Tienda online de bolsos - Práctica 3

Descripción general del proyecto

Esta aplicación permite a los usuarios explorar diferentes productos en una interfaz simple. En la aplicación se ha implementado persistencia entre páginas añadiendo funcionalidades como marcar como favoritos los productos, un selector de idioma y un selector de metodo de ordenación de la lista de productos. A continuación se explicarán las funciones implementadas.

Explicación de funciones implementadas

1. **Sistema de favorito**

Se ha creado una clase que maneja el funcionamiento del favorito mediante SharedPreferences.

```
Unset
class FavoritesManager {
  static const String _favoritesKey = 'favorites';

  static Future<void> toggleFavorite(String productId) async {
    final prefs = await SharedPreferences.getInstance();
    final favorites = prefs.getStringList(_favoritesKey) ??
[];

    if(favorites.contains(productId)) {
      favorites.remove(productId);
    }
    else {
      favorites.add(productId);
    }

    await prefs.setStringList(_favoritesKey, favorites);
  }

  static Future<List<String>> getFavorites() async {
```

```
    final prefs = await SharedPreferences.getInstance();
    return prefs.getStringList(_favoritesKey) ?? [];
  }

  static Future<bool> isFavorite(String productId) async {
    final favorites = await getFavorites();
    return favorites.contains(productId);
  }
}
```

Mediante la utilización de la clase tanto en la pagina de la lista de productos como en el detalle se puede controlar si un producto es marcado como favorito o no.

Página de detalle de productos

```
Unset
Future<void> _toggleFavorite(Product product) async {
    product.isFavorite = !product.isFavorite;
    await FavoritesManager.toggleFavorite(product.id);
    setState(() {});
  }
************************************************************
            IconButton(
                icon: Icon(
                  widget.product.isFavorite ? Icons.favorite :
Icons.favorite_border,
                  color: widget.product.isFavorite ? Colors.red :
Colors.white,
                ),
                onPressed: () => _toggleFavorite(widget.product))
```

Página de lista de productos

```
Unset
Future<void> _toggleFavorite(Product product) async {
    product.isFavorite = !product.isFavorite;
    await FavoritesManager.toggleFavorite(product.id);
    widget.parentState.loadProducts();
    setState(() {});
  }
```

```
onTap: () async {
        bool? isFavoriteChanged = await Navigator.of(context).push(
          MaterialPageRoute(
            builder: (ctx) => ProductDetailScreen(widget.product,
widget.parentState),
          ),
        );

        if(isFavoriteChanged != null && isFavoriteChanged !=
widget.product.isFavorite)
          {
            setState(() {
              widget.product.isFavorite = isFavoriteChanged;
            });

          }


        widget.parentState.loadProducts();
```

2. **Sistema de idoma**

Se ha creado la clase de LanguageManager para el control del idioma seleccionado. Mediante un archivo json se obtienen todas las traducciones necesarias para la aplicación tanto en español como en inglés.

```
Unset
import 'dart:convert';
import 'package:flutter/services.dart';

class LanguageManager
{
  static final LanguageManager _instance = LanguageManager._internal();
  factory LanguageManager() => _instance;

  LanguageManager._internal();

  Map<String, dynamic> _translations = {};
  String _currentLanguage = 'en';


  Future<void> loadTranslations(String languageCode) async
  {
```

```
    try {
      final String jsonString =
      await rootBundle.loadString('assets/language.json');
      final Map<String, dynamic> jsonData = json.decode(jsonString);
      _translations = jsonData[languageCode] ?? {};
      _currentLanguage = languageCode;
    } catch (e) {
      throw Exception('Error loading translations: $e');
    }
  }

  Future<void> changeLanguage(String languageCode) async
  {
    await loadTranslations(languageCode);
  }

  String translate(String key)
  {
    return _translations[key] ?? key;
  }

  String get currentLanguage => _currentLanguage;
}
```

Para la implementación de la persistencia entre páginas se ha escrito el siguiente código tanto en la página de detalles como en la página de la lista de productos

Lista de detalle de producto

```
Unset
final result = await Navigator.of(context).push(
                  MaterialPageRoute(
                    builder: (ctx) => ConfigurationScreen(),
                  ),
              );

              if(result != null)
                {
                  setState(() {
                    LanguageManager().changeLanguage(result['language']);
                    FilterManager().saveFilter(result['filter']);
                    // Si necesitas actualizar los productos basados en el
filtro.
                    widget.parentState.loadProducts();
                  });
                }
```

Página de lista de productos

```
Unset
final result = await Navigator.of(context).push(
              MaterialPageRoute(
                builder: (ctx) => ConfigurationScreen(),
              ),
            );

            if (result != null) {
              setState(() {
                LanguageManager().changeLanguage(result['language']);
                FilterManager().saveFilter(result['filter']);
                loadProducts(); // Actualiza la lista con el filtro
aplicado.
              });
            }
```

3. **Sistema de ordenación**

   Se ha implementado un sistema de ordenación para la lista de productos. Se puede ordenar alfabéticamente, por precio, por valoración y por favoritos. Todo esto se controla a partir de la clase FilterManager.

```
Unset
class FilterManager
{
  static final FilterManager _instance = FilterManager._internal();
  factory FilterManager() => _instance;

  FilterManager._internal();

  String _currentFilter = 'title';

  Future<void> loadFilter() async
  {
    final prefs = await SharedPreferences.getInstance();
    _currentFilter = prefs.getString('productFilter') ?? 'title';
  }

  Future<void> saveFilter(String filterCode) async
  {
    final prefs = await SharedPreferences.getInstance();
```

```
    await prefs.setString('productFilter', filterCode);
  }

  String get currentFilter => _currentFilter;
}
```

Para la implementación de la persistencia entre páginas se ha escrito el siguiente código tanto en la página de detalles como en la página de la lista de productos. Es el mismo proceso que en el apartado anterior pero en la página de la lista de productos se ordenan los productos según el método de filtrado seleccionado.

Página de lista de productos

Unset

```
Future<void> _loadFilter() async {
    await FilterManager().loadFilter();
    _applyFilter(FilterManager().currentFilter);
  }

  void _applyFilter(String filter){
    setState(() {
      switch(filter)
      {
        case 'alphabetic':
          _filteredProducts = [..._products]..sort((a, b) =>
a.title.compareTo(b.title));
          break;
        case 'price':
          _filteredProducts = [..._products]..sort((a, b) =>
a.price.compareTo(b.price));
          break;
        case 'rating':
          _filteredProducts = [..._products]..sort((a, b) =>
b.rating.compareTo(a.rating));
          break;
        case 'favorites':
          _filteredProducts = [..._products]..sort((a, b) => (b.isFavorite ?
1 : 0).compareTo(a.isFavorite ? 1: 0));
        default:
          _filteredProducts = [..._products];
      }
    });
```

4. **Página de Configuración**

En la página de configuración se ha implementado la posibilidad de cambiar la aplicación de idioma: “Español” o “Ingles” y seleccionar el metodo de ordenación que se quiera.

Unset

```
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:task2/local_persistence/FilterManager.dart';
import 'package:task2/local_persistence/LanguageManager.dart';

class ConfigurationScreen extends StatefulWidget {

  @override
  _ConfigurationScreenState createState() => _ConfigurationScreenState();
}

class _ConfigurationScreenState extends State<ConfigurationScreen> {
  late String _selectedLanguage;
  late String _selectedFilter;

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

  void _loadSettings()
  {
    _selectedLanguage = LanguageManager().currentLanguage;
    _selectedFilter = FilterManager().currentFilter;
    setState(() {});
  }

  Future<void> _changeLanguage(String language) async
  {
    setState(() {
      _selectedLanguage = language;
    });
    await LanguageManager().changeLanguage(language);
  }

  Future<void> _changeFilter(String filter) async
  {
    setState(() {
      _selectedFilter = filter;
    });
```

```
    await FilterManager().saveFilter(filter);
  }

  Future<bool> _onWillPop() async {
    Navigator.of(context).pop({
      'language' : _selectedLanguage,
      'filter' : _selectedFilter,
    });
    return false;
  }

  @override
  Widget build(BuildContext context) {
    final textTheme = Theme.of(context).textTheme;




    return WillPopScope(onWillPop: _onWillPop,
    child: Scaffold(
      appBar: AppBar(
        automaticallyImplyLeading: false,
        title: Text(LanguageManager().translate('configuration')),
      ),
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Text(LanguageManager().translate('language'),
              style: TextStyle(fontSize: 16, fontWeight: FontWeight.bold),),
            DropdownButton<String>(
              value: _selectedLanguage,
              items: [
                DropdownMenuItem(
                    value: 'en',
                    child:
Text(LanguageManager().translate('language_english'))
                ),
                DropdownMenuItem(
                    value: 'es',
                    child:
Text(LanguageManager().translate('language_spanish'))
                ),
              ],
              onChanged: (newValue) {
                if(newValue != null) _changeLanguage(newValue);
              },
```

```
            ),
            SizedBox(height: 20,),
            Text(LanguageManager().translate('sort_products'),
              style: TextStyle(fontSize: 16, fontWeight: FontWeight.bold),
            ),
            DropdownButton<String>(
              value: _selectedFilter,
              items: [
                DropdownMenuItem(
                  value : 'title',
                  child:
Text(LanguageManager().translate('sort_by_alphabetic')),
                ),
                DropdownMenuItem(
                  value : 'price',
                  child: Text(LanguageManager().translate('sort_by_price')),
                ),
                DropdownMenuItem(
                  value : 'rating',
                  child:
Text(LanguageManager().translate('sort_by_rating')),
                ),
                DropdownMenuItem(
                  value : 'favorites',
                  child:
Text(LanguageManager().translate('sort_by_favorites')),
                )

              ],
              onChanged: (newValue) {
                if(newValue != null) _changeFilter(newValue);
              },
            ),
            const SizedBox(height: 20,),
            const Divider(),
            FilledButton(
              style: FilledButton.styleFrom(
                minimumSize: const Size(double.infinity, 48.0),
              ),
              onPressed: () {
                Navigator.of(context).pop({
                  'language' : _selectedLanguage,
                  'filter' : _selectedFilter,
                });
              },
              child: Text(
                '${LanguageManager().translate('back')}',
                style: textTheme.titleMedium!.copyWith(
```

```
                  color: Colors.white,
                  fontWeight: FontWeight.bold,
                ),
              ),
            ),
          ],
        ),
      ),
    )
    );
  }
}
```

Del mismo modo en el que se ha explicado en el apartado de la implementación de idiomas, para mantener la persistencia se ha tenido en cuenta, cualquier cambio en cualquiera de los dos apartados ya sea idioma o filtro.

```
Unset
setState(() {
                LanguageManager().changeLanguage(result['language']);
                FilterManager().saveFilter(result['filter']);
                loadProducts(); // Actualiza la lista con el filtro
aplicado.
              });
```

Como se puede observar, este trozo de código se ejecuta al volver de la página de configuración, incluso haciendo Pop.