Dry exercise

Answer the following questions:

1. Identify the 2 lines of code that make the list **infinitely scrolling**. What happens if we remove these lines, and scroll to the end of the list? Assume _suggestions starts with a non-empty list of 10 word-pairs. You can use `generateWordPairs().take(10).toList()`
   Hint: run the changed code to see if you're right

Answer:

The two lines of code are:

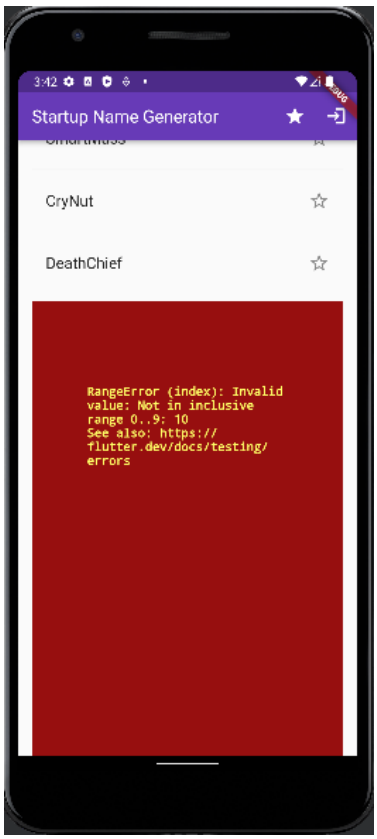
```
if (index >= _suggestions.length) {
  // ...then generate 10 more and add them to the
  // suggestions list.
  _suggestions.addAll(generateWordPairs().take(10));
}
```

After removing them, the list will contain the first ten words (As described for initializing in the question description), And after the ten words we will get an Error (Red screen in the emulator), The error is:

```
======== Exception caught by widgets library =======================================================
The following RangeError was thrown building:
RangeError (index): Invalid value: Not in inclusive range 0..9: 10

When the exception was thrown, this was the stack:
#0      List.[] (dart:core-patch/growable_array.dart:264:36)
#1      _RandomWordsState._buildSuggestions.<anonymous closure> (package:hello_me/randomWordsPage.dart:49:38)
#2      SliverChildBuilderDelegate.build (package:flutter/src/widgets/sliver.dart:465:22)
#3      SliverMultiBoxAdaptorElement._build (package:flutter/src/widgets/sliver.dart:1230:28)
#4      SliverMultiBoxAdaptorElement.createChild.<anonymous closure> (package:flutter/src/widgets/sliver.dart:1244:55)
#5      BuildOwner.buildScope (package:flutter/src/widgets/framework.dart:2660:19)
#6      SliverMultiBoxAdaptorElement.createChild (package:flutter/src/widgets/sliver.dart:1236:12)
#7      RenderSliverMultiBoxAdaptor._createOrObtainChild.<anonymous closure> (package:flutter/src/rendering/sliver_multi_box_adaptor.dart:349:23)
[93 more...]
(elided 11 frames from class _RawReceivePortImpl, class _Timer, dart:async, and dart:async-patch)
```

And as shown in the emulator (the red screen after the tenth word):



2. Our list is separated with **Divider** widgets. Visit the **ListView** widget documentation here: https://api.flutter.dev/flutter/widgets/ListView-class.html; Give a different method to construct such a list with dividers. Which way do you think is better, and why?
   You may assume for this question only that the list is finite and contains 100 items from the start.

Answer:

Assuming that we are using a list containing 100 items, We could use ListTile.divideTiles.

Divider (which is used in our code) gets the same effect as divideTiles but manually, meanwhile divideTiles works on the static list, and divider works on the dynamic list.

On our code, we are working on ListView.Builder, but the divideTiles works on ListView(Children: ListTile.divideTiles…

I think working with the divider is better because it supports a dynamic list, and in real life usually lists are dynamic and not static.

3. `_buildRow` contains a call to `setState()` inside the `onTap()` handler. Why do we need it there?

Answer:

The purpose of the setState() call here is to identify the tile state, when we click on the tile for the first time (onTap() function) we add the tile to the _saved list, and we click the second time we remove the tile from the _saved list.

On each tap on the tile, we should determine whether to add it to _saved or remove it from there, and that's why it's called inside the onTap function.

Dry exercise

Answer the following questions:

1. What is the purpose of the **MaterialApp** widget? Provide examples of 3 of its properties followed by a short explanation.

Answer:

An application that uses material design.

A convenience widget that wraps several widgets that are commonly required for material design applications. It builds upon a WidgetApp by adding material-design specific functionality, such as AnimatedTheme and GridPaper.

The MaterialApp configures the top-level Navigator to search for routes in the following order:

1. For the / route, the home property, if non-null is used.
2. Otherwise, the routes table is used, if it has an entry for the route.
3. Otherwise, onGenerateRoute is called, if provided. It should return a non-null value for any *valid* route not handled by home and routes.
4. Finally, if all else fails onUnknownRoute is called.

Examples of three properties:

1. **Title**: A one-line description used by the device to identify the app for the user.
2. **Theme**: Default visual properties, like colors fonts, and shapes, for this app's material widgets.
3. **Locale**: The initial locale for this app's Localization widget is based on this value (for the initial language of the app purposes).

2. The **Dismissible** widget has a key property. What does it mean and why is it required?

Answer:

Key: Each dismissible must contain a Key. Keys allow Flutter to uniquely identify widgets.

You need to assign each data to a unique identifier. Something unique enough for it to not contain any duplicates. Then you can associate that unique identifier to a **Key**.

This can't be done just with a primitive object such as **String** or **Int**. We'll need to map our data to a custom object.

We can generate a custom ID for our data using the **UUID** package or using a custom algorithm (such as an incremental index).

But we must be sure that our ID is unique for each item and stays the same for the whole lifetime of that item (even after updates).

For example:

```
Dismissible(
    key: Key(widget.data[i].id),
    ...
)
```

As we can see in the example, the key is unique for the widget, because of the uniqueness of the id.


Now let's see who you really are
Widget!
Widget
Widget
Widget
Widget
Widget
Widget
Widget
Widget