Unidirectional Data Flow in Compose

ViewModel and Navigation

ДВФУ · ИМКТ · Департамент МиКМ · Android-разработка · осень-2025

Agenda

- Thinking in Compose(recap)
- ViewModel
- Navigation Compose

Outcome

- State holder -> ViewModel
- If/else экраны -> Navigation Compose
- Всё ещё без сети и БД

Recap: Thinking in Compose

- @Composable функции
- UI = функция от state
- Обновление UI происходит через recomposition
- Удобная формула: data down, events up

UDF: “state вниз, события вверх”

- Вниз: UiState (данные для отрисовки)
- Вверх: Events (коллбеки/методы)
- Плюсы:
  - предсказуемо
  - проще тестировать
  - UI становится “тупым”

State hoisting

- Hoist state к lowest common ancestor
- Из state owner наружу:
  - immutable state
  - events для изменения state

https://developer.android.com/develop/ui/compose/state-hoisting

ViewModel, Navigation - это просто инструменты, чтобы эту формулу сохранить на большом проекте.

State holder: “мозги” приложения

- State holder = место, где:
  - хранится UiState
  - обрабатываются события
  - вычисляются видимые списки / фильтры
- Реализация:
  - plain class
  - или ViewModel

remember / rememberSaveable / ViewModel

- remember: живёт пока composable остаётся в композиции
- rememberSaveable: переживает пересоздание, когда это возможно
- ViewModel: основной способ отдавать UI state в Compose

remember / rememberSaveable / ViewModel

- rememberSaveable: удобен для простого UI-state (в пределах экрана)
- ViewModel: screen-level state holder, переживает конфиг-изменения

plain class -> ViewModel

- Было: val holder = remember { Holder() }
- Стало: val holder: Holder = viewModel()
- И class Holder : ViewModel()

Но можно же в rememberSaveable plain class?

- **rememberSaveable** складывает данные в saved instance state (Bundle). Это “общий мешок” для всего Activity, у него есть жёсткие ограничения по размеру, и Google прямо предупреждает: не хранить большие/сложные объекты и списки, иначе можно словить TransactionTooLarge. Вместо этого хранить минимум (id/keys) и восстанавливать остальное другим способом.

https://developer.android.com/develop/ui/compose/state-saving

А зачем это вообще?

- Plain class = “состояние живёт, пока живёт composable”.
- ViewModel = “состояние живёт, пока живёт экран (и переживает поворот)

Где должна жить логика

- UI:
  - рисует
  - вызывает callbacks
- State holder / ViewModel:
  - решает “что делать”
  - меняет state

Navigation

Почему нам нужна навигация

- if (selectedId == null) … else … - ок для демо
- Но дальше нужно:
  - back stack
  - аргументы
  - единое описание графа экранов
  - depplink
- Решение: Navigation Compose

https://developer.android.com/develop/ui/compose/navigation

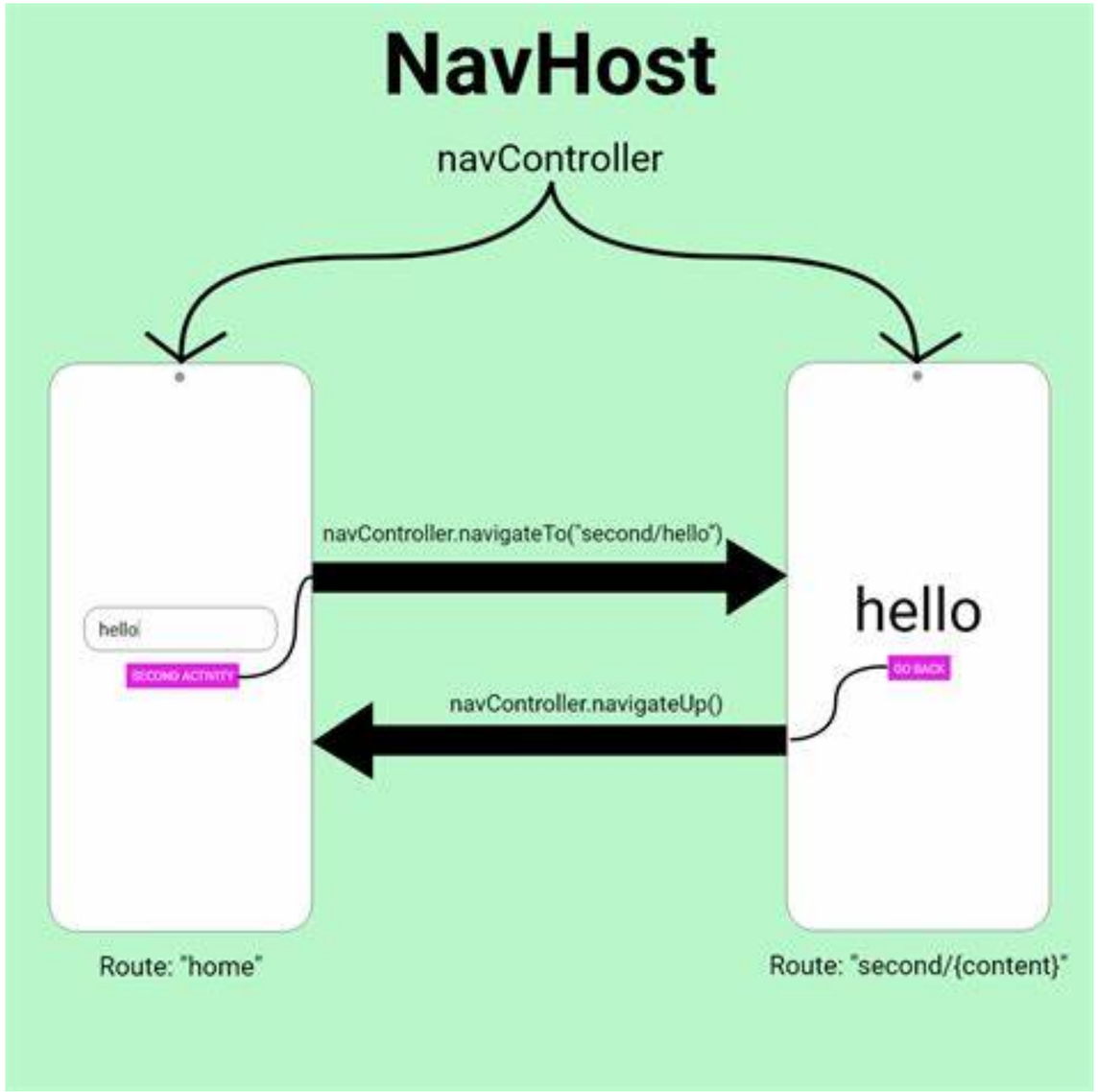
Navigation Compose: словарь

- NavController - управляет переходами
- NavHost - граф навигации
- route - строка маршрута (например, "detail/{id}")

https://developer.android.com/develop/ui/compose/navigation

Аргументы: передаём ID, не объект

- ✅ detail/7
- ❌ “передать весь Anime”
  - Почему:
  - проще
  - безопаснее
  - меньше связанности



Back stack и “назад”

- navigate("detail/7") -> пуш в стек
- “назад” -> popBackStack()
- “вверх” (up) - обычно то же самое в простом кейсе