

Лабораторная работа № 2

Навигация с помощью JetPack Compose

Навигация — это библиотека Jetpack, которая позволяет перемещаться из одного пункта назначения в приложении в другой.

Подключение функционала навигации в проект

По умолчанию проект не содержит функционала навигации, и поэтому надо добавить все необходимые зависимости. Для этого в секцию [versions] файла `libs.version.toml` необходимо добавить версию подключаемой зависимости, а в секцию [libraries] имя подключаемой библиотеки:

```
[versions]
navigationCompose = "2.7.7"

.....

[libraries]
androidx-navigation-compose = { module = "androidx.navigation:navigation-
compose", version.ref = "navigationCompose" }
.....
```

Затем в файл `build.gradle.kts` (Module :app) в секцию `dependencies` надо добавить следующую директиву:

```
dependencies {

    implementation(libs.androidx.navigation.compose)

    .....
```

Синхронизировать и обновить проект

NavHostController

Контроллер навигации **NavHostController** отслеживает стек составных функций при переходе с одного экрана на другой, при этом сохраняя состояние каждого экрана.

Для создания объекта `NavHostController` применяется функция `rememberNavController()`. Данная функция позволяет сохранить целостность стека навигации в процессе рекомпозиции компонентов пользовательского интерфейса.

```
val navController = rememberNavController()
```

Ключевые особенности NavHostController

в нем представлены методы навигации между различными пунктами назначения.

он управляет обратным стеком составных функций.

в нем представлены методы передачи данных между экранами.

NavHost

После создания контроллера навигации его необходимо назначить хосту навигации – объекту NavHost.

NavHost – это составная функция, которая используется для создания навигационного графика, определяющего структуру навигации приложения. По сути, внутри него будут находиться экраны (составная функция), которые сообщают вам, куда вам нужно перемещаться.

При вызове в NavHost передается объект NavController, компонент, который будет служить начальным экраном/начальным пунктом назначения, и граф навигации (navigation graph):

```
val navController = rememberNavController()
NavHost(navController = navController, startDestination = начальный_маршрут) {

    // Пункты назначения навигационного графа
}
```

Для передачи графа навигации применяется третий параметр – концевая лямбда. Граф навигации состоит из всех компонентов, которые должны быть доступны в качестве пунктов назначения навигации в контексте контроллера навигации. Пункты назначения добавляются в граф навигации путем вызова функции composable(), в который передается маршрут (route) и пункт назначения. Маршрут представляет обычную строку, которая однозначно идентифицирует пункт назначения в контексте текущего контроллера навигации. Пункт назначения – это компонент, который будет вызываться при выполнении навигации. Например:

```
NavHost(navController = navController, startDestination = "home") {
    composable("home") {
        Home()
    }
    composable("contact") {
        Contacts()
    }
    composable("about") {
        About()
    }
}
```

Здесь NavHost включает граф навигации, состоящий из трех пунктов назначения, где маршрут "home" настроен как начальный пункт назначения (начальный экран).

В качестве альтернативы жесткому кодированию строк маршрута в вызовах функции composable() можно определять маршруты в закрытом классе:

```
sealed class Routes(val route: String) {

    object Home : Routes("home")
    object Contacts : Routes("contact")
    object About : Routes("about")
}

NavHost(navController = navController, startDestination = Routes.Home.route)
{
```

```
composable(Routes.Home.route) { Home() }
composable(Routes.Contacts.route) { Contacts() }
composable(Routes.About.route) { About() }
}
```

Использование закрытого класса позволяет определить единую точку для управления маршрутами.

Переход к точкам назначения

Для перехода между точками назначения/экранами применяется метод `navigation()` контроллера навигации. Данный метод определяет маршрут для пункта назначения. Например, в следующем коде по нажатию на кнопку происходит переход к экрану "Contacts":

```
Button(onClick = {
    navController.navigate(Routes.Contacts.route)
}) {
    Text(text = "Contacts")
}
```

Метод `navigation()` также с помощью концевой лямбды позволяет задать параметры навигации с помощью различных функций. Так, функция `popUpTo()` позволяет очистить стек навигации вплоть до определенного элемента. Это упрощает навигацию, когда пользователь хочет вернуться не просто назад к экрану, который хранится на верхушке стека навигации, а в какое-то определенное место, например, на начальный экран. В этом случае можно было бы сделать следующим образом:

```
Button(onClick = {
    navController.navigate(Routes.Contacts.route) {
        popUpTo(Routes.Home.route)
    }
}) {
    Text(text = "To Contact Page")
}
```

Теперь, когда пользователь нажимает кнопку "To Contact Page", стек очищается вплоть до главного экрана `Home`. И после посещения экрана `Contacts` при возвращении назад пользователь перейдет к экрану `Home`.

Для определения начального пункта назначения также можно использовать метод `navController.graph.findStartDestination()`, а очистка стека вплоть до этого пункта будет представлять следующий вызов `popUpTo`:

```
popUpTo(navController.graph.findStartDestination().id)
```

Функция `popUpTo()` также может принимать дополнительные параметры. Например, параметр `inclusive` указывает, надо ли извлечь из стека навигации также и тот пункт назначения, который передан в функцию.

Так, значение `inclusive = true` в следующем коде также извлекает из стека навигации пункт назначения `Home` перед выполнением навигации:

```
Button(onClick = {
    navController.navigate(Routes.Contacts.route) {
        popUpTo(Routes.Home.route) {
            inclusive = true
        }
    }
}) {
    Text(text = "To Contact Page")
}
```

По умолчанию попытка перехода от текущего пункта назначения к самому себе помещает в стек навигации дополнительный экземпляр пункта назначения. В большинстве ситуаций такое поведение вряд ли будет желательным. Чтобы этого предотвратить, для параметра `launchSingleTop` устанавливается значение `true`:

```
Button(onClick = {
    navController.navigate(Routes.Contacts.route) {
        launchSingleTop = true
    }
}) {
    Text(text = "To Contact Page")
}
```

Если для параметров `saveState` и `restoreState` установлено значение `true`, они будут автоматически сохранять и восстанавливать состояние записей стека, когда пользователь повторно выбирает пункт назначения, который был выбран ранее.

Пример навигации

```
package com.example.helloapp
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.navigation.NavController
import androidx.navigation.compose.NavHost
import androidx.navigation.compose.composable
import androidx.navigation.compose.rememberNavController

class MainActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            Main()
        }
    }
}
```

```

    }
}
}
@Composable
fun Main() {
    val navController = rememberNavController()
    Column(Modifier.padding(8.dp)) {
        NavBar(navController = navController)
        NavHost(navController, startDestination = NavRoutes.Home.route) {
            composable(NavRoutes.Home.route) { Home() }
            composable(NavRoutes.Contacts.route) { Contacts() }
            composable(NavRoutes.About.route) { About() }
        }
    }
}
@Composable
fun NavBar(navController: NavController){
    Row(
        Modifier.fillMaxWidth().padding(bottom = 8.dp)){
        Text("Home",
            Modifier
                .weight(0.33f)
                .clickable { navController.navigate(NavRoutes.Home.route) },
            fontSize = 22.sp, color= Color(0xFF6650a4))
        Text("Contacts",
            Modifier
                .weight(0.33f)
                .clickable { navController.navigate(NavRoutes.Contacts.route) },
            fontSize = 22.sp, color= Color(0xFF6650a4))
        Text("About",
            Modifier
                .weight(0.33f)
                .clickable { navController.navigate(NavRoutes.About.route) },
            fontSize = 22.sp, color= Color(0xFF6650a4))
    }
}

@Composable
fun Home() {
    Text("Home Page", fontSize = 30.sp)
}
@Composable
fun Contacts() {
    Text("Contact Page", fontSize = 30.sp)
}
@Composable
fun About() {
    Text("About Page", fontSize = 30.sp)
}

sealed class NavRoutes(val route: String) {
    object Home : NavRoutes("home")
    object Contacts : NavRoutes("contact")
    object About : NavRoutes("about")
}

```

Итак, здесь для описания маршрутов применяется класс `NavRoutes`, который через параметр получает маршрут. В этом классе определены три возможных маршрута – `Home`, `Contacts` и `About`.

Для каждого из этих маршрутов определены соответствующие одноименные компоненты:

```
@Composable
fun Home() {
    Text("Home Page", fontSize = 30.sp)
}
@Composable
fun Contacts() {
    Text("Contact Page", fontSize = 30.sp)
}
@Composable
fun About() {
    Text("About Page", fontSize = 30.sp)
}
```

Для простоты каждый компонент просто выводит некоторый текст.

Чтобы в приложении можно было программно переходить по компонентам определена своего рода навигационная панель — компонент `NavBar` с тремя условными кнопками:

```
@Composable
fun NavBar(navController: NavController) {
    Row(
        Modifier.fillMaxWidth().padding(bottom = 8.dp) {
            Text("Home",
                Modifier
                    .weight(0.33f)
                    .clickable { navController.navigate(NavRoutes.Home.route) },
            fontSize = 22.sp, color= Color(0xFF6650a4))
            Text("Contacts",
                Modifier
                    .weight(0.33f)
                    .clickable { navController.navigate(NavRoutes.Contacts.route) },
            fontSize = 22.sp, color= Color(0xFF6650a4))
            Text("About",
                Modifier
                    .weight(0.33f)
                    .clickable { navController.navigate(NavRoutes.About.route) },
            fontSize = 22.sp, color= Color(0xFF6650a4))
        }
    )
}
```

Этот компонент извне принимает объект `NavController`, с помощью метода `navigate` которого можно перейти по определенному маршруту.

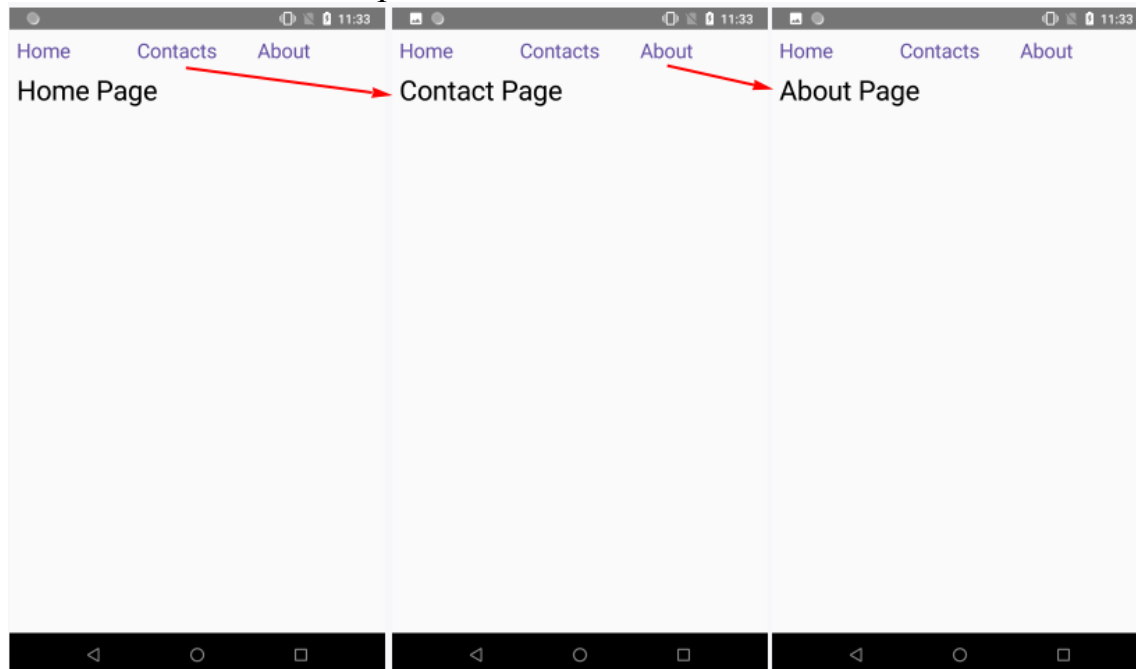
И чтобы все это объединить определен компонент `Main`:

```
@Composable
fun Main() {
    val navController = rememberNavController()
    Column(Modifier.padding(8.dp)) {
        NavBar(navController = navController)
        NavHost(navController, startDestination = NavRoutes.Home.route) {
            composable(NavRoutes.Home.route) { Home() }
            composable(NavRoutes.Contacts.route) { Contacts() }
            composable(NavRoutes.About.route) { About() }
        }
    }
}
```

Вначале с помощью функции `rememberNavController()` в нем создается объект `NavController`. Затем создается вертикальная компоновка, где в верхней

части располагается навигационная панель, а за ней идет компонент NavHost, внутри которого и разворачиваются навигационные компоненты. По умолчанию NavHost отображает компонент Home.

Таким образом в приложении можно переходить с помощью ссылок по различным компонентам в приложении:



Параметры навигации

Иногда бывает необходимо при переходе от одного экрана к другому передать аргумент в пункт назначения. Compose поддерживает передачу аргументов самых различных типов. Для этого сначала надо добавить имя аргумента к маршруту назначения:

```
NavHost(navController = navController, startDestination = Routes.Home.route) {
    composable(Routes.User.route +("/{userId}") { stackEntry ->
        val userId = stackEntry.arguments?.getString("userId")
        User(userId)
    }
}
```

В данном случае в маршрут User передается аргумент "userId". Когда приложение запускает переход к месту назначения, значение аргумента сохраняется в соответствующей записи стека навигации. Запись стека передается в качестве параметра в концевую лямбду функции composable(), откуда ее можно извлечь и передать компоненту User.

По умолчанию предполагается, что аргумент навигации имеет тип String, соответственно для его извлечения из записи стека применяется метод getString(). Чтобы передать аргументы разных типов, тип необходимо указать с помощью перечисления NavType через параметр arguments функции composable().

В следующем примере тип параметра объявлен как тип Int, соответственно для его извлечения используется метод getInt():

```
composable(
    Routes.User.route + "{userId}",
    arguments = listOf(navArgument("userId") { type = NavType.IntType })
){
    navBackStack -> User(navBackStack.arguments?.getInt("userId"))
}
```

На своей стороне компонент User должен ожидать получения аргумента `userId` через одноименный параметр:

```
@Composable
fun User(userId: String?) {
    .....
}
```

И на последнем шаге надо передать значение аргумента при вызове метода `navigation()`, например, добавив значение аргумента в конец маршрута назначения:

```
val someId = 22 // некоторое значение, которое передается аргументу навигации
Button(onClick = {
    navController.navigate(Routes.User.route + " /$someId")
}) {
    Text(text = "Show User")
}
```

В итоге при нажатии на кнопку произойдет следующая последовательность событий:

- Для текущего пункта назначения создается запись для стека
- Текущее значение `someId` сохраняется в эту запись
- Запись помещается в стек навигации
- Вызывается функция `composable()` в объявлении `NavHost`
- Концевая лямбда функции `composable()` извлекает значение аргумента из записи стека и передает его в компонент `User`

Пример использования параметров при навигации

```
package com.example.helloapp
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.navigation.NavController
import androidx.navigation.NavType
import androidx.navigation.compose.NavHost
import androidx.navigation.compose.composable
```

```
import androidx.navigation.compose.rememberNavController
import androidx.navigation.navArgument

class MainActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            Main()
        }
    }
}

sealed class UserRoutes(val route: String) {
    object Users : UserRoutes("users")
    object User : UserRoutes("user")
}

@Composable
fun Main() {
    val navController = rememberNavController()
    val employees = listOf(
        Employee(1, "Tom", 39),
        Employee(2, "Bob", 43),
        Employee(3, "Sam", 28)
    )
    NavHost(navController, startDestination = UserRoutes.Users.route) {
        composable(UserRoutes.Users.route) { Users(employees, navController) }
        composable(UserRoutes.User.route +("/{userId}",
            arguments = listOf(navArgument("userId") { type = NavType.IntType })))
    {
        stackEntry ->
        val userId = stackEntry.arguments?.getInt("userId")
        User(userId, employees)
    }
    }
}

@Composable
fun Users(data: List<Employee>, navController: NavController){
    LazyColumn {
        items(data) {
            u->
            Row(Modifier.fillMaxWidth()){
                Text(u.name,
                    Modifier.padding(8.dp).clickable {
navController.navigate("user/${u.id}") },
                    fontSize = 28.sp)
            }
        }
    }
}

@Composable
fun User(userId: Int?, data: List<Employee>){
    val user = data.find { it.id==userId }
    if(user!=null) {
        Column {
            Text("Id: ${user.id}", Modifier.padding(8.dp), fontSize = 22.sp)
            Text("Name: ${user.name}", Modifier.padding(8.dp), fontSize = 22.sp)
            Text("Age: ${user.age}", Modifier.padding(8.dp), fontSize = 22.sp)
        }
    }
    else{
        Text("User Not Found")
    }
}
}
```

```
data class Employee(val id:Int, val name:String, val age:Int)
```

Здесь для описания данных используется класс Employee с тремя свойствами, из которых свойство id – идентификатор пользователя.

Компонент User применяется для отображения одного объекта Employee. Через параметры он получает идентификатор пользователя и список пользователей:

```
@Composable
fun User(userId:Int?, data: List<Employee>){
    val user = data.find { it.id==userId }
    .....
```

Получив идентификатор, компонент извлекает из переданного списка нужного пользователя и выводит его данные на экран.

Компонент Users отображает список пользователей

```
@Composable
fun Users(data: List<Employee>, navController: NavController){
    LazyColumn {
        items(data) {
            u->
                Row(Modifier.fillMaxWidth()){
                    Text(u.name,
                        Modifier.padding(8.dp).clickable {
                            navController.navigate("user/${u.id}") },
                        fontSize = 28.sp)
                }
        }
    }
}
```

По нажатию на каждый элемент списка будет происходить переход к отображению соответствующего пользователя. И при переходе передается в качестве аргумента навигации идентификатор пользователя

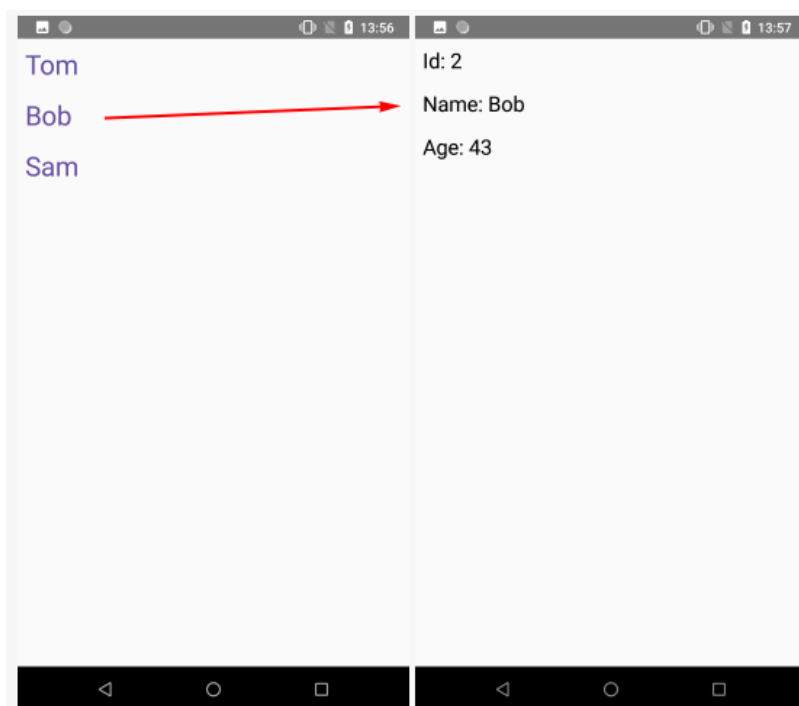
В компоненте Main создается список пользователей для отображения и разворачивается NavHost для его отображения:

```
@Composable
fun Main() {
    val navController = rememberNavController()
    val employees = listOf(
        Employee(1, "Tom", 39),
        Employee(2, "Bob", 43),
        Employee(3, "Sam", 28)
    )
    NavHost(navController, startDestination = UserRoutes.Users.route) {
        composable(UserRoutes.Users.route) { Users(employees, navController) }
        composable(UserRoutes.User.route +("/{userId})",
            arguments = listOf(navArgument("userId") { type = NavType.IntType })))
    }
    {
        stackEntry ->
        val userId = stackEntry.arguments?.getInt("userId")
        User(userId, employees)
    }
}
```

```
}  
}
```

В данном случае параметр `userId` определяется как значение типа `Int`, и для его получения применяется метод `getInt()`

Таким образом, если запустить приложение, то на экране отобразится список пользователей. При нажатии на одного из пользователей в списке произойдет переход к компоненту `User`, который отобразит информацию по данному пользователю:



Панель навигации

Для управления навигацией можно использовать стандартные встроенные компоненты типа кнопок, определять свои компоненты, однако специально для целей навигации Compose также предоставляет специальный компонент — `NavigationBar`, который представляет навигационную панель. Каждый отдельный элемент этой панели представляет компонент `NavigationBarItem`.

Реализация `NavigationBar` обычно содержит цикл `forEach`, который проходит по списку и для каждого его элемента создает отдельный компонент `BNavigationBarItem`. Для `NavigationBarItem` настраивается иконка и текст, а также обработчик `onClick` для перехода к соответствующему пункту назначения.

Пример

```
package com.example.helloapp  
import android.os.Bundle  
import androidx.activity.ComponentActivity  
import androidx.activity.compose.setContent  
import androidx.compose.foundation.layout.Column  
import androidx.compose.foundation.layout.padding  
import androidx.compose.material.icons.Icons  
import androidx.compose.material.icons.filled.Face  
import androidx.compose.material.icons.filled.Home  
import androidx.compose.material.icons.filled.Info
```

```
import androidx.compose.material3.Icon
import androidx.compose.material3.NavigationBar
import androidx.compose.material3.NavigationBarItem
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.vector.ImageVector
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.navigation.NavController
import androidx.navigation.NavGraph.Companion.findStartDestination
import androidx.navigation.compose.NavHost
import androidx.navigation.compose.composable
import androidx.navigation.compose.currentBackStackEntryAsState
import androidx.navigation.compose.rememberNavController

class MainActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            Main()
        }
    }
}

@Composable
fun Main() {
    val navController = rememberNavController()
    Column(modifier.padding(8.dp)) {
        NavHost(navController, startDestination = NavRoutes.Home.route, modifier = Modifier.weight(1f)) {
            composable(NavRoutes.Home.route) { Home() }
            composable(NavRoutes.Contacts.route) { Contacts() }
            composable(NavRoutes.About.route) { About() }
        }
        BottomNavigationBar(navController = navController)
    }
}

@Composable
fun BottomNavigationBar(navController: NavController) {
    NavigationBar {
        val backStackEntry by navController.currentBackStackEntryAsState()
        val currentRoute = backStackEntry?.destination?.route

        NavBarItems.BarItems.forEach { navItem ->
            NavigationBarItem(
                selected = currentRoute == navItem.route,
                onClick = {
                    navController.navigate(navItem.route) {
                        popUpTo(navController.graph.findStartDestination().id)
                    }
                    launchSingleTop = true
                    restoreState = true
                }
            ),
            icon = {
                Icon(imageVector = navItem.image,
                    contentDescription = navItem.title)
            },
            label = {
                Text(text = navItem.title)
            }
        }
    }
}

```

```

    )
  }
}

object NavBarItems {
    val BarItems = listOf(
       BarItem(
            title = "Home",
            image = Icons.Filled.Home,
            route = "home"
        ),
       BarItem(
            title = "Contacts",
            image = Icons.Filled.Face,
            route = "contacts"
        ),
       BarItem(
            title = "About",
            image = Icons.Filled.Info,
            route = "about"
        )
    )
}

data class BarItem(
    val title: String,
    val image: ImageVector,
    val route: String
)

@Composable
fun Home() {
    Text("Home Page", fontSize = 30.sp)
}

@Composable
fun Contacts() {
    Text("Contact Page", fontSize = 30.sp)
}

@Composable
fun About() {
    Text("About Page", fontSize = 30.sp)
}

sealed class NavRoutes(val route: String) {
    object Home : NavRoutes("home")
    object Contacts : NavRoutes("contacts")
    object About : NavRoutes("about")
}

```

Прежде всего для представления отдельного элемента панели навигации определяем класс `BarItem`:

```

data class BarItem(
    val title: String,
    val image: ImageVector,
    val route: String
)

```

Данный класс будет хранить текст ссылки навигации, иконку и маршрут для перехода к другому экрану.

Для представления набора ссылок навигации определяется объект `NavBarItems`, который содержит список из трех навигационных ссылок:

```
object NavBarItems {
    val BarItems = listOf(
       BarItem(
            title = "Home",
            image = Icons.Filled.Home,
            route = "home"
        ),
       BarItem(
            title = "Contacts",
            image = Icons.Filled.Face,
            route = "contacts"
        ),
       BarItem(
            title = "About",
            image = Icons.Filled.Info,
            route = "about"
        )
    )
}
```

Для создания самой навигационной панели определяется компонент **BottomNavigationBar**:

```
@Composable
fun BottomNavigationBar(navController: NavController) {
    NavigationBar {
        val backStackEntry by navController.currentBackStackEntryAsState()
        val currentRoute = backStackEntry?.destination?.route

        NavBarItems.BarItems.forEach { navItem ->
            NavigationBarItem(
                selected = currentRoute == navItem.route,
                onClick = {
                    navController.navigate(navItem.route) {
                        popUpTo(navController.graph.findStartDestination().id)
                    }
                    {saveState = true}
                    launchSingleTop = true
                    restoreState = true
                }
            ),
            icon = {
                Icon(imageVector = navItem.image,
                    contentDescription = navItem.title)
            },
            label = {
                Text(text = navItem.title)
            }
        }
    }
}
```

Фактически этот компонент является оберткой над **NavigationBar**, в который передается объект **NavController** для выполнения навигации.

Внутри **BottomNavigationBar** можно идентифицировать текущий маршрут с помощью метода **currentBackStackEntryAsState()** контроллера навигации. Это позволит провести некоторую стилизацию элементов панели на основе текущего маршрута:

```
val backStackEntry by navController.currentBackStackEntryAsState()
val currentRoute = backStackEntry?.destination?.route
```

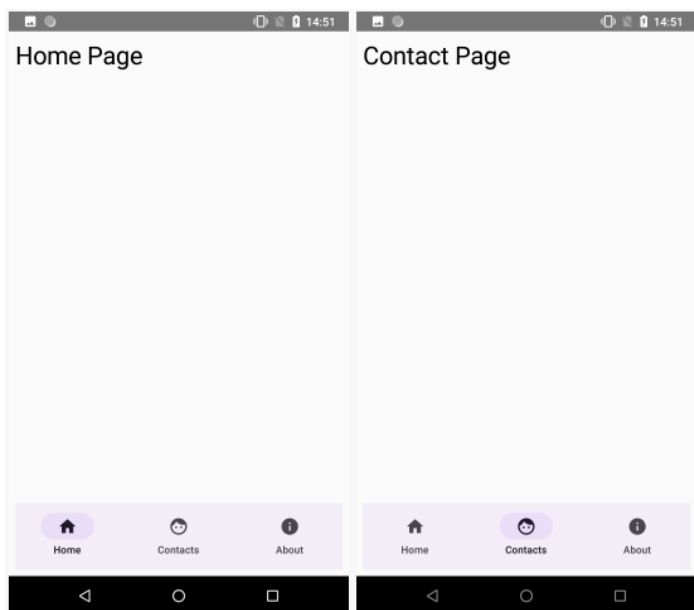
Далее осуществляется проход по всем элементам в списке и выводится каждый элемент с помощью компонента `NavigationBarItem`:

```
NavBarItems.BarItems.forEach { navItem ->
    NavigationBarItem(
        selected = currentRoute == navItem.route,
        onClick = {
            navController.navigate(navItem.route) {
                popUpTo(navController.graph.findStartDestination().id) {saveState
= true}

                launchSingleTop = true
                restoreState = true
            }
        },
        icon = {
            Icon(imageVector = navItem.image,
                contentDescription = navItem.title)
        },
        label = {
            Text(text = navItem.title)
        }
    )
}
```

Обратите внимание, что при нажатии здесь вызывается функция `popUpTo()`, чтобы гарантировать, что если пользователь нажмет кнопку "Назад", навигация вернется к начальному пункту назначения. Можно определить начальный пункт назначения, вызвав метод `findStartDestination()` в графе навигации:

```
onClick = {
    navController.navigate(navItem.route) {
        popUpTo(navController.graph.findStartDestination().id) {saveState = true}
        launchSingleTop = true
        restoreState = true
    }
},
```



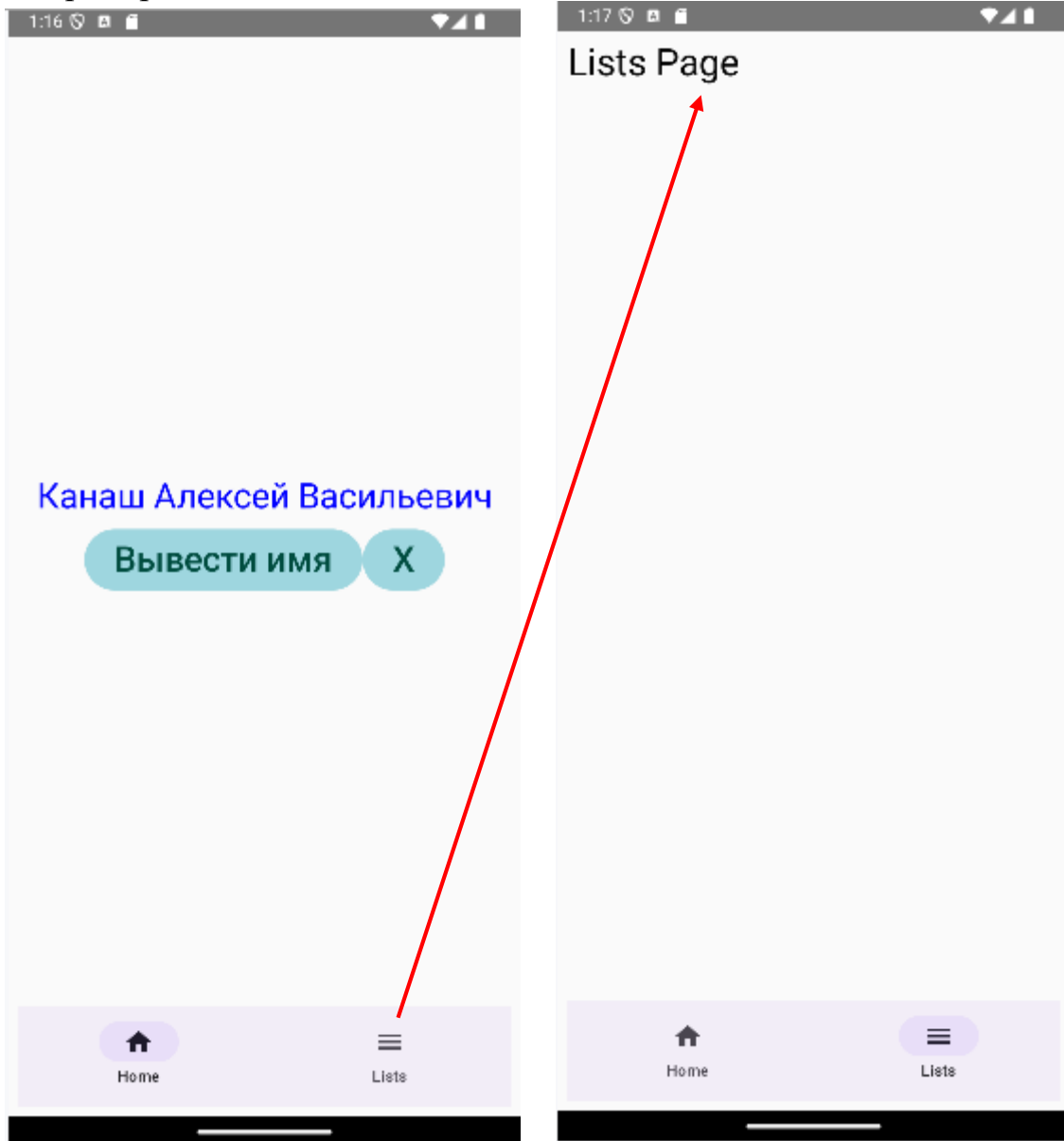
Задание на лабораторную работу

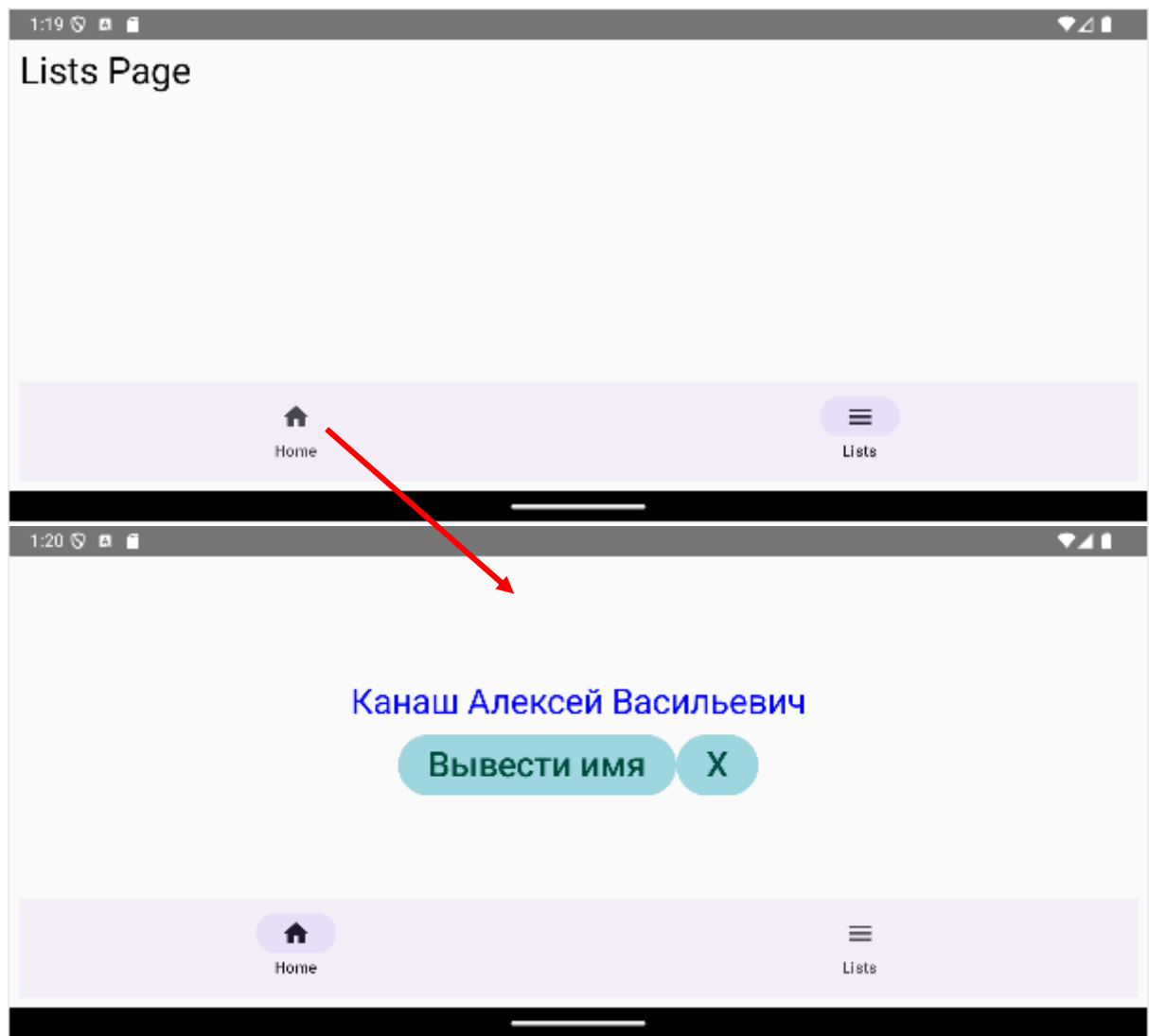
В приложение, которое было создано в первой лабораторной работе добавить навигацию между двумя пунктами:

1. Home /Main / – основной экран (созданный в ЛР № 1)
2. Lists – на экран добавить любое текстовое поле

При изменении ориентации информация должна сохраняться

Например, навигация с использованием панели навигации





Делать можно любым способом: ссылки, кнопки, панель навигации...