Language Integrated Query). Co se výkonu týče, blíží se Xamarin nativním aplikacím a dovoluje vývojářům používat nativní prvky uživatelského rozhraní (pomocí Xamarin.iOS a Xamarin.Android) nebo definovat společné UI, které se bude překládat na nativní komponenty za běhu (pomocí Xamarin.Forms).

Xamarin je zdarma pro individuální vývojáře, ale pro větší týmy je třeba zakoupit Visual Studio Proffesional, které stojí desítky tisíc ročně. Další nevýhodou je i to, že Xamarin.Forms není úplně odladěný a používá se proto spíše pro vytváření prototypů, než pro vývoj finálních aplikací. A bez Xamarin.Forms je pro vývoj potřeba umět pracovat i s nativními jazyky.

3.1.3 Hybridně-webový přístup

Hybridně webová aplikace [14] je tvořena kódem v jazycích HTML, CSS a JavaScript, který se vykonává uvnitř webového prostředí. Kód se tedy nepřekládá přímo na nativní komponenty, ale aplikace je jen obalena standardizovaným webovým prostředím, ve kterém se kód vykonává. Díky tomu dosáhneme konzistentního vzhledu aplikace na všech platformách. Další výhodou je naprostá kontrola nad uživatelským prostředím, jelikož vývojář není limitován žádným API mapovaným na nativní komponenty. Hybridně-webový přístup přináší i jistotu stability uživatelského prostředí, jelikož ho neovlivní aktualizace operačního systému (a potažmo vzhledu nativních komponent), na kterém poběží.

Nevýhodou tohoto přístupu je zpomalení aplikace přidáním režie webového prostředí a to i oproti hybridně-nativnímu přístupu.

Hybridně-webový přístup je tedy vhodný zvláště pro projekty, ve kterých se vytváří mobilní aplikace pro již existující webovou stránku a lze v aplikaci znovu využít část kódu webové stránky, nebo pokud chceme vyvinou novou aplikaci a máme k dispozici tým zkušených webových vývojářů.

Ionic

Ionic [9] je open-source framework, který staví na frameworcích Apache Cordova a Angular. Díky konzistenci vzhledu webových komponent je vhodný pro vývoj aplikací, které budou vydány nejen pro Android a iOS ale i pro další platformy (Windows, webové prohlížeče...). Proces vývoje aplikací pomocí Ionic je velmi podobný tvorbě webových stránek (využívá HTML, CSS a JavaScript), což umožňuje webovým vývojářům přejít k vývoji mobilních a desktopových aplikací bez nutnosti učit se nové jazyky a postupy.

Flutter

Flutter [8] je framework, který by možná zasloužil kategorii sám pro sebe. Nevyužívá totiž k vykreslení grafických komponent webového prostředí ani nativních komponent, ale vlastní engine (tzn. ústřední část programu, která je optimalizovaná pro řešení výpočetně náročných operací) optimalizovaný pro 2D aplikace. Flutter používá pro předpis komponent programovací jazyk Dart a nabízí API pro jednotkové a integrační testování.

Vývoj tohoto frameworku zaštiťuje společnost Google, ale zároveň se jedná o open-source projekt podobně jako u frameworku React Native. Oproti němu se zatím nemůže pochlubit podporou tak rozsáhlé komunity.

3.1.4 Progresivní webové aplikace

Progresivní webová aplikace [13] (dále jen PWA) je webová stránka, která se jeví jako aplikace, když je otevřena v prohlížeči na mobilním zařízení (a odkaz na ni si uživatelé mohou uložit i přímo na plochu). Aplikace je tak vyhledatelná přímo v prohlížeči a uživatelé ji nemusí vyhledat a stáhnout z App Store nebo Google Play Store. Odkaz na PWA zabírá výrazně méně úložného prostoru, než tradiční aplikace a použití PWA namísto vydání nativní aplikace je také způsob, jak obejít poplatky za vývojářskou licenci a nákupy v aplikaci, které si účtují App Store a Google Play Store.

Nevýhodou PWA je, že některé prohlížeče v čele se Safari (výchozí prohlížeč na zařízeních s iOS) nedávají přístup ke všem funkcionalitám, které mohou využívat nativní aplikace, jako například push notifikace, synchronizace na pozadí nebo využití webového manifestu.

Další nevýhodu představuje to, že si uživatel musí sám uložit odkaz na plochu, což je funkce, o které nemusí někteří uživatelé vědět.

Vyvíjet PWA je tedy vhodné hlavně pro webové stránky, které už vešly v povědomí a mají stálou návštěvnost, u projektů, které se snaží minimalizovat poplatky obchodům s aplikacemi a nebo pokud je pro vývojáře důležité, aby byla jejich služba vyhledatelná v prohlížeči.

Frameworky vhodné pro vývoj PWA

Vývoj PWA znamená jen přeprogramování webové stránky tak, aby se na mobilním zařízení chovala jinak než v prohlížeči stolního počítače. Za tímto účelem by šly využít všechny frameworky pro vývoj webových aplikací (například Angular, Vue, Ionic, React apod.). Tato práce se však soustředí na vývoj specificky pro mobilní zařízení a proto se jim zde nebudu věnovat.

3.2 Databázové služby

V této kapitole se budu zabývat porovnáním relačního a nerelačního přístupu k ukládání dat [10]. Tuto problematiku nastíním jen stručně, jelikož je Challenge Master z pohledu ukládání dat spíše jednoduchá aplikace a způsobu ukládáni dat nebude mít na její běh příliš velký dopad.

3.2.1 Relační databáze

Relační (neboli SQL) databáze pracují s daty organizovanými do podoby tabulek, které se skládají z řádků a sloupců, kde záhlaví každého sloupce reprezentuje atribut tabulky a každý řádek představuje datovou entitu. Jelikož jsou relační databáze dobře zavedené a vyzkoušené, existují standardizované postupy pro práci s nimi, které minimalizují riziko redundance a nekonzistence dat.

Nevýhodou je složitá škálovatelnost relačních databází. S větším počtem záznamů v tabulce se totiž zpomaluje vyhledávání a abychom docílili dřívější rychlosti, musíme vylepšit hardware vybavení serveru nebo databázi rozdělit na několik serverů, což je časově i finančně náročné.

3.2.2 Nerelační databáze

Potenciál nerelačních (neboli NoSQL) databází spočívá v možnosti provádět operace nad velkým množstvím dat, která nejsou propojená komplexními vztahy. NoSQL databáze se dělí podle způsobů ukládání dat a jednotlivé způsoby jsou přizpůsobeny řešení specifických problémů. Mezi nejčastější přístupy patří ukládání dat do souborů ve formátu JSON. Více souborů potom tvoří kolekci, ale soubory v kolekci nemusí mít stejně strukturovaný obsah. Další častý přístup je ukládání dat v podobě dvojice „klíč: hodnota“.

Nevýhodou NoSQL databází může být to, že se pro co nejrychlejší operace nad daty obětovaly zásady a pravidla, která z relačních databází dělají tak spolehlivá úložiště dat. Programátor má tedy zodpovědnost za dodržování konzistence dat, jelikož ji samotná databáze nevynutí.

3.3 Finální výběr

Pro výběr vhodných technologií je třeba brát v potaz několik skutečností týkající se vyvíjené aplikace a týmu programátorů, které na ní pracují. Nejrelevantnější jsou nároky aplikace na výkon a využívání nativních komponent, v druhé řadě je dobré zohlednit zkušenosti a preference programátorů.

Framework

Aplikace Challenge Master bude sloužit především ke statickému zobrazování dat, která se nebudou muset obnovovat s vysokou frekvencí. Načítání dat a vykreslování komponent je závislé na akcích uživatele, který v aplikaci chce data hlavně prohlížet nebo zadávat a v komunikaci s databází budou tím pádem prodlevy v řádech nejméně desetin sekund.

Vývojářský tým tvořím já sám a vyvíjet aplikaci nativně kvůli časové náročnosti tohoto přístupu nepřipadá v úvahu. Jelikož je mým cílem aplikaci vydat na App Store a Play Store, nebudu se ubírat ani cestou progresivních webových aplikací. Osobně bych se rád seznámil s hybridně-nativním vývojem a kvůli podpoře rozsáhlé komunity a oblíbenosti v praxi jsem nakonec zvolil React Native s nadstavbou Expo pro rychlejší vytváření prototypů a snadnější testování.

Databázová služba

Co se týče databázových služeb, rozhodl jsem se pro nerelační přístup k ukládání dat, jelikož se moje aplikace bude měnit v závislosti na zpětné vazbě od uživatelů a nerelační databázi lze jednodušeji rozšiřovat s rostoucím počtem vlastností aplikace.

Specificky jsem se rozhodl pro službu Firebase [6], jelikož nabízí jak dokumentově orientovanou NoSQL databázi Firestore (zdarma do 50 000 čtení souborů za den) tak Autentizaci (zdarma do 10 000 přihlášení za měsíc). Databáze Firestore automaticky vytváří index pro každý soubor, záznam v souboru a pole v seznamu. Firestore navíc garantuje rychlost vyhledávání závislou na počtu výsledků vyhledávání a nezávislou na počtu souborů, ze kterých vybírá, díky čemuž aplikace s rostoucím počtem uživatelů nezpomaluje.

Kapitola 4

Návrh aplikace na tvorbu výzev a účast v nich

V této kapitole rozeberu požadavky na aplikaci, které získám od potenciálních uživatelů. Na základě těchto požadavků sestavím základní use case diagram popisující minimální vlastnosti, se kterými se dá aplikace smysluplně používat. Z tohoto diagramu budou vycházet další kroky při návrhu aplikace, jako je návrh struktury databáze, návrh grafického designu aplikace a potažmo i výběr názvu aplikace. Právě těmto krokům a jim předcházejícímu průzkumu požadavků na funkcionalitu aplikace se budu v této kapitole věnovat.

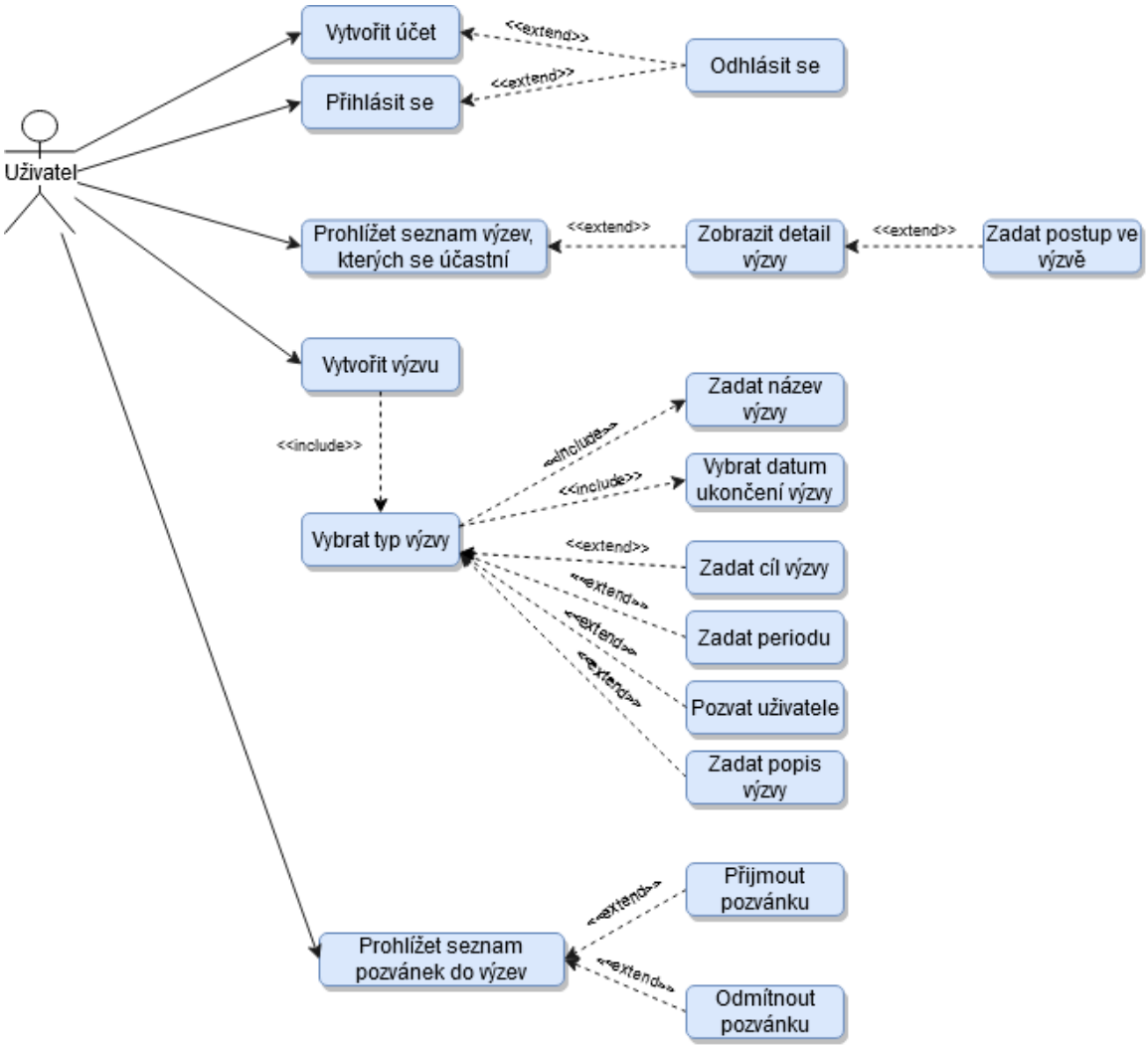
4.1 Výběr vlastností aplikace

Myšlenka vyvinout aplikaci, která by umožnila „hecovat se“ mezi kamarády vznikla z mé vlastní poptávky po této funkcionalitě, kterou neuspokojovala žádná v té době dostupná aplikace. Většinu výzev jsme tedy s kamarády museli realizovat pomocí skupinové konverzace na Facebooku, kde musel každý psát, jestli má už splněno. To bylo ovšem zdlouhavé a mnohdy vedlo i k vypnutí upozornění z dané skupiny.

Rozhodl jsem se tedy prozkoumat názory lidí, kteří by o podobnou aplikaci mohli mít zájem. Tento průzkum jsem neprováděl pomocí dotazníku zaslaného velkému počtu respondentů, protože nemám potřebné prostředky, abych získal odpovědi od několika set subjektů a hlavně mi v tomto případě přijde důležitější kvalita informací získaná od každého dotázaného, než celkový počet dotazovaných. Aplikace je totiž zajímavá hlavně pro lidi, kteří mají chuť překonávat překážky a osobně se rozvíjet, což neplatí pro každého. Sestavil jsem tedy skupinu zhruba deseti kamarádů, z nichž někteří se se mnou na výzvách už dříve účastnili a dialogem s každým z nich jsem zjistil, jaké funkce by podle nich měla aplikace nabízet. Na základě takto získaných informací jsem sestavil minimální use case diagram, jak je vidět na obrázku 4.1. Podrobnější informace o sestavení této testovací skupiny uživatelů a jejich zpětné vazbě k funkcionalitě aplikace v průběhu vývoje naleznete v kapitole 5.1.

4.2 Architektura aplikace

V této kapitole se budu věnovat výběru nejvhodnější architektury [12] pro naši aplikaci. Mobilní aplikace je v principu velmi podobná té webové, nabízí se nám tedy architektura typu klient-server. Jelikož jsem v kapitole 3.3 zvolili službu Firebase, nemusíme se zabývat problematikou výběru mezi centralizovaným systémem (jeden server a jedna databáze) a



Obrázek 4.1: Obrázek znázorňuje akce, které může uživatel v aplikaci provést.

distribuovaným systémem (více serverů a více databází). Firebase vyřeší problém rozšiřování serverů a databází s rostoucím počtem uživatelů za nás – může se změnit cena za využívání těchto služeb, ale na dotazech do databáze se nic nezmění.

Zbývá tedy zvolit správný přístup ke vztahu mezi klientem a serverem. Mobilní aplikace na rozdíl od těch webových nevyužívají server ke stahování celých souborů, které by obsahovaly jak data, tak kód definující GUI. Vzhled uživatelského prostředí mají pevně daný ve zdrojovém kódu, který je stažený na mobilním zařízení. Tloušťka klienta se tedy vztahuje k tomu, jestli se informace, které má aplikace zobrazovat, zpracovávají na serveru a klient data pouze odesílá a přijímá (tenký klient), nebo jestli server slouží hlavně jako databáze a operace s daty se provedou přímo v aplikaci ještě před odesláním či zobrazením (tlustý klient).

Výhodou tenkého klienta je to, že rychlost aplikace je závislá spíše na rychlosti internetového připojení, než na výkonu zařízení, na kterém běží. Za předpokladu, že je postaráno o škálování výkonu serveru s rostoucím počtem uživatelů, lze pomocí tohoto přístupu dosáhnout konzistentní rychlosti aplikace nezávisle na kvalitě hardwaru, na kterém je nainstalována. Challenge Master je ale na výkon velmi nenáročná aplikace a tudíž získává na váze argument ve prospěch tlustého klienta. Pokud totiž zvolíme tento přístup, bude aplikace závislá pouze na službě Firebase. Primární složkou této služby je databáze Firestore a kód na straně serveru, který by se v případě tenkých klientů musel vykonávat, se spouští se zpožděním v řádech desítek sekund. A to je pro naši aplikaci příliš pomalé. Pokud tedy zpracujeme data ještě před odesláním do databáze, nebude potřeba komunikovat s další službou, která by umožnila zpracovávat informace na straně serveru v reálném čase.

Při vývoji aplikace se tedy budu snažit provést veškeré operace nad daty před jejich odesláním do databáze a z aplikace se stane čistě tlustý klient, který využívá server pouze k uložení dat.

4.3 Struktura databáze

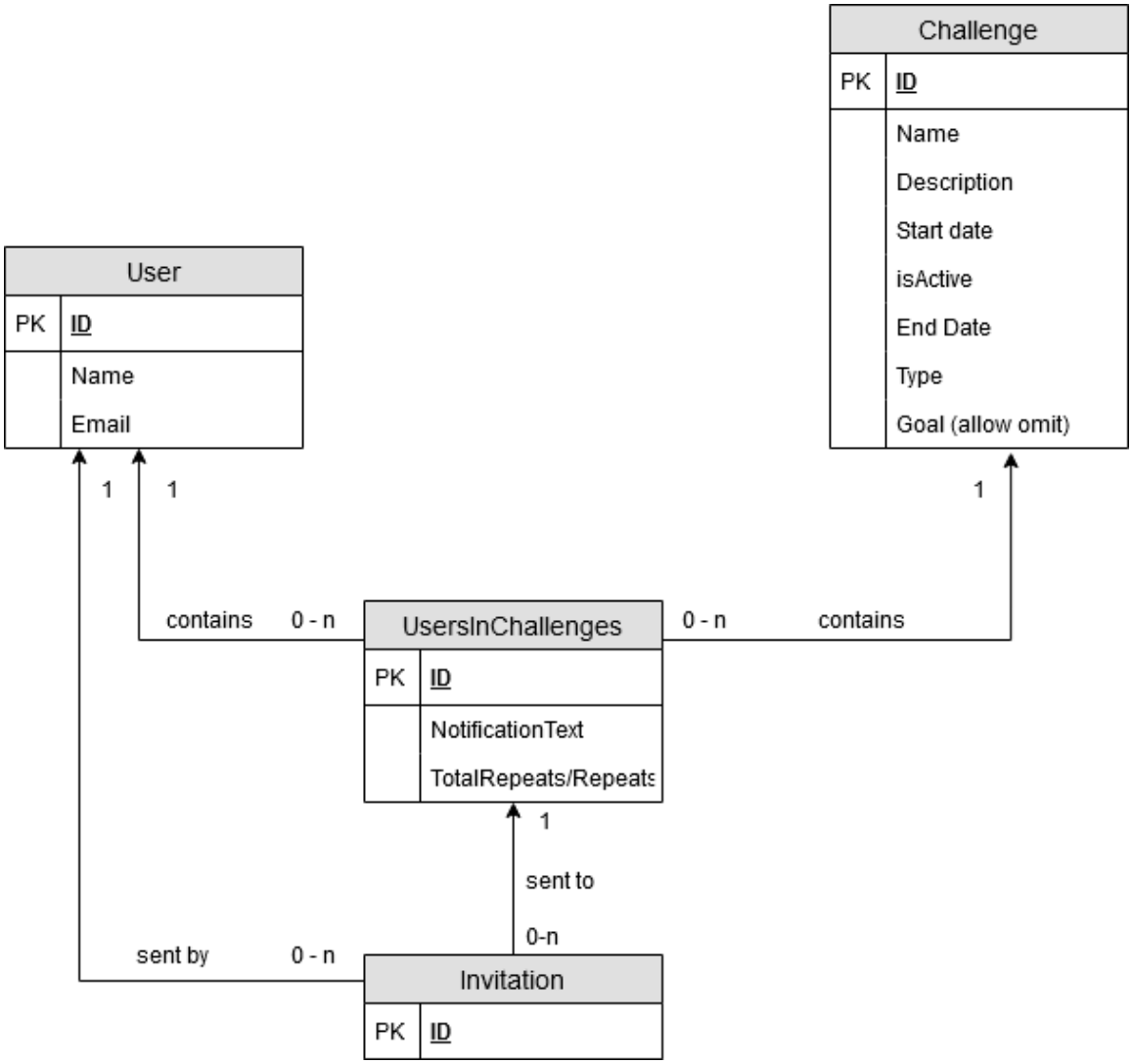
Pro NoSQL databáze neexistuje standardizovaná podoba diagramu pro předpis struktury databáze a proto jsem se na základě use case diagramu (viz obrázek 4.1) rozhodl sestavit ERD (Entity Relationship Diagram) s následujícími sémantickými změnami.

V našem případě třída nereprezentuje tabulku, ale předpis pro soubor v kolekci. Veřejný klíč (na obrázku 4.2 jako PK ID) reprezentuje název souboru v podobě unikátního identifikátoru. Každý atribut třídy představuje klíč v souboru (tedy v objektu ve formátu JSON) a název třídy udává název kolekce těchto souborů. Vztah mezi dvěma třídami představuje položku ve formátu „název kolekce: název souboru“ v souboru, ze kterého vychází šipka udávající orientaci vztahu.

Značka „allow omit“ poukazuje na to, že se daný klíč nemusí vyskytovat ve všech souborech dané kolekce a lomítko mezi dvěma klíči značí, že bude soubor obsahovat vždy jen jeden z klíčů oddělených lomítkem. Je nutné mít ale stále na paměti, že tato pravidla nevynucuje sama databáze a je odpovědností programátora je dodržovat.

Ve Firestore, kterou používám jako NoSQL databázi, se nelze dotázat pouze na část dokumentu. Při úspěšném dotazu do databáze se vždy vrací nejméně jeden celý soubor. Proto je důležité seskupovat do souboru data, která budeme většinou potřebovat všechna zároveň, abychom zbytečně nestahovali nic navíc.

Jak je vidět na obrázku 4.2, objem dat ukládaných pro každou entitu je velice malý, mnohdy jen v řádu desítek bajtů. Proto jsem se rozhodl upřednostnit rychlost dotazování nad dodržením neredundantnosti databáze a upravil jsem ERD tak, aby se pro zobrazení



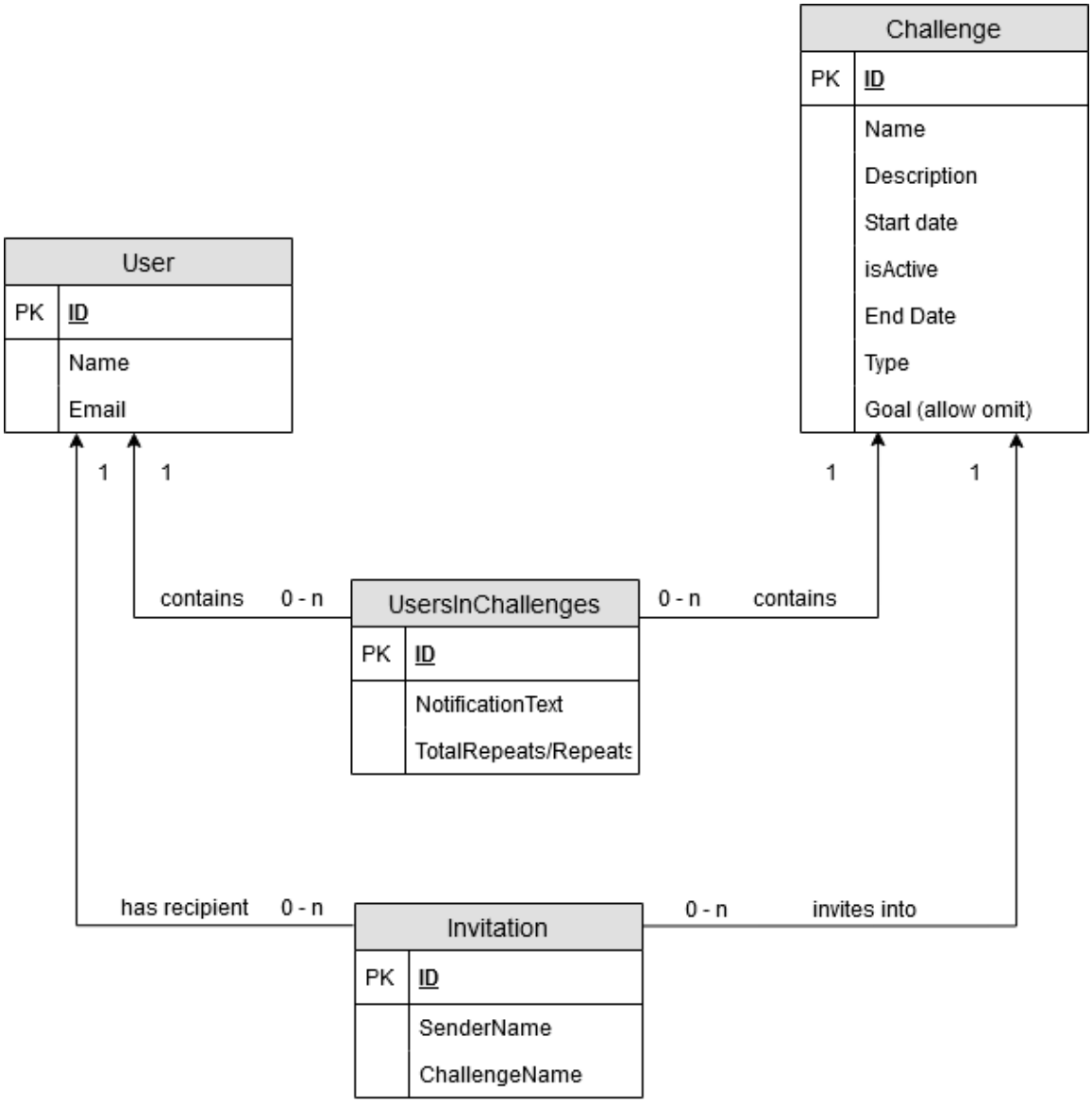
Obrázek 4.2: Obrázek znázorňuje předpis pro soubory a kolekce.

každé pozvánky nemusel stahovat i soubor se všemi informacemi o výzvě a soubor se všemi informacemi o uživateli, který pozvánku odeslal.

Soubory v kolekci Invitation budou podle obrázku 4.3 oproti předchozímu návrhu na obrázku 4.2 obsahovat navíc jméno výzvy a jméno uživatele, který pozvánku odeslal. Obě položky mohou mít velikost maximálně 80 znaků, přičemž jeden znak zabere jeden bajt. Celkem se tedy velikost obsahu jednoho souboru v této kolekci zvýší maximálně o 160 bajtů (z 56 bajtů na 216 bajtů), ale nebudeme muset pro zobrazení každé pozvánky vyhledat a stáhnout jeden souboru ze všech ostatních kolekcí.

Toto řešení podporuje taky fakt, že v průběhu testování aplikace žádný uživatel nezadal uživatelské jméno ani název výzvy delší než 40 znaků. Pozvánky jsou navíc v databázi uchovávány pouze do doby, kdy na ně uživatel zareaguje a poté se smažou.

Mohlo by se zdát, že díky stejným vazbám na soubory v ostatních kolekcích by bylo možné entity UsersInChallenges a Invitation sloučit do jedné. Pozvánek do jedné výzvy ale může jeden uživatel obdržet klidně několik, kdežto účastnit se výzvy může jen jednou a proto musejí tyto entity zůstat oddělené.



Obrázek 4.3: Obrázek znázorňuje změněný obsah souborů v kolekci Invitation.

4.4 Grafický vzhled aplikace

V této kapitole odůvodním výběr loga a základních barev aplikace. Následně představím prvotní návrhy rozložení uživatelského prostředí – pro každou stránku aplikace vytvořím wireframe (návrh GUI zaměřený na funkcionalitu a obsah) a stručně jej popíšu.

Od finálního vzhledu a funkcionalit aplikace se zde uvedené informace mohou lišit, jelikož aplikace procházela (a stále prochází) iterativním vylepšováním na základě zpětné vazby od uživatelů, což je popsáno v kapitole 5.1.

4.4.1 Výběr loga, ikony a základních barev

Logo by se mělo vztahovat buď k názvu, nebo k obsahu aplikace. Z estetických důvodů jsem se místo tvorby loga z iniciál „CH. M.“ snažil vybrat logo, které by evokovalo snahu zakončenou úspěchem. Nakonec jsem zvolil ikonu zlatého poháru (kterou naleznete na obrázku 4.4), jelikož symbolizuje úspěch v nějaké soutěži.

Při výběru vzhledu ikony aplikace jsem se zaměřil na to, aby byla graficky jednoduchá, působila pozitivně a aby v ní dominovalo logo celé aplikace. Výsledkem je ikona s logem na bílém pozadí, kterou můžete vidět na obrázku 4.5.

Obrázek 4.4: Logo aplikace.

Obrázek 4.5: Ikona aplikace.

Při výběru barevné palety pro aplikaci jsem se soustředil na to, aby barvy evokovaly nějaké ocenění či překonanou překážku. Chtěl jsem se také vyhnout použití modré dominantní barvy, protože ji podle mě používá už příliš mnoho aplikací.

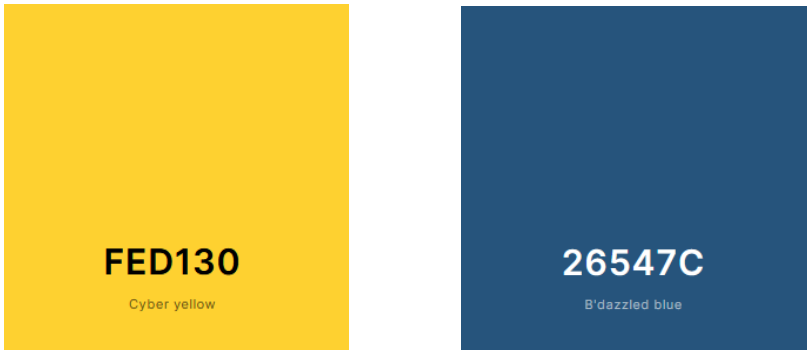
Nakonec jsem zvolil zlatou a modrou, které jsou vyobrazené na obrázku 4.6. Tato barevná kombinace symbolizuje zlatou medaili na modré stuze, nebo například zlatý pohár. K výběru vhodné kombinace jsem použil online nástroj pro vytváření barevných palet Colors[1].

4.4.2 Tvorba wireframů pro jednotlivé stránky

Tato kapitola je rozdělena na několik sekcí, z nichž každá se věnuje stránkám aplikace se společnou funkcí či vlastností (například stránky pro autentizaci uživatele). Tyto sekce zde postupně popíšu pokud možno ve stejném pořadí, v jakém by s nimi přišel do kontaktu uživatel při prvním spuštění aplikace.

Wireframy z této kapitoly použiji ve zjednodušené podobě při vývoji prvního prototypu a je pravděpodobné, že projdou výraznými změnami v návaznosti na zpětnou vazbu od uživatelů. Více o úpravách vzhledu a funkcionalit aplikace v průběhu vývoje najdete v kapitole 5.1.

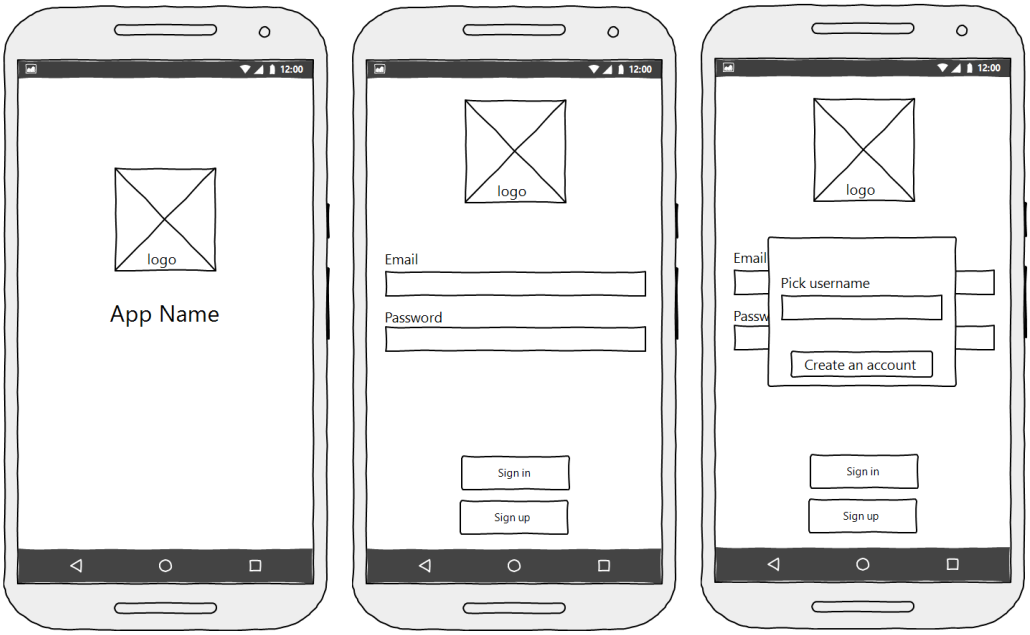
[1] https://coolors.co/



Obrázek 4.6: Primární a sekundární barva aplikace.

Autentizace

Stránky pro autentizaci jsem se snažil tvořit co nejjednodušší a nejgeneričtější, aby uživatel přišel do kontaktu s tím, co už zná z jiných aplikací. Zleva doprava je na obrázku 4.7 vyobrazena splash screen (obrazovka s logem při spouštění aplikace), obrazovka pro zadání údajů potřebných pro přihlášení a ta samá obrazovka s vyskakovacím oknem pro výběr uživatelského jména, které se objeví po stisknutí tlačítka „Sign up“ (tedy pokud chce uživatel vytvořit nový účet).

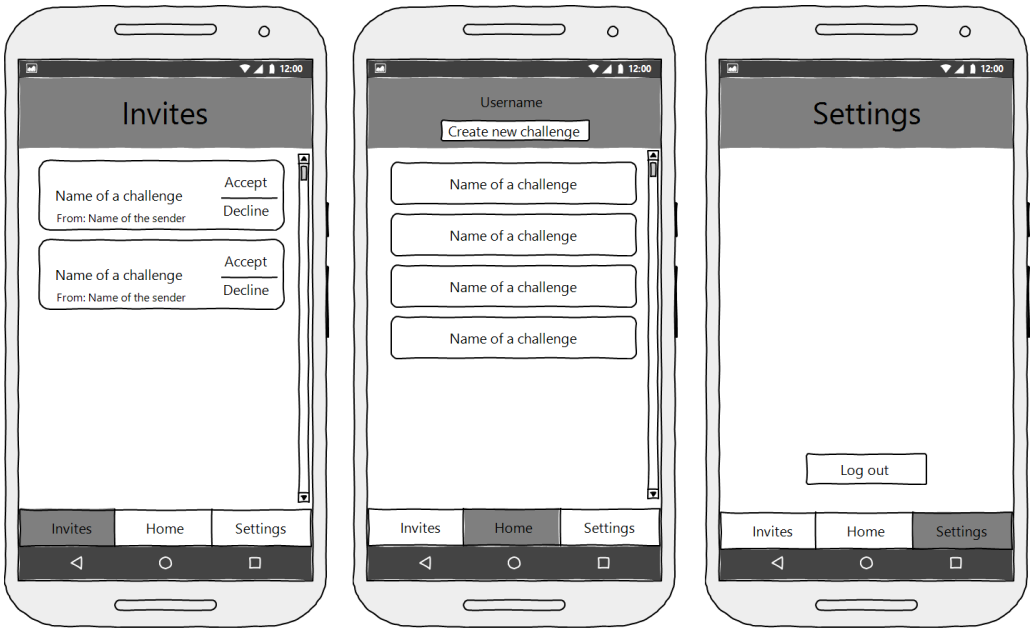


Obrázek 4.7: Obrázek znázorňuje wireframy pro stránky spojené s autentizací.

Domovská stránka, nastavení a přehled pozvánek

Tyto tři stránky vyobrazené na obrázku 4.8 jsou jako jediné dostupné ze spodní lišty sloužící pro navigaci. Vybral jsem je proto, že obsahují nejzákladnější funkce aplikace a je pravděpodobné, že mezi nimi bude uživatel muset často přepínat.

Předpokládaná nejčastější uživatelská akce je výběr výzvy, jejíž detail bude chtít uživatel zobrazit. Proto se stránka se seznamem výzev (na záložce Home) zobrazí uživateli po přihlášení jako první.



Obrázek 4.8: Obrázek znázorňuje wireframy pro hlavní stránky dostupné ze spodní lišty.

Tvorba výzvy

Uživatelské rozhraní pro tvorbu výzev (viz obrázek 4.9) dává uživateli možnost zadat všechna potřebná data pro vytvoření výzvy. Snažil jsem se o co nejmenší počet položek, které je nutné nastavit, aby se stránky nejevily nepřehledné. Do textového pole s nápisem „Invite other users“ může uživatel začít psát jméno a aplikace mu ve vygenerovaném seznamu (který není v návrhu zobrazen) bude nabízet nejlepší shody z databáze uživatelů.

Detail výzvy

Společným prvkem pro všechny typy výzev je záhlaví, které obsahuje základní informace o výzvě a tlačítko pro opuštění výzvy. V čem se detaily jednotlivých typů výzev liší jsou data zobrazovaná u účastníků výzvy (podrobnější popis typů výzev naleznete v kapitole 2.1.1).

U výzvy typu Nejvíce opakování (na obrázku 4.10 uprostřed) je u každého uživatele zobrazen jednoduše počet splněných opakování. U výzvy typu Dosáhni cíle (na obrázku 4.10 vpravo) je k počtu splněných opakování přidána ještě informace o cílovém počtu opakování. Nejodlišnější je pak Periodická výzva (na obrázku 4.10 vlevo), u které je zobrazeno, jestli účastník daný den výzvu splnil.

Obrázek 4.9: Obrázek znázorňuje wireframy stránek pro tvorbu výzev různých typů.

Obrázek 4.10: Obrázek znázorňuje wireframy stránek pro detaily výzev různých typů.

4.5 Výběr názvu pro aplikaci

Jména služeb, společností, produktů a aplikací se dají roztřídit do několika kategorií, z nichž každá má specifické vlastnosti. Pro název aplikace pro tvorbu výzev a účast v nich je klíčové, aby předal informaci o smyslu aplikace, byl snadno zapamatovatelný a neobsahoval slova, která se vyslovují a píší v různých jazycích různě a jejich správné hláskování není všeobecně známé.

4.5.1 Různé kategorie názvů produktů

Názvy existujících produktů lze rozdělit do několika kategorií[17]. Mnohé názvy však spadají do více než jedné kategorie, takže je tento výčet třeba brát pouze jako orientační seznam sloužící pro inspiraci a setřídění myšlenek při výběru názvu pro nový produkt.

Popisná

Popisná jména se vztahují k hlavní službě, kterou aplikace nabízí. Výhodou jmen z této kategorie je to, že předají informaci o tom, co aplikace nabízí, aniž by o ní musel potencionální uživatel předtím cokoliv vědět. Jako příklad lze uvést Autobazar.cz, což je webová stránka umožňující uživatelům prodávat a nakupovat ojeté vozy.

Sugestivní

Pro tuto kategorii jsou typické názvy, které jsou asociované k vlastnostem produktu, ale často jen metaforicky. Jako příklad lze uvést sociální síť Twitter (což v češtině znamená cvrlikat či štěbetat), která svým uživatelům nabízí možnost vyslat svou zprávu do světa, jako by ji opakovali ptáci na stromech.

Bez vztahu k produktu

Do této kategorie spadají jak názvy, jejichž účel je upoutat pozornost a vštípit se do paměti potencionálního konzumenta, spíše než předat jakoukoliv informaci o produktu. Tyto názvy mohou být tvořené jak reálnými slovy (například Apple), tak novotvary (například Xerox).

Název z této kategorie se hodí zejména pro aplikaci podporovanou silnou marketingovou kampaní, která u zákazníků vytvoří asociaci mezi názvem a produktem. Aplikace potom může plně těžit z neotřelosti a jednoduché zapamatovatelnosti, kterou jména z této kategorie nabízí.

4.5.2 Finální výběr názvu aplikace

V průběhu vývoje aplikace jsem s uživateli konzultoval následující verze názvu:

- **Strive** – v angličtině znamená o něco usilovat, mnohdy ve smyslu „usilovat o zlepšení“. Je na pomezí popisného a sugestivního názvu a jedná se o relativně krátké slovo. Většina lidí mimo anglicky mluvící země ale nebude znát přesný význam tohoto slova ani jeho správné hláskování.
- **Enkidu** – tento název je čistě sugestivní (Enkidu byl pomocník Gilgameše v jeho honbě za nesmrtelností v Eposu o Gilgamešovi) a ve většině jazyků se bude psát stejně jako vyslovovat. Většina lidí však o tomto literárním díle nikdy neslyšela. Název by je tudíž mátl a těžko by se pamatoval.

- **Challenge Master** – popisný název, který dá uživateli okamžitě vědět, že se v aplikaci budou soutěživou formou plnit výzvy. Slova jsou sice v angličtině a píší se jinak, než se vyslovují, ale jedná se o dobře známá slovíčka, která se často objevují na internetu tak v názvech televizních pořadů (například Masterchef, Euro Hockey Challenge apod.)

Při výběru finálního názvu pro mou aplikace jsem měl na paměti také následující skutečnosti:

- Moje aplikace nebude podporována žádnou rozsáhlou marketingovou kampaní, ale bude se šířit od prvotních uživatelů k jejich přátelům, které budou chtít pozvat do výzvy. Od těchto přátel zase k jejich přátelům a dál.
- Název by se měl vztahovat k obsahu aplikace, aby byl po ústním doporučení lehce zapamatovatelný.
- Vztah k obsahu aplikace má prioritu nad délkou názvu (většinou platí čím kratší název, tím lepší).

Po zvážení zpětné vazby od uživatelů a výše zmíněných skutečností jsem vybral název **Challenge Master**.

Kapitola 5

Vývoj aplikace na tvorbu výzev a účast v nich

V následujících kapitolách popíšu plán vývoje aplikace, prototyp aplikace, výběr uživatelů pro testování (dále je budu adresovat jako „testery“) a jednotlivé změny v aplikaci, které byly následkem zapracování jejich požadavků. Dále popíšu způsob implementace některých zajímavých oblastí aplikace, problematiku vydání aplikace na Play Store a App Store a uvedu kolik uživatelů se aplikaci podařilo získat a jak ji využívají.

5.1 Vývoj zaměřený na uživatele

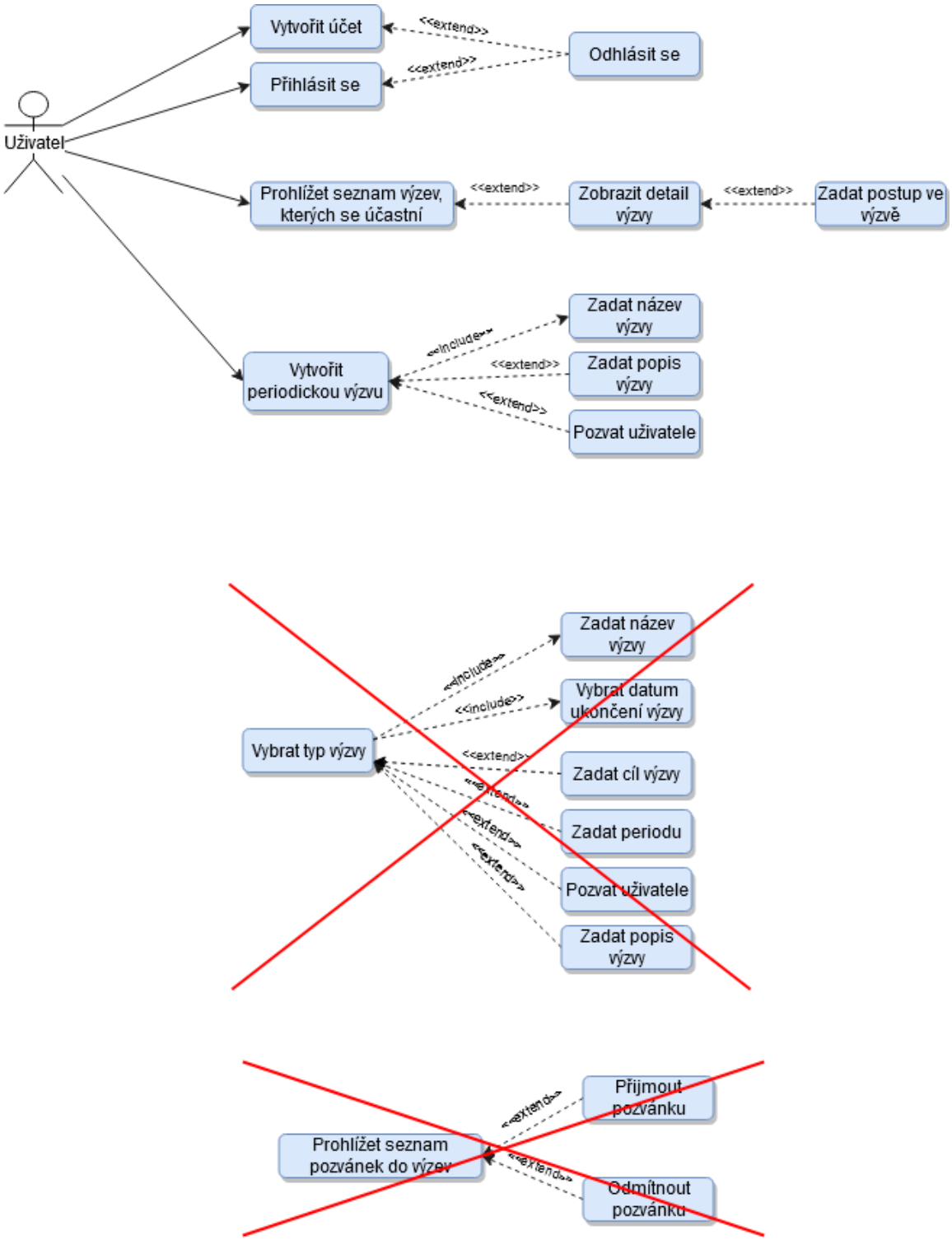
Rozhodl jsem se postavit názory uživatelů na první místo a zvolil jsem způsob vývoje aplikace zaměřený na uživatele [7] (z anglického „User Centered Design“, dále jen UCD). Jedná se o iterativní proces, v jehož každém cyklu dojde k získání zpětné vazby od uživatelů, vyhodnocení těchto informací a upravení aplikace s ohledem na ně. Sestavil jsem plán vývoje, ve kterém jsem měl prvně vytvořit funkční prototyp aplikace pomocí frameworku Expo, protože tento framework umožňuje jednoduchou distribuci aplikace mezi uživatele bez nutnosti aplikaci vydat. Prototyp jsem plánoval v závislosti na požadavcích uživatelů postupně vylepšovat a nakonec ho vydat jako oficiální verzi aplikace.

V této kapitole se budu věnovat postupu ve vývoji aplikace podle výše zmíněného plánu, od kterého jsem se ale musel kvůli neovlivnitelným komplikacím zmíněným v kapitole 5.1.2 odklonit. I přes tyto komplikace jsem se ale stále držel přístupu UCD.

5.1.1 Prototyp

Prototyp aplikace má sloužit hlavně ke sběru informací ohledně toho, jaké funkce má nabízet oficiální verze aplikace a jaké činnosti má co nejvíce zjednodušovat. Je důležité, aby byl prototyp rychle přístupný testerům a aby se mohl agilně vyvíjet, nejlépe bez nutnosti čekat několik dní na schválení nové aktualizace na App Store nebo Play Store. K tomu jsem využil skvělé vlastnosti frameworku Expo, který umožňuje publikovat aplikaci tak, že je dostupná uvnitř aplikace Expo CLI. Mé skupině testerů tedy stačilo stáhnout si Expo CLI a měli okamžitě přístup k prototypu aplikace Challenge Master.

Funkce prototypu jsem oproti požadavkům na aplikaci v kapitole 2.1 omezil, abych mohl zpřístupnit funkční verzi k testování co nejdříve. Rozdíly oproti plánované funkcionalitě jsou zobrazené v use case diagramu na obrázku 5.1.



Obrázek 5.1: Obrázek znázorňuje změny v use case diagramu prototypu oproti návrhu finální aplikace.

5.1.2 Změna frameworku

Po několika týdnech testování aplikace pomocí prototypu bylo na čase začít přidávat funkce a postupně tak splnit všechny požadavky na aplikaci z kapitoly 2.1. Po nasbírání nových zkušeností s jazykem React Native jsem se ale rozhodl prototyp opustit a začít od začátku. K mnoha základním problémům jsem díky novým znalostem chtěl přistupovat jinak, než jak jsem je v prototypu naprogramoval. Nová aplikace mi tedy přišla jako spolehlivější řešení, než měnit fundamentální části prototypu, což by nevyhnutelně vedlo k novým problémům.

Toto rozhodnutí se ukázalo jako šťastné, jelikož jsem zjistil, že komunikace mezi Expem a momentálně nejvíce používaným a rozvíjeným balíčkem pro komunikaci s Firebase (react-native-firebase) prakticky není možná [1]. Místo frameworku Expo jsem tedy novou verzi aplikace založil čistě na frameworku React Native, který umožňuje komunikaci s Firebase a přístup k nativním souborům, což s sebou nese větší kontrolu nad aplikací, ale také více odpovědnosti.

5.1.3 Sestavení skupiny uživatelů pro testování

Aplikaci jsem zpřístupnil skupině lidí již v průběhu vývoje (jak v případě prototypu, tak u finální verze aplikace). Při výběru jednotlivců do této skupiny jsem se soustředil na to, aby to byli typičtí uživatelé mé aplikace. U různorodé skupiny obsahující zástupce více věkových a sociálních kategorií by totiž hrozilo, že část testerů o aplikaci vůbec nebude mít zájem a neposkytnou mi tak kvalitní zpětnou vazbu, jako by poskytli zapálení jedinci.

Moje skupina uživatelů se tedy skládá hlavně z motivovaných a sportovně založených lidí v produktivním věku mezi 20 a 35 lety (i když jsem aplikaci několikrát nechal testovat uživateli staršími padesáti let). Někdo by mohl vytknout, že je taková skupina příliš monolitická. Já ale věřím, že právě od lidí, kteří využijí potenciál aplikace a budou mít potěšení z dosažených výsledků, získám nejkvalitnější nápady na vylepšení aplikace.

5.1.4 Nebezpečí testování aplikace na emulátoru

V prvních fázích vývoje finální verze aplikace jsem prováděl testování jen sám na emulovaném telefonu s operačním systémem Android. Po vydání aplikace jsem ale rychle zjistil, že přechody mezi stránkami a s nimi spjaté načítání dat z databáze neprobíhají na reálně vytížených procesorech tak hladce, jako na emulátoru. Hlášení o chybách od testerů mi ale pomohlo odhalit příčiny mnohem dříve, než bych to dokázal sám. Toto zjištění mě jen utvrdilo v přesvědčení, že zvolit UCD a spustit uživatelské testování co nejdříve byla správná volba.

5.1.5 Vývoj aplikace se zohledněním zpětné vazby od uživatelů

V této kapitole jsou uvedeny změny oproti prvotnímu návrhu aplikace, které vzešly z konverzací s testery v průběhu vývoje. Ne všechny jsou v době psaní této zprávy implementované, budou ale mít prioritu při další práci na aplikaci.

Testerům jsem před instalací zevrubně vysvětlil smysl aplikace, ale nijak jsem jim nepopisoval uživatelské prostředí. Snažil jsem se tím simulovat situaci, ve které si nový uživatel přečte popisek aplikace v Play Store nebo App Store a pak ji začne samostatně používat. Proto jsem jim ani nezadával žádné přesně specifikované úkoly. Jediným požadavkem na nové testery bylo zamyslet se a vytvořili alespoň jednu vlastní výzvu a přijímat pozvánky do výzev od ostatních uživatelů. Tím jsem zároveň ověřil, že typy výzev (popsané v kapitole

2.1) pokrývají dostatečnou škálu činností a situací na to, aby uživatele při tvorbě výzev nijak neomezovaly.

V jednotlivých sekcích budu popisovat hlavně změny ve funkcionalitě a rozložení prvků GUI. Grafická stránka aplikace prošla (a stále prochází) velkým množstvím malých změn, které zde nemá smysl uvádět jednotlivě. Prakticky všichni členové mojí testovací skupiny se ale shodli na tom, že prvotní návrh GUI byl příliš barevný a že kvůli tomu aplikace vypadala staromódně a kýčovitě. Ve většině sekcí uvedu dva obrázky. Nejprve vzhled a rozložení GUI z první verze aplikace a pak uživatelské rozhraní stejné části aplikace, které ale již prošlo přepracováním po zpětné vazbě od testerů.

Tutorial

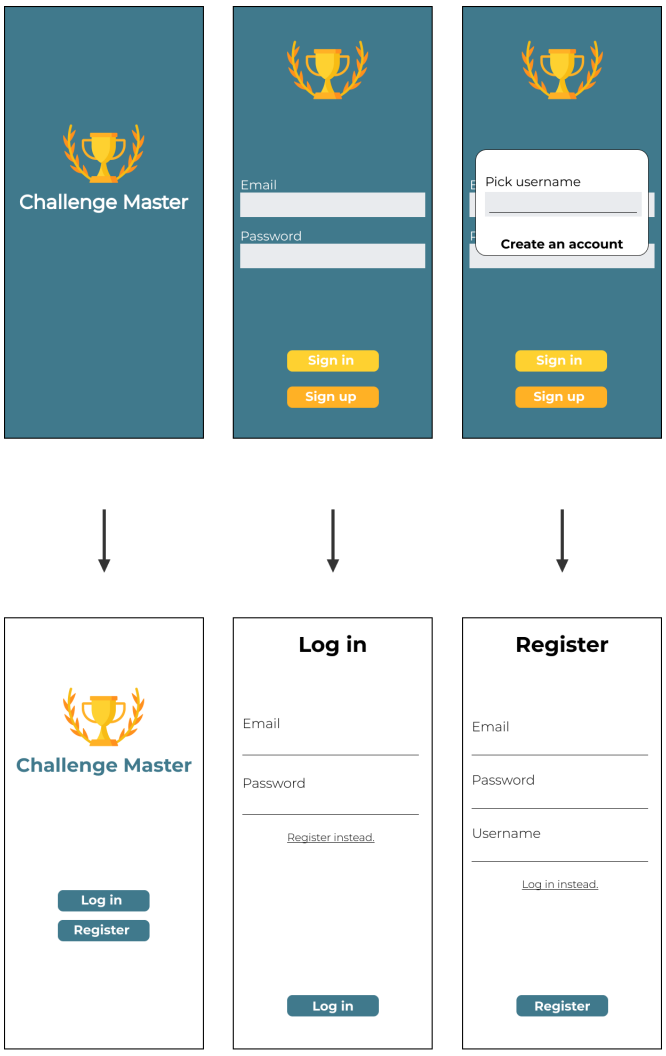
U některých testerů uběhlo několik dní mezi tím, kdy jsem jim vysvětlil princip aplikace a dobou, kdy ji začali samostatně používat. Po této promlce už si nebyli jistí hlavně rozdíly mezi jednotlivými typy výzev. Proto jsem se rozhodl přidat stránku stručně popisující vlastnosti a ovládání aplikace, která se zobrazí před prvním přihlášením. V současné podobě se jedná pouze o stránku s textem na bílém pozadí a nemá proto smysl ji zde uvádět. Pokud se ale podaří nasbírat větší počet uživatelů, bylo by na místě znázornit funkce aplikace spíše graficky a pomocí animací.

Autentizace

Jak se ukázalo, GUI pro autentizaci většinu uživatelů mátlo. Nevěděli například, jaký je rozdíl mezi „sign up“ (registrovat se) a „sign in“ (přihlásit se). Dialogové okno pro zadání uživatelského jména při registraci, které se objevilo po kliknutí na tlačítko Sign up, také nebylo nejšťastnější řešení. Část testerů se snažila pro dokončení registrace použít znovu tlačítko Sign up namísto tlačítka v dialogovém okně.

Rozhodl jsem se tedy tuto stránku nahradit třemi stránkami, z nichž by jedna sloužila pro registraci, jedna pro přihlášení a jedna jako rozcestník mezi těmito dvěma možnostmi. V novém návrhu jsem také použil jednoznačnější označení jednotlivých typů autentizace – namísto „Sign in“ a „Sign up“ jsem zvolil „Log in“ a „Register“.

K tomuto novému řešení jsem opět obdržel zpětnou vazbu. Uživatelům se zdálo zbytečné muset se nejdříve vrátit na rozcestník mezi přihlášením a registrací, když se omylem dostali na jinou stránku, než chtěli. Navrhli mi přidat odkaz na stránku pro registraci do stránky pro přihlášení a naopak. Celkový výsledek najdete na obrázku 5.2.

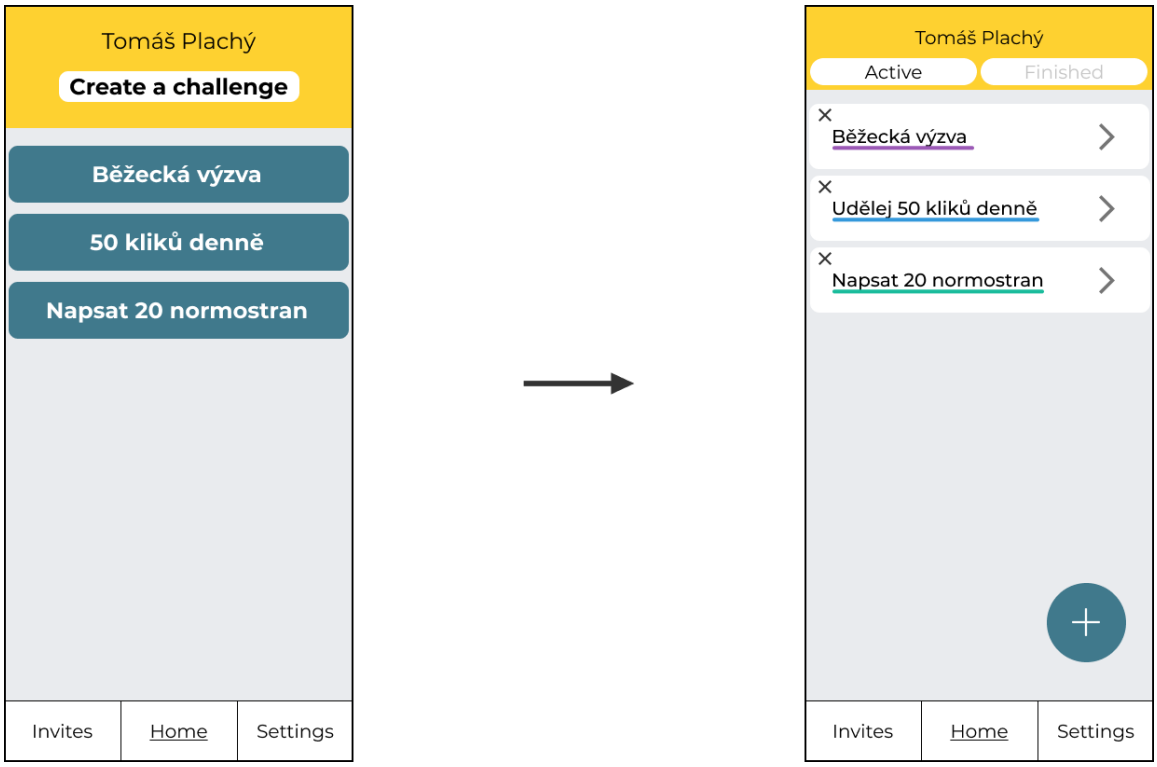


Obrázek 5.2: Obrázek znázorňuje přepracování stránek spojených s autentizací.

Seznam výzev

Se seznamem výzev byl problém hlavně v případě, kdy se uživatel účastnil většího počtu výzev zároveň a pouhé zobrazení všech najednou bylo nepřehledné. Míchaly se mezi sebou totiž výzvy tří různých typů, z nichž některé byly už uzavřené a jiné stále probíhaly. Po několika dialozích s uživateli jsem přišel s řešením umístit na hlavní stránku tři záložky a na každou umístit výzvy jednoho typu (probíhající a ukončené dohromady). Nakonec se ale ukázalo, že lepší řešení bude rozdělit do záložek zvlášť ukončené a zvlášť probíhající výzvy a u každé výzvy barevně indikovat o jaký typ se jedná. Většinu interakcí totiž uživatel provádí s probíhajícími výzvami a záložka s těmi ukončenými slouží spíše jako archív. Barvy přiřazené k typům výzev se dají navíc využít i u dalších prvků GUI, jak je zmíněno v kapitolách 5.1.5 a 5.1.5.

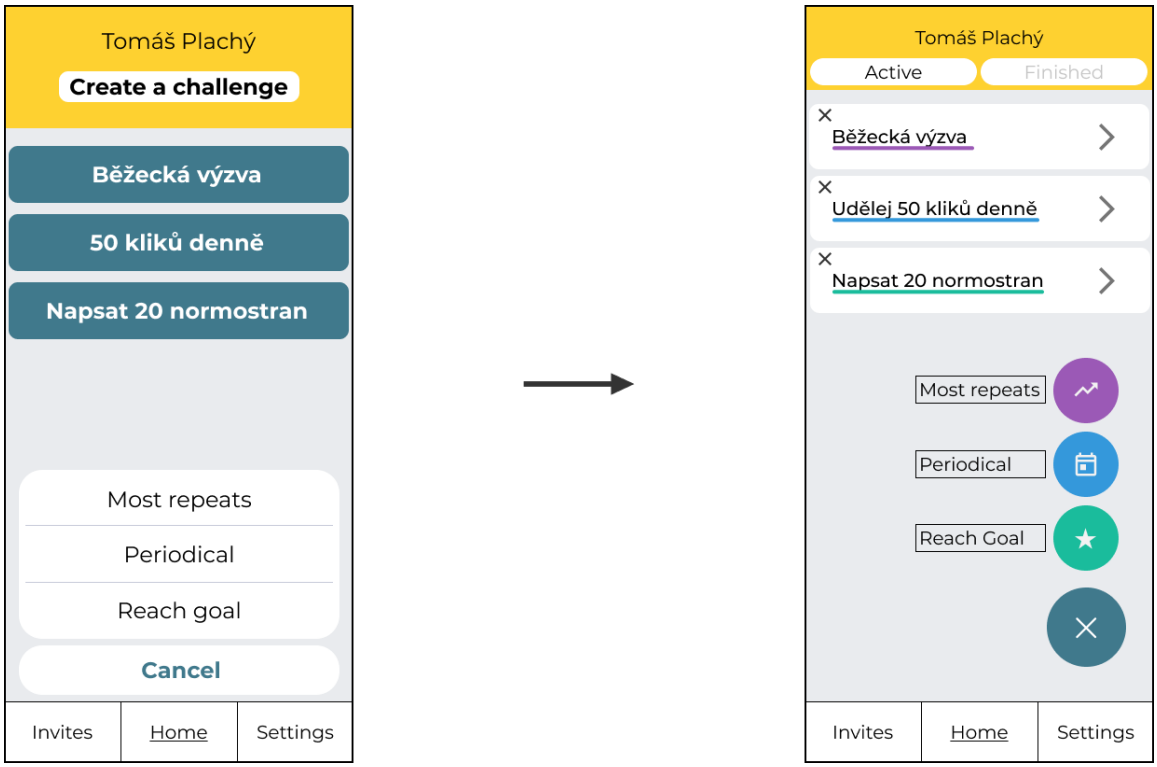
Dále jsem díky uživatelské zpětné vazbě přidal ke každé výzvě v seznamu šipku směřující doprava, protože starším uživatelům nebylo jasné, že každá položka v seznamu je zároveň tlačítko odkazující na detail výzvy. Testeři mi také doporučili přesunout tlačítko pro opuštění výzvy z jejího detailu do seznamu výzev. Detail byl už tak příliš barevný a zaplněný. A přidal jsem také text informující uživatele o tom jak založit novou výzvu, který se objeví, když bude seznam na záložce aktivních výzev prázdný (typicky po prvním přihlášení uživatele). Do budoucna je ještě možné přidat filtr pro řazení výzev podle typu nebo data ukončení. Aktuální podobu GUI najdete na obrázku 5.3.



Obrázek 5.3: Obrázek znázorňuje přepracování stránek spojených s autentizací.

Tlačítko pro tvorbu výzev

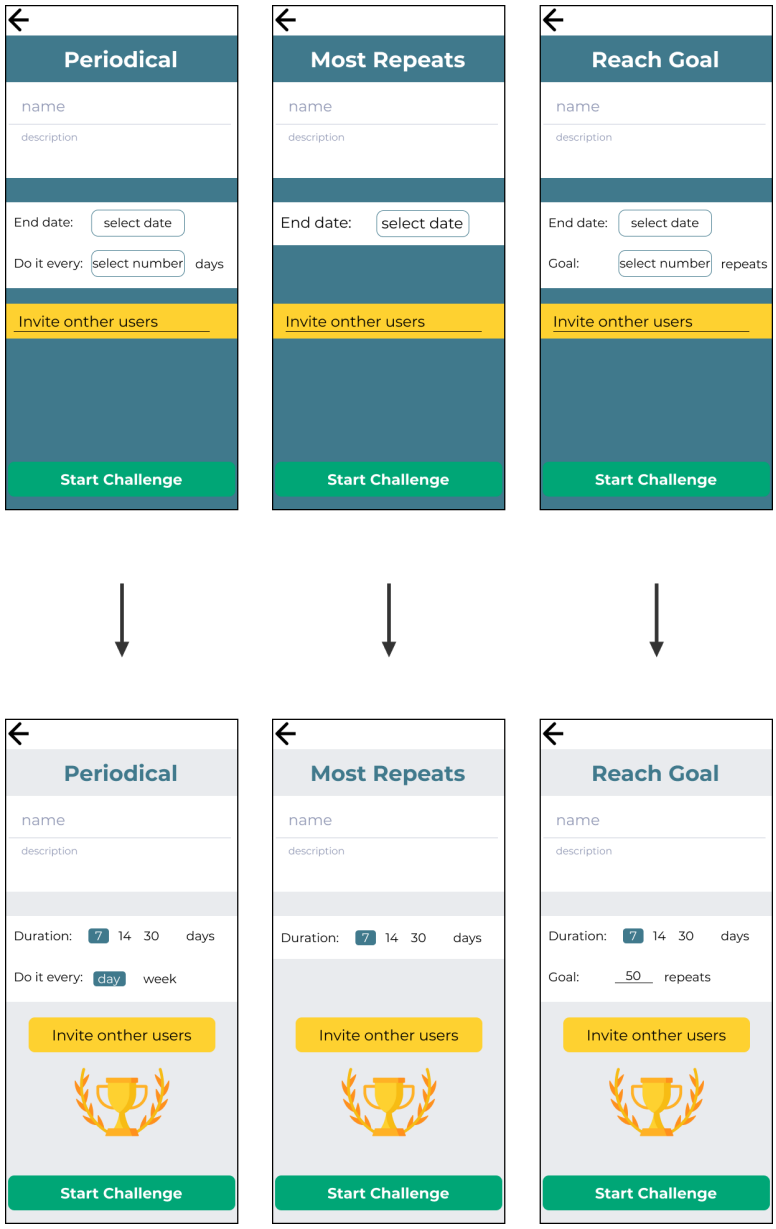
V průběhu uživatelského testování jsem zjistil, že poloha tlačítka pro tvorbu výzev v horní části obrazovky je dost neintuitivní a nepohodlná (uživatel na něj mnohdy nedosáhne palcem, když drží telefon v jedné ruce). Proto jsem ho nahradil plovoucím tlačítkem v pravé dolní části obrazovky, jak je vidět na obrázku 5.4. Po stisknutí tlačítka uživatel může vybrat, jakou výzvu chce vytvořit (jednotlivé typy jsou od sebe barevně odlišeny).



Obrázek 5.4: Obrázek znázorňuje přepracování tlačítek pro výběr typu nové výzvy.

Zakládání nové výzvy

Při přepracovávání stránek pro tvorbu výzev jsem se soustředil na to, aby granularita nastavovaných údajů nebyla příliš vysoká. Proto jsou u výběru délky výzvy nově jen tři možnosti a ne odkaz na výběr data v kalendáři. Tento přístup šetří uživateli čas a dává aplikaci možnost použít nejčastěji vybírané hodnoty jako výchozí pro příští výzvu. Při rozhovorech s testery jsem také zjistil, že většina uživatelů má zájem jen o periodické výzvy s opakováním každý den nebo každý týden. Bylo by tedy zbytečné nutit uživatele vypsat délku periody ručně. Místo toho nyní stačí vybrat z možností „každý den“ a „každý týden“, jak je vidět na obrázku 5.5.

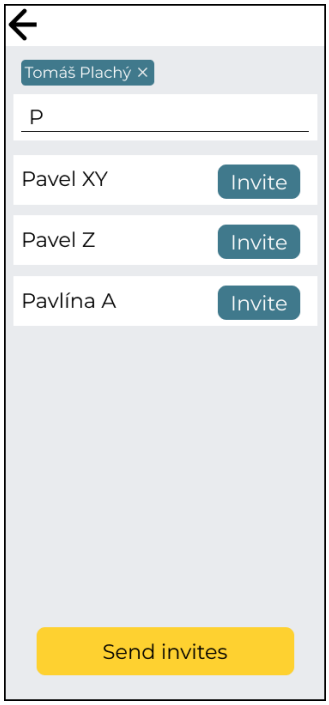


Obrázek 5.5: Obrázek znázorňuje přepracování detailů výzev.

Zvaní uživatelů do výzev

Zvaní uživatelů do výzvy bylo nejdříve možné jen při tvorbě nové výzvy. Na popud testerů jsem k tomuto účelu ale vytvořil samostatnou stránku, kterou najdete na obrázku 5.6. Tato stránka je dostupná jak z GUI pro tvorbu výzvy, tak z detailu výzvy. Uživatel začne psát jméno uživatele, kterého chce do výzvy pozvat a aplikace mu bude nabízet nejlepší shody z databáze. Takto lze přidat více uživatelů, kteří se po přidání objeví v seznamu na horním okraji obrazovky a pak odeslat všechny pozvánky najednou.

Firestore však nepodporuje fulltextové vyhledávání a kompatibilní služby třetích stran pro toto vyhledávání jsou pouze placené. Proto jsem tento problém vyřešil přidáním seznamu podřetězců uživatelského jména ke každému uživateli v databázi. Tvorba tohoto seznamu se provede na straně serveru pomocí služby Firebase Cloud Functions.



Obrázek 5.6: Obrázek znázorňuje novou obrazovku pro zvaní uživatelů do výzev.

Detail výzvy

Tato stránka nově mění barvu záhlaví podle toho, o jaký typ výzvy se jedná a dává účastníkům možnost pozvat do výzvy další uživatele (což bylo dříve možné jen v při tvorbě výzvy). Testeři také projevili zájem o to, aby se uživatelé ve výzvě řadili podle splněných opakování (od nejúspěšnějšího po nejlínějšího vyzyvatele). GUI pro každý typ výzvy naleznete na obrázku 5.7.

Původně se opakování přidávala po jednom, což se ukázalo být dosti neefektivní zvláště u výzev, které měly za cíl velký počet jednoduchých činností (například „Kdo dřív udělá 500 kliků“). Bylo by ale nepohodlné muset vždy vypsat, kolik opakování chceme do výzvy přidat. Tento problém jsem vyřešil přidáním číselného vstupu, který má implicitně nastavené číslo jedna. Uživatel se pak může rozhodnout, jestli raději několikrát stiskne tlačítko pro přidání

jednoho opakování, nebo do tohoto pole vepíše novou hodnotu a stisknutím tlačítka přidá všechna opakování zároveň.

Detail periodické výzvy prošel největšími změnami. V původním návrhu byla u každého účastníka zobrazena jen informace o tom, jestli výzvu splnil v daný den. Testery ale zajímalo mnohem víc to, jak si vedli v porovnání s ostatními v průběhu celé výzvy. Proto je nově u každého uživatele zobrazen počet uběhlých dnů výzvy v podobě obdélníků, které jsou vyplněné buď červenou (pokud uživatel výzvu v daný den nesplnil) nebo zelenou (pokud uživatel výzvu v daný den splnil) barvou. Na začátku čtrnáctidenní výzvy je tedy pod každým účastníkem jen jeden červený obdélník a na konci 14 obdélníků zbarvených podle toho, ve kterých dnech daný účastník výzvu splnil.

Obrázek 5.7: Obrázek znázorňuje přepracování detailů výzev.

5.2 Základní vlastnosti implementace

V této kapitole popíšu možnosti nabízené frameworkem React Native, které jsem ve své aplikaci využil. Při implementaci aplikace jsem se soustředil na co největší přehlednost kódu a nezávislost aplikace na dalších službách. Proto jsem se snažil vytvářet vlastní komponenty, aby dokumenty popisující jednotlivé obrazovky nedosahovaly několika set řádků kódu. Také jsem se rozhodl nevyužít knihovnu Redux, ale místo toho implementovat obsluhu globálního stavu aplikace sám.

Dále se budu věnovat vlastním řešením navigace v aplikaci a zasílání upozornění, které mají zásadní vliv na uživatelskou zkušenost s aplikací. O navigování v aplikaci by uživatel neměl vůbec přemýšlet a upozornění by měla být zasílána v takových intervalech, aby uživatele nerušila. Obě oblasti jsou tedy kritické, protože nepovedené řešení pravděpodobně povede ke smazání aplikace.

5.2.1 Stránky a komponenty

Komponenty [5] jsou v jazyce React Native základní kameny, ze kterých můžeme poskládat uživatelské prostředí aplikace. Každá stránka aplikace je tedy komponenta složená z několika dalších komponent, které mohou být již obsažené v jazyce React native, nebo vytvořené programátorem.

Můžeme vytvářet buď třídní, nebo funkční komponenty. Třídní komponenty mají vlastnosti tříd, jak je známe z objektově orientovaných jazyků. Pokud je chceme použít pro vykreslení části uživatelského rozhraní, musejí dědit od třídy „React.Component“ a metoda „render()“ musí vracet předpis pro vykreslení nějakého prvku GUI. Funkční komponenty se zase chovají jako funkce, které vracejí předpis pro GUI. Třídní komponenty umožňují využít dědičnost, ale to s sebou nese i určitou režii, která jejich vykreslování zpomaluje.

Já jsem se rozhodl používat ve své aplikaci funkční komponenty, jelikož díky nim aplikace běží rychleji a jsou v React Native komunitě mnohem oblíbenější, než ty třídní. Následující výňatek z kódu popisuje vykreslení seznamu pozvánek. Pro zpřehlednění se jedná jen o část návratové hodnoty funkční komponenty a je vynechán kód pro režii vnitřních stavů. Komponenta `<Flatlist />` načte seznam dat ze stavové proměnné `myInvites` a pro každou položku vykreslí komponentu `<InvitationToChallenge />`, kterou jsem sám vytvořil. Informace, které se mají vykreslit pro každou pozvánku a funkce, která se má zavolat po reakci na pozvánku se pošlou komponentě jako parametry.

```
<FlatList
  data={state.myInvites}
  renderItem={({ item }) => (
    <InvitationToChallenge
      challengeName={item.challengeName}
      message={item.message}
      respond={data => {
        respondToInvite(data);
        );
      }}
    />
  )}
  keyExtractor={item => item.inviteid}
/>
```

5.2.2 Stav a kontext

Stav komponenty obsahuje data, která se mohou v průběhu času měnit a každá jejich změna má nějaký vliv na chování či vzhled komponenty. Pokud použijeme stavovou proměnnou, která například určuje barvu tlačítka a nějaká situace zapříčiní změnu této proměnné, postará se React Native o to, aby se znovu načetla každá komponenta, ve které je tato stavová proměnná použita.

Stavové proměnné se dají předávat dalším komponentám, aby například změna uvnitř vnořené komponenty vedla ke změně i uvnitř mateřské komponenty. U dat, která jsou potřebná pro více komponent v různých úrovních vnoření by ale byl tento způsob předávání „shora dolů“ značně nepřehledný. React Native naštěstí nabízí řešení v podobě kontextu, díky kterému můžeme sdílet stavy globálně. Představme si určitou komponentu, která v sobě obsahuje další komponenty, jako kořen stromové struktury těchto komponent. Pokud tento kořen obalíme kontextovou komponentou, budou mít k datům v kontextu přístup i všechny ostatní komponenty v této stromové struktuře.

Ke správě globálního stavu v aplikaci by bylo možné použít například knihovnu Redux. Já jsem se ale rozhodl implementovat kontexty ručně, abych tuto problematiku lépe pochopil. Část zdrojového kódu pro generování kontextů jsem tedy přejal z tutoriálu[1].

5.2.3 Navigace v aplikaci

I jednoduchá aplikace, jako je Challenge Master, obsahuje (z pohledu uživatele) 13 různých obrazovek. Navigace mezi nimi musí být intuitivní a uživatel by se v žádném případě neměl v aplikaci ztratit. Je proto důležité mít možnost vrátit se zpátky po každé akci, která vede ke změně obrazovky. Musejí se ale ošetřit specifické případy, jako je například navigace z přihlašovací stránky dovnitř aplikace, které by měly být od zbytku aplikace oddělené. V následujících odstavcích jsem popsal možnosti, které nabízí balíček react-navigation [4]. Tyto různé navigátory do sebe lze zanořovat[2].

SwitchNavigator

Umožňuje uživateli navigaci mezi stránkami, ale v jeden moment může být načtená jen jedna stránka. To znamená, že například není možné navigovat zpět na dříve navštívenou stránku pomocí tlačítka zpět, které nabízí operační systém Android. Tento způsob navigace se využívá zejména pro přechod z autentizace do samotné aplikace, abychom zabránili vstupu bez přihlášení.

StackNavigator

při navigaci mezi stránkami uchovává pořadí navštívených stránek a umožňuje tak navigaci zpět ať už pomocí tlačítka zpět v operačním systému Android nebo lišty s tlačítkem zpět v horní části obrazovky, kterou k obrazovce automaticky připojí.

[1] https://www.udemy.com/course/the-complete-react-native-and-redux-course/learn/lecture/15707448

[2] 6. února 2020 byla vydána React Navigation 5.0, která obsahuje řadu významných změn jako je například vypuštění navigátoru SwitchNavigator. Naše aplikace zatím používá navigaci řady 4.

TabNavigator

vytvoří lištu na horním nebo spodním okraji obrazovky, pomocí které lze přecházet mezi obrazovkami. Tento způsob navigace se používá zejména pro nejdůležitější a nejnavštěvovanější stránky aplikace.

DrawerNavigator

vytvoří rozhraní pro navigaci na liště podél delší strany telefonu. Tato lišta je zcela skrytá a objeví se jen po specifické uživatelské akci (jako je například přetažení zleva doprava). Je to tedy ideální řešení pro aplikace se složitější navigací a větším počtem důležitých stránek, které by se nevešly na spodní lištu s využitím TabNavigator.

Implementace navigace v aplikaci

Já jsem zvolil StackNavigator pro přechody mezi různými typy autentizace a SwitchNavigator pro přechod z části pro autentizaci do zbytku aplikace. Uvnitř aplikace používám TabNavigator pro navigaci mezi hlavními stránkami a StackNavigator pro stránky pro tvorbu výzev, zobrazování detailu výzev a zvaní uživatelů do výzev.

Tohoto lze díky balíčku react-navigation dosáhnout velmi jednoduše. Jak je vidět v následujícím kusu kódu, tvorba navigátorů je velmi podobná zápisu objektů ve formátu JSON. Hodnotami jsou stránky v aplikaci (což jsou vlastně importované funkční komponenty) a klíče zde představují názvy stránek v navigátoru, které se budou používat pro přesměrování na dané stránky. Při vstupu do navigátoru se jako první načte odkaz, který je v objektu na prvním místě. V našem případě se tedy načte nejdříve LoadingScreen, po přesměrování na odkaz „loginFlow“ se zase načte jako první AuthCrossroadScreen a tak podobně. Jak je vidět, zanořování navigátorů je také velice intuitivní.

```
const appNavigator = createSwitchNavigator({
  Loading: LoadingScreen,
  loginFlow: createStackNavigator({
    AuthCrossroad: AuthCrossroadScreen,
    Register: RegisterScreen,
    LogIn: LogInScreen,
  }),
  mainFlow: createStackNavigator({
    tabFlow,
    NewChallenge: NewChallengeScreen,
    ChallengeDetail: ChallengeDetailScreen,
    SendInvites: SendInvitesScreen,
    SendInvitesWithoutCreatingChallenge: SendInvitesWithoutCreatingChallengeScreen,
  }),
});
```

Pro přehlednost jde v navigátorech používat i proměnné, což jsem využil pro tvorbu navigátoru se spodní lištou. V následujícím kusu kódu je vidět seznam stránek, které se zobrazí na liště, a také nastavení vzhledu a chování lišty, jako je například indikace aktivní karty změnou barvy jejího názvu.

```
const tabFlow = createBottomTabNavigator(
  {
```

```
    Invites: InvitesScreen,
    Home: HomeScreen,
    Settings: SettingsScreen,
  },
  {
    tabBarOptions: {
      activeTintColor: 'black',
      inactiveTintColor: 'gray',
      style: {
        height: 60,
        marginTop: 5,
        backgroundColor: '#fff',
      },
      labelStyle: {
        fontSize: 13,
        lineHeight: 20,
      },
    },
  }
);
```

5.2.4 Zasílání upozornění

K zasílání upozornění jsem se rozhodl využít API Expo Notifications. Tato služba je zcela zdarma a abstrahuje nutnost rozlišovat, jestli je notifikace zaslaná na operační systém Android, nebo iOS.

Pro každého uživatele je třeba vygenerovat při registraci identifikátor pro notifikace. Pokud uživatel po registraci vykoná akci, na kterou chceme upozornit ostatní uživatele, stačí potom zavolat jednoduchou funkci pro odeslání upozornění a Expo Notifications se postará, aby přišla na správné zařízení. V následujícím úryvku kódu je znázorněno odeslání notifikace o tom, že byl uživatel pozván do nové výzvy.

```
await fetch('https://exp.host/--/api/v2/push/send', {
      method: 'POST',
      headers: {
        Accept: 'application/json',
        'Accept-encoding': 'gzip, deflate',
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({
        to: 'idProNotifikacePříjemce',
        title: 'Název výzvy',
        body: 'Popis výzvy',
      }),
    });
```

Notifikace budou uživatele informovat o obdržení nové pozvánky do výzvy a o tom, že ve výzvě, které se už účastní, někdo splnil opakování. Druhá zmíněná situace by však mohla vést na velké množství upozornění, proto plánuji nastavit minimální časový interval,

který musí uběhnout mezi zasláním upozornění z jedné výzvy. Další možné řešení by bylo nastavit pro každého účastníka výzvy limit na jedno odeslání upozornění ve výzvě za den.

5.3 Publikace a propagace

Při publikaci aplikace jsem narazil na nečekané problémy s vytvořením Apple Developer účtu, který je potřeba pro vydávání aplikací na App Store a jehož vedení stojí 99 dolarů ročně. Celý proces se protáhl na několik týdnů, v průběhu kterých jsem několikrát volal s podporou pro vývojáře a musel jsem poslat společnosti Apple několik dokumentů. Problémy se ale bohužel podařilo vyřešit až těsně před datem pro odevzdání této práce, takže moje aplikace zatím zůstává pro iOS nedostupná. Zjistil jsem však, že má jejich systém problémy s vytvořením vývojářského účtu pro uživatele, který bydlí na adrese s diakritikou a že je pro platbu poplatku potřeba sehnat kreditní karta (ne debetní, jakou má většina lidí v České republice). Zaplacení poplatku za vývojářský účet pak nefungovalo z prohlížečů Safari a Firefox a proběhlo úspěšně pouze v prohlížeči Chrome. Ani tady však komplikace nekončily, jelikož mi Apple naúčtoval požadovanou částku dvakrát a musel jsem s nimi ještě vyjednat vrácení peněz. Naopak publikování na Play Store proběhlo bez problémů.

Při vydávání aplikace je třeba myslet i na to, jak na ni upozornit potenciální uživatele. Většinou se pro aplikaci vytvoří reklamní kampaň v obchodě nebo na sociálních sítích. V našem případě to však není nutné, jelikož naše aplikace má potenciál šířit se od jednoho uživatele na skupinu jeho kamarádů (které by chtěl pozvat do výzvy) a od nich potom dál stejným způsobem.

Kapitola 6

Závěr

V práci jsem se zabýval problematikou na uživatele zaměřeného návrhu a vývoje multiplatformní aplikace. V průběhu práce jsem se seznámil s různými přístupy k vývoji multiplatformních aplikací a zvolil jsem si ten, který je pro aplikaci pro tvorbu výzev a účast v nich nejvhodnější. Nejdříve jsem prozkoumal, jaké funkce by potenciální uživatelé ocenili a vytvořil prvotní návrh aplikace. Na základě tohoto návrhu jsem implementoval prototyp, který jsem použil ke sběru informací o tom, jaké akce provádějí uživatelé v aplikaci nejčastěji, které funkce jsou pro uživatele klíčové a které v prototypu chybí. Takto získaná data jsem využil při tvorbě první oficiální verze aplikace, kterou jsem pojmenoval Challenge Master. I tuto verzi jsem však nadále vylepšoval v závislosti na zpětné vazbě od uživatelů, čímž jsem splnil zadání této práce.

Jelikož je ale vývoj zaměřený na uživatele iterativní proces bez určitého konce, lze v mé práci pokračovat a aplikaci nadále vylepšovat prakticky neomezeně. Prostor pro zlepšení je zejména v režii zasílání upozornění, jejichž frekvence má velký vliv na to, jak aktivně bude uživatel aplikaci používat. Upozornění by ideálně měla obsahovat agregované informace (v případě, že by uživatel dostal větší množství upozornění v krátkém čase) a měla by sloužit i k tomu, aby se aplikace uživateli připomněla po tom, co přestane být aktivně používána. Při vylepšování GUI jsem se soustředil spíše na funkčnost než na vzhled a proto bych další práci zaměřil více na estetickou stránku aplikace a přidání animací.

Aplikaci jsem vydal na Play Store a za tu krátkou dobu, kdy je aplikace dostupná, se uskutečnilo několik desítek výzev se širokou škálou zaměření (od běhání po práci na školních projektech). Pro mě osobně je hlavním úspěchem to, že aplikaci používají nejen mí kamarádi, které jsem oslovil za účelem testování, ale i druhá generace uživatelů, ke kterým se Challenge Master dostal přirozenou cestou (nejčastěji přímým doporučením od uživatelů z první generace).

Práce na tomto projektu mi dala praktické zkušenosti ze všech etap vývoje aplikace a narazil jsem na mnohá úskalí, která skýtá vývoj za pomoci komunitou vytvářených frameworků a knihoven. Věřím tedy, že tato zpráva čtenáři nabídne jak možnost vyvarovat se stejným chybám, tak užitečné informace pro rozhodování v klíčových momentech vývoje aplikací.

Literatura

[1] Bacon, E. *Using Firebase in Expo* [online], 4. října 2018 [cit. 2020-05-12]. Dostupné z: https://blog.expo.io/using-firebase-in-expo-e13844061832.

[2] Braun, J. a community. *Getting Started With NativeScript* [online], 22. května 2019 [cit. 2020-04-27]. Dostupné z: https://github.com/NativeScript/docs/blob/master/docs/start/introduction.md.

[3] Charland, A. a Leroux, B. Mobile application development: web vs. native. *Communications of the ACM*. ACM. 2011, sv. 54, č. 5, s. 49–53. ISSN 00010782.

[4] Facebook a community, R. N. *Hello React Navigation* [online]. [cit. 2020-05-05]. Dostupné z: https://reactnavigation.org/docs/4.x/hello-react-navigation.

[5] Facebook a community, R. N. *React Fundamentals* [online]. [cit. 2020-05-10]. Dostupné z: https://reactnative.dev/docs/intro-react.

[6] Firebase. *Firebase helps mobile and web app teams succeed* [online]. [cit. 2020-04-29]. Dostupné z: https://firebase.google.com/products.

[7] Gasson, S. The reality of user-centered design. *Journal of Organizational and End User Computing (JOEUC)*. IGI Global. 1999, sv. 11, č. 4, s. 5–15.

[8] Google a community. FAQ. *Flutter* [online], 11. ledna 2020 [cit. 2020-04-14]. Dostupné z: https://flutter.dev/docs/resources/faq#what-is-flutter.

[9] Griffith, C. *Mobile app development with Ionic, revised edition : cross-platform apps with Ionic, Angular and Cordova*. First edition. Sebastopol, CA: O´Reilly, 2017. ISBN 978-1-491-99812-0.

[10] Gunnulfsen, M. K. *Scalable and Efficient Web: Thin-clients and SQL vs. Thick-clients and NoSQL*. Oslo, NOR, 2013. Master Thesis. University of Oslo Department of informatics. Dostupné z: https://www.duo.uio.no/handle/10852/37423.

[11] Hermes, D. *Xamarin mobile application development : cross-platform C# and Xamarin.forms fundamentals*. New York, NY: Apress, 2015. ISBN 978-1-4842-0215-9.

[12] Jern, M. "Thin" vs. "Fat" Visualization Clients. In: *Proceedings of the Working Conference on Advanced Visual Interfaces*. New York, NY, USA: Association for Computing Machinery, 1998, s. 270–273. AVI '98. DOI: 10.1145/948496.948535. ISBN 9781450374354.

[13] Kinsbruner, E. How progressive web applications (PWAs) are revolutionizing user experience. *InfoWorld.com*. San Mateo: Infoworld Media Group. 2018. Dostupné z: http://search.proquest.com/docview/2015175158/.

[14] Kremer, M. *Ionic vs. Everyone: Comparing Cross-Platform Frameworks* [online], 6. února 2019 [cit. 2020-04-07]. Dostupné z: https://ionicframework.com/resources/articles/ionic-vs-react-native-a-comparison-guide. Path: Resources; Articles; Ionic vs. Everyone: Comparing Cross-Platform Frameworks.

[15] Kruhlyk, Y. *Expo vs Vanilla React Native: What to Choose for Your Project* [online], 4. prosince 2018 [cit. 2020-04-14]. Dostupné z: https://apiko.com/blog/expo-vs-vanilla-react-native/.

[16] Lebensold, J. *React native cookbook : bringing the web to native platforms*. First edition. Sebastopol: O'Reilly, 2018. ISBN 978-1-491-99384-2.

[17] Miller, J. *Types of Brand Names: The Anatomy of a Name* [online], 1. srpna 2017 [cit. 2020-04-13]. Dostupné z: https://stickybranding.com/types-of-brand-names-the-anatomy-of-a-name/.