Lab on apps development for tablets, smartphones and smartwatches

Week 3: Screens and menus

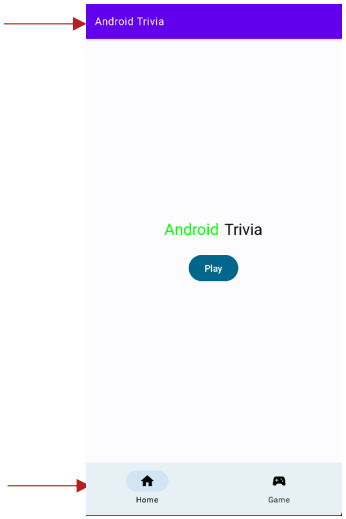
Giovanni Ansaloni

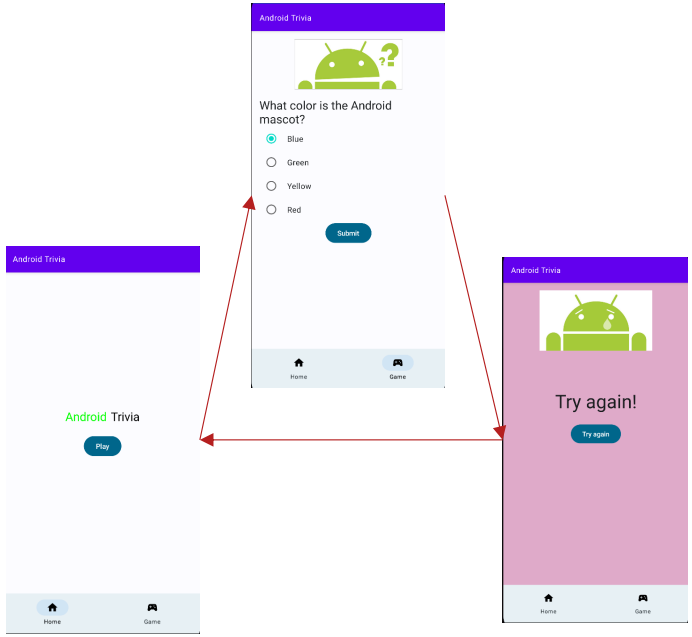
Dimitra Tatli, Riselda Kodra, Yuxuan Wang
Qunyou Liu, Amirhossein Shahbazinia, Christodoulos Kechris

School of Engineering (STI) – Institute of Electrical and Micro Engineering (IEM)

Class outline

- **Screens and navigation**
  - passing arguments between screens
- Menus
  - bottom menu
  - options menu
  - navigation drawer





Screens

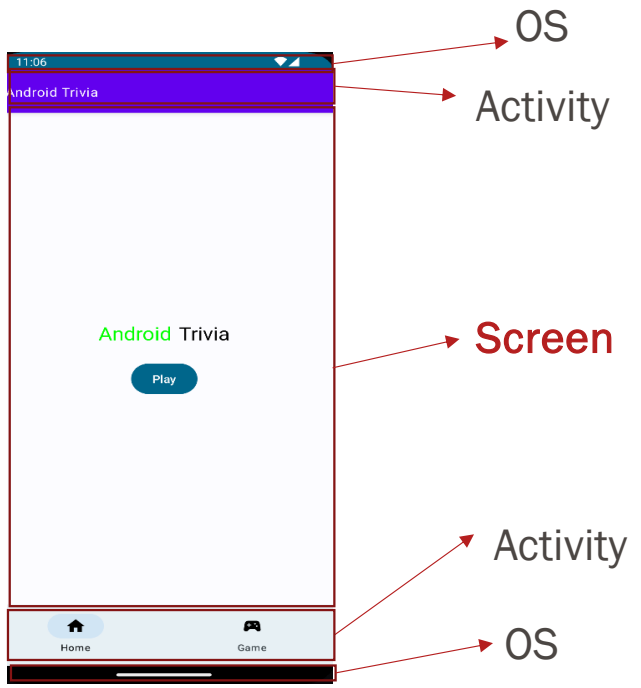
- Screens → composables functions
  that represent a modular portion of the user interface within an activity
  - Usually all screen area except:
    - Bottom navigation bar
    - Application bar

    managed by Activity

    - Top & bottom bars

    managed by OS

- Navigation component
  orchestrates user interaction across screens



Navigation basics

- NavController : the central API for Navigation.
  Keeps track of
  - the state of the composables
  - the order in which they have been accessed via recomposition
    → back stack of screen composables

- Created in the Activity “SetContent()” function:

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

REMEMBER: *add dependency in Gradle file:*

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

Navigation basics

- The NavHost links the NavController with a navigation graph
  - each NavController must be associated with a single NavHost
- Navigation graph
  - specifies the Screen Destinations, Routes and Actions
    - in onCreate() → setContent {}
  - user interactions can cause navigation events across destinations
- Destination composable functions are identified by a unique route
  - startDestination indicates the initial screen

```
NavHost(navController = navController, startDestination = "home") {
        composable("home") { HomeScreen(/*...*/) }
        composable("game") { GameScreen(/*...*/) }
        /*...*/
}
```

Routes Destinations Actions

Navigation Actions

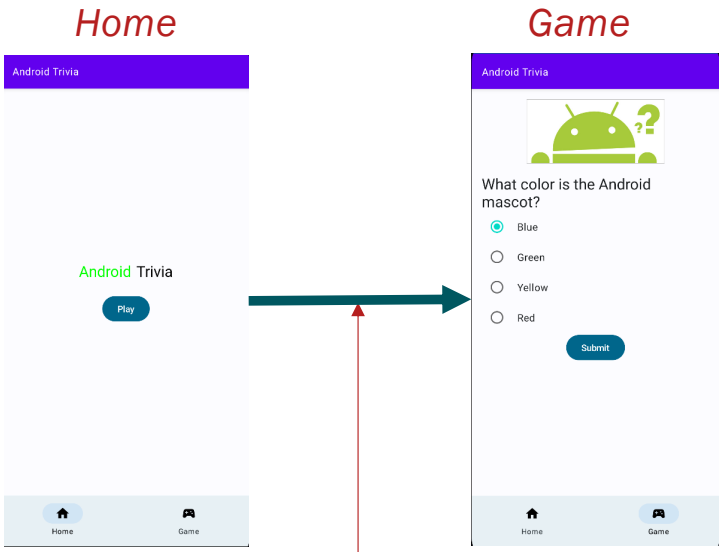
- Navigation between destinations with Actions

navController.navigate(<ROUTE>)

```
NavHost(navController = navController, startDestination = "home") {
   composable("home") {
      HomeScreen(onButtonClicked = { navController.navigate("game") })
   }
   composable("game") { GameScreen(/*...*/) }
   /*...*/
}
```

Route

Action



Navigation Actions

- Navigation between destinations with Actions

  navController.navigate(<ROUTE>)

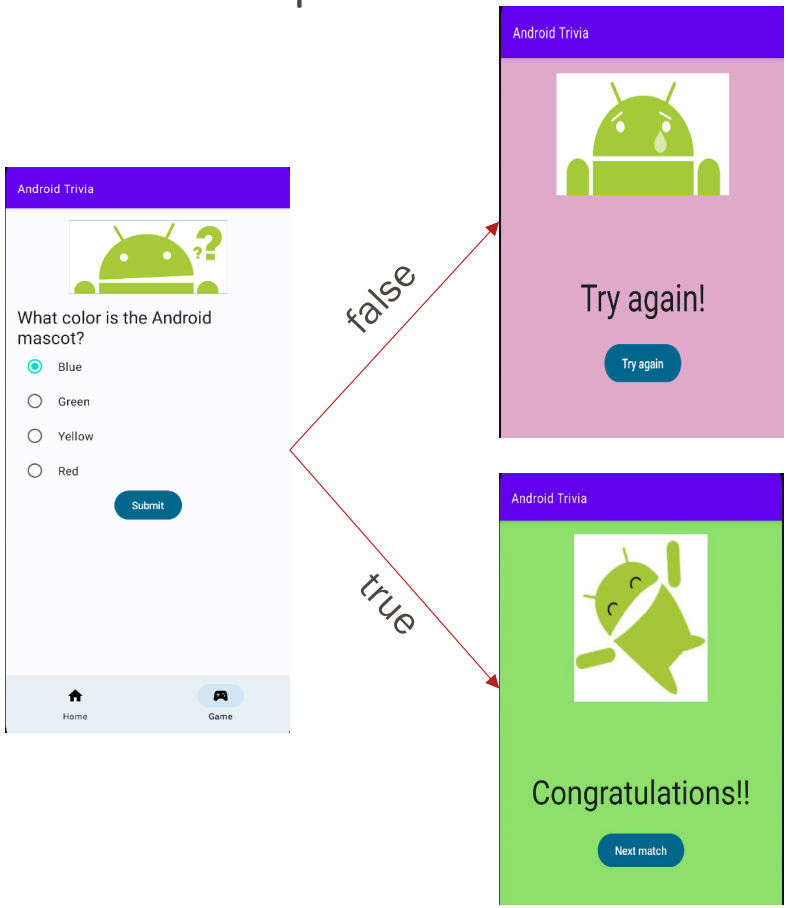
- Screens use Actions as lambda parameters

```
fun HomeScreen(
        onButtonClicked: () -> Unit,
        modifier: Modifier = Modifier
) {
        ...
        Button(
            onClick = onButtonClicked (),
            modifier = Modifier
        ) {
          Text(text = stringResource(R.string.button_text))
        }
}
```

Conditional Navigation

- Multiple Actions originating from the same composable can be specified
  - The proper one can than be selected programmatically

```
NavHost(navController = navController, startDestination = "home") {
   composable("game") {
      GameScreen(onSubmitClicked = {
         if (withCondition == true) {
            navController.navigate("gameWon")
         } else {
            navController.navigate("gameOver")
         }
      })
   }
   composable("gameWon") { GameWonScreen(/*...*/) }
   composable("gameOver") { GameOverScreen(/*...*/) }
   /*...*/
}
```



Pass data between destinations

- Destinations can specify arguments within Routes
  - Only primitive type: boolean, int, float, string
  - String type is the default

- Sending string:

```
var name = "John Doe"
   ...
navController.navigate("gameWon/$name")
```

- Retrieving string:

```
composable("gameWon/{name}") { backStackEntry ->
        GameWonScreen(backStackEntry.arguments?.getString("name"))
}
```

We might receive null value, so it needs to be handled properly

Pass data between destinations

- Destinations can specify arguments within Routes
  - Only primitive type: boolean, int, float, string

- Sending integer:

```
var score = 4
      ...
navController.navigate("gameWon/$score")
```

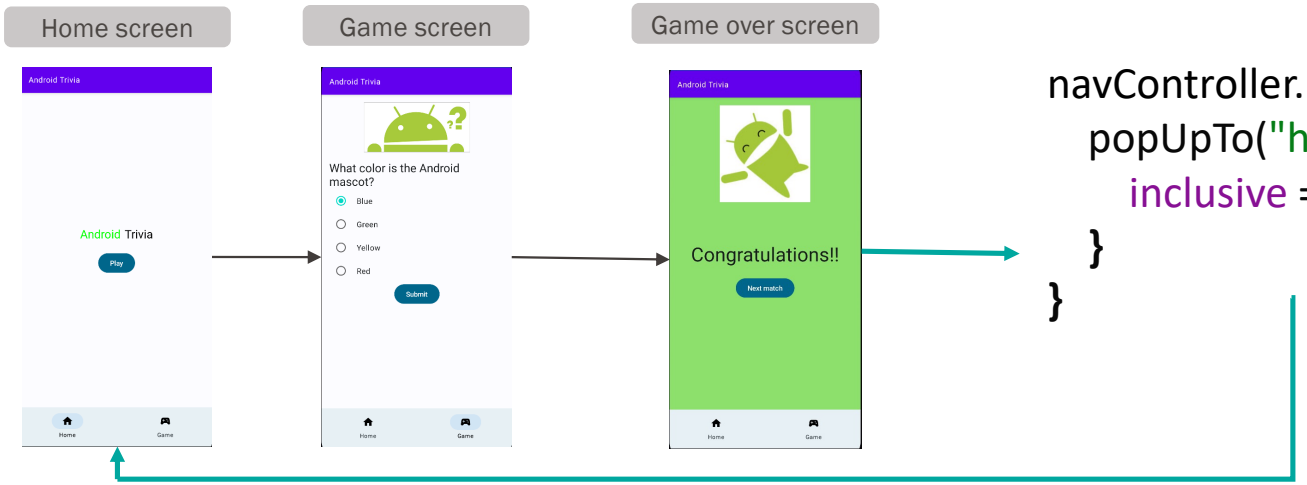
- Retrieving integer:

```
composable(
   "gameWon/{score}",
   arguments = listOf(navArgument("score") { type = NavType.IntType })
) { backStackEntry ->
   GameWonScreen(backStackEntry.arguments?.getInt("score"))
}
```

Back stack

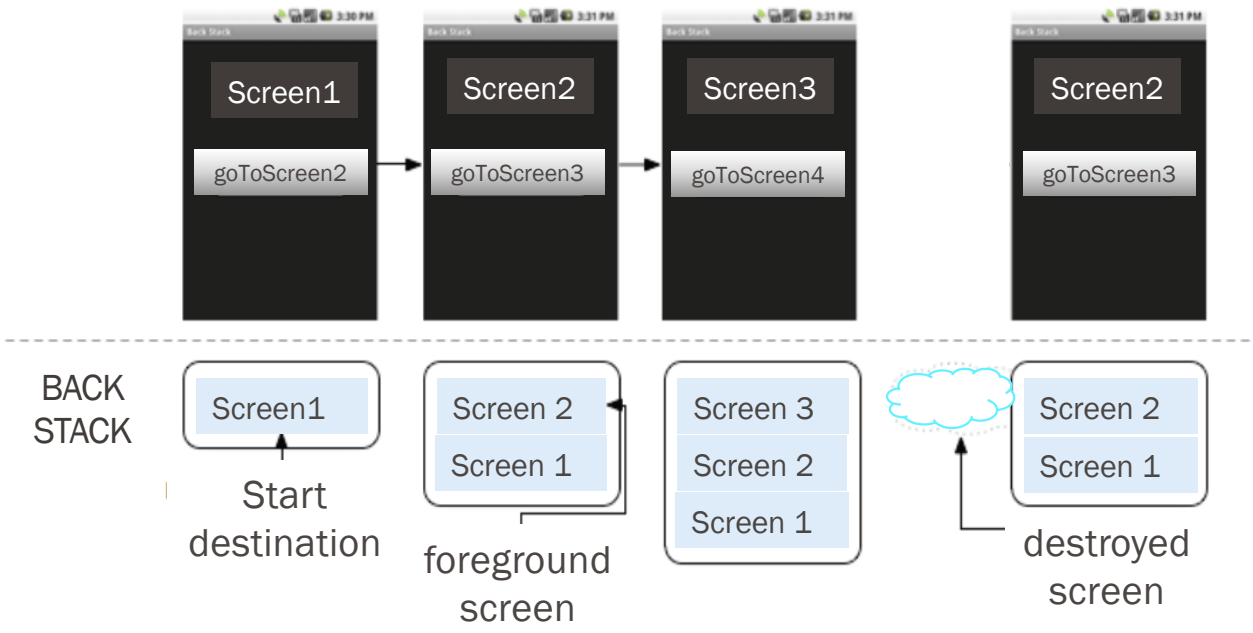
- Screens can be closed when navigating to free resources
  - going back to begin of game, or “Login” screen

- PopUp will clear the Activity stack up to the route destination
  - including the destination itself if inclusive is set to TRUE



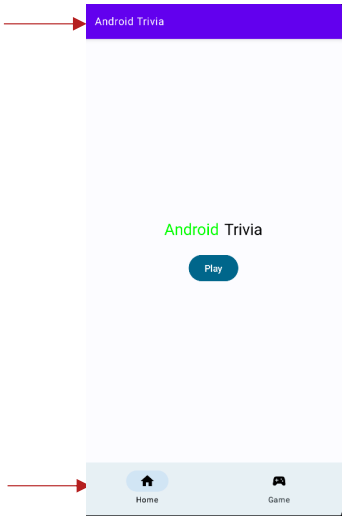
Back stack

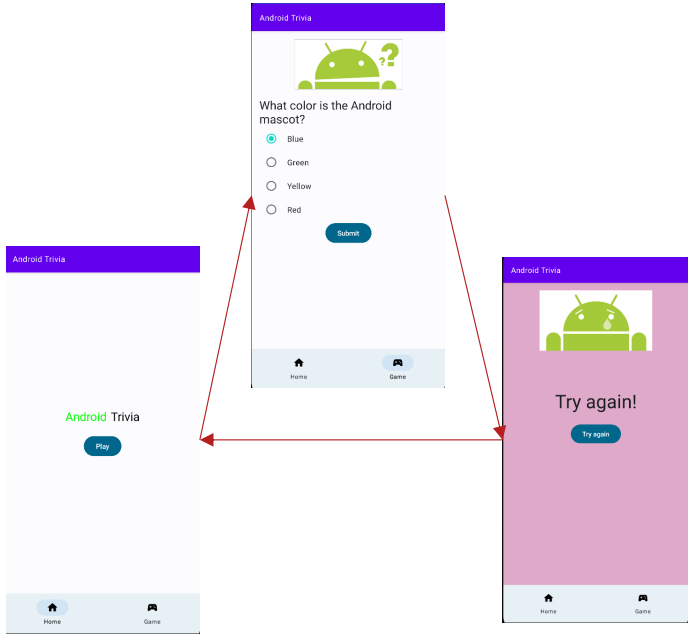
- When a new Screen is started, its reference is pushed on a stack managed by the OS → one stack per app
- When the current Screen ends executing, the Screen is popped from the stack as the previous resumes
- Stack top is always the foreground Screen
- When “popUpTo” is set, all Screens are popped up to the specified one



Class outline

- Screens and navigation
  - passing arguments between screens
- **Menus**
  - bottom menu
  - options menu
  - navigation drawer

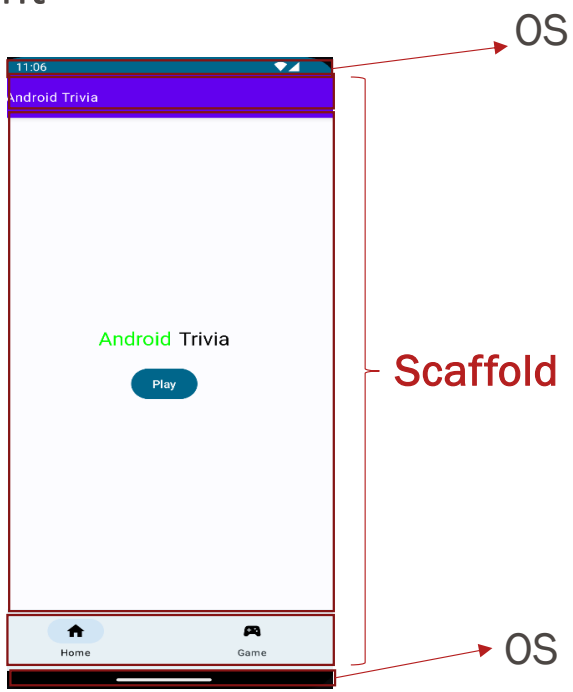




Scaffold

- Scaffold holds together:
  - topBar: The app bar on the top of the screen
  - bottomBar: The app bar on the bottom of the screen
  - floatingActionButton: A button that hovers over the bottom-right corner of the screen
  - main GUI area (e.g. Screens)
- Scaffold encapsulates top-most layout element
  - Usually, NavHost

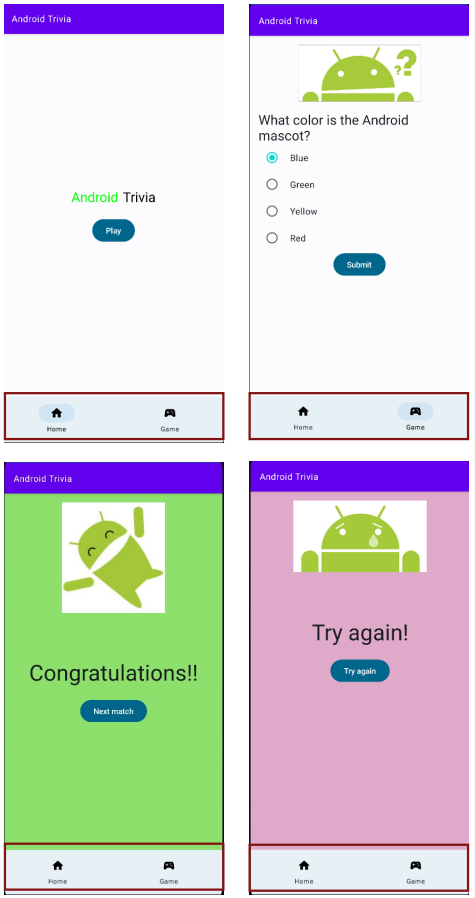
```
Scaffold(
    topBar = { /* ... */ },
    bottomBar = { /* ... */ },
    floatingActionButton = { /* ... */ },
) { innerPadding ->
    NavHost(
            /* ... */
    ) {
        /* ... */
    }
}
```



Bottom menu

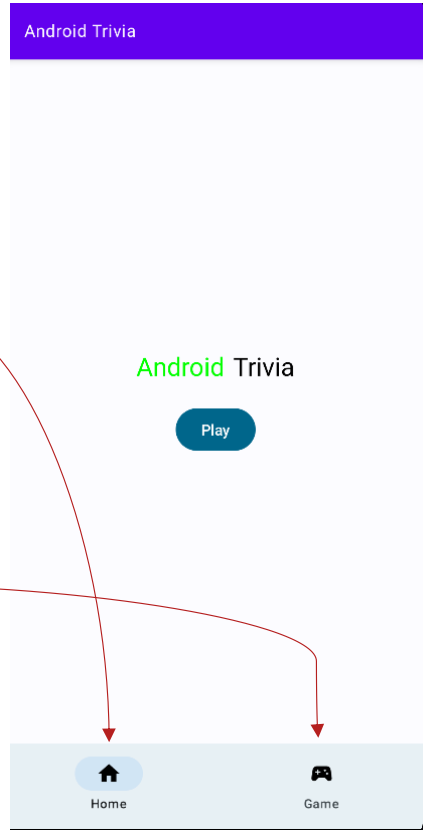
- Provides app-customizable navigation buttons at the bottom of the screen
  - User can tap on the button to navigate between screens

- Implementation:
  1. Add Scaffold
  2. Add BottomBar as a Scaffold parameter and a NavigationBar inside it
  3. Inside the NavigationBar add a NavigationBarItem for each item of the bottom menu
     - Icon, state and onClick() methods



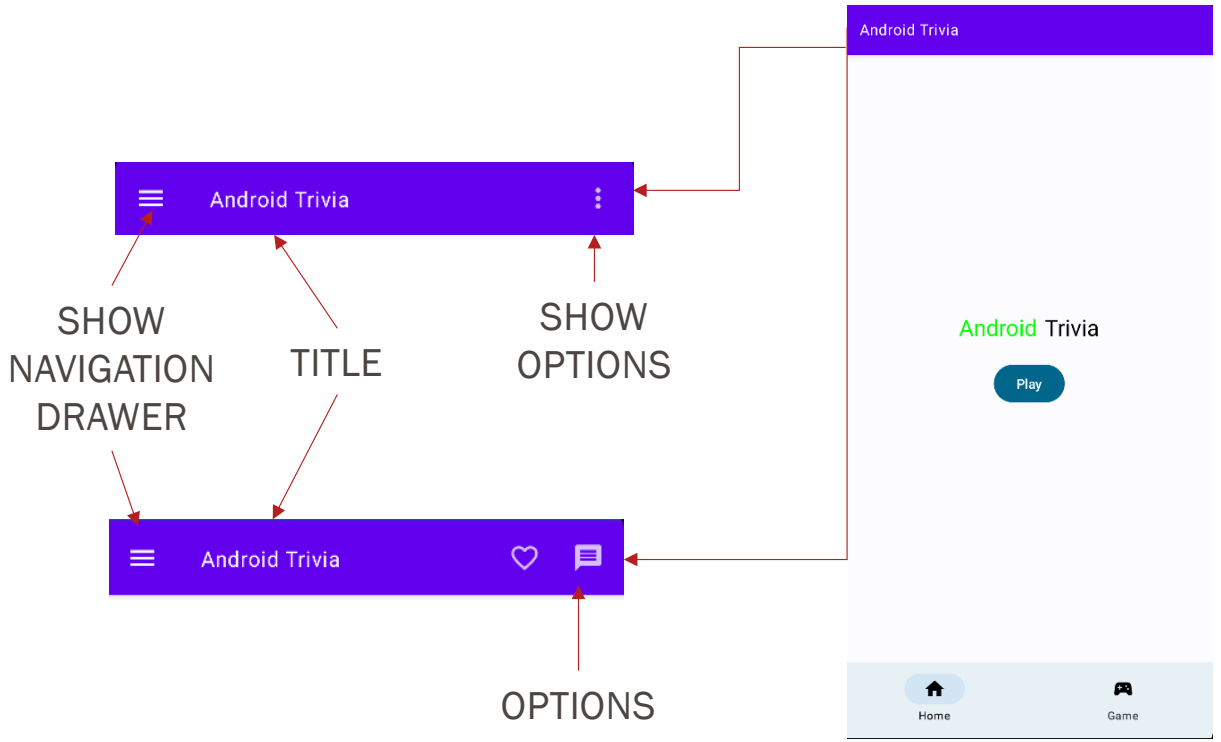
Bottom menu

```
1. → Scaffold(
2.    → bottomBar = {
         → NavigationBar {
3.          → NavigationBarItem(
                icon = { Icon(imageVector = Icons.Filled.Home, null) },
                selected = true ,
                onClick = { /* Navigate */ }
              )
              NavigationBarItem(
                icon = { Icon(imageVector = Icons.Filled.SportsEsports, null) },
                selected = false ,
                onClick = { /* Navigate */ }
              )
            }
         }
    ) { /* ... REST OF GUI with NavHost and Destinations ... */ }
```



The App bar

- Title
  - a descriptive string
- Show navigation drawer button
  - Navigation Drawer
    → panel that slides
    from the screen edge
- Options Menu / Options



Title

- Title
  - a descriptive string


```
Scaffold(
    topBar = {
        TopAppBar(
            title = {
                Text("Android Trivia", color = Color.White)
            },
        )
)
```

Options menu

1. Add actions parameter in the TopAppBar

2. Add a IconButton and DropdownMenu inside the actions

3. Define a boolean state variable to keep track of whether the options menu is shown

```
TopAppBar(
   title = {/* ... */},

   actions = {
      IconButton(onClick = { menuExpanded = !menuExpanded} ) {
            Icon(imageVector = Icons.Filled.MoreVert, null)
      }
      DropdownMenu(
           expanded = menuExpanded,
           onDismissRequest = { menuExpanded = false}
       ) {   /* NEXT SLIDE */}
} )
```

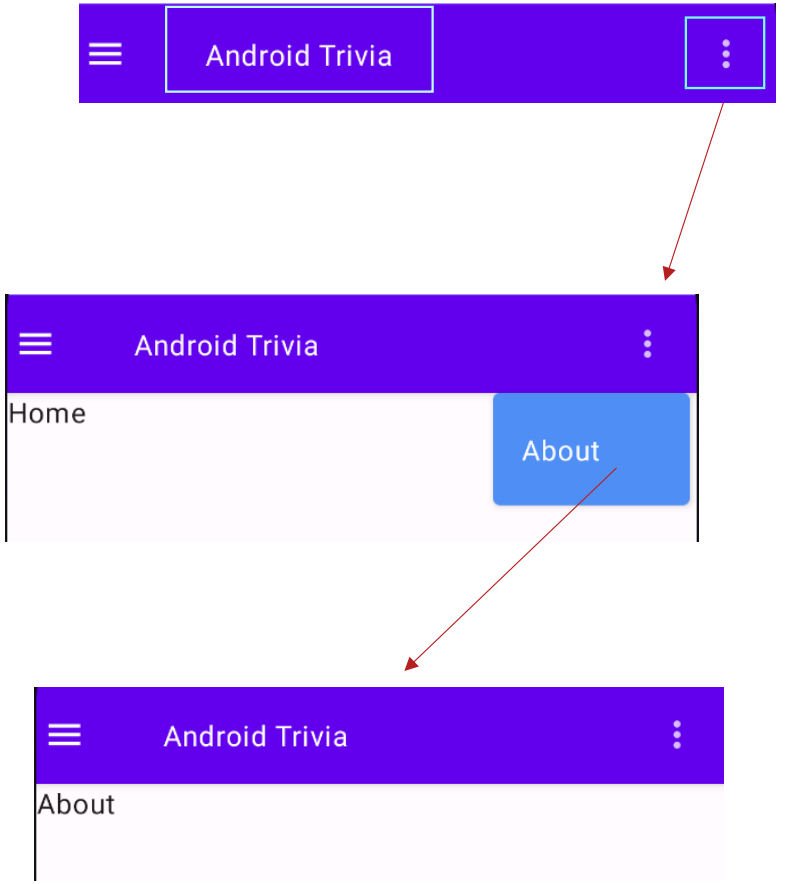
```
var menuExpanded by remember { mutableStateOf(false) }
```

Options dropdown menu

4. Add DropdownMenuItems that e.g. navigate somewhere

```
actions = {
    ...
     DropdownMenu(...){
       DropdownMenuItem(onClick = {
          menuExpanded = false
          navController.navigate("about")
       }) {
              Text(text = "About")
       }

       DropdownMenuItem(...){...}
     }
}
```



Options menu

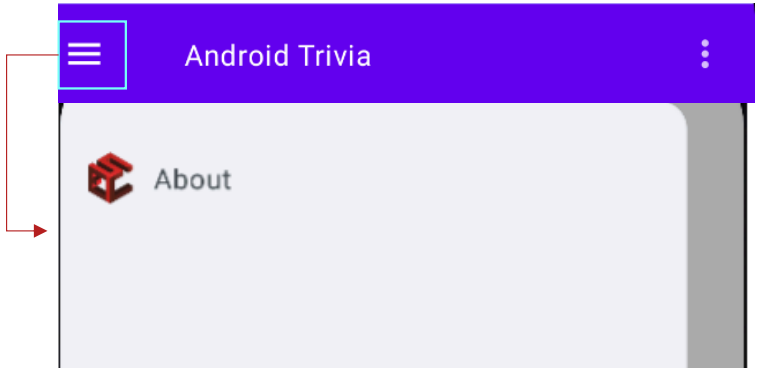
- Options without a dropdown menu.


```
actions = {
    IconButton(onClick = { navController.navigate("notifications") }) {
        Icon(
            imageVector = Icons.Filled.FavoriteBorder,
            contentDescription = null
        )
    }
    IconButton(onClick = { navController.navigate("messages") }) {
        Icon(
            imageVector = Icons.Filled.Message,
            contentDescription = null
        )
    }
}
```

Navigation drawer

1. Wrap the Scaffold with a ModalNavigationDrawer

2. Initialize drawer state with DrawerValue.Closed
   and add it as parameter to the ModalNavigationDrawer



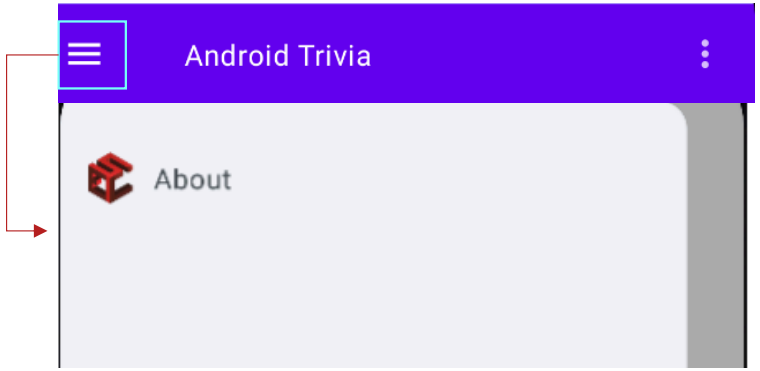

```
val drawerState = rememberDrawerState(
                        initialValue = DrawerValue.Closed)

ModalNavigationDrawer(
   drawerState = drawerState,
   drawerContent = {/*NEXT SLIDE*/ }
) {
  Scaffold(
     topBar = { … }
  )
}
```

Navigation drawer

3. Add a ModalDrawerSheet in the inside the drawerContent

```
ModalNavigationDrawer(
  drawerState = drawerState,
  drawerContent = {
    ModalDrawerSheet {
        ...
    }
})
```



4. Add NavigationDrawerItem inside ModalDrawerSheet for each item

```
ModalDrawerSheet {
  NavigationDrawerItem(
    label = { …},
    Icon = { …},
    selected = false,
    onClick = { /*Close menu, Navigate to Destination*/ })
  NavigationDrawerItem(…)
  …
}
```

Navigation drawer

5. Implement an IconButton inside the topAppBar as a navigationIcon

```
ModalNavigationDrawer( ... ) {
   Scaffold(
      topBar = {
         TopAppBar(
            title = { … },
            navigationIcon = {
               IconButton(onClick = { /* NEXT SLIDE */ }) {
                  Icon(imageVector = Icons.Filled.Menu, null)
               }
            })
         })
}
```

Navigation drawer

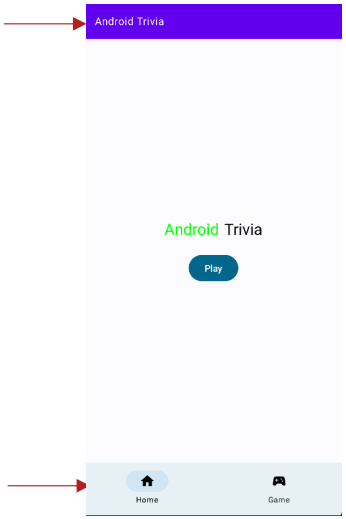
6. Implement the onClick callback to open the drawer
   - drawerState.open()
   - done with coroutines in the example below (explained in later classes).

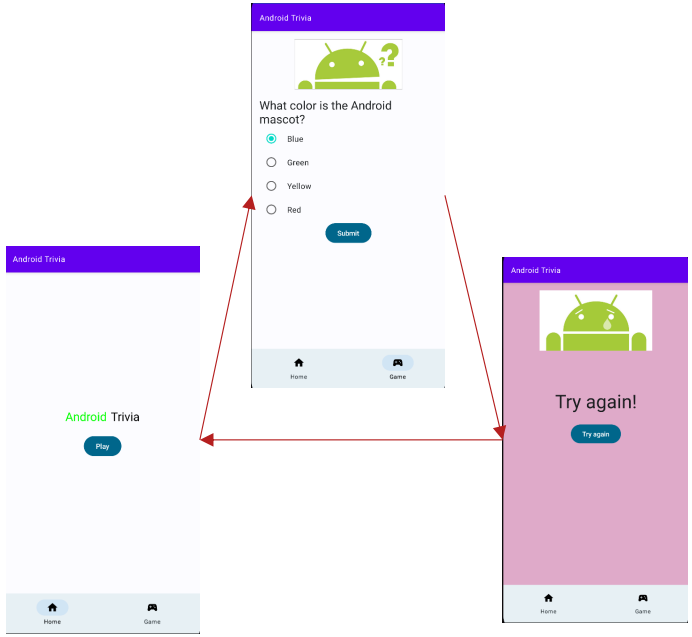
```
val drawerState = rememberDrawerState(initialValue = DrawerValue.Closed)
val scope = rememberCoroutineScope() // creates a scope for background operations

ModalNavigationDrawer( … ) {
  Scaffold(
        ….
        navigationIcon = {
          IconButton(onClick = {
            scope.launch { // "launches" the code in the block to be done in the background
              drawerState.open()
            }
          }) { … }
```

Class outline

- Screens and navigation
  - passing arguments between screens
- Menus
  - bottom menu
  - options menu
  - navigation drawer

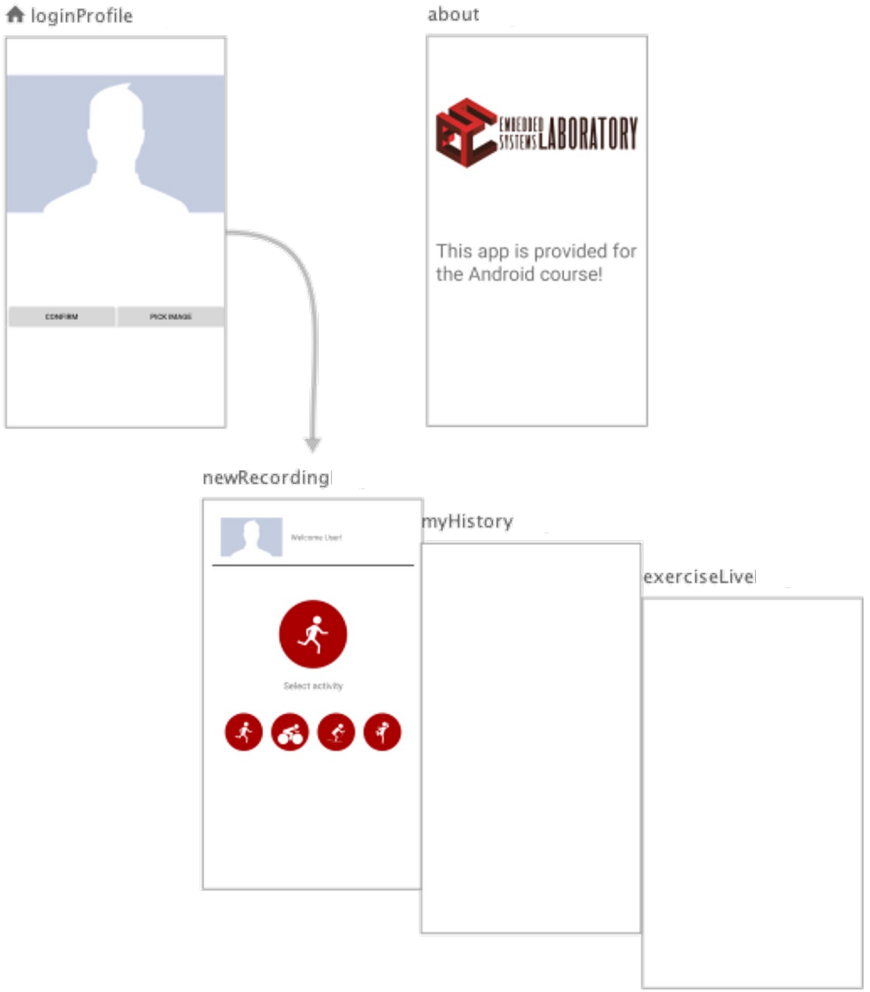




Lab of today

- Develop multiple Screens for the SportTracking application

- Provide navigations capabilities
  - Actions
  - BottomMenu
  - DrawerMenu



Questions?