

KTU kviečių paskaita

Susipažinkime su Jetpack Compose





Lektorius

Gytis Jakutonis



Daugiau kaip 20 metų dirbu IT srityje. Teko susidurti su įvairiausiomis technologijomis, projektais bei iššūkiais.

Teko dalyvauti keliuose projektuose kuriant Android native sprendimus:

- Kayak
- TransferGO
- StockManager
- ...



Visma ribose nedalyvauju Android projektuose, taigi pateikiamos įžvalgos bus iš ankstesnių karjeros etapų. Visma vystomi Android sprendimai remiasi naujausiomis technologijomis bei tendencijomis, taigi taiko tuos pačius principus, kurie bus išdėstyti toliau.



Auditorija

Susipažinkime

- Teko programuoti už universiteto ribų
- Teko kurti mobilias aplikacijas
- Teko naudoti Jetpack Compose
- Svarstau apie karjerą mobilių aplikacijų kūrimo srityje
- Turiu telefone įjungtą Developer režimą
- Tikiuosi Žalgiris pateks į Top-10





Kontekstas

Paskaitos tikslai

- <https://github.com/gytisjakutonis/KTUEtpackCompose>



 VISMA

Jetpack Compose - viena iš pagrindinių UI kūrimų priemonių/platformų. Todėl svarbu ją žinoti. Nėra tikslas detaliai išdėstyti, nes tema gana plati. Medžiagos apstu internete, svarbu tik žinoti, ko reikia ieškoti. Todėl esminis tikslas - duoti pradinį supratimą, raktinius žodžius bei impulsą savarankiškai gilinti žinias.



Įvadas

Truputis istorijos

Jetpack Compose idėja kilo siekiant supaprastinti Android native UI kūrimą.

1. 2017 - Pradinė konцепcija bei vizija
2. 2020 - Alfa ir Beta versijos
3. 2021 - Pirma stabili versija
4. Šiandien - standartas kuriant Android UI

Ivadas

```
@Composable  
fun MyScreenWithButton () {  
    Row {  
        Button(onClick = {}) {  
            Text(text = "Click Me")  
        }  
  
        Text(text = "123")  
    }  
}
```



Click Me

123

Konsepcija

UI programuojamas deklaratyviu būdu,
pasitelkiant Composable funkcijas.

Kiekviena funkcija aprašo tam tikrą UI
elementą.

Composable funkcijos žymimos su @Composable anotacija. Komponentai į UI įtraukiami programinio Kotlin kodo pavidalu.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk
    /res/android"
    android:layout_width="match parent"
    android:layout_height="match parent"
    android:orientation="horizontal"
    android:gravity="center"
    android:padding="16dp">

    <Button
        android:id="@+id/buttonClickMe"
        android:layout_width="wrap content"
        android:layout_height="wrap_content"
        android:text="Click Me" />

    <TextView
        android:id="@+id/textViewNumber"
        android:layout_width="wrap content"
        android:layout_height="wrap_content"
        android:text="123"
        android:layout_marginStart="16dp" />

</LinearLayout>
```

Jvadas

Konceptacija



Klasikinis XML aprašymas ne toks intuityvus ir gremėzdiškas.

Įvadas

Privalumai

```
@Composable
fun MyScreenWithButton () {
    Row {
        Button (onClick = {})
            Text (text = "Click Me")
    }

    Text (text = "123")
}
```

**Intuityvus ir lengvai suprantamas UI
aprašymas.**

Tiesioginis duomenų susiejimas.

Integruota įvykių apdorojimo logika.



Duomenys bei veiksmų apdorojimo logika pateikiami tiesiogiai per parametrus.

this is text

click me



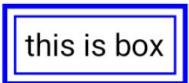
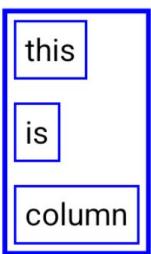
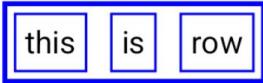
Maketavimas

Komponentai

Informaciniai bei valdymo komponentai:

- **Button**
- **Text**
- **Switch**
- **RadioButton**
- **ProgressIndicator**
- ...

Aibė standartinių komponentų, daug papildomų bibliotekų ir platus komponentų pasirinkimas.



Maketavimas

Išdėstymo įrankiai

Išdėstymo valdymo komponentai:

- Column
- Row
- Box
- ...

Išdėstymas atliekamas vienu etapu. Kiekvienam elementui apskaičiuojamas dydis, rekursyviai atliekamas vidinių elementų dydžio nustatymas. Taip apdorojamas visas medis vienu etapu.

Maketavimas

```
Text(  
    text = "this is decorated text",  
    modifier = Modifier  
        .padding(20.dp)  
        .background(  
            color = Color.Yellow)  
        .border(  
            width = 2.dp,  
            color = Color.Blue)  
        .padding(10.dp)  
        .clickable { }  
)
```

this is decorated text

Dekoravimo būdai

Dekoravimo komponentai:

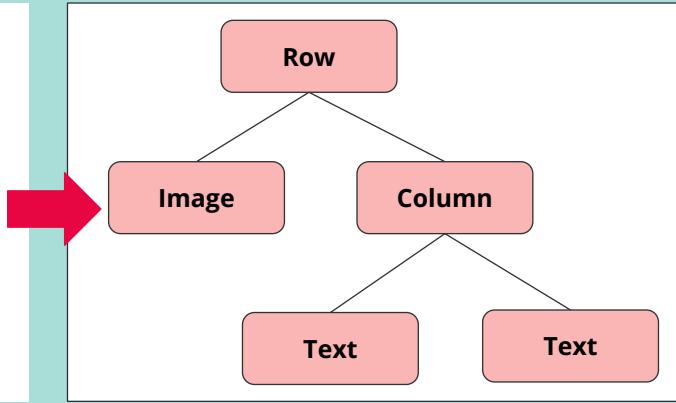
- **Modifier.padding**
- **Modifier.background**
- **Modifier.clickable**
- **Modifier.border**
- ...

Dekoravimo įrankiai įgalina keisti komponento vaizdavimą, pridėti papildomų vizualiinių detalių ar netgi interaktyvumo.

Maketavimas

UI formavimo eiga - Kompozicija

```
Row {  
    Image()  
    Column {  
        Text()  
        Text()  
    }  
}
```

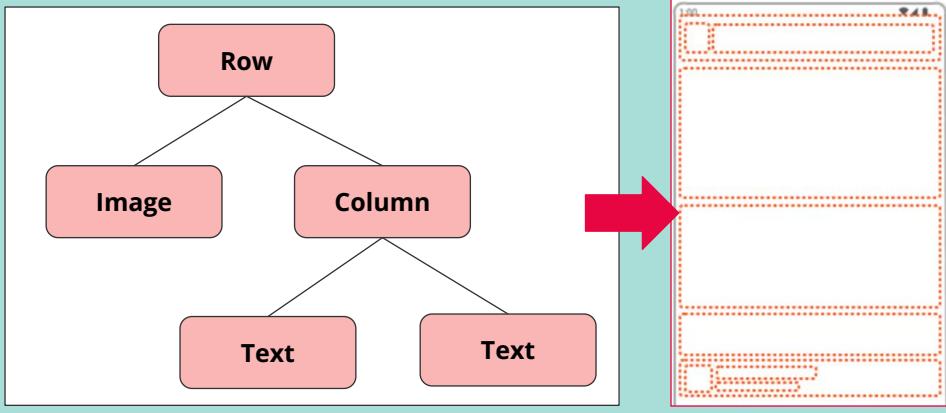


VISMA

Kompozicija - medžio formavimas. Kiekvienai funkcijai surinkiamas objektas medyje.

Maketavimas

UI formavimo eiga - Išdėstymas

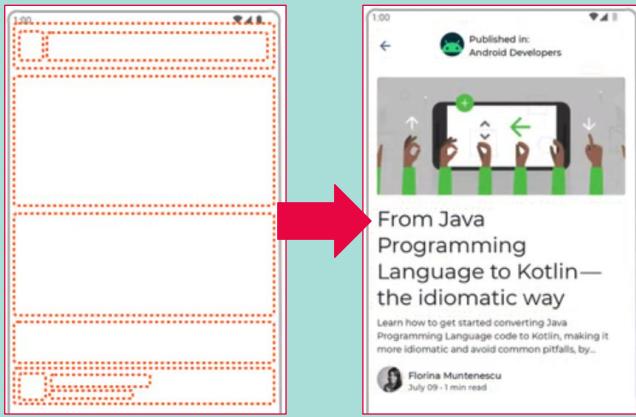


 VISMA

Išdėstymas - dydžio bei pozicijos perskaičiavimas. Perskaičiuojamas rekursyviai visas medis.

Maketavimas

UI formavimo eiga - Piešimas

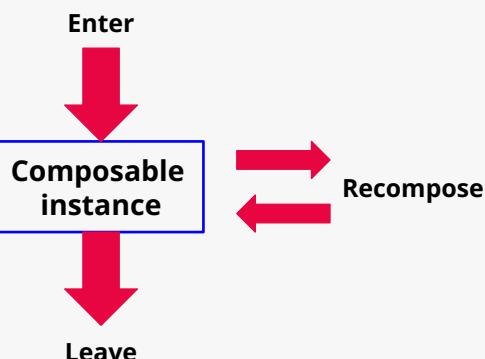


 VISMA

Piešimas - komponentų atvaizdavimas ekrane.

Rekompozicija

Gyvavimo ciklas



Rekompozicija - pagrindinis Jetpack Compose UI atnaujinimo principas, užtikrinantis reaktyvų veikimą bei gerą vartotojo patirtį



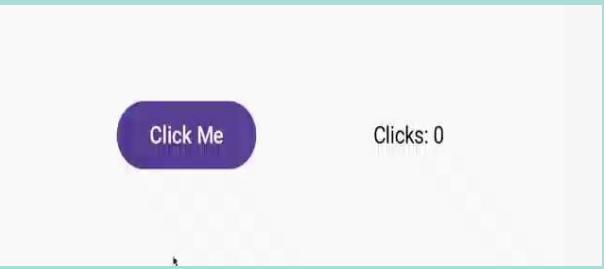
Kiekvienas iškvietimas sukuria atskirą Composable objektą bendrame kompozicijos medyje. Rekompozicija - tai procesas, kurio metu UI performuojamas įtraukiant tik tuos objektus, kurių vaizdavimas priklauso nuo pasikeitusios būsenos State<T>. Panašus principas taikomas React bei SwiftUI.

```
@Composable
fun MyClickCounter() {
    var clicks = 0
    Button(onClick = { clicks++ }) {
        Text(text = "Click Me")
    }
    Text(text = "Clicks: $clicks")
}
```

Rekompozicija

State<T>

Rekompozicija remiasi State<T> tipo parametrais. Kitais atvejais rekompozicija nevyksta.



Click Me

Clicks: 0



Rekompozicija neatliekama, jeigu nėra State<T> parametru arba nėra jų pokyčių. Šiuo atveju clicks pokytis neinicijuoja rekompozicijos, nes tai yra paprastas Int tipas - turėtų būti State<Int>.

```
@Composable
fun MyClickCounterWithState() {
    var clicks by remember {
        mutableIntStateOf(0)
    }
    Button(onClick = { clicks++ }) {
        Text(text = "Click Me")
    }
    Text(text = "Clicks: $clicks")
}
```

Rekompozicija

State<T>

State<T> tipo click parametras inicijuoja rekompozicija, kurioje atnaujinamas tik Text() komponentas

Rekompozicija

mutableIntStateOf

```
var clicks1 by remember {  
    mutableIntStateOf(0)  
}  
var clicks2 by remember {  
    mutableStateOf(0)  
}
```

Sukuriamas MutableIntState tipo objektas su pradine reikšme 0.

Bendru atveju galima naudoti ir mutableStateOf(0).

MutableState<> papildo State<> galimybe keisti saugomą reikšmę bei taip inicijuoti rekompoziciją



mutableStateOf() yra tam tikra State<> versija, kuri leidzia keisti saugomą reikšmę, taigi inicijuoti rekompoziciją. MutableIntState - tai specializuotas MutableState<T> tipas būtent Int reikšmėms saugoti.

Rekompozicija

remember

```
var clicks by remember {  
    mutableIntStateOf(0)  
}
```

Remember išsaugo inicializuotą objektą Composable gyvavimo metu, nepaisant to, kiek kartų būtų atlikta rekompozicija.

Kitu atveju objektas būtų perkuriamas kiekvienos rekompozicijos metu.



Remember naudojamas ne tik `MutableState<>` parametrams saugoti, bet ir bet kokiems duomenims, kurių reikšmės turėtų išlikti kol gyvuoja kompozicija, bei kurių inicializavimas yra brangus. Jie inicializuojami tik pirmos kompozicijos metu, o rekompozicijų metu išlaiko savo reikšmę. Kadangi Composable yra funkcija, tai rekompozicijos metu ji vykdoma nuosekliai, taigi kintamieji inicializuojami iš naujo, jeigu jie nėra registruoti su remember.

Rekompozicija

by

```
var clicks by remember {  
    mutableIntStateOf(0)  
}  
clicks++  
val i: Int = clicks
```

```
val clicks = remember {  
    mutableIntStateOf(0)  
}  
clicks.intValue++  
val i: Int = clicks intValue
```

by yra kotlin delegavimo funkcija, šiuo atveju
'paverčianti' MutableIntState į Int.



By operacija yra kotlin standartinė delegavimo funkcija, kuri supaprastina šiuo atveju MutableState<> naudojimą. Kodas atrodo natūraliau. Bet ji nėra privaloma naudoti.

Rekompozicija

rememberSaveable

```
var clicks by rememberSaveable {  
    mutableIntStateOf(0)  
}
```

remember funkcija neišsaugo parametru
reikšmių pilnos rekompozicijos atveju.



Pilna rekompozicija vyksta config change atveju, kai perkuriame visi Activity ir Fragment. Tai vyksta pvz po: rotacijos, kalbos pakeitimo, telefono fontų pakeitimo. rememberSaveable naudoja Activity SavedInstanceState mechanizmą, taigi saugomiems objektams taikomi tam tikri apribojimai. Taigi tai nėra idealus sprendimas visiems atvejams, dažniau naudojami kiti būdai būsenai atstatyti.

```
@Composable
fun ClickCounter() {
    var clicks by remember {
        mutableIntStateOf(0)
    }
    ClickButton { clicks++ }
    ClickText(clicks)
}

@Composable
fun ClickButton(onClick: () -> Unit) {
    Button(onClick = onClick) {
        Text(text = "Click Me")
    }
}

@Composable
fun ClickText(clicks: Int) {
    Text(text = "Clicks: $clicks")
}
```

Rekompozicija

Būsenos iškėlimas

State hoisting - viena iš esminų Jetpack

Compose konceptijų formuojant UI struktūrą.

Būsena turi būti iškeliamā į kuo aukštesnį kompozitų medžio lygį bei perduodama per parametrus.



ClickButton gali būti panaudotas ir kitiems veiksmams. Tuo tarpu ClickText pats neturi būsenos ir gauna reikšmę per parametrus.

```
var rotate by remember {  
    mutableStateOf(false)  
}  
  
val rotationAngle by  
    animateFloatAsState(  
        targetValue =  
            if (rotate) 90f  
            else 0f,  
        animationSpec = tween(500)  
    )  
  
Box(  
    modifier = Modifier  
        .size(100.dp)  
        .rotate(degrees = rotationAngle)  
        .background(color = Color.Blue)  
        .clickable { rotate = !rotate }  
)
```

Animacija

Būsenos animacija

Komponentų dekoravimas, pagrįstas būsenos pokyčiu.



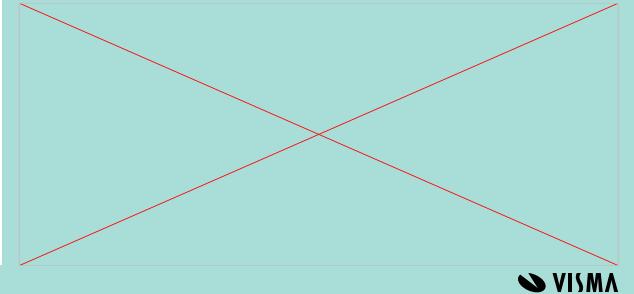
Jetpack Compose turi plačias animacijos taikymo galimybes. Taikymas intuityvus ir integruotas į bendrą Composable API. Būsenos animacija tolygiai (pagal nurodytą animacijos modelį) keičia būseną iš pradinės į nurodytą. Būsena naudojama dekoruoti komponentus, kurie tolygiai atnaujinami kintant būsenai. Šiuo atveju Box pasukimo kampus tolygiai kinta nuo 0 iki 90 ir atgal. Pokytį iniciuoja click įvykis.

```
var rotate by remember {  
    mutableStateOf(false)  
}  
val rotationAngle by animateFloatAsState(  
    targetValue = if (rotate) 90f else 0f,  
    animationSpec = tween(500)  
)  
  
val backgroundColor by  
    animateColorAsState(  
        targetValue = if (rotate)  
            Color.Green else Color.Blue,  
        animationSpec = tween(1000)  
)  
  
Box(  
    modifier = Modifier  
        .size(100.dp)  
        .rotate(degrees = rotationAngle)  
        .background(color=backgroundColor)  
        .clickable { rotate = !rotate }  
)
```

Animacija

Būsenos animacija

Animacijos gali būti įvairiai konstruojamos naudojant ne vieną animuotą būseną



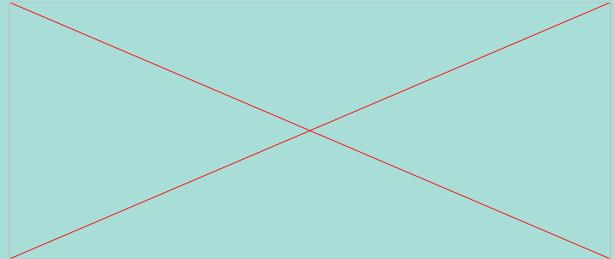
Vienam Box pritaikomos kelios animacijos - tiek rotacijos, tiek užpildymo spalvos. Abi animacijos veikia nepriklausomai.

```
var show by remember {  
    mutableStateOf(false)  
}  
  
Row {  
    Button(onClick = { show = !show }) {  
        Text(if (show) "Hide" else "Show")  
    }  
  
    AnimatedVisibility(visible = show) {  
        Box(  
            modifier = Modifier  
                .size(  
                    height = 40.dp,  
                    width = 100.dp)  
                .background(color = Color.Red)  
                .clickable { show = !show }  
        )  
    }  
}
```

Animacija

Rodymo animacija

Animuotas komponentų rodymas.



Komponentas rodomas ar slepiamas pritaikant animaciją. Animaciją galima keisti keičiant AnimatedVisibility parametrus.

Animacija

Rodymo animacija

```
@Composable
fun RowScope.AnimatedVisibility (
    visible: Boolean,
    modifier: Modifier = Modifier,
    enter: EnterTransition = fadeIn() +
        expandHorizontally(),
    exit: ExitTransition = fadeOut() +
        shrinkHorizontally(),
```

Transformacijos gali būti apjungiamos.



Rodymo animacijas galima konfigūruoti įvairiai apjungiant transformacijas. Šiuo atveju fadeln apjungiamas su expand, taigi komponentas tolygiai išryškėja ir tuo pat metu išsiplečia horizontaliai.

```
@Composable
fun AnimatedContents() {
    var count by remember {
        mutableIntStateOf(0)
    }

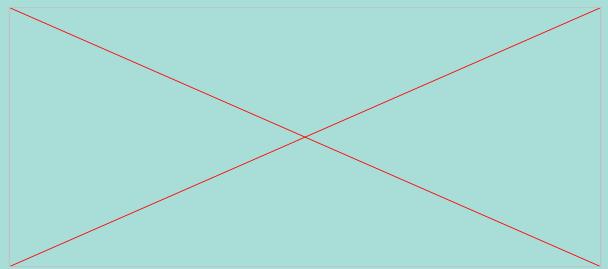
    Row {
        ClickButton { count++ }

        AnimatedContent(
            targetState = count,
            transitionSpec = {
                slideInHorizontally { it } + fadeIn()
                togetherWith
                slideOutHorizontally { -it } + fadeOut()
            }
        ) { targetClicks ->
            ClickText(targetClicks)
        }
    }
}
```

Animacija

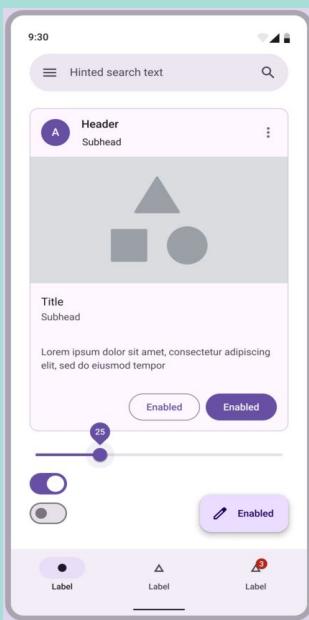
Turinio animacija

Būsenos pokytis inicijuoja rekompoziciją, pritaikant nurodytą transformaciją.



 VISMA

Transformacijos taikomos tiek senam turiniui, tiek naujam, taip sukuriant sklandaus perėjimo efektą. Senas turinys keičiamas nauju, pasitelkiant nurodytą perėjimo animaciją. Animaciją inicijuoja targetState pokytis. Čia taip pat transformacijos apjungiamos, taip sukuriamas natūralus ir tolygus perėjimas.



Animacija

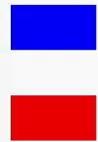
Materialus dizainas

Materialaus dizaino principai:

- Natūralios spalvos, formos ir animacija
- Akcentuotas išdėstymas
- Tikslinė animacija

Išdėstymas pabrėžia svarbius veiksmus bei informaciją. Animacija perteikia papildomą infomaciją - ryšius tarp elementų, veiksmų atsaką ir pan. Jetpack Compose orientuotas į materialų dizainą: plati aibė komponentų, užtikrintas adaptyvumas, integracija su platformomis, temų naudojimas, prieinamumo įrankiai.

```
val tweenOffsetX by animateDpAsState(  
    targetValue =  
        if (isAtEnd) 200.dp else 0.dp,  
    animationSpec = tween(  
        durationMillis = 1000,  
        easing = LinearOutSlowInEasing)  
)  
  
val springOffsetX by animateDpAsState(  
    targetValue =  
        if (isAtEnd) 200.dp else 0.dp,  
    animationSpec = spring(  
        dampingRatio =  
            Spring.DampingRatioMediumBouncy)  
)
```



Animacija

Konfigūravimas

Animacijų konfigūravimas remiasi pokyčio modelio parametrais. Pokyčio modeliai:

- **tween**
- **spring**
- **keyframes**
- **repeatable**
- ...

Pokyčio konfigūravimas įgalina sukurti netiesines pokyčio funkcijas. Taip sukuriamas natūralesnė animacija, t.y. Išlaikomas Material Design principas. Easing - pokyčio spartos kontrolė, šiuo atveju lėtėjanties pokytis. Spring atveju pakeičiamas spyruokliaus intensyvumas.

Temos

UI nuoseklumas

```
@Composable  
fun MaterialTheme (  
    colorScheme: ColorScheme,  
    shapes: Shapes,  
    typography: Typography,  
)
```

UI temos užtikriną nuoseklią sąsają, taigi teigiamą vartotojo patirtį. Tema tai:

- Spalvų paletė
- Tipografija
- Formos

Pagal nutylėjimą UI komponentai naudoja aktyvią temą.

```
val Purple80 = Color(0xFFD0BCFF)
val PurpleGrey80 = Color(0xFFCCC2DC)
val Pink80 = Color(0xFFEFB8C8)

val Purple40 = Color(0xFF6650a4)
val PurpleGrey40 = Color(0xFF625b71)
val Pink40 = Color(0xFF7D5260)

private val DarkColorScheme =
    darkColorScheme(
        primary = Purple80,
        secondary = PurpleGrey80,
        tertiary = Pink80
    )

private val LightColorScheme =
    lightColorScheme(
        primary = Purple40,
        secondary = PurpleGrey40,
        tertiary = Pink40
    )
```

Temos

Spalvų paletės

Paletėse nustatomos komponentuose naudojamos spalvos.

Spalvų paletė aprašoma tiek dienos tiek nakties režimui. Esant poreikiui - kuriamos papildomos paletės, pvz didelio kontrasto, monochrominės ir pan.

```
fun darkColorScheme  
    primary: Color = ColorDarkTokens.Primary,  
    onPrimary: Color = ColorDarkTokens.OnPrimary,  
    primaryContainer: Color = ColorDarkTokens.PrimaryContainer,  
    onPrimaryContainer: Color = ColorDarkTokens.OnPrimaryContainer,  
    inversePrimary: Color = ColorDarkTokens.InversePrimary,  
    secondary: Color = ColorDarkTokens.Secondary,  
    onSecondary: Color = ColorDarkTokens.OnSecondary,  
    secondaryContainer: Color = ColorDarkTokens.SecondaryContainer,  
    onSecondaryContainer: Color = ColorDarkTokens.OnSecondaryContainer,  
    tertiary: Color = ColorDarkTokens.Tertiary,  
    onTertiary: Color = ColorDarkTokens.OnTertiary,  
    tertiaryContainer: Color = ColorDarkTokens.TertiaryContainer,  
    onTertiaryContainer: Color = ColorDarkTokens.OnTertiaryContainer,  
    background: Color = ColorDarkTokens.Background,  
    onBackground: Color = ColorDarkTokens.OnBackground,  
    surface: Color = ColorDarkTokens.Surface,  
    onSurface: Color = ColorDarkTokens.OnSurface,  
    surfaceVariant: Color = ColorDarkTokens.SurfaceVariant,  
    onSurfaceVariant: Color = ColorDarkTokens.OnSurfaceVariant,  
    surfaceTint: Color = primary,  
    inverseSurface: Color = ColorDarkTokens.InverseSurface,  
    inverseOnSurface: Color = ColorDarkTokens.InverseOnSurface,  
    error: Color = ColorDarkTokens.Error,  
    onError: Color = ColorDarkTokens.OnError,  
    errorContainer: Color = ColorDarkTokens.ErrorContainer,  
    onErrorContainer: Color = ColorDarkTokens.OnErrorContainer,  
    outline: Color = ColorDarkTokens.Outline,  
    outlineVariant: Color = ColorDarkTokens.OutlineVariant,  
    scrim: Color = ColorDarkTokens.Scrim,  
    surfaceBright: Color = ColorDarkTokens.SurfaceBright,  
    surfaceContainer: Color = ColorDarkTokens.SurfaceContainer,  
    surfaceContainerHigh: Color = ColorDarkTokens.SurfaceContainerHigh,  
    surfaceContainerHighest: Color = ColorDarkTokens.SurfaceContainerHighest,  
    surfaceContainerLow: Color = ColorDarkTokens.SurfaceContainerLow,  
    surfaceContainerLowest: Color = ColorDarkTokens.SurfaceContainerLowest,  
    surfaceDim: Color = ColorDarkTokens.SurfaceDim,  
): ColorScheme =
```

Temos

Spalvų paletės

Paletę sudaro daugybė spalvų.



Peržiūrėdami standartinių komponentų parametrus galite rasti kokia spalva naudojama pagal nutylėjimą. Tai bus viena iš paletėje esančių spalvų.

Temos

Dinaminės spalvos

```
val colorScheme = when {
    Build.VERSION.SDK_INT >=
Build.VERSION_CODES.S -> {
        val context = LocalContext.current
        if (darkTheme)
            dynamicDarkColorScheme(context)
        else
            dynamicLightColorScheme(context)
    }

    darkTheme -> DarkColorScheme
    else -> LightColorScheme
}
```

Spalvų paletė sugeneruojama automatiškai,
pagal mobiliaus įrenginio foną. Tik nuo API v31

Temos

Tipografija

```
val Typography = Typography(  
    bodyLarge = TextStyle(  
        fontFamily = FontFamily.Default,  
        fontWeight = FontWeight.Normal,  
        fontSize = 16.sp,  
        lineHeight = 24.sp,  
        letterSpacing = 0.5.sp  
    )  
)
```

Tipografijos pagalba konfigūruojami sriftai bei jų dydžiai.

```
Typography(  
    displayLarge = displayLarge,  
    displayMedium = displayMedium,  
    displaySmall = displaySmall,  
    headlineLarge = headlineLarge,  
    headlineMedium = headlineMedium,  
    headlineSmall = headlineSmall,  
    titleLarge = titleLarge,  
    titleMedium = titleMedium,  
    titleSmall = titleSmall,  
    bodyLarge = bodyLarge,  
    bodyMedium = bodyMedium,  
    bodySmall = bodySmall,  
    labelLarge = labelLarge,  
    labelMedium = labelMedium,  
    labelSmall = labelSmall  
)
```

Temos

Tipografija

Kaip ir spalvų palečių atveju - daugybė konfigūravimo parametrų.

Kaip ir spalvų atveju, komponentų parametruose galime rasti, koks tipografijos tipas yra naudojama tam komponentui vaizduoti.

Temos

Formos

```
Shapes (  
    val extraSmall: CornerBasedShape,  
    val small: CornerBasedShape,  
    val medium: CornerBasedShape,  
    val large: CornerBasedShape,  
    val extraLarge: CornerBasedShape,  
)
```

Formose nustatomas tiek dydis, tiek geometrinė forma.

Forma - pvz. Stačiakampis su užapvalintais kampais, nustatant netgi tų kampų spindulį.

```
@Composable  
fun DemoTheme ( //  
    darkTheme: Boolean =  
    isSystemInDarkTheme (),  
    content: @Composable () -> Unit  
) {  
    val colorScheme = when {  
        darkTheme -> DarkColorScheme  
        else -> LightColorScheme  
    }  
  
    MaterialTheme ( //  
        colorScheme = colorScheme,  
        typography = Typography,  
        shapes = MyShapes,  
        content = content  
    )  
}
```

Temos

Taikymas

Temos įprastai nustatomos aukščiausiamame UI kompozicijos medžio taške.

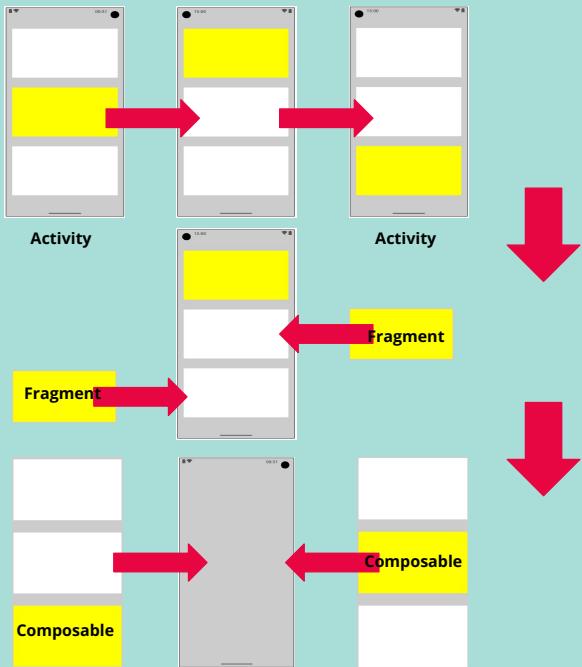
```
class MainActivity : ComponentActivity() {  
    override fun onCreate(  
        savedInstanceState: Bundle?  
    ) {  
        super.onCreate(savedInstanceState)  
  
        enableEdgeToEdge()  
  
        setContent {  
            DemoTheme {  
                MyScreenWithAnimations()  
            }  
        }  
    }  
}
```

Temos

Taikymas

Kompozicija įdedama į Activity.

Norint naudoti Composable, Activity turi būti išvestas iš ComponentActivity. Tada pakanka tik perduoti norimą Composable į setContent.



Struktūra

Ekranų struktūra

Iprastinis multi-Activity ir multi-Fragment konceptas keičiamas į single-Activity multi Composable.

Multi-Activity keičiami į multi-Fragment, o šie į multi-Composable. Kiekvienas ekranas aprašomas kaip savarankiškas Composable. Tuo tarpu Activity yra vienas. Fragmentų naudojimas praranda prasmę. Lengvasvorai komponentai, skirti pakartotiniam naudojimui ir UI fragmentacijai - tai pilnai išpildo Composable. Rekompozicija įgalina lengvai atnaujinti atskirus ekrano komponentus. Atskiri activity turi prasmę specifiniais atvejais: skirtinges darbo režimas (prisijungės vartotojas ar svečias), push-notification apdorojimas, modularizacija (business ar basic versijos) ir pan.

```
@Composable
fun MyNavigation() {
    val navController =
        rememberNavController()

    NavHost(
        navController = navController,
        startDestination = "home"
    ) {
        composable("home") {
            HomeScreen(navController)
        }
        composable("detail") {
            DetailScreen(navController)
        }
    }
}
```

Struktūra

Navigacija

Jetpack Compose turi priemones, užtikrinančias navigaciją tarp atskirų Composable.

NavController yra pagrindinis navigacijos valdymo komponentas. NavHost padeda suformuoti navigacijos grafą, kuris susideda iš atskirų composable. Kiekvienas grafo elementas turi tekstinį raktą. startDestination - grafo pradžios raktas. Kaip ir visur, NavHost galima konfigūruoti navigavimo animacijas per papildomus parametrus.

Struktūra

Navigacija

```
@Composable
fun HomeScreen(
    navController: NavHostController
) {
    Box {
        Button(
            onClick = {
                navController.navigate("detail")
            }) {
            Text("Go to Detail")
        }
    }
}
```

NavHostController turi API navigacijai valdyti.



Navigacijos elementai - įprasti Composable. Navigacija vyksta grafo raktų pagalba.

Struktūra

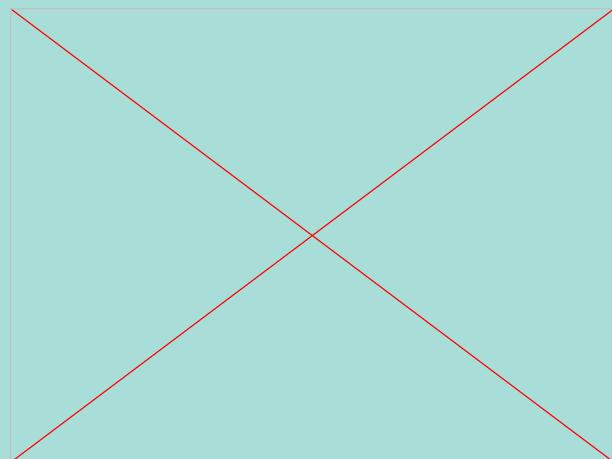
Navigacija

```
@Composable
fun DetailScreen(
    navController: NavHostController
) {
    Box {
        Button(
            onClick = {
                navController.popBackStack()
            }) {
            Text("Back to Home")
        }
    }
}
```

Kiekvienas NavHostController turi savo backstack.



BackStack įgalina naudotis navigacijos istorija.



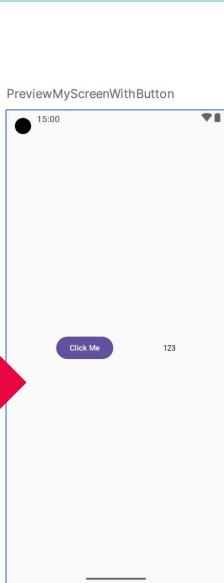
Struktūra

Navigacija

Vienu metu galima naudotis keliais navigacijos grafais, taip suformuojant gana sudėtingą UI konfigūraciją.

Atkreipkite dėmesį - navigacija taip pat turi animacijos galimybes. Pagal nutylėjimą taikoma fadeIn/fadeOut su tween.

```
@Composable  
fun MyScreenWithButton() {  
    Row(  
        modifier = Modifier.fillMaxSize()  
        verticalAlignment = Alignment.C  
        horizontalArrangement = Arrange  
) {  
        Button(onClick = {}) {  
            Text(text = "Click Me")  
        }  
  
        Text(text = "123")  
    }  
}  
  
@Preview(showSystemUi = true)  
@Composable  
private fun PreviewMyScreenWithButton()  
    MyScreenWithButton()  
}
```



Jrankiai

AndroidStudio peržiūros režimas



Sukuriamas atskira kompozicija peržiūrai. I ją įtraukiame originalią kompoziciją, kuria norime pamatyti.

Today

💡 During the winter your plants slow down and need less water.

Living Room

- Water hoya australis
- Feed monstera siltepecana



Kitchen

- Water pilea peperomioides
- Water hoyo australis



Bedroom

- Feed monstera siltepecana
- Water philodendron brandtii



Įrankiai

Theme builder

Pagalbinė priemonė kuriant temos spalvinę paletę.

<https://material-foundation.github.io/material-theme-builder/>

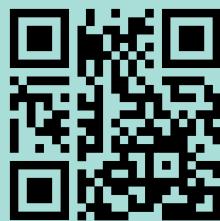


VISMA

Spalvų paletė suformuojama pagal nurodytą foną arba pagrindinę spalvą.

Informacija

Informacijos šaltiniai



<https://developer.android.com/compose>

<https://m3.material.io/>

<https://fonts.google.com/>

<https://www.youtube.com/@codingwithmitch>

<https://composables.com/>

Informacija

Verta žinoti

Darbas su vaizdais: Coil / Glide / Picasso

Darbas su duomenimis: Room / Realm

Komunikacijos: Retrofit / OkHttp / Moshi

Ryšių valdymas: Dagger / Hilt / Koin

Asinhroniniai procesai: Coroutines / Flow

Žemėlapiai: Maps SDK

Irenginiai: Bluetooth Classic / LE

Saugumas: EncryptedPrefs / Firebase Auth

Pranešimai: Firebase Cloud Messaging

Monitoringas: Firebase Analytics / Timber

Auditai: Firebase Crashlytics / LeakCanary

Testavimas: Mockito / Espresso



Néra prasmės viską žinoti mintinai, bet tai dalis technologijų, su kuriomis tenka susidurti.



Perspektyvos

Karjera

Mobile sprendimų kūrimas: dinamiška, inovatyvi, konkurencinga sritis.

Multiplatform žinios užtikrina geras pozicijas darbo rinkoje.

Jetpack Compose Multiplatform



Daug pokyčių vyksta nuolatos. Didelė konkurencija dėl gana gero prieinamumo ir viešinimo. Nuolatinis mokymasis, ypač UX srityje. Šią sritį, tikėtina, vėliausiai palies AI. Platus žinių spektras, lyginant su specializuotomis rolėmis (backend, frontend, UX design, ...). Multiplatform žinių svoris: Android, Swift, Flutter, Kotlin Multiplatform Mobile, Jetpack Compose Multiplatform.



Kas mes

Visma

Visma - tai įmonių grupė, kur kiekviena įmonė išlaiko savo startuolio mentalitetą, unikalią kultūrą ir lyderystę.

Kuriame ir tiekiame programinę įrangą mažoms, vidutinėms ir didelėms įmonėms, taip pat viešajam sektoriui.



Įmonių grupė, apjungianti įmones Europoje (ypač Skandinavų šalyse, Danijoje, Nyderlanduose) bei Lotynų Amerikoje. Nemaža dalis sprendimų skirta viešajam sektoriui, kaip švietimo įstaigos, institucijos bei viešojo sektoriaus įmonės.



Kas mes

Visma skaičiais

klientų

1.9M

smulkijų versly

1.3m

Įmonių

180+

bendrų pajamų

€2,719M

 VISM^A



Kas mes

Visma-Lease a Bike

Dviračių nuomos paslauga

ARBA

Viena iš stipriausių šiuo metu dviračių sporto komandų, 2024 metų Tour de France prizininkai individualioje bei komandinėje įskaitose.



Dviračių nuomos paslauga arba sėkminga dviračių komanda.



Kas mes

Visma Tech Lietuva

Vilnius Kaunas

kolegų

270+

partnerių

20+





Kas mes

Technologijos Vismoje

Vismoje naudojamas platus technologijų spektras.

Programinės įrangos kūrimo metu sutelkiame dėmesį į vartotojų poreikius ir iššūkius.

Kurdami produktus, siekiame tobulumo, pritaikydamis modernias technologijas.

 VISMA

Daug vidinių iniciatyvų (komitetų), tiek susijusių su kvalifikacijos kėlimu, patirties bei žinių sklaida, tiek su laisvalaikio veiklomis.



Kas mes

Visma Android sprendimai

Visma sprendimų gausa neapsieina be Android technologijų. Mobilieji sprendimai kuriami pasitelkiant naujausios technologijas bei principus:

- Visma eAccounting
- Visma Scanner
- Min Barnehage Foresatt
- Min skole - foresatt
- ...





Ačiū už dēmesī

Klausimai ???

 VISMA