# 20 Material Design в Jetpack Compose

[Material Design](https://material.io/) — это система дизайна, созданная и поддерживаемая дизайнерами и разработчиками Google для создания высококачественных цифровых приложений для Android, а также других мобильных и веб-платформ. Он содержит рекомендации по созданию пользовательского интерфейса вашего приложения в читабельной, привлекательной и последовательной форме.

В этой лаборатории кода вы узнаете о Material Theming, которая позволяет использовать Material Design в вашем приложении, а также получите рекомендации по настройке цветов, типографики и форм. Вы можете настроить для своего приложения столько, сколько захотите. Вы также узнаете, как добавить верхнюю панель приложения для отображения имени и значка приложения.

Что вы узнаете

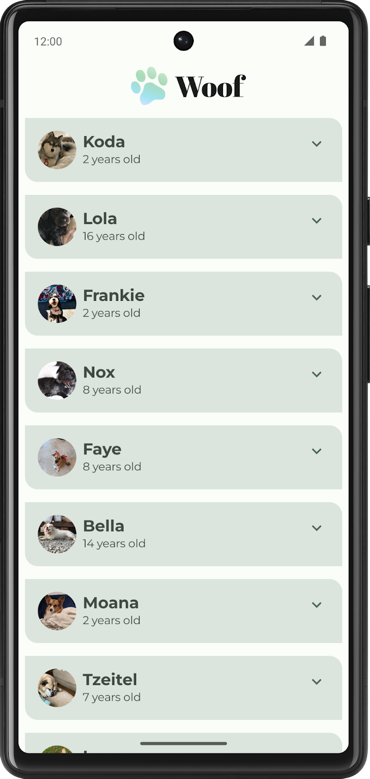
* Как применить тему материала к приложению Compose.
* Как добавить собственную цветовую палитру в ваше приложение.
* Как добавить собственные шрифты в ваше приложение.
* Как добавить собственные формы к элементам вашего приложения.
* Как добавить верхнюю панель приложения в ваше приложение.

Что ты построишь

* Вы создадите красивое приложение, включающее в себя лучшие практики Material Design.

## [2. Обзор приложения](https://developer.android.com/codelabs/basic-android-kotlin-compose-material-theming?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-3-pathway-3%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-material-theming#1)

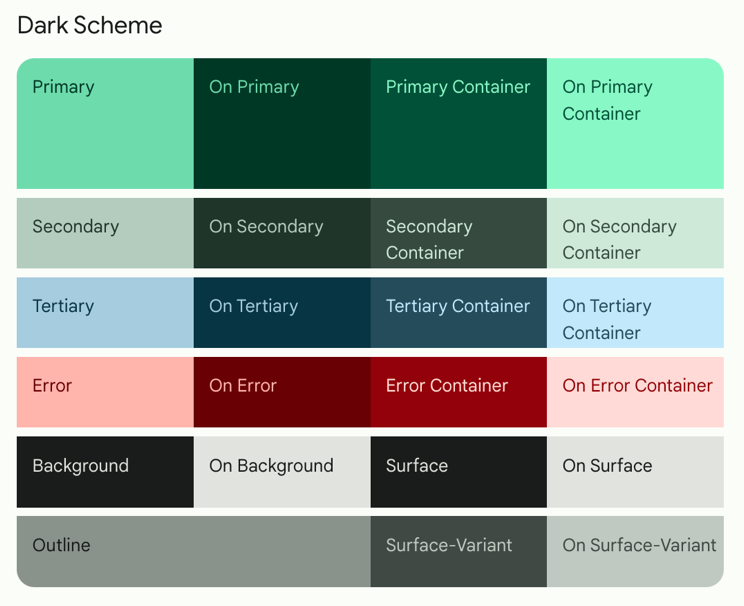
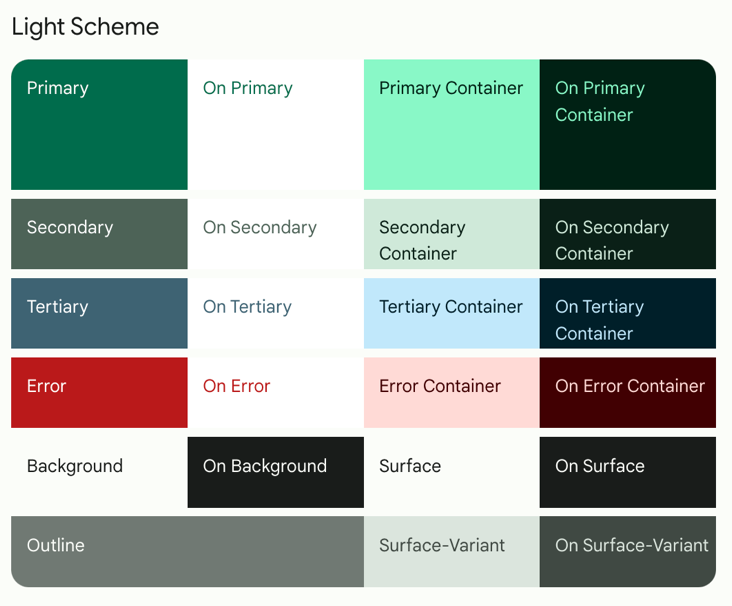
В этой лаборатории кода вы создадите **Woof** — приложение, которое отображает список собак и использует Material Design для создания красивого интерфейса приложения.



В этой лаборатории кода вы найдете идеи о том, как использовать Material Theming для улучшения внешнего вида приложений, которые вы создадите в будущем.

### **Цветовая палитра**

Ниже приведены цветовые палитры для светлых и темных тем, которые мы создадим.

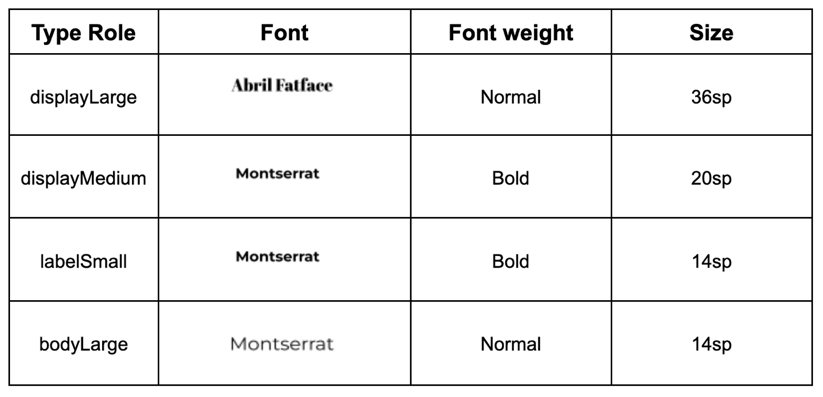


Приложение как в светлой, так и в темной теме.

|  |  |
| --- | --- |
| **Светлая тема** | **Темная тема** |
|  |  |

### **Типография**

Ниже приведены стили шрифтов, которые вы будете использовать в приложении.



### **Файл темы**

Файл Theme.kt — это файл **,** содержащий всю информацию о теме приложения, которая определяется цветом, типографикой и формой. Это важный файл, который вам следует знать. Внутри файла находится Composable  компонент WoofTheme(), который задает цвета, типографику и формы приложения.

@Composable  
fun WoofTheme(  
    darkTheme: Boolean = isSystemInDarkTheme(),  
    // Dynamic color is available on Android 12+  
    dynamicColor: Boolean = false,  
    content: @Composable () -> Unit  
) {  
    val colorScheme = when {  
        dynamicColor && Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.S -> {  
            val context = LocalContext.current  
            if (darkTheme) dynamicDarkColorScheme(context) else dynamicLightColorScheme(context)  
        }  
  
        darkTheme -> DarkColors  
        else -> LightColors  
    }  
    val view = LocalView.current  
    if (!view.isInEditMode) {  
        SideEffect {  
            setUpEdgeToEdge(view, darkTheme)  
        }  
    }  
  
    MaterialTheme(  
        colorScheme = colorScheme,  
        shapes = Shapes,  
        typography = Typography,  
        content = content  
    )  
}  
  
/\*\*  
 \* Sets up edge-to-edge for the window of this [view]. The system icon colors are set to either  
 \* light or dark depending on whether the [darkTheme] is enabled or not.  
 \*/  
private fun setUpEdgeToEdge(view: View, darkTheme: Boolean) {  
    val window = (view.context as Activity).window  
    WindowCompat.setDecorFitsSystemWindows(window, false)  
    window.statusBarColor = Color.Transparent.toArgb()  
    val navigationBarColor = when {  
        Build.VERSION.SDK\_INT >= 29 -> Color.Transparent.toArgb()  
        Build.VERSION.SDK\_INT >= 26 -> Color(0xFF, 0xFF, 0xFF, 0x63).toArgb()  
        // Min sdk version for this app is 24, this block is for SDK versions 24 and 25  
        else -> Color(0x00, 0x00, 0x00, 0x50).toArgb()  
    }  
    window.navigationBarColor = navigationBarColor  
    val controller = WindowCompat.getInsetsController(window, view)  
    controller.isAppearanceLightStatusBars = !darkTheme  
    controller.isAppearanceLightNavigationBars = !darkTheme  
}

В **MainActivity.kt** добавлена WoofTheme()​​тема оформления материала для всего приложения.

class MainActivity : ComponentActivity() {  
   override fun onCreate(savedInstanceState: Bundle?) {  
       super.onCreate(savedInstanceState)  
       setContent {  
           **WoofTheme** {  
               Surface(  
                   modifier = Modifier.fillMaxSize()  
               ) {  
                   WoofApp()  
               }  
           }  
       }  
   }  
}

Взгляните на WoofPreview(). Добавлен WoofTheme() для предоставления тематики материала, которую вы видите в файле WoofPreview().

@Preview  
@Composable  
fun WoofPreview() {  
    **WoofTheme(darkTheme = false)** {  
        WoofApp()  
    }  
}

[3. Получите стартовый код](https://developer.android.com/codelabs/basic-android-kotlin-compose-material-theming?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-3-pathway-3%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-material-theming#2)

Для начала загрузите стартовый код:

[file\_downloadСкачать zip](https://github.com/google-developer-training/basic-android-kotlin-compose-training-woof/archive/refs/heads/starter.zip)

Альтернативно вы можете клонировать репозиторий GitHub для кода:

$ git clone https://github.com/google-developer-training/basic-android-kotlin-compose-training-woof.git

$ cd basic-android-kotlin-compose-training-woof

$ git checkout starter

**Примечание.** Стартовый код находится в  ветке starter скачанного репозитория.

Вы можете просмотреть код в  репозитории GitHub [Woof app](https://github.com/google-developer-training/basic-android-kotlin-compose-training-woof/tree/starter).

Изучите стартовый код

1. Откройте стартовый код в Android Studio.
2. Откройте **com.example.woof** > **data** > **Dog.kt.** ​Он содержит изображение Dog data class, которое будет использоваться для представления фотографии, имени, возраста и хобби собаки. Он также содержит список собак и информацию, которую вы будете использовать в качестве данных в своем приложении.



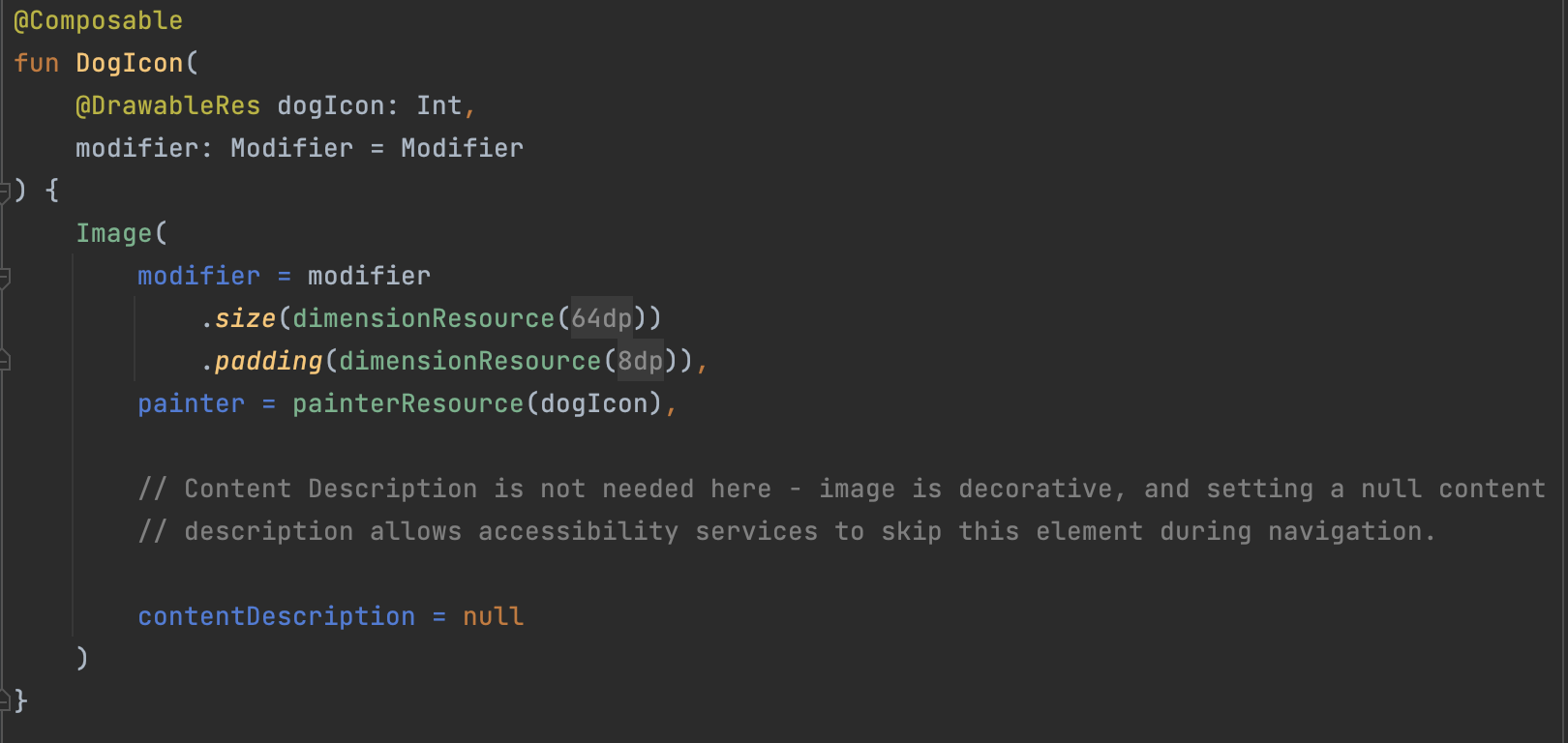
1. Открытое **res** > **drawable**. Он содержит все ресурсы изображений, необходимые для этого проекта, включая значок приложения, изображения собак и значки.
2. Откройте **res** > **values** ​​> **strings.xml** . Он содержит строки, которые вы используете в этом приложении, включая имя приложения, имена собак, их описания и многое другое.
3. Откройте **MainActivity.kt** . Он содержит код для создания простого списка, отображающего фотографию собаки, ее имя и возраст.
4. WoofApp() содержит LazyColumn, который отображает DogItem.



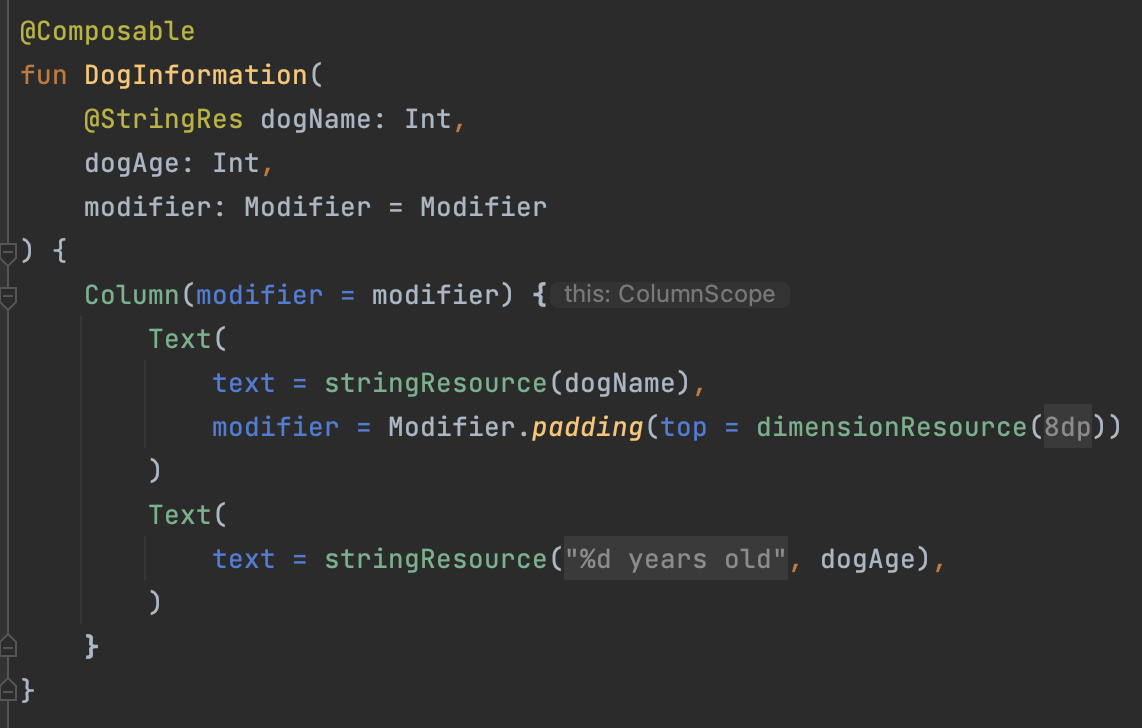
1. DogItem() содержит контейнер Row, в котором отображается фотография собаки и информация о ней.



1. DogIcon()показывает фотографию собаки.



1. DogInformation()отображает имя и возраст собаки.



1. WoofPreview()позволяет предварительно просмотреть приложение на панели **«Дизайн»** .



**Примечание** . Вы можете заметить, что каждый @Composable метод имеет модификатор, добавленный в качестве параметра. В Compose рекомендуется передавать параметр-модификатор компонуемым функциям. Это связано с тем, что он позволяет родительскому составному элементу передавать контекстную информацию дочернему составному элементу.

Например, если у вас есть кнопка, один родитель может захотеть, чтобы ее дочерняя кнопка использовала полный доступный размер, а другой родитель мог бы захотеть, чтобы она обертывала содержимое. Это делает код более пригодным для повторного использования. Этот параметр обычно называется «модификатором» и должен отображаться как первый необязательный параметр в списке параметров функции. Этот модификатор применяется к первому дочернему элементу метода. Подробнее об этом читайте в [Руководстве по API для Jetpack Compose](https://android.googlesource.com/platform/frameworks/support/+/androidx-main/compose/docs/compose-api-guidelines.md#elements-accept-and-respect-a-modifier-parameter) .

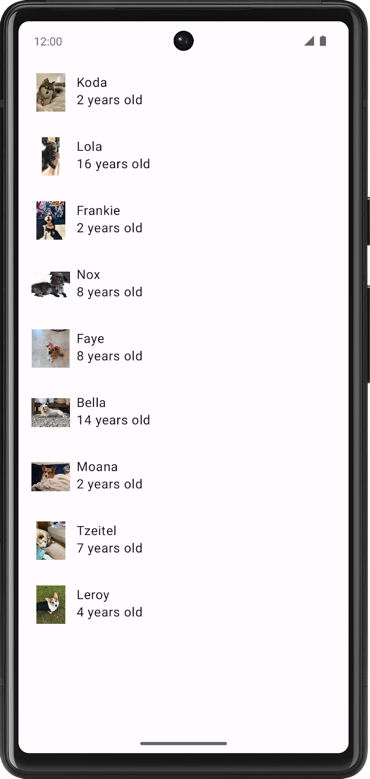
Убедитесь, что ваш эмулятор/устройство имеет светлую тему.

В этой кодовой лаборатории вы будете работать как со светлыми, так и с темными темами, однако большая часть кодовой лаборатории выполнена в светлой теме. Прежде чем начать, убедитесь, что на вашем устройстве/эмуляторе установлена ​​светлая тема.

Чтобы просмотреть свое приложение в светлой теме на эмуляторе или физическом устройстве:

1. Зайдите в приложение **«Settings»** на устройстве.
2. Найдите **Dark theme** и щелкните ее.
3. Если **Dark theme** включена, отключите ее.

Запустите стартовый код, чтобы увидеть, с чего вы начинаете; это список, в котором показаны собаки с их фотографиями, именами и возрастом. Он функционален, но выглядит не очень хорошо, поэтому мы собираемся это исправить.

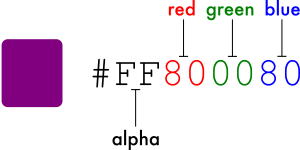


[4. Добавьте цвет](https://developer.android.com/codelabs/basic-android-kotlin-compose-material-theming?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-3-pathway-3%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-material-theming#3)

Первое, что вы собираетесь изменить в приложении **Woof,** — это цветовая схема.

Цветовая схема — это комбинация цветов, которые использует ваше приложение. Различные цветовые комбинации вызывают разное настроение, что влияет на то, как люди себя чувствуют, когда используют ваше приложение.

Цвет в системе Android представлен шестнадцатеричным (шестнадцатеричным) значением цвета. Шестнадцатеричный код цвета начинается с символа решетки (#), за которым следуют шесть букв и/или цифр, обозначающих красный, зеленый и синий (RGB) компоненты этого цвета. Первые две буквы/цифры относятся к красному цвету, следующие две относятся к зеленому, а последние две относятся к синему.



Цвет также может включать в себя альфа-значение — буквы и/или цифры, — которое представляет прозрачность цвета (#00 — непрозрачность 0 % (полностью прозрачный), #FF — непрозрачность 100 % (полностью непрозрачный)). Если оно включено, альфа-значение представляет собой первые два символа шестнадцатеричного кода цвета после символа решетки (#). Если значение альфа не включено, предполагается, что оно равно #FF, что соответствует 100% непрозрачности (полностью непрозрачно).

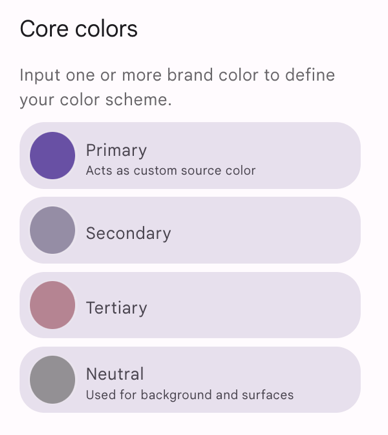
Ниже приведены примеры цветов и их шестнадцатеричные значения.



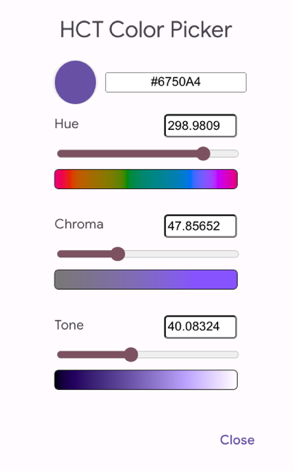
Используйте Material Theme Builder для создания цветовой схемы.

Чтобы создать собственную цветовую схему для нашего приложения, мы будем использовать Material Theme Builder.

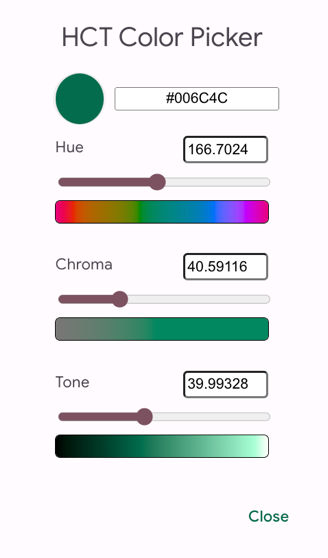
1. Нажмите эту ссылку, чтобы перейти к [конструктору тем материалов](https://m3.material.io/theme-builder#/custom) .
2. На левой панели вы увидите Core Colors, нажмите «Primary»:



1. Откроется палитра цветов HCT.



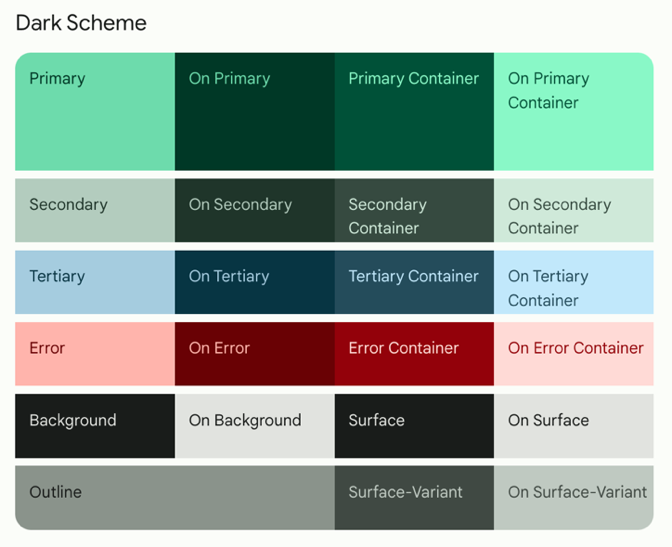
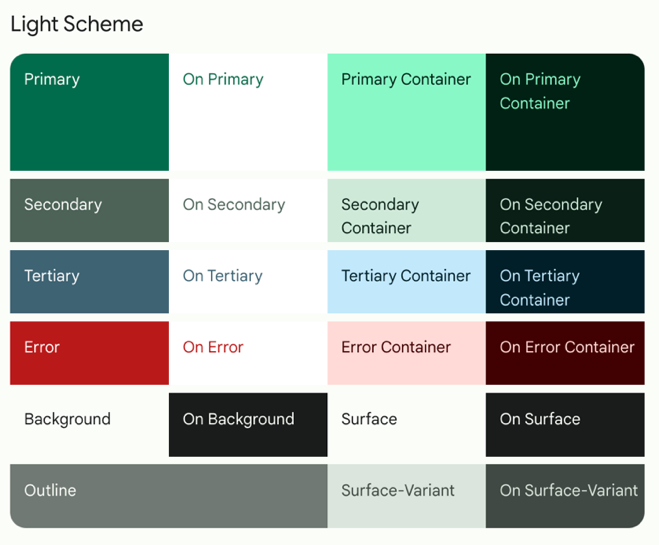
1. Чтобы создать цветовую схему, показанную на скриншотах приложения, вы измените основной цвет в этой палитре цветов. В текстовом поле замените текущий текст на **#006C4C** . Нажмите Apply. Это сделает основной цвет приложения зеленым.



Обратите внимание, как при этом приложения на экране приобретают зеленую цветовую схему.



1. Прокрутите страницу вниз, и вы увидите полную цветовую схему для светлой и темной темы, созданную на основе введенного вами цвета.



Вы можете задаться вопросом, что это за роли и как они используются. Вот некоторые из основных:

* **Основные primary цвета** используются для ключевых компонентов пользовательского интерфейса.
* **Вторичные secondary цвета** используются для менее заметных компонентов пользовательского интерфейса.
* Третичные **tertiary** цвета используются для контрастных акцентов, которые можно использовать, чтобы сбалансировать первичные и вторичные цвета или привлечь повышенное внимание к элементу, например, к полю ввода **.**
* Элементы **цвета** отображаются **поверх** других цветов в палитре и в основном применяются к тексту, значкам и штрихам. В нашей цветовой палитре есть цвет **onSurface** , который появляется поверх цвета **поверхности** , и цвет **onPrimary** , который появляется поверх **основного** цвета.

Наличие этих слотов приводит к созданию целостной системы дизайна, в которой связанные компоненты окрашены одинаково.

Хватит теории о цветах — пора добавить в приложение эту прекрасную цветовую палитру!

Добавить цветовую палитру в тему

На странице сайта «Материал Theme Builder» есть возможность нажать кнопку **«Export»** , чтобы загрузить файл **Color.kt** и файл **Theme.kt** с пользовательской темой, созданной вами в Theme Builder.

Примечание: кнопка Export спрятана на панели, которая вызывается нажатием кнопки в правом верхнем углу 

Это поможет добавить созданную нами пользовательскую тему в ваше приложение. Однако, поскольку сгенерированный файл **Theme.kt** не включает код динамического цвета, который мы рассмотрим позже в лаборатории кода, скопируйте файлы в него.

**Примечание** . Если вы решите использовать файлы, созданные с помощью Material Theme Builder, для другого проекта, вам нужно будет обновить имя пакета на имя пакета вашего проекта.

1. Откройте файл **Color.kt** и замените его содержимое приведенным ниже кодом, чтобы скопировать новую цветовую схему.

package com.example.woof.ui.theme  
  
import androidx.compose.ui.graphics.Color  
  
val md\_theme\_light\_primary = Color(0xFF006C4C)  
val md\_theme\_light\_onPrimary = Color(0xFFFFFFFF)  
val md\_theme\_light\_primaryContainer = Color(0xFF89F8C7)  
val md\_theme\_light\_onPrimaryContainer = Color(0xFF002114)  
val md\_theme\_light\_secondary = Color(0xFF4D6357)  
val md\_theme\_light\_onSecondary = Color(0xFFFFFFFF)  
val md\_theme\_light\_secondaryContainer = Color(0xFFCFE9D9)  
val md\_theme\_light\_onSecondaryContainer = Color(0xFF092016)  
val md\_theme\_light\_tertiary = Color(0xFF3D6373)  
val md\_theme\_light\_onTertiary = Color(0xFFFFFFFF)  
val md\_theme\_light\_tertiaryContainer = Color(0xFFC1E8FB)  
val md\_theme\_light\_onTertiaryContainer = Color(0xFF001F29)  
val md\_theme\_light\_error = Color(0xFFBA1A1A)  
val md\_theme\_light\_errorContainer = Color(0xFFFFDAD6)  
val md\_theme\_light\_onError = Color(0xFFFFFFFF)  
val md\_theme\_light\_onErrorContainer = Color(0xFF410002)  
val md\_theme\_light\_background = Color(0xFFFBFDF9)  
val md\_theme\_light\_onBackground = Color(0xFF191C1A)  
val md\_theme\_light\_surface = Color(0xFFFBFDF9)  
val md\_theme\_light\_onSurface = Color(0xFF191C1A)  
val md\_theme\_light\_surfaceVariant = Color(0xFFDBE5DD)  
val md\_theme\_light\_onSurfaceVariant = Color(0xFF404943)  
val md\_theme\_light\_outline = Color(0xFF707973)  
val md\_theme\_light\_inverseOnSurface = Color(0xFFEFF1ED)  
val md\_theme\_light\_inverseSurface = Color(0xFF2E312F)  
val md\_theme\_light\_inversePrimary = Color(0xFF6CDBAC)  
val md\_theme\_light\_shadow = Color(0xFF000000)  
val md\_theme\_light\_surfaceTint = Color(0xFF006C4C)  
val md\_theme\_light\_outlineVariant = Color(0xFFBFC9C2)  
val md\_theme\_light\_scrim = Color(0xFF000000)  
  
val md\_theme\_dark\_primary = Color(0xFF6CDBAC)  
val md\_theme\_dark\_onPrimary = Color(0xFF003826)  
val md\_theme\_dark\_primaryContainer = Color(0xFF005138)  
val md\_theme\_dark\_onPrimaryContainer = Color(0xFF89F8C7)  
val md\_theme\_dark\_secondary = Color(0xFFB3CCBE)  
val md\_theme\_dark\_onSecondary = Color(0xFF1F352A)  
val md\_theme\_dark\_secondaryContainer = Color(0xFF354B40)  
val md\_theme\_dark\_onSecondaryContainer = Color(0xFFCFE9D9)  
val md\_theme\_dark\_tertiary = Color(0xFFA5CCDF)  
val md\_theme\_dark\_onTertiary = Color(0xFF073543)  
val md\_theme\_dark\_tertiaryContainer = Color(0xFF244C5B)  
val md\_theme\_dark\_onTertiaryContainer = Color(0xFFC1E8FB)  
val md\_theme\_dark\_error = Color(0xFFFFB4AB)  
val md\_theme\_dark\_errorContainer = Color(0xFF93000A)  
val md\_theme\_dark\_onError = Color(0xFF690005)  
val md\_theme\_dark\_onErrorContainer = Color(0xFFFFDAD6)  
val md\_theme\_dark\_background = Color(0xFF191C1A)  
val md\_theme\_dark\_onBackground = Color(0xFFE1E3DF)  
val md\_theme\_dark\_surface = Color(0xFF191C1A)  
val md\_theme\_dark\_onSurface = Color(0xFFE1E3DF)  
val md\_theme\_dark\_surfaceVariant = Color(0xFF404943)  
val md\_theme\_dark\_onSurfaceVariant = Color(0xFFBFC9C2)  
val md\_theme\_dark\_outline = Color(0xFF8A938C)  
val md\_theme\_dark\_inverseOnSurface = Color(0xFF191C1A)  
val md\_theme\_dark\_inverseSurface = Color(0xFFE1E3DF)  
val md\_theme\_dark\_inversePrimary = Color(0xFF006C4C)  
val md\_theme\_dark\_shadow = Color(0xFF000000)  
val md\_theme\_dark\_surfaceTint = Color(0xFF6CDBAC)  
val md\_theme\_dark\_outlineVariant = Color(0xFF404943)  
val md\_theme\_dark\_scrim = Color(0xFF000000)

1. Откройте файл **Theme.kt** и замените его содержимое приведенным ниже кодом, чтобы добавить в тему новые цвета.

package com.example.woof.ui.theme  
  
import android.app.Activity  
import android.os.Build  
import android.view.View  
import androidx.compose.foundation.isSystemInDarkTheme  
import androidx.compose.material3.MaterialTheme  
import androidx.compose.material3.darkColorScheme  
import androidx.compose.material3.dynamicDarkColorScheme  
import androidx.compose.material3.dynamicLightColorScheme  
import androidx.compose.material3.lightColorScheme  
import androidx.compose.runtime.Composable  
import androidx.compose.runtime.SideEffect  
import androidx.compose.ui.graphics.Color  
import androidx.compose.ui.graphics.toArgb  
import androidx.compose.ui.platform.LocalContext  
import androidx.compose.ui.platform.LocalView  
import androidx.core.view.WindowCompat  
  
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,  
    onSecondary = md\_theme\_light\_onSecondary,  
    secondaryContainer = md\_theme\_light\_secondaryContainer,  
    onSecondaryContainer = md\_theme\_light\_onSecondaryContainer,  
    tertiary = md\_theme\_light\_tertiary,  
    onTertiary = md\_theme\_light\_onTertiary,  
    tertiaryContainer = md\_theme\_light\_tertiaryContainer,  
    onTertiaryContainer = md\_theme\_light\_onTertiaryContainer,  
    error = md\_theme\_light\_error,  
    errorContainer = md\_theme\_light\_errorContainer,  
    onError = md\_theme\_light\_onError,  
    onErrorContainer = md\_theme\_light\_onErrorContainer,  
    background = md\_theme\_light\_background,  
    onBackground = md\_theme\_light\_onBackground,  
    surface = md\_theme\_light\_surface,  
    onSurface = md\_theme\_light\_onSurface,  
    surfaceVariant = md\_theme\_light\_surfaceVariant,  
    onSurfaceVariant = md\_theme\_light\_onSurfaceVariant,  
    outline = md\_theme\_light\_outline,  
    inverseOnSurface = md\_theme\_light\_inverseOnSurface,  
    inverseSurface = md\_theme\_light\_inverseSurface,  
    inversePrimary = md\_theme\_light\_inversePrimary,  
    surfaceTint = md\_theme\_light\_surfaceTint,  
    outlineVariant = md\_theme\_light\_outlineVariant,  
    scrim = md\_theme\_light\_scrim,  
)  
  
  
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,  
    onSecondary = md\_theme\_dark\_onSecondary,  
    secondaryContainer = md\_theme\_dark\_secondaryContainer,  
    onSecondaryContainer = md\_theme\_dark\_onSecondaryContainer,  
    tertiary = md\_theme\_dark\_tertiary,  
    onTertiary = md\_theme\_dark\_onTertiary,  
    tertiaryContainer = md\_theme\_dark\_tertiaryContainer,  
    onTertiaryContainer = md\_theme\_dark\_onTertiaryContainer,  
    error = md\_theme\_dark\_error,  
    errorContainer = md\_theme\_dark\_errorContainer,  
    onError = md\_theme\_dark\_onError,  
    onErrorContainer = md\_theme\_dark\_onErrorContainer,  
    background = md\_theme\_dark\_background,  
    onBackground = md\_theme\_dark\_onBackground,  
    surface = md\_theme\_dark\_surface,  
    onSurface = md\_theme\_dark\_onSurface,  
    surfaceVariant = md\_theme\_dark\_surfaceVariant,  
    onSurfaceVariant = md\_theme\_dark\_onSurfaceVariant,  
    outline = md\_theme\_dark\_outline,  
    inverseOnSurface = md\_theme\_dark\_inverseOnSurface,  
    inverseSurface = md\_theme\_dark\_inverseSurface,  
    inversePrimary = md\_theme\_dark\_inversePrimary,  
    surfaceTint = md\_theme\_dark\_surfaceTint,  
    outlineVariant = md\_theme\_dark\_outlineVariant,  
    scrim = md\_theme\_dark\_scrim,  
)  
  
@Composable  
fun WoofTheme(  
    darkTheme: Boolean = isSystemInDarkTheme(),  
    // Dynamic color is available on Android 12+  
    dynamicColor: Boolean = false,  
    content: @Composable () -> Unit  
) {  
    val colorScheme = when {  
        dynamicColor && Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.S -> {  
            val context = LocalContext.current  
            if (darkTheme) dynamicDarkColorScheme(context) else dynamicLightColorScheme(context)  
        }  
  
        darkTheme -> DarkColors  
        else -> LightColors  
    }  
    val view = LocalView.current  
    if (!view.isInEditMode) {  
        SideEffect {  
            setUpEdgeToEdge(view, darkTheme)  
        }  
    }  
  
    MaterialTheme(  
        colorScheme = colorScheme,  
        shapes = Shapes,  
        typography = Typography,  
        content = content  
    )  
}  
  
/\*\*  
 \* Sets up edge-to-edge for the window of this [view]. The system icon colors are set to either  
 \* light or dark depending on whether the [darkTheme] is enabled or not.  
 \*/  
private fun setUpEdgeToEdge(view: View, darkTheme: Boolean) {  
    val window = (view.context as Activity).window  
    WindowCompat.setDecorFitsSystemWindows(window, false)  
    window.statusBarColor = Color.Transparent.toArgb()  
    val navigationBarColor = when {  
        Build.VERSION.SDK\_INT >= 29 -> Color.Transparent.toArgb()  
        Build.VERSION.SDK\_INT >= 26 -> Color(0xFF, 0xFF, 0xFF, 0x63).toArgb()  
        // Min sdk version for this app is 24, this block is for SDK versions 24 and 25  
        else -> Color(0x00, 0x00, 0x00, 0x50).toArgb()  
    }  
    window.navigationBarColor = navigationBarColor  
    val controller = WindowCompat.getInsetsController(window, view)  
    controller.isAppearanceLightStatusBars = !darkTheme  
    controller.isAppearanceLightNavigationBars = !darkTheme  
}

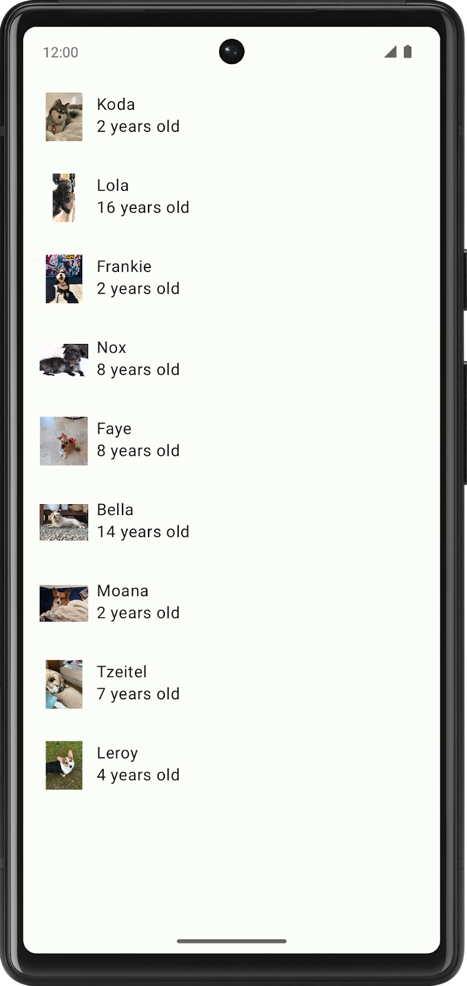
В WoofTheme()операторе colorScheme valuseswhen​

* Если dynamicColorэто правда и версия сборки S или выше, он проверяет, включено устройство darkThemeили нет.
* Если это темная тема, colorSchemeбудет установлено значение dynamicDarkColorScheme.
* Если его нет в темной теме, будет установлено значение dynamicLightColorScheme.
* Если приложение не используется dynamicColor, оно проверяет, включено ли ваше приложение darkTheme. Если да, то colorSchemeбудет установлено значение DarkColors.
* Если ни одно из этих значений не соответствует действительности, colorSchemeбудет установлено значение LightColors.

В файле, скопированном в **Theme.kt**dynamicColor , установлено значение false, а устройства, с которыми мы работаем, находятся в облегченном режиме, поэтому для него colorSchemeбудет установлено значение LightColors.

val colorScheme = when {  
       dynamicColor && Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.S -> {  
           val context = LocalContext.current  
           if (darkTheme) dynamicDarkColorScheme(context) else dynamicLightColorScheme(context)  
       }  
  
       darkTheme -> DarkColors  
       else -> LightColors  
   }

1. Перезапустите приложение и обратите внимание, что панель приложения автоматически изменила цвет.



Цветовое картографирование

Компоненты материала автоматически сопоставляются с цветовыми слотами. Другие ключевые компоненты пользовательского интерфейса, такие как плавающие кнопки действий, также по умолчанию имеют основной цвет. Это означает, что вам не нужно явно назначать цвет компоненту; он автоматически сопоставляется с цветовым слотом, когда вы устанавливаете цветовую тему в своем приложении. Вы можете переопределить это, явно задав цвет в коде. Подробнее о цветовых ролях читайте [здесь](https://m3.material.io/styles/color/the-color-system/color-roles) .

В этом разделе мы обернем элемент, Rowсодержащий DogIcon()и DogInformation(), чтобы Cardразличать цвета элементов списка с фоном.

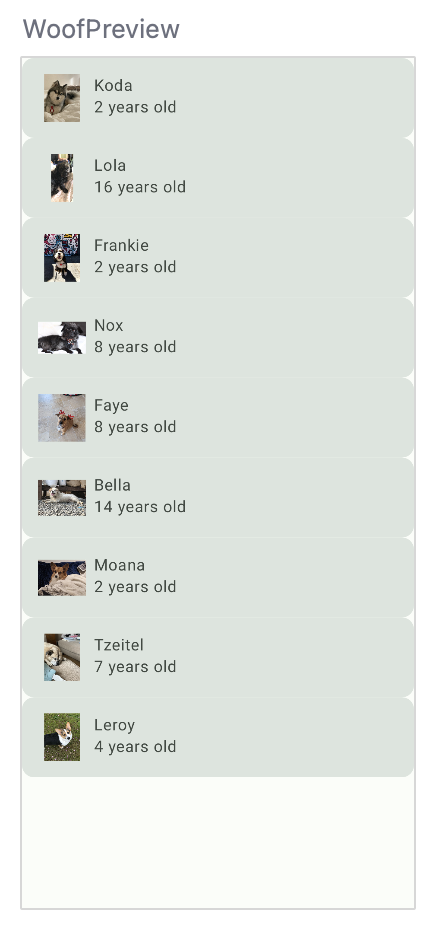
1. В DogItem()составной функции оберните Row()ее Card().

Card() {  
   Row(  
       modifier = modifier  
           .fillMaxWidth()  
           .padding(dimensionResource(id = R.dimen.padding\_small))  
   ) {  
       DogIcon(dog.imageResourceId)  
       DogInformation(dog.name, dog.age)  
   }  
}

1. Поскольку Cardтеперь это первый дочерний элемент, который можно компоновать DogItem(), передайте модификатор from DogItem()в Cardи обновите Rowмодификатор 's, указав новый экземпляр Modifier.

Card(modifier = modifier) {  
   Row(  
       modifier = Modifier  
           .fillMaxWidth()  
           .padding(dimensionResource(id = R.dimen.padding\_small))  
   ) {  
       DogIcon(dog.imageResourceId)  
       DogInformation(dog.name, dog.age)  
   }  
}

1. Взгляни на WoofPreview(). Элементы списка теперь автоматически изменили цвет из-за CardComposables. Цвета выглядят великолепно, но между элементами списка нет пробелов.



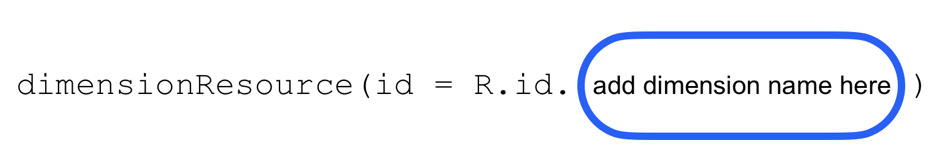
Файл размеров

Точно так же, как вы используете файл **strings.xml** для хранения строк в своем приложении, хорошей практикой является использование файла **dimens.xml** для хранения значений измерений. Это полезно, поскольку вам не нужно жестко кодировать значения, и при необходимости вы можете изменить их в одном месте.

Перейдите в **приложение** > **разрешения** > **значения** > **dimens.xml** и просмотрите файл. Он хранит значения измерений для padding\_small, padding\_mediumи image\_size. Эти размеры будут использоваться во всем приложении.

<resources>  
   <dimen name="padding\_small">8dp</dimen>  
   <dimen name="padding\_medium">16dp</dimen>  
   <dimen name="image\_size">64dp</dimen>  
</resources>

Чтобы добавить значение из файла **dimens.xml** , используйте правильный формат:

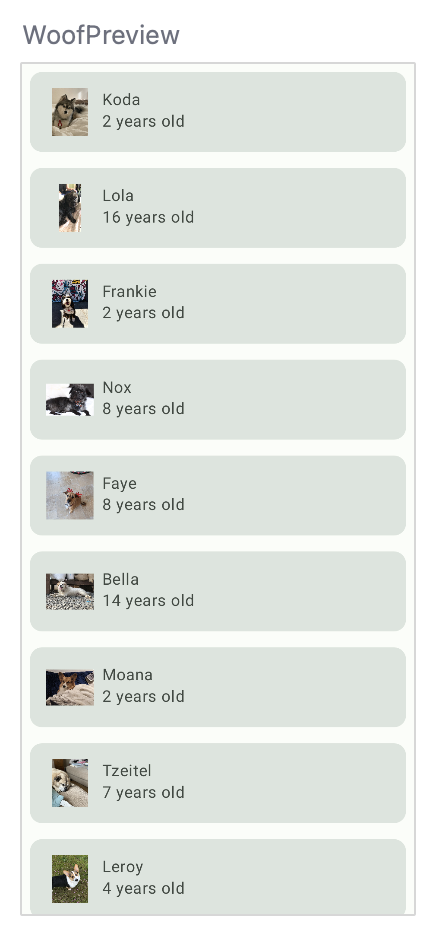


Например, чтобы добавить padding\_small, вам нужно передать .dimensionResource(id = R.dimen.*padding\_small*)

1. В WoofApp()добавьте modifierв padding\_smallвызов DogItem().

@Composable  
fun WoofApp() {  
    Scaffold { it ->  
        LazyColumn(contentPadding = it) {  
            items(dogs) {  
                DogItem(  
                    dog = it,  
                    modifier = Modifier.padding(dimensionResource(R.dimen.padding\_small))  
                )  
            }  
        }  
    }  
}

В WoofPreview(), теперь есть больше определений между элементами списка.



Темная тема

В системе Android есть возможность переключить ваше устройство на темную тему. В темной теме используются более темные, более приглушенные цвета и:

* Может значительно снизить энергопотребление (в зависимости от технологии экрана устройства).
* Улучшает видимость для пользователей со слабым зрением и тех, кто чувствителен к яркому свету.
* Облегчает использование устройства в условиях низкой освещенности.

Ваше приложение может выбрать [Force Dark](https://developer.android.com/guide/topics/ui/look-and-feel/darktheme#force-dark) , что означает, что система реализует для вас темную тему. Однако для ваших пользователей будет лучше, если вы реализуете темную тему, чтобы вы сохранили полный контроль над темой приложения.

При выборе собственной темной темы важно учитывать, что цвета темной темы должны соответствовать [стандартам контрастности специальных возможностей](https://webaim.org/resources/contrastchecker/) . В темных темах используется темный цвет поверхности с ограниченным количеством цветовых акцентов.

Посмотреть темную тему в предварительном просмотре

На предыдущем шаге вы уже добавили цвета для темной темы. Чтобы увидеть темную тему в действии, вы добавите еще один Preview Composable в **MainActivity.kt** . Таким образом, когда вы меняете макет пользовательского интерфейса в своем коде, вы можете видеть, как предварительный просмотр светлой и темной темы выглядят одновременно.

1. В разделе WoofPreview()создайте новую функцию с именем WoofDarkThemePreview()и добавьте к ней аннотации @Previewи @Composable.

@Preview  
@Composable  
fun WoofDarkThemePreview() {  
  
}

1. Внутри DarkThemePreview()добавьте WoofTheme(). Без добавления WoofTheme()вы не увидите ни одного стиля, который мы добавили в приложение. Установите darkThemeдля параметра значение **true.**

@Preview  
@Composable  
fun WoofDarkThemePreview() {  
   WoofTheme(darkTheme = true) {  
  
   }  
}

1. Позвоните WoofApp()внутри WoofTheme().

@Preview  
@Composable  
fun WoofDarkThemePreview() {  
   WoofTheme(darkTheme = true) {  
       WoofApp()  
   }  
}

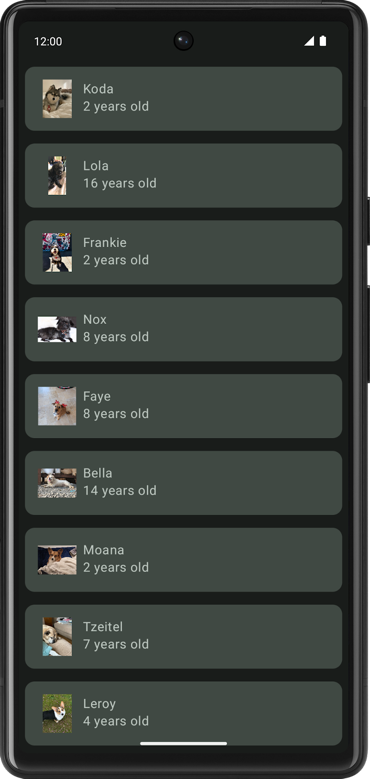
Теперь прокрутите вниз панель **«Дизайн»** , чтобы увидеть приложение в темной теме, включая более темный фон приложения/элемента списка и более светлый текст. Сравните различия между темной и светлой темами.

|  |  |
| --- | --- |
| **Темная тема** | **Светлая тема** |
|  |  |

Просмотр темной темы на вашем устройстве или в эмуляторе

Чтобы просмотреть свое приложение в темной теме на эмуляторе или физическом устройстве:

1. Зайдите в приложение **«Настройки»** на устройстве.
2. Найдите **темную тему** и щелкните ее.
3. Включите **темную тему** .
4. Снова откройте приложение **Woof** , и оно будет в **темной теме** .



В этой кодовой лаборатории основное внимание уделяется светлой теме, поэтому, прежде чем продолжить работу с приложением, отключите темную тему.

1. Зайдите в приложение **«Настройки»** на устройстве.
2. Выберите **«Дисплей»** .
3. Отключите **темную тему** .

Сравните, как приложение выглядело в начале раздела и сейчас. Элементы списка и текст стали более четкими, а цветовая схема стала более визуально привлекательной.

|  |  |  |
| --- | --- | --- |
| **Без цвета** | **С цветом (светлая тема)** | **С цветом (темная тема)** |
|  |  |  |

Динамический цвет

В Material 3 особое внимание уделяется персонализации пользователей: новая функция Material 3 — это динамический цвет, который создает тему для вашего приложения на основе обоев пользователя. Таким образом, если пользователь любит зеленый цвет и у него синий фон телефона, его приложение Woof также будет синим, чтобы отразить это. Динамические темы доступны только на некоторых устройствах под управлением Android 12 и более поздних версий.

Пользовательскую тему можно использовать для приложений с яркими фирменными цветами, а также ее необходимо реализовать для устройств, которые не поддерживают динамическое оформление тем, чтобы ваше приложение по-прежнему оставалось тематичным.

1. Чтобы включить динамический цвет, откройте **Theme.kt** , перейдите к WoofTheme()компоненту и установите dynamicColorдля параметра значение **true** .

@Composable  
fun WoofTheme(  
   darkTheme: Boolean = isSystemInDarkTheme(),  
   dynamicColor: Boolean = true,  
   content: @Composable () -> Unit  
)

1. Чтобы изменить фон устройства или эмулятора, перейдите в **«Настройки»** и выполните поиск **«Обои»** .
2. Измените обои на цвет или набор цветов.
3. Перезапустите приложение, чтобы увидеть динамическую тему (обратите внимание, что ваше устройство или эмулятор должны быть Android 12+, чтобы увидеть динамический цвет), не стесняйтесь экспериментировать с разными обоями!



1. Эта лаборатория кода ориентирована на пользовательские темы, поэтому отключите ее, dynamicColorпрежде чем двигаться дальше.

@Composable  
fun WoofTheme(  
   darkTheme: Boolean = isSystemInDarkTheme(),  
   dynamicColor: Boolean = false,  
   content: @Composable () -> Unit  
)

[5. Добавьте форму](https://developer.android.com/codelabs/basic-android-kotlin-compose-material-theming?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-3-pathway-3%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-material-theming#4)

Применение формы может очень сильно изменить внешний вид составного объекта. Формы направляют внимание, определяют компоненты, сообщают о состоянии и выражают бренд.

Многие фигуры определяются с помощью [RoundedCornerShape](https://developer.android.com/reference/kotlin/androidx/compose/foundation/shape/RoundedCornerShape" \t "_blank), который описывает прямоугольник с закругленными углами. Переданное число определяет, насколько скруглены углы. Если RoundedCornerShape(0.dp)используется, прямоугольник не имеет закругленных углов; если RoundedCornerShape(50.dp)используется, углы будут полностью круглыми.

|  |  |  |
| --- | --- | --- |
| 0.дп | 25.дп | 50.дп |
| Элемент списка Гав с формированием | Элемент списка Гав с формированием | Элемент списка Гав с формированием |

Вы также можете дополнительно настроить формы, добавив разные проценты закругления в каждом углу. Очень интересно играть с фигурами!

|  |  |  |
| --- | --- | --- |
| Вверху слева: 50.dp Внизу слева: 25.dp Вверху справа: 0.dp Внизу справа: 15.dp | Вверху слева: 15.dp Внизу слева: 50.dp Вверху справа: 50.dp Внизу справа: 15.dp | Вверху слева: 0.dp Внизу слева: 50.dp Вверху справа: 0.dp Внизу справа: 50.dp |
| Элемент списка Гав с формированием | Элемент списка Гав с формированием | Элемент списка Гав с формированием |

Файл **Shape.kt** используется для определения формы компонентов в Compose. Существует три типа компонентов: малые, средние и крупные. В этом разделе вы измените Cardкомпонент, который определяется как mediumразмер. Компоненты сгруппированы по [категориям форм](https://m3.material.io/styles/shape/shape-scale-tokens#b09934f1-1b0f-4ce4-ade6-4a1f138add6c) в зависимости от их размера.

В этом разделе вы придадите изображению собаки форму круга и измените форму элемента списка.

Сформируйте из изображения собаки круг.

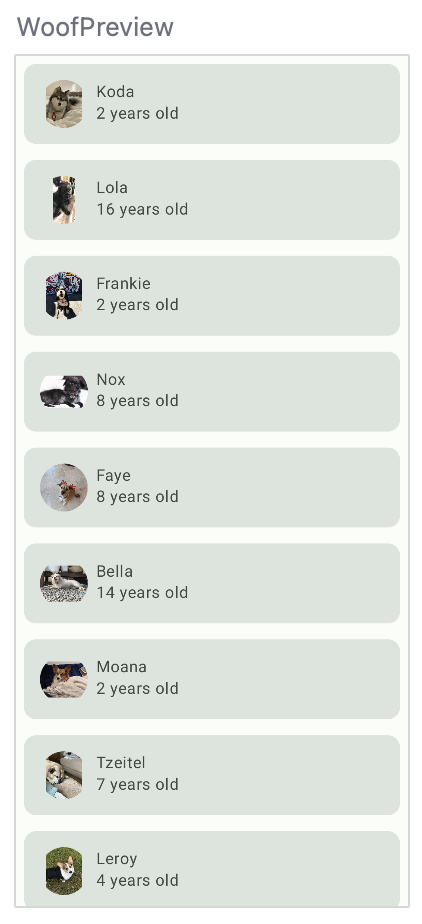
1. Откройте файл **Shape.kt** и обратите внимание, что для параметра small установлено значение RoundedCornerShape(50.dp). Это будет использоваться для придания изображению формы круга.

val Shapes = Shapes(  
   small = RoundedCornerShape(50.dp),  
)

1. Откройте **MainActivity.kt** . В DogIcon()добавьте атрибут [clip](https://developer.android.com/reference/kotlin/androidx/compose/ui/Modifier#(androidx.compose.ui.Modifier).clip(androidx.compose.ui.graphics.Shape))в modifier; Imageэто закрепит изображение в форме. Пройдите в MaterialTheme.shapes.small.

import androidx.compose.ui.draw.clip  
  
@Composable  
fun DogIcon(  
   @DrawableRes dogIcon: Int,  
   modifier: Modifier = Modifier  
) {  
   Image(  
       modifier = modifier  
           .size(dimensionResource(id = R.dimen.image\_size))  
           .padding(dimensionResource(id = R.dimen.padding\_small))  
           .clip(MaterialTheme.shapes.small),

Когда вы посмотрите на WoofPreview(), вы заметите, что значки собак имеют круглую форму! Однако некоторые фотографии обрезаны по бокам и не выглядят полностью круглыми.



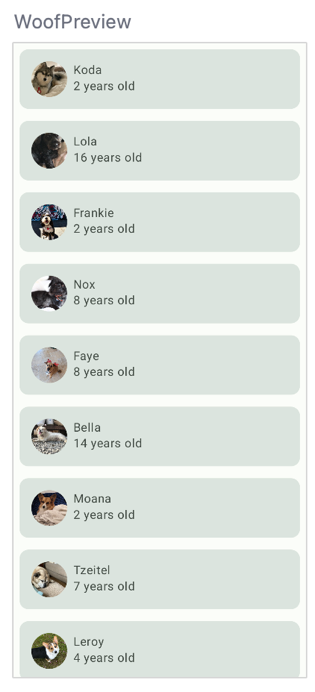
1. Чтобы сделать все фотографии круглыми, добавьте атрибут [ContentScale](https://developer.android.com/reference/kotlin/androidx/glance/layout/ContentScale" \t "_blank)и [Crop](https://developer.android.com/reference/kotlin/androidx/glance/layout/ContentScale" \l "Crop()" \t "_blank); это обрезает изображение по размеру. Обратите внимание, что contentScaleэто атрибут Image, а не часть modifier.

import androidx.compose.ui.layout.ContentScale  
  
@Composable  
fun DogIcon(  
   @DrawableRes dogIcon: Int,  
   modifier: Modifier = Modifier  
) {  
   Image(  
       modifier = modifier  
           .size(dimensionResource(id = R.dimen.image\_size))  
           .padding(dimensionResource(id = R.dimen.padding\_small))  
           .clip(MaterialTheme.shapes.small),  
       contentScale = ContentScale.Crop,

Это полный DogIcon()Composable.

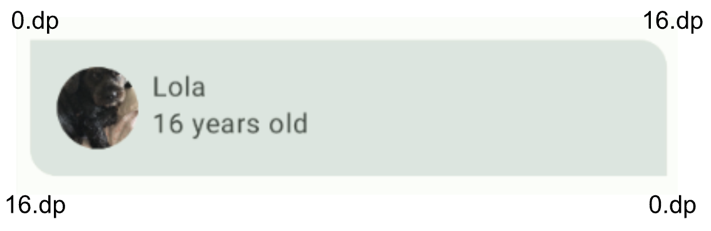
@Composable  
fun DogIcon(  
    @DrawableRes dogIcon: Int,  
    modifier: Modifier = Modifier  
) {  
    Image(  
        modifier = modifier  
            .size(dimensionResource(R.dimen.image\_size))  
            .padding(dimensionResource(R.dimen.padding\_small))  
            .clip(MaterialTheme.shapes.small),  
        contentScale = ContentScale.Crop,  
        painter = painterResource(dogIcon),  
  
        // Content Description is not needed here - image is decorative, and setting a null content  
        // description allows accessibility services to skip this element during navigation.  
  
        contentDescription = null  
    )  
}

Теперь в WoofPreview()значки имеют круглую форму.



Добавить фигуру в элемент списка

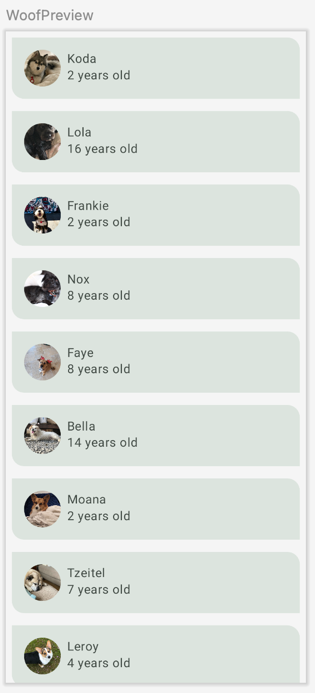
В этом разделе вы добавите фигуру к элементу списка. Элемент списка уже отображается через файл Card. A Card— это поверхность, которая может содержать один составной объект и содержит варианты оформления. Украшение можно добавить через рамку, форму и многое другое. В этом разделе вы будете использовать , Cardчтобы добавить фигуру к элементу списка.



1. Откройте файл **Shape.kt** . A Card— средний компонент, поэтому вы добавляете средний параметр объекта Shapes. В этом приложении верхний правый и нижний левый углы элемента списка не должны быть полностью круглыми. Для этого перейдите 16.dpк mediumатрибуту.

medium = RoundedCornerShape(bottomStart = 16.dp, topEnd = 16.dp)

Поскольку Cardпо умолчанию уже используется средняя форма, вам не нужно явно задавать для нее среднюю форму. Посмотрите **превью** и увидите новую форму Card!



Если вы вернетесь к файлу **Theme.kt**WoofTheme() и посмотрите на него MaterialTheme(), вы увидите, что shapesдля атрибута установлено значение, Shapes valкоторое вы только что обновили.

MaterialTheme(  
   colors = colors,  
   typography = Typography,  
   shapes = Shapes,  
   content = content  
)

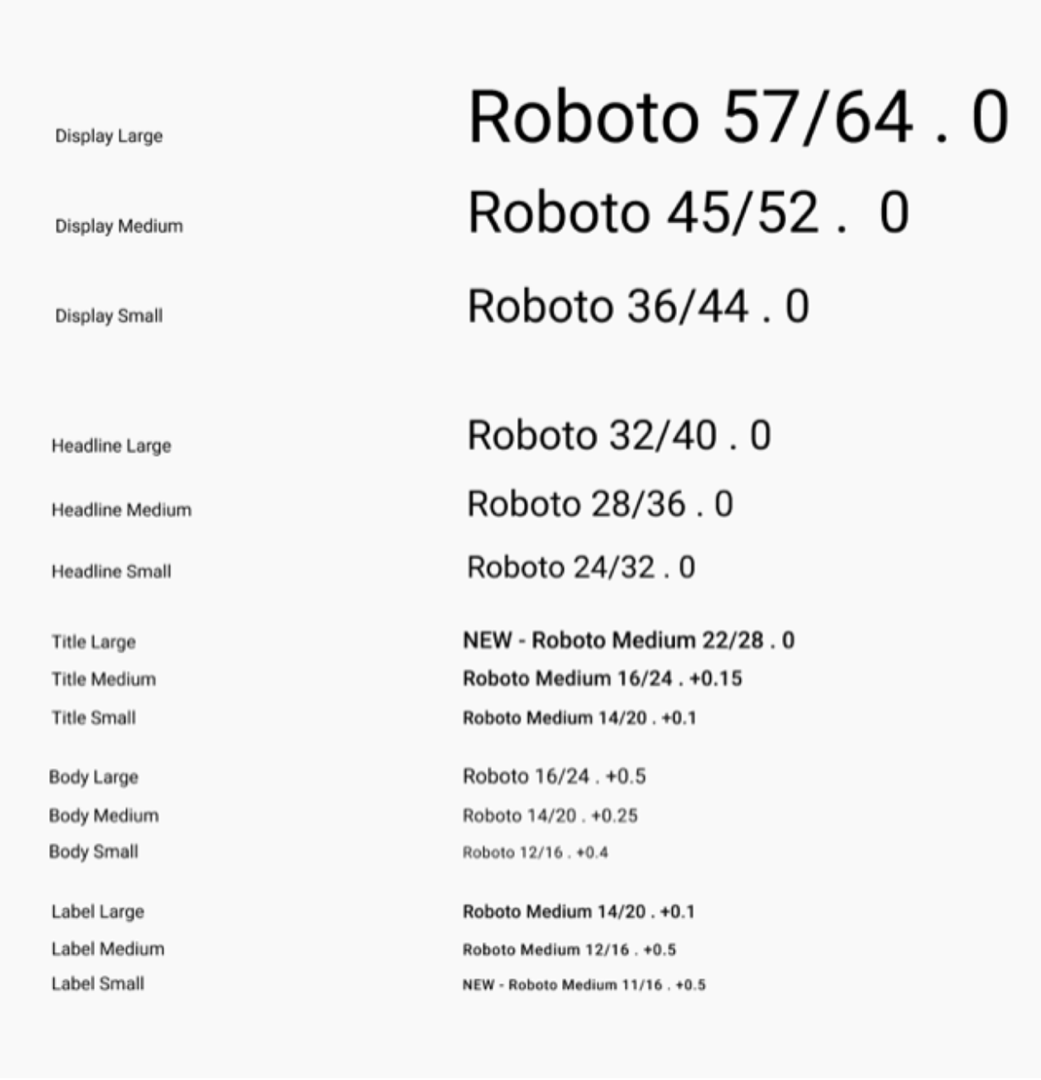
Ниже представлен параллельный вид элементов списка до и после формирования. Обратите внимание, насколько визуально привлекательным стало приложение с добавлением форм.

|  |  |
| --- | --- |
| **Без формирования** | **С формированием** |
|  |  |

## [6. Добавьте типографику](https://developer.android.com/codelabs/basic-android-kotlin-compose-material-theming?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-3-pathway-3%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-material-theming#5)

## Масштаб типа Material Design

Шкала шрифтов — это набор стилей шрифтов, которые можно использовать во всем приложении, обеспечивая гибкий, но последовательный стиль. Шкала типов Material [Design](https://m3.material.io/styles/typography/type-scale-tokens) включает пятнадцать стилей шрифтов, которые поддерживаются системой типов. Именование и группировка были упрощены: отображение, заголовок, заголовок, тело и метка с большими, средними и маленькими размерами для каждого. Вам нужно использовать эти варианты только в том случае, если вы хотите настроить свое приложение. Если вы не знаете, что установить для каждой категории масштаба шрифта, знайте, что существует типографский масштаб по умолчанию, который вы можете использовать.



Шкала шрифтов содержит многократно используемые категории текста, каждая из которых имеет предполагаемое применение и значение.

### **Отображать**

Поскольку текст является самым крупным на экране, стили отображения зарезервированы для короткого важного текста или цифр. Лучше всего они работают на больших экранах.

### **Заголовок**

Заголовки лучше всего подходят для короткого, выделенного текста на небольших экранах. Эти стили могут быть полезны для обозначения основных отрывков текста или важных областей контента.

### **Заголовок**

Заголовки меньше, чем стили заголовков, и их следует использовать для текста со средним выделением, который остается относительно коротким.

### **Тело**

Стили тела используются для длинных фрагментов текста в вашем приложении.

### **Этикетка**

Стили меток — это меньшие по размеру утилитарные стили, используемые для таких вещей, как текст внутри компонентов или для очень маленького текста в теле содержимого, например подписей.

## Шрифты

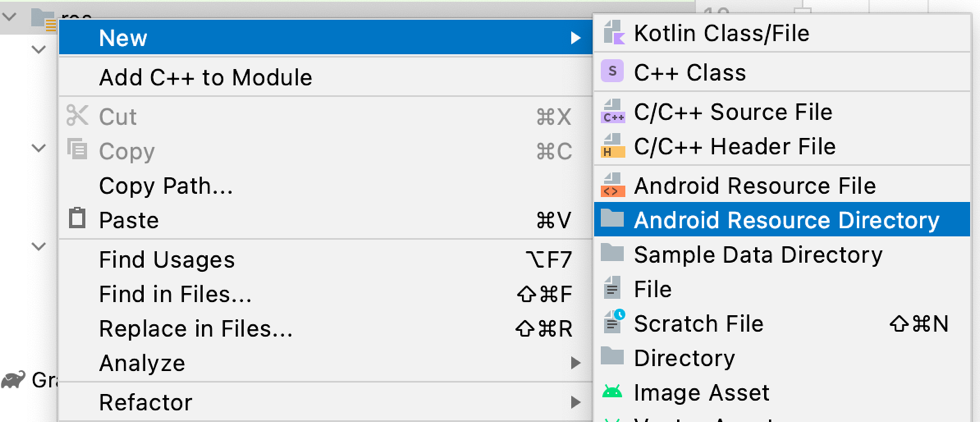
Платформа Android предоставляет множество шрифтов, но вы можете настроить свое приложение, используя шрифт, не предусмотренный по умолчанию. Пользовательские шрифты могут добавить индивидуальности и использоваться для брендинга.

В этом разделе вы добавите пользовательские шрифты под названием **Abril Fatface** , **Montserrat Bold** и **Montserrat Regular.** Вы будете использовать заголовки displayLarge и displayMedium, а также текст bodyLarge из системы типов материалов и добавить их к тексту в своем приложении.

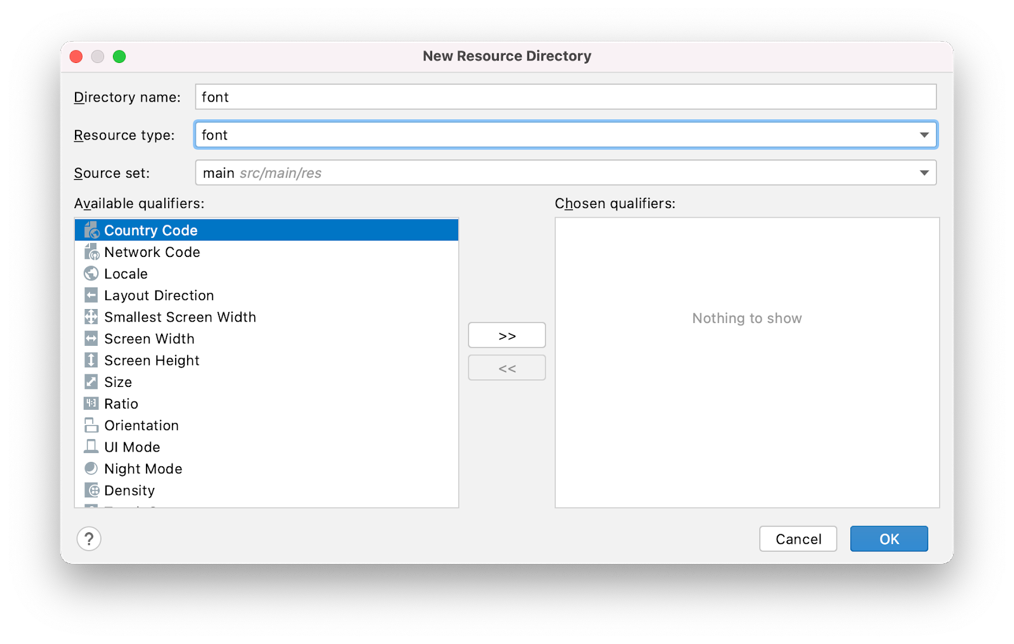
### **Создайте каталог ресурсов Android шрифта.**

Прежде чем добавлять шрифты в свое приложение, вам необходимо добавить каталог шрифтов.

1. В представлении проекта Android Studio щелкните правой кнопкой мыши папку **res** .
2. Выберите **«Создать»** > **«Каталог ресурсов Android»** .



1. **Назовите шрифт** «Каталог» , установите тип «Ресурс» в качестве **шрифта** и нажмите **«ОК»** .



1. Откройте новый каталог ресурсов шрифтов, расположенный по адресу **res > font** .

### **Загрузите пользовательские шрифты**

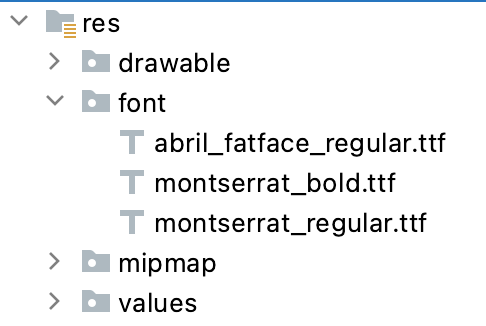
Поскольку вы используете шрифты, не предусмотренные платформой Android, вам необходимо загрузить собственные шрифты.

1. Перейдите на <https://fonts.google.com/> .
2. Найдите [**Монтсеррат**](https://fonts.google.com/specimen/Montserrat?query=mon) и нажмите « **Загрузить семью»** .
3. Разархивируйте zip-файл.
4. Откройте скачанную папку **Монтсеррат** . В папке **static** найдите **Montserrat-Bold.ttf** и **Montserrat-Regular.ttf** ( **ttf** означает шрифт TrueType и формат файлов шрифтов). Выберите оба шрифта и перетащите их в каталог ресурсов шрифтов вашего проекта в Android Studio.



1. В папке шрифтов переименуйте **Montserrat-Bold.ttf** в **montserrat\_bold.ttf** и переименуйте **Montserrat-Regular.ttf** в **montserrat\_regular.ttf** .
2. Найдите **[Abril Fatface](https://fonts.google.com/specimen/Abril+Fatface?query=abril" \t "_blank)** и нажмите « **Загрузить семью»** .
3. Откройте скачанную папку **Abril\_Fatface** . Выберите **AbrilFatface-Regular.ttf** и перетащите его в каталог ресурсов шрифтов.
4. В папке шрифтов переименуйте **Abril\_Fatface\_Regular.ttf** в **abril\_fatface\_regular.ttf** .

Вот как должен выглядеть каталог ресурсов шрифтов в вашем проекте с тремя файлами пользовательских шрифтов:



### **Инициализировать шрифты**

1. В окне проекта откройте **ui.theme** > **Type.kt.** Инициализируйте загруженные шрифты под операторами импорта и над файлом Typography val. Сначала инициализируйте **Abril Fatface** , установив его равным FontFamilyи передав ему Fontфайл шрифта abril\_fatface\_regular.

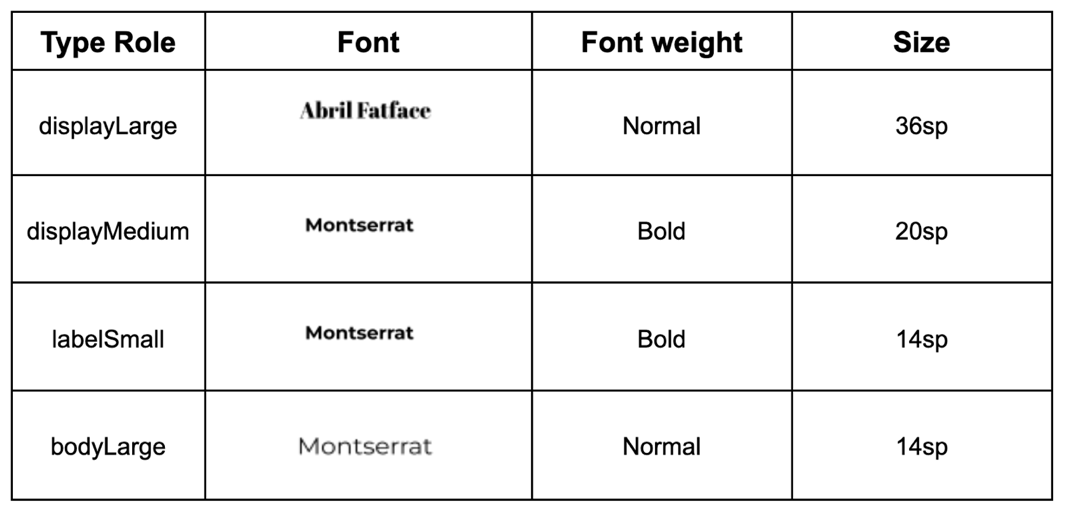
​​import androidx.compose.ui.text.font.Font  
import androidx.compose.ui.text.font.FontFamily  
import com.example.woof.R  
  
val AbrilFatface = FontFamily(  
   Font(R.font.abril\_fatface\_regular)  
)

1. Инициализируйте **Montserrat** под **Abril Fatface** , установив его равным FontFamilyи передав Fontфайл шрифта montserrat\_regular. Для montserrat\_boldтакже включите FontWeight.Bold. Даже если вы передаете версию файла шрифта, выделенную полужирным шрифтом, Compose не знает, что файл выделен полужирным шрифтом, поэтому вам необходимо явно связать файл с FontWeight.Bold.

import androidx.compose.ui.text.font.FontWeight  
  
val AbrilFatface = FontFamily(  
   Font(R.font.abril\_fatface\_regular)  
)  
  
val Montserrat = FontFamily(  
   Font(R.font.montserrat\_regular),  
   Font(R.font.montserrat\_bold, FontWeight.Bold)  
)

Затем вы устанавливаете различные типы заголовков для только что добавленных шрифтов. Объект Typographyимеет параметры для 13 различных гарнитур, описанных выше. Вы можете определить столько, сколько вам нужно. В этом приложении мы установим displayLarge, displayMediumи bodyLarge. В следующей части этого приложения вы используете labelSmall, поэтому добавите его сюда.

Ниже приведена таблица, в которой указаны шрифт, насыщенность и размер каждого добавляемого заголовка.



1. Для displayLargeатрибута установите его равным TextStyleи заполните fontFamily, fontWeightи fontSizeинформацией из таблицы выше. Это означает, что весь установленный текст displayLargeбудет иметь шрифт **Abril Fatface**fontSize с нормальной толщиной шрифта и расширением 36.sp.

Повторите этот процесс для displayMedium, labelSmallи bodyLarge.

import androidx.compose.ui.text.TextStyle  
import androidx.compose.ui.unit.sp  
  
  
val Typography = Typography(  
   displayLarge = TextStyle(  
       fontFamily = AbrilFatface,  
       fontWeight = FontWeight.Normal,  
       fontSize = 36.sp  
   ),  
   displayMedium = TextStyle(  
       fontFamily = Montserrat,  
       fontWeight = FontWeight.Bold,  
       fontSize = 20.sp  
   ),  
   labelSmall = TextStyle(  
       fontFamily = Montserrat,  
       fontWeight = FontWeight.Bold,  
       fontSize = 14.sp  
   ),  
   bodyLarge = TextStyle(  
       fontFamily = Montserrat,  
       fontWeight = FontWeight.Normal,  
       fontSize = 14.sp  
   )  
)

Если вы зайдете в файл **Theme.kt**WoofTheme() и посмотрите на параметр MaterialTheme(), он typographyбудет равен тому Typography val, который вы только что обновили.

MaterialTheme(  
   colors = colors,  
   typography = Typography,  
   shapes = Shapes,  
   content = content  
)

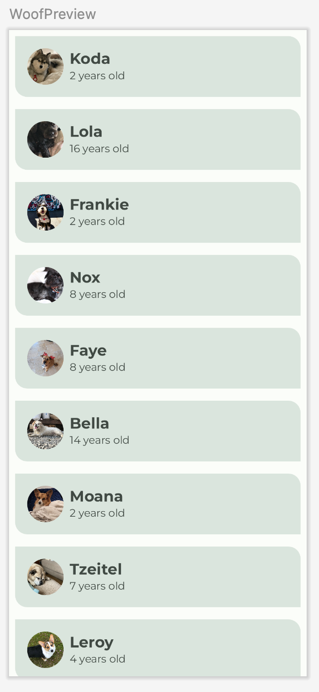
### **Добавьте типографику в текст приложения**

Теперь вы добавите типы заголовков к каждому экземпляру текста в приложении.

1. Добавьте displayMediumстиль, dogNameпотому что это короткая и важная информация. Добавьте его bodyLargeв качестве стиля, dogAgeпотому что он хорошо работает с текстом меньшего размера.

@Composable  
fun DogInformation(  
   @StringRes dogName: Int,  
   dogAge: Int,  
   modifier: Modifier = Modifier  
) {  
   Column(modifier = modifier) {  
       Text(  
           text = stringResource(dogName),  
           style = MaterialTheme.typography.displayMedium,  
           modifier = Modifier.padding(top = dimensionResource(id = R.dimen.padding\_small))  
       )  
       Text(  
           text = stringResource(R.string.years\_old, dogAge),  
           style = MaterialTheme.typography.bodyLarge  
       )  
   }  
}

1. Теперь в файле WoofPreview()имя собаки отображается жирным шрифтом **Montserrat**20.sp , а возраст собаки отображается обычным шрифтом **Montserrat**14.sp в формате .



Ниже представлен параллельный обзор элементов списка до и после добавления типографики. Обратите внимание на разницу шрифтов между именем собаки и возрастом собаки.

|  |  |
| --- | --- |
| **Без типографики** | **С типографикой** |
|  |  |

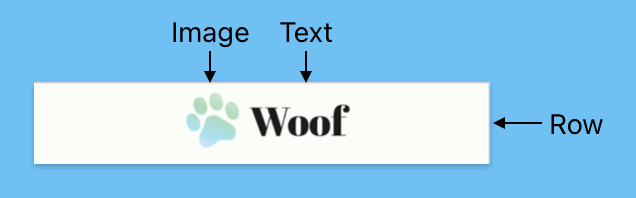
[7. Добавьте верхнюю панель](https://developer.android.com/codelabs/basic-android-kotlin-compose-material-theming?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-3-pathway-3%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-material-theming#6)

A [Scaffold](https://developer.android.com/reference/kotlin/androidx/compose/material3/package-summary" \l "Scaffold(androidx.compose.ui.Modifier,kotlin.Function0,kotlin.Function0,kotlin.Function0,kotlin.Function0,androidx.compose.material3.FabPosition,androidx.compose.ui.graphics.Color,androidx.compose.ui.graphics.Color,androidx.compose.foundation.layout.WindowInsets,kotlin.Function1)" \t "_blank)— это макет, в котором предусмотрены слоты для различных компонентов и элементов экрана, таких как Image, Rowили Column. A Scaffoldтакже предоставляет слот для [TopAppBar](https://developer.android.com/reference/kotlin/androidx/compose/material3/package-summary" \l "TopAppBar(kotlin.Function0,androidx.compose.ui.Modifier,kotlin.Function0,kotlin.Function1,androidx.compose.foundation.layout.WindowInsets,androidx.compose.material3.TopAppBarColors,androidx.compose.material3.TopAppBarScrollBehavior)" \t "_blank), который вы будете использовать в этом разделе.

A TopAppBarможно использовать для многих целей, но в данном случае вы будете использовать его для брендинга и придания индивидуальности своему приложению. Существует четыре различных типа TopAppBar: центральный, маленький, средний и большой. В этой кодовой лаборатории вы реализуете центральную верхнюю панель приложения. Вы создадите составной объект, похожий на скриншот ниже, и вставите его в topBarраздел файла Scaffold.



Для этого приложения наша верхняя панель состоит из Rowизображения логотипа и текста названия приложения. На логотипе изображена милая градиентная лапа и название приложения!



Добавьте изображение и текст на верхнюю панель

1. В **MainActivity.kt** создайте составной объект, вызываемый WoofTopAppBar()с необязательным modifier.

@Composable  
fun WoofTopAppBar(modifier: Modifier = Modifier) {  
    
}

1. [Scaffold](https://developer.android.com/jetpack/compose/layouts/material#scaffold)поддерживает contentWindowInsetsпараметр, который может помочь указать вставки для содержимого scaffold. [WindowInsets](https://developer.android.com/reference/android/view/WindowInsets" \t "_blank)— это части вашего экрана, где ваше приложение может пересекаться с системным пользовательским интерфейсом, они должны быть переданы в слот контента через параметры PaddingValues. Подробнее читайте [здесь](https://developer.android.com/develop/ui/views/layout/insets) .

Значение contentWindowInsetsпередается в LazyColumnкачестве contentPadding.

@Composable  
fun WoofApp() {  
    Scaffold { it ->  
        LazyColumn(contentPadding = it) {  
            items(dogs) {  
                DogItem(  
                    dog = it,  
                    modifier = Modifier.padding(dimensionResource(R.dimen.padding\_small))  
                )  
            }  
        }  
    }  
}

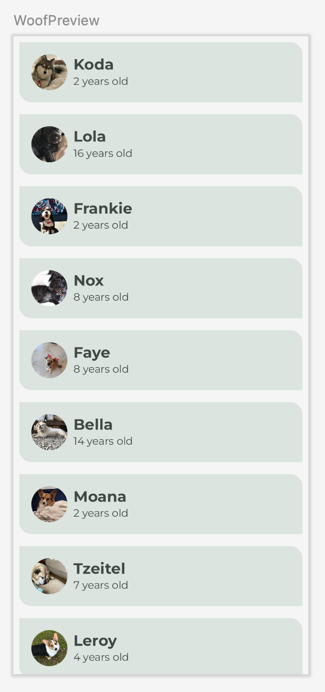
1. Внутри Scaffoldдобавьте topBarатрибут и установите для него значение WoofTopAppBar().

Scaffold(  
   topBar = {  
       WoofTopAppBar()  
   }  
)

Ниже показано, как WoofApp()будет выглядеть компоновка:

@Composable  
fun WoofApp() {  
    Scaffold(  
        topBar = {  
            WoofTopAppBar()  
        }  
    ) { it ->  
        LazyColumn(contentPadding = it) {  
            items(dogs) {  
                DogItem(  
                    dog = it,  
                    modifier = Modifier.padding(dimensionResource(R.dimen.padding\_small))  
                )  
            }  
        }  
    }  
}

Ничего не изменилось, WoofPreview()потому что в WoofTopAppBar(). Давайте изменим это!



1. Внутри WoofTopAppBar() Composableдобавьте CenterAlignedTopAppBar()и установите параметр модификатора в соответствии с модификатором, переданным в WoofTopAppBar().

import androidx.compose.material3.CenterAlignedTopAppBar  
  
@Composable  
fun WoofTopAppBar(modifier: Modifier = Modifier) {  
   CenterAlignedTopAppBar(  
       modifier = modifier  
   )  
}

1. В качестве параметра title передайте a, Rowкоторый будет содержать символы Imageи .TextCenterAlignedTopAppBar

@Composable  
fun WoofTopAppBar(modifier: Modifier = Modifier){  
   CenterAlignedTopAppBar(  
       title = {  
           Row() {  
                
           }  
       },  
       modifier = modifier  
   )  
}

1. Добавьте логотип Imageв файл Row.

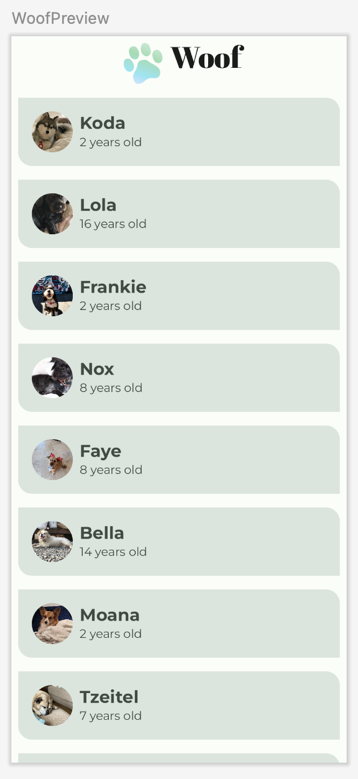
* Установите размер изображения modifierкак image\_sizeв dimens.xmlфайле и отступы как padding\_smallиз dimens.xmlфайла.
* Используйте painter, чтобы установить Imageкак ic\_woof\_logoиз папки с возможностью рисования.
* Установите contentDescriptionзначение **null** . В этой ситуации логотип приложения не добавляет никакой смысловой информации для пользователей с ослабленным зрением, поэтому нам не нужно добавлять описание контента.

Row() {  
   Image(  
       modifier = Modifier  
           .size(dimensionResource(id = R.dimen.image\_size))  
           .padding(dimensionResource(id = R.dimen.padding\_small)),  
       painter = painterResource(R.drawable.ic\_woof\_logo),  
       contentDescription = null  
   )  
}

1. Затем добавьте TextComposable внутри RowпослеImage.

* Используйте *stringResource()*для установки значения app\_name. При этом текст будет соответствовать имени приложения, которое хранится в формате strings.xml.
* Установите стиль текста, displayLargeпоскольку имя приложения представляет собой короткий и важный текст.

Text(  
   text = stringResource(R.string.app\_name),  
   style = MaterialTheme.typography.displayLarge  
)



Вот что показано на рисунке WoofPreview(): оно выглядит немного необычно, потому что значок и текст не выровнены по вертикали.

1. Чтобы это исправить, добавьте verticalAlignmentпараметр-значение в Rowи установите его равным Alignment.CenterVertically.

import androidx.compose.ui.Alignment  
  
Row(  
   verticalAlignment = Alignment.CenterVertically  
)



Это выглядит намного лучше!

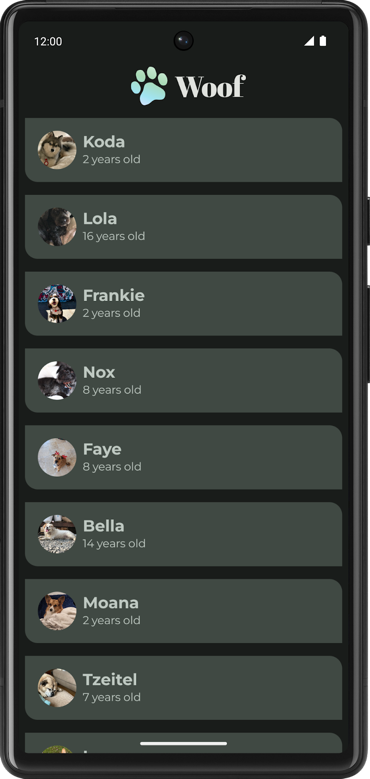
Это полная WoofTopAppBar()композиция:

@Composable  
fun WoofTopAppBar(modifier: Modifier = Modifier) {  
   CenterAlignedTopAppBar(  
       title = {  
           Row(  
               verticalAlignment = Alignment.CenterVertically  
           ) {  
               Image(  
                   modifier = Modifier  
                       .size(dimensionResource(id = R.dimen.image\_size))  
                       .padding(dimensionResource(id = R.dimen.padding\_small)),  
                   painter = painterResource(R.drawable.ic\_woof\_logo),  
  
                   contentDescription = null  
               )  
               Text(  
                   text = stringResource(R.string.app\_name),  
                   style = MaterialTheme.typography.displayLarge  
               )  
           }  
       },  
       modifier = modifier  
   )  
}

Запустите приложение и полюбуйтесь, как прекрасно оно TopAppBarобъединяет все приложения.

|  |  |
| --- | --- |
| **Без верхней панели приложений** | **С верхней панелью приложений** |
|  |  |

Теперь взгляните на финальную версию приложения в темной теме!



Поздравляем, вы дошли до конца кодовой лаборатории!

[9. Заключение](https://developer.android.com/codelabs/basic-android-kotlin-compose-material-theming?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-3-pathway-3%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-material-theming#8)

Вы только что создали свое первое приложение Material! Вы создали собственную цветовую палитру для светлых и темных тем, создали формы для различных компонентов, загрузили шрифты и добавили их в приложение, а также создали красивую верхнюю панель, чтобы связать все это воедино. Воспользуйтесь навыками, полученными в этой лаборатории, и меняйте цвета, формы и типографику, чтобы сделать приложения полностью своими!

Краткое содержание

* Material Theming позволяет использовать Material Design в вашем приложении с инструкциями по настройке цветов, типографики и форм.
* **В файле Theme.kt** определяется тема с помощью составного имени [your app name]+Theme()— WoofTheme()в случае этого приложения. В рамках этой функции MaterialThemeобъект устанавливает color, typography, shapesи contentприложения.
* **Color.kt** — это место, где вы перечисляете цвета, которые вы используете в приложении. Затем в **Theme.kt** вы назначаете цвета LightColorPaletteопределенным DarkColorPaletteслотам. Не все слоты необходимо назначать.
* Ваше приложение может выбрать [Force Dark](https://developer.android.com/guide/topics/ui/look-and-feel/darktheme#force-dark) , что означает, что система реализует для вас темную тему. Однако для ваших пользователей будет лучше, если вы реализуете темную тему, чтобы сохранить полный контроль над темой приложения.
* **Shape.kt** — это место, где вы определяете формы своего приложения. Существует три размера фигур (маленький, средний, большой), и вы можете указать, как закругляются углы.
* Формы направляют внимание, определяют компоненты, сообщают о состоянии и выражают бренд.
* **Type.kt** — это место, где вы инициализируете свои шрифты и назначаете fontFamily, fontWeightи fontSizeдля шкалы типа Material Design.
* Шкала типов Material [Design](https://m3.material.io/styles/typography/type-scale-tokens) включает в себя ряд контрастных стилей, которые соответствуют потребностям вашего приложения и его контента. Масштаб шрифта представляет собой комбинацию 15 стилей, поддерживаемых системой шрифтов.