Prezentacja:

https://pgedupl-my.sharepoint.com/:p:/g/personal/s198143_student_pg_edu_pl/IQBS6P9-woQ4T5cIz5LSUHtfAUEyKWYJIpA9tVFRZUeVi3k?e=THNnud

Aplikacja dla dzieci do nauki tabliczki mnożenia - slajd 1

Autorzy - slajd 2

Zespół projektowy:

- Marta Kociszewska
- Lidia Zawrzykraj
- Piotr Kierznowski
- Weronika Kankowska

Opiekun projektu:

- dr inż. Barbara Stawarz - Graczyk

Cel projektu - slajd 3 i 4

Innowacyjna edukacja poprzez grywalizację

- zaprojektowanie i zaimplementowanie mobilnej aplikacji edukacyjnej, która wspomaga naukę tabliczki mnożenia u dzieci
- wykorzystanie mechanizmów grywalizacji, takich jak tryby praktyki, sesje egzaminacyjne oraz system odblokowywania kolejnych poziomów trudności
- stworzenie interaktywnego środowiska motywującego do systematycznej nauki

Rozwój kompetencji inżynierskich i zespołowych

- zdobycie praktycznych umiejętności w zakresie projektowania, implementacji i dokumentacji aplikacji mobilnych
- rozwijanie umiejętności pracy zespołowej, zarządzania projektem oraz efektywnej komunikacji w grupie
- doskonalenie kompetencji technicznych związanych z wykorzystaniem nowoczesnych technologii, takich jak Flutter i Firebase
- nauka profesjonalnych praktyk programistycznych, w tym zarządzania wersjami kodu za pomocą Git i GitHub

Organizacja pracy i komunikacja - slajd 5 i 6

Nadzór merytoryczny

- comiesięczne spotkania z opiekunem projektu w celu :
  - weryfikacji postępów,
  - uzyskania wskazówek,
  - omówienia kluczowych decyzji
  - rozwiązywania problemów
  - dostosowania planu pracy

Współpraca wewnątrz zespołu

- systematyczne spotkania zespołowe, na żywo lub online
- dynamiczne rozdzielanie zadań w zależności od aktualnych potrzeb i umiejętności członków zespołu
- synchronizacja wizji projektu i rozwiązywanie problemów technicznych
- kultura feedbacku i otwartej komunikacji - regularne dzielenie się postępami i wyzwaniami, wspólne poszukiwanie rozwiązań

Wersjonowanie i współpraca zespołowa - slajd 7

- realizacja projektu z wykorzystaniem systemu kontroli wersji Git oraz platformy GitHub
- tworzenie oddzielnych gałęzi (branchy) dla nowych funkcjonalności i poprawek, co umożliwia równoległą pracę nad różnymi aspektami aplikacji

Organizacja pracy zespołowej - slajd 8

- każdy etap projektu jest planowany i dokumentowany w dedykowanych issue na GitHubie
- stworzone zostały labele (etykiety) do kategoryzacji zadań według ich rodzaju i priorytetu
- przypisywanie issue do konkretnych członków zespołu, co ułatwia śledzenie odpowiedzialności i postępów prac
- regularne aktualizacje statusu issue, co pozwala na bieżąco monitorować realizację zadań i identyfikować ewentualne blokady

Proces wprowadzania zmian - slajd 9

- każda zmiana w kodzie jest wprowadzana poprzez pull requesty
- pull requesty są przeglądane i zatwierdzane przez innych członków zespołu przed scaleniem z główną gałęzią
- umożliwia to wczesne wykrywanie błędów, zapewnia zgodność z ustalonymi standardami kodowania oraz promuje współpracę i wymianę wiedzy w zespole
- w razie błędów lub konfliktów, project manager koordynuje ich rozwiązanie, zapewniając płynny przebieg prac nad projektem
- dyskusje dotyczące pull requestów odbywają się na platformie GitHub, co umożliwia dokumentowanie decyzji i ułatwia śledzenie historii zmian

Tech stack - slajd 10

- wykorzystane technologie z podziałem na frontend i backend:

Frontend:

- Flutter
- Dart

Backend:

- Firebase

Wybór technologii - slajd 11

- Architektura "Everything is a Widget"
  - architektura Fluttera oparta na komponentach zwanych widgetami, które definiują wygląd i zachowanie interfejsu użytkownika
  - umożliwia tworzenie złożonych interfejsów poprzez łączenie prostszych elementów
  - ułatwia ponowne wykorzystanie kodu i modularność aplikacji
- Cross-platformowość
  - Flutter umożliwia tworzenie aplikacji mobilnych działających na różnych platformach, takich jak Android i iOS, z jednego kodu źródłowego
- Hot Reload
  - funkcja Fluttera pozwalająca na natychmiastowe wprowadzanie zmian w kodzie i ich szybkie podglądanie w aplikacji bez konieczności ponownego uruchamiania całego projektu

- Backend as a Service
  - Firebase oferuje gotowe rozwiązania backendowe, takie jak bazy danych, uwierzytelnianie użytkowników i hosting
  - przyspiesza rozwój aplikacji
  - redukuje potrzebę zarządzania infrastrukturą serwerową
- Firestore – realtime database
  - baza danych NoSQL oferująca synchronizację danych w czasie rzeczywistym między klientami a serwerem
  - umożliwia tworzenie dynamicznych aplikacji z aktualizacjami danych na żywo

Od projektu w Figma do aplikacji - slajd 12 i 13

- w pierwszej fazie projektu zaprojektowano interfejs użytkownika aplikacji w Figma
  - uwzględniono potrzeby i preferencje docelowej grupy użytkowników, czyli dzieci uczących się tabliczki mnożenia
- Figma jako narzędzie do projektowania interfejsu użytkownika
  - umożliwia tworzenie prototypów i wizualizacji aplikacji przed rozpoczęciem implementacji
  - ułatwia współpracę zespołową poprzez możliwość komentowania i udostępniania projektów
- Integracja projektu z implementacją
  - przeniesienie zaprojektowanych elementów interfejsu z Figmy do kodu Fluttera
  - zapewnienie spójności wizualnej i funkcjonalnej między projektem a finalną aplikacją
- Proces implementacji
  - podział pracy na moduły odpowiadające poszczególnym ekranom i funkcjonalnościom aplikacji
  - iteracyjne testowanie i dostosowywanie interfejsu w trakcie implementacji, aby zapewnić intuicyjność i atrakcyjność dla użytkowników końcowych

Architektura aplikacji - slajd 14

- architektura zdarzeniowa

  - aplikacja reaguje na zdarzenia generowane przez użytkownika lub system, co pozwala na dynamiczne i responsywne działanie
  - ułatwia zarządzanie stanem aplikacji i przepływem danych
- Baas (Backend as a Service)
  - wykorzystanie Firebase jako gotowego rozwiązania backendowego, co przyspiesza rozwój aplikacji i redukuje potrzebę zarządzania infrastrukturą serwerową
- Real-time data flow
  - synchronizacja danych w czasie rzeczywistym między klientami a serwerem
- Warstwa bezpieczeństwa
  - implementacja mechanizmów uwierzytelniania i autoryzacji użytkowników
  - zabezpieczenie danych przechowywanych w bazie Firestore poprzez reguły bezpieczeństwa danych

Backend i zarządzanie danymi - slajd 15

- struktura cloud Firestore
  - kolekcje i dokumenty przechowujące dane użytkowników, wyniki nauki oraz postępy w aplikacji
- cloud functions
  - funkcje serwerowe obsługujące logikę biznesową
  - zaimplementowane dwie funkcje:
    - `onUserCreate` - wywoływana przy rejestracji nowego użytkownika
      - Tworzy dokument users/{uid}
      - Inicjalizuje profil z danymi z Firebase Auth
      - Ustawia początkowe statystyki (0 gier, 0 punktów)
    - `onResultsWrite` - wyzwalana przy zapisie wyniku, aktualizuje statystyki użytkownika
      - Oblicza streak (seria dni z rzędu)
      - Dodaje punkty do totalPoints
      - Inkrementuje totalGamesPlayed
- uwierzytelnianie użytkowników (firebase authentication)

  - **STATUS: Logika zaimplementowana, UI niezaimplementowane**
  - zaimplementowano `AuthService` w aplikacji Flutter do obsługi procesów uwierzytelniania
    * rejestracja nowych użytkowników za pomocą email i hasła
    * walidacja hasła - min 8 znaków, wielka litera, cyfra, znak specjalny
    * sprawdzanie unikalności username w bazie Firestore
    * reset hasła poprzez email
    * logowanie z obslugą błędów (nieprawidłowy email, złe hasło, brak konta)
    * tworzenie dokumentu użytkownika w Firestore przy rejestracji
  - **BRAKUJE**: ekran logowania i rejestracji - aplikacja obecnie działa bez uwierzytelniania
  - **KONSEKWENCJA**: użytkownicy nie mogą tworzyć kont, więc dane nie są synchronizowane z Firebase
- automatyzacja statystyk
  - **STATUS: Cloud Functions zaimplementowane i przetestowane**
  - wcześniej opisana cloud functions do automatycznego aktualizowania statystyk użytkowników po każdej sesji nauki - `onResultsWrite`
  - zbieranie i analiza danych dotyczących postępów użytkowników w nauce tabliczki mnożenia
  - **OGRANICZENIE**: funkcje działają, ale nie są wywoływane w praktyce, ponieważ wyniki nie są zapisywane do Firestore (brak połączenia logiki gry z ResultsService)
- bezpieczeństwo danych
  - implementacja reguł bezpieczeństwa Firestore w celu ochrony danych użytkowników i zapewnienia odpowiednich poziomów dostępu
    * Firebase Security Rules - dostęp tylko do własnych danych
    * weryfikacja UID użytkownika przy każdej operacji odczytu i zapisu
    * wyniki są nie mutowalne - nie można ich edytować po zapisaniu

Baza danych - slajd 16

- struktura bazy danych w Firestore

- kolekcja “users” - dane użytkowników
  - “profile” - dane profilowe użytkownika - username, email, data utworzenia profilu
  - “stats” - statystyki nauki - liczba ukończonych sesji, wyniki, czas nauki (streak), data ostatniej sesji
  - **STATUS**: struktura zdefiniowana i tworzona przy rejestracji, ale nie jest używana (brak UI logowania)
- kolekcja “users results” - historia wyników
  - przechowująca wyniki z sesji po każdej ukończonej sesji gry
  - immutable - nieedytowalne po zapisaniu
  - **STATUS**: kolekcja zdefiniowana, ale wyniki gier NIE są zapisywane do Firestore (logika gry używa tylko LocalSaves)
- kolekcja “game progress” - synchronizacja postępów gry
  - przechowująca informacje o postępach w trakcie trwania rozgrywki w trybie offline, umożliwiająca kontynuację gry po przerwaniu lub utracie połączenia internetowego
  - **STATUS**: struktura zdefiniowana, ale nie jest używana (logika gry nie wywołuje ProgressService)

Synchronizacja danych - slajd 17

- architekutura **offline-first**
  - **STATUS: Zaimplementowana, ale niepołączona z logiką gry**
  - dane zapisywane lokalnie na urządzeniu użytkownika
  - synchronizacja z bazą Firestore w tle, gdy dostępne jest połączenie z internetem
  - aplikacja działa płynnie nawet przy niestabilnym lub braku połączenia sieciowego
- lokalny storage - **DWA NIEZALEŻNE SYSTEMY**
  - technologia: **Hive** - lekka baza NoSQL dla Fluttera
  - **System 1: LocalSaves** - używany przez grę
    - przechowuje: dane użytkownika, postępy poziomów, odblokowane poziomy, leaderboard
    - działa z hardcoded userId: “user1”
    - **NIE synchronizowany z Firebase**

  - **System 2: OfflineStore** - dla synchronizacji
    * przechowuje: wyniki gier (GameResult), postępy gier (GameProgress)
    * **PROBLEM**: nigdy nie wywoływany przez logikę gry
    * synchronizacja zaimplementowana, ale nie ma danych do synchronizacji

- kolejka synchronizacji
  - **STATUS: W pełni zaimplementowana**
  - wykorzystuje Hive do przechowywania
  - po przywróceniu połączenia z internetem, dane zgromadzone w lokalnym storage są automatycznie synchronizowane z bazą Firestore
  - kolejka persystowana - przetrwa zamknięcie aplikacji i ponowne uruchomienie urządzenia
  - **PROBLEM**: kolejka pozostaje pusta, ponieważ:
    * brak uwierzytelnionego użytkownika (wymóg: FirebaseAuth.currentUser)
    * logika gry nie wywołuje ResultsService/ProgressService

- wyzwalacze i proces synchronizacji
  - **Zaimplementowane**: synchronizacja okresowa (co 15 minut) lub zdarzeniowa (po wykryciu połączenia z internetem)
  - **Działanie warunkowe**: synchronizuje tylko gdy użytkownik jest zalogowany

- retry z exponential backoff
  - w przypadku błędu sieciowego czas przeznaczony na oczekiwanie na połączenia jest zwiększany wykładniczo (Próba n: czekaj 2^(n-1) sekund)
  - Przy błędzie sieciowym:
    * Próba 1: czekaj 1 sekundę
    * Próba 2: czekaj 2 sekundy
    * Próba 3: czekaj 4 sekundy
    * ...
    * Maksymalnie: 300 sekund (5 minut)

Pytanie: Jak rozwiązywane są konfilkty wynikające z synchronizacji?a

- automatyczne rozwiązywanie konfliktów danych podczas synchronizacji
  - dla `user_result`:

        * deduplikacja - jeśli wynik o tym samym ID już istnieje w Firestore, nie dodawaj go ponownie
    - dla `game_progress`:
        * nadpisywanie (last-write-wins) - najnowszy zapisany postęp gry zastępuje starszy w Firestore
        * porównanie timestampów w celu ustalenia najnowszych danych
        * synchronizacja tylko jeśli lokalna wersja jest nowsza

Przykładowe scenariusze:

- użytkownik gra offline, a następnie łączy się z internetem
    - lokalne dane `game_progress` są synchronizowane z Firestore
    - jeśli istnieje konflikt (tj. dane lokalne i z Firestore różnią się), najnowsze dane nadpisują starsze
- użytkownik ma zduplikowane wyniki w lokalnym storage
    - podczas synchronizacji, duplikaty są usuwane, a tylko unikalne wyniki są zapisywane w Firestore
    - zapobiega to powielaniu danych i utrzymuje spójność bazy danych
- użytkownik edytuje postęp gry na dwóch urządzeniach jednocześnie
    - podczas synchronizacji, najnowszy zapisany postęp gry (na podstawie timestampu) jest zachowywany w Firestore
    - starszy zapis jest odrzucany, zapewniając, że tylko aktualne dane są przechowywane

Stan aktualny implementacji i planowane prace

Co działa obecnie:

- Aplikacja jako samodzielna gra lokalna
    - Pełna funkcjonalność rozgrywki (tryby praktyki i egzaminy)
    - System odblokowania poziomów
    - Przechowywanie postępów lokalnie (LocalSaves/Hive)
    - Interfejs użytkownika zgodny z projektem Figma
- Infrastruktura backend (Firebase)
    - Cloud Functions (onUserCreate, onResultsWrite) - przetestowane i działające
    - Struktura bazy danych Firestore zdefiniowana
    - Reguły bezpieczeństwa skonfigurowane
    - AuthService z pełną logiką uwierzytelniania
- Architektura offline-first
    - OfflineStore zaimplementowany
    - SyncService z kolejką, retry logic, exponential backoff
    - Mechanizmy rozwiązywania konfliktów

Co wymaga uzupełnienia:

1. **UI warstwy uwierzytelniania**
   - Ekran logowania
   - Ekran rejestracji
   - Routing dla niezalogowanych użytkowników
2. **Połączenie logiki gry z synchronizacją**
   - Wywołanie ResultsService przy zakończeniu sesji gry
   - Wywołanie ProgressService podczas rozgrywki
   - Migracja z hardcoded "user1" na rzeczywisty Firebase UID
3. **Integracja dwóch systemów storage**
   - LocalSaves (obecny system gry) → pozostaje dla offline gameplay
   - OfflineStore + Firebase → dla cross-device sync i statystyk
   - Wspólne zarządzanie danymi między systemami

Dlaczego taka architektura:

- **Modułowość**: każdy komponent działa niezależnie i jest testowalny
- **Iteracyjny rozwój**: możliwość rozwijania aplikacji krok po kroku
- **Separation of concerns**: logika gry oddzielona od synchronizacji danych