

Android Is Not Just Mobile: Lessons from TV, Auto, and Wear



THE MOBILE PRIVILEGE DASHBOARD

The Assumptions We Never Questioned

1. DIRECT TOUCH

The "Perfect Precision" Gauge



INPUT PRECISION

100%

Infinite TouchTargets

(No rotary. No D-pad. No 2-inch screens.)

3. UNLIMITED POWER

The "Infinite Battery" Icon



SoC PERFORMANCE

100%

Thermal Throttling? What's that?

2. FOCUSED ATTENTION

The "Zero Distraction" Beam



ATTENTION

100%

Guaranteed Glances

(Always foreground.
Always you.)

4. PERFECT ENVIRONMENT

The "Always Connected" Assumption



CONNECTION QUALITY

100%

Gigabit Fantasy

(5G, Wi-Fi 6E, Everywhere.)

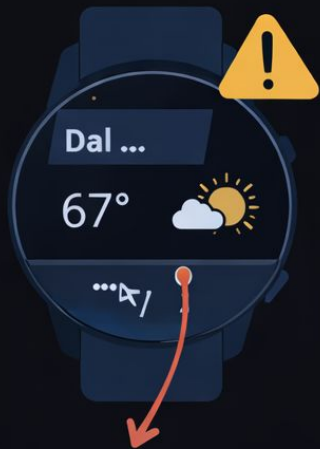


WE'VE BEEN SPOILED.

Mobile set the bar. Cars and TVs don't play by these rules.

Wait, What?

Wear OS



Overflowing text

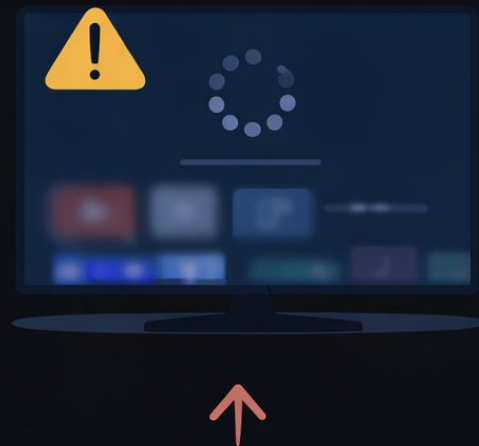
Car



Cut-off elements

Overlapping UI

Android TV



Endless loading

Low resolution graphics

Who am I

Senior Mobile Developer | IEEE Senior Member

- **Experience:** 10+ Years (IG Group, ex-Kleinanzeigen)
- **Tech:** Flutter & KMP Enthusiast | Clean Architecture OSS Maintainer
- **Community:** Judge Volunteer | Raptors Hackathon
- **Research:** Published on ProAndroidDev & IJCA





The 10-Foot Reality: Designing for Distance

The Foundation: The Leanback Pattern

```
class MainTVFragment : BrowseSupportFragment() {

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        setupUI()
        loadRows()
    }

    private fun setupUI() {
        title = "Ecosystem VOD"
        headersState = HEADERS_ENABLED
        isHeadersTransitionOnBackEnabled = true
        brandColor = TV_BRAND_COLOR
    }

    private fun loadRows() {
        val rowsAdapter = ArrayObjectAdapter(ListRowPresenter())
        val cardPresenter = CardPresenter()
        val header = HeaderItem(0, "Continue Watching")
        val listRowAdapter = ArrayObjectAdapter(cardPresenter).apply {
            add(Movie("Interstellar", "Sci-Fi"))
            add(Movie("Inception", "Sci-Fi"))
            add(Movie("The Dark Knight", "Action"))
        }

        rowsAdapter.add(ListRow(header, listRowAdapter))
        adapter = rowsAdapter
    }
}
```

```
class CardPresenter : Presenter() {

    override fun onCreateViewHolder(parent: ViewGroup): ViewHolder {
        // cardView logic creating
        return ViewHolder(cardView)
    }

    override fun onBindViewHolder(viewHolder: ViewHolder, item: Any?) {
        // bind logic
    }

    override fun onUnbindViewHolder(viewHolder: ViewHolder?) {
        // Clean up resources
    }
}
```

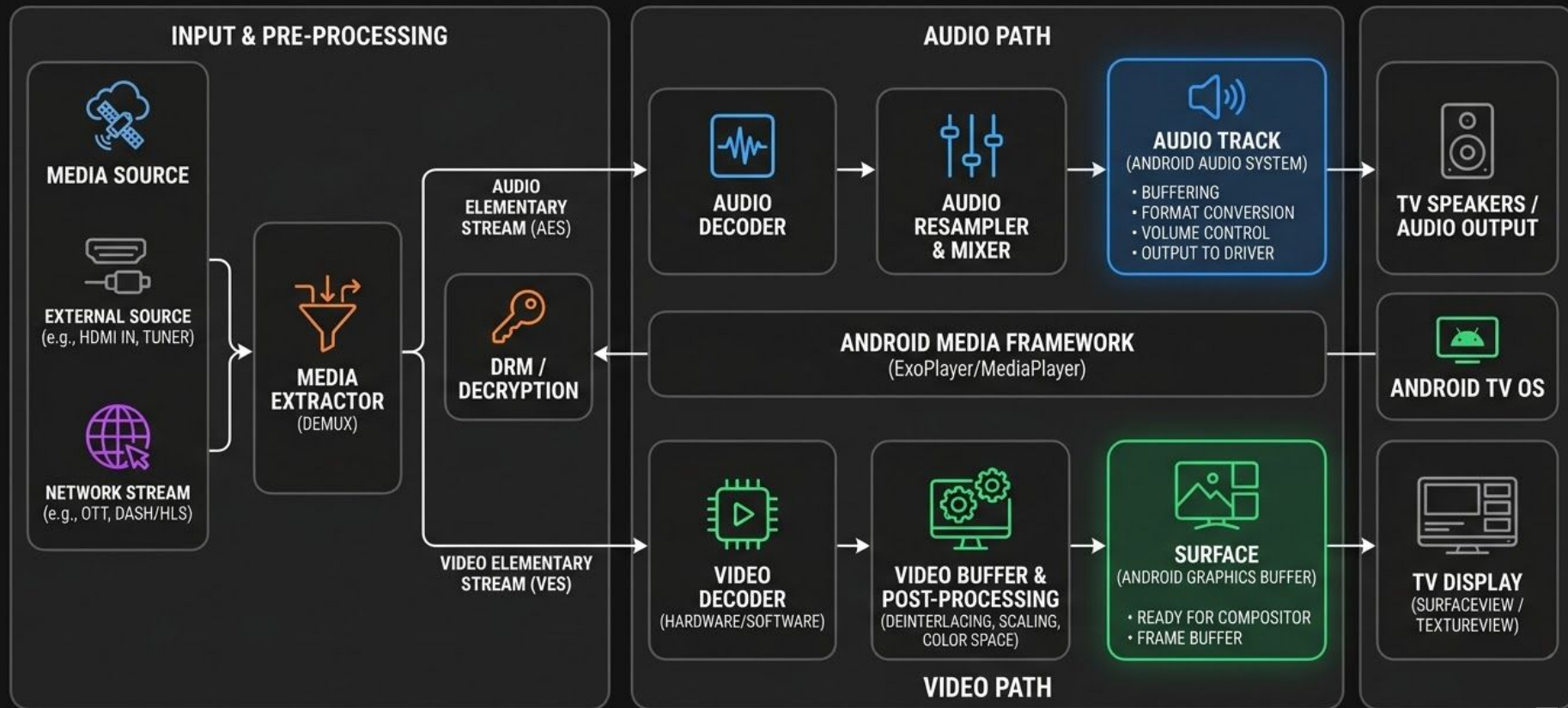
The Modern Shift: Declarative TV UIs

```
@Composable
fun MovieBrowserScreen() {
    val movies = listOf(
        Movie("Inception", "2010", R.drawable.inception_poster)
    )
    TvLazyColumn(
        modifier = Modifier
            .fillMaxSize()
            .background(Color.Black)
            .padding(16.dp)
    ) {
        item {
            Text(
                text = "Popular",
                color = Color.White,
                fontSize = 24.sp,
                modifier = Modifier.padding(bottom = 8.dp)
            )
        }
        item {
            TvLazyRow(
                horizontalArrangement = Arrangement.spacedBy(12.dp)
            ) {
                items(movies) { movie →
                    MovieCard(movie)
                }
            }
        }
    }
}
```

```
@Composable
fun MovieCard(movie: Movie) {
    var hasFocus by remember { mutableStateOf(false) }

    TvFocusableSurface(
        onFocusChanged = { hasFocus = it },
        onClick = { /* Handle click */ },
        modifier = Modifier
            .size(160.dp, 240.dp)
            .border(
                width = if (hasFocus) 2.dp else 0.dp,
                color = Color.Yellow,
                shape = RoundedCornerShape(8.dp)
            )
            .clip(RoundedCornerShape(8.dp))
            .background(Color.DarkGray)
    ) {
        Column {
            Image(
                painter = painterResource(id = movie.posterRes),
                contentDescription = movie.title,
                contentScale = ContentScale.Crop,
                modifier = Modifier.fillMaxSize()
            )
        }
    }
}
```

The Streaming Pipeline: Architecture Overview



The Playback Engine – Orchestrating the Immersive Layer

```
@Composable
fun GdgVideoPlayerSample(videoUri: String) {
    val context = LocalContext.current

    val exoPlayer = remember {
        ExoPlayer.Builder(context).build().apply {
            setMediaItem(MediaItem.fromUri(videoUri))
            prepare()
            playWhenReady = true
            repeatMode = Player.REPEAT_MODE_ONE
        }
    }

    DisposableEffect(Unit) {
        onDispose { exoPlayer.release() }
    }

    Box(
        modifier = Modifier
            .fillMaxSize()
            .background(Color.Black),
        contentAlignment = Alignment.Center
    ) {
        PlayerSurface(
            player = exoPlayer,
            modifier = Modifier.fillMaxSize()
        )

        PlayerControlsOverlay(player = exoPlayer)
    }
}
```

```
@Composable
fun PlayerControlsOverlay(player: Player) {
    var isPlaying by remember { mutableStateOf(player.isPlaying) }

    DisposableEffect(player) {
        val listener = object : Player.Listener {
            override fun onIsPlayingChanged(playing: Boolean) {
                isPlaying = playing
            }
        }
        player.addListener(listener)
        onDispose { player.removeListener(listener) }
    }

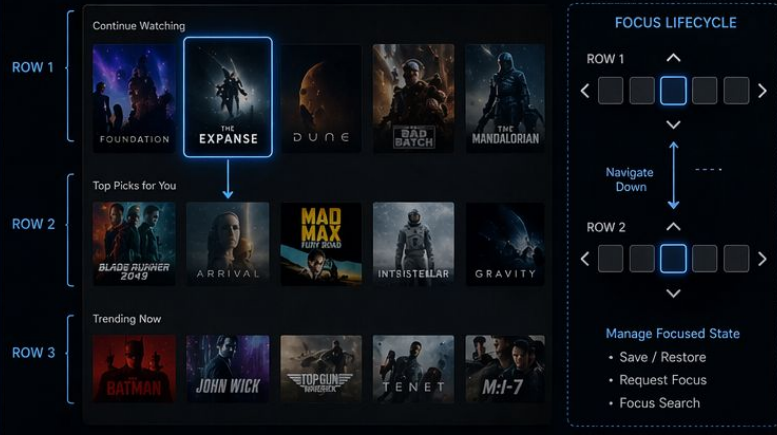
    Row(
        modifier = Modifier
            .align(Alignment.BottomCenter)
            .fillMaxWidth()
            .background(Color.Black.copy(alpha = 0.5f))
            .padding(16.dp),
        horizontalArrangement = Arrangement.Center,
        verticalAlignment = Alignment.CenterVertically
    ) {
        Button(
            onClick = {
                if (isPlaying) player.pause() else player.play()
            },
            colors = ButtonDefaults.buttonColors(
                containerColor = if (isPlaying) MaterialTheme.colorScheme.error
                else MaterialTheme.colorScheme.primary
            )
        ) { Text(if (isPlaying) "PAUSE" else "PLAY") }

        Spacer(modifier = Modifier.width(16.dp))

        Slider(
            value = 0.5f,
            onValueChange = { /* player.seekTo(...) */ },
            modifier = Modifier.weight(1f)
        )
    }
}
```

1. THE LEANBACK EXPERIENCE (UX)

BrowseSupportFragment - Focus Management Pattern



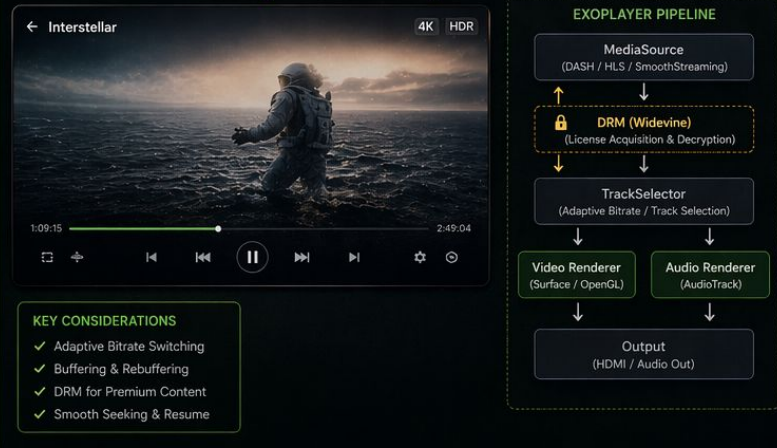
2. PERFORMANCE vs. PRESENTATION (RENDERING)

DetailsFragment - High-Resolution Asset Loading



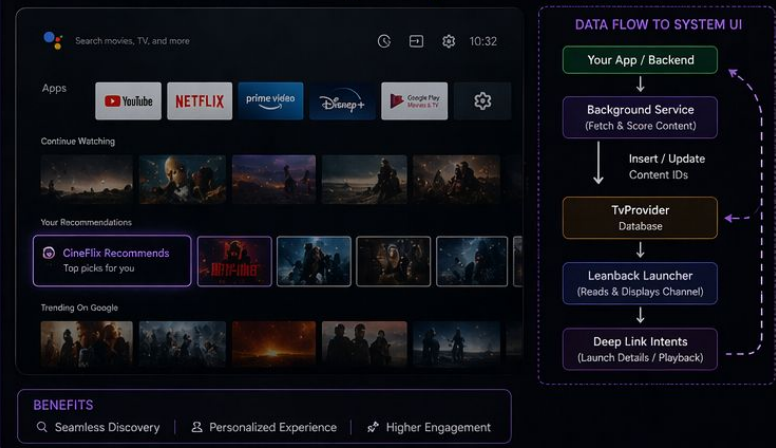
3. THE VIDEO CORE (PLAYBACK LOGIC)

ExoPlayer - The Content Engine



4. THE CONTEXTUAL HOOK (SYSTEM INTEGRATION)

Recommendation Channels - System Integration & Deep-Linking





The Wrist Constraint – Engineering for the
5-Second Glance

5 SECONDS

The Glance vs. The Session.

80% of Wear OS interactions are
under 10 seconds



The Battery Budget: Tiles vs. Activities



```
class QuickActionTile : TileService() {  
    override fun onTileRequest(request: TileRequest) =  
        Futures.immediateFuture(  
            Tile.Builder()  
                .setResourcesVersion("1.0")  
                .setTimeline(Timeline.fromLayout(  
                    Layout.Builder().setRoot(  
                        // Pre-rendered, low-power UI  
                        primaryActionLayout()  
                    ).build()  
                ))).build()  
        )  
}
```

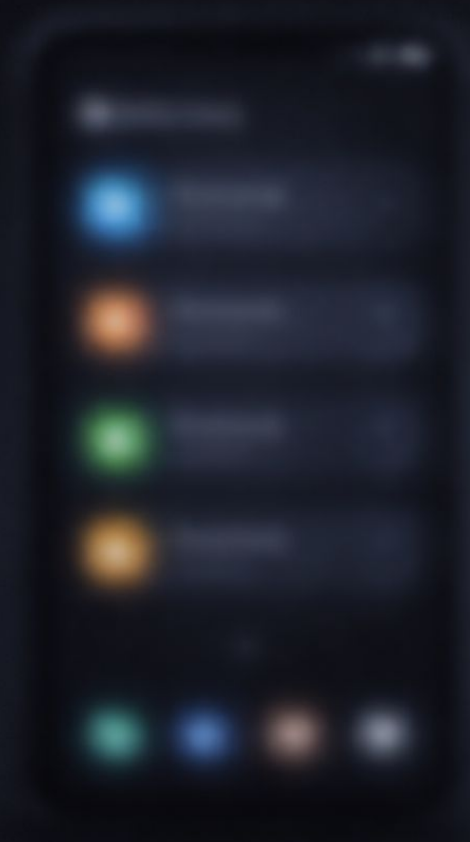
Non-Euclidean UI – Solving the Coordinate Mismatch



```
val scalingParams = ScalingParams(  
    edgeScale = 0.5f,           // The mathematical 'drop-off' at the bezel  
    edgeAlpha = 0.0f,          // Fading out into the 'physical' dead-zone  
    minElementHeight = 0.2f // Preventing layout 'jumping' during scroll  
)  
  
ScalingLazyColumn(  
    autoCentering = AutoCenteringParams(itemIndex = 0),  
    anchorType = ScalingLazyListAnchorType.ItemCenter, // Anchor to center, not top  
    modifier = Modifier.rotaryScrollable(rotaryHandler) // Direct Crown integration  
) {  
    item { CurvedText(text = "9:41 AM", style = CurvedTextStyle(fontSize = 14.sp)) }  
    items(data) { Chip(...) }  
}
```

The 2-Tap Mandate: Ruthless Prioritization

If it takes more than 2 taps, move it to the phone.



WEAR OS PRODUCTION RESULTS: **GLANCEABLE, NOT BROWSABLE**

2-TAP MANDATE • CIRCULAR GEOMETRY • GLANCE-FIRST ARCHITECTURE

1. ENTRY POINTS LIVE OUTSIDE THE APP

The 5-second glance happens here.



DYNAMIC TILE
High value, zero friction.



COMPLICATIONS
Context in the watch face.



VALUE DELIVERED
Before opening the main app.

2. RESPECT THE CIRCULAR GEOMETRY

ScalingLazyColumn maintains the Safe Zone.



Shrinks into the bezel

Safe Zone

Readable content lives here

Shrinks into the bezel

SCALINGLAZYCOLUMN

- ✓ Items scale at the edges
- ✓ Maintains readability
- ✓ Natural focus flow
- ✓ Better ergonomics



3. MAIN APP EXPERIENCE (2-TAP MANDATE)

Deep content, fast access, minimal friction.



Focused item is larger and clear

Safe Zone ensures readability

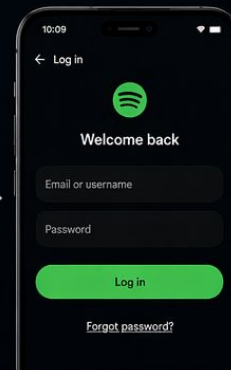
Smooth scale at edges

4. THE HAND-OFF: REMOTE ACTIVITY

Don't make users type on a 1-inch screen.



Triggers Remote Activity



**THE TAKEAWAY
FOR WEAR OS**

Your success isn't defined by how much time the user spends in your app.
It's defined by **how little** they needed to look at it to get what they wanted.



**GLANCEABLE
NOT BROWSABLE**

The challenge — don't kill the driver



The Template Trap – Engineering for Zero Distraction



Custom UI
Any layout,
any visuals



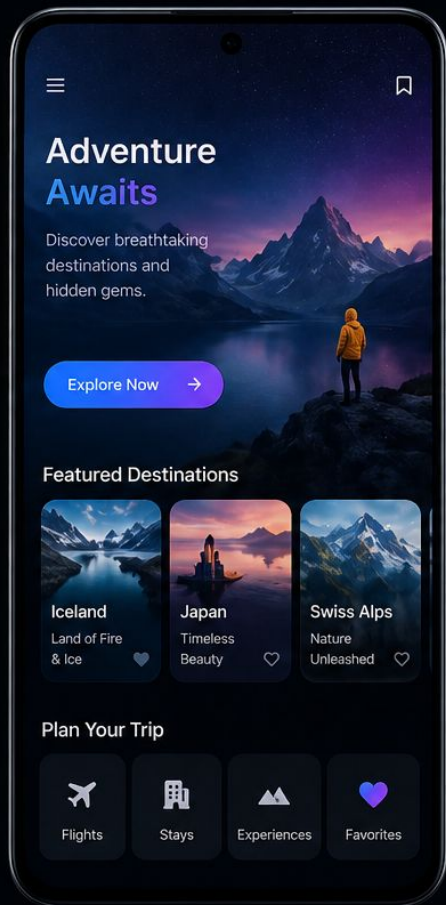
Rich Motion
Animations and
micro-interactions



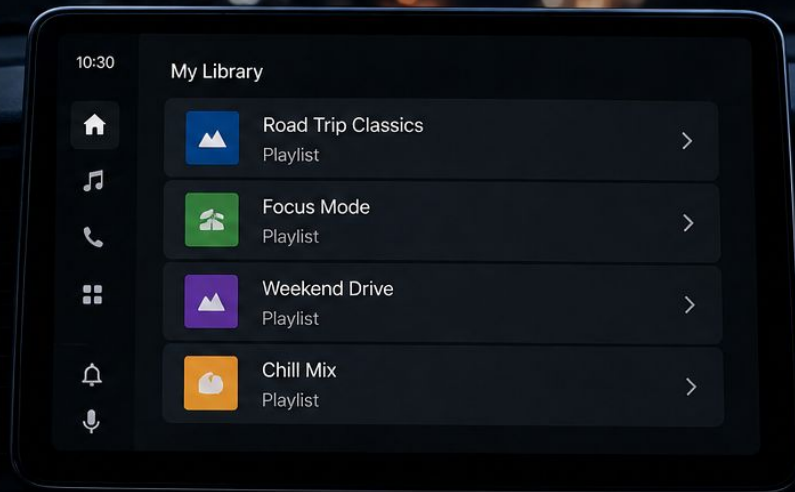
Brand Identity
Full control over
colors, shapes,
typography



Complex Navigation
Nested flows,
deep structures



VS



Large Touch Targets
Min. 48dp



High Contrast
Always



Low Cognitive Load
Simple > Complex



System Limits
6 Tap Limit
Refresh Rate Limit



Break the rules, break safety.

Excessive taps or distractions can lead to app termination.

The Orchestration Layer – State vs. Quota

```
class SearchScreen(carContext: CarContext) : Screen(carContext) {

    private var isSearching = false

    override fun onGetTemplate(): Template {
        return SearchTemplate.Builder(object : SearchTemplate.SearchCallback {
            override fun onSearchSubmitted(searchText: String) {

                processVoiceSearch(searchText)
            }
            override fun onSearchTextChanged(searchText: String) {

                debounceSearch(searchText)
            }
        })
        .setHeaderAction(Action.BACK)
        .setShowKeyboardByDefault(false)
        .build()
    }

    private fun processVoiceSearch(query: String) {
        // Implementation of the Search-to-Drive logic
    }
}
```

Adapting for the Driver: Safe & Glanceable UI



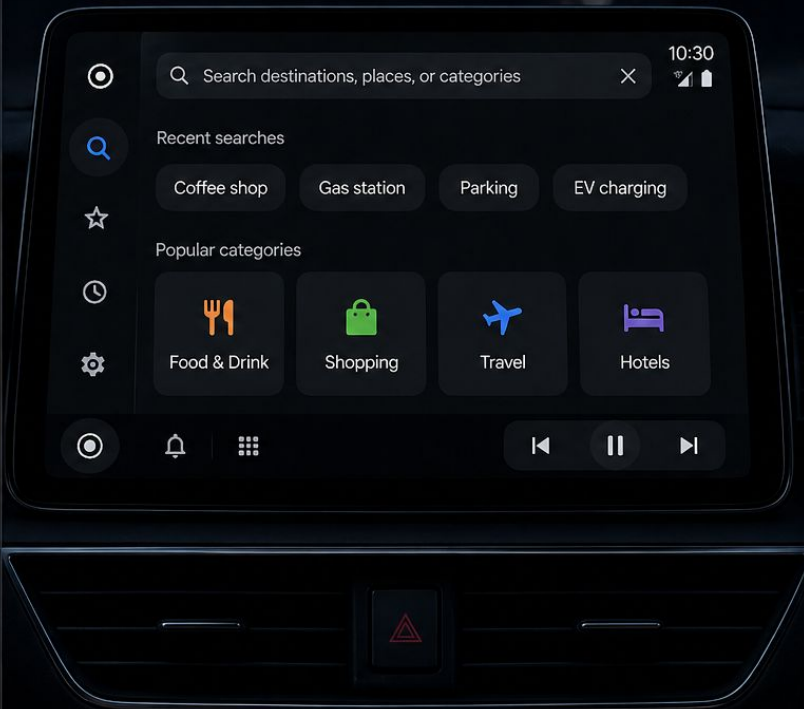
ListTemplate

Best for browsing and structured content



SearchTemplate

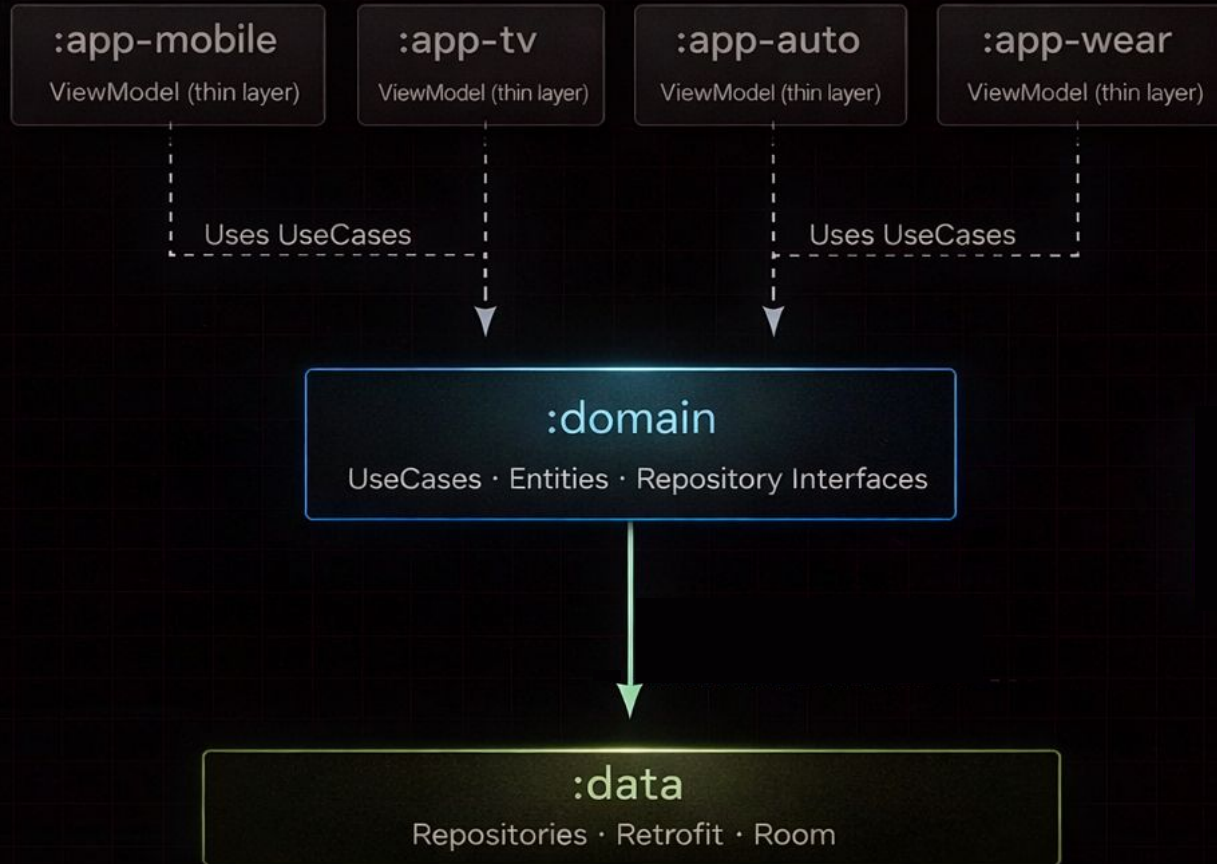
Best for search and suggestions



The Orchestration Layer – State vs. Quota



The Orchestration Layer – State vs. Quota



The Code of Sharing – Logic Without Borders



```
class HeartRateViewModel(  
    private val repository: HealthRepository  
) : ViewModel() {  
  
    private val _uiState = MutableStateFlow(HeartRateState())  
    val uiState: StateFlow<HeartRateState> = _uiState.asStateFlow()  
  
    fun toggleMonitoring() {  
        viewModelScope.launch {  
            repository.toggle()  
        }  
    }  
}
```

The Code of Sharing – Logic Without Borders

```
// Inside :app-wear (Consumer 1)
@Composable
fun WearScreen(vm: HeartRateViewModel) {
    val state by vm.uiState.collectAsState()
    ScalingLazyColumn { // Circular UI
        item { CurvedText(state.bpmLabel) }
    }
}

// Inside :app-tv (Consumer 2)
@Composable
fun TvScreen(vm: HeartRateViewModel) {
    val state by vm.uiState.collectAsState()
    TvLazyRow { // 4K Landscape UI
        Card { Text("Heart Rate: ${state.bpm}") }
    }
}
```

```
// Inside :app-auto (The third consumer)
class HeartRateCarScreen(
    carContext: CarContext,
    vm: HeartRateViewModel
) : Screen(carContext) {
    override fun onGetTemplate(): Template {
        val state = vm.uiState.value

        return ListTemplate.Builder()
            .setTitle("Heart Rate Monitor")
            .setSingleList(
                ItemList.Builder()
                    .addItem(
                        Row.Builder()
                            .setTitle("Current BPM")
                            .addText("${state.bpm}")
                            .build()
                    )
                    .build()
            )
            .build()
    }
}
```

Battle Scars – What the Documentation Doesn't Tell You



1. The Emulator Lie:

Desktop vs. Target Hardware.



2. The Connectivity Gap:

“Eventually Consistent” sync.



3. The Lifecycle Ghost:

Background restrictions.



Resources and Useful Links



Wear OS



Android Auto



Android TV

THANK YOU

