

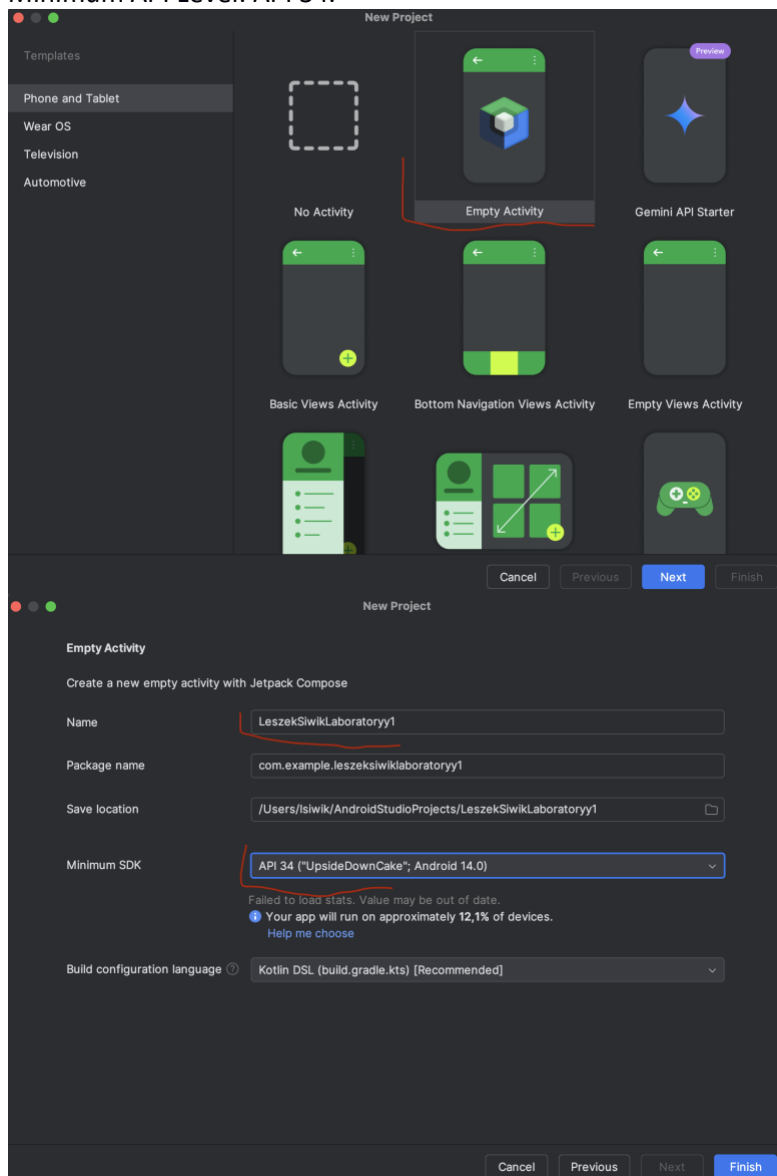
Android App Development – Laboratory 1A

Prerequisites

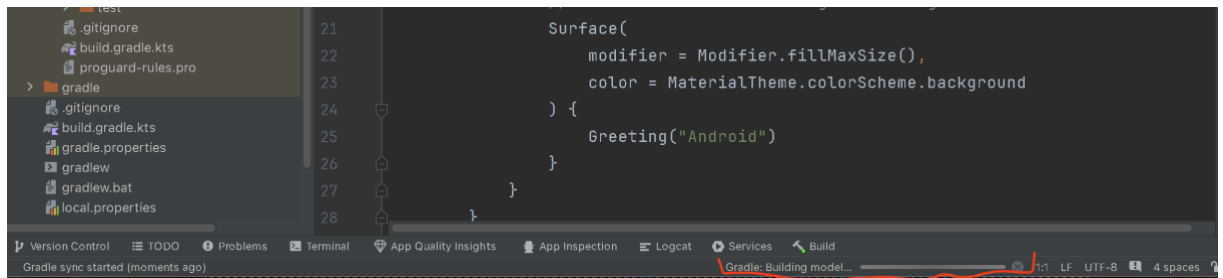
- If you are using your own computers but have not installed and/or configured the Android Studio yet, pls navigate to <https://developer.android.com/studio> to download it and follow the instructions <https://developer.android.com/studio/install> <https://developer.android.com/studio/intro/studio-config> to get it up and running

Let's start

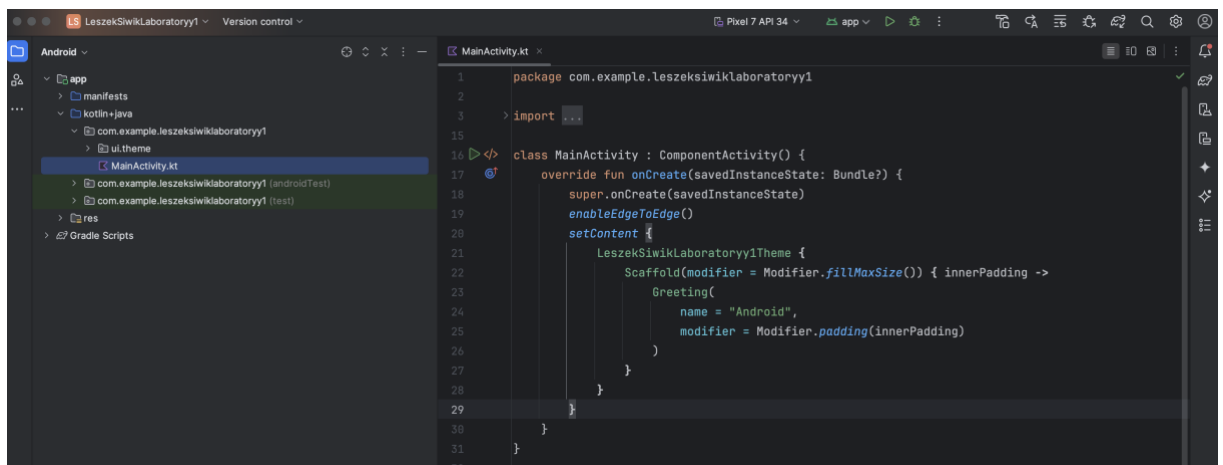
- Launch Android Studio.
- Create a new project by selecting: File -> New Project -> Phone and Tablet.
- Choose the "Empty Activity" template.
- Name the project "FirstNameLastNameLaboratory1".
- Minimum API Level: API 34.



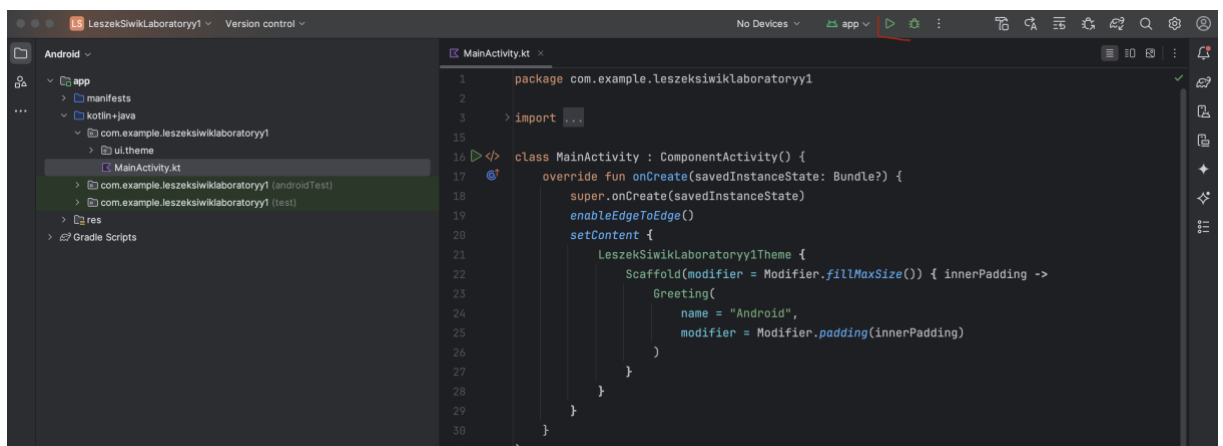
- We need to wait (which unfortunately may take a moment 😊) until Gradle completes all tasks related to creating the project



- But finally, we should get the environment up and ready for work

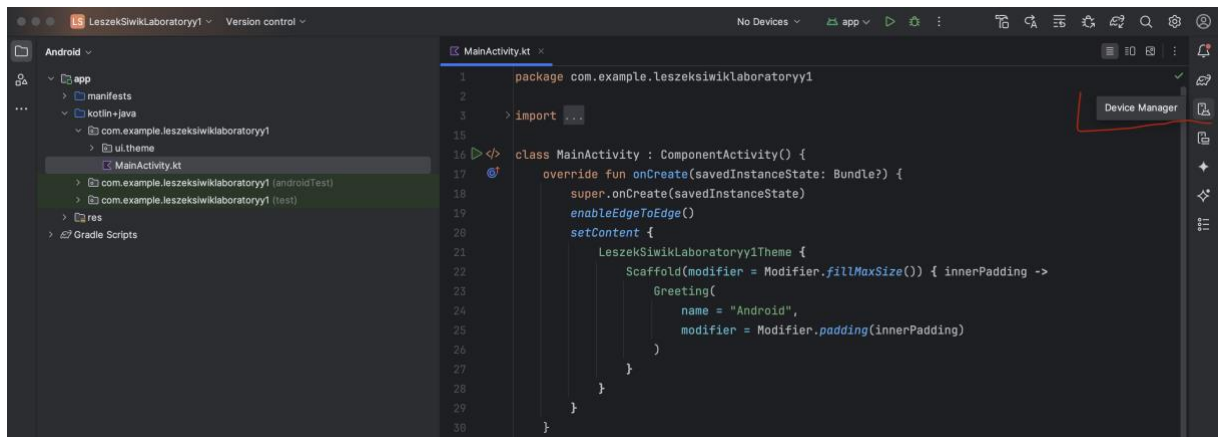


- Let's start by test-running the project/application on the emulator. We do this by clicking the "Run" button (green arrow) in the upper part of the Studio.

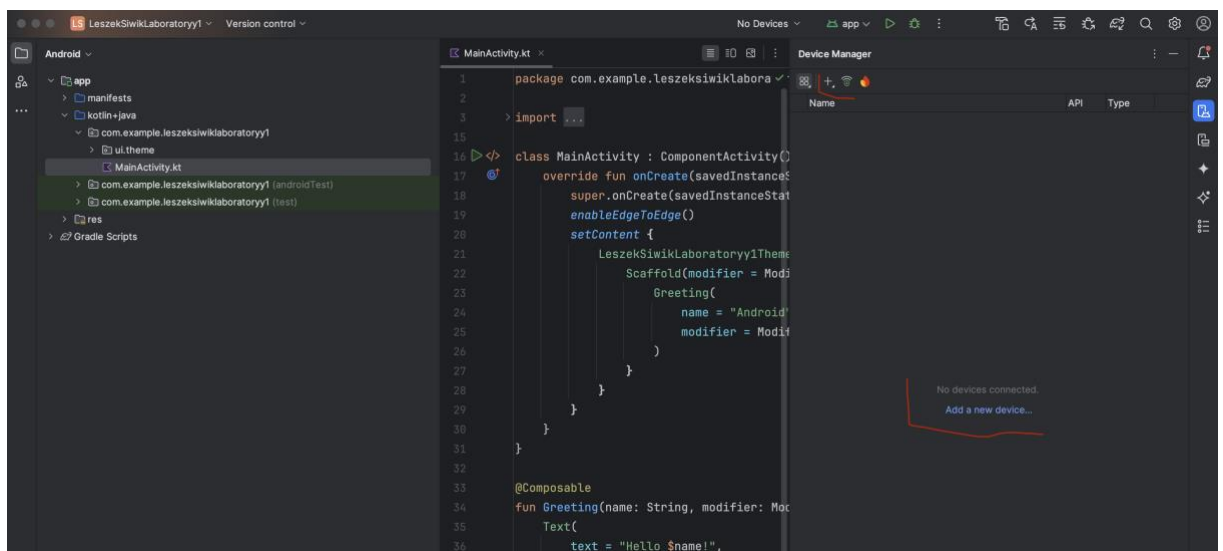


- Or by using the appropriate shortcuts: macOS Control + R, Windows and Linux: Shift + F10.
- It's important to specify the device, physical or - more probably the virtual one - i.e. the emulator on which the application will be launched.
- We specify it in the dropdown menu on the left-hand side of the run-project button mentioned above.
- It may happen that we haven't defined any virtual devices yet. In such a case, we need to create one. We do it as follows.

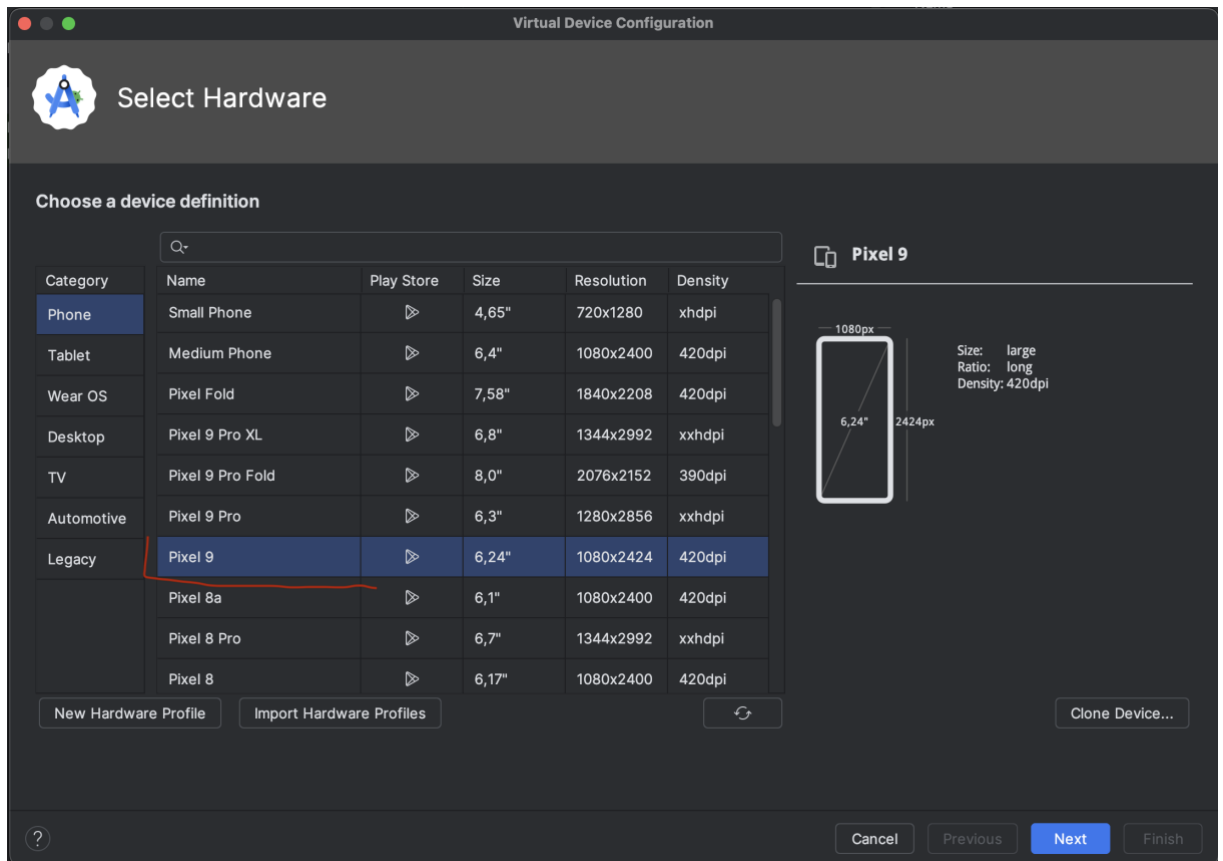
- We click on the Device Manager button.



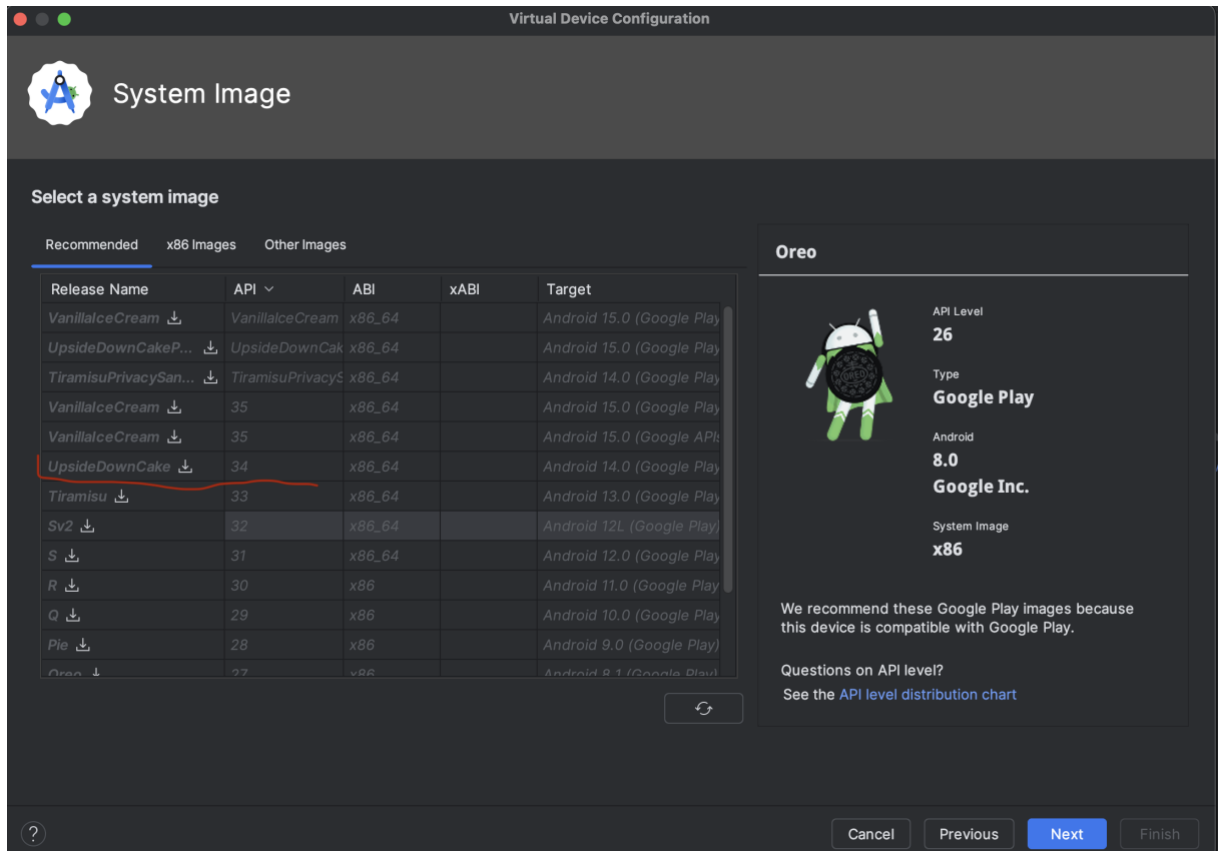
- We click on the Add / Create Virtual Device button.



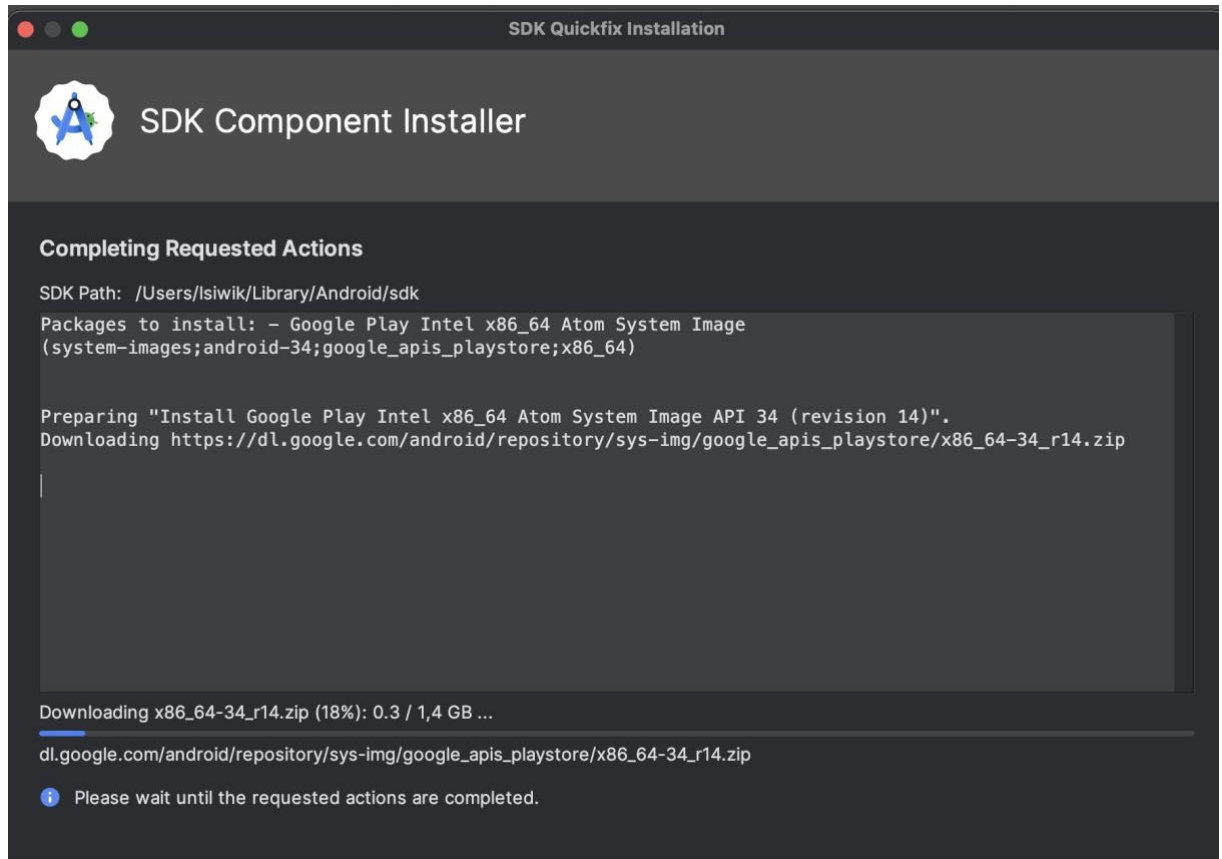
- On the device selection screen, let's choose e.g. Pixel 9.



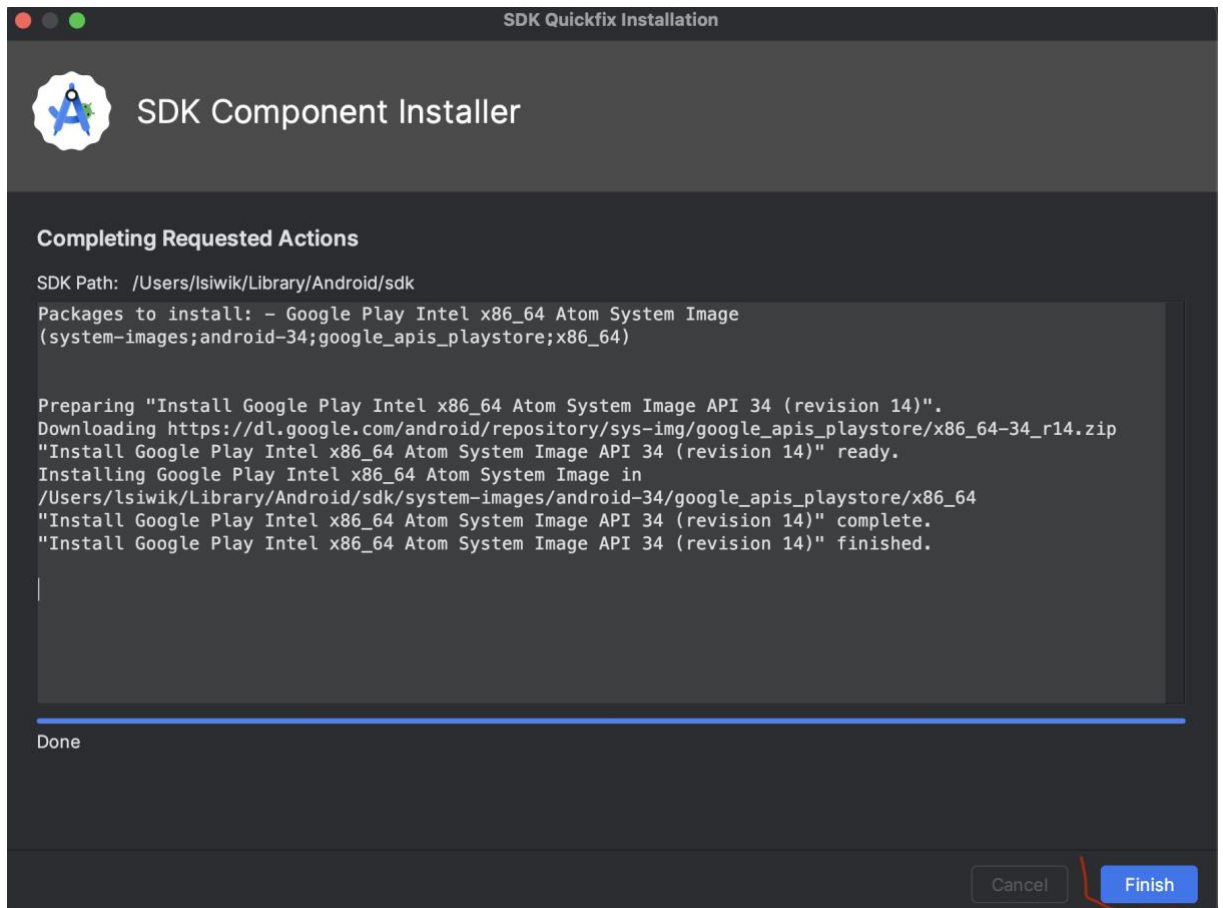
- On the next screen (System Image), let's choose UpsideDownCake (API Level 34).



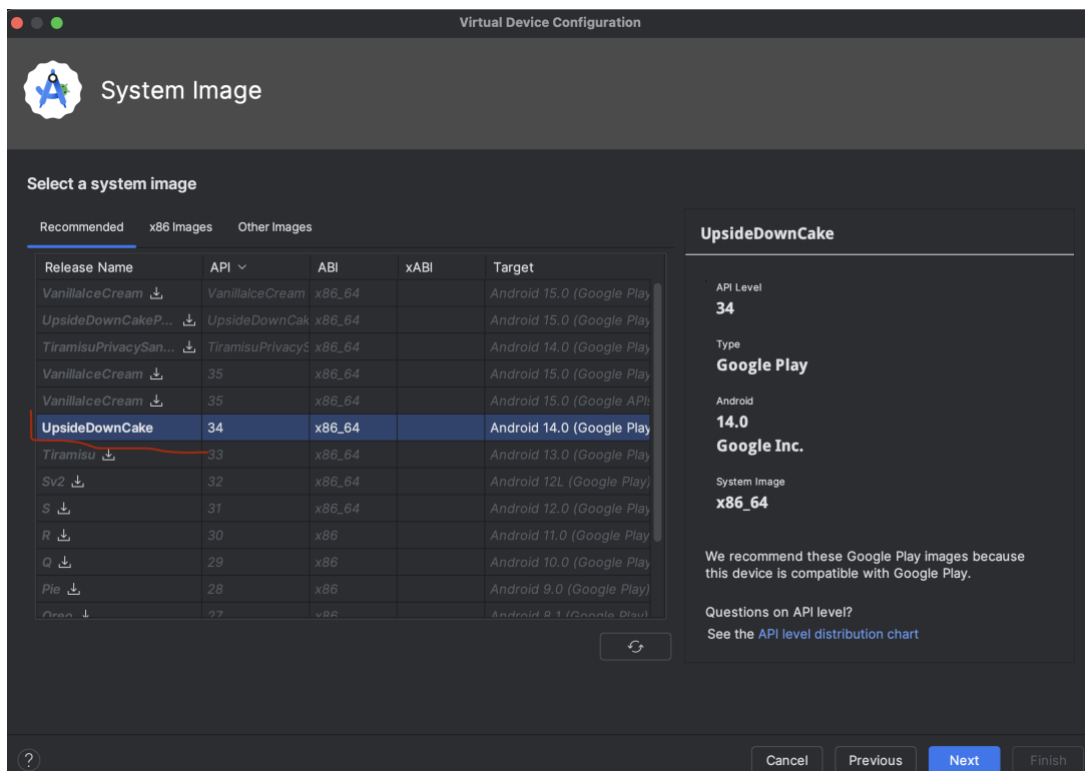
- If there's a download arrow next to the Image name, it means that we haven't downloaded/added it to our environment yet, so we need to do that.
- If this is the case, we click on the download arrow.
- We wait for the installer to complete all its tasks (which may take a moment 😊).



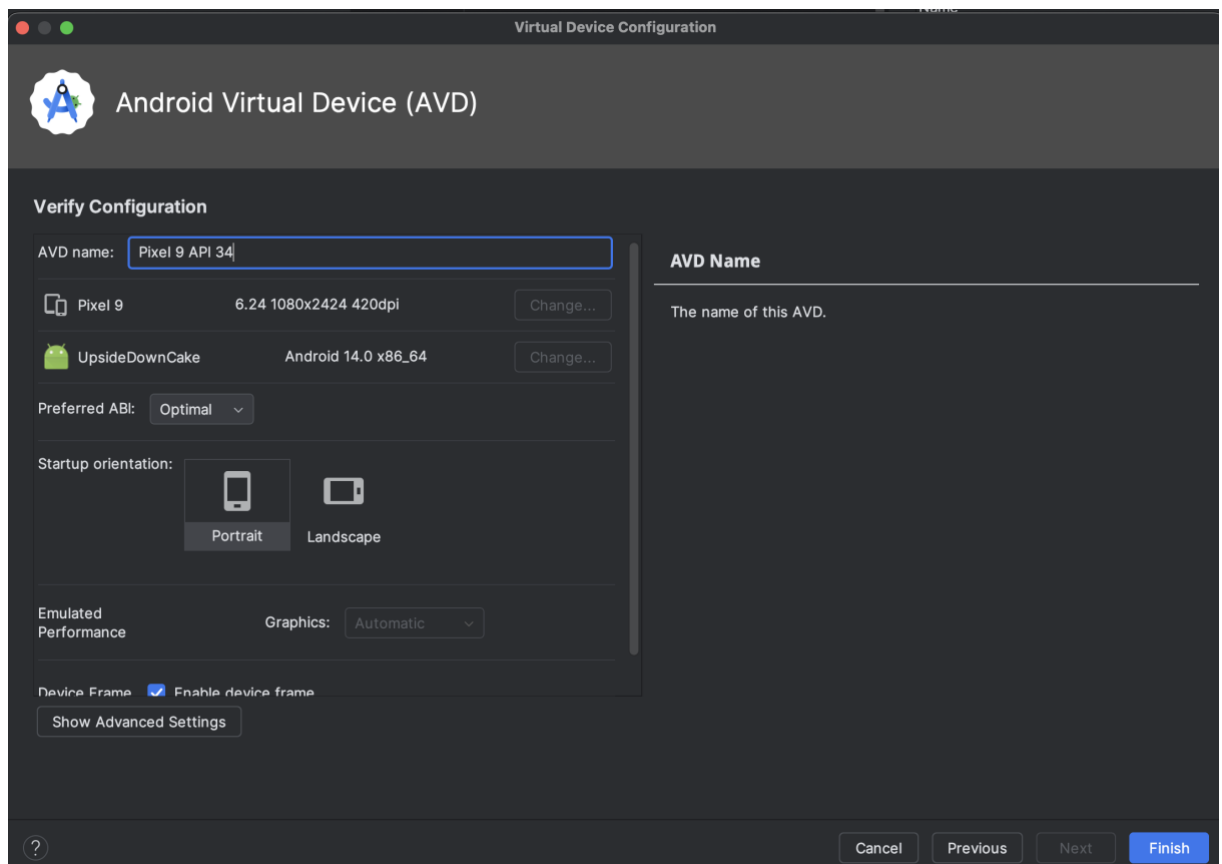
- And, once it's done, we click on Finish, and we'll return to the system image selection screen.



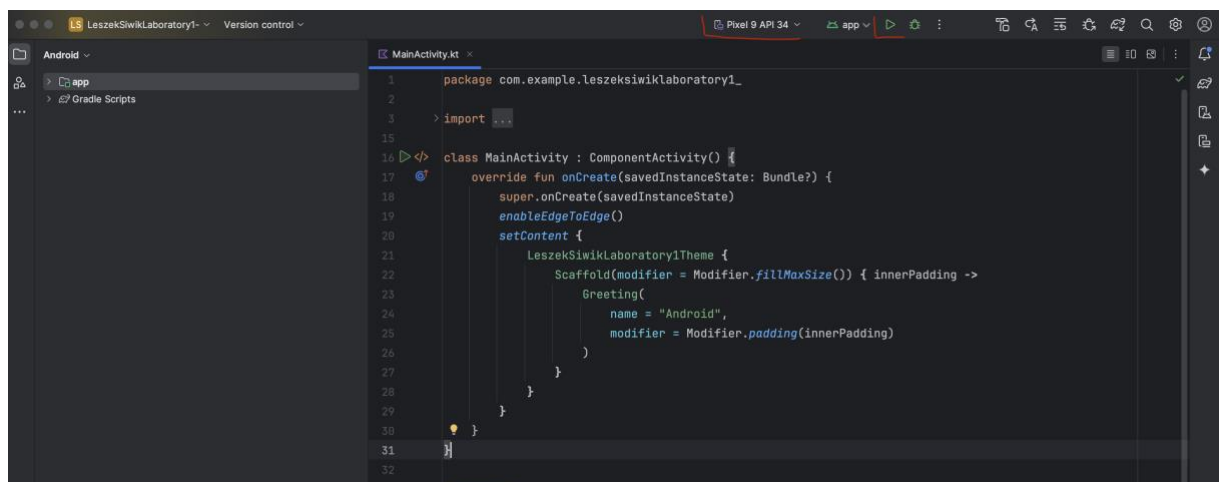
- This time, the image we're interested in should not have a download arrow (just like in the example below), indicating that it is available in the environment and that we can use it.



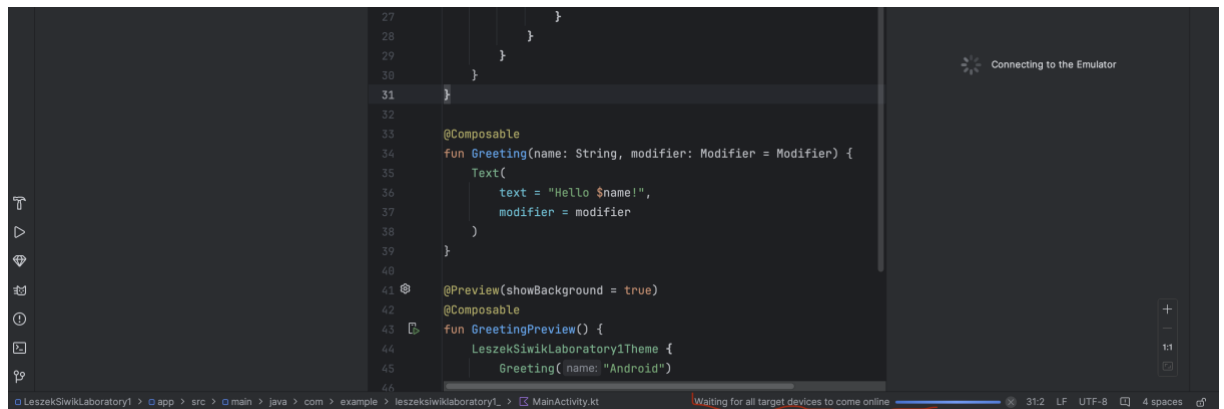
- So, we click Next. On the next screen, we can further customize our virtual device (name, whether it should be launched in landscape or portrait mode by default, etc.).



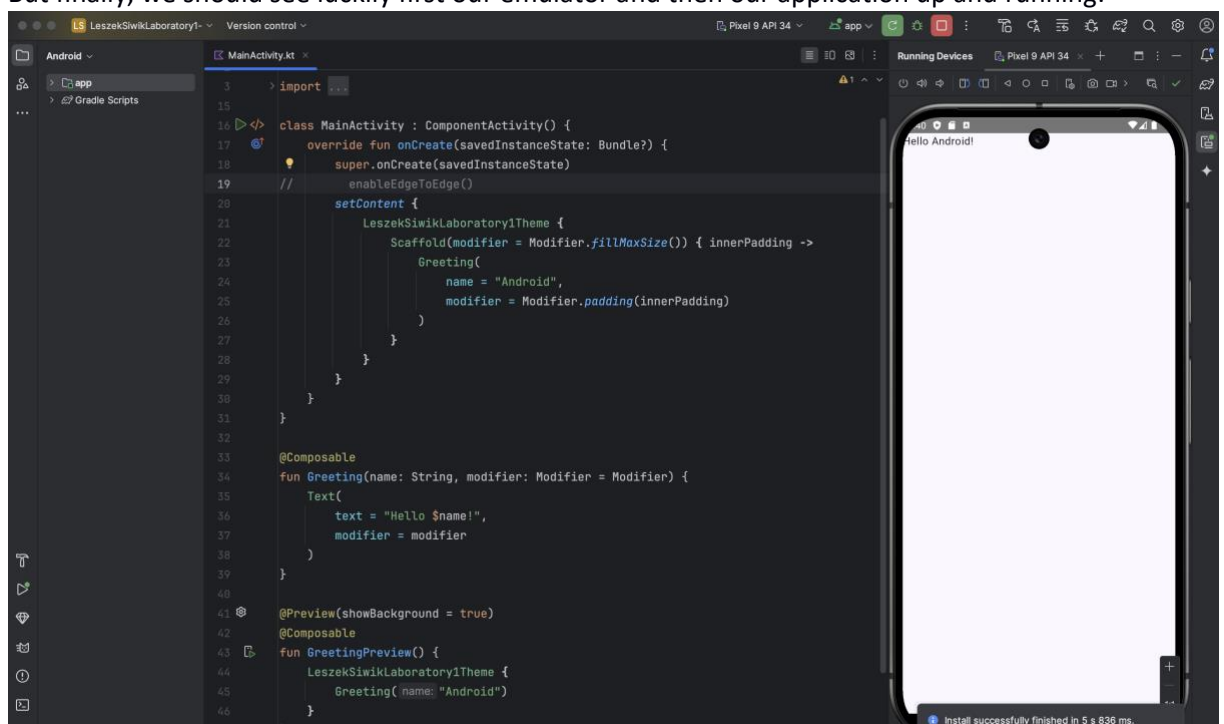
- But there is nothing obligatory that we need to change, so we can keep all the options with default settings and click Finish.
- When everything goes well here, our emulator should be visible/available in the dropdown menu at the top of Android Studio on the left of the run-project button



- So, let's click the green arrow to launch our application (what, while doing it for the first time may take a while)



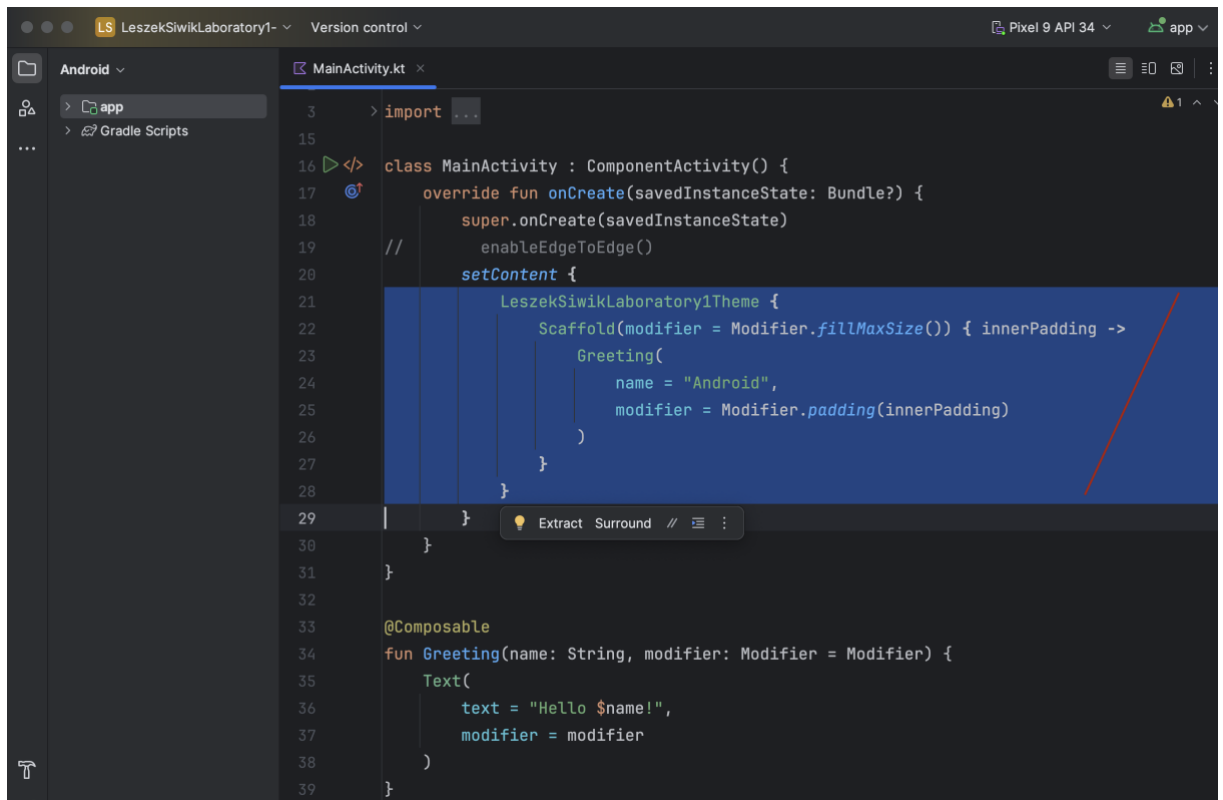
- But finally, we should see luckily first our emulator and then our application up and running:



- If you faced any problems/errors at this stage, you need to fix / resolve them since without that we won't be able to move on.

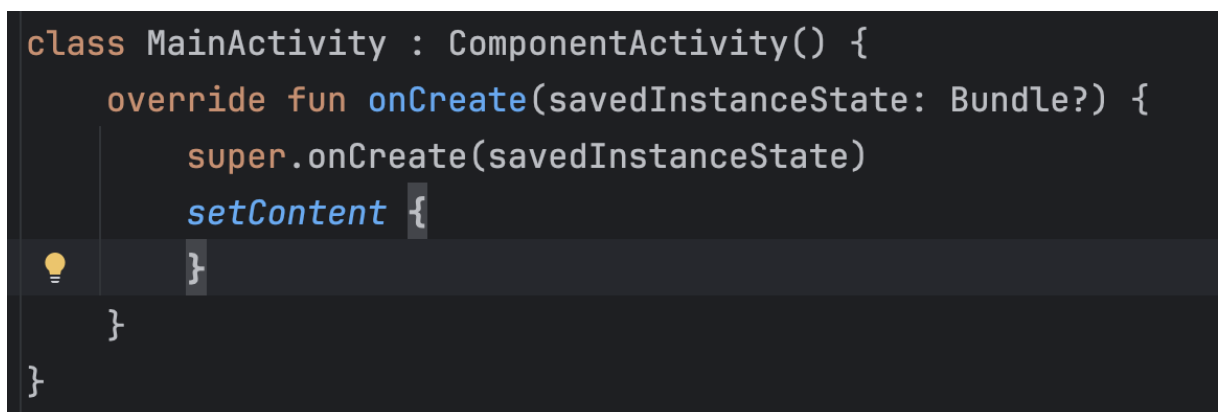
Let's start working on our first Android app

- So, let's work with our application. Let's go to onCreate function of the MainActivity class (it is presented by default in the main part of the Studio after creating the project).



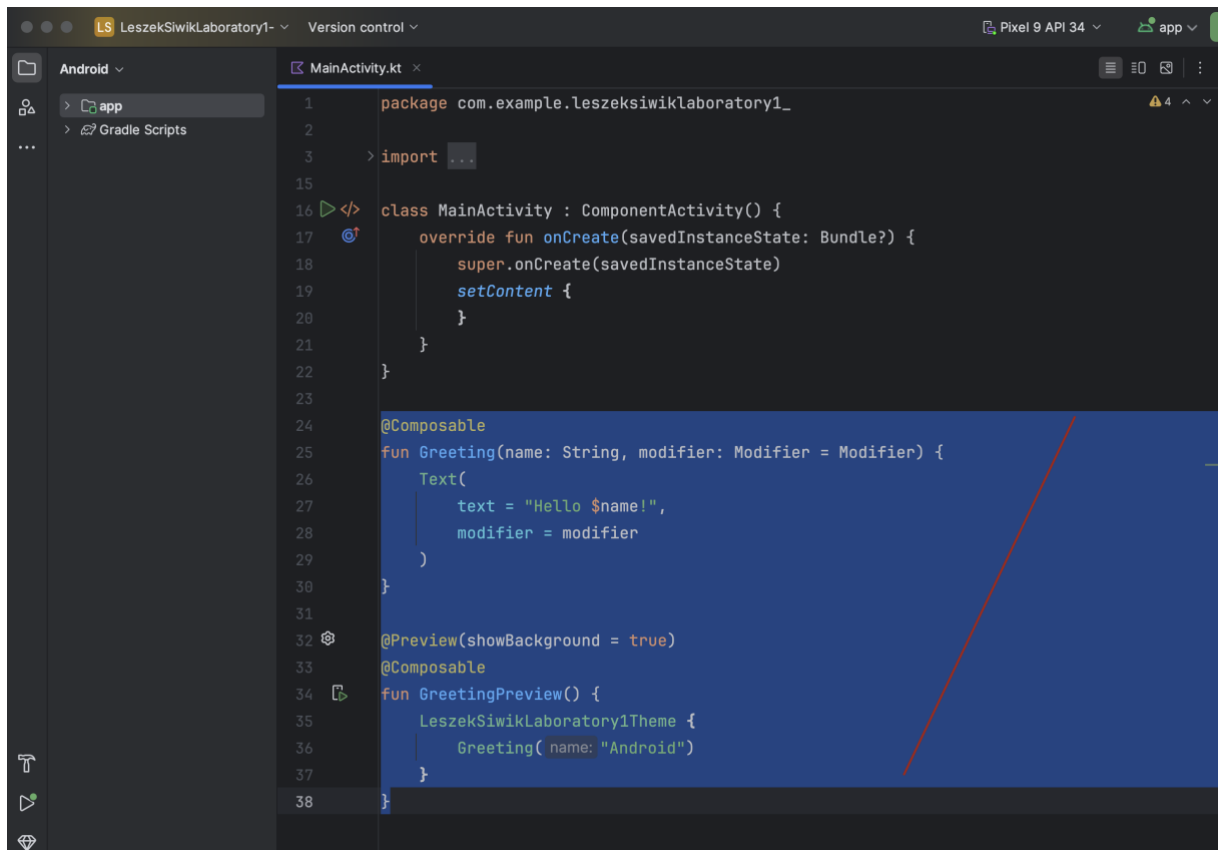
```
3  > import ...
15
16  class MainActivity : ComponentActivity() {
17      override fun onCreate(savedInstanceState: Bundle?) {
18          super.onCreate(savedInstanceState)
19          // enableEdgeToEdge()
20          setContent {
21              LeszekSiwikLaboratory1Theme {
22                  Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->
23                      Greeting(
24                          name = "Android",
25                          modifier = Modifier.padding(innerPadding)
26                      )
27                  }
28              }
29          }
30      }
31  }
32
33  @Composable
34  fun Greeting(name: String, modifier: Modifier = Modifier) {
35      Text(
36          text = "Hello $name!",
37          modifier = modifier
38      )
39  }
```

-
- Let's remove the content of the setContent function (I mean removing everything I highlighted in the previous screen). So currently, it should look like this:



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

-
- Additionally, let's remove the functions Greeting and GreetingPreview (i.e everything I highlighted in the next screen):

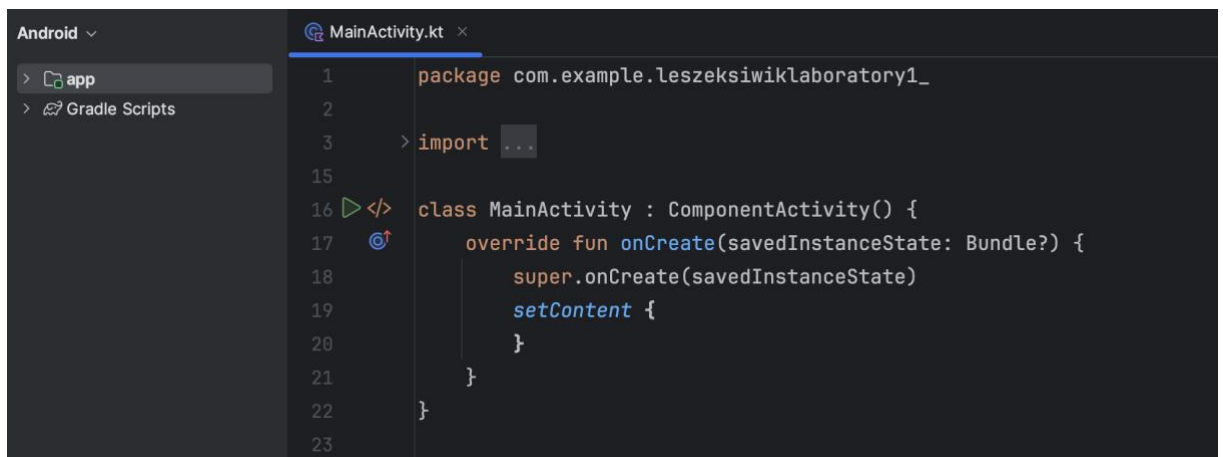


The screenshot shows an IDE window for a project named 'LeszekSiwikLaboratory1'. The file 'MainActivity.kt' is open, showing the following code:

```
1 package com.example.leszeksiwiklaboratory1_
2
3 > import ...
4
15
16 class MainActivity : ComponentActivity() {
17     override fun onCreate(savedInstanceState: Bundle?) {
18         super.onCreate(savedInstanceState)
19         setContent {
20             // Composable content goes here
21         }
22     }
23
24 @Composable
25 fun Greeting(name: String, modifier: Modifier = Modifier) {
26     Text(
27         text = "Hello $name!",
28         modifier = modifier
29     )
30 }
31
32 @Preview(showBackground = true)
33 @Composable
34 fun GreetingPreview() {
35     LeszekSiwikLaboratory1Theme {
36         Greeting(name = "Android")
37     }
38 }
```

The code includes a package declaration, an import statement, a class MainActivity extending ComponentActivity, and two composable functions: Greeting and GreetingPreview. The GreetingPreview function uses the LeszekSiwikLaboratory1Theme and the Greeting function to display a preview of the greeting text.

- Finally, in the MainActivity file, we should have at this stage only the following:

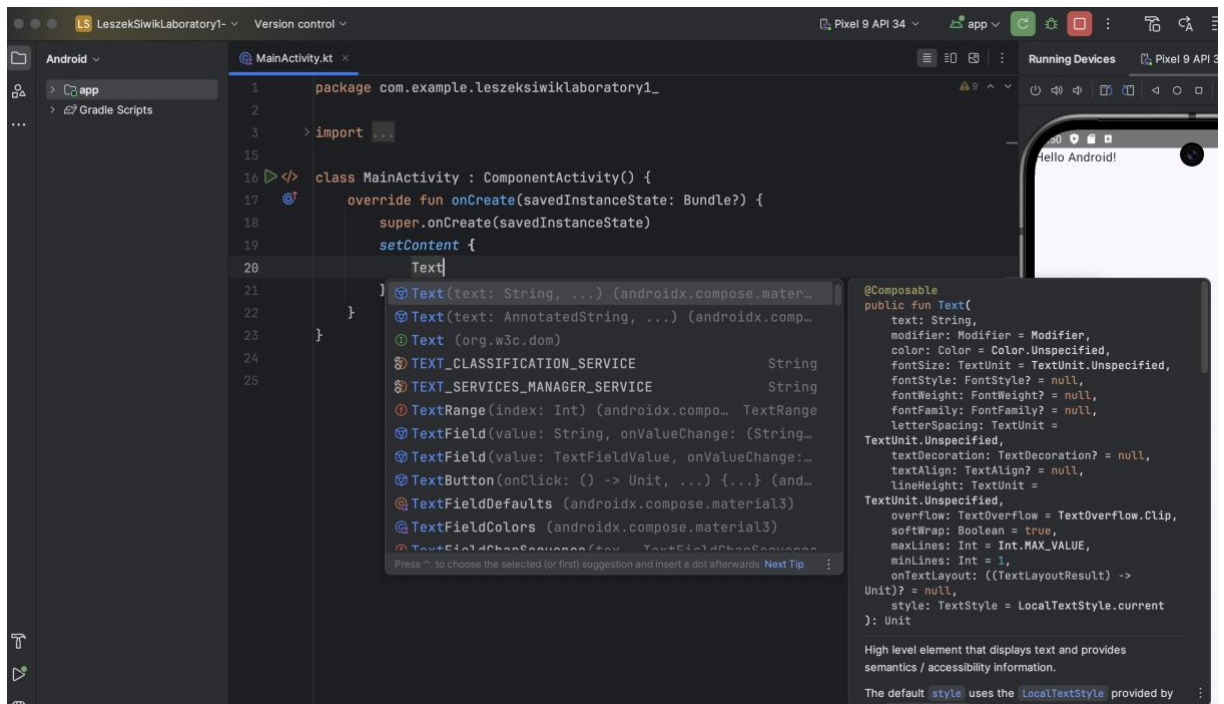


The screenshot shows the same IDE window, but the MainActivity.kt file now contains only the onCreate method, with the composable functions removed:

```
1 package com.example.leszeksiwiklaboratory1_
2
3 > import ...
4
15
16 class MainActivity : ComponentActivity() {
17     override fun onCreate(savedInstanceState: Bundle?) {
18         super.onCreate(savedInstanceState)
19         setContent {
20             // Composable content goes here
21         }
22     }
23 }
```

The code now only includes the package declaration, the import statement, and the MainActivity class with the onCreate method. The composable functions Greeting and GreetingPreview have been removed.

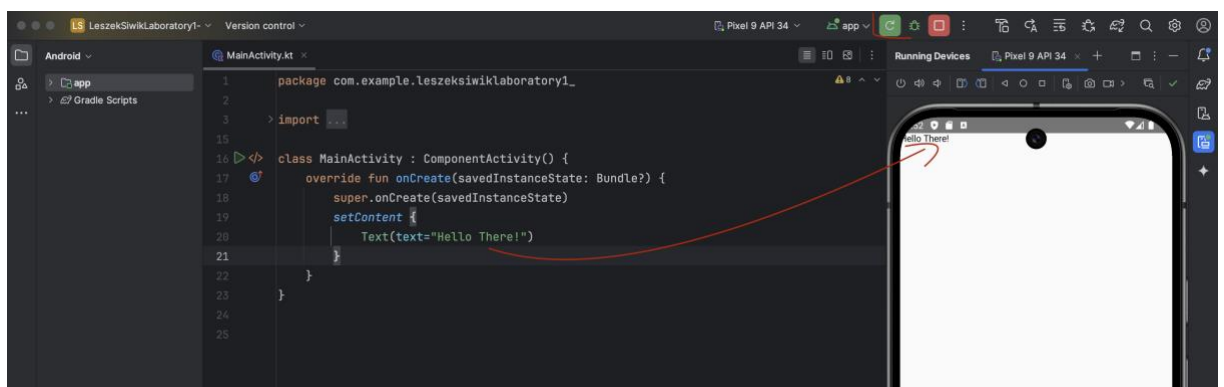
- Let's start working on our application by adding the text "Hello There".
- So, let's go inside the setContent function. The component / function that we need to use is Text, so as we begin typing "Text", Android Studio should provide us with available options.



- We want to use the first option, so let's press Enter and fill in the "text" attribute with the text we want to be displayed, i.e. in our case "Hello There", like this:



- Let's rebuild our application (we click on the green arrow / the button next to the "emulator selection drop-down menu"), and we should see our text on the emulator:



- Let's try to customize our text a bit. If we look at the parameters of the Text function ((e.g. by hovering over the Text function.):

```
3      > import ...
15
16  class MainActivity : AppCompatActivity() {
17      override fun onCreate(savedInstanceState: Bundle?) {
18          super.onCreate(savedInstanceState)
19          setContent {
20              Text(text="Hello There!")
21          }
22      }
23  }
24
25
```

@Composable

public fun Text(
text: String,
modifier: Modifier = Modifier,
color: Color = Color.Unspecified,
fontSize: TextUnit = TextUnit.Unspecified,
fontStyle: FontStyle? = null,
fontWeight: FontWeight? = null,
fontFamily: FontFamily? = null,
letterSpacing: TextUnit = TextUnit.Unspecified,
textDecoration: TextDecoration? = null,
textAlign: TextAlign? = null,
lineHeight: TextUnit = TextUnit.Unspecified,
overflow: TextOverflow = TextOverflow.Clip,
softWrap: Boolean = true,
maxLines: Int = Int.MAX_VALUE,
minLines: Int = 1,
onTextLayout: ((TextLayoutResult) -> Unit)? = null,
style: TextStyle = LocalTextStyle.current
): Unit

High level element that displays text and provides semantics / accessibility information.

The default `style` uses the `LocalTextStyle` provided by the `MaterialTheme` / components. If you are setting your own style, you may want to consider first retrieving `LocalTextStyle` and using `TextStyle`.

-
- We may see that besides the "text" parameter allowing us to set the content/inscription that we want to make it displayed, there are many other parameters that "format" our text. So let's try to use a few of them. For example, let's change the font color to red (set the "color" parameter to `Color.Red`), increase the font size (set the "fontSize" parameter to, for example, `20.sp`), and add spacing between characters in the text by setting the "letterSpacing" parameter to, for example, `3.sp`. So, finally, our Text is currently constructed as follows:

```

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView {
            Text(
                text = "Hello There!",
                color = Color.Red,
                fontSize = 20.sp,
                letterSpacing = 3.sp
            )
        }
    }
}

```

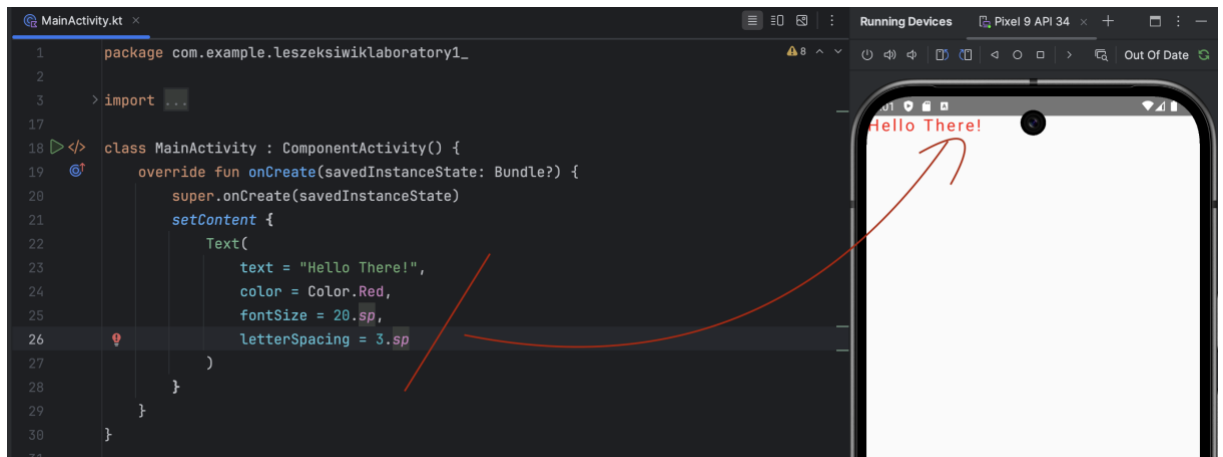
-
- While using some parameters / functions / components for the first time (like the “sp” option shown below), you potentially need to import it to your project / class by pressing Alt+Enter (btw sp comes from “scalable pixels and is the typical unit we use for defining the text/font size to make it displayed properly/similarly on the devices with different screen size/resolution:

```

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView {
            Text(
                text = "Hello There!",
                color = Color.Red,
                fontSize = 20.sp,
                letterSpacing = 3.sp
            )
        }
    }
}

```

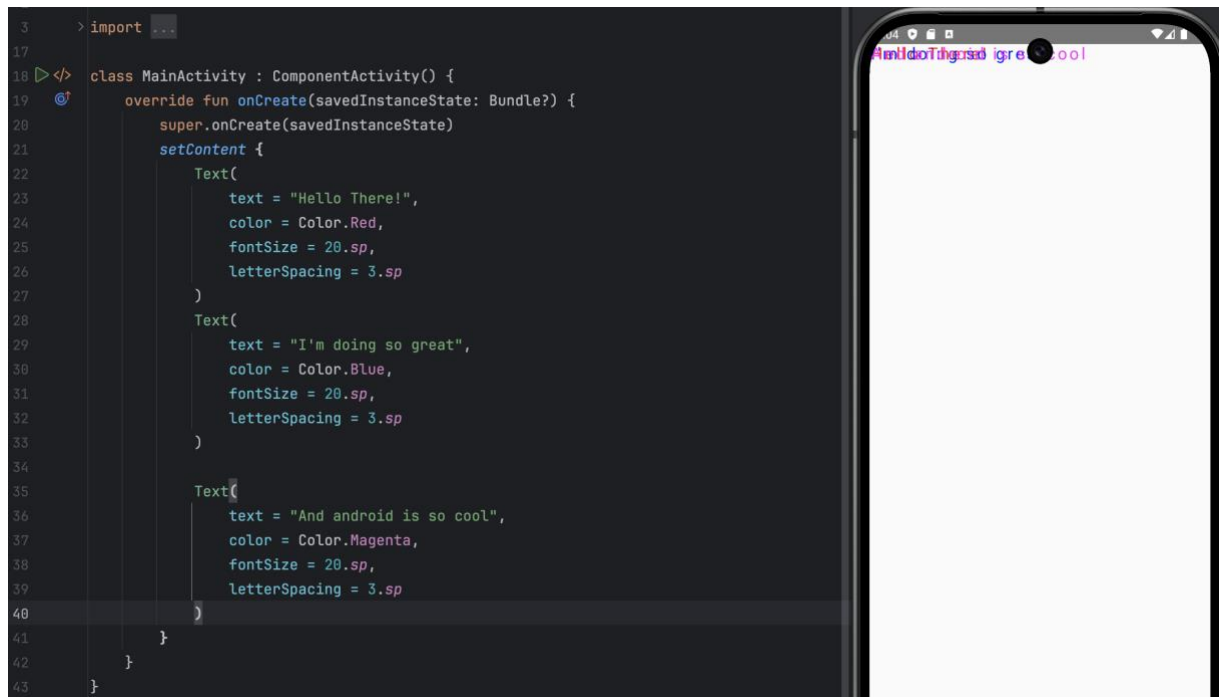
-
- Ok, when done, let’s see how it looks like:



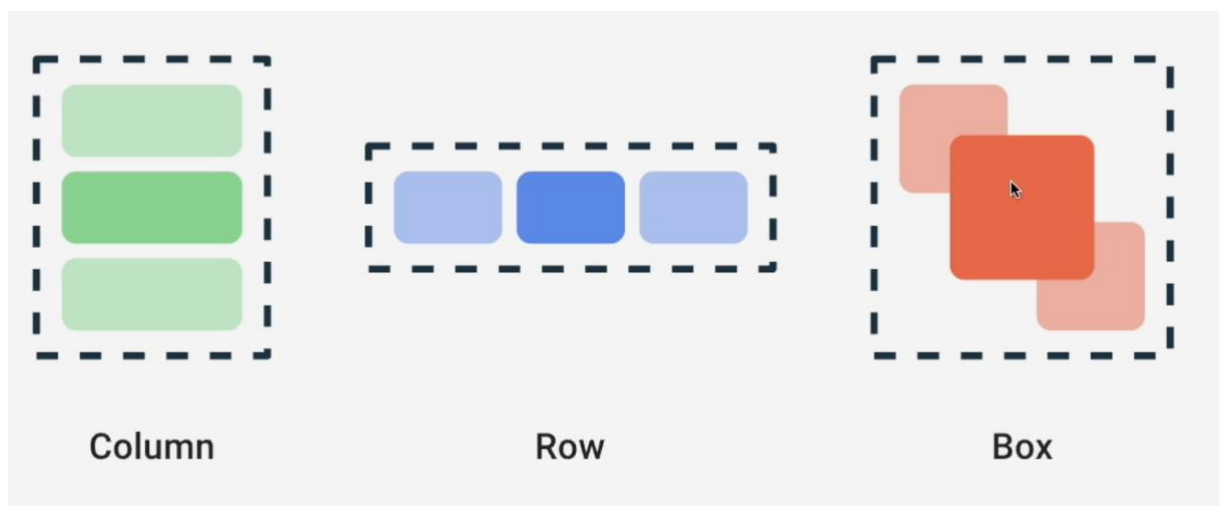
- If works, let's add a few more Text elements in a similar way.



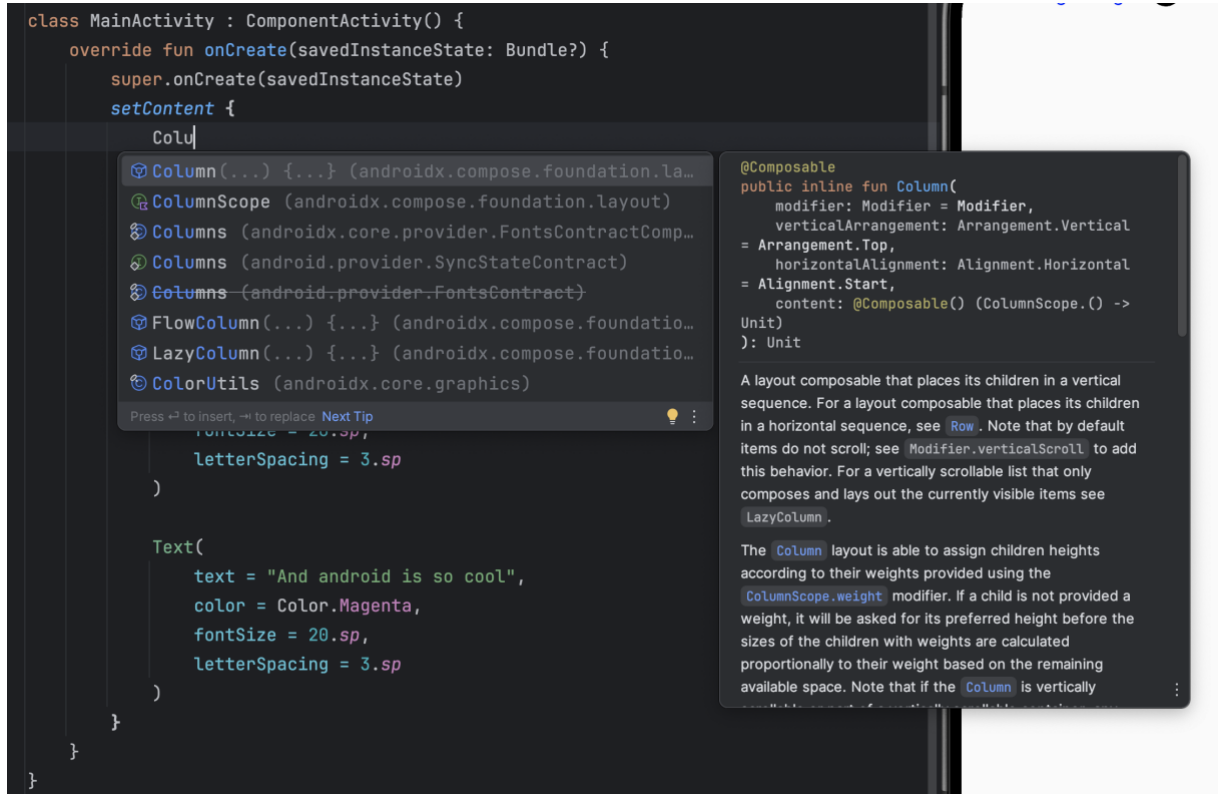
- And let's see how it looks like:



- Well,.....☺
- The issue arises because we didn't specify how these additional elements should be positioned on the screen, so they all positioned themselves by default, which is in the top-left corner of the application and that is why they are overlapping each other. The simplest way to solve this problem is to "wrap" our texts in containers like Column, Row, or Box, depending on how we want to arrange them.



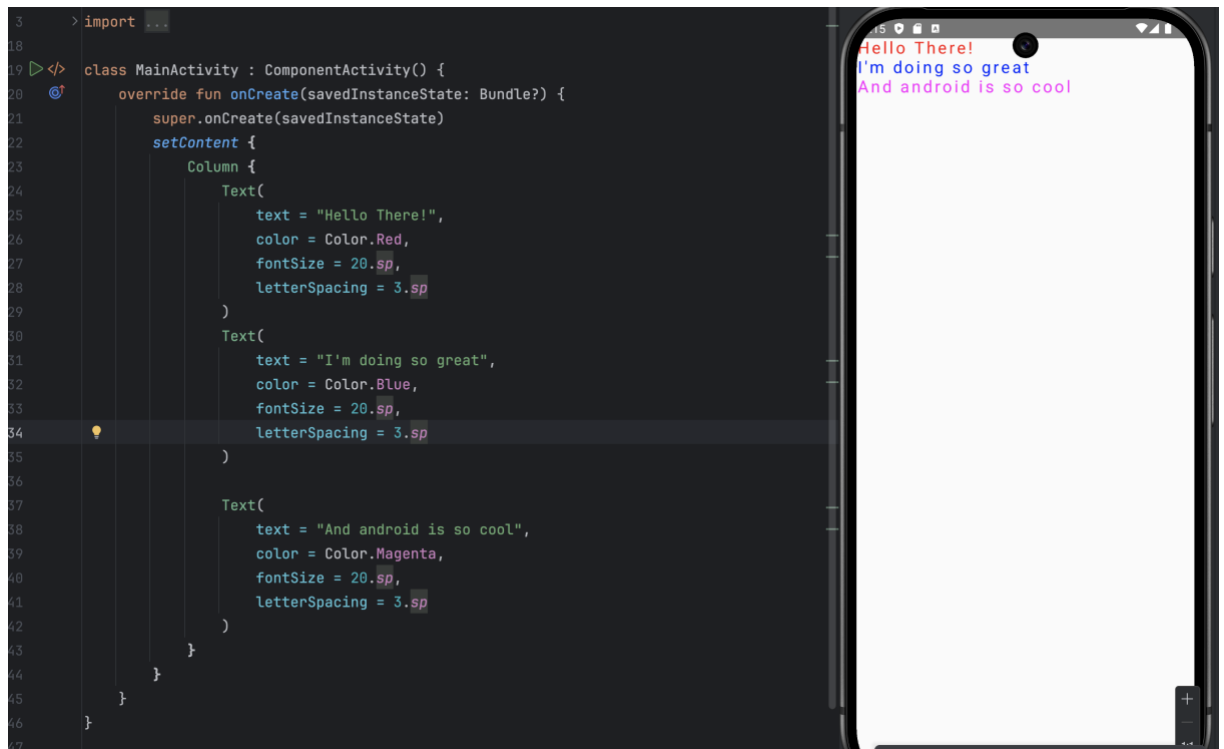
- Let's assume we want to place them one below the other; we'll use a Column container. So, we go back to our setContentView function, add an empty line before our first text element, and start typing Column. Android Studio will show us the available options.



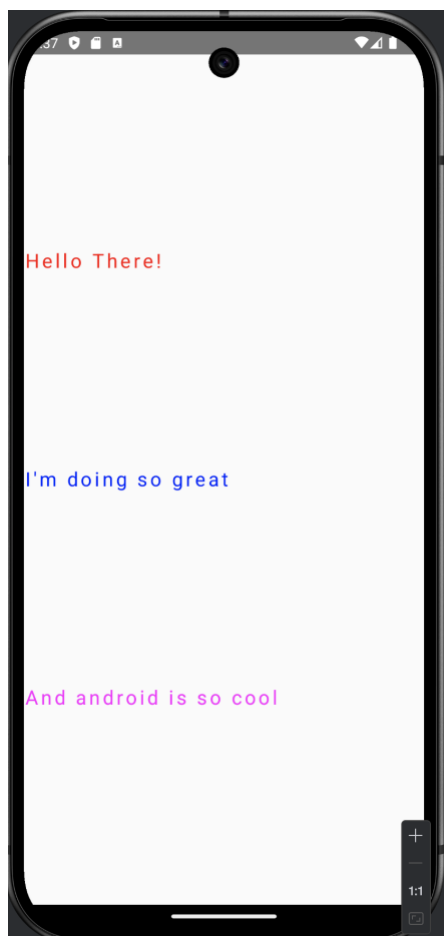
- We're interested in the first option, so we press Enter and move our Text elements inside our column.



- And let's see how it looks like now:



-
- I would say that much better 😊
- Ok, what if we would like to "spread out" our texts evenly across the screen like below:



-

- If we check out the parameters / arguments of our function Column:

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            Column(
                ) {
                    @Composable
                    public inline fun Column(
                        modifier: Modifier = Modifier,
                        verticalArrangement: Arrangement.Vertical = Arrangement.Top,
                        horizontalAlignment: Alignment.Horizontal = Alignment.Start,
                        content: @Composable() () -> Unit
                    ): Unit

                    A layout composable that places its children in a vertical sequence. For a layout
                    composable that places its children in a horizontal sequence, see Row. Note that
                    by default items do not scroll; see Modifier.verticalScroll to add this behavior.
                    For a vertically scrollable list that only composes and lays out the currently visible
                    items see LazyColumn.

                    The Column layout is able to assign children heights according to their weights
                    provided using the ColumnScope.weight modifier. If a child is not provided a
                    weight, it will be asked for its preferred height before the sizes of the children
                    with weights are calculated proportionally to their weight based on the remaining
                    available space. Note that if the Column is vertically scrollable or part of a
                    vertically scrollable container, any provided weights will be disregarded as the
                    remaining available space will be infinite.

                    When none of its children have weights, a Column will be as small as possible to
                    fit its children one on top of the other. In order to change the height of the
                    Column, use the Modifier.height modifiers; e.g. to make it fill the available
                    height, use Modifier.fillMaxHeight.

                    font-size = 20.sp,
                    letterSpacing = 3.sp
                }
            }
        }
    }
}
```

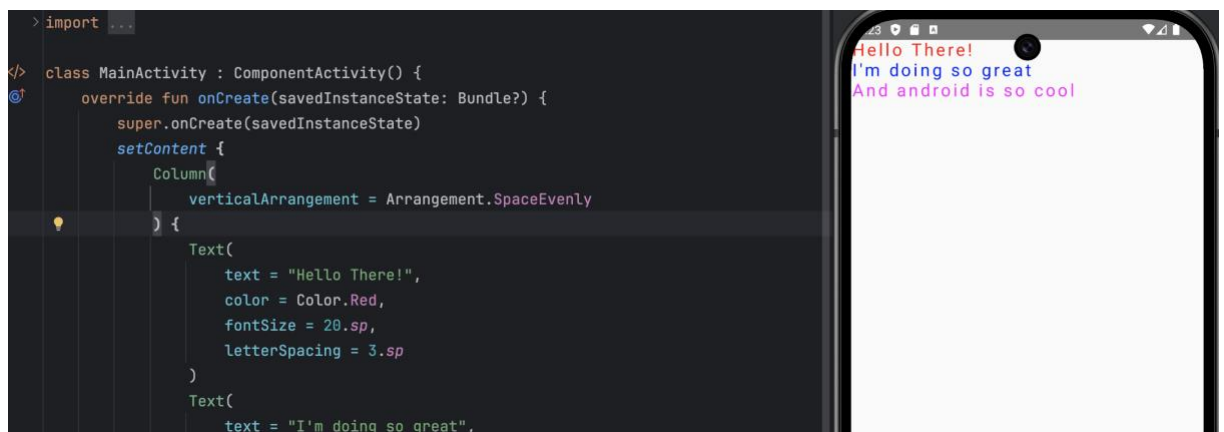
- We may see that there, among other parameters, the verticalArrangement parameter responsible for the vertical arrangement of elements in a column. So let's try using it:

```

> class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            Column(
                verticalArrangement = Arrangement.SpaceEvenly
            ) {
                Text(
                    text = "Hello There!",
                    color = Color.Red,
                    fontSize = 20.sp,
                    letterSpacing = 3.sp
                )
                Text(
                    text = "I'm doing so great",
                    color = Color.Blue,
                    fontSize = 20.sp,
                    letterSpacing = 3.sp
                )
            }
        }
    }
}

```

-
- Let's rebuild our app and see how it looks like :



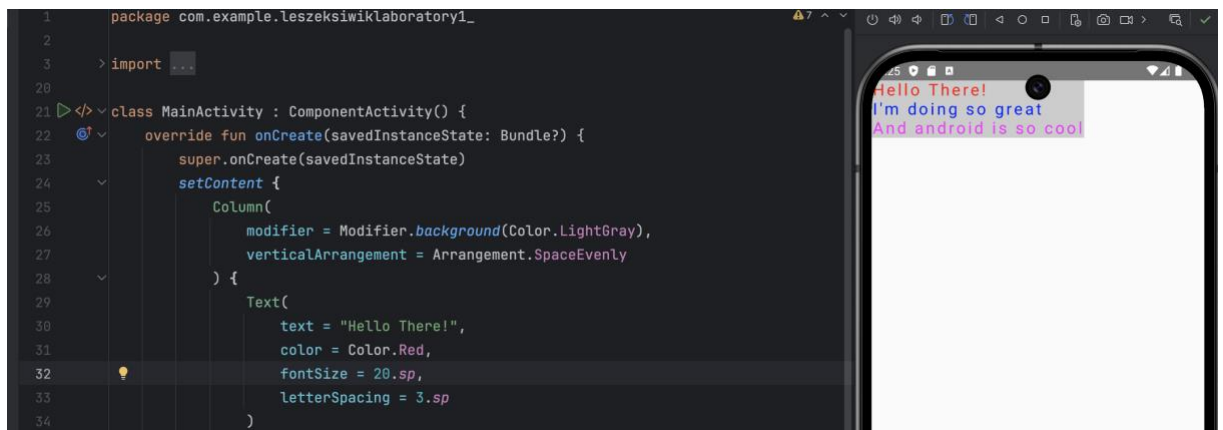
- Well..... It is not really what we expected.
- Let's take a look at how the Column itself is currently positioned on the screen. The easiest way to do this is by setting some background color for it, and we'll see exactly what's happening there.
- The Column function doesn't have explicitly the parameter like "background" or "backgroundColor," but it does have a modifier parameter through which we can modify/set many parameters of the element that weren't extracted as explicitly visible parameters of the element we want to use. This is a quite typical situation i.e. in many cases, the most commonly used parameters (like size or font color for text) are available explicitly, while the less commonly used ones are available through a modifier.

- Ok, so let's set the background color for our column:

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            Column(
                modifier = Modifier.background(Color.LightGray),
                verticalArrangement = Arrangement.SpaceEvenly
            ) {
                Text(
                    text = "Hello There!",
                    color = Color.Red,

```

- And let's find out the result:

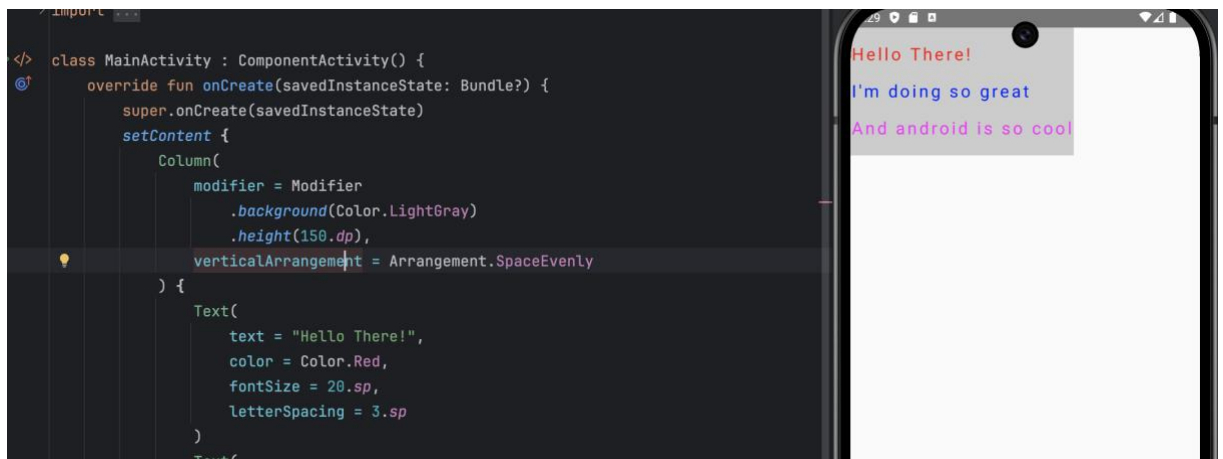


- And that is the reason. As you can see, if we don't define it explicitly, the size of the column (both height and width) is set by default just to "cover" the elements placed in it. And in this case, the .SpaceEvenly option doesn't have a chance to demonstrate itself because currently, there is simply no space in the column to "spread out" the texts.
- Ok, so let's increase the height of the column and see if SpaceEvenly will have a chance to "work". Again, the height attribute is not directly visible among the arguments of the Column function, so we'll do this by adding an appropriate (additional) option for the modifier, for example, like this:

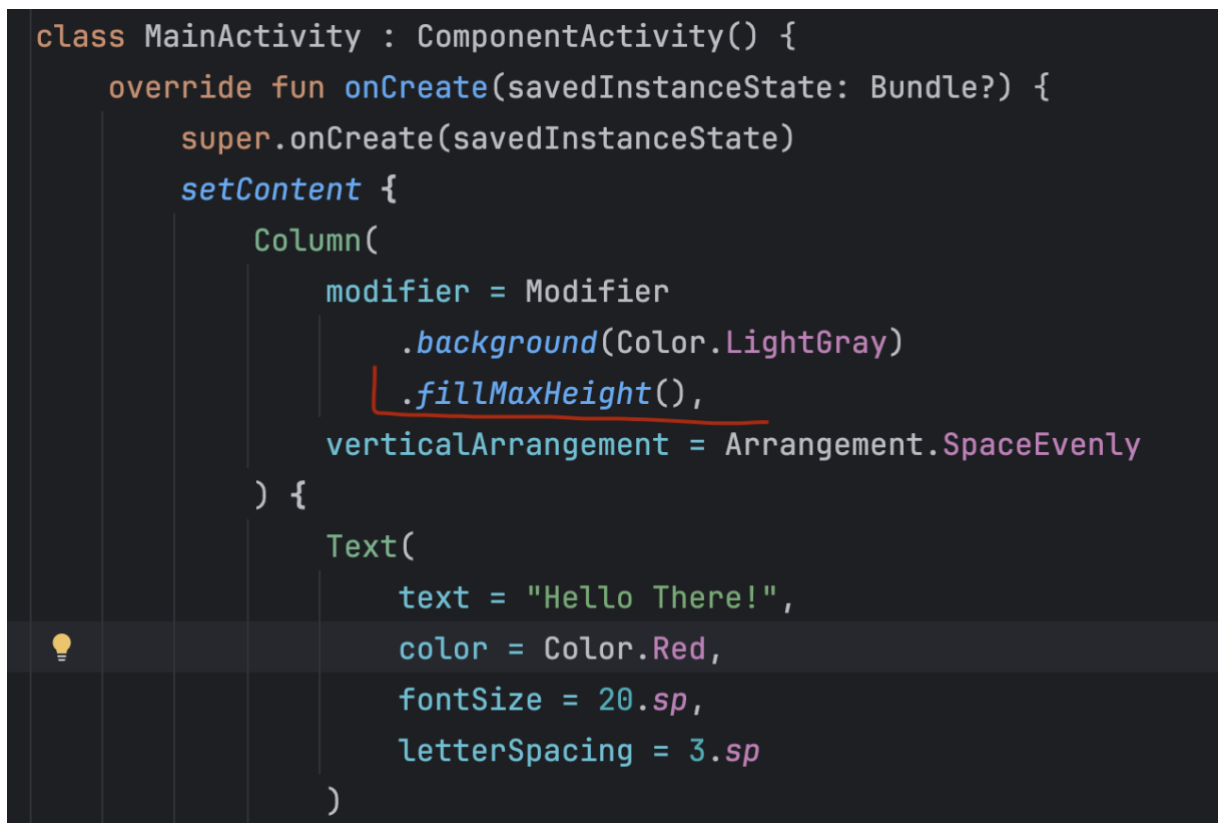
```
setContent {
    Column(
        modifier = Modifier
            .background(Color.LightGray)
            .height(150.dp),
        verticalArrangement = Arrangement.SpaceEvenly
    ) {

```

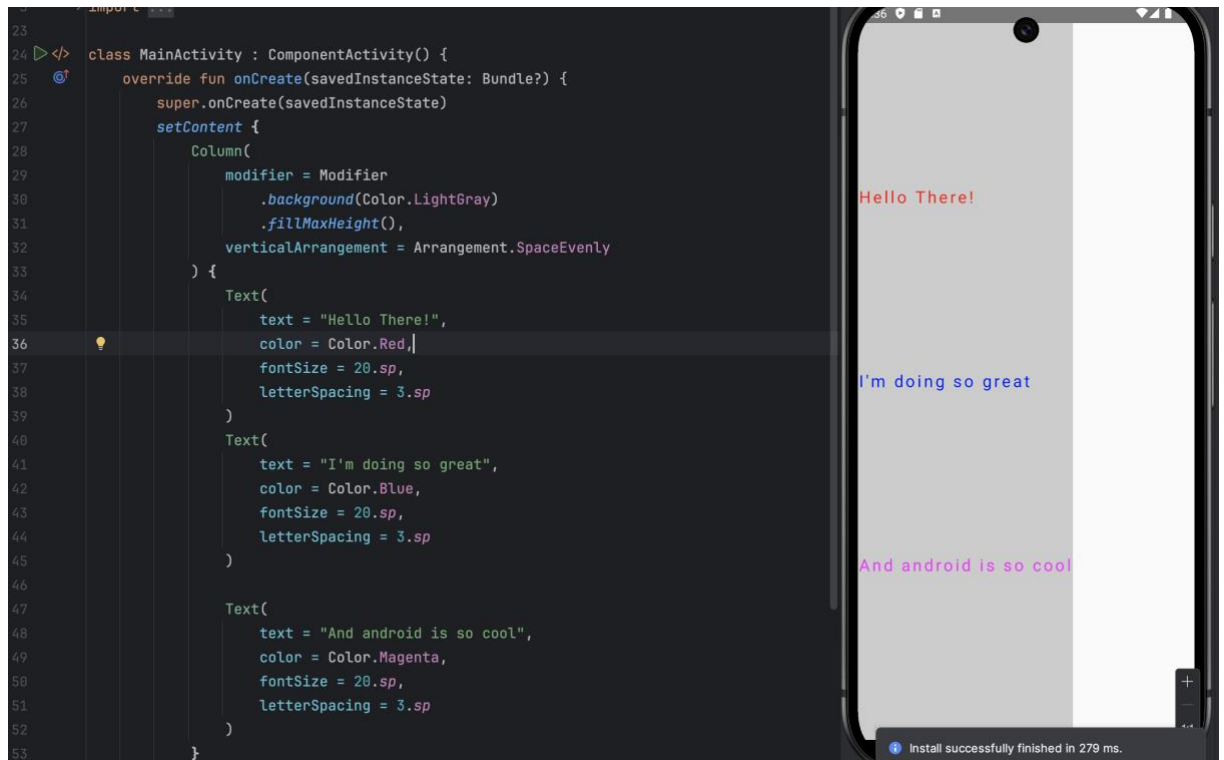
- And let's check the result:



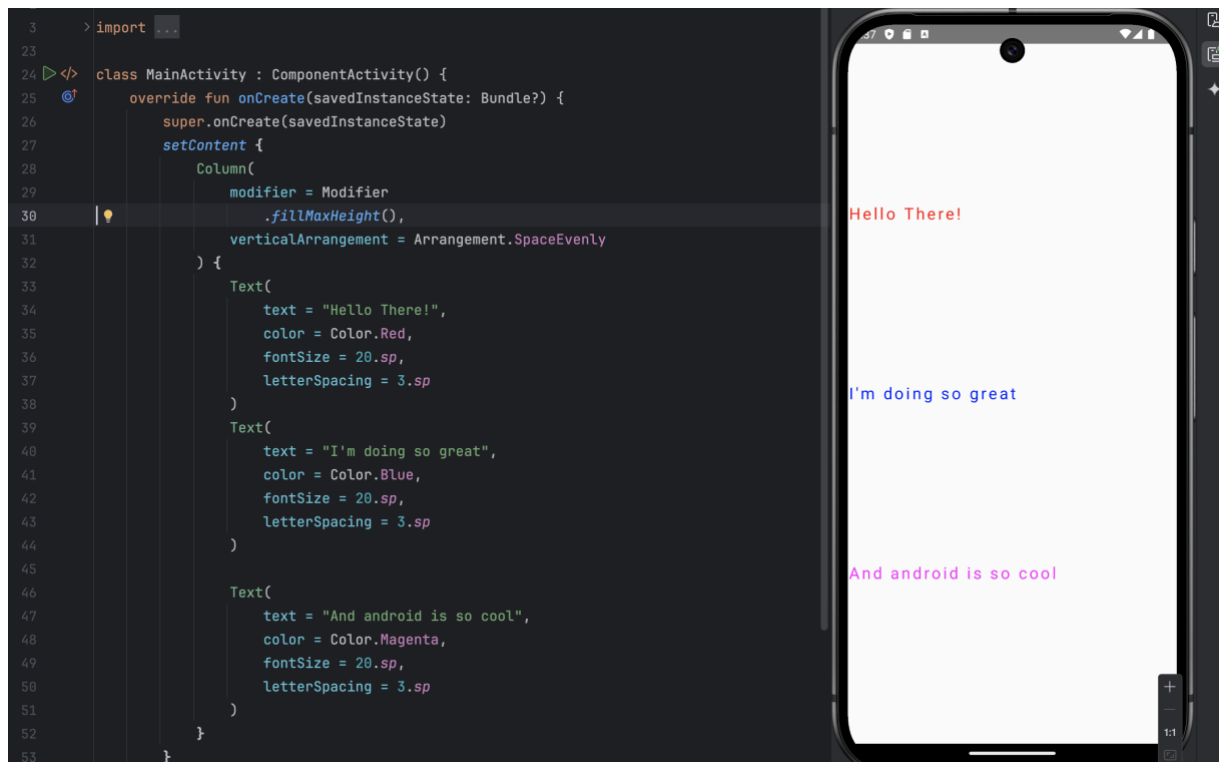
- Exactly, so it seems to work. Therefore, if we set the height of the column equal to the height of the phone screen, we should achieve what we want. By specifying a specific value in the .height parameter of the modifier, we potentially face two problems: what value to use there and whether it will look the same on phones with different screen sizes. Fortunately, one of the options for the modifier is .fillMaxHeight, which will take care of it for us, so let's use it instead of the height option.



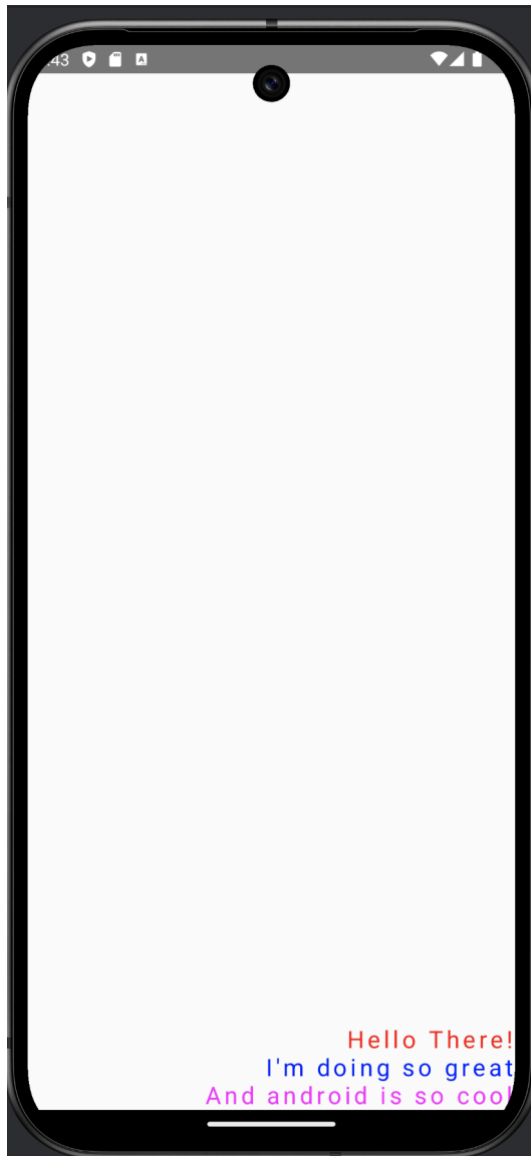
- And let's see how it looks like:



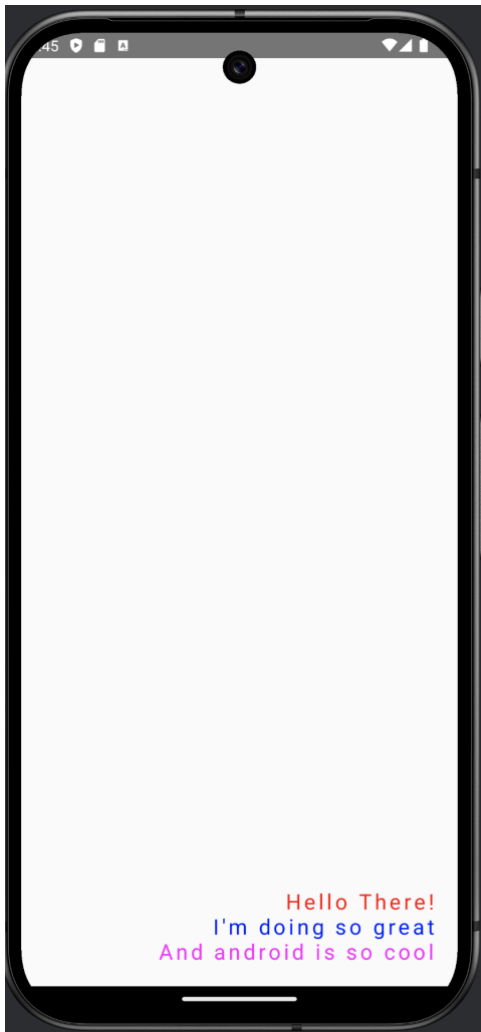
- Almost there. If we now remove our "debugging" background color for the column, then we'll get exactly what we wanted.



- Ok, then. So as for practicing pls try to get the following result:



-
- Or, even better, with some paddings on the right-hand side and at the bottom like:



-
- Before we move on let's extract what we have done so far into the separate function/method called e.g. MyTexts

```
@Composable
fun MyTexts() {
    Column(
        modifier = Modifier
```

-
- When done, please upload on upel platform the screenshot of your code and the emulator to confirm that everything is up and running