INSTITUTO TECNOLÓGICO DE CIUDAD VALLES

CARRERA:

INGENIERÍA EN SISTEMAS COMPUTACIONALES

ACTIVIDAD:

Investigación Modelo, Proveedor, Servicio

NOMBRE DEL DOCENTE:

Ismael Gómez Rodríguez

ALUMNO:

Jorge Emmanuel Pérez Martínez 21690270

SEMESTRE:	FECHA:	MATERIA:
8°	*30/03/2025*	*DAMM*

Modelo de datos y estructuración desde una fuente como JSON

¿Qué es un modelo de datos en Flutter?

Un modelo de datos en Flutter representa la estructura de la información que la aplicación maneja. Se define mediante clases en Dart, que organizan los datos y facilitan su manipulación. En aplicaciones modernas, los datos suelen provenir de APIs REST, bases de datos locales o archivos JSON.

Cómo estructurar datos desde JSON en Flutter

Cuando se recibe información en formato JSON (JavaScript Object Notation), es necesario convertirla en objetos Dart. Este proceso se conoce como deserialización.

Conversión de JSON a modelo Dart (fromJson)

Para interpretar los datos de un JSON en un objeto de Dart, se usa un constructor `fromJson()`.

Ejemplo:

```
{
  "nombre": "Carlos",
  "edad": 25,
  "email": "carlos@example.com"
}
class Usuario {
  final String nombre;
  final int edad;
  final String email;

  Usuario({required this.nombre, required this.edad, required
this.email});

  factory Usuario.fromJson(Map<String, dynamic> json) {
    return Usuario(
      nombre: json['nombre'],
      edad: json['edad'],
      email: json['email'],
    );
  }
}
```

Conversión de modelo Dart a JSON (toJson)

Cuando se necesita enviar datos a una API o guardar información localmente, se debe convertir el objeto Dart a JSON.

```
class Usuario {
  String nombre;
  int edad;
  String email;
```

```
  Usuario({required this.nombre, required this.edad, required
this.email});

  Map<String, dynamic> toJson() {
    return {
      'nombre': nombre,
      'edad': edad,
      'email': email,
    };
  }
}
```

Uso de json_serializable para conversión automática

En aplicaciones grandes, escribir fromJson() y toJson() manualmente puede ser tedioso. Flutter ofrece el paquete json_serializable para generar automáticamente estas funciones.

Ejemplo de implementación:

Instalar dependencias en `pubspec.yaml`:

```
dependencies:
  json_annotation: ^4.8.1

dev_dependencies:
  build_runner: ^2.4.5
  json_serializable: ^6.7.1
```

Crear el modelo con anotaciones:

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

part 'usuario.g.dart';

@JsonSerializable()
class Usuario {
  String nombre;
  int edad;
  String email;

  Usuario({required this.nombre, required this.edad, required
this.email});

  factory Usuario.fromJson(Map<String, dynamic> json) =>
_$UsuarioFromJson(json);
  Map<String, dynamic> toJson() => _$UsuarioToJson(this);
}
```

Generar el código automático ejecutando:

```
flutter pub run build_runner build
```

Paquete proveedor de datos (Provider y/o Bloc)

¿Qué es el manejo de estado en Flutter?

Flutter utiliza un árbol de widgets donde el estado puede cambiar. Para compartir y administrar datos entre widgets, se usan patrones de estado como Provider y Bloc.

¿Qué es Provider y cómo funciona?

Provider es un paquete recomendado por Flutter para la gestión del estado. Permite compartir datos de manera eficiente sin necesidad de pasarlos manualmente a través de los widgets.

Ejemplo básico de Provider

Instalar el paquete en `pubspec.yaml`:

```
dependencies:
  provider: ^6.1.0
```

Crear un modelo de estado con **`ChangeNotifier`**:

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

class ContadorProvider with ChangeNotifier {
  int _contador = 0;

  int get contador => _contador;

  void incrementar() {
    _contador++;
    notifyListeners(); // Notifica a los widgets que deben reconstruirse
  }
}
```

¿Qué es Bloc y cómo funciona?

Bloc (Business Logic Component) usa eventos y estados para manejar datos de manera más estructurada.

Ejemplo básico de Bloc

Instalar dependencias:

```
dependencies:
  flutter_bloc: ^8.1.3
```

Crear un Cubit para manejar el estado:

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

class ContadorCubit extends Cubit<int> {
  ContadorCubit() : super(0);

  void incrementar() => emit(state + 1);

}
```

Diferencias entre Provider y Bloc

Característica	Provider	Bloc
Simplicidad	Fácil de usar	Más complejo
Escalabilidad	Adecuado para apps pequeñas	Mejor en apps grandes
Patrón	ChangeNotifier	Eventos y estados

Servicio: carga de datos locales y remotos, transformación de JSON al modelo Dart

¿Qué es un servicio en Flutter?

Un servicio encapsula la lógica de obtención de datos. Puede obtener información de:

APIs REST (mediante http)

Bases de datos locales (SQLite, Hive, SharedPreferences)

Ejemplo de servicio con http para datos remotos

```
import 'dart:convert';
import 'package:http/http.dart' as http;

class UsuarioService {
  Future<Usuario> obtenerUsuario() async {
    final response = await
http.get(Uri.parse('https://api.ejemplo.com/usuario'));

    if (response.statusCode == 200) {
      return Usuario.fromJson(jsonDecode(response.body));
    } else {
      throw Exception('Error al cargar los datos');
    }
  }
}
```

Bibliografia

Bloc state management library. (s/f). Bloc. Recuperado el 31 de marzo de 2025, de https://bloclibrary.dev/

Flutter documentation. (s/f). Flutter.dev. Recuperado el 31 de marzo de 2025, de https://docs.flutter.dev/

Json_serializable. (s/f). Dart Packages. Recuperado el 31 de marzo de 2025, de https://pub.dev/packages/json_serializable

Provider. (s/f). Dart Packages. Recuperado el 31 de marzo de 2025, de https://pub.dev/packages/provider