Android Applications Development using Kotlin

Ahmed Abdallah

Course Contents

- Kotlin what, when and why ?
- Kotlin syntax
- Android Environment Installation
- First Android App
- Android App Anatomy
- MVC Design Pattern
- Accessing Device Storage
- Networking and APIs
- Project Structure
- SOLID Principles

FAQs

- Should I have development background ?
- How sessions work ?
- What is next ?
- What is the objective ?

About Me

- Name: Ahmed Abdallah (a.k.a. Yossef)
- Title: Software Security Consultant
- Experience: 12 years
- Languages: Java, Kotlin, Swift, Python, JS, Typescript, Perl, PHP,…
- Platforms: Desktop, Web, iOS, Android, Cloud, IoT
- Security: Penetration Testing and Ethical Hacking

Kotlin Overview

Kotiln

- Kotlin is a programming language introduced by JetBrains, the official designer of the most intelligent Java IDE, named Intellij IDEA. This is a strongly statically typed language that runs on JVM.
- In 2017, Google announced Kotlin is an official language for android development. Kotlin is an open source programming language that combines object-oriented programming and functional features into a unique platform.

Why Kotlin

- Promoted by Google
- Easy Language
- Concise, you can do more with less code
- Smaller runtime and better performance
- Modern Language, easy to learn and to use

Kotlin Usage

- Kotlin is a high level programming language
- Kotlin code is compiled to either:
    - Byte code for JVM runtime environment
    - ES5 (JavaScript Compatible code)

Kotlin Usage

- Kotlin is famous for developing Android Applications, however much more can be done with Kotlin

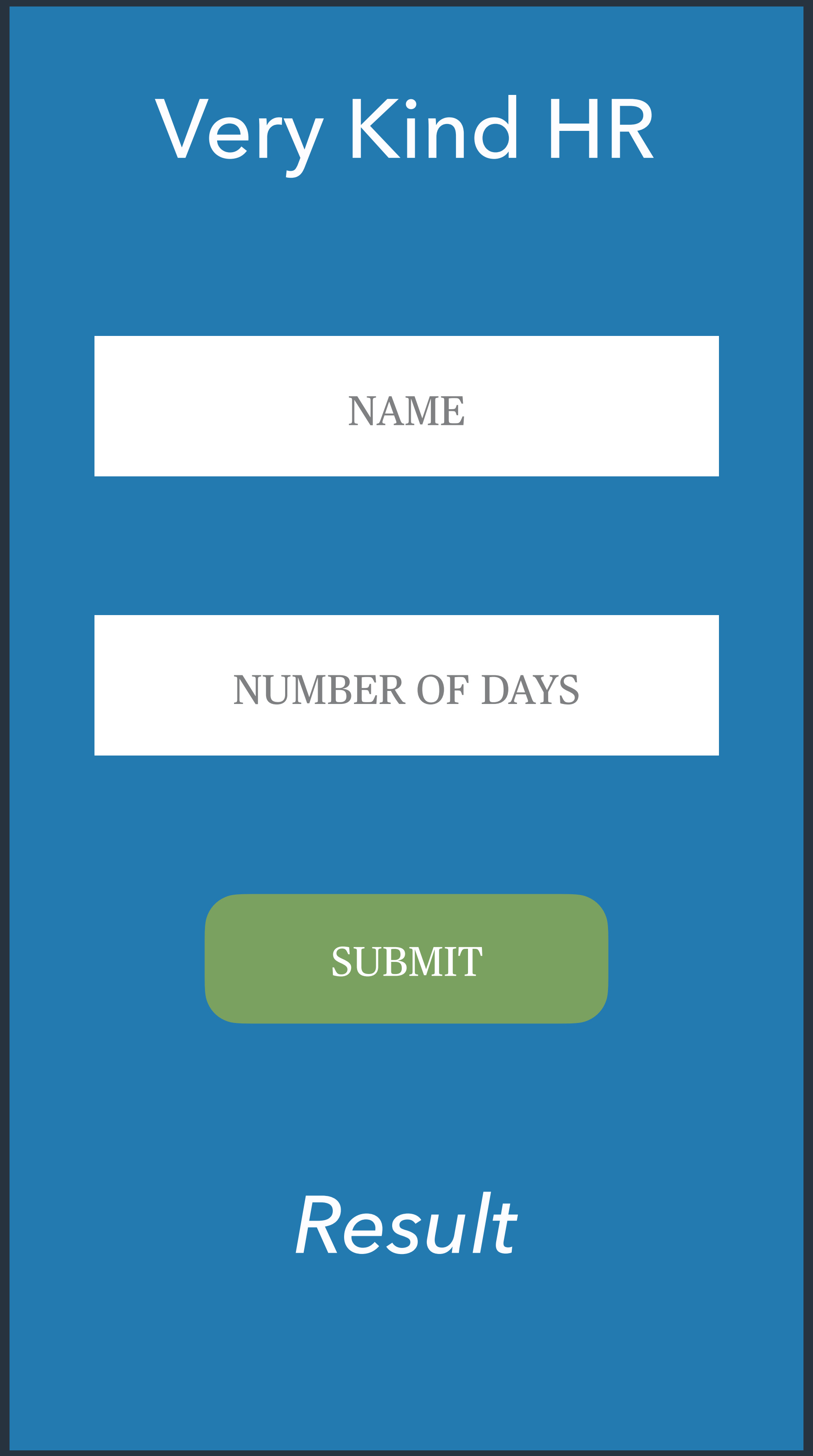
Installation

1. Java 8 installation: as kotlin uses JVM we need to install Java first
2. IDE Installation: You can choose between eclipse, netbeans, IntelliJ (Android Studio) or even your favorite editor along with kotlin command line compiler
3. Configure your tools
4. Write your first application

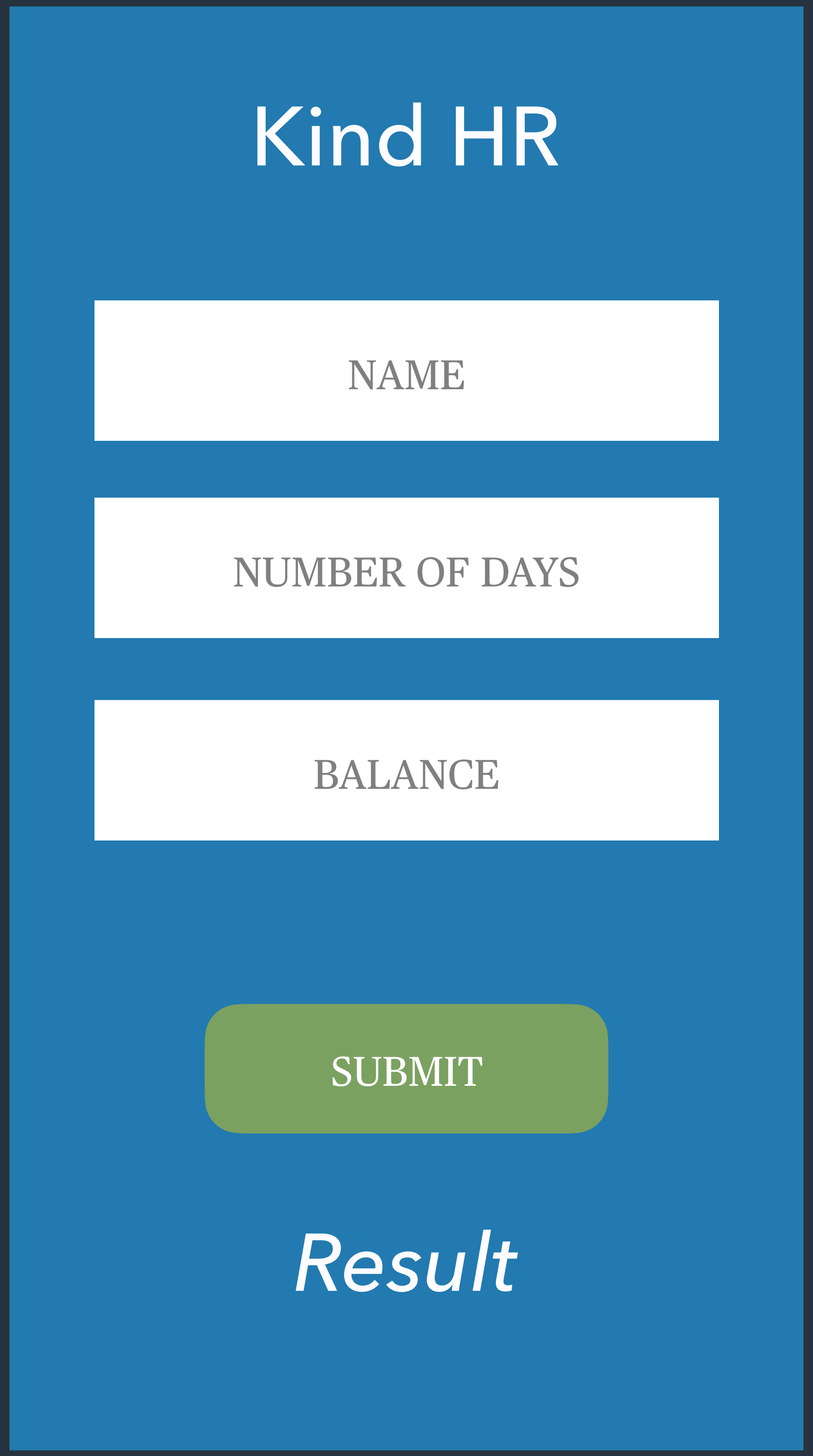
First Kotlin App

```
fun main() {
println("Hello, World!")
}
```

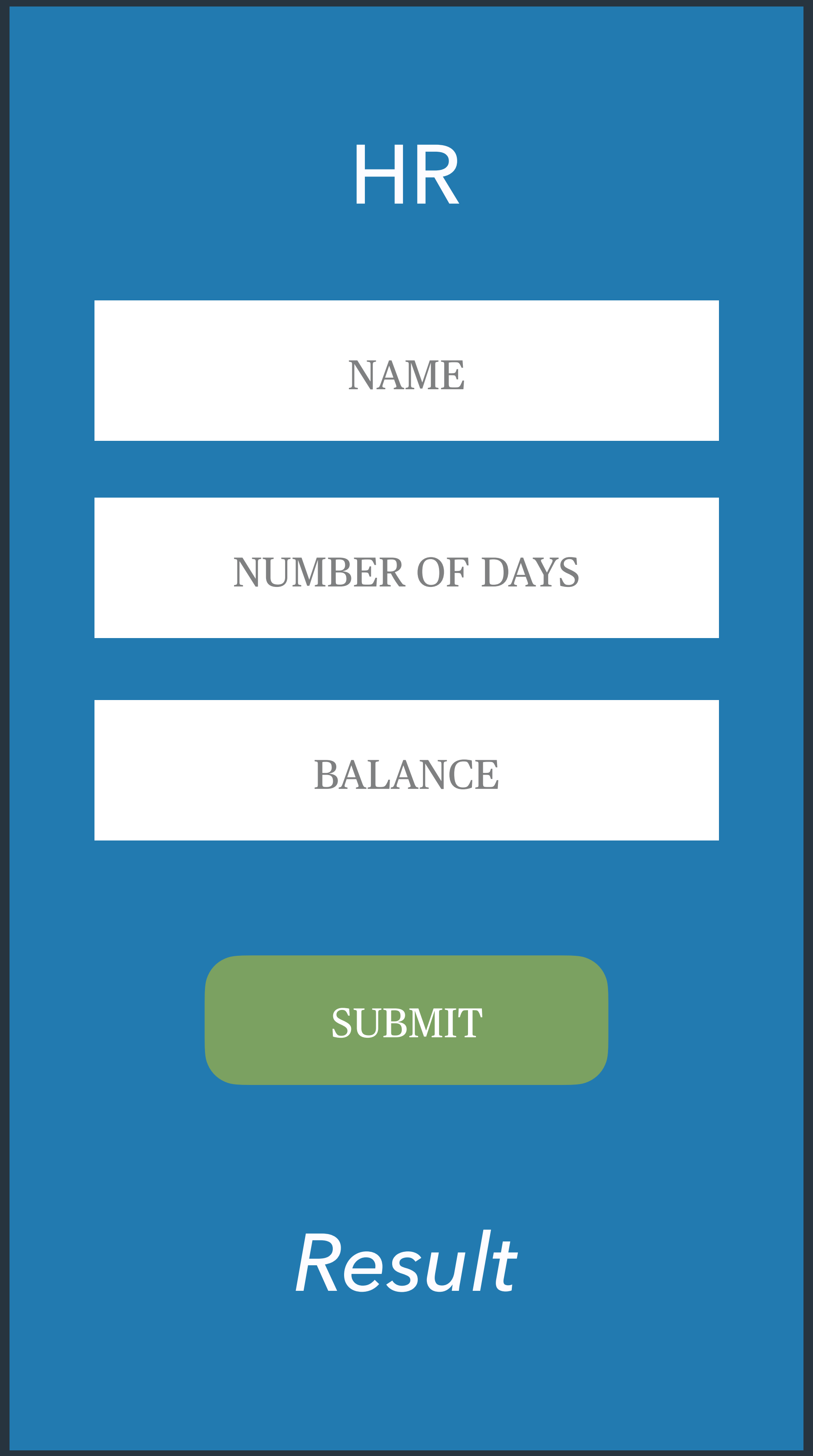
Demo



Demo



Demo



Basics of Kotlin

Variables

> To define a variable in Kotlin we use the keyword “Var”

```
var variableName: DataType
```

Constant

> To define a variable in Kotlin we use the keyword “val”

```
val constantName: DataType
```

Data Types - Numbers

Type	Size
Double	64
Float	32
Long	64
Int	32
Short	16
Byte	8

Data Types - Char

> Char: A datatype that represents a single character

```
val letter: Char     // defining a variable
letter = 'A'         // Assigning a value to it
println("$letter")
```

Data Types - String

> String: A datatype that represent a list of characters

```
val name: String    // defining a variable
name = ”Ahmed“       // Assigning a value to it
println(“$name")
```

Data Types - Boolean

> Boolean: A datatype that can hold only one of two values (true/false)

```
val flag: Boolean    // defining a variable
 flag = false        // Assigning a value to it
println("$flag")
```

Data Types - Array

- Arrays are a collection of homogeneous data

```
val numbers: IntArray = intArrayOf(1, 2, 3, 4, 5)
println(“List starts with ${numbers[0]}”)
```

Data Types - Collections

- Kotlin has two types of collection:
- **Immutable collection** it is a fixed list, map or set that cannot be changed (constant values)
- **Mutable collection** it is changeable (contains variables)

Data Types - Collections

```
val numbers: MutableList<Int> = mutableListOf(1, 2, 3) //mutable List

val numbers: List<Int> = listOf(1, 2, 3) //immutable List
```

Loops - For

```
val items = listOf(1, 2, 3, 4)
for (i in items) println("values of the array"+i)
```

Loops - For

```
val items = listOf(1, 22, 83, 4)
for ((index, value) in items.withIndex()) {
 println("the element at $index is $value")
}
```

Loops - While

```
var x:Int = 0
while(x< = 10) {
   println(x)
    x++}
```

Loops - Control

- **Continue:** skips the rest of the current iteration and go to the next iteration
- **Break:** stops the entire loop and exit to execute what is after the loop

Kotlin Functions - Lambda Function

```
fun sum(numbers:List<Int>):Int{
var sum = 0
numbers.forEach {num -> sum+=num}
return sum
}
```

Conditions

```
If (condition) {
// do something }
else {
// do another thing
}
```

When

```
when (expression) {
Val1 -> / / do something
Val2 -> / / do another thing
}
```

Kotlin Functions

```
fun sayHello(){

    println("Hello")

}

fun main(args : Array<String>){

   sayHello()

}
```

Kotlin Functions

```
fun sayHelloName(name: String){
    println(“Hello, ${name}“)
}
```

Kotlin Functions

```
fun sayHelloName(name: String):String{
    return “Hello, ${name}“
}
```

Kotlin Functions

```
fun sum(num1:Int, num2:Int):Int{
    return num1 + num2
}
```

Kotlin Functions - vararg

```
fun sum(vararg numbers:Int):Int{
var sum = 0
for (num in numbers) sum += num
return sum
}
```

Kotlin Functions - Default Values

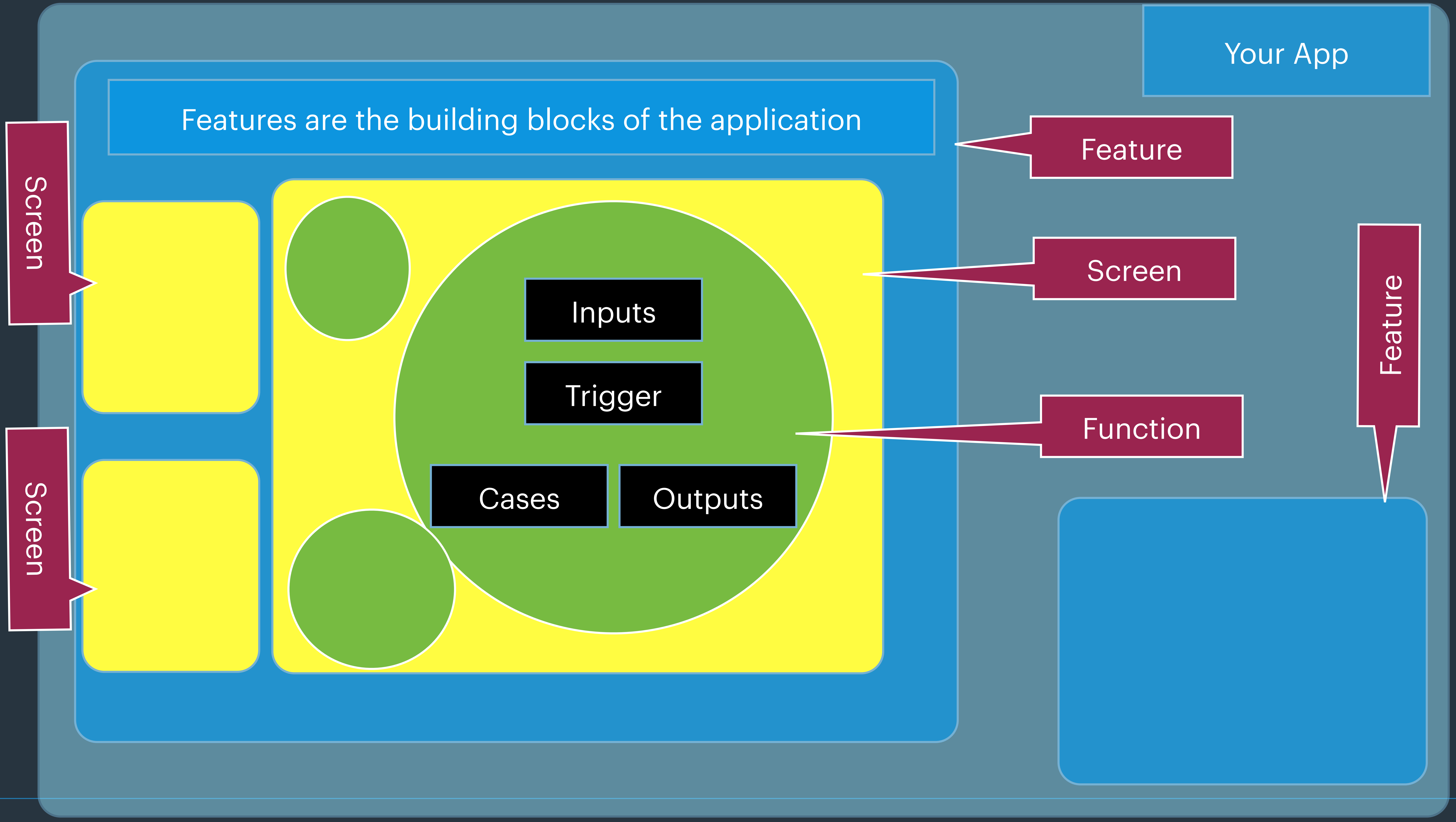
```
fun main(args: Array<String>) {
test()
test(50,”NO”)
}

fun test(num:Int= 10, str: String =“OK”){
print("Number is: $num and String is: $str")
}
```

Kotlin Functions - Inline Function

```
fun main(args: Array<String>){
//lambda function
val sum = {num1: Int, num2: Int -> num1 + num2}
println("10+5: ${sum(10,5)}")
}
```

Android App Anatomy

Application Overview



App Componentes

SCREEN (ACTIVITY)

Layout

Activity Class

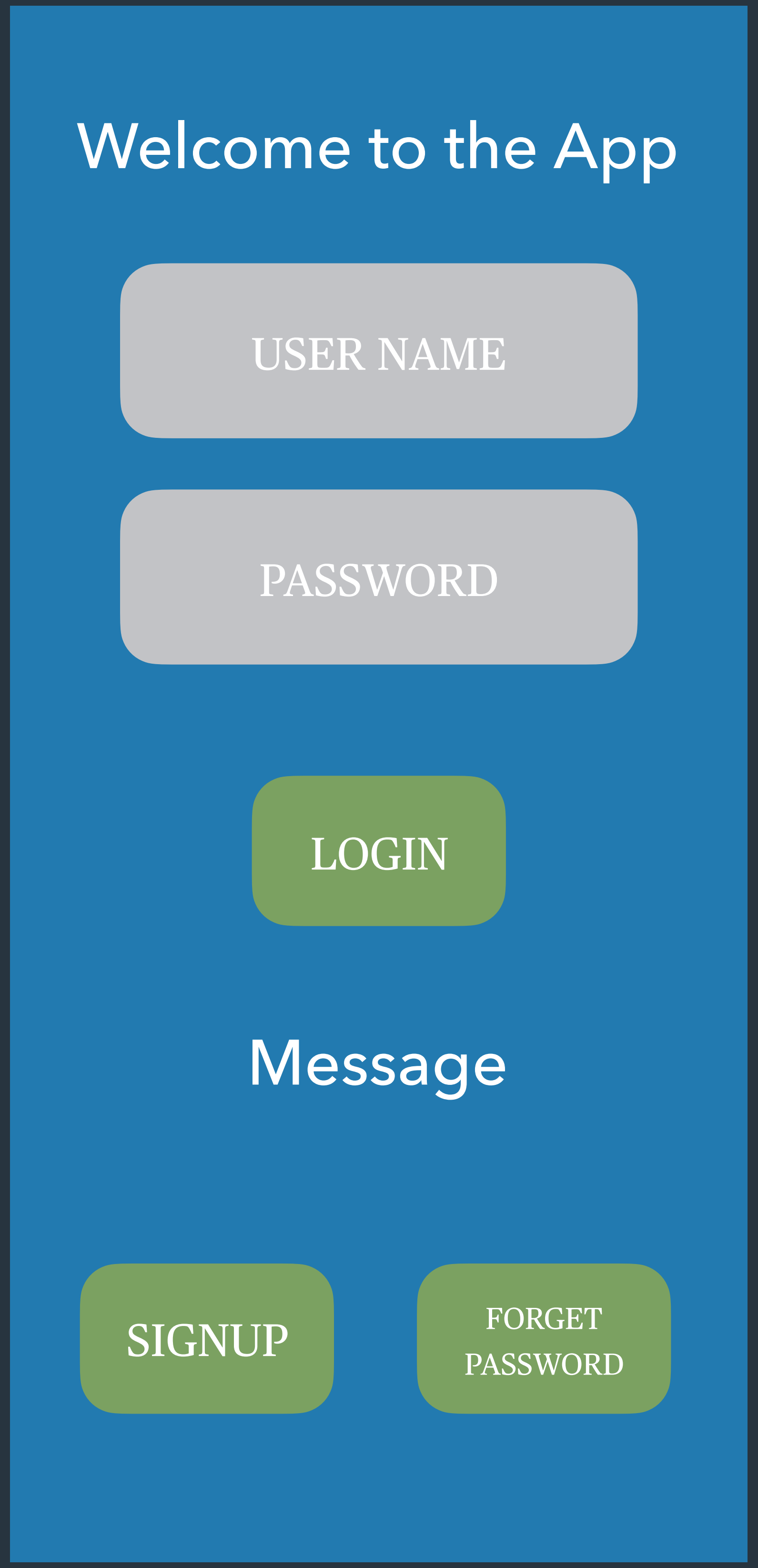
I/O Components

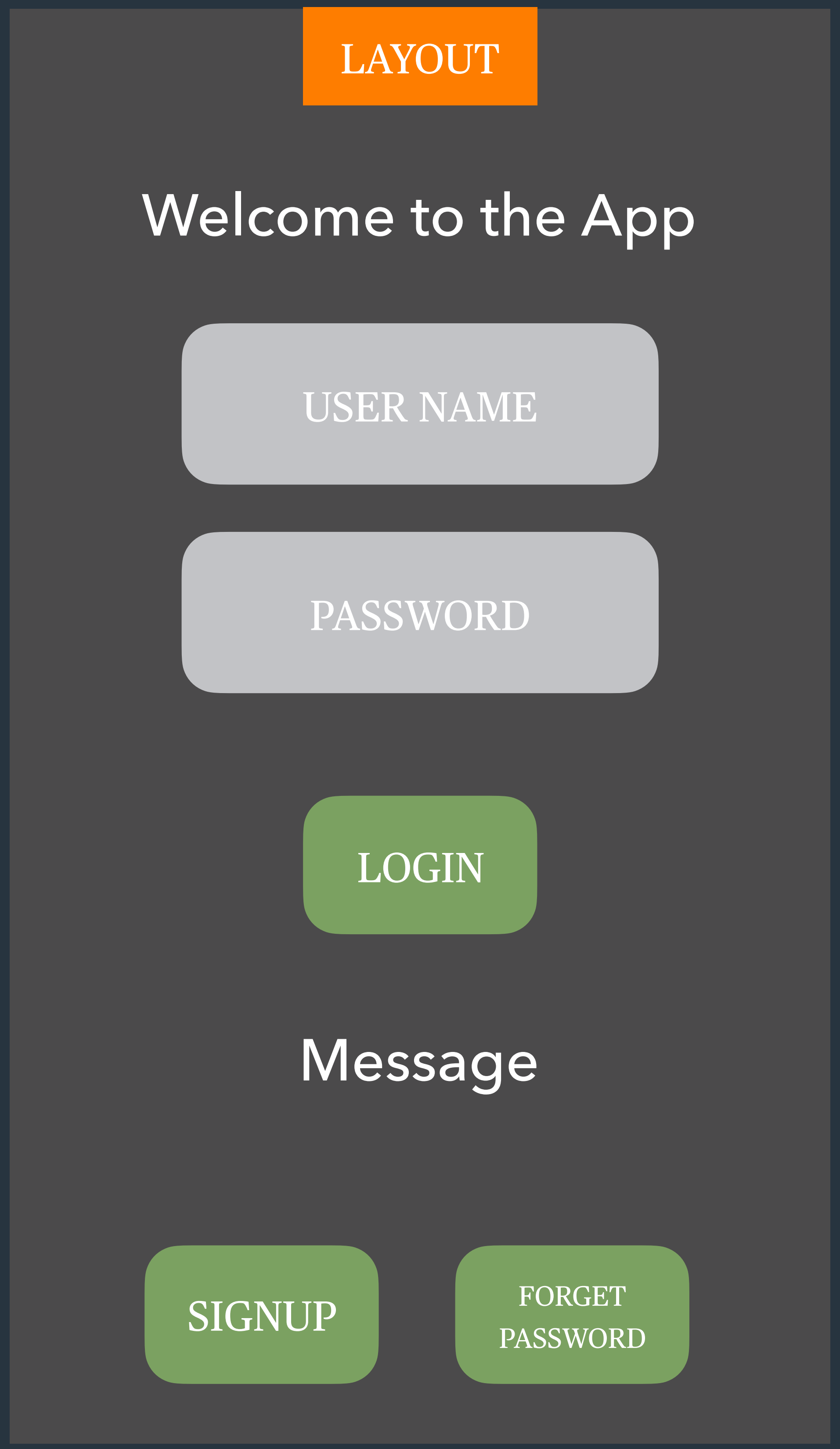
Attributes

Event Sources

Functions

Login Example





ACTIVITY CLASS

usernameEditText

passwordEditText

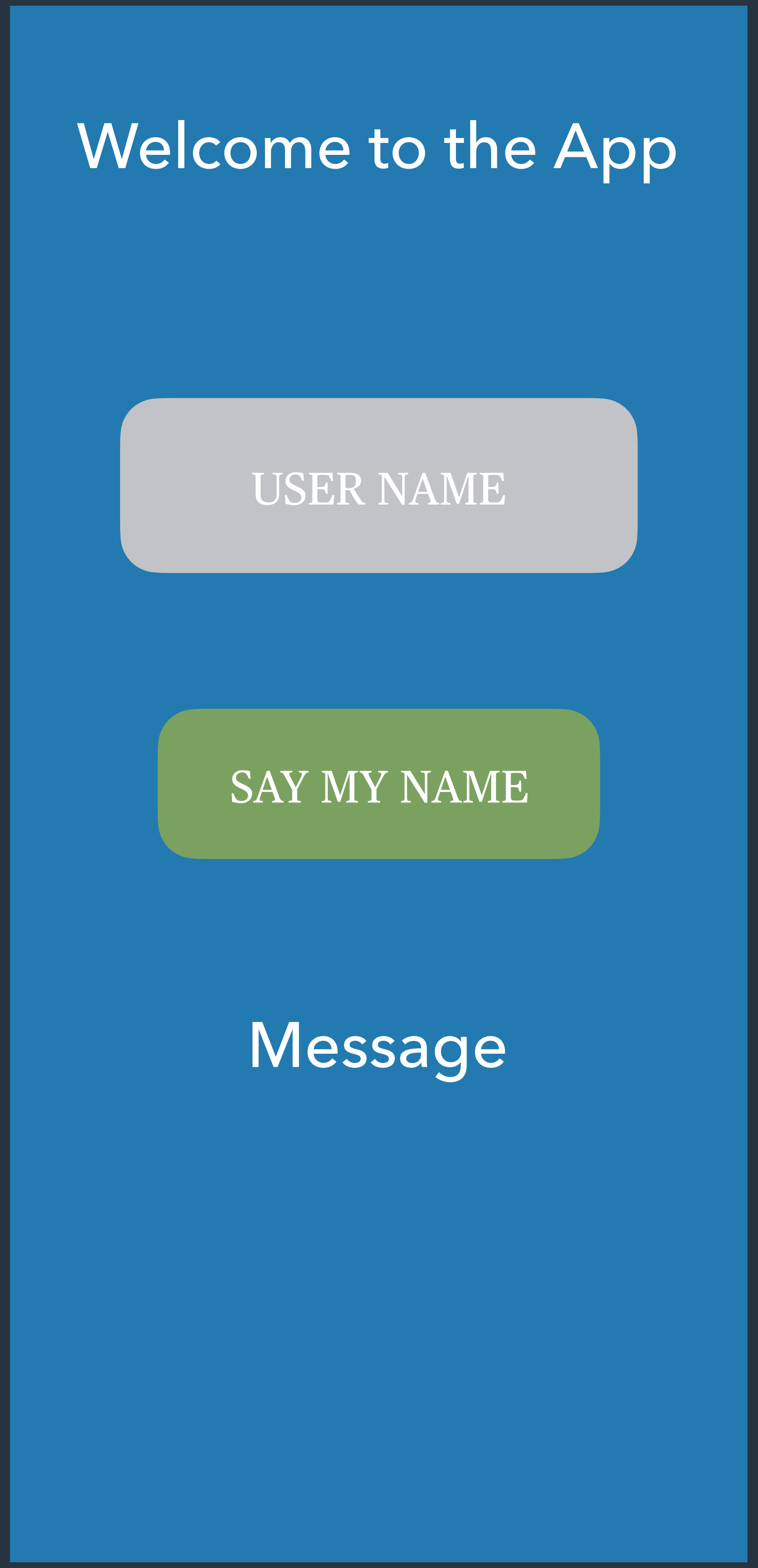
messageTextView

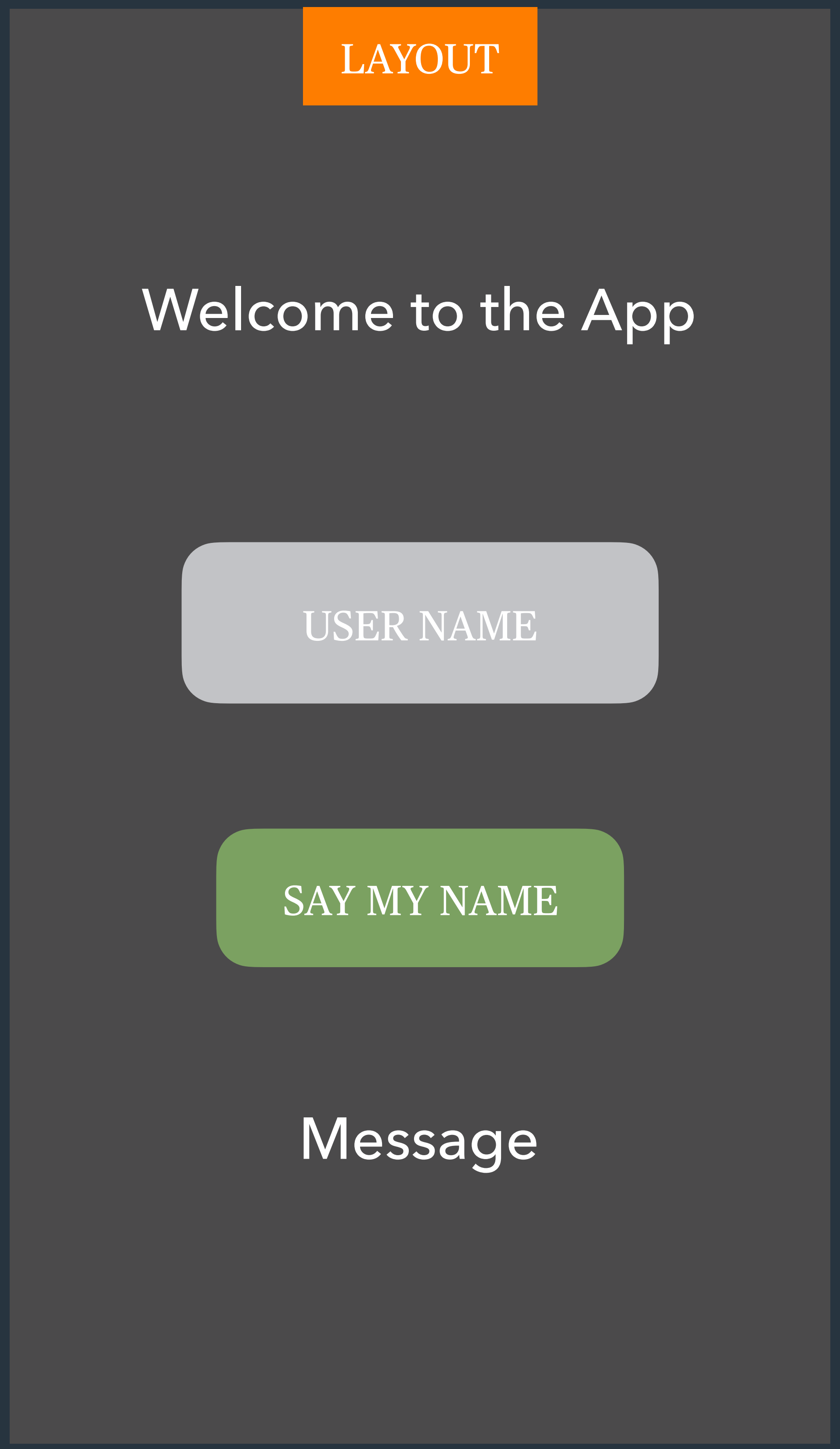
fun loginAction(){ }

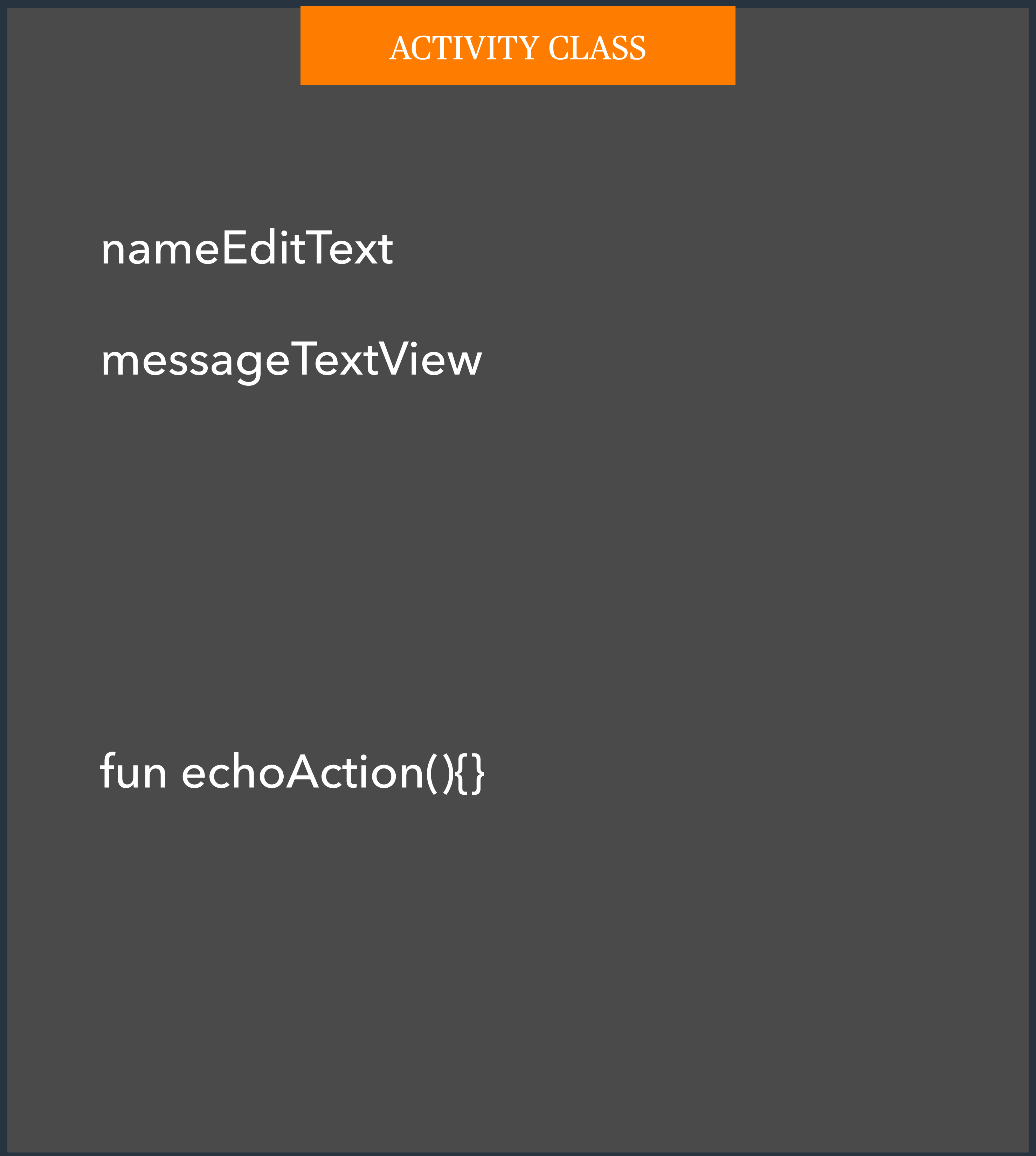
fun signupAction(){}

fun forgetPasswordAction(){}

Echo Example

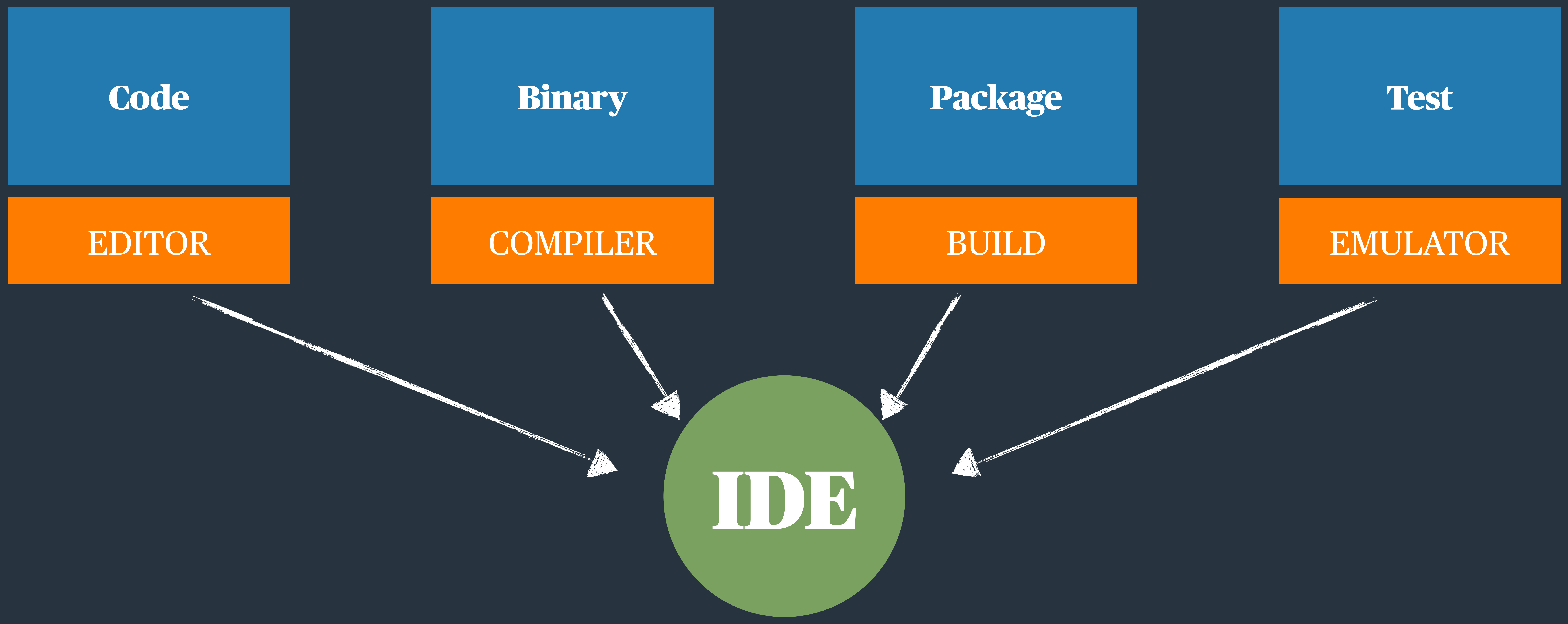






```
nameEditText

messageTextView




fun echoAction(){}
```

Development Tools



Integrated Development Environment (Android Studio)

Required Actions

- Install JDK
- Install Android Studio
- Install SDK(s)
- Create AVD(s)

MVC

MVC in Action

	View	Controller	Model
5	BUTTON CLICK	VAL1 = 5	
3	BUTTON CLICK	VAL1 = 53	
2	BUTTON CLICK	VAL1 = 532	
+	BUTTON CLICK	OPERATION = ADD	
8	BUTTON CLICK	VAL2 = 8	
=	BUTTON CLICK	CALL MODEL	SUM(532,8)
540	DISPLAY RESULT	UPDATE VIEW	RETURN 540

Extra Kotlin

Maps

```
val numbersMap = mapOf("one" to 1, "two" to 2,
"three" to 3)
```

Maps

```
val numbersMap = mapOf("one" to 1, "two" to 2,
"three" to 3)

println(numbersMap.get("one"))
println(numbersMap["one"])
println(numbersMap.getOrDefault("four", 10))
```

Maps

```
val numbersMap = mapOf("one" to 1, "two" to 2,
"three" to 3)

println(numbersMap.keys)
println(numbersMap.values)
```

Classes

```
class Number {
// data member
private var number: Int = 5
// member function
fun calculateSquare(): Int {
    return number*number
}
}
```

Objects

```
Example e = Example() // No new Keyword

//Access data member
e.number

//Access member function
e.calculateSquare()
```

Getters and Setters

```
var email:String = ""
    get() {return field}
    set(value) { field = value }
```

Getters and Setters

```
var email:String = ""
    get() {return field}
    set(value) { field = value }


val std = Student()
std.email = "ahmed@gmail.com"
println(std.email) // prints ahmed@gmail.com
```

Getters and Setters

```kotlin
var name:String? = null
    get() { return field?.substring(1) }
    set(value) { field = value?.toUpperCase() }
```

Getters and Setters

```
var name:String? = null
    get() { return field?.substring(1) }
    set(value) { field = value?.toUpperCase() }

val std = Student()
std.name = "Ahmed"
println(std.name) // prints HMED
```

Nullable

```
var a: String = "abc"
a = null // compilation error

var b: String? = "abc"
b = null // ok
print(b?.length) // prints null
print(b!!.length) // NPE
```

Nullable - Check for Null

```
val l: Int = if (b != null) b.length else -1
```

Nullable - Check for Null - Elvis

```
val l: Int = if (b != null) b.length else -1

val l = b?.length ?: -1 // Elvis ?:
```

Constructors

- We have two different types of constructors in Kotlin:
- Primary Constructors
- Secondary Constructors

Primary Constructor

```
fun main(args: Array<String>) {

   val stu = Student("Ali", 22)

    println("Student Name: ${stu.name}")
    println("Student Age: ${stu.age}")
}

class Student(var name: String, var age: Int)
{
}
```

Primary Constructor - Init Block

```kotlin
class Student(var name: String = “N/A", var age: Int = –1) {
    var stName: String
    var stAge: Int
    init{
        if(name == “N/A") {
            stName = ""
            stAge = 0
        }
        else {
            stName = name.toUpperCase()
            stAge = age
        }
        println("Student Name is initialized as : $stName")
        println("Student Age is initialized as : $stAge")
    }
}
```

Secondary Constructor

```
class Student{
    constructor(name: String, age: Int){
        println("Student Name: $name")
        println("Student Age: $age")
    }
}
```

Inheritance

- It is done by using colon Symbol
- **Note:** By default all classes in Kotlin are final so you have to use the open annotation in the parent class, this tells the compiler that this class can be inherited by other classes.

Inheritance - Function Override

```
open class Animal() {
    open fun sound() {
        println("Animal makes a sound")
    }
}

class Dog: Animal() {
    override fun sound() {
        println("Dog makes a sound of woof")
    }
}
```

Inheritance - Data Override

```kotlin
open class Animal() {
    open var colour: String = "White"
}

class Dog: Animal() {
    override var colour: String = "Black"
    fun sound() {
        println("Dog makes a sound")
    }
}
```

Visibility Modifier

- **Public:** If you do not specify any visibility modifier, public is used by default, which means that your declarations will be visible everywhere;
- **Private:** If you mark a declaration private, it will only be visible inside the file containing the declaration
- **Internal:** If you mark it internal, it is visible everywhere in the same module

Iterators

```
val numbers = listOf("one", "two", "three", “four")

val numbersIterator = numbers.iterator()

while (numbersIterator.hasNext()) {
    println(numbersIterator.next())
}
```

Filters

```
val numbers = listOf("one", "two", "three")

val longerThan3 = numbers.filter
{ it.length > 3 }

println(longerThan3)
```

Filters

```
val numbersMap = mapOf("key1" to 1, "key2" to 2,
"key3" to 3, "key11" to 11)

val filteredMap = numbersMap.filter { (key, value) ->
key.endsWith("1") && value > 10}

println(filteredMap)
```

Lazy Init

```
val lazyValue: String by lazy {
    println(“New Value")
    “Hello"
}

println(lazyValue) // prints New Value then Hello
println(lazyValue) // prints Hello
```

Kotlin Functions - Higher Order

```
calc(30,40,::add) // send add fun as argument
calc(50,10,::sub) // send sub fun as argument

fun add(num1:Int, num2:Int):Int {
    return num1 + num2
}
fun sub(num1:Int, num2:Int):Int{
    return num1 - num2
}
```

Kotlin Functions - Higher Order

```
calc(30,40,::add)
calc(50,10,::sub)

fun calc(x:Int, y:Int, op:(num1:Int, num2:Int)->Int){
    println (op(x,y))
}

fun add(num1:Int, num2:Int):Int {
    return num1 + num2
}
fun sub(num1:Int, num2:Int):Int{
    return num1 - num2
}
```

Kotlin Functions - Higher Order

```
fun calc(x:Int, y:Int, op:(num1:Int, num2:Int)->Int){
    println (op(x,y))
}
```

Accessing Device Storage

Android Device Storage

- Shared Preferences
- Files
- SQLite Database

Shared Preferences

- It is simple way of saving data
- It is used to save preferences (settings, flags,…)
- It uses key-value approach
- Data are saved on the device as XML files

Files

- We use streams to access files for read and write operations
- It is suitable for saving bulk of text data
- It is handles larger data amount compared to Shared Preferences, but it doesn’t support efficient searching and retrieving solution unlike database

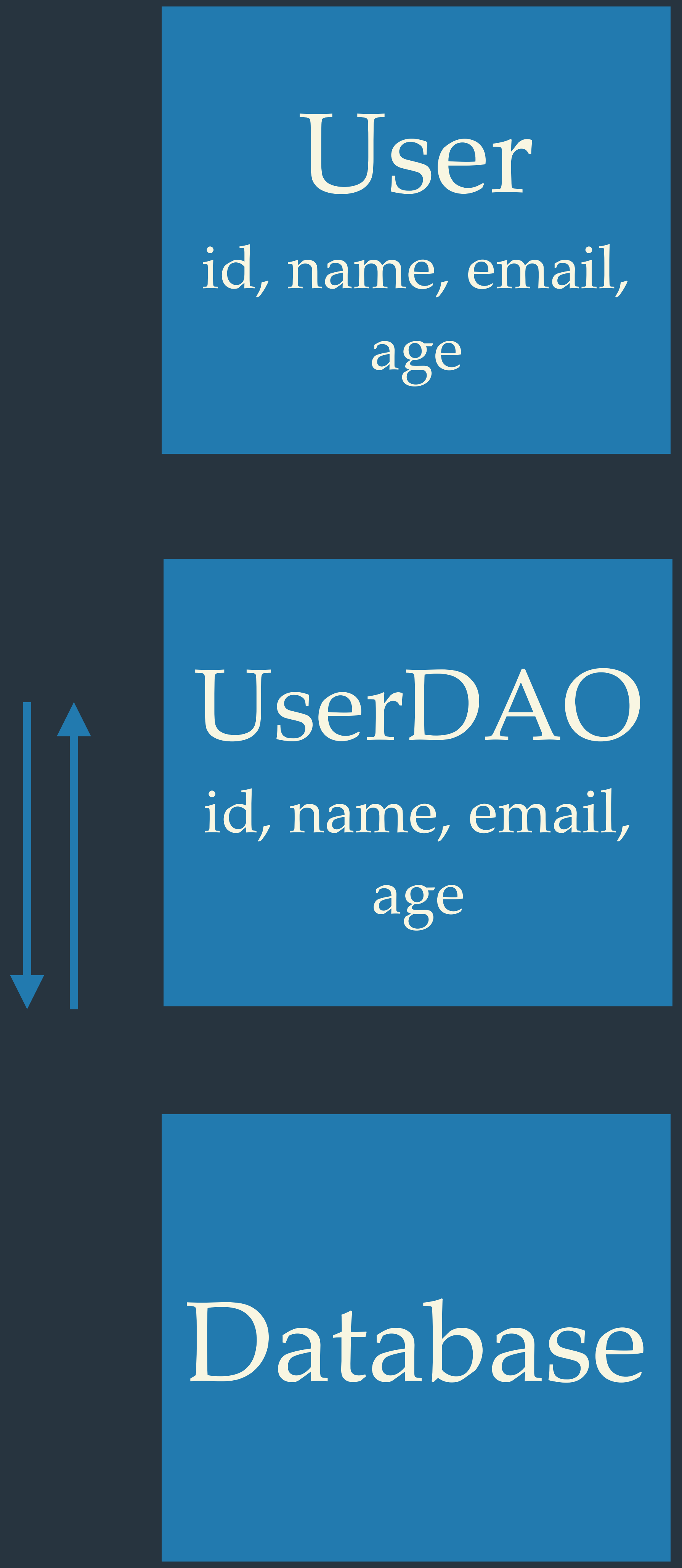
SQLite Database

- It is the best solution for saving structured data and provide search, update and delete capabilities in an efficient way
- It requires a specific setup that takes a little bit longer than files

SQLite vs DBMS

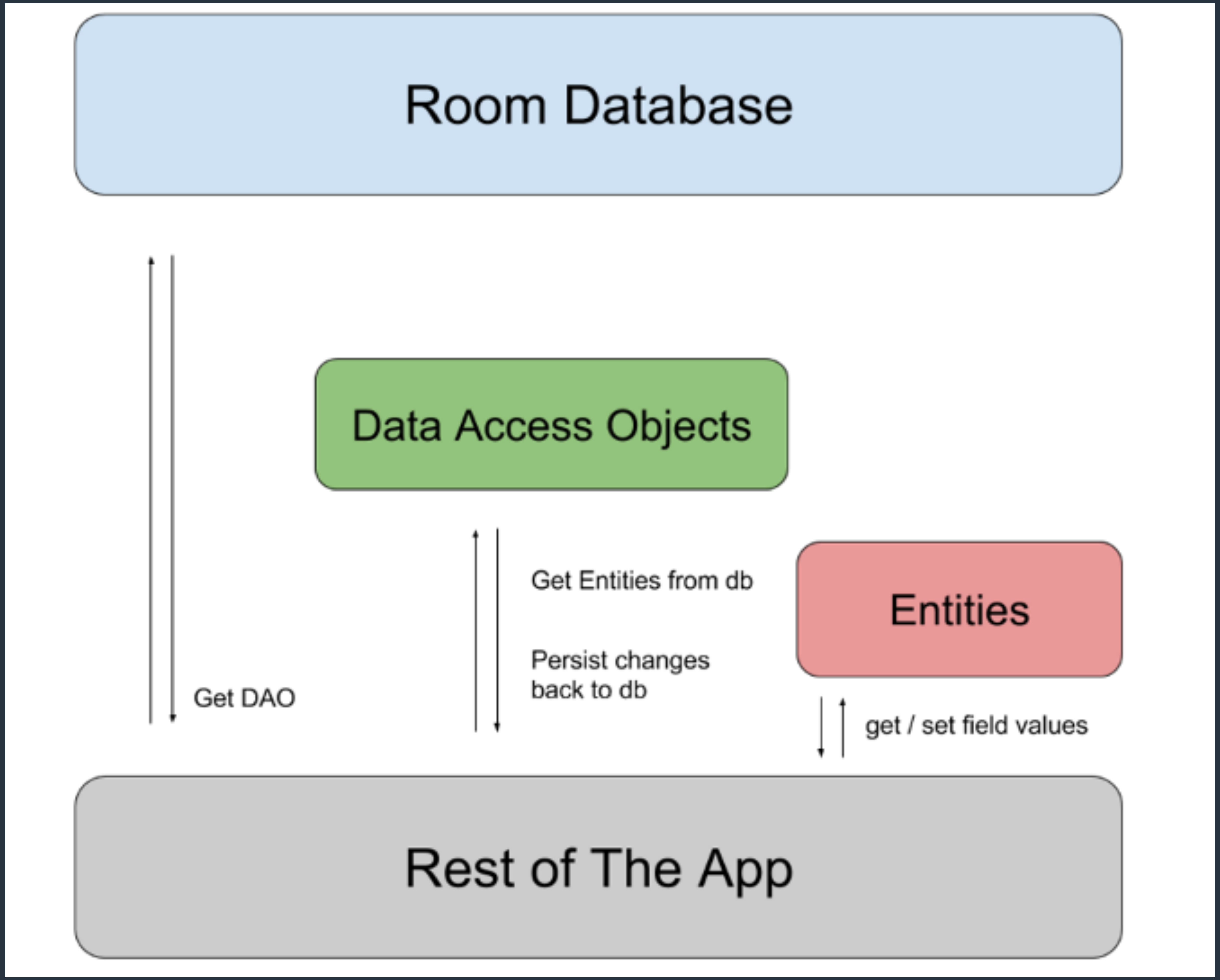
- SQLite is a file
- DBMS (Database Management System - such as Mysql, MS SQL and Oracle) is a standalone application that manages a storage of data
- SQLite can be stored on a mobile device
- DBMS are installed on servers and accessed by mobile applications through web app (Web API)

Room Database



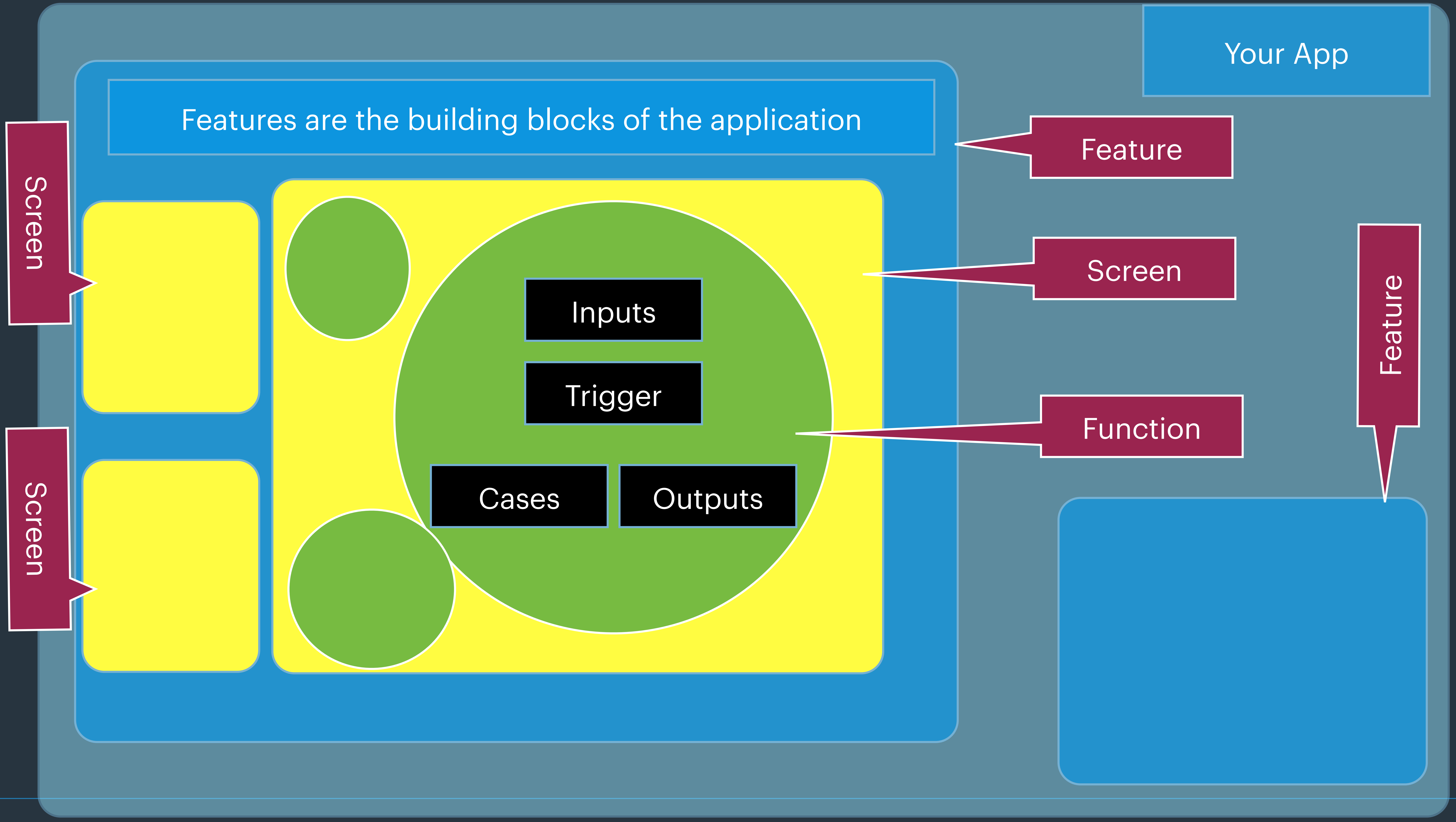
Id	Name	Email	Age
1	Ahmed	ah@mail.com	28
2	Hasan		
3			
4			

Room Database



Project Design and Scoping

Application Overview



Application

Application Summery

Application Description

List of features

Feature 1

Feature 2

Feature 3

Feature 1

About Feature

Feature Description

List of Screens

Screen 1

Screen 2

Screen 3

S11 [screen 1 feature1]

About Screen

List of Functions

Function 1

Function 2

Function 3

S11 [screen 1 feature1]

Screen Design

F111 [function 1 screen 1 feature 1]

About Function

Inputs

List of cases

Case 1- Output

Case 2 - Output