

| Kotlin

📄 Introducción a la programación en Kotlin | Android Basics Compose - First Android app | Android Developers

Aprende conceptos de programación introductorios en Kotlin y crea apps para Android en Kotlin.

<https://developer.android.com/courses/pathways/android-basics-compose-unit-1-pathway-1?hl=es-419&authuser=1#codelab->

<https://developer.android.com/codelabs/basic-android-kotlin-compose-variables>

| Basics

Tipo de datos de Kotlin	Qué tipo de datos puede contener	Ejemplos de valores literales
<code>String</code>	Texto	<code>"Add contact"</code> <code>"Search"</code> <code>"Sign in"</code>
<code>Int</code>	Número entero	<code>32</code> <code>1293490</code> <code>-59281</code>
<code>Double</code>	Número decimal	<code>2.0</code> <code>501.0292</code> <code>-31723.99999</code>
<code>Float</code>	Número decimal (que es menos preciso que un <code>Double</code>). Tiene un <code>f</code> o <code>F</code> al final del número.	<code>5.0f</code> <code>-1630.209f</code> <code>1.2940278F</code>
<code>Boolean</code>	<code>true</code> o <code>false</code> . Usa este tipo de datos cuando solo haya dos valores posibles. Ten en cuenta que <code>true</code> y <code>false</code> son palabras clave en Kotlin.	<code>true</code> <code>false</code>

para la instanciación de las variables, tendremos que poner el nombre, el tipo de dato y finalmente el valor que le asignemos.

`val` **name** `:` **data type** `=` **initial value**

para incorporar la variable en los mensajes de impresión por consola, lo haremos igual que en php, añadiendo el símbolo \$

```
fun main() {  
    val count: Int = 2  
    println("You have $count unread messages.")  
}
```

VALOR FIJO & VARIABLE

- Palabra clave **val**: Úsala cuando esperes que el valor de la variable no cambie. (solo de lectura)
- Palabra clave **var**: Úsala cuando esperes que el valor de la variable pueda cambiar. (es modificable, cambiante)
- **INFERENCIA DE DATOS**

La inferencia de datos en programación es una característica que permite el tipo de dato de una variable o expresión sin que el programador tenga que especificarlo explícitamente. En el contexto de Kotlin, esto significa que:

- El compilador puede determinar el tipo de una variable basándose en el valor que se le asigna inicialmente.
- No es necesario declarar explícitamente el tipo de dato de una variable si se puede inferir del contexto.
- Esto hace que el código sea más conciso y legible, reduciendo la verbosidad.

Por ejemplo, en Kotlin:

```
val nombre = "Juan" // El compilador infiere que es de tipo String  
val edad = 25 // El compilador infiere que es de tipo Int  
val altura = 1.75 // El compilador infiere que es de tipo Double
```

En estos casos, no es necesario especificar explícitamente **String**, **Int** o **Double**, ya que Kotlin puede inferirlo automáticamente. Esto simplifica la escritura del código y reduce la posibilidad de errores al declarar tipos.

- **FUNCIONES**

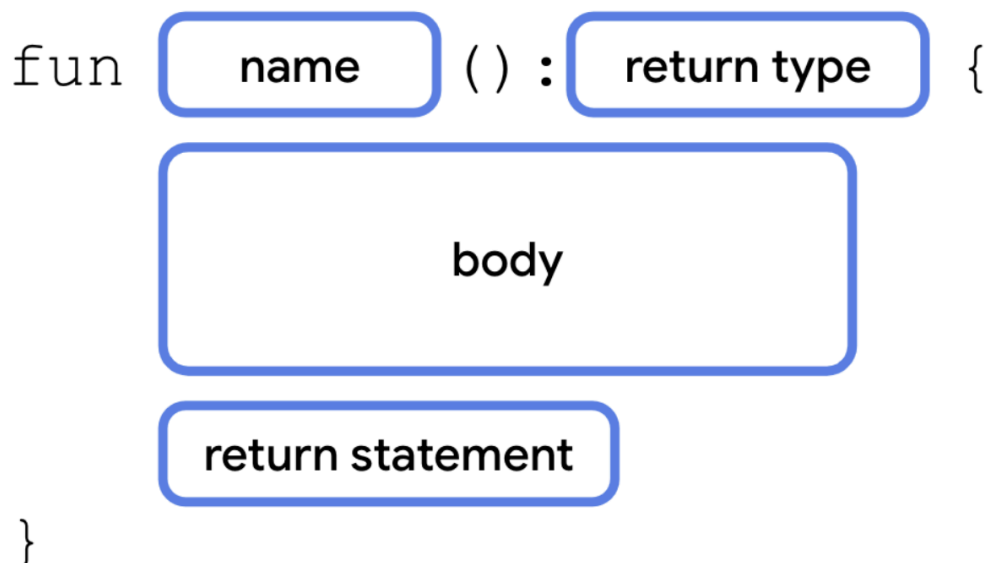
- Las funciones se definen con la palabra clave **fun** y contienen fragmentos de código reutilizables.
- Las funciones facilitan el mantenimiento de los programas más grandes y **evitan la repetición innecesaria de código**.
- Las funciones pueden mostrar un valor que puedes almacenar en una variable para usarlo más tarde.
- Las funciones pueden tomar **parámetros**, que son variables disponibles dentro del cuerpo de una función.
- Los **argumentos** son los valores que pasas cuando llamas a una función.
- Puedes **nombrar argumentos** cuando llamas a una función. Cuando usas argumentos con nombre, puedes reordenarlos sin afectar el resultado.
- Puedes especificar un **argumento predeterminado** que te permita omitirlo cuando llames a una función.

- sintaxis para la definicion:

```
fun name ( ) {  
  
    body  
  
}
```

- COMO DEVOLVER EL VALOR DE UNA FUNCION
 - en la declaracion de las funciones, se puede especificar el tipo de dato que queremos que se muestre, o el tipo de dato que nos retornará la función, como

por ejemplo un String



- en el caso de que no defina el tipo de la funcion, será de tipo `Unit`, es decir q la funcion no muestra ningun valor → es equivalente a los tipos nulos de datos que se muestran en otros lenguajes (`void` en Java) → se le pueden pasar parametros a las funciones, pero estos serán inmutable, no podremos reasignar el valor del parametro desde el cuerpo de la funcion
- FUNCIONES CON VARIOS PARÁMETROS
 - en las funciones se pueden definir mas de un parametro, incluso de diferentes tipos de datos.

→ **separaremos** los parametros **con comas** dentro de los parentesis que definen la funcion

🔄 los parametros tendrán que estar definidos en la funcion main pero llamando a la funcion a la que le vamos a pasar los parametros

```
fun main() {  
    println(birthdayGreeting("Luka", 3))  
    println(birthdayGreeting("Noa", 2))  
}  
  
//le pasamos los parametros a la funcion definidos en el método main  
fun birthdayGreeting(name:String, age:Int): String{  
    val nameGreeting = ("Happy Birthday, $name!")  
    val ageGreeting = ("You are now $age years old!")  
    return "$nameGreeting\n$ageGreeting\n"  
}
```

- la definicion del nombre de la funcion, los parametros, los return.. se conoce como
⇒ **FIRMA DE LA FUNCIÓN**

```

    fun birthdayGreeting(name: String, age: Int): String {
        val nameGreeting = "Happy Birthday, $name!"
        val ageGreeting = "You are now $age years old!"
        return "$nameGreeting\n$ageGreeting"
    }
  
```

SINTAXIS DE LAS FUNCIONES

- ARGUMENTOS CON NOMBRE

El ejemplo de código anterior, al que le pasábamos los parámetros de nombre y edad, no le hemos pasado el nombre como argumento, es decir lo ha cogido por el orden en el que lo hemos puesto en la función Main. Pero también podemos pasar los **argumentos con nombre** y podremos disponerlo sin importar el orden:

```

fun main() {
    /*como vemos el orden ha cambiado, y le hemos pasado nombre a los
    argumentos*/
    println(birthdayGreeting(age=3, name="Luka"))
    println(birthdayGreeting(name="Noa", age=2))
}

//FUNCIONAA EXACTAMENTE IGUAL

//le pasamos los parametros a la funcion definidos en el método main
fun birthdayGreeting(name:String, age:Int): String{
    val nameGreeting = ("Happy Birthday, $name!")
    val ageGreeting = ("You are now $age years old!")
    return "$nameGreeting\n$ageGreeting\n"
}
  
```

- ARGUMENTOS PREDETERMINADOS

- la diferencia con los anteriores es que los anteriores se pueden definir en el método main, mientras q siendo de tipo predeterminado se le puede pasar el nombre, tipo y valor en el mismo pase de parámetros a nuestra función

```

fun main() {
    /*como vemos el orden ha cambiado, y le hemos pasado nombre a los
    argumentos
    println(birthdayGreeting(age=3, name="Luka"))
    println(birthdayGreeting(name="Noa", age=2))*/

    //ESTO PODRIAMOS OMITIRLO
  
```

```

}

//FUNCIONAA EXACTAMENTE IGUAL

//le pasamos los parametros a la funcion definidos en el método main
//PARÁMETROS + VALOR
fun birthdayGreeting(name:String = "Luka", age:Int = "2"): String{
    val nameGreeting = ("Happy Birthday, $name!")
    val ageGreeting = ("You are now $age years old!")
    return "$nameGreeting\n$ageGreeting\n"
}

```

- **EJEMPLO DE CONCATENACIÓN DE CADENAS**

```

fun main() {
    val numberOfAdults = 20
    val numberOfKids = 30
    //al sumar dos cadenas se juntan, no se suman aunque sean numeros
    /*val numberOfAdults = "20"
    *val numberOfKids = "30"*/
    val total = numberOfAdults + numberOfKids
    println("The total party size is: $total")

    //EJ2
    val total = ("$numberOfAdults + $numberOfKids")
    println(total)
}

```

- **EJEMPLOS DE FUNCIONES**

- función ADD ()

```

fun main() {
    val firstNumber = 10
    val secondNumber = 5
    val thirdNumber = 8

    // Llamamos a la función add
    val result = add(firstNumber, secondNumber)
    val anotherResult = add(firstNumber, thirdNumber)

    println("$firstNumber + $secondNumber = $result")
    println("$firstNumber + $thirdNumber = $anotherResult")
}

/* Definimos la función add, con los parametros de vamos a
pasarle,

```



```

que serán siempre numeros en este caso*/
fun add(a: Int, b: Int): Int {
    return a + b
}

```

- función SUBTRACT ()

```

fun main() {
    val firstNumber = 10
    val secondNumber = 5
    val thirdNumber = 8

    // Llamamos a la función subtract
    val result = subtract(firstNumber, secondNumber)
    val anotherResult = subtract(firstNumber, thirdNumber)

    println("$firstNumber - $secondNumber = $result")
    println("$firstNumber - $thirdNumber = $anotherResult")
}

// Definimos la función subtract, QUE RESTARÁ
fun subtract(a: Int, b: Int): Int {
    return a - b
}

```

- PARÁMETROS PREDETERMINADOS

→ son como las cosas que “vienen por defecto” si no dices otra cosa para usar

→ son como las opciones que Kotlin ya tiene listas, pero se pueden cambiar para hacerlo diferente

```

fun hacerGalleta(tipo: String = "chispas de chocolate") {
    println("Hiciste una galleta con $tipo.")
}

fun main() {
    hacerGalleta() // No le decimos nada, usará "chispas de chocolate"
    hacerGalleta("pasas") // Aquí elegimos "pasas" en vez de las chispas
}

```

```

fun main() {
    val firstUserEmailId = "user_one@gmail.com"
}

```

```
// The following line of code assumes that you named your
parameter as emailId.
// If you named it differently, feel free to update the name.
println(displayAlertMessage(emailId = firstUserEmailId))
println()

val secondUserOperatingSystem = "Windows"
val secondUserEmailId = "user_two@gmail.com"

println(displayAlertMessage(secondUserOperatingSystem,
secondUserEmailId))
println()

val thirdUserOperatingSystem = "Mac OS"
val thirdUserEmailId = "user_three@gmail.com"

println(displayAlertMessage(thirdUserOperatingSystem,
thirdUserEmailId))
println()
}
fun displayAlertMessage(operatingSystem:String = "ChromeBOOK",
emailId:String = "ej2@gmail.com"){
    println("There's a new sign-in request on $operatingSystem for
your Google Account $emailId")
}
```

- **EJEMPLO COMPARACIÓN NÚMEROS**

```
fun main() {
    println("tiempo de uso de hoy: ${comparacionMinutos(300, 250)}")
    println("tiempo de uso de hoy: ${comparacionMinutos(300, 300)}")
    println("tiempo de uso de hoy: ${comparacionMinutos(200, 220)}")
}
fun comparacionMinutos(minutosHoy:Int, minutosAyer:Int): Boolean{
    /*
    * con la palabra boolean lo que le estamos indicando es el TIPO
DE DATO q va a
    devolver la funcion

    *las variables están definidas en los parametros que se le pasa a
la funcion,
    y el valor se le da en main cuando se va a hacer la comparación

    */
    return minutosHoy > minutosAyer
}
```

- **EJEMPLO DISMINUIR EL CODIGO REPETIDO**

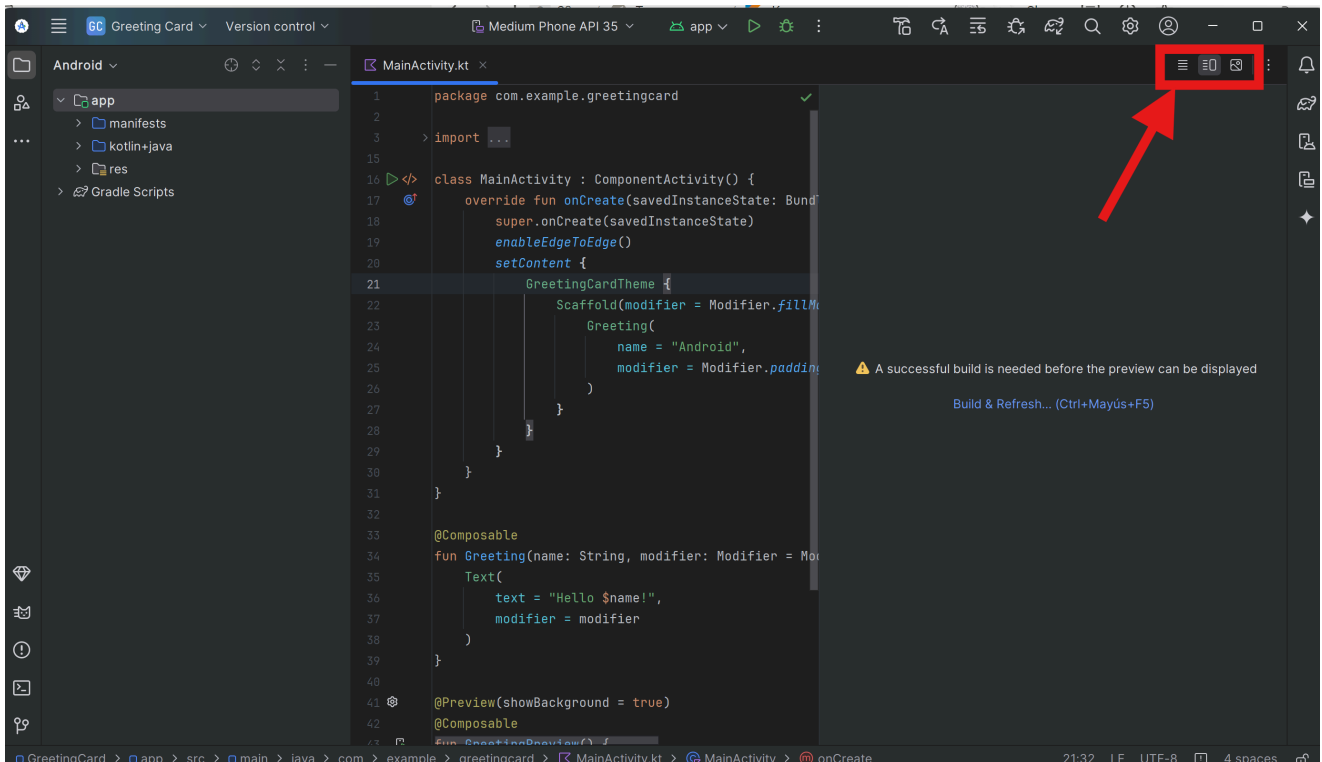

```

fun main() {
    tiempoCiudad("Madrid", 27, 31, 82)
    tiempoCiudad("Tokyo", 32, 36, 10)
    tiempoCiudad("NewYork", 59, 64, 2)
    tiempoCiudad("Paris", 50, 55, 7)
}

fun tiempoCiudad(ciudad: String, bajaTemp: Int, altaTemp: Int,
                 lluvia: Int) {
    println("Ciudad: $ciudad")
    println("temp mas baja: $bajaTemp, temp mas alta: $altaTemp")
    println("probabilidad lluvia: $lluvia %")
    println()
}

```

Como crear un proyecto



3 ÁREAS:

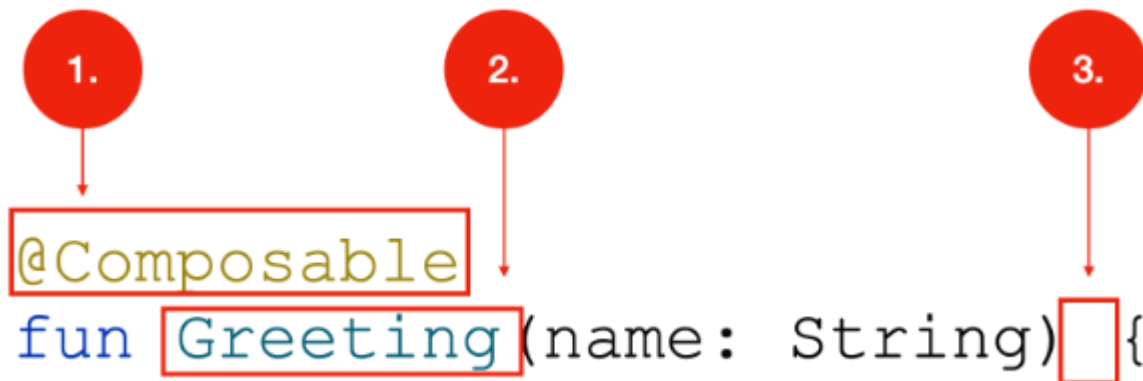
- vista Project → se muestra la estructura de carpetas y archivos del proyecto
- vista Code → para editar el código
- vista Design → vista previa de como se ve la app mientras q la vas desarrollando



CODE ⇒ para la visualización del código

SPLIT ⇒ para ver tanto el código como el diseño al mismo tiempo

DESIGN ⇒ para ver solo el diseño



- Los nombres de las funciones `@Composable` llevan mayuscula
- Antes de la función se añade la anotación `@Composable`
- esta funcion `@Composable` no pueden mostrar nada, (no tendrá ningún return devolviendo información) *me he quedado en como cambiar el color del fondo

📄 Cómo crear tu primera app para Android | Android Developers

Aprende a crear tu primera app para Android.

<https://developer.android.com/codelabs/basic-android-kotlin-compose-first-app?authuser=1&continue=https://developer.android.com/courses/pathways/android-basics-compose-unit-1-pathway-2?authuser=1&hl=es-419#5>

DUDA:

- si hay dos componentes TEXT dentro de un elemento COLUMN que tienen las mismas propiedades, **hay que ponerlas las propiedades (estilos) en cada uno de los componentes TEXT? o vale con ponerlo en COLUMN y como es padre de TEXT se heredan las propiedades??**

→ según CHATY ⇒ las propiedades no se heredan por lo que aunque sean iguales hay que hacerlas **por separado** :) → LA SOLUCION QUE DA ⇒ **crear una funcion con los estilos en los textos y llamarla desde el componente en este caso COLUMN**

```
@Composable
fun StyledText(text: String) {
    Text(
        text = text,
        fontSize = 16.sp,
        fontWeight = FontWeight.Bold
    )
}
```


```
Column {  
  StyledText("Texto 1")  
  StyledText("Texto 2")  
}
```


[imgs](#) lazyColumn lazyRow en el scaffold ⇒ en este orden, no cambiarlo de sitio


- topbar
- bottombar

⇒ en la app q estoy haciendo del crud en la vista del PROJECT → si q esta el json de firebase


Project ▾

▾  **appCrudNotas** C:\Users\beatr\AndroidSt

>  .gradle


>  .idea

▾  **app**


>  build


>  src

 .gitignore


 build.gradle.kts


 google-services.json


 proguard-rules.pro


>  gradle


 .gitignore


 build.gradle.kts


 gradle.properties


 gradlew

 gradlew.bat

 local.properties

 settings.gradle.kts

>  External Libraries

 Scratches and Consoles

```
gradle project: // Top-level build file where you can add configuration
options common to all sub-projects/modules.plugins **{**
//alias(libs.plugins.android.application) apply false
alias( libs . plugins . kotlin . android ) apply false
id("com.google.gms.google-services") version "4.4.2" apply false`**}**
//en el scaffold hay dos propiedades relacionadas con los iconos:
```

- **navigationIcon** ⇒ para menus y navegacion, en la esquina superior izquierda
- **action** ⇒ parte superior derecha

| Cambiar tema claro → tema oscuro

Para crear la paleta de colores

Material Theme Builder

Theming for Material Design 3

<https://material-foundation.github.io/material-theme-builder/>

→ la ventaja de esta pagina es que en dos variables LightColors y DarkColors agrupa los colores para cada componente

```
private val LightColors = lightColorScheme(
    primary = md_theme_light_primary,
    onPrimary = md_theme_light_onPrimary,
    primaryContainer = md_theme_light_primaryContainer,
    onPrimaryContainer = md_theme_light_onPrimaryContainer,
    secondary = md_theme_light_secondary,
```

```
private val DarkColors = darkColorScheme(
    primary = md_theme_dark_primary,
    onPrimary = md_theme_dark_onPrimary,
    primaryContainer = md_theme_dark_primaryContainer,
    onPrimaryContainer = md_theme_dark_onPrimaryContainer,
    secondary = md_theme_dark_secondary,
```

Estas son las importaciones


```
import androidx.compose.foundation.isSystemInDarkTheme
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.lightColorScheme
import androidx.compose.material3.darkColorScheme
import androidx.compose.runtime.Composable
```

Función para cambiar el tema de la aplicación, con un switch por ejemplo de claro a oscuro:

```
@Composable
fun AppTheme(
    useDarkTheme: Boolean = isSystemInDarkTheme(),
    content: @Composable() () -> Unit
) {
    val colors = if (!useDarkTheme) {
        LightColors
    } else {
        DarkColors
    }

    MaterialTheme(
        colorScheme = colors,
        content = content
    )
}
```

*no lo he terminado, me falta implementarlo para las zonas que quiero q tengan color

| ViewBinding → solo necesario cuando se trabaja con xml no con compose 😊

Qué es ?

- forma de acceder a las vistas xml desde el código

Ventajas

- accesos a las propiedades se corresponden con los tipos definidos en el layout xml

genera clases → una clase java por cada layout xml

[MVVM](#)

| Room ⇒ librería que simplifica el manejo de BD en SQLite

| su función es realizar el mapeo de objetos (ORM)

→ gestion de nuestras bases de datos

→ es como sql local

- **PARTES ⇒ para el uso de Room**

- **Database class** ⇒ configuración a la base de datos, acceso a las tablas..., maneja la conexión de la BD, y provee los DAOs
- **DAOs** ⇒ clases que aportan métodos que va a usar la aplicación (se definirán los métodos del CRUD)
- **Entities** ⇒ representan las tablas de base de datos

| tipografía

APLICAR FUENTE A LA APLICACIÓN - pasos a seguir

1. descargamos la fuente de --> <https://fonts.google.com/> (en este proyecto he descargado poppins)
2. en la carpeta res --> **creamos una carpeta font** res/font (a la altura de drawable)

“_img.png” could not be found.

3. añadimos en el archivo **build.gradle :app** las siguientes **dependencias**
`implementation ("androidx.compose.material3:material3:1.2.0")`
`implementation("androidx.compose.ui:ui-text-google-fonts:1.2.0")`
4. copiamos los archivos con la extensión **.ttf** a la carpeta **res/font** > IMPORTANTE --> el nombre del archivo .ttf tiene q estar en minúsculas y con "_" (poppins_reglar.ttf ---> no Poppins-Regular.ttf)
5. en el archivo **TYPE.KT** - incorporamos la fuente que vamos a querer utilizar

“_img_1.png” could not be found.

- definimos el esquema que va a tener nuestra tipografía en nuestro proyecto

“_img_2.png” could not be found.

6. hacemos **llamada** de nuestra fuente en el texto al que la queramos incorporar

“_img_3.png” could not be found.

| Inicio de sesión con Google

Authentication

Usuarios **Método de acceso** Plantillas Uso Configuración Extensions

Proveedores de acceso

Agregar proveedor nuevo

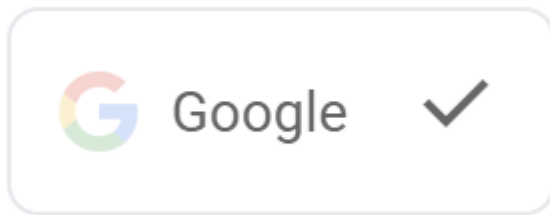
Proveedor

Estado

✉ Correo electrónico/contraseña

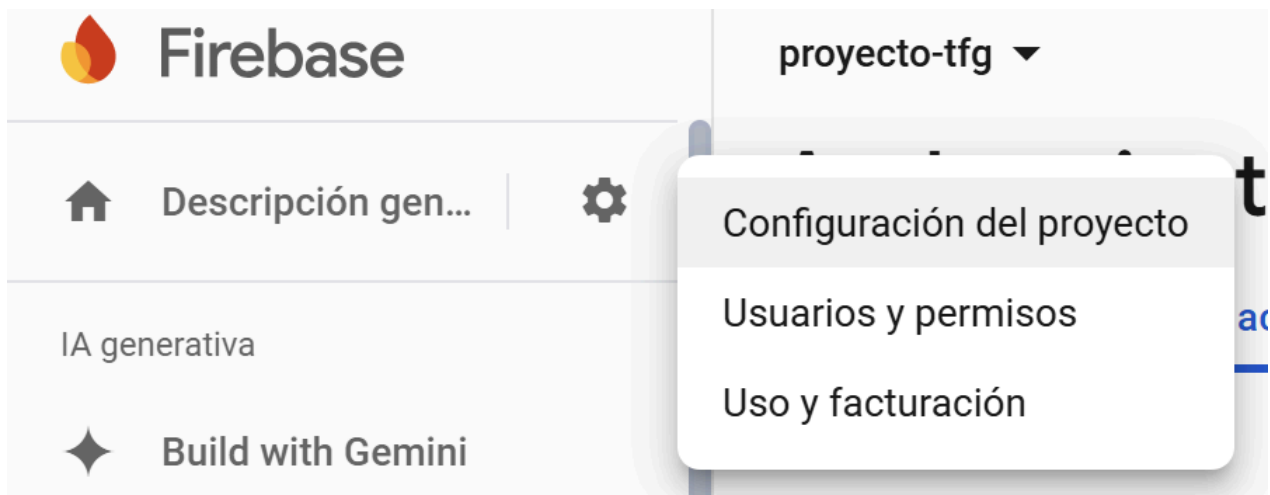
✔ Habilitado

- agregamos proveedor nuevo → seleccionamos **Google**
- habilitamos y le damos a guardar




- **huella digital sha1**

1. configuracion del proyecto




1. buscamos agregar huella digital - (hacemos scroll hasta el final de la pagina)


Apps para Android


 **proyecto-tfg-viajes**
com.appclass.myapplication

Configuración del SDK

¿Necesitas volver a configurar los SDK de Firebase en tu app? Revisa las instrucciones de configuración del SDK o descarga el archivo de configuración con las claves y los identificadores de tu app.


 [Ver las instrucciones del SDK](#)

 [google-services.json](#)

ID de la app 


1:91706639541:android:c64ea5c6956097c654aafc


Sobrenombre de la app

proyecto-tfg-viajes 

Nombre del paquete

com.appclass.myapplication

Huellas digitales del certificado SHA 

Tipo 

[Agregar huella digital](#)

Quitar esta app

1. cuando le damos a agregar, nos sale la siguiente ventana emergente, y para rellenar eso nos iremos a la terminal en nuestro proyecto, en android studio y pondremos el comando q dejo a continuacion ⇒

```
./gradlew signingReport
```

- usaremos la clave **SHA1**

- <https://developer.android.com/identity>



forgot password developer android



Todo Videos Imágenes Noticias Web Libros Finanzas

Videos :



How to Implement Forgot Password in Android Studio ✓

YouTube · Codes Easy

21 dic 2022

10 momentos clave en este video ^



CIÑETE A LO IMPORTANTE - NO TE METAS CON TONTERIAS