

Navigation

App Compose con schermate multiple

Laboratorio 2

Recap: Navigazione in Compose

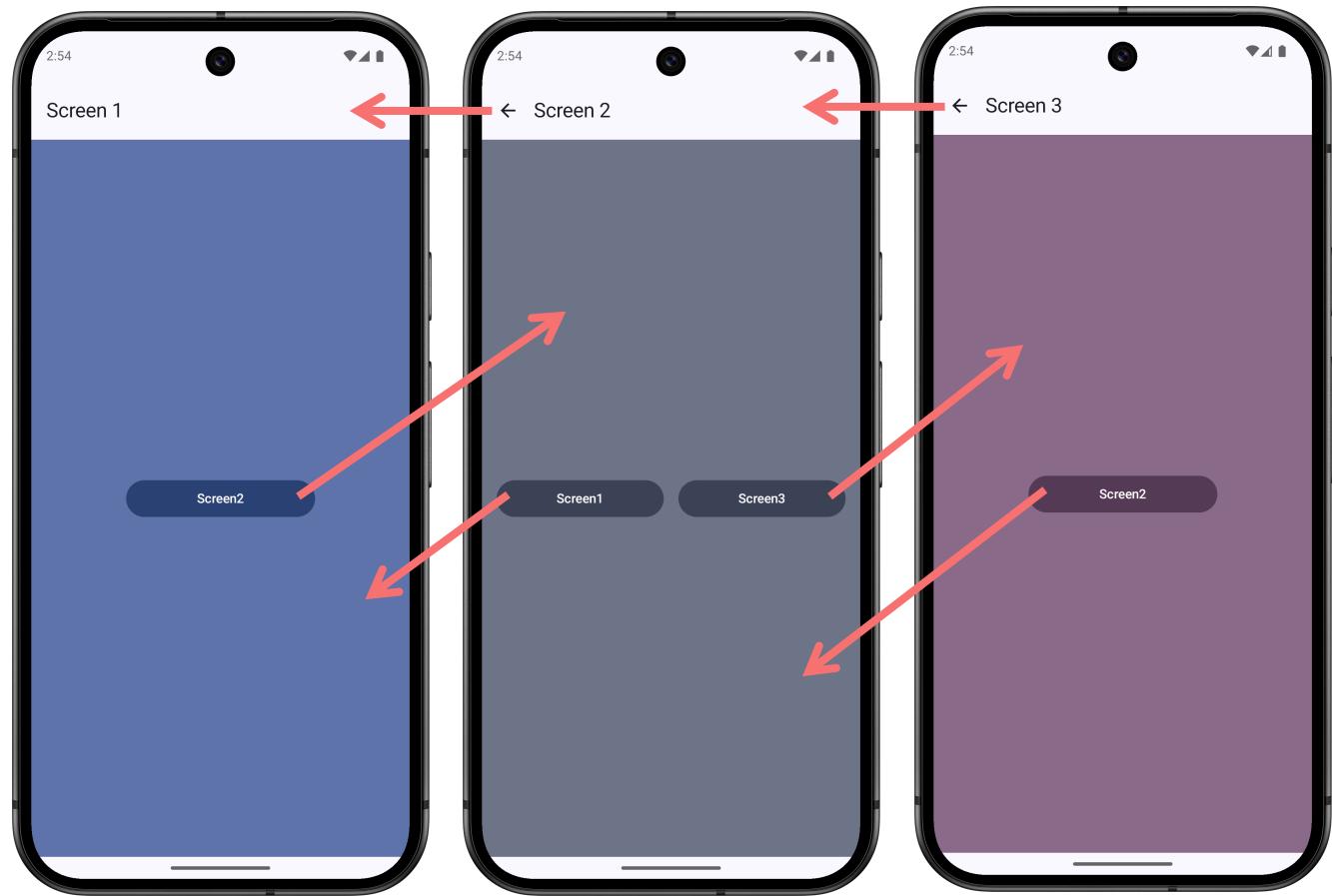
- Un progetto Compose che implementa le funzionalità di navigazione è dotato di tre componenti:
 - **Grafo di navigazione**: definisce le destinazioni raggiungibili tramite navigazione all'interno dell'app, indicando per ciascuna il composable che ne contiene la UI
 - **NavHost**: è il composable utilizzato 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
 - Con migliore gestione delle stringhe per l'internazionalizzazione
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

1. Installazione dipendenze
2. Definizione delle destinazioni
3. Creazione del grafo di navigazione
4. Creazione del **NavController**
5. Composable per l'interfaccia utente

Nota: navigazione type safe

- Ci sono due modi per implementare la navigazione in un'app Jetpack Compose
- Nel primo, un po' come in un sito web, **ogni schermata è identificata da una stringa**

```
composable("home-screen") { HomeScreen() }
```

```
navController.navigate("home-screen")
```

– Benché semplice da implementare, questo metodo presenta un problema: in fase di navigazione, non siamo in grado di stabilire se la destinazione che forniamo esista o meno

```
navController.navigate("non-existent-screen")
```

Errore a runtime!

– Questo è particolarmente problematico ed error-prone in caso di rotte con parametri, in cui è necessario creare la rota concatenando delle stringhe:

```
navController.navigate("profile/$profileId")
```

Nota: navigazione type safe

- Per ovviare a questo problema, dalla versione 2.8.0 della libreria Navigation, è possibile implementare la navigazione in un secondo modo, ossia **identificando ogni schermata con un oggetto**

```
composable<Home> { HomeScreen() }
```

```
navController.navigate(Profile(id = profileId))
```

```
navController.navigate(Home)
```

```
navController.navigate(NonExistentScreen)
```

- Questa è l'implementazione che adotteremo per le esercitazioni in laboratorio

Errore a compile time

1.1. Installazione dipendenze

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

```
implementation("androidx.navigation:navigation-compose:2.8.9")
implementation("io.ktor:ktor-serialization-kotlinx-json:2.3.8")
```

Replace with version catalog entries

- Nel blocco **plugins**

```
alias(libs.plugins.serialization)
```

- Nel blocco **[plugins]** del file **libs.versions.toml**, aggiungere:

```
serialization = { id = "org.jetbrains.kotlin.plugin.serialization", version.ref = "kotlin" }
```

1.2. Definizione delle destinazioni

- Ogni destinazione è identificata da un oggetto marcato come **@Serializable**, che può essere passato 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 interface o sealed class**:

```
sealed interface NavigationRoute {  
    @Serializable data object Screen1 : NavigationRoute  
    @Serializable data object Screen2 : NavigationRoute  
    @Serializable data object Screen3 : NavigationRoute  
}
```

1.3. Creazione del grafo di navigazione

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

1.3. Creazione del grafo di navigazione

```
@Composable
fun NavGraph(
    navController: NavHostController,
    modifier: Modifier = Modifier ←
) {
    NavHost(
        navController = navController,
        startDestination = NavigationRoute.Screen1,
        modifier = modifier
    ) {
        composable<NavigationRoute.Screen1> {
            Screen1(navController)
        }
        composable<NavigationRoute.Screen2> {
            Screen2(navController)
        }
        composable<NavigationRoute.Screen3> {
            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 già accennato in precedenza, 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 {  
    NavigationTheme {  
        val navController = rememberNavController()  
        Scaffold(  
            topBar = { AppBar(navController) },  
            modifier = Modifier.fillMaxSize()  
        ) { innerPadding ->  
            NavGraph(navController, Modifier.padding(innerPadding))  
        }  
    }  
}
```

1.5. Composable per l'interfaccia utente

AppBar

```
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun AppBar(navController: NavHostController) {
    TopAppBar(
        title = { Text("???" ) },
        navigationIcon = {
            IconButton(onClick = { navController.navigateUp() }) {
                Icon(Icons.AutoMirrored.Outlined.ArrowBack, "Go Back")
            }
        },
    )
}
```

Nota: funzionalità dinamiche in base alla destinazione attuale

- Se abbiamo dei componenti condivisi tra più destinazioni, potremmo volerne modificare il comportamento in base alla destinazione attuale
 - Ad esempio, come cambiamo il titolo della nostra AppBar?
- Soluzione:
 - Possiamo recuperare la rotta attuale con:

```
val backStackEntry by navController.currentBackStackEntryAsState()
```
 - Quella precedente con:

```
navController.previousBackStackEntry
```
 - E verificare se ci troviamo in una certa rotta con:

```
backStackEntry?.destination?.hasRoute<NavigationRoute.Screen1>() == true
```

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?.hasRoute<NavigationRoute.Screen1>() == true -> "Screen 1"
        backStackEntry?.destination?.hasRoute<NavigationRoute.Screen2>() == true -> "Screen 2"
        backStackEntry?.destination?.hasRoute<NavigationRoute.Screen3>() == true -> "Screen 3"
        else -> "Unknown Screen"
    }
    TopAppBar(
        title = { Text(title) },
        navigationIcon = {
            if (navController.previousBackStackEntry != null) {
                IconButton(onClick = { navController.navigateUp() }) {
                    Icon(Icons.AutoMirrored.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?.hasRoute<NavigationRoute.Screen1>() == true -> "Screen 1"
        backStackEntry?.destination?.hasRoute<NavigationRoute.Screen2>() == true -> "Screen 2"
        backStackEntry?.destination?.hasRoute<NavigationRoute.Screen3>() == true -> "Screen 3"
        else -> "Unknown Screen"
    }
    TopAppBar(
        title = { Text(title) },
        navigationIcon = {
            if (navController.previousBackStackEntry != null) {
                IconButton(onClick = { navController.navigateUp() }) {
                    Icon(Icons.AutoMirrored.Outlined.ArrowBack, "Go Back")
                }
            }
        },
    )
}
```

Fa scattare una recomposition al cambiare dell'entry

Definisce il titolo in base alla rotta attuale

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?.hasRoute<NavigationRoute.Screen1>() == true -> "Screen 1"
        backStackEntry?.destination?.hasRoute<NavigationRoute.Screen2>() == true -> "Screen 2"
        backStackEntry?.destination?.hasRoute<NavigationRoute.Screen3>() == true -> "Screen 3"
        else -> "Unknown Screen"
    }
}

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

Mostra il tasto indietro
solo se il backstack
contiene almeno 2 entry

1.5. Composable per l'interfaccia utente

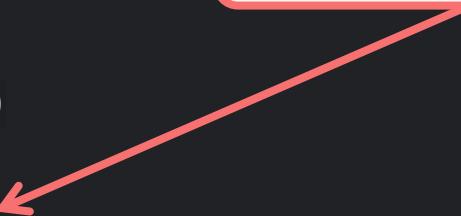
Screen1

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



Naviga alla seconda schermata

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() },
            colors = ButtonDefaults.buttonColors(MaterialTheme.colorScheme.secondary, MaterialTheme.colorScheme.onSecondary),
            modifier = Modifier.weight(1F)
        ) { Text("Screen1") }
        Button(
            onClick = { navController.navigate(NavigationRoute.Screen3) },
            colors = ButtonDefaults.buttonColors(MaterialTheme.colorScheme.secondary, MaterialTheme.colorScheme.onSecondary),
            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() },
            colors = ButtonDefaults.buttonColors(
                containerColor = MaterialTheme.colorScheme.tertiary,
                contentColor = MaterialTheme.colorScheme.onTertiary,
            ),
            modifier = Modifier.width(LocalConfiguration.current.screenWidthDp.dp / 2)
        ) {
            Text("Screen2")
        }
    }
}
```

Bonus: gestione delle stringhe e internazionalizzazione

- Possiamo gestire meglio le stringhe all'interno della nostra app
- E aggiungere il supporto multilingua!

Bonus: gestione delle stringhe e internazionalizzazione

1. Nel file **res/values/strings.xml**, creiamo una entry per tutte le stringhe dell'interfaccia utente

```
<resources>
    <string name="app_name">Navigation</string>
    <string name="screen1_name">Screen1</string>
    <string name="screen2_name">Screen2</string>
    <string name="screen3_name">Screen3</string>
    <string name="unknown_screen_name">Unknown Screen</string>
    <string name="back_button_description">Go Back</string>
</resources>
```

Bonus: gestione delle stringhe e internazionalizzazione

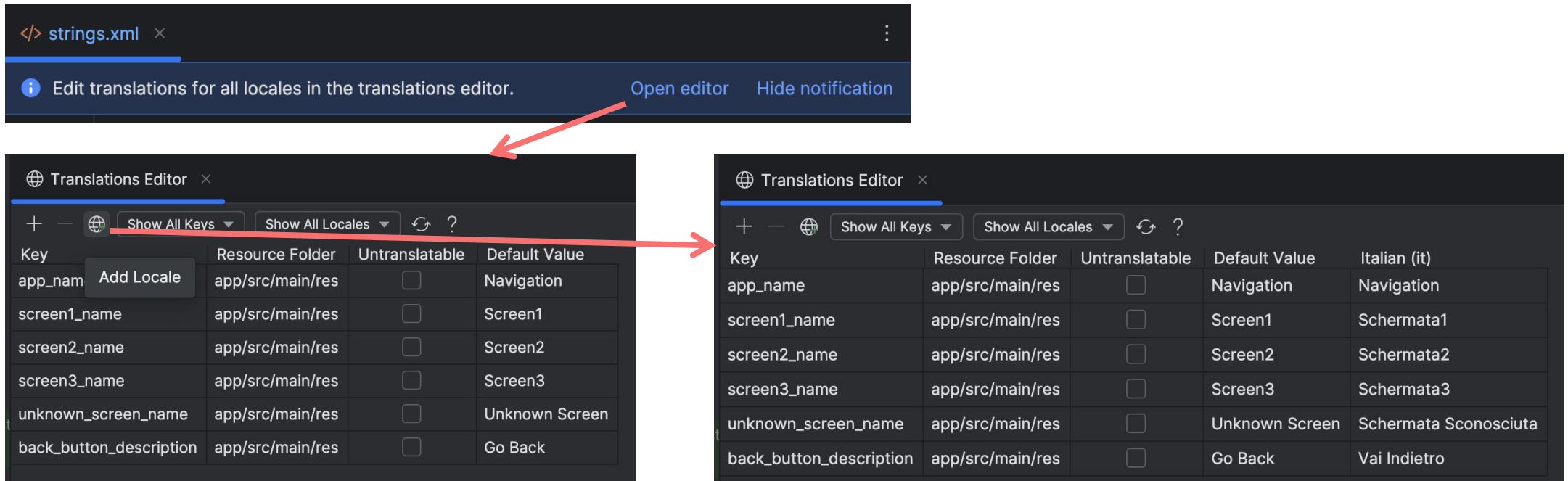
2. Utilizziamo i valori definiti nel file all'interno dell'app

Esempio:

```
Text("Screen 2") → Text(stringResource(R.string.screen2_name))
```

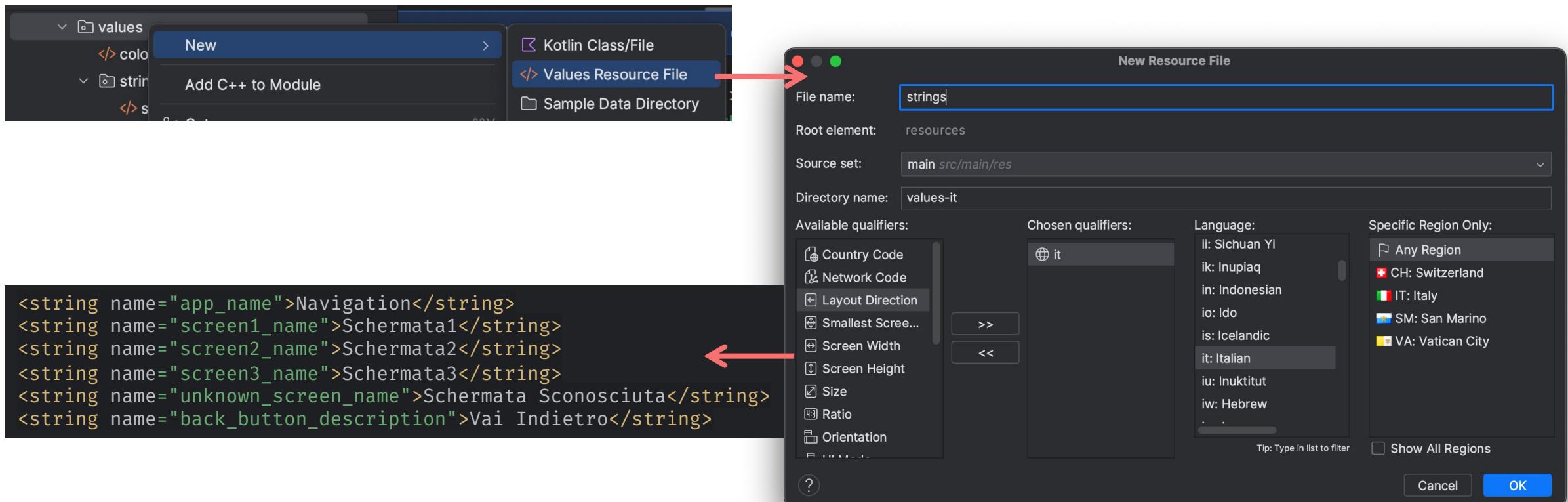
Bonus: gestione delle stringhe e internazionalizzazione

3. Apriamo l'editor per le traduzioni, aggiungiamo la lingua italiana e scriviamo le relative stringhe



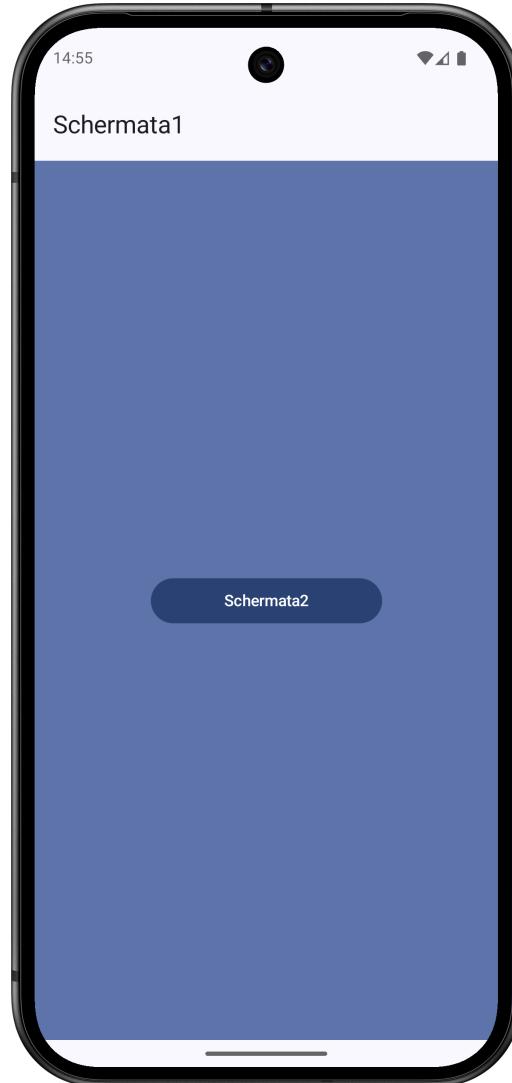
Bonus: gestione delle stringhe e internazionalizzazione

- È anche possibile creare il file manualmente



Bonus: gestione delle stringhe e internazionalizzazione

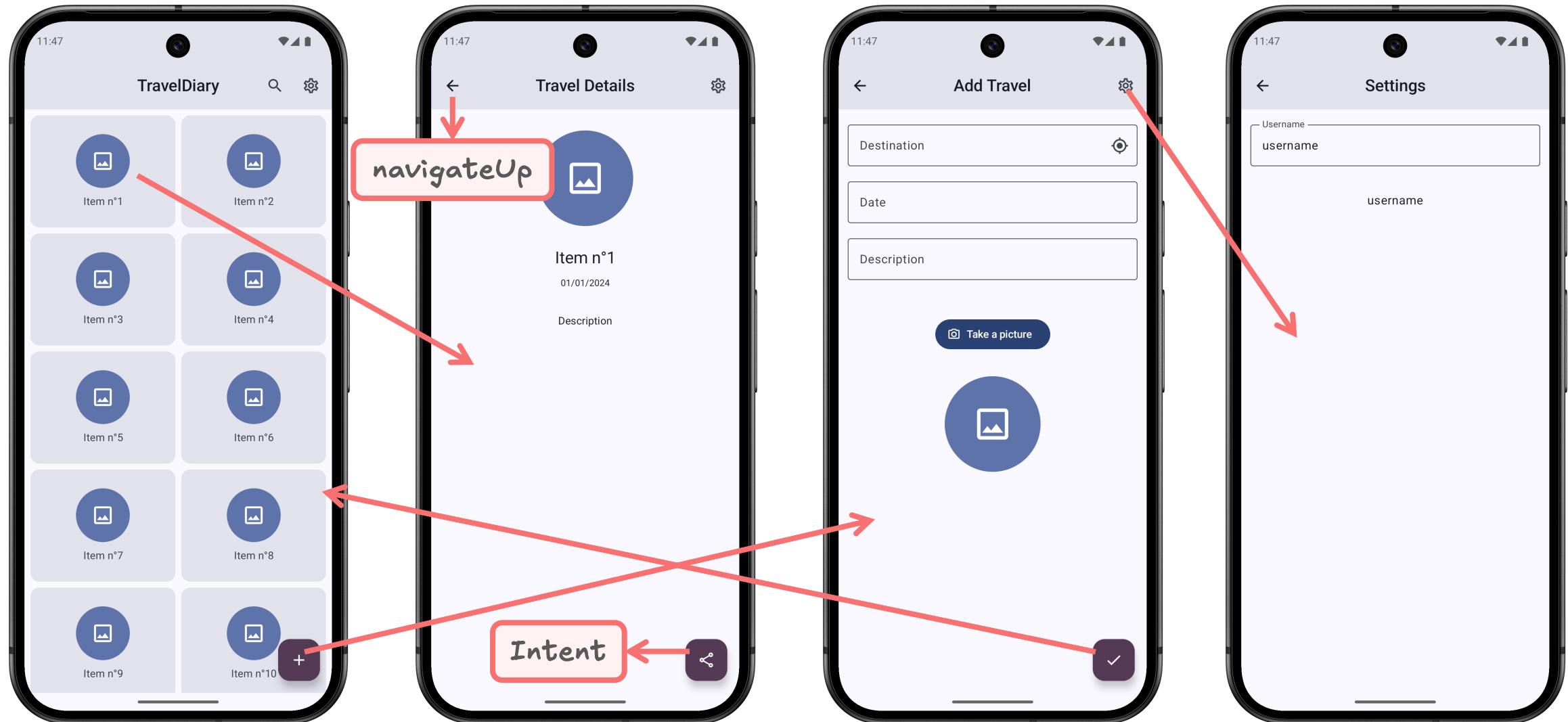
- Cambiando la lingua del dispositivo in italiano, cambierà anche il testo all'interno della nostra app
 - Settings → System → Languages → System Languages → Add a Language
 - Selezionare “Italiano (Italia)” e trascinarlo sopra all'inglese



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
- Potete partire dal vostro codice, oppure dalla soluzione dello scorso laboratorio
- 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 **TravelDetails**. La rotta assume un valore diverso in base all'ID del viaggio, pertanto non è un **object**, ma una **class**:

```
@Serializable  
data class TravelDetails(val travelId: String) : TravelDiaryRoute
```

prende nome da
sealed class/interface

Rotte dinamiche

- Navigando verso **TravelDetails("1234")**, visitiamo la rotta con **travelId="1234"**
- All'interno della rotta, è possibile reperire il parametro **travelId** tramite la **backstackEntry**

```
composable<TravelDiaryRoute.TravelDetails> { backStackEntry ->
    val route = backStackEntry.toRoute<TravelDiaryRoute.TravelDetails>()
    TravelDetailsScreen(navController, route.travelId)
}
```

Tocca a voi!

Riferimenti

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