Навигируемся в Compose

Игорь Кареньков

Игорь Кареньков

6 лет в android разработке

TeamLead Mobile-Core @ hh.ru

В свободное время лазаю по скалам 🧗

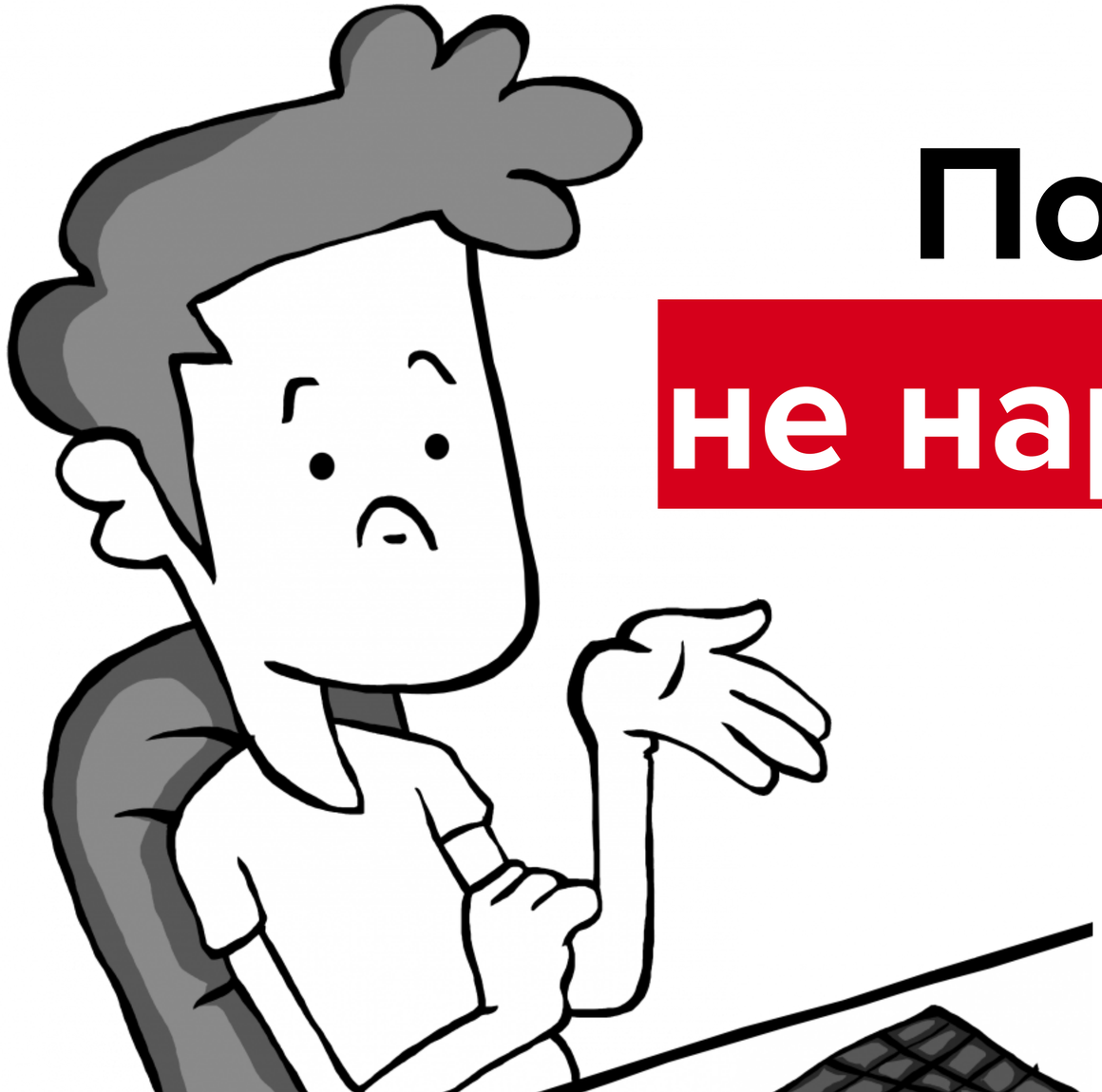
@karenkovigor

KarenkovID

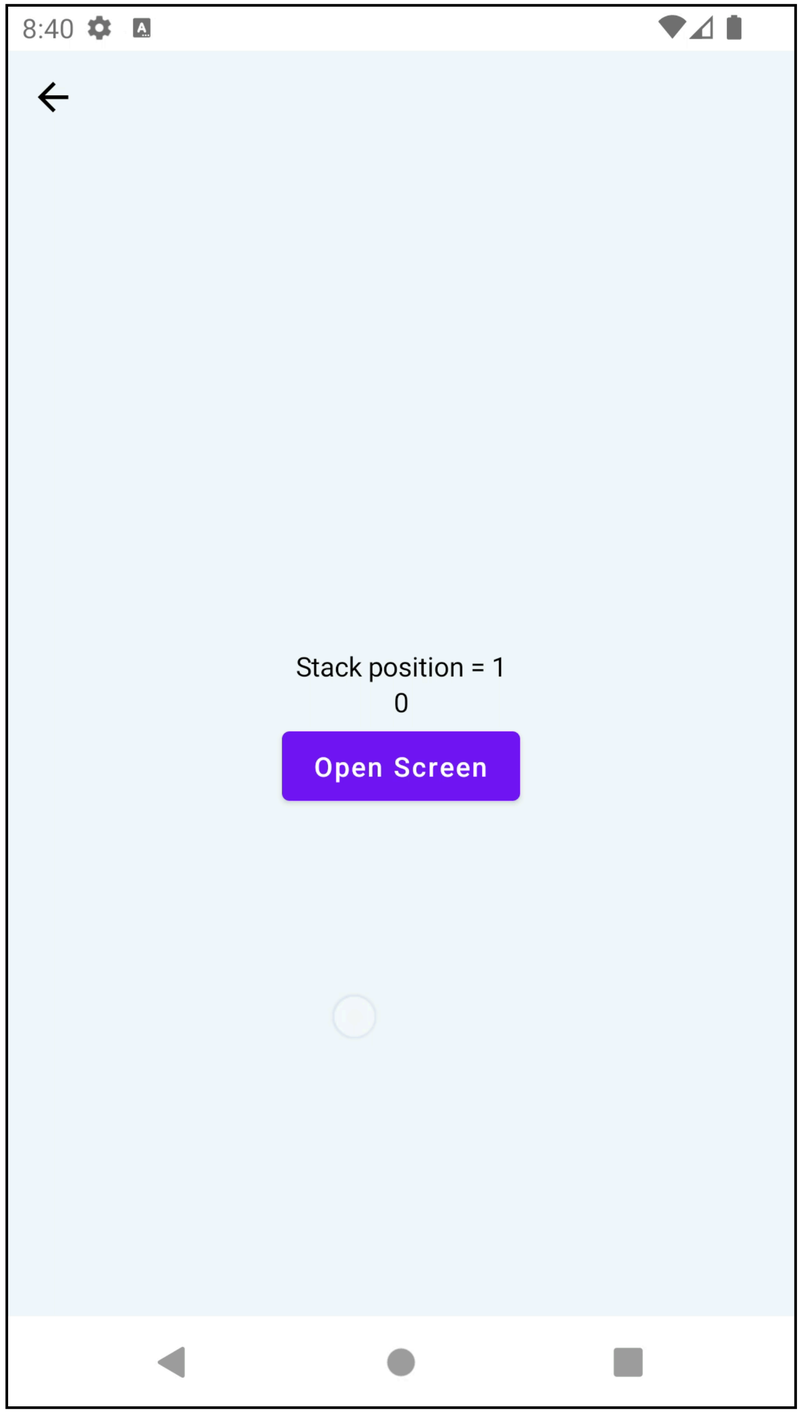
О чём сегодня будем говорить?

1. Зачем нам библиотеки навигации в Compose?
2. Что мы хотим от навигации?
3. Краткий обзор библиотек навигации
4. Что выбрали?
5. Итоги

Ссылка на слайды



Почему бы тупо не нарисовать state?


8:40
Stack position = 1
0
Open Screen

```
@Parcelize
internal class ScreenState(
    val screenPos: Int,
) : Parcelable {
    @Composable
    fun Content() {
        val navigateForward = LocalNavigateForward.current
        ScreenWithButtons(
            buttons = listOf("Open Screen" to navigateForward),
            screenTitle = "Stack position = $screenPos",
            modifier = Modifier
                .randomBackground()
                .fillMaxSize()
        )
    }
}
```

```
@Parcelize
internal class ScreenState(
    val screenPos: Int,
) : Parcelable {
    @Composable
    fun Content() {
        val navigateForward = LocalNavigateForward.current
        ScreenWithButtons(
            buttons = listOf("Open Screen" to navigateForward),
            screenTitle = "Stack position = $screenPos",
            modifier = Modifier
                .randomBackground()
                .fillMaxSize()
        )
    }
}
```

```
@Parcelize
internal class ScreenState(
    val screenPos: Int,
) : Parcelable {
    @Composable
    fun Content() {
        val navigateForward = LocalNavigateForward.current
        ScreenWithButtons(
            buttons = listOf("Open Screen" to navigateForward),
            screenTitle = "Stack position = $screenPos",
            modifier = Modifier
                .randomBackground()
                .fillMaxSize()
        )
    }
}
```

```
@Parcelize
internal class ScreenState(
    val screenPos: Int,
) : Parcelable {
    @Composable
    fun Content() {
        val navigateForward = LocalNavigateForward.current
        ScreenWithButtons(
            buttons = listOf("Open Screen" to navigateForward),
            screenTitle = "Stack position = $screenPos",
            modifier = Modifier
.composed {
    val backgroundColorInt = rememberSaveable { Random.nextInt() }
    val backgroundColor = remember { Color(backgroundColorInt) }
    then(Modifier.background(backgroundColor))
}
                .fillMaxSize()
        )
    }
```

```kotlin
@Parcelize
internal data class NavigationState<T : Parcelable>(
    val stack: List<T>,
) : Parcelable
```

```
@Composable
internal fun SelfMadeNavigation() {
    var navigationState: NavigationState<ScreenState> by rememberSaveable {
        mutableStateOf(NavigationState(listOf(ScreenState(1))))
    }
    ...
}
```

```
@Composable
internal fun SelfMadeNavigation() {
    var navigationState: NavigationState<ScreenState> by rememberSaveable {
        mutableStateOf(NavigationState(listOf(ScreenState(1))))
    }
    ...
}
```

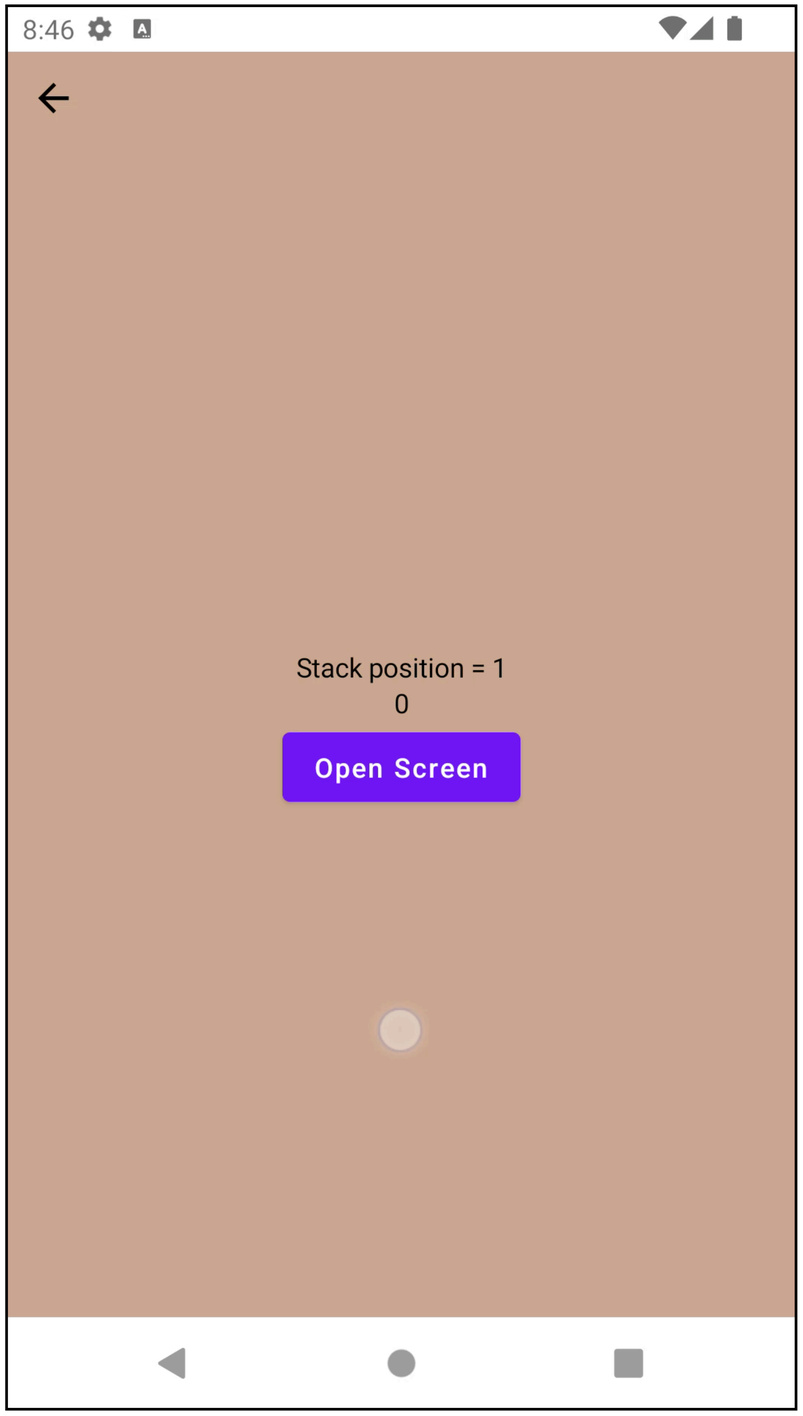
```
@Composable
internal fun SelfMadeNavigation() {
    var navigationState: NavigationState<ScreenState> by rememberSaveable {
        mutableStateOf(NavigationState(listOf(ScreenState(1))))
    }
    BackHandler(enabled = navigationState.stack.size > 1) {
        navigationState = navigationState.copy(
            stack = navigationState.stack.dropLast(1)
        )
    }
    ...
}
```

```
@Composable
internal fun SelfMadeNavigation() {
    var navigationState: NavigationState<ScreenState> by rememberSaveable {
        mutableStateOf(NavigationState(listOf(ScreenState(1))))
    }
    BackHandler(enabled = navigationState.stack.size > 1) {
        navigationState = navigationState.copy(
            stack = navigationState.stack.dropLast(1)
        )
    }
    val renderScreen by remember {
        derivedStateOf { navigationState.stack.last() }
    }
    ...
}
```

```
@Composable
internal fun SelfMadeNavigation() {
    ...
    val renderScreen by remember {
        derivedStateOf { navigationState.stack.last() }
    }
    renderScreen.Content()
}
```

```
@Composable
internal fun SelfMadeNavigation() {
    ...
    val renderScreen by remember {
        derivedStateOf { navigationState.stack.last() }
    }
    CompositionLocalProvider(
        LocalNavigateForward provides {
            navigationState = navigationState.copy(
                stack = navigationState.stack +
                    ScreenState(renderScreen.screenPos + 1)
            )
        }
    ) {
        renderScreen.Content()
    }
}
```

```
@Composable
internal fun SelfMadeNavigation() {
    ...
    val renderScreen by remember {
        derivedStateOf { navigationState.stack.last() }
    }
    CompositionLocalProvider(
        LocalNavigateForward provides {
            navigationState = navigationState.copy(
                stack = navigationState.stack +
                    ScreenState(renderScreen.screenPos + 1)
            )
        }
    ) {
        renderScreen.Content()
    }
}
```


8:46
Stack position = 1
0
Open Screen

rememberSaveable не работают*


Как работает
rememberSaveable?

```
@Composable
fun <T : Any> rememberSaveable(
    vararg inputs: Any?,
    saver: Saver<T, out Any> = autoSaver(),
    key: String? = null,
    init: () -> T
): T {
    // key is the one provided by the user or the one generated by the
compose runtime
    val finalKey = if (!key.isNullOrEmpty()) {
        key
    } else {
        currentCompositeKeyHash.toString(MaxSupportedRadix)
    }
    @Suppress("UNCHECKED_CAST")
    (saver as Saver<T, Any>)

    val registry = LocalSaveableStateRegistry.current
    // value is restored using the registry or created via [init] lambda
```

```
@Composable
fun <T : Any> rememberSaveable(
    vararg inputs: Any?,
    saver: Saver<T, out Any> = autoSaver(),
    key: String? = null,
    init: () -> T
): T {
    // key is the one provided by the user or the one generated by the
compose runtime
    val finalKey = if (!key.isNullOrEmpty()) {
        key
    } else {
        currentCompositeKeyHash.toString(MaxSupportedRadix)
    }
    @Suppress("UNCHECKED_CAST")
    (saver as Saver<T, Any>)

    val registry = LocalSaveableStateRegistry.current
    // value is restored using the registry or created via [init] lambda
```

rememberSaveable

```
override fun SaveableStateProvider(key: Any, content: @Composable () -> Unit) {
    ReusableContent(key) {
        val registryHolder = remember {
            require(parentSaveableStateRegistry?.canBeSaved(key) ?: true) {
                "Type of the key $key is not supported. On Android you can only
use types " +
                        "which can be stored inside the Bundle."
            }
            RegistryHolder(key)
        }
        CompositionLocalProvider(
            LocalSaveableStateRegistry provides registryHolder.registry,
            content = content
        )
```

rememberSaveable

```
override fun SaveableStateProvider(key: Any, content: @Composable () -> Unit) {
    ReusableContent(key) {
        val registryHolder = remember {
            require(parentSaveableStateRegistry?.canBeSaved(key) ?: true) {
                "Type of the key $key is not supported. On Android you can only
use types " +
                        "which can be stored inside the Bundle."
            }
            RegistryHolder(key)
        }
        CompositionLocalProvider(
            LocalSaveableStateRegistry provides registryHolder.registry,
            content = content
        )
```

rememberSaveable

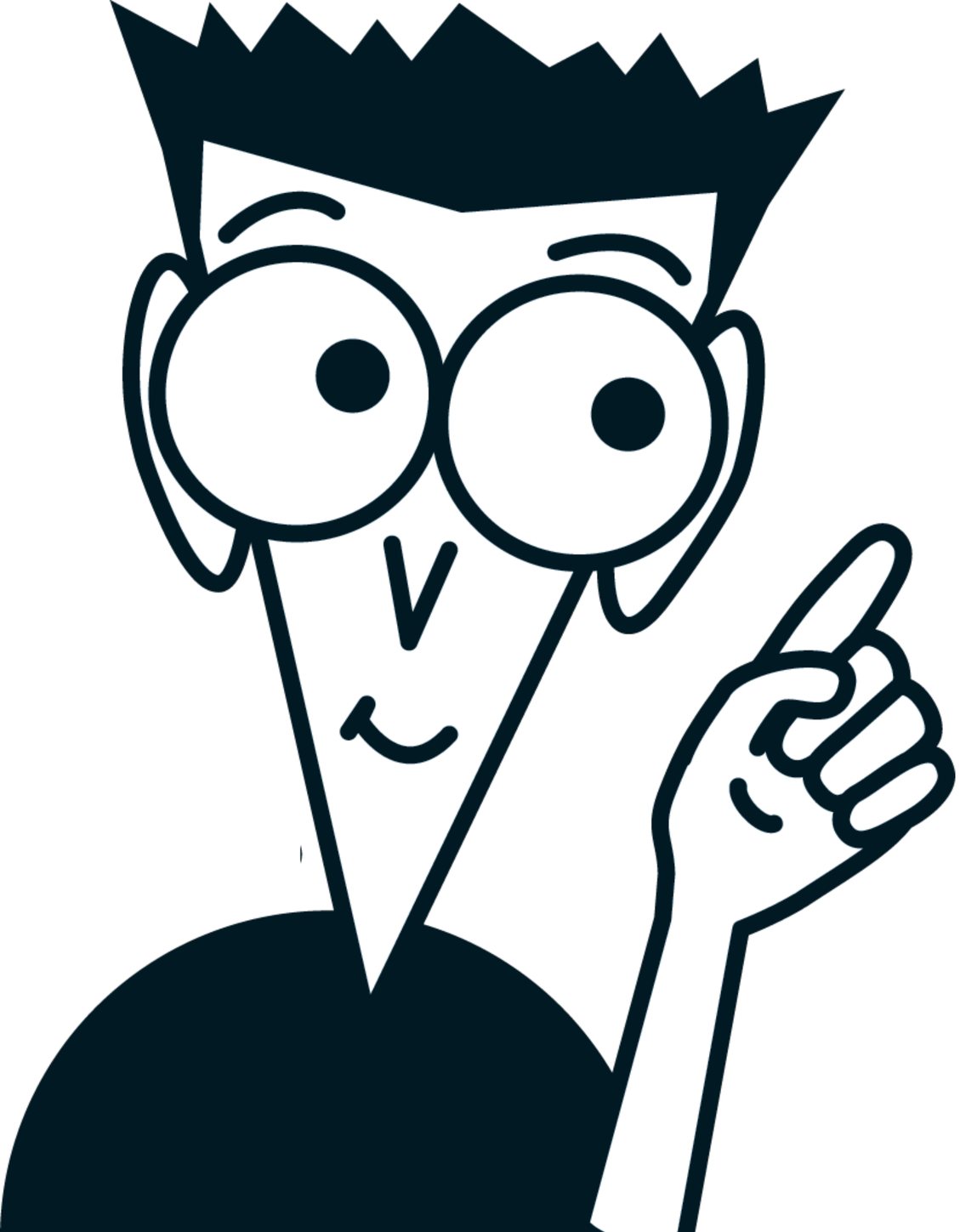
```
interface SaveableStateHolder {
    /**
     * Put your content associated with a [key] inside the [content]. This will automatically
     * save all the states defined with [rememberSaveable] before disposing the content and will
     * restore the states when you compose with this key again.
     *
     * @param key to be used for saving and restoring the states for the subtree. Note that on
     * Android you can only use types which can be stored inside the Bundle.
     */
    @Composable
    fun SaveableStateProvider(key: Any, content: @Composable () -> Unit)

    /**
     * Removes the saved state associated with the passed [key].
     */
    fun removeState(key: Any)
}
```

rememberSaveable

```
interface SaveableStateHolder {
    /**
     * Put your content associated with a [key] inside the [content]. This will automatically
     * save all the states defined with [rememberSaveable] before disposing the content and will
     * restore the states when you compose with this key again.
     *
     * @param key to be used for saving and restoring the states for the subtree. Note that on
     * Android you can only use types which can be stored inside the Bundle.
     */
    @Composable
    fun SaveableStateProvider(key: Any, content: @Composable () -> Unit)

    /**
     * Removes the saved state associated with the passed [key].
     */
    fun removeState(key: Any)
}
```

rememberSaveable

```
interface SaveableStateHolder {
    /**
     * Put your content associated with a [key] inside the [content]. This will automatically
     * save all the states defined with [rememberSaveable] before disposing the content and will
     * restore the states when you compose with this key again.
     *
     * @param key to be used for saving and restoring the states for the subtree. Note that on
     * Android you can only use types which can be stored inside the Bundle.
     */
    @Composable
    fun SaveableStateProvider(key: Any, content: @Composable () -> Unit)

    /**
     * Removes the saved state associated with the passed [key].
     */
    fun removeState(key: Any)
}
```

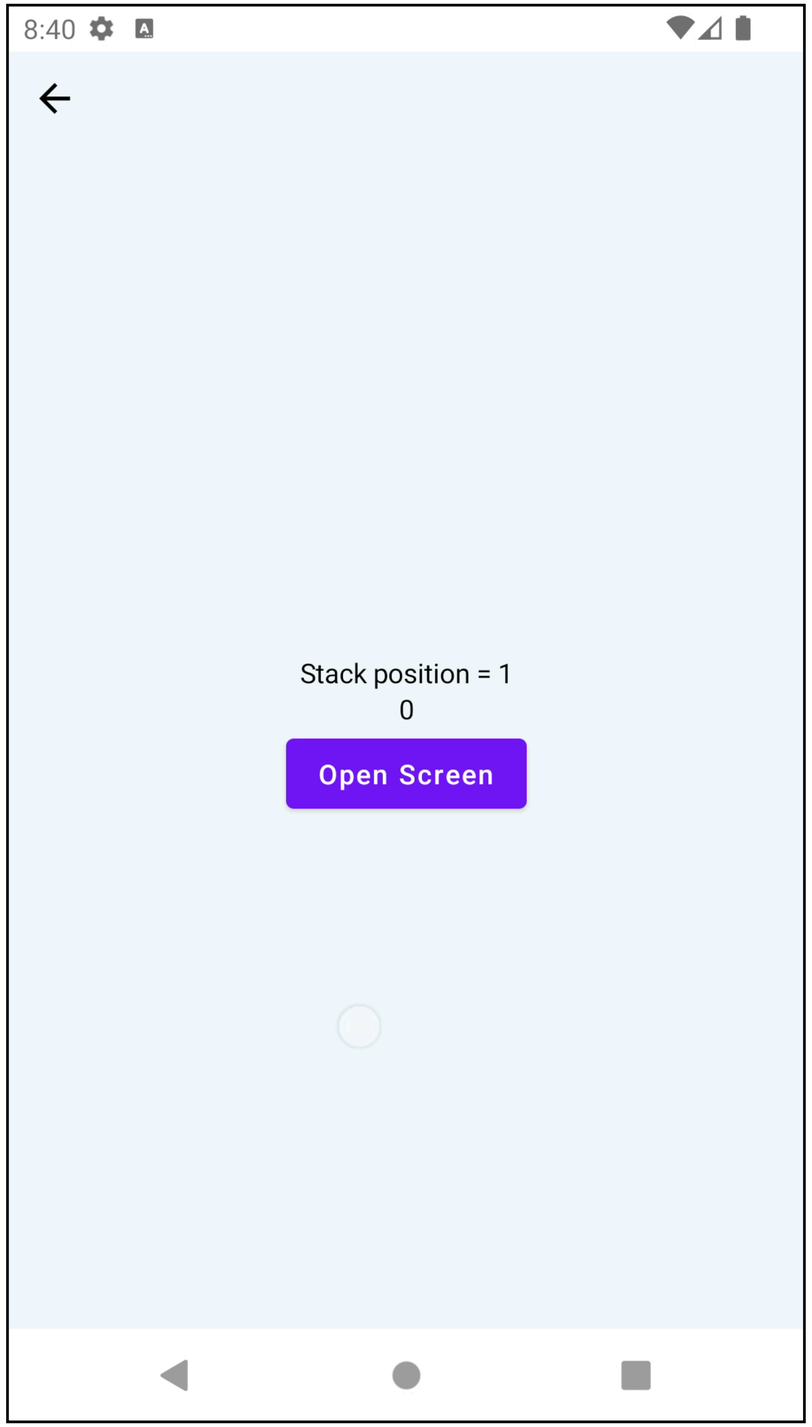


Все библиотеки под капотом используют SaveableStateHolder

А что если задать key?

```
@Parcelize
internal class FixedScreenState(
    val screenPos: Int,
    val key: String = UUID.randomUUID().toString(),
) : Parcelable


@Composable
internal fun FixedSelfMadeNavigation() {
    ...
    val saveableStateHolder = rememberSaveableStateHolder()
    saveableStateHolder.SaveableStateProvider(key = renderState.key) {
        ScreenWithButtons(
            …
        )
    }
}
```

```
@Parcelize
internal class FixedScreenState(
    val screenPos: Int,
    val key: String = UUID.randomUUID().toString(),
) : Parcelable


@Composable
internal fun FixedSelfMadeNavigation() {
    ...
    val saveableStateHolder = rememberSaveableStateHolder()
    saveableStateHolder.SaveableStateProvider(key = renderState.key) {
        ScreenWithButtons(
            …
        )
    }
}
```


8:40
Stack position = 1
0
Open Screen

Self-made: проблемы

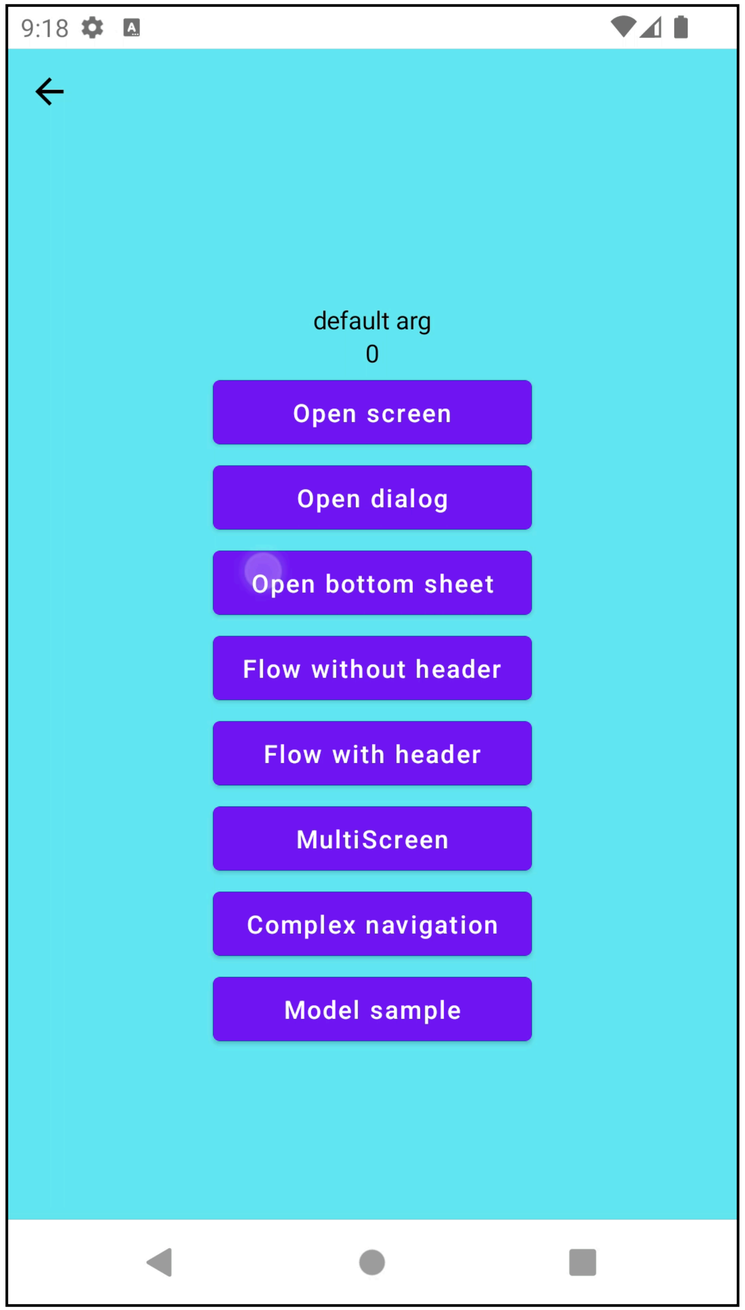
1. Ручное управление состоянием
2. Model для экрана с LifeCycle нужно писать самому
3. Нужно самостоятельно придумать дизайн
4. И это лишь верхушка айсберга

Может просто выберем библиотеку?

По каким критериям оценивать?

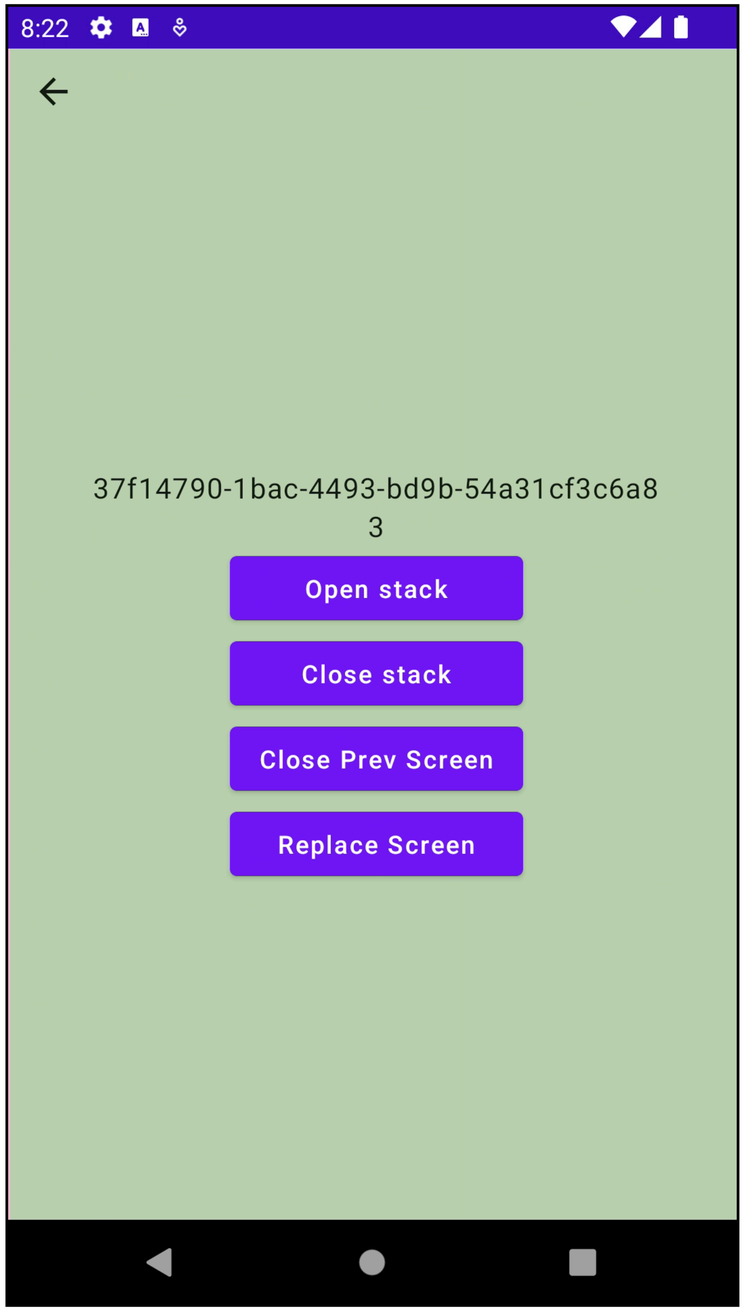
Что мы хотим от навигации

1. Поддержка базовых команд навигации
2. Поддержка манипуляций со стэком
3. Корректность
4. Наличие модели экрана
5. Возможность постепенной интеграции



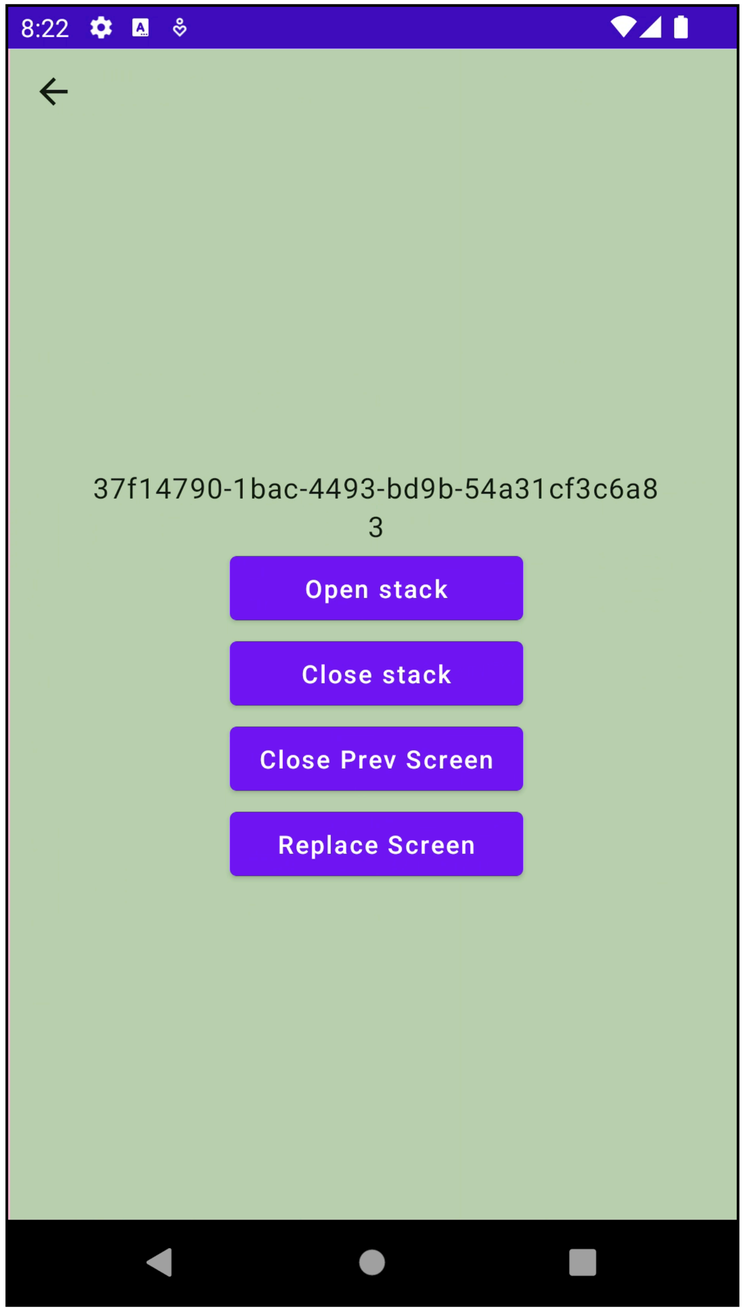
1. Функционал навигации

1. Открытие/Закрытие экрана
2. Вложенная навигация
3. Мультистек
4. Диалог
5. Bottom Sheet
6. Анимации
7. Модель экрана



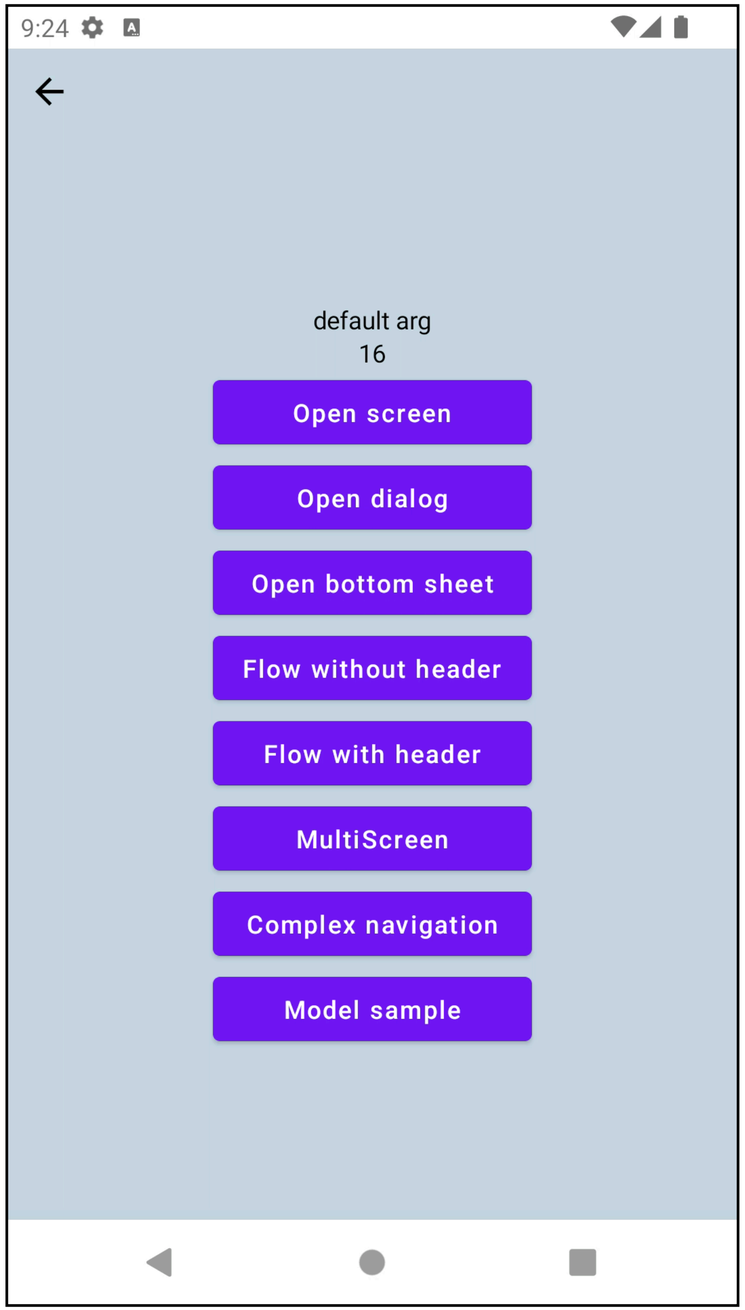
2. Навигация через состояние

1. Установка произвольного состояния
2. Получение состояния навигации



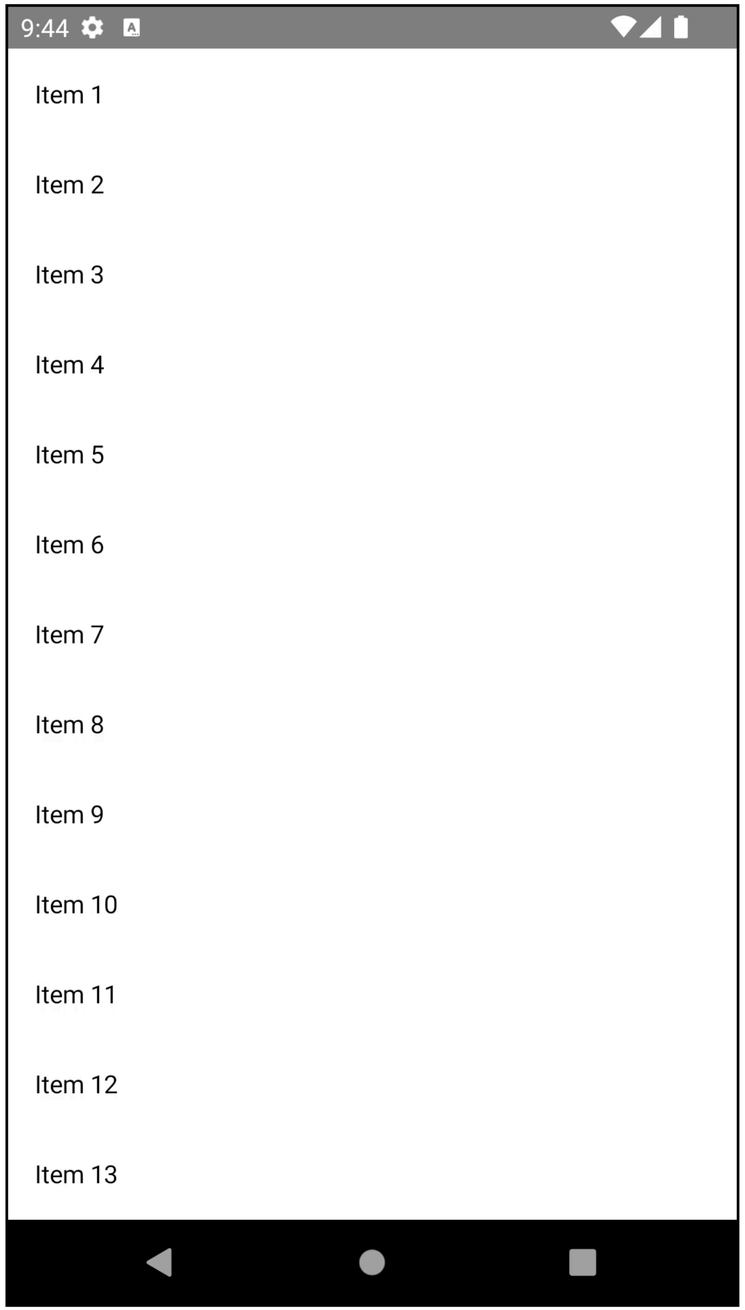
2*. Сложная навигация

1. Открытие стека экранов
2. Закрытие стека экранов
3. Закрытие отдельных экранов
4. Получение состояния навигации



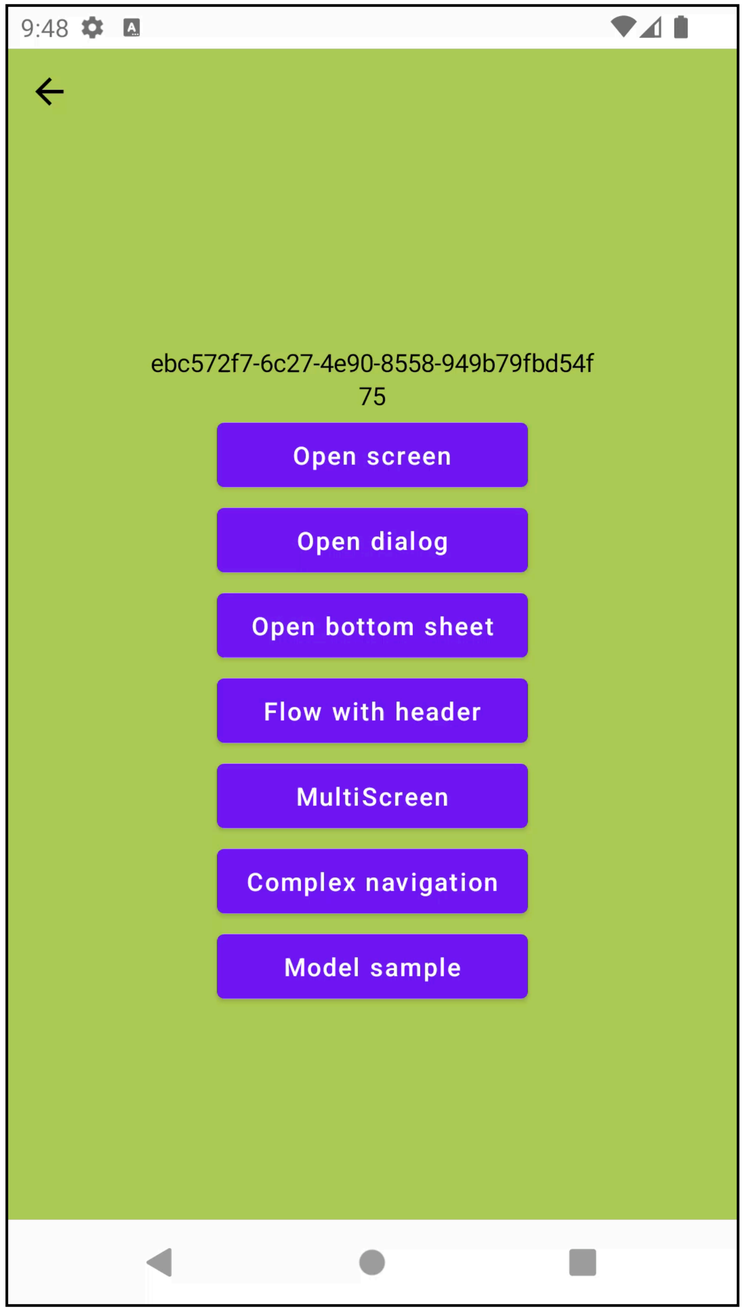
3. Корректность

1. Поворот экрана
2. Activity death
3. Process death



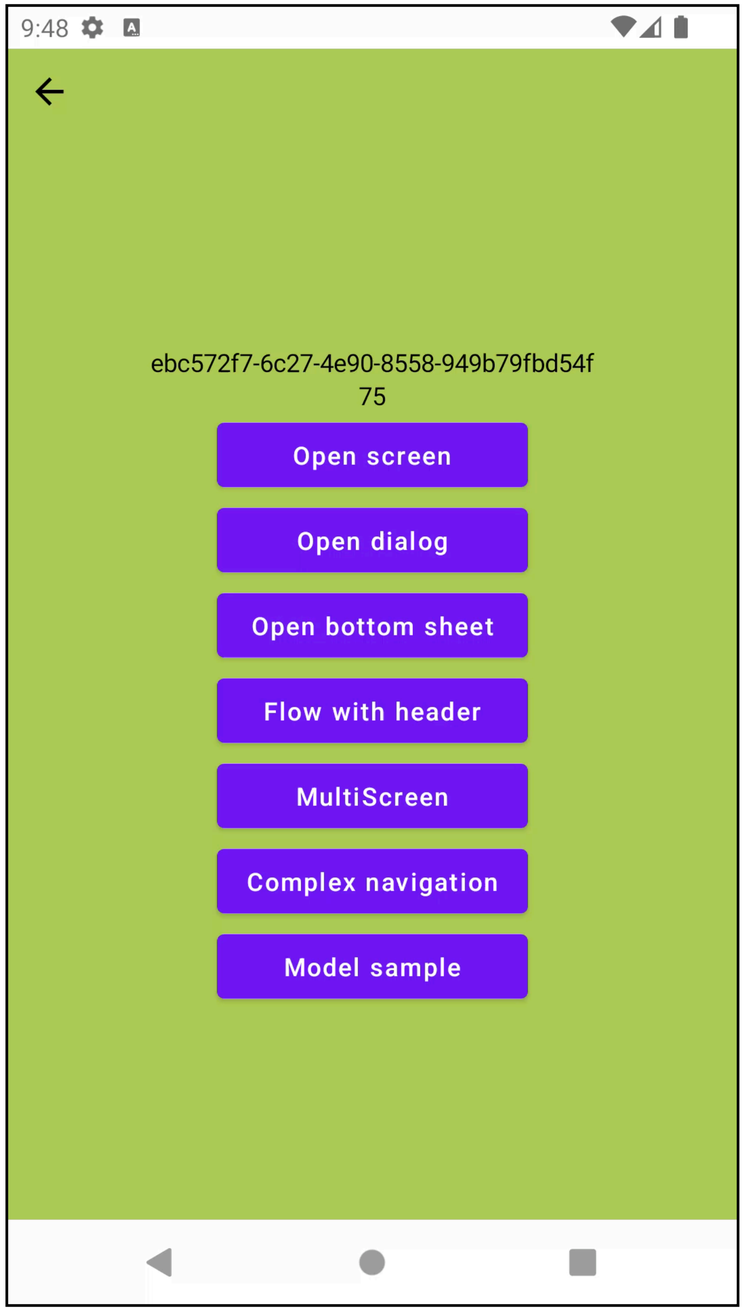
4. Модель экрана

1. Dispose модели экрана
2. LifeCycle для аналитики
3. Восстановление состояния после Activity/Process death



5. Декомпозиция навигации

1. Вложенная
2. В отдельном модуле
3. Внутри fragment



6. Интеграция с Fragment

1. Какие могут быть проблемы?
2. Как интегрировать?
3. Как открывать экраны на Fragment?

Что сравниваем

1. Google navigation
2. Appyx
3. Voyager
4. Modo
5. Odyssey

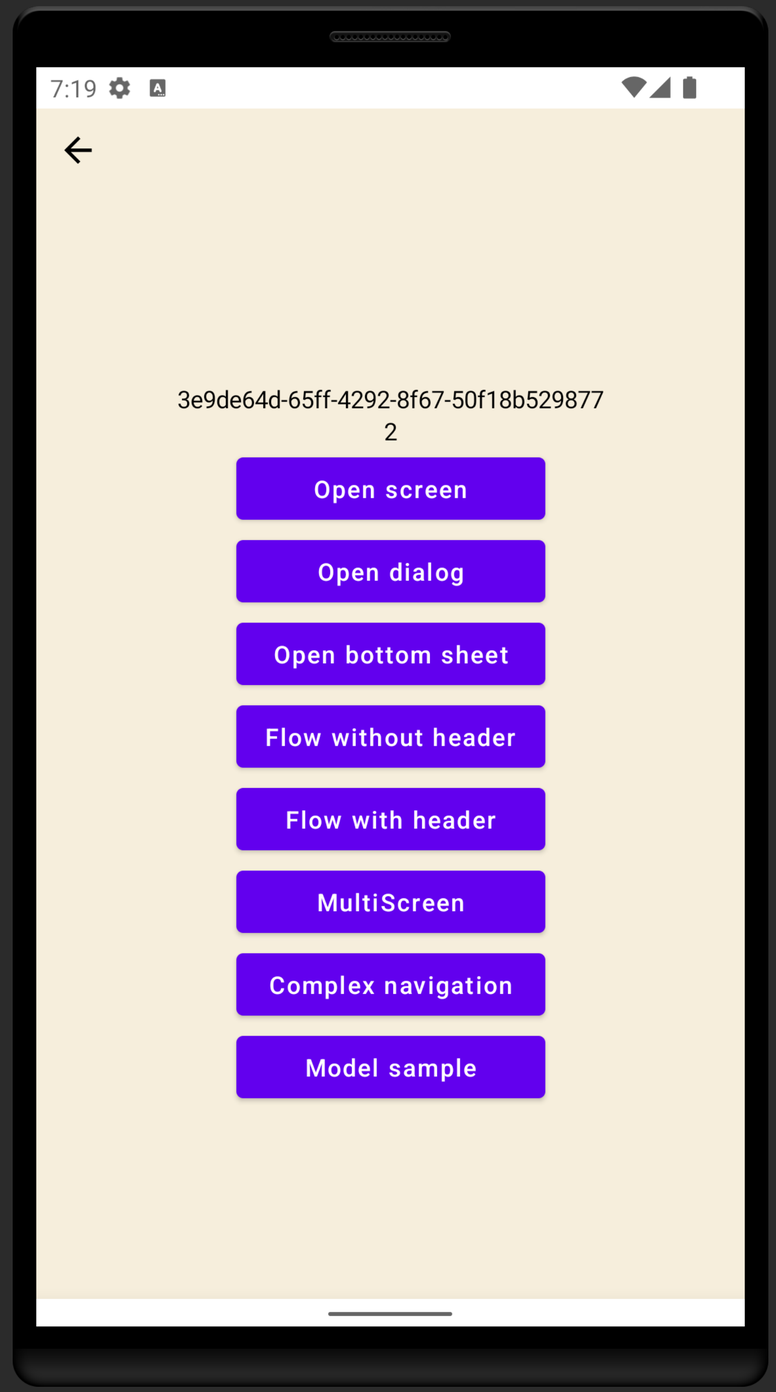

Готовим
Sample app

Sample App

https://github.com/hhru/jetpack-compose-navigation-research

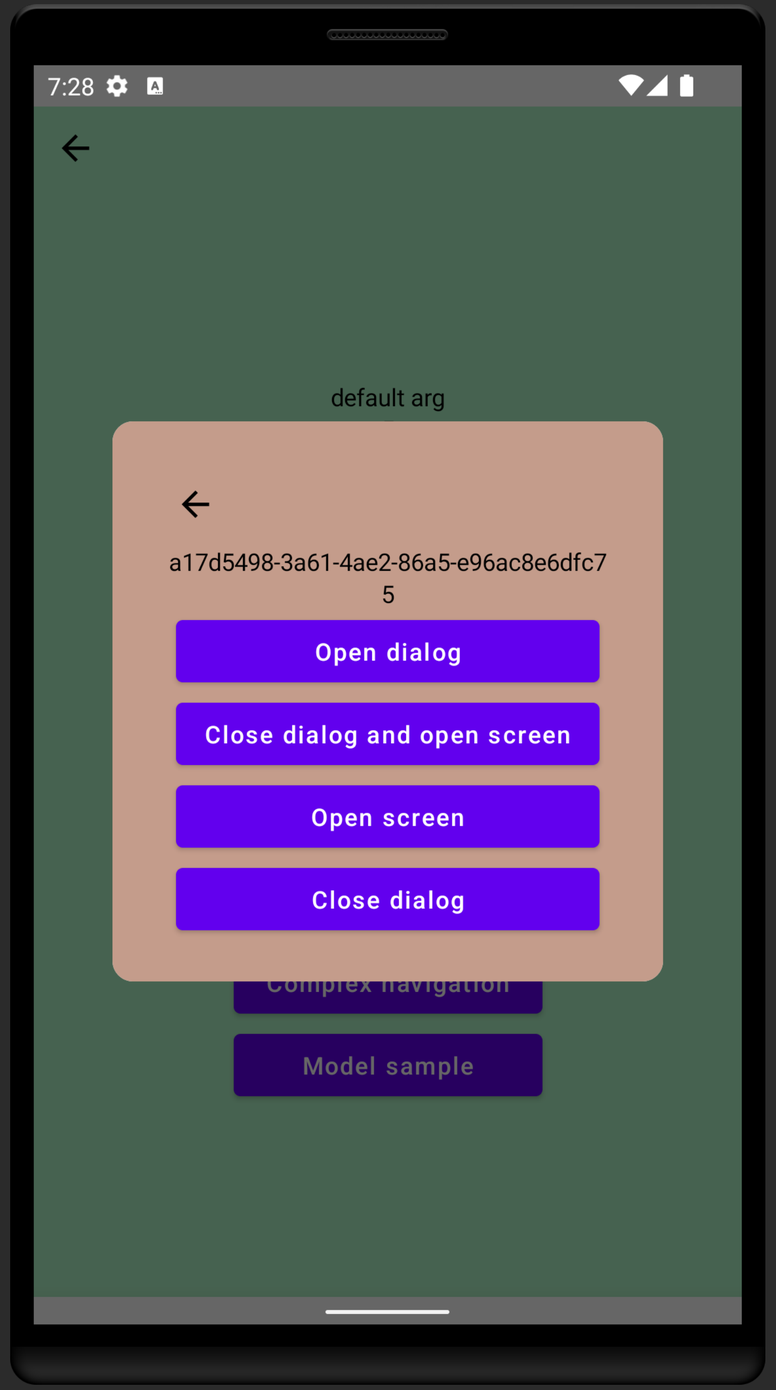
Экраны

```
@Composable
fun SampleScreenContent(
    openScreen: () -> Unit,
    openDialog: () -> Unit,
    openBottomSheet: () -> Unit,
    startFlowWithoutHeader: (() -> Unit)?,
    startFlowWithHeader: () -> Unit,
    openMultiscreen: () -> Unit,
    openScreenModel: () -> Unit,
    openComplexNavigation: () -> Unit,
    modifier: Modifier = Modifier,
    screenTitle: String? = null,
) {...}
```

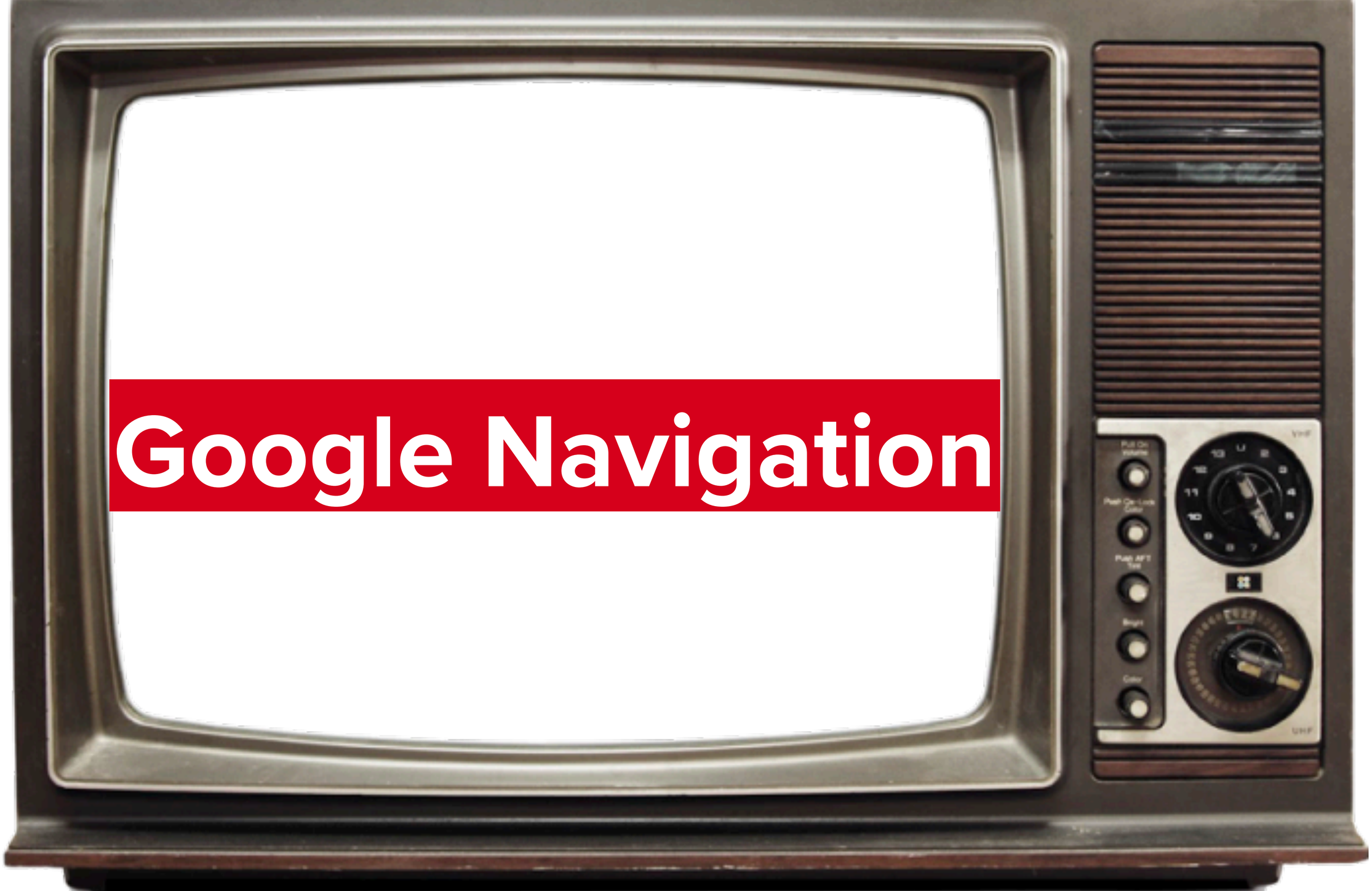


Экраны

```
@Composable
fun DialogContent(
    openDialog: () -> Unit,
    openScreen: (closeDialog: Boolean) -> Unit
    closeDialog: () -> Unit,
    modifier: Modifier = Modifier,
    screenTitle: String? = null,
) {...}
```




Обзор библиотек


Google Navigation

Google Navigation

https://developer.android.com/jetpack/compose/navigation

Google navigation component

1. Объявляем граф навигации в Compose
2. NavigationController - основное API
3. Аргументы и навигация через строки
4. Больно работать с аргументами
5. Поддержка анимаций, dialog, bottom sheet

Google navigation component

1. **Объявляем граф навигации в Compose**
2. NavigationController - основное API
3. Аргументы и навигация через строки
4. Больно работать с аргументами
5. Поддержка анимаций, dialog, bottom sheet

Google navigation component

1. Объявляем граф навигации в Compose
2. **NavigationController - основное API**
3. Аргументы и навигация через строки
4. Больно работать с аргументами
5. Поддержка анимаций, dialog, bottom sheet

Google navigation component

```
val navController = rememberNavController()
NavHost(
    navController = navController,
    startDestination = "screenNew?{title}"
) {
    composable(
        route = "screenNew?{title}",
        arguments = listOf(
            navArgument("title") {
                type = NavType.StringType
                defaultValue = "default arg"
            }
        )
    ) {
        SampleScreenContent(navController, it)
    }
    composable(...)
}
```

Google navigation component

```
val navController = rememberNavController()
NavHost(
    navController = navController,
    startDestination = "screenNew?{title}"
) {
    composable(
        route = "screenNew?{title}",
        arguments = listOf(
            navArgument("title") {
                type = NavType.StringType
                defaultValue = "default arg"
            }
        )
    ) {
        SampleScreenContent(navController, it)
    }
    composable(...)
```

Google navigation component

```
val navController = rememberNavController()
NavHost(
    navController = navController,
    startDestination = "screenNew?{title}"
) {
    composable(
        route = "screenNew?{title}",
        arguments = listOf(
            navArgument("title") {
                type = NavType.StringType
                defaultValue = "default arg"
            }
        )
    ) {
        SampleScreenContent(navController, it)
    }
    composable(...)
```

Google navigation component

```
val navController = rememberNavController()
NavHost(
    navController = navController,
    startDestination = "screenNew?{title}"
) {
    composable(
        route = "screenNew?{title}",
        arguments = listOf(
            navArgument("title") {
                type = NavType.StringType
                defaultValue = "default arg"
            }
        )
    ) {
        SampleScreenContent(navController, it)
    }
    composable(...)
```

Google navigation component

```
val navController = rememberNavController()
NavHost(
    navController = navController,
    startDestination = "screenNew?{title}"
) {
    composable(
        route = "screenNew?{title}",
        arguments = listOf(
            navArgument("title") {
                type = NavType.StringType
                defaultValue = "default arg"
            }
        )
    ) {
        SampleScreenContent(navController, it)
    }
    composable(...)
}
```

Google navigation component

```
val navController = rememberNavController()
NavHost(
    navController = navController,
    startDestination = "screenNew?{title}"
) {
    composable(
        route = "screenNew?{title}",
        arguments = listOf(
            navArgument("title") {
                type = NavType.StringType
                defaultValue = "default arg"
            }
        )
    ) {
        SampleScreenContent(navController, it)
    }
    composable(...)
}
```

```
composable(
    ...
) {
    SampleScreenContent(navController, it)
}
flowGraphWithoutHeader(navController)
flowGraphWithHeader(navController)
multiScreenGraph(modifier)
bottomSheet(route = Screens.bottomSheet) {
    SampleScreenContent(navController, it)
}
dialog(
    ...
) {
    DialogContent(
        ...
    )
}
```

Google navigation component

```
val navController = rememberNavController()
NavHost(
    navController = navController,
    startDestination = "screenNew?{title}"
) {
    composable(
        route = "screenNew?{title}",
        arguments = listOf(
            navArgument("title") {
                type = NavType.StringType
                defaultValue = "default arg"
            }
        )
    ) {
        SampleScreenContent(navController, it)
    }
    composable(...)
}
```

Google navigation component

1. Объявляем граф навигации в Compose
2. NavigationController - основное API
3. **Аргументы и навигация через строки**
4. Больно работать с аргументами
5. Поддержка анимаций, dialog, bottom sheet

Google navigation component

```
val navController = rememberNavController()
NavHost(
    navController = navController,
    startDestination = "screenNew?{title}"
) {
    composable(
        route = "screenNew?{title}",
        arguments = listOf(
            navArgument("title") {
                type = NavType.StringType
                defaultValue = "default arg"
            }
        )
    ) {
        SampleScreenContent(navController, it)
    }
    composable(...)
}
```

Google navigation component

```
val navController = rememberNavController()
NavHost(
    navController = navController,
    startDestination = "screenNew?{title}"
) {
    composable(
        route = "screenNew?{title}",
        arguments = listOf(
            navArgument("title") {
                type = NavType.StringType
                defaultValue = "default arg"
            }
        )
    ) {
        SampleScreenContent(navController, it)
    }
    composable(...)
```

Google navigation component

```
val navController = rememberNavController()
NavHost(
    navController = navController,
    startDestination = "screenNew?{title}"
) {
    composable(
        route = "screenNew?{title}",
        arguments = listOf(
            navArgument("title") {
                type = NavType.StringType
                defaultValue = "default arg"
            }
        )
    ) {
        SampleScreenContent(navController, it)
    }
    composable(...)
```

Google navigation component

```
val navController = rememberNavController()
NavHost(
    navController = navController,
    startDestination = "screenNew?{title}"
) {
    composable(
        route = "screenNew?{title}",
        arguments = listOf(
            navArgument("title") {
                type = NavType.StringType
                defaultValue = "default arg"
            }
        )
    ) {
        SampleScreenContent(navController, it)
    }
    composable(...)
```

Как сделать чуточку лучше

```
object Sample {

    const val titleArg = "title"
    const val baseRoute = "screen"
    const val route = "$baseRoute?{$titleArg}"
    fun destination(title: String?) = "$baseRoute?${title.orEmpty()}"
}

composable(
    route = Screens.Sample.route,
    arguments = listOf(
        navArgument(Screens.Sample.titleArg) {
            type = NavType.StringType
            defaultValue = "default arg"
        }
    )
) {
    SampleScreenContent(navController, it)
}
```

```
object Sample {

    const val titleArg = "title"
    const val baseRoute = "screen"
    const val route = "$baseRoute?{$titleArg}"
    fun destination(title: String?) = "$baseRoute?${title.orEmpty()}"
}

composable(
    route = Screens.Sample.route,
    arguments = listOf(
        navArgument(Screens.Sample.titleArg) {
            type = NavType.StringType
            defaultValue = "default arg"
        }
    )
) {
    SampleScreenContent(navController, it)
}
```

Google navigation component

1. Объявляем граф навигации в Compose
2. NavigationController - основное API
3. Аргументы и навигация через строки
4. **Больно работать с аргументами**
5. Поддержка анимаций, dialog, bottom sheet

```
SampleScreenContent(
    openScreen = { navController.openSample() },
    openBottomSheet = { navController.navigate(Screens.bottomSheet) },
    openDialog = {
        navController.navigate(Screens.Dialog.destination(randomString()))
    },
    startFlowWithHeader = { navController.navigate(Screens.flowWithHeader) },
    startFlowWithoutHeader = {
        navController.navigate(Screens.flowWithoutHeader)
    },
    openMultiscreen = {
        navController.navigate(Screens.multiScreen)
    },
    openScreenModel = {
        navController.navigate(Screens.Sample.destination(null))
    },
    openComplexNavigation = { },
    screenTitle = it.arguments?.getString(Screens.Sample.titleArg),
    modifier = Modifier.fillMaxSize()
)
```

```
SampleScreenContent(
    openScreen = { navController.navigate("screen?{${randomString()}}") },
    openBottomSheet = { navController.navigate(Screens.bottomSheet) },
    openDialog = {
        navController.navigate(Screens.Dialog.destination(randomString()))
    },
    startFlowWithHeader = { navController.navigate(Screens.flowWithHeader) },
    startFlowWithoutHeader = {
        navController.navigate(Screens.flowWithoutHeader)
    },
    openMultiscreen = {
        navController.navigate(Screens.multiScreen)
    },
    openScreenModel = {
        navController.navigate(Screens.Sample.destination(null))
    },
    openComplexNavigation = { },
    screenTitle = it.arguments?.getString(Screens.Sample.titleArg),
    modifier = Modifier.fillMaxSize()
)
```

```
SampleScreenContent(
    openScreen = { navController.openSample() },
    openBottomSheet = { navController.navigate(Screens.bottomSheet) },
    openDialog = {
        navController.navigate(Screens.Dialog.destination(randomString()))
    },
    startFlowWithHeader = { navController.navigate(Screens.flowWithHeader) },
    startFlowWithoutHeader = {
        navController.navigate(Screens.flowWithoutHeader)
    },
    openMultiscreen = {
        navController.navigate(Screens.multiScreen)
    },
    openScreenModel = {
        navController.navigate(Screens.Sample.destination(null))
    },
    openComplexNavigation = { },
    screenTitle = it.arguments?.getString(Screens.Sample.titleArg),
    modifier = Modifier.fillMaxSize()
)
```

Когда нет compile time проверок

Google navigation component

1. Объявляем граф навигации в Compose
2. NavigationController - основное API
3. Аргументы и навигация через строки
4. Больно работать с аргументами
5. **Поддержка анимаций, dialog, bottom sheet**

Animation & Bottom Sheet

```
ModalBottomSheetLayout(bottomSheetNavigator = bottomSheetNavigator) {
    AnimatedNavHost(
        navController = navController,
        startDestination = Screens.Sample.route,
        modifier = modifier,
        enterTransition = {
            slideIntoContainer(
                AnimatedContentScope.SlideDirection.Left,
                animationSpec = tween()
            )
        },
        exitTransition = {
            slideOutOfContainer(
                AnimatedContentScope.SlideDirection.Left,
                animationSpec = tween()
            )
        },
        ...
```

Animation & Bottom Sheet

```
ModalBottomSheetLayout(bottomSheetNavigator = bottomSheetNavigator) {
    AnimatedNavHost(
        navController = navController,
        startDestination = Screens.Sample.route,
        modifier = modifier,
        enterTransition = {
            slideIntoContainer(
                AnimatedContentScope.SlideDirection.Left,
                animationSpec = tween()
            )
        },
        exitTransition = {
            slideOutOfContainer(
                AnimatedContentScope.SlideDirection.Left,
                animationSpec = tween()
            )
        },
        ...
```

Animation & Bottom Sheet

```
ModalBottomSheetLayout(bottomSheetNavigator = bottomSheetNavigator) {
    AnimatedNavHost() {
        bottomSheet(route = Screens.bottomSheet) {
            SampleScreenContent(navController, it)
        }

        ...
```

Google navigation: плюсы

1. Ну это же Гугл!
2. Просто реализовать
3. Поддержка VM из коробки
4. Встроенная поддержка deeplink*
5. Можно передавать лямбды в навигацию
5. Dialog & BootomSheet из коробки

Google navigation: минусы

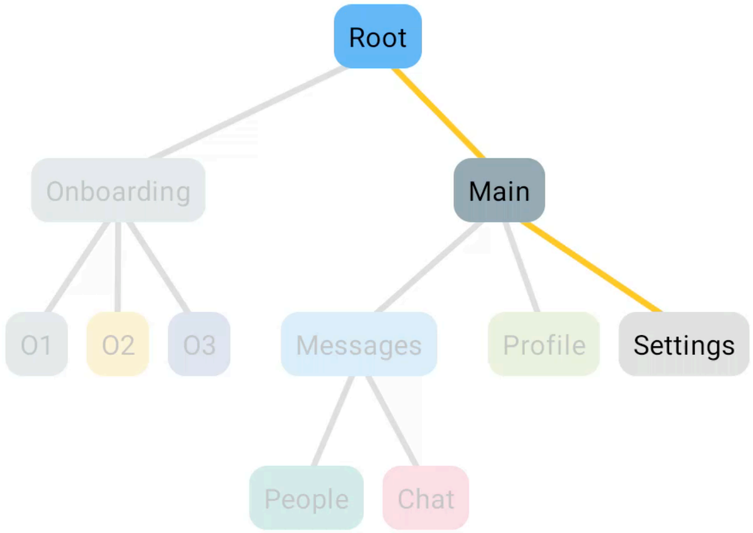
1. Route - это просто строка
2. Передача аргументов - боль
3. Навигация через Composable fun
4. Нет compile checks
5. Сложная навигация - боль


Appyx (bumble)

Appyx

https://bumble-tech.github.io/appyx/

Экран - вершина графа



Nodes

The app is organised into a tree of Nodes.

Nodes have @Composable UI, each have their own lifecycle on and off the screen, and can choose which of their children to delegate the control flow to.

Аррух

1. Экран - это вершина графа (Node & ParentNode)
2. Навигация через NavModel
3. NavigationResolver & NavTarget для навигации
4. Гибкие анимации

Аррух

1. **Экран - это вершина графа (Node & ParentNode)**
2. Навигация через NavModel
3. NavigationResolver & NavTarget для навигации
4. Гибкие анимации

Appyx: Activity

```
class DemoActivity : BaseActivity(R.layout.activity_demo_root),
IntegrationPointProvider {

    override lateinit var appyxIntegrationPoint: ActivityIntegrationPoint
        protected set

    fun createIntegrationPoint(savedInstanceState: Bundle?) =
        ActivityIntegrationPoint(
            activity = this,
            savedInstanceState = savedInstanceState
        )

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        appyxIntegrationPoint = createIntegrationPoint(savedInstanceState)
    }
    …
```

Appyx: Fragment

```
override fun onCreateView(...): View =
    ComposeView(requireContext()).apply {
        setContent {
            NodeHost(
                integrationPoint = ActivityIntegrationPoint
                    .getIntegrationPoint(requireContext())
            ) {
                StackNode(it)
            }
        }
    }
```

Appyx: Fragment

```
override fun onCreateView(...): View =
    ComposeView(requireContext()).apply {
        setContent {
            NodeHost(
                integrationPoint = ActivityIntegrationPoint
                    .getIntegrationPoint(requireContext())
            ) {
                StackNode(it)
            }
        }
    }
```

Appyx: Fragment

```
override fun onCreateView(...): View =
    ComposeView(requireContext()).apply {
        setContent {
            NodeHost(
                integrationPoint = ActivityIntegrationPoint
                    .getIntegrationPoint(requireContext())
            ) {
                StackNode(it)
            }
        }
    }
```

Appyx: StackNode

```
internal class StackNode(
    buildContext: BuildContext,
    initialNavTarget: NavTarget = NavTarget.SampleScreen(randomString()),
    private val backStack: BackStack<NavTarget> = BackStack(
        initialElement = initialNavTarget,
        savedStateMap = buildContext.savedStateMap,
    ),
) : ParentNode<NavTarget>(
    navModel = backStack,
    buildContext = buildContext
), Resolver<NavTarget> by NavigationResolver(
    backStack,
    { backStack.pop() }
) {

    @Composable
    override fun View(modifier: Modifier) {
        Children(
            navModel = backStack
```

```
internal class StackNode(
    buildContext: BuildContext,
    initialNavTarget: NavTarget = NavTarget.SampleScreen(randomString()),
    private val backStack: BackStack<NavTarget> = BackStack(
        initialElement = initialNavTarget,
        savedStateMap = buildContext.savedStateMap,
    ),
) : ParentNode<NavTarget>(
    navModel = backStack,
    buildContext = buildContext
), Resolver<NavTarget> by NavigationResolver(
    backStack,
    { backStack.pop() }
) {

    @Composable
    override fun View(modifier: Modifier) {
        Children(
            navModel = backStack
        )
    }
}
```

```
internal class StackNode(
    buildContext: BuildContext,
    initialNavTarget: NavTarget = NavTarget.SampleScreen(randomString()),
    private val backStack: BackStack<NavTarget> = BackStack(
        initialElement = initialNavTarget,
        savedStateMap = buildContext.savedStateMap,
    ),
) : ParentNode<NavTarget>(
    navModel = backStack,
    buildContext = buildContext
), Resolver<NavTarget> by NavigationResolver(
    backStack,
    { backStack.pop() }
) {

    @Composable
    override fun View(modifier: Modifier) {
        Children(
            navModel = backStack
        )
    }
}
```

```
internal class SampleScreen(
    buildContext: BuildContext,
    private val title: String,
    private val backStack: BackStack<NavTarget>,
) : Node(buildContext) {
    @Composable
    override fun View(modifier: Modifier) {
        val showNotSupported = LocalShowNotSupported.current
        SampleScreenContent(
            openScreen = {
                backStack.push(NavTarget.SampleScreen(randomString()))
            },
            ...
            screenTitle = title,
            modifier = Modifier.fillMaxSize()
        )
    }
}
```

```
internal class SampleScreen(
    buildContext: BuildContext,
    private val title: String,
    private val backStack: BackStack<NavTarget>,
) : Node(buildContext) {
    @Composable
    override fun View(modifier: Modifier) {
        val showNotSupported = LocalShowNotSupported.current
        SampleScreenContent(
            openScreen = {
                backStack.push(NavTarget.SampleScreen(randomString()))
            },
            ...
            screenTitle = title,
            modifier = Modifier.fillMaxSize()
        )
    }
}
```

```
internal class SampleScreen(
    buildContext: BuildContext,
    private val title: String,
    private val backStack: BackStack<NavTarget>,
) : Node(buildContext) {
    @Composable
    override fun View(modifier: Modifier) {
        val showNotSupported = LocalShowNotSupported.current
        SampleScreenContent(
            openScreen = {
                backStack.push(NavTarget.SampleScreen(randomString()))
            },
            ...
            screenTitle = title,
            modifier = Modifier.fillMaxSize()
        )
    }
}
```

```
internal class SampleScreen(
    buildContext: BuildContext,
    private val title: String,
    private val backStack: BackStack<NavTarget>,
) : Node(buildContext) {
    @Composable
    override fun View(modifier: Modifier) {
        val showNotSupported = LocalShowNotSupported.current
        SampleScreenContent(
            openScreen = {
                backStack.push(NavTarget.SampleScreen(randomString()))
            },
            ...
            screenTitle = title,
            modifier = Modifier.fillMaxSize()
        )
    }
}
```

Аррух

1. Экран - это вершина графа (Node & ParentNode)
2. **Навигация через NavModel**
3. NavigationResolver & NavTarget для навигации
4. Гибкие анимации

```
internal class StackNode(
    buildContext: BuildContext,
    initialNavTarget: NavTarget = NavTarget.SampleScreen(randomString()),
    private val backStack: BackStack<NavTarget> = BackStack(
        initialElement = initialNavTarget,
        savedStateMap = buildContext.savedStateMap,
    ),
) : ParentNode<NavTarget>(
    navModel = backStack,
    buildContext = buildContext
), Resolver<NavTarget> by NavigationResolver(
    backStack,
    { backStack.pop() }
) {

    @Composable
    override fun View(modifier: Modifier) {
        Children(
            navModel = backStack
        )
    }
}
```

```
internal class StackNode(
    buildContext: BuildContext,
    initialNavTarget: NavTarget = NavTarget.SampleScreen(randomString()),
    private val backStack: BackStack<NavTarget> = BackStack(
        initialElement = initialNavTarget,
        savedStateMap = buildContext.savedStateMap,
    ),
) : ParentNode<NavTarget>(
    navModel = backStack,
    buildContext = buildContext
), Resolver<NavTarget> by NavigationResolver(
    backStack,
    { backStack.pop() }
) {

    @Composable
    override fun View(modifier: Modifier) {
        Children(
            navModel = backStack
        )
    }
}
```

```
internal class SampleScreen(
    buildContext: BuildContext,
    private val title: String,
    private val backStack: BackStack<NavTarget>,
) : Node(buildContext) {
    @Composable
    override fun View(modifier: Modifier) {
        val showNotSupported = LocalShowNotSupported.current
        SampleScreenContent(
            openScreen = {
                backStack.push(NavTarget.SampleScreen(randomString()))
            },
            ...
            screenTitle = title,
            modifier = Modifier.fillMaxSize()
        )
    }
}
```

Аррух

1. Экран - это вершина графа (Node & ParentNode)
2. Навигация через NavModel
3. **NavigationResolver & NavTarget для навигации**
4. Гибкие анимации

```
internal class StackNode(
    buildContext: BuildContext,
    initialNavTarget: NavTarget = NavTarget.SampleScreen(randomString()),
    private val backStack: BackStack<NavTarget> = BackStack(
        initialElement = initialNavTarget,
        savedStateMap = buildContext.savedStateMap,
    ),
) : ParentNode<NavTarget>(
    navModel = backStack,
    buildContext = buildContext
), Resolver<NavTarget> by NavigationResolver(
    backStack,
    { backStack.pop() }
) {

    @Composable
    override fun View(modifier: Modifier) {
        Children(
            navModel = backStack
        )
    }
}
```

```
internal class SampleScreen(
    buildContext: BuildContext,
    private val title: String,
    private val backStack: BackStack<NavTarget>,
) : Node(buildContext) {
    @Composable
    override fun View(modifier: Modifier) {
        val showNotSupported = LocalShowNotSupported.current
        SampleScreenContent(
            openScreen = {
                backStack.push(NavTarget.SampleScreen(randomString()))
            },
            ...
            screenTitle = title,
            modifier = Modifier.fillMaxSize()
        )
    }
}
```

```
internal sealed interface NavTarget : Parcelable {

    @Parcelize
    data class SampleScreen(
        val tittle: String,
    ) : NavTarget

    @Parcelize
    data class ComplexNavigation(
        val title: String,
    ) : NavTarget

    @Parcelize
    data class Stack(
        val initialNavTarget: NavTarget = SampleScreen(randomString()),
    ) : NavTarget

    @Parcelize
    object MultiStack : NavTarget

    @Parcelize
    data class Flow(val firstScreenTitle: String) : NavTarget

}
```

```
internal class NavigationResolver(
    private val stack: BackStack<NavTarget>,
    private val onBack: () -> Unit,
) : Resolver<NavTarget> {

    override fun resolve(navTarget: NavTarget, buildContext: BuildContext): Node =
        when (navTarget) {
            is NavTarget.SampleScreen ->
                SampleScreen(buildContext, navTarget.tittle, stack)
            is NavTarget.ComplexNavigation ->
                ComplexNavigationScreen(buildContext, navTarget.title, stack)
            is NavTarget.Stack ->
                StackNode(buildContext, navTarget.initialNavTarget)
            is NavTarget.MultiStack ->
                MultiStackScreen(buildContext, 1)
            is NavTarget.Flow ->
                FlowNavigationScreen(buildContext, navTarget.firstScreenTitle, onB
        }

}
```

```
internal class NavigationResolver(
    private val stack: BackStack<NavTarget>,
    private val onBack: () -> Unit,
) : Resolver<NavTarget> {

override fun resolve(navTarget: NavTarget, buildContext: BuildContext): Node =
    when (navTarget) {
        is NavTarget.SampleScreen ->
            SampleScreen(buildContext, navTarget.tittle, stack)
        is NavTarget.ComplexNavigation ->
            ComplexNavigationScreen(buildContext, navTarget.title, stack)
        is NavTarget.Stack ->
            StackNode(buildContext, navTarget.initialNavTarget)
        is NavTarget.MultiStack ->
            MultiStackScreen(buildContext, 1)
        is NavTarget.Flow ->
            FlowNavigationScreen(buildContext, navTarget.firstScreenTitle, onBack)
    }

}
```

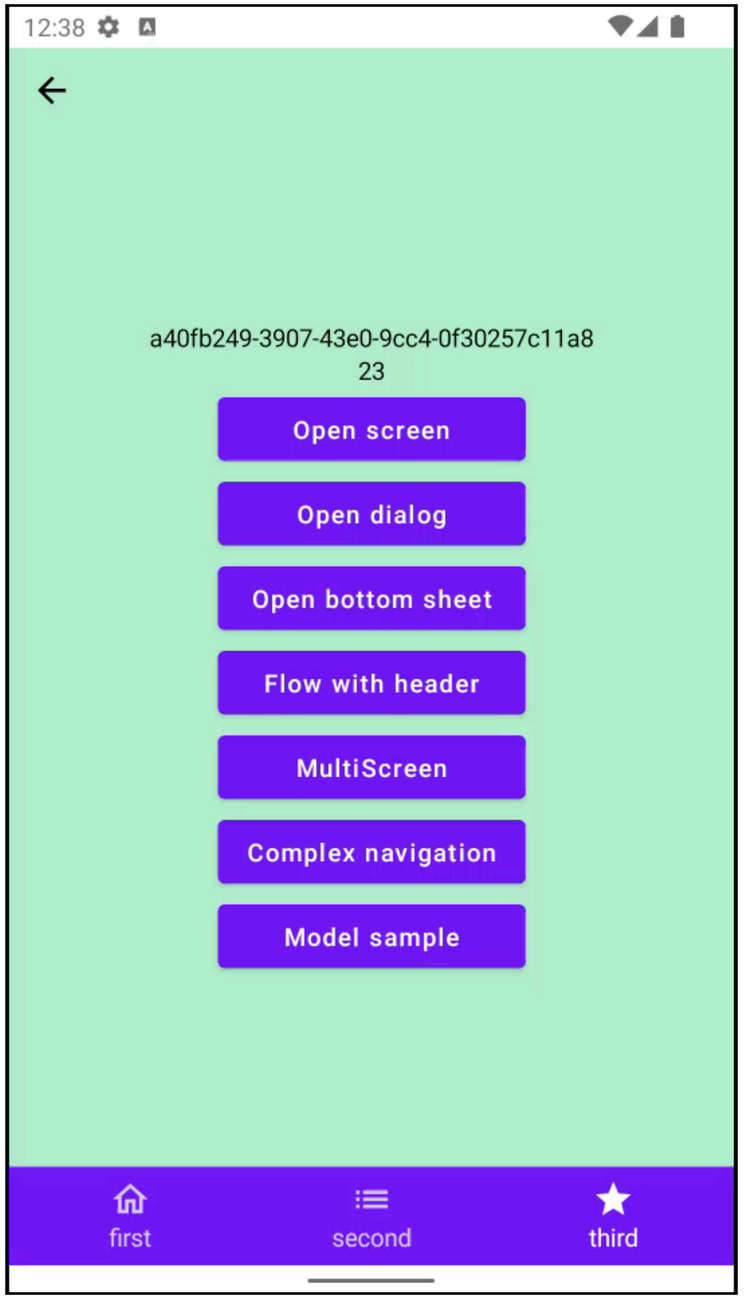
Аррух

1. Экран - это вершина графа (Node & ParentNode)
2. Навигация через NavModel
3. NavigationResolver & NavTarget для навигации
4. **Гибкие анимации**

```
Children(
    navModel = backStack,
    transitionHandler = rememberBackstackSlider(
        clipToBounds = true,
        transitionSpec = { spring(stiffness = Spring.StiffnessMediumLow) }
    ),
)
```

```
Children(
    navModel = backStack,
    transitionHandler = rememberBackstackSlider(
        clipToBounds = true,
        transitionSpec = { spring(stiffness = Spring.StiffnessMediumLow) }
    ),
)
```

Аррух: визуальные баги



Аррух: плюсы

1. Строим дерево навигации*
2. Аргументы - типизированы
3. Навигация может быть чем угодно
4. Анимации
5. Сами можем контролировать стейт*

Аррух: минусы

1. Нельзя произвольно изменить состояние*
2. Boilerplate для открытия экрана детей
3. Нет поддержки dialog & bottom sheet
4. Стейт экрана - Flow, нужно об этом помнить
5. Очень не хватает доки по интеграции в Fragment


Voyager

Voyager

https://voyager.adriel.cafe/

Voyager

1. Navigator - основа библиотеки
2. Экран - это объект
3. Навигируемся
4. Интеграция с популярными библиотеками
5. Bottom sheet

Voyager

1. **Navigator - основа библиотеки**
2. Экран - это объект
3. Навигируемся
4. Интеграция с популярными библиотеками
5. Bottom sheet

Voyager: Fragment

```
override fun onCreateView(...): View =
    ComposeView(requireContext()).apply {
        setContent {
            HHDynamicTheme {
                Navigator(screen = SampleScreen(randomString())) {
                    SlideTransition(it)
                }
            }
        }
    }
```

Voyager: Fragment

```
override fun onCreateView(...): View =
    ComposeView(requireContext()).apply {
        setContent {
            HHDynamicTheme {
                Navigator(screen = SampleScreen(randomString())) {
                    SlideTransition(it)
                }
            }
        }
    }
```

Voyager: Fragment

```
override fun onCreateView(...): View =
    ComposeView(requireContext()).apply {
        setContent {
            HHDynamicTheme {
                Navigator(screen = SampleScreen(randomString())) {
                    SlideTransition(it)
                }
            }
        }
    }
```

Voyager

1. Navigator - основа библиотеки
2. **Экран - это объект**
3. Навигируемся
4. Интеграция с популярными библиотеками
5. Bottom sheet

Voyager

```
internal class SampleScreen(
    private val title: String,
) : AndroidScreen() {

    @Composable
    override fun Content() {
        ...
    }

}
```

Voyager

```
internal class SampleScreen(
    private val title: String,
) : AndroidScreen() {

    @Composable
    override fun Content() {
        ...
    }

}
```

Voyager

```
internal class SampleScreen(
    private val title: String,
) : AndroidScreen() {

    @Composable
    override fun Content() {
        ...
    }

}
```

Voyager

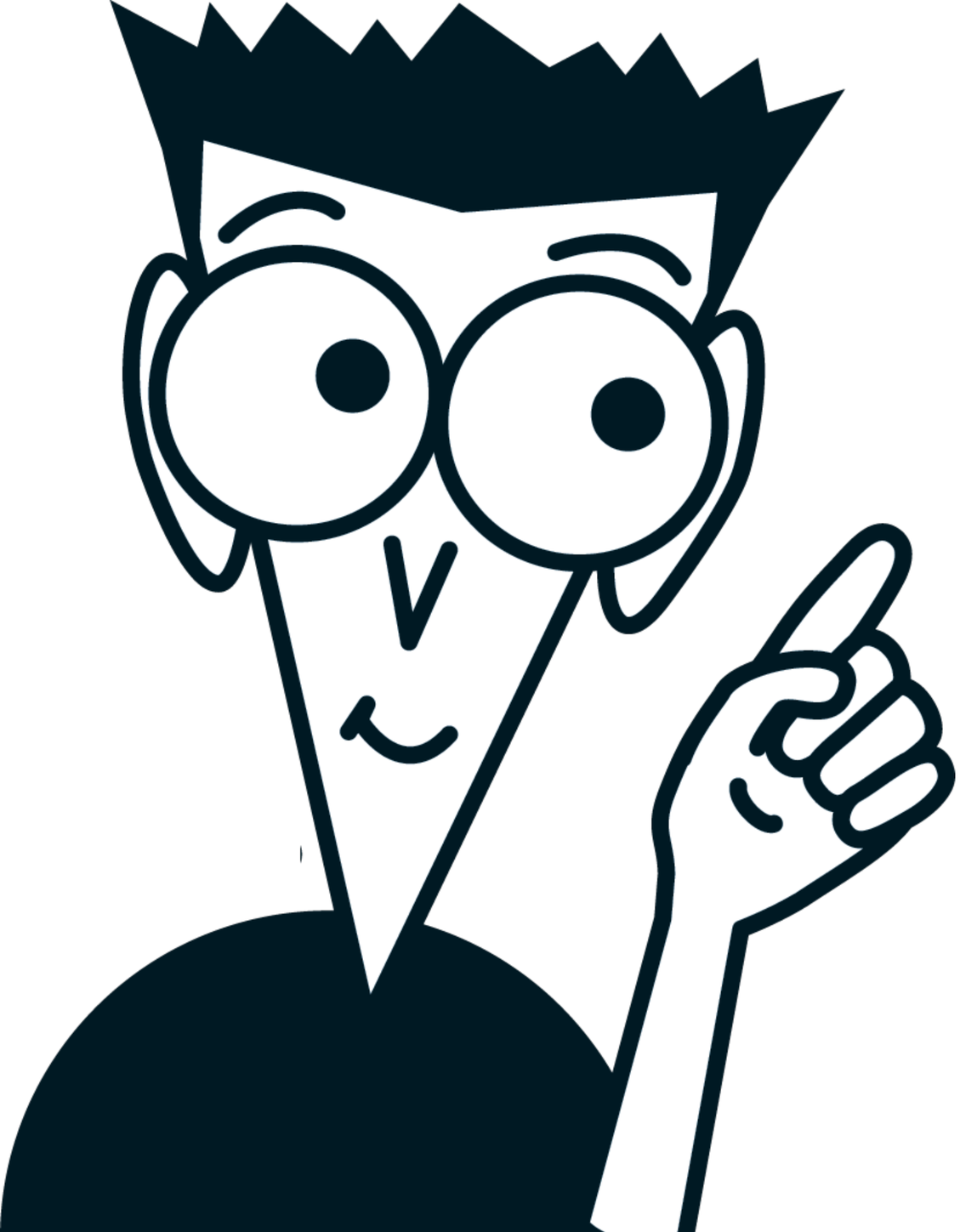
```
@Composable
override fun Content() {
    val navigator = LocalNavigator.currentOrThrow
    val bottomSheetNavigator = LocalBottomSheetNavigator.current
    SampleScreenContent(
        openScreen = { navigator.push(SampleScreen(title = randomString())) },
        openDialog = { /* нет решения },
        openBottomSheet = {
            bottomSheetNavigator.show(SampleScreen(randomString()))
        },
        // этот просто копия кейса ниже, поэтому не будем его рисовать
        startFlowWithoutHeader = null,
        startFlowWithHeader = {
            navigator.push(FlowNavigationScreen(randomString()))
        },
        openMultiscreen = { navigator.push(MultiStackScreen(0)) },
        openScreenModel = { },
```

Voyager

1. Navigator - основа библиотеки
2. Экран - это объект
3. **Навигируемся**
4. Интеграция с популярными библиотеками
5. Bottom sheet

```
@Composable
override fun Content() {
    val navigator = LocalNavigator.currentOrThrow
    val bottomSheetNavigator = LocalBottomSheetNavigator.current
    SampleScreenContent(
        openScreen = { navigator.push(SampleScreen(title = randomString())) },
        openDialog = { /* нет решения },
        openBottomSheet = {
            bottomSheetNavigator.show(SampleScreen(randomString()))
        },
        // этот просто копия кейса ниже, поэтому не будем его рисовать
        startFlowWithoutHeader = null,
        startFlowWithHeader = {
            navigator.push(FlowNavigationScreen(randomString()))
        },
        openMultiscreen = { navigator.push(MultiStackScreen(0)) },
        openScreenModel = { },
        openComplexNavigation = { navigator.push(ComplexNavigationScreen()) },
        screenTitle = title,
        modifier = Modifier.fillMaxSize()
    )
}
```

```
@Composable
override fun Content() {
    val navigator = LocalNavigator.currentOrThrow
    val bottomSheetNavigator = LocalBottomSheetNavigator.current
    SampleScreenContent(
        openScreen = { navigator.push(SampleScreen(title = randomString())) },
        openDialog = { /* нет решения },
        openBottomSheet = {
            bottomSheetNavigator.show(SampleScreen(randomString()))
        },
        // этот просто копия кейса ниже, поэтому не будем его рисовать
        startFlowWithoutHeader = null,
        startFlowWithHeader = {
            navigator.push(FlowNavigationScreen(randomString()))
        },
        openMultiscreen = { navigator.push(MultiStackScreen(0)) },
        openScreenModel = { },
        openComplexNavigation = { navigator.push(ComplexNavigationScreen()) },
        screenTitle = title,
        modifier = Modifier.fillMaxSize()
    )
}
```



Просто создаём экран

Хочу что-нибудь сложное

```
internal class MultiStackScreen(
    private val firstSelectedTab: Int,
) : AndroidScreen() {

    private val tabs = TABS.map {
        StackTab()
    }

    @Composable
    override fun Content() {
        TabNavigator(tabs[firstSelectedTab]) { tabNavigator ->
            val index = remember { derivedStateOf { tabs.indexOf(tabNavigator
            MultiStackScreenContent(
                selectedPos = index.value,
                onTabClick = {
                    tabNavigator.current = tabs[it]
                }
            ) {
                Box(Modifier.padding(it)) {
                    CurrentTab()
                }
            }
```

```
internal class MultiStackScreen(
    private val firstSelectedTab: Int,
) : AndroidScreen() {

    private val tabs = TABS.map {
        StackTab()
    }

    @Composable
    override fun Content() {
        TabNavigator(tabs[firstSelectedTab]) { tabNavigator ->
            val index = remember { derivedStateOf { tabs.indexOf(tabNavigator
            MultiStackScreenContent(
                selectedPos = index.value,
                onTabClick = {
                    tabNavigator.current = tabs[it]
                }
            ) {
                Box(Modifier.padding(it)) {
                    CurrentTab()
                }
            }
```

```
internal class MultiStackScreen(
    private val firstSelectedTab: Int,
) : AndroidScreen() {

    private val tabs = TABS.map {
        StackTab()
    }

    @Composable
    override fun Content() {
        TabNavigator(tabs[firstSelectedTab]) { tabNavigator ->
            val index = remember { derivedStateOf { tabs.indexOf(tabNavigator
            MultiStackScreenContent(
                selectedPos = index.value,
                onTabClick = {
                    tabNavigator.current = tabs[it]
                }
            ) {
                Box(Modifier.padding(it)) {
                    CurrentTab()
                }
            }
```

```
internal class MultiStackScreen(
    private val firstSelectedTab: Int,
) : AndroidScreen() {

    private val tabs = TABS.map {
        StackTab()
    }

    @Composable
    override fun Content() {
        TabNavigator(tabs[firstSelectedTab]) { tabNavigator ->
            val index = remember { derivedStateOf { tabs.indexOf(tabNavigator
            MultiStackScreenContent(
                selectedPos = index.value,
                onTabClick = {
                    tabNavigator.current = tabs[it]
                }
            ) {
                Box(Modifier.padding(it)) {
                    CurrentTab()
                }
            }
```

```
internal class MultiStackScreen(
    private val firstSelectedTab: Int,
) : AndroidScreen() {

    private val tabs = TABS.map {
        StackTab()
    }

    @Composable
    override fun Content() {
        TabNavigator(tabs[firstSelectedTab]) { tabNavigator ->
val index = remember { derivedStateOf { tabs.indexOf(tabNavigator.current) } }
            MultiStackScreenContent(
                selectedPos = index.value,
                onTabClick = {
                    tabNavigator.current = tabs[it]
                }
            ) {
                Box(Modifier.padding(it)) {
                    CurrentTab()
                }
            }
```

```
internal class MultiStackScreen(
    private val firstSelectedTab: Int,
) : AndroidScreen() {

    private val tabs = TABS.map {
        StackTab()
    }

    @Composable
    override fun Content() {
        TabNavigator(tabs[firstSelectedTab]) { tabNavigator ->
            val index = remember { derivedStateOf { tabs.indexOf(tabNavigator
            MultiStackScreenContent(
                selectedPos = index.value,
                onTabClick = {
                    tabNavigator.current = tabs[it]
                }
            ) {
                Box(Modifier.padding(it)) {
                    CurrentTab()
                }
            }
```

```
internal class MultiStackScreen(
    private val firstSelectedTab: Int,
) : AndroidScreen() {

    private val tabs = TABS.map {
        StackTab()
    }

    @Composable
    override fun Content() {
        TabNavigator(tabs[firstSelectedTab]) { tabNavigator ->
            val index = remember { derivedStateOf { tabs.indexOf(tabNavigator
            MultiStackScreenContent(
                selectedPos = index.value,
                onTabClick = {
                    tabNavigator.current = tabs[it]
                }
            ) {
                Box(Modifier.padding(it)) {
                    CurrentTab()
                }
            }
```

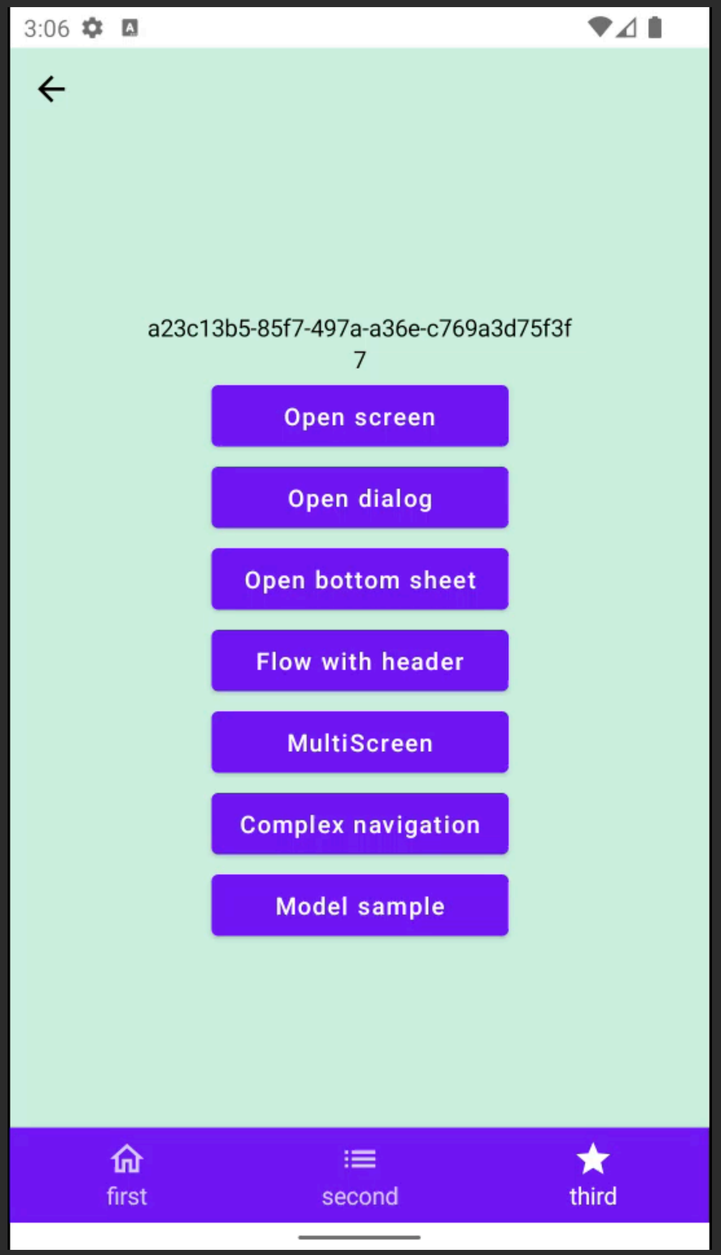
```
internal class StackTab : Tab {

    override val key: ScreenKey = uniqueScreenKey

    override val options: TabOptions
        @Composable
        get() {
            // просто заглушка, т.к. по факту мы это не используем
            return remember {
                TabOptions(
                    index = 0u,
                    title = "",
                    icon = null
                )
            }
        }

    @Composable
    override fun Content() {
        Navigator(screen = SampleScreen(randomString()))
    }
```

```
internal class StackTab : Tab {

    override val key: ScreenKey = uniqueScreenKey

    override val options: TabOptions
        @Composable
        get() {
            // просто заглушка, т.к. по факту мы это не используем
            return remember {
                TabOptions(
                    index = 0u,
                    title = "",
                    icon = null
                )
            }
        }

    @Composable
    override fun Content() {
        Navigator(screen = SampleScreen(randomString()))
    }
```

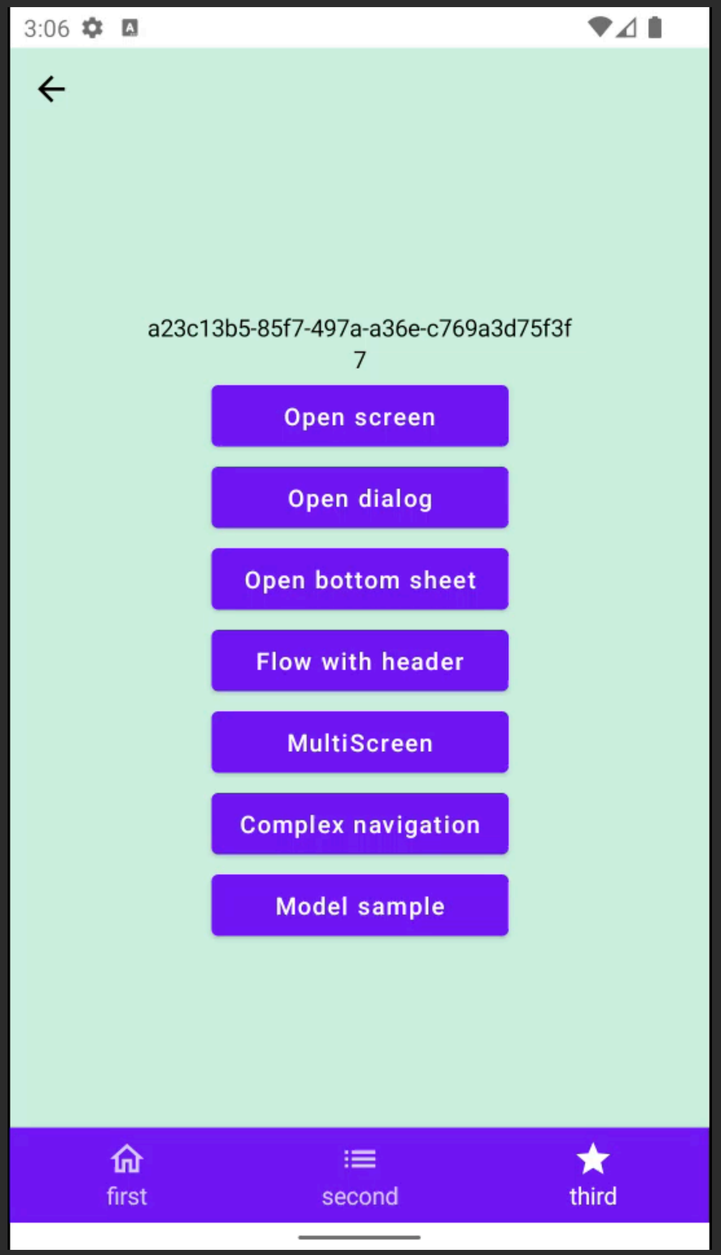
```
internal class StackTab : Tab {

    override val key: ScreenKey = uniqueScreenKey

    override val options: TabOptions
        @Composable
        get() {
            // просто заглушка, т.к. по факту мы это не используем
            return remember {
                TabOptions(
                    index = 0u,
                    title = "",
                    icon = null
                )
            }
        }

    @Composable
    override fun Content() {
        Navigator(screen = SampleScreen(randomString()))
    }
```

```
internal class StackTab : Tab {

    override val key: ScreenKey = uniqueScreenKey

    override val options: TabOptions
        @Composable
        get() {
            // просто заглушка, т.к. по факту мы это не используем
            return remember {
                TabOptions(
                    index = 0u,
                    title = "",
                    icon = null
                )
            }
        }

    @Composable
    override fun Content() {
        Navigator(screen = SampleScreen(randomString()))
    }
```

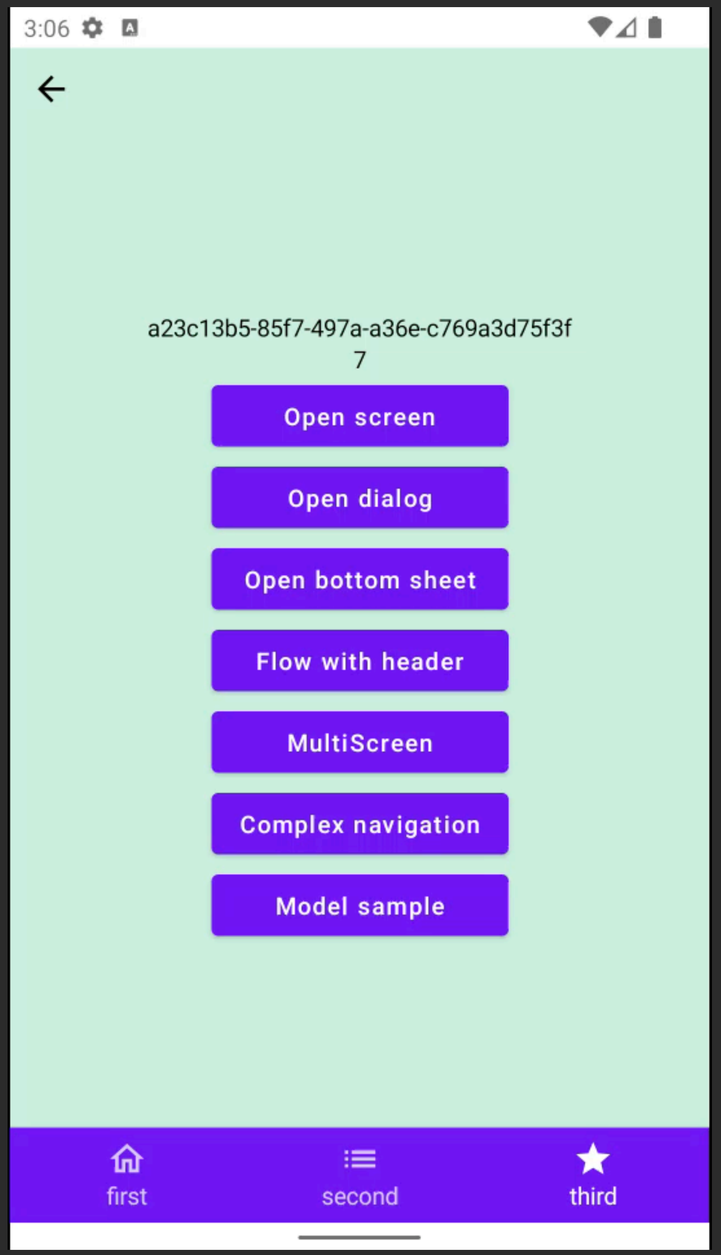
А что там с Корректностью?




```
public abstract class AndroidScreen :
    Screen, ScreenLifecycleProvider {

    override val key: ScreenKey = uniqueScreenKey

}
```




```
public abstract class AndroidScreen :
    Screen, ScreenLifecycleProvider {

    override val key: ScreenKey = uniqueScreenKey

}

public val Screen.uniqueScreenKey: ScreenKey
    get() = "Screen#${nextScreenKey.getAndIncrement()}"
```




```
public abstract class AndroidScreen :
    Screen, ScreenLifecycleProvider {

    override val key: ScreenKey = uniqueScreenKey

}

private val nextScreenKey = AtomicInt32(0)

public val Screen.uniqueScreenKey: ScreenKey
    get() = "Screen#${nextScreenKey.getAndIncrement()}"
```

Voyager: плюсы

1. Простота открытия экранов
2. Аргументы - типизированы
3. Анимации
4. Поддержка множества способов создать model экрана, в том числе VM, Koin и т.п.
5. Связка с LifeCycle

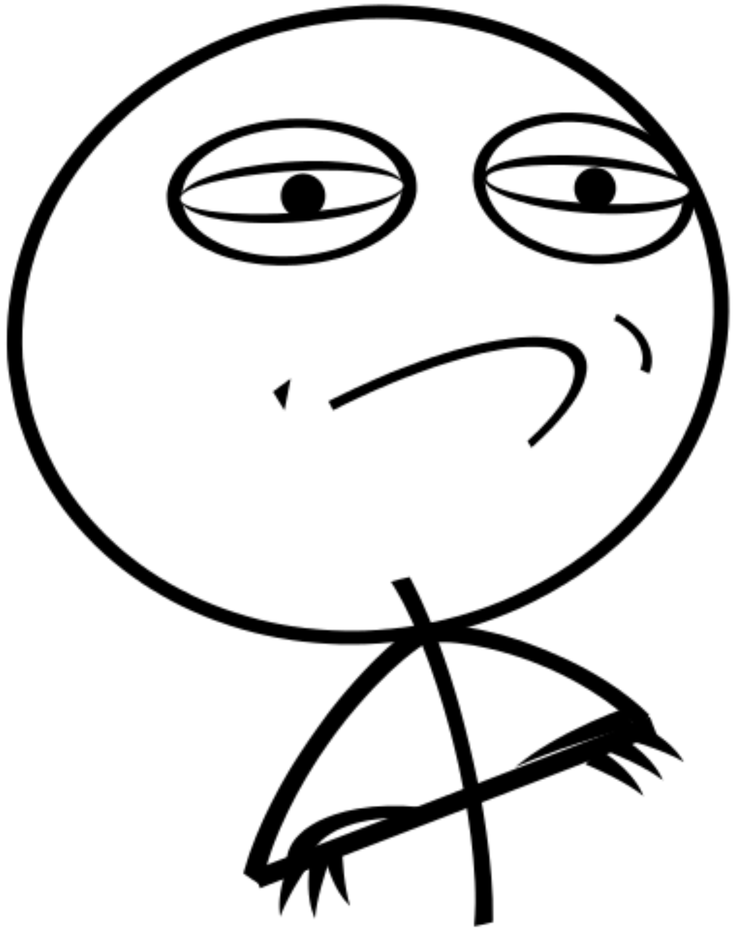
Voyager: минусы

1. Навигация внутри Composable fun
2. Баги, баги, баги
3. Нет поддержки dialog*
4. Почти не кастомизировать


Modo

Modo

https://github.com/terrakok/Modo



Навигация = Состоянию

Modo

01

State

02

Reducer + Commands

03

Screen/Container

Modo

1. Fragment
2. ContainerScreen/Screen
3. Навигируемся
4. NavModel/ContainerScreen
5. Поддержка анимаций, dialog

Modo: Fragment

```
private var rootScreen: ModoStackScreen? = null

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    rootScreen = Modo.init(savedInstanceState, rootScreen) {
        ModoStackScreen()
    }
}

override fun onCreateView(...): View =
    ComposeView(requireContext()).apply {
        setContent {
            rootScreen?.Content()
        }
    }
```

Modo: Fragment

```
private var rootScreen: ModoStackScreen? = null

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    rootScreen = Modo.init(savedInstanceState, rootScreen) {
        ModoStackScreen()
    }
}

override fun onCreateView(...): View =
    ComposeView(requireContext()).apply {
        setContent {
            rootScreen?.Content()
        }
    }
```

Modo: Fragment

```
private var rootScreen: ModoStackScreen? = null

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    rootScreen = Modo.init(savedInstanceState, rootScreen) {
        ModoStackScreen()
    }
}

override fun onCreateView(...): View =
    ComposeView(requireContext()).apply {
        setContent {
            rootScreen?.Content()
        }
    }
```

Modo: Fragment

```
private var rootScreen: ModoStackScreen? = null

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    rootScreen = Modo.init(savedInstanceState, rootScreen) {
        ModoStackScreen()
    }
}

override fun onCreateView(...): View =
    ComposeView(requireContext()).apply {
        setContent {
            rootScreen?.Content()
        }
    }
```

Modo: Fragment

```
private var rootScreen: ModoStackScreen? = null

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    rootScreen = Modo.init(savedInstanceState, rootScreen) {
        ModoStackScreen()
    }
}

override fun onCreateView(...): View =
    ComposeView(requireContext()).apply {
        setContent {
            rootScreen?.Content()
        }
    }
```

Modo: Fragment

```
private var rootScreen: ModoStackScreen? = null

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    rootScreen = Modo.init(savedInstanceState, rootScreen) {
        ModoStackScreen()
    }
}

override fun onCreateView(...): View =
    ComposeView(requireContext()).apply {
        setContent {
            rootScreen?.Content()
        }
    }
```

Modo: Fragment

```
private var rootScreen: ModoStackScreen? = null

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    rootScreen = Modo.init(savedInstanceState, rootScreen) {
        ModoStackScreen()
    }
}

...

override fun onSaveInstanceState(outState: Bundle) {
    super.onSaveInstanceState(outState)
    Modo.save(outState, rootScreen)
}
```

Modo

1. Fragment
2. **ContainerScreen/Screen**
3. Навигируемся
4. NavModel/ContainerScreen
5. Поддержка анимаций, dialog

Modo: StackScreen

```
@Parcelize
internal class ModoStackScreen(
    private val navModel: StackNavModel =
        StackNavModel(SampleScreen(randomString())),
) : StackScreen(navModel) {

    @Composable
    override fun Content() {
        TopScreenContent { SlideTransition() }
    }
}
```

Modo: StackScreen

```
@Parcelize
internal class ModoStackScreen(
    private val navModel: StackNavModel =
        StackNavModel(SampleScreen(randomString())),
) : StackScreen(navModel) {

    @Composable
    override fun Content() {
        TopScreenContent { SlideTransition() }
    }
}
```

Modo: StackScreen

```
@Parcelize
internal class ModoStackScreen(
    private val navModel: StackNavModel =
        StackNavModel(SampleScreen(randomString())),
) : StackScreen(navModel) {

    @Composable
    override fun Content() {
        TopScreenContent { SlideTransition() }
    }
}
```

Modo: StackScreen

```
@Parcelize
internal class ModoStackScreen(
    private val navModel: StackNavModel =
        StackNavModel(SampleScreen(randomString())),
) : StackScreen(navModel) {

    @Composable
    override fun Content() {
        TopScreenContent { SlideTransition() }
    }
}
```

Modo: StackScreen

```
@Parcelize
internal class ModoStackScreen(
    private val navModel: StackNavModel =
        StackNavModel(SampleScreen(randomString())),
) : StackScreen(navModel) {

    @Composable
    override fun Content() {
        TopScreenContent { SlideTransition() }
    }
}
```

Modo: StackScreen

```
@Parcelize
internal class ModoStackScreen(
    private val navModel: StackNavModel =
        StackNavModel(SampleScreen(randomString())),
) : StackScreen(navModel) {

    @Composable
    override fun Content() {
        TopScreenContent { SlideTransition() }
    }
}
```

Modo: Screen

```
@Parcelize
internal class SampleScreen(
    private val title: String,
    override val screenKey: ScreenKey = generateScreenKey(),
) : Screen {

    @Composable
    override fun Content() {
        val parent = LocalContainerScreen.current
        SampleContent(i, parent)
    }
}
```

Modo: Screen

```
@Parcelize
internal class SampleScreen(
    private val title: String,
    override val screenKey: ScreenKey = generateScreenKey(),
) : Screen {

    @Composable
    override fun Content() {
        val parent = LocalContainerScreen.current
        SampleContent(i, parent)
    }
}
```

Modo: Screen

```
@Parcelize
internal class SampleScreen(
    private val title: String,
    override val screenKey: ScreenKey = generateScreenKey(),
) : Screen {

    @Composable
    override fun Content() {
        val parent = LocalContainerScreen.current
        SampleContent(i, parent)
    }
}
```

Modo: Screen

```
@Parcelize
internal class SampleScreen(
    private val title: String,
    override val screenKey: ScreenKey = generateScreenKey(),
) : Screen {

    @Composable
    override fun Content() {
        val parent = LocalContainerScreen.current
        SampleContent(i, parent)
    }
}
```

Modo: Screen

```
@Parcelize
internal class SampleScreen(
    private val title: String,
    override val screenKey: ScreenKey = generateScreenKey(),
) : Screen {

    @Composable
    override fun Content() {
        val parent = LocalContainerScreen.current
        SampleContent(i, parent)
    }
}
```

Modo: Screen

```
@Parcelize
internal class SampleScreen(
    private val title: String,
    override val screenKey: ScreenKey = generateScreenKey(),
) : Screen {

    @Composable
    override fun Content() {
        val parent = LocalContainerScreen.current
        SampleContent(i, parent)
    }
}
```

Modo: Screen

```
@Parcelize
internal class SampleScreen(
    private val title: String,
    override val screenKey: ScreenKey = generateScreenKey(),
) : Screen {

    @Composable
    override fun Content() {
        val parent = LocalContainerScreen.current
        SampleContent(i, parent)
    }
}
```

Modo

1. Fragment
2. ContainerScreen/Screen
3. **Навигируемся**
4. NavModel/ContainerScreen
5. Поддержка анимаций, dialog

Modo: навигируемся

```
@Composable
override fun Content() {
    val parent = LocalContainerScreen.current
    val showNotSupported = LocalShowNotSupported.current
    SampleScreenContent(
        openScreen = { parent.forward(SampleScreen(randomString())) },
        openDialog = { parent.forward(SampleDialogScreen(randomString())) },
        openBottomSheet = { showNotSupported() },
        // этот кейс просто копия кейса ниже, поэтому не будем его рисовать
        startFlowWithoutHeader = null,
        startFlowWithHeader = {
            parent.forward(FlowNavigationScreen(randomString()))
        },
        openMultiscreen = { parent.forward(MultiStackScreen(1)) },
        openScreenModel = { },
        openComplexNavigation = {
            parent.forward(ComplexNavigationScreen(randomString()))
        },
        screenTitle = title,
```

Modo: навигируемся

```
@Composable
override fun Content() {
    val parent = LocalContainerScreen.current
    val showNotSupported = LocalShowNotSupported.current
    SampleScreenContent(
        openScreen = { parent.forward(SampleScreen(randomString())) },
        openDialog = { parent.forward(SampleDialogScreen(randomString())) },
        openBottomSheet = { showNotSupported() },
        // этот кейс просто копия кейса ниже, поэтому не будем его рисовать
        startFlowWithoutHeader = null,
        startFlowWithHeader = {
            parent.forward(FlowNavigationScreen(randomString()))
        },
        openMultiscreen = { parent.forward(MultiStackScreen(1)) },
        openScreenModel = { },
        openComplexNavigation = {
            parent.forward(ComplexNavigationScreen(randomString()))
        },
        screenTitle = title,
```

Modo: навигируемся

```
@Composable
override fun Content() {
    val parent = LocalContainerScreen.current
    val showNotSupported = LocalShowNotSupported.current
    SampleScreenContent(
        openScreen = { parent.forward(SampleScreen(randomString())) },
        openDialog = { parent.forward(SampleDialogScreen(randomString())) },
        openBottomSheet = { showNotSupported() },
        // этот кейс просто копия кейса ниже, поэтому не будем его рисовать
        startFlowWithoutHeader = null,
        startFlowWithHeader = {
            parent.forward(FlowNavigationScreen(randomString()))
        },
        openMultiscreen = { parent.forward(MultiStackScreen(1)) },
        openScreenModel = { },
        openComplexNavigation = {
            parent.forward(ComplexNavigationScreen(randomString()))
        },
        screenTitle = title,
```

Modo

1. Fragment
2. ContainerScreen/Screen
3. Навигируемся
4. **NavModel/ContainerScreen**
5. Поддержка анимаций, dialog

Modo: NavModel

```
fun NavigationContainer<StackState>.forward(
    screen: Screen,
    vararg screens: Screen
) = dispatch(Forward(screen, *screens))
```

Modo: NavModel

```
fun NavigationContainer<StackState>.forward(
    screen: Screen,
    vararg screens: Screen
) = dispatch(Forward(screen, *screens))

interface NavigationContainer<State : NavigationState> {
    val navigationState: State

    fun dispatch(action: NavigationAction)
}
```

Modo: NavModel

```
...

interface NavigationContainer<State : NavigationState> {
    val navigationState: State

    fun dispatch(action: NavigationAction)
}

@Parcelize
data class StackState(
    val stack: List<Screen> = emptyList(),
) : NavigationState, Parcelable {

    override fun getChildScreens(): List<Screen> = stack

}
```

Modo: NavModel

```
abstract class ContainerScreen<State : NavigationState>(
    private val navModel: NavModel<State>
) : Screen, NavigationContainer<State> by navModel {
    abstract val reducer: NavigationReducer<State>
    ...
    init {
        navModel.init(
            reducerProvider = { reducer },
            renderer = ComposeRenderer(this)
        )
    }
    @Composable
    protected fun InternalContent(
        screen: Screen,
        content: RendererContent<State> = defaultRendererContent
    ) {
        val composeRenderer = renderer as ComposeRenderer
        composeRenderer.Content(screen, content)
    }
```

Modo: NavModel

```
abstract class ContainerScreen<State : NavigationState>(
    private val navModel: NavModel<State>
) : Screen, NavigationContainer<State> by navModel {
    abstract val reducer: NavigationReducer<State>
    ...
    init {
        navModel.init(
            reducerProvider = { reducer },
            renderer = ComposeRenderer(this)
        )
    }
    @Composable
    protected fun InternalContent(
        screen: Screen,
        content: RendererContent<State> = defaultRendererContent
    ) {
        val composeRenderer = renderer as ComposeRenderer
        composeRenderer.Content(screen, content)
    }
```

Modo: NavModel

```
abstract class ContainerScreen<State : NavigationState>(
    private val navModel: NavModel<State>
) : Screen, NavigationContainer<State> by navModel {
    abstract val reducer: NavigationReducer<State>
    ...
    init {
        navModel.init(
            reducerProvider = { reducer },
            renderer = ComposeRenderer(this)
        )
    }
    @Composable
    protected fun InternalContent(
        screen: Screen,
        content: RendererContent<State> = defaultRendererContent
    ) {
        val composeRenderer = renderer as ComposeRenderer
        composeRenderer.Content(screen, content)
    }
```

Modo: NavModel

```
abstract class ContainerScreen<State : NavigationState>(
    private val navModel: NavModel<State>
) : Screen, NavigationContainer<State> by navModel {
    abstract val reducer: NavigationReducer<State>
    ...
    init {
        navModel.init(
            reducerProvider = { reducer },
            renderer = ComposeRenderer(this)
        )
    }
    @Composable
    protected fun InternalContent(
        screen: Screen,
        content: RendererContent<State> = defaultRendererContent
    ) {
        val composeRenderer = renderer as ComposeRenderer
        composeRenderer.Content(screen, content)
    }
```

Modo: NavModel

```
class NavModel<State : NavigationState>(
    initialState: State,
    val screenKey: ScreenKey = generateScreenKey()
) : NavigationContainer<State>, Parcelable {
    override var navigationState: State = initialState
        get() = (renderer as ComposeRenderer<State>).state ?: field
        set(value) {
            field = value
            renderer?.render(value)
        }
    private var reducerProvider: ReducerProvider<State>? = null
    internal var renderer: ComposeRenderer<State>? = null
        private set

    override fun dispatch(action: NavigationAction) {
        val reducer = reducerProvider!!()
        navigationState = reduce(reducer, navigationState, action)
    }
```

Modo: NavModel

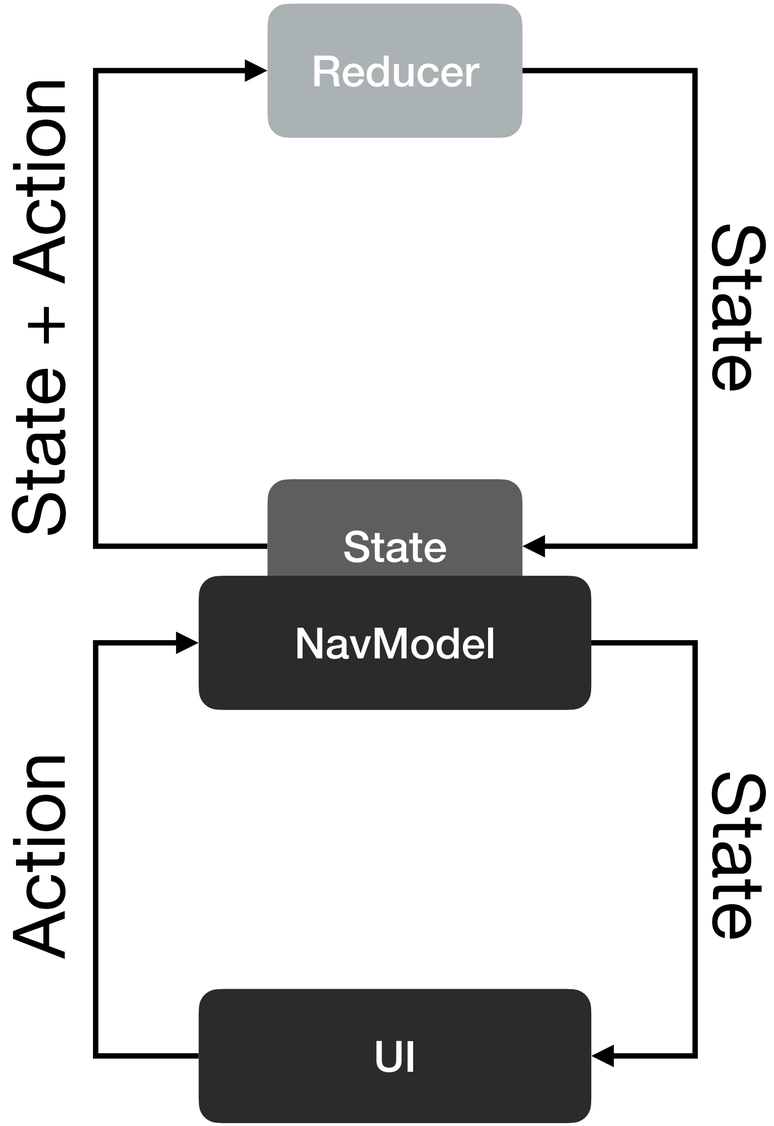
```
class NavModel<State : NavigationState>(
    initialState: State,
    val screenKey: ScreenKey = generateScreenKey()
) : NavigationContainer<State>, Parcelable {
    override var navigationState: State = initialState
        get() = (renderer as ComposeRenderer<State>).state ?: field
        set(value) {
            field = value
            renderer?.render(value)
        }
    private var reducerProvider: ReducerProvider<State>? = null
    internal var renderer: ComposeRenderer<State>? = null
        private set

    override fun dispatch(action: NavigationAction) {
        val reducer = reducerProvider!!()
        navigationState = reduce(reducer, navigationState, action)
    }
```

Modo: NavModel

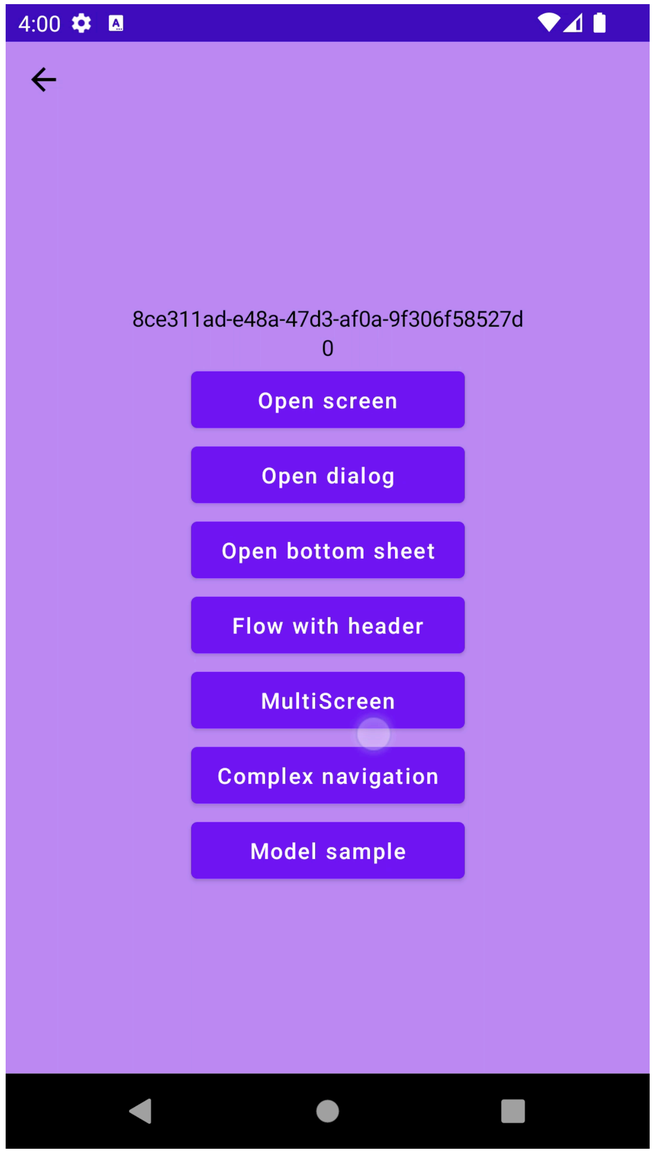
```
class NavModel<State : NavigationState>(
    initialState: State,
    val screenKey: ScreenKey = generateScreenKey()
) : NavigationContainer<State>, Parcelable {
    override var navigationState: State = initialState
        get() = (renderer as ComposeRenderer<State>).state ?: field
        set(value) {
            field = value
            renderer?.render(value)
        }
    private var reducerProvider: ReducerProvider<State>? = null
    internal var renderer: ComposeRenderer<State>? = null
        private set

    override fun dispatch(action: NavigationAction) {
        val reducer = reducerProvider!!()
        navigationState = reduce(reducer, navigationState, action)
    }
```

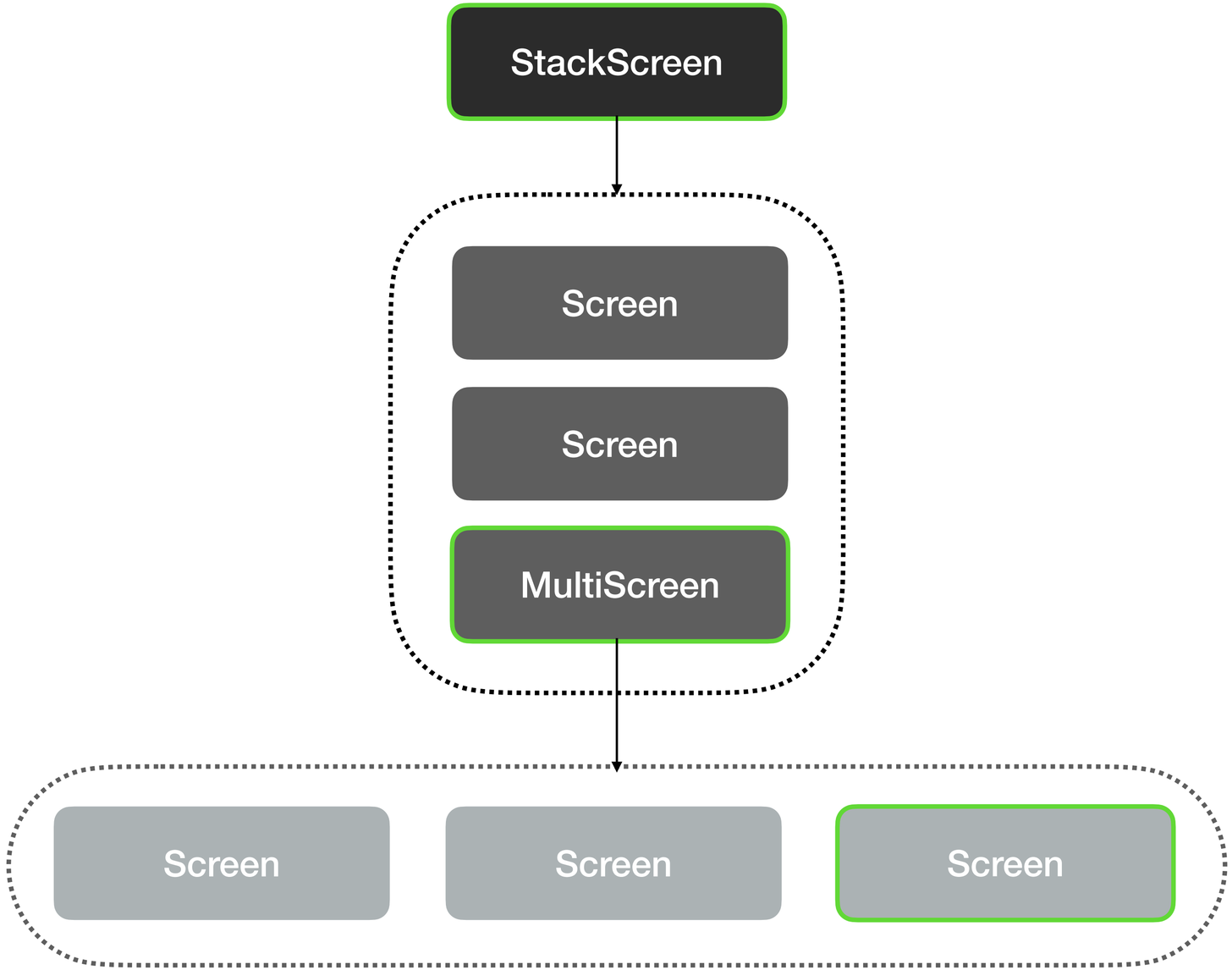
Выглядит как UDF

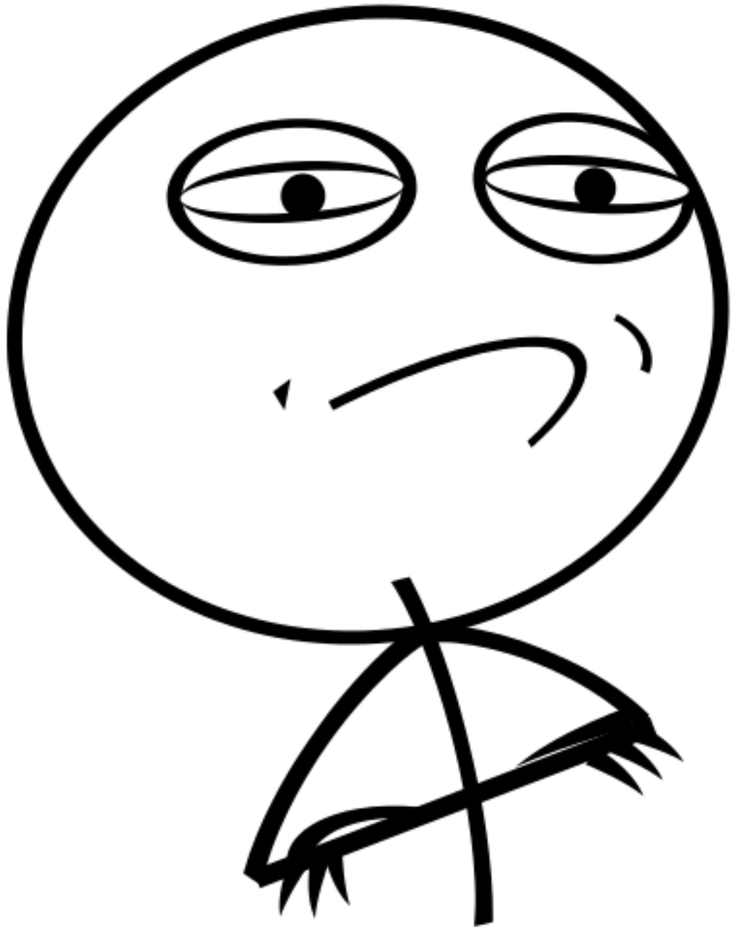
Modo: UDF



Modo: Граф экранов







Можно установить

Произвольный State

Modo

1. Fragment
2. ContainerScreen/Screen
3. Навигируемся
4. NavModel/ContainerScreen
5. **Поддержка анимаций, dialog**

Modo: Animation

```
@Parcelize
internal class ModoStackScreen(
    private val navModel: StackNavModel =
        StackNavModel(SampleScreen(randomString())),
) : StackScreen(navModel) {

    @Composable
    override fun Content() {
        TopScreenContent { SlideTransition() }
    }
}
```

Modo: Animation

```
@Parcelize
internal class ModoStackScreen(
    private val navModel: StackNavModel =
        StackNavModel(SampleScreen(randomString())),
) : StackScreen(navModel) {

    @Composable
    override fun Content() {
        TopScreenContent { SlideTransition() }
    }
}
```

Modo: Animation

```
@Composable
protected fun TopScreenContent(
    content: RendererContent<StackState> = defaultRendererContent
)
```

Modo: Animation

```
@Composable
protected fun TopScreenContent(
    content: RendererContent<StackState> = defaultRendererContent
)

typealias RendererContent<State> =
    @Composable ComposeRendererScope<State>.() -> Unit
```

Modo: Animation

```
@Composable
protected fun TopScreenContent(
    content: RendererContent<StackState> = defaultRendererContent
)

typealias RendererContent<State> =
    @Composable ComposeRendererScope<State>.() -> Unit

class ComposeRendererScope<State: NavigationState>(
    val oldState: State?,
    val newState: State?,
    val screen: Screen,
)
```

Modo: Animation

```
@Composable
protected fun TopScreenContent(
    content: RendererContent<StackState> = defaultRendererContent
)

typealias RendererContent<State> =
    @Composable ComposeRendererScope<State>.() -> Unit

class ComposeRendererScope<State: NavigationState>(
    val oldState: State?,
    val newState: State?,
    val screen: Screen,
)
```

Modo: Animation

```
@Composable
@OptIn(ExperimentalAnimationApi::class)
fun ComposeRendererScope<StackState>.SlideTransition() {
    ScreenTransition(
        transitionSpec = {
            val transitionType = calculateStackTransitionType()
            ...
        }
    )
}
```

Modo: Animation

```
val transitionType = calculateStackTransitionType()
```

Modo: Animation

```
val transitionType: StackTransitionType =
                                calculateStackTransitionType(oldState, newState)
```

Modo: Animation

```
val transitionType: StackTransitionType =
                              calculateStackTransitionType(oldState, newState)

enum class StackTransitionType {
    Push,
    Replace,
    Pop,
    Idle
}
```

Modo: Animation

```
val transitionType = calculateStackTransitionType()
when {
    transitionType == StackTransitionType.Replace -> {
        scaleIn(initialScale = 2f) + fadeIn() with fadeOut()
    }
    else -> {
        val (initialOffset, targetOffset) = when (transitionType) {
            StackTransitionType.Pop ->
                ({ size: Int -> -size }) to ({ size: Int -> size })
            else ->
                ({ size: Int -> size }) to ({ size: Int -> -size })
        }
        slideInHorizontally(initialOffsetX = initialOffset) with
                slideOutHorizontally(targetOffsetX = targetOffset)
    }
}
```

Modo: Animation

```
val transitionType = calculateStackTransitionType()
when {
    transitionType == StackTransitionType.Replace -> {
        scaleIn(initialScale = 2f) + fadeIn() with fadeOut()
    }
    else -> {
        val (initialOffset, targetOffset) = when (transitionType) {
            StackTransitionType.Pop ->
                ({ size: Int -> -size }) to ({ size: Int -> size })
            else ->
                ({ size: Int -> size }) to ({ size: Int -> -size })
        }
        slideInHorizontally(initialOffsetX = initialOffset) with
                slideOutHorizontally(targetOffsetX = targetOffset)
    }
}
```

Modo: Animation

```
val transitionType = calculateStackTransitionType()
when {
    transitionType == StackTransitionType.Replace -> {
        scaleIn(initialScale = 2f) + fadeIn() with fadeOut()
    }
    else -> {
        val (initialOffset, targetOffset) = when (transitionType) {
            StackTransitionType.Pop ->
                ({ size: Int -> -size }) to ({ size: Int -> size })
            else ->
                ({ size: Int -> size }) to ({ size: Int -> -size })
        }
        slideInHorizontally(initialOffsetX = initialOffset) with
                slideOutHorizontally(targetOffsetX = targetOffset)
    }
}
```

Modo: Animation

```
val transitionType = calculateStackTransitionType()
when {
    transitionType == StackTransitionType.Replace -> {
        scaleIn(initialScale = 2f) + fadeIn() with fadeOut()
    }
    else -> {
        val (initialOffset, targetOffset) = when (transitionType) {
            StackTransitionType.Pop ->
                ({ size: Int -> -size }) to ({ size: Int -> size })
            else ->
                ({ size: Int -> size }) to ({ size: Int -> -size })
        }
        slideInHorizontally(initialOffsetX = initialOffset) with
                slideOutHorizontally(targetOffsetX = targetOffset)
    }
}
```

Modo: Animation

```
val transitionType = calculateStackTransitionType()
when {
    transitionType == StackTransitionType.Replace -> {
        scaleIn(initialScale = 2f) + fadeIn() with fadeOut()
    }
    else -> {
        val (initialOffset, targetOffset) = when (transitionType) {
            StackTransitionType.Pop ->
                ({ size: Int -> -size }) to ({ size: Int -> size })
            else ->
                ({ size: Int -> size }) to ({ size: Int -> -size })
        }
        slideInHorizontally(initialOffsetX = initialOffset) with
                slideOutHorizontally(targetOffsetX = targetOffset)
    }
}
```

Modo: Animation

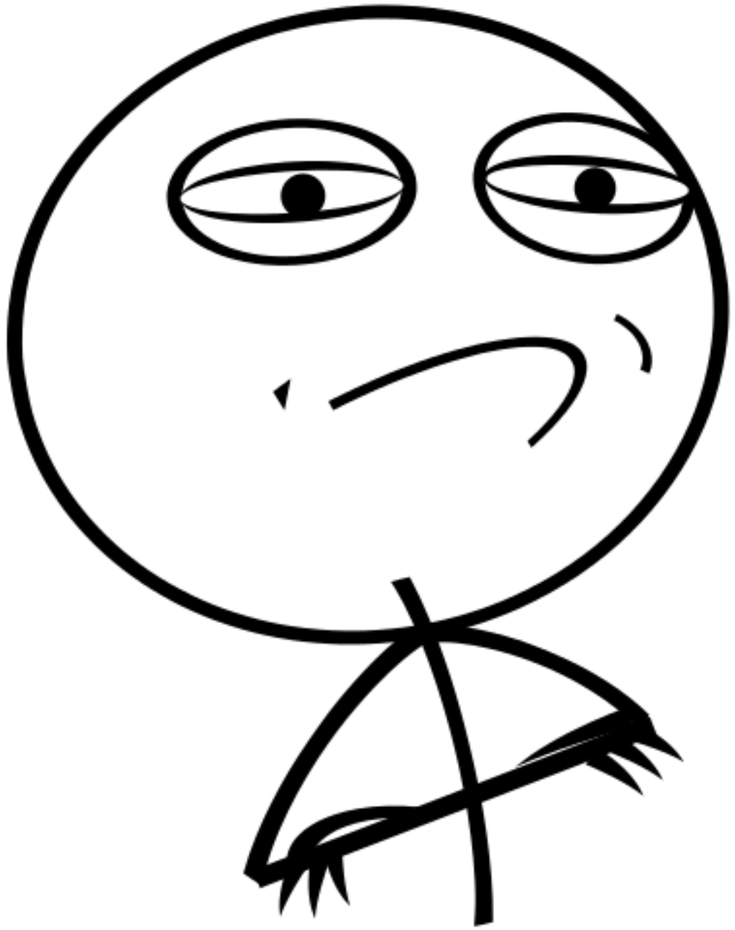
```
val transitionType = calculateStackTransitionType()
when {
    transitionType == StackTransitionType.Replace -> {
        scaleIn(initialScale = 2f) + fadeIn() with fadeOut()
    }
    else -> {
        val (initialOffset, targetOffset) = when (transitionType) {
            StackTransitionType.Pop ->
                ({ size: Int -> -size }) to ({ size: Int -> size })
            else ->
                ({ size: Int -> size }) to ({ size: Int -> -size })
        }
        slideInHorizontally(initialOffsetX = initialOffset) with
                slideOutHorizontally(targetOffsetX = targetOffset)
    }
}
```

Modo: плюсы

1. UDF + простое изменение состояния
2. Корректность
3. Базовый набор команд как у Cicerone
4. Простота реализации внутри
5. Кастомизации (анимации, контейнеры и т.п.)

Modo: минусы

1. Обслуживающий код для Di
2. Немного boilerplate для экранов-контейнеров
3. Нет Bottom sheet
4. Всё ещё в разработке



Мы выбрали

MODO

Почему?

Почему не что-то другое?

1. Простота создания Screen
2. Экраны вложены в друг-друга на уровне моделей (в Аррух сложнее)
3. Состояние - источник правды

Дальнейшие шаги

1. Завозить в реальные фичи проекта
2. Пример связывания с UDF framework (TEA, MVI)
3. Покрытие Modo UI тестами
4. Стабилизация диалогов
5. BottomSheet

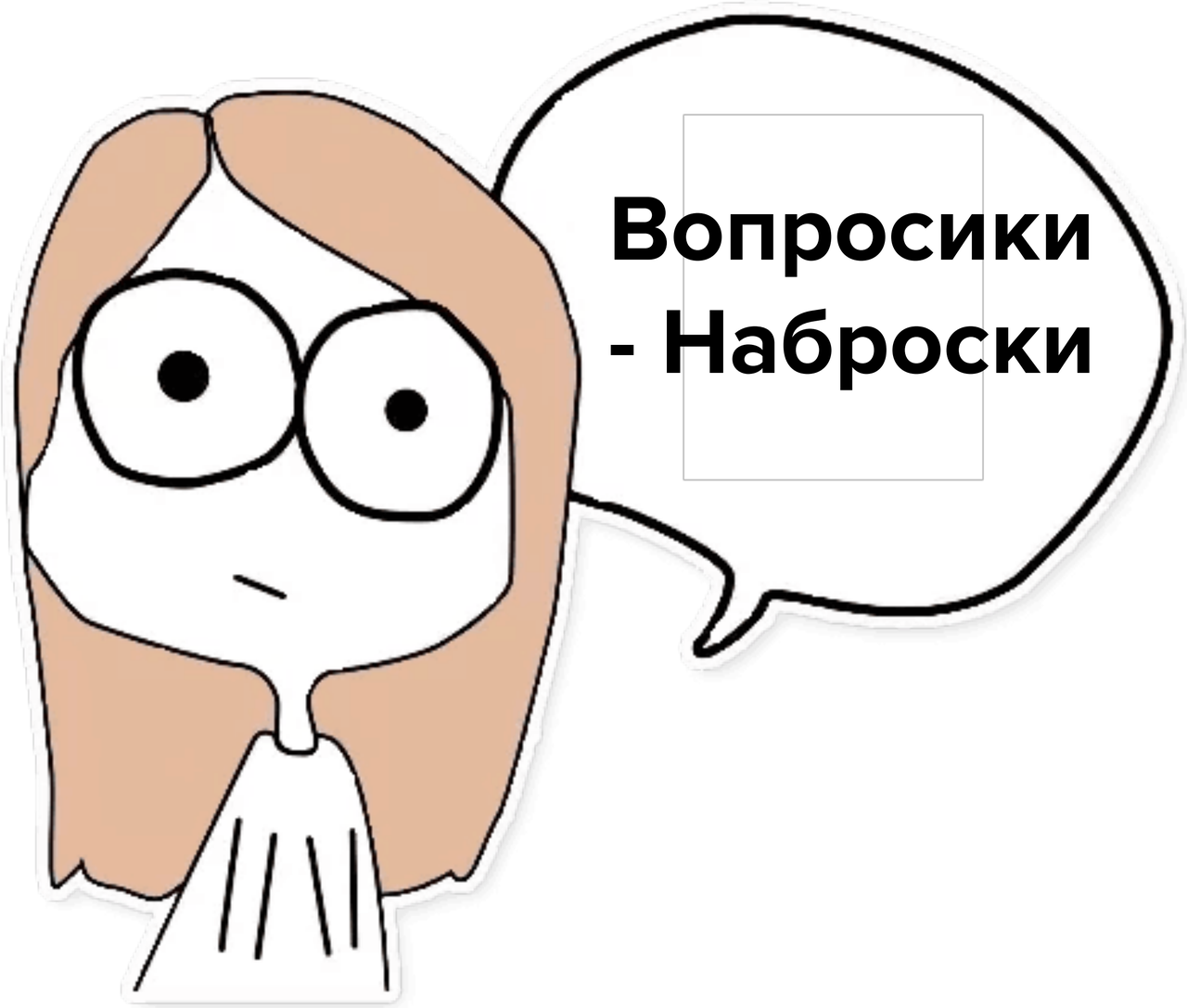
HH Tech

https://t.me/hh_tech

https://t.me/hh_tech_news

Что мы сегодня узнали

1. Как работает rememberSaveable
2. 4 классные библиотеки навигации
3. Каждая библиотека уникальна
4. Modo


Вопросики
- Наброски