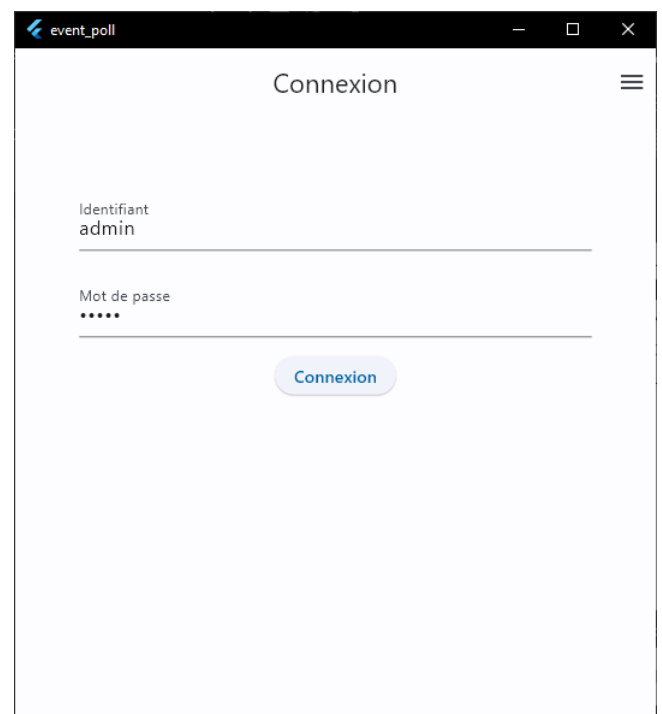
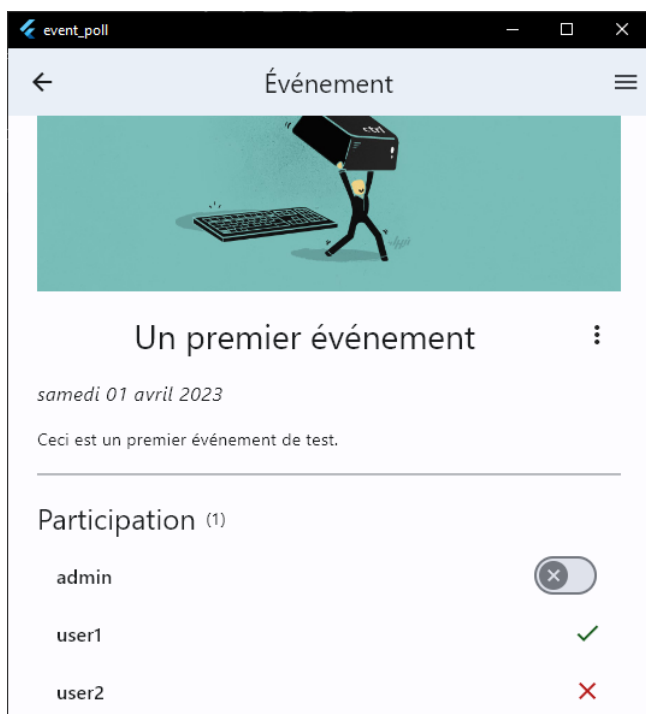
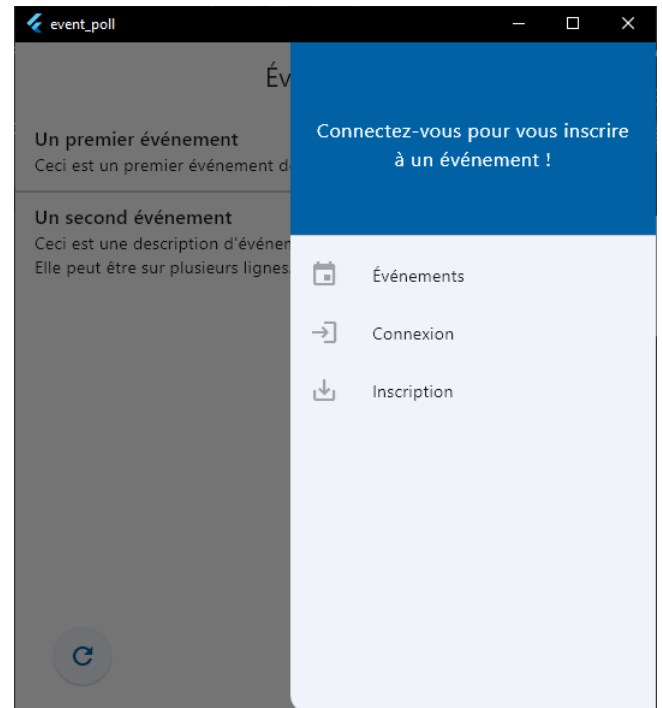
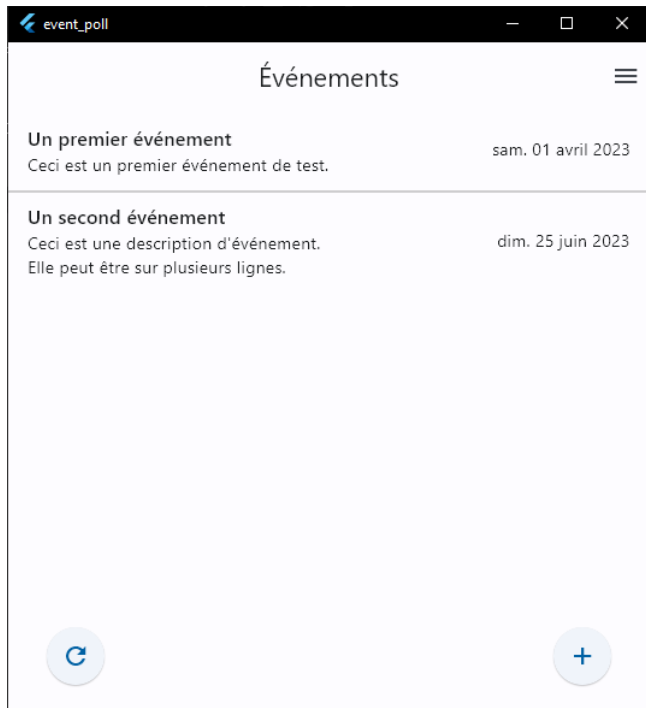


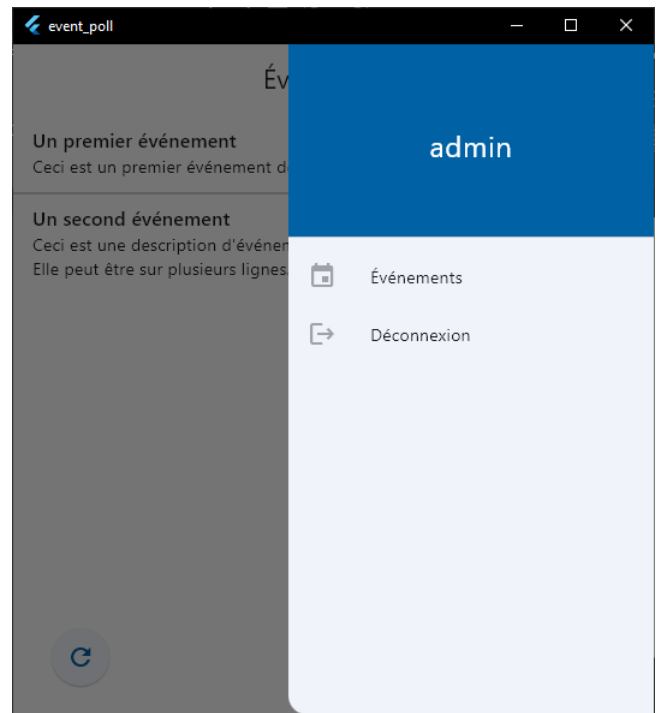
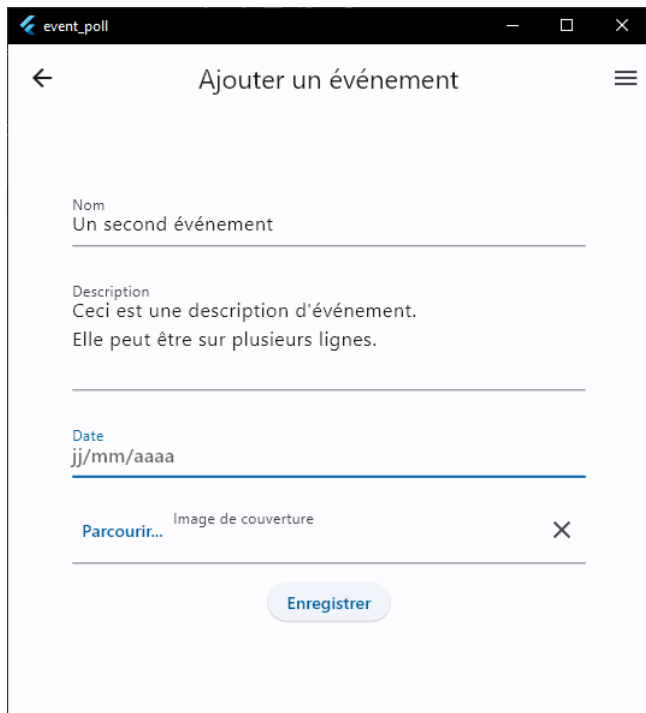
1 Démarrage

1.1 Objectif

Créer une application mobile se connectant à une API et permettant :

- À un administrateur de créer et modifier des événements
- À un utilisateur de se créer un compte et de signaler sa présence à un événement





1.2 Création du projet

S'il n'est pas présent sur votre poste, installez Flutter en utilisant la commande suivante dans un terminal (à la racine du disque D:\) :

```
git clone https://github.com/flutter/flutter.git -b stable
```

Vous pouvez ensuite vérifier la bonne installation de flutter en lançant la commande de vérification :

```
flutter doctor -v
```

Attention : Si votre installation a été réalisée sur un PC pour lequel vous n'êtes pas administrateur, vous ne pouvez pas ajouter la commande `flutter` en variable d'environnement.

Vous devez donc utiliser, au choix :

- Le chemin complet d'installation pour appeler une commande : `D:\flutter\bin\flutter`
- L'utilitaire de commande `flutter_console.bat` à la racine du dossier d'installation de Flutter

Dans un terminal de commande, à la racine de vos projets (sur le disque P:\), utilisez la commande suivante :

```
flutter create event_poll
```

1.3 Initialisation du projet

1.3.1 Installation des dépendances

Pour commencer le projet, installer les dépendances `provider`, `http` et la localisation.

```
flutter pub add provider http
flutter pub add flutter_localizations --sdk=flutter
flutter pub add intl:any
```

Note : Lorsque vous récupérez un projet déjà existant, vous pouvez recharger toutes ses dépendances via la commande : `flutter pub get`

1.3.2 Base de l'UI

- Remplacez le fichier `main.dart` de votre application par celui fourni avec le sujet.
- Supprimez le fichier `tests/widget_test.dart`
- Ajoutez le fichier `app_scaffold.dart` fourni dans le dossier un nouveau dossier `lib/ui/`
- Ajoutez les fichiers `result.dart` et `configs.dart` fournis à la racine du dossier `lib/`

1.3.3 Serveur

Vous pouvez utiliser le serveur en local sur votre PC ou le déployer sur Azure.

Ouvrez la solution `EventPollBackend` et chargez les packages NuGet.

Démarrez le serveur et modifiez si besoin l'URL dans le fichier de configuration du projet Flutter (`lib/configs.dart`).

Au lancement du projet la définition de l'API s'affiche, elle est sinon disponible à l'adresse :

<http://localhost:5240/swagger/index.html>

Le serveur s'initialise au lancement avec 3 utilisateurs (*admin*, *user1*, *user2*), 1 événement et des votes. Pour chaque utilisateur, son mot de passe a la même valeur que son nom d'utilisateur.

2 Authentification

2.1 *User*

Dans un fichier `user.dart` (dans un nouveau dossier `lib/models/`) ajoutez une classe `User`.

```
class User {
  User({
    required this.id,
    required this.username,
    this.role,
  });

  int id;
  String username;
  String? role;

  bool get isAdmin => role == 'admin';

  User.fromJson(Map<String, dynamic> json)
    : this(
        id: json['id'] as int,
        username: json['username'] as String,
        role: json['role'] as String?,
      );
}
```

2.2 *AuthState*

Afin de centraliser la gestion de l'authentification, créez une classe `AuthState` héritant de `ChangeNotifier` dans un fichier `auth_state.dart` (dans un nouveau dossier `lib/states/`).

Ajoutez dans cette classe :

- Une variable `_currentUser` (de type `User?`) et un getter associé
- Une variable `_token` (de type `String?`) et un getter associé
- Un getter `isLoggedIn` (de type `bool`)
- Une directive d'import :

```
import 'package:http/http.dart' as http;
```

- Une méthode login :

```
Future<User?> login(String username, String password) async {
  final loginResponse = await http.post(
    Uri.parse('${Configs.baseUrl}/auth/login'),
    headers: {HttpHeaders.contentTypeHeader: 'application/json'},
    body: json.encode({
      'username': username,
      'password': password,
    }),
  );

  if (loginResponse.statusCode == HttpStatus.ok) {
    _token = json.decode(loginResponse.body)['token'];

    final userResponse = await http.get(
      Uri.parse('${Configs.baseUrl}/users/me'),
      headers: {
        HttpHeaders.authorizationHeader: 'Bearer $_token',
        HttpHeaders.contentTypeHeader: 'application/json',
      },
    );

    if (userResponse.statusCode == HttpStatus.ok) {
      _currentUser = User.fromJson(json.decode(userResponse.body));
      notifyListeners();
      return _currentUser;
    }
  }

  logout();
  return null;
}
```

- Ajoutez une méthode logout qui remet à null les variables _token et _currentUser puis envoie une notification de changement.

2.3 Page de connexion

Dans le dossier lib/ui/ créez un fichier login_page.dart contenant un composant stateful LoginPage. Utilisez la classe de state suivante :

```

class _LoginPageState extends State<LoginPage> {
  String username = '';
  String password = '';
  String? error;
  final _formKey = GlobalKey<FormState>();

  String? _validateRequired(String? value) {
    return value == null || value.isEmpty ? 'Ce champ est obligatoire.' : null;
  }

  void _submit() async {
    if (!_formKey.currentState!.validate()) {
      return;
    }
    final user = await context.read<AuthState>().login(username, password);
    if (user != null) {
      if (context.mounted) {
        Navigator.pushNamedAndRemoveUntil(context, '/polls', (_) => false);
      }
    } else {
      setState(() { error = 'Une erreur est survenue.'; });
    }
  }

  @override
  Widget build(BuildContext context) {
    final theme = Theme.of(context);
    return Form(
      key: _formKey,
      child: Column(
        children: [
          TextFormField(
            decoration: const InputDecoration(labelText: 'Identifiant'),
            onChanged: (value) => username = value,
            validator: _validateRequired,
          ),
          const SizedBox(height: 16),
          TextFormField(
            decoration: const InputDecoration(labelText: 'Mot de passe'),
            obscureText: true,
            onChanged: (value) => password = value,
            validator: _validateRequired,
          ),
          const SizedBox(height: 16),
          if (error != null)
            Text(error!, style: theme.textTheme.labelMedium!.copyWith(color: theme.colorScheme.error)),
          ElevatedButton(
            onPressed: _submit,
            child: const Text('Connexion'),
          ),
        ],
      ),
    );
  }
}

```

2.4 Communication entre composants

Dans le fichier `main.dart`, modifiez l'appel à la fonction `runApp()` pour englober le composant `App` avec un `ChangeNotifierProvider<AuthState>`

Ajoutez ensuite la nouvelle `LoginPage` à la place du `Placeholder` dans la définition du routeur.

Modifiez ensuite le composant `AppScaffold` pour appeler la méthode `logout` de `AuthState` sur le clic du bouton de déconnexion.

Vous pouvez également modifier le `DrawerHeader` pour afficher le nom de l'utilisateur connecté.

2.5 Inscription

En utilisant une logique similaire, vous pouvez ajouter la page d'inscription qui fait appel à une méthode `signup` de `AuthState`.

3 Fonctionnement

3.1 Navigation

Pour la navigation dans l'application, il est possible d'utiliser le package `go_router` comme dans le TD précédent ou d'utiliser le composant `Navigator` inclus de base dans le SDK.

Ce composant est initialisé par `MaterialApp` et peut être utilisé via `Navigator.of(context)` ou directement via les méthodes statiques de la classe `Navigator`.

Les routes sont définies dans l'instanciation du composant `MaterialApp`.

```
// Navigue vers une page en l'ajoutant à l'historique.  
Navigator.pushNamed(context, '/polls/create');  
  
// Navigue vers une page avec des paramètres (en l'ajoutant à l'historique).  
Navigator.pushNamed(context, '/polls/detail', arguments: poll);  
  
// Navigue vers une page après avoir vidé l'historique.  
Navigator.pushNamedAndRemoveUntil(context, '/polls', (_) => false);  
  
// Supprime le dernier élément de l'historique (reviens à la page précédente).  
Navigator.pop(context);
```

Pour récupérer un argument donné à une page, vous pouvez utiliser la `ModalRoute` du contexte (générée automatiquement par `Navigator`) :

```
ModalRoute.of(context)?.settings.arguments as Poll?
```

3.2 Operation Result Pattern

Ce pattern se base sur la classe `Result` (fournie avec le sujet dans le fichier `result.dart`) qui permet à la fin d'une opération de retourner un objet indiquant le succès ou l'échec de cette dernière.

Par exemple, il est possible de modifier la méthode `AuthState.login` pour qu'elle retourne un objet de type `Future<Result<User, String>>`, indiquant un retour asynchrone (`Future`) contenant un résultat d'opération (`Result`) avec en cas de succès un utilisateur (`User`) ou une erreur (`String`).

Note : Dans le cas où l'on ne souhaite pas retourner de valeur lors du succès, on peut utiliser le type de retour `Result<Unit, String>`, et retournant le succès : `return Result.success(unit);` (`unit` est une constante déclarée dans `result.dart`).

Vous pouvez modifier le code de la méthode `AuthState.login` pour ajouter ce pattern :

```
// ...
if (loginResponse.statusCode == HttpStatus.ok) {
    // ...
    if (userResponse.statusCode == HttpStatus.ok) {
        _currentUser = User.fromJson(json.decode(userResponse.body));
        notifyListeners();
        return Result.success(_currentUser!);
    }

    error = 'Une erreur est survenue';
} else {
    error = loginResponse.statusCode == HttpStatus.badRequest ||
        loginResponse.statusCode == HttpStatus.unauthorized
        ? 'Identifiant ou mot de passe incorrect'
        : 'Une erreur est survenue';
}

logout();
return Result.failure(error);
```

On peut ensuite récupérer la valeur ou le détail de l'erreur lors de l'appel à la méthode `login`.

4 Événements

Pour la mise en place des différentes interfaces liées aux sondages d'événements et aux votes de participation, vous pouvez créer une nouvelle classe d'état `PollsState` prenant un paramètre un token d'authentification fourni par le `AuthState` du contexte dans la fonction `main` de l'application, via un `ChangeNotifierProxyProvider()` :

```
void main() {
    runApp(
        MultiProvider(
            providers: [
                ChangeNotifierProvider(
                    create: (_) => AuthState(),
                ),
                ChangeNotifierProxyProvider<AuthState, PollsState>(
                    create: (_) => PollsState(),
                    update: (_, authState, pollsState) => pollsState!..setAuthToken(authState.token),
                ),
            ],
            child: const App(),
        ),
    );
}
```

Cette classe d'état pourra contenir une liste des derniers événements récupérés ainsi que les différentes méthodes nécessaires pour obtenir ou modifier ces événements via des appels au serveur.

Pour afficher et modifier un événement, plusieurs solutions sont possibles :

- Passer l'objet `Poll` en paramètre à la page.
- Ajouter un `currentPoll` au `PollsState`, initialisé avec l'événement en cours de lecture/écriture.
- Créer un nouveau State (`PollState`) qui englobe un objet d'événement pour réaliser des actions avec, qui sera donner à chaque page en ayant besoin via un `Provider`.

5 Aides

5.1 Formater des dates

Pour mettre en forme un objet `DateTime` en `String`, il faut utiliser une instance d'un objet `DateFormat` :

```
final dateFormatter = DateFormat('dd/MM/yyyy', 'fr');
String dateString = dateFormatter.format(DateTime(2023, 04, 01));
```

Le premier paramètre est le format de la date souhaité, le second est la culture utilisée pour la localisation (qui doit avoir été initialisée dans le `MaterialApp`).

5.2 Affichage adaptatif

Afin de pouvoir s'adapter aux différents formats d'affichage, il est possible d'utiliser les propriétés fournis par `MediaQuery.of(context)` :

```
final mq = MediaQuery.of(context);
return Padding(
  padding: EdgeInsets.symmetric(
    horizontal: mq.size.width * 0.1,
    vertical: mq.size.height * 0.1,
  ),
  // ...
```

5.3 Affichage des erreurs

Pour afficher des erreurs à différents endroits dans l'application, on peut soit utiliser un affichage direct dans le composant, comme pour `LoginPage` où un composant `Text` apparaît de manière conditionnelle.

Sinon, on peut utiliser un composant `SnackBar`, qui affiche un bandeau en bas de l'application et peut être déclenché via un `ScaffoldMessenger` :

```
ScaffoldMessenger.of(context).showSnackBar(
  SnackBar(
    content: Text(
      "Message d'erreur",
      style: theme.textTheme.labelLarge!.copyWith(
        color: theme.colorScheme.onErrorContainer,
      ),
    ),
    showCloseIcon: true,
    closeIconColor: theme.colorScheme.onErrorContainer,
    backgroundColor: theme.colorScheme.errorContainer,
    duration: const Duration(days: 1),
  ),
);
```