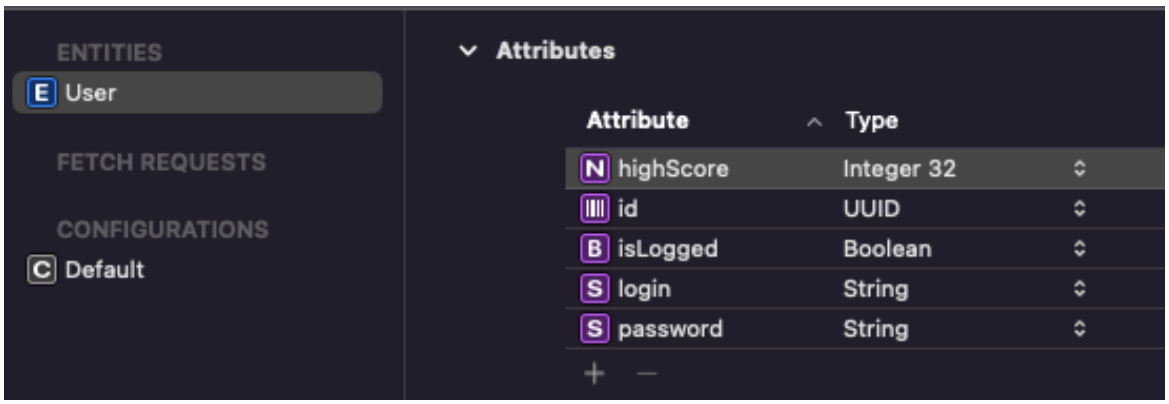
Quizzer App
Projekt

Opis projektu

Wstęp

Aplikacja wykorzystuje CoreData celem możliwości rejestracji i logowania użytkownika, oraz zapamiętywania najwyższego wyniku uzyskanego po rozwiązaniu Quiz'u, który jest generowany na podstawie API.

Core Data



Plik przedstawiający bazę danych

```swift
class DataController: ObservableObject {
    let container = NSPersistentContainer(name: "QuizzModel")

    init() {
        container.loadPersistentStores { description, error in
            if let error = error {
                print("Failed to laod data \(error.localizedDescription)")
            }

        }
    }   //  init()

    func save(context: NSManagedObjectContext) {
        do {
            try context.save()
            print("Data saved successfully!")
        }   catch {
            print("Data NOT saved!")
        }
    }   //  save()
```

Plik DataModel

Plik DataModel zawiera instrukcje kontrolujące dane w bazie danych Core Data. Zapisuje, wczytuje i pozwala na edycję danych.

```swift
import SwiftUI

@main
struct QuizzerApp: App {

    @StateObject private var dataController = DataController()

    var body: some Scene {
        WindowGroup {
            ContentView()
                .environment(\.managedObjectContext, dataController.container.viewContext)
        }
    }
}
```

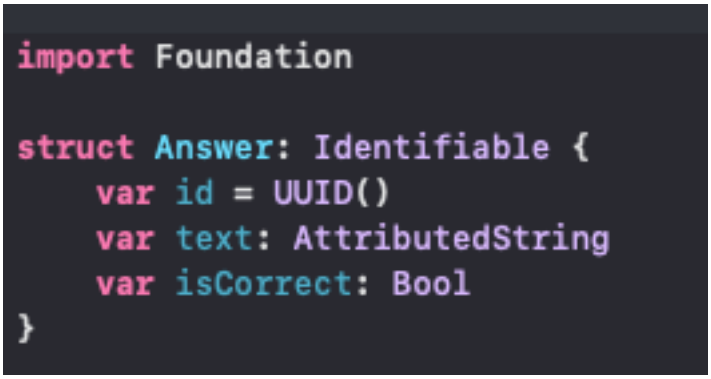
Podpięcie bazy danych do aplikacji

Modele

```swift
import Foundation

struct Quizz: Decodable {
    var results: [Result]

    struct Result: Decodable, Identifiable {
        var id: UUID {
            UUID()
        }
        var category: String
        var type: String
        var difficulty: String
        var question: String
        var correctAnswer: String
        var incorrectAnswers: [String]

        var formattedQuestion: AttributedString {
            do {
               return try AttributedString(markdown: question)
            }   catch {
                print("Error setting formattedQuestion: \(error)")
                return ""
            }
        }   //formattedQuestion

        var answers: [Answer] {
            do {
                let correct = [Answer(text: try AttributedString(markdown:
                    correctAnswer), isCorrect: true)]
                let incorrects = try incorrectAnswers.map { answer in
                    Answer(text: try AttributedString(markdown: answer), isCorrect:
                        false)
                }
                let allAnswers = correct + incorrects
                return allAnswers.shuffled()
            }   catch {
                print("Error setting answers: \(error)")
                return []
            }
        }   //answers
    }   //Result
}
```

Model Quiz

Model Quiz zawiera logikę głównej części aplikacji.



```
import Foundation

struct Answer: Identifiable {
    var id = UUID()
    var text: AttributedString
    var isCorrect: Bool
}
```

Model Answer

Model Answer pokazuje jakie zmienne musi zawierać obiekt Answer. Z tego obiektu korzysta plik AnswerRow, który tworzy wiersz każdej odpowiedzi w każdym pytaniu.

Widoki

ContentView to początkowy widok aplikacji, w którym użytkownik może się zalogować, bądź jeżeli nie ma konta przejść do widoku rejestracji. Pola są odpowiednio walidowane. RegisterView to widok, w którym można stworzyć konto. Jeżeli nazwa konta, którą podał użytkownik, znajduje się już w bazie danych to jest on o tym powiadamiany, a konto nie zostaje utworzone. QuizzView to widok, który sprawdza czy wszystkie pytania zostały rozwiązane i jeżeli tak to wyświetla wynik gracza, a jeżeli nie to wyświetla odpowiednie pytanie. QuestionView to widok pojedynczego pytania, które wyświetla się po rozpoczęciu gry. LeaderboardView pokazuje najlepsze wyniki wszystkich zarejestrowanych graczy, którzy mają wynik większy niż zero oraz pozwala na usuwanie tych wyników za pomocą gestu Swipe.

API

QuizManager odpowiada za obsługę API, z którego aplikacja pobiera pytania i odpowiedzi w formacie JSON. Funkcja fetchQuiz() rozpakowuje quiz do aplikacji asynchronicznie, co przyśpiesza działanie aplikacji. Kolejne funkcje służą do: rozpoznawania na, którym pytaniu znajduje się użytkownik lub, czy skończył quiz; ustawiania pytania; sprawdzania, czy zaznaczona odpowiedź jest prawidłowa.

```
class QuizzManager: ObservableObject {
    private(set) var quizz: [Quizz.Result] = []
    @Published private(set) var len = 0
    @Published private(set) var i = 0
    @Published private(set) var end = false
    @Published private(set) var answerSel = false
    @Published private(set) var question: AttributedString = ""
    @Published private(set) var answerChoices: [Answer] = []
    @Published private(set) var progress: CGFloat = 0.0
    @Published private(set) var score = 0

    init() {
        Task.init {
            await fetchQuizz()
        }
    }

    func fetchQuizz() async {
        guard let url = URL(string: "https://opentdb.com/api.php?amount=10") else {
            fatalError("Missing URL!")}

        let urlRequest = URLRequest(url: url)
        do {
            let (data, response) = try await URLSession.shared.data(for: urlRequest)
            guard (response as? HTTPURLResponse)?.statusCode == 200 else {
                fatalError("Error fetching data!")}
            let decoder = JSONDecoder()
            decoder.keyDecodingStrategy = .convertFromSnakeCase
            let decoded = try decoder.decode(Quizz.self, from: data)

            DispatchQueue.main.async {
                self.i = 0
                self.score = 0
                self.progress = 0.00
                self.end = false
                self.quizz = decoded.results
                self.len = self.quizz.count
                self.setQuestion()
            }
        }   catch {
            print("Error fetching quizz: \(error)")
        }
    }   //fetchQuizz()
```

Quiz Manager 1

```
    func nextQuestion() {
        if i+1 < len {
            i += 1
            setQuestion()
        }   else {
            end = true
        }
    }   //nextQuestion()

    func setQuestion() {
        answerSel = false
        progress = CGFloat(Double(i+1) / Double(len) * 350)
        if i < len {
            let currentQuizzQuestion = quizz[i]
            question = currentQuizzQuestion.formattedQuestion
            answerChoices = currentQuizzQuestion.answers
        }
    }   //setQuestion()

    func selectAnswer(answer: Answer) {
        answerSel = true
        if answer.isCorrect {
            score += 1
        }
    }   //select answer
}
```

Quiz Manager 2