Пагинация

Пагинация или постраничный вывод в приложении является одной из распространенных задач, и для упрощения ее реализации Compose предоставляет ряд компонентов-пагинаторов, в частности, компоненты VerticalPager и HorizontalPager. Эти компоненты позволяют перелистывать контент по горизонтали или по вертикали. Поведение по умолчанию — перелистывание страниц с помощью свайпа пальцем влево и вправо для HorizontalPager и вверх и вниз для VerticalPager. Пагинаторы также предоставляют интерфейс для перехода к определенным страницам и различные варианты управления и контроля поведением компонента пагинации. Стоит отметить, что эти компоненты используют ленивую загрузку (как LazyColumn/LazyRow), то есть по умолчанию непосредственно компонуются только те страницы, которые находятся в видимой области.

Компонент HorizontalPager применяется для создания горизонтальной пагинации. Он имеет следующие параметры:

```
@Composable
fun HorizontalPager(
    state: PagerState,
    modifier: Modifier = Modifier,
    contentPadding: PaddingValues = PaddingValues(0.dp),
    pageSize: PageSize = PageSize.Fill,
    beyondViewportPageCount: Int = PagerDefaults.BeyondViewportPageCount,
    pageSpacing: Dp = 0.dp,
    verticalAlignment: Alignment.Vertical = Alignment.CenterVertically,
    flingBehavior: TargetedFlingBehavior = PagerDefaults.flingBehavior(state =
state),
    userScrollEnabled: Boolean = true,
    reverseLayout: Boolean = false,
    key: ((index: Int) -> Any)? = null,
    pageNestedScrollConnection: NestedScrollConnection =
PagerDefaults.pageNestedScrollConnection(
        state,
        Orientation.Horizontal
    ),
    snapPosition: SnapPosition = SnapPosition.Start,
    pageContent: @Composable PagerScope.(page: Int) -> Unit
): Unit
```

- state: состояние пагинатора в виде объекта PagerState
- modifier: применяемые к компоненту функции модификатора
- contentPadding: отступ вокруг контента в виде объекта PaddingValues. Может применяться для добавления отступов перед первой страницей или после последней страницы.
- pageSize: определяет, как страницы будут выглядеть внутри компонента. Представляет значение типа PageSize.

- beyondViewportPageCount: количество страниц, которые располагаются до или после видимой части и которые надо предварительно скомпоновать. Служит для цели оптимизации для ускорения загрузки страниц.
- pageSpacing: отступы в пикселях между отдельными страницами в компоненте
- verticalAlignment: вертикальное выравнивание страниц внутри компонента.
- flingBehavior: определяет поведение для момента, когда выполнена прокрутка пальцем.
- userScrollEnabled: указывает, будет ли доступна прокрутка.
- reverseLayout: указывает, надо ли размещать содержимое (страницы) в обратном порядке.
- key: определяет уникальный ключ для каждого элемента с помощью функции типа (index: Int) -> Any. Ключ применяется для определения позиции при прокрутке. Если передается null, то позиция при прокрутке определяется с помощью позиции в списке.
- pageNestedScrollConnection: NestedScrollConnection = PagerDefaults.pageNestedScrollConnection(state,Orientation.Horizontal) A NestedScrollConnection that dictates how this Pager behaves with nested lists. The default behavior will see Pager to consume all nested deltas.
- snapPosition: определяет, как компонент будет выполнять привязку страниц.
- pageContent: определяет содержимое страницы с помощью функции-компонента PagerScope. (page: Int) -> Unit.

VerticalPager располагает страницы по вертикали и имеет аналогичный набор параметров:

```
@Composable
fun VerticalPager(
    state: PagerState,
    modifier: Modifier = Modifier,
    contentPadding: PaddingValues = PaddingValues(0.dp),
    pageSize: PageSize = PageSize.Fill,
    beyondViewportPageCount: Int = PagerDefaults.BeyondViewportPageCount,
    pageSpacing: Dp = 0.dp,
    horizontalAlignment: Alignment.Horizontal = Alignment.CenterHorizontally,
    flingBehavior: TargetedFlingBehavior = PagerDefaults.flingBehavior(state =
state),
    userScrollEnabled: Boolean = true,
    reverseLayout: Boolean = false,
    key: ((index: Int) -> Any)? = null,
    pageNestedScrollConnection: NestedScrollConnection =
PagerDefaults.pageNestedScrollConnection(
        state,
        Orientation.Vertical
    ),
    snapPosition: SnapPosition = SnapPosition.Start,
    pageContent: @Composable PagerScope.(page: Int) -> Unit
): Unit
```

Единственное отличие, что для страниц здесь используется выравнивание по вертикали.

Состояние PagerState

Для определения HorizontalPager или VerticalPager прежде всего необходимо определить их состояние. Для управления состоянием пагинатора применяется объект PagerState. Класс PagerState определяет ряд свойств и методов, которые позволяют получить информацию о состоянии пагинатора, либо управлять его поведением, например, прокручивать контент. Наиболее распространенные свойства:

- pageCount: количество страниц
- currentPage: "текущая" страница (страница, наиболее близкая к текущей точке прокрутки)

Среди методов PagerState следует отметить следующие:

- scrollToPage(page): мгновенный переход к странице с номером page
- animateScrollToPage(page): плавный переход к странице с номером page

Для определения состояния применяется функция rememberPagerState()

```
@ExperimentalFoundationApi
@Composable
public fun rememberPagerState(
    initialPage: Int = 0,
    initialPageOffsetFraction: Float = 0f,
    pageCount: () -> Int
): PagerState
```

Эта функция принимает три параметра:

- initialPage: номер начальной страницы, которая будет отображаться первой.
- initialPageOffsetFraction: смещение начальной страницы в виде доли от размера страниц
- pageCount: количество страниц в пагинаторе.

Например, определим состояние для пагинатора, в котормом будет 10 страниц:

```
val pagerState = rememberPagerState { 10 }
```

Определение содержимого Последний параметр компонентов-пагинаторов устанавливает содержимое, в качестве которого может выступать любой компонент. Для установки содержимого применяется функция (page: Int) -> Unit. Ее единственный параметр представляет индекс страницы (начиная с 0). Например, определим вертикальный пагинатор, где содержимое, то есть каждая страница представляет компонент Text, который отображает текущий номер страницы:

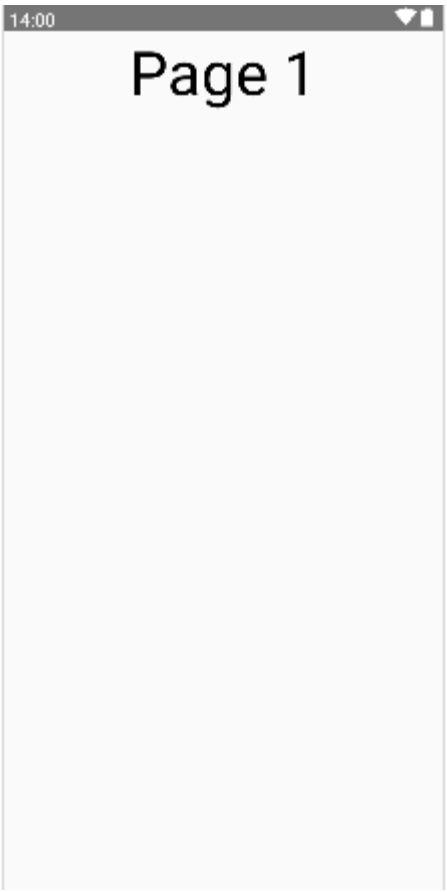
```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.ExperimentalFoundationApi
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.pager.VerticalPager
import androidx.compose.foundation.pager.rememberPagerState
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent @OptIn(ExperimentalFoundationApi::class){
            val pagerState = rememberPagerState { 10 }
            VerticalPager(state = pagerState, Modifier.fillMaxWidth()) { page ->
                Text("Page $page", fontSize = 55.sp)
            }
        }
    }
}
```

В итоге создается 10 страниц, и с помощью свайпа пальцем вверх-вниз мы можем перемещаться по ним:



Пример пагинации. Навигационные кнопки

В предыдущем пункте были описаны общие моменты применения компонентов-пагинаторов VerticalPager и HorizontalPager. В этой статье рассмотрим примитивный пример применения постраничного вывода. В частности, определим следующее приложение:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.ExperimentalFoundationApi
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxHeight
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.pager.HorizontalPager
import androidx.compose.foundation.pager.rememberPagerState
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
```

```
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import kotlin.random.Random

class MainActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent @OptIn(ExperimentalFoundationApi::class){
            // данные для отображения
            val data = listOf("iPhone 15 Pro", "Redmi Note 12 Pro+", "Galaxy S23
Ultra", "Infinix NOTE 30 Pro", "Honor 90")
            // состояние
            val pagerState = rememberPagerState { data.size }

            HorizontalPager(state = pagerState, Modifier.fillMaxHeight()) { page -
>
                Column(Modifier.fillMaxSize(), horizontalAlignment =
Alignment.CenterHorizontally) {
                    val product = data[page]
                    Text(product, fontSize = 43.sp)
                    Box(Modifier
                        .fillMaxWidth(0.9f)
                        .fillMaxHeight(0.7f)
                        .padding(top=40.dp)
                        .background(
                            Color(
                                Random.nextInt(255),
                                Random.nextInt(255),
                                Random.nextInt(255),
                                255
                            )
                        )
                    )
                    Box(Modifier
                        .fillMaxWidth(0.9f)
                        .padding(top=20.dp)){
                        Text("Описание товара $product", fontSize = 28.sp)}
                }
            }
        }
    }
}
```

Здесь данные для отображения на страницах определены в виде списка data - списка мобильных устройств:

```
val data = listOf("iPhone 15 Pro", "Redmi Note 12 Pro+", "Galaxy S23 Ultra",
"Infinix NOTE 30 Pro", "Honor 90")
```

Для каждого из элементов списка будет создаваться своя страница.

Создаем состояние PagerState, которое инициализируется количеством элементов списка (фактически количеством страниц):

```
val pagerState = rememberPagerState { data.size }
```

Далее определяем горизонтальный пагинатор HorizontalPager:

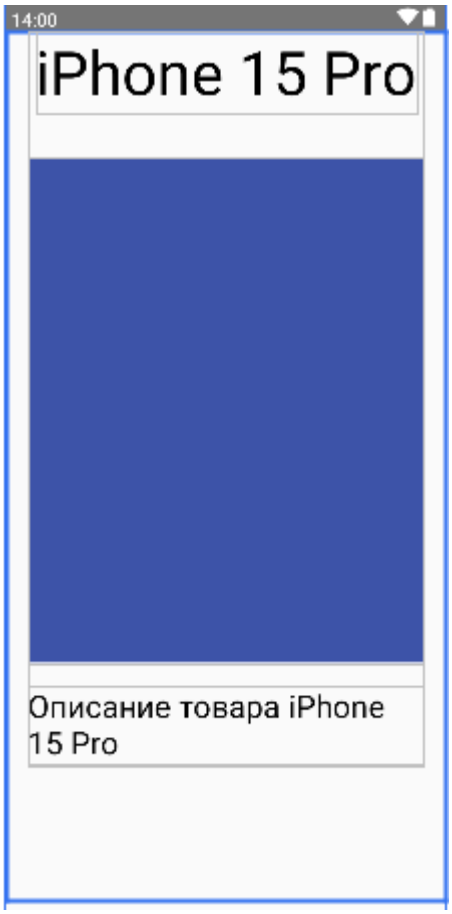
```
HorizontalPager(state = pagerState, Modifier.fillMaxHeight()) { page ->
```

Каждую страницу в пагинаторе фактически будет представлять столбец Column:

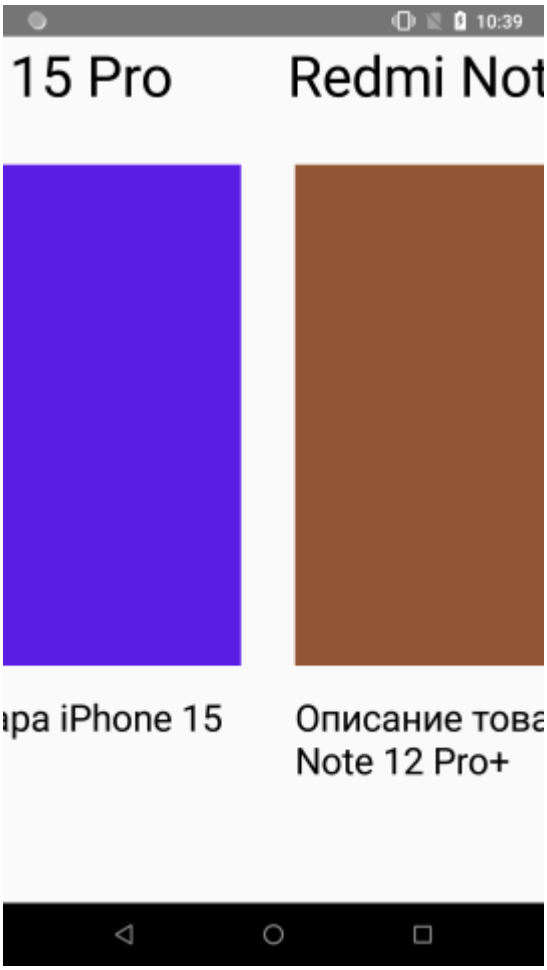
```
Column(Modifier.fillMaxSize(), horizontalAlignment = Alignment.CenterHorizontally)
{
    val product = data[page]
    Text(product, fontSize = 43.sp)
    Box(Modifier
        .fillMaxWidth(0.9f)
        .fillMaxHeight(0.7f)
        .padding(top=40.dp)
        .background(
            Color(Random.nextInt(255),Random.nextInt(255),Random.nextInt(255),255)
        )
    )
    Box(Modifier.fillMaxWidth(0.9f).padding(top=20.dp)){
        Text("Описание товара $product", fontSize = 28.sp)
    }
}
```

Страница может представлять любой компонент, содержать различные компоненты в зависимости от задачи приложения. Но в данном случае функционально страница разбивается на три части. В начале идет компонент Text с заголовком товара. Потом идет компонент Box, который принимает случайный цвет (здесь можно было бы вставить картинку, но, я думаю, для демонстрации сойдет и обычный Box). И под ним расположено описание товара в виде компонента Box со вложенным компонентом Text.

Таким образом, у нас получится следующий интерфейс страницы:



В итоге при запуске приложения мы сможем переходить к другим страницам с помощью прокручивания по горизонтали:



Хотя здесь применяется горизонтальный пагинатор HorizontalPager, но аналогично можно было бы использовать и вертикальный пагинатор VerticalPager.

Кнопки навигации

Нередко для перемещения по страницам в приложениях применяются специальные кнопки навигации. Посмотрим, как их реализовать. Для этого определим следующее приложение:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.ExperimentalFoundationApi
import androidx.compose.foundation.background
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.fillMaxHeight
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.pager.HorizontalPager
import androidx.compose.foundation.pager.rememberPagerState
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.KeyboardArrowLeft
import androidx.compose.material.icons.filled.KeyboardArrowRight
import androidx.compose.material3.Icon
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.rememberCoroutineScope
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import kotlinx.coroutines.launch
import kotlin.random.Random

class MainActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent @OptIn(ExperimentalFoundationApi::class){
            val data = listOf("iPhone 15 Pro", "Redmi Note 12 Pro+", "Galaxy S23
Ultra", "Infinix NOTE 30 Pro", "Honor 90")
            val pagerState = rememberPagerState { data.size }
            val coroutineScope = rememberCoroutineScope()
            HorizontalPager(state = pagerState, Modifier.fillMaxHeight()) { page -
>
```

```
                Column(Modifier.fillMaxSize(), horizontalAlignment =
Alignment.CenterHorizontally) {
                    val product = data[page]
                    Text(product, fontSize = 43.sp)
                    Box(Modifier
                        .fillMaxWidth(0.9f)
                        .fillMaxHeight(0.7f)
                        .padding(top=40.dp)
                        .background(

Color(Random.nextInt(255),Random.nextInt(255),Random.nextInt(255),255)
                        )
                    )
                    Box(Modifier.fillMaxWidth(0.9f).padding(top=20.dp)){
                        Text("Описание товара $product", fontSize = 28.sp)
                    }
                    Row {
                        Icon(
                            imageVector = Icons.Default.KeyboardArrowLeft,
                            contentDescription = "Next Page",
                            modifier = Modifier.size(75.dp).clickable {
                                coroutineScope.launch {

pagerState.animateScrollToPage(pagerState.currentPage - 1)
                                }
                            }
                        )
                        Icon(
                            imageVector = Icons.Default.KeyboardArrowRight,
                            contentDescription = "Next Page",
                            modifier = Modifier.size(75.dp).clickable {
                                coroutineScope.launch {

pagerState.animateScrollToPage(pagerState.currentPage + 1)
                                }
                            }
                        )
                    }
                }
            }
        }
    }
}
```

Для определения кнопок навигации здесь определена строка Row, которая содержит пару иконок Icon:

Для каждой иконки определен обработчик нажатия. Для выполнения перехода между страницами применяется метод pagerState.animateScrollToPage(), который переходит к странице с определенным индексом. Но поскольку этот метод представляет suspend-функцию, то он запускается из области корутины

```
Icon(
    ........................
    modifier = Modifier.size(75.dp).clickable {
        coroutineScope.launch {
            pagerState.animateScrollToPage(pagerState.currentPage - 1)
        }
    }
)
```

Для перехода по страницам странице с помощью состояния пагинатора получаем текущую страницу - pagerState.currentPage. Соответственно для получения индекса следующей страницы нам надо прибаить 1, а для получения индекса предыдущей страницы - отнять 1 от номера текущей страницы.