Google Developers

GDG

Exploring Jetpack Compose

Diving into the framework

Contents

Things we'll talk about

- What is Compose?
- Compose Basics
- State handling
- Complex composables
- Advanced Compose

What is Jetpack Compose?

What is Jetpack Compose?

The idea behind the framework

Jetpack Compose is a new and **fully declarative** way to design and render Android user interfaces.

It relies on **composable functions**, **modifiers** and **reacting to state changes** to display beautiful, material-design-inspired UI **using Kotlin**!

Well how is this better?

Straightforward benefits

Things you get right off the bat

You **no longer have to write XML** to produce the UI spec you're given!

You can have both **business logic & the UI in Kotlin**!

New, updated and decoupled framework where everything is a function!

Just call **setContent()** and that's it!

Less talk more code! :]

Composable functions

```
@Composable
fun MyComposable() {
 Text(text = "Compose is so awesome!")
}
```

```
@Composable
fun MyComposable() {
 Text(text = "Compose is so awesome!")
}
```

```
@Composable
fun MyComposable() {
  Text(text = "Compose is so awesome!")
}
```

```
@Composable
fun MyComposable() {
 Text(text = "Compose is so awesome!")
}
```

```
@Composable
fun MyComposable(text: String) {
 Text(text = text)
}
```

Basic Composables

```
Text(text = "Compose is so awesome!")

Button(onClick = { }) {
 // content
}

Icon(imageVector = vectorResource(id =))

Image(bitmap = imageResource(id =))

TextField(value = "", onValueChange = { })

FloatingActionButton(onClick = { }) {
 // content
}
```

```
Text(text = "Compose is so awesome!")

Button(onClick = { }) {
 // content
}

Icon(imageVector = vectorResource(id =))

Image(bitmap = imageResource(id =))

TextField(value = "", onValueChange = { })

FloatingActionButton(onClick = { }) {
 // content
}
```

Previewing compose elements

Previews without running the app

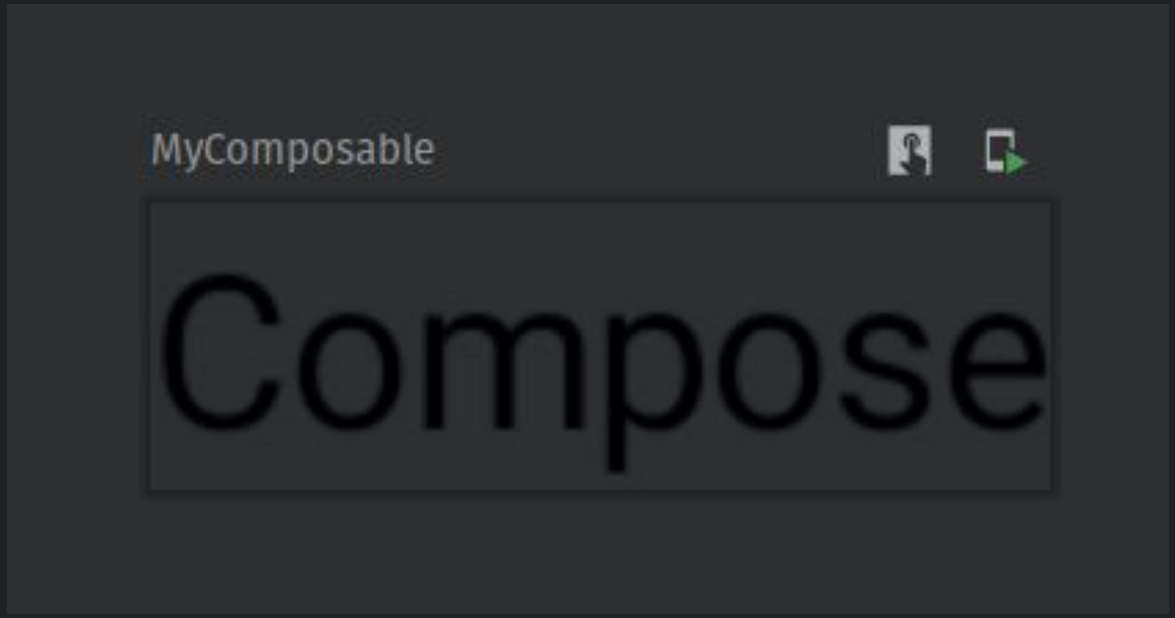
Android Studio features for Compose

Jetpack Compose lets you add **@Preview** to all your composables, to see what the elements would look like when you run the app.

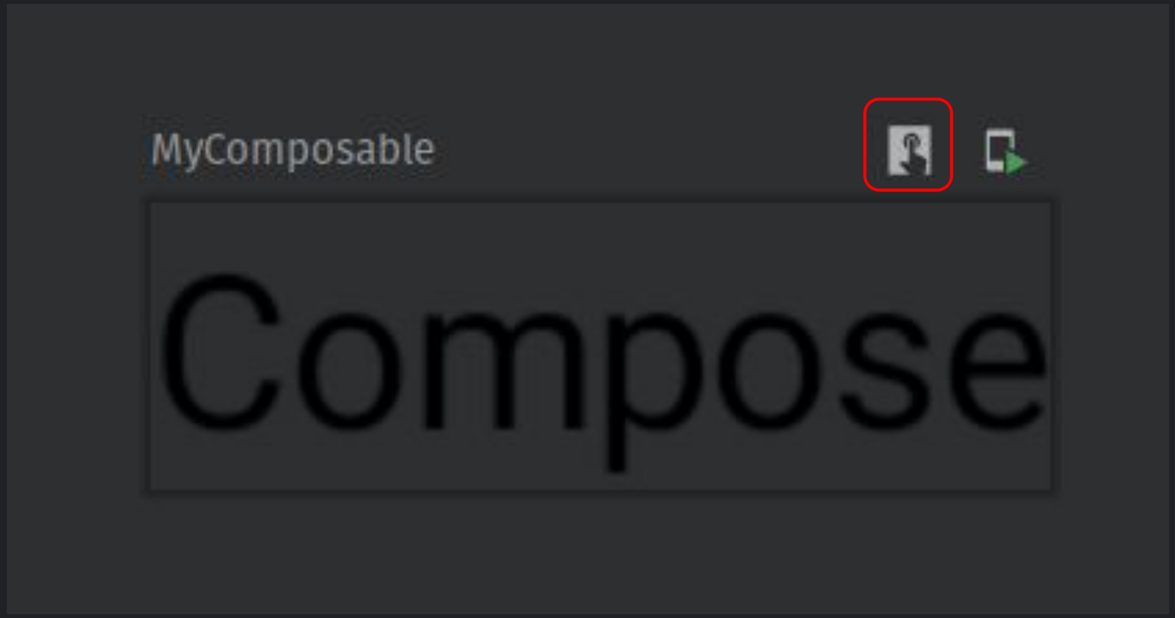
Without running the app.

You can also **interact with the elements** - like clicking it.

```
@Preview
@Composable
fun MyComposable(text: String = "Compose") {
 Text(text = text)
}
```

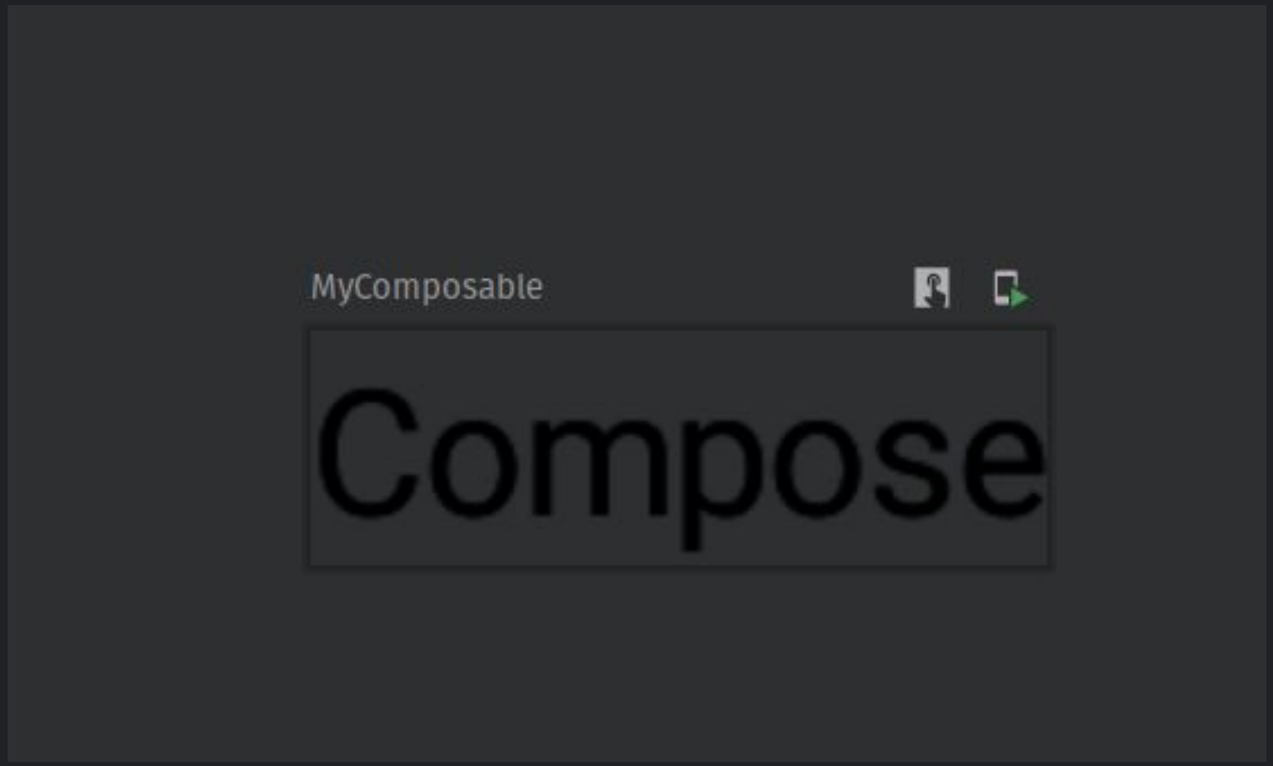


```
@Preview
@Composable
fun MyComposable(text: String = "Compose") {
 Text(text = text)
}
```



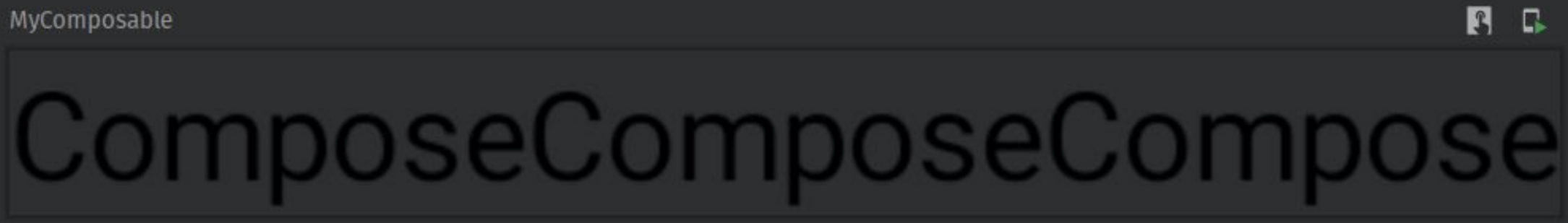
Compose Containers/Layouts

```
@Composable
@Preview
fun MyComposable(text: String = "Compose") {
 Box {
    Text(text = text)

    Text(text = text)

    Text(text = text)
 }
}
```


MyComposable
Compose

```
@Composable
@Preview
fun MyComposable(text: String = "Compose") {
 Row {
    Text(text = text)

    Text(text = text)

    Text(text = text)
 }
}
```

MyComposable



```
@Composable
@Preview
fun MyComposable(text: String = "Compose") {
 Column {
   Text(text = text)

   Text(text = text)

   Text(text = text)
 }
}
```


MyComposable
Compose
Compose
Compose

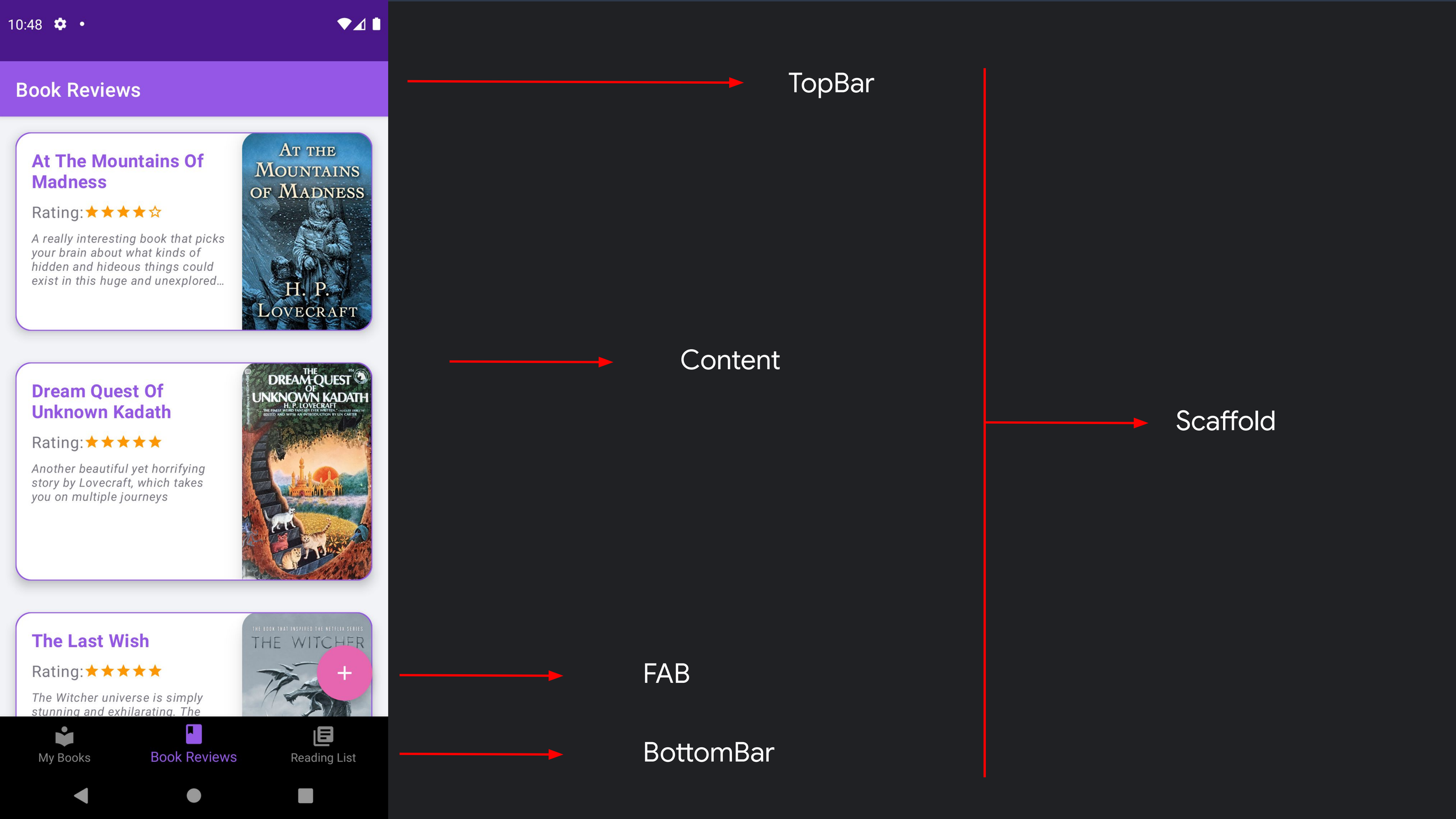
Complex/Beautiful Composables

Building Complex UI

Styling & combining composables

You can style specific UI components through **modifiers**, or you can **apply themes** across entire Compose trees.

Modifiers are styling functions you can **chain** & that **respect the chain order**. Background, alignment, size, padding, clickability...


10:48
Book Reviews
At The Mountains Of Madness
Rating:
A really interesting book that picks your brain about what kinds of hidden and hideous things could exist in this huge and unexplored...
AT THE MOUNTAINS OF MADNESS
H. P. LOVECRAFT
Dream Quest Of Unknown Kadath
Rating:
Another beautiful yet horrifying story by Lovecraft, which takes you on multiple journeys
THE DREAM-QUEST OF UNKNOWN KADATH
H. P. LOVECRAFT
The Last Wish
Rating:
The Witcher universe is simply stunning and exhilarating. The
THE WITCHER
My Books
Book Reviews
Reading List
TopBar
Content
Scaffold
FAB
BottomBar

```
@Composable
fun MainScreen() {
  Scaffold(
    topBar = { MainTopBar() },
    bottomBar = {
      BottomNavigation {
        BottomNavigationItem(icon, label, selected, onClick),
        BottomNavigationItem(icon, label, selected, onClick),
        BottomNavigationItem(icon, label, selected, onClick)
      }
    },
    floatingActionButton = { MainScreenFloatingAction() }
  ) { // content
    MainScreenContent()
  }
}
```

Row - Content | Image

LazyRow - 5 Icons

Row - Text | Rating Icons

Column - Text | Rating | Description

```
@Composable
fun BookReviewItem(
  bookReview: BookReview,
  onItemClick: (BookReview) -> Unit,
  onLongItemTap: (BookReview) -> Unit
) {
  Card(
    elevation = 8.dp,
    border = BorderStroke(1.dp, MaterialTheme.colors.primary),
    shape = RoundedCornerShape(16.dp),
    modifier = Modifier
      .wrapContentHeight()
      .padding(16.dp)
      .combinedClickable(
        onClick = { onItemClick(bookReview) },
        indication = null,
        onLongClick = { onLongItemTap(bookReview) }
      )
  ) {}
}
```

```
Row(modifier = Modifier.fillMaxSize()) {
  Spacer(modifier = Modifier.size(16.dp))

  Column(
    modifier = Modifier
      .weight(0.6f)
      .fillMaxHeight()
  ) {
    Spacer(modifier = Modifier.height(16.dp))

    Text(
      text = bookReview.book.name,
      color = MaterialTheme.colors.primary,
      fontSize = 18.sp,
      fontWeight = FontWeight.Bold
    )

    Spacer(modifier = Modifier.height(8.dp))
...
```

Book Title

```
Row {
  Text(
    text = stringResource(id = R.string.rating_text),
    color = MaterialTheme.colors.onPrimary
  )

  RatingBar(
    range = 1..5,
    currentRating = bookReview.review.rating,
    isSelectable = false,
    isLargeRating = false
  )
}

Spacer(modifier = Modifier.height(8.dp))
```

Rating

```
...
  Text(
    text = bookReview.review.notes,
    fontSize = 12.sp,
    modifier = Modifier.fillMaxSize(),
    overflow = TextOverflow.Ellipsis,
    fontStyle = FontStyle.Italic,
    maxLines = 4,
    color = MaterialTheme.colors.onPrimary
  )

  Spacer(modifier = Modifier.height(16.dp))
}
```

Description

Column end

```
Spacer(modifier = Modifier.width(16.dp))

 Card(
   modifier = Modifier.weight(0.4f),
   shape = RoundedCornerShape(
     topRight = 16.dp,
     topLeft = 16.dp,
     bottomLeft = 0.dp,
     bottomRight = 16.dp
   ),
   elevation = 16.dp
 ) {
   Image(
     painter =
        rememberImagePainter(bookReview.review.imageUrl),
     contentScale = ContentScale.FillWidth
   )
 }
}
}
```

Image Card Shape

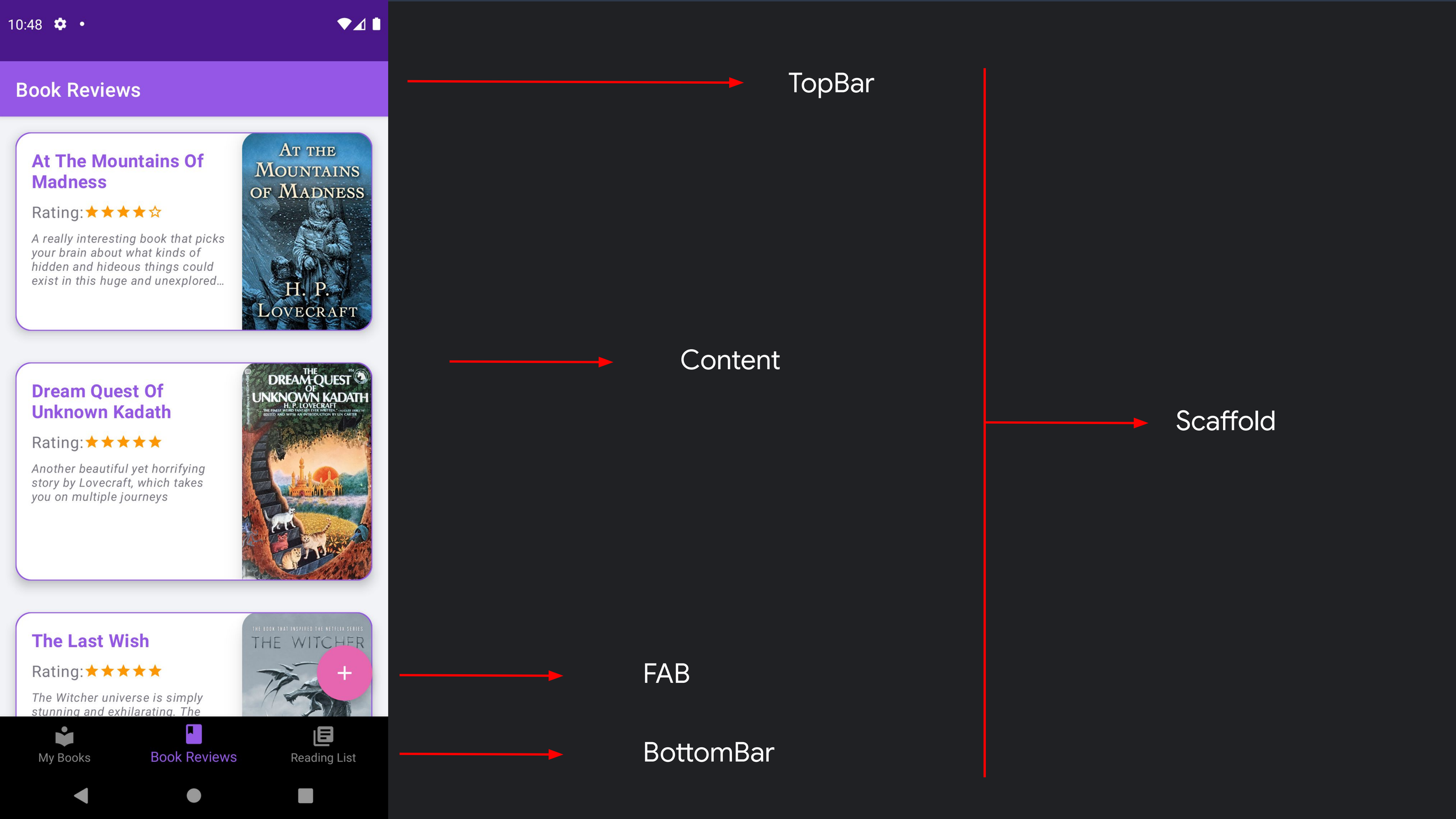
Coil powered Image

Row - Content | Image

LazyRow- 5 Icons

Row - Text | Rating Icons

Column - Text | Rating | Description


10:48
Book Reviews
At The Mountains Of Madness
Rating:
A really interesting book that picks your brain about what kinds of hidden and hideous things could exist in this huge and unexplored...
AT THE MOUNTAINS OF MADNESS
H. P. LOVECRAFT
Dream Quest Of Unknown Kadath
Rating:
Another beautiful yet horrifying story by Lovecraft, which takes you on multiple journeys
THE DREAM-QUEST OF UNKNOWN KADATH
H. P. LOVECRAFT
The Last Wish
Rating:
The Witcher universe is simply stunning and exhilarating. The
THE WITCHER
My Books
Book Reviews
Reading List
TopBar
Content
Scaffold
FAB
BottomBar

Building Lists/Dynamic Composables

Building Lists

Dynamic Composables are just Composables

Building lists and dynamic components is as easy as calling a **for-loop** or using one of the predefined “lazy components”.

LazyColumn - building **vertical** lists.

LazyRow - building **horizontal** lists.

```
@Composable
fun BookReviewsList(
 bookReviews: List<BookReview>,
 onItemClick: (BookReview) -> Unit,
 onLongItemTap: (BookReview) -> Unit
) {

 LazyColumn {
   items(bookReviews) { bookReview ->
      BookReviewItem(bookReview, onItemClick, onLongItemTap)
   }
 }
}
```

```
@Composable
@Preview
fun MyComposable(items: List<Data>) {
 for (item in items) {
   MyItem(item)
 }
}
```

Adding state

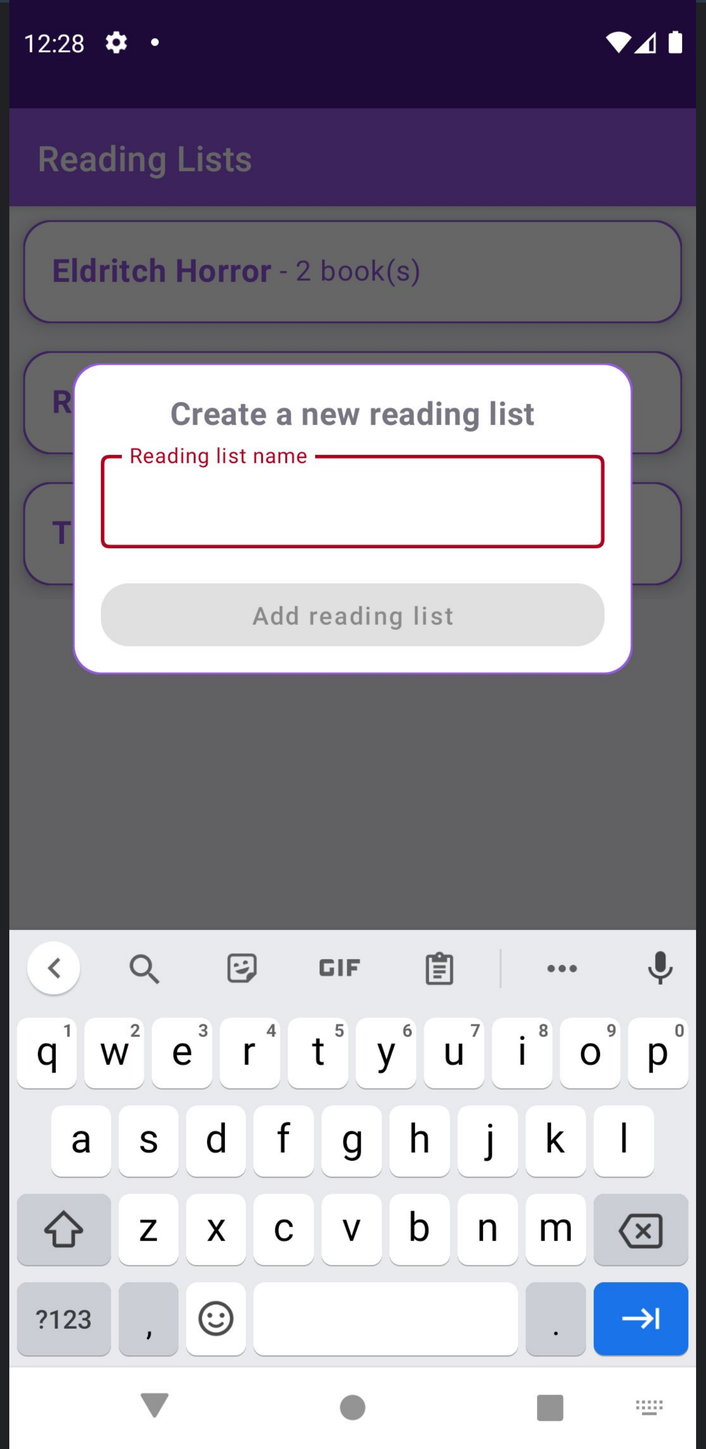
State Handling

The most important thing about UI

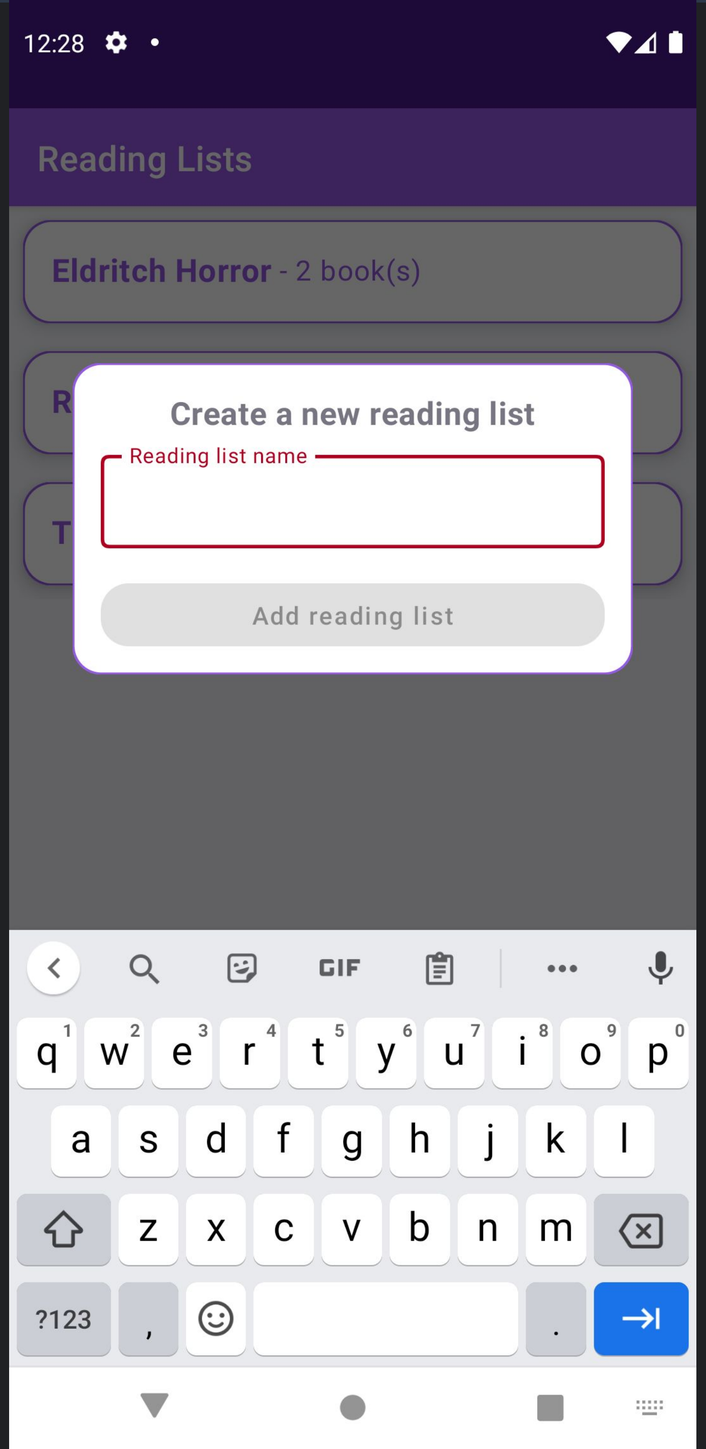
Compose allows you to easily listen to and **remember** the state of your components.

Using **remember()** you create State objects that store data within the component lifecycle.

It's also important to know that Compose doesn't work without state - **forces declarative programming**.


12:28
Reading Lists
Eldritch Horror - 2 book(s)
Create a new reading list
Reading list name
Add reading list

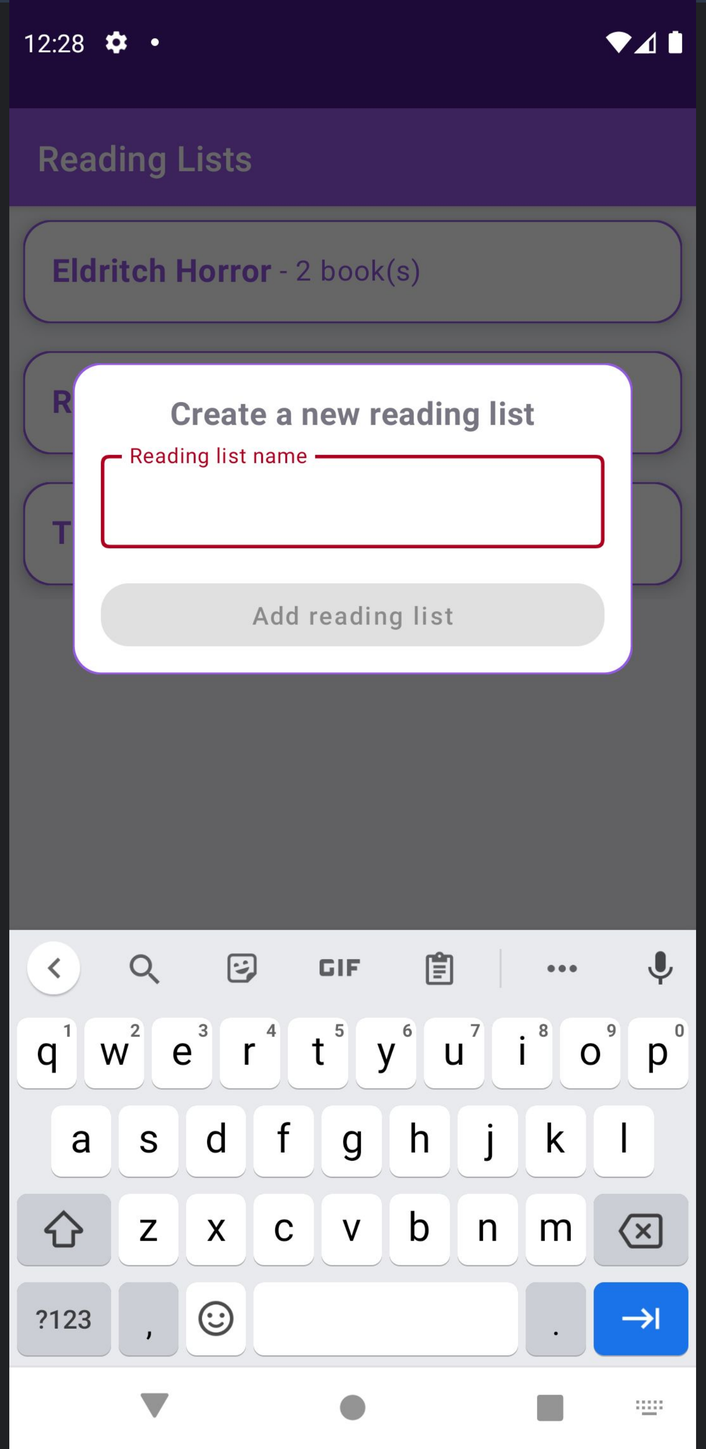
Typing


12:28
Reading Lists
Eldritch Horror - 2 book(s)
Create a new reading list
Reading list name
Add reading list

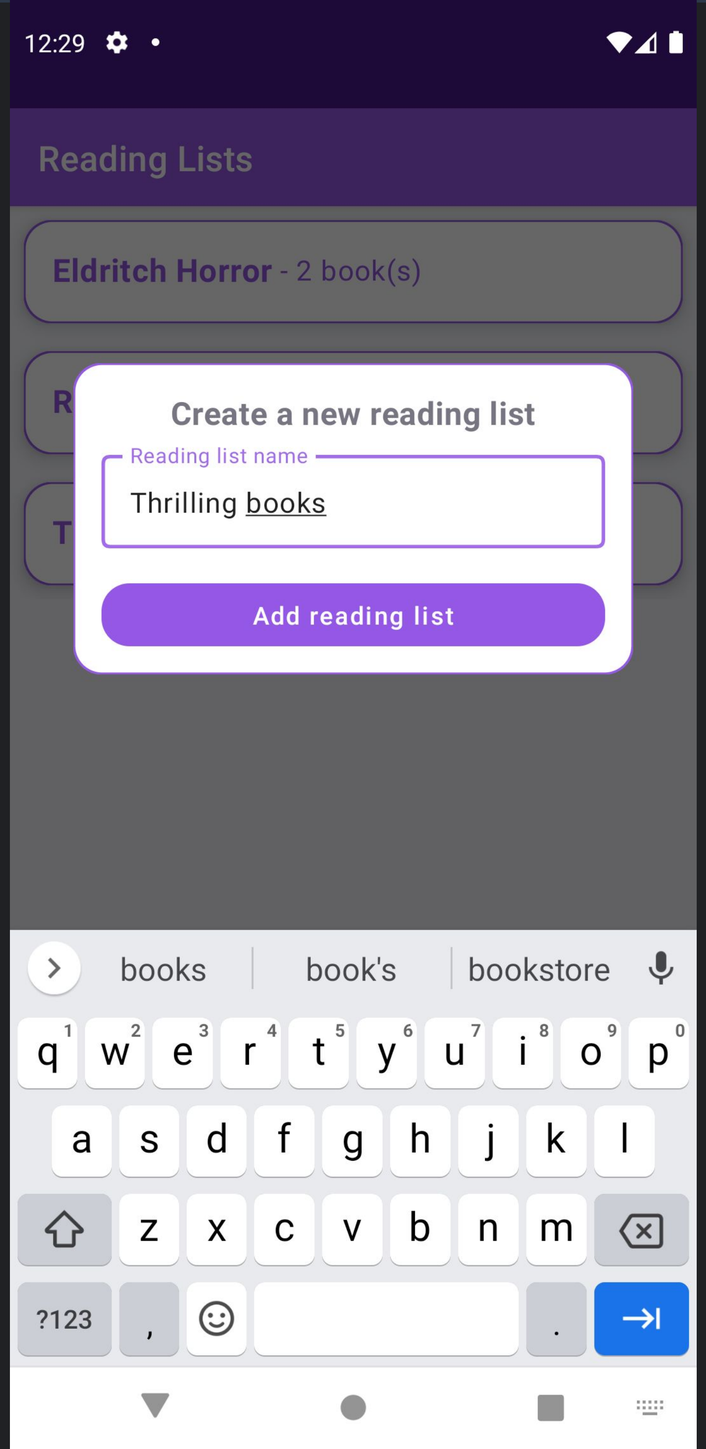
```
@Composable
fun AddReadingList(
  onAddList: (String) -> Unit,
  onDismiss: () -> Unit
) {
  val inputState = remember { mutableStateOf("") }
}
```

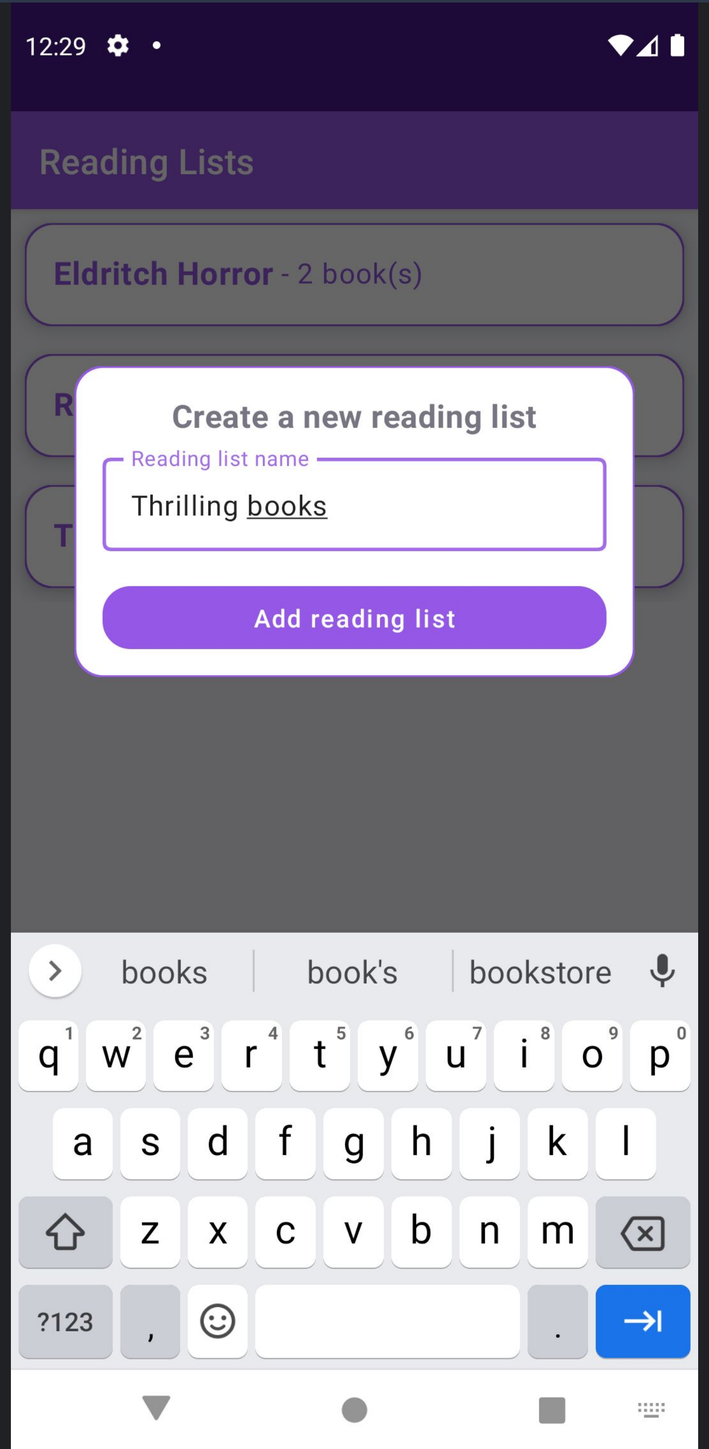
```
@Composable
fun AddReadingList(
  onAddList: (String) -> Unit,
  onDismiss: () -> Unit
) {
  val inputState = remember { mutableStateOf("") }

      ....
      InputField(
        value = inputState.value,
        isInputValid = inputState.value.isNotEmpty(),
        onStateChanged = { newValue -> inputState.value = newValue }
      )

      ActionButton(
        isEnabled = inputState.value.isNotEmpty(),
        onClick = { onAddList(inputState.value) }
      )
    }
}
```

```
@Composable
fun AddReadingList(
  onAddList: (String) -> Unit,
  onDismiss: () -> Unit
) {
  val inputState = remember { mutableStateOf("") }

      ....
      InputField(
        value = inputState.value,
        isInputValid = inputState.value.isNotEmpty(),
        onStateChanged = { newValue -> inputState.value = newValue } // state change
      )

      ActionButton(
        isEnabled = inputState.value.isNotEmpty(),
        onClick = { onAddList(inputState.value) }
      )
    }
}
```


12:28
Reading Lists
Eldritch Horror - 2 book(s)
Create a new reading list
Reading list name
Add reading list

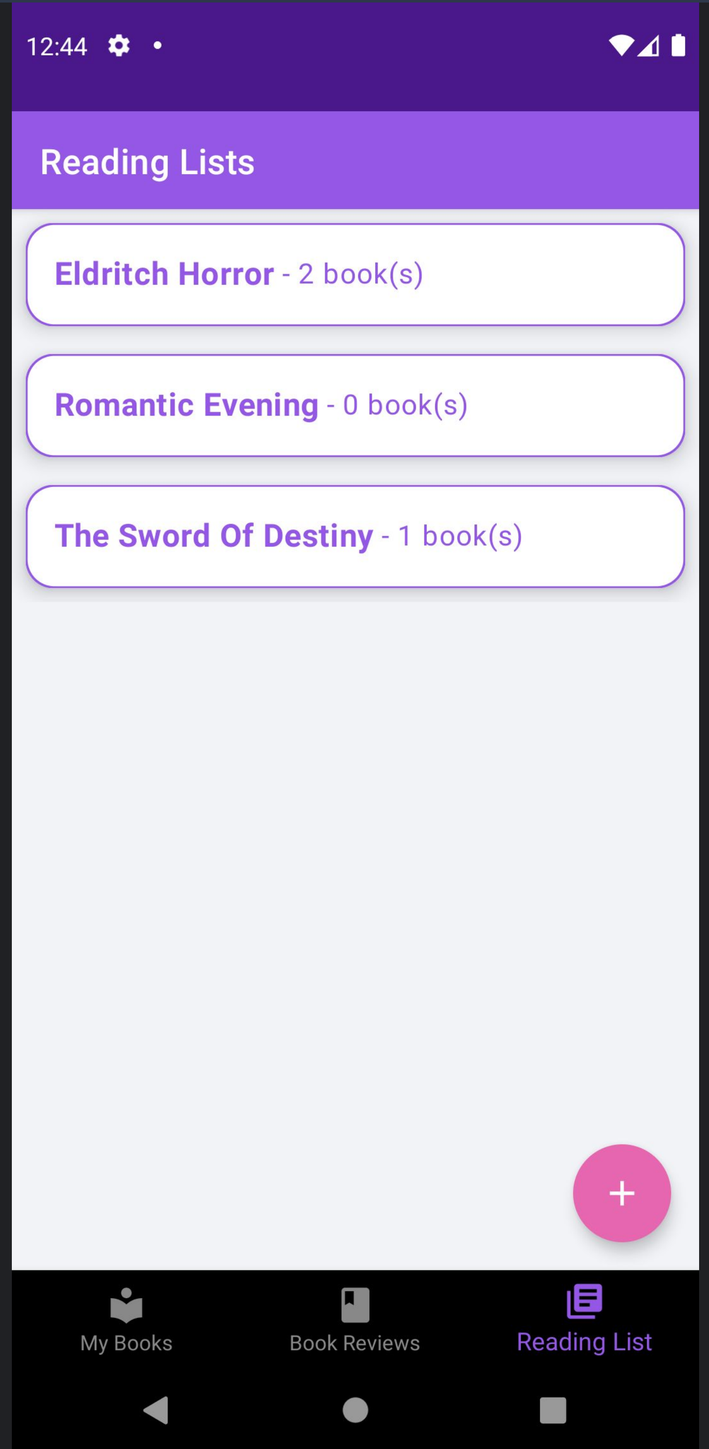
Typing

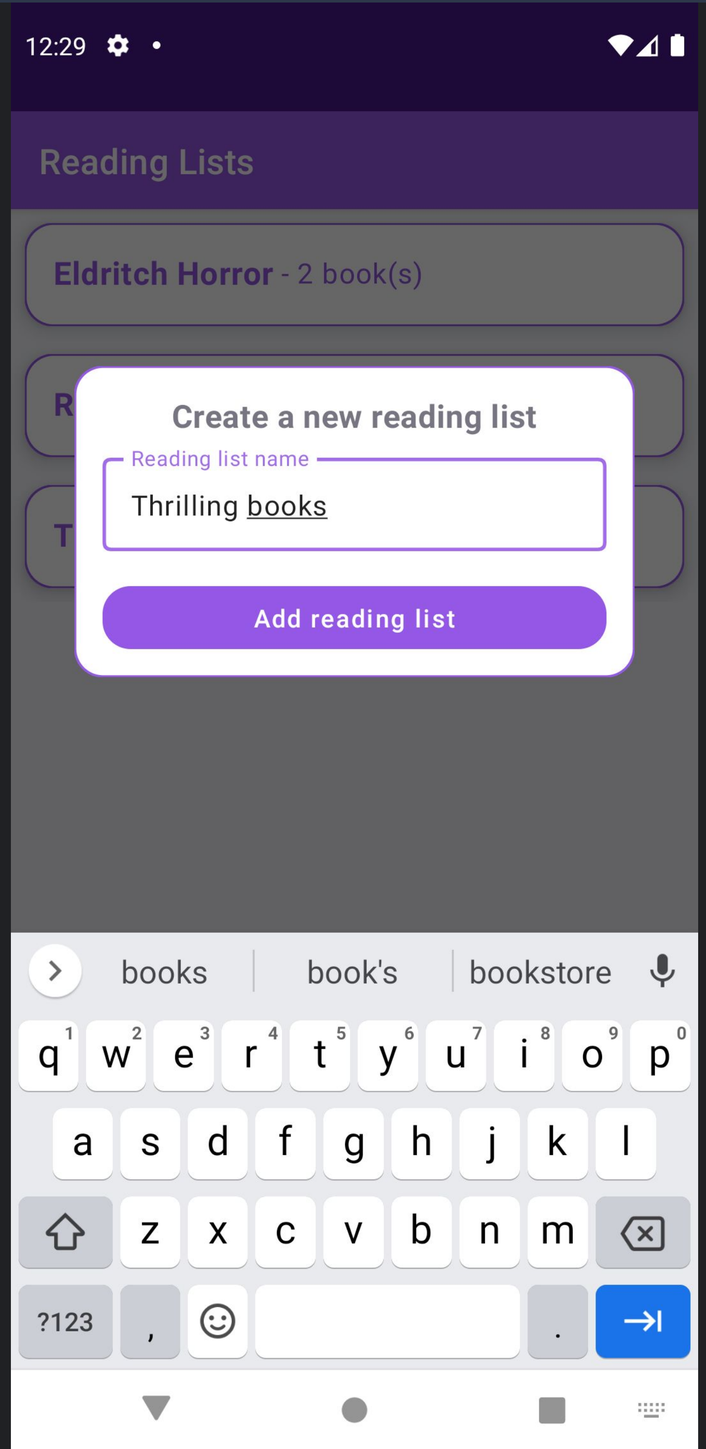

12:29
Reading Lists
Eldritch Horror - 2 book(s)
Create a new reading list
Reading list name
Thrilling books
Add reading list
books
book's
bookstore



Without observing the state properties

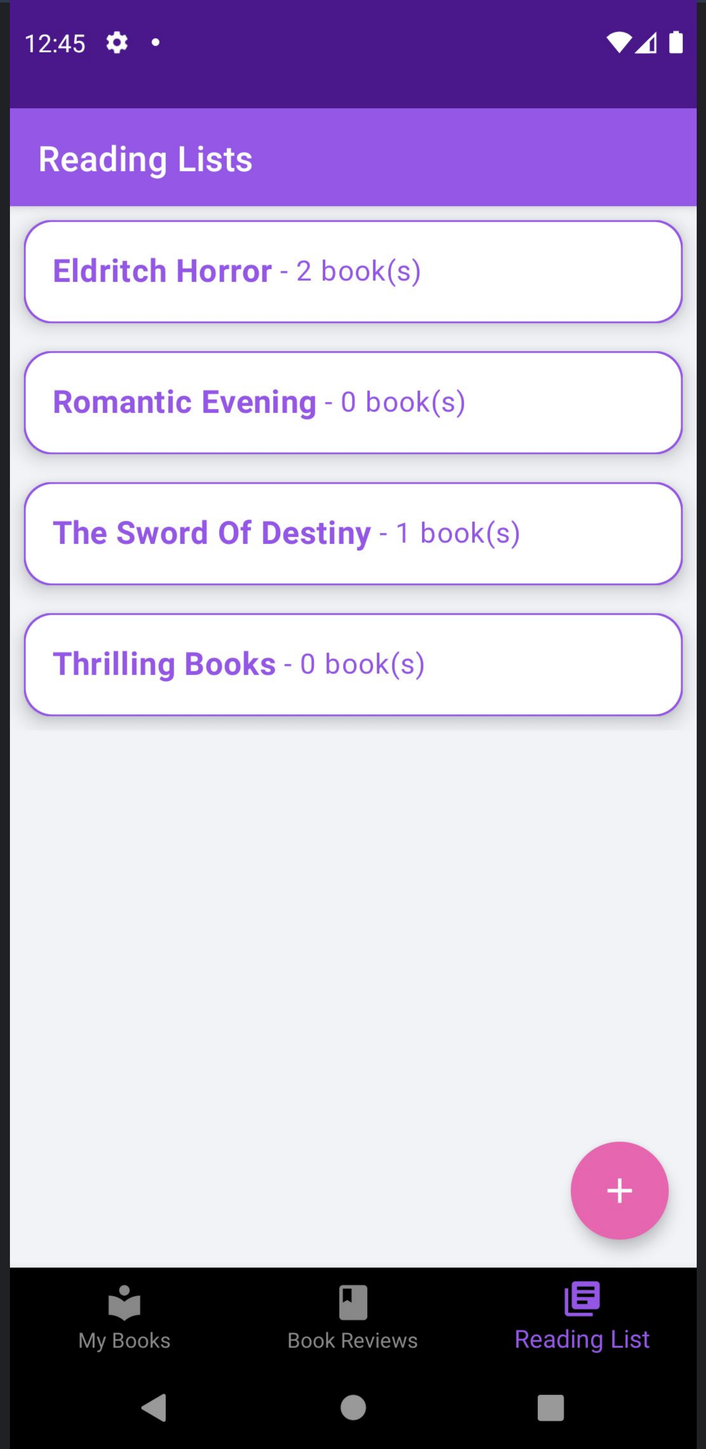
Add list





Using remember() for list items

Add list



What happens when state changes?

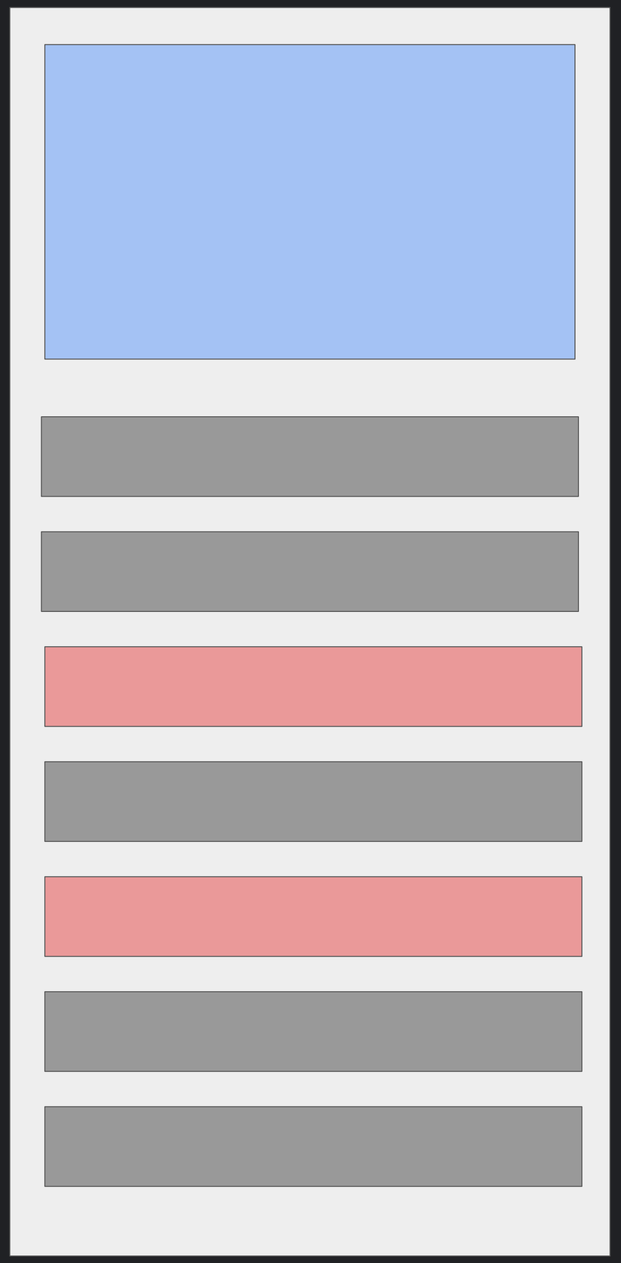
State Handling

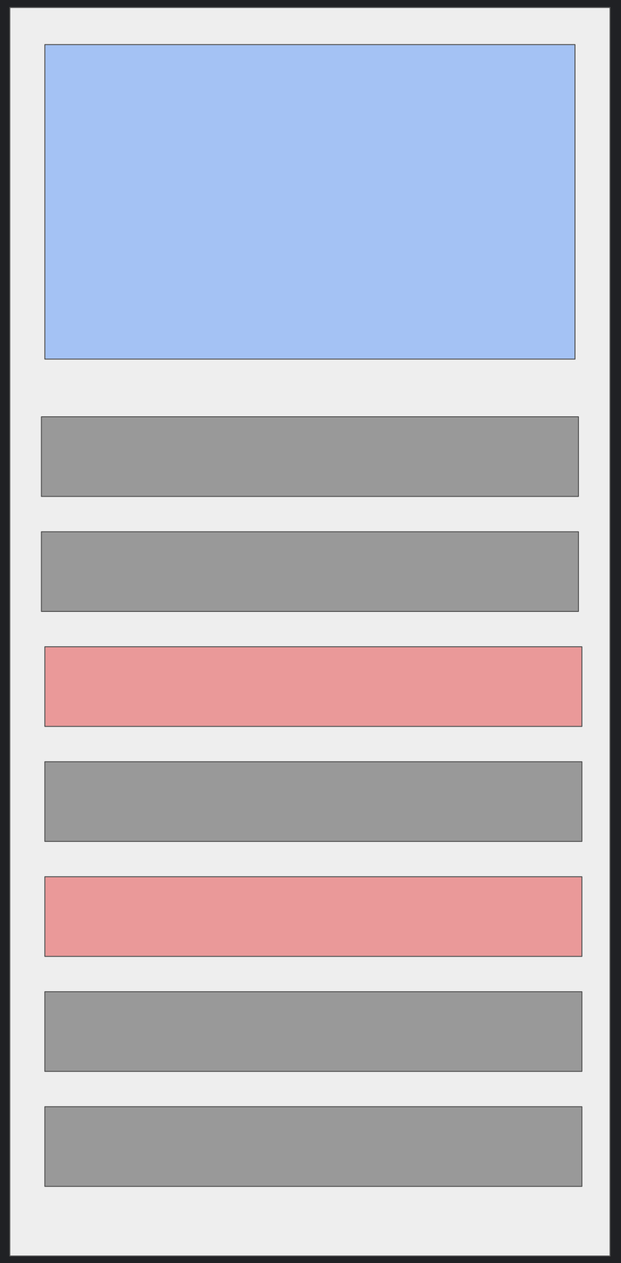
State tends to change

If the state changes, the UI re-draws itself in a smart way, applying minimal changes to match the new state.

This process is called **recomposition**.


Container state
Header state
List data state


Container - no changes
Header - no changes
List state - changed



Container - doesn’t re-draw

Header - doesn’t re-draw

List - recomposed

Compose is pretty big!

Here are the main items to *remember*

- **Build your UI based on state:** represent the state within the UI. If the state changes, the UI reflects that.
- **Start small:** You can slowly integrate Compose in your XML powered apps. When building components, deconstruct them into smaller parts.

Resources

Everything you need to check out to master Compose

- GitHub repo example
- Jetpack Compose course
- Ping me if you have any questions!

Questions?

Google Developers

GDG

Thank You!