Le Dé-tâcheur

Application Mobile de Gestion de Tâches

Cahier des Charges

Projet Final Flutter

Étudiant :	Samuel Kensley GAËTAN
Code :	2450-C
Professeur :	Phawens LOUIS-JEAN
Date :	Décembre 2025

Faculté des sciences informatiques de l'Université Épiscopale d'Haïti

Table des matières

1. Présentation générale du projet

1.1 Contexte

Le Dé-tâcheur est une application mobile de gestion de tâches personnelles développée dans le cadre du projet final du cours de Flutter. Dans un monde où la productivité et l'organisation sont essentielles, cette application vise à aider les utilisateurs à gérer efficacement leurs tâches quotidiennes, qu'elles soient professionnelles, personnelles ou urgentes.

1.2 Objectifs

L'objectif principal de ce projet est de créer une application mobile complète et fonctionnelle permettant :

- La gestion complète des tâches (création, modification, suppression)
- L'organisation des tâches par catégories (Travail, Personnel, Urgent)
- Le suivi de la progression avec des statistiques visuelles
- La recherche et le filtrage avancés des tâches

- Une authentification sécurisée des utilisateurs
- La persistance locale des données avec SQLite

1.3 Public cible

L'application s'adresse à toute personne souhaitant améliorer sa productivité et organiser ses tâches quotidiennes :

- **Étudiants** : pour gérer leurs devoirs, projets et examens
- **Professionnels** : pour organiser leurs tâches professionnelles et deadlines
- **Particuliers** : pour suivre leurs tâches personnelles et quotidiennes

1.4 Valeur ajoutée

> **Le Dé-tâcheur** se distingue par son interface intuitive, sa catégorisation intelligente des tâches, et son système de statistiques visuelles qui permet aux utilisateurs de suivre leur progression et d'améliorer leur productivité. L'application fonctionne entièrement hors ligne grâce à SQLite, tout en offrant une authentification sécurisée via Firebase.

2. Fonctionnalités détaillées

1. ## Authentification des utilisateurs

   Inscription : Formulaire avec nom complet, email et mot de passe. Validation : email valide, mot de passe minimum 6 caractères, confirmation du mot de passe. Gestion des erreurs Firebase.

   Connexion : Authentification par email et mot de passe. Gestion des erreurs (utilisateur non trouvé, mot de passe incorrect). Persistance de la session utilisateur.

   Déconnexion : Bouton de déconnexion accessible depuis l'écran principal. Retour automatique à l'écran de connexion.

2. ## Gestion des tâches (CRUD)

   Créer : Titre, description, catégorie (Travail/Personnel/Urgent), date automatique, statut non complété par défaut.

   Afficher : Liste complète avec badges colorés, indicateur de statut, date de création.

   Modifier : Édition de tous les champs, pré-remplissage du formulaire, conservation de la date de création.

   Supprimer : Confirmation avant suppression, suppression définitive de la base de données.

   Marquer complété : Checkbox circulaire, bascule instantanée, effet visuel (texte barré, grisé).

3. ## Recherche et filtrage

   Recherche : Barre de recherche en temps réel dans le titre et la description.

Filtrage : Menu déroulant par catégorie (Toutes, Travail, Personnel, Urgent).

Tri : Par date (récentes en premier), par catégorie (alphabétique), par statut (non complétées puis complétées).

4 ### Statistiques et suivi

Vue d'ensemble : Total, complétées, en cours, pourcentage de progression.

Par catégorie : Répartition, progression avec barres colorées, compteurs individuels.

Indicateurs visuels : Barre de progression linéaire, indicateur circulaire, messages de motivation adaptatifs, codes couleur (Bleu, Vert, Rouge).

3. Architecture de l'application

3.1 Architecture générale

L'application suit une architecture **MVC (Model-View-Controller)** adaptée à Flutter, avec une séparation claire entre les modèles de données, les services et l'interface utilisateur.

3.2 Structure des dossiers

```
lib/
├── main.dart
├── firebase_options.dart
├── models/
│   └── task.dart
├── services/
│   └── database_service.dart
├── screens/
│   ├── login_screen.dart
│   ├── register_screen.dart
│   ├── task_list_screen.dart
│   ├── add_edit_task_screen.dart
│   └── statistics_screen.dart
└── widgets/
    └── task_card.dart
```

3.3 Couches de l'application

Couche de présentation (UI)

Widgets Flutter (Stateless et Stateful). Gère l'affichage et les interactions. Écrans, widgets réutilisables, gestion de l'état avec setState().

Couche de logique métier (Services)

Services encapsulant la logique applicative. DatabaseService pour CRUD sur SQLite, Firebase Authentication pour gestion des utilisateurs.

Couche de données (Models)

Structures de données et méthodes de conversion. Task avec toMap() et fromMap(), conversion entre Dart et SQLite.

3.4 Flux de données

1. **Utilisateur** interagit avec l'interface (UI)
2. **Écran** appelle le service approprié (DatabaseService)
3. **Service** effectue l'opération sur la base de données
4. **Données** retournent au service sous forme de modèle
5. **UI** se met à jour avec setState() et affiche les données

4. Modèle de données

4.1 Classe Task (Dart)

La classe `Task` représente une tâche dans l'application.

```
class Task {
  final int? id;
  final String title;
  final String description;
  final String category;  // 'Travail', 'Personnel', 'Urgent'
  final bool completed;
  final DateTime createdAt;

  Task({this.id, required this.title, required this.description,
        required this.category, this.completed = false,
        DateTime? createdAt}) : createdAt = createdAt ?? DateTime.now();

  Map<String, dynamic> toMap() {
    return {'id': id, 'title': title, 'description': description,
            'category': category, 'completed': completed ? 1 : 0,
            'createdAt': createdAt.toIso8601String()};
  }

  factory Task.fromMap(Map<String, dynamic> map) {
    return Task(id: map['id'], title: map['title'],
                description: map['description'], category: map['category'],
                completed: map['completed'] == 1,
                createdAt: DateTime.parse(map['createdAt']));
  }
}
```

4.2 Schéma de la base de données SQLite

Table : tasks

Champ	Type	Contraintes
`id`	INTEGER	PRIMARY KEY, AUTOINCREMENT

Champ	Type	Contraintes
`title`	TEXT	NOT NULL
`description`	TEXT	NOT NULL
`category`	TEXT	NOT NULL
`completed`	INTEGER	NOT NULL (0 ou 1)
`createdAt`	TEXT	NOT NULL (ISO 8601)

4.3 Opérations CRUD

CREATE

`createTask(Task task)`

READ

`getAllTasks()`

UPDATE

`updateTask(Task task)`

DELETE

`deleteTask(int id)`

5. Description des écrans

1. Écran de connexion (login_screen.dart)

Composants : Logo, champs email et mot de passe avec icônes, bouton afficher/masquer, bouton connexion, lien inscription. **Validations :** Format email valide, mot de passe ≥ 6 caractères, champs non vides.

2. Écran d'inscription (register_screen.dart)

Composants : Logo, champs nom/email/mot de passe/confirmation, boutons afficher/masquer, bouton inscription, lien connexion. **Validations :** Tous champs requis, email valide, mot de passe ≥ 6 caractères, mots de passe identiques.

3. Liste des tâches (task_list_screen.dart)

Composants : AppBar (titre, statistiques, déconnexion), barre de recherche, filtres catégorie et tri, liste scrollable de TaskCard, compteurs, FloatingActionButton. **Interactions :** Pull-to-refresh, checkbox pour marquer complété, boutons éditer/supprimer.

4. Ajout/Modification (add_edit_task_screen.dart)

Composants : AppBar dynamique, champ titre, champ description multiligne, menu catégorie avec couleurs, date de création (édition), carte de conseils, boutons Annuler/Enregistrer. **Validations :** Titre et description non vides, catégorie sélectionnée.

5. Statistiques (statistics_screen.dart)

Composants : Carte progression avec gradient, pourcentage dans cercle blanc, barre de progression linéaire, 3 cartes résumé (Total/Terminées/En cours), section répartition par catégorie avec barres colorées, indicateur circulaire, messages de motivation adaptatifs.

6. Technologies utilisées

6.1 Framework et langage

Flutter
Framework UI - Version 3.0+

Dart
Langage - Version 3.10.4+

6.2 Packages et dépendances

```
dependencies:
  firebase_core: ^4.3.0        # Configuration Firebase
  firebase_auth: ^6.1.3        # Authentification
  sqflite: ^2.4.2              # SQLite pour Flutter
  path: ^1.9.1                 # Gestion des chemins
  intl: ^0.20.2                # Internationalisation/dates
  cupertino_icons: ^1.0.8      # Icônes iOS

dev_dependencies:
  flutter_launcher_icons: ^0.13.1
  flutter_lints: ^6.0.0        # Analyse de code
```

6.3 Services externes

Firebase Authentication
Service d'authentification de Google. Gestion sécurisée avec email/mot de passe, sessions, validation des emails, gestion des erreurs.

SQLite
Base de données locale embarquée pour Android. Stockage sans connexion Internet, opérations CRUD rapides, requêtes SQL, données persistantes.

6.4 Widgets principaux

MaterialApp	Scaffold	AppBar	ListView

Card	TextFormField	ElevatedButton	AlertDialog
CircularProgress	FloatingActionBtn	StreamBuilder	DropdownButton

7. Planning de réalisation

Le projet est prévu pour être réalisé sur **3 jours**, avec une organisation intensive permettant de compléter toutes les fonctionnalités.

Jour 1 : Configuration et Authentification — J1

1. Configuration initiale (2h)
Créer projet Flutter, configurer Firebase Android, installer dépendances, créer structure dossiers, configurer firebase_options.dart.

2. Modèle et DatabaseService (3h)
Créer classe Task avec toMap()/fromMap(), implémenter DatabaseService complet avec Singleton, créer table SQLite, implémenter CRUD, tester avec print().

3. Authentification (3h)
Développer écrans connexion et inscription avec validations complètes, implémenter Firebase Auth (signIn, signUp, signOut), gérer toutes les erreurs, tester sur émulateur et appareil réel.

Jour 2 : Interface principale et CRUD — J2

1. Liste des tâches (4h)
Créer TaskCard widget, développer TaskListScreen complet, implémenter affichage, marquer complété/non complété, suppression avec confirmation, ajouter FloatingActionButton.

2. Ajout et modification (4h)
Créer AddEditTaskScreen avec formulaire complet, implémenter validations, gérer modes ajout et édition, intégrer createTask() et updateTask(), ajouter messages de confirmation, tester toutes les opérations CRUD.

Jour 3 : Fonctionnalités avancées et finalisation — J3

1. Recherche et filtrage (2h)
Implémenter barre de recherche en temps réel, ajouter filtres par catégorie, implémenter tri (date, catégorie, statut), implémenter searchTasks() et getTasksByCategory(), tester tous les cas.

2. Statistiques (3h)
Créer StatisticsScreen complet, calculer tous les compteurs, afficher pourcentage de progression, créer barres de progression par catégorie, implémenter indicateur circulaire, ajouter messages de motivation.

3. Tests et finalisation (3h)

Tester tous les flux utilisateur de bout en bout, vérifier validations, tester sur appareil Android réel, corriger bugs et problèmes d'overflow, améliorer l'UI, commenter le code, préparer documentation, commits réguliers sur GitHub.

Points de contrôle

- Fin J1 : Authentification et base de données fonctionnelles
- Fin J2 : CRUD complet opérationnel avec interface principale
- Fin J3 : Application complète, testée et prête à soumettre

8. Difficultés anticipées et solutions

Bien que le projet soit bien structuré, certaines difficultés techniques peuvent survenir. Voici les défis principaux anticipés et les stratégies pour les surmonter :

1. Configuration de Firebase

Problème : Configuration Firebase complexe, fichier google-services.json.

Solutions : Suivre documentation FlutterFire, vérifier emplacement (android/app/), utiliser flutterfire configure, consulter logs détaillés.

2. Gestion de l'état avec setState()

Problème : Mise à jour de l'interface après CRUD si setState() mal appelé.

Solutions : Toujours appeler setState() après modification, recharger liste après CRUD, utiliser async/await correctement, vérifier widget monté.

3. Requêtes SQLite et conversions

Problème : Erreurs de conversion entre objets Dart et Maps SQLite.

Solutions : Tester toMap() et fromMap() séparément, utiliser toIso8601String() pour dates, stocker booléens comme INTEGER (0/1), utiliser try-catch.

4. Validation des formulaires

Problème : Validations côté client mal implémentées.

Solutions : Utiliser GlobalKey pour validation, validators personnalisés, messages d'erreur clairs, utiliser .trim() pour enlever espaces.

5. Erreurs d'overflow (RenderFlex)

Problème : "RenderFlex overflowed" quand contenu dépasse l'espace disponible.

Solutions : Utiliser Expanded/Flexible dans Row/Column, envelopper textes longs dans Flexible, ajouter overflow: TextOverflow.ellipsis, utiliser SingleChildScrollView, définir maxLines, tester sur différentes tailles d'écran.

6. Navigation entre les écrans

Problème : Gestion de la navigation créant bugs si données non rafraîchies.

Solutions : Recharger tâches dans initState(), utiliser await avec Navigator.push(), appeler _loadTasks() après navigation, utiliser pushReplacement() pour connexion/déconnexion.

7. Tests sur appareil Android

Problème : Application fonctionne sur émulateur mais problèmes sur appareil réel.

Solutions : Tester régulièrement sur appareil réel, vérifier permissions Android (Internet), utiliser mode debug, tester sur plusieurs versions d'Android, optimiser ressources.

Conseils généraux

✓ **Déboguer méthodiquement** : Utiliser print() pour suivre le flux
✓ **Consulter la documentation** : Flutter et Firebase ont d'excellentes docs
✓ **Commencer simple** : Implémenter les bases avant les features complexes
✓ **Tester fréquemment** : Ne pas attendre d'avoir tout codé
✓ **Commits réguliers** : Sauvegarder le progrès sur GitHub

Conclusion

Le Dé-tâcheur est un projet complet qui permet de mettre en pratique l'ensemble des compétences acquises dans le cours de Flutter. De l'authentification Firebase à la gestion de bases de données SQLite, en passant par la création d'interfaces utilisateur intuitives et responsive, ce projet couvre tous les aspects fondamentaux du développement d'applications mobiles modernes pour Android.

Compétences développées

✓ Maîtrise de Dart et Flutter
✓ Gestion de l'état avec setState()
✓ Opérations CRUD avec SQLite
✓ Authentification Firebase

> ✓ Navigation entre écrans
> ✓ Validation de formulaires
> ✓ Architecture MVC
> ✓ Design UI/UX moderne

Ce projet représente une opportunité excellente pour démontrer la capacité à concevoir, développer et déployer une application mobile fonctionnelle et professionnelle. En suivant ce cahier des charges et le planning proposé, le développement devrait se dérouler de manière structurée et efficace.