Predavanje br. 6

Ponavljanje

Primjer

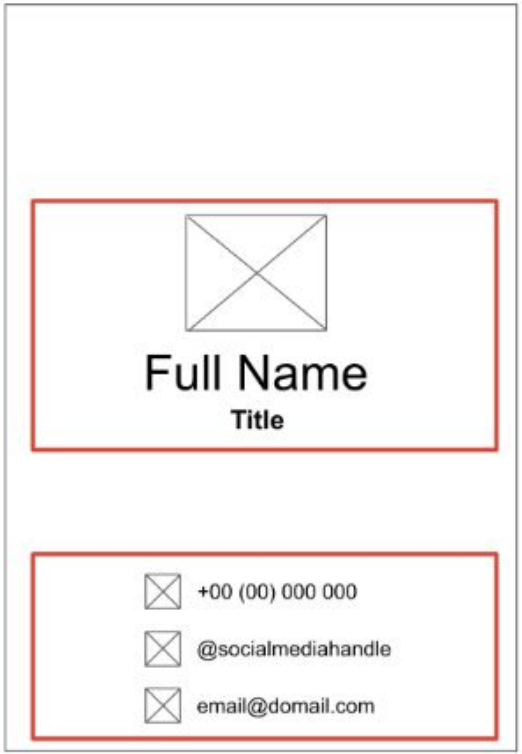

12:00
android
Jennifer Doe
Android Developer Extraordinaire
+11 (123) 444 555 666
@AndroidDev
jen.doe@android.com


Full Name
Title
+00 (00) 000 000
@socialmediahandle
email@domail.com

Primjer

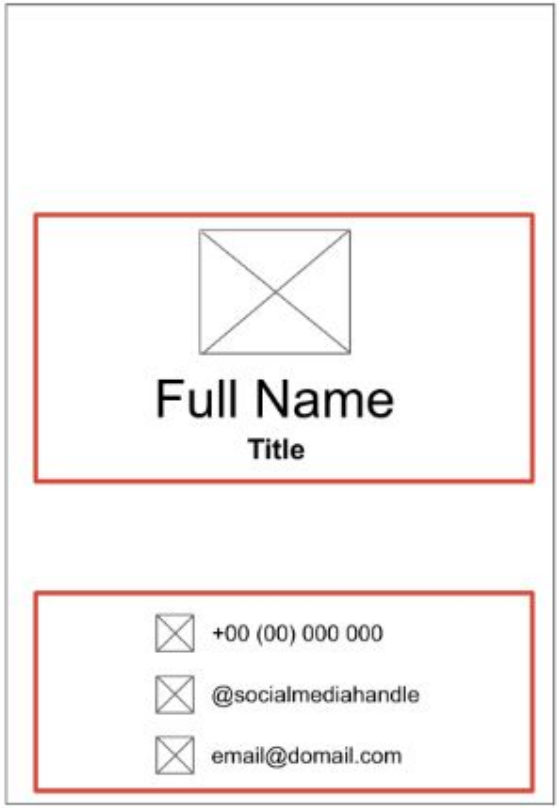

12:00
android
Jennifer Doe
Android Developer Extraordinaire
+11 (123) 444 555 666
@AndroidDev
jen.doe@android.com

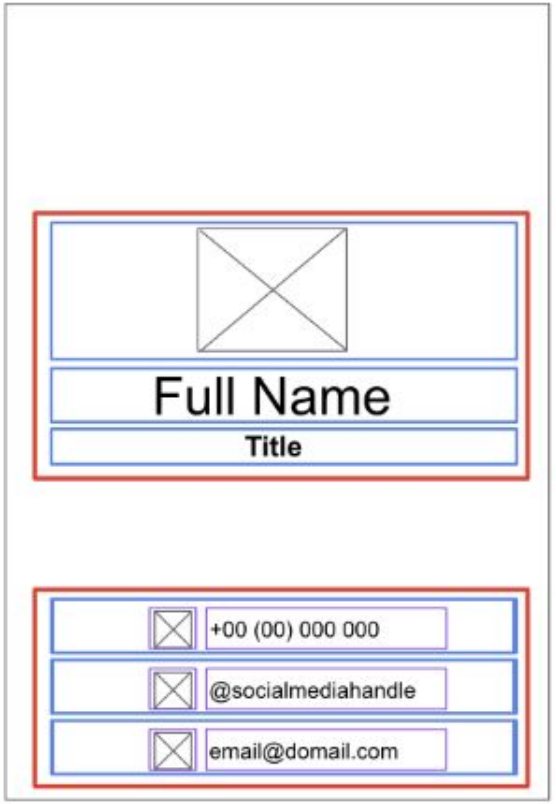

Full Name
Title
+00 (00) 000 000
@socialmediahandle
email@domail.com


Full Name
Title
+00 (00) 000 000
@socialmediahandle
email@domail.com

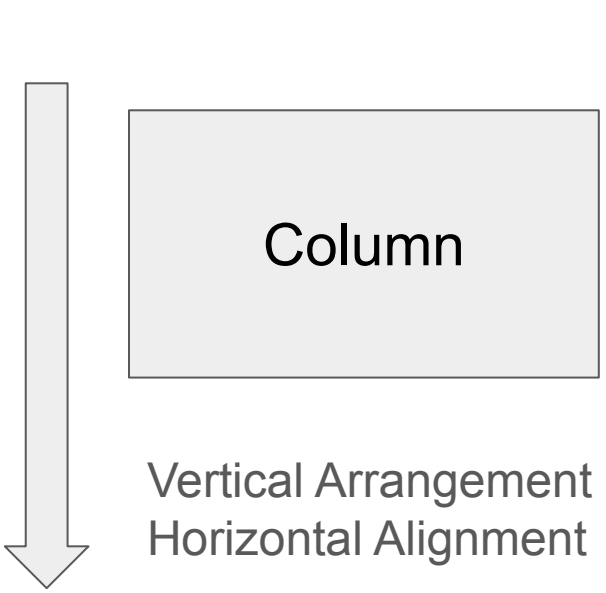
Primjer


12:00
android
Jennifer Doe
Android Developer Extraordinaire
+11 (123) 444 555 666
@AndroidDev
jen.doe@android.com

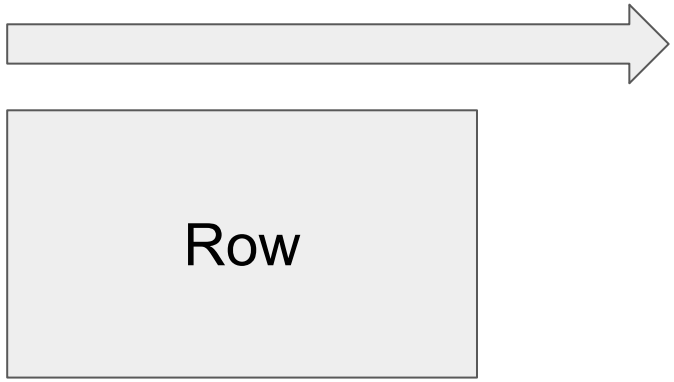

Full Name
Title
+00 (00) 000 000
@socialmediahandle
email@domail.com


Full Name
Title
+00 (00) 000 000
@socialmediahandle
email@domail.com

Row & Column layout

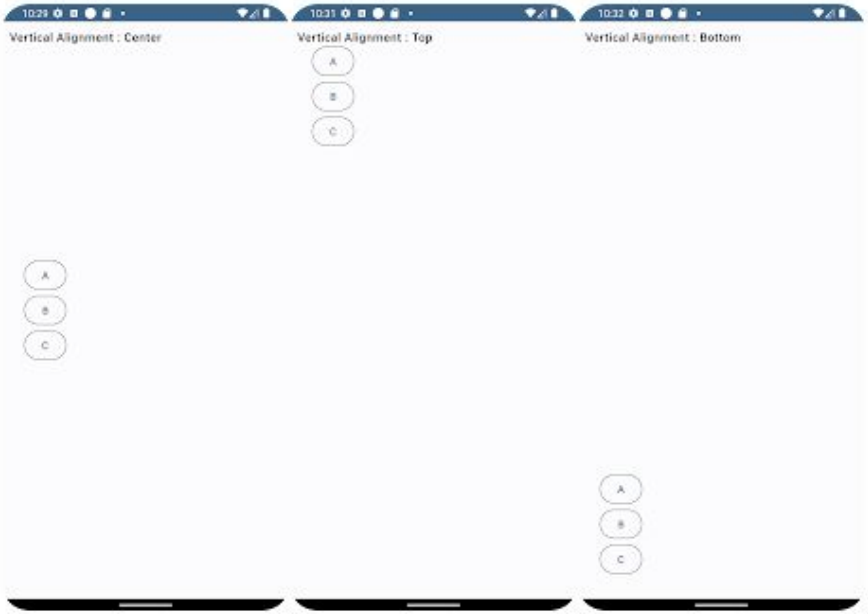


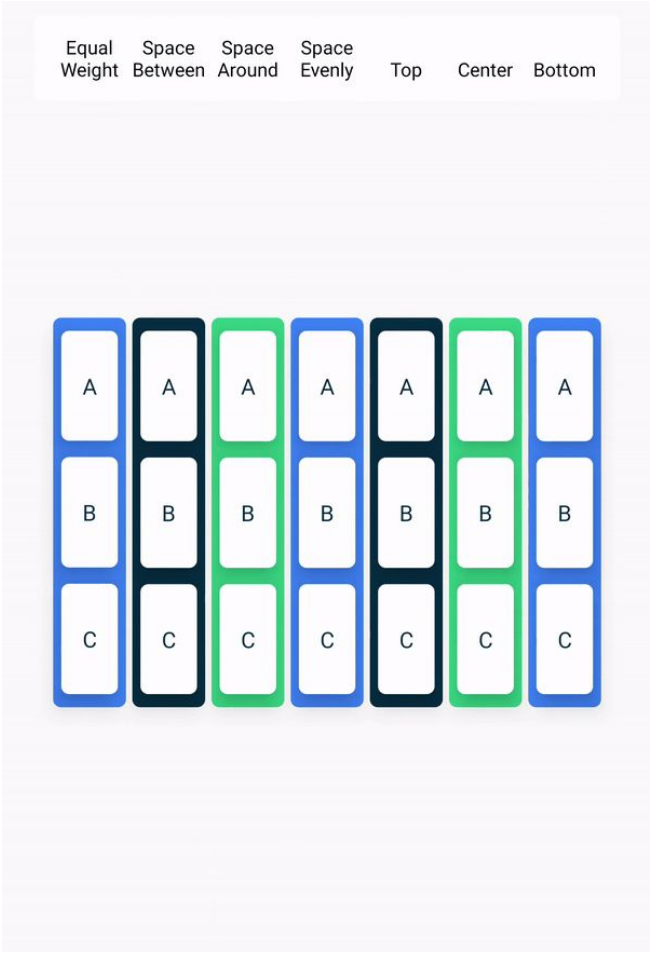
Vertical Arrangement
Horizontal Alignment



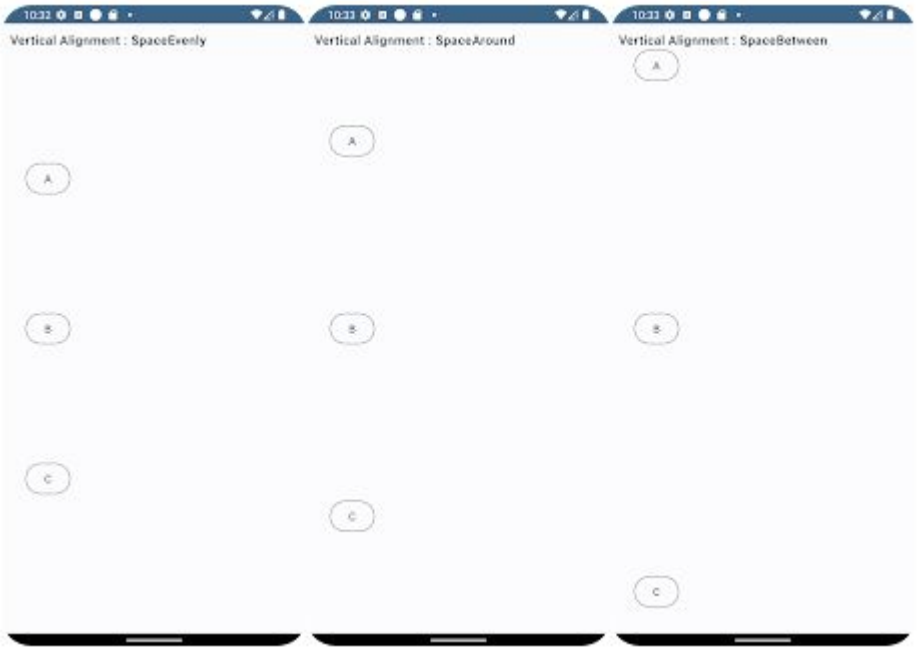
Horizontal Arrangement
Vertical Alignment

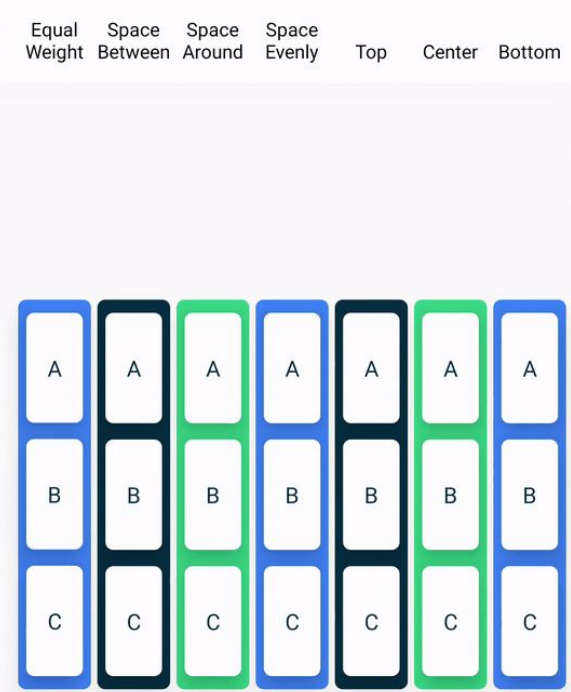
Column - Vertical Arrangement





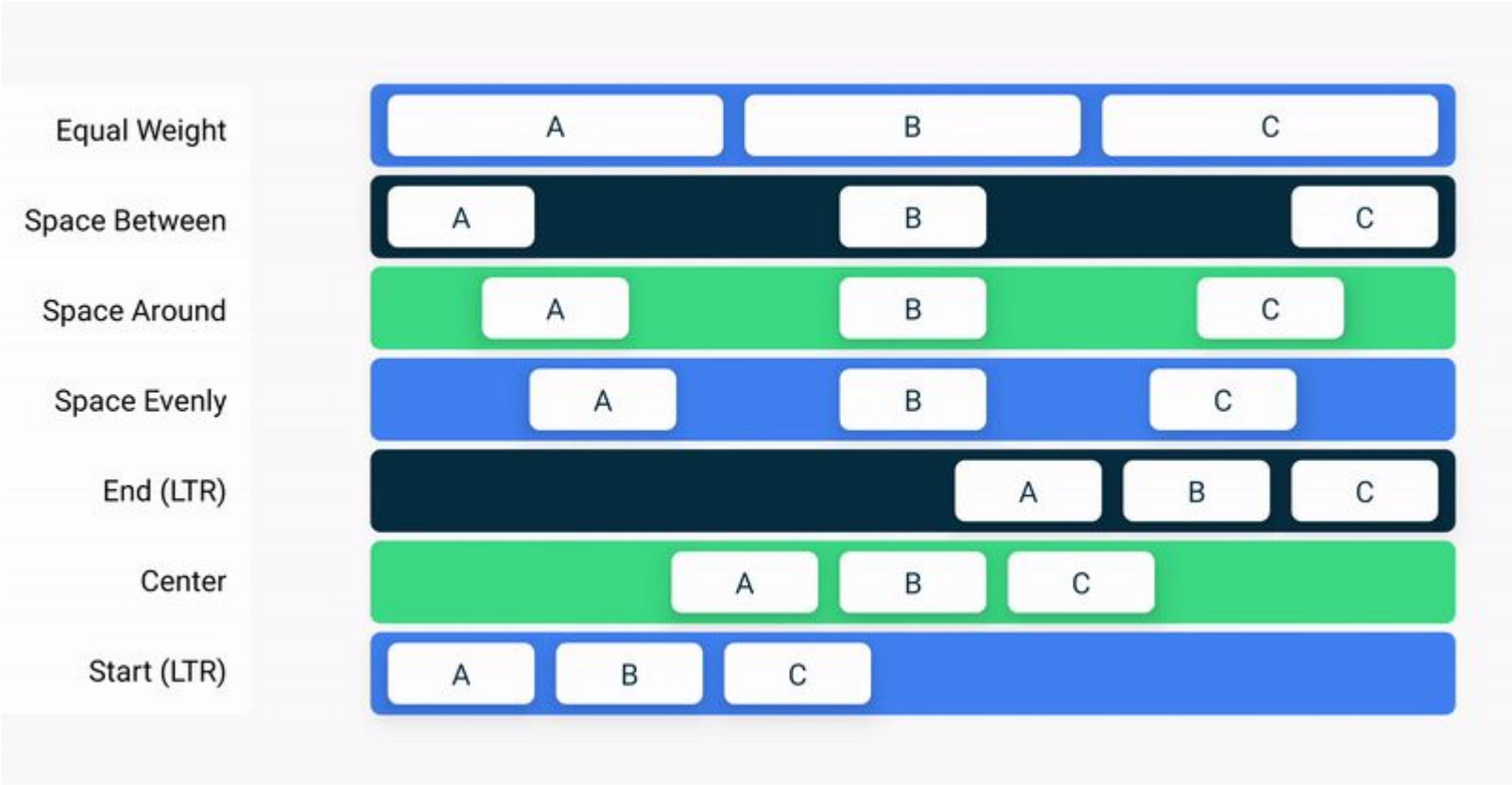
Column - Vertical Arrangement



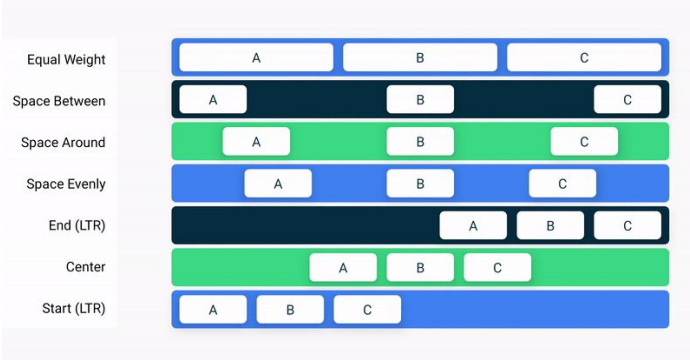


Column - Horizontal Alignment

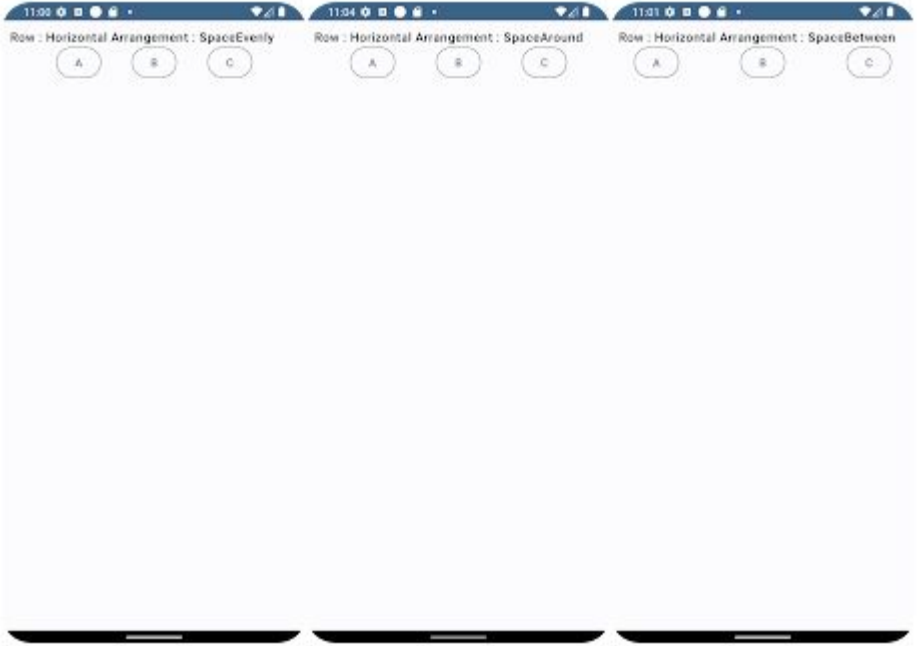
Row

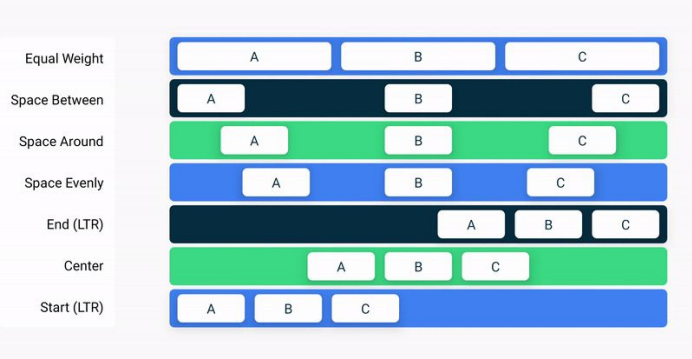

Equal Weight
A
B
C
Space Between
A
B
C
Space Around
A
B
C
Space Evenly
A
B
C
End (LTR)
A
B
C
Center
A
B
C
Start (LTR)
A
B
C

Row - Horizontal Arrangement

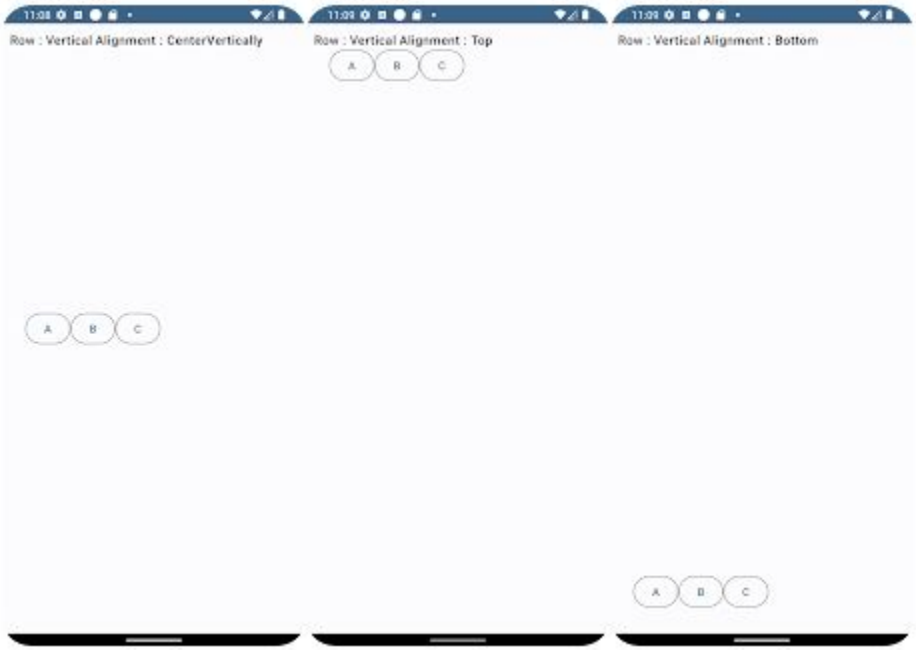


Row - Horizontal Arrangement





Row - Vertical Alignment



Funkcije vs. Components

Components <-> UI
Funkcije <-> logika

Komponente su u suštini funkcije – ali za vaš korisnički interfejs (UI)!

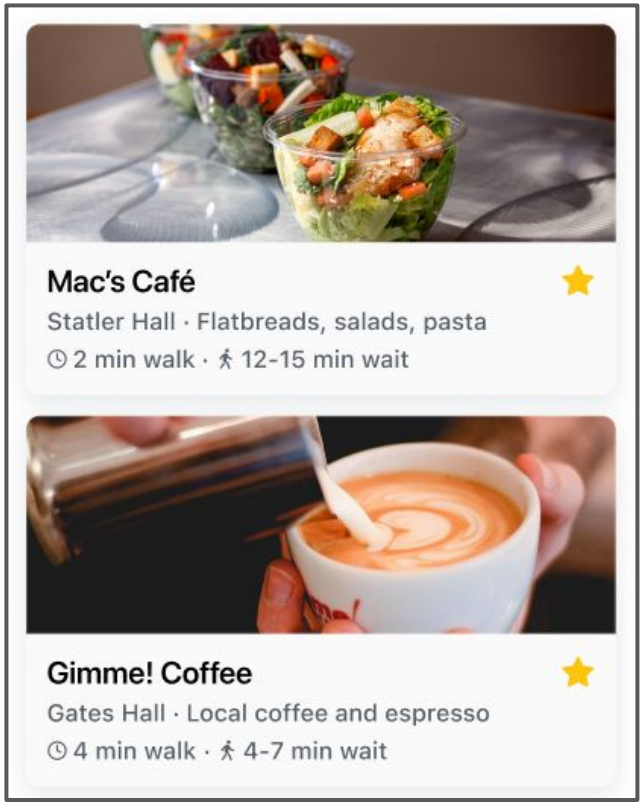
- Obje omogućavaju:
  - Ponovno korištenje koda
  - Prosljeđivanje argumenata za promjenu funkcionalnosti
  - Organizaciju koda i fajlova
  - Apstrakciju i jednostavnost korištenja (posebno kada su dobro definisane)

Composable & Komponente

- Već smo vidjeli neke ugrađene komponente (Row, Column, Box, Button, Text, Image ...).

- Možemo pisati nove komponente koje pozivaju ove ugrađene komponente, kao i druge prilagođene komponente.

Primjeri komponenti



Ista `EateryCard` kompoenta, koja prikazuje drugačiji UI zato što komponente uzimaju druge argumente.

Argumenti (primjeri):

- Ime restorana
- Lokacija restorana
- Omiljeni?
- Slika

i dr....

Primjer

```
@Composable
fun EateryCard(
    eatery: Eatery,
    isFavorite: Boolean,
    modifier: Modifier = Modifier.fillMaxWidth(),
    ...
) {
    ...
}
```

Komponente

Kada razvijamo komponente, često je veoma korisno vidjeti ono što trenutno gradimo!
(U suprotnom, mogli bismo pogrešno postaviti padding, veličinu teksta itd.)

Rješenje: **@Preview** funkcije!

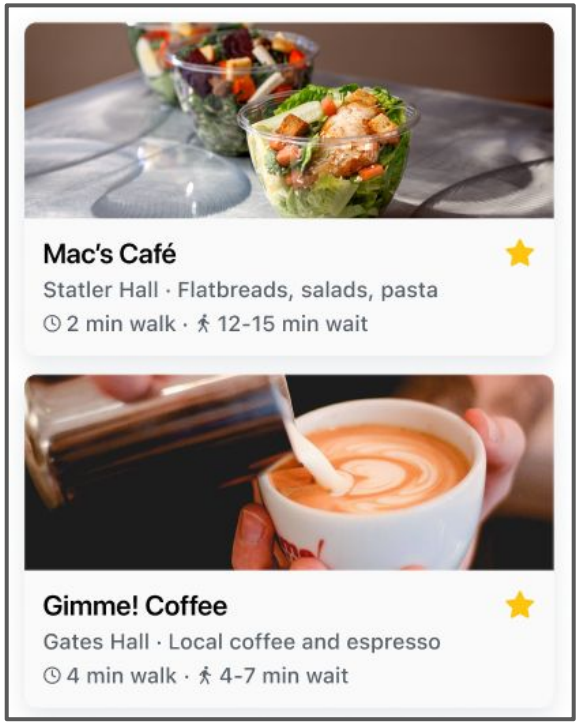
Potencijalno rješenje: Kreirajte poseban fajl za svaku komponentu, a zatim dodajte **@Preview** funkciju za tu komponentu kako biste je mogli razvijati i testirati zasebno.

Dobivanje ulaza / podataka OD komponente

Argumenti nam omogućavaju da šaljemo podatke DOLE u komponentu...

ali kako da dobijemo podatke ili korisnički unos OD komponente?

(Na primjer, kako možemo saznati kada je neki od ovih `EateryCard` elemenata kliknut, kako bismo mogli preći na drugi ekran?)



OnClick / Lambda funkcije

- Proslijediti `onClick` (ili neku drugu [Action] funkciju) komponenti:

```
onClick: () -> Unit
// Sintaksa funkcije koja ne prima argumente i
// ne vraća ništa.
```

- ...onda obavezno pozovite ovu funkciju gdje je potrebno!

```
Modifier.clickable(onClick = onClick)
```

Primjer

```
@Composable
fun EateryCard(name: String, onClick: () ->
Unit) {
    Card(
        modifier = Modifier
            .fillMaxWidth()
            .padding(8.dp)
            .clickable(onClick =  onClick)
    ) {
        Text(name, Modifier.padding(16.dp))
    }
}
```

```
@Composable
fun EateryList() {
        val eateries = listOf("Cafe
Cafe", "Just")
        Column {
            eateries.forEach { eatery ->
                EateryCard(name = eatery,
onClick = { Log.d("Kliknuto: $eatery") })
            }
        }
    }
```

State

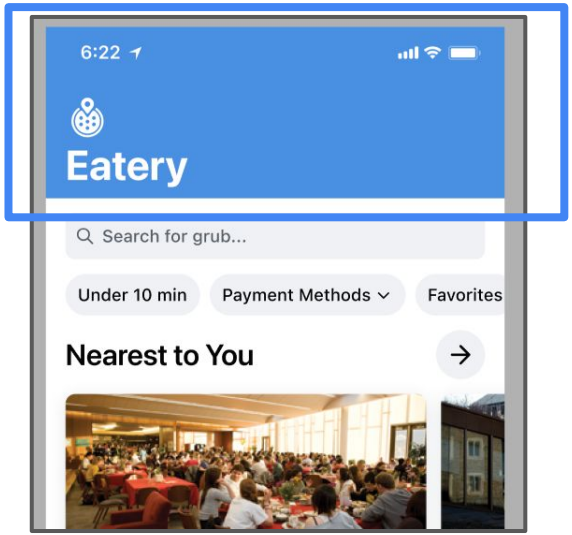
& rekompozicija

UI elementi & UI state

- **UI state** su podaci koje aplikacija navodi da bi korisnici trebali vidjeti
- **UI elements** su povezani sa UI stanjem kako bi se stvorio UI
- **UI** je ono što korisnik vidi

Dinamični podaci

Nije svim podacima potrebno stanje. Na primjer...



Eatery zaglavlje će uvijek pisati "Eatery"--nema potrebe za stanjem! (Možemo samo hard-code string)

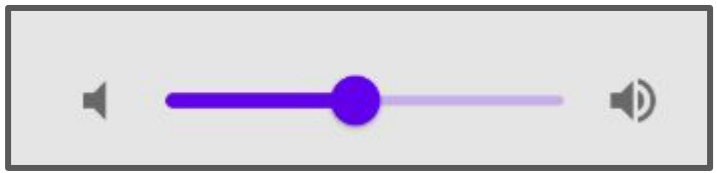
Stanja su potrebna samo da bi se korisničkom interfejsu rekli dinamički (**promjenjivi**) podaci.

Primjeri promjene stanja

- Neki jasni primjeri komponenti koje trebaju koristiti stanje da bi odredile šta trebaju prikazati (na osnovu korisničkog unosa):

Tekstualna polja! Kako korisnik kuca, stanje otkucanog teksta se mora mijenjati.

Klizači! Kako korisnik pomiče dugme, vrijednost se mora mijenjati.



Konceptualna ideja

- Ako bilo koji dio ekrana treba prikazivati dinamičke (promjenjive) podatke, potrebno je koristiti `State` u Compose-u da bi se UI mogao mijenjati.

`State<T>` klasa

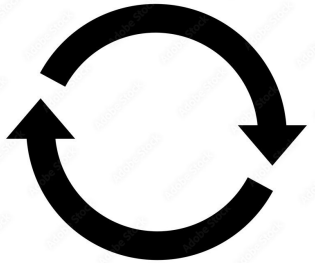
- Osnovna klasa koja objašnjava kako Jetpack Compose funkcioniše pri prikazivanju UI-ja.

- `T` je generički tip koji može predstavljati bilo koji tip podataka!
  - e.g. `State<Int>, State<Double>, State<List<Pair<Int, Int>>>`

Životni ciklus recomposition-a

- Compose pokušava biti **efikasan**. Ponovo će iscrtat (re-render) samo onaj dio UI-ja (Composable) kada mu se to “kaže”.

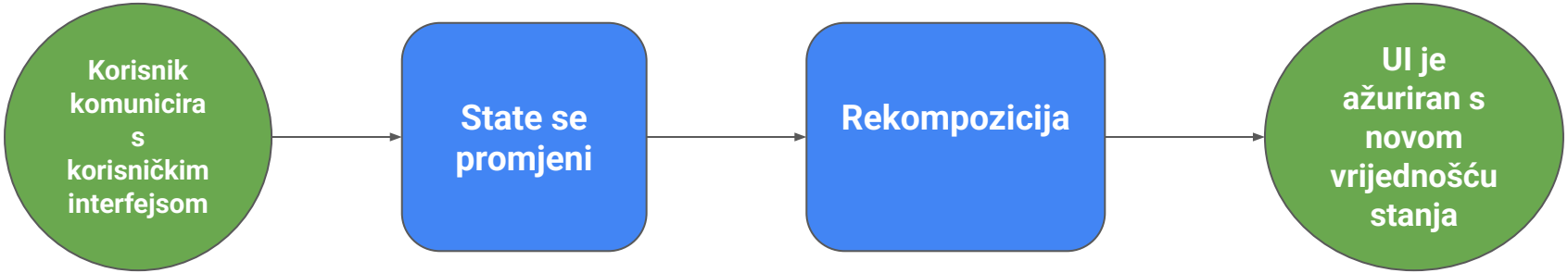
- Compose koristi `State<T>` da odredi kada treba promijeniti UI, odnosno kada treba ponovo “rekomponovati” UI.



Ideja:

- Kad god se promijeni vrijednost tipa `State<T>`, avaki dio UI-j taj state automatski će se ažurirati!
  - (taj UI će "*reagovati*"?)

Rekompozicija



Primjer:

Korisnik kuca znak u tekstualno polje
→ State koji prati vrijednost stringa tekstualnog polja je ažuriran
→ Compose ponovo renderira kompozitne elemente koji sadrže tekstualno stanje
→ UI prikaže ažurirani tekst

MutableState<T>

- Deklarišemo novu state vrijednost (s početnom vrijednošću, npr. "Android" i 1.00):

```
val name = remember { mutableStateOf("Android") }
val cost = remember { mutableStateOf(1.00) }
```

- **mutableStateOf** kreira posmatrani (observable) objekat tipa **MutableState<T>** i pokreće *recomposition* svaki put kada se njegova vrijednost promijeni.
- **remember** čuva objekat u memoriji tokom više recomposition-a.
- Bez remember, promjena **MutableState** vrijednosti bi pokrenula recomposition,ali bi se UI i dalje prikazivao s početnom vrijednošću.

MutableState<T>

```
name.value = "Custom"
cost.value = 4.00
// UI
Text(text = name.value)
```

Čitanje i upisivanje u ove state vrijednosti se vrši pomoću `.value`

Glavna ideja: Kako su `name` i `cost` `State` varijable, one će sada **automatski** uzrokovati da se UI ažurira svaki put kada se njihove vrijednosti promjene. Ako biste koristili obične varijable tipa `String` i `Double`, UI se ne bi promjenio

Napomena: by ključna riječ

```
var name by remember { mutableStateOf("Android") }
// …
name = "Custom"
Text(text = name)
```

- Ako vam je previše teško pisati svaki put `.value` za svako čitanje i pisanje, možete koristiti `by` ključnu riječ.

- Ova ključna riječ pretvara state vrijednost u običnu promjenjivu kojom možete direktno upravljati (čitati i mijenjati), ali Compose i dalje u pozadini koristi `State` da bi izazvao rekompoziciju.

Na kraju: povezivanje `Components` & `State`

Kada razvijate komponentu, vjerovatno želite da se ona automatski ažurira kad se promijeni njen `State`.

Međutim, vrlo često ne morate direktno dodavati `State` unutar same komponente.
Sve dok:

1. Prosljeđujete obične vrijednosti kao argumente komponenti
2. Koristite `State` vrijednosti izvan komponente (u roditelju) kako biste pokrenuli recomposition

...vaše će se komponente automatski ažurirati zahvaljujući recomposition životnom ciklusu!

Primjer

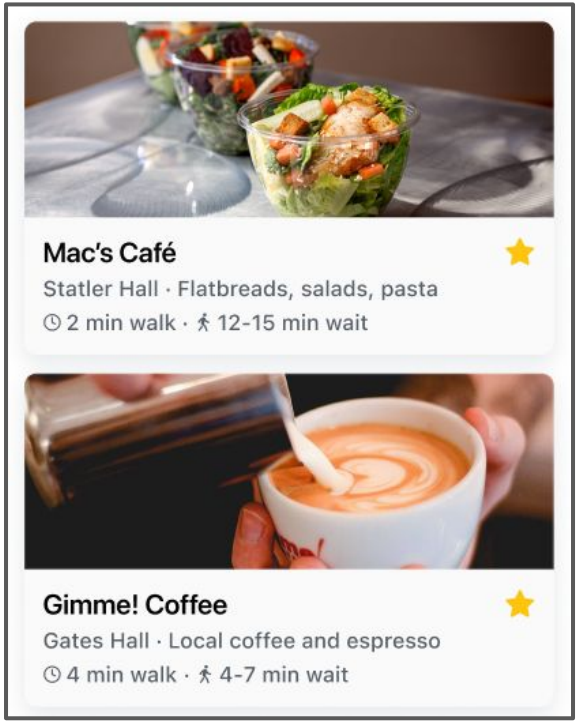
```
@Composable
fun GreetingCard(name: String) {
    Text("Hello, $name!")
}

@Composable
fun GreetingList() {
    var currentName by remember { mutableStateOf("Alma") }
    GreetingCard(name = currentName)
}
```

Pitanje

Koliko može komponenta ***da razmišlja?***

Drugim riječima — treba li komponenta samo primati jednostavne argumente za prikaz (npr. tekstualne vrijednosti tipa String),
ili bi trebala imati unutrašnju logiku i svoje stanje (`State`)?

Npr. da li komponenta EateryCard će samo primati tekstualne argumente (naziv, opis, vrijeme čekanja), ili bi trebala primiti ID restorana i sama izračunati svoje interno stanje?



Demo

https://github.com/fet-tk-course/RMAS2025