Université de Montpellier
Master 2 Informatique Génie Logiciel

Développement mobile avancé, IoT et embarqué

Création d'interfaces graphiques avec Flutter

Rapport du TP1

Étudiante :
GARCIA Léa

Chapitre 1

Lien vers dépôt GitHub

Lien vers dépôt GitHub https://github.com/Ravenn2203/RenduTP1Flutter.git

Chapitre 2

Structure du projet

2.1 Classes du projet

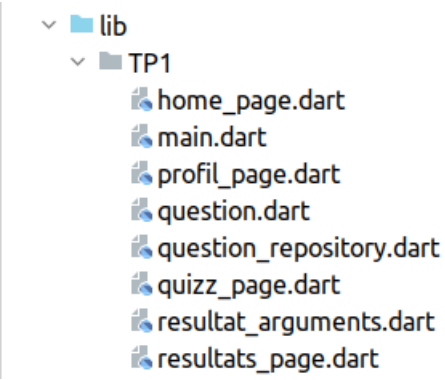

FIGURE 2.1 – Classes permettant de structurer le projet

2.2 La classe main.dart

C'est dans la classe main.dart que j'ai défini, via le widget MaterialApp contenant toute l'application, la propriété 'routes'. Ainsi, pour les besoins du TP j'ai défini quatre routes que j'ai rangé dans un dictionnaire et affecté à 'routes'. On retrouve par exemple la route '/profil' qui lorsqu'elle est appelée, crée le widget ProfilPage, l'ajoute à la pile et l'affiche à l'écran. Nous verrons dans la suite de ce rendu comment j'ai utilisé ces routes.

2.3 Page d'accueil

Afin d'avoir une application plus complète j'ai créé une page d'accueil permettant d'accéder aux deux exercices. Voici donc l'écran d'accueil au démarrage :

FIGURE 2.2 – Ecran d'accueil de l'application

Cette page est une classe mais aussi un widget stateless. L'arbre de widgets est constitué uniquement de l'appBar (appBar widget) et de deux boutons texte (TextButton widget) rangés dans une colonne (Column widget), elle même contenue dans un widget Padding permettant de mettre de l'espace à l'intérieur.

Chapitre 3

Exercice 1 : Profil Card

Pour réaliser cette page de profil j'ai utilisé principalement les widgets Scaffold, Container, BoxDecoration, Stack, Column, Row, SizedBox et CircleAvatar. Voici le rendu graphique :

FIGURE 3.1 – Page de profil

Pour réaliser ceci j'ai découpé le code et extrait les fonctions 'getCard()' et 'getAvatar()'. Avant de décrire ces fonctions voici ci-dessous, le code qui permet de superposer les deux widgets retournés par les fonctions. J'ai utilisé le widget Flutter Stack qui permet de superposer ses children et j'ai utilisé le widget Positioned pour placer correctement mes

deux widgets.

```
body: Padding(
  padding: const EdgeInsets.all(20.0),
  child: Stack(children: [
    Positioned(top: 120, child: _getCard()),
    Positioned(top: 20, left: 50, child: _getAvatar()),
  ]),  // Stack
),  // Padding
```

FIGURE 3.2 – Utilisation du widget Stack pour la page de profil

3.1 La fonction getCard()

Pour créer la card du profil j'ai décidé d'utiliser un Container associé à un BoxDecoration plutôt qu'un widget Card, que j'ai utilisé dans le second exercice. Le BoxDecoration permet d'ajouter de la décoration au widget Container avec la propriété 'decoration' et d'arrondir les bords. J'ai utilisé un widget Padding pour mettre de l'espace à l'intérieur du container et au milieu j'ai mis une Column mettant plusieurs widget Text, SizedBox ainsi qu'un widget Row contenant un widget Icon et un Text, les uns sur les autres.

```
Container _getCard() {
  return Container(
    decoration: BoxDecoration(
        color: Colors.grey[600],
        borderRadius: BorderRadius.all(Radius.circular(25))),  // BoxDecoration
    child: Padding(
      padding: const EdgeInsets.fromLTRB(40, 180, 30, 50),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          Text('NOM, PRÉNOM',
```

FIGURE 3.3 – Création de la carte du profil avec différents widgets

3.2 La fonction getAvatar()

Pour mettre une photo de profil j'ai utilisé à deux reprises le widget CircleAvatar. Le premier permet d'entourer le second d'un rebord gris car il a une propriété radius égale à 125 et le second une propriété plus petite à 120 et un background gris. Le second CircleAvatar contient quant à lui l'image du profil.

Chapitre 4

Exercice 2 : Quizz

4.1 Question et QuestionRepository

Pour structurer le quizz j'ai utilisé deux classes, la classe Question qui structure mes questions avec une image (String représentant le chemin vers le dossier d'assets), un texte de question et un booléen pour savoir si le texte énonce quelque chose de juste ou faux. J'ai utilisé la classe QuestionRepository pour stocker dans une liste mes questions et pour définir une fonction qui récupère la première question et une fonction qui renvoie la question suivante en fonction du numéro de celle qui vient d'être posée.

4.2 La page de quizz

J'ai regroupé le code des différentes parties de la page dans des fonctions : getQuestionCard(), getAnswerButtons(), checkAnswer(), getImage() et j'ai arrangé le tout dans un widget Column que j'ai lui même mis comme child d'un widget Padding pour positionner le tout comme on peut le voir dans la figure 4.2. J'ai aussi utilisé plusieurs variables dont la valeur est actualisée dans des setState() ce qui permet de mettre à jour l'état de mon widget QuizzPage avec un rappel à la fonction build.

Tout d'abord, je récupère la première question avec la ligne : 'Question question = QuestionRepository().getQuestion() ;' puis dans getImage() et getCard(), j'utilise 'question.imagePath' et 'question.questionText' pour remplir mes widgets, je ne les décris pas plus car ils ressemblent à ceux créés et déjà discutés dans l'exerice 1.

Ensuite j'ai écrit la fonction getAnswerButtons() qui retourne un widget Row contenant les trois boutons, 'vrai', 'faux' et 'question suivante'. A chacun des boutons j'ai donné du style, une couleur grise au départ mais associée à une variable que je vais par la suite modifier en fonction de l'exactitude ou non de la réponse. Enfin, j'ai donné des instructions en cas de 'onPressed' pour chacun des boutons, pour 'vrai' et 'faux' j'ai fait la même chose (valeurs inversées). Vous pouvez voir par exemple le code associé à la propriété onPressed() du bouton 'vrai' dans la figure 4.1.

Dans cette fonction je valide le fait que l'utilisateur a répondu à la question et pourra passer à la question suivante puis je vérifie la réponse donnée par l'utilisateur avec la fonction checkAnswer() qui récupère via le context, la question posée et donc le booléen associé. Ici on passe true en argument car on est dans le cas du bouton 'vrai' mais pour la bouton 'faux' ce serait 'false'. Ensuite, si la réponse est la bonne j'utilise la fonction

```
onPressed: () {
  if (!reponse) {
    reponse = true;
    if (_checkAnswer(true, context)) {
      setState(() {
        couleurBouttonVrai = Colors.green;
      });
      const snackBar = SnackBar(
        content: Text('Bien joué!'),
        duration: Duration(seconds: 1),
      );  // SnackBar
      ScaffoldMessenger.of(context).showSnackBar(snackBar);
    } else {
      setState(() {
        couleurBouttonVrai = Colors.red;
      });
```

FIGURE 4.1 – Code de la fonction onPressed appelé lors du clic sur le bouton 'vrai'.

setState pour modifier la couleur associée à la variable couleurBouttonVrai (pour du vert) qui une fois changée, va être actualisée dans le design du bouton qui l'utilise. En plus de ceci, j'affiche une SnackBar au bas de l'écran pendant une seconde pour dire 'bien joué' à l'utilisateur, comme on peut le voir à la figure 4.4. Si au contraire l'utilisateur a dit 'vrai' alors que la réponse était 'faux', je change la couleur associée à la variable couleurBouttonVrai (qui était en gris) par du rouge comme on peut le voir dans la figure 4.6.

Enfin, toujours dans la fonction qui concerne les boutons, pour le bouton 'question suivante' j'ai mis dans le code associé à la propriété 'onPressed()', une condition qui empêche de passer à la question suivante sans y avoir répondu, si l'utilisateur essaie alors une snackbar apparaît comme on peut le voir à la figure 4.3. Si l'utilisateur est autorisé et que ce n'était pas la dernière question, j'utilise un setState() pour récupérer la prochaine question et remettre la couleur des boutons en gris. Si c'était la dernière question alors comme nous le verrons dans la dernière section, je passe au widget de la page de résultat auquel je donne des arguments.

Vous trouverez les différents écrans possibles de mon quizz dans la dernière section de ce rapport.

4.3 Le passage d'arguments entre deux widgets avec la classe ResultatArguments

Pour pouvoir afficher le résultat du quizz j'ai créé un dernier widget stateful appelé ResultatsPage mais il a fallut également créer un widget ResultatArguments pour pouvoir stocker les variables de score et les passer du widget QuizzPage au widget ResultatsPage.

Lorsque l'utilisateur clique une dernière fois sur le bouton pour passer à la question suivante, on utilise la fonction setState() et le widget Navigator avec la propriété arguments à laquelle j'affecte donc une instance du widget ResultatArguments. A cette instance j'ajoute la variable contenant le nombre de réponses et j'ajoute la variable contenant le nombre de bonnes réponses de l'utilisateur. Enfin, dans un dernier widget contenant une Card, j'affiche les résultats récupérés avec l'instruction : 'final args = ModalRoute.of(context) !.settings.arguments as ResultatArguments ;'

FIGURE 4.2 – Écran de résultat du quizz

4.4 Figures montrant les différents états de mon application lors du quizz (6 figures)

FIGURE 4.3 – Etat de la page du quizz avant toute réponse de l'utilisateur

FIGURE 4.4 – Etat de la page du quizz lorsque l'on essaie de passer à la question suivante sans réponse, une snackbar apparaît.

FIGURE 4.5 – Etat de la page du quizz lorsque l'on choisit 'vrai' et que la bonne réponse est 'vrai'.

FIGURE 4.6 – Etat de la page du quizz lorsque l'on choisit 'vrai' alors que la bonne réponse est 'faux'.

FIGURE 4.7 – Etat de la page du quizz lorsque l'on choisit 'faux' et que la bonne réponse est 'faux'.

FIGURE 4.8 – Etat de la page du quizz lorsque l'on choisit 'faux' et que la bonne réponse est 'vrai'.