



UNIWERSYTET RZESZOWSKI

Kolegium Nauk Przyrodniczych

Magdalena Rowicka

99780

Informatyka

**Projekt i implementacja aplikacji wspomagającej naukę gry na gitarze**

**Praca:** inżynierska

**Praca wykonana pod kierunkiem:**  
dr hab. inż. prof. UR Krzysztof Pancerz

Rzeszów, 2020





RZESZOW UNIVERSITY

College of Natural Sciences

Magdalena Rowicka

99780

Computer Science

**Design and implementation of an application supporting guitar learning**

**Type of the thesis:** Engineer

**The thesis written under the supervision of**  
dr hab. inż. prof. UR Krzysztof Pancerz

Rzeszów, 2020



# **Spis treści**

|        |  |    |
|--------|--|----|
| 1.     | Wstęp i cel pracy .....                                    | 7  |
| 2.     | Opis rozwiązywanego problemu .....                         | 9  |
| 2.1.   | Specyfikacja problemu.....                                 | 9  |
| 2.2.   | Przegląd istniejących rozwiązań .....                      | 10 |
| 3.     | Aspekty technologiczne i metodyka realizacji projektu..... | 12 |
| 3.1.   | Wykorzystane technologie .....                             | 12 |
| 3.1.1. | Android Studio.....  | 12 |
| 3.1.2. | Java .....   | 13 |
| 3.1.3. | Gradle.....  | 13 |
| 3.1.4. | ADB .....  | 13 |
| 3.1.5. | Firebase.....  | 14 |
| 3.1.6. | GIMP .....   | 14 |
| 3.2.   | Dodatkowe biblioteki .....                                 | 15 |
| 3.2.1. | Biblioteka TarsosDSP .....                                 | 15 |
| 3.2.2. | Biblioteka Picasso.....                                    | 17 |
| 3.3.   | Specyfikacja wymagań funkcjonalnych aplikacji.....         | 17 |
| 4.     | Przedstawienie systemu.....                                | 20 |
| 4.1.   | Strojenie gitary .....                                     | 20 |
| 4.2.   | Metronom.....  | 23 |
| 4.3.   | Profil użytkownika .....                                   | 24 |
| 4.4.   | Nauka akordów .....  | 28 |
| 4.5.   | Ranking .....  | 34 |
| 5.     | Implementacja systemu .....                                | 36 |
| 5.1.   | Kompozycja struktury folderów .....                        | 36 |
| 5.2.   | Implementacja stroika .....                                | 38 |
| 5.3.   | Implementacja metronomu.....                               | 40 |

|      |   |    |
|------|---|----|
| 5.4. | Tworzenie i uzupełnienie bazy danych z poziomu aplikacji..... | 42 |
| 5.5. | Administracja użytkownikami.....                              | 46 |
| 5.6. | Sprawdzenie poprawności akordów .....                         | 49 |
| 5.7. | Wyświetlenie rankingu z podziałem na poszczególne grupy.....  | 52 |
| 6.   | Prezentacja systemu .....                                     | 55 |
| 6.1. | Obsługa aplikacji przez użytkownika niezalogowanego .....     | 55 |
| 6.2. | Obsługa aplikacji przez użytkownika zalogowanego.....         | 64 |
| 7.   | Uruchomienie systemu.....                                     | 70 |
| 8.   | Podsumowanie .....  | 77 |
| 9.   | Literatura.....   | 78 |
|      | Wykaz rysunków .....  | 80 |
|      | Streszczenie pracy .....                                      | 85 |
|      | Zawartość płyty .....   | 86 |

## **1. Wstęp i cel pracy**

Co raz więcej czasu poświęcamy telefonom komórkowym co nie uchodzi uwadze projektantom systemów informatycznych. Rosnąca ilość nowych aplikacji mobilnych, dostępnych chociażby na platformie Google Play oraz stale rosnąca ilość pobrania aplikacji zdaje się potwierdzać, że ściągamy i korzystamy z coraz liczniejszych aplikacji. Przeglądając platformę Google Play natknąć się można na wiele kategorii aplikacji i gier. Możemy tam znaleźć aplikacje do zarządzania czasem, nauki języków oraz wspomagające zapamiętywanie. Przeglądając nowe aplikacje można zauważyc, że aplikacje do nauki gry na jakimś instrumencie zdają się pojawiać co raz częściej.

Największą popularnością wśród instrumentów od wielu lat cieszy się gitara. Wpływ na taki stan rzeczy może mieć fakt, że gitara jest bardzo prosta w swojej budowie, stosunkowo tania oraz uczenie się nowych akordów na tym instrumencie jest stosunkowo proste. Problemem jaki mogą napotkać gitarzyści-samouki to nastrojenie gitary, trzymanie tempa granych akordów/dźwięków czy nawet poprawność trzymania akordu. Ostatni problem, czyli poprawne trzymanie akordu, można podzielić na dwa mniejsze problemy. Otóż, pierwszy z nich to ułożenie palców na gryfie, a drugi to siła z jaką przykładane są palce do gryfu. Jeśli zbyt słabo naciśniemy strunę to dźwięk, który wydobędzie się z pudła rezonansowego będzie niewłaściwy.

Celem projektu jest stworzenie aplikacji, która ułatwi niedoświadczonym gitarzystom postawić pierwsze kroki do nauki na prostym instrumencie, którym jest gitara. Głównym zadaniem ma być pomoc przy nastrojeniu instrumentu posługując się dźwiękami nagrywanymi przez urządzenie. Przy strojeniu gitary, potrzebne są obie ręce. Jedna z nich szarpie za struny, a druga odpowiednio manewruje kluczami. Z tego też powodu aplikacja powinna pozwolić na nastrojenie gitary, bez trzymania telefonu w ręce. Drugim ważnym aspektem przy nauce gry na gitarze jest trzymanie odpowiedniego tempa. Pomocnym urządzeniem do tego zadania jest metronom, który wybija odpowiedni rytm we właściwym tempie. Aplikacja powinna mieć możliwość zasymulowania metronomu, która jak w poprzednim przypadku, powinna być obsługiwana bez potrzeby trzymania telefonu w dłoni. Trzecim aspektem są akordy. Zaczynając od łatwych do chwycenia aż do tych, na które trzeba poświęcić więcej czasu na poprawne chwycenie – każdy gitarzysta powinien znać podstawowe akordy. To one w głównej mierze odpowiadają za melodię graną w utworach. Tak więc aplikacja powinna wspomóc młodego gitarzystę na

tych trzech płaszczyznach. Powinna pomóc mu nastroić gitarę, nauczyć podstawowych akordów oraz nauczyć trzymania odpowiedniego tempa podczas grania.

## **2. Opis rozwiązywanego problemu**

### **2.1.Specyfikacja problemu**

Stawiając pierwsze kroki w nauce gry na instrumencie wielu zastanawiało się jak to zrobić. Oczywiście mam tu na myśli tych młodych muzyków, którzy chcieli się nauczyć samemu grać na wybranym instrumencie. Przed młodym muzykiem stawiane są trudne, jak na razie, działania.

Jednym z takich działań jest nastrojenie gitary. Jest to najważniejszy z punktów, który musi przejść każdy muzyk. Jest to działanie, które musi wykonać zawsze przed rozpoczęciem grania na instrumencie. Jest to zadanie czasem bardzo trudne, a zwłaszcza dla niedoświadczonych muzyków. Pierwszym punktem strojenia jest wybranie strojenia (naciągnięcie strun, w taki sposób aby dźwięk na strunie, którą szarpiemy, ale nie przyciskamy na żadnym z progów, był odpowiedni). W zależności od gatunku muzycznego w jakim chcemy grać, gitara otrzyma inne strój. Gdy młody gitarzysta wybierze już odpowiedni strój gitary, przechodzi do właściwego strojenia naciągając struny poprzez klucze umieszczone na główce gitary. Młody gitarzysta, który nie ma wyrobionego na tyle słuchu, aby samodzielnie nastroić instrument, ma utrudnione zadanie. Musi wiedzieć jaki dźwięk chce uzyskać, a nie znając go na tyle dobrze, aby określić czy dźwięk jest prawidłowy, wtenczas zadanie staje się trudne, jeśli nie awykonalne. Z pomocą przychodzą stroiki, które określając częstotliwość ustalają dźwięk, który wydobył się z gitary. Manewrując kluczami gitarzysta przybliża się do wymaganego dźwięku.

Drugą trudnością z jaką spotka się gitarzysta-samouk jest trzymanie tempa. Jest ono bardzo istotną częścią podczas nie tylko gry na instrumencie, co nawet śpiewania lub w tańcu. Przykładem tego mogą być same piosenki. Wiele z nich ma ten sam układ akordów, jednak bicie i tempo grania jest inne co sprawia, że słyszalna melodia jest inna. Metronom, w tym przypadku jest odpowiednim narzędziem. Działa on na zasadzie wybijania dźwięku co jakiś określony czas. Nowoczesne narzędzia opierają się właśnie na tej zasadzie. Jednak jest to kolejne narzędzie, które trzeba mieć podczas nauki gry.

Trzecią sprawą jest to, na co wszyscy młodzi gitarzyści czekają – nauka akordów. Jest to bardzo ważna część grania na gitarze. Trzymając odpowiednio palce na gryfie gitary i przyciskając dostatecznie mocno struny do gryfu, ale jednocześnie nie dotykając w żaden sposób innych strun, wydobywamy upragniony dźwięk z gitary szarpiąc odpowiednie struny. Jest to dość skomplikowana rzecz do nauki. Jednak gdy opanuje się podstawy, czyli siłę nacisku na struny przy danej gitarze oraz sposób ułożenia palców, aby nie

dotykały przez przypadek innych strun, nauka akordów staje się łatwiejsza. Jednak aby dalej przejść, trzeba wiedzieć jak ułożyć palce na gryfie. Tu potrzebne są schematy akordów, zazwyczaj zebrane w mała książeczkę. Od którego akordu zacząć? Jakie warto nauczyć się na początku, a jakie zostawić na później? Są to bardzo ważne kwestie, ponieważ zbyt trudne akordy na początku mogą zniechęcić młodych gitarzystów.

Podsumowując te trzy punkty. Mamy w kieszeni stroik, metronom oraz książeczkę z schematami akordów. Jest to bardzo dobry początek. Co jeśli stroik albo metronom zgubi się gdzieś? Te nowe urządzenia są zazwyczaj małe, więc bardzo łatwo można je zawieruszyć. Otwierając książeczkę akordów i zobaczywszy ich masę oraz dziwne i niezrozumiałe znaki, młody gitarzysta, od razu ją zamyka. Gdyby tak tylko to ułatwić. Dlatego też funkcje metronomu, stroika i spis akordów łączy się w aplikacje dostępne na wyciągnięcie ręki w telefonie komórkowym typu smartphone.

Taki zestaw dla początkujących jest nie raz wybawieniem, jednak potrzeba czegoś więcej. Trzeba ich zachęcić, podtrzymać zapał do nauki gry na gitarze. Jak to zrobić? Nie od dziś wiadomo, że rywalizacja jest najlepszym sposobem, aby zmotywować kogoś do działania. Tak jesteśmy stworzeni. Kiedyś determinowało to o naszym przetrwaniu (rywalizacja o jedzenie, o miejsce do spania, o miejsce w hierarchii). Dlatego też rywalizacja jest bardzo dobrym sposobem do zachęcenia nowych gitarzystów do nauki gry, na tym bardzo prostym instrumencie jakim jest gitara. Z tego powodu użytkownicy powinni mieć dostęp do rywalizacji między sobą, np. rankingu. Podział takiego rankingu powinien być odpowiedni względem zaawansowania użytkowników. Każda grupa akordów powinna mieć osobny ranking oraz powinien się pojawić ranking zawierający podsumowanie wszystkich grup akordów.

## 2.2. Przegląd istniejących rozwiązań

Aplikacją cieszącą się największą popularnością dostępną na platformie Google Play jest „GuitarTuna” od „Yousician Ltd.”. Posiada ona stroik gitarowy, metronom ze zmiennymi parametrami oraz naukę akordów. Wszystkie funkcjonalności są bardzo dobrze zrobione. Aby nastroić gitarę nie trzeba przełączać strun w aplikacji, a nauka akordów wskazuje, która struna brzmi inaczej niż powinna.

Aplikacja ta jest bardzo dobrze wykonana, jednak ma kilka wad. Pierwszą z nich jest brak dostępu do wszystkich funkcjonalności. Nauka akordów jest ograniczona do kilku podstawowych akordów, reszta jest dostępna po wykupieniu usługi premium. Drugą bardzo poważną wadą jest rozmiar aplikacji. Dla urządzeń mobilnych opartych

o systemy Android jest to bardzo poważną wadą. Ciągłe informacje o braku miejsca na urządzeniu sprawiają, że użytkownicy zmuszeni są do kupna nowego urządzenia lub usunięcia aplikacji. Kolejnym problemem jest ciągłe wyświetlanie reklam, który jest obecny w większości aplikacji dostępnych na platformie Google Play. Jest to aż tak uciążliwe i częste, że firma Google postanowiła walczyć z twórcami aplikacji, którzy nadużywają wyświetlania reklam.

Kolejną aplikacją wartą uwagi jest „Ultimate Tuner” od „Tabs4Acoustic – Free guitar tools”. Ta aplikacja skupia się wyłącznie na strojeniu. Użytkownik ma do wyboru większość strojeń gitarowych z przeróżnych typów muzycznych oraz różne gitary, od gitary klasycznej do hawajskiej wyłącznie. Po kliknięciu w strunę dostaje jej poprawne brzmienie i tak może nastroić ją ze słuchu. Aplikacja ta jest bardzo rozwinięta od strony strojenia, jednak jest to nie wystarczające jeśli chodzi o narzędzia wspomagające naukę gry na gitarze. Z tego też powodu powstaje problem z miejscem na urządzeniu, na którym trzeba zainstalować kolejne aplikacje.

Wiele innych aplikacji nie oferuje strojenia poprzez mikrofon. Przykładem może być aplikacja „Guitar Tuner” od „appsmz”. Działa ona na zasadzie odtwarzania dźwięku po kliknięciu w daną strunę. Dźwięk można zapętlić poprzez kliknięcie w przycisk w górnej części ekranu. Dla początkujących jest to duże utrudnienie, ponieważ nie są w stanie stwierdzić czy dźwięk odtwarzany z aplikacji jest taki sam jak dźwięk uzyskany po szarpnięciu za strunę gitary, którą trzyma w ręku. Ten typ aplikacji do strojenia gitary jest raczej dla zaawansowanych gitarzystów, jednak dla młodych w ogóle się nie sprawdza.

Przeglądając sklep Google Play pod kątem aplikacji wspomagających gry na gitarze można odebrać wrażenie, że nie ma takiej aplikacji, która zachęcałaby do kontynuacji nauki gry na gitarze. Są aplikacje rozbudowanie i kuszące funkcjami nauki gry akordów, są też takie, które oferują tylko dane funkcjonalności rozwijając je do potężnych narzędzi. Jednak żaden z twórców tych aplikacji nie zastał się nad przyciągnięciem użytkownika nie tyle do aplikacji, ale do samego grania na instrumencie co sprawiłoby, że ich aplikacje byłby uruchamiane częściej ze znacznie większym zapałem. Użytkownicy mając takie narzędzie, które motywowało by ich do stawiania nowych kroków przybliżając do celu, staliby się lojalnymi odbiorcami, którzy pomogli by w rozwoju samej aplikacji. Tacy użytkownicy mogliby podpowiedzieć kierunek w jakim dana aplikacja mogłaby się rozwinąć.

### **3. Aspekty technologiczne i metodyka realizacji projektu**

#### **3.1. Wykorzystane technologie**

##### **3.1.1. Android Studio**

Android Studio to najpopularniejsze IDE (ang. Integrated Development Environment – zintegrowane środowisko programistyczne) wykorzystywane w procesie tworzenia aplikacji mobilnych w technologii Android. Narzędzie to w procesie tworzenia aktywności oraz fragmentów proponuje użytkownikowi wiele szablonów. Decydując się na stworzenie aktywności bądź fragmentu z dostępnych szablonów generuje się kod, odpowiadający za poprawne działanie stworzonego elementu w Javie lub Kotlinie (w zależności jaki język został wybrany podczas tworzenia projektu) oraz powiązane pliki XML [1].

Do tworzenia układów elementów wykorzystuje się język XML, jednak Android Studio oferuje tworzenie go poprzez graficzny interfejs. Dostarcza on pogrupowany spis dostępnych komponentów, listę możliwych ustawień danego komponentu oraz aktualny podgląd okna w dwóch trybach. Pierwszy z nich to widok przedstawiający wygląd na telefonie, a drugi to ułożenie wszystkich elementów wraz z tymi niewidocznymi dla użytkowników na ekranie.

Jako, że Android Studio oparty jest na IDE od JetBrains posiada takie same funkcjonalności jak inne IDE od wydawcy. Posiada on takie rozszerzenie jak zarządzanie repozytorium zdalnym np. GitHub, przez które łatwo można utworzyć repozytorium, wysyłać zmiany na zewnętrzny serwer, jak również można wycofać zmiany [1].

Android Studio dostarcza też możliwość otwarcia aplikacji na emulatorze. Aby to zrobić najpierw należy utworzyć urządzenie o wybranych parametrach oraz wybrać odpowiedni system operacyjny. Następnie można uruchomić budowanie i uruchomienie aplikacji poprzez naciśnięcie przycisku „Run”, zaraz koło listy rozwijanej z dostępными urządzeniami.

Kolejnym rozszerzeniem wartym uwagi jest asystent Firebase'a (platforma oferująca szereg funkcji serwerowych). Z jego pomocą można utworzyć projekt na platformie Firebase, nawiązać połączenie z istniejącym projektem, dodać odpowiednie zależności do pliku gradle i otrzymać przykładowy kod na łączenie się z funkcjonalnościami z platformy Firebase'a.

### **3.1.2. Java**

Wysokopoziomowy język programowania utworzony przez Sun Microsystems w 1995 roku. Język ten wykorzystywany jest w aplikacjach desktopowych, aplikacjach mobilnych, aplikacjach webowych, czy nawet dla systemów wbudowanych takich jak karty SIM, telefony VOIP czy odtwarzacze dysków Blu-ray. Często jest wykorzystywany w systemach bankowych, co tylko potwierdza bezpieczeństwo jakie daje ten język programowania. Był usadowiony na czołowych miejscach w ostatnich latach w rankingu najpopularniejszych języków programowania.

Java polepszyła swoją wydajność poprzez zastosowanie kompilatora JIT (Just in time), który potrafi zoptymalizować najczęściej używaną część kodu. Pomiędzy mechanizmami, które umożliwiają takie działania jest dobrze znany „Garbage Collection”. Działa tak, aby zwolnić pamięć urządzenia z niepotrzebnych obiektów. Niepotrzebne obiekty uważa za takie, do których nie ma referencji. Te działanie wykonuje się automatycznie, niezależnie od użytkownika aplikacji czy nawet programisty [2].

### **3.1.3. Gradle**

Jest to narzędzie służące do zautomatyzowania procesu budowania projektu. Za pomocą jednej linii pobierane są odpowiednie zależności oraz w szybki sposób uruchamiane są testy aplikacji. Gradle jest najczęściej wybieranym narzędziem do zautomatyzowanego budowania aplikacji na systemy Android. Inne narzędzia podobne do Gradle, to również popularny Maven, Ant oraz Make [3,4].

### **3.1.4. ADB**

Ten interfejs pozwala na wiele operacji na urządzeniu mobilnym. Jednak najważniejszą funkcjonalnością, do której najczęściej wykorzystuje się ADB, to możliwość instalowania aplikacji bezpośrednio na urządzenie mobilne. Wystarczy włączyć na urządzeniu tryb debugowania, zainstalować odpowiedni sterownik do telefonu (zazwyczaj znajduje się na stronie producenta) i uruchomić w oknie polecień komendę adb install <ścieżka do pliku apk> [5]. Jednak Android Studio pozwala na uruchomienie polecenia poprzez jeden przycisk „Run” zaraz koło listy rozwijanej z dostępnymi urządzeniami. Interfejs ten pozwolił na szybkie zainstalowanie aplikacji na urządzeniach użytkowników, którzy testowali aplikacje. Pomógł też w sprawdzeniu poprawności dźwięku, ponieważ na symulatorze telefonu komórkowego na komputerze, aplikacja nie była w stanie działać poprawnie. Dźwięk, który symulator odbierał był zestawem cyfrowych sygnałów, w większości przypadków o jednej częstotliwości, co uniemożliwiało przetestowanie

działania stroika. Dlatego też należało aplikacje zainstalować na urządzeniu fizycznym i sprawdzić działanie aplikacji.

### **3.1.5. Firebase**

Jest to platforma wspierana przez firmę Google LLC posiadającą wiele funkcjonalności, które wspomagają pracę nad aplikacjami mobilnymi oraz webowymi [6]. Łatwość zarządzania danymi przez intuicyjny graficzny interfejs sprawia, że platformą interesuje się coraz większe grono programistów.

Firebase jest klasyfikowany jako BaaS (z ang. Backend as a Service). Dostarcza on przeróżne narzędzia wspomagające tworzenie aplikacji. Dzięki temu programista nie musi na nowo tworzyć systemu logowania i rejestracji czy wysyłki wiadomości email. Platforma ta, udostępniając takie przydatne funkcje jak „Cloud Function”, „Authentication”, „Hosting” czy nawet „Cloud Messaging”, przyspiesza tworzenie oprogramowania. Jest to zestaw podstawowych funkcjonalności we współczesnych aplikacjach. Widoczny rozwój platformy znaczaco wpływa na rosnącą jej popularność.

W aplikacji użyte zostały takie funkcje jak: „Authentication” (pozwala na łatwe zarządzanie zarejestrowanymi użytkownikami), „Cloud Firestoire” (jest to baza danych, która pozwala na ładowanie danych w czasie rzeczywistym, jest to baza NoSQL) oraz „Firebase Storage” (jest to miejsce, w którym można przechowywać pliki, które później można pobierać w aplikacji).

### **3.1.6. GIMP**

Pierwsza wersja tego programu została wydana w listopadzie 1995 r. przez Petera Mattisa i nosiła nazwę „The GIMP”. Program został oparty na bibliotece Motif jednak autor zaznaczał, że późniejsze wersje nie będą oparte na tej bibliotece. Tak też się stało. W wydaniu rozwojowym rozpoczęto korzystanie z autorskich bibliotek. Warto też wspomnieć, że GIMP pozwala na zaimplementowanie własnych funkcjonalności poprzez skrypty w językach Scheme, Perl, Ruby i Python.

GIMP to darmowa aplikacja służąca do obróbki plików graficznych. Licencja tej aplikacji pozwala na korzystanie z programu przez użytkownika prywatnego jak i do celu komercyjnych co jest największą zaletą tego programu. To środowisko rozwijane przez wielu programistów jest na tyle rozbudowane, że stało się bardzo poważną konkurencją dla takiego środowiska jak Adobe Photoshop. Kolejną zaletą tego środowiska jest przyjemny dla użytkownika interfejs co było w założeniu tworzenia projektu. GIMP posiada podstawowe funkcje jak malowanie, zaznaczanie, mazanie/usuwanie, ale też te bardziej

złożone jak warstwy, złożone pędzle do malowania, skalowanie, usuwanie szumów z obrazów czy operacje na plikach. Za pomocą jednego kliknięcia jesteśmy w stanie wyeksportować zdjęcie do wszystkich podstawowych rozszerzeń graficznych [7,8].

### 3.2. Dodatkowe biblioteki

#### 3.2.1. Biblioteka TarsosDSP

Jest to biblioteka otwartoźródłowa do przetwarzania dźwięku. Cała biblioteka, w większości przypadków, opiera się na czystej Javie. W repozytorium tej biblioteki można uzyskać link do przykładowych programów, stworzonych na podstawie tej biblioteki. Można zobaczyć takie aplikacje jak: detektor głośności dźwięku, spektrogram oraz gra polegająca na zaśpiewaniu dźwięku najlepiej jak to możliwe. Aby dołączyć tą bibliotekę należy ze strony projektu na platformie GitHub pobrać jej najnowszą wersję i zapisać ją do folderu libs w module app. Następnie do pliku build.gradle w module app dopisać tą linię zawartą na rysunku 1.

```
implementation fileTree(dir: 'libs', include: ['*.jar'])
```

Rysunek 1. Dodanie biblioteki z plików aplikacji. Źródło: opracowanie własne

Ważne, aby dopisać tą linię w obiekcie dependencies, w którym zapisane są wszystkie zależności w projekcie [9].

Biblioteka ta udostępnia gotowe algorytmy. Jednym z nich jest algorytm szybkiej transformaty Fouriera (ang. Fast Fourier Transform – FFT). Jest ona implementacją innego algorytmu – dyskretniej transformaty Fouriera (ang. Discrete Fourier Transform - DFT).

Do badania dźwięku stosuje się analizę widmową (częstotliwościową). Do tego procesu stosuje się metodę digitalizacji (cyfryzacji) sygnałów i analizę cyfrową. Do prowadzenia takiej analizy widmowej, która powala na przekształcenie funkcji zależnej od czasu na funkcje zależną od częstotliwości, wykorzystywana jest transformata Fouriera (widmo):

$$X(f) = \int_{-\infty}^{\infty} x(t)e^{-i2\pi ft} dt \quad (1)$$

f- częstotliwość,

x(t) – wartość amplitudy zależnej od czasu.

Aby uzyskać odwrotny wynik, czyli funkcję zależną od częstotliwości przekształcić na zależną od czasu, stosuje się odwrotną transformatę Fouriera:

$$x(t) = \int_{-\infty}^{\infty} X(f) e^{i2\pi f t} df \quad (2)$$

Analiza cyfrowa jest przeprowadzana poprzez wykorzystanie DFT:

$$X(k) = \sum_{n=0}^{N-1} x(n) \omega_N^{-kn}, 0 \leq k \leq N-1 \quad (3)$$

$$\omega_N = e^{i\frac{2\pi n}{N}} \quad (4)$$

k – numer próbki sygnału w dziedzinie częstotliwości,

n – numer próbki sygnału w dziedzinie czasu,

x(n) – amplituda próbki sygnału w dziedzinie czasu,

N – liczba próbek.

Odwrotną dyskretną transformatą Fouriera można opisać wzorem:

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) \omega_N^{kn}, 0 \leq n \leq N-1 \quad (5)$$

Tylko pierwsza połowa próbek widmowych niesie za sobą istotnie informacje, wynika to z tego, że funkcja DFT dla sygnałów rzeczywistych (dokładnie moduł zespolonej funkcji DFT) jest symetryczna. Częstotliwość Nyquista nazywa się częstotliwością odpowiadającą za punkt odcięcia i dla sygnału o częstotliwości 44,1Hz wynosi 22,05kHz.

Wadą stosowania DFT jest złożoność obliczeniowa, wzrasta ona kwadratowo ( $N^2$ ) dla ilości próbek. Dlatego też szukano innych sposobów na uzyskanie tych samych wyników za pomocą mniejszej ilości wykonywanych operacji. W 1965 r. został opublikowany artykuł, autorstwa Cooley'a oraz Tuckey'a, przedstawiający znacznie wydajniejszy algorytm implementujący DFT. Algorytm ten nazywany jest algorytmem Cooleya-Tuckeya lub szybką transformatą Fouriera (ang. Fast Fourier Transform – FFT).

Algorytm ten bazując na metodzie dziel i zwycięzaj obniża złożoność obliczeniową (w zależności od implementacji algorytmu) nawet do  $N \log_2 N$ . Polega on na rozbięciu sumy na dwie podsumy – sumę parzystych i nieparzystych elementów. Najefektywniejszą wersją tego algorytmu jest tzw. FFT o podstawie 2, w tym przypadku podawane są takie ilości próbek, aby te były wielokrotnością liczby 2, dzięki czemu możliwe jest zastosowanie schematu motylkowego.

Już w pierwszym etapie rozbicia wykonywane są  $2N^2+N$  obliczeń zamiast  $4N^2$ . Zagnębiając się co raz bardziej, stosowane są coraz mniejsze ilości obliczeń, a w ostatniej fazie otrzymane zostaje 2-punktowe DFT. Zrealizowany zostaje on przez, wspomniany

już wcześniej, schemat motylkowy. Wykonuje on tylko jedno mnożenie, co z kolei pozwala na osiągnięcie złożoności obliczeniowej równą  $N \log_2 N$  [10-13].

### 3.2.2. Biblioteka Picasso

Jest to biblioteka służąca do dołączania obrazów do aplikacji. Dołączane zdjęcia mogą znajdować się w folderze zasobów aplikacji lub na zewnętrznym serwerze. Dołączanie zdjęć do aplikacji staje się znacznie łatwiejsze niż przez standardowy plik XML. W przypadku, gdy chcemy dołączyć zdjęcie bezpośrednio do znacznika XML, plik z obrazkiem musi znajdować się w folderze zasobów aplikacji lub Androida. Używając biblioteki Picasso dołączanie i ustawianie wartości dla zdjęcia odbywa się w jednej linii kodu [14]. Standardowym kodem jest kod umieszczony na rysunku 2.

```
Picasso.with(context).load(url).into(target);
```

Rysunek 2. Dodanie zdjęcia w danym elemencie za pomocą biblioteki Picasso. Źródło: opracowanie własne

Aby dołączyć tą bibliotekę wystarczy do pliku build.gradle w module app dopisać tą linię, gdzie liczbami na końcu oznaczona jest aktualna, podczas tworzenia aplikacji, wersja biblioteki (rysunek 3).

```
implementation 'com.squareup.picasso:picasso:2.5.2'
```

Rysunek 3. Dodanie biblioteki za pomocą zależności. Źródło: opracowanie własne

Ważne, aby ta linia znalazła się w obiekcie dependencies, w którym zapisane są wszystkie zależności w projekcie [15,16].

## 3.3.Specyfikacja wymagań funkcjonalnych aplikacji

W niektórych przypadkach, użytkownik powinien mieć możliwość korzystania z aplikacji bez użycia rąk. Jest to ważne, ponieważ strojąc gitarę użytkownik nie jest w stanie trzymać telefonu w dłoni i wykonywać jakikolwiek działań na telefonie. Drugą kwestią jest przejrzysty interfejs. Jest to motywowane tymi samymi pobudkami. Strojąc gitarę użytkownik aplikacji położy telefon komórkowy w dogodnej dla niego odległości. Trzeba umożliwić użytkownikowi na swobodne działanie nie ograniczając go poprzez przywiązanie do telefonu.

Aplikacja powinna oferować stroiki dla kilku strojeń. Pierwszy z nich powinien być standardowy, kolejne muszą być najpopularniejsze w swoim gatunku muzycznym. Po przeprowadzeniu rozpoznania oprócz strojenia standardowego (Standard – EADGHe) będzie dostępny strój E-Flat (EbG#C#F#BbEb – dźwięk obniżony o półtonu) oraz Drop-

D (DGCFAD – dźwięk obniżony o cały ton) [17]. Stroik powinien mieć dwa sposoby działania. Pierwszy z nich to strojenie jednej struny. Powinno być to możliwe po wyborze struny. Poprawność dźwięku dla tej struny powinna być mierzona cały czas dopóki nie zmieni się struny lub nie wejdzie się w drugi sposób działania stroika. Drugie działanie powinno umożliwiać nastrojenie wszystkich strun bez konieczności przełączania się między strunami. W tym trybie sprawdzanie poprawności dźwięku powinno odbywać się dla wszystkich strun wskazując poprawność dla poszczególnych. Jeśli dźwięk, który użytkownik zagra jest bardzo zbliżony do którejś ze strun, aplikacja powinna zaznaczyć tą właśnie strunę i wskazać o ile trzeba zmienić dźwięk (manewrując przy kluczach gitary) aby uzyskać odpowiedni strój dla danej struny.

Kolejną funkcją, która powinna oferować aplikacja jest metronom. Metronom jako proste narzędzie do mierzenia tempa, jest wręcz niezbędny do nauki gry na gitarze. Metronom powinien oferować możliwość zmiany tempa nawet w trakcie swojego działania. Dużym ułatwieniem będzie zastosowanie dwóch różnych dźwięków, z których jeden oznacza rozpoczęcie taktu oraz dostarczenie listy, w której zawarte będą przykładowe metrum. Wybierając metrum 4/4 w cyklu będą cztery dźwięki. Pierwszy z nich będzie odznaczał się od pozostałych. Analogicznie działać się będzie tak dla metrum np. 5/8. Będzie pięć dźwięków w jednym cyklu i tylko jeden z nich, oznaczający początek nowego cyklu, będzie odznaczał się innym dźwiękiem. Metronom ten powinien mieć możliwość zmiany tempa wybijanych dźwięków poprzez wybór pomiędzy dostępymi tempami (np. Largo, Allegro, Presto) oraz pozwolić użytkownikowi łatwo określić własne tempo poprzez suwak lub przyciski +/-.

Funkcją skłaniającą do dalszej nauki będzie ranking zalogowanych użytkowników aplikacji. Będzie ona wyświetlać ranking w zależności od grupy akordów lub całościowy. Będzie on wyświetlał dane użytkowników takie jak nazwa użytkownika, ikona użytkownika i jego punkty zebrane w danej kategorii. Każdy użytkownik zalogowany będzie miał możliwość przeglądnięcia poszczególnych kategorii rankingów.

Użytkownik aplikacji powinien mieć możliwość zarządzania własnym profilem. Aplikacja powinna umożliwić zalogowanym użytkownikom zamianę takich danych jak nazwa oraz ikona. Nazwa będzie zmieniana poprzez udostępnienie pola tekstowego z wpisaną aktualną nazwą, a ikona będzie możliwa do zmiany poprzez kliknięcie w aktualną ikonę i wybór pomiędzy dostępnymi ikonami.

Początkujący gitarzysta powinien mieć możliwość nauki akordów. Taki tryb zostanie dostarczony przez aplikację. Po wyborze odpowiedniej grupy, następnie danego

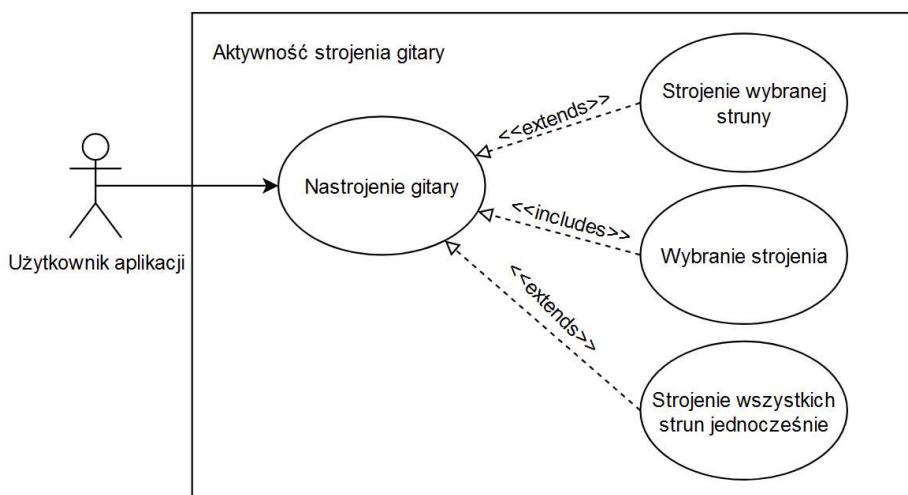
akordu, użytkownikowi wyświetli się schemat akordu oraz 6 pasków symbolizujących struny. Po poprawnym zagraniu dźwięku odpowiednia struna zmieni kolor na zielony. Jeśli wszystkie struny zmienią kolor na zielony użytkownik zostanie poinformowany o poprawnym zagraniu akordu.

Użytkownik chcący brać udział w rankingu będzie spltany przy wyborze akordu, w który tryb chce wejść. Wybierając ranking, czas zostanie uruchomiony wraz z każdą próbą, a schemat zostanie ukryty. Klikając w przycisk użytkownik może wyświetlić schemat, jednak punkty możliwe do zdobyć zostaną zmniejszone. W zależności od czasu jaki użytkownik będzie potrzebował na zagranie danego akordu punkty zostaną odpowiednio przydzielone. To samo dotyczy się ilości podejścia do jednego akordu. Im więcej użytkownik chce podejść do jednego akordu tym punkty dostępne do zdobycia będą mniejsze. Będzie to miało na celu wyeliminowanie problemu z nieuczciwymi użytkownikami.

## 4. Przedstawienie systemu

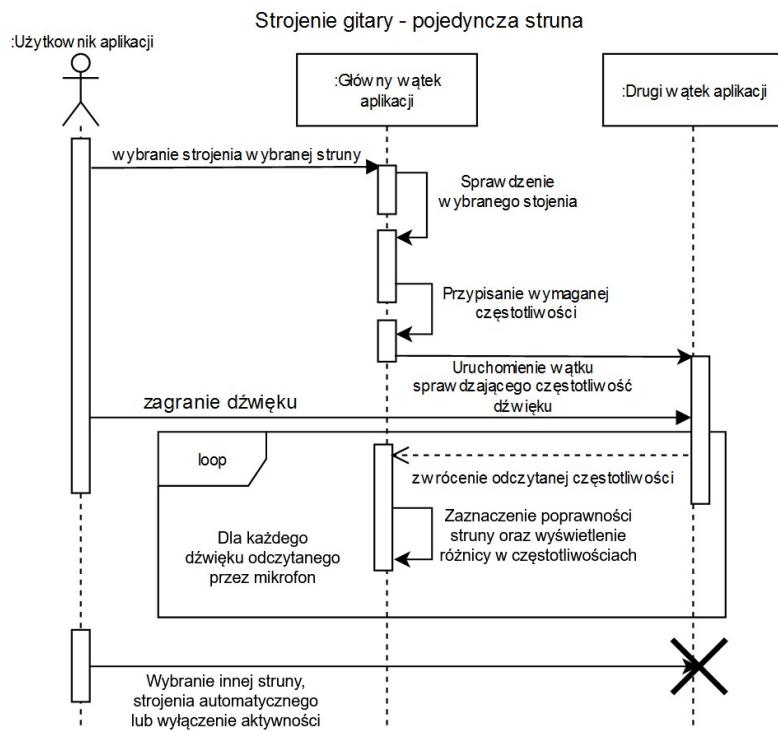
### 4.1. Strojenie gitary

Zgodnie z diagramem na rysunku 4, użytkownik uruchamiając okno aplikacji odpowiedzialne za nastrojenie gitary może wybrać rodzaj strojenia. Aby nastroić gitarę użytkownik ma dwie możliwości. Pierwszą z nich jest nastrojenie gitary w sposób taki, że wybierze daną strunę, sprawdzi jej częstotliwość i gdy częstotliwość będzie zadowalająca zmieni strojoną strunę, a w przypadku ostatniej zakończy proces strojenia. Drugim wyjściem jest włączenie trybu auto, gdzie użytkownik może szarpać struny dowolnie i gdy częstotliwość na którejś z nich będzie zadowalająca to przejdzie do kolejnej bez konieczności operacji na urządzeniu lub wyłączy dane okno.

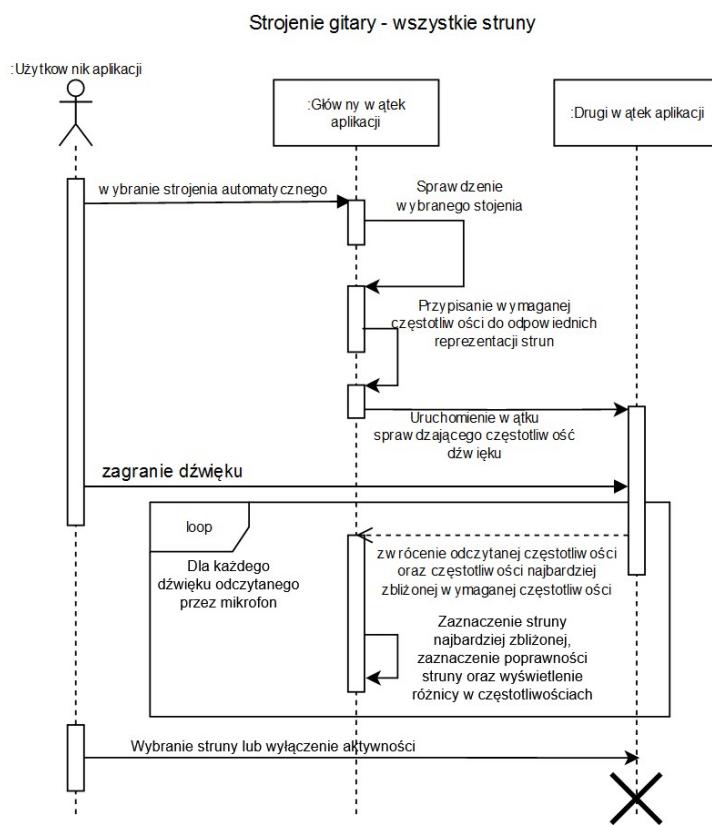


Rysunek 4. Diagram przypadków użycia dla funkcjonalności strojenia gitary. Źródło: opracowanie własne

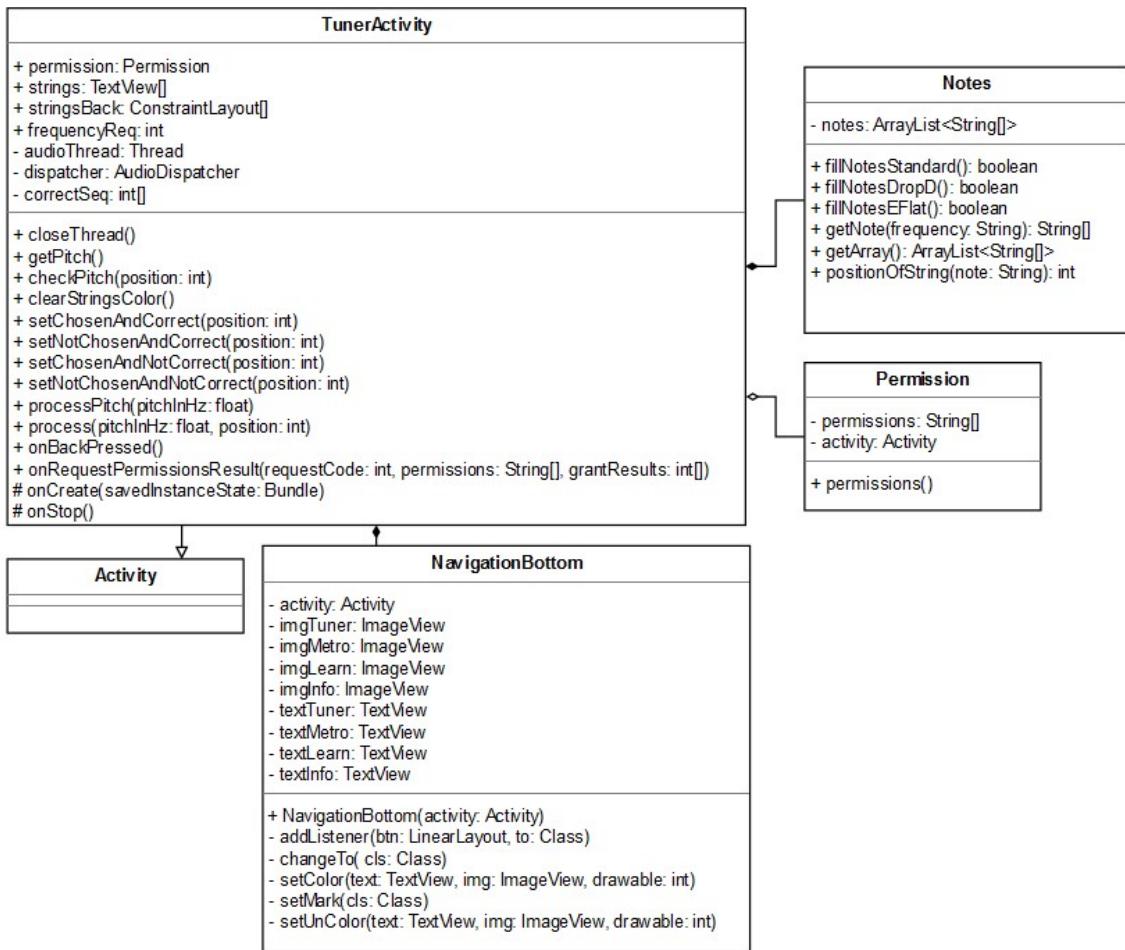
Działanie strojka dla strojenia pojedynczej struny i strojenia automatycznego różni się w działaniach wykonywanych przez aplikację (przedstawione na rysunku 5 oraz rysunku 6). Najważniejszą zmianą jest sposób przypisania wymaganej częstotliwości. W pierwszym przypadku wymagana częstotliwość jest zapisywana w zmiennej, podczas gdy w drugim podejściu wymagane częstotliwości są przypisywane do zmiennych, które reprezentują dane struny. Umożliwia to później porównanie częstotliwości dźwięku uderzonej struny z częstotliwościami jakie przyjmuje dana struna oraz zaznaczenie struny, która ma najbardziej zbliżoną częstotliwość.



Rysunek 5. Diagram sekwencji – strojenie pojedynczej struny. Źródło: opracowanie własne



Rysunek 6. Diagram sekwencji – strojenie wszystkich strun jednocześnie. Źródło: opracowanie własne

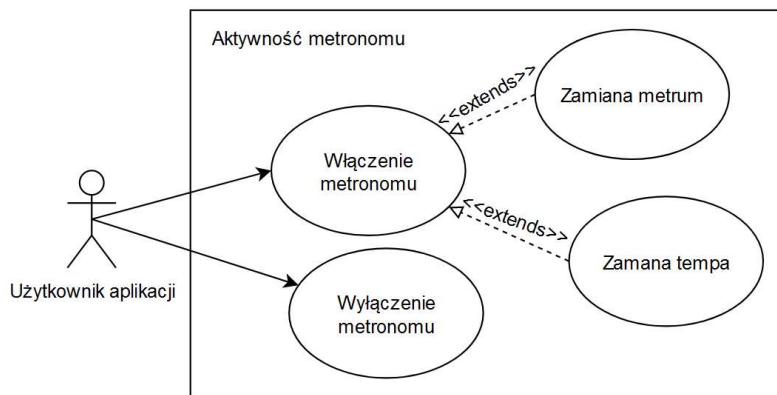


Rysunek 7. Diagram klas – funkcjonalność strojenia gitary. Źródło: opracowanie własne

Klasy, które oferują dane działania są zawarte na rysunku 7. Klasa oferująca wygląd okna i jego działanie jest zawarta w klasie TunerActivity, który dziedzicząc po klasie Activity pozwala na użycie metod działających w danym stanie (np. metoda onCreate jest wykorzystywana podczas tworzenia wyglądu aplikacji łącząc kod języka Javy z plikiem XML tworzącym rozmieszczenie elementów). Do tej klasy dołączane są inne klasy oferujące działanie w poszczególnych aspektach. Z klasy Notes pobierana jest lista strun połączona z wymaganą częstotliwością, w klasie Permission oferowane są działania odpowiedzialne za poprawne działanie aplikacji ze względu na pozwolenia zatwierdzane przez użytkownika. Klasa Premission odpowiada także za sprawdzenie czy dane pozwolenie jest włączone, w innym przypadku prosi o pozwolenie, jeśli zostanie odmówione, blokuje działanie aplikacji. W klasie NavigationBottom znajduje się szereg działań sprawiających, że dolna nawigacja w aplikacji działa poprawnie.

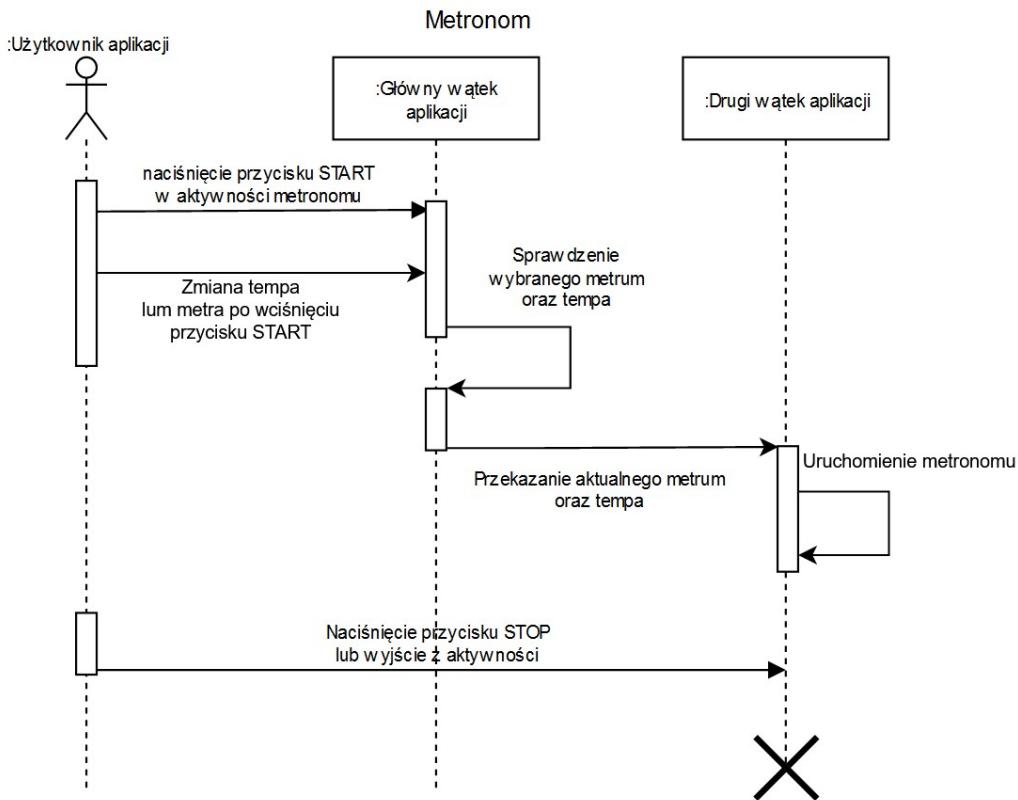
## 4.2.Metronom

Zgodnie z rysunkiem 8, użytkownik ma możliwość włączyć oraz wyłączyć działanie metronomu. Podczas działania metronomu użytkownik ma możliwość zmiany tempa (w zakresie od 40 do 208) oraz zmiany metrum spośród podanych.

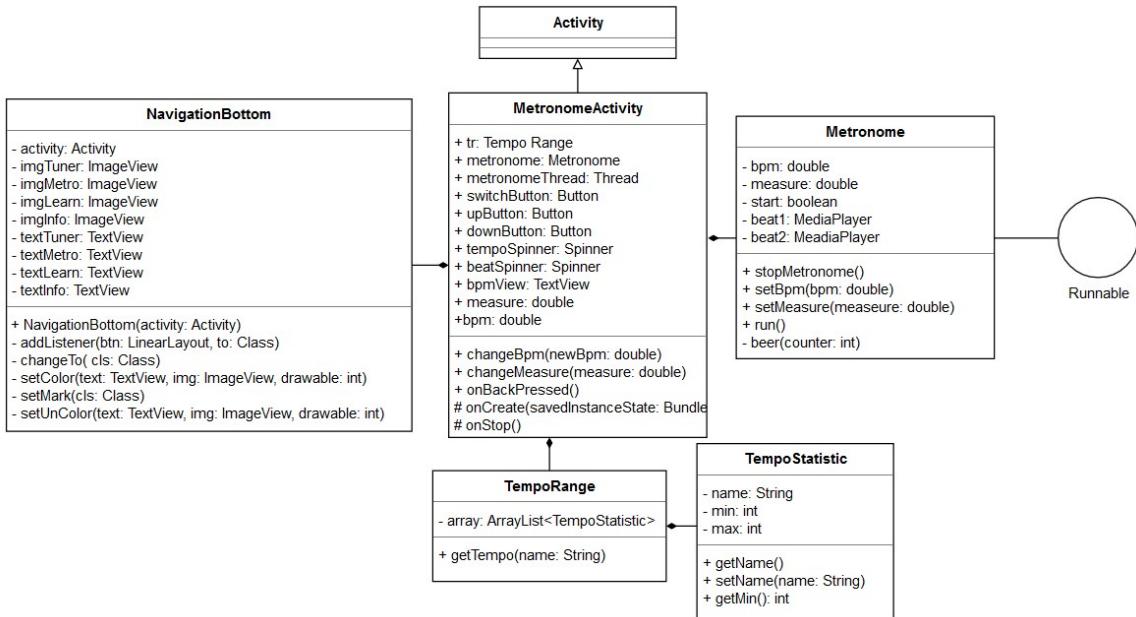


Rysunek 8. Diagram przypadków użycia dla funkcjonalności metronomu. Źródło: opracowanie własne

Główne działanie aplikacji nie może być zaburzone lub zablokowane, dlatego też zgodnie z rysunkiem 9 metronom powinien zostać uruchomiony w nowym wątku. Dzięki temu możliwa jest zmiana tempa i metrum w czasie działania wątku z metronomem. Dlatego też klasa Metronome, przedstawiona na rysunku 10, implementuje interfejs Runnable, dzięki czemu w klasie MetronomeActivity zostaje uruchomiony wątek, który działa w sposób opisany w klasie Metronome. Klasami pomocniczymi w tym zestawieniu są TempoRange oraz TempoStatistic. W klasie TempoStatistic określone są dane na temat danego wyróżnionego tempa, natomiast w TempoRange tworzona jest lista obiektów typu TempoStatistic, reprezentująca 8 podstawowych temp. Dzięki pracy tych dwóch klas użytkownik może wybrać wśród podanych temp, następnie wartość minimalna wybranego tempa zostaje ustalona w wątku metronomu na wartość bieżącego tempa. Do MetronomeActivity dołączana jest klasa NavigationBottom odpowiedzialna za dolną nawigację aplikacji.



Rysunek 9. Diagram sekwencji – działanie metronomu. Źródło: opracowanie własne

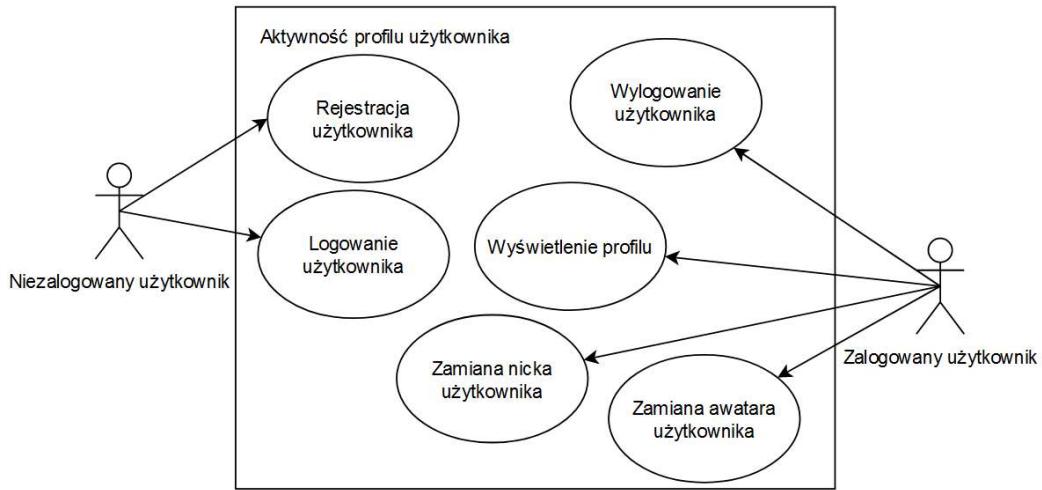


Rysunek 10. Diagram klas – funkcjonalność strojka. Źródło: opracowanie własne

#### 4.3. Profil użytkownika

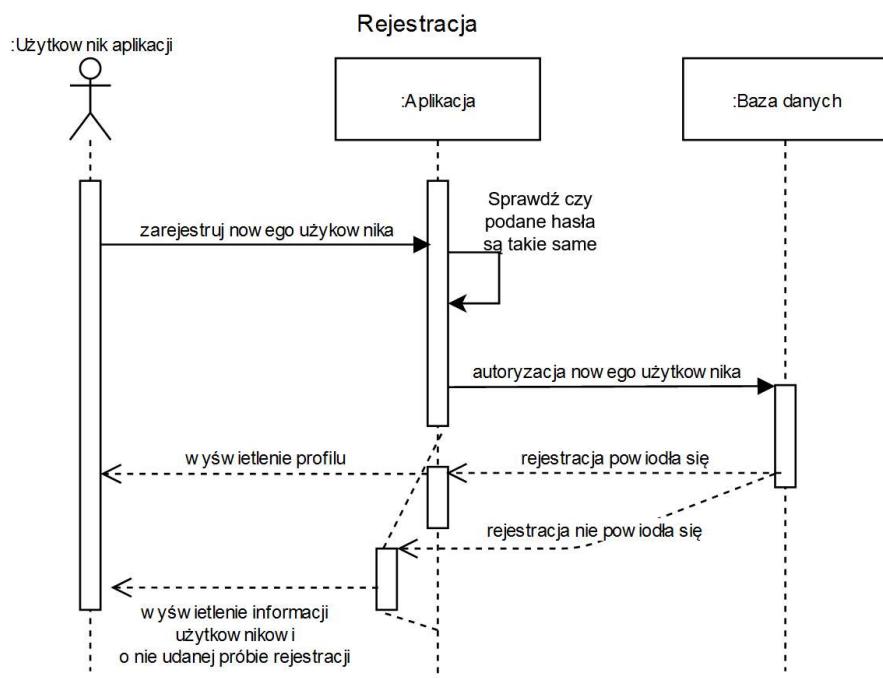
Wyróżniane są dwa rodzaje użytkowników (rysunek 11). Pierwszy z nich to użytkownik niezalogowany. Może on się zalogować, zarejestrować oraz korzystać z funkcji strojenia oraz metronomu. Drugi z nich oprócz działań, które będzie mógł wykonywać

użytkownik niezalogowany, będzie miał dostęp do zarządzania swoim profilem (wyświetlanie profilu, zmianę nazwę oraz awatara) oraz wylogowania się.

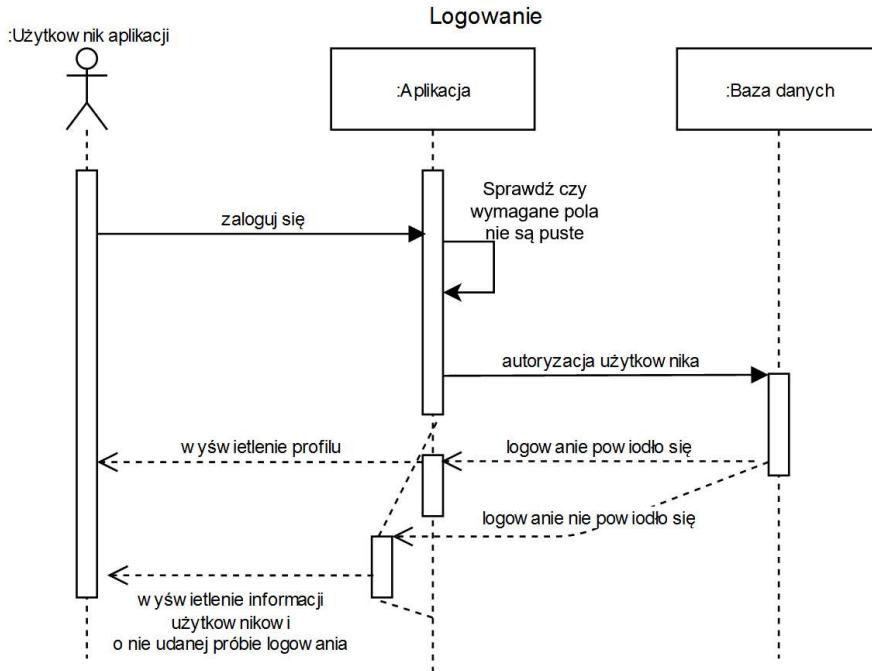


Rysunek 11. Diagram przypadków użycia – profil użytkownika. Źródło: opracowanie własne

Aby dodać nowego użytkownika (diagram 12) należy po wypełnieniu trzech pól (mail, hasło oraz potwierdzenia hasła) wysłać zapytanie do bazy czy podany mail już istnieje, jeśli tak, zostaje zwrocona informacja o niepowodzeniu autentykacji, w innym przypadku użytkownik zostaje przekierowany do okna profilu w aplikacji. Do logowania zostaje zastosowany ten sam schemat działania z tym, że hasło nie wymaga powtórzenia (diagram 13).

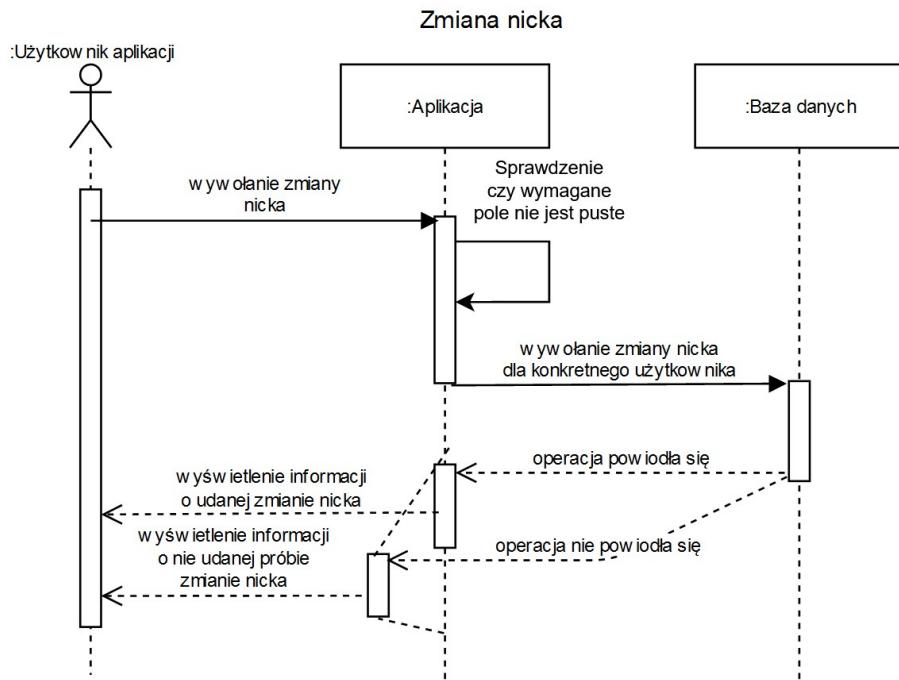


Rysunek 12. Diagram sekwencji – rejestracja. Źródło: opracowanie własne

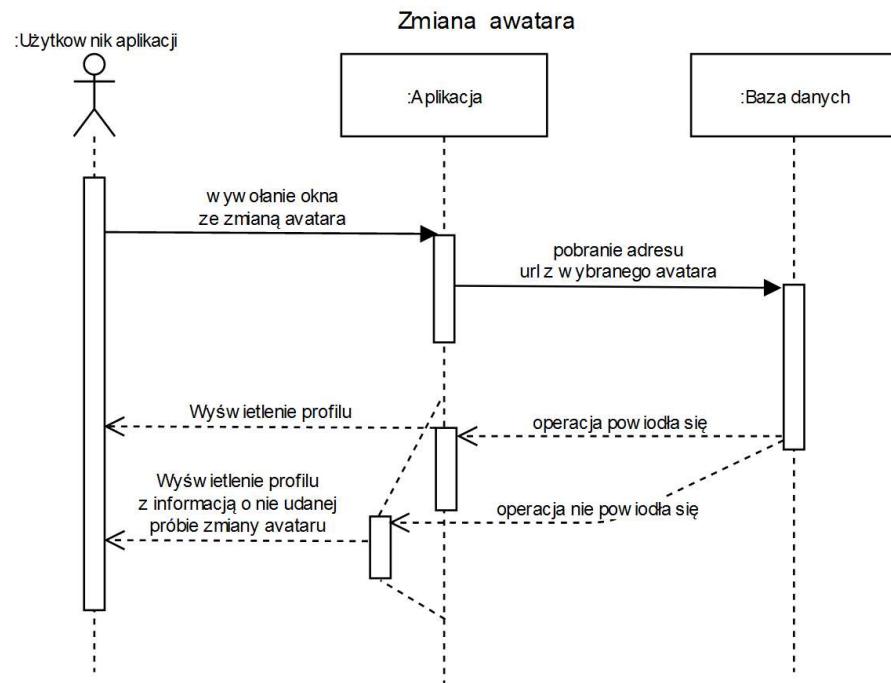


Rysunek 13. Diagram sekwencji – logowanie. Źródło: opracowanie własne

Użytkownik może dokonać dwóch zmian w swoim profilu. Pierwszym z nich jest dokonanie zmian w nazwie użytkownika (rysunek 14). Nazwa użytkownika jest wykorzystywana w wyświetleniu rankingu. Po wpisaniu nowej nazwy, użytkownik zatwierdza zmianę poprzez kliknięcie w przycisk, co wywołuje zapytanie do bazy. Jeśli zapytanie zostanie poprawnie wykonane, to na urządzeniu użytkownika wyświetli się informacja o poprawnym przebiegu zmian, w innym przypadku zostanie poinformowany o błędzie. Kolejną rzeczą możliwą do zmiany oraz wykorzystywaną do wyświetlenia rankingu jest awatar użytkownika (rysunek 15). Może on zmieniać swoją ikonę na jedną z dostępnych. Odbywa się to na podobnej zasadzie co zmiana nazwy użytkownika. Jednak użytkownik nie zatwierdza zmiany poprzez kliknięcie w przycisk, a w zdjęcie, które przypisuje w bazie danych adres URL danej ikony do danego użytkownika.



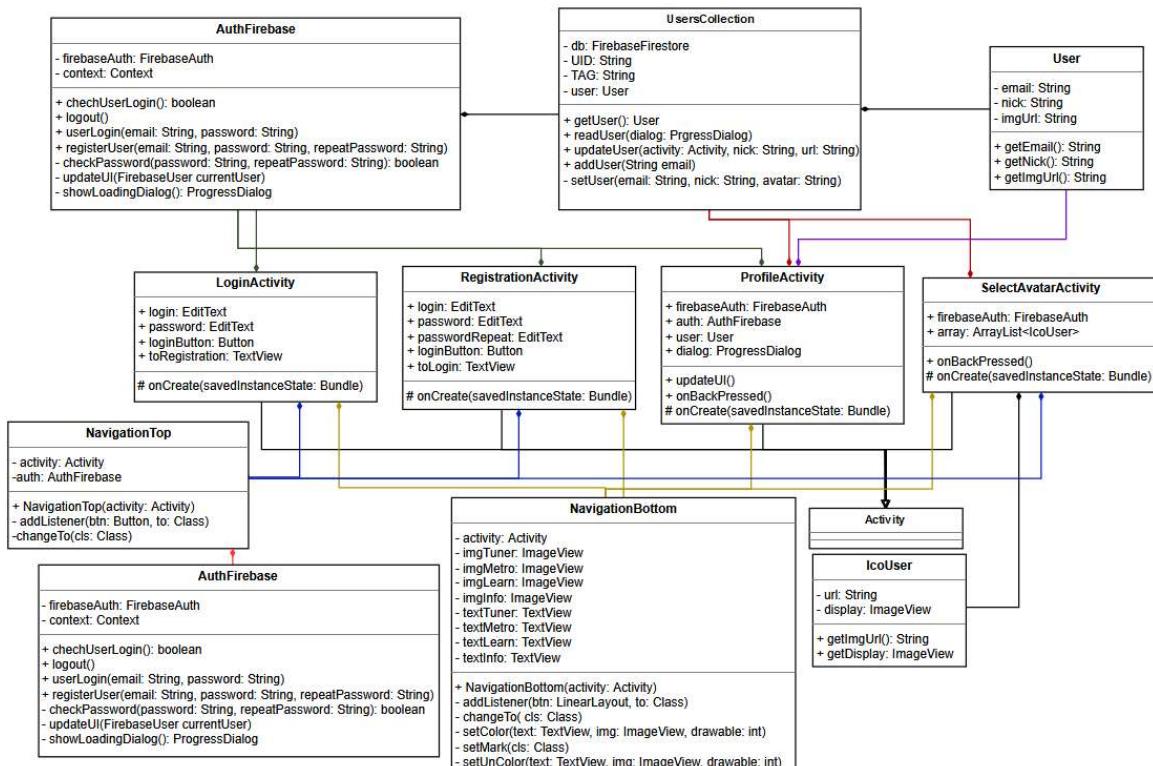
Rysunek 14. Diagram sekwencji – zmiana nazwy użytkownika. Źródło: opracowanie własne



Rysunek 15. Diagram sekwencji – zmiana ikony użytkownika. Źródło: opracowanie własne

Na rysunku 16 przedstawione są klasy biorące udział w prezentacji i działaniu opisanych sytuacji. Za wygląd okien aplikacji odpowiadają klasy LoginActivity (odpowiada za okno logowania), RegistrationActivity (odpowiada za okno rejestracji), ProfilActivity (odpowiada za wyświetlenie okna profilu użytkownika) oraz SelectAvatarActivity (wyświetla dostępne ikony użytkownika – do zmiany awatara użytkownika) wszystkie dzie-

dziczą po klasie Activity. Do każdej z podanych klas (oprócz klasy ProfileActivity) dołączane są klasy NavigationBottom (odpowiedzialna za dolną nawigację aplikacji) oraz NavigationTop (odpowiedzialna za górną nawigację aplikacji). Na tym diagramie widoczne są też klasy pomocnicze. Są to klasy User, IconUser oraz AuthFirebase. Pierwsza z nich odpowiada za stworzenie obiektu użytkownika, który posiada informacje o mailu, ikonie oraz nazwie użytkownika. IconUser odpowiada za prezentację dostępnych ikon, łącząc adres URL z miejscem wyświetlonym w oknie SelectAvatarActivity. Ostatnią z pomocniczych klas jest AuthFirebase. Jest on odpowiedzialny za komunikacje pomiędzy funkcją autoryzacji na platformie Firebase z aplikacją. Ostatnią klasą przedstawioną na diagramie jest UserCollection. Jest to klasa łącząca bazę danych z aplikacją. Dzięki niej można wyciągnąć dane o użytkowniku oraz zmieniać informacje w bazie danych o danym użytkowniku.

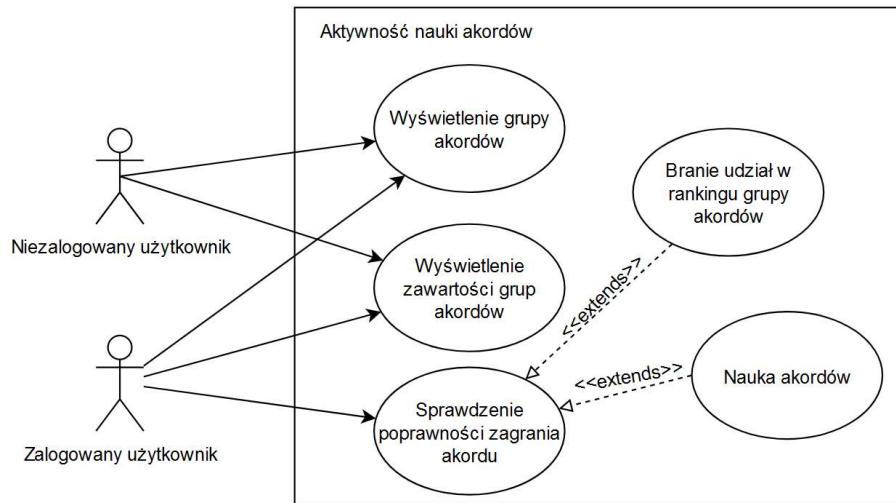


Rysunek 16. Diagram klas – funkcjonalność profilu. Źródło: opracowanie własne

#### 4.4.Nauka akordów

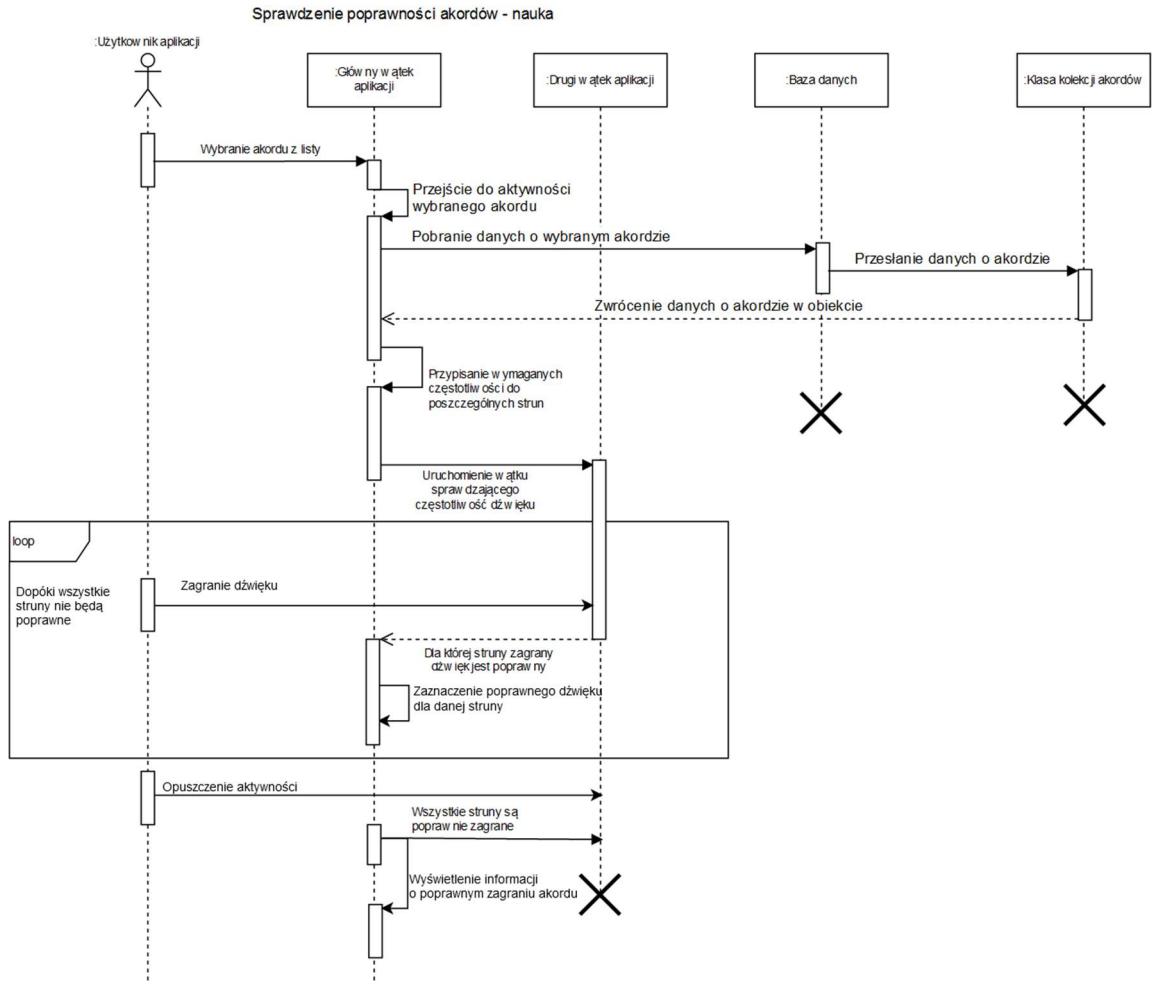
Do okien, które oferują naukę akordów dostęp będą mieli zalogowani, ale też niezalogowani użytkownicy. Jednak w przypadku niezalogowanego użytkownika dostęp do funkcji jest ograniczony. Może on tylko wyświetlić listę dostępnych akordów oraz listę z podziałem na grupy. Zalogowany użytkownik może sprawdzać poprawność granych

akordów w trybie nauki lub w trybie rankingu (rysunek 17). Tryb rankingu różni się od trybu nauki tym, że przydziela użytkownikowi punkty za dany akord.

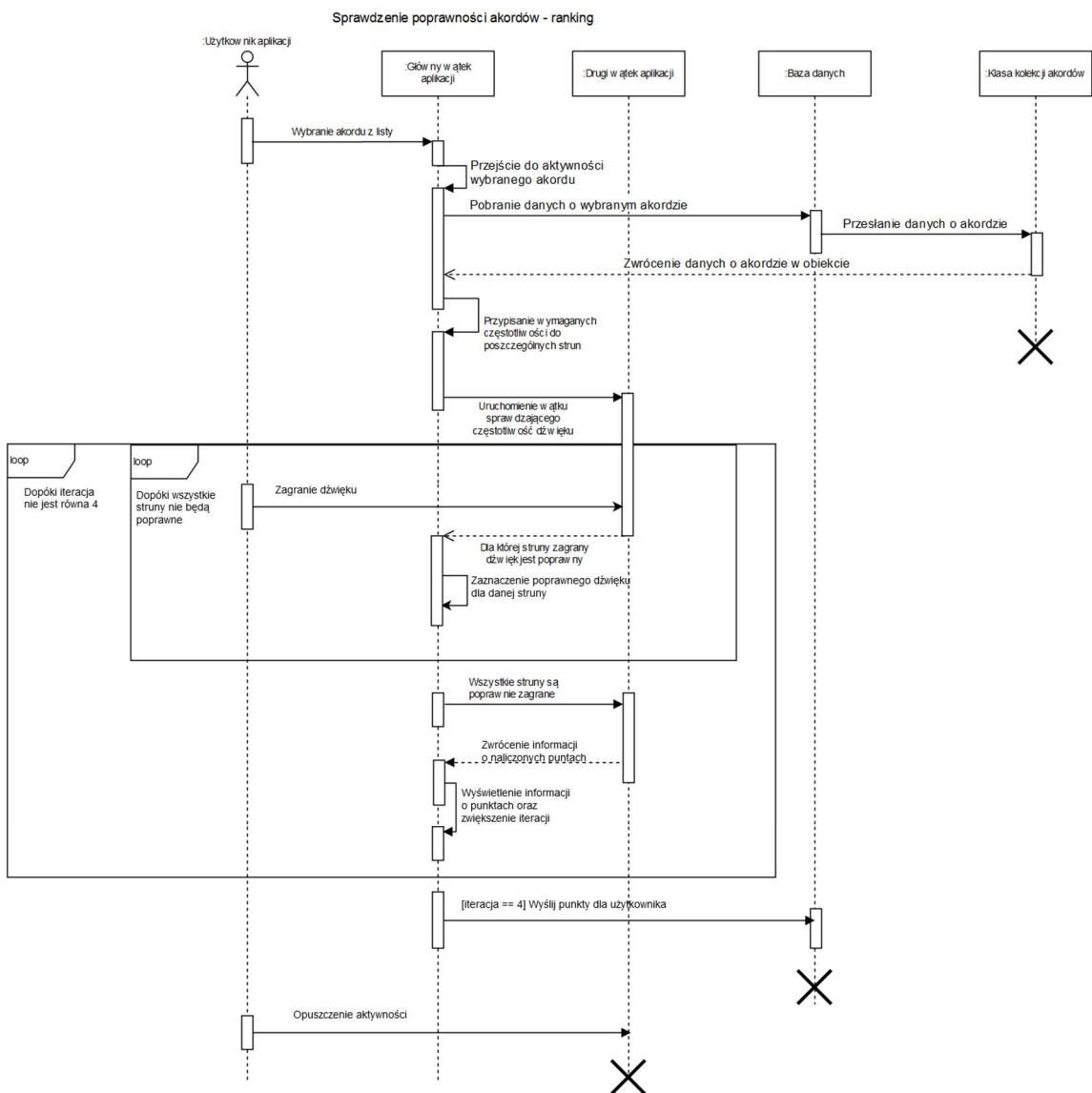


Rysunek 17. Diagram przypadków użycia – nauka akordów. Źródło: opracowanie własne

Przed rozpoczęciem głównego działania zostają pobrane dane z bazy o wybranym przez użytkownika akordzie (rysunek 18). W klasie łączącej bazę danych z aplikacją pobierane są dane i tworzony jest obiekt akordu z wymaganymi częstotliwościami na odpowiednich strunach oraz adres URL do schematu akordu. Następnie tworzony jest wątek sprawdzający poprawność granych dźwięków i zaznaczane są struny, których częstotliwość wymagana i zagrana są takie same lub różnica mieści się w granicy błędu. Gdy już wszystkie struny zostają oznaczone jako poprawne, to wyświetlana jest informacja o poprawnym zagraniu oraz wyłączany jest wątek sprawdzający poprawność akordu. W przypadku trybu rankingu (rysunek 19), gdy wszystkie struny zostają zaznaczone jako poprawne, to zostają naliczane punkty, zwiększana jest liczba prób oraz zostaje wyświetlona informacja o punktach zdobytych w próbie oraz w całej rundzie. Gdy liczba prób osiągnie 4, to zostaje zakończona runda, a dane o zdobytych punktach zostają przesłane do bazy danych.



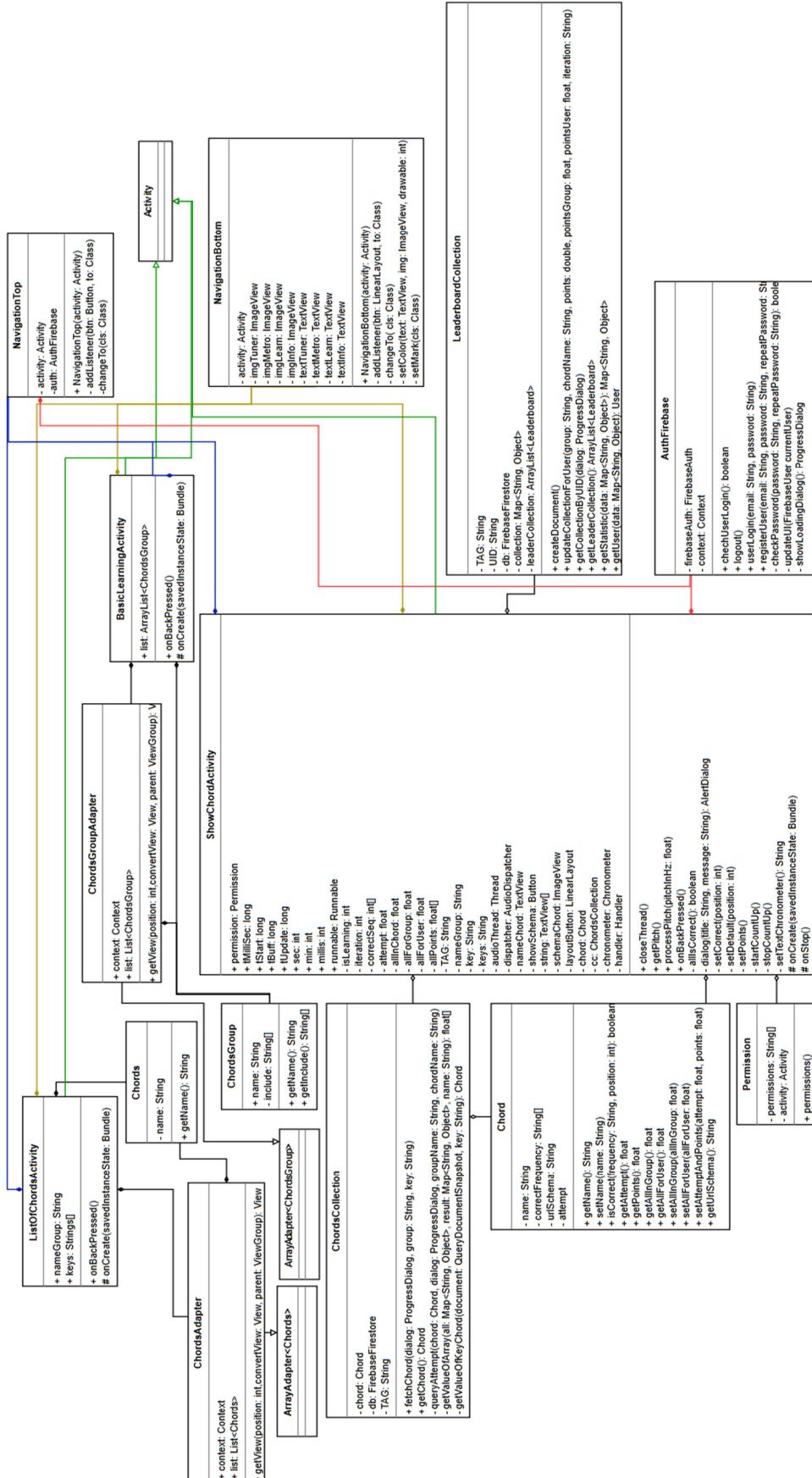
Rysunek 18. Diagram sekwencji – poprawność akordów w trybie nauki. Źródło: opracowanie własne



Rysunek 19. Diagram sekwencji – poprawność akordów w trybie rankingu. Źródło: opracowanie własne

Na rysunku 20 przedstawiono współprace klas realizujących działanie nauki akordów. Klasy odpowiadające za funkcje nauki akordów to BasicLearningActivity (lista grup akordów), ListOfChordsActivity (spis akordów w grupie) oraz ShowChordActivity (okno z danym akordem) wszystkie dziedziczą po klasie Activity i dołączane są klasy pomocnicze takie jak NavigationTop, NavigationBottom (nawigacje aplikacji). Do poprawnego wyświetlania listy zostały wykorzystane klasy adapterów ChordsAdapter (lista akordów) oraz ChordsGroupAdapter (lista grup akordów), dziedziczące odpowiednio z klas ArrayAdapter<Chords> oraz ArrayAdapter<ChordsGroup> oraz klasy pomocnicze Chords (zawiera nazwę akordu) i ChordsGroup (łączy nazwę grupy z listą akordów). Do okna ShowChordActivity dołączane są klasy pomocnicze Permission, Chord (tworzy powiązanie nazwy akordu z wymaganymi częstotliwościami, adresem URL do schematu

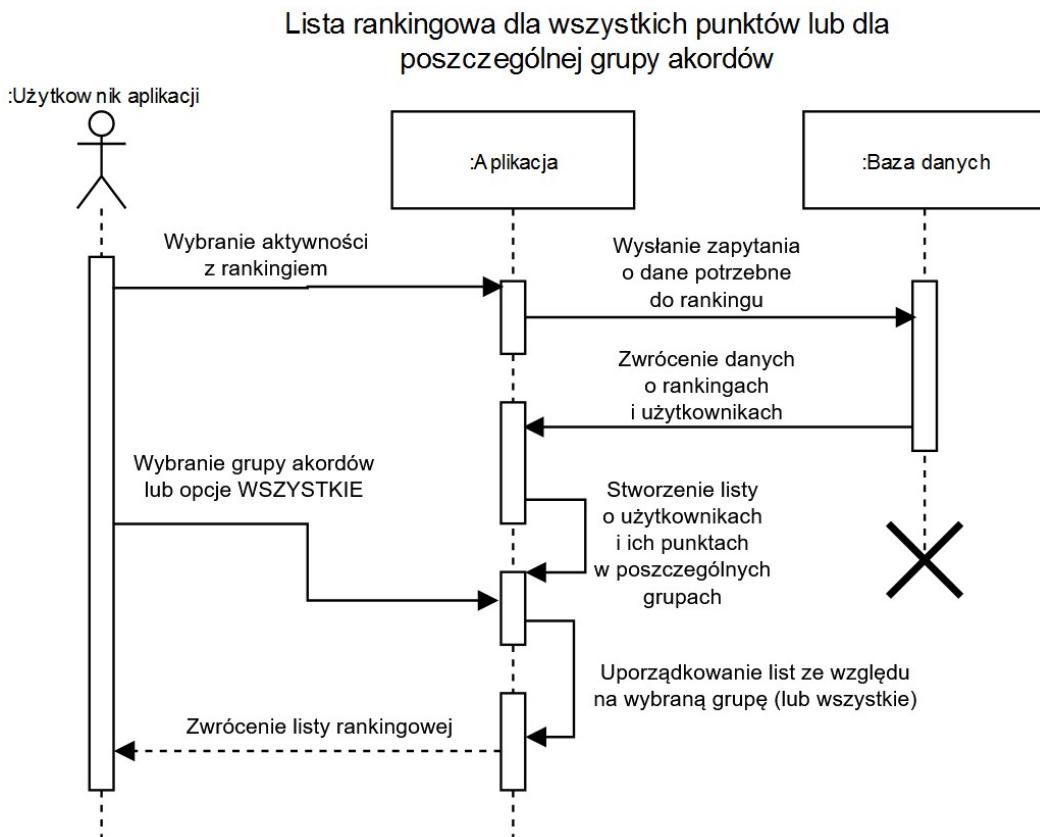
akordu oraz liczbą prób zagrania akordu w trybie rankingu przez danego użytkownika) oraz AuthFirebase. Do połączenia się z bazą służą klasy ChordsCollection oraz LeaderboardCollection. Pierwsza z nich odpowiada za działaniach na kolekcji akordów. Posiada ona informacje o podziale na grupy, wymagane częstotliwości podanego akordu oraz jego schemat. Druga zaś działa na kolekcji leaderboard, która zbiera informację o punktach danego użytkownika w grupie akordów, o danym akordzie oraz o podsumowaniu wszystkich grup.



Rysunek 20. Diagram klas – funkcjonalność sprawdzania poprawności akordów. Źródło: opracowanie własne

## 4.5.Ranking

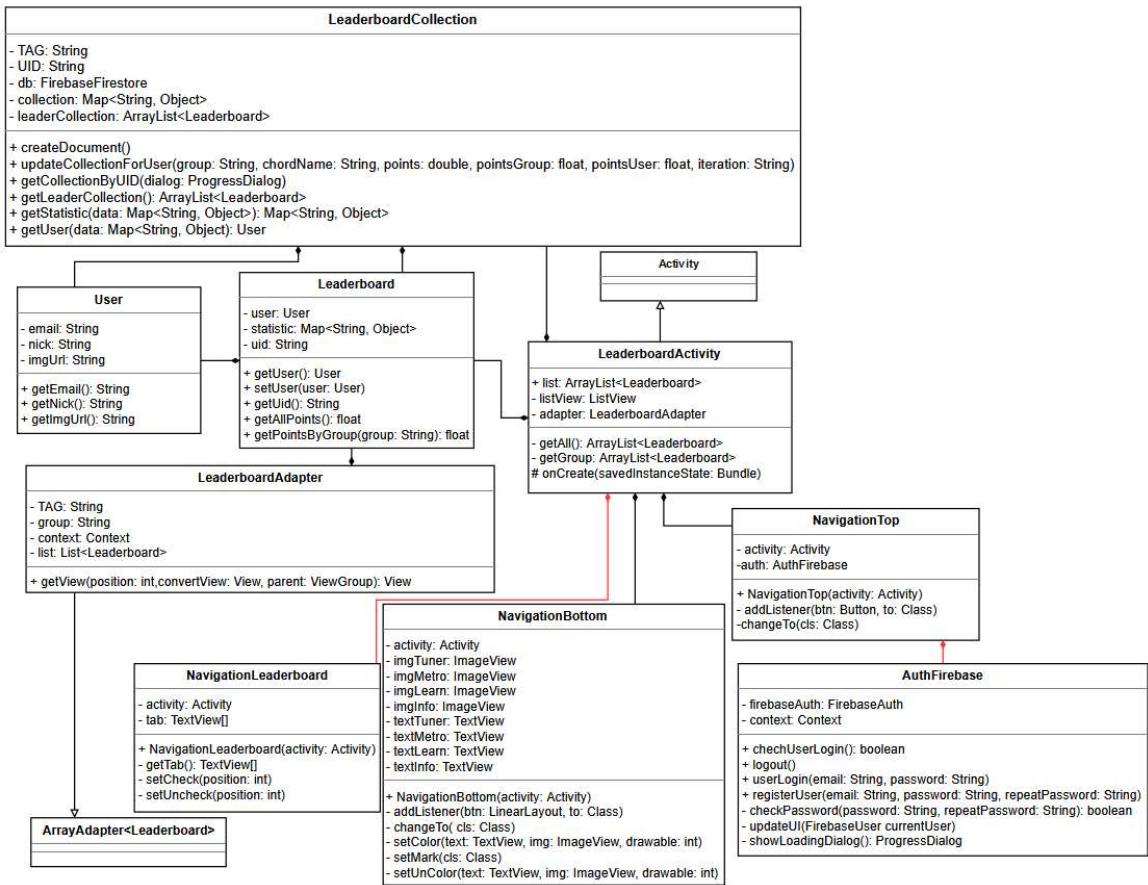
Do okna z rankingiem dostęp będą mieli tylko zalogowani użytkownicy aplikacji. Gdy użytkownik wejdzie w okno, które wyświetla ranking, to zostaje wysłane zapytanie do bazy o spis użytkowników i ich punkty. Gdy baza wyśle takie dane, aplikacja łączy użytkowników z danymi punktami oraz dodaje ich do listy. Następnie lista ta zostaje posortowana ze względu na liczbę punktów przypisanych do użytkownika. Sortowanie danych odbywa się za każdym razem gdy użytkownik zmieni grupę akordów lub gdy wybierze kategorie wszystkie, w którym punkty są sumowane ze wszystkich grup dla danego użytkownika. Przedstawiono to na rysunku 21.



Rysunek 21. Diagram sekwencji – wyświetlanie listy rankingowej. Źródło: opracowanie własne

Na rysunku 22 przedstawiono klasy odpowiedzialne za funkcjonowanie rankingu. Za pomocą klasy LeaderboardActivity powiązany zostaje wygląd okna z jego działaniem rozszerzając klasę Activity. Do LeaderboardActivity dołączane są klasy pomocnicze takie jak NavigationTop, NavigationBottom odpowiedzialne za nawigacje górną oraz dolną w aplikacji oraz NavigationLeaderboard odpowiedzialną za przełączanie się pomiędzy kategoriami rankingu. Do poprawnego wyświetlania listy został zastosowany adapter, dziedziczący po ArrayAdapter, LeaderboardAdapter. Do połączenia z bazą danych użyta

została klasa LeaderboardCollection oraz klasy pomocnicze takie jak User oraz Leaderboard. Ostatnia zawiera informacje o UID użytkownika, jego punktach w poszczególnych kategoriach oraz obiekt typu User informujący o jego nazwie i ikonie.



Rysunek 22. Diagram klas – funkcjonalność list rankingowych. Źródło: opracowanie własne

## **5. Implementacja systemu**

### **5.1. Kompozycja struktury folderów**

Po włączeniu programu za pomocą narzędzia Android Studio po lewej stronie ujrzymy strukturę folderów jakie na rysunku 23. Jest to wygląd folderów stworzonych przez IDE pod Androida. W przypadku gdy ustawiemy, że chcemy zobaczyć faktyczny wygląd struktury folderów wygląd ten nieco się zmieni. Przykładem tego może być folder z układem elementów na ekranie (folder layout). W rzeczywistości jest tam folder o nazwie „layouts” a w nim poszczególne foldery, które odpowiadają za poszczególne elementy – (items – wygląd elementów dołączanych do poszczególnych aktywności, chords – wszystkie układy elementów dla czynności związanych z poprawnością akordów, profile – wszystkie układy aktywności dla okien dotyczących logowania, rejestracji i zarządzania kontem oraz activity – definiuje wygląd pozostałych aktywności).

Folder „java” zawiera wszystkie pliki Javy odpowiedzialne za poprawne działanie aplikacji (funkcje operujące na zdarzeniach oraz wprowadzające odpowiednie zmiany w wyglądzie okien). Struktura w tym folderze jest spójna – tak więc widoczne są foldery, które przechowują wszystkie potrzebne pliki dla poszczególnych okien (leaderboard – rankingi, learnig – wszystkie elementy potrzebne do utworzenia nauki akordów, metronome – pliki potrzebne w oknie metronomu, profile – profil użytkownika oraz logowanie i rejestracja, tuner – pliki odpowiedzialne za działanie stroika oraz folder utility, w którym znajdują się pliki potrzebne w kilku miejscach i nie można ich było przydzielić do innych folderów). Folder collections zawiera podfoldery chords, leaderboard oraz user, a w nich są zawarte pliki Javy potrzebne w komunikacji aplikacji z bazą danych Firebase.



Rysunek 23. Kompozycja folderów w aplikacji. Źródło: opracowanie własne

## **5.2.Implementacja stroika**

Stroik jest dostępny zarówno dla użytkowników zalogowanych jak i dla niezalogowanych. Wygląd tego okna jest determinowany za pomocą pliku XML o nazwie activity\_tuner. Do niego dołączany jest plik XML z nawigacją dolną aplikacji (plik bottom\_nav).

Stroik powinien pozwolić na nastrojenie gitary przy wybranym strojeniu oraz pomóc w nastrojeniu jednej struny jak i wszystkich jednocześnie. Podczas strojenia wszystkich strun aplikacja powinna pozwolić użytkownikowi na strojenie gitary bez konieczności wykonywania czynności na telefonie.

Używając biblioteki TarsosDSP można uzyskać informacje o częstotliwości dźwięku, który dotarł do mikrofonu urządzenia. Z jego pomocą można utworzyć funkcjonalny stroik gitarowy. Funkcja, która na to pozwala znajduje się w klasie PitchDetectionResult, getPitch, która określa częstotliwość za pomocą algorytmu FFT. Funkcja ta jest uruchamiana na osobnym wątku aby nie blokowała ona działania aplikacji.

Zaczynając od strojenia jednej struny aplikacja czeka aż użytkownik określi, którą strunę chce nastroić. Gdy już to uczyni aplikacja sprawdza, które strojenie zostało wybrane. W każdej chwili użytkownik może zmienić wybrany strój jak i strunę, którą chce nastroić. Po wyborze struny sprawdzany zostaje element w liście strun z odpowiednim ID, przypisanym przy tworzeniu ułożenia strun w trakcie tworzenia wyglądu lub po zmianie strojenia. Gdy struna zostanie znaleziona w liście pobrana zostaje wymagana częstotliwość i zapisana do zmiennej pomocniczej. Gdy została już wybrana struna, więc i wymagana częstotliwość, można przejść do kolejnego punktu jakim jest utworzenie nowego wątku i sprawdzenie częstotliwości dla dźwięku pobieranego przez mikrofon urządzenia (rysunek 24).

```

public void checkPitch(final int position) {
    dispatcher = AudioDispatcherFactory.fromDefaultMicrophone( 22050, 1024, 0);
    PitchDetectionHandler pdh = new PitchDetectionHandler() {
        @Override
        public void handlePitch(PitchDetectionResult res, AudioEvent e) {
            final float pitchInHz = res.getPitch();
            runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    for (int i = 0; i < correctSeq.length; i++) {
                        if (correctSeq[i] == 1) {
                            setChosenAndCorrect(i);
                        }
                    }
                    TextView correctText = (TextView) findViewById(R.id.correctPitch);
                    correctText.setText(String.valueOf(frequencyReq));
                    process(pitchInHz, position);
                }
            });
        }
    };
    AudioProcessor pitchProcessor = new PitchProcessor(PitchProcessor.PitchEstimationAlgorithm.FFT_YIN, 22050, 1024, pdh);
    dispatcher.addAudioProcessor(pitchProcessor);
    audioThread = new Thread(dispatcher, name: "Audio Thread");
    audioThread.start();
}

```

Rysunek 24. Kod otwierający nowy wątek sprawdzenia częstotliwości pobranej przez mikrofon urządzenia. Źródło: opracowanie własne

Otwierany jest wątek w metodzie checkPitch w klasie TunerActivity. W tej metodzie nadpisywana jest funkcja dostępna w bibliotece TarsosDSP, która będzie przekazywać na bieżąco wyniki działania algorytmu FFT. Kolejnym krokiem jest sprawdzenie czy odebrana częstotliwość jest zgodna z wymaganą częstotliwością.

Aby przejść w tryb automatycznego strojenia należy przełącznik „Auto” przesunąć w prawo. Jeśli wcześniej został uruchomiony wątek strojenia gitary po jednej strunie to zostanie on wyłączony. Do momentu uruchomienia nowego wątku działanie aplikacji jest takie samo z wyjątkiem wybierania strun. Wątek po odczytaniu częstotliwości sprawdza, której najbliżej znajduje się dźwięk. Każda ze strun ma przypisaną częstotliwość dzięki czemu aplikacja jest w stanie określić, do której z częstotliwości jest najbardziej przybliżona częstotliwość odczytana przez mikrofon. Następnie aplikacja wskazuje, która struna, według częstotliwości, jest grana i wskazuje jak bardzo ta częstotliwość odbiega od wymaganej (to samo dzieje się dla działania aplikacji w trybie strojenia jednej struny).

Sprawdzenie czy dany dźwięk jest najbardziej zbliżony, do którejś z częstotliwości przypisanych do strun, odbywa się poprzez metodę dostępną w klasie Notes, nazwaną „getNote” (rysunek 25).

```

public static String[] getNote(String frequency) {
    if (!notes.isEmpty()) {
        for (String[] a : notes) {
            float noteFreq = Float.parseFloat(a[1]); //częstotliwość dla danej struny
            float freq = Float.parseFloat(frequency);
            float difference = (int) Math.round(noteFreq - freq);
            if (Math.abs(difference) <= 10 ) {
                return new String[]{a[0], String.valueOf(difference), a[1]};
            }
        }
    }
    return null;
}

```

Rysunek 25. Funkcja zwracająca tablice z nazwą struny, różnicę z podaną w parametrze oraz wymaganą częstotliwość. Źródło: opracowanie własne

Na początku sprawdzone zostaje czy w tablicy strun zostały wczytane struny (podczas zmiany strojenia). Następnie zostają utworzone zmienne dla każdej ze strun w tablicy, każda ma typ float i są kolejno odpowiedzialne za częstotliwość aktualnie sprawdzanej struny, odczytaną przez mikrofon częstotliwość oraz różnicę tych dwóch wartości. Ta różnica zostaje zaokrąglona, a później zostaje pozbawiona znaku. Różnica ta jest sprawdzana pod kątem wielkości i jeśli różnica będzie większa niż 10 to zostanie zbadana kolejna struna, w innym przypadku zostanie zwrócona tablica z nazwą struny, różnicą oraz poprawną częstotliwością dla danej struny.

### 5.3. Implementacja metronomu

Metronom jest dostępny dla użytkowników zalogowanych oraz niezalogowanych. Wygląd okna jest determinowany przez plik XML o nazwie activity\_metronome. Do niego dołączany jest plik XML z dolną nawigacją aplikacji (plik bottom\_nav).

Metronom powinien pozwalać na uruchomienie funkcji odtwarzającej dźwięk w danym rytmie oraz pozwalać na zmianę tempa oraz metrum. Uruchomienie metronomu powinno odbywać się na osobnym wątku w aplikacji aby nie zablokować działania całej aplikacji. Ponadto dostępne powinny być dwa różne sygnały dźwiękowe.

Uruchomienie działania metronomu odbywa się poprzez kliknięcie w przycisk START. Uruchamia on wątek, który sprawdzając aktualny stan ustawień odpowiednio ustawia odległość kolejnego dźwięku. W ustawieniach możemy określić metrum oraz tempo w jakim ma działać metronom. Do wyboru są przygotowane osiem możliwych temp oraz jedenaście metrum. Jeśli użytkownik po kliknięciu przycisku START zmieni które z tych ustawień, metronom zmieni odległość dźwięków i dostosuje ich rodzaj. Działanie wątku zakończy się tylko wtedy, gdy użytkownik kliknie przycisk STOP lub przejdzie do innego okna.

Aby uniknąć błędów spowodowanych przez brak odpowiednich danych jak i też przepełnieniem pamięci wykorzystany został stan wątku (rysunek 26).

```
if (metronomeThread.getState() == Thread.State.NEW) {
    metronomeThread.start();
} else {
    metronomeThread = new Thread(metronome);
    metronomeThread.start();
}
```

Rysunek 26. Poprawne uruchomienie przerwanego i nowego wątku. Źródło: opracowanie własne

Działanie nieprzerwane metronomu jest uzyskanie poprzez budowę odpowiedniej klasy implementującej interfejs Runnable i nadpisanie metody run widocznej na rysunku 27.

```
@Override
public void run() {
    this.start = true;
    while (start) {
        beep(counter);
        counter++;
        try {
            /*wyliczenie odległości pomiędzy dźwiękami*/
            Thread.sleep((long)(1000 * (60.0/bpm)));
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

Rysunek 27. Kod wątku metronomu. Źródło: opracowanie własne

Klasa metronomu pozwala na zmianę tempa oraz metrum w dowolnej chwili co powoduje zmianę odległości między dźwiękami oraz uruchomienie właściwego dźwięku (rysunek 27 wraz z rysunkiem 28).

```

public void setBpm(double bpm) {
    this.bpm = bpm;
}

public void setMeasure(double measure) {
    this.measure = measure;
}

private void beep(int counter) {
    if (counter%measure == 0) {
        beat2.start();
    }else {
        beat1.start();
    }
}

```

Rysunek 28. Funkcje metronomu. Od góry ustalenie tempa, ustalenie metrum oraz funkcja uruchamiająca właściwy sygnał dźwiękowy. Źródło: opracowanie własne

#### 5.4. Tworzenie i uzupełnienie bazy danych z poziomu aplikacji

Baza danych, jak i inne funkcje serwerowe, jest umieszczona na platformie Firebase od Google i jest ona typu NoSQL. Na platformie Firebase przyjęto model dokumentów. Każdy dokument ma swój unikalny klucz i jest przechowywany w zbiorze nazywanym kolekcją. W tym przypadku są dostępne 3 kolekcje. Pierwsza z nich to kolekcja chords (rysunek 29). W niej znajdują się aktualnie 5 dokumentów nazwanych grupami akordów. W każdej z grup znajdują się minimum 3 akordy. Ich klucze odpowiadają nazwom akordów i zawierają tablice. Pierwszy element z tablicy zawiera listę zbudowaną jako klucz-wartość i zawiera nazwę struny oraz wymaganą częstotliwość. Drugim elementem w tablicy jest schemat akordu, a dokładnie adres URL wskazujący na zasób umieszczony w Firebase Storage.

```

▶ D-dur: [{StringA: "0", StringD: "..."}]
▶ E-moll: [{StringA: "123", StringD: "..."}]
▼ G-dur
  ▼ 0
    StringA: "123"
    StringD: "147"
    StringE: "98"
    StringG: "196"
    StringH: "294"
    StringI: "392"
  ▶ 1 {schema: "https://firebase..."}

```

Rysunek 29. Przykładowy dokument w kolekcji chords. Źródło: opracowanie własne

Kolekcja chords jest wykorzystywana do funkcjonalności aplikacji zajmujących się nauką akordów zarówno w trybie nauki jak i w trybie rankingu.

Kolejną kolekcją jest kolekcja users (rysunek 30), której dokumenty przyjmują jako klucz wygenerowane podczas rejestracji UID. Wewnątrz dokumentu znajdują się informacje o adresie email, nazwie użytkownika oraz adres URL ikony użytkownika. Ta kolekcja jest wykorzystywana do wyświetlania i edycji danych w oknie profilu oraz, współpracując z kolekcją leaderboard, do poprawnego wyświetlania danych na temat rankingu użytkowników.

```

avatar: "https://firebasestorage.googleapis.com/v0/b/guitar-
tuna.appspot.com/o/user-ico%2Fuser-22.png?alt=media&
token=e0938cd8-1cc3-4413-bd1e-507561b801a8"
email: "test@test.pl"
nick: "test"

```

Rysunek 30. Przykładowy dokument w kolekcji users. Źródło: opracowanie własne

Zawartość kolekcji leaderboard (rysunek 31) służy do pobrania takich informacji jak liczba prób w trybie rankingu dla danego akordu, ogólna liczba punktów uzyskana podczas grania danego akordu, w danej grupie czy ogólna liczba punktów uzyskana we

wszystkich akordach. Aby to ułatwić akordy są podzielone w taki sam sposób jak w kolekcji chords jednak z tym wyjątkiem, że zamiast zawierać informacje o wymaganych częstotliwościach czy adresie URL schematu zawierają tylko informacje o punktach i próbach. Do szybszego pobrania danych zastosowano jeszcze jedno pole dla każdej grupy oraz dla podsumowania, które zawierają informacje odpowiednio o wszystkich punktach w grupie i liczbie punktów ogółem. Dla rozpoznania, dla którego gracza są to statystyki zastosowano jako klucz dokumentu UID użytkownika.

```
all: 12000
  ▼ group1
    ▼ D-dur
      attempt: "0"
      points: "0"
      ▶ E-moll: {attempt: "0", points: "0"}
      ▶ G-dur: {attempt: "0", points: "0"}
      all: 0
      ▶ group2: {A-dur: {attempt: "0", poi...}}
      ▶ group3: {A7: {attempt: "0", points...}}
      ▶ group4: {A5: {attempt: "0", points...}}
      ▶ group5: {B7: {attempt: "0", points...}}
```

Rysunek 31. Przykładowy dokument w kolekcji leaderboard. Źródło: opracowanie własne

Połączenie z bazą oferują klasy ChordsCollection, LeaderboardCollection oraz UsersCollection. Aby można było dokonać jakikolwiek działań na bazie danych najpierw należy nawiązać połączenie poprzez użycie metody ujętej na rysunku 32, a następnie zapisać ją do zmiennej typu FirebaseFirestore.

```
this.db = FirebaseFirestore.getInstance();
```

Rysunek 32. Przypisanie do zmiennej db instancji bazy danych. Źródło: opracowanie własne

Firebase korzysta z dokumentów, które oferują obiekty oparte na zasadzie klucz-wartość. Z tego też powodu podczas działań na bazie też powinno korzystać się z obiektów opartych o schemat klucz-wartość. Najlepszym wyborem zdaje się być przechowywanie danych w mapach. Tak też jest skonstruowane przetłumaczenie danych z bazy na

wspomniane wcześniej mapy w aplikacji. Przykładem może być kawałek kodu odpowiadający za stworzenie nowego dokumentu w kolekcji users pod kluczem UID stworzonego właśnie użytkownika.

```
Map<String, Object> user = new HashMap<>();
```

Rysunek 33. Utworzenie obiektu mapy. Źródło: opracowanie własne

Na początku tworzony jest obiekt mapy, któremu określano przechowywany typ klucza na string, a wartość na typ object (rysunku 33). Następnie do tego obiektu należy dodać kolejne obiekty. Na rysunku 34 dodaje pod kluczem email wartość określająca adres email użytkownika.

```
user.put( k: "email", email);
```

Rysunek 34. Dodanie do obiektu mapy obiekt typu klucz wartość. Źródło: opracowanie własne

```
db.collection( collectionPath: "users" ).document(UID)
    .set(user, SetOptions.merge());
```

Rysunek 35. Dodanie zmian do kolekcji users pod danym UID. Źródło: opracowanie własne

Ostatnim krokiem (rysunek 35) do dodania nowego dokumentu jest na obiekcie bazy danych (w tym przypadku zmienna db) wywołać metodę określającą, w której kolekcji mają nastąpić zmiany (w tym przypadku jest to kolekcja users). Poprzez metodę document określa się identyfikator dokumentu, który w tym przypadku jest UID stworzonego użytkownika. Następnie wybierana zostaje akcja, w tym przypadku wybrana zostaje metoda set i w parametrze określony zostaje obiekt user jako, że to właśnie w tym obiekcie są przechowane dane, które należy umieścić w bazie. W przypadku gdy chcemy zmodyfikować tylko niektóre dane w danym dokumencie należy jako drugi parametr określić jako SetOptions.merge(). W takim przypadku, gdy nie będzie któregoś z kluczy, który jest w bazie, to nie zostanie on nadpisany. Gdyby zabrakło drugiego parametru w metodzie set to wskazany dokument zostałby całkowicie zmieniony przez ten, który został stworzony w obiekcie user.

```
.addOnCompleteListener(new OnCompleteListener<QuerySnapshot>() {
    @Override
    public void onComplete(@NonNull Task<QuerySnapshot> task) {
        if (task.isSuccessful()) {
```

Rysunek 36. Metoda wykonywana po poprawnym pobraniu kolekcji. Źródło: opracowanie własne

```
for (QueryDocumentSnapshot document : Objects.requireNonNull(task.getResult())) {  
    if (document.getId().equals(userID)) {
```

Rysunek 37. Wyszukanie dokumentu z danym kluczem. Źródło: opracowanie własne

Do pobrania dokumentów służy metoda get. Jest ona bezparametrowa i zwraca całą kolekcję. Dlatego też, po sprawdzeniu czy dana kolekcja się pobrała (rysunek 36), wyszukiwany jest interesujący dokument przez sprawdzenie jego klucza (rysunek 37).

## 5.5. Administracja użytkownikami

Zalogowany użytkownik zdobywa dostęp do nowych funkcjonalności. Aby się zalogować użytkownik musi podać adres email i hasło, którymi się rejestrował. Okno logowania jest determinowane przez plik XML z nazwą activity\_login, a rejestracji przez plik activity\_registration. Rejestracja użytkownika przebiega w następujących krokach. Pierwszym z nich jest przejście przez użytkownika do okna rejestracji oraz wypełnieni formularza, na który składa się adres email oraz dwa pola z hasłem, które muszą się zgadzać oraz posiadać minimum 6 znaków. W kodzie, pierwszym krokiem, który następuje zaraz po uruchomieniu okna rejestracji jest nawiązanie połączenia z funkcją Firebase Authentication (tu zmienna firebaseAuth), która jest odpowiedzialna za autentykację użytkowników oraz ułatwienie zarządzania nimi, po stronie konsoli Firebase. Po kliknięciu w przycisk rejestracji na ekranie uruchamiana jest funkcja registerUser znajdująca się w klasie AuthFirebase. Po usunięciu białych znaków i sprawdzeniu poprawności danych przychodzi czas na wywołanie odpowiednich metod z klas Firebase.

```
firebaseAuth.createUserWithEmailAndPassword(email, password)  
.addOnCompleteListener((Activity) context, new OnCompleteListener<AuthResult>() {  
    @Override  
    public void onComplete(@NonNull Task<AuthResult> task) {  
        if (task.isSuccessful()) {  
            Log.d(tag: "Registration", msg: "createUserWithEmailAndPassword:success");  
            firebaseUser user = firebaseAuth.getCurrentUser();  
            UsersCollection uc = new UsersCollection(user.getUid());  
            uc.addUser(finalEmail);  
            LeaderboardCollection lc = new LeaderboardCollection(user.getUid());  
            lc.createDocument();  
            dialog.dismiss();  
            updateUI(user);  
        } else {  
            Log.w(tag: "Registration", msg: "createUserWithEmailAndPassword:failure", task.getException());  
            dialog.dismiss();  
            Toast.makeText(context, "Authentication failed.",  
                Toast.LENGTH_SHORT).show();  
            updateUI(currentUser: null);  
        }  
    }  
});
```

Rysunek 38. Metody wywoływanie na obiekcie firebaseAuth do utworzenia nowego użytkownika. Źródło: opracowanie własne

Metodą createUserWithEmailAndPassword (rysunek 38) zostają przekazane wpisane przez użytkownika dane oraz zostaje otwarte konto danego użytkownika. Po poprawnym utworzeniu konta (sprawdzone to zostaje poprzez sprawdzenie warunku task.isSuccessful) tworzone zostają dokumenty w kolekcji users oraz leaderboard. Następnie użytkownik poprzez metodę updateUI zostaje przekierowany do widoku swojego profilu. Jednak gdy z jakiegoś powodu nie można było stworzyć konta (np. podany adres email jest przypisany do innego konta) zostanie wyświetlona informacja o niepowodzeniu procesu autentykacji.

Wygląd okna profilu wykonany jest za pomocą pliku XML activity\_profile. Za pomocą metody onCreate klasa ProfileActivity zostaje powiązana ze wspomnianym wcześniej widokiem. Do poprawnego działania tego okna potrzebne jest uzyskanie informacji o użytkowniku. W tym celu zostaje uruchomiona funkcja readUser, która pobiera dane o użytkowniku z kolekcji users. Dane zawarte w tej kolekcji to adres URL odnoszący się ikony użytkownika, adres email przypisany do konta oraz nazwa użytkownika. Te dane zostają pobrane i wyświetlane za pomocą kodu przedstawionego na rysunku 39.

```
TextView email = findViewById(R.id.email);
EditText nick = findViewById(R.id.nick);
ImageView avatar = findViewById(R.id.avatar);
Button updateData = findViewById(R.id.change);

email.setText(user.getEmail());
nick.setText(user.getNick());

Picasso.with(this) Picasso
    .load(user.getImgUrl()) RequestCreator
    .placeholder(R.mipmap.ic_guitar_round) RequestCreator
    .into(avatar);
```

Rysunek 39. Wyświetlanie danych o użytkowniku. Źródło: opracowanie własne

Do ikony użytkownika (zmienna avatar) oraz przycisku (zmienna updateData) zostają przypisani słuchacze. Są oni uruchamiani przy kliknięciu w dany obszar. Na rysunku 40 został umieszczony kod słuchacza na zmiennej updateData.

```
updateData.setOnClickListener((view) → {
    final UsersCollection uc = new UsersCollection(Objects.requireNonNull(firebaseAuth.getCurrentUser()).getUid());
    EditText nick = findViewById(R.id.nick);
    if (!TextUtils.isEmpty(nick.getText().toString().trim())) {
        uc.updateUser(
            activity: ProfileActivity.this,
            nick.getText().toString().trim(),
            url: "");
    }
});
```

Rysunek 40. Sekwencja operacji po kliknięciu w obiekt updateData. Źródło: opracowanie własne

W przypadku przycisku zostaje pobrany tekst w polu nick (okno do wpisywania danych z klawiatury urządzenia) i przekazany zostaje jako parametr do metody updateUser, która zmieni zawartość danych w bazie danych w kolekcji users dla zalogowanego użytkownika.

Metoda updateUser (rysunek 41) przyjmuje 3 parametry. Pierwszy z nich to kontekst aplikacji, w tym przypadku to klasa ProfileActivity. Następnie przyjmuje dwa obiekty typu string. Pierwszy z nich odpowiada za nową nazwę użytkownika, a drugi za nowy adres URL ikony użytkownika.

```
public void updateUser(final Activity activity, String nick, String url) {
    Map<String, Object> user = new HashMap<>();
    if (!url.equals(""))
        user.put("avatar", url);
    if (!nick.equals(""))
        user.put("nick", nick);

    db.collection("users").document(userID).set(user, SetOptions.merge());
    .addOnSuccessListener(new OnSuccessListener<Void>() {
        @Override
        public void onSuccess(Void aVoid) {
            Toast.makeText(activity, "Dane zostały zmienione", Toast.LENGTH_SHORT).show();
        }
    }).addOnFailureListener(new OnFailureListener() {
        @Override
        public void onFailure(@NonNull Exception e) {
            Toast.makeText(activity, "Nastąpił nieoczekiwany błąd. Spróbuj ponownie później.", Toast.LENGTH_SHORT).show();
        }
    });
}
```

Rysunek 41. Metoda zmieniająca odpowiednie dane w kolekcji users dla zalogowanego użytkownika. Źródło: opracowanie własne

Po sprawdzeniu, który z nich jest wartością pustą i odpowiednio dodaniu do wartości do obiektu zostaje uruchomiona metoda set (dla dokumentu w kolekcji users, który klucz ma taki jak UID zalogowanego użytkownika), która przyjmuje jako parametr nowy obiekt. Ważne jest aby w set jako drugi parametr ustawić SetOptions.merge(), co sprawi że dokument nie nadpisze się, a zmienią się tylko te dane, które mają inną wartość ustawiając to co, w tym przypadku, znajduje się w obiekcie user.

```
avatar.setOnClickListener((view) -> {
    finish();
    overridePendingTransition(enterAnim: 0, exitAnim: 0);
    startActivity(new Intent(packageContext: ProfileActivity.this, SelectAvatarActivity.class));
    overridePendingTransition(enterAnim: 0, exitAnim: 0);
});
```

Rysunek 42. Sekwencja operacji po kliknięciu w obiekt avatar. Źródło: opracowanie własne

W przypadku chęci zmiany ikony użytkownika trzeba najpierw wywołać (poprzez kliknięcie w aktualną ikonę użytkownika) okno z dostępnymi ikonami – SelectAvatarActivity (rysunek 42). Dzieje się to za pomocą metody startActivity. Pierwszy parametr

to aktualny kontekst, a drugi to klasa, która ma zostać wywołana. Reszta odpowiada za animację zmiany okna.

```
setContentView(R.layout.activity_select_avatar);

array.add(
    new IcoUser(
        activity: this,
        url: "user-1.png?alt=media&token=0123456789012345678901234567890",
        R.id.ico_1
);
}
```

Rysunek 43. Wskazanie pliku XML wyglądu okna oraz dodanie do listy obiekt typu IcoUser. Źródło:

opracowanie własne

Plik JAVA zostaje powiązany z plikiem XML odpowiadającym za ułożenie elementów (activity\_select\_avatar). Następnie do listy (zmienna array) są dodawane obiekty typu IcoUser. Obiekty klasy IcoUser zawierają pole z adresem URL obrazka i pole z miejscem wyświetlania obrazka (rysunek 43).

```
for (int i = 0; i < array.size(); i++) {
    Picasso.with(this) Picasso
        .load(array.get(i).getUrl()) RequestCreator
        .placeholder(R.mipmap.ic_guitar_round) RequestCreator
        .resize(targetWidth: 150, targetHeight: 150) RequestCreator
        .centerCrop() RequestCreator
        .into(array.get(i).getDisplay());
    final int finalI = i;
    array.get(i).setOnClickListener((view) -> {
        final UsersCollection uc = new UsersCollection(Objects.requireNonNull(firebaseAuth.getCurrentUser()).getUid());
        uc.updateUser(
            activity: SelectAvatarActivity.this,
            nick: "",
            array.get(finalI).getUrl());
        finish();
        overridePendingTransition(enterAnim: 0, exitAnim: 0);
        startActivity(new Intent(packageContext: SelectAvatarActivity.this, ProfileActivity.class));
        overridePendingTransition(enterAnim: 0, exitAnim: 0);
    });
}
}
```

Rysunek 44. Sekwencja operacji pozwalająca na poprawne wyświetlanie dostępnych ikon oraz wywołujące zmiany dla zalogowanego użytkownika. Źródło: opracowanie własne

Następnie dla każdego obiektu zostaje wyświetlony obraz oraz dołączony słuchacz pozwalający na zmianę adres URL ikony dla zalogowanego użytkownika (rysunek 44). W tym celu znowu została zastosowana metoda updateUser, tylko jako nowa nazwa użytkownika został przekazany pusty string, a dla nowego adresu został wyciągnięty adres z klikniętego obiektu ikony i przekazany do metody jako parametr URL. Następnie została zastosowana zmiana okna i jego animacja.

## 5.6.Sprawdzenie poprawności akordów

Funkcjonalność dotycząca nauki akordów jest dostępna tylko dla zalogowanych użytkowników, a dla niezalogowanych jest możliwość tylko przeglądnięcia dostępnych

akordów. Wygląd dla okien spełniających tę funkcjonalność to pliki XML o nazwie activity\_basic\_learning, activity\_show\_chord, item\_chords\_group (adapter dla obiektu ListView, odpowiada za poprawne wyświetlenie listy grupy akordów), item\_chords (adapter dla obiektu ListView, odpowiada za poprawne wyświetlenie listy akordów w grupie). Do niego dołączany jest plik XML z nawigacją dolną aplikacji (plik bottom\_nav) oraz z nawigacją górną aplikacji (plik top\_nav).

Sprawdzenie akordu odbywa się, dla każdej struny na tej samej zasadzie. Odbywa się ono poprzez porównanie częstotliwości nagranej przez mikrofon z danym dźwiękiem akordu, dla którego częstotliwość została pobrana z bazy danych. Częstotliwość nagrana przez mikrofon odpowiada częstotliwości konkretnego dźwięku akordu, czyli struny przyciśniętej w odpowiednim miejscu na gryfie. Dane zostają pobrane dopiero w chwili wyboru przez użytkownika danego akordu.

```
public void fetchChord(final ProgressDialog dialog, final String group, final String key) {
    db.collection( collectionPath: "chords")
        .get()
        .addOnCompleteListener(new OnCompleteListener<QuerySnapshot>() {
            @Override
            public void onComplete(@NonNull Task<QuerySnapshot> task) {
                if (task.isSuccessful()) {
                    for (QueryDocumentSnapshot document : Objects.requireNonNull(task.getResult())) {
                        if (document.getId().equals(group)) {
                            chord = getValuesOfKeyChord(document, key);
                            queryAttempt(chord, dialog, group, key);
                        }
                    }
                } else {
                    Log.w(TAG, msg: "Error getting documents.", task.getException());
                }
            }
        });
}
```

Rysunek 45. Metoda pobierająca z kolekcji chords odpowiedni akord oraz pobranie danych statystycznych użytkownika. Źródło: opracowanie własne

Dane o akordzie pobierane zostają z kolekcji chords przy pomocy funkcji zawartej na rysunku 45, a takie dane jak informacja o ilości prób lub punkty w kategoriach akord, grupa akordów lub podsumowanie pobierane są z kolekcji leaderboard (na zdjęciu metoda queryAttempt). Nadpisanie metody addOnCompileListener daje nam dostęp do danych w momencie powodzenia pobrania kolekcji. Jest to bardzo pomocne, ponieważ już teraz kod jest zabezpieczony przed działaniem na pustych obiektach. Następnie przechodzi po kolekcjach i wybiera ten dokument, którego kluczem jest odpowiednia nazwa grupy akordu. Kolejno są wywołane funkcje odpowiedzialne za tworzenie akordu z wymaganymi częstotliwościami i schematem danego akordu (rysunek 46) oraz funkcja odpowiedzialna za dane o postępach użytkownika.

```

private Chord getValuesOfKeyChord(QueryDocumentSnapshot document, String key) {
    Map<String, Object> objJSON = new HashMap<>();
    objJSON = (Map<String, Object>) document.getData();
    ArrayList array = (ArrayList) objJSON.get(key);
    HashMap schema = (HashMap) array.get(1);
    HashMap correctFreq = (HashMap) array.get(0);
    return new Chord(
        key,
        new String[] { // wymagane częstotliwości
            (String) correctFreq.get("StringE"),
            (String) correctFreq.get("StringA"),
            (String) correctFreq.get("StringD"),
            (String) correctFreq.get("StringG"),
            (String) correctFreq.get("StringH"),
            (String) correctFreq.get("StringE"),
        },
        schema.get("schema").toString() //schemat
    );
}

```

Rysunek 46. Metoda zwracająca obiekt Chord z wymaganymi częstotliwościami na danych strunach oraz schematem akordu. Źródło: opracowanie własne

Obie funkcje są dość podobne do siebie. Dokumenty są dzielone na mniejsze obiekty i rzutowane zostają na mapy typu HashMap, dzięki czemu łatwiej można pobrać dane. Wystarczy wywołać metodę get a jako parametr podać klucz, który w tym przypadku jest też kluczem w dokumencie.

Po pobraniu danych o akordzie (i po wcześniejszym wybraniu trybu – w tym przypadku jest to tryb nauka) zostaje uruchomiony wątek, który sprawdza poprawność dźwięku na strunie. Jeśli dźwięk jest poprawny to odpowiednia struna zostaje zaznaczona na kolor zielony. W kolejnej iteracji pętli ta struna zostaje pominięta w sprawdzaniu. Jeśli wszystkie struny zostają oznaczone jako prawidłowe, pętla kończy się i użytkownik zostaje poinformowany o poprawnym zagraniu akordu.

W przypadku trybu ranking następują trzy zmiany. Otóż pierwsza rzucająca się zmiana to zniknięcie schematu i pojawienie się przycisku „pokaż schemat”. Pozwala ona na określenie punktów w dalszej części działania programu. Gdy schemat zostaje wyświetlony poprzez kliknięcie w przycisk „Pokaż schemat” zostaje on włączny przez resztę działania rundy (składa się z 4 prób) oraz nie zostanie naliczone 100 punktów. Kolejną rzucającą się w oczy zmianą jest pojawienie się licznika czasu (zestawienie punktów dla poszczególnych przedziałów czasowych umieszczone w tabeli 1). Jest on bardzo ważny w procesie przyznawania punktów. Trzecim składnikiem w przyznawaniu punktów jest ogólna liczba podejść przez użytkownika. Wyróżnione są 3 etapy. Jeśli użytkownik pod-

chodzi do danego akordu pierwszy, drugi lub trzeci raz to liczba punktów nadana w próbie jest mnożona odpowiednio przez 5, 4, 3 a w pozostałych przypadkach mnożnik punktów nie występuje.

*Tabela 1. Zestawienie punktów i przedziałów czasowych. Źródło: opracowanie własne*

| czas [s]   | <8  | 8-9 | 10-19 | 20-29 | 30-39 | 40-49 | 50-59 | 60-120 |
|------------|-----|-----|-------|-------|-------|-------|-------|--------|
| L. punktów | 200 | 150 | 140   | 130   | 120   | 110   | 100   | 50     |

Kolejną zmianą w trybie rankingu jest okienko, które pojawia się po zagraniu wszystkich strun poprawnie. Otóż użytkownik zostaje poinformowany o numerze próby w danej rundzie oraz o punktach zdobytych podczas trwania próby i podsumowanie punktów zdobytych w rundzie.

## **5.7. Wyświetlenie rankingu z podziałem na poszczególne grupy**

Funkcjonalność dotycząca rankingu jest dostępna tylko dla zalogowanych użytkowników. Pliki XML activity\_leaderboard oraz item\_leaderboard odpowiadają za wygląd okna z listą rankingową. Do niego dołączany jest plik XML z nawigacją dolną aplikacji (plik bottom\_nav), nawigacją górną aplikacji (plik top\_nav) oraz z nawigacją rankingu czyli wybór odpowiedniej kategorii rankingu (plik leaderboard\_nav). Wyświetlanie listy rankingowej odbywa się poprzez połączenia z danymi otrzymanymi z bazy danych i poprzez adapter wyświetlona zostaje lista posortowanych użytkowników ze względu na liczbę zdobytych punktów.

Po kliknięciu w przycisk uruchamiający okno z rankingiem, otwarte zostaje okno ładowania, które zamyka się dopiero gdy uzyskamy kompletne dane z dwóch kolekcji – „leaderboard” oraz „users”. Gdy dane zostają pobrane i zwrócona zostaje lista danych o użytkownikach i ich statystykach, następuje wyświetlanie tej listy przez adapter. Dzieje się to za pomocą klasy LeaderboardAdapter (rozszerza ArrayAdapter), która w konstruktorze przyjmuje okno aktualnie wyświetlonej aktywności, listę z danymi oraz nazwę grupy, której dane mają zostać wyświetcone. W przypadku podania pustego stringu lub wartości null wyświetlane dane będą dotyczyć danych podsumowujących. Aby wyświetlić dane w widoku dodanym do XML odpowiedzialnym za wygląd okna, należy nadpisać metodę getView (dostępna z ArrayAdapter), która zwraca widok elementu listy.

Zdecydowanie się na stworzenie własnego adaptora jest w głównej mierze determinowane przez stworzenie własnego wyglądu listy. Jest to na tyle ważne, ponieważ nazwy użytkowników mogą się powtórzyć, a żaden z dostępnych domyślnie adapterów listy nie

był wstanie wyświetlić tyle informacji (avatar i nick użytkownika, numer pozycji w rankingu oraz punkty zdobyte przez użytkownika w danej kategorii) w przyjemnej i czytelnej formie. Dlatego też w pliku item\_leaderboard.xml zapisany został wygląd pojedynczego elementu listy.

```
View listItem = convertView;
if (listItem == null) {
    listItem = LayoutInflater
        .from(context)
        .inflate(R.layout.item_leaderboard, parent, attachToRoot: false);
```

Rysunek 47. Sekwencja operacji łącząca wygląd poszczególnych elementów w liście. Źródło: opracowanie własne

W adapterze zostaje nadpisana metoda oferująca domyślny wygląd listy, która będzie wywoływana do każdego elementu listy. W pierwszym kroku (rysunek 47) należy przypisać do zmiennej aktualny widok listy. Jeśli wcześniej nie został wybrany należy przypisać wynikową wartość z LayoutInflater, w tym miejscu zostaje wskazany widok, dzięki któremu zostanie zwrócona lista z własnym widokiem.

```
TextView nick = listItem.findViewById(R.id.nick);
TextView place = listItem.findViewById(R.id.place);
TextView points = listItem.findViewById(R.id.points);
ImageView avk = listItem.findViewById(R.id.avk);
```

Rysunek 48. Łączenie obiektów JAVA z obiektami wyświetlającymi się na ekranie. Źródło: opracowanie własne

Kolejnym krokiem (rysunek 48) jest przypisanie pól z pliku XML do zmiennych w pliku Java. Można tego dokonać, ponieważ pod zmienną listItem będzie aktualny widok z pliku XML. Zmienną listItem będzie traktować jako kontekst aplikacji, co umożliwi wybór pól, które zostaną przypisane do zmiennych odpowiadających za miejsce wyświetlenia np. ikony użytkownika.

```
Picasso.with(context) Picasso
    .load(current.getUser().getImgUrl()) RequestCreator
    .placeholder(R.mipmap.ic_guitar_round) RequestCreator
    .resize( targetWidth: 150, targetHeight: 150) RequestCreator
    .centerCrop() RequestCreator
    .into(avk);

nick.setText(current.getUser().getNick());
```

Rysunek 49. Sekwencja operacji umożliwiająca wyświetlenie obrazu oraz wyświetlić nazwę użytkownika. Źródło: opracowanie własne

Gdy pod zmiennymi będą obiekty z pliku XML można przejść do przypisania danych z listy otrzymanej w konstruktorze klasy LeaderboardAdapter (rysunek 49). Do obiektów typu TextView używa się metody setText, która w parametrze przyjmuje tekst jaki ustawi na ekranie. W przypadku obiektu ImageView wykorzystana została biblioteka Picasso, która w łatwy sposób ustawia obraz ze ścieżki URL, przy czym przed załadowaniem obrazka pokazuje się wcześniej ustalony obrazek. W tym przypadku to obraz, który jest ustalony jako ikona aplikacji (ic\_guitar\_round). Po ustaleniu co ma się wyświetlać, w którym miejscu, zwrócony zostaje obiekt listItem.

```
tab[i].setOnItemClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        nav.setChecked(finalI);
        if (finalI == 0) {
            list = getAll();
            adapter = new LeaderboardAdapter(context: LeaderboardActivity.this,
                list, group: null);
        } else {
            list = getGroup(str: "group" + finalI);
            adapter = new LeaderboardAdapter(context: LeaderboardActivity.this,
                list, group: "group"+finalI);
        }
        listView.setAdapter(adapter);
    }
});
```

Rysunek 50. Sekwencja operacji uruchamiana po kliknięciu w obiekt w tablicy. Źródło: opracowanie własne

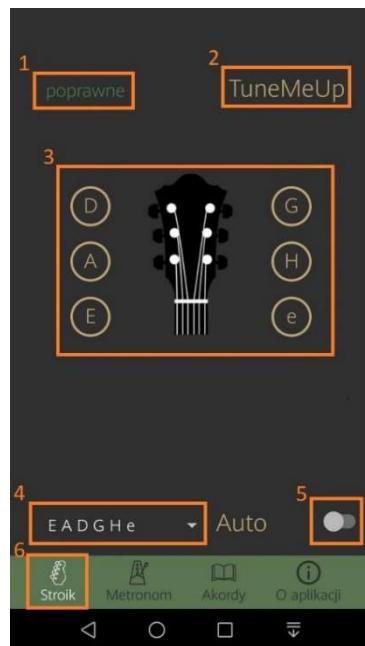
Aby wywołać taki adapter został zastosowany słuchacz akcji kliknięcia na nawigację rankingu. Po kliknięciu w którykolwiek element, obiekt adaptera zostaje nadpisany aby mogła zostać wyświetlona lista rankingowa dla odpowiedniej kategorii (rysunek 50). Do sortowania wyników zastosowany został algorytm bąbelkowy ukryty w metodach getAll oraz getGroup w zależności czy wyświetlane mają być dane o ogólnych punktach, czy w danej grupie określonej przez parametr metody getGroup. Gdy zostanie stworzony nowy adapter, stary jest przez niego nadpisywany, co powoduje odświeżenie widoku listy na ekranie.

## 6. Prezentacja systemu

Stworzona aplikacja posiada podział użytkowników na zalogowanych i niezalogowanych. Wynika to z ograniczenia dostępu do niektórych funkcjonalności aplikacji. W aplikacji są funkcjonalności, które są dostępne dla obu grup, przy czym użytkownik zalogowany otrzymuje dostęp do nowych funkcjonalności. Dla niezalogowanych przewidywane są okna, które oferują funkcjonalności takie jak stroik, metrom. Ponadto użytkownik ma dostęp do informacji o aplikacji, okna pomocy (dotyczące nauki akordów), okna ze spisem akordów i podziałem ich na grupy oraz okna umożliwiające zalogowanie i rejestracje nowego konta. Dla użytkowników zalogowanych odblokowywane są takie funkcje jak wyświetlanie profilu, zmiana nazwy i ikony użytkownika, wyświetlenie rankingu oraz nauka akordów w trybie nauki i rankingu. Można zauważyć, że góra nawigacja aplikacji różni się w zależności od rodzaju użytkownika.

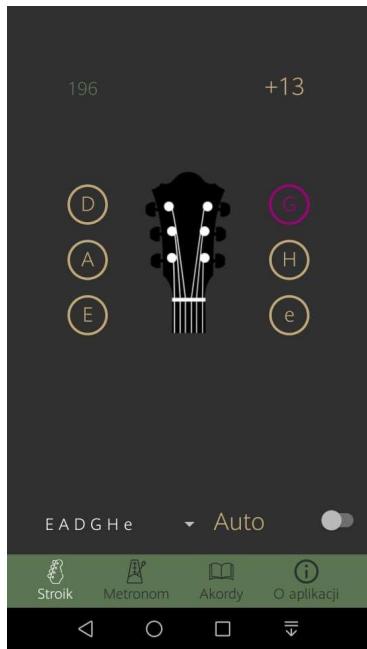
### 6.1. Obsługa aplikacji przez użytkownika niezalogowanego

Po uruchomieniu aplikacji, na ekranie ukazuje się okno oferujące funkcje stroika, które zostało ukazane na rysunku 51.



Rysunek 51. Okno stroika. Źródło: opracowanie własne

W punkcie 1 wpisana zostaje wymagana wartość dla odpowiedniej struny, a w punkcie 2 zostaje wpisana wartość odczytanej częstotliwości przez urządzenie. W obszarze 3 są stworzone imitacje strun, dla ułatwienia wizualnego została dodana główka gitary. Nie zależnie od tego czy struna została odczytana (rysunek 52), czy też wybrana przez użytkowania (rysunek 53) zostaje ona zaznaczona kolorem fioletowym.

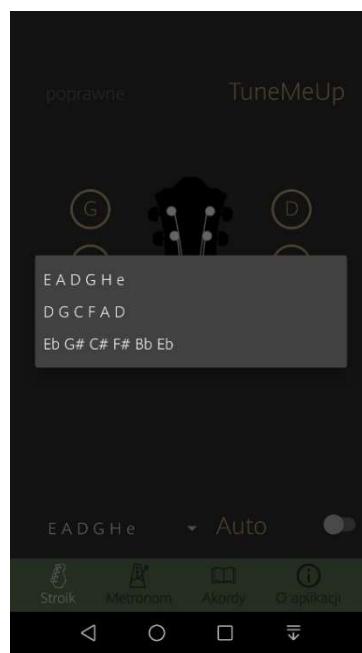


Rysunek 52. Zaznaczenie strojonej struny. Źródło: opracowanie własne



Rysunek 53. Zaznaczenie strojonej struny w trybie automatycznym. Źródło: opracowanie własne

Poniżej w punkcie 4 (rysunek 51) znajduje się lista z dostępnymi strojeniami. Po kliknięciu w nią na ekranie zostaje wyświetlona lista (rysunek 54).

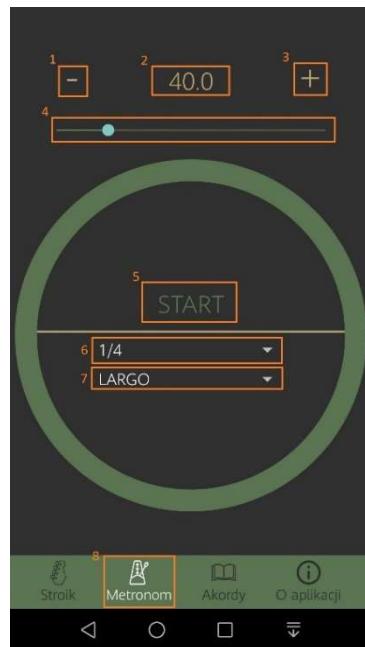


Rysunek 54. Lista dostępnych strojeń. Źródło: opracowanie własne

Po kliknięciu w interesujący element dane dla strojenia (wymagane częstotliwości oraz nazwy strun) zostają zmienione. Ostatnim elementem na ekranie jest punkt 5 (rys-

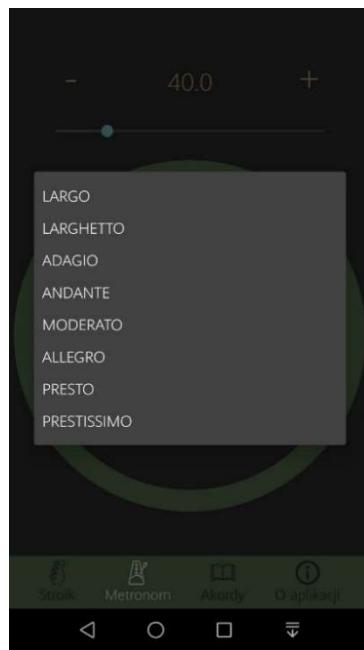
nek 51). Pod tym punktem znajduje się przełącznik strojenia automatycznego. Po zaznaczeniu tego przełącznika uruchamiany zostaje tryb strojenia wszystkich strun. Ten tryb wyłącza się gdy użytkownik kliknie w przełącznik jeszcze raz lub wybierze którąś ze strun. Do tego okna użytkownik może się dostać poprzez wybranie pola 6 widocznego na rysunku 51.

Przechodząc do kolejnego okna za pomocą nawigacji dolnej (na rysunku 55 punkt 8) na ekranie użytkownika pojawia się okno zaprezentowane na rysunku 55.



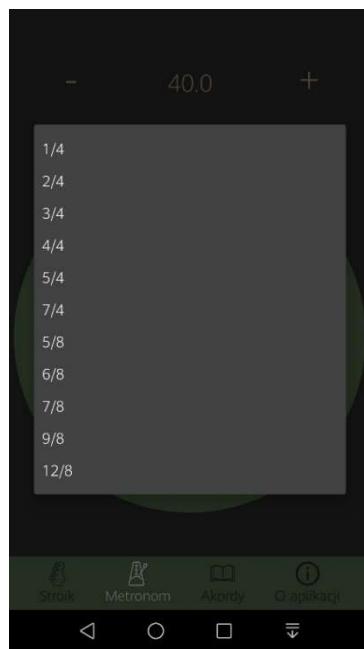
Rysunek 55. Okno metronomu. Źródło: opracowanie własne

W punkcie 1 i 3 znajdują się przyciski odpowiednio do zmniejszania i zwiększania wartości w polu 2, odpowiedzialne za aktualne tempo wybijane przez metronom. W polu 4 znajduje się suwak, który też manipuluje wartością w polu 2. Został dodany dla szybszego zmieniania wartości. Kolejnym polem do manipulowania wartością pola 2 jest lista rozwijana, która została opisana na rysunku 55 punktem 7. Po kliknięciu w to pole ukazuje się lista z nazwami różnych temp (rysunek 56). Wybór nazwy zamyka okienko z listą. W polu 7 znajduje się wybrana nazwa, a w polu 2 znajduje się minimalna wartość tempa przypisana do nazwy.



Rysunek 56. Lista dostępnych nazw temp. Źródło: opracowanie własne

Pole w punkcie 6 (rysunek 55) odpowiada za metrum metronomu i po kliknięciu włączona zostaje lista z dostępnymi metrami, która została zaprezentowana na rysunku 57. Po wyborze elementu z listy, ta zamyka się a w polu 6 na rysunku 55 zmienia się wartość na wybraną.



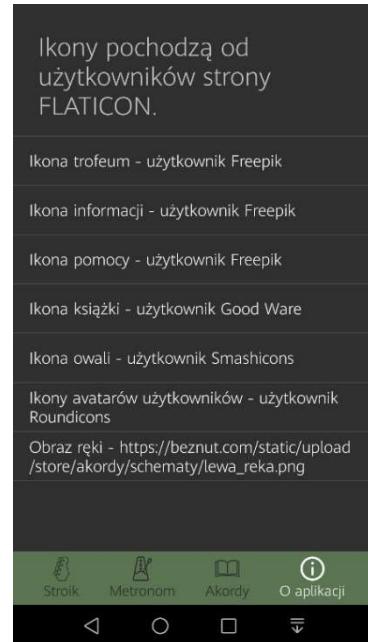
Rysunek 57. Lista dostępnych metrum. Źródło: opracowanie własne

Metronom można uruchomić poprzez kliknięcie w napis oznaczony numerem 5 (rysunek 55). W tym samym miejscu należy kliknąć jeśli użytkownik zechce wyłączyć działanie metronomu.

Pod ostatnim przyciskiem w nawigacji dolnej znajduje się dostęp do listy, w której są wypisane informacje o autorach grafik wykorzystanych w aplikacji (rysunek 58 i 59). Przycisk znajduje się w polu 1 na rysunku 58.



Rysunek 58. Okno zasobów zewnętrznych - pierwsza strona. Źródło: opracowanie własne



Rysunek 59. Okno zasobów zewnętrznych - druga strona. Źródło: opracowanie własne

Kolejnym oknem dostępnym dla użytkowników niezalogowanych jest ekran, ukaźany na rysunku 60, odpowiedzialny za wyświetlenie listy grup akordów. Do tego okna użytkownik może się dostać poprzez kliknięcie w pole 5.



Rysunek 60. Okno listy grup akordów - niezalogowani. Źródło: opracowanie własne

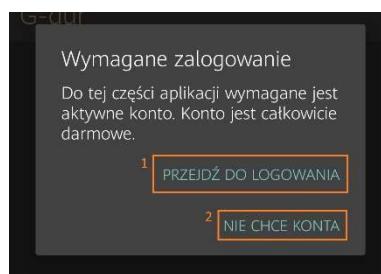
Dla uproszczenia i czytelności oznaczony został pierwszy element widocznej listy i tak w polu 3 widoczna jest nazwa grupy akordów a w punkcie 4 zawartość tej grupy. Pod punktem 1 znajduje się przekierowanie do logowania zaś w polu 2 do okna pomocy korzystania z funkcji nauki akordów.

Klikając w element w widocznej liście (rysunek 60 pole 3 i pole 4) zostaje otwarte okno z listą akordów w danej grupie, które zostało przedstawione na rysunku 61.



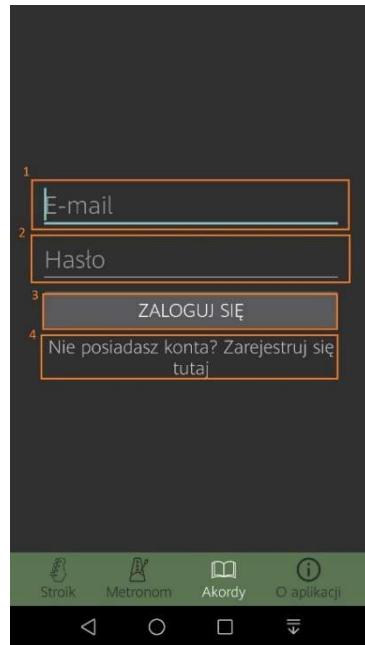
Rysunek 61. Lista akordów - niezalogowani. Źródło: opracowanie własne

W obszarze nawigacji górnej dodana została strzałka (pole 1), która przekierowuje do okna z listą grup akordów. Lista akordów składa się z identycznych elementów, tak jak w przypadku listy grup akordów oznaczony został tylko pierwszy element i tak w polu 2 wyświetlana jest nazwa akordu. Po kliknięciu w którykolwiek element w przypadku niezalogowanego użytkownika pojawi się stosowna informacja zawarta na rysunku 62. Pole 2 zamknie okno, a pole 1 przeniesie użytkownika do okna logowania.



Rysunek 62. Informacja dla niezalogowanych użytkowników. Źródło: opracowanie własne

Okno logowania zostało przedstawione na rysunku 63. W polu 1 i 2 użytkownik wpisuje odpowiednio adres email i hasło powiązane z kontem, a za pomocą przycisku oznaczonego numerem 3 dokonuje logowania. Jeśli proces powiedzie się to użytkownikowi zostanie wyświetlony jego profil. Aby użytkownik mógł przejść do rejestracji musi kliknąć w obszar zaznaczony numerem 4.



Rysunek 63. Okno logowania. Źródło: opracowanie własne

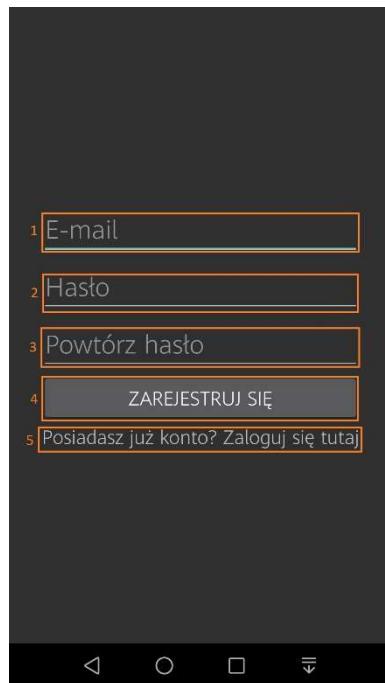
Gdy wpisane hasło dla tego adresu email jest niepoprawne użytkownik zostaje o tym poinformowany przez wyświetlenie komunikatu w dole ekranu (rysunek 64 pole 1).



Rysunek 64. Błąd logowania – niepoprawne dane. Źródło: opracowanie własne

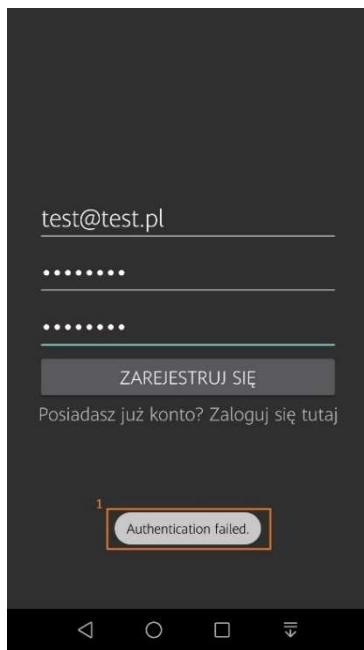
W oknie rejestracji przedstawionym na rysunku 65 w polu 1, użytkownik ma możliwość wpisania adresu email, a w polach 2 i 3 wpisuje hasło. Po kliknięciu w przycisk oznaczony numerem 4 dane zostają sprawdzone (czy email spełnia warunki oraz czy hasła są identyczne). Po pomyślnym procesie weryfikacji następuje rejestracja użytkownika. Jeśli operacja powiedzie się, użytkownik jest przekierowywany do swojego profilu.

Podczas kliknięcia w tekst (pole 5) poniżej przycisku, użytkownik zostanie przekierowany do okna logowania.

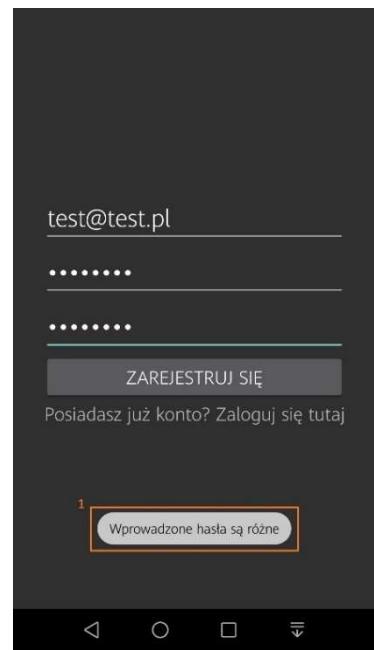


Rysunek 65. Okno rejestracji. Źródło: opracowanie własne

Podczas błędów rejestracji (np. adres email już istnieje lub połączenie z Internetem zostało przerwane) użytkownikowi ukaże się dymek na dole ekranu (rysunek 66 pole 1), jednak gdy użytkownik wpisze dwa różne hasła zostanie o tym poinformowany komunikatem na rysunku 67 pod polem 1.

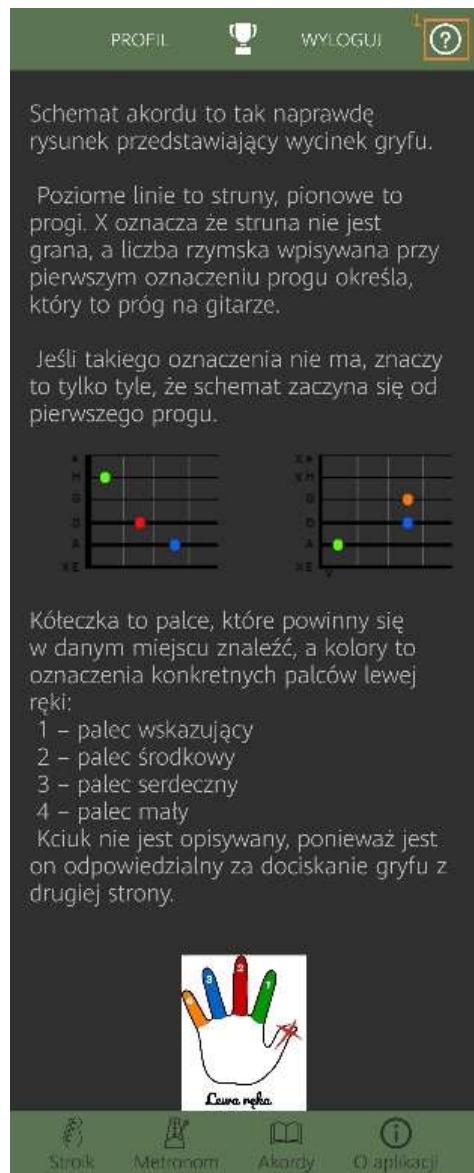


Rysunek 66. Błąd rejestracji – problem ze-wnętrzny, niepoprawny/zajęty adres email. Źró- dło: opracowanie własne



Rysunek 67. Błąd rejestracji – różne hasła. Źró- dło: opracowanie własne

Ostatnim dostępnym oknem dla użytkowników niezalogowanych jest okno pomocy (rysunek 68). Aby do niego przejść należy kliknąć w pole 1. Jest ono oznaczone jako znak zapytania wpisany w okrąg. W tym oknie znajdują się wskazówki jak korzystać z funkcji nauki akordów. Przedstawione są tam dwa przykładowe schematy akordów wraz z wyjaśnieniem oznaczeń oraz obrazek ręki pokazujący kolory poszczególnych palców. Te kolory są wykorzystywane do pokazywania akordów. Zrzut ekranu ukazany na zdjęciu poniżej jest zrobiony za pomocą funkcji zrzutu przewijanego dostępnego na urządzeniu Huawei P9 Lite 2018.



Rysunek 68. Okno pomocy. Źródło: opracowanie własne

## 6.2. Obsługa aplikacji przez użytkownika zalogowanego

Użytkownik zalogowany zdobywa dostęp do nowych funkcjonalności, dlatego też nawigacja góra aplikacji zmienia się. Teraz dostępne są dwa nowe przyciski. Jeden z nich pozwala na wejście do profilu użytkownika. Jest to przycisk w polu numer 1 na rysunku 69, natomiast w polu 2 na tym samym rysunku użytkownik może przejść do funkcji rankingów. Za pomocą pola 3 użytkownicy mogą się wylogować.

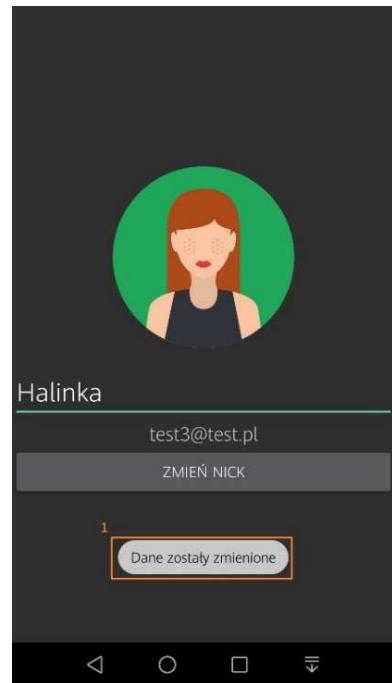


Rysunek 69. Nawigacja góra aplikacji - zalogowani. Źródło: opracowanie własne

Przechodząc do profilu ukazuje się ekran z danymi użytkownika (rysunek 70). W polu 1 znajduje się ikona użytkownika, w 2 aktualna nazwa użytkownika, a w polu 3 jest wyświetlony adres email powiązany z kontem. Poprzez zmianę tekstu w polu 2, a następnie w przycisk oznaczony numerem 4 użytkownik dokonuje zmiany nazwy użytkownika przez co zostaje powiadomiony przez dymek na dole aplikacji (rysunek 71 pole 1). Jest on taki sam jak w przypadku zmiany ikony użytkownika.



Rysunek 70. Okno profilu. Źródło: opracowanie własne



Rysunek 71. Okno profilu – informacja o zmianie danych.  
 Źródło: opracowanie własne

Aby użytkownik zmienił swoją ikonę musi przejść do okna z dostępnymi ikonami. Aby to zrobić wystarczy, że kliknie w obszar zaznaczony numerem 1 na rysunku 70, wtedy na ekranie pojawi się okno ukazane na rysunku 72. Dla uproszczenia i polepszenia czytelności tylko pierwsza ikona użytkownika została opisana. Jednak kliknięcie w którykolwiek ikonę spowoduje to samo działanie. W polu 2 znajduje się wspomniana ikona. Po kliknięciu w nią użytkownik zmienia swoją ikonę na klikniętą, następnie zostaje przeniesiony do okna profilu i zostaje wyświetlona specjalna informacja ujęta na rysunku 73 w polu 2. Na tym rysunku można zauważyć, że ikona użytkownika (pole 1 na rysunku 73) faktycznie się zmieniła. Na rysunku 72 zostaje nieomówione jeszcze jedno pole, jest to pole z numerem 1. Jest to przycisk anulujący proces zmiany ikony użytkownika. W tym przypadku użytkownik zostaje przeniesiony do okna logowania, bez informacji o zmianie ikony co jest potwierdzone w niezmienionej ikonie wyświetlnej w oknie profilu.

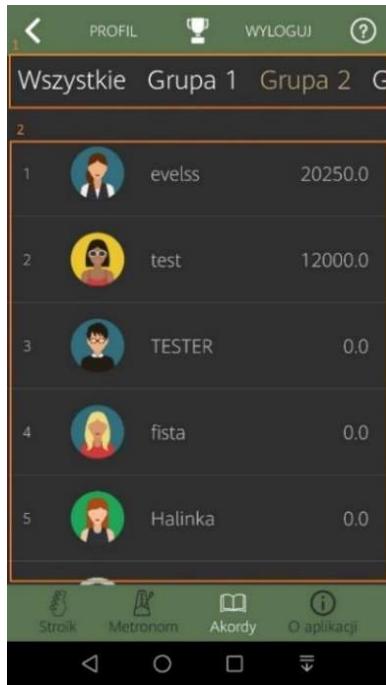


Rysunek 72. Okno zmiany ikony użytkownika. Źródło: opracowanie własne

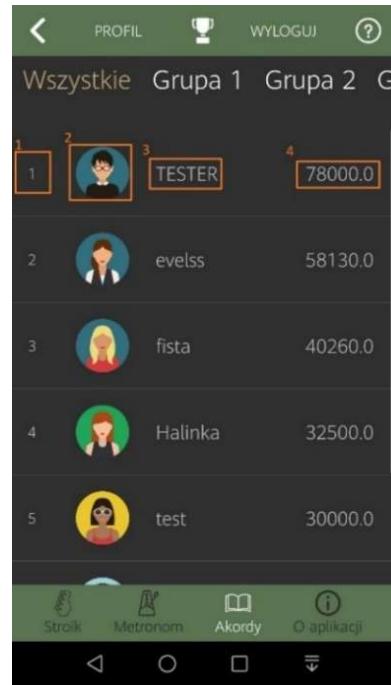


Rysunek 73. Okno profilu – informacja o zmianie danych. Źródło: opracowanie własne

Opisywane pole 2 na rysunku 69 przenosi użytkownika do okna ukazanego na rysunku 74. Jest to okno, które wyświetla listę rankingową w poszczególnych grupach. W polu 1 znajduje się lista przewijana w lewo i prawo dzięki, której może wybrać interesującą go grupę. W polu 2 znajduje się lista użytkowników posortowana ze względu na otrzymane punkty. Dla uproszczenia i zwiększenia czytelności na rysunku 75 znajdują się oznaczenia tylko dla pierwszego elementu w liście rankingowej i tak w polu 1 znajduje się zajęte miejsce, w 2 jest ikona danego użytkownika, a w polu 3 znajduje się jego nazwa. W ostatnim polu (tj. pole 4) jest ilość punktów zebranych przez użytkownika w danej grupie.

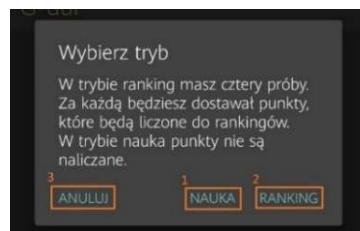


Rysunek 74. Okno rankingu. Źródło: opracowanie własne



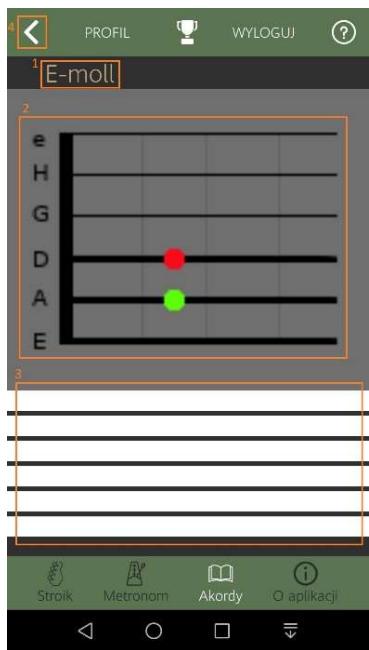
Rysunek 75. Okno rankingu – opis elementu listy. Źródło: opracowanie własne

Kolejną funkcjonalnością, do której użytkownik zalogowany otrzymuje dostęp jest nauka akordów w dwóch trybach. Pierwszy z nich to tryb nauki. Aby wejść w ten tryb użytkownik musi wybrać przycisk 1 na rysunku 76, do trybu rankingu użytkownik zostaje przeniesiony po kliknięciu w przycisk w polu 2, a w polu 3 znajduje się przycisk zamkajający okno ukazany na tym właśnie zdjęciu.

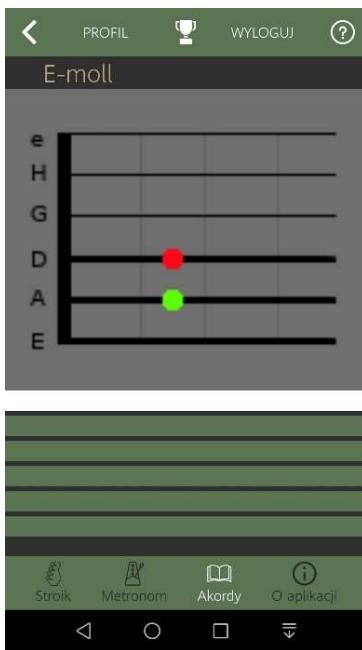


Rysunek 76. Okno wyboru trybu sprawdzania poprawności akordów. Źródło: opracowanie własne

W trybie nauki użytkownikowi wyświetla się takie okno jak na rysunku 77. W polu 1 znajduje się nazwa akordu, a w 2 schemat tego akordu. Poniżej w polu 3 znajdują się reprezentacje strun ułożone w taki sam sposób jak są ułożone na schemacie. Po poprawnym zagraniu danej struny kolor odpowiedniej zmienia się na zielony. Ukażane jest to na rysunku 78, gdzie wszystkie struny, oprócz pierwszej, zostały poprawnie zagrane. Gdy wszystkie struny zostaną oznaczone jako zagrane poprawne na ekranie pojawi się okno dialogowe z informacją o poprawnym zagraniu danego akordu. Zostało to ukażane na rysunku 79.



Rysunek 77. Okno nauki akordów – tryb nauki. Źródło: opracowanie własne



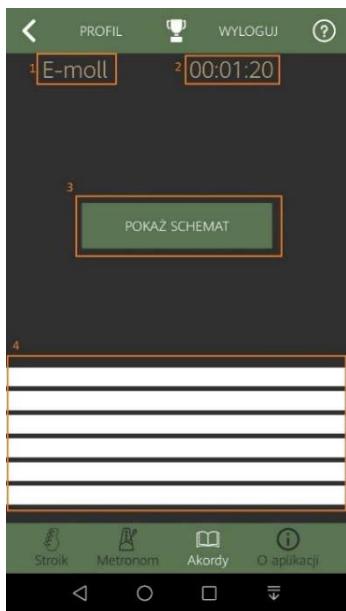
Rysunek 78. Zaznaczenie poprawnie zagranych strun. Źródło: opracowanie własne



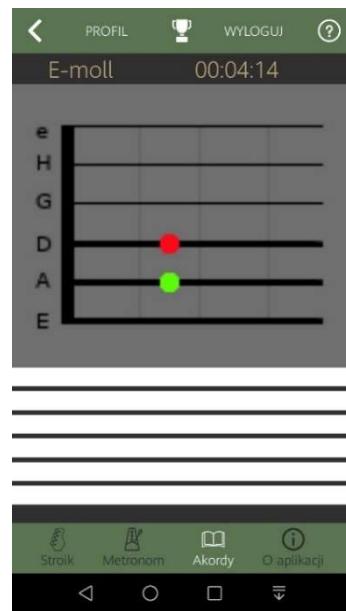
Rysunek 79. Informacja o poprawnym zagraniu akordu. Źródło: opracowanie własne

Po kliknięciu w przycisk OK użytkownik może powtórzyć ten sam akord, dopóki nie wyjdzie z danego okna poprzez kliknięcie w pole 4 oznaczone na rysunku 77.

Okno w trybie rankingu nieco różni się w wyglądzie, co widać na rysunku 80. Pole 1 pokazuje nazwę akordu, a w polu 4 widać reprezentacje strun. Schemat akordu został zastąpiony przez przycisk pokazany w polu 3. Po kliknięciu w przycisk, schemat akordu pokazuje się (rysunek 81), jednak użytkownik nie dostaje dodatkowych punktów. Nowym elementem jest w polu 2 licznik czasu. Jest on ważny ze względu na przyznawanie punktów (szczegółowe informacje zawarte są w tabeli 1 w rozdziale 5.6).

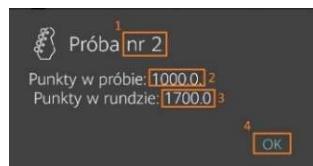


Rysunek 80. Okno nauki akordów – tryb rankingu. Źródło: opracowanie własne

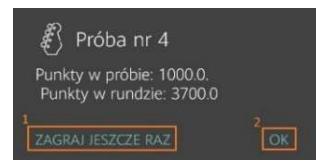


Rysunek 81. Okno nauki akordów – tryb rankingu ze schematem. Źródło: opracowanie własne

Po poprawnym zagraniu wszystkich strun użytkownikowi ukazuje się informacja w formie okna dialogowego (rysunek 82). W tym oknie są takie informacje jak numer próby (pole 1), liczba punktów zdobyta w próbie (pole 2) oraz podsumowanie wszystkich prób (pole 3). Aby przejść do kolejnej próby użytkownik musi kliknąć w pole 4. Po zagraniu wszystkich prób (tj. 4) użytkownikowi pojawi się okno dialogowe, które zostało pokazane na rysunku 83. Przycisk z pola 1 zapisuje do bazy daną rundę i rozpoczyna kolejną, a przycisk z pola 2 zapisuje do bazy rundę i przenosi użytkownika do listy akordów w grupie.



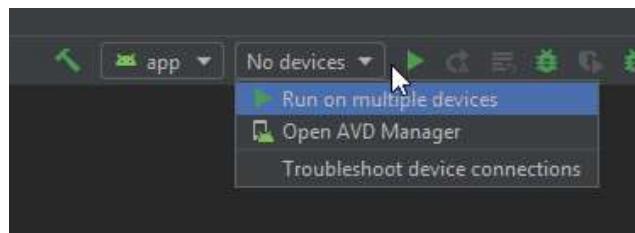
Rysunek 82. Okno dialogowe – zakończenie próby. Źródło: opracowanie własne



Rysunek 83. Okno dialogowe – zakończenie rundy. Źródło: opracowanie własne

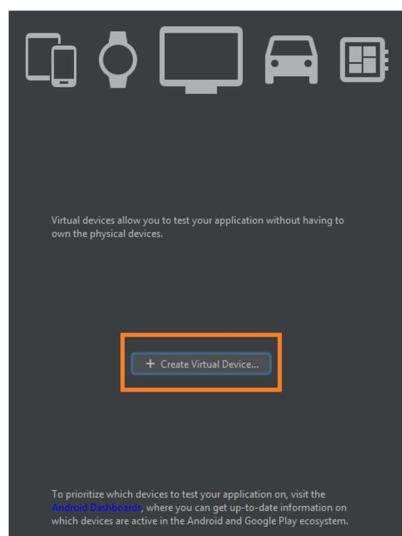
## 7. Uruchomienie systemu

Aplikację można uruchomić na dwa sposoby. Pierwszym sposobem jest uruchomienie aplikacji na emulatorze. Do tego celu potrzebne jest stworzenie emulatora w środowisku deweloperskim Android Studio.

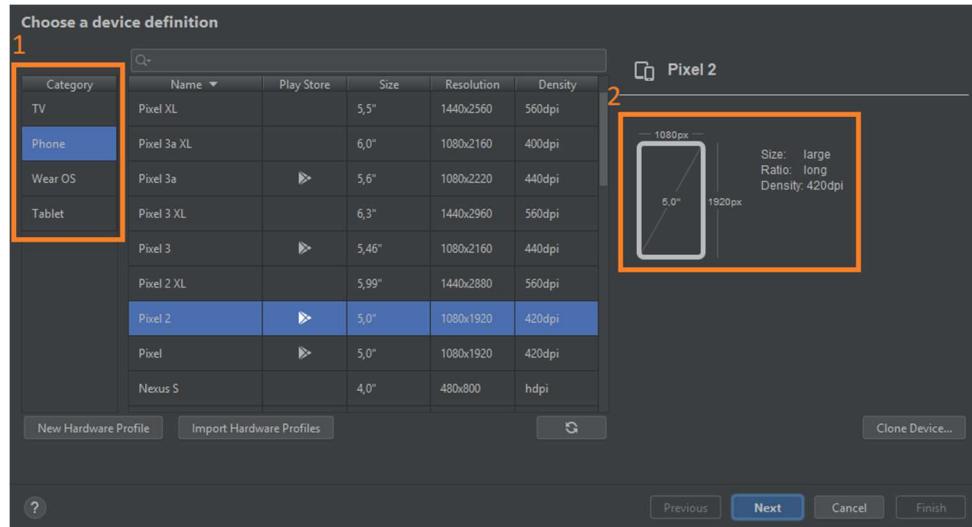


Rysunek 84. Lista rozwijana dostępnych urządzeń. Źródło: opracowanie własne

Aby utworzyć nowy emulator należy z listy rozwijanej ujętej na rysunku 84, która znajduje się w górnej części okna, wybrać opcję „Open AVD Manager”. Zostaje wyświetlone okno, które zawiera spis dostępnych emulatorów (AVD – ang. Android Virtual Device). Gdy lista jest pusta, tak jak w tym przypadku, wyświetlane zostaje takie okno jakie zostało pokazane na rysunku 85. Po kliknięciu przycisku w zaznaczonym polu, zostaje otwarte kolejne okno. W tym oknie zostaje udostępnione konfigurowanie nowego urządzenia.

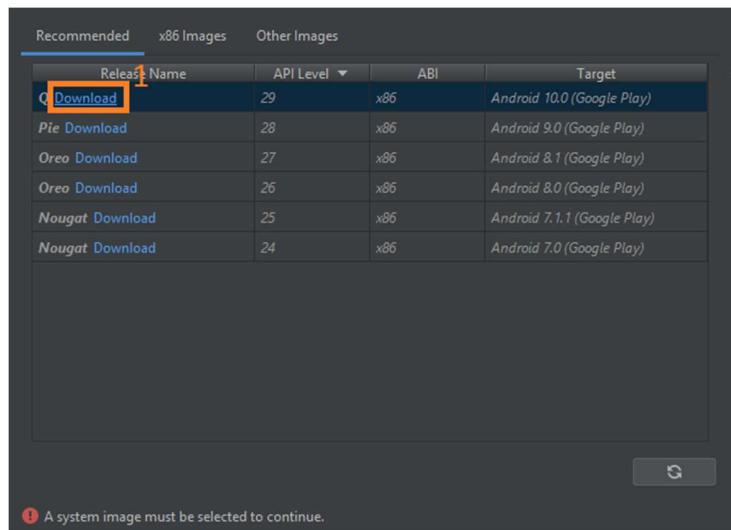


Rysunek 85. Okno wyświetlane w przypadku braku emulatorów. Źródło: opracowanie własne



Rysunek 86. Tworzenie emulatora - wybór telefonu. Źródło: opracowanie własne

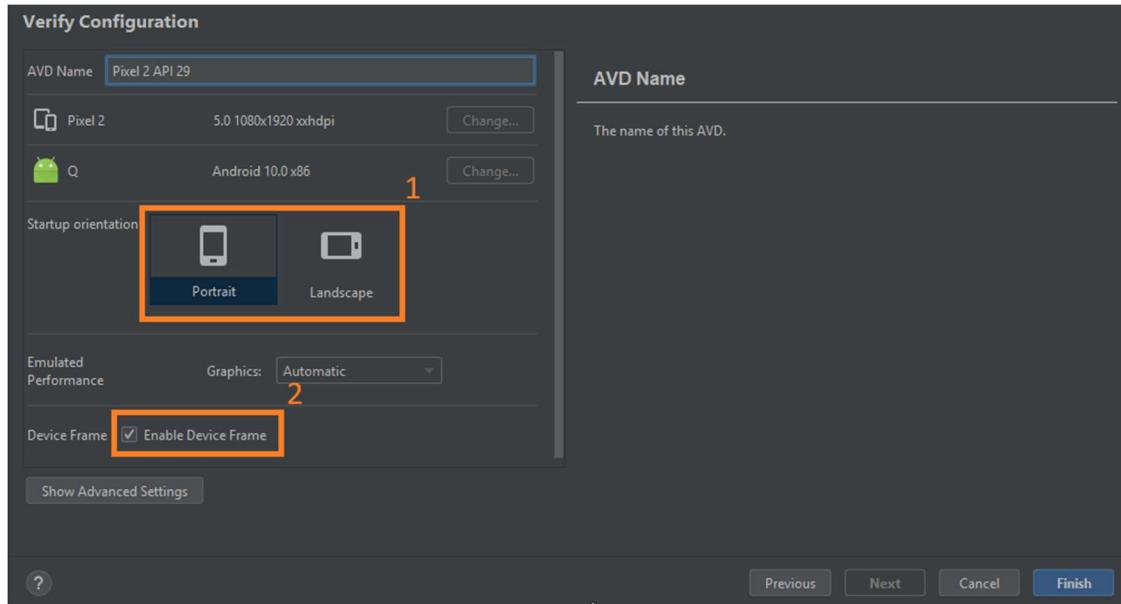
W pierwszym kroku została wyświetlona lista z dostępnymi urządzeniami (rysunek 86). Zostały one podzielone na 4 kategorie (pole 1). W kategorii drugiej znajdują się telefony, z której można wybrać różne modele telefonów wpierwanych przez firmę Google. Najważniejszą kwestią w wyborze emulatora są trzy ostatnie kolumny, odpowiedzialne za ekran. W polu 2 zostało przedstawione rozmieszczenie odpowiednich wartości.



Rysunek 87. Tworzenie emulatora - wybór oprogramowania. Źródło: opracowanie własne

Po wyborze odpowiedniego telefonu zostaje zmienione okno na okno z wyborem oprogramowania (rysunek 87). Przed utworzeniem pierwszego emulatora należy pobrać odpowiedni system. Warto zwrócić uwagę na drugą kolumnę. Zawiera ona API Level, które mają znaczenie podczas uruchomienia aplikacji na tym systemie. Jeśli będzie on niższy niż ten wpisany w pliku gradle w module app (obiekt minSdkVersion), aplikacja

nie uruchomi się. W tym projekcie minimalna wartość API Level to 21. Po zainstalowaniu (poprzez kliknięcie w pole 1) i zatwierdzeniu oprogramowania zostanie wyświetlane podsumowanie (rysunek 88) wraz z możliwością zmiany takich informacji jak z jakiej pozycji emulator ma zostać uruchomiony (pole 1) czy powinna zostać wyświetlona ramka telefonu (pole 2).



Rysunek 88. Tworzenie emulatora - podsumowanie. Źródło: opracowanie własne

Po zatwierdzeniu zostanie zamknięte okno, a w liście (rysunek 89) zostanie dodany emulator. Teraz można go uruchomić poprzez kliknięcie w pole 1 lub edytować jego dane pokazane w polu 2.

| Your Virtual Devices |                |            |                  |     |                  |         |              |         |
|----------------------|----------------|------------|------------------|-----|------------------|---------|--------------|---------|
| Type                 | Name           | Play Store | Resolution       | API | Target           | CPU/ABI | Size on Disk | Actions |
| Pixel                | Pixel 2 API 29 | ▶          | 1080 x 1920: ... | 29  | Android 10.0 ... | x86     | 513 MB       |         |
|                      |                |            |                  |     |                  |         |              |         |

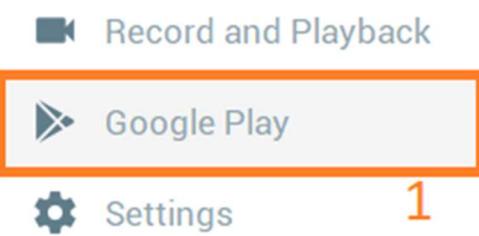
Rysunek 89. Lista emulatorów. Źródło: opracowanie własne

Po stworzeniu emulatora należy jeszcze zaktualizować usługę Google Play. Aby to zrobić należy w słupku po prawej stronie wybrać ikonę wielokropku (rysunek 90 pole 1).



Rysunek 90. Menu emulatora. Źródło: opracowanie własne

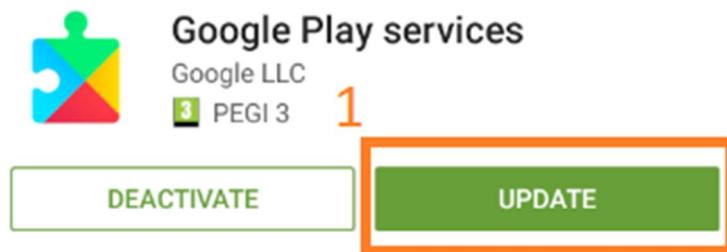
Zostanie wyświetlone okno, z którego po lewej stronie należy wybrać zakładkę oznaczoną polem 1 na rysunku 91, a następnie kliknąć w przycisk z rysunku 92, który znajduje się w drugiej części ekranu na samej górze. Po kliknięciu w ten przycisk na emulatorze pokaże się okno logowania do konta Google. Należy się zalogować, aby mieć dostęp do sklepu Play.



Rysunek 91. Zakładki w oknie ustawień emulatora.  
Źródło: opracowanie własne

Rysunek 92. Przycisk uruchamiający okno do aktualizacji usługi Google Play. Źródło: opracowanie własne

Po zalogowaniu zostaje wyświetlone okno z profilem usług Google Play (rysunek 93). Należy w tym momencie kliknąć w pole oznaczone 1. Po zaktualizowaniu aplikacja będzie działać z elementami potrzebującymi dostępu do Internetu (profil, ranking, nauka akordów).



Rysunek 93. Pole do zarządzania aplikacją. Źródło [18]

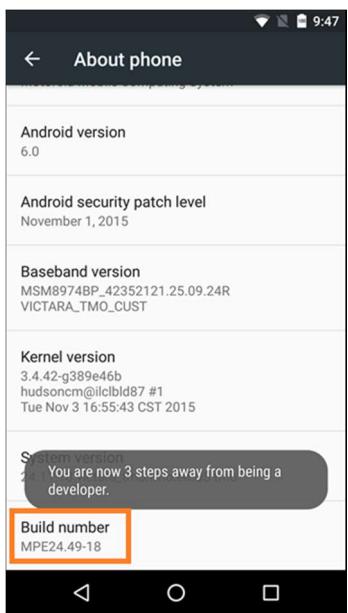
Drugim sposobem na uruchomienie aplikacji jest wykorzystanie urządzenia rzeczywistego opartego na systemie Android. Do tego celu potrzebne będzie pobranie ADB (ang. Android Debug Bridge). W nowszych wersjach Android SDK jest ono dodawane automatycznie. Jeśli ścieżka nie została zmieniona to plik adb.exe znajduje się w ścieżce ukazanej na rysunku 94.

```
C:\Users\marcin\AppData\Local\Android\Sdk\platform-tools
```

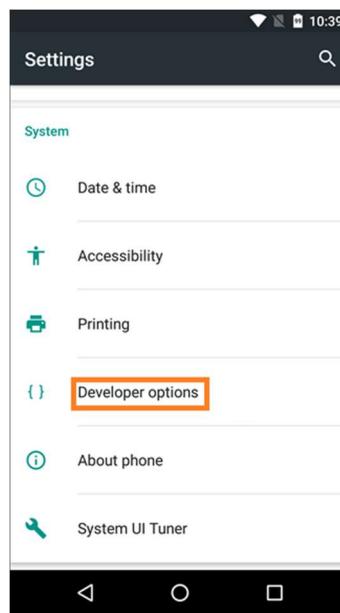
Rysunek 94. Ścieżka do pakietu ADB. Źródło: opracowanie własne

Jeśli w tym miejscu nie ma wspomnianego ADB należy pobrać i rozpakować ten pakiet. Gdy już zostanie to zrobione należy pobrać i zainstalować odpowiedni sterownik urządzenia rzeczywistego dla komputera.

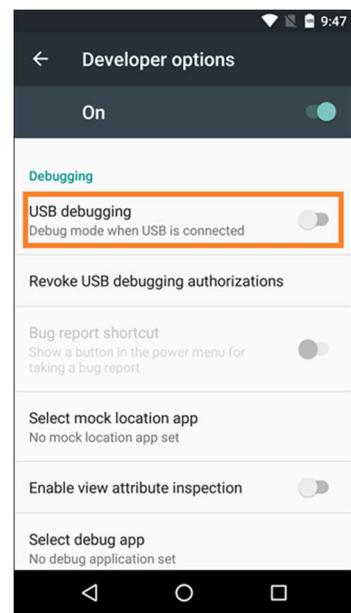
Kolejnym krokiem jest włączenie trybu debugowania na urządzeniu. Aby to zrobić trzeba najpierw włączyć opcje programistyczne. Dla każdej wersji Androida ten krok nieco się różni. Jednak najpopularniejszym sposobem jest wejście w ustawienia, a następnie wyszukanie informacji o telefonie i kolejno informacji o oprogramowaniu. Następnie należy znaleźć i kliknąć w numer wersji siedem razy. Odliczenie kliknięć pojawi się w dole ekranu. Teraz użytkownik telefonu ma dostęp do opcji programistycznych i powinien odnaleźć opcję debugowanie USB.



Rysunek 95. Włączanie opcji programistycznych. Źródło [19]



Rysunek 96. Opcje programistyczne w ustawieniach. Źródło [19]



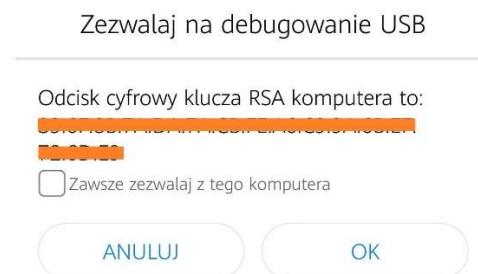
Rysunek 97. Włączenie trybu debugowania. Źródło [19]

Po włączeniu opcji debugowania należy podłączyć telefon kablem USB do komputera, a następnie kliknąć w zielony trójkąt (rysunek 98).



Rysunek 98. Uruchomienie aplikacji na wybranym urządzeniu. Źródło: opracowanie własne

Na ekranie telefonu pojawi się taka informacja jak na rysunku 99. Należy zatwierdzić, a środowisko programistyczne przejdzie do budowania aplikacji oraz instalacji jej na telefonie. Po pozytywnym przebiegu całego procesu aplikacja od razu uruchomi się na telefonie.



Rysunek 99. Pytanie o pozwolenie na debugowanie na urządzeniu. Źródło: opracowanie własne

Po uruchomieniu projektu w środowisku Android Studio może wystąpić błąd, który jest spowodowany brakiem Android SDK. Środowisko poprosi o jego instalację, a nawet udostępnii przycisk przejścia do okna z instalacją brakujących elementów.

```
ERROR: Failed to install the following Android SDK packages as some licences have not been accepted.  
    build-tools;29.0.1 Android SDK Build-Tools 29.0.1  
To build this project, accept the SDK license agreements and install the missing components using the Android Studio SDK Manager.  
Alternatively, to transfer the license agreements from one workstation to another, see http://d.android.com/r/studio-ui/export-licenses.html  
  
Using Android SDK: C:\Users\maxto\AppData\Local\Android\Sdk  
Install missing SDK package(s).
```

Rysunek 100. Błąd spowodowany brakiem SDK. Źródło: opracowanie własne

## **8. Podsumowanie**

Celem pracy było stworzenie aplikacji, która w znacznym stopniu ułatwi naukę gry na gitarze. Dzięki ocenie użytkowników testujących aplikacje można stwierdzić, że cel ten został osiągnięty. Aplikacja została przetestowana na telefonach: FLOW 5 Krugger&Matz, Huawei P9 Lite 2018, Huawei Mate 10 lite, Evolveo StrongPhone G4, LG G4c oraz na emulacji takich telefonów jak: Pixel 2, Pixel 3 oraz Pixel XL, przy czym funkcje bazujące na dźwiękach z otoczenia nie zostały sprawdzone na emulatorach. Wynika to z tego, że odbierany dźwięk przez emulacje telefonu jest zestawem wygenerowanych sygnałów o często jednej częstotliwości.

Większość użytkowników stwierdziła, że aplikacja spełnia przedstawione wymagania oraz określiła przejrzystość aplikacji na bardzo dobry. Aplikacja została stworzona tak, by w łatwy sposób dodać nowe funkcjonalności.

Aplikacje można rozwinąć poprzez poprawę działania funkcji sprawdzania poprawności akordów. Sprawdzanie mogłoby się odbywać poprzez szarpięcie wszystkich strun na raz oraz wskazanie, która ze strun jest zagrana nieprawidłowo. W tym momencie sprawdzanie akordów odbywa się poprzez zagranie struny po strunie. Kolejną możliwością rozwoju aplikacji jest poszerzenie bazy akordów oraz bazy dostępnych strojeń gitary. W przyszłości możliwe by było też ustawnienie przez użytkownika własnych niestandardowych strojeń przez zmianę częstotliwości na poszczególnych strunach.

## 9. Literatura

- [1] Oficjalna strona Android Studio <https://developer.android.com/studio/index.html> (dostęp 18.12.2019r.)
- [2] Lemay L., Perkins C.L.: Teach Yourself JAVA in 21 Days. Sams.net, 1th edition, 1996.
- [3] Wstęp do Gradle <https://www.samouczekprogramisty.pl/wstep-do-gradle/> (dostęp 22.08.2019r.)
- [4] Gradle – Mocarne narzędzie do budowy projektów [http://www.javaexpress.pl/article/show/Gradle\\_Mocarne\\_narzedzie\\_do\\_budowy\\_projektow?lang=pl](http://www.javaexpress.pl/article/show/Gradle_Mocarne_narzedzie_do_budowy_projektow?lang=pl) (dostęp 09.12.2009r.)
- [5] How to Install and Use ADB, the Android Debug Bridge Utility <https://www.howtogeek.com/125769/how-to-install-and-use-abd-the-android-debug-bridge-utility/> (dostęp 20.06.2017r.)
- [6] Wikipedia.org – Firebase <https://en.wikipedia.org/wiki/Firebase> (dostęp 20.12.2019r.)
- [7] Oficjalna strona GIMP <https://www.gimp.org/> (dostęp 31.10.2019r.)
- [8] Wikipedia.org – GIMP <https://en.wikipedia.org/wiki/GIMP> (dostęp 21.12.2019r. )
- [9] Repozytorium GitHub – JorenSix/TarsosDSP <https://github.com/JorenSix/TarsosDSP> (dostęp 21.10.2019r.)
- [10] <https://absta.pl/rozdzia-akustyka-4-historia-akustyki-i-elektronomikrostyki-4.html?page=4> (dostęp 30.12.2019r.)
- [11] <http://zasoby.open.agh.edu.pl/~10swlabaj/fourier/algorytmfft.html> (dostęp 30.12.2019r.)
- [12] Lyons R.G., „Wprowadzenie do cyfrowego przetwarzania sygnałów”, WKiŁ, Warszawa 1999
- [13] Wikipedia.org – Fast Fourier Transform [https://en.wikipedia.org/wiki/Fast\\_Fourier\\_transform](https://en.wikipedia.org/wiki/Fast_Fourier_transform) (dostęp 30.12.2019r.)
- [14] Oficjalna strona Picasso <https://square.github.io/picasso/> (dostęp 05.12.2019r.)
- [15] Repozytorium GitHub – square/picasso <https://github.com/square/picasso> (dostęp 15.12.2019r.)
- [16] <https://www.journaldev.com/13759/android-picasso-tutorial> (dostęp 9.03.2019r.)
- [17] Different Hz Tunings for a Guitar <https://ourpastimes.com/different-hz-tunings-for-a-guitar-12489422.html> (dostęp 15.09.2017r.)

- [18] Google Play Services – profil na platformie Google Play  
<https://play.google.com/store/apps/details?id=com.google.android.gms&hl=en>  
(dostęp 17.01.2020r.)
- [19] How to Access Developer Options and Enable USB Debugging on Android  
<https://www.howtogeek.com/129728/how-to-access-the-developer-options-menu-and-enable-usb-debugging-on-android-4.2/> (dostęp 3.07.2017r.)

## **Wykaz rysункów**

|   |    |
|---|----|
| Rysunek 1. Dodanie biblioteki z plików aplikacji. Źródło: opracowanie własne.....                           | 15 |
| Rysunek 2. Dodanie zdjęcia w danym elemencie za pomocą biblioteki Picasso. Źródło: opracowanie własne ..... | 17 |
| Rysunek 3. Dodanie biblioteki za pomocą zależności. Źródło: opracowanie własne.....                         | 17 |
| Rysunek 4. Diagram przypadków użycia dla funkcjonalności strojenia gitary. Źródło: opracowanie własne ..... | 20 |
| Rysunek 5. Diagram sekwencji – strojenie pojedynczej struny. Źródło: opracowanie własne.....                | 21 |
| Rysunek 6. Diagram sekwencji – strojenie wszystkich strun jednocześnie. Źródło: opracowanie własne .....    | 21 |
| Rysunek 7. Diagram klas – funkcjonalność strojenia gitary. Źródło: opracowanie własne .....                 | 22 |
| Rysunek 8. Diagram przypadków użycia dla funkcjonalności metronomu. Źródło: opracowanie własne .....        | 23 |
| Rysunek 9. Diagram sekwencji – działanie metronomu. Źródło: opracowanie własne .                            | 24 |
| Rysunek 10. Diagram klas – funkcjonalność strojka. Źródło: opracowanie własne.....                          | 24 |
| Rysunek 11. Diagram przypadków użycia – profil użytkownika. Źródło: opracowanie własne.....                 | 25 |
| Rysunek 12. Diagram sekwencji – rejestracja. Źródło: opracowanie własne .....                               | 25 |
| Rysunek 13. Diagram sekwencji – logowanie. Źródło: opracowanie własne.....                                  | 26 |
| Rysunek 14. Diagram sekwencji – zmiana nazwy użytkownika. Źródło: opracowanie własne.....                   | 27 |
| Rysunek 15. Diagram sekwencji – zmiana ikony użytkownika. Źródło: opracowanie własne .....                  | 27 |
| Rysunek 16. Diagram klas – funkcjonalność profilu. Źródło: opracowanie własne.....                          | 28 |
| Rysunek 17. Diagram przypadków użycia – nauka akordów. Źródło: opracowanie własne .....                     | 29 |
| Rysunek 18. Diagram sekwencji – poprawność akordów w trybie nauki. Źródło: opracowanie własne .....         | 30 |
| Rysunek 19. Diagram sekwencji – poprawność akordów w trybie rankingu. Źródło: opracowanie własne .....      | 31 |

|  |    |
|--|----|
| Rysunek 20. Diagram klas – funkcjonalność sprawdzania poprawności akordów. Źródło: opracowanie własne .....  | 33 |
| Rysunek 21. Diagram sekwencji – wyświetlanie listy rankingowej. Źródło: opracowanie własne .....   | 34 |
| Rysunek 22. Diagram klas – funkcjonalność list rankingowych. Źródło: opracowanie własne .....  | 35 |
| Rysunek 23. Kompozycja folderów w aplikacji. Źródło: opracowanie własne .....  | 37 |
| Rysunek 24. Kod otwierający nowy wątek sprawdzenia częstotliwości pobranej przez mikrofon urządzenia. Źródło: opracowanie własne.....                          | 39 |
| Rysunek 25. Funkcja zwracająca tablice z nazwą struny, różnice z podaną w parametrze oraz wymaganą częstotliwość. Źródło: opracowanie własne .....             | 40 |
| Rysunek 26. Poprawne uruchomienie przerwanego i nowego wątku. Źródło: opracowanie własne .....   | 41 |
| Rysunek 27. Kod wątku metronomu. Źródło: opracowanie własne .....  | 41 |
| Rysunek 28. Funkcje metronomu. Od góry ustalenie tempa, ustalenie metrum oraz funkcja uruchamiająca właściwy sygnał dźwiękowy. Źródło: opracowanie własne..... | 42 |
| Rysunek 29. Przykładowy dokument w kolekcji chords. Źródło: opracowanie własne   | 43 |
| Rysunek 30. Przykładowy dokument w kolekcji users. Źródło: opracowanie własne... 43  |    |
| Rysunek 31. Przykładowy dokument w kolekcji leaderboard. Źródło: opracowanie własne .....  | 44 |
| Rysunek 32. Przypisanie do zmiennej db instancji bazy danych. Źródło: opracowanie własne .....   | 44 |
| Rysunek 33. Utworzenie obiektu mapy. Źródło: opracowanie własne .....  | 45 |
| Rysunek 34. Dodanie do obiektu mapy obiekt typu klucz wartość. Źródło: opracowanie własne .....  | 45 |
| Rysunek 35. Dodanie zmian do kolekcji users pod danym UID. Źródło: opracowanie własne .....  | 45 |
| Rysunek 36. Metoda wykonywana po poprawnym pobraniu kolekcji. Źródło: opracowanie własne .....   | 45 |
| Rysunek 37. Wyszukanie dokumentu z danym kluczem. Źródło: opracowanie własne 46  |    |
| Rysunek 38. Metody wywoywane na obiekcie firebaseAuth do utworzenia nowego użytkownika. Źródło: opracowanie własne .....                                       | 46 |
| Rysunek 39. Wyświetlanie danych o użytkowniku. Źródło: opracowanie własne ..... 47   |    |

|   |    |
|---|----|
| Rysunek 40. Sekwencja operacji po kliknięciu w obiekt updateData. Źródło: opracowanie własne.....   | 47 |
| Rysunek 41. Metoda zmieniająca odpowiednie dane w kolekcji users dla zalogowanego użytkownika. Źródło: opracowanie własne.....  | 48 |
| Rysunek 42. Sekwencja operacji po kliknięciu w obiekt avatar. Źródło: opracowanie własne.....   | 48 |
| Rysunek 43. Wskazanie pliku XML wyglądu okna oraz dodanie do listy obiekt typu IcoUser. Źródło: opracowanie własne.....   | 49 |
| Rysunek 44. Sekwencja operacji pozwalająca na poprawne wyświetlanie dostępnych ikon oraz wywołujące zmiany dla zalogowanego użytkownika. Źródło: opracowanie własne ..... | 49 |
| Rysunek 45. Metoda pobierająca z kolekcji chords odpowiedni akord oraz pobranie danych statystycznych użytkownika. Źródło: opracowanie własne.....                        | 50 |
| Rysunek 46. Metoda zwracająca obiekt Chord z wymaganymi częstotliwościami na danych strunach oraz schematem akordu. Źródło: opracowanie własne .....                      | 51 |
| Rysunek 47. Sekwencja operacji łącząca wygląd poszczególnych elementów w liście. Źródło: opracowanie własne .....   | 53 |
| Rysunek 48. Łączenie obiektów JAVA z obiektami wyświetlającymi się na ekranie. Źródło: opracowanie własne .....   | 53 |
| Rysunek 49. Sekwencja operacji umożliwiająca wyświetlenie obrazu oraz wyświetlić nazwę użytkownika. Źródło: opracowanie własne .....                                      | 53 |
| Rysunek 50. Sekwencja operacji uruchamiana po kliknięciu w obiekt w tablicy. Źródło: opracowanie własne .....   | 54 |
| Rysunek 51. Okno stroika. Źródło: opracowanie własne.....   | 55 |
| Rysunek 52. Zaznaczenie strojonej struny. Źródło: opracowanie własne .....  | 56 |
| Rysunek 53. Zaznaczenie strojonej struny w trybie automatycznym. Źródło: opracowanie własne.....  | 56 |
| Rysunek 54. Lista dostępnych strojeń. Źródło: opracowanie własne .....  | 56 |
| Rysunek 55. Okno metronomu. Źródło: opracowanie własne .....  | 57 |
| Rysunek 56. Lista dostępnych nazw temp. Źródło: opracowanie własne.....   | 58 |
| Rysunek 57. Lista dostępnych metrum. Źródło: opracowanie własne.....  | 58 |
| Rysunek 58. Okno zasobów zewnętrznych - pierwsza strona. Źródło: opracowanie własne .....   | 59 |

|   |    |
|---|----|
| Rysunek 59. Okno zasobów zewnętrznych - druga strona. Źródło: opracowanie własne .....                              | 59 |
| Rysunek 60. Okno listy grup akordów - niezalogowani. Źródło: opracowanie własne .                                   | 59 |
| Rysunek 61. Lista akordów - niezalogowani. Źródło: opracowanie własne .....   | 60 |
| Rysunek 62. Informacja dla niezalogowanych użytkowników. Źródło: opracowanie własne .....                           | 60 |
| Rysunek 63. Okno logowania. Źródło: opracowanie własne .....  | 61 |
| Rysunek 64. Błąd logowania – niepoprawne dane. Źródło: opracowanie własne .....                                     | 61 |
| Rysunek 65. Okno rejestracji. Źródło: opracowanie własne .....  | 62 |
| Rysunek 66. Błąd rejestracji – problem zewnętrzny, niepoprawny/zajęty adres email. Źródło: opracowanie własne ..... | 63 |
| Rysunek 67. Błąd rejestracji – różne hasła. Źródło: opracowanie własne .....  | 63 |
| Rysunek 68. Okno pomocy. Źródło: opracowanie własne .....   | 64 |
| Rysunek 69. Nawigacja górną aplikacji - zalogowani. Źródło: opracowanie własne ....                                 | 64 |
| Rysunek 70. Okno profilu. Źródło: opracowanie własne .....  | 65 |
| Rysunek 71. Okno profilu – informacja o zmianie danych. Źródło: opracowanie własne .....                            | 65 |
| Rysunek 72. Okno zmiany ikony użytkownika. Źródło: opracowanie własne .....   | 66 |
| Rysunek 73. Okno profilu – informacja o zmianie danych. Źródło: opracowanie własne .....                            | 66 |
| Rysunek 74. Okno rankingu. Źródło: opracowanie własne.....  | 67 |
| Rysunek 75. Okno rankingu – opis elementu listy. Źródło: opracowanie własne.....                                    | 67 |
| Rysunek 76. Okno wyboru trybu sprawdzania poprawności akordów. Źródło: opracowanie własne .....                     | 67 |
| Rysunek 77. Okno nauki akordów – tryb nauki. Źródło: opracowanie własne.....  | 68 |
| Rysunek 78. Zaznaczenie poprawnie zagranych strun. Źródło: opracowanie własne....                                   | 68 |
| Rysunek 79. Informacja o poprawnym zagraniu akordu. Źródło: opracowanie własne 68                                   |    |
| Rysunek 80. Okno nauki akordów – tryb rankingu. Źródło: opracowanie własne .....                                    | 69 |
| Rysunek 81. Okno nauki akordów – tryb rankingu ze schematem. Źródło: opracowanie własne .....                       | 69 |
| Rysunek 82. Okno dialogowe – zakończenie próby. Źródło: opracowanie własne .....                                    | 69 |
| Rysunek 83. Okno dialogowe – zakończenie rundy. Źródło: opracowanie własne .....                                    | 69 |
| Rysunek 84. Lista rozwijana dostępnych urządzeń. Źródło: opracowanie własne .....                                   | 70 |

|  |    |
|--|----|
| Rysunek 85. Okno wyświetlane w przypadku braku emulatorów. Źródło: opracowanie własne.....                   | 70 |
| Rysunek 86. Tworzenie emulatora - wybór telefonu. Źródło: opracowanie własne.....                            | 71 |
| Rysunek 87. Tworzenie emulatora - wybór oprogramowania. Źródło: opracowanie własne.....                      | 71 |
| Rysunek 88. Tworzenie emulatora - podsumowanie. Źródło: opracowanie własne.....                              | 72 |
| Rysunek 89. Lista emulatorów. Źródło: opracowanie własne .....   | 72 |
| Rysunek 90. Menu emulatora. Źródło: opracowanie własne .....   | 73 |
| Rysunek 91. Zakładki w oknie ustawień emulatora. Źródło: opracowanie własne .....                            | 73 |
| Rysunek 92. Przycisk uruchamiający okno do aktualizacji usługi Google Play. Źródło: opracowanie własne ..... | 73 |
| Rysunek 93. Pole do zarządzania aplikacją. Źródło [18] .....   | 74 |
| Rysunek 94. Ścieżka do pakietu ADB. Źródło: opracowanie własne.....  | 74 |
| Rysunek 95. Włączanie opcji programistycznych. Źródło [19].....  | 75 |
| Rysunek 96. Opcje programistyczne w ustawieniach. Źródło [19] .....  | 75 |
| Rysunek 97. Włączenie trybu debugowania. Źródło [19].....  | 75 |
| Rysunek 98. Uruchomienie aplikacji na wybranym urządzeniu. Źródło: opracowanie własne.....                   | 75 |
| Rysunek 99. Pytanie o pozwolenie na debugowanie na urządzeniu. Źródło: opracowanie własne.....               | 75 |
| Rysunek 100. Błąd spowodowany brakiem SDK. Źródło: opracowanie własne.....                                   | 76 |

## **Streszczenie pracy**

Niniejsza praca dotyczy sposobu stworzenia aplikacji pozwalającej na nastrojenie gitary, naukę akordów, użycie metronomu oraz rywalizacje pomiędzy zalogowanymi użytkownikami. Aplikacja została napisana na telefony typu smartphone oparte na systemie Android. W pracy zostały opisane narzędzia oraz użyte technologie wykorzystane podczas tworzenia aplikacji. Dla lepszego zrozumienia zamysłu aplikacji zostały dodane diagramy UML (diagram klas, diagram przypadków użycia oraz diagram sekwencji). W dalszej części pracy została opisana implementacja ważniejszych funkcji oraz przedstawienie systemu na telefonie komórkowym. Następnie został opisany sposób uruchomienia aplikacji na emulatorze oraz na urządzeniu rzeczywistym.

Słowa kluczowe: aplikacja mobilna, gitara, metronom, metrum, tempo, akordy, ranking, TarsosDSP, Picasso, Java, Android, Firebase, Android Studio

## **Zawartość płyty**

Na płycie znajduje się elektroniczna wersja pracy w formacie DOCX oraz PDF oraz spakowana aplikacja w formacie 7z. Aplikacja powinna być otwarta, po rozpakowaniu, za pomocą środowiska Android Studio.