Navigation

Esempi di utilizzo

Recap: Navigazione in Compose

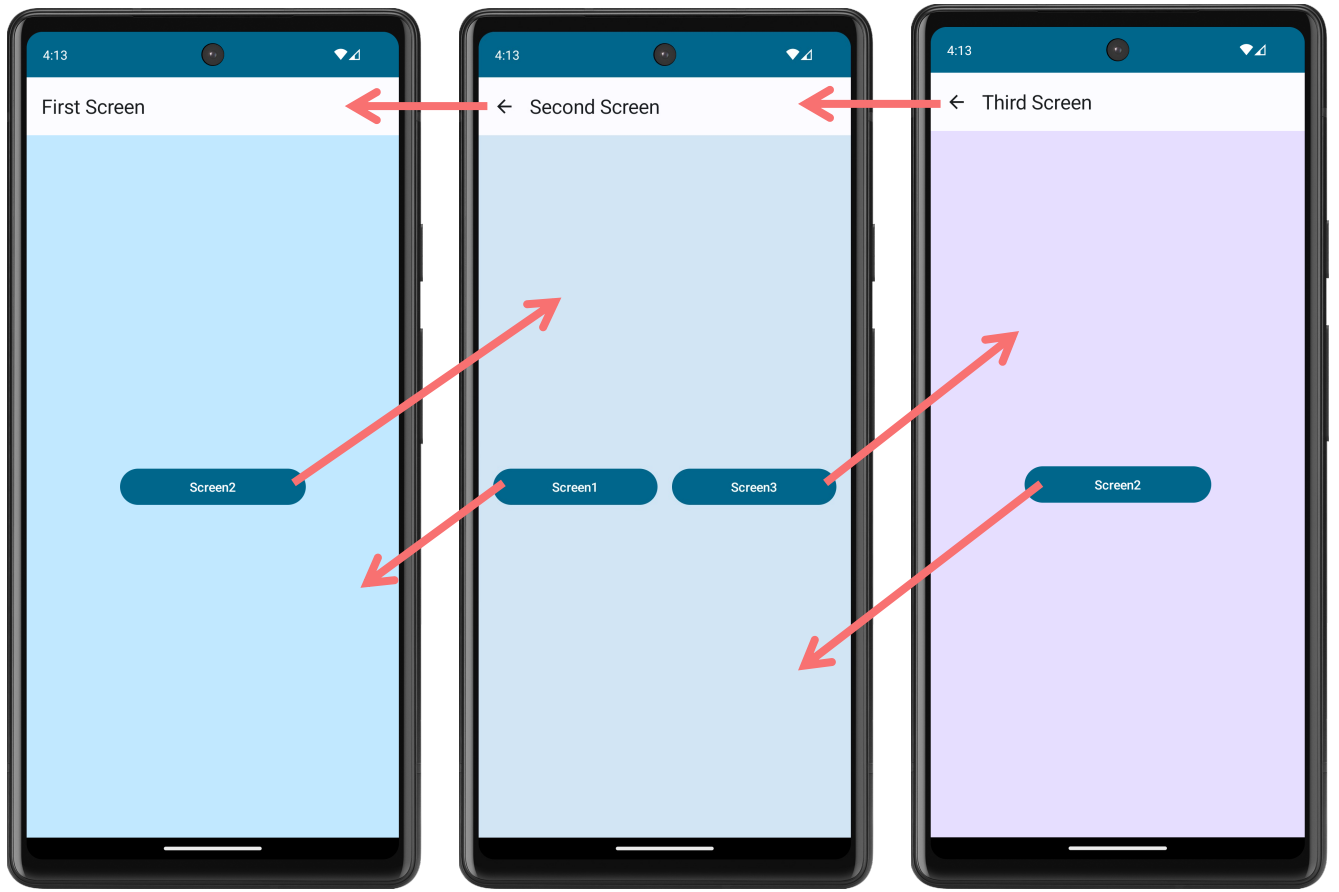
- Un progetto Compose che implementa le funzionalità di navigazione è dotato di tre componenti:
  - **Grafo di navigazione**: definisce l'elenco delle destinazioni raggiungibili tramite navigazione all'interno dell'app, indicando per ciascuna il composable che ne contiene la UI
  - **`NavHost`**: è il composable utilizzato per l'implementazione del grafo di navigazione
  - **`NavController`**: è l'oggetto che gestisce la navigazione, tenendo traccia della destinazione attuale e fornendo metodi per muoversi tra le destinazioni definite nel grafo

Esercizi

1. Navigazione in Jetpack Compose
2. TravelDiary - Navigazione

1. Navigazione in Jetpack Compose

- Creare un'app Compose che permetta di navigare fra tre schermate, come nella figura
- Colori:
  - Prima schermata: `primaryContainer`
  - Seconda schermata: `secondaryContainer`
  - Terza schermata: `tertiaryContainer`



1. Ingredienti

- Installazione dipendenze
- Definizione delle destinazioni
- Creazione del grafo di navigazione
- Composable per l’interfaccia utente

1.1. Installazione dipendenze

- Nel blocco dependencies del file **`build.gradle.kts`** (modulo :app), aggiungere:

```
implementation("androidx.navigation:navigation-compose:2.7.7")
```

1.2. Definizione delle destinazioni

- Ogni destinazione è identificata da una stringa univoca detta rotta, o **route**, che può essere passata al **`NavController`** per navigare verso la schermata corrispondente
- Per un miglior livello di organizzazione e safety, è consigliabile definire l'elenco di rotte in una **`sealed class`** o un **`enum`**:

```
sealed class NavigationRoute(
    val route: String
) {
    data object Screen1 : NavigationRoute("screen1")
    data object Screen2 : NavigationRoute("screen2")
    data object Screen3 : NavigationRoute("screen3")
}
```

1.3. Creazione del grafo di navigazione

```
@Composable
fun NavGraph(
    navController: NavHostController,
    modifier: Modifier = Modifier
) {
    NavHost(
        navController = navController,
        startDestination = NavigationRoute.Screen1.route,
        modifier
    ) {
        with(NavigationRoute.Screen1) {
            composable(route) { Screen1(navController) }
        }
        with(NavigationRoute.Screen2) {
            composable(route) { Screen2(navController) }
        }
        with(NavigationRoute.Screen3) {
            composable(route) { Screen3(navController) }
        }
    }
}
```

1.3. Creazione del grafo di navigazione

```
@Composable
fun NavGraph(
    navController: NavHostController,
    modifier: Modifier = Modifier
) {
    NavHost(
        navController = navController,
        startDestination = NavigationRoute.Screen1.route,
        modifier
    ) {
        with(NavigationRoute.Screen1) {
            composable(route) { Screen1(navController) }
        }
        with(NavigationRoute.Screen2) {
            composable(route) { Screen2(navController) }
        }
        with(NavigationRoute.Screen3) {
            composable(route) { Screen3(navController) }
        }
    }
}
```

Non scriviamo mai le destinazioni come stringhe, ma vi accediamo tramite la sealed class definita in precedenza

1.3. Creazione del grafo di navigazione

```
@Composable
fun NavGraph(
    navController: NavHostController,
    modifier: Modifier = Modifier
) {
    NavHost(
        navController = navController,
        startDestination = NavigationRoute.Screen1.route,
        modifier
    ) {
        with(NavigationRoute.Screen1) {
            composable(route) { Screen1(navController) }
        }
        with(NavigationRoute.Screen2) {
            composable(route) { Screen2(navController) }
        }
        with(NavigationRoute.Screen3) {
            composable(route) { Screen3(navController) }
        }
    }
}
```

È possibile personalizzare il composable tramite modifier (vedi slide successiva)

Best practice: Modifier

- La maggior parte dei composable accetta un parametro **`modifier`** opzionale, tramite il quale il chiamante può personalizzare alcuni aspetti estetici del componente
- È buona pratica seguire questa convenzione quando si creano composable custom

1.4. Creazione del NavController

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

Nota: remember

- Una funzione **`@Composable`** viene rieseguita ogni volta che i suoi parametri vengono modificati
- Per questo motivo, è preferibile non effettuare computazioni costose all'interno dei composable
- Come accennato in teoria, la funzione **`remember`** permette ai composable di computare il valore al suo interno solo durante la prima composition, per poi salvarlo in memoria e restituirlo in tutte le recomposition successive

```
val value = remember { someExpensiveFunctionReturningAValue() }
```

Eseguita solo una volta

Nota: remember

- Il meccanismo di **`remember`** è utilizzato dalle funzioni di moltissime librerie per Jetpack Compose
  - La convenzione è di prefissare il nome di queste funzioni con la parola *remember*, per indicare l'utilizzo interno di questa feature
- Un'esempio è appunto **`rememberNavController`**, che restituisce un oggetto **`NavController`** e ne evita la ricreazione ad ogni recomposition

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

1.5. Composable per l’interfaccia utente

MainActivity

```
setContent {
    MyApplicationTheme {
        Surface(
            modifier = Modifier.fillMaxSize(),
            color = MaterialTheme.colorScheme.background
        ) {
            val navController = rememberNavController()

            Scaffold(
                topBar = { AppBar(navController) },
            ) { contentPadding ->
                NavGraph(navController, modifier = Modifier.padding(contentPadding))
            }
        }
    }
}
```

1.5. Composable per l’interfaccia utente

AppBar

```
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun AppBar(navController: NavHostController) {
    val backStackEntry by navController.currentBackStackEntryAsState()
    val title = when (backStackEntry?.destination?.route) {
        "screen1" -> "First Screen"
        "screen2" -> "Second Screen"
        "screen3" -> "Third Screen"
        else -> "Unknown Screen"
    }

    TopAppBar(
        title = { Text(title) },
        navigationIcon = {
            if (navController.previousBackStackEntry != null) {
                IconButton(onClick = { navController.navigateUp() }) {
                    Icon(Icons.Outlined.ArrowBack, "Go Back")
                }
            }
        },
    )
}
```

1.5. Composable per l'interfaccia utente

AppBar

```
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun AppBar(navController: NavHostController) {
    val backStackEntry by navController.currentBackStackEntryAsState()
    val title = when (backStackEntry?.destination?.route) {
        "screen1" -> "First Screen"
        "screen2" -> "Second Screen"
        "screen3" -> "Third Screen"
        else -> "Unknown Screen"
    }

    TopAppBar(
        title = { Text(title) },
        navigationIcon = {
            if (navController.previousBackStackEntry != null) {
                IconButton(onClick = { navController.navigateUp() }) {
                    Icon(Icons.Outlined.ArrowBack, "Go Back")
                }
            }
        },
    )
}
```

Fa scattare una recomposition al cambiare dell'entry

1.5. Composable per l'interfaccia utente

AppBar

```
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun AppBar(navController: NavHostController) {
    val backStackEntry by navController.currentBackStackEntryAsState()
    val title = when (backStackEntry?.destination?.route) {
        "screen1" -> "First Screen"
        "screen2" -> "Second Screen"
        "screen3" -> "Third Screen"
        else -> "Unknown Screen"
    }

    TopAppBar(
        title = { Text(title) },
        navigationIcon = {
            if (navController.previousBackStackEntry != null) {
                IconButton(onClick = { navController.navigateUp() }) {
                    Icon(Icons.Outlined.ArrowBack, "Go Back")
                }
            }
        },
    )
}
```

Naviga alla schermata precedente, uscendo dall'app se non ce n'è una

1.5. Composable per l’interfaccia utente

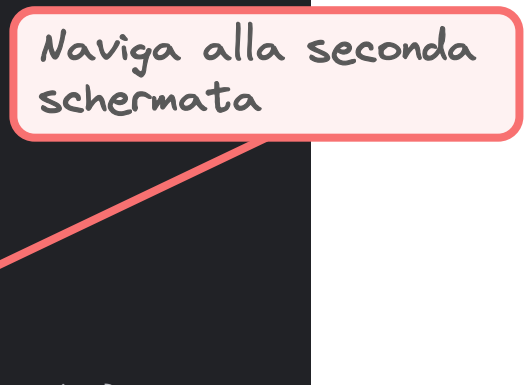
Screen1

```
@Composable
fun Screen1(navController: NavHostController) {
    Box(
        contentAlignment = Alignment.Center,
        modifier = Modifier
            .fillMaxSize()
            .background(MaterialTheme.colorScheme.primaryContainer)
    ) {
        Button(
            onClick = { navController.navigate(NavigationRoute.Screen2.route) },
            modifier = Modifier.width(LocalConfiguration.current.screenWidthDp.dp / 2)
        ) {
            Text("Screen2")
        }
    }
}
```

1.5. Composable per l'interfaccia utente

Screen1

```
@Composable
fun Screen1(navController: NavHostController) {
    Box(
        contentAlignment = Alignment.Center,
        modifier = Modifier
            .fillMaxSize()
            .background(MaterialTheme.colorScheme.primaryContainer)
    ) {
        Button(
            onClick = { navController.navigate(NavigationRoute.Screen2.route) },
            modifier = Modifier.width(LocalConfiguration.current.screenWidthDp.dp / 2)
        ) {
            Text("Screen2")
        }
    }
}
```



1.5. Composable per l'interfaccia utente

Screen2

```
@Composable
fun Screen2(navController: NavHostController) {
    Row(
        verticalAlignment = Alignment.CenterVertically,
        horizontalArrangement = Arrangement.spacedBy(16.dp),
        modifier = Modifier
            .fillMaxSize()
            .background(MaterialTheme.colorScheme.secondaryContainer)
            .padding(16.dp)
            .fillMaxWidth()
    ) {
        Button(
            onClick = { navController.navigateUp() },
            modifier = Modifier.weight(1F)
        ) { Text("Screen1") }
        Button(
            onClick = { navController.navigate(NavigationRoute.Screen3.route) },
            modifier = Modifier.weight(1F)
        ) { Text("Screen3") }
    }
}
```

1.5. Composable per l’interfaccia utente

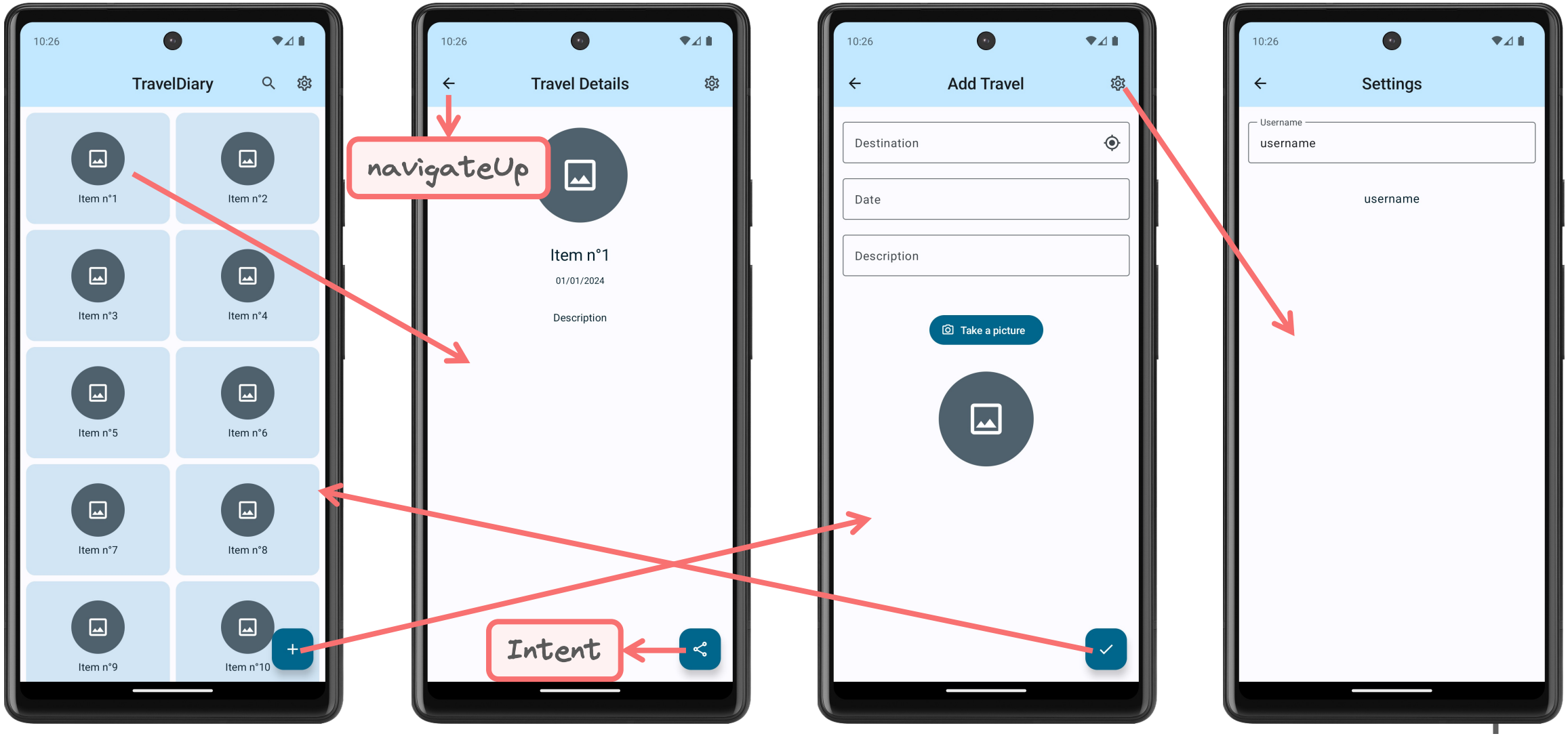
Screen3

```
@Composable
fun Screen3(navController: NavHostController) {
    Box(
        contentAlignment = Alignment.Center,
        modifier = Modifier
            .fillMaxSize()
            .background(MaterialTheme.colorScheme.tertiaryContainer)
    ) {
        Button(
            onClick = { navController.navigateUp() },
            modifier = Modifier.width(LocalConfiguration.current.screenWidthDp.dp / 2)
        ) {
            Text("Screen2")
        }
    }
}
```

2. TravelDiary - Navigazione

- Arricchire l'applicazione TravelDiary con le seguenti funzionalità:
  - Navigazione tra le varie schermate
  - Passaggio del **`travelId`** dalla schermata Home a TravelDetails tramite **rotta dinamica**
  - Bonus: condivisione del viaggio tramite Intent al click del FAB nella schermata TravelDetails
- Alcuni indizi su come procedere nelle prossime slide
- Link alla documentazione necessaria nell'ultima slide

2. TravelDiary - Navigazione



Rotte dinamiche

- È possibile passare parametri tra due destinazioni tramite una rotta dinamica
- Ad esempio, vogliamo passare l'ID del viaggio selezionato quando navighiamo dalla schermata home a quella di dettaglio:
  - Definiamo una rotta **`travels/{travelId}`**
    - Le parentesi graffe indicano la parte dinamica della rotta
  - Navigando verso **`travels/1234`**, visitiamo la rotta con **`travelId`** = **`"1234"`**
  - All'interno della rotta, è possibile reperire il parametro **`travelId`** tramite:

```
backStackEntry.arguments?.getString("travelId")
```

Hint: miglioramenti rispetto all'esercizio precedente

- Arricchiamo la struttura dati che memorizza le rotte della nostra app:
  - Gestendo anche il titolo della rotta
  - Gestendo rotte dinamiche
  - Memorizzando l'elenco delle rotte
- **Nota**: in caso di necessità, è possibile estendere ulteriormente questa implementazione, memorizzando ad esempio l'icona di ogni rotta da mostrare in una Navigation Bar

Hint: definizione delle destinazioni

```
sealed class TravelDiaryRoute(
    val route: String,
    val title: String,
    val arguments: List<NamedNavArgument> = emptyList()
) {
    // TODO: Home route
    data object TravelDetails : TravelDiaryRoute(
        "travels/{travelId}",
        "Travel Details",
        listOf(navArgument("travelId") { type = NavType.StringType })
    ) {
        fun buildRoute(travelId: String) = "travels/$travelId"
    }
    // TODO: AddTravel route
    // TODO: Settings route

    companion object {
       val routes = setOf(Home, TravelDetails, AddTravel, Settings)
    }
}
```

Hint: definizione delle destinazioni

```
sealed class TravelDiaryRoute(
    val route: String,
    val title: String,
    val arguments: List<NamedNavArgument> = emptyList()
) {
    // TODO: Home route
    data object TravelDetails : TravelDiaryRoute(
        "travels/{travelId}",
        "Travel Details",
        listOf(navArgument("travelId") { type = NavType.StringType })
    ) {
        fun buildRoute(travelId: String) = "travels/$travelId"
    }
    // TODO: AddTravel route
    // TODO: Settings route

    companion object {
       val routes = setOf(Home, TravelDetails, AddTravel, Settings)
    }
}
```

Lista di parametri della rotta TravelDetails

Metodo buildRoute per la creazione delle rotte dinamiche

Hint: definizione delle destinazioni

```
sealed class TravelDiaryRoute(
    val route: String,
    val title: String,
    val arguments: List<NamedNavArgument> = emptyList()
) {
    // TODO: Home route
    data object TravelDetails : TravelDiaryRoute(
        "travels/{travelId}",
        "Travel Details",
        listOf(navArgument("travelId") { type = NavType.StringType })
    ) {
        fun buildRoute(travelId: String) = "travels/$travelId"
    }
    // TODO: AddTravel route
    // TODO: Settings route

    companion object {
       val routes = setOf(Home, TravelDetails, AddTravel, Settings)
    }
}
```

Permette di ottenere l'elenco di tutte le rotte tramite TravelDiaryRoute.routes

Hint: grafo di navigazione con rotte dinamiche

- All’interno del NavGraph, è possibile ottenere un riferimento all’attuale entry nel backStack in modo da reperire l’ID del viaggio e passarlo alla schermata TravelDetails

```
with(TravelDiaryRoute.TravelDetails) {
    composable(route, arguments) { backStackEntry ->
        TravelDetailsScreen(backStackEntry.arguments?.getString("travelId") ?: "")
    }
}
```

Hint: navigazione verso una rotta dinamica

- Grazie al metodo **`buildRoute`**, possiamo navigare verso la rotta dinamica in maniera safe

```
navController.navigate(TravelDiaryRoute.TravelDetails.buildRoute(travelId = "1234"))
```

Hint: tenere traccia della rotta attuale

- Con qualche modifica al codice dell'esercizio precedente, è possibile ottenere un riferimento all'oggetto della schermata attuale, in modo da poterne reperire sia la route che il titolo

```
val navController = rememberNavController()
val backStackEntry by navController.currentBackStackEntryAsState()
val currentRoute by remember {
    derivedStateOf {
        TravelDiaryRoute.routes.find {
            it.route == backStackEntry?.destination?.route
        } ?: TravelDiaryRoute.Home
    }
}
```

Hint: tenere traccia della rotta attuale

- Con qualche modifica al codice dell'esercizio precedente, è possibile ottenere un riferimento all'oggetto della schermata attuale, in modo da poterne reperire sia la route che il titolo

```
val navController = rememberNavController()
val backStackEntry by navController.currentBackStackEntryAsState()
val currentRoute by remember {
    derivedStateOf {
        TravelDiaryRoute.routes.find {
            it.route == backStackEntry?.destination?.route
        } ?: TravelDiaryRoute.Home
    }
}
```

Il codice interno a derivedStateOf viene rieseguito solo se le sue dipendenze cambiano. In questo caso, al cambiare di backStackEntry.

Tocca a voi!

Riferimenti

- Navigation with Compose
  https://developer.android.com/jetpack/compose/navigation
- Sent text content via Intent
  https://developer.android.com/training/sharing/send#send-text-content