Lección 3: Diseño de la Interfaz de Usuario (UI) con Soporte para Modo Claro y Oscuro

Objetivos de la Lección

- **Diseñar** una interfaz de usuario moderna y amigable para la aplicación móvil.
- **Implementar** widgets de Flutter para crear pantallas atractivas y funcionales.
- **Añadir** soporte para alternar entre modo claro y oscuro, y persistir la preferencia del usuario.
- **Personalizar** el tema de la aplicación utilizando `ThemeData` y estilos personalizados.
- **Mejorar** la experiencia del usuario con una interfaz similar a las redes sociales populares.

Introducción a la Lección

Una interfaz de usuario bien diseñada es crucial para el éxito de cualquier aplicación móvil. En esta lección, nos enfocaremos en crear una interfaz moderna y atractiva para nuestra aplicación de red social básica. Utilizaremos los widgets de Flutter para construir una experiencia de usuario intuitiva y agradable, inspirándonos en las redes sociales populares.

Además, implementaremos la funcionalidad para que el usuario pueda alternar entre el modo claro y oscuro, y almacenaremos su preferencia utilizando `shared_preferences`. Esto no solo mejora la experiencia del usuario, sino que también demuestra cómo persistir datos en la aplicación.

Desarrollo de Conceptos

Personalización del Tema en Flutter

Flutter ofrece una gran flexibilidad para personalizar el aspecto de las aplicaciones. Podemos definir temas globales que afecten a toda la aplicación o aplicar estilos a widgets específicos.

Modo Claro y Oscuro

El modo oscuro es una característica popular que reduce la fatiga visual y ahorra batería en dispositivos con pantallas OLED. Implementar soporte para ambos modos mejora la accesibilidad y personalización.

Persistencia de Datos con shared_preferences

El paquete `shared_preferences` permite almacenar datos simples en el dispositivo, como las preferencias del usuario. Es ideal para guardar configuraciones que deben persistir entre sesiones de la aplicación.

Secciones Técnicas Específicas

1. Personalización del Tema de la Aplicación

Paso 1: Definir Temas Personalizados

En **lib/main.dart**, podemos definir temas más personalizados.

```
// lib/main.dart
import 'package:flutter/material.dart';
import 'screens/home_screen.dart';

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

```
}

class MyApp extends StatefulWidget {
  // Este widget es la raíz de la aplicación.
  @override
  _MyAppState createState() => _MyAppState();
}

class _MyAppState extends State<MyApp> {
  ThemeMode _themeMode = ThemeMode.system;

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

  // Cargar la preferencia de tema almacenada
  void _loadThemePreference() async {
    // Implementaremos esto en el siguiente paso
  }

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Red Social Básica',
      theme: ThemeData(
        brightness: Brightness.light,
        primarySwatch: Colors.blue,
        accentColor: Colors.blueAccent,
        appBarTheme: AppBarTheme(
          color: Colors.white,
          iconTheme: IconThemeData(color: Colors.blue),
          textTheme: TextTheme(
```

```
            headline6: TextStyle(color: Colors.blue, fontSize:
20),
          ),
        ),
      ),
      darkTheme: ThemeData(
        brightness: Brightness.dark,
        primarySwatch: Colors.blue,
        accentColor: Colors.blueAccent,
      ),
      themeMode: _themeMode, // Modo de tema seleccionado
      home: HomeScreen(
        onThemeChanged: _toggleThemeMode,
      ),
    );
  }

  void _toggleThemeMode(ThemeMode mode) {
    setState(() {
      _themeMode = mode;
    });
    _saveThemePreference(mode);
  }

  // Guardar la preferencia de tema
  void _saveThemePreference(ThemeMode mode) async {
    // Implementaremos esto en el siguiente paso
  }
}
```

Paso 2: Personalizar el AppBar y Colores

En los temas, hemos personalizado el `AppBarTheme` para cambiar el color de fondo y los iconos.

2. Implementar Persistencia de Preferencias con shared_preferences

Paso 1: Importar el Paquete

Asegúrate de que `shared_preferences` esté en el archivo `pubspec.yaml` y ejecuta `flutter pub get` si es necesario.

```
dependencies:
  flutter:
    sdk: flutter
  http: ^0.13.4
  shared_preferences: ^2.0.7
```

Paso 2: Importar `shared_preferences` en `main.dart`

```
import 'package:shared_preferences/shared_preferences.dart';
```

Paso 3: Implementar Métodos para Guardar y Cargar Preferencias

```
// En _MyAppState

// Cargar la preferencia de tema almacenada
void _loadThemePreference() async {
  SharedPreferences prefs = await
SharedPreferences.getInstance();
  int? themeIndex = prefs.getInt('themeMode');
  setState(() {
    _themeMode = ThemeMode.values[themeIndex ?? 0];
  });
}

// Guardar la preferencia de tema
void _saveThemePreference(ThemeMode mode) async {
  SharedPreferences prefs = await
SharedPreferences.getInstance();
  await prefs.setInt('themeMode', mode.index);
```

```
}
```

Paso 4: Probar la Persistencia

- Ejecuta la aplicación.
- Cambia entre modos de tema.
- Cierra y vuelve a abrir la aplicación para verificar que la preferencia se ha guardado.

3. Mejorar el Diseño de la Interfaz de Usuario

3.1. Rediseñar la Pantalla Principal (`HomeScreen`)

Actualizaremos la interfaz para que se asemeje más a una red social.

```
// lib/screens/home_screen.dart
import 'package:flutter/material.dart';
import '../models/post.dart';
import '../services/api_service.dart';
import 'post_detail_screen.dart';
import 'users_screen.dart';

class HomeScreen extends StatefulWidget {
  final Function(ThemeMode) onThemeChanged;

  HomeScreen({required this.onThemeChanged});

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

class _HomeScreenState extends State<HomeScreen> {
  ApiService apiService = ApiService();
  late Future<List<Post>> futurePosts;

  @override
```

```
  void initState() {
    super.initState();
    futurePosts = apiService.getPosts();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Red Social'),
        actions: [
          IconButton(
            icon: Icon(Icons.people),
            onPressed: () {
              Navigator.push(
                context,
                MaterialPageRoute(builder: (context) =>
UsersScreen()),
              );
            },
          ),
          PopupMenuButton<ThemeMode>(
            onSelected: widget.onThemeChanged,
            itemBuilder: (context) => [
              PopupMenuItem(
                value: ThemeMode.system,
                child: Text('Predeterminado'),
              ),
              PopupMenuItem(
                value: ThemeMode.light,
                child: Text('Modo Claro'),
              ),
              PopupMenuItem(
                value: ThemeMode.dark,
                child: Text('Modo Oscuro'),
```

```
            ),
          ],
        ),
      ],
    ),
    body: FutureBuilder<List<Post>>(
      future: futurePosts,
      builder: (context, snapshot) {
        if (snapshot.hasData) {
          List<Post> posts = snapshot.data!;
          return ListView.builder(
            itemCount: posts.length,
            itemBuilder: (context, index) {
              return _buildPostCard(posts[index]);
            },
          );
        } else if (snapshot.hasError) {
          return Center(child: Text('Error:
${snapshot.error}'));
        }
        return Center(child: CircularProgressIndicator());
      },
    ),
  );
}

Widget _buildPostCard(Post post) {
  return Card(
    margin: EdgeInsets.symmetric(vertical: 8, horizontal: 16),
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        ListTile(
          title: Text(
            post.title,
```

```
              style: TextStyle(fontWeight: FontWeight.bold),
            ),
            subtitle: Text('Autor ID: ${post.userId}'),
            onTap: () {
              Navigator.push(
                context,
                MaterialPageRoute(builder: (context) =>
PostDetailScreen(post: post)),
              );
            },
          ),
          Padding(
            padding: EdgeInsets.symmetric(horizontal: 16),
            child: Text(post.body),
          ),
          ButtonBar(
            alignment: MainAxisAlignment.start,
            children: [
              TextButton.icon(
                onPressed: () {
                  // Funcionalidad de Me Gusta (simulada)
                },
                icon: Icon(Icons.thumb_up),
                label: Text('Me Gusta'),
              ),
              TextButton.icon(
                onPressed: () {
                  // Funcionalidad de Compartir (simulada)
                },
                icon: Icon(Icons.share),
                label: Text('Compartir'),
              ),
            ],
          ),
        ],
```

```
      ),
    );
  }
}
```

Explicación

- **Card**: Utilizamos el widget `Card` para darle un aspecto de publicación de red social.
- **ListTile**: Muestra el título y autor de la publicación.
- **ButtonBar**: Añade botones para "Me Gusta" y "Compartir" (funcionalidades simuladas).
- **Estilo**: Mejoramos la estética con padding y estilos de texto.

3.2. Añadir Imágenes de Perfil y Publicaciones

Aunque los datos de JSONPlaceholder no incluyen imágenes, podemos utilizar imágenes genéricas.

- **Avatar del Usuario**: Usaremos el widget `CircleAvatar` en el `ListTile`.

```
ListTile(
  leading: CircleAvatar(
    child: Text(post.userId.toString()),
  ),
  title: Text(
    post.title,
    style: TextStyle(fontWeight: FontWeight.bold),
  ),
  // ...
),
```

- **Imagen en la Publicación**: Podemos añadir una imagen de placeholder.

```
// Después del ListTile y antes del Padding
Image.network('https://via.placeholder.com/150'),
```

4. Rediseñar la Pantalla de Detalles de Publicación

En **lib/screens/post_detail_screen.dart**, mejoramos la interfaz.

```
// Añadimos estilos y organizamos mejor el contenido
class PostDetailScreen extends StatefulWidget {
  // ...
}

class _PostDetailScreenState extends State<PostDetailScreen> {
  // ...
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Publicación'),
      ),
      body: SingleChildScrollView(
        child: Column(
          children: [
            // Información de la publicación
            ListTile(
              leading: CircleAvatar(
                child: Text(widget.post.userId.toString()),
              ),
              title: Text(
                widget.post.title,
                style: TextStyle(fontWeight: FontWeight.bold),
              ),
```

```
              subtitle: Text('Autor ID: ${widget.post.userId}'),
            ),
            Padding(
              padding: EdgeInsets.all(16),
              child: Text(widget.post.body),
            ),
            Divider(),
            // Comentarios
            Padding(
              padding: EdgeInsets.all(16),
              child: Text('Comentarios', style:
TextStyle(fontSize: 18)),
            ),
            _buildCommentsSection(),
          ],
        ),
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: () {
          // Navegar a la pantalla para añadir un comentario
        },
        child: Icon(Icons.add_comment),
      ),
    );
  }

  Widget _buildCommentsSection() {
    return FutureBuilder<List<Comment>>(
      future: futureComments,
      builder: (context, snapshot) {
        if (snapshot.hasData) {
          List<Comment> comments = snapshot.data!;
          return ListView.builder(
            shrinkWrap: true,
            physics: NeverScrollableScrollPhysics(),
```

```
            itemCount: comments.length,
            itemBuilder: (context, index) {
              return ListTile(
                leading: Icon(Icons.comment),
                title: Text(comments[index].name),
                subtitle: Text(comments[index].body),
              );
            },
          );
        } else if (snapshot.hasError) {
          return Center(child: Text('Error:
${snapshot.error}'));
        }
        return Center(child: CircularProgressIndicator());
      },
    );
  }
}
```

5. Mejorar la Pantalla de Usuarios

En **lib/screens/users_screen.dart**, añadiremos más detalles.

```
// En ListTile
ListTile(
  leading: CircleAvatar(
    child: Text(users[index].id.toString()),
  ),
  title: Text(users[index].name),
  subtitle: Text('@${users[index].username}'),
  onTap: () {
```

```
    // Navegar al perfil del usuario
  },
),
```

6. Añadir la Pantalla de Perfil de Usuario

Creamos **lib/screens/user_profile_screen.dart**.

```
import 'package:flutter/material.dart';
import '../models/user.dart';

class UserProfileScreen extends StatelessWidget {
  final User user;

  UserProfileScreen({required this.user});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(user.name),
      ),
      body: Column(
        children: [
          SizedBox(height: 20),
          CircleAvatar(
            radius: 50,
            child: Text(user.name.substring(0, 1)),
          ),
          SizedBox(height: 20),
          Text(
            user.name,
```

```
            style: TextStyle(fontSize: 24, fontWeight:
FontWeight.bold),
          ),
          Text('@${user.username}'),
          SizedBox(height: 20),
          ListTile(
            leading: Icon(Icons.email),
            title: Text(user.email),
          ),
          // Puedes añadir más información como dirección,
teléfono, etc.
        ],
      ),
    );
  }
}
```

En **UsersScreen**, modificamos el onTap:

```
onTap: () {
  Navigator.push(
    context,
    MaterialPageRoute(builder: (context) =>
UserProfileScreen(user: users[index])),
  );
},
```

7. Implementar el Cambio de Modo Claro y Oscuro en Todas las Pantallas

Asegúrate de que el cambio de tema afecte a todas las pantallas. Si es necesario, pasa la función onThemeChanged a las demás pantallas.

8. Mejorar la Experiencia de Usuario

- **Animaciones**: Añade animaciones sutiles al navegar entre pantallas.
- **Feedback**: Utiliza `InkWell` o `GestureDetector` para dar feedback táctil al usuario.
- **Consistencia**: Mantén una paleta de colores y estilos consistentes en toda la aplicación.

Ejemplos en Código

Ejemplo: Uso de shared_preferences para Persistir Preferencias

```
void _loadThemePreference() async {
  SharedPreferences prefs = await
SharedPreferences.getInstance();
  int? themeIndex = prefs.getInt('themeMode');
  setState(() {
    _themeMode = ThemeMode.values[themeIndex ?? 0];
  });
}

void _saveThemePreference(ThemeMode mode) async {
  SharedPreferences prefs = await
SharedPreferences.getInstance();
  await prefs.setInt('themeMode', mode.index);
}
```

Explicación:

- **SharedPreferences**: Permite guardar pares clave-valor de manera persistente.
- **themeMode.index**: Convierte el `ThemeMode` en un entero para almacenarlo.
- **ThemeMode.values[themeIndex]**: Recupera el `ThemeMode` correspondiente al índice almacenado.

Relación con Otros Temas

Esta lección se relaciona con:

- **Programación Orientada a Objetos**: Al diseñar componentes reutilizables y estructurar la interfaz.
- **Desarrollo de Interfaces (UX/UI)**: Al aplicar principios de diseño y mejorar la experiencia del usuario.
- **Persistencia de Datos**: Utilizando `shared_preferences` para guardar configuraciones del usuario.
- **Temas Anteriores**: Construye sobre las lecciones anteriores al mejorar y ampliar la funcionalidad de la aplicación.

Resumen de la Lección

En esta lección, hemos diseñado una interfaz de usuario moderna y atractiva para nuestra aplicación de red social básica. Implementamos el soporte para modo claro y oscuro, y persistimos la preferencia del usuario utilizando `shared_preferences`. Mejoramos la experiencia del usuario al diseñar las pantallas principales, añadir elementos visuales como imágenes y avatares, y mantener una estética consistente en toda la aplicación.

Actividad de la Lección

Objetivo de la Actividad:

- Demostrar que puedes diseñar y mejorar la interfaz de usuario de una aplicación Flutter.
- Verificar que has implementado correctamente el soporte para modo claro y oscuro, y que la preferencia del usuario se persiste.
- Prepararte para las siguientes lecciones, donde añadiremos funcionalidades adicionales como login y manejo de comentarios.

Tarea Práctica:

1. **Personaliza el tema de la aplicación** siguiendo los ejemplos proporcionados, asegurándose de que el modo claro y oscuro funcionen correctamente.
2. **Implementa la persistencia de la preferencia de tema** utilizando `shared_preferences`.
3. **Rediseña las pantallas** de publicaciones, detalles de publicación y usuarios para mejorar la interfaz de usuario.
4. **Añade imágenes y avatares** a las publicaciones y perfiles de usuario.
5. **Prueba la aplicación** en un emulador o dispositivo físico para verificar que los cambios se reflejan correctamente.
6. **Documenta tu proceso**: Crea un documento en PDF que incluya capturas de pantalla antes y después de los cambios, y explica las mejoras realizadas.
7. **Entrega**: Sube el PDF y el código fuente actualizado del proyecto en una carpeta comprimida.