Нюансы перехода на Jetpack Compose

Алексей Панов
Контур

О чем доклад

- Стейт менеджмент в Jetpack Compose
- Работа с эффектами
- Будущее Android разработки
- План рефакторинга
  - Работа с темой
  - Понимание рекомпозиции
  - Compose и MVP/MVVM/MVI
  - Навигация

Обо мне

- Ведущий Android разработчик в Контуре
- Живу в Екатеринбурге
- Выступал с докладами по Kotlin и Flutter
- Фанат новых технологий

Алексей Панов

Jetpack Compose

Jetpack Compose — это новый способ создания пользовательского интерфейса в декларативном стиле с помощью Kotlin кода

Jetpack Compose

Jetpack Compose — это новый способ создания пользовательского интерфейса в декларативном стиле с помощью Kotlin кода

Декларативный подход

В императивном –
конструируем виджеты

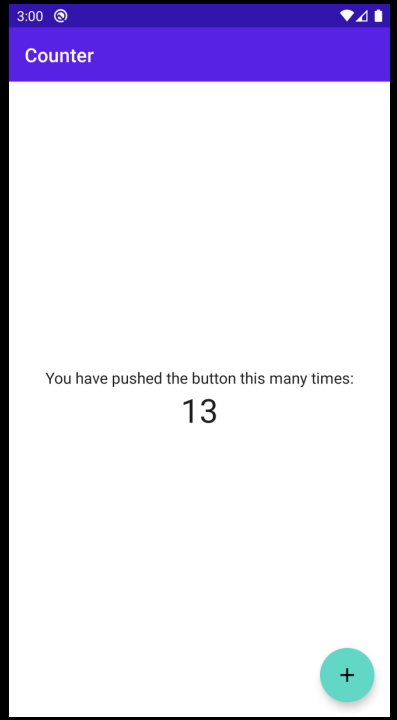
```
val layout = FrameLayout( context: this)
val textView = TextView( context: this)
textView.text = "Hello world"
layout.addView(textView)
```

В декларативном –
описываем экран

```
@Composable
fun Greeting(name: String) {
    Text( text: "Hello $name")
}
```

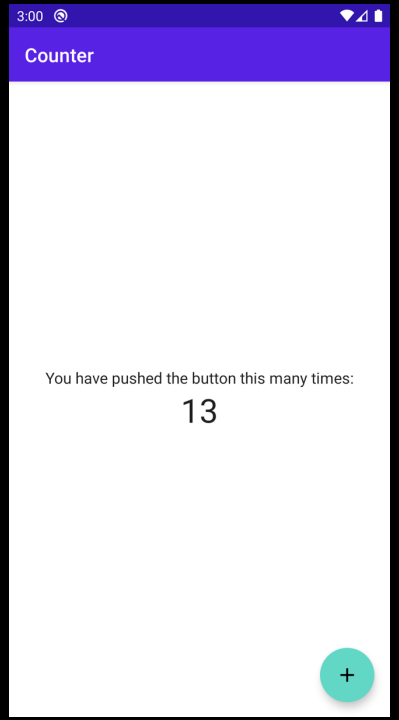
Jetpack Compose

```
@Composable
fun MyApp() {
    ComposeTalkTheme {
        MyHomePage( title: "Counter")
    }
}

@Composable
fun MyHomePage(title: String) {
    var counter by remember { mutableStateOf( value: 0) }

    Scaffold(
        topBar = { TopAppBar(title = { Text(title) }) },
        floatingActionButton = {
            FloatingActionButton(onClick = { counter++ }) {
                Icon(Icons.Default.Add, contentDescription = "Increment")
            }
        }
    ) { it: PaddingValues
        Column(
            Modifier.fillMaxSize(),
            horizontalAlignment = Alignment.CenterHorizontally,
            verticalArrangement = Arrangement.Center
        ) { this: ColumnScope
            Text( text: "You have pushed the button this many times:")
            Text(counter.toString(), style = MaterialTheme.typography.h4)
        }
    }
}
```



Jetpack Compose

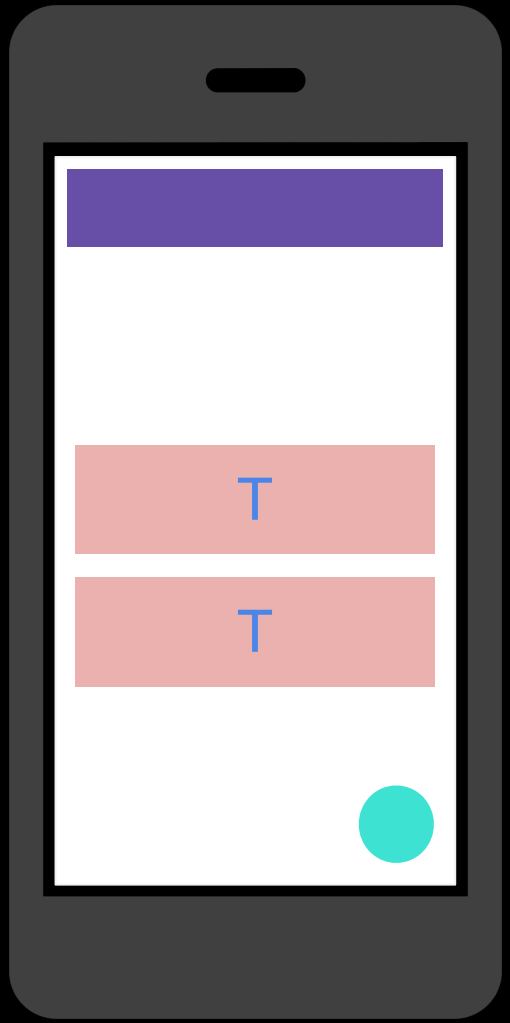
```
@Composable
fun MyApp() {
    ComposeTalkTheme {
        MyHomePage( title: "Counter")
    }
}

@Composable
fun MyHomePage(title: String) {
    var counter by remember { mutableStateOf( value: 0) }

    Scaffold(
        topBar = { TopAppBar(title = { Text(title) }) },
        floatingActionButton = {
            FloatingActionButton(onClick = { counter++ }) {
                Icon(Icons.Default.Add, contentDescription = "Increment")
            }
        }
    ) { it: PaddingValues
        Column(
            Modifier.fillMaxSize(),
            horizontalAlignment = Alignment.CenterHorizontally,
            verticalArrangement = Arrangement.Center
        ) { this: ColumnScope
            Text( text: "You have pushed the button this many times:")
            Text(counter.toString(), style = MaterialTheme.typography.h4)
        }
    }
}
```

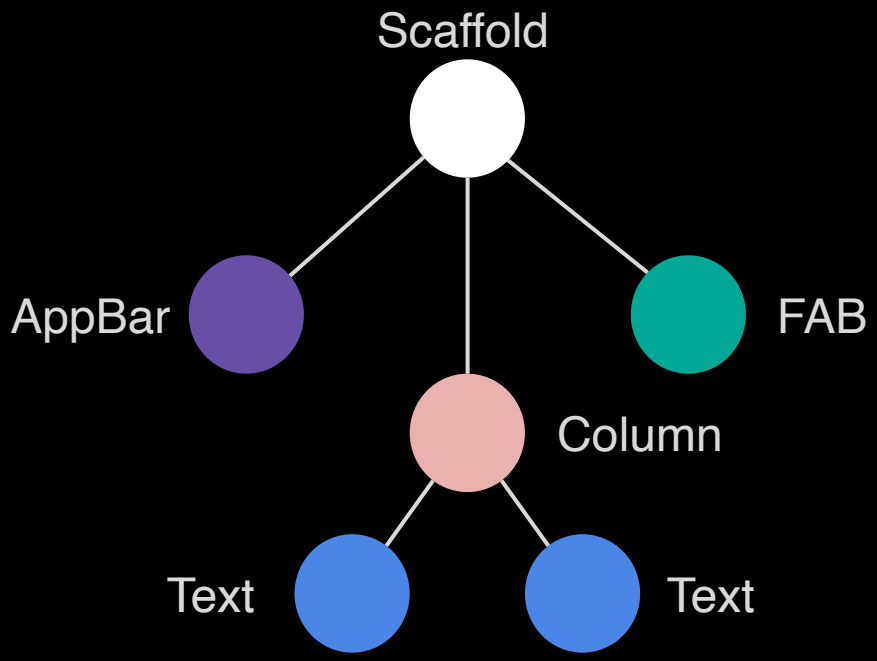


Flutter

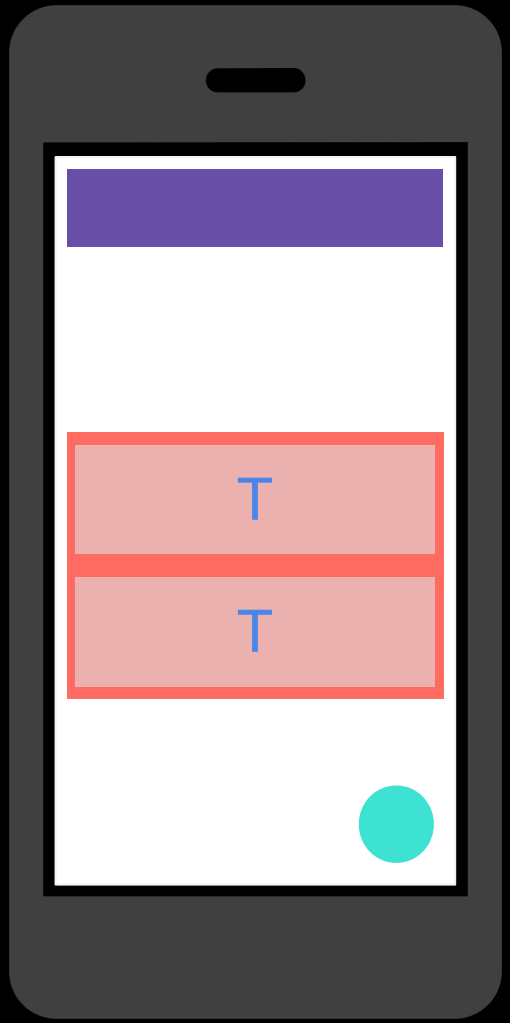
```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      theme: ThemeData(primarySwatch: Colors.blue),
      home: MyHomePage(title: 'Counter'),
    ); // MaterialApp
  }
}

class MyHomePage extends StatefulWidget {
  MyHomePage({Key key, this.title}) : super(key: key);

  final String title;

  @override
  _MyHomePageState createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  int _counter = 0;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text(widget.title)),
      body: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        crossAxisAlignment: CrossAxisAlignment.center,
        children: <Widget>[
          Text('You have pushed the button this many times:'),
          Text('$_counter', style: Theme.of(context).textTheme.headline4)
        ], // <Widget>[]
      ), // Column
      floatingActionButton: FloatingActionButton(
        onPressed: () {
          setState(() { _counter++; });
        },
        tooltip: 'Increment',
        child: Icon(Icons.add),
      ), // FloatingActionButton
    ); // Scaffold
  }
}
```

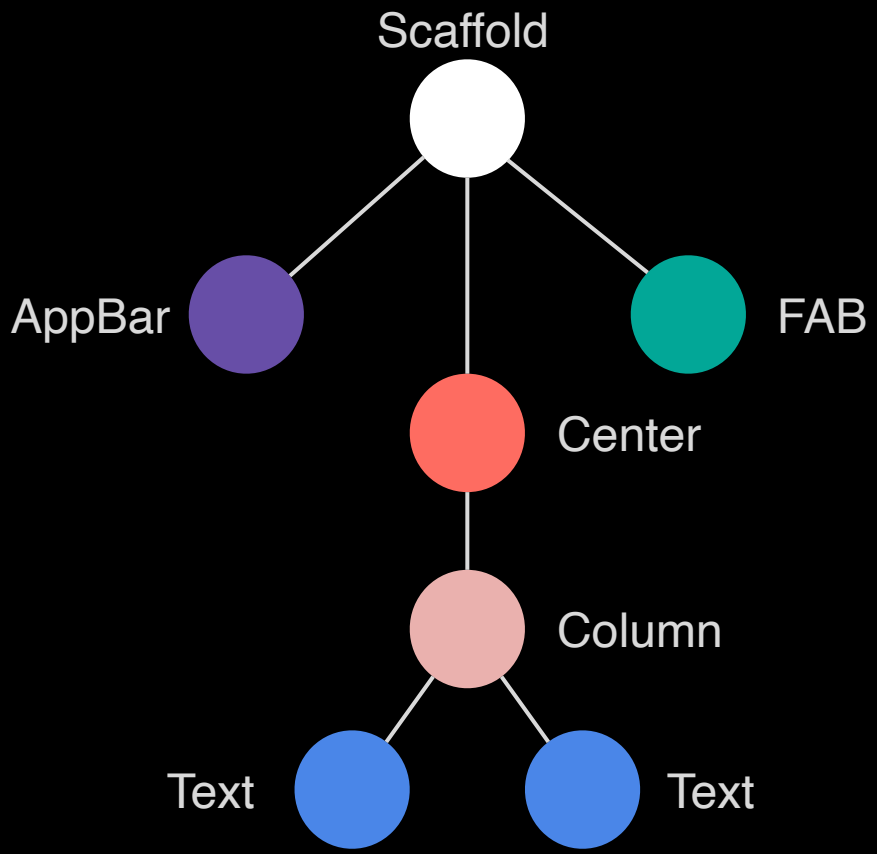
Jetpack Compose





Flutter





Анатомия Composable функции

```
@Composable
fun Greeting(name: String) {
    Text( text: "Hello $name")
}
```

Анатомия Composable функции

```
@Composable
fun Greeting(name: String) {
    Text( text: "Hello $name")
}
```

- Должна иметь @Composable аннотацию

Анатомия Composable функции

```
@Composable
fun Greeting(name: String) {
    Text( text: "Hello $name")
}
```

- Должна иметь @Composable аннотацию
- Создает UI, вызывая другие Composable функции

Анатомия Composable функции

```
@Composable
fun Greeting(name: String) {
    Text( text: "Hello $name")
}
```

- Должна иметь @Composable аннотацию
- Создает UI, вызывая другие Composable функции
- Ничего не возвращает

Анатомия Composable функции

```
@Composable
fun Greeting(name: String) {
    Text( text: "Hello $name")
}
```

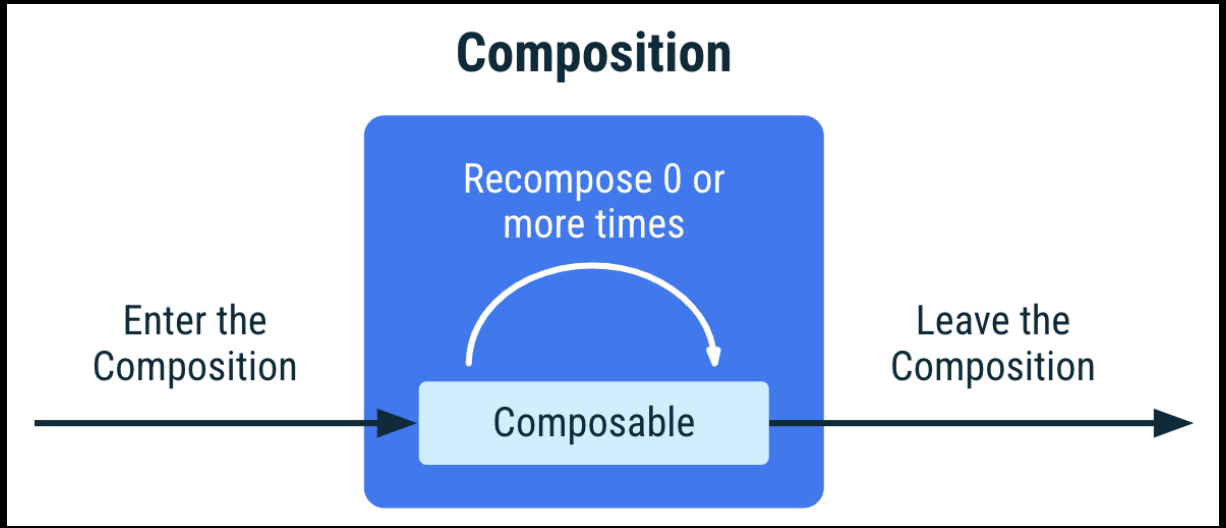
- Должна иметь @Composable аннотацию
- Создает UI, вызывая другие Composable функции
- Ничего не возвращает
- Не может быть вызвана из обычной функции

Анатомия Composable функции

```
@Composable
fun Greeting(name: String) {
    Text( text: "Hello $name")
}
```

- Должна иметь @Composable аннотацию
- Создает UI, вызывая другие Composable функции
- Ничего не возвращает
- Не может быть вызвана из обычной функции
- Должна быть быстрой, идемпотентной и без сайд эффектов

Жизненный цикл



Пропуск рекомпозиции

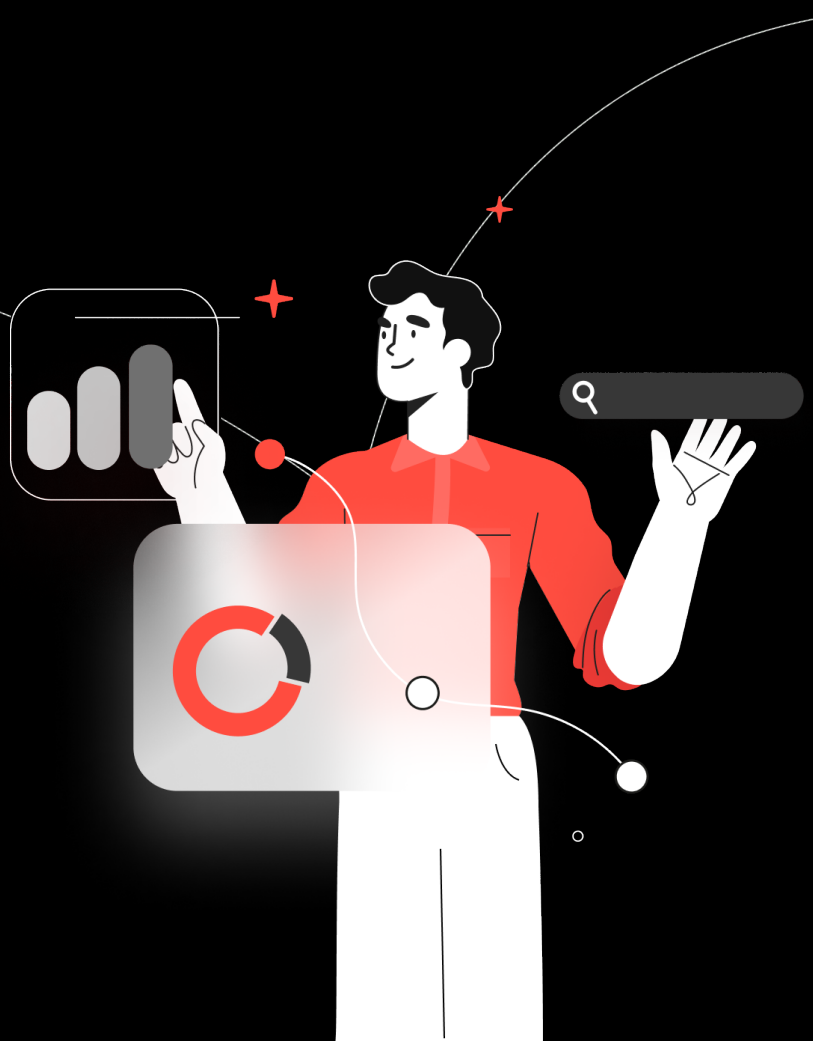
Рекомпозиция может быть пропущена, если все входные параметры являются Stable и не изменились

Stable типы:

- Все примитивы: Boolean, Int, Long, Float, Char и т.д.
- Строки
- Функциональные типы (лямбды)

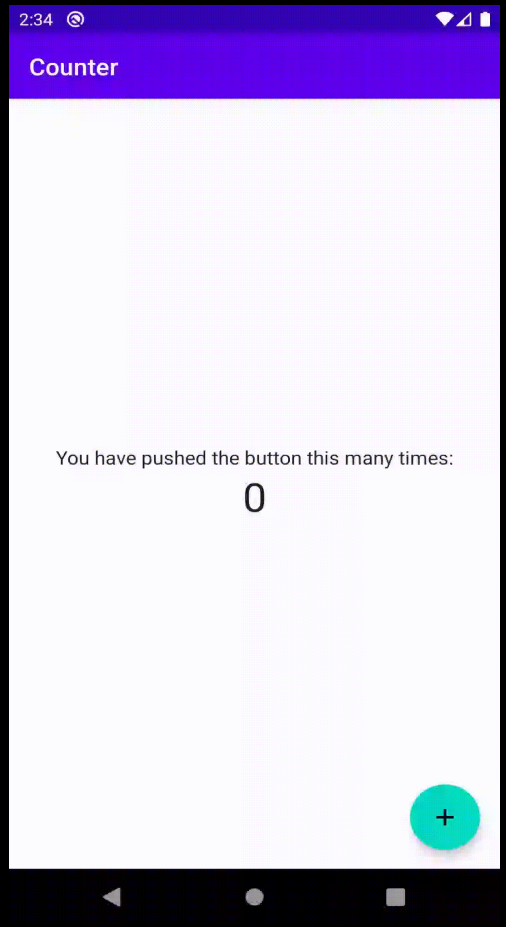
Все вложенные неизменяемые типы будут также выведены как Stable

Разбираемся со стейтом



```
@Composable
fun MyHomePage(title: String) {
    var counter  = 0

    Scaffold(
        topBar = { TopAppBar(title = { Text(title) }) },
        floatingActionButton = {
            FloatingActionButton(onClick = { counter++ }) {
                Icon(Icons.Default.Add, contentDescription = "Increment")
            }
        }
    ) { it: PaddingValues
        Column(
            Modifier.fillMaxSize(),
            horizontalAlignment = Alignment.CenterHorizontally,
            verticalArrangement = Arrangement.Center
        ) { this: ColumnScope
            Text( text: "You have pushed the button this many times:")
            Text(counter.toString(), style = MaterialTheme.typography.h4)
        }
    }
}
```

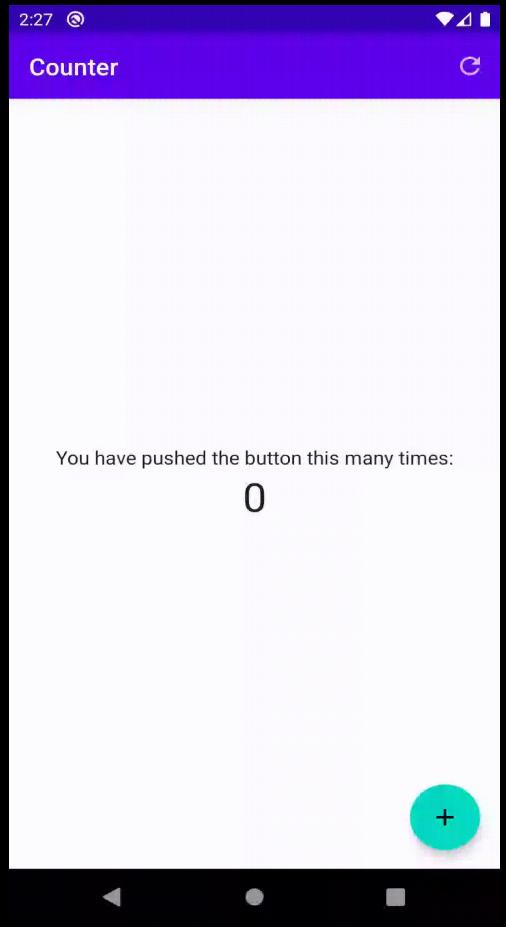
```
@Composable
fun MyHomePage(title: String) {
    var counter  = 0

    Scaffold(
        topBar = { TopAppBar(title = { Text(title) }) },
        floatingActionButton = {
            FloatingActionButton(onClick = { counter++ }) {
                Icon(Icons.Default.Add, contentDescription = "Increment")
            }
        }
    ) { it: PaddingValues
        Column(
            Modifier.fillMaxSize(),
            horizontalAlignment = Alignment.CenterHorizontally,
            verticalArrangement = Arrangement.Center
        ) { this: ColumnScope
            Text( text: "You have pushed the button this many times:")
            Text(counter.toString(), style = MaterialTheme.typography.h4)
        }
    }
}
```



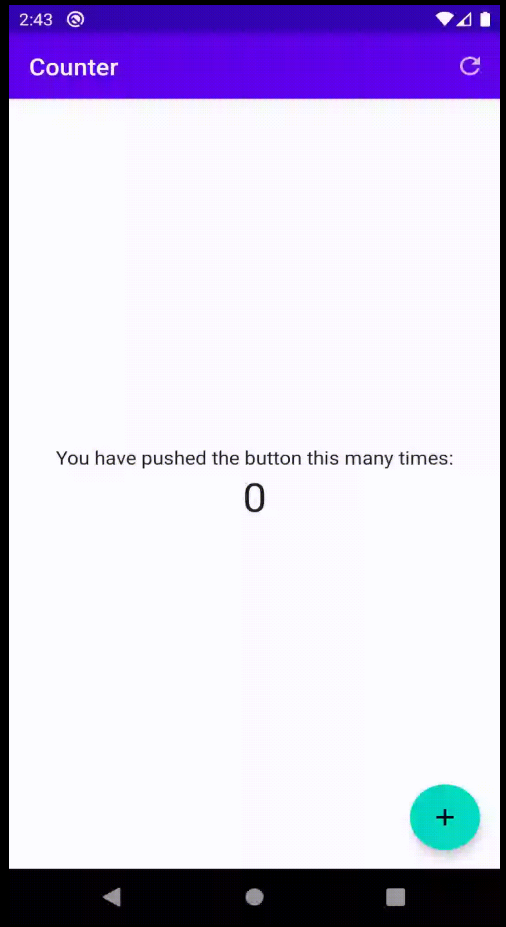
```kotlin
@Stable
interface State<out T> {
    val value: T
}

@Stable
interface MutableState<T> : State<T> {
    override var value: T
    operator fun component1(): T
    operator fun component2(): (T) -> Unit
}
```

```
@Composable
fun MyHomePage(title: String, onTitleChanged: () → Unit) {
    var counter by mutableStateOf( value: 0)

    Scaffold(
        topBar = { TopAppBar(title = { Text(title) }, actions = {...}) },
        floatingActionButton = {
            FloatingActionButton(onClick = { counter++ }) {
                Icon(Icons.Default.Add, contentDescription = "Increment")
            }
        }
    ) { it: PaddingValues
        Column(
            Modifier.fillMaxSize(),
            horizontalAlignment = Alignment.CenterHorizontally,
            verticalArrangement = Arrangement.Center
        ) { this: ColumnScope
            Text( text: "You have pushed the button this many times:")
            Text(counter.toString(), style = MaterialTheme.typography.h4)
        }
    }
}
```

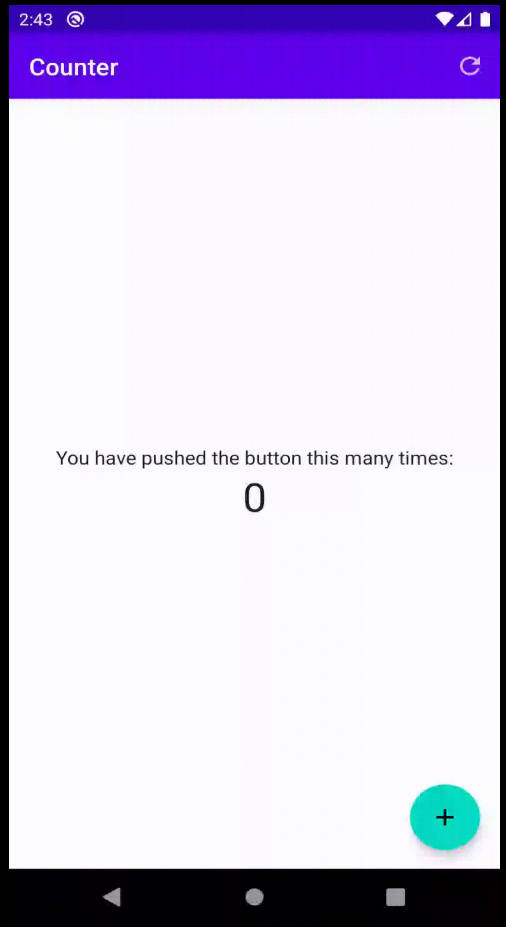
```
@Composable
fun MyHomePage(title: String, onTitleChanged: () → Unit) {
    var counter by mutableStateOf( value: 0)

    Scaffold(
        topBar = { TopAppBar(title = { Text(title) }, actions = {...}) },
        floatingActionButton = {
            FloatingActionButton(onClick = { counter++ }) {
                Icon(Icons.Default.Add, contentDescription = "Increment")
            }
        }
    ) { it: PaddingValues
        Column(
            Modifier.fillMaxSize(),
            horizontalAlignment = Alignment.CenterHorizontally,
            verticalArrangement = Arrangement.Center
        ) { this: ColumnScope
            Text( text: "You have pushed the button this many times:")
            Text(counter.toString(), style = MaterialTheme.typography.h4)
        }
    }
}
```



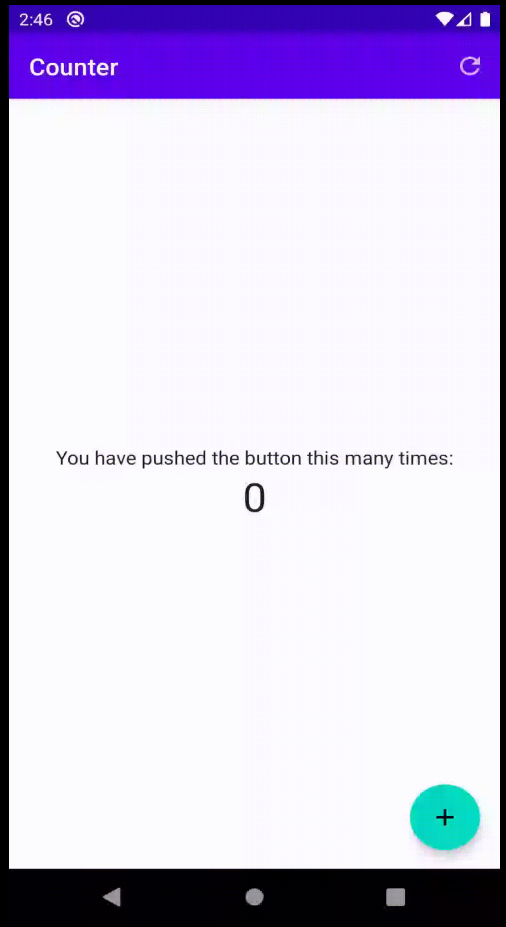
```
@Composable
fun MyHomePage(title: String, onTitleChanged: () → Unit) {
    var counter by remember { mutableStateOf( value: 0) }

    Scaffold(
        topBar = { TopAppBar(title = { Text(title) }, actions = {...}) },
        floatingActionButton = {
            FloatingActionButton(onClick = { counter++ }) {
                Icon(Icons.Default.Add, contentDescription = "Increment")
            }
        }
    ) { it: PaddingValues
        Column(
            Modifier.fillMaxSize(),
            horizontalAlignment = Alignment.CenterHorizontally,
            verticalArrangement = Arrangement.Center
        ) { this: ColumnScope
            Text( text: "You have pushed the button this many times:")
            Text(counter.toString(), style = MaterialTheme.typography.h4)
        }
    }
}
```



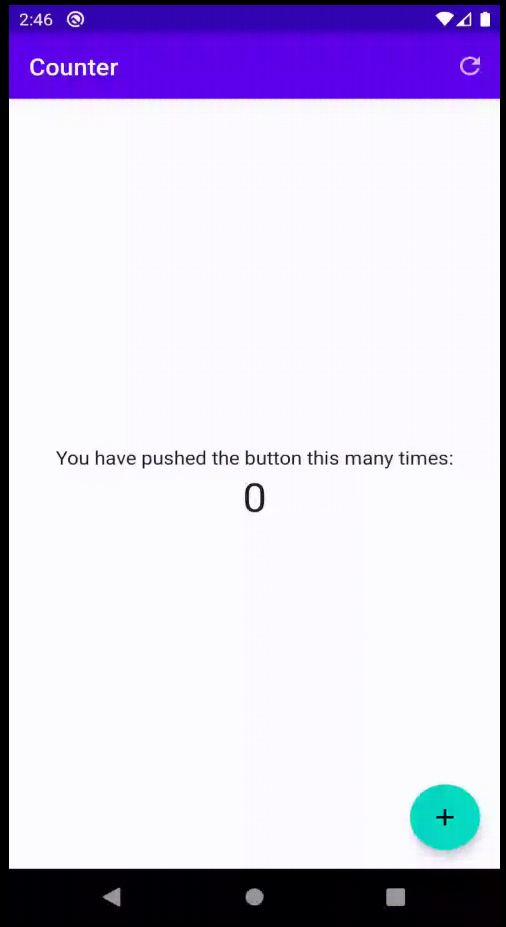
```
@Composable
fun MyHomePage(title: String, onTitleChanged: () → Unit) {
    var counter by remember { mutableStateOf( value: 0) }

    Scaffold(
        topBar = { TopAppBar(title = { Text(title) }, actions = {...}) },
        floatingActionButton = {
            FloatingActionButton(onClick = { counter++ }) {
                Icon(Icons.Default.Add, contentDescription = "Increment")
            }
        }
    ) { it: PaddingValues
        Column(
            Modifier.fillMaxSize(),
            horizontalAlignment = Alignment.CenterHorizontally,
            verticalArrangement = Arrangement.Center
        ) { this: ColumnScope
            Text( text: "You have pushed the button this many times:")
            Text(counter.toString(), style = MaterialTheme.typography.h4)
        }
    }
}
```



```
@Composable
fun MyHomePage(title: String, onTitleChanged: () → Unit) {
    var counter by rememberSaveable { mutableStateOf( value: 0) }

    Scaffold(
        topBar = { TopAppBar(title = { Text(title) }, actions = {...}) },
        floatingActionButton = {
            FloatingActionButton(onClick = { counter++ }) {
                Icon(Icons.Default.Add, contentDescription = "Increment")
            }
        }
    ) { it: PaddingValues
        Column(
            Modifier.fillMaxSize(),
            horizontalAlignment = Alignment.CenterHorizontally,
            verticalArrangement = Arrangement.Center
        ) { this: ColumnScope
            Text( text: "You have pushed the button this many times:")
            Text(counter.toString(), style = MaterialTheme.typography.h4)
        }
    }
}
```



```
@Composable
fun MyHomePage(title: String, onTitleChanged: () → Unit) {
    var counter by rememberSaveable { mutableStateOf( value: 0) }

    Scaffold(
        topBar = { TopAppBar(title = { Text(title) }, actions = {...}) },
        floatingActionButton = {
            FloatingActionButton(onClick = { counter++ }) {
                Icon(Icons.Default.Add, contentDescription = "Increment")
            }
        }
    ) { it: PaddingValues
        Column(
            Modifier.fillMaxSize(),
            horizontalAlignment = Alignment.CenterHorizontally,
            verticalArrangement = Arrangement.Center
        ) { this: ColumnScope
            Text( text: "You have pushed the button this many times:")
            Text(counter.toString(), style = MaterialTheme.typography.h4)
        }
    }
}
```



```
@Composable
fun MyHomePage(title: String, onTitleChanged: () -> Unit) {
    var counter by rememberSaveable { mutableStateOf( value: 0) }

    Scaffold(
        topBar = { TopAppBar(title = { Text(title) }, actions = {...}) },
        floatingActionButton = {
            FloatingActionButton(onClick = { counter++ }) {
                Icon(Icons.Default.Add, contentDescription = "Increment")
            }
        }
    ) {  it: PaddingValues
        Column(
            Modifier.fillMaxSize(),
            horizontalAlignment = Alignment.CenterHorizontally,
            verticalArrangement = Arrangement.Center
        ) {  this: ColumnScope
            Text( text: "You have pushed the button this many times:")
            Text(counter.toString(), style = MaterialTheme.typography.h4)
        }
    }
}
```

Modifiers

Модификаторы позволяют делать следующие вещи:

- Изменять поведение, размер, внешний вид
- Добавлять информацию, например для accessibility
- Делать элемент кликабельным
- И так далее

Порядок модификаторов

```
@Composable
fun ModifierSample1() {
    Text(
        text = "Hello World",
        modifier = Modifier
            .padding(20.dp)
            .background(Color.Green)
    )
}
```

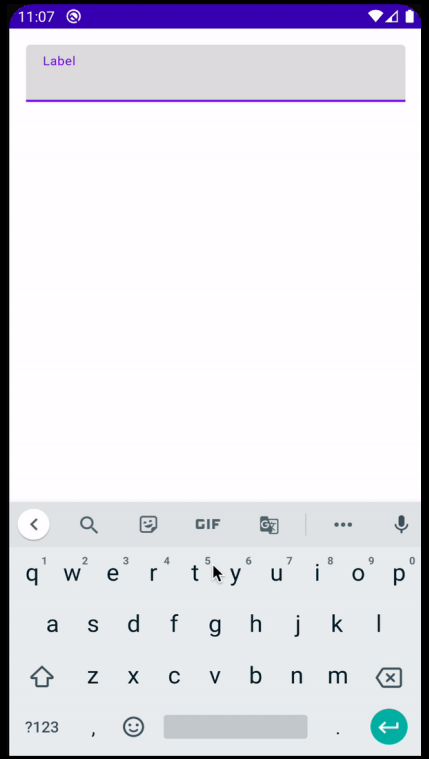

```
@Composable
fun ModifierSample2() {
    Text(
        text = "Hello World",
        modifier = Modifier
            .background(Color.Green)
            .padding(20.dp)
    )
}
```

Область видимости модификаторов

```
@LayoutScopeMarker
@Immutable
interface BoxScope {
    @Stable
    fun Modifier.align(alignment: Alignment): Modifier

    @Stable
    fun Modifier.matchParentSize(): Modifier
}
```

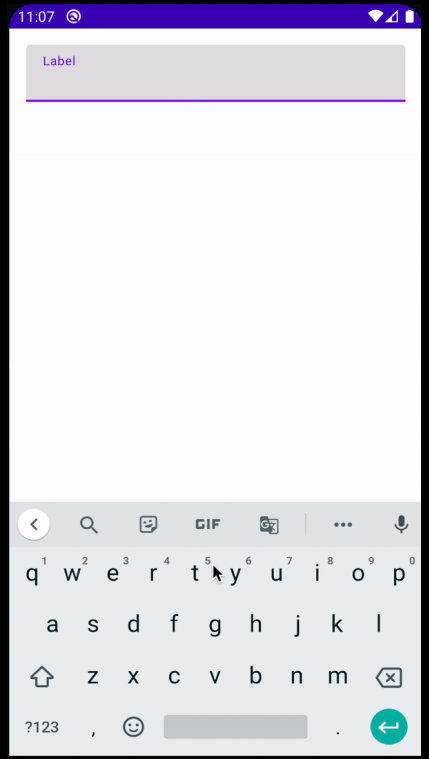
Виджеты без состояния

```
@Composable
fun TextFieldSample() {
    TextField(
        modifier = Modifier
            .fillMaxWidth()
            .padding(16.dp),
        value = "",
        onValueChange = {},
        label = { Text( text: "Label") }
    )
}
```



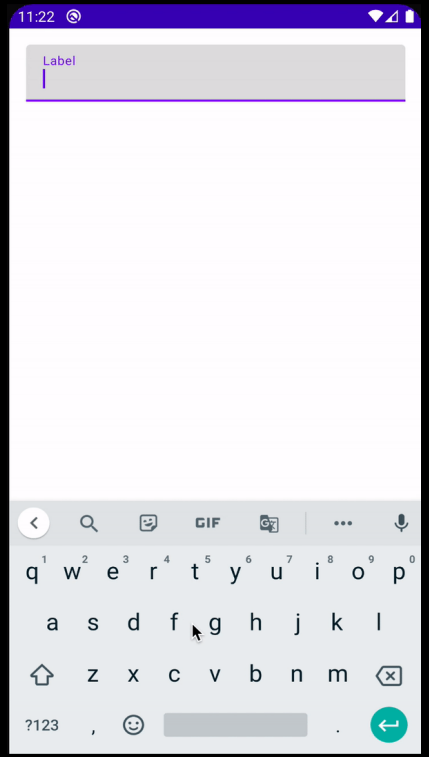
Виджеты без состояния

```
@Composable
fun TextFieldSample() {
    TextField(
        modifier = Modifier
            .fillMaxWidth()
            .padding(16.dp),
        value = "",
        onValueChange = {},
        label = { Text( text: "Label") }
    )
}
```



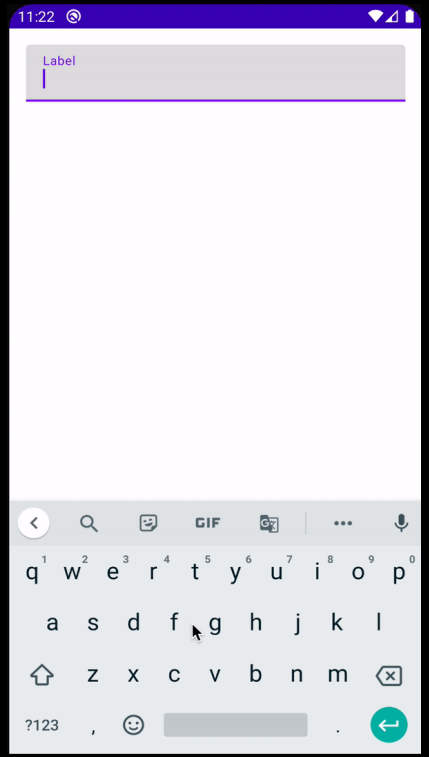
Виджеты без состояния

```
@Composable
fun TextFieldSample() {
    var value by rememberSaveable { mutableStateOf( value: "") }
    TextField(
        modifier = Modifier
            .fillMaxWidth()
            .padding(16.dp),
        value = value,
        onValueChange = { value = it },
        label = { Text( text: "Label") }
    )
}
```

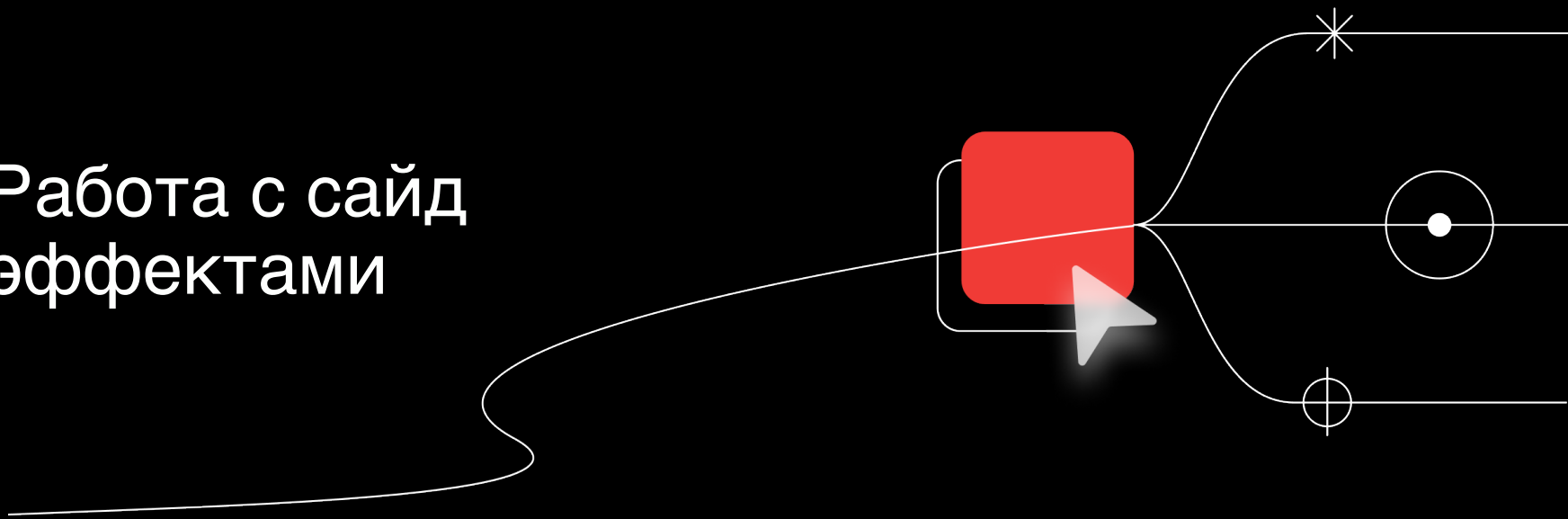


Виджеты без состояния

```
@Composable
fun TextFieldSample() {
    var value by rememberSaveable { mutableStateOf( value: "") }
    TextField(
        modifier = Modifier
            .fillMaxWidth()
            .padding(16.dp),
        value = value,
        onValueChange = { value = it },
        label = { Text( text: "Label") }
    )
}
```



Работа с сайд эффектами



Асинхронные операции

```
@Composable
fun SnackbarExample() {
    val scaffoldState = rememberScaffoldState()
    Scaffold(scaffoldState = scaffoldState) { it: PaddingValues
        Button(onClick = {
            scaffoldState.snackbarHostState.showSnackbar( message: "Hey")
        }) { this: RowScope
            Text( text: "Show message")
        }
    }
}
```

Асинхронные операции

```
@Composable
fun SnackbarExample() {
    val scaffoldState = rememberScaffoldState()
    Scaffold(scaffoldState = scaffoldState) { it: PaddingValues
        Button(onClick = {
            scaffoldState.snackbarHostState.showSnackbar( message: "Hey")
        }) { this: RowScope
            Text( text: "Show message")
        }
    }
}
```

```
suspend fun showSnackbar(
    message: String,
    actionLabel: String? = null,
    duration: SnackbarDuration = SnackbarDuration.Short
): SnackbarResult = mutex.withLock {...}
```

Асинхронные операции

```
@Composable
fun SnackbarExample() {
    val scaffoldState = rememberScaffoldState()
    val coroutineScope = rememberCoroutineScope()
    Scaffold(scaffoldState = scaffoldState) { it: PaddingValues
        Button(onClick = {
            coroutineScope.launch { this: CoroutineScope
                scaffoldState.snackbarHostState.showSnackbar( message: "Hey")
            }
        }) { this: RowScope
            Text( text: "Show message")
        }
    }
}
```

Launched Effect

```
@Composable
fun SnackbarExample(showErrorMessage: Boolean) {
    val scaffoldState = rememberScaffoldState()
    if (showErrorMessage) {
        LaunchedEffect(scaffoldState.snackbarHostState) { this: CoroutineSc
            scaffoldState.snackbarHostState.showSnackbar( message: "Hey")
        }
    }
    Scaffold(scaffoldState = scaffoldState) {...}
}
```

Launched Effect

```
@Composable
@NonRestartableComposable
@Suppress( ...names: "ArrayReturn")
@OptIn(InternalComposeApi::class)
fun LaunchedEffect(
    vararg keys: Any?,
    block: suspend CoroutineScope.() → Unit
) {
    val applyContext = currentComposer.applyCoroutineContext
    remember(*keys) { LaunchedEffectImpl(applyContext, block) }
}
```

Disposable Effect

```
@Composable
fun LifecycleListener(onStart: () → Unit, onStop: () → Unit) {
    val lifecycleObserver = remember {
        LifecycleEventObserver { _, event →
            when (event) {...}
        }
    }
    val lifecycle = LocalLifecycleOwner.current.lifecycle
    DisposableEffect(lifecycle) { this: DisposableEffectScope
        lifecycle.addObserver(lifecycleObserver)
        onDispose {
            lifecycle.removeObserver(lifecycleObserver)
        } ^DisposableEffect
    }
}
```

Disposable Effect

```
@Composable
fun LifecycleListener(onStart: () → Unit, onStop: () → Unit) {
    val lifecycleObserver = remember {
        LifecycleEventObserver { _, event →
            when (event) {...}
        }
    }
    val lifecycle = LocalLifecycleOwner.current.lifecycle
    DisposableEffect(lifecycle) {  this: DisposableEffectScope
        lifecycle.addObserver(lifecycleObserver)
        onDispose {
            lifecycle.removeObserver(lifecycleObserver)
        }  ^DisposableEffect
    }
}
```

```kotlin
@Composable
fun BackHandler(onBack: () -> Unit) {
    // Remember in Composition a back callback that calls the `onBack` lambda
    val backCallback = remember {
        object : OnBackPressedCallback( enabled: true) {
            override fun handleOnBackPressed() {
                onBack()
            }
        }
    }
    val backDispatcher = LocalOnBackPressedDispatcherOwner.current?.onBackPressedDispatcher
    // If `backDispatcher` changes, dispose and reset the effect
    DisposableEffect(backDispatcher) { this: DisposableEffectScope
        // Add callback to the backDispatcher
        backDispatcher?.addCallback(backCallback)
        // When the effect leaves the Composition, remove the callback
        onDispose {
            backCallback.remove()
        } ^DisposableEffect
    }
}
```

```kotlin
@Composable
fun BackHandler(onBack: () -> Unit) {
    // Remember in Composition a back callback that calls the `onBack` lambda
    val backCallback = remember {
        object : OnBackPressedCallback( enabled: true) {
            override fun handleOnBackPressed() {
                onBack()
            }
        }
    }
    val backDispatcher = LocalOnBackPressedDispatcherOwner.current?.onBackPressedDispatcher
    // If `backDispatcher` changes, dispose and reset the effect
    DisposableEffect(backDispatcher) {  this: DisposableEffectScope
        // Add callback to the backDispatcher
        backDispatcher?.addCallback(backCallback)
        // When the effect leaves the Composition, remove the callback
        onDispose {
            backCallback.remove()
        }  ^DisposableEffect
    }
}
```

```kotlin
@Composable
fun BackHandler(onBack: () -> Unit) {
    // Remember in Composition a back callback that calls the `onBack` lambda
    val backCallback = remember {
        object : OnBackPressedCallback( enabled: true) {
            override fun handleOnBackPressed() {
                onBack()
            }
        }
    }
    val backDispatcher = LocalOnBackPressedDispatcherOwner.current?.onBackPressedDispatcher
    // If `backDispatcher` changes, dispose and reset the effect
    DisposableEffect(backDispatcher) { this: DisposableEffectScope
        // Add callback to the backDispatcher
        backDispatcher?.addCallback(backCallback)
        // When the effect leaves the Composition, remove the callback
        onDispose {
            backCallback.remove()
        } ^DisposableEffect
    }
}
```

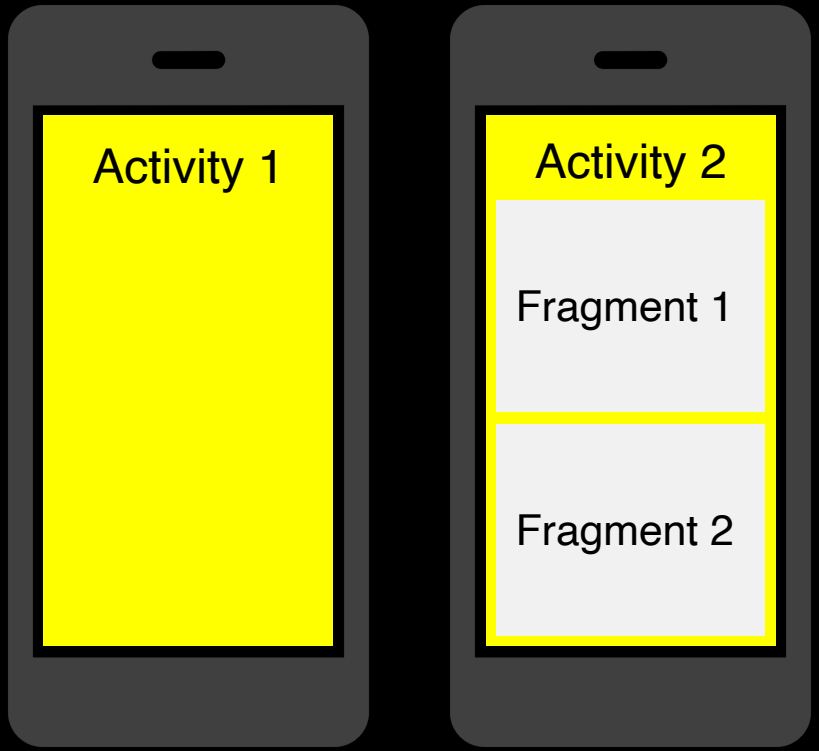
```kotlin
@Composable
fun BackHandler(onBack: () -> Unit) {
    // Remember in Composition a back callback that calls the `onBack` lambda
    val backCallback = remember {
        object : OnBackPressedCallback( enabled: true) {
            override fun handleOnBackPressed() {
                onBack()
            }
        }
    }
    val backDispatcher = LocalOnBackPressedDispatcherOwner.current?.onBackPressedDispatcher
    // If `backDispatcher` changes, dispose and reset the effect
    DisposableEffect(backDispatcher) { this: DisposableEffectScope
        // Add callback to the backDispatcher
        backDispatcher?.addCallback(backCallback)
        // When the effect leaves the Composition, remove the callback
        onDispose {
            backCallback.remove()
        } ^DisposableEffect
    }
}
```

```kotlin
@Composable
fun BackHandler(onBack: () -> Unit) {
    // Safely update the current `onBack` lambda when a new one is provided
    val currentOnBack by rememberUpdatedState(onBack)
    // Remember in Composition a back callback that calls the `onBack` lambda
    val backCallback = remember {
        object : OnBackPressedCallback( enabled: true) {
            override fun handleOnBackPressed() {
                currentOnBack()
            }
        }
    }
    val backDispatcher = LocalOnBackPressedDispatcherOwner.current?.onBackPressedDispatcher
    // If `backDispatcher` changes, dispose and reset the effect
    DisposableEffect(backDispatcher) {...}
}
```

```kotlin
@Composable
fun BackHandler(enabled: Boolean, onBack: () -> Unit) {
    // Safely update the current `onBack` lambda when a new one is provided
    val currentOnBack by rememberUpdatedState(onBack)
    // Remember in Composition a back callback that calls the `onBack` lambda
    val backCallback = remember {
        object : OnBackPressedCallback(enabled) {
            override fun handleOnBackPressed() {
                currentOnBack()
            }
        }
    }
    // On every successful composition, update the callback with the `enabled` value
    SideEffect {
        backCallback.isEnabled = enabled
    }
    val backDispatcher = LocalOnBackPressedDispatcherOwner.current?.onBackPressedDispatcher
    // If `backDispatcher` changes, dispose and reset the effect
    DisposableEffect(backDispatcher) {...}
}
```

Что изменится с приходом Jetpack Compose

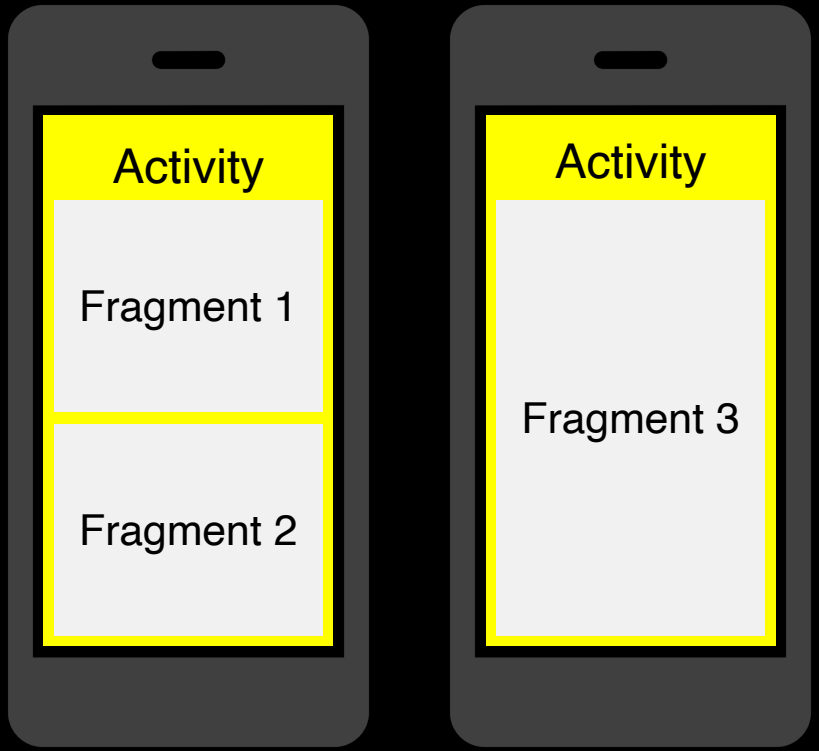
Как это было



Multiple activities

Multiple fragments

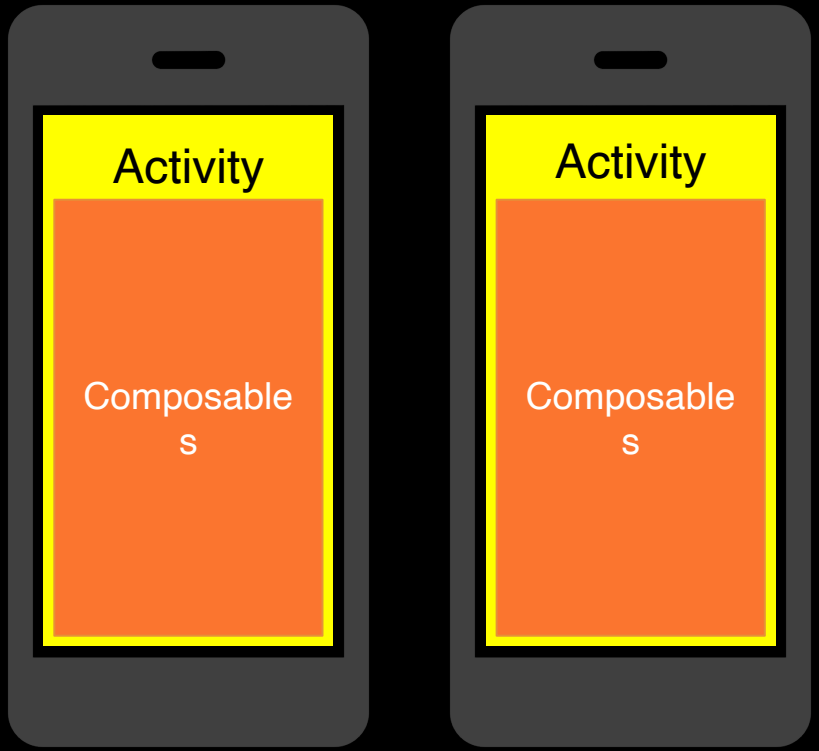
Как сейчас



Single activity

Multiple fragments

Как будет



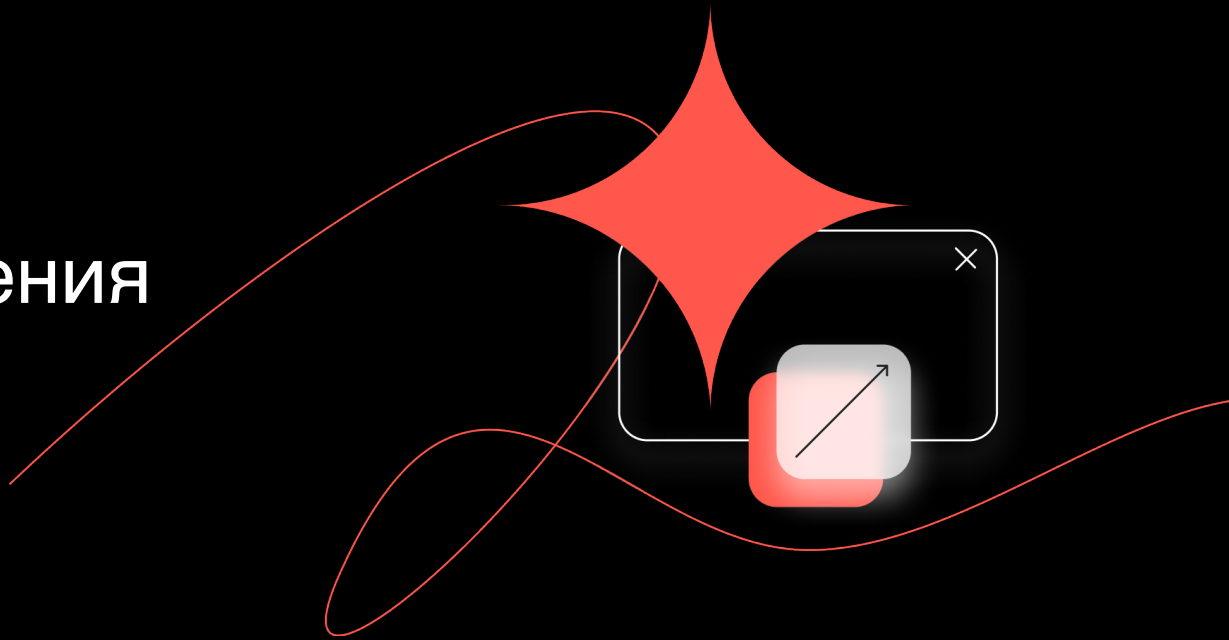
Single activity

Zero fragments

Multiple composables

План рефакторинга

Тема приложения



Material theme

```
@Composable
fun AppTheme(
    darkTheme: Boolean = isSystemInDarkTheme(),
    content: @Composable() () → Unit
) {
    val colors = if (darkTheme) DarkColorPalette else LightColorPalette

    MaterialTheme(
        colors = colors,
        typography = Typography,
        shapes = Shapes,
        content = content
    )
}
```

Material theme

```
@Composable
fun AppTheme(
    darkTheme: Boolean = isSystemInDarkTheme(),
    content: @Composable() () → Unit
) {
    val colors = if (darkTheme) DarkColorPalette else LightColorPalette

    MaterialTheme(
        colors = colors,
        typography = Typography,
        shapes = Shapes,
        content = content
    )
}
```

```
@Composable
fun MaterialThemeSample() {
    AppTheme {
        Text(
            text = "Hey",
            style = MaterialTheme.typography.body1
        )
    }
}
```

Кастомизация темы

```
data class Elevations(val card: Dp = 0.dp)

internal val LocalElevations = staticCompositionLocalOf { Elevations() }
```

Кастомизация темы

```
data class Elevations(val card: Dp = 0.dp)

internal val LocalElevations = staticCompositionLocalOf { Elevations() }

private val LightElevations = Elevations()
private val DarkElevations = Elevations(card = 4.dp)
```

Кастомизация темы

```
data class Elevations(val card: Dp = 0.dp)

internal val LocalElevations = staticCompositionLocalOf { Elevations() }

private val LightElevations = Elevations()
private val DarkElevations = Elevations(card = 4.dp)

val elevations = if (darkTheme) DarkElevations else LightElevations

CompositionLocalProvider( ...values:
    LocalElevations provides elevations
) {...}
```

Кастомизация темы

```
data class Elevations(val card: Dp = 0.dp)

internal val LocalElevations = staticCompositionLocalOf { Elevations() }

private val LightElevations = Elevations()
private val DarkElevations = Elevations(card = 4.dp)

val elevations = if (darkTheme) DarkElevations else LightElevations

CompositionLocalProvider( ...values:
    LocalElevations provides elevations
) {...}
```

```
object AppTheme {
    val elevations: Elevations
        @Composable
        get() = LocalElevations.current
}
```

Кастомные значения темы

```
val Colors.progressBackground: Color
    @Composable get() = if (isLight) SemiLightGray else DarkGray

val Typography.dialogTitle: TextStyle
    @Composable get() = TextStyle(fontSize = 16.sp)
```

Адаптер для темы

```
dependencies {
    // When using a MDC theme
    implementation "com.google.android.material:compose-theme-adapter:1.0.2"

    // When using a AppCompat theme
    implementation "com.google.accompanist:accompanist-appcompat-theme:0.16.0"
}
```

Адаптер для темы

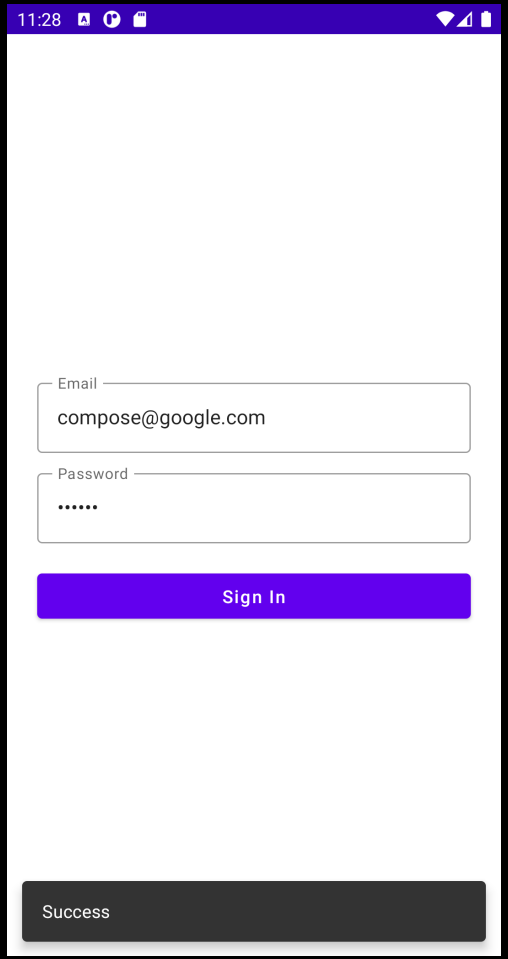
Обеспечивает быстрый старт

Один источник правды

Сложно кастомизировать для Compose

Не подходит при редизайне приложения

Переписываем верстку на Compose


11:28
Email
compose@google.com
Password
••••••
Sign In
Success

LoginContent

```
val focusManager = LocalFocusManager.current
val passwordFocusRequester = remember { FocusRequester() }

Scaffold(scaffoldState = scaffoldState) { it: PaddingValues
    Box(Modifier.fillMaxSize()) { this: BoxScope
        Column(
            Modifier.align(Alignment.Center),
            horizontalAlignment = Alignment.CenterHorizontally
        ) { this: ColumnScope
            LoginField(
                value = login,
                onValueChange = onLoginChange,
                onNext = {
                    passwordFocusRequester.requestFocus()
                }
            )
            PasswordField(
                value = password,
                onValueChange = onPasswordChange,
                onDone = {
                    focusManager.clearFocus()
                    onSignInClick()
                },
                modifier = Modifier.focusRequester(passwordFocusRequester)
            )
            SignInButton(onClick = onSignInClick, enabled = signInButtonEnabled)
        }
    }
}
```

LoginContent

```
val focusManager = LocalFocusManager.current
val passwordFocusRequester = remember { FocusRequester() }

Scaffold(scaffoldState = scaffoldState) { it: PaddingValues
    Box(Modifier.fillMaxSize()) { this: BoxScope
        Column(
            Modifier.align(Alignment.Center),
            horizontalAlignment = Alignment.CenterHorizontally
        ) { this: ColumnScope
            LoginField(
                value = login,
                onValueChange = onLoginChange,
                onNext = {
                    passwordFocusRequester.requestFocus()
                }
            )
            PasswordField(
                value = password,
                onValueChange = onPasswordChange,
                onDone = {
                    focusManager.clearFocus()
                    onSignInClick()
                },
                modifier = Modifier.focusRequester(passwordFocusRequester)
            )
            SignInButton(onClick = onSignInClick, enabled = signInButtonEnabled)
        }
    }
}
```

PasswordField

```
@Composable
fun PasswordField(
    value: String,
    onValueChange: (String) → Unit,
    onDone: () → Unit,
    modifier: Modifier = Modifier
) {
    OutlinedTextField(
        modifier = Modifier
            .fillMaxWidth()
            .padding(bottom = 24.dp, start = 24.dp, end = 24.dp)
            .then(modifier),
        value = value,
        onValueChange = onValueChange,
        label = { Text(stringResource("Password")) },
        keyboardOptions = KeyboardOptions.Default.copy(
            keyboardType = KeyboardType.Password,
            imeAction = ImeAction.Done
        ),
        visualTransformation = PasswordVisualTransformation(),
        keyboardActions = KeyboardActions(onDone = { onDone() })
    )
}
```

Local state management

```
var email by rememberSaveable { mutableStateOf( value: "") }
var password by rememberSaveable { mutableStateOf( value: "") }
val signInEnabled by remember(email, password) {
    derivedStateOf { email.isNotBlank() && password.isNotBlank() }
}
val scaffoldState = rememberScaffoldState()
val coroutineScope = rememberCoroutineScope()

LoginContent(
    scaffoldState = scaffoldState,
    login = email,
    password = password,
    signInButtonEnabled = signInEnabled,
    onLoginChange = { email = it },
    onPasswordChange = { password = it },
    onSignInClick = {
        if (signInEnabled) {
            coroutineScope.launch { this: CoroutineScope
                scaffoldState.snackbarHostState.currentSnackbarData?.dismiss()
                scaffoldState.snackbarHostState.showSnackbar( message: "Success")
            }
        }
    }
)
```

Local state management

```
var email by rememberSaveable { mutableStateOf( value: "") }
var password by rememberSaveable { mutableStateOf( value: "") }
val signInEnabled by remember(email, password) {
    derivedStateOf { email.isNotBlank() && password.isNotBlank() }
}
val scaffoldState = rememberScaffoldState()
val coroutineScope = rememberCoroutineScope()

LoginContent(
    scaffoldState = scaffoldState,
    login = email,
    password = password,
    signInButtonEnabled = signInEnabled,
    onLoginChange = { email = it },
    onPasswordChange = { password = it },
    onSignInClick = {
        if (signInEnabled) {
            coroutineScope.launch { this: CoroutineScope
                scaffoldState.snackbarHostState.currentSnackbarData?.dismiss()
                scaffoldState.snackbarHostState.showSnackbar( message: "Success")
            }
        }
    }
)
```

Local state management

```
var email by rememberSaveable { mutableStateOf( value: "") }
var password by rememberSaveable { mutableStateOf( value: "") }
val signInEnabled by remember(email, password) {
    derivedStateOf { email.isNotBlank() && password.isNotBlank() }
}
val scaffoldState = rememberScaffoldState()
val coroutineScope = rememberCoroutineScope()

LoginContent(
    scaffoldState = scaffoldState,
    login = email,
    password = password,
    signInButtonEnabled = signInEnabled,
    onLoginChange = { email = it },
    onPasswordChange = { password = it },
    onSignInClick = {
        if (signInEnabled) {
            coroutineScope.launch { this: CoroutineScope
                scaffoldState.snackbarHostState.currentSnackbarData?.dismiss()
                scaffoldState.snackbarHostState.showSnackbar( message: "Success")
            }
        }
    }
)
```

Где произойдет рекомпозиция?

При изменении текста в LoginField

1

```
@Composable
fun LoginField(
    value: String,
    onValueChange: (String) → Unit,
    onNext: () → Unit
) {...}
```

2

```
@Composable
fun PasswordField(
    value: String,
    onValueChange: (String) → Unit,
    onDone: () → Unit,
    modifier: Modifier = Modifier
) {...}
```

3

```
@Composable
fun SignInButton(
    onClick: () → Unit,
    enabled: Boolean
) {...}
```

Где произойдет рекомпозиция?

При изменении текста в LoginField

1

```
@Composable
fun LoginField(
    value: String,
    onValueChange: (String) → Unit,
    onNext: () → Unit
) {...}
```

2

```
@Composable
fun PasswordField(
    value: String,
    onValueChange: (String) → Unit,
    onDone: () → Unit,
    modifier: Modifier = Modifier
) {...}
```

3

```
@Composable
fun SignInButton(
    onClick: () → Unit,
    enabled: Boolean
) {...}
```

```
var email by rememberSaveable { mutableStateOf( value: "") }
var password by rememberSaveable { mutableStateOf( value: "") }
val signInEnabled by remember(email, password) {
    derivedStateOf { email.isNotBlank() && password.isNotBlank() }
}
val scaffoldState = rememberScaffoldState()
val coroutineScope = rememberCoroutineScope()

LoginContent(
    scaffoldState = scaffoldState,
    login = email,
    password = password,
    signInButtonEnabled = signInEnabled,
    onLoginChange = { email = it },
    onPasswordChange = { password = it },
    onSignInClick = {
        if (signInEnabled) {
            coroutineScope.launch { this: CoroutineScope
                scaffoldState.snackbarHostState.currentSnackbarData?.dismiss()
                scaffoldState.snackbarHostState.showSnackbar( message: "Success")
            }
        }
    }
)
```

```
var email by rememberSaveable { mutableStateOf( value: "") }
var password by rememberSaveable { mutableStateOf( value: "") }
val signInEnabled by remember(email, password) {
    derivedStateOf { email.isNotBlank() && password.isNotBlank() }
}
val scaffoldState = rememberScaffoldState()
val coroutineScope = rememberCoroutineScope()

val onSignInClick = remember { {
    if (signInEnabled) {
        coroutineScope.launch { this: CoroutineScope
            scaffoldState.snackbarHostState.currentSnackbarData?.dismiss()
            scaffoldState.snackbarHostState.showSnackbar( message: "Success")
        }
    }
} }

LoginContent(
    scaffoldState = scaffoldState,
    login = email,
    password = password,
    signInButtonEnabled = signInEnabled,
    onLoginChange = { email = it },
    onPasswordChange = { password = it },
    onSignInClick = onSignInClick
)
```

Где произойдет рекомпозиция?

При изменении текста в LoginField

1

```
@Composable
fun LoginField(
    value: String,
    onValueChange: (String) → Unit,
    onNext: () → Unit
) {...}
```

2

```
@Composable
fun PasswordField(
    value: String,
    onValueChange: (String) → Unit,
    onDone: () → Unit,
    modifier: Modifier = Modifier
) {...}
```

3

```
@Composable
fun SignInButton(
    onClick: () → Unit,
    enabled: Boolean
) {...}
```

```
val focusManager = LocalFocusManager.current
val passwordFocusRequester = remember { FocusRequester() }

Scaffold(scaffoldState = scaffoldState) { it: PaddingValues
    Box(Modifier.fillMaxSize()) { this: BoxScope
        Column(
            Modifier.align(Alignment.Center),
            horizontalAlignment = Alignment.CenterHorizontally
        ) { this: ColumnScope
            LoginField(
                value = login,
                onValueChange = onLoginChange,
                onNext = {
                    passwordFocusRequester.requestFocus()
                }
            )
            PasswordField(
                value = password,
                onValueChange = onPasswordChange,
                onDone = {
                    focusManager.clearFocus()
                    onSignInClick()
                },
                modifier = Modifier.focusRequester(passwordFocusRequester)
            )
            SignInButton(onClick = onSignInClick, enabled = signInButtonEnabled)
        }
    }
}
```

```
val focusManager = LocalFocusManager.current
val passwordFocusRequester = remember { FocusRequester() }

Scaffold(scaffoldState = scaffoldState) { it: PaddingValues
    Box(Modifier.fillMaxSize()) { this: BoxScope
        Column(
            Modifier.align(Alignment.Center),
            horizontalAlignment = Alignment.CenterHorizontally
        ) { this: ColumnScope
            LoginField(
                value = login,
                onValueChange = onLoginChange,
                onNext = {
                    passwordFocusRequester.requestFocus()
                }
            )
            PasswordField(
                value = password,
                onValueChange = onPasswordChange,
                onDone = {
                    focusManager.clearFocus()
                    onSignInClick()
                },
                modifier = Modifier.focusRequester(passwordFocusRequester)
            )
            SignInButton(onClick = onSignInClick, enabled = signInButtonEnabled)
        }
    }
}
```

```kotlin
@Composable
fun LoginContent(
    scaffoldState: ScaffoldState = rememberScaffoldState(),
    login: String,
    password: String,
    signInButtonEnabled: Boolean,
    onLoginChange: (String) -> Unit,
    onPasswordChange: (String) -> Unit,
    onSignInClick: () -> Unit,
) {
    val focusManager = LocalFocusManager.current
    val passwordFocusRequester = remember { FocusRequester() }
    val onDone = remember { { focusManager.clearFocus(); onSignInClick() } }
    val onNext = remember { { passwordFocusRequester.requestFocus() } }
    val focusRequesterModifier = remember { Modifier.focusRequester(passwordFocusRequester) }

    Scaffold(scaffoldState = scaffoldState) {...}
}
```

Где произойдет рекомпозиция?

При изменении текста в LoginField

1

```
@Composable
fun LoginField(
    value: String,
    onValueChange: (String) → Unit,
    onNext: () → Unit
) {...}
```

👎

2

```
@Composable
fun PasswordField(
    value: String,
    onValueChange: (String) → Unit,
    onDone: () → Unit,
    modifier: Modifier = Modifier
) {...}
```

👍

3

```
@Composable
fun SignInButton(
    onClick: () → Unit,
    enabled: Boolean
) {...}
```

👍

Обратная совместимость

Обратная совместимость

```
override fun onCreateView(
    inflater: LayoutInflater,
    container: ViewGroup?,
    savedInstanceState: Bundle?
): View {
    return ComposeView(requireContext()).apply {
        setContent {
            MaterialTheme {
                Text( text: "Hello Compose!")
            }
        }
    }
}
```

```
<androidx.compose.ui.platform.ComposeView
    android:id="@+id/composeView"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

Обратная совместимость

```
override fun onCreateView(
    inflater: LayoutInflater,
    container: ViewGroup?,
    savedInstanceState: Bundle?
): View {
    return ComposeView(requireContext()).apply {
        setContent {
            MaterialTheme {
                Text( text: "Hello Compose!")
            }
        }
    }
}
```

```
@Composable
fun AndroidProgressView() {
    AndroidView(factory = { context →
        ProgressBar(context)
    })
}
```

```
<androidx.compose.ui.platform.ComposeView
    android:id="@+id/composeView"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

Compose custom view

```
class CustomTextFieldView @JvmOverloads constructor(
    context: Context,
    attrs: AttributeSet? = null,
    defStyle: Int = 0
) : AbstractComposeView(context, attrs, defStyle) {
    var text by mutableStateOf( value: "")

    @Composable
    override fun Content() {
        TextField(text, onValueChange = { text = it })
    }
}
```

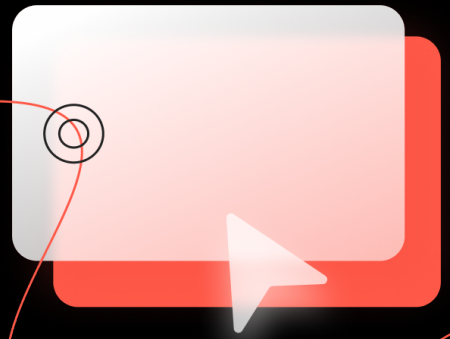
Composition strategy

```
override fun onCreateView(
    inflater: LayoutInflater,
    container: ViewGroup?,
    savedInstanceState: Bundle?
): View {
    return ComposeView(requireContext()).apply { this: ComposeView
        setViewCompositionStrategy(ViewCompositionStrategy.DisposeOnViewTreeLifecycleDestroyed)
        setContent {
            MaterialTheme {
                Text( text: "Hello Compose!")
            }
        }
    }
}
```

Compose в RecyclerView

```
class ComposeAdapter : RecyclerView.Adapter<ComposeAdapter.ComposeViewHolder>() {
    override fun onCreateViewHolder(
        parent: ViewGroup,
        viewType: Int,
    ): ComposeViewHolder {
        return ComposeViewHolder(ComposeView(parent.context))
    }

    override fun onViewRecycled(holder: ComposeViewHolder) {
        holder.composeView.disposeComposition()
    }

    class ComposeViewHolder(
        val composeView: ComposeView
    ) : RecyclerView.ViewHolder(composeView) {

        init {
            composeView.setViewCompositionStrategy(
                ViewCompositionStrategy.DisposeOnViewTreeLifecycleDestroyed
            )
        }
```

Презентационные паттерны и Compose



MVP

Model View Presenter

MVP Presenter

```
@InjectViewState
class LoginPresenter : MvpPresenter<LoginView>() {
    private var login: String = ""
    private var password: String = ""

    fun setLogin(value: String) {...}

    fun setPassword(value: String) {...}

    fun authorize() {
        if (validateFields().not()) return

        viewState.showMessage( value: "Success")
    }

    private fun checkAuthEnabled() {
        viewState.setAuthEnabled(validateFields())
    }

    private fun validateFields() = login.isNotBlank() && password.isNotBlank()
}
```

MVP
ViewAdapter

```
interface LoginViewAdapter : LoginView {
    val state: MutableState<LoginState>

    override fun setAuthEnabled(value: Boolean) {
        state.value = state.value.copy(isAuthEnabled = value)
    }

    override fun showMessage(value: String) {
        state.value = state.value.copy(message = value)
    }

    fun hideMessage() {
        state.value = state.value.copy(message = null)
    }
}
```

```
data class LoginState(
    val isAuthEnabled: Boolean = false,
    val message: String? = null
)
```

MVP Compose

```
var login by rememberSaveable { mutableStateOf( value: "") }
var password by rememberSaveable { mutableStateOf( value: "") }
val scaffoldState = rememberScaffoldState()
val onLoginChange: (String) → Unit = remember { { login = it; presenter.setLogin(it) } }
val onPasswordChange: (String) → Unit = remember { { password = it; presenter.setPassword(it) } }

state.value.message?.let { it: String
    LaunchedEffect(scaffoldState.snackbarHostState) { this: CoroutineScope
        scaffoldState.snackbarHostState.showSnackbar(it)
        hideMessage()
    }
}

LoginContent(
    scaffoldState = scaffoldState,
    login = login,
    password = password,
    signInButtonEnabled = state.value.isAuthEnabled,
    onLoginChange = onLoginChange,
    onPasswordChange = onPasswordChange,
    onSignInClick = presenter::authorize
)
```

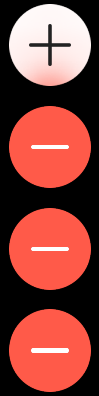
MVP Compose

```
var login by rememberSaveable { mutableStateOf( value: "") }
var password by rememberSaveable { mutableStateOf( value: "") }
val scaffoldState = rememberScaffoldState()
val onLoginChange: (String) → Unit = remember { { login = it; presenter.setLogin(it) } }
val onPasswordChange: (String) → Unit = remember { { password = it; presenter.setPassword(it) } }

state.value.message?.let { it: String
    LaunchedEffect(scaffoldState.snackbarHostState) { this: CoroutineScope
        scaffoldState.snackbarHostState.showSnackbar(it)
        hideMessage()
    }
}

LoginContent(
    scaffoldState = scaffoldState,
    login = login,
    password = password,
    signInButtonEnabled = state.value.isAuthEnabled,
    onLoginChange = onLoginChange,
    onPasswordChange = onPasswordChange,
    onSignInClick = presenter::authorize
)
```

MVP Compose

```
var login by rememberSaveable { mutableStateOf( value: "") }
var password by rememberSaveable { mutableStateOf( value: "") }
val scaffoldState = rememberScaffoldState()
val onLoginChange: (String) → Unit = remember { { login = it; presenter.setLogin(it) } }
val onPasswordChange: (String) → Unit = remember { { password = it; presenter.setPassword(it) } }

state.value.message?.let { it: String
    LaunchedEffect(scaffoldState.snackbarHostState) { this: CoroutineScope
        scaffoldState.snackbarHostState.showSnackbar(it)
        hideMessage()
    }
}

LoginContent(
    scaffoldState = scaffoldState,
    login = login,
    password = password,
    signInButtonEnabled = state.value.isAuthEnabled,
    onLoginChange = onLoginChange,
    onPasswordChange = onPasswordChange,
    onSignInClick = presenter::authorize
)
```

MVP + Jetpack Compose



+ Работает с адаптером для View
- Следует императивному стилю
- Стейт размазан по экрану
- Подвержен ошибкам при написании адаптера

MVVM

Model View ViewModel

MVVM ViewModel

```
class LoginViewModel : ViewModel() {
    private val _login = MutableLiveData<String>()
    val login: LiveData<String> get() = _login

    private val _password = MutableLiveData<String>()
    val password: LiveData<String> get() = _password

    val message = SingleLiveData<String>()

    val isAuthEnabled = MediatorLiveData<Boolean>().apply {
        addSource(login) { checkAuthEnabled() }
        addSource(password) { checkAuthEnabled() }
    }

    fun setLogin(value: String) {...}

    fun setPassword(value: String) {...}

    fun authorize() {...}

    private fun checkAuthEnabled() {...}
```

MVVM Compose

```
val login by viewModel.login.observeAsState( initial: "")
val password by viewModel.password.observeAsState( initial: "")
val isAuthEnabled by viewModel.isAuthEnabled.observeAsState( initial: false)
val scaffoldState = rememberScaffoldState()
val scope = rememberCoroutineScope()

viewModel.message.observeEvent { it: String!
    scope.launch { this: CoroutineScope
        scaffoldState.snackbarHostState.currentSnackbarData?.dismiss()
        scaffoldState.snackbarHostState.showSnackbar(it)
    }
}

LoginContent(
    scaffoldState = scaffoldState,
    login = login,
    password = password,
    signInButtonEnabled = isAuthEnabled,
    onLoginChange = viewModel::setLogin,
    onPasswordChange = viewModel::setPassword,
    onSignInClick = viewModel::authorize
)
```

State converters

```
@Composable
fun <R, T : R> Observable<T>.subscribeAsState(initial: R): State<R>

@Composable
fun <R, T : R> LiveData<T>.observeAsState(initial: R): State<R>

@Composable
fun <T> StateFlow<T>.collectAsState(
    context: CoroutineContext = EmptyCoroutineContext
): State<T>
```

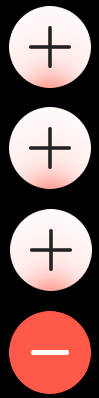
MVVM Compose

```
val login by viewModel.login.observeAsState( initial: "")
val password by viewModel.password.observeAsState( initial: "")
val isAuthEnabled by viewModel.isAuthEnabled.observeAsState( initial: false)
val scaffoldState = rememberScaffoldState()
val scope = rememberCoroutineScope()

viewModel.message.observeEvent { it: String!
    scope.launch { this: CoroutineScope
        scaffoldState.snackbarHostState.currentSnackbarData?.dismiss()
        scaffoldState.snackbarHostState.showSnackbar(it)
    }
}

LoginContent(
    scaffoldState = scaffoldState,
    login = login,
    password = password,
    signInButtonEnabled = isAuthEnabled,
    onLoginChange = viewModel::setLogin,
    onPasswordChange = viewModel::setPassword,
    onSignInClick = viewModel::authorize
)
```

SingleLiveData adapter

```
@Composable
fun <R, T : R> LiveData<T>.observeEvent(callback: (T) → Unit) {
    val lifecycleOwner = LocalLifecycleOwner.current

    DisposableEffect( key1: this, lifecycleOwner) {  this: Disposable
        val observer = Observer<T> { callback(it) }
        observe(lifecycleOwner, observer)
        onDispose { removeObserver(observer) }  ^DisposableEffect
    }
}
```

MVVM + Jetpack Compose



- \+ Не требует сильных изменений
- \+ Можно работать напрямую с LiveData
- \+ Для single-live операций можно написать свой экстеншн
- – Не удобно работать с большим количеством полей

MVVM + DataBinding

Model View ViewModel

DataBinding ViewModel

```
class LoginViewModel : ViewModel() {
    val login = ObservableField<String>()

    val password = ObservableField<String>()

    val message = SingleLiveData<String>()

    val isAuthEnabled = ObservableField<Boolean>(login, password).apply { this: ObservableField
        this.addOnPropertyChangedCallback(object : Observable.OnPropertyChangedCallback() {
            override fun onPropertyChanged(sender: Observable?, propertyId: Int) {
                checkAuthEnabled()
            }
        })
    }

    fun authorize() {...}

    private fun checkAuthEnabled() {...}

    private fun validateFields(): Boolean {...}
}
```

DataBinding View

```
<com.google.android.material.textfield.TextInputEditText
    android:id="@+id/etEmail"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:imeOptions="actionNext"
    android:inputType="textEmailAddress"
    android:nextFocusDown="@id/etPassword"
    android:text="@={ vm.login }" />

<com.google.android.material.button.MaterialButton
    android:id="@+id/btnSignIn"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginHorizontal="32dp"
    android:layout_marginTop="24dp"
    android:enabled="@{ vm.isAuthEnabled }"
    android:onClick="@{ () -> vm.authorize() }"
    android:text="@string/sign_in_btn" />
```

DataBinding ViewModel

```
class LoginComposeViewModel : ViewModel() {
    private val _login = MutableStateFlow( value: "")
    val login = _login.asStateFlow()

    private val _password = MutableStateFlow( value: "")
    val password = _password.asStateFlow()

    private val _message = MutableSharedFlow<String>()
    val message = _message.asSharedFlow()

    val isAuthEnabled: StateFlow<Boolean> = login.combine(password) { _, _ →
        validateFields()
    }.stateIn(viewModelScope, SharingStarted.Eagerly,  initialValue: false)

    fun setLogin(value: String) {...}

    fun setPassword(value: String) {...}

    fun authorize() {...}

    private fun validateFields(): Boolean {...}
}
```

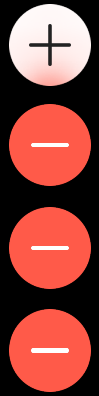
DataBinding Compose

```
val login by viewModel.login.collectAsState()
val password by viewModel.password.collectAsState()
val isAuthEnabled by viewModel.isAuthEnabled.collectAsState( initial: false)
val scaffoldState = rememberScaffoldState()

LaunchedEffect(scaffoldState.snackbarHostState) { this: CoroutineScope
    viewModel.message.onEach { it: String
        scaffoldState.snackbarHostState.currentSnackbarData?.dismiss()
        scaffoldState.snackbarHostState.showSnackbar(it)
    }.launchIn( scope: this)
}

LoginContent(
    scaffoldState = scaffoldState,
    login = login,
    password = password,
    signInButtonEnabled = isAuthEnabled,
    onLoginChange = viewModel::setLogin,
    onPasswordChange = viewModel::setPassword,
    onSignInClick = viewModel::authorize
)
```

DataBinding Compose

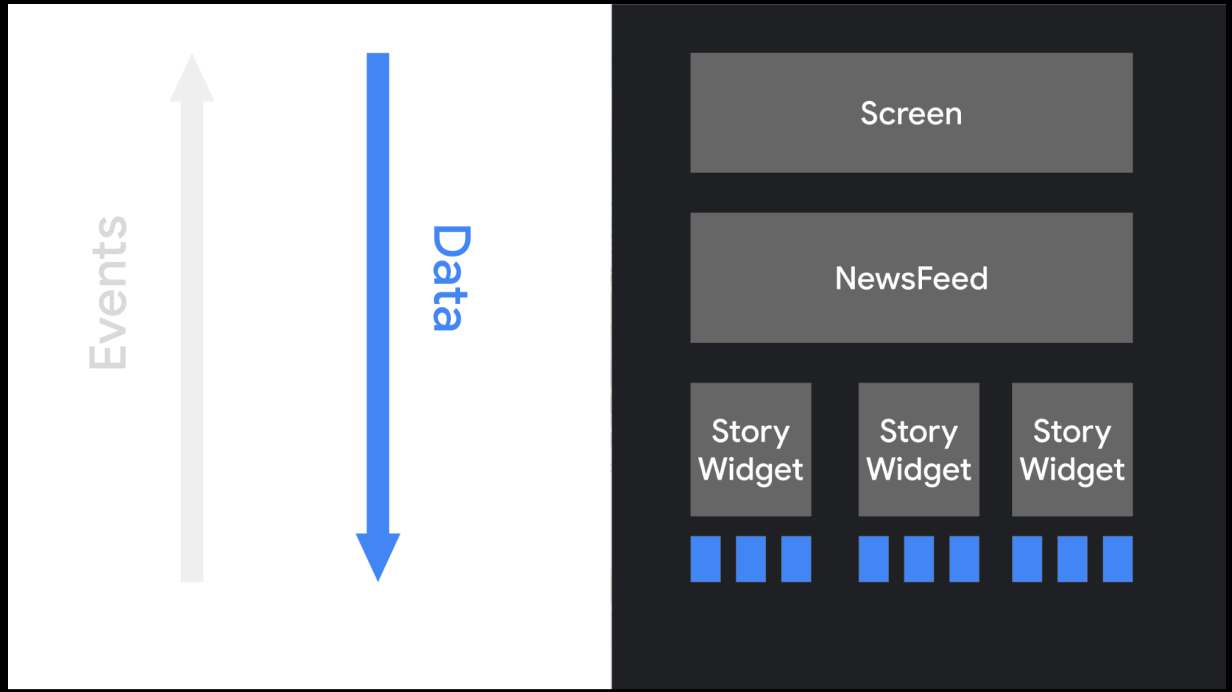
```
val login by viewModel.login.collectAsState()
val password by viewModel.password.collectAsState()
val isAuthEnabled by viewModel.isAuthEnabled.collectAsState( initial: false)
val scaffoldState = rememberScaffoldState()

LaunchedEffect(scaffoldState.snackbarHostState) { this: CoroutineScope
    viewModel.message.onEach { it: String
        scaffoldState.snackbarHostState.currentSnackbarData?.dismiss()
        scaffoldState.snackbarHostState.showSnackbar(it)
    }.launchIn( scope: this)
}

LoginContent(
    scaffoldState = scaffoldState,
    login = login,
    password = password,
    signInButtonEnabled = isAuthEnabled,
    onLoginChange = viewModel::setLogin,
    onPasswordChange = viewModel::setPassword,
    onSignInClick = viewModel::authorize
)
```

MVVM(DataBinding) + Jetpack Compose

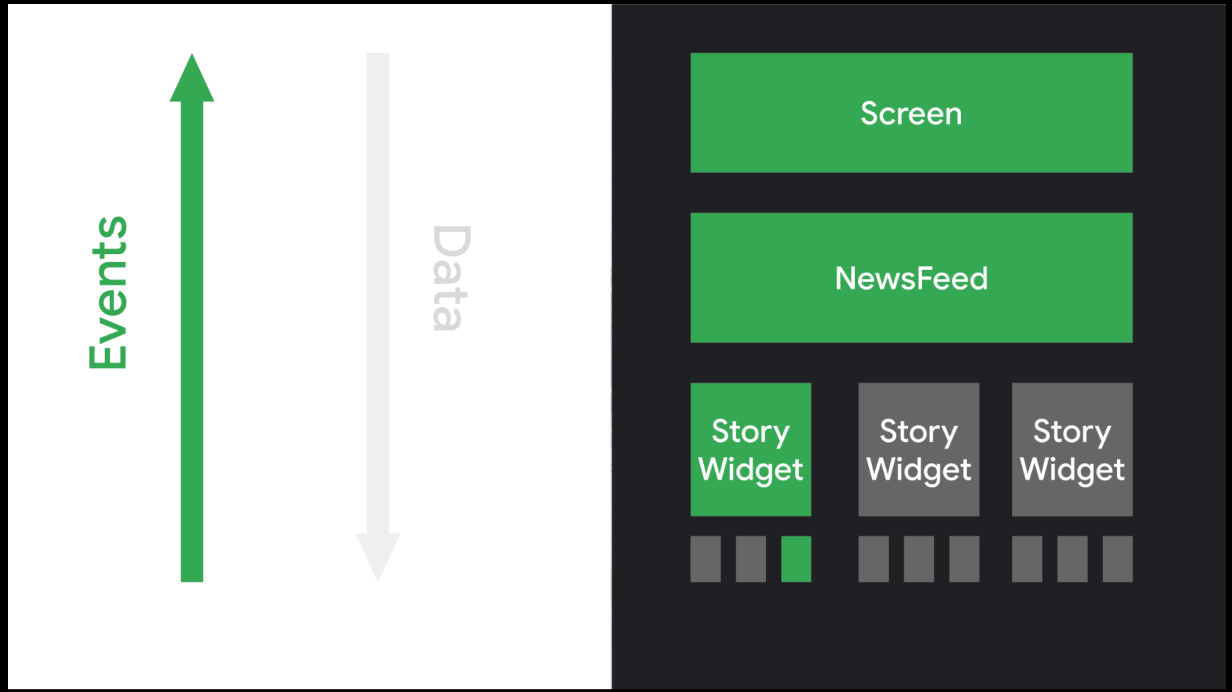


+ Легко мигрировать на StateFlow/SharedFlow
- Нет конвертера для ObservableField
- Нет поддержки двухстороннего биндинга
- Не удобно работать с большим количеством полей

Unidirectional Data Flow



Unidirectional Data Flow



MVI

Model View Intent

MVI Contract

```
class LoginContract {
    data class State(
        val email: String = "",
        val password: String = "",
    ) : UiState {
        val isSignInEnabled: Boolean
            get() = email.isNotBlank() && password.isNotBlank()
    }

    sealed class Event : UiEvent {
        data class EmailChanged(val email: String) : Event()
        data class PasswordChanged(val password: String) : Event()
        object SignInClicked : Event()
    }

    sealed class SideEffect : UiSideEffect {
        data class ShowMessage(val message: String) : SideEffect()
    }
}
```

MVI Store

```
class LoginStore : BaseStore<Event, State, SideEffect>() {
    override fun createInitialState(): State = State()

    override fun handleEvent(event: Event) {
        when (event) {
            is Event.EmailChanged -> {
                setState { copy(email = event.email) }
            }
            is Event.PasswordChanged -> {
                setState { copy(password = event.password) }
            }
            Event.SignInClicked -> authorize()
        }
    }

    private fun authorize() {
        if (currentState.isSignInEnabled.not()) return

        setSideEffect { SideEffect.ShowMessage("Success") }
    }
}
```

MVI Store

```
class LoginStore : BaseStore<Event, State, SideEffect>() {
    override fun createInitialState(): State = State()

    override fun handleEvent(event: Event) {
        when (event) {
            is Event.EmailChanged -> {
                setState { copy(email = event.email) }
            }
            is Event.PasswordChanged -> {
                setState { copy(password = event.password) }
            }
            Event.SignInClicked -> authorize()
        }
    }

    private fun authorize() {
        if (currentState.isSignInEnabled.not()) return

        setSideEffect { SideEffect.ShowMessage("Success") }
    }
}
```

MVI Store

```
class LoginStore : BaseStore<Event, State, SideEffect>() {
    override fun createInitialState(): State = State()

    override fun handleEvent(event: Event) {
        when (event) {
            is Event.EmailChanged -> {
                setState { copy(email = event.email) }
            }
            is Event.PasswordChanged -> {
                setState { copy(password = event.password) }
            }
            Event.SignInClicked -> authorize()
        }
    }

    private fun authorize() {
        if (currentState.isSignInEnabled.not()) return

        setSideEffect { SideEffect.ShowMessage("Success") }
    }
}
```

```kotlin
class LoginStore : BaseStore<Event, State, SideEffect>() {
    override fun createInitialState(): State = State()

    override fun handleEvent(event: Event) {
        when (event) {
            is Event.EmailChanged -> {
                setState { copy(email = event.email) }
            }
            is Event.PasswordChanged -> {
                setState { copy(password = event.password) }
            }
            Event.SignInClicked -> authorize()
        }
    }

    private fun authorize() {
        if (currentState.isSignInEnabled.not()) return

        setSideEffect { SideEffect.ShowMessage("Success") }
    }
}
```

MVI Store

BaseStore < 60 строк

MVI Compose

```
val state by store.uiState.collectAsState()
val scaffoldState = rememberScaffoldState()

LaunchedEffect(scaffoldState.snackbarHostState) { this: CoroutineScope
    store.sideEffect.onEach { sideEffect →
        when (sideEffect) {
            is SideEffect.ShowMessage → {
                scaffoldState.snackbarHostState.currentSnackbarData?.dismiss()
                scaffoldState.snackbarHostState.showSnackbar(sideEffect.message)
            }
        }
    }.launchIn( scope: this)
}

LoginContent(
    scaffoldState = scaffoldState,
    login = state.email,
    password = state.password,
    signInButtonEnabled = state.isSignInEnabled,
    onLoginChange = store::onLoginChange,
    onPasswordChange = store::onPasswordChange,
    onSignInClick = store::onSignInClick
)
```

MVI Compose

```
val state by store.uiState.collectAsState()
val scaffoldState = rememberScaffoldState()

LaunchedEffect(scaffoldState.snackbarHostState) { this: CoroutineScope
    store.sideEffect.onEach { sideEffect →
        when (sideEffect) {
            is SideEffect.ShowMessage → {
                scaffoldState.snackbarHostState.currentSnackbarData?.dismiss()
                scaffoldState.snackbarHostState.showSnackbar(sideEffect.message)
            }
        }
    }.launchIn( scope: this)
}

LoginContent(
    scaffoldState = scaffoldState,
    login = state.email,
    password = state.password,
    signInButtonEnabled = state.isSignInEnabled,
    onLoginChange = store::onLoginChange,
    onPasswordChange = store::onPasswordChange,
    onSignInClick = store::onSignInClick
)
```

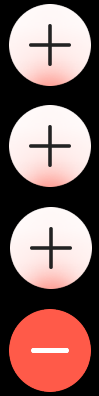
MVI Compose

```
val state by store.uiState.collectAsState()
val scaffoldState = rememberScaffoldState()

LaunchedEffect(scaffoldState.snackbarHostState) { this: CoroutineScope
    store.sideEffect.onEach { sideEffect →
        when (sideEffect) {
            is SideEffect.ShowMessage → {
                scaffoldState.snackbarHostState.currentSnackbarData?.dismiss()
                scaffoldState.snackbarHostState.showSnackbar(sideEffect.message)
            }
        }
    }.launchIn( scope: this)
}
```

```
LoginContent(
    scaffoldState = scaffoldState,
    login = state.email,
    password = state.password,
    signInButtonEnabled = state.isSignInEnabled,
    onLoginChange = store::onLoginChange,
    onPasswordChange = store::onPasswordChange,
    onSignInClick = store::onSignInClick
)
```

```
fun LoginStore.onLoginChange(value: String) {
    setEvent(LoginContract.Event.EmailChanged(value))
}
fun LoginStore.onPasswordChange(value: String) {
    setEvent(LoginContract.Event.PasswordChanged(value))
}
fun LoginStore.onSignInClick() {
    setEvent(LoginContract.Event.SignInClicked)
}
```

MVI + Jetpack Compose



- \+ Идеально подходит для Compose
- \+ Однонаправленный поток данных и единый стейт
- \+ Легче тестируется
- – Может работать неэффективно для Android Views

Сравнение MVI библиотек

	Ease of use	DI support	Side effect handling	Coroutine support	Kotlin multiplatform support	Testing support	Additional features	**Overall score**
Mavericks	3/5	2/5	❌	3/5	❌	3/5	4/5	**3/5**
MVIKotlin	2/5	3/5	✅	5/5	✅	2/5	5/5	**3/5**
Orbit	5/5	5/5	✅	5/5	✅	4/5	3/5	**4/5**
Redux-Kotlin	3/5	4/5	❌	2/5	✅	❌	❌	**2.5/5**
Roxie	3/5	4/5	❌	❌	❌	❌	❌	**2/5**
Uniflow	4/5	5/5	✅	3/5	❌	4/5	4/5	**3.5/5**

https://appmattus.medium.com/top-android-mvi-libraries-in-2021-de1afe890f27

Избавляемся от фрагментов

Навигация в Jetpack Compose

Навигация в Jetpack Compose

1. Пишем свое решение
   - Управление бэкстеком
   - Обновление текущего экрана
   - Обработка кнопки назад
   - Анимации
   - И так далее

Навигация в Jetpack Compose

1. Пишем свое решение
   - Управление бэкстеком
   - Обновление текущего экрана
   - Обработка кнопки назад
   - Анимации
   - И так далее
2. Используем Jetpack Navigation

Навигация в Jetpack Compose

1. Пишем свое решение
   - Управление бэкстеком
   - Обновление текущего экрана
   - Обработка кнопки назад
   - Анимации
   - И так далее
2. Используем Jetpack Navigation
3. Или другую библиотеку

Jetpack Navigation

```
@Composable
fun NavGraph(startDestination: String = MainDestinations.WELCOME_ROUTE) {
    val navController = rememberNavController()
    NavHost(
        navController = navController,
        startDestination = startDestination
    ) { this: NavGraphBuilder
        composable(MainDestinations.LOGIN_ROUTE) { it: NavBackStackEntry
            LogInScreen {
                navController.navigate(MainDestinations.HOME_ROUTE)
            }
        }
        composable(MainDestinations.HOME_ROUTE) { it: NavBackStackEntry
            HomeScreen()
        }
    }
}
```

Dialogs

```
AlertDialog(
    title = {...},
    text = {...},
    buttons = {
        Row(
            Modifier
                .fillMaxWidth()
                .padding(4.dp),
            horizontalArrangement = Arrangement.End
        ) { this: RowScope
            TextButton(
                onClick = onDismiss,
                modifier = Modifier.padding(4.dp)
            ) {...}
            TextButton(
                onClick = onPositiveAction,
                modifier = Modifier.padding(4.dp)
            ) {...}
        }
    },
    onDismissRequest = onDismiss
)
```

Dialogs

```
AlertDialog(
    title = {...},
    text = {...},
    buttons = {
        Row(
            Modifier
                .fillMaxWidth()
                .padding(4.dp),
            horizontalArrangement = Arrangement.End
        ) { this: RowScope
            TextButton(
                onClick = onDismiss,
                modifier = Modifier.padding(4.dp)
            ) {...}
            TextButton(
                onClick = onPositiveAction,
                modifier = Modifier.padding(4.dp)
            ) {...}
        }
    },
    onDismissRequest = onDismiss
)
```

```
@Composable
fun MyDialog() {
    val openDialog = remember { mutableStateOf( value: true) }

    if (openDialog.value) {
        Dialog(onDismissRequest = { openDialog.value = false }) {
            Box(
                Modifier
                    .size(200.dp, 200.dp)
                    .background(Color.White))
        }
    }
}
```

Bottom Sheet

```
@ExperimentalMaterialApi
@Composable
fun ScaffoldContent(
    sheetState: ModalBottomSheetState =
        rememberModalBottomSheetState(initialValue = ModalBottomSheetValue.Hidden)
) {
    ModalBottomSheetLayout(
        modifier = Modifier.fillMaxSize(),
        sheetState = sheetState,
        sheetContent = { this: ColumnScope
            BottomSheetContent()
        }
    ) {
        Column(modifier = Modifier.fillMaxSize()) {...}
    }
}
```

Bottom Sheet Navigation

```
@ExperimentalMaterialNavigationApi
@ExperimentalMaterialApi
@Composable
fun MyApp() {
    val navController = rememberNavController()
    val bottomSheetNavigator = rememberBottomSheetNavigator()
    navController.navigatorProvider += bottomSheetNavigator
    ModalBottomSheetLayout(bottomSheetNavigator) {
        NavHost(navController, startDestination: "home") { this: NavGraphBuilder
            composable(route = "home") { it: NavBackStackEntry
                Button(onClick = { navController.navigate( route: "sheet") }) {
                    Text( text: "Navigate")
                }
            }
            bottomSheet(route = "sheet") { this: ColumnScope
                Box(Modifier.fillMaxWidth().height(300.dp)) { this: BoxScope
                    Text( text: "Bottom sheet")
                }
            }
        }
    }
}
```

Accompanist navigation material library

ViewModel

```
NavHost(navController, startDestination: "home") { this: NavGraphBuilder
    composable(route = "home") { it: NavBackStackEntry
        Button(onClick = { navController.navigate( route: "sheet") }) {
            Text( text: "Navigate")
        }
    }
    bottomSheet(route = "sheet") { this: ColumnScope
        val viewModel = viewModel<MainViewModel>()
        BottomSheetContent(viewModel)
    }
}
```

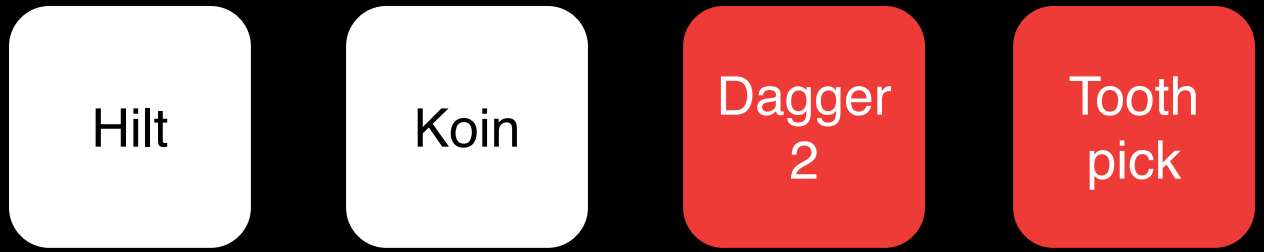
Accompanist navigation material library

Accompanist (must have библиотеки)

- Insets
- System UI Controller
- Permissions
- Placeholders
- Pull-to-refresh
- Material navigation

https://google.github.io/accompanist/

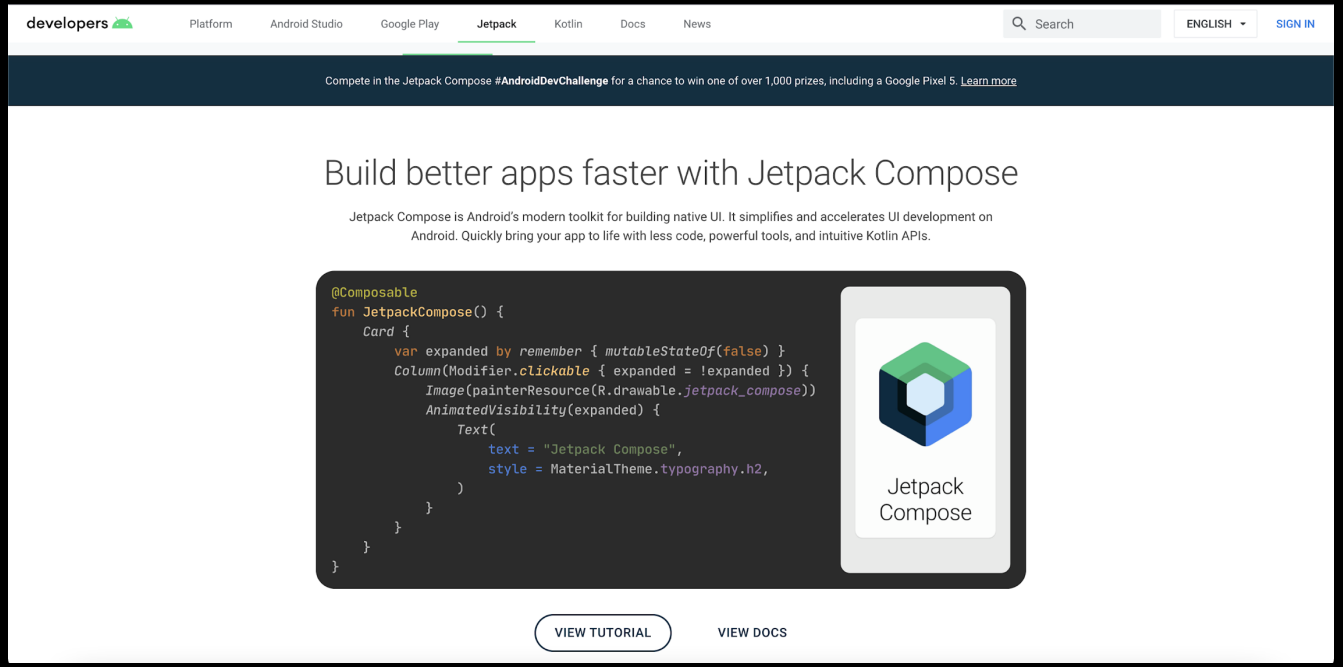
Dependency Injection в Compose



UI тестирование

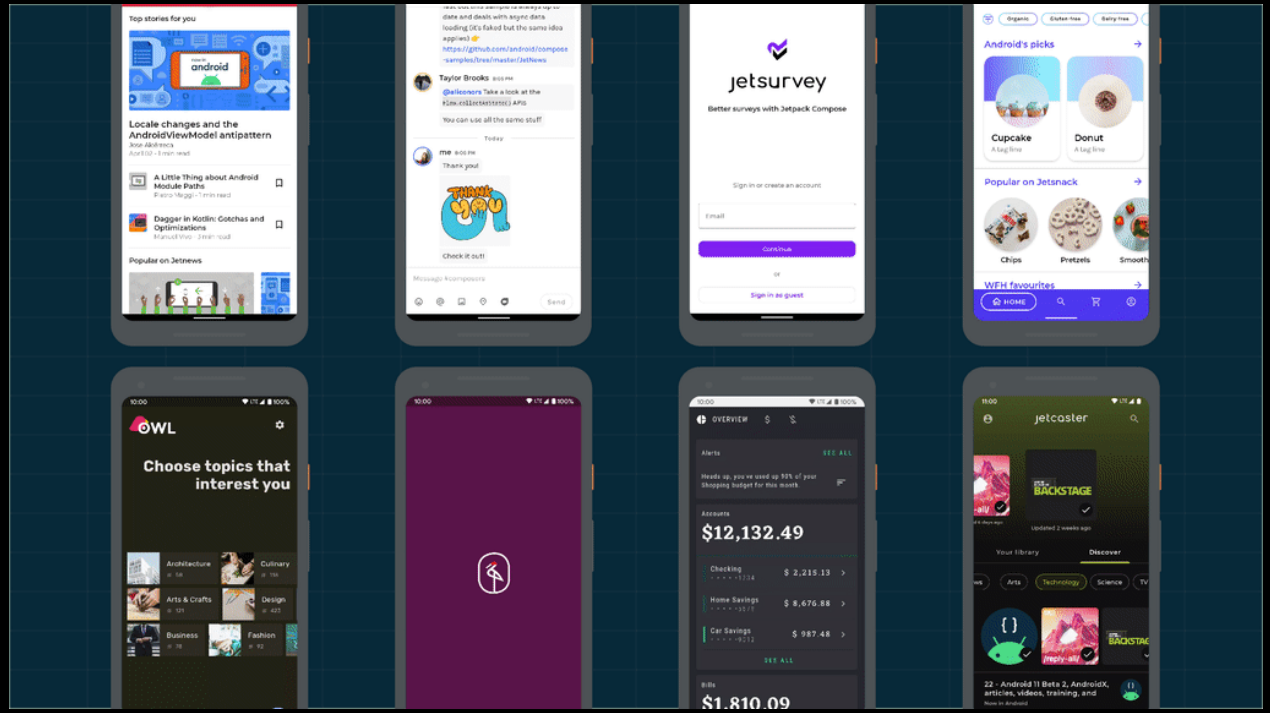
```
@get:Rule
val composeTestRule = createAndroidComposeRule<LoginComposeActivity>()

@Test
fun snackbarDisplayedTest() {
    composeTestRule.setContent {
        composeTestRule.activity.LoginScreen()
    }

    composeTestRule.onNodeWithText( text: "Email").performTextInput( text: "a@b.ru")
    composeTestRule.onNodeWithText( text: "Password").performTextInput( text: "123")
    composeTestRule.onNodeWithText( text: "Sign In").performClick()
    composeTestRule.onNodeWithText( text: "Success").assertIsDisplayed()
}
```

Как учить Jetpack Compose?



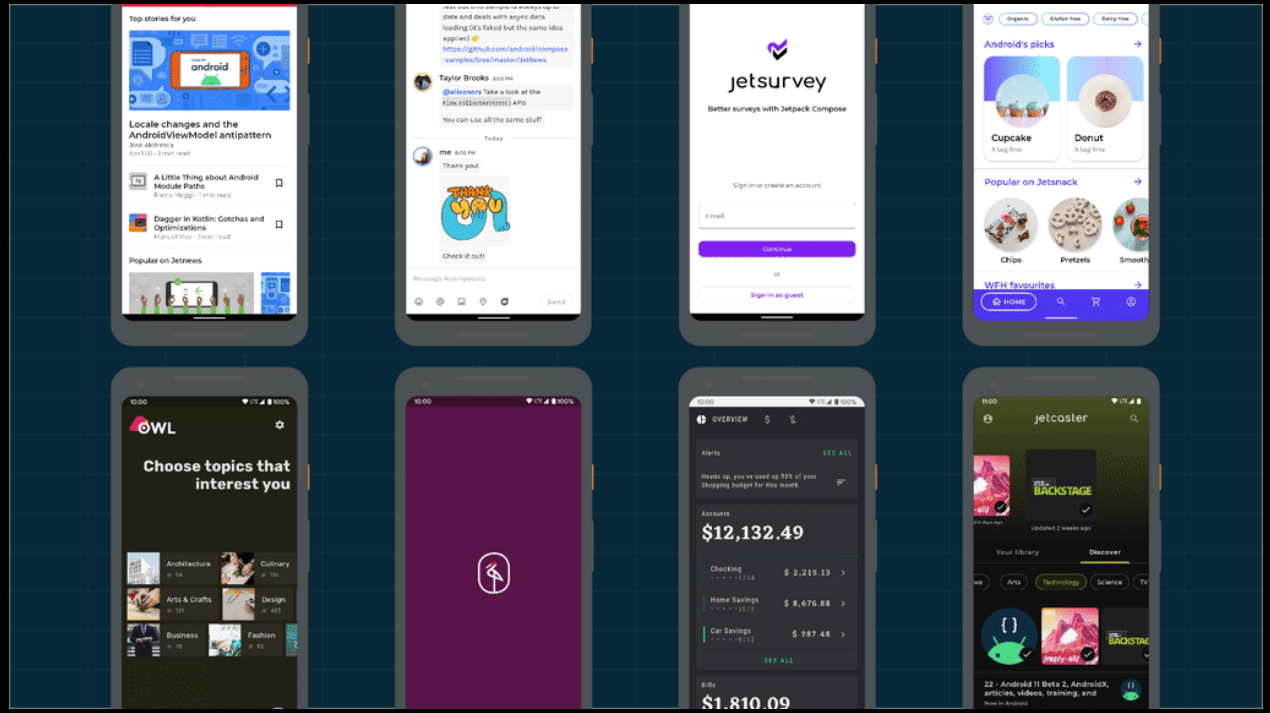
https://developer.android.com/jetpack/compose

Примеры Jetpack Compos



https://github.com/android/compose-samples

Примеры Jetpack Compos



https://github.com/android/compose-samples

Jetpack Compose Awesome

https://github.com/jetpack-compose/jetpack-compose-awesome

Примеры кода из презентации

https://github.com/AJIEKCX/ComposeTalk

Заключение

- Мы рассмотрели ключевые концепты Jetpack Compose
- Научились эффективно управлять стейтом в Compose
- Узнали как правильно работать с сайд эффектами
- Заглянули в будущее Android разработки
- Составили план рефакторинга для миграции на Jetpack Compose

Спасибо за внимание

Остались вопросы?

Контур

ajiekcx