Dry Part

Exercise 1

1.
```
if (index >= _suggestions.length) {
        _suggestions.addAll(generateWordPairs().take(10)); }
```

If we remove these two lines and scroll to the end of the list – assuming we have a list of 10 items – we will get an error that index is out of range. That is because of the following line:

```
return _buildRow(_suggestions[index]);
```

index is incremented (as a result of the iterator's incrementation) as we scroll through the list because of the itemBuilder function that is part of the listView's builder property. And since _suggestions is a list of length 10, we cannot access any elements with a higher index.

2. We can construct a list with Divider widgets by using the **ListView.separated** constructor, which takes two IndexedWidgetBuilders: **itemBuilder** builds child items on demand, and **separatorBuilder** similarly builds separator children which appear in between the child items. We can use the Divider widget as a separator to get the same list – assuming that it's finite.

This constructor is appropriate for list views with a fixed number of children, so for an infinite list, we need to use ListView.builder and our list with the current dividers is better. But, for a finite list, using ListView.seperated is better because it already provides separators that we can just declare to be the Divider widget, and we don't need to add them in ourselves.

3. In _buildRow, we need to call setState() inside the onTap() handler in order to notify the framework that state has changed when tapping on the icon in that row.

When tapping on an icon, we want the word pair in that row to either be added to or removed from the _saved set in _RandomWordsState. Calling setState() triggers a call to the build() method for_RandomWordsState, resulting in an update to the UI.

Exercise 2

1.MaterialApp widget adds Material Design-specific functionality and styling options to our app, and it is using this widget that we can make an attractive app. It also allows us to use widgets that are specifically in the Material widgets collection; using the MaterialApp widget, we can access many components and widgets provided by Flutter SDK.

Some properties of MaterialApp widget:
• **home:** This property takes in *widget* as the object to show on the default route of the app.
• **title:** The *title* property takes in a *string* as the object to decide the one-line description of the app for the device.
• **theme:** This property takes in *ThemeData* class as the object to describe the theme for the MaterialApp.

2. The key property controls how one widget replaces another widget in the tree. The Dismissible widget requires a key property because Dismissibles are commonly used in lists and removed from the list when dismissed. Without keys, the default behavior is to sync widgets based on their index in the list, which means the item after the dismissed item would be synced with the state of the dismissed item. Using keys causes the widgets to sync according to their keys and avoids this pitfall.