Levando em consideração que as classes em Flutter funcionam de forma um pouco diferente, o presente guia tem como objetivo expor tudo que foi criado em cada arquivo explicando de modo sucinto cada método e classe criada em cada arquivo.

CORE DA APLICAÇÃO

main.dart

main()

A função main é uma função assíncrona que inicia alguns estados para o funcionamento do App.

Class MyApp

Essa classe estende um Stateless Widget, que constrói a tela inicial do app. Ainda construtor da tela, retorna um MultiProvider que busca um provedor de informações, para usuário e para o jogo. Tendo como base o Material App ela retorna em tela os routes e a home do app

Class Roteador Tela

Essa classe estende um Stateless Widget, que decide qual a primeira tela a ser mostrada com base na existência de um usuário logado

named_routes.dart

Aqui estão presentes todas as rotas da aplicação nomeadas, a fim de facilitar a chamada

routes.dart

Esse arquivo implementa a lógica para o direcionamento do usuário quando for solicitado a mudança de tela, seja por um botão ou outra ação que possa vir a acontecer no app, nele ainda, está elencado todas as rotas nomeadas e a chamada das classes que estendem as telas.

MODELS

- ## entities.dart

- **Class Entities{}**
  Esta classe representa uma entidade no Firestore.

- **Entities**
  Este construtor inicializa uma instância de Entities.

- **Entities.fromFirestore (Map<String, dynamic> data)**
  Construtor alternativo para criar uma instância de Entities a partir de dados do Firestore.

- **Map<String, dynamic> toJson()**
  Converte a instância de Entities para um mapa compatível com o Firestone.

- **static Future<Map<String,dynamic>?>getDocumentByName(String name)**
  Obtém um documento do Firestone pelo nome da entidade.

- **dynamic getProperty(String propertyName)**
  Obtém o valor de uma propriedade específica da entidade

- **game.dart**

- **Class Game{}**
  Esta classe tem como objetivo representar o jogo.

  int id: identificador único do jogo.

  Entities eotd: Representa a entidade do dia.

  int trys: Números de tentativas realizadas no jogo.

  int points: Pontuação obtida no jogo

- **Game()**
  Construtor que inicializa uma instância de Game.

- **factory Game.toMap(map)**
  Fábrica para criar uma instância de Game a partir de um mapa.

  Parâmetros:

  - **'map':** Mapa que contém os dados do jogo.

- **rest_client.dart**

  Classe abstrata usada para abstrair as funções padrões de uma API, todas têm- a mesma estrutura, que é Future<Map>//func//(String path, Map<String, dynamic>data);

  É uma função futura pois ela irá ser preenchida posteriormente, do tipo Map, pois o flutter não lida bem com Json, o nome da requisição(get, post, patch, delete) o caminho que foi definido na API e os dados que a rota retorna.

- **user.dart**
  Modelo usado para receber usuários que vem da API
  Attributes

  _documentId: str Faz referência ao documento do Firestore.

  email: str Email cadastrado no Firebase Auth e Firestore no momento do sign in.

  nick: str Nick escolhido pelo jogador quando efetuou o cadastro.

  gamesPlayed: int Quantidade de jogos que o jogador já participou.

  gamesWon: int Quantidade de jogos ganhos pelo jogador.

  maxStreak: int Quantidade máxima de jogos ganhos em seguida

points: int Quantidade de pontos que o usuário somou durante seus jogos.

streak: int Quantidade de jogos que o jogador vence em seguida.

winPercentage: double Taxa de vitória do jogador, durante seus jogos.

User.fromMap(DocumentSnapshot document, UserStatistics statistics) Método usado para povoar o documento de usuário no Firestore.

toMap() : Método usado para o Flutter conseguir entender como estão dispostos os dados no Json.

PROVIDERS

count_down.dart:

- **class CountdownTimer {}**

  Um temporizador regressivo que emite o tempo restante até a meia-noite seguinte.

- **CountdownTimer()**

  Inicializa uma nova instância da classe. Este construtor automaticamente inicializa os fusos horários, define o fuso horário desejado e calcula a próxima meia-noite e inicia o temporizador.

- **_calculateNextMidnight()**

  Calcula a próxima meia-noite no fuso horário local. Retorna um [TZDateTime] representando o momento da próxima meia-noite.

- **void _startTimer()**

  Inicia o temporizador para atualizar o tempo restante até a próxima meia-noite. Este método cria um [Timer] periódico que atualiza o tempo restante a cada segundo.

- **Stream<Duration> get stream => _timeRemainingController.stream;**

Obtém o fluxo de [Duration] representando o tempo restante. Este é um fluxo assíncrono que emite a diferença de tempo entre a próxima meia-noite e o momento atual.

- **void dispose()**

  Descarta o temporizador e fecha o controlador de fluxo. Este método deve ser chamado para liberar os recursos quando o temporizador não for mais necessário.

game_stats.dart:

- **class UserStatistics {**
  int totalGamesPlayed = 0;
  int totalGamesWon = 0;
  int winPercentage = 0;

int currentStreak = 0;
int maxStreak = 0;

Classe responsável por gerenciar as estatísticas do usuário.

- **UserStatistics()**

  Construtor da classe [UserStatistics]. Inicializa as estatísticas do usuário lendo os valores armazenados nas preferências compartilhadas.

- **void readStatsFromSharedPreferences() async**

  Lê as estatísticas do usuário das preferências compartilhadas. Se existirem estatísticas válidas, atualiza os campos da classe com os valores lidos.

- **void saveStatsToSharedPreferences() async**

  Salva as estatísticas do usuário nas preferências compartilhadas. Este método é chamado para persistir as estatísticas do usuário.

- **void displayStatistics(BuildContext context)**

  Exibe um diálogo mostrando as estatísticas do usuário. O contexto fornecido é usado para construir o diálogo com as estatísticas atuais do usuário.

- **void syncStatsWithFirestore(DocumentReference userDocument) async**

  Sincroniza as estatísticas do usuário com o Firestore. Atualiza as estatísticas do usuário com os valores armazenados no Firestore, se disponíveis.

- **void updateStatistics(bool gameWon, DocumentReference userDocument) async**

  Atualiza as estatísticas do usuário após um jogo. O parâmetro [gameWon] indica se o jogo foi vencido, e [userDocument] é a referência do documento do usuário no Firestore.

- **int calculatePoints(int attempts)**

  Calcula a pontuação com base na sequência atual de tentativas. Retorna a pontuação calculada com base na sequência atual de tentativas.

user_provider.dart:

- **class UserProvider with ChangeNotifier {**

  late String _nick;
  late String _email;
  late int _points;

  Classe que fornece informações do usuário e notifica ouvintes sobre mudanças.

- **String get getNick => _nick:** Obtém o nome do usuário.
- **String get getEmail => _email:** Obtém o email do usuário.

- **int get getPoints => _points:** Obtém os pontos do usuário.
- **void changeNickName(String val) {**
  _nick = val;
  notifyListeners();
  }

  Atualiza o nome do usuário e notifica os ouvintes sobre a mudança.
- **void updatingPoints(String val) {**

  _points = int.parse(val);
  notifyListeners();
  }

  Atualiza os pontos do usuário e notifica os ouvintes sobre a mudança.