FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Snadné vykazování činností – OS Android
Student:	Jakub Nagy
Vedoucí:	Ing. Jiří Hunka
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2023/2024

Pokyny pro vypracování

Cílem této práce je zajistit snadné vykazování a trackování času uživatelů, kteří realizují část své práce za pomocí mobilních zařízení s OS Android. Například volají s klientem, zajišťují telefonní support apod. Zároveň je třeba zajistit, aby byl vykazovaný čas zapsán k patřičným projektům a úkolům v používaném issue tracking systému.

Postupujte v těchto krocích:
1. Analyzujte potřeby uživatelů trackujících čas pomocí mobilního zařízení s OS Android. Zároveň analyzujte možnosti OS Android v této oblasti. Analyzujte také možnosti využití existujícího řešení pro synchronizaci časových výkazů Timer2Ticket.
2. Na základě analýzy proveďte vhodný návrh.
3. Implementujte funkční prototyp Vámi navrženého řešení.
4. Proveďte testování prototypu s cílovými uživateli.
5. Na základě testování navrhněte a realizujte úpravy, které zlepší celkovou použitelnost Vašeho řešení.
6. Pokuste se zajistit vydání hotové aplikace.
7. Shrňte dosažené výsledky, navrhněte možná rozšíření do budoucna.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 17. února 2023 v Praze.

Bakalářská práce

SNADNÉ VYKAZOVÁNÍ ČINNOSTÍ – OS ANDROID

Jakub Nagy

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Jiří Hunka
16. května 2024

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Chtěl bych poděkovat všem, kteří mi pomohli při tvorbě této bakalářské práce. Velké poděkování patří vedoucímu práce Ing. Jiřímu Hunkovi za jeho cenné rady, připomínky a za čas věnovaný této práci. Dále děkuji Ing. Oldřichu Malcovi za pomoc při specifikaci požadavků aplikace. Poděkování patří také účastníkům uživatelského testování, kteří mi poskytli cennou zpětnou vazbu. Děkuji také své rodině, která mi byla během celého studia velkou podporou. Na závěr bych rád poděkoval své přítelkyni a všem přátelům, díky kterým bylo celé studium o mnoho příjemnější.

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 16. května 2024

Abstrakt

Tato práce se zaměřuje na návrh, vývoj a implementaci Android aplikace, která automaticky sleduje a eviduje čas strávený telefonováním v rámci pracovních aktivit. Aplikace tak umožňuje uživatelům ušetřit čas a zefektivnit proces vykazování odpracované doby.

Práce nejprve analyzuje existující řešení a potřeby uživatelů v dané oblasti. Následně se zaměřuje na prozkoumání možností programování Android aplikací s důrazem na sledování a evidenci telefonních hovorů. Na základě provedené analýzy je navržen a implementován funkční prototyp aplikace. Při návrhu je brán zřetel na doporučené architektonické principy a implementace využívá moderní nástroje i knihovny. Aplikace spolupracuje se známým nástrojem pro sledování času Toggl Track. Práce dále popisuje proces vydání aplikace na distribučních platformách. Vytvořený prototyp aplikace je následně podroben uživatelskému testování s cílem získat zpětnou vazbu ohledně použitelnosti a funkčnosti. Na základě zjištěných poznatků jsou provedeny úpravy aplikace. Závěrečná část práce shrnuje dosažené výsledky, hodnotí splnění cílů a navrhuje také možné úpravy a vylepšení pro budoucí verze aplikace.

Klíčová slova mobilní aplikace, Android, Kotlin, vykazování času, Jetpack Compose, Toggl Track, automatické vykazování hovorů

Abstract

This thesis focuses on the design, development and implementation of an Android application that automatically monitors and records the time spent on the phone during work activities. The application thus enables users to save time and streamline the time reporting process.

The work first analyzes existing solutions and user needs in the given area. Subsequently, it focuses on exploring the possibilities of programming Android applications with an emphasis on tracking and logging phone calls. Based on the analysis, a functional prototype of the application is designed and implemented. The design takes into account the recommended architectural principles, and the implementation uses modern tools and libraries. The application works with the well-known time tracking tool Toggl Track. The thesis further describes the process of releasing the application on distribution platforms. The created prototype of the application is subsequently subjected to user testing in order to obtain feedback regarding usability and functionality. Based on the findings, adjustments to the application are made. The final part of the work summarizes the achieved results, evaluates the fulfillment of the objectives and also suggests possible modifications and improvements for future versions of the application.

Keywords mobile application, Android, Kotlin, time tracking, Jetpack Compose, Toggl Track, automatic call reporting

Seznam zkratek

REST	Representational State Transfer
API	Application Programming Interface
FR	Functional Requirement
NFR	Nonfunctional Requirement
UC	Use Case
URL	Uniform Resource Locator
SDK	Software Development Kit
AAB	Android App Bundle
APK	Android Application Package
XML	Extensible Markup Language
UI	User Interface
SSOT	Single Source of Truth
ORM	Object Relational Mapping
SQL	Structured Query Language
DI	Dependency Injection
JSON	JavaScript Object Notation
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
AES	Advanced Encryption Standard
CBC	Cipher Block Chaining
AI	Artificial Intelligence

Kapitola 1
Úvod

Trendem moderní firemní kultury je flexibilní pracovní doba. Zaměstnanci si odpracovaný čas sami zaznamenávají a vykazují. Zaměstnavatel potom vidí, kolik času svou prací strávili. Výkazy se pomocí firemních aplikací přiřazují k projektům a konkrétním úkolům, aby se v rámci firmy vytvářel přehled, kolik času se strávilo na jednotlivých projektech a úkolech.

Flexibilní pracovní doba přináší mnoho benefitů. Bohužel s každou pozitivní věcí se obvykle pojí i problémy. Jeden z těchto problémů spočívá v zapomínání vykazování času. Čas, který si zaměstnanec nevykáže, potom pro vedení jakoby neexistuje. Hlavním aktérem, který na této chybě trpí, je právě zaměstnanec. Pokud je placený hodinově, neodpracovaný čas nebude zaplacen, a pokud pracuje na úvazek, musí časový úvazek v budoucnu splnit další prací.

Dalším trendem moderní doby, který souvisí s technologickým pokrokem a změnami v pracovní kultuře, je používání mobilních telefonů v rámci zaměstnání. Díky mobilním technologiím jsou lidé schopni pracovat kdykoliv a kdekoliv, pokud mají po ruce telefon nebo tablet.

Spolu s tímto trendem se ale na povrch dostává nemalý problém. Stále častěji se setkáváme s lidmi, kteří pracují i mimo svou pracovní dobu. Rychlá odpověď na zprávu, krátký telefonát řešící pracovní záležitosti nebo kontrola emailové schránky. Žádná z těchto činností netrvá dlouho. Až by se dokonce mohlo zdát, že takto krátké činnosti v porovnání s celkovou pracovní dobou vůbec nestojí za zmínku. Opak je ale pravdou. Pokud jsou tyto činnosti vykonávány opakovaně celý den, může jejich celkový čas překvapit.

Tato práce si klade za cíl vyvinout Android aplikaci, která automaticky sleduje a eviduje čas strávený u telefonních hovorů. Aplikace uživatelům ušetří čas a zvýší přesnost vykazování. Tím pomůže vyhnout se ztrátě mzdy a sníží práci nad rámec úvazku. Díky automatickému sledování telefonních hovorů bude aplikace představovat inovativní řešení v porovnání s existujícími nástroji, které se obvykle zaměřují na manuální zadávání údajů.

Téma práce bylo autorem zvoleno především z důvodu, že úspěšně vytvořená aplikace by mohla pomoci mnoha lidem, kteří mají problém s kontrolou své pracovní doby. Zároveň autor bere celou práci jako velkou výzvu, protože se jedná o jeho první zkušenost s programováním pro systém Android. Autor věří, že ho práce na bakalářské práci naučí mnoho nových věcí a že získá cenné zkušenosti do budoucího profesního života.

Práce úzce souvisí s bakalářskou prací od Adama Breznena „Snadné vykazování činností - iOS“ [1], která se zabývá podobnou aplikací, ovšem pro operační systém iOS. Díky větší flexibilitě a otevřenosti systému Android, které spolu umožňují zcela odlišný přístup k telefonickým datům, bude řešení zcela odlišné.

Další prací, která se v textu několikrát zmíní, je práce „Synchronizační middleware mezi projektovým systémem a aplikací na sledování času stráveného na projektech“ od Víta Štefana [2]. Na základě analýzy tohoto nástroje bude rozhodnuto, zda-li by nástroj nemohl sloužit jako jedna z komponent vyvíjené aplikace.

Kapitola 2
Cíle práce

Hlavním cílem této bakalářské práce je navrhnout a implementovat funkční aplikaci, která umožní uživatelům automaticky sledovat a evidovat čas strávený telefonováním při práci na projektech a úkolech. Aplikace bude určena pro systém Android a využijí ji všichni, pro které je telefonování součástí jejich zaměstnání.

Před samotným vývojem bude nutné provést analýzu potřeb uživatelů a současných řešení dané problematiky. Dále také prozkoumat moderní možnosti programování Android aplikací, s hlavním zaměřením na možnosti systému Android v oblasti sledování a evidence telefonních hovorů. Dílčím úkolem v analytické části bude také prozkoumání aplikace Timer2Ticket a zvážení jejího využití v rámci vyvíjené aplikace.

Na základě provedené analýzy bude navržen a implementován funkční prototyp aplikace. Následně proběhne uživatelské testování, které získá zpětnou vazbu ohledně použitelnosti a funkčnosti aplikace. Díky zjištěným poznatkům z testování budou navrženy a implementovány vhodné úpravy pro zlepšení použitelnosti aplikace.

V závěrečné fázi práce budou prozkoumány možnosti vydávání Android aplikací. Na základě zjištěných informací bude proveden pokus o vydání aplikace.

Závěrem celé práce bude shrnutí dosažených výsledků, zhodnocení splnění cílů a vyhodnocení přínosů aplikace. Na základě zjištěných informací budou navrženy možné úpravy a vylepšení, které bude možné implementovat v budoucích verzích aplikace.

Kapitola 3
Analýza

Na úvod analytické části práce představí autor problém, se kterým se lidé ve svém zaměstnání mohou setkávat. Na základě popisu problému představí autor jeho řešení v podobě mobilní aplikace. Dále provede analýzu existujících řešení, pomocí nichž lze vykazovat čas, a pokusí se najít aplikaci, která se zaměřuje na čas strávený telefonováním. Dále provede analýzu potřeb cílových uživatelů aplikace. Po zvážení všech zjištěných informací vytvoří a popíše softwarové požadavky aplikace a představí také její případy užití. Na závěr kapitoly představí nástroj Timer2Ticket a prozkoumá možnosti jeho využití v aplikaci.

3.1 Nastínění současné situace

Ideálním uživatelem aplikace, na kterou se tato práce zaměřuje, je člověk, který si ve svém zaměstnání sleduje odpracovaný čas. Pracuje ve společnosti, která se zabývá různými projekty a používá nástroje pro správu těchto projektů (viz sekce 3.3).

Jeho pracovní náplň zahrnuje komunikaci s ostatními osobami, přičemž významnou část této komunikace tvoří telefonní hovory. Protože si sleduje svůj odpracovaný čas, musí každý hovor manuálně zaznamenávat. Existují sice aplikace, které zjednodušují vytváření časových výkazy, ale do většiny z nich je nutné manuálně zasahovat. To pro zaměstnance představuje práci navíc a ztrátu času, který by mohl věnovat jiným činnostem.

Ruční způsob vykazování telefonních hovorů je rutinní činností, která s sebou nese riziko chyb při přepisování a zadávání údajů. Existuje mnoho situací, kdy nelze telefon přímo používat, ale přesto lze telefonovat, například při řízení vozidla. Takový hovor lze vykázat až po dokončení jízdy, ale v tu dobu může být telefonát zapomenut nebo mohou být opomenuty důležité detaily, které se hovoru týkaly.

Z popisu je patrné, že ruční vykazování telefonních hovorů je zbytečná rutina, která zabírá zaměstnancům čas a snižuje jejich efektivitu. Existují možnosti, jak lze tento proces automatizovat, a jednou z nich je právě mobilní aplikace. Výstupem této práce bude aplikace, která automaticky vykazuje telefonní hovory uživatele a spolupracuje s projektovými nástroji, umožňujícími přiřazení hovorů k projektům a úkolům.

3.2 Analýza existujících řešení

Na trhu existuje řada aplikací, které umožňují uživatelům sledovat a vykazovat svůj čas. Tyto aplikace podporují široké spektrum platforem – existují jako webové, desktopové i mobilní aplikace. V této části několik takových aplikací analyzuji a popíšu.

Většina aplikací se soustředí na obecné činnosti, které uživatelé měří pomocí časomíry. Málo aplikací se zaměřuje na sledování telefonních hovorů. Provedením této analýzy se takové aplikace pokusím identifikovat.

Hlavní vlastnosti, na které se zaměřím, budou:

- **Integrace s jinými aplikacemi**: Lze aplikaci propojit s dalšími aplikacemi? Je možné ji propojit s nástroji pro správu projektů?
- **Kompatibilita se systémem Android**
- **Podpora sledování aktivity na telefonu**: Dokáže aplikace sledovat aktivity provedené na mobilním zařízení a jednoduše je vykazovat?
- **Podpora sledování telefonních hovorů**: Dokáže aplikace detekovat telefonní hovory? Umožňuje uživateli jejich jednoduché vykazování? Nabízí možnost automatického vykazování telefonních hovorů?

V závěru ze získaných informací rozhodnu, zda má smysl aplikaci popsanou v úvodu vytvářet. Pokud bude analýzou zjištěno, že existující aplikace dokážou problém snadno vyřešit, tvorba nové aplikace nebude mít smysl.

Aplikace, které jsem analyzoval, byly vyhledány pomocí nejpoužívanějšího webového vyhledávače Google. Hesla, která byla použita, byla: „call time tracking app“, „call time tracker“ a „automatic call time tracking app“. Ve výsledcích se objevilo mnoho produktů určených pro velká call centra. Použití těchto aplikací je zcela odlišné od zamýšleného použití vyvíjené aplikace, proto jsem je ignoroval. Pro analýzu jsem zvolil čtyři aplikace, které měly podle popisu největší šanci na vyřešení problému s automatickým vykazováním telefonních hovorů.

3.2.1 Toggl Track

Toggl Track je nástroj určený pro jednotlivce i týmy, který umožňuje snadné sledování a vykazování času. Je dostupný jako webová aplikace, desktopová aplikace pro Windows a MacOS, doplněk do prohlížečů Chrome a Firefox a také jako mobilní aplikace pro systémy iOS a Android. [3]

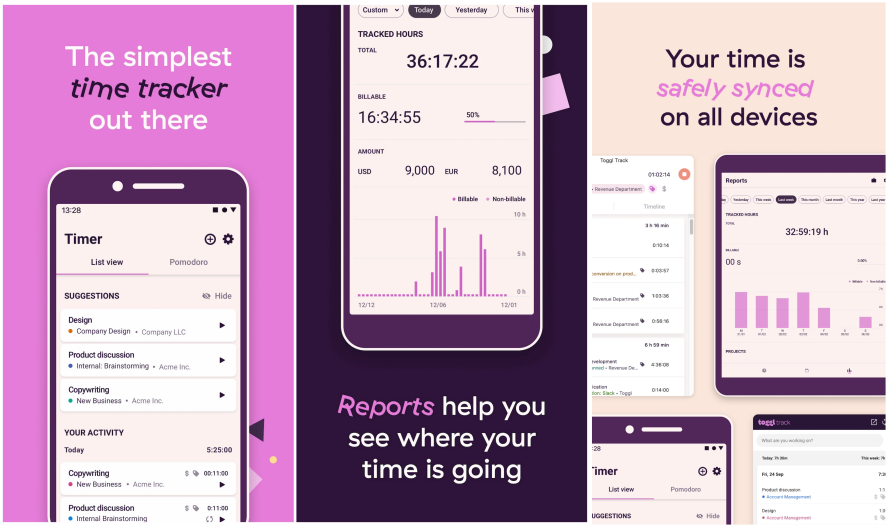
Aplikace se používá velmi intuitivně. Základem je časomíra, kterou uživatel spouští a pozastavuje. Po zastavení časomíry se vytvoří časový záznam, který lze přiřadit k projektu a popsat, nebo označit různými značkami (*tagy*) pro následnou kategorizaci.

Kromě vykazování času umožňuje aplikace vytvářet detailní přehledy o vykázaném čase, které lze filtrovat podle projektů, úkolů, značek a dalších kritérií. Toggl Track také umožňuje vytvářet a spravovat projekty, ke kterým lze přiřazovat jednotlivé časové záznamy. Další funkcí je správa týmů, ve kterých lze s ostatními členy sdílet projekty a sledovat aktivitu ostatních.

Toggl Track nabízí bezplatný tarif s omezenými funkcemi a placené tarify. Bezplatný tarif je určen pouze pro 5 uživatelů v týmu a je vhodný pro jednotlivce i menší firmy. Placené tarify rozšiřují maximální velikost týmu a jsou vhodnější pro větší společnosti s více zaměstnanci. Pro samotné vykazování času pomocí časomíry postačuje bezplatná verze.

Aplikace definuje několik pojmů, které uživatel během používání potká. Pro lepší představu zde uvádím popis některých z nich: [4]

- **Organizace**: Představuje obdobu firmy v reálném světě. V rámci organizace se spravují uživatelé a jejich přístupy.
- **Workspace**: Prostor, kam se ukládají všechny aktivity v Toggl Track.
- **Projekt**: Soubor aktivit zaměřených na dosažení určitého cíle. K projektům lze přiřazovat časové záznamy.



Obrázek 3.1 Snímky z Android aplikace Toggl Track. Získáno z [7].

.

- **Časový záznam**: Záznam o čase stráveném na určité aktivitě, který obsahuje informace o startu, konci, délce trvání a popisu. Přiřazuje se k projektům a může být označen tagy.
- **Tagy (značky)**: Pomáhají kategorizovat časové záznamy a usnadňují tak jejich vyhledávání a filtrování.

Výhodou nástroje je automatická synchronizace dat mezi všemi aplikacemi přihlášenými ke stejnému účtu, což umožňuje například spouštět časomíru z mobilní aplikace a upravovat výkazy v přehlednější webové aplikaci.

Integrace s jinými aplikacemi

Toggl Track nabízí mnoho integrací s aplikacemi třetích stran. Nechybí možnost propojení s nástroji pro správu projektů Asana a Jira, s kalendáři Google Calendar a Outlook Calendar, a dokonce i s aplikací Adobe Photoshop pro pohodlné vykazování času přímo v této aplikaci. Dále Toggl Track nabízí přes 100 doplňků do prohlížečů Chrome, Firefox a Edge, pomocí nichž lze spolupracovat například s GitHubem, Gitlabem nebo Slackem. [5]

Pokud tyto možnosti nestačí, poskytuje aplikace spolehlivé REST API, které umožňuje vytvářet vlastní integrace. API nabízí širokou škálu funkcí, jako je vytváření časových výkazů, spouštění a zastavování časovače, vytváření projektů nebo značek a mnoho dalšího. Veřejná dokumentace je dostupná na [6].

Android aplikace

Toggl Track je dostupný jako mobilní aplikace pro systém Android. Oproti webové verzi postrádá některé funkcionality, které umožňují spravovat účet nebo projekty, ale na druhou stranu obsahuje navíc pomodoro časovač pro zvýšení soustředěnosti a produktivity. Na obrázku 3.1 je vidět několik snímků z aplikace.

Sledování a vykazování aktivity na telefonu

Podle dostupných informací mobilní aplikace Toggl Track neumožňuje sledování ani vykazování aktivity provedené na zařízení. Tato funkce je momentálně podporována pouze v desktopové

aplikaci pro zařízení s operačními systémy Windows a macOS. Nicméně je možné, že v budoucnu bude tato funkce rozšířena i do Android aplikace.

Vykazování telefonních hovorů

Stejně jako v případě obecné aktivity na telefonu, neumožňuje mobilní aplikace sledování telefonních hovorů a jejich automatické vykazování.

3.2.2 Timing

Timing je aplikace pro macOS. Aplikace na pozadí sleduje veškerou činnost uživatele. Čas strávený činnostmi ukládá a poté přiřazuje pod projekty. Uživatel po dokončení práce může celou svou časovou historii zkontrolovat a případně upravit nebo přeřadit pod jiné projekty. Aplikace umožňuje vytvářet časové záznamy i ručně. Například pokud uživatel stráví hodinu na schůzce v kavárně, může tuto činnost ručně do aplikace přidat. Aplikace je na prvních 30 dní zdarma, později je ale potřeba zakoupit měsíční plán. Cena nejlevnějšího z nabízených začíná na 10 eurech. [8]

Integrace s jinými aplikacemi

Timing jako desktopová aplikace nenabízí veřejné API. Nemá ani integraci s žádným nástrojem pro správu projektů. Jediné integrace, které lze v aplikaci zapnout, jsou s kalendářovou aplikací, poznámkovou aplikace a iPhonem nebo iPadem.

Android aplikace

Aplikace Timing je dostupná pouze pro macOS, není k dispozici žádná alternativa pro zařízení se systémem Android. Jelikož se ale jedná o jedinou nalezenou aplikaci, která podle popisu dokáže vykazovat telefonní hovory, považuji za vhodné zde aplikaci zmínit.

Sledování a vykazování aktivity na telefonu

V nastavení aplikace je možné zapnout sledování času stráveného na iPhonech nebo iPadech. Informace o těchto činnostech aplikace získává pomocí funkce „Čas u obrazovky“, která na Apple zařízeních sleduje a kontroluje čas strávený v konkretních aplikacích. Kvůli tomuto specifickému způsobu implementace se tento způsob nedá využít na zařízeních se systémem Android. [9]

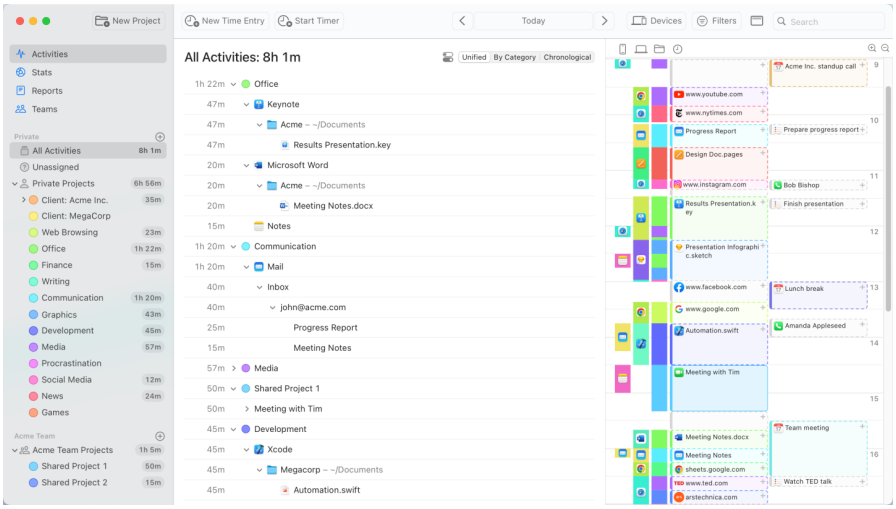
Na obrázku 3.2 je k vidění snímek z aplikace, na kterém je seznam úspěšně vysledovaných aktivit.

Vykazování telefonních hovorů

Aplikace Timing inzeruje možnost sledování a vykazování telefonních hovorů [10]. Nicméně, během testování nebyla tato funkce úspěšně zprovozněna, přestože bylo postupováno podle oficiálního návodu.

3.2.3 Timely

Timely je aplikace pro automatické zaznamenávání a vykazování činností, které uživatel provádí na svém zařízení. Podle oficiálního popisu dokáže sledovat veškerou provedenou aktivitu. Jednotlivé aktivity lze následně přiřazovat pod projekty a dále upravovat. Timely je dostupná pro řadu operačních systémů – macOS, Windows, Android i iOS.



Obrázek 3.2 Snímek obrazovky z aplikace Timing. Získáno z [8].

Integrace s jinými aplikacemi

Timely podporuje integraci s kalendáři, emaily a nabízí veřejné API pro vlastní integrace. Z nástrojů pro správu projektů podporuje integraci s nástrojem Asana a Jira.

Android aplikace

Timely nabízí pro své uživatele Android aplikaci. Po nainstalování a spuštění je nutné se zaregistrovat nebo přihlásit. Není potřeba zadávat žádné platební údaje. Po zaregistrování se spustí 14denní zkušební lhůta, která je zdarma. Na obrázku 3.3 lze vidět snímek obrazovky z aplikace.

Sledování a vykazování aktivity na telefonu

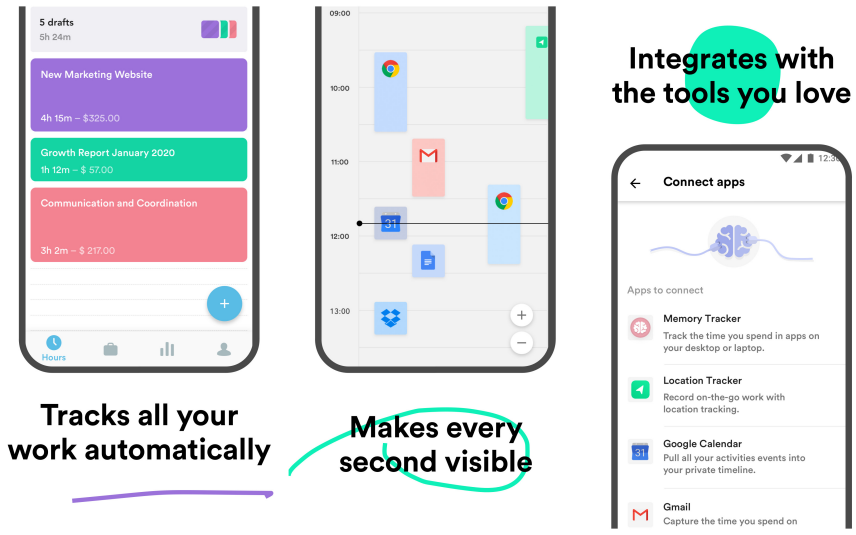
Podle popisu na Google Play by aplikace měla automaticky sledovat téměř veškerý čas strávený na telefonu [11]. Nicméně, během testování se žádné aktivity neobjevovaly. Při hledání příčiny bylo zjištěno, že pro sledování činností je nutné nainstalovat jinou aplikaci Memory, která se s Timely propojí. Právě tato aplikace má potom na starosti sledování všech aktivit. Bohužel je tato aplikace dostupná pouze pro macOS a Windows. Timely tedy nelze využít pro sledování činností na zařízeních se systémem Android.

Vykazování telefonních hovorů

Podle oficiálního článku zveřejněného na stránkách Timely, umožňovala předchozí verze aplikace automatické sledování a vykazování telefonních hovorů. Kvůli změně v Google Play podmínkách musela být ale tato funkce odstraněna. V aktuální verzi tato funkcionalita chybí, a proto ji nebylo možné otestovat. Jediná dostupná informace je ve zmíněném článku, podle kterého dokázala aplikace sledovat a vykazovat provedené hovory, stejně jako je hlavní požadavek vyvíjené aplikace. [12]

3.2.4 RescueTime

RescueTime je aplikace pro automatické sledování a vykazování času stráveného v aplikacích i na webu. Je dostupná pro operační systémy macOS, Windows, iOS a Android, a také jako webová aplikace. Kromě sledování času nabízí také funkce pro soustředění, jako jsou upozornění



Obrázek 3.3 Snímek obrazovky z aplikace Timely. Získáno z [11].

na ztrátu soustředění nebo blokování vybraných stránek a aplikací v době soustředění. Aplikace umožňuje i manuální zaznamenávání činností. [13]

Integrace s jinými aplikacemi

RescueTime nabízí stejně jako Toggl Track více druhů aplikací, mezi kterými probíhá vzájemná synchronizace dat. Má také veřejné API pro vlastní integrace, ale dokumentace není tak přehledná jako u aplikace Toggl Track. Neposkytuje nativní integrace s jinými aplikacemi. Pro propojení lze ale využít služby třetích stran jako například Zapier.

Android aplikace

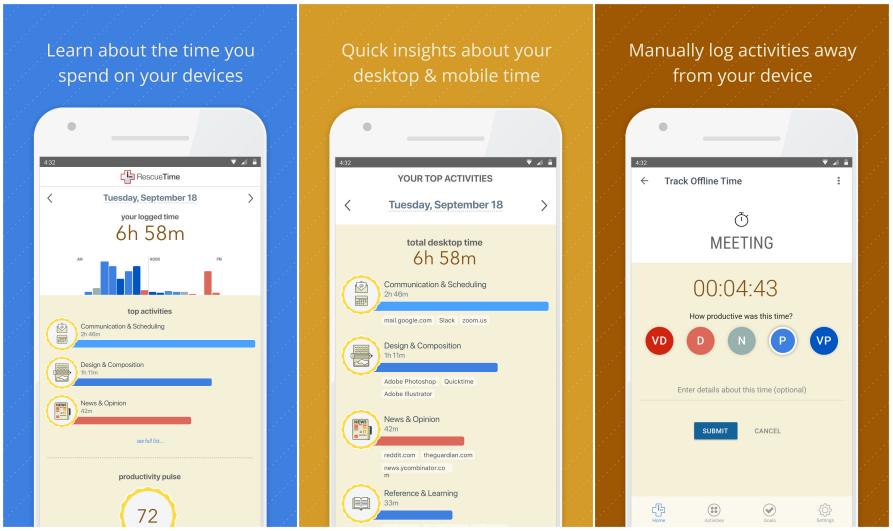
Aplikace Rescue Time je k dispozici i pro systém Android. Po prvním spuštění je nutné povolit sledování veškeré aktivity na zařízení. Následně se provede registrace pomocí emailové adresy a hesla. Po registraci začne aplikace sledovat veškeré činnosti. Na obrázku 3.4 jsou k vidění tři obrazovky z Android aplikace.

Sledování a vykazování aktivity na telefonu

Sledování aktivity je hlavní funkcionalitou, kterou aplikace nabízí. Činnosti, které uživatel na zařízení provádí, se zaznamenávají a v časových intervalech se synchronizují. Záznamy je možné zobrazit přímo v aplikaci nebo podrobněji ve webové aplikaci.

Vykazování telefonních hovorů

Aplikace dokáže sledovat i činnost spojenou s telefonováním. Pokud uživatel zůstane v telefonní aplikaci, ve které hovor probíhá, vytvoří se záznam o volání. Pokud ale uživatel nechá hovor probíhat na pozadí nebo otevře jinou aplikaci, vytvoří se pouze krátký časový záznam, který reprezentuje dobu, kdy uživatel hovor přijímal a měl telefonní aplikaci otevřenou. Telefonní aplikaci může uživatel otevírat i bez probíhajícího hovoru, v tomto případě se vytvoří záznam jako kdyby telefonoval. Tento způsob vykazování není kvůli výše popsaným problémům spolehlivý.



Obrázek 3.4 Snímky obrazovky z Android aplikace RescueTime Classic. Získáno z [14].

Navíc výkaz postrádá jakékoli další informace. Není k dispozici telefonní číslo ani případné jméno kontaktu. Nelze ani přidat žádný popis nebo výkaz kategorizovat. Často ani není známý přesný čas začátku hovoru, protože aplikace přiřazuje činnosti pouze k celým hodinám.

3.2.5 Shrnutí

Na trhu existuje mnoho aplikací na vykazování času. Řada z nich se soustředí především na činnosti prováděné na počítačích. Nástroje sledují činnosti provedené na počítači a záznamy poté přenáší do mobilních aplikací, kde je možné je zobrazovat a upravovat.

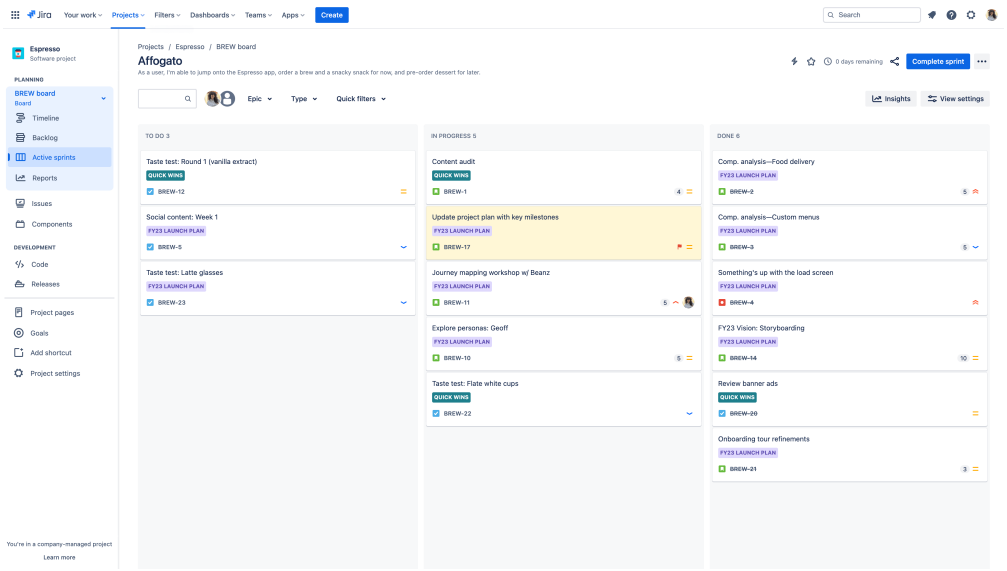
Příkladem takové aplikace je Timely. Dokáže sledovat činnosti provedené na počítačích s různými operačními systémy a tyto činnosti potom přehledně zobrazuje v mobilní aplikaci. Nedokáže ale sledovat činnosti provedené na mobilním zařízení. V předchozích verzích aplikace podle dostupných informací dokázala detekovat a vykazovat činnosti spojené s telefonováním, ale tato funkcionalita byla z aplikace odstraněna.

Některé aplikace se pokouší i o sledování činností prováděných na mobilních zařízeních. Například aplikace Timing pro macOS dokáže importovat čas strávený na iPhonech a iPadech. Toto importování je ale velmi nespolehlivé a často trvá i několik hodin než se data z telefonu objeví v aplikaci. Inzerovanou schopnost sledovat a vykazovat telefonní hovory se nepodařilo zprovoznit.

Android aplikace RescueTime dokáže vcelku úspěšně zaznamenávat činnosti provedené na Android zařízení. Bohužel následný výpis není nejlépe vyřešený a je pro uživatele nepřehledný. Mobilní aplikace postrádá detailnější popis vykázané činnosti, který je ale dostupný ve webové verzi aplikace. RescueTime detekuje, že měl uživatel otevřenou telefonní aplikaci, nedokáže ale zobrazit přesný čas, telefonní číslo ani žádné další informace týkající se hovoru. Tento způsob vykazování telefonních hovorů není spolehlivý.

Aplikace Toggl Track je jedna z nejpoužívanějších aplikací svého druhu. Dokáže spolehlivě vykazovat manuálně spuštěné činnosti a nabízí propracovaný systém projektů a značek, pomocí kterých lze výkazy kategorizovat. Desktopové aplikace dokáží sledovat aktivitu provedenou na počítači. Android aplikace nedisponuje žádnou funkcí, která by umožnila jednoduše vykazovat telefonní hovory.

Provedená analýza podporuje názor, že v současné době neexistuje jednoduše vyhledatelná a dostupná aplikace, která by dokázala automaticky detekovat a vykazovat činnosti spojené s



Obrázek 3.5 Snímek z aplikace Jira, na kterém je vidět projekt s několika požadavky v různých stavech. Získáno z [17].

telefonováním. Existující aplikace často umožňují manuální vytváření těchto výkazů, ale automatičnost ve všech zmíněných aplikacích chybí. Z tohoto důvodu má smysl pokusit se o vývoj vlastní aplikace, která svým uživatelům chybějící funkcionality nabídne.

3.3 Issue tracking systémy

Issue tracking systémy reprezentují velkou skupinu softwarových aplikací, které se při vývoji softwaru využívají pro správu projektů a požadavků. Aplikace většinou umožňují vytvářet a spravovat projekty, ke kterým lze přiřazovat jednotlivé požadavky neboli úkoly (issues). Obvykle je aplikace současně používána celým týmem vývojářů, kteří společně pracují na splnění všech požadavků a dokončení projektů. Níže představím dva velmi používané nástroje – Jira Software a Redmine.

3.3.1 Jira Software

Jira Software je jeden z nejpopulárnějších nástrojů pro správu a řízení projektů. Produkt byl vyvinutý společností Atlassian a je určen především pro týmy využívající metody agilního vývoje[1]. Používá se k efektivnímu plánování, sledování, vydávání i následné správě softwarových produktů. [15]

Jedná se o mnohostrannou aplikace, která nabízí nástroje pro správu agilního workflow, sledování softwarových chyb, plánování projektů, spolupráci v týmu či reporting a analýzu.

Hlavní entitou v aplikaci Jira je projekt. Projekt se definuje jako soubor úkolů *(issues)*, které je potřeba dokončit pro získání určitého výsledku. K projektu se přidávají lidé, kteří společně řeší úkoly tvořící daný projekt. Ke každému úkolu lze vykazovat strávený čas. [16]

Na obrázku 3.5 lze vidět snímek z aplikace, který ukazuje stránku s projektem a několika požadavky, které se nacházejí v různých stavech.

[1] Agilní vývoj je skupina metod pro vývoj softwaru, které umožňují rychlý vývoj a zároveň rychlé reagování na změny požadavků v průběhu vývoje.

3.3.2 Redmine

Redmine je open-source [2] a bezplatná webová aplikace pro řízení projektů a správu požadavků. Aplikace nabízí velkou flexibilitu konfigurace a podporuje až 49 světových jazyků [18, 19].

Mezi hlavní funkcionality patří:

- Správa projektů
- Správa požadavků
- Vykazování času
- Projektové wiki stránky a fóra

3.4 Analýza potřeb uživatelů

Analýza potřeb uživatelů je klíčová pro pochopení jejich požadavků a očekávání od aplikace. Na jejím základě se specifikují softwarové požadavky, navrhují se funkcionality a následně probíhá vývoj. V tomto případě byla provedena analýza mezi cílovou skupinou uživatelů, která pomohla definovat klíčové potřeby, které by aplikace měla splňovat.

3.4.1 Cílová skupina

Pro efektivní analýzu potřeb je nezbytné jasně definovat cílovou skupinu, pro kterou je aplikace určena. V rámci této práce se jedná o uživatele, kteří splňují následující vlastnosti:

- Používají zařízení s operačním systémem Android.
- Používají nástroje pro správu projektů.
- Evidují odpracovaný čas do nástrojů pro správu projektů.
- V rámci pracovní činnosti komunikují s kolegy a klienty pomocí mobilního telefonu.

Komunikace jako součást pracovní činnost spolu s vykazováním odpracovaného času implikují, že uživatelé čas strávený telefonováním vykazují stejně jako jiné pracovní činnosti. V současnosti je manuální zadávání telefonních hovorů do nástrojů pro správu projektů zdlouhavé a neefektivní.

3.4.2 Potřeby uživatelů

Analýza zahrnovala kvalitativní výzkum s malou skupinou uživatelů. S uživatel byly vedeny rozhovory, které poskytly detailní a relevantní informace o jejich potřebách. Na základě výzkumu byly zjištěny tyto klíčové požadavky:

- **Automatické vykázání telefonního hovoru**: Uživatelé očekávají, že aplikace automaticky vykáže proběhlý telefonní hovor. Po ukončení hovoru by se v nástroji pro správu projektů měl automaticky objevit časový výkaz reprezentující ukončený telefonní hovor.
- **Automatické přiřazení projektu a úkolů**: V nástrojích pro správu projektů se časové výkazy obvykle přiřazují ke konkrétnímu projektu a úkolu. Uživatelé požadují, aby tato přiřazení aplikace prováděla automaticky. Po automatickém vykázaní hovoru by se k němu měl automaticky přiřadit odpovídající projekt a úkol.

[2] Open-source je označení pro program nebo systém, jehož zdrojový kód je veřejně přístupný.

- **Možnost úpravy probíhajícího výkazu hovoru**: Uživatelé by měli mít možnost upravovat výkaz hovoru i během jeho průběhu. Představují si, že po přijetí hovoru na mobilním telefonu aplikace spustí automatické vykazování a uživatel bude moci k hovoru manuálně vybrat projekt i úkol. Po skončení hovoru se v nástroji pro správu projektů bude nacházet výkaz s vlastnostmi, které uživatel během jeho průběhu nastavil.

- **Úprava vykázaných hovorů**: Uživatelé by měli mít možnost v aplikaci upravovat již vykázané hovory. To by jim umožnilo například změnit přiřazený projekt nebo úkol, pokud by se po ukončení automaticky vykázaného hovoru ukázalo, že původní přiřazení nebylo správné.

Tento seznam představuje základní souhrn požadavků uživatelů a slouží jako podklad pro další kroky v procesu vývoje aplikace, včetně specifikace softwarových požadavků a definování případů užití.

3.5 Softwarové požadavky

V této sekci budou popsány všechny softwarové požadavky kladené na vyvíjenou aplikaci. Požadavky vycházejí z průzkumu provedeného mezi potenciálními uživateli a diskuzí s vedoucím práce. Do tvorby požadavků se také promítla analýza existujících řešení, která pomohla identifikovat nedostatky současných aplikací.

3.5.1 Definice softwarového požadavku

Při zahájení vývoje softwarového produktu je klíčové definovat jeho požadavky. Softwarový požadavek je specifikace chování, vlastností nebo funkce, kterou musí softwarový systém splňovat, aby byl uspokojivý pro uživatele nebo zákazníka. Správně vytvořené požadavky slouží zákazníkům jako komunikační kanál pro vyjádření toho, co přesně od systému požadují. Pro softwarové vývojáře jsou pak klíčové pro přesné pochopení toho, co zákazník očekává. [20, 21]

Na základě toho, jakou část softwaru požadavek popisuje, rozlišujeme dva základní druhy požadavků: [22]

funkční požadavky - popisují funkce a chování softwaru

nefunkční požadavky - popisují vlastnosti a kvalitu softwaru

3.5.2 Metoda FURPS a FURPS+

Někdy bývá obtížné odlišit funkční požadavky od těch nefunkčních. Z tohoto důvodu byla vytvořena metoda FURPS, která pomáhá najít hranici mezi funkcemi a vlastnostmi softwaru. Tato metoda se dnes často používá pro popis softwarových požadavků. [23, 24]

Název FURPS je zkratkou pěti slov:

- **Functionality**: Funkční požadavky podle definice výše.

- **Usability**: Nefunkční požadavky týkající se použitelnosti aplikace a uživatelského rozhraní. Patří sem požadavky na konzistenci a vzhled aplikace a dále také definice speciálních potřeb cílových uživatelů.

- **Reliability**: Nefunkční požadavky týkající se spolehlivosti. Patří sem požadavky definující pravděpodobnost výpadků, dostupnost systému nebo přesnost systému.

- **Performance**: Nefunkční požadavky týkající se rychlosti a efektivity fungování softwaru. Tato skupina zahrnuje faktory, jako je propustnost, rychlost zpracování, doba odezvy nebo doba zapnutí a vypínaní.

- **Supportability**: Nefunkční požadavky týkající se podporovatelnosti. Do této skupiny patří požadavky definující vlastnosti jako je testovatelnost, udržovatelnost, kompatibilita, konfigurovatelnost a lokalizovatelnost.

V průběhu používání FURPS se objevily požadavky, které nepatřily ani do jedné z existujících kategorií. Proto vzniklo rozšíření FURPS+, které k původním pěti kategoriím přidává ještě další čtyři:

- **Návrhový požadavek**: Požadavky, které specifikují nebo omezují možnosti návrhu systému. Jedná se například o požadavek specifikující nutnost použití relační databáze.

- **Implementační požadavek**: Požadavek specifikující nebo omezující implementační detaily produktu. Mohou zahrnovat požadované jazyky nebo standardy.

- **Požadavek na rozhraní**: Požadavky definující externí služby, se kterými musí systém interagovat, a také omezení formátů používaných při interakci.

- **Fyzický požadavek**: Požadavky specifikující omezení kladené na hardware.

Jak je z popisu patrné, každá skupina požadavků reprezentuje určitou část softwarového produktu. Metoda zaručuje, že pro každou část jsou definované potřebné vlastnosti. Při tvorbě požadavků je každý z nich přiřazen právě do jedné skupiny. [23, 24]

3.5.3 Metoda MoSCoW

MoSCoW je metoda pro určování priority požadavků nejen v oblasti vývoje softwaru. Název MoSCoW je zkratkou z anglických slov, které přímo odrážejí prioritu daného požadavku: [25]

- **Must Have (Musí mít)** - Základní požadavek, který je nutné splnit a bez kterého produkt nebude fungovat. Bez tohoto požadavku nebude mít produkt pro uživatele žádnou hodnotu.

- **Should Have (Měl by mít)** - Důležitý požadavek, který výrazně zvyšuje použitelnost a užitečnost produktu pro uživatele.

- **Could Have (Může mít)** - Požadavek, který pro produkt není klíčový, ale může zvýšit jeho atraktivitu pro uživatele.

- **Won't Have This Time (Tentokrát nebude mít)** - Požadavek, který pro aktuální verzi není důležitý. Tyto požadavky mohou být zohledněny v budoucích verzích produktu.

3.5.4 Funkční požadavky

Na základě rozhovorů s potenciálními uživateli aplikace a vedoucím práce vzešlo několik funkčních požadavků. Níže tyto funkční požadavky představím. Pro přehlednost a správné pracování s požadavky v dalších částech práce bude každý funkční požadavek respektovat určitou strukturu, kterou popíšu níže.

Struktura požadavků

Každý funkční požadavek se bude skládat z následujících částí:

- **kód** - umožní snadnou identifikaci požadavku při ověřování jeho splnění v závěrečných fázích vývoje
- **název** - výstižné shrnutí požadavku
- **popis** - podrobný popis požadavku (popis musí být jednoznačný, aby mohl být požadavek ověřitelný)
- **priorita** - priorita požadavku podle metody MoSCoW

Jelikož jsou všechny požadavky v této části funkční, nemá smysl psát ke každému požadavku jeho FURPS+ skupinu, která je u všech požadavků stejná – funkční (Functionality). Z tohoto důvodu není ve struktuře požadavku pole pro typ požadavku.

Seznam požadavků

Vzhledem k náročnosti některých požadavků jsou tyto požadavky označeny prioritou „Won't have" a tedy nebudou v prvotní verzi aplikace implementovány. Splnění těchto požadavků by bylo nad rámec této práce. Zůstanou zde ale zmíněny jako podklad pro budoucí vývoj.

FR1 – Zobrazení výpisu telefonních hovorů

Aplikace uživateli přehledně zobrazí všechny telefonní hovory provedené jeho telefonem. Nepřijaté hovory nebudou zobrazeny, jelikož nemají žádný trvající čas, a tedy není vhodné je zahrnovat do výpisu. Aplikace rovněž nahradí telefonní čísla jmény kontaktů, pokud jsou tyto kontakty uloženy v telefonu.

Priorita: **Must Have**

FR2 – Filtrování výpisu telefonních hovorů

Uživatel bude mít možnost filtrovat zobrazený výpis telefonních hovorů podle telefonního čísla, jména kontaktu a typu hovoru (odchozí, příchozí).

Priorita: **Should Have**

FR3 – Vykázání konkrétního zvoleného telefonního hovoru

Uživatel bude schopen vybrat konkrétní hovor z výpisu a snadno ho vykázat jako časový záznam. Výkaz bude mít přidělený projekt a úkoly na základě historie výkazů hovorů s daným kontaktem, pokud taková historie existuje. Uživatel bude mít možnost tyto parametry upravit.

Priorita: **Must Have**

FR4 – Spuštění časového záznamu při zahájení telefonního hovoru

Aplikace automaticky spustí časovač při zahájení hovoru. Pokud časovač v době zahájení hovoru již běží v rámci jiného výkazu, bude tento původní výkaz pozastaven a spuštěn nový pro aktuální hovor.

Priorita: **Must Have**

FR5 – Ukončení časového záznamu při ukončení telefonního hovoru

Aplikace při skončení hovoru zastaví časovač spuštěný při zahájení hovoru. Vytvoří výkaz, který bude nastaven podle historie výkazů hovorů s daným kontaktem. Pokud není možné tyto informace přiřadit, bude výkaz zařazen do projektu, který si uživatel předem zvolil.

Výkazy vytvořené aplikací budou označeny speciální značkou pro snadnější filtrování. Telefonní číslo bude uloženo v značce místo popisu výkazu, aby bylo zabráněno nežádoucímu sdílení telefonních čísel.

Priorita: **Must Have**

FR6 – Podpora jiných než telefonních typů hovorů

Aplikace bude schopna pracovat i s hovory jiných typů, zejména internetovými hovory provedenými z externích aplikací.

Priorita: **Won't Have**

FR7 – Zobrazení časových výkazů vytvořených aplikací

Aplikace přehledně zobrazí seznam všech vykázaných hovorů, které byly v aplikaci vykázány.

Priorita: **Must Have**

FR8 – Filtrování časových výkazů z výpisu

Uživatel bude mít možnost jednoduchého filtrování zobrazených výkazů hovorů na základě telefonního čísla nebo jména kontaktu.

Priorita: **Should Have**

FR9 – Úprava časových výkazů z výpisu

Uživatel bude moci vybrat konkrétní časový výkaz a provést jeho úpravu. Bude moci změnit popis, přiřadit výkaz k jinému projektu nebo upravit značky. Nebude možné změnit značku určující, že výkaz vytvořila aplikace, a značku reprezentující telefonní číslo.

Priorita: **Must Have**

FR10 – Propojení s Toggl Track pomocí API klíče

Aplikace bude vyžadovat propojení s aplikací Toggl Track. Jednou z možností propojení je pomocí API klíče, který je přiřazen ke každému Toggl Track účtu. Uživatel bude moci tento klíč k propojení použít.

Priorita: **Must Have**

FR11 – Propojení s Toggl Track pomocí přihlašovacích údajů

Další možností propojení s Toggl Track bude pomocí přihlašovacích údajů. Uživatel bude moci k propojení využít své přihlašovací údaje.

Priorita: **Won't Have**

FR12 – Odpojení nebo změna Toggl Track účtu

Aplikace umožní změnit nebo odstranit přihlášený Toggl Track účet.

Priorita: **Must Have**

FR13 – Konfigurace místa uložení pro hovory, které nelze automaticky přiřadit

V aplikaci bude možné nastavit Toggl Track projekt, pod který budou zařazeny výkazy, u kterých nelze určit projekt na základě historie.

Priorita: **Must Have**

FR14 – Automatické přiřazení hovoru k projektu a úkolům

Aplikace bude projekt a značky k vykázaným hovorům přiřazovat automaticky podle historie výkazů. Nový hovor se stejnou osobou bude vykázán pod projekt a se značkami, které jsou nastaveny u posledního výkazu hovoru se stejnou osobou. Pokud se jedná o první vykázaný hovor, bude hovor vykázán pod předem nastavený výchozí projekt.

Priorita: **Must Have**

FR15 – Zapamatování zvolené konfigurace napříč zařízeními

Všechna uživatelská nastavení budou přenosná napříč zařízeními pomocí Google účtu. Pokud uživatel nainstaluje aplikaci na jiném zařízení, které je přihlášeno pod stejný Google účet, aplikace se sama nastaví podle nastavení již nainstalované aplikace.

Priorita: **Won't Have**

FR16 – Rozdělení probíhajícího hovoru na více částí

Při jediném hovoru se může řešit více projektů a úkolů. Proto bude aplikace umožňovat rozdělit probíhající hovor na více částí. Po kliknutí na příslušné tlačítko se ukončí časovač a okamžitě se spustí nový. Všechny takto rozdělené výkazy bude uživatel moci následně přiřadit k projektům a úkolům podle svého uvážení.

Priorita: **Won't Have**

FR17 – Pozastavení a spuštění automatického vykazování hovorů

V nastavení aplikace bude možné pozastavit a opět spustit automatické spouštění časovače výkazu při detekci zahájení hovoru.

Priorita: **Must Have**

3.5.5 Nefunkční požadavky

Nefunkční požadavky aplikace popisují, jak by měl software fungovat, na rozdíl od funkčních požadavků, které říkají, co by měl software dělat. Zaměřují se na kvalitu a chování softwaru. U nefunkčních požadavků bude využita dříve popsaná metoda FURPS+, která nefunkční požadavky rozděluje podle vlastností do dalších dílčích skupin. Každý nefunkční požadavek bude označen příslušnou skupinou.

Seznam nefunkčních požadavků

Níže je uveden seznam nefunkčních požadavků aplikace. Při jejich popisu je použita stejná struktura, která byla použita při definici funkčních požadavků – tedy kód, název, popis a priorita podle metody MoSCoW. Tato struktura je navíc rozšířena o typ požadavků podle metody FURPS+.

NFR1 – Jazyk aplikace

Aplikace bude dostupná v angličtině.

Priorita: **Must Have**

Typ: **Supportability**

NFR2 – Programovací jazyk

Aplikace bude implementována v jazyce, který je vhodný pro programování Android aplikací.

Priorita: **Must Have**

Typ: **Implementační požadavek**

NFR3 – Minimální verze Androidu

Aplikace bude kompatibilní s operačním systémem Android s verzí 12 a vyšší.

Priorita: **Must Have**

Typ: **Supportability**

NFR4 – Integrace s aplikací Toggl Track

Aplikace bude spolupracovat s aplikací Toggl Track. K integraci bude použito veřejné Toggl Track API.

Priorita: **Must Have**

Typ: **Požadavek na rozhraní**

NFR5 – Přidání dalšího jazyka

Rozšíření o další jazyk nebude implementačně složité.

Priorita: **Should Have**

Typ: **Supportability**

NFR6 – Jednoduché a přehledné UI

Aplikace bude mít jednoduché UI, které přispěje k tomu, aby uživatel nemusel v aplikaci strávit mnoho času.

Priorita: **Must Have**

Typ: **Usability**

NFR7 – Konzistentní UI

Vzhled a barvy UI budou konzistentní napříč celou aplikací.

Priorita: **Must Have**

Typ: **Usability**

NFR8 – Typické UI komponenty

Použité UI komponenty budou vzhledem podobné těm, které se běžně používají v jiných Android aplikacích.

Priorita: **Must Have**

Typ: **Usability**

NFR9 – Předcházení pádům

Aplikace bude uživatele upozorňovat na vzniklé chyby. Bude předcházet nečekaným pádům.

Priorita: **Must Have**

Typ: **Reliability**

3.6 Případy užití

Případy užití (use case) popisují chování softwaru z pohledu uživatele. Jedná se o popis sledu interakcí mezi softwarem a jeho uživateli. Každý případ užití je spojen s nějakým cílem, který se vztahuje k jednomu určitému aktérovi (hlavní aktér). Každý možný sled interakcí se označuje jako scénář. Scénáře mohou skončit úspěchem, kdy je cíl splněn, nebo neúspěchem. Případy užití jsou spojeny s jedním nebo více funkčními požadavky. Jednotlivé kroky ve scénáři totiž vyžadují, aby systém tyto kroky dokázal vykonat, což právě zaručují specifikované funkční požadavky. [26]

3.6.1 Seznam případu užití

Níže je představen seznam vytvořených případů užití. Hlavními zdroji při jejich tvorbě byly funkční požadavky a rozhovory s potenciálními zákazníky i vedoucím práce. Na těchto případech je možné vytvořit si představu o budoucím používání vyvíjené aplikace.

UC1 - Manuální vykázání zvoleného hovoru

V aplikaci lze vytvořit časový výkaz na základě telefonního hovoru.

Hlavní aktér: Uživatel

Hlavní scénář:

1. Uživatel přejde na obrazovku s hovory.
2. Zvolí konkrétní hovor.
3. Aplikace nastaví parametry podle historie vykazování.
4. Zobrazí se podoba výkazu.
5. Uživatel s rekapitulací souhlasí.
6. Aplikace vykáže zvolený hovor.

Vedlejší scénář: Uživatel se po zhlédnutí rekapitulace rozhodne změnit parametry výkazu.

1. Uživatel přejde na obrazovku s hovory.
2. Zvolí konkrétní hovor.
3. Aplikace nastaví parametry podle historie vykazování.
4. Zobrazí se podoba výkazu.
5. Uživatel vybere parametr, který chce změnit.
6. Uživatel změní parametr.
7. Potvrzením se hovor vykáže.

Vedlejší scénář: Uživatel se po zhlédnutí rekapitulace rozhodne zrušit vykazování.

1. Uživatel přejde na obrazovku s hovory.
2. Uživatel na obrazovce s hovory zvolí konkrétní hovor.
3. Aplikace nastaví parametry podle historie vykazování.
4. Zobrazí se podoba výkazu.
5. Uživatel s rekapitulací nesouhlasí.
6. Klikne na tlačítko zpět, čímž se vrátí na obrazovku s hovory.

UC2 - Filtrování seznamu hovorů

Seznam hovorů lze filtrovat podle čísla, jména a typu hovoru.

Hlavní aktér: Uživatel

Hlavní scénář:

1. Uživatel přejde na obrazovku s hovory.
2. Aplikace zobrazí seznam hovorů.
3. Uživatel nastaví požadované filtrování.
4. Aplikace upraví zobrazený seznam.

UC3 - Automatické vykázání hovoru

Aplikace sama vytvoří časový výkaz proběhnutého telefonního hovoru.

Hlavní aktér: Aplikace

Hlavní scénář:

1. Zahájí se hovor.
2. Aplikace spustí časovač.
3. Hovor je ukončen.
4. Aplikace ukončí časovač.
5. Aplikace na základě časovače a historie hovorů vytvoří časový výkaz.

UC4 - Zobrazení výkazů telefonních hovorů

Aplikace zobrazí přehledný seznam výkazů telefonních hovorů.

Hlavní aktér: Uživatel

Hlavní scénář:

1. Uživatel přejde na stránku s výkazy.
2. Aplikace zobrazí všechny výkazy.

UC5 - Filtrování výkazů telefonních hovorů

Seznam výkazů telefonních hovorů lze filtrovat podle jména kontaktu a telefonního čísla.

Hlavní aktér: Uživatel

Hlavní scénář:

1. Uživatel přejde na stránku s výkazy.
2. Aplikace zobrazí všechny výkazy.
3. Uživatel nastaví hledaný text.
4. Aplikace zobrazí pouze hledané hovory.

UC6 - Úprava výkazu telefonního hovoru

Výkazy telefonních hovorů je možné v aplikaci upravovat.

Hlavní aktér: Uživatel

Hlavní scénář:

1. Uživatel vybere výkaz ze seznamu.
2. Změní požadované vlastnosti.
3. Potvrdí svoje změny.
4. Systém změny uloží.
5. Uživatel se vrátí na seznam výkazů.

Vedlejší scénář: Uživatel nepotvrdí svoje rozhodnutí.

1. Uživatel klikne na zvolený výkaz ze seznamu.
2. Změní požadované vlastnosti.
3. Uživatel se rozhodne nepotvrdit změny.
4. Klikne na tlačítko zpět.
5. Uživatel je vrácen na seznam výkazů.

UC7 - Vypnutí automatického vykazování

Uživatel může v aplikaci zapínat a vypínat automatické vykazování hovorů.

Hlavní aktér: Uživatel

Hlavní scénář:

1. Uživatel přejde do nastavení aplikace.
2. Stiskne příslušné tlačítko.
3. Aplikace na základě předchozího stavu spustí nebo vypne automatické vykazování.

UC8 - Přihlášení

Uživatel aplikaci propojí se svým Toggl Track účtem.

Hlavní aktér: Uživatel

Hlavní scénář: Zadané údaje jsou funkční.

1. Uživatel zadá správné přihlašovací údaje.
2. Aplikace otestuje funkčnost.
3. Aplikace údaje uloží.
4. Uživatel pokračuje na další stránku.

Vedlejší scénář: Zadané údaje jsou nefunkční.

1. Uživatel přejde na stránku přihlášení.
2. Zadá špatné přihlašovací údaje.
3. Aplikace otestuje funkčnost.
4. Aplikace uživatele upozorní na nefunkčnost.
5. Uživatel pokračuje od druhého kroku.

UC9 - Odhlášení

Uživatel zruší propojení aplikace se svým Toggl Track účtem.

Hlavní aktér: Uživatel

Hlavní scénář:

1. Uživatel přejde na stránku nastavení.
2. Stiskne tlačítko pro odhlášení.
3. Aplikace uživatele odhlásí.
4. Uživatel je přesměrován na úvodní přihlašovací stránku.

UC10 - Změna výchozího projektu

Uživatel změní výchozí projekt, pod který se vykazují hovory, které nelze přiřadit na základě historie vykazování.

Hlavní aktér: Uživatel

Hlavní scénář:

1. Uživatel přejde na stránku nastavení.
2. Vybere pole pro změnu projektu.
3. Aplikace nabídne možnosti.
4. Uživatel vybere jednu z možností.

5. Aplikace uloží změnu.

Vedlejší scénář: Uživatel chce vytvořit zcela nový projekt.

1. Uživatel přejde na stránku nastavení.
2. Vybere tlačítko pro vytvoření nového projektu.
3. Aplikace otevře okno pro zadání názvu.
4. Uživatel zvolí název.
5. Potvrdí svou volbu.
6. Aplikace vytvoří nový projekt a uloží ho jako výchozí.

UC11 - Rozdělení probíhajícího projektu

Uživatel může výkaz probíhajícího hovoru rozdělit na dílčí výkazy.

Hlavní aktér: Uživatel

Hlavní scénář:

1. Uživatel během probíhajícího hovoru otevře aplikaci.
2. Přejde na speciální stránku.
3. Aplikace uživatele upozorní, že probíhá výkaz hovoru.
4. Uživatel jedním tlačítkem ukončí původní výkaz a spustí nový.
5. Pozastavený výkaz bude moci upravit dle svých představ.

UC12 - Instalace na dalším zařízení

Aplikace se sama nakonfiguruje při instalaci na dalším uživatelově zařízení.

Hlavní aktér: Aplikace

Hlavní scénář:

1. Uživatel nainstaluje aplikaci na svém dalším zařízení.
2. Aplikace získá data z Google účtu.
3. Provede nastavení parametrů podle získaných dat.

3.6.2 Tabulka pokrytí

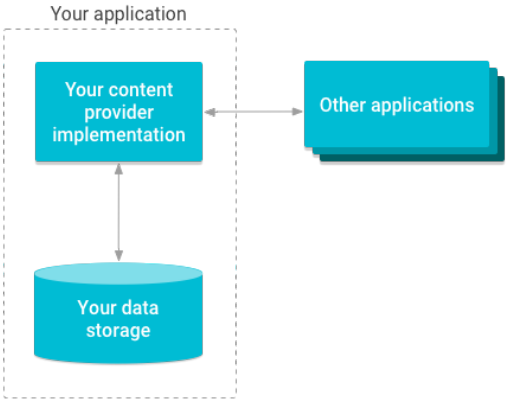
Případy užití jsou obvykle doprovázeny tabulkou pokrytí požadavků. V této tabulce je pro každý případ užití vidět seznam funkčních požadavků, které jsou jím pokryty. Sloupce tabulky odpovídají případům použití. Řádky potom jednotlivým funkčním požadavkům. Pokrytí je v tabulce vyznačeno symbolem v buňce. Pokud se vyznačený symbol neobjeví v sloupci tabulky, znamená to, že případ užití nepokrývá žádný funkční požadavek. Takový případ užití je buď zbytečný, nebo chybí dodefinovat další funkční požadavky. Naopak, pokud chybí symbol v řádku, znamená to, že požadavek nevyužívá žádný případ užití. Příklad tabulky vytvořené na základě požadavků a případů užití vyvíjené aplikace je vidět v tabulce 3.1.

3.7 Android a telefonní hovory

Podle popsaných požadavků bude nutné, aby aplikace pracovala s telefonními hovory. Přinejmenším bude potřeba číst historii hovorů a také detekovat zahájení a ukončení telefonního hovoru. Jelikož bývá telefonování nativně implementováno v systému, je možné, že zasahování do chování bude systémem omezeno. V této části budou analyzovány programovací možnosti, pomocí kterých lze s telefonními hovory v Android zařízeních pracovat.

	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	UC10	UC11	UC12
FR1	✓	✓										
FR2		✓										
FR3	✓											
FR4			✓									
FR5			✓									
FR6	✓	✓	✓									
FR7				✓								
FR8					✓							
FR9						✓						
FR10								✓				
FR11								✓				
FR12									✓			
FR13										✓		
FR14	✓		✓									
FR15												✓
FR16											✓	
FR17							✓					

Tabulka 3.1 Tabulka pokrytí případů užití jejich funkčními požadavky.



Obrázek 3.6 Diagram znázorňující použití vlastní implementace *Content provider*u. Získáno z webu [27].

3.7.1 Historie hovorů

Pro splnění požadavků **FR1**, **FR2** a **FR3** je potřeba získat historii telefonních hovorů. V Androidu lze k historii telefonování přistupovat pomocí tzv. Content provideru.

Content providers

Content providers reprezentují v Androidu třídy, které slouží ke dvěma účelům. Jednak umožňují poskytovat přístup k datům uloženým vlastní aplikací a jednak lze pomocí nich přistupovat k datům, které uložily jiné aplikace. Díky těmto třídám lze mezi aplikacemi data jednoduše sdílet.

První účel se používá, pokud požadujeme, aby aplikace sdílela svá data mezi jinými aplikacemi. Diagram znázorňující toto použití je vidět na obrázku 3.6. Vyvíjená aplikace tuto funkcionalitu potřebovat nebude. Proto se implementací vlastních Content providers dále nebudu zabývat. [27]

Pro náš účel využijeme fakt, že v Android frameworku existuje několik již implementovaných Content providers. Ty se používají pro přístup například k audio a video souborům, obrázkům a

nebo právě ke kontaktům a výpisu telefonní historie. Ve vyvíjené aplikaci bude Content provider použit právě pro přístup k výpisu telefonních hovorů.

CallLog

CallLog je Content provider dostupný v Android frameworku. CallLog obsahuje informace o provedených telefonních hovorech [28]. Příklad použití je k vidění na ukázce kódu 3.1.

```
fun getCalls(): List<SystemCall> {

    val callLogs = mutableListOf<SystemCall>()

    val projection = arrayOf(
        CallLog.Calls.CACHED_NAME,
        CallLog.Calls.NUMBER,
        CallLog.Calls.DURATION,
        CallLog.Calls.DATE
    )

    val cursor = context.contentResolver.query(
        CallLog.Calls.CONTENT_URI,
        projection,
        null,
        null,
        "${CallLog.Calls.DATE} DESC"
    )

    cursor?.use {
        while (cursor.moveToNext()) {
            val name = cursor.getString(0)
            val number = cursor.getString(1)
            val duration = cursor.getString(2)
            val date = cursor.getString(3)
            callLogs.add(SystemCall(name, number, duration, date))
        }
    }
    return callLogs
}
```

Výpis kódu 3.1 Ukázka použití CallLog pro získání celého výpisu telefonních hovorů.

3.7.2 Detekce zahájení a ukončení hovoru

Pro splnění požadavků **FR4** a **FR5** je třeba detekovat zahájení resp. ukončení hovoru. Pro tento účel se v Androidu používá třída TelephonyCallback.

Třída TelephonyCallback umožňuje v Androidu sledovat změny v telekomunikačním stavu zařízení, včetně stavu služby, síly signálu, indikátoru zmeškaných zpráv a dalších. Tuto třídu lze využít také pro sledování stavu telefonního spojení a díky tomu lze detekovat zahájení a ukončení hovoru. [29]

Vytvoření callbacku

Pro sledování telekomunikačního stavu zařízení je potřeba vytvořit vlastní implementaci třídy, která dědí z třídy TelephonyCallback. Dále musí nově vytvořená třída implementovat jeden nebo více listenerů. Ty jsou k dispozici jako vnořené rozhraní rodičovské třídy TelephonyCallback. Každý z nich sleduje odlišné vlastnosti telekomunikačního zařízení a u každého, jakožto u rozhraní, je potřeba implementovat jiné metody, které souvisí s jeho konkrétní činností.

CallStateListener

CallStateListener je jedním z možných listenerů, které se dají při implementaci TelephonyCallbacku využít. CallStateListener detekuje změny stavu telefonního spojení. U tohoto listeneru je potřeba implementovat pouze jedinou metodu onCallStateChanged, která má jako parametr číslo reprezentující stav telefonního spojení.

Android rozlišuje 3 stavy telefonního spojení:

- **TelephonyManager.CALL_STATE_IDLE** - Žádná aktivita.
- **TelephonyManager.CALL_STATE_OFFHOOK** - Alespoň jeden hovor je aktivní, právě vytáčen nebo přidržen.
- **TelephonyManager.CALL_STATE_RINGING** - Přišel nový hovor, který vyzvání nebo čeká.

Na výpisu kódu 3.2 je vidět ukázka použití CallStateListner při vytváření třídy CallEndedTelephonyCallback. Metoda onCallStateChanged se volá automaticky při každé změně stavu telefonního spojení. Jak je z ukázky patrné, při zjištění ukončení hovoru se zavolá metoda handleCallEnded. [29]

Registrace callbacku

Vytvořený callback objekt je potřeba následně zaregistrovat v TelephonyManageru pomocí metody registerTelephonyCallback. Metoda má dva parametry, první je typu Executor a reprezentuje Executor objekt, který bude při změně stavu vykonávat definovaný kód. Druhým parametrem je právě vlastní TelephonyCallback objekt. Příklad registrace vytvořené třídy CallEndedTelephonyCallback z textu výše je vidět na ukázce kódu 3.3.

Permissions

Každý z listenerů, jež vytvářený callback implementuje, vyžaduje různá oprávnění. Před registrací callbacku je nutné ověřit, zda uživatel aplikaci požadovaná oprávnění přidělil. Seznam požadovaných oprávnění je k dispozici v oficiální dokumentaci každého z listenerů. Například výše zmíněný CallStateListener vyžaduje oprávnění **READ PHONE STATE**. [30, 29].

3.7.3 Shrnutí

Z výše popsané analýzy vyplývá, že hlavní požadavky týkající se automatického vykazování hovorů (FR4, FR5) a požadavky ohledně seznamu hovorů (FR1, FR2, FR3) lze v Androidu splnit a implementovat.

3.8 Android oprávnění

Android oprávnění slouží v operačním systému Android jako mechanismus ochrany přístupu k systémovým zdrojům a datům. Poskytují kontrolu nad tím, co aplikace smí dělat a k jakým

```
class CallEndedTelephonyCallback()
: TelephonyCallback(), TelephonyCallback.CallStateListener {

    private var previousState: Int = -1

    override fun onCallStateChanged(state: Int) {
        when (state) {
            TelephonyManager.CALL_STATE_IDLE -> {
                if (previousState == TelephonyManager.CALL_STATE_OFFHOOK)
                    handleCallEnded()
                previousState = TelephonyManager.CALL_STATE_IDLE
            }

            TelephonyManager.CALL_STATE_OFFHOOK -> {
                previousState = TelephonyManager.CALL_STATE_OFFHOOK
            }

            TelephonyManager.CALL_STATE_RINGING -> {
                previousState = TelephonyManager.CALL_STATE_RINGING
            }
        }
    }
}
```

Výpis kódu 3.2 Ukázka použití třídy CallStateListner pro detekci změn stavu telefonního spojení.

informacím má přístup. Oprávnění pomáhají chránit uživatele před škodlivými aplikacemi a zajišťují, že aplikace nemohou zneužívat jejich data nebo zařízení.

3.8.1 Druhy oprávnění v Androidu

Existují dva druhy Android oprávnění:

- **Install-time oprávnění**: Tato oprávnění jsou udělována automaticky při instalaci aplikace. Uživatel je nemůže odmítnout. Programátor tyto oprávnění pouze vypisuje do speciálního souboru *AndroidManifest.xml*. Obvykle se používají pro základní funkce, které aplikace potřebuje k fungování, jako je přístup k internetu nebo úložišti.
- **Runtime oprávnění**: Tato oprávnění mohou představovat riziko pro soukromí nebo bezpečnost uživatele. Aplikace musí o jejich přidělení požádat až v době svého běhu. Je potřeba počítat s možností odmítnutí oprávnění, v takovém případě je vhodné omezeny některé funkcionality aplikace a umožnit uživateli pokračovat bez nich. Tato skupina oprávnění zahrnuje například přístup ke kontaktům, zprávám, fotoaparátu, mikrofonu a nebo poloze.

3.8.2 Oprávnění vyžadovaná aplikací

Z provedené analýzy vyplývá potřeba přistupovat k historii hovorů. K tomu je potřeba získat oprávnění **READ CALL LOG**. Dále je pro detekci příchozích a odchozích hovorů potřeba sledovat změny stavu telekomunikačního zařízení. K tomu je potřeba mít oprávnění **READ PHONE STATE**.

```
@HiltAndroidApp
class Application : Application() {

    private var telephonyManager: TelephonyManager? = null

    private var telephonyCallback: TelephonyCallback? = null

    private val executor: Executor = Executors.newSingleThreadExecutor()

    override fun onCreate() {
        super.onCreate()

        telephonyManager = getSystemService(Context.TELEPHONY_SERVICE)
                as TelephonyManager
        telephonyCallback = CallEndedTelephonyCallback()

        telephonyManager.registerTelephonyCallback(executor, telephonyCallback)
    }
}
```

Výpis kódu 3.3 Ukázka registrace CallEndedTelephonyCallback objektu v TelephonyManageru během spouštění aplikace

Obě tato oprávnění jsou runtime oprávnění. Proto bude potřeba vyžádat od uživatele jejich potvrzení až v běžící aplikaci. Je také potřeba počítat s tím, že uživatel může v nastavení aplikace oprávnění později odebrat. Proto je nutné navrhnout aplikaci tak, aby v omezeném režimu fungovala i po pozdějším odebrání a dokázala uživatele upozornit na chybějící oprávnění. Více bude o implementaci žádání o oprávnění rozepsáno v kapitole věnující se implementaci.

3.9 Aplikace Timer2Ticket

Timer2Ticket je webová aplikace, která automaticky provádí vzájemnou synchronizaci mezi nástroji pro správu projektů (např. Redmine) a aplikacemi na vykazování času (např. Toggl Track). Aplikace byla vyvinuta v rámci diplomové práce Vítem Štefanem [2].

Současná verze aplikace umožňuje synchronizaci mezi aplikacemi Toggl Track a Redmine. Použití s těmito dvěma aplikacemi je přímočaré:

1. Uživatel se do aplikace zaregistruje.
2. Zapíše Redmine API klíč a URL adresu, na které Redmine běží.
3. Vybere výchozí typ Redmine aktivity.
4. Zapíše Toggl API klíč.
5. Nastaví požadovanou frekvenci synchronizace.

Po těchto krocích je nástroj funkční a periodicky provádí synchronizaci mezi oběma výše zmíněnými aplikacemi.

3.9.1 Využití nástroje Timer2Ticket pro synchronizaci dat aplikace

Jedním z úkolů práce je analyzovat možnosti využití nástroje Timer2Ticket pro synchronizaci dat mezi vyvíjenou aplikací a nástroji pro správu projektů. Většina aplikací vytváří data a aplikace vyvíjená v rámci této práce není výjimkou. Bude vytvářet časové výkazy, které budou odpovídat provedeným telefonním hovorům. Zároveň aplikace bude nutně interagovat s nástroji pro správu projektů.

Protože data bude možné vytvářet jak v aplikaci, tak přímo v nástroji pro správu projektů, je nutné vyřešit jejich vzájemnou synchronizaci. K tomuto účelu lze využít existující nástroj Timer2Ticket. Níže popíšu hlavní problémy, které se s využitím nástroje pojí a na závěr rozhodnu o jeho využití v rámci vyvíjené aplikace.

Požadavky na vyvíjenou aplikaci

Služba, kterou chceme do Timer2Ticketu přidat, musí svá data nabízet přes veřejné API. Synchronizační job díky tomu data periodicky získává a následně s nimi pracuje. Pro přidání služby by tedy bylo nutné navrhnout aplikaci tak, aby se skládala ze serverové části (backend) a vlastní Android aplikace (frontend). Implementace takové architektury by zabrala příliš mnoho času.

Prodleva mezi synchronizací

Timer2Ticket pracuje na základě synchronizačních jobů, které jsou spouštěny v určitých časových intervalech. Jedním z požadavků na vyvíjenou aplikaci je možnost ručního upravení vykázaných časových záznamů v dalších synchronizovaných aplikacích. Pokud by se využil Timer2Ticket, mohl by uživatel po vykázání hovoru čekat nepříjemně dlouho, než proběhne synchronizace, aby mohl svoje časové záznamy upravit.

Náročnost

Jak bylo dříve zmíněno v sekci 3.9, v současnosti dokáže Timer2Ticket synchronizovat pouze mezi aplikací *Toggl Track* a nástrojem pro správu projektů *Redmine*. Pro přidání další služby, reprezentující aplikaci vyvíjenou v této práci, by bylo nutné provést zásah do kódu a ruční doprogramování, neboť Timer2Ticket nedisponuje API, které by umožnilo jednoduché přidaní další služby. S ohledem na autorovy téměř nulové znalosti jazyka TypeScript, ve kterém je Timer2Ticket naprogramován, by bylo doprogramování časově velmi náročné.

Shrnutí

I když je Timer2Ticket velice užitečný nástroj, z výše uvedeného popisu vyplývá, že jeho použití pro specifický případ vyvíjené aplikace není vhodným řešením. Z těchto důvodů nebude aplikace nástroj Timer2Ticket přímo používat pro synchronizaci svých dat.

Existují však možnosti, jak nástroj i přesto využít. Toggl Track, jehož data umí Timer2Ticket synchronizovat, disponuje veřejným a spolehlivým API, které umožňuje ukládat a spravovat data, včetně časových výkazů. Myšlenka spočívá v tom, že do Toggl Tracku budou z vyvíjené aplikace ukládány časové výkazy hovorů. Uživatel se pak může sám rozhodnout, zda svůj Toggl Track účet pomocí Timer2Ticketu s nástroji pro správu projektů propojí a využije tak vzájemnou synchronizaci, nebo ne. Více bude o této myšlence rozepsáno v sekci 4.7.

Kapitola 4
Návrh

V předchozí kapitole se autor zabýval prvními kroky při vývoji nového softwaru. Především specifikoval softwarové požadavky a případy užití. V této kapitole budou popsány možnosti, jakým způsobem lze požadované funkcionality aplikace implementovat. Budou představeny možné programovací jazyky a technologie. Dále bude ukázána architektura systému spolu se způsobem persistence dat. Bude představen návrh nejdůležitějšího procesu v aplikaci – proces automatického vykazování hovoru. Na závěr kapitoly budou popsány návrhy konkrétních obrazovek aplikace.

4.1 Programovací jazyk

Android aplikace může být napsána v jednom ze tří jazyků: Kotlin, Java nebo C++. Volba jazyka závisí na preferencích vývojáře, požadovaných vlastnostech aplikace a specifických požadavcích projektu.

Kód napsaný v jakémkoliv z nabízených programovacích jazyků spolu s dalšími daty a soubory projektu se pomocí Android Software Development Kit (SDK) zkompiluje do Android Package (soubor APK) nebo do Android App Bundle (soubor AAB). Android Package je archivní soubor s příponou *.apk*, který obsahuje vše, co Android aplikace potřebuje pro svůj běh. Jedná se o soubor, pomocí kterého se na Android zařízeních aplikace instalují. [31]

Níže představím dva nejpoužívanější jazyky pro vývoj Android aplikací – Java a Kotlin.

4.1.1 Java

Java je objektově orientovaný programovací jazyk, který byl prvním oficiálním jazykem pro vývoj Android aplikací. Je to robustní a stabilní jazyk s rozsáhlou komunitou a bohatou dokumentací. Java nabízí vysoký stupeň kontroly nad kódem a umožňuje vývojářům vytvářet výkonné a efektivní aplikace.

4.1.2 Kotlin

Kotlin je moderní staticky typovaný programovací jazyk vyvinutý společností JetBrains.

Je navržen tak, aby plně spolupracoval s Javou, díky čemuž lze při vývoji používat oba jazyky současně. Moderní vývojová prostředí často nabízejí automatickou konverzi mezi oběma jazyky. Tato vlastnost vývojářům usnadňuje přechod z Javy na Kotlin.

Jedná se o multiplatformní jazyk – usnadňuje vývoj projektů, které jsou určeny pro více než jednu platformu či operační systém. Hlavním příkladem využití této vlastnosti je vývoj stejné

aplikace pro Android a iOS. Části kódu, které implementují síťování, ukládání a validaci dat, výpočty a další aplikační logiku lze použít pro obě platformy. Pomocí Compose Mutliplatform lze mezi platformami sdílet dokonce i části kódu, které definují UI. Díky tomu lze v Kotlinu vytvářet kompletně multiplatformní mobilní aplikace. [32]

Vývoj Android aplikací pomocí Kotlinu nabízí několik benefitů. Podle oficiálních statistik jsou Android aplikace, které obsahují Kotlin kód, o 20% méně náchylné k pádům. Dále Kotlin podporuje Jetpack Compose, což je doporučovaný nástroj pro tvorbu UI, který bude blíže popsán v dalších částech této kapitoly. [33]

Od roku 2019 je Kotlin společností Google označen jako preferovaný jazyk pro vývoj Android aplikací. Podle oficiálních stránek 50% profesionálních Android vývojářů používá Kotlin jako jejich hlavní jazyk, zatímco pouze 30% používá jazyk Java. Většina programátorů, kteří Kotlin používají, říká, že jsou díky Kotlinu produktivnější. [34]

4.1.3 Shrnutí

Vzhledem k výše uvedeným faktorům, zejména s ohledem na oficiální podporu Google, moderní funkce a integraci s Jetpack Compose, byl Kotlin zvolen jako optimální jazyk pro vývoj této Android aplikace.

4.2 Definice UI

V Androidu existují dva hlavní přístupy k definici uživatelského rozhraní mobilních aplikací - Views a Jetpack Compose. Níže oba způsoby popíši a na závěr jeden z nich zvolím pro vývoj aplikace.

4.2.1 Views

Views je tradiční metoda, která využívá View objekty a XML rozložení pro definici UI.

View objekty představují jednotlivé komponenty, jako jsou tlačítka, textová pole nebo obrázky. Tyto komponenty mají různé atributy a dokáží reagovat na uživatelské akce. Tvoří základní stavební kameny uživatelského rozhraní.

Rozložení jednotlivých komponent se definuje pomocí XML souborů. Do těchto rozložení lze vkládat jednotlivé View objekty. Jedná se například o LinearLayout, který umožňuje vertikální nebo horizontální rozložení.

Na výpisu kódu 4.1 je ukázka definice jednoduchého UI pomocí Views. Tento UI prvek obsahuje text a tlačítko, které jsou seřazené vertikálně pod sebou.

Views je osvědčená metoda a nabízí velkou flexibilitu a kontrolu nad vzhledem a chováním UI, proto se hodí především pro komplexní a složité UI. Na druhou stranu je práce s XML soubory a View objekty náročnější. Další nevýhodou je mnoho boilerplate kódu[1], který se opakuje ve všech XML souborech. [35]

4.2.2 Jetpack Compose

Jetpack Compose je moderní a deklarativní přístup k definici uživatelského rozhraní v Androidu pomocí Kotlin kódu. Nabízí vývojářům možnost popsat UI přímo v kódu, čímž eliminuje nutnost XML souborů a zjednodušuje proces vývoje ve srovnání s tradiční metodou pomocí View objektů. Compose se stále nachází v relativně rané fázi vývoje, ale rychle získává na popularitě díky

[1] Boilerplate kód je termín označující části kódu, které se opakují s drobnými nebo žádnými změnami. Jedná se o kód, který není specifický pro funkcionalitu daného programu, ale je nutný pro základní nastavení.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="match_parent"
              android:layout_height="match_parent"
              android:orientation="vertical" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello, I am a Button" />
</LinearLayout>
```

Výpis kódu 4.1 Ukázka definice jednoduchého UI pomocí Views. UI zobrazuje text a tlačítko ve vertikálním rozložení. Získáno z [35].

```
@Composable
fun ShoppingListItem() {
    Row(verticalAlignment = Alignment.CenterVertically) {
        Icon(imageVector = Icons.Filled.ShoppingCart, contentDescription = "")
        Column(Modifier.padding(5.dp)) {
            Text(text = "Apples")
            Text(text = "20")
        }
    }
}
```

Výpis kódu 4.2 Ukázka definice jednoduchého UI pomocí Jetpack Compose. Výsledek je k vidění na obrázku 4.1.

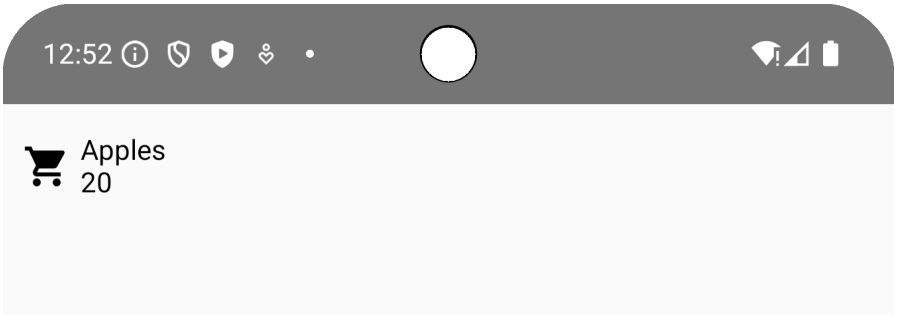
své deklarativní povaze, snadné syntaxi a podpoře moderních funkcí, jako je živý náhled (live preview) a animace.

Základním stavebním kamenem jsou Composable funkce. Tyto funkce jsou označené anotací @Composable, která informuje Compose kompilátor, že daná funkce definuje UI prvek. V parametrech mohou přijímat data, která následně zobrazují. Composable funkce nemají návratovou hodnotu. [36, 37]

Opakovaným zavoláním Composable funkce s novými daty dojde k aktualizaci a překreslení UI. Tento proces se nazývá rekompozice. Jetpack Compose framework dokáže inteligentně vybrat prvky, které je potřeba při změně zobrazovaných dat rekomponovat. Tím šetří čas i výpočetní prostředky zařízení.

Bezespornou výhodou je jednoduchost a intuice, se kterou lze UI definovat. UI se definuje přímo v kódu, čímž se eliminuje potřeba XML souborů a zjednodušuje se vývoj. Tento způsob se hodí především pro rychlý a iterativní vývoj. Jetpack Compose je pro tvorbu UI Androidem doporučovaný nástroj. Některé funkcionality jsou stále ve vývoji a mnoho věcí se může v průběhu času měnit, což může být nevýhodou při údržbě aplikací.

Na výpisu kódu 4.2 je k vidění ukázka použití Jetpack Compose k vytvoření jednoduchého UI. Výsledný UI prvek je k vidění na obrázku 4.1.



Obrázek 4.1 Výsledek definice UI z ukázky kódu 4.2.

4.2.3 Shrnutí

Potřeba rychlého vývoje a zároveň nepříliš komplexního UI byly hlavní důvody pro zvolení Jetpack Compose jako hlavního nástroje pro definici UI vyvíjené aplikace. Důležitým důvodem bylo také oficiální doporučení, které označuje tento nástroj jako nástroj číslo jedna pro tvorbu UI u moderních Android aplikací.

4.3 Architektura

Architekturu softwaru můžeme chápat jako základní organizaci systému. Zahrnuje komponenty systému, jejich vzájemné vztahy, vztah k prostředí a principy, které řídí návrh a vývoj systému. Je to vlastně koncept, který definuje fundamentální aspekty systému. [38]

4.3.1 Obecné architektonické principy

Na oficiálních stránkách Androidu je sekce, která se přímo věnuje doporučené architektuře a osvědčeným architektonickým principům při vývoji aplikací. Níže přiblížím všechny doporučené principy. [39]

Separation of concerns

Separation of concerns (oddělení odpovědnosti) je velmi obecný princip, který říká, že bychom měli počítačový program rozdělit na oddělené části. Každá část by potom měla mít svou vlastní a jedinou odpovědnost. Vícevrstvá architektura je příkladem použití tohoto principu.

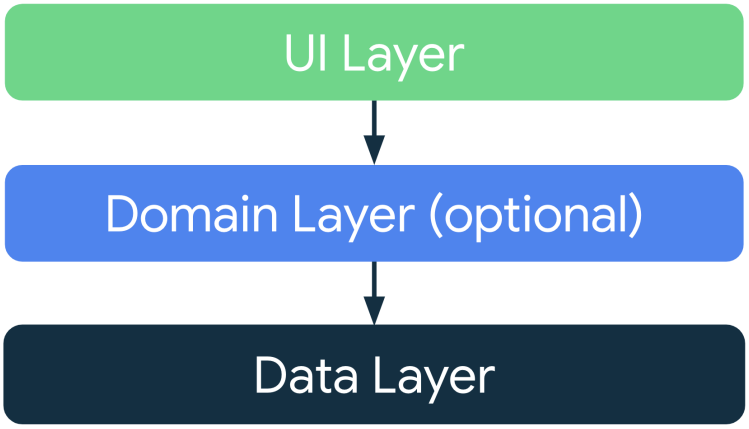
V Androidu je chybou psát veškerý kód do jediné Activity nebo Fragmentu. V těchto třídách by se měl nacházet pouze kód, který se stará o UI a interakce s operačním systémem. Pro omezení problémů spojených s životním cyklem těchto Android tříd a pro jednodušší testování, je potřeba udělovat těmto třídám pouze jedinou odpovědnost.

Tento princip bude splněn díky použití vícevrstvé architektury, která bude ukázána a popsána v pozdějších částech této kapitoly.

Řízení UI z datových modelů

Tento princip říká, že logika prezentace (UI) by měla být oddělena od logiky dat (datové modely). UI prvky aplikace by neměly přímo manipulovat s daty, naopak by měly využívat datové modely, které data reprezentují a spravují.

Prvky UI mají omezený životní cyklus, který nebývá pod kontrolou uživatele, ale bývá často spravován systémem. Uživatel by mohl ztratit svá data, pokud by data přímo závisela na UI prvcích. Proto je potřeba data ukládat pomocí datových modelů.



Obrázek 4.2 Diagram znázorňující typickou vícevrstvou architekturu Android aplikace. Získáno z [39].

Single Source of Truth

Single source of truth (SSOT) je koncept, který zajišťuje, že existuje pouze jediné místo, kde jsou data uložena a spravována. Tento princip zaručuje konzistenci dat v různých částech aplikace. Tím snižuje riziko chyb způsobených datovou inkonzistencí. Dále usnadňuje sledovatelnost, protože změny probíhají pouze na jednom místě. Implementace SSOT usnadňuje správu dat, zjednodušuje údržbu a vede k celkově robustnější a testovatelnější aplikaci.

Unidirectional Data Flow

Unidirectional Data Flow (Jednosměrný datový tok) je princip, který říká, že data tečou v aplikaci jedním směrem a naopak události, které data modifikují, tečou opačným směrem.

Tento princip bývá často spojován s předchozím principem Single Source of Truth. Na příkladu popsaném níže je vidět použití obou principů.

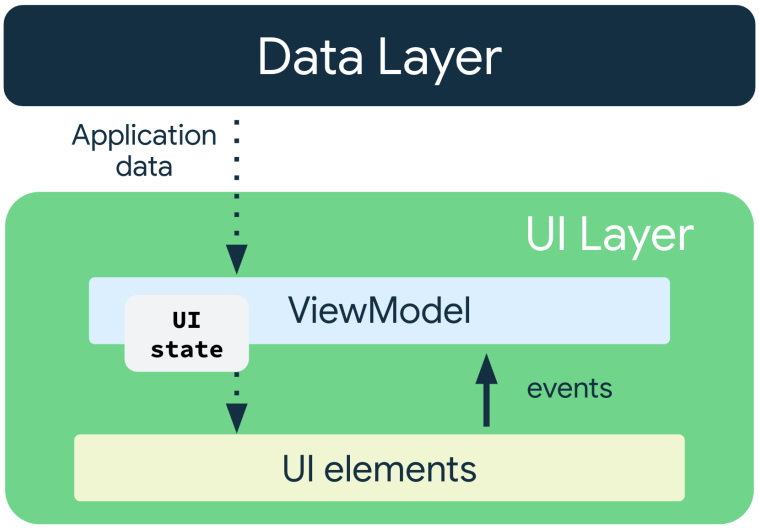
Příkladem může být Android aplikace, která zobrazuje nákupní seznam. Jednotlivé položky jsou data, která tečou z databáze (datový zdroj a zároveň SSOT) do uživatelského rozhraní, kde se zobrazují. Akce „stisknutí tlačítka smazat“ potom naopak putuje z uživatelského rozhraní do datového zdroje. Zde jsou vybraná data smazána. Směrem z datového zdroje do uživatelského rozhraní následně k zobrazení putují upravená data.

4.3.2 Architektura aplikace

Na základě doporučení z oficiálních stránek Androidu a obecných architektonických principů popsaných výše bude vyvíjená aplikace respektovat vícevrstvou architekturu. Aplikace se bude skládat ze tří vrstev:

- **Prezentační vrstva (také UI vrstva)** – zobrazuje data na obrazovce
- **Doménová vrstva** – zapouzdřuje složitější logiku aplikace a obsahuje hlavní funkcionality
- **Datová vrstva** – spravuje data aplikace a dodává je ostatním vrstvám

Na obrázku 4.2 lze vidět diagram typické architektury Android aplikace. Šipky v diagramu představují směr závislostí. Jak je vidět, UI vrstva (prezentační vrstva) závisí na doménové vrstvě a doménová vrstva závisí na datové vrstvě.



Obrázek 4.3 Diagram znázorňující typické použití ViewModelu a UI stavu v prezentační vrstvě Android aplikace. Získáno z [40].

Prezentační vrstva

Prezentační vrstva zobrazuje data na obrazovce. Vrstva přijímá data z datové vrstvy a převádí je do formy, kterou lze zobrazit. Poté tato data vykresluje na obrazovku pomocí UI prvků a reaguje na uživatelské akce.

Tato vrstva se skládá ze dvou typů objektů:

- **UI prvky** – V případě naší aplikace jsou to prvky definované pomocí Jetpack Compose. Tyto prvky zobrazují na obrazovce data aplikace.
- **State holders** – Třídy, které udržují aktuální stav UI a dodávají data pro UI prvky. Tyto třídy také reagují na uživatelské akce.

V Android aplikacích je State holder obvykle implementován pomocí třídy ViewModel. Tato třída získává data z datové vrstvy a udržuje je jako stav UI. Tento stav následně dodává do UI prvků k zobrazení a zároveň dokáže reagovat na akce, které přicházejí z UI prvků. Na obrázku 4.3 je vidět diagram znázorňující popsané použití. Tento způsob implementace zároveň respektuje princip Unidirectional Data Flow (Jednosměrný datový tok) popsaný v předchozích textu. [40]

Doménová vrstva

Doménová vrstva slouží především k zapouzdření funkcí, které se používají ve více než jednom ViewModelu. V této vrstvě se také nachází složitější byznysová operace, které je pro přehlednost vhodnější přesunout z prezentační vrstvy do doménové. Díky tomu se zvyšuje znovupoužitelnost a snižuje duplikace kódu.

Příkladem může být situace, kdy si přejeme veškeré datum v aplikaci formátovat určitým způsobem, který si uživatel zvolil. Vytvořili bychom třídu FormatDateUseCase, která by využívala UserRepository pro získání informací o uživatelské volbě. Na základě této volby by datum formátovala. Všechny ViewModely by tuto třídu využívaly k formátování dat podle uživatelské volby. [41]

Datová vrstva

Datová vrstva má na starosti data a byznysovou logiku aplikace. Byznysová logika je něco, co aplikaci odlišuje od jiných aplikací a dává aplikaci hodnotu. Jedná se o logiku, která určuje, jak aplikace vytváří, ukládá a mění data.

V Android aplikacích datovou vrstvu obvykle tvoří několik tříd Repository. Každá z těchto tříd pracuje s jinými daty aplikace. Konkrétním příkladem, který vyplývá z požadavků naší aplikace, může být třída PhoneCallRepository, která pracuje s telefonními hovory, a třída TimeEntryRepository, která pracuje s časovými výkazy. [42]

4.3.3 Shrnutí

Aplikace bude vyvíjena podle vícevrstvé architektury, která se skládá z prezentační, doménové a datové vrstvy. Tato architektura přináší řadu benefitů, jako je oddělení zodpovědností, snížení duplicity kódu, zvýšení testovatelnosti a usnadnění údržby. Díky rozdělení funkcionality do tří vrstev bude kód aplikace lépe strukturovaný a srozumitelnější. Navíc bude jednodušší přidávat nové funkce a upravovat existující bez nutnosti zasahovat do jiných částí kódu.

4.4 Persistence dat

Z popsaných požadavků vyplývá potřeba ukládat určitá data v aplikaci. Především bude nutné uložit API klíč, který uživatel zadá do aplikace. Pokud chceme, aby aplikace fungovala i bez připojení k internetu, je nezbytné ukládat data získaná během dostupného připojení do lokálního úložiště telefonu. V případě ztráty připojení zůstanou data dostupná.

V Androidu existuje řada možností, jak ukládat data aplikace. Každá z možností má své výhody i nevýhody, a proto se hodí pro různé případy. Níže představím několik hlavních možností, které lze využít pro ukládání dat.

4.4.1 DataStore

Jetpack DataStore je knihovna pro asynchronní, konzistentní a transakční ukládání dat v Android aplikacích. Díky těmto vlastnostem nabízí řadu výhod:

- **Asynchronní operace**: Ukládání a načítání dat probíhá asynchronně, čímž se nezpomaluje hlavní vlákno aplikace.
- **Konzistence**: DataStore zaručuje konzistenci dat i při více souběžných přístupech z více vláken.
- **Transakce**: Ukládání dat probíhá transakčně, což znamená, že se buď uloží všechna data nebo žádná.

DataStore umožňuje ukládat dva typy dat:

- **Páry klíč-hodnota**: Ukládá data pomocí klíče, podle kterého se k datům velmi jednoduše přistupuje. Na druhou stranu postrádá typovou bezpečnost. Vhodné pro jednoduché konfigurace a uživatelská nastavení.
- **Typované objekty**: Umožňuje ukládat strukturovaná data s typovou bezpečností, což je ideální pro komplexní nastavení a konfigurace.

Data uložená pomocí DataStore jsou přístupná pouze samotné aplikaci a spolu s aplikací jsou odstraněna při odinstalování. Jedná se o nástupce knihovny SharedPreferences, a je doporučeno na ni přejít [43].

Knihovna je ideální volbou pro ukládání uživatelských nastavení, výchozích hodnot a konfigurací aplikací. V případě vyvíjené aplikace lze ukládat výchozí hodnoty pro vykazování nebo nastavení jazyka aplikace. [44]

4.4.2 Keystore

Android Keystore je součástí Android platformy a umožňuje bezpečné ukládání kryptografických klíčů v zařízení. Data uložená v Keystore jsou chráněna před neoprávněným přístupem a zneužitím pomocí hardwarových bezpečnostních funkcí a softwarových kryptografických mechanismů. Hlavní využití u vyvíjené aplikace najde knihovna při ukládání API klíčů a dalších přihlašovacích údajů pro přístup do aplikací třetích stran. [45]

4.4.3 Room

Room je knihovna, která zjednodušuje práci s databázemi SQLite v Android aplikacích. Na pozadí knihovna ukládá data do SQLite databáze. Díky ORM (Object Relational Mapping) umožňuje mapovat objekty v aplikaci na entity v databázi. Knihovna přináší oproti přímé práci s SQLite několik výhod:

- **Zjednodušení kódu**: Room eliminuje nutnost psát SQL dotazy ručně a snižuje tak boilerplate kód.
- **Typová bezpečnost**: Room kontroluje SQL příkazy v době kompilace, čímž se snižuje riziko chyb.
- **Migrace databáze**: Room podporuje migraci databáze, což umožňuje plynulé aktualizace aplikace bez ztráty dat.
- **Asynchronní operace**: Room umožňuje asynchronní operace s databází, což zlepšuje responzivitu aplikace.

Stejně jako u DataStore jsou data v Room databázi přístupná pouze samotné aplikaci a jsou odstraněna spolu s odinstalováním aplikace.

Room je ideální pro ukládání objektů s velkým počtem atributů, ale i pro menší datové sady. Díky své jednoduchosti a robustnosti je Room ideální volbou pro efektivní práci s databázemi. [46]

4.5 Dependency Injection

Dependency Injection (DI) neboli vkládání závislostí je návrhový vzor, který usnadňuje vývoj udržitelných a testovatelných aplikací. Díky DI se snižuje provázanost mezi třídami a zvyšuje se jejich znovupoužitelnost.

Většina tříd má závislosti, které využívá pro svou funkcionalitu. Například ViewModel má jako závislost Repository, aby mohl získávat data. Třída si může sama při svém vytváření vytvořit instanci daného Repository. Tento způsob ale zvyšuje provázanost obou tříd. Při testování potom nelze jednoduše změnit implementaci Repository, což způsobuje problémy.

Jinou možností je přenechat dodání závislosti na tom, kdo instanci třídy vytváří. Přesně na této myšlence je založená DI. **Princip DI je postaven na dodání potřebných závislostí až při konstrukci instance dané třídy**.

V příkladu výše by konstruktor třídy ViewModel přijímal v parametru instanci Repository. Úkolem toho, kdo vytváří ViewModel, je potom požadovanou instanci dodat a uspokojit tak potřebné závislosti. Tomuto způsobu se říká manuální vkládání závislostí.

Manuální vkládání závislostí vyžaduje psaní opakujícího se kódu, čímž se zvyšuje riziko chyb. Proto existuje řada knihoven, které se starají o automatickou správu a vkládání závislostí. V

Androidu je nejrozšířenější a zároveň oficiálně doporučovanou volbou knihovna Hilt, která je založená na populárním DI nástroji pro Javu Dagger. [47]

Použití DI je doporučovanou technikou při vývoji moderních a udržitelných Android aplikací. Z tohoto důvodu i na základě oficiálního doporučení bude při implementaci knihovna Hilt použita. [48]

4.6 Ostatní technologie

4.6.1 Retrofit

Retrofit je open-source knihovna pro Android, která slouží k usnadnění síťových požadavků a asynchronní komunikace s webovými API. Umožňuje jednodušeji a efektivněji konstruovat a odesílat HTTP požadavky na servery a zpracovávat jejich odpovědi. [49]

V aplikaci bude Retrofit využíván pro posílání API požadavků do externích aplikací.

4.6.2 Kotlinx Serialization

Kotlinx Serialization je open-source knihovna vyvinutá společností JetBrains specificky pro Kotlin projekty. Zjednodušuje proces převodu Kotlin objektů do různých datových formátů, jako je JSON, CBOR, Protobuf a další, a i naopak opačným směrem (deserializace).[50]

V rámci aplikace bude knihovna použita pro serializaci a deserializaci JSON formátu dat, který se běžně používá při komunikaci s webovými API.

4.7 Toggl Track jako primární úložiště

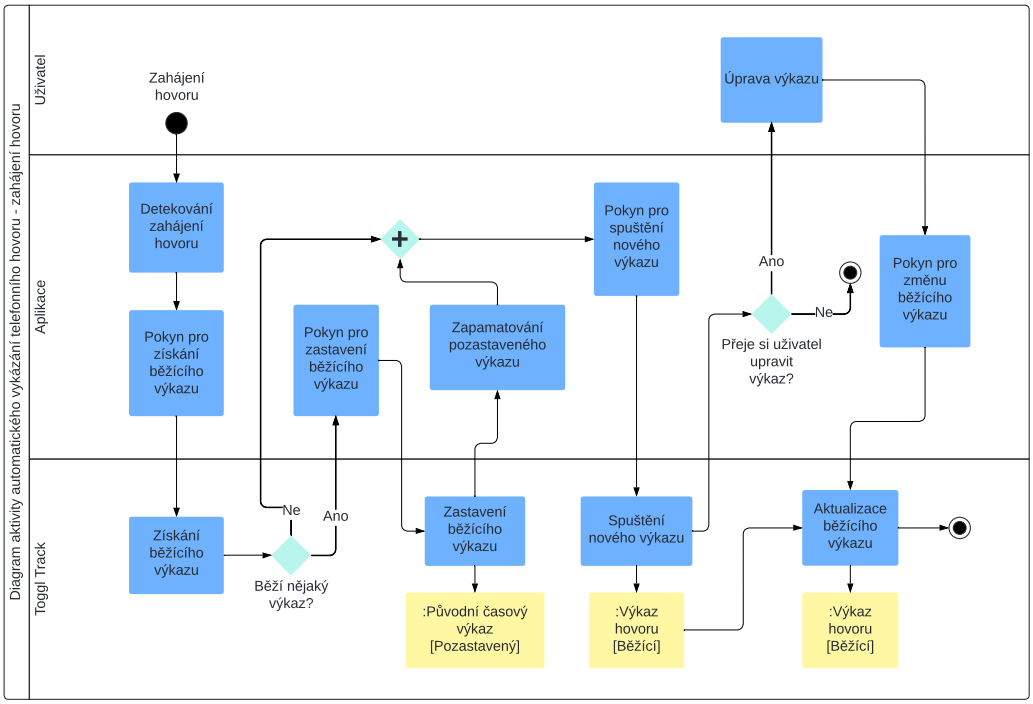
Jednou z možností implementace je využití nástroje Toggl Track jako primárního úložiště pro některá data aplikace. Jak bylo podrobněji popsáno v analýze, Toggl Track nabízí spolehlivé API, pomocí kterého lze provádět řadu akcí a pracovat s daty.

Především je možné přes API vytvářet časové výkazy. Díky možnému značkám lze jednoduše odlišit výkazy vytvořené aplikací od těch ostatních. Dále je uživateli umožněno vidět a upravovat vykázané hovory i v dalších aplikacích nástroje Toggl Track, jako například desktopové nebo webové aplikaci. Další možností API je spouštět a zastavovat časomíru výkazů. Toho lze využívat při detekování zahájení a ukončení telefonních hovorů.

V tomto případě bude vyvíjená aplikace fungovat v podstatě jako nadstavba existující aplikace Toggl Track. Samozřejmě není v autorových silách vytvořit plnohodnotnou kopii aplikace a přidat funkcionality týkající se telefonních hovorů. Taková aplikace by mnohokrát přesahovala rozsah této práce. Lze ale vytvořit odlehčenou verzi, která oproti původní Toggl Track aplikaci dokáže automaticky nebo manuálně vykazovat telefonní hovory a následně dovolí uživateli tyto výkazy spravovat.

Další výhodou tohoto způsobu implementace je možnost využití aplikace Timer2Ticket. Jak bylo popsáno v analýze, přímá integrace vyvíjené aplikace s aplikací Timer2Ticket je náročná. Díky využití aplikace Toggl Track bude umožněno jednoduše využít existující řešení Timer2Ticket a výkazy hovorů s nástroji pro správu projektů synchronizovat.

Nevýhodou je přímá provázanost s Toggl Trackem. Pro používání vyvíjené aplikace bude nutné mít vytvořený Toggl Track účet. Propojení s API bude vést při případných změnách API k nutné úpravě aplikace.



■ **Obrázek 4.4** Aktivity diagram procesu automatického vykázání telefonního hovoru - část zahájení hovoru.

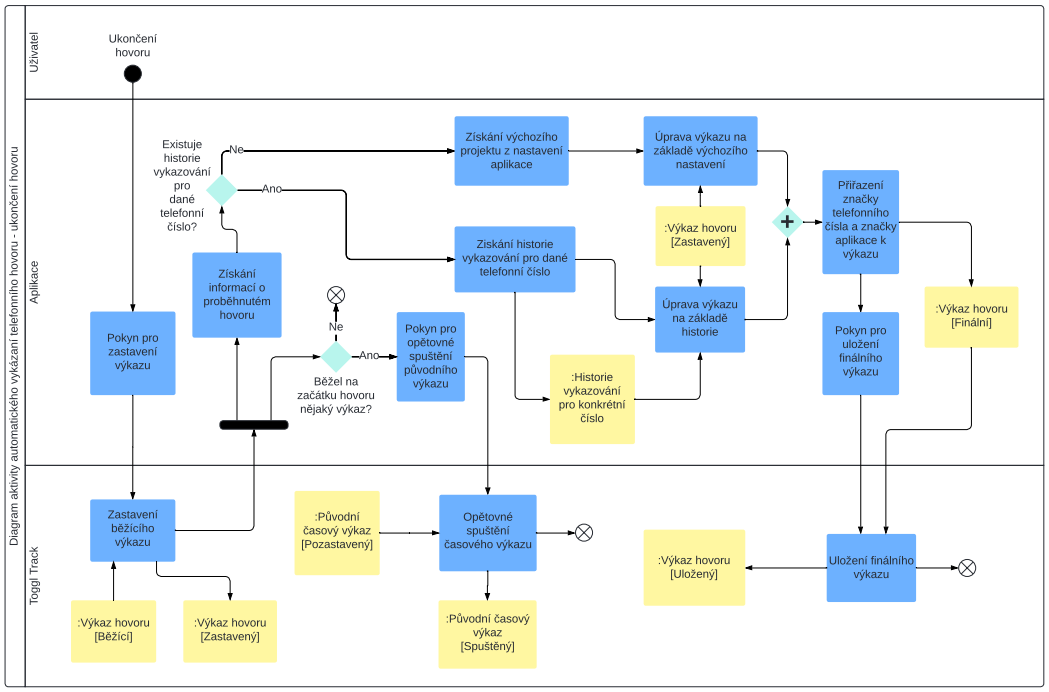
Shrnutí

Využití Toggl Track jako primárního úložiště dat nabízí řadu benefitů, včetně spolehlivého API, možnosti přidávání značek a integrace s existujícími nástroji (Timer2Ticket). Hlavní nevýhodou je závislost na Toggl Track účtu a nutnost úprav aplikace v případě změn API. Po důkladném zvážení všech aspektů autor dospěl k závěru, že Toggl Track představuje optimální řešení pro implementaci úložiště dat v aplikaci.

Pro představu byl vytvořen diagram aktivity, který popisuje proces automatického vykázání telefonního hovoru. Na obrázku 4.4 je vidět jeho první část. Tato část popisuje proces při zahájení telefonního hovoru. Na dalším obrázku 4.5 se nachází druhá část, která se zaměřuje na popis procesu během ukončení telefonního hovoru. Z diagramu je patrný zamýšlený způsob využití aplikace Toggl Track pro vytváření a správu časových výkazů.

4.8 Návrh obrazovek

V této části práce budou na základě architektury a získaných požadavků popsány konkrétní návrhy hlavních obrazovek aplikace. Návrh obrazovek bude popsán pomocí drátěných modelů (wireframes), což jsou velmi jednoduché náčrty, které se soustředí pouze na rozložení a hlavní funkcionality dané obrazovky a opomíjejí vizuální design. Tyto náčrty budou vytvořeny v aplikaci Sketch [51].



Obrázek 4.5 Aktivity diagram procesu automatického vykázání telefonního hovoru - část ukončení hovoru.

4.8.1 Úvodní obrazovka

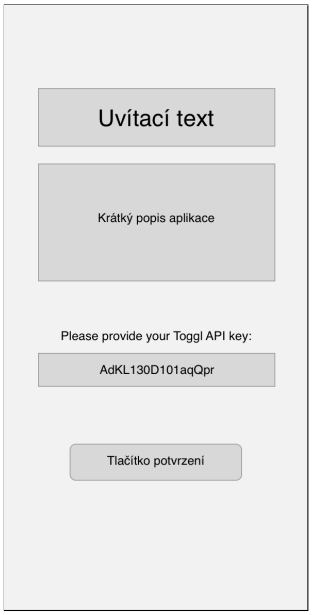
Úvodní obrazovka uživatele přivítá při prvotním spuštění aplikace po jejím nainstalování. Na obrazovce bude pole, do kterého se zadá Toggl Track API klíč. Dále obrazovka bude obsahovat krátký popis aplikace a tlačítko, kterým bude možné zadaný klíč potvrdit. Po stisknutí se provede kontrola klíče, v případě že klíč bude v pořádku, uživatel bude přesunut na obrazovku *Calls*. V opačném případě bude vyzván k opravě klíče. Bez API klíče nelze aplikaci používat, proto uživatel tuto obrazovku uvidí, dokud nezadá platný klíč. Na obrázku 4.6 lze vidět wireframe úvodní obrazovky.

4.8.2 Obrazovka *Settings*

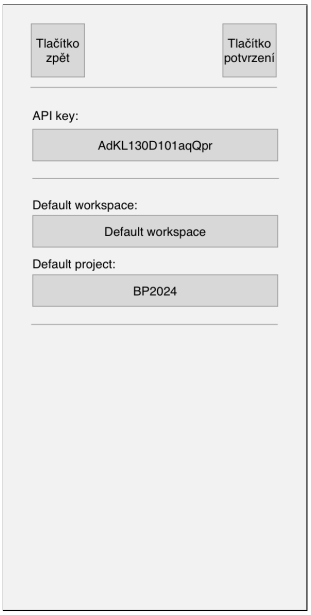
Obrazovka *Settings* umožní uživateli měnit nastavení aplikace. Především lze změnit nebo smazat API klíč. Dále je možné upravit výchozí hodnoty pro vykazování hovorů. Hovory se budou automaticky vykazovat k nastavenému workspace a projektu, pokud aplikace nedokáže z historie vykazování určit jinak. Nastavení bude nutné uložit potvrzovacím tlačítkem. Během potvrzení proběhne kontrola funkčnosti API klíče a v případě nefunkčnosti bude uživatel upozorněn. Dalším tlačítkem se bude možné bez uložení vrátit na předchozí obrazovku. Na obrázku 4.7 lze vidět wireframe obrazovky *Settings*.

4.8.3 Obrazovka *Calls*

Obrazovka *Calls* slouží k zobrazení telefonních hovorů, které je z této obrazovky možné manuálně vykázat. V horní části obrazovky se nachází tlačítko pro přechod na obrazovku *Settings*. Dále



Obrázek 4.6 Wireframe úvodní obrazovky aplikace



Obrázek 4.7 Wireframe obrazovky *Settings*, na které lze měnit nastavení aplikace.

je na obrazovce panel pro filtrování seznamu telefonních hovorů. Pod tímto panelem se nachází samotný seznam hovorů. Pomocí navigačního panelu na spodní straně je možné přecházet na obrazovku *Entries*. Na obrázku 4.8 je vidět wireframe této obrazovky.

4.8.4 Obrazovka *Call Export*

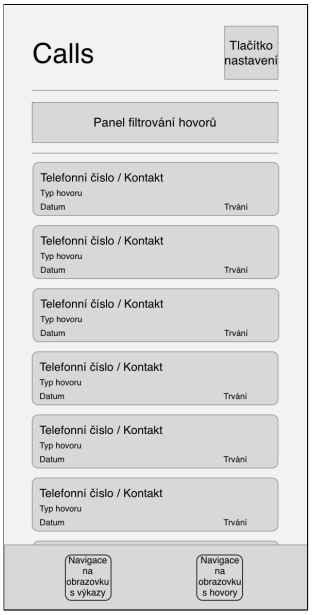
Na obrazovku *Call Export* se uživatel přesune po kliknutí na položku ze seznamu hovorů na obrazovce *Calls*. Obrazovka slouží především jako shrnutí výkazu, který se uživatel chystá vytvořit. Uživatel zde vidí k výkazu přiřazený workspace, projekt i značky a má zde možnost tyto parametry změnit. Potvrzovacím tlačítkem hovor vykáže. Tlačítkem zpět se uživatel může vrátit na předchozí obrazovku. Na obrázku 4.9 lze vidět wireframe této obrazovky.

4.8.5 Obrazovka *Time Entries*

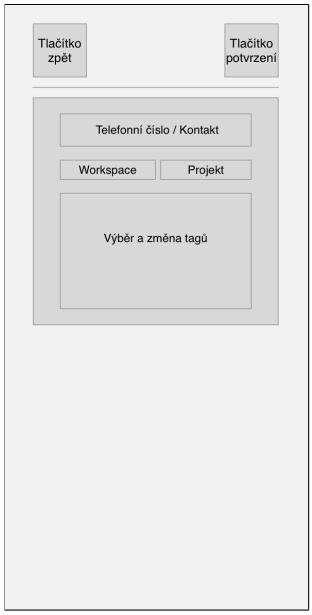
Obrazovka *Time Entries* zobrazuje výkazy vytvořené aplikací, tedy všechny vykázané hovory. Z obrazovky je možné pomocí tlačítka přejít na obrazovku *Settings*. Stejně jako *Calls* umožňuje i tato obrazovka filtrování seznamu a pomocí navigačního panelu přejít na obrazovku *Calls*. Na obrázku 4.10 je zobrazen wireframe obrazovky *Time Entries*.

4.8.6 Obrazovka *Time Entry Detail*

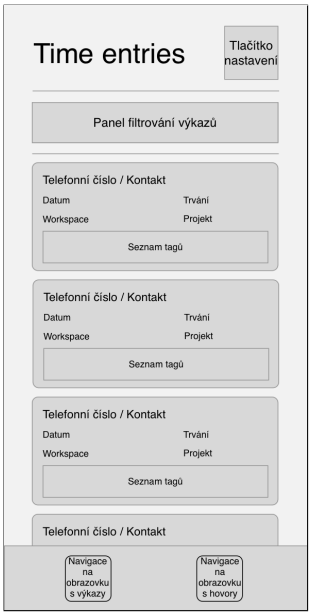
Na obrazovku *Time Entry Detail* se uživatel dostane z obrazovky *Time Entries* po kliknutí na konkrétní výkaz ze seznamu. Na obrazovce je možné měnit parametry výkazu a také celý výkaz smazat. Změny parametrů uživatel musí potvrdit potvrzovacím tlačítkem. Tlačítkem zpět se uživatel vrátí na obrazovku *Time Entries*. Na obrázku 4.11 lze vidět wireframe této obrazovky.



■ **Obrázek 4.8** Wireframe obrazovky *Calls* zobrazující telefonní hovory.



■ **Obrázek 4.9** Wireframe obrazovky *Calls Export* umožňující vykázat zvolený telefonní hovor jako časový výkaz.



■ **Obrázek 4.10** Wireframe obrazovky *Time Entries* zobrazující všechny časové výkazy.

■ **Obrázek 4.11** Wireframe obrazovky *Time Entry Detail* zobrazující detailní popis vybraného časového výkazu.

Kapitola 5
Implementace

V této kapitole se autor bude zabývat implementačními detaily aplikace. Na začátku krátce vysvětlí, jak byly implementovány jednotlivé vrstvy aplikace. Dále se zaměří na popis implementace složitějších a komplexnějších funkcionalit. Také porovná realizaci jednotlivých obrazovek s jejich původním návrhem a odůvodní provedené změny. Na závěr autor popíše proces vydání aplikace a přiblíží problémy, které proces doprovázely.

5.1 Vývojové prostředí

Vývoj aplikace probíhal výhradně v oficiálním vývojovém prostředí pro Android aplikace, a to v Android Studiu od firmy Google. Toto IDE je založeno na známém produktu JetBrains – IntelliJ IDEA. Android Studio je dostupné pro operační systémy Windows, macOS a také pro systémy založené na Linuxu. [52]

Android Studio je speciálně navrženo pro tvorbu Android aplikací a nabízí řadu funkcí, které vývoj výrazně usnadňují:

- **UI Editor** - umožňuje jednoduše měnit a konfigurovat uživatelské rozhraní vyvíjené aplikace
- **Live Preview** - umožňuje zobrazovat definované UI komponenty bez spuštění aplikace a okamžitě sledovat změny kódu
- **Android Emulátor** - virtuální zařízení s operačním systémem Android, na kterém lze spouštět a testovat vyvíjené aplikace

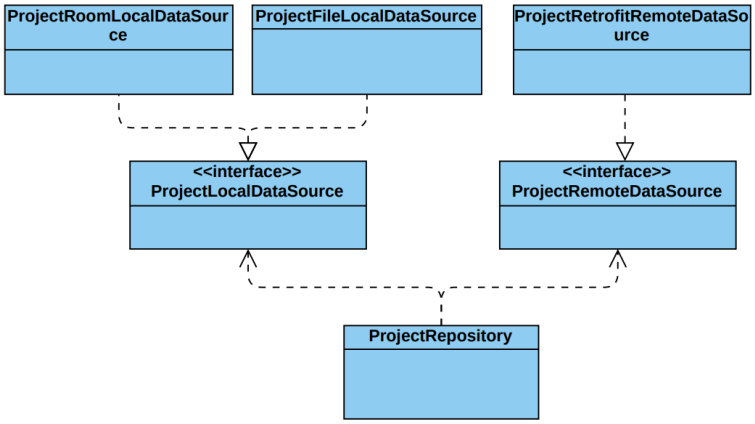
Autor nemá k dispozici žádné zařízení se systémem Android, proto byl Android Emulátor nezbytným nástrojem během vývoje aplikace.

5.2 Implementace vrstev architektury

Jak bylo popsáno v kapitole věnující se návrhu, je aplikace navržena podle vícevrstvé architektury. V této sekci bude popsáno, jak jsou jednotlivé vrstvy implementovány. U každé vrstvy zmíním hlavní třídy, které se ve vrstvě nacházejí, a krátce popíšu tyto třídy.

5.2.1 Datová vrstva

Datová vrstva poskytuje zbytku aplikace přístup k datům. Tvoří ji repozitáře (třídy *Repository*) a zdroje dat (třídy *DataSource*). Pro každý typ dat, se kterým se v aplikaci pracuje, je vytvořen



Obrázek 5.1 Ukázka teoretického diagramu tříd, který by mohl tvořit datovou vrstvu pro projekty.

jeden repozitář (např. ProjectRepository nebo *CallRepository*), který závisí na jednom nebo více zdrojích dat (např. *ProjectRetrofitDataSource* nebo *CallSystemDataSource*).

Obecně jsou zdroje dat dvou druhů - lokální (rozhraní *LocalDataSource*) a vzdálený (rozhraní *RemoteDataSource*). Tyto třídy představují pouze rozhraní, které potom konkrétní datové zdroje implementují.

Pro lepší představu o implementaci této vrstvy je na obrázku 5.1 vidět teoretický příklad diagramu tříd, které tvoří datovou vrstvu pro data o projektech. Jedná se o teoretický model, který by se využil, pokud by aplikace ukládala data v lokálním úložišti, které by se využívalo, kdyby aplikace neměla přístup k internetu. Na obrázku je lokální úložiště implementováno dvěma způsoby – pomocí vlastních souborů a pomocí Room databáze. Repozitář pro svou práci využívá jednu z těchto implementací. Dále potom využívá implementaci vzdáleného zdroje dat.

Jelikož aplikace prozatím nepodporuje fungování bez připojení k internetu, nebylo nutné implementovat lokální zdroj dat. Projektový repozitář (třída *ProjectRepository*) má jediný zdroj dat – vzdálený Toggl Track API zdroj implementovaný pomocí knihovny Retrofit (třída *ProjectRetrofitRemoteDataSource*).

Výhodou použití tohoto způsobu implementace je jednodušší přidání dalších datových zdrojů do datové vrstvy. Při budoucím rozšíření aplikace o fungování bez internetového připojení nebude díky dědičnosti nutné provádět velké zásahy do kódu aplikace.

5.2.2 Doménová vrstva

Doménová vrstva aplikace obsahuje doménové datové třídy, které reprezentují data, se kterými aplikace pracuje (např. třída *TimeEntry*). Tyto třídy jsou využívány jak datovou vrstvou jako výstupní data repozitářů, tak prezentační vrstvou jako vstupní data pro své State holdery. Doménové datové třídy slouží k propojení datové a prezentační vrstvy aplikace. Většina datových zdrojů pracuje se speciální datovou třídou, která slouží výhradně danému zdroji dat (např. *ApiTimeEntry* nebo *ApiProject*). Tyto třídy obsahují nadbytečné informace, které jsou relevantní pouze pro zdroj dat. Pro oddělení těchto informací bylo zvoleno právě využití výše zmíněných doménových datových tříd, na které se mapují datové třídy zdrojů dat.

Dále se doménová vrstva skládá z tzv. případů užití (třídy *UseCase*). Jedná se o třídy, které zapouzdřují komplexnější byznysovou logiku. V aplikaci se jedná o jedinou třídu *ExportCallUseCase*, která má za úkol vykazovat telefonní hovory. Pro správné vykázání je nutné získat historii vykazování a případně také hodnoty z nastavení aplikace, a s těmito daty pracovat při vytváření výkazu. Původně byla tato logika součástí třídy *ViewModel* v prezentační vrstvě. Ale jelikož je

celý proces složitý a je opakovaně využíván na více místech aplikace, byla logika přesunuta právě do doménové vrstvy pod třídu *ExportCallUseCase*.

5.2.3 Prezentační vrstva

Prezentační vrstva slouží k zobrazování dat a reagování na uživatelem spuštěné akce. Tvoří ji samotné obrazovky (třídy *Screen*) a třídy, které reagují na akce a udržují data obrazovek (třídy *ViewModel*).

Každá třída *Screen*, reprezentující obrazovku v aplikaci, je závislá na svém *ViewModelu*. Pomocí funkcí z Jetpack Compose sleduje změny dat svého ViewModelu a v případě změn se spouští rekompozice dotčených UI prvků. Při akcích vyvolaných uživatelem, volají metody svých *ViewModelů*. Tyto metody dále modifikují data nebo provádějí složitější akce.

ViewModel třídy jsou většinou závislé na několika repozitářích (datová vrstva), případně některé závisí na třídě *UseCase* (doménová vrstva). Tyto závislosti *ViewModel* využívá k získávání dat i k reagování na akce z obrazovek.

Podle architektury by měla prezentační vrstva záviset pouze na vrstvě doménové. Tedy *ViewModel* by měl záviset pouze na *UseCase* třídě. V aplikaci byla ale závislost *ViewModelu* přímo na repozitáři povolena. Ve většině případů je totiž v *ViewModelu* nutné pouze získávat data a neprovádět žádnou složitější logiku. Pokud by *ViewModel* mohl záviset pouze na *UseCase* třídách, bylo by nutné pro každý repozitář vytvořit zvláštní *UseCase* třídu, která by pouze delegovala volání metody pro získání dat na odpovídající repozitář. To by vedlo k nadbytečnému kódu. Z tohoto důvodu bylo upuštěno od striktního dodržování závislostí jednotlivých vrstev a umožněno ve výše popsaném případě prezentační vrstvě přímo záviset na datové vrstvě.

5.3 Komunikace s Toggl Track API

Jak bylo popsáno v kapitole o návrhu, aplikace Toggl Track slouží jako primární úložiště dat. Pro práci s těmito daty je využíváno Toggl Track API, díky kterému jsou data uložena na serveru a jsou přístupná i z jiných aplikací.

Toggl Track API je RESTful API, kde každý zdroj je jednoznačně identifikován svým URI a pomocí HTTP metod s ním lze jednoduše pracovat. V současné době jsou v aplikaci využívány čtyři Toggl Track zdroje – workspaces, projekty, značky a časové záznamy.

Pro komunikaci s API je používána knihovna Retrofit (viz sekce 4.6.1). Pro převod dat ve formátu JSON, s nimiž RESTful API pracuje, na Kotlin objekty a obráceně, se používá knihovna Kotlinx Serialization (viz sekce 4.6.2).

Na ukázce kódu 5.1 je vidět inicializace potřebných Retrofit objektů pro komunikaci s Toggl Track API. Nejprve je vytvořen interceptor, který přidává API klíč do hlaviček všech HTTP požadavků. Poté je inicializována třída OkHttpClient, která se stará o komunikaci se serverem, a do ní je zaregistrován předchozí interceptor. Nakonec je pomocí tohoto klienta vytvořena instance Retrofitu, která slouží pro komunikaci s API.

V ukázkách jsou použity anotace z DI knihovny Hilt. Funkce s anotací *@Provides* definují poskytovatele závislostí. Návratový typ u anotované funkce *provideRetrofit* je typ *Retrofit*. Kdykoliv bude v programu potřeba závislosti na třídě *Retrofit*, použije se instance vytvořená v této funkci s anotací *@Provides*. Díky anotaci *@Singleton* máme zaručeno, že se vždy dodá stejná a jediná instance třídy *Retrofit*.

5.4 Vykázání telefonního hovoru

Vykazování telefonních hovorů je řízeno třídou *ExportCallUseCase*. Tato třída pro svou práci využívá několik repozitářů – *TimeEntryRepository*, *SettingsRepository*, *CallRepository* a *ProjectRepository*.

```
private const val BASE_URL = "https://api.track.toggl.com/api/v9/"

@Singleton
class ApiKeyInterceptor @Inject constructor(
    private val settingsRepository: SettingsRepository
) : Interceptor {

    private var key: String
    init { runBlocking { key = settingsRepository.getApiKey().first() } }
    fun updateApiToken(token: String) {key = token}

    override fun intercept(chain: Interceptor.Chain): Response {
        val credentials = Credentials.basic(key, "api_token")
        val newRequest: Request =
            chain.request()
                .newBuilder()
                .addHeader("Authorization", credentials)
                .addHeader("Content-Type", "application/json")
                .build()
        return chain.proceed(newRequest)
    }
}

@Singleton
@Provides
fun provideOkHttpClient(keyInterceptor: ApiKeyInterceptor): OkHttpClient {
    return OkHttpClient.Builder()
        .addInterceptor(keyInterceptor)
        .build()
}

@Singleton
@Provides
fun provideRetrofit(
    client: OkHttpClient,
    exceptionMappers: @JvmSuppressWildcards List<HttpExceptionMapper>
): Retrofit {
    val json = Json { ignoreUnknownKeys = true }
    return Retrofit.Builder()
        .baseUrl(BASE_URL)
        .client(client)
        .addCallAdapterFactory(ErrorsCallAdapterFactory(exceptionMappers))
        .addConverterFactory(
            json.asConverterFactory("application/json".toMediaType())
        )
        .build()
}
```

Výpis kódu 5.1 Ukázka inicializace Retrofit objektů potřebných pro komunikaci s Toggl Track API.

Třída obsahuje několik metod, které slouží k vykazování hovorů. Za prvé, pomocí metody *getCallHistory* dokáže zjistit projekt a značky, které by měly být přiřazeny k výkazu na základě historie vykazování. Postup je podrobně popsán v následující sekci 5.5.

Dále třída umí připravit časový výkaz pro zadaný telefonní hovor, který je určen k exportování. Tento výkaz slouží k zobrazení rekapitulace, kterou uživatel vidí po zahájení procesu manuálního vykazování zvoleného telefonního hovoru. Třída také zajišťuje samotné exportování časového výkazu pomocí repozitáře *TimeEntryRepository*.

Třída obsahuje i část logiky pro automatické vykazování telefonních hovorů. K tomu slouží metody *exportLatestCall*, *startPhoneCallTimer* a *restartTimeEntry*.

Třída *ExportCallUseCase* spolu s implementací popsaných metod je zobrazena v ukázkách kódu 5.2 a 5.3.

Výkazy telefonních hovorů z aplikace mají několik společných vlastností, které je odlišují od ostatních výkazů. Všechny mají značku *CallTracker App* a značku *Phone number:* , která dále obsahuje telefonní číslo volajícího. Nastavení těchto parametrů je součástí byznys logiky aplikace a probíhá ve třídě *ExportCallUseCase*.

5.5 Algoritmus pro přiřazení projektu a úkolů k výkazu hovoru

Jedním z požadavků aplikace je automatické přiřazení projektu a úkolů k vykazovanému hovoru. Existuje několik možností, jak takové přiřazení provést. Jednou z nich je inspirovat se posledním výkazem k danému telefonnímu číslu. Toto řešení bylo zvoleno i v rámci aplikace – projekt i úkoly u vykazovaného hovoru jsou nastaveny podle posledního výkazu se stejným telefonním číslem. Výjimkou je situace, kdy dané telefonní číslo nebylo dosud vykázáno. V tomto případě se u výkazu použijí výchozí hodnoty nastavené v nastavení aplikace.

V praxi se s klientem pracuje dlouhou dobu na stejném projektu a často také na stejných úkolech. Proto dává zvolený způsob přiřazení smysl i z praktického hlediska.

Jelikož je možné výkazy upravovat z jiných Toggl Track aplikací, bylo nutné při implementaci vyřešit problém externích změn údajů u posledního výkazu. Z tohoto důvodu nebylo zvoleno řešení ukládat ke každému telefonnímu číslu historii vykazování lokálně. V takovém případě by bylo nutné synchronizovat externí změny a přepisovat lokální historii.

Pro získání historie bylo zvoleno využití Toggl Track API. V ukázce kódu 5.4 je vidět implementace metody *getCallHistory*. Tato funkce získává z Toggl Track API všechny výkazy za poslední 2 měsíce a pokusí se najít první výkaz, který odpovídá danému telefonnímu číslu. Výkazy jsou seřazeny od nejmladšího. Pokud takový výkaz existuje, extrahuje z něj ID workspace, ID projektu a seznam značek. V opačném případě vrátí funkce hodnoty získané z nastavení aplikace.

5.6 Detekce zahájení a ukončení hovoru

Dalším ze softwarových požadavků aplikace bylo automatické vykazování telefonních hovorů. Tuto funkcionalitu se podařilo splnit pomocí třídy *CallStateListener*, jejíž možnosti využití byly popsány během analýzy v sekci 3.7.2. Aplikace tedy dokáže detekovat zahájení a ukončení hovorů.

Při detekci zahájení hovoru je uživatel upozorněn notifikací a pomocí API je v aplikaci Toggl Track spuštěn časovač. Pokud má uživatel během zahájení hovoru ve svém Toggl Tracku spuštěn jiný výkaz, je tento výkaz pozastaven. Aplikace získá a uloží jeho identifikátor, aby mohla po skončení hovoru zajistit jeho opětovné spuštění.

Po detekci ukončení hovoru je časovač zastaven a vytvořen nový výkaz. Jelikož mohl být běžící výkaz upraven z jiných aplikací, je z Toggl Tracku získána jeho aktuální verze, která je dále upravována. K výkazu jsou přidány značky reprezentující telefonní číslo a zdroj výkazu, tedy vlastní aplikace. Výkaz je dále upraven pomocí algoritmu pro přiřazení projektu a úkolů

```
class ExportCallUseCase @Inject constructor(
    private val callRepository: CallRepository,
    private val settingsRepository: SettingsRepository,
    private val projectRepository: ProjectRepository,
    private val timeEntryRepository: TimeEntryRepository,
) {

    private var stoppedTimeEntry: TimeEntry? = null

    fun getTagTextForPhoneNumber(phoneNumber: String): String =
        "Phone number: $phoneNumber"

    fun getAppTag(): String = "CallTracker App"

    private suspend fun prepareTimeEntry(call: Call): TimeEntry {
        val callHistory = getCallHistory(call.number)
        return TimeEntry(
            id = 0,
            workspaceId = callHistory.workspaceId,
            projectId = callHistory.projectId,
            billable = false,
            description = "Phone Call with ${call.name}",
            duration = call.duration,
            startTime = call.date,
            endTime = call.date.plusSeconds(call.duration.inWholeSeconds),
            tagNames = callHistory.tagNames
                .plus(getTagTextForPhoneNumber(call.number))
                .plus(getAppTag())
        )
    }

    private suspend fun restartTimeEntry(timeEntry: TimeEntry) {
        createTimeEntry( timeEntry.copy(
                startTime = OffsetDateTime.now(),
                duration = (-1).toDuration(DurationUnit.SECONDS),
                endTime = null
            )
        )
    }
```

Výpis kódu 5.2 První část ukázka implementace třídy ExportCallUseCase.

```
    suspend fun exportLatestCall() {
        val call = callRepository.getLatestCall()
            ?: throw NoSuchElementException("Call was not found in the phone")

        val callTimeEntry = stopRunningTimeEntry()
            ?: throw NoSuchElementException("No running time entry found")

        if (call.duration.inWholeSeconds == 0.toLong()) {
            timeEntryRepository.deleteTimeEntry(callTimeEntry)
        } else {
            val historyTimeEntry = prepareTimeEntry(call)
            val finalTimeEntry = callTimeEntry.copy(
                workspaceId = historyTimeEntry.workspaceId,
                projectId = callTimeEntry.projectId
                    ?: historyTimeEntry.projectId,
                tagNames = callTimeEntry.tagNames
                    .plus(historyTimeEntry.tagNames),
                description = historyTimeEntry.description,
            )
            updateTimeEntry(finalTimeEntry)
        }
        when (val stoppedTimeEntry = stoppedTimeEntry) {
            null -> {}
            else -> {
                restartTimeEntry(stoppedTimeEntry)
                this.stoppedTimeEntry = null
            }
        }
    }

    suspend fun startPhoneCallTimeEntry(): TimeEntry {
        // Ulozeni puvodniho beziciho vykazu
        stoppedTimeEntry = this.stopRunningTimeEntry()
        val defaultProject = settingsRepository.getDefaultProjectId().first()
        val projects = projectRepository.getAllProjects()
        val timeEntry = StartApiTimeEntry(
            workspaceId = projects.first{it.id == defaultProject}.workspaceId,
            start = OffsetDateTime.now().toString(),
            duration = -1,
            createdWith = "CallTracker",
            description = "Ongoing phone call..."
        )
        return timeEntryRepository.startTimer(timeEntry)
    }
}
```

Výpis kódu 5.3 Druhá část ukázky implementace třídy ExportCallUseCase.

```
data class CallExportHistory(
    val workspaceId: Long,
    val projectId: Long,
    val tagNames: List<String>
)

private suspend fun getCallHistory(phoneNumber: String): CallExportHistory {

    val timeEntries = timeEntryRepository.getLastTwoMonthTimeEntries()

    return try {
        val lastTimeEntry = timeEntries.first { timeEntry ->
            timeEntry.tagNames.contains(getTagTextForPhoneNumber(phoneNumber))
        }
        CallExportHistory(
            lastTimeEntry.workspaceId,
            lastTimeEntry.projectId!!,
            lastTimeEntry.tagNames,
        )
    } catch (exception: NoSuchElementException) {
        CallExportHistory(
            settingsRepository.getDefaultWorkspaceId().first(),
            settingsRepository.getDefaultProjectId().first(),
            listOf()
        )
    }
}
```

Výpis kódu 5.4 Ukázka metody pro získání parametrů výkazu ke konkrétnímu telefonnímu číslu na základě historie vykazovaní.

k výkazu hovoru (viz předchozí sekce 5.5). Finální verze výkazu je nahrána do aplikace Toggl Track.

Jelikož nebyl nalezen žádný spolehlivý způsob, jak přistupovat k údajům o probíhajícím hovoru, bylo nutné upravovat výkaz až po skončení daného hovoru. Informace o proběhlém hovoru lze získat až po jeho skončení z výpisu hovorů. Systém Android neukládá do výpisu provedený hovor okamžitě po jeho ukončení a stávalo se, že proběhlý hovor nebyl v době vykazování uložen. Proto bylo v aplikaci nastavena prodleva 0.5 sekundy po ukončení hovoru, než se začne hovor vykazovat.

5.6.1 Způsoby implementace

Během vývoje byly vyzkoušeny dva způsoby implementace. První způsob se ukázal jako nespolehlivý, a proto bylo nutné najít jiné řešení. Zde jsou oba způsoby představeny spolu s popisem problémů, které vznikaly.

Prvotní implementace pomocí Activity

V prvotní verzi byla třída *CallStateListener* zaregistrována uvnitř hlavní aktivity aplikace. Během testování bylo zjištěno, že v některých případech nebylo detekováno zahájení ani ukončení hovoru. Toto chování se dělo v případě, že aplikace byla delší dobu na pozadí.

Pravděpodobně zde byl problém s operačním systémem. Android má pod kontrolou životní cyklus všech spuštěných aplikací a aktivit. Pokud se aktivita přesune mimo obrazovku, může ji systém pro šetření výpočetních prostředků pozastavit. Jelikož bylo sledování telefonních hovorů spojeno s aktivitou aplikace, docházelo při pozastavení aktivity také k pozastavení ostatních procesů aktivity, což způsobovalo nespolehlivé detekování.

Implementace pomocí Foreground Service

Pro překonání problému s nespolehlivou detekcí hovorů byla využita Android komponenta Foreground service. Tato komponenta umožňuje aplikaci běžet v popředí i v případě, že je na pozadí, čímž umožňuje kontinuální sledování stavu telekomunikačního zařízení.[53]

Byla vytvořena třída *CallDetectService*, která dědí od třídy *Service*. Implementace této třídy je k vidění na výpisu kódu 5.5. Tato třída implementuje metody životního cyklu služby (třída *Service*), jako je *onStartCommand* a *onDestroy*, a zodpovídá za sledování stavu telekomunikačního zařízení. Z nastavení aplikace lze tlačítkem službu spustit. Při spuštění se zavolá metoda *onStartCommand*. Zobrazí se notifikaci, která informuje uživatele o tom, že aplikace sleduje telefonní hovory. Notifikace je nezbytná pro udržení služby v popředí a zabránění jejímu automatickému ukončení systémem Android. V metodě *onStartCommand* se zaregistruje *CallStateListener* pro sledování změn stavu hovorů. Kdykoli dojde k zahájení nebo ukončení hovoru, metoda *onCallStateChanged* zachytí tuto událost a spustí odpovídající akce.

Hlavní výhodou použití Foreground service je zajištění neustálé detekce zahájení nebo ukončení hovorů, a to i v případě, že je aplikace na pozadí, čímž se eliminuje problém s nespolehlivou detekcí hovorů. Foreground service také zobrazuje nenápadnou notifikaci, která informuje uživatele o tom, že aplikace sleduje telefonní hovory. To se hodí především pro větší transparentnost aplikace. Nevýhodou je mírné zvýšení spotřeby baterie a dopad na výkon zařízení, protože služba běží neustále. Implementace pomocí Foreground service se ukázala jako dobré řešení pro vyřešení problémů s nespolehlivou detekcí hovorů v případě, že je aplikace na pozadí.

5.7 Android oprávnění

Pro některé své funkcionality vyžaduje aplikace přidělení oprávnění. V rámci vývoje bylo nutné implementovat metody, pomocí kterých lze o oprávnění požádat. Také bylo nutné ošetřit odmítnutí

```
@AndroidEntryPoint
class CallDetectService: Service() {
    private val serviceId = 2000
    private val callbackExecutor = Executors.newFixedThreadPool(2)

    @Inject lateinit var callStateListener: CallStateListener
    @Inject lateinit var notificationHelper: NotificationHelper
    @Inject lateinit var permissionManager: PermissionManager

    private lateinit var telephonyManager: TelephonyManager

    override fun onCreate() {
        super.onCreate()

        val intent = Intent(this, CallDetectService::class.java)
        startForegroundService(intent)

        val permission = Manifest.permission.READ_CALL_LOG
        if (permissionManager.checkPermission(permission)) {
            telephonyManager = getSystemService(Context.TELEPHONY_SERVICE)
                    as TelephonyManager
            telephonyManager.registerTelephonyCallback(callbackExecutor,
                callStateListener)
        }
    }


    override fun onStartCommand(int: Intent?, flags: Int, startId: Int): Int {
        ServiceCompat.startForeground(
            this,
            serviceId,
            notificationHelper.getCallDetectServiceNotification(),
            FOREGROUND_SERVICE_TYPE_DATA_SYNC)
        return START_STICKY
    }


    override fun onDestroy() {
        super.onDestroy()
        telephonyManager.unregisterTelephonyCallback(callStateListener)
    }

    override fun onBind(intent: Intent?): IBinder? {
        return null
    }
}
```

Výpis kódu 5.5 Ukázka implementace komponenty Foreground Service pro detekování změn v telekomunikačním zařízení telefonu.

oprávnění.

5.7.1 Seznam potřebných oprávnění

Během implementace bylo postupně využito několik systémových prostředků, které jsou chráněny. K jejich používání je potřeba získat oprávnění. Pro transparentnost zde uvádím všechna potřebná oprávnění a stručně popisuji jejich využití v aplikaci.

- **INTERNET**: Oprávnění, které umožňuje aplikaci přistupovat k internetu.
- **FOREGROUND SERVICE**: Toto oprávnění umožňuje aplikaci spouštět popřední služby. Tyto služby nejsou spuštěné na obrazovce, takže je uživatel přímo nevidí, ale přesto běží. Z tohoto důvodu Android toto oprávnění vyžaduje. V aplikaci se využívá pro spuštění služby, která zajišťuje automatické vykazování hovorů.
- **FOREGROUND SERVICE DATA SYNC**: Toto oprávnění oznamuje uživateli, že popřední služba bude synchronizovat data. Aplikace, která spouští popřední služby, musí pomocí tohoto typu oprávnění definovat, k čemu popřední služby využívá.

Oprávnění uvedená výše jsou přidělena při instalaci aplikace (*install-time oprávnění*), a proto není nutné řešit jejich přidělení za běhu aplikace.

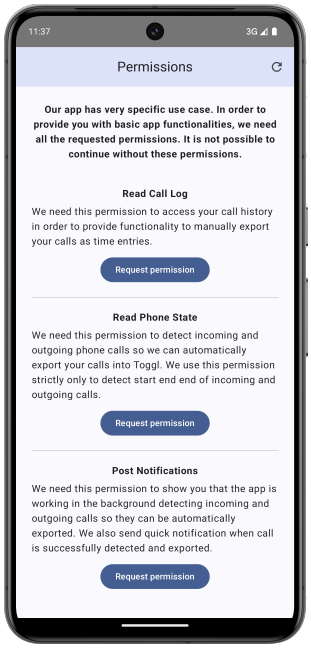
Naopak další tři oprávnění uvedené níže jsou oprávnění pro běh aplikace (*runtime oprávnění*). O tato oprávnění je nutné požádat až při běhu aplikace. Při implementaci je potřeba mít na paměti možnost jejich odmítnutí nebo odebrání kdykoliv v nastavení aplikace.

- **READ CALL LOG**: Toto oprávnění umožňuje číst historii hovorů. V aplikaci je nezbytné pro zobrazení seznamu hovorů a jejich manuální vykazování. Je také potřebné pro automatické vykazování hovorů, které získává telefonní čísla a jména kontaktů. Aplikace přistupuje pouze k telefonním číslům, jménům kontaktů, typům hovorů (odchozí, příchozí, zmeškaný), datům a dobám trvání.
- **READ PHONE STATE**: Toto oprávnění umožňuje aplikaci číst stav telekomunikačního zařízení. Aplikace ho využívá pouze k detekci zahájení a ukončení telefonních hovorů, což je nutné pro automatické vykazování hovorů.
- **POST NOTIFICATIONS**: Oprávnění nutné pro zobrazování notifikací. Aplikace zobrazuje dočasné notifikace při zahájení hovoru a po úspěšném dokončení automatického vykazování hovoru. Dále zobrazuje trvalou notifikaci, když služba pro automatické vykazování běží.

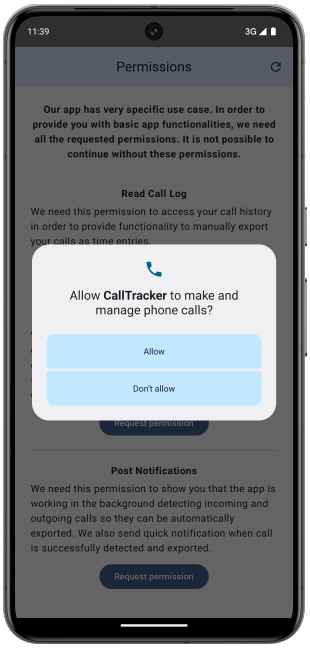
5.7.2 Kontrola oprávnění a implementace žádostí

Z popisu oprávnění vyplývá, že bez těchto oprávnění nelze používat základní funkcionality aplikace. Navzdory oficiálním doporučením bylo rozhodnuto, že uživateli bude aplikace zcela omezena, dokud všechna potřebná oprávnění nepřidělí. Pokud některá oprávnění chybí, uživatel je přesměrován na obrazovku s vysvětlením, proč aplikace nemůže pokračovat ve fungování. Na této obrazovce může uživatel oprávnění udělit.

Na obrázku 5.2 lze vidět obrazovku, která se zobrazí, pokud není některé z oprávnění uděleno. Na obrázku 5.3 je vidět žádost, která se objeví po stisknutí tlačítka *Grant* z předchozí obrazovky. Na obrázku 5.4 je vidět aplikace ve stavu, kdy uživatel udělil již dvě oprávnění. Na posledním obrázku 5.5 je vidět stav po odmítnutí žádosti o oprávnění. Systém Android nedovolí znovu žádat, takže je uživatel nasměrován do nastavení aplikace, kde může oprávnění přidělit manuálně.



Obrázek 5.2 Obrazovka aplikace, která se zobrazí, pokud chybí některé z runtime oprávnění.



Obrázek 5.3 Obrazovka, na které je vidět dialog se žádostí o udělení oprávnění.

5.8 Implementace zabezpečení dat

Jelikož aplikace ukládá citlivá data, jako je API klíč, je nutné zajistit, aby tyto informace nebyly v případě odcizení zařízení nebo kompromitace aplikace přístupné neoprávněným osobám.

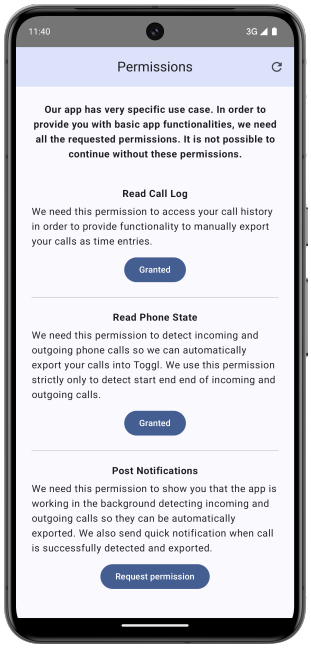
Aplikace využívá knihovnu DataStore, která byla popsána v sekci 4.4.1, pro ukládání různých konfiguračních dat, nastavení i API klíče. DataStore sám o sobě neposkytuje šifrování, proto bylo nutné implementovat dodatečnou vrstvu zabezpečení. Citlivá data jsou před uložením šifrována pomocí knihovny Keystore (viz sekce 4.4.2). Pomocí této knihovny se při prvním spuštění aplikace vygeneruje tajný klíč, který následně knihovna bezpečně uloží. Tímto klíčem je možné citlivá data šifrovat a dešifrovat.

Než je uživatelem zadaný API klíč uložen do DataStore, je nejprve zašifrován. Pomocí knihovny Keystore je před samotným šifrováním bezpečně získán tajný klíč, který se použije k vytvoření šifrovací funkce. Tato funkce je nastavena na symetrický šifrovací algoritmus AES v režimu CBC s PKCS7 Paddingem. Při zašifrování vrací šifrovací funkce použitý inicializační vektor. Ten je spolu se zašifrovaným API klíčem uložen do DataStore úložiště.

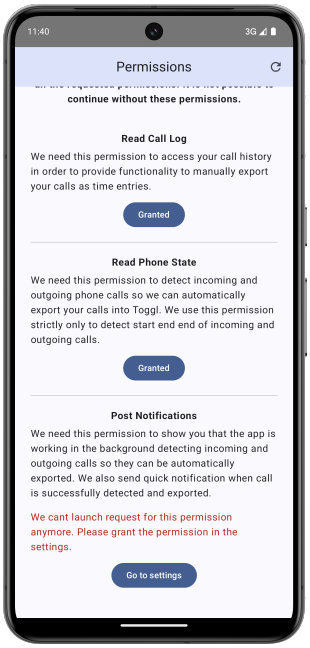
Při čtení zašifrovaného API klíče z DataStore jsou zašifrovaná data pomocí stejného šifrovacího algoritmu, tajného klíče z KeyStore a inicializačního vektoru uloženého v DataStore úspěšně dešifrována.

Na výpisu kódu 5.6 je vidět implementace třídy *KeystoreHelper*, která poskytuje výše uvedené funkce pro generování, ukládání a získávání klíče, a také pro šifrování a dešifrování dat.

Implementace šifrování dat zajišťuje, že i když útočník získá přístup k uloženým datům aplikace, budou pro něj tato data bezcenná, neboť budou zašifrována. Tímto způsobem chrání aplikace citlivé informace uživatele a splňuje minimální bezpečnostní požadavky.



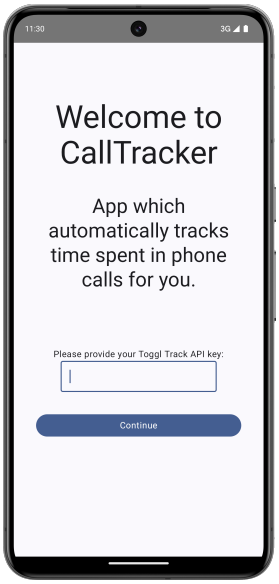
■ **Obrázek 5.4** Obrazovka aplikace po tom co uživatel přidělil dvě ze tří oprávnění.



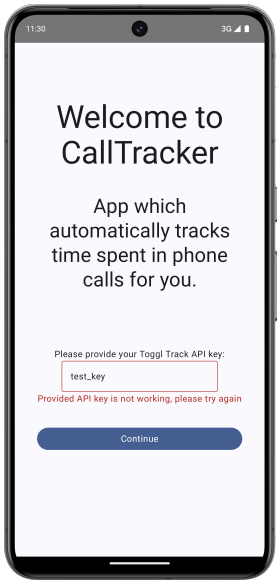
■ **Obrázek 5.5** Obrazovka aplikace po tom, co uživatel odmítl udělit oprávnění. Uživatel je vyzván k přidělení chybějícího oprávnění v nastavení aplikace.

```
@Singleton
class KeystoreHelper @Inject constructor(
    private val keyStore: KeyStore
) {
    private val KEY_ALIAS = "0010"
    private val cipherText = "AES/CBC/PKCS7Padding"
    init {
        keyStore.load(null)
        if (!keyStore.containsAlias(KEY_ALIAS)) generateKey()
    }

    private fun getKey(): KeyStore.SecretKeyEntry {
        return keyStore.getEntry(KEY_ALIAS, null) as KeyStore.SecretKeyEntry
    }

    fun encryptData(data: String): Pair<ByteArray, ByteArray> {
        val cipher = Cipher.getInstance(cipherText)
        val key = getKey()
        cipher.init(Cipher.ENCRYPT_MODE, key.secretKey)
        val encrypted = cipher.doFinal(data.encodeToByteArray())
        return Pair(encrypted, cipher.iv)
    }

    fun decryptData(encryptedData: ByteArray, iv: ByteArray): String {
        val cipher = Cipher.getInstance(cipherText)
        val keyEntry = getKey()
        cipher.init(Cipher.DECRYPT_MODE, keyEntry.secretKey,
            IvParameterSpec(iv))
        val decryptedText = cipher.doFinal(encryptedData)
        return decryptedText.toString(Charsets.UTF_8)
    }

    private fun generateKey() {
        val kg: KeyGenerator = KeyGenerator.getInstance(
            KeyProperties.KEY_ALGORITHM_AES, "AndroidKeyStore")
        val parameterSpec: KeyGenParameterSpec = KeyGenParameterSpec.Builder(
            KEY_ALIAS,
            KeyProperties.PURPOSE_ENCRYPT or KeyProperties.PURPOSE_DECRYPT
        ).run {
            setBlockModes(KeyProperties.BLOCK_MODE_CBC)
            setEncryptionPaddings(KeyProperties.ENCRYPTION_PADDING_PKCS7)
            setKeySize(128)
            build()
        }
        kg.init(parameterSpec)
        kg.generateKey()
    }
}
```

Výpis kódu 5.6 Implementace třídy KeystoreHelper, která slouží pro šifrování a dešifrování dat pomocí klíče, který je spravován knihovnou Keystore.



Obrázek 5.6 Přihlašovací obrazovka aplikace.



Obrázek 5.7 Přihlašovací obrazovka po pokusu pokračovat s nefunkčním API klíčem.

5.9 Realizace navržených obrazovek

V této části bude popsána výsledná implementace hlavních obrazovek aplikace. Především bude zdůrazněno srovnání mezi původním návrhem pomocí drátěných modelů (viz sekce 4.8) a reálnou podobou obrazovek. Dále budou zmíněny důvody případného odchýlení od návrhu.

5.9.1 Úvodní obrazovka

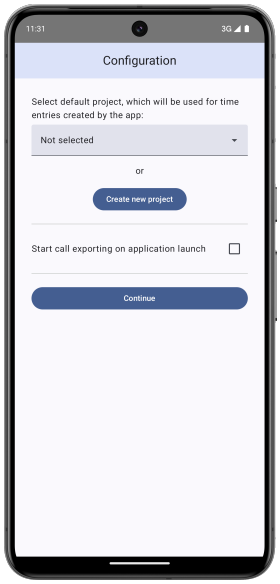
Úvodní obrazovka slouží k propojení s aplikací Toggl Track. Výslednou implementaci si lze prohlédnout na obrázku 5.6. Obrazovka se v podstatě nijak neliší od původního návrhu. Pokud se uživatel pokusí pokračovat s nefunkčním API klíčem, je upozorněn a vyzván k opravě. Tento stav je vidět na obrázku 5.7.

5.9.2 Obrazovka *First Configuration*

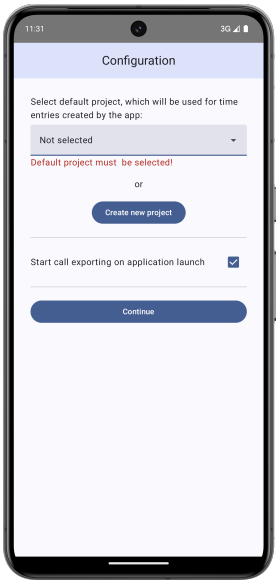
Obrazovka *First Configuration* nebyla zmíněna v návrhu, ale během implementační fáze vzešla nutnost ji přidat. Jejím hlavním účelem je nastavení potřebných parametrů před začátkem plného používání aplikace. Umožňuje nastavit výchozí Toggl Track projekt a zapnout nebo vypnout spuštění automatického vykazování hovorů po zapnutí aplikace. Vzhled této obrazovky je k vidění na obrázcích 5.8 a 5.9.

5.9.3 Obrazovka *Calls*

Obrazovka *Calls* zobrazuje seznam telefonních hovorů. Oproti návrhu bylo provedeno několik změn. Bylo přidáno tlačítko pro manuální synchronizaci dat, pomocí kterého lze na obrazovce obnovit data. Dále byl u každého telefonního hovoru přidán text zobrazující typ hovoru (odchozí nebo příchozí). Podobu obrazovky si lze prohlédnout na obrázku 5.13.



Obrázek 5.8 Obrazovka prvotního nastavení aplikace, která se uživateli zobrazí po úspěšném přihlášení.



Obrázek 5.9 Obrazovka prvotního nastavení po pokusu pokračovat bez vybraného výchozího projektu.

5.9.4 Obrazovka *Call Export*

Obrazovka *Call Export* umožňuje manuální exportování zvoleného telefonního hovoru ze seznamu na obrazovce *Calls*. Vzhled obrazovky je vidět na obrázcích 5.10, 5.11 a 5.12. Oproti návrhu přibyly pouze další informace o exportovaném hovoru – datum a trvání. Dále bylo přidáno tlačítko pro úpravu projektu, což je nutnost, která byla při návrhu opomenuta.

5.9.5 Obrazovka *Time Entries*

Obrazovka *Time Entries* zobrazuje seznam časových výkazů vykázaných pomocí aplikace. Oproti návrhu přibylo tlačítko pro manuální synchronizaci, stejně jako na obrazovce *Calls*. U časového výkazu není zobrazována doba trvání, jelikož ji lze vyčíst z dostupných informací na obrazovce. Vzhled obrazovky si lze prohlédnout na obrázku 5.15.

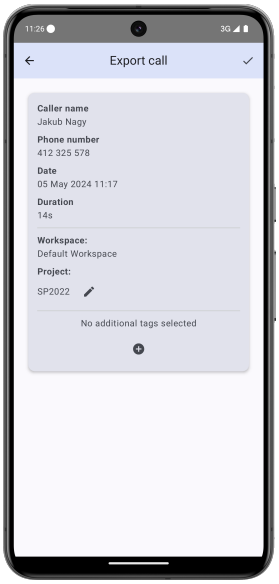
5.9.6 Obrazovka *Settings*

Obrazovka *Settings* prošla největšími změnami oproti návrhu. Výsledný vzhled obrazovky je vidět na obrázku 5.14.

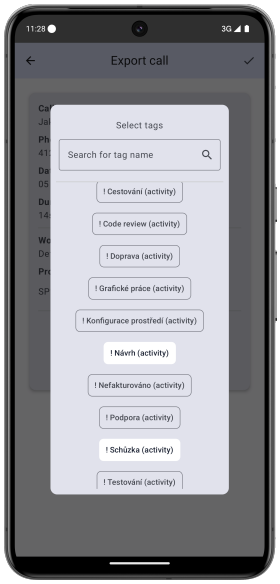
Místo pole pro přepsání API klíče bylo přidáno tlačítko pro odhlášení, které uživatele přesměruje na přihlašovací obrazovku a smaže uložený klíč. Ruční přepisování klíče s sebou přinášelo komplikace v podobě kontroly funkčnosti a nutnosti při změně upravit i výchozí projekt. Tento proces přímo kopíruje proces přihlášení, proto bylo místo textového pole zvoleno tlačítko pro odhlášení.

Přibyla sekce umožňující měnit nastavení pro automatické vykazování hovorů. Pomocí přepínače je možné automatické vykazování pozastavit nebo spustit. Dále je zde možnost povolit nebo zakázat spuštění této funkcionality při startu aplikace.

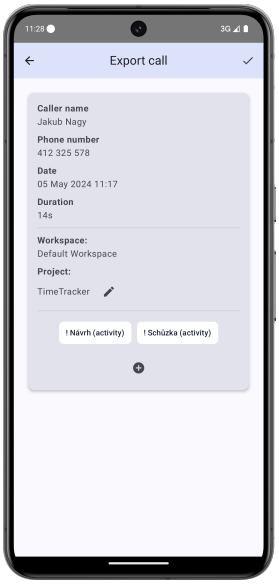
Byla odebrána sekce pro nastavení Toggl Track workspace, jelikož lze tento parametr určit z vybraného projektu. Výběr výchozího projektu byl rozšířen o možnost vytvoření zcela nového projektu, který je následně automaticky zvolen jako výchozí.



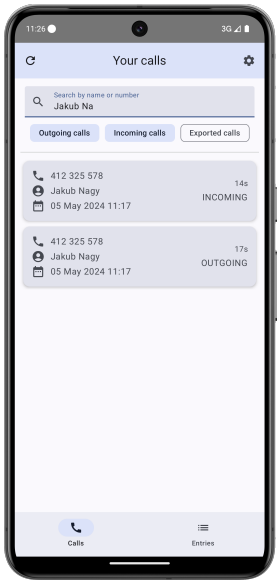
Obrázek 5.10 Obrazovka vykazování zvoleného telefonního hovoru.



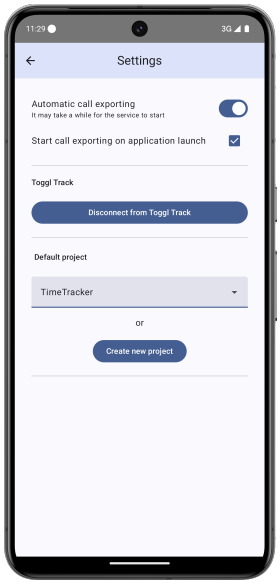
Obrázek 5.11 Obrazovka vykazování zvoleného telefonního hovoru po stisknutí tlačítko pro přidání značek.



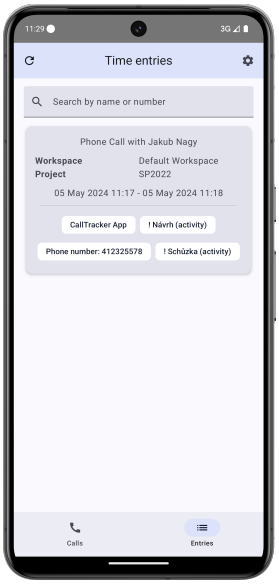
Obrázek 5.12 Obrazovka vykazování zvoleného telefonního hovoru po provedení změny projektu a přidání dvou značek.



Obrázek 5.13 Obrazovka zobrazující seznam telefonních hovorů. Na obrazovce je také vidět ukázka filtrování podle jména.



Obrázek 5.14 Obrazovka nastavení aplikace.



Obrázek 5.15 Obrazovka zobrazující seznam vykázaných telefonních hovorů.

5.9.7 Obrazovka *Time Entry Detail*

Obrazovka *Time Entry Detail* slouží k zobrazení a úpravě konkrétního časového výkazu hovoru. Na obrázcích 5.16, 5.17, 5.18 a 5.19 lze vidět obrazovku v různých stavech. Oproti návrhu bylo pozměněno rozložení jednotlivých atributů výkazů a nebyla implementována možnost změnit workspace. Jelikož je projekt spojen se svým workspace, byla ponechána možnost měnit pouze projekt. Tlačítko pro potvrzení se zobrazí pouze v případě, že je provedena změna některého z atributů.

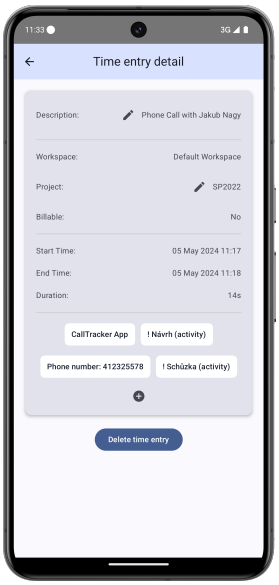
5.10 Vydání aplikace

Vydání aplikace je proces, při kterém se hotová a funkční aplikace zpřístupní uživatelům. Tento proces zahrnuje přípravu aplikace pro distribuci, výběr distribuční platformy a finální zveřejnění aplikace. V této sekci budou postupně popsány všechny aktivity, které byly v rámci procesu vydání vyvinuté aplikace provedeny. Budou také zmíněny zvolené distribuční platformy spolu s problémy, které se při jejich volbě objevily. Na závěr bude celý proces shrnut. Při vydávání bylo částečně postupováno podle oficiálního návodu na stránkách Androidu [54].

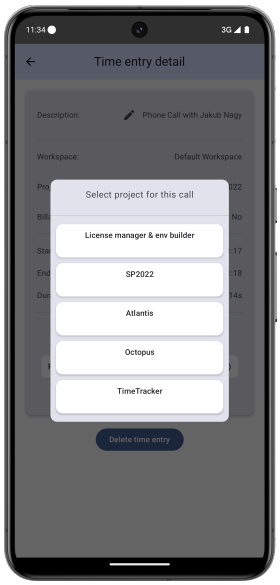
5.10.1 Vytvoření ikony

Každá aplikace má svou ikonu. Tato ikona je vidět například při spouštění aplikace nebo v seznamu instalovaných aplikací.

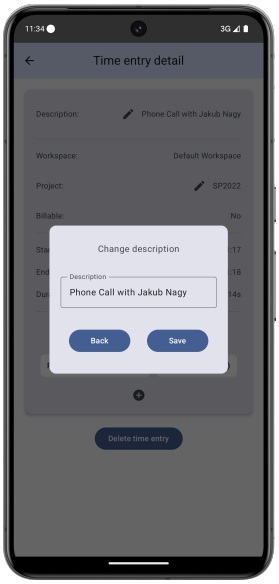
Ikona vyvinuté aplikace byla vytvořena nástrojem Image Asset Studio, díky kterému je možné na základě dodaného obrázku vytvořit sadu obrázků s různými velikostmi. Tyto obrázky se využívají pro ikonu aplikace při spuštění, ikonu na ploše, ikonu v seznamu aplikací nebo na distribučních platformách.



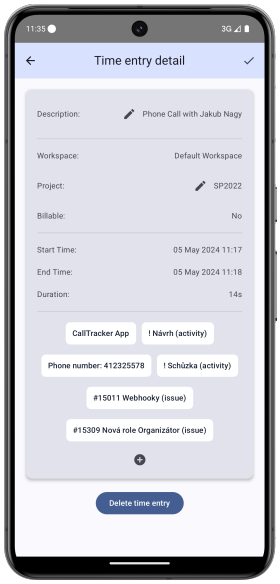
Obrázek 5.16 Obrazovka zobrazující detail výkazu telefonního hovoru s možností provádění úprav.



Obrázek 5.17 Obrazovka zobrazující detail výkazu telefonního hovoru po zvolení úpravy přiřazeného projektu.



Obrázek 5.18 Obrazovka zobrazující detail výkazu telefonního hovoru po zvolení úpravy popisu výkazu.



Obrázek 5.19 Obrazovka zobrazující upravený výkaz telefonního hovoru se změněným popisem a přidanými značkami.

Obrázek, ze kterého byly ikony vytvořeny, byl vygenerován nástrojem DeepAI [55]. Tento nástroj dokáže na základě textového vstupu generovat obrázky. Byl vybrán styl „AI Logo Generator“ a textový vstup „telephony receiver and time“.

5.10.2 Konfigurace

Před sestavením verze určené k zveřejnění je potřeba v aplikaci nastavit několik parametrů a vlastností. Je potřeba zvolit ID aplikace, které aplikaci reprezentuje po celou dobu jejího života. V rámci vyvíjené aplikace bylo zvoleno „cz.cvut.fit.calltracker“. Také je potřeba zkontrolovat a případně nastavit název aplikace a její verzi. V aplikaci byl zvolen název „Call Tracker“, a první verze byla označena jako „1.0.0“. Vytvořené ikony z předešlého kroku byly nastaveny jako ikony aplikace. Dále je vhodné zakázat debugovací mód, odstranit z kódu veškeré logování a zapnout optimalizace, které se použijí při sestavení aplikace. Všechny tyto kroky byly v aplikaci provedeny.

5.10.3 Podepsání

Každá aplikace, kterou chceme na Android zařízení spouštět, musí být kryptograficky podepsána. Podepsání lze provést přímo ve vývojovém prostředí Android Studio. Toto prostředí lze využít i pro vygenerování kryptografického klíče, kterým se sestavená aplikace podepisuje. Aplikace byla v Android Studiu podepsána vygenerovaným klíčem.

5.10.4 Vydání

Android aplikace lze vydávat několika způsoby. Obvyklým způsobem je zveřejnění aplikace na distribuční platformě, ze které mohou uživatelé aplikaci stahovat. Méně obvyklým způsobem je nahrání aplikace na vlastní webové stránky a umožnění její stahování. Posledním způsobem je přímé zaslání aplikace uživatelům.

Vlastní distribuce

Tento způsob distribuce je jednodušší než využití distribučních platforem. Jediné, co je potřeba udělat, je umožnit uživatelům sestavenou a podepsanou aplikaci získat. Jednou z možností je nahrát soubor na webové stránky a umožnit jeho stahování. Další možností je přímé zaslání například pomocí elektronické pošty. Uživatelé soubor spustí a aplikaci nainstalují. Někdy je nutné v systému povolit instalování aplikací z neznámých zdrojů.

Tento způsob je celkově jednodušší a umožňuje naprostou svobodu, nemusí se totiž dodržovat žádné podmínky prostředníka, který distribuci jinak zajišťuje. Na druhou stranu mohou aplikace, které nejsou distribuovány pomocí distribučních platforem, působit méně důvěryhodně. Propagace aplikací je náročnější a dostat aplikaci do povědomí potenciálních uživatelů je obtížné.

Autor práce momentálně nespravuje žádné webové stránky, kam by bylo možné instalační soubor ke stažení umístit. Z tohoto důvodu budou dále prozkoumány možnosti vydání aplikace na distribučních platformách. Přesto je v případě potřeby možné přímé zaslání aplikace uživatelům.

Google Play

Google Play je nejznámější distribuční platforma, která je předinstalována na většině zařízení s Androidem. Pokud chceme aplikaci sdílet přes Google Play, je potřeba vytvořit si vývojářský účet, jehož vytvoření je zpoplatněno a stojí 25 amerických dolarů. Fakulta podle zjištěných informací nedisponuje žádným společným vývojářským účtem, přes který by bylo možné aplikaci vydat.

Dalším problémem je zařazení oprávnění **READ CALL LOG** do skupiny nebezpečných oprávnění. Podle podmínek Google Play je toto oprávnění povoleno pouze určitým aplikací, a

pokud je využíváno, musí být aplikace podrobena důkladné analýze a kontrole. Pokud aplikace nesplňuje podmínky, je její vydání zakázáno. Oficiálně smí toto oprávnění používat pouze aplikace, které jsou uživatelem nastaveny jako výchozí pro správu hovorů. Aby mohla být aplikace zvolena jako výchozí, musí implementovat všechny funkcionality, které má nativní telefonní aplikace. To není případ Call Trackeru. K tomuto oficiálnímu povolení přidává Google řadu výjimek, které mohou ale nemusí být uděleny. V současné době žádná z nich přímo nesouvisí s funkcionalitou, kterou Call Tracker nabízí. Jedinou blízkou výjimkou, kterou by aplikace mohla využít, je „Device automation“. Ta se dá uplatnit u aplikací, které pro uživatele automatizují opakující se akce ve více oblastech operačního systému. Udělení výjimky je spojeno s žádáním a schvalováním, což může zabrat mnoho času. [56]

Z důvodu pozdního dokončení implementace a možného dlouhého procesu schvalování na Google Play se autor momentálně rozhodl od vydání aplikace na této distribuční platformě ustoupit. Pravděpodobná nízká šance na úspěšné zveřejnění a nutnost zaplatit poplatek za publikování taktéž hrály roli v tomto rozhodnutí. Po odevzdání bakalářské práce se autor plánuje znovu zaměřit na vydání aplikace na Google Play. V rámci tohoto procesu bude nutné vyřešit problém s povolením a ověřit domněnky o problémech se schvalováním.

Aptoide

Aptoide je alternativní obchod s aplikacemi pro operační systém Android. Na rozdíl od dominantního Google Play, který je předinstalován na většině zařízení s Androidem, je Aptoide nezávislý obchod třetí strany, který je nutné manuálně nainstalovat. Díky absenci kontroly ze strany Google umožňuje větší svobodu v typu dostupných aplikací.

Aptoide se jevil jako ideální platforma pro vydání prvotní verze aplikace Call Tracker. Při zahájení procesu vydání bylo ale zjištěno, že zprovoznění účtu, přes který je možné aplikace vydávat, je zpoplatněno ročním poplatkem ve výši 69 amerických dolarů. Z tohoto důvodu nebyl Aptoide pro distribuci aplikace využit. [57]

Amazon Appstore

Amazon Appstore je další distribuční platforma, která je využívána především k distribuci aplikací pro zařízení s operačním systémem Fire OS. Tento operační systém, spravovaný společností Amazon, je součástí například tabletů Amazon Fire a přehrávačů digitálních médií Amazon Fire TV. Amazon Appstore lze instalovat i do zařízení se systémem Android, a tak ho lze využít i k distribuci Android aplikací. Nahrání aplikace na tuto platformu je bezplatné. [58]

Call Tracker byl na platformu nahrán. Po nahrání byla aplikace podrobena testování a kontrole ze strany Amazonu. Výsledkem bylo schválení a zveřejnění aplikace na platformě. Aplikace je nyní dostupná ke stažení.

Před stažením je nejdříve nutné nainstalovat aplikaci Amazon Appstore, která je dostupná na oficiálních stránkách Amazonu. Po nainstalování a spuštění je potřeba se přihlásit nebo provést registraci. Následně lze ve vyhledávání pod heslem *Call Tracker* nalézt vydanou aplikaci a nainstalovat ji. Pravděpodobně bude před oběma staženími nutné v systému povolit stahování z cizích zdrojů.

5.10.5 Shrnutí

Implementovaná aplikace byla připravena k vydání. Nejdříve byly nastaveny potřebné parametry a vytvořena ikona aplikace. Poté byl vytvořen podepsaný instalační soubor. Následně bylo postupně vyzkoušeno několik distribučních platforem – Google Play, Aptoide a Amazon Appstore.

Google Play je jedna z nejznámějších a nejpoužívanějších distribučních platforem. Z tohoto důvodu je nahrávání aplikací nejvíce kontrolováno a aplikace musí pro zveřejnění splňovat řadu

požadavků. Tato služba je navíc zpoplatněna. Kvůli přístupu k výpisu telefonních hovorů vyvíjená aplikace přísné požadavky nesplňuje, a proto nebyla k distribuci aplikace využita.

Aptoide je alternativou k Google Play. Podmínky a požadavky pro zveřejněné aplikace nejsou tak přísné jako u konkurenční služby. Z tohoto důvodu je pravděpodobné, že zveřejnění vyvinuté aplikace by na této platformě bylo schváleno. Na druhou stranu je i tato platforma zpoplatněna a cena je ještě vyšší než v případě Google Play. Z tohoto důvodu nebyla ani tato platforma využita.

Poslední z vyzkoušených možností je Amazon Appstore. Tato služba se využívá především k distribuci aplikací určených pro zařízení od společnosti Amazon. Lze ji ale využít i k distribuci aplikací pro Android zařízení. Zveřejnění aplikací na této platformě je bezplatné. Aplikace zde byla nahrána, zkontrolována a na závěr také úspěšně zveřejněna.

Vyvinutá aplikace byla úspěšně vydána a v současné době si ji uživatelé mohou zdarma stáhnout přes distribuční platformu Amazon Appstore.

Kapitola 6
Testování

Testování aplikace je jedním z důležitých kroků v jejím vývojovém cyklu. V rámci vyvíjené aplikace bylo provedeno uživatelské testování.

6.1 Uživatelské testování

Uživatelské testování je jednou z kvalitativních metod pro ověření použitelnosti produktu. Probíhá postupně s několika subjekty, kterým jsou po jednom zadávány úkoly, které plní. Plnění je sledováno a analyzováno. Analýzou je shromážděna zpětná vazba, která pomáhá identifikovat oblasti pro zlepšení. Na jejím základě jsou navrhnuty konkrétní úpravy, které zajistí, aby produkt splňoval potřeby uživatelů. [59]

6.1.1 Návrh scénářů

Při uživatelském testování plní testovací uživatelé zadané úkoly. Tyto úkoly by měly odpovídat reálným úkonům, které budou budoucí uživatelé v aplikaci vykonávat. Při jejich stanovení je vhodné zaměřit se na problémové a složitější části aplikace. Jednotlivé úkoly se běžně popisují pomocí tzv. scénářů. Scénář není seznamem kroků, které uživateli říkají, co přesně má udělat. Naopak by se měl scénář skládat pouze z cíle, který je potřeba splnit, a případně z informací potřebných pro jeho splnění. Účelem testování je analyzovat použitelnost aplikace, tedy jestli uživatel dokáže aplikaci sám používat.

Jelikož testování probíhá v závěrečné fázi vývoje a s verzí aplikace, která by měla být funkční, bylo jeho cílem odhalit nedostatky při používání hlavních funkcionalit aplikace. Při definici scénářů se vycházelo především z případů užití (viz sekce 3.6) a softwarových požadavků (viz sekce 3.5). Scénáře byly navrženy tak, aby spolu pokryly všechny případy užití i požadavky, a byla tak otestována jejich funkčnost.

Níže je uveden seznam testovacích scénářů, které byly během testování použity:

1. Představte si, že jste aplikaci právě nainstalovali. Spusťte ji a uveďte ji do použitelného stavu. Pokud nemáte Toggl Track účet, budou vám dodány přihlašovací e-mail a heslo. Nutná nastavení nastavte podle své libosti.

2. Přejete si změnit přihlášený Toggl Track účet. Odhlaste se a přihlaste se pod účtem dodaným moderátorem. Tento účet se bude využívat v dalších úkolech. Projekt, na kterém nyní pracujete, se jmenuje „Caresa“. Také nechcete, aby se po spuštění aplikace ihned automaticky vykazovaly hovory.

3. Ve společnosti začínáte soustředit veškerou práci na projekt s názvem „Boroto“. Počítáte s tím, že se všechny vaše telefonní hovory budou týkat právě tohoto projektu. Upravte aplikaci tak, aby odrážela tuto skutečnost.

4. Začíná vaše pracovní doba a přejete si mít všechny své hovory vykázané. Nastavte aplikaci tak, aby nastavení odráželo tuto skutečnost.

5. Volá vám kolega (bude simulován telefonát). S kolegou řešíte projekt „Boroto“, konkrétně úkol „#1234“. Jelikož se jedná o pracovní hovor, chcete ho mít s těmito parametry vykázaný. Uveďte aplikaci do tohoto stavu.

6. Končí vaše pracovní doba a přestat vykazovat všechny telefonní hovory. Nastavte aplikaci tak, aby nastavení odráželo tuto skutečnost.

7. Po pracovní době vám telefonuje kolega, se kterým řešíte důležité záležitosti týkající se **analýzy** projektu „Boroto“. Jelikož se jedná o pracovní hovor, chcete ho mít s těmito parametry vykázaný. Postarejte se, aby byl hovor vykázaný.

8. Před tímto scénářem bude moderátorem opět zapnuto automatické vykazování hovorů. Volá vám kamarád ohledně plánů na dnešní večer. Bohužel jste po pracovní době zapomněli vypnout automatické vykazování. Zrušte vykázání tohoto hovoru.

9. Volá vám kolega (bude simulován telefonát). Od kolegy se dozvíte, že byl přeřazen z projektu „Boroto“ na práci na projektu „Caresa“. Zařiďte, aby byl tento hovor vykázán pod nový projekt. Zároveň si přejete, aby všechny další hovory s tímto kolegou byly automaticky vykazovány pod novým projektem.

6.1.2 Volba uživatelů

Při výběru testovacích subjektů je nutné zaměřit se na osoby, které co nejvíce reprezentují skutečné budoucí uživatele aplikace. Pokud budou vybráni zkušenější uživatelé, mohou přehlédnout nedostatky, se kterými se poté skuteční a méně zkušení uživatelé setkají. Naopak, pokud budou vybráni méně zkušení uživatelé, může to vést k provedení zbytečných změn, které nepovedou ke zlepšení použitelnosti pro reálné uživatele.

Vyvíjená aplikace není určena pro každého. Je určena pro lidi, kteří si vykazují svůj čas a ocenili by možnost vykazování telefonních hovorů. Proto byly testovací uživatelé vybráni z osob, které mají s vykazováním času alespoň minimální zkušenosti. Znalost aplikace Toggl Track je výhodou, ale není nutností. Pokud uživatel nemá Toggl Track účet, bude mu moderátorem testování poskytnut. Je nutné, aby měl uživatel zkušenost s používáním Android zařízení. Výhodou bude, pokud bude souhlasit s nainstalováním aplikace přímo do vlastního telefonu. Pokud ne, bude testování provedeno v emulátoru z Android Studia. Jelikož je aplikace v anglickém jazyce, je potřeba, aby testovací uživatelé rozuměli alespoň základům angličtiny.

Před samotným testováním bylo nutné stanovit počet testerů. Obvykle stačí, aby bylo dosaženo odhalení 85% problémů s použitelností, testovat s 5 až 7 uživateli. Přidáváním dalších se odhalují problémy, které již byly identifikovány předchozími testery. Z tohoto důvodu byl pro testování vyvinuté aplikace stanoven počet testerů na 5. [60]

Níže je krátké představení každého z testovacích subjektů:

- Testovací subjekt 1
  - Muž, 24 let, studuje na fakultě informačních technologií.
  - Má zkušenosti s vykazováním času.
  - Zná aplikaci Toggl Track.
- Testovací subjekt 2

  - Muž, 24 let, studuje na fakultě informačních technologií.
  - Má zkušenosti s vykazováním času.
  - Nezná aplikaci Toggl Track.
- Testovací subjekt 3
  - Žena, 24 let, studuje na fakultě informačních technologií a pracuje v oblasti IT.
  - Má zkušenosti s vykazováním času.
  - Zná aplikaci Toggl Track.
- Testovací subjekt 4
  - Muž, 24 let, pracuje v oblasti IT.
  - Má zkušenosti s vykazováním času.
  - Zná aplikaci Toggl Track.
- Testovací subjekt 5
  - Žena, 26 let, studentka fakulty agrobiologie, potravinových a přírodních zdrojů.
  - Má minimální zkušenosti s vykazováním času.
  - Nezná aplikaci Toggl Track.

6.1.3 Průběh testování

Před samotným testováním byl vytvořen Toggl Track účet, který simuluje účet vývojáře ve fiktivní společnosti, která pracuje na projektech **Boroto** a **Caresa**. V účtu bylo vytvořeno několik značek – **analýza**, **testování** a **úkol #1234**. Přihlašovací údaje k účtu jsou **dmecushhtpuirrywct@ckptr.com** a **abcd1234ABCD**. Tento účet je potřeba k simulování některých situací popsaných v testovacích scénářích.

Na začátku testování byla aplikace představena. Jelikož má aplikace složitější případ využití, byl na představení kladen větší důraz. Uživatelům byly popsány hlavní funkce aplikace – manuální vykázání hovoru, automatické vykázání hovoru a také princip automatického přiřazování projektů a úkolů. Dále jim byl vysvětlen způsob přihlášení přes Toggl Track a jelikož je Toggl Track součástí aplikace, byla uživatelům, pokud o to požádali, krátce představena i aplikace Toggl Track.

Každý uživatel byl na začátku testování požádán o stažení a nainstalování aplikace přes distribuční platformu Amazon Appstore, na které je aplikace vydána. Pokud uživatel nechtěl do svého zařízení tuto službu stahovat, byla mu aplikace distribuována přímo zasláním APK souboru. Pokud uživatel nechtěl používat své zařízení, byl využit emulátor z Android Studia. S každým byly postupně vykonány všechny testovací scénáře. Na závěr byl každý požádán o ohodnocení několika vlastností aplikace:

- **Užitečnost:** Zda aplikace splňuje očekávání uživatelů a zda jim skutečně pomáhá s úkoly, pro které byla navržena.
- **Uspořádání:** Zda je rozhraní aplikace logicky uspořádané a zda uživatelé snadno najdou to, co hledají.
- **Estetika:** Zda je design aplikace atraktivní a moderní a zda se uživatelům líbí.
- **Snadnost použití:** Zda je aplikace intuitivní a snadno se s ní pracuje i bez předchozích zkušeností.

Role moderátora jsem se zhostil já sám. Detailní popis průběhu testování s každým testovacím subjektem spolu s hodnocením aplikace je k dispozici v příloze **A – Detailní popis průběhu uživatelského testování**.

6.1.4 Vyhodnocení testů

Během některých testů se vyskytly problémy s aplikací Toggl Track. Jelikož se využíval stejný testovací účet pro všechny testovací subjekty, bylo několikrát zapomenuto účet po předešlém testování vymazat. Uživatelé byli upozorněni, vzniklá situace jim byla vysvětlena a moderátor data včas smazal, proto nebylo testování tímto problémem zásadně ovlivněno.

Objevila se také chyba při automatickém vykazování. Pokud byl výpis hovorů telefonu prázdný a aplikace se pokusila vykázat provedený hovor, vykazování selhalo. Příčinou bylo zpoždění mezi ukončením hovoru a jeho uložením do výpisu hovorů systémem Android. Aplikace přistoupila k výpisu hovorů příliš brzy, kdy v něm ještě provedený hovor nebyl uložený.

Většina uživatelů nebyla spokojena s nutností ruční aktualizace obrazovek s hovory a výkazy. Uživatelé byli kvůli neprovedení aktualizace často zmatení, protože se jim nezobrazovaly výsledky provedených akcí.

Dále jsou níže pro každý scénář popsány hlavní problémy, které se během jeho provádění objevovaly. Na závěr je uveden seznam identifikovaných problémů a jejich navržené řešení spolu se stavem implementace daného řešení v aplikaci.

Scénář 1

Během přihlašování byl největší problém nalézt v aplikaci Toggl Track sekci s API klíčem.

Někteří uživatelé na přihlašovací obrazovce klikali na klávesu Enter, aby klávesnici zavřeli. Tím ale do textu klíče přidávali znak odřádkování. Někteří také očekávali, že se klávesnice po kliknutí mimo textové pole zavře, což je běžné u většiny aplikací.

Scénář 2

Žádný z uživatelů neměl zásadní problém s odhlášením. Při přihlašování do nového Toggl Track účtu se objevovaly stejné problémy jako v případě prvního scénáře.

Scénář 3

Všichni uživatelé pochopili, že změnou výchozího projektu nastaví vykazování všech hovorů bez historie pod tento projekt. Někteří uživatelé si po opuštění obrazovky s nastavením nebyli jistí, zda se nastavení úspěšně uložilo.

Scénář 4

Nikdo z uživatelů neměl problém se spouštěním automatického vykazování hovorů. Ale žádný z uživatelů nepochopil zobrazenou notifikaci jako signalizaci toho, že je automatické vykazování hovorů zapnuté.

Scénář 5

Většina uživatelů nedokázala zjistit, že hovor byl již automaticky vykázán a vykázaný hovor manuálně opakovaně vykázala. Uživatelé se snažili změnit projekt a značky z obrazovky určené k manuálnímu vykázání hovoru. Opakované vykázání stejného hovoru zapříčinilo problémy se zjišťováním historie hovorů, jelikož se v seznamu výkazů nacházelo několik stejných hovorů, které měly různě nastavené projekty a značky. Po testování někteří uživatelé uvedli, že obrazovku s výkazy vůbec nebrali jako důležitou obrazovku, takže veškeré akce prováděly z obrazovky s hovory. Pravděpodobnou příčinou bylo nepochopení rozdílu mezi hovorem a výkazem.

Uživatelé, kteří správně začali upravovat existující záznam, měli problém se zavřením dialogu při přiřazování nových značek. Dialog byl téměř přes celou obrazovku a bylo obtížné kliknout mimo něj, aby se zavřel. Ve stejném dialogu se během úpravy objevily i značky telefonních čísel

z jiných výkazů, které ale nebylo možné zvolit. Jeden z uživatelů také nevěděl, že může zvolenou značku kliknutím odstranit.

Scénář 6

Nikdo z uživatelů neměl problém s vypnutím automatického vykazování hovorů.

Scénář 7

Uživatelům se většinou podařilo správně vykázat provedený hovor. Někteří uživatelé byli zmatení kvůli špatné funkčnosti filtrů na vykázané a nevykázané hovory a také kvůli nutnosti manuální aktualizace seznamu hovorů.

Scénář 8

Uživatelé měli stejně jako u pátého scénáře problém s použitím správné obrazovky. Většina se snažila výkaz smazat z obrazovky s hovory. Pokud se uživatelům podařilo přejít na obrazovku s výkazy, většinou zde správně vybrali daný výkaz, který následně smazali. Někteří uživatelé si stěžovali na chybějící upozornění po úspěšném smazání.

Scénář 9

Stejně jako u pátého scénáře uživatelé často nevěděli, že se daný hovor automaticky vykázal. Z tohoto důvodu daný hovor vykázali z obrazovky s hovory opakovaně. Někteří se snažili nastavit automaticky přiřazovaný projekt v nastavení aplikace, protože nevěděli, že aplikace provádí přiřazování na základě posledního výkazu hovoru se stejným kontaktem.

Seznam problémů

Na základě vyhodnocení testování je níže uveden seznam všech identifikovaných problémů, jejich plánované řešení a stav realizace daného řešení v aplikaci:

- **Nápověda pro nalezení Toggl Track API klíče**

  Popis: Uživatelé nevěděli, kde v aplikaci Toggl Track najít sekci s API klíčem.

  Řešení: Přidání nápovědy, kterou lze otevřít z přihlašovací obrazovky a která uživatelům vysvětlí, kde naleznou sekci s API klíčem.

  Stav řešení: Vyřešeno – Na přihlašovací obrazovku byl přidán text, na který lze kliknout. Po kliknutí se otevře nápověda, která krok po kroku vysvětlí, kde API klíč nalézt.

- **Zavírání klávesnice na přihlašovací obrazovce**

  Popis: Během zadávání API klíče na přihlašovací obrazovce nebylo jednoduché zavřít klávesnici, a kvůli tomu bylo skryté tlačítko pro pokračování. Mnoho uživatelů se snažilo kliknout mimo textové pole, aby tak klávesnici zavřeli.

  Řešení: Nastavit klávesnici na přihlašovací obrazovce tak, aby se po kliknutí mimo textové pole zavřela.

  Stav řešení: Vyřešeno

- **Potvrzovací tlačítko v nastavení**

  Popis: Někteří uživatelé si nebyli jistí, jestli se po stisknutí tlačítka pro návrat na obrazovce s nastavením jejich změny v nastavení provedly a uložily.

  Řešení: Přidání tlačítka potvrzení, které uživatele přesměruje na stejnou stránku jako tlačítko pro návrat, ale dodá jim pocit úspěchu.

Stav řešení: Vyřešeno

- **Zmatek mezi hovorem a výkazem**

  Popis: Uživatelé nepochopili rozdíl mezi obrazovkou s hovory a obrazovkou s výkazy.

  Řešení: Při prvním spuštění aplikace bude zobrazena obrazovka, která vysvětlí funkce obou obrazovek.

  Stav řešení: Vyřešeno – Do aplikace byla přidána úvodní obrazovka, která se objeví po prvním spuštění. Na této obrazovce jsou mimo jiné vysvětleny funkce obrazovky s hovory a obrazovky s výkazy. Na úvodní obrazovku se lze později vrátit z nastavení aplikace.

- **Vykazování již vykázaných hovorů**

  Popis: Uživatelé často opakovaně vykazovali již vykázané hovory.

  Řešení: Uživatel bude upozorněn, pokud se pokusí vykázat již vykázaný hovor.

  Stav řešení: Vyřešeno – Po zvolení již vykázaného hovoru se otevře dialog, který upozorňuje uživatele, že se chystá vykázat již vykázaný hovor. Uživatel se může vrátit nebo pokračovat ve vykazování.

- **Zmatek mezi filtrovacími možnostmi**

  Popis: Uživatelé nepochopili funkce filtrovacích možností na stránce s hovory, zejména filtrování exportovaných hovorů.

  Řešení: K možnosti filtrovat exportované hovory bude přidána další možnost pro filtrování neexportovaných hovorů. Pokud uživatel nezvolí žádné filtrování, budou zobrazeny všechny hovory. Pokud vybere jednu z možností, budou se zobrazovat pouze hovory s vybranou vlastností. Pokud vybere obě možnosti, zobrazí se všechny hovory.

  Stav řešení: Vyřešeno

- **Nutnost manuální aktualizace obrazovek**

  Popis: Uživatelé byli zmateni, že musejí manuálně aktualizovat obrazovku, když na ni byli přesměrováni z jiných obrazovek aplikace.

  Řešení: Při přesměrování z jiných obrazovek aplikace budou obrazovky automaticky aktualizovány.

  Stav řešení: Vyřešeno

- **Potvrzení o úspěšném manuálním vykázání hovoru**

  Popis: Uživatelé často nevěděli, jestli se manuální vykázání hovoru úspěšně provedlo. Kvůli tomu několikrát vykázali stejný hovor opakovaně.

  Řešení: Po úspěšném vykázání se na současné obrazovce zobrazí upozornění, které signalizuje, že se hovor úspěšně vykázal.

  Stav řešení: Vyřešeno

- **Změna přiřazeného projektu k budoucím výkazům ke konkrétnímu číslu**

  Popis: Uživatelé často netušili, že změnou vlastností posledního provedeného výkazu hovoru s konkrétním číslem provedou změnu přiřazení parametrů k budoucím hovorům se stejným číslem.

  Řešení: Při prvním spuštění se zobrazí uvítací obrazovka, která vysvětlí, jak funguje automatické přiřazování projektů a značek k hovorům.

  Stav řešení: Vyřešeno

- **Zavírání dialogu pro přidání značek**

  Popis: Dialog pro změnu značek bylo obtížné zavřít, protože byl téměř přes celou obrazovku.

  Řešení: Součástí dialogu bude tlačítko, kterým lze dialog zavřít.

  Stav řešení: Vyřešeno

- **Seznam značek v dialogu při úpravě výkazu**

  Popis: Mezi značkami v dialogu se objevily i značky reprezentující telefonní čísla z jiných výkazů. Na tyto značky nebylo možné kliknout.

  Řešení: V dialogu se nebudou zobrazovat značky telefonních čísel z jiných výkazů.

  Stav řešení: Vyřešeno

- **Odstranění přidaných značek při tvorbě nebo úpravě výkazu**

  Popis: Během tvorby nebo úpravy výkazu nebylo zjevné, že lze značku odstranit kliknutím na ni.

  Řešení: Ke všem zvoleným značkám bude přidána ikona křížku, která signalizuje, že je možné na ni kliknout a odstranit ji.

  Stav řešení: Vyřešeno

- **Chyba při automatickém vykazování**

  Popis: Několikrát se stalo, že aplikace nedokázala ve výpisu hovorů najít právě dokončený hovor. Pokud byl výpis před hovorem prázdný, vykazování zcela selhalo. Pokud se v něm nacházely hovory, aplikace místo provedeného hovoru vykázala hovor předchozí.

  Řešení: Zvýšení prodlevy mezi detekcí ukončení hovoru a spuštěním procesu vykazování z 0,5 sekundy na 3 sekundy, aby měl systém Android dostatek času na uložení ukončeného hovoru do výpisu hovorů a aplikace četla již upravený výpis hovorů.

  Stav: Vyřešeno

V tabulce 6.1 je uvedeno výsledné hodnocení aplikace všemi testovacími subjekty spolu s průměrným hodnocením každé z vlastností. Nejhorší hodnocení získala aplikace v oblasti jednoduchosti použití. To pravděpodobně souvisí se špatným popisem aplikace a složitým, specifickým případem použití. Naopak nejlepší hodnocení získala aplikace v oblasti estetiky, kde si uživatelé nejvíce chválili jednoduchost vzhledu.

	Užitečnost	**Uspořádání**	**Estetika**	**Snadnost požití**
Testovací subjekt 1	9	8	10	7
Testovací subjekt 2	9	6	9	7
Testovací subjekt 3	9	8	10	7
Testovací subjekt 4	8	7	9	6
Testovací subjekt 5	8	6	9	5
Průměr	**8.6**	**7**	**9.4**	**6.4**

Tabulka 6.1 Tabulka hodnocení aplikace testovacími subjekty.

Provedené uživatelské testování ukázalo, že má aplikace potenciál být užitečným nástrojem pro vykazování hovorů. Uživatelé obecně ocenili její jednoduchost a vzhled. Na základě testování bylo identifikováno několik problémů, které byly sepsány a popsány. Ke každému z problémů bylo navrženo jeho řešení. Všechna popsaná řešení se podařilo úspěšně implementovat. Následně byla nová vylepšená verze aplikace vydána na Amazon Appstore. V budoucnu by bylo vhodné provést opakované uživatelské testování, aby bylo zjištěno, zda provedené změny vedly ke zlepšení použitelnosti aplikace.

Kapitola 7

Závěr

Cílem této práce bylo vytvořit prototyp Android aplikace, která usnadní vykazování telefonních hovorů spojených s pracovní činností a pomůže uživatelům s přesnějším a jednodušším vykazováním odpracovaných hodin. Tento cíl byl splněn. Byla vyvinuta funkční aplikace, která dokáže automaticky i manuálně vykazovat čas strávený telefonováním. Aplikace spolupracuje s nástrojem Toggl Track, který využívá jako primární úložiště dat a zdroj projektů a úkolů. Ty jsou k výkazům přiřazeny automaticky na základě historie vykazování hovorů s konkrétním telefonním číslem.

V úvodní části práce byly před samotnou implementací prozkoumány existující aplikace, které lze použít k vykazování hovorů. Následně byly na základě zadání práce a rozhovorů s potenciálními uživateli postupně specifikovány potřeby uživatelů, softwarové požadavky a případy užití. Byly prozkoumány možnosti implementace požadovaných funkcionalit s ohledem na operační systém Android. Na závěr byly analyzovány potřebná Android oprávnění a krátce bylo popsáno i možné využití nástroje Timer2Ticket.

Po analytické části byl proveden návrh aplikace. Nejprve byly představeny doporučené principy pro vytváření udržitelných a moderních Android aplikací. Na základě těchto principů byla navržena a popsána architektura vyvíjené aplikace. Dále byly popsány důležité knihovny a nástroje, které se při implementaci použijí. Bylo rozhodnuto o využití aplikace Toggl Track jako primárního úložiště dat. Na závěr byly na základě všech zjištěných informací navrženy konkrétní obrazovky aplikace.

Třetí část práce se zabývala implementační fází vývoje. Byla popsána implementace jednotlivých vrstev navržené architektury a také složitějších a klíčových funkcionalit aplikace. Dále byly ukázány konečné realizace obrazovek, u kterých bylo provedeno srovnání s původními návrhy. V závěru byl popsán proces vydání hotové aplikace.

V poslední části práce bylo představeno uživatelské testování, které bylo následně provedeno. Byly vytvořeny scénáře, které sloužily jako podklady pro testování. S pěti testovacími subjekty byly zjištěny nedostatky v oblasti použitelnosti i funkčnosti aplikace. Po provedeném testování byly navrženy vhodné úpravy, jejichž cílem bylo odstranění zjištěných nedostatků. Navrhované změny byly v aplikaci realizovány.

Zadání i cíle práce lze považovat za splněné. Byly splněny všechny zadané dílčí úkoly, které vedly k vytvoření funkčního prototypu, který který splňuje všechny požadované funkcionality. V současné době je aplikace dostupná ke stažení na distribuční platformě Amazon Appstore pod názvem Call Tracker.

Budoucí vývoj

Je důležité zdůraznit, že vyvinutá aplikace je prototyp, nikoliv finální produkt. Aplikace nabízí řadu oblastí, na které se lze v budoucím vývoji zaměřit a které lze vylepšit. Níže uvádím pouze několik hlavních oblastí, které byly v průběhu práce nalezeny a které považuji za perspektivní.

- Při specifikaci požadavků byly některé požadavky označeny prioritou „Won't have". Mezi ně patří vykazování dalších typů hovorů, zapamatování nastavení aplikace napříč více zařízeními, možnost rozdělit probíhající výkaz na dílčí výkazy nebo přihlašování pomocí emailové adresy a hesla. Všechny tyto požadavky nebyly v současné verzi aplikace realizovány a nebyly v této práci dále rozebírány. Budoucí vývoj by se mohl zaměřit na tyto požadavky a zvážit jejich realizaci. Dalšími požadavky, které stojí za zvážení, jsou například fungování aplikace bez internetového připojení nebo podpora více jazyků.
- Možným nedostatkem je úzké napojení na aplikaci Toggl Track. Call Tracker tuto aplikaci využívá jako primární zdroj a úložiště dat. V budoucí verzi by bylo vhodné zvážit vytvoření vlastního backendového serveru a odpojení od této externí aplikace. Při vývoji bylo na tuto možnost pamatováno a díky použití vícevrstvé architektury a dědičnosti v datové vrstvě nebude tato změna náročná.
- První verze aplikace byla vydána na platformě Amazon Appstore. Úkolem do budoucna je zajistit vydání i na dalších distribučních platformách, zejména na Google Play. Při vydávání bude nutné řešit problémy s nebezpečnými oprávněními, které současná verze aplikace potřebuje, a které jsou na Google Play přísně kontrolovány.
- Dalším místem pro zlepšení jsou automatické testy, které v aplikaci chybí. Automatické testování nebylo prioritou této práce, ale v dalších verzích aplikace je nutné tyto testy přidat a aplikaci řádně otestovat.
- Během uživatelského testování uvedl jeden z účastníků, že by ocenil, kdyby se přiřazený projekt a úkoly zobrazily již během probíhajícího telefonátu v telefonní aplikaci. Tento nápad by bylo vhodné prozkoumat a zvážit jeho implementaci. Řešením by mohlo být vytvoření vlastní telefonní aplikace. Tento způsob implementace by přinesl hned několik benefitů. Umožnil by jednodušší ovládání, přesnější vykazování hovorů a snadnější přístup k informacím o probíhajícím hovoru. Takové řešení by si vyžádalo důkladnou analýzu a vývoj, ale mohla by se ukázat jako efektivní řešení pro budoucí verzi aplikace.

Příloha A

Detailní popis průběhu uživatelského testování

Během testování byl pořízen záznam obrazovky na testovacím zařízení. U čtvrtého testovacího subjektu se záznam neuložil, a proto není k dispozici. Záznamy jsou k dispozici na přiloženém médiu.

A.1 Testovací subjekt 1

Uživatel chtěl aplikaci nainstalovat z Amazon Appstore, ale neměl dostatek volné paměti. Proto byl využit Android emulátor.

Scénář 1

Uživatel spustil aplikaci a bez váhání povolil všechna potřebná oprávnění. Měl problém nalézt v aplikaci Toggl Track sekci s API klíčem, ale nakonec klíč našel. Ocenil by, kdyby aplikace napověděla, kde hledat.

Scénář 2

Uživatel správně přešel do sekce nastavení, ve které našel tlačítko pro odhlášení. Přihlášení pod nový účet zvládl správně. Nastavení aplikace provedl podle popisu.

Scénář 3

Uživatel správně pochopil, že musí nastavit výchozí projekt. Správně přesel do sekce nastavení a změnu provedl.

Scénář 4

Uživatel správně přešel do sekce nastavení a spustil automatické vykazování hovorů.

Scénář 5

Po ukončení hovoru uživatel očekával, že se hovor zobrazí ve výpisu hovorů. Snažil se zapínat a vypínat filtry, nevěděl však, že je potřeba stisknout tlačítko pro aktualizaci výpisu. Během tohoto scénáře byl omylem automaticky vykázaný hovor moderátorem smazán. Uživatel se nenechal zaskočit a zjistil, že hovor není vykázán. Hovor poté manuálně vykázal a správně přiřadil úkoly i projekt.

Scénář 6

Uživatel správně přešel do sekce nastavení a pozastavil automatické vykazování hovorů.

Scénář 7

Uživatel správně zvolil hovor ze seznamu a provedl jeho správné vykázání.

Scénář 8

Uživatel nejdříve nevěděl, ze které obrazovky výkaz smazat. Po chvilce hledání přešel na obrazovku s výkazy, kde zvolil výkaz daného hovoru a objevil tlačítko pro jeho smazání. Následně výkaz úspěšně smazal.

Scénář 9

Uživatel byl opět zmatený, že musí ručně aktualizovat výpis hovorů. Po chvilce klikl na ruční aktualizaci a objevil se mu provedený hovor. Přešel na výkaz tohoto hovoru a změnil projekt na ten požadovaný ze scénáře. Výkaz následně potvrdil. Nicméně během hovoru bylo spuštěno automatické vykazování, takže uživatel vytvořil další výkaz místo změny původního. Uživatel poté kontroloval změnu projektu zvolením stejného hovoru, ale u něj se brala historie z automaticky vykázaného výkazu, tedy se stále zobrazoval původní nezměněný projekt. Uživatel byl zmatený. Zkusil opět změnit projekt opětovným vykázáním stejného hovoru pod nový projekt. Následně přešel na obrazovku s výkazy, kde identifikoval první výkaz ze seznamu jako výkaz daného hovoru a změnil u něj projekt. Tím sice splnil úkol scénáře, ale nevšiml si, že vykázal tento hovor několikrát.

Finální hodnocení aplikace

Po vysvětlení problémů, které se mu během práce s aplikací objevily, uživatel řekl, že nepochopil, že pro editaci vykázaných hovorů musí přejít na obrazovku s výkazy. Snažil se upravit výkaz na stránce s hovory, čímž vytvářel nové výkazy již vykázaného hovoru. Nepřečetl si název „Export call“ na stránce pro manuální vykázání hovoru. Dodal, že kdyby se ho všiml, možná by takto nepostupoval. Ocenil by, kdyby již vykázané hovory nešlo znovu vykázat. Dále řekl, že obrazovku s výkazy vůbec nepovažoval za důležitou a nechápal, že se jedná o vykázané hovory. Možná by pomohly úvodní stránky při prvotním spuštění, které by ukázaly a vysvětlily k čemu slouží jednotlivé obrazovky aplikace. Také se mu nelíbilo, že musí manuálně aktualizovat obrazovku, když na ní přejde.

- Užitečnost: 9
- Uspořádání: 8
- Estetika: 10

- Snadnost použití: 7

A.2 Testovací subjekt 2

Uživatel nechtěl aplikaci instalovat z Amazon Appstore. Souhlasil s přímým zasláním aplikace a její instalací. Při instalaci se objevila chyba. Bylo zjištěno, že uživatelův telefon nemá požadovanou minimální verzi Androidu. Uživatel nechtěl provést aktualizaci systému, proto bylo nutné využít Android emulátor.

Scénář 1

Uživatel bez váhání přijal všechna oprávnění. Jelikož neměl zkušenosti s aplikací Toggl Track, trvalo mu, než na webu našel sekci s API klíčem. Při zadávání klíče očekával, že klávesnice zmizí po kliknutí mimo textové pole. Nevěděl, že musí klávesnici zavřít tlačítkem. Několikrát klikl na tlačítko Enter, čímž přidal odřádkování. Kvůli tomu zadaný klíč nefungoval. Smazal ho a vložil klíč nově zkopírovaný. Tentokrát již nestiskl Enter a tím nepřidal odřádkování. Klíč fungoval. Další nastavení aplikace proběhlo v pořádku.

Scénář 2

Uživatel správně přešel do sekce nastavení, ale zde chvíli váhal, jestli je tlačítko „Disconnect from Toggl Track“ to správné. Snažil se v aplikaci najít jiný způsob, který by ho odhlásil. Jelikož jiné nenašel, rozhodl se vrátit se do nastavení a kliknout na správné tlačítko, které ho odhlásilo. Opakované nalezení API klíče proběhlo již v pořádku stejně jako nastavení aplikace.

Scénář 3

Uživatel správně přešel na obrazovku nastavení, kde změnil výchozí projekt. Po nastavení odešel ze stránky. Nebyl si jistý, jestli se nastavení provedlo. Proto se vrátil na obrazovku a zkontroloval aktuální stav nastavení.

Scénář 4

Uživatel správně přešel na obrazovku nastavení. Zde nejdříve zaškrtl pouze checkbox, který spouští zapínaní automatického vykazování při startu aplikace. Odešel z obrazovky, ale pro kontrolu se na ni vrátil. Na podruhé si všiml tlačítka na zapnutí vykazování. Automatické vykazování poté úspěšně spustil.

Scénář 5

Po dokončení hovoru aktualizoval stránku s hovory. Následně zaškrtl filtr „Exported Calls“ a v seznamu se mu objevil vykázaný hovor. Na hovor klikl a výkaz nastavil podle scénáře. Při vybírání značky měl problém se zavřením dialogu. Tímto postupem ale vykázal již vykázaný hovor, jelikož měl v době hovoru spuštěné automatické vykazování.

Scénář 6

Uživatel správně přešel na stránku nastavení a zastavil automatické vykazování hovorů.

Scénář 7

Po dokončení hovoru uživatel přešel na obrazovku s hovory. Zde aktualizoval stránku a poté klikl na vypnutí filtru „Exported calls“. V seznamu se objevil provedený hovor. Uživatel ihned znovu klikl na tento filtr a zapl ho. Chybou v programu se mu zobrazil i provedený neexportovaný hovor. Uživatel na něj klikl, správně zvolil značku „analýza“. Měl problém se zavřením dialogu. Hovor úspěšně vykázal. Následně pro kontrolu klikl na hovor znovu a vykázal ho podruhé.

Scénář 8

Uživatel po dokončení hovoru přešel na stránku s hovory. Zde po aktualizaci obrazovky našel provedený hovor. Nevěděl, kde má výkaz smazat. Krátce přešel na správnou obrazovku „Entries“, která ale nebyla aktualizována, takže na ní žádný výkaz nebyl. Neustále klikal na hovor na obrazovce „Calls“, kde se snažil najít způsob, jak výkaz smazat. Dokázal přejít na stránku „Entries“, kterou aktualizoval a hovor smazal. Následně ale přešel zpět na stránku s hovory, kde hovor opět vykázal. Na závěr všechny výkazy tohoto hovoru úspěšně smazal. Poté se ale vrátil na stránku s hovory a hovor opět manuálně vykázal.

Scénář 9

Uživatel po dokončení hovoru manuálně tento hovor vykázal. Hovor byl již ale automaticky vykázaný. Při manuálním výkazu přenastavil projekt, ale v seznamu se objevoval dříve automaticky vykázaný hovor, z tohoto důvodu se přiřazoval stále původní projekt. Uživatel přešel na obrazovku s výkazy, kde vybral první zobrazený hovor se správným číslem a u něj správně přenastavil projekt. Následně chtěl otestovat nastavení opakovaným hovorem s daným číslem. Kontrolu správnosti výkazu provedl z obrazovky s hovory, ze které provedl opět manuální vykázání již vykázaného hovoru.

Finální hodnocení

Uživatel měl problém se smazáním výkazu. Stejně jako předchozí testovací subjekt si nespojil obrazovku „Entries“ s vykázanými hovory. Využíval hlavně stránku „Calls“, čímž opakovaně vykazoval již vykázaný hovor. Po vysvětlení chyb, které se objevily při jeho testování, řekl, že by omezil vykazování již vykázaných hovorů a při prvním zapnutí aplikace by ocenil onboarding obrazovky. Také řekl, že by chtěl, aby při zadávání API klíče mizela klávesnice po kliknutí mimo textové pole. Na obrazovce nastavení by ocenil potvrzovací tlačítko, které by ho ujistilo, že nastavení proběhlo. Při vybírání značek měl problém se zavřením dialogu.

- Užitečnost: 9
- Uspořádání: 6
- Estetika: 9
- Snadnost použití: 7

A.3 Testovací subjekt 3

Uživatelka nechtěla aplikaci instalovat do svého zařízení, proto byl využit Android emulátor.

Scénář 1

Uživatelka přijala všechna potřebná oprávnění. Při zadávání API klíče omylem při zavírání klávesnice vložila odřádkování a API klíč nefungoval. Rozhodla se vygenerovat nový klíč, ale při mazání původního nesmazala odřádkování. Proto ani nový klíč nefungoval. Následně odřádkování smazala a pokračovala dál. Nastavení aplikace proběhlo v pořádku.

Scénář 2

Uživatelka správně přešla na obrazovku s nastavením. Následně se vrátila na předchozí obrazovku, kde hledala další možnosti, jak se odhlásit. Poté se opět vrátila na obrazovku s nastavením, kde klikla na správné tlačítko pro odhlášení. Druhé přihlášení již probíhalo v pořádku, aplikaci uživatelka nastavila správně.

Scénář 3

Uživatelka správně přešla na obrazovku s nastavením, kde správně změnila výchozí projekt.

Scénář 4

Uživatelka na obrazovce s nastavením správně spustila automatické vykazování hovorů.

Scénář 5

Během tohoto scénáře se vyskytly problémy s automatickým vykazováním. Po opravě se ale podařilo provedený hovor vykázat. Uživatelka správně přešla na obrazovku s výkazy, kde zvolila požadovaný hovor. Vybrala možnost přidání značek. Správně přidala značku, ale v seznamu se objevily i značky, které nešlo přidat. Zároveň měla problém se zavřením dialogu. Následně ještě pro sebe vyzkoušela, jestli se změny provedly, pokud je nepotvrdí.

Scénář 6

Uživatelka správně přešla do sekce nastavení, kde vypnula automatické vykazování hovorů.

Scénář 7

Uživatelka hledala hovor na obrazovce s výkazy. Jelikož ale hovor nebyl vykázaný, nemohla ho zde nikdy najít. Následně přešla na obrazovku s hovory. Na obrazovce pomocí vyhledávání našla daný hovor a zvolila ho. Správně upravila značky u hovoru a úspěšně hovor vykázala.

Scénář 8

Uživatelka přešla na stránku s hovory. Zde našla provedený hovor a vykázala ho, čímž vytvořila další výkaz tohoto hovoru. Což bylo špatně. Přešla na obrazovku s výkazy, kde výkaz hovoru našla a správně provedla jeho smazání.

Scénář 9

Uživatelka po dokončení hovoru přešla na stránku s hovory. Zde nalezla dokončený hovor, který zvolila a změnila u něj projekt. Hovor vykázala, čímž vytvořila druhý výkaz stejného hovoru. To ještě několikrát zopakovala, protože historie hovorů se brala z předchozího automaticky vykázaného hovoru, u kterého zůstával nastavený původní projekt. Kvůli tomu se ve výkazu pořád objevoval původní projekt. Následně přešla na obrazovku s výkazy, kde poslední výkaz tohoto hovoru upravila a změnila správně projekt.

Finální hodnocení

Hlavním problémem bylo nepochopení rozdílu mezi obrazovkou s hovory a obrazovkou s výkazy. Uživatelka se snažila měnit historii výkazů na obrazovce s hovory, tím dokola vytvářela nové výkazy. Dále se při přidávání značky k výkazu objevily i značky, na které nešlo kliknout. Uživatelka byla kvůli tomu zmatená.

- Užitečnost: 9
- Uspořádání: 8
- Estetika: 10
- Snadnost použití: 7

A.4 Testovací subjekt 4

Testovací subjekt má zkušenosti s podobnou aplikací na vykazování času. Z tohoto důvodu poskytl na konci mnoho nápadů na zlepšení. Záznam obrazovky se při pořizování poškodil, a proto není k dispozici. Uživatel nevlastní Android zařízení, ale má zkušenosti s Android systémem. Pro testování byl opět využit Android emulátor.

Scénář 1

Uživatel po spuštění povolil bez váhání všechna oprávnění. Úspěšně našel API klíč ve svém Toggl Track účtu a zadal ho do aplikace. Stejně jako ostatní měl problém se zavřením klávesnice. Při nastavování aplikace využil možnost vytvořit nový projekt. Poté zkusil vytvořit nový projekt s již obsazeným názvem, aplikace zobrazila chybovou hlášku a umožnila uživateli opakovat akci. Uživatel na konci nastavil prvně vytvořený projekt a pokračoval do aplikace.

Scénář 2

Uživatel okamžitě přešel na obrazovku s nastavením a použil tlačítko pro odhlášení. Přihlášení pod novým účtem proběhlo v pořádku. Aplikaci uživatel nastavil podle zadání.

Scénář 3

Uživatel přešel do nastavení aplikace a správně pochopil, že musí změnit výchozí projekt, který také změnil. Po stisknutí tlačítka pro návrat nevěděl, jestli se nastavení uložilo, a vrátil se na předchozí obrazovku, kde nastavení ještě jednou zkontroloval.

Scénář 4

Uživatel pochopil, že může v nastavení zapnout automatické vykazování hovorů. Přešel na stránku s nastavením a vykazování zapnul.

Scénář 5

Uživatel po dokončení hovoru přešel na stránku s hovory, kterou aktualizoval. Zde našel dokončený hovor. Ten manuálně vykázal, což byla chyba. Hovor byl již automaticky vykázaný, ale uživatel si toto neuvědomil.

Scénář 6

Uživatel správně přešel do nastavení a vypnul automatické vykazování hovorů.

Scénář 7

Uživatel přešel na obrazovku s hovory, kde našel provedený hovor. Ten začal vykazovat. Při vykazování nezvolil značku, která symbolizuje, že se hovor týkal analýzy. Hovor potvrzovacím tlačítkem vykázal, ale nebyl si jistý, jestli se vykázání podařilo.

Scénář 8

Uživatel po dokončení hovoru přešel na obrazovku s hovory, kde aktualizoval obsah. Našel zde provedený hovor, ale nevěděl, jak výkaz smazat. Následně přešel na stránku s výkazy, kde našel vykázaný hovor. Ten zvolil a klikl na tlačítko smazání, čímž výkaz úspěšně smazal.

Scénář 9

Uživatel stejně jako všichni ostatní upravoval výkaz na stránce s hovory. Změnil projekt a hovor znovu vykázal, čímž považoval scénář za splněný.

Finální hodnocení

Uživatel po provedeném testování uvedl, že nepochopil rozdíl mezi hovorem a výkazem. Podle něj by stačila pouze jedna obrazovka s hovory, kde by bylo vidět, které hovory jsou vykázané a které nejsou. Obrazovka s výkazy je podle něj zbytečná. Dále by ocenil, kdyby mizela klávesnice po kliknutí mimo textové pole. V nastavení aplikace by měl rád potvrzovací tlačítko. Po kliknutí na něj by měl větší jistotu, že se nastavení úspěšně provedlo, než po kliknutí na tlačítko zpět. Po manuálním vykázaní by rád viděl zpětnou vazbu od aplikace, že vše proběhlo úspěšně. Nelíbila se mu ruční aktualizace obrazovek po přechodu na ně. U vybraných značek by rád viděl ikonu, která by signalizovala, že je možné je odstranit. Filtr na exportované hovory by podle něj měl tak, že po kliknutí na něj se zobrazí pouze exportované hovory, po jeho zrušení se zobrazí všechny hovory.

- Užitečnost: 8
- Uspořádání: 7
- Estetika: 9
- Snadnost použití: 6

A.5 Testovací subjekt 5

Uživatelka má nejmenší zkušenosti s vykazováním času. K vykazování v minulosti používala jednodušší aplikaci, která umožňovala pouze spouštění a zastavování časomíry a vytváření časových záznamů. Z tohoto důvodu poskytla cenné informace, které popisují, jak aplikaci používají lidé méně zkušení ve vykazování času. Z důvodu nesouhlasu s instalací do vlastního zařízení byl opět využit Android emulátor. Jelikož všichni předchozí testovací subjekty nepochopily funkce hlavních obrazovek, byly uživatelce obě hlavní obrazovky detailněji popsány.

Scénář 1

Uživatelka povolila všechna oprávnění a pokračovala na přihlašovací obrazovku. Nevěděla, kde najít Toggl Track API klíč, protože Toggl Track nikdy nepoužívala. Uživatelka byla moderátorem navedena do správné sekce, odkud API klíč získala a vložila do aplikace. Měla stejně jako ostatní problém se zavřením klávesnice, snažila se kliknout mimo textové pole. Následně využila k zavření správné tlačítko na klávesnici. Aplikaci nastavila podle svého uvážení. Při volbě projektu vytvořila nový projekt.

Scénář 2

Uživatelka správně přešla na obrazovku s nastavením, kde se odhlásila. Druhé přihlašování již proběhlo bez problémů. Aplikaci nastavila podle scénáře.

Scénář 3

Uživatelka přešla do sekce nastavení, kde správně změnila výchozí projekt. Následně ale klikla na tlačítko pro vytvoření nového projektu, které považovala za potvrzovací tlačítko. Následně z dialogu odešla. Po scénáři uvedla, že se domnívala, že musí změnu nějak potvrdit.

Scénář 4

Uživatelka pochopila, že musí v nastavení zapnout automatické vykazování a nastavení provedla.

Scénář 5

Po skončení hovoru neproběhlo automatické vykázání. Moderátor musel zasáhnout do probíhajícího scénáře, jelikož se jednalo o chybu aplikace. Následně byl vyvolán nový hovor. Ten se již správně vykázal. Uživatelka přešla na obrazovku s výpisy, kde se pokusila hovor upravit. Z neznámého důvodu úprava selhala kvůli špatnému formátu data. Uživatelka byla seznámena s chybou a testovací scénář skončil.

Scénář 6

Uživatelka přešla na obrazovku s nastavením a správně vypnula automatické vykazování.

Scénář 7

Uživatelka měla po dokončení hovorů zapnutý filtr pouze na exportované hovory. Snažila se provedený hovor vyhledat, ale neúspěšně. Následně filtr vypnula a hovor našla. Úspěšně přidala značku a hovor správně vykázala.

Scénář 8

Při tomto scénáři se automaticky vykázal předchozí hovor. Byl proveden další hovor, který se již vykázal správně. Uživatelka správně přešla na obrazovku s výkazy, kterou nejdříve neaktualizovala a snažila se vyhledat hledaný výkaz. Následně provedla aktualizaci a výkaz našla. Následně ho správně smazala.

Scénář 9

Opět se chybou aplikace automaticky místo provedeného hovoru vykázal hovor předchozí. Byl proveden opakovaný hovor, který se již vykázal správně. Uživatelka přešla na stránku s výkazy, kterou aktualizovala a našla provedený hovor. U provedeného hovoru správně změnila projekt. Následně se pokoušela na stránce nastavení najít způsob, jak nastavit, aby se všechny budoucí hovory vykazovaly pod daný projekt. Po dokončení testu byl uživatelce vysvětlen princip přiřazování projektů a vyzkoušen ještě jeden hovor, aby bylo vidět, že se automaticky přiřadí projekt, který předtím nastavila.

Finální hodnocení

Testování s tímto subjektem provázela mnoho neočekávaných chyb aplikace. Během testování se objevil problém s automatickým vykazováním hovorů. Bylo zjištěno, že se chyba objevila, pokud je výpis hovorů zařízení prázdný a po dokončení hovoru se aplikace pokusí přečíst první hovor z výpisu. Ten by již měl být ve výpisu zaznamenaný, neboť hovor v době čtení již skončil. Pravděpodobně ale trvá, než se hovor do výpisu propíše. Aplikace se pokusila získat hovor z prázdného seznamu a automatické vykazování tak selhalo. Následně se ještě několikrát stalo, že se aplikace pokoušela číst z výpisu ukončený hovor, ale ten ještě nebyl zaznamenaný. Místo něho získala a vykázala předchozí hovor. Řešení tohoto problému bude více rozebráno ve sekci věnující se zhodnocení výsledků uživatelského testování.

Díky vysvětlení funkce každé z hlavních obrazovek aplikace působila uživatelka méně zmateně než předchozí uživatelé. Správně přecházela na stránku s výkazy, pokud měla pracovat s již vykázaným hovorem. Nevykázané hovory hledala správně na obrazovce s hovory.

Uvedla, že na obrazovce s hovory nepochopila filtrování exportovaných hovorů. Ocenila by sjednocení názvů „exportované hovory“ z filtrovacího panelu a „záznamy“ jako název obrazovky, protože podle ní se jedná o stejnou věc. U všech dialogů jí chybělo potvrzovací tlačítko. Pro automatické přiřazování projektu a úkolů by v nastavení ocenila sekci, která by umožnila ke konkrétnímu kontaktu nebo číslu ručně nastavit a měnit přiřazovaný projekt. Na závěr uvedla, že by se jí líbilo, kdyby se přiřazený projekt a úkoly zobrazily již během probíhajícího telefonátu.

- Užitečnost: 8
- Uspořádání: 6
- Estetika: 9
- Snadnost použití: 5

Bibliografie

1. BREZNEN, Adam. *Snadné vykazování činností - iOS* [https://dspace.cvut.cz/handle/10467/110105]. [B.r.]. (Accessed on 03/19/2024).
2. ŠTEFAN, Vít. *Synchronizační middleware mezi projektovým systémem a aplikací na sledování času stráveného na projektech* [https://dspace.cvut.cz/handle/10467/94628]. [B.r.]. (Accessed on 03/19/2024).
3. *Toggl Track: Time Tracking Software for Any Workflow* [https://toggl.com/]. [B.r.]. (Accessed on 04/25/2024).
4. *Toggl Track Vocabulary — Toggl Track Knowledge Base* [https://support.toggl.com/en/articles/2219505-toggl-track-vocabulary]. [B.r.]. (Accessed on 04/25/2024).
5. *Features — Toggl Track* [https://toggl.com/track/features/#integrations-section]. [B.r.]. (Accessed on 04/25/2024).
6. *Home — Toggl Engineering* [https://engineering.toggl.com/]. [B.r.]. (Accessed on 04/25/2024).
7. *Toggl Track - Time Tracking – Aplikace na Google Play* [https://play.google.com/store/apps/details?id=com.toggl.giskard]. [B.r.]. (Accessed on 04/27/2024).
8. *Timing Automatic Mac Time Tracker – Manual Timers Optional* [https://timingapp.com/]. [B.r.]. (Accessed on 04/08/2024).
9. *Screen Time Integration – Timing Time Tracker* [https://timingapp.com/help/screen-time?utm_source=app&utm_campaign=appHelp&utm_medium=link]. [B.r.]. (Accessed on 04/04/2024).
10. *Phone Call Integration – Timing Time Tracker* [https://timingapp.com/help/phone-calls?utm_source=app&utm_campaign=appHelp&utm_medium=link]. [B.r.]. (Accessed on 04/04/2024).
11. *Timely Automatic Time Tracking – Aplikace na Google Play* [https://play.google.com/store/apps/details?id=com.timeapp.devlpmp]. [B.r.]. (Accessed on 04/03/2024).
12. *Android Phone Call Logs — Timely Help Center* [https://support.timelyapp.com/en/articles/2800829-android-phone-call-logs]. [B.r.]. (Accessed on 04/03/2024).
13. *RescueTime: Fully Automated Time Tracking Software* [https://www.rescuetime.com/]. [B.r.]. (Accessed on 04/17/2024).
14. *RescueTime Classic - Apps on Google Play* [https://play.google.com/store/apps/details?id=com.rescuetime.android&hl=en_US]. [B.r.]. (Accessed on 04/17/2024).
15. *Welcome to Jira Software — Atlassian* [https://www.atlassian.com/software/jira/guides/getting-started/introductio#nwhat-is-jira-software]. [B.r.]. (Accessed on 03/22/2024).

16. *Introduction to Jira Projects — Atlassian* [https://www.atlassian.com/software/jira/guides/projects/overview#what-is-a-jira-project]. [B.r.]. (Accessed on 03/22/2024).
17. *Introduction to Jira Boards — Atlassian* [https://www.atlassian.com/software/jira/guides/boards/overview#what-is-a-jira-board]. [B.r.]. (Accessed on 04/17/2024).
18. *Overview - Redmine* [https://www.redmine.org/projects/redmine/wiki]. [B.r.]. (Accessed on 04/02/2024).
19. *Features - Redmine* [https://www.redmine.org/projects/redmine/wiki/Features]. [B.r.]. (Accessed on 04/02/2024).
20. IEEE Recommended Practice for Software Requirements Specifications. *IEEE Std 830-1998*. 1998, s. 1–40. Dostupné z DOI: 10.1109/IEEESTD.1998.88286.
21. ISO/IEC/IEEE International Standard - Systems and software engineering – Life cycle processes –Requirements engineering. *ISO/IEC/IEEE 29148:2011(E)*. 2011, s. 1–94. Dostupné z DOI: 10.1109/IEEESTD.2011.6146379.
22. (IREB), International Requirements Engineering Board. *Glossary of Software Requirements Engineering Terminology*. 2020. Dostupné také z: https://www.ireb.org/en/cpre/glossary/.
23. *What do Requirements Stand for? – School of Information Systems* [https://sis.binus.ac.id/2018/02/12/what-do-requirements-stand-for/]. [B.r.]. (Accessed on 05/06/2024).
24. *Capturing Architectural Requirements* [https://web.archive.org/web/20201112020231/http://www.ibm.com/developerworks/rational/library/4706.html]. [B.r.]. (Accessed on 05/06/2024).
25. IIBA. *A guide to the business analysis body of knowlwdge (babok guide)*. IIBA, 2009.
26. COCKBURN, Alistair. *Writing effective use cases*. Addison-Wesley, 2016.
27. *Content providers — Android Developers* [https://developer.android.com/guide/topics/providers/content-providers]. [B.r.]. (Accessed on 04/05/2024).
28. *CallLog — Android Developers* [https://developer.android.com/reference/android/provider/CallLog]. [B.r.]. (Accessed on 04/08/2024).
29. *TelephonyCallback — Android Developers* [https://developer.android.com/reference/kotlin/android/telephony/TelephonyCallback]. [B.r.]. (Accessed on 04/09/2024).
30. *TelephonyCallback.CallStateListener — Android Developers* [https://developer.android.com/reference/kotlin/android/telephony/TelephonyCallback.CallStateListener]. [B.r.]. (Accessed on 04/09/2024).
31. *Application fundamentals — Android Developers* [https://developer.android.com/guide/components/fundamentals]. [B.r.]. (Accessed on 04/18/2024).
32. *Kotlin Multiplatform — Kotlin Documentation* [https://kotlinlang.org/docs/multiplatform.html#multiplatform-libraries]. [B.r.]. (Accessed on 04/16/2024).
33. *Kotlin for Android — Kotlin Documentation* [https://kotlinlang.org/docs/android-overview.html]. [B.r.]. (Accessed on 04/16/2024).
34. *Android's Kotlin-first approach — Android Developers* [https://developer.android.com/kotlin/first]. [B.r.]. (Accessed on 04/16/2024).
35. *Layouts in Views — Android Developers* [https://developer.android.com/develop/ui/views/layout/declaring-layout]. [B.r.]. (Accessed on 04/16/2024).
36. *Jetpack Compose UI App Development Toolkit - Android Developers* [https://developer.android.com/develop/ui/compose]. [B.r.]. (Accessed on 04/16/2024).

37. *Thinking in Compose — Jetpack Compose — Android Developers* [https://developer.android.com/develop/ui/compose/mental-model]. [B.r.]. (Accessed on 04/16/2024).
38. MAIER, M.W.; EMERY, D.; HILLIARD, R. Software architecture: introducing IEEE Standard 1471. *Computer*. 2001, roč. 34, č. 4, s. 107–109. Dostupné z DOI: 10.1109/2.917550.
39. *Guide to app architecture — Android Developers* [https://developer.android.com/topic/architecture]. [B.r.]. (Accessed on 04/16/2024).
40. *UI layer — Android Developers* [https://developer.android.com/topic/architecture/ui-layer]. [B.r.]. (Accessed on 04/16/2024).
41. *Domain layer — Android Developers* [https://developer.android.com/topic/architecture/domain-layer]. [B.r.]. (Accessed on 04/16/2024).
42. *Data layer — Android Developers* [https://developer.android.com/topic/architecture/data-layer]. [B.r.]. (Accessed on 04/16/2024).
43. *Settings — Views — Android Developers* [https://developer.android.com/develop/ui/views/components/settings]. [B.r.]. (Accessed on 04/18/2024).
44. *App Architecture: Data Layer - DataStore - Android Developers* [https://developer.android.com/topic/libraries/architecture/datastore]. [B.r.]. (Accessed on 04/18/2024).
45. *Android Keystore system — App quality — Android Developers* [https://developer.android.com/privacy-and-security/keystore]. [B.r.]. (Accessed on 04/18/2024).
46. *Save data in a local database using Room — Android Developers* [https://developer.android.com/training/data-storage/room]. [B.r.]. (Accessed on 04/18/2024).
47. *Hilt* [https://dagger.dev/hilt/]. [B.r.]. (Accessed on 04/16/2024).
48. *Dependency injection in Android — Android Developers* [https://developer.android.com/training/dependency-injection]. [B.r.]. (Accessed on 04/16/2024).
49. *Retrofit* [https://square.github.io/retrofit/]. [B.r.]. (Accessed on 04/18/2024).
50. *Serialization — Kotlin Documentation* [https://kotlinlang.org/docs/serialization.html]. [B.r.]. (Accessed on 04/18/2024).
51. *Sketch · Design, collaborate, prototype and handoff* [https://www.sketch.com/]. [B.r.]. (Accessed on 04/17/2024).
52. *Android Developers Blog: Android Studio: An IDE built for Android* [https://android-developers.googleblog.com/2013/05/android-studio-ide-built-for-android.html]. [B.r.]. (Accessed on 04/24/2024).
53. *Foreground services — Background work — Android Developers* [https://developer.android.com/develop/background-work/services/foreground-services]. [B.r.]. (Accessed on 04/25/2024).
54. *Publish your app — Android Studio — Android Developers* [https://developer.android.com/studio/publish]. [B.r.]. (Accessed on 04/27/2024).
55. *AI Image Generator* [https://deepai.org/machine-learning-model/text2img]. [B.r.]. (Accessed on 05/05/2024).
56. *Use of SMS or Call Log permission groups - Play Console Help* [https://support.google.com/googleplay/android-developer/answer/10208820?sjid=11397258777678189784-EU]. [B.r.]. (Accessed on 05/08/2024).
57. *Aptoide - The alternative Android app store* [https://en.aptoide.com/]. [B.r.]. (Accessed on 05/08/2024).
58. *The Amazon App* [https://www.amazon.com/gp/mas/get/amazonapp]. [B.r.]. (Accessed on 05/09/2024).

59. NIELSEN, J. *Usability Engineering*. Elsevier Science, 1994. Interactive Technologies. ISBN 9780125184069. Dostupné také z: https://books.google.cz/books?id=95As2OF67f0C.

60. *How Many Test Users in a Usability Study?* [https://www.nngroup.com/articles/how-many-test-users/]. [B.r.]. (Accessed on 05/12/2024).

Obsah příloh

- `readme.txt` ..... stručný popis obsahu média
- `app`
  - `app.apk` ..... instalační soubor vytvořené aplikace ve formátu APK
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text`
  - `thesis.pdf` ..... text práce ve formátu PDF
- `recordings` ..... nahrávky obrazovek z uživatelského testování ve formátu MP4