

Aplikacja - test do rozpoznawania płci mózgu wykonana z wykorzystaniem Android Studio

Michał Ankiersztajn

1 Wstęp

Projekt ma na celu stworzenie na Androida API 21+ aplikacji służącej do wykazywania kobiecości lub męskości mózgu użytkownika. Męskość lub kobiecość mózgu będzie określana na podstawie punktów otrzymywanych za zaznaczone lub niezaznaczone odpowiedzi na udzielone pytania oraz płci użytkownika. W zależności od uzyskanego wyniku użytkownik otrzyma informację zwrotną dotyczącą tego jakiego typu posiada mózg. Wynik wraz z opisem będzie można udostępnić poprzez jakąkolwiek posiadaną przez użytkownika aplikację, która posiada taką możliwość.

2 Problemy do rozwiązania

2.1 Zmienny system wag

Gdy ktoś wybiera płeć męską/żeńską waga przyznawanych punktów za odpowiedź zmienia się.

2.2 Przechowywanie pamięci podręcznej zaznaczonych odpowiedzi

problem związany z systemem Android, najczęściej przy przeciążeniu systemu lub obrocie ekranu niszczone jest Activity wraz ze wszystkimi aktualnie zapisanymi informacjami.

2.3 Aplikacja powinna wyglądać ładnie i mieć intuicyjny UX

W tym celu skorzystam ze stylów i motywów zgodnie z wskazówkami z Material Design.

2.4 Zmienność kolorów w zależności od pory dnia

W środku dnia UI powinno mieć jaskrawe kolory, a w wieczorem i w nocy ciemniejsze. Użytkownik nie powinnien wysilać wzroku korzystając z aplikacji. [Materiał na ten temat z Material Designu](#)

2.5 Dostęp do funkcjonalności w zależności od wersji Androida

Przy niższych API Androida niektóre funkcje nie działają i często trzeba tworzyć funkcje ze wsteczną kompatybilnością. Istnieje wiele bibliotek, które to robią.

2.6 Możliwość zapisu swoich rezultatów

Przy wyświetlonym wyniku, wyświetlony zostanie guzik, który pozwoli na udostępnienie swojego wyniku poprzez np. maila, grupę na fb, czy messengera. Na każdą aplikację, która umożliwia udostępnianie tekstu i jest zainstalowana na urządzeniu użytkownika. W ten sposób użytkownik może sam sobie wysłać na np. maila swój wynik testu.

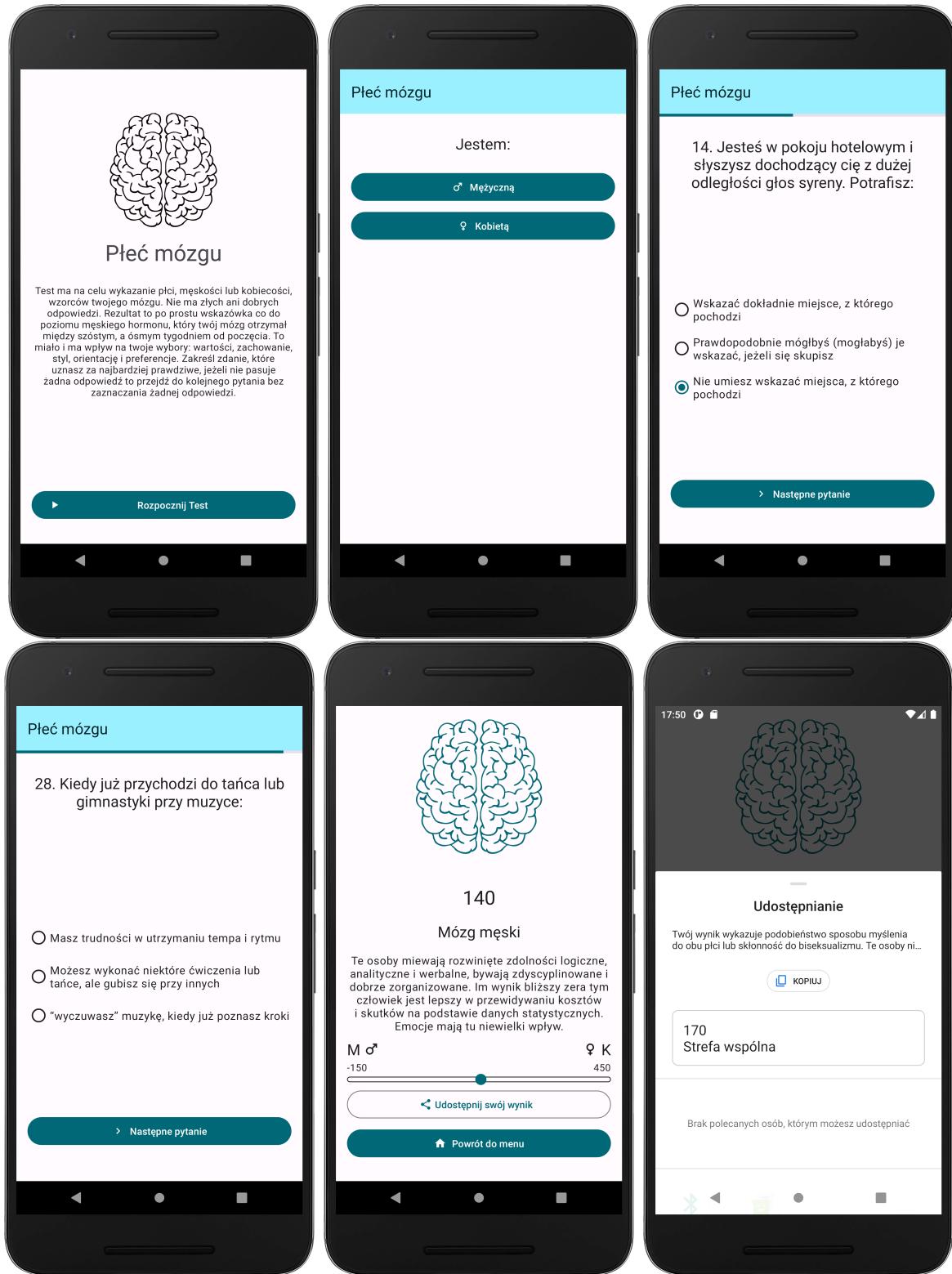
2.7 Losowość odpowiedzi

Przy testach, które mają dużo pytań, łatwo zauważać wzorzec, np. odpowiedzi A mogą kojarzyć nam się z typowo żeńskimi, a C z typowo męskimi, wtedy wybierając odpowiedź może nie kierować się prawdą, a tym co chcielibyśmy z takiego testu uzyskać (zwłaszcza przy znajomych). Dla każdego pytania odpowiedzi powinny być wyświetlane losowo.

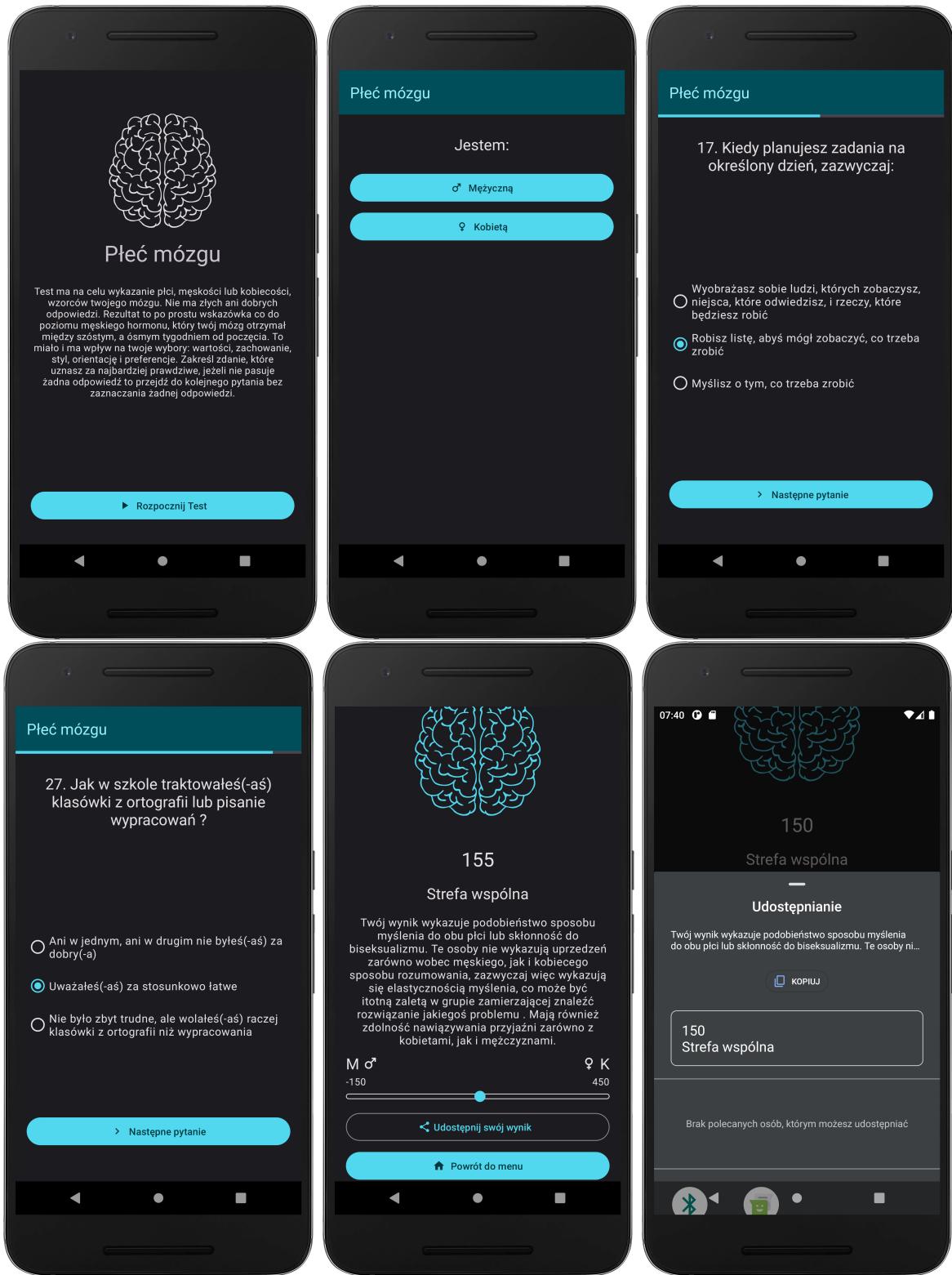
3 Wykorzystane narzędzia i technologie

- Środowisko Programistyczne - Android Studio
- Kotlin 1.6.1
- [Genrowanie stylu i kolorów z Material3](#)
- Git – system wersji kontroli
- Jetpack Navigation – biblioteka upraszczająca nawigację, przechowuje aktualnie wyświetlanaację w pamięci podręcznej, umożliwia prostsze przekazywanie argumentów. Będę z niej korzystał w wersji Kotlin DSL.
- Kotlinx Parcelize – biblioteka do generowania implementacji klasy Parcelable. W Androidzie serializacja jest bardzo wolnym procesem, więc korzysta się z parcelizacji, manualne pisanie implementacji Parcelable jest mało przejrzyste i powtarzalne.
- Architektura MVVM (Model, View, ViewModel)
- Kotlin flows - wykorzystywane do sekwencyjnego emitowania danych
- Dagger Hilt - biblioteka do wstrzykiwania zależności (DI)
- Gradle – Budowanie aplikacji, wersjonowanie aplikacji, zarządzanie bibliotekami w projekcie, konfiguracja projektu
- Resources z Androida – zarządzanie stałymi, co pozwala na zmianianie ich w zależności od pory dnia, języka i wielkości ekranu, pozwala także na nadanie nazwy jakiej wartości i użycie jej wiele razy. Jest to także sposób na tworzenie widoków oraz ekranów w aplikacji.
- ViewBinding - biblioteka do generowania referencji do widoków

4 Prezentacja powstałego projektu

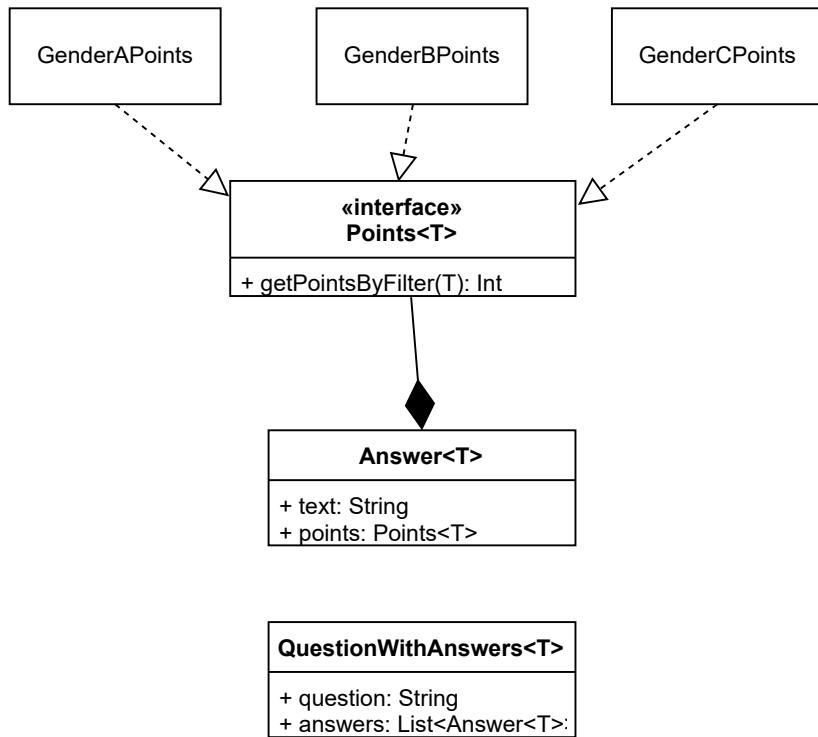


Rysunek 1: Zrzuty ekranów z aplikacji w wersji dziennej



Rysunek 2: Zrzuty ekranów z aplikacji w wersji nocnej

5 Diagram klas pytań z odpowiedziami



Rysunek 3: Diagram klas pytań z odpowiedziami

6 Wnioski

6.1 Jak bardzo skomplikowane jest pisanie oprogramowania

Pisząc oprogramowanie bardzo często wyobrażam sobie, jakie zachowanie i wygląd będzie posiadać dana funkcjonalność. Jednak napisanie tego samego w kodzie nawet dla bardzo prostych rzeczy potrafi być skomplikowane i trudne.

6.2 Samotny Programista musi nosić wiele masek

W dzisiejszych czasach oprogramowanie jest tworzone przez wieloosobowe zespoły. W takich zespołach znajdują się programiści, ale też designerzy, specjalisci od UX, marketingowcy, ludzie zajmujący się dźwiękiem itd. Pisząc samemu całe oprogramowanie często trzeba założyć maskę np. designera by zaprojektować wygląd danego ekranu, czy stworzyć ikonkę aplikacji. Oprogramowanie składa się z wielu małych elementów i każdy z nich trzeba dopieścić, więc samotni programiści muszą szybko się uczyć rzeczy nie związanych z ich zawodem, co potrafi spowolnić tworzenie oprogramowania i mocno je skomplikować, bo nie robimy czegoś na czym się tak bardzo znamy.

6.3 Porównanie Jetpack Navigation XML vs Kotlin DSL

W porównaniu do używanego przeze mnie dotychczas Jetpack Navigation pisanej w XML uważam, że Kotlin DSL jest bardziej skomplikowany, ale też bardziej elastyczny. Tracimy możliwość korzystania z Safe Args, wyświetlania prostego widoku naszej nawigacji, dużo częściej dodaje się akcje oraz argumenty. Jednak zyskujemy możliwość ponownego używania kodu np. można napisać funkcję, która zawsze nadawałaby taki sam typ animacji.

6.4

6.5 O co można rozszerzyć aplikację?

- Dodać inne testy o podobnej formie, można nawet użyć aktualnych widoków.
- Dodać możliwość cofania się do poprzednich pytań.
- Odblokować orientację z Portair i dodać widoki aplikacji w trybie Landscape, bo reszta aplikacji jest przystosowana tak, aby działała przy obydwu.
- Logowanie przez facebooka i wyświetlanie tabeli w której byśmy widzieli ile punktów zdobyli nasi znajomi i jaki tytuł im to daje.
- Zapisywanie lokalnie danych za pomocą ProtoDatastore, który jest idealny do zapisywania danych typu Klucz-Wartość np. Nazwa profilu mogła być naszym kluczem i nie powtarzać się. Wtedy warto dodać historię gier z nazwami profili i datami ich stworzenia.
- Dodać Firebase Analytics, tak abyśmy wiedzieli po ilu pytaniach użytkownicy np. najczęściej przerywają test i go nie kończą, dzięki temu moglibyśmy skrócić test o parę pytań.
- Wystawienie aplikacji na Sklep Play wraz z plikiem w wersji aab, który znaczaco zmniejsza rozmiar pobieranej aplikacji (sama aplikacja po pobraniu na telefon nie jest mniejsza).
- Dodać wiele prostych funkcjonalności do rozpowszechniania aplikacji jak np. guziki do udostępniania aplikacji w Menu, moglibyśmy nawet utworzyć Deep Linki, które od razu otwierałyby Sklep Play z naszą aplikacją, a gdy użytkownik nie ma Sklepu Play naszą stronę internetową.
- Dodanie reklam co ileś pytań, aby zmonetyzować aplikację.
- Odblokowywanie kolejnych testów za np. opłatą lub obejrzeniem [Rewarded Ad](#).

7 Bibliografia

- [Poradniki do Androida](#)
- [Grafiki wektorowe](#)

Kanały Youtube z których korzystałem ucząc się

- [PhilippLackner](#)
- [Android Developers](#)