Uniwersytet Wrocławski

Rafał Lewandków

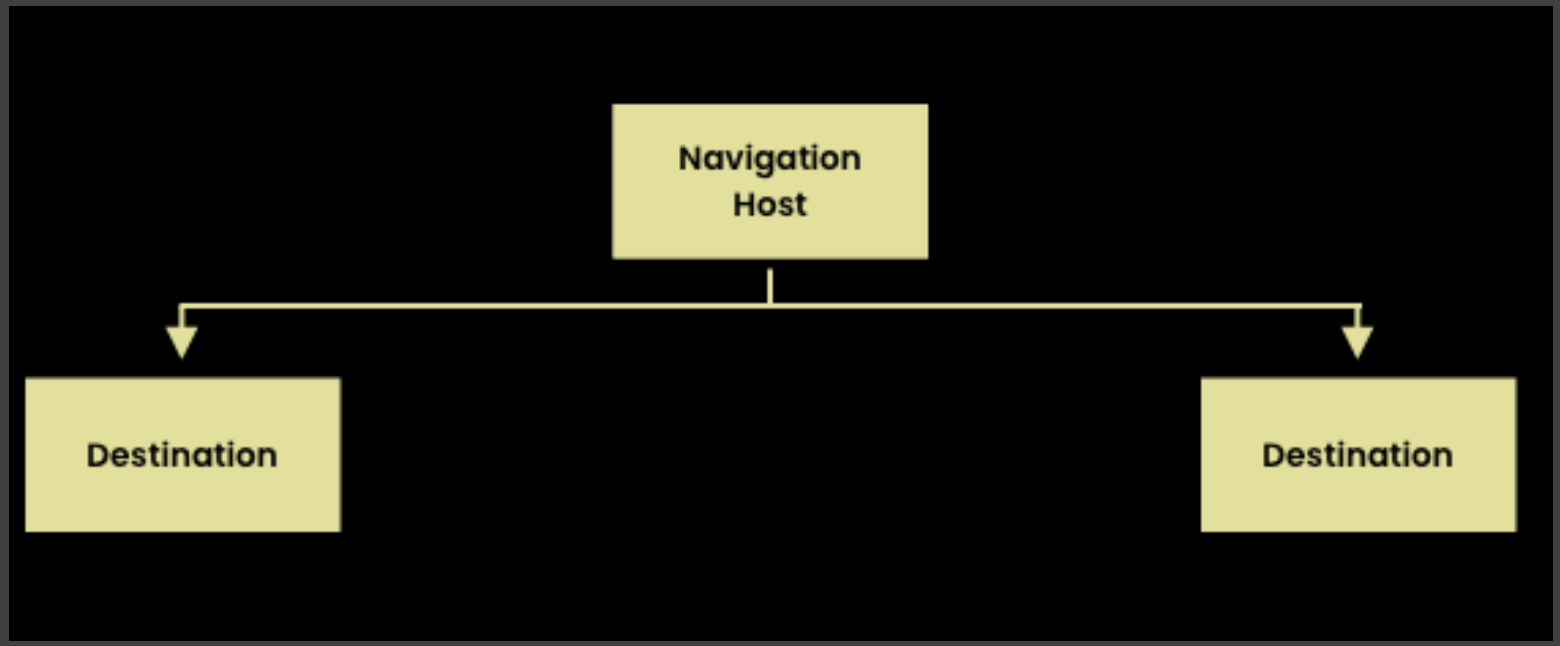
PROGRAMOWANIE URZĄDZEŃ MOBILNYCH

WYKŁAD 7

Jetpack Compose

- Compose Navigation
- Definicja ekranów
- Scaffold
- BottomNavigation

Jetpack Compose



Jetpack Compose

```
sealed class Screens(val route: String) {
    object MainScreen : Screens("main_screen")
    object SecondScreen : Screens("second_screen")
}
```

Jetpack Compose

```
sealed class Screens(val route: String) {
    object MainScreen : Screens("main_screen")
    object SecondScreen : Screens("second_screen")
}
```

```
fun MainScreen(onSecondScreen: () -> Unit) {
    Column(
        Modifier.fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {
        Text("Home Screen")
        Spacer(modifier = Modifier.height(8.dp))
        Button(onClick = onSecondScreen) {
            Text("Go to Second Screen")
        }
    }
}
```

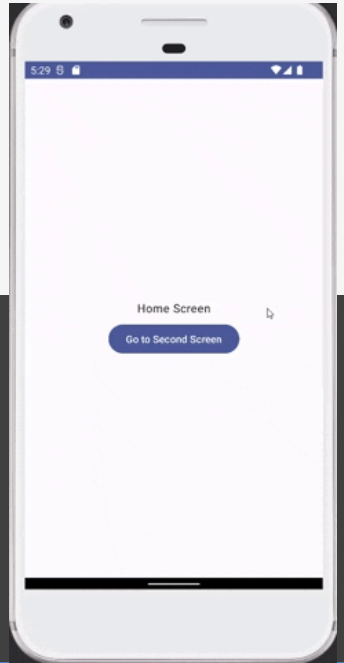
```
@Composable
fun SecondScreen(onHome: () -> Unit) {
    Column(
        Modifier.fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {
        Text("Second Screen")
        Spacer(modifier = Modifier.height(8.dp))
        Button(onClick = onHome) { Text("Go back to Main Screen") }
    }
}
```

Jetpack Compose

```
sealed class Screens(val route: String) {
    object MainScreen : Screens("main_screen")
    object SecondScreen : Screens("second_screen")
}
```

```
@Composable
fun Navigation() {
    val navController = rememberNavController()
    NavHost(navController = navController, startDestination = Screens.MainScreen.route) {
        composable(route = Screens.MainScreen.route){
            MainScreen{navController.navigate(Screens.SecondScreen.route)}
        }

        composable(route = Screens.SecondScreen.route){
            SecondScreen {navController.popBackStack()}
        }
    }
}
```



Jetpack Compose

```
sealed class Screens(val route: String) {
    object MainScreen : Screens("main_screen")
    object SecondScreen : Screens("second_screen")
}
```

```
composable(route = Screens.MainScreen.route){
    val arg = 5
    MainScreen{navController.navigate(Screens.SecondScreen.route + "/$arg")}
}
```

Uniwersytet Wrocławski

Jetpack Compose

```
sealed class Screens(val route: String) {
    object MainScreen : Screens("main_screen")
    object SecondScreen : Screens("second_screen")
}
```

```
composable(route = Screens.MainScreen.route){
    val arg = 5
    MainScreen{navController.navigate(Screens.SecondScreen.route + "/$arg")}
}
```

```
composable(route = Screens.SecondScreen.route + "/{arg}"){
    val arg = it.arguments?.getString("arg")
    SecondScreen(arg) {navController.popBackStack()}
}
```

Jetpack Compose

```
sealed class Screens(val route: String) {
    object MainScreen : Screens("main_screen")
    object SecondScreen : Screens("second_screen")
}
```

```
composable(route = Screens.MainScreen.route){
    val arg = 5
    MainScreen{navController.navigate(Screens.SecondScreen.route + "/$arg")}
}
```

```
composable(route = Screens.SecondScreen.route + "/{arg}"){
    val arg = it.arguments?.getString("arg")
    SecondScreen(arg) {navController.popBackStack()}
}
```

```
@Composable
fun SecondScreen(arg: String?, onHome: () -> Unit) {
    Column(
        ...
    ) {
        Text("Second Screen. Argument: $arg")
        ...
    }
}
```

Uniwersytet Wrocławski

Scaffold

```
@Composable
fun ScaffoldDemo() {
    Scaffold(
            scaffoldState = scaffoldState,
            topBar =
            floatingActionButtonPosition =
            floatingActionButton =
            drawerContent =
            content =
            bottomBar =
    )                                    ) }
}
```

Uniwersytet Wrocławski

BottomBar

```
sealed class Screens(val route: String) {
    object HomeScreen : Screens("home")
    object FirstScreen : Screens("first")
    object SecondScreen : Screens("second")
}
```

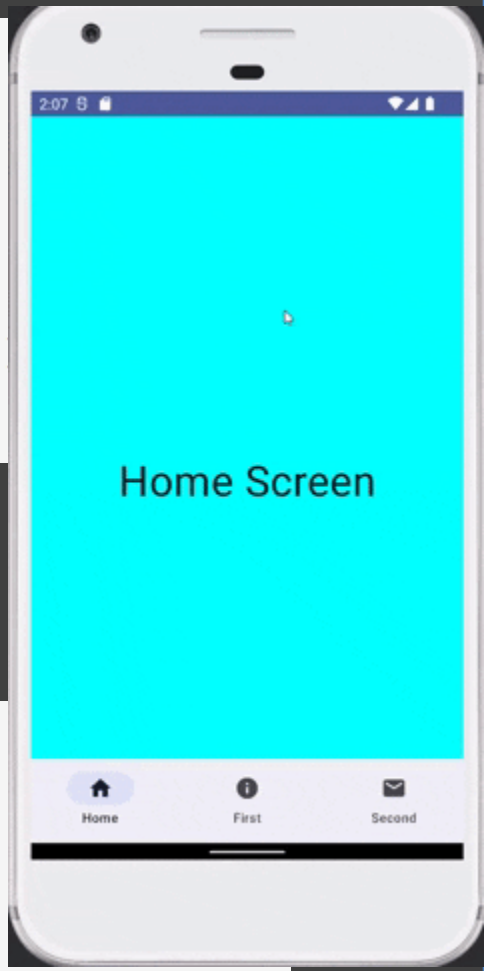
```
sealed class BottomBar(
    val route: String,
    val title: String,
    val icon: ImageVector
) {
    object Home : BottomBar(Screens.HomeScreen.route, "Home", Icons.Default.Home)
    object First : BottomBar(Screens.FirstScreen.route, "First", Icons.Default.Info)
    object Second : BottomBar(Screens.SecondScreen.route, "Second", Icons.Default.Email)
}
```

```
@SuppressLint("UnusedMaterial3ScaffoldPaddingParameter")
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun Navigation(){
    val navController = rememberNavController()
    Scaffold(
        bottomBar = { BottomMenu(navController = navController)},
        content = { BottomNavGraph(navController = navController) }
    )
}
```

Uniwersytet Wrocławski

BottomBar

```
@SuppressLint("UnusedMaterial3ScaffoldPaddingParameter")
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun Navigation(){
    val navController = rememberNavController()
    Scaffold(
        bottomBar = { BottomMenu(navController = navController)},
        content = { BottomNavGraph(navController = navController)
    )
}
```

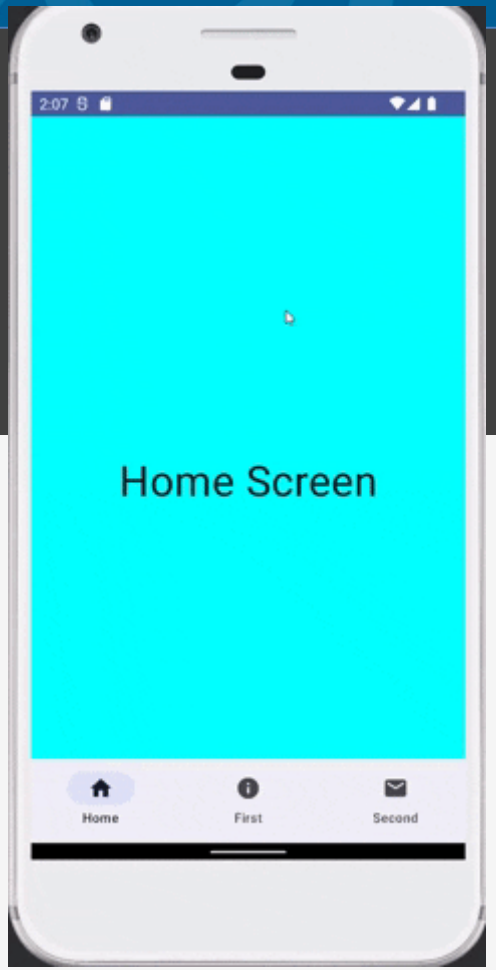



```
@Composable
fun BottomNavGraph(navController: NavHostController){
    NavHost(
        navController = navController,
        startDestination = Screens.HomeScreen.route
    ) {
        composable(route = Screens.HomeScreen.route){ HomeScreen() }
        composable(route = Screens.FirstScreen.route){ FirstScreen() }
        composable(route = Screens.SecondScreen.route){ SecondScreen() }
    }
}
```

Uniwersytet Wrocławski

BottomBar

```
@SuppressLint("UnusedMaterial3ScaffoldPaddingParameter")
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun Navigation(){
    val navController = rememberNavController()
    Scaffold(
        bottomBar = { BottomMenu(navController = navController)},
        content = { BottomNavGraph(navController = navController) }
    )
}
```



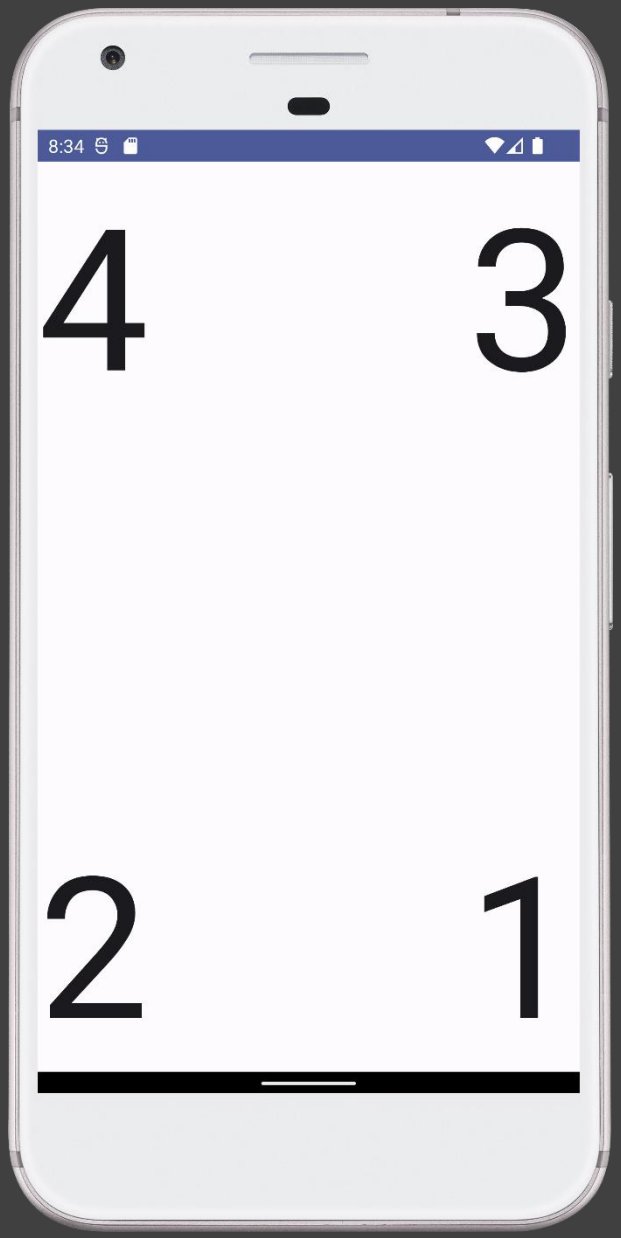

```
@Composable
fun BottomMenu(navController: NavHostController){
    val screens = listOf(
        BottomBar.Home, BottomBar.First, BottomBar.Second
    )

    val navBackStackEntry by navController.currentBackStackEntryAsState()
    val currentDestination = navBackStackEntry?.destination

    NavigationBar{
        screens.forEach{screen ->
            NavigationBarItem(
                label = { Text(text = screen.title)},
                icon = {Icon(imageVector = screen.icon, contentDescription = "icon")},
                selected = currentDestination?.hierarchy?.any { it.route == screen.route } == true,
                onClick = {navController.navigate(screen.route)}
            )
        }
    }
}
```

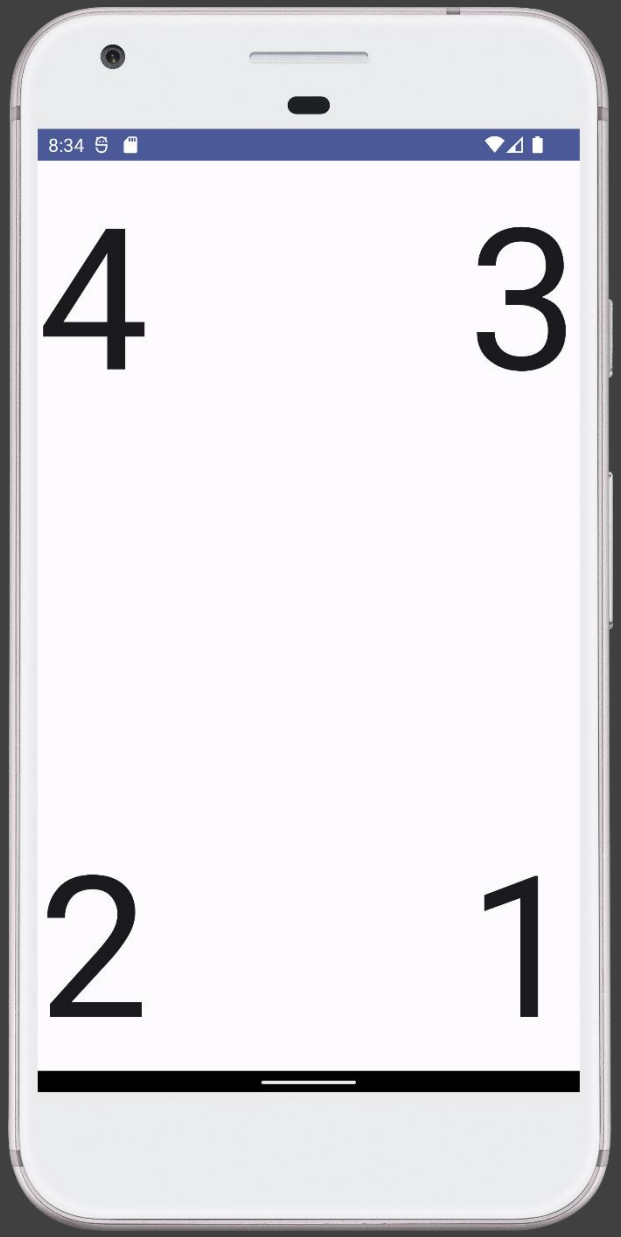
Uniwersytet Wrocławski

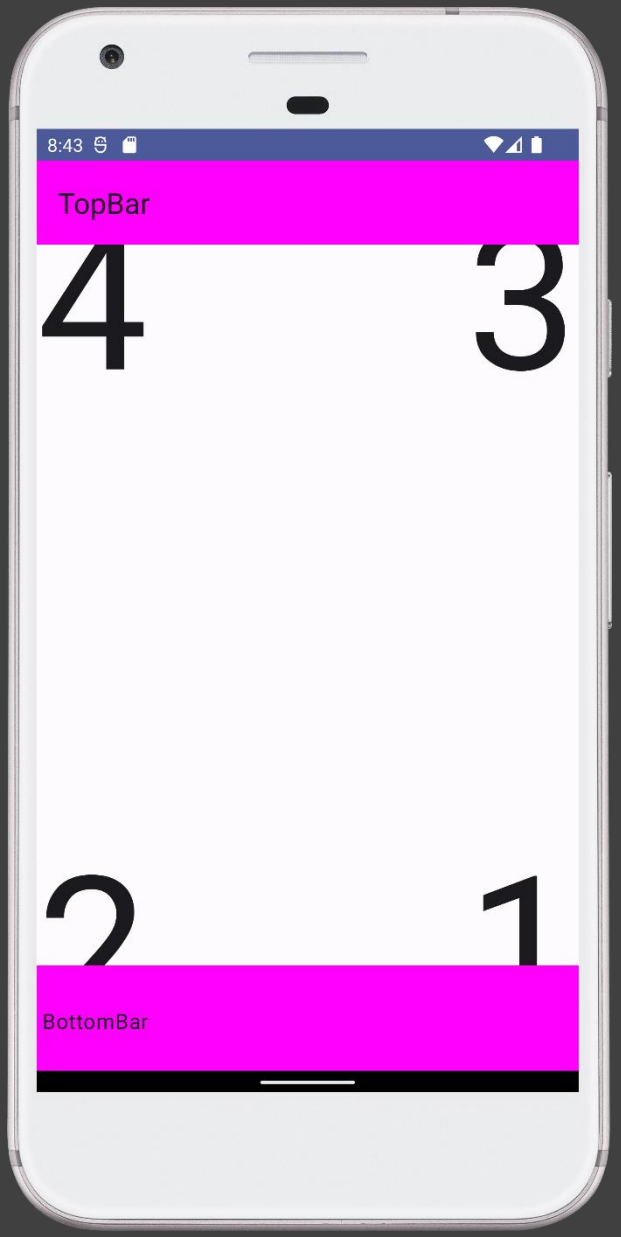
BottomBar



Uniwersytet Wrocławski

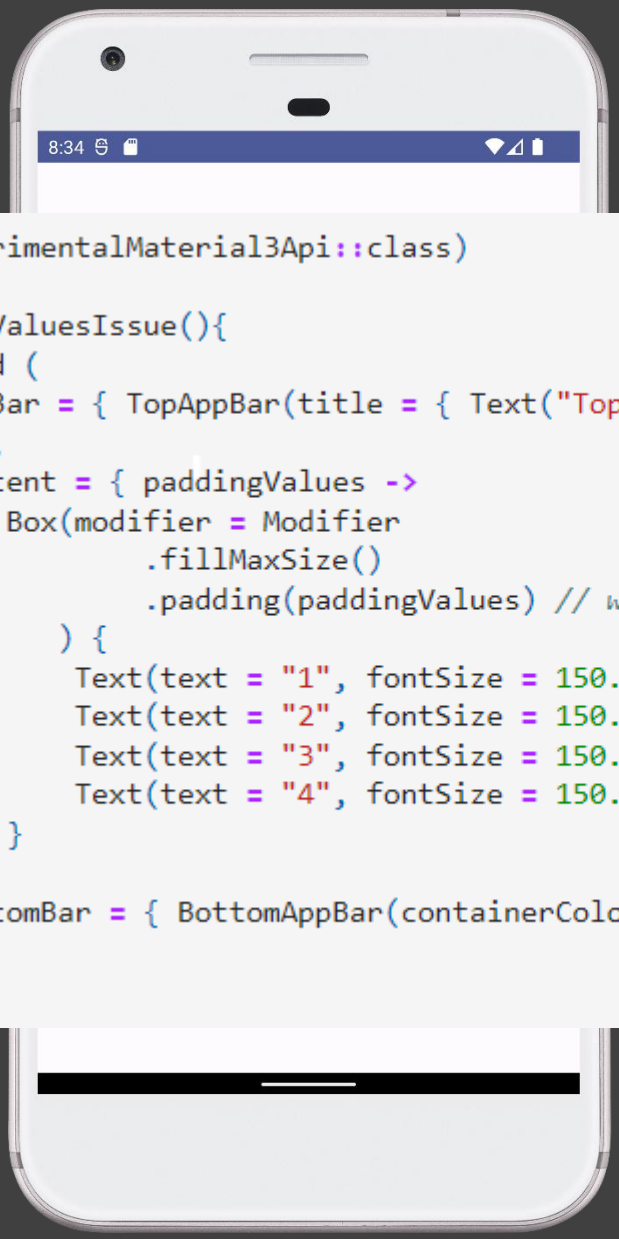
BottomBar





Uniwersytet Wrocławski

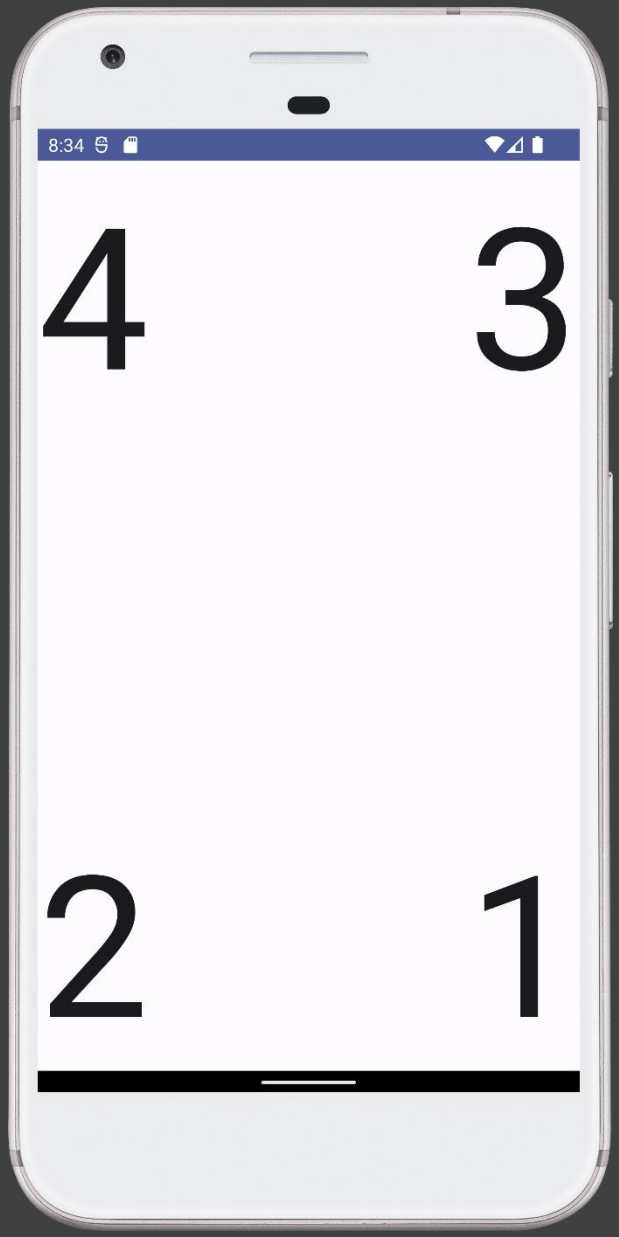
BottomBar

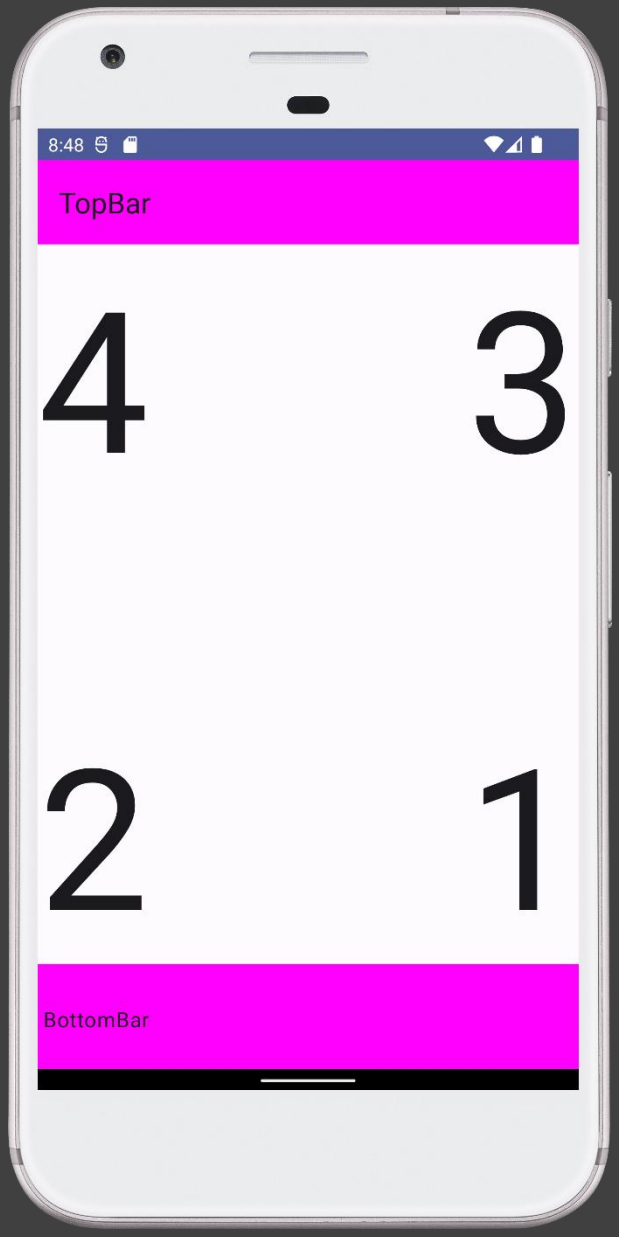



```
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun PaddingValuesIssue(){
    Scaffold (
        topBar = { TopAppBar(title = { Text("TopBar") }, colors = TopAppBarDefaults.smallTopAppBarColors(con
        ) },
        content = { paddingValues ->
            Box(modifier = Modifier
                    .fillMaxSize()
                    .padding(paddingValues) // wykorzystujemy automatycznie wygenerowaną wartość marginesów
                ) {
                 Text(text = "1", fontSize = 150.sp, modifier = Modifier.align(Alignment.BottomEnd))
                 Text(text = "2", fontSize = 150.sp, modifier = Modifier.align(Alignment.BottomStart))
                 Text(text = "3", fontSize = 150.sp, modifier = Modifier.align(Alignment.TopEnd))
                 Text(text = "4", fontSize = 150.sp, modifier = Modifier.align(Alignment.TopStart))
            }
        },
        bottomBar = { BottomAppBar(containerColor = Color.Magenta) { Text(text = "BottomBar") }}
    )
}
```

Uniwersytet Wrocławski

BottomBar





Uniwersytet Wrocławski

Pager + Tabs

```
@OptIn(ExperimentalFoundationApi::class)
@Composable
fun MainLayout(){
    val data = listOf(
        "Tab 1" to Icons.Filled.Home,
        "Tab 2" to Icons.Filled.Person,
        "Tab 3" to Icons.Filled.Phone,
        "Tab 4" to Icons.Filled.Email,
    )

    val pagerState = rememberPagerState()
    val coroutineScope = rememberCoroutineScope()

    Column {
        Tabs(pagerState = pagerState, coroutineScope = coroutineScope, data = data)
        Pages(pagerState = pagerState, data = data)
    }
}
```

Uniwersytet Wrocławski

Pager + Tabs

```
@OptIn(ExperimentalFoundationApi::class)
@Composable
fun MainLayout(){
    val data = listOf(
        "Tab 1" to Icons.Filled.Home,
        "Tab 2" to Icons.Filled.Person,
        "Tab 3" to Icons.Filled.Phone,
        "Tab 4" to Icons.Filled.Email,
    )

    val pagerState = rememberPagerState()
    val coroutineScope = rememberCoroutineScope()

    Column {
        Tabs(pagerState = pagerState, coroutineScope = coroutineScope, data = data)
        Pages(pagerState = pagerState, data = data)
    }
}
```

```
@OptIn(ExperimentalFoundationApi::class)
@Composable
fun Tabs(pagerState: PagerState, coroutineScope: CoroutineScope, data: List<Pair<String, ImageVector>>){
    TabRow(
        selectedTabIndex = pagerState.currentPage,
    ) {
        data.forEachIndexed { index, pair ->
            Tab(
                selected = pagerState.currentPage == index,
                onClick = { coroutineScope.launch { pagerState.animateScrollToPage(index)}},
                text = { Text(text = pair.first) },
                icon = {Icon(imageVector = pair.second, contentDescription = null)}
            )
        }
    }
}
```

Uniwersytet Wrocławski

Pager + Tabs

```
@OptIn(ExperimentalFoundationApi::class)
@Composable
fun MainLayout(){
    val data = listOf(
        "Tab 1" to Icons.Filled.Home,
        "Tab 2" to Icons.Filled.Person,
        "Tab 3" to Icons.Filled.Phone,
        "Tab 4" to Icons.Filled.Email,
    )

    val pagerState = rememberPagerState()
    val coroutineScope = rememberCoroutine

    Column {
        Tabs(pagerState = pagerState, corc
        Pages(pagerState = pagerState, dat
    }
}
```

```
@OptIn(ExperimentalFoundationApi::class)
@Composable
fun Pages(pagerState: PagerState, data: List<Pair<String, ImageVector>>) {
    HorizontalPager(
        state = pagerState,
        modifier = Modifier
            .fillMaxSize()
            .wrapContentSize(Alignment.Center),
        pageCount = data.size,
        pageSize = PageSize.Fill
    ) { index ->
        Column(
            modifier = Modifier.fillMaxSize(),
            verticalArrangement = Arrangement.Center,
            horizontalAlignment = Alignment.CenterHorizontally
        ) {
            Text(
                text = data[index].first,
            )
        }
    }
}
```

Uniwersytet Wrocławski

Pager + Tabs

```
@OptIn(ExperimentalFoundationApi::class)
@Composable
fun MainLayout(){
    val data = listOf(
        "Tab 1" to Icons.Filled.Home,
        "Tab 2" to Icons.Filled.Person,
        "Tab 3" to Icons.Filled.Phone,
        "Tab 4" to Icons.Filled.Email,
    )

    val pagerState = rememberPagerState()
    val coroutineScope = rememberCoroutine

    Column {
        Tabs(pagerState = pagerState, corc
        Pages(pagerState = pagerState, dat
    }
}
```

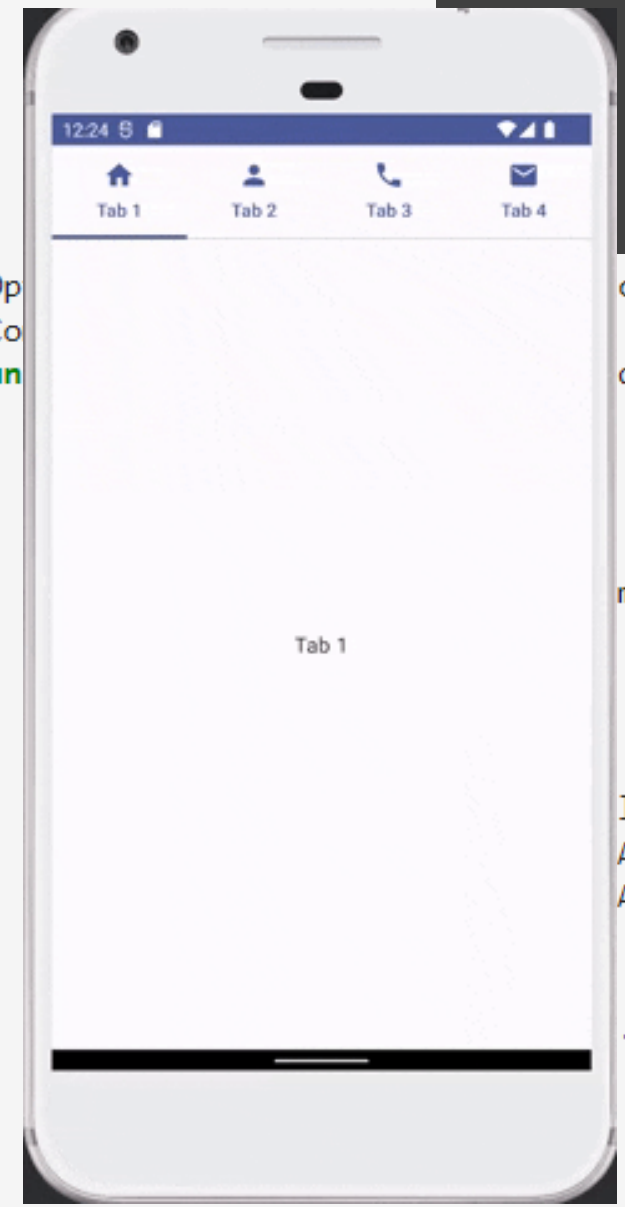
```
@OptIn(ExperimentalFoundationApi::class)
@Composable
fun Pages(pagerState: PagerState, data: List<Pair<String, ImageVector>>) {
    HorizontalPager(
        state = pagerState,
        modifier = Modifier
            .fillMaxSize()
            .wrapContentSize(Alignment.Center),
        pageCount = data.size,
        pageSize = PageSize.Fill
    ) { index ->
        Column(
            modifier = Modifier.fillMaxSize(),
            verticalArrangement = Arrangement.Center,
            horizontalAlignment = Alignment.CenterHorizontally
        ) {
            Text(
                text = data[index].first,
            )
        }
    }
}
```

Pager + Tabs

```
@OptIn(ExperimentalFoundationApi::class)
@Composable
fun MainLayout(){
    val data = listOf(
        "Tab 1" to Icons.Filled.Home,
        "Tab 2" to Icons.Filled.Person,
        "Tab 3" to Icons.Filled.Phone,
        "Tab 4" to Icons.Filled.Email,
    )

    val pagerState = rememberPagerState()
    val coroutineScope = rememberCoroutine

    Column {
        Tabs(pagerState = pagerState, corc
        Pages(pagerState = pagerState, dat
    }
}
```




```
@Op
@Co
fun
}
```

```
class)

data: List<Pair<String, ImageVector>>) {


ment.Center),


llMaxSize(),
Arrangement.Center,
Alignment.CenterHorizontally


.first,
```