TP 1 - Une première application

Installer Flutter

- Téléchargez le SDK de Flutter. Install Flutter SDK
- Décompressez le fichier dans le répertoire C:/
- Dans la barre de recherche windows en bas à gauche, entrez "variables d'environnement" et choisissez celles qui concernent votre compte seulement.
- Choisissez PATH et ajoutez-y le dossier bin du dossier où avez stocké flutter.

Vous pouvez maintenant utiliser Flutter dans un terminal, placez-vous dans un nouveau dossier et lancez la commande

```
flutter create premiereapp
```

L'application de Base

Flutter va générer les fichiers nécessaires à son fonctionnement. Il y a beaucoup de dossiers / fichiers, mais vous avez l'habitude. Le seul qui nous intéresse est le dossier `lib` . Remarquez qu'il y a un dossier pour ios, un pour android, un pour le web.

Ouvrez le dossier avec VS Code et installez l'extension pour Flutter.

L'application de base générée par Flutter est compliquée pour une première, remplaçons quelques fichiers pour commencer par des choses simples.

Commencer avec une application simple

Nous allons modifier 3 fichiers :

1. Remplacez votre fichier `pubspec.yaml` (équivalent du `package.json` de React) par ce contenu

```
name: namer_app

description: A new Flutter project.

publish_to: 'none'

version: 0.0.1+1

environment:
  sdk: '>=2.18.4 <3.0.0'

dependencies:
```

```
  flutter:
    sdk:
      flutter

  english_words: ^4.0.0
  provider: ^6.0.0

dev_dependencies:
  flutter_test:
    sdk:
      flutter

  flutter_lints: ^2.0.0

flutter:
        uses-material-design: true
```

Avec ces modifications, nous ajoutons deux packages : *english_words* et *provider*.

2. Remplacez le fichier `analysis_options.yaml` par

```
include: package:flutter_lints/flutter.yaml

linter:
   rules:
      prefer_const_constructors: false
      prefer_final_fields: false
      use_key_in_widget_constructors: false
      prefer_const_literals_to_create_immutables: false
      prefer_const_constructors_in_immutables: false
      avoid_print: false
```

Ceci assouplira les règles lors de l'analyse du code. Si on veut développer des app en production, nous rendrons ces règles un peu plus strictes.

3. Modifiez le contenu de l'application dans le fichier main.dart par :

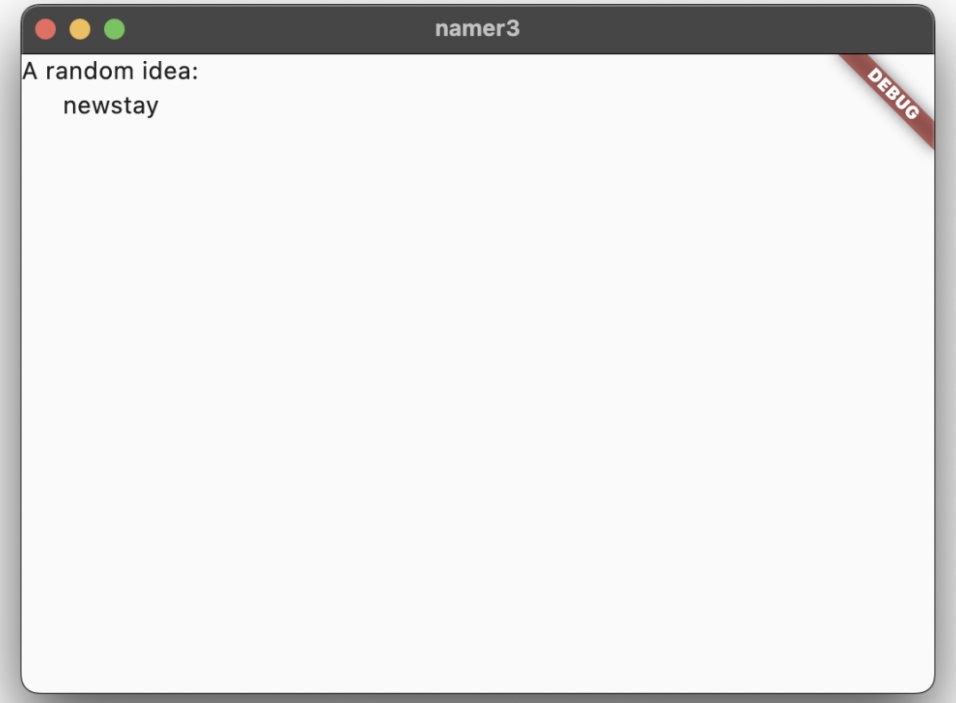
```
import 'package:english_words/english_words.dart';
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';

void main() {
        runApp(MyApp());
}

class MyApp extends StatelessWidget {
        const MyApp({super.key});
        @override
        Widget build(BuildContext context) {
                return ChangeNotifierProvider(
                        create: (context) => MyAppState(),
                        child: MaterialApp(
                                title: 'Namer App',
                                theme: ThemeData(
                                        useMaterial3: true,
                                        colorScheme:
```

```
ColorScheme.fromSeed(seedColor: Colors.deepOrange),
                                ),
                                home: MyHomePage(),
                            ),
                    );
            }
}

class MyAppState extends ChangeNotifier {
        var current = WordPair.random();
}

class MyHomePage extends StatelessWidget {
        @override
        Widget build(BuildContext context) {
                var appState = context.watch<MyAppState>();
                var pair = appState.current;

                return Scaffold(
                        body: Column(
                                children: [
                                        Text('A random idea:'),
                                        Text(pair.asLowerCase),
                                ],
                        ),
                );
        }
}
```

Pour lancer l'application, entrez dans la console :

```
flutter run
```

Vous devriez obtenir quelque chose qui ressemble à



L'appli va générer une paire de mots aléatoires et les associer.

Commencer à regarder un peu le code de l'application. Comme Flutter permet de faire plein de choses, la courbe d'apprentissage est un peu raide, mais nous allons la gravir par étape. Ne vous laissez pas submerger par le code.

Combien de classes contient-il ? Quelles lignes correspondent à la partie visible de l'application ? Allez relire dans le cours à quoi correspond Scaffold.

Le composant `MyAppState` contient l'équivalent d'un state React, les autres classes vont l'écouter et lorsqu'il sera modifié, on pourra les notifier pour qu'ils adaptent leur contenu.

Premières modifications

Flutter vient avec une compilation active, essayez de modifier la phrase "A random idea" puis avec la fenêtre du terminal active, appuyez sur la touche "r".

4. Vous pouvez également changer `asLowerCase` en d'autres casses, par exemple `asSnakeCase`, VSCode devrait vous proposer les autres possibilités. Que se passe-t-il si on remplace `Column` par `Row` ?
5. Nous allons ajouter un bouton pour générer une nouvelle paire de mots. Ajoutez-le à la liste des enfants du widget `Column`. Basez-vous sur l'exemple du cours. Comme action, vous pouvez faire afficher un message dans la console avec

```
print('le bouton a été pressé!');
```

6. Dans le composant `MyAppState`, écrivez une fonction pour générer une nouvelle paire de mots. Pour cela, vous devrez seulement modifier la variable `current` et ajoutez la ligne

   ```
   notifyListeners();
   ```

7. Appelez ensuite cette fonction pour qu'elle se déclenche lorsqu'on clique sur le bouton.

Une meilleure interface

Pour améliorer l'interface, nous allons tout d'abord créer un nouveau widget pour remplacer le simple `Text` qui affiche les deux mots. VS Code peut le faire pour nous: Cliquez droit sur le Text, puis refactor -> `Extract Widget`. Appelez ce nouveau Widget `BigCard`.

En répetant cette opération de clic droit sur un widget :

- Ajoutez du padding autours du texte avec le `Wrap with Padding`. Vous pouvez augmenter la taille du padding.
- Puis entourez-le d'un widget Card avec le `Wrap with Widget`

L'application possède aussi un contexte global pour le thème : Ajoutez les deux lignes du code suivant.

```
...
Widget build(BuildContext context) {
                var theme = Theme.of(context);

                return Card(
                        color: theme.colorScheme.primary,
                        ...
```

Cherchez à quel endroit du code la couleur du thème principal est définie et changez-la. Remarquez la transition douce entre les deux couleurs lorsque vous mettez à jour l'application. Remarquez aussi que le texte de votre bouton change de couleur.

Changez le style du texte en vous aidant des exemples de widget du cours. Vous pouvez changer la taille et la couleur. Pour la couleur, si vous la réglez à la main, la couleur du thème ne sera plus forcément compatible avec votre couleur. Le thème possède en fait plusieurs couleurs qui vont bien ensemble, pour le texte, vous pouvez choisir `theme.colorScheme.onPrimary`.

Pour centrer verticalement la carte, vous pouvez ajouter la propriété suivante au widget `Column`:

```
mainAxisAlignment: MainAxisAlignment.center,
```

Entourez tout votre widget `Colmun` d'un Widget `Center` pour centrer horizontalement le contenu. Ce widget fonctionne comme le widget `Card`.

Vous pouvez également supprimer le premier texte, l'interface est maintenant assez explicite. Vous pouvez également ajouter un widget `SizedBox` entre votre `BigCard` et votre bouton. Ce Widget fait simplement occuper de l'espace, cela fera respirer votre page.

Ajouter des couples de mots en favoris

Nous allons ajouter sur l'écran un bouton pour mettre en favori un couple de mots. Plus tard, nous ajouterons également une page pour lister tous les mots que nous avons mis en favoris.

Dans la classe `MyAppState` ajoutez une variable `favorites` qui sera une liste de `WordPair`.

Ajoutez une fonction `toggleFavorite` qui ajoute au tableau la paire de mots courante si elle n'y est pas déjà et qui l'enlève s'il y est. Sur les listes, vous avez accès aux méthodes suivantes : - contains - remove - add

Ajoutez un bouton pour ajouter une paire de mots aux favoris.

Entourez ces deux boutons du bon widget pour qu'ils soient l'un à côté de l'autre.

Pour dire à ce composant de ne pas prendre tout l'espace disponible, on peut ajouter la propriété

```
mainAxisSize: MainAxisSize.min,
```

Nous allons faire en sorte que ce bouton soit différent si la paire de mots est déjà en favoris ou non. Pour cela, déclarer une variable `icon` de type `IconData` au même niveau que la variable `pair`. Cette variable vaudra `Icons.favorite` si la paire est déjà en favoris, `Icons.favorite_border` si la paire de mots n'y est pas.

Vous pourrez alors ajouter au bouton qui ajoute aux favoris, la propriété suivante

```
icon: Icon(icon),
```

Si vous voulez aussi ajouter un texte, vous pourrez ajouter une propriété `label` avec un widget `Text`.

Ajoutez un troisième bouton pour modifier la casse lorsqu'on clique sur le bouton, la casse alternera entre lowercase, snake_case et camelCase.

Une nouvelle page

Nous allons créer une autre page pour voir la liste des mots favoris. Remplacer votre composant `MyHomePage` par le code suivant :

```
class MyHomePage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Row(
        children: [
```

```
          SafeArea(
            child: NavigationRail(
              extended: false,
              destinations: [
                NavigationRailDestination(
                  icon: Icon(Icons.home),
                  label: Text('Home'),
                ),
                NavigationRailDestination(
                  icon: Icon(Icons.favorite),
                  label: Text('Favorites'),
                ),
              ],
              selectedIndex: 0,
              onDestinationSelected: (value) {
                print('selected: $value');
              },
            ),
          ),
          Expanded(
            child: Container(
              color: Theme.of(context).colorScheme.primaryContainer,
              child: GeneratorPage(),
            ),
          ),
        ],
      ),
    );
  }
}


class GeneratorPage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    var appState = context.watch<MyAppState>();
    var pair = appState.current;

    IconData icon;
    if (appState.favorites.contains(pair)) {
      icon = Icons.favorite;
    } else {
      icon = Icons.favorite_border;
    }

    return Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          BigCard(pair: pair),
          SizedBox(height: 10),
          Row(
            mainAxisSize: MainAxisSize.min,
            children: [
              ElevatedButton.icon(
                onPressed: () {
                  appState.toggleFavorite();
                },
                icon: Icon(icon),
```

```
                label: Text('Like'),
              ),
              SizedBox(width: 10),
              ElevatedButton(
                onPressed: () {
                  appState.getNext();
                },
                child: Text('Next'),
              ),
            ],
          ),
        ],
      ),
    );
  }
}
```

Remarquez que tout le code anciennement contenu dans `MyHomePage` se trouve maintenant dans un nouveau Widget `GeneratorPage`, excepté le `Scaffold`.

- Quels sont les deux enfants du Widget `Row` du composant `MyHomePage` ?
  - Le widget `SafeArea` assure seulement qu'il n'y a pas de chevauchement
  - `Expanded` fait en sorte que l'espace pris soit bien celui qui est nécéssaire. C'est dans ce widget que la page `GeneratorPage` est appelée.
- Quels sont les deux enfants de la propriété `destinations` du Widget `NavigationRail` ? A quoi correspondent-ils ?
- Que se passe-t-il si on change la propriété `extended` à `true` ? Plus tard, nous apprendrons à faire cela lorsque la largeur sera suffisament grande.

C'est la propriété `selectedIndex` qui permet de naviguer d'une page à l'autre.

- S'il est à 0, c'est la première destination qui sera choisie.
- S'il est à 1, c'est la seconde, ...

Essayez de le changer à 1, vous verrez que le second menu est sélectionné. Pour le moment, c'est toujours le composant `GeneratorPage` qui s'affiche.

La propriété `onDestinationSelected` déclenche une fonction lorsqu'une destination est sélectionnée. Pour le moment elle affiche seulement le numéro du choix dans la console.

Jusqu'à maintenant, les variables de notre application (la paire de mots et la liste des favoris) était géré dans la classe `MyAppState`. Nous allons avoir besoin de stocker d'autres variables (`selectedIndex` parmi d'autres) Nous pourrions le mettre dans la même classe, mais seul le composant `MyHomePage` a besoin de cette variable, voyons un moyen de faire cela.

Jusqu'à maintenant, nous avons manipulé des classes qui étendaient **StatelessWidget**. Il existe aussi une classe **StatefulWidget**, qui correspond à une classe qui possède son propre state.

Cliquez droit sur la classe `MyHomePage`, refactor -> turn into Statefull Widget.

Une nouvelle classe a été créée, `_myHomePage` (le _ rend la classe privée.)

- Déclarez une variable `selectedIndex` qui vaut initialement 0 dans la classe `_MyHomePage` avant la méthode build.
- A la place du 1 codé en dur pour la propriété `selectedIndex`, insérez cette nouvelle variable.
- A la place du print qui se déclenche lorsqu'on sélectionne une autre page, insérez le code suivant :

```
setState( () {
  selectedIndex = value;
  }
)
```

Cette syntaxe `() { }` est l'équivalent des fonctions anonymes de javascript `() => { }`

Vous pouvez maintenant sélectionner un onglet différent dans la barre de navigation. Pour changer ce qui s'affiche à l'écran en fonction de l'onglet sélectionner, ajoutez une variable `page` de type `Widget` qui vaut

- `GeneratorPage()` si `selectedIndex` vaut 0,
- `Placeholder()` si `selectedIndex` vaut 1,
- Si `selectedIndex` vaut une autre valeur, vous pourrez déclencher une exception, par exemple avec le code

```
throw UnimplementedError('no widget for $selectedIndex');
```

Modifiez le code au bon endroit pour afficher la bonne page lorsqu'on clique sur le bon onglet.

Codez la page pour afficher la liste des favoris. Vous pourrez vous baser sur cet exemple

```
var messages = ['Hello', 'Salut', 'Ola']

return Column(
        children: [
                Text('Messages: '),
                for (var msg in messages)
                        Text(msg),
        ]
)
```

Allez voir cette version évoluée de cette application, remarquez le responsive.
https://dartpad.dev/?id=e7076b40fb17a0fa899f9f7a154a02e8

Identifiez les morceaux du code qui sont responsable de ce responsive et intégrez-le à votre code.

Ajoutez une troisième page qui détaillerait des conditions légales d'utilisation de l'app. Vous pouvez la remplir avec des titres de section, l'un à gauche, l'autre à droite et de jolis lorem ipsum.