Práctica 4: Formularios y Navegación

Asignatura: Desarrollo de Aplicaciones Móviles
Fecha: Noviembre 2025
Autor: Miguel Martínez Rosario

Índice

Introducción

Esta práctica implementa un sistema completo de navegación entre pantallas en Flutter, comenzando con un formulario de autenticación que valida las credenciales del usuario y permite navegar por las provincias y comarcas de la Comunidad Valenciana.

Objetivos cumplidos:

- ✅ Creación de un formulario de login con validación
- ✅ Implementación de diálogos personalizados
- ✅ Navegación jerárquica: Login → Provincias → Comarcas → Detalle
- ✅ Uso de GestureDetector para interactividad
- ✅ Paso de parámetros entre pantallas
- ✅ Integración con el repositorio de datos

Estructura del Proyecto

```
lib/
├── main.dart                          # Punto de entrada (usa LoginScreen)
├── models/
│   ├── comarca.dart                   # Modelo de datos Comarca
│   └── provincia.dart                 # Modelo de datos Provincia
├── repository/
│   └── repository_ejemplo.dart        # Fuente de datos estática
├── screens/
│   ├── login_screen.dart              # 🆕 NUEVA: Pantalla de autenticación
│   ├── provincias_screen.dart         # 🔄 MODIFICADA: Navegación a comarcas
│   ├── comarcas_screen.dart           # 🔄 MODIFICADA: Filtrado y navegación
│   ├── infocomarca_general.dart       # Información general de comarca
│   ├── infocomarca_detall.dart        # Información detallada de comarca
│   └── widgets/
│       └── my_weather_info.dart       # Widget de información meteorológica
└── themes/
    └── tema_comarcas.dart             # Tema personalizado
```

Implementación de Pantallas

1. Pantalla de Login (`login_screen.dart`)

Funcionalidad:

- Formulario con dos campos: usuario y contraseña
- Validación de credenciales (usuario: `admin`, contraseña: `flutter`)
- Diálogo de error con dos opciones:
  - **Volver**: Cierra el diálogo manteniendo los datos ingresados
  - **Rellenar usuario**: Autocompleta con las credenciales correctas

Componentes clave:

```
class LoginScreen extends StatefulWidget {
  // Uso de StatefulWidget para gestionar el estado del formulario

  final _formKey = GlobalKey<FormState>();
  final _userController = TextEditingController();
  final _passController = TextEditingController();

  // Credenciales válidas
  final String _validUser = 'admin';
  final String _validPass = 'flutter';
}
```

Validación implementada:

- Los campos no pueden estar vacíos
- Comparación exacta de credenciales
- Navegación con `pushReplacement` para evitar volver al login con el botón atrás

Diálogo personalizado:

```
AlertDialog(
  title: const Text('Error de autenticación'),
  content: const Text('Usuario o contraseña incorrectos.'),
  actions: [
    TextButton(onPressed: () => Navigator.pop(), child: const Text('Volver')),
    TextButton(
      onPressed: () {
        _userController.text = _validUser;
        _passController.text = _validPass;
        Navigator.pop();
      },
      child: const Text('Rellenar usuario'),
    ),
  ],
)
```

---

2. Pantalla de Provincias (`provincias_screen.dart`)

Modificaciones realizadas:

- Envolvimos cada `ProvinciaRoundButton` dentro de un `GestureDetector`
- Al pulsar una provincia, navegamos a `ComarcasScreen` pasando el nombre de la provincia

Código de navegación:

```
GestureDetector(
  onTap: () {
    Navigator.push(
      context,
      MaterialPageRoute(
        builder: (context) => ComarcasScreen(
          provinceName: provincia.nombre,
        ),
      ),
    );
  },
  child: ProvinciaRoundButton(provincia: provincia),
)
```

Características:

- Mantiene el diseño circular de las provincias
- Feedback visual al pulsar (proporcionado por Material)
- Transición fluida entre pantallas

---

3. Pantalla de Comarcas (`comarcas_screen.dart`)

Modificaciones realizadas:

1. **Parámetro opcional `provinceName`:**

```
class ComarcasScreen extends StatelessWidget {
  final String? provinceName;

  const ComarcasScreen({
    super.key,
    this.provinceName,
  });
}
```

2. **AppBar dinámica:**

```
AppBar(
  title: Text(provinceName != null
    ? 'Comarcas - $provinceName'
    : 'Comarcas'),
)
```

3. **Navegación al detalle:** Cada `ComarcaCard` está envuelta en un `GestureDetector` que navega a `InfoComarcaDetall` pasando el ID de la comarca:

```
GestureDetector(
  onTap: () {
    if (id.isNotEmpty) {
      Navigator.push(
        context,
        MaterialPageRoute(
          builder: (context) => InfoComarcaDetall(comarcaId: id),
        ),
      );
    }
  },
  child: ComarcaCard(comarca: nombre, img: img),
)
```

4. **Mejora en carga de imágenes:** Ahora soporta tanto imágenes de assets como URLs:

```
Widget _buildImage(String path) {
  if (path.startsWith('http')) {
    return Image.network(path, fit: BoxFit.cover, errorBuilder: ...);
  }
  if (path.startsWith('assets/')) {
    return Image.asset(path, fit: BoxFit.cover, errorBuilder: ...);
  }
  return Image.network(path, fit: BoxFit.cover, errorBuilder: ...);
}
```

---

4. Pantallas de Información de Comarca

`infocomarca_detall.dart`:

- Recibe el `comarcaId` como parámetro
- Usa `RepositoryEjemplo.obtenerInfoComarca(comarcaId)` para cargar los datos
- Muestra información detallada: población, coordenadas, información meteorológica

`infocomarca_general.dart`:

- Recibe un objeto `Comarca` completo
- Muestra imagen, capital, población y descripción
- Incluye botón para navegar a la vista detallada

Navegación entre Pantallas

Flujo de Navegación Implementado

Tipos de Navegación Utilizados

1. **pushReplacement (Login → Provincias):**
    - Evita que el usuario pueda volver al login con el botón atrás
    - Apropiado para flujos de autenticación
2. **push (resto de navegaciones):**
    - Permite navegación jerárquica
    - El botón atrás funciona correctamente
    - Se mantiene el stack de navegación

Paso de Parámetros

Origen	Destino	Parámetro	Tipo
Login	Provincias	-	-
Provincias	Comarcas	`provinceName`	`String?`
Comarcas	Detalle	`comarcaId`	`String`

Dificultades Encontradas

1. Estructura del Repositorio

Problema: El método `obtenerComarcas()` devuelve todas las comarcas sin filtrar por provincia.

Solución aplicada:

- De momento se muestran todas las comarcas independientemente de la provincia seleccionada
- Se pasa el nombre de la provincia solo para mostrarlo en la AppBar
- En prácticas futuras se implementará el filtrado cuando se conecte con la API

Código actual:

```
// TODO: En futuras prácticas filtrar por provincia
static Future<List<dynamic>> obtenerComarcas() async {
  await Future.delayed(const Duration(milliseconds: 500));
  return _comarcasData; // Devuelve todas
}
```

---

2. Identificadores en el Repositorio

Problema: El repositorio usa `id` para identificar comarcas, pero algunas pantallas esperaban el nombre.

Solución:

- Estandarización: siempre usamos el campo `id` para navegación
- En `InfoComarcaDetall` el parámetro `comarcaId` recibe el `id` (no el nombre)
- El método `obtenerInfoComarca(String id)` busca por `id` usando `firstWhere`

Ejemplo del repositorio:

```
static Comarca obtenerInfoComarca(String id) {
  final comarcaData = _comarcasData.firstWhere(
    (comarca) => comarca['id'] == id,
    orElse: () => _comarcasData.first,
  );
  return Comarca.fromJSON(comarcaData);
}
```

---

3. Gestión de Imágenes (Assets vs URLs)

Problema: Las comarcas usan rutas de assets (`assets/img/...`) pero el widget `ComarcaCard` solo soportaba URLs.

Solución: Implementamos un método `_buildImage()` que detecta el tipo de ruta:

```
Widget _buildImage(String path) {
  if (path.startsWith('http')) {
    return Image.network(path, ...);
  }
  if (path.startsWith('assets/')) {
    return Image.asset(path, ...);
  }
  return Image.network(path, ...); // Fallback
}
```

Nota: Se añadió `errorBuilder` para mostrar un icono de fallback si la imagen no carga.

---

4. Validación del Formulario

Problema inicial: El formulario no validaba correctamente los espacios en blanco.

Solución:

```
validator: (value) {
  if (value == null || value.trim().isEmpty) {
    return 'Introduce el usuario';
  }
  return null;
}
```

Se usa `.trim()` para eliminar espacios antes y después.

---

5. Gestión del Estado en el Diálogo

Problema: Al rellenar automáticamente el usuario desde el diálogo, los campos no se actualizaban visualmente.

Solución:

```
TextButton(
  onPressed: () {
    _userController.text = _validUser;
    _passController.text = _validPass;
    Navigator.of(context).pop();
    setState(() {}); // Forzar reconstrucción
  },
  child: const Text('Rellenar usuario'),
)
```

Llamar a `setState()` después de cerrar el diálogo fuerza la reconstrucción del widget.

---

6. Duplicación de Clase en comarcas_screen.dart

Problema técnico: Durante la edición del archivo se duplicó la declaración de la clase `ComarcasScreen`.

Solución:

- Detección mediante análisis de errores de compilación
- Eliminación de la declaración duplicada
- Verificación con `get_errors` para confirmar que no quedaban errores

Aprendizaje: Importancia de revisar errores de compilación después de cada modificación.

Conclusiones

Objetivos Alcanzados

✅ **Formulario funcional** con validación y manejo de errores
✅ **Navegación completa** entre todas las pantallas
✅ **Paso de parámetros** correctamente implementado
✅ **Experiencia de usuario** fluida con feedback visual
✅ **Gestión de imágenes** flexible (assets y URLs)
✅ **Código limpio** y bien estructurado

Habilidades Desarrolladas

1. **Navegación en Flutter:**
   - Uso de `Navigator.push()` y `Navigator.pushReplacement()`
   - Paso de parámetros entre pantallas
   - Gestión del stack de navegación
2. **Formularios y Validación:**
   - Uso de `Form` y `GlobalKey<FormState>`
   - Controladores de texto (`TextEditingController`)
   - Validación de campos
3. **Gestión de Estado:**
   - Diferencia entre `StatelessWidget` y `StatefulWidget`
   - Uso de `setState()` para actualizar la UI
   - Gestión del ciclo de vida con `dispose()`
4. **Interactividad:**
   - Uso de `GestureDetector` para detectar toques
   - Implementación de diálogos con `showDialog`
   - Feedback visual con Material Design
5. **Buenas Prácticas:**
   - Separación de responsabilidades
   - Código reutilizable (widgets personalizados)
   - Manejo de errores (imágenes, validación)
   - Limpieza de recursos (controladores)

Próximos Pasos (Prácticas Futuras)

- Conectar con la API REST para obtener datos reales
- Implementar filtrado de comarcas por provincia
- Añadir persistencia de sesión (login)
- Mejorar la información meteorológica con datos reales
- Implementar búsqueda de comarcas
- Añadir animaciones en las transiciones

Anexo: Comandos de Ejecución

Ejecutar la aplicación

```
flutter pub get
flutter run
```

Limpiar para entrega

```
flutter clean
```

Credenciales de prueba

- **Usuario:** `admin`
- **Contraseña:** `flutter`

Fin del documento