Notez que `TabBarDemo` n'est pas placé immédiatement au-dessus de `TabBar`.

En fait, `TabBar` va remonter l'arbre des `widgets` pour trouver le `DefaultTabController` le plus proche.

`TabBar` permet de paramétrer les onglets. Il prend bien sûr une propriété `tabs` pour définir les onglets, mais également d'autres propriétés, principalement :

`indicatorColor` : permet de définir la couleur de l'indicateur de sélection.

`indicatorPadding` : permet de définir le `padding` autour de l'indicateur de sélection.

`indicatorWeight` : permet de définir l'épaisseur de l'indicateur de sélection.

`isScrollable` : permet de définir si les onglets sont scrollables horizontalement.

`labelColor` : permet de définir la couleur du label sélectionné.

`unselectedLabelColor` : permet de définir la couleur des labels non sélectionnés.

La propriété `tabs` prend en argument une `list` de `Widget Tab` qui peuvent contenir une propriété `text`, `icon`, ou encore un `child` de type `Widget`.

Enfin, il faut définir un `TabBarView` qui va définir le contenu des onglets. Il faut bien sûr que l'index du contenu des onglets corresponde à celui de `tabs`.

Autrement dit, il faut que l'index sur la list `TabBarView` corresponde à l'index sur la propriété `tabs` de `TabBar`.

Utilisation dans l'application

Nous allons utiliser des onglets dans notre `Widget TripsView` :

```
import 'package:flutter/material.dart';
import 'widgets/trip_list.dart';
import '../../models/trip_model.dart';
import '../../widgets/dyma_drawer.dart';

class TripsView extends StatefulWidget {
  static const String routeName = '/trips';
  final List<Trip> trips;

  const TripsView({super.key, required this.trips});

  @override
  State<TripsView> createState() => _TripsViewState();
}
```