Módulo 5: Almacenamiento de Datos Locales en Aplicaciones Móviles

Lección 3: Uso de Hive para el Almacenamiento y Recuperación de Datos

Objetivos de la Lección

- Comprender cómo funciona Hive para el almacenamiento de datos en aplicaciones móviles.
- Implementar operaciones CRUD (Crear, Leer, Actualizar y Eliminar) en una aplicación móvil utilizando Hive.
- Aplicar Hive en un caso práctico para almacenar perfiles de cuentas de usuario, contraseñas y otros detalles importantes de servicios.

Introducción a la Lección

Hive es una base de datos NoSQL ligera y rápida para Flutter. Utiliza un modelo basado en cajas (similar a tablas) y es una excelente opción para almacenar datos sin conexión, ya que proporciona un almacenamiento persistente de alto rendimiento. Hive es adecuado para datos estructurados y semiestructurados, y no requiere un esquema fijo.

En esta lección, vamos a desarrollar una aplicación móvil simple que almacene perfiles de cuentas de usuario, incluyendo detalles como el nombre o título del servicio, tipo de servicio (página web, aplicación móvil, correo electrónico, etc.), nombre de usuario, contraseña y la fecha de creación o actualización.

Configuración de Hive en Flutter

Para usar Hive en tu proyecto de Flutter, sigue estos pasos:

1. **Agrega las dependencias de Hive** en tu archivo `pubspec.yaml`:

```
dependencies:
  flutter:
    sdk: flutter
  hive: ^2.2.3
  hive_flutter: ^1.1.0
  path_provider: ^2.1.5

dev_dependencies:
  hive_generator: ^2.0.1
  build_runner: ^2.4.13
```

2. **Almacena el archivo `pubspec.yaml`** o ejecuta el siguiente comando en la terminal para obtener las dependencias:

```
flutter pub get
```

3. **Revisa las conrfiguraciones de Android**; Aseguráte de que tienes los permisos necesarios en tu proyecto Android:

En `android/app/src/main/AndroidManifest.xml`, asegúrate de que tienes las siguientes líneas dentro de `<application>`:

```
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

```
<uses-permission
android:name="android.permission.READ_EXTERNAL_STORAGE"/>
```

4. **Importa Hive y Hive Flutter** en tu archivo Dart principal (`main.dart`):

```
import 'package:hive/hive.dart';
import 'package:hive_flutter/hive_flutter.dart';
```

4. **Inicializa Hive** al inicio de tu aplicación en el método `main()`:

```
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Hive.initFlutter();
  Hive.registerAdapter(AccountAdapter());
  await Hive.openBox<Account>('accounts');

  runApp(MyApp());
}
```

Modelado de Datos con Hive

¿Qué es un Modelo o el Modelado de Datos?

El **modelado de datos** es el proceso de definir y organizar la estructura de los datos que una aplicación o sistema necesita manejar. Un **modelo de datos** es una representación visual o conceptual de cómo se almacenan, organizan y relacionan los datos en un sistema, lo que ayuda a entender cómo interactúan entre sí. Este proceso es fundamental en el diseño de bases de datos y aplicaciones, ya que asegura que los datos se gestionen de manera eficiente y que las relaciones entre los datos se definan claramente.

Por ejemplo, en una aplicación de gestión de cuentas, el modelo de datos podría incluir entidades como "Usuario", "Servicio" y "Contraseña", y definir cómo se relacionan estas entidades para permitir un acceso y almacenamiento de datos efectivos.

Creando el Modelo de Datos con Hive

Antes de crear el CRUD, definiremos la estructura de datos para almacenar perfiles de cuentas. Utilizaremos Hive para generar un adaptador de tipo que nos permita guardar objetos personalizados.

1. **Crea una clase llamada `account.dart` en la carpeta `lib` para el modelo de datos**:

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

part 'account.g.dart';

@HiveType(typeId: 0)
class Account {
  @HiveField(0)
  String serviceName;

  @HiveField(1)
  String serviceType;

  @HiveField(2)
  String userName;

  @HiveField(3)
  String password;

  @HiveField(4)
```

```
  DateTime createdAt;

  Account({
    required this.serviceName,
    required this.serviceType,
    required this.userName,
    required this.password,
    required this.createdAt,
  });
}
```

2. **Genera el adaptador de tipo** ejecutando el siguiente comando en la terminal:

```
flutter pub run build_runner build
```

Esto generará un archivo llamado **`account.g.dart`** con el código necesario para que Hive maneje la serialización de los objetos `Account`.

Implementando el CRUD en Flutter con Hive

1. **Abrir y Configurar una Caja o "Box" en Hive**
    - Una caja o box en Hive es como una tabla en una base de datos relacional. Utilizaremos una caja llamada `accounts` para almacenar los datos.

```
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Hive.initFlutter();
  Hive.registerAdapter(AccountAdapter());
  await Hive.openBox<Account>('accounts');
  runApp(MyApp());
```

```
}
```

2. **Crear la Interfaz de Usuario**
    - La interfaz de la aplicación incluirá un formulario para agregar y actualizar cuentas, y una lista para mostrar las cuentas guardadas.

Código de Ejemplo Completo

```
import 'package:flutter/material.dart';
import 'package:hive/hive.dart';
import 'package:hive_flutter/hive_flutter.dart';
import 'account.dart';

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Hive.initFlutter();
  Hive.registerAdapter(AccountAdapter());
  await Hive.openBox<Account>('accounts');
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      theme: ThemeData.light(),
      home: AccountManager(),
    );
  }
}
```

```
class AccountManager extends StatefulWidget {
  @override
  _AccountManagerState createState() => _AccountManagerState();
}

class _AccountManagerState extends State<AccountManager> {
  final _serviceNameController = TextEditingController();
  final _userNameController = TextEditingController();
  final _passwordController = TextEditingController();
  String _selectedServiceType = 'WebPage';
  Box<Account> accountBox = Hive.box<Account>('accounts');
  bool _obscurePassword = true;
  int? _editingIndex;

  void _addOrUpdateAccount() {
    if (_editingIndex == null) {
      // Agregar nueva cuenta
      final account = Account(
        serviceName: _serviceNameController.text,
        serviceType: _selectedServiceType,
        userName: _userNameController.text,
        password: _passwordController.text,
        createdAt: DateTime.now(),
      );
      accountBox.add(account);
    } else {
      // Actualizar cuenta existente
      final account = accountBox.getAt(_editingIndex!)!;
      account
        ..serviceName = _serviceNameController.text
```

```
      ..serviceType = _selectedServiceType
      ..userName = _userNameController.text
      ..password = _passwordController.text
      ..createdAt = DateTime.now();
    accountBox.putAt(_editingIndex!, account);
  }
  _clearFields();
}

void _deleteAccount(int index) {
  accountBox.deleteAt(index);
  _clearFields();
}

void _clearFields() {
  _serviceNameController.clear();
  _userNameController.clear();
  _passwordController.clear();
  setState(() {
    _selectedServiceType = 'WebPage';
    _editingIndex = null;
  });
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text("Gestor de Cuentas"),
    ),
```

```
body: Column(
  children: [
    Padding(
      padding: const EdgeInsets.all(18.0),
      child: Column(
        children: [
          // Campos de entrada y DropdownButton
          Row(
            children: [
              const Text(
                'Servicio',
                style: TextStyle(fontSize: 16),
              ),
              const SizedBox(
                width: 10,
              ),
              Expanded(
                child: DropdownButton<String>(
                  value: _selectedServiceType,
                  items: [
                    'WebPage',
                    'Mobile App',
                    'Bank Account',
                    'eMail',
                    'PC Login',
                    'Others'
                  ].map((String value) {
                    return DropdownMenuItem<String>(
                      value: value,
                      child: Text(value),
```

```
                        );
                    }).toList(),
                    onChanged: (newValue) {
                      setState(() {
                        _selectedServiceType = newValue!;
                      });
                    },
                  ),
                ),
              ],
            ),
            TextField(
              controller: _serviceNameController,
              decoration: InputDecoration(labelText: "Nombre
del Servicio"),
            ),
            TextField(
              controller: _userNameController,
              decoration: InputDecoration(labelText: "Nombre
de Usuario"),
            ),
            Row(
              children: [
                Expanded(
                  child: TextField(
                    controller: _passwordController,
                    decoration: InputDecoration(labelText:
"Contraseña"),
                    obscureText: _obscurePassword,
                  ),
```

```
                    ),
                    IconButton(
                      icon: Icon(_obscurePassword
                          ? Icons.visibility
                          : Icons.visibility_off),
                      onPressed: () {
                        setState(() {
                          _obscurePassword = !_obscurePassword;
                        });
                      },
                    ),
                  ],
                ),
                // Campo de contraseña y botón de visibilidad
                ElevatedButton(
                  onPressed: _addOrUpdateAccount,
                  child: Text(_editingIndex == null
                      ? "Agregar Cuenta"
                      : "Actualizar Cuenta"),
                ),
              ],
            ),
          ),
          Expanded(
            child: ValueListenableBuilder(
              valueListenable: accountBox.listenable(),
              builder: (context, Box<Account> box, _) {
                if (box.values.isEmpty) {
                  return Center(child: Text("No hay cuentas
almacenadas."));
```

```
                }
                return ListView.builder(
                  itemCount: box.length,
                  itemBuilder: (context, index) {
                    final account = box.getAt(index)!;
                    return ListTile(
                      title: Text(account.serviceName),
                      subtitle: Text(account.serviceType),
                      trailing: Row(
                        mainAxisSize: MainAxisSize.min,
                        children: [
                          IconButton(
                            icon: Icon(Icons.edit),
                            onPressed: () {
                              setState(() {
                                _serviceNameController.text =
                                    account.serviceName;
                                _selectedServiceType =
account.serviceType;
                                _userNameController.text =
account.userName;
                                _passwordController.text =
account.password;
                                _editingIndex = index;
                              });
                            },
                          ),
                          IconButton(
                            icon: Icon(Icons.delete),
```

```
                            onPressed: () =>
_deleteAccount(index),
                          ),
                        ],
                      ),
                    );
                  },
                );
              },
            ),
          ),
        ],
      ),
    );
  }
}
```

Explicación del Código

1. **Inicialización y Configuración de Hive**:
    - `Hive.initFlutter()` inicializa Hive para Flutter.
    - `Hive.registerAdapter(AccountAdapter())` registra el adaptador del modelo de datos.
    - `Hive.openBox<Account>('accounts')` abre una caja para almacenar objetos `Account`.
2. **Modelo de Datos**:
    - La clase `Account` define el esquema de datos para almacenar perfiles de cuentas.
3. **Operaciones CRUD**:
    - **Crear**: `_addAccount()` agrega un nuevo perfil de cuenta a la caja.

- **Leer**: `ValueListenableBuilder` escucha los cambios en la caja y actualiza la interfaz de usuario.
- **Actualizar**: `_updateAccount()` actualiza un perfil de cuenta existente.
- **Eliminar**: `_deleteAccount()` elimina un perfil de cuenta de la caja.

4. **Interfaz de Usuario**:
   - Un formulario con campos de texto para ingresar detalles de la cuenta.
   - Una lista de las cuentas almacenadas, con opciones para editar o eliminar.

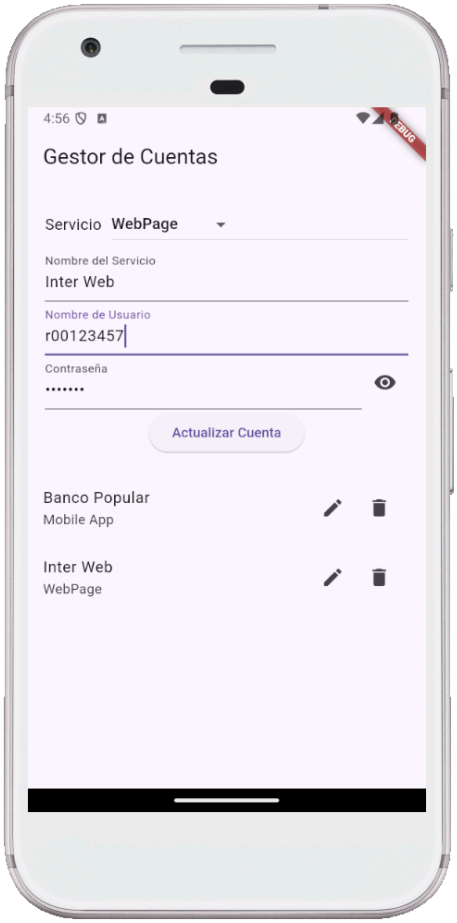

Imagen de aplicación móvil que almacena datos de cuentas de usuario utilizando Hive en Flutter.
Creada por Javier A. Dastas (2024)

Resumen de la Lección

En esta lección, aprendimos a utilizar Hive para almacenar y gestionar datos de manera eficiente en una aplicación Flutter. Implementamos un CRUD completo para un gestor de cuentas de usuario, lo que nos permite agregar, leer, actualizar y eliminar perfiles de cuentas de manera persistente. Hive es ideal para aplicaciones que requieren almacenamiento rápido y sin conexión.

Actividad de la Lección

Esta actividad te permitirá aplicar conocimientos de manipulación de listas y mejorar la funcionalidad de una aplicación de almacenamiento de datos en Flutter usando Hive.

Instrucciones:

1. **Historial de Modificaciones**: Modifica el código para almacenar un dato de historial de la última fecha de actualización de la cuenta. Así, cada vez que se actualice una cuenta, debe guardarse la fecha de actualización en un campo adicional llamado `updateHistory`, y mostrarse en la interfaz de usuario. Cada cuenta solo debe guardar la fecha de actualización más reciente.
2. Utiliza el siguiente código como ayuda para modificar tu código:

```
DateTime fechaActual = DateTime.now();
// Obtener la fecha en formato año-mes-día
String fecha =
'${fechaActual.year}-${fechaActual.month}-${fechaActual.day}';
```

3. Luego de realizar todos los cambios de la actividad desarrolla un documento con evidencia de la realización del código en esta lección y los requerimientos de esta actividad. Debes incluir imágenes de las pantallas de la aplicación, imagen de la estructura de archivos y carpetas de tu proyecto, y copia de todo el código.
4. Entrega el documento desarrollado en formato PDF en el enlace provisto para esta actividad.