

TP3 - Application multifenêtres et gestion des évènements

el mehdi aboulouafa

25 novembre 2025

1. Introduction

Ce TP porte sur la création d'une application Flutter de gestion de tâches (Todo List). Nous allons apprendre à structurer un projet, créer des interfaces utilisateur et gérer les événements.

2. Étape 1 : Création du projet

```
PS C:\Users\aboum\Desktop\ESI\Nouveau dossier\tpflutter> flutter create todolist_app

A new version of Flutter is available!
To update to the latest version, run "flutter upgrade".

Creating project todolist_app...
Resolving dependencies in 'todolist_app'... (4.4s)
Downloading packages... (1.0s)
Got dependencies in 'todolist_app'.
Wrote 130 files.

All done!
You can find general documentation for Flutter at: https://docs.flutter.dev/
Detailed API documentation is available at: https://api.flutter.dev/
If you prefer video documentation, consider: https://www.youtube.com/c/flutterdev

In order to run your application, type:
```

FIGURE 1 – a commande flutter create

Questions et Réponses

Quelle commande crée un projet Flutter ?

Réponse : `flutter create todolist_app`

Cette commande génère la structure de base.

Pourquoi initialiser Git ?

Réponse : Pour versionner le code et suivre les modifications.

Git permet de sauvegarder l'avancement.

```
PS C:\Users\aboum\Desktop\ESI\Nouveau dossier\tpflutter\todolist_app> git init
Initialized empty Git repository in C:/Users/aboum/Desktop/ESI/Nouveau dossier/tpflutter/todolist_app/.git/
PS C:\Users\aboum\Desktop\ESI\Nouveau dossier\tpflutter\todolist_app> git add .
warning: in the working copy of '.metadata', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'linux/flutter/generated_plugin_registrant.cc', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'linux/flutter/generated_plugin_registrant.h', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'linux/flutter/generated_plugins.cmake', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'macos/Flutter/GeneratedPluginRegistrant.swift', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'pubspec.lock', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'windows/flutter/generated_plugin_registrant.cc', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'windows/flutter/generated_plugin_registrant.h', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'windows/flutter/generated_plugins.cmake', LF will be replaced by CRLF the next time Git touches it
PS C:\Users\aboum\Desktop\ESI\Nouveau dossier\tpflutter\todolist_app> git commit -m "save adv mehdi"
git: 'comitt' is not a git command. See 'git --help'.
The most similar command is
  commit
PS C:\Users\aboum\Desktop\ESI\Nouveau dossier\tpflutter\todolist_app>
```

FIGURE 2 – git commands

3. Étape 2 : Fonction main() et MaterialApp



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

void main() {
  runApp(
    const MaterialApp(
      home: Tasks(),
    ),
  );
}
```

FIGURE 3 – Code de la fonction main() dans le fichier main.dart

```
1 void main() {
2   runApp(const MaterialApp(home: Tasks()));
3 }
```

Questions et Réponses

Quel est le rôle de main() ?

Réponse : Point d'entrée de l'application Dart.

Première fonction exécutée.

Que fait runApp() ?

Réponse : Initialise l'application Flutter.

Attache le widget racine à l'écran.

4. Étape 3 : Création du widget Tasks



```
main.dart M tasks.dart U X tasks_list.dart U task_item.dart U
todolist_app > lib > tasks.dart
4   class _TasksState extends State<Tasks> {
5     final List<Task> _registeredTasks = [
18       Task(
19         description: '',
20         date: DateTime.now().subtract(const Duration(days: 2)),
21         category: Category.personal,
22       ),
23     ],
24   ];
25
26   void _openAddTaskOverlay() {
27     showModalBottomSheet(
28       context: context,
29       builder: (ctx) => const Text('Exemple de fenêtre'),
30     );
31   }
32
33   @override
34   Widget build(BuildContext context) {
35     return Scaffold(
36       appBar: AppBar(
37         title: const Text('Flutter ToDoList'),
38         actions: [
39           IconButton(
40             onPressed: _openAddTaskOverlay,
41             icon: const Icon(Icons.add),
42           ),
43         ],
44       ),
45       body: const Column(
46         children: [
47           Text('The title'),
48           TasksList(tasks: _registeredTasks),
49         ],
50       ),
51     );
52   }
53 }
```

FIGURE 4 – Architecture d'un StatefulWidget avec sa classe d'état

Questions et Réponses

Pourquoi StatefulWidget ?

Réponse : Pour gérer les changements d'état.

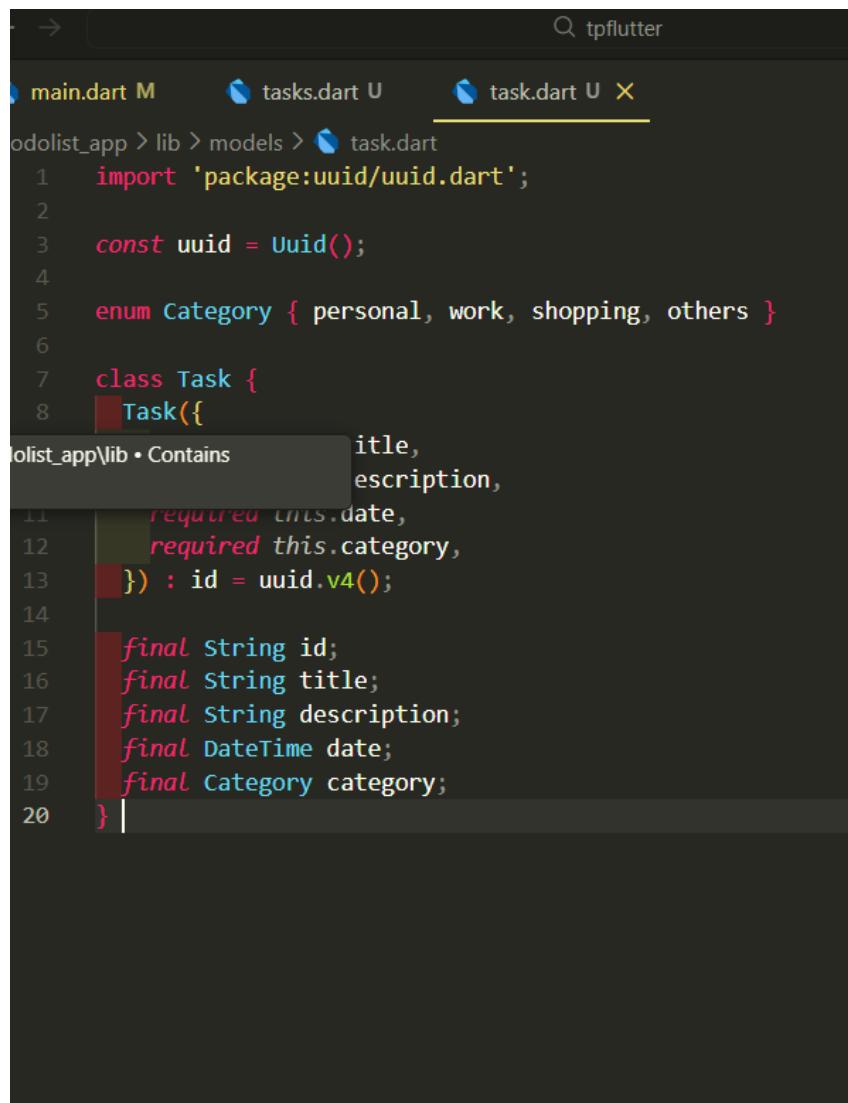
La liste des tâches évolue.

Que fait createState() ?

Réponse : Crée l'état associé au widget.

Retourne une instance de la classe d'état.

5. Étape 4 : Création du modèle Task



The screenshot shows a code editor with a dark theme. The search bar at the top right contains the text "tpflutter". Below the search bar, there are three tabs: "main.dart M", "tasks.dart U", and "task.dart U X". The "task.dart U X" tab is selected. The code editor displays the following Dart code:

```
odolist_app > lib > models > task.dart
1 import 'package:uuid/uuid.dart';
2
3 const Uuid = Uuid();
4
5 enum Category { personal, work, shopping, others }
6
7 class Task {
8     Task({
9         required String title,
10        required String description,
11        required DateTime date,
12        required Category category,
13     }) : id = Uuid.v4();
14
15     final String id;
16     final String title;
17     final String description;
18     final DateTime date;
19     final Category category;
20 }
```

FIGURE 5 – Structure de la classe Task avec ses propriétés finales

```
1 class Task {
2     final String id;
3     final String title;
4     final String description;
5     final DateTime date;
6     final Category category;
7 }
```

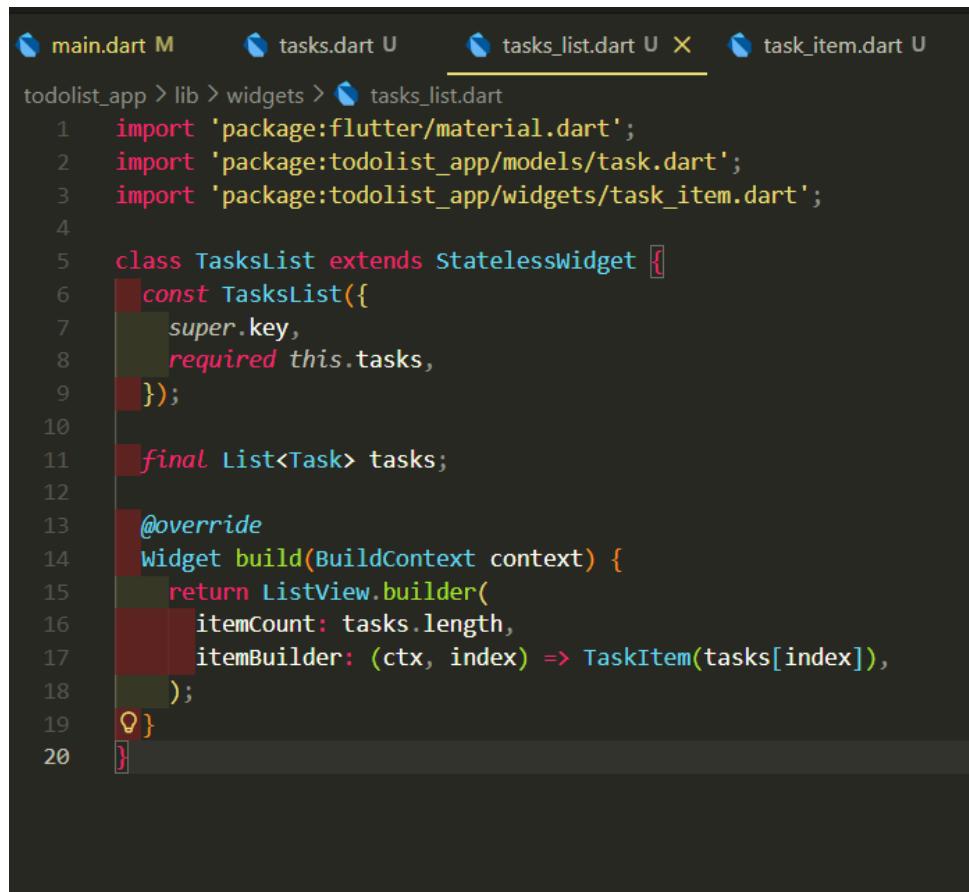
Questions et Réponses

Que fait flutter pub add uuid ?

Réponse : Ajoute le package uuid au projet.

Pour générer des identifiants uniques.

6. Étape 5 : Création de TasksList



```
main.dart M tasks.dart U tasks_list.dart U X task_item.dart U
todolist_app > lib > widgets > tasks_list.dart
1 import 'package:flutter/material.dart';
2 import 'package:todoist_app/models/task.dart';
3 import 'package:todoist_app/widgets/task_item.dart';
4
5 class TasksList extends StatelessWidget {
6   const TasksList({
7     super.key,
8     required this.tasks,
9   });
10
11   final List<Task> tasks;
12
13   @override
14   Widget build(BuildContext context) {
15     return ListView.builder(
16       itemCount: tasks.length,
17       itemBuilder: (ctx, index) => TaskItem(tasks[index]),
18     );
19   }
20 }
```

FIGURE 6 – Utilisation de ListView.builder pour les listes dynamiques

Questions et Réponses

Pourquoi StatelessWidget ?

Réponse : Car elle affiche des données fixes.

Ses propriétés ne changent pas.

Que signifie => en Dart ?

Réponse : Syntaxe pour fonctions fléchées.

Retourne une expression directement.

7. Étape 6 : Organisation des dossiers

Questions et Réponses

Pourquoi séparer models et widgets ?

Réponse : Pour une architecture claire.

Sépare données et interface.

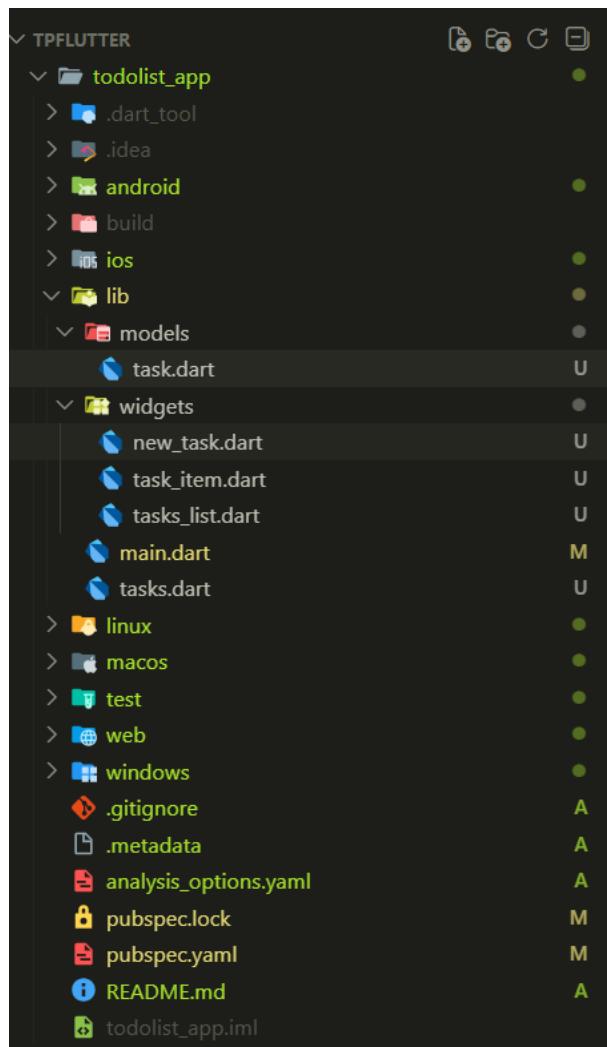


FIGURE 7 – Structure organisée des dossiers models et widgets

8. Étape 7 : Création de TaskItem

```

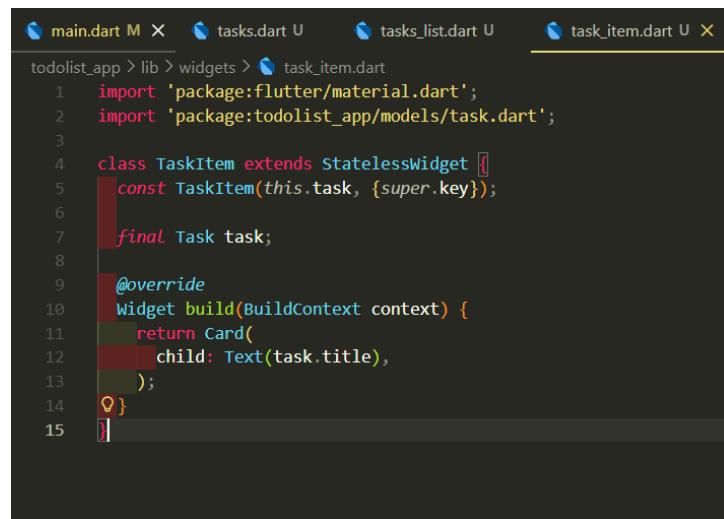
1 class TaskItem extends StatelessWidget {
2   const TaskItem(this.task, {super.key});
3   final Task task;
4
5   @override
6   Widget build(BuildContext context) {
7     return Card(child: Text(task.title));
8   }
9 }
```

Questions et Réponses

À quoi sert le widget Card ?

Réponse : Fournit un conteneur avec elevation.

Effet visuel Material Design.



```

1 import 'package:flutter/material.dart';
2 import 'package:todoist_app/models/task.dart';
3
4 class TaskItem extends StatelessWidget {
5   const TaskItem(this.task, {super.key});
6
7   final Task task;
8
9   @override
10  Widget build(BuildContext context) {
11    return Card(
12      child: Text(task.title),
13    );
14  }
15}

```

FIGURE 8 – Widget Card pour l'affichage des tâches

9. Étape 8 : Ajout du thème et AppBar

Questions et Réponses

Quel est ThemeData ?

Réponse : Configure le thème global de l'app.

Définit couleurs et styles.

Que fait onPressed() ?

Réponse : Gère l'action du bouton pressé.

Exécute du code au clic.

10. Étape 9 : Widget de saisie NewTask

Questions et Réponses

Pourquoi NewTask est StatefulWidget ?

Réponse : Pour gérer les champs de saisie.

Doit mémoriser le texte entré.

11. Étape 10 : Récupération de la saisie

```

1 // Méthode 1 : Callback
2 onChanged: _saveTitleInput
3
4 // Méthode 2 : Controller
5 controller: _titleController

```

Questions et Réponses

Qu'est-ce qu'un TextEditingController ?

Réponse : Gère programmatiquement les TextField.

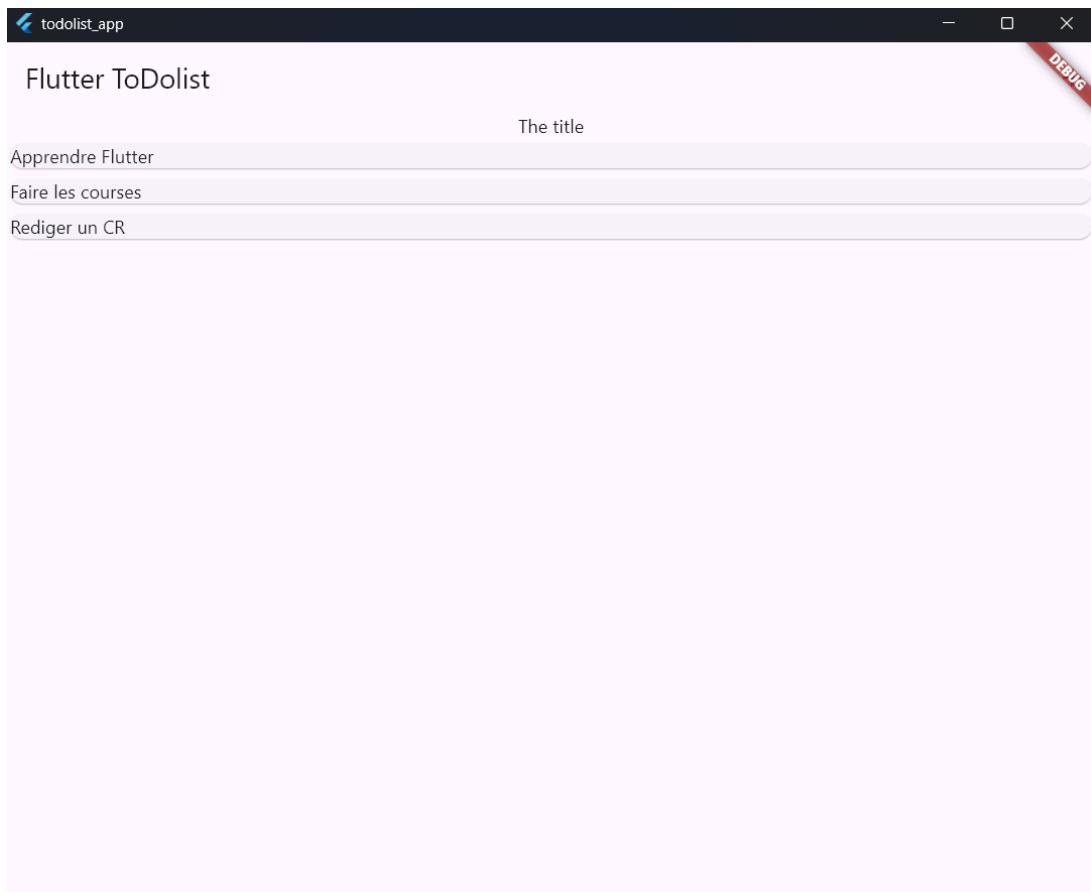


FIGURE 9 – AppBar avec icône d'ajout dans l'application

Permet de lire et écrire le texte.

12. Étape 11 : Contrôle de la saisie

Questions et Réponses

Que fait showDialog() ?

Réponse : Affiche une boîte de dialogue modale.

Pour les messages d'alerte.

Pourquoi trim().isEmpty ?

Réponse : Vérifie si le champ est vide.

Enlève les espaces avant.

13. Étapes 12-14 : Fonctionnalités avancées

Fonctionnalités implémentées

- Ajout de tâches avec tous les champs
- Validation des données saisies
- Affichage des catégories avec couleurs

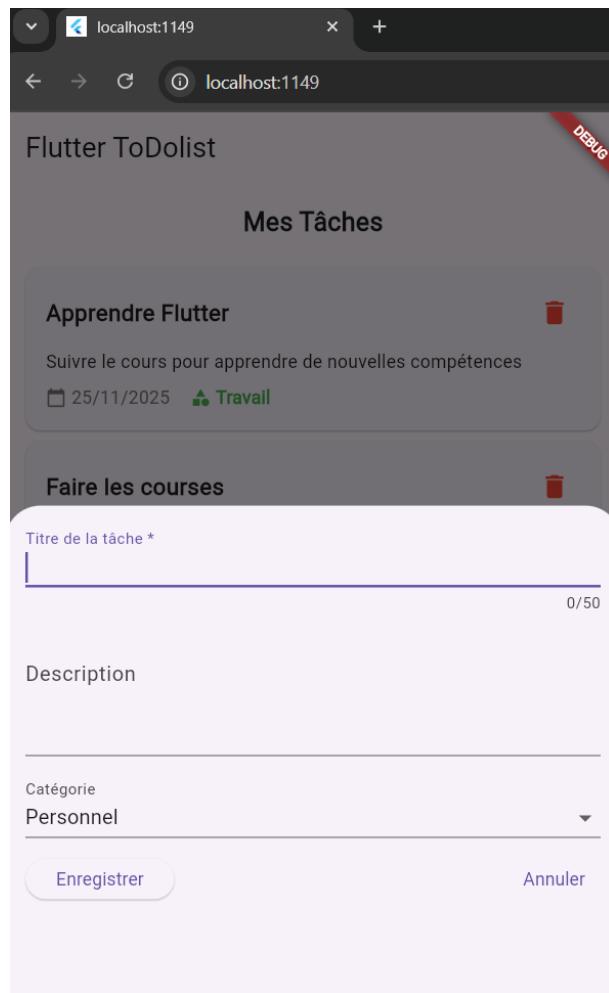


FIGURE 10 – Formulaire de saisie de nouvelle tâche

- Suppression de tâches
- Interface utilisateur complète

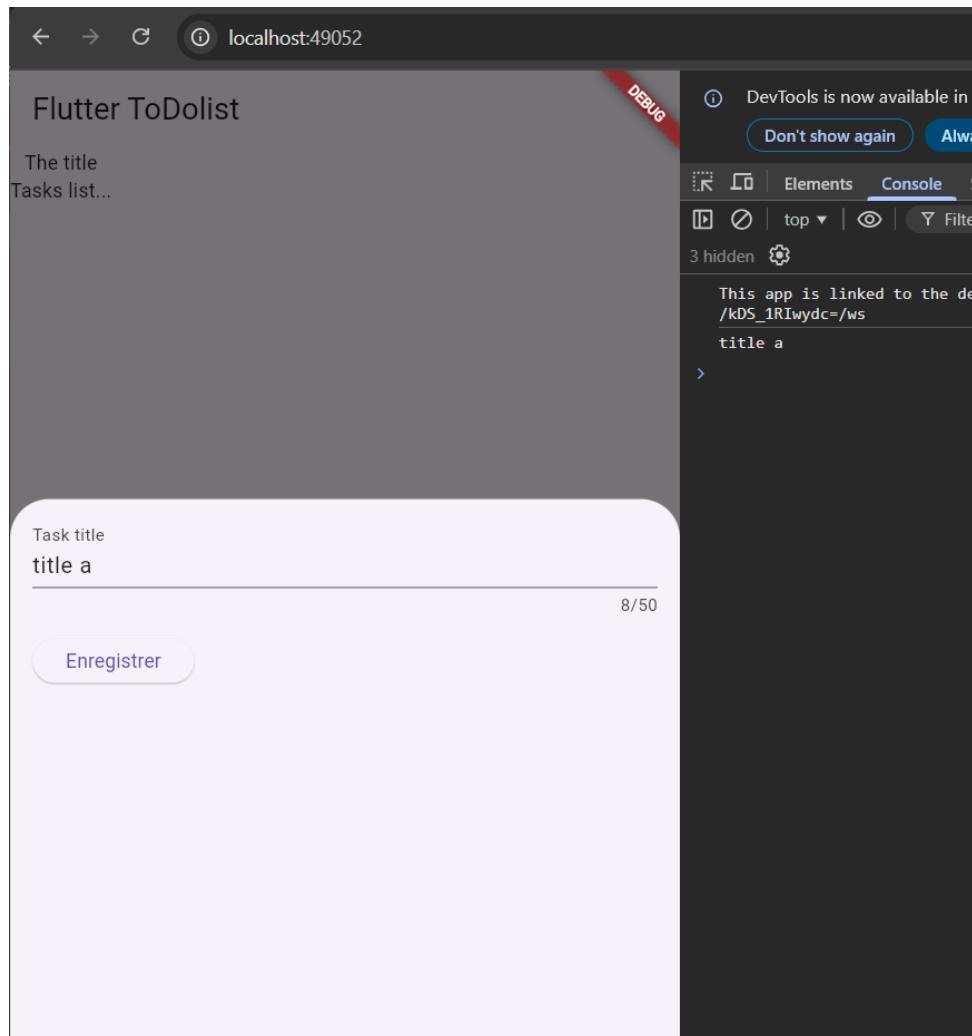
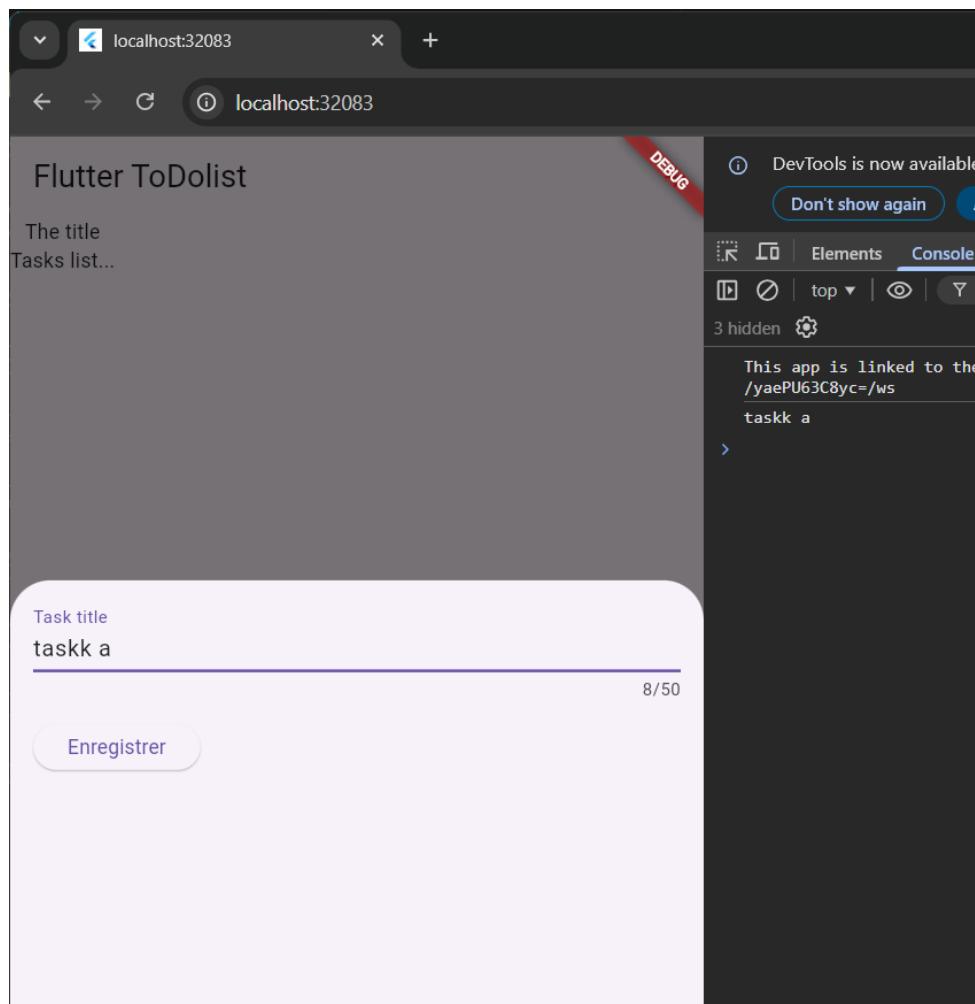


FIGURE 11 – Méthode 1 : Avec une fonction callback (onChanged)

FIGURE 12 – method 2 : $\text{text}_\text{editing}_\text{controller}$

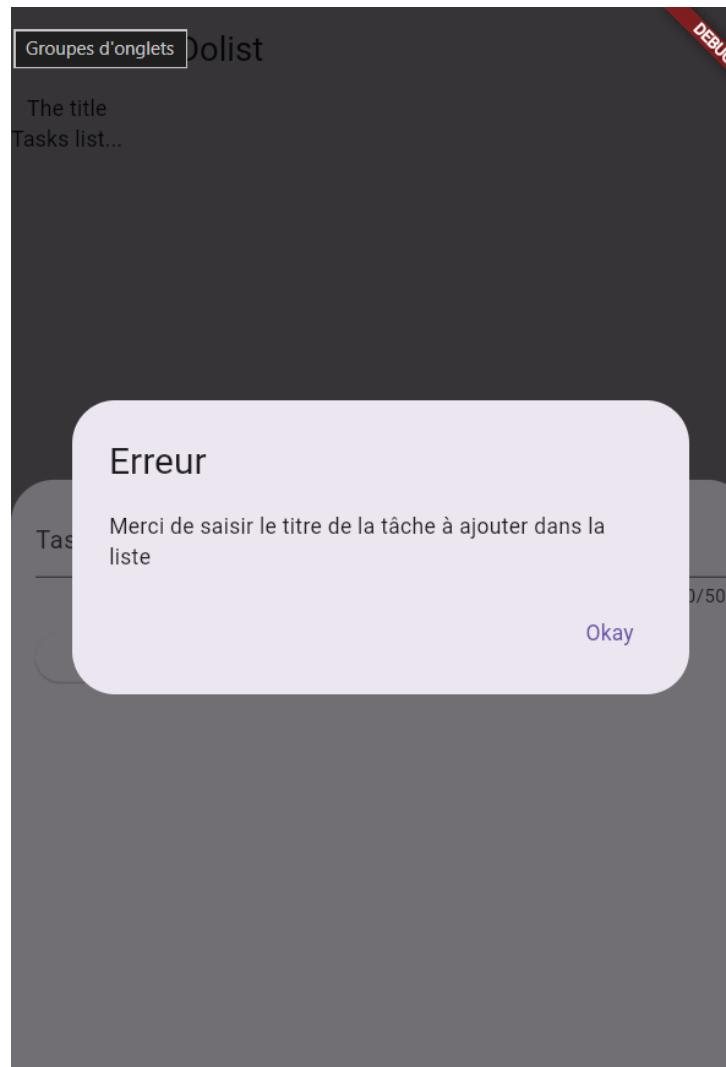


FIGURE 13 – Boîte de dialogue de validation des champs

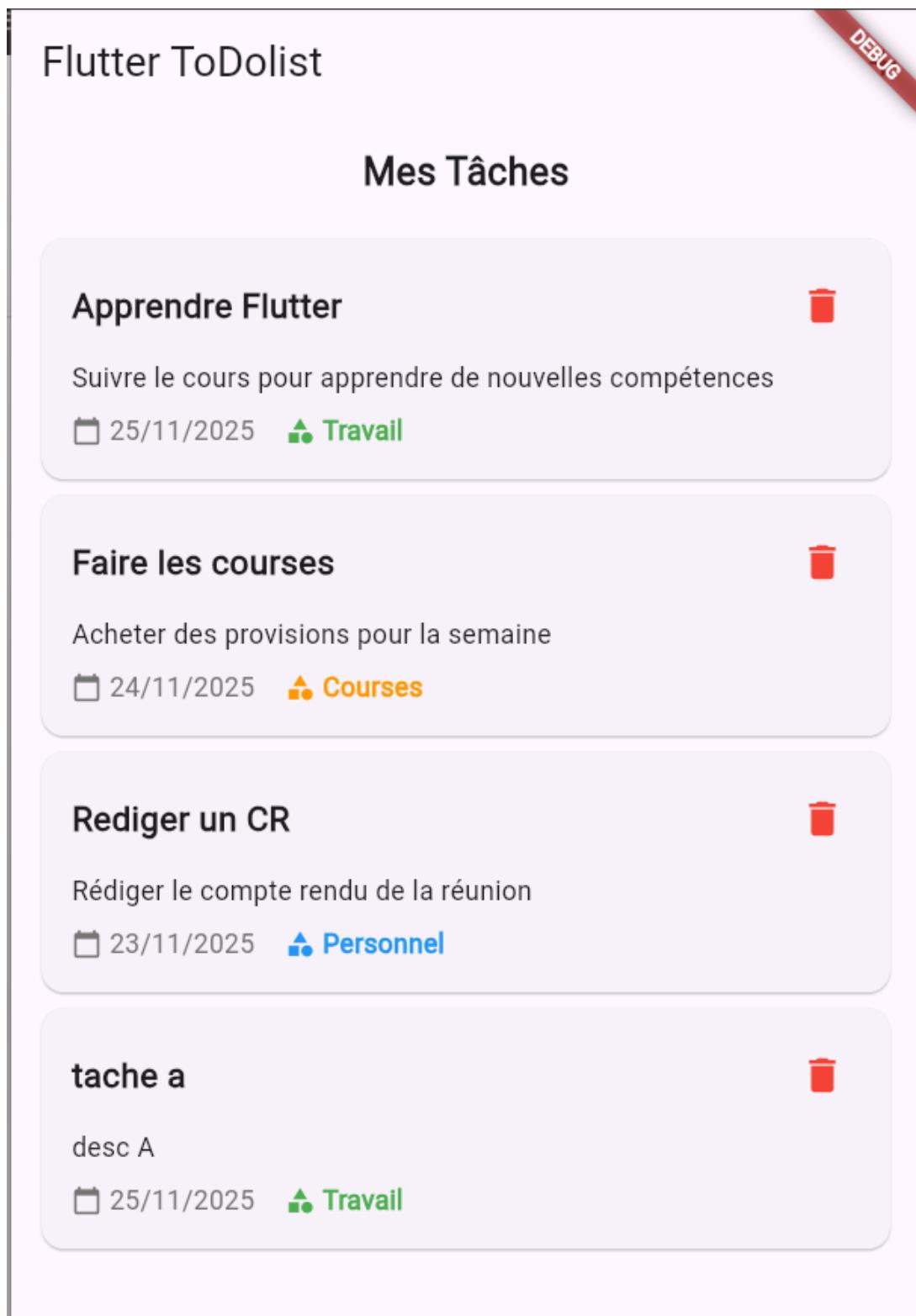


FIGURE 14 – Application TodoList complète et fonctionnelle