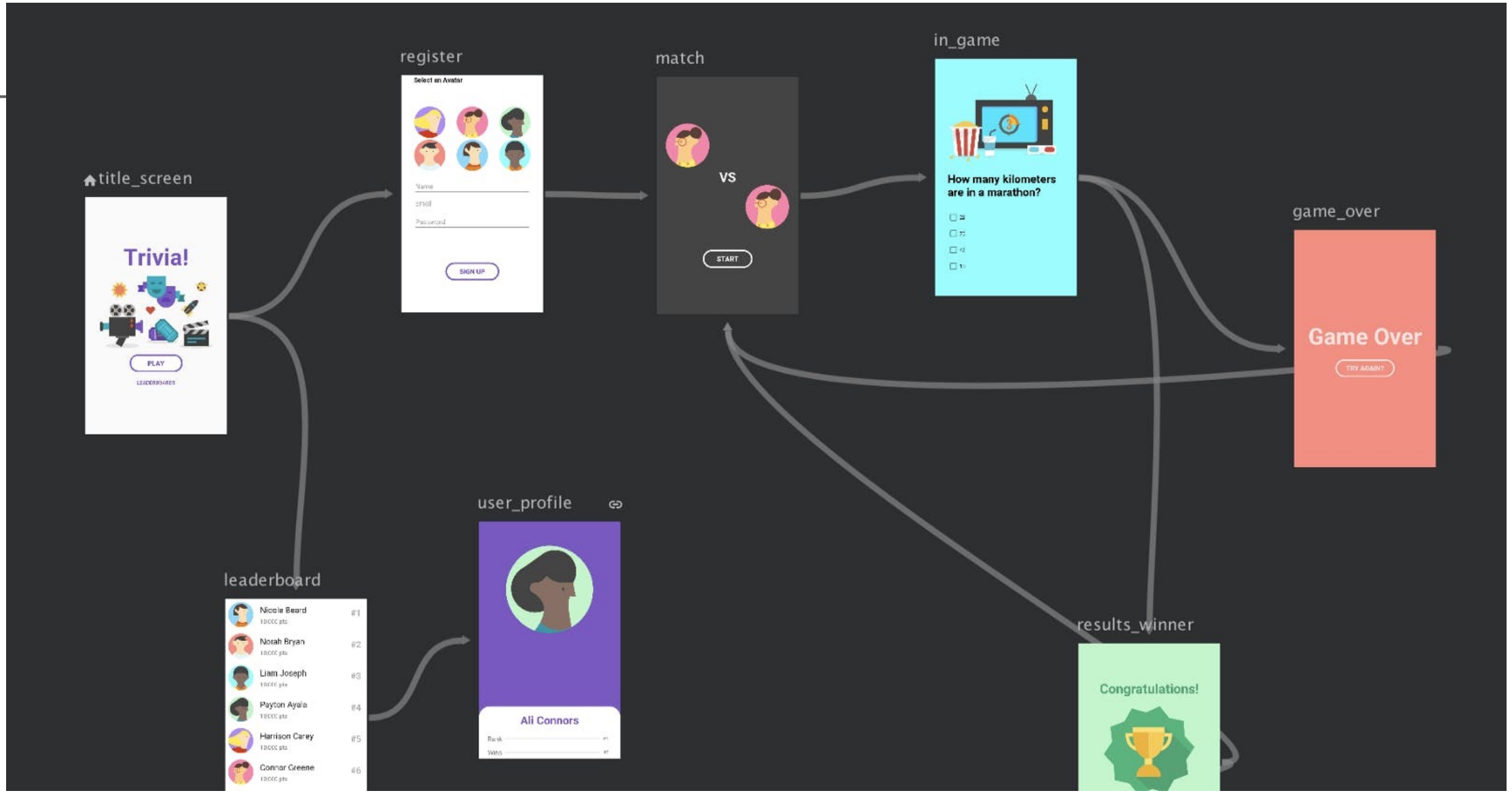
Navigation Component

Lezione 8

Navigation

- *La navigazione* fa riferimento alle interazioni che consentono agli utenti di spostarsi, entrare e uscire dai diversi contenuti all'interno dell'app
- Il componente di navigazione (Navigation component) di Android Jetpack ti aiuta a implementare la navigazione, dai semplici clic sui pulsanti a schemi più complessi, come le barre delle app e il cassetto di navigazione
- Il componente Navigation garantisce inoltre un'esperienza utente coerente e prevedibile aderendo a una serie di principi di navigazione ben definiti

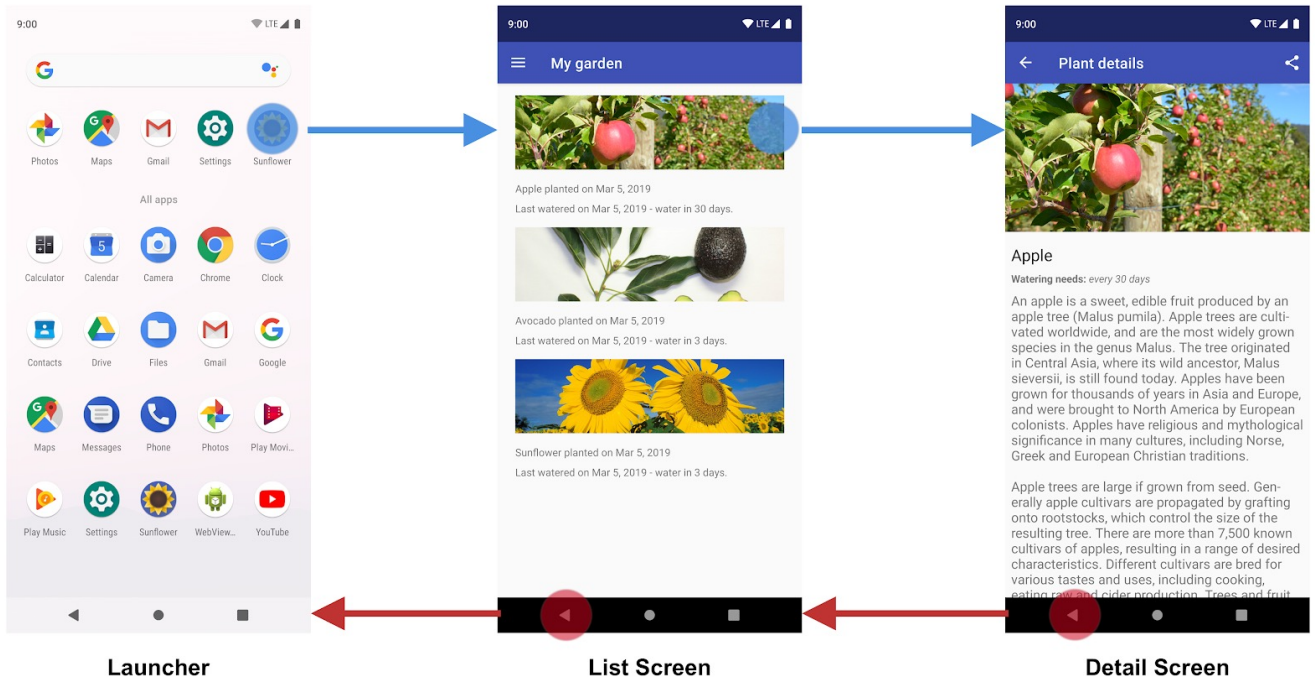

title_screen
Trivia!
PLAY
register
Select an Avatar
SIGN UP
match
VS
START
in_game
How many kilometers are in a marathon?
game_over
Game Over
TRY AGAIN?
leaderboard
Nicole Beard
Norah Bryan
Liam Joseph
Payton Ayala
Harrison Carey
Connor Greene
user_profile
Ali Connors
results_winner
Congratulations!

Principi di navigazione

- La navigazione tra diverse schermate e app è una parte fondamentale dell'esperienza dell'utente
- Il navigation component è progettato per implementare questi principi per impostazione predefinita, garantendo che gli utenti possano applicare gli stessi criteri euristici e gli stessi modelli nella navigazione mentre si spostano tra le app
- I principi presentati nelle prossime slide definiscono una linea di base per un'esperienza utente coerente e intuitiva tra le app

1. Destinazione di partenza fissa

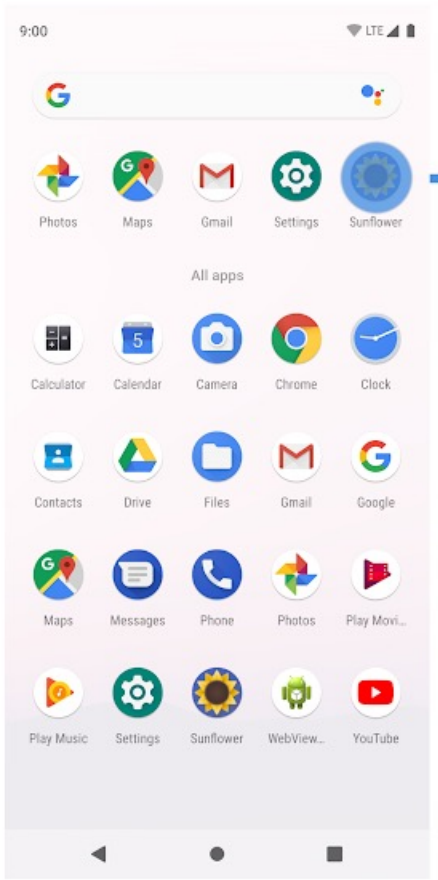
- Ogni app che si crea ha una destinazione iniziale fissa
  - Questa è la prima schermata che l'utente vede quando avvia la tua app dal programma di avvio
  - Questa destinazione è anche l'ultima schermata che l'utente vede quando torna al programma di avvio dopo aver premuto il pulsante Indietro



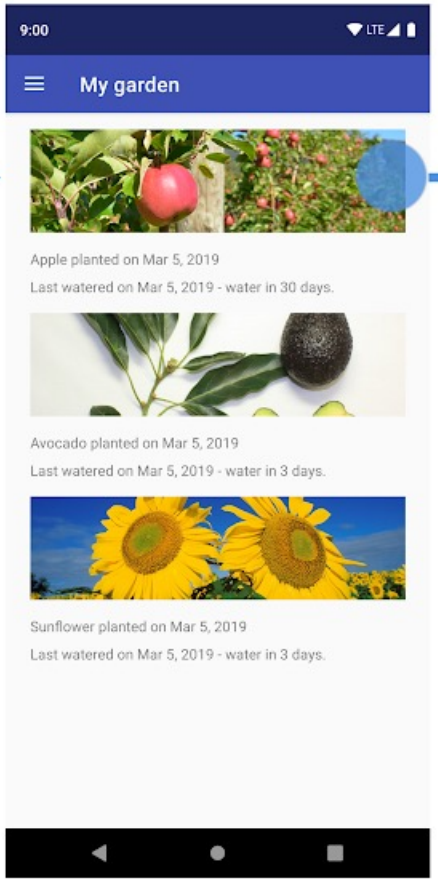
2. Lo stato di navigazione è rappresentato come una pila di destinazioni

- Quando l'app viene avviata per la prima volta, viene creata una nuova attività per l'utente e l'app visualizza la sua destinazione iniziale
- Questa diventa la destinazione di base di ciò che è noto come back stack ed è la base per lo stato di navigazione della tua app
  - La parte superiore dello stack è la schermata corrente e le destinazioni precedenti nello stack rappresentano la cronologia delle schermate in cui sei stato
  - Il back stack ha sempre la destinazione iniziale dell'app nella parte inferiore dello stack
  - Le operazioni che modificano il back stack operano sempre in cima allo stack, spingendo una nuova destinazione in cima allo stack o estraendo la destinazione più in alto dallo stack
  - La navigazione verso una destinazione spinge tale destinazione in cima allo stack
- Il componente Navigation **gestisce per te** tutti gli ordini del tuo back stack, sebbene tu possa anche scegliere di gestire tu stesso il back stack

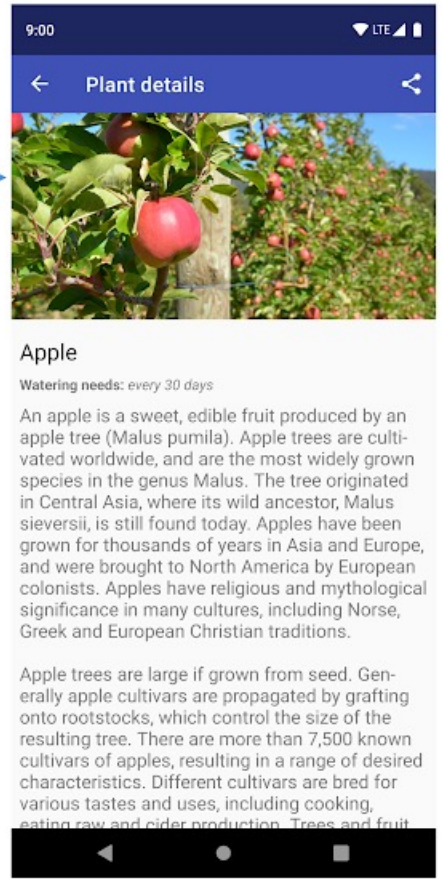
Organic Navigation through Sunflower (Browsing to apple details)



Launch Screen

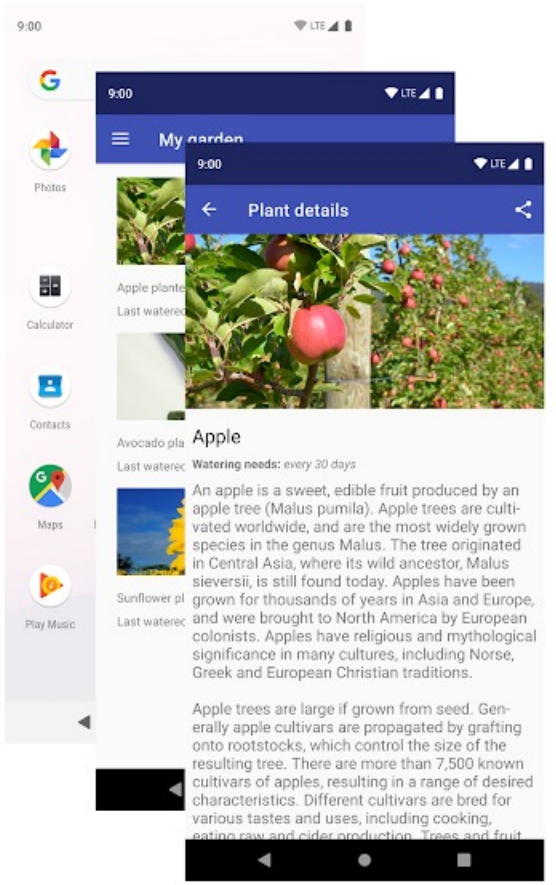


List Screen



Detail Screen

Resulting Sunflower Task Back Stack



3. Up e back sono identici all'interno dell'attività della tua app

- Il pulsante Back viene visualizzato nella barra di navigazione del sistema nella parte inferiore dello schermo e viene utilizzato per navigare in ordine cronologico inverso attraverso la cronologia delle schermate con cui l'utente ha lavorato di recente
  - Quando si preme il pulsante Back, la destinazione corrente viene estratta dalla parte superiore del back stack e quindi si naviga verso la destinazione precedente
- Il pulsante Up viene visualizzato nella barra dell'app nella parte superiore dello schermo
- All'interno dell'attività dell'app, i pulsanti Up e Back si comportano in modo identico

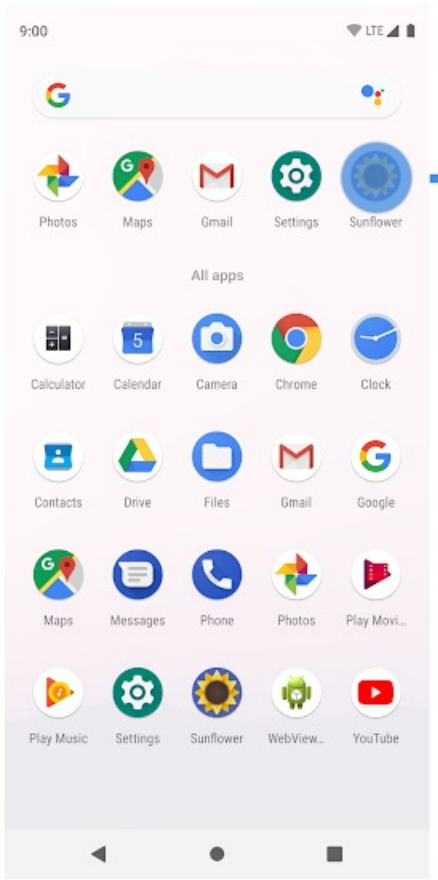
4. Il pulsante Up non chiude mai l'app

- Se un utente si trova nella destinazione iniziale dell'app, il pulsante Up non viene visualizzato perché il pulsante UP non permette mai di uscire dall’app
  - Il pulsante Back, tuttavia, viene visualizzato e chiude l’app
- Quando la tua app viene avviata utilizzando un collegamento diretto sull’activity di un'altra app, Up riporta gli utenti all’activity della tua app attraverso un back stack simulato e non all'app che ha attivato il collegamento diretto
  - Il pulsante Back, tuttavia, ti riporta all'altra app

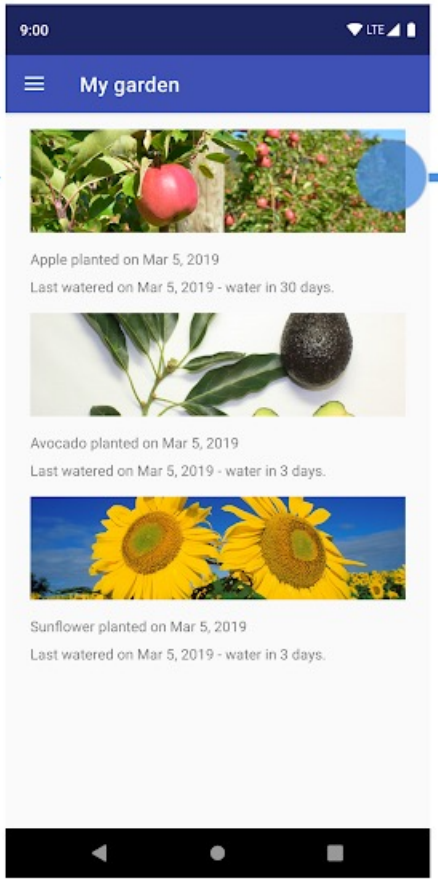
5. Il deep linking simula la navigazione manuale

- Che si tratti di deep linking (collegamento – URL - che porta direttamente a una destinazione specifica- activity - all'interno di un'app) o navigazione manuale verso una destinazione specifica, si può utilizzare il pulsante Up per navigare tra le destinazioni fino alla destinazione di partenza
- Quando si crea un collegamento diretto (deep linking) a una destinazione all'interno dell'attività dell'app, qualsiasi back stack esistente per l'attività dell'app viene rimosso e sostituito con lo stack back collegato in modo diretto

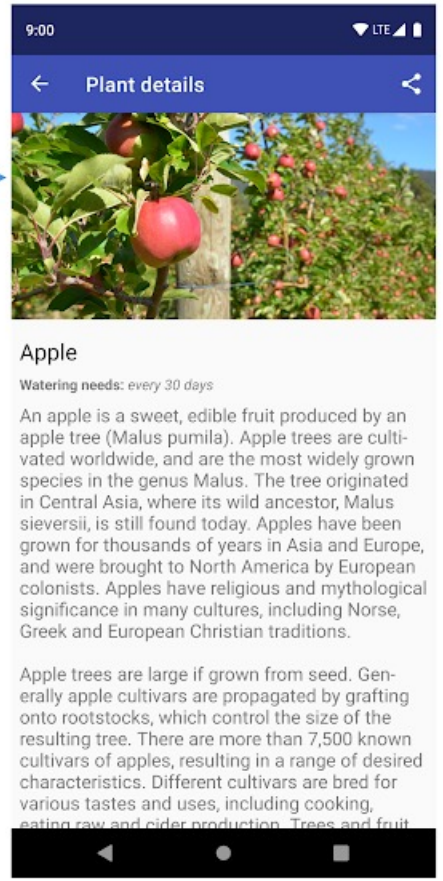
Organic Navigation through Sunflower (Browsing to apple details)



Launch Screen

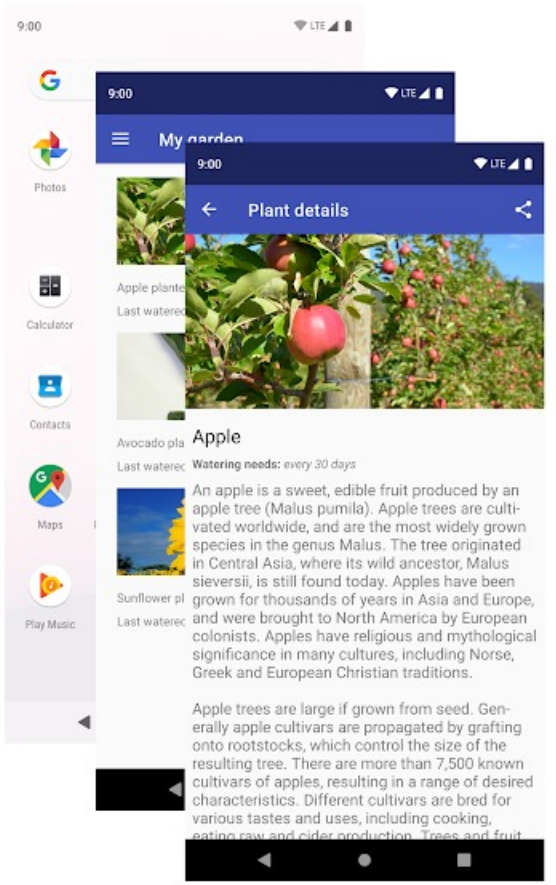


List Screen

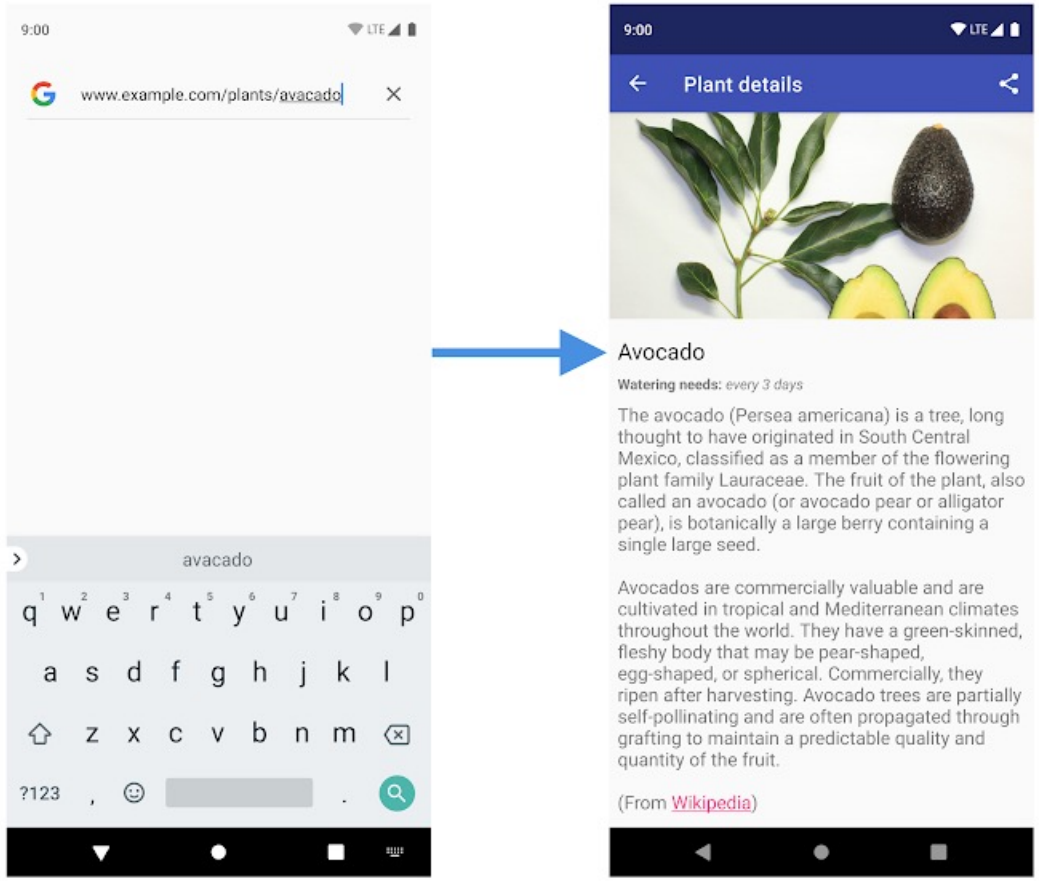


Detail Screen

Resulting Sunflower Task Back Stack



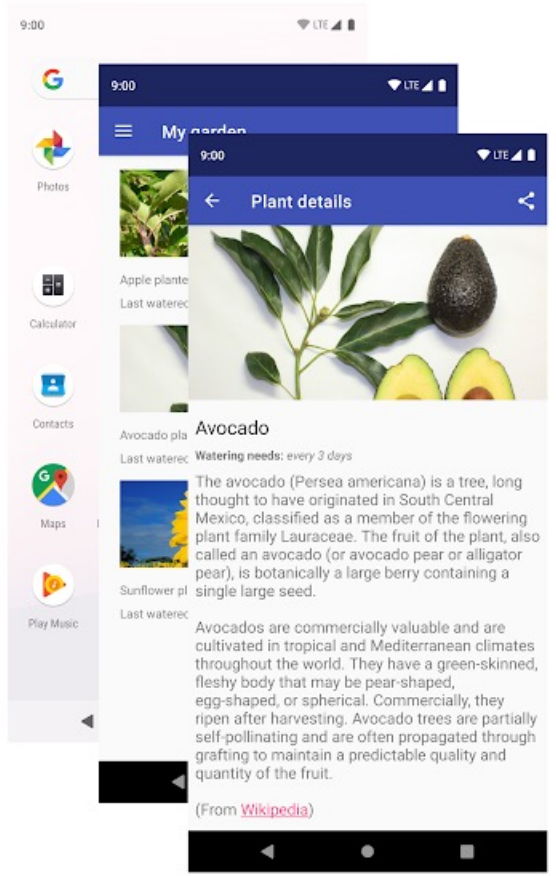
Deep Link into Plant Details (Linking to avacado details)



Follow Deep Link

Detail Screen

Resulting Sunflower Task Back Stack (after user enters through deep link)



Navigation component

- Il Navigation component fornisce supporto per le applicazioni Jetpack Compose
- È possibile navigare tra i composable sfruttando l'infrastruttura e le funzionalità del Navigation component

Navigation component

- E’ composto da tre parti principali:
  - *Grafico di navigazione*: contiene tutte le informazioni relative alla navigazione
    - Ciò include tutte le singole aree di contenuto all'interno della tua app, denominate destinazioni, nonché i possibili percorsi che un utente può intraprendere attraverso la tua app
  - NavHost: un contenitore vuoto che mostra le destinazioni del tuo grafico di navigazione
  - NavController: un oggetto che gestisce la navigazione dell'app all'interno di un NavHost
    - Il NavController orchestra lo scambio del contenuto di destinazione nel NavHost man mano che gli utenti si spostano all'interno dell'app

Setup

- Per supportare **Compose**, utilizzare la seguente dipendenza nel file *build.gradle* del modulo dell'applicazione:

```
dependencies {
    def nav_version = "2.7.7"

    implementation "androidx.navigation:navigation-
compose:$nav_version"
}
```

Getting started

- **NavController è l'API centrale del componente Navigation**
  - È statico e tiene traccia nel back stack delle schermate dell'applicazione e dello stato di ciascuna schermata

- È possibile creare un *NavController* utilizzando il metodo rememberNavController() nel composable:

```
val navController = rememberNavController()
```

Getting started

- Si dovrebbe creare il NavController in un punto della gerarchia dei composable in cui tutti i composable che devono farvi riferimento abbiano accesso
- Questo segue i principi di state hoisting e consente di utilizzare il NavController e lo stato che fornisce tramite currentBackStackEntryAsState() come *fonte di verità (single source of truth)* per l'aggiornamento dei composable

Creare un NavHost

- Ogni *NavController* deve essere associato ad un singolo *NavHost composable*
- Il NavHost collega il NavController con un grafo di navigazione che specifica le destinazioni composable tra le quali dovresti essere in grado di navigare
- Durante la navigazione tra i composable, il contenuto del NavHost viene ricomposto automaticamente
- Ogni destinazione composable nel grafo di navigazione è associata a un percorso (route)
  - Route è una stringa che definisce il percorso del tuo composable che conduce a una destinazione specifica
  - Ogni destinazione DEVE avere un percorso unico

Creare un NavHost

- La creazione del NavHost richiede il NavController precedentemente creato tramite **RememberNavController()** e il percorso della destinazione iniziale del grafo
- Puoi **aggiungere elementi alla tua struttura di navigazione** utilizzando il metodo composable()
  - Questo metodo richiede di fornire un percorso e il composable che dovrebbe essere collegato alla destinazione:

```
NavHost(navController = navController, startDestination = "profile") {
    composable("profile") { Profile(/*...*/) }
    composable("friendslist") { FriendsList(/*...*/) }
    /*...*/
}
```

Nota!

- Ricordatevi che il Navigation component richiede di seguire i principi di navigazione e utilizzare una destinazione di partenza fissa!
  - Principio 1 visto all'inizio delle slide

Navigare verso un composable

- Per navigare verso una destinazione composable nel grafo di navigazione, è necessario utilizzare il metodo *navigate*
- *navigate* accetta un singolo parametro String che rappresenta il percorso della destinazione

```
navController.navigate("friendslist")
```

Navigare verso un composable: modifica back stack

- Per impostazione predefinita, Navigation aggiunge la tua nuova destinazione al back stack
- Puoi modificare il comportamento di *navigate* aggiungendo ulteriori opzioni di navigazione alla chiamata *navigate()*
  - Ovvero PopUpTo() che permette di estrarre destinazioni aggiuntive dal back stack

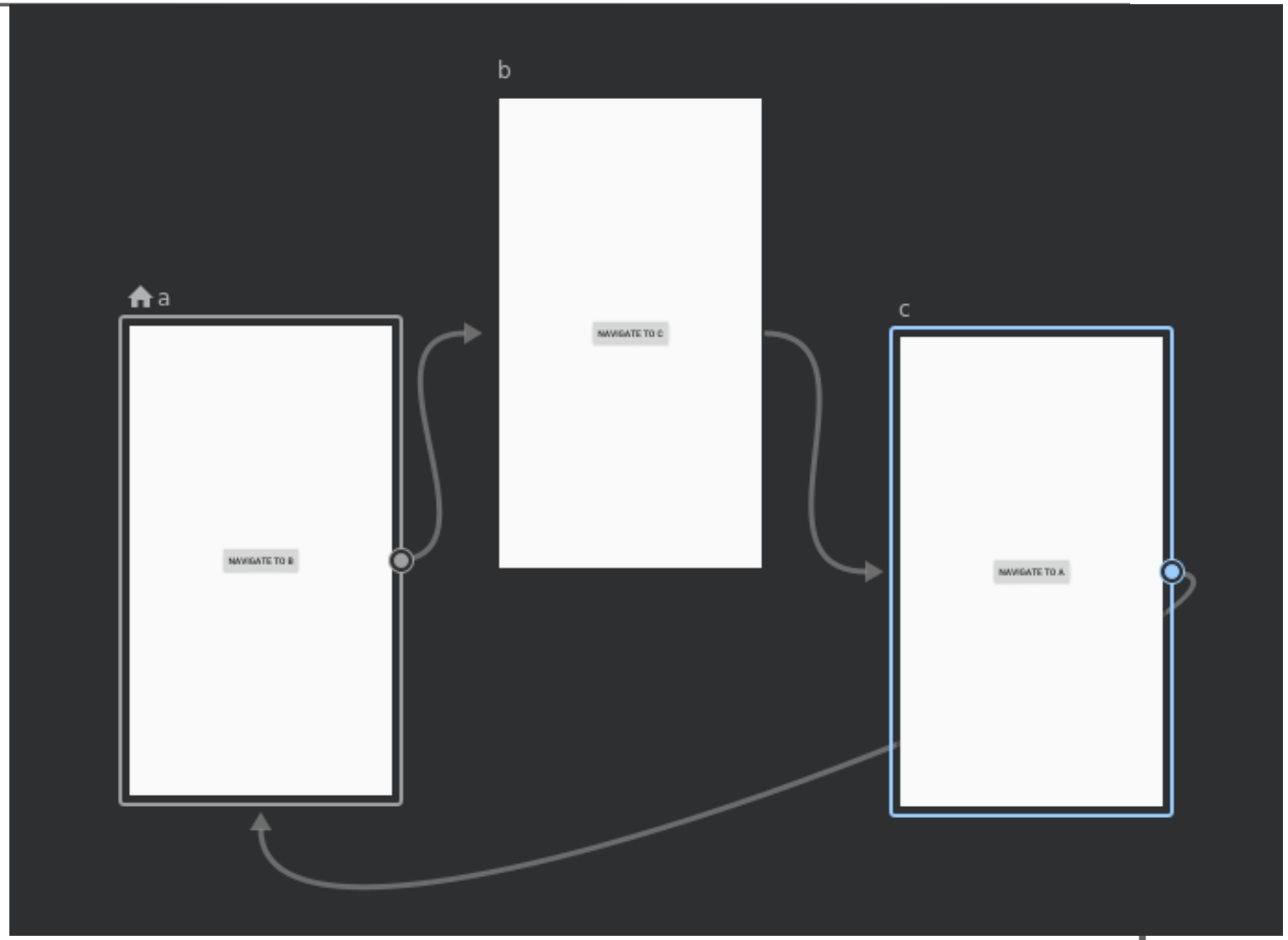
Esempio

```
// Pop everything up to the "home" destination off the back stack before
// navigating to the "friendslist" destination
navController.navigate("friendslist") {
    popUpTo("home")
}

// Pop everything up to and including the "home" destination off
// the back stack before navigating to the "friendslist" destination
navController.navigate("friendslist") {
    popUpTo("home") { inclusive = true }
}

// Navigate to the "search” destination only if we’re not already on
// the "search" destination, avoiding multiple copies on the top of the
// back stack
navController.navigate("search") {
    launchSingleTop = true
}
```

https://developer.android.com/guide/navigation/navigation-navigate#pop

Approfondimento: **popUpTo**

- popUpTo permette di rimuovere dal back stack tutte le destinazioni sopra quella definita
- Con *inclusive*, incluso quella definita (nell'esempio prima Home)
- Questo permette di evitare il loop che si vede in figura
- Pensate al login..



Chiamate navigate() attivate da altre funzioni composable

- La funzione navigate() del NavController modifica lo stato interno del NavController
- Per conformarsi il più possibile al principio dell'unica fonte di verità, solo la funzione composable o lo state holder dell'istanza di NavController e quelle funzioni composable che accettano il NavController come parametro dovrebbero effettuare chiamate di navigation
- Gli eventi di navigazione attivati da altre funzioni composable inferiori nella gerarchia dell'interfaccia utente (es. un bottone) devono esporre tali eventi al chiamante in modo appropriato

Esempio

```
@Composable
fun MyAppNavHost(
    modifier: Modifier = Modifier,
    navController: NavHostController = rememberNavController(),
    startDestination: String = "profile"
) {
    NavHost(
        modifier = modifier,
        navController = navController,
        startDestination = startDestination
    ) {
        composable("profile") {
            ProfileScreen(    //genera la schermata del profilo
                onNavigateToFriends = { navController.navigate("friendsList") },
                /*...*/
            )
        }
        composable("friendslist") { FriendsListScreen(/*...*/) }
    }
}

@Composable
fun ProfileScreen(
    onNavigateToFriends: () -> Unit,
    /*...*/
) {
    /*...*/
    Button(onClick = onNavigateToFriends) {
        Text(text = "See friends list")
    }
}
```

- L'esempio mostra la funzione composable MyAppNavHost come singola origine attendibile (single source of truth) per l'istanza NavController
- ProfileScreen espone un evento come una funzione che viene chiamata quando l'utente interagisce con un pulsante
- MyAppNavHost, che possiede la navigazione verso le diverse schermate nell'app, effettua la chiamata di navigazione verso la destinazione corretta quando si chiama ProfileScreen

Navigate con argomenti

- Navigation Compose supporta anche il passaggio di argomenti tra destinazioni composable
- Per fare ciò, devi aggiungere argument placeholders al tuo percorso (route)

```
NavHost(startDestination = "profile/{userId}") {
    ...
    composable("profile/{userId}") {...}
}
```

Navigate con argomenti

- Per impostazione predefinita, tutti gli argomenti vengono analizzati come stringhe
- Il parametro arguments di composable() accetta un elenco di NamedNavArguments
  - Puoi creare un NamedNavArgument utilizzando il metodo navArgument e quindi specificarne il tipo esatto:

```
NavHost(startDestination = "profile/{userId}") {
    ...
    composable(
        "profile/{userId}",
        arguments = listOf(navArgument("userId") { type = NavType.StringType })
    ) {...}
}
```

Navigate con argomenti

- È necessario estrarre gli argomenti da NavBackStackEntry (rappresentazione di un entry nel back stack) disponibili nel lambda della funzione composable()

```
composable("profile/{userId}") { backStackEntry ->
    Profile(navController,
backStackEntry.arguments?.getString("userId"))
}
```

Navigate con argomenti

- Per passare l'argomento alla destinazione, devi aggiungerlo alla rotta quando effettui la chiamata di navigazione:

```
navController.navigate("profile/user1234")
```

Recupero di dati complessi durante la navigazione

- Si consiglia fortemente di non passare oggetti di dati complessi durante la navigazione, ma di passare invece le informazioni minime necessarie, come un identificatore univoco o altra forma di ID, come argomenti durante l'esecuzione delle azioni di navigazione:

```
// Pass only the user ID when navigating to a new destination as argument
navController.navigate("profile/user1234")
```

Recupero di dati complessi durante la navigazione

- Gli oggetti complessi dovrebbero essere archiviati come dati in *un'unica fonte di verità*, come il livello Data
- Una volta nella destinazione dopo la navigazione, puoi quindi caricare le informazioni richieste dall'unica fonte di verità utilizzando l'ID trasmesso
- Per recuperare gli argomenti useremo il *ViewModel* (ne parleremo in modo dettagliato nella prossima lezione!) responsabile dell'accesso al livello dati
- Questo approccio aiuta a prevenire la perdita di dati durante le modifiche alla configurazione e qualsiasi incoerenza quando l'oggetto in questione viene aggiornato o modificato

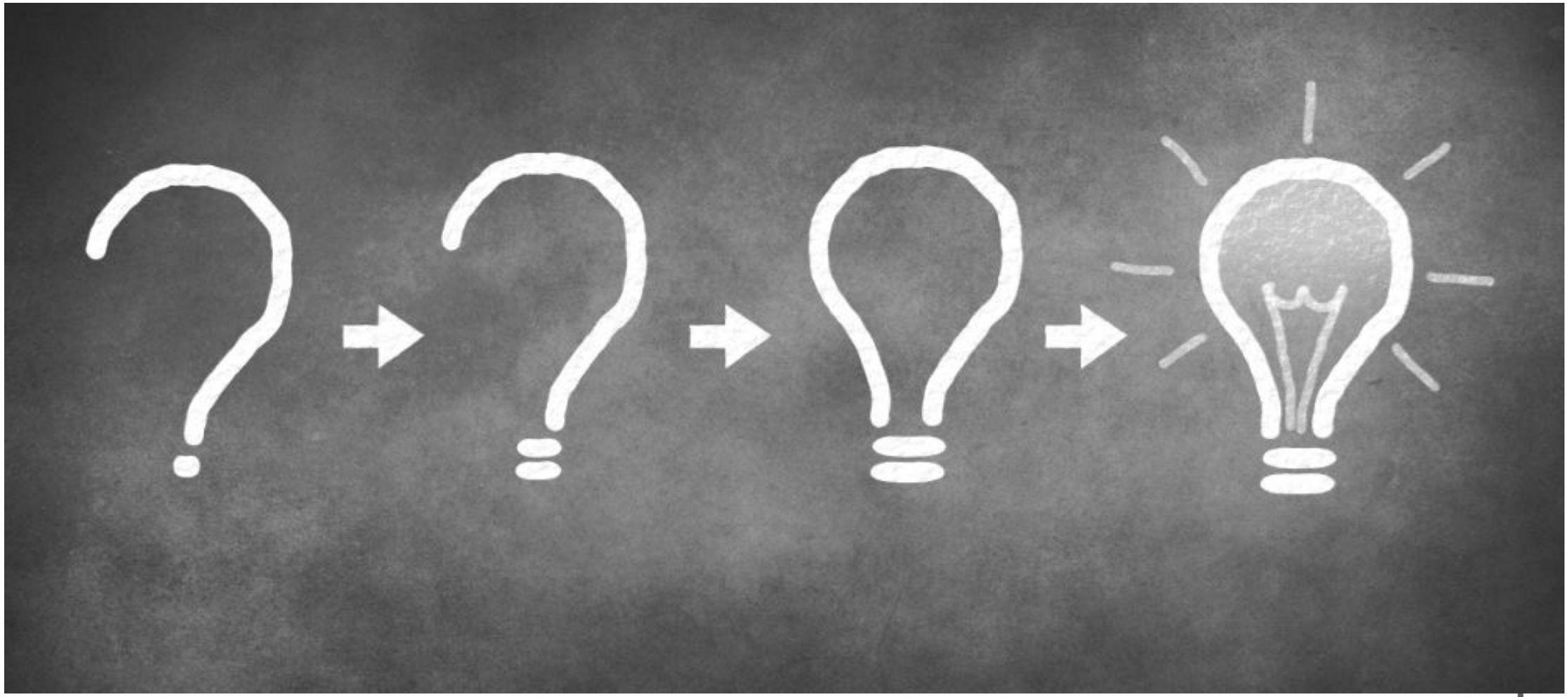
Aggiunta di argomenti facoltativi

- Navigation Compose supporta anche argomenti di navigazione facoltativi
- Gli argomenti facoltativi differiscono dagli argomenti obbligatori in due modi:
  - Devono essere inclusi utilizzando la sintassi dei parametri di query ("?argName={argName}") – invece che con /
  - Devono avere un defaultValue impostato o avere nullability = true (che imposta implicitamente il valore predefinito su null)
    - Ciò significa che tutti gli argomenti facoltativi devono essere aggiunti esplicitamente alla funzione composable() come liste
- La struttura della gestione degli argomenti attraverso i percorsi significa che i tuoi composable rimangono completamente indipendenti dalla navigazione e li rende molto più testabili

Aggiunta di argomenti facoltativi

```
composable(
    "profile?userId={userId}",
    arguments = listOf(navArgument("userId") { defaultValue = "user1234" })
) {
backStackEntry ->
    Profile(navController, backStackEntry.arguments?.getString("userId"))
}
```

Domande?



Riferimenti

- https://developer.android.com/jetpack/compose/navigation