Work with WebSockets

Contents

In addition to normal HTTP requests, you can connect to servers using `WebSockets`. `WebSockets` allow for two-way communication with a server without polling.

In this example, connect to a test server provided by websocket.org. The server sends back the same message you send to it. This recipe uses the following steps:

1. Connect to a WebSocket server.
2. Listen for messages from the server.
3. Send data to the server.
4. Close the WebSocket connection.

1. Connect to a WebSocket server

The `web_socket_channel` package provides the tools you need to connect to a WebSocket server.

The package provides a `WebSocketChannel` that allows you to both listen for messages from the server and push messages to the server.

In Flutter, use the following line to create a `WebSocketChannel` that connects to a server:

```
final channel = WebSocketChannel.connect(
  Uri.parse('wss://echo.websocket.org'),
);
```

2. Listen for messages from the server

Now that you’ve established a connection, listen to messages from the server.

After sending a message to the test server, it sends the same message back.

In this example, use a `StreamBuilder` widget to listen for new messages, and a `Text` widget to display them.

```
StreamBuilder(
  stream: channel.stream,
  builder: (context, snapshot) {
    return Text(snapshot.hasData ? '${snapshot.data}' : '');
  },
)
```

How this works

The `WebSocketChannel` provides a `Stream` of messages from the server.

The `Stream` class is a fundamental part of the `dart:async` package. It provides a way to listen to async events from a data source. Unlike `Future`, which returns a single async response, the `Stream` class can deliver many events over time.

The `StreamBuilder` widget connects to a `Stream` and asks Flutter to rebuild every time it receives an event using the given `builder` function.

3. Send data to the server

To send data to the server, `add()` messages to the `sink` provided by the `WebSocketChannel`.

```
channel.sink.add('Hello!');
```

How this works

The `WebSocketChannel` provides a `StreamSink` to push messages to the server.

The `StreamSink` class provides a general way to add sync or async events to a data source.

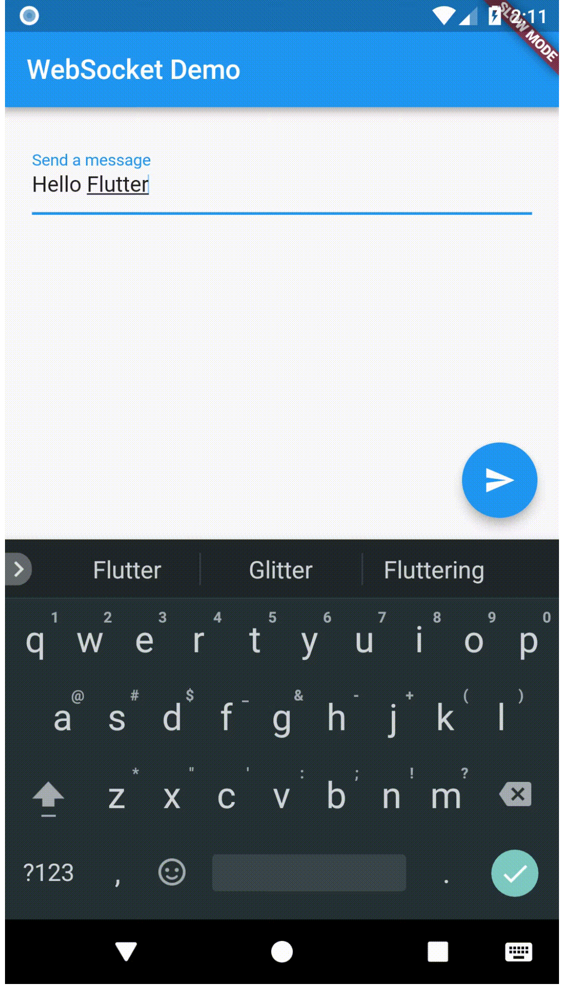
4. Close the WebSocket connection

After you’re done using the WebSocket, close the connection:

```
channel.sink.close();
```

Complete example

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

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

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    const title = 'WebSocket Demo';
    return const MaterialApp(
      title: title,
      home: MyHomePage(
        title: title,
      ),
    );
  }
}

class MyHomePage extends StatefulWidget {
  const MyHomePage({
    Key? key,
    required this.title,
  }) : super(key: key);

  final String title;

  @override
  _MyHomePageState createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  final TextEditingController _controller = TextEditingController();
  final _channel = WebSocketChannel.connect(
    Uri.parse('wss://echo.websocket.org'),
  );

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(widget.title),
      ),
      body: Padding(
        padding: const EdgeInsets.all(20.0),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Form(
              child: TextFormField(
                controller: _controller,
                decoration: const InputDecoration(labelText: 'Send a message'),
              ),
            ),
            const SizedBox(height: 24),
            StreamBuilder(
              stream: _channel.stream,
              builder: (context, snapshot) {
                return Text(snapshot.hasData ? '${snapshot.data}' : '');
              },
            )
          ],
        ),
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: _sendMessage,
        tooltip: 'Send message',
        child: const Icon(Icons.send),
      ), // This trailing comma makes auto-formatting nicer for build methods.
    );
  }

  void _sendMessage() {
    if (_controller.text.isNotEmpty) {
      _channel.sink.add(_controller.text);
    }
  }

  @override
  void dispose() {
    _channel.sink.close();
    _controller.dispose();
    super.dispose();
  }
}
```



〈 Update data over the internet

Persist data with SQLite 〉