Lección 5: Desarrollo de Funcionalidades de Publicaciones, Comentarios y Perfiles

Objetivos de la Lección

- **Implementar** funcionalidades que permitan a los usuarios ver publicaciones de otros usuarios.
- **Permitir** a los usuarios comentar en publicaciones.
- **Visualizar** perfiles de usuarios y sus publicaciones.
- **Mejorar** la interacción social en la aplicación.
- **Aplicar** conocimientos de manejo de estado y navegación en Flutter.

Introducción a la Lección

En esta lección, nos enfocaremos en añadir funcionalidades clave que acercarán nuestra aplicación a una red social real. Implementaremos la posibilidad de que los usuarios vean publicaciones de otros, comenten en ellas y visualicen perfiles de usuarios. Esto permitirá una mayor interacción y personalización dentro de la aplicación.

Utilizaremos los endpoints proporcionados por **JSONPlaceholder** para obtener y enviar datos de publicaciones y comentarios. Aprenderemos a manejar peticiones POST para añadir comentarios y a actualizar la interfaz de usuario en consecuencia.

Esta lección es fundamental para enriquecer la funcionalidad de nuestra aplicación y ofrecer una experiencia más completa a los usuarios.

Desarrollo de Conceptos

Interacción Social en Aplicaciones Móviles

Las redes sociales se basan en la interacción entre usuarios a través de contenido compartido. Permitir a los usuarios ver, comentar y reaccionar a publicaciones es esencial para fomentar esa interacción.

Peticiones HTTP POST en Flutter

Además de las peticiones GET para obtener datos, es importante aprender a realizar peticiones POST para enviar datos al servidor. Aunque en nuestro caso los cambios no se reflejarán en el servidor (porque es una API falsa), podremos simular la funcionalidad de añadir comentarios.

Actualización de la Interfaz de Usuario

Es importante actualizar la interfaz para reflejar los cambios realizados por el usuario, como añadir un comentario. Aprenderemos a manejar el estado y refrescar la UI en consecuencia.

Secciones Técnicas Específicas

1. Añadir Comentarios a las Publicaciones

Paso 1: Crear la Pantalla para Añadir Comentarios

Crea un nuevo archivo **lib/screens/add_comment_screen.dart**.

```
// lib/screens/add_comment_screen.dart
import 'package:flutter/material.dart';
import 'package:shared_preferences/shared_preferences.dart';
import '../services/api_service.dart';
import '../models/comment.dart';
```

```
class AddCommentScreen extends StatefulWidget {
  final int postId;

  AddCommentScreen({required this.postId});

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

class _AddCommentScreenState extends State<AddCommentScreen> {
  final _formKey = GlobalKey<FormState>();
  String _commentBody = '';
  bool _isLoading = false;
  ApiService apiService = ApiService();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Añadir Comentario'),
      ),
      body: Padding(
        padding: EdgeInsets.all(16),
        child: _isLoading
            ? Center(child: CircularProgressIndicator())
```

```
            : Form(
                key: _formKey,
                child: Column(
                  children: [
                    TextFormField(
                      decoration: InputDecoration(labelText:
'Comentario'),
                      maxLines: 5,
                      validator: (value) {
                        if (value == null || value.isEmpty) {
                          return 'Por favor ingresa un
comentario';
                        }
                        return null;
                      },
                      onSaved: (value) {
                        _commentBody = value!.trim();
                      },
                    ),
                    SizedBox(height: 20),
                    ElevatedButton(
                      onPressed: _addComment,
                      child: Text('Enviar'),
                    ),
                  ],
                ),
              ),
```

```
      ),
    );
  }

  void _addComment() async {
    if (_formKey.currentState!.validate()) {
      _formKey.currentState!.save();
      setState(() {
        _isLoading = true;
      });
      try {
        // Obtener información del usuario autenticado
        SharedPreferences prefs = await
SharedPreferences.getInstance();
        String? userName = prefs.getString('userName');
        String? userEmail = prefs.getString('userEmail'); //
Debemos almacenar esto previamente

        if (userName == null || userEmail == null) {
          _showError('Error al obtener información del
usuario');
          setState(() {
            _isLoading = false;
          });
          return;
        }
```

```
        Comment newComment = await apiService.addComment(
          widget.postId,
          userName,
          userEmail,
          _commentBody,
        );

        Navigator.pop(context, newComment);
      } catch (e) {
        _showError('Error al enviar el comentario. Inténtalo de
nuevo.');
        setState(() {
          _isLoading = false;
        });
      }
    }
  }

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

Explicación

- **Formulario para Añadir Comentario**: Creamos un formulario con un campo de texto para el comentario.

- **Peticiones POST**: Utilizamos el método `addComment` de `ApiService` para simular el envío del comentario.
- **Manejo de Respuestas**: Al obtener una respuesta exitosa, volvemos a la pantalla anterior y pasamos el nuevo comentario.

Paso 2: Modificar `ApiService` para Almacenar el Email del Usuario

En **lib/services/api_service.dart**, ya tenemos el método `addComment`, pero necesitamos asegurarnos de que estamos enviando el email del usuario.

Asegúrate de que el método `addComment` está implementado correctamente:

```
// lib/services/api_service.dart
Future<Comment> addComment(int postId, String name, String email, String body) async {
  final response = await http.post(
    Uri.parse('$baseUrl/comments'),
    headers: <String, String>{
      'Content-Type': 'application/json; charset=UTF-8',
    },
    body: jsonEncode(<String, dynamic>{
      'postId': postId,
      'name': name,
      'email': email,
      'body': body,
    }),
  );

  if (response.statusCode == 201) {
    return Comment.fromJson(json.decode(response.body));
```

```
  } else {
    throw Exception('Error al añadir comentario');
  }
}
```

Paso 3: Almacenar el Email del Usuario en SharedPreferences

En el método `_login` de **login_screen.dart**, debemos almacenar el email del usuario autenticado.

```
await prefs.setString('userEmail', user.email);
```

Paso 4: Modificar PostDetailScreen para Navegar a AddCommentScreen

En **lib/screens/post_detail_screen.dart**, actualiza el método `onPressed` del `FloatingActionButton`.

```
floatingActionButton: FloatingActionButton(
  onPressed: () async {
    final result = await Navigator.push(
      context,
      MaterialPageRoute(
        builder: (context) => AddCommentScreen(postId:
widget.post.id),
      ),
    );
    if (result != null) {
      // Actualizamos la lista de comentarios
      setState(() {
        futureComments = apiService.getComments(widget.post.id);
```

```
      });
    }
  },
  child: Icon(Icons.add_comment),
),
```

Explicación

- **Navegación**: Al presionar el botón, navegamos a `AddCommentScreen`.
- **Actualización de Comentarios**: Si se añade un nuevo comentario, actualizamos la lista de comentarios al volver.

2. Actualizar la Interfaz de Comentarios

En **PostDetailScreen**, podemos mejorar la visualización de los comentarios para que muestren el nombre y email del autor.

En el método `_buildCommentsSection()`:

```
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: CircleAvatar(
                child: Text(comments[index].name.substring(0,
1)),
              ),
              title: Text(comments[index].name),
              subtitle: Text(comments[index].body),
              trailing: Text(comments[index].email),
            );
          },
        );
      } else if (snapshot.hasError) {
        return Center(child: Text('Error: ${snapshot.error}'));
      }
      return Center(child: CircularProgressIndicator());
    },
  );
}
```

3. Visualizar Perfiles de Usuarios y Sus Publicaciones

Paso 1: Modificar `UserProfileScreen` para Mostrar Publicaciones del Usuario

En **lib/screens/user_profile_screen.dart**, añade una sección que muestra las publicaciones del usuario.

```
// Añade los imports necesarios
import '../models/post.dart';
import '../services/api_service.dart';
```

```
// Dentro de UserProfileScreen
class UserProfileScreen extends StatelessWidget {
  final User user;
  final ApiService apiService = ApiService();

  UserProfileScreen({required this.user});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(user.name),
      ),
      body: SingleChildScrollView(
        child: Column(
          children: [
            // Información del usuario
            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),
),
Divider(),
// Publicaciones del usuario
Padding(
  padding: EdgeInsets.all(16),
  child: Text(
    'Publicaciones',
    style: TextStyle(fontSize: 18),
  ),
),
FutureBuilder<List<Post>>(
  future: apiService.getPostsByUser(user.id),
  builder: (context, snapshot) {
    if (snapshot.hasData) {
      List<Post> posts = snapshot.data!;
      return ListView.builder(
        shrinkWrap: true,
        physics: NeverScrollableScrollPhysics(),
        itemCount: posts.length,
        itemBuilder: (context, index) {
          return ListTile(
```

```
                        title: Text(posts[index].title),
                        subtitle: Text(posts[index].body),
                        onTap: () {
                          Navigator.push(
                            context,
                            MaterialPageRoute(
                              builder: (context) =>
PostDetailScreen(post: posts[index]),
                            ),
                          );
                        },
                      );
                    },
                  );
                } else if (snapshot.hasError) {
                  return Center(child: Text('Error:
${snapshot.error}'));
                }
                return Center(child:
CircularProgressIndicator());
              },
            ),
          ],
        ),
      ),
    );
  }
```

```
}
```

Paso 2: Añadir Método `getPostsByUser` en `ApiService`

En **lib/services/api_service.dart**, añade el método para obtener las publicaciones de un usuario específico.

```
// En ApiService
Future<List<Post>> getPostsByUser(int userId) async {
  final response =
      await
http.get(Uri.parse('$baseUrl/posts?userId=$userId'));
  if (response.statusCode == 200) {
    List jsonResponse = json.decode(response.body);
    return jsonResponse.map((post) =>
Post.fromJson(post)).toList();
  } else {
    throw Exception('Error al cargar publicaciones del
usuario');
  }
}
```

Explicación

- **Filtrado por `userId`**: Utilizamos el parámetro de consulta `userId` para obtener solo las publicaciones del usuario.

Paso 3: Modificar Navegación en `UsersScreen`

En **lib/screens/users_screen.dart**, asegura que al seleccionar un usuario, navegues a su perfil.

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

4. Mejorar la Interacción Social

Paso 1: Simular Funcionalidades de "Me Gusta" y "Compartir"

En **HomeScreen**, modifica el método `_buildPostCard`.

```
// Agrega una variable para manejar el estado de "Me Gusta"
Set<int> likedPosts = Set<int>();

Widget _buildPostCard(Post post) {
  bool isLiked = likedPosts.contains(post.id);

  return Card(
    margin: EdgeInsets.symmetric(vertical: 8, horizontal: 16),
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
```

```
children: [
  // ... código existente
  ButtonBar(
    alignment: MainAxisAlignment.start,
    children: [
      TextButton.icon(
        onPressed: () {
          setState(() {
            if (isLiked) {
              likedPosts.remove(post.id);
            } else {
              likedPosts.add(post.id);
            }
          });
        },
        icon: Icon(
          Icons.thumb_up,
          color: isLiked ? Colors.blue : Colors.grey,
        ),
        label: Text('Me Gusta'),
      ),
      TextButton.icon(
        onPressed: () {
          // Simular compartir
          ScaffoldMessenger.of(context).showSnackBar(
            SnackBar(content: Text('Compartido!')),
          );
```

```
              },
              icon: Icon(Icons.share),
              label: Text('Compartir'),
            ),
          ],
        ),
      ],
    ),
  );
}
```

Explicación

- **Manejo de Estado**: Utilizamos un `Set` para mantener el estado de las publicaciones a las que el usuario ha dado "Me Gusta".
- **Actualizar Icono**: Cambiamos el color del icono de "Me Gusta" según el estado.
- **Funcionalidad de Compartir**: Mostramos un mensaje al usuario al presionar "Compartir".

5. Mejoras en la Experiencia de Usuario

Paso 1: Mostrar Contador de "Me Gusta"

Podemos añadir un contador de "Me Gusta" para cada publicación. Como no tenemos datos reales, podemos simularlos.

En **Post** (lib/models/post.dart), añade un campo opcional:

```
// Agrega una variable para el contador de "Me Gusta"
int likes;


Post({
```

```
  required this.userId,
  required this.id,
  required this.title,
  required this.body,
  this.likes = 0,
});
```

En el método `fromJson`:

```
likes: 0, // Inicializamos en 0 o un valor aleatorio
```

En **HomeScreen**, actualiza el método `_buildPostCard` para mostrar el contador.

```
ButtonBar(
  alignment: MainAxisAlignment.start,
  children: [
    TextButton.icon(
      onPressed: () {
        setState(() {
          if (isLiked) {
            likedPosts.remove(post.id);
            post.likes--;
          } else {
            likedPosts.add(post.id);
            post.likes++;
          }
        });
      },
```

```
      icon: Icon(
        Icons.thumb_up,
        color: isLiked ? Colors.blue : Colors.grey,
      ),
      label: Text('Me Gusta (${post.likes})'),
    ),
    // ... código restante
  ],
),
```

Explicación

- **Actualizar Contador**: Incrementamos o decrementamos el contador al dar o quitar "Me Gusta".
- **Mostrar Contador**: Incluimos el número de "Me Gusta" en el label del botón.

6. Manejo de Estado y Actualización de la Interfaz

Es importante manejar adecuadamente el estado para reflejar los cambios en la interfaz.

- **`setState`**: Utilizamos `setState` para notificar a Flutter que el estado ha cambiado y debe redibujar la interfaz.
- **Optimización**: Evitamos llamadas innecesarias a `setState` para mejorar el rendimiento.

Ejemplos en Código

Ejemplo: Añadir un Comentario a una Publicación

En **AddCommentScreen**:

```
void _addComment() async {
  if (_formKey.currentState!.validate()) {
    _formKey.currentState!.save();
    setState(() {
      _isLoading = true;
    });
    try {
      SharedPreferences prefs = await
SharedPreferences.getInstance();
      String? userName = prefs.getString('userName');
      String? userEmail = prefs.getString('userEmail');

      if (userName == null || userEmail == null) {
        _showError('Error al obtener información del usuario');
        setState(() {
          _isLoading = false;
        });
        return;
      }

      Comment newComment = await apiService.addComment(
        widget.postId,
```

```
        userName,
        userEmail,
        _commentBody,
      );

      Navigator.pop(context, newComment);
    } catch (e) {
      _showError('Error al enviar el comentario. Inténtalo de
nuevo.');
      setState(() {
        _isLoading = false;
      });
    }
  }
}
```

Explicación:

- **Validación**: Verificamos que los campos estén llenos antes de enviar.
- **Enviar Comentario**: Utilizamos el método `addComment` del `ApiService`.
- **Actualizar Interfaz**: Al regresar, actualizamos la lista de comentarios en la pantalla anterior.

Relación con Otros Temas

Esta lección se relaciona con:

- **Manejo de Peticiones HTTP**: Al realizar peticiones POST para añadir comentarios.

- **Gestión de Estado**: Al manejar interacciones como "Me Gusta" y actualizar la interfaz.
- **Navegación y Rutas**: Al navegar entre pantallas y pasar datos entre ellas.
- **Persistencia de Datos**: Al almacenar información del usuario para usar en comentarios y perfiles.

Resumen de la Lección

En esta lección, hemos desarrollado funcionalidades clave que enriquecen nuestra aplicación de red social básica. Hemos permitido a los usuarios añadir comentarios a las publicaciones, visualizar perfiles de otros usuarios y sus publicaciones, y hemos simulado funcionalidades de "Me Gusta" y "Compartir". Estas mejoras fomentan la interacción social dentro de la aplicación y ofrecen una experiencia más completa y cercana a una red social real.

Actividad de la Lección

Objetivo de la Actividad:

- Verificar que has implementado correctamente las funcionalidades de publicaciones, comentarios y perfiles.
- Demostrar tu capacidad para manejar interacciones y actualizar la interfaz de usuario en Flutter.
- Prepararte para la siguiente lección, donde añadiremos más funcionalidades y refinamientos a la aplicación.

Tarea Práctica:

1. **Implementa la funcionalidad para añadir comentarios** a las publicaciones en tu proyecto.
2. **Actualiza el `PostDetailScreen`** para mostrar los comentarios y permitir añadir nuevos.

3. **Modifica el `UserProfileScreen`** para mostrar las publicaciones del usuario.
4. **Añade la funcionalidad de "Me Gusta" y "Compartir"** en las publicaciones, simulándolas localmente.
5. **Prueba la aplicación** para verificar que todas las nuevas funcionalidades funcionan correctamente.
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.