FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Využití multiplatformního frameworku pro sjednocení frontendu aplikace
Student:	Bc. Martin Šach
Vedoucí:	Ing. Jan Blizničenko
Studijní program:	Informatika
Obor / specializace:	Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Společnost Daktela poskytuje zákazníkům služby pro komunikaci s jejich klienty pomocí různých komunikačních kanálů v jediné webové aplikaci (hovory, emaily, SMS, chat). Kromě rozsáhlé webové aplikace spadají do ekosystému společnosti také mobilní aplikace, desktop aplikace a webové rozšíření. Všechny tyto aplikace byly ovšem vyvinuty zcela samostatně a je náročné udržovat každou z nich samostatně. Cílem práce je sjednotit uvedené služby ekosystému (kromě webové aplikace) a usnadnit tak jejich údržbu a další rozvoj.

Postupujte v následujících krocích:
- analyzujte stávající produkty ekosystému (mobilní aplikaci, desktop aplikaci a webové rozšíření) a podobné aplikace a projekty - analyzujte požadavky zadavatele a zvolte vhodné technologie
- navrhněte, implementujte a otestujte mobilní aplikaci
- zdokumentujte mobilní aplikaci a nasaďte ji do produkčního prostředí
- zprovozněte demo desktop aplikace a webového rozšíření
- shrňte přínos práce a popište další rozvoj

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 17. ledna 2022 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Využití multiplatformního frameworku pro sjednocení frontendu aplikace

Bc. Martin Šach

Katedra softwarového inženýrství
Vedoucí práce: Ing. Jan Blizničenko

3. května 2022

Poděkování

Na tomto místě bych rád poděkoval panu Ing. Janu Blizničenkovi za odborné vedení této práce, jeho cenné rady a přípomínky. Současně bych rád poděkoval společnosti Daktela s.r.o. za možnost realizovat projekt, jímž se práce zabývá, a všem svým kolegům, kteří se na jeho realizaci jakkoliv podíleli. Závěrem děkuji své rodině a přátelům ze jejich podporu při studiích.

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 3. května 2022 . . . . . .

Odkaz na tuto práci

Šach, Martin. *Využití multiplatformního frameworku pro sjednocení frontendu aplikace*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2022.

Abstrakt

Tato diplomová práce popisuje využití multiplatformního frameworku pro zhotovení multiplatformní aplikace. Konkrétněji pak práce popisuje především implementaci (a všechny její nezbytné součásti) multiplatformní mobilní aplikace produktu Daktela prostřednictvím moderního frameworku Flutter. Dále práce demonstruje, jak využít zmíněný framework na dalších platformách jako desktop aplikace nebo rozšíření pro webový prohlížeč.

Klíčová slova multiplatformní vývoj, mobilní vývoj, Dart, Flutter, BLoC, Daktela

Abstract

This master's thesis describes the usage of a cross-platform framework for making a cross-platform app. More specifically, thesis describes an implementation (and all its necessary parts) of the cross-platform mobile app Daktela through modern framework Flutter. Furthermore, it demonstrates how to use mentioned framework on other platforms such as desktop app or web browser extension.

Keywords cross-platform development, mobile developent, Dart, Flutter, BLoC, Daktela

Obsah

Seznam obrázků

Seznam tabulek

Seznam ukázek kódu

Úvod

Tato práce se zabývá využitím multiplatformních technologií ve vývoji software. Konkrétně by pak měla dokázat (respektive její výsledná aplikace), že jsou tyto technologie vhodnou alternativou k nativním a to nejen pro mobilní aplikace, ale také na dalších platformách jako web a desktop. Společnost Daktela dodává komplexní softwarové řešení pro kontaktní centra, jehož součástí jsou kromě rozsáhlé webové aplikace i další, menší aplikace běžící na různých platformách. Jsou to mobilní aplikace, desktop aplikace (neboli program spustitelný z operačního systému počítače) a rozšíření pro webový prohlížeč (doplněk, případně *add-on*, dostupný typicky ve správě rozšíření daného webového prohlížeče – pozn. v práci uváděno zkráceně jako *webové rozšíření*). Navzdory tomu, že jsou tyto menší aplikace relativně podobné, byly doposud vyvíjeny odděleně (včetně odděleného vývoje pro obě primární mobilní platformy – Android a iOS). Právě multiplatformní frameworky umožňují aplikace sjednotit a vytvořit *single codebase*, neboli společný kód pro všechny tyto aplikace, díky kterému bude ve výsledku zrychlený a ulehčený proces vývoje a údržby.

V kapitole 1 jsou stručně shrnuty cíle práce. Hlavním cílem je dodat multiplatformní aplikaci, která (s využitím vhodného frameworku) dokáže nahradit stávající doplňkové aplikace společnosti Daktela. Tím, by mělo dojít ke snížení potřeb na technologické know-how, ale také zrychlení vývojového a release procesu.

Kapitola 2 obecně popisuje produkt Daktela a analyzuje současný stav aplikací, které mají být nahrazeny novou multiplatformní technologií (mobilní aplikace, desktop aplikace a webové rozšíření), a současně také analyzuje eventuální konkurenční aplikace či podobné práce. Shrnuty jsou zde také funkční a nefunkční požadavky zadavatele (společnosti Daktela) na novou mobilní aplikaci. Kapitola též zkoumá možné multiplatformní frameworky a jejich trendy, přičemž na jejím konci je okomentována finálně zvolená technologie.

Obecný návrh, architekturu či způsob komunikace se serverem mapuje

kapitola 3. Vysloveny jsou zde také změny provedené v uživatelském rozhraní aplikace.

Samotnou implementací se zabývá kapitola 4, která postupně shrnuje jednotlivé komponenty, třídy a přístupy použité při zhotovení aplikace. Současně je zde také demostrováno využití aplikace na platformách desktop a web.

Kapitola 5 se zaobírá možnostmi automatizovaného testovaní stejně jako release procesem mobilní aplikace.

Kapitola 1

Cíl práce

Hlavním cílem práce je dokázat, že je možné využít multiplatformní framework pro sjednocení frontendu více aplikací běžících na různých platformách. To konkrétněji znamená navhrnout a zhotovit multiplatformní aplikaci doplňující softwarový produkt Daktela, která nahradí stavající „podpůrné“ aplikace tohoto systému. Důraz je kladen především na její mobilní verzi, která v současné době není navržena tak, aby byla snadno rozšiřitelná. S příchodem multiplatformního frameworku by pak mělo dojít ke snížení počtu rozdílných technologických platforem vývoje a tedy ke snížení požadavku na technologické know-how společnosti. Aktuálně jsou totiž aplikace pro hlavní mobilní platformy (iOS a Android) napsány v nativních technologiích, což značně brzdí vývoj, testování a celý release proces. Stejně tak desktop aplikace a webové rozšíření (neboli doplňek prohlížečů Chrome a Firefox) jsou zcela samostatné projekty. Výsledná aplikace by tedy měla využít takovou technologii, která podporuje multiplatformní vývoj. Tím by mělo mimo jiné dojít ke snížení nákladů na vývoj a údržbu těchto aplikací.

Výsledkem práce by měla být multiplatformní mobilní aplikace minimálně pokrývající funkcionality současné aplikace. Výsledná aplikace by měla být navržena tak, aby bylo možné ji rozšířit/obohatit o nové funkcionality. Zvolená multiplatformní technologie by zároveň měla podporovat tvorbu aplikací i pro další platformy — desktop a web, přičemž by měl autor minimálně demonstrovat, že je to skutečně možné. Verze pro mobilní platformy by pak měla být řádně otestována a nasazena do produkčního prostředí.

Kapitola 2

Analýza

Tato kapitola se zabývá analýzou zpracovanou pro účely této práce. Nejprve obecně shrnuje produkt Daktela a mapuje současný stav aplikací, které mají být nahrazeny aplikací, jejíž zpracování popisuje tato práce. Následně je provedena rešerše konkurečních řešení a podobných prací. Poté jsou popsány požadavky zadavatele, zanalyzovány dostupné technologie a okomentována ta finálně zvolená.

2.1 Daktela

Daktela je softwarová společnost vyvíjející tzv. all-in-one software produkt pro kontaktní centra (call centra). All-in-one znamená, že produkt obsahuje desítky různých modulů, které je možné nejrůzněji konfigurovat, případně customizovat (upravit na míru) tak, aby výsledný produkt přesně splňoval požadavky zákazníka (call centra).

Služba obsahuje modul pro komunikaci se zákazníkem (a to nejrůznějšími kanály: hovor, email, SMS, web chat, Facebook Messenger, Whatsapp, Viber). Dále pak *CRM* modul pro evidenci zákazníků a helpdesk modul modul (tzv. *Tickety*) pro zpracování a následnou obsluhu požadavků zákazníka. Nedílnou součástí celého systému jsou samozřejmě i detailní statistiky, ve kterých je možné detailně filtrovat, což ocení především manažeři call center. Manažeři mohou rovněž využit i customizovatelné wallboardy pro přehled o aktuálním dění v jejich kontaktním centru.

V neposlední řadě obsahuje produkt také detailní nastavení, kde je možné vytvořit uživatele a přiřadit mu příslušný přístup (roli) a oprávnění (profil). Tyto role a opravnění jsou pochopitelně konfigurovatelné a plně v gesci zákazníka využívajícího aplikaci. Dále je v nastavení možné definovat formuláře pro CRM a helpdesk moduly.

Požadavky zákazníků v helpdesk modulu *Tickety* je možné (respektive nutné) členit do kategorií a každá tato kategorie může mít svůj vlastní formulář. U nastavení helpdesk modulu by neměly být opomenuty tzv. pohledy,

které umožňují zákazníkovi definovat si vlastní filtrování nad požadavky (tedy tickety) v helpdesku.

Ostatní moduly aplikace (stejně jako jejich nastavení) není v tuto chvíli nutné popisovat a budou v případě potřeby popsány dále. Všechny aplikace společnosti Daktela jsou licencované a tudíž vyžadují přihlášení (uživatelské jméno a heslo).

2.2 Současný stav

V současné době existují k webové aplikaci tři aplikace podpůrné: mobilní aplikace, desktop aplikace a webové rozšíření. Všechny tyto aplikace byly doposud vyvíjeny odděleně a každá z nich zajišťuje trochu jinou funkcionalitu. Na následujících řádcích budou proto popsány jednotlivě. Jejich hlavní nevýhodou je pochopitelně to, že jsou to samostatné projekty, což značně komplikuje jejich vývoj a release. Všechny níže zmíněné aplikace Daktela ekosystému komunikují se serverem prostřednictvím REST API.

2.2.1 Mobilní aplikace

Aktuální mobilní aplikace poskytují uživatelům přibližně stejnou funkcionalitu, nicméně jsou zastaralé (především jejich UI část). Odlehčená mobilní verze produktu byla zpočátku spíše pilotní projekt. Volba to byla vcelku logická, protože zadavatel (společnost Daktela) potřeboval nejprve ověřit, zda bude o mobilní aplikaci v segmentu jeho podnikání vůbec zájem. Z toho také vycházela i samotná aplikace, která byla zpočátku velmi minimalistická a obsahovala pouze základní funkcionality.

Vyvinuta byla nejprve aplikace pro Android, na platformu iOS se dostalo až posléze. Tato volba byla logická vzhledem k tomu, že v operačních systémech pro mobilní telefony pevnou rukou vládne Android. Podle dat [1] ze serveru `gs.statcounter.com` z prosince 2021 používá Android telefony přibližně 70 % uživatelů, „jablečné“ telefony téměř 30 % uživatelů a zbylé desetiny procent zaujímají ostatní operační systémy (Samsung, KaiOS, Nokia, ...). V České republice je zastoupení Androidu výraznější o další 2 procentní body [2]. Tyto skutečnosti znázorňuje tabulka 2.1. Poznámka: většina funkcionalit je shodná na obou platformách, proto budou tyto funkcionality detailněji popsány pouze jednou (v Android sekci 2.2.1.1).

2.2.1.1 Android

Android aplikace byla vyvinuta v programovacím jazyku Java [3] a již od samého počátku podporovala 3 základní moduly: *Dashboard*, *CRM* a *Tickety* a umožňovala provádět také několik vesměs jednoduchých interakcí.

Dashboard obrazovka sloužila a dodnes slouží především jako statická obrazovka s přehledem denních statistik, aktivit a oznámení uživatele.

OS – svět	%
Android	70.01
iOS	29.24
Samsung	0.43
ostatní	0.32

OS – ČR	%
Android	72.14
iOS	27.27
Samsung	0.5
ostatní	0.09

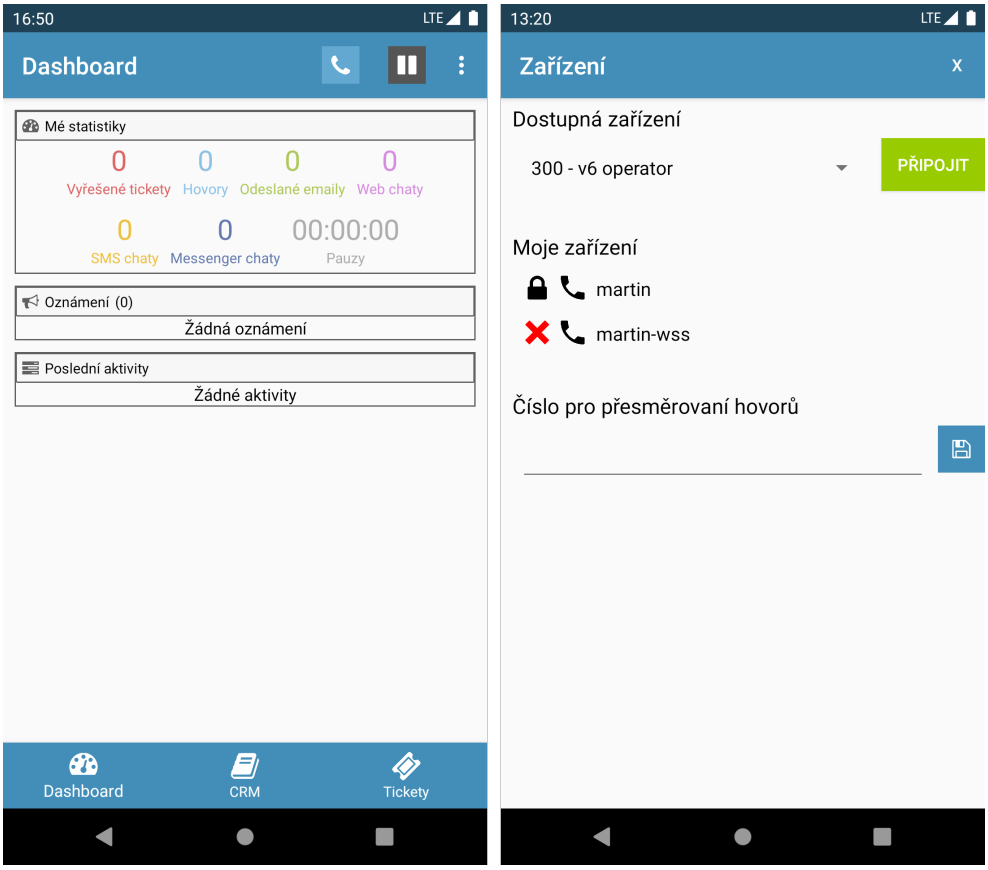
Tabulka 2.1: Operační systémy v telefonech (globálně [1] a v ČR [2])

CRM modul se skládá ze submodulů *Kontakty* a *Společnosti*. Zde je možné ukládat si zákazníky a kategorizovat je podle společností, pro které pracují. Oba submoduly mají kromě základních hodnot (název, vlastník, popis) definovány i základní formuláře (obsahující např. telefonní číslo a e-mail), ale zákazník si může samořejmě definovat i vlastní políčka a sekce formuláře (tato políčka budou dále v práci označována jako *custom fieldy*). Tyto formuláře a jejich políčka typicky definuje zkušenější uživatel systému (administrátor) v modulu *Spravovat*, který však není (až na výjimky) využíván v mobilní aplikaci ani v jiných produktech ekosytému mimo webovou aplikaci (desktopová aplikace, webové rozšíření). Custom fieldy mají několik základních typů: *text*, *textová oblast* (textarea), *select box*, *multi select box*, *check box*, *radio button* a *date/time field*. V pilotní verzi byla všechna políčka v modulu CRM statická a nebylo tedy možné je editovat. Nicméně na některých políčkách bylo možné zavolat základní akce: např. pokud měl kontakt přiřazenou společnost, bylo možné kliknutím na tlačítko přejít na detail společnosti. Oba submoduly obsahovaly také fulltextové vyhledávání.

Jak již bylo zmíněno, modul Tickety je možné třídit do tzv. pohledů. Pohled je specifický objekt, do kterého je možné uložit si vlastní filtrování a řazení ticketů. Dále pak je možné obarvit pohled podle počtu ticketů v něm obsažených (neboli nastavit různé tresholdy). Např. do 5 ticketů v pohledu bude barva zelená, od 5 do 10 oranžová a od 10 červená. Je dokonce možné nastavit si i výchozí hodnoty pro nový ticket, který bude vytvářen z nějakého pohledu (typicky výchozí kategorie nového ticketu). Názorným příkladem využití pohledů můžou být následující pohledy:

- **Moje otevřené** – tickety aktuálně přihlášeného uživatele, které jsou ve stavu *Otevřený*,
- **Moje čekající** – tickety aktuálně přihlášeného uživatele, které jsou ve stavu *Čekající*,
- **Všechny** – tickety bez specifického filtrování řazeny sestupně podle data změny.

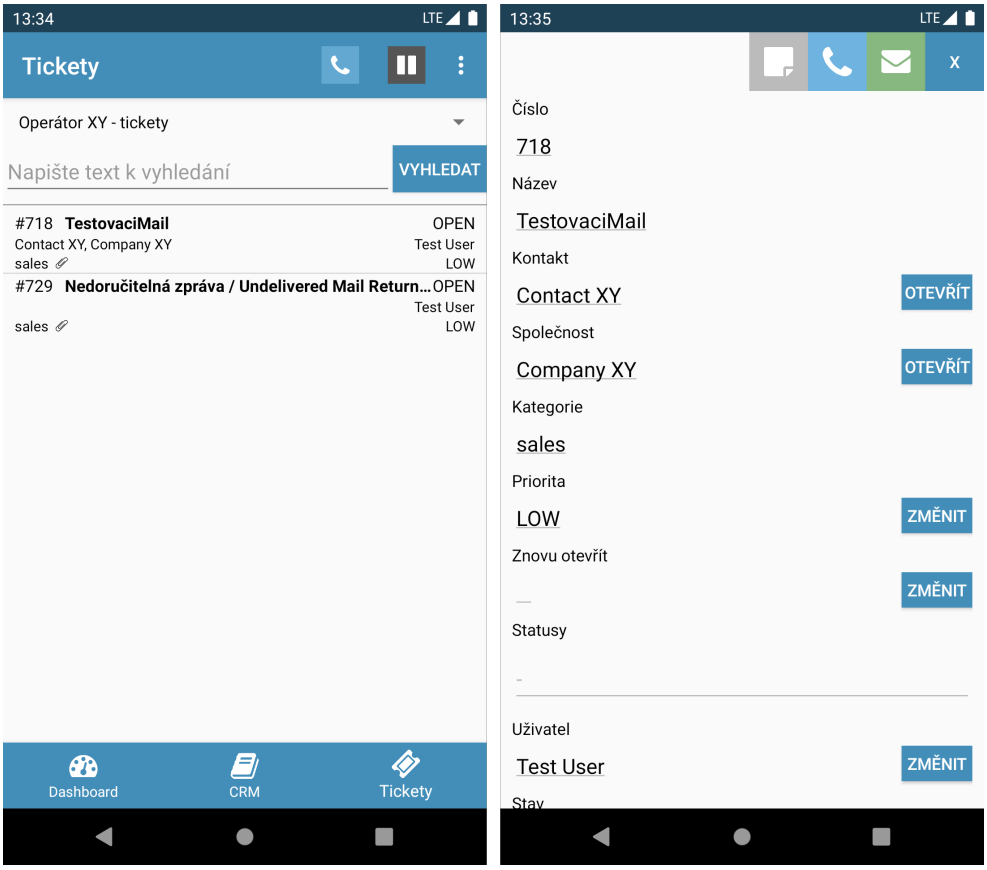
Třídění ticketů do pohledů není nutností, avšak je to téměř samozřejmostí (pokud zákazník pohledy nepoužívá, zobrazí se mu zkrátka všechny tickety



Obrázek 2.1: Pilotní Android verze – obrazovky Dashboard a Zařízení

pohromadě). Pilotní verze aplikace podporovala filtrování pomocí pohledů a také pomocí fulltextového vyhledávání v rámci pohledu (tzn. že fráze zadaná do vyhledávače je vyhledávána v ticketech spadajících do aktuálně zvoleného pohledu). Ticket, podobně jako kontakt nebo společnost, obsahuje základní položky (např. číslo, kontakt, priorita, kategorie, atd.) a dále je pro každou kategorii možné definovat si vlastní formulář (složený z custom fieldlů). Většina těchto informací byla zobrazována staticky bez možnosti interakce, avšak bylo možné prokliknout na detail kontaktu a společnosti (pokud byl ticketu přiřazen kontakt s/bez společnosti). Rovněž bylo možné editovat základní políčka: *priorita* (enumerátor s hodnotami nízká, střední a vysoká), *stav* (enumerátor s hodnotami otevřený, čekající a uzavřený), *uživatel* (seznam dostupných uživatelů systému) a *znovu otevřít* (využívání pro automatické otevření ticketu, který není otevřený – uživatel zadal datum a čas). Tyto úpravy byly prováděny prostřednictvím Android AlertDialogu [4], případně jeho rozšíření jako DatePickerDialog [5] a TimePickerDialog [6].

K ticketu bylo možné ještě přidat aktivitu: napsat komentář, napsat e-



Obrázek 2.2: Pilotní Android verze – obrazovky Tickety a detail ticketu

mail nebo vytvořit hovor. Komentář rovněž prostřednictvím výše zmíněného AlertDialogu a e-mail pak skrze novou obrazovku, na které bylo možné zadat příjemce, předmět a samotný text. Pro text byl vyhrazen HTML editor se základními operacemi (tučně, kurziva, podtržení a barva textu). HTML editor kvůli kompatibilitě s webovou aplikací a dalšími emailovými klienty. Vytvoření hovoru vyžadovalo mít připravené zařízení, které hovor obslouží (mohlo to být tedy i zařízení, které právě obsluhuje aplikaci, avšak byla vyžadována pokročilejší konfigurace, viz dále). Akce e-mailu a hovoru se zobrazily pouze v případě, že byl s ticketem spárován nějaký e-mail nebo telefonní číslo (tzn. byl(o) obsažen(o) v custom fieldech ticketu nebo jeho kontaktu).

Dalšími akcemi, které bylo možné provádět typicky přes horní panel (dále v práci označován jen jako *app bar*), byly:

- Zahájení hovoru (prostřednictvím AlertDialogu pro zadání telefonního čísla).
- Nastavení pauz – pauzy slouží pro dočasné zneaktivnění uživatele pro

příchozí aktivity a mohou být nejen manuální – např. oběd, ale také automatické – např. WRAP nebo Lajdak. Automatiké pauzy jsou nastavovány na základě hlubšího nastavení systému, kde např. WRAP pauza se nastaví operátorovi po ukončení aktivity a je to jakási forma oddechu pro opátora. Naproti tomu pauza Lajdak se nastaví, pokud operátor není v systému aktivní.

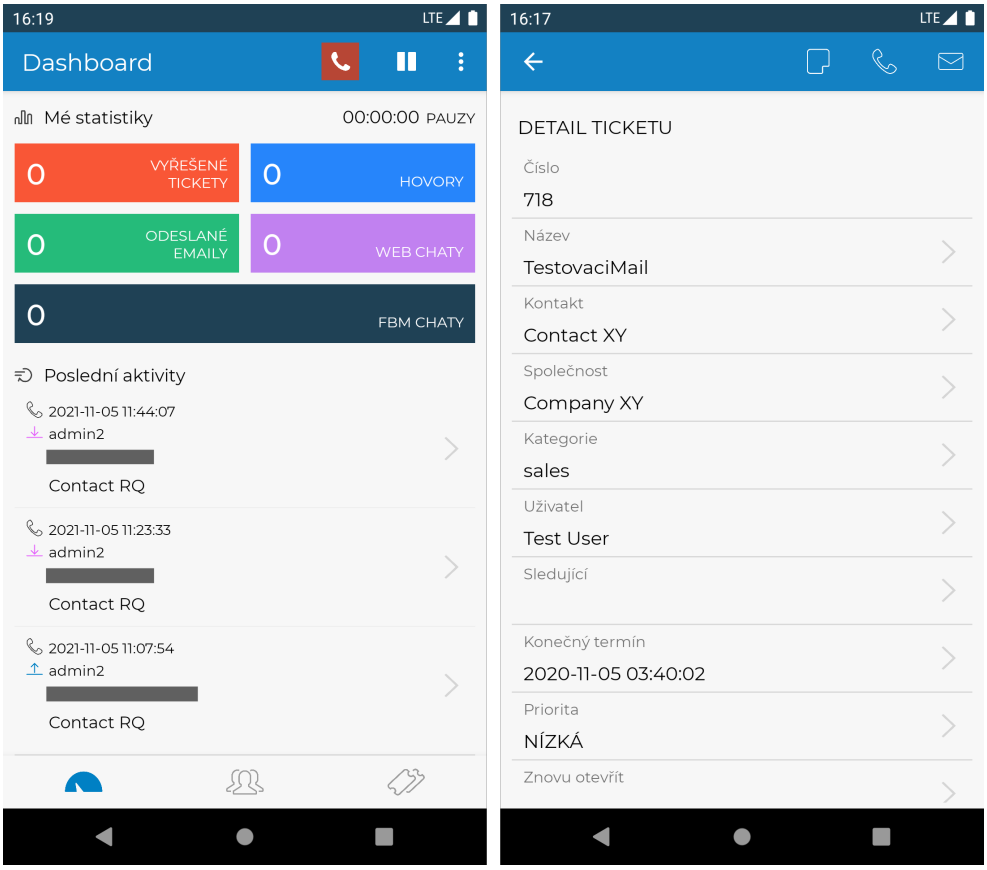
- Správa zařízení – zařízením se rozumí hardwarové nebo softwarové SIP zařízení, které dokáže obsloužit hovor. Příkladem fyzického zařízení jsou IP telefony [7] a softwarového pak aplikace Zoiper [8] nebo MicroSIP [9]. Spárování zařízení se systémem Daktela je pokročilejší úkon a je možné ho (stejně jako např. časy vyzvánení na jednotlivých zařízeních) provést v nastavení webové aplikace. Přes obrazovku Zařízení v mobilní aplikaci bylo možné připojit se k volným zařízením, případně se odpojit od již připojeného.

Pilotní verze Android aplikace byla postupně testována uživateli a ukázalo se, že o aplikaci mají uživatelé zájem (ačkoliv se logicky jednalo o řádově méně početný segment uživatelů než tomu bylo u webové aplikace). Započal tedy iterační vývoj [10] Android aplikace, který postupně vedl až k dnešnímu stavu aplikace. Během tohoto vývoje došlo ke změně/aktualizaci grafického designu a také k přidání řady nových funkcionalit. Mapovat ho budou následující řádky.

Nejzásadnější úpravou z hlediska funkcionalit byla bezpochyby možnost editovat kontakty, společnosti a tickety. Tedy možnost nejen editovat všechna políčka ze základních informací, ale také všechny typy custom fieldů. Jak již bylo zmíněno výše, custom fieldy jsou uživatelem definovaná políčka pomocí kterých je možné sestavit si vlastní formulář. S touto úpravou úzce souvisí také nová možnost vytvoření nového ticketu, kontaktu nebo společnosti.

Daktela je rozsáhlý systém pro různé skupiny uživatelů a zatímco v pilotní verzi nebyla zohledňována oprávnění uživatele, nyní už to byla nutná funkcionalita. Tedy přístupy do jednotlivých sekcí (CRM, Tickety) byly podmíněny nastavením dostatečných oprávnění, stejně tak jako přístupy do případných subsekcí (Kontakty a Společnosti v modulu CRM). Podobně se tato skutečnost odrazila i v případě vytváření nového a editace existujícího ticketu, kontaktu nebo společnosti.

Dále přibyla také práce s přílohami. Nejen, že bylo možné z komentáře nebo e-mailu přílohy stáhnout do telefonu, ale k těmto aktivitám bylo možné přílohy z telefonu i přikládat. Přibyla kontrola neuložených změn (při přechodu na předchozí obrazovku se v případě, že měl uživatel neuložené změny v ticketu/kontaktu/společnosti, se zobrazil AlertDialog, který na tuto skutečnost uživatele upozornil). Vylepšena byla rovněž práce s aktivitami: byla přidána zcela nová obrazovka Aktivity pro zjednodušenou správu otevřených nebo čekajících aktivit. Tyto aktivity bylo nově možné uzavřít (včetně vyplnění statusu). Jednalo-li se o hovor, pak ho bylo možné přijmout/ukončit (s ohledem



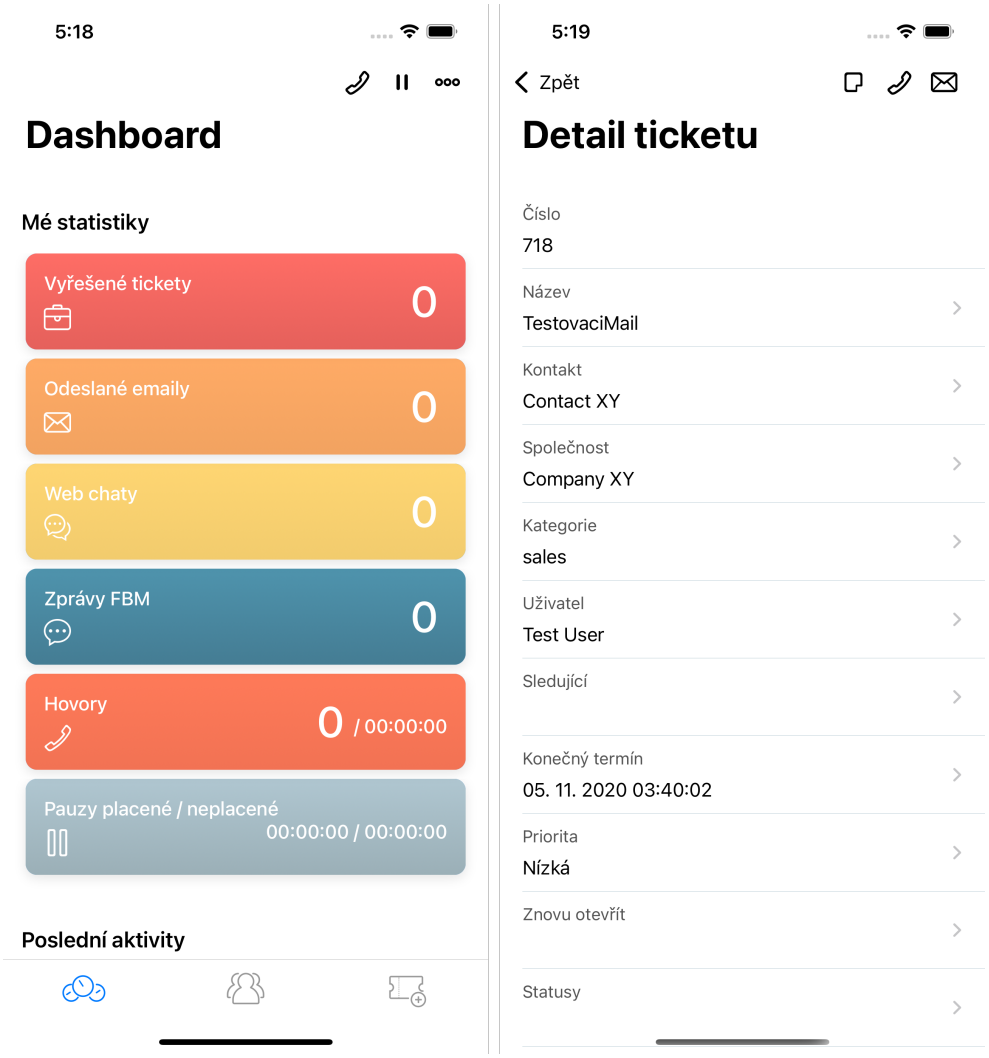
Obrázek 2.3: Ukázka dnešní Android aplikace – obrazovky Dashboard a detail ticketu

na zařízení, na kterém hovor probíhá). Status je jakýsi tag umožňující třídění aktivit podle jejich výsledku (např. nemá zájem, volat později, lead). Má-li fronta příslušné aktivity nastaveny statusy, pak je třeba před uzavřením aktivity nějaký status vyplnit. Frontou se rozumí jakýsi konfigurátor pro komunikaci příslušným kanálem, který umožňuje nejrůznější nastavení (přesahující rozsah této práce). Velmi zjednodušeně se dá říci: aby mohl uživatel obsluhovat (komunikovat přes) nějaký kanál, musí být přihlášený v nějaké jeho frontě.

2.2.1.2 iOS

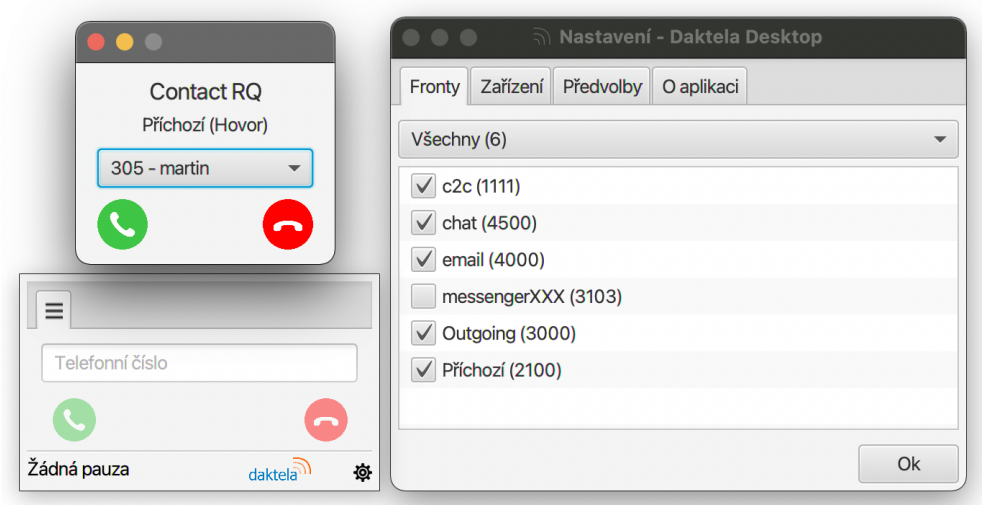
Dnešní iOS aplikace produktu Daktela V6 se velmi podobá výše zmíněné Android aplikaci. Obě aplikace pokrývají v zásadě stejné funkcionality, vzhled aplikace je ovšem mírně odlišný. Aplikace pro Apple zařízení byla vyvíjena v populárním programovacím jazyku Swift [11] a samotný vývoj započal nedlouho po té, co vyšla pilotní verze Android aplikace.

Aplikace zpočátku uměla obsluhovat pouze základní úkony a sloužila spíše



Obrázek 2.4: Ukázka dnešní iOS aplikace – obrazovky Dashboard a detail ticketu

ke zobrazení základních informací (podobně jako tomu bylo zpočátku u Androidu). Tedy zobrazení základních statistiky na Dashboard obrazovce a pak prohlížení kontaktu a společností v modulu CRM a prohlížení ticketu. Nebylo možné s moduly nijak interaktivně pracovat (například editovat tickety). Postupnými úpravami a přidáváním nových funkcionalit se podařilo aplikaci dostat na stejnou úroveň z pohledu funkcionalit jakou měla Android verze. Tedy bylo možné především provádět CRUD operace s moduly CRM a Tickety. Mimo to aplikace zahrnovala také základní práci s aktivitami – kromě jejich zobrazení/čtení, uživatel mohl také napsat e-mail nebo komentář k ticketu a také si zavolat (za předpokladu, že měl aktivní nějké SIP zařízení – viz výše).

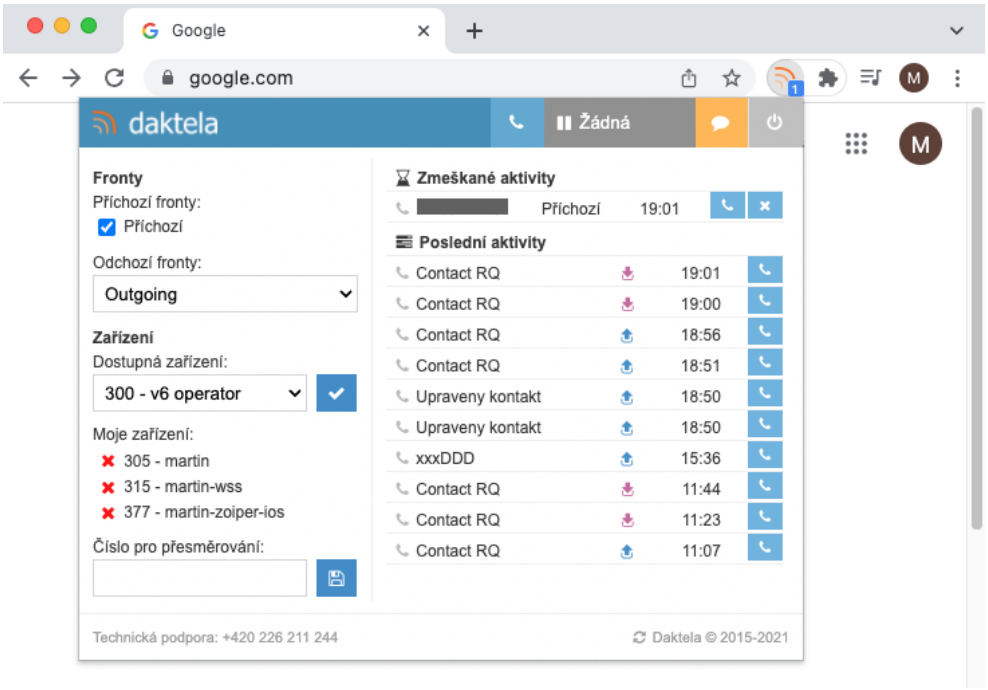


Obrázek 2.5: Ukázka dnešní aplikace Daktela Desktop

2.2.2 Desktop aplikace

Desktop aplikace, tedy aplikace nainstalovaná jako spustitelný program v operačním systému počítače, je nejnovější produkt ekosystému Daktela. Účelem tohoto produktu není pokrýt co nejširší spektrum funkcionalit, spíše naopak. Je to aplikace určená například pro operátory, kteří při práci používají i další informační systémy. Vyvinuta byla v jazyce Java [3] a využívá grafickou knihovnu JavaFX [12], kterou společnost Oracle od verze JDK 11 (Java Development Kit) poskytla jako open source.

Aplikace je velmi minimalistická, její okno je co možná nejmenší a slouží především k vytáčení a obsloužení příchozích hovorů. Okno navíc zůstane „vždy navrchu“ obrazovky (always on top), nepřekruje ho tedy jiná aplikace. K vytáčení hovorů slouží jednoduchý asynchronní našeptávač CRM kontaktů/společností z klientské aplikace. Je možné přijmout nebo odmítnout také jiné než hovorové aktivity (např. chaty). Pokud opertáor přijme chat prostřednictvím aplikace, může skrze tlačítko *Otevřít v Daktele* otevřít aktivitu ve webové aplikaci, která chatování podporuje. Přes nastavení aplikace se operátor může přihlásit/odhlásit do/z front. A dále může spravovat svá SIP zařízení. Tyto moduly nastavení jsou identické s moduly mobilních aplikací. V předvolbách je také možné povolit/zakázat zvukovou notifikaci na příchozí aktivitu. Aplikace zahrnuje i základní uzavírání otevřených aktivit pomocí jednoduchého tab baru (třída TabPane [13] knihovny JavaFX). Není-li aktivita (respektive její fronta) spárována se žádným formulářem (případně jsou-li jediným formulářovým políčkem statusy), pak je možné aktivitu v desktop aplikaci i uzavřít. V neposlední řadě se může operátor také „zapauzovat“, aby na něho nebyly směrovány příchozí aktivity.



Obrázek 2.6: Ukázka dnešního webového rozšíření Daktela

2.2.3 Webové rozšíření

Webové rozšíření (Mozilla terminologií *add-on*) umožňuje upravovat a vylepšovat možnosti prohlížeče [14]. Aplikace samotná může být vytvořena v jakékoliv webové technologii, ale pro pokročilejší funkcionality a kooperaci se samotným prohlížečem je nutné využít Mozilla API [15], případně Chrome API [16] pro Chromium-based prohlížeče (Google Chrome, Opera, Microsoft Edge). Obě rozhraní se od sebe mírně liší. Namátkou:

- různé názvy namespaců (Mozilla má `browser`, Chrome pak `chrome`),
- callbacks vs. promises (Mozilla vs. Chrome).

Mozilla nicméně zavedla podporu i pro výše zmíněné rozdíly a vytvořila dokonce i *polyfill* (část kódu, která umožňuje využívat moderní funkcionality ve starších prohlížečích, které je nativně nepodporují [17]) umožňující překonvertovat webové rozšíření využívající Mozilla namespace `browser` na namespace `chrome`.

Webové rozšíření Daktela má podobný účel jako již zmíněná desktop aplikace, tedy umožnit uživatelům kromě hlavního produktu Daktela využívat také další (doplňkový) systém – v tomto případě webový. V podstatě celé nastavení desktop aplikace je v levé části hlavní obrazovky webového rozšíření

(viz obrázek 2.6). V její pravé části jsou pak otevřené, zmeškané a poslední aktivity (např. widget posledních aktivit v desktop aplikaci není obsažen vůbec). Z rozšíření je možné vytočit hovor, ale také poslat SMS zprávu (uživatel kromě příjemce a samotného obsahu vybere také frontu, prostřednictvím které bude zpráva odeslána, a případně status). V aplikaci pochopitelně nechybí základní uživatelské úkony jako nastavení pauzy, či možnost se odpojit nebo odhlásit. Zajímavostí je, že pro přihlášení do webového rozšíření se využívá session klientské aplikace. Je-li tedy uživatel přihlášen do webové aplikace Daktela, je přihlášen také do webového rozšíření (má-li ho nainstalované). Výše zmíněného Mozilla/Chrome API rozšíření využívá například při prvotním nastavení aplikace, kdy je třeba nastavit URL klientské Daktela aplikace. S pomocí API je možné toto provést přes standardní nastavení rozšíření. Např. v prohlížeči Google Chrome se do něho uživatel dostane tak, že na ikonu rozšíření klikne pravým tlačítkem a následně vybere *Možnosti*. Následně se otevře stránka s informacemi o verzi, velikosti a dalších možnostech. Jednou z nich je také stránka Možnosti rozšíření, kterou implementují vývojáři rozšíření.

2.3 Konkureční řešení a podobné práce

Řešení pro call centra jsou téměř výhradně licencovaná a přístupná pouze s aktivní licencí. Existuje mnoho takových řešení, které poskytují různé typy služeb a modulů většího či menšího rozsahu. Popsat tato řešení je mimo rozsah této práce, nicméně relevantí informací je, jaké technologie používají aplikace (viz tabulka 2.3). Nutno však říci, že odhad použitých technologií je orientační, protože ke zdrojovým kódům autor pochopitelně přístup nemá. Prázdná buňka značí, že aplikace na příslušné platformě není podporována, případně není dostupná. Jak je vidět, mobilní aplikace řešení ClouldTalk a 8x8 využívají multiplatformní framework React Native (popsaný v subsekci 2.5.2), avšak většinou je využívano spíše nativních knihoven a frameworků dané platformy. Pozn. PWA (progresive web app) je webová aplikace (typicky napsaná ve standardních webových technologiích jako HTML, JavaScript a CSS) uzpůsobená tak, aby dokázala běžet i na newebových platformách (desktop nebo mobilní aplikace).

Multiplatformním vývojem se zabývá také mnoho odborných prací, ačkoliv většina z nich se zaměřuje pouze na mobilní aplikace.

Jednou z nich je diplomová práce [18], která popisuje vývoj mobilní aplikace ve frameworku Flutter. V práci jsou nejprve vyřčeny rozdíly mezi nativním a multiplatformním vývojem. Následně jsou popsány multiplatformní frameworky s důrazem právě na Flutter, který je z nich (Xamarin, Ionic, React Native) nejnovější (pozn. multiplatformními technologiemi se zabývá

Aplikace	mobilní	desktop	webové rozšíření
Zendesk	nativní	–	–
CloudTalk	React Native	PWA	nativní
Talkdesk	nativní	PWA	nativní
Five9	nativní	–	nativní
8x8	React Native	PWA	nativní

Tabulka 2.2: Přehled konkurenčních řešení a odhad použitých SDK/frameworků v jejich aplikacích na různých platformách

sekce 2.5). Dále autor práce rozebírá možné vývojové přístupy ve Flutteru, přičemž tyto přístupy jsou vesměs založeny na asynchronním programování založeném na událostech a observerech (pozorovatel), kde jsou tyto události odeslány, následně zpracovány a konečně jsou notifikovány objekty (nejčastěji widgety), které zajímá výsledek. V práci autor demonstruje použití knihoven `flutter_bloc`, `flutter_redux` a `flutter_control`. Posledně jmenovaná knihovna je komplexnější a nabízí např. i možnost práce s lokalizacemi či s routovaním, avšak z dnešního pohledu není až tak populární. Zbývající knihovny jsou méně komplexní a jsou si velmi podobné. Jsou také populárnější a mohou se pyšnit i odznakem *Flutter Favorite,* který na základě několika metrik označuje knihovny (např. skóre aplikace, kvalita dokumentace, licence, atd.), které by měl vývojař upřednostnit (proces udělení tohoto odznaku zajišťuje *Flutter Ecosystem Committee* – skupina lidí kolem Flutter týmu a komunity).

Podobným tématem (s větším důrazem na samotnou implementaci multiplatformní aplikace) se zabývá i diplomová práce [19]. Práce přehledně shrnuje rozdíly mezi multiplatformními frameworky, přičemž autor kromě jiného vysvětluje rozdíl mezi UI komponentami frameworků Flutter a React Native. Zajímavým poznatkem je, že zatímco Flutter má vlastní UI komponenty, které se vykreslí na obrazovce pomocí Flutter engine, React Native své UI komponenty mapuje na nativní komponenty dané platformy. Podrobně zmapován je programovací jazyk Dart i základní principy a komponenty frameworku Flutter. Praktická čast práce zaznamenává implementaci aplikace pro vykazování času odpracovaného na projektech. Pro oddělení aplikační logiky od prezentační vrsty autor používá výše zmiňovaný návrhový vzor BLoC a jeho Flutter knihovnu `flutter_bloc` (pozn. tímto vzorem a vývojovým přístupem se zabývá sekce 3.2). Sám autor uvádí, že aplikace obsahuje 7 hlavních obrazovek a několik podpůrných a je tedy spíše menšího rozsahu. Pro komunikaci se serverem používá aplikace předem definované API, čímž se autorovo dílo v jistém smyslu podobá této diplomové práci. Autor během vývoje narazil na problém se zobrazováním kontextového menu (způsobující pád aplikace), nutno ovšem říci, že v době implementace aplikace nebyla první stabilní verze Flutteru stará ani rok. Pozn. v roce 2021 vyšla druhá generace framewroku –

Flutter 2 – a v únoru 2022 pak již 5. major této generace (2.10.0).

2.4 Funkční a nefunkční požadavky zadavatele

Ačkoliv se jednotlivé aplikace Daktela ekosystému liší funkcionalitami, jejich jádro zůstává vždy stejné. Stejné je grafické rozložení prvků, UX i UI se napříč aplikacemi podobají. Vcelku logicky se tedy zrodila myšlenka, zda by nebylo možné vývoj výše zmíněných aplikací sjednotit. Technologie, které umožňují vývoj pro více platforem popisuje následující sekce 2.5.

Tato práce pokrývá především přepracování mobilní aplikace, proto budou uvedeny požadavky zadavatele na mobilní aplikaci. Ty jsou v zásadě velmi prosté – výsledná mobilní aplikace aplikace by měla pokrýt všechny funkcionality, které pokrývaly předchozí (implementačně ještě oddělené) aplikace pro platformy Android a iOS.

To znamená (vyjma možnosti přihlášení a odhlášení) pokrýt 3 základní moduly Dashboard, CRM a Tickety a jejich dosavadní funkcionality.

- **F1** U modulu Dashboard jde predevším o základní widgety ze systému Daktela: Mé statistiky, Poslední aktivity a Oznámení.
- **F2** Modul CRM by pak měl zahrnovat přehled a všechny CRUD operace s kontakty a společnostmi, včetně práce s custom fieldy. U každého kontaktu/společnosti by také měl být přehled jeho aktivit a ticketů.
- **F3** V modulu Tickety by měl být dostupný přehled ticketů dle aktuálně zvoleného pohledu uživatele. Podobně jako u CRM modulu je i zde nutná implementace všech CRUD operací včetně práce s custom fieldy a změny kategorie ticketu (samozřejmě s ohledem na přístupy a práva uživatele). U každého ticketu se pak budou zobrazovat také jeho aktivity.
- **F4** Dále je vyžadována práce s aktivitami minimálně na úrovni hovorů a e-mailů (to znamená být schopný vytočit hovor přes aplikaci, napsat/přečíst e-mail) napříč aplikací s logickými návaznostmi podle aktuálního modulu. Tedy například v detailu ticketu bude možné přidat také komentář včetně možnosti nahrát či stáhnout přílohu.
- **F5** V neposlední řadě by měl mít uživatel také možnost změnit svůj status (nastavit si pauzu nebo se odpojit). S tím úzce souvisí také správa přihlášení do front a také správa uživatelských zařízení.

Nefunkční požadavky pak zahrnují:

- **N1** Grafické vylepšení aplikace (po vzoru redesignu webové aplikace Daktela ve verzi 6.20), kde je samozřejmostí také responzivní design celé aplikace. Mělo by tedy být podporováno zobrazení na výšku i na šířku a to včetně zobrazení na větších displejích (např. tablet).

- **N2** Dále poskytnutí (zachování) integrace s překladatelským nástrojem Crowdin [20]. Nástroj Crowdin umožňje prostřednictvím webového klienta překládat textové řetězce do různých jazyků.
- **N3** Podobně by také měl být integrován nástroj Firebase, respektive jeho Crashlytics [21] a Analytics [22], pro monitorování a evidenci chyb a také pro sbírání základních statistik o uživatelském způsobu používání aplikace.

2.5 Technologie

V současnosti existují v zásadě dva přístupy k vývoji mobilních aplikací: nativní a multiplatformní. Každý z nich má své výhody a nevýhody a stejně tak početnou skupinu zastánců i odpůrců. Primární rozdíly mezi oběma přítupy shrnuje např. článek [23] z webu `netsolutions.com` a jsou to:

- cena – vývoj pro nativní platformy je řádově dražší (už jen proto, že je nutné vyvinout minimálně dvě aplikace pro Android a iOS – dnes je prakticky nemyslitelné, že by jakákoliv aplikace nebyla dostupná pro některou z těchto platforem),
- použitelnost kódu – multiplatformní framework umožňuje používat jednotný kód (což samozřejmě zrychluje vývoj a zjednodušuje následnou údržbu),
- přístup k nativním funkcionalitám – nativní SDK konkrétní platformy umožňuje bezproblémový přístup k API zařízení, zatímco u multiplatformních frameworků tento přístup není garantovaný a jeho implementace může být problematická (potenciálně tak může nastat problém s hardwarovou kompatibilitou),
- konzistence UI a UX – uživatelé jsou zvyklí na určité uživatelské rozhraní a odezvu podle používaného operačního systému, což je potenciální problém u multiplatformních frameworků (komponenty mohou být konzistentní pouze částečně, případně je nutné rozhodnout, které platformě bude výsledná aplikace „bližší“),
- výkon – u nativních aplikací bez problému, protože aplikace je vyvinuta pro cílovou platformu zařízení, nicméně rychlé jsou i multiplatformní frameworky, takže z pohledu výkonu jsou oba přístupy srovnatelné.

Před samotným rozhodnutím, zda aplikaci vyvíjet v nativních technologiích nebo v multiplatformním frameworku, je tedy nutné zvážit minimálně výše zmíněné body. Pro aplikace, které nelpí primárně na nativních funkcionalitách (nebo jsou-li jejich vývojáři smíření s potenciálními problémy při

Framework	2019 (v %)	2020 (v %)	2021 (v %)
Flutter	30	39	42
React Native	42	42	38
Cordova	29	18	16
Ionic	28	18	16
Xamarin	26	14	11

Tabulka 2.3: Nejpoužívanější multiplatformní frameworky pro mobilní vývoj v letech 2019-2021 (v %) podle JetBrains [24] [25] [26]

jejich integraci) může být multiplatformní vývoj správnou volbou. Stejně tak pro menší aplikace, případně pro start-upy a menší firmy. Vývoj aplikace je rychlý a za řádově nižší náklady. Doba dodání výsledné aplikace (TTM – time to market) je také rychlá, což může být důležitý faktor právě pro start-upy (potřebující prostředky pro další rozvoj). Dalším případem, kdy multiplatformní framework stojí za zvážení, jsou aplikace, které cílí i na web, případně mají být instalovatelné (desktop aplikace).

V současné době existuje mnoho frameworků umožňujících multiplatformní vývoj, predevším pak frameworků pro multiplatformní mobilní vývoj. V něm v posledních letech dominují především frameworky React Native a Flutter. Statistiky společnosti JetBrains, která se zabývá vývojem nástrojů a produktů pro softwarové vývojáře, jasně ukazují vzestupný trend Flutteru, který v roce 2021 React Native dokonce již překonal a se 42 procentními body je aktuálně nejpouživanějším multiplatformním mobilním frameworkem (tzn. 42 % vývojářů, kteří vyplnili dotazník, používalo v roce 2021 framework Flutter). Sesbíraná data znázorňuje tabulka 2.3. Dotazníku se postupně v letech 2019 [24], 2020 [25] a 2021 [26] zúčastnilo přibližně 7, 19,5 a 31 tisíc vývojářů. Podle dat sesbíraných v roce 2021 navíc plyne, že více než polovina (53 %) mobilních vývojářů používá multiplatformní frameworky.

Další zajímavé informace přinášejí roční dotazníky populárního webu pro vývojáře `stackoverflow.com`. Mezi ostatními (newebovými) frameworky/technologiemi vládne Node.js. React Native postupně v letech 2019 (58 543 respondentů [27]), 2020 (40 314 respondentů [28]) a 2021 (59 921 respondentů [29]) používalo 10,5 %, 11,5 % a 14,51 % vývojářů. Ve stejném období framework Flutter vzrostl z 3,4 % v roce 2019 na 13,55 % v roce 2021. Téměř tak dohnal React Native a nebylo by velkým překvapením, pokud by ho v přístím roce předehnal. Apache Cordova se stabilně drží mezi 6-7 procentními body, zatímco používání Xamarinu postupně klesá (z 6,5 % v roce 2019 na 3,9 % v roce 2019). Data jsou přehledně znázorněna v tabulce 2.4.

Ze zmíněných framemorků se navíc Flutter opakovaně umísťuje na předních příčkách ve statistice nejoblíběnějších (most loved) newebových frameworků

Technologie	2019 (v %)	2020 (v %)	2021 (v %)
React Native	10,5	11,5	14,51
Flutter	3,4	7,2	13,55
Cordova	7,1	6,0	7,18
Xamarin	6,5	5,8	3,9

Tabulka 2.4: Nejpoužívanější ne-webové technologie podle Stack Overflow v letech 2019-2021 (v %) – pozn. vypsány pouze multiplatformní technologie [27] [28] [29]

– pozn. podle statistik `stackoverflow.com`. Tato statistika mapuje vývojáře, kteří mají zkušenosti s vývojem v daném frameworku a vyjádřili zájem používat tento framework i dále.

2.5.1 Flutter

Flutter je open source framework [30] vyvinutý společností Google, který vyšel v roce 2017. Jeho vývoj byl oznámen na konferenci v roce 2015 [31] (tehdy ještě pod označením Sky). Framework stejně jako aplikace v něm vytvořené využívají programovací jazyk Dart, taktéž z dílny Google. Core engine (Flutter Engine), který zajišťuje vykreslování jednotlivých stránek je pak napsán primárně v jazyce C++. Navzdory tomu, že je tento framework relativně nový, těší se mezi vývojáři značné popularitě (jak ostatně dokumentují statistiky výše).

Základní stavební jednotkou je *widget*, který se dělí na bezstavový (*stateless*) a stavový (*stateful*). Widget deklaruje své uživatelské rozhraní přetížením metody `build`. Jak název napovídá, stateless widget si neuchovává žádnou vlastnost (stav), kterou je možné změnit v čase – například ikona nebo label. Naproti tomu stateful widget si dokáže pamatovat nějakou informaci a překreslí se, když dojde k její změně. Příkladem může být jednoduché počítadlo kliků – po kliknutí na tlačítko se k aktuálnímu počtu kliků přičte 1. K překreslení stránky zajistí metoda `setState`. Ve Flutteru je každá komponenta widget. To poskytuje prakticky neomezené možnosti, jak poskládat obrazovku aplikace (widgety je možné vkládat do sebe).

Na iOS a macOS se Flutter načte do embedderu jako UIViewController [32]. Embedder platformy vytvoří FlutterEngine, který je hostitelem Dart VM a Flutter aplikace, a FlutterViewController, který se připojí k FlutterEngine, aby předával UIKit nebo Cocoa vstupní události do Flutter aplikace a aby zobrazoval snímky, které vytvoří FlutterEngine.

V Androidu je Flutter načten do embederru jako aktivita (třída Activity [33]). Zobrazení řídí FlutterView, který vykresluje obsah jako view nebo jako texturu.

Flutter nabízí pro vývojáře také tzv. *hot reload*, který načte změny v pro-

vedené ve (stateful) widgetu bez nutnosti rekompilace celé aplikace. To se hodí např. při jednoduchých úpravách widgetu (odsazení, velikost textu).

V mobilních a desktop aplikacích napsaných ve Flutteru je možné zavolat nativní kód prostřednictvím kanálu platformy (komunikace mezi Dart kódem a kódem hostitelské platformy – např. Swift nebo Kotlin). Prostřednictvím vlastního kanálu je možné komunikovat – odesílat a přijímat zprávy – mezi Flutter aplikací a nativní aplikací. Data jsou serializována prostřednictvím Dart třídy Map (obdoba HashMap v Kotlinu nebo Dictionary ve Swiftu).

Flutter podporuje také vytváření webových aplikací. K tomu bylo nutné naimplementovat nový engine a v současnosti je podporováno vykreslovaní pro web prostřednictvím HTML a WebGL.

Za zmínku také stojí, že Flutter používá deklarativní styl zápisu kódu [34], namísto imperativního stylu, který je při vývoji pro Android/iOS běžnější. Pro změnu barvy widgetu (např. `Box`) imperativním stylem je nejdříve třeba najít jeho instanci prostřednictvím selektoru (např. `findViewById`) a následně tuto barvu nastavit prostřednictvím setteru (`box.setColor(Colors.red)`). Podobná situace nastává při přidávání potomka k widgetu. Nejdříve je třeba vytvořit jeho instanci (např. `View c = new View(...)`) a následně ji skrze metodu (např. `box.add(c)`) přidat. Naproti tomu v deklarativním zápisu jsou jednotlivé widgety neměnné – pro změnu UI je nutné widget rebuildovat (např. `setState`) a následně se vytvoří nové instance widgetů. Je tak možné konstruovat strom widgetů (tedy přesně to, co ve výsledku uvidí uživatel na obrazovce). Viz následující ukázka kódu 2.1.

```
return Box(
  color: Colors.red,
  child: View(...),
)
```

Ukázky kódu 2.1: Ukázka deklarativního zápisu v Dartu

2.5.2 React Native

S React Native přišel Facebook (pozn. dnes Meta) v roce 2015 [35]. Vychází z Reactu [36], což je javascriptová knihovna pro vytváření uživatelských rozhraní. React byl původně vyvíjen pouze pro interní potřeby Facebooku, projevili však o něj zájem i vývojáři Instragramu (po té, co Facebook v roce 2013 zakoupil tuto sociální síť). Vývojová větev Reactu se tak oddělila od vývoje Facebooku a v roce 2013 byl celý React projekt open sourcován veřejnosti. React Native umožňuje vývojářům psát kód v populárním jazyku JavaScript [37] (průzkumy JetBrains [26] resp. StackOverflow [29] za rok 2021 označují JavaScript za nejpopulárnější programovací jazyk – využívalo ho 69 % resp. 64,96 % vývojářů) aplikace kompilovatelné nejen pro Android a iOS, ale také pro další (webové a desktopové) platformy.

Komponenta je základní stavební prvek (widget ve Flutteru) a je možné ji vytvořit klasicky za pomocí třídy (která dědí ze třídy `Component`) a přetížit metodu `render` a nebo lze využít také funkčního přístupu (tzv. hooků), které byly představeny ve verzi 0.59 (a vycházejí z React Hooks API [38]) [39]. Rozdíl mezi oběha přístupy demonstruje následující ukázka kódu 2.2.

```
// class
class Box extends Component {
  render() {
    return (
      <Text>Hello, world!</Text>
    );
  }
}

// hooks
const Box = () => {
  return (
    <Text>Hello, world!</Text>
  );
}
```

Ukázky kódu 2.2: React Native - class vs. hook

React Native za běhu aplikace namapuje komponenty vytvořené v Reactu na Android/iOS „pohledy“ (anglický výraz view je názornější). Základní množina takových komponent je v React Native terminologii označována jako *Core Components* [40]. Core komponent je mnoho, nicméně mezi ty nejzákladnější patří například:

- `View` – ekvivaletní `ViewGroup` v Androidu, `UIView` v iOS a HTML tagu `div`,
- `Text` – ekvivaletní `TextView` v Androidu, `UITextView` v iOS a HTML tagu `p`,
- `ScrollView` – ekvivaletní `ScrollView` v Androidu, `UIScrollView`.

2.5.3 Xamarin

Podobně jako Flutter a React Native je i Xamarin open source projekt, který dává vývojářům možnost psát aplikace pro Android, iOS a Windows pomocí frameworku .NET [41]. Xamarin je abstraktní vrstva, která spravuje komunikaci mezi sdíleným (společným) kódem a nativním kódem dané platformy. Microsoft uvádí, že Xamarin umožňuje vývojářům sdílet průměrně 90 % kódu

napříč platformami. Je tedy možné napsat aplikační (business) logiku aplikace v jednom jazyce.

Xamarin poskytuje knihovny Xamarin.Android, Xamarin.iOS a Xamarin.Mac, které umožňují .NET vývojářům napsat nativní aplikaci pro Android, iOS a macOS prostřednictvím C# (knihovny pokrývají kompletní Android, iOS a macOS SDK) [42]. Zajímavější z pohledu multiplatformního vývoje je pak knihovna Xamarin.Forms, která abstrahuje uživatelské rozhraní od jednotlivých platforem. Podobně jako předešlé frameworky má i Xamarion.Forms sadu základních elementů, které se překonvertují na nativní elementy určité platformy (např. `Label` převede na iOS `UILabel`). Nespornou výhodou tohoto frameworku je, že vývojářům „stačí“ znát C# a .NET a mohou ihned začít s multiplatformním vývojem.

2.5.4 Cordova

Apache Cordova je open source framework umožňující vývojářům využívat standardní webové technologie HTML5, CSS3 a JavaScript pro multiplatformní vývoj (je tedy vhodný např. pro webové vývojáře). Samotné aplikace jsou zabaleny do modulu specifického pro cílovou platfomu, prostřednictvím kterého lze přistupovat k funkcím zařízení (senzory, stav sítě, atd.).

Architektura Cordova aplikace je rozdělena do tří komponent: webová aplikace (web app), WebView a Cordova pluginy [43]. První komponentou je standardní webová aplikace, obsahuje tedy *index.html*, který pak dále referencuje další soubory (jako CSS, JavaScript, média, atd.). Nachází se zde i konfigurační soubor *config.xml* obsahující např. jméno a verzi aplikace. WebView zajišťuje zobrazení (vykreslení) aplikace na cílovém zařízení (na některých platformách může být dokonce komponentou větší hybridní aplikace, která míchá WebView a nativní komponenty). Cordova pluginy jsou rozhraní (zajišťující komunikaci) mezi samotnou aplikací a nativními službami zařízení (geolokace, fotoaparát, atd.). V aplikacích je tak možné prostřednictvím JavaScriptu zavolat služby nativního API konkrétního zařízení.

2.5.5 Ionic

Stejně jako Cordova je i Ionic open source framework založený na webových technologiích [44]. Je tak možné vyvíjet mobilní a desktop aplikace prostřednictvím HTML, CSS a JavaScriptu. Nicméně umožňuje také integraci s populárními javascriptvými frameworky jako Angular, React a Vue. Pro definici layoutu (rozložení) stránky framework využívá (v čistém JavaScriptu) vlastní tagy, např. uvnitř tagu `ion-toolbar` se definují jednotlivé položky toolbaru. Toolbar se na obrazovce zobrazí buď nahoře nebo dole v závisloti na tom, v jaké kořenové komponentě se nachází (`ion-header` nebo `ion-footer`). Zbývající základní kořenovou komponentou je pak `ion-content`, která je určena pro samotný obsah stránky (v této komponentě je možné scrollovat). Tyto tři

kořenové komponenty jsou zabaleny do tagu `ion-app`, který zapouzdřuje celou Ionic aplikaci.

2.6 Zvolená technologie

Autor práce si z výše popsaných technologií zvolil ke zhotovení produktu framework Flutter a to hned z několika důvodů:

- jedná se relativně nový a mezi vývojáři velmi populární framework (viz kapitola Technologie 2.5),
- vývojáři na něm aktivně pracují a neustále ho vylepšují – za posledních 12 měsíců (březen 2021 až únor 2022) vyšlo 5 stabilních verzí (od 2.0.0 až po 2.10.0) [45],
- je schopný zkompilovat aplikaci na všech požadovaných platformách (mobilní, webová i desktop aplikace),
- na rozdíl od React Native má Flutter zcela vlastní widgety (komponenty), které nemapuje na ty nativní. To nejen snižuje závislost na dané nativní platformě, ale ulehčuje samotné stylování komponent, které může být v nativních aplikacích komplikovanější.

2.6.1 Dart

Programovací jazyk Dart [46] je silně typovaný programovací jazyk v jistém smyslu podobný Javě nebo Kotlinu. Navzdory tomu je psaní typových anotací volitelné, protože je Dart schopný datový typ vydedukovat. Vše, co je možné uložit do proměnné, je v Dartu objekt (tedy i čísla, funkce a `null`). Všechny objekty vyjma `null` dědí ze základní třídy `Object`.

Při používání tzv. *null safety* není možné do proměnné uložit `null` hodnotu, dokud to programátor nedovolí – tak, že přidá otazník za název datového typu (např. `Object?`). Pokud si je programátor jistý, že v *nullable* proměnné není hodnota `null`, může hodnotu přetypovat na *non-nullable* datový typ prostřednictvím vykřičníku (např. `Object x = prevVal!`).

Podobně jako mnoho jiných jazyků podporuje i Dart tzv. *lazy* inicializaci (klíčové slovo `late`). Kromě standardního použití, kdy k inizicializaci takové proměnné dojde až když je tato proměnná potřeba, je možné tuto vlastnost využít pro deklaraci *non-nullable* proměnné, k jejíž inicializaci dojde až posléze.

Dart nepodporuje (na rozdíl například od Javy) klíčová slova `public`, `protected` a `private`. Na místo toho je možné identifikátor (nejen proměnné nebo funkce, ale také třeba název třídy) prefixovat podtržítkem, které značí, že daný identifikátor je privatní v rámci své knihovny. V praxi to znamená, že je takový identifikátor dostupný pouze v souboru, ve kterém byl definován.

Zajímavá je i tzv. *kaskádová notace*, která umožňuje provést s objektem více operací najednou. Slouží k tomu operátor dvou teček (**..**, příp. pro nullable objekty **?..**).

```
var fullName = User()
  ..firstName = 'Test'
  ..lastName = 'User'
  ..fullName;
```

Ukázky kódu 2.3: Kaskádová notace v Dartu

V Dartu jsou k dispozici také jmenné parametry, které jsou definovány ve složených závorkách v parametrech metody, ale třeba také konstruktoru třídy. Klíčovým slovem `required` je pak možné označit jmenný parametr jako povinný. Volitelné poziční parametry jsou naopak vkládány do hranatých závorek.

Pomocí rozšiřovacích metod (angl. *extension methods*) je možné dodefinovat funkcionality (metody) k již definovaným třídám. To může být velmi užitečné a bude to vysvětleno a použito dále v implementaci.

Asynchronní operace se dají provádět dvěma způsoby:

- pomocí klíčových slov `async` a `await` – tedy v asynchronní metodě (funkci) se prostřednictvím `await` počká na dokončení asynchronní operace,
- nebo pomocí Future API [47] (tedy např. pomocí metod `then` nebo `whenComplete`).

Kapitola 3

Návrh řešení

V této kapitole je popsán hrubý návrh řešení. Nejprve je zdokumentován způsob komunikace s Daktela API. Následuje vysvětlení implementačního návrhového vzoru (design pattern) BLoC, který je v aplikaci využíván. Dále je diskutováno uživatelské rozhraní především mobilní aplikace, popsána obecná architektura aplikace. Konečně je vnesen důvod do nástroje Firebase a lokalizace aplikace (včetně nástroje Crowdin).

3.1 Daktela API

Společnost Daktela vystavuje strandardní REST API [48], se kterým je možné komunikovat prostřednictvím HTTP metod POST, PUT, GET a DELETE. Podporovány jsou datové formáty JSON (defaultní) a JSONP. Velkou výhodou celého produktu je, že toto API je veřejné a využívá ho i robustní webová aplikace. Díky tomu si může zákazník (při dostatečné znalosti produktu a API) sestavit zcela vlastní klientskou aplikaci. API má mnoho různých endpointů, které budou podrobněji rozepsány až v implementační části 4, nicméně tato kapitola shrne základní práci s API a také popíše jeho obecné vlastnosti.

Zákaznické aplikace mají standardně rezervovanou subdoménu (doménu 3. řádu) domény `daktela.com`. Tedy např. `moje.daktela.com`. Všechny veřejné endpointy jsou prefixovány řetězcem `/api/v6` (verze API). Tyto endpointy pak pochopitelně fungují podle REST standardů. Tedy, že úspěšný POST požadavek (request) zapříčiní vytvoření nového unikátního zdrojového identifikátoru (URI). Naopak DELETE požadavek na zmíněný zdroj ho odstraní a ten tedy nebude dále dostupný. POST požadavek se obvykle posílá na modelový endpoint (modely jsou standardně nazývány v množném čísle). GET požadavek lze poslat jak na model, tak i na specifický objekt modelu (v rámci každého modelu má objekt svůj unikatní indentifikátor – obvykle *name*). PUT a DELETE požadavek se posílá na konkrétní objekt daného modelu. CRUD operace popisuje následující příklad na modelu `users` (uživatelé):

- **POST** `/api/v6/users.json` s odpovídajícím JSON payloadem vytvoří nového uživatele,
- **GET** `/api/v6/users.json` vrátí všechny uživatele (maximálně 100 objektů – viz níže),
- **GET** `/api/v6/users/example.json` vrátí uživatele s identifikátorem *example*,
- **PUT** `/api/v6/users/example.json` s odpovídajícím payloadem upraví objekt uživatele s identifikátorem *example*,
- **DELETE** `/api/v6/users/example.json` odstraní uživatele s identifikátorem *example*.

Odpověď (response) HTTP požadavku vrací standardní HTTP status kódy a datová část odpovědi je obvykle složena ze tří částí:

- **error** – chybová hláška (uložena jako pole řetězců, připadně také jako objekt) v případě, že došlo k chybě, jinak `null`,
- **result** – nedošlo-li k chybě, jsou zde uložena vrácená data (nejčastěji jako objekt při dotazu na nějaký konkrétní nebo jako pole objektů např. při obecném dotazu na nějaký model),
- **_time** – časové razítko požadavku.

Důležitou dosud nevyslovenou informací je, že všechny endpointy vyjma endpointu pro přihlášení (`/api/v6/login.json`) vyžadují autentizaci (oveření) uživatele. A to prostřednicvtím tzv. *access tokenu*, který je jedinečný pro každého systémového uživatele v rámci klientské aplikace. Není-li součástí požadavku platný access token, odpoví server status kódem 401 – *Unauthorized*. Tento access token lze získat právě přihlášením – pokud je v payloadu POST requestu na endpoint `/api/v6/ogin.json` správná kombinace přihlašovacího jména a hesla, pak se v odpovědi nachází mimo jiné i access token právě přihlášeného uživatele, který je pak možné dále používat. Získaný access token je možné v zasílat buď prostřednictvím cookies nebo přímo jako query parametr (klíč *accessToken*) požadavku.

V případě práce s větším balíkem dat přijde vhod stránkování, řazení, filtrování (selekce) a projekce. Typicky jsou tyto funkcionality využívány při práci s množinou dat nějakého modelu. Každý požadavek je může obsahovat a posílají se prostřednictvím tzv. query parametrů (součást URL umožňující přidat k požadavku množinu dat).

Stránkovat záznamy je možné prostřednictvím parametrů `take` a `skip`, kde, jak názvy napovídají, první určuje, jak velký balík záznamů se má vrátit,

druhý pak kolik prvních záznamů (s ohledem na řazení) má být přeskočeno – stránkování je tedy v podstatě ekvivaletní MySQL klauzuli LIMIT [49].

Řadit záznamy je možné pomocí jednoho či více polí (sloupců). Slouží k tomu parametry `field` pro specifiaci sloupce a `dir` pro specifikaci směru řazení (vzetupně: `asc`, sestupně: `desc`).

Selekce (neboli filtrování) záznamů je naimplentována komplexně a dokonce se dá používat „vnořeně“. Její základní jednotkou je objekt se třemi hodnotami: `field` (název políčka), `operator` (selekční operátor) a `value` (hledaná hodnota). Selekčích operátorů je celá řada (viz Daktela API), nicméně jsou to např. `eq` (porovnává, zda jsou shodné hodnota políčka a hledaná hodnota), `gt` (zjišťuje, zda je hodnota políčka větší než zadaná hodnota), `contains` (kontroluje, zda hodnota políčka obsahuje jako podřetězec zadanou hodnotu). Takový objekt může být v požadavku buď jeden, případně několik v poli (pak musejí být splněny podmínky ze všech objektů – logický AND). A jak již bylo naznačeno Daktela API umožňuje i komplexní filtrování, kde lze prostřednictvím parametru `logic` (možné hodnoty `and` pro logický AND a `or` pro logický OR) určit, zda musí platit všechny nebo alespoň jeden filtr z pole *filters* (kde jsou tyto filtery obsaženy). Komplexní filtrování, respektive strukturu jeho query parameterů, znázorňuje následující ukázka:

```
{
  "logic": "or",
  "filters": [
    {
      "field": "firstname",
      "operator": "eq",
      "value": "Martin"
    },
    {
      "field": "firstname",
      "operator": "eq",
      "value": "Karel"
    }
  ]
}
```

Ukázky kódu 3.1: Daktela API – ukázka query objektu pro komplexní filtrování

Zmínku zasluhují také dva endpointy, jimiž navrácené datové objekty závisí na tom, jaký uživatel je právě přihlášený. První z nich je endpoint `/api/v6/whoim.json` zahrnující statické informace o aktuálním uživateli. Kromě objektu typu `User` (který zahrnuje např. i přístupy a práva) je možné v odpovědi

nalézt také dostupné fronty či pauzy nebo také verzi systému. Není-li součástí requestu validní access token, odpověď bude obsahovat pouze informaci o verzi systému.

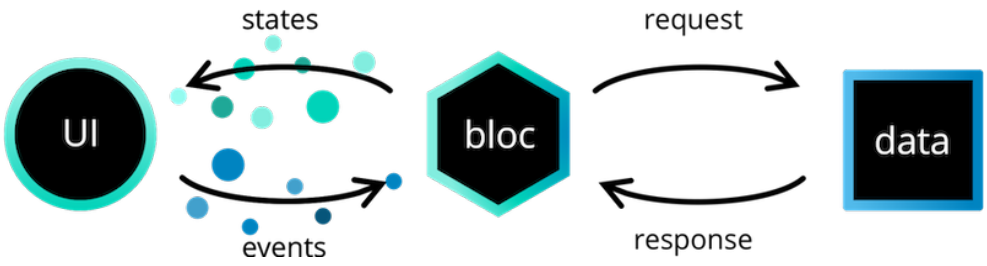
Aby bylo možné zachytávat nejrůznější události, které nastanou (např. příchozí hovor) podporuje Daktela API i long polling [50] endpoint `/api/v6/appPullData.json`, pomocí kterého je možné udržovat persistentní spojení se serverem. Tento endpoint vrací dynamické informace o aktuálně přihlášeném uživateli a přijímá query atribut *hash*, který identifikuje klientský požadavek. Používá se následujícím způsobem:

- klient pošle požadavek na server bez parametru *hash*,
- server ihned odpoví se všemi dostupnými informacemi o přihlášeném uživateli (např. aktivity), přičemž součástí této odpovědi je i atribut *hash*,
- následně klient odešle nový požadavek i s nově příchozím *hash* tokenem,
- server následně čeká na jakoukoli událost vztahující se k přihlášenému uživateli,
- pokud do 15 sekund žádná nenastane, vrátí server odpověď s nově vygenerovaným *hashem* a prázdnými datovými objekty,
- pokud taková událost nastane, server ihned vrátí odpověď s konkrétními daty, které tato událost ovlivnila (nevracejí se tedy všechna dostupná data jako při navazování spojení) a samozřejmě s nově vygenerovaným *hashem*.

Součástí reponse může být mimo jiné i boolean atribut *reloadUser*, značící, že došlo ke změně statických uživatelských dat (např. úprava přístupů či práv) a že by klient měl znovu načíst tato statická data (prostřednictvím endpointu `/api/v6/whoim.json`). Po navrácení odpovědi ze serveru, je nutné pochopitelně znovu odeslat nový požadavek s novým hashem na server, tak aby klient zůstal se serverem v kontaktu a aby mohl nadále zpracovávat nastalé události.

3.2 Návrhový vzor BLoC

Pro oddělení aplikační logiky od prezentační vrstvy (samotných obrazovek) bude využívána knihovna BLoC (**B**usiness **Lo**gic **C**omponent) [51], která je určená pro aplikace napsané v Dartu (včetně Flutter a AngularDart aplikací) a byla již zmíněna v analytické části této práce. Zjednodušeně se dá říct, že BLoC reaguje na události, které obdrží od uživatelského rozhraní. Tyto události zpracuje a následně vrací odpovídající stav, podle kterého je jasné, co má být vykresleno na prezentační vrstvě.



Obrázek 3.1: BLoC architektura [51]

Architekturu naznačuje obrázek 3.1. Datovou část v této práci reprezentují jednotlivé endpointy Daktela API, na které klient zasílá požadavky. Vrstva aplikační logiky, jak název napovídá, obsahuje samotnou logiku a je prostředníkem mezi uživatelským rozhraním a daty. V praxi (při použití Flutter knihovny `flutter_bloc` [52]) se jedná o třídu, jenž dědí z abstraktní třídy `Bloc` s tím, že se této abstraktní třídě pošlou také generické datové typy reprezentující událost (event) a stav (state), náležející tomuto Blocu (např. `MyBloc extends Bloc<MyEvent, MyState>`). Klíčovou metodou Bloc třídy byla `mapEventToState`, která přeměnila příchozí událost (vyvolanou UI) na odpovídající stav, který byl návratovou hodnotou. Přesněji návratovou hodnotou byl stream (zdroj asynchronních událostí [53]) možných stavů. Od tohoto principu se upustilo od verze 8 této knihovny, kde byly streamy nahrazeny klasickými event handlery. Zatímco u streamů se na příchozí události reagovalo příkazem `yield`, který poslal do streamu nový stav, v nové verzi se nový stav odesílá prostřednictvím metody `emit`. Rozdíl mezi oběma způsoby naznačuje následující ukázka kódu (předpokladem je, že jsou definované třídy `Increment` a `Decrement` a dědí ze třídy `CounterEvent`):

```
class CounterBloc extends Bloc<CounterEvent, int> {
  // version 8
  CounterBloc() : super(0) {
    on<Increment>((event, emit) => emit(state + 1));
    on<Decrement>((event, emit) => emit(state - 1));
  }

  // version 7
  Stream<int> mapEventToState(CounterEvent event)
     async* {
    if (event is Increment) yield state + 1;
    else if (event is Decrement) yield state - 1;
  }
}
```

Ukázky kódu 3.2: BLoC knihovna – stream vs. event handler

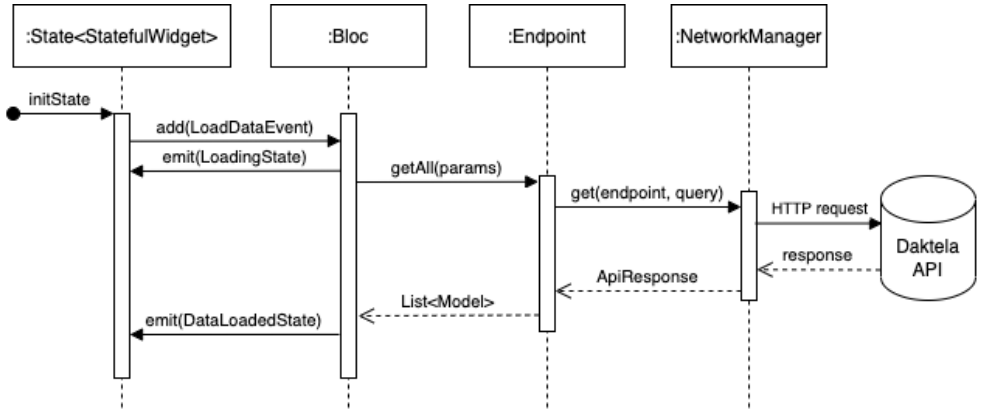
V prezentační vrstvě je nutné instanciovat příslušnou Bloc třídu. Události se pak posílají prostřednictvím metody `add`. BLoC knihovna pak nabízí několik widgetů, které jsou schopny zpracovat stavy odeslané `Bloc` třídou. Základními widgety jsou `BlocBuilder`, `BlocListener` a `BlocConsumer`. Kromě samotné Bloc třídy vyžaduje `BlocBuilder` také *builder* funkci, kterou implementuje programátor. Tato funkce vrací widget v závislosti na aktuálním stavu (jejími vstupními parametry jsou `BuildContext` a stav). `BlocListener` je podobný, má ale namísto *builder* funkce programovatelnou *listener* funkci (se stejnými vstupními parametry jako *builder*). Rozdíl je v tom, že zatímco *builder* vykresluje widgety podle aktuálního stavu, *listener* umožňuje na příchozí stavy reagovat (např. zobrazením dialogového okna, případně otevřením nové nebo ukončením aktuální obrazovky). Kombinací těchto dvou widgetů je pak widget `BlocConsumer`, který má programovatelné funkce *builder* a *listener*.

3.3 Uživatelské rozhraní

Vylepšení uživatelského rozhraní je jedním z nefunkčních požadavků zadavatele. Jak bylo zmíněno, s verzí 6.20 systému Daktela došlo k zásadním úpravám UI ve webové aplikaci. S těmi by měla pochopitelně korespondovat mobilní aplikace, desktop aplikace i webové rozšíření. Současně by mělo dojít i k odstranění neduhů původních aplikací.

Příkladem může být práce s aktivitami a pokročilejšími operacemi (jako je např. správa zařízení). Obecně se aktivity vytvářely prostřednictvím horního app baru (ukazuje to obrázek 2.3 z předchozí verze mobilní aplikace). Z obrázku je patrné, že zatímco v detailu ticketu (vpravo) se na app bar vměstnaly tři tlačítka pro aktivity (komentář, hovor a e-mail), na Dashboard obrazovce (vlevo) bylo na app baru (v případě, že byl uživatel připojený – tzn. v ready stavu) zobrazeno defaultně pouze tlačítko pro vytvoření hovoru. Další tlačítko bylo pro správu stavu uživatele (nastavení pauzy) a zbytek možných akcí byl schovaný pod „třemi tečkami“ – mezi nimi např. obrazovka s aktivitami uživatele, správa zařízení či možnost odpojit se a odhlásit se.

V nové mobilní aplikaci je zmíněný problém řešený jinak. Vzniklo nové plovoucí rozbalovací tlačítko (viz obrázek 4.2), pod kterým jsou schovány všechny dostupné typy aktivit (je tvořeno widgetem `ActivityExpandableFab` a je popsáno v implementační částí této práce – kapitola 4). Toto řešení nejen uvolnilo místo na horním app baru pro další tlačítka relevatní s danou obrazovkou, ale vyřešilo problém s novými typy aktivit – původní aplikace vůbec nepodporovala chatové aktivity, ale i kdyby je podporovala, bylo by obtížné najít pro ně vhodné umístění. Současně byly také přidány dvě nové položky na dolní navigační bar: aktivity a doplňkové menu. Aktivity byly odebrány z rozšířeného menu na horním app baru („tři tečky“) a byla jim přidělena



Obrázek 3.2: Sekvenční diagram komunikace v aplikaci

vlastní fragmentová obrazovka (pozn. fragmentová obrazovka je vysvětlena v implementační části), protože se předpokládá, že se zdokonalováním aplikace budou uživatelé chtít využívat tento modul více a více. Doplňkové menu pak zcela nahrazuje rozšířenou nabídku z horního app baru. Aktuálně sice toto menu neobsahuje mnoho tlačítek (základní informace o aplikaci, novinky v aktuální verzi a odkaz na dokumentaci), ale zde se opět myslí do budoucna – do aplikace je možné přidat prakticky neomezeně modulů, které uživatel vždy přehledně najde na této obrazovce. Krom toho je zde také tlačítko pro odhlášení z aplikace.

Vylepšené uživatelské rozhraní mnohem více používá tab bar, tedy element pro rozdělení obrazovky do menších logických částí (tabů). V původní aplikaci se tento přístup používal jen v modulu CRM pro oddělení submodulů kontakty a společnosti. Nově se tab bar používá třeba právě u aktivit (jejich rozdělení podle aktuální akce: otevřené, čekající, odložené a zmeškané) nebo na společné obrazovce front a zařízení, kde jsou oba tyto submoduly tvořeny vlastním tabem. Dochází také rozdělení jednotlivých částí každé obrazovky do zabalovacích (bílých) boxů. Ty může uživatel libovolně rozbalovat či zabalovat a případně je možné příslušný box zabalit defaultně.

3.4 Architektura

Obecně komunikace v aplikaci probíhá tak, že každá obrazovka má svou BLoC třídu s příslušnými událostmi a stavy. BLoC třída si pak instanciuje endpointy, srkze které načítá data ze serveru (většina operací v BLoC třídách je asynchronní). Obvykle je tedy pro BLoC třídu definován také stav udávající, že je požadavek právě zpracováván. To, že je aplikace v tomto stavu uživatel pozná tak, že se na obrazovce objeví animace symbolizující načítání dat. V momentě, kdy jsou data načtena ze serveru a rozparsována na modely, odešle je BLoC

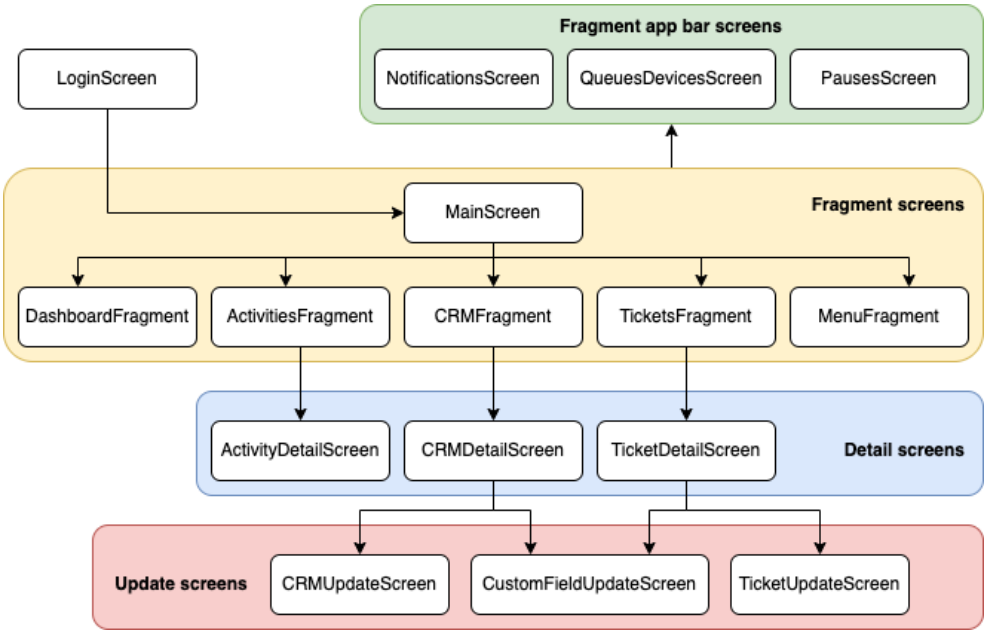
třída (prostřednictvím metody `emit`) na prezentační vrstvu, tedy na konkrétní obrazovku, která se následně překreslí.

Názorněji komunikační *flow* (tok) ukazuje obrázek 3.2. Obrazovka (stateful widget respektive jeho state) v metodě `initState` pošle Bloc třídě událost `LoadDataEvent`. Bloc třída následně odpoví obrazovce stavem `LoadingState`, který je na prezentační vrstvě typicky reprezentovaný načítací animací. Pak už je zavolána metoda `get` třídy `NetworkManager` skrze instanci konkrétního endpointu, která již odešle požadavek na server (Daktela API). Jakmile server odpoví, manažer endpointu pošle instanci třídy `ApiResponse`. V endpointu dojde k dekódování zprávy na model a tento model je konečně navrácen Bloc třídě, která odešle prezentační vrstvě nový stav `DataLoadedState`, jehož součástí jsou i načtená data.

Aplikace je rozdělena do několika základních modulů (balíčků), které budou podrobněji popsány v kapitole 4. Jsou to:

- **main.dart** – je zde `main` funkce celé aplikace a inicializují se zde některé balíčky a knihovny (např. Firebase),
- **blocs** – aplikační (business) logika pro jednotlivé obrazovky (typicky má každá obrazovka svou BLoC třídu a dále pak události a stavy mají také své samostatné třídy),
- **components** – jak název napovídá, jsou zde komponenty (widgety) ale také servisní třídy nebo dialogy, které se používají napříč aplikací nebo tzv. *buňky* (anglicky *cells*), tedy malé náhledové widgety typicky reprezentující nějaký model,
- **extensions** – rozšíření základních Dart nebo Flutter tříd,
- **l10n** – řetězce a překlady (do aplikace je naintegrován překladatelský nástroj Crowdin – viz sekce 3.6),
- **managers** – základní manažery (např. network manager, shared preferences manager nebo Firebase manager) a globální uložiště aplikace (store manager),
- **models** – převážně entitní třídy vrácené Daktela API, případně také pomocné modely,
- **screens** – jednotlivé obrazovky aplikace (pozn. v podložce **desktop** jsou uloženy obrazovky (widgety) určené pro desktop aplikaci).

V aplikaci se pracuje s mnoha obrazovkami, které budou postupně vysvětleny v implementační části. Pro lepší orientaci mezi přechody hlavních obrazovek slouží diagram 3.3. Je v něm naznačeno, že po úspěšném přihlášení dojde



Obrázek 3.3: Hlavní obrazovky v mobilní aplikaci a jejich základní přechody

k přesměrování na obrazovku `MainScreen`. Ta mimo jiné slouží k přepínání mezi tzv. fragmentovými obrazovkami neboli fragmenty (vysvětleny jsou dále v práci). Ze všech fragmentových obrazovek je možné dostat se na *fragment app bar screens*. Některé fragmenty pak mají i své detailní obrazovky a ty mají případně i své editační obrazovky. Diagram skutečně slouží pouze k vizualizaci vztahů mezi základními obrazovkami – nejsou v něm zohledněny výběrové (picker) obrazovky, obrazovky pro vytvoření aktivit (nový email, chat nebo hovor) ani žádné dialogy. Stejně tak se v něm nevyskytují možné přechody mezi detailními obrazovkami.

3.5 Nástroj Firebase

Firebase je označení rozsáhlého produktu od společnosti Google, který umožňuje prostřednictvím různých modulů zdokonalovat a monitorovat aplikace. Mobilní verze aplikace v současnosti využívá dva moduly: *Crashlytics* a *Analytics*.

Prvně jmenovaný nástroj slouží k odchytávání chyb. Chyby jsou monitorovány, přesně detekovány a prioritizovány, což značně usnadňuje proces jejich odstranění [21].

Analytics slouží k monitorování nejrůznějších statistik o používání aplikace – kolik lidí aplikaci používá a na jaké verzi, jak často, v jakém regionu, na jakém zařízení a podobně [22].

Výhledově by také mohla být integrována služba *Cloud Messaging*, která umí zasílat push notifikace. Ty jsou schopny uživatele upozornit na nastalou událost i v případě, že je aplikace vypnutá. V kontextu aplikace Daktela by push notifikace mohla uživatele informovat o příchozí (zvonící) aktivitě (např. hovor či chat).

Jednotlivé moduly mají své knihovny, přičemž tou základní je *firebase_core* [54], která je zodpovědná za propojení Flutter aplikace s Firebase projektem. Dalšími v aplikaci využívanými knihovnami jsou podle očekávání *firebase_crashlytics* [55] a *firebase_analytics* [56]. V aplikaci je pro komunikaci s těmito knihovnami vytvořen manažer `FirebaseManager`.

3.6 Lokalizace a Crowdin

Ve vývoji software obecně dominuje angličtina, nicméně Daktela je mezinárodní společnost, jejíž produkty využívají zákazníci v různých koutech světa. V rámci pohodlného používání aplikace v různých regionech je nutná lokalizace této aplikace. Flutter pro tyto účely nativně poskytuje knihovnu *flutter_localizations* [57].

Pro překlady je využíván formát ARB (založený na JSON). Mimo základní funkcionality klíč-hodnota, kde klíč definuje název příslušného stringu a hodnota pak samotný překlad, je možné využívat i funkcionality pokročilejší. K příslušnému klíči je lze přiřadit vytvořením nového klíče, jehož název bude stejný jako klíč původní ovšem s prefixem **@**. V jeho hodnotě pak může být klasický JSON objekt. Jedním příkladem za všechny jsou tzv. *placeholders* neboli dynamické řetězce z aplikace, které je možné vložit do překladu. [58]

Šablonou (a zároveň defaultním překladem) je soubor `app_en.arb` obsahující anglické řetězce. Z této šablony (zdrojového souboru) se vygeneruje třída (v případě této aplikace je to třída `AppLocalizations`), která jednotlivé překlady vystavuje jako gettery, případě jako metody má-li zdrojový překlad nějaké parametry (např. výše zmíněný placeholder). Ke konfiguraci generátoru slouží soubor `l10n.yaml` v kořenovém adresáři projektu, který definuje, kde jsou překlady umístěny, jaký je zdrojový soubor a do kterého souboru mají být generovány.

Jak již bylo zmíněno zdrojovým jazykem je angličtina, která je spravována přímo ve zdrojovém ARB souboru aplikace. Nicméně spravovat ostatní jazyky přímo v kódu aplikace není nikterak pohodlné, obzvláště ne pro překladatele. Pro tyto účely je využíván překladatelský nástroj Crowdin [20], který poskytuje řadu integrací [59], z nichž je (pro účely této práce) nejzajímavější ta s verzovacím nástrojem GitLab [60], který se ve společnosti Daktela používá a prostřednictvím kterého bude verzována i implementační část této práce. Funguje to tak, že se v nastavení Crowdinu specifikuje zdrojová větev projektu, ze které se mají zdrojové řetězce načíst. Přeložené řetězce se pak propisují zpět

do cílové větve projektu (typicky je jiná než zdrojová – pouze překladová). Pro definici zdrojových souborů a specifikaci souborů pro překlady se pak používá konfigurační soubor `crowdin.yml` umístěný v kořenovém adresáři projektu.

```
// app_en.arb
{
  ...
  "greeting": "Hello {username}",
  "@greeting": {
    "placeholders": {
      "username": {}
    }
  },
  ...
}

// application
...
print(AppLocalizations.of(context)!.greeting('XY'));
// 'Hello XY'
...
```

Ukázky kódu 3.3: Ukázka používání překladů v aplikaci

Kapitola 4

Implementace

Tato kapitola popisuje implementaci celé aplikace prostřednictvím programovacího jazyka Dart a frameworku Flutter, především pak její mobilní části. Postupně jsou tu shrnovány jednotlivé moduly a části aplikace. Na závěr je demonstrováno rozběhnutí multiplatformní aplikace i na desktopové a webové platformě.

4.1 Spuštění a inicializace aplikace

Podobně jako v mnoha jiných programovacích jazycích je i v Dartu funkce `main` tzv. *top-level* funkcí, která slouží jako vstupní bod do aplikace. Právě v ní se také volá základní funkce Flutter aplikace, totiž `runApp` [61]. Ta, jak název napovídá, spustí Flutter aplikaci (jejím argumentem je instance třídy `Widget` [62]). Před samotným spuštěním aplikace dojde ještě k inicializaci některých manažerů, které jsou v aplikaci využívány. Jedním z nich je např. `FirebaseManager` (viz sekce 4.12). Než k této nezbytné inicializaci dojde (stejně tak, než nativní aplikace stihne načíst Flutter), zobrazí se uživateli defaultně bílá obrazovka (tzv. splash screen). Knihovny *flutter_native_splash* [63] umožňuje pohodlně nahradit tento defaultní obrázek nějakým více užitečným – např. logem společnosti.

Samotnou aplikaci definuje widget MaterialApp [64], který má mnoho možných parametrů. Několik parametrů se např. týká routování (směrování) mezi obrazovkami. V `routes` je uložena hlavní routovací tabulka aplikace – vždy jako dvojice název routy a funkce, která vrací widget (obrazovku). Právě zde je možné namapovat jednotlivé obrazovky na příslušné routy. Při zavolání metody `pushNamed` třídy `Navigator` [65] jsou pak routy hledány právě v této tabulce. Úzce související je pak parametr `initialRoute`, přes který je možné nastavit iniciální routu (v tomto případě routu na přihlašovací obrazovku).

4.2 NetworkManager

Je to pravděpodobně nejdůležitejší manažer v aplikaci, protože zajišťuje veškerý přenos dat mezi klientem a serverem. Jeho metody jsou vzhledem k jeho účelu výhradně statické. Poznámka k volání asynchronních operací – v aplikaci jsou v naprosté většině případů volány stylem `async/await`.

Nejzákladnější jsou asynchronní metody `get`, `post`, `put` a `delete`. Z logiky věci jsou tyto metody velmi podobné. Jediným povinným parametrem je URI specifikující endpoint, další parametry jako například mapy pro payload a query parametry jsou volitelné. O samotné odeslání HTTP požadavků se stará populární knihovna `http` [66] od autorů jazyku Dart. Návratovým typem těchto metod je instance třídy `ApiResponse`, do které se uloží data dekódovaná z JSON formátu (dynamicá proměnná `result` a případně číselná hodnota `total`). Přímo v NetworkManageru jsou odchytávány také 4 základní druhy chyb:

- **bad request** – nejčastější chyba, která nastane při zaslání nevalidního požadavku (např. chybějící povinná pole v payloadu), součástí odpovědi většinou bývá parametr `error`, který popisuje chybu (k rozparsování chybové hlášky slouží metoda `_parseApiError`, která vyhodí výjimku `ApiException` s právě rozparsovaným popiskem),
- **unauthorized** – je-li návratový kód roven 401, je vyhozena výjimka (třída `UnauthorizedException`) – téměř výhradně při zadání neplatných přihlašovacích údajů,
- **not found** – vrátí-li Daktela API odpověď s kódem 404, je vyhozena výjimka `NotFoundException` – např. při snaze načíst neexistující URI,
- nastane-li timeout (odchycena výjimka `TimeoutException`), dojde k vyhození výjimky `ApiException` se zprávou o vypršení časového limitu požadavku.

V manažeru je také metoda pro nahrání souboru na server (třída `File` z knihovny `dart.io` [67]).

4.2.1 Workers

Jako *workeři* jsou v aplikaci označeny třídy, které zajišťují průběžné stahování statických resp. dynamických dat přihlášeného uživatele (endpointy `/api/v6/whoim.json` resp. `/api/v6/appPullData.json`). Základem workera je abstraktní třída `BaseWorker` s metodami `addObserver`, `removeObserver` a `notifyObserver`. Tyto metody slouží pro přidání, odebraní a notifikování observerů spojených s tímto workerem. Oba výše zmíněné endpointy mají svůj vlastní observer s vlastní abstraktní notifikační metodou. Widget (obrazovka),

který chce být notifikován při načtení nových (aktualizaci) dat musí implementovat právě zmíněné notifikační metody. To se v aplikaci týká především stavových (stateful) widgetů. Aby byly tyto widgety notifikovány, musejí se také registrovat mezi observery daného workera. To se typicky dělá v metodě `initState` statové třídy [68] (odregistrace pak v metodě `dispose`).

Konkrétní worker pak implementuje abstraktní metody `start`, `stop` a `loadAndStore` třídy `BaseWorker`. Jejich implementace se pochopitelně liší podle typu workera (jiné endpointy, timeouty mezi voláními a dekódování dat).

4.2.2 Pagination, Sort a Filter

Nedílnou součástí komunikace s Daktela API jsou pochopitelně i věci jako stránkování, řazení či filtrování záznamů, které byly již zmíněny v sekci 3.1.

Stránkování zajišťuje jednoduchá třída `Pagination`, která má definované tři metody `next`, `back` a `reset`, které posouvají čítač dopředu, dozadu nebo ho nulují. Konstruktorem je možné nastavit velikost stránky (případně se použije velikost výchozí).

Řadit výsledky je možné vzestupně i sestupně podle různých hodnot. K tomu slouží pár (třída) `SortField`, který nese název políčka, podle kterého budou výslekdy řazeny, a dále také směr řazení (reprezentovaný enumerátorem `SortDirection`). Celé řazení je pak tvořeno třídou `Sort` se seznamem instancí třídy `SortField`, nicméně vzhledem k tomu, že nejčastější jsou záznamy řazeny podle jedné hodnoty, je implementován také jmenný (factory) konstruktor `simple`, který přijímá jediný `SortField`.

V aplikaci je naimplementován komplexní filtr, který se skládá z enumerátoru `FilterLogic` a seznamů políček (`FilterField`) a připadných dalších vnořených filterů (`Filter`). Podobně jako u řazení je zde možné vytvořit filtr pomocí jmenného konstruktoru `simple`, který obsahuje jediné filtrovací políčko. Třída `FilterField` obsahuje název políčka, porovnávací operátor (enumerátor `FilterOperator`) a volitelně také hodnotu k porovnání (vzorovou hodnotu).

Výše zmíněné argumenty HTTP dotazu se posílají prostřednictvím query parametrů URI adresy (v Dartu mapa typu `Map<String, dynamic>`). Je tedy třeba objekty stránkování, řazení a filtrování překovertovat na tuto mapu. Toho je dosaženo prostřednictvím rozšíření `QueryMap` třídy `Map`, respektive jeho metody `buildQueryMap` s parametry `Filter`, `Sort` a `Pagination`. Tato metoda navrátí mapu, které bude rozumět Daktela API. Typicky je zmíněná metoda volána v třídách jednotlivých endpointů, kde se volá také samotný request (viz níže). Následující metoda ukazuje překonvertování objektu třídy `Sort` do mapy:

```
void _enrichWithSort(Sort sort) {
  sort.fields.asMap().forEach((i, val) {
    this['sort[$i][field]'] = val.field;
    this['sort[$i][dir]'] = val.direction.value;
  });
}
```

Ukázky kódu 4.1: Dart – uložení Sort objektu do mapy

4.2.3 Endpoints

Poslední dosud nezmíněnou částí Network Manageru je složka `endpoints`. Jak název napovídá, nacházejí se zde jednotlivé endpointy Daktela API využívané v aplikaci. Každý endpoint má svou třídu, ve které jsou definovány metody, jejichž prostřednictvím aplikace komunikuje se serverem. Tyto metody obvykle pokrývají klasické HTTP metody (např. `get`, `getAll` nebo `post` v různých endpointech). V některých endpointech mohou být i specifičtější metody (např. `login` ve třídě `LoginEndpoint`).

Každá metoda endpointu pak volá jednu ze základních metod manažeru `NetworkManager` (`get`, `post`, `put`, `delete`), popsaných výše. Nejčastěji s endpointy komunikují Bloc třídy a to tak, že si endpoint instaciují a následně volají jeho metody s příslušnými parametry (např. filtry, stránkování, řazení nebo identifikátory). V metodě se pak sestaví mapa query parametrů (metoda `buildQueryParams` rozšíření `QueryMap` zmíněná výše) a odešle se požadavek. Odpověď ze serveru se následně dekóduje na příslušný model, kterému již rozumí aplikace.

4.3 BLoCs

Každá obrazovka v aplikaci má svou BLoC třídu se suffixem `Bloc`), ke které vážou i její třídy pro události (suffix `Event`) a stavy (suffix `State`). Konkrétní události a stavy pak dědí ze základních (právě zmíněných) tříd.

Obrazovka využívající Bloc dědí z jednoduché třídy `BlocScreen`, což je abstraktní třída se čtyřmi generickými datovými typy:

- `W` – třída stavového widgetu (tedy třída dědící ze třídy `StatefulWidget`),
- `B` – Bloc třída (dědí ze třídy `Bloc<E, S>` knihovny `flutter_bloc` [52]),
- `E` a `S` – abstraktní event a state třídy výše zmíněné Bloc třídy `B`.

`BlocScreen` pak dědí ze třídy `State<W>` a je abtraktní, přičemž jejími abstraktními metodami jsou `listener` a `builder`. V konstruktoru přijímá instanci Bloc třídy, které se typicky vytváří v metodě `createState` statového widgetu (`StatefulWidget`). Třída přetěžuje (override) metodu `build` a ta

vrací widget `BlocConsumer` (z knihovny knihovny `flutter_bloc` [52]), který umí reagovat na změnu stavu dvěma způsoby. Prostřednictvím listeneru, který se používá pro vyvolání jednorázových akcí (jako je zobrazení dialogu nebo při práci s navigátorem – např. otevření nové obrazovky). A dále prostřednictvím builderu, který v závislosti na aktuálním stavu vykresluje widgety. Tedy např. při načítání dat ze serveru se zobrazí načítací bar a jakmile se vrátí odpověď, zobrazí se její obsah. `BlocConsumer` slučuje widgety `BlocBuilder` a `BlocListener` do jediného. Je možné také detailněji specifikovat, kdy má listener naschlouchat a builder překreslovat – prostřednictvím parametrů `listenWhen` a `buildWhen`, což jsou funkce, jejichž argumenty jsou předchozí a aktuální stav a vracejí boolean hodnotu (defaultně obě vracejí `true` nezávisle na stavech v argumentech).

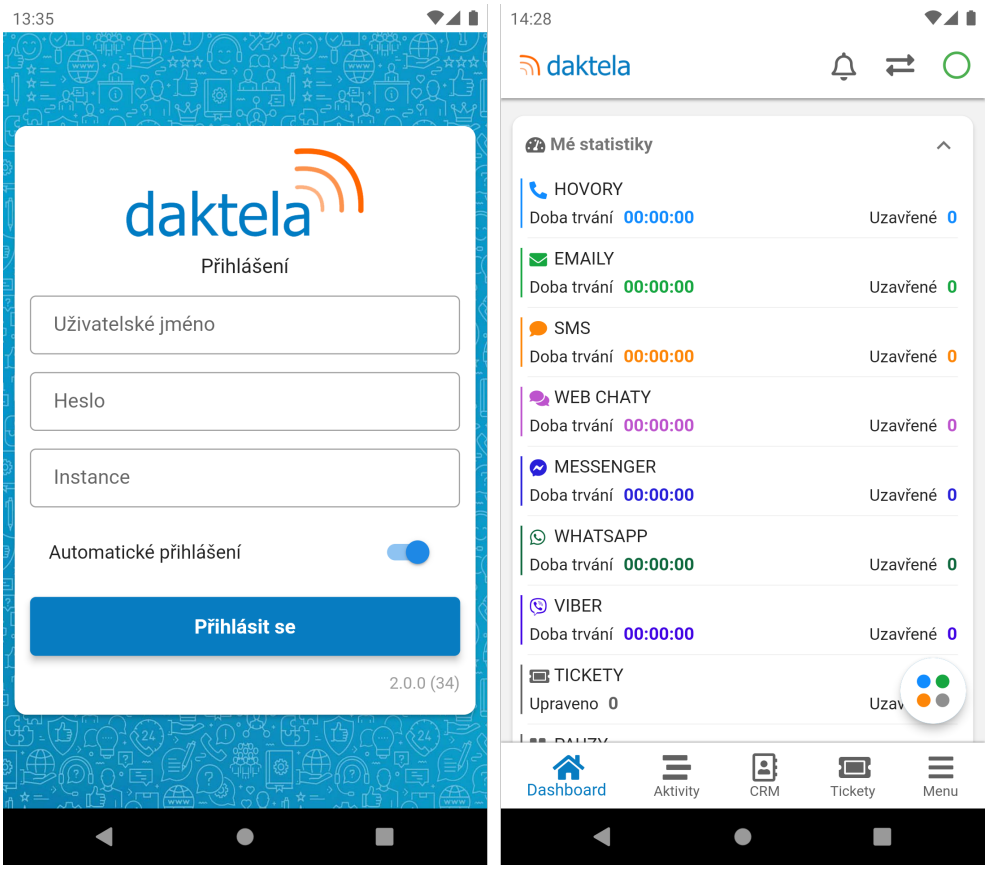
Komunikace mezi prezentační vrstvou, Bloc třídou a serverem sestává z několika částí. Obrazovka musí být spárována s Bloc třídou, což se dělá tak, že příslušná obrazovka dědí z výše zmíněné třídy `BlocScreen`) a implementuje její metody `listener` a `builder`, aby mohla reagovat na změny stavu. Následně je z prezentační vrstvy možné volat (odeslat ke zpracování) události (skrze metodu `add` Bloc třídy). Pokrývá-li Bloc třída i asynchronní komunikaci se serverem, pak obvykle instanciuje také příslušné endpointy (ve kterých se volají HTTP metody Network Managera a data se parsují na modely). Po zpracování požadavku (např. načtení dat) odešle Bloc třída nový stav, na který zareaguje `BlocConsumer`, respektive jeho `builder` a `listener`.

Odchytávání výjimek při komunikaci se serverem zajišťuje `errorHandler`, což je jednoduchá funkce, která obaluje (wrap) operace vykonávané (volané) v Bloc třídě. Dojde-li k výjimce při komunikaci s API (`ApiException`), vrátí se rozparsovaná chybová hláška. Při jakékoliv jiné výjimce se vypíše hláška „Něco se pokazilo“. Tyto chybové hlášky se vracejí prostřednictvím Bloc stavu a funkce emit. Každá Bloc třída má ve svých stavech obykle i jeden chybový stav, který implementuje mixin `BaseError` (obsahující proměnnou pro uložení chybové hlášky).

4.4 Přihlášení

Jak již bylo zmíněno, systém Daktela běží vetšině zákazníků na subdoméně domény `daktela.com`. Při přihlášení do aplikace je tedy nutné zadat (kromě přihlašovavích údajů) také tzv. *instanci* zákazníka, tedy jeho subdoménu.

Na obrazovce je kromě těchto textových polí ještě možnost automatického přihlášení. Kdykoliv uživatel spustí aplikaci, zobrazí se mu (po první inicializaci) právě přihlašovací obrazovka. Vyplňovat přihlašovací údaje při každém spuštění aplikace je ovšem značně nepohodlné. Proto se údaje ukládají do persistentního úložiště dané platformy (např. *SharedPreferences* pro Android). K tomu slouží knihovna `shared_preferences` [69]. Prostředníkem mezi apli-



Obrázek 4.1: Přihlášení a dashboard v mobilní aplikaci

kací a touto knihovnou je pak `SharedPreferencesManager`, který definuje vlastní metody pro zápis, čtení a mazání dat, ve kterých pak využívá nativní metody zmíněné knihovny. Jednotlivé klíče jsou definovány skrze emulátor `SharedPreferencesKey`. Manažer tak slouží jako most (bridge) mezi knihovnou aplikací. Tento princip je v aplikaci aplikován relativně často (např. `FirebaseManager`, `EncryptManager`, `DeviceInfoManager` atd.) ve snaze minimalizovat závislost aplikace na nějaké knihovně – dojde-li k razantnější úpravě knihovny, stačí tuto úpravu aplikovat na jednom místě – v jejím manažerovi.

Do persistentního úložiště se tak ukládají uživatelské jméno, heslo, instance a již zmiňovaná volba automatického přihlášení. Je-li zaškrtnuta, pak se aplikace pokusí přihlásit uživatele podle údajů nalezených v persistentním úložišti. Údaje se pochopitelně ukládají zašifrované – šifrování a dešifrování zajišťuje `EncryptManager` s pomocí knihovny *encrypt* [70] – a ukládají se až po úspěšném přihlášení. Při odhlášení uživatele vždy dojde k vymazání hesla z úložiště a také ke zrušení případného automatického přihlášení. Samotný

`LoginEvent` zpracovává `LoginBloc`. Nejdříve se zkontroluje licence zadané instance (zda má uživatel přístup např. k mobilní nebo desktop aplikaci). Následně se prostřednictvím endpointu `/api/v6/login.json` ověří přihlašovací údaje a jsou-li validní (neboli vrátí se validní response, jejíž součástí je i access token uživatele), načtou se také statická a dynamická data právě přihlášeného uživatele (skrze `/api/v6/whoim.json` a `/api/v6/appPullData.json`). Jsou-li všechna zmíněná API volání úspěšná, vyšle `LoginBloc` stav `LoginSuccess`, na který aplikace zareguje otevřením obrazovky `MainScreen`.

4.5 Základní obrazovky a komponenty

V aplikaci je několik základních obrazovek ve smyslu komponent, které jsou dále využívány v aplikaci pro vytvoření obrazovek konkrétních. Výjimkou je obrazovka `MainScreen`, která je poněkud specifická (viz níže).

4.5.1 MainScreen

Tato obrazovka je v jistém smyslu pomocná. Jak již bylo napsáno, po úspešném přihlášení se aplikace přesměruje právě sem. V `initState` resp. `dispose` metodách stavu stateful widgetu (třída dědící ze `State<MainScreen>`) dochází ke spuštění resp. zastavení aplikačních workerů (`AppPullDataWorker` a `WhoImWorker`).

V metodě `build` se definuje navigátor pro aplikaci a také dolní navigační bar (bottom navigation bar), který se nachází v dolní části obrazovky (viz obrázek 4.1). Widget `CustomBottomNavigationBar` tvoří navigační bar (dále označovaný navigátor). Jeho součástí je i callback `_menuItemChanged`, který se zavolá po kliknutí na položku v navigátoru. Jednotlivé položky navigátoru definuje enumerátor `NavigationItem` a každá z nich je svázána s tzv. fragment obrazovkou, kterou zobrazuje/vykresluje (to zajišťuje právě zmíněný callback). Základními položkami (fragmenty) jsou:

- **Dashboard** zobrazující denní přehled uživatele, jeho poslední aktivity a případná oznámení.
- **Aktivity** (otevřené, čekající, odložené a zmeškané), se kterými může uživatel interagovat.
- **CRM** modul dělící se dva podmoduly: **kontakty** a **společnosti**, které uživatel může spravovat.
- Přehled **ticketů** a všech akcí/interakcí k nim se vázajících (jejich úprava či iniciování aktivit).
- **Menu** pro další moduly a informace (o aplikaci, novinky ve verzi, dokumentace).

`CustomBottomNavigationBar` implementuje notify metody obou workerů. `WhoImWorker` kvůli oprávněním uživatele pro zobrazování jednotlivých modulů aplikace – v případě, že dojde k odebrání přístupu k modulu Tickety (přes webovou aplikaci), pak z navigátoru tato položka zmizí. Vyčíst a aktualizovat počty aktivit (otevřených, čekajících, odložených nebo zmeškaných) je pak možné z `AppPullDataWorker`, respektive jeho modelu. Jejich součet se zobrazuje jako tzv. badge ikona u navigační položky Aktivity, tedy jako malé číslo částečně překrývající tuto položku.

Pro samotné vykreslení bottom navigation baru v metodě `build` se využívá nativní Flutter widget `BottonNavigationBar` [71].

4.5.2 Obrazovkové komponenty

Základem většiny obrazovek je bezstatový widget `AbstractScreen`. Z argumentů vyčte všechny potřebné informace pro vykreslení obrazovky (využívá se zde Flutter widget `Scaffold` [72]). Nejsou to jen samotné widgety k vykreslení, ale také plovoucí tlačítka (floating buttons), skrolovací controllery, `InfiniteLoader`, refresh callback atd. `InfiteLoader` je jednoduchá služba, která umožňuje postupně načítat nekonečné seznamy (stránkování). Dojde-li k překonání určité hranice při skrolování (threshold), zavolá se callback `nextPageCallback`, ve kterém je z konkrétní obrazovky poslána funkce pro načtení další stránky (neboli zavolání Bloc události). Refresh callback je pak funkce, která se zavolá při obnovení obrazovky uživatelem (swipe nahoru).

Widget `AbstractScreen` využívájí 3 základní obrazovky, které jsou pak využívány dále v aplikaci:

- `BaseFragmentScreen` – obrazovka, která má jednotný app bar (horní bar) a na kterou se dá typicky dostat skrze dolní navigátor (viz Dashboard na obrázku 4.1),
- `BaseScreen` – nejobecnější obrazovka umožňující nadefinovat si app bar (tlačítko vlevo, název, akce vpravo), nejpoužívanější v aplikaci,
- `BaseScreenTabBar` – obrazovka s tab barem (přepínečem) nacházejícím se hned pod app barem, kterou je možné využít jako fragment obrazovku (`BaseFragmentScreen`) i jako obecnou obrazovku (`BaseScreen`).

Na app baru obrazovky `BaseFragmentScreen` je mimo jiné vidět aktuální stav přihlášeného uživatele (zda je nečinný, na pauze nebo odpojený). Tato informace je čitelná z app pull dat. Výše zmíněná obrazovka `MainScreen` implementuje notifikační metodu workera `AppPullDataWorker`, ve které volá událost `Refresh` blocu `MainBloc`. `BlocBuilder` je tedy základem fragmentového app baru a zajišťuje překreslení obrazovky při změně stavu. V případě, že je uživatel odpojený (stav unready) na horním app baru (viz obrázek 4.1 vpravo) bude viditelné pouze jediné červené tlačítko, pomocí kterého se uživatel může připojit.

4.6 Dashboard

Dashboard je fragmentová obrazovka zobrazující statistiky přihlášeného uživatele a je to první obrazovka, která se zobrazí po přihlášení. Data pro tuto obrazovku jsou načítána z endpointu `/api/v6/portlets.json`. Obrazovka se skládá ze tří částí: *mé statistiky*, *oznámení* a *poslední aktivity*. Načítání dat zajišťuje Bloc třída `DashboardFragmentBloc`.

Statistiky znázorňují aktivitu uživatele v aktuálním dni, tedy počty aktivit (hovorů, emailů) a vyřešených ticketů a čas strávený na pauze. Widget `DashboardStatCell` tvoří buňku (cell) jedné statistiky.

Oznamení je jednoduchá zpráva, která může sloužit např. jako připomínka. Vykresluje ho `AnnouncementCell` na základě modelu `Announcement`.

Pro zobrazení posledních 10 aktivit slouží widget `ActivityCell`, který zobrazuje základní informace o aktivitě v závislosti na jejím typu (hovor, email, atd.). Umí si poradit také s modelem zmeškaných aktivit a je v aplikaci hojně používán. Zobrazí detailní obrazovky (detailu aktivity) zajišťuje právě `ActivityCell` (je-li povoleno).

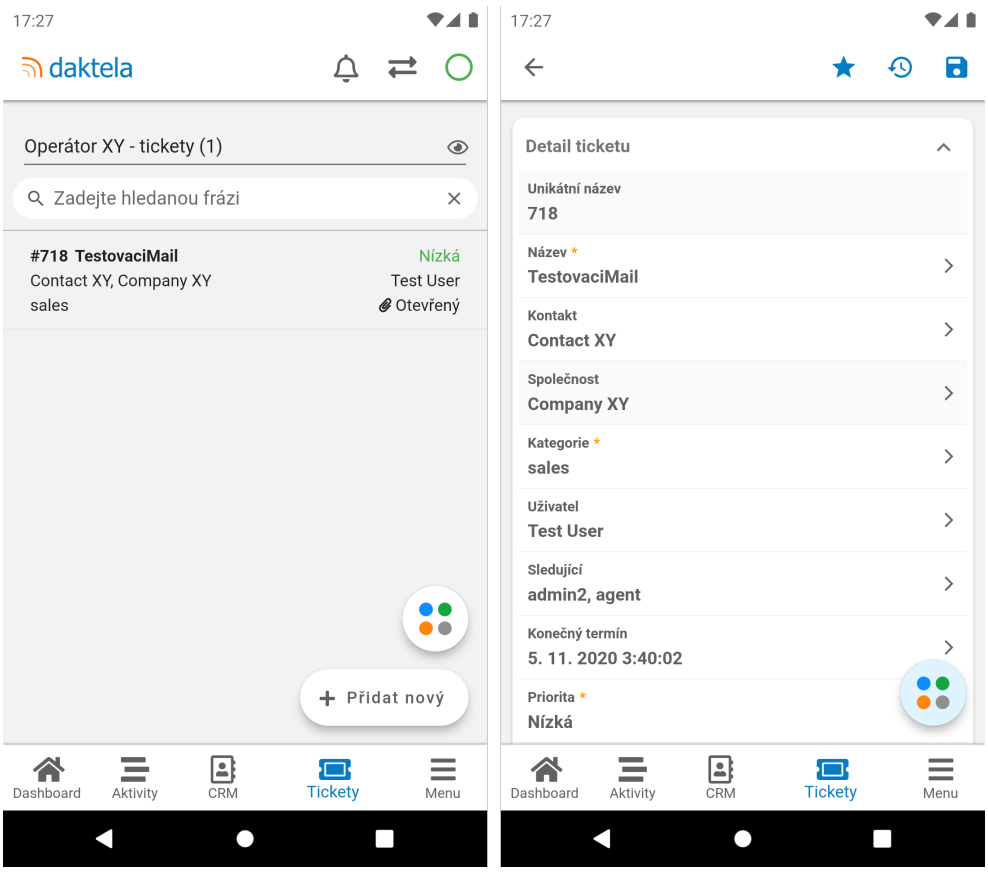
4.7 Tickety

Tickety jsou pravděpodobně nejrozsáhlejším modulem v celé aplikaci. Ten se skládá ze dvou základních částí: přehledová obrazovka s pohledy a full textovým vyhledávačem a dále pak detailní obrazovka konkrétního ticketu. Současně budou níže i pomocné obrazovky využívané např. při editaci ticketu.

4.7.1 Přehledová obrazovka (`TicketsFragment`)

Jedná se o tzv. fragmentovou obrazovku, tedy o jednu z obrazovek s jednotným horním app barem. V horní části se nachází dropdown s pohledy přihlášeného uživatele. Jejich správa je možná skrze nastavení (které mobilní aplikace nepokrývá). Jak již bylo zmíněno, pohled reprezentuje nějaký uložený filtr, podle kterého se uživateli zobrazí cílový seznam ticketů. Pro vykreslení této rozbalovací nabídky se využívá Flutter komponenta `DropdownButton` [73], což je generická třída, jejíž typem je třída prvků této nabídky – v tomto případě model `TicketView`, jehož vlastnostmi jsou mimo i jiné i instance již popsaných tříd `Filter` a `Sort`.

Pod pohledovým dropdownem se nachází `SearchTextField`, což je stateless widget využíváný typicky pro full textové vyhledávání na různých obrazovkách aplikace (skrze konstruktor widgetu se posílá callback pro spuštění vyhledávání – nejčastěji funkce volající událost přílušené Bloc třídy, v tomto případě `TicketsFragmentFetchTickets`). Jeho základem je pochopitelně nativní Flutter widget `TextField` [74]. Důležitou infomací je, že při vyhledávání ticketu pomocí full textového vyhledávače se zohledňuje také právě zvolený



Obrázek 4.2: Náhledová a detailní obrazovka modulu Tickety

pohled. Chce-li tedy uživatel prohledat např. všechny tickety full textovým vyhledáváním, musí mít zvolený pohled, který je všechny vrátí.

Zbývá už jen samotný seznam nalezených ticketů. Náhled jednoho ticketu reprezentuje `TicketCell` zobrazující základní informace o ticketu (číslo, název, prioritu, kontakt, kategorii a stav). Každý tento cell je klikatelný a otevře detailní obrazovku ticketu (za předpokladu, že má přihlášený uživatel dostatečná oprávnění).

Před načtením samotných ticketů se musejí nejdříve načíst pohledy. Vzhledem k tomu, že k úpravě pohledů dochází zřídkakdy, kešují se (respektive jsou uloženy ve manažeru `StoreManager`). Časový inverval pro znovunačtení pohledů ze serveru je 5 minut.

4.7.2 Detailní obrazovka (`TicketDetailScreen`)

Základem detailní obrazovky konkrétního ticketu je `BaseScreen` – nejedná se tak o fragment obrazovku a je možné prostřednictvím parametrů nadefinovat

horní app bar. Ten je složen z volitelného tzv. *leading* widgetu (vlevo) a dále seznamu tzv. *actions* widgetů (vpravo). V tomto případě je leading widgetem komponenta `DiscardChangesButton`, která má jednoduchý účel – totiž ověřit (skrze dialog), zda chce uživatel zahodit rozpracované změny. Dialog se zobrazí pouze v případě, že k nějakým změnám došlo. Vpravo jsou pak základní akce na přídání ticketu do oblíbených, zobrazení jeho historie a uložení změn.

Historie ticketu, neboli snapshoty, ukládá stav ticketu při jeho uložení, aby bylo možné sledovat jeho historii a podívat se na úpravy zpětně. V aplikaci jsou v současnosti 3 typy snapshotů: *tickets*, *contacts* a *accounts*, které jsou reprezentovány enumerátorem `SnapshotType`. Pro výběr snapshotu slouží obrazovka `SnapshotPickerScreen`, která na základě typu snapshotu a identifikátoru objektu načte příslušné snapshoty. Jakmile uživatel nějaký vybere, picker (vybírací) obrazovka se uzavře a vybraný snapshot se přepošle zpět na detail ticketu, kde se následně překreslí hodnoty políček ticketu na hodnoty ve zvoleném snapshotu. K propsání nějaké informace na předešlou obrazovku slouží pomocná třída `UpdateResult`, která je detailěji rozepsána u ticketového kometáře v podsekci 4.7.3.

V zabalovacím widgetu `ExpandableBox` jsou pak všechna pole ticketu včetně custom fieldů. Tato pole ticketu jsou reprezentována enumerátorem `TicketField`, ve kterém (respektive v jeho rozšíření) jsou definovány název pole, jeho API klíč, typ (enumerátor `FieldType`) a další vlastnosti konkrétního políčka. Jsou tu dvě specifické metody `titleOf` a `nameOf`, obě s dynamickým parametrem `element` – instance nějakého modelu, případně pole jeho instancí.

Metoda `titleOf` převede hodnotu políčka (nějaký objekt) na řetězec, neboli na hodnotu, kterou reálně uvidí uživatel v aplikaci. Je-li cílovým objektem např. kontakt (model `Contact`), metoda vrátí jeho jméno a příjmení. Může-li políčko nabývat více hodnot, jsou jednotlivé hodnoty odděleny čárkou. Tato metoda je volána ve právě popisované obrazovce (detail ticketu) na každé políčko ticketu.

Druhá metoda `nameOf` se pak volá při vytváření/ukládání ticketu, tedy při posílání POST/PUT požadavku a z jednotlivých hodnot (objektů) vyextrahuje identifikátory (ve většině modelů řetězec `name`), kterým rozumí Daktela API.

Samotné pořadí políček ticketu a jeho hodnoty jsou definovány v modelu `Ticket`. Slouží k tomu gettery `fields` (instační) `newTicketFields` (třídní), které vracejí pole instancí `TicketFieldValue`, což je pomocná třída pro dvojici `TicketField` a příslušnou hodnotu (hodnota má dynamický datový typ). Widget jednoho políčka reprezentuje komponenta `TitleValueActionCell`, která mimo jiné přijímá i seznam akcí. Tyto akce jsou definovány třídou `TitleAction` a jejich zobrazení zajišťuje služba `TitleActionService`, která je popsána dále v práci (v sekci 4.12.3). Jedním z argumentů komponenty `TitleValueActionCell` je také přepínač `executeSingleCommandDirectly`,

který umožňuje, v případě, že je v poli akcí přesně jedna akce, zavolat tuto akci rovnou – bez zobrazení `TitleActionService` dialogu.

U základních políček ticketu (enumerátor `TicketField`) převládá akce editovat. Nachází se téměř u všech políček vyjma např. společnosti, časů vytvoření, poslední editace atd. O editaci hodnoty ticketového políčka se stará níže popsaná obrazovka `TicketUpdateScreen`. U CRM políček (kontakt a společnost) je pak také možné dostat se na detailní CRM obrazovku `CRMDetailScreen`.

Custom fieldy mají své vlastní schéma, které se váže na kategorii ticketu. I z toho důvodu je kategorie u ticketu vždy povinná. Na základě kategorie ticketu je tedy nezbytné toto schéma načíst. Hodnoty custom fieldů jsou v konkrétním objektu (např. v ticketu) reprezentovány jako páry klíč-hodnota (identifikátor názvu custom fieldu a pole řetězců). Další detaily jako např. název políčka, jeho typ atd. je nutné vyčíst právě ze schématu. Dojde-li ke změně kategorie ticketu, je nutné pochopitelně načíst a vykreslit schéma nové (jak jinak než přes příslušnou Bloc třídu – tentokrát `TicketDetailBloc`).

Podobně jako u základních polí ticketu, je i u custom fieldů nejčastější akcí jeho editace – k tomu slouží obrazovka `CustomFieldEditScreen`. Ostatní akce definuje služba `CustomFieldService` v závislosti na custom fieldu a jeho hodnotách. Podle typu custom fieldu jsou dostupné následující akce:

- pro typ *EMAIL* je možné napsat e-mail – obrazovka `NewEmailScreen` (více v sekci 4.9),
- je-li custom field typu *PHONE*, je možné číslo vytočit, případně na něj odeslat SMS zprávu (více opět v sekci 4.9 – Aktivity),
- konečně pro custom fieldy typu *URL* lze otevřít URL adresu v prohlížeči daného zařízení (pomocí služby `UrlLauncherService` prostřednictvím knihovny *url_launcher* [75]).

Pod zabalovacím widgetem se seznamem polí ticketu se nachází seznam jeho aktivit. Ty jsou reprezentovány v aplikaci hojně využívaným widgetem `ActivityCell` a každý tento cell je pochopitelně klikatelný a směřuje na detail dané aktivity (`ActivityDetailScreen`).

4.7.3 Přidat komentář k ticketu (`TicketCommentScreen`)

Přidat komentář k aktivitě je možné přes `ActivityExpandableFab` v detailu ticketu (viz sekce 4.9). Vzhledem k tomu, že Daktela jako produkt je především webová aplikace, komentáře (ale také e-maily) podporují HTML formátování. Podporuje ho tedy pochopitelně i tato aplikace a to s pomocí knihovny *html_editor_enhanced* [76], která mimo jiné umožňuje specifikovat,

která tlačítka se na toolbaru editoru zobrazí (např. tlačítka na nastavení fontu, barvy či seznamů).

Ke komentáři je možné rovněž přidat přílohu (soubor, fotku nebo video). O to se stará služba (dialogové okno) `AttachmentService`. Dojde-li k úspěšnému uložení komentáře, obrazovka `TicketCommentScreen` se uzavře (klasicky pomocí metody `pop` třídy `Navigator` [65]). Do této metody je možné vložit jako volitelný parametr také výsledek (result), což může být instance libovolné třídy (dědící ze třídy `Object`, což je základní třída v jazyku Dart). Pro tyto účely je definována pomocná třída `UpdateResult` se dvěma parametry: `result` s dynamickým datovým typem a pak přepínač `reload`, který používá právě v případě úspěšného uložení komentáře pro znovunačtení aktivit ticketu.

4.7.4 Editace základních polí ticketu (`TicketUpdateScreen`)

Klíčovou dvojící pro editaci políčka ticketu je `TicketField` a pak samotná hodnota políčka (dynamický datový typ). Na editační obrazovku se posílá také aktuální kategorie ticketu pro určení povinnosti a multiplicity statusů ticketu. K tomu slouží metody `isOptional` a `isMultiple` již zmiňovaného enumerátoru `TicketField`.

Typy polí ticketu jsou různé, proto má každý `TicketField` nadefinovaný svůj pomocí enumerátoru `FieldType`. Jeho možnými hodnotami (typy) jsou:

- `text` – obyčejný jednořádkový text (název ticketu),
- `textarea` – víceřádkový text (popis ticketu),
- `list` – možnost výběru z předem definovaných hodnot (uživatel, sledující, kontakt, kategorie, priorita, stav nebo sledující),
- `datetime`, `date` a `time` – různé typy časových hodnot (konečný termín nebo znovuotevření ticketu, pro účely základních polí tickety stačí typ `datetime`).

Právě typ pole určuje, co (jaký widget) se uživateli zobrazí na editační obrazovce. Je-li typ *list*, načtení položek zajišťuje Bloc `TicketUpdateBloc`, ať už je jedná o dynamické položky, které je třeba načíst ze serveru (např. uživatelé), nebo o položky statické jako priorita nebo stav ticketu. U tohoto jediného `FieldType` se také nad samotnými položkami zobrazí vyhledávač (`SearchTextField`). `DateTimePickerService` se používá pro výběr hodnoty u časových typů. U *text* a *textarea* typů pak `CustomTextField`.

Jakmile chce uživatel uložit změny provedené na editační obrazovce, provede se ještě validace dat, tedy kontrola, zda jsou vyplněny povinné položky (opět skrze Bloc událost). Pokud ano, přes zmíněnou třídu `UpdateResult` se data propíší zpět na obrazovku `TicketDetailScreen`.

4.7.5 Editace custom fieldů (`CustomFieldUpdateScreen`)

Editace custom fieldů funguje na podobném principu jako editace ticketů. Na obrazovku `CustomFieldUpdateScreen` se prostřednictvím konstruktoru předají 2 parametry: samotný custom field (model `CustomField`) a jeho hodnoty (pole řetězců).

Zatímco u základních políček ticketu mohly více hodnot nabývat pouze ty seznamové, tedy ty s `FieldType.list` (konkrétně sledující a případně statusy), u custom fieldů mohou více hodnot teoreticky nabývat všechny jejich typy. Z toho důvodu přibyla tlačítka plus/mínus pro přidání/odebrání hodnot, které se zobrazí, je-li custom field multiplicitní.

Naopak odpadla nutnost načítat seznamy ze serveru — custom fieldy mají pouze stringové hodnoty, které se vracejí rovnou se schématem custom fieldu.

Navzdory tomu, že existuje mnoho typů custom fieldů, postačí zavedený enumerátor `FieldType` pro zobrazení widgetů repreentujících jednotlivé typy. Je to jednoduše proto, že např. typy custom fieldů *CHECKBOX*, *RADIO*, *SELECTBOX*, *MULTISELECBOX* pokryje `FieldType.list`. Typy *ADDRESS*, *EMAIL*, *PHONE*, *TEXT* a *URL* pak pokryje `FieldType.text`. Ostatním typům (*DATE*, *DATETIME*, *TIME*, *TEXTAREA*) už náleží jejich vlastní `FieldType`.

Na základě `FieldType` se podobně jako při úpravě ticketu vykreslí widgety (např. pro výběr časových hodnot se použije `DateTimePickerService`). Před uložením se provede validace skrze `CustomFieldUpdateBloc` – kromě ověření toho, zda jsou vyplněné povinné custom fieldy, se ověřuje také jejich obsah. Některé typy totiž mohou mít nastavený vzor, který se kontroluje regularními výrazy. Proběhně-li validace bezchybně, upravená data se opět předají prostřednictvím pomocné třídy `UpdateResult` na původní obrazovku, tedy `TicketDetailScreen` nebo `CRMDetailScreen` (viz dále v sekci 4.8).

Dovětek k editaci ticketu: zeditované položky (tedy instance enumerátoru `TicketField` nebo třídy `CustomField`) jsou drženy v mapách a jsou pro uživatele zvýrazněny červenou barvou. Při uložení ticketu se tyto dvě mapy poslány událostí `TicketDetailSaveTicket` do Bloc třídy `TicketDetailBloc`. Při úspěšném uložení ticketu se skrze stav `TicketDetailTicketSaved` přepošle nová (nově načtená) instance třídy `Ticket` a současně se vyprázdní výše zmíněné mapy.

4.7.6 Vytvoření nového ticketu

Vytvořit nový ticket je možné přes tzv. floating action button na obrazovce `TicketFragment` (viz obrázek 4.2). To je plovoucí tlačítko ve spodní části obrazovky, které přesměruje uživatele na obrazovku `TicketDetailScreen`, která je okleštěná oproti verzi, kdy zobrazuje detail konkrétního (již vytvořeného ticketu). Konrétně na horním app baru se nachází jen ukládací tlačítko a po-

chopitelně zcela chybí sekce aktivit. Vyjma priority a stavu jsou všechna pole nového ticketu defaultně prázdná (včetně kategorie, takže zcela chybí custom fieldy). Je nicméně možné (ve webové aplikaci Daktela) nastavit k pohledu u několika základních položek (jmenovitě název, kategorie, uživatel, priorita a statusy) výchozí hodnoty. Jsou-li takové hodnoty nastaveny, pak se rovnou předvyplní v novém ticketu.

4.8 CRM

Další modul přístupný přes navigaci (bottom navigation bar). Dělí se na dva submoduly: kontakty a společnosti, přičemž společnost může mít libovolně mnoho kontaktů a kontakt nejvýše jednu společnost. Vzhledem k tomu, že si jsou oba submoduly poměrně blízké, mají definovanou společnou detailní i editační obrazovku. Editace custom fieldů funguje stejným způsobem jako u ticketů (viz subsekce 4.7.3), proto u tohoto modulu nebude podrobněji rozebírána.

4.8.1 Přehledová obrazovka (CrmFragment)

S ohledem na submoduly kontakty a společnosti tvoří základ obrazovky tab bar, nacházející se pod horním app barem, který umožňuje mezi jednotlivými submoduly přepínat. Je tedy příznačné využít zde výše definovaný widget `BaseScreenTabBar`, kterému jsou mimo jiné předány i instance tříd `TabBar` [77] a `TabBarView` [78]. Jednotlivé taby tvoří widget `CrmTab` a v celém fragmentu je definován jediný Bloc `CrmFragmentBloc`. Jestli se mají načíst kontakty nebo společnosti se rozpozná podle aktuálně zvoleného tabu (s pomocí třídy `TabController` [79]), eventuelně přímo z konkrétního `CrmTab` widgetu, který ví, jakého je typu.

Samozřejmostí každého tabu je také full textové vyhledávání (opět pomocí widgetu `SearchTextField`). Jednotlivé (klikatelné) náhledy, které otevřou obrazovku `CRMDetailScreen`, tvoří widgety `ContactCell` resp. `AccountCell` v závislosti na aktuálně zvoleném tabu tab baru. Jsou to jednoduché bezstatové widgety, které zobrazují název kontaktu resp. společnosti a umí otevřít `CRMDetailScreen` s příslušnými parametry.

4.8.2 Detailní obrazovka (CRMDetailScreen)

Jak již bylo zmíněno, obrazovka `CRMDetailScreen` obshuluje detaily obou submodulů modulu CRM. Je to stavový widget se jmennými konstruktory `contact` a `account`, podle kterých widget rozpozná, s jakým modelem pracuje (jestli je to model `Contact` nebo `Account`).

Podobně jako v detailu ticketu, je možné si i v detailu kontaktu/společnosti zobrazit historii (snapshoty). Tato funkcionalita se opět nachází na horním

app baru a funguje úplně stejně jako v ticketech, tedy konkréítní snapshot se vybírá prostřednictvím obrazovky `SnapshotPickerScreen`.

Pole CRM jsou reprezentována svým enumerátorem `CRMField`, který funguje v podstatě shodně jako enumerátor `TicketField`. U každého políčka je informace o tom, zda je povinné nebo volitelné, jeho název a typ `FieldType`. Dále jsou tu rovněž metody `titleOf` a `nameOf` pro lidsky čitelnou reprezentaci hodnoty políčka a pro tu, které rozumí Daktela API.

Pořadí políček definují gettery `fields` a `newCrmFields`, které se nacházejí v obou modelech. Skrze widget `TitleValueActionCell` jsou pak políčka vykreslena na obrazovce. Vyjma editace hodnoty CRM políčka je jedinou dostupnou akcí otevření detailu společnosti (jedná-li se o model `Contact`, který má nějakou společnost přiřazenou).

Oba CRM modely mají pochopitelně i své vlastní custom fieldy, které ovšem fungují naprosto stejně jako v modulu Tickety 4.7.2. Situace v CRM je dokonce ještě jednodušší, protože oba submoduly mají své jednotné schéma custom fieldů, které stačí načíst jedinkrát – není tedy nutné načítat schéma pro každou kategorii jako je to u ticketů.

Pod rozbalovacím boxem s jednotlivými políčky je sekce tzv. externích kontaktů, pokud jsou nějaké takové kontaktem propojeny. Externí kontakty jsou uživatelé chatovacích služeb, které jsou integrované do systému Daktela. Konkrétně se jedná o služby Messenger, WhatsApp a Viber. Odbaví-li zákazník na své ústředně chatovou konverzaci z některé ze zmíněných služeb (za předpokladu, že má povolenou integraci těchto služeb), je pak možné odesílatele zprávy (tedy účet na platformách Messenger, WhatsApp nebo Viber) spárovat s kontaktem v Daktele. Při další příchozí zprávě bude již tento kontakt třetí strany spárován a operátor se hned dozví, kdo je odesílatelem zprávy. Stejně tak je možné těmto spárovaným kontaktům odeslat zprávu prostřednictvím Daktely, což je právě akce, kterou je možné zahájit z této sekce v detailu CRM kontaktu. Více v sekci věnované chatům 4.9.6.

Poslední sekcí je přehled aktivit a ticketů propojených s CRM modelem (kontaktem nebo společností). Přepínaní mezi aktivitami a tickety je řešeno již zmiňovaným nativním Flutter widgetem `TabBar` [77]. S ohledem na to, že v budoucnu přibudou do přepínače ještě další taby, je tvořen enumerátorem `CrmDataSection`. Načítaní aktivit i ticketu je opět řešeno v Bloc třídě (tentokrát `CrmDetailBloc`). V závislosti na tom, jaký tab je aktuálně zobrazený, se zavolá Bloc událost `CrmDetailFetchActivities` nebo `CrmDetailTickets`.

4.8.3 Editace základních CRM polí (CrmUpdateScreen)

Pro editaci základních CRM polí je nutné widgetu `CrmUpdateScreen` poslat editované pole (`CrmField`) a jeho aktuální hodnotu. Editace funguje naprosto

identicky jako editace základních polí ticketu. Data se načítají ze serveru prostřednictvím Bloc třídy (`CrmUpdateBloc`), stejně tak validace probíhá přes Bloc. Nová data se zpět na obrazovku `CrmDetailScreen` předají opět přes `UpdateResult`.

Provedené změny jsou do doby jejich uložení uchovány v mapách a jejich widgety instace `TitleValueActionCell` jsou zvýrazněny červenou barvu. Pro uložení CRM kontaktu/společnosti pak slouží událost `CrmDetailSaveCrm`.

4.8.4 Vytvoření nového kontaktu nebo společnosti

Shodně s modulem Tickety se nový kontakt nebo společnost vytváří přes plovoucí tlačítko (floating action button). V závisloti na aktuálně vybraném tabu se obrazovka `CrmDetailScreen` otevře buď přes jmenný konstruktor `contact` nebo `account`.

4.9 Aktivity

Ve webové aplikaci Daktela (především na backendu) jsou aktivity velmi rozsáhlým a složitým modulem tvořícím gró celého systému. Tato aplikace zatím veškerou práci s aktivitami nepokrývá a nutno říci, že to ani není jejím cílem. Nicméně některé úkony zvládá a jsou popsány právě v této sekci.

4.9.1 Přehledová obrazovka (ActivitiesFragment)

Fragmentová obrazovka Aktivity je další z těch, na kterou se uživatel dostane přes dolní navigační bar (navigátor). Podobně jako v případě `CrmFragment` obrazovky, je i zde využíván widget `BaseScreenTabBar`, jehož taby rozlišují v tomto případě otevřené (přesněji neuzavřené) aktivity podle jejich stavu a navíc jsou zde ještě aktivity zmeškané, což je specifický model popsaný níže.

Taby jsou reprezentovány enumerátorem `ActivitiesFragmentTab` a jsou aktuálně čtyři (otevřené, čekající, odložené a zmeškané), přičemž každý z nich zobrazuje za svým názvem také počet příslušných aktivit. Tyto počty jsou k dispozici v app pull datech, stejně tam jsou tam uloženy i samotné objekty otevřených (stav *OPEN*) a čekajících (stav *WAIT*) aktivit. Na této obrazovce je tedy nutné použít observer `AppPullDataObserver`. Odložené a zmeškané aktivity je nutné načíst přes jejich vlastní endpointy.

Všechny taby mají společnou Bloc třídu `ActivitiesFragmentBloc`, která zajišťuje načtení dat (aktivit) pro prezentační vrstvu. Samotná aktivita je pak reprezentována widgetem `ActivityCell`, který podle očekávání na kliknutí otevře detail aktivity. Taby jsou navíc řazeny dynamicky podle počtu jejich aktivit – zleva doprava od největšího po nejmenší.

Co jednotlivé taby odlišuje nejvíce jsou především akce, které lze s jejich aktivitami provádět. Na zmeškané aktivity může uživatel odpovědět. Nabídku akcí zajišťuje `TitleActionService`, služba definovaná v sekci 4.12. Odpověď na zmeškaný hovor obstarává níže zmíněná služba `CallService`. Odpověď na jakýkoliv typ chatu pak zajišťuje `NewChatScreen` ve spolupráci s `ChatConversationScreen`. Zmeškané aktivity je možné také smazat, má-li na to uživatel právo.

Otevřevnou emailovou nebo chatovou aktivitu lze otevřít a pokračovat – obrazovky `NewEmailScreen` resp. `ChatConversationScreen`. V podstatě identické je to také s odloženými aktivitami.

Zvonící (čekající) aktivity může uživatel přijmout nebo odmítnout. Jedná-li se navíc o hovor a uživatel má k dispozici více zařízení, musí jedno z nich zvolit skrze jednoduché dialogové okno (služba `DevicePickerService`).

4.9.2 Detailní obrazovka (ActivityDetailScreen)

Podobně jako ostatní detailní obrazovky i `ActivityDetailScreen` zobrazuje detaily o modelu, v tomto případě modelu `Activity`. Jednotlivé fieldy k zobrazení mají opět svůj enumerátor, tentokrát `ActivityField`. Vysledné zobrazení informace z modelu aktivity pak obstarává hojně využívaný widget `TitleValueActionCell`.

Typy aktivit se různí, nicméně všechny mají společný zabalovací widget se základními informacemi o aktivitě, které jsou totožné pro všechny typy, vyjma komentáře (u kterého jsou tyto infromace strohé – datum, autor neboli uživatel a případné přílohy). Která pole se mají zobrazit, definuje getter `fields` v modelu aktivity. Součástí těchto základních polí jsou také přílohy, vyskytující se k aktivitě. Speciálním typem fieldu jsou pak přílohy, které mají vlastní cell (widget) `AttachmentsCell`, který rovněž umožňuje tyto přílohy stáhnout do úložiště zařízení.

Vyjma komentáře má každý typ aktivity tzv. *item* objekt, což je objekt vázající se přímo ke konkrétnímu typu aktivity (nejsou to tedy obecné informace, ty jsou v modelu `Activity`). Základem itemu je abstraktní model `ActivityItem`, kde jsou společná data všech modelů těchto itemů. Konkrétními modely pak jsou `ActivityItemCall`, `ActivityItemSms`, `ActivityItemEmail`, `ActivityItemWeb` (příchozí web chaty) a `ActivityItemChat` (chaty Messenger, WhatsApp a Viber). Každý tento model má definovaný getter `fields`, který vrací pole instancí pomocné třídy `ActivityFieldValue`, která obsahuje přilušný `ActivityField` a jeho hodnotu. Vykreslení na prezentační vrstvě pak probíhá identificky jako v případě základních informací o aktivitě (viz výše).

Email nebo ticketový komentář typicky mají také nějaký textový obsah (ve formátu HTML). U těch to typů aktivit se zobrazí uvnitř tzv. `WebView` (prostřednictvím knihovny `flutter_inappwebview` [80]).

Poslední aktuálně podporovanou částí v detailu aktivity je přepis chatů (u chatových aktivit). Zpráva je reprezentována modelem `Message` a její vy-

kreslení zajišťuje widget `MessageCell`, který je popsán níže v subsekci 4.9.6 věnující se chatovým aktivitám.

Aktivitu není možné editovat, ve smyslu upravovat jako např. ticket. Tím je implementace detailu aktivity jednodušší než právě detailu ticketu nebo kontaktu či společnosti. Aktivitu je však možné uzavřít (typicky přes stejné ukládací tlačítko jako ve zmíněných detailních obrazovkách). V detailu aktivity se dají uzavřít pouze hovorové typy aktivity, ostatní typy aktivit se uzavírají buď na svých vlastních obrazovkách nebo je uzavření aktivity součástí komplexnější úkonu (např. odeslání e-mailu). Jedná-li se navíc o e-mailovou aktivitu, může uživatel na tuto aktivitu (zprávu) odpovědět.

Dodatek k uzavírání aktivit: aktivita (respektive její fronta) může mít nastavené statusy. Je-li tomu tak, pak je před uzavřením aktivity nutné je vyplnit. V takovém případě se zobrazí ještě `StatusesPickerScreen` (skrze `StatusesPickerService`) pro vybrání statusů. Fronty aktivit mohou mít také definovány vlastní formuláře, jejichž vyplnění v současnosti mobilní aplikace nepodporuje.

4.9.3 CallService

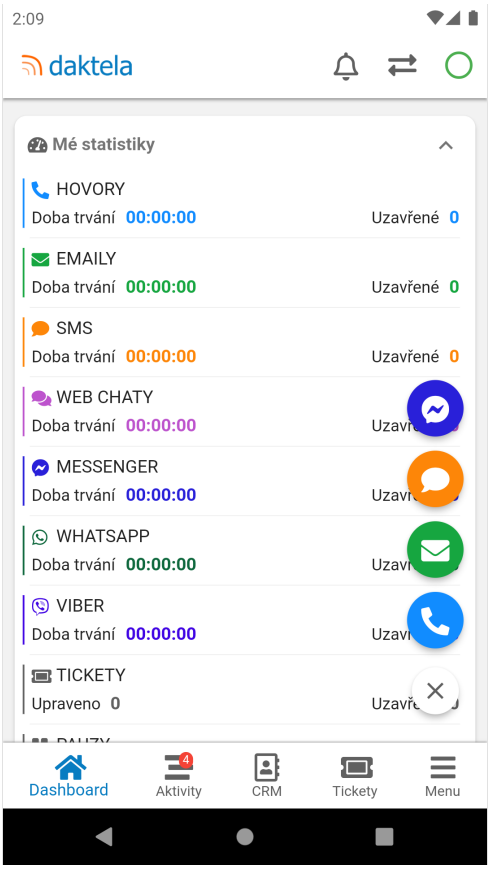
Volání je poměrně komplikovaný proces a v současnosti se aplikace spoléhá na systém Daktela. Nutným předpokladem pro vytočení hovoru z aplikace je mít připojené zařízení, které musí být v online stavu. Zařízením se rozumí softwarový/hardwarový telefon, případně mobilní zařízení. Konfigurace takového telefonu je dostupná v nastavení webové aplikace a je popsána v dokumentaci produktu Daktela [81]. Připojit či odpojit se od zařízení je možné přes obrazovku Fronty a zařízení (4.11.2).

`CallService` je jednoduchá služba zobrazující dialog pro vytočení hovoru. Součástí dialogu je widget `AutocompleteTextField`, který nabízí full textové vyhledávání (v tomto případě kontaktů nebo společností uložených v CRM). Samotnou inicializaci a distribuci hovoru zajišťuje systém Daktela, na straně aplikace stačí poslat požadavek na příslušný endpoint Daktela API.

4.9.4 ActivityExpandableFab

Jedná se o plovoucí tlačítko pro zahájení všech dostupných typů aktivit, které se používá na mnoha obrazovkách napříč aplikací. Nachází se v pravém rohu aplikace a jeho vychozí stav je takový, že je zabalené (tedy je viditelné jediné obecné tlačítko). Kliknutím dojde k jeho rozbalení – zobrazí se ikony všech dostupných aktivit a kliknutím na příslušnou ikonu může uživatel zahájit k ní vázající se typ aktivity (např. při kliknutí na hovor – ikonu sluchátka – se prostřednictvím výše zmíněné služby `CallService` otevře dialogové okno).

Kouzlo tlačítka je v tom, že je univerzální a globalně použitelné (volitelně přijímá několik parametrů, např. ticket nebo kontakt). Implementuje no-



Obrázek 4.3: Rozbalený ActivityExpandableFab

tify metody obou observerů (`AppPullDataObserver` a `WhoImObserver`) a žije si vlastním životem (samo se překresluje). Vidět je pouze, když je uživatel připojený (v ready stavu, pozn. komentář k ticketu je možné napsat i v unready stavu). Jednotlivé ikony aktivit zobrazuje pouze tehdy, když jsou splněny podmínky pro zahájení dané aktivity. Například ikona (ticketového) komentáře je vidět pouze v detailu konkrétního ticketu (neboli když do widgetu parametrem poslána instance modelu `Ticket`). Není-li dostupné uživatelovo zařízení, ikona telefonu se zobrazí červeně a nepůjde na ni kliknout. Nemá-li uživatel dostupnou žádnou e-mailovou frontu, pak se ikona pro vytvoření emailové aktivity vůbec nezobrazí (a nepůjde tedy takovou aktivitu vytvořit).

4.9.5 Napsat e-mail (NewEmailScreen)

Obrazovka umožňující napsat e-mail a odeslat ho přes nějakou frontu. Vzhledem k tomu, že vybírání e-mailové fronty je při psaní e-mailu relativně častý jev (napsat e-mail lze třeba také přes custom field typu *EMAIL*), je pro

tento účel vytvořena služba `EmailService`, která si před otevřením této obrazovky ověří, zda již byla vybrána fronta (odpovídá-li například uživatel na e-mail, předvyplní se fronta z původní aktivity). `QueuePickerScreen` případně umožní uživateli nějakou frontu vybrat.

V hlavičce e-mailu uživatel vybere příjemce zprávy, k čemuž slouží widget `AutocompleteTextField`. Je také možné vybrat příjemce ještě před otevřením obrazovky a ten se následně vyplní do widgetu při otevření obrazovky. Praktické použití je např. u custom fiedlu typu EMAIL v ticketech nebo CRM. Dále je v hlaviččce e-mailu také `CustomTextField` pro vyplnění předmětu. Má-li navíc fronta přiřazené statusy, zobrazí se v hlavičce rovněž také známý `TitleValueActionCell`, který po kliknutí otevře obrazovku pro výběr statusů (`StatusesPickerService`).

K emailu je samozřejmě možné přidat přílohy (prostřednictvím tlačítka na horním app baru, které otevře dialog služby `AttachmentService`). Jsou-li k e-mailu nějaké soubory přiloženy, zobrazí se box s jejich přehledem, ve kterém je možné tyto přiložené soubory odebrat.

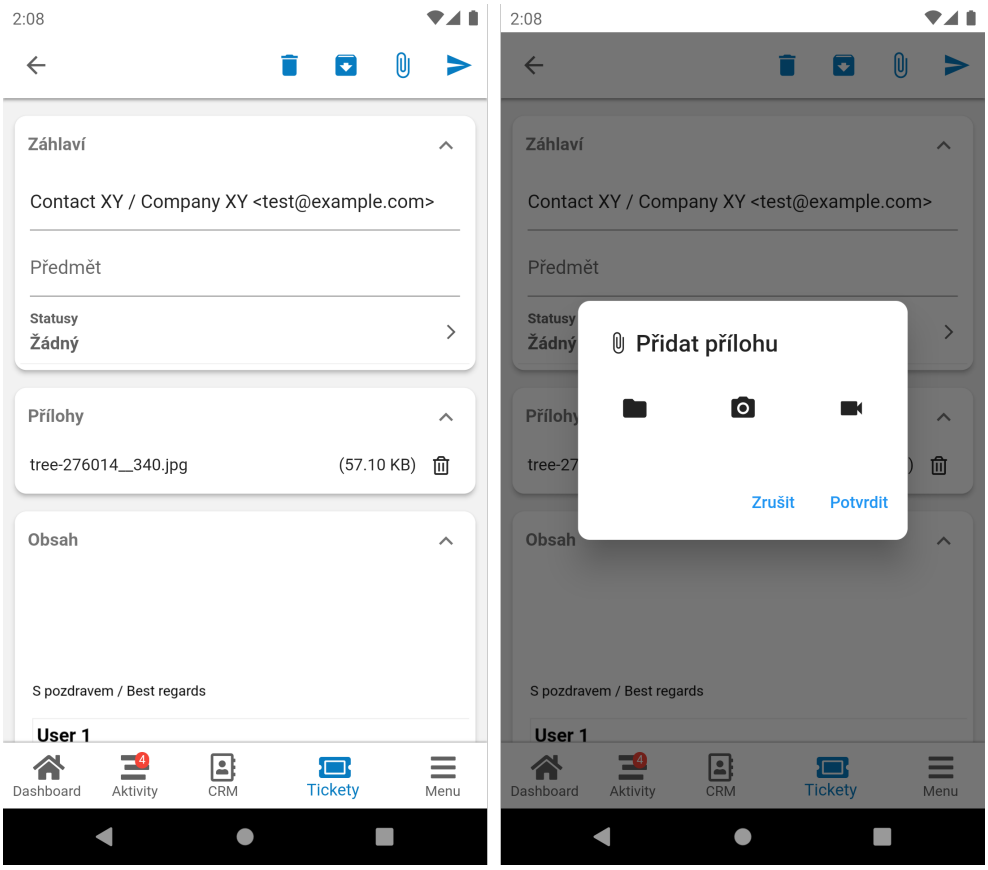
Poslední částí emailové obrazovky je samotný HTML editor. Zajišťuje ho stejný widget jako v případě ticketového komentáře (tedy `CustomHtmlEditor`). K emailové frontě může být mimo jiné přiřazena šablona (HTML nebo plain textová), stejně tak může mít každý uživatel systému Daktela definovaný svůj podpis (tedy vlastně osobní šablonu). Tyto šablony (jsou-li definovány) se vloží do HTML editoru jako defaultní obsah.

E-mail je možné samozřejmě odeslat příjemci, ale je také možné ho odložit (přeplánovat). V takovém případě se zobrazí v tabu odložených aktivit na obrazovce `ActivitiesFragment`. V případě znovuotevření e-mailu se podle očekávání předvyplní data uložená v aktivitě.

4.9.6 Práce s chaty (NewChatScreen a ChatConversationScreen)

V aplikaci je možné pracovat se pěti typy chatů: SMS, webový chat (pouze příchozí), Messenger, WhatsApp, Viber.

Při inicializaci nové chatové aktivity se uživateli nejprve zobrazí obrazovka `NewChatScreen`. Vzhledem k tomu, že všechny typy chatů využívají stejnou obrazovku, rozlišovaní typů aktivit probíhá na základě enumerátoru `ActivityType`. Na obrazovce je pochopitelně textové pole `CustomTextField` pro samotnou zprávu. Nad ním je pak ještě hlavička, v níž je pole pro vyplnění příjemce – `AutocompleteTextField`, který vždy vrací výsledky (kontakty) relevantní k typu aktivity (endpoint `/api/internal/autocomplete.json`). Např. je-li aktivita typu *FBM* neboli Messenger zpráva, hledaní probíhá pouze v rámci externích Messenger kontaktů. Má-li uživatel k dispozici více než jednu frontu, přes kterou zprávu daného typu může odeslat, může frontu zvo-



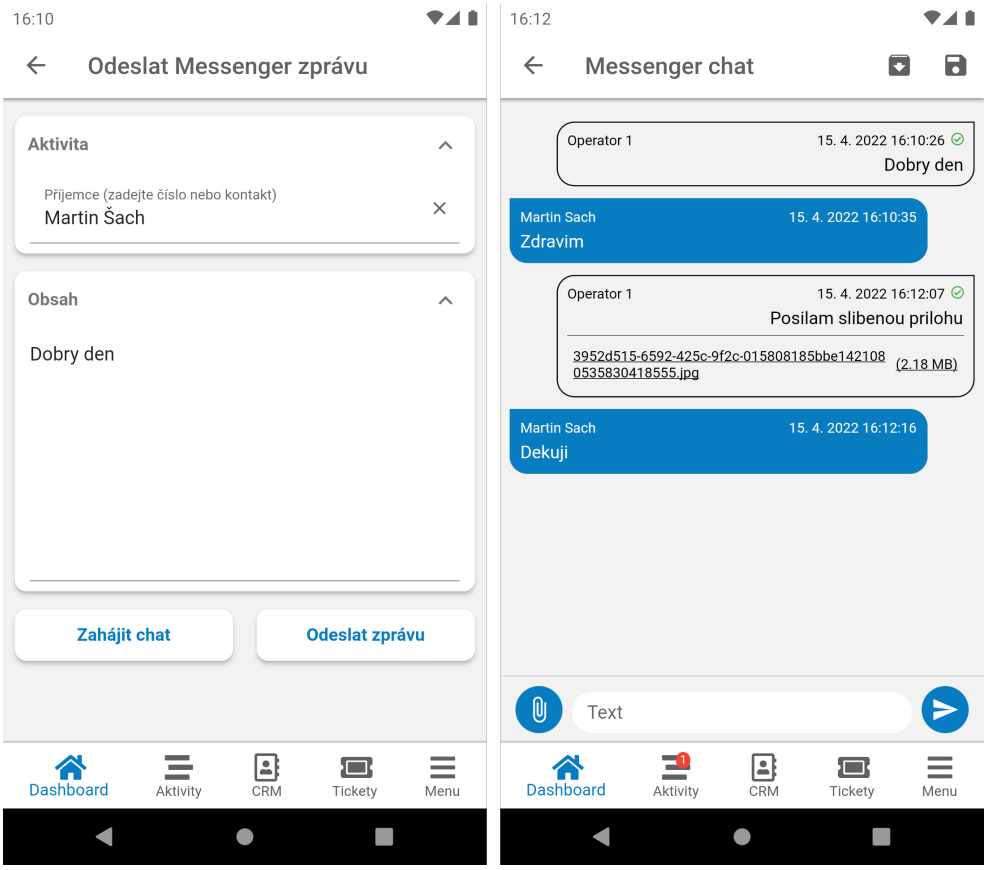
Obrázek 4.4: Ukázka práce s přílohami na obrazovce nového e-mailu

lit pomocí pickeru `QueuePickerScreen` (respektive musí, pokud již nebyla vybrána dříve). Jsou-li navíc u fronty povinné statusy, vybírají se také a to prostřednictvím pickeru `StatusesPickerScreen`.

V dolní části obrazovky jsou pak dvě tlačítka, která určují, co se má stát s aktivitou dále:

- **Odeslat zprávu** znamená, že se odešle pouze jednorázová zpráva a aktivita se uzavře.
- Tlačítko **Zahájit chat** odešle napsanou zprávu a zahájí chat, tedy otevře aktivitu a současně také obrazovku `ChatConversationScreen`.

Zatímco obrazovka `NewChatScreen` slouží pouze pro odchozí chaty, obrazovka `ChatConversationScreen` zajišťuje komunikaci mezi uživatelem a zákazníkem a je tedy využívána i u příchozích chatů. Jednotlivé zprávy jsou reprezentovány již zmiňovaným widgetem `MessageCell`, který přijímá jako



Obrázek 4.5: Obrazovka pro napsání zprávy a obrazovka chatové konverzace

argument jednu konkrétní zprávu (model `Message`). Dle směru zprávy se cell zobrazí buď vlevo (příchozí zpráva) nebo vpravo (odchozí zpráva). Případné systémové zprávy se pak zobrazí uprostřed. U každé zprávy se zobrazí také její přílohy (pokud nejaké má), které je možné stáhnout.

Ve spodní části obrazovky se pak nachází sekce pro poslání nové zprávy: `CustomTextField`, tlačítko pro přidání příloh (`AttachmentService`) a odesílací tlačítko. Tento řádek je prostřednictvím vlastnosti `persistentFooterButtons` widgetu `Scaffold` [72] připnutý ke spodní části obrazovky. V momentě kdy uživatel začne psát zprávu, text field nezmizí, ale bude umístěný bezprostředně nad klávesnicí.

Aby bylo možné zobrazovat příchozí zprávy, widget samozřejmě musí implementovat notify metodu observeru `AppPullDataObserver`. Jakmile dojde k načtení nových zpráv, scroll bar se posune na konec – tedy, aby byly tyto nejnovější zprávy vidět a nebyly schované.

Chatovou aktivitu je možné odložit a znovu otevřít (stejný princip jako v případě emailové aktivity) a je samozřejmě možné ji také ukončit – má-li

její fronta povinné statusy, pak se zobrazí ještě `StatusesPickerScreen` pro vybrání statusů.

4.10 Menu

Jedná se o poslední dosud ještě nezmíněnou fragmentovou obrazovku, tedy tu přístupnou přes navigátor. Navzdory tomu, že se jedná o fragment obrazovku, nemá jednotný horní app bar s ostatními fragmentovými obrazovkami. Namísto akčních tlačítek pro zobrazení, notifikací, správu front a zařízení či nastavení pauzy, je zde jediné tlačítko – tlačítko pro odhlášení z aplikace. Po odhlášení se aplikace přesměruje na přihlašovací obrazovku a současně se také smaže heslo uložené v úložišti telefonu. Ostatní údaje (username a instance) v úložišti zůstanou.

Myšlenkou menu obrazovky je, že může být nejen cestou jak otevřít nové moduly aplikace, ale třeba také osamocené funkcionality, které jinam do aplikace nezapadají. Samotný obsah obrazovky je v tuto chvíli strohý, nicméně postupem času bude právě tato obrazovka místem, kam budou přibývat další moduly.

Jednotlivé položky menu jsou definovány enumerátorem `MenuItem` a jsou to:

- `about` – zobrazí dialog se základními informacemi o aplikaci (verze a kontaktní údaje na dodavatele) a přihlášeném uživateli (uživatelské jméno a aktuální instance),
- `aboutVersion` – jednoduchý dialog s novinkami v aktuální verzi, tyto novinky jsou prostřednictvím Crowdinu přeloženy do všech cílových jazyků (viz 3.6),
- `documentation` – odkaz na webovou dokumentaci k mobilní aplikaci – odkaz se otevře uvnitř již zmiňovaného `WebView`, v aplikaci konkrétně to obstará widget `CustomWebView`.

4.11 Další obrazovky

Tato sekce stručně shrnuje dosud nezmíněné obrazovky, především pak ty z horního app baru. Je-li uživatel připojený (neboli v ready stavu), uvidí na horním app baru tři ikony (viz obrázek 4.2 vlevo) směřující na tři obrazovky: `NotificationsScreen`, `QueuesDevicesScreen` a `PausesScreen`. Dále budou také zmíněny tzv. picker obrazovky sloužící k výběru nějaké hodnoty.

4.11.1 Notifikační centrum (`NotificationsScreen`)

Notifikace upozorňují uživatele na událé změny. Ačkoliv je více typů notifikací, vevodí typ *MESSAGE*, tedy zpráva. Zpráva se může týkat několika

referenčních typů (modelů). Je-li k notifikaci přiložen i tzv. referenční objekt, právě referenční typ určuje, na jaký model se má tento objekt dekódovat z JSON formátu.

Samotná obrazovka je vykreslena jako seznam notifikací, kdy každá jedna notifikace je reprezentována widgetem `NotificationCell`. Je-li navíc referenční typ podporovaný aplikací (neboli existuje detailní obrazovka jejího modelu), je tento cell klikatelný a tuto detailní obrazovku otevře – v současné době detail ticketu, kontaktu nebo společnosti.

V případě, že je notifikace dosud nepřečtená, označí se jako přečtená prostřednictvím Bloc události `NotificationsMarkAsRead`. Stejnou událost je navíc možné zavolat na všechny notifikace – k tomu slouží tlačítko na horním app baru.

4.11.2 Fronty a zařízení (`QueuesDevicesScreen`)

Obrazovka umožňující správu (ve smyslu přihlášení/odhlášení) front a zařízení. Fronty, respektive být v nich přihlášený, je klíčem ke komunikaci se zákazníkem skrze systém Daktela.

Jedná se o dva submoduly a opět se tedy nabízí rozdělení na taby pomocí obrazovky `BaseScreenTabBar`. Oba submoduly používají shodný Bloc `QueuesDevicesBloc`.

Tab s frontami (`QueuesTab`) zobrazuje všechny fronty, na které má uživatel práva. Fronty jsou vypsány prostřednictvím `CheckboxListTile` [82], což je widget, který vedle samotného názvu zahrnuje i check box. Když je tento checkbox zaškrtnutý, znamená to, že je uživatel aktivní (přihlášený) v dané frontě. Změnit tento stav je možné prostým kliknutím na zmiňovaný widget. V jakých frontách je uživatel aktuálně přihlášený je možno vyčíst z workera `AppPullData`. To je také důvodem, proč je tento tab jejich pozorovatelem (observer) a jakmile dojde k zavolání notify metody, celý tab se překreslí. Nad samotným seznamem front je navíc dropdown seznam, umožňující filtrovat fronty podle jejich typu. Implementovaný je pomocí již zmiňovaného `DropdownButton` [73].

Druhý tab (Zařízení) se skládá ze tří částí. V první části jsou zařízení, se kterými je uživatel aktuálně propojen (do kterých je přihlášen). Není-li k nim přihlášen staticky (ale dynamicky), je možné se od zařízení také odhlásit. K zařízení je rovněž možné se přihlásit – druhá část. To zajišťuje tzv. picker obrazovka `AvailableDevicesPickerScreen`, přes kterou si uživatel vybere dostupné zařízení a rovnou se k němu přihlásí. Pro propsání těchto změn tab implementuje notify metodu observeru `WhoImObserver`. Konečně je možné pomocí textového pole (poslední část) nastavit tzv. *forwarding number*, tedy GSM číslo na které budou přesměrovány příchozí hovory.

4.11.3 Pauzy (`PausesScreen`)

Pauza umožňuje uživateli (operátorovi) zneaktivnit se v rámci systému. Pokud tak uživatel učiní, nebudou na něj směřovány (příchozí) aktivity. Pauzy mohou být automatické i manuální (uživatel si takovou pauzu nastavuje ručně) a pro účely statistik se dělí na placené a neplacené. Obrazovka funguje jako tzv. picker screen, tedy že po úspěšně provedené akci se sama zavře. Možné akce jsou:

- nastavit si, případně změnit si pauzu,
- zrušit si aktuálně nastavenou pauzu,
- odpojit se, tedy nastavit se do nepřipraveného stavu – v takovém případě není možné obsluhovat aktivity a aby se do něho uživatel mohl dostat, musí mít všechny své aktivity uzavřené.

Widget pro vykreslení samotné pauzy se podobá již zmíněnému check boxu v tabu `QueuesTab`, ale tentokrát je to radio tlačítko `RadioListTile` [83].

4.11.4 Výberové (picker) obrazovky

Picker obrazovkami se aplikaci rozumějí obrazovky, které slouží k vybrání nějaké hodnoty, která je pak v aplikaci dále používána, ale již mimo kompetence této obrazovky. Tyto obrazovky jsou obvykle spojeny s jednoduchou Bloc třídou, která zajišťuje načtení dat ze serveru. V závisloti na typu vybíraných dat mohou být jednotlivé datové záznamy reprezentovány check boxy, radio buttony nebo jako obyčejný text. Stejně tak u těch typů dat, kde má smysl používat full textové vyhledávání, je k dispozici také vyhledávač `SearchTextField`. Po vybrání hodnoty picker obrazovky ve většině případů zmizí a hodnota se na předešlou obrazovku propíše prostřednictvím třídy `UpdateResult`.

První takovou obrazovkou je widget `SnapshotsPickerScreen`, který slouží k vybrání snapshotu (historie) nějakého modelu, respektive jeho konkrétního objektu (v současnosti to může být ticket, kontakt nebo společnost).

V aktivitách je častou operací nastavování statusu. Proto vznikla picker obrazovka pro jeho vybrání – `StatusesPickerScreen`. Statusy se definují pro konkrétní frontu, je tedy nutné obrazovce předat i `Queue` objekt. Vzhledem k tomu, že se tato picker obrazovka používá v aplikaci na několika místech, její zobrazení a posbírání hodnot zajišťuje služba `StatusesPickerService` s jedinou metodou `show`, která je asynchronní a vrací seznam vybraných statusů, tedy seznam instancí třídy `Status`.

Velmi podobně jako předešlé obrazovky funguje také obrazovka pro výběr front `QueuePickerScreen`, která načte fronty nějakého konkrétního typu a prostřednictvím `UpdateResult` předá vybranou frontu předešlé obrazovce.

4.12 Manažery a služby

Aplikace používá mnoho vlastních manažerů a služeb. Některé již byly zmíněny (např. `NetworkManager`, `SharedPreferencesManager` nebo `EncryptManager`) a některé další jsou rozepsány v této sekci.

4.12.1 StoreManager

Jedná se o manažer, který udržuje data globálně používaná v aplikaci. Zatímco zmiňovaný `SharedPreferencesManager` ukládá data do persitentního úložiště zařízení, `StoreManager` udržuje tzv. *runtime* data, tedy data využívaná za běhu aplikace. Jsou to např. lokalizace (kvůli formátování data a času), URL adresa aktuální instance, access token přihlášeného uživatele. Dále pak aktuální data z workerů (modely `AppPullData` a `WhoIm`) a z nich extrahovaná oprávnění přihlášeného uživatele. Rovněž se sem kešují opakovaně načítaná data jako schémata custom fieldů z modulů kontakty a společnosti a také schémata jednotlivých ticketových kategorií nebo ticketové pohledy.

4.12.2 FirebaseManager

Manažer obsluhující Firebase knihovny. Probíhá zde inicializace Firebase Analytics stejně jako inicializace Firebase Crashlytics. Obecněji byl nástroj Firebase představen v sekci 3.5.

4.12.3 TitleActionService

Relativně často je možné po kliknutí na widget provést více než jednu akci (případně lze provést akci jedinou, ale není žadoucí, aby se provedla ihned po kliknutí na widget). Příkladem může být kontakt v ticketu – když uživatel klikne na jeho `TitleValueActionCell`, může buď kontakt upravit nebo otevřít jeho detail (obrazovka `CrmDetailScreen`). Přesně pro tyto účely je vhodná služba `TitleActionService`. Je to prostý alert dialog zobrazující dostupné akce, ze kterých si uživatel jednu vybere.

Akce je reprezentována třídou `TitleAction`, které uživatel předá její název a samotnout akci (callback). Volitelný je argument `confirmationRequired`, který slouží k ověření toho, že chce uživatel danou akci skutečně provést (defaultně však není aktivní). Využití tento argument nalézá například u akce smazat zmeškanou aktivitu, kdy se aplikace ještě jednou ujistí, že uživatel takovou akci chce skutečně provést.

Služba je v aplikaci hojně využívána, nejčastěji pravděpodobně ve widgetech `TitleValueActionCell`, které vykreslují data z modelů na detailních obrazovkách a umožňují s nimi provádět nejrůznější akce.

4.12.4 AttachmentService

Přidat přílohy je možné typicky k aktivitám – komentáři, e-mailu nebo chatu. Pro nahrání příloh je proto připravená služba (dialog) `AttachmentService`. Dialog umožňuje nahrát přílohy z dokumentů nebo z galerie, případně pořídit fotografii či nahrát video (prostřednictvím různých tlačítek). Seznam vybraných souborů se v dialogu zobrazuje pod akčními tlačítky. Jakmile uživatel dokončí výběr souborů a klikne na tlačítko *Potvrdit*, soubory se nahrají na server (zatím pouze do dočasného úložiště). Návratovou hodnotou metody `show`, která zobrazuje dialog, je seznam nahraných souborů (model `ServerFile`).

Spárování nahraných souborů s konkrétní aktivitou pak zajišťuje Bloc příslušeného typu aktivity. Na obrazovce aktivity je přiložené soubory možné také odstranit – pouze v aplikaci, na serveru se nepoužité soubory po čase smažou automaticky. Právě nahrané soubory v aplikaci reprezentuje widget `AttachmentCell`.

4.12.5 DownloadService

Stejně jako je možné soubory nahrát, je žádoucí, aby je aplikace uměla i číst (stáhnout). Zajišťuje to služba `DownloadService`, která využívá knihovnu *flutter_downloader* [84].

Služba si umí vyžádat přístup k úložišti aplikace, pokud ho již uživatel neudělil dříve. Umí iniciovat stahování, případně ho opakovat. Prostřednictvím zmíněné knihovny je také možné sledovat průběh stahovaní – k tomu slouží pomocná třída `DownloadTaskInfo`, která reprezentuje informaci o aktuálním stavu úlohy. Cílové obrazovky (např. detail aktivity) sledují průběh stahování prostřednictvím portu respektive jeho listeneru, kdy callback definovaný v `DownloadService` zajišťuje rozesílaní zpráv (o průběhu stahování). Tento callback se registruje ve třídě `FlutterDownloader` zmíněné knihovny při inicializaci aplikace.

4.12.6 DateTimePickerService

Jak název napovídá, jedná se o službu (dialog) pro výběr data a času. Obsahuje dvě základní metody `showDateDialog` resp. `showTimeDialog`, které volají nativní Flutter funkce `showDatePicker` resp. `showTimePicker` z knihovny *material* [85]. Třetí metodou služby pak `showDateTimeDialog`, která obě zmíněné metody kombinuje.

4.12.7 ClipboardService

Služba umožňující zkopírovat text do schránky. V aplikaci aktuálně používána ke kopírování hodnoty z `TitleValueActionCell` a názvu kontaktu/společnosti z CRM fragmentu (`ContactCell` nebo `AccountCell`).

4.13 Desktop aplikace a webové rozšíření

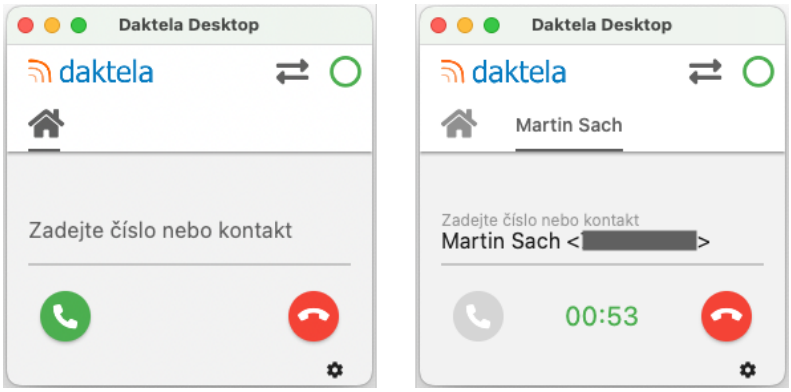
Vzhledem k tomu, že Flutter nativně podporuje nejen tvorbu aplikací pro mobilní zařízení, ale také pro operační systémy macOS, Windows a Linux a rovněž také web, rozběhnout demo verze desktop aplikace a webové rozšíření nebylo nijak zásadně obtížné. Hlavním úkolem, a pro samotné rozběhnutí aplikace v zásadě jediným, je vyvarovat se používání balíčků, které nepodporuje cílová platforma. Případně se vyvarovat volání nativních metod a vlastností, které nejsou podporovány. Jedním příkladem za všechny může být knihovna *firebase_core* [54] výše popsaného nástroje Firebase, která v současnoti nepodporuje platformy Windows a Linux.

Samotná Flutter aplikace je tedy (při splnění určitých podmínek) schopna fungovat i na jiných než mobilních platformách. První verze nové desktopové Flutter aplikace pokrývala stejné funkcionality a měla stejný design jako mobilní aplikace a bylo by ji možné použítat v podstatě stejně jako mobilní aplikaci. Nicméně jak již bylo zmíněno v analýze (2.2.2), cílem desktop aplikace není pokrýt co nejvíce funkcionalit, ale naopak udělat aplikaci minimalistikou, aby ji uživatelé mohli bez problému použít např. vedle jiného (primárního) systému. Pro účely desktop aplikace proto vznikly nové widgety (obrazovky) `DesktopDashboardScreen` a `DesktopTab`.

Prvně jmenovaná obrazovka zcela nahrazuje původní fragment obrazovky včetné dolního navigátoru. Jednotný s mobilní aplikací tak zůstal pouze horní app bar a jeho obrazovky pro `QueuesDevicesScreen` a `PausesScreen` (notifikace jsou skryty). Jejím základem je tab bar (`BaseScreenTabBar`), který zobrazuje domovskou obrazovku a pak obrazovky otevřených nebo zvonících aktivit. Na jakoukoliv změnu týkající se aktivit uživatele, reaguje aplikace pomocí mnohokrát zmíněného observeru `AppPullDataObserver`. Úkolem obrazovky `DesktopDashboardScreen` je na tyto změny reagovat a otevírat, případně uzavírat, taby reprezentované widgetem `DesktopTab`.

`DesktopTab` se mění v závislosti na typu aktivity a její akci. Zvoní-li uživateli nějaká aktivita (neboli její atribut `action` je ve stavu *WAIT*), pak se kromě základních informací o aktivitě zobrazí také tlačítka pro přijmutí a odmítnutí aktivity. Otevřenou (přijmutou) nehovorovou aktivitu není možné v desktop aplikaci obsloužit, nicméně je možné ji přes tlačítko otevřít ve webové aplikaci Daktela. Hovorové aktivity obloužit lze (samozřejmě za předpokladu, že má uživatel aktivní hardwarové nebo softwarové zařízení, které hovor dokáže obsloužit). Stejně tak lze hovor i vytočit – domovský (defaultní) tab se totiž velmi podobá tabu hovorové aktivity a je na něm umístěný mimo jiné i widget `AutocompleteTextField`.

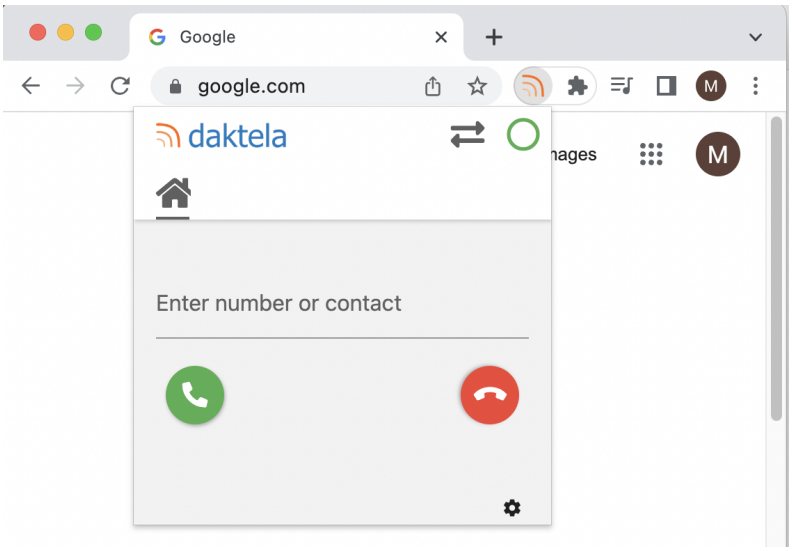
S ohledem na to, že původní desktop aplikace a webové rozšíření si byly velmi podobné, se logicky nabízí myšlenka zachovat to i ve Flutter aplikacích a využít tak uživatelské rozhraní výše zmíněné desktop aplikace.



Obrázek 4.6: Flutter desktop aplikace – základní tab a tab otevřené aktivity (hovoru)

I přes to má webové rozšíření jistá specifika. Tzv. *content skripty* jsou součástí rozšíření, avšak běží přímo na nějaké webové stránce. Jejich opakem jsou pak *background scripty*, které běží na pozadí rozšíření, mají přístup k jeho API [15], ale nemohou přímo přistupovat k obsahu webových stránek (na rozdíl od *content skriptů*). [86]. Kromě toho je specifikem také již zmiňované javascriptové API, nicméně Flutter poskytuje knihovnu [87], pomocí které se dá ve Flutteru (respektive Dartu) zavolat javascriptový kód nebo naopak.

V rámci této práce se podařilo zprovoznit webové rozšíření v demo módu, který pokrývá stejné funkcionality jako výše zmíněná dekstop aplikace, ale prozatím vůbec nepracuje s API pro webové rozšíření.



Obrázek 4.7: Ukázka dema webového rozšíření

Kapitola 5

Testování a nasazení aplikace

Před nasazením jakékoliv aplikace (či informačního systému) do produkčního prostředí je zcela nezbytné a nutné ji řádně otestovat. Vzhledem k povaze projektu není nutné vymýšlet testovací scénáře od nuly, protože již jednou vymyšleny byly (u předchozích verzí aplikací), a postačí tedy stávající scénáře upravit a případně přidat nové, které pokryjí nové funkcionality. Naopak vzhledem k tomu, že je mobilní aplikace pro obě platformy tvořena (až na výjimky) jednotným kódem, drtivou většinu funkcionalit není třeba testovat na každé platformě zvlášť. U desktop aplikace a webového rozšíření je tomu stejně tak.

Testování aplikací, jimiž se zabývá tato práce (mobilní aplikace, desktop aplikace a webové rozšíření), má v kompetenci QA (Quality Assurance) tým společnosti. Sepsání testovacích scénářů pak přímo plyne ze zadání příslušné funkcionality – jejího případu užití a očekávání uživatele – typicky vycházející z webové aplikace Daktela, která je základem všech zmíněných aplikací.

Samotný proces testování těchto „doplňkových“ aplikací k hlavní aplikaci Daktela doposud probíhal manuálně. Pro každou platformu byla vytvořena vlastní aplikace a kromě údržby samotných aplikací by byla údržba jejich automatických testů značně náročná a neefektivní. I v tomto odvětví tak multiplatformní aplikace má smysl. Flutter nativně podporuje 3 typy testů:

- **Unit testy** testující funkčnost základních logických jednotek (např metody nebo třídy), přičemž vnější závisloti jsou obvykle *mockovány* (nasimulovány). Jedná se o nejjednodušší formu testování.
- **Widgetové testy** sloužící k ověření toho, že UI příslušného widgetu vypadá tak jak má a stejně tak, že widget na akce uživatele reaguje podle očekávání.
- **Integrační testy** jsou nejkomplexnějším a na údržbu nejnáročnějším typem testování. Testují, zda kompletní aplikace, případně její větší část, funguje dle očekávání (všechny její widgety mezi sebou správně

interagují). Obvykle tyto testy běží přímo na cílovém zařízení, případně na jeho simulátoru. [88]

S automatickými testy počítají také vývojaři vzoru BLoC a vystavují pro tyto účely knihovnu `bloc_test` [89].

V současné době tyto automatické testy ještě nejsou v aplikaci používány, nicméně v budoucnu by měly být postupně doplňovány a měly by tak ušetřit práci testerům, zkvalitnit aplikaci a zefektivit nejen release proces, ale také samotný vývoj.

Flutter aplikace již několik měsíců běží v produkčním prostředí na mobilních platformách Android a iOS a jsou do ní postupně vyvíjeny nové funkcionality. Proces nasazení mobilní aplikace do produkce se v zásadě skládá ze dvou kroků:

- vytvoření (zaregistrování) aplikace, aby byla dostupná v App Store a Google Play,
- zbuildování aplikace a její nahrání do systému zajišťujícího distribuci aplikací ke koncovým zákazníkům (Google Play Console a App Store Connect).

Tento proces je zjednodušený, protože aplikace byly již dříve nasazeny v produkčním prostředí na obou platformách. Všechny potřebné kroky pro úspěšné zbuildování aplikace (jako např. nastavení názvu, označení verze, podepsání) jsou detailně popsány ve Flutter dokumentaci (Android [90] a iOS [91]) a nebudou zde detailně rozebrány. Samotný build aplikace je pak možné provést prostřednictvím příkazů v konzoli: `flutter build appbundle` pro Android a `flutter build ipa`.

Závěr

Tato práce pojednává o možnostech multiplatformních frameworků a jejich praktickém využití. Zákazníci využívající softwarový systém Daktela mohou efektivně komunikovat s jejich vlastní klientelou prostřednictvím mnoha komunikačních kanálů (hovory, e-maily, SMS, chaty). Primárním produktem systému je rozsáhlá webová aplikace. Dalšími produkty jsou pak mobilní aplikace pro oba hlavní operační systémy Android a iOS, desktop aplikace a webové rozšíření (rozšíření pro webový prohlížeč), které však byly doposud vyvíjeny odděleně. Hlavním cílem práce bylo sjednotit tyto další služby ekosystému Daktela do jediné multiplatformní aplikace. Konkrétněji vytvořit mobilní verzi po vzoru té existující – dle funkčních (F1-F5) a nefunkčních (N1-N3) požadavků, otestovat ji a konečně nasadit do produkčního prostředí. Dále vytvořit demo verze desktop aplikace a webového rozšíření.

V práci jsou nejprve zanalyzovány předešlé produkty ekosystému, shrnuta konkureční řešení i podobné práce a popsány požadavky zadavatele. Následně jsou rozebrány dostupné multiplatformní technologie, přičemž tou finálně zvolenou je framework Flutter. Dále jsou postupně popsány návrh a architektura aplikace, samotná implementace a konečně proces jejího testování a nasazení do produkčního prostředí.

Hlavním cílem bylo dodat multiplatformní mobilní aplikaci dle zmíněných požadavků zadavatele a nasadit ji do produkčního prostředí. Tento cíl byl splněn, mobilní aplikaci již používají zákazníci v reálném provozu. Nejen, že byly splněny všechny zmíněné požadavky (F1-F5 a N1-N3), ale po dohodě se zadatelem se v rámci zhotovení práce podařilo přidat do aplikace další funkcionality – např. možnost psát zprávy přes všechny chatovací kanály nebo číst a spravovat notifikace v notifikačním centru. Dalším cílem bylo zhotovit demo aplikace pro platformy desktop a web (respektive rozšíření pro webový prohlížeč). To se podařilo také, v případě desktop aplikace se navíc podařilo funkcionálně dorovnat předešlou verzi a v blízké době (po vyladění UI) by aplikace měla být přípravena pro zákazníky.

Výhledem do budoucna je doladit desktop aplikaci a rozvinout rozšíření

pro prohlížeč tak, aby obě tyto aplikace mohly být nasazeny v produkčním prostředí. Dále samozřejmě přidávat další funkcionality a moduly do mobilní aplikace tak, aby uspokojila co nejširší spektrum zákazníků – nejzajímavější funkcionalitou je integrace SIP knihovny, která by zákazníkům umožňovala volat skrze WebRTC přímo přes aplikaci. V neposlední řadě by mělo dojít také k zavedení standardního procesu pro automatické testování celé aplikace.

Literatura

[1] Rozdělení mobilních operačních systémů dle jejich zastoupení na trhu (globálně) [online]. *Statcounter GlobalStats*, Prosinec 2021, [cit. 2022-01-18]. Dostupné z: `https://gs.statcounter.com/os-market-share/mobile/worldwide/#monthly-202112-202112-bar`

[2] Rozdělení mobilních operačních systémů dle jejich zastoupení na trhu (v České republice) [online]. *Statcounter GlobalStats*, Prosinec 2021, [cit. 2022-01-18]. Dostupné z: `https://gs.statcounter.com/os-market-share/mobile/czech-republic/#monthly-202112-202112-bar`

[3] Oracle: *What is Java technology and why do I need it? [online]*. [cit. 2022-02-13]. Dostupné z: `https://www.java.com/en/download/help/whatis_java.html`

[4] Android Developer Documentation: *AlertDialog [online]*. [cit. 2022-02-07]. Dostupné z: `https://developer.android.com/reference/android/app/AlertDialog`

[5] Android Developer Documentation: *DatePickerDialog [online]*. [cit. 2022-02-07]. Dostupné z: `https://developer.android.com/reference/android/app/DatePickerDialog`

[6] Android Developer Documentation: *TimePickerDialog [online]*. [cit. 2022-02-07]. Dostupné z: `https://developer.android.com/reference/android/app/TimePickerDialog`

[7] Daktela: *Doporučovaný VOIP sortiment [online]*. [cit. 2022-02-07]. Dostupné z: `https://www.daktela.com/voip-sortiment/`

[8] *Zoiper [online]*. [cit. 2022-02-07]. Dostupné z: `https://www.zoiper.com`

[9] *MicroSIP [online]*. [cit. 2022-02-07]. Dostupné z: `https://www.microsip.org`

[10] Závrbský, M.: Softwarový proces [online]. 2021, soubor přístupný po přihlášení do sitě ČVUT [cit. 2022-02-07]. Dostupné z: https://campuscvut.sharepoint.com/sites/Predmet-B202-NI-PIS/Vukov%20materily/NIPIS_slides_12_2021_SWLifecycle.pdf

[11] Apple: *About Swift [online]*. [cit. 2022-02-13]. Dostupné z: https://www.swift.org/about/

[12] Oracle: *What Is JavaFX? [online]*. [cit. 2022-02-13]. Dostupné z: https://docs.oracle.com/javafx/2/overview/jfxpub-overview.htm

[13] Oracle - Java Documentation: *TabPane (JavaFX) [online]*. [cit. 2022-02-13]. Dostupné z: https://openjfx.io/javadoc/14/javafx.controls/javafx/scene/control/TabPane.html

[14] MDN Web Docs: *Browser Extensions [online]*. [cit. 2022-02-14]. Dostupné z: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions

[15] MDN Web Docs: *Browser Extensions API [online]*. [cit. 2022-02-14]. Dostupné z: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API

[16] Chrome Developers: *Extensions – API Reference [online]*. [cit. 2022-02-14]. Dostupné z: https://developer.chrome.com/docs/extensions/reference/

[17] MDN Web Docs: *Polyfill [online]*. [cit. 2022-02-14]. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/Polyfill

[18] Sedlář, D.: *Mobilní aplikace demonstrující doporučené postupy a návrhové vzory frameworku Flutter*. Zlín, 2020, str. 62, diplomová práce. Univerzita Tomáše Bati ve Zlíně. Fakulta aplikované informatiky, Ústav informatiky a umělé inteligence. Vedoucí práce Král, Erik. Dostupné z: http://hdl.handle.net/10563/47841

[19] Pokorný, M.: *Multiplatformní mobilní aplikace pomocí technologie Flutter*. Zlín, 2019, str. 83, diplomová práce. Univerzita Tomáše Bati ve Zlíně. Fakulta aplikované informatiky, Ústav počítačových a komunikačních systémů. Vedoucí práce Vala, Radek. Dostupné z: http://hdl.handle.net/10563/44468

[20] Crowdin: *Crowdin Introduction [online]*. [cit. 2022-02-14]. Dostupné z: https://support.crowdin.com/crowdin-intro/

[21] Firebase Documentation: *Firebase Crashlytics [online]*. [cit. 2022-04-17]. Dostupné z: https://firebase.google.com/docs/crashlytics

[22] Firebase Documentation: *Firebase Analytics [online]*. [cit. 2022-04-17]. Dostupné z: `https://firebase.google.com/docs/analytics`

[23] Manchanda, A.: The Ultimate Guide to Cross Platform App Development Frameworks in 2022 [online]. *Net Solutions*, Září 2021, [cit. 2022-02-20]. Dostupné z: `https://www.netsolutions.com/insights/cross-platform-app-frameworks-in-2019/`

[24] The State of Developer Ecosystem 2019 [online]. *JetBrains*, 2020, [cit. 2022-02-20]. Dostupné z: `https://www.jetbrains.com/lp/devecosystem-2019`

[25] The State of Developer Ecosystem 2020 [online]. *JetBrains*, 2021, [cit. 2022-02-20]. Dostupné z: `https://www.jetbrains.com/lp/devecosystem-2020`

[26] The State of Developer Ecosystem 2021 [online]. *JetBrains*, 2022, [cit. 2022-02-20]. Dostupné z: `https://www.jetbrains.com/lp/devecosystem-2021`

[27] 2019 Developer Survey [online]. *Stack Overflow*, 2020, [cit. 2022-02-20]. Dostupné z: `https://insights.stackoverflow.com/survey/2019`

[28] 2020 Developer Survey [online]. *Stack Overflow*, 2021, [cit. 2022-02-20]. Dostupné z: `https://insights.stackoverflow.com/survey/2020`

[29] 2021 Developer Survey [online]. *Stack Overflow*, 2022, [cit. 2022-02-20]. Dostupné z: `https://insights.stackoverflow.com/survey/2021`

[30] Flutter Documentation: *Flutter architectural overview [online]*. [cit. 2022-02-21]. Dostupné z: `https://docs.flutter.dev/resources/architectural-overview`

[31] Developers, G.: Sky: An Experiment Writing Dart for Mobile (Dart Developer Summit 2015) [online]. *Youtube*, 2015, [cit. 2022-02-20]. Dostupné z: `https://www.youtube.com/watch?v=PnIWl33YMwA`

[32] Apple Developer Documentation [online]: *UIViewController*. [cit. 2022-02-21]. Dostupné z: `https://developer.apple.com/documentation/uikit/uiviewcontroller`

[33] Android Developer Documentation: *Activity [online]*. [cit. 2022-02-21]. Dostupné z: `https://developer.android.com/reference/android/app/Activity`

[34] Flutter Documentation: *Introduction to declarative UI [online]*. [cit. 2022-02-21]. Dostupné z: `https://docs.flutter.dev/get-started/flutter-for/declarative`

[35] Team, T.: The history of React Native: Facebook's Open Source App Development Framework. *Teach Ahead [online]*, 2020, [cit. 2022-02-21]. Dostupné z: https://www.techaheadcorp.com/blog/history-of-react-native/

[36] React: *Tutorial: Intro to React [online]*. [cit. 2022-02-21]. Dostupné z: https://reactjs.org/tutorial/tutorial.html

[37] MDN Web Docs: *JavaScript [online]*. [cit. 2022-02-21]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript

[38] React Docs: *Hooks API Reference [online]*. [cit. 2022-02-21]. Dostupné z: https://reactjs.org/docs/hooks-reference.html

[39] React Native Guides: *Introduction [online]*. [cit. 2022-02-21]. Dostupné z: https://reactnative.dev/docs/getting-started

[40] React Native Guides: *Core Components and APIs [online]*. [cit. 2022-02-21]. Dostupné z: https://reactnative.dev/docs/components-and-apis

[41] Microsoft Documentation: *What is Xamarin? [online]*. [cit. 2022-02-21]. Dostupné z: https://docs.microsoft.com/en-us/xamarin/get-started/what-is-xamarin

[42] Team, T. T.: What is Xamarin.Forms? [online]. *Telerik Blog*, 2017, [cit. 2022-02-27]. Dostupné z: https://www.telerik.com/blogs/what-is-xamarin-forms

[43] Apache Cordova Documentation: *Overview [online]*. [cit. 2022-02-27]. Dostupné z: https://cordova.apache.org/docs/en/latest/guide/overview/

[44] Ionic Docs: *Introduction to Ionic [online]*. [cit. 2022-02-28]. Dostupné z: https://ionicframework.com/docs/

[45] Flutter Documentation: *Flutter SDK releases [online]*. [cit. 2022-02-28]. Dostupné z: https://docs.flutter.dev/development/tools/sdk/releases?tab=macos

[46] Dart Documentation: *A tour of the Dart language [online]*. [cit. 2022-03-14]. Dostupné z: https://dart.dev/guides/language/language-tour

[47] Flutter API: *Future<T> class [online]*. [cit. 2022-03-14]. Dostupné z: https://api.dart.dev/stable/2.16.1/dart-async/Future-class.html

[48] Daktela API: *General Information [online]*. [cit. 2022-02-28]. Dostupné z: https://www.daktela.com/apihelp/v6/global/general-information

[49] MySQL Documentation: *SELECT Statement [online].* [cit. 2022-02-28]. Dostupné z: https://dev.mysql.com/doc/refman/8.0/en/select.html

[50] Singh, S.: What is HTTP Long Polling? [online]. *Educative*, [cit. 2022-03-07]. Dostupné z: https://www.educative.io/edpresso/what-is-http-long-polling

[51] Bloc state management library: *Overview [online].* [cit. 2022-03-07]. Dostupné z: https://bloclibrary.dev/#/?id=overview

[52] Pub.dev – The official package repository for Dart and Flutter apps: *flutter_bloc [online].* [cit. 2022-03-07]. Dostupné z: https://pub.dev/packages/flutter_bloc

[53] Flutter API: *Stream<T> class [online].* [cit. 2022-03-07]. Dostupné z: https://api.flutter.dev/flutter/dart-async/Stream-class.html

[54] Pub.dev – The official package repository for Dart and Flutter apps: *firebase_core [online].* [cit. 2022-04-17]. Dostupné z: https://pub.dev/packages/firebase_core

[55] Pub.dev – The official package repository for Dart and Flutter apps: *firebase_crashlytics [online].* [cit. 2022-04-17]. Dostupné z: https://pub.dev/packages/firebase_crashlytics

[56] Pub.dev – The official package repository for Dart and Flutter apps: *firebase_analytics [online].* [cit. 2022-04-17]. Dostupné z: https://pub.dev/packages/firebase_analytics

[57] Flutter API: *flutter_localizations library [online].* [cit. 2022-04-16]. Dostupné z: https://api.flutter.dev/flutter/flutter_localizations/flutter_localizations-library.html

[58] Localizely: *Flutter ARB file (.arb) [online].* [cit. 2022-04-16]. Dostupné z: https://localizely.com/flutter-arb/

[59] Crowdin: *Integrations [online].* [cit. 2022-02-14]. Dostupné z: https://crowdin.com/page/integrations

[60] GitLab: *What is GitLab? [online].* [cit. 2022-02-14]. Dostupné z: https://about.gitlab.com/what-is-gitlab/

[61] Flutter API: *Function runApp [online].* [cit. 2022-03-20]. Dostupné z: https://api.flutter.dev/flutter/widgets/runApp.html

[62] Flutter API: *Widget class [online].* [cit. 2022-03-20]. Dostupné z: https://api.flutter.dev/flutter/widgets/Widget-class.html

[63] Pub.dev – The official package repository for Dart and Flutter apps: *flutter_native_splash [online]*. [cit. 2022-03-20]. Dostupné z: https://pub.dev/packages/flutter_native_splash

[64] Flutter API: *MaterialApp class [online]*. [cit. 2022-03-20]. Dostupné z: https://api.flutter.dev/flutter/material/MaterialApp-class.html

[65] Flutter API: *Navigator class [online]*. [cit. 2022-03-20]. Dostupné z: https://api.flutter.dev/flutter/widgets/Navigator-class.html

[66] Pub.dev – The official package repository for Dart and Flutter apps: *http [online]*. [cit. 2022-03-20]. Dostupné z: https://pub.dev/packages/http

[67] Flutter API: *Library dart:io [online]*. [cit. 2022-03-20]. Dostupné z: https://api.flutter.dev/flutter/dart-io/dart-io-library.html

[68] Flutter API: *State class [online]*. [cit. 2022-03-21]. Dostupné z: https://api.flutter.dev/flutter/widgets/State-class.html

[69] Pub.dev – The official package repository for Dart and Flutter apps: *shared_preferences [online]*. [cit. 2022-03-28]. Dostupné z: https://pub.dev/packages/shared_preferences

[70] Pub.dev – The official package repository for Dart and Flutter apps: *encrypt [online]*. [cit. 2022-03-28]. Dostupné z: https://pub.dev/packages/encrypt

[71] Flutter API: *BottomNavigationBar class [online]*. [cit. 2022-03-30]. Dostupné z: https://api.flutter.dev/flutter/material/BottomNavigationBar-class.html

[72] Flutter API: *Scaffold class [online]*. [cit. 2022-04-04]. Dostupné z: https://api.flutter.dev/flutter/material/Scaffold-class.html

[73] Flutter API: *DropdownButton class [online]*. [cit. 2022-04-09]. Dostupné z: https://api.flutter.dev/flutter/material/DropdownButton-class.html

[74] Flutter API: *TextField class [online]*. [cit. 2022-04-09]. Dostupné z: https://api.flutter.dev/flutter/material/TextField-class.html

[75] Pub.dev – The official package repository for Dart and Flutter apps: *url_launcher [online]*. [cit. 2022-04-15]. Dostupné z: https://pub.dev/packages/url_launcher

[76] Pub.dev – The official package repository for Dart and Flutter apps: *html_editor_enhanced [online]*. [cit. 2022-04-10]. Dostupné z: https://pub.dev/packages/html_editor_enhanced

[77] Flutter API: *TabBar class [online]*. [cit. 2022-04-11]. Dostupné z: https://api.flutter.dev/flutter/material/TabBar-class.html

[78] Flutter API: *TabBarView class [online]*. [cit. 2022-04-11]. Dostupné z: https://api.flutter.dev/flutter/material/TabBarView-class.html

[79] Flutter API: *TabController class [online]*. [cit. 2022-04-11]. Dostupné z: https://api.flutter.dev/flutter/material/TabController-class.html

[80] Pub.dev – The official package repository for Dart and Flutter apps: *flutter_inappwebview [online]*. [cit. 2022-04-11]. Dostupné z: https://pub.dev/packages/flutter_inappwebview

[81] Daktela: *SIP zařízení [online]*. [cit. 2022-04-30]. Dostupné z: https://doc.daktela.com/display/docs/SIP+Devices

[82] Flutter API: *CheckboxListTile class [online]*. [cit. 2022-04-12]. Dostupné z: https://api.flutter.dev/flutter/material/CheckboxListTile-class.html

[83] Flutter API: *RadioListTile class [online]*. [cit. 2022-04-12]. Dostupné z: https://api.flutter.dev/flutter/material/RadioListTile-class.html

[84] Pub.dev – The official package repository for Dart and Flutter apps: *flutter_downloader [online]*. [cit. 2022-04-16]. Dostupné z: https://pub.dev/packages/flutter_downloader

[85] Flutter API: *material library [online]*. [cit. 2022-04-15]. Dostupné z: https://api.flutter.dev/flutter/material/material-library.html

[86] MDN Web Docs: *Browser Extensions - Content scripts [online]*. [cit. 2022-04-25]. Dostupné z: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Content_scripts

[87] Pub.dev – The official package repository for Dart and Flutter apps: *js [online]*. [cit. 2022-04-25]. Dostupné z: https://pub.dev/packages/js

[88] Flutter Documentation: *Testing Flutter apps [online]*. [cit. 2022-02-25]. Dostupné z: https://docs.flutter.dev/testing

[89] Pub.dev – The official package repository for Dart and Flutter apps: *bloc_test [online]*. [cit. 2022-04-28]. Dostupné z: https://pub.dev/packages/bloc_test

[90] Flutter Documentation: *Build and release an Android app [online].* [cit. 2022-02-26]. Dostupné z: `https://docs.flutter.dev/deployment/android`

[91] Flutter Documentation: *Build and release an iOS app [online].* [cit. 2022-02-26]. Dostupné z: `https://docs.flutter.dev/deployment/ios`

Příloha **A**

Seznam použitých zkratek

CRM Customer Relationship Management

UI User Interface

UX User Experience

JSON Javascript Object Notation

CRUD Create Read Update Delete

API Application Programming Interface

REST Representational State Transfer

SDK Software Development Kit

URI Uniform Resource Identifier

URL Uniform Resource Locator

SIP Session Initiation Protocol

VM Virtual Machine

HTML HyperText Markup Language

GSM Global System for Mobile Communications, Groupe Spécial Mobile

Příloha **B**

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF