Purrfect Trivia

-a trivia based game-

Proiect Inginerie Software 2024-2025

Pantea Tania
Rujac Roxana
Sabau Emanuela
-grupa 30234-

Cuprins

Introducere

Proiectul nostru este o aplicatie simpla de tip Trivia, unde poti raspunde la diferite tipuri de intrebari pentru a aduna puncte. Numele este un joc de cuvinte deoarece multe elemente din grafica jocului (logo-ul, iconita de titlu, badge-urile) sunt bazate pe imagini cu pisici.

Jocul dispune de urmatoarele functionalitati:

- Functie de creare a unui cont - Register
- Functie de autentiticare intr-un cont deja existent - Login
- Modul Quick Game - selectarea unei categorii de joc din cele existente si inceperea unui joc
    - Space
    - General Knowledge
    - Sports
    - History
    - Movies & TV Shows
    - Music
    - Technology
    - Art & Literature
    - Mythology
    - Famous Personalities
    - Travel Destinations
    - Psychology
    - Hobbies

- Modul Challenge - se poate selecta un alt jucator din baza de date, o categorie si un timp pentru a “provoca” pe cineva la un joc
- Leaderboard - se poate vizualiza topul jucatorilor, inclusiv unde se situeaza persoana autentificata curent, in functie de scorurile obtinute la quiz-uri
- Pagina de help - informatii despre cum se desfasoara un joc
- Pagina Profile - se pot vizualiza scorul total, notificarile, realizarile (achievements), insignele (badges), poza de profil

Detalii tehnice

Frontend:

- Framework: Flutter
- Limbaj de programare: Dart

Backend:

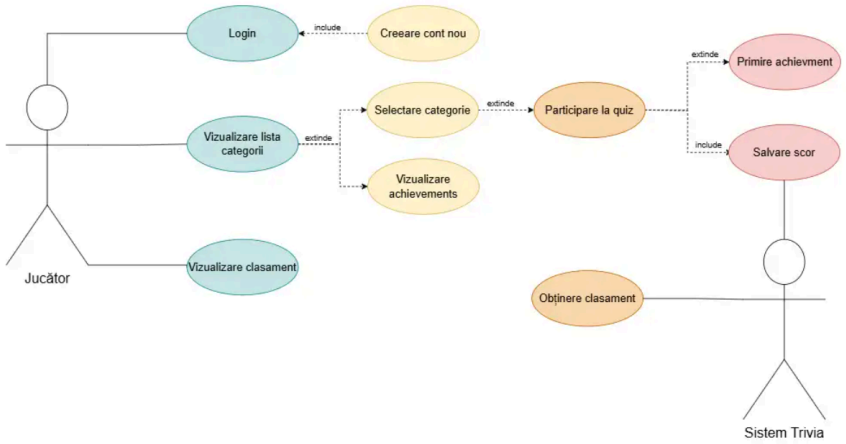
- Framework: Express.js
- Limbaj de programare: JavaScript
- Bază de date: MySQL

Arhitectura Aplicației

1. **Structură generală**:
    - **Frontend (Flutter)** comunică cu **Backend (Express.js)** folosind API-ul RESTful.
    - Backend-ul interacționează cu baza de date MySQL pentru a obține și a salva date.
2. **Flux de date**:
    - Utilizatorul trimite o cerere prin aplicația Flutter (de exemplu, autentificare sau selectarea întrebărilor).
    - Cererea este procesată de API-ul Express.js, care efectuează operații asupra bazei de date.
    - Backend-ul răspunde cu rezultatele cerute, care sunt afișate pe interfața utilizatorului.

Diagrama use-case

Initial, diagrama use case a fost urmatoarea:

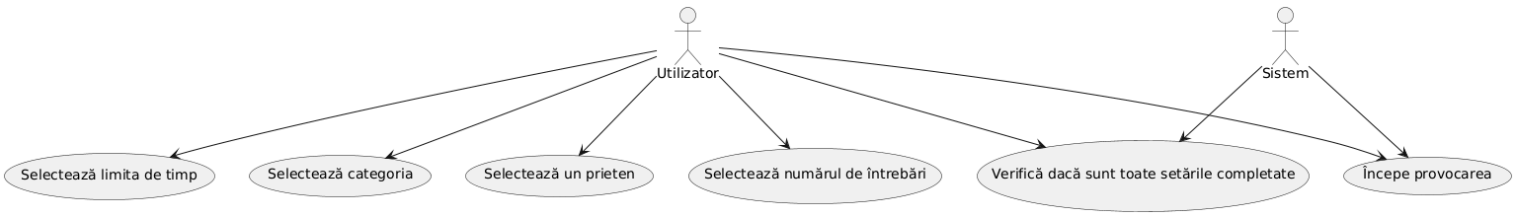


Insa, pe parcurs proiectul a suferit cateva modificari iar diagragrama actuala a devenit: https://drive.google.com/file/d/1TjiDkUFffaXFgd8_P0CkHt6ripzZZJlZ/view?usp=sharing

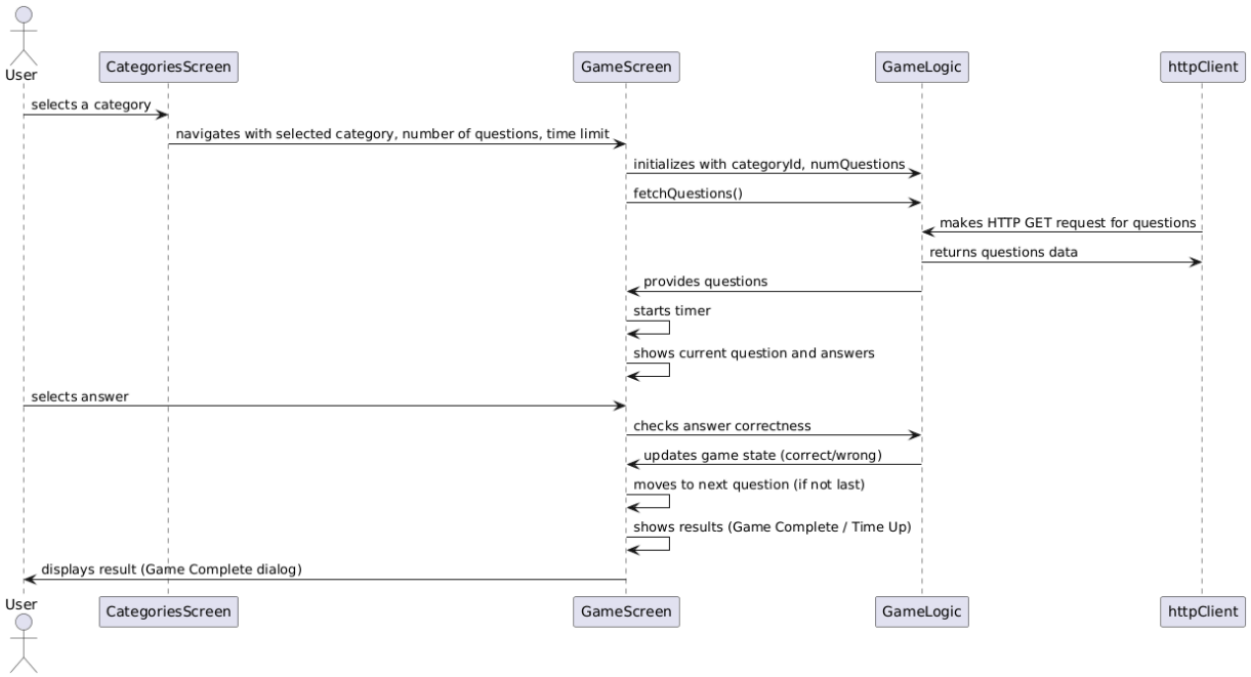
Diagrame individuale

Roxana: am realizat urmatoarele diagrame:

-diagrama de use case pentru functionalitatea de challenge friend:

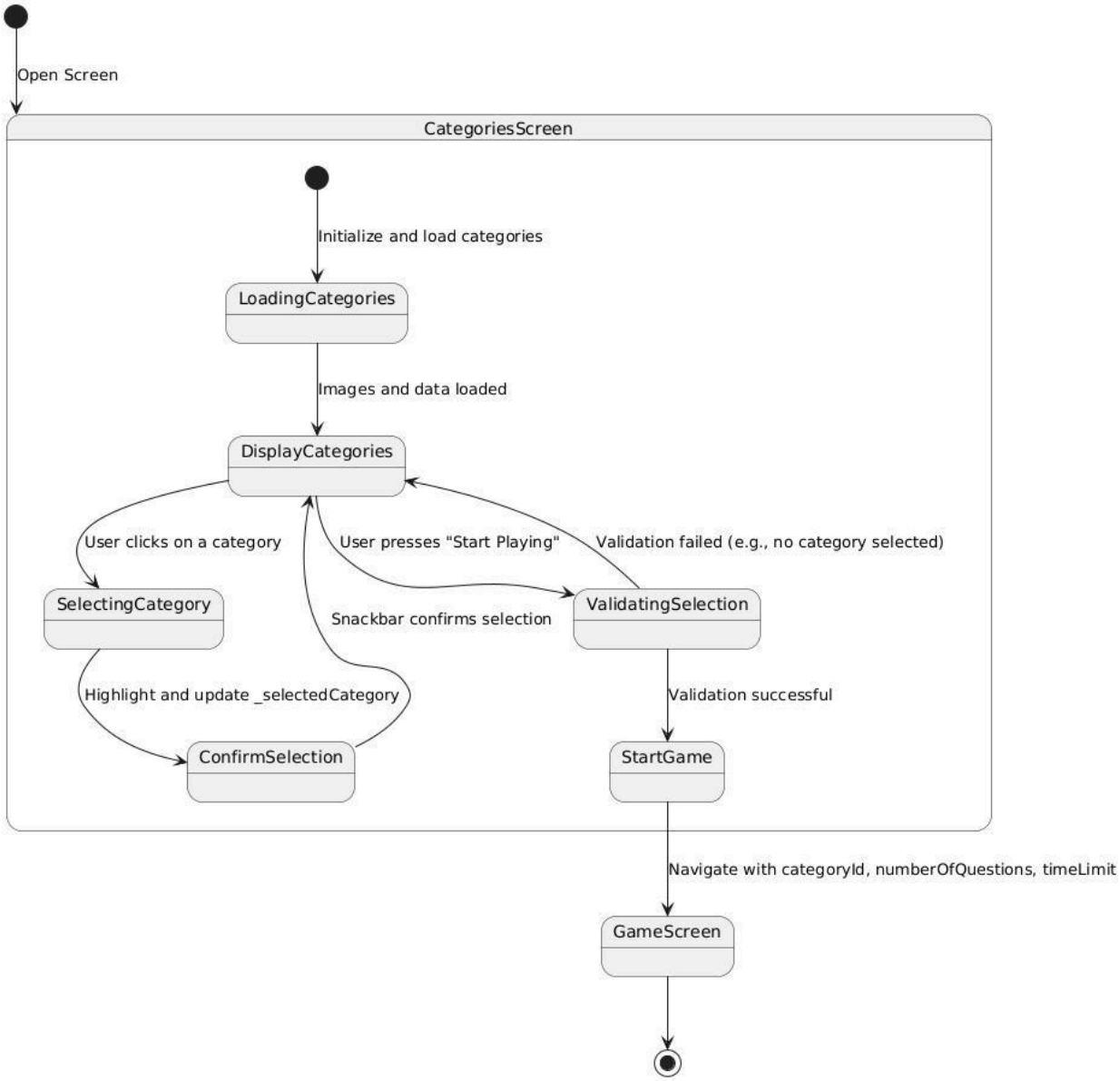


-diagrama de comunicare pentru functionaliatatea de game:

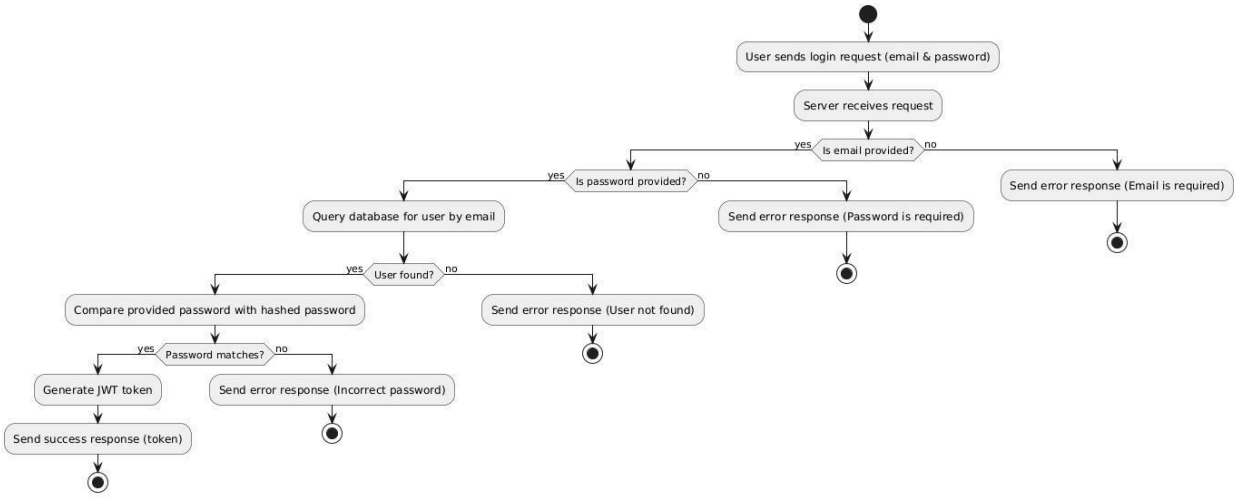


Tania: am realizat urmatoarele diagrame:

-diagrama de comunicare pentru categorii:

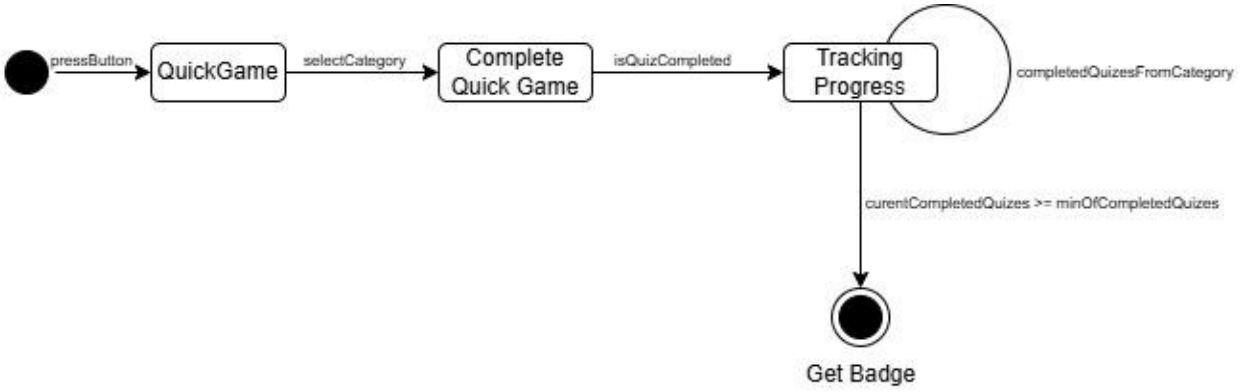


-diagrama de activitate pentru login:

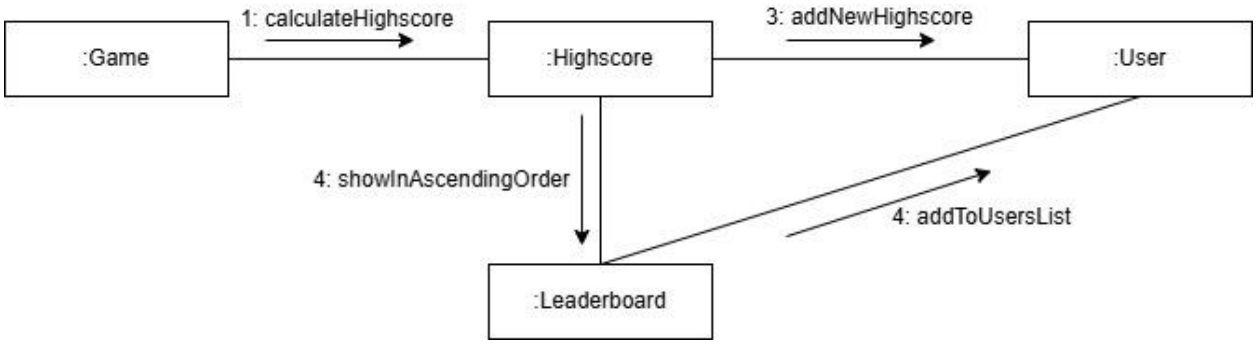


Ema: am realizat urmatoarele diagrame:

-diagrama de stari pentru obtinere badge:



-diagrama de comunicare pentru afisare scor



Design patterns

Roxana:

Am implemenentat design pattern-ul Builder pentru clasa de home_page deoarece ea are mai multe elemente la fel (de exemplu butoanele) iar folosirea acestui design pattern usureaza codul si logica din spatele acestuia.

Am avut nevoie sa creez clasa buttonBuilder care imi genereaza tipul specific de buton de care ma folosesc pentru crearea elementelor paginii.

```
class CustomButtonBuilder {
  final String text;
  final VoidCallback onPressed;
  final Color backgroundColor;
  final double borderRadius;
  final double width;
  final EdgeInsets padding;

  CustomButtonBuilder({
    required this.text,
    required this.onPressed,
    this.backgroundColor = const Color(0xFFE5A7EA),
    this.borderRadius = 20.0,
    this.width = 300.0,
    this.padding = const EdgeInsets.symmetric(vertical: 12.0),
  });

  Widget build() {
    return SizedBox(
      width: width,
      child: ElevatedButton(
        onPressed: onPressed,
        style: ElevatedButton.styleFrom(
```

Folosirea acestei clase:

```
Image.asset(
  'assets/images/logoCat.gif',
  height: 150,
), // Image.asset
const SizedBox(height: 5),
CustomButtonBuilder(
  text: 'Quick Game',
  onPressed: () {
    Navigator.pushNamed(context, '/categories');
  },
).build(), // CustomButtonBuilder
const SizedBox(height: 15),
CustomButtonBuilder(
  text: 'Challenge a Friend',
  onPressed: () {
    Navigator.pushNamed(context, '/challenge');
  },
).build(), // CustomButtonBuilder
const SizedBox(height: 15),
CustomButtonBuilder(
  text: 'Leaderboard',
  onPressed: () {
    Navigator.pushNamed(context, '/leaderboard');
  },
```

Tania:

Am implementat design pattern-ul State Management Pattern pentru clasa LeaderboardScreen, care este un StatefulWidget, ceea ce inseamna ca starea sa interna este modificata dinamic si se va schimba ui-ul, odata cu starea, cu ajutor de la setState, care notifică framework-ul Flutter că trebuie să reconstruiască UI-ul cu noua stare.

Email este starea care determină modificări în widget-urile UI, cum ar fi evidențierea utilizatorului curent în leaderboard.
Datele sunt obținute dintr-o sursă externă (serviciul LeaderboardService) şi actualizate în stare prin setState.

```dart
class _LeaderboardScreenState extends State<LeaderboardScreen> {
  final LeaderboardService _leaderboardService = LeaderboardService();
  String? email;

  @override
  void initState() {
    super.initState();
    _initializeUserProfile();
  }

  Future<void> _initializeUserProfile() async {
    final fetchedEmail = await _leaderboardService.getEmailFromPreferences();
    if (fetchedEmail != null) {
      setState(() {
        email = fetchedEmail;
      });
    }
```

Ema:

În Observer Pattern, implementarea pentru a primi notificări între clase este bazată pe relația dintre două concepte principale: "subiectul" (cel care emite notificări) și "observatorii" (cei care primesc notificările).

Subiectul este componenta care generează notificarea despre challenge-ul trimis. Aceasta este partea sistemului responsabilă pentru emiterea evenimentului de notificare.

Observatorii sunt utilizatorii care primesc notificarea. În acest caz, utilizatorul care primește challenge-ul este observatorul principal, deoarece el trebuie să fie notificat despre această acțiune pentru a putea răspunde sau reacționa.

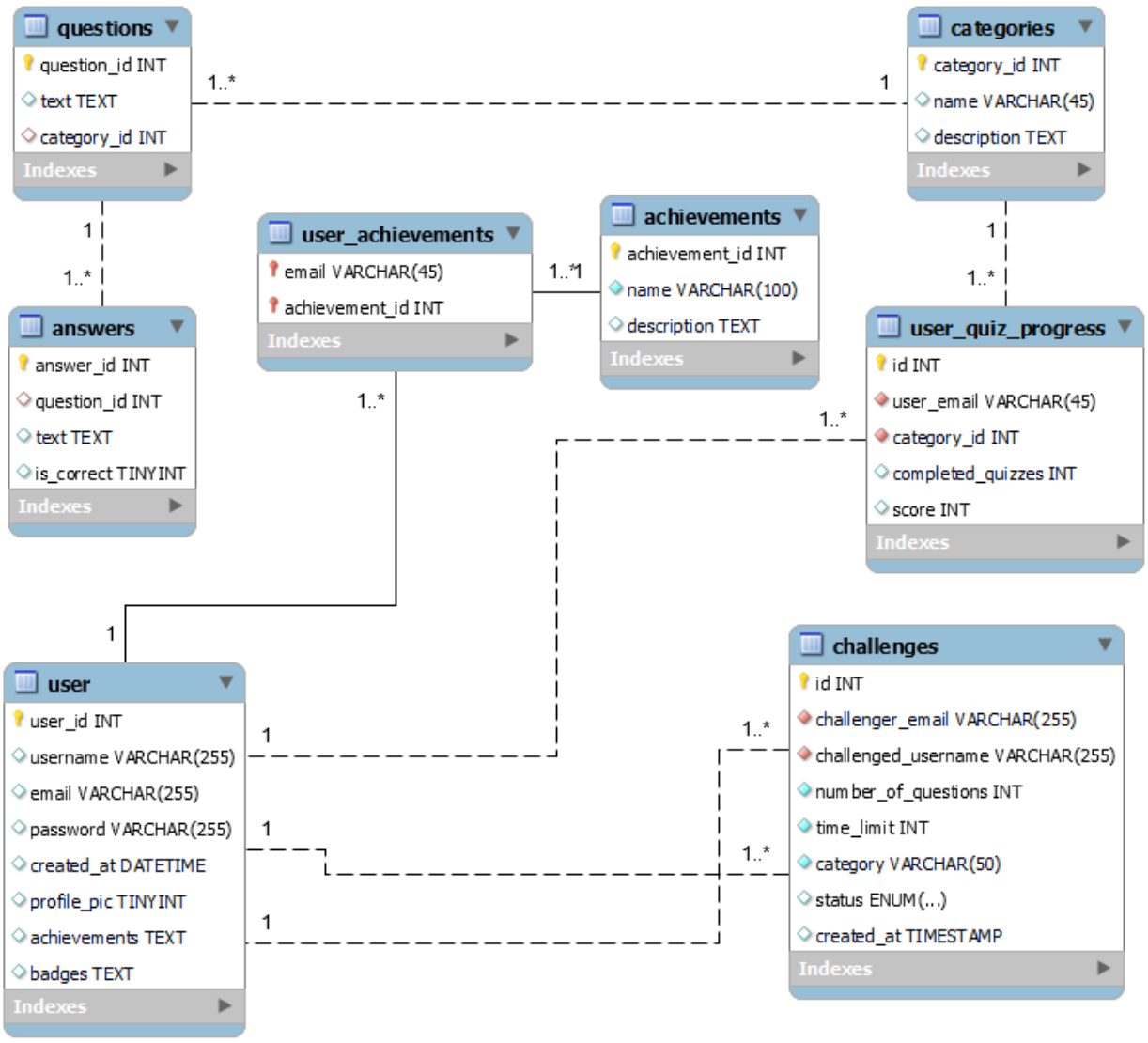
```dart
Future<List<Map<String, dynamic>>> fetchNotifications(String username) async {
  print('Fetching notifications for username: $username');
  try {
    final response = await http.post(
      Uri.parse('http://localhost:3000/getNotifications'),
      headers: {'Content-Type': 'application/json'},
      body: json.encode({'username': username}),
    );

    if (response.statusCode == 200) {
      final data = json.decode(response.body) as List<dynamic>;

      return data.map((notification) {
        return {
          'sender': notification['sender'] ?? '',
          'challengerEmail': notification['challengerEmail'] ?? '',
          'categoryName': notification['categoryName'] ?? '',
          'categoryId': notification['categoryId'] ?? 0,
          'numberOfQuestions': notification['numberOfQuestions'] ?? 0,
          'timeLimit': notification['timeLimit'] ?? 0,
        };
      }).toList();
    } else {
      print('Failed to fetch notifications: ${response.body}');
      return [];
    }
```

Baza de date

Am creat o baza de date in MySql pentru aplicatia noastra care contine urmatoarele tabele:

- user
- questions
- answers
- categories
- user_achievements
- user_quiz_progress
- achievements
- challenges

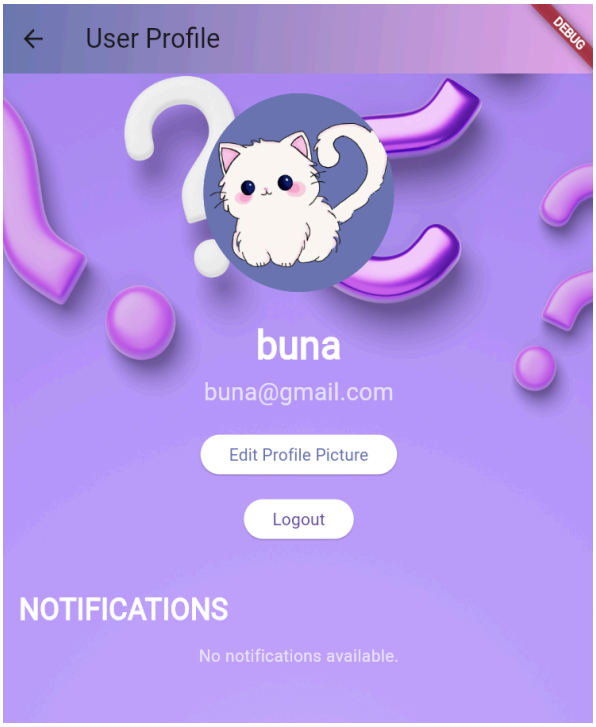
Diagrama interactiunii tabelelor este urmatoarea:

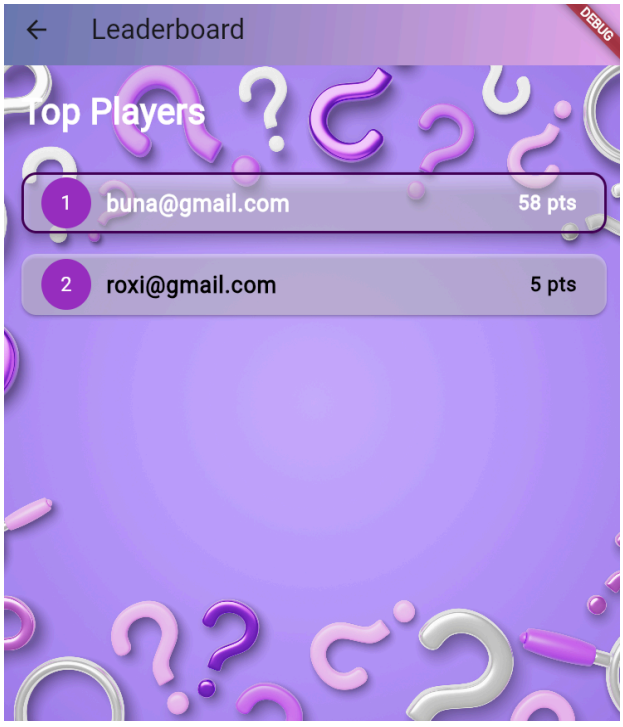


Aspect


Welcome Back!
DEBUG
PURRFECT TRIVIA
Quick Game
Challenge a Friend
Leaderboard


Trivia Game
DEBUG
Time Remaining: 04:54
Question 1: What does "API" stand for?
Automated Programming Interface
Algorithmic Programming Interface
Active Programming Interface
Application Programming Interface

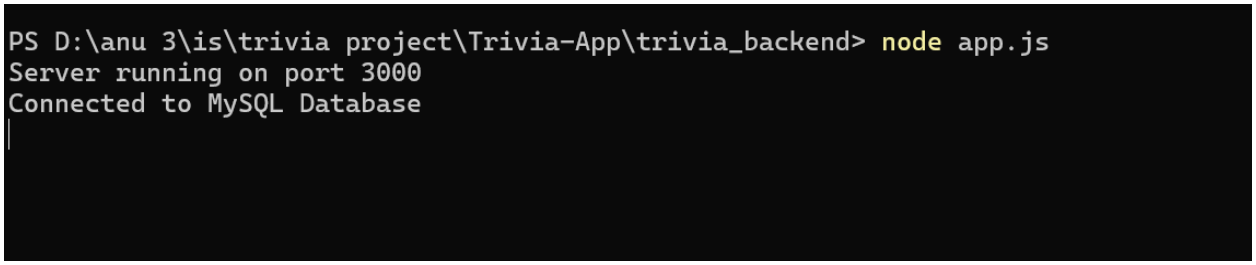

User Profile
DEBUG
buna
buna@gmail.com
Edit Profile Picture
Logout
NOTIFICATIONS
No notifications available.


Leaderboard
DEBUG
Top Players
1 buna@gmail.com 58 pts
2 roxi@gmail.com 5 pts

Resurse

Link-ul de github al proiectului: https://github.com/RoxanaRujac/Trivia-App
Am lucrat fiecare pe cate un branch separat, denumit corespunzator cu numele fiecareia.
Proiectul in varianta finala se gaseste pe branch-ul main.

Pentru a fi rulat trebuie intai pornit API-ul, din terminal, cu comanda “node app.js”.
Exemplu de pornire cu succes al API-ului:



Dupa acest pas, suntem conectati la baza de date si putem rula aplicatia din clasa “Main”.