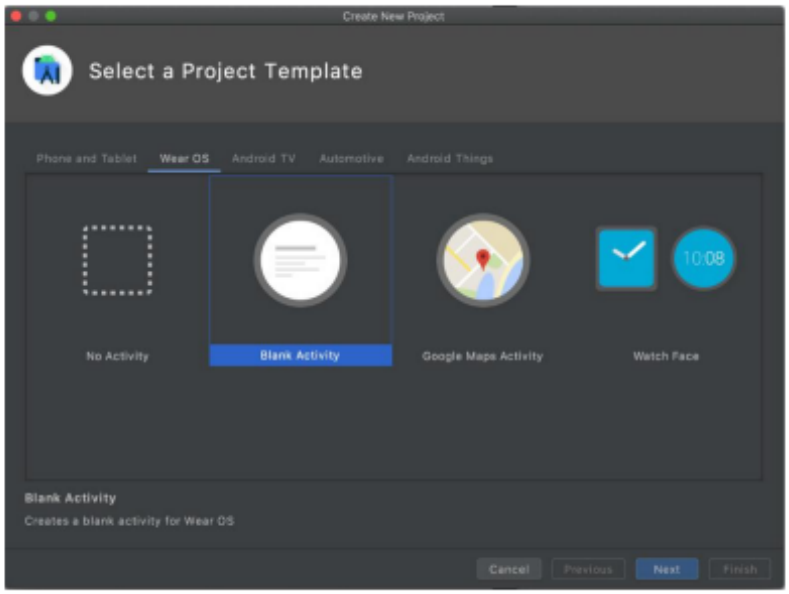
МОДУЛЬ КОМПЕТЕНЦИИ «РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ УМНЫХ ЧАСОВ»

Разработка приложения для часов.

После того, как мы научились создавать приложения для мобильных устройств, пора переходить на новый уровень и начинать писать программы для часов.

Создаём новый проект. Главное отличие при создании проекта вместо вкладки Phone and Tablet мы выбираем Wear OS (Blank Activity).

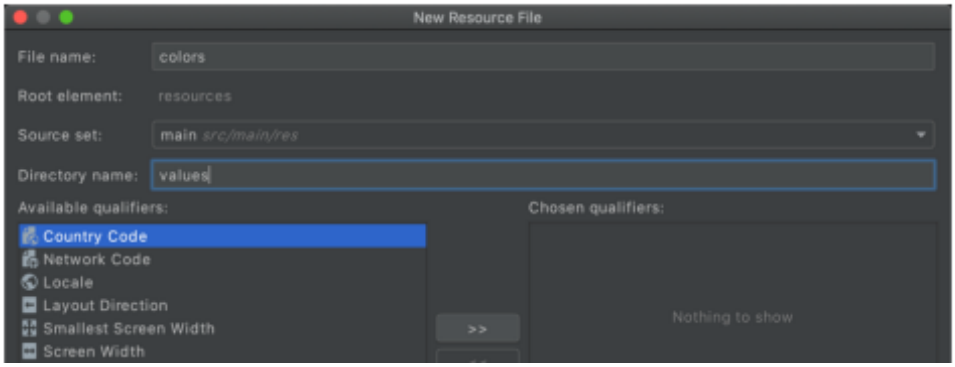


Как и в прошлом нашем приложении задаем название нашего проекта, язык на котором будем писать, в нашем случае это Kotlin и версию Android 9.0(Pie). Жмем кнопку Finish и завершаем генерацию приложения.

После создания нашего объекта изменим его название на MADVENTURE.

В данном разделе элементы повторяющиеся из прошлых модулей не будут расписаны подробно, описываются отличия верстки и реализации бизнес-логики на примере одного из макетов.

Откроем первый экран сделаем верстку для него согласно макету. Нам необходимо добавить логотип, создать кнопку и залить задний фон цветом **#171717**. В часах отсутствует файл с стандартными цветам, поэтому нам необходимо создать его. В папке values создадим файл с именем colors.



Заполним его базовыми цветами, которые необходимы нам в проекте.

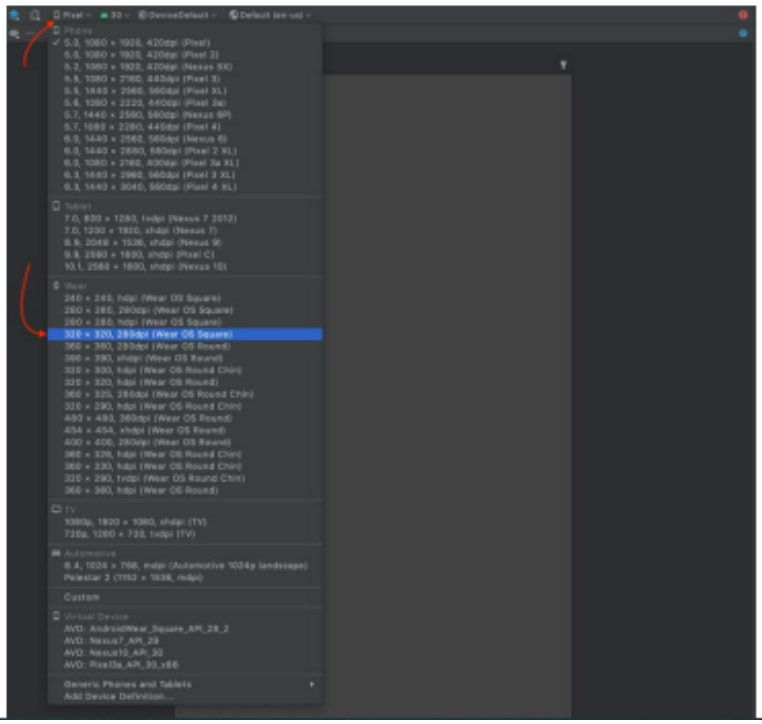
```
<resources>
    <color name="white">#FFFFFF</color>
    <color name="black">#000000</color>
    <color name="gray">#171717</color>
    <color name="aquamarine">#14C7AF</color>
    <color name="red">#FF2525</color>

</resources>
```

Аналогичным образом создадим папку drawable, в которую будем отправлять наши изображения.

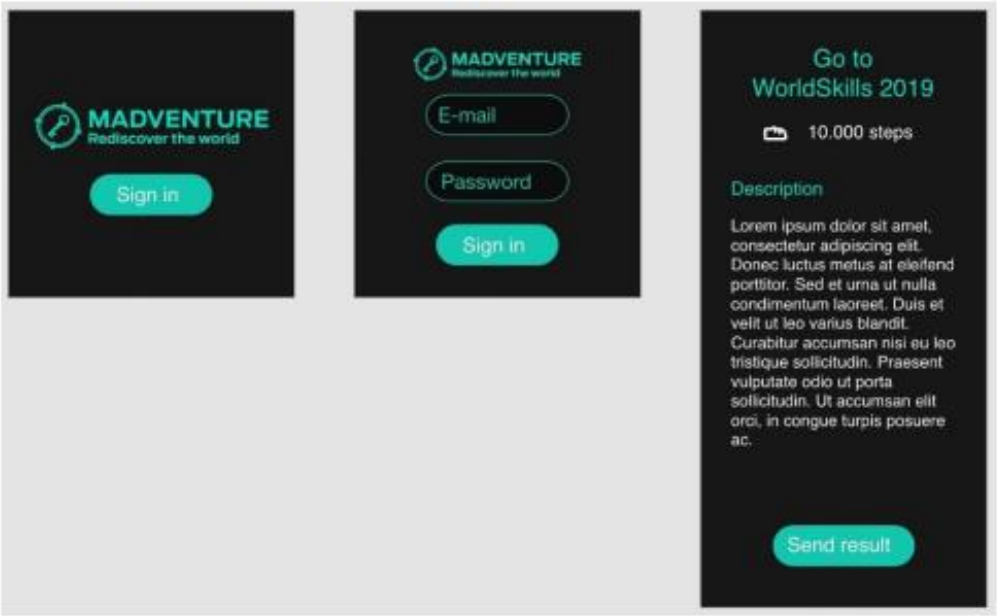
Разработка приложений для носимых устройств отличается от разработки под стандартные устройства. Прежде всего это связано с форм-фактором. На данный момент имеются круглые и квадратные часы. Отсюда могут возникнуть проблемы.

Перейдем к файлу с верстской проекта и изменим устройство, с которого мы хотим просматривать макет наших часов:



После изменения устройства макета начнем верстать наш экран. Мы будем использовать универсальный вид разметки BoxInsetLayout, который является потомком FrameLayout. Смысл разметки заключается в том, что прямоугольная разметка вписывается в круг.

Перейдем к верстке. Ориентироваться будем на данный макет.



Реализуем экран приветствия с переходом на экран входа.

Верстка экрана приветствия должна выглядеть следующим образом:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.wear.widget.BoxInsetLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/gray"
    android:padding="@dimen/box_inset_layout_padding"
    tools:context=".MainActivity"
    tools:deviceIds="wear">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        android:padding="@dimen/inner_frame_layout_padding"
        >
    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="40dp"
        android:paddingRight="10dp"
        android:paddingLeft="10dp"
        android:src="@drawable/logo"/>

        <Button
            android:id="@+id/button2"
            android:layout_width="120dp"
            android:layout_height="40dp"
            android:textAllCaps="false"
            android:layout_gravity="center"
            android:layout_marginTop="10dp"
            android:onClick="next_activity"
            android:background = "@drawable/bt_shape"
            android:text="Sign In" />
```

При нажатии на кнопку должен быть реализован переход на другой экран. Для этого нам необходимо создать новый экран с именем **SignInActivity**, реализовать переход с нашего экрана на SignInActivity с помощью Intent.

Результат сверстанного приветственного экрана

В файле **SignInActivity** мы должны заменить `class SignInActivity : AppCompatActivity()` на `class SignInActivity : WearableActivity() {` Этим действием мы меняем класс активности и указываем, что она будет использоваться на часах. Это действие необходимо повторять каждый раз при создании новой активности для часов. Так же необходимо каждый раз изменять разметку в новых файлах.

```
<androidx.wear.widget.BoxInsetLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/gray"
    android:padding="@dimen/box_inset_layout_padding"
    tools:context=".SignInActivity"
    tools:deviceIds="wear">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        android:padding="@dimen/inner_frame_layout_padding"
        >
    </LinearLayout>
</androidx.wear.widget.BoxInsetLayout>
```

Реализуем экран входа согласно макету, экран будет состоять из следующих элментов:

ImageView, 2 EditText, Button. Работу с этими объектами мы проходили ранее. Так же используем Toast.

Всплывающее уведомление (Toast Notification) является сообщением, которое появляется на поверхности окна приложения, заполняя необходимое ему количество пространства, требуемого для сообщения. При этом текущая деятельность приложения остаётся работоспособной для пользователя. В течение нескольких секунд сообщение плавно закрывается. Всплывающее уведомление также может быть создано службой, работающей в фоновом режиме. Как правило, всплывающее уведомление используется для показа коротких текстовых сообщений.

У метода makeText() есть три параметра:

- Контекст приложения
- Текстовое сообщение
- Продолжительность времени показа уведомления.

Можно использовать только две константы:

- **LENGTH_SHORT** - (По умолчанию) показывает текстовое уведомление на короткий промежуток времени;
- **LENGTH_LONG** - показывает текстовое уведомление в течение длительного периода времени

Пример Toast нотификации.

```
fun take_result(view: View) {
    Toast.makeText( context: this,  text: "Sucess", Toast.LENGTH_SHORT).show()
}
```

Экран входа можно реализовать следующим образом:

Аналогично мобильному устройству необходимо проверить поля на заполнение и сделать переход на следующий экран, при наличии ошибки– вывести ее.

После этого создадим 3ю активность с названием **ResultActivity**.

Откроем макет для верстки и добавим в него ScrollView. Контейнер ScrollView предназначен для создания прокрутки для такого интерфейса, все элементы которого одномоментно не могут поместиться на экране устройства.

ScrollView может вмещать только один элемент, поэтому если мы хотим разместить несколько элементов, то их надо поместить в какой-нибудь контейнер. Так как в ScrollView можно поместить только один элемент, то все элементы заключим в LinearLayout.

Если площадь экрана будет недостаточной, чтобы поместить все содержимое LinearLayout, то станет доступной прокрутка:

Результат верстки **ResultActivity**

```
<androidx.wear.widget.BoxInsetLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/gray"
    android:padding="@dimen/box_inset_layout_padding"
    tools:context=".ResultActivity"
    tools:deviceIds="wear">
<ScrollView
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:padding="@dimen/inner_frame_layout_padding"
        >
        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:textColor="@color/aquamarine"
            android:text="Go to \n WorldSkills 2019"
            android:textAlignment="center"/>
        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginLeft="20dp"
            android:layout_marginRight="20dp"
            android:textAlignment="center"
            android:paddingLeft="20dp"
            android:text="10,000 steps"
            android:textColor="@color/white"
            android:textSize="10sp"
            android:drawableLeft="@drawable/shoe"/>
        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="15dp"
            android:text="Description"
            android:layout_marginLeft="10dp"
            android:textColor="@color/aquamarine"
            />

        <TextView
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:text="Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec luctus
            android:textAlignment="textStart"
            android:layout_marginRight="10dp"
            android:layout_marginLeft="10dp"
            android:textColor="@color/white" />
        <Button
            android:id="@+id/button3"
            android:layout_width="120dp"
            android:layout_height="40dp"
            android:textAllCaps="false"
            android:layout_gravity="center"
            android:layout_marginTop="10dp"
            android:onClick="take_result"
```

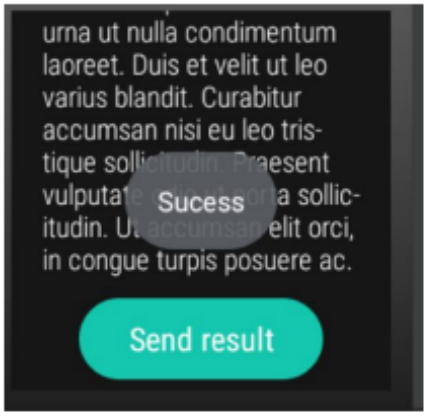
Текст для экрана вы можете скопировать в предоставленном макете.

Теперь информацию на нашем экране можно пролистывать. Теперь необходимо добавить обработчик нажатии на нашу кнопку, который будет выводить уведомление.

Для этого перейдем в файл ResultActivity и в обработчик нажатия впишем:

```
fun take_result(view: View) {
    Toast.makeText( context: this,  text: "Sucess", Toast.LENGTH_SHORT).show()
}
```

Результатом этого кода будет:



WearableRecyclerView

Списки – удобный паттерн, который активно используется в мобильном (и не только) UX. Wear-интерфейсы исключением не стали. Но из-за закругления углов дисплея верхние View у списка могут обрезаться. WearableRecyclerView помогает исправить такие недоразумения.Например, есть параметр isEdgeItemsCenteringEnabled, который позволяет задать компоновку элементов по изгибу экрана и расширять центральный элемент, делает список более удобным для чтения на маленьком экране.

Есть WearableLinearLayoutManager, который позволяет прокручивать список механическим колесиком на часах и доскроливать крайние элементы до середины экрана, что очень удобно на круглых интерфейсах.

Рисовать данные на экране – весело, но эти данные нужно откуда-то получать. В случае мобильного клиента, мы чаще используем REST API поверх привычных всем сетевых протоколов (HTTP/TCP). В Wear OS подобный подход тоже допустим, но Google его не рекомендует. В носимой электронике большую роль играет энергоэффективность. А активное интернет-соединение будет быстро сажать батарею, и могут регулярно происходить разрывы связи. Ещё носимые устройства предполагают активную синхронизацию, которую тоже нужно реализовывать. Все эти проблемы за нас любезно решает механизм обмена данными в Google Services под названием «Data Layer». Классы для работы с ним нашли свое место в пакете com.google.android.gms.wearable.