STREDNÁ PRIEMYSELNÁ ŠKOLA ELEKTROTECHNICKÁ
KOMENSKÉHO 44, 040 01 KOŠICE

MWP – MUSIC WITHOUT PAIN

LUKÁŠ RANDUŠKA

ADAM KROKAVEC

2024

STREDNÁ PRIEMYSELNÁ ŠKOLA ELEKTROTECHNICKÁ
KOMENSKÉHO 44, 040 01 KOŠICE

MWP – MUSIC WITHOUT PAIN

Záverečná práca

Praktická časť odbornej zložky maturitnej skúšky
Forma: Obhajoba vlastného projektu

2024
Košice

Riešitelia:
Lukáš Randuška
Adam Krokavec

Ročník štúdia: štvrtý

Konzultant:
Ing. Peter Keusch

ŠPECIFIKÁCIA PROJEKTU NA PRAKTICKÚ ČASŤ ODBORNEJ ZLOŽKY MATURITNEJ SKÚŠKY

Forma : Obhajoba vlastného projektu **Šk. rok:** 2023/2024

RIEŠITEĽ
Meno a priezvisko: Lukáš Randuška **Trieda:** IV.A
Študijný odbor a zameranie: 2561 M informačné a sieťové technológie - vývoj aplikácií

SPOLURIEŠITEĽ
Meno a priezvisko: Adam Krokavec **Trieda:** IV.A
Študijný odbor a zameranie: 2561 M informačné a sieťové technológie - správa systémov a sietí

Odborný konzultant: Ing. Peter Keusch
Metodický konzultant: Ing. Radoslav Bučko, PhD.

Názov projektu: **MWP (Music Without Pain) – Smart prehrávač hudby**

Charakteristika projektu: MWP je aplikácia určená pre mobilné zariadenia. Slúži na rýchle a efektívne spravovanie, prehliadanie a hlavne prehrávanie hudby s možnosťou vytvárania vlastných hudobných zoznamov a jednoduchým prenášaním alebo permanentnou synchronizáciou hudobnej knižnice medzi vašimi zariadeniami.

Úlohy:

Lukáš Randuška:

- Dizajn a implementácia užívateľského rozhrania pre spravovanie dôveryhodných WiFi sietí, host-ov/remote target-ov a synchronizačných cieľov.
- Dizajn a implementácia užívateľského rozhrania pre prezeranie, upravovanie a spravovanie ID3v2 tag-ov.
- Dizajn a implementácia užívateľského rozhrania pre kontrolu a nastavovanie používateľských preferencií (nastavení) aplikácie.
- Dizajn a implementácia užívateľského rozhrania pre prezeranie hudobnej knižnice vrátane všetkých hudobných súborov, albumov, interpretov a užívateľských zoznamov s možnosťou spúšťania hudobných súborov.
- Dizajn a implementácia užívateľského rozhrania pre sledovanie stavu a informácií o aktuálnej piesni s možnosťou pretočenia playback-u, prejdením na predchádzajúcu alebo nasledujúcu pieseň, zmenením poradia zoznamu aktuálne prehrávaných piesni (shuffle) a nastavením opakovania aktuálnej pesničky alebo zoznamu aktuálne prehrávaných pesničiek (loop).
- Dizajn a implementácia užívateľského rozhrania pre užívateľom vytvorené osobné zoznamy hudobných skladieb.
- Implementácia multimediálnej notifikácie a multimediálneho widget-u.

Schválil:

Odborný konzultant: Keusch Dátum schválenia: 24.10.2023

Vedúci PK: Kanková Dátum schválenia: 24.10.2023

Prevzal (podpis a dátum):

Riešiteľ projektu: 24.10.2023 Randuška Spoluriešiteľ projektu: 24.10.2023 Krokavec

Adam Krokavec:

- Rozšíriť aplikáciu o konsolidáciu hudobných súborov do jednej zložky kvôli jednoduchému zálohovaniu a údržbe s ohľadom na systémové súbory a rešpektovaním .nomedia súborov.
- Rozšíriť aplikáciu o integráciu služieb AcoustID a MusicBrainz pre automatické vyhľadávanie ID3v2 tag-ov cez fingerprint generovaný knižnicou Chromaprint.
- Rozšíriť aplikáciu o možnosť manuálnej úpravy ID3v2 tag-ov.
- Rozšíriť aplikáciu o jednorazovú, permanentnú alebo čiastočnú synchronizáciu hudobnej knižnice na iné zariadenia používateľa na vlastnom protokole aplikačnej vrstvy.
- Vytvoriť foreground service (aby sme dostali implicitný CPU wake-lock) na prehrávanie hudby s plným ohľadom na audio focus manažment operačného systému android.
- Integrácia multimediálnej notifikácie a widget-u.

Obaja:

- Napísať dokumentáciu v textovom editore.
- Vytvorenie prezentácie na obhajobu projektu.

Schválil:

Odborný konzultant: ..... Dátum schválenia: 24.10.2023

Vedúci PK: ..... Dátum schválenia: 24.10.2023

Prevzal (podpis a dátum):

Riešiteľ projektu: 24.10.2023 ..... Spoluriešiteľ projektu: 24.10.2023 .....

Obsah

1. Úvod

Ako kamaráti s veľkým zapálením pre hudbu sme už dlhšie hľadali alternatívu k mnohým komerčným riešeniam prehrávania hudby ako je Spotify alebo SoundCloud na Android zariadeniach. Chýbala nám možnosť, mať úplne internetovo nezávislú aplikáciu. Aplikácia, ktorá je plne závislá na statických mp3 súboroch. Áno, existuje viacero voľne dostupných aplikácií na Google Play, ktoré spĺňajú túto funkcionalitu, avšak my sme túžili aj po rýchlom a jednoduchom prenášaní hudby medzi zariadeniami. Existuje viacero riešení, avšak všetky fungujú mimo danej aplikácie prostredníctvom napríklad cloudových riešení, čo nie je najefektívnejšie a často sa spoliehate na úložisko tretej strany a dostupnosť skladieb na internete. Z toho dôvodu sme sa spojili a rozhodli vytvoriť aplikáciu Music Without Pain.

V aplikácii Music Without Pain sa používateľ stretne s mnohými už dobre známymi funkciami, ktoré musí každá správna aplikácia na prehrávanie hudby obsahovať. Pre používateľa disponuje aplikácia MWP viacerými možnosťami zobrazenia hudby ako hudobný zoznam, v ktorom sa nachádza všetka hudba na zariadení, no pre rýchlejšie prezeranie má používateľ dostupné vyhľadávanie a zoraďovanie. Skladby sa dajú prezerať aj pod jednotlivými autormi a albumami. Používateľ má k dispozícii vytváranie vlastných hudobných zoznamov. Čo sa týka prehrávania hudby, používateľ má počas celej doby používania aplikácie k dispozícii bočný prehrávač, kde si môže kontrolovať a riadiť stav prehrávania spolu s tlačidlami pre navigáciu aplikáciou. Mimo aplikácie túto úlohu zastupuje hudobná notifikácia a miniaplikácia widget. No tou najhlavnejšou funkcionalitou je automatická synchronizácia a prenos hudby po sieti medzi zariadeniami používateľa. Zabezpečujeme tak jednoduché počúvanie naprieč viacerými zariadeniami.

Na tvorbu všetkých vyššie spomenutých funkcionalít sú potrebné znalosti dizajnu a tvorby efektívneho používateľského prostredia, znalosti práce so súborovou štruktúrou v operačnom systéme mobilných zariadení Android, prenos dát po sieti a mnoho ďalších. No aj napriek veľkému množstvu rozdielnych úloh, ich rozdelenie nikdy nebol problém a práca prebiehala v symbióze, rýchle a efektívne. Lukáš Randuška ako frontend developer pracoval na dizajne a implementácii používateľského rozhrania a Adam Krokavec ako backend developer zabezpečil manažment súborov skladieb, prehrávanie hudby a prenášanie hudobných súborov po sieti.

2. Problematika a prehľad literatúry

2.1 Prehľad existujúcich riešení

Odpútanie sa od stream-ovacích platforiem je jedným z hlavných dôvodov prečo aplikácia MWP vznikla. Viacerým sa to môže zdať ako krok späť, keďže stream-ovacie platformy sú v dnešnej dobe veľmi atraktívne. Z pohľadu distribútora so záujmom zabezpečenia prehrávania hudby pre čo najväčšie percento ľudí, ktorým stačí iba zariadenie ako mobil alebo počítač schopné pripojiť sa na internet, je to veľmi lákavé riešenie, no pre bežného človeka to tak nemusí byť. Toto sú základné dôvody prečo:

- základným problémom je internet. Všetky stream-ovacie platformy si vyžadujú stabilné pripojenie na internet, ktoré nie vždy musí byť dostupné,
- hudba prenášaná prostredníctvom internetu je komprimovaná, a tak používateľ stráca maximálny úžitok z počúvania. Statické mp3 súbory zabezpečujú maximálnu kvalitu skladby,
- nedostupnosť istého interpreta alebo skladby,
- veľmi časté sú mesačné poplatky za využívanie všetkých alebo časti stream-ovacích služieb.

Najlepším príkladom aplikácie, pri ktorej sa používateľ stretáva so všetkými spomínanými problémami je Spotify. Spotify je asi najznámejšia stream-ovacia hudobná platforma na svete. Pre mnohých je to najoptimálnejšie riešenie, čo sa týka počúvania hudby, no nájdu sa ľudia ako my, ktorí spokojní nie sú. Aplikácia MWP vznikla za účelom vyriešenia vyššie spomínaných problémov a zachovania základnej funkcionality aplikácie Spotify, a to nekonkurenčnej synchronizácie obľúbenej hudby používateľa naprieč zariadeniami.

2.2 Použité technológie

2.2.1 Programovací jazyk C#

Programovací jazyk C# je vysokoúrovňový, objektovo orientovaný, platformovo nezávislý programovací jazyk vyvinutý spoločnosťou Microsoft. Tento programovací jazyk je mimoriadne podobný jazyku Java a C++, čo pre človeka so skúsenosťami v jednom z týchto jazykov znamená rýchlu adaptáciu na syntax a spôsob písania kódu, ktorý je takisto zväčša rovnaký. Jazyk C# je často používaný na vývoj webových aplikácií a služieb, cross-platform aplikácií prostredníctvom platformy .NET (dotnet) a mobilných aplikácií.

Základné vlastnosti jazyka C#:

- nepotrebuje a neobsahuje doprednú deklaráciu - poradie deklarácie metódy nie je dôležité,
- je to jediný komponentovo orientovaný jazyk, ktorý je dnes k dispozícii,
- jazyk C# podporuje unifikovaný typový systém, ktorý odstraňuje problém rôznych rozsahov celočíselných typov. Všetky typy sa považujú za objekty a vývojári môžu typový systém jednoducho a ľahko rozšíriť,
- jazyk C# je skutočne objektovo orientovaný. Podporuje všetky tri princípy objektovo orientovaných systémov, a to:
    - polymorfizmus,
    - enkapsuláciu,
    - dedenie.
- jazyk C# je typovo bezpečný:
    - všetky dynamicky alokované objekty a polia sú inicializované na nulu,
    - jazyk C# nepovoľuje nebezpečné pretypovanie,
    - jazyk C# zavádza kontrolu pretečenia pamäte pri aritmetických operáciách,
    - jazyk C# podporuje automatický garbage collection.

Jazyk C# nám prišiel ako vhodný jazyk pri práci na našom projekte hlavne z dôvodu, že celý tím je v tomto jazyku zbehlý a vlastnosti jazyka zabezpečuje, že naše programy budú efektívne, rýchle a optimalizované. Jazyk C# by sme ale určite nepoužili, pokiaľ by nebolo vývojovej platformy Xamarin.

2.2.2 Xamarin

Xamarin je open-source C# framework na tvorbu aplikácií. Xamarin poskytuje vývojárom širokú škálu nástrojov, ktoré možno použiť na vývoj multiplatformových mobilných aplikácií. Viacero framework-ov používa hneď niekoľko jazykov na tvorbu aplikácií, ako napríklad spojenie JavaScript a Html. Xamarin je iný, pretože ponúka jediný jazyk C# a runtime na všetko, čo potrebujete, ktorý funguje na troch mobilných platformách (Android, iOS a Windows). Prostredníctvom Xamarin-u vyvíjame mobilnú aplikáciu, ktorej vzhľad je úplne natívny.

Jednou z najzákladnejších výhod Xamarin-u pre nás, je možnosť využívať vlastnosti jazyka C# priamo pri tvorbe samotnej aplikácie. Xamarin nám tak isto dáva obrovský priestor na rast, keďže ako už bolo spomenuté, v rámci framework-u Xamarin môžeme okrem mobilných Android aplikácií programovať aj IOS či Windows aplikácie. To nám do budúcnosti umožňuje expandovať našu aplikáciu na viaceré platformy.

2.2.3 Figma

Figma je vo svojej podstate program na vektorové kreslenie. Využíva sa na dizajn a tvorbu všetkých druhov digitálnych vizuálnych prvkov aké poznáme. Je perfektný na tvorbu dizajnu

log, web stránok, mobilných aplikácií, počítačových aplikácií, instagramových príspevkov, prezentácií. Avšak okrem efektívneho a promptného prostredia na tvorbu dizajnu Figma ponúka funkciu kolaborácie, čo nám umožnilo v reálnom čase pracovať spolu a uvažovať na dizajne používateľského rozhrania spoločne. Figma je voľne dostupná aplikácia, na ktorej používanie netreba žiadny špeciálny počítač či operačný systém. Figma pracuje priamo v bežnom prehliadači ako je Google Chrome alebo Firefox. Nevyžaduje si veľa zdrojov počítača na prácu a funguje teda na väčšine zariadení. Takisto je aj založený na technológii Cloud, čo znamená, že všetky projekty sú uložené priamo do internetového úložiska automaticky. Prostredie aplikácie viď. Príloha A.1 .

2.2.4 Chromaprint

Je knižnica, ktorá vie vygenerovať odtlačok z mp3 súboru na základe algoritmu na mieru, ktorý následne používame so službou Acoustid na získanie ID3v2 tag-ov pre danú skladbu.

2.2.5 R8 code shrinker

R8 je celoprogramový optimalizačný kompilátor so zameraním na zmenšovanie veľkosti programov. R8 ponúka štyri funkcie určené na zmenšenie veľkosti aplikácie pre Android:

- používanie statickej analýzy kódu na vyhľadávanie a odstraňovanie nedosiahnuteľného kódu a neurčených typov,
- optimalizácia kódu z hľadiska veľkosti, a to odstránením mŕtveho kódu, selektívnym inline-ovaním, odstránením nepoužívaných argumentov a zlúčením tried,
- používanie krátkych názvov a squashing package namespace,
- kanonizácia informácií o ladení a komprimácia číslovania riadkov.

R8 code shrinker je potrebný hlavne z dôvodu používania knižníc tretej strany. Často sa v kóde používa iba malá časť knižnice, avšak pri tvorbe samotného spustiteľného súboru aplikácie sa zahŕňa kód celej knižnice, čo značne zväčšuje veľkosť, ktorú aplikácia zaberá na úložisku zariadenia.

2.2.6 JetBrains Rider

JetBrains Rider je integrované vývojové prostredie navrhnuté s ohľadom na moderného vývojára. Rider, vyvinutý spoločnosťou JetBrains, ktorá je známa výrobou vysokokvalitných IDE, je určený špeciálne pre vývojárov .NET a ponúka komplexný súbor nástrojov na vytváranie, ladenie a nasadzovanie aplikácií na rôznych platformách.

Pre nás ako vývojárov Android aplikácie a backend služieb je veľmi dôležité kvalitné a efektívne ladenie. Preto sme si vybrali JetBrains Rider, keďže poskytuje funkcie, ako je interaktívne ladenie, vizualizéry a integrácia unit testov, čo nám umožňuje rýchlo identifikovať a opraviť chyby.

Tento projekt je realizovaný v tíme ,čo pre nás vytvára potrebu jednoduchého zdieľania kódu a kontroly verzií. Kontrola verzií je pre spoločný vývoj kľúčová. Rider sa bezproblémovo integruje s populárnymi systémami na správu verzií, ako je Git, čo nám umožňuje spravovať zmeny zdrojového kódu bez toho, aby sme museli opustiť IDE.

Samozrejme okrem výhodných vlastností, JetBrains Rider poskytuje základnú funkcionalitu pre tvorbu Android aplikácií ako je Android Emulátor, ladenie Android aplikácií prostredníctvom WiFi siete alebo aj vizualizácia XML layout súborov. Pre názorné vyobrazenie rozhrania viď. Príloha A.2, Obrázok 3 Prostredie aplikácie JetBrains Rider IDE.

2.2.7 GitHub

GitHub je webová lokalita a cloudová služba, ktorá nám pomáha ukladať a upravovať náš kód, ako aj sledovať a kontrolovať zmeny v kóde. GitHub je založený na dvoch základných princípoch, a to:

- kontrola verzií,
- Git.

Kontrola verzií nám pomáha sledovať a spravovať zmeny v kóde softvérového projektu. Keď sa softvérový projekt rozrastá, kontrola verzií sa stáva nevyhnutnou. Kontrola verzií nám umožňuje bezpečne pracovať s vetvením a následným zlučovaním vetiev.

Git je špecifický open-source systém na správu verzií, ktorý vytvoril Linus Torvalds v roku 2005. Git je distribuovaný systém na správu verzií, čo znamená, že celá kódová základňa a história sú k dispozícii na každom počítači vývojára, čo umožňuje jednoduché vetvenie a zlučovanie. Keďže zdrojový kód Git-u je voľne dostupný, čo umožnilo mnohým spoločnostiam, ako je aj JetBrains integrovať systém Git do svojich produktov.

2.2.8 GitHub actions

GitHub actions sú skripty, pomocou ktorých ide automatizovať niektoré úlohy a majú rôzne spúšťače, ako napríklad push alebo merge. Samotné skripty sa vykonávajú na zariadeniach GitHub-u alebo na vašom zariadení, ak si to tak nastavíte. Takéto zariadenia sa nazývajú Runners. V našom projekte ich využívame na kompiláciu kódu v Master vetve a následne vydanie novej verzie .apk súboru na GitHub-e, čo nám umožňuje automatické aktualizácie bez potreby Google Play alebo iného poskytovateľa.

2.2.9 Android Isolated Storage

Izolované úložisko je mechanizmus, ktorý umožňuje aplikáciám ukladať údaje do oblasti súborového systému určenej pre používateľa alebo aplikáciu. Poskytuje bezpečnejší spôsob

ukladania údajov bez toho, aby ste museli určiť konkrétnu cestu alebo sa starať o prístupové oprávnenia.

Údaje sa uchovávajú oddelene pre každého používateľa a každú aplikáciu. To znamená, že k údajom má prístup len aplikácia, ktorá ich vytvorila, a len používateľ, ktorý tieto údaje vlastní. Jedno z možných využití Android Isolated Storage je, ak aplikácia potrebuje ukladať citlivé údaje (napríklad osobné údaje), izolované úložisko je bezpečnejšie ako bežné úložisko súborov.

2.2.10 RSA

Šifrovací algoritmus Rivest-Shamir-Adleman (RSA) je asymetrický šifrovací algoritmus, ktorý sa široko používa v mnohých produktoch a službách. Pri asymetrickom šifrovaní sa na šifrovanie a dešifrovanie údajov používa dvojica kľúčov, ktorá je matematicky prepojená. Vytvára sa súkromný a verejný kľúč, pričom verejný kľúč je prístupný komukoľvek a súkromný kľúč je tajomstvom, ktoré pozná len tvorca kľúčového páru. V prípade RSA môže súkromný alebo verejný kľúč zašifrovať údaje, zatiaľ čo druhý kľúč ich dešifruje. To je jeden z dôvodov, prečo je RSA najpoužívanejším asymetrickým šifrovacím algoritmom. Vývojový diagram si je možné pozrieť v Príloha A.3, Obrázok 4 Vývojový diagram šifrovacieho algoritmu RSA.

2.2.11 AES

Advanced Encryption Standard (AES) je špecifikácia na šifrovanie elektronických údajov, ktorú v roku 2001 vytvoril americký Národný inštitút pre štandardy a technológie (NIST). AES je dnes široko používaný, pretože je oveľa silnejší ako DES a triple DES, napriek tomu, že sa ťažšie implementuje.

Základné vlastnosti:

- AES je bloková šifra,
- veľkosť kľúča môže byť 128/192/256 bitov.
- šifruje údaje v blokoch po 128 bitov.

To znamená, že na vstupe je 128 bitov a na výstupe je 128 bitov zašifrovaného textu. AES sa spolieha na princíp substitučno-permutačnej siete, čo znamená, že sa vykonáva pomocou série prepojených operácií, ktoré zahŕňajú nahradenie a premiešanie vstupných údajov. Jedným z najčastejších využití AES je šifrovanie súborov a priečinkov v počítačoch, externých úložných zariadeniach a cloudových úložiskách. Chráni citlivé údaje uložené v zariadeniach alebo počas ich prenosu, aby sa zabránilo neoprávnenému prístupu. Vývojový diagram si je možné pozrieť v Príloha A.4, Obrázok 5 Vývojový diagram šifrovacieho algoritmu AES.

2.3 Finančná analýza

V rámci tvorby aplikácie Music Without Pain bol najdrahším výdavkom čas strávený tvorbou aplikácie. Na základe toho by sa dala aplikácia oceniť ako počet odpracovaných hodín krát minimálna hodinová mzda. Počet odpracovaných hodín Adama Krokavca je odhadovaný na 300 hodín. To isté platí aj pre Lukáša Randušku. Čo spolu činí 600 odpracovaných hodín krát zaokrúhlená minimálna hodinová mzda 5 EUR, a to vychádza na 3000 EUR, čo je aj celková cena vývoju aplikácie pri abstrahovaní platenia odvodov (nepočítame s celkovou cenou práce). Pri vývoji aplikácie bolo použitých viacero knižníc, avšak tie si nevyžadovali žiadne výdavky, keďže všetky použité knižnice, nástroje a zdrojové kódy tretích strán spadajú pod licencie typu Common Open Source Software Licenses ako je (1) GNU LGPL, (2) Apache License v2 alebo (3) The MIT License, čo znamená, že sú bezplatné pre nekomerčné účely. Jedinou výnimkou je integrované vývojové prostredie JetBrains Rider, ktoré je štandardne platený nástroj formou mesačného predplatného. JetBrains ako spoločnosť však ponúka 100%-nú zľavu na všetky produkty pre študentov a učiteľov všetkých stredných a vysokých škôl pod licenciou (4) Toolbox Subscription Agreement for Students and Teachers.

Distribuovaný finálny produkt aplikácie na platforme Google Play má dva možne scenáre. A to predaj aplikácie za účelom výnosu príjmu alebo distribuovanie aplikácie zadarmo pre verejnosť, bez akéhokoľvek zámeru za získaním zisku.

1. Distribúcia za účelom získania zisku:
    - Predikcia zisku na trhu v oblasti mobilných aplikácií bez akýchkoľvek prvotných hodnôt je náročná, no na základe priemerov podobných aplikácií je možné vypočítať predpokladaný zisk. Treba si zadať základné hodnoty, a to Conversion Rate, ktorý hovorí o percente používateľov, ktorí si kúpia aplikáciu, cenu za inštaláciu (CPI), ktorá hovorí o tom koľko zaplatíme za jedného používateľa, ktorý si stiahne našu aplikáciu a Lifetime value (LTV), čo je celkový predpokladaný zisk získaný od jedného používateľa.
    - Predpokladajme, že cena aplikácie na stiahnutie je 5 EUR a cena za propagáciu aplikácie je približne 1000 EUR. Pri CPI 1 EUR, náš 1000 EUR budget by nám priniesol 1000 potencionálnych kupcov, avšak pri predpokladanom Conversion Rate 1%, reálnych stiahnutí by bolo 10, z čoho vyplýva výnos 50 EUR pri náklade 1000 EUR za propagáciu a predpokladaných premenných na trhu Android aplikácií.
2. Distribúcia bez akéhokoľvek zámeru za získaním zisku:
    - Pokiaľ by sme nepočítali vývoj aplikácie ako zisk a obstarávacia cena aplikácie by taktiež bola nulová z dôvodu nevyužívania platených služieb tretích strán. Bez zámeru dosiahnutia zisku, by nebol teda potrebný ani marketing. Jediný výdavok predstavuje teda jednorazová platba 25 EUR za prvotnú publikáciu aplikácie na platformu Google Play.

3. Výsledky práce

3.1 Music Service

3.1.1 Používateľské rozhranie

V podkapitole používateľské rozhranie v rámci hudobných služieb sa bude jednať hlavne o elementy používateľského rozhrania slúžiaceho na sledovanie a ovládanie stavu prehrávania hudby. To sú:

- bočný prehrávač,
- widget,
- hudobná notifikácia.

3.1.2 Bočný prehrávač

Bočný prehrávač je hlavným prehrávačom aplikácie. Nachádza sa na bočnom paneli aplikácie spolu s hlavným ovládaním aplikácie prostredníctvom ktorého sa používateľ orientuje v rámci aplikácie. Bočný panel je používateľovi vždy dostupný naprieč celou aplikáciou. Používateľ má tak neustále prístup k stavu prehrávania. Bočný panel je vytvorený prostredníctvom NavigationView komponentu v XML súbore jedinej a hlavnej aktivite aplikácie. Samotné rozhranie je zabalené do vertikálneho LinearLayout komponentu v rámci NavigationView komponentu.

Používateľské rozhranie bočného prehrávača z hľadiska XML je ImageView element slúžiaci na zobrazenie obrázku skladby. Pod ImageView sa nachádzajú 3 TextView elementy slúžiace na vyobrazenie názvu skladby, interpreta a albumu. Nasleduje LinearLayout komponent s horizontálnym usporiadaním komponentov pre tlačidlá slúžiace na ovládanie prehrávania skladieb. Samotné tlačidlá sú vytvárané plne programovo z dôvodu menenia ich dizajnu počas behu programu vzhľadom k stavu prehrávania. Medzi tieto tlačidlá patrí: štart/stop, dopredu, dozadu, náhodné prehrávanie a opakovanie. Tlačidlo opakovanie má 3 stavy a to: žiadne opakovanie, opakovanie zoznamu a opakovanie jednej skladby. Posledný komponent prehrávača je SeekBar, ktorý je takisto riešený cez LinearLayout s horizontálnym usporiadaním, v ktorom sa nachádzajú dva TextView komponenty pre signalizáciu zbehnutého času a zostávajúceho času, medzi ktorými sa nachádza SeekBar prostredníctvom ktorého používateľ môže preskakovať pasáže a sledovať priebeh skladby. Stav prehrávania skladby ako aj jej názov, autor a album sú získavané prostredníctvom dopytov na backend premenné a funkcie, čo zabezpečuje zmenu dizajnu tlačidiel či zmenu obsahu TextView elementov v prípade zmeny v prehrávaní. Takisto aj zmeny vytvorené používateľom prostredníctvom

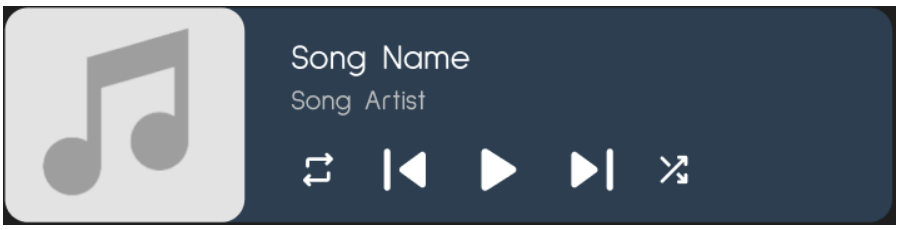
bočného prehrávača vytvárajú dopyty na backend, čím sa zabezpečí zmena stavu v rámci celého zariadenia. Pre vizualizáciu opísaného rozhrania viď. Príloha B.1, Obrázok 6 Dizajn a implementácia používateľského rozhrania bočného prehrávača.

Zvyšné tlačidlá, nachádzajúce sa pod prehrávačom, slúžia k navigácii medzi jednotlivými fragmentmi aplikácie.

3.1.3 Widget

Widget je miniatúrna aplikácia, ktorá môže byť vložená napríklad na domácu obrazovku zariadenia používateľa. V rámci aplikácie MWP widget slúži na sledovanie a riadenie stavu prehrávania hudobnej skladby.

Základom tvorby miniatúrnej aplikácie widget je vytvorenie triedy, ktorá dedí triedu AppWidgetProvider. Následne je potrebná implementácia viacerých metód, ktoré budú volané na pozadí pri vytváraní Widget-u. Ak teda vytvoríme triedu dediacu AppWidgetProvider, musíme tejto triede zadefinovať metadáta, ako nový atribút prostredníctvom triedy MetaDataAttribute, ktorý prijíma ako argumenty meno a zdroj. Meno v našom prípade je "android.appwidget.provider" a zdroj bude relatívna cesta odkazujúca na XML súbor udávajúci základné nastavenia Widget-u, ako čas oneskorenia aktualizácií (v našom prípade 1000 milisekúnd), výšku, šírku, úvodný layout a náhľadový obrázok, ktorý je viditeľný pri výbere Widget-u zo zoznamu. Ďalej prichádza implementácia jednotlivých metód zdedených z triedy AppWidgetProvider. V metóde OnUpdate zabezpečujeme, aby sa Widget pravidelne aktualizoval a prijímal tak zmeny od pozadia aplikácie. Prijíma ako argument metódu BuildRemoteViews volanú v prípade aktualizácie. V metóde BuildRemoteViews inicializujeme vzhľad Widget-u. Vytvárame si objekt typu RemoteViews, ktorý preberá ako argument XML layout Widget-u. Prostredníctvom objektu typu RemoteViews máme prístup k metódam na komunikáciu s používateľským rozhraním, to znamená možnosť ho upravovať a meniť. Rozhranie, s ktorým pracujeme, je potrebné pre vytvorenie používateľského rozhrania. Skladba je jednoznačná, základný layout je typu LinearLayout s horizontálnou orientáciou elementov. V layout-e sa nachádza ImageView slúžiaci na zobrazenie náhľadového obrázka prehrávanej skladby. TextView komponenty pre zobrazenie názvu skladby a meno autora, pod ktorými sa v LinearLayout komponente nachádza v riadku usporiadané tlačidlá slúžiace na riadenie prehrávania skladby. To je tlačidlo pauza/štart, predošlá skladba, nasledujúca skladba, zamiešanie a opakovanie.



Obrázok 1 používateľské rozhranie miniaplikácie widget

V metóde BuildRemoteViews pristupujeme k týmto elementom prostredníctvom objektu typu RemoteViews. Keďže Widget je aplikácia sama o sebe žijúca mimo hlavnej aplikácie, je tak potrebné pre akékoľvek zmeny pristupovať k metódam z pozadia aplikácie a s hlavnou aplikáciou komunikovať prostredníctvom metód na pozadí aplikácie. Vďaka RemoteViews tak máme prístup k týmto metódam a upravujeme komponenty rozhrania prostredníctvom nich. V metóde BuildRemoteViews teda upravujeme TextView elementy pre meno autora a názov skladby na základe stavu získaného z pozadia aplikácie. Vytvárame PendingIntent-y pre každé tlačidlo, ktoré slúžia na detekciu ich stlačenia. Nastavujeme náhľadový obrázok získavaný z pozadia aplikácie na základe stavu prehrávania skladby. Nakoniec vytvárame PendingIntent slúžiaci na otvorenie aplikácie v prípade, ak používateľ klikne na Widget bez zámeru kliknutia na jedno z tlačidiel. Poslednou implementovanou metódou je OnReceive, slúžiaca na odpovedania na rôzne akcie. V našom prípade to je prijímanie Intent-u, ktorý nesie správu o tom, ktoré tlačidlo bolo stlačené. Na základe akcie získanej z intent-u, ktorý bol pridelený pri jeho vytvorení, sa spúšťajú prislúchajúce metódy na pozadí aplikácie na upravenie daného stavu prehrávania skladby.

3.1.4 Hudobná notifikácia

Hudobná notifikácia umožňuje používateľovi sledovanie a kontrolu stavu prehrávania skladby počas celej doby používania zariadenia, bez potreby otvorenej aplikácie na obrazovke zariadenia. Notifikácia je aktívna aj počas neaktivity používateľa v aplikácií alebo v čase, ak je zariadenie zamknuté, jedinou podmienkou je potreba zapnutej aplikácie na pozadí zariadenia.

Samotná notifikácia si nevyžaduje vlastnú implementáciu rozhrania, je implementovaná prostredníctvom už vytvorených objektov Notification, NotificationCompat.Builder a NotificationManagerCompat a využíva štýl priamo zabudovaný natívne do Android API. To znamená, že dizajn aplikácie sa líši na základe verzie zariadenia používateľa. Tvorba notifikácie začína pri vytvorení notifikačného kanálu. Notifikačné kanály slúžia na triedenie rôznych typov notifikácií v prípade, ak ich aplikácia má viacero. Tieto kanály sa neskôr používajú na riadenie zapnutia a vypnutia jednotlivých notifikácií, a keďže každá z nich má kanál, dajú sa odlíšiť. Kanál vytvárame pomocou triedy NotificationChannel. Do konštruktora vkladáme základné

informácie ako názov kanálu, ID kanálu a vlajku pre indikáciu dôležitosti notifikácie. Pre nás to je NotificationImportance.Low, čiže nízka. Následne pomocou manažéra notifikácií vytvárame kanál. Po vytvorení kanála, prichádza na rad tvorba samotnej notifikácie pomocou triedy Builder, ktorá pri vytváraní objektu preberá do konštruktora ID kanála, na ktorý sa má naviazať. Následne pomocou priamo zabudovaných metód z objektu typu Builder nastavujeme ikonu notifikácie, PendingIntent slúžiaci na otvorenie aplikácie v prípade kliknutia a v neposlednom rade štýl - najdôležitejšie nastavenie. Vďaka štýlu MediaStyle získavame natívnu notifikáciu prispôsobenú na prehrávanie hudobných súborov. Vytvorenie notifikácie typu MediaStyle si vyžaduje MediaSession token, ktorý je vytvorený na pozadí v MediaService triede slúžiacej na riadenie prehrávania skladieb. Okrem iného, prostredníctvom metódy AddAction z triedy Builder pridávame notifikácii tlačidlá slúžiace na ovládanie prehrávania skladieb ako pauza/štart, predošlá skladba, nasledujúca skladba, zamiešanie a opakovanie. Všetky sú vytvárané pomocou PendingIntent, čo zabezpečuje po kliknutí na jednotlivé tlačidlo spustenie príslušnej metódy na pozadí, ktoré aktualizujú stav zodpovedajúcim spôsobom. Samotné tlačidlá, v podobe akcií, sú pridávané do notifikácie dynamicky, na základe stavu prehrávania. Napríklad, ak je nastavené opakovanie aktuálnej skladby, tlačidlá dopredu a dozadu nebudú dostupné. Nakoniec samotnú notifikáciu "postavíme" pomocou metódy Build a následne zavoláme metódu Notify z objektu typu NotificationManagerCompat, ktorá prijíma ako argument "postavenú" notifikáciu a ID notifikácie.

3.1.5 Backend

Music service je druhá najkomplexnejšia funkcionalita hneď po Network Manager-ovi. Music service pozostáva z troch zložiek a to Queue objektu, music manažéra a samotného service-u.

3.1.5.1 Queue Object

Keďže sa k poradiu na prehratie pristupuje z viacerých miest a hlavne vlákien naraz, čo by mohlo spôsobovať rôzne problémy, ako napríklad Shared Access Exception a Null Reference Exception, vznikol Queue Object. Táto trieda v podstate len udržuje index aktuálne prehrávanej skladby, zoznam skladieb v poradí, stav zamiešania a opakovania a zaručuje synchronizovaný prístup k nim.

3.1.5.2 Service

Music manažér bol pôvodne jediný objekt v MainActivity. Časom bolo treba, aby bol sprístupnený vo všetkých aktivitách a fragmentoch, a to primárne kvôli bočnému prehrávaču. A tak začala referencia na tento objekt cestovať medzi aktivitami ako pointer. Problém nastal, keď používateľ minimalizoval aplikáciu, a tak zanikli aktivity a tým pádom zanikli všetky

referencie na Music manažéra, čoho dôsledkom zanikol aj on. Museli sme teda vložiť Music manažéra do service-u, aby si referenciu na neho držal samotný Android. Toto zaručuje, že kým service niečo vykonáva (hrá hudbu) nezanikne. Implementácia je štandardná cez binder.

3.1.5.3 Music Manager

Srdce našej aplikácie. Okrem hlavného komponentu, patria sem aj BroadcastReceiver a SessionCallback.

Broadcasting je jednoduchý spôsob, ako posielať veľmi obmedzené správy medzi modulmi aplikácie alebo aplikáciami. Broadcasting používame na zaznamenávanie odpojenia slúchadiel a na zachytávanie pending intent-ov z Widget-u a notifikácie, keďže tie nepodporujú binder implementáciu service-u.

SessionCallback alebo presnejšie MediaSessionCompat.Callback je kolekcia override-ov systémových funkcií, ktoré môže systém zavolať na kontrolu playback-u. Využíva sa tam, kde sme tlačidlá negenerovali my, ale systém, takže v Android Auto a pri verzii Android 13 a vyššej aj v notifikácii, kde sa dajú generovať tlačidlá na základe custom actions a systémovej triede PlaybackState.

V hlavnom komponente sa odohráva celá mágia. Na začiatok nainicializujeme MediaPlayer so všetkými jeho callback-mi, ktoré využívame. Potom si nainicializujeme AudioManager, cez ktorý si neskôr žiadame AudioFocus. Ďalej si nainicializujeme MediaSessionCompat s naším MediaSessionCallback-om a nastavíme, aké rôzne funkcie podporuje naša MediaSession.

Session je spôsob Androidu, ako zdieľať do systému rôzne údaje o schopnostiach a stave prehrávania v mediálnych aplikáciách. Najvýznamnejšie sú metadáta o aktuálne prehrávanom súbore, ktoré sa využívajú v notifikácii a v Android Auto. Ďalšie dôležité parametre sú PlaybackState, vyjadrujúci, v akom stave je prehrávanie, napríklad či je pozastavené a AvailableActions, hovoriaci, aké akcie môže systém zavolať na našom Callback objekte.

Nakoniec si vytvoríme náš AudioFocus request a nainicializujeme notifikáciu s predvolenými hodnotami. Ďalej už len čakáme na požiadavky o vytvorenie poradia alebo spustenie prehrávania. Pri spustení prehrávania najskôr skontrolujeme, či už je niečo v poradí a ak nie, vytvoríme nové poradie so všetkými skladbami. Ďalej si vyžiadame AudioFocus, čo je spôsob Android-u ako zabezpečiť, aby nehralo viacero aplikácií cez seba. Výhodou registrácie focus-u je to, že vieme, či máme nárok na prehrávanie a zároveň sme upozornení, kedy by sme mali zastaviť prehrávanie, napríklad pri prichádzajúcom hovore. Nakoniec skontrolujeme, či súbor, ktorý ideme prehrávať, stále existuje na zariadení a prehrajeme ho.

3.2 Android Auto

Implementácia podpory pre Android Auto je veľmi jednoduchá, keďže celé používateľské rozhranie vytvára Android za nás. Treba vytvoriť triedu, ktorá bude derivovaná zo systémovej triedy MediaBrowserServiceCompat, override-núť v nej metódy OnGetRoot a OnLoadChildren a vytvoriť triedu, ktorá bude derivovaná zo systémovej triedy MediaSessionCompat.Callback (viď kapitola Music Service) kvôli funkčnosti prehrávania a ovládacích prvkov prehrávania.

OnGetRoot v podstate len overuje, či aplikácia, ktorá si žiada prístup, má na to práva. Ktoré aplikácie povolíte, je len na Vás. Ak aplikácia má práva, vracia sa BrowserRoot objekt inicializovaný s Root ID.

OnLoadChildren je funkcia, ktorá vracia všetky MediaItem-y patriace pre isté ID. ID je vždy reťazec. Prvé volanie tejto funkcie bude vždy s Root ID, s ktorým sme nainicializovali BrowserRoot. Každý MediaItem objekt ma vlastné ID a zvyšné volania tejto funkcie budú s jedným z týchto ID.

MediaItem sa delia na Browsable a Playable. Browsable objekty obsahujú ďalšie objekty. Playable objekty môžu byť prehrávané. Android dokumentácia síce tvrdí, že jeden objekt môže byť naraz aj Browsable aj Playable (napríklad album), ale na základe nášho testovania, internetových fór a implementácie podpory pre Android Auto od Spotify, toto nie je pravdivé a keď je objekt Browsable tak Playable flag je vždy odignorovaný.

Na vygenerovanie MediaItem-u potrebujeme MediaDescriptionCompat a jeden z vyššie spomínaných flag-ov.

Na vygenerovanie MediaDescriptionCompat sa používa builder s viacerými funkciami. Využívame z nich tieto:

- SetMediaId - nastaví ID,
- SetTitle - nastaví reťazec, ktorý sa bude zobrazovať ako meno objektu,
- SetSubtitle - nastaví reťazec, ktorý sa bude zobrazovať pod menom objektu a menším fontom, využívame ho na zobrazenie mena autora skladby/albumu,
- SetIconBitmap - nastaví obrázok, ktorý sa má zobrazovať s objektom.

Nepovinným krokom je vložiť do MediaSession (viď kapitola Music Service) MediaSessionCompat.QueueItem-y, ktoré sú v poradí na prehratie. Na vytvorenie tohoto objektu potrebujeme MediaDescriptionCompat a ID. Toto ID je typu long a vyjadruje iba, ktorý v poradí je daný objekt na prehratie.

3.3 Reprezentácia skladieb v pamäti

3.3.1 Používateľské rozhranie

V rámci aplikácie MWP si používateľ môže prehľadávať skladby uložené v rámci zariadenia viacerými spôsobmi, a to prostredníctvom kolónky Songs, kolónky Albums a kolónky Playlists, ktoré sú dostupné naprieč celou aplikáciou a počas celej doby používania aplikácie v bočnom panely pod prehrávačom.

3.3.1.1 Songs

Pod kolónkou Songs sa používateľovi načítavajú všetky skladby nainštalované v rámci zariadenia. Zoradenie načítania skladieb je zo začiatku od najnovších po najstaršie. Používateľ však má v rámci kolónky Songs prístup k vyhľadávaciemu panelu, pod ktorým sa nachádzajú aj možnosti zoradenia skladieb, a to abecedne vzostupne, abecedne zostupne, od najstarších po najnovšie a od najnovších po najstaršie. Vyhľadávací panel slúži na vyhľadávanie priamo podľa špecifického textu zadaného používateľom. Vyhľadávanie nefunguje len na základe presného názvu, funguje aj s čiastočnými názvami, čo znamená, že používateľ nemusí zadať plný názov skladby a nájdu sa všetky výsledky vyhovujúce zadanému textu.

Songs sa skladá zo statickej časti a dynamickej časti. Statická časť sú EditText komponent a tlačidlá zoradenia. Tie sú staticky definované v XML súbore. Dynamickú časť tvoria samotné skladby. Používateľské rozhranie skladieb je vytvárané plne programovo. Ako prvé je vytvorený ScrollView komponent, v ktorom sa nachádza obalový LinearLayout s vertikálnou orientáciou, keďže ScrollView môže obsahovať iba jeden komponent. Do obalového LinearLayout sú následne vykresľované jednotlivé skladby prostredníctvom programovo vytvorených LinearLayout-ov s ImageView komponentom pre náhľadový obrázok skladby a TextView komponentom pre názov skladby. Pri stlačení políčka skladby, sa zavolá príslušná metóda z pozadia aplikácie, ktorá spustí príslušnú skladbu. Pri dlhom podržaní sa otvorí dialóg s možnosťami úpravy jednotlivej skladby, to je vymazanie, pridanie do poradia prehrávania, pridanie do playlist-u a úprava metadát. Pre bližšie vyobrazenie rozhrania viď. Príloha B.2, Obrázok 7 Dizajn a funkcionálny diagram používateľského rozhrania Songs.

Pre optimalizáciu rýchlosti načítania rozhrania pri veľkom počte skladieb sme využili praktiku s menom Lazy Loading. Pri väčšom počte skladieb sa načítavanie jednotlivých obrázkov skladieb stáva príliš pomalým, čo spomaľuje aplikáciu. Aplikácia v danom momentu využíva všetky zdroje zariadenia a tak spôsobuje nepoužiteľnosť aplikácie na istú dobu. To sme však vyriešili “spomaleným” načítaním obrázkov. Obrázky sa pri načítavaní rozhrania začnú načítavať na rozdielnom vlákne ako je hlavné vlákno používateľského rozhrania, čo umožňuje

plynulé a rýchle načítanie políčok pre jednotlivé skladby. Akonáhle sa na pozadí načíta obrázok do pamäte, presúva sa na vlákno používateľského prostredia a načíta sa do príslušného políčka skladby.

3.3.1.2 Albums

V kolónke albumy sú jednotlivé skladby zoradené podľa názvov albumov a interpretov. Pre každú skladbu sú vytvárané políčka daného interpreta a albumu. Ak už tieto políčka existujú, skladba je automaticky zoradená pod príslušného interpreta a album. Kolónka Albums sa skladá z dvoch ScrollView, jeden pre políčka albumov a druhý pre políčka interpretov. Každé políčko je generované plne programovo na základe dát z pozadia aplikácie. Vykresľovanie políčok je z dôvodu optimalizácie riešené prostredníctvom Lazy Loading-u, tak ako aj pri kolónke Songs.

Po kliknutí na jedno z políčok pod kolónkou interpreti, je používateľ presunutý do nového fragmentu, kde sa nachádzajú albumy prislúchajúce danému interpretovi a skladby, ktoré nie sú kategorizované do žiadneho albumu, ale majú známeho interpreta. Po kliknutí na políčko pod kolónkou albumy je používateľ presunutý do nového fragmentu, kde sa postupne načítajú jednotlivé skladby prislúchajúce k danému albumu. Vykresľovanie je realizované rovnakým spôsobom ako pri kolónke Songs prostredníctvom Lazy Loading-u. Pre vyobrazenie dizajnu albumov viď. Príloha B.3, Obrázok 8 Dizajn a funkcionálny diagram používateľského rozhrania Albums.

3.3.1.3 Playlists

V kolónke Playlists má používateľ možnosť prehľadávať a vytvárať si vlastné zoznamy skladieb. Pri kliknutí na kolónku Playlists v bočnom prehrávači je používateľ presunutý do nového fragmentu. V pravom dolnom rohu sa nachádza FloatingActionButton, vďaka ktorému sa po kliknutí používateľovi zobrazí dialógové okno s EditText komponentom pre zadanie názvu nového zoznamu. Po kliknutí na tlačidlo “Submit” sa vytvorí nový zoznam a načíta sa znova fragment z dôvodu načítania novo pridaného zoznamu. Zoznam sa pridá medzi políčka s ostatnými zoznamami, ktoré sa nachádzajú v ScrollView komponente. Všetky informácie o zoznamoch (počet skladieb a dané skladby) sa ukladajú do pozadia aplikácie, čo znamená, že vykresľovanie zoznamov je plne programové na základe dát z pozadia. Po kliknutí na jeden zo zoznamov, je používateľ presunutý do nového fragmentu, kde sa mu následne načítajú všetky skladby, ktoré si používateľ manuálne pridal do daného zoznamu. Pre zobrazenie opísaného rozhrania viď. Príloha B.4, Obrázok 9 Dizajn a funkcionálny diagram rozhrania Playlists.

3.3.2 Backend

Všetky skladby sa držia v pamäti pre jednoduchý a rýchly prístup. Samozrejme sa nedržia celé skladby, ale len niektoré ich parametre. Tieto parametre sa držia v našich objektoch a získavajú sa pri kroku indexovania, ktorý sa spúšťa pri zapnutí aplikácie.

Indexovanie nájde všetky mp3 súbory na zariadení a získa o nich rôzne informácie, uloží ich do pamäte a pokúsi sa nájsť ich metadáta na internete, ak to používateľ povolil.

Základný blok štruktúry je MusicBaseClass. Táto trieda udržiava nasledujúce položky:

- meno uloženého objektu,
- UUID - je generované vždy nové a používa sa na hľadanie objektu medzi modulmi aplikácie kvôli limitáciám Android API,
- cache-nutý MediaDescriptionCompat Builder (viď kapitola Android Auto),
- cache-nutý MediaDescriptionCompat (viď kapitola Android Auto).

Funguje aj ako interface, keďže udáva nutnosť implementácie metód a properties pre vyššie spomenuté položky, na získavanie obrázku a rôzne typovacie operácie ako napríklad konverzia objektu na MediaItem (viď kapitola Android Auto).

Druhá základná trieda je MusicBaseClassStatic, ktorá udržiava statické položky pre zmenšenie memory footprint-u aplikácie ako napríklad image placeholder.

Tretia a posledná základná trieda je MusicBaseContainer, derivovaná z triedy MusicBaseClass a pridáva len nutnosť objektu implementovať zoznam skladieb, ktoré obsahuje. Používa sa len na jednoduché roztriedenie objektov umelca/albumu od objektov samotných skladieb.

Triedy, ku ktorým pristupujeme v samotnom kóde sú Song, Album a Artist. Album a Artist triedy sú odvodené z triedy MusicBaseContainer a trieda Song je odvodená z triedy MusicBaseClass. Tieto triedy už implementujú vyššie spomenuté metódy. Taktiež implementujú základné metódy ako Equals, GetHashCode a ToString. Okrem toho, každá implementuje niekoľko konštruktorov, ako aj statické metódy na zjednodušenie pracovania s nimi, napríklad vyhľadávanie špecifického objektu pomocou UUID.

Udržiavané položky (okrem už vyššie spomenutých) sú napríklad tiež:

- pre Song cesta na samotný mp3 súbor a zoznam všetkých albumov (Album objekty) a umelcov (Artist objekty), ku ktorým patrí na základe ID3v2 tag-ov,
- pre Artist cesta na obrázok umelca a zoznam všetkých albumov (Album objekty) a skladieb (Song objekty), ktoré k nemu patria,
- pre Album cesta na obrázok albumu a zoznam všetkých umelcov (Artist objekty), ku ktorým patrí a skladieb (Song objekty), ktoré k nemu patria.

Toto nám dovoľuje jednoducho prechádzať celú štruktúru, jasne priradzovať spojitosti objektov a jednoducho ich následne používať.

3.4 Tag Manager

3.4.1 Používateľské prostredie

Používateľ je schopný pristúpiť k úprave metadát istej skladby prostredníctvom dlhého kliknutia na skladbu, čo otvorí popup s možnosťami pre prácu s danou skladbou, medzi ktorými je aj tlačidlo edit, ktoré slúži na presmerovanie používateľa do nového fragmentu.

V rámci Tag manažéra je možné upravovať názov skladby, meno interpreta, názov albumu, v ktorom sa nachádza skladba a miniatúru skladby. To nám hovorí o tom, aké elementy rozhrania bude musieť obsahovať. Najrozumnejším výberom budú 3 individuálne EditText elementy, ktorými zabezpečíme úpravu názvov a jeden ImageView, v ktorom nie len že ukážeme miniatúru skladby, ale bude na ňu možné aj kliknúť, čo otvorí dialóg pre výber nového obrázka pre miniatúru. V rámci dialógu bude používateľ vyberať spomedzi obrázkov na jeho zariadení. Ak nastane ľubovoľná zmena v jednom z EditText elementov, alebo bude zmenený obrázok, v dolnom pravom rohu sa automaticky ukáže tlačidlo pre možnosť uloženia zmien. Po kliknutí na tlačidlo "uložiť", sa pred používateľom objaví popup s kontrolnou otázkou, či naozaj chce zmeny uložiť. Na výber bude mať tlačidlá "áno" alebo "nie". Po stlačení na tlačidlo "áno" sa zmeny na pozadí uložia a automaticky sa aplikujú do zoznamu skladieb v reálnom čase. To si používateľ bude môcť hneď skontrolovať, keďže bude po uložení presmerovaný naspäť do zoznamu skladieb. Pre zobrazenie opísaného rozhrania viď. Príloha B.5, Obrázok 10 Dizajn a funkcionálny diagram používateľského rozhrania Tag Manager.

3.4.2 Backend

Tag Manger je len vrstva abstrakcie nad už aj tak veľmi zabstraktneným prepisom ID3v2 tag-ov, aby nebežal veľký rozsah kódu vo fragmentoch a na UI vlákne. Tag Manager zodpovedá za správu pamäte, potrebnej na prepis tag-ov a jej následne uvoľnenie. Ďalej si udržiava pôvodné hodnoty tag-ov a kontroluje, ktoré sa zmenili, ak vôbec, aby sa predišlo zbytočným prepisom.

3.5 Settings Manager

3.5.1 Používateľské rozhranie

Po kliknutí na tlačidlo Settings v bočnom paneli, je používateľ presunutý do nového fragmentu. V rámci nastavení si používateľ môže zmeniť základné nastavenia napríklad ako povolenie používať sieť, povolenie sledovania aktualizácií a podobne. Používateľské rozhranie nastavení je generované plne programovo na základe dát z pozadia. Pri načítaní fragmentu sa ako prvé

načítajú dáta nastavení z pozadia, to je typ nastavenia, názov nastavenia, funkcia na zapísanie nových dát a funkcia na prečítanie posledného uloženého stavu nastavenia. Rozlišujeme 4 typy nastavení:

- bool, inak povedané Switch, je nastavenie na menenie stavu z 0 na 1 alebo opačne,
- int, alebo tiež Dropdown, je nastavenie, pri ktorom meníme viacero hodnôt ako je 0, 1, 2, atď.… a ku každému číslu prislúcha istý názov nastavenia. To sa realizuje pomocou komponentu Spinner typu dropdown,
- folder picker, nachádza sa v nastaveniach iba raz a je definovaný v XML súbore staticky. Používateľ sa k nemu dostane prostredníctvom nastavenia "Restricted locations for mp3 searching" a po kliknutí na príslušné tlačidlo sa otvorí dialóg so zoznamom ciest a tlačidlami "add new" pre pridanie novej cesty prostredníctvom Folder picker-a a cancel pre návrat do fragmentu. Každý riadok v zozname obsahuje tlačidlo na zmazanie riadka,
- string, reprezentované komponentom EditText, čo znamená, že sa jedná o nastavenie pre zadávanie textu.

Dizajn užívateľského rozhrania je znázornený v Príloha B.6, Obrázok 11 Dizajn a funkcionálny diagram používateľského rozhrania nastavení.

3.5.2 Backend

Na riešenie nastavení používame Xamarin.Essentials.Preferences, čo znamená že samotné nastavenia uchováva operačný systém. Problém s týmto prístupom je ten, že každé prečítanie alebo zapísanie hodnoty jedného nastavenia, si vyžaduje poznať meno "priečinka", v ktorom sa nachádza nastavenie (share name) a meno samotného nastavenia (key). Udržiavať identické hodnoty medzi nespočetnými zápismi a čítaniami v kóde je veľmi náročné, a tak vznikol Settings Manager.

Základný blok je Setting<T>, ktorý udržiava hodnoty share name a key. Z neho sú odvodené triedy StringSetting, BoolSetting a IntSetting, kvôli limitáciám jazyka, keďže C# je staticky typovaný jazyk. Tieto triedy už poskytujú prístup k samotným hodnotám nastavení. Teraz prichádza na scénu statická trieda SettingsManager, ktorá združuje všetky inštancie Setting tried na jedno miesto.

Kvôli zabráneniu prepisovania inštancií Setting tried a kvôli ďalšiemu zjednodušeniu používania SettingsManager triedy je prístup k Setting triedam umožnený iba cez property a to cez getter-y a setter-y. To nám umožňuje používať enumerátory ako hodnoty pre nastavenia. Celková implementácia nám zabraňuje dopustiť sa prípadných preklepov v share name a key parametroch, keďže ich zadávame iba raz a všetky ostatné preklepy sú odhaliteľné pomocou statických analyzátorov kódu, pretože sa už jedná o premenné. Je to najjednoduchší spôsob na vyriešenie problematiky robustne.

3.6 Network Manager

3.6.1 Používateľské rozhranie

Po kliknutí na tlačidlo Share v bočnom paneli je používateľ presunutý do nového Fragmentu. V rámci používateľského rozhrania Share si používateľ môže upravovať nastavenia vzdialených pripojení. Väčšina komponentov je definovaných staticky v príslušnom súbore XML. Koreňom fragmentu je LinearLayout, v ktorom sa nachádza viacero TextView elementov označujúce dané nastavenia. Na začiatku fragmentu má používateľ dostupný EditText slúžiaci na zmenu čísla portu pre vzdialené pripojenia. Ďalší v poradí je ScrollView zoznam so všetkými pridanými dôveryhodnými sieťami. ScrollView zoznam je generovaný plne programovo na základe dát z pozadia aplikácie. Pridať novú sieť si môže používateľ prostredníctvom tlačidla nachádzajúceho sa nad daným zoznamom. Po kliknutí sa zobrazí upozorňovací dialóg. V rámci každého políčka siete sa nachádza tlačidlo na jeho odstránenie. Pod zoznamom sietí sa nachádza ScrollView zoznam dostupných zariadení, na ktoré je možné sa pripojiť. Zoznam je generovaný programovo na základe dát z pozadia aplikácie. V rámci každého dostupného zariadenia je tlačidlo na pridanie zariadenia do zoznamu, ak už je pridané, tak sa tlačidlo mení na odstránenie. Po kliknutí na tlačidlo, sa zobrazí upozorňovací dialóg. Rozhranie je si možné prehliadnuť v Príloha B.7 Obrázok 12 Dizajn a funkcionálny diagram používateľského rozhrania pre vzdialené zdieľanie.

V pravom dolnom rohu rozhrania sa nachádza FloatingActionButton (skrátene FAB), slúžiaci na výber jednotlivých skladieb, ktoré chce používateľ poslať zariadeniu pripojenému do rovnakej siete ako zariadenie používateľa. Po kliknutí na FAB je používateľ presunutý do nového fragmentu. Vo fragmente sa používateľovi zobrazí list dostupných host-ov, ktorým je možné odoslať skladby. Po kliknutí na jedného z host-ov, sa daný host vyznačí oranžovou farbou a objavia sa tlačidlá na pokračovanie alebo na zrušenie výberu a vrátenie späť. Vybrať sa smie naraz iba jeden host. Po kliknutí na tlačidlo „pokračovať“, je používateľ presunutý do nového fragmentu. Na obrazovke sa zobrazí list so všetkými skladbami v rámci zariadenia. Používateľ si môže vybrať viacero skladieb naraz, a to kliknutím na ich príslušné políčka, ktoré zmenia farbu po zakliknutí pre signalizáciu výberu. Pokiaľ je zakliknutá aspoň jedna skladba, zobrazí sa posuvné tlačidlo na potvrdenie odoslania skladieb. Používateľ musí potiahnuť tlačidlo zľava doprava pre konfirmáciu. Tlačidlo následne zmení farbu na zelenú pre indikáciu úspešného zadania požiadavky na odoslanie skladieb a používateľ je vrátený naspäť do Share fragmentu. Pre názorné vyobrazenie dizajnu a procesu výberu skladieb viď. Príloha B.7 Obrázok 13 Dizajn a funkcionálny diagram používateľského rozhrania posielania skladieb.

3.6.2 Zhrnutie

Aplikácii sme napísali vlastný protokol aplikačnej vrstvy, ktorý sa spolieha na protokoly transportnej (TCP a UDP) a protokol internetovej vrstvy (IP). Transportná vrstva je využívaná vo všeobecnosti kvôli jej charakteristickej schopnosti deliť premávku pomocou portov, a tak umožňovať komunikovať viacerým aplikáciám naraz, dokonca aj s viacerými počítačmi naraz. Protokol TCP používame vo väčšej miere ako protokol UDP, kvôli jeho schopnosti skladať packet-y do správneho poradia a preposielať stratené packet-y. Protokol UDP využívame pre jeho absenciu three way handshake-u v situáciách, kedy ešte nie je rozhodnuté, kto bude server a kto bude klient, a tak three way handshake nie je možný. Je to hlavne z dôvodu, že aplikácia je plne peer-to-peer a v inicializačných fázach sú obe strany komunikácie rovnocenné. Neskôr sa rozdeľujú na server a klienta z dôvodu, že niektoré operácie nie sú symetrické (iba jedno zo zariadení generuje symetrický šifrovací kľuč, keďže AES pracuje iba s jedným kľúčom), a preto potrebujeme TCP funkcionalitu three way handshake-u a zostavovania packet-ov do správneho poradia. UDP sa taktiež používa v broadcast-ovom kontexte pri vyhľadávaní nových zariadení v sieti. Aplikácia udržuje dôvernosť použitím RSA šifrovania v kombinácii s úsudkom používateľa na overenie identity vzdialeného zariadenia a AES šifrovanie s "jednorazovým kľúčom" na prenos súborov. Protokol IP sa používa celý čas. O priebehu spojenia je používateľ informovaný pomocou notifikácií.

3.6.3 Rozpis status kódov

Aplikácia používa na signálovanie stavu aplikácie alebo typu posielaných dát status kódy. Pre ľahšie čítanie ďalšej pasáže, odporúčame prezretie ich pomenovaní a významov, ktoré nájdete v Príloha C.1, Tabuľka 2 Rozpis status kódov.

3.6.4 Charakteristika

3.6.4.1 Krok 0 – Zistenie SSID WiFi siete

Ak má aplikácia od používateľa povolenie používať sieť, zistí meno WiFi, na ktorú je pripojené zariadenie. Je to jediný dôvod prečo aplikácia vyžaduje systémové povolenie na presnú polohu. Problémom je, že aplikácia má zároveň prístup k GPS, ale tak je navrhnutý Android a nateraz sa s tým nedá nič robiť. Aplikácia následne skontroluje, či dané SSID WiFi siete je označené ako bezpečné. Ak nie je, dá sa pridať medzi bezpečné v používateľskom rozhraní. Toto je prvý krok ochrany, aby aplikácia nespúšťala žiadnu komunikáciu, keď je na verejných WiFi sieťach, kde by to mohlo byť nežiadúce.

3.6.4.2 Krok 1 – Broadcast discovery

Ak je WiFi sieť považovaná za bezpečnú, začne aplikácia rozposielať broadcast-y na prehľadanie siete a nájdenie iných zariadení, ktoré majú tiež nainštalovanú aplikáciu. Broadcast-y sa posielajú na adresu X.X.X.255, kde X je sieťová časť adresy. Povedané inými slovami, aplikácia posiela správy na všetky zariadenia v sieti, avšak len v rámci jednej siete, neposiela teda packet-y na adresu 255.255.255.255. Aplikácia funguje len v rámci LAN. Následne aplikácia pridá všetky zariadenia do zoznamu "živých zariadení" alebo, ak sú už v tomto zozname, tak len aktualizuje čas, kedy boli naposledy online. Ak zariadenie nie je v zozname dôveryhodných zariadení, sprístupní sa používateľovi možnosť zariadenie do tohto zoznamu pridať.

Aplikácia zároveň spúšťa broadcast listener, aby mohla reagovať na broadcast-y iných zariadení. Broadcast-y sa posielajú na porte 8008. Aplikácia reaguje na broadcast-y všetkých zariadení, aj tých, ktoré ešte neboli označené za dôveryhodné.

3.6.4.3 Krok 2 – Peer To Peer Decide

Krok 3 je posledným krokom pred nadviazaním TCP komunikácie a rozdelením zariadení na server a klienta. Zariadenia si posielajú tri typy správ o veľkosti štyroch bajtov:

- ak sú prvé 2 bajty 0 ide o číslo portu,
- ak sú prvé 2 bajty rovné 255, ale tretí bajt je menší ako 255 jedná sa o "rozhodovaciu správu". Táto správa pozostáva z 255.255.A.B, kde A je náhodná hodnota a B je číslo správy. Náhodná hodnota je číslo 0 alebo 1,
- ak sú prvé 3 bajty rovné 255, jedná sa o požiadavku o preposlanie stratenej správy. Táto správa pozostáva z 255.255.255.B, kde B je číslo správy, ktorú máme preposlať.

Samotný proces rozhodnutia, kto je server a kto je klient, prebieha tak, že obe zariadenia náhodne vyberú číslo 0 alebo 1. V ďalšom bajte je k tejto hodnote následne priradené ďalšie prirodzené číslo z poradia začínajúceho 0 indikujúce číslo správy. Správa je následne poslaná druhej strane a zároveň je prijatá hodnota od druhej strany. Ak sú náhodne hodnoty rovnaké, zahodia sa a generujú sa nové a proces sa opakuje. Ak sú rozdielne, strana ktorá vygenerovala 0 sa stáva serverom a strana, ktorá vygenerovala 1 sa stáva klientom. Ak nám prišla správa s číslom správy vyšším o viac ako 1 ako najvyššie prijaté číslo správy, pošle sa požiadavka o preposlanie chýbajúcich správ. Toto rozhodovanie prebieha na porte 8009. Pre jednoduchšie odôvodnenie efektivity tohto riešenia prikladáme tabuľku pravdepodobnosti, že sa 2 zariadenia dohodli pre N počet packet-ov viď. Príloha C.1, Tabuľka 1.

Podotýkam, že správa bez hlavičky je veľká 4 bajty, z toho vyplýva, že aj veľmi nepravdepodobných 10 pokusov je len 80 bajtov bez hlavičiek. S hlavičkami až po internetovú vrstvu vrátane, je celková veľkosť týchto výmen 640 bajtov.

Server následne generuje náhodné neprivilegované porty, až kým sa mu nepodarí na jeden bind-núť. Následne pošle tento port klientovi. Toto je posledný packet v UDP.

3.6.4.4 Krok 3 – Identifikácia zariadenia

Zariadenia si vymenia svoje hostname-y a prípadne aj to, či ide o jednorazové pripojenie. Na základe hostname-u druhej strany sa pokúsi aplikácia nájsť RSA kľúče v Android Isolated Storage. Vrátené kľúče sú v XML formáte, pričom private key je náš a public key patrí vzdialenej strane. Všetka komunikácia je následne šifrovaná týmito kľúčmi. Ak je druhá strana podvodník, ktorý sa len vydáva za toto zariadenie, tak sa jej nepodarí nič „prečítať". Pri prvotnom pripojení alebo jednorazovom pripojení tieto kľúče pochopiteľne neexistujú. V tom prípade pokračuje aplikácia krokom 4. Inak pokračuje rovno krokom 5.

3.6.4.5 Krok 4 - Šifrovanie

Obe strany vygenerujú kľúčový pár RSA. Public key sa pošle vzdialenej strane a príjme sa public key vzdialenej strany. Náš privátny kľúč a verejný kľúč vzdialenej strany sa uložia do Android Isolated Storage. Server vygeneruje AES kľúč, zašifruje ho a pošle ho klientovi. Klient pošle späť správu o jeho prijatí, čím ukončuje časť nadväzovania šifrovaného spojenia.

3.6.4.6 Krok 5 – Posielanie súborov

Obe strany si pozrú, či majú skladby, ktoré by mohli zdieľať. Ak nemajú, začnú posielať status kód 255 END. Ak majú, pošlú status kód 20 SyncRequest alebo 30 SongSendRequest a čakajú na odpoveď.

Ak dostanú kód 30 SongSendRequest, opýtajú sa používateľa, či prijíma pripojenie a po odpovedi pošlú kód 7 ConnectionAccepted alebo 8 ConnectionRejected. Ak pripojenie bolo prijaté, pošlú kód 31 SongRequestInfoRequest, na ktorý sa odpovedá kódom 32 SongRequestInfo a zoznamom skladieb, ktoré sa zobrazia používateľovi s otázkou, či ich prijíma a po odpovedi pošlú kód 33 SongRequestAccepted alebo 34 SongRequestRejected.

Ak dostanú odpoveď 22 SyncRejected, 34 SongRequestRejected alebo 8 ConnectionRejected začnú posielať status kód 255 END. Ak dostanú odpoveď 21 SyncAccepted alebo 33 SongRequestAccepted, začnú posielať status kód 40 SongSend po ktorom nasleduje samotný mp3 súbor. Na to odpovedá prijímajúca strana status kódom 254 ACK, čím potvrdzuje jeho prijatie. Ak prijímajúca strana zistí na základe ID3v2 tag-ov v danom súbore, že patrí umelcovi

alebo do albumu, od ktorého nemá náhľadový obrázok, odošle status kód 43 ArtistImageRequest alebo status kód 44 AlbumImageRequest, na ktorý odpovedá posielajúca strana status kódmi 41 ArtistImageSend a 42 AlbumImageSend, ak daný obrázok má. Ak ho nemá, posiela status kódy 45 ArtistImageNotFound a 46 AlbumImageNotFound. Na status kódy 41 ArtistImageSend a 42 AlbumImageSend odpovedá prijímajúca strana status kódom 254 ACK, čím potvrdzuje prijatie súboru.

Celá komunikácia je šifrovaná pomocou RSA, okrem samotných súborov, ktoré sú šifrované pomocou AES kvôli limitovanej veľkosti vstupných dát algoritmu RSA. Ďalší dôvod využitia AES je to, že je to výrazne rýchlejší šifrovací systém.

3.6.4.7 Krok 6 - Koniec

Po prijatí status kódu 255 END sa rozhodneme, či máme ešte niečo na poslanie. Ak áno, tento status kód ignorujeme a ďalej posielame súbory. Ak nie, alebo sme nedostali povolenie posielať súbory, odpovieme status kódom 255 END, načo počkáme 100 milisekúnd, aby stihla vzdialená strana našu odpoveď prijať. Následne zavrieme socket a uvoľníme použitú pamäť.

3.6.5 Dátové jednotky podľa status kódu

Všeobecne vyzerajú posielané dátové jednotky nasledovne: Príloha C.3, Obrázok 14 Všeobecné PDU.

Výnimkou sú:

- status kódy None, Wait, OnetimeSend, ConnectionAccepted, ConnectionRejected, AesReceived, SyncRequest, SyncAccepted, SyncRejected, SongSendRequest, SongRequestAccepted, SongRequestRejected, Ack a End pozostávajú iba zo samotného status kódu,
- status kódy, ktoré pracujú s dátami a to SongRequestInfo, SongSend, ArtistImageSend, AlbumImageSend. Ich dátové jednotky sú rozdelené na aspoň 2 časti, kde prvá je šifrovaná cez RSA a obsahuje status kód, AES inicializačný vektor, dĺžku súboru a v prípade, že posielame obrázok, tak aj meno umelca alebo albumu, ktorému patrí obrázok až do konca dátovej jednotky. Nasledujúce časti sú bloky samotného súboru šifrovaného pomocou AES. Pre zobrazenie dátovej jednotky prvej časti viď. Príloha C.3, Obrázok 15 PDU pre súborové príkazy.

4. Závery a zhrnutie

Music Without Pain (MWP) je Android aplikácia slúžiaca hlavne na prehrávanie hudby a správu hudobných súborov v zariadení kompletne na jednom mieste. Aplikácia MWP disponuje prehrávaním hudby, automatickým organizovaním hudby na základe albumov a interpretov a k tomu umožňuje tvorbu používateľom vytvorených zoznamov hudby pre viac personálne prispôsobené prehrávanie. V rámci albumov a zoznamov hudby si používateľ môže poradie hudby riadiť viacerými funkciami, ako náhodný výber, opakovanie albumu a opakovanie jednotlivej skladby. Dané funkcie používateľ nájde na viacerých miestach, nie len v rámci aplikácie, ale aj v rámci celého zariadenia. Aplikácia disponuje ovládaním a kontrolou stavu prehrávania skladieb v aplikácií v bočnom paneli, ktorý je dostupný pre používateľa po celú dobu trvania používania aplikácie. Okrem toho aplikácia, poskytuje globálnu notifikáciu, prostredníctvom ktorej používateľ môže kontrolovať ovládanie a stav hudby aj mimo aplikácie a aj keď je mobil zamknutý. Aplikácia tak isto ponúka widget ktorý si môže používateľ pripnúť na domovskú obrazovku zariadenia s možnosťami kontroly a ovládania stavu prehrávania. Jednou z najzákladnejších funkcionalít aplikácie je to, že samotné skladby sú staticky uložené na zariadení, čo používateľovi umožňuje počúvať hudbu aj bez prístupu na internet. Pre nás to je veľké plus, keďže nechceme míňať zbytočne veľa peňazí, za platený software resp. prístup na dáta, pokiaľ nie sme pripojení k WiFi sieti. To by však mohlo viacerých obmedzovať z hľadiska počúvania na viacerých zariadeniach. Statické súbory nie sú automaticky synchronizované s ostatnými zariadeniami používateľa. Tu prichádza na rad hlavná vlastnosť aplikácie MWP, a to automatická synchronizácia skladieb medzi zariadeniami. V rámci aplikácie je k dispozícii funkcia Share. Pri pripojení zariadenia na internet, aplikácia začne automaticky posielať a prijímať skladby medzi zariadeniami v rámci jednej siete. Tým sa zabezpečuje, že používateľ môže počúvať tú istú hudbu medzi všetkými zariadeniami siete.

Samozrejme, vytvorenie mobilnej aplikácie podobného charakteru a kalibru sa určite nezaobišlo bez množstva prekážok, ktoré na nás čakali na našej ceste. Či už to boli problémy s tvorbou používateľského rozhrania, problémy s prehrávaním hudby, prenášaním hudby alebo dizajnovanie štruktúry aplikácie, nikdy nás to nezastavilo. Dobrým príkladom by mohol byť lazy loading pri načítavaní jednotlivých skladieb do používateľského rozhrania, kedy bolo potrebné vymyslieť čo najefektívnejší spôsob načítania zoznamu skladieb bez akéhokoľvek spomalenia zariadenia. Aj napriek tomu, že to bol ťažký problém, podarilo sa nám ho vyriešiť s pomocou použitia viacerých vlákien spustených paralelne. Ďalší veľký problém bolo posielanie súborov medzi zariadeniami. Tu už nemáme k dispozícii pekné high-level-ové

funkcie, ale musíme si „zašpiniť ruky" a pracovať priamo s poľami bajtov a streamami. Čo by nebol ani taký problém, keďže si vieme vždy zapnúť debugger a pozrieť obsah pamäte… Až do momentu, kedy začne byť všetko šifrované, vidíme náhodné hodnoty a aplikácia padá kvôli nevhodnému počtu prečítaných bajtov a nie ja iná možnosť zistiť prečo, inak ako si celý kód odsimulovať v hlave a nájsť všetky chyby. Ale aj to sme nakoniec vyriešili a aplikácia je plne schopná synchronizovať súbory medzi zariadeniami.

Do budúcna si aplikácia ešte vyžaduje zvýšenie podpory pre všetky druhy zariadení čo sa týka prenášania hudby, keďže typov sietí a mobilov je mnoho a každý z nich má špecifické požiadavky, či už bezpečnostné alebo funkcionálne. Viacero miest v aplikácií si žiada optimalizáciu rýchlosti a zníženie využitia RAM pamäte zariadenia. Budúcnosť projektu vidíme v expanzii na viacero platforiem. Vytvorenie počítačovej aplikácie na rôzne platformy ako je Windows, Linux a Mac OS. Jednou z funkcií, ktorú by sme radi pridali do aplikácie, by bol ekvalizér pre dekompozíciu a upravovanie jednotlivých frekvencií hudobných skladieb.

Tvorbu aplikácia neberieme len ako prínos do sveta, ktorý niekomu uľahčí život alebo im zlepší deň, ale berieme to ako projekt, ktorý nás posunul vpred. Projekt, do ktorého sme vložili hodiny práce a vytvorili niečo, na čo môžeme byť hrdí, pretože sme sa počas tejto cesty naučili mnoho nového a zároveň veľa dokázali. To všetko by však nebolo možné bez nášho konzultanta projektu, preto by sme chceli poďakovať Petrovi Keuschovi za umožnenie tvorby tejto aplikácie a neustálej podpory. Umožnilo nám to okúsiť prácu v reálnom svete, za čo sme vďační. Finálnu verziu projektu považujeme za úspešnú a tešíme sa na budúce inovácie našej aplikácie Music Without Pain.

5. Resumé

Android application Music Without Pain (MWP) is an application focused on creating a full music experience without being dependent on the internet. In our application users are able to browse their music or any mp3 file they have stored on the device itself. MWP automatically takes all mp3 files, and sorts them by authors and albums. Apart from browsing music, our application is capable of playing already mentioned files. Users are able to control and track music playback by various means, either with global notification that is accessible even when the device is locked or through a widget on the home screen of the device. Of course users have access to track control station in the application at all times within the applications sidebar. Within MWP users are able to create their own custom playlists. Furthermore, MWP is capable of editing the metadata of individual music files. Users can change the title, image, author name or name of the album.

MWP is an application that is supposed to allow users to manipulate their music more freely. Unlike any other Android application or subscription based software, we seek to create an independent music environment throughout all devices of individual users. This brings us to the last feature that MWP offers. Within MWP users can share their music across all Android based devices. Users have the means for synchronizing their individual devices within MWP, once they are connected, music is automatically synchronized within all devices instantly when they are connected to the same network. This may be a network dependent feature, however, it can be a network of any kind that does not require internet connection. This creates a little music network right in the middle of the user's device environment.

In the future we strive to create an even greater music environment by creating Windows, Linux and Mac OS counterparts with the same functionality. We are on the lookout for better optimization and compatibility with different brands of networks and devices to ensure the best user experience for all.

6. Zoznam použitej literatúry

1. GNU Lesser General Public License v2.1. *GNU.org.* [Online] [Dátum: 3. Február 2024.] https://www.gnu.org/licenses/old-licenses/lgpl-2.1.html#SEC1.

2. Apache License, Version 2.0. *The Apache Software Foundation!* [Online] [Dátum: 3. Február 2024.] https://www.apache.org/licenses/LICENSE-2.0.

3. The MIT License – Open Source Initiative. *Open Source Initiative.* [Online] [Dátum: 3. Február 2024.] https://opensource.org/license/mit/.

4. Toolbox Subscription Agreement - Students and Teachers. *JetBrains.* [Online] [Dátum: 3. Február 2024.] https://www.jetbrains.com/legal/docs/toolbox/license_educational/.

5. Shanidevani. What is JetBrains Rider? A Comprehensive Overview | by Shanidevani. *Medium.* [Online] 27. August 2023. [Dátum: 30. Január 2024.] https://medium.com/@shani12devani/what-is-jetbrains-rider-a-comprehensive-overview-3ca07a096a4. ISO 690 - Číselný odkaz.

6. Kopf's, Ben Kopf and Ben. The Power of Figma as a Design Tool. *Toptal.* [Online] Toptal, LLC, 2019. [Dátum: 30. Január 2024.] https://www.toptal.com/designers/ui/figma-design-tool. ISO 690 - Číselný odkaz.

7. Jaiswal, Sonoo. What is Xamarin. *Javatpoint.* [Online] Javatpoint, 9. 7 2023. [Dátum: 30. 1 2024.] https://www.javatpoint.com/what-is-xamarin. ISO 690 - Číselny odkaz.

8. Jain, Sandeep. Advanced Encryption Standard (AES). *GeeksforGeeks.* [Online] 22. Máj 2023. [Dátum: 30. Január 2024.] https://www.geeksforgeeks.org/advanced-encryption-standard-aes/. ISO 690 - Číselný odkaz.

9. Adamsen, Søren Gjesse and Christoffer. Shrinking your app with R8. Posted by Søren Gjesse, Software… | by Android Developers | Android Developers. *Medium.* [Online] 28. Júl 2020. [Dátum: 30. Január 2024.] https://medium.com/androiddevelopers/shrinking-your-app-with-r8-909efac25de4. ISO 690 - Číselný odkaz.

10. What is RSA? How does an RSA work? *Encryption Consulting.* [Online] [Dátum: 30. Január 2024.] https://www.encryptionconsulting.com/education-center/what-is-rsa/. ISO 690 - Číselný odkaz.

11. Welcome to the .NET API browser. *Microsoft Learn.* [Online] https://learn.microsoft.com/en-us/dotnet/api/. ISO 690 - Číselný odkaz.

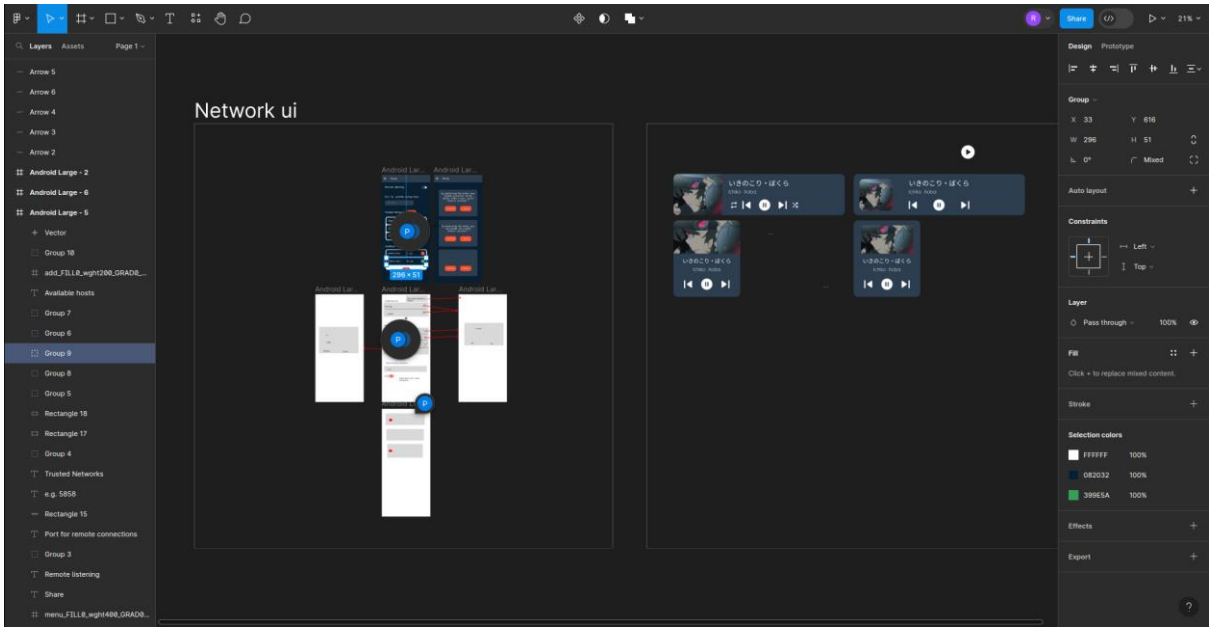
12. Isolated storage definition — Glossary. *NordVPN*. [Online] [Dátum: 2024. Január 2024.] https://nordvpn.com/cybersecurity/glossary/isolated-storage/. ISO 690 - Číselný odkaz.

13. Develop for Android. *Android Developers.* [Online] https://developer.android.com/develop/. ISO 690 - Číselný odkaz.

7. Prílohy

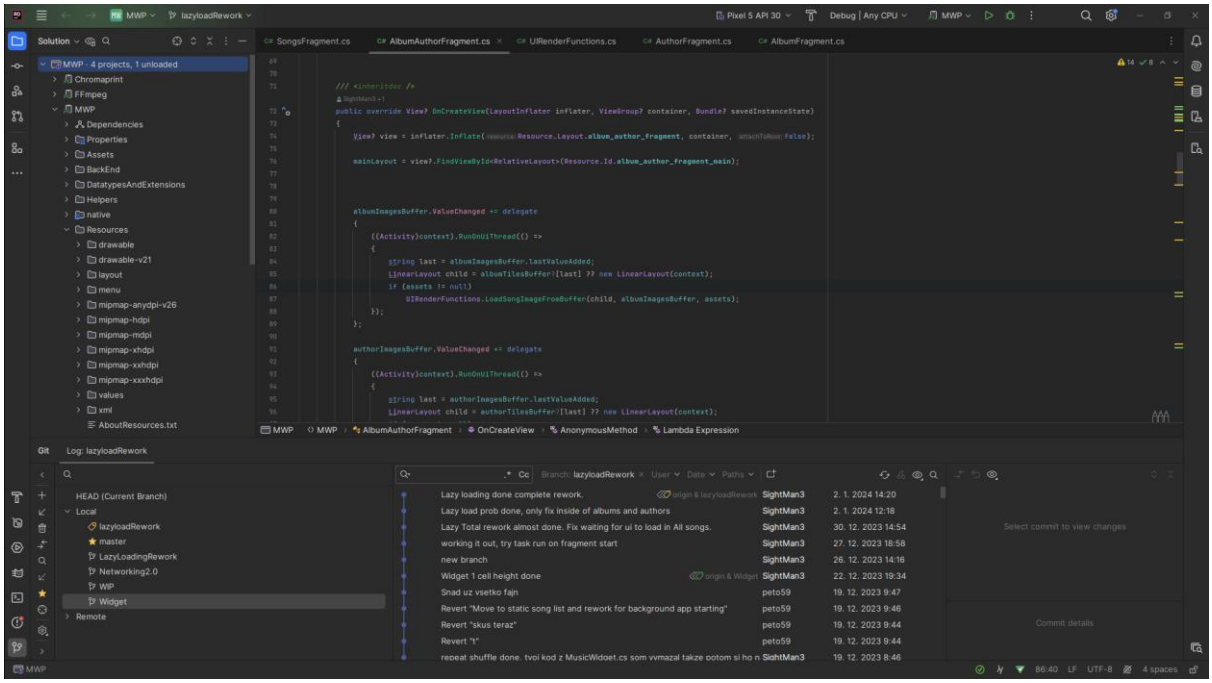
PRÍLOHA A – Použité technológie

PRÍLOHA A.1 – Prostredie aplikácie Figma



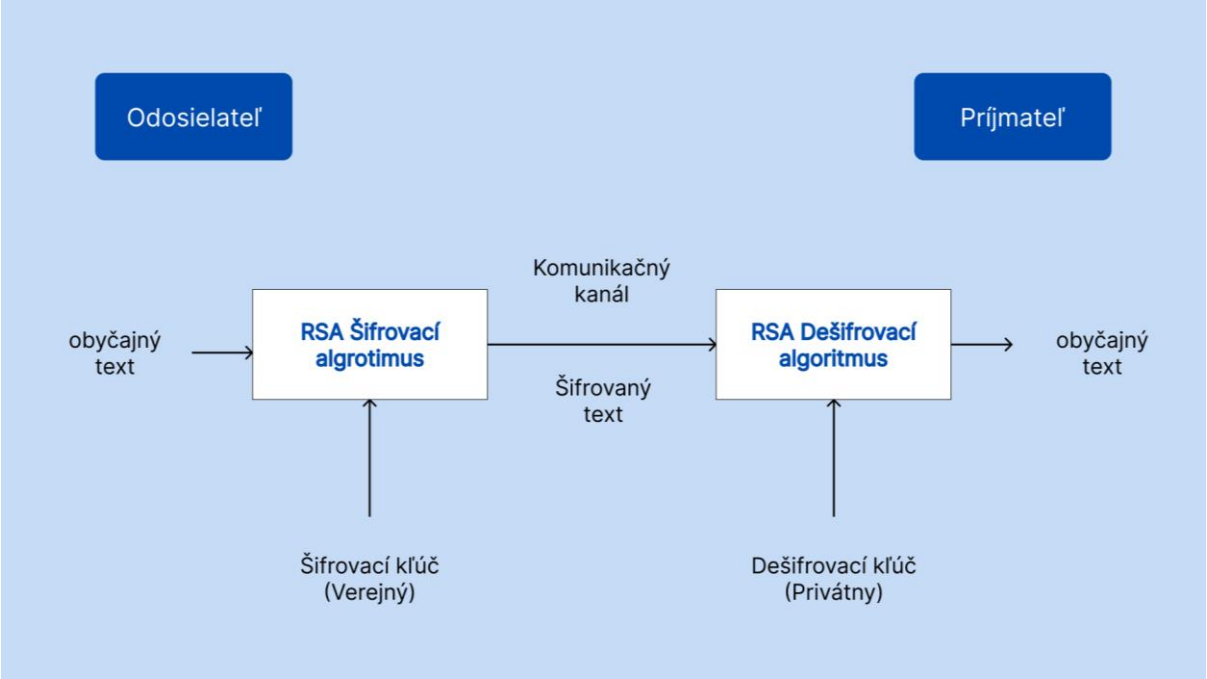
Obrázok 2 Prostredie aplikácie figma

PRÍLOHA A.2 – Prostredie aplikácie JetBrains Rider IDE



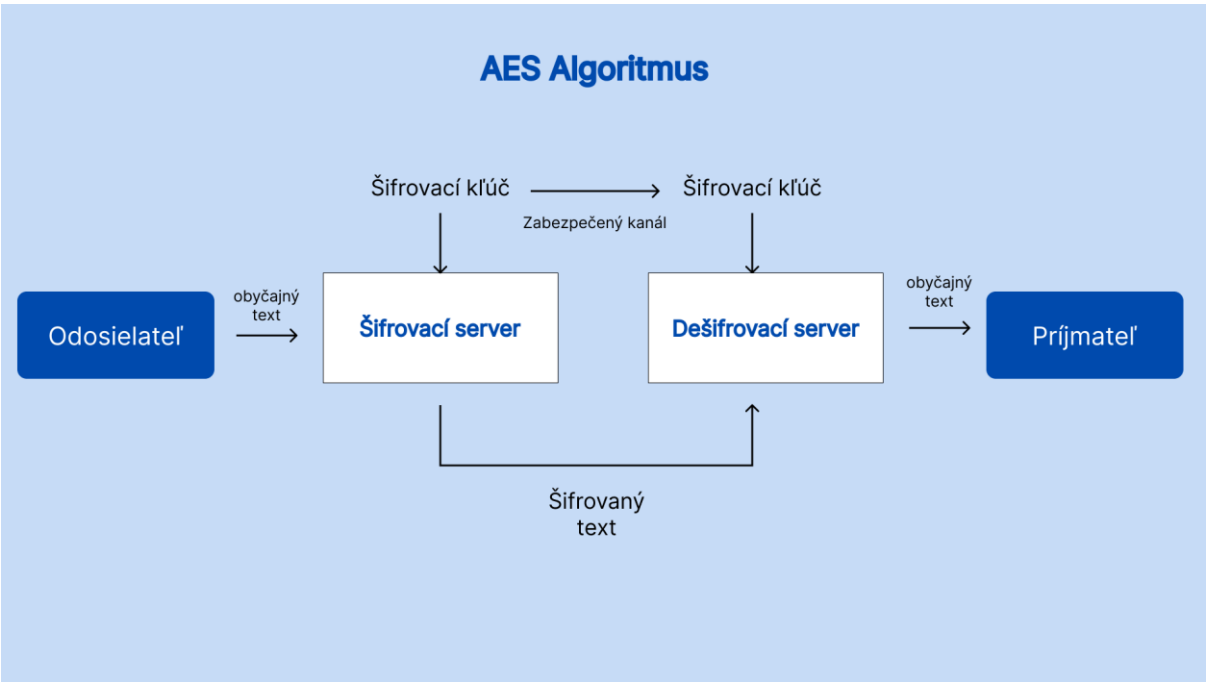
Obrázok 3 Prostredie aplikácie JetBrains Rider IDE

PRÍLOHA A.3 – Vývojový diagram šifrovacieho algoritmu RSA



Obrázok 4 Vývojový diagram šifrovacieho algoritmu RSA

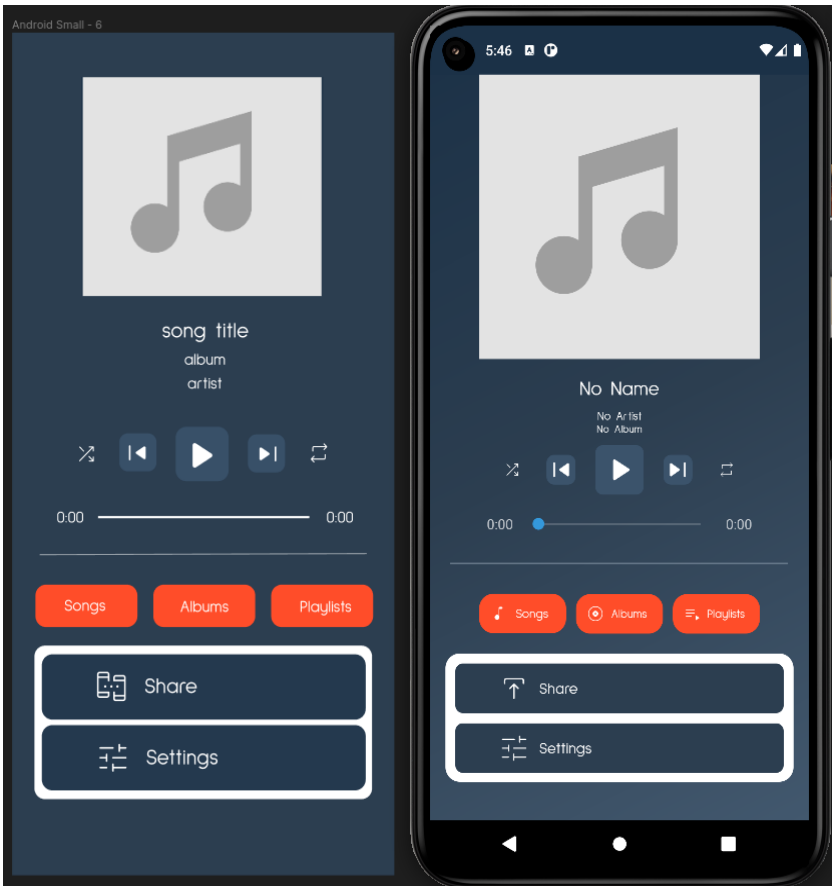
PRÍLOHA A.4 – Vývojový diagram šifrovacieho algoritmu AES



Obrázok 5 Vývojový diagram šifrovacieho algoritmu AES

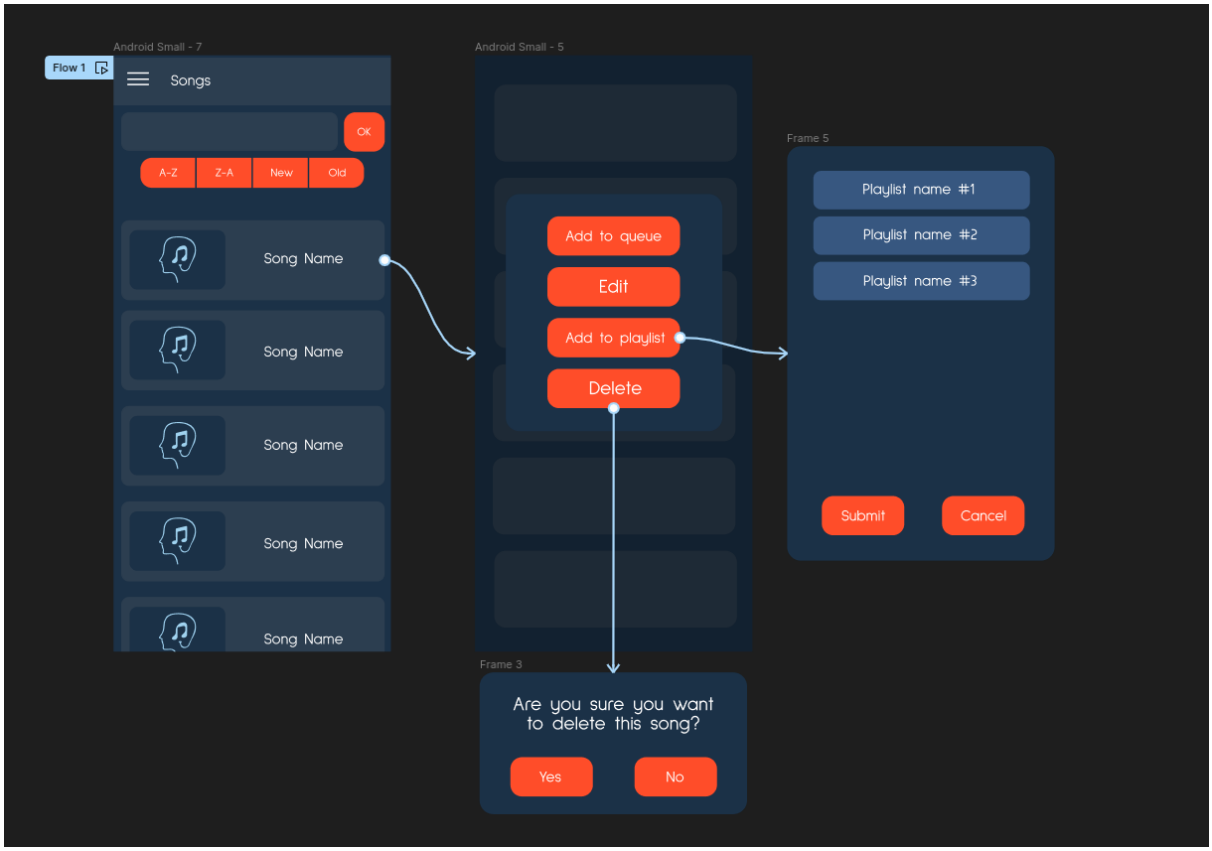
PRÍLOHA B – Používateľské rozhranie

PRÍLOHA B.1 – Dizajn a implementácia používateľského rozhrania bočného prehrávača



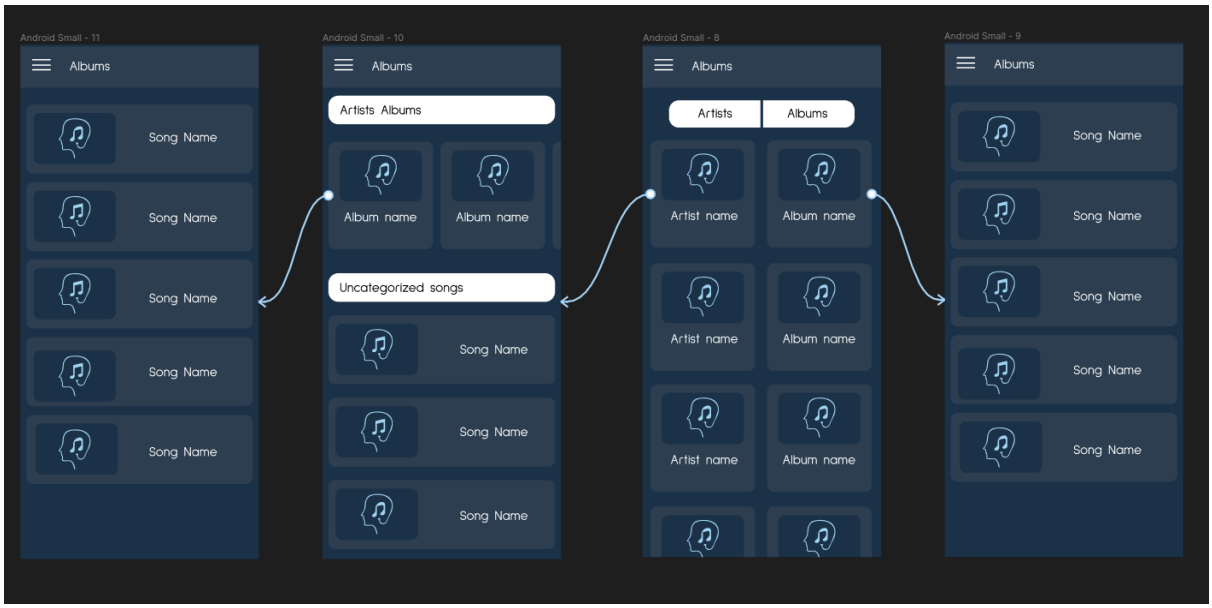
Obrázok 6 Dizajn a implementácia používateľského rozhrania bočného prehrávača

PRÍLOHA B.2 – Dizajn a funkcionálny diagram používateľského rozhrania Songs



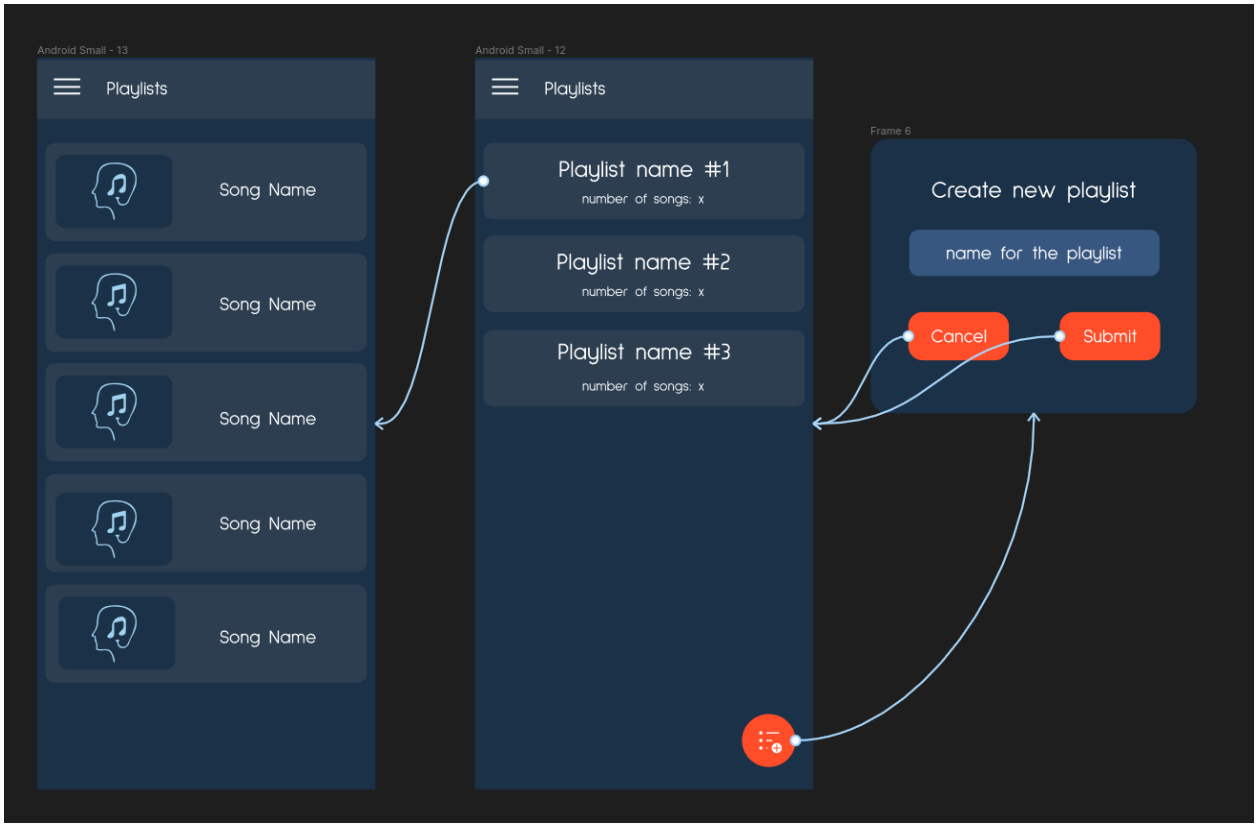
Obrázok 7 Dizajn a funkcionálny diagram používateľského rozhrania Songs

PRÍLOHA B.3 – Dizajn a funkcionálny diagram používateľského rozhrania Albums



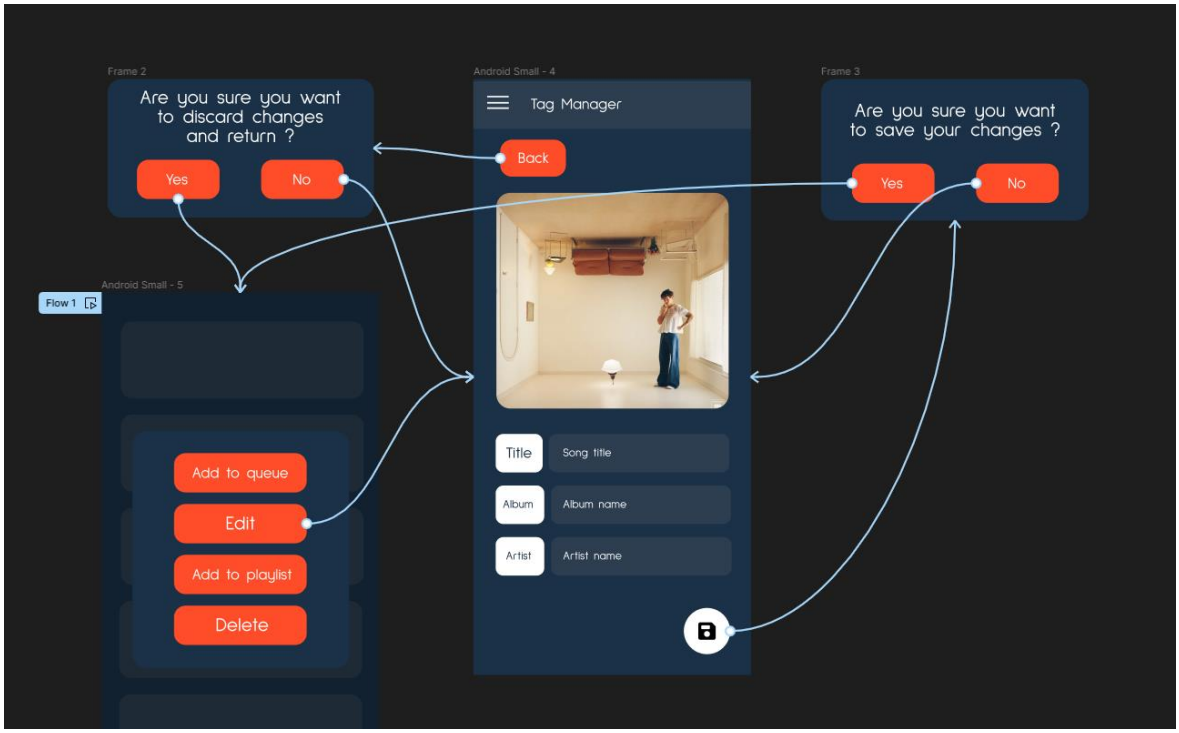
Obrázok 8 Dizajn a funkcionálny diagram používateľského rozhrania Albums

PRÍLOHA B.4 – Dizajn a funkcionálny diagram používateľského rozhrania Playlists



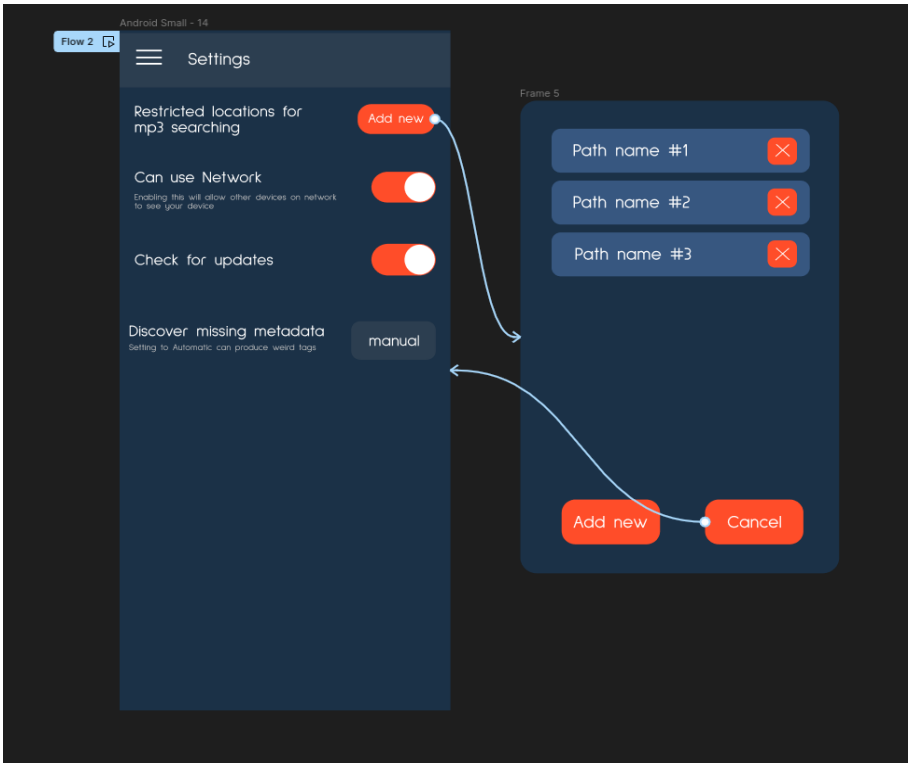
Obrázok 9 Dizajn a funkcionálny diagram rozhrania Playlists

PRÍLOHA B.5 – Dizajn a funkcionálny diagram používateľského rozhrania Tag Manager



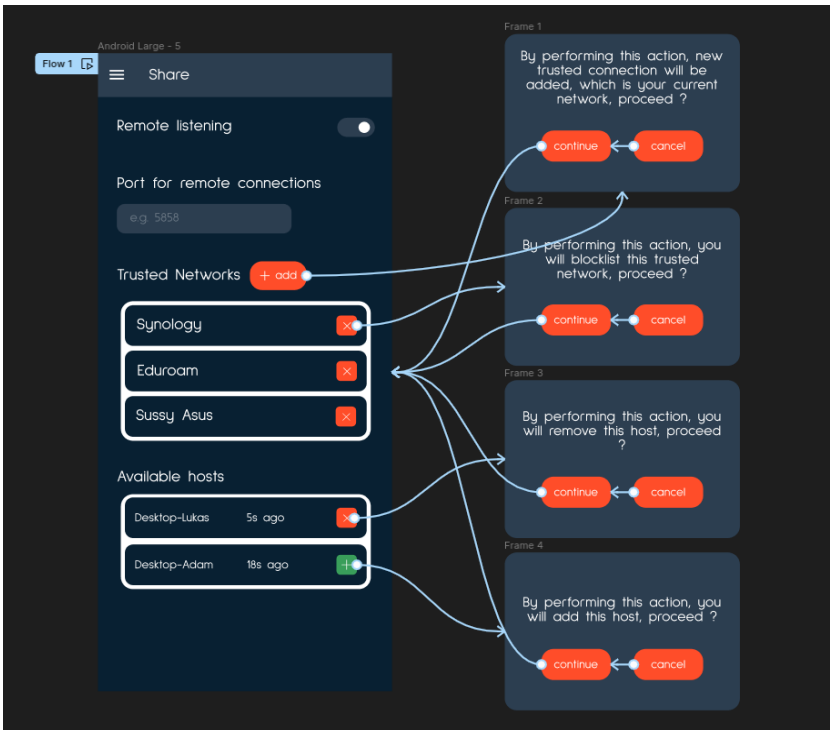
Obrázok 10 Dizajn a funkcionálny diagram používateľského rozhrania Tag Manager

PRÍLOHA B.6 – Dizajn a funkcionálny diagram používateľského rozhrania nastavení

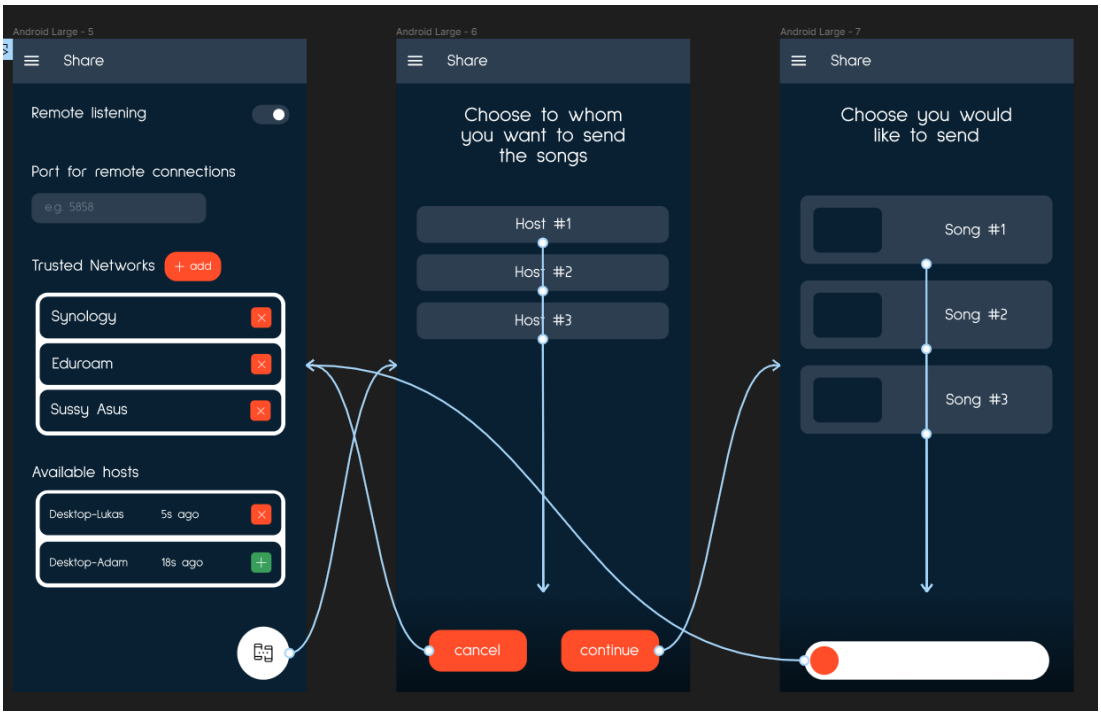


Obrázok 11 Dizajn a funkcionálny diagram používateľského rozhrania nastavení

PRÍLOHA B.7 – Dizajn a funkcionálny diagram používateľského rozhrania pre vzdialené zdieľanie



Obrázok 12 Dizajn a funkcionálny diagram používateľského rozhrania pre vzdialené zdieľanie



Obrázok 13 Dizajn a funkcionálny diagram používateľského rozhrania posielania skladieb

PRÍLOHA C – Network Manager

PRÍLOHA C.1 – Pravdepodobnosť nadviazania spojenia pre počet packet-ov

Tabuľka 1

Počet poslaných packet-ov	Pravdepodobnosť, že stále nie je rozhodnuté
1	50%
2	25%
3	12.5%
4	6.25%
5	3.125%
6	1.5625%
7	0.78125%
8	0.390625%
9	0.1953125%
10	0.09765625%

PRÍLOHA C.2 – Rozpis status kódov

Tabuľka 2 Rozpis status kódov

Status kód	Pomenovanie status kódu	Význam status kódu
0	None	Používaný pri socket timeout-e, funguje rovnako ako Wait len s 100 milisekundovým delay-om
1	Wait	Posiela sa, keď vzdialená strana robí nejakú CPU namáhavú úlohu. Po jeho prijatí čaká aplikácia 100 milisekúnd
5	OnetimeSend	Posiela sa, keď máme jednorazové pripojenie

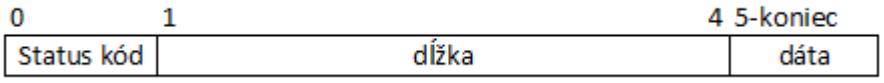
		vyžiadané používateľom
7	ConnectionAccepted	Odpoveď na OnetimeSend o tom, že prijímame pripojenie
8	ConnectionRejected	Odpoveď na OnetimeSend o tom, že odmietame pripojenie
10	Host	Nachádza sa pred hostname-mom
11	RsaExchange	Nachádza sa pred public RSA key
12	AesSend	Nachádza sa pred AES key, posiela ho len server
13	AesReceived	Posiela ho len klient, posiela sa ako potvrdenie prijatia AES kľúča
20	SyncRequest	Otázka, či môžeme začať posielať naše skladby
21	SyncAccepted	Kladná odpoveď na SyncRequest
22	SyncRejected	Záporná odpoveď na SyncRequest
23	SyncCount	Hovorí o počte skladieb, ktoré dostaneme počas synchronizácie
30	SongRequest	Otázka, či môžeme začať posielať skladby počas jednorazového pripojenie
31	SongRequestInfoRequest	Požiadavka o poslanie podrobných informácii o skladbách počas jednorazového pripojenia
32	SongRequestInfo	Podrobné informácie o skladbách počas jednorazového pripojenia
33	SongRequestAccepted	Kladná odpoveď na SongRequest
34	SongRequestRejected	Záporná odpoveď na SongRequest
40	SongSend	Oznamuje, že nasledujúce dáta sú mp3 súbor
41	ArtistImageSend	Odpoveď na ArtistImageRequest, nasleduje po

		ňom súbor
42	AlbumImageSend	Odpoveď na AlbumImageRequest, nasleduje po ňom súbor
43	ArtistImageRequest	Žiadosť o poslanie obrázku k špecifickému umelcovi
44	AlbumImageRequest	Žiadosť o poslanie obrázku k špecifickému albumu
45	ArtistImageNotFound	Alternatívna odpoveď na ArtistImageRequest, naznačuje, že taký súbor na zariadení nie je
46	AlbumImageNotFound	Alternatívna odpoveď na AlbumImageRequest, naznačuje, že taký súbor na zariadení nie je
254	Ack	Potvrdzuje prijatie súboru
255	End	Žiadosť o skončenie komunikácie

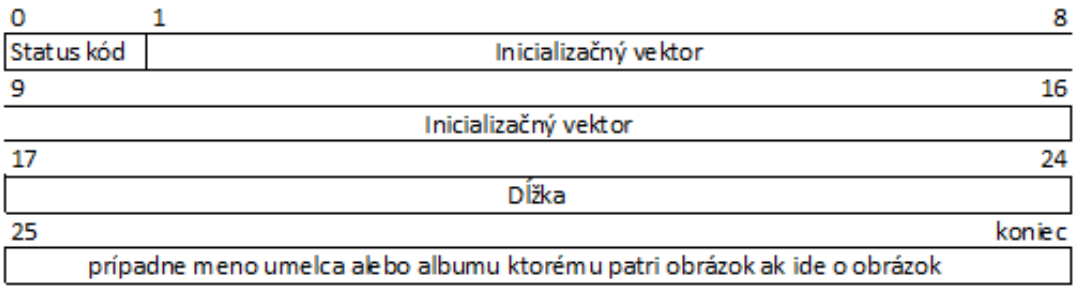
Status kódy, ktoré sa nenachádzajú v tabuľke, sú rezervované pre budúce použitie.

PRÍLOHA C.3 – Dátové jednotky podľa status kódu

Čísla nad bunkami označujú počet bajtov.



Obrázok 14 Všeobecné PDU



Obrázok 15 PDU pre súborové príkazy