I Problem – ankieta, analiza, rozwiązanie

Ankieta

W szkole Meritum została przeprowadzona anonimowa ankieta, przedstawione zostały pytania w następującej kolejności, można było opowiedzieć tak lub nie.

1: Czy osiągasz takie wyniki w szkole, jakie byś chciał?
2: Czy posiadasz aplikację do uczenia się?

Ankieta została przeprowadzona w 5 dni, zapytano 54 uczniów, podsumowane wyniki są następujące:

1-tak, 2-tak=0/54=0%

1-tak, 2-nie=22/54~40%

1-nie, 2-tak=5/54~9%

1-nie, 2-nie=27/54=50%

Analiza

Z wyników powyżej, można wyprowadzić wnioski:

-ok. 90% osób nie posiada aplikacji do uczenia się
-ok. 50% osób ma problemy z nauką, oraz nie ma aplikacji do uczenia się
-nie ma osoby, która posiada aplikacje do uczenia się, oraz jest usatysfakcjonowana swoimi wynikami w nauce

Rozwiązanie

Na podstawie powyższej analizy, postanowiliśmy zrobić aplikację do uczenia się. Wierzymy, że to czy się korzysta z aplikacji mobilnej proporcjonalnie wpływa na postępy w nauce.

II Zamysł

Stworzymy aplikację, która pomaga usystematyzować i uprościć własną naukę. Skupimy się na utrzymaniu naszej aplikacji ciekawej. Zrobimy z niej bardziej grę niż nudny podręcznik.

Główne funkcje aplikacji:

- Lista to-do - łatwe usystematyzowanie nauki. Pozwala użytkownikowi zapisywać ważne zadania takie jak “Pouczyć się na matmę”, “Sprawdzian biologia 20.04” lub służące jako zwykłe notatki.

- Osiągnięcia - zbieranie nagród za kamienie milowe w nauce. ‘Dopamine Rush’ dla użytkownika, który nakłania go do uczenia się więcej.

- Fiszki zwykłe – prosty, sprawdzony mechanizm do uczenia się

- Fiszki RUSH – tryb fiszek, który wymaga od użytkownika szybkiego wydobywania informacji z pamięci. Fiszki Rush to tryb, który daje użytkownikowi 90 sekund na odgadnięcie jak największej ilości fiszek we wcześniej utworzonym zestawie. Użytkownik może pomylić się 3 razy, zanim Rush zostanie przerwany. Po upływie 90 sekund gra automatycznie się kończy.

- Daily Streak – Seria dni, w których użytkownik ukończył dzienny cel (Ustalany wcześniej cel ukończonych codziennie fiszek oraz zadań). Daily Streak Nakłania użytkownika do korzystania z aplikacji codziennie, aby nie stracić “zdobytej” serii dni.

- Profil - możlwość dostosowania prędkości uczenia się, do własnych potrzeb.

Technologie których użyliśmy podczas tworzenia aplikacji:

- Aplikacja tworzona jest w języku dart, frameworku flutter, ponieważ bardzo łatwo jest pisać wieloplatformowy kod. Uczyć powinien się móc każdy. Sam Flutter jest bardzo przystępny od strony deweloperskiej, pozwalając nam na proste i szybkie ulepszanie Stoodee.

- SQLite - (SQFLITE dla flutter/dart). Szybki, prosty, niezawodny SQL. Nasza baza danych od strony lokalnej (Strony offline naszej aplikacji) została utworzona w SQLite, gdyż jest ono proste, szybkie, oraz dobrze wspomagane ze strony technologii Dart/Flutter. Spełnia ono wszystkie nasze potrzeby

- Firebase – Firestore. Skorzystaliśmy z technologii Firebase/Firestore dla naszego back-endu ze względu na prostotę. Baza danych Firestore zapewnia nam proste w użyciu oraz relatywnie szybkie zapisywanie i odczytywanie danych w chmurze. Firebase w naszej aplikacji działa we współpracy z SQLite. (Więcej w zakładce Back-end)

III Frontend

Oprawa graficzna aplikacji
-język: angielski
-motywy: jasny,ciemny

Kluczowe technologie/biblioteki:

- Go_router - nawigacja
- Introduction_screen – ekran wstępny
- flutter_native_splash – placeholder podczas ładowania aplikacji
- Cupertino-icons - ikonki

Start

Aplikacja przed samym załadowaniem, wita nas “splash screenem”. Jest to ekran, który pojawia się na czas ładowania aplikacji i znika po załadowaniu się potrzebnych rzeczy do poprawnego działania Stoodee. Wyświetla logo oraz przybiera kolor tła, zależnie od wybranego motywu systemu użytkownika.

Po załadowaniu

Stoodee wita nas ekranem Introduction_Screen, który wyświetli się tylko raz na obecnej maszynie, ma na celu płynne wprowadzenie użytkownika w sposób jaki Stoodee działa.

Introduction_screen ma 4 strony.

1. Wprowadzenie
2. Opisanie listy zadań
3. Opisanie fiszek
4. Zakończenie

Po naciśnięciu “done”, aplikacja przekieruje nas do ekranu logowania.

Obecne fiszki w introduction są nieinteraktywne. Gdy użytkownik spróbuje z nimi wejść w interakcje, dostanie komunikat na dole ekranu.

Ekran logowania

Mamy 2 pola, w pierwszym użytkownik wpisuje e-mail, w drugim hasło. Jest także checkbox, który włączy zapamiętywanie wpisanego loginu. Lewy przycisk rejestruje do bazy danych, prawy przycisk loguje do obecnego konta w bazie danych. O bazach danych i jak to jest zrobione, w sekcji IV backend.

Częsty problem jest taki, że jak się wpisuje hasło, następnie klika się login, aplikacja wykrywa jakby te hasło nie było wpisane. Wystarczy opuścić klawiaturę, kliknąć na białe pole na ekranie. Wtedy już przycisk logowania zadziała. W przyszłości warto myśleć o naprawieniu tego.

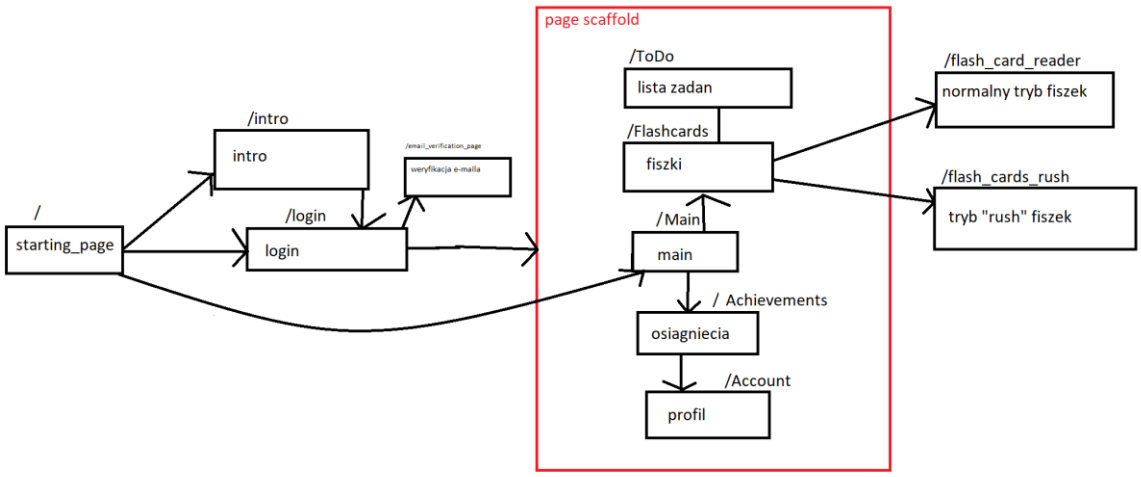
W lewym górnym rogu, jest opcja “skip log-in", wtedy użytkownik zaloguje się bez wpisywania danych, lecz opcje w takiej edycji aplikacji są limitowane. Zalecane jest korzystanie z logowania a bez logowania w wypadku braku internetu.

Po zalogowaniu się, aplikacja przechodzi do głównej części programu.

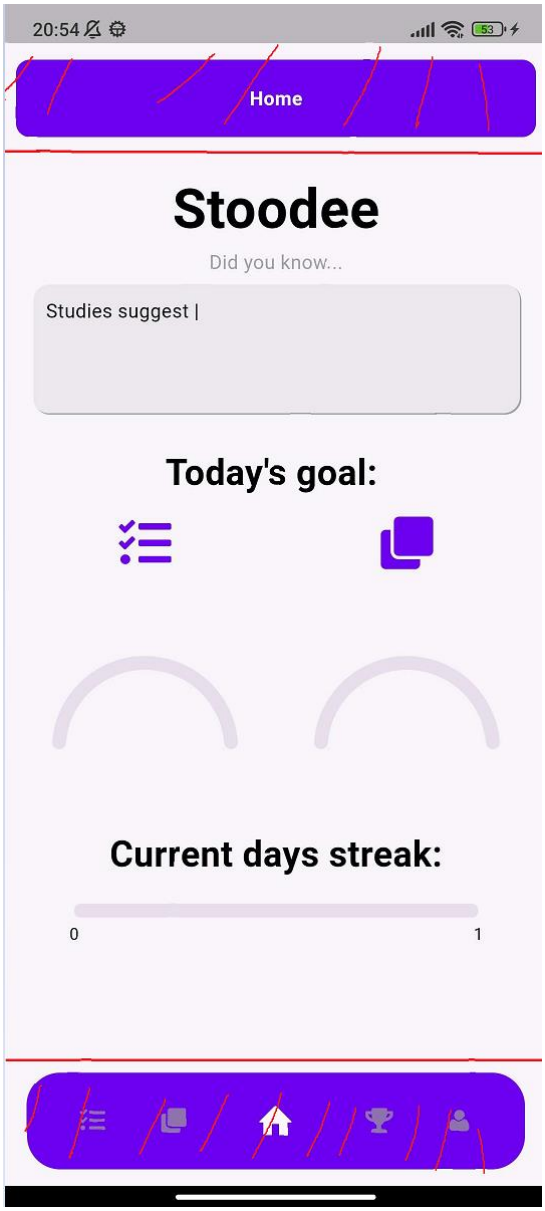
Teraz warto by było się zastanowić, jak dokłanie nawigowana jest aplikacja.

Aplikacja korzysta z biblioteki go_router. Jest to aplikacja która umożliwa nawigowanie pomiędzy ekranami dzięki ścieżkom dostępu, podobnie jak w przeglądarce. Gdyby aplikacja miała ekran2 w ekranie1, to ścieżka do niego by była: /ekran1/ekran2.

Poniżej znajduje się schemat nawigacji ekranów w aplikacji.



Zaczynamy od starting_page, ten ekran jest niewidoczny, sprawdza on czy zapamietywanie uzytkownika nie jest wlaczone, lub czy nie jest potrzebne pokazanie introduction_screena. Koniec końców kończymy w czerwonym bloczku, page_scaffold. Na screenie poniżej łatwiej da się zrozumieć co to jest. (WYJAŚNIENIE POD ZRZUTEM EKRANU)



Elementy które są zakolorowane na czerwono, to elementy należące do ekranu page_scaffold. Znajduje się tam nawigator oraz appbar, który daje napis, na którym ekranie jesteśmy. Dzięki takiemu rozwiązaniu, w każdym z 5 głównych ekranów, layout nawigatora pozostaje taki sam, jedyne co się zmienia to ekrany w środku nawigatora.

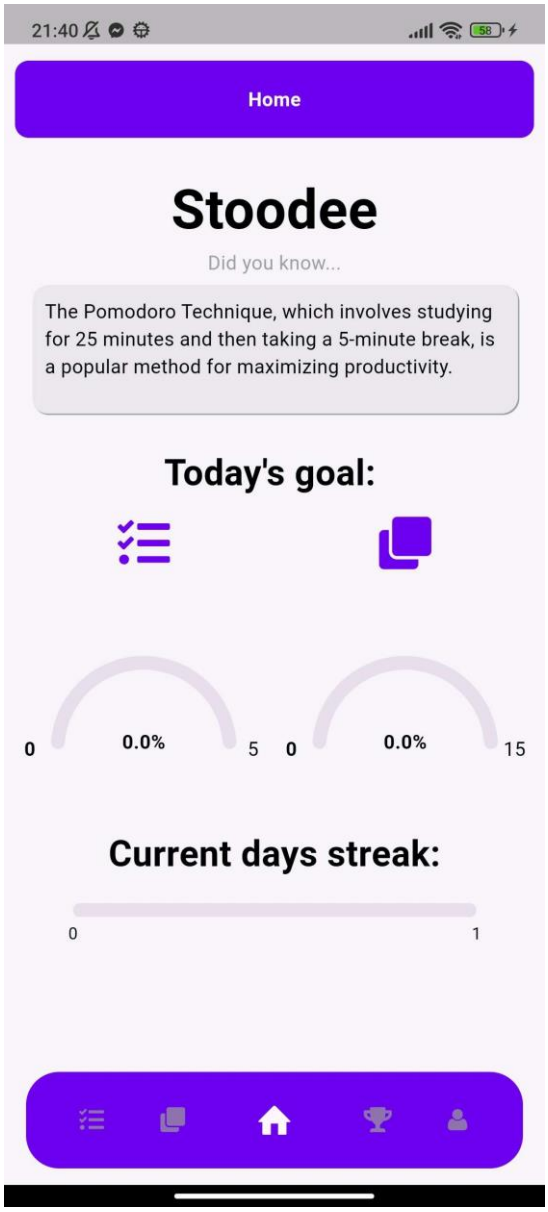
Ekrany można zmieniać poprzez przesuwanie palcem po ekranie oraz klikaniem na nawigator.

Część główna aplikacji składa się z 5 ekranów, od lewej:

1- Lista zadań
2- Fiszki
3- Strona główna
4- Osiągnięcia

5- Profil

Strona główna:



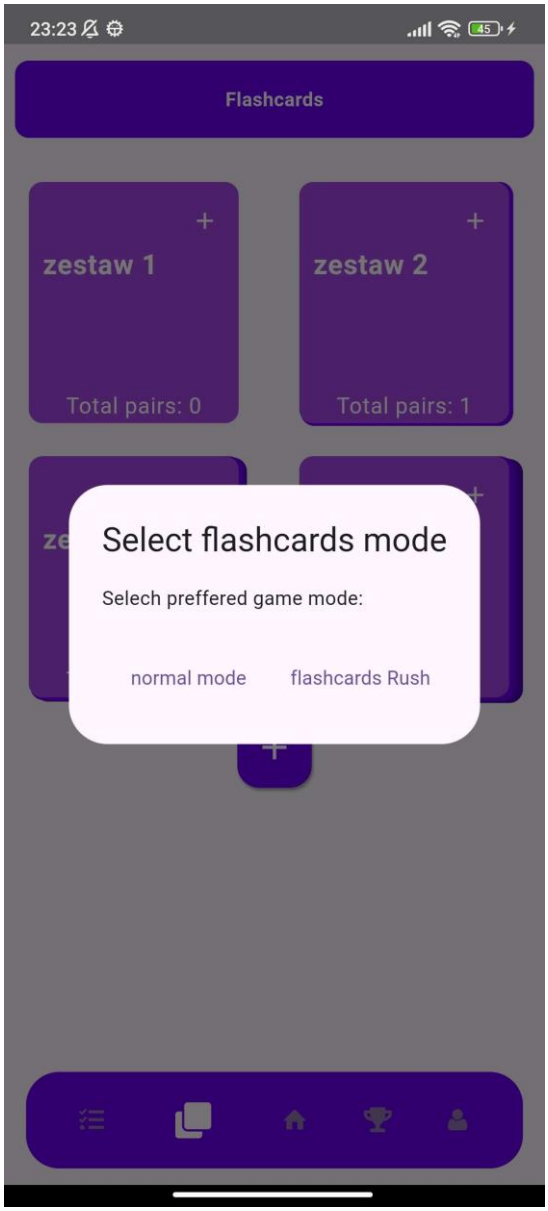
Strona główna ma w sobie:

- Ciekawostkę, która się losuje podczas każdego wywołania ekranu
- Miernik zrobionych zadań / zadań wyznaczonych na dzień
- Miernik zrobionych fiszek / fiszki wyznaczone na dzien
- Miernik dni, w których wszystko było zrobione, oraz są po kolei / obecny rekord. Czyli jeżeli mamy rekord, to miernik będzie wypełniony do pełna

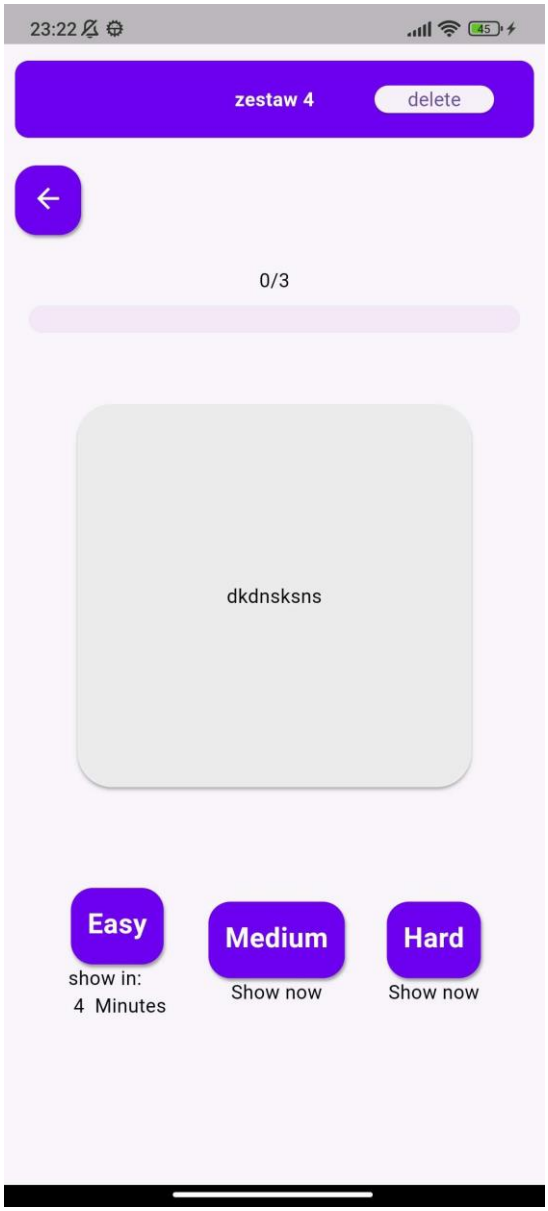
Jeden ekran w lewo,

Fiszki mają w sobie:

Przycisk dodania zestawu fiszek, po dodaniu pojawi się widżet reprezentujący dany zestaw. W prawym górnym rogu ma plus, który po kliknięciu wyświetla dialog, który dodaje fiszkę do zestawu fiszek.
Na dole zestawu fiszek jest ilość fiszek w zestawie.
Po przytrzymaniu pojawi się dialog potwierdzający usunięcie zestawu fiszek.
Po wciśnięciu zestawu, pojawi się dialog wybierający tryb otwarcia zestawu.

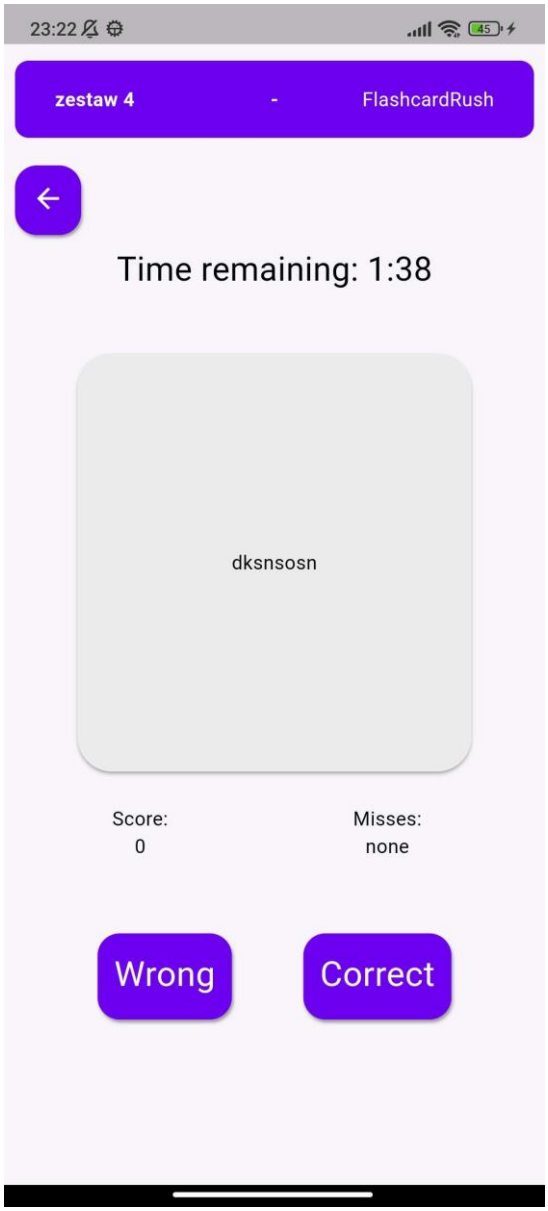


Tryb normalny:



W trybie normalnym, przechodzimy przez zestaw, po kliknięciu na fiszkę pojawiają się 3 różne przyciski. Łatwy zwiększy czas na ponowne pojawienie się fiszki, średni zostawi taki czas jaki jest, a hard zmniejszy czas, fiszka pojawi się szybciej.

Tryb fiszki rush:



Tryb fiszki rush polega na odgadnięciu jak najwięcej fiszek, w jak najkrótszym czasie. Użytkownik ma 100 sekund, po 3 złych odgadnięciach tryb się przerywa.

Decyzja czy fiszka była poprawnie odgadnięta, czy nie, zostawiamy dla użytkownika.

Jeszcze raz w lewo,
Lista zadań:

Po wciśnięciu przycisk wyświetla dialog, w którym wpisuje się nazwę zadania.

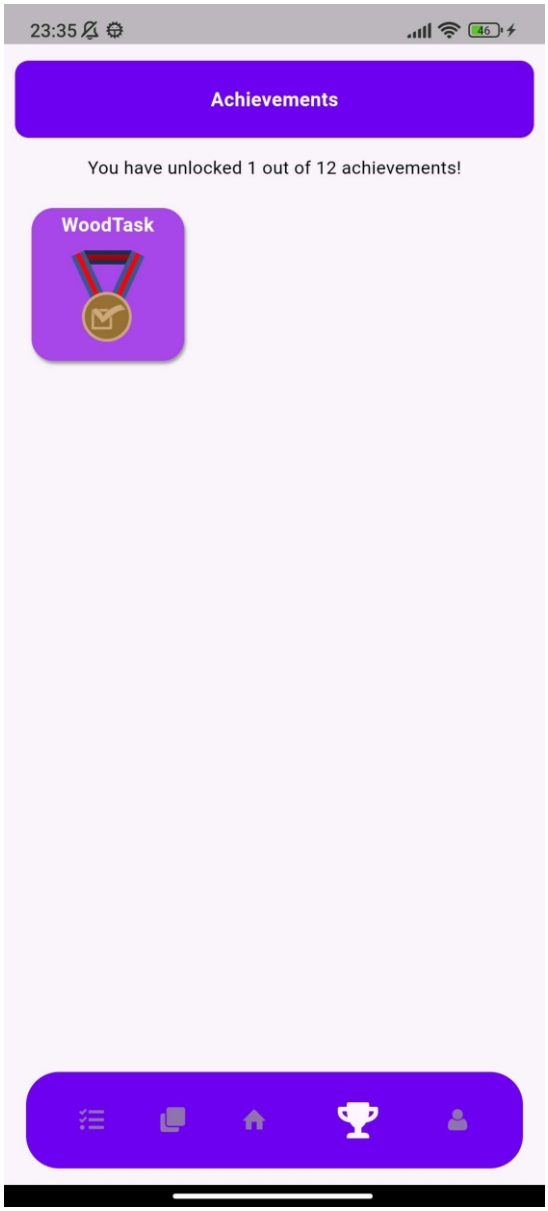
Przesuń w lewo, a zadanie zostanie usunięte i oznaczone jako nieukończone.

Przesuń w prawo, a zadanie zostanie oznaczone jako ukończone.

Strona osiągnięć, czwarta od lewej:

Pokazują się tutaj osiągnięcia, zdobyte za:

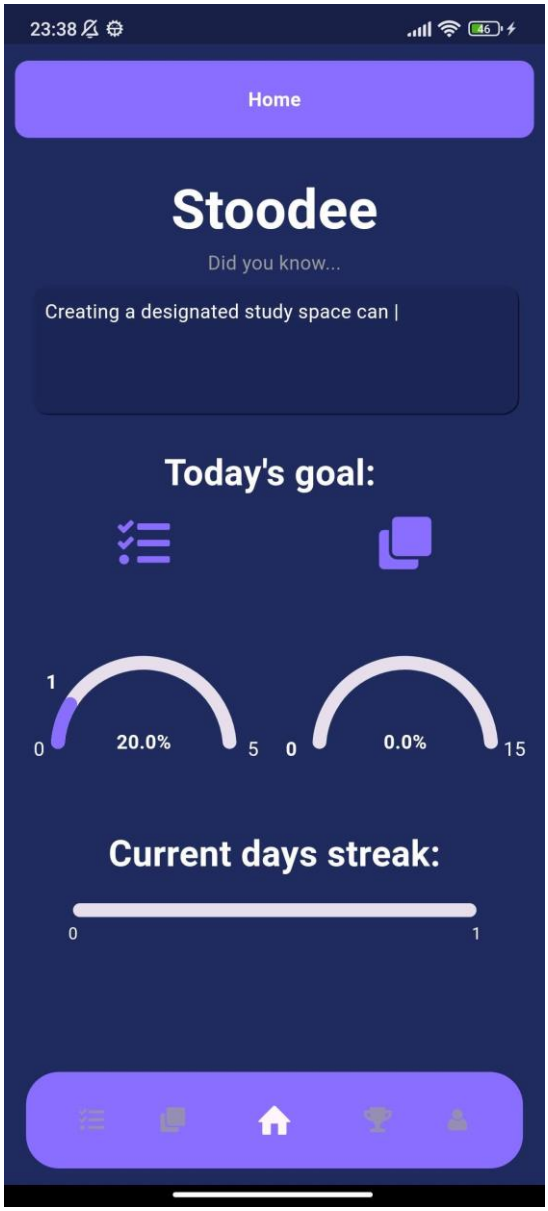
- Kończenie zadań
- Kończenie fiszek
- Punkty w flashcard rushu
- Serie dni



Profil, strona najbardziej po lewo:

Tutaj znajdują się wszystkie statystyki użytkownika, również z jego nazwą.
W prawym górnym rogu można zmienić nazwę użytkownika, wyznaczoną ilość zadań i fiszek na dzień, oraz motyw.

Ciemny motyw:



Nadal jesteśmy w profilu, w prawym dolnym rogu jest przycisk do synchronizacji danych użytkownika z chmurą. Na dole pośrodku jest przycisk do wylogowania się.

IV Backend

LOKALNY CRUD:

Korzeniem back-endu jest klasa LocalDbController. Technologia, z której korzysta baza

LocalDbController jest SQLite.

LocalDbController jest odpowiedzialne za lokalne Tworzenie, wczytywanie, edytowanie oraz usuwanie danych. Odpowiada ona również za komunikację z chmurą (klasą CloudDbController) oraz zapisywanie i wczytywanie preferencji użytkownika (klasa SharedPrefs. Np. Preferencja koloru tła).

Obiekty w bazie CloudDbController są reprezentowane klasami:

User – obiekt użytkownika lokalnego. **(UŻYTKOWNIK LOKALNY ORAZ UŻYTKOWNIK CHMUROWY TO DWA RÓŻNE OBIEKTY, KTÓRE ZAWSZE POWINNY DZIAŁAĆ SYNCHRONICZNIE)**

Task – obiekt zadania To-Do listy

Flashcard - obiekt fiszki

FlashcardSet – obiekt zestawu fiszek

Kolejnymi klasami podrzędnymi LocalDbController są:

FlashCardService:

FlashCardService zajmuje się back-endem fiszek. Będąc kolejną warstwą abstrakcji od bazy danych, udostępnia ono funkcje dotyczące Zapisywania, Usuwania, Edytowania i czytania fiszek, odwołując się do metod LocalDbController.

FlashcardService operuje na klasach FlashcardSet oraz Flashcard.

ToDoService:

ToDoService zajmuje się back-endem Zadań listy TO-DO użytkownika. Będąc kolejną warstwą abstrakcji od bazy danych, udostępnia ono funkcje dotyczące Zapisywania, Usuwania, Edytowania i czytania zadań, odwołując się do metod LocalDbController.

Zadanie w aplikacji reprezentowane jest przez klasę Task

AUTORYZACJA UŻYTKOWNIKA:

Do użytkownika lokalnego domyślnie jest przypisana wartość NullUser, oznaczająca użytkownika niezalogowanego. NullUser nie ma możliwości podglądania statystyk, utrzymywania Day Streak oraz synchronizacji z bazą.

Aby utworzyć konto lub zalogować się należy zostać autoryzowanym przez Firebase. Użytkownik jest autoryzowany wyłącznie poprzez klasę AuthService przez instancję .firebase().

Firebase dostarcza metody logowania, rejestracji i autoryzowania poprzez klasę FirebaseAuthProvider rozszerzającą klasę abstrakcyjną AuthProvider.

W praktyce wywołuje się ją: ***AuthService.firebase()***

Po utworzeniu konta w chmurze wymienioną wyżej instancją, automatycznie tworzy się lokalna reprezentacja konta poprzez LocalDbController. Logowanie chmurowe zawsze powinno być związane z logowaniem lokalnym, a lokalna reprezentacja użytkownika zawsze powinna być równa z chmurą.

ZAPISYWANIE CHMUROWE:

Bazą w chmurze jest Firestore. Kontakt z bazą jest rozwiązany poprzez klasę CloudDbController która powinna być wykorzystywana jedynie przez klasę LocalDbController.

Synchronizacja lokalnych danych użytkownika z bazą jest wykonywana poprzez naciśnięcie przycisku SyncWithCloud na profilu użytkownika, który wywołuje funkcję synchronizacji poprzez LocalDbController. LocalDbController upewnia się o możliwości synchronizacji po czym decyduje, czy zapisać, czy wczytać dane z bazy.

Jeżeli dane lokalne są przedawnione LocalDbController odwołuje się do klasy CloudDbController i wczytuje dane Firestore dla zalogowanego użytkownika, przypisując je do lokalnych wartości.

Jeżeli dane w bazie chmurowej są przedawnione, LocalDbController odwołuje się do klasy CloudDbController, wysyłając paczkę z nowymi danymi do chmury.

PREFERENCJE UŻYTKOWNIKA:

Małe preferencje użytkownika takie jak “czy zapamiętać login” lub “motyw ciemny czy jasny” są zapisywane lokalnie za pomocą pakietu SharedPreferences w klasie statycznej SharedPrefs.

NETWORK CONTROLLER:

Narazie rozwiązane jest to poprzez jedną prostą funkcję bool -> hasNetworkConnection() sprawdzającą czy urządzenie ma dostęp do internetu.