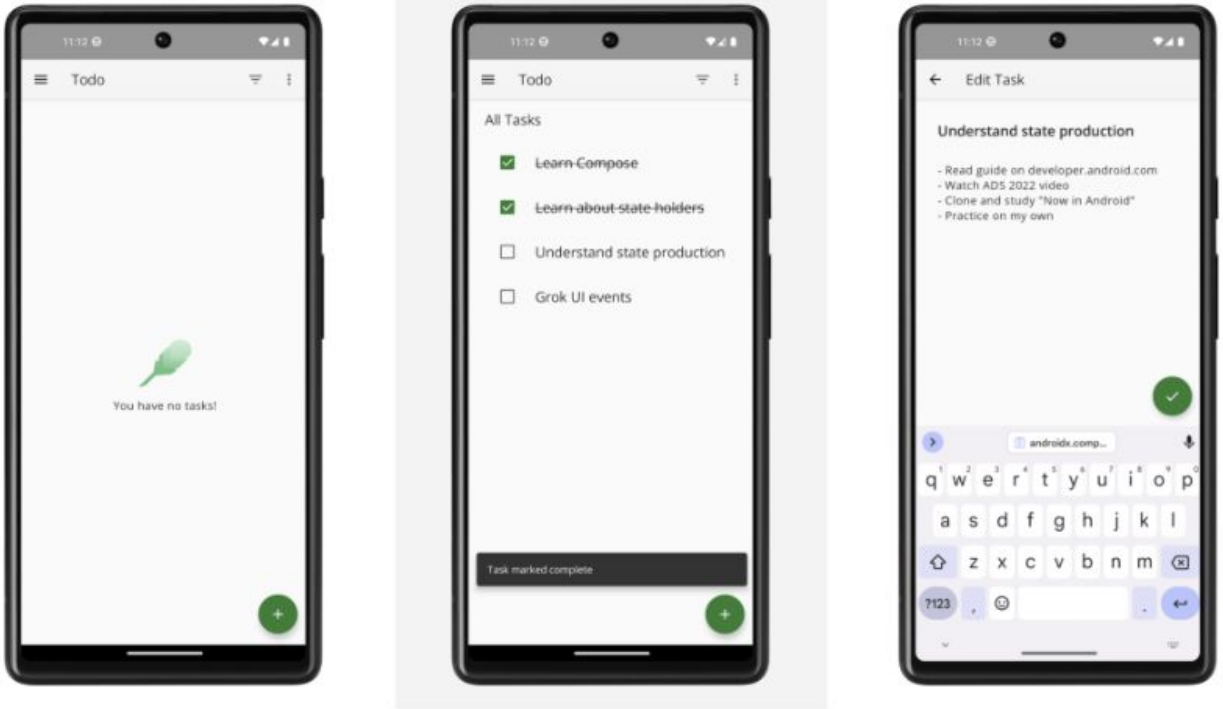
Predavanje br. 10

Navigation

Sadržaj

- Navigacija
- BottomBar

Single Activity vs. Multiple Activities



Preporuka

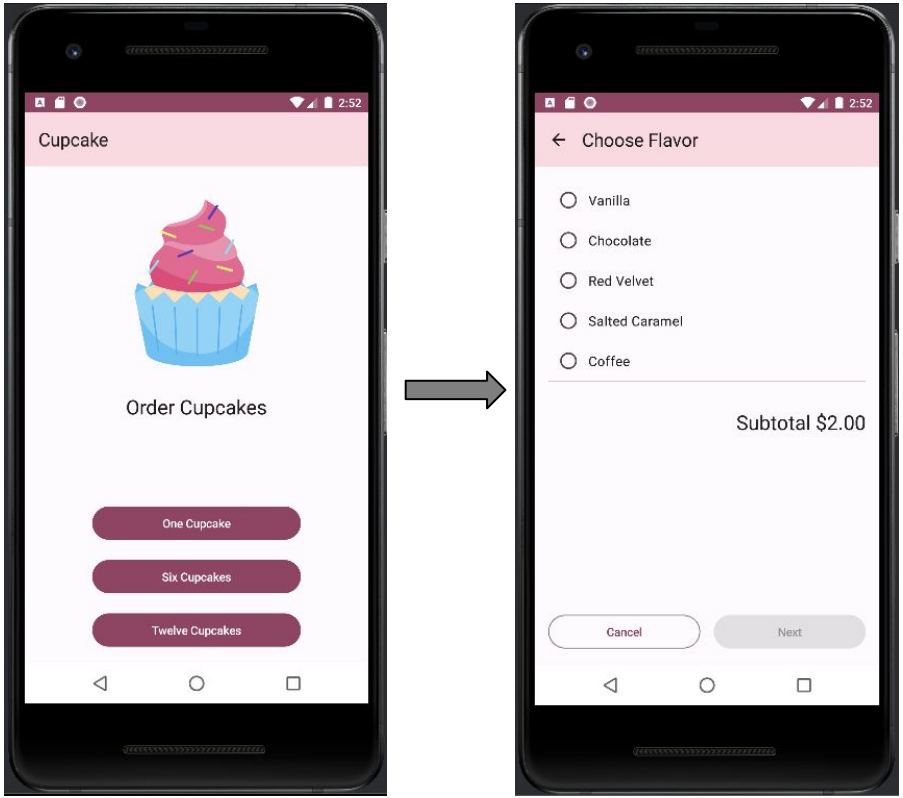
- Single-Activity arhitektura
  - Savremena preporuka je koristiti jednu Activity kao "host", a sve ekrane graditi kroz Jetpack Navigation. Ovo pojednostavljuje lifecycle, navigaciju i dijeljenje podataka.

- Korištenje Jetpack Compose-a za UI
  - Compose je preporučeni moderni UI toolkit za Android. Omogućava deklarativno kreiranje korisničkog interfejsa, jednostavniji rad sa state-om i bržu izgradnju funkcionalnog UI-ja.

- Navigation komponenta za kretanje kroz aplikaciju
  - Umjesto više aktivnosti i Intent-a, preporučuje se Navigation Component za upravljanje ekranima. Omogućava jasne rute, animacije, deep linkove i jednostavnu integraciju sa ViewModelima.

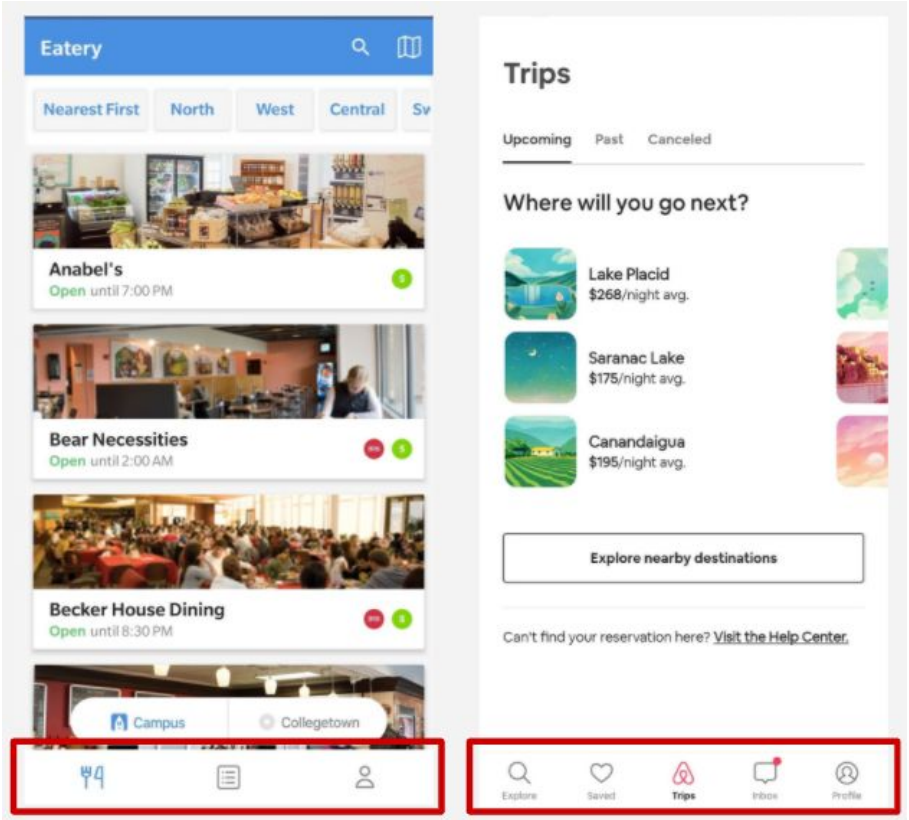
Motivacija

- Dobre aplikacije zahtijevaju mnogo ekrana!
- Mnogo ekrana zahtijeva dobru navigaciju!
- Možete sami implementirati navigaciju (uslovnim prikazivanjem ekrana) ili koristiti ugrađenu biblioteku kao što je NavHost!



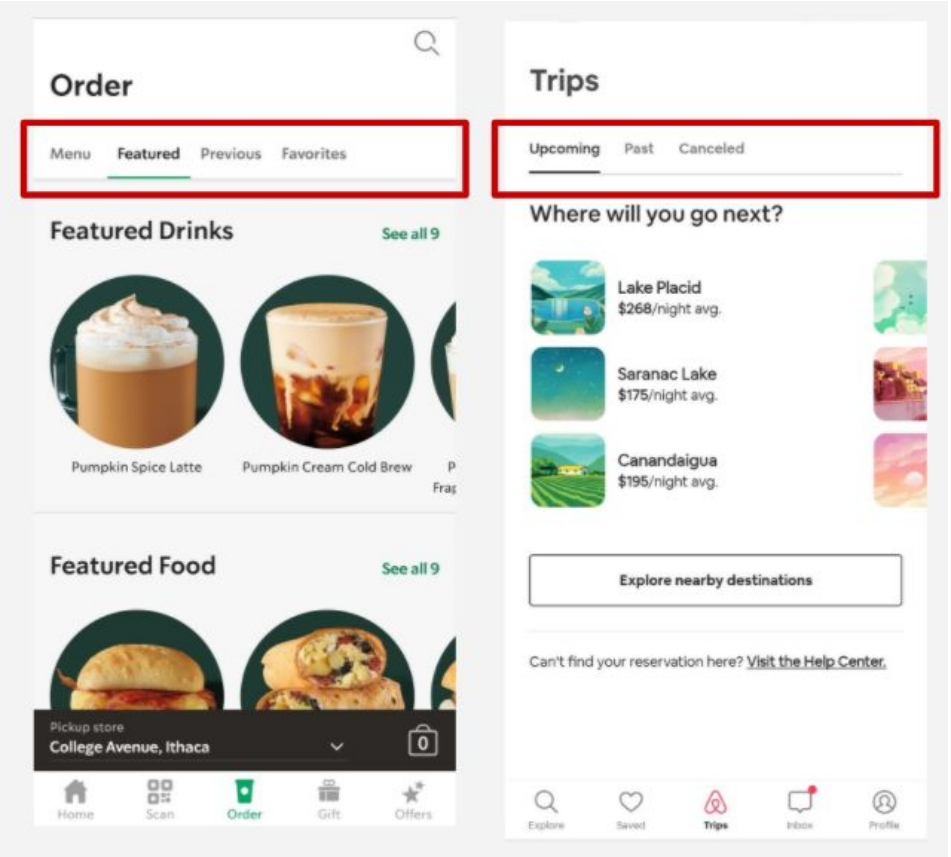
Vrste navigacija

- BottomBar



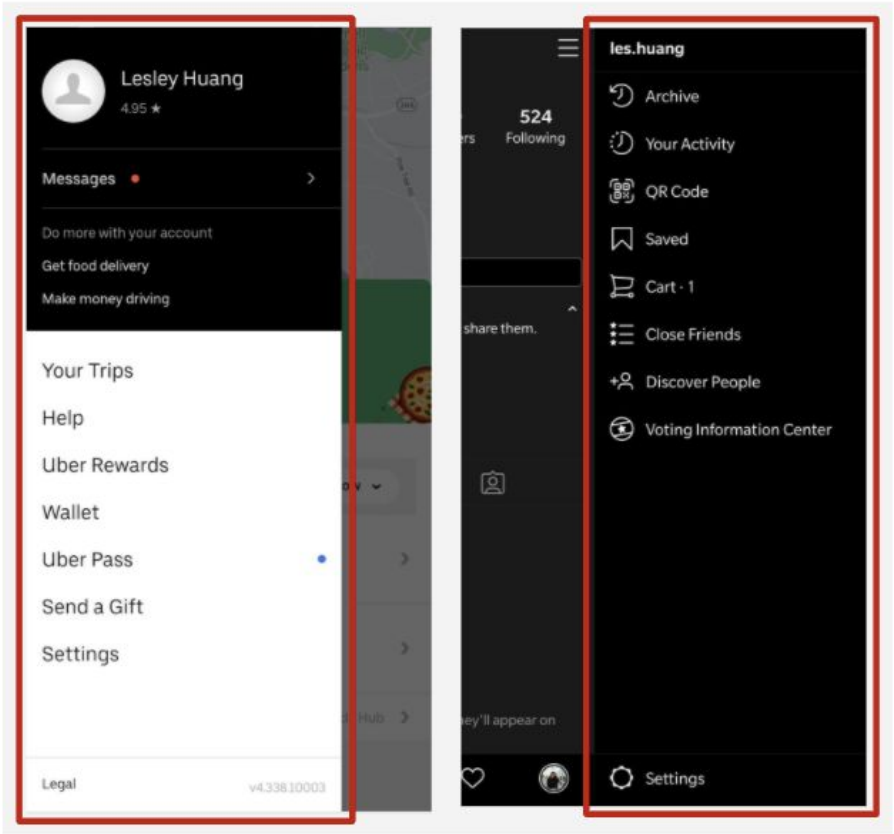
Vrste navigacija

- BottomBar
- Tab Layout



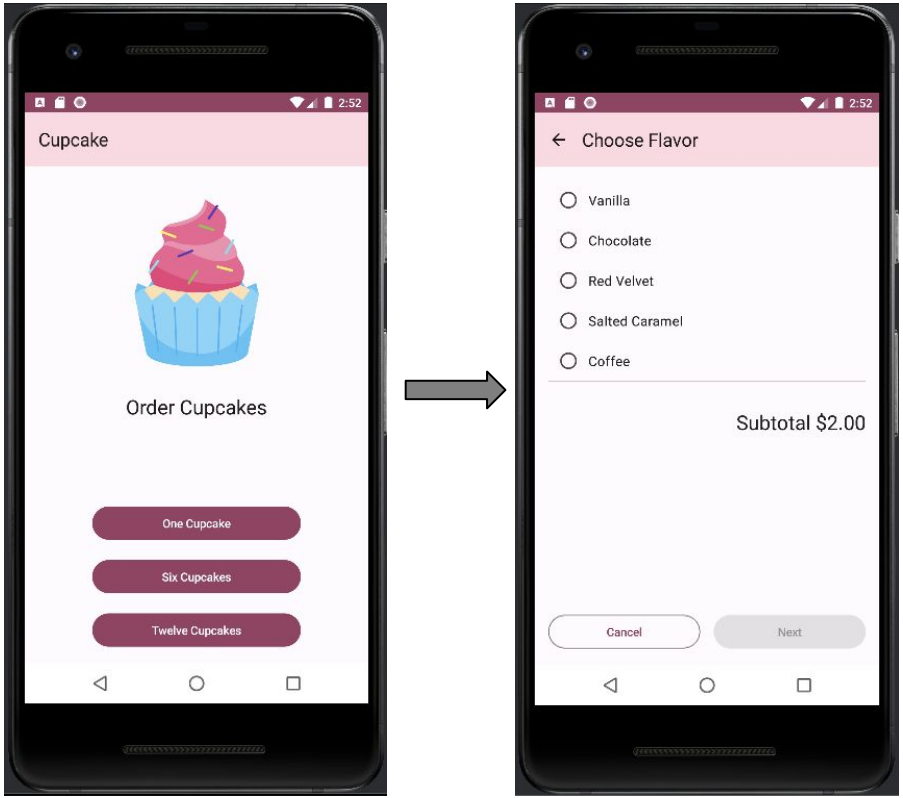
Vrste navigacija

- BottomBar
- Tab Layout
- Hamburger Menu



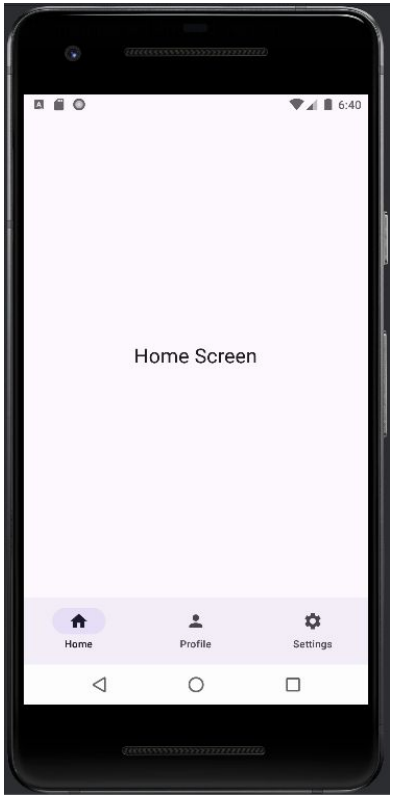
Ekran je komponenta

- U suštini, ekran je samo jedna velika komponenta!
  (samo što se ta komponenta obično poziva jednom.)
- To znači da treba da:
  - Svaki ekran ima svoj .kt fajl
  - Sadrži jednu @Composable funkciju
  - Ima @Preview, ako je moguće
- Poziva se zasebno unutar NavHost-a



StartOrderScreen *SelectOptionScreen*

BottomBar primjer



NavHost

- Da biste koristili NavHost i druge naprednije biblioteke, potrebno je dodati odgovarajuću dependency u build.gradle (app):

```
implementation("androidx.navigation:navigation-compose:2.8.6)
```

- 3 koraka za korištenje NavHost-a:
  1. Napraviti NavController
  2. Definisati Nav Graph
  3. Pozvati .navigate() da se krećete kroz ekrane!

Korak 1. NavController

- Da biste navigirali između ekrana, svaki ekran mora imati referencu na isti NavController.

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

- Zato je potrebno prosljeđivati navController kroz sve ekrane!

Korak 2. NavGraph

- Zatim definišete rute na koje se korisnik može kretati:

```
NavHost(...) {
    composable<MyRoutes.Home> {
        // …
    }

    composable<MyRoutes.Search> {
        // …
    }
}
```

Svaki **composable** predstavlja jedan ekran / rutu!

Korak 2. NavGraph

```
NavHost(...) {
    composable<MyRoutes.Home> {
        // …
    }

    composable<MyRoutes.Search> {
        // …
    }
}
```

- MyRoutes je sealed interface ili sealed class koja definiše sve ekrane aplikacije kao tipove.

```
sealed interface MyRoutes {
    @Serializable
    object Home : MyRoutes

    @Serializable
    object Search : MyRoutes
}
```

Korak 2. NavGraph

- Zatim definišete rute na koje se korisnik može kretati:

```
NavHost(...) {
    composable("home") {
        // …
    }

    composable("search") {
        // …
    }
}
```

Svaki **composable** predstavlja jedan ekran / rutu!

Korak 2. NavGraph

- Zatim definišete rute na koje se korisnik može kretati:

```
NavHost(...) {
    composable("home") {
        HomeScreen()
    }

    composable("search") {
        SearchScreen()
    }
}
```

Obično svaka ruta odgovara jednom ekranu i ničemu više.

(Pravite ekrane kao samostalne komponente!)

Korak 3. Navigacija preko .navigate()

```
NavHost(...) {
    composable(“home”) {
        HomeScreen()
    }

    composable(“search”) {
        SearchScreen()
    }
}
```

```
// za navigaciju prema home
navController.navigate(“home”)

// za navigaciju prema search
navController.navigate(“search”)
```

Koristite **.navigate()** sa odgovarajućom rutom da biste prešli na željeni ekran!

Korak 3. Navigacija preko .navigate()

```
NavHost(
    navController = navController,
    startDestination = "home"
) {
    composable("home") {
        HomeScreen(
            onCancel = { navController.popBackStack() }
        )
    }
    …
}
```