Chapter 4: Car Order Form Design

Passion for cars was something that I was never into growing up as a kid and into my adult life. My first car was a hand me down from my mom and 1984 Chevy Nova. Yes, I know it was nothing pretty, but it was better than walking. The first car that became something that caught my attention was the Chrysler 300. After I saw it, I became obsessed, and it was my first goal. I bought a die-cast version of this car and kept it on my desk as a goal.

Since then, I have owned two Chrysler 300's and I have never really thought that I would want another car as much as the Chrysler 300 until I sat in a Tesla. I do not own a die-cast version of this car yet, but I'll probably have one by the time this book is published. It is a goal for a few years down the road, but goals keep me focused.

When I designed the app for this chapter, I thought it would be fun to use Tesla as the car for my design. In this chapter, we are going to create a fake car-service order form. We'll learn how to take a generic form and customize it.

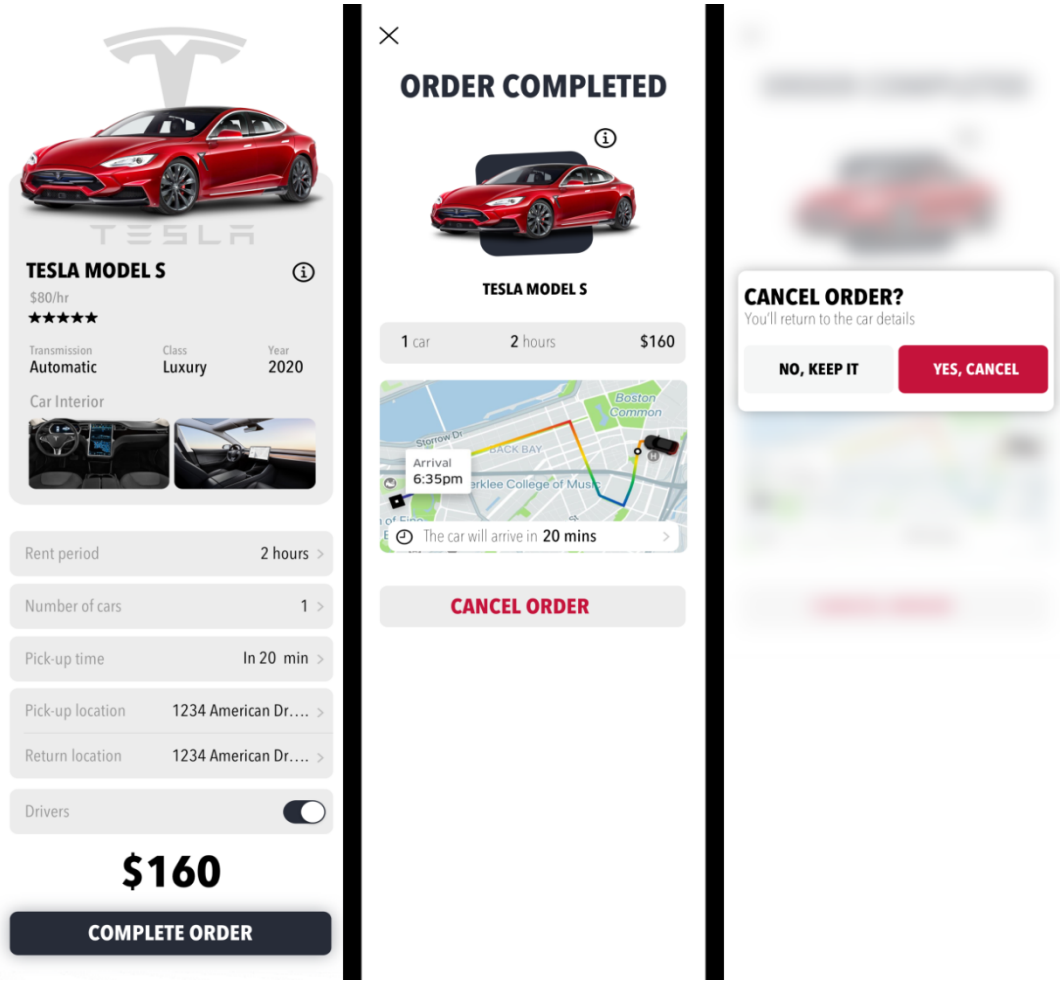
In this chapter, we'll work with the following:

- SwiftUI Custom Form Design
- Vibrancy Blur Effect

The Overall Design

In this chapter, we are going to work on a custom form design in SwiftUI. As we progress through the book, you'll notice that there are some design elements we can do in SwiftUI and others where we have to dig deeper into the framework. No matter how we have to do it in the end, you'll learn how to take a basic template and turn it into a beautiful design.

Here is what we are going to work on in the next two chapters:



B15463 0401

Our app does not contain a ton of views, but it does contain a ton of features. The features used in this app, are features you'll be able to reuse and give you the start you need for building your SwiftUI library. Let's walk through the overall structure of this app so we can better understand what we are going to do and my approach to building SwiftUI designs.

Understanding the Structure

One thing that you might notice in this book is that I like to plan how I am going to break up the design into smaller views. I find that this is the best approach before I work on an app in code. The reason I do this is because the code can get very long and overwhelming. I find it much easier to have smaller views and bring them into the main view. By doing this, it also helps me spot places where I can reuse code before I build it out. Let's retake a look at the design but let's focus on the main screen first:

TESLA

TESLA MODEL S

$80/hr

★★★★★

Transmission	Class	Year
Automatic	Luxury	2020

Car Interior

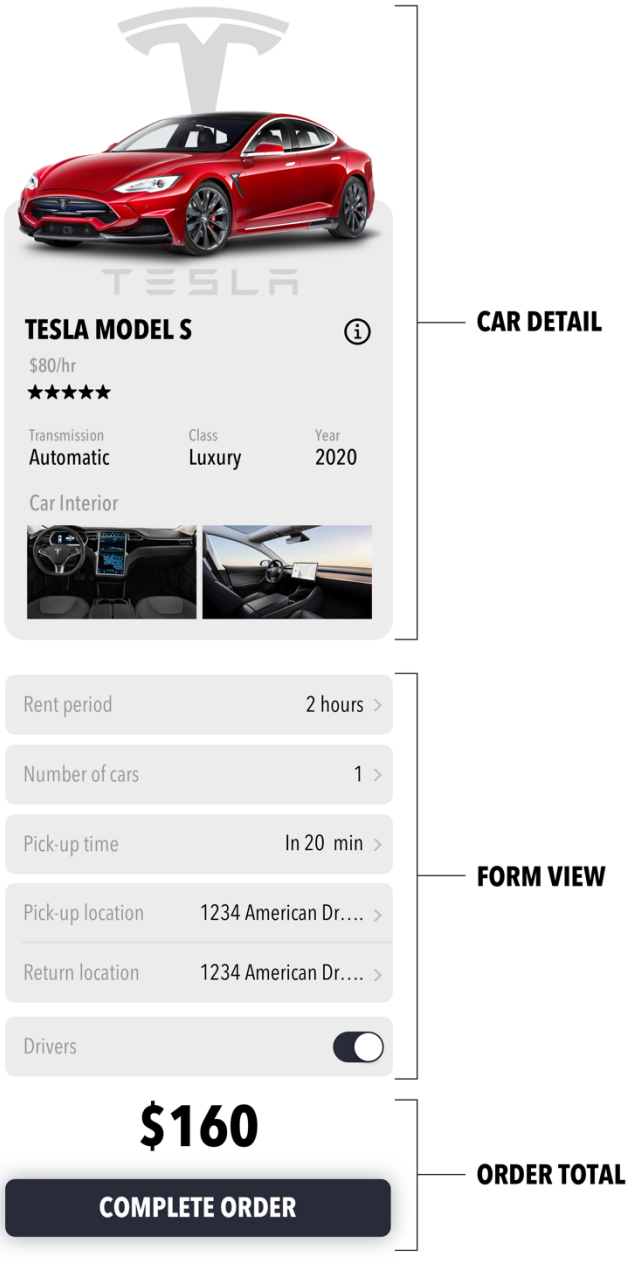
Rent period	2 hours >
Number of cars	1 >
Pick-up time	In 20 min >
Pick-up location	1234 American Dr.... >
Return location	1234 American Dr.... >
Drivers	

$160

COMPLETE ORDER

B15463 0402

In this view, I find that there are three main sections Car Detail, Form View and Order Total:

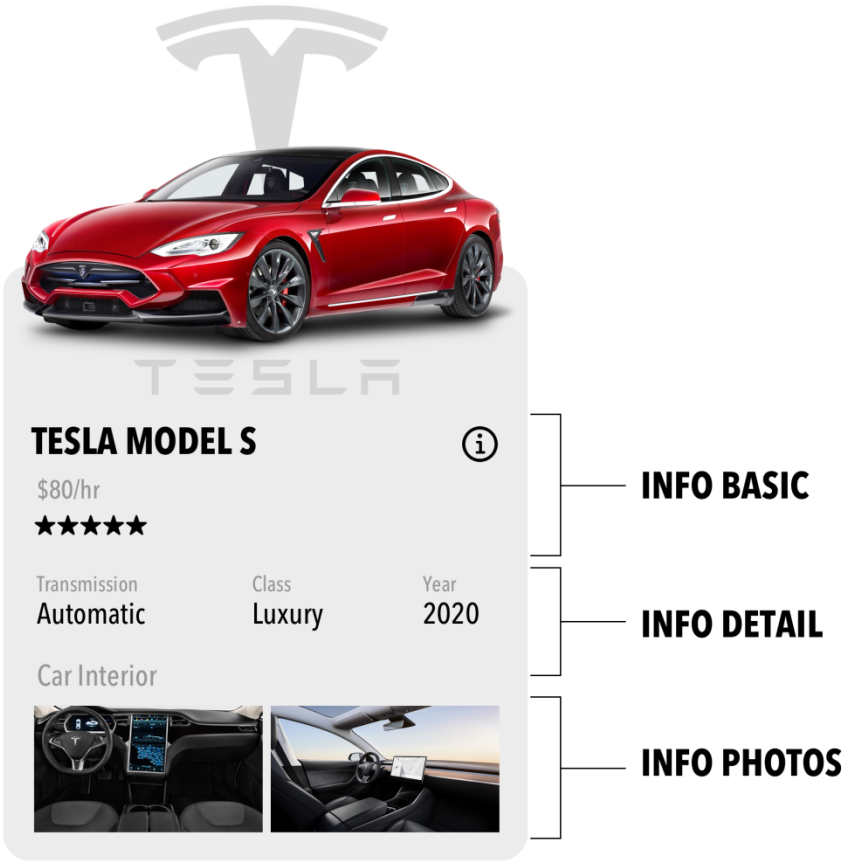


B15463 0403

These three views are a great way to separate from each other. I know that I can make them all one view, but again smaller views make it easier to work with; let's work on the Car Detail first.

Car Detail

We are going to work on the Car Detail section first. As I stated earlier, I break the views down into smaller views. I did the same for Car Detail as well. Now one thing I want to say is that breaking up the views in this section was done more for cleaner code than for it being reusable. Here is how I broke up Car Detail:



B15463 0404

We'll create three smaller views Basic, Detail, and Photos and then add them to Car Detail. Let's get started working on the Basic view first.

Car Info Basic

Now that we have an idea of what we are going to do in this chapter let's first start with the Info Basic. Open the starter project inside of the starter folder for chapter 4. Then open `CarInfoBasicView` and update Previews with the following add `pre-viewLayout`:

CarInfoBasicView().previewLayout(.fixed(width: 400, height: 100))

We have changed our Preview Layout to be a fixed size. Adding a fixed size helps me with focusing on the size instead of it being inside of a huge phone. Now, add the following code inside of the `body` variable by replacing `Text("Car Info Basic View")`:

```
VStack(alignment: .leading, spacing: 0) { // (1)
    HStack { // (2)
        Text("TESLA MODEL S")
            .customFont(.custom(.bold, 28))
        Spacer()
        Image(systemName: "info.circle")
            .font(.system(size: 28))
            .offset(y: -2)
    }

    VStack(alignment: .leading, spacing: 4) { // (3)
        Text("$80/hr")
            .customFont(.custom(.medium, 19))
            .foregroundColor(.baseDarkGray)

        HStack {
            ForEach(0..<5) { \_ in
                Image(systemName: "star.fill")
                    .font(.system(size: 15))
            }
        }
    }
}
```

Now, let's breakdown the code we just added:

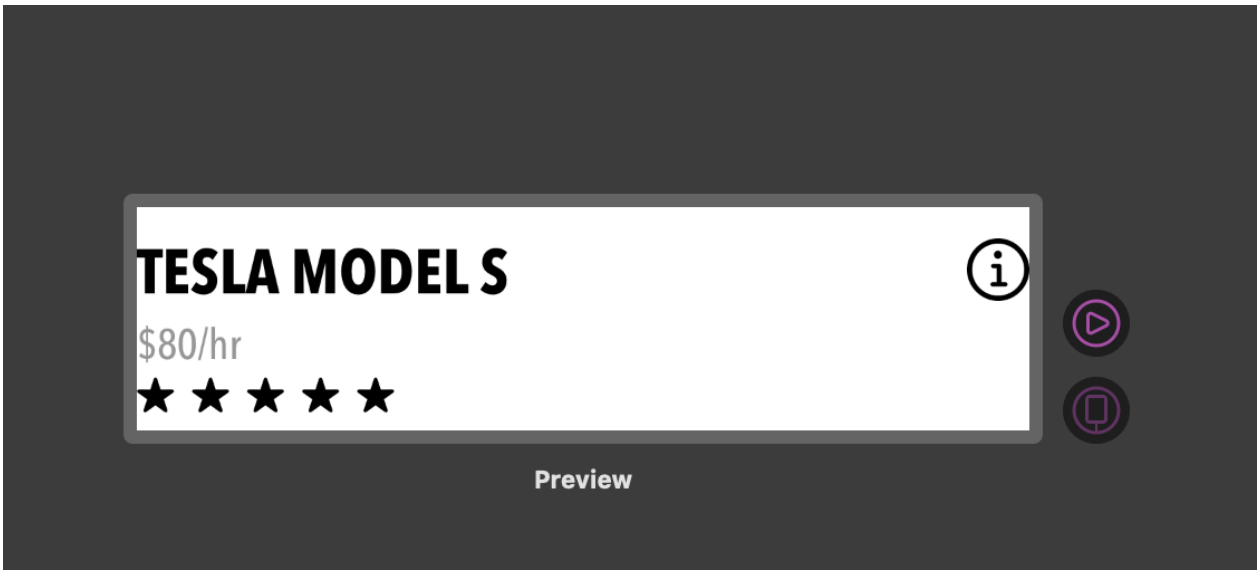
1. We are using a `VStack` is our main container with the `alignment` set to `.leading` and the `spacing` set to 0.

2. Inside of the `VStack`, we have two Stacks. Firstly, inside of the `HStack`, we have a `Text` view and an SF Symbol.
3. Finally, we have a `VStack` with another `Text` view and an `HStack` that uses a `ForEach` our Stars.

Note:

Since this is just a prototype, we are using the `ForEach`, but for a real app, you would not use the `ForEach`.

When you finish, you see the following in Previews:



B15463 0405

We are finished with `CarInfoBasicView` and we will move on to the next view `CarInfoDetailView`.

Car Info Detail

In Info Detail view, we have another simple layout. Open `CarInfoDetailView` next and update Previews with the following add `previewLayout`:

```
CarInfoDetailView()
    .previewLayout(.fixed(width: 400, height: 100))
```

Again, we are setting the `previewLayout`, with the width set to `400` and the height set to `100`. Next, let's add the following code inside of the `body` variable:

```
HStack { // (1)
    VStack(alignment: .leading) { // (2)
        Text("Transmission")
            .customFont(.custom(.medium, 16))
            .foregroundColor(.baseDarkGray)

        Text("Automatic")
        .customFont(.custom(.medium, 22))
    }

    Spacer() // (3)

    VStack(alignment: .leading) { // (4)
        Text("Class")
            .customFont(.custom(.medium, 16))
            .foregroundColor(.baseDarkGray)

        Text("Luxury")
            .customFont(.custom(.medium, 22))
    }

    Spacer()

    VStack(alignment: .leading) {
        Text("Year")
            .customFont(.custom(.medium, 16))
```

```
            .foregroundColor(.baseDarkGray)
        Text("2020")
        .customFont(.custom(.medium, 22))
    }
}.padding(.top, 15)
```

Let's breakdown the code we just added:

1. We are using a HStack as our main container, which has a top padding of 15.
2. In this `VStack` we have two text views stacked on top of each with the alignment set to `.leading`.
3. We use this `Spacer()`, so that we can add equal spacing horizontally.
4. We have another `VStack` which also has its two text views stacked just like the previous one.

When you are finished you will see the following inside of Previews:

B15463 0406

We have completed the detail section and now we can move to the photos section.

Car Info Photos

In this section, we are going to display two pictures. Open the `CarInfoPhotosView` and update Previews with the following add `previewLayout`:

```
CarInfoPhotosView()
    .previewLayout(.fixed(width: 400, height: 150))
```

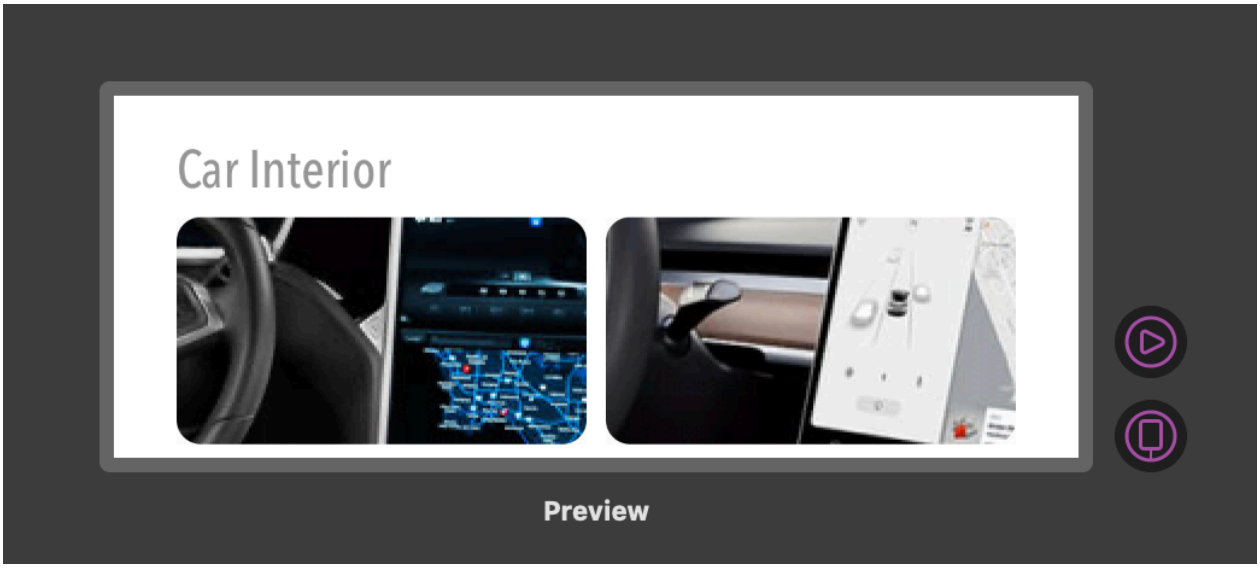
Now that we have the preview size setup add the following code inside of the `body` variable:

```
VStack(alignment: .leading, spacing: 4) { // (1)
    Text("Car Interior") // (2)
        .customFont(.custom(.medium, 22))
        .foregroundColor(.baseDarkGray)
    HStack { // (3)
        Image("pic1")
            .frame(width: 170, height: 94)
            .cornerRadius(10)

        Image("pic2")
            .frame(width: 170, height: 94)
            .cornerRadius(10)

    }.frame(height: 94)
}.padding(.top, 10)
```

Let's look at the code we just added.

1. First, we used a `VStack` has our main container with the alignment set to `.leading` and the `.spacing` set to 4.
2. Inside of the VStack, we have a Text view with a custom font.
3. Under the Text view, we are using an HStack to hold two images.

You will notice that our images are not sizing correctly, they are zoomed in instead of actually showing the entire image.



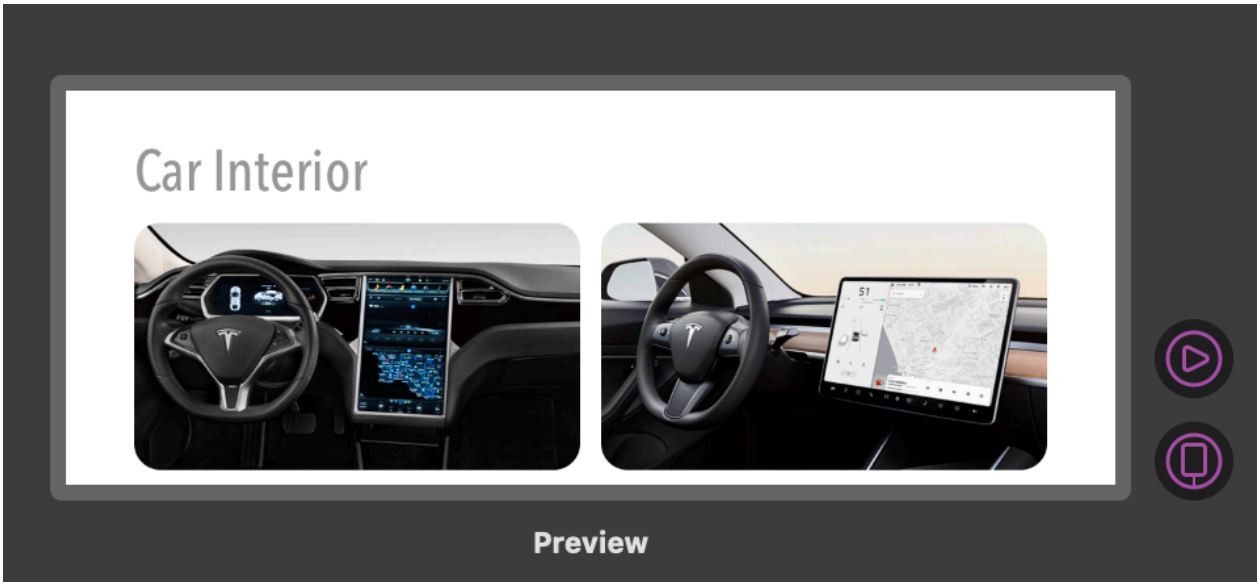
B15463 0407

We have to tell SwiftUI that these images can be resized. In order to fix this you need to add the `resizable()` property above the height. When you are done you will have the following for your images:

```
Image("pic1")
    .resizable()
    ....

Image("pic2")
    .resizable()
```

....

After you add `resizeable()` you'll see that the image scales and fits the size:



B15463 0408

Our photo section is complete let's put this all together inside of the `CarInfo-View`.

Car Info View

We are almost finished with our top section. Open the `CarInfoView` and update Previews with the following add `previewLayout`:

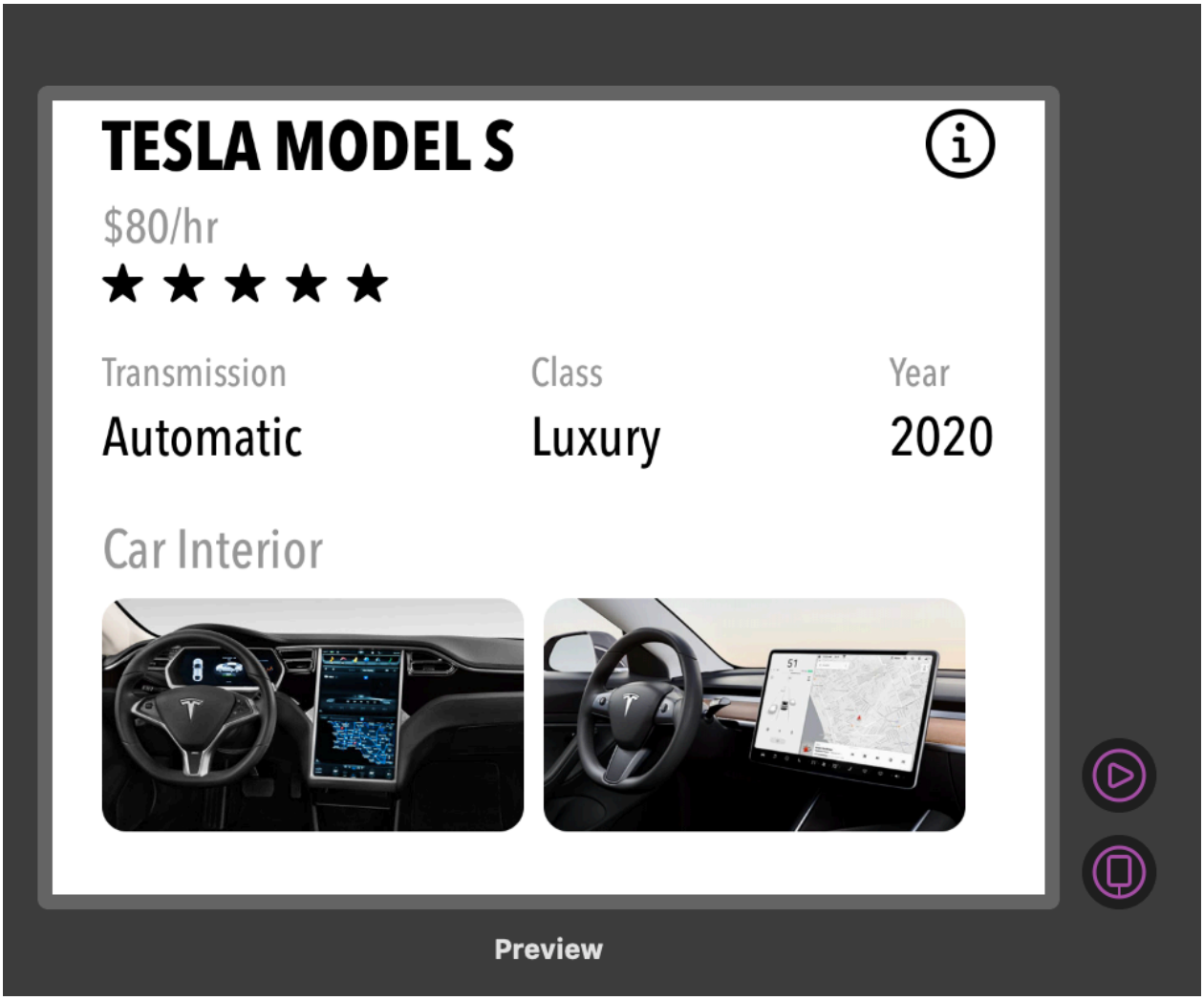
```
CarInfoView()
    .previewLayout(.fixed(width: 400, height: 320))
```

Next, add the following inside of the `body` variable:

```
VStack(alignment: .leading, spacing: 5) {
```

```
        CarInfoBasicView()
        CarInfoDetailView()
        CarInfoPhotosView()
        Spacer()
    }
    .frame(height: 320)
    .padding(.horizontal, 20)
```

When you are finished, you'll see the following inside of Previews:



We have completed the `CarInfoView` and now we are going to combine it inside of our `CarDetailView`.

Wrapping up Car Detail

The top section of our main view is almost complete. We need to update our Car Detail section and we will be finished. Open `CarDetailView` and update Previews with the following add `previewLayout`:

```
CarDetailView()
    .previewLayout(.fixed(width: 400, height: 650))
```

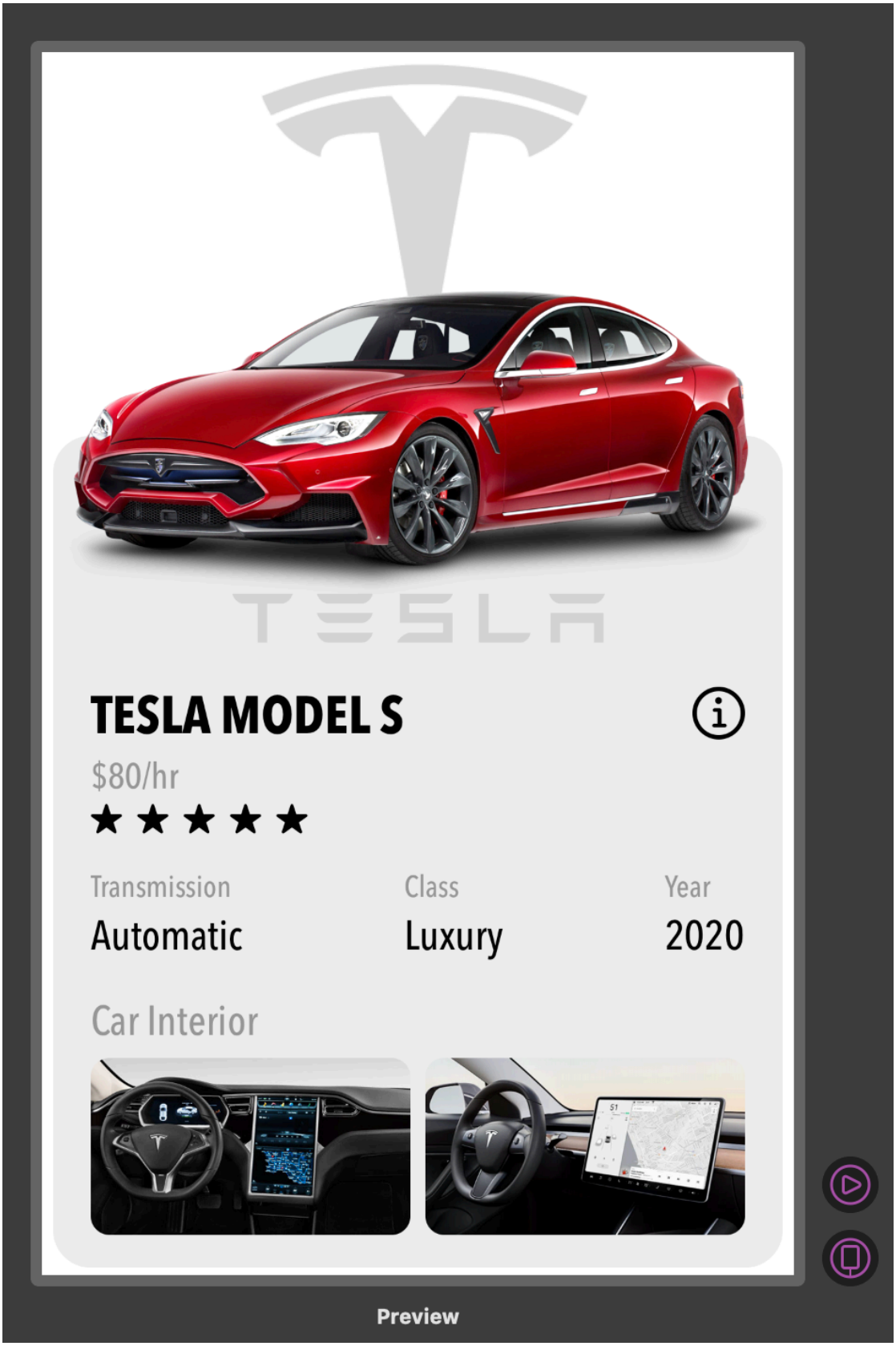
Now that we have our preview layout setup let's update the `body` with the following:

```
ZStack { // (1)
    RoundedRectangle(cornerRadius: 20) // (2)
        .fill(Color.baseGray)
        .frame(height: 442)

    VStack(spacing: 10) { // (3)
        VStack {
            Image("tesla-logo").offset(y: 120)
            Image("tesla-s")
            Image("tesla-text-logo").offset(y: -10)
        }

        CarInfoView() // (4)
    }.offset(y: -155)
}
.padding(.horizontal, 12)
.offset(y: 100)
.frame(height: 250)
```

Let's go over what we just added:

1. We use a `ZStack` as our main container. Our main container has a horizontal `padding` of 12, a y `offset` of 100 and a `height` of 250.
2. Inside of the ZStack, we put a `RoundedRectangle` in the back of the `ZStack`.
3. We have another `VStack` has a container and inside of this `VStack` we have another `VStack` with three images inside of it.
4. Finally, we add our `CarInfoView()` which we created earlier.

When you are done, you'll see the following in Previews:


TESLA
TESLA MODEL S
$80/hr
Transmission
Automatic
Class
Luxury
Year
2020
Car Interior
Preview

B15463 0410

Our top section is finally complete and we can now move to the bottom section.

Form View

In order to wrap up on main view we need to work on the bottom section which contains the form. You will learn how to customize a generic form. Since SwiftUI is new we will not be able to customize it only using SwiftUI. SwiftUI forms are built on top of UITableView, a lot of our customizations will be made to the table view itself. Let's get started by opening `FormView` and before we get started you will notice:

```
@ObservedObject var order = OrderViewModel()
```

You will see this in a few files throughout this chapter. I have added some basic defaults for this file that we will use just to get our Form working as well as some other logic we need in this app. This file and the default values are not important. We are just using it for prototyping and nothing more. In the next chapter we will rewrite this code. For now, we are going to just focus on Design in this chapter.

Inside of the `body` variable add the following:

```
VStack {
    Form {
        // Add next step here
    }
}
```

The `VStack` will be our main container and inside of this container we have our `Form` view. The form code is really long, but I separated into smaller chucks. We have a total of six sections, so we are going to start with three sections first and then add the re-

maining sections. Inside of **Form** view replace **// Add next step here** with the following:

```
Section {
    Picker(selection: $order.prototypeAmt, label: Text("Rental
period")) {
        ForEach(0 ..< order.prototypeArray.count) {
            Text("\\(self.order.prototypeArray\[$0])").tag($0)
        }
    }
}.listRowBackground(Color.baseGray)

// Add next step here
```

In the first section of our form, we are asking for the Rental period information. We are using prototype data for now, and we will replace this with real data in the next chapter.

Let's add the next section by replacing **// Add next step here** with the following:

```
Section {
    Picker(selection: $order.prototypeAmt, label: Text("Number of
cars")) {
        ForEach(0 ..< order.prototypeArray.count) {
            Text("\\(self.order.prototypeArray\[$0])").tag($0)
        }
    }
}.listRowBackground(Color.baseGray)

// Add next step here
```

Our second section is used for a Number of cars and as you can see it is exactly the same except for the label. Let's move to our third section next by replacing **// Add next step here** with the following:

```
Section {
    Picker(selection: $order.prototypeAmt, label: Text("Pick-up
time")) {
        ForEach(0 ..< order.prototypeArray.count) {
            Text("In \\(self.order.prototypeArray\[$0])
mins").tag($0)
        }
    }
}.listRowBackground(Color.baseGray)

// Add next step here
```

Pick-up time is used inside of our third section, and the code is exactly the same as the first two sections with the exception of the label being different. Let's work on the fourth section by replacing `// Add next step here` with the following:

```
Section {
    Picker(selection: $order.prototypeAmt, label: Text("Pick-up
location")) {
        ForEach(0 ..< order.prototypeArray.count) {
            Text("\\(self.order.prototypeArray\[$0])").tag($0)
        }
    }

    Picker(selection: $order.prototypeAmt, label: Text("Return
location")) {
        ForEach(0 ..< order.prototypeArray.count) {
            Text("\\(self.order.prototypeArray\[$0])").tag($0)
        }
    }
}
.listRowBackground(Color.baseGray)

// Add next step here
```

In the fourth section, we have two pickers one for Pick-up location and Return location. The only difference between the first three sections and this one is that we have two pickers in one section.

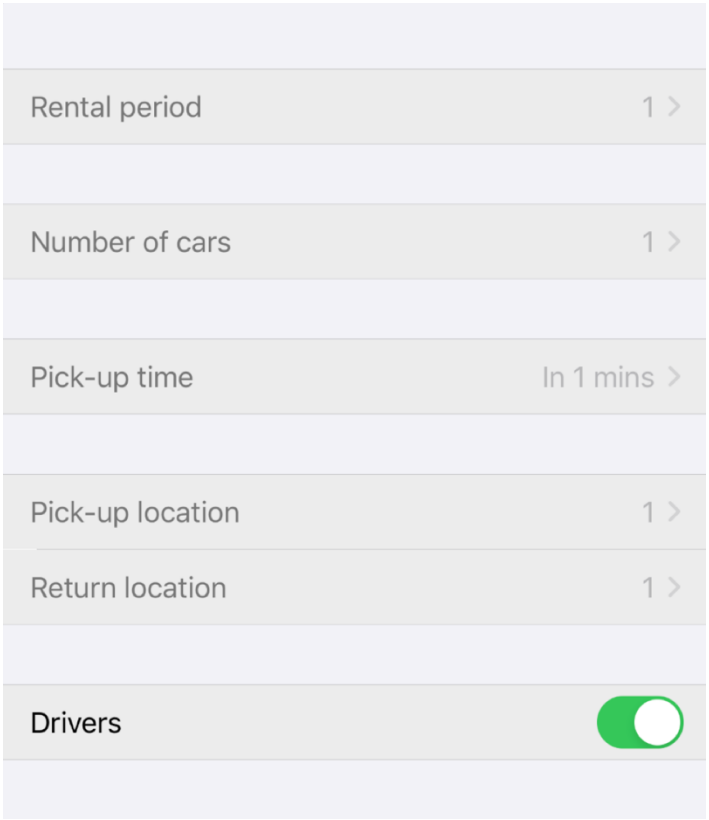
Sections in forms allow us to group similar data together. Design wise you could have all of them inside of one section but visually I decided to go with this layout. As I stated earlier, we will cover the data side of our `Picker` views in the next chapter but let me just explain them a bit more.

Under the `selection` we use this variable to keep track of selected variables. Inside of the `Picker`, we are using a `ForEach` and we loop through our prototype array and create a `Text` view. Since we have our `Picker` inside of the Section, our `Picker` displays in a detail view. Design wise we are using `listRowBackground(Color.baseGray)` which allows us to add a gray background color to our sections.

We have one more section to cover within the form. Let's add the final one in the form area by replacing `// Add next step here` with the following:

```
Section {
    Toggle(isOn: $order.prototypeBoolean) {
        Text("Drivers")
    }
}.listRowBackground(Color.baseGray)

// Add last section here
```

Inside of this section, we are using a toggle that will keep track if we need drivers or not. Our data is driven from prototype data and will be updated in the next chapter. We just added five pickers and one toggle, let's see what this looks like inside of Previews:



B15463 0411

Our second section of our design is done and now we just need to work on the final section the Order Total. Let's work on that now.

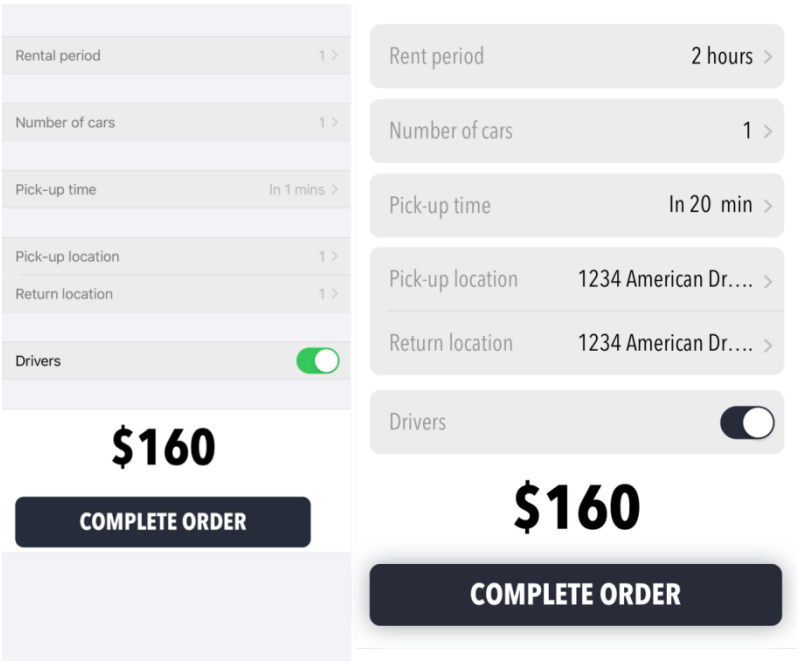
Order Total

We have one last section to add to our form and in this section, we just need to add a Text view and a button. Let's add the last section by replacing `// Add last section here` with the following:

```
Section {
    VStack {
        Text("$160")
```

```
            .customFont(.custom(.bold, 60))
            .padding(.bottom)

        Button(action: {}) {
            Text("COMPLETE ORDER")
        }
        .customFont(.custom(.bold, 28))
        .frame(minWidth: 0, maxWidth: .infinity)
        .frame(height: 60)
        .foregroundColor(.white)
        .background(Color.baseGreen)
        .cornerRadius(10)
    }
}
```

In this last section we added a custom Text view which will display our total and a button to complete the order. We now have a basic form but let's look at how we can update our form to match the design. Let's look at them side by side:



We have a few things we need to do in order to match the design. The one that stands out the most is making our form be grouped with rounded corners and not fully extend to the edge. We just added our sections into the `Form` view. Here is what it looks like with the Form section code hidden:

```
var body: some View {
    VStack {
        Form {
            // Section code is here
        } // Add attribute here
    }
}
```

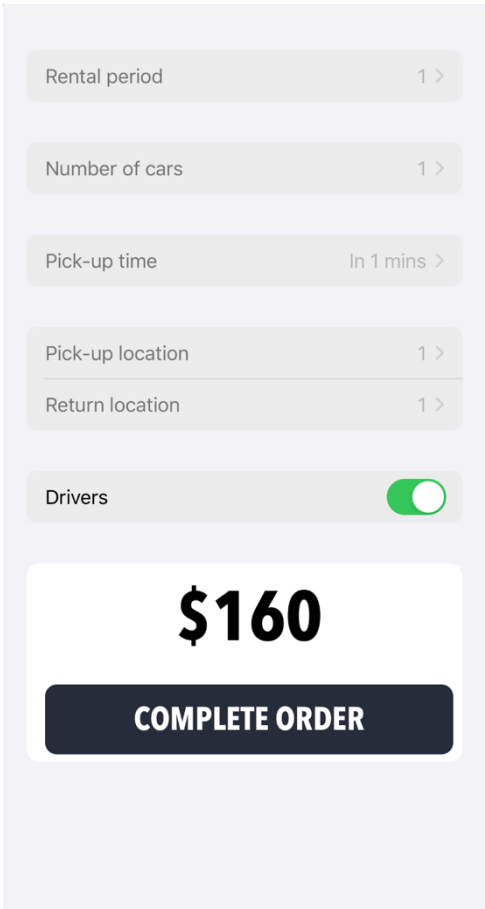
Add the following code to the closing curly brace for the Form by replacing `\\ Add attribute here` with:

```
.environment(\.horizontalSizeClass, .regular)
```

When you are finished your code should look like the following:

```
var body: some View {
    VStack {
        Form {
            // Section code is here
        }.environment(\\.horizontalSizeClass, .regular)
    }
}
```

After you add this you in Previews you will see that we now have a grouped form:

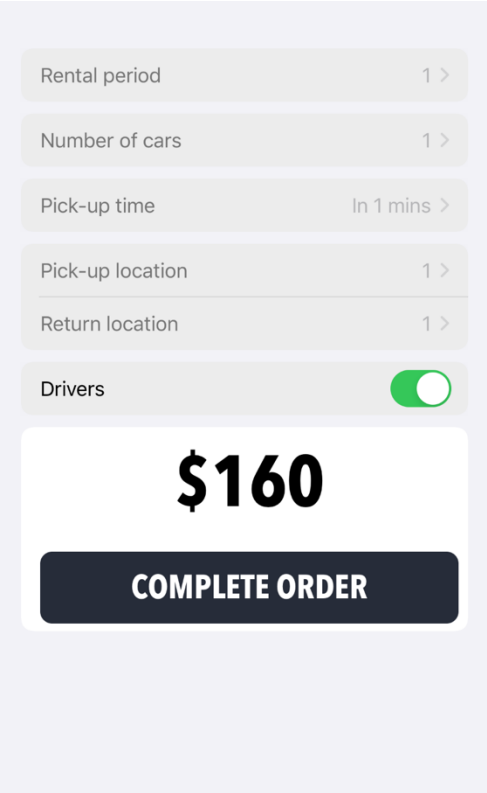


B15463 0413

The overall design is still missing a few things. Our spacing in between each section doesn't match, the complete order button and $160 text view is inside of a white box and finally our `Switch` is defaulting to a green color. Let's fix the spacing by adding the following function above the `body` variable:

```
init() {
    UITableView.appearance().sectionHeaderHeight = 0
    UITableView.appearance().sectionFooterHeight = 10
    // Add next step
}
```

When you are done you will see that are spacing is now fixed:

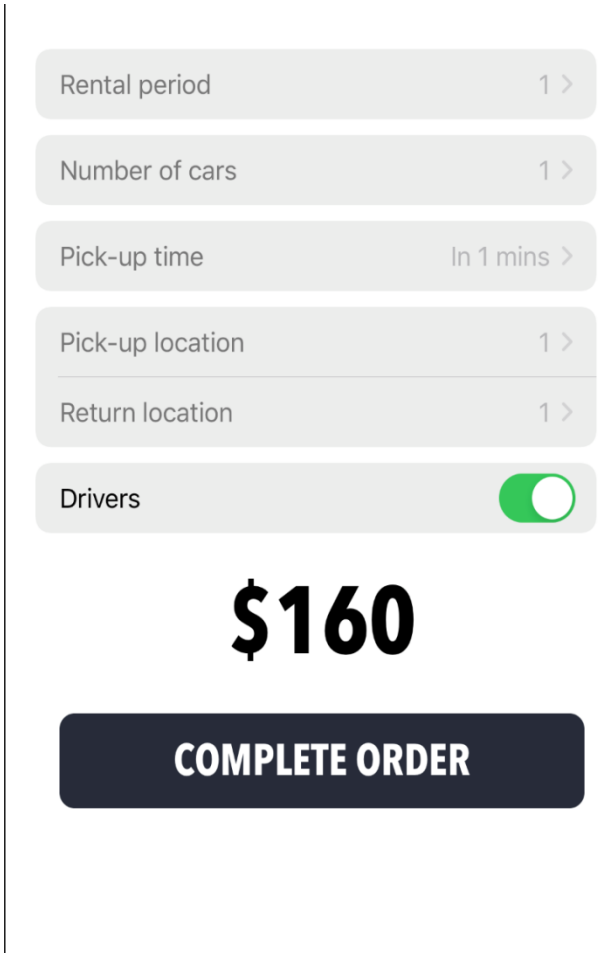


B15463 0414

Let's fix the white background by getting rid of the background color by replacing `// Add next step` with the following code:

```
UITableView.appearance().backgroundColor = .clear
// Add next step here
```

You will now see the following in Preview:

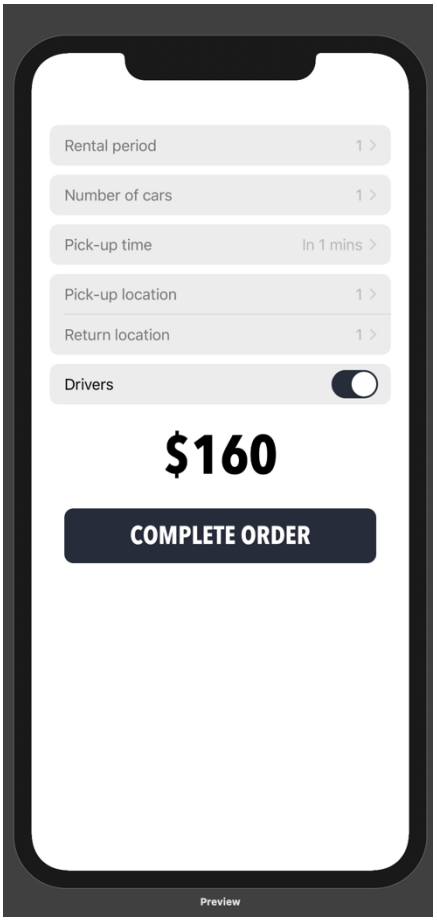


B15463 0415

Finally, we just need to update our `Switch` to have a blue tint instead of a green one. Replace `// Add next step here` with the following code:

```
UISwitch.appearance().onTintColor = UIColor.baseGreen
```

You will see in Previews that the `Switch` tint color has changed.



B15463 0415b

Now that our `Form` is setup, let's take a break from the design for a minute and take some time to look at Picker views and all of the different types we can use.

Pickers

In the form we just completed, we used the picker that goes to a detail view with a list of items. There are three other pickers I would like to cover before moving on. We are not using these pickers but I wanted to cover them regardless.

Wheel Picker

If you wanted to use the traditional `WheelPicker` you can add `.pickerStyle(WheelPickerStyle())` to the Picker. The code would look like the following:

```
Picker(selection: $order.prototypeAmt) {
    ForEach(0 ..< order.prototypeArray.count) {
        Text("\\(self.order.prototypeArray\[$0])").tag($0)
    }
}.pickerStyle(WheelPickerStyle())
```

This picker is similar to what we added in our form section expect we removed the label and added a `.pickerStyle()`. The Wheel picker is the traditional way that you would use a picker in UIKit. Next, let's look at a Segmented picker.

Segmented Picker

A Segmented picker would be a Segmented controller we use in UIKit. In order to display a Segmented picker we would do the following:

```
Picker(selection: $order.prototypeAmt, label: EmptyView()) {
    ForEach(0 ..< order.prototypeArray.count) {
        Text("\\(self.order.prototypeArray\[$0])").tag($0)
    }
}.pickerStyle(SegmentedPickerStyle())
```

Here, we are setting the label to an `EmptyView()` which is just blank view is SwiftUI. `EmptyViews` are great for prototyping as you can use them when you want to display something there but have not created the view. By changing the `.pickerStyle` to

`SegmentedPickerStyle()` you now have a segmented controller. Finally, let's look at a Date picker:

Date Picker

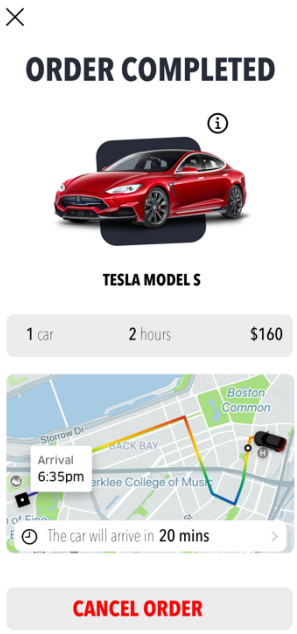
The `Date Picker` has a bit more but here is how you would create a `DatePicker`:

```
var dateFormatter: DateFormatter {
    let formatter = DateFormatter()
    formatter.dateStyle = .long
    return formatter
}

@State private var selectedDate = Date()

var body: some View {
    VStack {
        DatePicker(selection: $selectedDate, in: Date()...,
displayedComponents: .date) {
    Text("")
    }.padding(30)

    Text("Selected Date is \\(selectedDate, formatter:
dateFormatter)")
    .font(.title)
    }
}
```

Overall, the setup is similar to the other pickers we created earlier. The main different is we are using a date and a date formatter. In the above code, we have a date picker and when you select a date it gets displayed inside of the `Text` view below it. Now that we have taken a minute to look at the different pickers take a few minutes and play around with them. When you are done, move on to the next section.

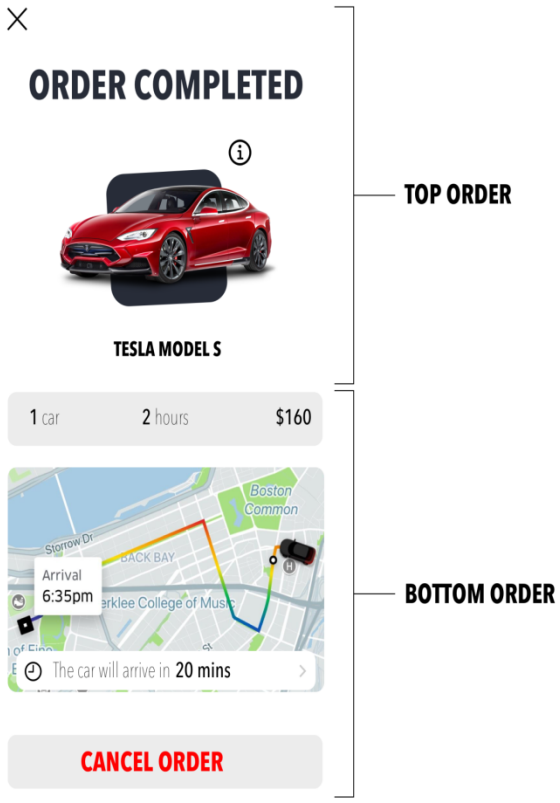
Complete Order Design

Let's get back to designing our app. We are moving to the Complete Order View; this view appears when you tap on the Complete Order Button. This view will be a modal that lays on top of our main view.



B15463 0416

We'll break this view up into two sections Top Order and Bottom Order:



B15463 0417

We could break this up into more for example, if we were using MapKit here we could create a view that would have all the code for displaying the map and the arrival time. Since we are just displaying an image we are keeping it simple. We are going to do a few more screens together and at the end of this chapter you will get to try it out on your own.

Let's work on the Order view by starting with the Top Order View.

Top Order View

Now that we have a better idea of the direction, let's work on `TopOrderView`. Open `TopOrderView` and update Previews with the following add `previewLayout`:

```
TopOrderView()
```

```
    .previewLayout(.fixed(width: 400, height: 450))
```

Next inside of the **body** section replace **Text("Top Order View")** with the following:

```
VStack { // (1)
    HStack {
        Button(action: {}) {
            Image("close-btn")
        }
        Spacer()
    }
    .padding(.bottom, 10)
    .buttonStyle(PlainButtonStyle())


    Text("ORDER COMPLETED")
        .customFont(.custom(.bold, 42))

    // Add next step here
}
```

In this view, we are using a **VStack** as our main container. Inside of the **VStack**, we added our Close button and a Text view which displays "Order Completed". Now we are going to display a car under the Text along with the car name. Replace **// Add next step here** with the following code:

```
VStack(spacing: 0) {
    HStack {
        Spacer()

        Image(systemName: "info.circle")
            .font(.system(size: 28))
            .offset(x: -30)
    }
```

```
    ZStack {
        Image("car-bg-shape")
        Image("tesla-s")
            .resizable()
            .frame(width: 268, height: 125)
    }

    Text("TESLA MODEL S")
        .customFont(.custom(.bold, 21))
        .padding(.top, 30)
}
.frame(width: 268)
.padding(.bottom, 20)
```

In the last container we have a VStack, and inside of that container we use a ZStack that container to stack the car and the square shape behind it. When you are finished, you'll see the following:

B15463 0418

Now that the `TopOrderView` is finished we can move to the `BottomOrderView`.

Bottom Order View

Next up is the Bottom Order view, open `BottomOrderView` and update Previews with the following add `previewLayout`:

```
BottomOrderView()
    .previewLayout(.fixed(width: 400, height: 400))
```

Next, replace `// Bottom Order View` and add the following:

```
VStack {
    HStack {
        HStack(spacing: 4) {
            Text("1")
                .customFont(.custom(.medium, 22))
            Text("car")
                .customFont(.custom(.ultralight, 22))
        }

        Spacer()

        HStack(spacing: 4) {
            Text("2")
                .customFont(.custom(.medium, 22))
            Text("hours")
                .customFont(.custom(.ultralight, 22))
        }

        Spacer()

        HStack(spacing: 4) {
            Text("$160")
                .customFont(.custom(.medium, 22))
        }
    }
    .padding(.horizontal, 15)
    .frame(height: 55)
    .frame(minWidth: 0, maxWidth: .infinity)
    .background(Color.baseGray)
    .cornerRadius(10)

    // Add next step here
}
```

We have completed the header for our Bottom Order View. We are using the Spacer() so that our text is evenly distributed inside of the view. Let's work on the Map part next. Add the following by replacing `// Add next step here`:

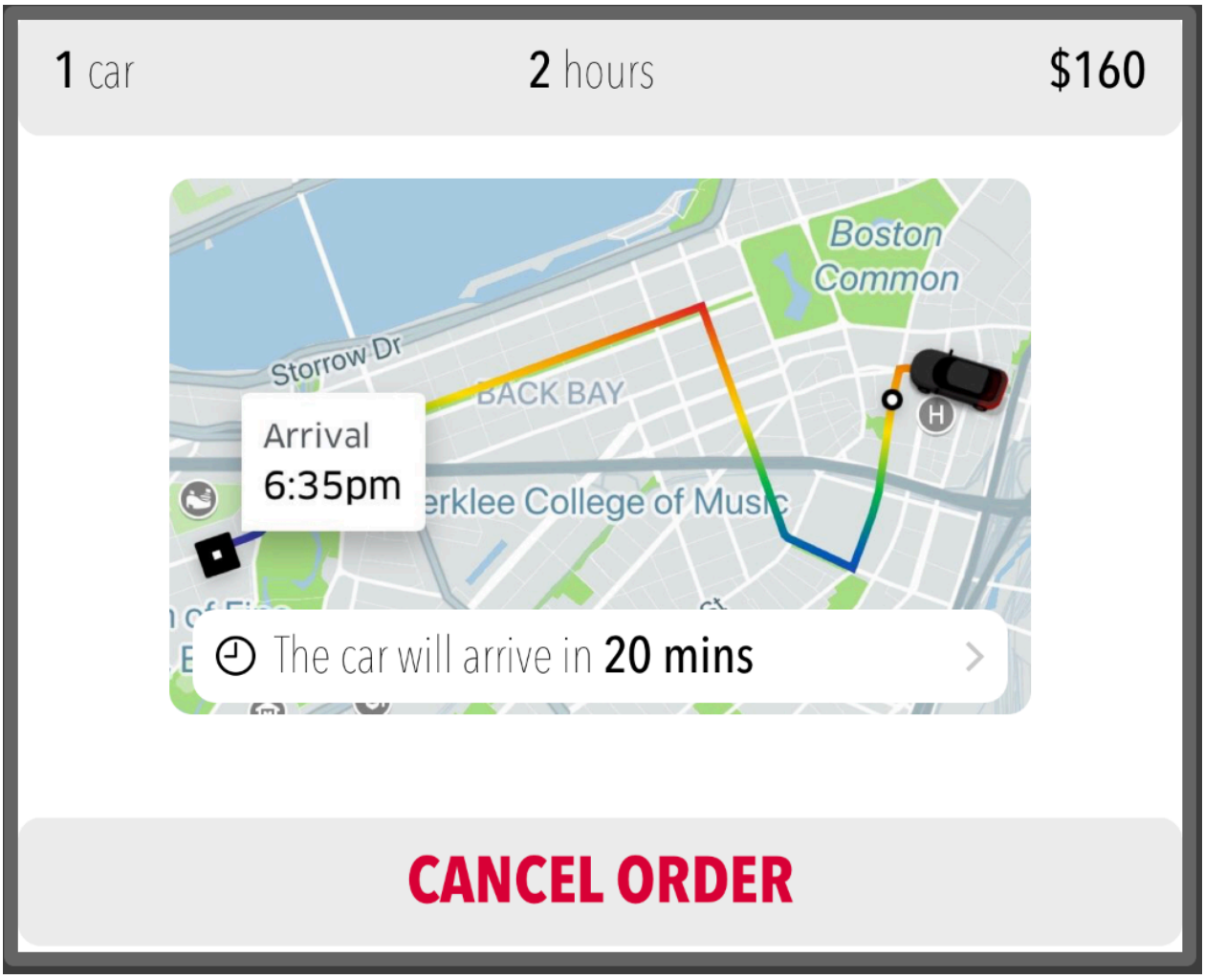
```
ZStack {
    Image("sample-map")
        .padding(.bottom, 30)

    HStack {
        Image(systemName: "clock")
        HStack(spacing: 4) {
            Text("The car will arrive
in").customFont(.custom(.ultralight, 22))
            Text("20 mins").customFont(.custom(.medium, 22))
        }
        Spacer()
        Image("disclosure-indicator")
    }
    .frame(height: 40)
    .padding(.horizontal, 10)
    .background(Color.white)
    .cornerRadius(10)
    .padding(.horizontal, 10)
    .offset(y: 75)
}.frame(width: 370)

// Add next step here
```

In the code we just added, we used a ZStack so that we can stack image and text on top of the map. In a real application, using MapKit would be a better idea or snapshot the map and annotation and add the image that way. Finally, we need to add our button next, replace `// Add next step here` with the following:

```
Button(action: { }) {
    Text("CANCEL ORDER")
}
```

```
.frame(height: 55)
.frame(minWidth: 0, maxWidth: .infinity)
.background(Color.baseGray)
.buttonStyle(PlainButtonStyle())
.cornerRadius(10)
.foregroundColor(.baseCardinal)
.customFont(.custom(.bold, 28))

Spacer()
```

Our button is used to cancel the order and we added some custom design to it. When you are finished, you'll see the following inside of Preview:



We are done with Bottom Order view, let's move on the Complete Order next.

Complete Order View

Now that we have both the Top Order and Bottom Order is complete let's put it all together to create the Complete Order view. Open `CompleteOrderView` and replace `Text("Complete Order View")` with the following code:

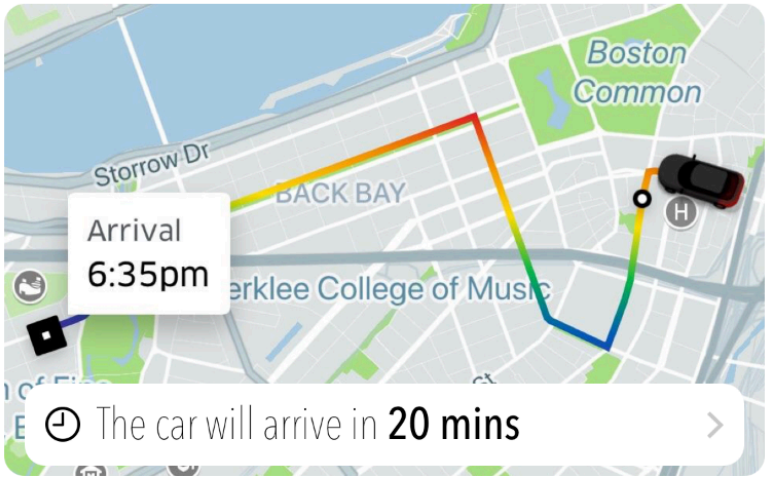
```
ZStack {
    Color.white
        .edgesIgnoringSafeArea(.all)
    VStack {
        TopOrderView()
        BottomOrderView()
    }
    .padding(.horizontal, 20)

    // Add Cancel Order View here
}
```

In the above code, we are combining `TopOrderView` and `BottomOrderView` we are adding them to a `VStack` which is located in a `ZStack`. When you are done adding this code, you'll see the following:

ORDER COMPLETED

TESLA MODEL S

1 car	2 hours	$160



CANCEL ORDER

B15463 0420

Complete Order View is done but we still need to finish up the content view. Let's do this next.

Combining up our Views

We have created `CarDetailView`, `FormView`, and `CompleteOrderView` and we need to combine them all together. Open `ContentView` and replace `Text("Content view")` with the following code:

```
ZStack {
    NavigationView {
        ScrollView(.vertical) {
            VStack(spacing: 0) {
                CarDetailView()
                    .frame(height: 600)
                FormView()
                    .frame(height: 570)
            }.padding(.top, 100)
        }.hideNavigationBar()
    }

    CompleteOrderView()
        .opacity(0)
        .animation(.default)
}.padding(.horizontal, 35)
```

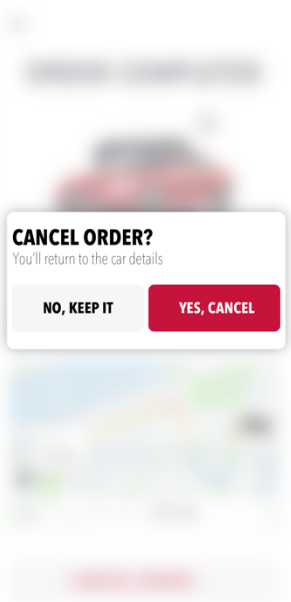
We are using a `ZStack` as our main container. Using a `NavigationView` will allow us to go to the list view for our forms and by wrapping all of it into a `ScrollView` gives us scrolling. As you can see, breaking up all of our views makes our main views relatively clean. Anytime you can separate out your views do so but just remember some-

times you might not be able to. For example, splitting up the form into multiple forms is a bit harder especially with the design but other designs might let you split it up.

We have one more view remaining and it will be your challenge.

Cancel Order Design Challenge

When you tap on Cancel Order in the `CompleteOrderView` you will see the following:



B15463 0421

You do not have to make this work in code we will code this together in the next chapter. The challenge is just for the design only.

Before, you start working on this view we will create the blur view together first. We are going to create our first `UIViewRepresentable`. Open `VibrancyBackground` and add the following:

```
struct VibrancyBackground: UIViewRepresentable {
    var style: UIBlurEffect.Style = .light

    func makeUIView(context: UIViewRepresentableContext
<VibrancyBackground>) -> UIVisualEffectView {
        return UIVisualEffectView(effect:
UIBlurEffect(style: style))
    }

    func updateUIView(\_ uiView: UIVisualEffectView,
context: Context) {
        uiView.effect = UIBlurEffect(style: style)
    }
}
```

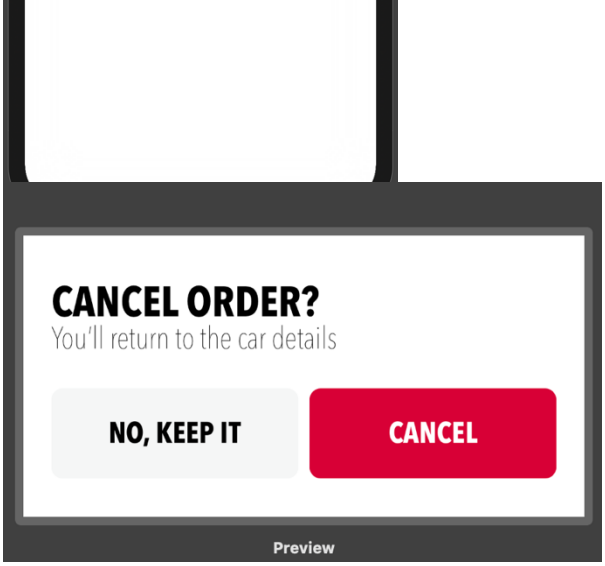
The VibrancyBackground will be used as a background and it can be used as followed:

```
Rectangle()
    .fill(Color.clear)
    .background(VibrancyBackground())
```

The above code sets the `Rectangle` to `.clear` and then we attach our `VibrancyBackground()` to the `.background`. Now it is time for you to create the modal view design. You do not need to worry about making it appear visually as we will cover that in the next chapter.

Here are the design challenges:

1. *Create* `CancelOrderView` *design with Vibrancy background.*

2. 

B15463 0423

1. *Bonus: Create a Button Style (named TeslaButtonStyle) for the two buttons.*

The bonus challenge is something we haven't covered in this chapter but we will in chapter 6. For now do the best you can and see if you can figure it out. All the code for this challenge will be available in the Chapter 4 completed folder. Remember that our code may not be exactly the same so focus on the end result. If yours matches the design then you did fine. There are numerous ways to accomplish this task so even if your code doesn't match mine it does not mean you did it wrong.

Summary

In this chapter, we worked on our first iPhone app using SwiftUI. We learned how to take a basic form in SwiftUI and customize the design. We also learned how to break up our views into smaller views. Smaller views make it great for reuse but they are also great for cleaner code. Finally, we created our first `UIViewRepresentable` in order to create a blurry effect for our modal.

In the next chapter, we will work on getting data into our Pickers, learn how to get data from our form and send it to an API.