# **Приложение Art Space**

Создадим собственное цифровое арт-пространство — приложение, которое отображает множество произведений искусства, которые вы можете продемонстрировать. В отличие от предыдущих лабораторных, в которых вам давались пошаговые инструкции, здесь вам предоставляются только рекомендации и предложения о том, что вы можете создать, используя концепции, которые вы уже изучили. Вам рекомендуется использовать свой творческий потенциал для самостоятельной разработки приложения с ограниченным руководством.

Создать приложение самостоятельно — непростая задача, но не волнуйтесь, ведь у вас достаточно практики! Вы можете использовать те же навыки, которые вы изучили, в этом новом контексте. Вы всегда можете обратиться к предыдущим лабораторным, если не знаете, как реализовать определенные части приложения.

Когда вы создаете это приложение самостоятельно и решаете проблемы, с которыми сталкиваетесь на своем пути, вы учитесь быстрее и дольше сохраняете концепции. Дополнительным преимуществом является то, что приложение полностью настраивается, поэтому вы можете использовать его, чтобы продемонстрировать свое творение как часть своего портфолио разработчика!

## [2. Создайте статический пользовательский интерфейс с помощью составных элементов.](https://developer.android.com/codelabs/basic-android-kotlin-compose-art-space?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-2-pathway-3%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-art-space#1)

## Создайте низкокачественный прототип

Прототип низкой точности (low-fi) представляет собой простую модель или рисунок, который дает базовое представление о том, как выглядит приложение.

Создайте низкокачественный прототип:

1. Подумайте, что вы хотите отображать в своем приложении Art Space и кто является его целевой аудиторией.
2. На предпочитаемом вами носителе добавьте элементы, составляющие ваше приложение. Некоторые элементы, которые следует учитывать, включают:

* Изображение произведения искусства
* Информация об произведении искусства, такая как его название, художник и год публикации.
* Любые другие элементы, например кнопки, которые делают приложение интерактивным и динамичным.

1. Расположите эти элементы в разных положениях и затем оцените их визуально. Не беспокойтесь о том, чтобы все получилось идеально с первого раза. Вы всегда можете остановиться на одном дизайне сейчас и постепенно улучшать его позже.
2. Вы можете придумать низкокачественный дизайн, похожий на это изображение:

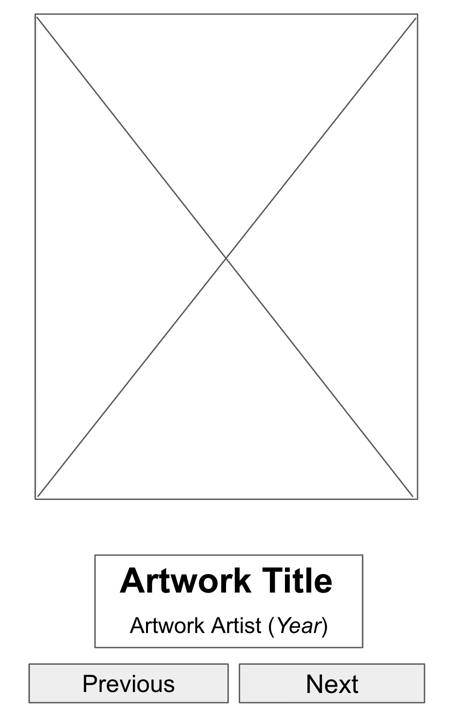


Рисунок 1. Элементы-заполнители в макетах пользовательского интерфейса помогают визуализировать конечный продукт.

## Превратите дизайн в код

Используйте свой прототип, чтобы перевести свой дизайн в код:

1. Определите элементы пользовательского интерфейса, необходимые для создания вашего приложения.

Например, из созданного вами примера дизайна вам нужно иметь в коде один Image, два Text и Button.

1. Определите различные логические разделы приложений и очертите их границы.

Этот шаг поможет вам разделить экран на небольшие составные элементы и подумать об их иерархии.

В этом примере вы можете разделить экран на три части:

* Стена с произведениями искусства
* Дескриптор произведения искусства
* Контроллер дисплея

Вы можете упорядочить каждый из этих разделов с помощью составных макетов, например Row или Column.

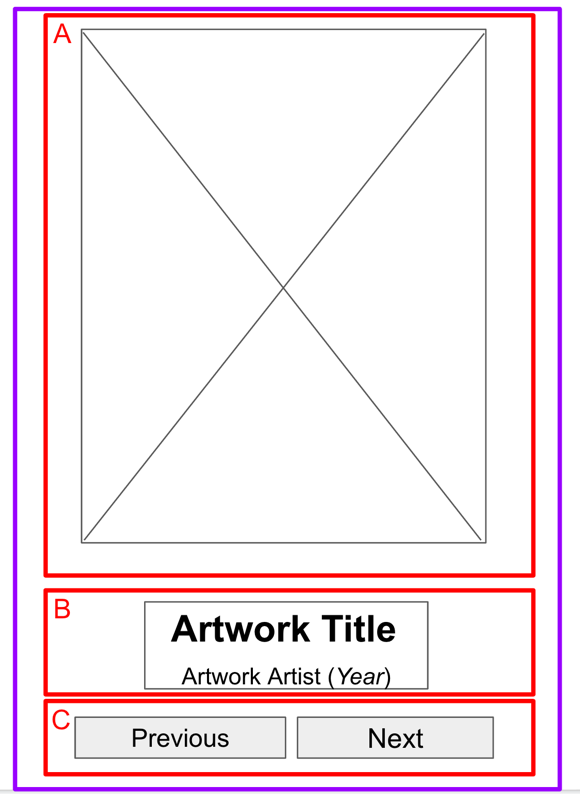


Рисунок 2. Границы вокруг разделов помогают разработчикам концептуализировать составные элементы.

1. Для каждого раздела приложения, содержащего несколько элементов пользовательского интерфейса, нарисуйте границы вокруг них.

Эти границы помогут вам увидеть, как один элемент связан с другим в разделе.

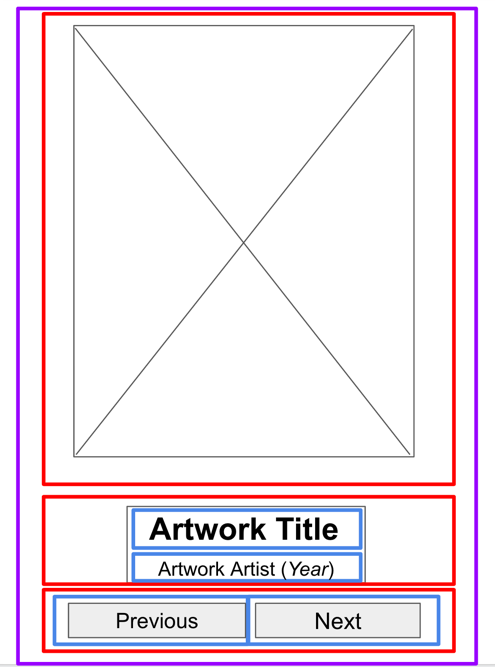


Рис. 3. Дополнительные границы вокруг текста и кнопок помогают разработчикам упорядочивать составные элементы.

Теперь стало проще понять, как можно упорядочить составные объекты, например Text и Button, с помощью составных элементов макета.

Некоторые заметки о различных составных компонентах, которые вы можете использовать:

* **Row или Column**. Поэкспериментируйте с различными параметрами horizontalArrangement и verticalAlignment и составными элементами Row и Column, чтобы они соответствовали имеющемуся у вас дизайну
* **Image** . Не забудьте заполнить параметр contentDescription. Как упоминалось в предыдущей лабораторной, **TalkBack** использует этот параметр contentDescription, чтобы обеспечить доступность приложения. Если Image используется только в декоративных целях или существует элемент Text, описывающий Image, вы можете установить  для параметра contentDescription значение null.
* **Text** . Вы можете поэкспериментировать с различными значениями fontSize, textAlign и fontWeight стилизовать текст. Вы также можете использовать функцию [buildAnnotatedString](https://developer.android.com/jetpack/compose/text#multiple-styles) для применения нескольких стилей к одному Text.
* **Surface** . Вы можете поэкспериментировать с различными значениями Elevation, Color и BorderStroke для Modifier.border, чтобы создавать разные пользовательские интерфейсы в Surface.
* **Интервал и выравнивание** . Вы можете использовать аргументы Modifier, такие как padding и weight, чтобы упростить расположение составных элементов.

**Примечание** . В простом приложении вы можете стилизовать каждый элемент пользовательского интерфейса отдельно. Однако этот подход создает накладные расходы при добавлении дополнительных экранов. Compose помогает поддерживать согласованность дизайна благодаря реализации Material Design. В будущих модулях вы узнаете больше о Material Design и Material Theming. Дополнительные сведения см. в [разделе Темы материалов в Compose](https://developer.android.com/jetpack/compose/designsystems/material3) .

1. Запустите приложение в эмуляторе или на своем устройстве Android.



Рисунок 4. Это приложение показывает статический контент, но пользователи пока не могут с ним взаимодействовать.

## [3. Сделайте приложение интерактивным](https://developer.android.com/codelabs/basic-android-kotlin-compose-art-space?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-2-pathway-3%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-art-space#2)

## Определить взаимодействие с пользователем

Преимущество создания арт-пространства в цифровом формате заключается в том, что вы можете сделать его интерактивным и динамичным для своих пользователей. В первоначальном дизайне вы создали две кнопки для взаимодействия пользователей. Однако это ваше собственное арт-пространство! Вы можете изменить дизайн и способ взаимодействия пользователей с приложением по своему усмотрению. Теперь подумайте о том, как вы хотите, чтобы ваши пользователи взаимодействовали с приложением и как приложение должно реагировать на такие взаимодействия. Некоторые возможные взаимодействия, которые вы можете добавить в свое приложение, включают:

* Показывать следующую или предыдущую иллюстрацию при нажатии кнопки.
* При пролистывании перемотка вперед отображает обложку к следующему альбому.
* Показывать всплывающую подсказку для получения дополнительной информации при длительном нажатии кнопки.

**Примечание** . Compose поддерживает множество жестов и анимаций, которые делают ваше приложение интерактивным. Больше об анимации вы узнаете в будущих модулях. Дополнительные сведения о сложных темах см. в [разделе Жесты](https://developer.android.com/jetpack/compose/gestures) .

## Создание состояний для динамических элементов

Поработайте над той частью пользовательского интерфейса, которая отображает следующую или предыдущую иллюстрацию при нажатии кнопки:

1. Сначала определите элементы пользовательского интерфейса, которые необходимо изменить при взаимодействии с пользователем.

В данном случае элементами пользовательского интерфейса являются изображение произведения искусства, название произведения искусства, художник и год.

1. При необходимости создайте состояние для каждого из динамических элементов пользовательского интерфейса с помощью объекта MutableState.
2. Не забудьте заменить жестко закодированные значения на определенные states.

**Примечание** . Хотя в настоящее время вы используете одно состояние для каждого динамического элемента пользовательского интерфейса, этот подход может быть не самым эффективным для читаемости кода и производительности приложения. Потенциально вы можете сгруппировать связанные элементы как одну сущность и объявить ее как единое состояние, что вы научитесь делать с помощью классов Collection и Data в будущих модулях. После того, как вы изучите эти концепции, вы можете вернуться к этому проекту и провести рефакторинг своего кода с использованием изученных вами концепций.

## Напишите условную логику для взаимодействия

1. Подумайте о том, какое поведение вам нужно, когда пользователь нажимает кнопки, начиная с кнопки **«Далее»** .

Когда пользователи нажимают кнопку **«Далее»** , они должны ожидать увидеть следующее изображение в последовательности. На данный момент может быть сложно определить, какое произведение искусства будет показано следующим.

1. Добавьте идентификаторы или идентификаторы в виде последовательных номеров, начинающихся с 1, для каждого произведения искусства.

Теперь понятно, что следующая иллюстрация относится к иллюстрации, которая имеет следующий идентификатор в последовательности.

Поскольку у вас нет бесконечного количества произведений искусства, вы также можете определить поведение кнопки «**Далее»** при отображении последнего произведения искусства в серии. Обычное поведение — вернуться назад, чтобы отобразить первый фрагмент рисунка после последнего.

1. Сначала напишите псевдокод, чтобы передать логику кода без синтаксиса Kotlin.

Если нужно показать три фрагмента изображения, псевдокод логики кнопки « **Далее»** может выглядеть примерно так:

if (current artwork is the first artwork) {  
    // Update states to show the second artwork.  
}  
else if (current artwork is the second artwork) {  
    // Update states to show the third artwork.  
}  
else if (current artwork is the last artwork) {  
   // Update state to show the first artwork.  
}

1. Преобразуйте псевдокод в код Kotlin.

Вы можете использовать оператор when для построения условной логики вместо if else операторов, чтобы сделать ваш код более читабельным, когда он управляет большим количеством случаев художественного оформления.

1. Чтобы эта логика выполнялась при нажатии кнопки, поместите ее в аргумент onClick() у Button.
2. Повторите те же шаги, чтобы построить логику для кнопки **«Предыдущий»** .
3. Запустите приложение, а затем нажмите кнопки, чтобы подтвердить, что они переключают отображение на предыдущее или следующее изображение.

## [4. Задача: создать приложение для экранов разных размеров.](https://developer.android.com/codelabs/basic-android-kotlin-compose-art-space?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-2-pathway-3%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-art-space#3)

Одной из сильных сторон Android является то, что он поддерживает множество устройств и размеров экранов, а это означает, что создаваемое вами приложение может охватить широкую аудиторию и использоваться множеством способов. Чтобы обеспечить наилучшее взаимодействие для всех пользователей, вам следует протестировать свои приложения на устройствах, которые ваше приложение намерено поддерживать. Например, в текущем примере приложения вы могли изначально спроектировать, создать и протестировать приложение для мобильных устройств в портретном режиме. Однако некоторым пользователям может быть интересно использовать ваше приложение на большом экране в ландшафтном режиме.

Хотя планшеты не являются основным поддерживаемым устройством для этого приложения, вы все равно хотите убедиться, что приложение не сломается, если пользователи будут использовать его на большом экране.

Проверьте свое приложение на большом экране планшета:

1. Если у вас нет планшетного устройства Android, [создайте виртуальное устройство Android (AVD)](https://developer.android.com/studio/run/managing-avds#createavd) .
2. Создайте и запустите приложение на планшете AVD в ландшафтном режиме.
3. Визуально убедитесь, что ничто не выглядит неприемлемым, например, некоторые элементы пользовательского интерфейса обрезаны, неровное выравнивание или взаимодействие кнопок не работают должным образом.

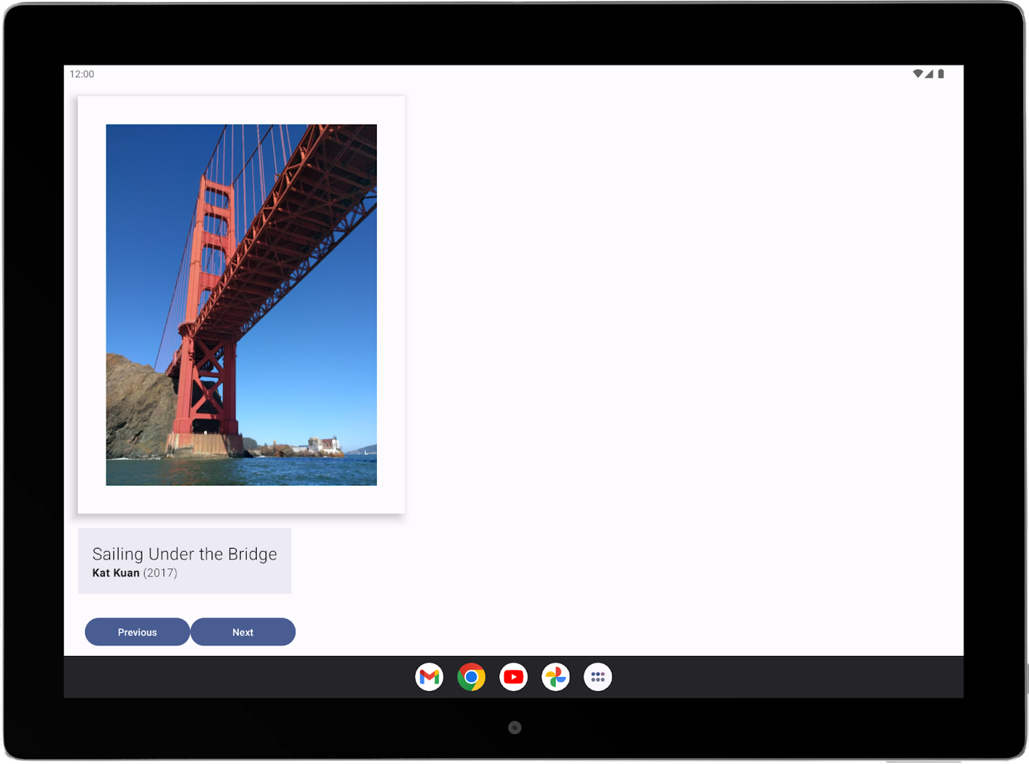


Рисунок 5. Приложение необходимо переработать, чтобы оно правильно отображалось на более крупном устройстве.

1. Измените код, чтобы исправить все найденные ошибки. В качестве руководства см. [основные рекомендации по совместимости приложений с большим экраном](https://developer.android.com/docs/quality-guidelines/large-screens-app-quality#basic-compat) .
2. Еще раз протестируйте приложение для планшета и телефона, чтобы убедиться, что исправление ошибки работает на обоих типах устройств.

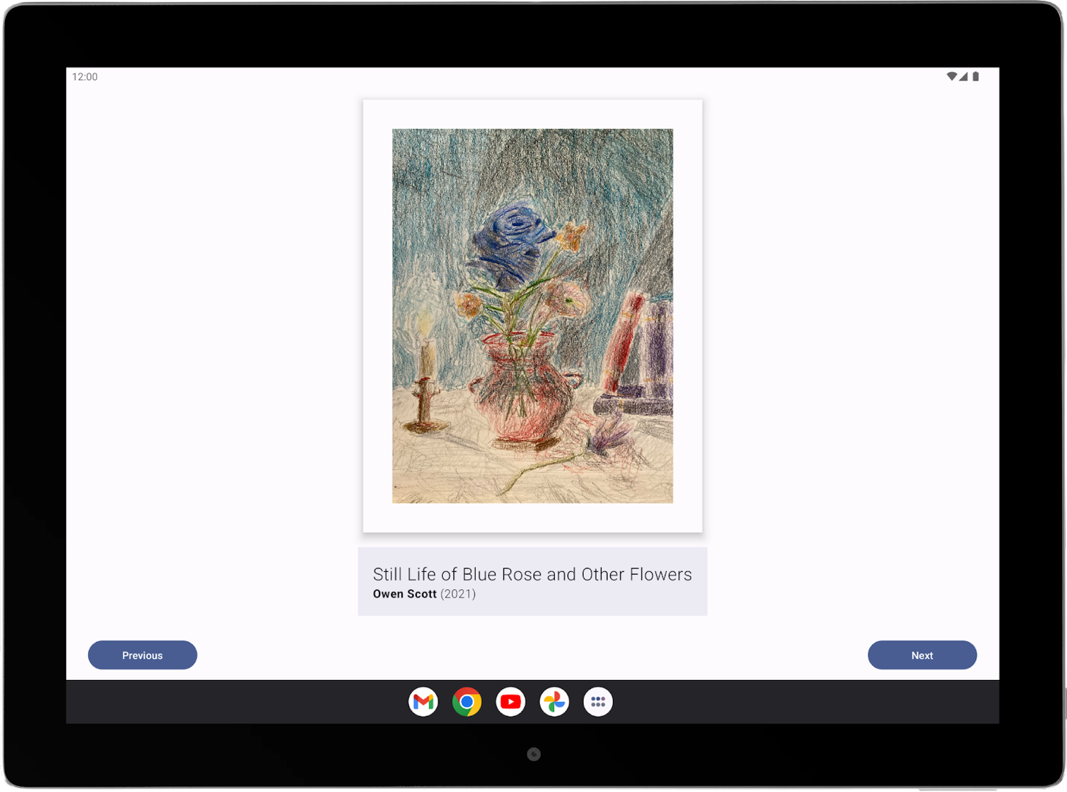


Рисунок 6. Приложение теперь хорошо выглядит на большом экране.

**Примечание** . Вы можете заметить, что многие приложения, поддерживающие планшеты и телефоны, могут выглядеть по-разному в разных форм-факторах. Эта разница возникает потому, что часто приложение поддерживает разные макеты для разных размеров экрана. Подробнее см. в разделе [Поддержка различных размеров экрана](https://developer.android.com/guide/topics/large-screens/support-different-screen-sizes#compose) .