1. Introduction

Ce document décrit les spécifications fonctionnelles et techniques du logiciel "mfm (mini finance manager)", une application conçue pour la gestion financière personnelle ou pour de petites entités. L'objectif est de créer un outil simple et intuitif pour le suivi des revenus, des dépenses et la gestion de budgets.

---

2. Objectifs du projet

- **Faciliter le suivi financier** : Permettre aux utilisateurs d'enregistrer et de visualiser leurs transactions financières de manière simple et rapide.
- **Aider à la prise de décision** : Fournir des rapports clairs pour que les utilisateurs puissent analyser leurs habitudes de dépenses et améliorer leur gestion budgétaire.
- **Assurer la sécurité des données** : Protéger les informations financières de l'utilisateur.
- **Offrir une expérience utilisateur fluide** : Proposer une interface simple et intuitive, même pour les non-spécialistes de la finance.

---

3. Fonctionnalités requises

Fonctionnalités principales :

- **Tableau de bord (Dashboard)** : Affichage d'un aperçu visuel de la situation financière (solde actuel, résumé des dépenses/revenus récents).
- **Gestion des transactions** :
    - Enregistrement manuel des revenus et des dépenses.
    - Saisie de la date, du montant, de la description et de la catégorie.
    - Possibilité de modifier et de supprimer les transactions.
- **Catégorisation des transactions** :
    - Gestion des catégories (Alimentation, Logement, Transport, Loisirs, etc.).
    - Affectation d'une catégorie à chaque transaction.
- **Gestion des budgets** :
    - Création de budgets personnalisés pour chaque catégorie (ex. : 200€ par mois pour les loisirs).
    - Suivi en temps réel de la progression du budget et alertes en cas de dépassement.
- **Comptes multiples** :
    - Possibilité de gérer plusieurs comptes (compte courant, épargne, carte de crédit, etc.).

- **Rapports et analyses** :
    - Génération de graphiques et de rapports basés sur les catégories, les dates ou les comptes.
    - Rapports sur les revenus vs. les dépenses.
- **Sécurité** :
    - Protection par mot de passe ou autre mécanisme d'authentification.
    - Chiffrement des données.

---

4. Exigences techniques

- **Architecture** : Application de bureau (desktop) ou web (SaaS).
- **Base de données** : SQLite pour une application de bureau simple, ou PostgreSQL/MySQL pour une application web.
- **Technologies front-end (pour une application web)** : HTML5, CSS3, JavaScript (avec un framework comme React ou Vue.js).
- **Technologies back-end** : Python (avec Flask ou Django), Node.js (avec Express), ou PHP.
- **Déploiement** : Solution autonome pour le desktop ou sur un service cloud pour l'application web.

---

5. Calendrier prévisionnel (estimation)

- **Phase 1 - Planification et conception (1 semaine)** : Finalisation du cahier des charges, maquettage de l'interface utilisateur.
- **Phase 2 - Développement (4 à 6 semaines)** :
    - Développement de l'API (back-end).
    - Création de l'interface utilisateur (front-end).
    - Intégration de la base de données.
- **Phase 3 - Tests et déploiement (1 à 2 semaines)** : Tests de bon fonctionnement, de sécurité et déploiement initial.
- **Phase 4 - Maintenance et améliorations** : Support utilisateur, corrections de bugs et développement de nouvelles fonctionnalités (par exemple, la synchronisation bancaire automatique).

Très bien. En tant que votre gestionnaire de projet, je vais vous guider à travers les prochaines étapes. Le cahier des charges que nous avons établi est notre feuille de route. Maintenant, passons à l'action.

Étape 1 : La Conception (Design)

C'est le moment de visualiser votre application. L'objectif est de créer une interface utilisateur (UI) simple et efficace.

- **Faites des croquis (Wireframes)** : Prenez un papier et un crayon ou utilisez un outil en ligne gratuit (comme Figma ou Balsamiq) pour dessiner les écrans principaux de "mfm". Comment se présentera le tableau de bord ? Où se trouvera le bouton pour ajouter une transaction ? À quoi ressemblera le graphique des dépenses ?
- **Définissez l'expérience utilisateur (UX)** : Pensez à l'enchaînement des actions. Par exemple, après avoir ajouté une transaction, l'utilisateur doit-il revenir au tableau de bord ou voir une confirmation ? L'idée est de rendre le processus aussi fluide que possible.

Étape 2 : Le Choix Technique

Le cahier des charges a déjà mentionné quelques technologies possibles. Maintenant, il faut en choisir une.

- **Application de bureau ou web ?** : Si vous débutez et que c'est un projet personnel, une application de bureau peut être plus simple. Si vous visez une utilisation sur plusieurs appareils, une application web est le bon choix.
- **Les langages de programmation** :
    - **Pour une application de bureau** : Python est un excellent choix pour les débutants (avec des bibliothèques comme Tkinter ou PyQt).
    - **Pour une application web** :
        - **Front-end** : HTML, CSS et JavaScript sont indispensables. Pour simplifier, un framework comme React ou Vue.js est recommandé.
        - **Back-end** : Python avec Flask ou Node.js (JavaScript) avec Express sont des options populaires et bien documentées.
- **Base de données** : SQLite est parfait pour démarrer avec une application de bureau. Pour le web, une base de données relationnelle comme PostgreSQL est un standard.

Étape 3 : Le Développement (Codage)

Maintenant, on met les mains dans le cambouis !

- **Approche itérative** : Ne cherchez pas à tout faire en une seule fois. Découpez le projet en petites "tâches" ou "fonctionnalités" que vous développerez une par une. Par exemple :
    1. Créer la base de données.
    2. Développer la page d'accueil avec le tableau de bord.
    3. Créer la fonctionnalité d'ajout de transaction.

4. Créer la page pour afficher l'historique des transactions.
5. Développer les fonctionnalités de rapport et de graphique.

- **Testez au fur et à mesure** : Après chaque fonctionnalité développée, testez-la pour vous assurer qu'elle fonctionne comme prévu. Cela vous évitera de devoir corriger de nombreux bugs à la fin.

Étape 4 : Les Tests et le Lancement

Une fois que vous avez développé toutes les fonctionnalités de base.

- **Tests utilisateurs** : Demandez à un ami ou à un membre de votre famille de tester votre logiciel. Regardez comment ils l'utilisent et notez les points de confusion ou les bugs qu'ils rencontrent.
- **Lancement (Déploiement)** : Mettez votre application à disposition. Si c'est une application de bureau, vous pouvez la partager en exécutable. Pour le web, vous devrez l'héberger sur un serveur (comme Heroku ou Vercel pour commencer).

Étape 5 : Amélioration Continue

Un logiciel n'est jamais vraiment "fini".

- **Maintenance** : Corrigez les bugs qui pourraient apparaître après le lancement.
- **Nouvelles fonctionnalités** : Ajoutez les fonctionnalités secondaires prévues (comme la synchronisation bancaire ou les alertes de budget) basées sur les retours des utilisateurs.

Rappelez-vous, le plus important est de commencer petit et de progresser pas à pas. Vous avez une vision, maintenant, construisons-la ensemble. N'hésitez pas si vous avez des questions sur l'une de ces étapes.

D'accord. La décision d'utiliser Python et la bibliothèque customtkinter pour l'interface utilisateur clarifie et simplifie notre plan de projet. En tant que votre gestionnaire de projet, voici le plan d'action ajusté, en se concentrant sur les étapes spécifiques pour un développement avec Python et customtkinter.

1. Préparation de l'environnement

- **Installation de Python** : Assurez-vous d'avoir une version récente de Python (3.7 ou plus) installée sur votre machine.
- **Installation de customtkinter** : Ouvrez votre terminal ou votre invite de commande et installez la bibliothèque avec pip :

  Bash

  ```
  pip install customtkinter
  ```

- **Installation d'une base de données** : Pour commencer simplement, installez la bibliothèque sqlite3 qui est incluse par défaut avec Python. Elle vous permet de créer et de gérer une base de données locale dans un simple fichier.

2. Conception de l'interface utilisateur

- **Maquettage avec customtkinter** : Maintenant que vous avez l'outil, traduisez les croquis que nous avons conçus plus tôt en code. customtkinter offre des widgets modernes (boutons, champs de texte, tableaux) qui sont plus esthétiques que ceux de la bibliothèque standard tkinter.
    - **Tableau de bord** : Créez une fenêtre principale avec des labels pour afficher le solde, un tableau pour les transactions récentes et des boutons pour l'ajout de transactions ou l'accès aux rapports.
    - **Formulaire de transaction** : Développez un petit formulaire avec des champs pour la date, la description, le montant et une liste déroulante pour la catégorie.

3. Développement du code

- **Création de la base de données** : Écrivez un script Python pour créer votre base de données SQLite. Vous aurez au moins une table pour les transactions, avec des colonnes comme id, date, montant, description et categorie_id.
- **Logique de l'application** : C'est la partie "cœur" de votre logiciel.
    - Créez des fonctions Python pour ajouter, modifier ou supprimer des transactions dans la base de données.
    - Développez des requêtes SQL pour récupérer les données nécessaires à l'affichage sur le tableau de bord ou dans les rapports.
- **Connexion de l'interface et de la logique** : Liez vos fonctions de gestion de données aux actions de l'utilisateur sur l'interface. Par exemple, le bouton "Ajouter" de l'interface doit appeler la fonction Python qui enregistre la nouvelle transaction dans la base de données.

4. Tests

- **Tests unitaires** : Vérifiez que chaque fonction (ajouter une transaction, calculer le solde) fonctionne correctement de manière isolée.
- **Tests d'intégration** : Testez le flux complet, par exemple en vérifiant qu'une transaction ajoutée via l'interface est bien affichée et prise en compte dans les totaux.

5. Lancement et amélioration

Une fois que votre application est fonctionnelle, vous pourrez la "packager" pour qu'elle puisse être utilisée sur d'autres machines sans avoir à installer Python (en utilisant des outils comme PyInstaller). Ensuite, vous pourrez ajouter les fonctionnalités plus avancées comme la gestion des budgets ou des rapports plus complexes.

Votre choix de technologie est excellent pour un projet de cette envergure. customtkinter vous permettra de créer une interface agréable sans les complexités d'une application web, ce qui est parfait pour se concentrer sur le cœur de l'application : la gestion financière.

Voici la structure complète du projet pour votre logiciel de gestion financière "mfm", organisée de manière logique et facile à suivre. Chaque dossier et fichier a un rôle précis pour assurer la clarté et la maintenance du projet.

```
mfm/
├── app.py
├── database.py
├── main_window.py
├── reports_window.py
├── transactions_window.py
├── assets/
│   ├── images/
│   │   ├── icon.png
│   │   └── logo.png
│   └── fonts/
│       ├── poppins.ttf
│       └── ...
├── data/
│   ├── mfm.db
│   └── ...
├── docs/
│   ├── cahier_des_charges.md
│   ├── design_ui.png
│   └── ...
├── requirements.txt
└── README.md
```

Explication de la structure

- **mfm/** : Le dossier racine du projet. C'est ici que tout le code de votre application sera stocké.

- **app.py** : C'est le point d'entrée de votre application. Ce fichier sera le premier à s'exécuter. Il contiendra la logique de démarrage de l'application et la création de la fenêtre principale.
- **database.py** : Ce fichier est dédié à toute la logique de connexion à la base de données. Il contiendra les fonctions pour :
    - Créer les tables si elles n'existent pas.
    - Ajouter une nouvelle transaction.
    - Mettre à jour une transaction.
    - Supprimer une transaction.
    - Récupérer toutes les transactions, le solde, etc.
- **main_window.py** : Ce fichier gérera la fenêtre principale de l'application (le "Dashboard"). Il contiendra la classe qui définit le design de la fenêtre, les widgets customtkinter (boutons, labels, tableaux) et les fonctions de rappel (callbacks) qui réagissent aux actions de l'utilisateur.
- **transactions_window.py** : Ce fichier gérera la fenêtre dédiée à la saisie, l'édition et la suppression des transactions. Cela permet de garder la logique de l'interface claire et séparée de la fenêtre principale.
- **reports_window.py** : Ce fichier contiendra la logique de l'interface pour la génération de graphiques et de rapports. Il affichera les analyses des dépenses et des revenus par catégorie ou par période.
- **assets/** : Ce dossier est pour toutes les ressources statiques de l'application.
    - images/ : Contient les icônes, logos ou autres images utilisées dans l'interface.
    - fonts/ : Si vous utilisez des polices personnalisées, mettez-les ici.
- **data/** : Ce dossier est destiné aux fichiers générés par l'application.
    - mfm.db : C'est votre base de données SQLite. L'avantage d'utiliser un fichier .db est qu'il est portable et facile à gérer.
- **docs/** : Un dossier de documentation. C'est une bonne pratique de garder une trace de vos documents de projet.
    - cahier_des_charges.md : Une version textuelle de votre cahier des charges.

        - design_ui.png : Des captures d'écran ou des maquettes de votre interface.
- **requirements.txt** : Ce fichier liste toutes les bibliothèques Python nécessaires pour faire fonctionner le projet (customtkinter, par exemple). Il est très utile pour que d'autres personnes (ou vous-même sur un autre ordinateur) puissent installer toutes les dépendances d'un coup en utilisant la commande pip install -r requirements.txt.
- **README.md** : Un fichier important qui explique ce qu'est le projet, comment l'installer et comment le lancer.

Cette structure modulaire vous permettra de travailler sur différentes parties du projet sans que le code ne devienne un "spaghetti", ce qui est essentiel pour un projet réussi et maintenable.

J'ai choisi cette structure de projet pour plusieurs raisons, toutes liées aux meilleures pratiques en matière de développement logiciel. Elle est conçue pour être **modulaire**, **maintenable**, et **évolutive**, des principes fondamentaux pour tout projet, qu'il soit petit ou grand.

1. Séparation des préoccupations (Separation of Concerns)

Chaque fichier ou dossier a une responsabilité unique et bien définie.

- app.py gère le démarrage de l'application et l'initialisation de l'interface graphique. C'est le chef d'orchestre.
- database.py se concentre uniquement sur l'interaction avec la base de données (lecture, écriture, etc.). Il ne se soucie pas de l'interface utilisateur.
- main_window.py, transactions_window.py, etc., se concentrent exclusivement sur la présentation des données à l'utilisateur et sur la gestion des interactions (clics, saisies).

Cette séparation signifie que si vous devez changer quelque chose dans votre base de données (par exemple, passer de SQLite à une base de données MySQL), vous n'aurez à modifier que le fichier database.py. Le reste de l'application restera inchangé, car il ne dépend que des fonctions de ce fichier, et non de son contenu.

---

2. Clarté et organisation

Une structure de dossiers bien organisée rend le projet plus facile à comprendre et à naviguer, pour vous-même et pour d'autres développeurs.

- Le dossier assets/ regroupe toutes les ressources graphiques, ce qui évite de les mélanger avec le code source.
- Le dossier docs/ est un endroit central pour tous les documents importants, ce qui est crucial pour le suivi du projet.
- Le fichier requirements.txt est standard dans l'écosystème Python. Il permet de garantir que l'environnement de développement est le même pour tout le monde, en listant les dépendances nécessaires.

---

3. Maintenance et débogage

Lorsque le code est organisé de manière logique, il est beaucoup plus facile de trouver et de corriger les erreurs (débogage). Si un problème survient avec l'ajout d'une transaction, vous savez immédiatement qu'il faut chercher la cause dans transactions_window.py (pour l'interface) ou dans database.py (pour la logique de base de données). Vous n'avez pas à chercher dans des centaines de lignes de code dans un seul et unique fichier.

---

4. Évolutivité

Cette structure est conçue pour grandir avec le projet. Si vous décidez d'ajouter de nouvelles fonctionnalités complexes, comme la gestion de multiples utilisateurs ou une synchronisation avec une API bancaire, vous pouvez simplement ajouter de nouveaux fichiers ou dossiers (user_manager.py, api_connector.py) sans perturber l'organisation existante.

En résumé, cette structure n'est pas juste une liste de fichiers aléatoires ; c'est un plan stratégique pour un développement **efficace et pérenne**. Elle vous permet de construire un logiciel solide, brique par brique, sans créer de confusion.