Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мережних технологій факультету інформатики

**Розробка мобільного додатку**

**Текстова частина до курсової роботи за спеціальністю «Інженерія програмного забезпечення»**

Керівник курсової роботи

Старший викладач

Борозенний С. О.

_____

*(прізвище та ініціали)*

_____

*(підпис)*

*“____” _____*

2020 р.

Виконав студент Муратов Віктор

*(прізвище та ініціали)* *“____”*

_____ 2021 р.

Київ 2021

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ
Зав.кафедри
мультимедійних систем,
доцент

_____О. П. Жежерун
(підпис)
„____”_____2021 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу

студенту_____ факультету _____ _____курсу

ТЕМА _____

Вихідні дані:

-

-

Зміст ТЧ до курсової роботи:

Індивідуальне завдання

Вступ

1 Огляд

2 Розробка алгоритму

3 Розробка програми

4 Джерела

5 Додатки

Дата видачі „___” _____ 2021 р.

Керівник _____(підпис)

Завдання отримав _____ (підпис)

**Тема: Розробка мобільного додатку**

**Календарний план виконання роботи:**

| № п/п | Назва етапу дипломного проекту (роботи) | Термін виконання етапу | Примітка |
|---|---|---|---|
| 1. | Отримання завдання на курсову роботу. | 01.01.2021 | |
| 2. | Огляд технічної літератури за темою роботи. | 01.01.2021 | |
| 3. | Виконати аналіз сучасних методів … | 01.01.2021 | |
| 3. | Розробка алгоритму … | 01.02. 2021 | |
| 4. | Програмування розробленого алгоритму | 01.02. 2021 | |
| 5. | Застосування розробленого алгоритму до … | 01.02. 2021 | |
| 6. | Написання пояснювальної роботи. | 11.03.2021 | |
| 7. | Створення слайдів для доповіді та написання доповіді. | 11.03.2021 | |
| 9. | Остаточне оформлення пояснювальної роботи та слайдів. | 11.03.2021 | |
| 10. | Захист курсової роботи (проекту) | Після 11.03.2021 | |

Студент _____ Керівник _____ “_____” _____

Зміст

**Анотація:**

Ціль курсової роботи: дослідження аудіоформатів, реалізація застосунку на Андроїд який є аудіопрогравачем

**Вступ**

**Оцінка сучасного стану цифрових аудіосистем**

Станом на сьогодній день існує багато різних аудіоформатів та програвачів. У кожного на ЕОМ з ОС Віндос є штатний ауділрогравач, на смартфонах за замовчуванням теж він встановлений. Проте завжди є куди розвиватися та робити більш ефективні та зручні рішення. В гугл плей присутній широкий вибір різних плеерів для вашого смартфону. Є безкоштовні та платні версії.

Прості та «наворочені». Різні пристрої можуть взаємно працювати через AUX та Bluetooth.

**Обґрунтування необхідності виконання роботи**

Аудіосистеми дуже затрубовані у нашому житті тимпаче цифрові. Особливо цифрові. Особисто я постійно користуюсь плеером на смартфоні: вдома, на вулиці і коли їзджу на машині. Музика – ритм нашого життя.

**Новизна теми**

Цифрові програвачі масово розпочались у 1984(1982) році. Першим програвачем був SONY Discman D-50. Разом с появою мобільних телефонів, комп'ютерів, смартфонів аудісистеми з'явились і там. А от цифрові формати запису з'явились значно раніше. 3 жовтня 1938 р. Британський телефонний інженер Алек Харлі Рівз подав до французького патентного відомства перший патент, що описує техніку, відому сьогодні як імпульсно-кодова модуляція (PCM). У 2000 році було представлено формат DVD-audio.

**Зв'язок з іншими науковими работами**

При створені проекта та при дослідженні теми у цілому я не використовував на пряму інші наукові роботи. Лише тематичні сайти, онлайн довідники. З цифровими аудісистемами пов’язано багато наукових робіт та досліджень. Цифрові аудіосистеми затребувана та перспективна тема.

# РОЗДІЛ 1: Концепція нашого аудіо застосунку .

## 1.1 Тема дослідження та проекту

Для початку необхідно визначитись з тим, який застосунок я буду розробляти. Вибір був не простий – треба було обрати цікаву тему, про яку є що розповісти, є що дослідити. При цьому ця тема буде реальною у якісній реалізації силами одного студента. Перебирав я різні теми: від геолокації до інтернет магазинів та різних алгоритмів та ігор. Вирішив зупинитись на аудіосистемах – існує багато різних форматів та аудіосистему мають багату історію та мають широке застосування у нашому житті. Навіть зараз,набираючи цей текст я слухаю музику. Темою курсової є мобільний застосунок тому це має буте система для мобільних пристроїв. І тому я буду досліджувати та створювати аудіопрогравач для Андроїд. Він має зручним,зрозумілим та ефективним. Має доставляти задоволення від використання. Саме такий додаток я маю ціль реалізувати.

## 1.2 Назва

### 1.2.1 Аналіз методів створення назв для проекту

Правильна назва значно підвищує шанси на успіх. Назва – це перший крок до успіху, або до невдачі. Існують наступні підходи до створення назви: Абстракція – деяка абстракція над котрою задумуючись потенціальний юзер отримає інтерес до проекту, внутрішня назва – назва яку можно зрозуміти лише використав програвач, назви пов’язані з процессом використання додатку – назви які описують функціональність або наповнення, інструкція – назва яка деє вказівку по використанню додатку, назва на честь єлементів застосунку – описує певний цікавий для користувача момент у застосунку який має виділити його на фоні інших продуктів.  Приклади назв цифрових програвачів:

1)»Музыкальный проигрователь по умолчанию»

2)»Бесплатный MP3»

3)»AIMP»

4)»Музыкальный плеер – MP3-плеер»

5)»Музыкальный плеер – Bass booster»

6)»USB Audio Player PRO»

7)»PowerAudio Free Music Player»

8)»Pulsar»

9)»VLC for android»

10)»Sun player»

Велика частина назв нагадує мені «дешеву замануху» і не відповідає дійсним можливостям. Проте є і назви які демонструють те що застосунок має багатий набір опцій для програвання аудіо. Також присутні цікаві унікальні назви

**Вплив назви на комерційний успіх:**

Ім'я, яке демонструє проект з цікавої точки зору, ім'я на честь успішної франшизи, ім'я, яке спонукає клієнта задуматися про проект і, що більш очевидно, позитивно впливає на успіх - це перше враження та бренд . А якщо навпаки… Типове або безглузде ім'я лякає, або користувач просто гортає плей маркет далі.

### 1.2.2 Яку назву дамо проекту?

Зроблено виводи що назва має:

1)Демонструвати суть застосунку

2)Виділятись на фоні конкруентів

3)Демонструвати сильні сторони застосунку

4)Закладатись у голові юзера

На основі цих умозаключень я обрав назву «Mental player» ця назва має демонструвати те що з нашим застосунком та музикою користувач може

«злитись воєдино» та отримувати задоволення від процесу прослуховування музики та аудіозаписів

## 1.3 Відмінність від подібних додатків

### 1.3.1 Дослідження конкурентів

#### 1.3.1.1 Дослідження «Музыкальный проигрыватель по умолчанию»

Назва проекту не до кінця мені зрозуміла, чому це він вдруг за замовчуванням? При вході до застосунку нас зустрічає інструкція яка допомогає засвоїти цей застосунок плані його використання, проте я особисто не бачу у цьому сенсу бо інтерфейс є інтїтивно простим та зрозумілим. Присутні можливості запису аудіо та навіть можливість наприклад обрізати аудізапис. Присутні альбоми,треки,те що прослуховувалось нещодавно, виконавці та різні режимі відображення. Аудіотреки зручно відображаются у вигляді впорядкованих «квадратиків». Інтерфейс застосунку приємний мінімалістичний.

#### 1.3.1.2 Дослідження “Music player”

Назва проекту є краткою та зрозумілою та… очевидною. Інтерфейс виглядає більш пафосно та є менш зручним проте основне є та дуже багато додаткових

фіч: плейлист, пісні,виконавці, альбом та папки. Можливо налаштувати сортування, записати аудіозапис та виконати пошук Є аудокниги  можливість нарізати рингтони. Є єквалайзер Присутня реклама яку можна відключити. Також можна сканувати медіа Є таймер. Дуже багатий функціонал у цілому. Цей додаток підходить для тих хто шукає широку функціональність та можливості та не підходить тим хто хоче простий та швидкий для основних задач додаток.

#### 1.3.1.3 Дослідження Pulsar

Назва проекту є цікавою та унікальную – музика як пульс. Застосунок має нетипиве розташування панелі с альбомами, виконавцями, папками, жанрами, та композиціями – зверху. Пульсар має значно меншу функціональність ніж попередній варіант не можна наприклад обрізати для рингтону аудіозапис. Є можливість встановити тему, встановити чергу прослуховування аудіо. Є таймер сну та єквалайзер. У цілому застосунок непоганий і має хорошу назву та збалансований дизайн, проте значно поступається попередньо розглянотому додатку по функціональності

#### 1.3.1.4 Висновок

Всі додатки мають складний та «нафарширований» інтерфейс. Проте чи потрібно воно? Можливо користувачу краще дати просте та швидке у

використанні рішення. Краще створити  зручний та зрозумілий мінімалистичний та зручний інтерфейс без зайвого.

## 1.4 Платформа

### 1.4.1 Огляд інструментів розробки

#### 1.4.1.1 XCode

Створено в 2013 році компанією Apple. Вся розробка в XCode відбувається в одному вікні. Apple LLVM ловить помилки "на ходу". У нас також є зручна панель для переміщення між файлами. Мови програмування Swift та Objective C. Середовище вдосконалене. Розробник матиме доступ до розробленої документації. Громада розвинена і допоможе вирішити складні питання. Interface Builder пришвидшує розробку проекту. Проблема полягає у необхідності пристрою з MacOS, але це питання можна вирішити за допомогою віртуальних машин, але це незручно. Недоліками з точки зору публікації є вартість $ 99 і складність. Інструменти розробки мають широкі можливості для висвітлення більшої кількості питань.

#### 1.4.1.2 Android studio

Створено в 2013 році Google. Середовище розвинене і вільне. Середовище підходить як для командного, так і для самостійного розвитку. Ми маємо

розробляти Android, Android TV, Android Wear, Android Auto, Glass у цьому середовищі. Існує можливість зручної роботи з GIT або іншими системами контролю версій. Мови програмування - Java, Kotlin та C ++. Розробник матиме доступ до розробленої документації. Громада розвинена і допоможе вирішити складні питання. Зручний фреймворк і багата функціональність. Багато плагінів. Великий вибір пристроїв, однак, якщо ви розробляєте для Android, важко зробити програму успішною для кожного пристрою. Недоліком є те, що для роботи з Android Studio вам потрібен потужний ПК, оскільки навколишнє середовище є вимогливим.

### 1.4.2 Огляд комерційної частини

Вважається, що розробляти під iOS вигідніше, ніж під Android. Той факт, що ринок - це щось і кожен має певний потенціал. Існує кілька статистичних даних щодо цього: Середній дохід 100 найкращих розробників становив 84 мільйони доларів проти 51 мільйона доларів у 2019 році та 84 мільйони доларів у 2018 році проти 45 мільйонів доларів. Це незважаючи на те, що кількість користувачів на Android більше. Користувачі IOS витрачають більше грошей на вміст і зацікавлені

### 1.4.3 Результуючи…

Розробка кожної з цих операційних систем має плюси і мінуси. Під Android легше запускати і дешевше в iOS більше порядку та кращої статистики. У мене немає пристрою з відповідною ОС, тому я буду розробляти для Android.

**Цільова аудиторія**

Мобільний програвач використовується мабуть кожним користувачем смартфону вне залежності від віку та інтересів. Програвання аудіо є однією із основних функцій мобільних пристроїв.

What? Ауділрогравач з можливістю розділення аудіо

Who? Кожен користувач мобільного пристрою є по сути нашим потенційнім юзером

Where? На єкрані смартфона

When? Клієнт або побаче рекламу, або буде шукати додаток самостійно у плей маркеті або хтось йому порадить

Why? Можно отримати задоволення від прослуховування музики або прослухати щось необхідне для професійної діяльності, тощо

## 1.5 Висновок

Не дивлячись на те що було створено та завантаженно у плей маркет багато подібних програмних продуктів. через те що він буде мати зручний функціонал, хороший дизайн та буде зручним та виділятись цим на фоні конкурентів. Також можна вкластись у рекламу. Необхідно дати користувачу щось своє і йому має це сподобатись. Наш додаток має дуже широке охоплення аудиторії. На мій погляд зараз ринку погтрібне просте та зручне рішення яке я і буду реалізовувати. На ринку багато нафаршированих зайвим

більшості користувачів функціоналов який негативно впливає на зручність та поскладнює освоєння застосунку. Головне дати користувачеві якість та зручність, а не багато зайвих фіч та пафосний дизайн.

## РОЗДІЛ 2: Про застосунок

### 2.1 Що має виконувати наш додаток?

Наш додаток має надавати необхідний для прослуховування музики функціонал: надавати можливість обрати музикальний трек з альбома та прослухати його. Також можна додати розділ рекомендованих треків для розширення музикальної палітри користувача.

### 2.2 Трохи історії музикальних програвачів

Мало хто знає, що прообразом сучасних плеєрів стало міні-радіо від компанії Sony. Попит на таке аудіо обладнання був величезним у 1960-х роках. Існують різні припущення, що на послужило поштовхом до ідеї створення першого плеєра. Одне з них свідчить, що пальма першості у винаході цього чуда техніки належить засновнику компанії Sony Нобутосі Кихара.

світ побачив перший касетний плеєр Sony Walkman TPS-L2. У 1979 році, пристрій не встигнувши вступити в масовий продаж, виробляє справжній фурор на ринку. Sony Walkman TPS-L2После виходу в 1980 році перших CD-

дисків, компанія Sony заміщає касетний носій пам'яті диском. Плеєр швидко еволюціонує і в 1984 році виходить перший портативний цифровий програвач Sony Discman D-50. Касети стрімкими темпами починають витіснятися з аудіо-ринку, на зміну їм приходять CD-диски. Слово «програвач» звучить все рідше, з'являється новий термін «плеєр».

У дев'яності роки фірма Sony зробила ставку на свій аудіоформат ATRAC та новітні носії Wisp Bulletin Board. Плеєр Sony Walkman Doctor of Medicine MZ1 був значно менше за розміром в порівнянні з першими CD-моделями, але коштував безмірно великих коштів - 750 баксів.

2001 рік став переломним моментом, який похитнув всю індустрію музичних портативних програвачів, був змінений не тільки спосіб прослуховування музики, а й в дизайні плеєра відбулися істотні зміни. Після винаходу Нобутосі Кихара в 1979-му році касетного плеєра, спроб винайти більш вдосконалену модель з кожним роком ставало все більше. Компанія Apple в 2001 році випускає програвач iPod Classic, що стало поворотним моментом в історії Mp3 проігривателей.IPod Classic 2001 рік Він володів рядом переваг: синхронізація з ПК, тривалий режим роботи, підтримка декількох аудіо форматів. У 2004 році світ побачила модель iPod mini, вигляд сучасних плеєрів став гармонійним і завершеним. Пам'ять у нового iPod 4 була невеликою - чотири гігабайти. Плеєр став відтворюватися належним чином на фото і відео.

З кожним роком плеєри тільки продовжували удосконалюватися.

### 2.3 Огляд основних музикальних форматів ОС Андроїд

### 2.3.1 Для високої якості

Сюди відносяться стиснені формати. PCM - імпульсно кодова модуляція. Оригінальний аналоговий звук діскретіруется "як є", без будь-яких змін. PCM - найбільш поширений формат запису звуку, який використовується на CD і DVD дисках. Багатоканальне довбай, суррaунд, за умови якісних динаміків звучання майже один-в-один з живим виконанням. Якщо любите засісти перед домашнім кінотеатром і зануритися з співпереживання головним і другорядним героям фільму - саме те. WAV Досить древній формат, розроблений аж в 1991 році. Ну, так старі майстри завжди думали про високу якість. Багато хто вважає WAV несжатим форматом. Але насправді - це контейнер і там можуть міститися в тому числі і стислі файли. У більшості випадків WAV містить нестислий звук в форматі PCM. Тому і якість висока. Але і на одну хвилину запису витрачається близько 32MB пам'яті. Досить хороша сумісність з Windows і Mac. AIFF Аналог WAV від розробників Apple. Теж контейнер і теж містить найчастіше звук в форматі PCM. Хороша сумісність з Windows.

### 2.3.2 Для шквидкодії та економії пам’яті (з втратами)

MP3: За стандартом MPEG-1 Audio Layer 3. З'явився ще в 1993 році і миттєво завоював загальну любов саме своєю економічністю у споживанні пам'яті.

На одному CD можна зберігати всю дискографію улюбленої групи.

Кілька альбомів залити на флешку і можна насолоджуватися музикою на всьому шляху з Києва до Одеси. Можна за цей час прослухати все книги всіх письменників, гідних щоб їх слухати. Формат MP3 - це такий звуковий скопець, у якого вирізали все саме нехочу, зате почали проявлятися здатності до накопичення і економії. Так і MP3 - ну дуже економічний формат. Основна перевага - підтримується на всьому, що тільки грає і співає. AAC: Високий рівень спосіб аудіо кодування. Молодший, але просунутий брат MP3. Має злегка поліпшені звукові характеристики і велику ступінь стиснення. Застосовується на Android, iOS, iTunes, YouTube, Nintendo і останніх версіях PlayStation. Теж народний формат, але для трохи більш просунутого народу. Що і відображено в назві. OGG: Загалом, це не формат, а контейнер і, по суті, назва OGG ні про що не говорить стосовно міститься в ньому звуку. Однак найчастіше за все містить кодек Vorbis.Значно покращено якість звучання щодо інших форматів звуку з втратами при стисненні.Надається можливість при однаковій якості звучання записувати файли з меншою вагою. Ще більш економічний формат, ніж MP3. Проблема - формат OGG вільний, тому ніхто в його промоушн грошей не вкладає. Так що може підтримується далеко не скрізь і можуть виникнути несумісність. Тоді доведеться конвертувати в MP3.

### 2.4 Дивно, але факт

Армія Великобританії використовувала пісні Брітні Спірс для того, щоб відлякувати сомалійських піратів. Хоча насправді вони не є жахливими, суть у іншому)

## РОЗДІЛ 3: Додаткова інформація

### 3.1 Як застосунки замінили плеери?

Мати сотню пісень в кишені - це було шиком в 1990-і і трендом в 2000-і роки. Без перебільшення можна сказати, що портативні музичні плеєри були скоріше стилем життя, ніж пристроями для прослуховування музики. Спочатку це були касетні і CD-плеєри, а на початку 21 століття почалася епоха цифрової музики, яку можна охарактеризувати двома словосполученнями: Apple iPod і Sony Ericsson Walkman. Плеєр компанії Apple став настільки популярним, що в 2002 році продажі iPod приносили компанії близько 20% прибутку. І це якщо врахувати той факт, що тоді Apple позиціонувала себе як виробник комп'ютерів. У 2005 році серйозним конкурентом iPod став мобільний телефон з вбудованим MP3-плеєром Sony Ericsson Walkman. Як показав час, класичні мобільні телефони не могли скласти серйозну конкуренцію портативним музичним плеєрів. Тільки після виходу iPhone в 2007 році і появи перших Android-смартфонів попит на MP3-плеєри істотно впав. За статистикою, за останнє десятиліття продажу музичних плеєрів знизилися в 10 разів. Фактично на сьогоднішній день єдиним затребуваним сегментом на цьому ринку залишаються Hi-Fi пристрої.

### 3.2 Огляд медіаплеерного функціоналу Андроїд

MediaPlayer - клас, який дозволить вам програвати аудіо / відео файли з можливістю зробити паузу і перемотати в потрібну позицію. MediaPlayer вміє працювати з різними джерелами, це може бути: шлях до файлу (на SD або в інеті), адреса потоку, Uri або файл з папки res / raw.

Наступні класи використовуються для відтворення звуку та відео в рамках Android: MediaPlayer Цей клас є основним API для відтворення звуку та відео. AudioManager Цей клас керує джерелами звуку та аудіовиходом на пристрої.

Щоб відтворювати аудіо, MediaPlayer повинен знати, який саме ресурс (файл) потрібно виробляти. Встановити потрібний ресурс для відтворення можна трьома способами:

в метод create() об'єкта MediaPlayer передається id ресурсу, що представляє аудіофайл

в метод create() об'єкта MediaPlayer передається об'єкт Uri, що представляє аудіофайл

в метод setDataSource() об'єкта MediaPlayer передається повний шлях до аудіофайлу

Після установки ресурсу викликається метод prepare()або prepareAsync()(асинхронний варіант prepare ()). Цей метод готує аудіофайл до відтворення, витягуючи з нього перші секунди. Якщо ми відтворюємо файл з мережі, то краще використовувати prepareAsync (). Для управління відтворенням в класі MediaPlayer визначені наступні методи:

start(): Запускає аудіо

pause(): Призупиняє відтворення

stop(): Повністю зупиняє відтворення

Для управління гучністю звуку застосовується клас AudioManager . audioManager.setStreamVolume(AudioManager.STREAM_MUSIC, progress, 0); в якості другого параметра можна передати потрібне значення гучності.

### 3.3 Поняття мультимедіа

Термін "мультимедіа" вперше з'явився в 1965 році і активно використовувався до кінця сімдесятих років для опису екстравагантних театральних шоу того часу, де використовуються різні типи та форми інформації: слайди, фільми, відео, аудіокліпи, світлові ефекти та жива музика , як, наприклад, під час вистави вибухової пластики Неминуче 1965 - 1967 рр. В кінці 1970-х - на початку 1980-х років мультимедіа розумілася як зображення, засноване на статичних або динамічних зображеннях декількох проекторів, що супроводжуються звуком або живою музикою.

Таким чином, засоби "мультимедіа" впливали на кілька людських почуттів і подавали інформацію у різних формах: зоровій, словесній та слуховій, що створювало (і створює) глибший емоційний вплив, що, у свою чергу, принесло успіх і популярність цього на увазі театрального вистави.

Останнім часом термін "мультимедіа" стає ще більш неоднозначним. Мультимедійна компанія SCALA дає таке тлумачення сучасного поняття "мультимедіа": "Деякі з нас використовують мультимедіа технологіями, включаючи кабельне телебачення. Термін «мультимедійні мережі»

використовується для опису потужних багатомільйонних систем управління контентом, що використовуються великими корпораціями для обслуговування баз відеоданих, а також цифрових білбордів та екранів. Комп’ютерні програми для редагування домашнього відео, деякі з яких зараз коштують менше 100 доларів, також можна розглядати як мультимедійні технології. Мультимедійні технології також включають сучасні мобільні телефони, які надсилають фотографії з голосовими підписами. Термін "мультимедіа" все ще розвивається, і в міру появи та використання нових технологій він буде включати всі нові поняття.

Сьогодні "мультимедіа" означає:

· Мультимедіа - англійська. мультимедіа з лат. multum - багато, а media - середньоклітинна; засоби), електронний носій, який включає кілька типів (текст, зображення, анімація тощо) "(Великий енциклопедичний словник).

· Мультимедіа - одночасне використання різних форм подання інформації та її обробка в одному об’єкті-контейнері. Наприклад, один об’єкт-контейнер (англ. Container) може містити текстову, звукову, графічну та відеоінформацію, а також, можливо, спосіб взаємодії з ним. Термін мультимедіа також ... використовується для позначення носіїв інформації, що дозволяють зберігати великі обсяги даних і забезпечують досить швидкий доступ до них (першими носіями цього типу були CD-ROM). У цьому випадку термін мультимедіа означає, що комп'ютер може використовувати засоби масової інформації та надавати інформацію користувачеві через усі можливі типи даних, такі як аудіо, відео, анімація, зображення та інші, на додаток до традиційних способів надання інформації, таких як текст " (Вікіпедія).

· Мультимедіа - це комп’ютерні системи, які забезпечують інтегративний доступ до різноманітної інформації за допомогою стимулювання людських почуттів за допомогою цифрових технологій (Джордж Тейлор, Університет Ньюкасла, Великобританія, Департамент геоматики Університету Ньюкасла на Тайн, Великобританія)

Мультимедійні дані інтегрують:

· Зображення, відео та графіки (статичні та анімовані), включаючи растрові та векторні, карти, фотографії;

· Текст у різних формах, включаючи літери та цифри бази даних;

· Звуковий;

· Найближчим часом - запах і смак.

Мультимедійні технології - одна з найбільш перспективних і популярних галузей інформатики. Вони прагнуть створити продукт, що містить "колекції зображень, тексту та даних, що супроводжуються звуком, відео, анімацією та іншими візуальними ефектами (моделювання), включаючи інтерактивний інтерфейс та інші елементи керування". Це визначення було сформульовано в 1988 р. Найбільшою Європейською комісією, яка займається впровадженням та використанням нових технологій. Інтерактивність - здатність реагувати на дії користувача, включаючи управління користувачем. Області застосування:

- Навчання з використанням комп'ютерних технологій (науково-просвітницька або освітня сфера);

- відеоенциклопедії, інтерактивні путівники, тренажери, ситуаційно-рольові ігри та ін .;

- Інформаційна та рекламна служба;

- Популяризаторська і розважальна сфери;

- Інтернет-мовлення;

- Розваги, ігри, системи віртуальної реальності;

- Презентаційна (вітринної реклами), ЗМІ;

- Творчість (станція мультимедіа стає незамінним авторським інструментом в кіно і Відеомистецтво. Автор фільму за екраном такої настільної системи збирає, «аранжує», пише твори з заздалегідь підготовлених

## РОЗДІЛ 4: Єтапи реалізації плеера на ОС Android

1)Створення або пошук зображень для графічних елементів

2)Створення головного меню

3)Створення інтерфейсу та функціоналу Альбомів

4)Створення інтерфейсу та функціоналу Рекомендованого

5)Створення інтерфейсу та функціоналу програвання музики

6)Тестування

## РОЗДІЛ 5: Тонкощі реалізації

*Рисунок 1. Демонстрація ігрового процесу*

Далі буде надано пояснення класів та основного файлу розмітки.

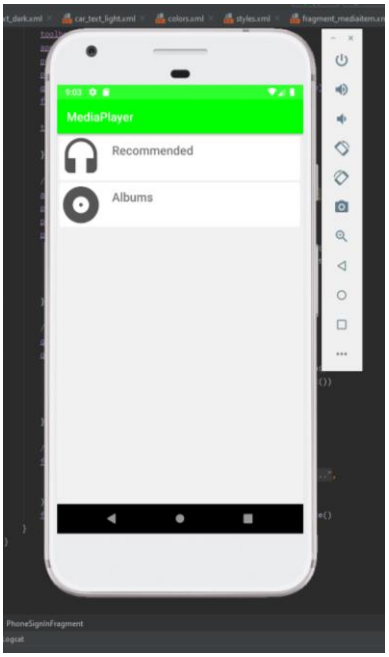



*Рисунок 1 Працююча програма(головне меню)*

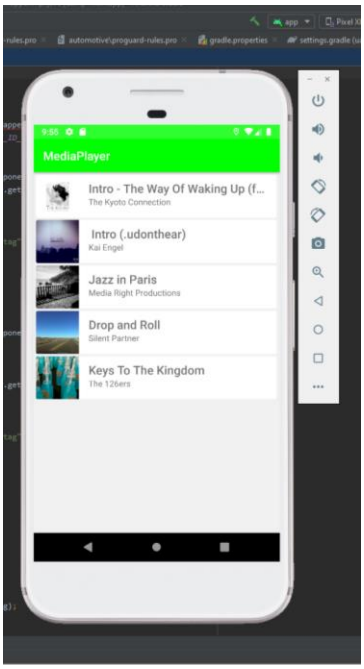



*Рисунок 2 Працююча програма(рекомендоване)*

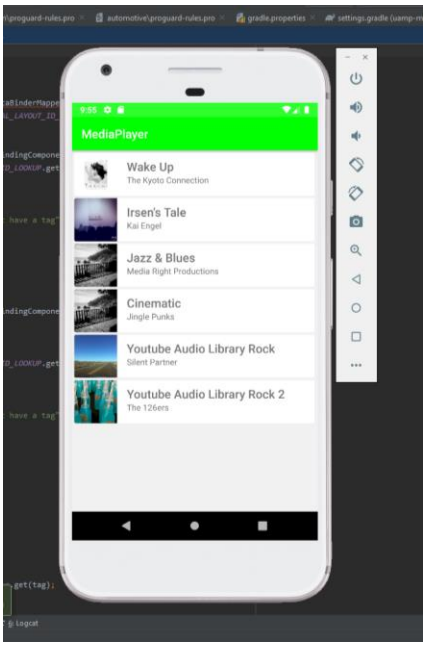



*Рисунок 3 Працююча програма(альбоми)*

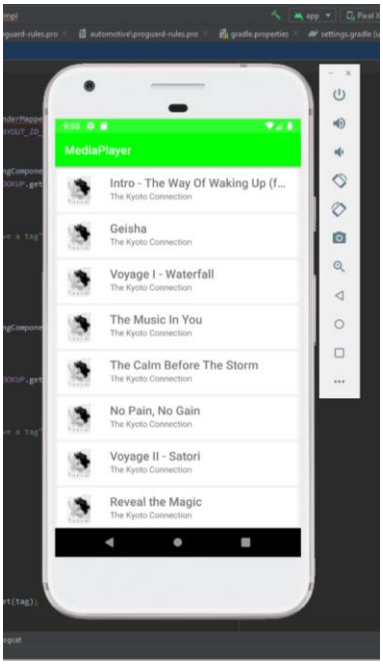



*Рисунок 4 Працююча програма(треки альбому)*

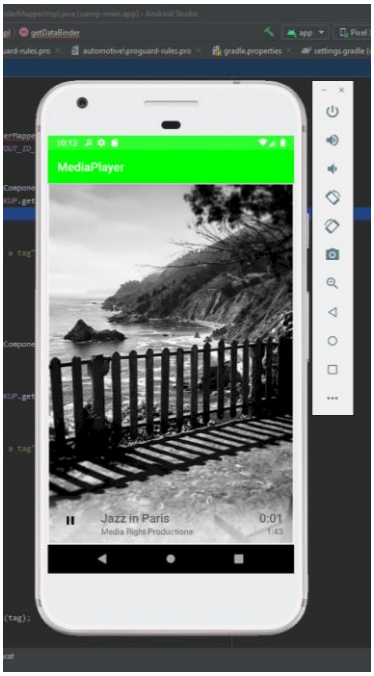



*Рисунок 5 Працююча програма(програвання)*

## 5.1 Інтерфейси, класи та інше

### 5.1.1 PhoneSignInFragment

Фрагмент, який використовується для полегшення входу на телефон. Фрагмент дозволяє користувачам вибирати між  процес входу PIN-коду або QR-коду.

### 5.1.2 PinCodeSignInFragment

Фрагмент, який використовується для полегшення входу в PIN-код. Цей фрагмент відображається з можливістю налаштування * PIN-код, який користувачі вводять на додатковому пристрої для входу. * * <p> Цей екран служить демонстрацією найкращих практик користувацького інтерфейсу для входу з PIN-кодом. Реалізація входу * стосуватиметься додатків і не входить у вартість.

### 5.1.3 QrCodeSignInFragment

Фрагмент, який використовується для полегшення входу в QR-код. Користувачі сканують QR-код, наданий цим  фрагмент на своїх телефонах, який виконує автентифікацію, необхідну для входу.

### 5.1.4 SettingsActivity

Цей клас надає налаштування програми для інтеграції з MediaCenter в Android Automotive.

### 5.1.5 SettingsFragment

Обробляє події з різними змінами налаштувань

### 5.1.6 SignInLandingPageFragment

Фрагмент, який відображає екран посадки для потоку входу. Цей екран можна налаштувати  для відображення сторонніх входів, входу за допомогою PIN-коду, входу в QR-код та / або входу через Google.

### 5.1.7 SignInLandingPageFragment

Фрагмент, який використовується для спрощення входу в ім’я користувача та пароль.

### 5.1.8 UsernameAndPasswordSignInFragment

Фрагмент, який використовується для спрощення входу в ім’я користувача та пароль.

### 5.1.9 MediaItemFragment

Фрагмент, що представляє список MediaItems.

### 5.1.20 NowPlayingFragment

Фрагмент, що відображає поточний елемент медіа, що відтворюється.

### 5.1.21 Event

Використовується як обгортка для даних, які відображаються через LiveData, що представляє подію.

### 5.1.21 InjectorUtils

Статичні методи, що використовуються для введення класів, необхідних для різних видів діяльності та фрагментів.

### 5.1.22 MediaItemData

Клас даних для інкапсуляції властивостей

## РОЗДІЛ 6: Джерела

Основні:

https://www.istockphoto.com/ru

https://habr.com/ru/

https://www.google.com/

https://hightech.fm/2019/06/25/ios-vs-android

http://um.co.ua

ukrbukva.net

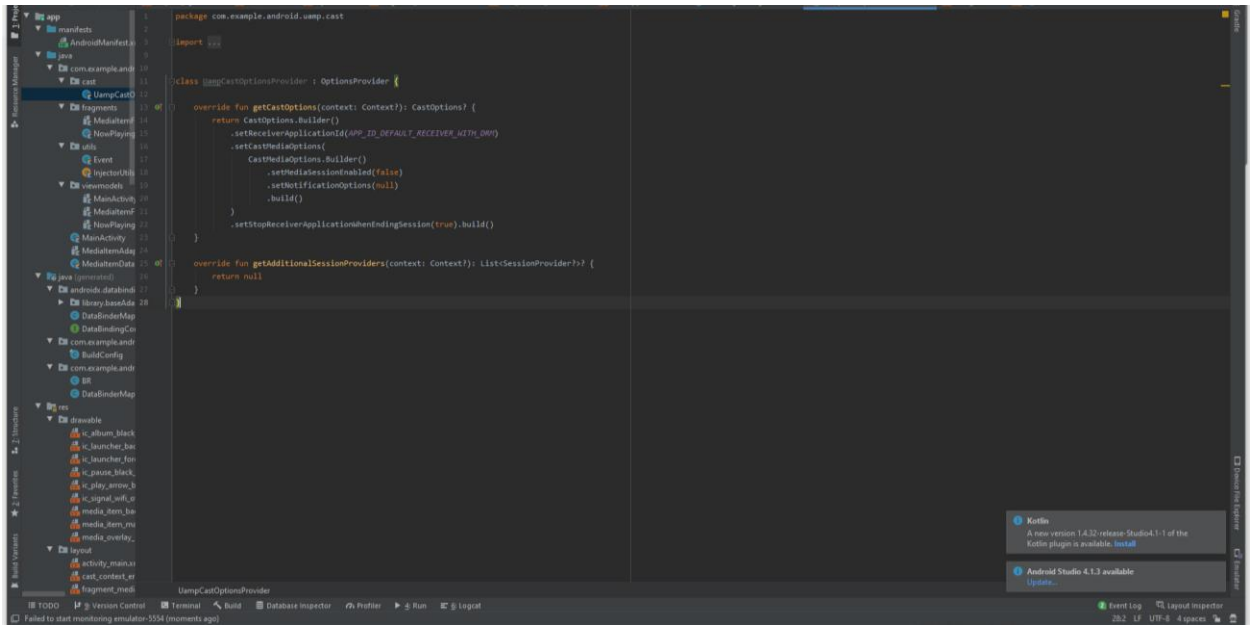
undio.org.ua

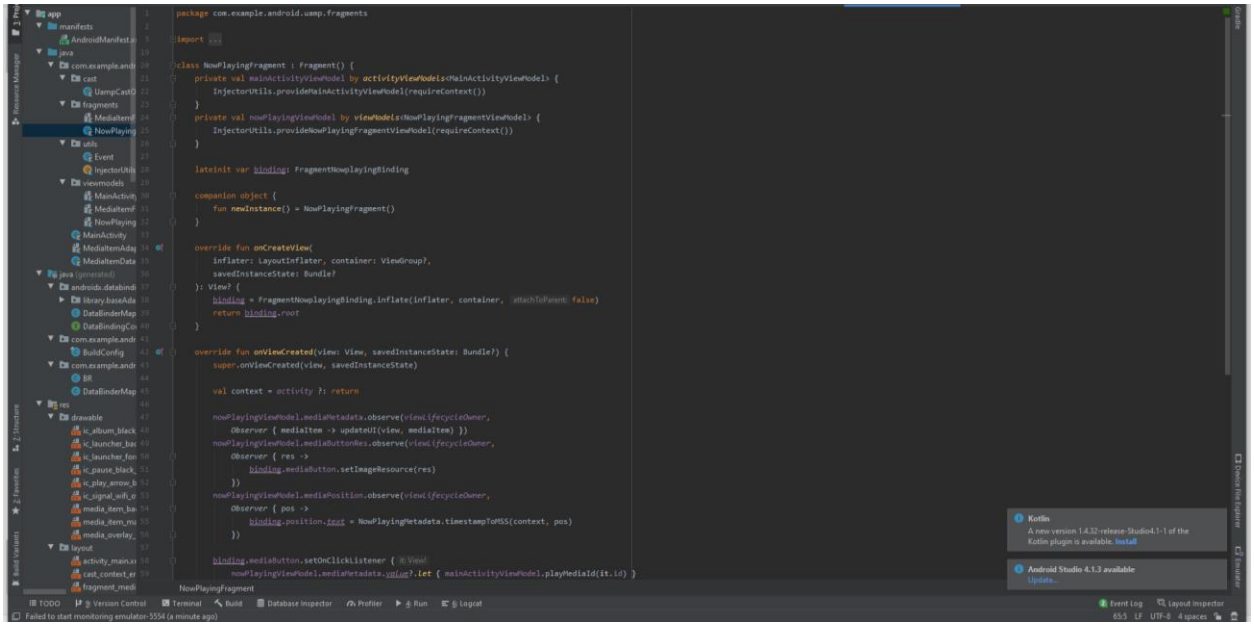
xreferat.combibrary.nlu.edu.ua

estetika.etica.in.ua

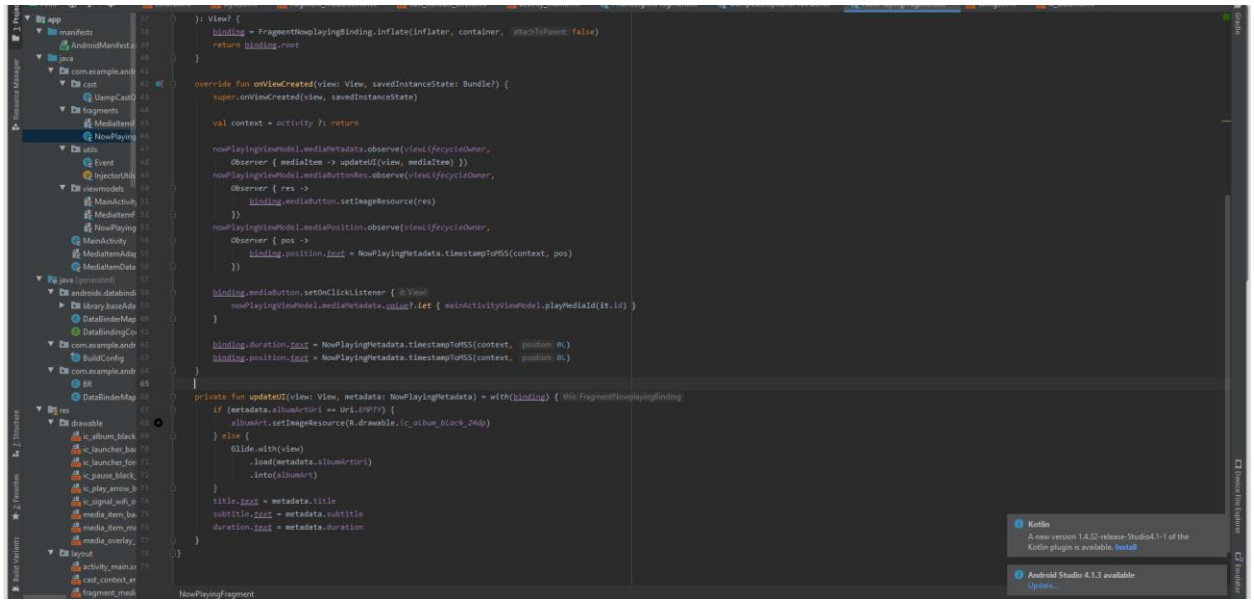
## РОЗДІЛ 7: Додатки

Код програми:

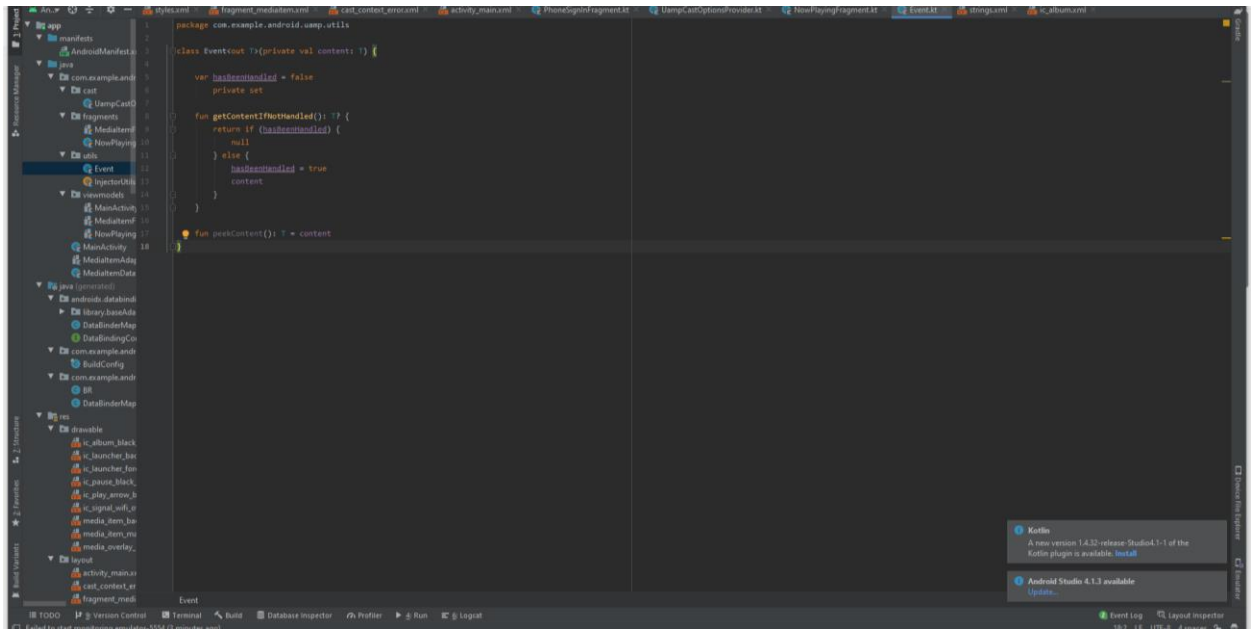




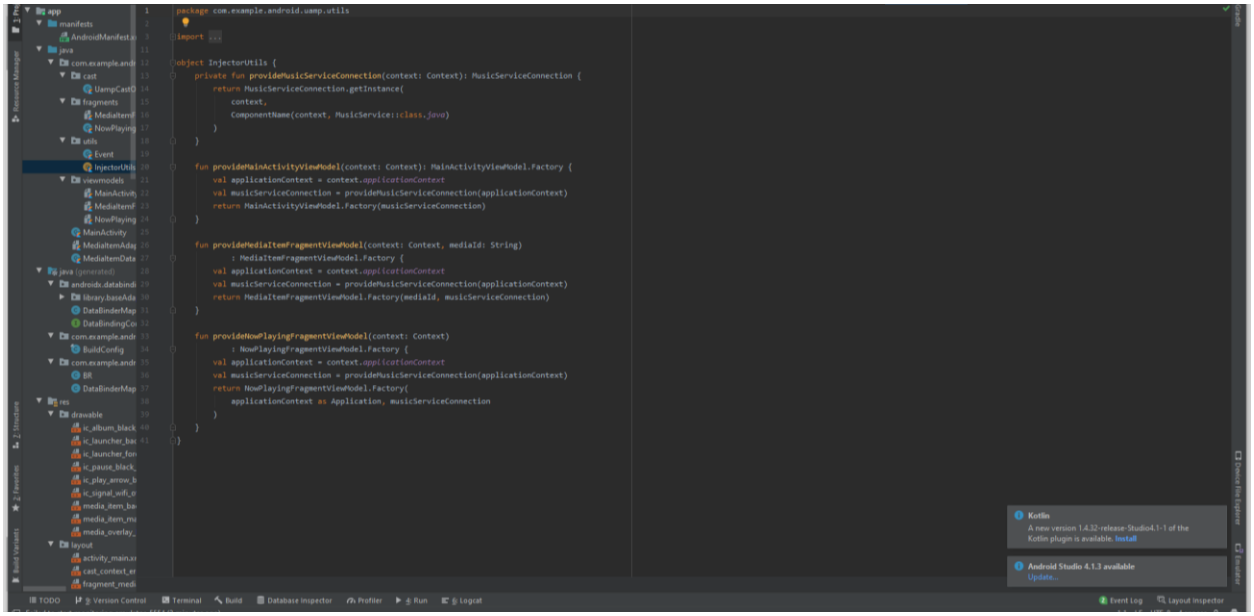
```
package com.example.android.uamp.fragments

import ...

class NowPlayingFragment : Fragment() {
    private val mainActivityViewModel by activityViewModels<MainActivityViewModel> {
        InjectorUtils.provideMainActivityViewModel(requireContext())
    }
    private val nowPlayingViewModel by viewModels<NowPlayingFragmentViewModel> {
        InjectorUtils.provideNowPlayingFragmentViewModel(requireContext())
    }

    lateinit var binding: FragmentNowplayingBinding

    companion object {
        fun newInstance() = NowPlayingFragment()
    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        binding = FragmentNowplayingBinding.inflate(inflater, container, attachToParent: false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        val context = activity ?: return

        nowPlayingViewModel.mediaMetadata.observe(viewLifecycleOwner,
            Observer { mediaItem -> updateUI(view, mediaItem) })
        nowPlayingViewModel.mediaButtonRes.observe(viewLifecycleOwner,
            Observer { res ->
                binding.mediaButton.setImageResource(res)
            })
        nowPlayingViewModel.mediaPosition.observe(viewLifecycleOwner,
            Observer { pos ->
                binding.position.text = NowPlayingMetadata.timestampToMSS(context, pos)
            })

        binding.mediaButton.setOnClickListener { it: View!
            nowPlayingViewModel.mediaMetadata.value?.let { mainActivityViewModel.playMediaId(it.id) }
        }
```



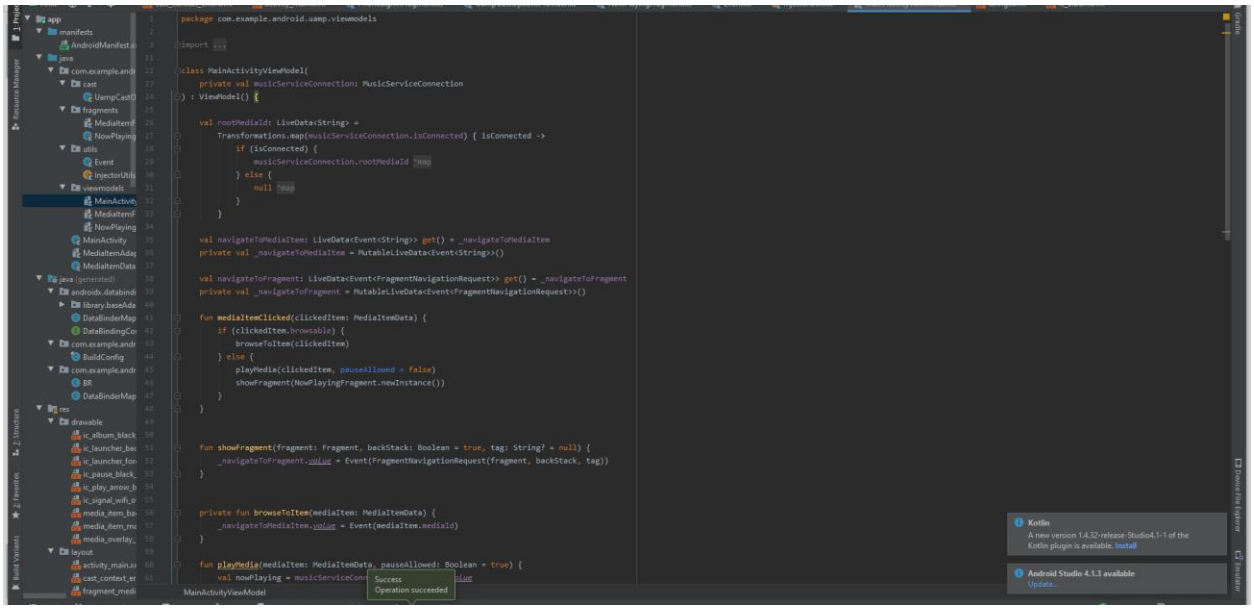
```
    ): View? {
        binding = FragmentNowplayingBinding.inflate(inflater, container, attachToParent: false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        val context = activity ?: return

        nowPlayingViewModel.mediaMetadata.observe(viewLifecycleOwner,
            Observer { mediaItem -> updateUI(view, mediaItem) })
        nowPlayingViewModel.mediaButtonRes.observe(viewLifecycleOwner,
            Observer { res ->
                binding.mediaButton.setImageResource(res)
            })
        nowPlayingViewModel.mediaPosition.observe(viewLifecycleOwner,
            Observer { pos ->
                binding.position.text = NowPlayingMetadata.timestampToMSS(context, pos)
            })

        binding.mediaButton.setOnClickListener { it: View!
            nowPlayingViewModel.mediaMetadata.value?.let { mainActivityViewModel.playMediaId(it.id) }
        }

        binding.duration.text = NowPlayingMetadata.timestampToMSS(context, position: 0L)
        binding.position.text = NowPlayingMetadata.timestampToMSS(context, position: 0L)
    }

    private fun updateUI(view: View, metadata: NowPlayingMetadata) = with(binding) { this: FragmentNowplayingBinding
        if (metadata.albumArtUri == Uri.EMPTY) {
            albumArt.setImageResource(R.drawable.ic_album_black_24dp)
        } else {
            Glide.with(view)
                .load(metadata.albumArtUri)
                .into(albumArt)
        }
        title.text = metadata.title
        subtitle.text = metadata.subtitle
        duration.text = metadata.duration
    }
}
```



```
package com.example.android.uamp.utils

class Event<out T>(private val content: T) {

    var hasBeenHandled = false
        private set

    fun getContentIfNotHandled(): T? {
        return if (hasBeenHandled) {
            null
        } else {
            hasBeenHandled = true
            content
        }
    }

    fun peekContent(): T = content
}
```



```
package com.example.android.uamp.utils

import ...

object InjectorUtils {
    private fun provideMusicServiceConnection(context: Context): MusicServiceConnection {
        return MusicServiceConnection.getInstance(
            context,
            ComponentName(context, MusicService::class.java)
        )
    }

    fun provideMainActivityViewModel(context: Context): MainActivityViewModel.Factory {
        val applicationContext = context.applicationContext
        val musicServiceConnection = provideMusicServiceConnection(applicationContext)
        return MainActivityViewModel.Factory(musicServiceConnection)
    }

    fun provideMediaItemFragmentViewModel(context: Context, mediaId: String)
            : MediaItemFragmentViewModel.Factory {
        val applicationContext = context.applicationContext
        val musicServiceConnection = provideMusicServiceConnection(applicationContext)
        return MediaItemFragmentViewModel.Factory(mediaId, musicServiceConnection)
    }

    fun provideNowPlayingFragmentViewModel(context: Context)
            : NowPlayingFragmentViewModel.Factory {
        val applicationContext = context.applicationContext
        val musicServiceConnection = provideMusicServiceConnection(applicationContext)
        return NowPlayingFragmentViewModel.Factory(
            applicationContext as Application, musicServiceConnection
        )
    }
}
```

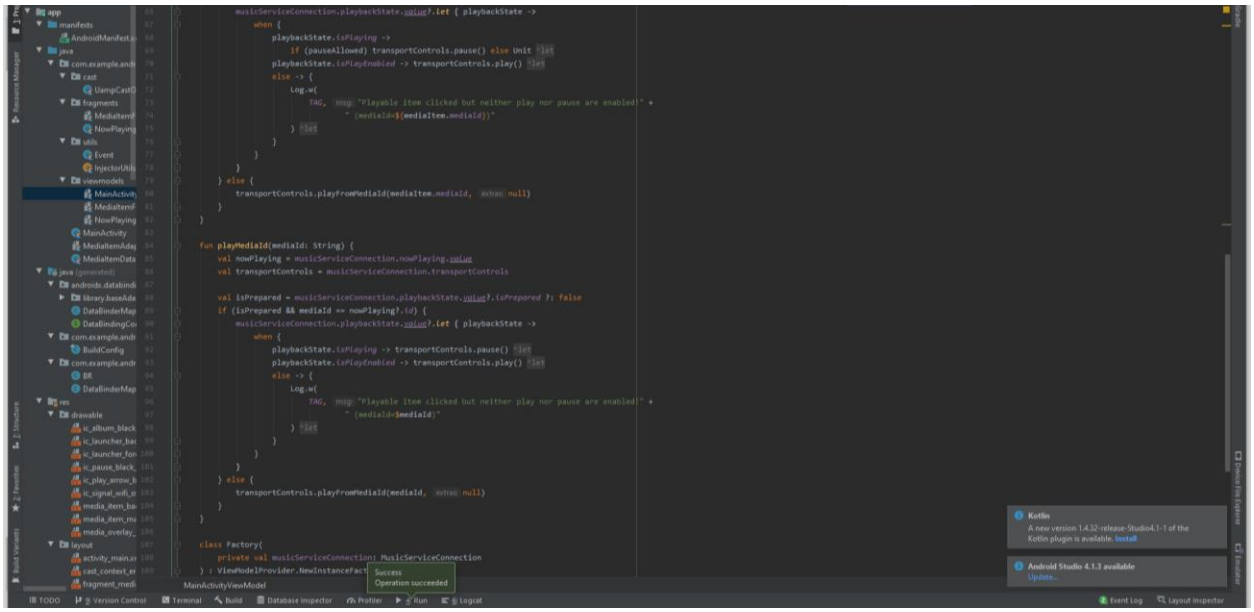


```
package com.example.android.uamp.viewmodels

import ...

class MainActivityViewModel(
    private val musicServiceConnection: MusicServiceConnection
) : ViewModel() {

    val rootMediaId: LiveData<String> =
        Transformations.map(musicServiceConnection.isConnected) { isConnected ->
            if (isConnected) {
                musicServiceConnection.rootMediaId ?: ""
            } else {
                null ?: ""
            }
        }

    val navigateToMediaItem: LiveData<Event<String>> get() = _navigateToMediaItem
    private val _navigateToMediaItem = MutableLiveData<Event<String>>()

    val navigateToFragment: LiveData<Event<FragmentNavigationRequest>> get() = _navigateToFragment
    private val _navigateToFragment = MutableLiveData<Event<FragmentNavigationRequest>>()

    fun mediaItemClicked(clickedItem: MediaItemData) {
        if (clickedItem.browsable) {
            browseToItem(clickedItem)
        } else {
            playMedia(clickedItem, pauseAllowed = false)
            showFragment(NowPlayingFragment.newInstance())
        }
    }


    fun showFragment(fragment: Fragment, backStack: Boolean = true, tag: String? = null) {
        _navigateToFragment.value = Event(FragmentNavigationRequest(fragment, backStack, tag))
    }


    private fun browseToItem(mediaItem: MediaItemData) {
        _navigateToMediaItem.value = Event(mediaItem.mediaId)
    }

    fun playMedia(mediaItem: MediaItemData, pauseAllowed: Boolean = true) {
        val nowPlaying = musicServiceConn                                    lue
```

Success
Operation succeeded

Kotlin
A new version 1.4.32-release-Studio4.1-1 of the Kotlin plugin is available. Install

Android Studio 4.1.3 available
Update...

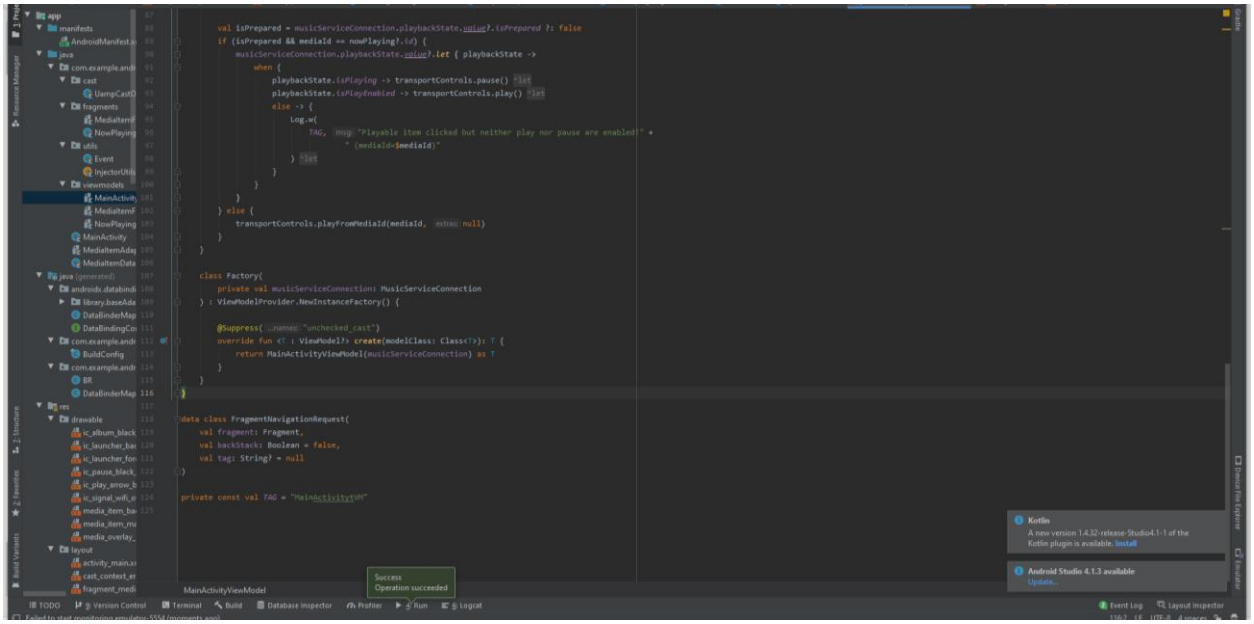


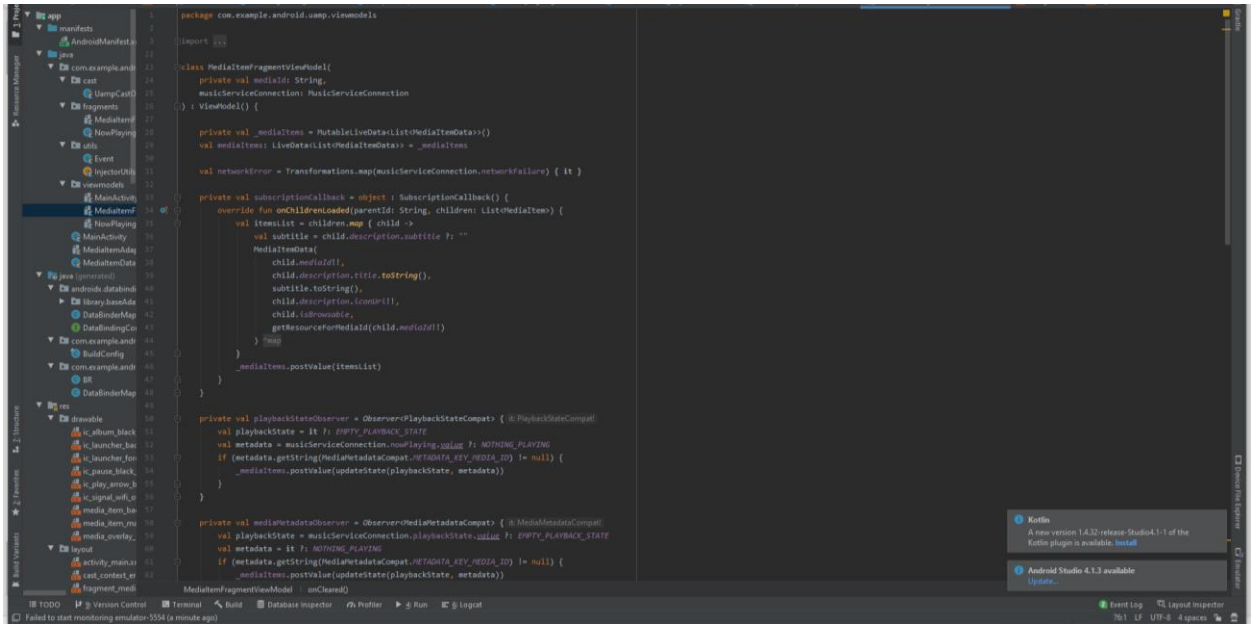
```
            musicServiceConnection.playbackState.value?.let { playbackState ->
                when {
                    playbackState.isPlaying ->
                        if (pauseAllowed) transportControls.pause() else Unit ?:let
                    playbackState.isPlayEnabled -> transportControls.play() ?:let
                    else -> {
                        Log.w(
                            TAG, msg: "Playable item clicked but neither play nor pause are enabled!" +
                                    " (mediaId=${mediaItem.mediaId})"
                        ) ?:let
                    }
                }
            }
        ) else {
            transportControls.playFromMediaId(mediaItem.mediaId, extras: null)
        }
    }

    fun playMediaId(mediaId: String) {
        val nowPlaying = musicServiceConnection.nowPlaying.value
        val transportControls = musicServiceConnection.transportControls

        val isPrepared = musicServiceConnection.playbackState.value?.isPrepared ?: false
        if (isPrepared && mediaId == nowPlaying?.id) {
            musicServiceConnection.playbackState.value?.let { playbackState ->
                when {
                    playbackState.isPlaying -> transportControls.pause() ?:let
                    playbackState.isPlayEnabled -> transportControls.play() ?:let
                    else -> {
                        Log.w(
                            TAG, msg: "Playable item clicked but neither play nor pause are enabled!" +
                                    " (mediaId=$mediaId)"
                        ) ?:let
                    }
                }
            }
        } else {
            transportControls.playFromMediaId(mediaId, extras: null)
        }
    }

    class Factory(
        private val musicServiceConnection: MusicServiceConnection
    ) : ViewModelProvider.NewInstanceFact                                    ion
```

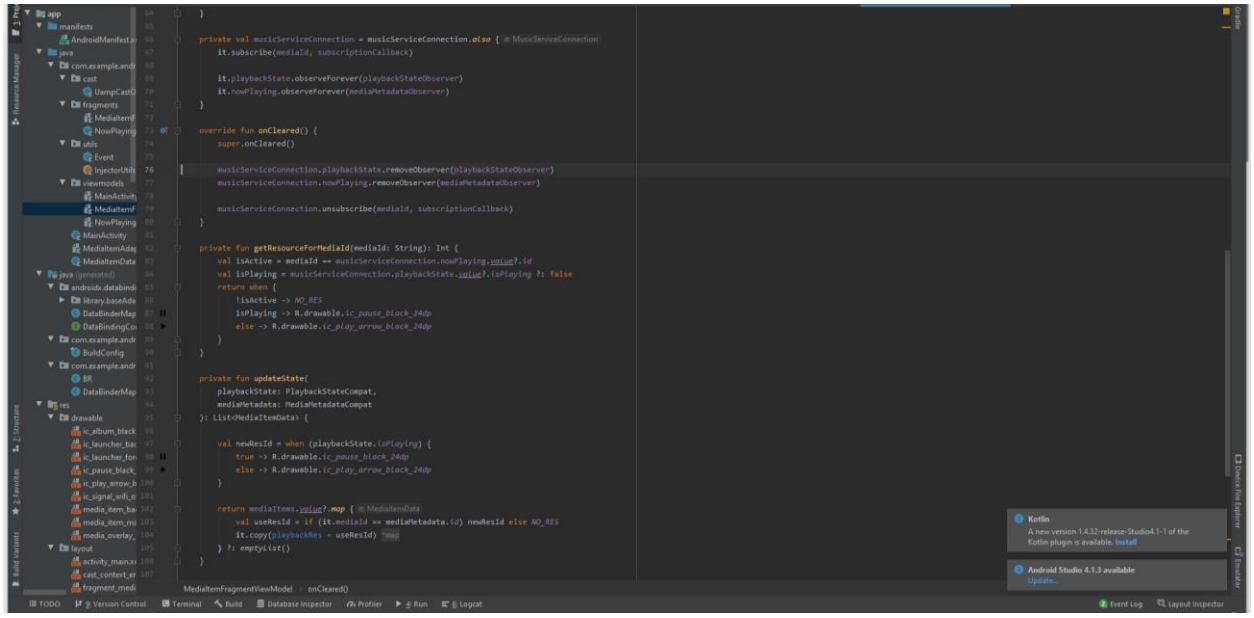
Success
Operation succeeded

Kotlin
A new version 1.4.32-release-Studio4.1-1 of the Kotlin plugin is available. Install

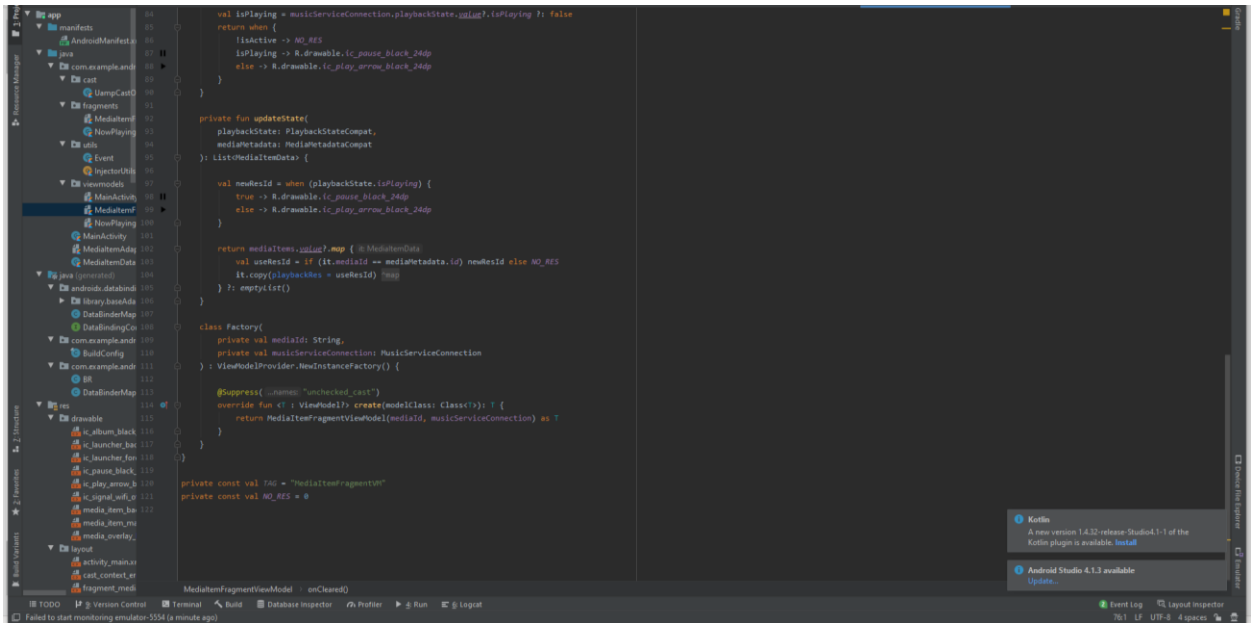
Android Studio 4.1.3 available
Update...



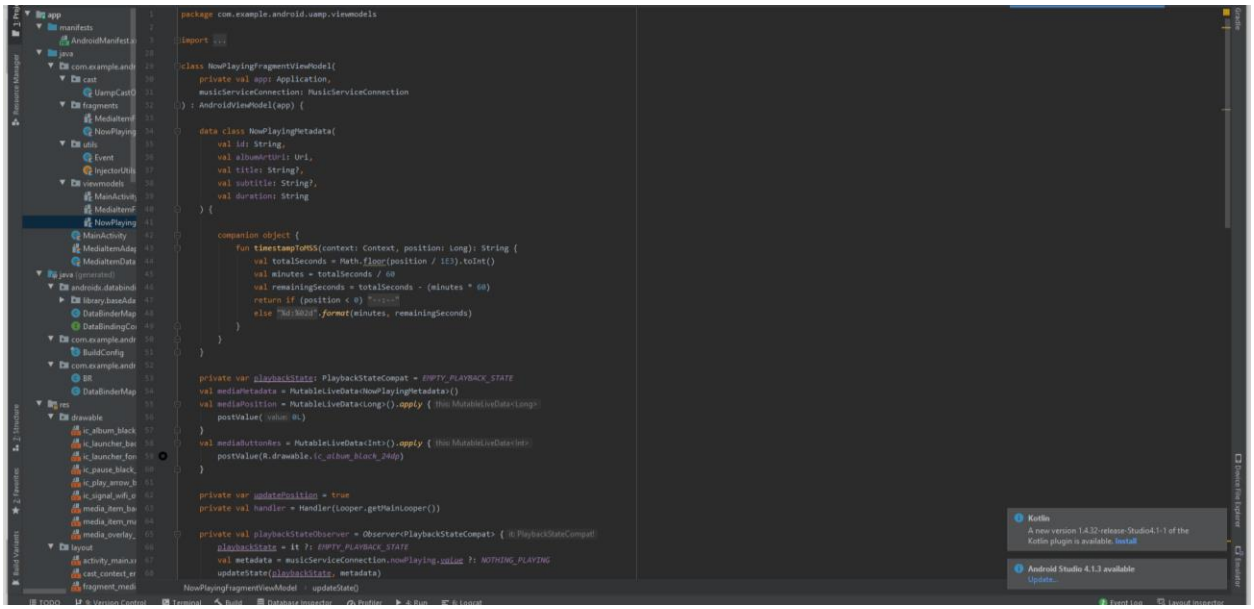




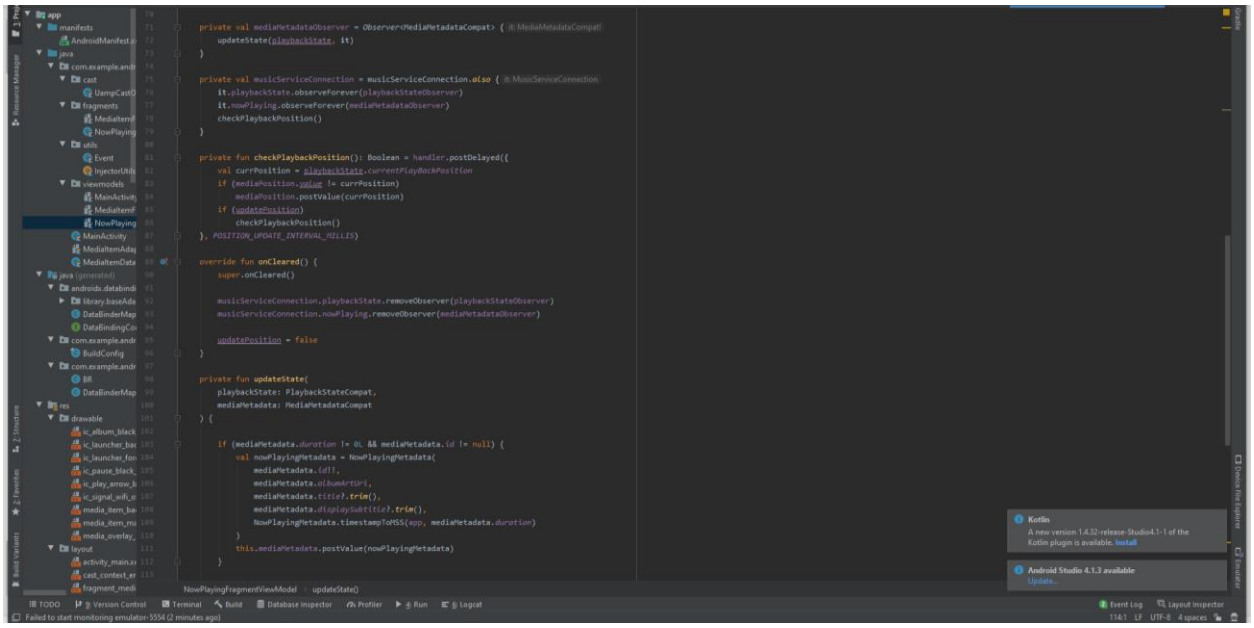
```
    }

    private val musicServiceConnection = musicServiceConnection.also { it: MusicServiceConnection
        it.subscribe(mediaId, subscriptionCallback)

        it.playbackState.observeForever(playbackStateObserver)
        it.nowPlaying.observeForever(mediaMetadataObserver)
    }

    override fun onCleared() {
        super.onCleared()

        musicServiceConnection.playbackState.removeObserver(playbackStateObserver)
        musicServiceConnection.nowPlaying.removeObserver(mediaMetadataObserver)

        musicServiceConnection.unsubscribe(mediaId, subscriptionCallback)
    }

    private fun getResourceForMediaId(mediaId: String): Int {
        val isActive = mediaId == musicServiceConnection.nowPlaying.value?.id
        val isPlaying = musicServiceConnection.playbackState.value?.isPlaying ?: false
        return when {
            !isActive -> NO_RES
            isPlaying -> R.drawable.ic_pause_black_24dp
            else -> R.drawable.ic_play_arrow_black_24dp
        }
    }

    private fun updateState(
        playbackState: PlaybackStateCompat,
        mediaMetadata: MediaMetadataCompat
    ): List<MediaItemData> {

        val newResId = when (playbackState.isPlaying) {
            true -> R.drawable.ic_pause_black_24dp
            else -> R.drawable.ic_play_arrow_black_24dp
        }

        return mediaItems.value?.map { it: MediaItemData
            val useResId = if (it.mediaId == mediaMetadata.id) newResId else NO_RES
            it.copy(playbackRes = useResId) *map
        } ?: emptyList()
    }
```



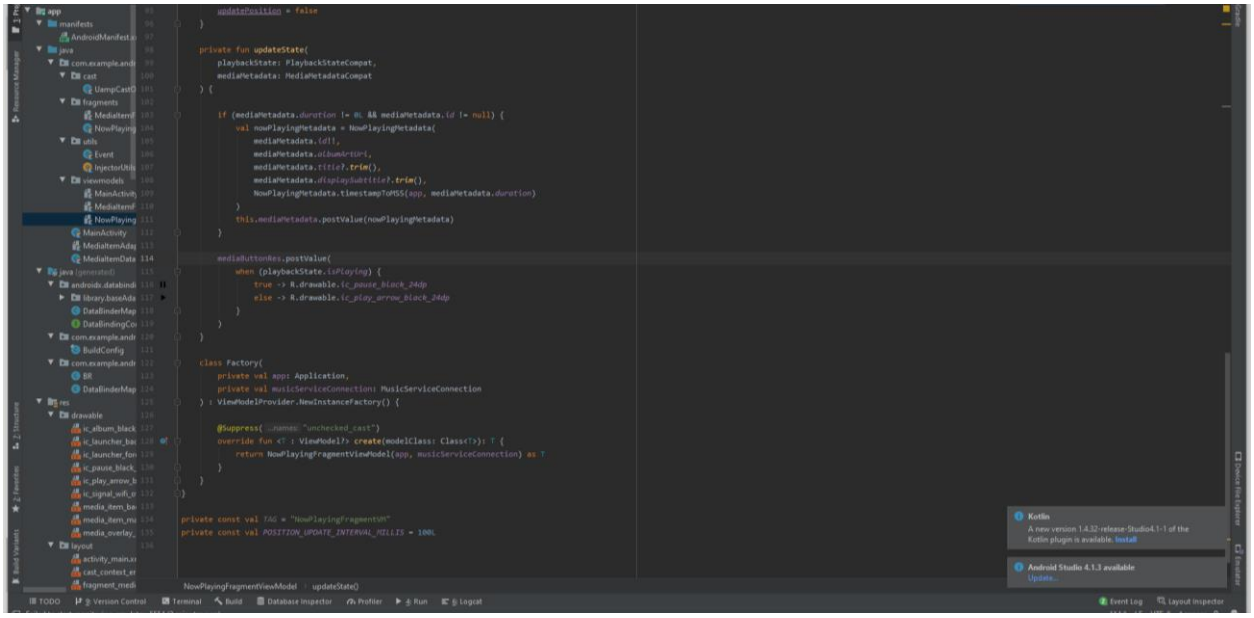
```
        val isPlaying = musicServiceConnection.playbackState.value?.isPlaying ?: false
        return when {
            !isActive -> NO_RES
            isPlaying -> R.drawable.ic_pause_black_24dp
            else -> R.drawable.ic_play_arrow_black_24dp
        }
    }

    private fun updateState(
        playbackState: PlaybackStateCompat,
        mediaMetadata: MediaMetadataCompat
    ): List<MediaItemData> {

        val newResId = when (playbackState.isPlaying) {
            true -> R.drawable.ic_pause_black_24dp
            else -> R.drawable.ic_play_arrow_black_24dp
        }

        return mediaItems.value?.map { it: MediaItemData
            val useResId = if (it.mediaId == mediaMetadata.id) newResId else NO_RES
            it.copy(playbackRes = useResId) *map
        } ?: emptyList()
    }

    class Factory(
        private val mediaId: String,
        private val musicServiceConnection: MusicServiceConnection
    ) : ViewModelProvider.NewInstanceFactory() {

        @Suppress( ...names: "unchecked_cast")
        override fun <T : ViewModel?> create(modelClass: Class<T>): T {
            return MediaItemFragmentViewModel(mediaId, musicServiceConnection) as T
        }
    }
}

private const val TAG = "MediaItemFragmentVM"
private const val NO_RES = 0
```

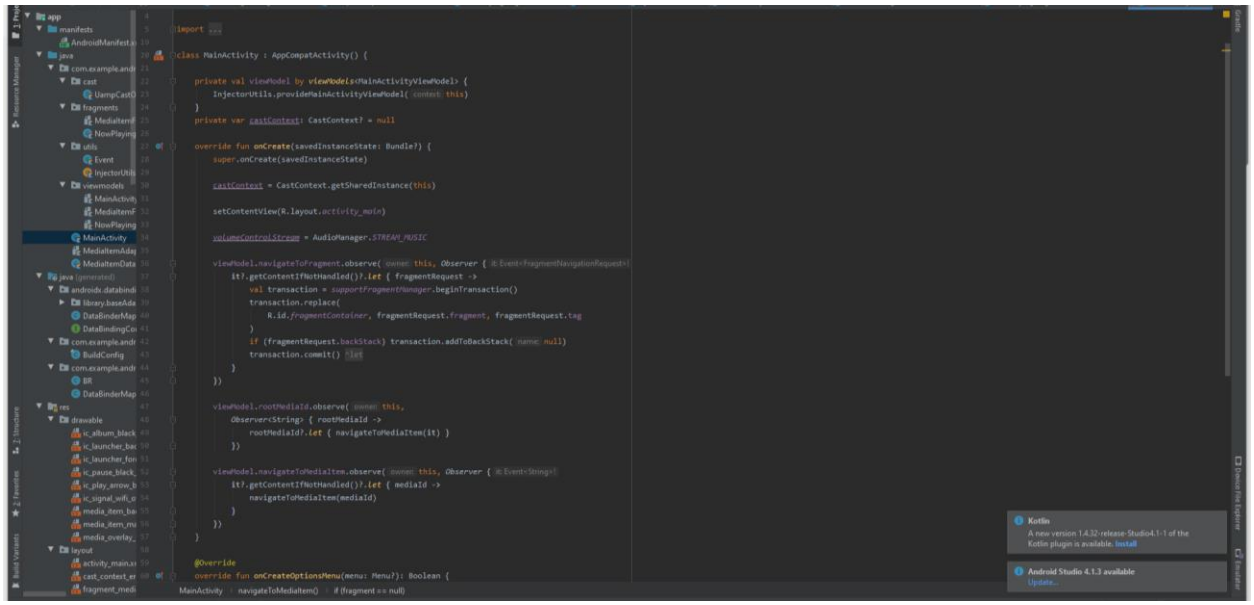


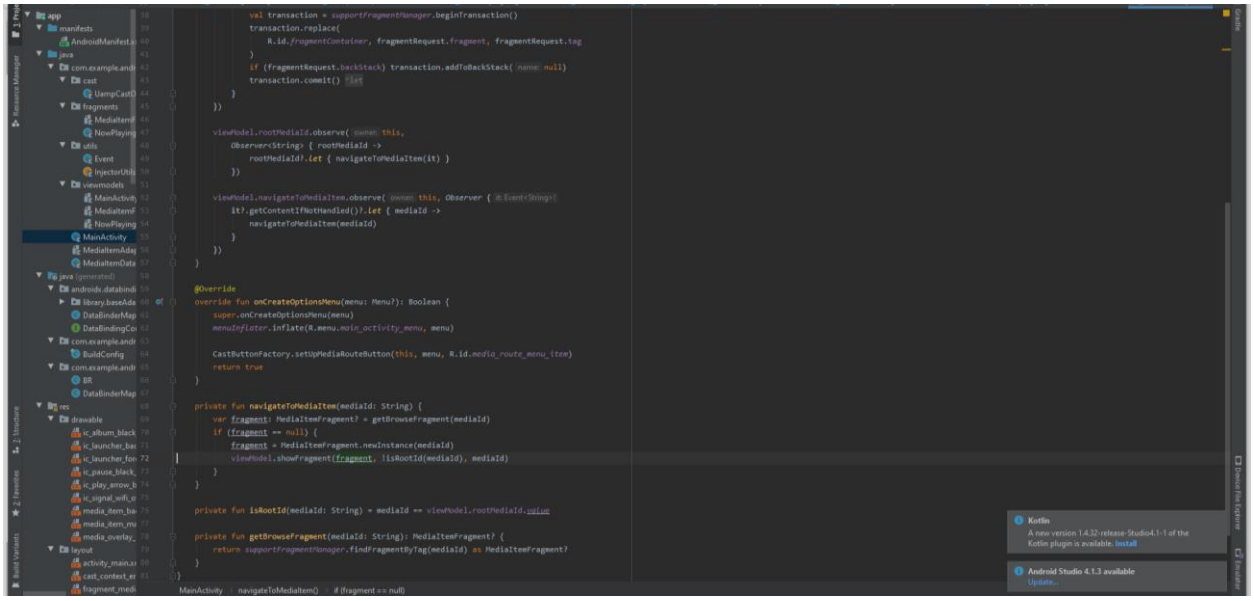
```
package com.example.android.uamp.viewmodels

import ...

class NowPlayingFragmentViewModel(
    private val app: Application,
    musicServiceConnection: MusicServiceConnection
) : AndroidViewModel(app) {

    data class NowPlayingMetadata(
        val id: String,
        val albumArtUri: Uri,
        val title: String?,
        val subtitle: String?,
        val duration: String
    ) {

        companion object {
            fun timestampToMSS(context: Context, position: Long): String {
                val totalSeconds = Math.floor(position / 1E3).toInt()
                val minutes = totalSeconds / 60
                val remainingSeconds = totalSeconds - (minutes * 60)
                return if (position < 0) "--:--"
                else "%d:%02d".format(minutes, remainingSeconds)
            }
        }
    }

    private var playbackState: PlaybackStateCompat = EMPTY_PLAYBACK_STATE
    val mediaMetadata = MutableLiveData<NowPlayingMetadata>()
    val mediaPosition = MutableLiveData<Long>().apply { this: MutableLiveData<Long>
        postValue( value: 0L)
    }
    val mediaButtonRes = MutableLiveData<Int>().apply { this: MutableLiveData<Int>
        postValue(R.drawable.ic_album_black_24dp)
    }

    private var updatePosition = true
    private val handler = Handler(Looper.getMainLooper())

    private val playbackStateObserver = Observer<PlaybackStateCompat> { it: PlaybackStateCompat!
        playbackState = it ?: EMPTY_PLAYBACK_STATE
        val metadata = musicServiceConnection.nowPlaying.value ?: NOTHING_PLAYING
        updateState(playbackState, metadata)
    }
```

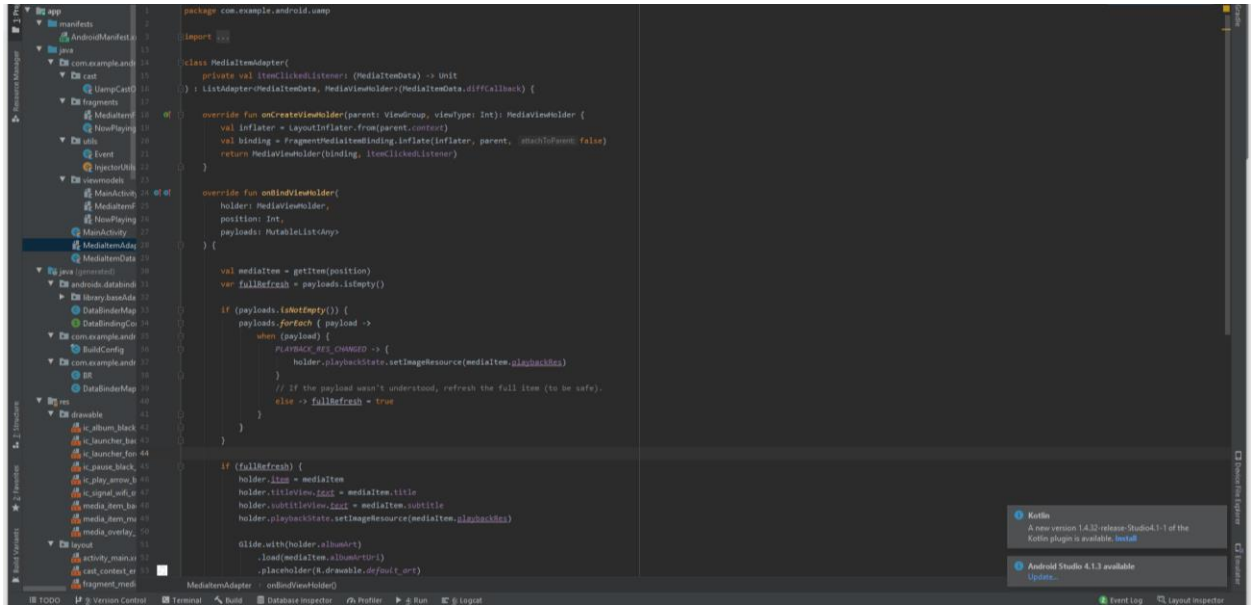


```
    private val mediaMetadataObserver = Observer<MediaMetadataCompat> { it: MediaMetadataCompat!
        updateState(playbackState, it)
    }

    private val musicServiceConnection = musicServiceConnection.also { it: MusicServiceConnection
        it.playbackState.observeForever(playbackStateObserver)
        it.nowPlaying.observeForever(mediaMetadataObserver)
        checkPlaybackPosition()
    }

    private fun checkPlaybackPosition(): Boolean = handler.postDelayed({
        val currPosition = playbackState.currentPlayBackPosition
        if (mediaPosition.value != currPosition)
            mediaPosition.postValue(currPosition)
        if (updatePosition)
            checkPlaybackPosition()
    }, POSITION_UPDATE_INTERVAL_MILLIS)

    override fun onCleared() {
        super.onCleared()

        musicServiceConnection.playbackState.removeObserver(playbackStateObserver)
        musicServiceConnection.nowPlaying.removeObserver(mediaMetadataObserver)

        updatePosition = false
    }

    private fun updateState(
        playbackState: PlaybackStateCompat,
        mediaMetadata: MediaMetadataCompat
    ) {

        if (mediaMetadata.duration != 0L && mediaMetadata.id != null) {
            val nowPlayingMetadata = NowPlayingMetadata(
                mediaMetadata.id!!,
                mediaMetadata.albumArtUri,
                mediaMetadata.title?.trim(),
                mediaMetadata.displaySubtitle?.trim(),
                NowPlayingMetadata.timestampToMSS(app, mediaMetadata.duration)
            )
            this.mediaMetadata.postValue(nowPlayingMetadata)
        }
```

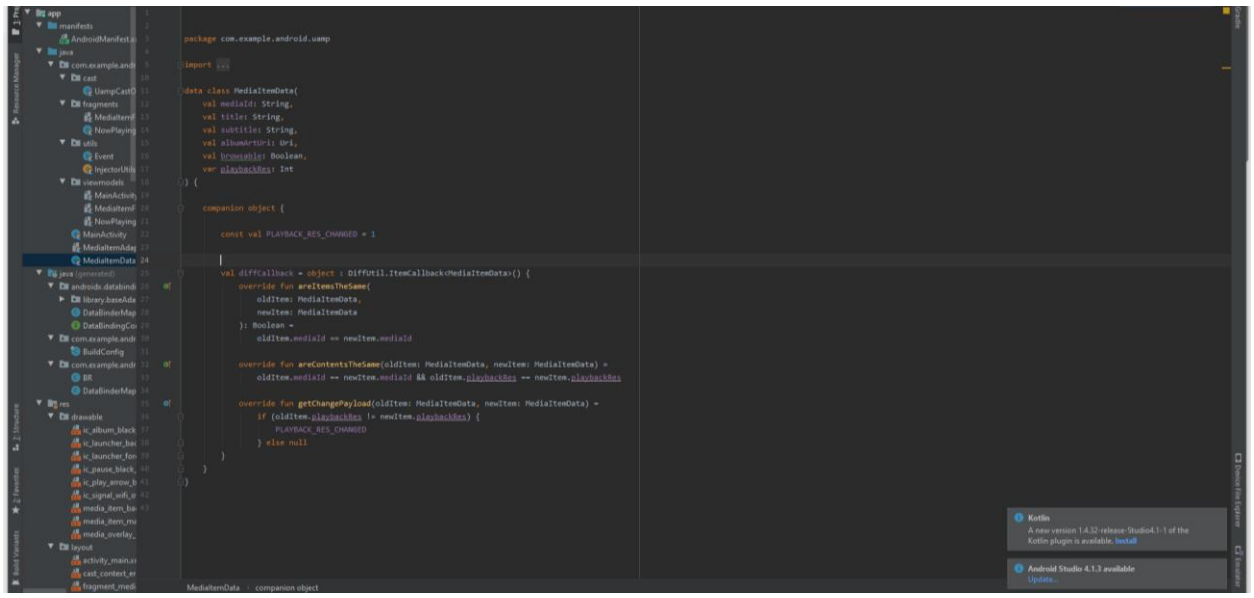




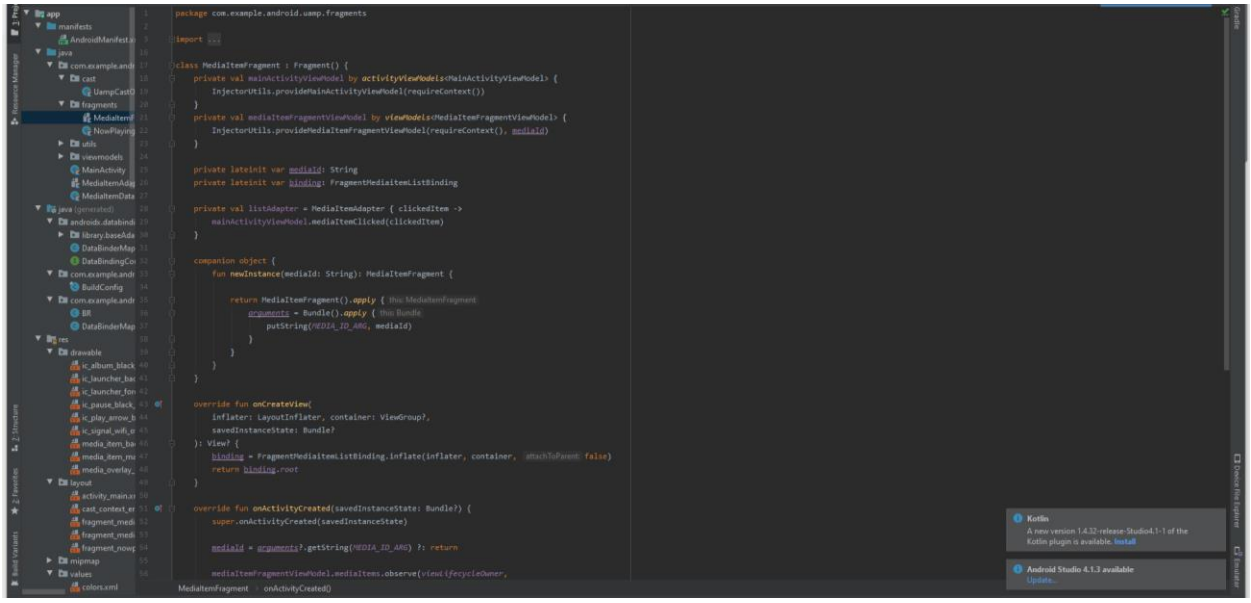




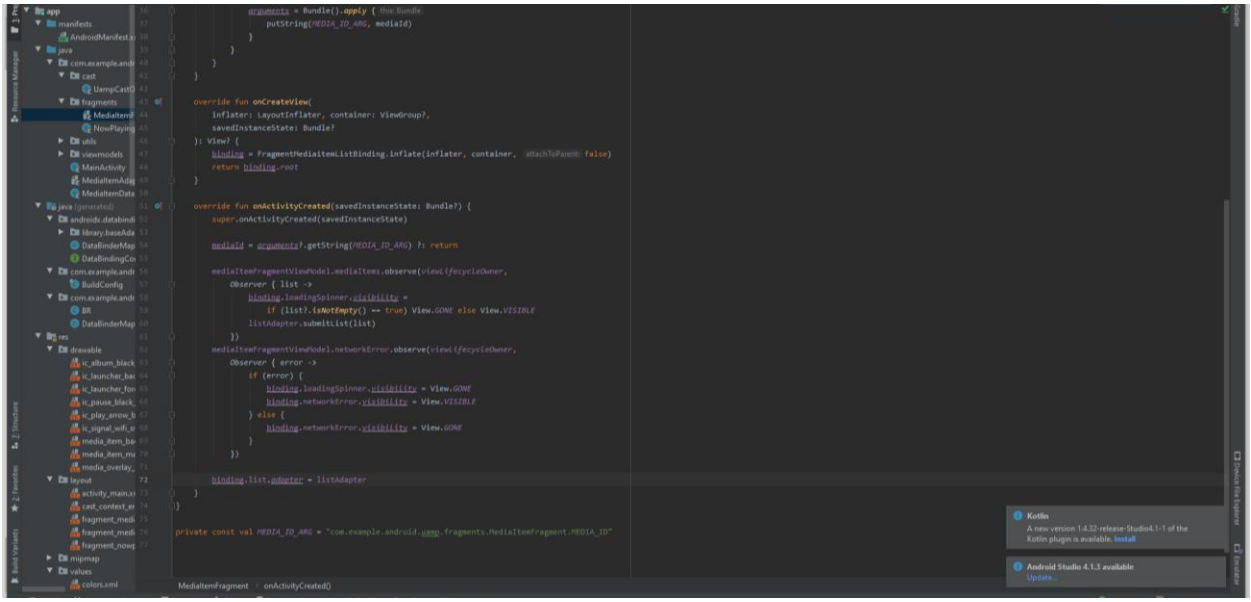
```
        }

        if (fullRefresh) {
            holder.item = mediaItem
            holder.titleView.text = mediaItem.title
            holder.subtitleView.text = mediaItem.subtitle
            holder.playbackState.setImageResource(mediaItem.playbackRes)

            Glide.with(holder.albumArt)
                .load(mediaItem.albumArtUri)
                .placeholder(R.drawable.default_art)
                .into(holder.albumArt)
        }
    }

    override fun onBindViewHolder(holder: MediaViewHolder, position: Int) {
        onBindViewHolder(holder, position, mutableListOf())
    }
}

class MediaViewHolder(
    binding: FragmentMediaItemBinding,
    itemClickedListener: (MediaItemData) -> Unit
) : RecyclerView.ViewHolder(binding.root) {

    val titleView: TextView = binding.title
    val subtitleView: TextView = binding.subtitle
    val albumArt: ImageView = binding.albumArt
    val playbackState: ImageView = binding.itemState

    var item: MediaItemData? = null

    init {
        binding.root.setOnClickListener { it.View!
            item?.let { itemClickedListener(it) }
        }
    }
}
```



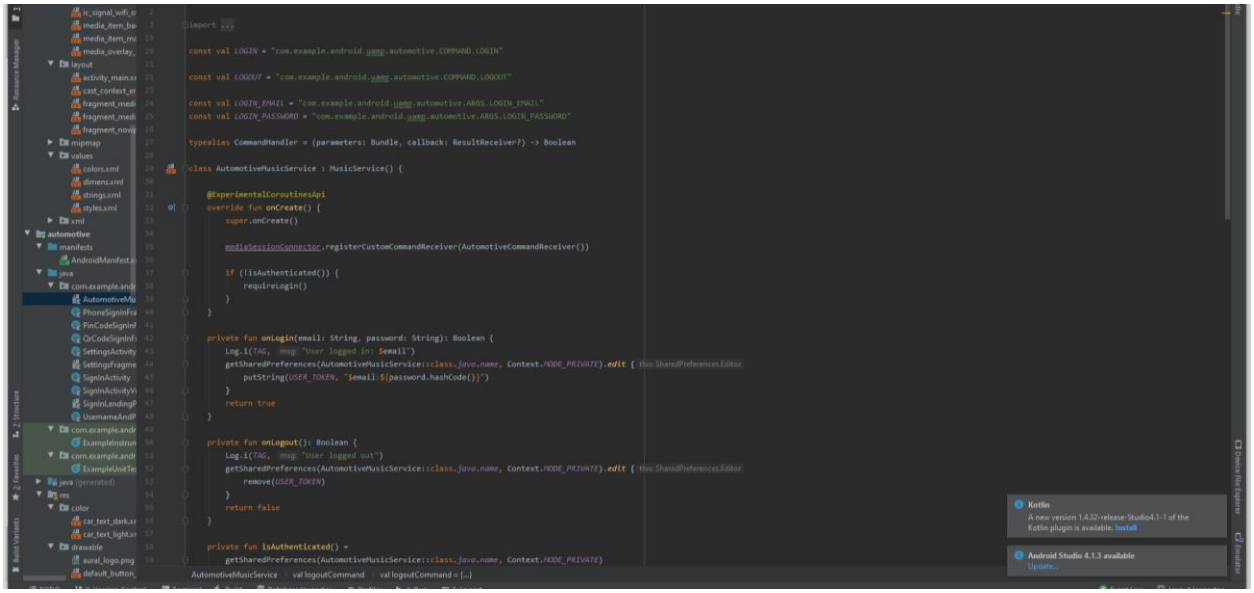
```
package com.example.android.uamp

import ...

data class MediaItemData(
    val mediaId: String,
    val title: String,
    val subtitle: String,
    val albumArtUri: Uri,
    val browsable: Boolean,
    var playbackRes: Int
) {

    companion object {

        const val PLAYBACK_RES_CHANGED = 1

        val diffCallback = object : DiffUtil.ItemCallback<MediaItemData>() {
            override fun areItemsTheSame(
                oldItem: MediaItemData,
                newItem: MediaItemData
            ): Boolean =
                oldItem.mediaId == newItem.mediaId

            override fun areContentsTheSame(oldItem: MediaItemData, newItem: MediaItemData) =
                oldItem.mediaId == newItem.mediaId && oldItem.playbackRes == newItem.playbackRes

            override fun getChangePayload(oldItem: MediaItemData, newItem: MediaItemData) =
                if (oldItem.playbackRes != newItem.playbackRes) {
                    PLAYBACK_RES_CHANGED
                } else null
        }
    }
}
```



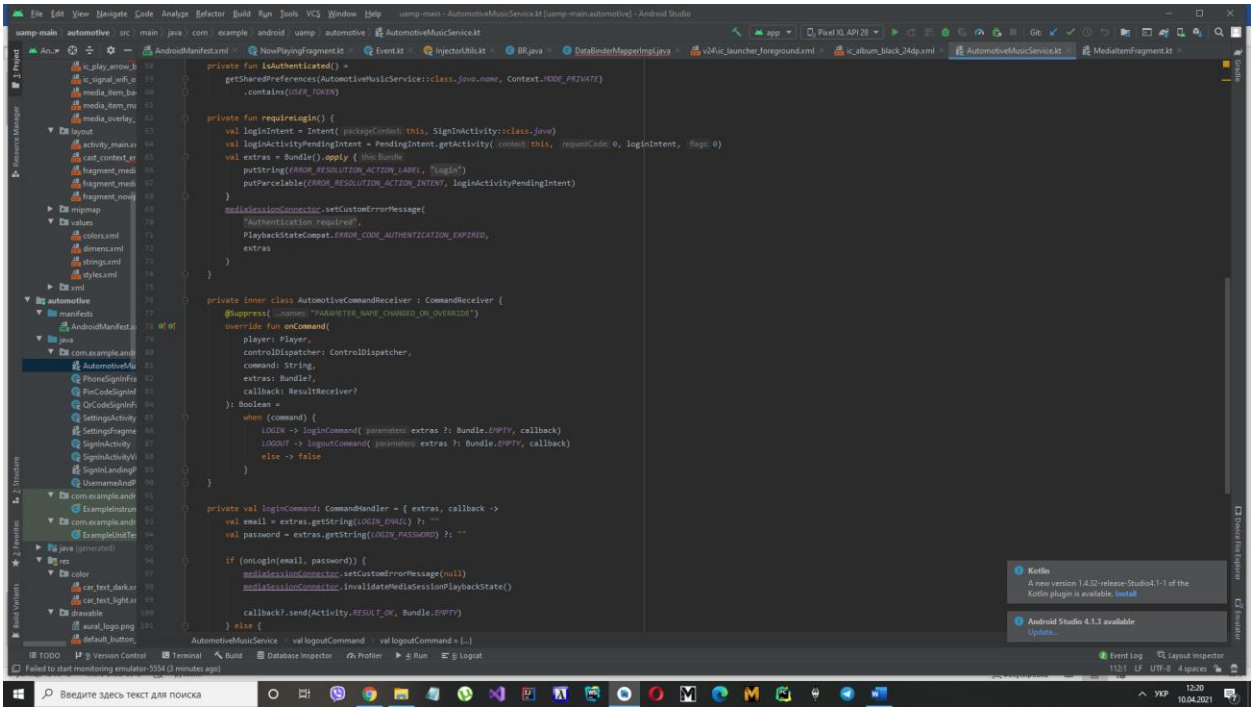
```
package com.example.android.uamp.fragments

import ...

class MediaItemFragment : Fragment() {
    private val mainActivityViewModel by activityViewModels<MainActivityViewModel> {
        InjectorUtils.provideMainActivityViewModel(requireContext())
    }
    private val mediaItemFragmentViewModel by viewModels<MediaItemFragmentViewModel> {
        InjectorUtils.provideMediaItemFragmentViewModel(requireContext(), mediaId)
    }

    private lateinit var mediaId: String
    private lateinit var binding: FragmentMediaitemListBinding

    private val listAdapter = MediaItemAdapter { clickedItem ->
        mainActivityViewModel.mediaItemClicked(clickedItem)
    }

    companion object {
        fun newInstance(mediaId: String): MediaItemFragment {

            return MediaItemFragment().apply { this:MediaItemFragment
                arguments = Bundle().apply { this:Bundle
                    putString(MEDIA_ID_ARG, mediaId)
                }
            }
        }
    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        binding = FragmentMediaitemListBinding.inflate(inflater, container, attachToParent: false)
        return binding.root
    }

    override fun onActivityCreated(savedInstanceState: Bundle?) {
        super.onActivityCreated(savedInstanceState)

        mediaId = arguments?.getString(MEDIA_ID_ARG) ?: return

        mediaItemFragmentViewModel.mediaItems.observe(viewLifecycleOwner,
```



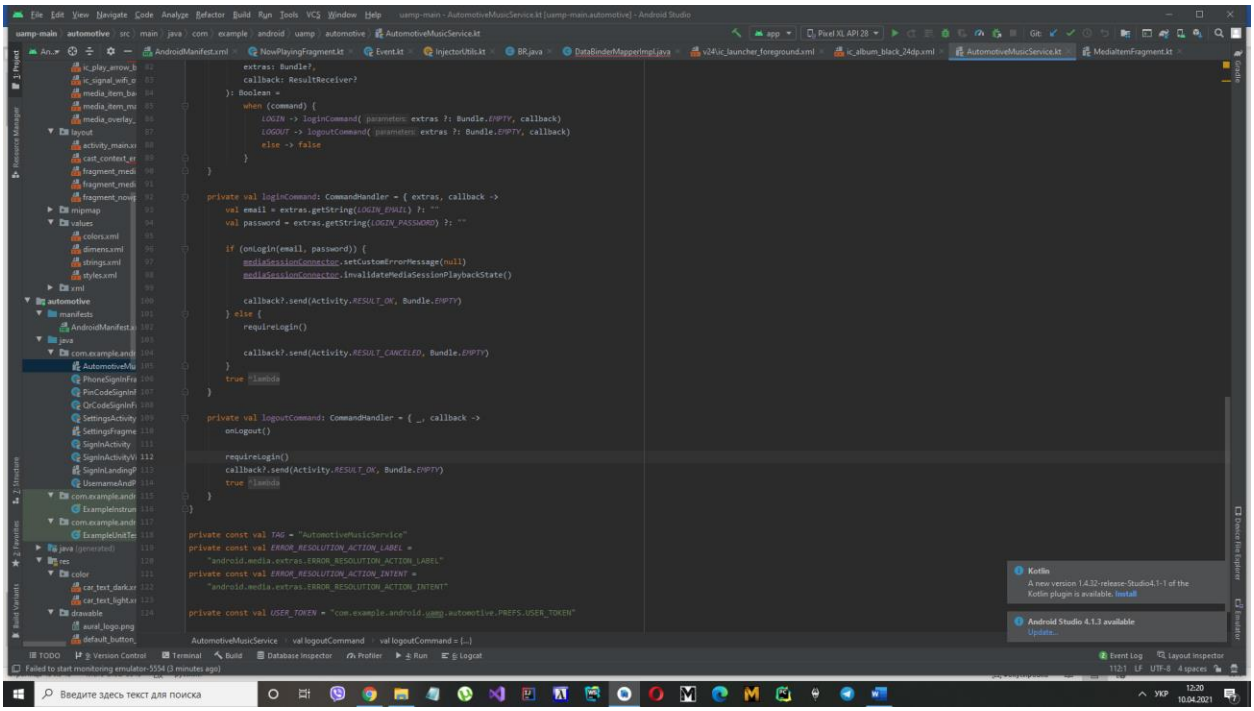
```
                arguments = Bundle().apply { this:Bundle
                    putString(MEDIA_ID_ARG, mediaId)
                }
            }
        }
    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        binding = FragmentMediaitemListBinding.inflate(inflater, container, attachToParent: false)
        return binding.root
    }

    override fun onActivityCreated(savedInstanceState: Bundle?) {
        super.onActivityCreated(savedInstanceState)

        mediaId = arguments?.getString(MEDIA_ID_ARG) ?: return

        mediaItemFragmentViewModel.mediaItems.observe(viewLifecycleOwner,
            Observer { list ->
                binding.loadingSpinner.visibility =
                    if (list?.isNotEmpty() == true) View.GONE else View.VISIBLE
                listAdapter.submitList(list)
            })
        mediaItemFragmentViewModel.networkError.observe(viewLifecycleOwner,
            Observer { error ->
                if (error) {
                    binding.loadingSpinner.visibility = View.GONE
                    binding.networkError.visibility = View.VISIBLE
                } else {
                    binding.networkError.visibility = View.GONE
                }
            })

        binding.list.adapter = listAdapter
    }
}

private const val MEDIA_ID_ARG = "com.example.android.uamp.fragments.MediaItemFragment.MEDIA_ID"
```

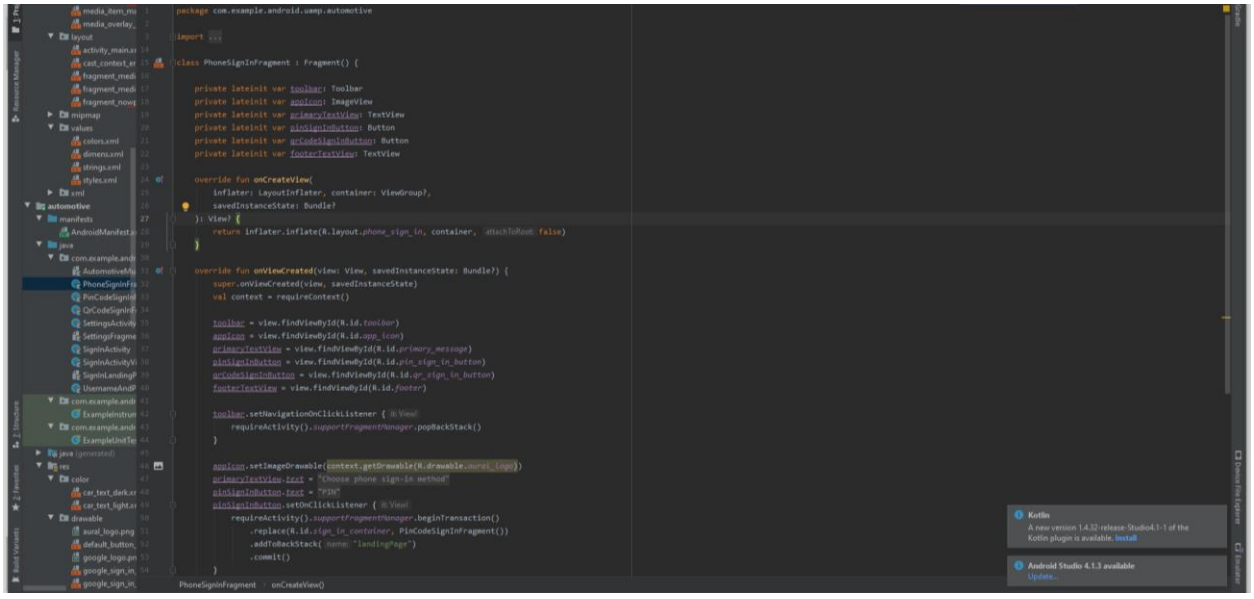


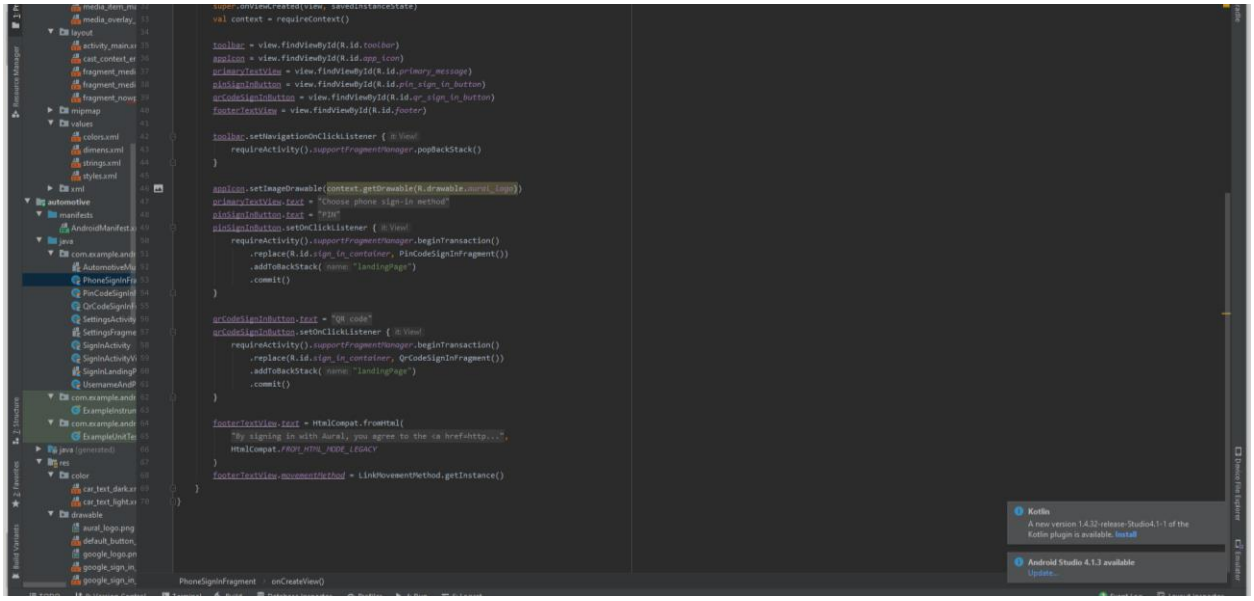
```
import ...

const val LOGIN = "com.example.android.uamp.automotive.COMMAND.LOGIN"

const val LOGOUT = "com.example.android.uamp.automotive.COMMAND.LOGOUT"

const val LOGIN_EMAIL = "com.example.android.uamp.automotive.ARGS.LOGIN_EMAIL"
const val LOGIN_PASSWORD = "com.example.android.uamp.automotive.ARGS.LOGIN_PASSWORD"

typealias CommandHandler = (parameters: Bundle, callback: ResultReceiver?) -> Boolean

class AutomotiveMusicService : MusicService() {

    @ExperimentalCoroutinesApi
    override fun onCreate() {
        super.onCreate()

        mediaSessionConnector.registerCustomCommandReceiver(AutomotiveCommandReceiver())

        if (!isAuthenticated()) {
            requireLogin()
        }
    }

    private fun onLogin(email: String, password: String): Boolean {
        Log.i(TAG, msg: "User logged in: $email")
        getSharedPreferences(AutomotiveMusicService::class.java.name, Context.MODE_PRIVATE).edit { this: SharedPreferences.Editor
            putString(USER_TOKEN, "$email:${password.hashCode()}")
        }
        return true
    }

    private fun onLogout(): Boolean {
        Log.i(TAG, msg: "User logged out")
        getSharedPreferences(AutomotiveMusicService::class.java.name, Context.MODE_PRIVATE).edit { this: SharedPreferences.Editor
            remove(USER_TOKEN)
        }
        return false
    }

    private fun isAuthenticated() =
        getSharedPreferences(AutomotiveMusicService::class.java.name, Context.MODE_PRIVATE)
```



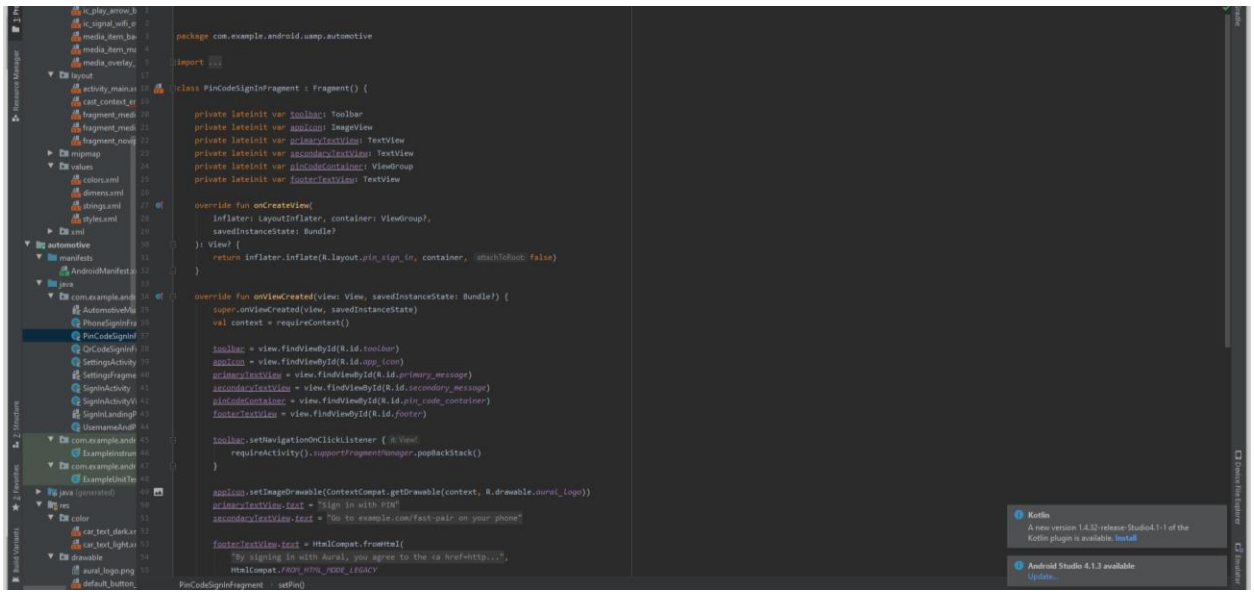
```
    private fun isAuthenticated() =
        getSharedPreferences(AutomotiveMusicService::class.java.name, Context.MODE_PRIVATE)
            .contains(USER_TOKEN)

    private fun requireLogin() {
        val loginIntent = Intent( packageContext: this, SignInActivity::class.java)
        val loginActivityPendingIntent = PendingIntent.getActivity( context: this,  requestCode: 0, loginIntent,  flags: 0)
        val extras = Bundle().apply { this: Bundle
            putString(ERROR_RESOLUTION_ACTION_LABEL, "Login")
            putParcelable(ERROR_RESOLUTION_ACTION_INTENT, loginActivityPendingIntent)
        }
        mediaSessionConnector.setCustomErrorMessage(
            "Authentication required",
            PlaybackStateCompat.ERROR_CODE_AUTHENTICATION_EXPIRED,
            extras
        )
    }

    private inner class AutomotiveCommandReceiver : CommandReceiver {
        @Suppress( ...names: "PARAMETER_NAME_CHANGED_ON_OVERRIDE")
        override fun onCommand(
            player: Player,
            controlDispatcher: ControlDispatcher,
            command: String,
            extras: Bundle?,
            callback: ResultReceiver?
        ): Boolean =
            when (command) {
                LOGIN -> loginCommand( parameters: extras ?: Bundle.EMPTY, callback)
                LOGOUT -> logoutCommand( parameters: extras ?: Bundle.EMPTY, callback)
                else -> false
            }
    }

    private val loginCommand: CommandHandler = { extras, callback ->
        val email = extras.getString(LOGIN_EMAIL) ?: ""
        val password = extras.getString(LOGIN_PASSWORD) ?: ""

        if (onLogin(email, password)) {
            mediaSessionConnector.setCustomErrorMessage(null)
            mediaSessionConnector.invalidateMediaSessionPlaybackState()

            callback?.send(Activity.RESULT_OK, Bundle.EMPTY)
        } else {
```



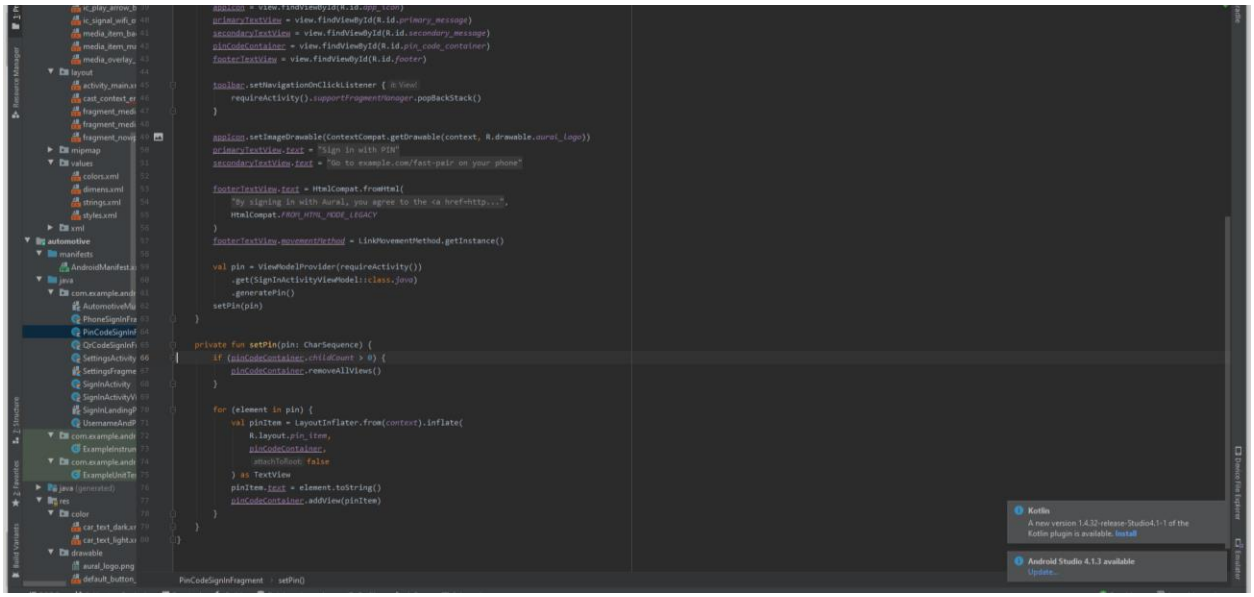


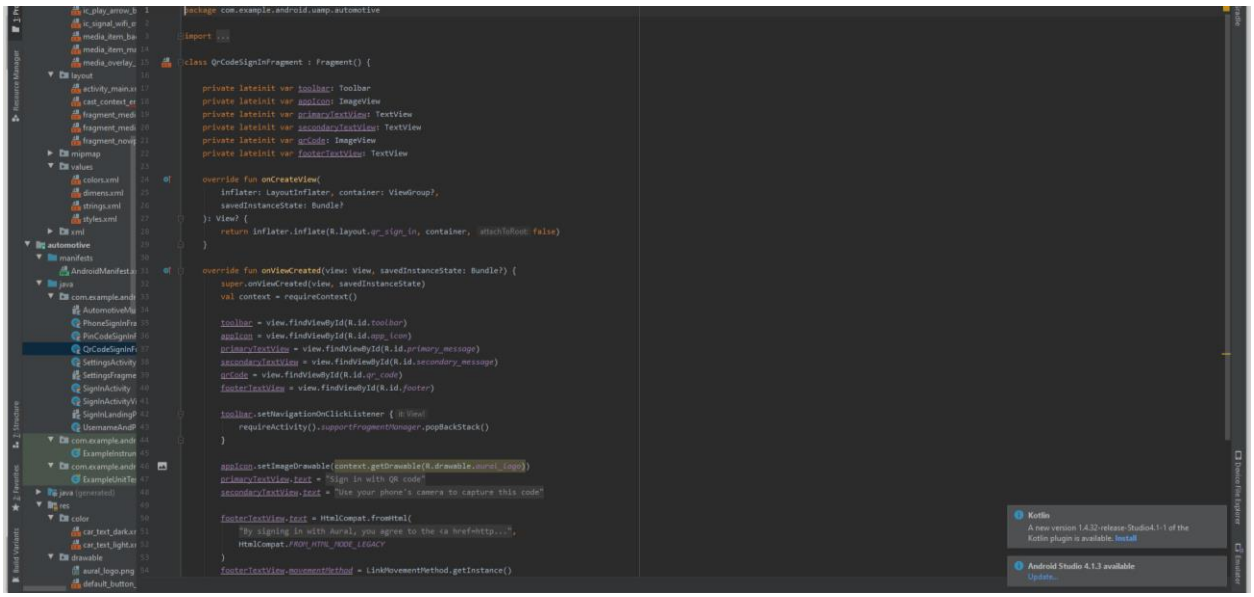


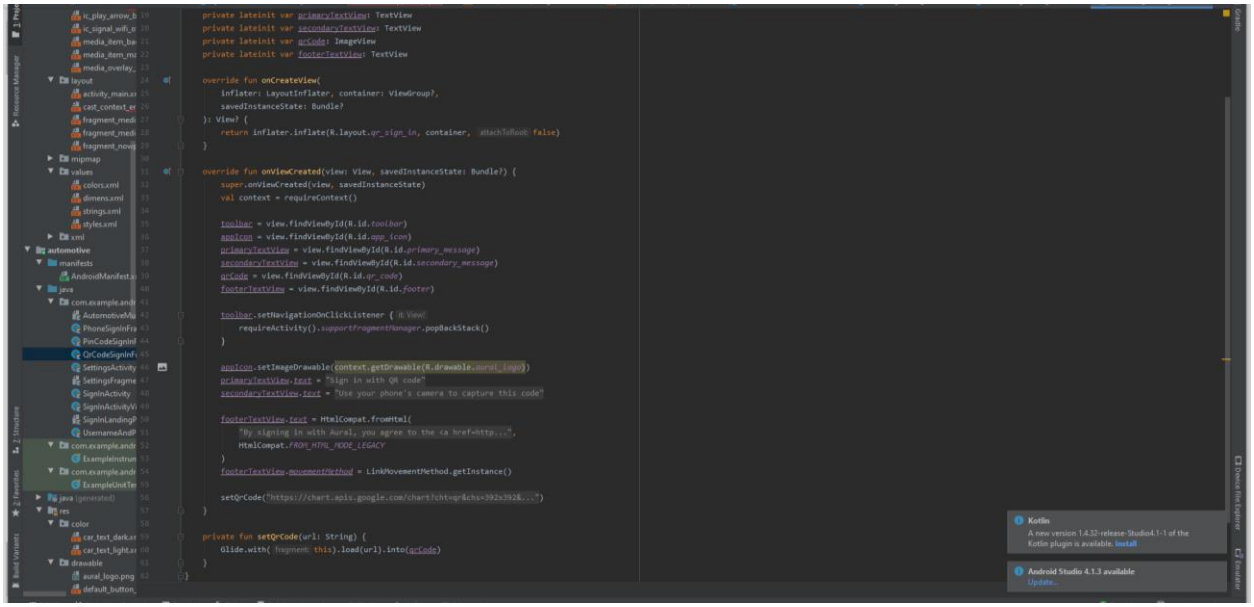
```
        super.onViewCreated(view, savedInstanceState)
        val context = requireContext()

        toolbar = view.findViewById(R.id.toolbar)
        appIcon = view.findViewById(R.id.app_icon)
        primaryTextView = view.findViewById(R.id.primary_message)
        pinSignInButton = view.findViewById(R.id.pin_sign_in_button)
        qrCodeSignInButton = view.findViewById(R.id.qr_sign_in_button)
        footerTextView = view.findViewById(R.id.footer)

        toolbar.setNavigationOnClickListener { it: View!
            requireActivity().supportFragmentManager.popBackStack()
        }

        appIcon.setImageDrawable(context.getDrawable(R.drawable.aural_logo))
        primaryTextView.text = "Choose phone sign-in method"
        pinSignInButton.text = "PIN"
        pinSignInButton.setOnClickListener { it: View!
            requireActivity().supportFragmentManager.beginTransaction()
                .replace(R.id.sign_in_container, PinCodeSignInFragment())
                .addToBackStack( name: "landingPage")
                .commit()
        }

        qrCodeSignInButton.text = "QR code"
        qrCodeSignInButton.setOnClickListener { it: View!
            requireActivity().supportFragmentManager.beginTransaction()
                .replace(R.id.sign_in_container, QrCodeSignInFragment())
                .addToBackStack( name: "landingPage")
                .commit()
        }

        footerTextView.text = HtmlCompat.fromHtml(
            "By signing in with Aural, you agree to the <a href=http...",
            HtmlCompat.FROM_HTML_MODE_LEGACY
        )
        footerTextView.movementMethod = LinkMovementMethod.getInstance()
    }
}
```



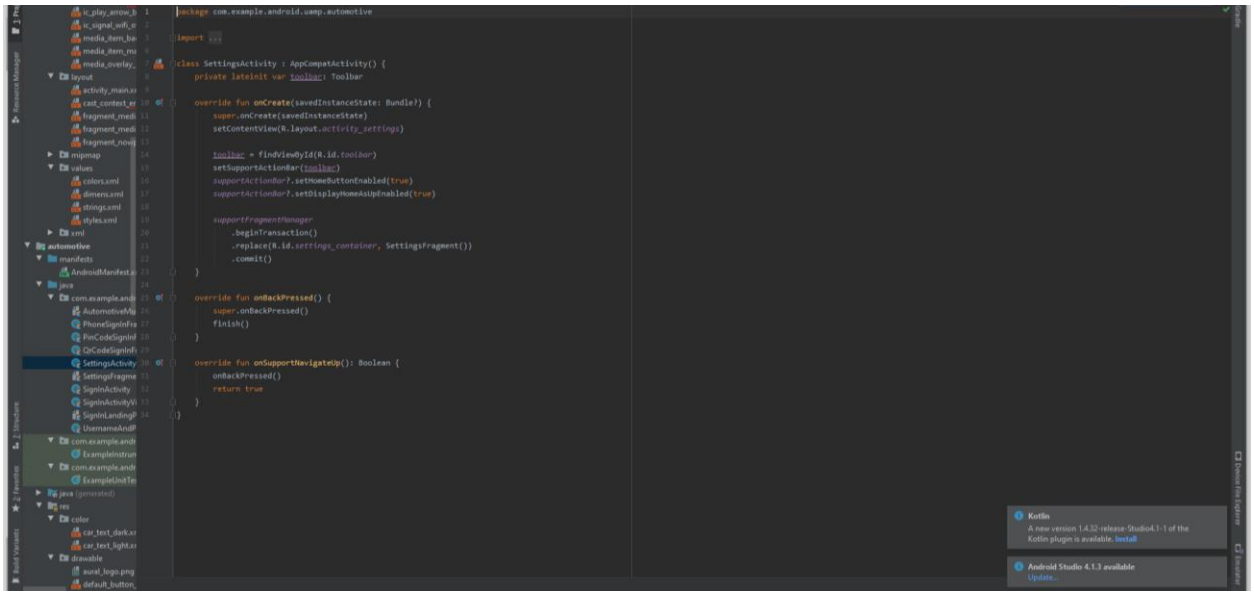
```
package com.example.android.uamp.automotive

import ...

class PinCodeSignInFragment : Fragment() {

    private lateinit var toolbar: Toolbar
    private lateinit var appIcon: ImageView
    private lateinit var primaryTextView: TextView
    private lateinit var secondaryTextView: TextView
    private lateinit var pinCodeContainer: ViewGroup
    private lateinit var footerTextView: TextView

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        return inflater.inflate(R.layout.pin_sign_in, container, attachToRoot: false)
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        val context = requireContext()

        toolbar = view.findViewById(R.id.toolbar)
        appIcon = view.findViewById(R.id.app_icon)
        primaryTextView = view.findViewById(R.id.primary_message)
        secondaryTextView = view.findViewById(R.id.secondary_message)
        pinCodeContainer = view.findViewById(R.id.pin_code_container)
        footerTextView = view.findViewById(R.id.footer)

        toolbar.setNavigationOnClickListener { it: View!
            requireActivity().supportFragmentManager.popBackStack()
        }

        appIcon.setImageDrawable(ContextCompat.getDrawable(context, R.drawable.aural_logo))
        primaryTextView.text = "Sign in with PIN"
        secondaryTextView.text = "Go to example.com/fast-pair on your phone"

        footerTextView.text = HtmlCompat.fromHtml(
            "By signing in with Aural, you agree to the <a href=http...",
            HtmlCompat.FROM_HTML_MODE_LEGACY
```



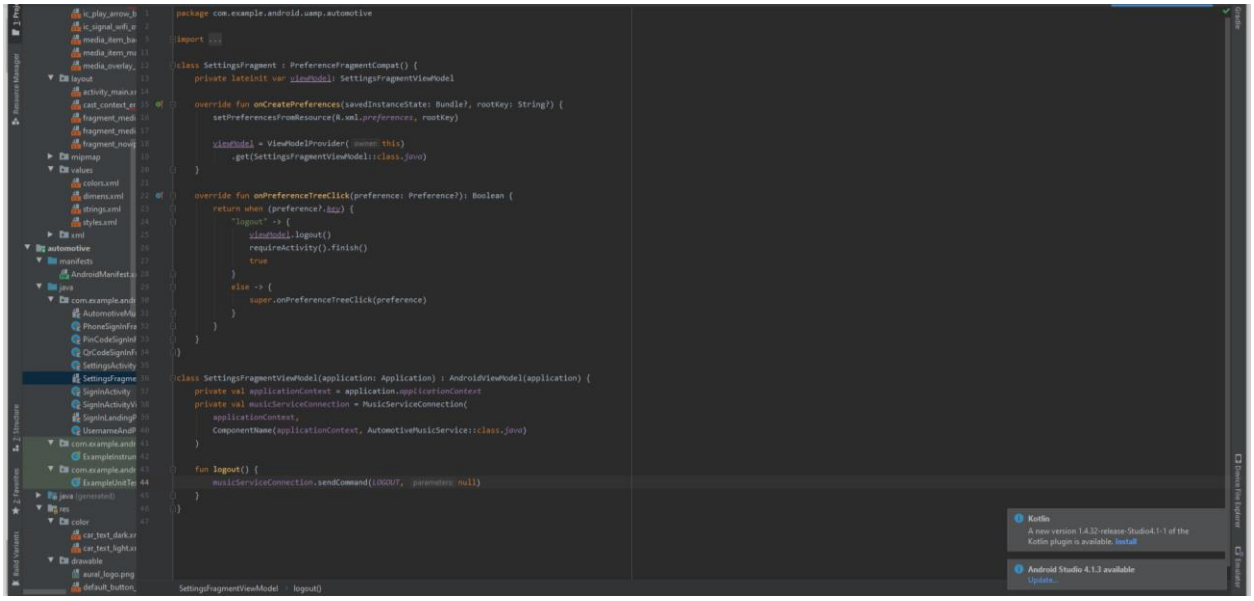




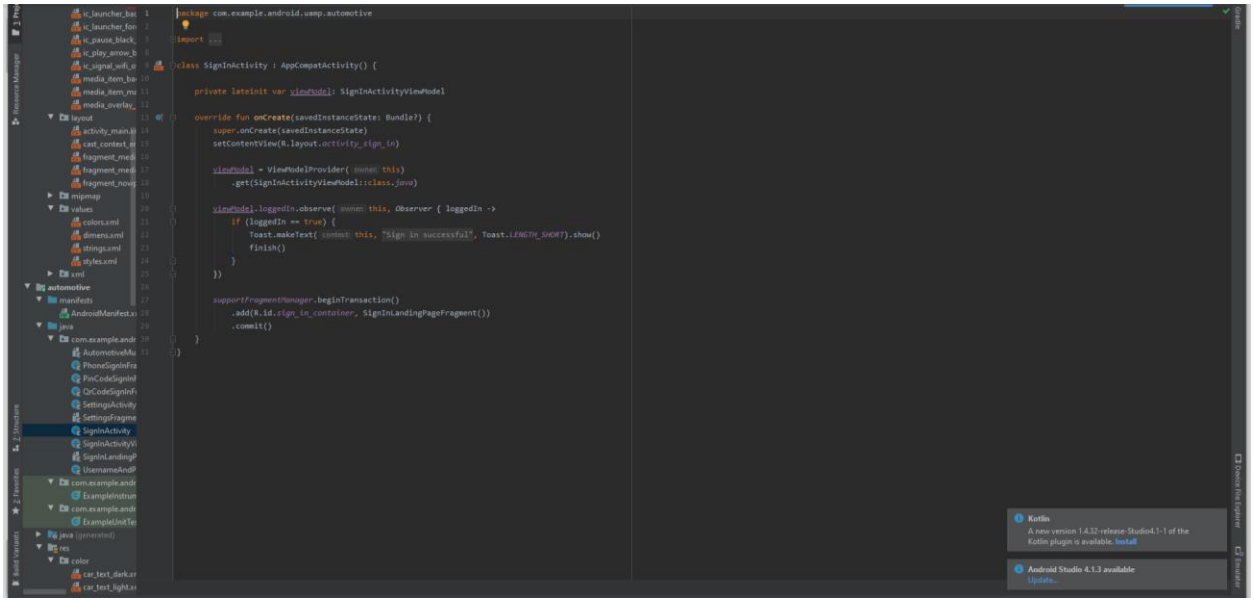
```
    private lateinit var primaryTextView: TextView
    private lateinit var secondaryTextView: TextView
    private lateinit var qrCode: ImageView
    private lateinit var footerTextView: TextView

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        return inflater.inflate(R.layout.qr_sign_in, container,  attachToRoot: false)
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        val context = requireContext()

        toolbar = view.findViewById(R.id.toolbar)
        appIcon = view.findViewById(R.id.app_icon)
        primaryTextView = view.findViewById(R.id.primary_message)
        secondaryTextView = view.findViewById(R.id.secondary_message)
        qrCode = view.findViewById(R.id.qr_code)
        footerTextView = view.findViewById(R.id.footer)

        toolbar.setNavigationOnClickListener { it: View!
            requireActivity().supportFragmentManager.popBackStack()
        }

        appIcon.setImageDrawable(context.getDrawable(R.drawable.aural_logo))
        primaryTextView.text = "Sign in with QR code"
        secondaryTextView.text = "Use your phone's camera to capture this code"

        footerTextView.text = HtmlCompat.fromHtml(
            "By signing in with Aural, you agree to the ca href=http...",
            HtmlCompat.FROM_HTML_MODE_LEGACY
        )
        footerTextView.movementMethod = LinkMovementMethod.getInstance()

        setQrCode("https://chart.apis.google.com/chart?cht=qr&chs=392x392&...")
    }

    private fun setQrCode(url: String) {
        Glide.with( fragment: this).load(url).into(qrCode)
    }
}
```



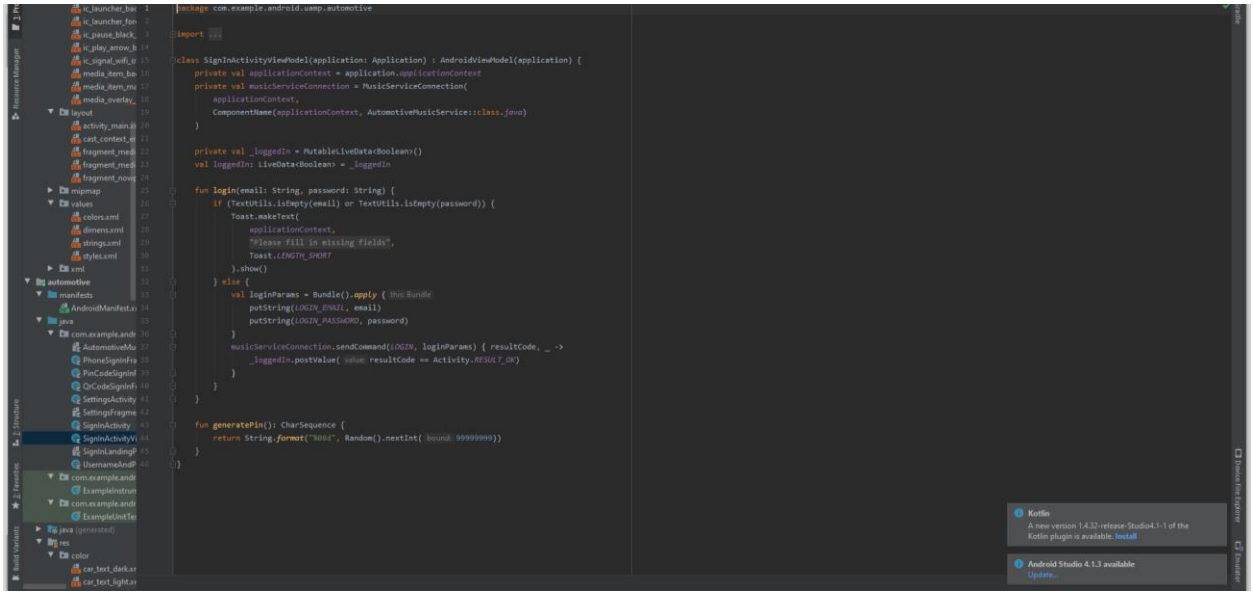
```
package com.example.android.uamp.automotive

import ...

class SettingsActivity : AppCompatActivity() {
    private lateinit var toolbar: Toolbar

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_settings)

        toolbar = findViewById(R.id.toolbar)
        setSupportActionBar(toolbar)
        supportActionBar?.setHomeButtonEnabled(true)
        supportActionBar?.setDisplayHomeAsUpEnabled(true)

        supportFragmentManager
            .beginTransaction()
            .replace(R.id.settings_container, SettingsFragment())
            .commit()
    }

    override fun onBackPressed() {
        super.onBackPressed()
        finish()
    }

    override fun onSupportNavigateUp(): Boolean {
        onBackPressed()
        return true
    }
}
```

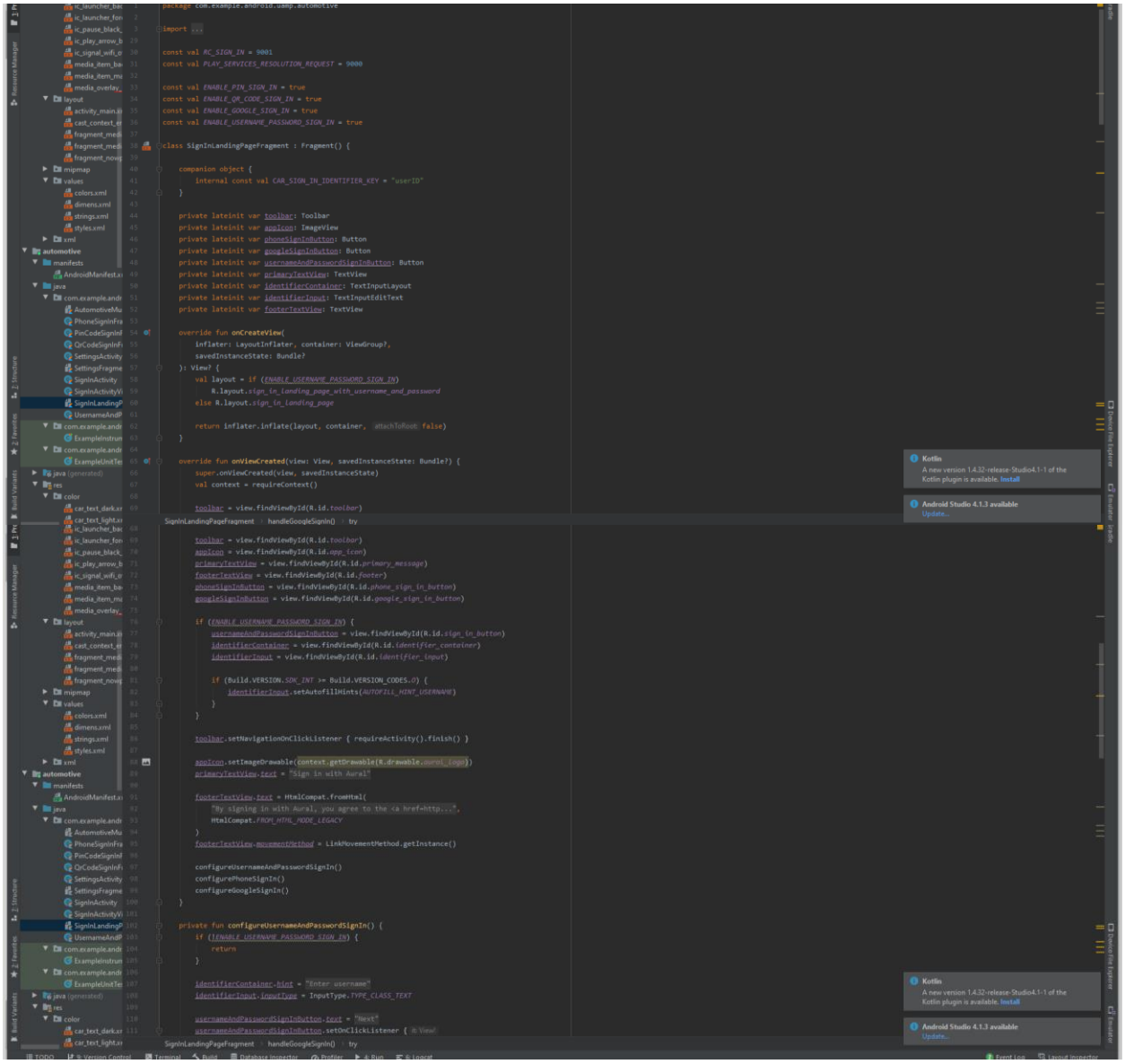


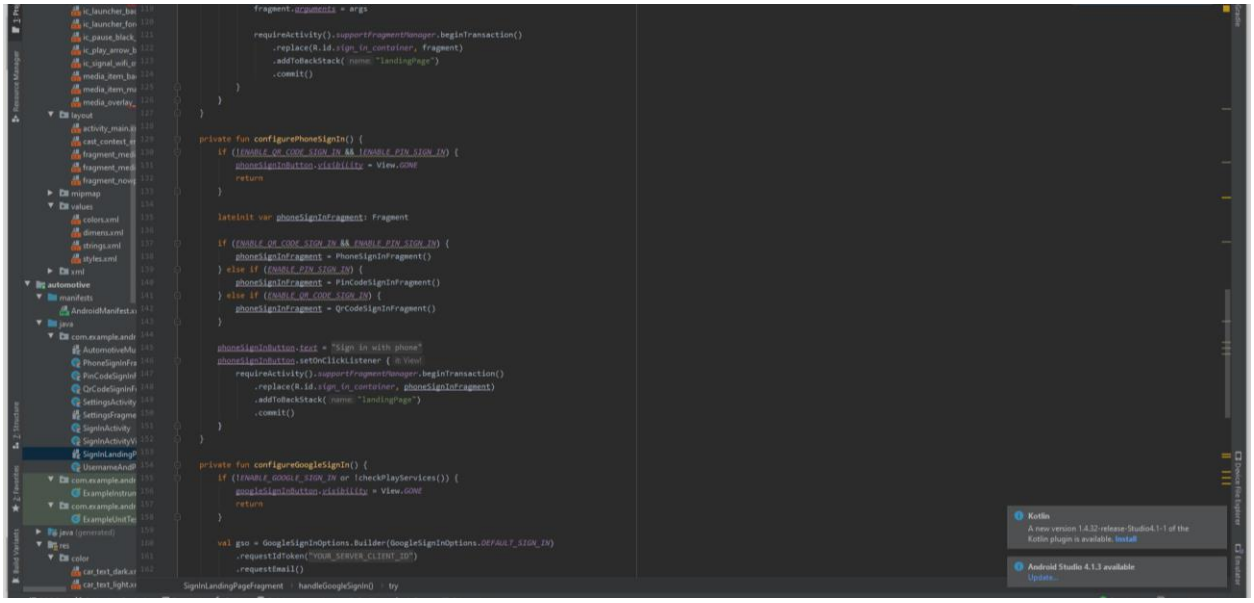
```
package com.example.android.uamp.automotive

import ...

class SettingsFragment : PreferenceFragmentCompat() {
    private lateinit var viewModel: SettingsFragmentViewModel

    override fun onCreatePreferences(savedInstanceState: Bundle?, rootKey: String?) {
        setPreferencesFromResource(R.xml.preferences, rootKey)

        viewModel = ViewModelProvider( owner: this)
            .get(SettingsFragmentViewModel::class.java)
    }

    override fun onPreferenceTreeClick(preference: Preference?): Boolean {
        return when (preference?.key) {
            "logout" -> {
                viewModel.logout()
                requireActivity().finish()
                true
            }
            else -> {
                super.onPreferenceTreeClick(preference)
            }
        }
    }
}

class SettingsFragmentViewModel(application: Application) : AndroidViewModel(application) {
    private val applicationContext = application.applicationContext
    private val musicServiceConnection = MusicServiceConnection(
        applicationContext,
        ComponentName(applicationContext, AutomotiveMusicService::class.java)
    )

    fun logout() {
        musicServiceConnection.sendCommand(LOGOUT,  parameters: null)
    }
}
```



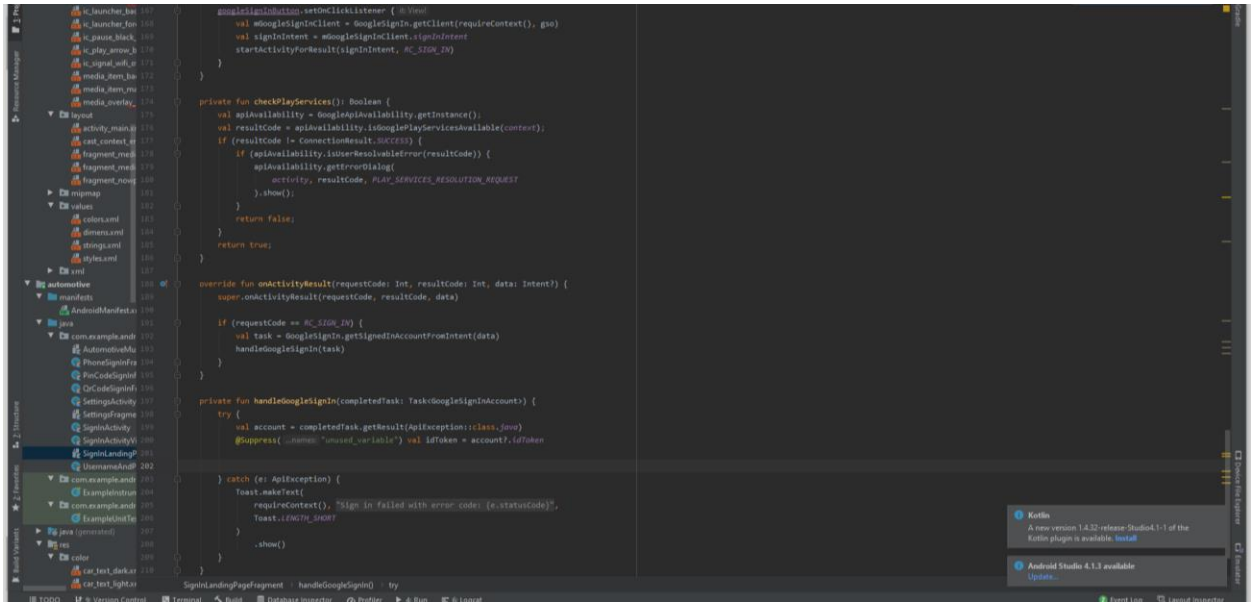
```
package com.example.android.uamp.automotive

import ...

class SignInActivity : AppCompatActivity() {

    private lateinit var viewModel: SignInActivityViewModel

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_sign_in)

        viewModel = ViewModelProvider( owner: this)
            .get(SignInActivityViewModel::class.java)

        viewModel.loggedIn.observe( owner: this, Observer { loggedIn ->
            if (loggedIn == true) {
                Toast.makeText( context: this, "Sign in successful", Toast.LENGTH_SHORT).show()
                finish()
            }
        })

        supportFragmentManager.beginTransaction()
            .add(R.id.sign_in_container, SignInLandingPageFragment())
            .commit()
    }
}
```


package com.example.android.uamp.automotive
import ...
class SignInActivityViewModel(application: Application) : AndroidViewModel(application) {
private val applicationContext = application.applicationContext
private val musicServiceConnection = MusicServiceConnection(
applicationContext,
ComponentName(applicationContext, AutomotiveMusicService::class.java)
)
private val _loggedIn = MutableLiveData<Boolean>()
val loggedIn: LiveData<Boolean> = _loggedIn
fun login(email: String, password: String) {
if (TextUtils.isEmpty(email) or TextUtils.isEmpty(password)) {
Toast.makeText(
applicationContext,
"Please fill in missing fields",
Toast.LENGTH_SHORT
).show()
} else {
val loginParams = Bundle().apply { this:Bundle
putString(LOGIN_EMAIL, email)
putString(LOGIN_PASSWORD, password)
}
musicServiceConnection.sendCommand(LOGIN, loginParams) { resultCode, _ ->
_loggedIn.postValue( value: resultCode == Activity.RESULT_OK)
}
}
}
fun generatePin(): CharSequence {
return String.format("%04d", Random().nextInt( bound: 99999999))
}
}
Kotlin
A new version 1.4.32-release-Studio4.1-1 of the Kotlin plugin is available. Install
Android Studio 4.1.3 available
Update...

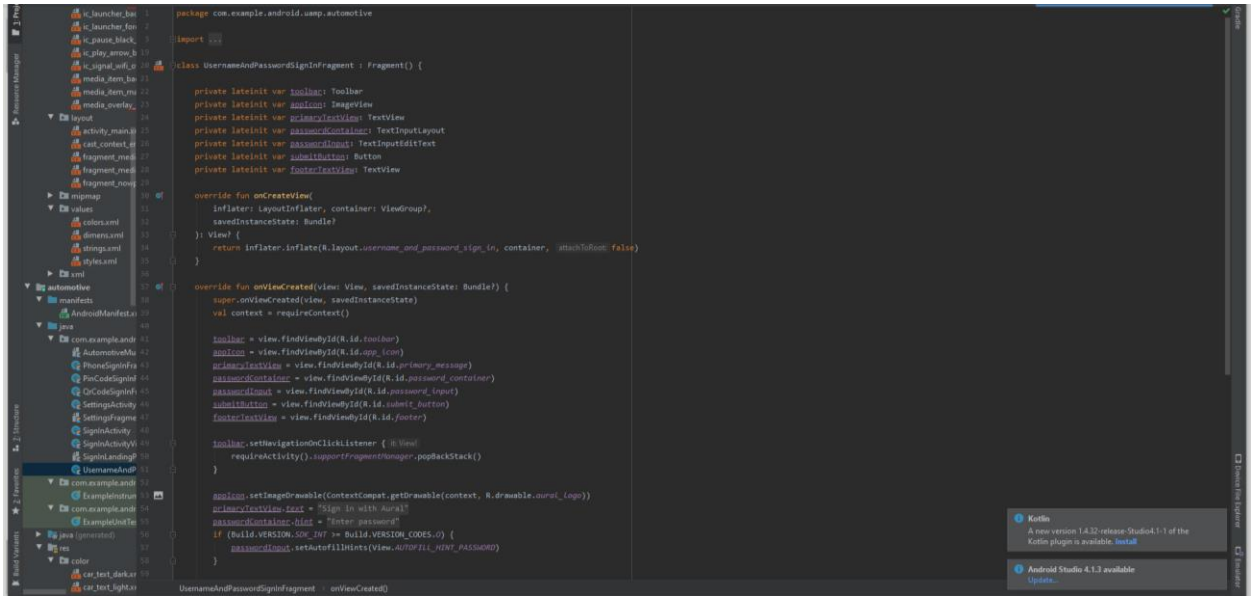




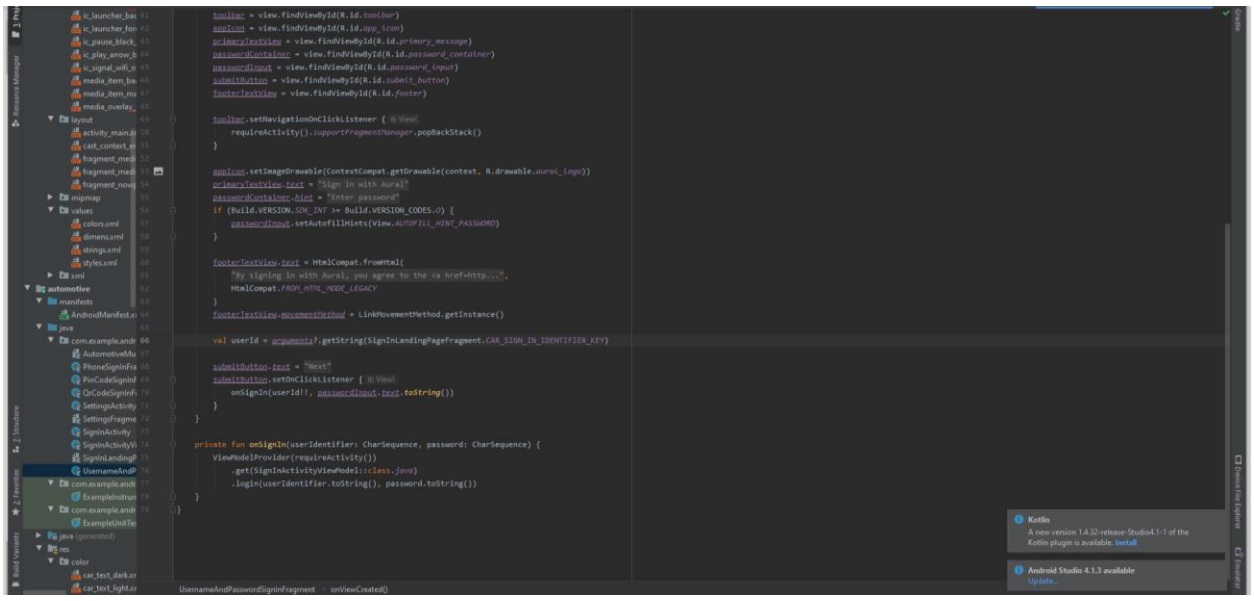
```
                fragment.arguments = args

                requireActivity().supportFragmentManager.beginTransaction()
                    .replace(R.id.sign_in_container, fragment)
                    .addToBackStack( name: "landingPage")
                    .commit()
            }
        }
    }

    private fun configurePhoneSignIn() {
        if (!ENABLE_QR_CODE_SIGN_IN && !ENABLE_PIN_SIGN_IN) {
            phoneSignInButton.visibility = View.GONE
            return
        }

        lateinit var phoneSignInFragment: Fragment

        if (ENABLE_QR_CODE_SIGN_IN && ENABLE_PIN_SIGN_IN) {
            phoneSignInFragment = PhoneSignInFragment()
        } else if (ENABLE_PIN_SIGN_IN) {
            phoneSignInFragment = PinCodeSignInFragment()
        } else if (ENABLE_QR_CODE_SIGN_IN) {
            phoneSignInFragment = QrCodeSignInFragment()
        }

        phoneSignInButton.text = "Sign in with phone"
        phoneSignInButton.setOnClickListener { it: View!
            requireActivity().supportFragmentManager.beginTransaction()
                .replace(R.id.sign_in_container, phoneSignInFragment)
                .addToBackStack( name: "landingPage")
                .commit()
        }
    }

    private fun configureGoogleSignIn() {
        if (!ENABLE_GOOGLE_SIGN_IN or !checkPlayServices()) {
            googleSignInButton.visibility = View.GONE
            return
        }

        val gso = GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
            .requestIdToken("YOUR_SERVER_CLIENT_ID")
            .requestEmail()

SignInLandingPageFragment  ›  handleGoogleSignIn()  ›  try
```



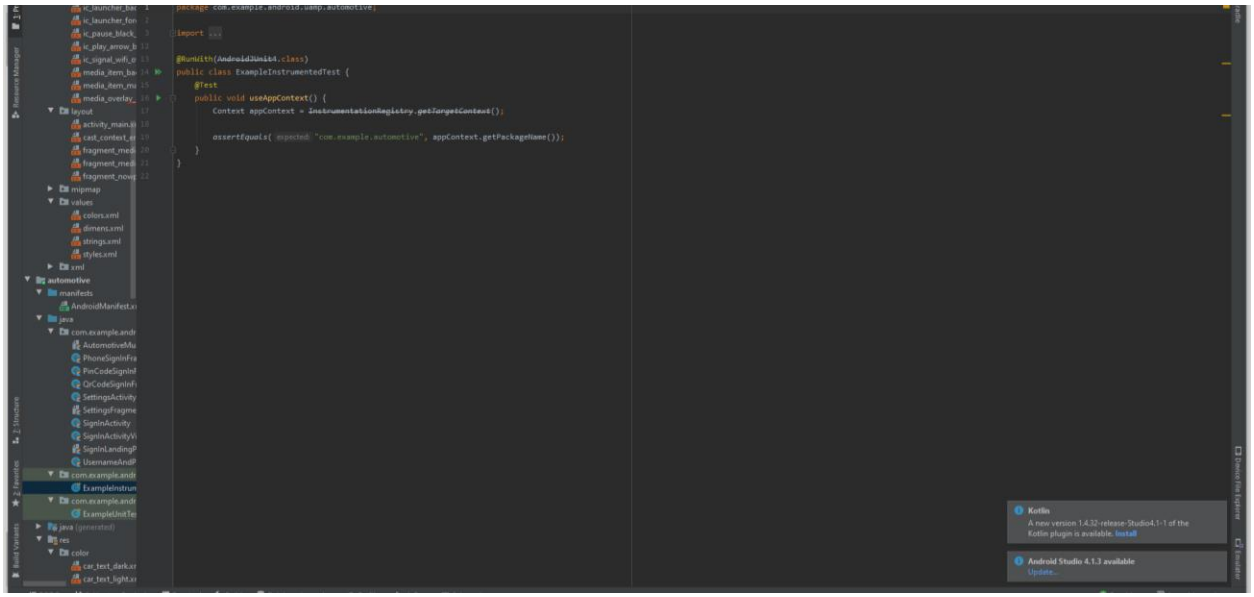
```
        googleSignInButton.setOnClickListener { it: View!
            val mGoogleSignInClient = GoogleSignIn.getClient(requireContext(), gso)
            val signInIntent = mGoogleSignInClient.signInIntent
            startActivityForResult(signInIntent, RC_SIGN_IN)
        }
    }

    private fun checkPlayServices(): Boolean {
        val apiAvailability = GoogleApiAvailability.getInstance();
        val resultCode = apiAvailability.isGooglePlayServicesAvailable(context);
        if (resultCode != ConnectionResult.SUCCESS) {
            if (apiAvailability.isUserResolvableError(resultCode)) {
                apiAvailability.getErrorDialog(
                    activity, resultCode, PLAY_SERVICES_RESOLUTION_REQUEST
                ).show();
            }
            return false;
        }
        return true;
    }

    override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
        super.onActivityResult(requestCode, resultCode, data)

        if (requestCode == RC_SIGN_IN) {
            val task = GoogleSignIn.getSignedInAccountFromIntent(data)
            handleGoogleSignIn(task)
        }
    }

    private fun handleGoogleSignIn(completedTask: Task<GoogleSignInAccount>) {
        try {
            val account = completedTask.getResult(ApiException::class.java)
            @Suppress( ...names: "unused_variable") val idToken = account?.idToken


        } catch (e: ApiException) {
            Toast.makeText(
                requireContext(), "Sign in failed with error code: ${e.statusCode}",
                Toast.LENGTH_SHORT
            )
                .show()
        }
    }

SignInLandingPageFragment  ›  handleGoogleSignIn()  ›  try
```

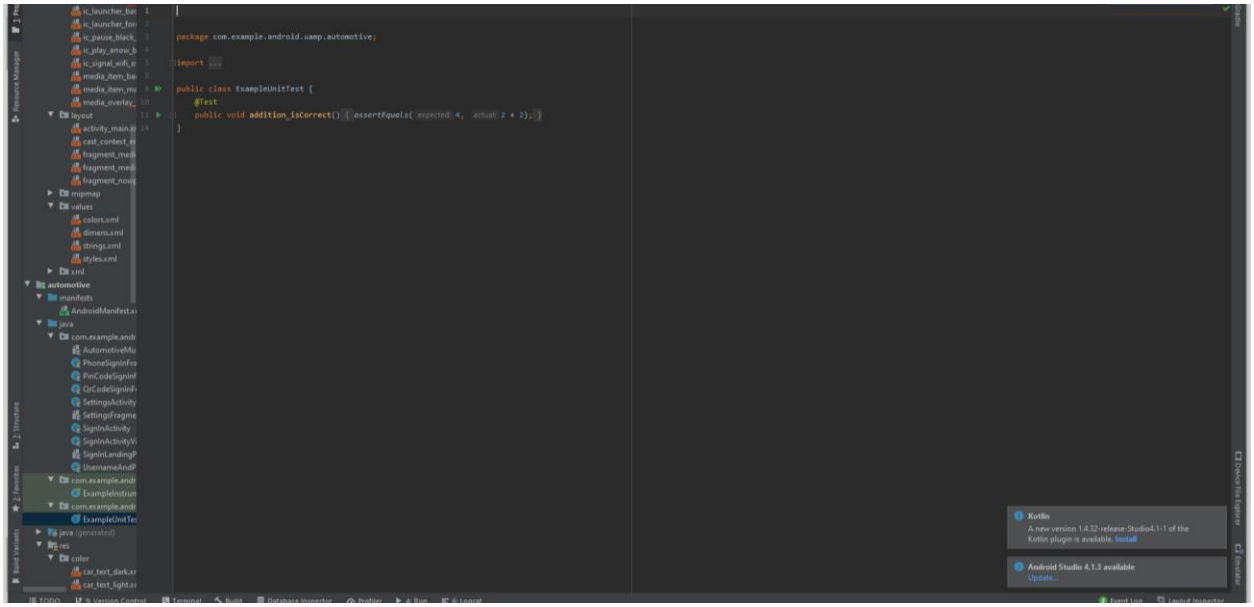


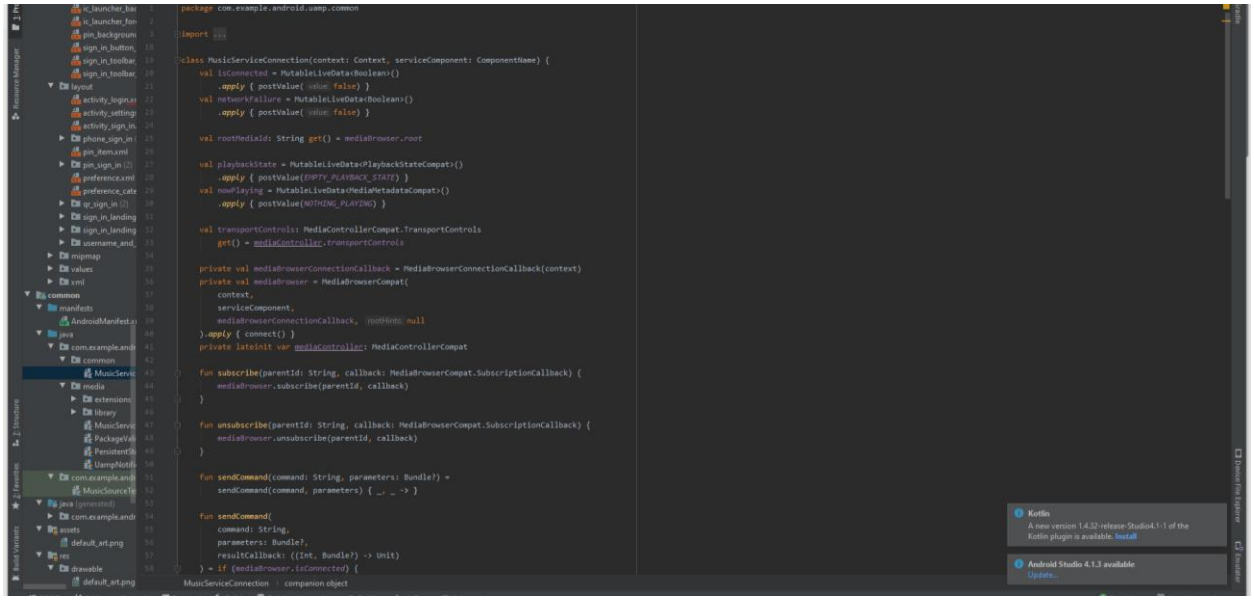
```
package com.example.android.uamp.automotive

import ...

class UsernameAndPasswordSignInFragment : Fragment() {

    private lateinit var toolbar: Toolbar
    private lateinit var appIcon: ImageView
    private lateinit var primaryTextView: TextView
    private lateinit var passwordContainer: TextInputLayout
    private lateinit var passwordInput: TextInputEditText
    private lateinit var submitButton: Button
    private lateinit var footerTextView: TextView

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        return inflater.inflate(R.layout.username_and_password_sign_in, container,  attachToRoot: false)
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        val context = requireContext()

        toolbar = view.findViewById(R.id.toolbar)
        appIcon = view.findViewById(R.id.app_icon)
        primaryTextView = view.findViewById(R.id.primary_message)
        passwordContainer = view.findViewById(R.id.password_container)
        passwordInput = view.findViewById(R.id.password_input)
        submitButton = view.findViewById(R.id.submit_button)
        footerTextView = view.findViewById(R.id.footer)

        toolbar.setNavigationOnClickListener { it: View!
            requireActivity().supportFragmentManager.popBackStack()
        }

        appIcon.setImageDrawable(ContextCompat.getDrawable(context, R.drawable.aural_logo))
        primaryTextView.text = "Sign in with Aural"
        passwordContainer.hint = "Enter password"
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            passwordInput.setAutofillHints(View.AUTOFILL_HINT_PASSWORD)
        }
```



```
        toolbar = view.findViewById(R.id.toolbar)
        appIcon = view.findViewById(R.id.app_icon)
        primaryTextView = view.findViewById(R.id.primary_message)
        passwordContainer = view.findViewById(R.id.password_container)
        passwordInput = view.findViewById(R.id.password_input)
        submitButton = view.findViewById(R.id.submit_button)
        footerTextView = view.findViewById(R.id.footer)

        toolbar.setNavigationOnClickListener { it: View!
            requireActivity().supportFragmentManager.popBackStack()
        }

        appIcon.setImageDrawable(ContextCompat.getDrawable(context, R.drawable.aural_logo))
        primaryTextView.text = "Sign in with Aural"
        passwordContainer.hint = "Enter password"
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            passwordInput.setAutofillHints(View.AUTOFILL_HINT_PASSWORD)
        }

        footerTextView.text = HtmlCompat.fromHtml(
            "By signing in with Aural, you agree to the <a href=http...",
            HtmlCompat.FROM_HTML_MODE_LEGACY
        )
        footerTextView.movementMethod = LinkMovementMethod.getInstance()

        val userId = arguments?.getString(SignInLandingPageFragment.CAR_SIGN_IN_IDENTIFIER_KEY)

        submitButton.text = "Next"
        submitButton.setOnClickListener { it: View!
            onSignIn(userId!!, passwordInput.text.toString())
        }
    }

    private fun onSignIn(userIdentifier: CharSequence, password: CharSequence) {
        ViewModelProvider(requireActivity())
            .get(SignInActivityViewModel::class.java)
            .login(userIdentifier.toString(), password.toString())
    }
}
```


package com.example.android.uamp.automotive;
import ...
@RunWith(AndroidJUnit4.class)
public class ExampleInstrumentedTest {
@Test
public void useAppContext() {
Context appContext = InstrumentationRegistry.getTargetContext();
assertEquals( expected: "com.example.automotive", appContext.getPackageName());
}
}
Kotlin
A new version 1.4.32-release-Studio4.1-1 of the Kotlin plugin is available. Install
Android Studio 4.1.3 available
Update...


package com.example.android.uamp.automotive;
import ...
public class ExampleUnitTest {
@Test
public void addition_isCorrect() { assertEquals( expected: 4, actual: 2 + 2); }
}
Kotlin
A new version 1.4.32-release-Studio4.1-1 of the Kotlin plugin is available. Install
Android Studio 4.1.3 available
Update...

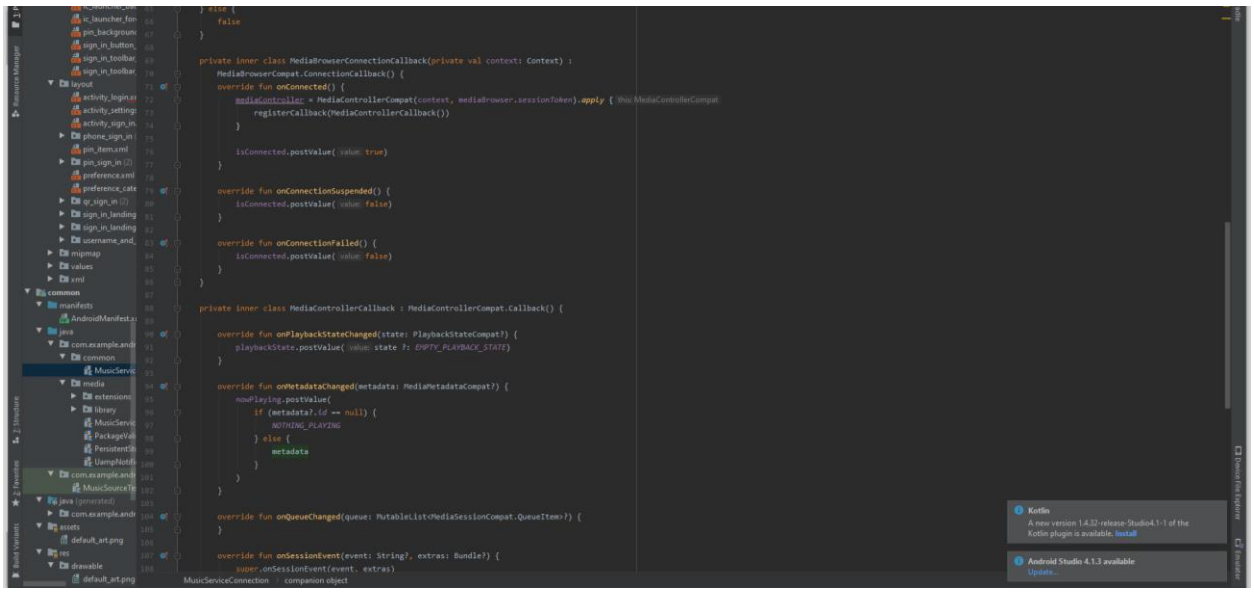


```
package com.example.android.uamp.common

import ...

class MusicServiceConnection(context: Context, serviceComponent: ComponentName) {
    val isConnected = MutableLiveData<Boolean>()
        .apply { postValue( value: false) }
    val networkFailure = MutableLiveData<Boolean>()
        .apply { postValue( value: false) }

    val rootMediaId: String get() = mediaBrowser.root

    val playbackState = MutableLiveData<PlaybackStateCompat>()
        .apply { postValue(EMPTY_PLAYBACK_STATE) }
    val nowPlaying = MutableLiveData<MediaMetadataCompat>()
        .apply { postValue(NOTHING_PLAYING) }

    val transportControls: MediaControllerCompat.TransportControls
        get() = mediaController.transportControls

    private val mediaBrowserConnectionCallback = MediaBrowserConnectionCallback(context)
    private val mediaBrowser = MediaBrowserCompat(
        context,
        serviceComponent,
        mediaBrowserConnectionCallback, rootHints: null
    ).apply { connect() }
    private lateinit var mediaController: MediaControllerCompat

    fun subscribe(parentId: String, callback: MediaBrowserCompat.SubscriptionCallback) {
        mediaBrowser.subscribe(parentId, callback)
    }

    fun unsubscribe(parentId: String, callback: MediaBrowserCompat.SubscriptionCallback) {
        mediaBrowser.unsubscribe(parentId, callback)
    }

    fun sendCommand(command: String, parameters: Bundle?) =
        sendCommand(command, parameters) { _, _ -> }

    fun sendCommand(
        command: String,
        parameters: Bundle?,
        resultCallback: ((Int, Bundle?) -> Unit)
    ) = if (mediaBrowser.isConnected) {
```

Kotlin
A new version 1.4.32-release-Studio4.1-1 of the Kotlin plugin is available. Install

Android Studio 4.1.3 available
Update...

MusicServiceConnection › companion object

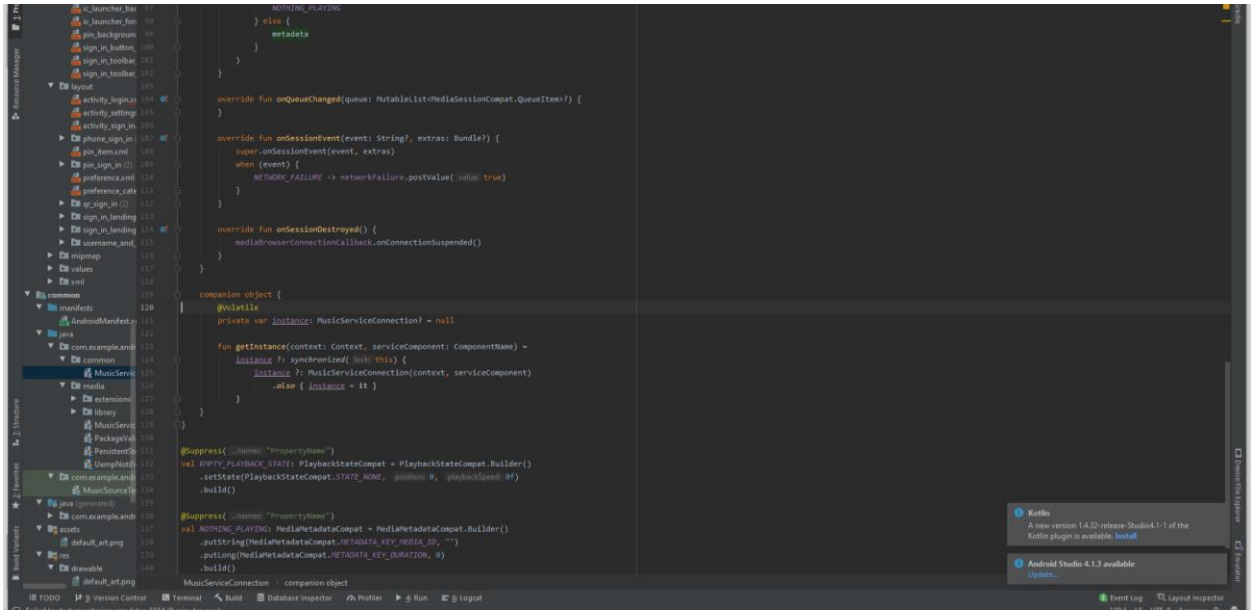


```
    } else {
        false
    }

    private inner class MediaBrowserConnectionCallback(private val context: Context) :
        MediaBrowserCompat.ConnectionCallback() {
        override fun onConnected() {
            mediaController = MediaControllerCompat(context, mediaBrowser.sessionToken).apply { this: MediaControllerCompat
                registerCallback(MediaControllerCallback())
            }

            isConnected.postValue( value: true)
        }

        override fun onConnectionSuspended() {
            isConnected.postValue( value: false)
        }

        override fun onConnectionFailed() {
            isConnected.postValue( value: false)
        }
    }

    private inner class MediaControllerCallback : MediaControllerCompat.Callback() {

        override fun onPlaybackStateChanged(state: PlaybackStateCompat?) {
            playbackState.postValue( value: state ?: EMPTY_PLAYBACK_STATE)
        }

        override fun onMetadataChanged(metadata: MediaMetadataCompat?) {
            nowPlaying.postValue(
                if (metadata?.id == null) {
                    NOTHING_PLAYING
                } else {
                    metadata
                }
            )
        }

        override fun onQueueChanged(queue: MutableList<MediaSessionCompat.QueueItem>?) {
        }

        override fun onSessionEvent(event: String?, extras: Bundle?) {
            super.onSessionEvent(event, extras)
```

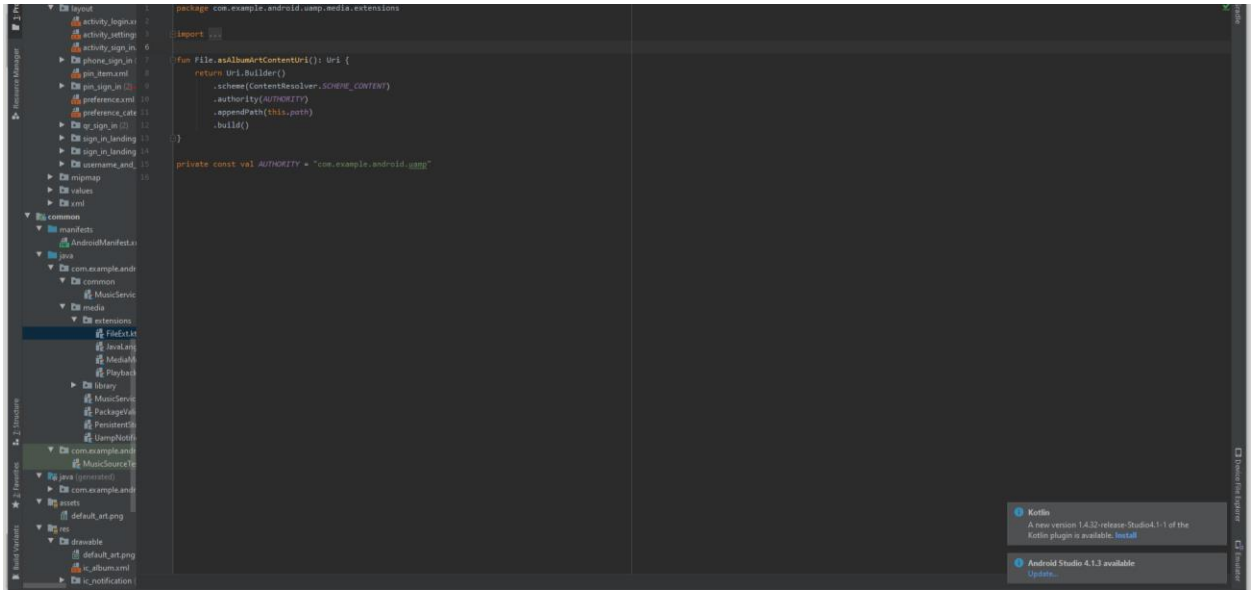
Kotlin
A new version 1.4.32-release-Studio4.1-1 of the Kotlin plugin is available. Install

Android Studio 4.1.3 available
Update...

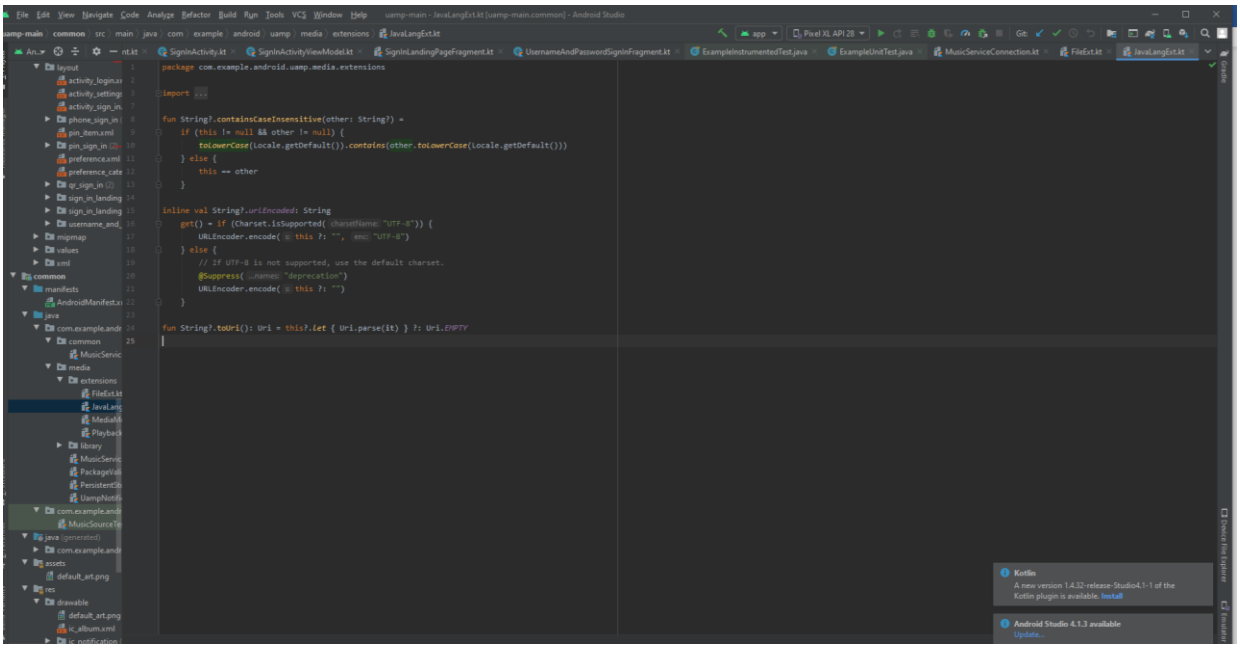
MusicServiceConnection › companion object



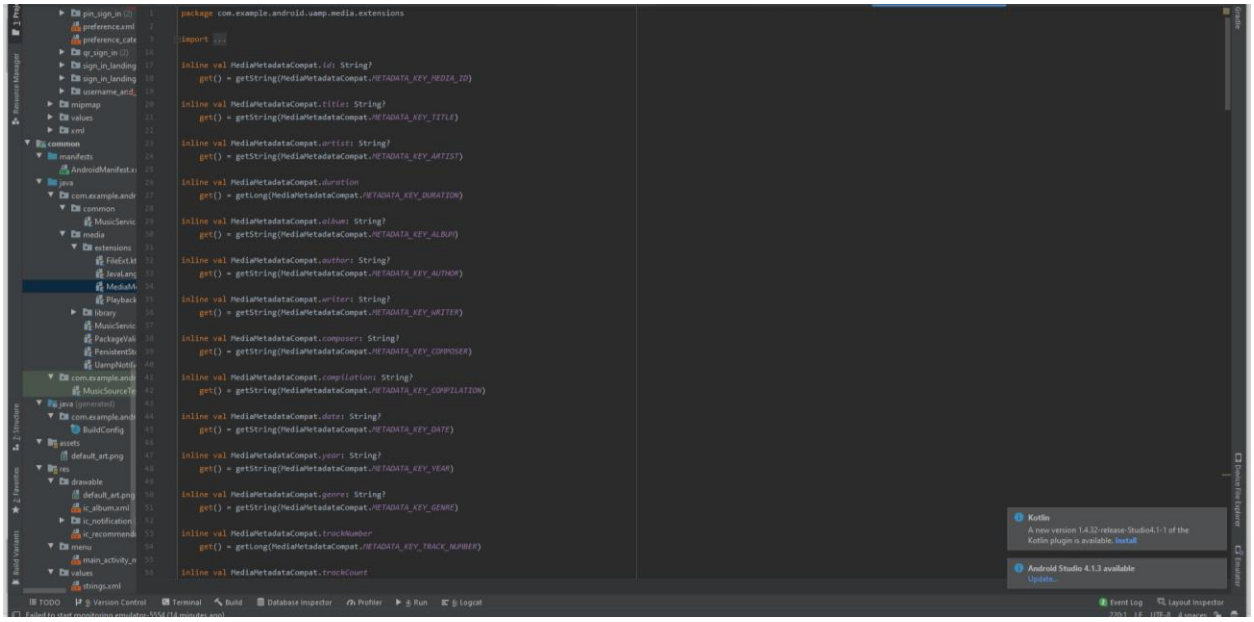
```
                    NOTHING_PLAYING
                } else {
                    metadata
                }
            }
        }

        override fun onQueueChanged(queue: MutableList<MediaSessionCompat.QueueItem>?) {
        }

        override fun onSessionEvent(event: String?, extras: Bundle?) {
            super.onSessionEvent(event, extras)
            when (event) {
                NETWORK_FAILURE -> networkFailure.postValue( value: true)
            }
        }

        override fun onSessionDestroyed() {
            mediaBrowserConnectionCallback.onConnectionSuspended()
        }
    }

    companion object {
        @Volatile
        private var instance: MusicServiceConnection? = null

        fun getInstance(context: Context, serviceComponent: ComponentName) =
            instance ?: synchronized( lock: this) {
                instance ?: MusicServiceConnection(context, serviceComponent)
                    .also { instance = it }
            }
    }
}

@Suppress( ...names: "PropertyName")
val EMPTY_PLAYBACK_STATE: PlaybackStateCompat = PlaybackStateCompat.Builder()
    .setState(PlaybackStateCompat.STATE_NONE,  position: 0,  playbackSpeed: 0f)
    .build()

@Suppress( ...names: "PropertyName")
val NOTHING_PLAYING: MediaMetadataCompat = MediaMetadataCompat.Builder()
    .putString(MediaMetadataCompat.METADATA_KEY_MEDIA_ID, "")
    .putLong(MediaMetadataCompat.METADATA_KEY_DURATION, 0)
    .build()
```



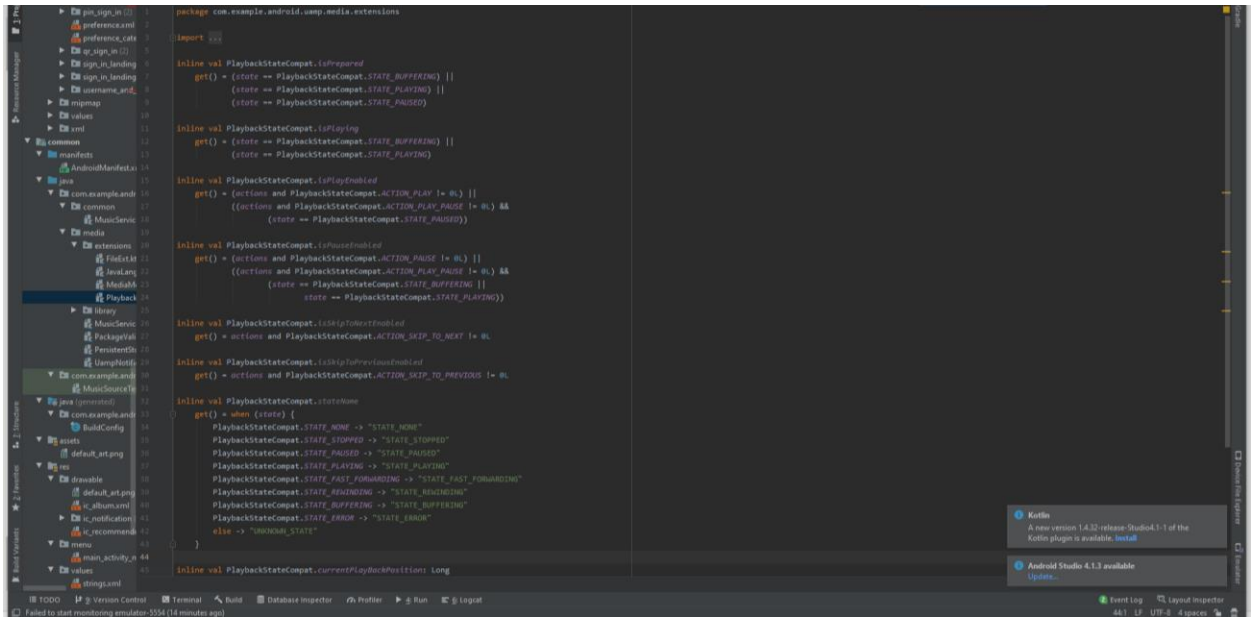
```
package com.example.android.uamp.media.extensions

import ...

fun File.asAlbumArtContentUri(): Uri {
    return Uri.Builder()
        .scheme(ContentResolver.SCHEME_CONTENT)
        .authority(AUTHORITY)
        .appendPath(this.path)
        .build()
}

private const val AUTHORITY = "com.example.android.uamp"
```



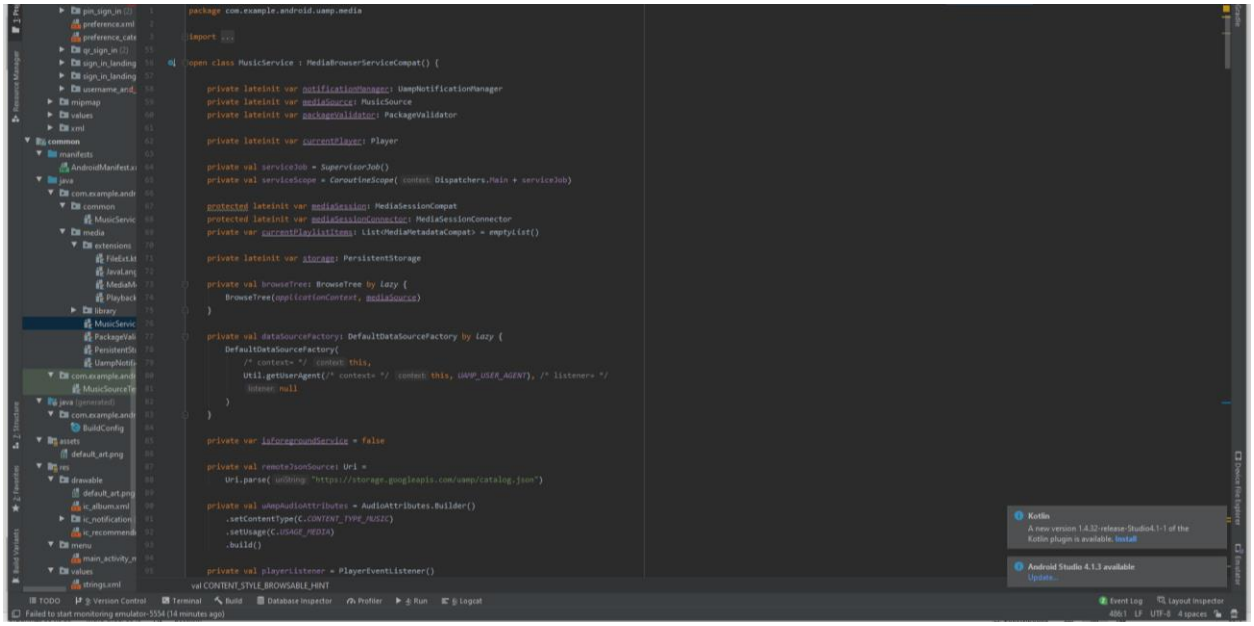
```
package com.example.android.uamp.media.extensions

import ...

fun String?.containsCaseInsensitive(other: String?) =
    if (this != null && other != null) {
        toLowerCase(Locale.getDefault()).contains(other.toLowerCase(Locale.getDefault()))
    } else {
        this == other
    }

inline val String?.urlEncoded: String
    get() = if (Charset.isSupported( charsetName: "UTF-8")) {
        URLEncoder.encode( s: this ?: "",  enc: "UTF-8")
    } else {
        // If UTF-8 is not supported, use the default charset.
        @Suppress( ...names: "deprecation")
        URLEncoder.encode( s: this ?: "")
    }

fun String?.toUri(): Uri = this?.let { Uri.parse(it) } ?: Uri.EMPTY
```



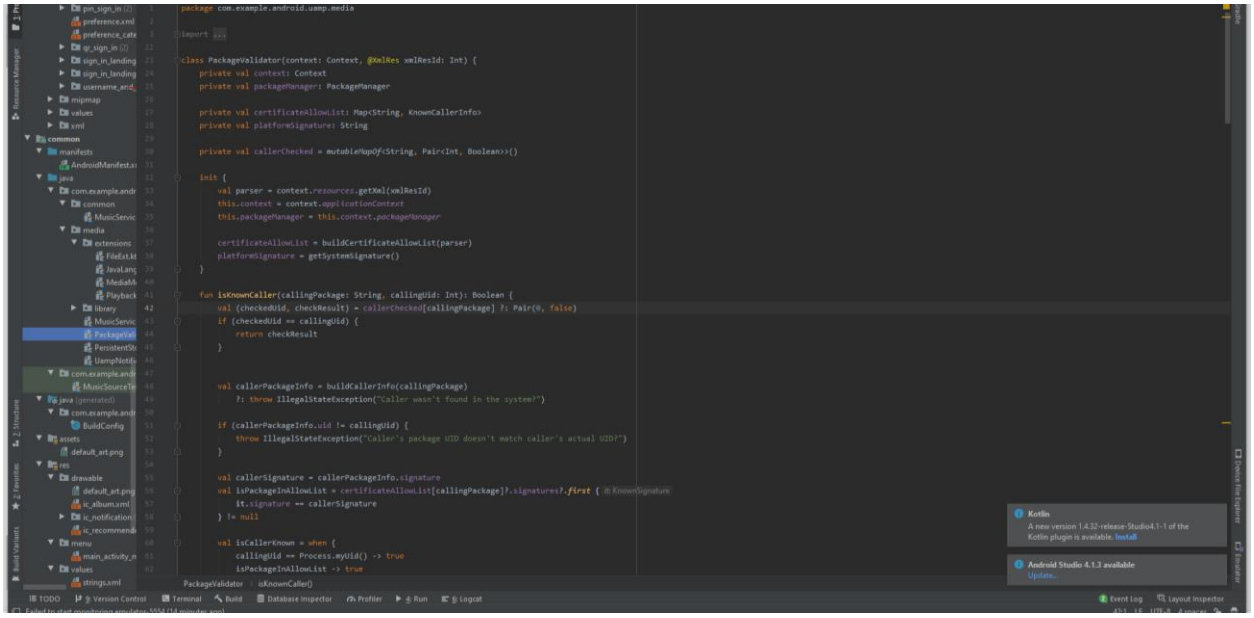
```
package com.example.android.uamp.media.extensions

import ...

inline val MediaMetadataCompat.id: String?
    get() = getString(MediaMetadataCompat.METADATA_KEY_MEDIA_ID)

inline val MediaMetadataCompat.title: String?
    get() = getString(MediaMetadataCompat.METADATA_KEY_TITLE)

inline val MediaMetadataCompat.artist: String?
    get() = getString(MediaMetadataCompat.METADATA_KEY_ARTIST)

inline val MediaMetadataCompat.duration
    get() = getLong(MediaMetadataCompat.METADATA_KEY_DURATION)

inline val MediaMetadataCompat.album: String?
    get() = getString(MediaMetadataCompat.METADATA_KEY_ALBUM)

inline val MediaMetadataCompat.author: String?
    get() = getString(MediaMetadataCompat.METADATA_KEY_AUTHOR)

inline val MediaMetadataCompat.writer: String?
    get() = getString(MediaMetadataCompat.METADATA_KEY_WRITER)

inline val MediaMetadataCompat.composer: String?
    get() = getString(MediaMetadataCompat.METADATA_KEY_COMPOSER)

inline val MediaMetadataCompat.compilation: String?
    get() = getString(MediaMetadataCompat.METADATA_KEY_COMPILATION)

inline val MediaMetadataCompat.date: String?
    get() = getString(MediaMetadataCompat.METADATA_KEY_DATE)

inline val MediaMetadataCompat.year: String?
    get() = getString(MediaMetadataCompat.METADATA_KEY_YEAR)

inline val MediaMetadataCompat.genre: String?
    get() = getString(MediaMetadataCompat.METADATA_KEY_GENRE)

inline val MediaMetadataCompat.trackNumber
    get() = getLong(MediaMetadataCompat.METADATA_KEY_TRACK_NUMBER)

inline val MediaMetadataCompat.trackCount
```



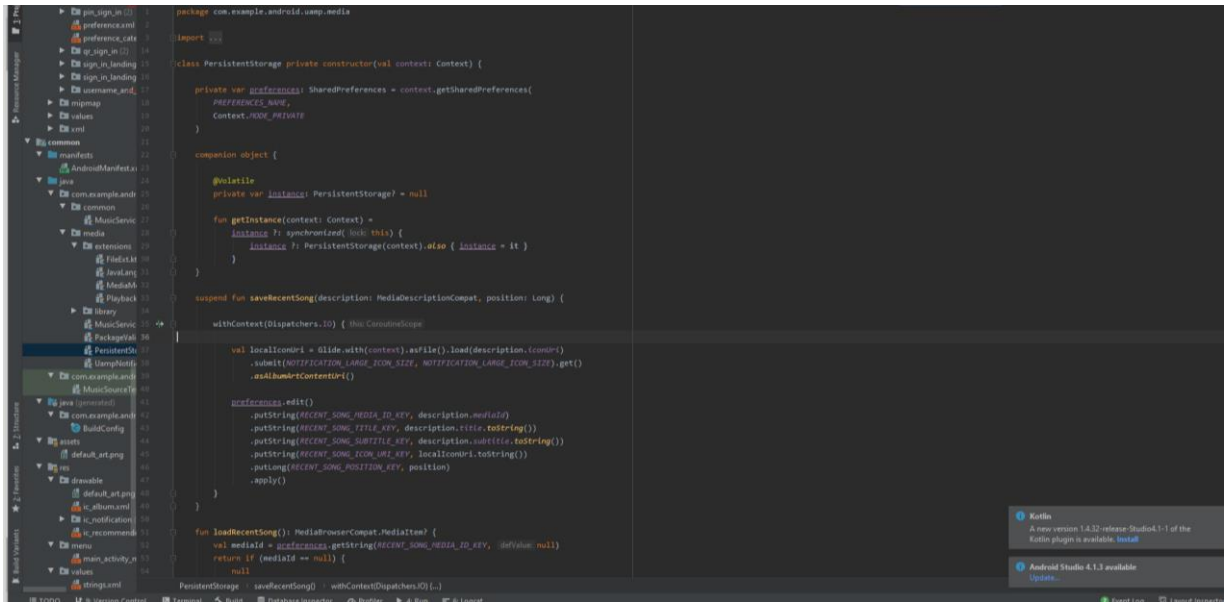
```
package com.example.android.uamp.media.extensions

import ...

inline val PlaybackStateCompat.isPrepared
    get() = (state == PlaybackStateCompat.STATE_BUFFERING) ||
            (state == PlaybackStateCompat.STATE_PLAYING) ||
            (state == PlaybackStateCompat.STATE_PAUSED)

inline val PlaybackStateCompat.isPlaying
    get() = (state == PlaybackStateCompat.STATE_BUFFERING) ||
            (state == PlaybackStateCompat.STATE_PLAYING)

inline val PlaybackStateCompat.isPlayEnabled
    get() = (actions and PlaybackStateCompat.ACTION_PLAY != 0L) ||
            ((actions and PlaybackStateCompat.ACTION_PLAY_PAUSE != 0L) &&
                    (state == PlaybackStateCompat.STATE_PAUSED))

inline val PlaybackStateCompat.isPauseEnabled
    get() = (actions and PlaybackStateCompat.ACTION_PAUSE != 0L) ||
            ((actions and PlaybackStateCompat.ACTION_PLAY_PAUSE != 0L) &&
                    (state == PlaybackStateCompat.STATE_BUFFERING ||
                            state == PlaybackStateCompat.STATE_PLAYING))

inline val PlaybackStateCompat.isSkipToNextEnabled
    get() = actions and PlaybackStateCompat.ACTION_SKIP_TO_NEXT != 0L

inline val PlaybackStateCompat.isSkipToPreviousEnabled
    get() = actions and PlaybackStateCompat.ACTION_SKIP_TO_PREVIOUS != 0L

inline val PlaybackStateCompat.stateName
    get() = when (state) {
        PlaybackStateCompat.STATE_NONE -> "STATE_NONE"
        PlaybackStateCompat.STATE_STOPPED -> "STATE_STOPPED"
        PlaybackStateCompat.STATE_PAUSED -> "STATE_PAUSED"
        PlaybackStateCompat.STATE_PLAYING -> "STATE_PLAYING"
        PlaybackStateCompat.STATE_FAST_FORWARDING -> "STATE_FAST_FORWARDING"
        PlaybackStateCompat.STATE_REWINDING -> "STATE_REWINDING"
        PlaybackStateCompat.STATE_BUFFERING -> "STATE_BUFFERING"
        PlaybackStateCompat.STATE_ERROR -> "STATE_ERROR"
        else -> "UNKNOWN_STATE"
    }

inline val PlaybackStateCompat.currentPlayBackPosition: Long
```



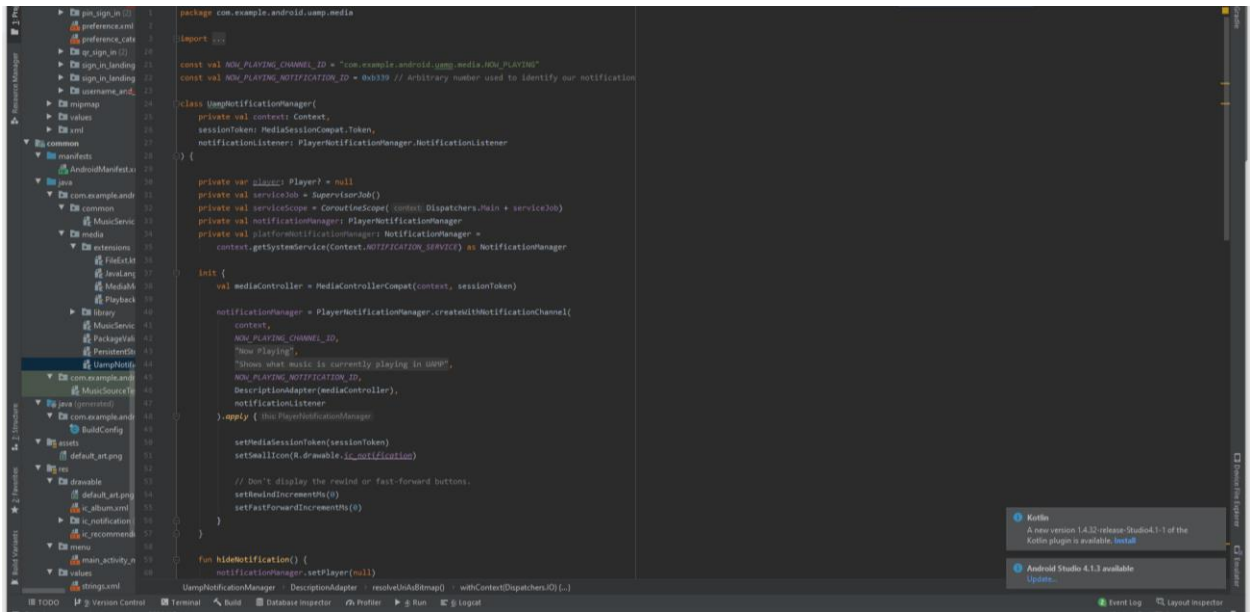
```
package com.example.android.uamp.media

import ...

open class MusicService : MediaBrowserServiceCompat() {

    private lateinit var notificationManager: UampNotificationManager
    private lateinit var mediaSource: MusicSource
    private lateinit var packageValidator: PackageValidator

    private lateinit var currentPlayer: Player

    private val serviceJob = SupervisorJob()
    private val serviceScope = CoroutineScope( context: Dispatchers.Main + serviceJob)

    protected lateinit var mediaSession: MediaSessionCompat
    protected lateinit var mediaSessionConnector: MediaSessionConnector
    private var currentPlaylistItems: List<MediaMetadataCompat> = emptyList()

    private lateinit var storage: PersistentStorage

    private val browseTree: BrowseTree by lazy {
        BrowseTree(applicationContext, mediaSource)
    }

    private val dataSourceFactory: DefaultDataSourceFactory by lazy {
        DefaultDataSourceFactory(
            /* context= */ context: this,
            Util.getUserAgent(/* context= */ context: this, UAMP_USER_AGENT), /* listener= */
            listener: null
        )
    }

    private var isForegroundService = false

    private val remoteJsonSource: Uri =
        Uri.parse( uriString: "https://storage.googleapis.com/uamp/catalog.json")

    private val uAmpAudioAttributes = AudioAttributes.Builder()
        .setContentType(C.CONTENT_TYPE_MUSIC)
        .setUsage(C.USAGE_MEDIA)
        .build()

    private val playerListener = PlayerEventListener()

    val CONTENT_STYLE_BROWSABLE_HINT
```



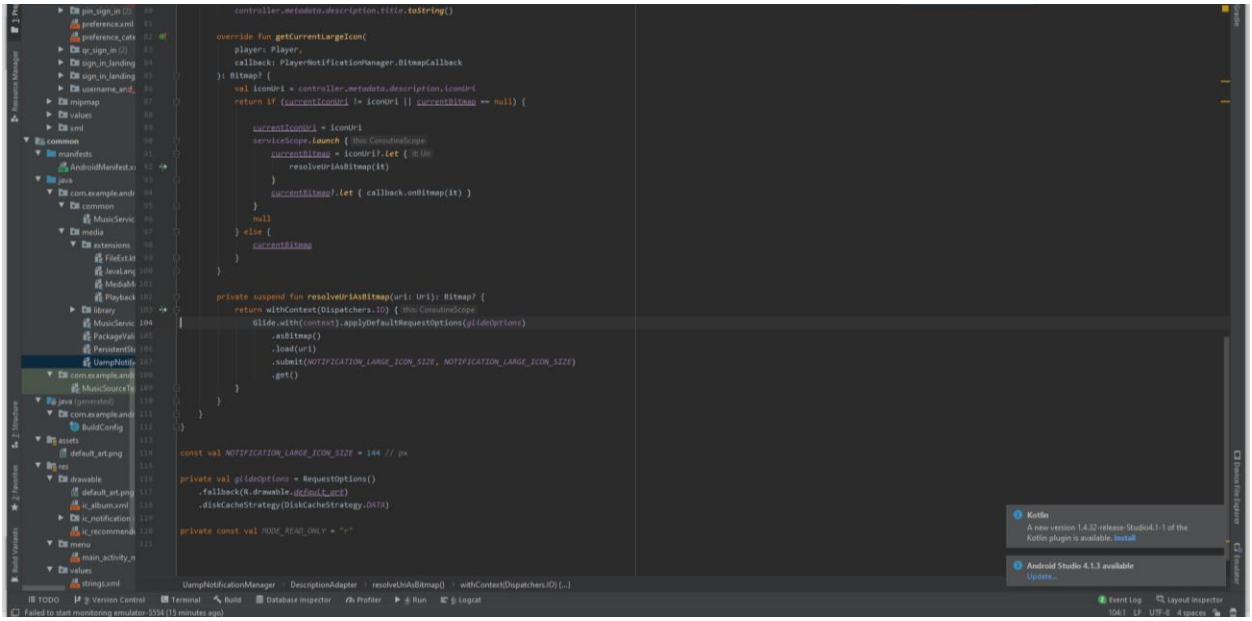
```
package com.example.android.uamp.media

import ...

class PackageValidator(context: Context, @XmlRes xmlResId: Int) {
    private val context: Context
    private val packageManager: PackageManager

    private val certificateAllowList: Map<String, KnownCallerInfo>
    private val platformSignature: String

    private val callerChecked = mutableMapOf<String, Pair<Int, Boolean>>()

    init {
        val parser = context.resources.getXml(xmlResId)
        this.context = context.applicationContext
        this.packageManager = this.context.packageManager

        certificateAllowList = buildCertificateAllowList(parser)
        platformSignature = getSystemSignature()
    }

    fun isKnownCaller(callingPackage: String, callingUid: Int): Boolean {
        val (checkedUid, checkResult) = callerChecked[callingPackage] ?: Pair(0, false)
        if (checkedUid == callingUid) {
            return checkResult
        }


        val callerPackageInfo = buildCallerInfo(callingPackage)
            ?: throw IllegalStateException("Caller wasn't found in the system?")

        if (callerPackageInfo.uid != callingUid) {
            throw IllegalStateException("Caller's package UID doesn't match caller's actual UID?")
        }

        val callerSignature = callerPackageInfo.signature
        val isPackageInAllowList = certificateAllowList[callingPackage]?.signatures?.first {
            it.signature == callerSignature
        } != null

        val isCallerKnown = when {
            callingUid == Process.myUid() -> true
            isPackageInAllowList -> true
```



```
package com.example.android.uamp.media

import ...

class PersistentStorage private constructor(val context: Context) {

    private var preferences: SharedPreferences = context.getSharedPreferences(
        PREFERENCES_NAME,
        Context.MODE_PRIVATE
    )

    companion object {

        @Volatile
        private var instance: PersistentStorage? = null

        fun getInstance(context: Context) =
            instance ?: synchronized( lock: this) {
                instance ?: PersistentStorage(context).also { instance = it }
            }
    }

    suspend fun saveRecentSong(description: MediaDescriptionCompat, position: Long) {

        withContext(Dispatchers.IO) { this: CoroutineScope

            val localIconUri = Glide.with(context).asFile().load(description.iconUri)
                .submit(NOTIFICATION_LARGE_ICON_SIZE, NOTIFICATION_LARGE_ICON_SIZE).get()
                .asAlbumArtContentUri()

            preferences.edit()
                .putString(RECENT_SONG_MEDIA_ID_KEY, description.mediaId)
                .putString(RECENT_SONG_TITLE_KEY, description.title.toString())
                .putString(RECENT_SONG_SUBTITLE_KEY, description.subtitle.toString())
                .putString(RECENT_SONG_ICON_URI_KEY, localIconUri.toString())
                .putLong(RECENT_SONG_POSITION_KEY, position)
                .apply()
        }
    }

    fun loadRecentSong(): MediaBrowserCompat.MediaItem? {
        val mediaId = preferences.getString(RECENT_SONG_MEDIA_ID_KEY,  defValue: null)
        return if (mediaId == null) {
            null
```



```
package com.example.android.uamp.media

import ...

const val NOW_PLAYING_CHANNEL_ID = "com.example.android.uamp.media.NOW_PLAYING"
const val NOW_PLAYING_NOTIFICATION_ID = 0xb339 // Arbitrary number used to identify our notification

class UampNotificationManager(
    private val context: Context,
    sessionToken: MediaSessionCompat.Token,
    notificationListener: PlayerNotificationManager.NotificationListener
) {

    private var player: Player? = null
    private val serviceJob = SupervisorJob()
    private val serviceScope = CoroutineScope( context: Dispatchers.Main + serviceJob)
    private val notificationManager: PlayerNotificationManager
    private val platformNotificationManager: NotificationManager =
        context.getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager

    init {
        val mediaController = MediaControllerCompat(context, sessionToken)

        notificationManager = PlayerNotificationManager.createWithNotificationChannel(
            context,
            NOW_PLAYING_CHANNEL_ID,
            "Now Playing",
            "Shows what music is currently playing in UAMP",
            NOW_PLAYING_NOTIFICATION_ID,
            DescriptionAdapter(mediaController),
            notificationListener
        ).apply { this: PlayerNotificationManager

            setMediaSessionToken(sessionToken)
            setSmallIcon(R.drawable.ic_notification)

            // Don't display the rewind or fast-forward buttons.
            setRewindIncrementMs(0)
            setFastForwardIncrementMs(0)
        }
    }

    fun hideNotification() {
        notificationManager.setPlayer(null)
```



```
            controller.metadata.description.title.toString()

        override fun getCurrentLargeIcon(
            player: Player,
            callback: PlayerNotificationManager.BitmapCallback
        ): Bitmap? {
            val iconUri = controller.metadata.description.iconUri
            return if (currentIconUri != iconUri || currentBitmap == null) {

                currentIconUri = iconUri
                serviceScope.launch { this: CoroutineScope
                    currentBitmap = iconUri?.let { it: Uri
                        resolveUriAsBitmap(it)
                    }
                    currentBitmap?.let { callback.onBitmap(it) }
                }
                null
            } else {
                currentBitmap
            }
        }

        private suspend fun resolveUriAsBitmap(uri: Uri): Bitmap? {
            return withContext(Dispatchers.IO) { this: CoroutineScope
                Glide.with(context).applyDefaultRequestOptions(glideOptions)
                    .asBitmap()
                    .load(uri)
                    .submit(NOTIFICATION_LARGE_ICON_SIZE, NOTIFICATION_LARGE_ICON_SIZE)
                    .get()
            }
        }
    }
}

const val NOTIFICATION_LARGE_ICON_SIZE = 144 // px

private val glideOptions = RequestOptions()
    .fallback(R.drawable.default_art)
    .diskCacheStrategy(DiskCacheStrategy.DATA)

private const val MODE_READ_ONLY = "r"
```