배달의민족에 Jetpack Compose 적용을 시도해보았다.

우아콘 WOOWA CON 2021

강경완
배민Android개발팀

들어가기전에

- Jetpack Compose 도입 시도해본 경험을 공유
- 적용기 테스트기
- Jetpack Compose 도입을 해볼까..?
- 어떤 문제가 있을까..?

Jetpack Compose is now 1.0

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

Let’s test Jetpack Compose

Jetpack Compose

1. 코드 감소
2. 직관적
3. 빠른 개발과정
4. 강력한 성능

명령형 프로그래밍

```
val layout = LinearLayout(context)
layout.setBackgroundColor(red)
layout.removeAllViews()
val childTextView = TextView(context)
childTextView.text = "childTextView"
layout.addView(childTextView)
```

선언형 프로그래밍

```
Column(
  modifier = Modifier
    .background(red)
) {
  Text("childText")
}
```

현재 배민앱

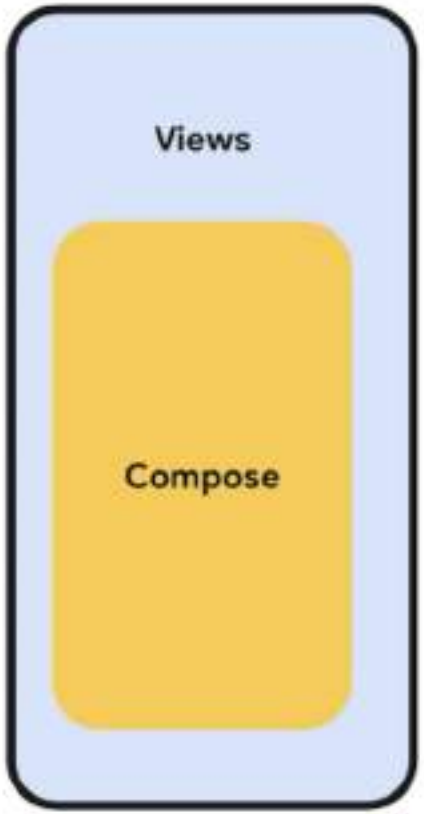
MVP 패턴 사용중

권장사항

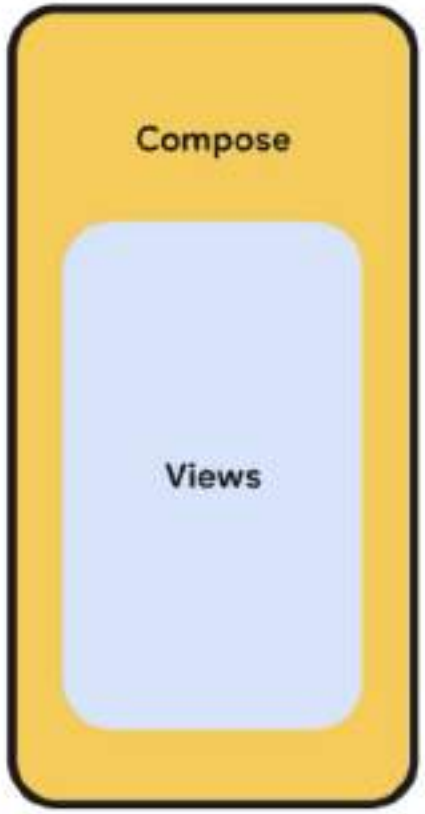
MVVM 등 단방향 데이터 흐름 아키텍처 권장
targetSdkVersion 30
Gradle 7.0
Kotlin Version 1.5.21
minSdkVersion 21

* Using Android Studio Arctic Fox

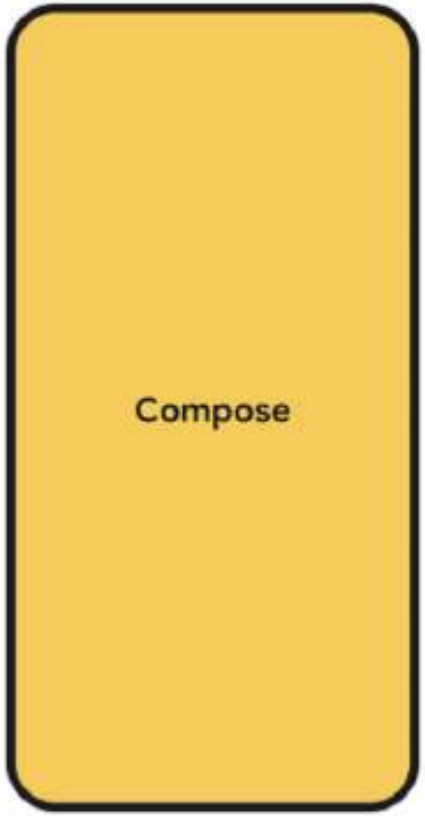
Compose로 앱을 만드는 방법



기존 View에
Compose 요소추가



새 화면을
Compose로 생성



처음부터
Compose 로 구성

TextView -> Compose

```
<LinearLayout
  android:id="@+id/locationLayout"
  ... >
  <TextView
      android:id="@+id/locationTextView"
      android:layout_width="wrap_content"
      android:layout_height="match_parent"
      android:textColor="@color/text_white"
      tools:text="올림픽로 295" />

  <ImageView
      ... />
</LinearLayout>
```

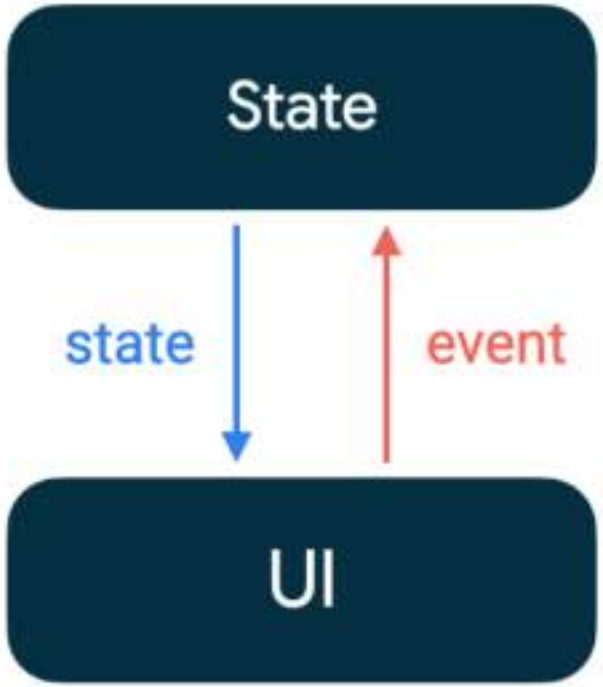
TextView -> Compose

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

  <ImageView
      ... />
</LinearLayout>
```

TextView -> Compose

```
binding?.locationTextComposeView?.setContent {
  LocationText(viewModel.getTitle())
}

@Composable
fun LocationText(title: LiveData<String>) {
  val text = title.observeAsState()

  title.value?.let {
    Text(
      text = it,
      fontSize = dpToSp(dp = 14.dp),
      fontWeight = FontWeight.Bold,
      color = colorResource(id = R.color.text…),
      textAlign = TextAlign.Center,
      overflow = TextOverflow.Ellipsis,
      ...
    )
  }
}
```

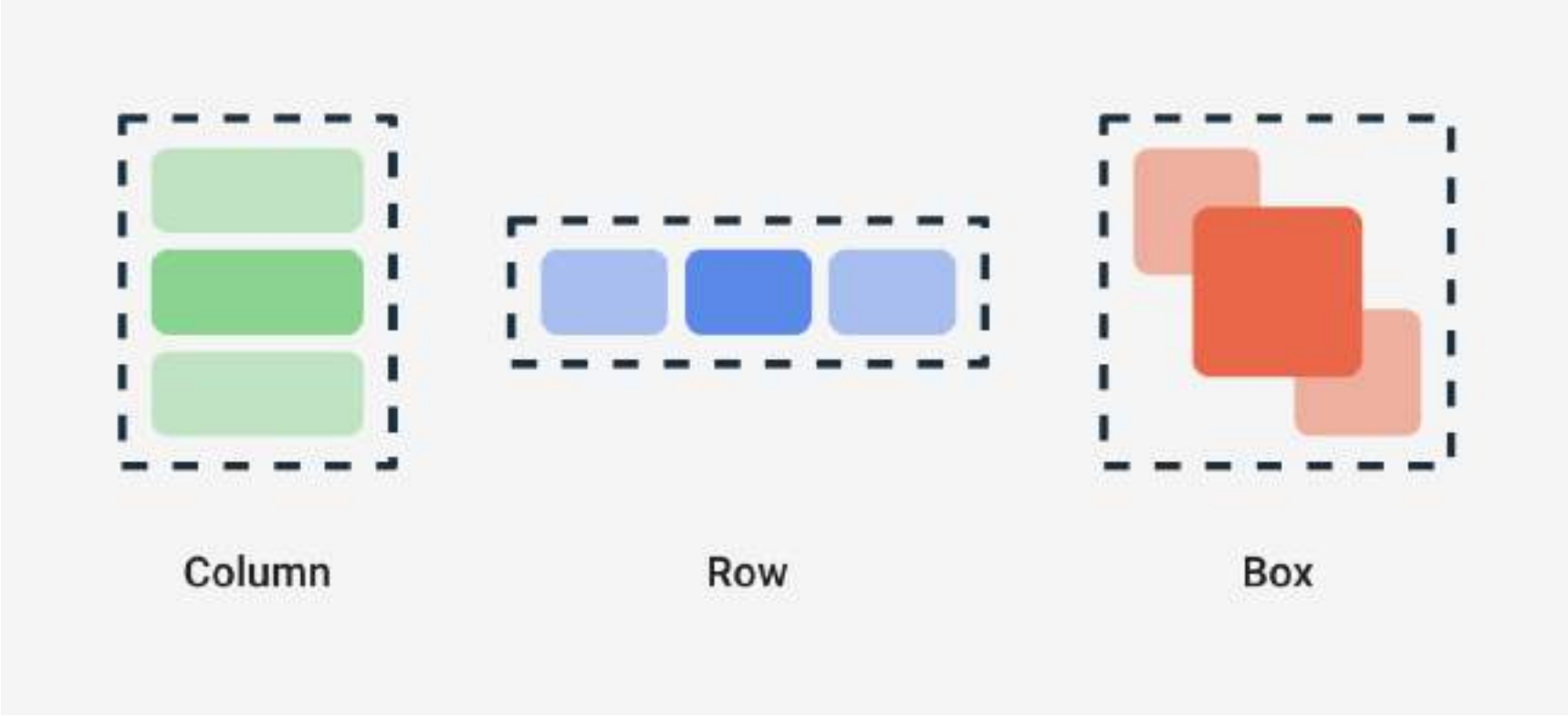
단방향 데이터 흐름



MVVM 등 단방향 데이터 흐름 아키텍처 권장

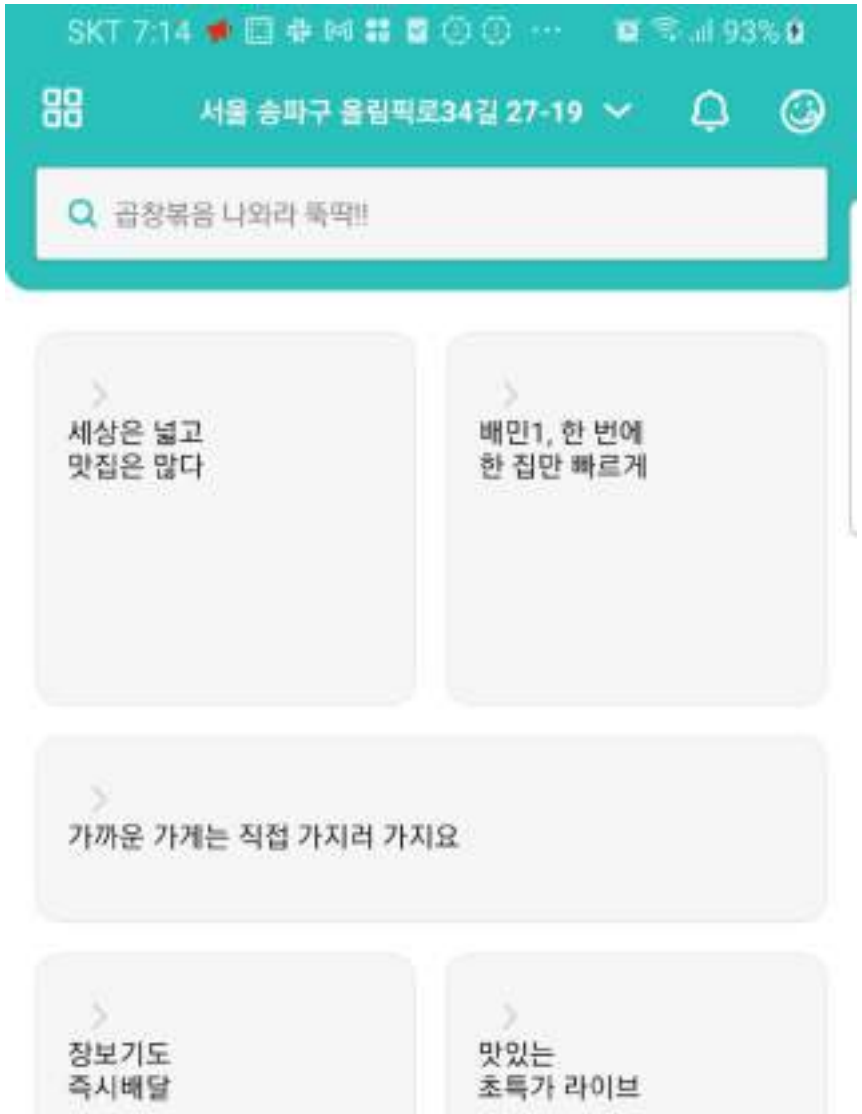
State<T> 가 변경될 때 ReComposition

Compose Basic Layout



Layout -> Compose

```
@Composable
fun LocationLayout(title: LiveData<String>) {
  Row(
    modifier = Modifier.fillMaxSize(),
    verticalAlignment = Alignment.CenterVertically,
  ) {
    LocationText(text = title)
    Spacer(modifier = Modifier.size(4.dp))
    LocationIconImage()
  }
}

@Composable
fun LocationIconImage() {
  Image(
    painter = painterResource(R.drawable.icon_open),
    ...
  )
}
```



RecyclerView -> Compose

LazyListScope DSL

`LazyListScope` 의 DSL은 레이아웃의 항목을 설명하는 여러 함수를 제공합니다. 가장 기본적인 `item()` 함수는 단일 항목을 추가하고 `items(Int)` 는 여러 항목을 추가합니다.

```
LazyColumn {
    // Add a single item
    item {
        Text(text = "First item")
    }

    // Add 5 items
    items(5) { index ->
        Text(text = "Item: $index")
    }

    // Add another single item
    item {
        Text(text = "Last item")
    }
}
```

RecyclerView -> Compose

Jetpack Compose는 `DisposeOnDetachedFromWindow` 를 기본 `ViewCompositionStrategy` 로 사용합니다. 즉, 뷰가 창에서 분리될 때마다 Composition이 삭제됩니다.

`ComposeView` 를 `RecyclerView` 뷰 홀더의 일부로 사용할 경우, `RecyclerView` 가 창에서 분리될 때까지 기본 Composition 인스턴스가 메모리에 남아 있기 때문에 기본 전략은 비효율적입니다. `RecyclerView` 가 더 이상 `ComposeView` 에 필요하지 않은 경우 기본 Composition을 삭제하는 것이 좋습니다.

`disposeComposition` 함수를 사용하면 `ComposeView` 의 기본 Composition을 수동으로 삭제할 수 있습니다. 다음과 같이 뷰가 재활용될 때 이 함수를 호출할 수 있습니다.

```
import androidx.compose.ui.platform.ComposeView

class MyComposeAdapter : RecyclerView.Adapter<MyComposeViewHolder>() {

    override fun onCreateViewHolder(
        parent: ViewGroup,
        viewType: Int,
    ): MyComposeViewHolder {
        return MyComposeViewHolder(ComposeView(parent.context))
    }

    override fun onViewRecycled(holder: MyComposeViewHolder) {
        // Dispose of the underlying Composition of the ComposeView
        // when RecyclerView has recycled this ViewHolder
        holder.composeView.disposeComposition()
    }

    /* Other methods */
}

    val composeView: ComposeView
) : RecyclerView.ViewHolder(composeView) {
    /* ... */
```

https://developer.android.com/jetpack/compose/interop/compose-in-existing-ui#compose-recyclerview

RecyclerView -> Compose


```
abstract class ComposeListAdapterDelegate
  <T: DisplayModel, VH: ComposeViewHolder<T>>
  : AbsListItemAdapterDelegate<T,
DisplayModel, ComposeViewHolder<T>>() {

  override fun onViewRecycled(
    holder: RecyclerView.ViewHolder) {
      (holder as VH)?.let {
          it.composeView.disposeComposition()
      }
      super.onViewRecycled(holder)
  }
}
```

```
abstract class ComposeViewHolder<T>(
    val composeView: ComposeView
) : RecyclerView.ViewHolder(composeView) {

  @Composable
  abstract fun ViewHolder(input: T)

  init {
    composeView.setViewCompositionStrategy(
      ViewCompositionStrategy.DisposeOnViewTreeLi
      fecycleDestroyed)
  }

  fun bindViewHolder(input: T) {
    composeView.setContent {
      ViewHolder(input)
    }
  }
}
```

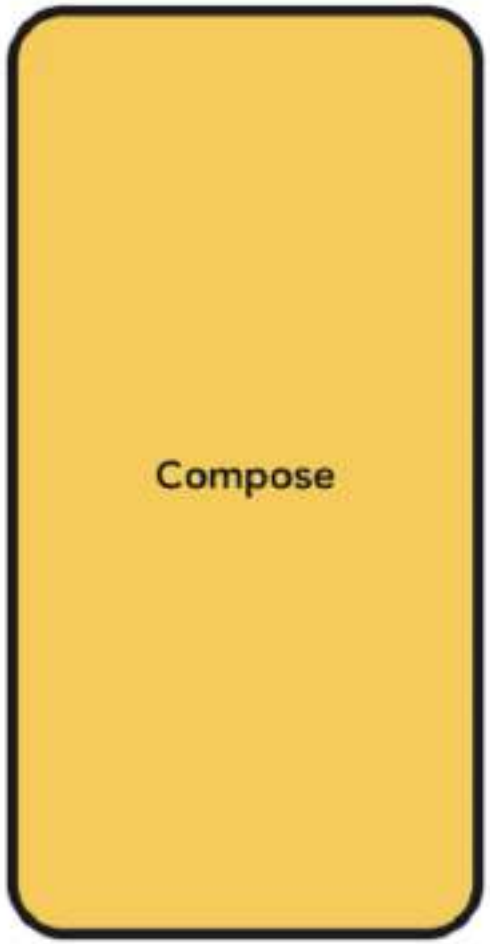
RecyclerView -> Compose


```
class GatewayAdapterDelegate: ComposeListAdapterDelegate<GatewayDisplayModel,
GatewayAdapterDelegate.GatewayViewHolder>() {

    class GatewayBusinessInfoViewHolder(
        composeView: ComposeView
    ) : ComposeViewHolder<GatewayDisplayModel>(composeView) {

        @Composable
        override fun ViewHolder(input: GatewayBusinessInfoDisplayModel) {
            ... // View
        }
    }

    override fun onCreateViewHolder(parent: ViewGroup): ComposeViewHolder<GatewayDisplayModel> {
        return GatewayViewHolder(ComposeView(parent.context))
    }

    override fun isForViewType(item: DisplayModel, items: MutableList<DisplayModel>, position: Int): Boolean {
        return item is GatewayDisplayModel
    }

    override fun onBindViewHolder(
        item: GatewayDisplayModel,
        holder: ComposeViewHolder<GatewayDisplayModel>,
        payloads: MutableList<Any>
    ) {
        holder.bindViewHolder(item)
    }
}
```

RecyclerView -> Compose


```
private fun initAdapter() = ListDelegationAdapter(
        gatewaySearchAdapterDelegate(
            clickListener = {
                presenter.onSearchClicked()
            }
        ),
        ...
        GatewayComposeAdapterDelegate(),
    )
```

RecyclerView -> Compose




```
@Composable
fun ShopList(models: LiveData<List<DisplayModel>>) {
    val displayModels = models.observeAsState()

    displayModels.value?.let {
        LazyColumn {
            itemsIndexed(it) { index, displayModel ->
                when (displayModel) {
                    is ShopModel -> Shop(displayModel)
                    is  ...
                }
            }
        }
    }
}
```

RecyclerView 첫 로딩

ViewHolder(RecyclerView)

Compose (LazyColumn)

Compose + ViewHolder

RecyclerView 첫 로딩

막대의 구성요소	렌더링 단계	설명
	버퍼 전환	GPU가 작업을 마칠 때까지 CPU가 대기하는 시간을 나타냅니다. 이 막대가 높아지면 앱이 GPU에서 너무 많은 작업을 하고 있다는 것을 의미합니다.
	명령어 사용	Android의 2D 렌더기기가 표시 목록을 그리거나 다시 그리기 위해 OpenGL에 명령을 내리는 데 소요되는 시간을 나타냅니다. 이 막대의 높이는 각 표시 목록을 실행하는 데 걸리는 시간의 합계에 정비례합니다. 표시 목록이 많을수록 빨간색 막대가 더 깁니다.
	동기화 및 업로드	비트맵 정보를 GPU에 업로드하는 데 걸리는 시간을 나타냅니다. 큰 세그먼트는 앱에서 많은 양의 그래픽을 로드하는 데 상당한 시간이 걸린다는 것을 나타냅니다.
	그리기	뷰의 표시 목록을 생성하고 업데이트하는 데 사용되는 시간을 나타냅니다. 막대의 이 부분이 높으면 onDraw 메서드에서 많은 맞춤 뷰 그리기 또는 많은 작업이 실행될 수 있습니다.
	측정/레이아웃	뷰 계층 구조의 onLayout 및 onMeasure 콜백에서 소요되는 시간을 나타냅니다. 큰 세그먼트는 뷰 계층 구조에서 작업을 처리하는 데 오랜 시간이 걸린다는 것을 나타냅니다.
	애니메이션	해당 프레임을 실행 중인 모든 애니메이터를 평가하는 데 걸린 시간을 나타냅니다. 이 세그먼트가 크면 앱에서 제대로 실행되지 않는 맞춤 애니메이터를 사용 중이거나 속성 업데이트의 결과 의도하지 않은 작업이 발생할 수 있습니다.
	입력 처리	앱이 입력 이벤트 콜백 내부에서 코드를 실행하는 데 소요된 시간을 나타냅니다. 이 세그먼트가 크면 앱에서 사용자 입력을 처리하는 데 시간이 너무 오래 걸린다는 것을 나타냅니다. 이러한 작업을 다른 스레드로 넘기는 것을 고려해보세요.
	기타 시간/VSync 지연	앱이 연속되는 두 프레임 사이에서 작업을 실행하는 데 소요되는 시간을 나타냅니다. 이는 UI 스레드에서 다른 스레드로 넘길 수 있는 너무 많은 작업을 처리 중임을 나타낼 수도 있습니다.

ViewHolde

iewHolder

RecyclerView 빠르게 스크롤

ViewHolder(RecyclerView)

Compose(LazyColumn)

Compose + ViewHolder

지도 화면을 전환해보았다.

배달의민족 주소 설정 지면

네이버 지도를 사용중

네이버 지도 SDK 에서

Compose 지원 X

AndroidView 로 감싸서 사용

지도 화면을 전환해보았다.

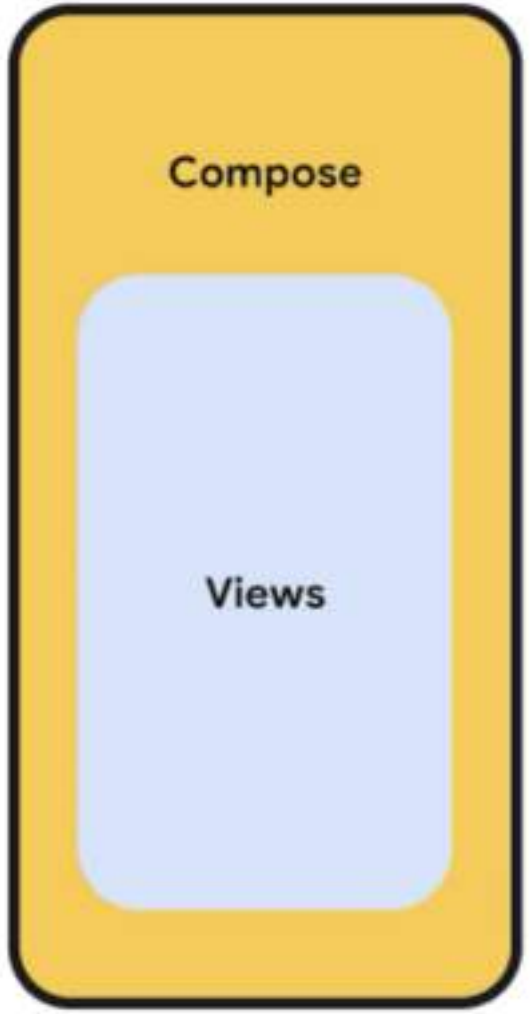
```
val coroutineScope = rememberCoroutineScope()
val savedInstanceState = rememberSavedInstanceState()
val mapView = rememberMapViewWithLifecycle(savedInstanceState)
AndroidView(
  factory = {
    mapView.apply {
      coroutineScope.launch {
        val naverMap = suspendCoroutine<NaverMap> { continuation ->
          getMapAsync {
            continuation.resume(it)
          }
        }

        naverMap.apply {
          minZoom = Constants.ZOOM_LEVEL_MIN.toDouble()
          maxZoom = Constants.ZOOM_LEVEL_MAX.toDouble()
          uiSettings.defaultSetting()
        }

        naverMap.addOnCameraIdleListener { onCameraIdle() }
        naverMap.addOnCameraChangeListener { i, b -> onCameraChange(i, b) }
        onMapReady(naverMap)
      }
    }
  }
)
```

코드참고 : https://github.com/Moop-App/Moop-Android/

지도 화면을 전환해보았다.

```
val coroutineScope = rememberCoroutineScope()
val savedInstanceState = rememberSavedInstanceState()
val mapView = rememberMapViewWithLifecycle(savedInstanceState)
AndroidView(
  factory = {
    mapView.apply {
      coroutineScope.launch {
        val naverMap = suspendCoroutine<NaverMap> { continuation ->
          getMapAsync {
            continuation.resume(it)
          }
        }

        naverMap.apply {
          minZoom = Constants.ZOOM_LEVEL_MIN.toDouble()
          maxZoom = Constants.ZOOM_LEVEL_MAX.toDouble()
          uiSettings.defaultSetting()
        }

        naverMap.addOnCameraIdleListener { onCameraIdle() }
        naverMap.addOnCameraChangeListener { i, b -> onCameraChange(i, b) }
        onMapReady(naverMap)
      }
    }
  }
)
```

코드참고 : https://github.com/Moop-App/Moop-Android/



지도 화면을 전환해보았다.

```
val coroutineScope = rememberCoroutineScope()
val savedInstanceState = rememberSavedInstanceState()
val mapView = rememberMapViewWithLifecycle(savedInstanceState)
AndroidView(
  factory = {
    mapView.apply {
      coroutineScope.launch {
        val naverMap = suspendCoroutine<NaverMap> { continuation ->
          getMapAsync {
            continuation.resume(it)
          }
        }

        naverMap.apply {
          minZoom = Constants.ZOOM_LEVEL_MIN.toDouble()
          maxZoom = Constants.ZOOM_LEVEL_MAX.toDouble()
          uiSettings.defaultSetting()
        }

        naverMap.addOnCameraIdleListener { onCameraIdle() }
        naverMap.addOnCameraChangeListener { i, b -> onCameraChange(i, b) }
        onMapReady(naverMap)
      }
    }
  }
)
```

코드참고 : https://github.com/Moop-App/Moop-Android/

지도 화면을 전환해보았다.

```
val coroutineScope = rememberCoroutineScope()
val savedInstanceState = rememberSavedInstanceState()
val mapView = rememberMapViewWithLifecycle(savedInstanceState)
AndroidView(
  factory = {
    mapView.apply {
      coroutineScope.launch {
        val naverMap = suspendCoroutine<NaverMap> { continuation ->
          getMapAsync {
            continuation.resume(it)
          }
        }

        naverMap.apply {
          minZoom = Constants.ZOOM_LEVEL_MIN.toDouble()
          maxZoom = Constants.ZOOM_LEVEL_MAX.toDouble()
          uiSettings.defaultSetting()
        }

        naverMap.addOnCameraIdleListener { onCameraIdle() }
        naverMap.addOnCameraChangeListener { i, b -> onCameraChange(i, b) }
        onMapReady(naverMap)
      }
    }
  }
)
```

코드참고 : https://github.com/Moop-App/Moop-Android/

지도 화면을 전환해보았다.

```
val coroutineScope = rememberCoroutineScope()
val savedInstanceState = rememberSavedInstanceState()
val mapView = rememberMapViewWithLifecycle(savedInstanceState)
AndroidView(
  factory = {
    mapView.apply {
      coroutineScope.launch {
        val naverMap = suspendCoroutine<NaverMap> { continuation ->
          getMapAsync {
            continuation.resume(it)
          }
        }

        naverMap.apply {
          minZoom = Constants.ZOOM_LEVEL_MIN.toDouble()
          maxZoom = Constants.ZOOM_LEVEL_MAX.toDouble()
          uiSettings.defaultSetting()
        }

        naverMap.addOnCameraIdleListener { onCameraIdle() }
        naverMap.addOnCameraChangeListener { i, b -> onCameraChange(i, b) }
        onMapReady(naverMap)
      }
    }
  }
)
```

코드참고 : https://github.com/Moop-App/Moop-Android/

지도 화면을 전환해보았다.

```
val lifecycle = LocalLifecycleOwner.current.lifecycle
DisposableEffect(lifecycle, mapView, savedInstanceState) {
    val lifecycleObserver = LifecycleEventObserver { _, event ->
        when (event) {
            Lifecycle.Event.ON_CREATE -> mapView.onCreate(
               savedInstanceState.takeUnless { it.isEmpty })
            Lifecycle.Event.ON_START -> mapView.onStart()
            Lifecycle.Event.ON_RESUME -> mapView.onResume()
            Lifecycle.Event.ON_PAUSE -> mapView.onPause()
            Lifecycle.Event.ON_STOP -> mapView.onStop()
            Lifecycle.Event.ON_DESTROY -> mapView.onDestroy()
            else -> throw IllegalStateException()
        }
    }

    lifecycle.addObserver(lifecycleObserver)
    onDispose {
        mapView.onSaveInstanceState(savedInstanceState)
        lifecycle.removeObserver(lifecycleObserver)
    }
}
```

코드참고 : https://github.com/Moop-App/Moop-Android/

LottieAnimation을 사용해보자

다양한 애니메이션으로 활용

4.2.0 부터 Compose 지원

장바구니 버튼을 바꾸어 보았다.

LottieView in View(기존 구조)

```
class FloatingCartView {

  fun updateCartCount() {
   getCartItemCountUseCase?.execute(onNext = { newCartCount ->
      val foodCount = cartCount.count
      val newCount = newCartCount.count
      val differenceCount = newCount - foodCount

      setCartCount(newCount)
      playAddAnimation(differenceCount)

      cartCount = newCartCount
    })
  }

  private fun playAddAnimation(differenceCount: Int) {
    if (floatingCartImageView.isAnimating) {
        floatingCartImageView.cancelAnimation()
    }
    floatingCartImageView.setAnimation("cart_ani.json")
    floatingCartImageView.playAnimation()
    animateCountPlusCount(differenceCount)
  }
}
```

LottieView in Compose

```
@Composable
fun FloatingCartViewCompose(
    getCartItemCountUseCase = GetCartItemCountUseCase(),
    clickable: () -> Unit = {},
) {
  Box(
    modifier = Modifier
        ...
        .clickable { clickable() }
  ) {
    ...
    val eventHandler = remember { mutableStateOf(Lifecycle.Event.ON_ANY) }
    val lifecycleOwner = rememberUpdatedState(LocalLifecycleOwner.current)
    DisposableEffect(lifecycleOwner.value) {
        val lifecycle = lifecycleOwner.value.lifecycle
        val observer = LifecycleEventObserver { owner, event ->
            eventHandler.value = event
        }

        lifecycle.addObserver(observer)
        onDispose {
            lifecycle.removeObserver(observer)
        }
    }
```

LottieView in Compose

```
            eventHandler.value = event
        }

        lifecycle.addObserver(observer)
        onDispose {
            lifecycle.removeObserver(observer)
        }
    }

    eventHandler.value.let {
        AndroidView(factory = {
            val view = LottieAnimationView(it).apply {}
            return@AndroidView view
        }, update = {
          if (event == Lifecycle.Event.ON_RESUME) {
            getCartItemCountUseCase.execute(onNext = { newCartCount ->
                ...
                playAddAnimation(differenceCount, it)
            })
          }
        })
    }
  }
}
```

LottieCompose

```
val composition by rememberLottieComposition(
            LottieCompositionSpec.Asset("cart_ani.json"))
val state = remember { CartAniState() }

LaunchedEffect(eventHandler.value) {
    if (eventHandler.value == Lifecycle.Event.ON_RESUME) {
        state.animatable.animate(
            composition,
            iterations = 1,
            initialProgress = 0f,
            speed = 1f,
            continueFromPreviousAnimate = false,
        )
    }
}

LottieAnimation(
    composition,
    progress = state.animatable.progress
)
```

몇몇 아쉬운 점들

몇몇 라이브러리는 사용할 수 없음

- ExoPlayer
- 이미지 라이브러리 (Glide 등)
- MotionLayout
- ViewPager..?

어차피

전부 바로 변환하기는 힘들겠지만..

몇몇 아쉬운 점들

Resource 관련 사용할 수 없는 점들

- Shape Drawable 사용 불가능 (xml)
- Selector Drawable 사용 불가능
- ColorStateList 사용 불가능
- 나인패치 이미지 사용 불가능

어차피

전부 바로 변환하기는 힘들겠지만..

몇몇 아쉬운 점들

개발 생산성은??

- MVVM 으로 대체 언제 옮겨 갈까
- 러닝 커브가 확실히 존재
- Preview 만으로는 xml 에서 작성하는 것보다는 불편하다 -> 생산성 저하
- Build 시간, apk 사이즈

어차피

전부 바로 변환하기는 힘들겠지만..

그래서 앞으로는요?

그래도 이제는 조금씩 도입할 때

- 팀 내에서 학습하며 러닝커브를 극복
- 간단한 화면부터 적용해 볼 예정
- 간결하고 UI 위주의 코드는 👍
- 아키텍쳐부터 천천히 고민해보자

들어주셔서
감사합니다

Thank you!

C 우아콘
WOOWA
N 2021