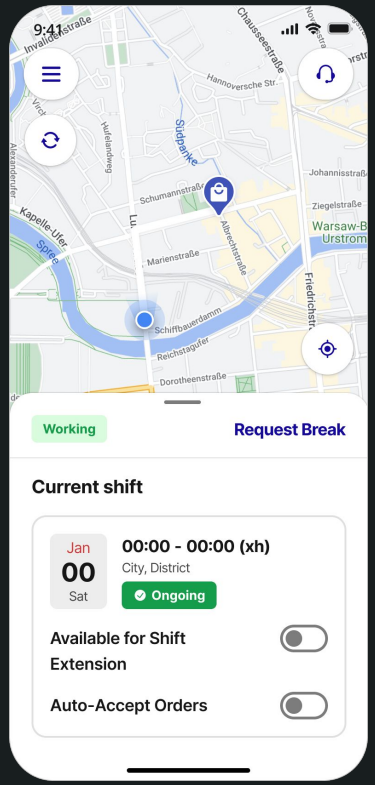
Composing ViewModels

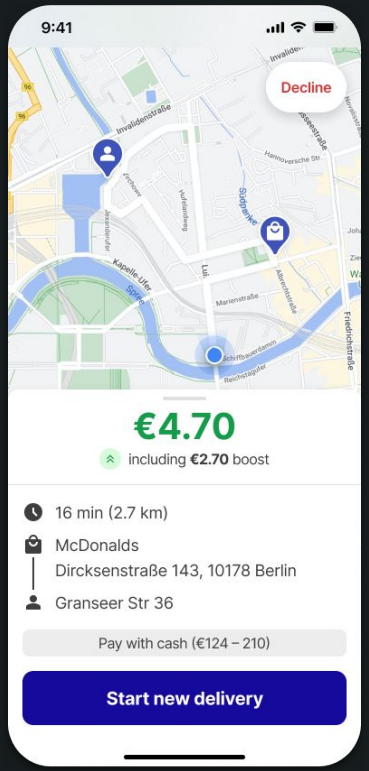
Breaking ViewModels into smaller self-contained UI models

Hakan Bagci
@hknbgcdev

Micro Features

Rider Application





Delivery Hero

panda rider

foodora rider

HungerStation rider

talabat rider

PedidosYa Rider

efood rider

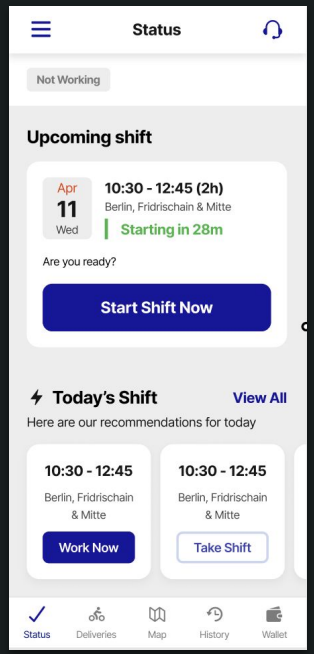
Yemeksepeti rider

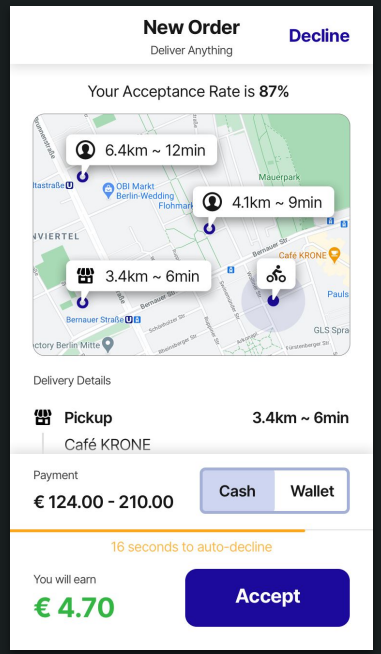
배민커넥트 오산시

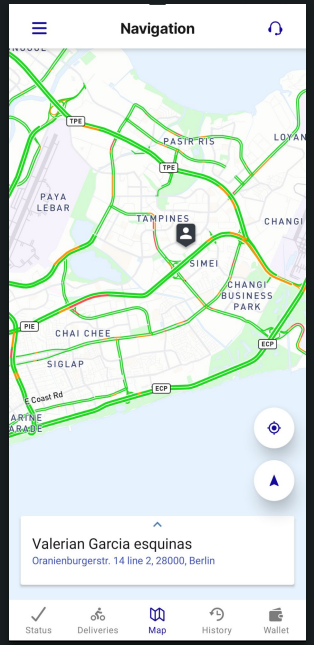
70+ Countries

500K+ Monthly Active Riders

Rider Application - Before

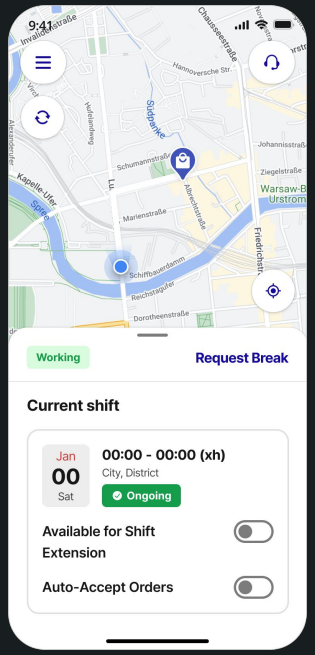


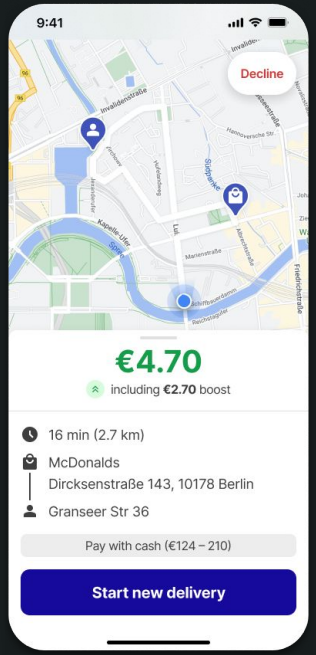


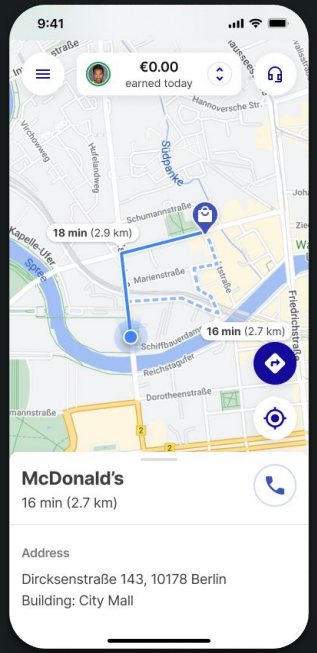


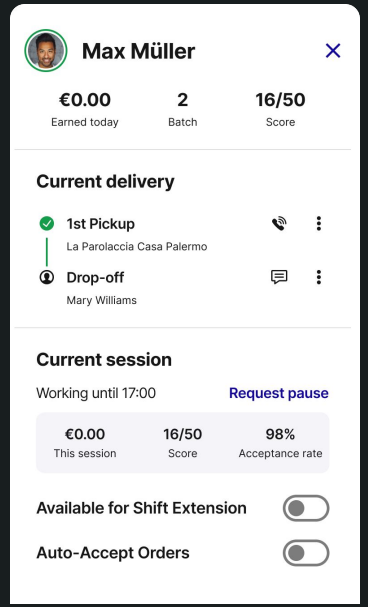
Bottom navigation with isolated features

Rider Application - New Design









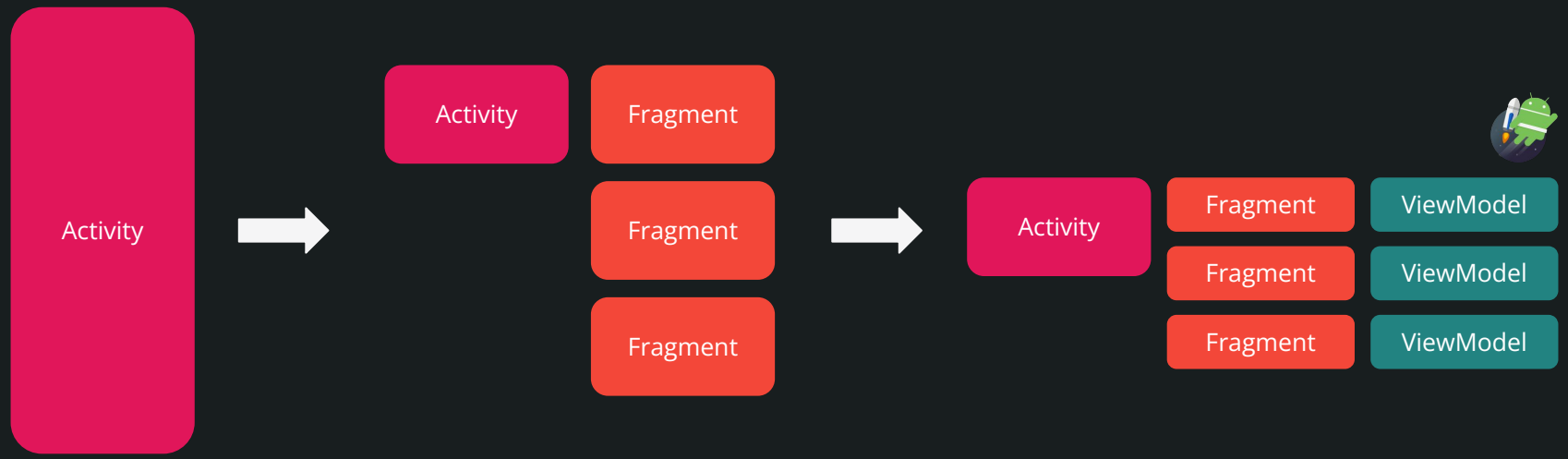
Single shared contextual screen hosting all features

Architecture Requirements

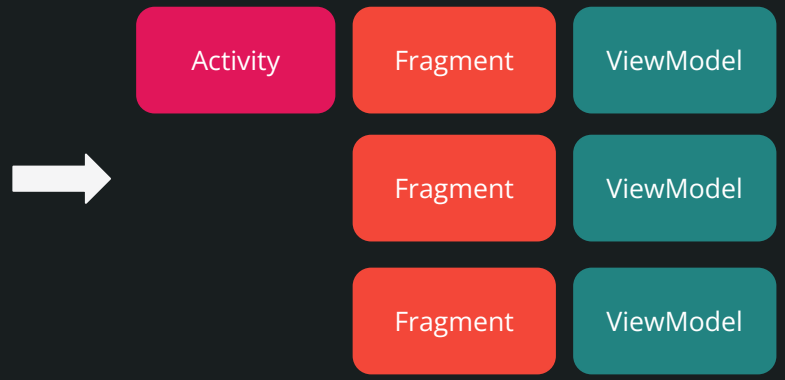
- Enable showing dynamic/contextual content on a single screen
- Enable showing UI components that are backed by different data sources
- Foster portability by enabling composition of UIs in any host
- Keep loose coupling between features/squads/domains

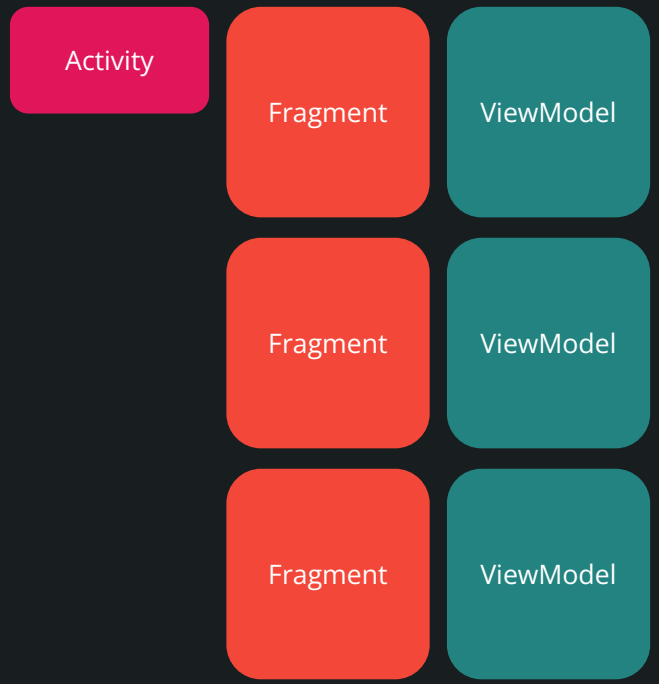
Does our current architecture support these requirements?

Architecture Review



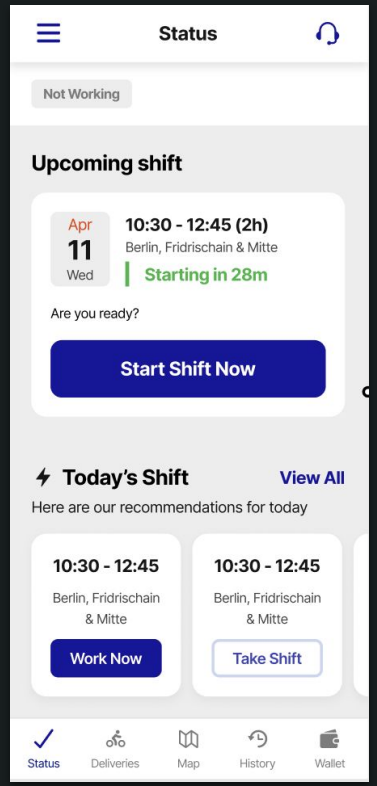
Architecture Review





Is it possible to reuse UI logic from the existing ViewModels?

Architecture Review



Is it possible to use existing UI components?

Tightly coupled with the host fragment

RecyclerView complexity

One UI state mapper for whole screen

Not written in Compose

❌

Can we reuse existing domain/data layer?

Observable repositories

Domain use cases

✅

Paradigm Shift 🚀

Create an isolated screen → Implement UI components specific to that screen

Create self-contained host-agnostic UI components → Compose UIs using these components

Micro Features ✨

Book sessions

Take orders by booking a session

See available sessions

How is the pick up experience?

Suspended

Update profile

To revoke suspension, update missing data or documents

Current Shift

Apr
12
Tue

Until 13:00
Kadikoy
Ongoing

Searching for orders...

To get more orders, go to hot areas

Instant Working

Start working now and stop whenever you want.

Work Area
Tucholskystr

Stop Working

Settings

Manual Order Selection

Auto accept orders

Auto accept is not available when using Manual Order selection

Current Zone name

Demand is high, you can work now (1h guaranteed)

Work now

Starting area

Navigate

You are late! To start working, go to the highlighted area.

Burgeramt

Navigate

Address for the pick up destination

Offer to Work

Set yourself as available and you will get notified when there's a session for you in the next 2 hours.

Become available

Working

Cancel Break

Your break will start after your current order drop-off

Your session is almost over

You can end your session after your current order is completed

Apr
11
Tue

13:30 - 15:30 (2h)
Kreuzberg
Starting in 15 min.

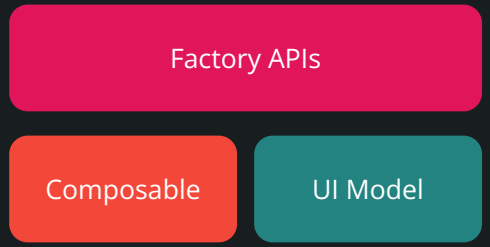
Start session

Available for session extension

Your session will be extended if it's needed

What is a micro-feature?

Micro-features



What is a UI Model?

How is it different from a ViewModel?

- Consists of a UI Model and a Composable
- Self-contained, implements its own Unidirectional Data Flow (UDF)
- Provides APIs (factory methods) to create its UI model and composable
- Host agnostic, needs a host to start functioning
- Fosters composition of UI Models and Composables

Jetpack ViewModel vs UI Model

	Jetpack ViewModel	UI Model	
Populate and publish state to the UI	✅	✅	
React to events from UI	✅	✅	
Persist data through configuration changes	✅	✅	Hosted by ViewModel
Platform independent	❌	✅	
Have coroutine scope access	✅	✅	Hosted by ViewModel
Easier test setup	❌	✅	

Jetpack ViewModel vs UI Model

Are we getting rid of ViewModels?

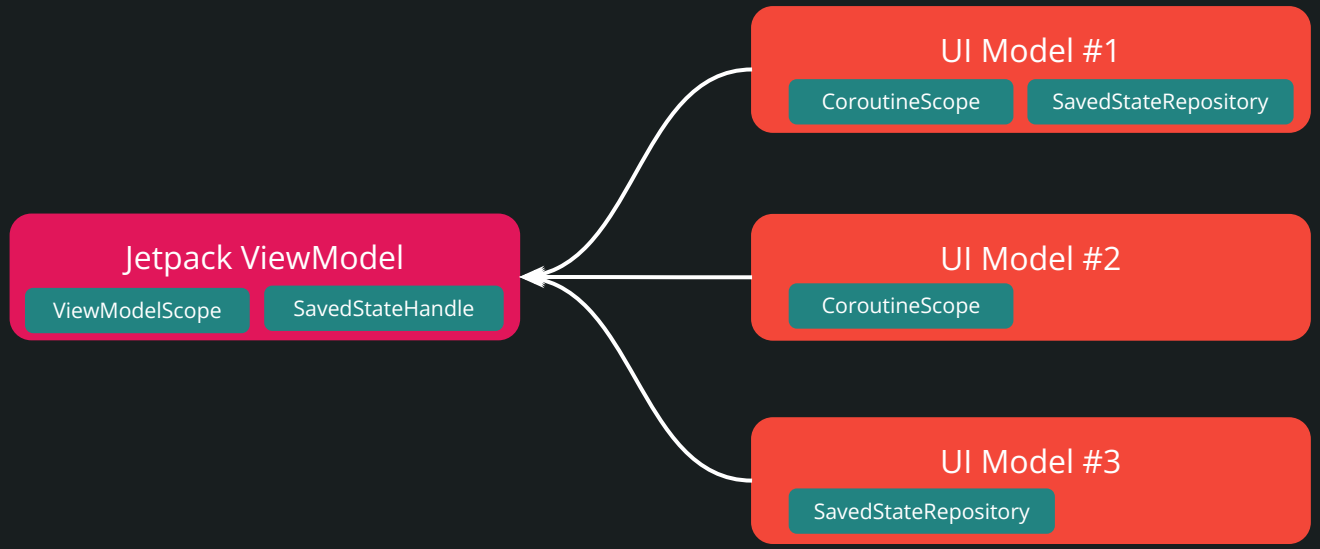
Idea is to use the powers of ViewModel while decreasing the platform dependency

ViewModelScope

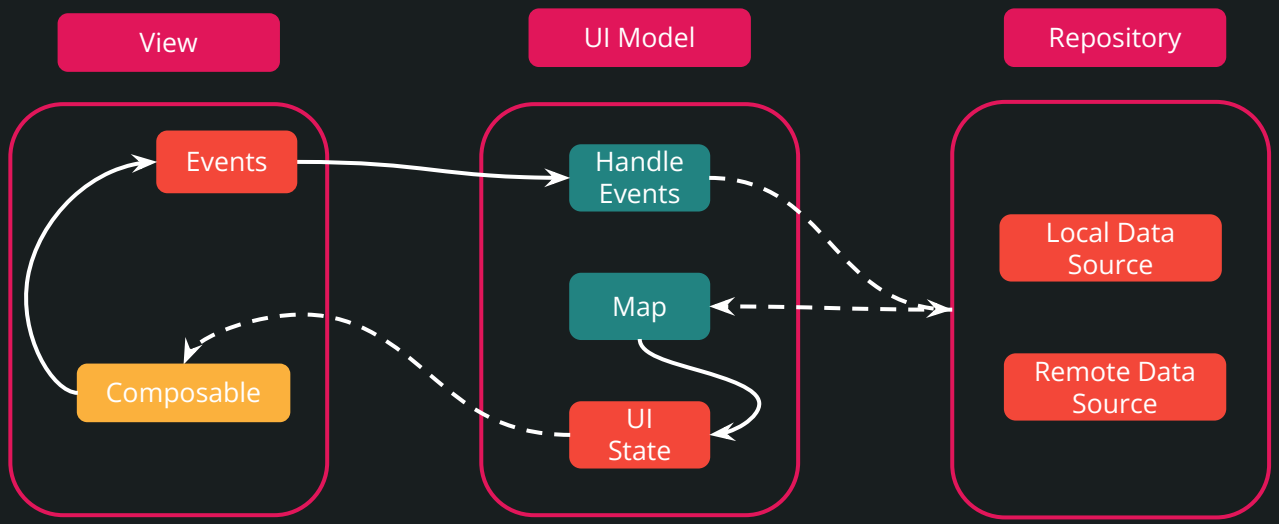
Configuration Change Survival

SavedStateHandle

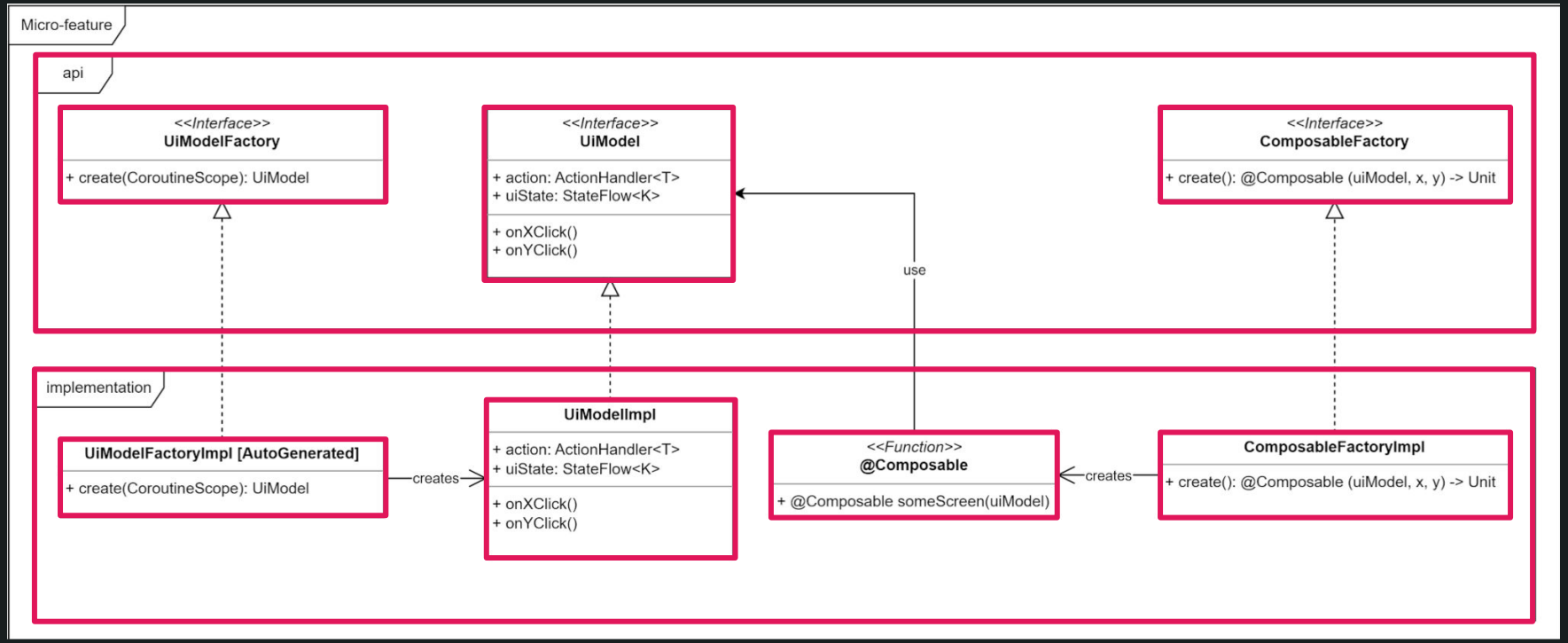
Composing ViewModels



Unidirectional Data Flow



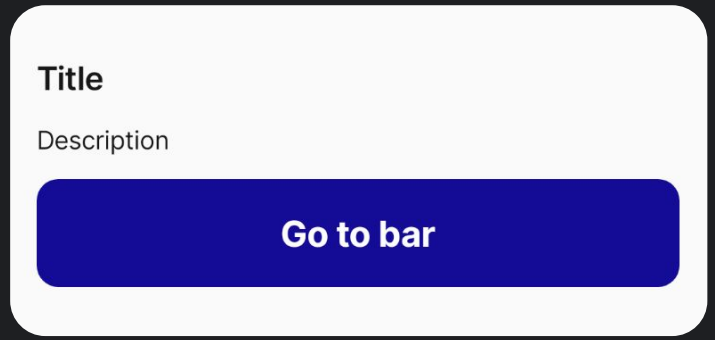
Blueprint of a Micro-feature



Sample micro-feature

Micro-feature Sample - Api

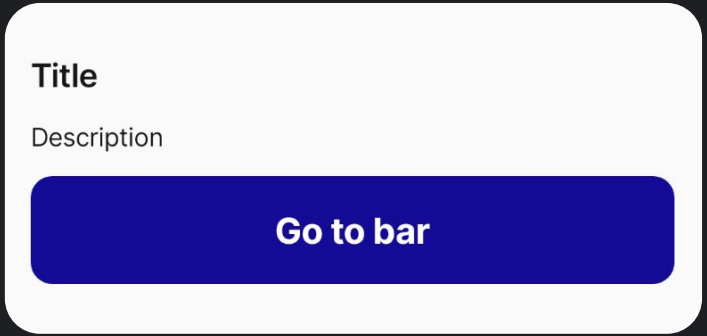
```
interface FooUiModel {

    val uiState: StateFlow<FooUiState>

    val action: ActionHandler<FooAction>

    fun onGoToBarClicked()

    interface Factory {

        fun create(
            coroutineScope: CoroutineScope,
        ): FooUiModel
    }
}
```



Micro-feature Sample - Api

```kotlin
sealed class FooUiState {

    object Unavailable : FooUiState()

    object Loading : FooUiState()

    data class Available(
        val title: String,
        val description: String,
        val buttonText: String,
    ) : FooUiState()
}
```

```kotlin
sealed class FooAction {

    object GoToBar : FooAction()

    data class ShowError(
        val message: String,
    ) : FooAction()
}
```



Micro-feature Sample - Implementation

```
class FooUiModelImpl @AssistedInject constructor(
    private val getFooUiState: GetFooUiState,
    override val action: ActionHandler<FooAction>,
    @Assisted private val coroutineScope: CoroutineScope,
) : FooUiModel {

    ...

    @AssistedFactory
    interface Factory : FooUiModel.Factory {
        override fun create(
            coroutineScope: CoroutineScope,
        ): FooUiModelImpl
    }
}
```

Micro-feature Sample - Implementation

```
class FooUiModelImpl @AssistedInject constructor(...) : FooUiModel {

    private val _uiState: MutableStateFlow<FooUiState> = MutableStateFlow(Unavailable)
    override val uiState: StateFlow<FooUiState> = _uiState.asStateFlow()

    init {
        coroutineScope.launch {
            getFooUiState().collect {
                _uiState.value = it
            }
        }
    }

    override fun onGoToBarClicked() {
        action.update(FooAction.GoToBar)
    }

    ...
}
```

Micro-feature Sample - Api

```
interface FooComposableFactory {

   fun create(): FooComposable
}
```

```
typealias FooComposable = @Composable (
    uiModel: FooUiModel,
    showSnackbar: (String) → Unit,
) → Unit
```

Micro-feature Sample - Implementation

```
class FooComposableFactoryImpl @Inject constructor(
   private val barNavigator: BarNavigator,
) : FooComposableFactory {

   override fun create(): FooComposable = { uiModel, showSnackbar →
       Foo(
           uiModel = uiModel,
           onGoToBarClicked = barNavigator::goToBar,
           showSnackbar = showSnackbar,
       )
   }
}
```

Micro-feature Sample - Implementation

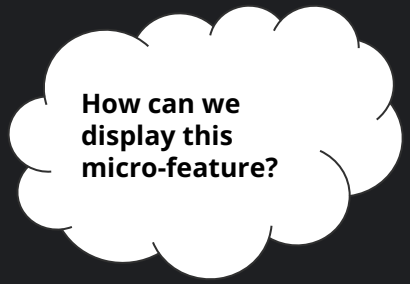
```
@Composable
fun Foo(
    uiModel: FooUiModel,
    onGoToBarClicked: () → Unit,
    showSnackbar: (String) → Unit,
    modifier: Modifier = Modifier,
) {
    ActionHandlerDisposableEffect(
        actionHandler = uiModel.action,
    ) { action →
        when (action) {
            FooAction.GoToBar → onGoToBarClicked()
            is FooAction.ShowError → showSnackbar(action.message)
        }
    }
    ...
}
```

Micro-feature Sample - Implementation

```
@Composable
fun Foo(
  ...
) {
    ...

    val uiState = uiModel.uiState.collectAsStateWithLifecycle()

    when (uiState) {
        is Available → FooContent(
            uiState = uiState,
            onGoToBarClicked = uiModel::onGoToBarClicked,
            modifier = modifier,
        )
        Unavailable → Unit
        Loading → {
            // Show loading state
        }
    }
}
```

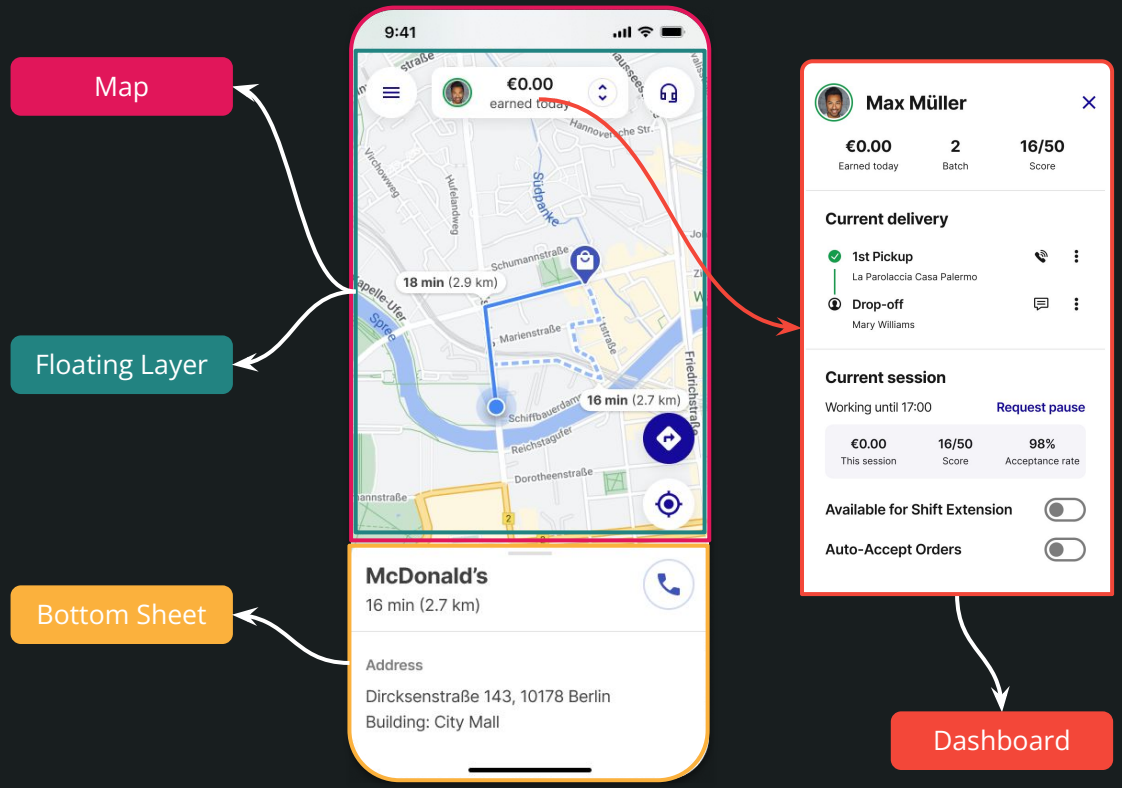
Micro-feature Sample - Implementation

```
@Composable
fun FooContent(
    uiState: FooUiState.Available,
    onGoToBarClicked: () -> Unit,
    modifier: Modifier = Modifier,
) {

    Column {

        Text(text = uiState.title)

        Text(text = uiState.description)

        Button(
            text = uiState.buttonText,
            onClick = onGoToBarClicked,
        )
    }
}
```



Hosts

- App scaffold is composed of multiple hosts
- Each host uses micro-feature factory APIs to populate its content
- Hosts, as containers, may define rules to layout micro-features



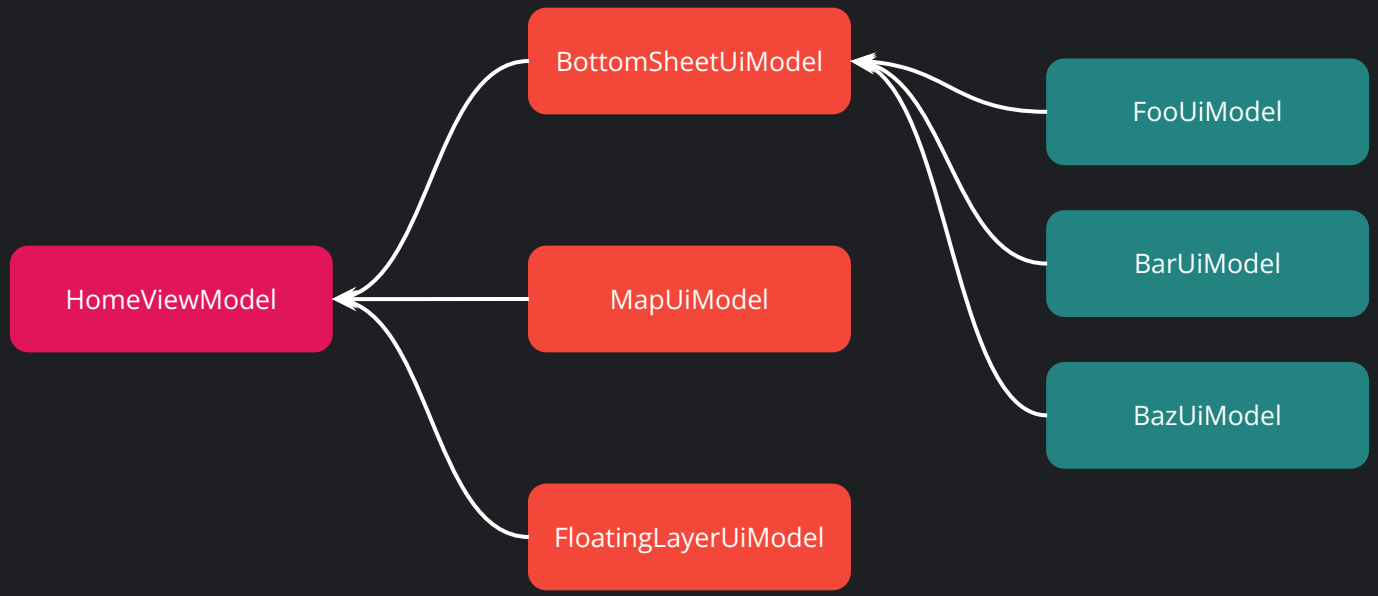
Micro-feature Host Integration - UI Model

```
class HomeViewModel @Inject constructor(
    bottomSheetUiModelFactory: BottomSheetUiModel.Factory,
    mapUiModelFactory: MapUiModel.Factory,
    floatingLayerUiModelFactory: FloatingLayerUiModel.Factory,
) : ViewModel() {

    val bottomSheetUiModel = bottomSheetUiModelFactory
        .create(viewModelScope)

    val mapUiModel = mapUiModelFactory
        .create(viewModelScope)

    val floatingLayerUiModel = floatingLayerUiModelFactory
        .create(viewModelScope)
}
```

Micro-feature Host Integration - UI Model

```
class BottomSheetUiModelImpl @AssistedInject constructor(
    fooUiModelFactory: FooUiModel.Factory,
    barUiModelFactory: BarUiModel.Factory,
    bazUiModelFactory: BazUiModel.Factory,
    @Assisted coroutineScope: CoroutineScope,
) : BottomSheetUiModel {

    override val fooUiModel = fooUiModelFactory
        .create(coroutineScope)

    override val barUiModel = barUiModelFactory
        .create(coroutineScope)

    override val bazUiModel = bazUiModelFactory
        .create(coroutineScope)
}
```

UI Model Composition Tree



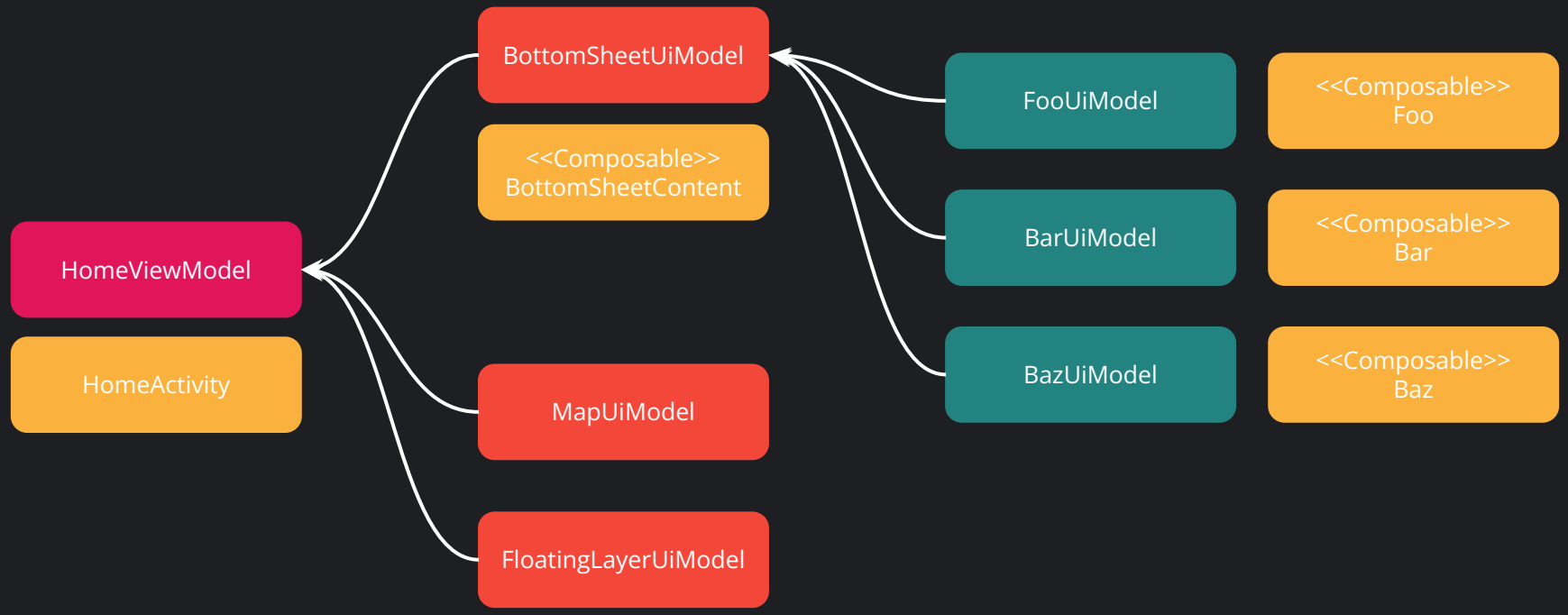
Micro-feature Host Integration - Compose

```
@Composable
fun BottomSheetContent(
    uiModel: BottomSheetUiModel,
    fooComposableFactory: FooComposableFactory,
    barComposableFactory: BarComposableFactory,
    bazComposableFactory: BazComposableFactory,
    modifier: Modifier = Modifier,
    showSnackbar: (String) → Unit,
) {
    ...
}
```

Micro-feature Host Integration - Compose

```
@Composable
fun BottomSheetContent(...) {

    val foo = remember { fooComposableFactory.create() }
    val bar = remember { barComposableFactory.create() }
    val baz = remember { bazComposableFactory.create() }

    Column(modifier) {
        foo(
            uiModel = uiModel.fooUiModel,
            showSnackbar = showSnackbar,
        )

        bar(
            uiModel = uiModel.barUiModel,
        )

        baz(
            uiModel = uiModel.bazUiModel,
        )
    }
}
```

Composition Tree



Micro-feature architecture

✅

- Good for highly dynamic apps with shared hosts
- Designed keeping host independence and portability in mind
- Enables working with several teams on a single screen in collaboration

❌

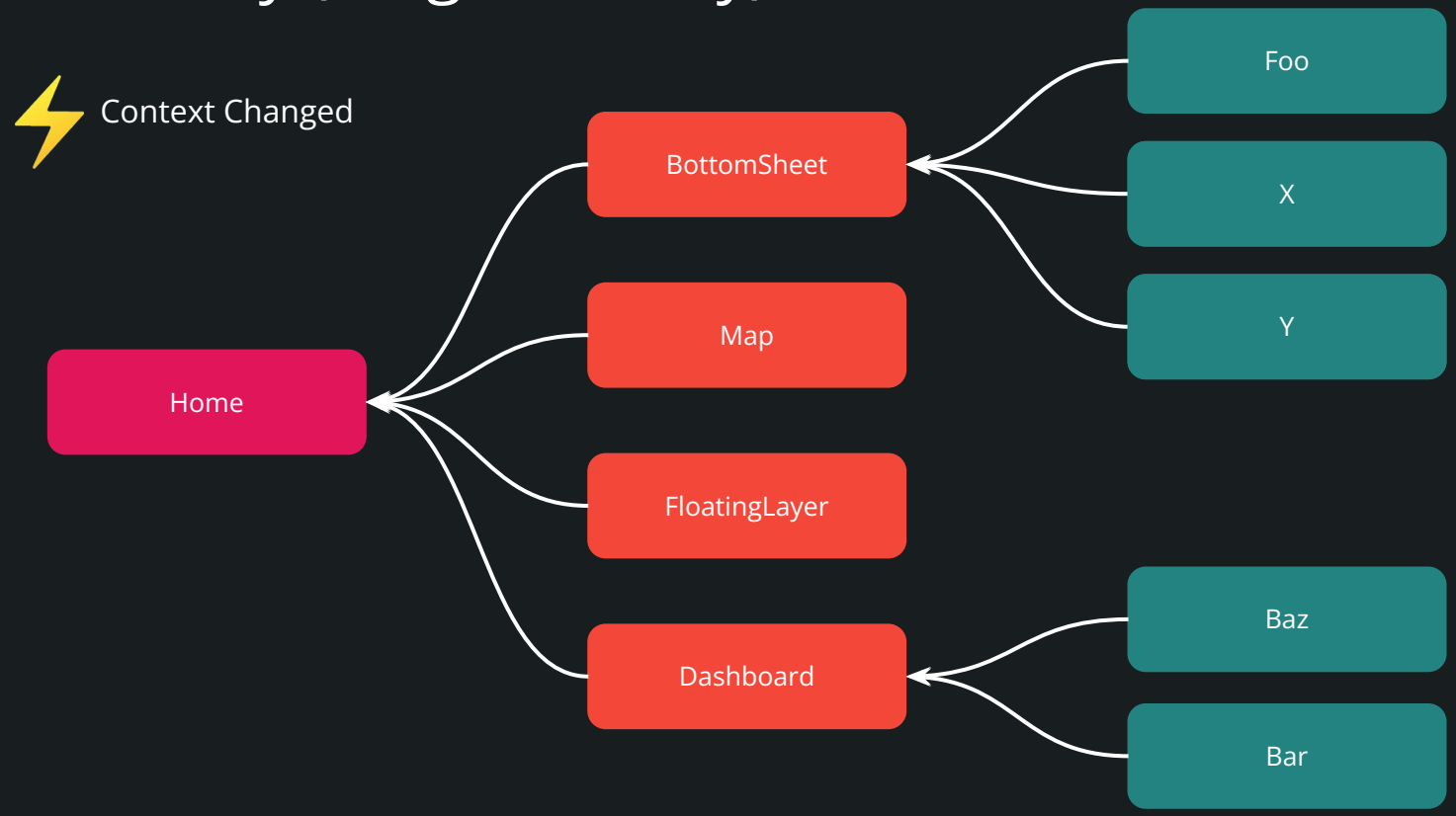
- Could be an overkill for small apps with static layout
- Could be a bit over-engineering if there is no need for host independence and portability
- Might not be the best choice for small teams

Micro-feature pitfalls

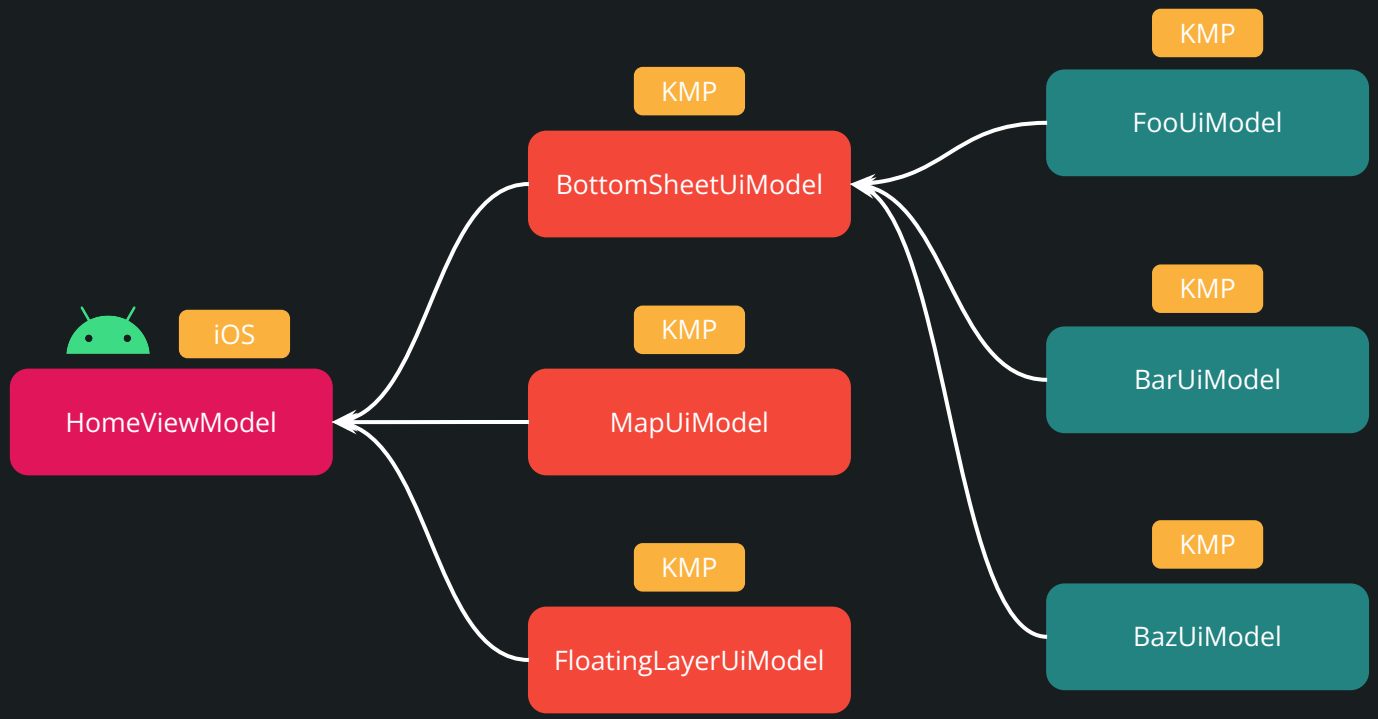
- Non-logical micro-features
- Over splitting a micro-feature
- Deep UI model hierarchies
- Not keeping host-independence in mind

Aspects of micro-features

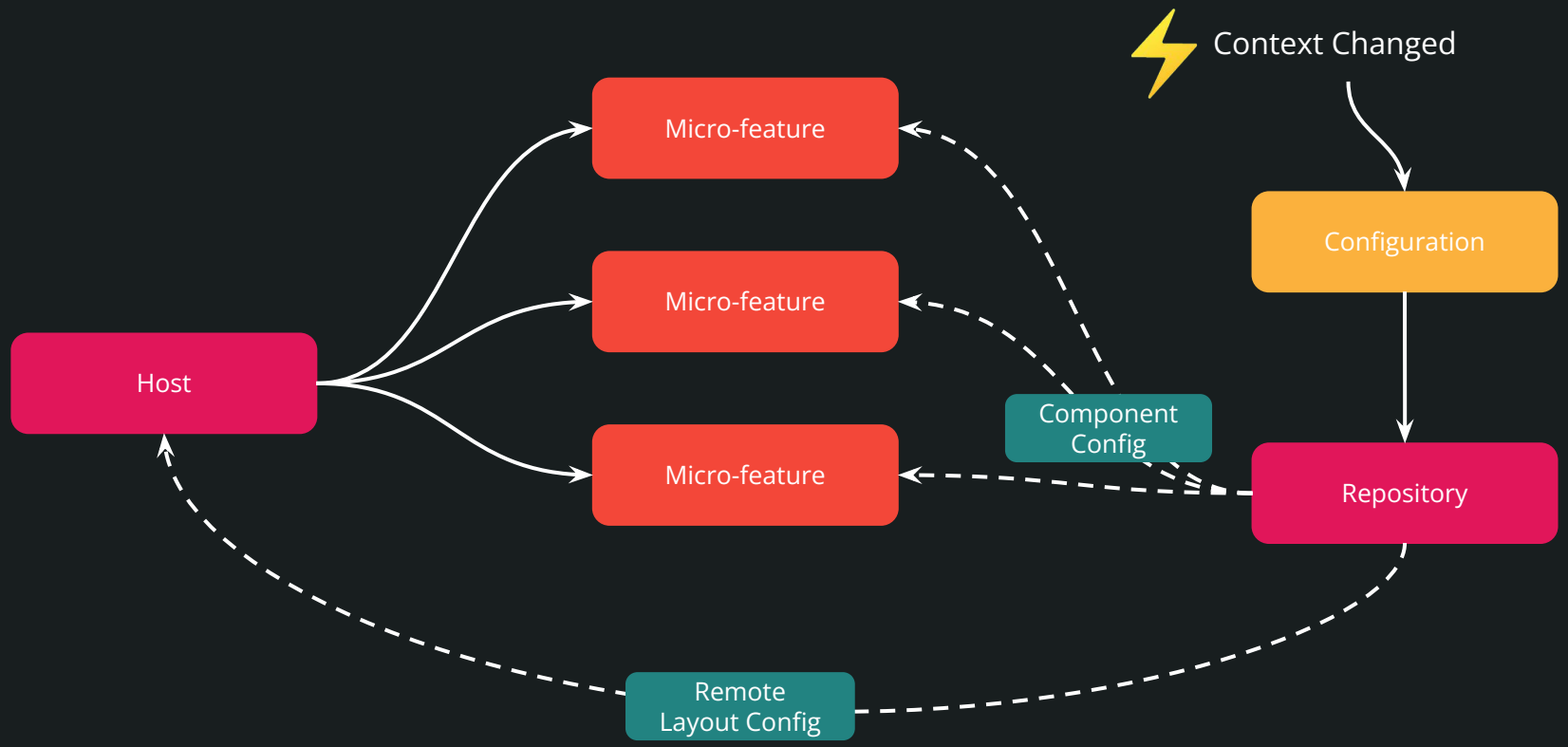
Portability (Plug-and-Play)



Multiplatform - UI Model



Server Driven UI



What’s next?

- Focusing on developer experience
- Sharing a KMM micro-feature between Android and iOS
- Moving towards server driven UI

Delivery Hero

Thank you!

Questions?

@hknbgcdev