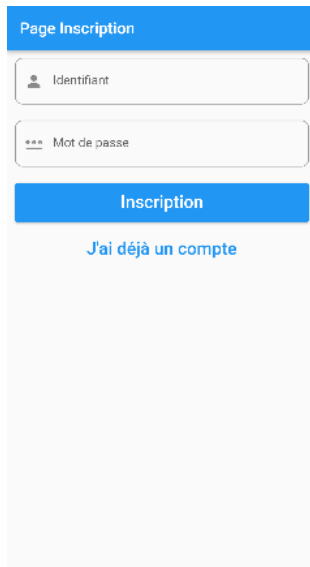


Atelier 3

Enseignant : S. Hadhri

GLID 2

L'objectif de cet atelier est de créer un système d'authentification de notre application qui est formé essentiellement par 3 pages :



Page Inscription

Identifiant

Mot de passe

Inscription

J'ai déjà un compte

Page Inscription



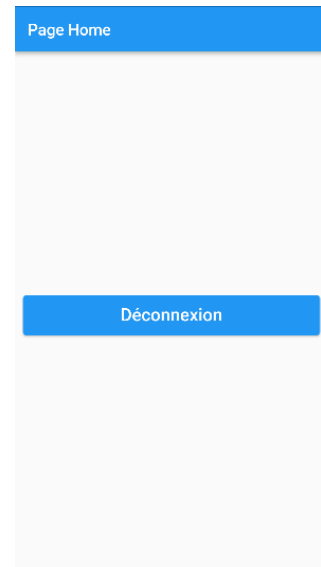
Page Authentification

Sami

Connexion

Nouvel Utilisateur

Page Authentification



Page Home

Déconnexion

Page Home (Accueil)

1. Créer un nouveau projet Flutter nommé « voyage »
2. Sous le dossier « lib », créer un sous-dossier « pages » qui contient 3 fichiers « inscription.page.dart », « authentication.page.dart » et « home.page.dart ».
3. Ecrire le code de ces 3 fichiers permettant d'afficher le nom de chaque page au centre. Ecrire également un code basique dans « main.dart » permettant d'afficher la page « inscription.page.dart »

Fichier inscription.page.dart

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

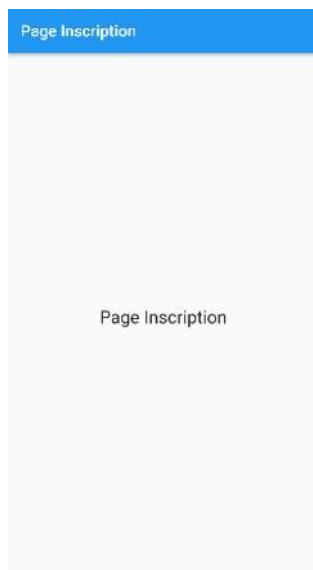
class InscriptionPage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Page Inscription')),
      body: Center(
        child: Text(
          'Page Inscription',
          style: TextStyle(fontSize: 22),
        ),
      ),
    );
  }
}
```

Fichier main.dart

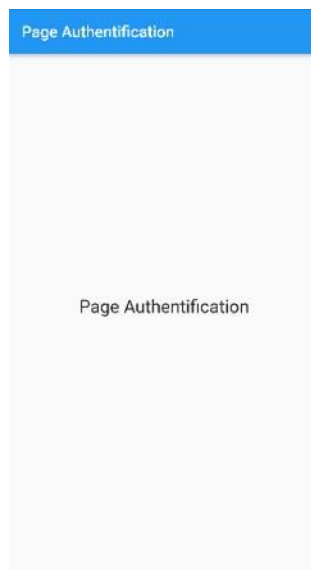
```
import 'package:flutter/material.dart';
import 'package:voyage/pages/inscription.page.dart';

void main() => runApp(MyApp());

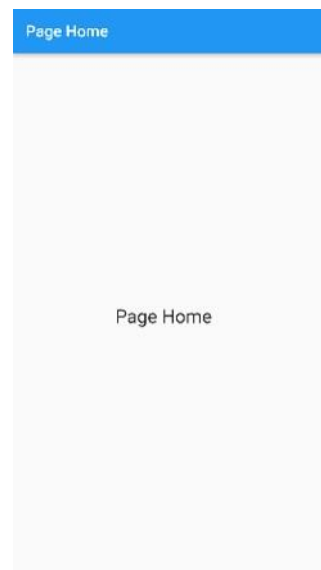
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      home: InscriptionPage(),
    );
  }
}
```



Page Inscription



Page Authentification



Page Home (Accueil)

4. Créer dans la page d'inscription un formulaire formé par 2 zones de texte (widget TextFormField) et un bouton (widget ElevatedButton) organisés dans une colonne.

Fichier inscription.page.dart

```
...
TextEditingController txt_login = new TextEditingController();
...
    Container(
      padding: EdgeInsets.all(10),
      child: TextFormField(
        controller: txt_login,
        decoration: InputDecoration(
          prefixIcon: Icon(Icons.person),
          hintText: "Utilisateur",
          border: OutlineInputBorder( borderSide:
            BorderSide(width: 1), borderRadius:
              BorderRadius.circular(10))),
      ),
    ),
...
    Container(
      padding: EdgeInsets.all(10),
      child: ElevatedButton(
        style: ElevatedButton.styleFrom(
          minimumSize: const Size.fromHeight(50)),
        onPressed: () {},
      ),
    ),
  ),
),
```

```

        child: Text('Inscription', style: TextStyle(fontSize: 22)),
      ),
    ),
  ),
  ...

```

5. Créer dans la page d'authentification un formulaire formé par 2 zones de texte et un bouton.

6. Ajouter un système de routage dans le fichier « *main.dart* » selon le schéma suivant :

- **'/home'** : path vers la page Home
- **'/inscription'** : path vers la page Inscription
- **'/authentification'** : path vers la page Authentification

Fichier main.dart

```

...
class MyApp extends StatelessWidget {
  final routes = {
    '/home': (context) => HomePage(),
    '/inscription': (context) => InscriptionPage(),
    '/authentification': (context) => AuthentificationPage(),
  };

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,

```

```

    routes: routes,
    home: InscriptionPage(),
  },
}

```

7. Ajouter sous les boutons « Inscription » et « Connexion » des liens qui permettent de basculer d'une page à l'autre.

Fichier inscription.page.dart

```

...
TextButton(
  child: Text("J'ai déjà un compte",
    style: TextStyle(fontSize: 22)),
  onPressed: () {
    Navigator.pop(context);
    Navigator.pushNamed(context, '/authentification');
  },
)
...

```



Notion théorique : Stockage des données localement dans Flutter

Le stockage est un aspect très important de toute application. Dans une application Flutter, il existe plusieurs façons de stocker des données hors ligne :

Méthode 1 : Utilisation de fichiers texte/CSV/JSON

Les types de fichiers les plus couramment utilisés pour enregistrer des données sont TXT, CSV et JSON. L'emplacement où placer les fichiers de données est le répertoire Documents d'un appareil. Sur Android, c'est le répertoire AppData et sur iOS, c'est NSDocumentDirectory.

https://pub.dev/packages/path_provider

Méthode 2 : Base de données SQLite

SQLite est une base de données SQL qui peut être utilisée pour stocker des données localement. Il est très utile si vous souhaitez stocker une grande quantité de données de manière ordonnée.

<https://pub.dev/packages/sqlite>

Méthode 3 : Les préférences partagées

Les préférences partagées sont l'un des moyens les plus populaires de stocker des données localement. Il utilise le concept de paires clé-valeur pour stocker les données localement. Les préférences partagées offrent la possibilité de stocker des entiers, des doubles, des flottants et des chaînes. C'est un excellent outil pour stocker par exemple certaines préférences de l'utilisateur comme le thème de l'application ou bien le mode sombre ou le mode clair.

https://pub.dev/packages/shared_preferences

Il existe un concept similaire aux préférences partagées qui s'appelle le stockage sécurisé : il permet de stocker les données sous forme de paires clé-valeur mais avec la seule différence est que dans le stockage sécurisé, les données stockées sont cryptées.

https://pub.dev/packages/flutter_secure_storage

Méthode 4 : Base de données Hive

Hive est un stockage de données local très populaire parmi les développeurs Flutter. Les deux

principales raisons de sa popularité sont qu'il s'agit d'une base de données NoSQL et qu'elle peut non seulement stocker des types de données primitifs, mais également des classes Dart.

Hive est utilisé pour mettre en cache les messages texte d'un chat particulier, pour mettre en cache les détails du profil de l'utilisateur et des données beaucoup plus complexes comme celle-ci.

<https://pub.dev/packages/hive>

Il existe un concept similaire à Hive nommé Objectbox. Il s'agit d'une base de données clé-valeur native Dart. Elle est extrêmement rapide et améliore les taux de réponse tout en permettant des applications en temps réel.

<https://pub.dev/packages/objectbox>

8. On souhaite gérer les utilisateurs à travers les préférences partagées. Développer la méthode privée `onInscrire()` appelée au clic sur le bouton « Inscription » qui permet d'ajouter l'identifiant et le mot de passe dans les préférences partagées puis de basculer vers la page Home.

Remarque :

Pour pouvoir utiliser les préférences partagées, il faudrait ajouter dans la partie « dependencies » du fichier « `pubspec.yaml` » la ligne suivante :

`dependencies:`

`shared_preferences:`

`flutter:`

`sdk: flutter`

Puis lancer la commande : `flutter pub get`

Fichier inscription.page.dart

```
...
class InscriptionPage extends StatelessWidget {
  late SharedPreferences prefs;
  ...
  onPressed: () {
    _onInscrire(context);
  }
  ...
  Future<void> _onInscrire(BuildContext context) async {
    prefs = await SharedPreferences.getInstance();
    if (!txt_login.text.isEmpty && !txt_password.text.isEmpty) {
      prefs.setString("login", txt_login.text);
      prefs.setString("password", txt_password.text);
      Navigator.pop(context);
      Navigator.pushNamed(context, '/home');
    }
  }
  ...
}
```

Remarque :

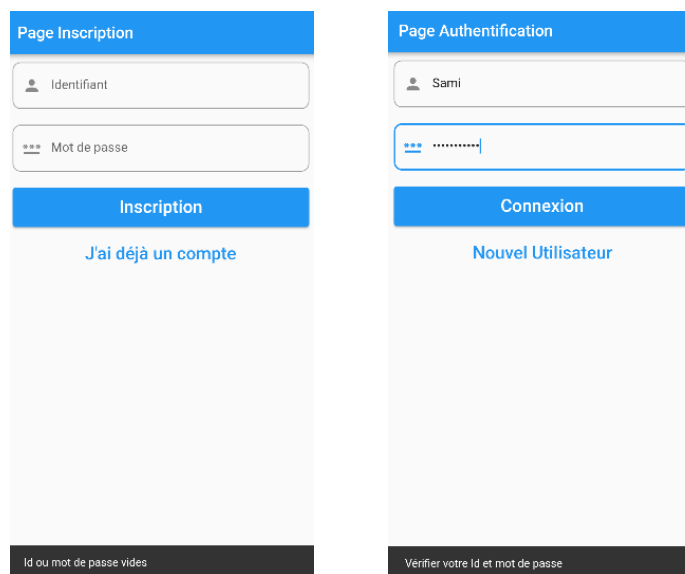
Future est une classe Dart de base pour travailler avec des opérations asynchrones. Un objet Futur représente une valeur ou une erreur potentielle qui sera disponible à un moment donné dans le futur.

9. Développer la méthode privée `_onAuthentifier()` appelée au clic sur le bouton « Connexion » qui permet de vérifier que l'identifiant et le mot de passe saisis sont identiques à ceux enregistrés dans les préférences partagées puis de basculer vers la page Home.

10. Modifier les méthodes `_onInscrire()` et `_onAuthentifier()` de façon à ce qu'un message d'erreur s'affiche si l'identifiant ou le mot de passe sont vides pour l'inscription ou si l'utilisateur n'existe pas pour l'authentification.

Fichier inscription.page.dart

```
Future<void> _onInscrire(BuildContext context) async {  
  ...  
  const snackBar = SnackBar(  
    content: Text('Id ou mot de passe vides'),  
  );  
  ScaffoldMessenger.of(context).showSnackBar(snackBar);  
  ...  
}
```



11. On souhaite maintenant gérer l'état de connexion de l'utilisateur de telle sorte qu'elle démarre automatiquement sur la page Home si l'utilisateur était connecté lors de la dernière utilisation de l'application. Modifier de nouveau les méthodes `_onInscrire()` et `_onAuthentifier()` de façon à enregistrer une variable booléenne « connecte » (qui représente si l'utilisateur est connecté ou non) à True dans les préférences partagées.

Fichier inscription.page.dart

```
Future<void> _onInscrire(BuildContext context) async {  
  ...  
  prefs.setBool("connecte", true);  
  ...  
}
```

12. Ajouter dans la page « Home » un bouton de déconnexion qui permet d'enregistrer la variable booléenne « connecte » à False et de basculer vers la page d'inscription.

Fichier home.page.dart

```
...  
class HomePage extends StatelessWidget {  
  late SharedPreferences prefs;  
  ...  
  body: Center(  
    child: Container(  
      padding: EdgeInsets.all(10),  

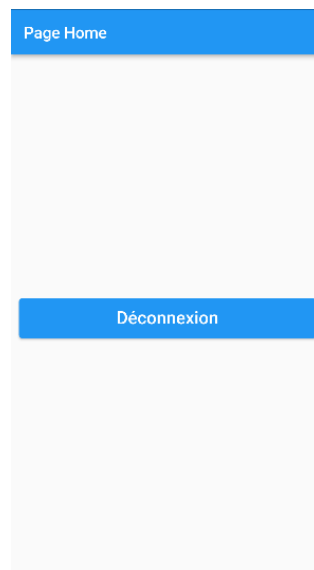
```

```

        child: ElevatedButton(
          style: ElevatedButton.styleFrom(
            minimumSize: const Size.fromHeight(50)),
          onPressed: () {
            _Deconnexion(context);
          },
          child: Text('Déconnexion', style: TextStyle(fontSize: 22))),
      ));
}

Future<void> _Deconnexion(context) async {
  prefs = await SharedPreferences.getInstance(); prefs.setBool("connecte",
  false); Navigator.pushNamedAndRemoveUntil(context, '/authentification',
  (route) =>
false);
}
}

```



13. Modifier la page « *main.dart* » pour ne pas afficher la page Inscription à chaque fois mais plutôt récupérer la variable booléenne « connecte » des préférences partagées et d'afficher :
- La page « Home » si elle vaut True
 - La page « Inscription » si elle vaut False

Fichier main.dart

```

...
home: FutureBuilder(
  future: SharedPreferences.getInstance(),
  builder: (context, snapshot)
  {
    if (snapshot.hasData) {
      bool conn = snapshot.data?.getBool('connecte') ?? false;
      if (conn)
        return HomePage();
    }
    return AuthentificationPage();
  })
...

```