Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Sběr a odesílání logů z Android zařízení na server

Plzeň 2023 Jan Vašátko

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2022/2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Jan VAŠÁTKO**
Osobní číslo:	**A19B0269P**
Studijní program:	**B0613A140015 Informatika a výpočetní technika**
Specializace:	**Výpočetní technika**
Téma práce:	**Sběr a odesílání logů z Android zařízení na server**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Seznamte se s aktuálními možnostmi logování v Android aplikaci Aimtec Touch Client. Analyzujte uživatelské a systémové požadavky na sběr logů.
2. Prozkoumejte vhodné technologie pro logování a přenos logů z klientských zařízení na server.
3. Navrhněte rozšíření klientské Android aplikace Aimtec Touch Client a serveru DCIx. Definujte komunikační rozhraní mezi nimi.
4. Implementujte navržené změny v klientské aplikaci Aimtec Touch Client, potřebné změny v serveru DCIx a vývoj nové serverové komponenty pro sběr logů. Kód pokryjte vhodnými testy.
5. Ověřte funkčnost vytvořeného řešení a navrhněte možná rozšíření.

Rozsah bakalářské práce: **doporuč. 30 s. původního textu**
Rozsah grafických prací: **dle potřeby**
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

Dodá vedoucí bakalářské práce

Vedoucí bakalářské práce: **Ing. Jiří Dobrý**
Aimtec a. s. , Plzeň

Konzultant bakalářské práce: **Ing. Ladislav Pešička**
Katedra informatiky a výpočetní techniky

Datum zadání bakalářské práce: **3. října 2022**
Termín odevzdání bakalářské práce: **4. května 2023**

L.S.

Doc. Ing. Miloš Železný, Ph.D.
děkan

Doc. Ing. Přemysl Brada, MSc., Ph.D.
vedoucí katedry

V Plzni dne 25. října 2022

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 4. května 2023

Jan Vašátko

Abstract

This bachelor's thesis, named *Collecting and Uploading Logs from Android Devices to a Server*, is related to the area of warehouse management systems (WMS) and deals with uploading of logs from Android devices to a server. The goal was to eliminate the need for physical access to the device and to enable acquisition of client app's logs through web interface. The implementation was done by specifying REST API, setting up Spring Boot application for log collection and establishing a WebSocket connection between the server and mobile device which can be used to request dispatching of logs from the device. The result is a solution which makes the process of logs acquisition significantly simpler and faster. Part of the thesis's text is also devoted to systems used in manufacturing and logistics and also to the topic of logging.

Abstrakt

Bakalářská práce se pohybuje v prostředí systémů pro řízení skladu (WMS) a zabývá se sběrem logů z Android zařízení na server. Cílem řešené úlohy bylo eliminovat nutnost fyzického přístupu k zařízení a umožnit získání logů klientské aplikace pomocí webového rozhraní. Implementace zahrnovala specifikaci REST API, založení Spring Boot aplikace pro sběr logů a vytvoření WebSocket spojení mezi serverem a mobilním zařízením, pomocí kterého lze vyžádat odeslání logů ze zařízení. Výsledkem je řešení, které významně usnadňuje a urychluje proces získávání logů. Součástí textu je také přiblížení systémů používaných ve výrobě a logistice a problematika logování.

Poděkování

Tímto bych rád poděkoval společnosti AIMTEC a.s. za poskytnutí příležitosti zpracovat tuto práci, především pak Janu Klikovi za ochotu, vstřícnost a čas strávený při konzultacích a Ing. Jiřímu Dobrému za přínosné konzultace a odborné vedení práce. Také bych chtěl poděkovat Ing. Ladislavu Pešičkovi ze Západočeské univerzity v Plzni za věcné připomínky k textu práce.

Obsah

1 Úvod

Vytváření logů je často důležitou součástí běžícího počítačového programu. Logy mohou uchovávat důležité informace o běhu programu a jejich analýzou je možné například zjistit, v jakém místě programu došlo k chybě a co chybě předcházelo. Při absenci logů lze jen velmi obtížně reprodukovat stav, který k výskytu chyby vedl, neboť jediné, co může být k dispozici, je výpověď uživatele, která ale popisuje pouze chování programu navenek. V případě programů, které běží bez přímé interakce s uživatelem, pak navíc ani tato výpověď není k dispozici. Existence a rychlá dostupnost logů je tedy u počítačových programů klíčová pro úspěšné odhalování a opravu chyb.

Tato bakalářská práce je řešena jako externí práce zadaná společností AIMTEC a.s. Informační systém DCIx, který tato společnost poskytuje svým zákazníkům, komunikuje s klientskou Android aplikací Aimtec Touch Client používanou zaměstnanci zákaznické firmy při provádění skladových operací. V současné době aplikace vytváří logové soubory, které jsou uchovávány v úložišti daného Android zařízení. Při výskytu chyby je tedy nutné kontaktovat ICT oddělení zákazníka, aby programátorům dané soubory poskytl. Hlavním cílem této práce je usnadnit proces získávání logů tak, aby bylo možné uživatelem systému DCIx získat požadované logy a poskytnout je programátorům bez nutnosti fyzického přístupu k zařízení.

Text práce je členěn do následujících kapitol. Kapitola 2 představuje softwarové produkty, které budou předmětem práce, a popis prostředí v němž jsou provozovány. Navazuje kapitola 3, která tento popis doplňuje pohledem na související hardware. V kapitole 4 je přiblížen koncept logování. Následuje kapitola 5, jež se zabývá analýzou řešené úlohy, kapitola 6 s celkovým pohledem na návrh řešení a poté kapitola 7 s popisem konkrétní implementace. Způsob a výstupy z testování vytvořeného řešení lze nalézt v kapitole 8. Předposlední kapitola 9 popisuje možná rozšíření řešení. Kapitola 10 je poslední a shrnuje text práce a dosažené výsledky.

2 Výroba a logistika

Cílem této kapitoly zcela jistě není popsání všech existujících výrobních a logistických systémů, ale spíše uvedení kontextu, v němž se celá práce bude pohybovat. Jde tedy hlavně o představení systému DCIx a jeho klientské aplikace Aimtec Touch Client. Aby však mohl být poskytnut ucelený obraz, je nejdříve stručně popsáno prostředí výroby a logistiky, v němž se oba produkty používají. Zároveň by zde měl být přiblížen i význam těchto oblastí pro lidskou společnost a také důležitost informačních technologií pro firmy nabízející výrobní a logistická řešení.

Důvodem pro zařazení kapitoly je skutečnost, že jakékoli softwarové řešení bude téměř vždy součástí většího celku, jehož poznání může vést autora ke kvalitnějšímu návrhu a čtenáře textu k lepšímu porozmění.

2.1 Vysvětlení pojmů

Výrobu lze definovat jako „*transformaci výrobních faktorů do ekonomických statků a služeb, které pak procházejí spotřebou*“ [1].

Zjednodušeně lze také říct, že výroba je činnost, při které dochází k přeměně vstupů na výstupy.

Logistika je pojem, který nemá zcela jednotnou definici. Pro představu lze použít například definici Evropské logistické asociace, podle níž je logistika „*organizace, plánování, řízení a výkon toků zboží vývojem a nákupem počínaje, výrobou a distribucí podle objednávky finálního zákazníka konče tak, aby byly splněny všechny požadavky trhu při minimálních nákladech a minimálních kapitálových výdajích*“ [2].

Obecně lze říct, že společným předmětem definic logistiky je tok zboží.

Obě oblasti jsou klíčové pro úspěch firmy na trhu. Správně nastavené výrobní procesy by měly zajistit efektivní hospodaření se zdroji a také vyšší kvalitu výrobků, tedy vyšší spokojenost zákazníků. V dnešní proměnlivé době je pak obzvláště důležitá flexiblita výroby. Sethi a Sethi definují 11 druhů flexibility z nichž lze vybrat např. *flexibilita zdrojů*, *flexibilita zacházení s materiálem* či *flexibilita postupu výroby* [3]. Role, kterou zde hrají informační technologie, je nepostradatelná.

Stejně jako výrobní procesy i logistické procesy mají vliv na hospodaření se zdroji a dále na spokojenost zákazníků s dodáním výrobků. Posuzování

účinnosti logistiky je složitý proces, který je třeba provádět na několika úrovních: účinnost dodavatelského řetězce, účinnost logistického systému (a subsystémů) distribučního centra a účinnost aktivit a procesů. Ty jsou ovlivněny finančními, technickými, environmentálními, energetickými, sociálními a mnoha dalšími faktory [4]. Firma, která chce být úspěšná, by měla při řízení logistiky zohledňovat všechny tyto faktory.

Efektivní řízení výroby i logistiky v dnešní době umožňují podnikové informační systémy. Existuje více typů podnikových informačních systémů zaměřených na výrobu a logistiku, ale mezi ty nejdůležitější, které budou dále popsány, patří SCM, MOM, MES a WMS. Tyto systémy jsou obvykle integrovány s dalšími podnikovými systémy, nejčastěji s ERP.

2.2 Supply chain management

Supply chain management (SCM) představuje řízení dodavatelského řetězce a také software k tomuto účelu používaný. Řízení dodavatelského řetězce zahrnuje řízení všech aktivit, které přeměňují suroviny na hotové výrobky, a také všech aktivit, které zajišťují předání těchto hotových výrobků do rukou zákazníků [5]. Jde tedy o činnost, která propojuje všechny články dodavatelského řetězce (dodavatel, zpracovatel, výrobce, distributor, velkoobchodník, maloobchodník, spotřebitel) v jeden systém. Mezi oblasti SCM patří řízení výroby i řízení logistiky. Dále například nákup, správa životního cyklu produktu, plánování, či správa objednávek [6].

2.3 Manufacturing operations management

Manufacturing operations management (MOM) je řízení výrobních operací, tj. aktivity, které řídí osoby, zařízení a materiál ve výrobě. Nejde jen o řízení operací zaměřených na produkci, ale také na kvalitu, údržbu či skladování.

V kontextu softwaru se jedná o vrstvu informačního systému, která je nejblíže fyzickým procesům [7]. Zatímco SCM se zabývá celým dodavatelským řetězcem, MOM samotné se zaměřuje především na dění uvnitř jedné společnosti. Obě oblasti však mohou být řešeny společně v jednom softwarovém produktu.

Oba systémy MES a WMS uvedené dále mohou být součástí MOM.

2.4 Manufacturing execution system

Manufacturing execution system (MES), který bývá do češtiny překládán jako výrobní informační systém, je systém zaměřený na vykonávání výroby. Jeho typickou vlastností je fungování v reálném čase. Má na starosti například sledování a přidělování výrobních zdrojů, řízení samotné výroby či monitorování a sběr dat o výrobě. Tato data, která získává komunikací s automatizačními systémy na nižší úrovni, pak obvykle předává systému ERP.

2.5 Warehouse management system

Warehouse management system (WMS) je systém pro řízení skladu. Tento systém řídí celý chod skladu, tedy činnosti související s příjmem, zaskladněním, vychystáním a expedicí zboží. Základní úlohy skladového hospodářství lze sice řešit i pomocí systému ERP, WMS však nabízí daleko rozsáhlejší možnosti. Mezi ně patří například hledání optimálních cest pro skladníky, efektivní využívání skladového prostoru, načítání položek pomocí skenerů a kontroly objednávek.

V případě rozsáhlejších skladů mohou tyto možnosti výrazně snižovat náklady a také počet vrácených objednávek. Stále rostoucí zájem společností o WMS dokládá například zpráva společnosti Grand View Research, dle které lze očekávat nárůst velikosti trhu s WMS ze 3 miliard amerických dolarů v roce 2022 na 11 miliard amerických dolarů v roce 2030 [8].

Systémy řízení skladu spolupracují s různými zařízeními a technologiemi, jako jsou skenery čárových kódů, RFID, nositelná zařízení, robotická zařízení či rozšířená realita.

Bližší pohled na hardware související s WMS nabídne kapitola 3.

2.6 Společnost Aimtec

AIMTEC a.s. (dále jen *zadavatel*; logo na obrázku 2.1) je plzeňská společnost specializující se na informační systémy pro výrobu a logistiku. Od svého založení před více než dvaceti lety se rozrostla v globální společnost se zákazníky v odvětví automobilového průmyslu, kovovýroby, průmyslové výroby a mnohých dalších [9].

Obrázek 2.1: Logo Aimtec [10]

Jedním z nejvýznamnějších produktů společnosti je systém DCIx, jenž je předmětem této práce, a je tedy blíže popsán v následující části.

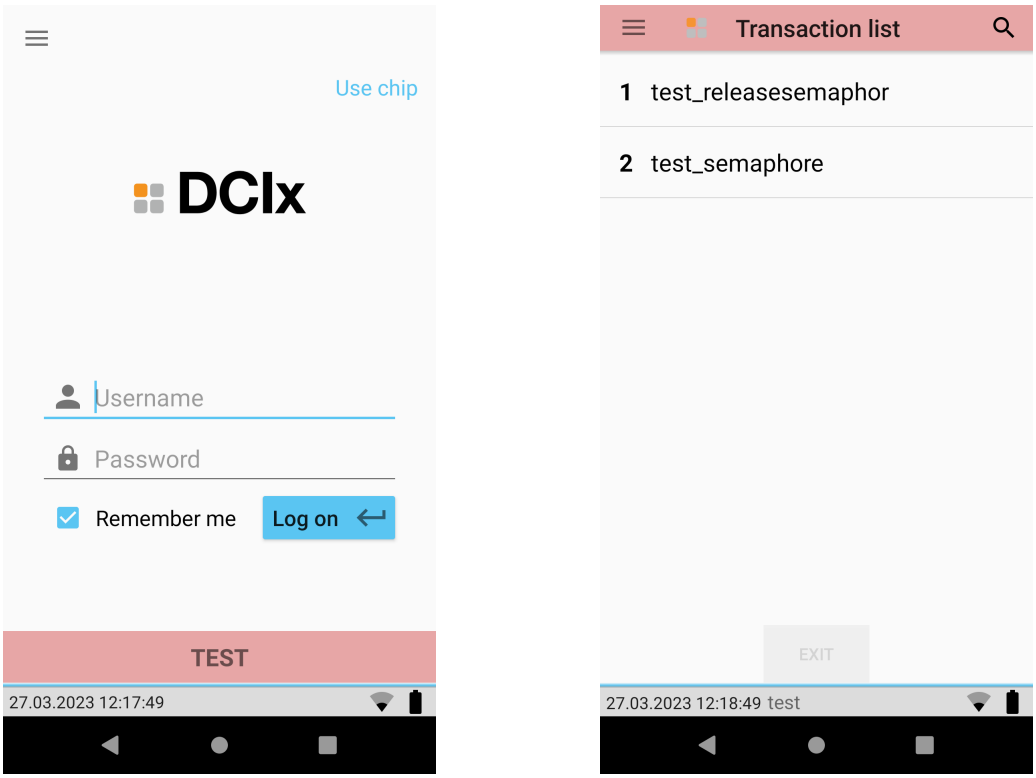
2.6.1 DCIx

DCIx (z anglického Delivery Chain Integrator) je „*produkt z kategorie MOM, který integruje celý dodavatelsko-odběratelský řetězec*“ [11]. Kromě modulů MES a WMS, jejichž význam byl vysvětlen v předchozím textu, obsahuje i několik dalších modulů (QMS, JIT/JIS, PPS, Portal, YMS, MFC), které zde nebudou podrobněji rozebírány.

DCIx je univerzální řešení v tom, že umožňuje nadefinovat pro zákazníka tzv. transakce, které představují procesy a aktivity prováděné v jeho vlastním podniku.

2.6.2 Aimtec Touch Client

Aimtec Touch Client je klientská aplikace, která komunikuje se serverovou částí DCIx a umožňuje zaměstnancům zákaznické firmy ve skladu procházení nadefinovaných transakcí. Aplikace tedy komunikuje s modulem WMS systému DCIx. Vzhled přihlašovací obrazovky aplikace a seznamu transakcí po přihlášení je vidět na obrázku 2.2.



(a) Přihlašovací obrazovka (b) Seznam transakcí

Obrázek 2.2: Aplikace Aimtec Touch Client

V současné době je aplikace dostupná pro operační systém Android a její původní podoba vznikla jako diplomová práce Tomáše Krásného na Fakultě aplikovaných věd Západočeské univerzity v Plzni. Ve zmíněné práci lze najít podrobný popis návrhu a implementace celé aplikace [12]. Od doby vzniku je aplikace aktivně nasazována u zákazníků společnosti Aimtec a dále se vyvíjí. Mezi některé z novějších vlastností patří podpora formátu JSON, zasílání uživatelských zpráv či možnost přizpůsobení uživatelského rozhraní.

3 Mobilní zařízení ve skladech

Softwarové systémy popisované v předchozí kapitole potřebují ke svému fungování odpovídající hardware. Vzhledem k tomu, že se řešená úloha týká oblasti WMS, budou zde nejdříve uvedeny typy zařízení používaných ve skladech a poté zaměření na mobilní terminály.

3.1 Čtečky čárových kódů

Běžným způsobem identifikace položek ve skladu je využití čteček čárových kódů. Typů čárových kódů dnes existuje velké množství, přičemž některé mají široké využití a jiné se používají jen pro speciální účely. V základu je lze rozdělit na jednorozměrné (1D), které mají podobu paralelních čar, a dvojrozměrné (2D), které používají obrazce a umožňují zakódovat větší množství informací.[1] 1D kódy obvykle obsahují pouze základní identifikaci položky a další informace lze získat dotazem na server. V případě 2D kódů mohou být veškeré informace o položce uloženy přímo v kódu. Mezi 1D kódy patří např. EAN či UPC, mezi dvojrozměrné pak např. QR kódy.

Čtečky mohou být buď samostatná specializovaná zařízení, nebo se mohou vyskytovat jako součást jiných zařízení. Může také jít o adaptéry, které lze připevnit k mobilním terminálům. Moderní čtečky nabízejí podporu pro řadu nejznámějších kódů, některé jsou však určené pouze pro čtení 1D kódů. Čtečky 2D kódů bývají obvykle dražší, ale dokaží číst 1D i 2D kódy. Některé umožňují skenování více kódů najednou, skenování z velké vzdálenosti či skenování velmi malých kódů.

3.2 Tiskárny

K vytištění čárových kódů na štítky se používají tiskárny štítků. Ty se vyrábějí v různých velikostech – od velkých industriálních, přes stolní, až po malé přenosné, které mají při pohybování ve skladech velké uplatnění. Jde nejčastěji o tiskárny termální. Tyto tiskárny vytvářejí obraz využitím teplocitlivého materiálu, nepoužívají tedy žádný inkoust. Termální tiskárny se dělí na 2 typy: tiskárny s přímým termálním tiskem a termotransferové tiskárny. Tiskárny s přímým termálním tiskem vytvářejí obraz přímo na

[1] Například dvojrozměrný QR kód dokáže uchovat až 7089 číslic oproti jedorozměrnému UPC, který uchová pouze 12 číslic [13].

teplocitlivém materiálu, který zčerná při průchodu pod tiskovou hlavou. Využívají se pro svou jednoduchost, nevýhodou však může být tmavnutí materiálu při nadměrném vystavení světlu a teplu. V případě termotransferové tiskárny zahřátá tisková hlava nejdříve aplikuje teplo na teplocitlivou pásku, která se roztaví a následně přenáší barvu na požadovaný materiál (např. běžný papír) [14].

3.3 RFID čtečky

Pro identifikaci položek lze využít také technologii RFID. RFID štítky obsahují čip a komunikují s RFID čtečkou pomocí rádiových vln. Štítky lze tisknout a zakódovat do nich informace pomocí speciálních tiskáren. Výhodou oproti čárovým kódům může být např. možnost skenovat přes překážky nebo větší bezpečnost. Nevýhodou jsou vyšší náklady a složitost.

3.4 Mobilní terminály

Mobilní datový terminál je přenosné koncové zařízení určené ke sběru dat a komunikaci se serverem při provádění pracovních úkolů. Jedná se o podnikové (enterprise-grade) zařízení. Často má podobu ručního počítače, který může připomínat klasický spotřební chytrý telefon. Může se ale také jednat o tzv. vehicle mounted computer (vozíkový terminál) či wearable computer (nositelný terminál). Vozíkové terminály připomínají velké tablety, které lze připevnit k vysokozdvižnému vozíku. Měly by tedy být schopné odolávat vibracím při pohybu vozíku. Nositelné terminály jsou pak malá zařízení, která se nejčastěji připevňují na hřbet ruky či kolem zápěstí a jsou vhodná pro práci, která vyžaduje jen občasnou interakci s displejem.

Podniková zařízení mají na rozdíl od spotřebních zařízení software i hardware přizpůsobený pro práci zaměstnanců v podniku. Při výběru zařízení se firma může rozhodnout pro nákup specializovaných podnikových zařízení, spotřebních zařízení nebo pro politiku BYOD (Bring Your Own Device), kdy si zaměstnanci nosí svá vlastní zařízení. Ačkoli nákup specializovaných zařízení může na první pohled vypadat jako nejdražší řešení, může lépe splňovat požadavky a ve výsledku stát méně [15].

U zařízení, která pracovník používá celou pracovní směnu je kladen velký důraz na ergonomii a odolnost. Ruční počítače mohou být vybaveny pistolovou rukojetí, která usnadňuje časté skenování. Také mohou kromě dotykového displeje mít i klávesnici, která může být užitečná při práci v rukavicích (displej sice může být optimalizován pro práci v rukavicích, klávesnice ale i

tak může být výhodou). Zařízení bývají odolná vůči pádu, vlhkosti, prachu či extrémním teplotám. Například výrobce Zebra ve specifikaci zařízení MC9300 (obrázek 3.1) uvádí následující charakteristiky [16]:

- Provozní teplota: −20 °C až +50 °C
- Pád: 3.1 metrů při pokojové teplotě
- Bubnový test: 6000 jednometrových pádů
- Teplotní šok: −40 °C až +70 °C

Důležitá je výdrž baterie, která by měla být dostatečná alespoň pro jednu pracovní směnu (displej zařízení se z praktických důvodů jen málokdy zhasíná). Zajímavá je možnost tzv. hot swap baterie (tj. možnost výměny baterie bez nutnosti vypnutí zařízení).

Obrázek 3.1: Mobilní terminál MC9300 [16]

Převládajícími operačními systémy mobilních terminálů byly nejdříve systémy od Microsoftu (Windows CE, Windows Mobile, Windows Phone). Od roku 2018 však Microsoft postupně ukončoval podporu těchto systémů a jeho pozici zde převzal operační systém Android [17].

Android systém instalovaný na podnikových zařízeních může, ale nemusí obsahovat Google Mobile Services (GMS), tj. skupinu aplikací od Googlu, které jsou předinstalované na většině spotřebních Android zařízeních. Jde například o aplikace jako Google Maps či Google Play. Tyto aplikace nejsou součástí Android Open Source Project (AOSP) a nejsou open source. Pro

jejich získání je nutné nejdříve získat certifikaci zařízení a dále také samotnou licenci [18].[2] K nepoužívání GMS mohou vést např. bezpečnostní důvody, nevýhodou však je, že u zařízení bez GMS není podporován Android Enterprise, tj. Googlem poskytované Android řešení pro správu podnikové mobility.

Velmi důležitá je délka podpory systému. Standardní doba bezpečnostní podpory, kterou poskytuje Google pro danou verzi Androidu, je 3 roky. Například v případě řešení Zebra LifeGuard může však tato doba být prodloužena až na 10 let od uvedení zařízení na trh [19].

Výrobci pro svá zařízení dále poskytují množství užitečného softwaru. Například nástroj StageNow od Zebra Technologies umožňuje vygenerovat 1D nebo 2D kód, který stačí zařízením pouze naskenovat, a veškeré konfigurace (instalace aplikací, nastavení Wi-Fi, uspávání terminálu, nastavení skeneru a další funkce) se automaticky nastaví. Enterprise Home Screen umožňuje nastavení domovské obrazovky tak, aby obsahovala pouze aplikace určené pro práci [19]. Nástroje EMDK (Enterprise Mobility Development Kit) umožňují naplno využít možností terminálu pro vytváření podnikových aplikací. Velmi užitečná je také aplikace DataWedge, která umí číst data ze čtečky čárových kódů nebo ze čtečky RFID.

Uvedená zařízení mohou vzájemně (a také s WMS serverem) komunikovat bezdrátově např. pomocí Bluetooth nebo Wi-Fi. Mezi významné výrobce patří Zebra Technologies, dále Honeywell a Datalogic.

[2] Android je ve skutečnosti ochranná známka společnosti Google a systémy, které nejsou certifikované, ji nesmějí používat [18].

4 Logování

Tato kapitola pojednává o tom, co je to log, jaký má log význam, jaký mají logové záznamy formát a jaké jsou činnosti související s vytvářením logů.

4.1 Log

Log je záznam o činnosti počítačového programu či soubor s takovými záznamy. Může uchovávat informace o běhu programu a zpracovaných datech. Logy jsou tedy užitečné pro zpětné dohledání informací a jejich analýzou je možné obvykle i zjistit, v jakém místě programu došlo k chybě a co chybě předcházelo. Logování (tj. vytváření logů) tímto způsobem výrazně usnadňuje opravy chyb, ladění programu, další vývoj programu či vizualizaci a zpracování různých statistických informací.

Logování lze nastavit tak, aby probíhalo například do souboru, proudu, socketu, databáze či do logovacího systému. Různé typy událostí mohou být ukládány na různá místa – například lze do jednoho souboru ukládat přístupové logy a do jiného souboru chybové logy.

4.2 Logované události

Logovanou informací může být obecně cokoli – záleží na povaze daného programu a na úvaze programátora. Nejčastěji se jedná o nastalé chyby, odchycené výjimky, důležité orientační body v programu, informace o tom, kdo a jak program využíval, vstupy programu či síťovou komunikaci. Důležité také mohou být informace o stavu zařízení či jeho periferií.

Záznamy by vždy měly mít jasnou vypovídající hodnotu a neměly by být duplikovány, neboť jejich množství může velmi rychle narůstat a orientace v nich ztrácet na přehlednosti. Rozhodování o tom, jaké informace zalogovat, může někdy usnadňovat politika logování (logging policy) nastavená vývojářským týmem či zadavatelem projektu [20].

4.3 Formát logu

Jelikož logování produkuje velké množství informací, vzniká přirozeně potřeba stanovit formát záznamů. A to nejen za účelem dosažení přehlednosti při čtení lidmi, ale především za účelem jejich dalšího strojového zpracování.

Záznamy mohou mít podobu prostého textu (například soubory s příponou `.log`) s jedním záznamem na jeden řádek. Pro příklad je ve výpisu 4.1 uveden log ve formátu CLF (Common Log Format), jenž je používán webovými servery pro ukládání přístupových logů.

```
127.0.0.1 - - [17/Oct/2022:10:24:11 -0700]
    "GET /foo.gif HTTP/1.0" 200 200
```

Výpis kódu 4.1: Příklad CLF logu

Zázam představuje jeden HTTP požadavek na server a obsahuje IP adresu klienta, časovou značku požadavku, použitou metodu HTTP, požadovaný zdroj, stavový kód a počet bajtů odpovědi. Obdobným formátem je ELF (Extended Log Format), který je navíc přizpůsobitelný [21].

Jednoduššího strojové zpracování logů může být dosaženo použitím strukturovaného logování. Strukturované logy místo prostého textu používají některý z jednodušeji parsovatelných formátů, nejčastěji JSON či XML. Takový přístup pak umožňuje vytvářet konceptuální modely logů například pomocí UML [22].

4.3.1 Úroveň logu

Úroveň logu je informace obsažená v logu, která určuje jeho závažnost (prioritu). Na základě této informace pak lze logy filtrovat a také lze v logovacím frameworku nastavovat úrovně logů, které se mají generovat. Ačkoli je možné zavést si úrovně vlastní, obvykle se lze setkat s následujícími pěti [23]:

- INFO: Zprávy o normálním běhu a stavech aplikace
- DEBUG: Zprávy, které mouhou pomáhat vývojářům při ladění
- WARNING: Zprávy o potenciálně škodlivých událostech
- ERROR: Zprávy o chybách, které ale nemusí znamenat pád aplikace
- FATAL: Zprávy o závažných situacích, které způsobí zastavení aplikace

4.4 Správa logů

S vytvářením logů bývá spojena řada dalších činností. Soubory, které už nebudou dále doplňovány, mohou být archivovány. Dále se pro šetření místa na disku používá rotace logů – novější logy nahrazují logy starší, které jsou mazány či přesouvány do jiného úložiště [20].

Další důležitou činností u složitějších systémů může být sběr logů z různých softwarů či zařízení na jednom centrálním místě. Pro odesílání jednotlivých záznamů lze využít protokol Syslog, který je standardizován RFC 5424 [24]. Pokud jsou záznamy ukládány do souborů, lze tyto soubory monitorovat a nové záznamy odesílat například pomocí nástroje Filebeat (viz sekce 5.6.3). Jinou možností je soubory daného programu odesílat pouze tehdy, když o to uživatel či jiný program zažádá.

V případě nestrukturovaného logování může mít význam použití některé z technik automatické abstrakce logů (automated log abstraction techniques – ALAT) Tyto techniky převádějí nestrukturované logy v prostém textu na seznamy strukturovaných událostí, což umožňuje jejich další analýzu. Vstupní logové záznamy jsou složeny ze statických částí, které jsou přímo zapsány ve zdrojovém kódu programu, a dynamických částí, které jsou vyhodnoceny za běhu programu. Cílem technik ALAT je oddělení statických částí od dynamických, zamaskování dynamických částí a určení typů událostí, které se v záznamech vyskytují. [25].

Analýza logů může nalézt chyby, bezpečnostní rizika či neobvyklé situace. Následnými činnostmi může být ladění aplikace, vizualizace logů, vytváření reportů a rozhodování.

S příchodem cloudu se objevil nový model služby Logging as a Service (LaaS), v němž poskytovatel této služby zajišťuje příjem a správu logů v cloudovém prostředí. [26]. Výhodou je, že odpadá správa hardwaru i softwaru, nevýhodou pak mohou být otázky týkající se bezpečnosti (dané např. i odlišnou legislativou, pokud je poskytovatel cloudu z jiné země), které ale lze zmírnit zavedením hybridního cloudu.

4.5 Logování a bezpečnost

Jelikož při vytváření logů dochází k ukládání velkého množství informací, je třeba brát v úvahu bezpečnostní aspekty. Při logování by především mělo být omezeno ukládání tajných a osobních údajů. Co je považováno za osobní údaj se může v různých zemích lišit a je třeba řídit se vždy platnou legislativou.

Ani k logům, které neobsahují tajné informace, by neměl být povolen neautorizovaný přístup. Potenciální útočník by totiž například mohl změnit záznamy tak, aby zakryl svou škodlivou aktivitu. Falešné pozměňování logů se nazývá log injection a útočník jej může dosáhnout například zadáváním nevalidních vstupů do aplikace. Tímto způsobem lze dokonce vložit i škodlivý kód, který by později mohl být vykonán [27]. Při odesílání logů může být také žádoucí jejich zašifrování.

Stejně jako jsou informace, které nesmí být logovány, mohou být i informace, které logovány být musí. Například dle Zákona o zpracování osobních údajů je při automatizovaném zpracování osobních údajů povinnost „*pořizovat elektronické záznamy, které umožní určit a ověřit, kdy, kým a z jakého důvodu byly osobní údaje zaznamenány nebo jinak zpracovány*“ [28].

Řízení logování je možné pomocí některého z logovacích frameworků, jejichž příklady budou uvedeny v sekci 5.4. Sběr, správa a vizualizace pak budou více popsány v sekcích 5.5 a 5.6.

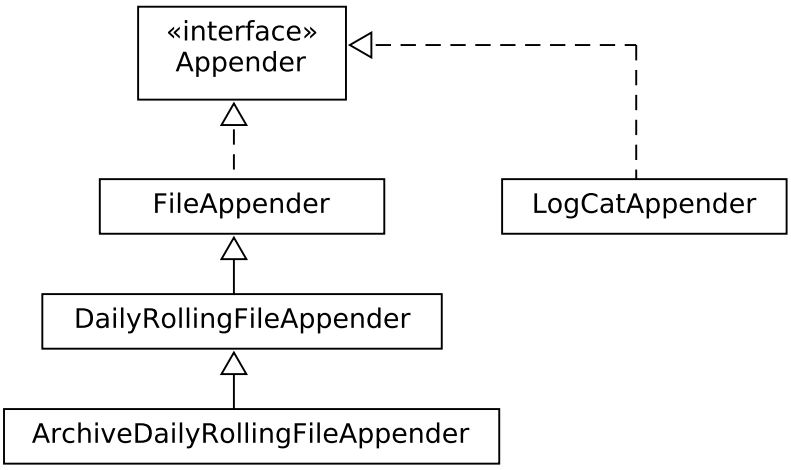
5 Analýza úlohy

Tato kapitola začíná popisem stávajícího stavu logování v aplikaci Aimtec Touch Client a rozebráním požadavků kladených na řešení úlohy. Následuje popis existující služby LogCollector, jež bude při řešení využita. Zbytek kapitoly se pak věnuje technologiím souvisejícím s řešeným problémem.

5.1 Logování v aplikaci Aimtec Touch Client

Logování aplikace je zajišťováno frameworkem Log4j, který bude více popsán v části 5.4.1. Jeho konfiguraci zajišťuje knihovní třída `Log4jConfigurator` s jedinou veřejnou metodou `setUp`, jež je volána při spuštění aplikace.

Metoda `setUp` konfiguruje kořenový `Logger` a přidává mu dvě implementace rozhraní `Appender` – jednu pro zápis do logcatu (nástroj pro zobrazování logů Android aplikací) a druhou pro zápis do souboru. Implementace pro zápis do logcatu je instancí třídy `LogCatAppender`[1]. Implementace pro zápis do souboru je instancí třídy `ArchiveDailyRollingFileAppender`. Ta rozšiřuje třídu `DailyRollingFileAppender` z Log4j zajišťující denní rotaci souborů s logy a přidává možnost jejich archivace – obrázek 5.1.



Obrázek 5.1: Implementace rozhraní `Appender`

Logované události jsou všeho druhu (např. informace o požadavcích na server, změnách nastavení aplikace apod.) a všechny záznamy jsou společně

[1] z knihovny android-logging-log4j

ukládány do aktivního logovacího souboru. K definici formátu je použita třída `PatternLayout`. Každý záznam začíná na nové řádce a obsahuje časovou značku, úroveň logu, název vlákna, loggeru a metody, v níž došlo k zalogování, a nakonec samotnou zprávu. Názvy delší než stanovený počet znaků jsou oříznuty, názvy kratší jsou naopak doplněny mezerami. Příklad logů informujících o práci s EMDK lze vidět ve výpisu 5.1.

```
2022-03-07 06:12:59,007 INFO  [        main] nning.
   DciEmdkManager.          <init> - EMDK library is
   installed on this device
2022-03-07 06:12:59,042 INFO  [        main] nning.
   DciEmdkManager.   loadProfiles - EMDKManager created
```

Výpis kódu 5.1: Příklad logů aplikace

Správa logů je přizpůsobena možnostem konkrétního zařízení. Pokud zařízení obsahuje externí úložiště (např. SD karta) a aplikace má oprávnění na něj zapisovat, ukládají se soubory s logy na něj, v opačném případě se ukládají do interního úložiště zařízení. Každý soubor představuje jeden den logování. Maximální počet uchovávaných souborů s logy (tedy maximální počet uchovávaných dnů) před tím, než začnou být nahrazovány novějšími, je odvozen od velikosti dostupného úložného prostoru dle tabulky 5.1.

Dostupný úložný prostor (MiB)	Počet souborů s logy
>= 100	5
>= 500	10
>= 1024	30
>= 5120	60

Tabulka 5.1: Maximální počet souborů s logy v zařízení

Soubory jsou ukládány do adresáře `logs`. Název aktivního logovacího souboru je vždy `TouchClient.log`. Po skončení dne je k názvu připojen datum daného dne a soubor je zazipován. Jak může vypadat struktura adresáře `logs` je vidět na obrázku 5.2 .

```
logs
├── TouchClient.log
├── TouchClient.log.2023-01-01.zip
├── TouchClient.log.2023-01-02.zip
├── TouchClient.log.2023-01-03.zip
├── TouchClient.log.2023-01-04.zip
└── TouchClient.log.2023-01-05.zip
```

Obrázek 5.2: Příklad adresáře s logy v zařízení

5.2 Požadavky na řešení

Jak již bylo popsáno v předchozi sekci, aplikace Aimtec Touch Client vytváří logové soubory, které jsou uchovávány v úložišti daného Android zařízení. Pro získání těchto souborů programátory aplikace (např. za účelem odhalení příčiny chyby) je nutné nejdříve kontaktovat ICT oddělení zákazníka. Pracovník z tohoto oddělení pak musí obvykle získat dané zařízení od pracovníka skladu, připojit jej kabelem k počítači, nalézt požadované soubory v souborovém systému a poskytnout je programátorům.

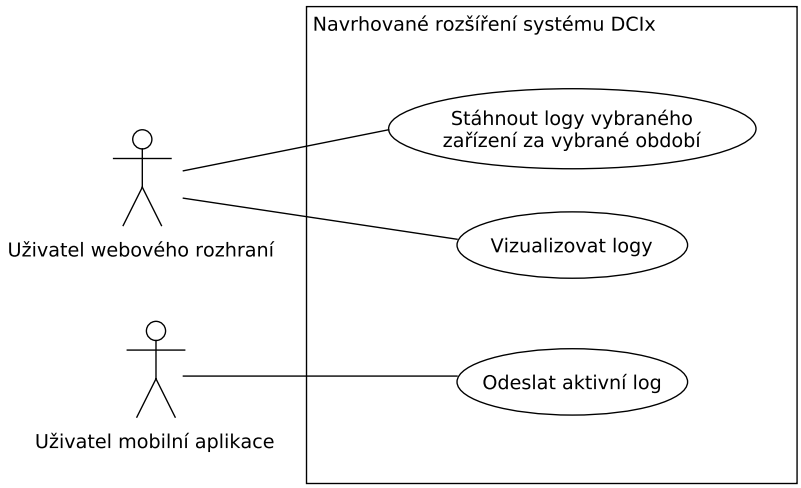
Navrhované řešení by mělo usnadnit proces sběru a odesílání logů ze zařízení tak, aby nebyl nutný zásah oddělení ICT a logy bylo možné dostat na sběrné místo přímo uživatelem systému DCIx bez nutnosti fyzického přístupu k zařízení a také bez větších technických znalostí.

Požadavky na řešení byly diskutovány se zadavatelem projektu. Po jejich analýze lze definovat základní případy užití znázorněné na obrázku 5.3. Hlavní případ užití bude realizován uživatelem webového rozhraní DCIx. V něm bude možnost výběru zařízení, z nějž mají být získány logy. Po výběru časového rozmezí a odeslání požadavku dojde k odeslání požadovaných logů ze zařízení na sběrné místo. Uživatel bude informován o úspěšnosti operace a způsobu, jakým lze logy zobrazit.

Vizualizaci logů lze vyčlenit jako samostatný případ užití. Logy budou po nějakou dobu uchovávány na sběrném místě a uživatel webového rozhraní bude mít možnost zobrazení všech nahraných logů.

Další případ užití bude realizován uživatelem aplikace Aimtec Touch Client přímo na mobilním zařízení. Zde bude možnost snadno odeslat aktivní (tj. právě zapisovaný) logový soubor na sběrné místo. Tato možnost bude užitečná v případě, kdy bude potřeba provést rychle analýzu posledních událostí. Po odeslání bude uživatel informován o úspěšnosti operace.

Volitelně lze ve webovém rozhraní implementovat možnost pro nastavení stahování logů v pravidelných časových intervalech nebo možnost pro stahování pouze logů od zvolené úrovně výše.



Obrázek 5.3: Základní případy užití navrhovaného řešení

Možnost pro stažení (resp. odeslání) logů by měl mít pouze uživatel přihlášený do systému DCIx. Na každý požadavek zaslaný po síti by měla přijít odpovídající odpověď. V případě, že nebude zařízení připojeno k síti, musí o tom být uživatel webového rozhraní informován. Také je třeba myslet na to, že komunikující zařízení mohou používat různá nastavení času. Při zadávání časového intervalu bude uživatel webového rozhraní zadávat hodnoty v jeho lokálním čase, které pak musí být správně porovnány s časovými značkami logů v zařízení.

Vytvořené řešení by dále mělo šetřit množství přenášených a uchovávaných dat. Přenášené soubory by měly být komprimované v archivu. Při odeslání stejných souborů vícekrát nesmí docházet k jejich duplikaci na sběrném místě. Aby nedocházelo k zaplňování sběrného místa, budou logy uchovávány pouze po určitou dobu, po jejímž uplynutí dojde k jejich automatickému smazání.

Ačkoli řešení bude zatím použito pro jednu Android aplikaci, měla by jeho serverová část být dostatečně obecná i pro komunikaci se zařízením odlišného operačního systému.

5.3 DCIx služby

Server produktu DCIx existoval až do verze 7 jako jeden velký monolit obsahující veškerou funkčnost. Tento monolit byl rozdělen při přechodu na verzi 7 do menších celků – služeb, které běží v Kubernetes clusteru v cloudu nebo přímo u zákazníka (on-premises). Tyto služby spolu komunikují přes API, ale jsou navzájem nezávislé.

Pro řešenou úlohu mají význam především následující služby:

- DciCore
- API Gateway
- Authorizer
- Log Collector

Služba DciCore je hlavní a nejrozsáhlejší částí celého systému. Tato služba mimo jiné vytváří uživatelské webové rozhraní[2], které bude potřeba rozšířit o možnosti pro sběr logů.

Služba API Gateway poskytuje jediný veřejný vstupní bod k serveru DCIx. Nachází se mezi klienty a dalšími službami a řídí provoz API volání. Bude-li přidáno nové veřejné API, bude třeba přidat jeho definici do služby API Gateway.

Služby mohou vyžadovat autentizaci pomocí tokenu, který lze získat přihlášením na službu Authorizer. Jeho platnost pak při přístupu k dalším službám kontroluje služba API Gateway.

Z dalších služeb má pro tuto práci význam především služba Log Collector. Ta slouží k centrálnímu sběru logů ze všech ostatních služeb. Využívá k tomu nástroj Filebeat, který monitoruje standardní a chybový výstup všech DCIx kontejnerů a také logovací soubory ve specifikovaném adresáři. Dále využívá Elasticsearch jako databázi logů a Kibanu pro vizualizaci logů. Bližší popis těchto nástrojů je v části 5.6.3. Jelikož tato služba již shromažďuje logy ze všech částí serveru DCIx, bude zřejmě vhodným místem i pro shromažďování logů z klientských mobilních zařízení.

5.4 Logovací frameworky

Logovací frameworky usnadňují logování tím, že umožňují jeho snadnou konfiguraci, parametrizaci výstupů, logování v různých logovacích úrovních apod. V následujícím textu budou přiblíženy některé z nejvýznamnějších frameworků pro jazyk Java (a platformu Android). K analýze úlohy jsou podstatné jejich možnosti vzdáleného logování.

[2]Ve firmě však již běží projekt na oddělení webového rozhraní do samostatné služby.

5.4.1 Log4j

Log4j je jeden z nejpopulárnějších a také nejstarších logovacích frameworků. První release frameworku pochází z roku 1999, postupně se vyvíjel jako Log4j 1 a později byl nahrazen jeho nástupcem Log4j 2 [29].

Konfiguraci frameworku je možné provádět programově nebo pomocí konfiguračního souboru (XML, JSON, YAML či properties formát). Logování v aplikaci se provádí pomocí `Logger` objektů. Ty samy o sobě neprovádějí žádné akce, ale mají přiřazen jeden nebo více objektů `Appender`, které zpracovávají logovací události a odesílají je do cílových destinací. Výstupní formát lze přizpůsobovat pomocí objektů `Layout`. Např. `PatternLayout` umožňuje specifikovat výstupní formát podobně jako funkce `printf` v jazyce C. Dalšími objekty jsou filtry `Filter`, které umožňují ignorování některých události [30].

Pro vzdálené logování lze vytvořit vlastní nebo využít již existující `Appender`. Například `HttpAppender` umožňuje posílání logů pomocí HTTP protokolu. `SMTPAppender` posílá e-mail při výskytu nějaké události, typicky chyby. `SocketAppender` umožňuje zapisovat do socketu specifikovaného adresou a portem prostřednictvím TCP nebo UDP protokolu. `SyslogAppender` slouží pro využití protokolu Syslog. Dále jsou k dispozici různé `Appendery` pro logování do databáze. Zabezpečení komunikace je u některých `Appenderů` možné pomocí třídy `SslConfiguration` [31].

5.4.2 Logback

API pro Log4j je pro zajištění dopředné kompatibility oddělené od jeho implementace. To znamená, že Log4j API je logovací fasáda, která může být použita s Log4j implementací, ale může také stát před jinými logovacími implementacemi, jako je např. Logback [32].

Logback je v mnohém podobný Log4j a údajně by měl být až desetkrát rychlejší než Log4j 1 [33]. To už však nemusí platit v případě Log4j 2. Oproti Log4j 2 je Logback odlehčenější a nemá tolik možností rozšíření.

Vzhledem k podobnosti obou frameworků, jsou možnosti vzdáleného logování pro Logback podobné jako pro Log4j.

5.4.3 SLF4J

SLF4J (Simple Logging Facade for Java) není ani tak framework, jako spíše realizace návrhové vzoru fasáda. Používání logovací fasády může být důležité při vytváření knihovny, protože nevynucuje konkrétní implementaci logování. Implementaci SLF4J poskytuje např. Log4J i Logback.

5.4.4 java.util.logging

Jde o balík, který je součástí JDK a obsahuje třídy sloužící k základnímu logování. Pro vzdálené logování pomocí socket lze použít třídu `SocketHandler`. Pro malé projekty je výhodou, že tento balík je přímo v JDK. Poskytuje však pouze základní výbavu a pro náročnější projekty je lepší použít některý z frameworků.

5.4.5 android.util.Log

Třída poskytující jednoduché rozhraní pro logování na platformě Android. K dispozici je několik metod pro zalogování událostí různých logovacích úrovní. Zalogované události pak lze prohlížet nástrojem logcat.

5.5 Technologie pro přenos logů

V této části jsou uvedené technologie, které lze využít pro přenos logů z klientských zařízení na server.

5.5.1 Vzdálené logování

Vzdálené logování znamená odesílání logových záznamů počítačovou sítí na sběrné místo hned po výskytu logových událostí. Není tedy nutné uchovávání záznamů lokálně v zařízení. Možnosti vzdáleného logování poskytují logovací frameworky a některé z nich byly popsány v předchozím textu.

Nevýhodou tohoto přístupu je zatěžování sítě, které v případě většího počtu připojených zařízení může být znatelné. Důsledkem by teoreticky mohlo být i zpomalení chodu aplikace. Dále také může docházet ke ztrátě spojení a je otázka, jak se v takových situacích zachovat. Je možné začít logy ukládat do zařízení, ale po obnovení spojení bude nutné řešit jejich odeslání na server.

Aby se předcházelo zatěžování spojení, je při vzdáleném logování vhodnější logování a přenášení menšího množství dat.

Z hlediska úlohy řešené v této práci se nejedná o vhodné řešení, protože logy jsou ukládány v zařízení a potřeba jejich prohlížení není až tak častá. Je tedy vhodnější logy přenést pouze na vyžádání.

5.5.2 Sběr crashlogů

Sběr crashlogů zde znamená odesílání logů, pouze pokud dojde k pádu aplikace. Nástroj, který identifikuje pády softwaru, vytváří reporty a může odesílat logy na sběrné místo, se nazývá *crash reporter*. Crash reporters, které

jsou dostupné pro platformu Android a umožňují odeslání logů, jsou ACRA a Crashlytics.

Řešení může být užitečné v případě, kdy mají logy sloužit ke hledání chyb po pádu aplikace, ale nehodí se pro úlohu řešenou v této práci, kdy mají logy být dostupné kdykoli, nikoli jen při pádu aplikace.

5.5.3 REST API

REST (Representational State Transfer) API je typ rozhraní, které vyhovuje podmínkám architektonického stylu REST, mezi než patří např. bezstavová komunikace nebo architektura typu klient–server.

REST API lze používat k vytváření webových služeb. Pro komunikaci se využívá protokol HTTP a data bývají přenášena nejčastěji ve formátu JSON nebo XML.

Pro účely sběru logů lze implementovat REST API služby, která bude přijímat logovací soubory a ukládat je na svůj souborový systém. Na dané API pak bude možné po zažádání odesílat logy z mobilního zařízení.

5.5.4 WebSocket

WebSocket je komunikační protokol umožňující obousměrnou komunikaci nad protokolem TCP. Navázání spojení (handshake) se provádí pomocí HTTP požadavku s hlavičkou Upgrade. Jelikož se provádí pouze jeden HTTP požadavek a zbytek komunikace je čistě TCP, jedná se o odlehčený protokol, který by neměl zbytečně zatěžovat síť a zařízení.

Alternativou k WebSocket mohou být např. Server-Sent Events (klient může odebírat události generované serverem) nebo long polling (dlouhé dotazování na server ze strany klienta) [34].

Technologii WebSocket lze využít pro zasílání požadavků na nahrání logů ze serveru do mobilního zařízení. Po přijetí požadavku pak zařízení může odeslat logy např. na REST API.

5.6 Další související technologie

Následující technologie úzce souvisí s řešenou úlohou, jde o frameworky pro webové aplikace, monitorování a vizualizaci logů.

5.6.1 Spring Boot

Spring Boot je framework pro tvorbu webových aplikací v Javě. Nabízí mnohé možnosti, které ale nejsou předmětem této práce. Z hlediska řešené úlohy je však důležitá anotace @Scheduled.

Anotace @Scheduled

Tato anotace umožňuje naplánovat provádění úloh, resp. anotovaných metod. Způsob provádění se určuje nastavením hodnoty jednoho ze tří atributů [35]:

- `fixedDelay`: Vykonávání anotované metody s danou dobou mezi koncem posledního volání metody a začátkem dalšho volání
- `fixedRate`: Vykonávání anotované metody s danou dobou mezi jejími jednotlivými voláními
- `cron`: Vykonávání anotované metody na základě cron výrazu

U prvních dvou způsobů lze také nastavit `initialDelay`, tedy zpoždění před prvním voláním metody. Pokud není nastaveno, provede se první volání okamžitě po startu aplikace.

Cron výraz je výraz umožňující specifikovat přesné časy (sekundy, minuty, hodiny, dny v měsíci, měsíce a dny v týdnu), v nichž se má úloha opakovaně spouštět. Oproti předchozím způsobům nabízí tento větší flexibilitu. Časy spouštění úloh jsou zde navíc přesně dány a neodvíjí se od času spuštění aplikace.

5.6.2 Struts

Struts je starší framework pro tvorbu webových aplikací v Javě. Dodržuje vzor MVC pomocí tzv. akcí (tříd dědících od `Action`), formů (tříd dědících od `ActionForm`) a JSP souborů představujících View. Tento framework používá služba DciCore.

5.6.3 ELK Stack

ELK Stack je sada nástrojů Elasticsearch, Logstash a Kibana, které se používají pro práci s logy. Logstash přijímá a zpracovává data od různých instancí a ukládá je do databáze Elasticsearch. Kibana je pak nástroj pro jejich vizualizaci.

Data do Logstash je možné dodávat nástrojem Filebeat. Pro jeho konfiguraci se používají konfigurační soubory, v nichž jsou definované tzv. *prospektory* neboli vstupy.

6 Návrh řešení

V této kapitole je uveden celkový pohled na řešení bez implementačních záležitostí. Popis implementace lze pak pro každou sekci najít v kapitole 7.

6.1 REST API

Pro přenos logů z mobilních zařízení na server byla vybrána technologie REST API. Možnost vzdáleného logování byla zavržena vzhledem k možnému zbytečnému zatěžování sítě a úložiště na serveru. Mobilních zařízení mohou být běžně desítky a potřeba prohlížet logy není až tak častá, je tudíž zbytečné přenášet na server všechny logy ze všech zařízení. Pro řešenou úlohu bude tedy vhodnější přenášet požadované logy pouze po vyžádání uživatelem. Možnost sběru crash logů je zajímavá, ale umožňuje získat logy pouze v případě pádu aplikace, což není pro řešenou úlohu dostačující. U obou předchozích možností by navíc v případě smazání logu ze serveru neexistoval způsob, kterým by zařízení mohlo jednoduše log odeslat znovu. Řešení pomocí REST API umožní zařízení zaslat vybrané logy kdykoli to bude vyžadováno. REST API je navíc standard, který používají i ostatní služby serveru DCIx. Navržené rozhraní bude popsáno pomocí OpenAPI specifikace.

6.2 Spring Boot aplikace

K implementaci API bude použita Spring Boot aplikace, která bude nasazována jako součást služby Log Collector. Přijaté logy budou ukládány na souborový systém služby a jejich automatické mazání bude zajišťeno pomocí možností Spring frameworku (konkrétně anotace @Scheduled).

6.3 Odeslání aktivního logu

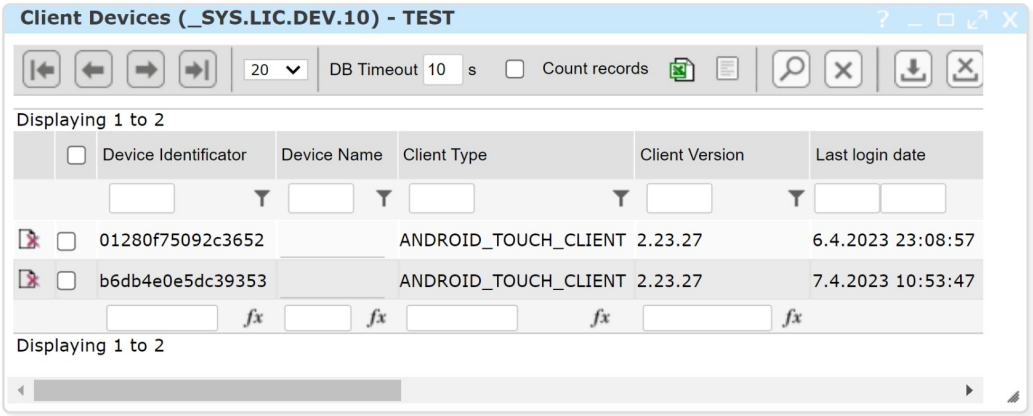
Odeslání aktivního logu z Android aplikace bude realizováno přidáním nového tlačítka do nastavení, které bude vidět jen po přihlášení uživatele. Po jeho stisku bude v souborovém systému nalezen aktivní logovací soubor a ten bude odeslán na službu Log Collector. O úspěšnosti operace bude uživatel informován Android Toastem.

6.4 WebSocket spojení

Z webového rozhraní DCIx bude potřeba zasílat do mobilního zařízení požadavky pro odeslání logů na službu Log Collector. Požadavky budou muset obsahovat minimálně časové rozmezí požadovaných logů. K jejich zasílání bude použita technologie WebSocket, která se již používá jak na serveru, tak v mobilní aplikaci, momentálně však k jiným účelům. Bude proto přidán nový typ spojení a refaktorován dosavadní kód tak, aby dokázal pracovat s více typy WebSocket spojení. Nové spojení bude aktivní po celou dobu přihlášení uživatele do aplikace. Na serveru i v aplikaci bude indikátor stavu spojení. V případě ztráty spojení bude aplikace zkoušet jeho opětovné navázání. Po několika neúspěšných pokusech zůstane spojení uzavřeno, ale bude možné ručně zažádat o nové pokusy o připojení.

6.5 Vyžádání logů uživatelem DCIx

Vyžádání logů bude možné přes obrazovku se zařízeními (obrázek 6.1), která se již v DCIx nachází. Ta bude pomocí nové Struts akce rozšířena o možnost výběru časového rozmezí a odeslání požadavku do daného zařízení.



Obrázek 6.1: Obrazovka se zařízeními v DCIx

Jakmile zařízení přijme požadavek, nalezne požadované soubory ve svém úložišti, odešle je v jednom zazipovaném souboru na službu Log Collector a pošle zpět na DciCore pomocí WebSocket odpověď o úspěšnosti operace. Jelikož jde o asynchronní komunikaci, je třeba vyřešit, jak uživateli zobrazovat přijaté odpovědi. Ty lze na DciCore po přijetí ukládat do bufferu a ze strany uživatele se pak dotazovat na jejich existenci. Dotazování by bylo možné řešit pomocí JavaScript časovače, pro jednoduchost však postačí tlačítko, které bude reagovat na stisk uživatele. Aby bylo možné rozeznat, ke kterému

požadavku patří která odpověď, budou i informace o požadavcích ukládány a bude jim přiřazen číselný identifikátor. Číslo požadavku pak bude součástí zasílaných zpráv mezi DciCore a mobilním zařízením.

6.6 Vizualizace logů

Detekce a vizualizace logů budou zajištěny podobně jako je tomu u logů ze služeb serveru. Nástroj Filebeat bude na službě Log Collector monitorovat adresář určený pro logy ze zařízení, přidá jim vyhrazený tag a logy poté bude možné prohlížet v Kibaně. Odkaz do Kibany sloužící k nalezení požadovaných logů bude součástí odpovědi zobrazované uživateli webového rozhraní.

7 Implementace

Tato kapitola odpovídá svou strukturou kapitole předchozí a popisuje implementaci navrženého řešení. Pořadí jednotlivých částí také odpovídá tomu, jak bylo řešení skutečně realizováno – tedy od návrhu API a služby pro příjem logů přes tlačítko pro odeslání aktivního logu, založení WebSocket spojení a zaslání požadavku do zařízení až po vizualizaci odeslaných logů.

7.1 REST API

Vytvářené API pro nahrání logů bude mít jediný endpoint s POST metodou nacházející se na relativní cestě `/uploadLogs`. Tělo požadavku bude mít dvě vyžadované části: `uploadLogsRequest` popisující požadavek a `logs` představující samotný archiv (resp. soubor) s logy – viz výpis 7.1 ve formátu YAML.

```yaml
requestBody:
  required: true
  content:
    multipart/form-data:
      schema:
        type: object
        required:
          - uploadLogsRequest
          - logs
        properties:
          uploadLogsRequest:
            $ref: '#/components/schemas/UploadLogsRequest'
          logs:
            type: string
            format: binary
      encoding:
        logs:
          contentType: application/zip, application/octet-stream
```

Výpis kódu 7.1: Tělo požadavku na endpoint `/uploadLogs`

`UploadLogsRequest` je JSON objekt s 5 atributy:

- `files`: Pole objektů `Log`, z nichž každý popisuje jeden logovací soubor
- `isZipped`: Zda je přenášený soubor archivem

- `clientType`: Jméno klientské aplikace (např. `ANDROID_TOUCH_CLIENT`)
- `deviceName`: Jméno zařízení (nepovinné)
- `deviceIdentifier`: Unikátní identifikátor zařízení

Každý objekt `Log` v poli `files` obsahuje název jednoho souboru v archivu a datum logovacího dne. Pokud přenášený soubor nebude archivem, pak musí pole `files` obsahovat právě jeden `Log` popisující právě tento soubor.

Příklad JSON objektu `UploadLogsRequest` popisujícího požadavek pro nahrání 2 logovacích souborů v jednom archivu je uveden ve výpisu 7.2. Nepovinný atribut se jménem zařízení zde není vyplněn.

```
{
  "files": [
    {
      "name": "TouchClient.log.2023-01-05",
      "date": "2023-01-05T00:00:00+02:00"
    },
    {
      "name": "TouchClient.log",
      "date": "2023-01-06T00:00:00+02:00"
    }
  ],
  "isZipped": true,
  "deviceIdentifier": "b6db4e0e5dc39353",
  "clientType": "ANDROID_TOUCH_CLIENT"
}
```

Výpis kódu 7.2: `UploadLogsRequest` objekt

Jako odpověď je zasílán JSON objekt `UploadLogsResponse`, jenž obsahuje pole chyb `Error`, které nastaly.

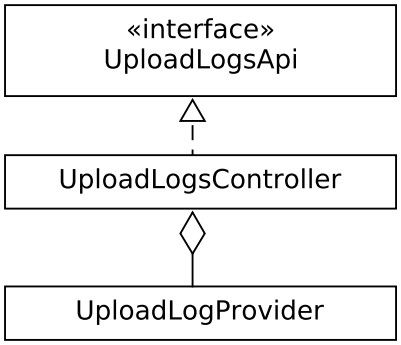
7.2 Spring Boot aplikace

Služba Log Collector je vytvořena dle standardní šablony používané pro služby DCIx a skládá se ze 3 modulů:

- `log-collector-client`: Klient, kterého mohou používat ostatní služby
- `log-collector-model`: Datový model využívaný klientem i serverem
- `log-collector-server`: Serverová část představovaná Spring Boot aplikací

K vytvoření controlleru navrženého API je možné využít Maven plugin OpenAPI Generator. Tento plugin dokáže z API definice vygenerovat Java rozhraní, které pak lze implementovat. Dle schémat uvedených v definici API vygeneruje také datový model – třídy `UploadLogsRequest`, `Log`, `Error` a `UploadLogsResponse`. Díky konfiguaci, která se nachází v předkovi Maven projektu, se soubory generují na správné místo (tj. datový model do datového modelu a controller do serverové části).

Vygenerované rozhraní controlleru `UploadLogsApi` implementuje třída `UploadLogsController`, která pouze vrací HTTP stavový kód 200 a `UploadLogsResponse`. Veškerá další logika pro příjem logů je prováděna poskytovatelem služby `UploadLogProvider` – viz obrázek 7.1.



Obrázek 7.1: Diagram tříd služby

V obslužné metodě poskytovatele je nejdříve složena cesta k adresáři, do něhož budou uloženy přijaté logy. Ta je složena z adresáře `logs`, do něhož jsou ukládány všechny logovací soubory shromažďované a produkované službou Log Collector, dále adresáře `clients` a poté adresářů odpovídajících jménu klientské aplikace, zařízení a identifikátoru zařízení. Příklad možné struktury je na obrázku 7.2. Tato struktura odpovídá požadavku, jenž byl popsán ve výpisu 7.2. Jelikož v popisu požadavku nebyl uveden název zařízení, použil se místo něj výchozí řetězec „UNNAMED_DEVICE“.

Po složení cesty je rozbalen přijatý archiv a všechny soubory jsou pojmenovány dle jejich logovacího data a uloženy na souborový systém serveru. Rozbalení archivu je nutné z důvodu dalšího zpracování logů nástrojem Filebeat. Jestliže přijatý soubor nebyl archivem, bude jen uložen pod odpovídajícím názvem.

```
logs
└── clients
    └── ANDROID_TOUCH_CLIENT
        └── UNNAMED_DEVICE_b6db4e0e5dc39353
            ├── 2023-01-05.log
            └── 2023-01-06.log
```

Obrázek 7.2: Příklad adresáře s logy na službě Log Collector

7.2.1 Automatický úklid logů

Automatický úklid logů zajišťuje třída `MaintenanceScheduler` s využitím Spring anotace `@Scheduled`. Tato anotace plánuje provádění anotované metody pomocí cron výrazu (viz 5.6.1). Použitý výraz je uložen jako application property v souboru `application.properties` a jeho podoba je

$$0\ 0/15\ *\ *\ *\ *$$

, úklid logů tedy bude probíhat na začátku každé hodiny a v rámci dané hodiny také po každých 15 minutách od jejího začátku.

Volaná metoda `recursiveDeleteFilesOlderThanLimit` vždy zkontroluje všechny soubory v daném adresáři (i jeho podadresářích, rekurzivně) a smaže všechny soubory, jejichž stáří je větší než stanovený limit, a také všechny prázdné adresáře – viz kód 7.3:

```
1  private void recursiveDeleteFilesOlderThanLimit(String dirPath) {
2      try (Stream<Path> stream = Files.list(Paths.get(dirPath))) {
3          stream.forEach(path -> {
4              if (Files.isDirectory(path)) {
5                  recursiveDeleteFilesOlderThanLimit(path.toString());
6                  deleteDirectoryIfEmpty(path);
7              } else {
8                  deleteFileIfOld(path);
9              }
10         });
11     }
12     ⋮
```

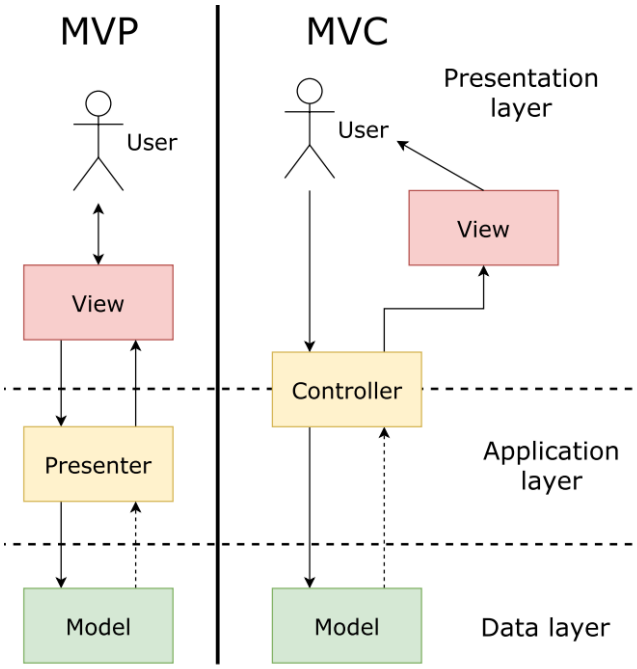
Výpis kódu 7.3: Automatický úklid logů

7.3 Odeslání aktivního logu

Nastavení aplikace Aimtec Touch Client, do nějž má být umístěno tlačítko pro odeslání aktivního logu, je spravováno pomocí Android aktivity `PreferencesActivity`. Tato aktivita je dostupná skrze vyjížděcí menu jak z přihlašovací obrazovky, tak ze seznamu transakcí po přihlášení (viz obrázky 2.2a a 2.2b). Pro objasnění toho, jak byla tato aktivita rozšířena, následuje popis vzoru MVP, na němž je aplikace založena.

7.3.1 Model–View–Presenter

Architektura aplikace je vystavěna na architektonickém vzoru Model–View–Presenter (MVP), jehož význam a implementace jsou podrobně popsány v diplomové práci původního autora aplikace [12]. Zmiňovaný vzor MVP provádí kompletní oddělení grafického uživatelského rozhraní (View) a dat (Model) skrze tzv. prezentéra (Presenter). Komunikace View a prezentéra probíhá na základě definovaného kontraktu – prezentér tedy neví, jakého typu bude jím aktualizovaný View, a podobně View neví, na jaký typ prezentéra bude směřovat akce od uživatele. Oproti vzoru MVC (Model–View–Controller), v němž se prostřední část Controller nachází na pomezí prezentační a aplikační logiky, představuje prezentér u vzoru MVP čistě aplikační logiku – viz obrázek 7.3.



Obrázek 7.3: Architektonické vzory MVC a MVP [12]

Jednotlivé Android aktivity mají svůj layout definovaný v XML souborech a mohou tedy realizovat View. Mohou se ale také skládat z fragmentů, z nichž každý realizuje samostatný View. Kontrakty mezi Views a prezentéry jsou definovány pomocí rozhraní s názvy končícími na `Contract`. Každé takové rozhraní obsahuje 2 vnořená rozhraní `Presenter` a `View`, která musí daný prezentér, resp. View, implementovat. Rozhraní `Presenter` definuje akce, na něž může View směřovat akce od uživatele, a rozhraní `View` definuje akce, kterými může prezentér aktualizovat View.

Prezentéři, kteří komunikují s externím zdrojem (serverem DCIx), dále mohou mít svůj interaktor (tj. instance tříd s názvy končícími na `Interactor`). Interaktor obvykle slouží k vytvoření a spuštění nové asynchronní úlohy `AsyncTask`, jež poběží v novém vláknu.

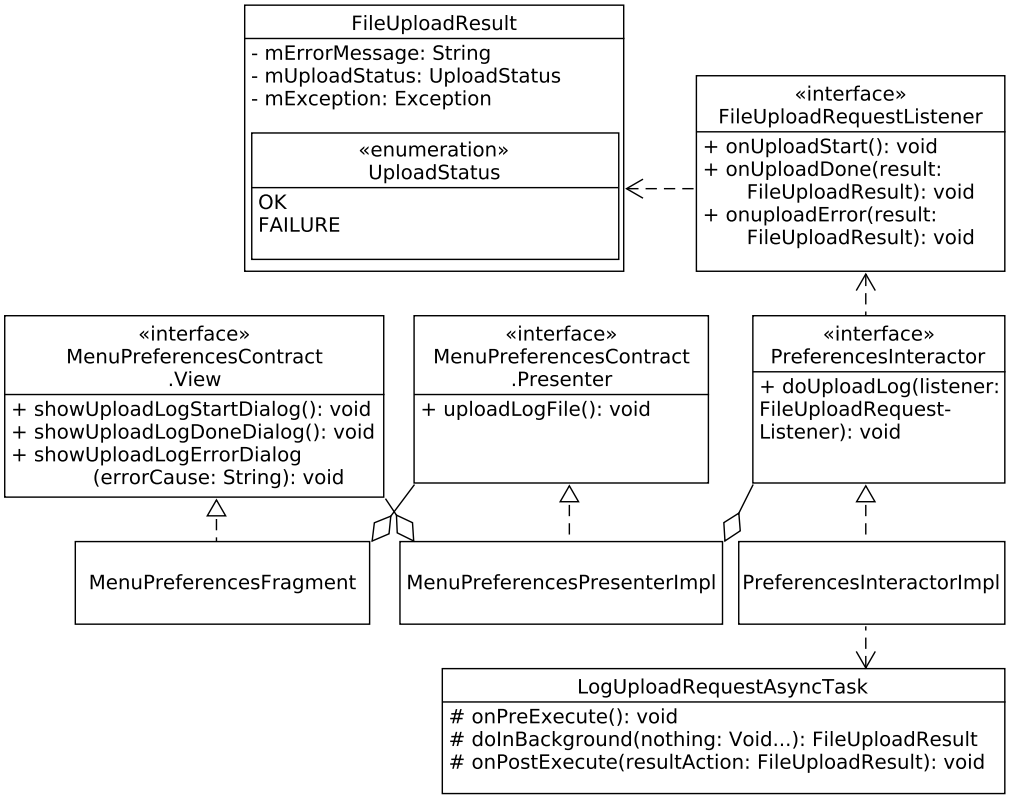
7.3.2 Vytvoření asynchronní úlohy

Aktivita `PreferencesActivity` je tvořena fragmenty rozšiřujícími společnou abstraktní třídu `PreferencesFragment` – konkrétně pro nové tlačítko byl využit fragment `MenuPreferencesFragment`. Pro vytvoření tlačítka stačí přidat nový element `Preference` do souboru s layoutem fragmentu. Dále je třeba zajistit, aby tlačítko bylo viditelné jen po přihlášení uživatele. K tomu byla doplněna metoda `setUpPreferenceAccessibility`, která je volána při vytváření fragmentu a v případě, že uživatel není přihlášen, danou `Preference` odstraní.

Fragment `MenuPreferencesFragment` je dle vzoru MVP View implementující rozhraní `MenuPreferencesContract.View` a obsahující referenci na příslušného prezentéra `MenuPreferencesContract.Presenter`. Kontrakt prezentéra byl tedy rozšířen o novou metodu `uploadLogFile`, která bude volána z View po stisku tlačítka.

Implementace prezentéra v této nové metodě využívá instanci interaktoru `PreferencesInteractor`. Ten byl rozšířen o novou metodu `doUploadLog`, jež je určena k nahrání logu na server a informování o průběhu pomocí callbacku. Pro callback bylo vytvořeno nové rozhraní `FileUploadRequestListener` s metodami `onUploadStart`, `onUploadDone` a `onuploadError`. Toto rozhraní bude možné používat jako callback pro nahrání libovolného souboru na server a podrobnosti o výsledku mu lze předávat přepravkou `FileUploadResult`. Prezentér tedy pouze vytváří implementaci zmíněných tří metod pomocí jeho View a předává řízení interaktoru.

Pro implementaci callbacku byl rozšířen kontrakt View o tři nové metody: `showUploadLogStartDialog`, `showUploadLogDoneDialog` a `showUploadLogErrorDialog`. Implementace View v těchto metodách zobrazuje odpovídající



Obrázek 7.4: Odeslání aktivního logu (1. část)

Android Toasty, díky použitému návrhu MVP lze však metody implementovat i jiným způsobem a kontrakt pro prezentéra zůstane zachován.

Implementace metody `doUploadLog` pouze vytváří a spouští novou asynchronní úlohu `LogUploadRequestAsyncTask` a předává jí instanci pro callback, jež bude použita před spuštěním a po dokončení úlohy. Popsané třídy a metody, které byly vytvořeny či doplněny, lze nalézt na obrázku 7.4

Po spuštění asynchronní úlohy je v novém vlákně spuštěna metoda `uploadActiveLog` třídy `LogManager`. Tato třída byla vytvořena za účelem správy nahrávání logů na server a bude popsána v části 7.5.1. Volaná metoda nejprve vytvoří dočasný adresář, do něhož zazipuje aktivní logovací soubor. Aby bylo možné nalézt cestu k tomuto souboru, je vždy po konfiguraci logovacího frameworku při spuštění aplikace ukládána do `SharedPreferences` úložiště. Metoda dále vytvoří žádost pro nahrání logů a předá řízení instanci `RequestExecutor`. Po návratu je odstraněn dočasný adresář a návratovou hodnotou předána přepravka `FileUploadResult` do výsledku asynchronní úlohy. Metoda `uploadActiveLog` je vidět na ukázce 7.4.

```
1  public FileUploadResult uploadActiveLog() {
2      File archivedActiveLog = zipActiveLog();
3      List<Log> logsToSend = new ArrayList<>();
4      logsToSend.add(Log.builder()
5              .name(archivedActiveLog.getName())
6              .date(OffsetDateTime.now())
7              .build()
8      );
9      FileUploadResult result = mRequestExecutor.uploadLogs(
           createUploadLogsRequest(logsToSend), archivedActiveLog);
10     removeTmpLogsDir();
11     return result;
12 }
```

Výpis kódu 7.4: Metoda `uploadActiveLog`

7.3.3 Nahrání souboru

Navržené API služby LogCollector očekává *multipart/form-data* tělo požadavku s jednou částí popisující požadavek a s druhou částí obsahující samotný soubor. Jelikož tímto způsobem lze odesílat libovolný soubor a informace o něm, bylo samotné nahrávání logů ze zařízení na server řešeno jako nahrávání libovolného souboru na server.

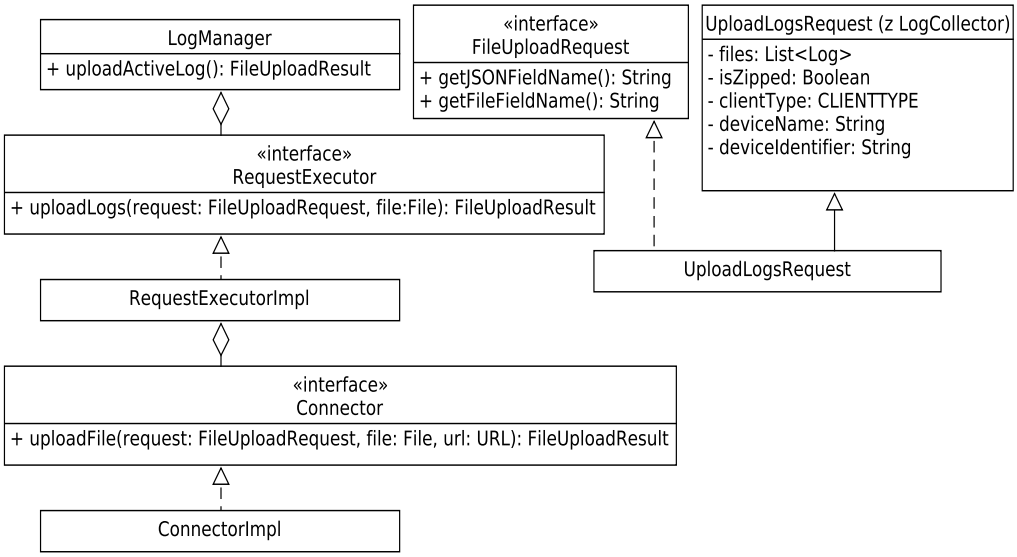
Veškeré požadavky aplikace na server jsou iniciovány pomocí instance `RequestExecutor`. Ta pomocí instance `Connector` připravuje URL a těla požadavků a následně je nechává odeslat. Pro nahrávání souborů tedy byla do rozhraní `Connector` přidána metoda `uploadFile`. Jejími parametry jsou URL, nahrávaný soubor a žádost `FileUploadRequest`, která se použije pro vytvoření těla HTTP požadavku. V nejobecnějším případě tato žádost musí dodat název první a druhé části *multipart/form-data* těla.

Implementace metody `uploadFile` využívá odlehčenou knihovnu OkHttp. Žádost `FileUploadRequest` se použije k pojmenování částí těla HTTP požadavku a v serializované podobě se uloží do jeho první části. Do druhé části se pak uloží odesílaný soubor – viz kód 7.5. Následně se přes službu Authorizer získá přístupový token, doplní se hlavička a požadavek se odešle na danou URL. Nakonec se ošetří případné chyby a vrátí se `FileUploadResult`.

```
1  @Override
2  public FileUploadResult uploadFile(FileUploadRequest
       fileUploadRequest, File file, URL url) {
3      String form = WSJsonParserImpl.getInstance().
           serializeWithJackson(fileUploadRequest);
4
5      RequestBody body = new MultipartBody.Builder()
6              .setType(MultipartBody.FORM)
7              .addFormDataPart(
8                      fileUploadRequest.getJSONFieldName(),
9                      null,
10                     RequestBody.create(okhttp3.MediaType.parse(
                           "application/json"), form))
11             .addFormDataPart(
12                     fileUploadRequest.getFileFieldName(),
13                     file.getName(),
14                     RequestBody.create(okhttp3.MediaType.parse(
                           "application/octet-stream"), file))
15             .build();
16                 ⋮
```

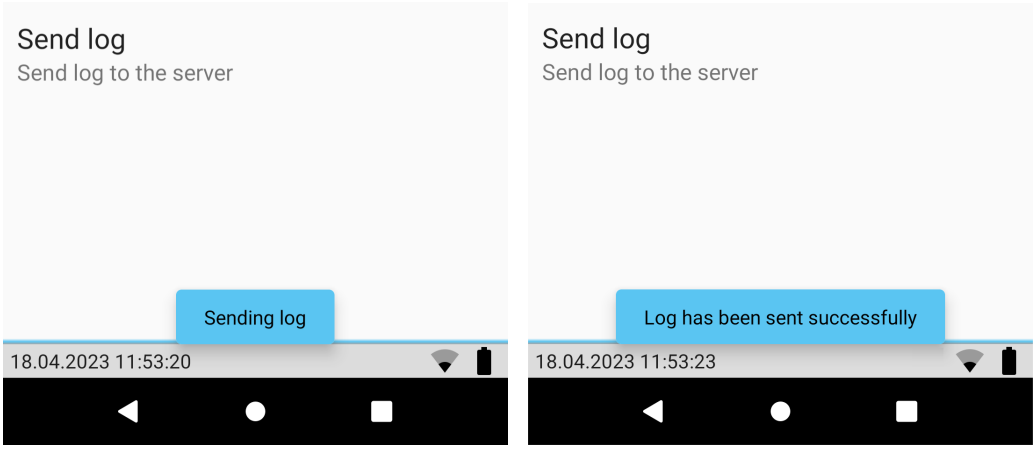
Výpis kódu 7.5: Začátek metody `uploadFile`

Pro nahrávání logů lze využít žádost `UploadLogsRequest` generovanou přímo z OpenAPI specifikace služby LogCollector (viz 7.2). Aby ji bylo možné použít v metodě `uploadFile`, je od ní oděděna nová třída se stejným názvem implementující obě metody rozhraní `FileUploadRequest`. Žádost je vytvářena ve třídě `LogManager` a společně s nahrávanýn souborem předávána metodě `uploadLogs` v `RequestExecutor`, která vytváří URL ke službě LogCollector a odesílá požadavek pomocí `Connectoru`. Hlavní třídy a metody pro nahrávání logů (resp. souborů) na server lze vidět na obrázku 7.5.



Obrázek 7.5: Odeslání aktivního logu (2. část)

Průběh úspěšného odeslání aktivního logu z nastavení aplikace na službu LogCollector lze vidět na obrázcích 7.6a a 7.6b.



(a) Informace o startu odesílání (b) Informace o úspěchu

Obrázek 7.6: Úspěšné odeslání aktivního logu

7.4 WebSocket spojení

Pro zasílání zpráv ze serveru do mobilního zařízení byla využita technologie WebSocket, která již byla v aplikaci používána, konkrétně ve třídě `SemaphoRePresenter` související s jednou konkrétní transakcí. Téměř veškerý kód

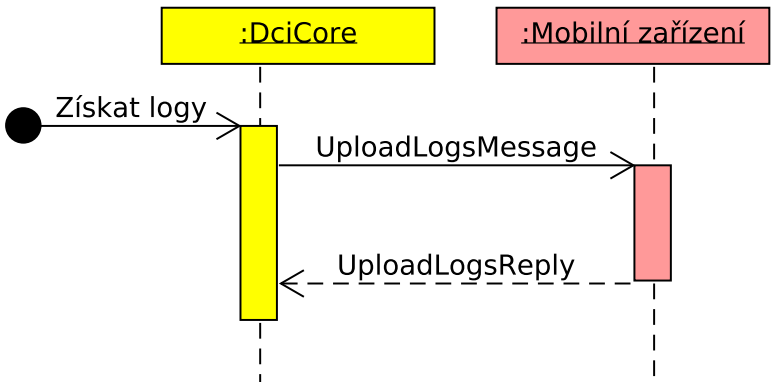
pro vytváření a správu spojení se nacházel v této třídě a zajišťoval také *pong* odpovědi na *ping* zprávy zasílané ze serveru a opětovné navazování spojení při chybě.

Pro potřeby této práce bylo zavedeno nové spojení, tzv. *klientské spojení*, které bude otevřené po celou dobu přihlášení uživatele v aplikaci. Spojení bude zatím používáno jen pro posílání požadavků na nahrání logů, v budoucnu je ale možnost používat jej i k jiným účelům. Jelikož dosavadní spojení v `SemaphorePresenter` zůstane v aplikaci i nadále a jeho kód tvořil dobrý základ pro nové spojení, byl tento kód refaktorován a rozšířen tak, aby umožňoval jednotnou správu obou spojení a také aby umožňoval přidat nový typ spojení, pokud to bude někdy potřeba.

Zasílání požadavků má iniciovat uživatel webového rozhraní, jež je obsluhováno službou DciCore. Komunikace serveru se zařízeními tedy probíhá z této služby. Stejně tomu bylo i v případě již existujícího spojení, neboť služba DciCore je obecně využívána pro více typů WebSocket spojení.

7.4.1 Komunikační rozhraní

Zasílané zprávy jsou definované pomocí objektů pro přenos dat (Data Transfer Objects), které využívá jak klientská, tak serverová část. Veškeré zprávy dědí od abstraktní třídy `Message`, která obsahuje pouze typ zprávy. Požadavek na odeslání logů `UploadLogsMessage` přidává počáteční a koncový datum logů a identifikátor požadavku. Odpověď `UploadLogsReply` pak posílá zprávu o (ne)úspěchu, případný chybový kód a stejný identifikátor zpět. Průběh komunikace mezi serverem DciCore a mobilním zařízením je vidět na sekvenčním diagramu na obrázku 7.7.



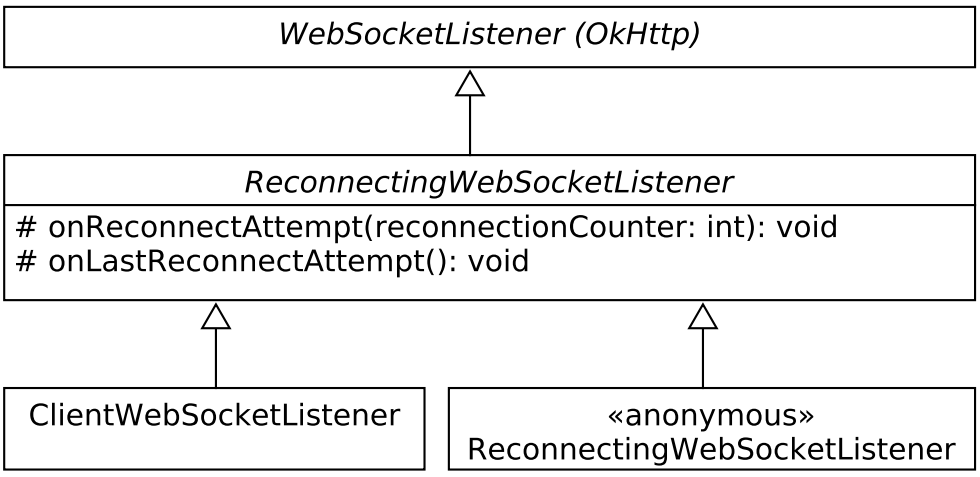
Obrázek 7.7: WebSocket zprávy `UploadLogsMessage` a `UploadLogsReply`

Dále server, stejně jako tomu bylo dosud, posílá v pravidelných časových intervalech *ping* zprávy, na které klient odpovídá *pong* zprávami. Nově však klient kontroluje, že tyto zprávy chodí, a pokud nedorazí do vypršení časového limitu, pokouší se iniciovat nové spojení (viz dále).

7.4.2 Klientská část

Klientská část pro implementaci WebSocket vychází z kódu, který byl původně součástí třídy `SemaphorePresenter` a který využívá knihovnu OkHttp. Pro jeho rozšíření byly ve výčtovém typu `WebSocketType` definovány 2 typy spojení – `SEMAPHORE` a `CLIENT`. Pro správu spojení pak byla zavedena třída `WebSocketManager`, která uchovává mapu spojení a poskytuje obecné metody pro vytváření nových spojení, opětovné vytváření existujících spojení či uzavření všech spojení.

Většina kódu ze `SemaphorePresenter` byla převedena do nové abstraktní třídy `ReconnectingWebSocketListener`. Tato třída tedy obsahuje společný kód zajišťující opětovné navazování spojení v případě jeho ztráty a nové typy spojení mohou tuto třídu rozšiřovat. Konkrétně se jedná o třídy `ClientWebSocketListener` a anonymní vnitřní třídu v `SemaphorePresenter` – viz obrázek 7.8.



Obrázek 7.8: Listenery pro WebSocket spojení

Do abstraktní třídy byly dále přidány metody `onReconnectAttempt` a `onLastReconnectAttempt`, které mohou potomci implementovat a reagovat odpovídajícím způsobem na pokusy o opětovné připojení. Také je z této třídy

startováno nově vytvořené vlákno `WebSocketPing`, které pravidelně kontroluje, zda přišla *ping* zpráva ze serveru, a tedy zda je spojení stále aktivní. Pokud tomu tak není, pak dané spojení ruší a třída `ReconnectingWebSocketListener` se bude snažit o vytvoření nového spojení.

Klientské spojení je vytvářeno ze třídy `ClientWebSocketFactory`. K vytvoření spojení se využívá již zmíněný `ClientWebSocketListener`, který je do této třídy injektován. Ve třídě bylo implementováno rozhraní `LogonRequestListener`, aby bylo možné ji použít jako posluchače žádosti o přihlášení uživatele do aplikace. Po úspešném přihlášení do aplikace tato třída vytvoří klientské spojení, které zůstane aktivní a bude zrušeno až po odhlášení z aplikace.

Indikace aktivního klientského spojení je řešena ikonou na spodní liště obrazovky. Jde o vlastní grafickou komponentu `WebSocketImageView` rozšiřující `AppCompatImageView`. Ikona monitoruje stav spojení, který je ukládán v `ClientWebSocketListener` pomocí Android třídy `MutableLiveData`, jež je určena pro držení pozorovatelných dat. Samotná ikona pak může nabývat 4 stavů:

- Skrytá ikona: Uživatel není přihlášen.
- Černá barva: Uživatel přihlášen, klientské spojení aktivní.
- Oranžová barva: Klientské spojení bylo ztraceno, probíhají pokusy o opětovné připojení.
- Červená barva: Klientské spojení bylo ztraceno a nepodařilo se obnovit, nové pokusy lze vyvolat kliknutím na ikonu a potvrzením dialogu.

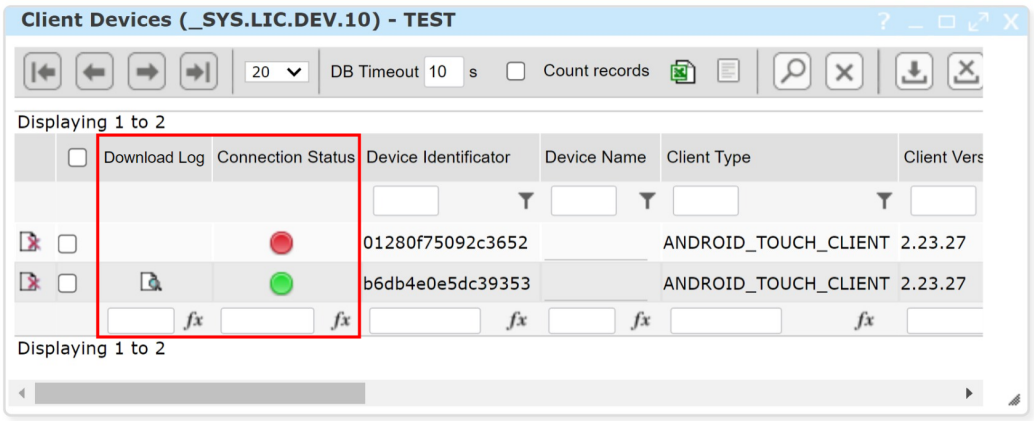
7.4.3 Serverová část

Serverová část již obsahovala výčtový typ s typy WebSocket spojení. Zde tedy byl jen definován nový typ spojení `CLIENT` a přidán handler pro zpracování zpráv – třída `ClientMessageHandler`. Ta kromě toho, že dědí od standardního `PingHandler` zpracovávajícího odpovědi na *ping* zprávy, přidává také zpracování zpráv `Message` (viz 7.4.1). Zpracování zde znamená jejich uložení do mapy, kde klíčem je typ zprávy a hodnotou fronta zpráv daného typu. Odtud je pak bude možné vyzvednout k dalšímu zpracování.

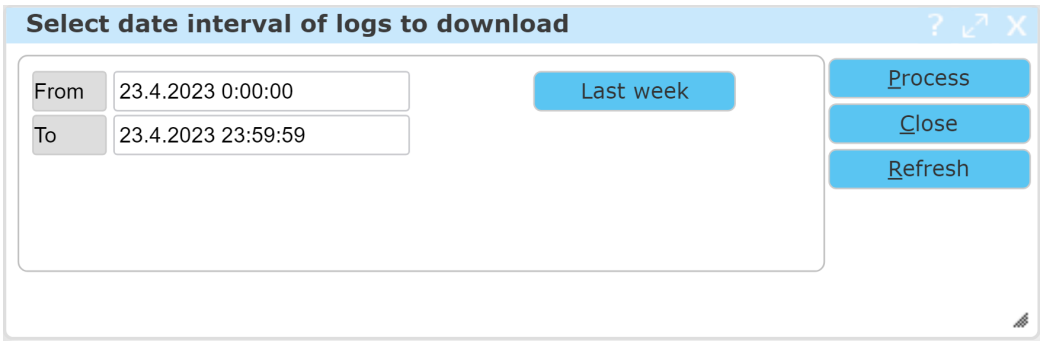
Pro indikaci aktivního spojení byl přidán nový sloupec Connection Status na obrazovku se zařízeními (obrázek 7.9). Při vytváření obrazovky se vždy podle identifikátoru zařízení zjistí, zda pro dané zařízení existuje aktivní spojení, a dle toho se nastaví červená/zelená ikonka v tomto sloupci.

7.5 Vyžádání logů uživatelem DCIx

Pro vyžádání logů z webového rozhraní DCIx byl na obrazovku se zařízeními přidán nový sloupec Download Log (obrázek 7.9). Je-li WebSocket spojení s daným zařízením aktivní, lze z tohoto sloupce otevřít novou obrazovku pro výběr data a zaslání požadavku (obrázek 7.10).



Obrázek 7.9: Obrazovka se zařízeními v DCIx po rozšíření



Obrázek 7.10: Obrazovka pro výběr data a zaslání požadavku

Pro tento účel byla vytvořena nová Struts akce `UploadClientLogsAction`. Akce má jeden forward s názvem `selectDate`, jímž je JSP vytvářející obrazovku, a pro ukládání dat slouží akci form bean třída `UploadClientLogsForm`. Při vytváření akce je na instanci této třídy uložen identifikátor zařízení, jméno zařízení a typ klientské aplikace, aby je bylo možné později použít. Samotná URL akce má jeden parametr, který určuje, co bude akce vykonávat, a může nabývat jedné ze 3 hodnot: *Prepare*, *Execute* a *Refresh*.

Prepare připravuje hodnoty počátečního a koncového data logů, aby nebylo nutné je zadávat ručně. Vykonává se jako první kód při otevření obrazovky, kdy nastavuje hodnoty pro aktuální den, a také po kliknutí

na tlačítko Last week, kdy nastavuje hodnoty pro poslední týden. Pokud je potřeba vybrat jiný interval, lze k tomu použít vyskakovací kalendáře u políček From a To.

Execute se provádí po odeslání formuláře uživatelem kliknutím na tlačítko Process. Hodnoty pro počáteční a koncový datum, jež byly získány z formuláře a uloženy do instance `UploadClientLogsForm`, se použijí pro vytvoření zprávy `UploadLogsMessage`, která se společně s jedinečným identifikátorem odešle do zařízení. Následně je zpráva uložena mezi ostatní nevyřízené požadavky na `UploadClientLogsForm`.

Refresh se vykonává po kliknutí na tlačítko Refresh a dotazuje se na přítomnost odpovědí na požadavky. Nejprve nalezne handler WebSocket spojení daného zařízení a z něj získá frontu odpovědí `UploadLogsReply`. Poté odpovědi prochází a dle identifikátoru je přiřazuje k nevyřízeným požadavkům. Požadavky s přiřazenou odpovědí budou zobrazeny pomocí JSP a následně při dalším vykonávání akce odebrány.

Obrazovka na obrázku 7.10 je vytvářena pomocí JSP s názvem `selectDate`. Kromě elementů viditelných na obrázku může dále pod výběrem dat zobrazovat případné chyby, které mohly vzniknout při kliknutí na Process (prázdný datum, nevalidní datum, From starší než To nebo odpojené zařízení), a také informativní zprávy. Dále prochází uložené požadavky a ty, které mají přiřazenu odpověď, jsou zobrazeny. U každého zobrazeného požadavku je buď chybová zpráva obdržená ze zařízení, nebo zpráva o úspěchu doprovázená odkazem do Kibany k prohlížení požadovaných logů.

7.5.1 LogManager

`LogManager` je třída v klientské aplikaci sloužící k nahrávání logů na server. Kromě metody `uploadActiveLog` popisované v části 7.3.2 obsahuje dále metodu `uploadLogs`, která je volána po obdržení WebSocket zprávy `UploadLogsMessage`.

V této metodě je nejdříve procházen adresář s logovacími soubory a z názvu každého souboru a časové zóny zařízení je určen okamžik na časové ose představující začátek daného dne. Počáteční a koncový datum požadavku ve skutečnosti nejsou jen data, ale obsahují i čas a časový posun, tj. také představují okamžiky na časové ose, a lze tedy provést porovnání a určit, zda soubor patří do vymezeného intervalu. Ačkoli logovací soubory představují dny a jsou vždy odesílány celé, mají čas a časový posun smysl, neboť pro pokrytí části dne v jedné časové zóně může v jiné časové zóně být potřeba i den předchozí či následující a pro pokrytí doby na rozhraní dvou dnů může naopak postačit jediný den.

Soubory, které byly vybrány k odeslání, jsou nejprve rozbaleny do dočasného adresáře (zařízení totiž všechny logovací soubory kromě aktivního uchovává v archivech) a následně je z nich vytvořen jeden společný archiv. Dále se podobně jako při odesílání aktivního logu vytvoří žádost `UploadLogsRequest` a logy se odešlou pomocí `RequestExecutor`. Dle vráceného `FileUploadResult` je vytvořena odpověď `UploadLogsReply`, která je poslána zpět na DciCore. Texty odpovědí nejsou lokalizovány do jazyka zařízení, protože ten může být odlišný od jazyka používaného na DciCore, a je tedy lepší použít vždy univerzální angličtinu.

7.6 Vizualizace logů

Aby byl na službě LogCollector monitorován nástrojem Filebeat nově i adresář `clients` s logy ze zařízení, byl přidán do souboru s prospektory nový prospektor. Jelikož záznamy v logovacích souborech mohou být i víceřádkové a Filebeat ve výchozím nastavení zpracovává logovací soubory řádku po řádce, je v prospektoru pro správné parsování víceřádkových záznamů uveden regulární výraz popisující časovou značku, jíž každý logovací záznam začíná. Pokud pak začátek řádky nevyhovuje uvedenému výrazu, bude řádka připojena k řádce předchozí.

V prospektoru je také uveden tag `client-log`, pod nímž bude Filebeat nahrávat záznamy do Elasticu. Tento tag pak lze použít k vyfiltrování záznamů v Kibaně a taktéž je použit jako filtr při sestavování odkazu pro uživatele webového rozhraní, jenž logy vyžádal.

8 Testování řešení

V této kapitole je popsáno, jakým způsobem byla ověřována funkčnost vytvořeného řešení. Kapitola je rozdělena do dvou sekcí – automatické testování, které využívá testovací software, a ruční testování, které bylo prováděno vykonáváním testovacích scénářů. Dle jejich zaměření jsou zde automatické testy dále rozděleny na jednotkové, Android UI a kontraktové testy.

Testy lze také dělit dle spolupráce testovaných komponent na jednotkové, integrační (mezi něž lze zařadit Android UI a kontraktové testy) a systémové testy, které zde byly provedeny ručně. V případě Androidu se ještě rozlišují lokální testy vyžadující pro svůj běh pouze Java Virtual Machine a instrumentační testy, jež musí být spouštěny v Android zařízení (reálném či emulátoru).

8.1 Automatické testování

Automatické testování provádí stroj za pomoci testovacího softwaru. Pro účely této práce bylo vytvořeno několik testů, které lze rozdělit na jednotkové, Android UI a kontraktové.

8.1.1 Jednotkové testování

Jednotkové testy se zaměřují na testování izolovaných jednotek programu (např. metod nebo tříd). Mezi nejvýznamnější frameworky používané k jednotkovému testování v Javě patří JUnit a Mockito. Kromě obou zmíněných byl pro účely práce použit také AssertJ, který umožňuje vytváření „plynulých“ testovacích tvrzení (asercí), jež připomínají přirozený jazyk.

Vytvořeno bylo několik testů pro klientskou aplikaci, službu LogCollector a DciCore. Při psaní testů byly použity jmenné konvence. Testovací třídy mají stejný název jako třídy testované a přidávají řetězec `Test`. Testovací metody používají pojmenování `given-when-then` s možným vynecháním části `given`. První část představuje předpoklady prováděné akce, prostřední část samotnou akci a poslední část očekávaný výsledek.

Příklad primitivního testu používajícího uvedenou konvenci je ve výpisu 8.1. Název metody lze interpretovat následujícím způsobem: *„Existuje-li klientské spojení a má-li být vytvořeno, pak vytvořeno nebude“*

```
1 @Test
2 void givenClientConnectionExists_whenCreate_thenDontCreate() {
3     when(mWebSocketManagerMock.getClientConnection()).thenReturn(
          mock(WebSocket.class));
4
5     mClientWebSocketFactory.createClientWebSocketConnection();
6
7     verify(mWebSocketManagerMock, never()).
          createWebSocketConnection(any(), any(), any(), any());
8 }
```

Výpis kódu 8.1: Pojmenování `given-when-then`

8.1.2 Testování Android UI

Při testování uživatelského rozhraní již nejsou testované jednotky v izolaci, ale spolupracují. Rozšířeným frameworkem pro UI testování Android aplikací je Espresso. Jelikož však spouštění těchto testů vyžaduje běh Android zařízení, jsou znatelně pomalejší než jednotkové testy. Ze stejného důvodu se jich také píše menší počet.

Framework Espresso byl využit pro otestování nového tlačítka *Send log* v nastavení.

8.1.3 Kontraktové testování

Endpointy služeb lze testovat pomocí kontraktových testů. Kontraktové testy testují kontrakt, tj. chování, k němuž se poskytovatel API zavazuje.

Scénář pro otestování kontraktu obsahuje popis požadavku a očekávanou odpověď (HTTP stavový kód a tělo odpovědi). Lze jej popsat ručně v jazyce Groovy a z tohoto popisu poté např. pomocí pluginu Spring Cloud Contract Verifier nechat vygenerovat JUnit test. Druhou možností je scénáře psát přímo do OpenAPI specifikace mezi příklady a JUnit testy pak generovat přímo z těchto příkladů.

Pro otestování endpointu `/uploadLogs` byl vytvořen jeden kontraktový test. Jelikož generování kontraktových testů z OpenAPI je problematické pro požadavky typu *multipart/form-data*, byl scénář popsán ručně v souboru `uploadLogs_200.groovy` (pro úplnost byly doplněny i příklady do OpenAPI, ale kontraktový test se z nich negeneruje). Scénář posílá jako první část požadavek z výpisu 7.2 a jako druhou část odpovídající soubor `example.zip`. Očekává pak stavový kód 200 a prázdný JSON objekt.

Při nahrávání logů na LogCollector se cesta k adresáři s logy zjišťuje za běhu programu pomocí logovacího frameworku. Ten ale při kontraktovém testu není spuštěn, a proto musí být metoda pro zjištění cesty namockována. Mock se nachází ve třídě `BaseTestClass`, která je rodičem generovaných kontraktových testů a slouží k podobným účelům. Místo adresáře logovacího frameworku je pro uložení logů použita cesta `target/test-logs`, čímž je i zajištěno, že soubory budou uklízeny při úklidu pomocí Mavenu.

8.2 Ruční testování

Ruční (resp. manuální) testování bylo prováděno vykonáváním definovaných testovacích scénářů a probíhalo na zařízení Pixel 2, které je dostupné v emulátoru Android Studia. Následuje specifikace scénářů, které byly testovány. Scénáře vycházejí z požadavků uvedených v kapitole 5 a z návrhu řešení. V průběhu testování bylo odhaleno několik míst, kde skutečnost neodpovídala scénáři. Chyby však byly opraveny a všechny scénáře tak byly splněny.

Odeslání aktivního logu

Aktéři:

- Uživatel mobilní aplikace

Základní tok:

1. Uživatel se přihlásí do aplikace a jde do nastavení.
2. Uživatel klikne na tlačítko pro odeslání aktivního logu.
3. Aplikace odešle log a informuje uživatele o úspěchu.

Alternativní tok 1:

1.1 Uživatel se nepřihlásí a jde do nastavení.

1.2 Uživatel nevidí tlačítko pro odeslání aktivního logu.

Alternativní tok 2:

3.1 Aplikaci se nepodaří log odeslat a informuje uživatele o neúspěchu.

Trvalé spojení

Aktéři:

- Uživatel mobilní aplikace

Základní tok:

1. Uživatel se přihlásí do aplikace.
2. Aplikace zobrazí černou ikonku na spodní liště. Po kliknutí je uživatel informován o aktivním spojení.
3. Uživatel se odhlásí.
4. Aplikace skryje ikonku.

Alternativní tok 1:

3.1 Aplikace ztratí spojení a zobrazí oranžovou ikonku. Po kliknutí je uživatel informován o pokusech navázat spojení.

3.2 Aplikaci se podaří navázat spojení a pokračuje se bodem 2.

Alternativní tok 2:

3.2.1 Aplikaci se nepodaří navázat spojení a zobrazí červenou ikonku. Po kliknutí je uživatel informován o ztrátě spojení a možnosti opětovných pokusů.

3.2.2 Uživatel zažádá o opětovné pokusy a pokračuje se bodem 2 nebo 3.1.

Vizualizace logů

Aktéři:

- Uživatel webového rozhraní

Základní tok:

1. Uživatel otevře Kibanu, vidí všechny logy a tagem `client-log` může vyfiltrovat logy z klientských zařízení.

Stažení logů zařízení pomocí webového rozhraní

Aktéři:

- Uživatel webového rozhraní

Základní tok:

1. Uživatel se přihlásí do webového rozhraní DCIx a otevře obrazovku se zařízeními.
2. Vybrané zařízení má aktivní spojení, zobrazuje se zelená ikonka a možnost stáhnout logy.

3. Uživatel zvolí možnost stáhnout logy.
4. Zobrazí se obrazovka pro výběr data a zaslání požadavku. Jsou předvyplněny hodnoty pro dnešní den.
5. Uživatel zadá počáteční a koncový datum a odešle požadavek.
6. Zobrazí se informace o odeslání požadavku do zařízení.
7. Uživatel se tlačítkem dotáže na existenci odpovědi nebo opakuje bod 5.
8. Zobrazí se všechny obdržené a dosud nezobrazené odpovědi na požadavky zaslané do daného zařízení. Společně s odpovědí se zobrazí informace o požadavku a odkaz do Kibany v případě úspěchu požadavku.
9. Uživatel klikne na odkaz do Kibany.
10. Otevře se Kibana s filtrem na logy nahrané z daného zařízení.

Alternativní tok 1:

2.1 Vybrané zařízení nemá aktivní spojení, zobrazuje se červené ikonka a chybí možnost stáhnout logy.

Alternativní tok 2:

6.1 Požadavek se nepodaří odeslat a zobrazí se informace o příčině.

Alternativní tok 3:

8.1 Zobrazí se informace, že nejsou k dispozici žádné odpovědi.

9 Možná rozšíření

Vytvořené řešení se skládá z několika částí a pro každou z nich si lze představit její rozšíření, stejně jako si lze představit rozšíření či vylepšení jejich vzájemné spolupráce.

9.1 Vzájemná komunikace

Co se týká nahrávání souborů na službu LogCollector, bylo by vhodné umožnit přijímání více souborů, které nejsou v archivu. Tato možnost nebyla implementována, protože se počítá především se soubory v jediném archivu a příjem více souborů by zřejmě vyžadoval složitější návrh API. Na druhou stranu je umožněno nahrát jeden soubor, který není archivem, a nahrávání více souborů by tedy bylo logické.

API LogCollectoru by také bylo možné rozšířit o nový endpoint umožňují dynamickou konfiguraci úklidu logů. Z webového rozhraní by pak šlo nastavovat, kdy má probíhat úklid logů, a určovat limit pro stáří souborů.

Komunikaci mezi DciCore a mobilním zařízením by šlo rozšířit o dotazy do zařízení pro zjištění, jaké logovací soubory má k dispozici. Grafické uživatelské rozhraní by pak uživateli místo zadávání dat do vyskakovacích kalendářů umožňovalo výběr přímo z dostupných souborů.

Dalším rozšířením komunikace by byla možnost pro nastavení stahování logů v pravidelných časových intervalech, možnost pro stahování pouze logů od zvolené logovací úrovně výše nebo možnost přepínání logovací úrovně zařízení. Otázkou však je, jak často by takové funkcionality byly využívány.

Vytvořené klientské WebSocket spojení lze využít i k libovolnému jinému účelu. Například by bylo možné po nastavení jména zařízení ve webovém rozhraní poslat WebSocket zprávu, která by informovala zařízení o jeho novém jménu. Také si lze představit situaci, kdy bude užitečné pro nějakou novou DCIx transakci vytvořit nový typ WebSocket spojení.

9.2 LogCollector

Na službě LogCollector by bylo vhodné najít způsob jak otagovat logové záznamy identifikátorem zařízení. Identifikátor zařízení je sice součástí cesty k logovému souboru, která je v Kibaně také zobrazována, ale pro lepší filtrování by bylo vhodné mít samostatný tag pro identifikátor.

Z hlediska testování by se hodilo rozšíření pluginu pro kontraktové testy tak, aby dokázal generovat testy i pro *multipart/form-data* požadavky a nebylo nutné je psát ručně.

9.3 Android aplikace

Užitečným rozšířením Android aplikace by bylo, pokud by uměla odesílat jen části logovacích souborů. Jelikož služby DciCore i LogCollector již stejně pracují s datem i časem, nebylo by nutné na těchto službách nic měnit a byl by realizován jemnější výběr logů. Řešení by však zřejmě bylo výpočetně i implementačně náročnější než odesílání celých souborů.

Jiným velkým rozšířením Android aplikace by bylo vytvoření samostatné knihovny na odesílání logů. Lze si představit knihovnu nezávislou na aplikaci, u které by bylo možné nakonfigurovat:

- URL, kam se mají posílat logy,
- úložiště logů v zařízení, z něhož se mají posílat logy,
- způsob jak získat přístupový token,
- způsob šifrování přenosu a
- parametry požadavku (identifikátor a jméno zařízení, typ aplikace).

Součástí knihovny by byly i zprávy `UploadLogsMessage` a po přijetí těchto zpráv nebo po kliknutí na tlačítko pro nahrání aktivního logu by řízení přebírala knihovna. V prvním případě by pak vracela `UploadLogsReply`.

9.4 DciCore

U obrazovky zařízení by při větším počtu zařízení bylo užitečné řazení nebo filtrování dle sloupce Connection Status, aby bylo možné zobrazit jen zařízení, která (ne)mají aktivní spojení.

Na obrazovce pro vyžádání logů by nebylo na škodu zobrazovat po kliknutí na tlačítko Refresh nevyřízené požadavky, na něž zatím nepřišla odpověď.

10 Závěr

Hlavním cílem této bakalářské práce bylo usnadnit proces získávání logů z Android zařízení používaných ve skladech. Zmíněná zařízení provozují klientskou aplikaci komunikující s informačním systémem zadavatele a umožňují provádění nadefinovaných skladových operací. Aplikace vytváří logové soubory, které uchovává ve svém úložišti, a pro jejich získání byl dříve nutný fyzický přístup k zařízení. Požadavkem bylo fyzický přístup eliminovat a umožnit sběr a přenos logů ze zařízení na server vzdáleně.

První polovina práce je teoretická a začíná popisem rozšiřovaných produktů a prostředí, v němž se používají. Následuje část věnující se důležitosti a problematice logování, analýze úlohy a požadavků a popisu souvisejících technologií. Druhá polovina práce začíná návrhem řešení bez uvádění implementačních detailů. Ty jsou pak podrobně popsány v následující části se zcela stejnou strukturou. Následuje část věnující se testování řešení a poté je text práce zakončen uvedením možných rozšíření.

Pro sběr logů bylo navrženo REST API popsané pomocí OpenAPI specifikace a implementované pomocí Spring Boot aplikace. Toto API není nijak vázáno na Android a bude možné jej použít i pro jinou platformu. Kromě samotného sběru logů byl umožněn i jejich automatický úklid a vizualizace ve vizualizačním systému Kibana.

V klientské Android aplikaci bylo založeno nové WebSocket spojení, které je aktivní po celou dobu přihlášení uživatele do aplikace a přijímá požadavky na odeslání logů. Správa spojení byla řešena dostatečně obecně, aby byla možnost v budoucnu jednoduše přidat další spojení. Stav spojení je uživateli indikován novou ikonkou. Také odeslání logů ze zařízení na REST API bylo řešeno dostatečně obecně a vytvořené řešení bude možné použít i k nahrávání jiných souborů na server.

Vyžádání logů bylo umožněno přidáním nové obrazovky do webového rozhraní systému. Ta také umožňuje zobrazovat odpovědi na požadavky zaslané do zařízení. Podobně jako do mobilního zařízení byla i do webového rozhraní přidána indikace stavu spojení.

Vytvořené řešení je funkční a splňuje požadavky, jejichž specifikace byla uvedena na konci první části textu. Kód byl otestován jednotkovými testy, testy uživatelského rozhraní systému Android a jedním kontraktovým testem. Celková funkčnost pak byla ověřena vykonáním testovacích scénářů.

Ačkoli to nebylo na první pohled zřejmé, vyžadovala úloha vyřešení celé řady situací. Práce tak zcela jistě přinesla mnoho nových zkušeností.

Literatura

[1] Keřkovský, M. *Moderní přístupy k řízení výroby, 3. doplněné vydání.* Nakladatelství CH Beck, 2012.

[2] *Definice logistiky Evropské logistické asociace* [online]. Euro.cz, 2003. [cit. 2022/12/06]. Dostupné z: https://www.euro.cz/clanky/definice-logistiky-evropske-logisticke-asociace-867920/.

[3] Sethi, A. K. – Sethi, S. P. Flexibility in Manufacturing: A Survey. *International Journal of Flexible Manufacturing Systems.* 1990, 2, 4, s. 289–328.

[4] Andrejić, M. M. Measuring Efficiency in Logistics. *Vojnotehnički glasnik.* 2013, 61, 2, s. 84–104.

[5] *What is SCM (supply chain management) and why is it important?* [online]. SAP. [cit. 2022/12/18]. Dostupné z: https://www.sap.com/insights/what-is-supply-chain-management-scm.html.

[6] *Logistický slovník – SCM* [online]. Česká logistika. [cit. 2022/12/18]. Dostupné z: https://www.ceskalogistika.cz/scm/.

[7] Žák, R. Manufacturing Operations Management. *IT Systems.* 2015, 6, s. 2–4. Dostupné z: https://www.systemonline.cz/rizeni-vyroby/mom-manufacturing-operations-management.htm.

[8] *Warehouse Management Systems Market Size* [online]. Grand View Research. [cit. 2022/12/02]. Dostupné z: https://www.grandviewresearch.com/industry-analysis/warehouse-management-system-wms-market.

[9] *Aimtec a.s. – O společnosti* [online]. Aimtec. [cit. 2022/12/06]. Dostupné z: https://www.aimtecglobal.com/o-spolecnosti/.

[10] Aimtec. Logo Aimtec, 2017. Dostupné z: interní dokument.

[11] *DCIx* [online]. Aimtec. [cit. 2022/12/20]. Dostupné z: https://www.aimtecglobal.com/dcix/.

[12] Krásný, T. *Implementace Android klienta pro webovou aplikaci DCIx.* Západočeská univerzita v Plzni, 2016.

[13] *GS1 General Specifications Standard* [online]. GS1. [cit. 2023/03/25]. Dostupné z: https://www.gs1.org/genspecs.

[14] *What Is a Thermal Printer?* [online]. Zebra Technologies. [cit. 2023/03/18]. Dostupné z: https://www.zebra.com/gb/en/resource-library/faq/printing/what-is-a-thermal-printer.html.

[15] *Selecting the Right Mobile Device* [online]. Zebra Technologies, 2016. [cit. 2023/03/20]. Dostupné z: https://www.zebra.com/content/dam/zebra_new_ia/en-us/campaigns/os-migration/find-the-right-devices/20963_ZEB_Consumer_vs_Enterprise_WhitePaper_INTERACTIVE%202.pdf.

[16] *MC9300 Handheld Mobile Computer Spec Sheet* [online]. Zebra Technologies. [cit. 2023/03/20]. Dostupné z: https://www.zebra.com/gb/en/products/spec-sheets/mobile-computers/handheld/mc9300.html.

[17] *Buyer's Guide: Current Trends in Mobile WMS Devices* [online]. Magaya. [cit. 2023/03/26]. Dostupné z: https://www.magaya.com/buyers-guide-current-trends-in-mobile-wms-devices/.

[18] *Frequently asked questions* [online]. Android. [cit. 2023/03/20]. Dostupné z: https://source.android.com/docs/setup/about/faqs#what-does-compatibility-mean.

[19] Steinberger, A. *Konec podpory Windows CE a Windows Mobile se blíží. Přejděte na Android.* [online]. Aimtec. [cit. 2023/03/21]. Dostupné z: https://www.aimtecglobal.com/aimagazine/konec-podpory-windows-ce-a-windows-mobile-se-blizi-prejdete-na-android/.

[20] Chuvakin, A. – Schmidt, K. – Phillips, C. Chapter 20 - Planning Your Own Log Analysis System. In Chuvakin, A. – Schmidt, K. – Phillips, C. (Ed.) *Logging and Log Management*. Boston: Syngress, 2013. s. 367–379. doi: https://doi.org/10.1016/B978-1-59-749635-3.00020-8. Dostupné z: https://www.sciencedirect.com/science/article/pii/B9781597496353000208. ISBN 978-1-59749-635-3.

[21] *Extended Log File Format* [online]. W3C. [cit. 2023/04/08]. Poznámka: Jedná se o pracovní návrh (working draft). Dostupné z: http://www.w3.org/pub/WWW/TR/WD-logfile-960221.html.

[22] Rakhmetova, E. – Garosi, M. – Combi, C. A comprehensive approach to conceptual modelling and visual representation of log files. In *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*, s. 385–388, 2022.

[23] Chuvakin, A. – Schmidt, K. – Phillips, C. Chapter 18 - Logging for Programmers. In Chuvakin, A. – Schmidt, K. – Phillips, C. (Ed.) *Logging and Log Management*. Boston: Syngress, 2013. s. 329–342. doi: https://doi.org/10.1016/B978-1-59-749635-3.00018-X. Dostupné z: https:

//www.sciencedirect.com/science/article/pii/B978159749635300018X. ISBN 978-1-59749-635-3.

[24] Gerhards, R. RFC 5424: The syslog protocol, 2009. Dostupné z: https://www.rfc-editor.org/rfc/rfc5424.

[25] El-Masri, D. et al. A systematic literature review on automated log abstraction techniques. *Information and Software Technology*. 2020, 122. ISSN 0950-5849. doi: https://doi.org/10.1016/j.infsof.2020.106276. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0950584920300264.

[26] Chuvakin, A. – Schmidt, K. – Phillips, C. Chapter 21 - Cloud Logging. In Chuvakin, A. – Schmidt, K. – Phillips, C. (Ed.) *Logging and Log Management*. Boston: Syngress, 2013. s. 381–399. doi: https://doi.org/10.1016/B978-1-59-749635-3.00021-X. Dostupné z: https://www.sciencedirect.com/science/article/pii/B978159749635300021X. ISBN 978-1-59749-635-3.

[27] *Log Injection* [online]. OWASP. [cit. 2023/03/01]. Dostupné z: https://owasp.org/www-community/attacks/Log_Injection.

[28] Česko. § 46 odst. 4 písm. c) zákona č. 110/2019 Sb., o zpracování osobních údajů - znění od 24. 4. 2019, 2019. Dostupné z: https://www.zakonyprolidi.cz/cs/2019-110#p46-4-c.

[29] Sally. *Apache™ Logging Services™ Project Announces Log4j™ 1 End-Of-Life; Recommends Upgrade to Log4j 2* [online]. The Apache Software Foundation Blog. [cit. 2023/05/03]. Dostupné z: https://news.apache.org/foundation/entry/apache_logging_services_project_announces.

[30] *Log4j 2 Architecture* [online]. The Apache Software Foundation. [cit. 2023/05/03]. Dostupné z: https://logging.apache.org/log4j/2.x/manual/architecture.html.

[31] *Log4j 2 Appenders* [online]. The Apache Software Foundation. [cit. 2023/05/03]. Dostupné z: https://logging.apache.org/log4j/2.x/manual/appenders.html.

[32] *Apache Log4j™ 2* [online]. The Apache Software Foundation. [cit. 2023/05/03]. Dostupné z: https://logging.apache.org/log4j/2.x/.

[33] *Reasons to prefer logback over log4j 1.x* [online]. QOS.ch Sarl (Switzerland). [cit. 2023/05/03]. Dostupné z: https://logback.qos.ch/reasonsToSwitch.html.

[34] Diaconu, A. *WebSocket alternatives* [online]. Ably. [cit. 2023/05/03]. Dostupné z: https://ably.com/topic/websocket-alternatives.

[35] *Spring Framework – Annotation Interface Scheduled* [online]. [cit. 2023/04/12]. Dostupné z: https://docs.spring.io/spring-framework/docs/6.0.7/javadoc-api/org/springframework/scheduling/annotation/Scheduled.html.

Seznam zkratek

ALAT Automated Log Abstraction Techniques

AOSP Android Open Source Project

API Application Programming Interface

BYOD Bring Your Own Device

CLF Common Log Format

EAN European Article Number

ELF Extended Log Format

EMDK Enterprise Mobility Development Kit

GMS Google Mobile Services

HTTP Hypertext Transfer Protocol

ICT Information and Communication Technologies

JDK Java Development Kit

JIS Just in Sequence

JIT Just in Time

JSON JavaScript Object Notation

JSP Jakarta Server Pages

LaaS Logging as a Service

MES Manufacturing Execution System

MFC Material Flow Control

MOM Manufacturing Operations Management

MVC Model–View–Controller

MVP Model–View–Presenter

PPS Production Planning System

QMS Quality Management System

QR Quick Response

REST Representational State Transfer

RFC Request for Comments

RFID Radio Frequency Identification

SCM Supply Chain Management

SLF4J Simple Logging Facade for Java

TCP Transmission Control Protocol

UDP User Datagram Protocol

UI User Interface

UML Unified Modeling Language

UPC Universal Product Code

URL Uniform Resource Locator

WMS Warehouse Management System

XML Extensible Markup Language

YAML YAML Ain't Markup Language

YMS Yard Management System