FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Redesign a reimplementace aplikace Index studenta ČVUT
Student:	Ondřej Kešner
Vedoucí:	Ing. Michal Valenta, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Softwarové inženýrství 2021
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2024/2025

Pokyny pro vypracování

Studentská unie ČVUT provozovala aplikaci Index studenta ČVUT, která poskytovala přehled o studentských akcích a umožňovala studentům registrovat svou účast, za kterou sbírali kredity. Za určitý počet kreditů získali odměnu. Aplikace musela být stažena, protože byla technologicky neudržitelná a nefungovala na nových verzích mobilních zařízení. V další diskusi se zadavatelem bylo rozhodnuto, že nová verze bude navržena jako modul mobilní aplikace Univerzita [1].

Postupujte dle níže uvedených kroků:
1. Analyzujte a popište funkcionality staré aplikace.
2. Společně s konzultantem práce a případně na základě interview s uživateli původní aplikace revidujte požadavky na novou verzi.
3. Navrhněte architekturu nového řešení včetně výběru vhodných technologií.
4. S použitím metod softwarového inženýrství navrhněte a realizujte funkční prototyp nové verze této aplikace.
5. Zhodnoťte dosažené výsledky a diskutujte další možný rozvoj.

Odkazy:
[1] Mobilní aplikace Univerzita: https://play.google.com/store/apps/details?id=cz.weissar.university

---

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

Bakalářská práce

REDESIGN A REIMPLEMENTACE APLIKACE INDEX STUDENTA ČVUT

Ondřej Kešner

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Michal Valenta, Ph.D.
14. května 2024

Obsah

Seznam obrázků

Seznam výpisů kódu

Chtěl bych poděkovat především Ing. Michalovi Valentovi, Ph.D. za velmi příjemné a lidské vedení mé bakalářské práce. Byl mi nápomocný ve všech případech, když jsem potřeboval pomoc, abych se nezdržel s řešením problémů na příliš dlouhou dobu. Dále bych chtěl poděkovat Ing. Ondřejovi Váňovi za poskytnutí kvalitních a přesných informací při vytváření samotné komponenty. V neposlední řadě bych chtěl poděkovat majiteli aplikace Univerzita App Petrovi Weissarovi za velmi vstřícný přístup hlavně z počátku, kdy jsem se postupně seznamoval s jazykem Kotlin. Při řešení vzniklých problémů se mě snažil vždy nasměrovat správným směrem, a také mi hodně pomohl udržet struktury kódů v souladu s nejlepšími praktikami.

Prohlášení

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

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

V Praze dne 14. května 2024

Abstrakt

Bakalářská práce je zaměřena na vytvoření nové verze mobilní aplikace Index Studenta ČVUT, jako komponentu do již existující aplikace Univerzita App. V rámci této bakalářské práce vznikl funkční prototyp komponenty pro operační systémy Android a IOS, taktéž se podařilo vyvinout administrativní stránku pro správce aplikace. Tyto prototypy jsou navrženy na základě informací získaných na konzultacích se zadavatelem práce, kterým je Studentská Unie ČVUT, pomocí metod softwarového inženýrství. Výsledek této práce ulehčí a zpříjemní používání nové komponenty nejenom studentům, ale i členům Studentské unie ČVUT, kteří se o tuto aplikaci budou starat, díky automatickému přidávání akcí do aplikace a používání GPS k potvrzení účasti na akci.

V práci jsou popsány veškeré použité technologie a jejich implementace do samotné komponenty. Dále je práce doplněná o návrhy na budoucí vylepšení, která by ještě více zpříjemnila správu či používání samotné komponenty.

Klíčová slova Android, Čvuťákem jsi jednou!, Index studenta ČVUT, IOS, mobilní aplikace, redesign, reimplementace, Studentská unie ČVUT

Abstract

Bachelor thesis is focused on the creation of new version of the mobile application Index Studenta ČVUT, as a component of the existing application Univerzita App. The functional prototype was created during this bachelor thesis for operation systems Android and IOS, also an administration website was created for administrators of the application. These prototypes are designed based on the informations colected on consultations with the thesis founder, which is Studentská unie ČVUT, using software engineering methods. Result of this thesis will make the component usage easier and more enjoyable not only for the students but also for the members of the Studentská unie

ČVUT, who will take care of it, thanks to automatic event addition to the application and usage of GPS for confirming the event attendance.

In this thesis are described all used technologies and their implementation to the component. Furthermore, the thesis is finished with recomendations for future improvements, that would make the management or use of the component itself even more comfortable.

Keywords Android, Čvuťákem jsi jednou!, Index studenta ČVUT, IOS, mobile application, redesign, reimplementation, Studentská unie ČVUT

Seznam zkratek

API	Application Programming Interface, přeloženo jako Aplikační rozhraní
ČJJ	Čvuťákem jsi jednou!
SU	Studentská unie ČVUT
UI	User Interface, přeloženo jako Uživatelské rozhraní
AR	Akademický rok

Úvod

Někteří studenti magisterského či doktorského studia si ještě mohou pamatovat aplikaci Index studenta ČVUT. Jedná se o aplikaci, které byla vytvořena za účelem zpestření návštěv jednotlivých akcí pořádaných Studentskou unií ČVUT. Bohužel se technologie posouvají velmi rychle kupředu a tak již bylo velmi náročné, někdy až neřešitelné, tuto aplikaci udržovat v provozu a aktualizovat ji. Také samotné zadávání akcí do aplikace bylo velmi zdlouhavé a náročné, díky složitosti distribuce QR kódů. Z těchto důvodů oslovila SU naší fakultu s nabídkou tématu bakalářské práce na vytvoření nové verze již zmiňované aplikace.

Zadané téma mě velice zaujalo, jelikož jsem předchozí verzi aplikace zaregistroval, ale bohužel jsem jí již nestihl vyzkoušet, a tak jsem se s ní chtěl blíže seznámit. Také mě zaujala možnost vytvořit novou aplikaci, kterou mohou používat téměř všichni studenti, a tímto jim zpestřit návštěvy jednotlivých akcí.

Hlavním cílem práce je vytvořit novou verzi aplikace Index studenta ČVUT, jako komponentu do již existující aplikace Univerzita App. Tato komponenta ponese název „Čvuťákem jsi jednou!“ (ČJJ) a měla by zjednodušit používání aplikace jak pro samotné studenty, tak i pro členy SU, kteří tuto aplikaci budou spravovat. Zjednodušení se bude týkat hlavně procesů zadávání akcí do aplikace a způsobu potvrzování účasti na akcích. Součástí této práce bude analýza a implementace novějších technologií, které povedou ke zjednodušení správy komponenty.

Studentská Unie ČVUT měla již před zadáním tohoto tématu rozjednanou spolupráci s vývojáři, kteří vytvořili aplikaci Univerzita App, která již funguje na některých ostatních vysokých školách a má u studentů velký úspěch. Tato aplikace sjednocuje veškeré informace o Vašem studiu na jedno místo, ať už se jedná o zobrazení rozvrhu, jídelníčku menz, přihlašování na zkoušky atd. Díky zakomponování nové verze do této aplikace jsem měl automaticky stanoveno, jaké technologie a programovací jazyk bude moje komponenta využívat, protože se musely shodovat s těmi, které používá aplikace Univerzita App.

Kapitola 1

Analýza a popis funkcionalit aplikace Index Studenta ČVUT

Veškeré informace v této kapitole jsou získané z interních materiálů Studentské unie ČVUT.

Celá myšlenka se inspiruje na technické univerzitě v Tampere ve Finsku, kde studenti během akademického roku sbírají razítka za účast na akcích. Pokud nasbírají dostatek bodů, mohou se zúčastnit závěrečné akce, která se charakterem podobá křtu, kde jsou studenti pomocí jeřábu ponořeni v koši do ledové řeky v Tampere. Tato myšlenka se zalíbila tehdejšímu studentovi Petrovi Šatrovi, který byl v Tampere na Erasmu a rozhodl se podobnou myšlenku zrealizovat i na naší univerzitě. První pokusy o realizaci se prováděly na Fakultě dopravní, na které Petr Šatra studoval. Tento projekt dostal název Drivex.

Jelikož se Drivex těšil na Fakultě dopravní úspěchu, rozšířil se na celou univerzitu ČVUT a dostal název Index studenta ČVUT. Jeho původní podoba byla papírová knížečka podobající se klasickému indexu, do kterého byly studentům zapisovány studijní výsledky. Tyto papírové indexy se studentům rozdávaly na seznamovacích akcích nebo na oficiální párty ČVUT k zahájení AR (Akademického roku). Seznam akcí na AR musel být známý již před rozdávání indexů na již zmíněných akcích, aby je bylo možné do indexu zařadit. Tyto akce se rozdělovaly do jednotlivých kategorií a podle toho jim byly přiděleny kredity. Účast na akcích se potvrzovala razítkem a podpisem pověřené osoby. U některých akcí se k přiznání kreditů musel splnit i zadaný úkol. Pokud studenti dosáhli určitého počtu bodů, mohli se zúčastnit závěrečné akce, která se konala poslední den platnosti indexu. Studenti, kteří nedokázali za účast na akcích získat dostatečný počet kreditů, mohli splněním dalších úkolů svojí kreditovou ztrátu dohnat. Úspěšní studenti na závěrečném ceremoniálu získali

nový status a čekal je zde neobvyklý zážitek, po vzoru ponoření do ledové vody v Tampere.

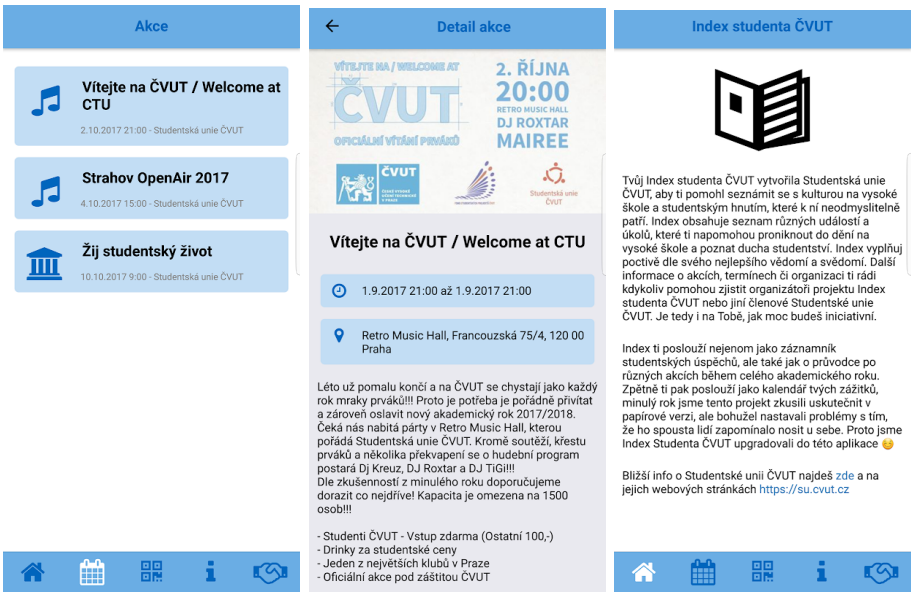
Nicméně jak se technologie dostávaly do popředí, tak se SU (Studentská Unie ČVUT) rozhodla papírovou formu indexu nahradit za mobilní aplikaci. Pro mojí bakalářskou práci je toto velmi důležité, jelikož tato mobilní aplikace je původní verze, kterou předělávám do novější a vylepšené verze a snažím se zautomatizovat její fungování. Původní verze měla přihlášení vyřešené přes Facebook. Aplikace si ho po přihlášení uložila a poté, když chtěl student přejít na jiné zařízení, pouze se přihlásil přes Facebook a pokračoval ve sbírání kreditů. Zadávání akcí již bylo flexibilnější, jelikož se mohly jednotlivé akce přidávat i během AR. Stačilo, když pořadatelé akce, odeslali SU email s podrobnostmi o jejím konání. Příslušný projektový manažér z SU, na základě těchto informací vyplnil formulář, který přidal akci do aplikace a vygeneroval QR kód. Tento kód byl poté odeslán pořadatelům, kteří již měli za úkol ho pouze viditelně vyvěsit na dané akci, aby k němu měli přístup účastníci. Ti museli tento kód přes aplikaci naskenovat a toto sloužilo k potvrzení účasti na akci. Projektový manažér musel akci přidělit i určitý počet kreditů, tento počet nebyl ničím omezen a záležel čistě jen na úsudku daného manažéra. Díky fungování této aplikace, bylo možné nabídnout studentům i dílčí odměny za nasbírané kredity. K tomu sloužil speciální email SU, kam mohli studenti poslat požadavek na získání ceny. Email musel obsahovat jejich jméno na Facebook a aktuální počet kreditů. Tento požadavek byl poté zpracován členem SU, který zkontroloval, že student má nárok na získání dané odměny. Po této kontrole se člen SU domluvil se studentem na způsobu předání odměny. Seznam odměn i s jejich kreditovou hodnotou byl k dispozici na stránkách SU.

Tato aplikace fungovala dobře, i přes její administrativní náročnost. Bohužel SU neměla k aplikaci zdrojové kódy, tudíž její další rozvoj nebo aktualizace byly nemožné. Toto bylo jedním z důvodů, proč se projektový manažér SU Ondřej Váňa rozhodl fungování aplikace přerušit a vydat se novým směrem, kterým se zabývá moje bakalářská práce. Dalšími problémy byly např. náročnost administrace aplikace, distribuce QR kódu atd.

U QR kódu bych se na chvíli zastavil, jelikož se s nimi pojí ještě jedna špatná vlastnost, kterou je falšování. Díky tomu, že pořadatelé tyto QR kódy vystavili někde na svojí akci a byly veřejně dostupné, tak bylo jednoduché je vyfotit a poslat někomu, kdo se akce vůbec neúčastnil a díky tomu mohl získat body. Což je v rozporu s myšlenkou celé aplikace.

Na obrázcích 1.1 můžete vidět obrazovky z původní aplikace. Na obrázku 1.1a můžete vidět seznam akcí, které jsou naplánované. Jednotlivé akce se dají rozklinout a studenti se mohou podívat na jejich detail, který můžete vidět na obrázku 1.1b. V tomto detailu je vidět název a datum konání akce, poté odkaz na Facebook stránky a bližší informace ke konání, které jsou zkopírované buď z Facebook nebo z webové stránky. V záhlaví se ještě nachází úvodní obrázek z Facebook stránky. Pomocí menu ve spodní části aplikace je možné si zobrazit domovskou stránku viz. 1.1c a informační stránku. Dále se zde uživatel může

překliknout na svůj profil, kde uvidí počet kreditů nebo seznam všech svých zúčastněných akcí. Následně je zde prostřední ikona z dolního kontextového menu, která slouží k otevření fotoaparátu pro skenování jednotlivých QR kódů.



(a) Seznam akcí **(b)** Detail akce **(c)** Domovská obrazovka

Obrázek 1.1 Fotografie předchozí verze mobilní aplikace. Získano z webové stránky zálohující Android aplikace.[1]

Kapitola 2

Vybrané technologie pro novou komponentu ČJJ

Pro vytvoření nové komponenty ČJJ jsem využil nejnovější technologie, které pomůžou zautomatizovat fungování aplikace. Jelikož je tato nová komponenta zapojená do již existující aplikace, musel jsem dodržet strukturu a verze technologií, které jsou v ní již použité. Jelikož aplikace Univerzita App[1] má jak Android verzi, tak i IOS verzi, musel jsem naší komponentu vytvořit pro oba operační systémy. Nejdůležitější pro mě byla Android verze, se kterou jsem začínal a vydal jsem ji jako první. Funkční prototyp pro IOS jsem vytvořil v rámci semestrální práce v předmětu BI-IOS. Jeho vytvoření bylo jednoduché, jelikož jazyky Kotlin (pro Android verzi) a Swift (pro IOS verzi) jsou si dost podobné, a proto jsem kód mohl pouze upravovat do odpovídající syntaxe. Finální verzi jsem poté dodělával již mimo předmět BI-IOS. Jelikož jedním z požadavků na naší komponentu byla hlavně co největší automatizace procesů, rozhodl jsem se pro vytvoření backend server, který se stará o všechny administrativní úkony, jakými jsou správa uživatelů a cen, aktualizování informací o akcích atd.

2.1 Android verze

Jak jsem zmiňoval výše, tak Android verze byla pro mě nejdůležitější, jelikož jsem s ní začínal a zkoušel jsme si na ní technologie, které mohu využít. Před začátkem práce na této komponentě jsem neměl s jazykem Kotlin žádné zkušenosti. Kvůli tomu mi vývojáři Univerzita App poskytli prázdný projekt se všemi dostupnými technologiemi, které používají. Zde jsem si mohl vyzkoušet jak technologie fungují a jak s nimi pracovat. Celá Android verze je psaná v jazyce Kotlin s využitím balíčku Jetpack Compose. Pro ukládání většího množství komplexních dat se využívá knihovna Room Database.

[1] Odkaz na stránky aplikace: `https://univerzita.app/`

2.1.1 Kotlin

Kotlin je statický typovaný programovací jazyk, který se zaměřuje na platformu Java. Jeho hlavním cílem je poskytnout alternativu k jazyku Java. Tento cíl splňuje díky jeho velkým výhodám, jakými jsou stručnost, bezpečnost a kompatibilita s jazykem Java. V jazyce Kotlin je možné využívat veškeré knihovny a frameworky z jazyka Java. Díky tomu můžeme využívat Kotlin na stejných místech jako Javu např. vývoj backend serverů nebo mobilních aplikací pro platformu Android atd. Kotlin sází na všestrannost, a proto se nezaměřuje pouze na určitou problematiku, ale snaží se programátorům poskytnout výkonnější řešení pro veškeré problémy, které se během vývoje mohou objevit.[2]

2.1.2 Jetpack Compose

Jetpack Compose je moderní balíček nástrojů pro vytvoření UI, který pracuje pouze se třídami, které jsou napsány v jazyce Kotlin. Pomáhá zrychlovat a zjednodušovat vývoj UI, díky výhodám jakými jsou stručnost, před vytvořené nástroje a využívání intuitivních API, které jsou napsány v jazyce Kotlin. Tyto výhody potom vedou k rychlejšímu a jednoduššímu vývoji UI.[3]

2.1.3 Room Database

Součástí balíčku Android Jetpack je také knihovna Room database. Jedná se o nadstavbu nad SQLite, která poskytuje plynulý přístup do databáze při zachování výkonu SQLite.[4] Tato knihovna nám dává možnost jak ukládat velké množství komplexních dat umístěných lokálně na mobilním zařízení.

2.2 Backend server

V našem případě je server nejdůležitější část celé komponenty, jelikož je to hlavní středobod komunikace pro všechny zbylé části. Všechny části komunikují se serverem pomocí API (bližší popis najdete níže). Server využívá databázi pro uchovávání dat. Hlavními důvody existence serveru jsou hladký přechod uživatele mezi zařízeními, uchovávání informací o uživatelích, které jsou důležité pro kontrolu zisku odměny, a sjednocení všech procesů na jedno místo. V našem případě jsem se rozhodl napsat server v jazyce Java s využitím framework Spring a připojení k PostgreSQL databázi. Pro tento postup jsem se rozhodl z důvodu toho, že jsme si ho ukazovali během studia v předmětu BI-TJV.

2.2.1 Databáze PostgreSQL

PostgreSQL je objektově relační databázový systém. PostgreSQL je celosvětově považován za nejpokročilejší open-source [2] databázový systém, který poskytuje funkcionality, které jsou většinou vidět pouze v komerčních produktech. V klasickém slova smyslu open-source znamená, že se můžete podílet na vývoji a používat software zadarmo. V tom databázovém světě má toto slovo ještě další význam a to ten, že dostanete přístup ke statistikám o výkonu, které má většina společností jako např. Oracle zakázané.[5]

2.2.2 API

API neboli Application Programming Interface je technika v programování, která slouží pro komunikaci mezi jednotlivými prvky aplikace po síti. Tato komunikace probíhá pomocí endpoint[3]. Tyto endpoint mají sloužit ke spuštění nějaké funkce/procedury, která vykoná předem definovanou úlohu a po jejím skončení může odeslat nějaká data zpět nebo nic nevrátit. Tato technika se využívá hlavně z důvodu rozdělení výkonu mimo koncové zařízení. V praxi to poté vypadá tak, že se složité výpočty nebo business logika provádí na serveru, který má implementované API, a koncová aplikace, ať už je to Android, IOS aplikace nebo jenom webové UI, už jenom zobrazuje komponenty nebo informace v závislosti na odpovědích z volání endpoint na API.

2.2.3 Java

Jedná se o objektově orientovaný programovací jazyk. Hlavním cílem designerů bylo vyhnout se složitým funkcionalitám, na které trpí jiné objektově orientované jazyky. Samotný název Java může pod sebou skrývat mnoho věcí. Jednou z nich je samotný Java programovací jazyk, což je syntaxe jazyka a soubory, ve kterých jsou programy uchovávány. Poté máme Java VM (virtual machine[4]), což by se dalo přirovnat k interpretru samotného jazyka Java. Tento virtuální stroj se stará o spouštění takzvaného byte kódu. Jedná se o přeložený zdrojový kód, který se dá spustit na Java VM. Díky tomu, že se byte kód spouští na vlastní VM, má Java obrovskou výhodu a tou je přenositelnost samotného programu, protože program spustíte na veškerých zařízeních, které mají nainstalovanou právě Java VM. Nemusíte se tudíž bát, že by se program na některém počítači mohl chovat odlišně. Dále nedílnou součástí je také platforma Java, což je sada před připravených tříd, které můžeme ve svých programech využívat. Tato sada obsahuje třídy pro vytvoření GUI (Graphical User Interface[5]), zabezpečení, komunikaci přes síť nebo práci se soubory. Tudíž všechny tyto tři části dohromady tvoří programovací jazyk Java.[6]

[2] Přeloženo do češtiny jako otevřený zdrojový kód
[3] přeloženo do češtiny jako cílový bod
[4] Přeloženo do češtiny jako virtuální stroj
[5] Přeloženo do češtiny jako grafické uživatelské rozhraní

2.2.4 Spring Framework

Jedná se o framework pro usnadnění vytvoření aplikací. Toto usnadnění spočívá ve správě jednotlivých komponent. Abych to upřesnil, každá aplikace se skládá z více komponent, které spolupracují, aby dosáhly kýženého výsledku. Bohužel vytvoření jednotlivých instancí komponent a jejich distribuce je pro programátora velmi náročná. Z tohoto důvodu používáme Spring, který nám pomáhá s vytvořením jednotlivých instancí. Ty se následně sdružují do aplikačního kontejneru, který se stará o jejich administraci. Samotný kontejner je poté schopen instance distribuovat do komponent, které je potřebují. Této distribuci se také říká Dependency injection[6].[7] Samotný framework také obsahuje další balíčky, které je možné do aplikace nahrát, a anotace, které umožňují dát příslušné funkci nebo třídě speciální funkcionalitu. Ve své práci jsem využíval hlavně anotace pro mapování endpoint nebo konfiguraci zabezpečení samotné aplikace.

2.3 IOS verze

Jak jsem zmiňoval výše, na začátku pro mě bylo důležité, abych vytvořil verzi pro Android, kvůli brzkému spuštění funkčního prototypu. Po jeho vytvoření jsem se v rámci předmětu BI- -IOS věnoval jeho předělání pro zařízení s operačním systémem IOS. V rámci tohoto předmětu se mi povedlo vytvořit funkční prototyp, který jsem později ještě dodělával do plně funkční komponenty, která se mohla vložit do aplikace Univerzita App pro IOS. Tato verze je napsaná v jazyce Swift s použitím framework SwiftUI pro vytváření UI, který se nativně používá pro vytváření aplikací na IOS.

2.3.1 Swift

Jedná se o programovací jazyk představen společností Apple v roce 2014, který umožňuje vytvářet aplikace pro operační systémy od stejnojmenné společnosti. Hlavním záměrem bylo vytvořit programovací jazyk, který nahradí jazyk Objective-C, který se do té doby používal pro vývoj aplikací pro operační systém IOS. Na rozdíl od jiných programovacích jazyků, které jsou postaveny na starších procedurálních jazycích, byl Swift od začátku vytvářen jako moderní objektově orientovaný programovací jazyk, který měl vývojářům ulehčit a zrychlit psaní kódu a poskytnout menší náchylnost k vytvoření chyb. I přes otevřenost zakladatelů jazyka k používaní Switf na jiných platformách se primárně používá jen pro vývoj aplikací na platformy IOS, macOS a dalších operačních systémů od společnosti Apple.[8]

[6] Přeloženo do češtiny jako vložení závislostí

2.3.2 SwiftUI

Jedná se o deklarativní programovací framework na vytváření UI pro všechny platformy na zařízeních od společnost Apple, který je vytvořený pro Swift. Tento framework přinesl do vytváření UI velké zjednodušení oproti používání framework UIKit. Jedním z nich je zjednodušení syntaxe a takzvané stavy, které se starají o aktualizaci UI při změně prvků, které jsou označeny pomocí anotace `@State`, což velmi ulehčilo práci s UI. Tento framework také přináší další důležité vylepšení, kterým je živý náhled. Toto je integrováno do XCode (programovací prostředí pro práci se Swift), kde si pouze v souboru, kde UI vytváříte, nadefinujete, že chcete vytvořit náhled. Pokud jsou potřeba nějaká data, vytvoříte si tzv. mock data (jedná se o před definovaná data, která se pošlou do vytvořených prvků).[9]

2.4 Administrátorský web

Hlavní myšlenkou celé aplikace je sbírání bodů za účast na jednotlivých akcích, které potom mohou uživatelé využít k získání odměn. Aby však bylo možné ověřit, že daný uživatel má dostatečný počet bodů, je potřeba se podívat na server, kde jsou tyto informace uloženy. Kvůli bezpečnosti jsem musel všechny endpoint zabezpečit. Toto zabezpečení však zkomplikovalo přístup pro členy SU, kteří by vydávání cen měly na starosti. To mě přivedlo na myšlenku vytvořit jednoduchý web, který by sloužil určitým členům SU, kde by měly veškeré potřebné informace pohromadě, ať už by to byl seznam registrovaných studentů a jejich bodový zisk nebo informace o chybách v zadaných akcích nebo administrace aktivních cen. Tuto myšlenku jsem realizoval jako samostatnou webovou stránku, která bude komunikovat se serverem pomocí API. K tomuto vytvoření jsem si vybral framework Vue.

2.4.1 Vue

Jedná se o Javascript framework pro tvoření UI, který zahrnuje téměř všechny potřebné nástroje pro vývoj UI. Je postaven nad klasickým HTML, CSS a Javascript. Programátorům poskytuje programovací model, který je postaven na komponentách a deklarativnosti. Což vývojářům ulehčuje vytváření jednoduchých, ale i složitých UI. Dvě základní myšlenky Vue jsou deklarativní rendering, který nám pomáhá zobrazovat HTML prvky v závislosti na Javascript stavu, a reaktivnost, která se stará o kontrolování, zda se Javascript stav nezměnil. Pokud se tento stav změnil, zajistí správné překreslení daných HTML prvků. Vue má také tu výhodu, že je možné UI vytvářet v mnoha stylech, ať už mluvíme o statickém HTML, SPA (Single-Page Application, která funguje na principu zobrazení pouze jedné obrazovky, která se postupně překresluje) atd.

Kapitola 3

Návrh nové komponenty ČJJ

V této kapitole se budeme zabývat návrhem nové verze aplikace Index Studenta ČVUT. Jak jsem již zmiňoval výše, chci usnadnit a zautomatizovat používání této aplikace. Proto jsme se společně s SU a vývojáři aplikace Univerzita App rozhodli, že novou verzi zakomponujeme do aplikace Univerzita App tak, aby studenti měli veškeré důležité informace jak o studiu, tak i o konaných akcích na jednom místě. Tudíž už se nebude jednat o samostatnou aplikaci, ale pouze o komponentu. Tato komponenta ponese nový název ČJJ (Čvuťákem jsi jednou!), bude mít nový vzhled a získané body se budou jmenovat Áčka.

3.1 Požadavky na novou komponentu ČJJ

Veškeré požadavky jsem získával od Ing. Ondřeje Váňi, který je projektovým manažerem SU, a byly směřovány na zjednodušení obsluhy aplikace a částečné zautomatizování jejího provozu. Jelikož se celá aplikace stala komponentou aplikace Univerzita App, tak jsem již měl předem určené technologie a jazyk, ve kterých budu komponentu vyvíjet, aby zakomponování do již existující aplikace bylo co nejjednodušší.

3.1.1 Funkční požadavky

Funkční požadavky specifikují chování a požadované funkcionality, které může uživatel přímo ovládat a používat.

F1 Poskytování aktuálního seznamu pořádaných akcích

Na úvodní stránce aplikace je zobrazen seznam všech aktuálně naplánovaných akcí, který je seřazen podle data konání akce. Uživatel může každou akci rozkliknout, aby se dokázal podívat na detail akce, a následně má také možnost se prokliknout na oficiální stránky akce.

F2 Možnost získání bodů za účast na akci

Každá akce má přiřazenou určitou kategorii, kterou vybírá samotný člen SU a je zadána k akci do kalendáře. Tato kategorie má poté přidělené určité množství Áček, které můžou studenti získat, pokud se v čase konání pohybují s mobilním zařízením a staženou aplikací v dostatečné blízkosti akce. Poloha akce je braná z adresy konání, která je uvedená v kalendáři, a aktuální poloha mobilu je samozřejmě braná přímo z mobilního zařízení. Kvůli tomuto je potřeba, aby aplikace měla povoleno použít polohovací služby telefonu a byla zapnutá poloha. Pokud nějaký z předchozích požadavků není splněn, je na to uživatel upozorněn.

F3 Podat žádost o získání odměny

Jak jsme se dozvěděli v předchozím požadavku, tak uživatel získává Áčka, které poté může vyměnit za určité odměny. Veškeré odměny a jejich hodnota v Áčkách jsou definovány jak na webu SU, tak i přímo v aplikaci. Aby fungování této aplikace bylo co nejjednodušší, tak je zde uvedeno tlačítko, které odkáže uživatele na formulář, který slouží jako žádost o získání ceny. Zaslané formuláře zpracovávají členové SU, kteří mají k dispozici přístup na API ke zjištění aktuálního bodového zisku uživatel podle jeho uživatelského jména.

F4 Poskytnutí podrobných informací o akci

Jak jsem již zmínil, uživatel má možnost se podívat na detail akce kliknutím na políčko v seznamu. Tento detail obsahuje podrobné informace jako je popis, který z kalendáře dostaneme ve formátu HTML, a aplikace ho zobrazí ve stejném formátu jako anotovaný řetězec. Dále se zde nachází přesný čas, datum začátku i konce akce a samotné souřadnice, u kterých musí být uživatel při zadávání účasti dostatečně blízko. Je zde také klasická písemná adresa konání akce a dvě tlačítka jedno slouží k odkazování na oficiální stránku akce a druhé na přidání účasti, které se zobrazuje pouze pokud akce právě probíhá.

F5 Zobrazení oficiální stránky akce

Zobrazení oficiální stránky akce je důležité hlavně pro organizátory akce, jelikož je to pro ně jediný komunikační kanál s lidmi, kteří mají o akci zájem. Proto je důležité, aby se lidé na tyto stránky mohli jednoduše dostat. K tomu jim slouží tlačítko v detailu aplikace, které je přesměruje z aplikace přímo na url adresu, která je zadaná u akce v kalendáři.

F6 Možnost filtrování podle typu akce

Jelikož každá akce má přidělenou kategorii, tak je možné podle těchto kategorií filtrovat. Toto filtrování je umístěno přímo nad seznamem všech akcí. Z tohoto

důvodu je důležité, aby v kalendáři byly správně přiřazené kategorie, o které se starají projektový manžéři SU.

F7 Administrace cen

Aplikace má svůj vlastní backend server, na kterém je také uložen seznam odměn. Toto slouží k administraci zobrazení oznámení o tom, že uživatel dosáhl potřebného počtu Áček pro nárok na novou odměnu. Správce může nové odměny přidávat, upravovat nebo mazat.

F8 Upozornění uživatele na novou cenu

Mobilní aplikace by měla uživatele upozornit, pokud má nárok na získání nové hodnotnější ceny. Toto upozornění by se uživateli mělo zobrazit po přičtení Áček z aktuálně navštívené akce.

F9 Zadávání nových akcí

Možnost přidávat nové akce mají pouze určení členové SU, kteří mají přístup k samotnému kalendáři. Při zadávání akcí je důležité zachovat požadovanou strukturu, aby aplikace informace zobrazovala správně. Každý člen, který má k tomuto právo, dostane od SU dokument, kde jsou uvedené veškeré bližší informace a stručný návod.

F10 Spravování žádostí o odměnu

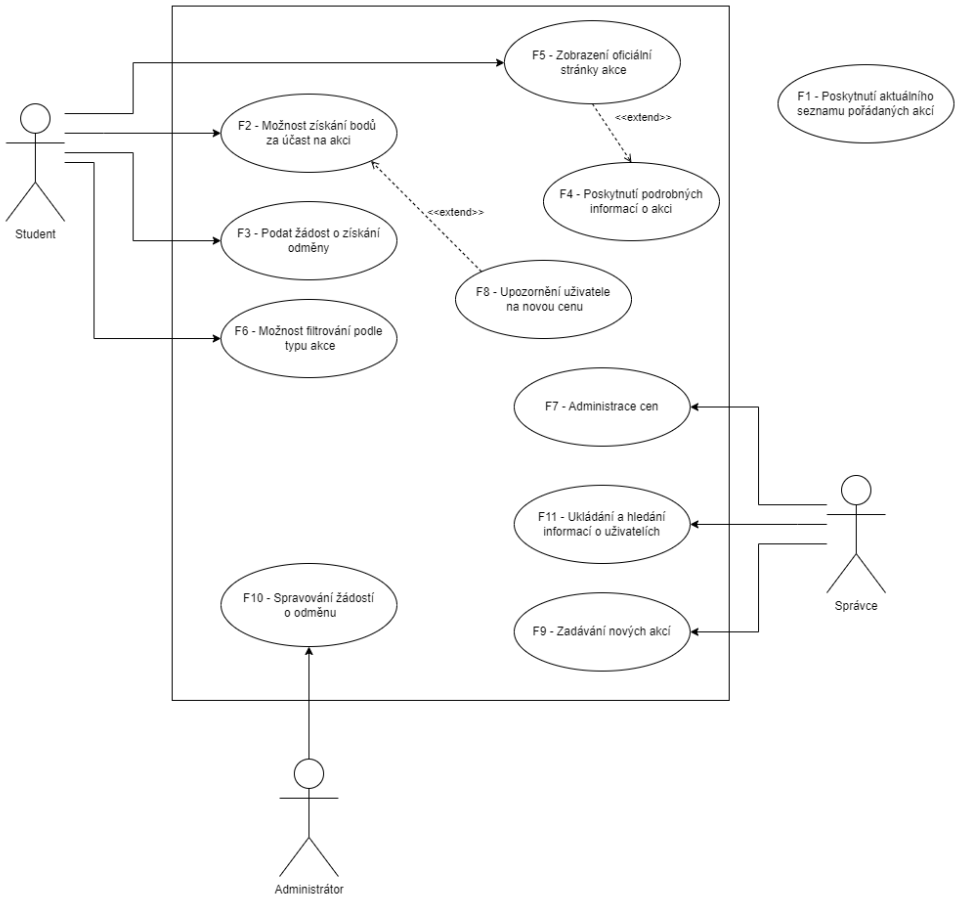
Seznam žádostí je uložený u formuláře. Správce, který poté žádosti bude zpracovávat má možnost si jednotlivé uživatele zobrazit, aby zkontroloval, zda mají požadovaný počet Áček na získání odměny. Tyto informace o uživatelích jsou uložené na API, kde se každému uživateli vytvoří profil poté, co se poprvé přihlásí do aplikace. Jednoznačnost jednotlivých uživatelů je garantovaná, pomocí unikátních uživatelských jmen ČVUT, ke kterým máme přístup díky integraci přihlášení do aplikace pomocí SSO.

F11 Ukládání a hledání informací o uživatelích

Pro účely snadného kontrolování počtu Áček je důležité, aby bylo možné se podívat na detail každého uživatele. Tento detail by se měl vyhledat na základě uživatelského jména. Tento požadavek souvisí i s jednodušším přechodem uživatelů mezi různými zařízeními.

Obrázek 3.1 ukazuje diagram případů užití všech funkčních požadavků. V diagramu máme 3 aktory, studenta jakožto uživatele aplikace, správce který má na starosti správný chod celé aplikace a administrátora který zpracovává všechny požadavky na odměny. Jak si můžete všimnout, tak jsem umístil jeden

funkční požadavek, přesněji F1, mimo diagram. Tento požadavek je nejdůležitější a závisí na něm celý chod aplikace. Dle mého názoru nemá přesně určeného uživatele, který ho vykonává.



Obrázek 3.1 Diagram případů užití

3.1.2 Nefunkční požadavky

Nefunkční požadavky specifikují vlastnosti a omezující podmínky daného systému nebo aplikace.

N1 Verze Kotlinu

Jelikož se vytváří komponenta do již existující aplikace je potřeba, aby se verze Kotlinu shodovala s tou, která je použita v aplikaci Univerzita App.

N2 Verze Swiftu

Jelikož se vytváří komponenta do již existující aplikace je potřeba, aby se verze Swiftu shodovala s tou, která je použita v aplikaci Univerzita App.

N3 Získávání informací o akcích

Aplikace by měla veškeré informace získávat automaticky přímo z Google kalendáře SU, kam se všechny akce zapisují.

N4 Přihlášení pomocí univerzitního účtu

Přihlášení do aplikace by mělo být pro uživatele co nejjednodušší, takže je vhodné použít školní účet. Toto také omezí případný přístup do komponenty pro studenty z jiných vysokých škol.

N5 Možnost snadného přechodu mezi zařízeními

Může se stát, že uživatel vymění své zařízení, případně se bude chtít přihlásit z jiného telefonu. Tato možnost by zde měla být a je důležité, aby se uživateli správně zobrazily body a zúčastněné akce po přechodu mezi zařízeními.

N6 Ověření přidání účasti na akci

Celkový požadavek na zautomatizování fungování aplikace se týká i zjednodušení administrace před konáním akce. Kvůli tomu je potřeba použít novějších technologií k ověření, že je uživatel opravdu přítomen na akci.

N7 Vícejazyčná aplikace

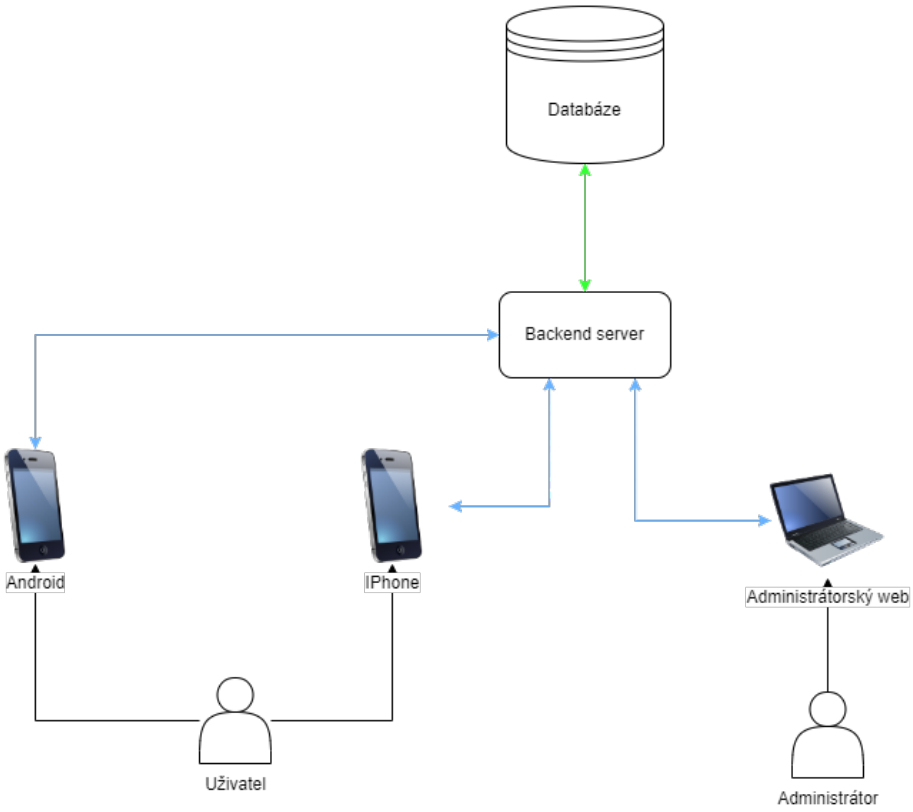
Jelikož na naší škole studují i studenti z jiných zemí než jenom z České republiky je potřeba, aby aplikace byla připravena i na tento problém. Z tohoto důvodu musí nová verze podporovat češtinu i angličtinu, aby se otevřela možnost použití aplikace i pro zahraniční studenty.

3.2 Architektura ČJJ

V této kapitole se podíváme na návrh architektury komponenty ČJJ. Jak spolu jednotlivé platformy komunikují nebo jak probíhají jednotlivé procesy, o které se naše komponenta stará. Tyto diagramy jsou důležité, abychom udrželi konzistentní chování na všech platformách. Veškeré diagramy jsou kresleny v UML notaci.

Na digramu 3.2 můžete vidět, návrh komunikace mezi jednotlivými platformami. Nejdůležitější platforma pro nás bude backend server, který se bude starat o správu všech dat a jejich distribuci všem platformám. Tudíž server

komunikuje se všemi platformami. Dokumentaci k veřejnému rozhraní (API) serveru najdete v příloze ve formátu PDF, které je vytvořeno z JSON, který reprezentuje Swagger dokumentaci. Jako další platformu máme databázi, která se stará o uchovávání všech našich dat, ať už to jsou jednotlivé akce nebo počet Áček u jednotlivých uživatelů atd. Poté jsou na diagramu koncové platformy, kterými jsou mobilní telefony s operačními systémy Android/IOS a webová stránka pro administraci. Tyto platformy jsou pro nás nejdůležitější, jelikož zobrazují veškerá data jednotlivým uživatelům, která se získávají ze serveru, a samotní uživatelé s nimi interagují.

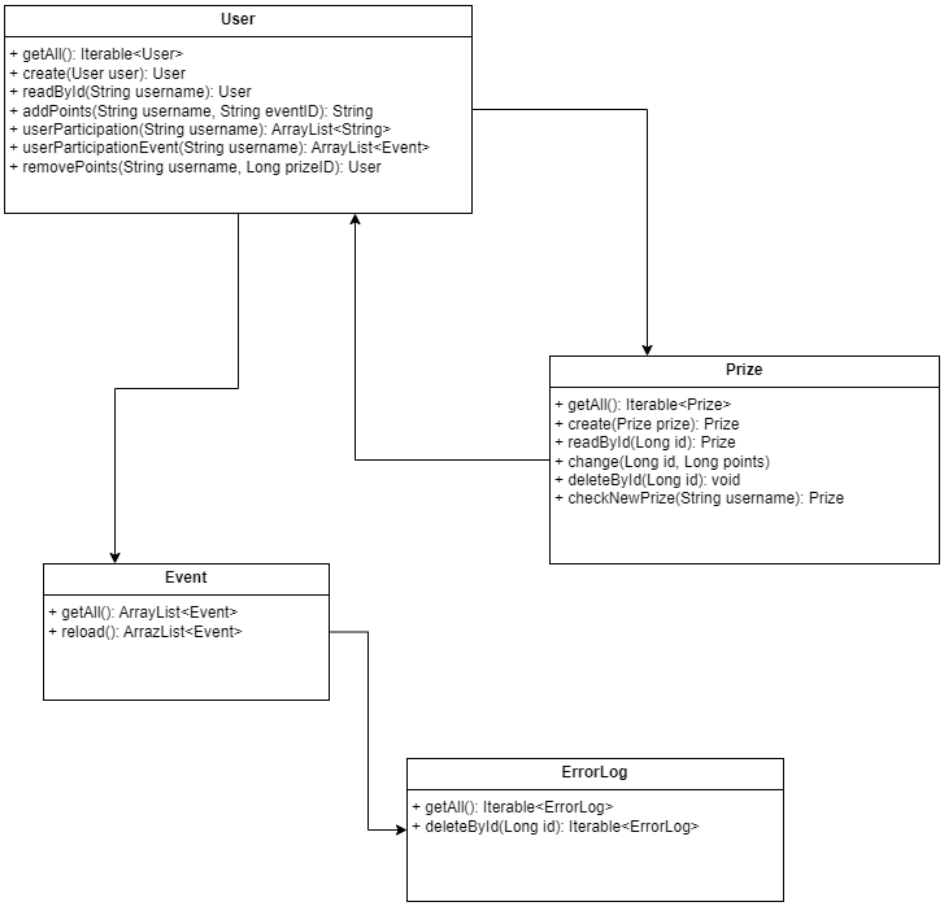


Obrázek 3.2 Struktura komunikace mezi jednotlivými platformami

Jak můžete vidět na diagramu 3.2 nejdůležitější část celé aplikace je server, který zprostředkovává veškeré akce nad daty uloženými v databázi. Z tohoto důvodu jsem vytvořil následující diagram 3.3, který ukazuje komunikaci mezi jednotlivými entitami na serveru. Samozřejmě každá třída na serveru má vytvořený svůj vlastní repozitář, který se stará o komunikaci s databází, service, který implementuje business logiku, a kontrolér, který zpracovává jednotlivé požadavky a posílá je dále do příslušných service. Pro zjednodušení diagramu

jsem se rozhodl všechny tyto třídy sloučit do jedné buňky, která představuje jednotlivé entity jako celky.

Jak můžete na diagramu 3.3 vidět, tak jediná entita, která se stará pouze sama o sebe je ErrorLog, tato třída zastupuje všechny záznamy chyb, při zpracovávání jednotlivých akcích z kalendáře. Poté zde máme Event, tato entita zastupuje objekty s informacemi o akcích, které si dokáže sama stáhnout z kalendáře a zpracovat je. Zde se dostáváme k nejdůležitější funkcionalitě serveru, kterou je zpracovávání jednotlivých akcí, o kterou se stará funkce reload(). Tato funkce si stáhne informace o všech událostech, které jsou v kalendáři zadané. Při zadávání události do kalendáře je možné zadat pouze: název, čas konání, adresu konání a popis. Z tohoto důvodu jsem vytvořil speciální strukturu popisu, která umožní zadavatelům předat do aplikace další, pro naší aplikaci důležité, informace, kterými jsou typ akce, popis, pořadatelský klub a odkaz na stránku akce. Z tohoto důvodu je důležité dodržovat strukturu popisu, aby se jednotlivé akce zobrazily v aplikaci správně. Podrobnější popis této funkce je uveden v diagramu 3.4. Poslední dvě entity jsou User a Prize, které se starají o administraci uživatelů a cen. Tyto dvě entity se vidí navzájem kvůli tomu, abychom mohli uživateli bezpečně odebrat body za odměnu a uživatel dostal v aplikaci oznámení o možném vyzvednutí další ceny, jakmile na ní bude mít nárok.



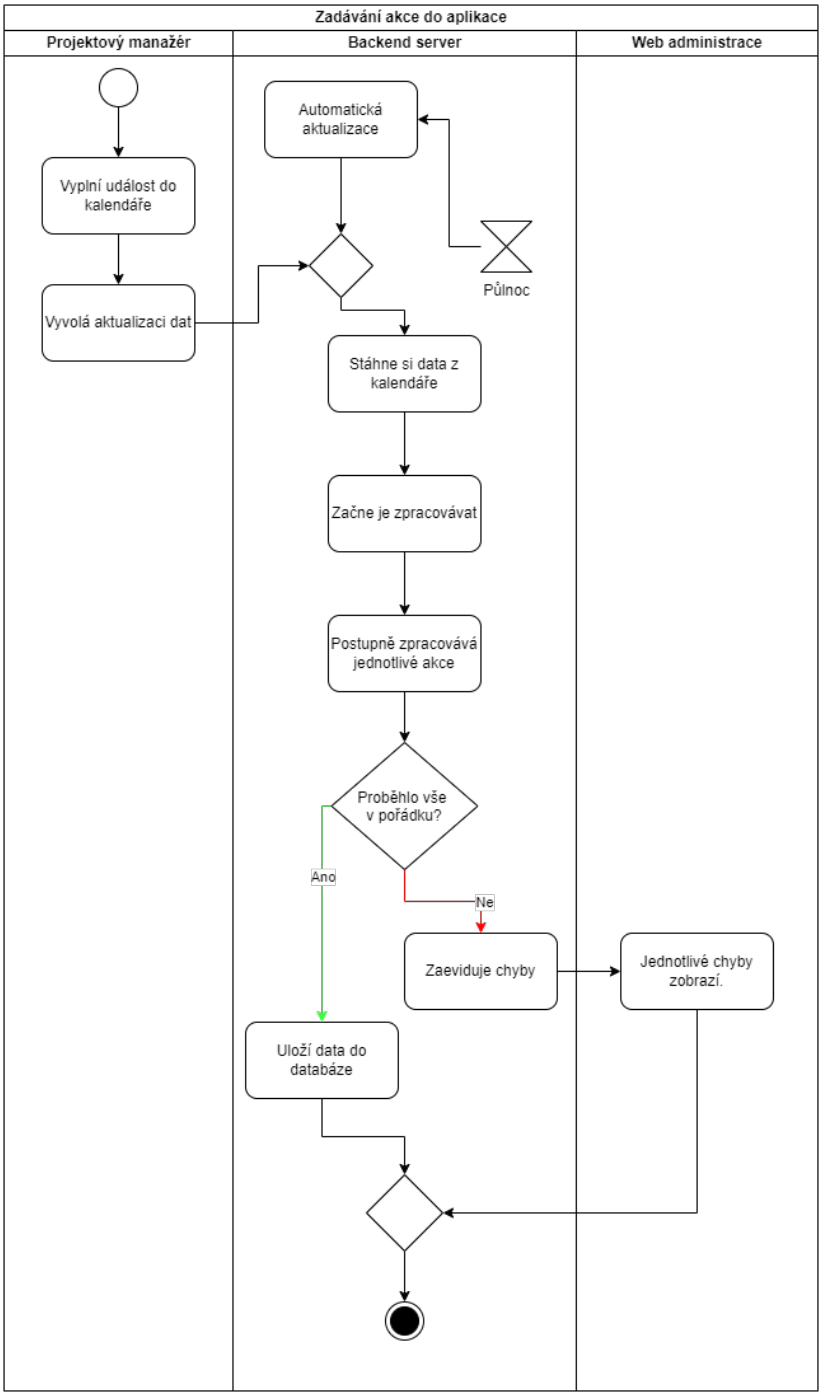
Obrázek 3.3 Diagram komunikace mezi entitami na serveru

3.2.1 Diagramy aktivit

Diagramy aktivit jsou důležitou součástí návrhu architektury každé aplikace, jelikož nám přesně definují procesy a jak jednotlivé úkony v procesech na sebe navazují. V této podkapitole se podíváme na nejdůležitější procesy naší aplikace, kterými jsou přidávání nových akcích do aplikace, přidávání účasti na akcích a žádost o cenu.

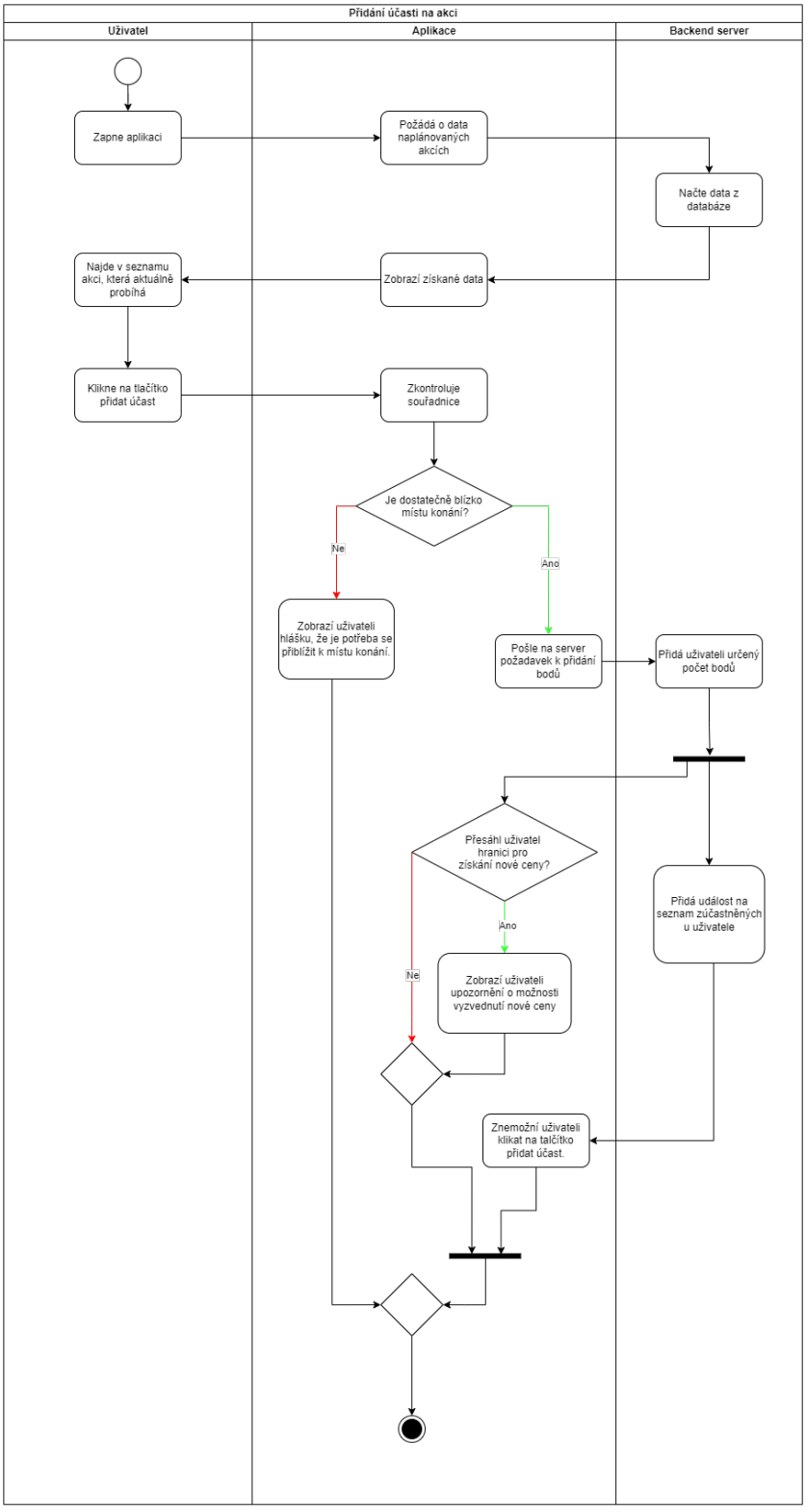
Jako první se podíváme na diagram 3.4 o přidávání nových akcích do aplikace přes kalendář. Celá transakce začíná vyplněním události v kalendáři. Potom má administrátor dvě možnosti, buď provede aktualizaci dat na serveru ručně, pomocí endpointu „/events/reload“ nebo počká do půlnoci, kdy server provádí automatickou aktualizaci. Jakmile se spustí aktualizace, server si stáhne data z kalendáře a začne je postupně zpracovávat. Díky speciálnímu formátu popisu viz. výše je možné, že aplikace bude mít se zpracováním problém. Pokud se nějaký problém objeví, přeruší se zpracovávání. Daná chyba se

zaeviduje a zobrazí se administrátorům ve webové aplikaci. Jestliže vše proběhne v pořádku, data se uloží do databáze a aktualizace je hotová.



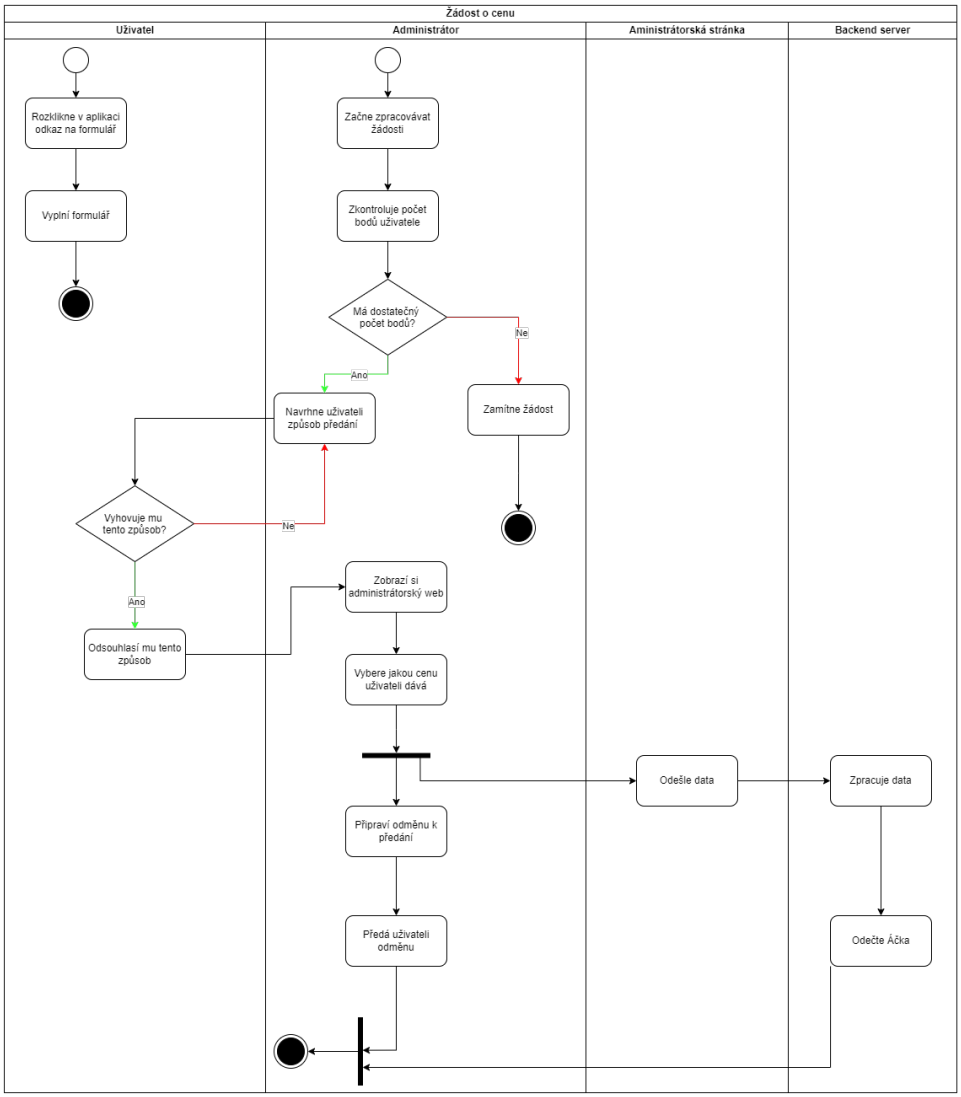
Obrázek 3.4 Diagram aktivity pro přidávání nové akce do aplikace

Další zde máme diagram 3.5 o přidávání účasti na akci. Tento diagram se skládá ze třech účastníku. Jedním z nich je uživatel, který aplikaci otevře a potom v seznamu akcí najde právě konanou akci a klikne na tlačítko „Přidat účast!“. Toto kliknutí se přenese do aplikace, která si načte souřadnice akce a aktuálního uživatele. Porovná je a pokud je uživatel maximálně 100 m od souřadnic akce, tak odešle na server požadavek o přidání bodů. Tento požadavek server zpracuje a přidá akci danému uživateli mezi zúčastněné. Toto pole zúčastněných akcí následně aplikace využívá k tomu, aby znemožnila uživatelům přidat účast vícekrát. Server ještě vyhodnocuje, jestli ziskem nových bodů nemá uživatel nárok na novou hodnotnější cenu. Pokud ano, vrátí aplikaci název hodnotnější ceny, která na to upozorní uživatele. Pokud není uživatel dostatečně blízko, tak aplikace vypíše hlášku, že se musí přiblížit. Aplikace bere souřadnice z adresy, která je uvedená v kalendáři, takže pokud je v něm adresa zadaná špatně nebo úplně chybí, tak aplikace nastaví souřadnice na 0,00; 0,00 a tudíž uživatelé nebudou schopni získat body za účast.



Obrázek 3.5 Diagram aktivity pro přidávání účasti na akce

Poslední je diagram 3.6 ohledně žádosti o cenu. Jak můžete z obrázku vidět, tak máme dva začátky. První je ze strany uživatele, který v aplikaci klikne na tlačítko v profilu a vyplní formulář. Druhý je pro administrátora, který zpracovává žádosti. Ten si otevře žádost a zkontroluje počet Áček uživatele. Pokud má nedostatek Áček, tak je mu žádost zamítnutá a proces končí. Pokud však má dostatečný počet Áček, tak administrátor kontaktuje uživatele, aby se domluvili na předání odměny. Toto může trvat i několik dní, proto je v diagramu uveden cyklus. Jakmile se domluví, administrátor si otevře administrátorský web, kde vybere jaký uživatel zažádal o jakou cenu. Tím se spustí paralelní činnosti, kdy jedna z nich je, že server zpracuje požadavek na odečtení bodů. Druhá je pro administrátora, který musí odměnu připravit a předat uživateli. Jakmile je cena předaná proces končí.



Obrázek 3.6 Diagram aktivity pro zadání a zpracování žádosti o cenu

Kapitola 4

Implementace komponenty ČJJ

Jak jsem již zmiňoval výše, tak pro nás byla nejdůležitější verze pro Android. Od vývojářů Univerzita App jsem dostal jednoduchý projekt, na kterém jsem si mohl vyzkoušet a hlavně se seznámit s programovacím jazykem Kotlin. Jelikož ale z požadavků na novou komponentu vyplynula povinnost mít funkční backend server, který bude obstarávat veškerou business logiku, začal jsem jako první vyvíjet samotný server. Jakmile byl server připravený, přesunul jsem se k práci na funkčním prototypu mobilní aplikace pro operační systém Android. Když byl tento funkční prototyp hotový, bylo potřeba vytvořit jeho kopii i pro operační systém IOS. Část kopie jsem vytvořil v rámci semestrální práce v předmětu BI-IOS a zbytek funkcionalit jsem do aplikace přidal už mimo zmiňovaný předmět. Tyto dva funkční prototypy bylo potřeba vytvořit co nejdříve, aby si je mohli vyzkoušet samotní uživatelé. Ti nám mohli zanechat zpětnou vazbu pomocí formuláře. Poté bylo potřeba ještě vytvořit administrační web, na který SU tolik nepospíchala z důvodu malého počtu uživatelů využívajících samotnou komponentu.

4.1 Vývoj backend serveru

Jelikož jsem se během svého studia seznámil s vývojem serveru v jazyce Java s využitím framework Spring, použil jsem tento postup i ve své bakalářské práci. Server má tří vrstvou architekturu. Jednotlivé vrstvy jsou:

- Repository vrstva – tato vrstva se využívá pro komunikaci s úložištěm dat (v našem případě se jedná o PostgreSQL databázi),
- Service vrstva – tato vrstva se stará o veškerou business logiku a manipulaci s daty,

- Controller vrstva – tato vrstva se stará o zpracování požadavků a případné odeslání odpovídajících dat zpět.

Prvním úkolem bylo propojit náš server s Google kalendářem. Pro tento problém existují dvě řešení, jedno z nich je implementovat do projektu Google API, díky kterému můžeme používat HTTP požadavky k získávání informací o jednotlivých akcích. Druhé z nich je dotaz na specifickou url adresu daného kalendáře, který nám vrátí strukturovaný řetězec se všemi informacemi. Snažil jsem se implementovat první ze zmíněných způsobů, ale bohužel jsem narazil na problém se založením projektu přímo na vývojářských stránkách Google, jelikož bych projekt zakládal na svůj účet, který by s ním byl svázaný, a případné pozdější přenastavování by mohlo způsobit nefunkčnost serveru. Rozhodl jsem se tedy pro druhou variantu, která je snadnější na nastavení. Jediným problém je manuální zpracovávání řetězce, kterému se u některých prvků může měnit formát. Vytvořil jsem tudíž metodu, která je schopná zpracovat tento řetězec. Samozřejmě tato metoda prošla postupem času revizemi, které zlepšovaly zpracovávání. Tato metoda má i samostatný přístupový bod v API, který spustí proces aktualizace akcí z kalendáře viz. obrázek 3.4. Fungování tohoto přístupového bodu bylo zprvu zamýšleno tak, že správce z SU zadá nové akce nebo upraví akce v kalendáři a až bude chtít tyto úpravy zveřejnit, pošle požadavek na tento přístupový bod a na serveru se provede aktualizace. Bohužel se správcům stávalo, že zapomínali poslat požadavek na aktualizaci a následně v aplikaci své změny neviděli. Z tohoto důvodů jsme se s SU dohodli na tom, že se aktualizace bude spouštět vždy o půlnoci. Samotný přístupový bod zůstal zveřejněný z důvodu, kdyby někdo potřeboval změny zveřejnit dříve.

Jakmile jsem měl první úkol hotový, pokračoval jsem s implementací jednotlivých základních přístupových bodů. Původně měl server přístupové body ohledně uživatelů a akcí. Tyto body nám postačili pro vytvoření prvního funkčního prototypu Android verze a vydání první verze nové komponenty.

Později na základě funkčního požadavku F8 viz. 3.1.1 jsem přístupové body rozšířil o možnost administrace cen. Tato nová funkcionalita se využila i v již existujících bodech hlavně v přístupovém bodu „/users/{username}-/addPoints/{eventID}“, který původně nic nevracel, ale s přidáním nové funkcionality začal vracet řetězec, který je buď prázdný (pokud uživatel neměl nárok na hodnotnější cenu) nebo obsahuje název ceny (pokud účastí na aktuální akci překročilo hranici pro získání hodnotnější ceny).

Zároveň s postupným vydáváním nových verzí a rozšiřováním serveru jsem začal přemýšlet nad bezpečností. Tuto otázku jsem si pokládal už na začátku. Z tohoto důvodu API neobsahuje všechny dostupné metody požadavků (hlavní nepoužívanou metodou je DELETE). Toto omezení však nezaručí 100% bezpečnost serveru. Stále jsou zde některé přístupové body jako např. pro přičtení a odečtení bodů, které mohou ovlivnit fungování aplikace hlavně z pohledu podvodů s potvrzením účasti na akci. Po konzultaci s vývojáři Univerzita App na toto téma jsem se rozhodl k vytvoření jednoduchého zabezpečení pomocí hešovaných řetězců posílaných v hlavičce HTTP požadavků. Z důvodu bez-

pečnosti Vám zde bohužel neukáži celý kód zabezpečení, pouze Vám ho slovně nastíním. Jeho hlavní myšlenkou je získat aktuální čas v milisekundách, ke kterému poté různými matematickými operacemi přidáme různá čísla. Z toho nám vznikne první hodnota, která se v hlavičce požadavku posílá na server. Druhá vznikne přidáním náhodného textu k první hodnotě. Tu následně zahešujeme pomocí tajného klíče. Tento hešovací proces je vykonáván pomocí techniky HMAC s využitím hešovací funkce SHA512.

Finální rozšíření serveru přišlo až na konci vývoje v rámci mé bakalářské práce a jednalo se o přidání výpisu chyb. Jelikož se akce zpracovávají programově, může se stát, že v průběhu zpracování nastane nějaká neočekávaná chyba, která vede k ukončení aktualizace a další akce v pořadí již nebudou aktualizovány. Jedná se o poměrně závažný problém a je potřeba, aby o tom byli správci informováni. Naším prvním záměrem bylo informovat správce pomocí emailů. K nastavení emailového serveru by však byl potřeba přístup na server SU, kde je náš backend server spuštěný. Tento přístup však bohužel nemám, jelikož se samotný server vydává pomocí zrcadlení mého repozitáře do repozitáře SU, odkud se poté nahrává na samotný server. Vydali jsme se tedy znovu jednodušší cestou, kterou je vytvoření koncových bodů pro evidenci chyb. Tyto chyby se poté zobrazují na administrátorské webové stránce.

4.2 Vývoj komponenty pro operační systém Android

Jak jsem již zmiňoval výše, verze komponenty pro Android se začala vyvíjet jako první, tudíž jsem si na ní zkoušel technologie, které bych mohl použít pro vylepšení použitelnosti komponenty. Jelikož jsem neměl žádné předchozí zkušenosti s programováním v jazyce Kotlin a celkově s vývojem aplikací pro Android, poskytli mi vývojáři aplikace Univerzita App zkušební projekt, který měl stejné nastavení jako samotná aplikace, a mohl jsem se v něm seznámit s novými technologiemi. Toto mi velmi pomohlo a jakmile jsem se s nimi trochu seznámil, začal jsem s vývojem samotné komponenty ČJJ. Nechal jsem se inspirovat některými komponentami, které byly ve zkušebním projektu, a použil je v prvních návrzích vzhledu naší nové komponenty. Tyto návrhy jsem konzultoval s Ondřejem Váňou, který mi na ně dával zpětnou vazbu. Jakmile jsme se shodli na finálním vzhledu, začal jsem s implementací komunikace se serverem.

Implementace této komunikace byla zpočátku náročnější, protože jsem se seznamoval s framework Ktor. Tento framework se využívá k implementaci požadované síťové komunikace. Jak-mile jsem měl hotovou komunikaci s prvním endpoint, kód pro další endpoint byl stejný, tudíž se dal zkopírovat. Níže můžete vidět ukázku právě zmíněné komunikace.

Ve výpisu kódu 4.1 můžete vidět, že voláme endpoint „/events“, který nám vrátí json reprezentaci akcí, které následně můžeme zobrazit. Poté kon-

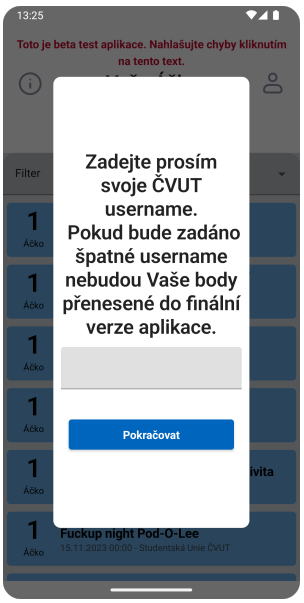
trolujeme, že veškerá komunikace proběhla v pořádku, pokud ne tak vracíme prázdné pole akcí a nastavíme si proměnou localLoad. Tato proměnná nám slouží k rozlišení, jestli má aplikace fungovat normálně nebo je momentálně zařízení bez internetu a mají se pouze zobrazit předchozí načtené akce a tlačítko na znovu načtení po připojení k internetu. Pokud bylo během komunikace vše v pořádku, zpracujeme si json z odpovědi na pole akcí, které se nám uloží do proměnné events. K ukázce je ještě důležité zmínit, že se pro všechny dotazy na server používá jednotná instance client. Ten se vytváří společně s naším viewModel a nastavuje se mu základní url a bezpečnostní hlavička, jak je zmíněno v předchozí podkapitole 4.1.

```
val responseData = client.get { url("/events") }

var events = mutableListOf<Event>()

if (responseData == null || responseData.status != HttpStatusCode.OK)
↪  {
    localLoad.value = true
    return events
}

events =
↪  Json.decodeFromString<MutableList<Event>>(responseData.body());
```

Výpis kódu 4.1 Komunikace Android se serverem pomocí Ktor

První vydaná verze komponenty měla udělané přihlášení přes jednoduchý formulář, kde uživatelé měli vyplnit svoje školní username. Pro toto jsme se rozhodli z důvodu, že na začátku zimního semestru 2023/2024 jsme stále řešili potíže se získáním a nastavením univerzitního SSO (Shibboleth)[1], a SU tlačila na vydání komponenty, aby ji již studenti mohli využívat. Vzhled přihlašování můžete vidět na obrázku 4.1.

[1]Jedná se o přihlašovací bránu ČVUT, která pracuje s údaji studentů a používá se pro přihlášení do univerzitních systémů.



Obrázek 4.1 Přihlašovací obrazovka ve verzi bez SSO

Bohužel tento způsob přihlašování byl velmi problematický, jelikož studenti do formuláře zadávali i emaily a různé další věci. Z tohoto důvodu jsme velice naléhali na zprovoznění SSO, abychom předešli špatným vstupům od jednotlivých studentů. Jednání o získání SSO se však protáhlo a do naší komponenty jsme ho implementovali až na konci zmiňovaného zimního semestru.

Implementace SSO však nebyla příliš jednoduchá. Vývojáři z Univerzita App mi pomohli vytvořit widget, který zobrazí přihlašovací obrazovku. Po správném přihlášení je stránka přesměrovaná na stránku, která zobrazuje všechny dostupné informace o přihlášeném uživateli. Z této stránky si poté vezmeme uživatelské jméno, které si uchováváme na serveru. Jelikož by tato stránka neměla být zobrazena uživateli, tak je potřeba jí skrýt. Bohužel jsem zde narazil na problém, že v některých situacích se stránka uživateli zobrazí a občas je správně skrytá. Kontaktoval jsem s tímto problémem i vývojáře Univerzita App, kteří si s tímto problémem také nevěděli rady. A však postupem vývoje jsem přišel na problém, že si widget, který zobrazuje webovou stránku s přihlášením, uchovává cookies. Což znamená, že když se chce stránka otevřít znovu, tak se vezmou informace z cookies a stránka automaticky přihlásí minulého uživatele. To poté znamená, že se aktualizace rovnou zobrazí na url, kde se uživatel standardně přihlašuje. Tudíž jsem se rozhodl pro widget vypnout využívání cookies, což zabraňuje automatickému přihlašování a tak se vždy proces přihlášení ve widget provede stejně. Toto vyřešilo problém se zobrazováním údajů.

Níže ve výpisu kódu 4.2 můžete vidět proces, který z webové stránky, kde její obsah dostaneme jako řetězec source, získá uživatelské jméno a aktuální uživatelský kód. Původní myšlenka použití uživatelského kódu byla pro zabez-

pečení serveru. Nakonec jsem se rozhodl pro využití jiné metody a tudíž se uživatelský kód nikde nepoužívá. V kódu poté vidíme práci s ViewModel, kde si pouze přeneseme informace o správném přihlášení a uživatelském jméně. Poté už jenom přes context spustíme načítání komponenty znovu, což nám zajistí správné načtení všech potřebných informací ze serveru.

```
try {
    val userCode =
    ↪  source.split("<span>")[1].substringBefore("</span>")
    val username = source.split("<dd>")[1].split("]")[0].replace("[",
    ↪  "")
    viewModel.loginDialog.value = false
    viewModel.changeName(username)
    context.let {
        val intent = context.intent
        context.finish()
        context.startActivity(intent)
    }
} catch (e: Exception) {
    e.printStackTrace()
}
```

Výpis kódu 4.2 Získávání uživatelského jména a aktuálního uživatelského kódu

Poslední velkou částí byla implementace zabezpečení serveru. Znovu zmíním, že z bezpečnostních důvodů nebudu v tom textu uvádět přesné vytvoření zabezpečovacích hlaviček. Pro bližší popis se podívejte do předchozí podkapitoly 4.1. Jelikož jsem byl v kontaktu s vývojáři Univerzita App, měl jsem od nich přístup k implementaci samotné hešovací funkce a následný převod výsledku do šestnáctkové soustavy, tudíž mi pouze stačilo vytvořit správný vzorec pro milisekundy a spustit příslušnou funkci.

Jelikož toto byl můj první projekt v jazyce Kotlin, měl jsem vždy před vydáním každé verze schůzku s vývojáři Univerzita App, kde jsme společně procházeli kód a oni mi vysvětlovali různé tipy a triky, aby byl kód přehlednější a hlavně v souladu s osvědčenými postupy. Z tohoto důvodu je v repozitáři od nich několik commit, které se zabývají právě zmiňovanými estetickými úpravami. Náš vývoj měl dvě fáze, první fáze trvala zhruba do vydání SSO verze a druhá trvá doteď. V každé fázi jsme používali různé repozitáře. Jak jsem již zmínil, měl jsem zkušební repozitář, ve kterém jsem vyvíjel až do implementace SSO. Kluci vždy brali všechny změny a ručně je přenášeli do jejich repozitáře. Při růstu velikosti projektu se tento proces stával téměř neproveditelný. Proto jsme se po implementaci SSO rozhodli, že dostanu přístup do jejich repozitáře, kde budu vyvíjet další funkcionalitu a pomocí pull request se budou vydávat nové verze. Změny jsem poté ke konci vývoje ručně přenesl do původního repozitáře, abych měl všechny změny na jednom místě.

4.3 Vývoj komponenty pro operační systém IOS

Jak je v textu již zmíněno, první funkční prototyp komponenty pro operační systém IOS byl vytvořen v rámci předmět BI-IOS. Jeho vytvoření bylo jednodušší, jelikož jsem nabyl potřebné informace pro vývoj na cvičeních tohoto předmětu a programovací jazyk Swift, ve kterém je komponenta pro IOS napsaná, je velmi podobný Kotlinu. Měl jsem štěstí, že aplikace Univerzita App pro IOS je také psaná ve Swiftu a využívá architekturu MVVM (Model–View–ViewModel), kterou jsme probírali v předmětu BI-IOS. Tato architektura se skládá ze tří částí:

- Model – tato část se stará o uchovávání dat,
- View – tato část se stará o zobrazování dat,
- ViewModel – tato část se stará o veškerou logiku a předává data z Modelu do View.

Dalším urychlením práce byl fakt, že jsem měl hotovou předlohu pro Android, takže jsem v podstatě jenom přepisoval kód z Kotlinu do Swiftu. Samozřejmě, že některé věci jsem musel implementovat jinak. Například IOS verze nemá žádná modální okna, taktéž různé upozorňovací hláška má IOS verze implementované pomocí alert, které musí uživatel ručně odkliknout, aby zmizely.

Znovu jsem se potýkal s podobnými problémy jako u Android verze. Prvním z nich byla implementace komunikace se serverem i přesto, že jsem měl ukázky kódu ze cvičení, musel jsem najít způsob, jak do HTTP dotazů přidat ověřovací hlavičky. Bohužel jsem nenašel univerzální způsob, a proto hlavičky přidávám pomocí rozšíření třídy URLRequest, kde jsem si vytvořil vlastní metodu, která si spočítá potřebné údaje a přidá je do požadavku hlavičky. Jak jsem již zmiňoval jazyky Swift a Kotlin jsou si velmi podobné, proto bylo přenesení některých potřebných funkcí jednoduché. Bohužel samotné hešování se v programovacím jazyce používá trochu jinak, a proto jsem musel tuto funkci vytvořit znovu.

Druhým problémem byla znovu implementace SSO. Vytvoření samotného widget pro přihlášení bylo jednoduché, bohužel získání potřebných informací po přihlášení se mi již nepodařilo docílit, a proto jsem kontaktoval kolegy z Univerzita App, kteří mi pomohli tento problém vyřešit. I zde se objevil podobný problém jako na Android s probliknutím přihlašovacích údajů. Stejně jako u Android jsem problém vyřešil zamezením používání cookies pro automatické přihlášení.

Dalším problémem, na který jsem přišel až při ručním testování, byl Geocoder. Tuto třídu CLGeocoder využívám ve své komponentě k zisku souřadnic z adresy. Na Androidu používám taktéž Geocoder, který funguje správně. Bohužel tento Swift Geocoder u některých adres není schopen získat souřadnice, i když je adresa zadaná správně. Kontaktoval jsem znovu s tímto problémem vývojáře Univerzita App, zda náhodou nepoužívám tuto třídu špatně. Bohužel

jsme způsob na vyřešení tohoto problému nenašli, a tak jsme se rozhodli pro její použití i přes tento problém.

Tuto komponentu jsem kompletně vytvářel ve svém předmětovém repozitáři a následně se do Univerzita App ručně přenesla až finální verze.

4.4 Vývoj administrační webové stránky

Jako poslední jsem začal vyvíjet webovou stránku pro administraci. Myšlenka této webové stránky mě napadla, když jsem zabezpečoval server. Jelikož původní myšlenka byla taková, že lidé z SU, kteří budou mít na starosti zpracovávání žádostí na získání nových cen, budou informace ze serveru získávat přes prohlížeč, kde se budou dotazovat na jednotlivé endpoint API. K tomuto postupu by byl samozřejmě vytvořen návod, aby to mohli vykonávat lidé, kteří nemají velké zkušenosti v IT. Bohužel se zabezpečením serveru pomocí hlavičkových proměnných by tento způsob byl téměř neproveditelný. To mě přivedlo na myšlenku, vytvoření jednoduché webové stránky, kde by se prováděly veškeré administrativní úkony.

Tuto myšlenku jsem poté prezentoval zadavateli Ondřejovi Váňovi, který s tím souhlasil a měl požadavek, aby se přístup do administrace omezil pouze na lidi, kteří mají zároveň přístup do samotného Google kalendáře, odkud bere náš server informace o akcích.

Začal jsem tedy na tomto webu pracovat. K jeho napsání jsem si vybral Javascript framework Vue. Vytvoření základního webu bylo jednoduché díky tomu, že jsem měl vytvořené všechny potřebné endpoint. Problém se však objevil při implementaci komunikace se serverem. Znovu jsem si vytvořil funkce pro výpočet položek do hlaviček HTTP dotazů, bohužel jsem i přes správnost těchto hlaviček dostával error hlášky ze serveru o problémech s přístupem na server. Začal jsem tedy hledat, kde by mohl být problém. Konzole v prohlížeči mi hlásila problém s CORS, což je tzv. Cross-Origin Resource Sharing.

Problém s CORS nastává, když se dotazujete na API z jiného Origin, než na kterém je server spuštěn. Origin potřebuje každá služba dostupná na internetu a představuje nám spojení hostu a portu. Každá taková služba běží na určitém hostovském serveru, který má daný IPv4 nebo IPv6 identifikátor. Tento identifikátor označujeme jako host. Na jednom hostovském serveru je možné mít spuštěných několik služeb, kdy každá služba má svůj specifický port. Origin také bere v potaz označení používaného komunikačního protokolu např. http nebo https.

Tento problém jsem vyřešil povolením specifické adresy pro CORS. Toto nám pomohlo zobrazit data získané ze serveru. Další problém nastal, jakmile jsem se snažil vytvořit POST nebo PUT požadavky. Server mi odpovídal s chybou neautorizováno, i když jsem měl hlavičky nastavené dobře. Až později jsem zjistil, že browser automaticky posílá OPTIONS http dotaz, který nemá nastavené hlavičky, a proto je nedokázal server autorizovat, tento problém jsem vyřešil povolením všech OPTIONS požadavků. To však u POST a PUT po-

žadavků nebylo vše, poté se zde objevili problémy s ochranou proti CSRF útokům.

Tyto útoky se provádí pomocí otevření podvodné stránky, kterou připravil útočník. Otevření této stránky způsobí spuštění určité akce, která odešle HTTP požadavek na server. Pokud je tedy uživatel, který otevře podvodnou stránku, právě přihlášen do naší webové stránky, která má CSRF ochranu vypnutou, tak se dokáže spustit libovolný HTTP dotaz, který může mít destruktivní následky.

Zpět k naší webové stránce. V ní jsem se snažil tento problém s ověřením CSRF vyřešit, ale bohužel žádný ze způsobu implementace mi nechtěl fungovat. Každopádně díky tomu, že je náš server zabezpečen pomocí autorizačních hlaviček, jsem se rozhodl CSRF vypnout. Jelikož aktuálně by útočník musel správně nastavit autorizační hlavičky, k tomu aby se jeho požadavky na API vykonaly. Což by znamenalo znalost aktuální implementace. A kdyby útočník dokázal prolomit aktuální zabezpečení, nemusel by ani vytvářet žádnou podvodnou stránku a mohl by požadavky na API posílat na přímo.

Dalším požadavkem bylo vytvoření jednoduchého přihlášení do administrace. Jelikož všichni lidé, kteří zadávají akce do kalendáře, by měli mít přístup i do administrace. Rozhodl jsem se vytvořit jednoduché řešení, které spočívá v tom, že ve webové stránce je tlačítko na přidávání dalších administrátorů. K jejich přidání bude potřeba znát jejich emailovou adresu, která se vyplní do formuláře. Odeslání formuláře zajistí přidání emailové adresy do seznamu známých adres. Při přihlášení do administrace zadá administrátor jen svojí emailovou adresu, pokud se emailová adresa nachází v databázi, administrátor je přihlášen.

Kapitola 5
Testování

Testování komponenty bych rozdělil do 2 části:

- automatizované testy,
- uživatelské testování.

Obě tyto části jsou velmi důležité, jelikož prověřují vždy jinou funkcionalitu. Při testování jsem se snažil pokrýt co největší část komponenty. K testování jsem využil získané zkušenosti během studia.

5.1 Automatizované testy

V této sekci bych se rád zaměřil na automatizované testy. Jedná se o techniku, kdy se testy spouštějí pomocí příkazu. Ve svém projektu tyto testy spouštím při každém commit pomocí pipeline v gitu, abych odhalil případné chyby, které jsem novými změnami způsobil. Spouštění testů tímto způsobem má výhodu, že se testy spouštějí mimo zařízení vývojáře, které není zatíženo jejich provedením a vývojář tak může pokračovat v další práci. Tyto testy jsem vytvořil pro náš server, který obsahuje veškerou logiku komponenty. Vytvářel jsem Unit i integrační testy. Unit testy vždy testují pouze malou izolovanou funkcionalitu. Integrační testy se používají k testování více částí aplikace najednou.

Jelikož je server napsán v jazyce Java, rozhodl jsem se k testování využít framework JUnit. Jak je zmíněno v kapitole 4.1, server se skládá ze 3 vrstev. Vytvořil jsem testovací sady pro všechny tyto vrstvy. K testování repozitářů jsem si vytvořil jednoduchou paměťovou databázi, pomocí H2 databázového procesoru. Vytvoření probíhalo pomocí anotací nad testovací třídou, tyto anotace vidíte v následujícím výpisu kódu 5.1.

```
@DataJpaTest(properties = {
    "spring.datasource.url=jdbc:h2:mem:testdb",
    "spring.jpa.hibernate.ddl-auto=create-drop"
})
```

Výpis kódu 5.1 Nastavení testovací paměťové databáze

Testování dalších vrstev již probíhalo s automaticky nastavenou testovací databází. Využil jsem v nich techniku mock, která spočívá ve vytvoření testovacích objektů, kterým lze nadefinovat fungování jednotlivých metod. Touto technikou jsem vždy vytvářel objekty z nižších vrstev, jelikož ty už byly otestovány jejich vlastními testy.

Další technikou, kterou bych rád zmínil, je testování kontrolerů. U těchto testů jsem používal MockMvc framework. Tento framework nám dovoluje dotazovat se jednotlivých endpoint a kontrolovat jejich výsledky. Vytvoření této instance MockMvc probíhalo přes anotace v jazyce Java. Tyto anotace můžete vidět ve výpisu kódu 5.2.

Na druhém řádku výpisu kódu můžete vidět konfiguraci objektu MockMvc pomocí anotace `@AutoConfigureMockMvc(addFilters = false)`, kde nastavujeme parametr addFilters na hodnotu false. Toto nastavení nám definuje, že veškeré HTTP požadavky nebudou podléhat kontrole zabezpečovacích hlaviček. Pro toto nastavení jsem se rozhodl z důvodu ulehčení práce s nastavováním hlaviček pro každý testovací požadavek. Ve výpisu kódu 5.2 taktéž můžete vidět ukázku jednoho testu, ve kterém se využívá samotný MockMvc objekt.

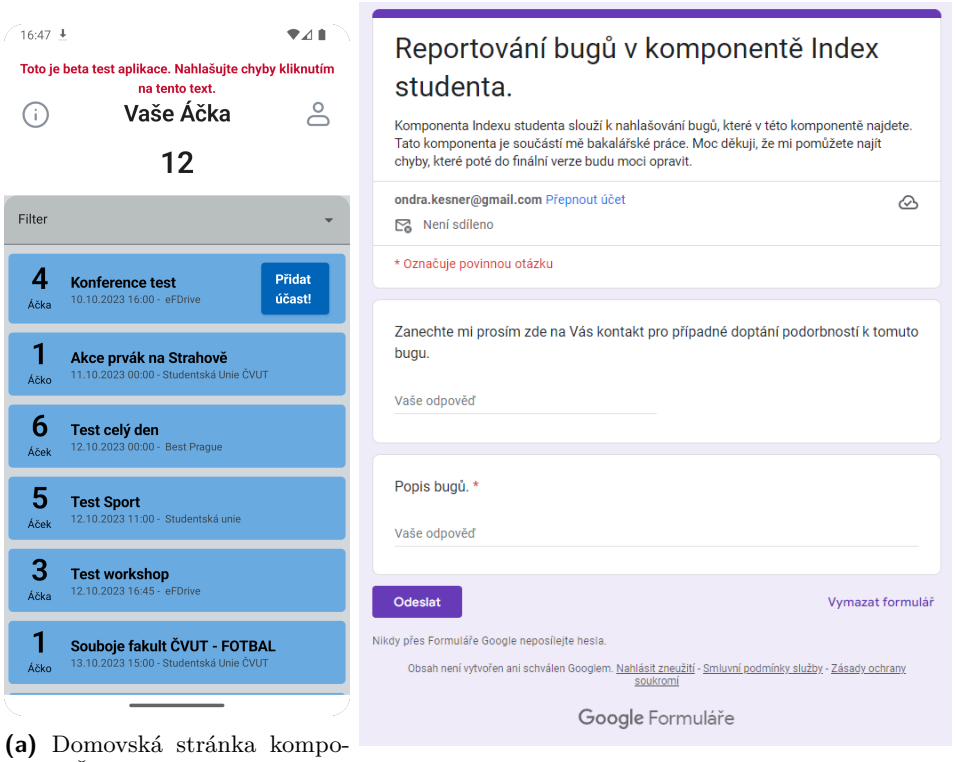
```
@WebMvcTest(UserController.class)
@AutoConfigureMockMvc(addFilters = false)
public class UserControllerTest {
    @Autowired
    MockMvc mockMvc;

    @Test
    @DisplayName("Testing create user method")
    void shouldCreateNewUser() throws Exception {
        User_index user = new User_index();
        user.setUsername("test");
        user.setPoints(0L);

        this.mockMvc.perform(
            MockMvcRequestBuilders
                .post("/users")
                .contentType(MediaType.APPLICATION_JSON)
                .content("{\"username\":\"test\",\"points\":0,\"id\":
                ↪  null}"))
        .andExpect(status().isOk());
    }
}
```

Výpis kódu 5.2 Inicializace MockMvc objektu

5.2 Uživatelské testování

Tato část testování se již zabývala uživatelským testováním samotné komponenty. Jak jsem již zmínil dříve, tak verze komponenty pro Android vyšla již na začátku zimního semestru 2023/2024, kdy se do ní postupně přidávaly aktualizace. Uživatelé měli možnost si pomocí odkazu na domovské stránce komponenty otevřít formulář, který sloužil ke shromažďování chyb v aplikaci. Na obrázku 5.1a můžete vidět domovskou stránku komponenty ČJJ, kde se v horní části obrazovky nachází text, který oznamuje, že se jedná o beta verzi a kliknutím na tento text byl uživatel přesměrován na formulář.

Tento formulář můžete vidět na obrázku 5.1b. Uživatelé do něho mohli uvést kontakt na sebe a podrobný popis problému. Kontakt byl určen pro případnou možnost doptání se. Tento formulář byl využíván hlavně pro verzi na Android, která byla zveřejněna mnohem dříve, tudíž studenti měli delší čas na vyzkoušení aplikace. V těchto problémech se objevily i různé návrhy na vylepšení.



(a) Domovská stránka komponenty ČJJ

(b) Google formulář pro zadání chyby v komponentě

Obrázek 5.1 Ukázky

Jedním z vylepšení bylo vytvoření prokliku pomocí souřadnic přímo do map, kde se uživateli rovnou označilo místo, kde se daná akce bude konat. Tato funkcionalita se mi velmi zalíbila, a tak jsem se rozhodl jí do komponenty zařadit. Implementace byla velmi jednoduchá, jelikož představovala pouze zavolání url adresy Google maps, které automaticky otevřelo prohlížeč map s vyznačenou polohou akce.

Další vylepšení se týkalo odkazů v popisu akce, kdy se mi bohužel nepovedlo zobrazit klikatelné odkazy v textu. Toto jsem následně vyřešil povolením označování v popisu, tudíž uživatelé si mohou odkazy z textu zkopírovat a poté je vložit do prohlížeče.

Samozřejmě problémy z tohoto formuláře byly zachyceny až v produkční verzi komponenty. Já jsem vždy během vývoje aplikaci procházel a snažil jsem se ve vyvíjené části najít jakoukoli chybu. Samotná komponenta byla vždy ručně testovaná jako celek před vydáním nové verze. Toto testování probíhalo i ve venkovních podmínkách, abych se co nejvíce přiblížil k reálné situaci, ve které bude tato komponenta používaná. Tento proces testování se znovu velmi přibližuje tomu automatickému, kde testuji ty nejmenší celky samostatně a poté se postupně spojují dohromady, aby se otestovalo, jestli jednotlivé části fungují společně jako celek.

Kapitola 6

Zhodnocení dosažených výsledků

Při řešení jsem postupoval pomocí kroků, které jsou uvedené v zadání. Začali jsme společně s Ondřejem Váňou a vývojáři Univerzita App vymýšlet, jak by celá nová komponenta ČJJ mohla fungovat, abychom nenarušili strukturu aplikace Univerzita App. Nesmírnou výhodou ve vymýšlení samotné komponenty byl fakt, že jsem jí začínal vytvářet úplně od začátku. Z tohoto důvodu jsem měl volné ruce ve vybírání technologií, které jsme do aplikace implementovali. Jediným omezením byla podmínka, aby se komponenta dala umístit do aplikace Univerzita App. Naštěstí tato aplikace je relativně nová a podporuje nejnovější technologie. Na prvních schůzkách jsme si řekli hlavní požadavky, které by měla komponenta podporovat. Tyto požadavky byly směřovány na ulehčení správy dané komponenty.

Na základě těchto požadavků jsem vymyslel architekturu celé komponenty. V návrhu jsem se snažil použít technologie, které jsem se naučil během studia na vysoké škole, ale všechny technologie pro vývoj mobilních aplikací byly pro mě neznámé. Jednou z neznámých technologií byl programovací jazyk Kotlin a balíček Jetpack Compose. Jak zmiňuji v předchozím textu, vývojáři z Univerzita App mi v začátcích velmi pomohli a snažili se mi vždy vysvětlit, jak správně psát kód, což také vedlo k tomu, že se výsledný kód snaží dodržovat nejlepší praktiky jednotlivých jazyků.

Následně bylo důležité pochopit, jaké procesy musí umět aplikace obstarat. Díky tomu, že už existovala předchozí verze, bylo jednoduché všechny tyto procesy identifikovat. Nejdůležitější však byla jejich úprava, která musela odpovídat navržené architektuře. Veškeré detaily jsem vždy konzultoval s Ondřejem Váňou, který mi byl vždy velmi nápomocný a snažil se do diskuzí přinést vhled samotného uživatele, který s vývojem aplikací nemá takové zkušenosti. Díky tomu jsme v průběhu práce společně vymysleli mnoho vylepšení, které zpříjemní uživateli práci s komponentou.

Samozřejmě v průběhu vývoje komponenty jsme naráželi na různé komplikace, ale také na nové nápady na vylepšení. Jejich řešení jsem se snažil udělat co nejméně invazivní, abych nenarušil navrženou architekturu, a nemusel jsem díky nim předělávat větší část komponenty. Stále jsem se však držel požadavku na ulehčení práce s celou komponentou jako celkem. Tyto akce občas vedly k rozšíření počtu požadavků. Například díky zabezpečení nám do výsledku přibyla i samotná administrátorská stránka, která je určena pro lidi, kteří budou tuto komponentu spravovat.

Výsledkem této práce jsou uživatelsky přívětivé funkční prototypy komponenty do aplikace Univerzita App jak pro operační systém Android, tak i pro IOS. Pro oba operační systémy je komponenta napsaná zvlášť ve specifickém programovacím jazyce pro Android v Kotlinu a pro IOS ve Swiftu. Obě tyto verze jsem se snažil vytvořit co nejvíce podobné, bohužel každá technologie podporuje různé funkcionality jinak, takže drobných nuancí si uživatel může všimnout. Jednou z nich je třeba zobrazení hlášek, kdy Android verze má veškerá upozornění tvořena pomocí hlášek vespod obrazovky, zatímco IOS je má řešené pomocí nativního alert, který se zobrazuje uprostřed obrazovky. Vždy jsem daný kód přizpůsoboval danému prostřední a snažil jsem se využívat nejjednodušší řešení.

Mezi výsledky můžeme také zařadit podrobný popis všech procesů, které běží na pozadí aplikace, a návrh řešení samotné komponenty. Všechny tyto podklady jsou důležité pro udržení této komponenty v provozu co nejdéle. Díky nim bude její rozšíření jednodušší, jelikož tyto podklady pomůžou případným vývojářů se zorientovat v kódu a budou moci rovnou začít s návrhem nové funkcionality. V příloze najdete kompletní dokumentaci endpoint API, které se dají v aktuální verzi použít.

Kapitola 7

Diskuze pro další rozvoj

Jak je zmíněno v předchozí kapitole, tak výsledkem práce jsou uživatelsky přívětivé funkční prototypy. Samozřejmě vždy je možné vylepšovat aktuální verze. V této kapitole bych se rád zaměřil na moje návrhy vylepšení, které by bylo možné do naší komponenty přidat. Veškeré nápady jsem dostal již během tvoření samotné komponenty, ale nebylo časově možné je implementovat do výsledku mojí bakalářské práce. Aktuální verze komponenty má implementované veškeré potřebné funkcionality pro plynulý běh. Následující podkapitoly bych rád věnoval mým návrhům pro další vylepšení uživatelského zážitku z používání komponenty.

7.1 Offline režim

První vylepšení by se mohlo týkat používání aplikace v offline režimu. I přesto že v dnešní době mají téměř všichni lidé mobilní data, tak se může stát, že někteří z nás jich mají méně a nechtějí si je vypotřebovat na používání naší komponenty. To mě přivedlo na myšlenku přidat do komponent tzv. offline režim. Ten by fungoval na bázi zobrazování akcí, které byly načteny ze serveru při předchozím načtení přes internet. Nástřel této funkcionality je připraven v Android verzi, kde se komplexní data uchovávají pomocí knihovny Room Database. Během vývoje jsme se však domluvili, že tuto funkcionalitu do funkčního prototypu dávat nebudeme.

Tato funkcionalita by tedy potřeboval uchovávat komplexní data v telefonu při každém načtení akcí ze serveru, aby se mohlo uživateli něco zobrazit. Následně by bylo nutné vymyslet mechanismus synchronizace dat po znovu připojení k internetu. Toto by se týkalo hlavně synchronizace Áček a seznamu zúčastněných akcí. Na tuto synchronizaci by nejspíše musel vzniknout nový endpoint v API.

Přidání tohoto vylepšení by vyřešilo problém, pokud by se akce konaly mimo internetové připojení např. seznamováky. V rámci tohoto vylepšení by

se taktéž muselo vyzkoušet, jak kvalitně zařízení získává data o poloze, pokud není připojené k internetu.

7.2 Zabezpečení administrátorské webové stránky

Jak zmiňuji v kapitole 4.4, tak kvůli zabezpečení serveru jsme museli vytvořit administrační stránku. Na tuto stránku by měli mít přístup pouze autorizovaní administrátoři. V aktuální verzi je přístup omezen pouze na vyplnění platného emailu, který je uložen v databázi. Tyto emaily jsou do databáze přidávány pomocí tlačítka přímo v administrátorském webu. Jak můžete vytušit, tato metoda zabezpečení (pouze pomocí emailu) není moc bezpečná. Zde se tedy nabízí další prvek pro vylepšení.

V aktuální situaci mě nenapadá žádný bezpečnější způsob přihlašování, abych se držel prosby od Ondřeje Váňi, aby mohl přidávat administrátory do naší webové stránky stejným způsobem, jako když přidává správce do Google kalendáře, odkud si stahujeme veškeré informace o pořádaných akcí.

Nabízela se zde otázka ověřování pomocí odesílání emailů. Nastavení této funkcionality by však bylo velmi náročné, jelikož server a administrátorská stránka jsou umístěny na serverech SU, na které bohužel nemám přístup. Nastavování odesílání emailů ze serveru by bylo velmi náročné a znamenalo by to mnohem větší zapojení správce serverů SU, tudíž jsem od této myšlenky upustil.

7.3 Vylepšení oznamování chyb

Další funkcionalita, která by se mohla vylepšit je oznámení o chybě během získávání dat z kalendáře Google, jelikož teď se toto oznámení ukazuje pouze v administrátorské stránce. Tudíž administrátoři si toho všimnou pouze v situaci, kdy otevřou a přihlásí se do administrátorské stránky. Což si myslím, že administrátoři nebudou dělat tak často. Takže může nastat situace, kdy se například týden nebo i déle neaktualizují data o akcích, což by mohlo ovlivnit správnost informací v komponentě.

K vyřešení tohoto problému se znovu nabízí řešení pomocí odesílání emailů, ale jak již zmiňuji v předchozí podkapitole, tak samotné nastavení by v našem případě bylo velmi zdlouhavé. Zkusil jsem samotné zpracovávání udělat co nejvíce obecné tak, aby k chybám během zpracovávání docházelo co nejméně. Jelikož se však jedná o vstup, který zadává člověk, tak se očekává, že občas tato chyba nastane.

7.4 Vylepšení integrace Google kalendáře

Tato problematika je zmiňovaná i v samotné kapitole 4.1 a je svázaná s předchozím zmíněným vylepšením. Problematika má jednoduché řešení, kterým je

vytvoření Google Cloud projektu, díky kterému bychom poté dostali přístup k Google API, ze kterého bychom získávali již kompletní objekty jednotlivých akcí a nemuseli bychom si je zvlášť zpracovávat. Bohužel vytvořením Google Cloud projektu by se celý projekt s ním svázal. Jelikož jsem tento projekt realizoval sám, bylo by později velmi náročné přenastavovat projekt pod záštitu Studentské unie ČVUT.

Každopádně uvidíme, jak se chování a vlastní zpracovávání dat z kalendáře bude chovat a jestli bude potřeba tuto funkcionalitu předělávat. Kvůli naší vlastní struktuře popisu se stejně bude manuálně zpracovávat popis akce, což může samo o sobě taktéž způsobovat v určitých momentech problém.

7.5 Vylepšení získávání souřadnic

Poslední věc, která by se dala vylepšit je způsob získávání souřadnic. Na tento problém si stěžuji již v kapitole 4.3. Občas se stává, že je adresa zadaná správně, ale bohužel geocoder jí nedokáže rozpoznat a vrátí tak neplatné souřadnice. Tento problém poté může znemožňovat získávání Áček z účasti na dané akci. Největší záhadou je, že problém je pouze v IOS verzi, ale na Android toto funguje správně.

Tudíž jsem se obrátil na vývojáře Univerzita App, jestli se s něčím podobným nesetkali. Bohužel ani oni netušili, čím by mohl být tento problém způsoben. Proto jsme se rozhodli vydat první verzi v této podobě a uvidíme, jak se bude komponenta chovat na koncových zařízení a ne jenom v simulátoru.

Každopádně do budoucna by bylo dobré tento problém vyřešit. Napadlo mě jedno řešení, kterým je rozpoznávání souřadnic již na serveru při aktualizaci. Dalo by se to vyřešit, stejně jako předchozí vylepšení, přímo přes Google pomocí vytvoření Google Cloud projektu. Díky němu bychom dostali přístup ke Google API, které dokáže zmiňovanou funkcionalitu, kdy z adresy potřebujeme získat souřadnice, provést. Čímž by i samotné souřadnice mohly být přesnější díky tomu, že adresy jsou zadávané do kalendáře na základě míst z Google Maps.

Pokud by se tedy implementovalo toto vylepšení, můžeme určitě přemýšlet o implementaci předchozího zmíněného (Vylepšení integrace Google kalendáře).

Příloha A

Documentace API

ČJJ component API

Overview

This API is used by ČJJ component in application Univerzita App

Paths

GET `/events` GET events

Responses

Code	Description	Links
200	OK *Content* /	No Links

GET `/events/reload` GET Reloads the events data from calendar

Responses

Code	Description	Links
200	OK *Content* /	No Links

GET `/users` GET users

Responses

Code	Description	Links
200	OK *Content* /	No Links

POST `/users` POST user

Responses

Code	Description	Links
200	OK *Content* /	No Links

GET `/users/{username}` GET specific user

Parameters

Type	Name	Description	Schema
path	**username** *required*		string

Responses

Code	Description	Links
200	OK *Content* /	No Links

POST `/users/{username}/addPoints/{eventID}` POST Add points for attending specific event

Parameters

Type	Name	Description	Schema
path	**eventID** *required*		string
path	**username** *required*		string

Responses

Code	Description	Links
200	OK *Content* /	No Links
406	Not Acceptable *Content* /	No Links

Code	Description	Links
500	Internal Server Error *Content* /	No Links

GET `/users/{username}/participation` GET ids of participated events of specific user

Parameters

Type	Name	Description	Schema
path	**username** *required*		string

Responses

Code	Description	Links
200	OK *Content* /	No Links
500	Internal Server Error *Content* /	No Links

GET `/users/{username}/participation/profile` GET participated event entities

Parameters

Type	Name	Description	Schema
path	**username** *required*		string

Responses

Code	Description	Links
200	OK *Content* /	No Links

Code	Description	Links
500	Internal Server Error *Content* /	No Links

POST `/users/{username}/removePoints/{prizeID}` POST User buy a specific prize

Parameters

Type	Name	Description	Schema
path	**prizeID** *required*		integer (int64)
path	**username** *required*		string

Responses

Code	Description	Links
200	OK *Content* /	No Links
406	Not Acceptable *Content* /	No Links
500	Internal Server Error *Content* /	No Links

GET `/prize` GET prizes

Responses

Code	Description	Links
200	OK *Content* /	No Links

POST `/prize` POST prize

Responses

Code	Description	Links
200	OK *Content* /	No Links

GET `/prize/enough/{username}` GET Check if user has enough points for new prize

Parameters

Type	Name	Description	Schema
path	**username** *required*		string

Responses

Code	Description	Links
200	OK *Content* /	No Links

GET `/prize/{id}` GET prize by id

Parameters

Type	Name	Description	Schema
path	**id** *required*		integer (int64)

Responses

Code	Description	Links
200	OK *Content* /	No Links
500	Internal Server Error *Content* /	No Links

PUT `/prize/{id}` PUT Edit prize of id

Parameters

Type	Name	Description	Schema
path	**id** *required*		integer (int64)

Responses

Code	Description	Links
200	OK *Content* /	No Links
500	Internal Server Error *Content* /	No Links

DELETE `/prize/{id}` DELETE prize with id

Parameters

Type	Name	Description	Schema
path	**id** *required*		integer (int64)

Responses

Code	Description	Links
200	OK	No Links

GET `/errors` GET errors

Responses

Code	Description	Links
200	OK *Content* /	No Links

DELETE `/errors/{id}` DELETE errors with id

Parameters

Type	Name	Description	Schema
path	**id** *required*		integer (int64)

Responses

Code	Description	Links
200	OK *Content* /	No Links

GET `/admin` GET admins

Responses

Code	Description	Links
200	OK *Content* /	No Links

POST `/admin` POST admin

Responses

Code	Description	Links
200	OK *Content* /	No Links

POST `/admin/login` POST Get specific admin

Responses

Code	Description	Links
200	OK *Content* /	No Links

DELETE `/admin/{id}` DELETE admin with id

Parameters

Type	Name	Description	Schema
path	**id** *required*		integer (int64)

Responses

Code	Description	Links
200	OK	No Links
500	Internal Server Error *Content* /	No Links

Components

Schemas

Event

Properties

Name	Description	Schema
title *optional*		string
description *optional*		string
url *optional*		string
startDate *optional*		string
endDate *optional*		string
points *optional*		integer (int32)
address *optional*		string
longitude *optional*		number (double)
latitude *optional*		number (double)
canAdd *optional*		boolean

Name	Description	Schema
club *optional*		string
category *optional*		string

User_index

Properties

Name	Description	Schema
ID *optional*		integer (int64)
username *optional*		string
points *optional*		integer (int64)

Prize

Properties

Name	Description	Schema
ID *optional*		integer (int64)
name *optional*		string
points *optional*		integer (int64)

ErrorLog

Properties

Name	Description	Schema
errorMesage *optional*		string
ID *optional*		integer (int64)
eventSerialization *optional*		string

User_admin

Properties

Name	Description	Schema
ID *optional*		integer (int64)
email *optional*		string

Bibliografie

1. *Index studenta ČVUT - APK Download for Android | Aptoide* [online]. 2024. [cit. 17. dub. 2024]. Dostupné z: https://index-studenta-cvut.en.aptoide.com/app.
2. JEMEROV, D.; ISAKOVA, S. *Kotlin in Action* [online]. Manning, 2017 [cit. 3. břez. 2024]. ISBN 9781638353690. Dostupné z: https://books.google.cz/books?id=OzkzEAAAQBAJ.
3. *Why Compose - Jetpack Compose - Android Developers* [online]. 2024. [cit. 3. břez. 2024]. Dostupné z: https://developer.android.com/jetpack/compose/why-adopt.
4. *Save data in a local database using Room - Android Developers* [online]. 2024. [cit. 3. břez. 2024]. Dostupné z: https://developer.android.com/training/data-storage/room.
5. DRAKE, J.D.; WORSLEY, J.C. *Practical PostgreSQL* [online]. O'Reilly Media, 2002 [cit. 20. dub. 2024]. ISBN 9781449310288. Dostupné z: https://books.google.cz/books?id=fI1lAgAAQBAJ.
6. FLANAGAN, D. *Java in a Nutshell* [online]. O'Reilly Media, Incorporated, 2005 [cit. 23. břez. 2024]. Desktop quick reference. ISBN 9780596007737. Dostupné z: https://books.google.cz/books?id=mvzgNSmHEUAC.
7. WALLS, C. *Spring in Action, Sixth Edition* [online]. Manning, 2022 [cit. 23. břez. 2024]. ISBN 9781617297571. Dostupné z: https://books.google.cz/books?id=2zVbEAAAQBAJ.
8. KERR, R.; MORSTØL, K. *Beginning Swift: Master the fundamentals of programming in Swift 4* [online]. Packt Publishing, 2018 [cit. 23. břez. 2024]. ISBN 9781789538649. Dostupné z: https://books.google.cz/books?id=EnxeDwAAQBAJ.

9. BARKER, C. *Learn SwiftUI: An introductory guide to creating intuitive cross-platform user interfaces using Swift 5* [online]. Packt Publishing, 2020 [cit. 23. břez. 2024]. ISBN 9781839210877. Dostupné z: `https://books.google.cz/books?id=t1TbDwAAQBAJ`.

Obsah příloh

A.1 Repozitář s backend serverem

Dostupný na `https://gitlab.fit.cvut.cz/kesneond/cjj-api`

- `README.md` ..... stručný popis obsahu média
- `server` ..... adresář s projekt backend serveru
  - `src`
    - `main` ..... zdrojové kódy implementace
    - `test` ..... zdrojové kódy testů
  - `build.gradle.kts` ..... konfigurační soubor pro správu závislostí
- `administration_web` ..... adresář pro administrační webovou stránku
  - `src` ..... zdrojové kódy
  - `package.json` ..... konfigurační soubor pro správu závislostí
- `.gitlab-ci.yml` ..... soubor s nastavením pipeline v Gitlab
- `docker-compose.yml` ..... soubor pro inicializaci Docker kontajneru

A.2 Repozitář s komponentou pro Android

Dostupný na `https://github.com/thenracker/index_cvut`. Tento repozitář se nachází na GitHub z důvodu snadnějšího sdílení obsahu s vývojáři Univerzita App.

- `README.md` ..... stručný popis obsahu média
- `app` ..... adresář s projekt pro Android
  - `src`
    - `main` ..... zdrojové kódy implementace

A.3 Repozitář s komponentou pro IOS

Dostupný na `https://gitlab.fit.cvut.cz/kesneond/ios`

- `README.md` ..... stručný popis obsahu média
- `CJJ` ..... adresář s projekt pro IOS
  - `CJJ` ..... zdrojové kódy implementace

A.4 Repozitář s textem

Dostupný na `https://gitlab.fit.cvut.cz/kesneond/kesneond-bp-text`

- `readme.txt` ..... stručný popis obsahu média
- `images` ..... složka s použitými obrázky
- `text` ..... zdrojová forma práce ve formátu LaTeX
- `ctufit-thesis.cls` ..... soubor nastavující formátování dokumentu
- `ctufit-thesis.tex` .... soubor spojující části ze složky text do celistvého textu