**Tema 7. Desarrollo con Flutter**

1. **Introducción:**

* **¿Qué es Flutter?:** Es un framework de código abierto desarrollado por Google que permite crear una UI para varias plataformas con un único código base. Utiliza Dart para beneficiarse de la sencillez que ofrece para crear UI así como el control de nulos (errores comunes). Ofrece un rendimiento similar al de un desarrollo nativo pero en diferentes plataformas.

1. **Instalación de Flutter:**

* **Descargar versión para Windows:** Extraer el .zip y guardar la carpeta *“flutter”* en un path sin privilegios (se recomienda C:\src\flutter).
* **Modificar la variable de entorno:** En la barra de búsqueda escribir *“env”* y abrir el gestor de variables de entornos de la cuenta. En la variable PATH, clicar editar y añadir una nueva entrada con el path hacia flutter\bin (por ejemplo, C:\src\flutter\bin).
* **Abrir la consola de Windows (*“cmd”* en búsqueda):** El comando *“where flutter dart”* confirma que Dart está instalado con Flutter. El comando *“flutter doctor”* inspecciona si todo está correctamente instalado.
* **Problemas:**
* **Actualizar Android Studio:** Una vez instalado Android Studio, se deben añadir varios elementos:
  + Actualizar el Android SDK Build Tool.
  + Instalar el Android SDK Command-line Tools ( en File/Settings/Android SDK).
* **Aceptar las licencias:** El comando *“flutter-config –android-licenses”* abre el configurador de licencias. Se deben aceptar todas las licencias para continuar.
* **Instalar el kit de desarrollo para Visual Studio:** Instalar “Desarrollo para escritorio” en Visual Studio.
* **Seleccionar el editor:** Se puede elegir VS Code, IntelliJ o Android Studio. En todos ellos hay que instalar un plugin para que funcione Flutter.

1. **Nuevo proyecto con Android Studio:**

* **Crear el proyecto:** Desde File/New/New Flutter Project se crea un nuevo proyecto.
* **Seleccionar Flutter:** En la ventana de Nuevo Proyecto, seleccionar Flutter y añadir el path del SDK de Flutter.
* **Configurar el proyecto:** Se le da nombre, se indica la librería del proyecto y se indica que se va a desarrollar Application.
* **Configurar el emulador:** Se puede importar el emulador de Android ya creado o utilizar uno nuevo.

1. **Nuevo proyecto con VS Code:**

* **Crear el proyecto:** Desde View/Command Palette, lanzar Flutter: New Project.
* **Configurar el proyecto:** Se indica que se va a desarrollar Application y se le da nombre.
* **Configurar el emulador:** En la barra azul inferior, se indica si hay algún dispositivo conectado o No devices. Si no hay dispositivos, se puede añadir el emulador creado para Android Studio.

1. **Hot reload:** No es necesario reiniciar la aplicación al hacer modificaciones, sino que se puede pulsar el botón *“Hot reload”* (el rayito) y se aplicarán los cambios.
2. **Primera aplicación con Flutter (Codelab usando VS Code):**

* **Crear nuevo proyecto:** Desde el Command Palette de VS Code (F1) > Flutter: New Project > Application. Seleccionar ubicación y nombrar la aplicación.
* **Comprobar el archivo pubspec.yaml:** Este archivo es el que controla las dependencias y funcionalidades externas del proyecto. Se deben incluir las dependencias del proyecto.
* **Rebajar las necesidades de Dart en el archivo análisis\_options.yaml:** El intérprete de Dart es muy potente, pero para aplicaciones sencillas no se necesitan todas sus funcionalidades. Por eso, se ponen a false muchas de sus características.
* **Reemplazar el código de lib/main.dart:** Para trabajar con una app personalizada en lugar de con la que viene por defecto.
* **Lanzar la aplicación:** Antes, hay que comprobar que se está ejecutando en el dispositivo adecuado: se puede comprobar en la parte derecha, debajo, de la aplicación. Para ejecutar, pulsar en el botón arriba a la derecha con el bug (Start Debugging):

****

* **Realizar un pequeño cambio en la aplicación:** Lo mejor que tiene Dart y Flutter es que se pueden hacer modificaciones en la aplicación sin necesidad de reiniciarla (lo que se conoce como Stateful Hot Reload), de modo que solamente se lanza la aplicación una vez. El estado de la aplicación se mantiene, cambiando solamente ese pequeño elemento que se ha modificado.
* **Widget:** Es el elemento esencial de Flutter, todo es un widget. Los widget vienen ya predefinidos (Column, Row, Text), pero también se pueden crear usando clases. Para ello, se debe indicar si tienen estado (guardan información que se desea almacenar) o no, por lo que heredarán de una clase u otra (StatelessWidget o StatefulWidget).
* **Entendiendo Flutter:**
* **Función main:** Todas las aplicaciones Flutter comienzan con esta función. Básicamente, indica que se ejecute la aplicación:

void main() {

  runApp(MyApp());

}

* **Clase MyApp:** Se dice que en Flutter todo lo que se muestra en pantalla es un widget, por lo que extiende la funcionadlidad de la clase StatelessWidget. Todos los widgets tienen un método (heredado de la clase StatelessWidget) que se debe sobrescribir: el método **build**. Básicamente, indica a la aplicación qué elementos contiene ese widget. En concreto, para la clase MyApp se añade un widget MaterialApp con varios children útiles: title, theme (que incluye un esquema de colores y el tipo de botones -useMaterial3-) y home.
* **MaterialApp:** Crea aplicaciones con estilos de Google, basadas en Material Design. Si se utiliza Cupertino es para crear aplicaciones de iOS. Ambos llevan varias etiquetas predefinidas (title, home, etc.).
* **SafeArea:** Permite definir un margen para añadir título, iconos, navegación, etc.
* **Scaffold (andamio):** Es un widget útil para mostrar mensajes o ventanas, pues permite construir elementos. Viene con algunas propiedades predefinidas que facilitan la construcción (appBar, body, floatingActionButton, etc.).
* **Material:** Widget de hoja en blanco.
* **AppBar:** Barra superior con icono, título, accesos directos, etc.
* **Expanded:** Widget que crece hasta ocupar todo el espacio disponible.
* **Column:** Otro widget muy útil, que permite ordenar elementos verticalmente. Se organiza en children (elementos hijos), que se colocan uno sobre otro. Ejemplo:

return Scaffold(

      body: Column(

        children: [

          Text('A random AWESOME idea:'),

          Text(appState.current.asLowerCase),

          ElevatedButton(

            onPressed: () {

              print('button pressed!');

            },

            child: Text('Next'),

          ),

        ],

      ),

    );

* **Row:** Lo mismo que Column, pero en horizontal.
* **Text:** Widget que permite mostrar texto en pantalla.
* **ElevatedButton:** Widget para botones. Tiene propiedades como onPressed u onTap.
* **Stack:** Permite la navegación por pilas.
* **Container:** Para márgenes, rellenos, etc.
* **ElevatedButton:** Widget que muestra un botón. Se puede combinar con un Text como child para mostrar un texto dentro del botón.
* **Center:** Es un widget que permite centrar otros widgets.
* **SizedBox:** Widget que permite colocar elementos vacíos para hacer espacio.
* **Placeholder:** Crea un cuadrado vacío que sirve de elemento temporal para comprobar funcionalidad.
* **Gestionar los cambios en la aplicación:** Para poder controlar los cambios en la aplicación se emplea una clase que herede de ChangeNotifier, en este caso, MyAppState. Esta clase contiene elementos que cambian entre estados de la aplicación. Por ejemplo, una palabra, que, al cambiar, se notifica a aquellos elementos que estén escuchando:

class MyAppState extends ChangeNotifier {

  var current = WordPair.random();

  void getNext() {

    current = WordPair.random();

    notifyListeners();

  }

}

**→ ¿Cómo crear un elemento de escucha?:**

var appState = context.watch<MyAppState>()

* **Elementos visuales:** Es importante tratar elementos en widgets individuales en lugar de hacer widgets enormes, sino construir widgets dentro de widgets. Para hacerlo sencillo, se puede extraer el widget (Ctrl + . > Extract Widget).
* **Widget Inspector:** Es la lupa que está arriba del todo. Permite inspeccionar individualmente cada elemento. Al abrir el panel lateral, se puede seleccionar Widget mode para inspeccionar cada elemento:

A screenshot of a computer

Description automatically generated

* **Elementos privados:** En Dart, la visibilidad privada se determina con una barra baja ( \_ ) al inicio del nombre de la clase, método o propiedad.

1. **Trabajo con Flutter:**