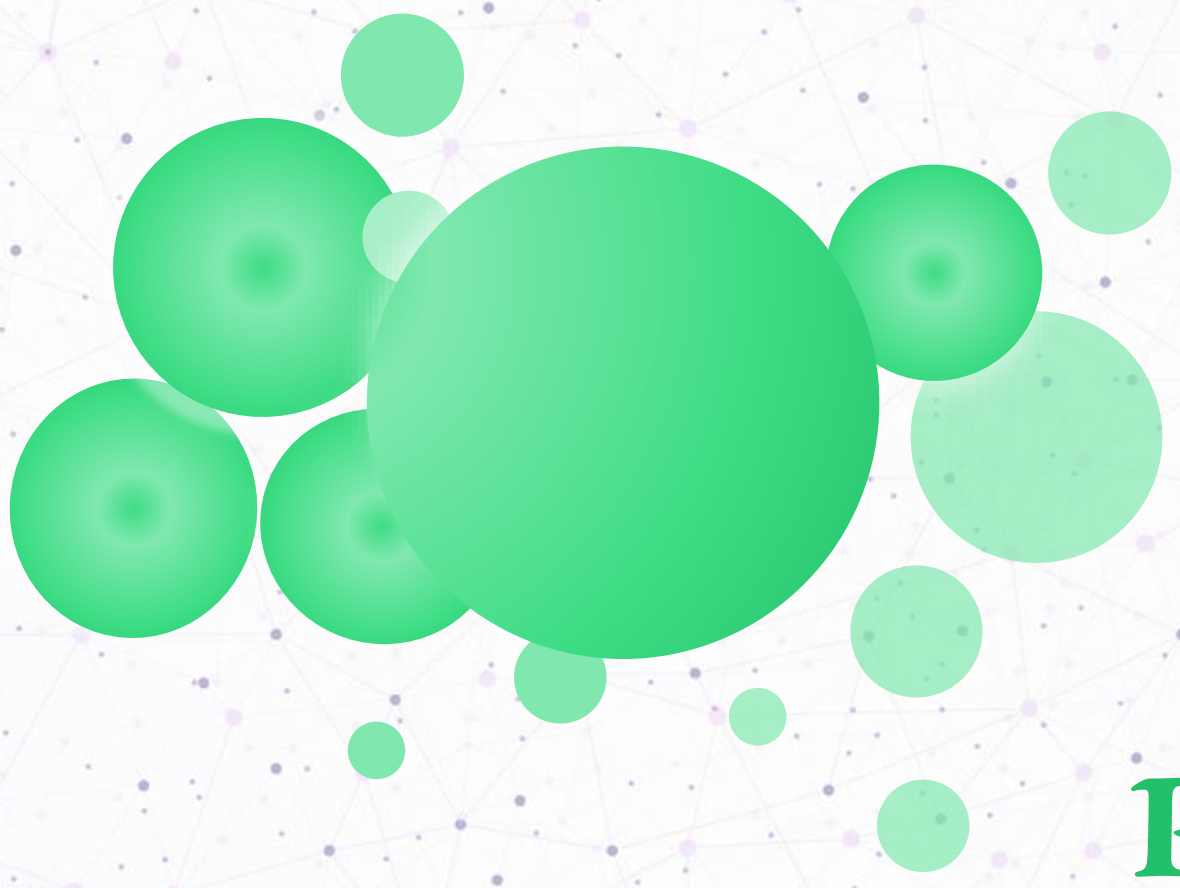


# PV256
# VÝVOJ NA ANDROID

Ing. Štefan Krajanec

3. prednáška – 5. 3. 2024

# Ciele prednášky

- Compose komponenty
- Compose téma
- Compose Modifier
- Príklad animácie
- Pamätanie hodnôt
- Aktivita: základy

# Compose – výhody

Jetpack Compose

- integracia Kotlinu
- zjednodušenie kódu
- odstránenie xml layotu
- efektívna práca s animáciou

```
@Composable
fun Greeting() {
    var text by remember { mutableStateOf( value: "Ahoj svet!") }
    Box { this: BoxScope
        Text(text = text, style = MaterialTheme.typography.bodyLarge)
        Button(onClick = { text = "Zmenený text" }) { this: RowScope
            Text( text: "Zmeniť text", style = MaterialTheme.typography.bodySmall)
        }
    }
}
```

# Compose – zjednodušenie kódu

Jetpack Compose

```xml
<?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">
    <!-- res/layout/activity_main.xml -->
    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Ahoj svet!" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="button"/>
</LinearLayout>
```

```kotlin
@Composable
fun Greeting() {
    var text by remember { mutableStateOf( value: "Ahoj svet!") }
    Text(text = text)
    Button(onClick = { text = "Zmenený text" }) { this: RowScope
        Text( text: "Zmeniť text")
    }
}
```

# Compose – integracia s Kotlinom

**Asynchronnost**

- Coroutine
  - » StateFlow
  - » SharedFlow
  - + mutable
- LiveData
  - » LiveData
  - » MutableLiveData

**Dispachers**

- Main
- IO

**Operátory**

- ? : , !! …
- let
- apply

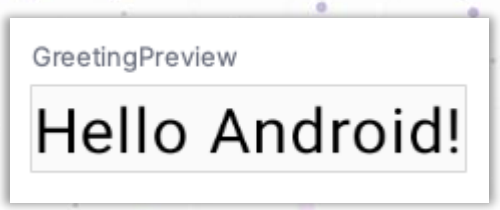
```
@Composable
fun UserProfile(userId: String) {
    val userData by rememberCoroutineScope().async { this: CoroutineScope
        getUserData(userId)
    }.collectAsState(initial = null)

    userData?.let { user ->
        Text( text: "Meno: ${user.name}")
    }
}
```

# Compose – UI nástroje

## Téma

- Témy v Jetpack Compose umožňujú definovať konzistentný vizuálny štýl pre celú aplikáciu. Téma môže obsahovať farby, typografiu, tvarovanie a iné dizajnové atribúty.
- V Compose sa témy aplikujú deklaratívne, čo znamená, že definujete tému na vysokej úrovni a automaticky sa aplikuje na všetky UI komponenty.

```
@Preview(showBackground = true)
@Composable
fun GreetingPreview() {
    P3Theme {
        Greeting(name = "Android")
    }
}
```




```
stringResource(id = R.string.app_name)
painterResource(id = R.drawable.ic_launcher_foreground)
colorResource(id = R.color.white)
```

# Compose – Téma

```
18  private val DarkColorScheme = darkColorScheme(
19      primary = Purple80,
20      secondary = PurpleGrey80,
21      tertiary = Pink80
22  )
23
24  private val LightColorScheme = lightColorScheme(
25      primary = Purple40,
26      secondary = PurpleGrey40,
27      tertiary = Pink40
```

```
val MyShapes = Shapes(
    small = RoundedCornerShape(4.dp),
    medium = RoundedCornerShape(8.dp),
    large = RoundedCornerShape(16.dp)
)
```

```
val Typography = Typography(
    bodyLarge = TextStyle(
        fontFamily = FontFamily.Default,
        fontWeight = FontWeight.Normal,
        fontSize = 16.sp,
        lineHeight = 24.sp,
        letterSpacing = 0.5.sp
    ),
    titleLarge = TextStyle(
        fontFamily = FontFamily.Default,
        fontWeight = FontWeight.Normal,
        fontSize = 22.sp,
        lineHeight = 28.sp,
        letterSpacing = 0.sp
    ),
    labelSmall = TextStyle(
        fontFamily = FontFamily.Default,
        fontWeight = FontWeight.Medium,
        fontSize = 11.sp,
        lineHeight = 16.sp,
        letterSpacing = 0.5.sp
    )
)
```

# Compose – Téma – použitie

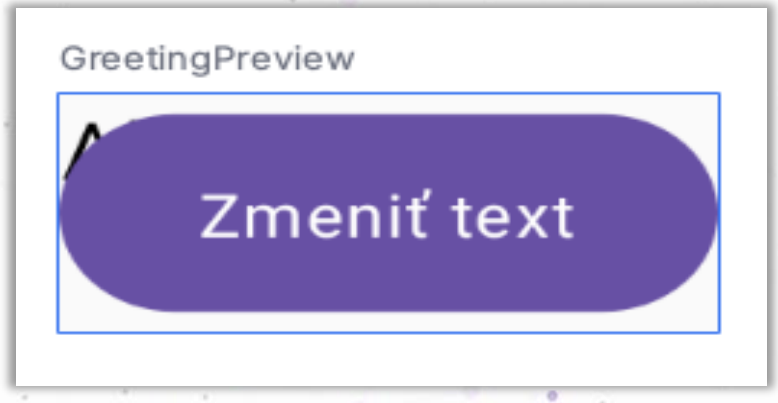
```
val colorScheme = when {
    dynamicColor && Build.VERSION.SDK_INT >= Build.VERSION_CODES.S -> {
        val context = LocalContext.current
        if (darkTheme) dynamicDarkColorScheme(context) else dynamicLightColorScheme(context)
    }

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

```
P3Theme() {
    // Použitie štýlu 'body1' z vašej typografie na Text komponent
    Text(text = "Toto je bežný text", style = MaterialTheme.typography.bodyLarge)
    // Použitie štýlu 'h1' na iný Text komponent
    Text(text = "Toto je nadpis", style = MaterialTheme.typography.titleLarge)
}
```

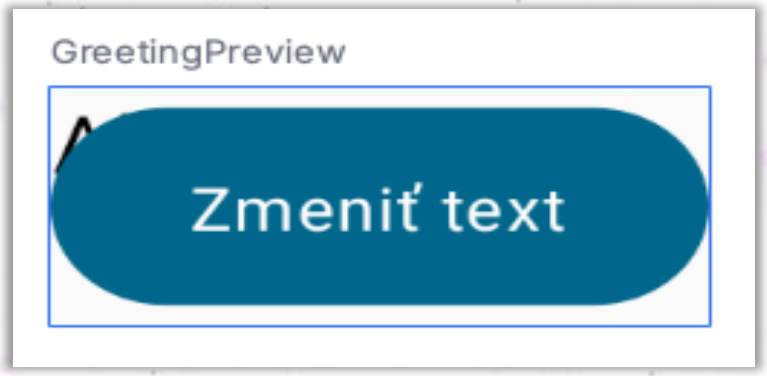
```
MaterialTheme(
    colorScheme = colorScheme,
    typography = Typography,
    shapes = MyShapes,
    content = content
)
```

# Compose – Layouts – Preview – Box

```
@Preview(showBackground = true)
@Composable
fun GreetingPreview() {
        Greeting()
}
```

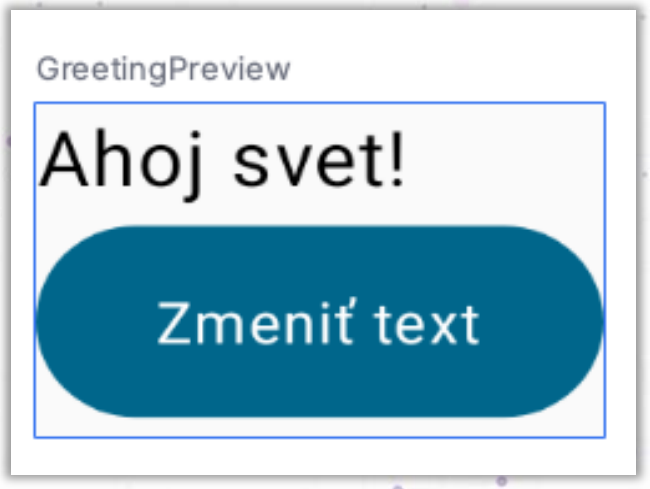



```
@Preview(showBackground = true)
@Composable
fun GreetingPreview() {
    P3Theme {
        Greeting()
    }
}
```



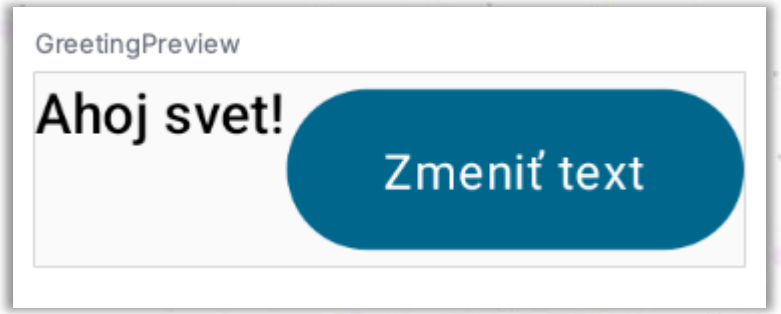
# Compose – Layouts – Column

```
@Composable
fun Greeting() {
    var text by remember { mutableStateOf( value: "Ahoj svet!") }
    Column { this: ColumnScope
        Text(text = text, style = MaterialTheme.typography.bodyLarge)
        Button(onClick = { text = "Zmenený text" }) { this: RowScope
            Text( text: "Zmeniť text", style = MaterialTheme.typography.bodySmall)
        }
    }
}
```



# Compose – Layouts – Row

```
@Composable
fun Greeting() {
    var text by remember { mutableStateOf( value: "Ahoj svet!") }
    Row { this: RowScope
        Text(text = text, style = MaterialTheme.typography.titleSmall)
        Button(onClick = { text = "Zmenený text" }) { this: RowScope
            Text( text: "Zmeniť text", style = MaterialTheme.typography.bodySmall)
        }
    }
}
```



# Compose – Layouts – Iné

- Image
- Icon
- Card
- Divider
- Space
- LazyRow

# Compose – Layouts – Collection

```
@Composable
fun SimpleLazyColumn() {
    val itemList = listOf("Položka 1", "Položka 2", "Položka 3")
    LazyColumn { this: LazyListScope
        items(itemList) { this: LazyItemScope item ->
            Text(text = item)
        }
    }
}
```

```
@Composable
fun LazyColumnWithIndex() {
    val itemList = listOf("Položka 1", "Položka 2", "Položka 3")

    LazyColumn { this: LazyListScope
        itemsIndexed(itemList) { this: LazyItemScope index, item ->
            Text(text = "Položka $index: $item")
        }
    }
}
```

LazyColumn

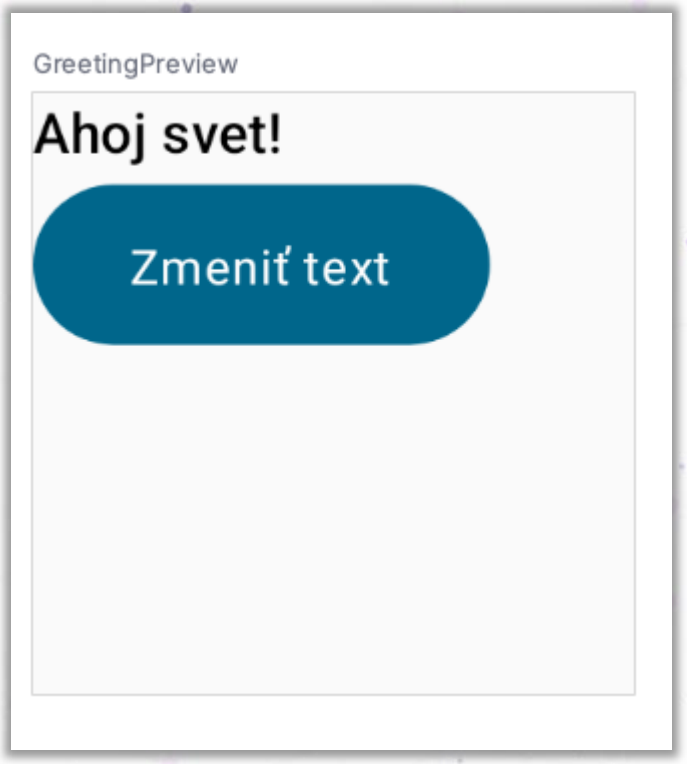
Položka 1 Položka 0: Položka 1
Položka 2 Položka 1: Položka 2
Položka 3 Položka 2: Položka 3

```
inline fun <T> LazyListScope.items(
    items: List<T>,
    noinline key: ((item: T) -> Any)? = null,
    noinline contentType: (item: T) -> Any? = { null },
    crossinline itemContent: @Composable LazyItemScope.(item: T) -> Unit
) = items(
```

```
inline fun <T> LazyListScope.itemsIndexed(
    items: List<T>,
    noinline key: ((index: Int, item: T) -> Any)? = null,
    crossinline contentType: (index: Int, item: T) -> Any? = { _, _ -> null },
    crossinline itemContent: @Composable LazyItemScope.(index: Int, item: T) -> Unit
) = items(
```

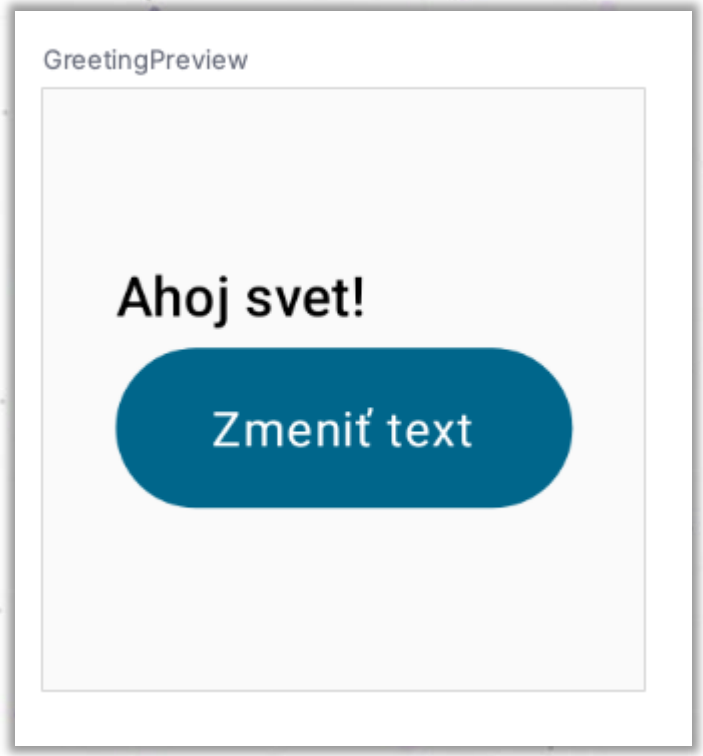
# Compose – Layouts – Modifier

```
@Composable
fun Greeting() {
    var text by remember { mutableStateOf( value: "Ahoj svet!") }
    Box(Modifier.size(150.dp)) { this: BoxScope
        Column { this: ColumnScope
            Text(text = text, style = MaterialTheme.typography.titleSmall)
            Button(onClick = { text = "Zmenený text" }) { this: RowScope
                Text( text: "Zmeniť text", style = MaterialTheme.typography.bodySmall)
            }
        }
    }
}
```



# Compose – Layouts – Modifier – Center

```
@Composable
fun Greeting() {
    var text by remember { mutableStateOf( value: "Ahoj svet!") }
    Box(modifier = Modifier.size(150.dp)) { this: BoxScope
        Column(modifier = Modifier.align(Alignment.Center)) { this: ColumnScope
            Text(text = text, style = MaterialTheme.typography.titleSmall)
            Button(onClick = { text = "Zmenený text" }) { this: RowScope
                Text( text: "Zmeniť text", style = MaterialTheme.typography.bodySmall)
            }
        }
    }
}
```

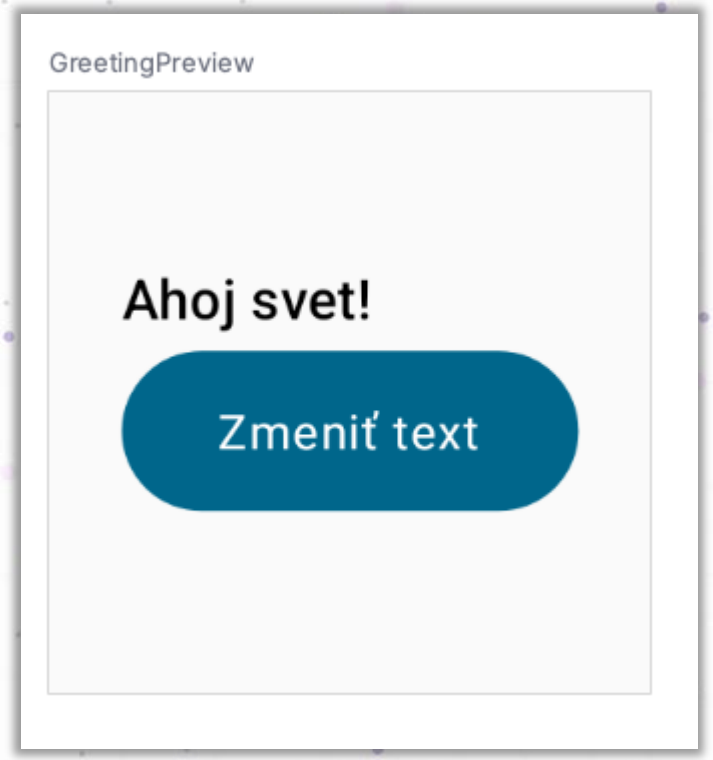


# Compose – Layouts – Modifier – Center

```
@Composable
fun Greeting() {
    var text by remember { mutableStateOf( value: "Ahoj svet!") }
    Box(modifier = Modifier.size(150.dp)) { this: BoxScope
        Column(modifier = Modifier.align(Alignment.Center)) { this: ColumnScope
            Text(text = text, style = MaterialTheme.typography.titleSmall)
            Button(onClick = { text = "Zmenený text" }) { this: RowScope
                Text( text: "Zmeniť text", style = MaterialTheme.typography.bodySmall)
            }
        }
    }
}
```

```
n(Alignment.Center)) { this: ColumnScope
aterialTh
Zmenený t
, style =
```

- BottomCenter (androidx.compose.ui) Alignment
- BottomEnd (androidx.compose.ui) Alignment
- BottomStart (androidx.compose.ui) Alignment
- Center (androidx.compose.ui) Alignment
- CenterEnd (androidx.compose.ui) Alignment
- CenterStart (androidx.compose.ui) Alignment
- TopCenter (androidx.compose.ui) Alignment
- TopEnd (androidx.compose.ui) Alignment
- TopStart (androidx.compose.ui) Alignment



# Compose – Layouts – Modifier – Detail

```
@Composable
fun Text(
    text: String,
    modifier: Modifier = Modifier,
    color: Color = Color.Unspecified,
    fontSize: TextUnit = TextUnit.Unspecified,
    fontStyle: FontStyle? = null,
    fontWeight: FontWeight? = null,
    fontFamily: FontFamily? = null,
    letterSpacing: TextUnit = TextUnit.Unspecified,
    textDecoration: TextDecoration? = null,
    textAlign: TextAlign? = null,
    lineHeight: TextUnit = TextUnit.Unspecified,
    overflow: TextOverflow = TextOverflow.Clip,
    softWrap: Boolean = true,
    maxLines: Int = Int.MAX_VALUE,
    minLines: Int = 1,
    onTextLayout: (TextLayoutResult) -> Unit = {},
    style: TextStyle = LocalTextStyle.current
) {
```

```
@Composable
fun Button(
    onClick: () -> Unit,
    modifier: Modifier = Modifier,
    enabled: Boolean = true,
    shape: Shape = ButtonDefaults.shape,
    colors: ButtonColors = ButtonDefaults.buttonColors(),
    elevation: ButtonElevation? = ButtonDefaults.buttonElevation(),
    border: BorderStroke? = null,
    contentPadding: PaddingValues = ButtonDefaults.ContentPadding,
    interactionSource: MutableInteractionSource = remember { MutableInteractionSource() },
    content: @Composable RowScope.() -> Unit
) {
```

# Compose – Layouts – Animácie

```
@Composable
fun SimpleAnimationExample() {
    // Stav, ktorý sleduje, či je box rozšírený
    var isExpanded by remember { mutableStateOf( value: false) }
    // Animovaná hodnota veľkosti, ktorá sa mení podľa stavu isExpanded
    val size by animateDpAsState(targetValue = if (isExpanded) 200.dp else 150.dp)
    Column { this: ColumnScope
        Button(onClick = { isExpanded = !isExpanded }) { this: RowScope
            Text( text: "Animuj")
        }
        Box(
            modifier = Modifier
                .size(size)
                .background(Color.Green)
        ) { this: BoxScope
            Greeting(modifier = Modifier.align(Alignment.Center))
        }
    }
}
```

# Compose – Layouts – Vysvetlivky

Fire-and-forget animation function for `Dp`. This Composable function is overloaded for different parameter types such as `Float`, `Color`, `Offset`, etc. When the provided `targetValue` is changed, the animation will run automatically. If there is already an animation in-flight when `targetValue` changes, the on-going animation will adjust course to animate towards the new target value.

`animateDpAsState` returns a `State` object. The value of the state object will continuously be updated by the animation until the animation finishes.

Note, `animateDpAsState` cannot be canceled/stopped without removing this composable function from the tree. See `Animatable` for cancelable animations.

Params: `targetValue` - Target value of the animation
`animationSpec` - The animation that will be used to change the value through time. Physics animation will be used by default.
`label` - An optional label to differentiate from other animations in Android Studio.
`finishedListener` - An optional end listener to get notified when the animation is finished.

Returns: A `State` object, the value of which is updated by animation.

Samples: `androidx.compose.animation.core.samples.DpAnimationSample`

```
// Unresolved
```

```
@Composable
fun animateDpAsState(
    targetValue: Dp,
    animationSpec: AnimationSpec<Dp> = dpDefaultSpring,
    label: String = "DpAnimation",
    finishedListener: ((Dp) -> Unit)? = null
): State<Dp> {
```

# UI

**Material Design 3:**

- Známy aj ako Material You, je najnovšia aktualizácia dizajnového systému od Google, ktorá prináša personalizáciu, expresivitu a adaptabilitu do popredia.

**Personalizácia:**

- Material You umožňuje aplikáciám prispôsobiť vzhľad a pocit na základe používateľských preferencií a témy zariadenia, vrátane dynamických farieb, ktoré sa menia podľa tapety používateľa.

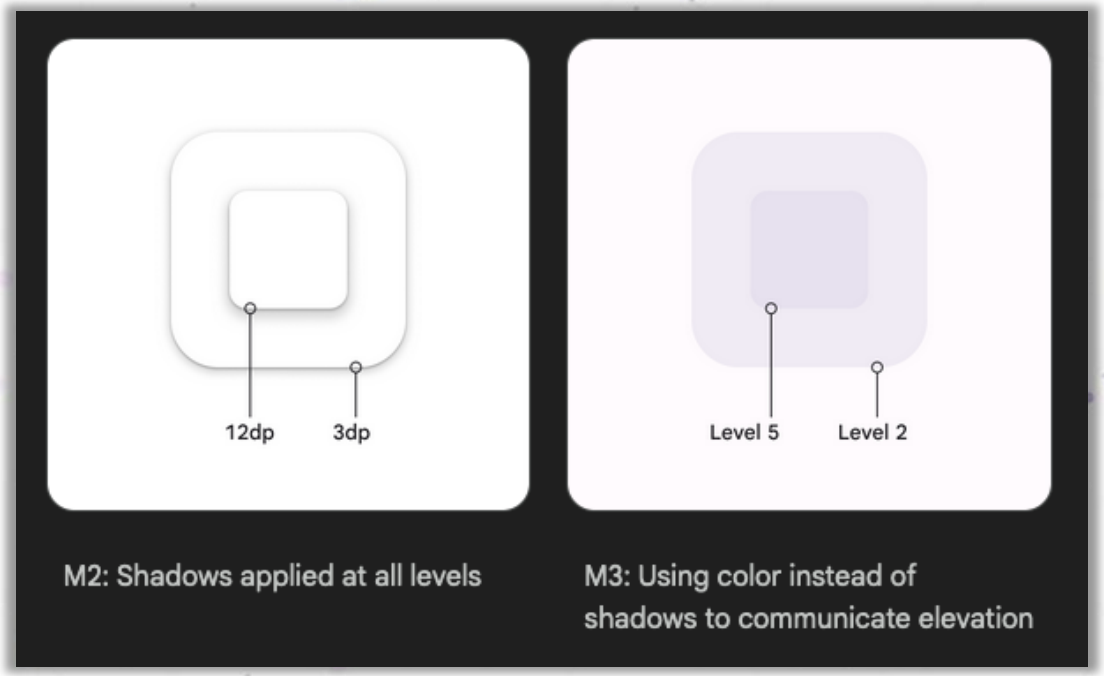
**Expresivita a Adaptabilita:**

- Nové komponenty a štýly poskytujú väčšiu flexibilitu a umožňujú aplikáciám vyjadriť jedinečnú značku a estetiku, zatiaľ čo súčasne zabezpečujú prístupnosť a koherentnosť naprieč zariadeniami a platformami.

**Nové Komponenty a Funkcie:**

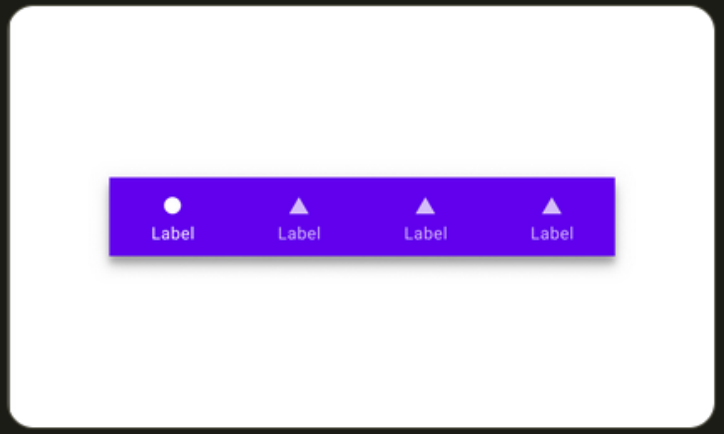
- Predstavenie nových komponentov, ako sú tlačidlá s variabilnou šírkou, karty s lepším zvýraznením obsahu a rozšírené možnosti pre navigačné lišty a menu.

# Material Design 3

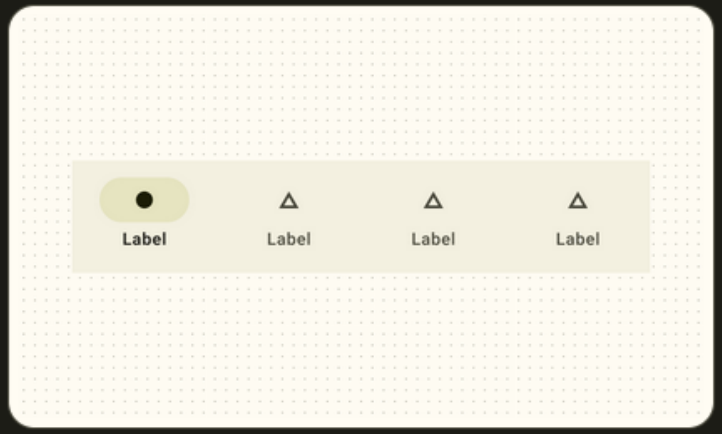



M2: Shadows applied at all levels

M3: Using color instead of shadows to communicate elevation




M2: The navigation bar is positioned above page content, as indicated by a drop shadow. Filled and regular weight icons indicate active states; regular outline icons indicate inactive states.




M3: Active states are represented with filled icons and a contrasting pill-shaped active indicator, while inactive states are represented with outlined icons. The height of the navigation bar is taller.

# Compose – Modifier

**Základné informácie:**

- **Modifier** je nástroj v Jetpack Compose, ktorý umožňuje deklaratívne pridávať vlastnosti ako veľkosť, ohraničenie, padding, margin, klikateľnosť a mnoho ďalších k Composable funkciám.

**Reťazenie:**

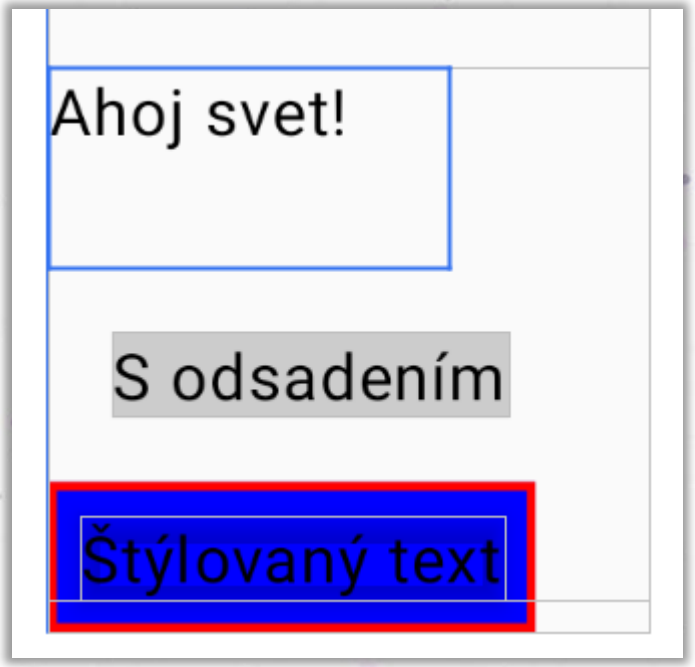
- Jednou z hlavných výhod **Modifier** je možnosť reťazenia viacerých modifikátorov pomocou . syntaxe, čo umožňuje kombinovať viacero vlastností v jednej linke kódu.

# Compose – Modifier

```
Text(
    text = "Ahoj svet!",
    modifier = Modifier.size(width = 100.dp, height = 50.dp)
)
```

```
Box(
    modifier = Modifier
        .background(Color.Blue)
        .border(2.dp, Color.Red)
        .padding(8.dp)
        .shadow(4.dp)
) { this: BoxScope
    Text(text = "Štýlovaný text")
}
```

```
Box(
    modifier = Modifier
        .padding(16.dp)
        .background(Color.LightGray)
        .clickable { /* Akcia po kliknutí */ }
) { this: BoxScope
    Text(text = "S odsadením")
}
```



# Compose – Modifier – Zhrnutie

**Konzistentné použitie:**

- Snažte se používať **Modifier** konzistentne v celej aplikácii pre jednotný vzhľad a správanie UI.

**Reťazenie s rozmyslom:**

- Hoci **Modifier** umožňuje reťazenie, je dôležité robiť to premyslene, aby ste predišli zbytočnej komplexite a zachovali čitateľnosť kódu.

**Modifier** je mocný nástroj v Jetpack Compose, ktorý vám dáva veľkú flexibilitu pri definovaní vzhľadu a správania vašich UI komponentov. Jeho efektívne využitie môže výrazne prispieť k efektívnemu a udržateľnému vývoju aplikácií.

# Compose – State – remember

**ŕemember** sa používa na "zapamätanie" hodnoty alebo objektu počas rekompozícií. To znamená, že hodnota uložená v **remember** sa nevytvorí znova pri každej rekompozícii, pokiaľ sa nezmení závislosť, na ktorej je **remember** založené.

**Použitie:**

- Typicky sa používa na uchovávanie stavu UI komponentov, ako sú text vstupného poľa, pozícia skrolovania alebo akýkoľvek iný stav, ktorý by mal byť zachovaný medzi rekompozíciami.

# Compose – State – remember

**Optimalizácia výkonu:**

- **remember** pomáha optimalizovať výkon tým, že znižuje potrebu opätovného vytvárania objektov alebo výpočtov pri každej rekompozícii.

**Závislosti:**

- Hodnota v **remember** sa resetuje len vtedy, ak sa zmenia závislosti, ktoré sú jej súčasťou. Môžete explicitne určiť závislosti pomocou `remember(key1, key2, ...) { ... }`.

**Bezpečnosť:**

- Keďže **remember** uchováva dáta len počas životného cyklu Composable funkcie, mali by ste byť opatrní pri uchovávaní citlivých alebo veľkých dát, ktoré by mohli ovplyvniť výkon alebo bezpečnosť.

# Compose – State – remember

```
@Composable
fun RememberExample() {
    // Použitie remember na uchovanie stavu textového poľa
    val textState = remember { mutableStateOf( value: "") }

    Column { this: ColumnScope
        TextField(
            value = textState.value,
            onValueChange = { textState.value = it },
            label = { Text( text: "Zadajte text") }
        )
        Button(
            onClick = { /* Akcia, ktorá môže spôsobiť rekompozíciu */ }
        ) { this: RowScope
            Text( text: "Kliknite")
        }
    }
}
```

# Compose – State – rememberSaveable

**rememberSaveable** je funkcia v Jetpack Compose, ktorá rozširuje koncept remember tým, že umožňuje uchovávať stav cez procesové hranice, ako sú napríklad rekonfigurácie obrazovky pri otočení zariadenia. Táto funkcia je obzvlášť užitočná na uchovávanie malých kúskov dát, ako sú užívateľské vstupy alebo výbery, ktoré by mali byť zachované aj po zmenách konfigurácie alebo po reštarte aktivít.

Použitie **rememberSaveable** pre dáta, ktoré by mali byť zachované pri rekonfigurácii alebo reštarte UI, ako sú užívateľské vstupy alebo výbery.
Vhodné pre uchovávanie malých kúskov dát; pre veľké objekty alebo komplexné stavy by ste mali zvážiť iné riešenia, ako sú ViewModely alebo databázy.

**rememberSaveable** je mocný nástroj na zlepšenie užívateľskej skúsenosti tým, že zabezpečuje kontinuitu užívateľského rozhrania naprieč zmenami konfigurácie, čím sa znižuje frustrácia užívateľov zo straty dát.

# Compose – State – RememberSaveable

```
@Composable
fun RememberSaveableExample() {
    // Použitie rememberSaveable na uchovanie stavu
    // textového poľa cez rekonfigurácie
    val textState = rememberSaveable { mutableStateOf( value: "") }

    TextField(
        value = textState.value,
        onValueChange = { textState.value = it },
        label = { Text( text: "Zadajte text") }
    )
}
```

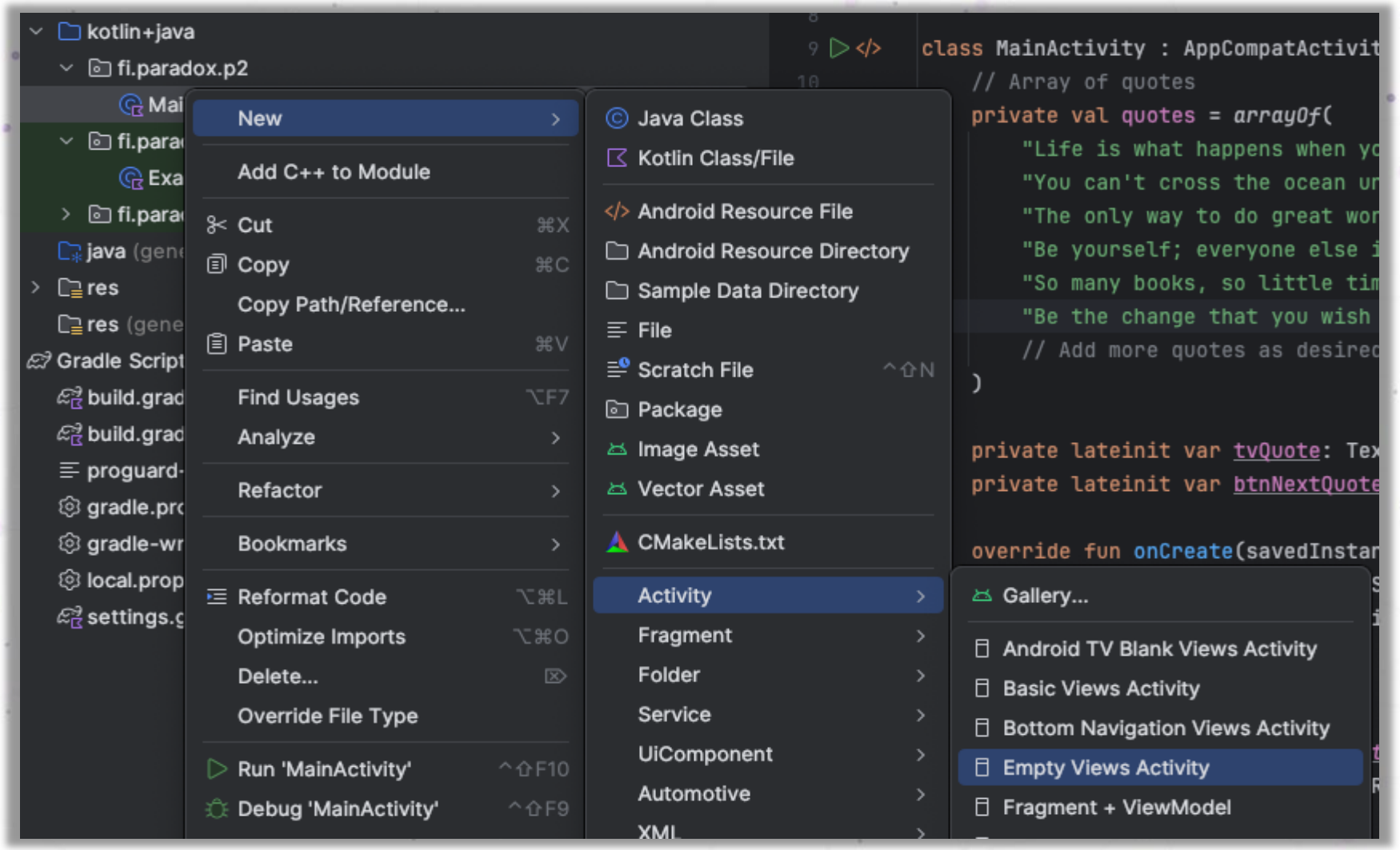
# remember × rememberSaveable

**remember** uchováva dáta len počas životného cyklu aktuálnej kompozície. Ak sa aktivita alebo fragment reštartuje (napríklad pri otočení zariadenia), stav uchovaný pomocou **remember** sa stratí a bude re-inicializovaný

×

**rememberSaveable** uchováva dáta cez reštarty aktivít alebo fragmentov tým, že automaticky ukladá stav do Bundle a obnovuje ho po rekonfigurácii. **rememberSaveable** používa internú logiku na serializáciu a deserializáciu stavu, aby bol stav zachovaný aj po zmene konfigurácie.

# Aktivita – Vytvorenie aktivity



# Aktivita – AndroidManifest

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="P2"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/Theme.P"
    tools:targetApi="31">
    <activity
        android:name=".MainActivity"
        android:exported="true"
        android:label="P2"
        android:theme="@style/Theme.P">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>

    <activity android:name=".MainActivity2"/>
</application>
```

# Aktivita – Intent

**implicitne**

×

**explicitne**

```
val intent = Intent( packageContext: this, MainActivity2::class.java)
startActivity(intent)
```

```
class MainActivity2 : AppCompatActivity() {
    companion object {
        fun startActivity(context: Context) {
            val intent = Intent(context, MainActivity2::class.java)
            context.startActivity(intent)
        }
    }
```

```
fun openUri(context: Context) {
    val intent = Intent(Intent.ACTION_VIEW, Uri.parse( uriString: "https://www.example.com"))
    context.startActivity(intent)
}
```

# Aktivita – Prenos dát

Možnosť prečítať vo ViewModel

## Primitivne typy a ich pole:

```
putExtra(String name, boolean value)
putExtra(String name, byte value)
putExtra(String name, char value)
putExtra(String name, short value)
putExtra(String name, int value)
putExtra(String name, long value)
putExtra(String name, float value)
putExtra(String name, double value)
```

```
fun startActivity(context: Context, detailId: String) {
    val intent = Intent(context, MainActivity2::class.java)
    intent.putExtra(KEY, detailId)
    context.startActivity(intent)
}
```

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    val detailId = intent.getStringExtra(KEY)
```

A ich pole ekvivalenty, napríklad: `putExtra(String name, boolean[] value)`, `putExtra(String name, int[] value)`, atď.

# Aktivita – Prenos dát

Možnosť prečítať vo ViewModel

## Špeciálne typy:

putExtra(String name, Bundle value)

- pre Bundle, ktorý môže obsahovať zložitejšie dátové štruktúry.

putExtra(String name, Parcelable value)

- pre objekty, ktoré implementujú Parcelable rozhranie, umožňujúce komplexnejšie objekty byť prenášané medzi komponentami.

putExtra(String name, Serializable value)

- pre objekty, ktoré implementujú Serializable rozhranie, hoci použitie Serializable je **menej** efektívne ako **Parcelable**.

# Aktivita – StartActivityForResult

```
val resultLauncher = registerForActivityResult(ActivityResultContracts.StartActivityForResult()) { result ->
    if (result.resultCode == Activity.RESULT_OK) {
        // Spracovanie výsledku
        val data:Intent? = result.data
        val resultValue = data?.getStringExtra( name: "resultKey")
    }
}
```

```
val intent = Intent( packageContext: this, MainActivity2::class.java)
resultLauncher.launch(intent)
```

```
//        ukončenie MainActivity2
val returnIntent = Intent()
returnIntent.putExtra( name: "resultKey",  value: "resultValue")
setResult(Activity.RESULT_OK, returnIntent)
finish()
```

# Zdroje


- Compose : https://www.jetpackcompose.net/themes-in-jetpack-compose
- Manage state in Jetpack Compose: https://developer.android.com/jetpack/compose/state
- Activities : https://developer.android.com/guide/components/activities/intro-activities
- Animations in Jetpack Compose: https://developer.android.com/jetpack/compose/animation
- MATERIAL DESIGN 3 : https://m3.material.io/

# Kotlin Coroutine

- Coroutines sa spúšťajú v tzv. "scopes", ktoré určujú kontext, v ktorom korutína beží. Najčastejšie používané sú GlobalScope, lifecycleScope (pre aktivity a fragmenty) a viewModelScope (pre ViewModel).

```
lifecycleScope.launch(Dispatchers.IO) { this: CoroutineScope
    // Vykonajte asynchrónnu operáciu
    val data = fetchData()
    // Aktualizujte UI s novými dátami
    uiState.update { currentState ->
        currentState.copy(/* aktualizované dáta */)
    }
}
```

# ViewModel

```
class MyViewModel : ViewModel() {
    private val _uiState = MutableStateFlow(/* inicializácia stavu */)
    val uiState: StateFlow</* typ stavu */> = _uiState.asStateFlow()

    fun fetchData() {
        viewModelScope.launch(Dispatchers.IO) { this: CoroutineScope
            val data = fetchData()
            _uiState.update { currentState ->
                currentState.copy(/* aktualizované dáta */)
            }
        }
    }
}
```

ViewModel je komponenta architektúry Android, ktorá je navrhnutá na uchovávanie a správu UI-súvislých dát v životnom cykle, ktorý je dlhší ako životný cyklus aktivity alebo fragmentu.

implementation "org.jetbrains.kotlinx:kotlinx-coroutines-core:1.3.9"
implementation "org.jetbrains.kotlinx:kotlinx-coroutines-android:1.3.9"

```
suspend fun fetchData(): Data {
    // Simulácia dlhotrvajúcej operácie
    delay( timeMillis: 1000)
    return Data( value: "data")
}
```

- suspend funkcie sú základným stavebným kameňom korutín a umožňujú asynchrónne operácie bez blokovania vlákna.
- Správa chýb v korutínach môže byť realizovaná pomocou try-catch blokov alebo štruktúr ako CoroutineExceptionHandler.
- ViewModel je komponenta architektúry Android, ktorá pomáha spravovať UI-súvislé dáta v kontexte životného cyklu aktivity alebo fragmentu.

# ViewModel a aktivita

```
// Získanie inštancie ViewModelu
val myViewModel = ViewModelProvider( owner: this).get(MyViewModel::class.java)
```

## LiveData

```
myViewModel.uiState.observe(this) { uiState ->
    // Aktualizujte vaše Views podľa novej hodnoty uiState
}
```

```
lifecycleScope.launchWhenStarted { this: CoroutineScope
    myViewModel.uiState.collect { uiState ->
        // Aktualizujte vaše Views podľa novej hodnoty uiState
    }
}
```

## StateFlow

```
val uiState = myViewModel.uiState.collectAsState()
```