Lección 4: Implementación de Funcionalidades de Login y Persistencia de Preferencias

Objetivos de la Lección

- **Implementar** una pantalla de login que permita a los usuarios autenticarse utilizando su email y username.
- **Persistir** la sesión del usuario para mantenerlo autenticado entre sesiones.
- **Actualizar** la aplicación para mostrar contenido personalizado según el usuario autenticado.
- **Mantener** la persistencia de las preferencias de tema y asegurar su correcto funcionamiento con el sistema de login.
- **Mejorar** la experiencia de usuario al integrar la autenticación y personalización en la aplicación.

Introducción a la Lección

En esta lección, nos enfocaremos en añadir funcionalidades de autenticación a nuestra aplicación de red social básica. Permitiremos que los usuarios inicien sesión utilizando su **email** como nombre de usuario y su **username** como contraseña, basándonos en los datos proporcionados por el endpoint `/users` de **JSONPlaceholder**.

Implementaremos la persistencia de la sesión para que el usuario no tenga que iniciar sesión cada vez que abra la aplicación. Además, aseguraremos que las preferencias de tema (modo claro y oscuro) sigan funcionando correctamente y se mantengan almacenadas.

Esta lección es crucial para convertir nuestra aplicación en una herramienta más personalizada y funcional, brindando una experiencia más cercana a una red social real.

Desarrollo de Conceptos

Autenticación en Aplicaciones Móviles

La autenticación es el proceso de verificar la identidad de un usuario. En aplicaciones móviles, es común que los usuarios inicien sesión para acceder a contenido personalizado y funcionalidades específicas.

Persistencia de Sesión

La persistencia de sesión permite que un usuario permanezca autenticado incluso después de cerrar y reabrir la aplicación. Esto mejora la experiencia del usuario al evitar que tenga que iniciar sesión repetidamente.

Uso de `shared_preferences` para Datos Simples

El paquete `shared_preferences` es ideal para almacenar datos simples y persistir información como tokens de autenticación, preferencias de usuario y configuraciones.

Secciones Técnicas Específicas

1. Implementación de la Pantalla de Login

Paso 1: Crear el Archivo `login_screen.dart`

En **lib/screens/**, crea un nuevo archivo llamado **`login_screen.dart`**.

```
// lib/screens/login_screen.dart
import 'package:flutter/material.dart';
import 'package:shared_preferences/shared_preferences.dart';
import '../services/api_service.dart';
import '../models/user.dart';
import 'home_screen.dart';
```

```
class LoginScreen extends StatefulWidget {
  final Function(ThemeMode) onThemeChanged;

  LoginScreen({required this.onThemeChanged});

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

class _LoginScreenState extends State<LoginScreen> {
  final _formKey = GlobalKey<FormState>();
  String _email = '';
  String _password = '';
  bool _isLoading = false;
  ApiService apiService = ApiService();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Iniciar Sesión'),
        actions: [
          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: Padding(
        padding: EdgeInsets.all(16),
        child: _isLoading
            ? Center(child: CircularProgressIndicator())
            : Form(
                key: _formKey,
                child: Column(
                  children: [
                    TextFormField(
                      decoration: InputDecoration(labelText:
'Correo Electrónico'),
                      keyboardType: TextInputType.emailAddress,
```

```
                    validator: (value) {
                      if (value == null || value.isEmpty) {
                        return 'Por favor ingresa tu correo
electrónico';
                      }
                      return null;
                    },
                    onSaved: (value) {
                      _email = value!.trim();
                    },
                  ),
                  TextFormField(
                    decoration: InputDecoration(labelText:
'Nombre de Usuario'),
                    obscureText: true,
                    validator: (value) {
                      if (value == null || value.isEmpty) {
                        return 'Por favor ingresa tu nombre de
usuario';
                      }
                      return null;
                    },
                    onSaved: (value) {
                      _password = value!.trim();
                    },
                  ),
                  SizedBox(height: 20),
```

```
                    ElevatedButton(
                      onPressed: _login,
                      child: Text('Iniciar Sesión'),
                    ),
                  ],
                ),
              ),
            ),
        );
    }

    void _login() async {
      if (_formKey.currentState!.validate()) {
        _formKey.currentState!.save();
        setState(() {
          _isLoading = true;
        });
        try {
          List<User> users = await apiService.getUsers();
          User? user = users.firstWhere(
            (user) => user.email.toLowerCase() ==
_email.toLowerCase() && user.username == _password,
            orElse: () => null,
          );
          if (user != null) {
            // Guardar la sesión del usuario
```

```
          SharedPreferences prefs = await
SharedPreferences.getInstance();
          await prefs.setInt('userId', user.id);
          await prefs.setString('userName', user.name);
          // Navegar a la pantalla principal
          Navigator.pushReplacement(
            context,
            MaterialPageRoute(
              builder: (context) => HomeScreen(
                onThemeChanged: widget.onThemeChanged,
              ),
            ),
          );
        } else {
          _showError('Credenciales incorrectas');
        }
      } catch (e) {
        _showError('Error al iniciar sesión. Inténtalo de
nuevo.');
      } finally {
        setState(() {
          _isLoading = false;
        });
      }
    }
  }
```

```
  void _showError(String message) {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(content: Text(message)),
    );
  }
}
```

Explicación

- **Formulario de Login**: Creamos un formulario con campos para email y nombre de usuario (password).
- **Validación**: Utilizamos `Form` y `TextFormField` con validadores para asegurar que los campos no estén vacíos.
- **Autenticación**: Comparamos las credenciales ingresadas con los usuarios obtenidos del endpoint `/users`.
- **Persistencia de Sesión**: Almacenamos el `userId` y `userName` en `SharedPreferences` para mantener la sesión.
- **Navegación**: Redirigimos al usuario a la pantalla principal (`HomeScreen`) después de iniciar sesión.

Paso 2: Modificar `main.dart` para Usar `LoginScreen`

En **lib/main.dart**, debemos verificar si el usuario ya ha iniciado sesión y mostrar la pantalla correspondiente.

```
// lib/main.dart
import 'package:flutter/material.dart';
import 'package:shared_preferences/shared_preferences.dart';
import 'screens/home_screen.dart';
import 'screens/login_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;
  Widget _defaultHome = CircularProgressIndicator();

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

  // 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];
    });
  }

  // Verificar si el usuario ha iniciado sesión
  void _checkLoginStatus() async {
    SharedPreferences prefs = await
SharedPreferences.getInstance();
    int? userId = prefs.getInt('userId');
    setState(() {
      if (userId != null) {
        _defaultHome = HomeScreen(
          onThemeChanged: _toggleThemeMode,
        );
      } else {
        _defaultHome = LoginScreen(
          onThemeChanged: _toggleThemeMode,
        );
      }
    });
  }

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Red Social Básica',
      theme: ThemeData(
```

```
        brightness: Brightness.light,
        primarySwatch: Colors.blue,
        accentColor: Colors.blueAccent,
      ),
      darkTheme: ThemeData(
        brightness: Brightness.dark,
        primarySwatch: Colors.blue,
        accentColor: Colors.blueAccent,
      ),
      themeMode: _themeMode, // Modo de tema seleccionado
      home: _defaultHome,
    );
  }

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

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

```
}
```

Explicación

- **Verificación de Sesión**: En `initState()`, llamamos a `_checkLoginStatus()` para determinar si el usuario ha iniciado sesión.
- **Pantalla Predeterminada**: Usamos `_defaultHome` para establecer la pantalla inicial según el estado de la sesión.
- **Actualización de onThemeChanged**: Pasamos `onThemeChanged` a `HomeScreen` y `LoginScreen` para mantener la funcionalidad de cambio de tema.

2. Persistencia de Sesión y Cierre de Sesión

Paso 1: Modificar `HomeScreen` para Soportar Cierre de Sesión

En **lib/screens/home_screen.dart**, añadimos la opción de cerrar sesión.

```
// lib/screens/home_screen.dart
// Importamos SharedPreferences
import 'package:shared_preferences/shared_preferences.dart';
// ... resto de los imports

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

  HomeScreen({required this.onThemeChanged});

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

```
class _HomeScreenState extends State<HomeScreen> {
  // ... código existente

  @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<String>(
            onSelected: (value) {
              if (value == 'theme') {
                // Mostrar opciones de tema
                _showThemeSelection();
              } else if (value == 'logout') {
                _logout();
```

```
            }
          },
          itemBuilder: (context) => [
            PopupMenuItem(
              value: 'theme',
              child: Text('Cambiar Tema'),
            ),
            PopupMenuItem(
              value: 'logout',
              child: Text('Cerrar Sesión'),
            ),
          ],
        ),
      ],
    ),
    // ... resto del código
  );
}

void _showThemeSelection() {
  showDialog(
    context: context,
    builder: (context) => AlertDialog(
      title: Text('Seleccionar Tema'),
      content: Column(
        mainAxisSize: MainAxisSize.min,
        children: [
```

```
RadioListTile<ThemeMode>(
  title: Text('Predeterminado'),
  value: ThemeMode.system,
  groupValue: ThemeMode.system,
  onChanged: (value) {
    widget.onThemeChanged(value!);
    Navigator.of(context).pop();
  },
),
RadioListTile<ThemeMode>(
  title: Text('Modo Claro'),
  value: ThemeMode.light,
  groupValue: ThemeMode.light,
  onChanged: (value) {
    widget.onThemeChanged(value!);
    Navigator.of(context).pop();
  },
),
RadioListTile<ThemeMode>(
  title: Text('Modo Oscuro'),
  value: ThemeMode.dark,
  groupValue: ThemeMode.dark,
  onChanged: (value) {
    widget.onThemeChanged(value!);
    Navigator.of(context).pop();
  },
),
```

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

  void _logout() async {
    SharedPreferences prefs = await
SharedPreferences.getInstance();
    await prefs.remove('userId');
    await prefs.remove('userName');
    Navigator.pushReplacement(
      context,
      MaterialPageRoute(
        builder: (context) => LoginScreen(
          onThemeChanged: widget.onThemeChanged,
        ),
      ),
    );
  }
}
```

Explicación

- **Menú de Opciones**: Reemplazamos el `PopupMenuButton<ThemeMode>` por `PopupMenuButton<String>` para incluir más opciones.
- **Cambio de Tema**: Al seleccionar 'Cambiar Tema', mostramos un `AlertDialog` con opciones.

- **Cierre de Sesión**: Al seleccionar 'Cerrar Sesión', eliminamos los datos de sesión de `SharedPreferences` y navegamos de regreso a `LoginScreen`.

3. Mostrar Contenido Personalizado

Paso 1: Obtener Información del Usuario Autenticado

En **HomeScreen**, obtenemos el nombre del usuario autenticado y lo mostramos.

```
class _HomeScreenState extends State<HomeScreen> {
  // ... código existente
  String _userName = '';

  @override
  void initState() {
    super.initState();
    futurePosts = apiService.getPosts();
    _loadUserInfo();
  }

  void _loadUserInfo() async {
    SharedPreferences prefs = await
SharedPreferences.getInstance();
    setState(() {
      _userName = prefs.getString('userName') ?? '';
    });
  }

  @override
  Widget build(BuildContext context) {
```

```
    return Scaffold(
      appBar: AppBar(
        title: Text('Bienvenido, $_userName'),
        // ... resto del código
      ),
      // ... resto del código
    );
  }
}
```

Explicación

- **Cargar Información del Usuario**: Usamos `_loadUserInfo()` para obtener el `userName` almacenado.
- **Mostrar Nombre del Usuario**: Mostramos el nombre en el `AppBar` para personalizar la experiencia.

Paso 2: Filtrar Publicaciones del Usuario Autenticado (Opcional)

Si deseamos mostrar solo las publicaciones del usuario autenticado:

```
void _loadPosts() async {
  int? userId = prefs.getInt('userId');
  List<Post> allPosts = await apiService.getPosts();
  setState(() {
    posts = allPosts.where((post) => post.userId ==
userId).toList();
  });
}
```

Sin embargo, para mantener la funcionalidad de ver todas las publicaciones, podemos dejarlo como está.

4. Mantener la Persistencia de las Preferencias de Tema

Aseguramos que las preferencias de tema sigan funcionando correctamente con el sistema de login. Ya hemos implementado el almacenamiento y carga de las preferencias en `main.dart`.

5. Mejoras en la Experiencia de Usuario

5.1. Manejo de Errores y Mensajes

- **Mensajes de Error**: Utilizamos `ScaffoldMessenger` para mostrar mensajes de error en el login.
- **Indicador de Carga**: Mostramos un `CircularProgressIndicator` mientras se realiza la autenticación.

5.2. Validación de Credenciales

- **Comparación sin Distinción de Mayúsculas**: Al comparar emails, usamos `toLowerCase()` para evitar problemas con mayúsculas y minúsculas.
- **Mensajes Claros**: Informamos al usuario si las credenciales son incorrectas.

6. Actualizar Navegación en la Aplicación

Aseguramos que todas las pantallas estén accesibles y la navegación sea fluida.

- **Pasar onThemeChanged**: A las pantallas que lo requieran, para mantener la funcionalidad de cambio de tema.
- **Volver al Login tras Cerrar Sesión**: Utilizamos `Navigator.pushReplacement` para reemplazar la ruta actual.

Ejemplos en Código

Ejemplo: Autenticación de Usuario

```
void _login() async {
  if (_formKey.currentState!.validate()) {
    _formKey.currentState!.save();
    setState(() {
      _isLoading = true;
    });
    try {
      List<User> users = await apiService.getUsers();
      User? user = users.firstWhere(
        (user) => user.email.toLowerCase() ==
_email.toLowerCase() && user.username == _password,
        orElse: () => null,
      );
      if (user != null) {
        // Guardar la sesión del usuario
        SharedPreferences prefs = await
SharedPreferences.getInstance();
        await prefs.setInt('userId', user.id);
        await prefs.setString('userName', user.name);
        // Navegar a la pantalla principal
        Navigator.pushReplacement(
          context,
          MaterialPageRoute(
            builder: (context) => HomeScreen(
```

```
              onThemeChanged: widget.onThemeChanged,
            ),
          ),
        );
      } else {
        _showError('Credenciales incorrectas');
      }
    } catch (e) {
      _showError('Error al iniciar sesión. Inténtalo de
nuevo.');
    } finally {
      setState(() {
        _isLoading = false;
      });
    }
  }
}
```

Explicación:

- **Validación**: Aseguramos que los campos no estén vacíos antes de proceder.
- **Peticiones Asíncronas**: Utilizamos `async` y `await` para manejar la llamada al API.
- **Comparación de Credenciales**: Buscamos un usuario que coincida con el email y username ingresados.
- **Persistencia**: Almacenamos información del usuario en `SharedPreferences` para mantener la sesión.

Relación con Otros Temas

Esta lección se relaciona con:

- **Persistencia de Datos**: Almacenamos información en `SharedPreferences`, un concepto clave en desarrollo móvil.
- **Seguridad y Autenticación**: Aunque nuestra implementación es básica y no segura para producción, introduce conceptos de autenticación de usuarios.
- **Manejo de Estados**: Gestionamos el estado de la aplicación al verificar si el usuario está autenticado y al manejar las preferencias de tema.
- **Interfaz de Usuario Dinámica**: Personalizamos la UI basándonos en los datos del usuario autenticado.

Resumen de la Lección

En esta lección, hemos implementado la funcionalidad de login en nuestra aplicación, permitiendo a los usuarios autenticarse utilizando su email y username. Hemos persistido la sesión del usuario para mantenerlo autenticado entre sesiones y actualizado la aplicación para mostrar contenido personalizado. Además, aseguramos que las preferencias de tema siguen funcionando correctamente y se mantienen almacenadas. Con estas mejoras, nuestra aplicación ofrece una experiencia más personalizada y cercana a una red social real.

Actividad de la Lección

Objetivo de la Actividad:

- Verificar que has implementado correctamente la funcionalidad de login y persistencia de sesión.
- Demostrar tu capacidad para manejar autenticación y personalización en una aplicación Flutter.
- Prepararte para las siguientes lecciones, donde añadiremos funcionalidades adicionales como manejo de comentarios y mejoras en la interacción social

Tarea Práctica:

1. **Implementa la pantalla de login** en tu proyecto, siguiendo los ejemplos proporcionados.
2. **Modifica `main.dart`** para verificar el estado de la sesión y mostrar la pantalla correspondiente.
3. **Añade la funcionalidad de cierre de sesión** en `HomeScreen`.
4. **Asegúrate de que las preferencias de tema** siguen funcionando y se mantienen almacenadas.
5. **Prueba la aplicación** iniciando sesión con diferentes usuarios y verificando que la sesión se mantiene al cerrar y reabrir la aplicación.
6. **Documenta tu proceso**: Crea un documento en PDF que incluya capturas de pantalla de las nuevas funcionalidades y explica cualquier problema que hayas encontrado y cómo lo solucionaste.
7. **Entrega**: Sube el PDF y el código fuente actualizado del proyecto en una carpeta comprimida.