Взаимодействие с кодом Kotlin и состояние компонентов

Взаимодействие с кодом Kotlin

Jetpack Compose предполагает применение декларативного стиля при определении визуального интерфейса и работы с ним. В этом отношении иногда возникает вопрос как определять какие-то свои собственные данные в виде переменных или действия в виде функций и использовать эти переменные и функции в компонентах для создания графического интерфейса. Рассмотрим некоторые случаи.

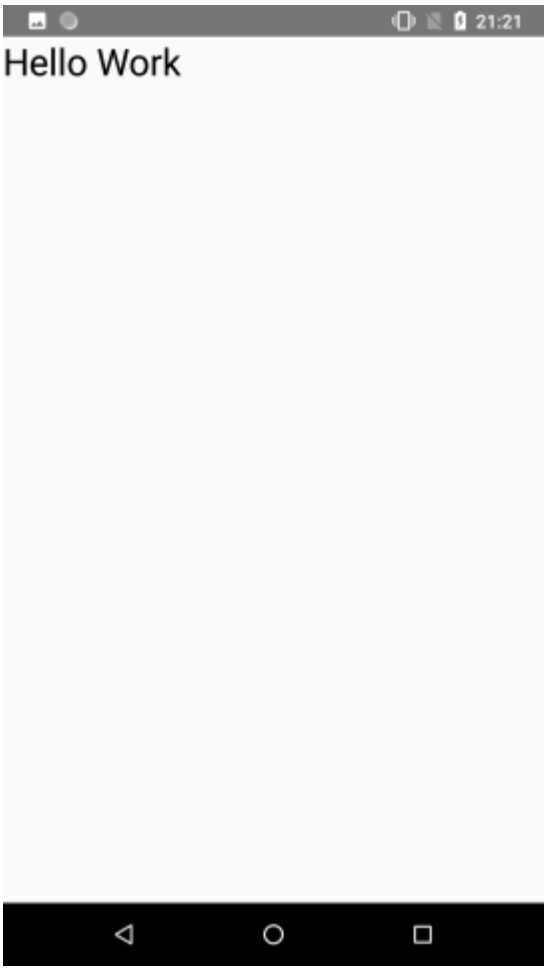
Использование переменных

Как и в общем случае переменные можно определять на уровне класса (например, на уровне класса MainActivity), либо на уровне отдельного метода/функции. Однако не всегда очевидно, что, в ом чсле можно определять переменные на уровне функций, которые принимают компоненты в качестве параметров. Например, используем пру переменных:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.material.Text
import androidx.compose.ui.unit.sp
import androidx.compose.foundation.cl
import androidx.compose.ui.unit.TextUnit

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val message = "Hello Metanit.com"
            val size: TextUnit = 28.sp
            Text(
                text = message,
                fontSize = size
            )
        }
    }
}
```



Второй параметр функции setContent() представляет функцию типа @Composable () -> Unit, то есть функцию, которая принимает в качестве параметра компонент или объект с аннотацией @Composable. Kotlin позволяет определить эту функцию после названия функции setContent и внутрь нее передать нуюный компонент:

```
setContent {

    // компонент
}
```

Но также в рамках этой функции мы также можем использовать все определения и инструкции, которые поддерживает язык Kotlin, в том числе определять переменные, как в примере выше. А затем эти переменные передать в компонент в качестве значений для его параметров.

Это относится ко всем функциям, которые принимают компоненты в качестве параметров. Причем функция каждого компонента определяет свою область видимости, в рамках которой доступны ее переменные. Вложенные компоненты имеют доступ к переменным, определенным в родительских компонентах. Например:

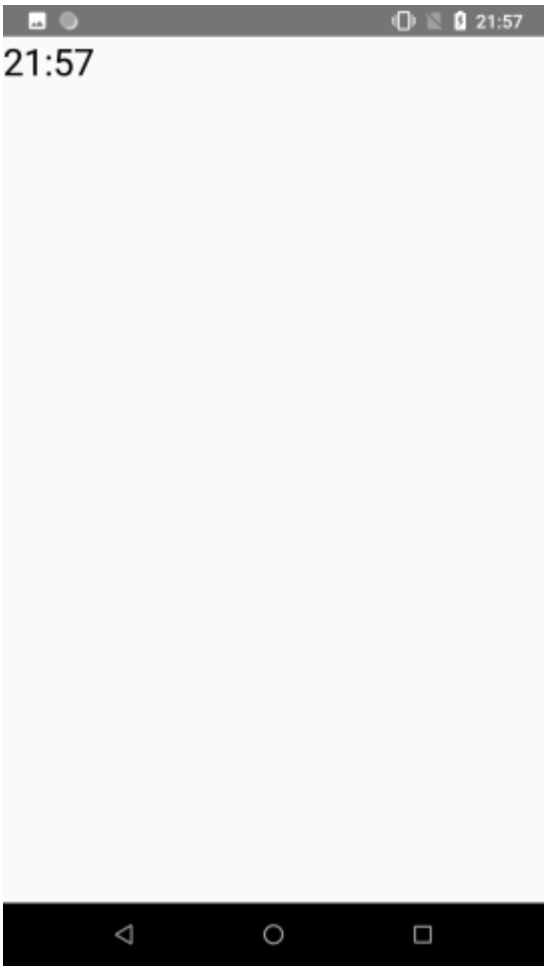
```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
```

```
import androidx.activity.compose.setContent
import androidx.compose.material.Text
import androidx.compose.ui.unit.sp
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val setContentMessage = "Hello setContent"
            Column{
                val columnMessage = "Hello Column"
                Row{
                    Text(text = setContentMessage, fontSize = 28.sp)
                }
                Row{
                    Text(text = columnMessage, fontSize = 28.sp)
                }
                Row{
                    val rowMessage = "Hello Row 1"
                    Text(text = rowMessage, fontSize = 28.sp)
                }
                Row{
                    val rowMessage = "Hello Row 2"
                    Text(text = rowMessage, fontSize = 28.sp)
                }

            }
        }
    }
}
```



Здесь иерархию компонентов можно представить следующим образом: setContent -> Column -> Row -> Text. На уровне функции setContent() определяется переменная setContentMessage, которая видна во всех вложенных компонентах. На уровне комонента Column определяется переменная columnMessage, которая видна на уровне компонентов Row.

Два последних компонента Row определяют переменную rowMessage, которые доступны во вложенных компонентах Text, но недоступны вне соответствующих компонентов Row.

Условные конструкции

Подобным образом можно использовать внутри компонентов управляющие конструкции, например, условные конструкции:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Column
import androidx.compose.material.Text
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
```

```
        super.onCreate(savedInstanceState)
        setContent {
            val hour = 19
            if(hour < 18) {
                Text(text = "Добрый день", fontSize = 28.sp)
            }
            else{
                Column{
                    Text(text = "Добрый", fontSize = 23.sp)
                    Text(text = "вечер", fontSize = 23.sp)
                }
            }
        }
    }
}
```

Здесь в зависимости от переменной hour в качестве внутренного содержимого установливается определенный компонент.

Циклы

Если необходимо создать ряд отднотиных компонентов, то удобно определить различающиеся данные в виде списка, а в цикле по этому списку создать набор компонентов:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Column
import androidx.compose.material.Text
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val langs = listOf("Kotlin", "Java", "JavaScript", "Scala")
            Column{
                for(lang in langs){
                    Text(text = lang, fontSize = 28.sp)
                }
            }
        }
    }
}
```

Здесь определяем данные в виде списке langs. В столбце пробегаемся по этому списку и создаем для каждого свой компонент Text

Причем в данном случае мы также могли бы применить и встроенную функцию forEach() списка, которая позволяет перебрать все объекты:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Column
import androidx.compose.material.Text
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val langs = listOf("Kotlin", "Java", "JavaScript", "Scala")
            Column{
                langs.forEach{lang ->
                        Text(text = lang, fontSize = 28.sp)
                }
            }
        }
    }
}
```

Использование функций

Определим функцию и используем ее в комонентах:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Column
import androidx.compose.material.Text
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            Column{
                Text(text = createMessage(5), fontSize = 28.sp)
                Text(text = createMessage(15), fontSize = 28.sp)
                Text(text = createMessage(20), fontSize = 28.sp)
            }
        }
    }

    fun createMessage(hour: Int): String =
        if(hour >18){ "Добрый вечер"}
        else if(hour >10){ "Добрый день"}
        else { "Доброе утро"}
}
```

В данном случае функция createMessage() в зависимости от переданного в нее значения возвращает то или иное сообщение, которое затем можно применить для установки текста в компоненте Text.

Подобным образом можно было бы определить функцию в виде анонимной функции или лямбда-выражения:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Column
import androidx.compose.material.Text
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
```

```
            Column{
                val createMessage = {hour: Int ->
                    if (hour > 18) {
                        "Добрый вечер"
                    } else if (hour > 10) {
                        "Добрый день"
                    } else {
                        "Доброе утро"
                    }
                }

                Text(text = createMessage(5), fontSize = 28.sp)
                Text(text = createMessage(15), fontSize = 28.sp)
                Text(text = createMessage(2), fontSize = 28.sp)
            }
        }
    }
}
```

Введение в состояние компонентов

Поскольку Jetpack Compose применяет декларативный подход, то единственный способ обновить визуальный компонент представляет повторный вызов функции этого компонента, в которую передаются новые значения. И чтобы упростить обновление компонентов и вообще визуального интерфейса Jetpack Compose предоставляет концепцию состояние или state. Состояние представляет некоторое значение, которое хранится в приложении и которое в процессе его работы может изменяться.

MutableState

Измененяемое состояние в Jetpack Compose представлено объектом интерфейса MutableState, который имеет следующее определение:

```
interface MutableState<T> : State<T> {
    override var value: T
}
```

Объект этого интерфейса хранит значение, которое будет изменяться, в виде переменной value. Причем данное значение может представлять любой тип.

Объект MutableState интегрирован со средой выполнения Compose и позволяет отслеживать изменения хранимого в нем значения. Любые изменения этого значения приведут к обоновлению (или рекомпозиции) любого компонента, который использует данное значение.

Для создания объекта MutableState Jetpack Compose предоставляет функцию mutableStateOf():

```
val mutableState = mutableStateOf(значение)
```

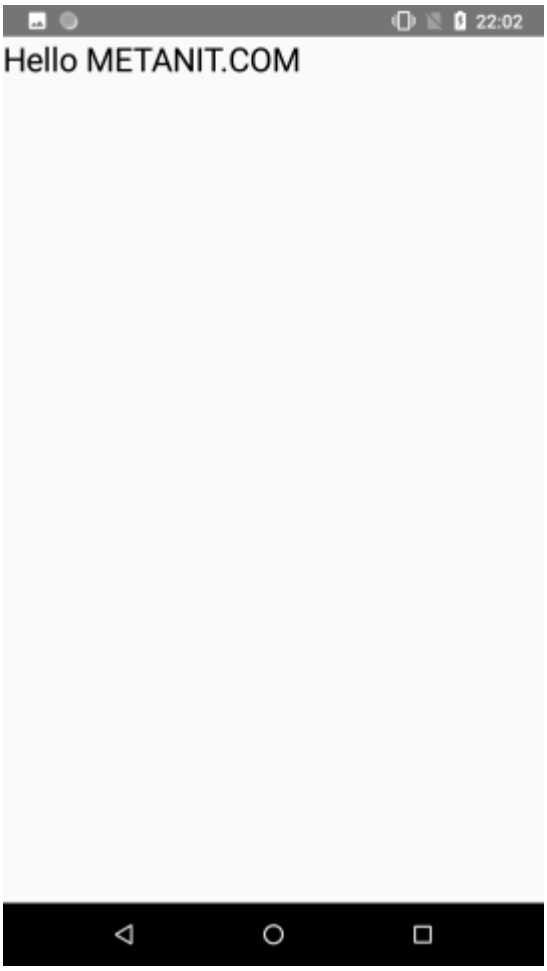
В функцию mutableStateOf() в качестве параметра передается отслеживаемое значение. Возвращаемый из функции объект MutableState позволяет связать отслеживаемое значение с компонентом.

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.material.Text
import androidx.compose.runtime.mutableStateOf
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val mutableState = mutableStateOf("Hello Jetpack")
            Text(text = mutableState.value, fontSize = 25.sp)
        }
    }
}
```

Здесь в функцию mutableStateOf() передается строка, то есть объект String, поэтому возвращаемый функцией объект будет представлять тип MutableState. С помощью свойства value можно получить отслеживаемое значение: mutableState.value. В данном случае это значение передается в компонент Text для отображения.



Обращаясь к свойству value также можно изменить отслеживаемое значение:

```
val mutableState = mutableStateOf("Hello Jetpack")
mutableState.value = "Hello World"
```

Теперь посмотрим, как динамически изменять это состояние. Компонент Text позволяет применить модификатор clickable, который обрабатывает нажатия на компонент. Применим этот модификатор для динамического изменения текста в компоненте Text по нажатию на него:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.material.Text
import androidx.compose.runtime.mutableStateOf
import androidx.compose.ui.unit.sp
import androidx.compose.foundation.clickable
import androidx.compose.ui.Modifier

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
```

```
            val message = mutableStateOf("Hello Jetpack")

            Text(
                text = message.value,
                fontSize = 28.sp,
                modifier = Modifier.clickable( onClick = { message.value = "Hello
Work!" })
            )
        }
    }
}
```

Здесь в модификаторе clickable параметру onClick передается обаботчик нажатия, который изменяет отслеживаемое значение. То есть мы ожидаем, что по нажатию на текст он изменится с "Hello Jetpack" на "Hello Work!". Однако, если мы запустим проект, то увидим, что понажатию ничего не меняется. Чего-то не хватает - а именно функции - remember

Функция remember

Для сохранения некоторого объекта в памяти применяется специальная функция remember. Она может хранить как изменяемые (mutable), так и неизменяемые (immutable) объекты. Причем данные объекты сохраняются во время начального построения интерфейса (то что называется initial composition) и продолжают храняться во время обновлений интерфейса (recomposition).

В сочетании с функцией mutableStateOf есть три способа определения состояния:

```
val mutableState = remember { mutableStateOf(значение) }
var value by remember { mutableStateOf(значение) }
val (value, setValue) = remember { mutableStateOf(значение) }
```

Во втором случае применятся делегат by, для работы с которым необходимо импортировать следующие функции:

```
import androidx.compose.runtime.getValue
import androidx.compose.runtime.setValue
```

В третьем случае возвращается собственно отслеживаемое значение (value) и функция обработки изменения этого значения (setValue)

Изменим пример выше с изменением текста, применив функцию remember:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
```

```
import androidx.compose.material.Text
import androidx.compose.runtime.mutableStateOf
import androidx.compose.ui.unit.sp
import androidx.compose.foundation.clickable
import androidx.compose.runtime.remember
import androidx.compose.ui.Modifier

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val message = remember{mutableStateOf("Hello Jetpack")}

            Text(
                text = message.value,
                fontSize = 28.sp,
                modifier = Modifier.clickable( onClick = { message.value = "Hello
Work!" })
            )
        }
    }
}
```

Теперь по нажатию на текст он будет изменяться:

Для примера рассмотрим применение третьего варианта:

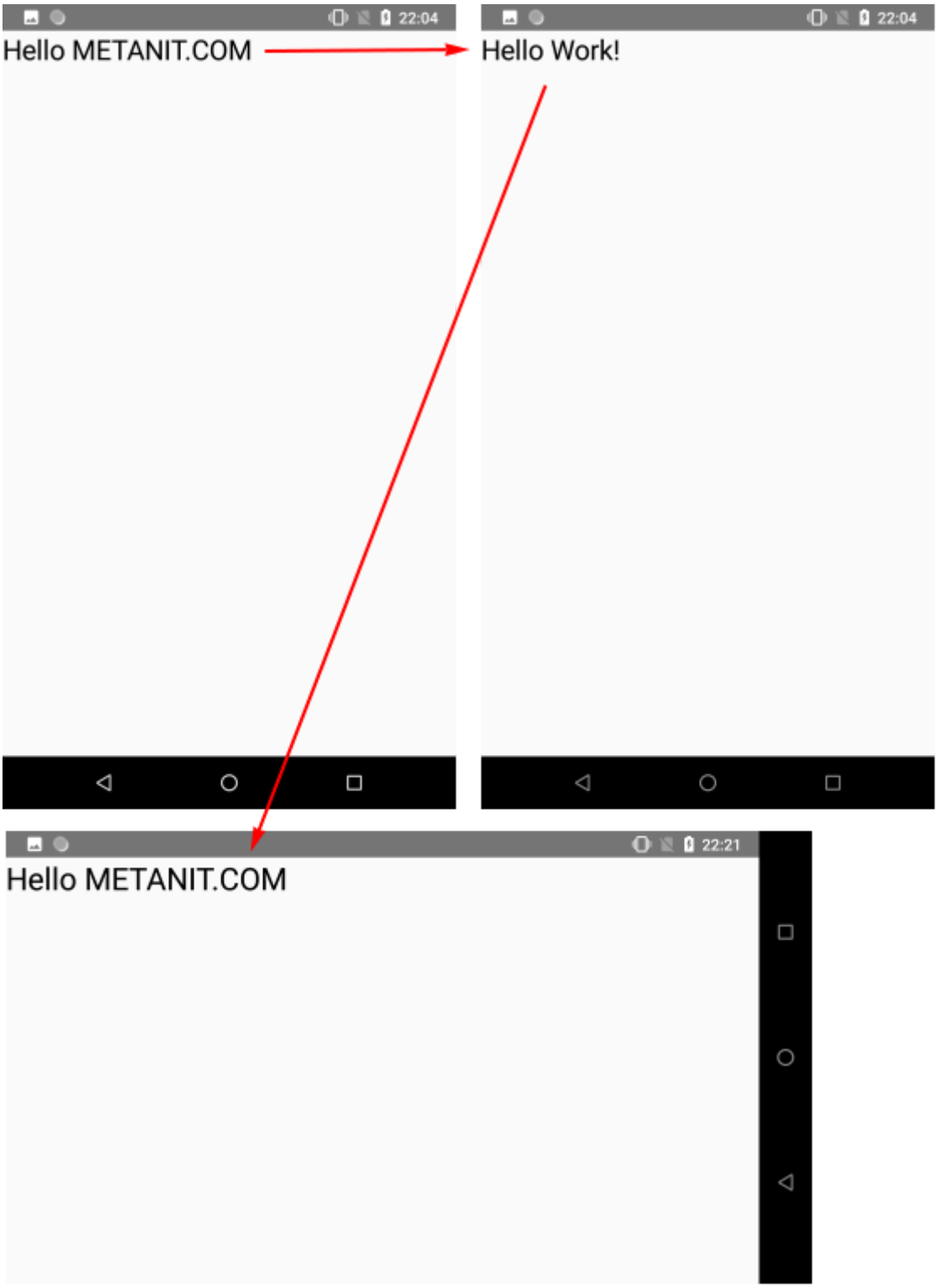
```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.material.Text
import androidx.compose.runtime.mutableStateOf
import androidx.compose.ui.unit.sp
import androidx.compose.foundation.clickable
import androidx.compose.runtime.remember
import androidx.compose.ui.Modifier

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val (value, setValue) = remember{mutableStateOf("Hello Jetpack")}

            Text(
                text = value,
                fontSize = 28.sp,
                modifier = Modifier.clickable( onClick = { setValue("Hello
Metanit.com")})
            )
        }
    }
}
```

Теперь мы берем из MutableState отдельно само отслеживаемое значение в виде переменной value и отдельно обаботчик обновления этого значения в виде функции setValue. Поскольку отслеживаемое значение представляет объект String, то функция setValue() также принимает объект String. И в данном случае мы передаем подобный объект: setValue("Hello Metanit.com"). А обаботчик сам установит данное значение.


22:04
Hello METANIT.COM
22:04
Hello Work!
22:21
Hello METANIT.COM