Cours 8. Les Listes et les Scrollables

O.R. Merad Boudia

Université d'Oran 1, Ahmed Ben Bella

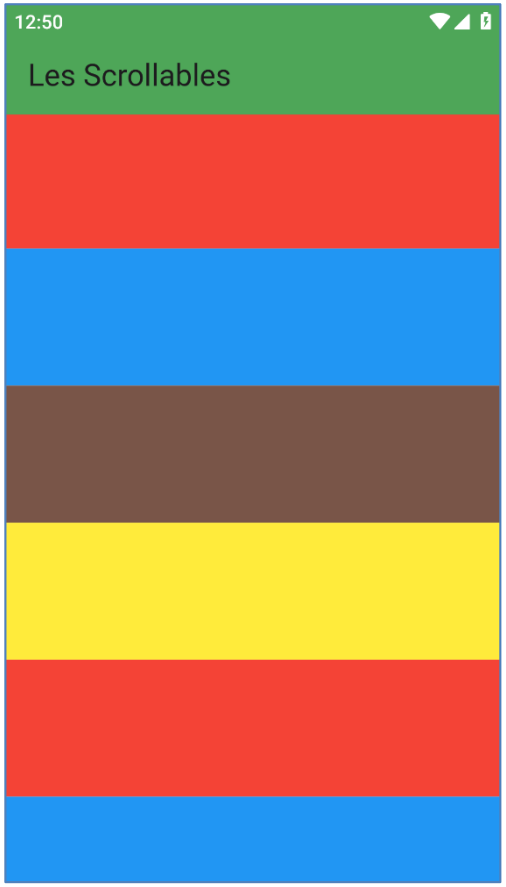
M1 GBM : 2023/2024

SingleChildScrollView

- ✓ Le widget "SingleChildScrollView" est utilisé pour créer une zone déroulante (scrollable) qui peut contenir un seul enfant.

- ✓ Il est utile lorsque le contenu à l'intérieur du widget est plus grand que l'écran et nécessite un défilement pour être complètement visible.

```dart
class _MyHomePageState extends State<MyHomePage> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        backgroundColor: Colors.green,
        title: Text(widget.title),
      ),  // AppBar
      body: SingleChildScrollView(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            Container(height: 100.0, color: Colors.red,),
            Container(height: 100.0, color: Colors.blue,),
            Container(height: 100.0, color: Colors.brown,),
            Container(height: 100.0, color: Colors.yellow,),
            Container(height: 100.0, color: Colors.red,),
            Container(height: 100.0, color: Colors.blue,),
            Container(height: 100.0, color: Colors.brown,),
            Container(height: 100.0, color: Colors.yellow,),
          ],  // <Widget>[]
        ),  // Column
      ),  // SingleChildScrollView
    );  // Scaffold
  }
}
```



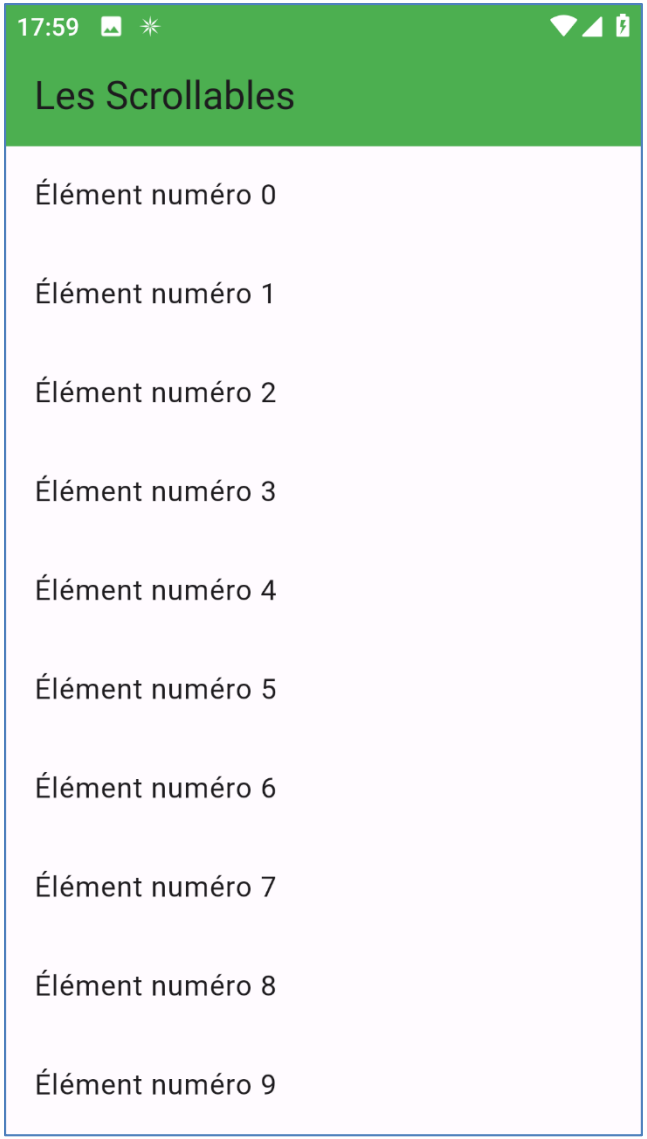
ListView et ListTile

- ✓ **ListView** est un widget qui permet d'afficher une liste déroulante de widgets enfants.
- ✓ Cela signifie que si vous avez une liste d'éléments à afficher, vous pouvez les placer à l'intérieur d'un **ListView** pour permettre aux utilisateurs de faire défiler et voir tous les éléments, même si l'espace à l'écran est limité.
- ✓ **ListView** prend en charge différents types de constructeurs, tels que **ListView.builder**, **ListView.separated**, etc., pour adapter la manière dont la liste est affichée et construite.
- ✓ **ListTile** est un widget fréquemment utilisé à l'intérieur d'un **ListView** pour représenter un élément de la liste.
- ✓ Cet élément de liste comprend généralement un titre, un sous-titre et une icône, bien que ces éléments puissent être personnalisés en fonction de vos besoins.
- ✓ **ListTile** simplifie la création de mises en page d'éléments de liste communs en fournissant une structure cohérente.

ListView et ListTile

✓ Exemple 1:

```
class _MyHomePageState extends State<MyHomePage> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
        appBar: AppBar(
          backgroundColor: Colors.green,
          title: Text(widget.title),
        ),  // AppBar
        body: Center(
          child: ListView.builder(
            itemBuilder: (context, i) {
              return ListTile(
                title: Text("Élément numéro $i"),
              );  // ListTile
            },
          ),  // ListView.builder
        ));  // Center, Scaffold
  }
}
```



ListView et ListTile

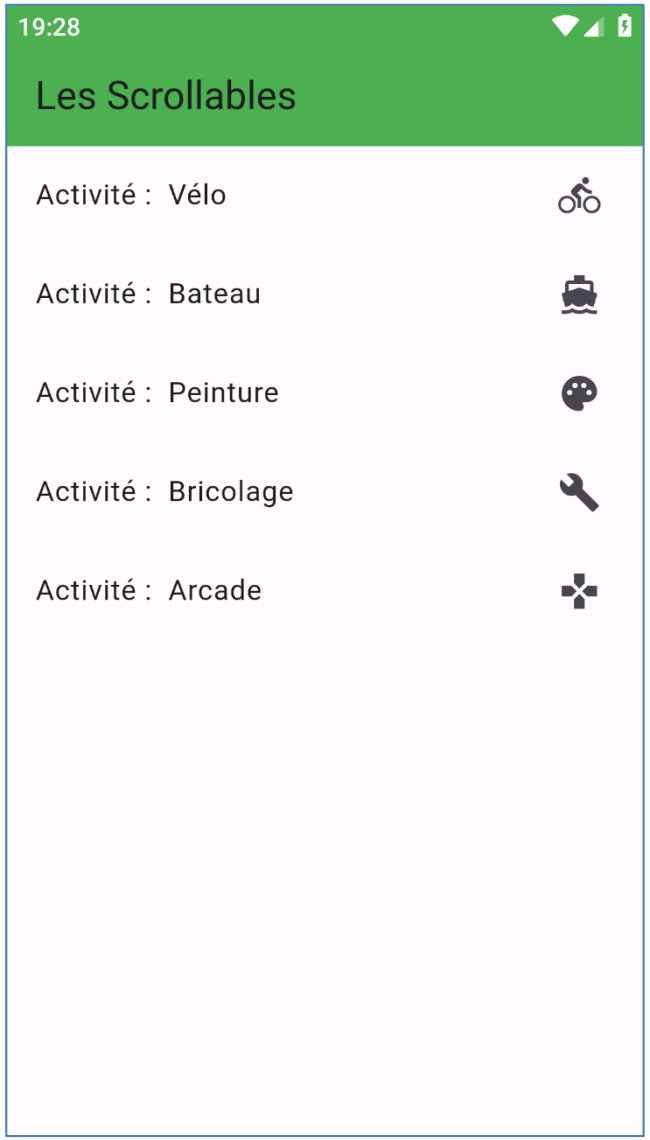
✓ Exemple 2:

```
class _MyHomePageState extends State<MyHomePage> {
  List<Activity> myActivities = [
    Activity("Vélo", Icons.directions_bike),
    Activity("Bateau", Icons.directions_boat),
    Activity("Peinture", Icons.palette),
    Activity("Bricolage", Icons.build),
    Activity("Arcade", Icons.gamepad),
  ];

  @override
  Widget build(BuildContext context) {
    return Scaffold(
        appBar: AppBar(
          backgroundColor: Colors.green,
          title: Text(widget.title),
        ),  // AppBar
        body: Center(
          child: ListView.builder(
            itemCount: myActivities.length,
            itemBuilder: (context, i) {
              return ListTile(
                title: Text("Activité :  ${myActivities[i].nom}"),
                trailing: Icon(myActivities[i].icone),
              );  // ListTile
            },
          ), )); }}  // ListView.builder, Center, Scaffold

class Activity {
  String nom;
  IconData icone;

  Activity(this.nom, this.icone);
}
```

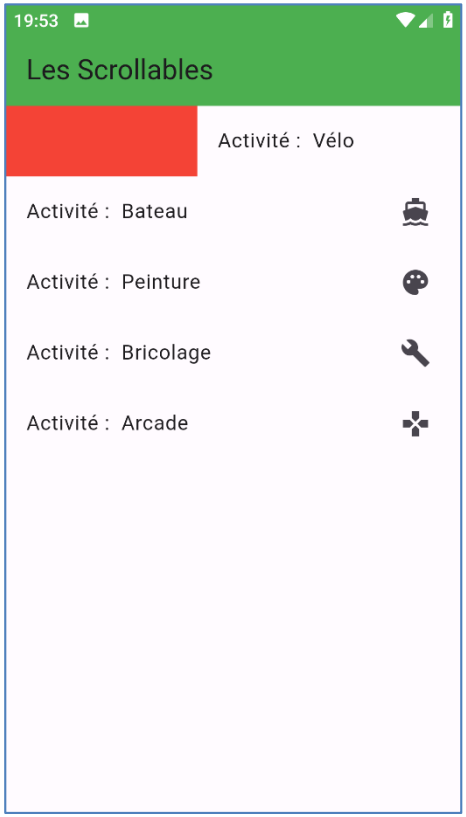


Dismissible

- ✓ **Dismissible** est un widget qui permet à l'utilisateur de faire glisser un élément de la liste horizontalement pour effectuer une action telle que la suppression de l'élément.

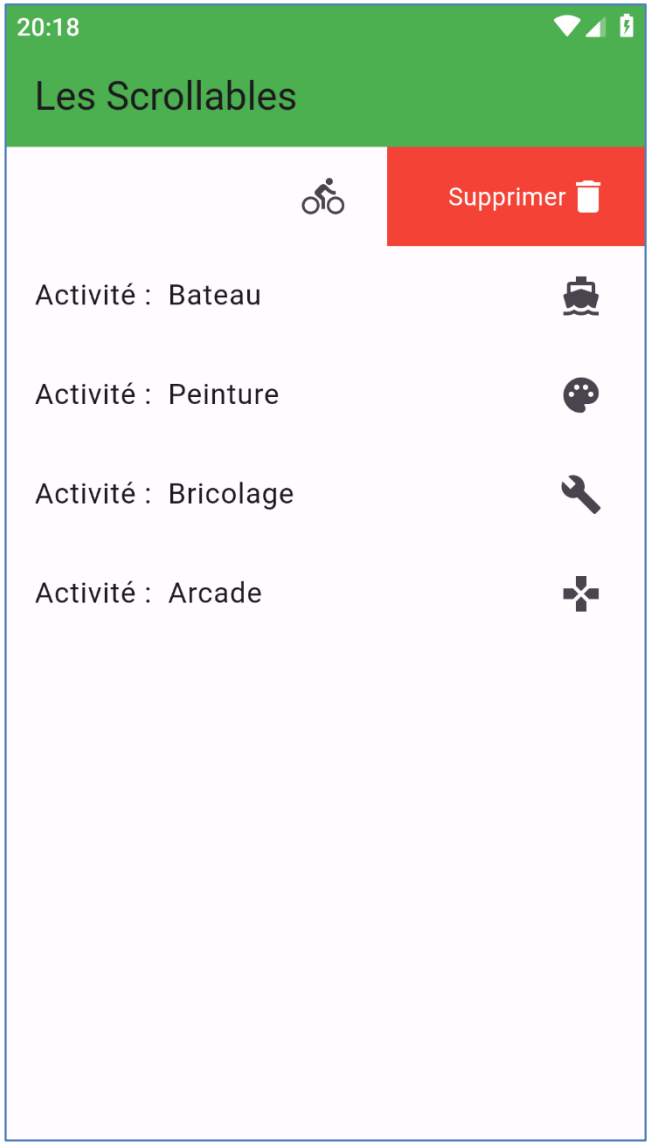
- ✓ Dans le contexte d'une **ListView**, un **Dismissible** est utilisé pour créer une interaction de balayage où l'utilisateur peut faire disparaître des éléments de la liste en les faisant glisser latéralement.

```dart
@override
Widget build(BuildContext context) {
  return Scaffold(
      appBar: AppBar(
        backgroundColor: Colors.green,
        title: Text(widget.title),
      ),  // AppBar
      body: Center(
        child: ListView.builder(
          itemCount: myActivities.length,
          itemBuilder: (context, i) {
            Activity activity = myActivities[i];
            String key = activity.nom;
            return Dismissible(
                key: Key(key),
                background: Container(
                  color: Colors.red,
                ),  // Container
                child: ListTile(
                  title: Text("Activité :  ${activity.nom}"),
                  trailing: Icon(activity.icone),
                ),  // ListTile
                );  // Dismissible
          },
        ), )); }}  // ListView.builder, Center, Scaffold
```



Dismissible

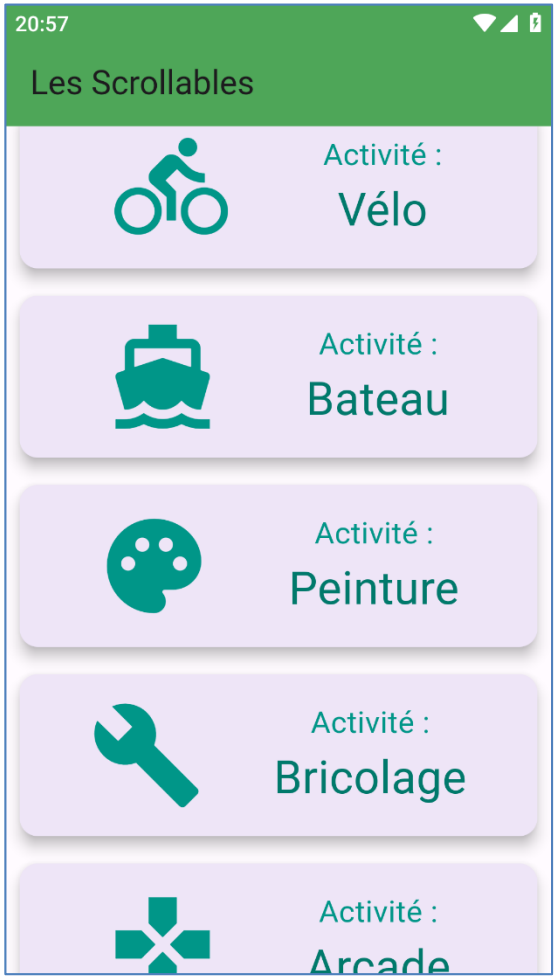
```dart
return Dismissible(
    key: Key(key),
    background: Container(
      padding: const EdgeInsets.only(right: 20.0),
      color: Colors.red,
      child: const Row(
        mainAxisAlignment: MainAxisAlignment.end,
        children: [
          Text("Supprimer", style: TextStyle(color: Colors.white),),
          Icon(Icons.delete, color: Colors.white,)
        ],
      ),  // Row
),  // Container
    onDismissed: (direction){
      setState(() {
        myActivities.removeAt(i);
      });
    },
    child: ListTile(
      title: Text("Activité :  ${activity.nom}"),
      trailing: Icon(activity.icone),
    ),  // ListTile
    );  // Dismissible
```



Un Tile personnalisé

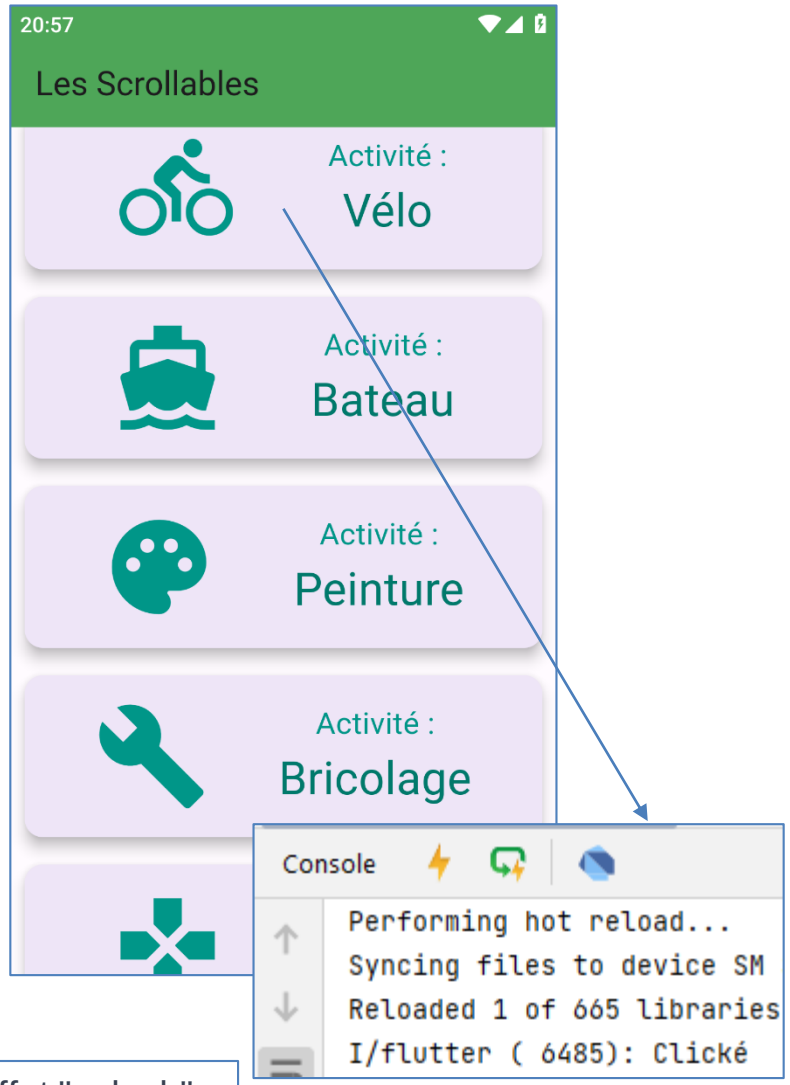
- Remplacer ListTile par ce Container :

```
child: Container(
  padding: const EdgeInsets.all(5.0),
  height: 125.0,
  child: Card(
    elevation: 7.5,
    child: Row(
      mainAxisAlignment: MainAxisAlignment.spaceEvenly,
      children: [
        Icon(
          activity.icone,
          color: Colors.teal,
          size: 75.0,
        ),  // Icon
        Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            const Text(
              "Activité :",
              style: TextStyle(
                  color: Colors.teal, fontSize: 20.0),  // TextStyle
            ),  // Text
            Text(
              activity.nom,
              style: TextStyle(
                  color: Colors.teal[700], fontSize: 30.0),  // TextStyle
            )  // Text
          ],
        )  // Column
      ],
    ),  // Row
  ),  // Card
));  // Container, Dismissible
```



Gestion du clic

```
child: Container(
  padding: const EdgeInsets.all(5.0),
  height: 125.0,
  child: Card(
    elevation: 7.5,
    child: InkWell(
      onTap: (() => print("Clické")),
      child: Row(
        mainAxisAlignment: MainAxisAlignment.spaceEvenly,
        children: [
          Icon(
            activity.icone,
            color: Colors.teal,
            size: 75.0,
          ),  // Icon
          Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              const Text(
                "Activité :",
                style: TextStyle(
                    color: Colors.teal, fontSize: 20.0),  // TextStyle
              ),  // Text
              Text(
                activity.nom,
                style: TextStyle(
                    color: Colors.teal[700], fontSize: 30.0),  // TextStyle
              )  // Text
            ],
          ) ], ), )), )  // Column, Row, InkWell, Card, Container
```



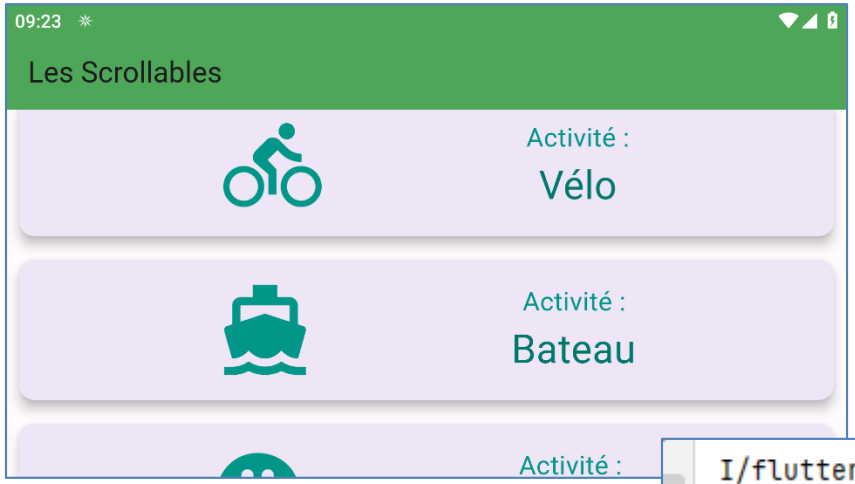
Lorsque l'utilisateur appuie sur la zone couverte par l'InkWell, l'effet "splash" est affiché, donnant une rétroaction visuelle à l'utilisateur. Cela simule l'apparence d'encre se propageant au point d'appui, d'où le nom "InkWell"

Identifier la rotation

- ✓ Dans Flutter, vous pouvez utiliser la propriété **MediaQuery.of(context).orientation** pour identifier l'orientation actuelle du smartphone.

- ✓ Cette propriété renvoie un objet Orientation, qui peut prendre deux valeurs : **Orientation.portrait** et **Orientation.landscape**

```
@override
Widget build(BuildContext context) {
  Orientation orientation = MediaQuery.of(context).orientation;
  print(orientation);
  return Scaffold(
```

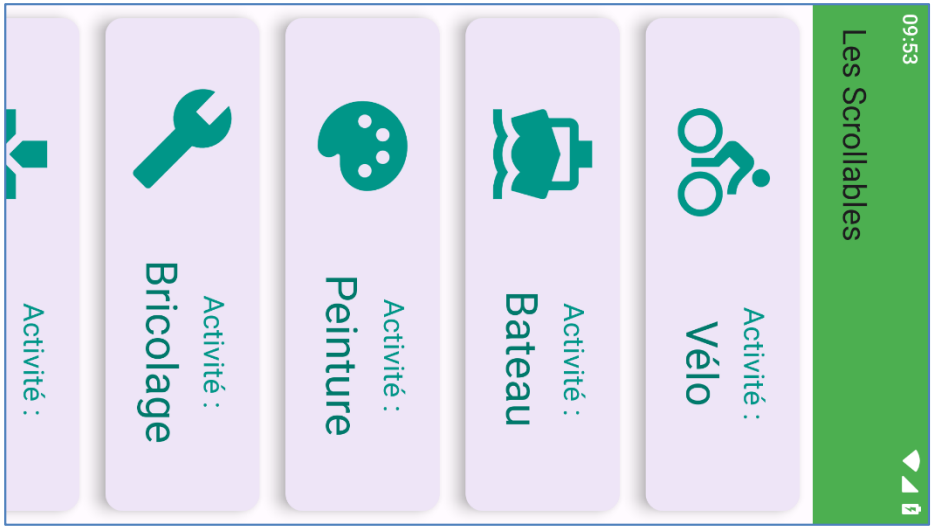



```
I/flutter ( 3840): Orientation.landscape
```

Bloquer l'orientation

- Dans Flutter, vous pouvez bloquer l'orientation du smartphone en utilisant la méthode **SystemChrome.setPreferredOrientations** fournie par la classe **SystemChrome**.

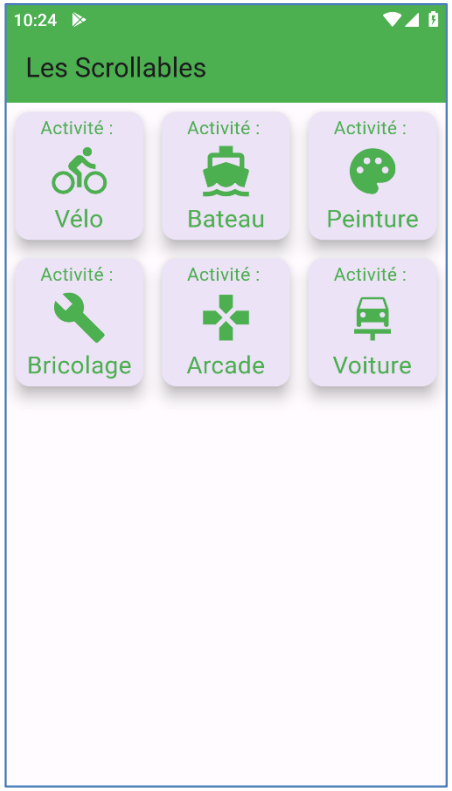
```dart
void main() {
  runApp(const MyApp());
  SystemChrome.setPreferredOrientations([
    DeviceOrientation.portraitUp,
    DeviceOrientation.portraitDown,
  ]);
}
```



GridView.builder

- ✓ **GridView.builder** est un widget utilisé pour afficher une grille (grid) de widgets de manière efficace.
- ✓ Cela vous permet de créer une liste scrollable de widgets dans une disposition en grille.
- ✓ Plutôt que de créer tous les widgets en une seule fois, **GridView.builder** construit dynamiquement les widgets à mesure qu'ils deviennent visibles à l'écran, ce qui améliore les performances et la consommation de mémoire.

```
body: Center(
    child: GridView.builder(
        gridDelegate: const SliverGridDelegateWithFixedCrossAxisCount(
            crossAxisCount: 3),  // SliverGridDelegateWithFixedCrossAxisCount
        itemCount: myActivities.length,
        itemBuilder: (context, i) {
          return Container(
            margin: const EdgeInsets.all(3.5),
            child: Card(
              elevation: 10.0,
              child: Column(
                mainAxisAlignment: MainAxisAlignment.spaceEvenly,
                children: [
                  const Text(
                    "Activité : ",
                    style:
                        TextStyle(color: Colors.green, fontSize: 15.0),
                  ), // Text
                  Icon(
                    myActivities[i].icone,
                    color: Colors.green,
                    size: 45.0,
                  ), // Icon
                  Text(
                    myActivities[i].nom,
                    style: const TextStyle(
                        color: Colors.green, fontSize: 20.0),  // TextStyle
                    )  // Text
                ],
              ), // Column
            ), // Card
          ); // Container
        })) // GridView.builder, Center
```



Gestion de la rotation

- ✓ Je veux afficher la liste en mode portrait et la grille en mode paysage:
- ✓ Je commence par mettre les deux codes (liste et grilles) dans des fonctions:

```
Widget grille(){
  return GridView.builder(
      gridDelegate: const SliverGridDelegateWithFixedCrossAxisCount(
          crossAxisCount: 4),  // SliverGridDelegateWithFixedCrossAxis
```

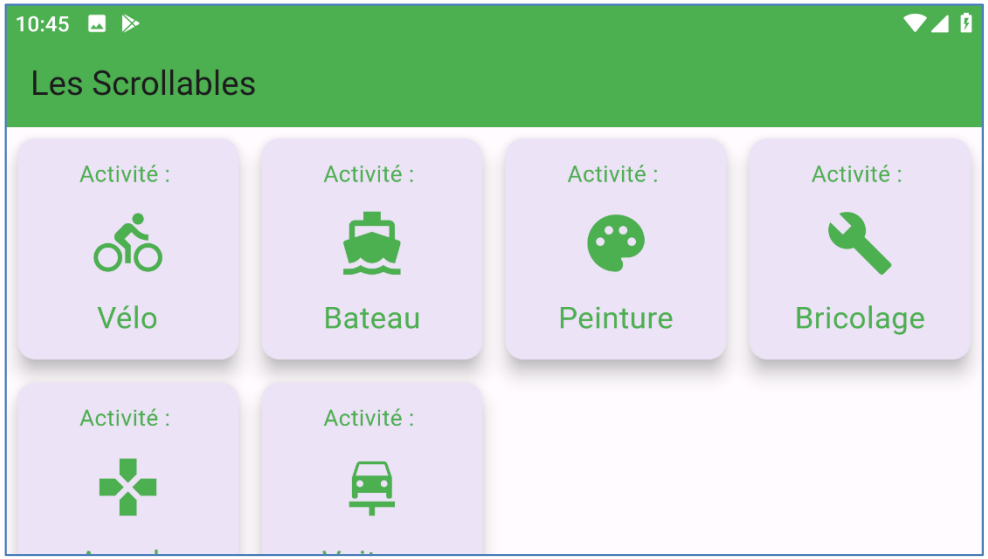
```
Widget liste() {
  return ListView.builder(
    itemCount: myActivities.length,
    itemBuilder: (context, i) {
```

- ✓ Ensuite afficher selon le mode :

```
return Scaffold(
    appBar: AppBar(
      backgroundColor: Colors.green,
      title: Text(widget.title),
    ),  // AppBar
    body: Center(
        child: (orientation==Orientation.portrait)? liste(): grille(),
    )  // Center
);  // Scaffold
```

Gestion de la rotation

- Résultat:



Références

- ✓ Sanjib Sinha. Beginning Flutter with Dart: A Step by Step Guide for Beginners to Build an Android or iOS Mobile Application (Flutter, Dart and Algorithm), 2021.
- ✓ Mike Katz et al. Flutter Apprentice Learn to Build Cross-Platform Apps, 2nd Edition, 2021.
- ✓ Dieter Meiller. Modern App Development with Dart and Flutter 2. 2021