Emmanuel Etukudo FollowI am a full-stack developer with more than five years of experience, with a preference for JavaScript, Node.js, Go React, Redux, and MongoDB.

Create and customize Flutter radio buttons

October 13, 2021 5 min read

A recurring challenge when collecting user data is formatting answer choices so that a question's instructions are implicit to the user. A radio button is a graphical UI element that presents a predefined list of mutually exclusive answer choices, solving the problem through its design.

Radio buttons are frequently brief and easy to scan, making them a great choice for mobile applications. In Flutter, the default radio button doesn't maintain any state. Instead, it invokes the `onChange` callback function each time an option is selected.

In this tutorial, we'll build a simple radio button using the built-in method from Flutter, then build our own Flutter widget for customizing a radio button. To follow along with this tutorial, you'll need:

- Flutter installed in your machine
- Basic knowledge of Flutter
- Familiarity with Dart
- Xcode or Android Studio installed on your machine
- iOS Simulator or Android emulator for testing
- A code editor, i.e., VS Code

Let's get started!

Getting started

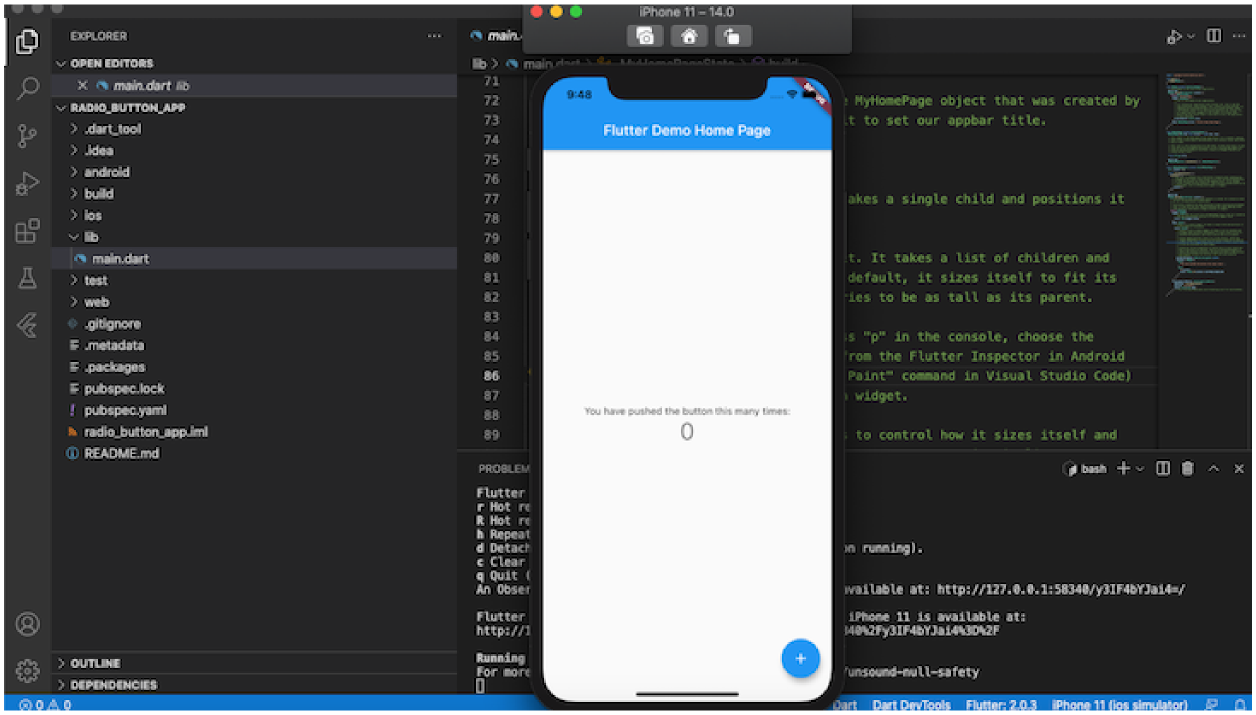
First, we'll set up a new Flutter project. Navigate into your work directory, then copy and paste the code below:

```
flutter create radio_button_app
```

Once initialization is complete, open either your Android emulator or iOS Simulator. Navigate into the `stripe_app` folder and run the app with the command below:

```
cd radio_button_app && flutter run
```

Your app should look similar to the screenshot below, indicating that your app installation was successful:



```
e MyHomePage object that was created by
t to set our appbar title.

akes a single child and positions it

t. It takes a list of children and
default, it sizes itself to fit its
ies to be as tall as its parent.

s "p" in the console, choose the
rom the Flutter Inspector in Android
Paint" command in Visual Studio Code)
 widget.

 to control how it sizes itself and
```

```
Flutter
r Hot r
R Hot r
h Repeat
d Detach
c Clear
q Quit (
An Obser

Flutter
http://1

Running
For more
```

```
n running).

vailable at: http://127.0.0.1:58340/y3IF4bYJai4=/

 iPhone 11 is available at:
340%2Fy3IF4bYJai4%3D%2F

/unsound-null-safety
```

Build a radio button in Flutter

First, let's build a radio button using Flutter Material Components widgets. Import the package by adding the code below to the top of the `main.dart` file:

```
import 'package:flutter/material.dart';
```

Create a stateless widget

Next, we need to create a stateless widget, which is immutable, allowing us to build other apps on top of it. Let's create a stateless widget called `MyApp`, which will act as the root widget and hold our application's scaffold. Add the code below to `main.dart`:

```
void main() => runApp(const MyApp());

class MyApp extends StatelessWidget {
  const MyApp({Key key}) : super(key: key);
  static const String _title = 'Radio buttons';
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: _title,
      debugShowCheckedModeBanner: false,
      theme: ThemeData(
        primaryColor: Colors.black,
        accentColor: Colors.black,
        ),
      home: Scaffold(
        appBar: AppBar(title: const Text(_title)),
        body: const Center(
          child: MyStatefulWidget(),
        ),
      ),
    );
  }
}
```

In the code above, we pass the `MyApp` widget to the `runApp` function, making it the root of the widget tree:

```
void main() => runApp(const MyApp());
```

The `MyApp` widget does not hold any state. Therefore, to build components that are mutable or stateful widgets like the radio button, we'll need to create a stateful widget and pass it to the root widget as a child:

```
child: MyStatefulWidget(),
```

Create a stateful widget

Next, let's create `MyStatefulWidget` by adding the code below to `main.dart`:

```
class MyStatefulWidget extends StatefulWidget {
  const MyStatefulWidget({Key key}) : super(key: key);
  @override
  State<MyStatefulWidget> createState() =>
_MyStatefulWidgetState();
}
```

`MyStatefulWidget` depends on a private state, which is triggered by `createState`, an immediately invoked function (IIF). As a result, `MyStatefulWidget` calls the `_MyStatefulWidgetState` private state.

To establish the `_MyStatefulWidgetState` private state, let's add the code below to our `main.dart` file:

```
class _MyStatefulWidgetState extends State<MyStatefulWidget> {
  Pet _pet = Pet.dog;

  @override
  Widget build(BuildContext context) {

    return Column(

      children: <Widget>[
        ListTile(
          title: const Text('Dog'),
          leading: Radio<Pet>(
            fillColor: MaterialStateColor.resolveWith((states)
=> Colors.green),
            focusColor: MaterialStateColor.resolveWith((states)
=> Colors.green),
            value: Pet.dog,
            groupValue: _pet,
            onChanged: (Pet value) {
              setState(() {
                _pet = value;
              });
            },
          ),
        ),
        ListTile(
          title: const Text('Cart'),
          leading: Radio<Pet>(
            fillColor: MaterialStateColor.resolveWith((states)
=> Colors.green),
            value: Pet.cat,
            groupValue: _pet,
            onChanged: (Pet value) {
              setState(() {
                _pet = value;
              });
            },
          ),
        ),
      ],
    );
  }
}
```

In this example, we'll create a simple radion button that asks a user to select between cat or dog. First, we set the value of `Pet` to `Pet.dog`, which is an enum declared in the global context of `main.dart`:

```
enum Pet { dog, cat }
```

You can add the code for the enum value anywhere in `main.dart` as long as it is accessible via the global context.

Each radio button is built with the `ListTile` material class, allowing for a combination of text, icons, and buttons.

The value of each selected option is passed to the `groupValue`, which is maintained by `MyStatefulWidget`. Whenever a radio button is selected, the button state is updated, changing the value of `_pet` to the currently selected option.

The full code for this section is below:

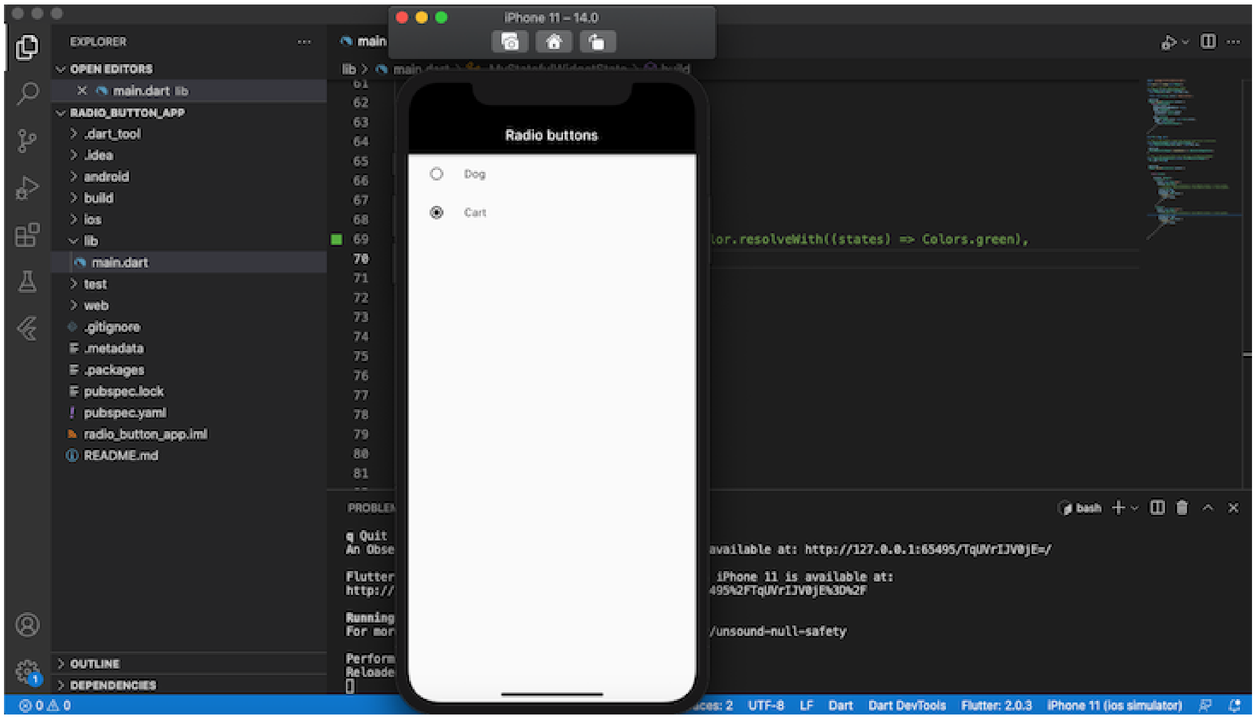
```
import 'package:flutter/material.dart';
void main() => runApp(const MyApp());
/// This is the main application widget.
class MyApp extends StatelessWidget {
  const MyApp({Key key}) : super(key: key);
  static const String _title = 'Radio buttons';

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: _title,
      debugShowCheckedModeBanner: false,
      theme: ThemeData(
        primaryColor: Colors.black,
        accentColor: Colors.black,
        ),
      home: Scaffold(
        appBar: AppBar(title: const Text(_title)),
        body: const Center(
          child: MyStatefulWidget(),
        ),
      ),
    );
  }
}
enum Pet { dog, cat }
/// This is the stateful widget that the main application
instantiates.
class MyStatefulWidget extends StatefulWidget {
  const MyStatefulWidget({Key key}) : super(key: key);
  @override
  State<MyStatefulWidget> createState() =>
_MyStatefulWidgetState();
}
/// This is the private State class that goes with
MyStatefulWidget.
class _MyStatefulWidgetState extends State<MyStatefulWidget> {
  Pet _pet = Pet.dog;
  @override
  Widget build(BuildContext context) {

    return Column(

      children: <Widget>[
        ListTile(
          title: const Text('Dog'),
          leading: Radio<Pet>(
            value: Pet.dog,
            groupValue: _pet,
            onChanged: (Pet value) {
              setState(() {
                _pet = value;
              });
            },
          ),
        ),
        ListTile(
          title: const Text('Cart'),
          leading: Radio<Pet>(
            value: Pet.cat,
            groupValue: _pet,
            onChanged: (Pet value) {
              setState(() {
                _pet = value;
              });
            },
          ),
        ),
      ],
    );
  }
}
```

When you run your app, it should look similar to the screenshot below:



```
lor.resolveWith((states) => Colors.green),
```

```
q Quit
An Obse                available at: http://127.0.0.1:65495/TqUVrIJV0jE=/

Flutter                 iPhone 11 is available at:
http://                495%2FTqUVrIJV0jE%3D%2F

Running
For mor                /unsound-null-safety

Perform
Reloade
```

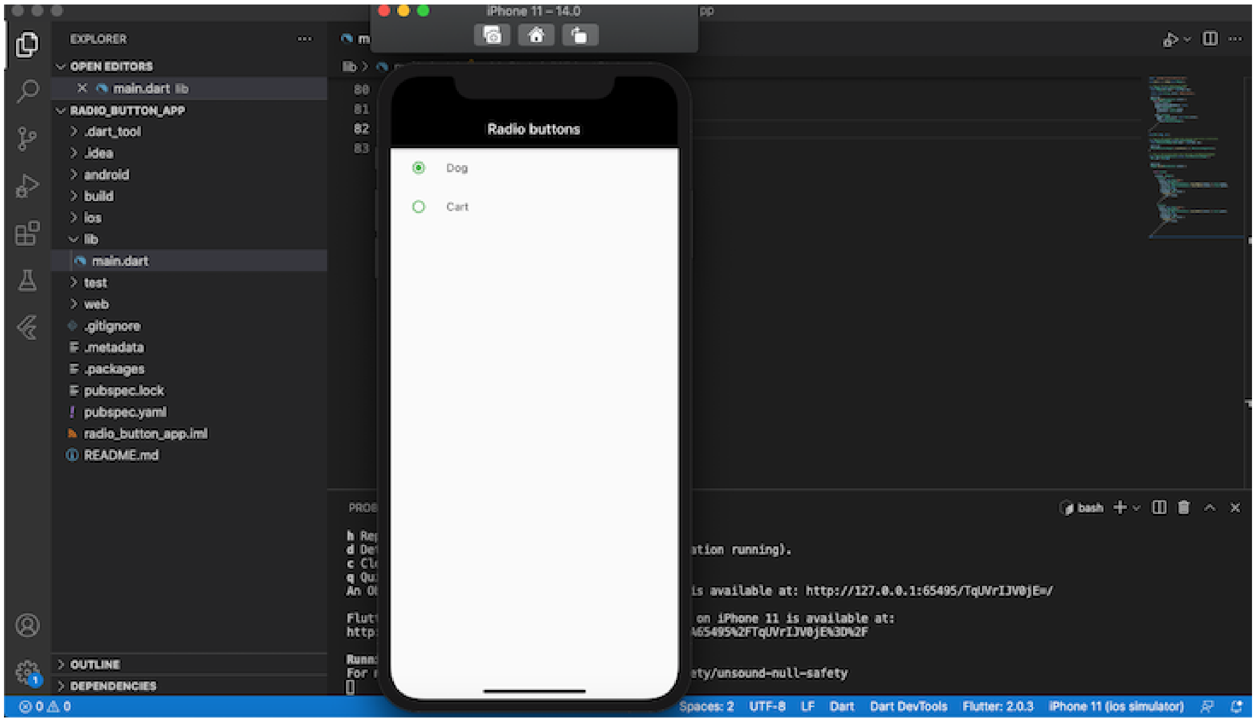
Styling a radio button

Now that we've finished building our radio button in Flutter, we can change the look and feel of it using styling properties like `activeColor`, `focusColor`, `fillColor`, and `hoverColor`.

Let's update our two `ListTile` components with the code in between `Start copy` and `End copy`:

```
ListTile(
          title: const Text('Dog'),
          leading: Radio<Pet>(
//Start copy
            fillColor: MaterialStateColor.resolveWith((states)
=> Colors.green),
            focusColor: MaterialStateColor.resolveWith((states)
=> Colors.green),
// End copy
            value: Pet.dog,
            groupValue: _pet,
            onChanged: (Pet value) {
              setState(() {
                _pet = value;
              });
            },
          ),
        ),
```

Now, your app should look like the screenshot below:



```
h Re                    ation running).
d De
c Cl
q Qu                    is available at: http://127.0.0.1:65495/TqUVrIJV0jE=/
An O
                         on iPhone 11 is available at:
Flut                    %65495%2FTqUVrIJV0jE%3D%2F
http

Runn
For                     ety/unsound-null-safety
```

Customizing a radio button in Flutter

Although it is functional, the default radio button may be too simple for your needs depending on the type of application you're building. Now that we know how to build a radio button using the default Flutter radio widget, let's build our own custom radio button for more advanced use cases.

First, let's create our own widget called `CustomRadioButton`:

```
int value = 0;
Widget CustomRadioButton(String text, int index){
      return OutlineButton(onPressed: (){
        setState(() {
          value = index;
        });
      },
      child: Text(
        text,
        style: TextStyle(
          color: (value == index) ? Colors.green  :
Colors.black,
        ),
      ),
      shape: RoundedRectangleBorder(borderRadius:
BorderRadius.circular(10)),
      borderSide: BorderSide(color: (value == index) ?
Colors.green  : Colors.black),
      );
  }
```

In the code above, we build our radio button using `OutlineButton`, similar to how we used `ListStyle` in the previous section.

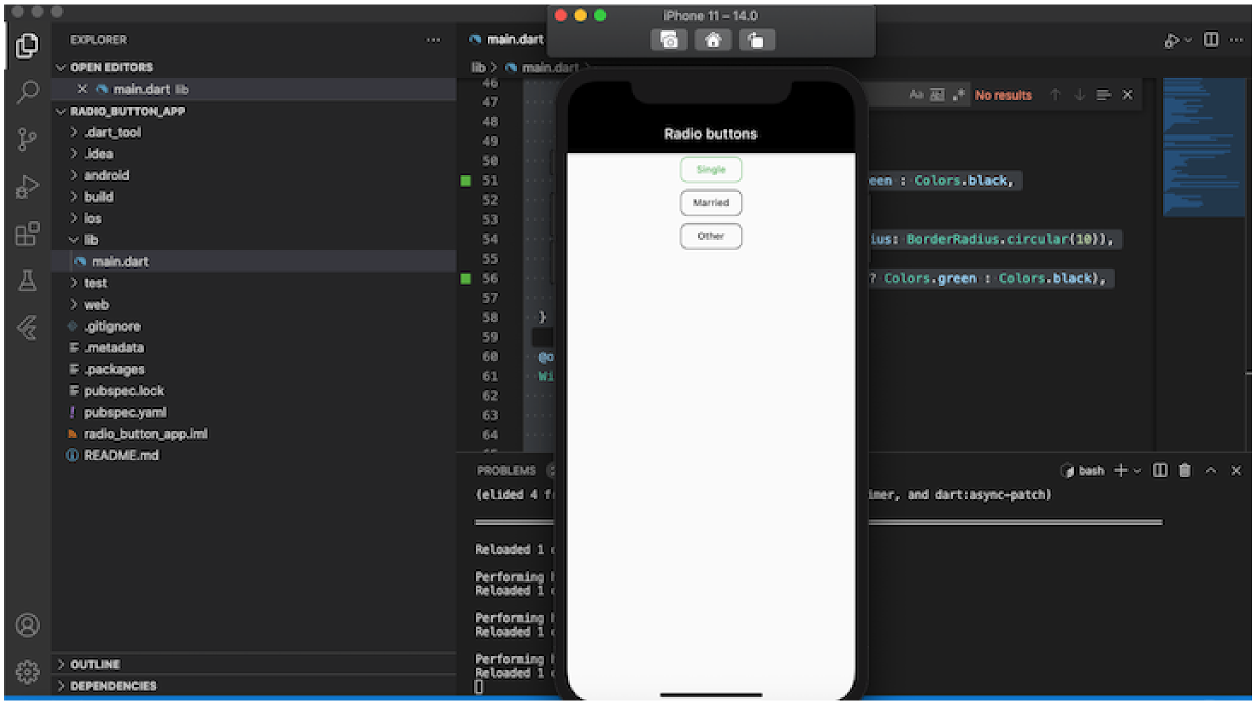
The `CustomRadioButton` widget has two parameters, `text` and `index`. `text` is the name of the radio, while the `index` contains the index number of the radio that is currently selected.

When a user selects a button, the `value` of the `index` will be updated based on the value set on `CustomRadioButton`, causing the `Radio` buttons to re-render with a new state.

Let’s build a radio button that asks a user to select between single, married, or other:

```
import 'package:flutter/material.dart';
void main() => runApp(const MyApp());
/// This is the main application widget.
class MyApp extends StatelessWidget {
  const MyApp({Key key}) : super(key: key);
  static const String _title = 'Radio buttons';
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: _title,
      debugShowCheckedModeBanner: false,
      theme: ThemeData(
        primaryColor: Colors.black,
        accentColor: Colors.black,
      ),
      home: Scaffold(
        appBar: AppBar(title: const Text(_title)),
        body: const Center(
          child: MyStatefulWidget(),
        ),
      ),
    );
  }
}
/// This is the stateful widget that the main application
instantiates.
class MyStatefulWidget extends StatefulWidget {
  const MyStatefulWidget({Key key}) : super(key: key);
  @override
  State<MyStatefulWidget> createState() =>
_MyStatefulWidgetState();
}
/// This is the private State class that goes with
MyStatefulWidget.
class _MyStatefulWidgetState extends State<MyStatefulWidget> {
  int value = 0;
  Widget CustomRadioButton(String text, int index) {
    return OutlineButton(
      onPressed: () {
        setState(() {
          value = index;
        });
      },
      child: Text(
        text,
        style: TextStyle(
          color: (value == index) ? Colors.green : Colors.black,
        ),
      ),
      shape: RoundedRectangleBorder(borderRadius:
BorderRadius.circular(10)),
      borderSide:
          BorderSide(color: (value == index) ? Colors.green :
Colors.black),
    );
  }
  @override
  Widget build(BuildContext context) {
    return Column(
      children: <Widget>[
        CustomRadioButton("Single", 1),
        CustomRadioButton("Married", 2),
        CustomRadioButton("Other", 3)
      ],
    );
  }
}
```

The output of the code above should look similar to the screenshot below:



```
een : Colors.black,
ius: BorderRadius.circular(10)),
? Colors.green : Colors.black),
```

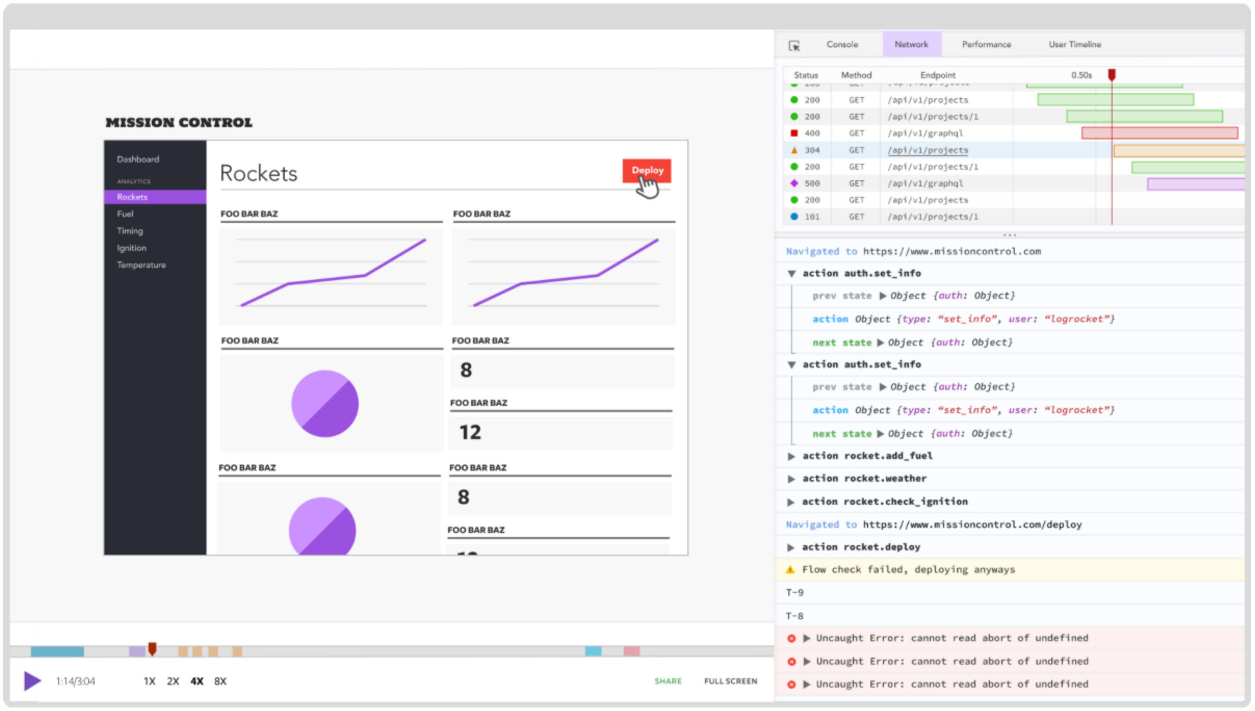
```
(elided 4 f                          iner, and dart:async-patch)

Reloaded 1 
Performing 
Reloaded 1 
Performing 
Reloaded 1 
Performing 
Reloaded 1 
```

Conclusion

A radio button is a popular UI element for its simplicity and effectiveness, particularly in mobile applications. In this tutorial, we built a simple radio button using the default Flutter radio button widget, and we built our own widget for customizing and styling a complex radio button. I hope you enjoyed this tutorial!

LogRocket: Full visibility into your web apps



Status	Method	Endpoint
200	GET	/api/v1/projects
200	GET	/api/v1/projects/1
400	GET	/api/v1/graphql
304	GET	/api/v1/projects
200	GET	/api/v1/projects/1
500	GET	/api/v1/graphql
200	GET	/api/v1/projects
101	GET	/api/v1/projects/1

```
Navigated to https://www.missioncontrol.com
▼ action auth.set_info
    prev state ▶ Object {auth: Object}
    action Object {type: "set_info", user: "logrocket"}
    next state ▶ Object {auth: Object}
▼ action auth.set_info
    prev state ▶ Object {auth: Object}
    action Object {type: "set_info", user: "logrocket"}
    next state ▶ Object {auth: Object}
▶ action rocket.add_fuel
▶ action rocket.weather
▶ action rocket.check_ignition
Navigated to https://www.missioncontrol.com/deploy
▶ action rocket.deploy
⚠ Flow check failed, deploying anyways
T-9
T-8
⊗ ▶ Uncaught Error: cannot read abort of undefined
⊗ ▶ Uncaught Error: cannot read abort of undefined
⊗ ▶ Uncaught Error: cannot read abort of undefined
```

LogRocket is a frontend application monitoring solution that lets you replay problems as if they happened in your own browser. Instead of guessing why errors happen, or asking users for screenshots and log dumps, LogRocket lets you replay the session to quickly understand what went wrong. It works perfectly with any app, regardless of framework, and has plugins to log additional context from Redux, Vuex, and @ngrx/store.

In addition to logging Redux actions and state, LogRocket records console logs, JavaScript errors, stacktraces, network requests/responses with headers + bodies, browser metadata, and custom logs. It also instruments the DOM to record the HTML and CSS on the page, recreating pixel-perfect videos of even the most complex single-page apps.