Code Explanation

Created	@January 14, 2022 10:48 AM
Last Edited Time	@January 14, 2022 12:54 PM
Created By	

main.dart

`void main`

- Responsible for the initialization of
  - ▼ the `EasyLocalization` (language localization package),

```
await EasyLocalization.ensureInitialized();
...
EasyLocalization(
        supportedLocales: const [
          Locale('en'),
          Locale('fr'),
          Locale('pt'),
          Locale('de')
        ],
        path: 'assets/languages',
        fallbackLocale: const Locale('en'),
        child: const GbloxApp()));
```

  - ▼ Setting the orientation of the application to landscape and status bar to `immsersiveSticky`

```
SystemChrome.setPreferredOrientations(
         [DeviceOrientation.landscapeLeft, DeviceOrientation.landscapeRight])
```

```
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  SystemChrome.setEnabledSystemUIMode(SystemUiMode.immersiveSticky);
  await EasyLocalization.ensureInitialized();
  SystemChrome.setPreferredOrientations(
         [DeviceOrientation.landscapeLeft, DeviceOrientation.landscapeRight])
      .then((_) {
    runApp(EasyLocalization(
        supportedLocales: const [
          Locale('en'),
          Locale('fr'),
          Locale('pt'),
          Locale('de')
        ],
        path: 'assets/languages',
        fallbackLocale: const Locale('en'),
        child: const GbloxApp()));
  });
}
```

`class GbloxApp`

- Stateful Widget extended by `_GbloxAppState`
- Responsible for loading in initial `BuildContext context`, `localizationsDelegates`, `navigatorKey` and `theme`.
- Creates `GBloxCards` using `startCards`

Components

▼ `class CardDetails`

- Class for holding data used to generate `GBloxCards`

```
String svg;
  String title;
  Color textBackgroundColor;
  Function? pressed;
  bool compressSVG;

  _CardDetails(this.svg, this.title, this.textBackgroundColor, this.pressed,
      this.compressSVG);
```

▼ `List<_CardDetails> startCards`

- Empty growable list of `CardDetails` which is used to generate the row of `GBloxCards` in the body of the widget.
- Data is added in `void initState()`

```
late List<_CardDetails> startCards = List<_CardDetails>.empty(growable: true);
...
void initState() {
    super.initState();

    startCards.add(_CardDetails(svgs.playMode, "Select Mode", Colors.red, () {
      Navigator.push(
        context,
        MaterialPageRoute(builder: (context) => ModeSelector()),
      );
    }, false));
    startCards.add(_CardDetails(svgs.mingo, "Select Device", Colors.orange, () {
      Navigator.push(
        context,
        MaterialPageRoute(builder: (context) => DiscoveryPage()),
      );
    }, false));
  }
```

global_variables.dart

- Library used to store various variables used throughout the application
- Variables included:

Global Classes

▼ `class ToolboxClass`

- Class for holding data used to generate `ToolboxCategoryButtons`

```
BluetoothConnection activeConnection
GlobalKey<NavigatorState> navigatorKey
ThemeData globalTheme
InAppWebViewController? webController
String selectedDevice
```

- Stores `displayToast` function used to display toast for debugging purposes.

```
class ToolboxClass {
  String name;
  bool category;
  int index;
  String click;
  ToolboxClass(this.name, this.category, this.index, this.click);
}
```

▼ `class CardDetails`

- Class for holding data used to generate `GBloxCards`

```
String svg;
  String title;
  Color textBackgroundColor;
  Function? pressed;
  bool compressSVG;

  _CardDetails(this.svg, this.title, this.textBackgroundColor, this.pressed,
      this.compressSVG);
```

▼ `class SaveInformation`

- Class for holding data used to save and load `.gbx` files

```
class SaveInformation {
  String xml;
  String device;
  String variables;
  String filename;
  String filepath;
  bool internal;

  SaveInformation(this.xml, this.device, this.variables, this.filename,
      this.filepath, this.internal);
}
```

▼ `class ShapeData`

- Class for holding data used to generate sketch shapes

```
class ShapeData {
  final String name;
  final String svg;
  final Path path;
  ShapeData(this.name, this.svg, this.path);
}
```

- Calculates device height and device width and saves it to `Size device_size`.

Modular Widgets

- Custom Widgets created for various features of the application.

Button

- Generates `GBloxButtons` used in various pages.

Use case Example

```
GBloxButtons(
  buttonType: "controller_circle",
  icon: GBloxCustomSVGs.gBloxLogo,
  pressed: () {},
  buttonColor: 0xff1D184B),
```

```
@required
  late final String buttonType;
  @required
  late final IconData icon;
  @required
  late final Function? pressed;
  late final Function? onLongPress;
  late final bool rotate;
  late final int buttonColor;
  late final String buttonName;
```

Arguments:

▼ `buttonType` - **required**

- Accepts a `String`
- Used in a switch statement to determine what type of button to return
- e.g

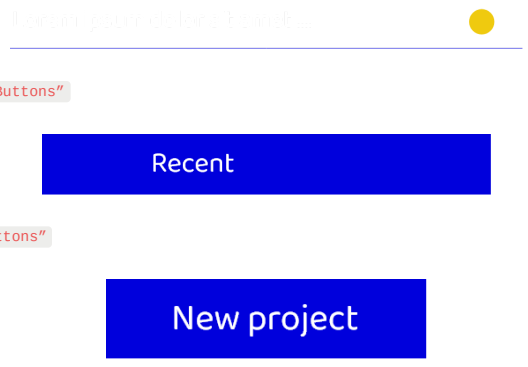
```
buttonType: "controller_circle",
```

- List of buttonTypes:
  - ▼ `“controller_square”`

  - ▼ `“controller_circle”`

  - ▼ `“menuButtons”`

  - ▼ `“fileButtons”`



▼ `icon` - **required**

- Accepts `IconData`
- Used to display Icon for certain `buttonTypes`

- e.g

```
icon: Icons.home,
```

▼ `pressed` - **required**

- Accepts a `Function`
- Runs function when button is tapped/pressed

- e.g

```
pressed: (){
  print("This button has been pressed");
},
```

▼ `onLongPress`

- Accepts a `Function`
- Runs function when button is pressed and held

- e.g

```
onLongPress: (){
  print("This button has been pressed");
},
```

▼ `rotate`

- Accepts a `bool`
- Determines if button icon is rotated

- e.g

```
rotate: true,
```

▼ `buttonColor`

- Accepts an `integer` as a hex
- Used to return the color of the button with applied effects

- e.g

```
buttonColor: 0xff0000DC,
```

▼ `buttonName`

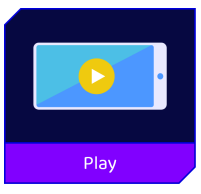
- Accepts a `String`
- Used to display the button name for some buttonTypes

- e.g

```
buttonName: "New Project"
```

Cards

- Generates `GBloxCards` used in various pages.



Use case Example

```
GBloxCards(
  svg: svgs.playMode,
  text: "mode_select_page",
  textBackgroundColor: Colors.blue,
  pressed: () {
    Navigator.push(
      context,
      MaterialPageRoute(
        builder: (context) =>ModeSelector()),
    );
  })
```

Arguments:

▼ `svg` - **required**

- Accepts a `String` svg.
- Displays svg on the card.

- e.g

```
svg: "<svg xmlns="http://www.w3.org/2000/svg" width="294.225" height="294.225" viewBox="0
  <g id="Group_785" data-name="Group 785" transform="translate(0 0)">
    <path id="Path_1363" data-name="Path 1363" d="M276.762,96.673A48.384,48.384,0,0,1,230
    <path id="Path_1362" data-name="Path 1362" d="M276.762,48.337H42.661V25.59H0a48.362,48
    <path id="Path_1366" data-name="Path 1366" d="M0,42.649V14.458L64.02,0,91.1,13.269H30
```

```
    <path id="Path_1367" data-name="Path 1367" d="M26.538,0h89.091a26.538,26.538,0,0,1,26
    <path id="Path_1365" data-name="Path 1365" d="M64.855,28.809h0L0,13.268V0H309.622V16.
    <path id="Path_1368" data-name="Path 1368" d="M115.629,35.068H26.539A26.456,26.456,0,0
  </g>
</svg>"
```

▼ `pressed` - **required**

- Accepts a `Function`
- Runs function when card is tapped/pressed

- e.g

```
pressed: (){
  print("This button has been pressed");
},
```

▼ `compressSVG`

- Accepts a `bool`
- Used to determine if the `svg` is compressed into the space or not.

- e.g

```
compressSVG: true,
```

▼ `textBackgroundColor` - **required**

- Accepts a `Color`
- Fills the text background of the card with the color

- e.g

```
textBackgroundColor:Color(0xff0000DC),
```

▼ `backgroundColor` - **required**

- Accepts a `Color`
- Fills the card with the color

- e.g

```
backgroundColor:Color(0xff0000DC),
```

▼ `text`

- Accepts a `String`
- Used to display the Card name

- e.g

```
text: "Build"
```

Selector

- Generates `SelectorButtons` used in various pages.

Use case Example

```
SelectorButtons(
  activeColor: 0xff0000DC,
  initialIndex: index,
  buttons: ["GBlox", "Phone"],
  functionList: [
    () {},
    () {
    getInteralDir();
    }
  ],
)),
```

Arguments:

▼ `buttons` - **required**

- Accepts a `List<String>`.
- Names for buttons generated

- e.g

```
buttons: [
  "Recent",
  "Help"
],
```

▼ `functionList` - **required**

- Accepts a List of functions corresponding to each button
- Runs function based on the list

- e.g

```
functionList: [
  (){
  print("This button has been pressed");
  },
  (){
  }],
```

▼ `buttonType`

- Accepts a `String`
- Generates the buttons based on **Button** `buttonTypes`

- e.g

```
buttonType: "projectButtons",
```

▼ `activeColor`

- Accepts an `integer`
- Generates `Color` used for highlighted button

- e.g

```
activeColor: 0xff0000DC,
```

▼ `initialIndex`

- Accepts an integer
- Selects the initial selected button

- e.g

```
initialIndex: 1,
```