Le Dé-tâcheur

Application Mobile de Gestion de Tâches

Cahier des Charges

Projet Final Flutter

Étudiant :	_____
Code :	_____
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 3

2. Fonctionnalités détaillées 4

3. Architecture de l'application 5

4. Modèle de données 6

5. Description des écrans 7

6. Technologies utilisées 8

7. Planning de réalisation 9

8. Difficultés anticipées 10

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 (email déjà utilisé, mot de passe faible)

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 une tâche :

- Titre (obligatoire)
- Description détaillée (obligatoire)
- Catégorie : Travail, Personnel ou Urgent (obligatoire)
- Date de création automatique
- Statut : non complété par défaut

Afficher les tâches :

- Liste complète avec badges de catégorie colorés
- Indicateur visuel de statut (complété/non complété)
- Affichage de la date de création

Modifier une tâche :

- Édition de tous les champs (titre, description, catégorie)
- Pré-remplissage du formulaire avec les données actuelles
- Conservation de la date de création

Supprimer une tâche :

- Boîte de dialogue de confirmation avant suppression
- Suppression définitive de la base de données
- Message de confirmation après suppression

Marquer comme complété/non complété :

- Bouton de checkbox circulaire intuitif
- Bascule instantanée du statut
- Effet visuel (texte barré, couleur grisée) pour les tâches complétées

3 Recherche et filtrage

Recherche par titre :

- Barre de recherche en temps réel
- Recherche dans le titre et la description
- Résultats instantanés pendant la saisie

Filtrage par catégorie :

- Menu déroulant avec options : Toutes, Travail, Personnel, Urgent
- Affichage uniquement des tâches de la catégorie sélectionnée

Tri des tâches :

- Par date (les plus récentes en premier)
- Par cat��gorie (ordre alphabétique)
- Par statut (non complétées puis complétées)

4 Statistiques et suivi

Vue d'ensemble :

- Nombre total de tâches
- Nombre de tâches complétées
- Nombre de tâches en cours
- Pourcentage de progression global

Statistiques par catégorie :

- Répartition des tâches par catégorie
- Progression par catégorie avec barres de progression
- Compteurs individuels pour chaque catégorie

Indicateurs visuels :

- Barre de progression linéaire globale
- Indicateur circulaire de progression
- Messages de motivation adaptatifs selon le pourcentage
- Codes couleur par catégorie (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                    # Point d'entrée de l'application
├── firebase_options.dart        # Configuration Firebase
├── models/
│   └── task.dart               # Modèle de données Task
├── services/
│   └── database_service.dart   # Service de gestion SQLite
├── screens/
│   ├── login_screen.dart       # Écran de connexion
│   ├── register_screen.dart    # Écran d'inscription
│   ├── task_list_screen.dart   # Liste des tâches
│   ├── add_edit_task_screen.dart # Ajout/Modification
│   └── statistics_screen.dart  # Statistiques
└── widgets/
    └── task_card.dart          # Widget de carte de tâche
```

3.3 Couches de l'application

Couche de présentation (UI)

Composée de widgets Flutter (Stateless et Stateful). Gère l'affichage et les interactions utilisateur.

- Écrans (Screens)
- Widgets réutilisables
- Gestion de l'état local avec setState()

Couche de logique métier (Services)

Services qui encapsulent la logique applicative et les opérations sur les données.

- DatabaseService : opérations CRUD sur SQLite
- Firebase Authentication : gestion des utilisateurs

Couche de données (Models)

Définition des structures de données et méthodes de conversion.

- Task : modèle de tâche avec méthodes toMap() et fromMap()
- Conversion entre objets 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;              // ID auto-incrémenté (nullable)
  final String title;         // Titre de la tâche
  final String description;   // Description détaillée
  final String category;      // 'Travail', 'Personnel', 'Urgent'
  final bool completed;       // Statut de complétion
  final DateTime createdAt;   // Date de création

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

  // Convertir en Map pour SQLite
  Map<String, dynamic> toMap() {
    return {
      'id': id,
      'title': title,
      'description': description,
      'category': category,
      'completed': completed ? 1 : 0,
      'createdAt': createdAt.toIso8601String(),
```

```
    };
  }

  // Créer depuis Map
  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	Description
`id`	INTEGER	PRIMARY KEY, AUTOINCREMENT	Identifiant unique
`title`	TEXT	NOT NULL	Titre de la tâche
`description`	TEXT	NOT NULL	Description détaillée
`category`	TEXT	NOT NULL	Catégorie (Travail/Personnel/Urgent)
`completed`	INTEGER	NOT NULL	Statut (0 = non, 1 = oui)
`createdAt`	TEXT	NOT NULL	Date de création (ISO 8601)

4.3 Opérations CRUD

CREATE

`createTask(Task task)`

READ

`getAllTasks()`

Insère une nouvelle tâche dans la base

Récupère toutes les tâches

UPDATE

`updateTask(Task task)`

Met à jour une tâche existante

DELETE

`deleteTask(int id)`

Supprime une tâche par son ID

4.4 Méthodes additionnelles

- `getTasksByCategory(String category)`
  Filtre les tâches par catégorie
- `searchTasks(String query)`
  Recherche dans le titre et la description
- `getTask(int id)`
  Récupère une tâche spécifique par son ID

5. Description des écrans

Écran de connexion

Fichier : `login_screen.dart`

Description : Premier écran de l'application permettant aux utilisateurs de se connecter.

Composants :

- Logo de l'application "Le Dé-tâcheur"
- Champ de saisie email avec icône
- Champ de saisie mot de passe avec bouton afficher/masquer
- Bouton "Se connecter"
- Lien vers l'écran d'inscription
- Messages d'erreur contextuel (email invalide, mot de passe incorrect)

Validations :

- Format email valide (regex)
- Mot de passe minimum 6 caractères

- Champs non vides

Écran d'inscription

Fichier : `register_screen.dart`

Description : Permet aux nouveaux utilisateurs de créer un compte.

Composants :

- Logo de l'application
- Champ nom complet
- Champ email
- Champ mot de passe
- Champ confirmation mot de passe
- Bouton "S'inscrire"
- Lien vers l'écran de connexion

Validations :

- Tous les champs requis
- Email au format valide
- Mot de passe minimum 6 caractères
- Mots de passe identiques
- Gestion des erreurs Firebase (email déjà utilisé, mot de passe faible)

Liste des tâches

Fichier : `task_list_screen.dart`

Description : Écran principal affichant toutes les tâches de l'utilisateur.

Composants :

- AppBar avec titre, bouton statistiques et déconnexion
- Barre de recherche en temps réel
- Menu déroulant de filtrage par catégorie
- Menu déroulant de tri (date, catégorie, statut)
- Liste scrollable de cartes de tâches (TaskCard widget)

- Compteur de tâches affichées et complétées
- FloatingActionButton pour ajouter une tâche
- Message d'état vide si aucune tâche

Interactions :

- Pull-to-refresh pour actualiser
- Tap sur checkbox pour marquer complété
- Tap sur bouton éditer pour modifier
- Tap sur bouton supprimer avec confirmation

Ajout/Modification de tâche

Fichier : `add_edit_task_screen.dart`

Description : Formulaire pour créer une nouvelle tâche ou modifier une existante.

Composants :

- AppBar avec titre dynamique (Nouvelle/Modifier)
- Champ titre avec validation
- Champ description multiligne
- Menu déroulant catégorie avec indicateurs de couleur
- Affichage de la date de création (en mode édition)
- Carte de conseils avec tips
- Boutons Annuler et Enregistrer

Validations :

- Titre non vide
- Description non vide
- Catégorie sélectionnée

Statistiques

Fichier : `statistics_screen.dart`

Description : Affiche des statistiques détaillées sur les tâches de l'utilisateur.

Composants :

- Carte de progression globale avec gradient
- Pourcentage de complétion dans un cercle blanc
- Barre de progression linéaire
- 3 cartes résumé : Total, Terminées, En cours
- Section "Répartition par catégorie" avec barres de progression colorées
- Indicateur circulaire de progression (CircularProgressIndicator)
- Message de motivation adaptatif selon le pourcentage

Calculs dynamiques :

- Pourcentage global : (complétées / total) x 100
- Tâches par catégorie avec compteurs individuels
- Progression par catégorie avec barres de couleur

6. Technologies utilisées

6.1 Framework et langage

Flutter
Framework UI

Version : 3.0+ (SDK stable)

Dart
Langage

Version : 3.10.4+

6.2 Packages et dépendances

```
dependencies:
  flutter:
    sdk: flutter

  # Firebase - Authentification
  firebase_core: ^4.3.0          # Configuration Firebase
  firebase_auth: ^6.1.3          # Authentification utilisateurs

  # Base de données locale
  sqflite: ^2.4.2                # SQLite pour Flutter
  path: ^1.9.1                   # Gestion des chemins

  # UI et formatage
```

```
  intl: ^0.20.2                 # Internationalisation/dates
  cupertino_icons: ^1.0.8       # Icônes iOS

dev_dependencies:
  flutter_test:
    sdk: flutter
  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 permettant la gestion sécurisée des utilisateurs avec email/mot de passe.

- Inscription et connexion
- Gestion des sessions
- Validation des emails
- Gestion des erreurs d'authentification

SQLite

Base de données locale embarquée pour la persistance des tâches sur l'appareil Android.

- Stockage local sans connexion Internet
- Opérations CRUD rapides
- Requêtes SQL pour filtrage et recherche
- Données persistantes même après fermeture

6.4 Widgets principaux utilisés

MaterialApp Point d'entrée Material	**Scaffold** Structure de base	**AppBar** Barre de navigation
ListView Liste scrollable	**Card** Cartes pour tâches	**TextFormField** Champs de formulaire
ElevatedButton Boutons d'action	**AlertDialog** Boîtes de confirmation	**CircularProgressIndicator** Indicateurs chargement
FloatingActionButton	**StreamBuilder**	**DropdownButton**

Bouton flottant	Gestion flux Firebase	Menus déroulants

7. Planning de réalisation

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

Jour 1 : Configuration et Authentification J1

1 **Configuration initiale (2h)**

- Créer le projet Flutter
- Configurer Firebase pour Android
- Installer toutes les dépendances (firebase_core, firebase_auth, sqflite, path, intl)
- Créer la structure des dossiers complète
- Configurer firebase_options.dart

2 **Modèle de données et DatabaseService (3h)**

- Créer la classe Task avec toMap() et fromMap()
- Implémenter DatabaseService complet avec Singleton
- Créer la table SQLite
- Implémenter toutes les méthodes CRUD
- Tester chaque méthode avec print()

3 **Authentification (3h)**

- Développer l'écran de connexion avec validations complètes
- Développer l'écran d'inscription avec validations
- Implémenter Firebase Auth (signIn, signUp, signOut)
- Gérer toutes les erreurs et messages utilisateur
- Tester sur émulateur et appareil réel

Jour 2 : Interface principale et CRUD J2

1 **Liste des tâches (4h)**

- Créer TaskCard widget réutilisable
- Développer TaskListScreen complet
- Implémenter l'affichage de toutes les tâches
- Ajouter la fonction marquer comme complété/non complété
- Implémenter la suppression avec confirmation

- Ajouter FloatingActionButton

2 Ajout et modification (4h)

- Créer AddEditTaskScreen avec formulaire complet
- Implémenter toutes les validations de champs
- Gérer les modes ajout et édition
- Intégrer les opérations createTask() et updateTask()
- Ajouter les messages de confirmation
- Tester toutes les opérations CRUD

Jour 3 : Fonctionnalités avancées et finalisation J3

1 Recherche et filtrage (2h)

- Implémenter la barre de recherche en temps réel
- Ajouter les filtres par catégorie (Toutes, Travail, Personnel, Urgent)
- Implémenter le tri (date, catégorie, statut)
- Implémenter searchTasks() et getTasksByCategory()
- Tester tous les cas de filtrage et recherche

2 Écran de statistiques (3h)

- Créer StatisticsScreen complet
- Calculer tous les compteurs (total, complétées, en cours)
- Afficher le pourcentage de progression global
- Créer les barres de progression par catégorie
- Implémenter l'indicateur circulaire
- Ajouter les messages de motivation adaptatifs

3 Tests et finalisation (3h)

- Tester tous les flux utilisateur de bout en bout
- Vérifier toutes les validations
- Tester sur appareil Android réel
- Corriger les bugs et problèmes d'overflow
- Améliorer l'UI si nécessaire
- Commenter le code
- Préparer la 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 potentiel :

La configuration initiale de Firebase peut être complexe, notamment pour gérer le fichier `google-services.json` pour Android.

Solutions :

- Suivre la documentation officielle de FlutterFire
- Vérifier que le fichier est au bon emplacement (android/app/)
- Utiliser flutterfire configure pour automatiser
- Consulter les logs détaillés en cas d'erreur

2 Gestion de l'état avec setState()

Problème potentiel :

La mise à jour de l'interface après des opérations CRUD peut ne pas fonctionner si setState() n'est pas appelé correctement.

Solutions :

- Toujours appeler setState() après modification des données
- Recharger la liste après chaque opération CRUD
- Utiliser async/await correctement
- Vérifier que le widget est monté avant setState()

3 Requêtes SQLite et conversions

Problème potentiel :

Les erreurs de conversion entre objets Dart et Maps SQLite peuvent causer des crashes.

Solutions :

- Tester toMap() et fromMap() séparément
- Utiliser toIso8601String() pour les dates
- Stocker booléens comme INTEGER (0 ou 1)
- Utiliser des try-catch pour gérer les erreurs

4 Validation des formulaires

Problème potentiel :

Les validations côté client peuvent être mal implémentées.

Solutions :

- Utiliser GlobalKey pour la validation
- Implémenter des validators personnalisés
- Afficher des messages d'erreur clairs
- Utiliser .trim() pour enlever les espaces

5 Erreurs d'overflow (RenderFlex overflow)

Problème potentiel :

Des erreurs "RenderFlex overflowed" peuvent apparaître lorsque le contenu dépasse l'espace disponible, notamment sur différentes tailles d'écran ou avec du texte long.

Solutions :

- Utiliser Expanded ou Flexible dans les Row/Column
- Envelopper les textes longs dans un widget Flexible
- Ajouter overflow: TextOverflow.ellipsis pour les textes

- Utiliser SingleChildScrollView pour les écrans avec beaucoup de contenu
- Définir maxLines pour les champs de texte
- Tester sur différentes tailles d'écran Android

6 Navigation entre les écrans

Problème potentiel :

La gestion de la navigation peut créer des bugs si les données ne sont pas rafraîchies.

Solutions :

- Recharger les tâches dans initState()
- Utiliser await avec Navigator.push()
- Appeler _loadTasks() après navigation
- Utiliser pushReplacement() pour connexion/déconnexion

7 Tests sur appareil Android

Problème potentiel :

L'application peut fonctionner sur émulateur mais avoir des problèmes sur appareil réel.

Solutions :

- Tester régulièrement sur appareil réel
- Vérifier les permissions Android (Internet)
- Utiliser le mode debug pour identifier les erreurs
- Tester sur plusieurs versions d'Android
- Optimiser les ressources pour la performance

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

✓ Opérations CRUD avec SQLite

✓ Navigation entre écrans

✓ Architecture MVC

✓ Gestion de l'état avec setState()

✓ Authentification Firebase

✓ Validation de formulaires

✓ 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.