

Introduction to Jetpack Compose

Mohit Sarveiya

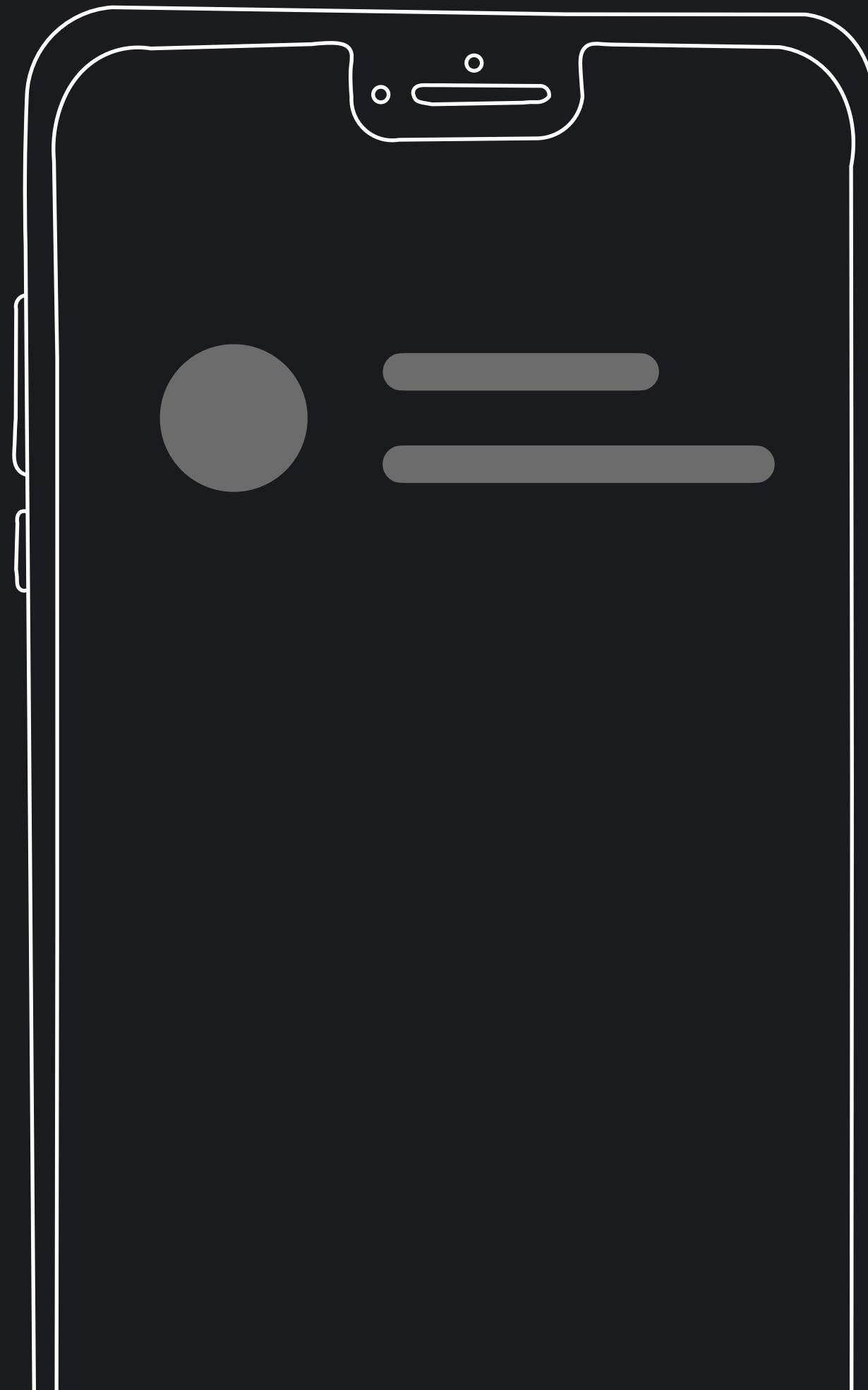
 @heyitsmohit

Introduction to Jetpack Compose

- Thinking in Compose
- Layouts
- Managing State
- Side Effects

Thinking in Compose

Imperative



Imperative

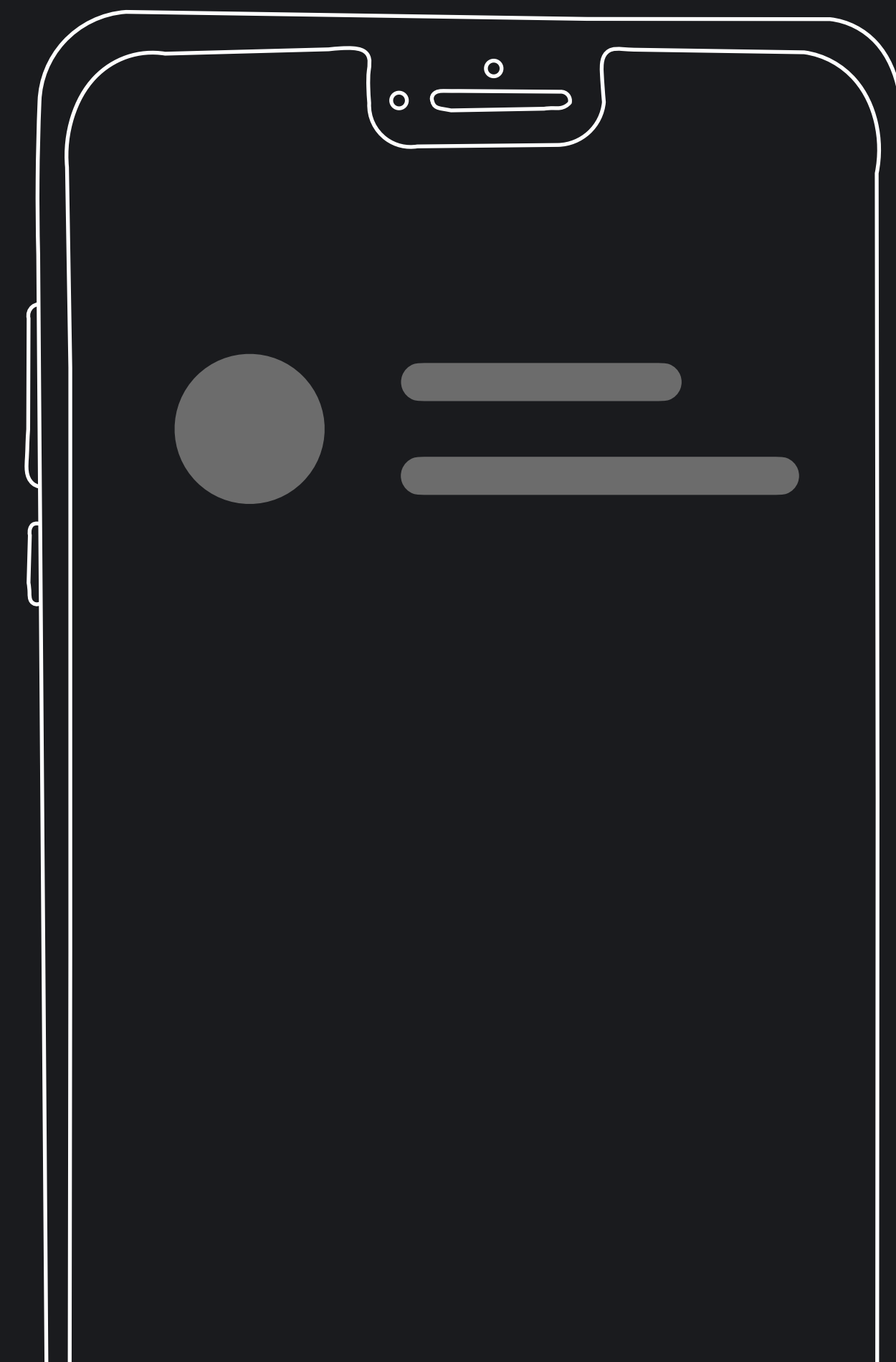
```
<ConstraintLayout>
```

```
    <ImageView />
```

```
    <TextView />
```

```
    <TextView />
```

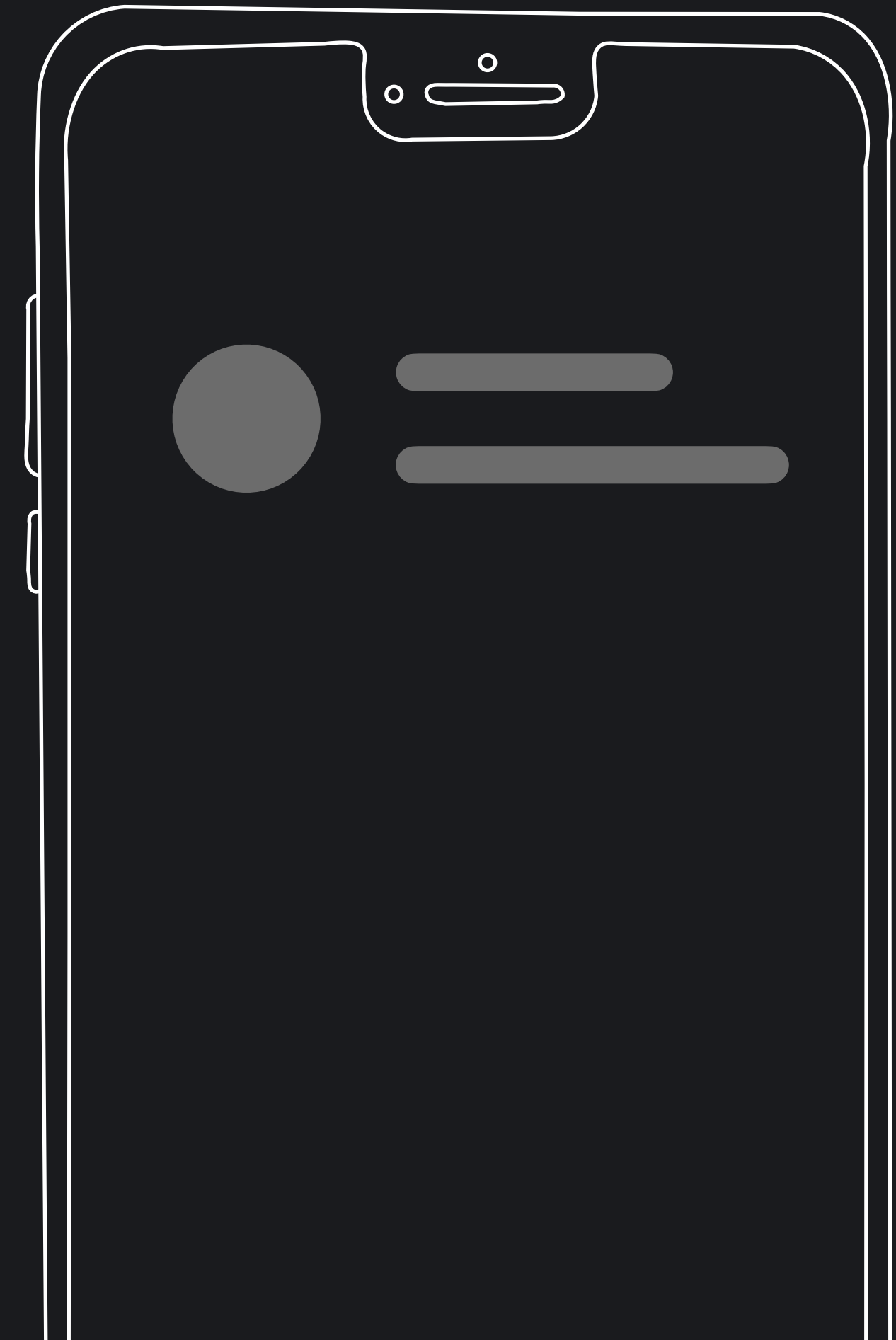
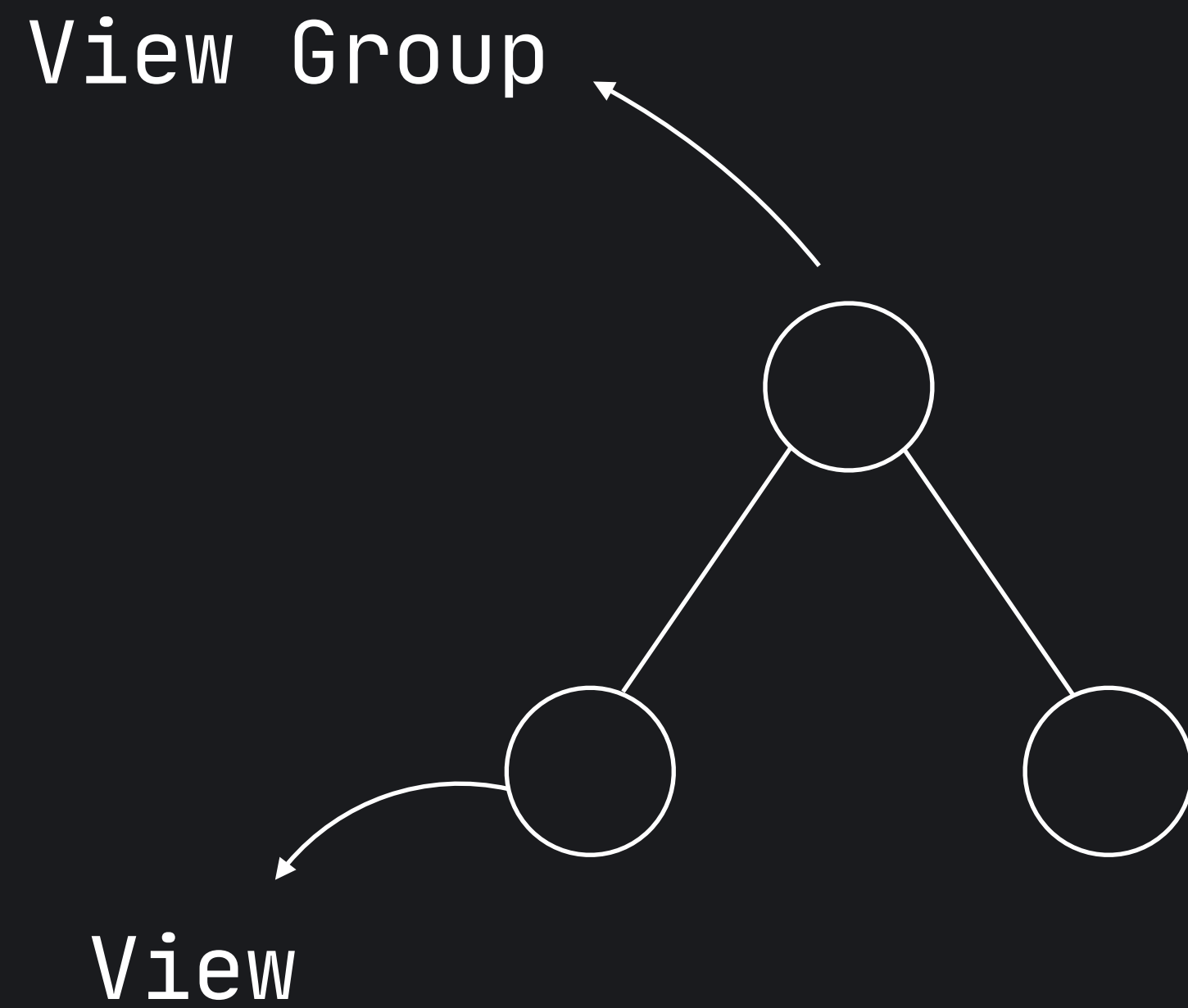
```
</ConstraintLayout>
```



Imperative

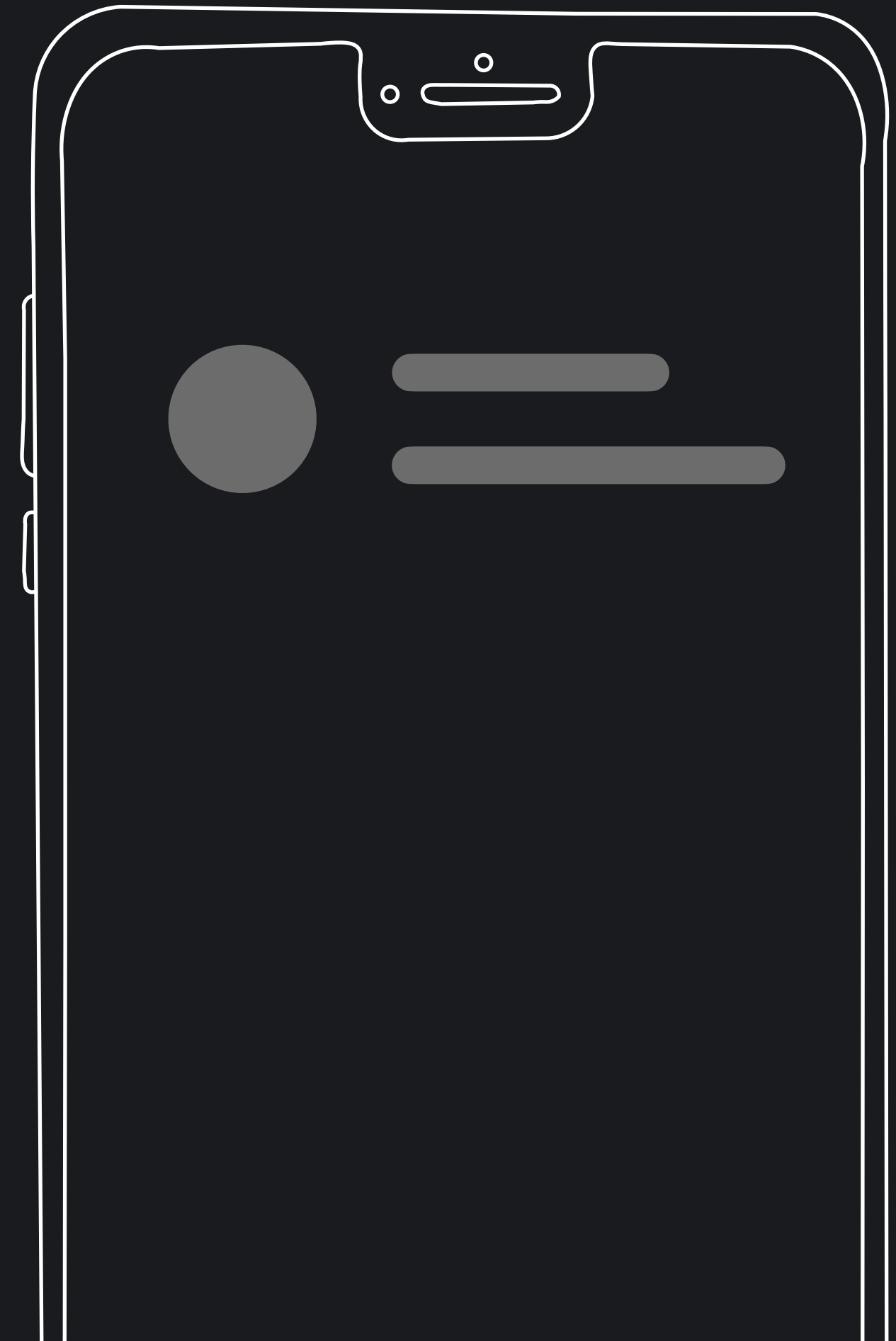
View Group

View



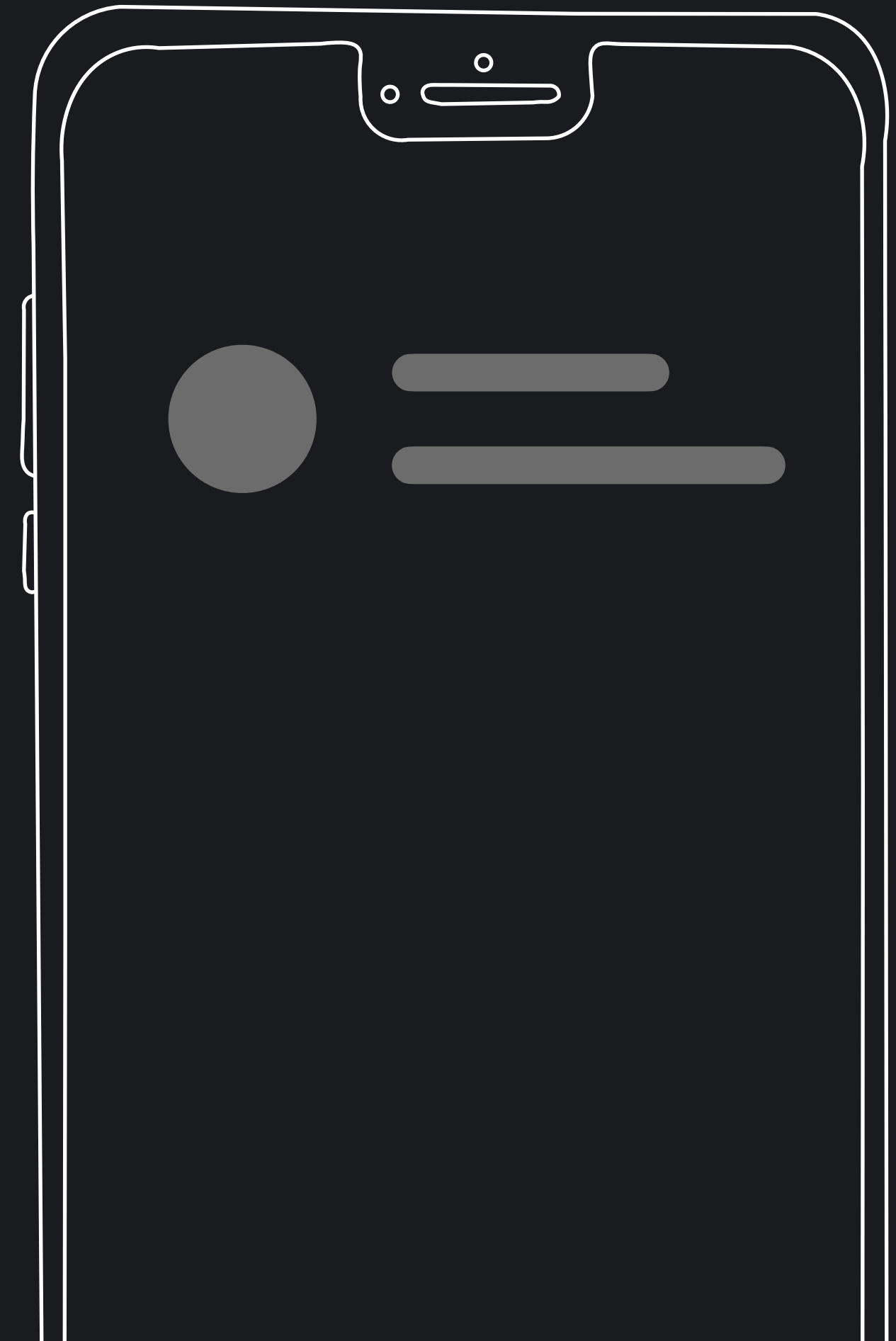
Imperative

```
val textView = findViewById(R.id.textView)
```



Imperative

```
val textView = findViewById(R.id.textView)  
textView.text = ...
```



Declarative UI

- Widgets are stateless
- No getters or setters to update state

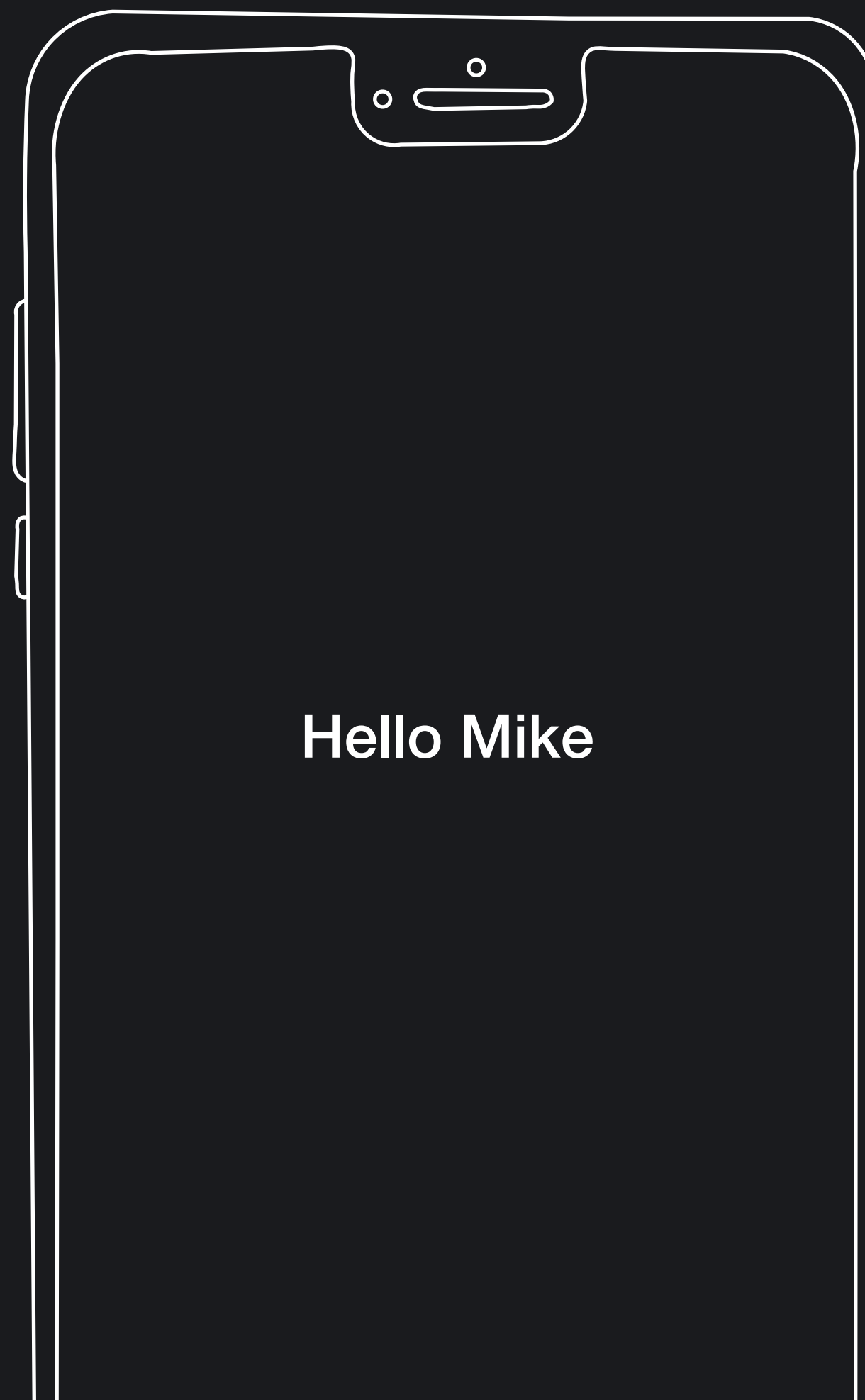
What is Jetpack Compose UI?

- UI toolkit
- Declarative UI framework


Benefits

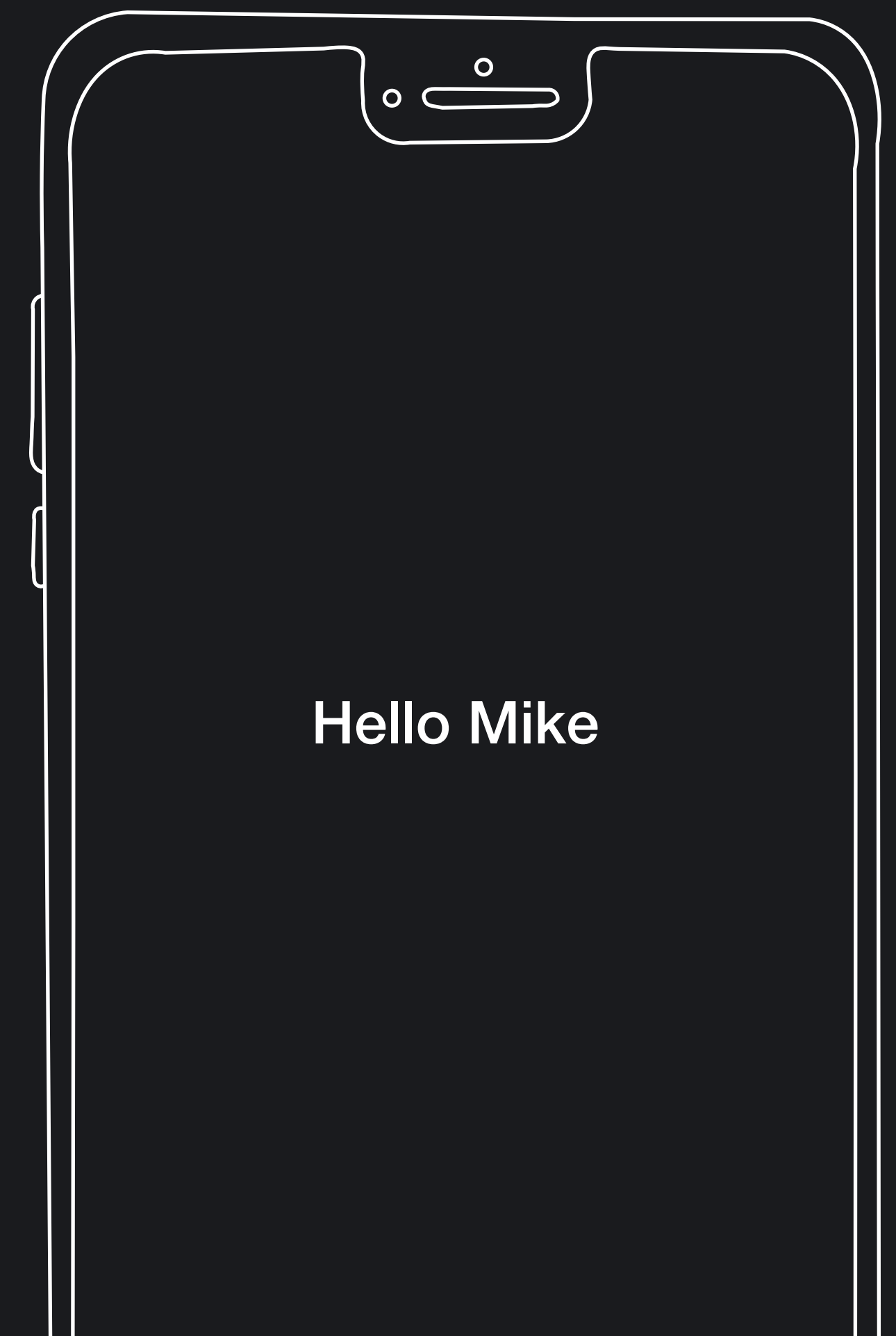
- Faster development
- Build UIs with less code

Declarative UI



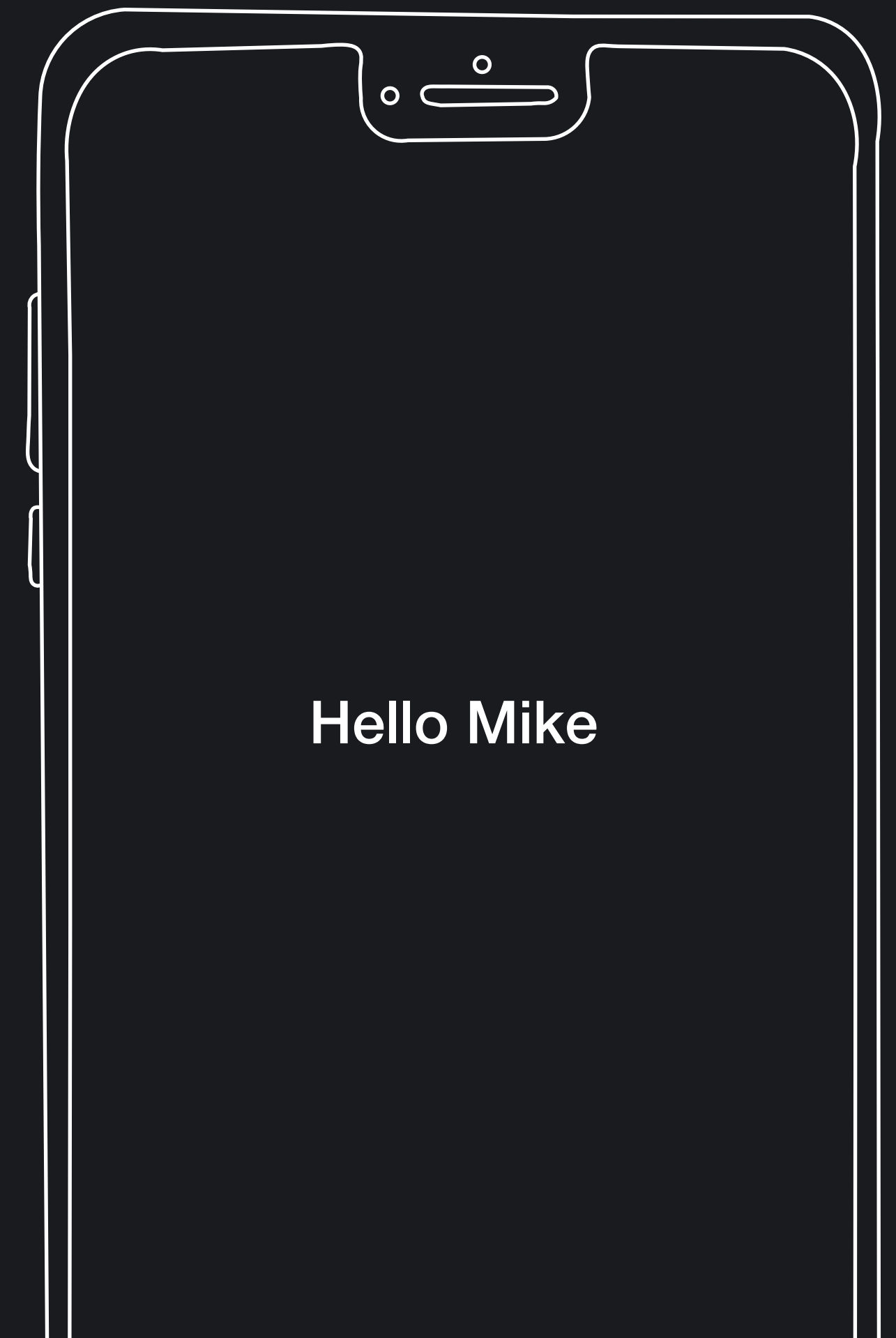
Declarative UI

```
@Composable  
fun GreetingView(name: String) {  
      
}
```



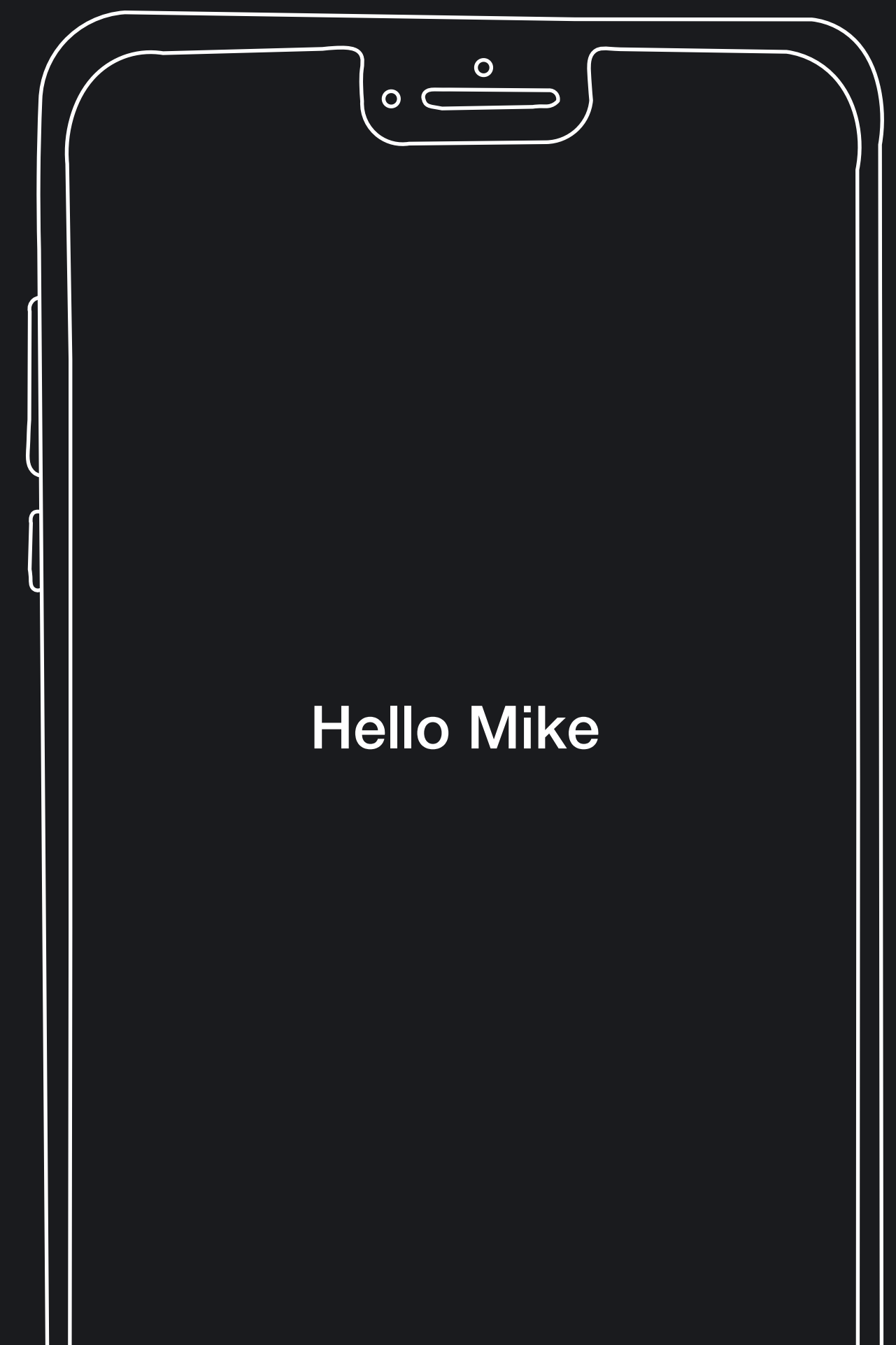
Declarative UI

```
@Composable
fun GreetingView(name: String) {
    Text(text = "Hello $name")
}
```

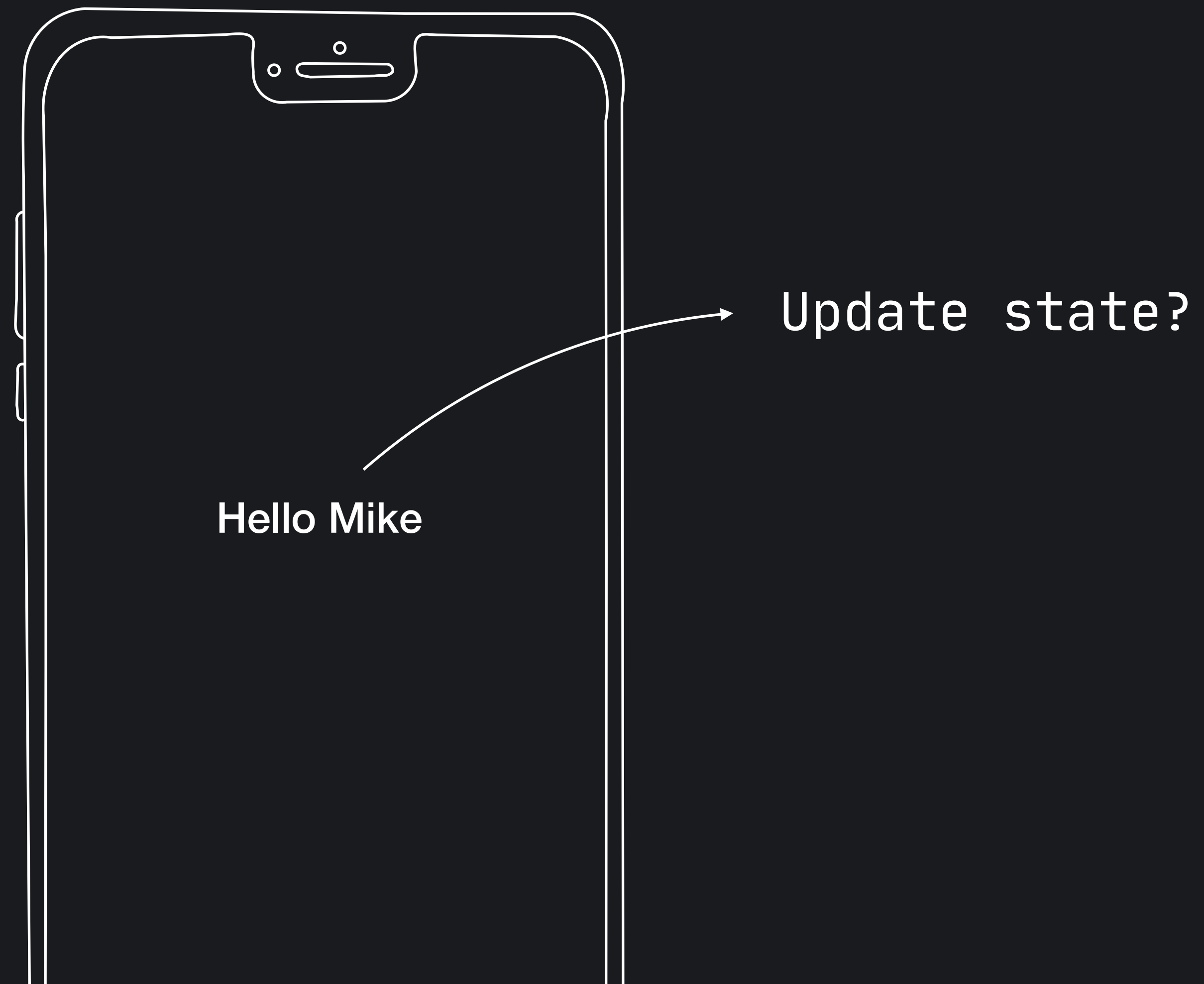


Declarative UI

```
class MainActivity {  
    setContent {  
        GreetingView(name = "Mike")  
    }  
}
```



Declarative UI



Declarative UI

```
GreetingView(name = "Bill")
```

A white outline of a smartphone screen. Inside the screen, the text "Hello Bill" is displayed in white. The phone has a notch at the top and a home button at the bottom.

Hello Bill

Recomposition



Recomposition



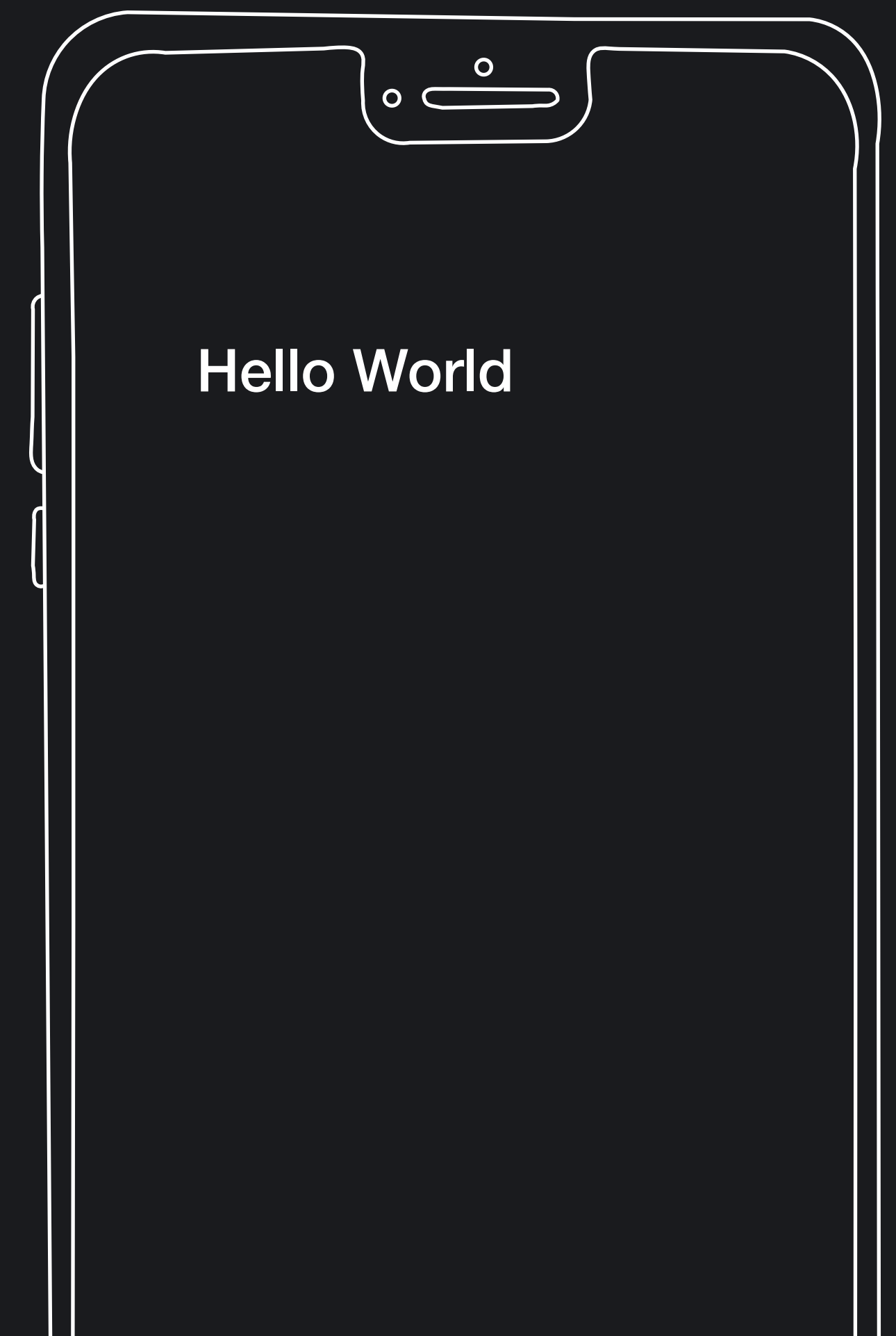
Recomposition



Modifiers

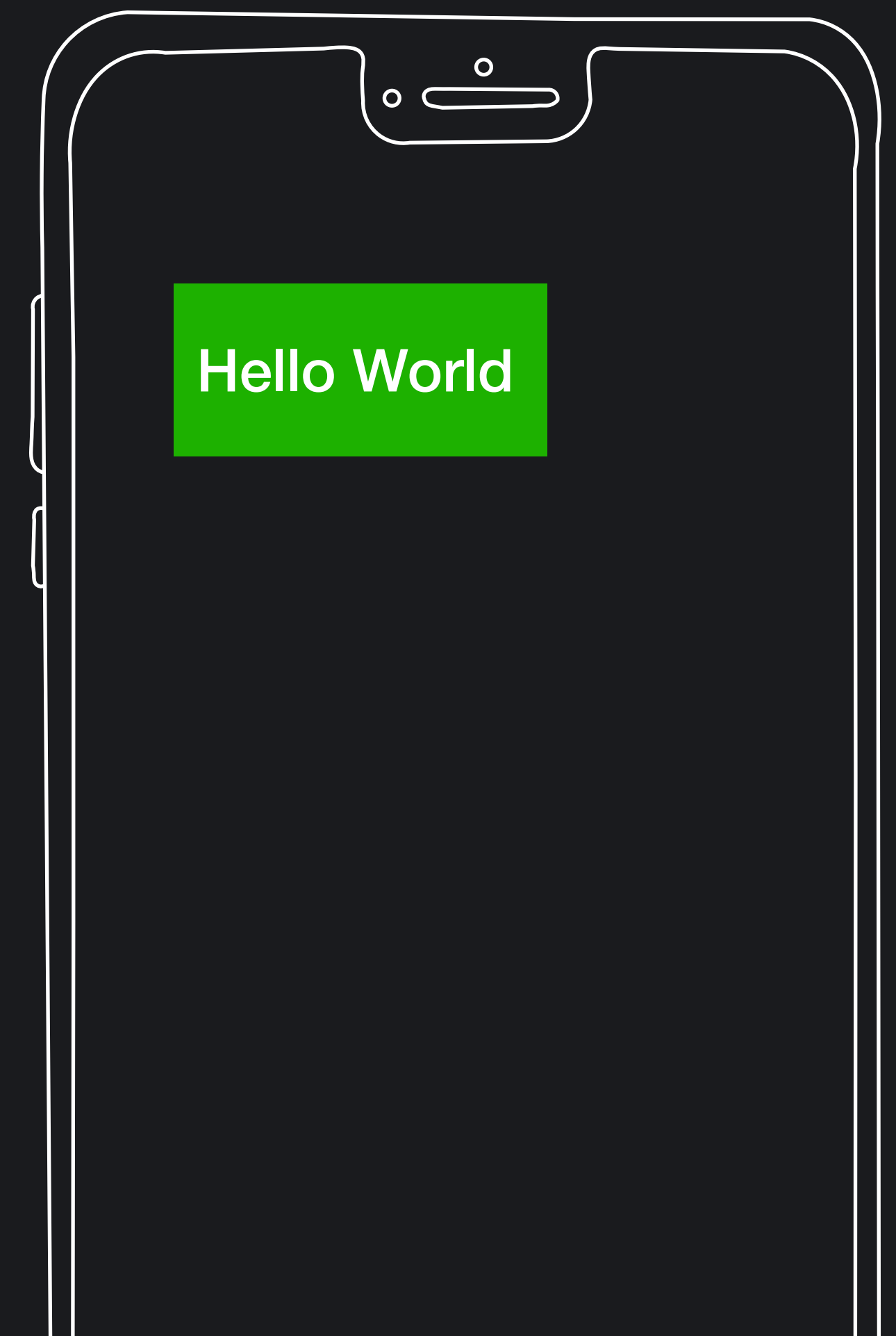
Modifiers

```
Text(  
    modifier = Modifier  
        .padding(10.dp),  
    text = "Hello World"  
)
```



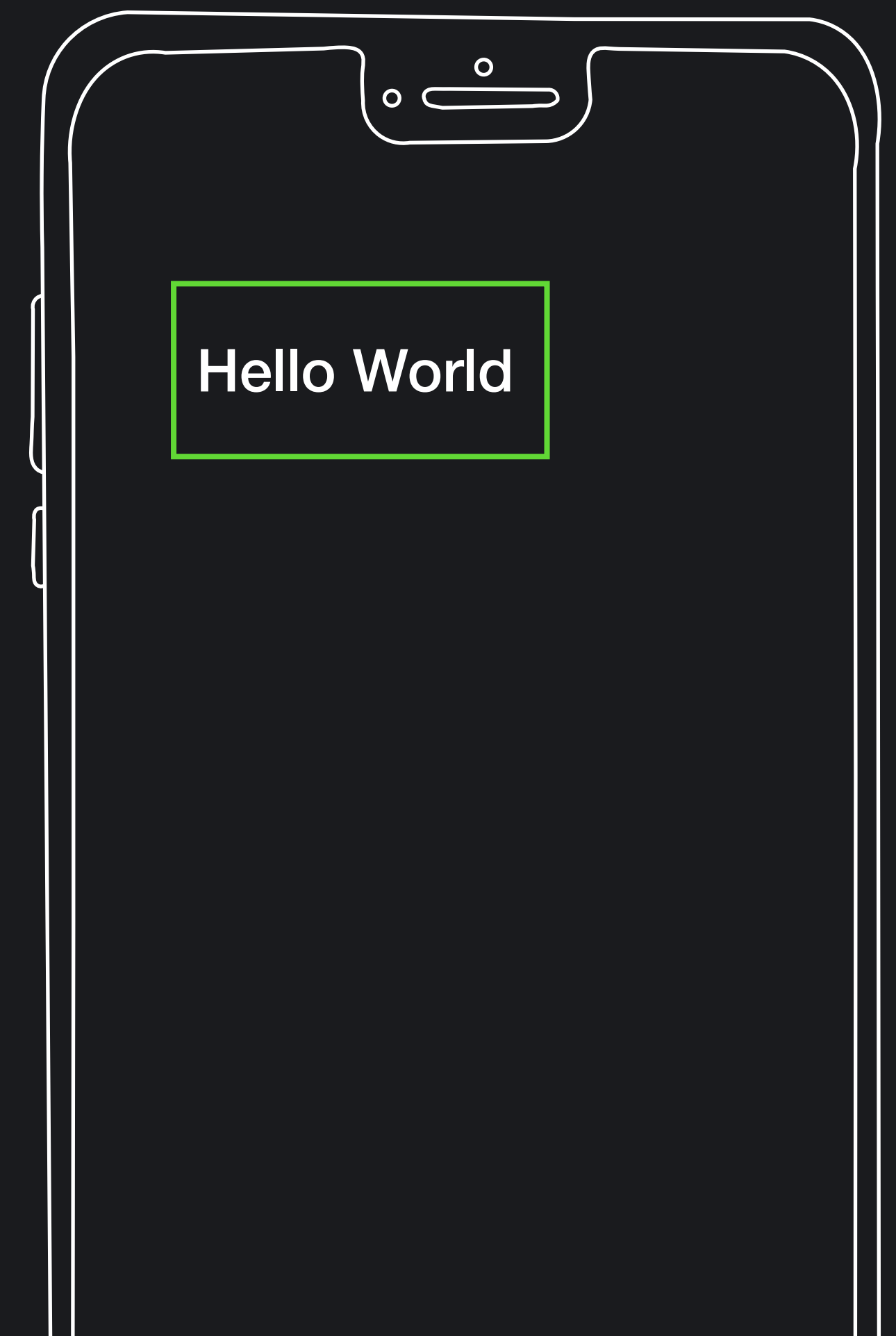
Modifiers

```
Text(  
    modifier = Modifier  
        .background(Color.Green),  
    text = "Hello World"  
)
```



Modifiers

```
Text(  
    modifier = Modifier  
        .border(width = 2.dp, Color.Green),  
    text = "Hello World"  
)
```

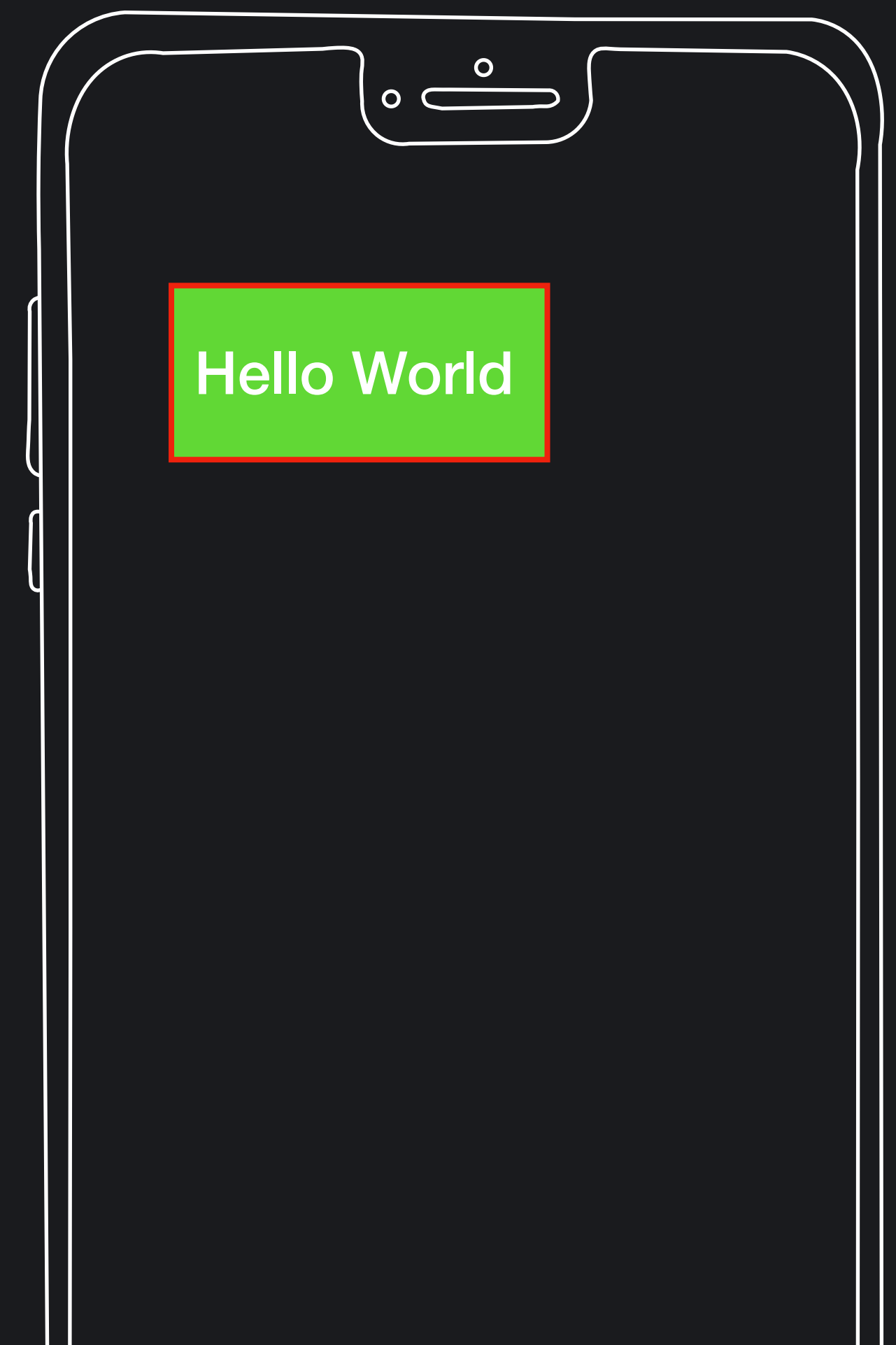


Modifiers

- Order matters

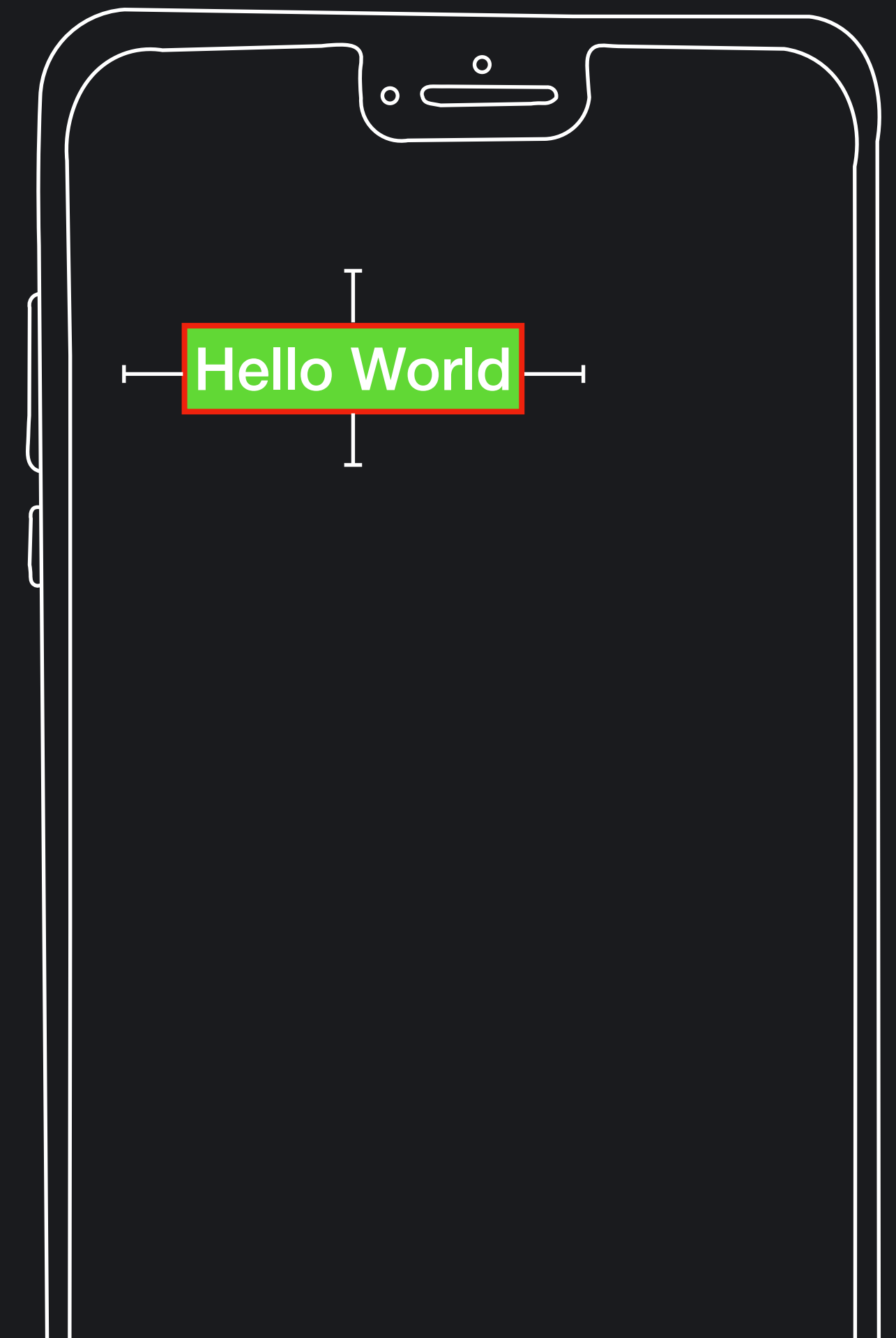
Modifiers

```
Text(  
    modifier = Modifier  
        .border(width = 2.dp, Color.Red)  
        .background(Color.Green)  
        .padding(12.dp),  
    text = "Hello World"  
)
```



Modifiers

```
Text(  
    modifier = Modifier  
        .padding(12.dp)  
        .border(width = 2.dp, Color.Red)  
        .background(Color.Green),  
    text = "Hello World"  
)
```



List of Compose modifiers

Actions

Scope: Any

```
Modifier.clickable(  
    enabled: Boolean,  
    onClickLabel: String?,  
    role: Role?,  
    onClick: () -> Unit  
)
```

Configure component to receive clicks via input or accessibility "click" event.

Scope: Any

```
Modifier.clickable(  
    interactionSource: MutableInteractionSource,  
    indication: Indication?,  
    enabled: Boolean,  
    onClickLabel: String?,  
    role: Role?
```

On this page

[Actions](#)[Alignment](#)[Animation](#)[Border](#)[Drawing](#)[Focus](#)[Graphics](#)[Keyboard](#)[Layout](#)[Padding](#)[Pointer](#)[Position](#)[Semantics](#)[Scroll](#)[Size](#)[Testing](#)

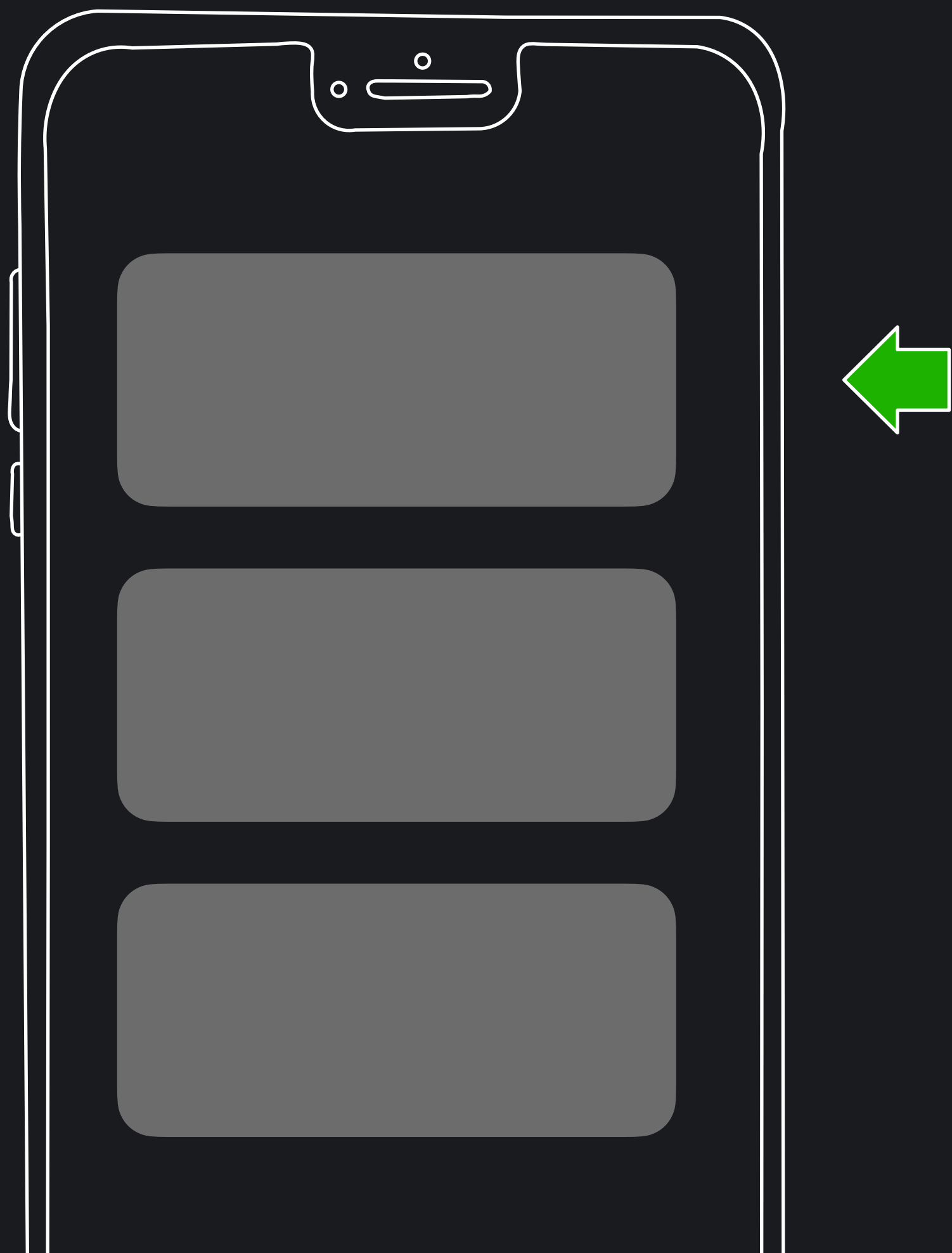
✦ Recommendations

[Get started with Jetpack Compose](#)

Updated Mar 9, 2022

Layouts

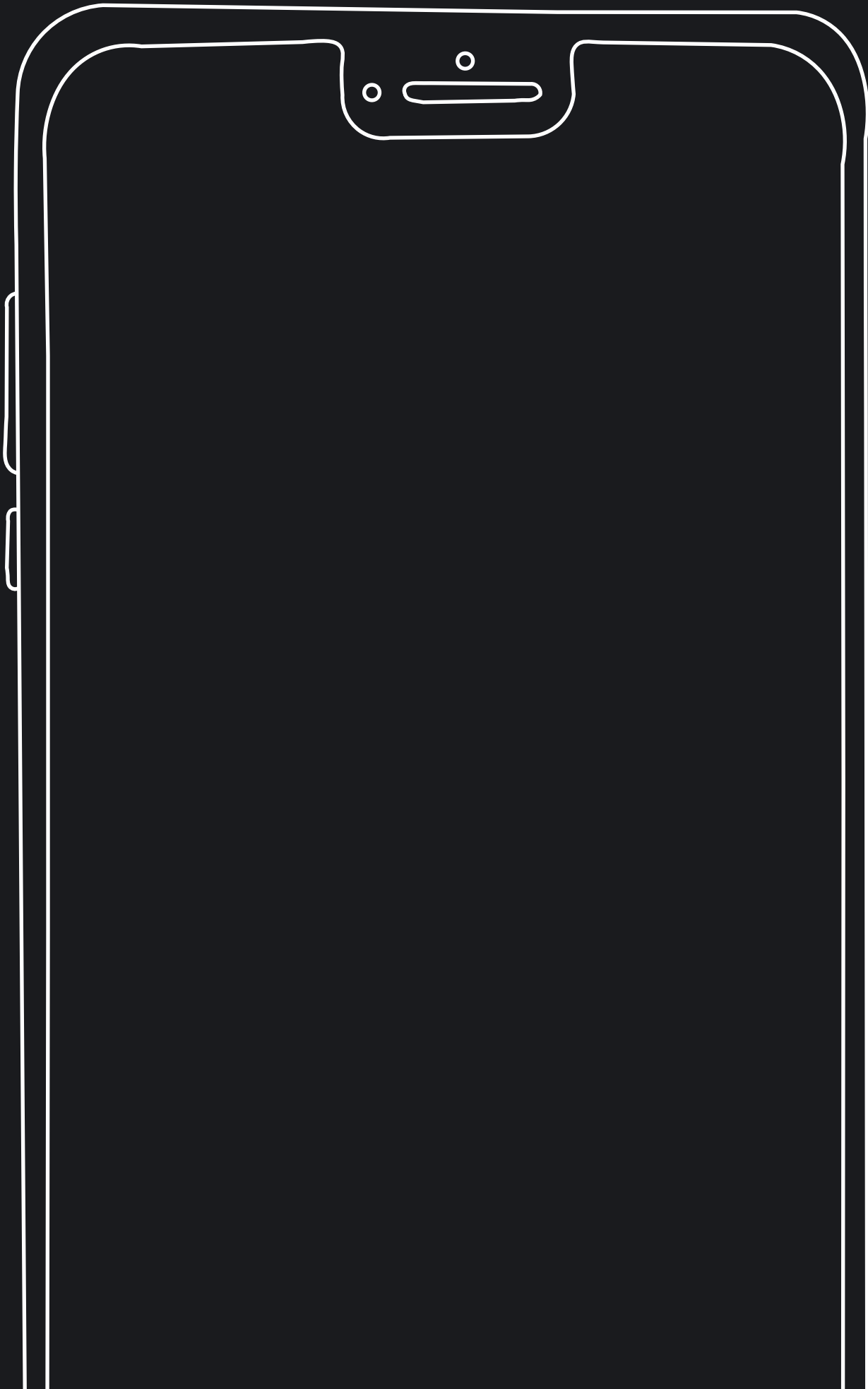
Column



Column

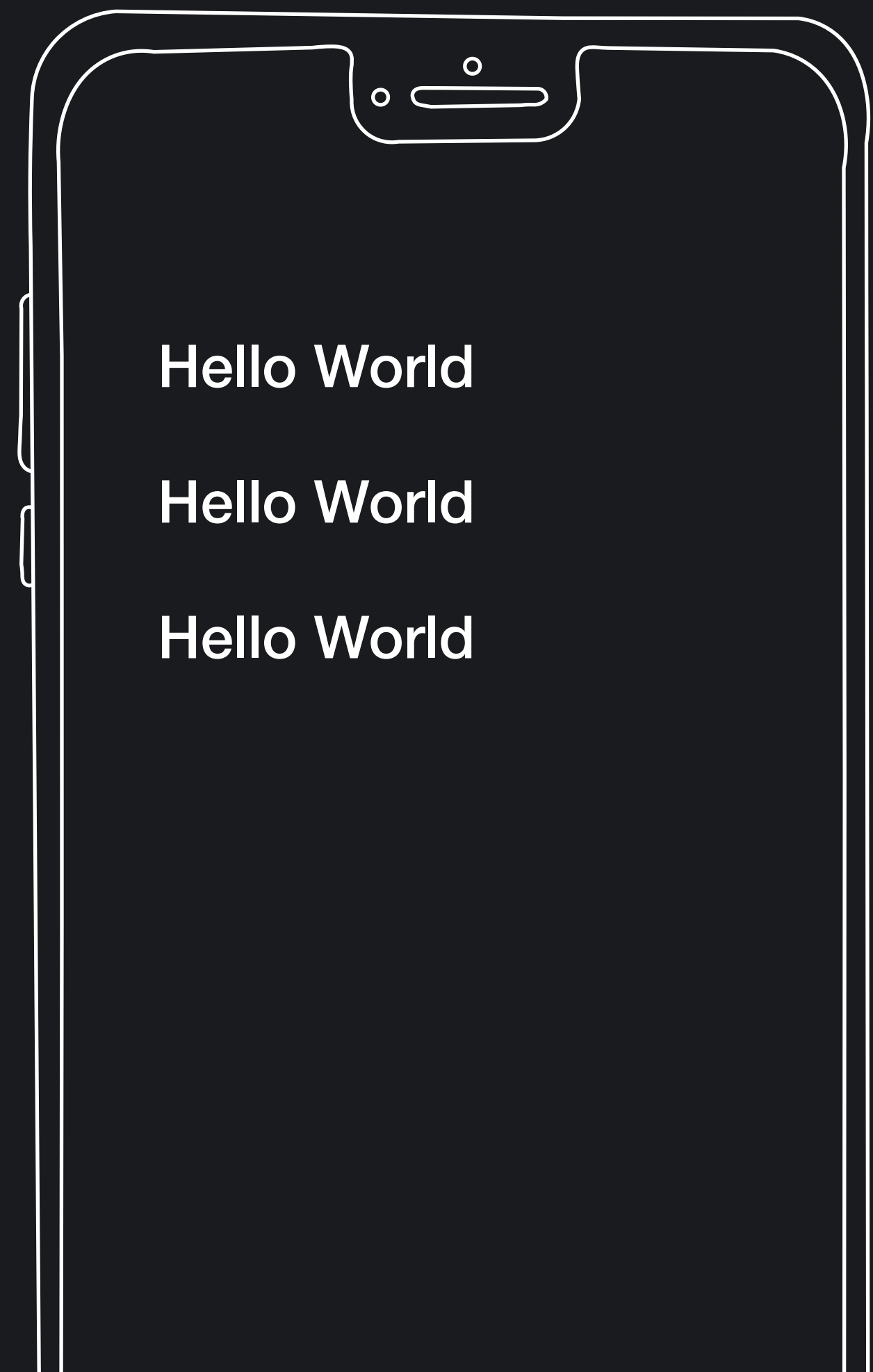
➡ *Column* {

}



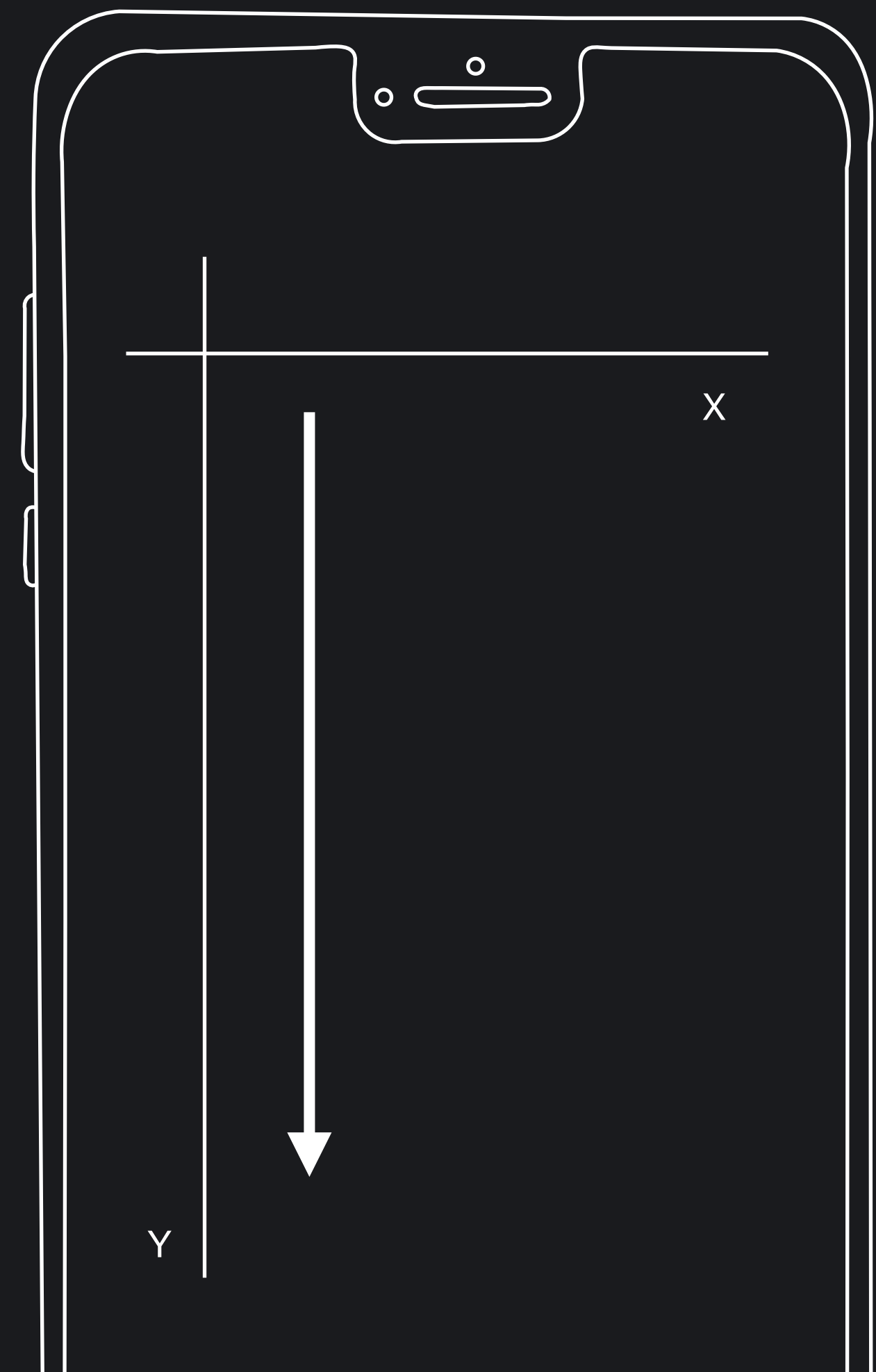
Column

```
Column {  
    Text("Hello World")  
    ➔ Text("Hello World")  
    Text("Hello World")  
}
```



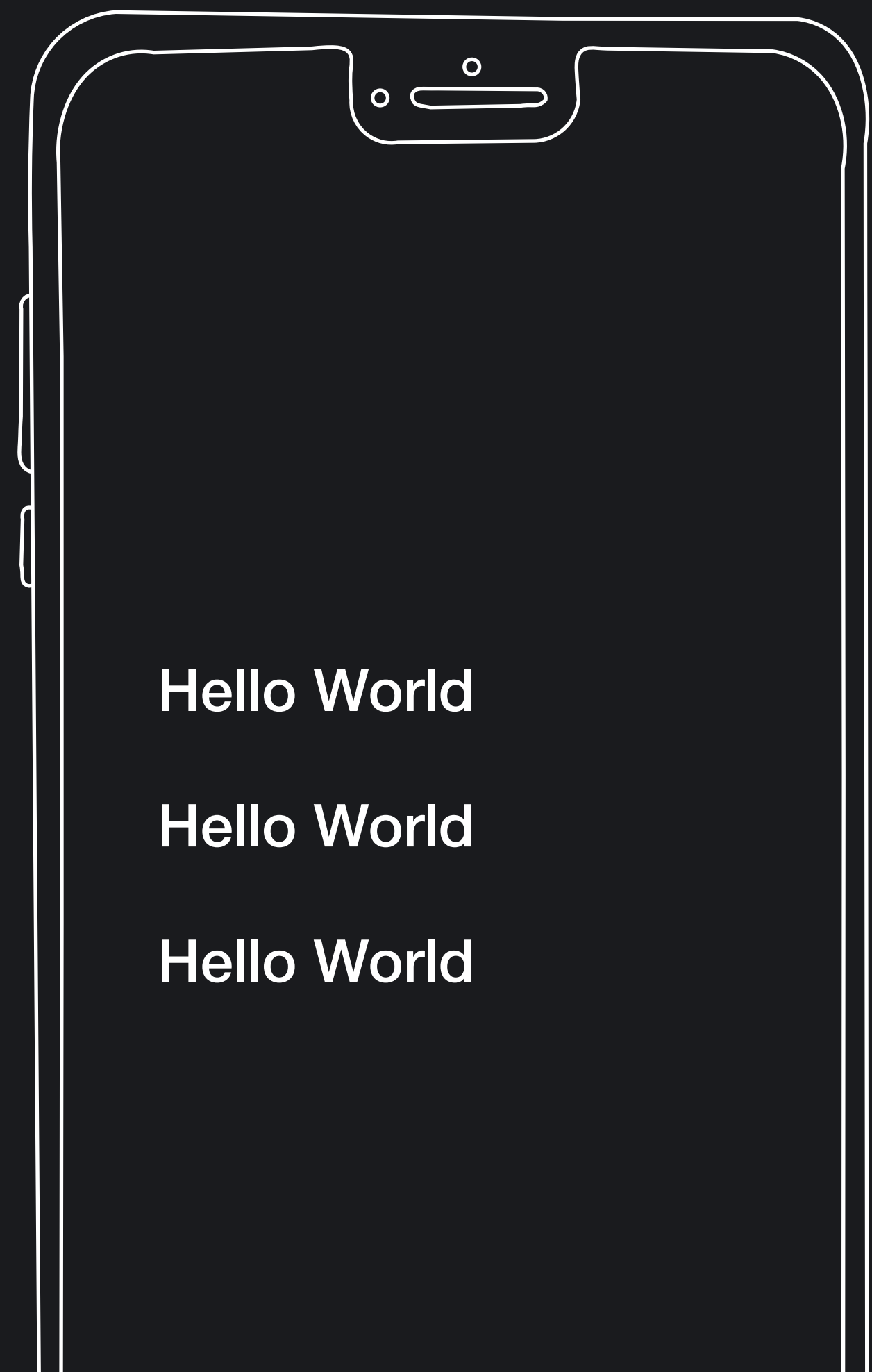
Column

```
Column(  
    verticalArrangement =  
) {  
    Text("Hello World")  
    Text("Hello World")  
    Text("Hello World")  
}
```



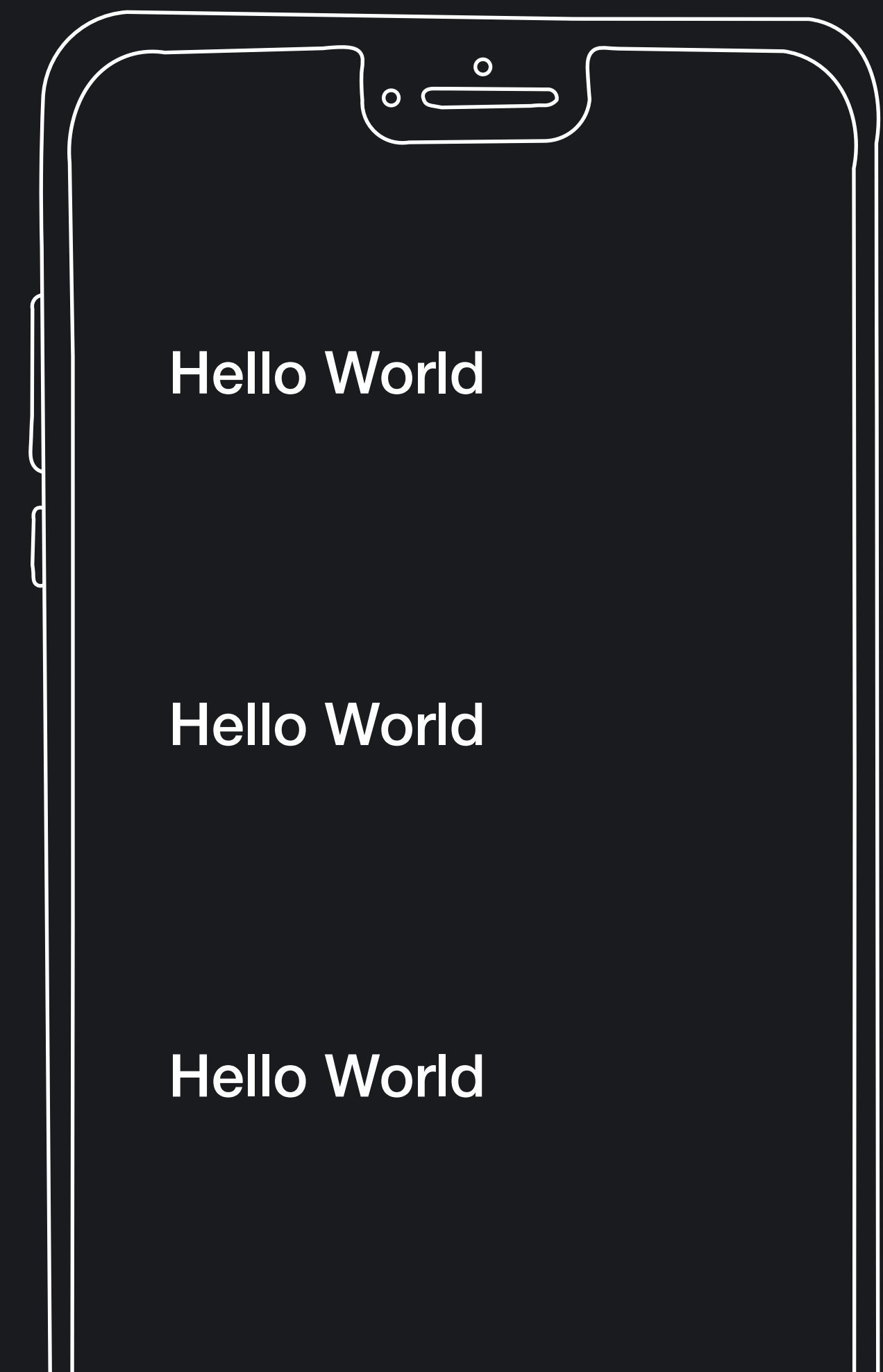
Column

```
Column(  
➔   verticalArrangement = Arrangement.Center  
) {  
    Text("Hello World")  
    Text("Hello World")  
    Text("Hello World")  
}
```



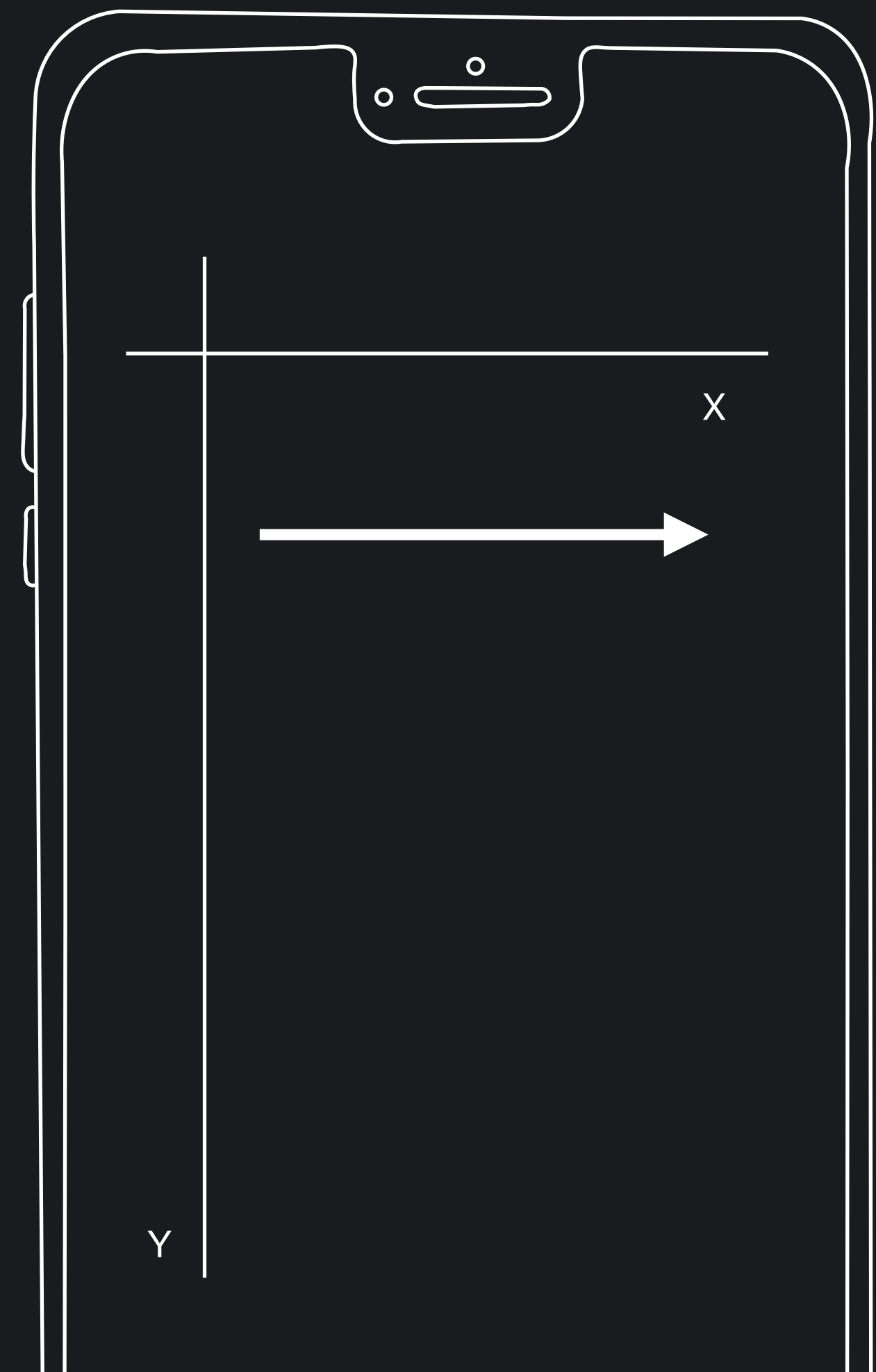
Column

```
Column(  
➡ Arrangement.SpaceBetween  
) {  
    Text("Hello World")  
    Text("Hello World")  
    Text("Hello World")  
}
```



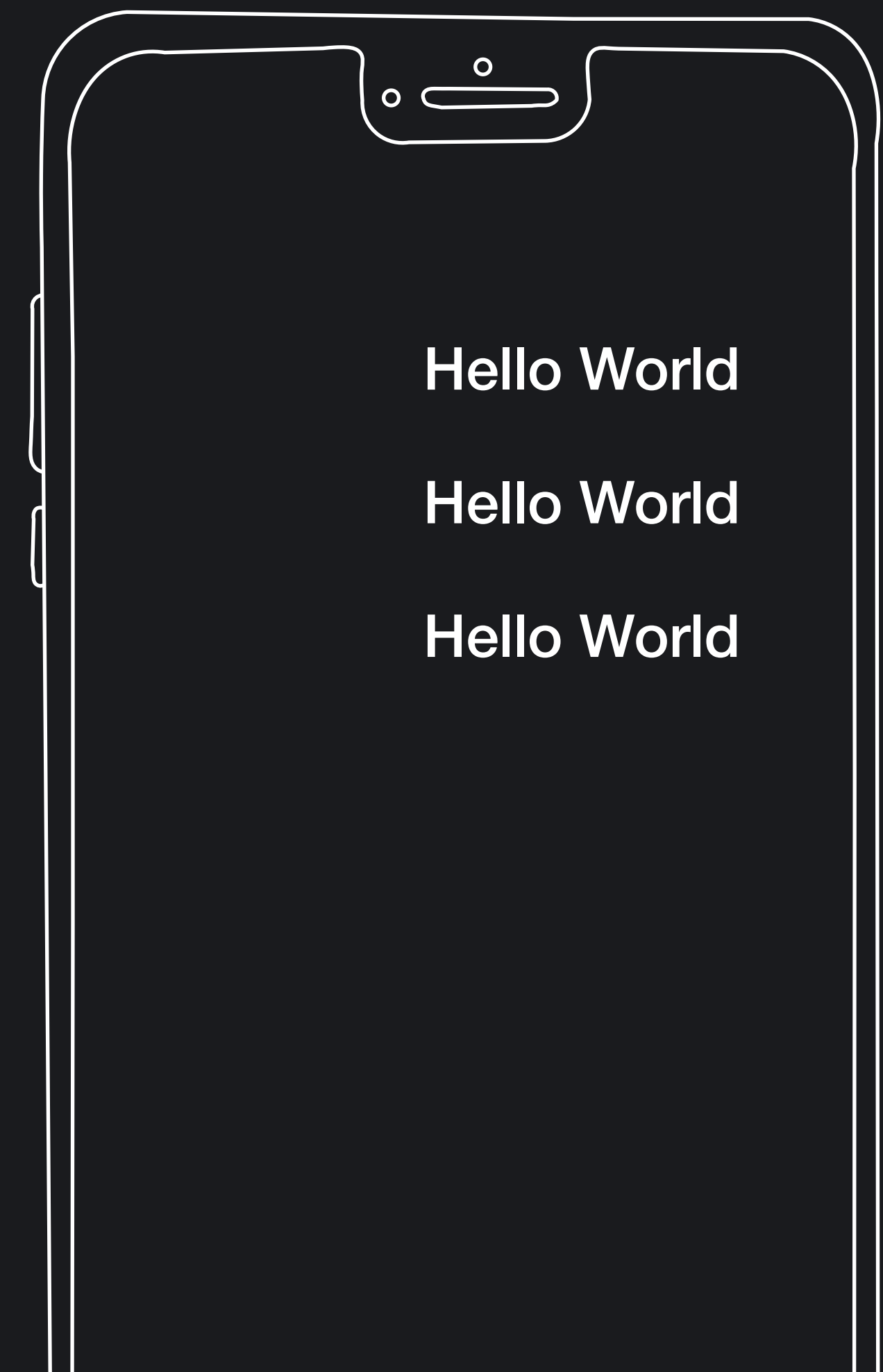
Column

```
Column(  
    horizontalAlignment =  
) {  
    Text("Hello World")  
    Text("Hello World")  
    Text("Hello World")  
}
```



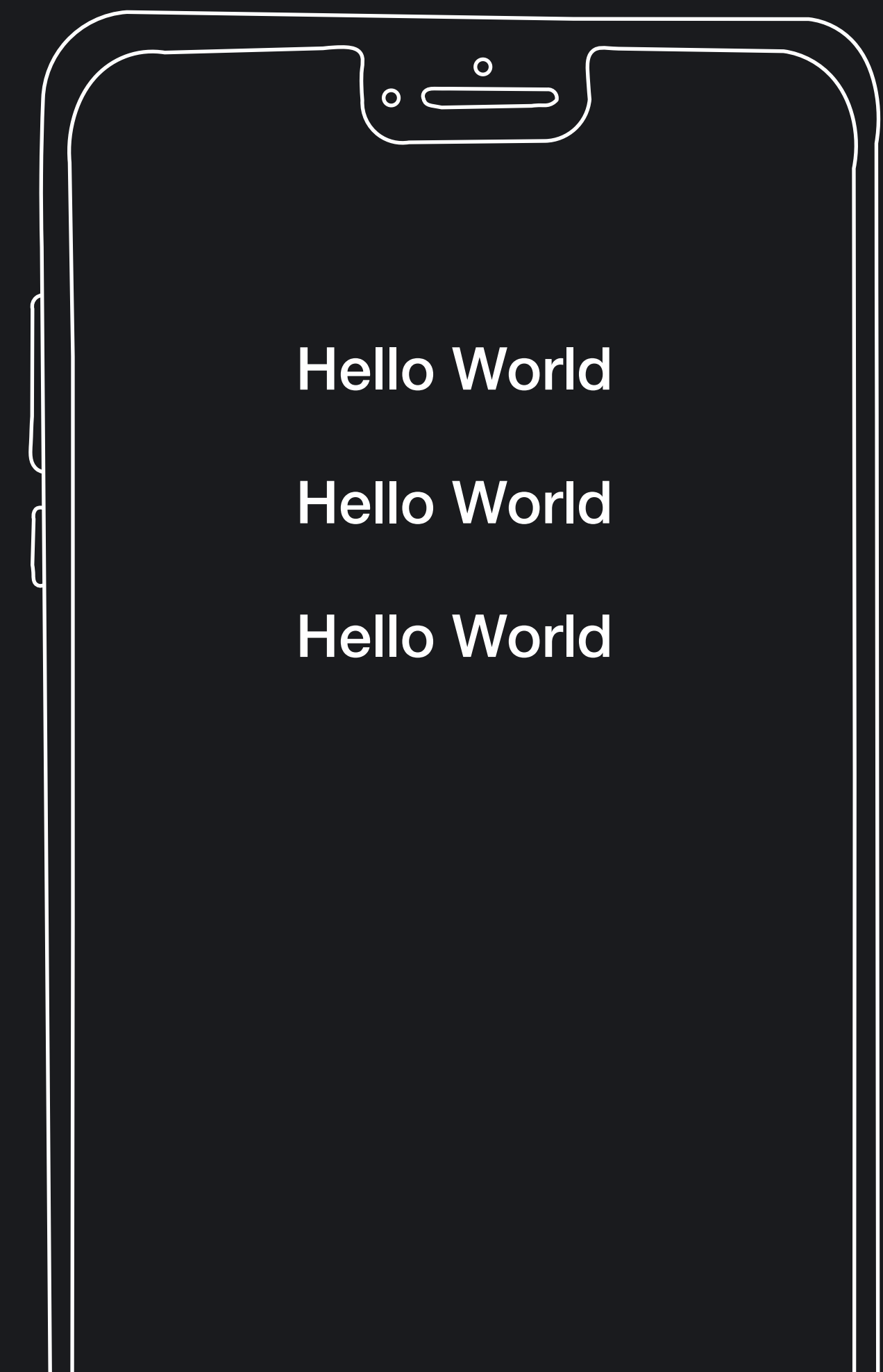
Column

```
Column(  
➡ horizontalAlignment = Alignment.End  
) {  
    Text("Hello World")  
    Text("Hello World")  
    Text("Hello World")  
}
```

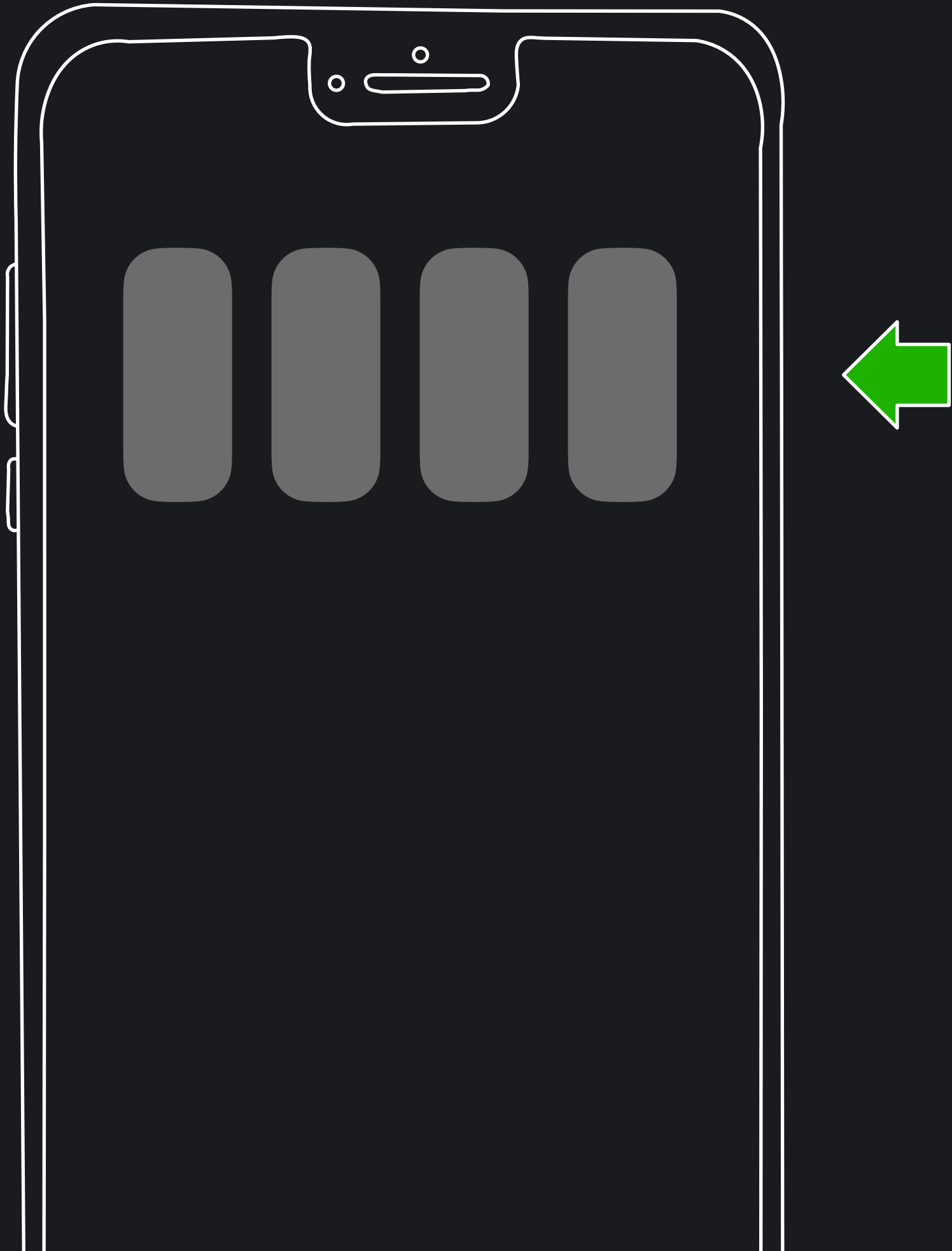


Column

```
Column(  
➡ Alignment.CenterHorizontally  
) {  
    Text("Hello World")  
    Text("Hello World")  
    Text("Hello World")  
}
```

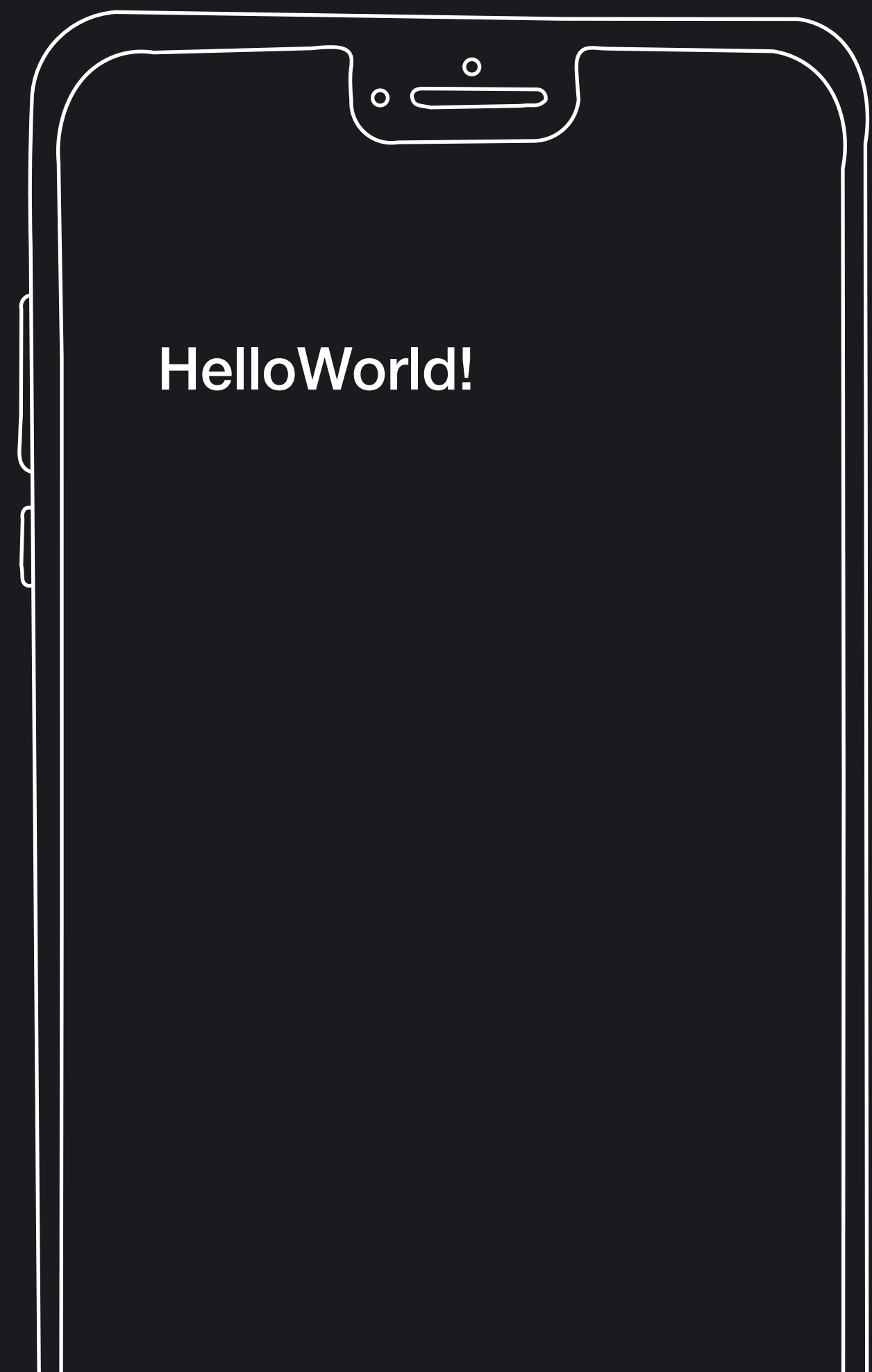


Row



Row

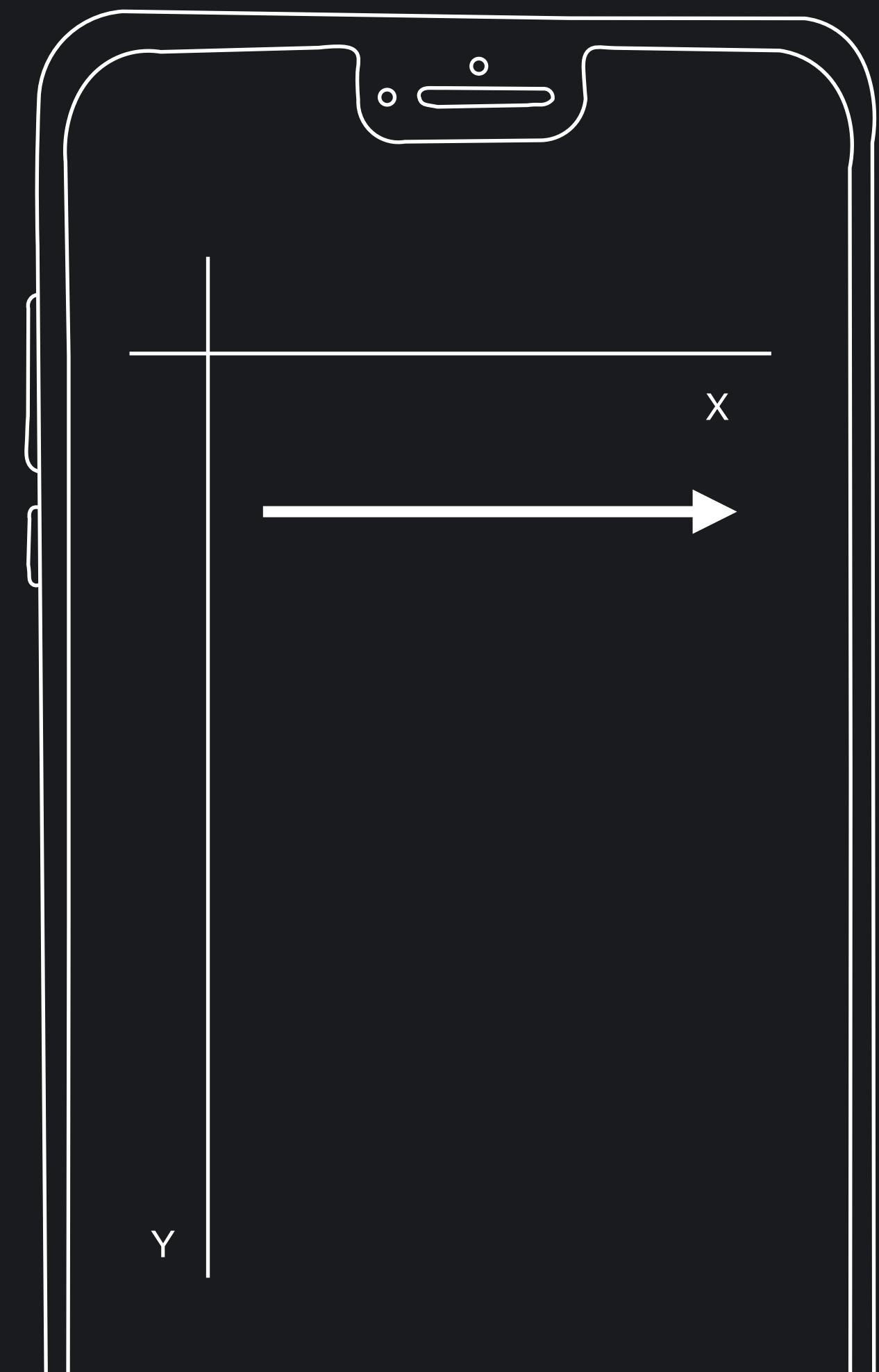
```
Row {  
  Text("Hello")  
  Text("World")  
  Text("!")  
}
```



Row

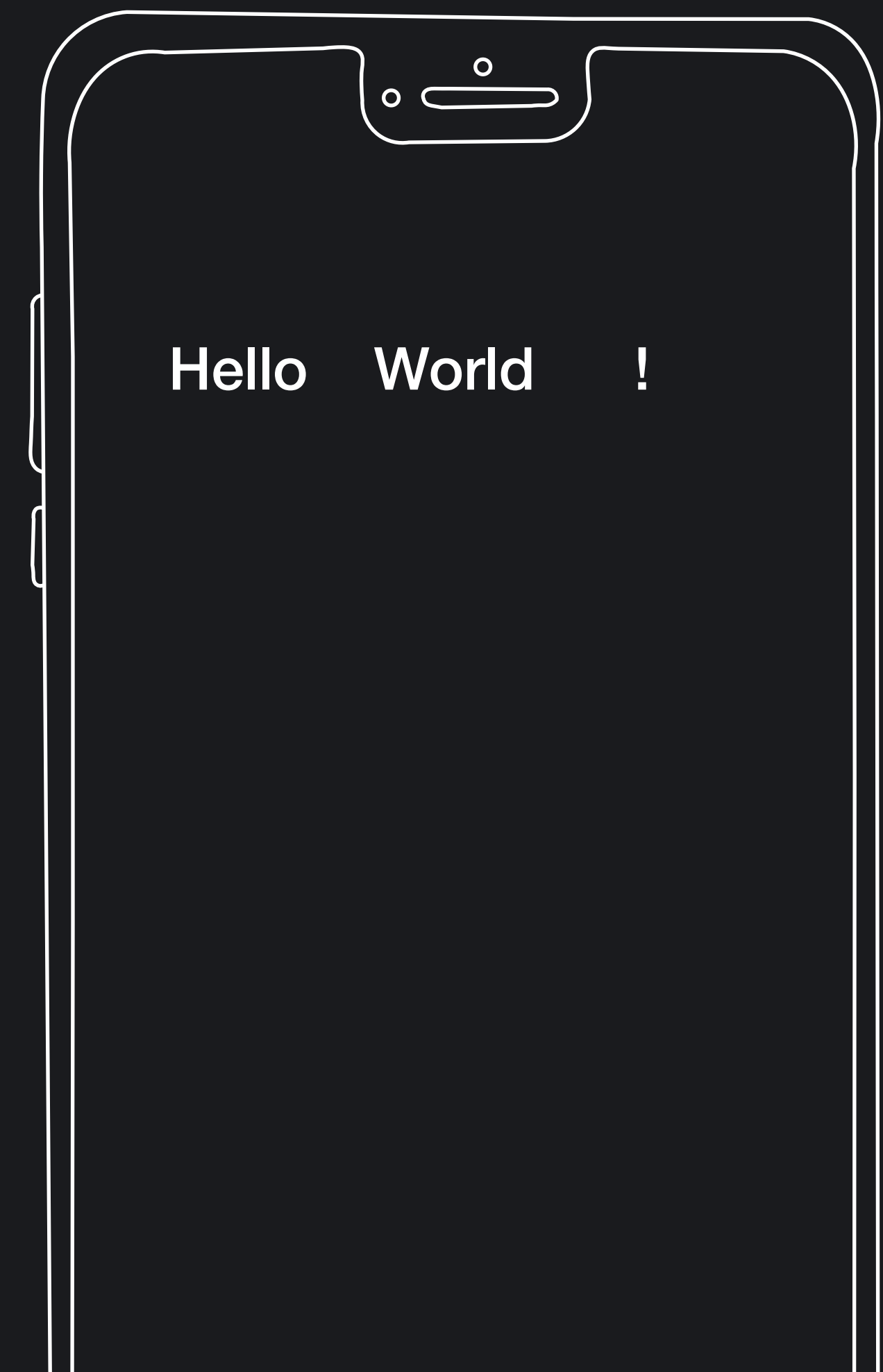
➡

```
Row(  
  horizontalArrangement =  
) {  
  Text("Hello World")  
  Text("Hello World")  
  Text("Hello World")  
}
```



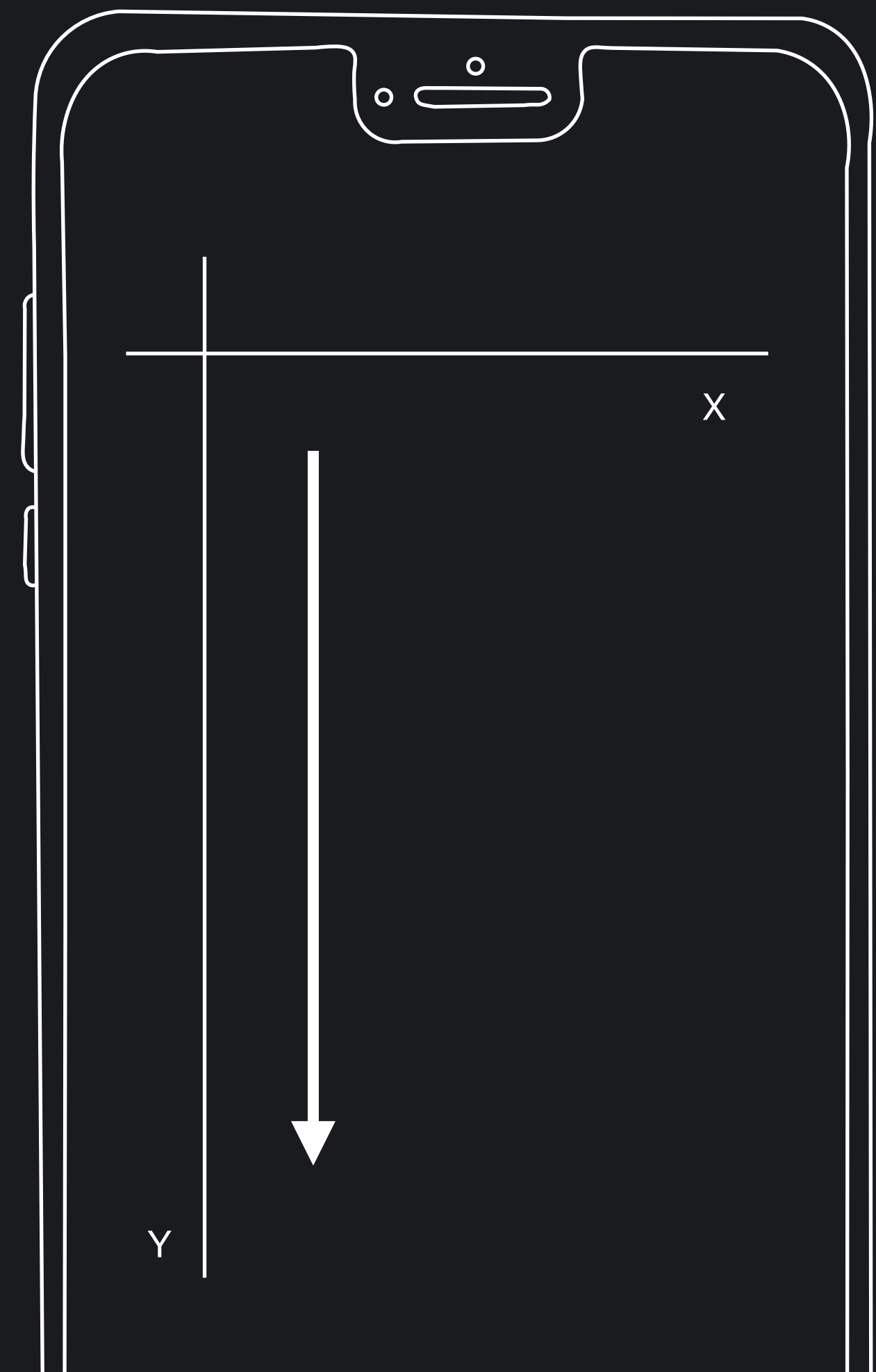
Row

```
Row(  
    Arrangement.SpaceBetween  
) {  
    Text("Hello")  
    Text("World")  
    Text("!")  
}
```



Row

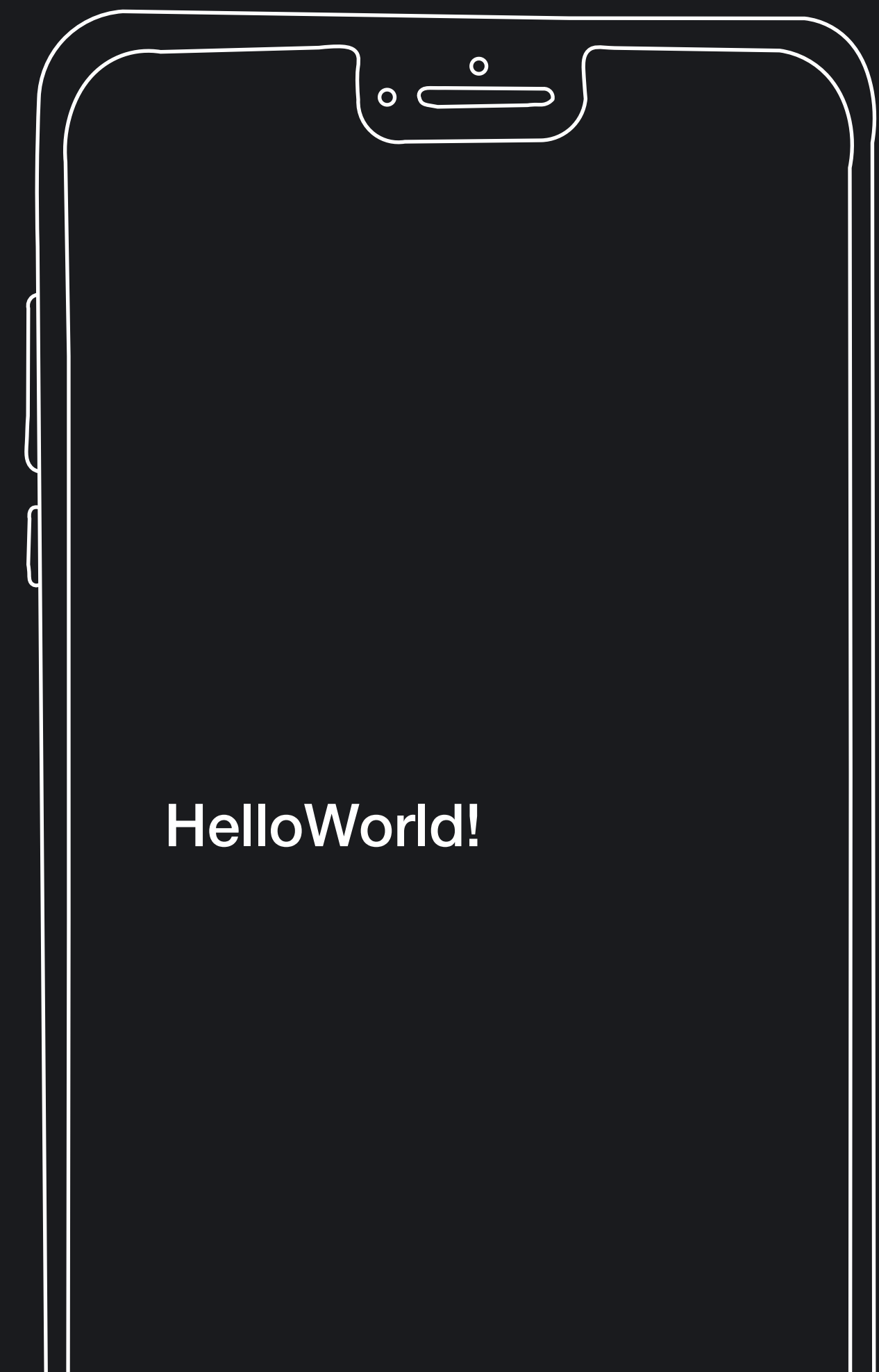
```
Row(  
  verticalAlignment =  
) {  
  Text("Hello World")  
  Text("Hello World")  
  Text("Hello World")  
}
```



Row



```
Row(  
  Alignment.CenterVertically  
) {  
  Text("Hello World")  
  Text("Hello World")  
  Text("Hello World")  
}
```



Column & Row

Column



Vertical Arrangement

Horizontal Alignment

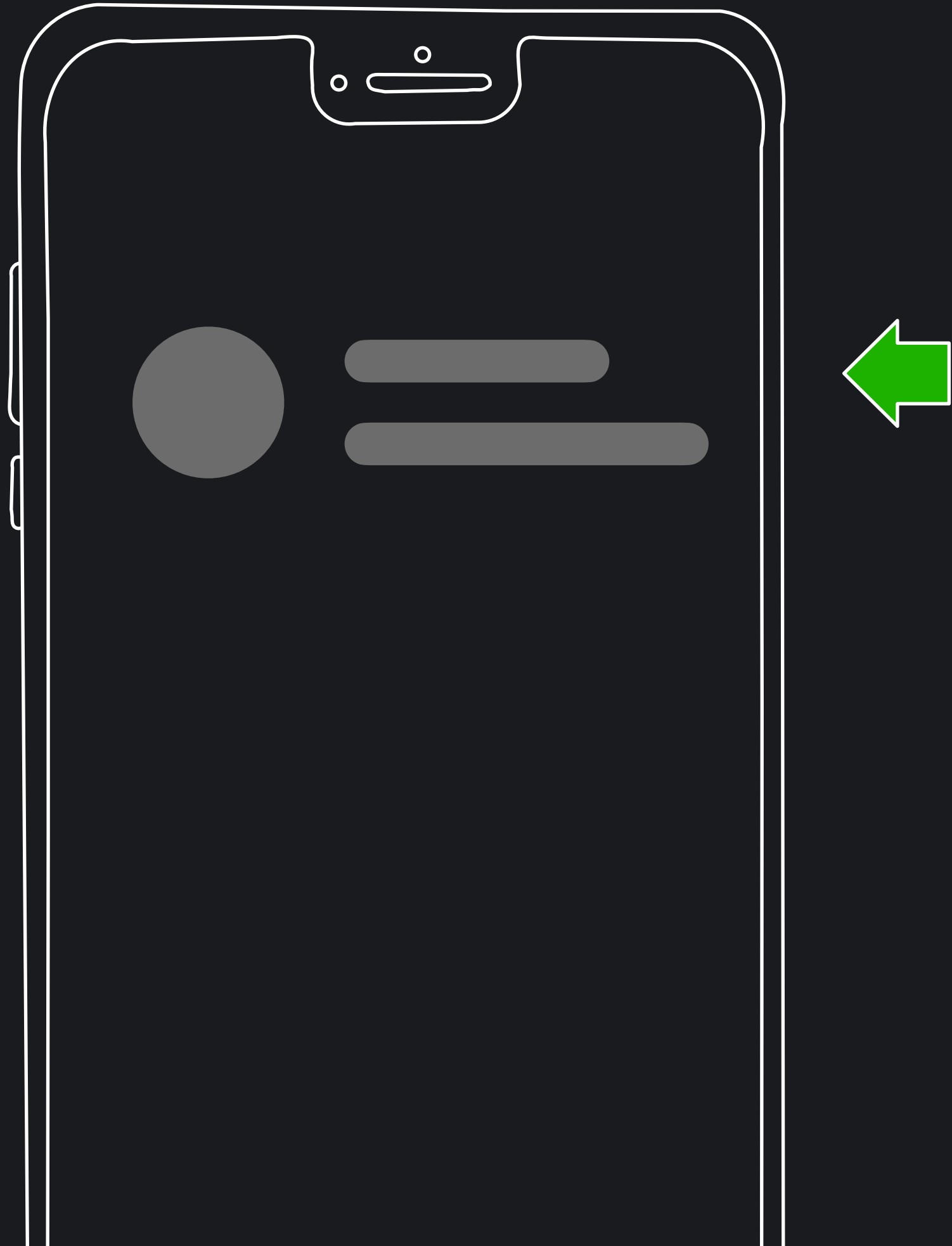
Row



Horizontal Arrangement

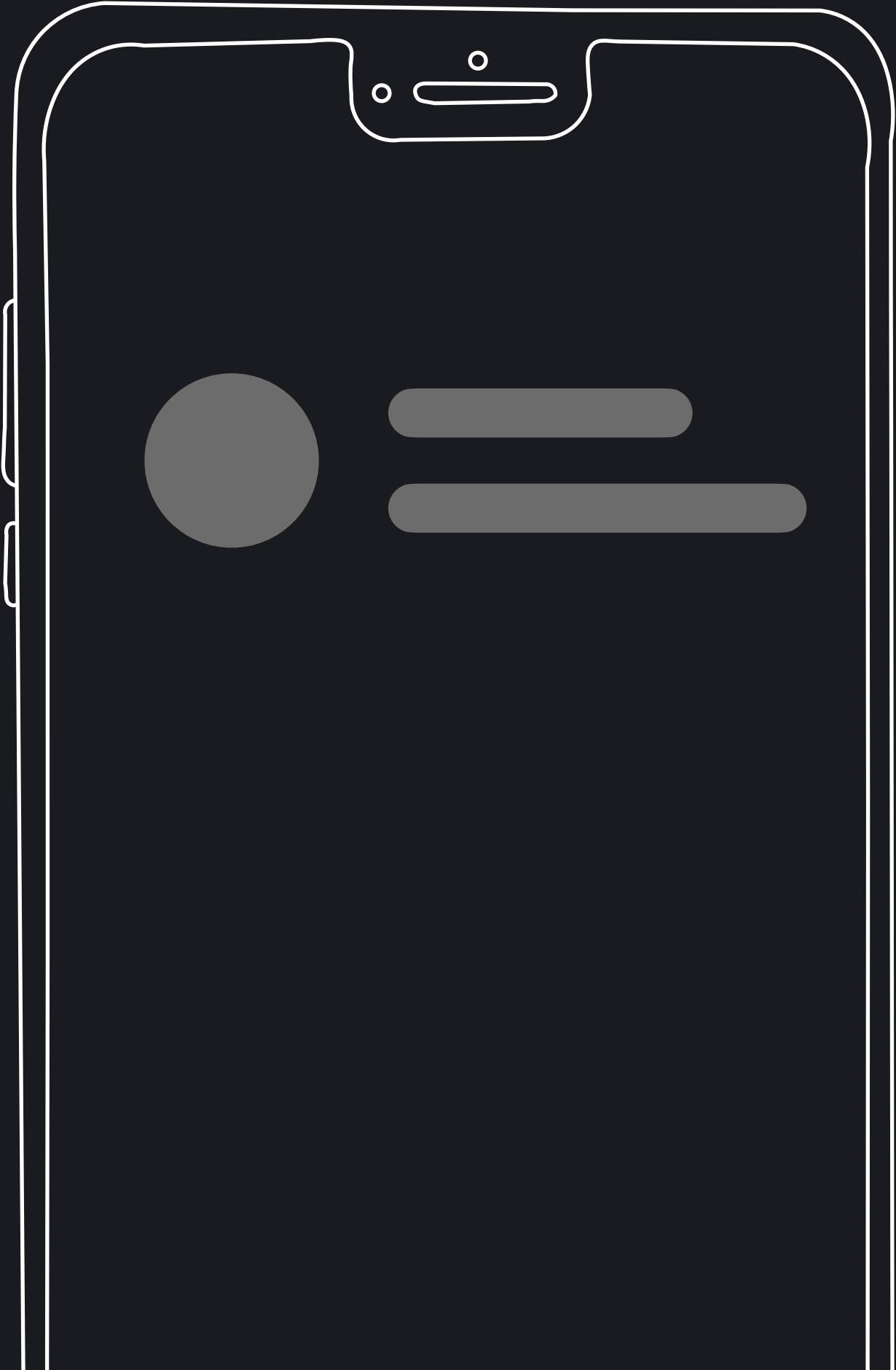
Vertical Alignment

Column & Row



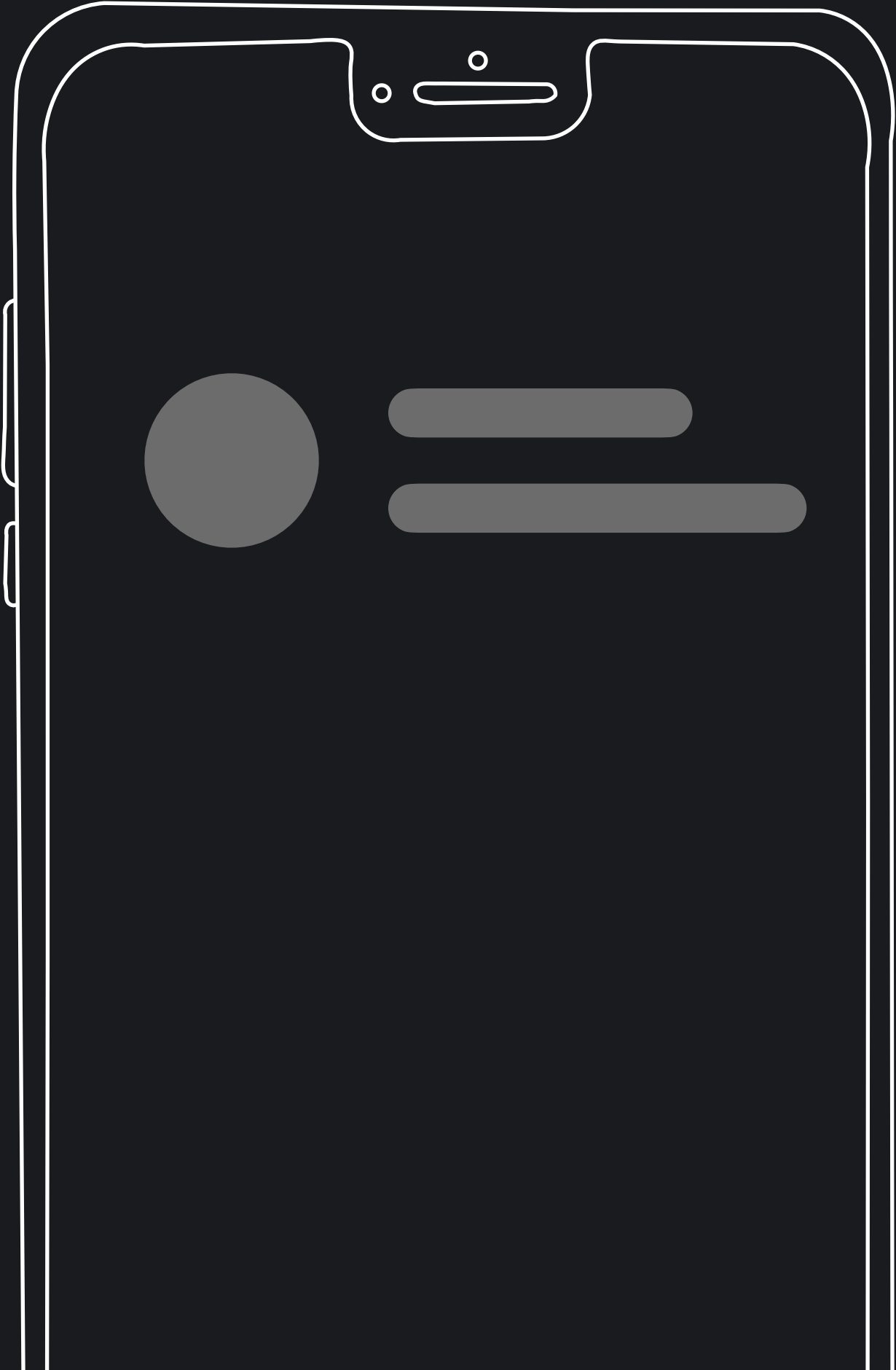
Column & Row

```
Row {  
    Image(...)  
  
}
```



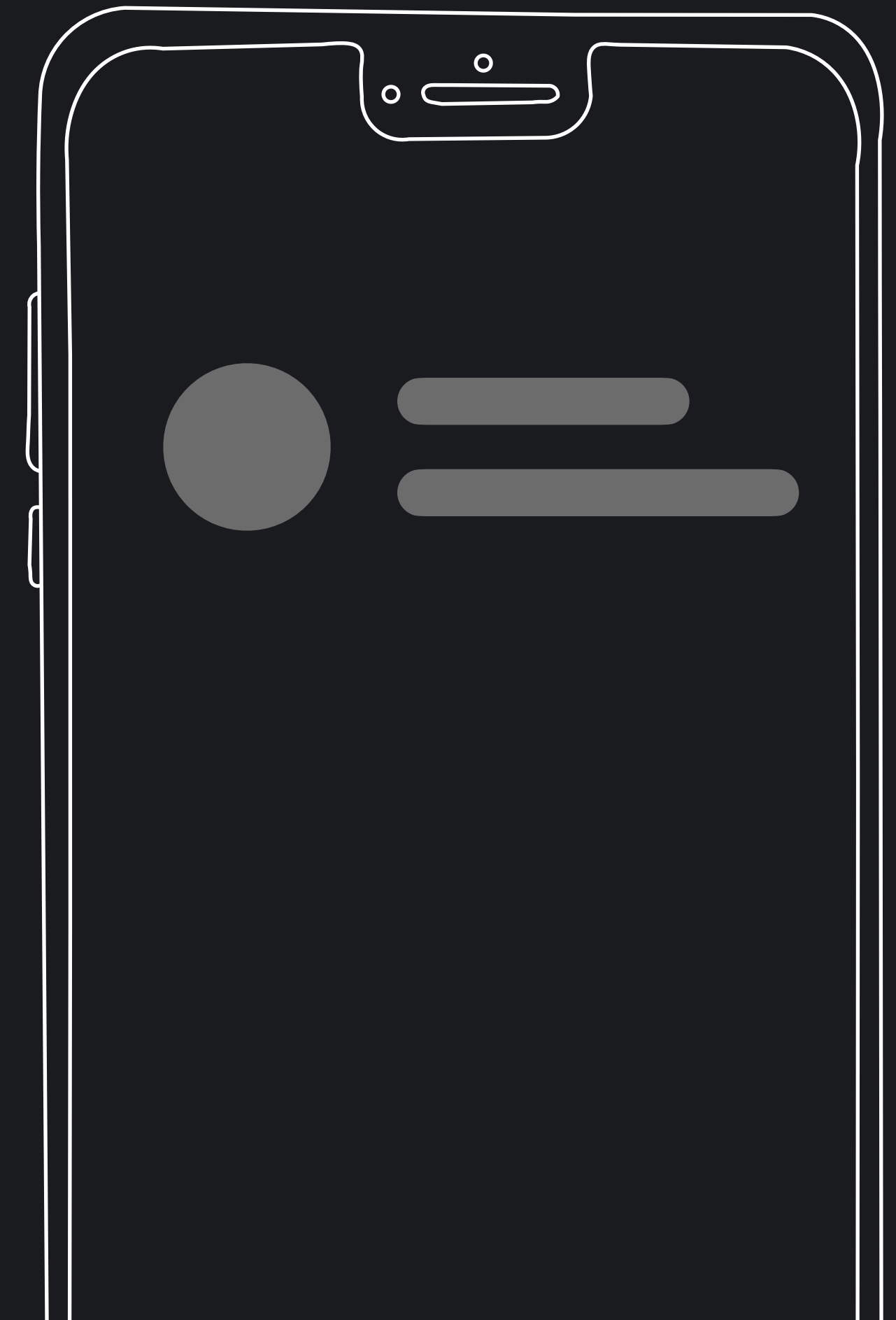
Column & Row

```
Row {  
  Image(...)  
  Column {  
  
  }  
}
```

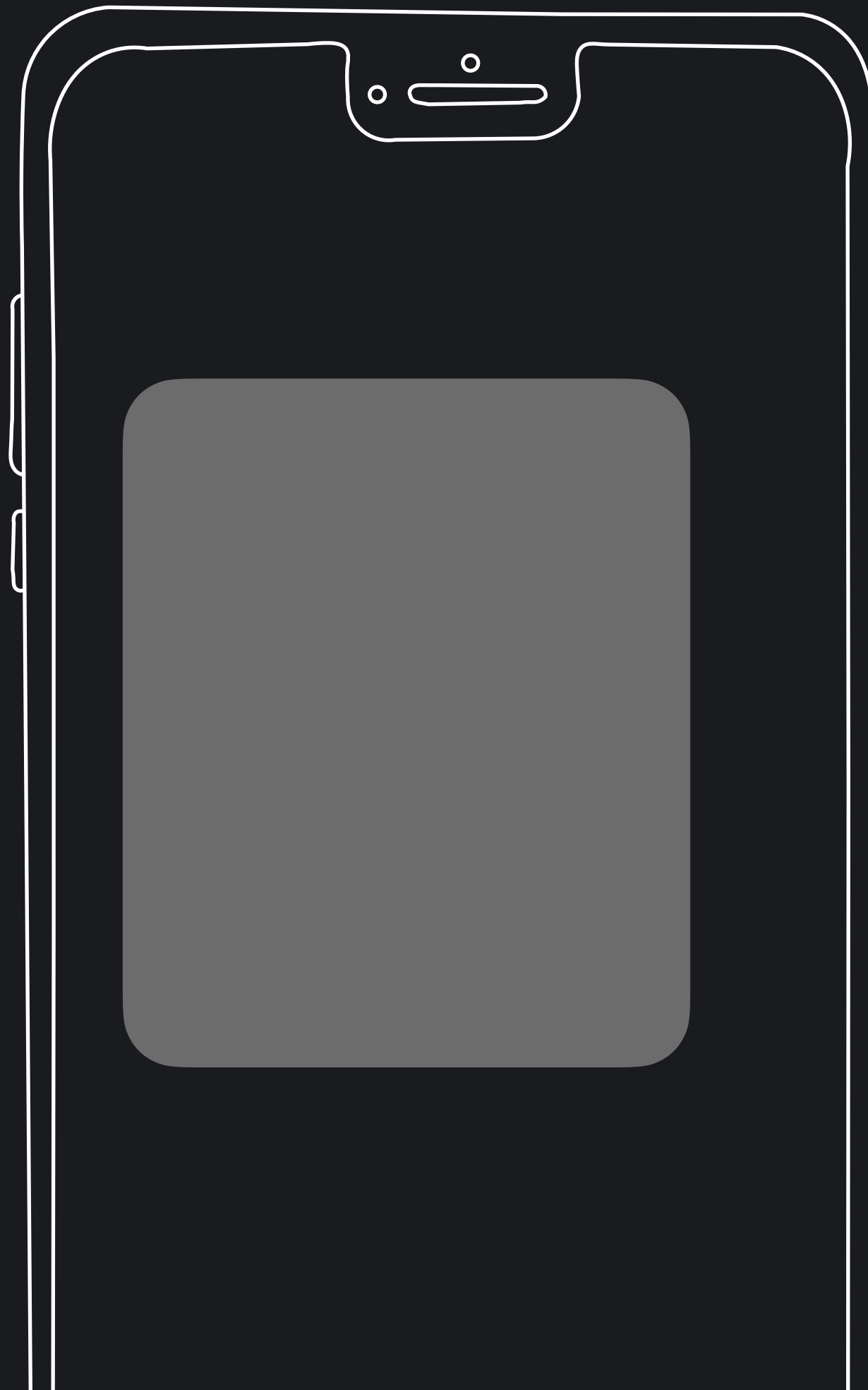


Column & Row

```
Row {  
    Image(...)  
    Column {  
        Text("...")  
        Text("...")  
    }  
}
```

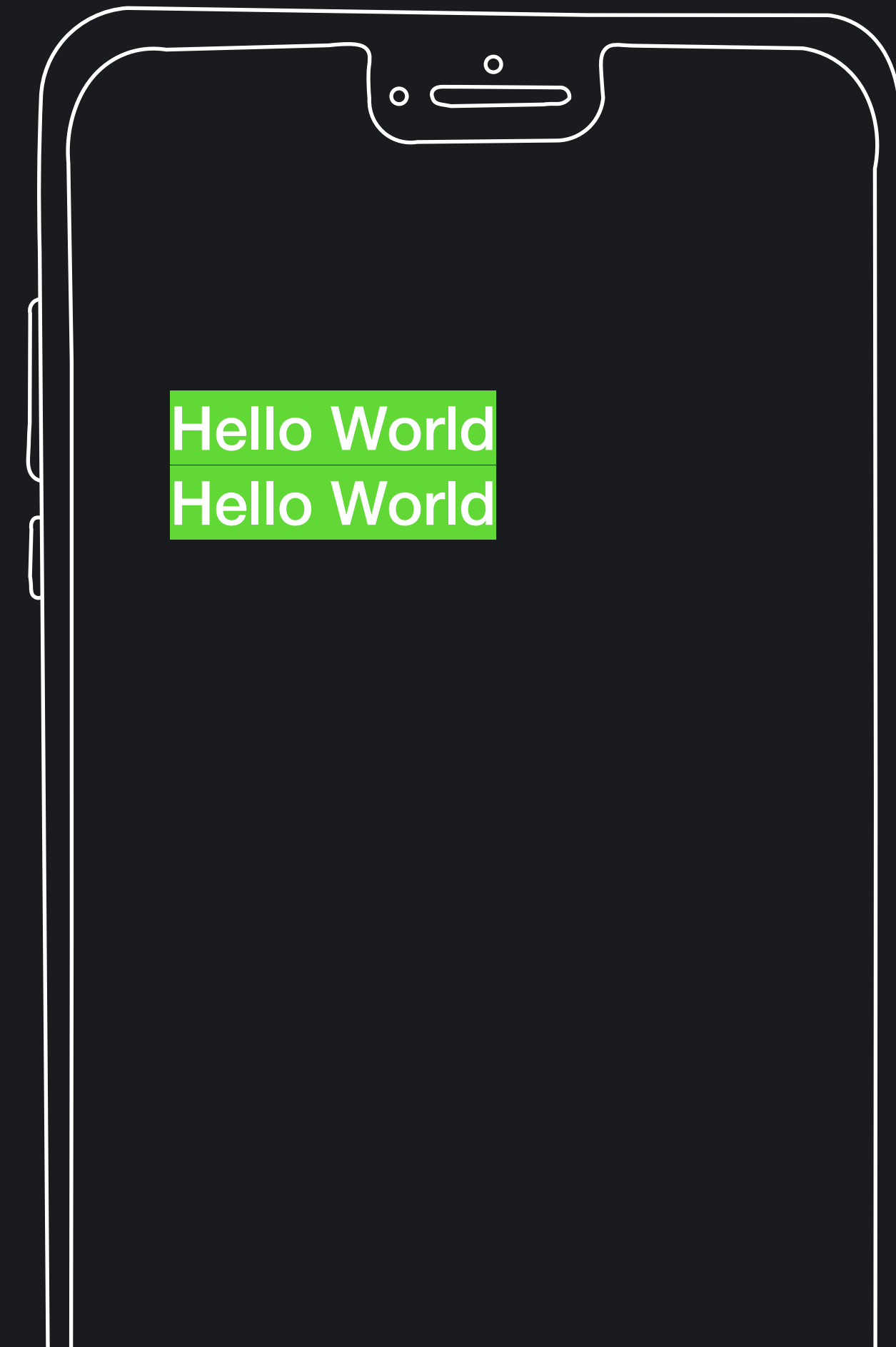


Layout Modifiers



Layout Modifiers

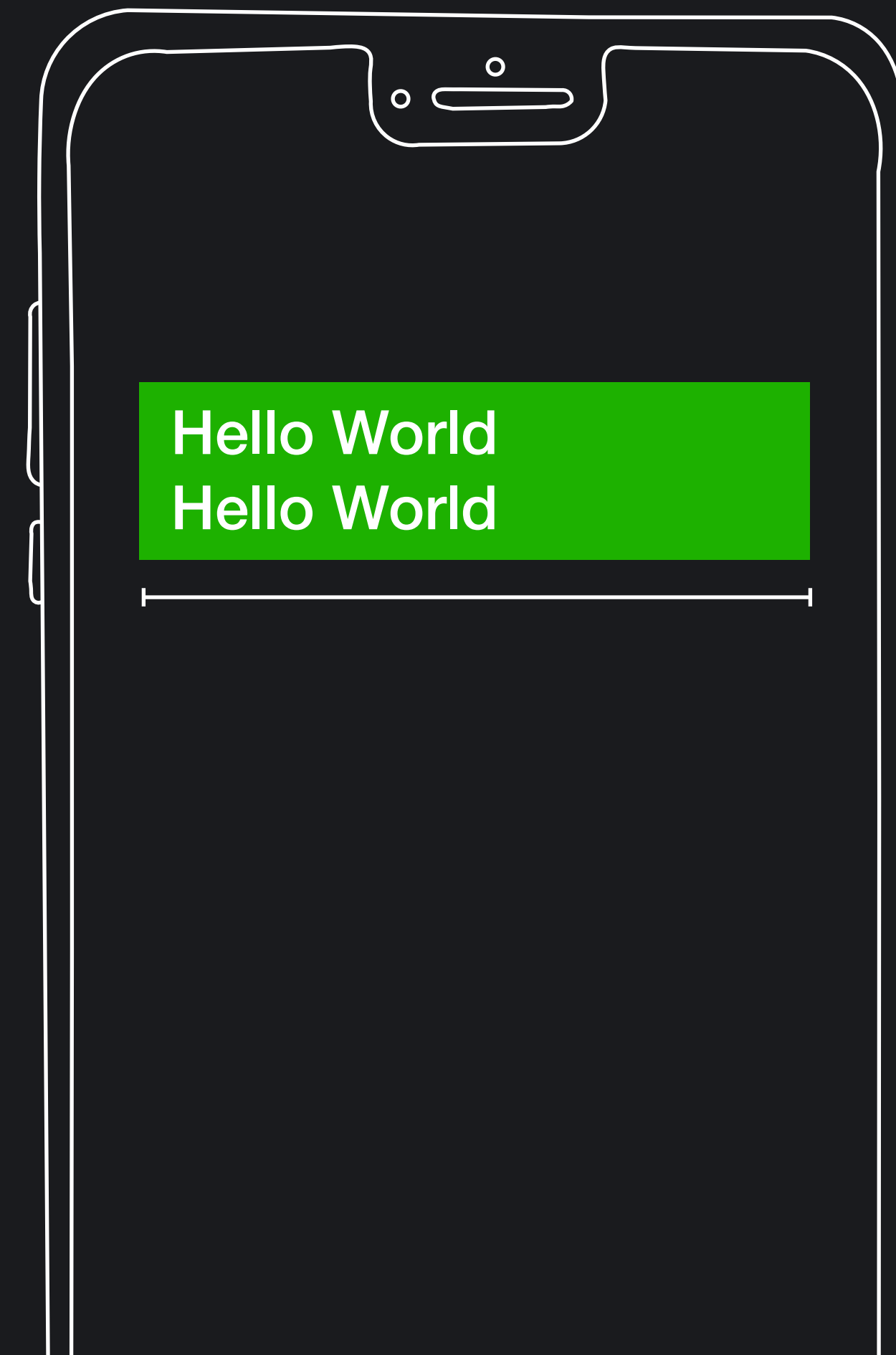
```
Column(  
    Modifier.background(Color.Green)  
) {  
    Text(text = "Hello World")  
    Text(text = "Hello World")  
}
```



Layout Modifiers



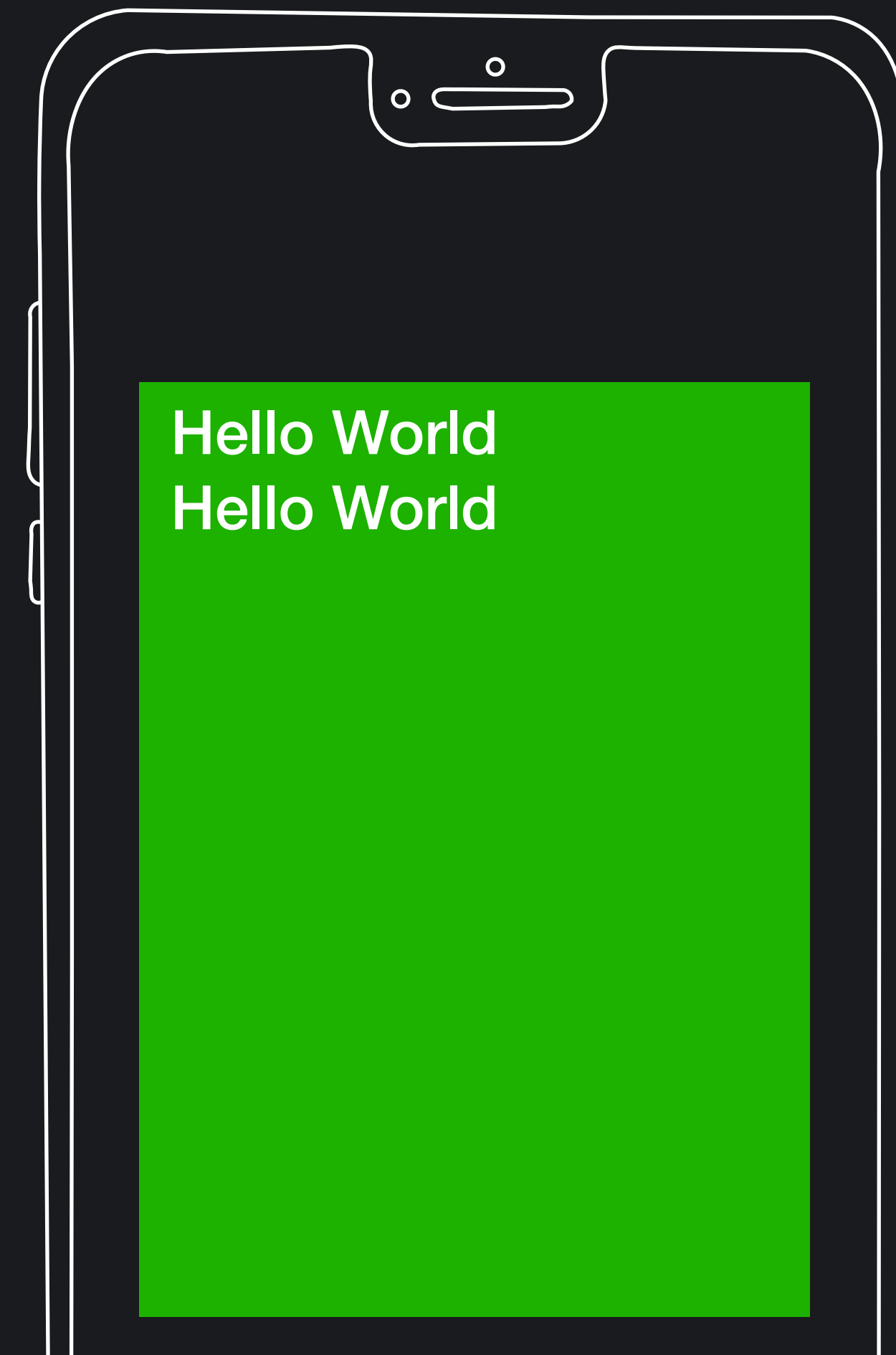
```
Column(  
    Modifier  
        .background(Color.Green)  
        .fillMaxWidth()  
) {  
    Text(text = "Hello World")  
    Text(text = "Hello World")  
}
```



Layout Modifiers



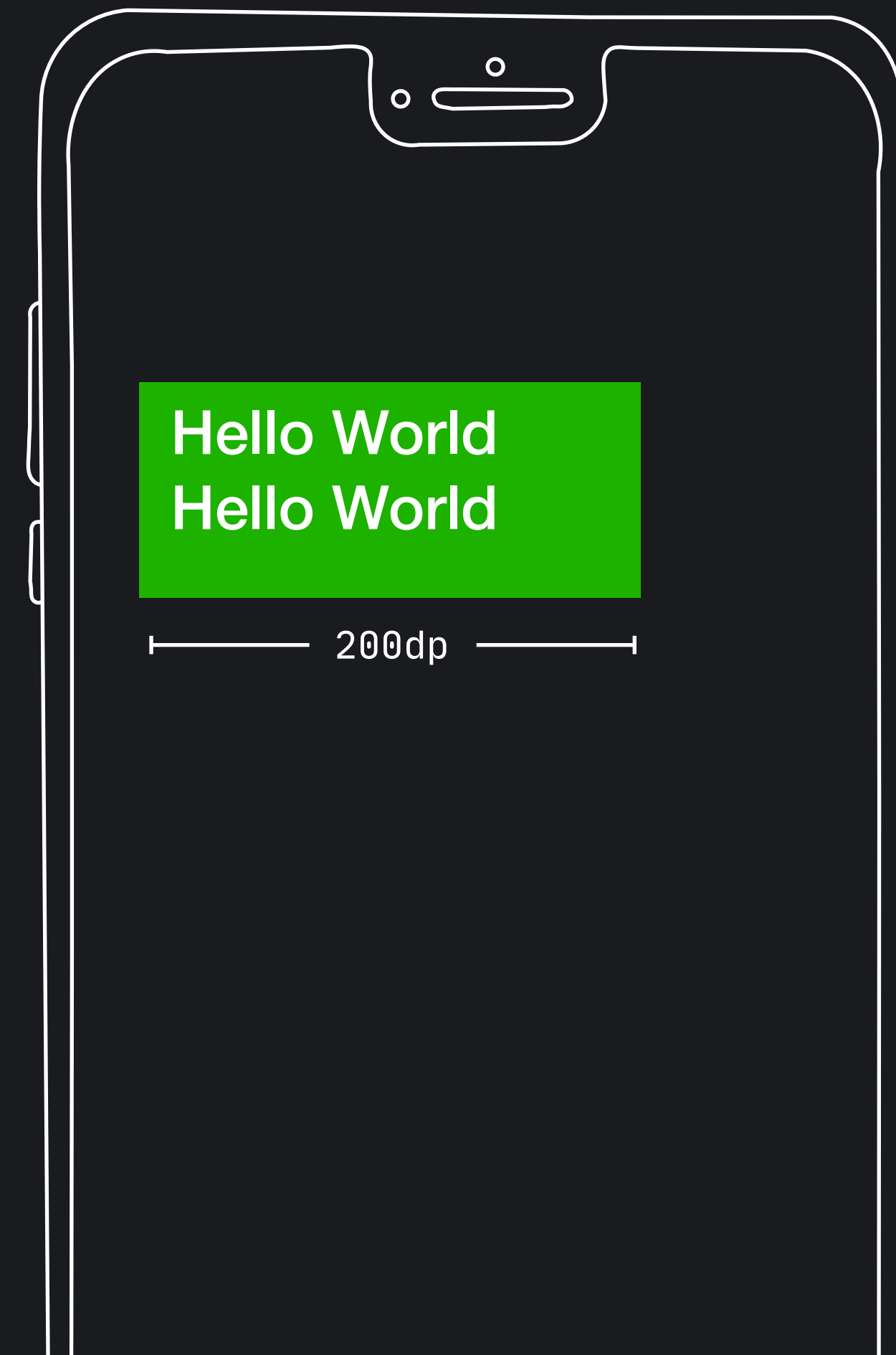
```
Column(  
    Modifier  
        .background(Color.Green)  
        .fillMaxSize()  
) {  
    Text(text = "Hello World")  
    Text(text = "Hello World")  
}
```



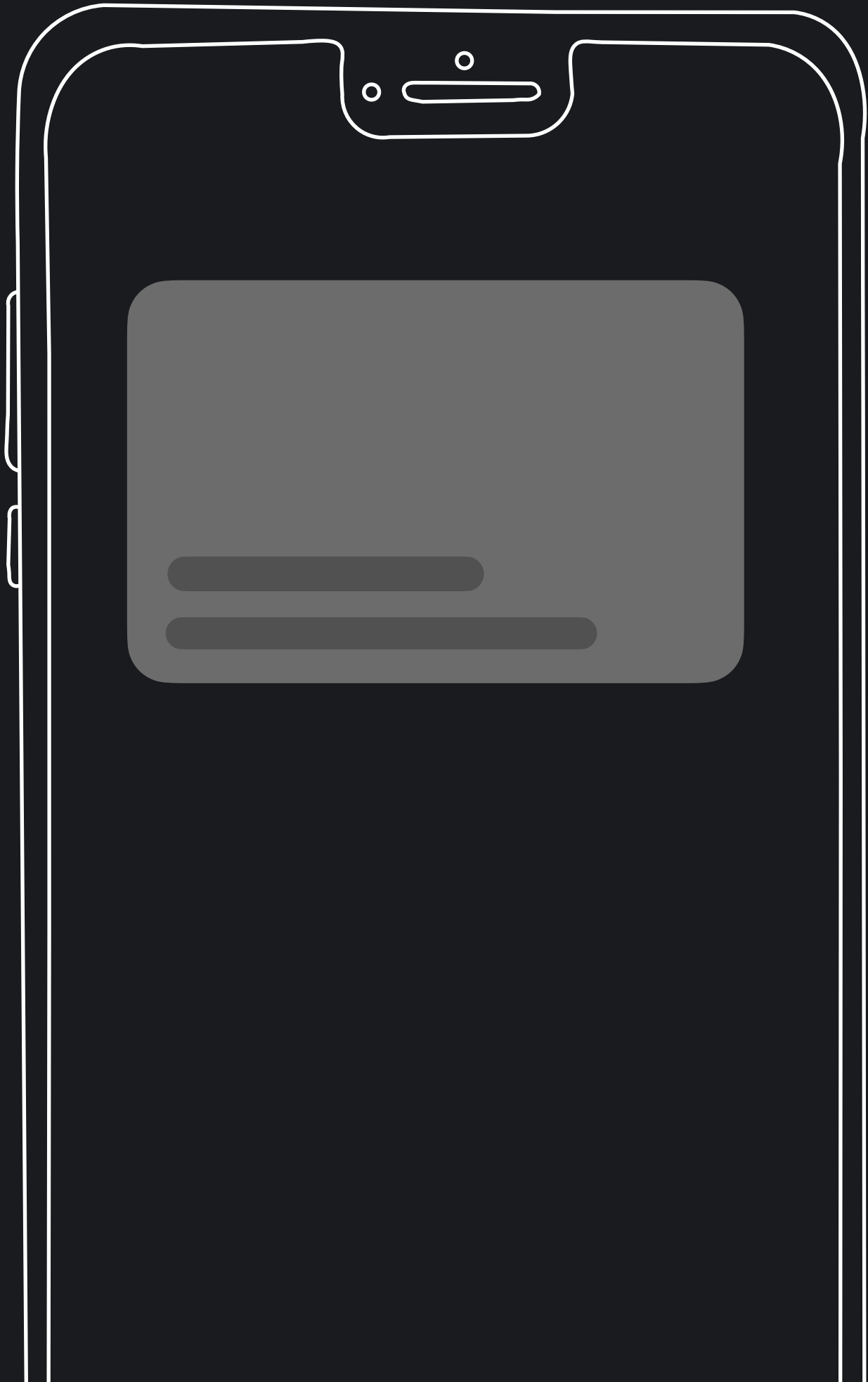
Layout Modifiers



```
Column(  
    Modifier  
        .background(Color.Green)  
        .width(200.dp)  
) {  
    Text(text = "Hello World")  
    Text(text = "Hello World")  
}
```

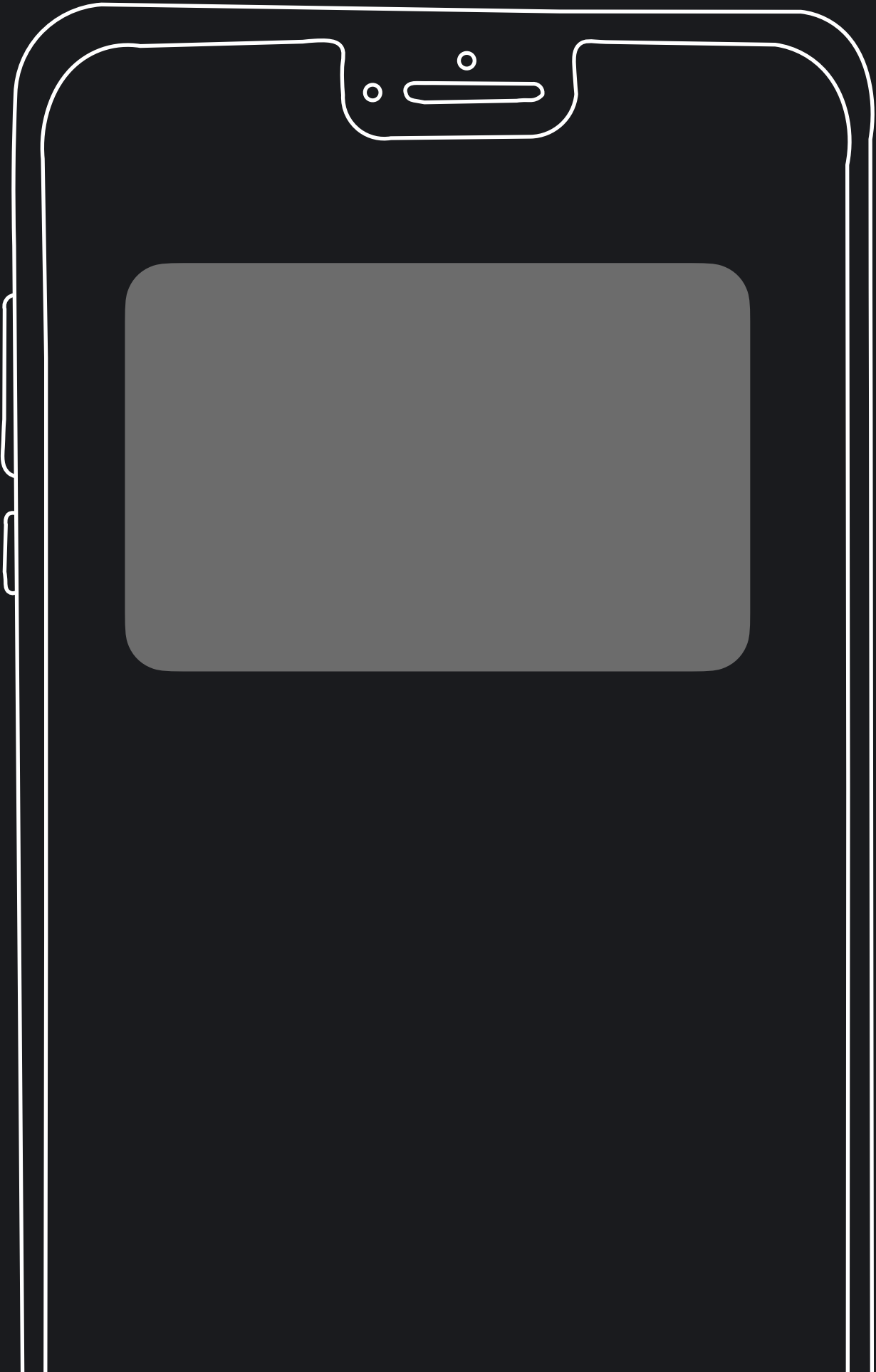


Box



Box

```
Box {  
  Image(...)  
}
```



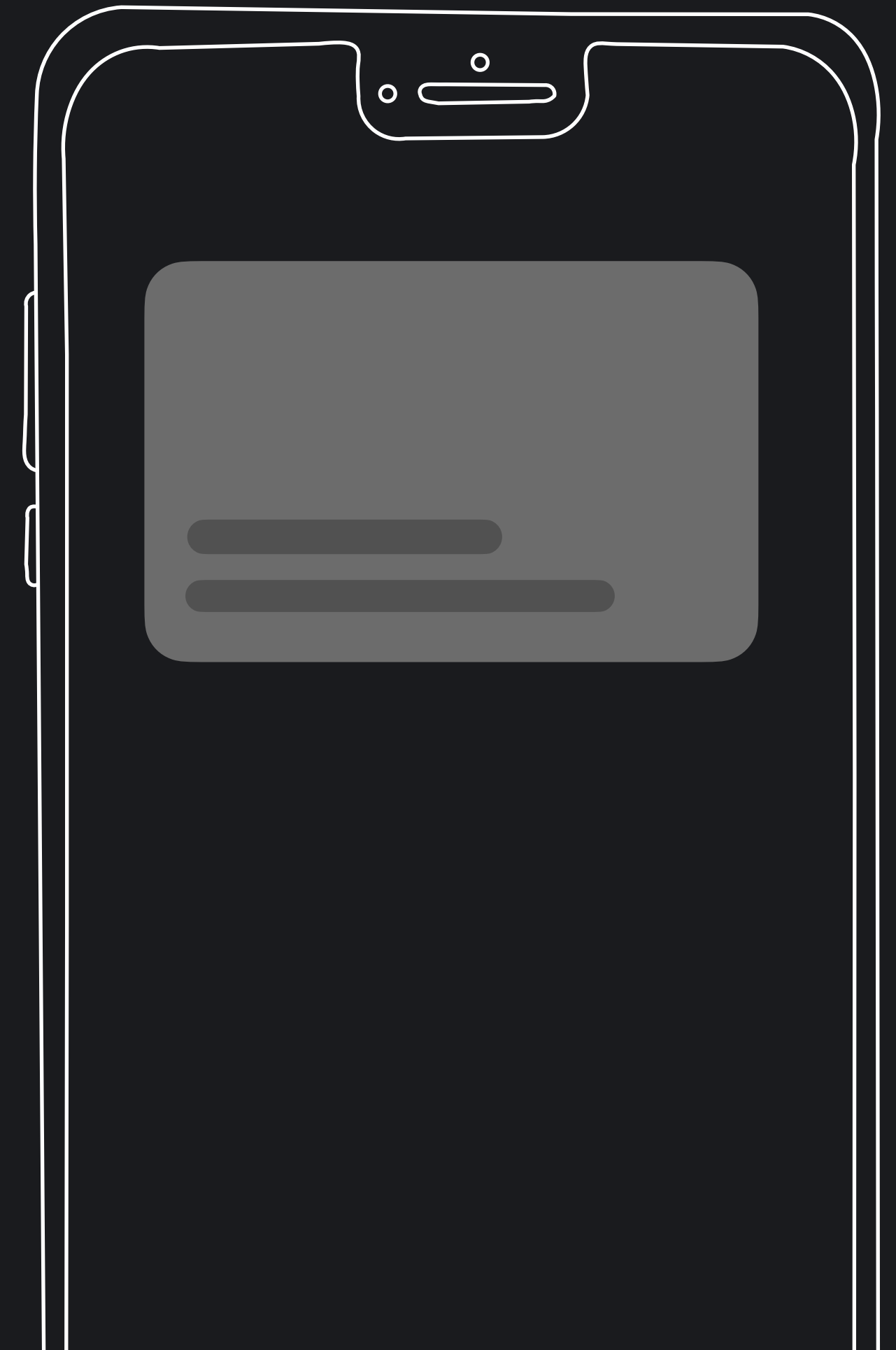

Box

```
Box {  
  Image(...)  
  
  Column {  
    Text(...)  
    Text(...)  
  }  
}
```

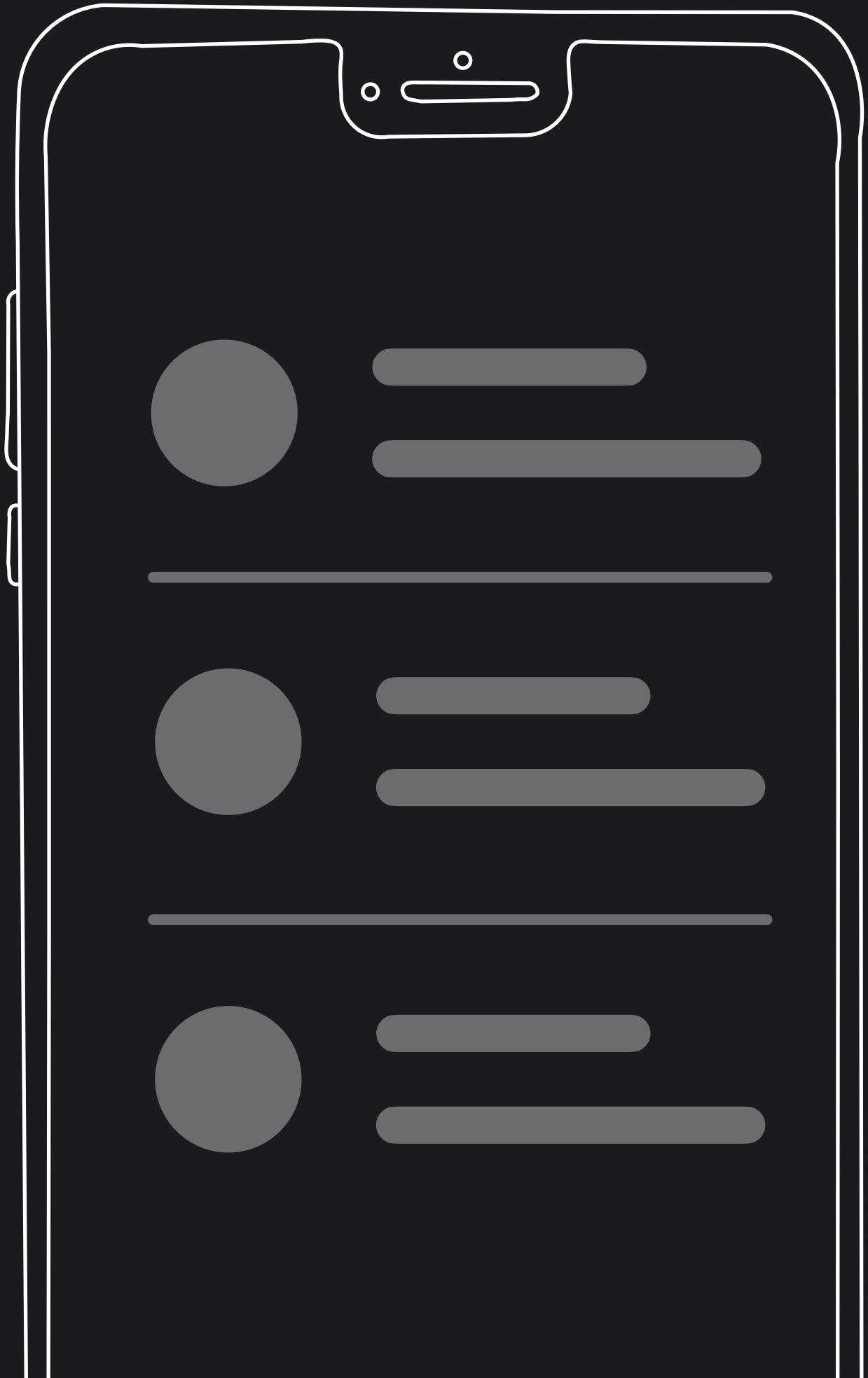


Box

```
Box {  
    Image(...)  
  
    Column(  
        Modifier.align(  
            Alignment.BottomEnd  
        )  
    ) {  
        Text(...)  
        Text(...)  
    }  
}
```



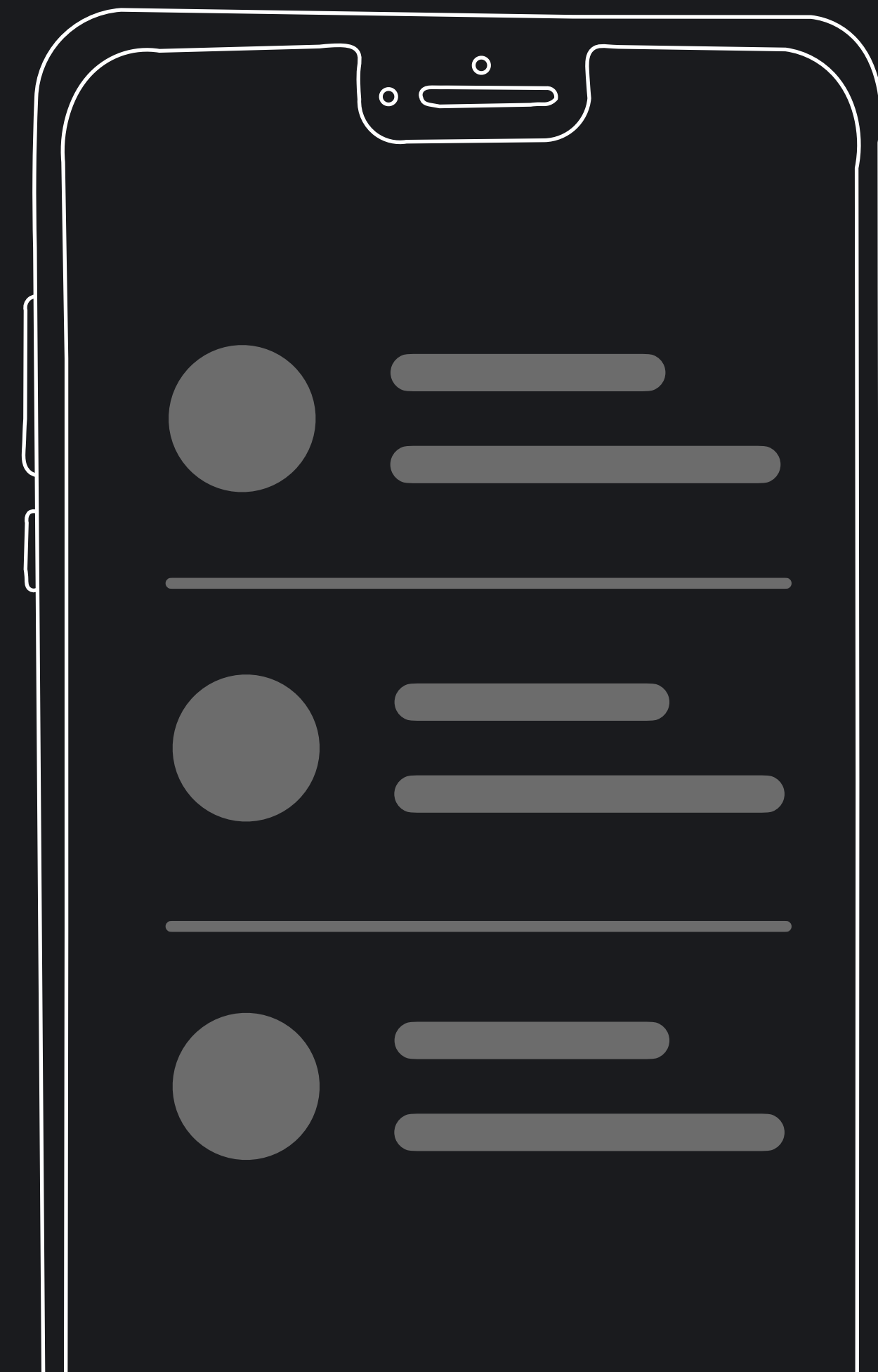
List



List

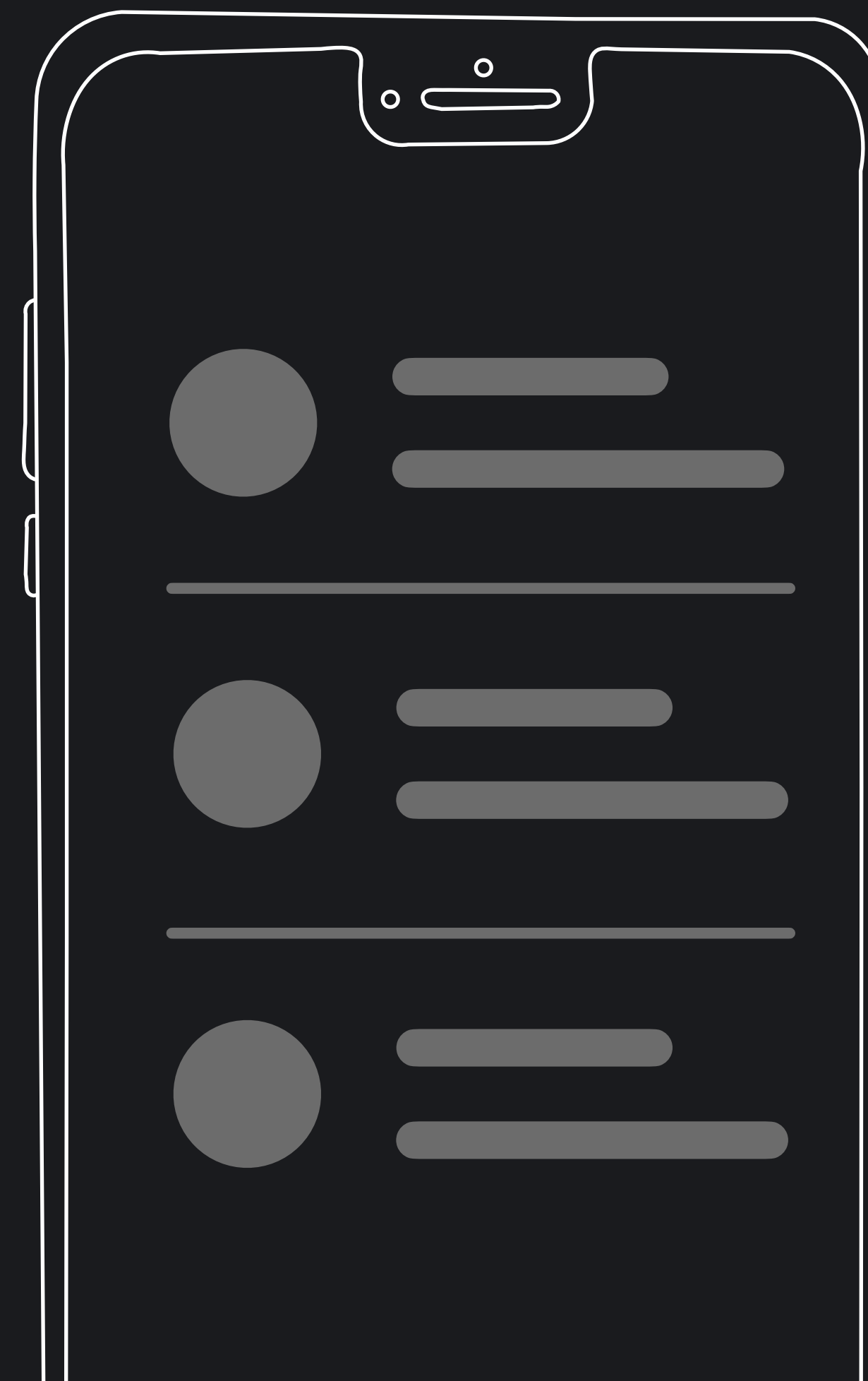
```
@Composable
fun EmployeeListView(items: List<Item>) {
    LazyColumn {

    }
}
```



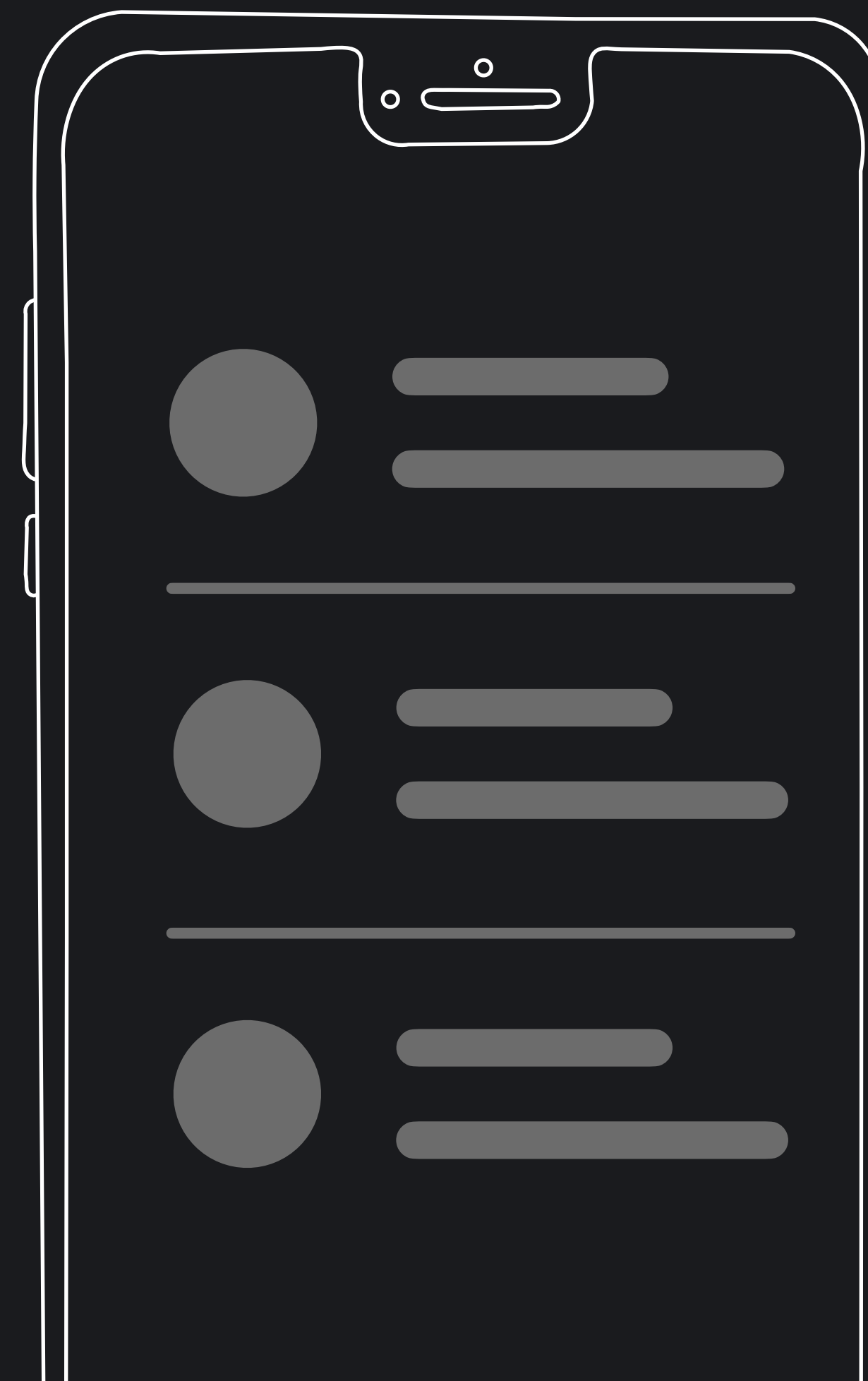
List

```
@Composable
fun EmployeeListView(items: List<Item>) {
    LazyColumn {
        items(items) { item →
        }
    }
}
```

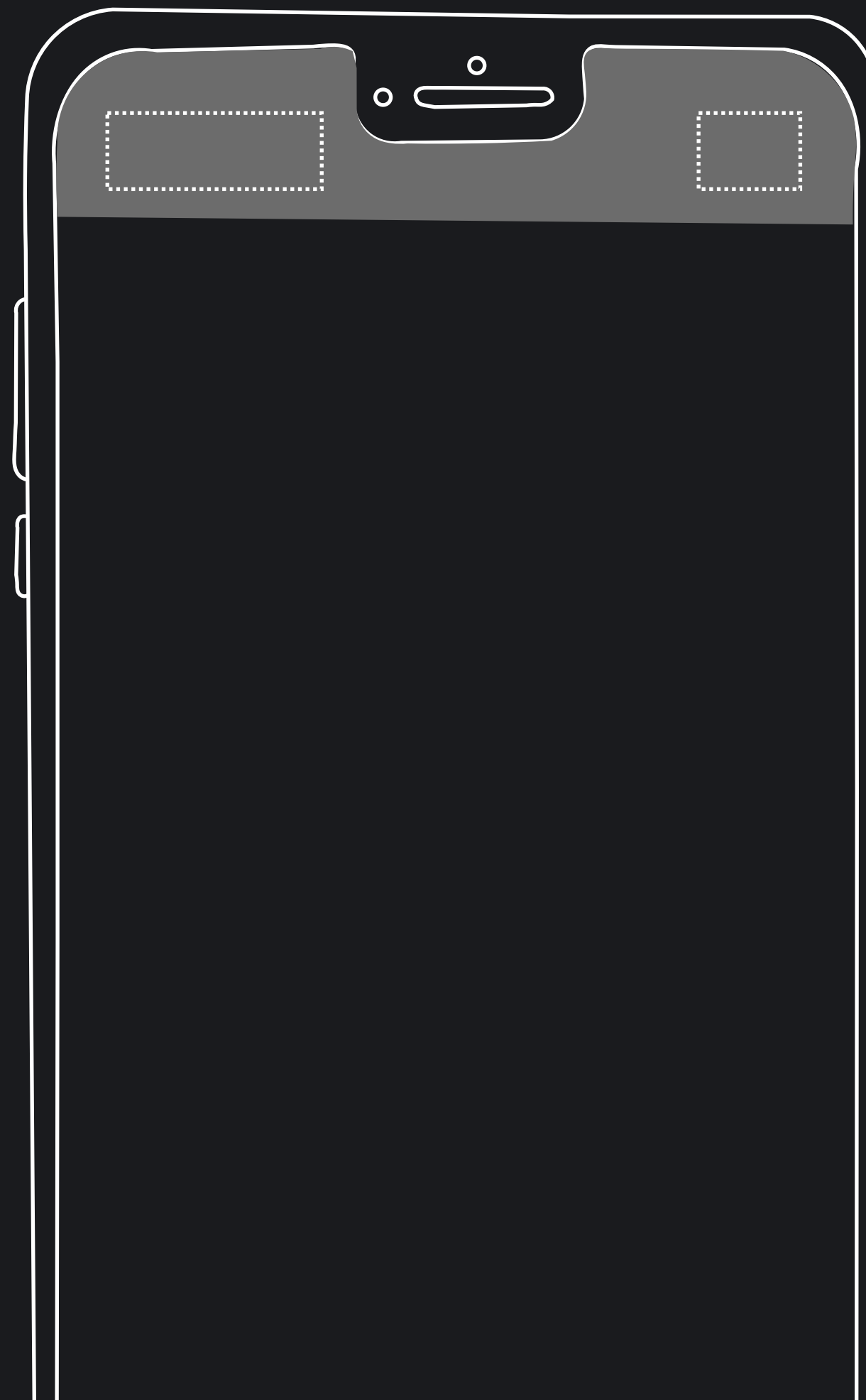


List

```
@Composable
fun EmployeeListView(items: List<Item>) {
    LazyColumn {
        items(items) { item →
            ItemRow(item)
        }
    }
}
```

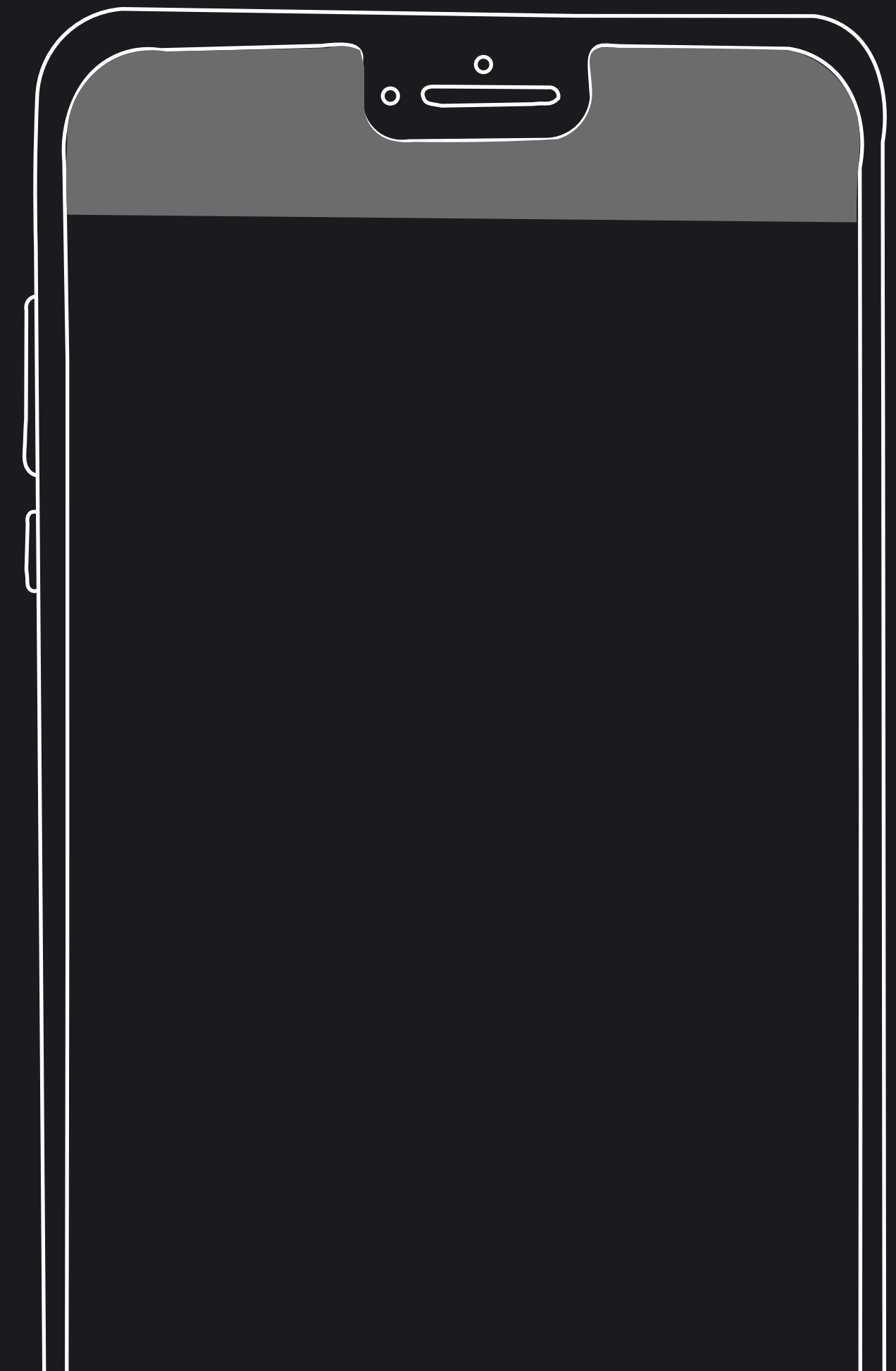


Slot Layouts



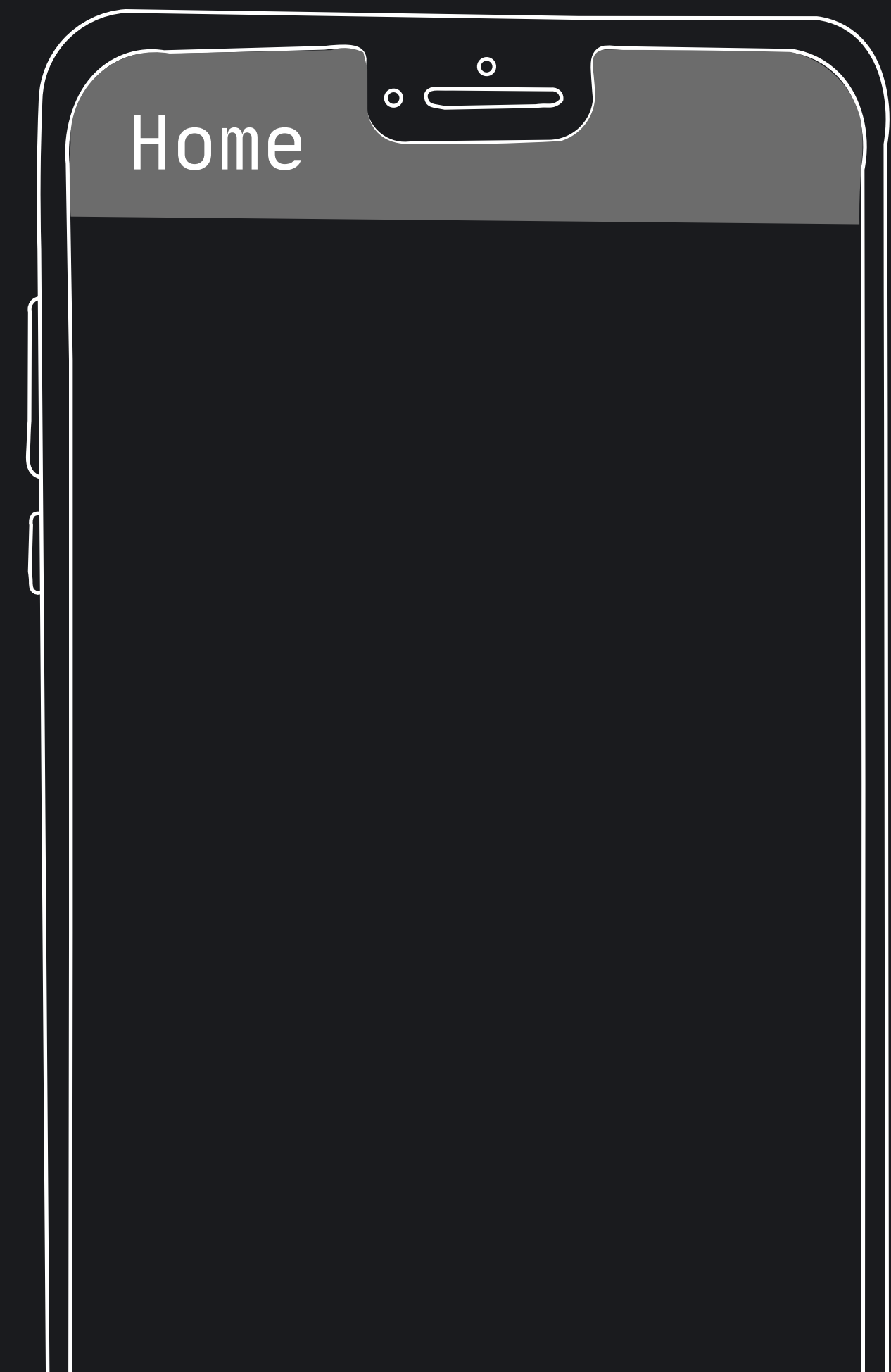
Scaffold

```
Scaffold(  
  topBar = {  
    }  
)  
{  
}
```



Scaffold

```
Scaffold(  
  topBar = {  
    Text(text = "Home")  
  }  
) {  
}
```



Scaffold

```
Scaffold(  
  topBar = {  
    Text(text = "Home")  
  }  
) {  
  Text(text = "Hello World")  
}
```



Layouts

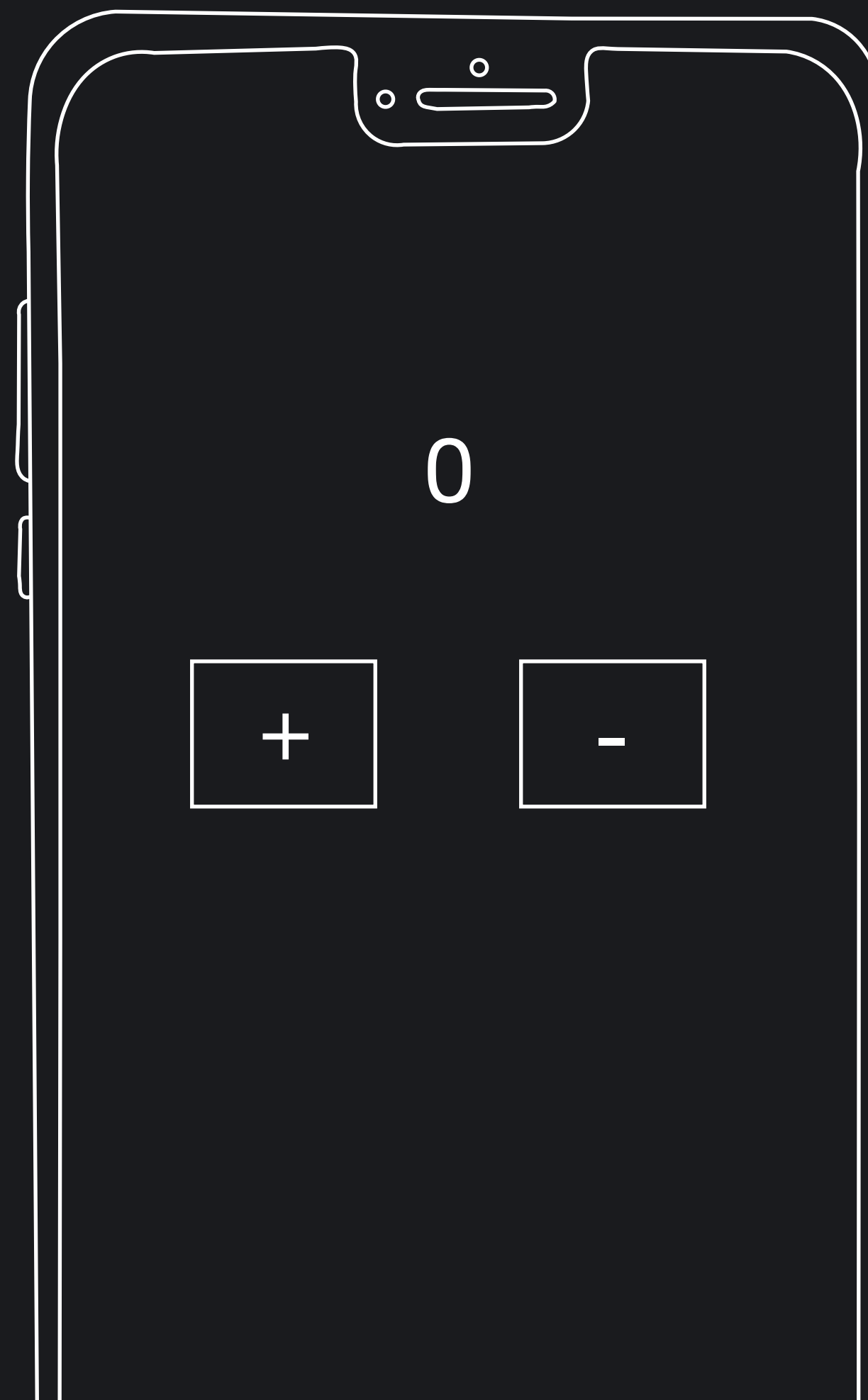
- Column
- Row
- Box
- Scaffold

Managing State

Managing State

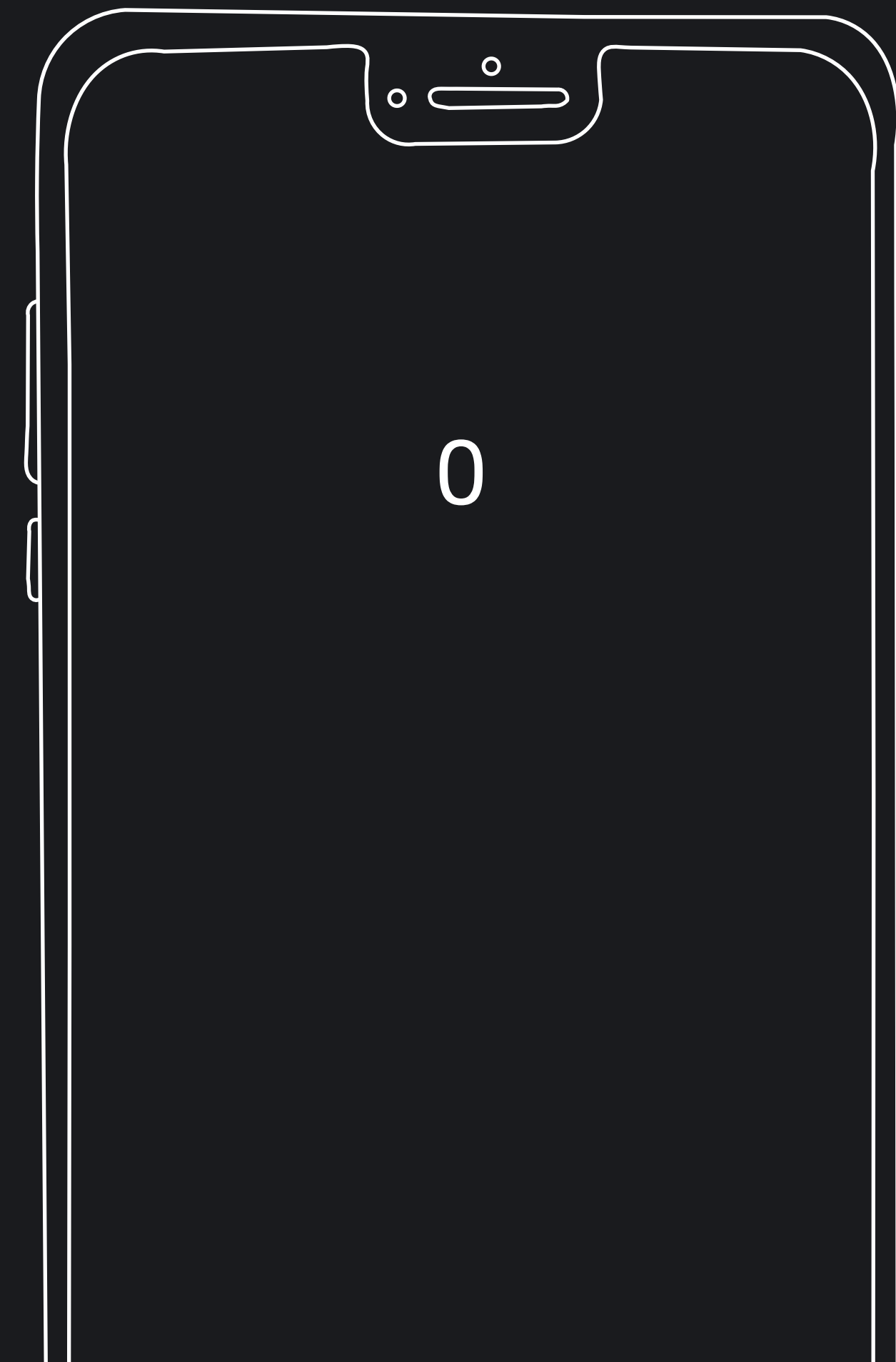
- Setup state using *mutableStateOf*

Managing State



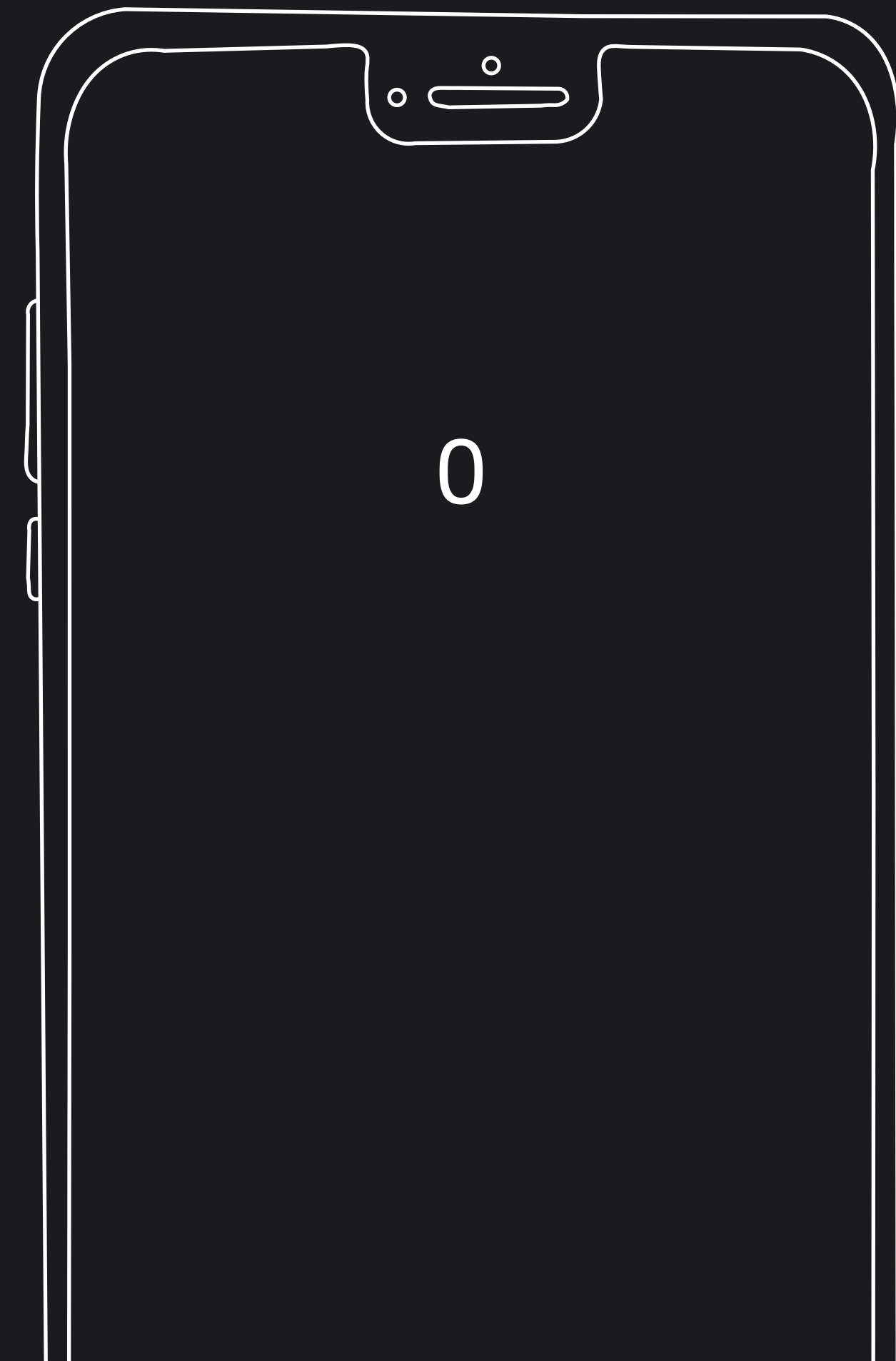
Managing State

```
var counter by remember {  
    mutableStateOf(0)  
}
```



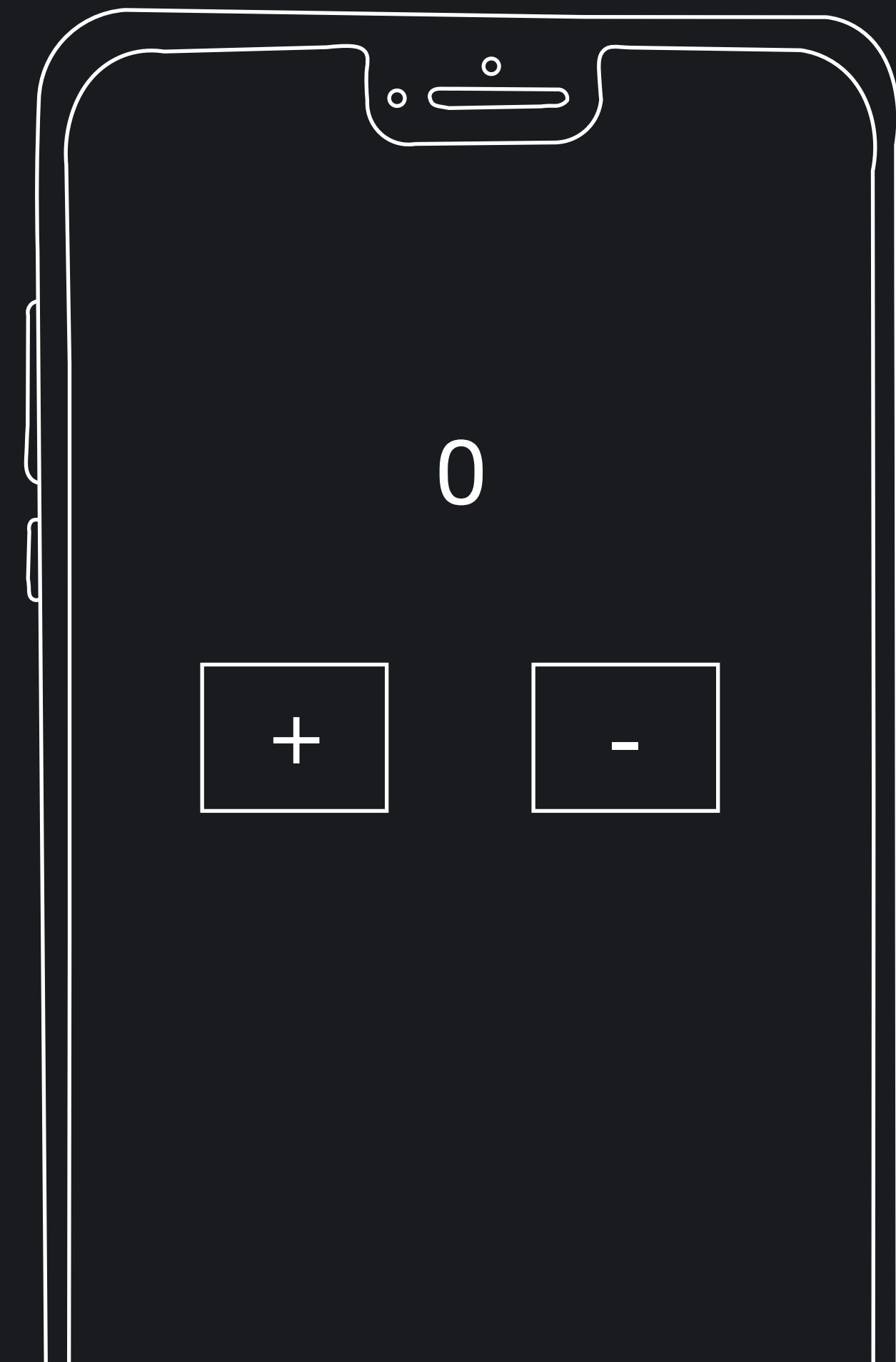
Managing State

```
Column {  
    Text(  
        text = counter.toString()  
    )  
    ...  
}
```



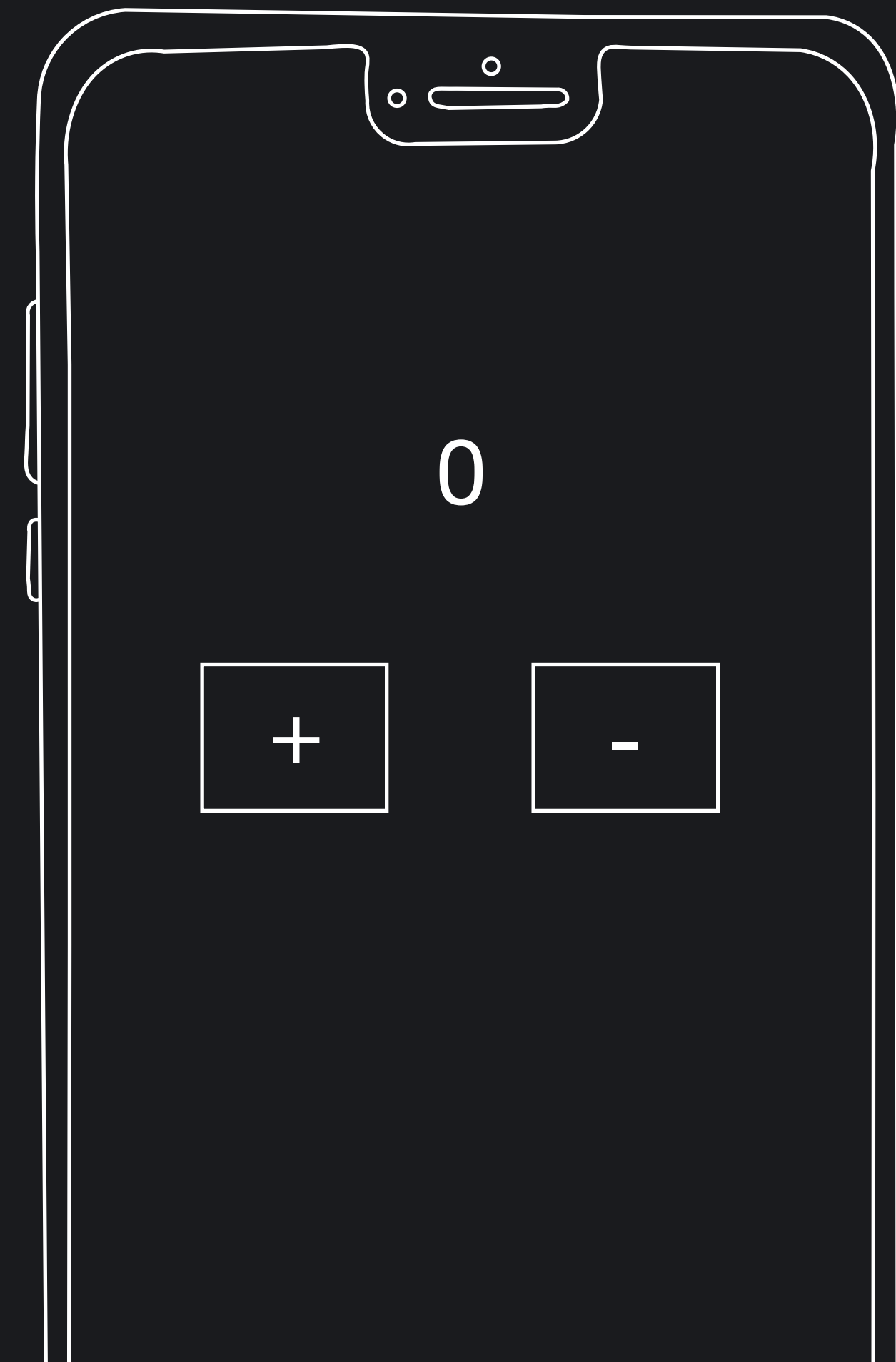
Managing State

```
Button(  
    onClick = { counter++ }  
) {  
    Text(text = "+")  
}
```



Managing State

```
Button(  
    onClick = { counter-- }  
) {  
    Text(text = "-")  
}
```

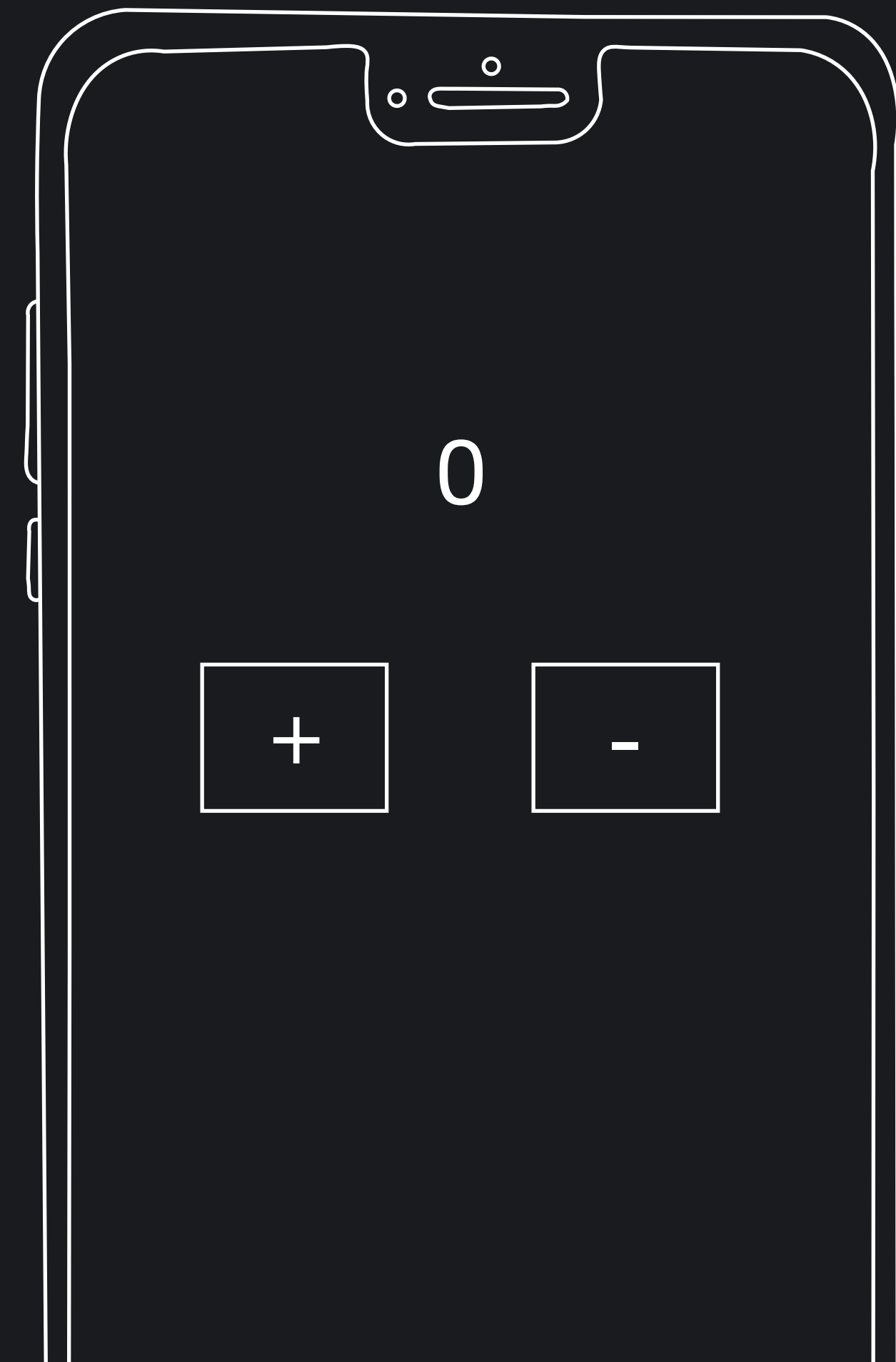


Recomposition



Managing State

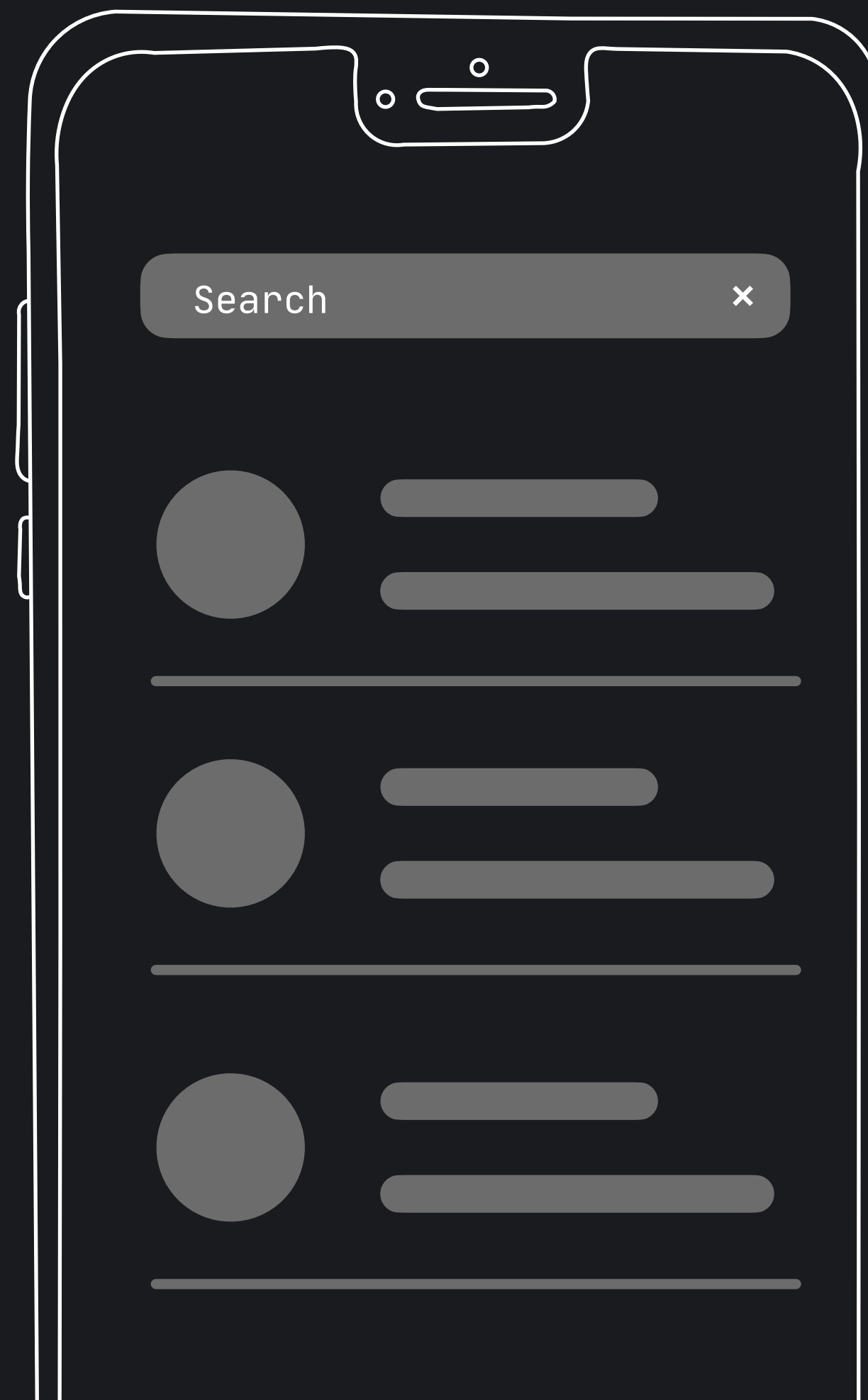
```
var counter by remember {  
    mutableStateOf(0)  
}
```



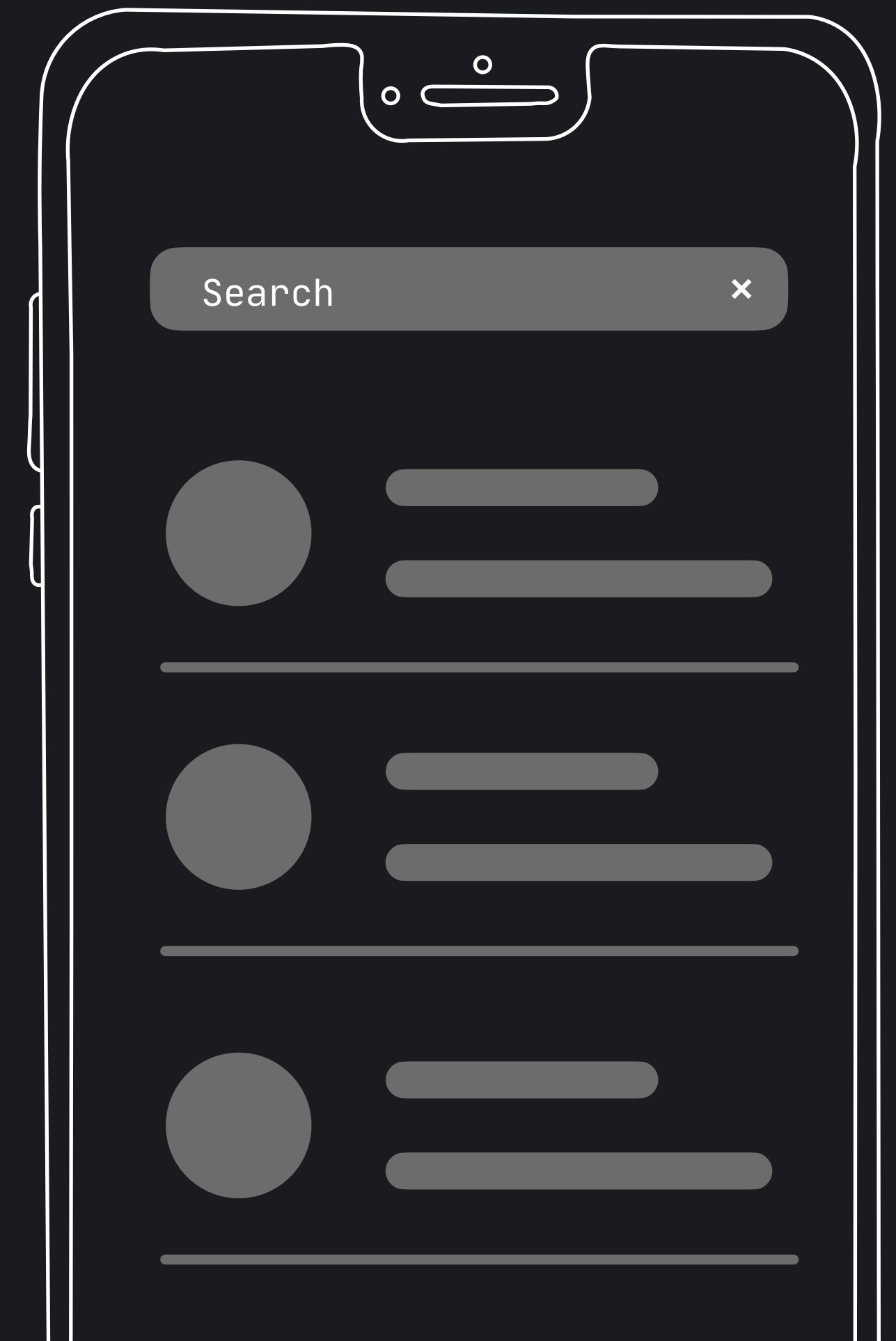
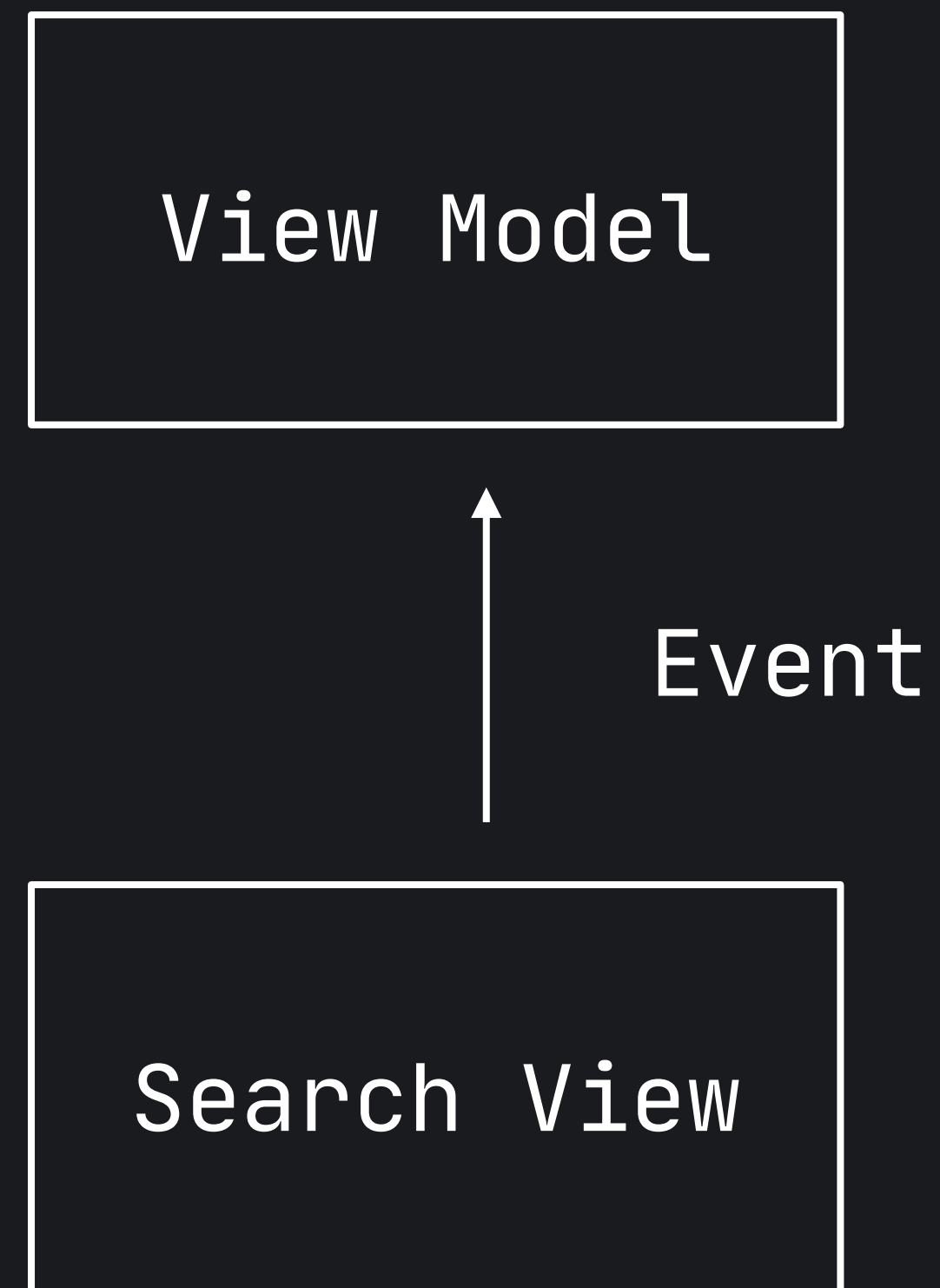
Managing State

- Setup state using *mutableStateOf*
- Setting up search

Managing State



Managing State

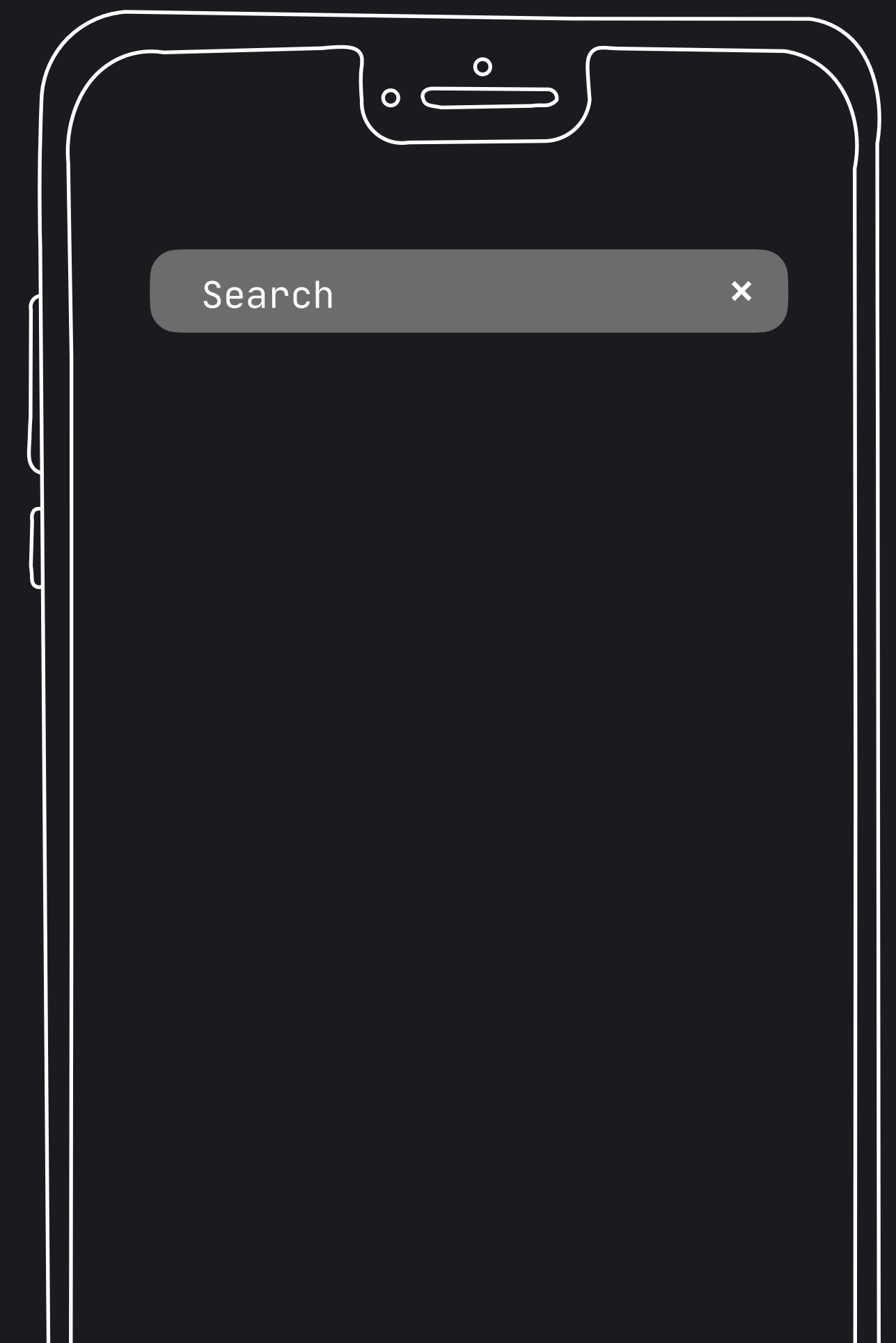


Managing State

```
@Composable
fun SearchView() {

    var query by remember {
        mutableStateOf("")
    }

}
```

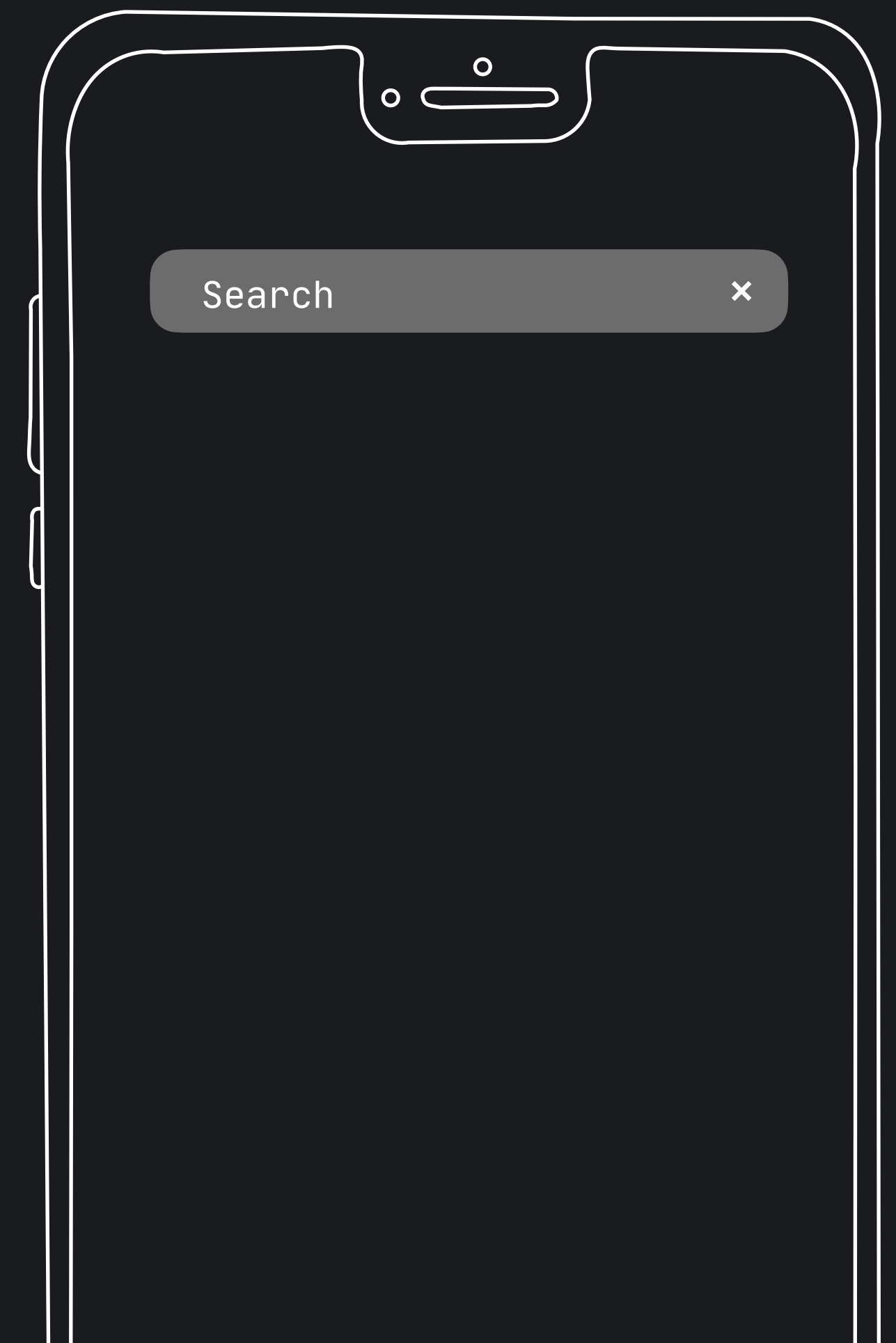


Managing State

```
@Composable
fun SearchView() {

    OutlinedTextField(
        value = query
    )

}
```

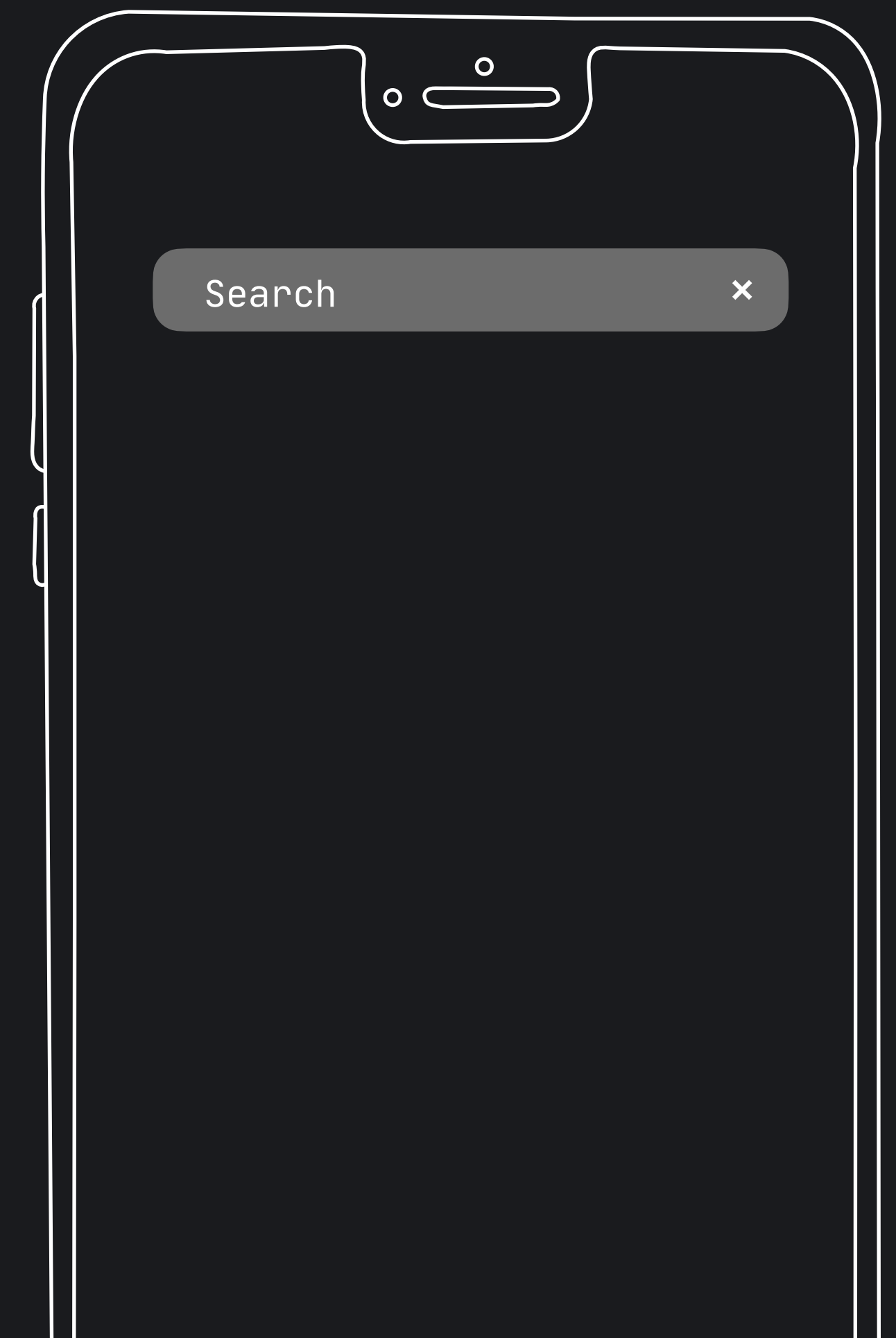


Managing State

```
@Composable
fun SearchView() {

    OutlinedTextField(
        onChange = {
            query = it
        }
    )

}
```

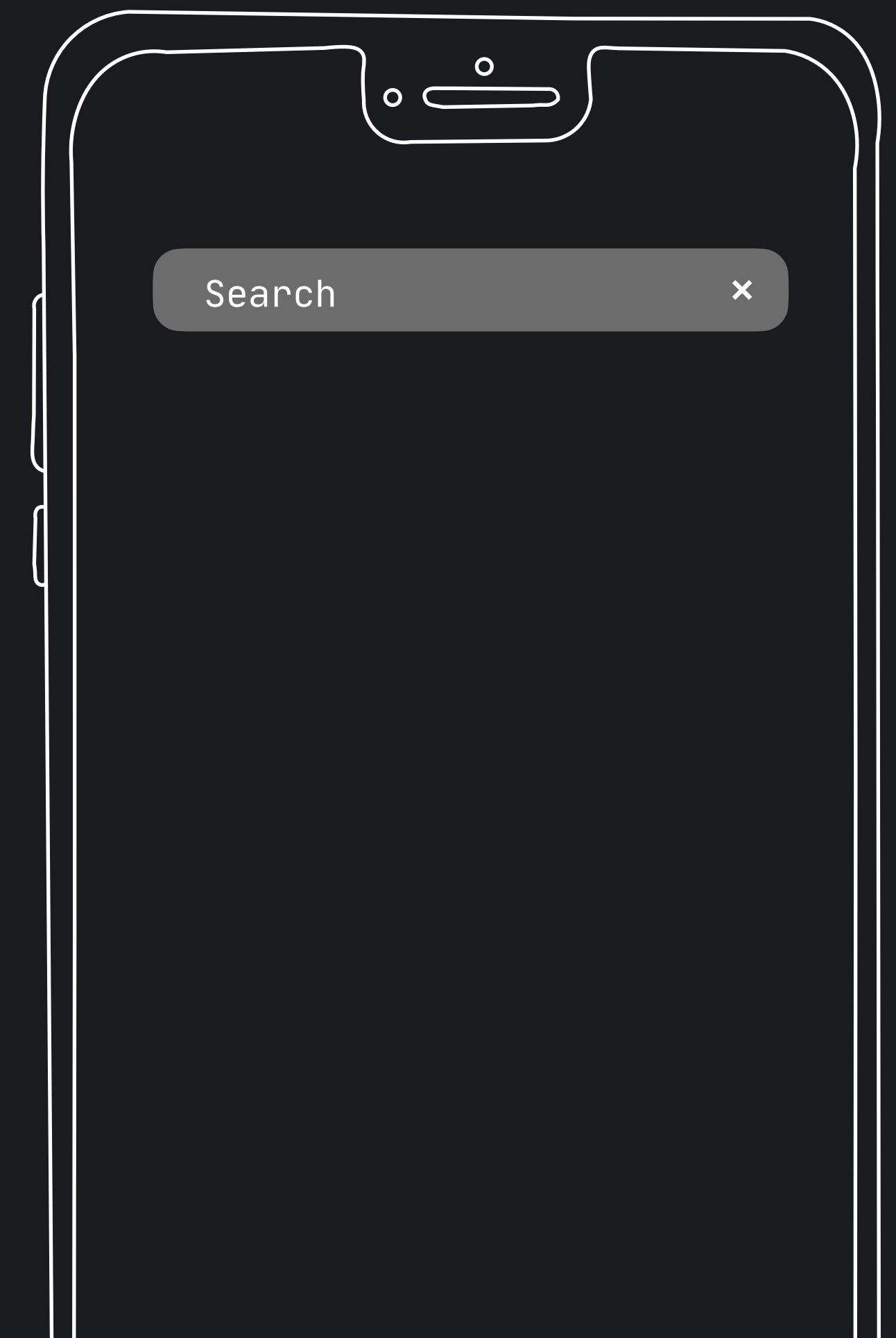


Managing State

```
@Composable
fun SearchView() {

    OutlinedTextField(
        keyboardOptions = KeyboardOptions(
            imeAction = ImeAction.Search
        )
    )

}
```



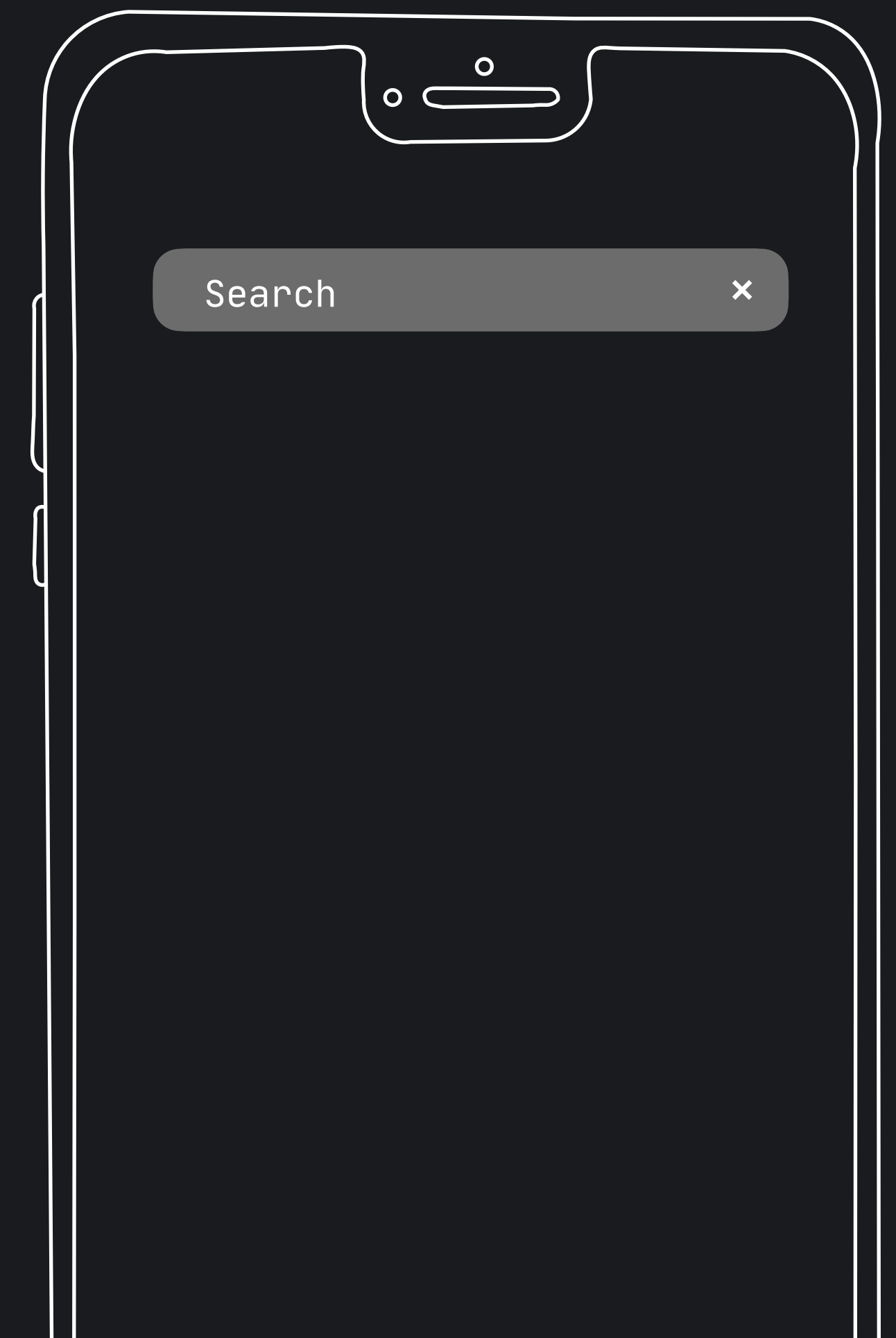
Managing State

```
@Composable
fun SearchView() {

    OutlinedTextField(
        keyboardOptions = KeyboardActions(
            onSearch = {

            }
        )
    )

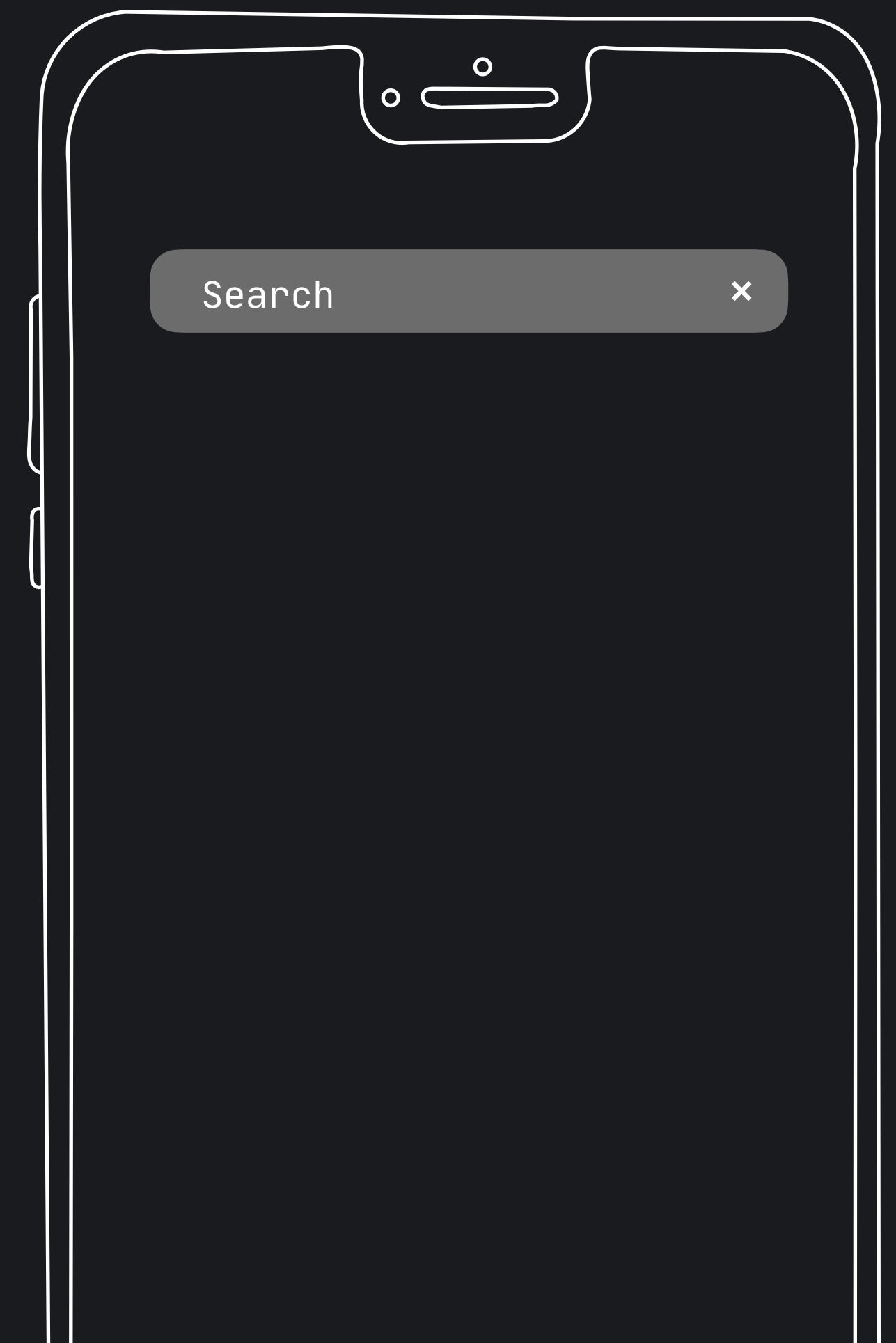
}
```



Managing State

```
@Composable
fun SearchView(onSearch: (String) → Unit) {

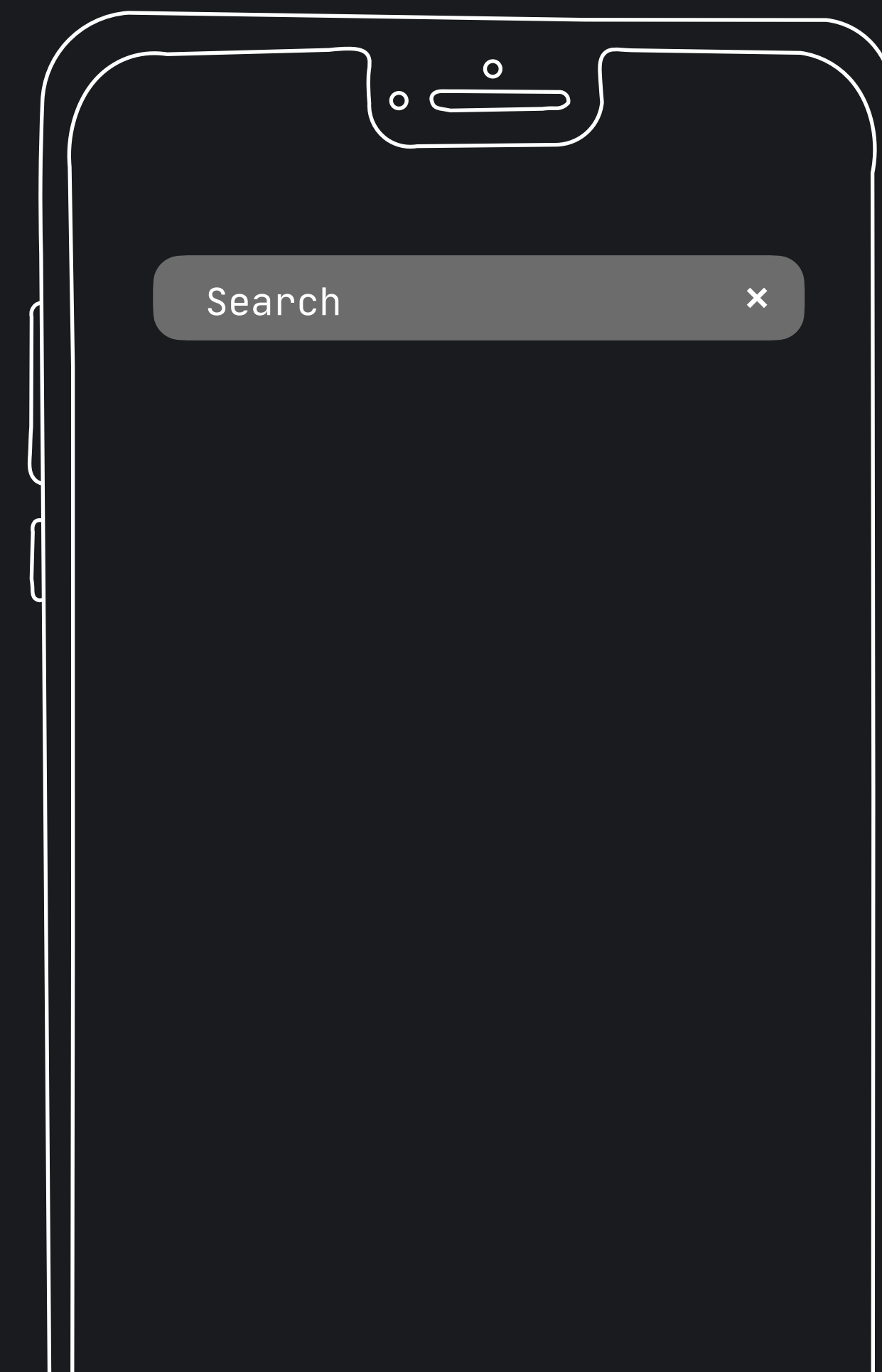
}
```



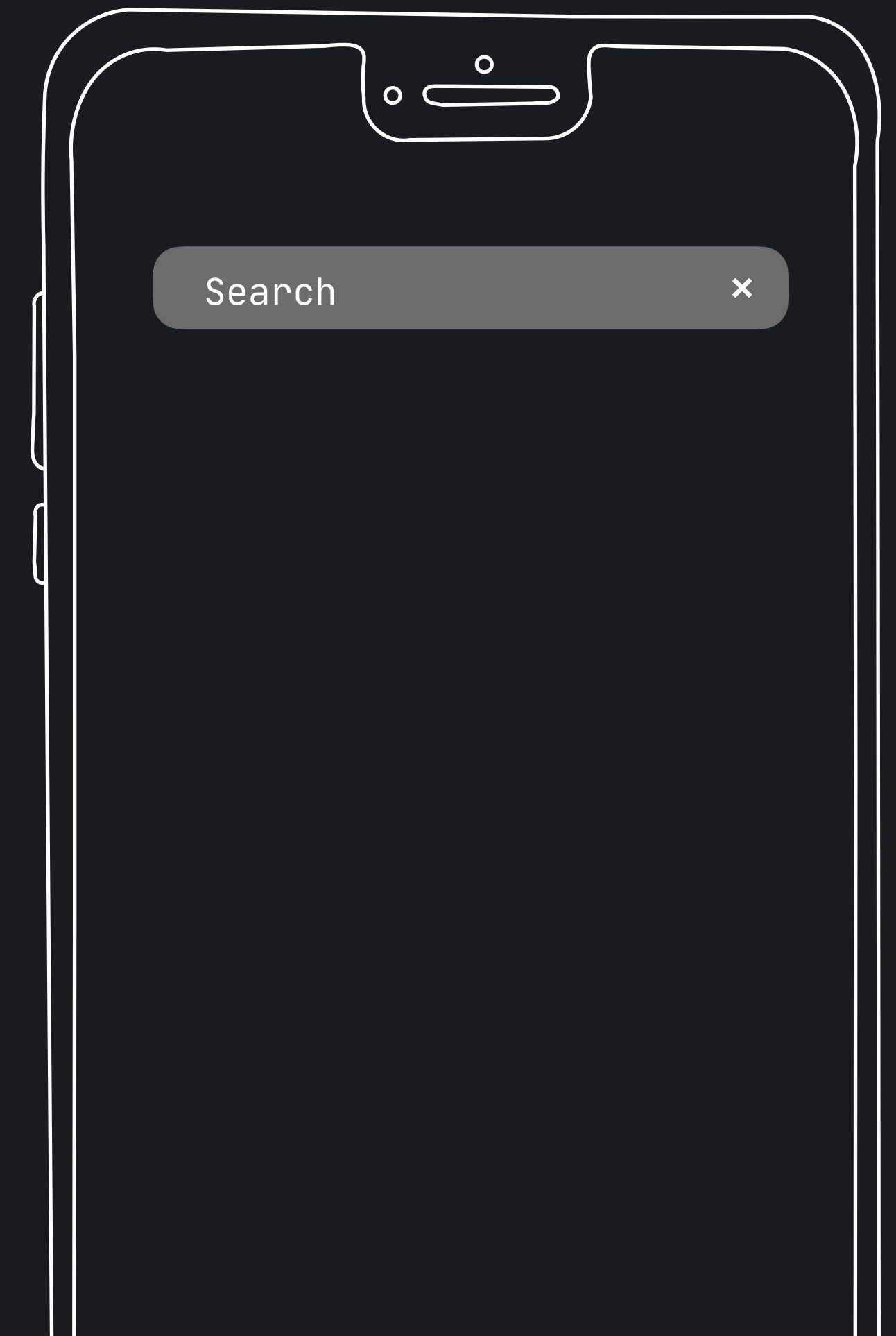
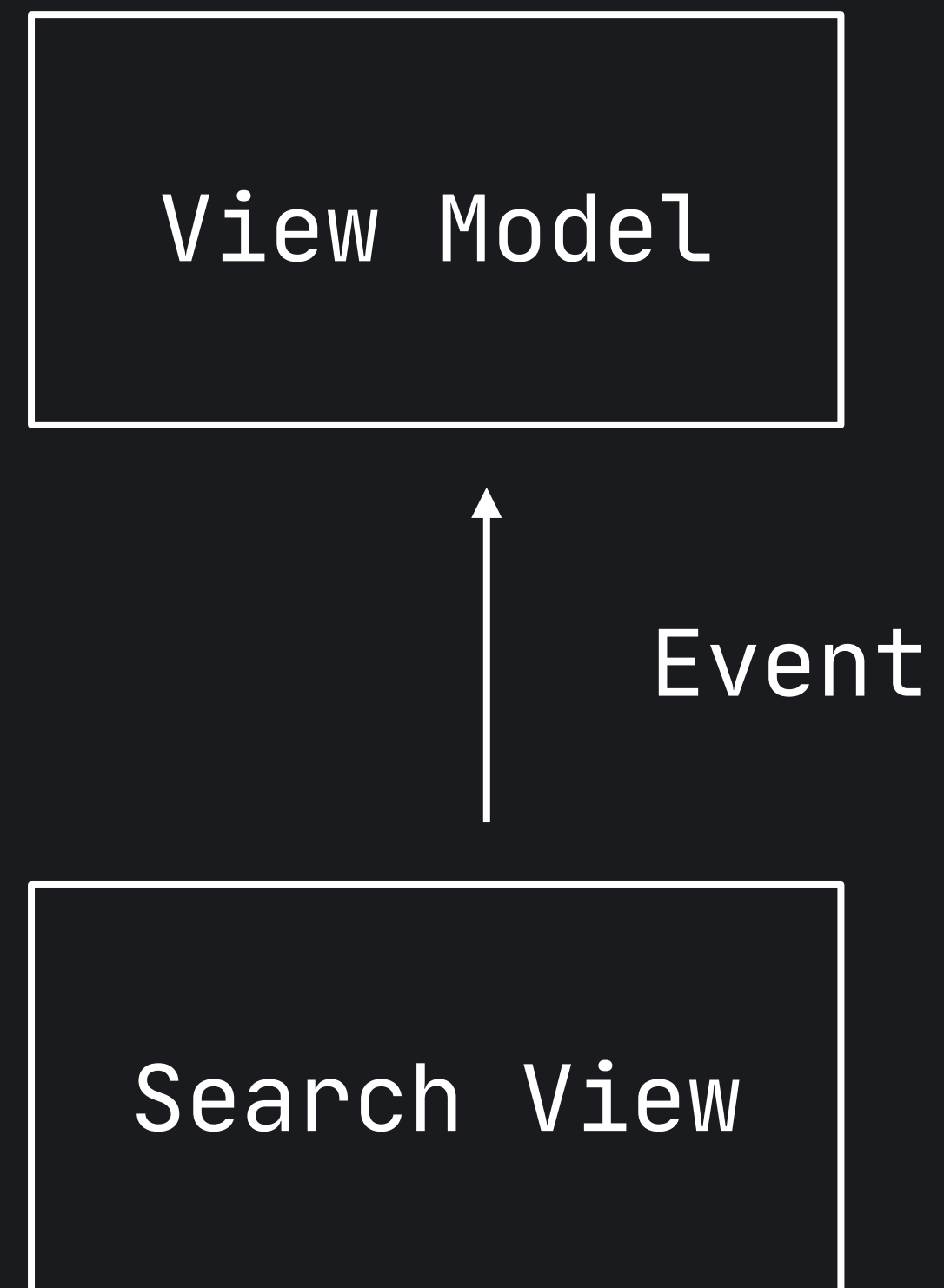
Managing State

```
@Composable
fun SearchView(onSearch: (String) → Unit) {

    OutlinedTextField(
        keyboardOptions = KeyboardActions(
            onSearch = {
                onSearch(query)
            }
        )
    )
}
```

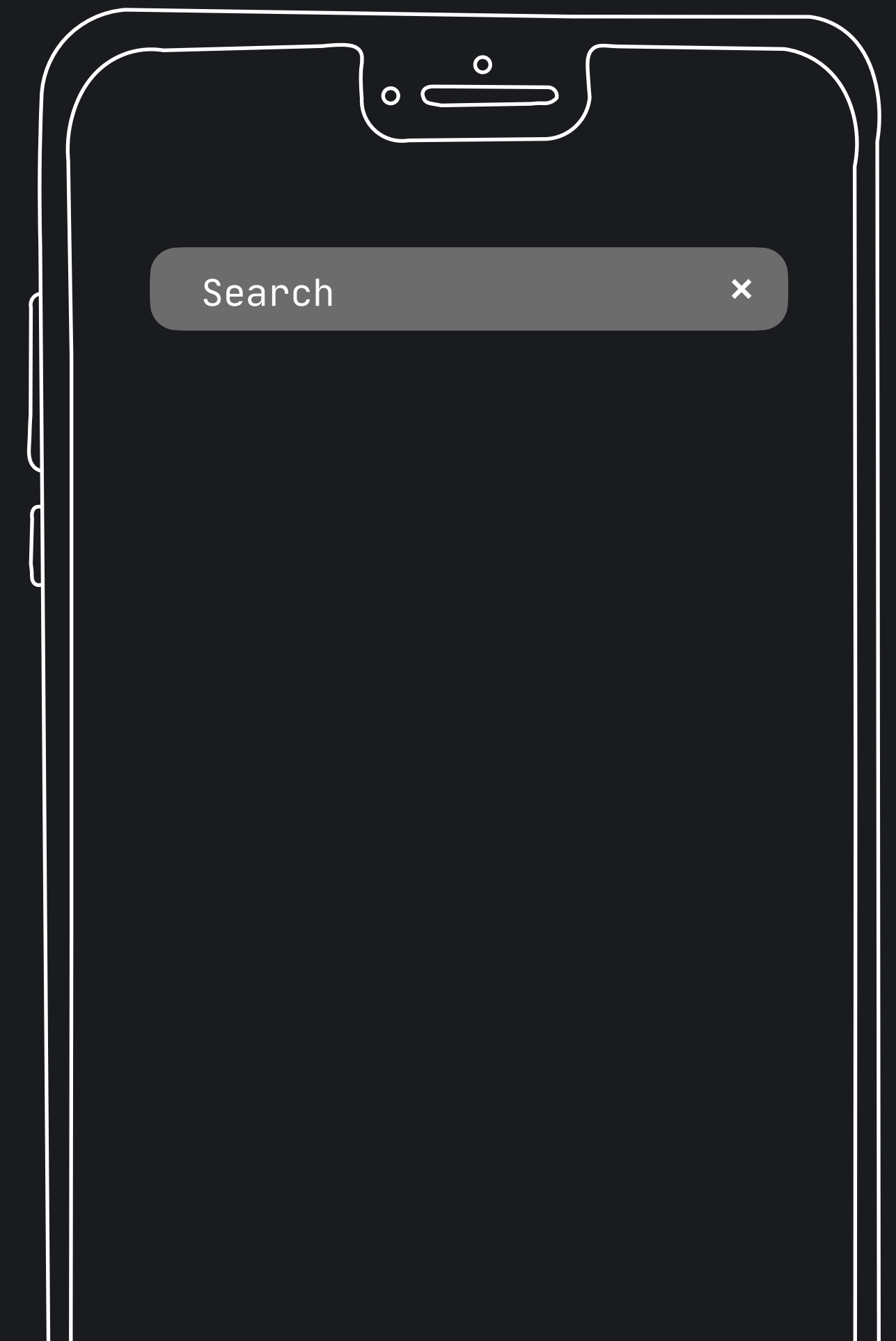


Managing State



Managing State

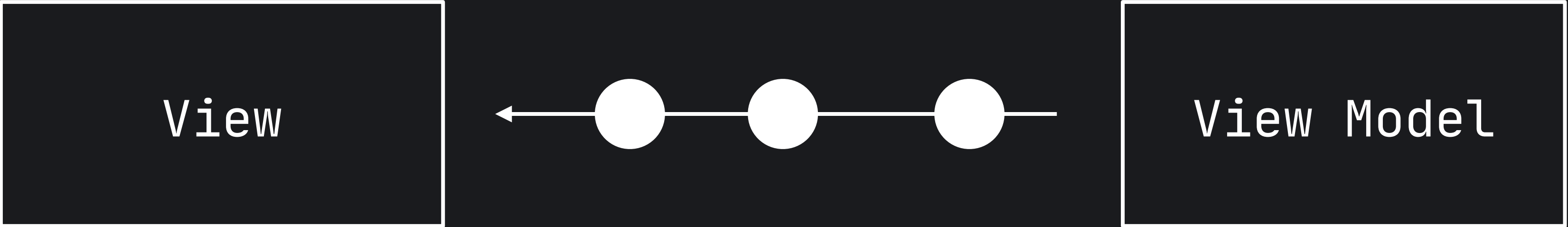
```
SearchView { query →  
    viewModel.onSearch(query)  
}
```



Managing State

- Setup state using *mutableStateOf*
- Setting up search
- Working with Flows

Flows



Flows

- *collectAsState*

Flows

```
class MyViewModel: ViewModel() {  
  
    val stateFlow = _stateFlow.asStateFlow()  
  
}
```

Flows

```
@Composable  
fun HomeView() {  
    viewModel.stateFlow.collectAsState()  
}
```

Managing State

- `flow.collectAsState`
- `LiveData.observeAsState`
- `observable.subscribeAsState`

Managing State

- Setup state using *mutableStateOf*
- Setting up search
- Working with Flows

Side Effects

What is a side effect?

- Work outside of composable function
- Open new screen when tapping button
- Show no network message

Side Effects

- Launched Effect
- Disposable Effect

Launched Effect

- Triggers on first composition or key change

Launched Effect

```
@Composable
fun HomeView() {

    var counter by remember { mutableStateOf(0) }

}
```

Launched Effect

```
@Composable
fun HomeView() {

    var counter by remember { mutableStateOf(0) }

    LaunchedEffect {
        while (true) {

        }
    }
}
```

Launched Effect

```
@Composable
fun HomeView() {

    var counter by remember { mutableStateOf(0) }

    LaunchedEffect {
        while (true) {
            delay(2000)
            counter++
        }
    }
}
```

Launched Effect

```
@Composable
fun HomeView() {

    var counter by remember { mutableStateOf(0) }

    LaunchedEffect(key1 = Unit) {
        while (true) {
            delay(2000)
            counter++
        }
    }
}
```


Disposable Effect

- Triggers on first composition or key change
- Calls `onDispose` on terminate

Disposable Effect

```
@Composable
fun HomeView() {

    DisposableEffect(...) {
        onDispose {
            callback.remove()
        }
    }

}
```

Side Effects

- Launched Effect
- Disposable Effect

Introduction to Jetpack Compose

- Thinking in Compose
- Layouts
- Managing State
- Side Effects

Thank You!

www.codingwithmohit.com

 @heyitsmohit