Título de este proyecto de ayuda

Tabla de contenido

Introducción

Introduccion

Descripción de los Widgets

* Widget de Progreso Interactivo: Una barra de progreso circular con animaciones y opciones de personalización.

* Widget de Tarjeta De Presentacion: Una tarjeta que incluye una imagen, título, descripción y botones interactivos.

* Widget de Quiz Personalizado: Un widget diseñado para crear preguntas de opción múltiple, verdadero/falso, etc.

Loading Screen

Descripción General

El widget `LoadingScreen` es un componente visual de Flutter diseñado para mostrar diversas animaciones de carga. Este widget es personalizable, permitiendo al usuario seleccionar el tipo de animación de carga, así como los colores primarios y secundarios de la animación. Utiliza la arquitectura BLoC (Business Logic Component) para manejar el estado y las interacciones del usuario.

Uso

Para utilizar el widget `LoadingScreen`, es necesario integrarlo en la estructura de tu aplicación Flutter. El widget debe ser envuelto en un `BlocProvider` que provea la instancia del `LoadingScreenBloc`.

```
import 'package:excuela_widget_master/loading_screen/bloc/loading_screen_bloc.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'loading_screen.dart';

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
```

```
      home: BlocProvider(
        create: (_) => LoadingScreenBloc(),
        child: LoadingScreen(),
      ),
    );
  }
}
```

Personalización

El `LoadingScreen` permite varias opciones de personalización a través de su interfaz de usuario. A continuación se detallan las opciones disponibles:

1. **Tipo de Animación**:
   - `InkDrop`
   - `FourRotatingDots`
   - `DiscreteCircular`
   - `ThreeArchedCircle`
   - `Flickr`
   - `Beat`
   - `TwoRotatingArc`

   Para seleccionar el tipo de animación, se utiliza un `DropdownButton` en la interfaz que permite elegir entre los tipos de animación mencionados. La selección se maneja mediante eventos de BLoC.
2. **Color Primario del Widget**:
   - Rojo
   - Azul
   - Verde
   - Amarillo

   La selección del color primario se realiza a través de botones en la interfaz. Al seleccionar un color, se dispara un evento de BLoC para actualizar el estado con el nuevo color primario.
3. **Color Secundario del Widget**:
   - Negro
   - Naranja
   - Rosa
   - Morado

   Similar a la selección del color primario, los botones permiten elegir el color secundario del widget, y se actualiza el estado mediante eventos de BLoC.

Implementación

Widget `LoadingScreen`

El widget `LoadingScreen` utiliza el `BlocBuilder` para escuchar los cambios en el estado y actualizar la animación de carga y sus colores en consecuencia.

Constructor:

const LoadingScreen({super.key});

Métodos de Mapeo:

String mapTypeToString(LoadingAnimationType type) { ... }

LoadingAnimationType stringToMapType(String type) { ... }

Método `build`:
@override
Widget build(BuildContext context) { ... }

En el método `build`, se define la estructura del widget, incluyendo la barra de navegación (`AppBar`), el contenedor principal para la animación (`SizedBox`), y los controles de personalización (dropdown y botones).

Estado `LoadingScreenState`

El estado `LoadingScreenState` contiene la información necesaria para representar la animación de carga, incluyendo el tipo de animación, el color primario y el color secundario.

Enumeración `LoadingAnimationType`:

```
enum LoadingAnimationType {
  inkDrop,
  fourRotatingDots,
  discreteCircular,
  threeArchedCircle,
  flickr,
  beat,
  twoRotatingArc
}
```

BLoC `LoadingScreenBloc`

El `LoadingScreenBloc` maneja los eventos y actualiza el estado del `LoadingScreen`.

- Clase BLoC:

```
class LoadingScreenBloc extends Bloc<LoadingScreenEvent, LoadingScreenState> {
  LoadingScreenBloc() : super(LoadingScreenState.initial()) {
    on<LoadingScreenEvent>((event, emit) async {
      event.when(
        newAnimationSelected: (LoadingAnimationType newAnimationType) {
          emit(state.copyWith(loadingAnimationType: newAnimationType));
        },
        newPrimaryColorSelected: (Color newColor) {
          emit(state.copyWith(primaryColor: newColor));
        },
        newSecondaryColorSelected: (Color newColor) {
          emit(state.copyWith(secondaryColor: newColor));
        },
      );
    });
  }
```

```
}
```

Este BLoC responde a los eventos de selección de nueva animación (`newAnimationSelected`), nuevo color primario (`newPrimaryColorSelected`) y nuevo color secundario (`newSecondaryColorSelected`), actualizando el estado en consecuencia.

Conclusión

El widget `LoadingScreen` es altamente personalizable y utiliza la arquitectura BLoC para gestionar su estado. Los usuarios pueden seleccionar diferentes tipos de animaciones de carga y colores, lo que permite una integración flexible y adaptable en diversas aplicaciones de Flutter.

User Card

User Card Widget

Descripción

El widget **User Card** es una tarjeta de perfil de usuario que muestra información básica del usuario, como su nombre, correo electrónico, número de teléfono, ubicación y una imagen de perfil. Además, incluye enlaces interactivos para abrir el cliente de correo, LinkedIn, realizar una llamada telefónica y ver la ubicación en Google Maps. También permite mostrar información adicional y marcar al usuario como favorito.

Estado

El estado del widget **User Card** se maneja mediante un BLoC (`UserCardBloc`). El estado (`UserCardState`) incluye los siguientes campos:

- `isExpansionTileExpanded`: Indica si la sección de información adicional está expandida.
- `isContainerExpanded`: Indica si el contenedor está expandido.
- `name`: Nombre del usuario.
- `email`: Correo electrónico del usuario.
- `imageUrl`: URL de la imagen de perfil del usuario.
- `phoneNumber`: Número de teléfono del usuario.
- `location`: Ubicación del usuario.
- `additionalData`: Información adicional sobre el usuario.
- `isLiked`: Indica si el usuario ha sido marcado como favorito.

Ejemplo de uso

A continuación, se muestra un ejemplo de cómo utilizar el widget **User Card** en una aplicación Flutter:

```
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
```

```
import 'package:excuela_widget_master/user_card/bloc/user_card_bloc.dart';
import 'package:excuela_widget_master/user_card/user_card.dart';

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: BlocProvider(
        create: (_) => UserCardBloc(),
        child: const UserCard(),
      ),
    );
  }
}
```

El widget **User Card** se divide en varios subcomponentes para mantener el código organizado y modular. A continuación, se describen estos subcomponentes:

Subcomponentes

1. **UserCardProfileImage**: Muestra la imagen de perfil del usuario.
2. **UserCardPersonalInfoWithLinks**: Muestra la información personal del usuario junto con enlaces interactivos.
3. **UserCardShowMoreInformationWidget**: Widget para mostrar/ocultar información adicional.
4. **UserCardAdditionalInfo**: Muestra la información adicional del usuario.

Código del BLoC:

```
import 'package:bloc/bloc.dart';
import 'package:freezed_annotation/freezed_annotation.dart';

part 'user_card_event.dart';
part 'user_card_state.dart';
part 'user_card_bloc.freezed.dart';

class UserCardBloc extends Bloc<UserCardEvent, UserCardState> {
  UserCardBloc() : super(UserCardState.initial()) {
    on<UserCardEvent>((event, emit) async {
      await event.when(changeExpansionTileValue: (bool newValue) async {
        emit(state.copyWith(isExpansionTileExpanded: newValue));
```

```
        if (newValue) {
          await Future.delayed(const Duration(milliseconds: 300));
          emit(state.copyWith(isContainerExpanded: newValue));
        } else {
          emit(state.copyWith(isContainerExpanded: newValue));
        }
      }, changeLikeValue: (bool newValue) {
        emit(state.copyWith(isLiked: newValue));
      });
    });
  }
}
```


Quiz

Quiz Widget

Descripción

El widget Quiz es una interfaz de usuario interactiva diseñada para presentar preguntas y respuestas en forma de cuestionario. Facilita la interacción del usuario al permitirle responder preguntas y recibir retroalimentación inmediata sobre la corrección de sus respuestas.

Estado

El estado del widget Quiz se gestiona mediante un BLoC (QuizScreenBloc). El estado (QuizScreenState) incluye los siguientes campos:

- **data**: Datos del cuestionario, que incluyen preguntas y respuestas.
- **rightAnswerCount**: Contador de respuestas correctas.
- **wrongAnswerCount**: Contador de respuestas incorrectas.
- **currentQuestionIndex**: Índice de la pregunta actual.
- **currentUserSelectedOption**: Índice de la opción seleccionada por el usuario.
- **enableNextButton**: Indicador para habilitar el botón "Siguiente".
- **currentAnswerStatus**: Estado de la respuesta actual (vacío, correcta o incorrecta).
- **showMetrics**: Indica si se deben mostrar métricas al finalizar el cuestionario.

Ejemplo de uso

A continuación, se muestra un ejemplo básico de cómo utilizar el widget Quiz en una aplicación Flutter:

dart

```
    Copiar  código
import  'package:flutter/material.dart';
import  'package:flutter_bloc/flutter_bloc.dart';
import   'package:excuela_widget_master/quiz_screen/bloc/quiz_screen_bloc.dart';
import   'package:excuela_widget_master/quiz_screen/quiz_screen.dart';
```

```
void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: BlocProvider(
        create: (_) => QuizScreenBloc(),
        child: const QuizScreen(),
      ),
    );
  }
}
```

Subcomponentes

El widget Quiz se divide en varios subcomponentes para manejar diferentes partes de la interfaz:

- **AppBar**: Barra de título que muestra "Quiz".
- **Body**: Cuerpo principal que incluye la lista de preguntas y respuestas, botones de navegación y métricas finales.

Código del BLoC

El BLoC asociado al widget Quiz gestiona la lógica de negocio y el estado de la interfaz. A continuación, se muestra un ejemplo simplificado del BLoC (QuizScreenBloc):

dart

```
    Copiar código
import 'package:bloc/bloc.dart';
import 'package:freezed_annotation/freezed_annotation.dart';
import 'package:excuela_widget_master/quiz_screen/domain/quiz_data.dart';
import 'package:excuela_widget_master/quiz_screen/domain/quiz_question.dart';

part 'quiz_screen_event.dart';
part 'quiz_screen_state.dart';
part 'quiz_screen_bloc.freezed.dart';

class QuizScreenBloc extends Bloc<QuizScreenEvent, QuizScreenState> {
  QuizScreenBloc() : super(QuizScreenState.initial()) {
    on<QuizScreenEvent>((event, emit) {
      event.when(
        initialize: () {
           emit(state.copyWith(data: data, currentQuestionIndex: 0));
        },
        nextQuestion: () {
          if (state.currentQuestionIndex == data.questions.length - 1) {
            emit(state.copyWith(showMetrics: true));
          }
          emit(state.copyWith(
            currentQuestionIndex: state.currentQuestionIndex + 1,
            currentUserSelectedOption: null,
            currentAnswerStatus: CurrentQuestionAnswer.empty,
```

```
          ));
        },
         validateAnswer: (int questionIndex, int userAnswerIndex) {
           emit(state.copyWith(currentUserSelectedOption: userAnswerIndex));
          if (userAnswerIndex ==
              data.questions[questionIndex].correctAnswerIndex) {
            emit(state.copyWith(
              rightAnswerCount: state.rightAnswerCount + 1,
              currentAnswerStatus: CurrentQuestionAnswer.success,
              enableNextButton: true,
            ));
          } else {
            emit(state.copyWith(
              wrongAnswerCount: state.wrongAnswerCount + 1,
              currentAnswerStatus: CurrentQuestionAnswer.failure,
              enableNextButton: true,
            ));
          }
        },
      );
    });
  }

  final data = const QuizData(
    questions: [
      QuizQuestion(
        topic: 'Tecnología',
        question:
            '¿Cuál es el nombre de la pieza que actúa como el cerebro de una computadora?
        answers: ['Motherboard', 'CPU', 'GPU', 'RAM'],
        correctAnswerIndex: 1,
      ),
      // Otras preguntas...
    ],
    correctAnswers: 0,
    wrongAnswers: 0,
    percentage: 0,
  );
}
```

Código del Widget Principal

El widget principal (QuizScreen) se encarga de mostrar las preguntas, opciones de respuesta y controlar la navegación entre preguntas:

dart

```
    Copiar código
import  'package:flutter/material.dart';
import  'package:flutter_bloc/flutter_bloc.dart';
import   'package:excuela_widget_master/quiz_screen/bloc/quiz_screen_bloc.dart';

class QuizScreen extends StatelessWidget {
  const QuizScreen({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Quiz'),
```

```
        centerTitle: true,
      ),
      body: BlocBuilder<QuizScreenBloc, QuizScreenState>(
        builder: (context, state) {
          // Construcción dinámica de la interfaz basada en el estado
          return ListView.builder(
            itemCount: state.data.questions.length,
            itemBuilder: (context, index) {
              final question = state.data.questions[index];
              return Card(
                child: Column(
                  children: [
                    Text(question.question),
                    Column(
                      children: question.answers.map((answer) {
                        return RadioListTile(
                          title: Text(answer),
                          value: answer,
                          groupValue: state.currentUserSelectedOption,
                          onChanged: (value) {
                            // Lógica para manejar la selección de respuestas
                            // context.read<QuizScreenBloc>().add(...)
                          },
                        );
                      }).toList(),
                    ),
                    if (state.currentQuestionIndex == index)
                      ElevatedButton(
                        onPressed: () {
                          // Lógica para navegar a la siguiente pregunta
                          // context.read<QuizScreenBloc>().add(...)
                        },
                        child: const Text('Siguiente'),
                      ),
                  ],
                ),
              );
            },
          );
        },
      ),
    );
  }
}
```