**🎵 MusicPartMate - Analyse détaillée des dépendances**

**📋 Table des matières**

* [📄 main.py](https://claude.ai/chat/e5990f2a-ffb8-4302-ac1a-dd2879cd6061#main-py)
* [📊 models/song.py](https://claude.ai/chat/e5990f2a-ffb8-4302-ac1a-dd2879cd6061#models-song-py)
* [📚 models/library.py](https://claude.ai/chat/e5990f2a-ffb8-4302-ac1a-dd2879cd6061#models-library-py)
* [🖥️ ui/main\_window.py](https://claude.ai/chat/e5990f2a-ffb8-4302-ac1a-dd2879cd6061#ui-main-window-py)
* [📄 ui/document\_viewer.py](https://claude.ai/chat/e5990f2a-ffb8-4302-ac1a-dd2879cd6061#ui-document-viewer-py)
* [🎵 ui/media\_player.py](https://claude.ai/chat/e5990f2a-ffb8-4302-ac1a-dd2879cd6061#ui-media-player-py)
* [✏️ ui/song\_dialog.py](https://claude.ai/chat/e5990f2a-ffb8-4302-ac1a-dd2879cd6061#ui-song-dialog-py)
* [⚙️ utils/config.py](https://claude.ai/chat/e5990f2a-ffb8-4302-ac1a-dd2879cd6061#utils-config-py)
* [📁 utils/file\_utils.py](https://claude.ai/chat/e5990f2a-ffb8-4302-ac1a-dd2879cd6061#utils-file-utils-py)

**📄 main.py**

**Rôle** : Point d'entrée de l'application

**🔧 Méthodes**

* main() : Configure et lance l'application

**🔗 Dépendances UTILISÉES**

* PySide6.QtWidgets.QApplication
* src.ui.main\_window.MainWindow → MainWindow()

**📤 Méthodes APPELÉES par d'autres fichiers**

* Aucune (point d'entrée)

**📊 models/song.py**

**Rôle** : Modèle de données pour une chanson

**🔧 Méthodes/Propriétés**

* \_\_post\_init\_\_() : Normalise les données après création
* display\_name : Propriété - nom d'affichage
* has\_documents, has\_audio, has\_video, has\_link : Propriétés booléennes
* primary\_document : Propriété - premier document
* add\_document(file\_path) : Ajoute un document
* add\_audio(file\_path) : Ajoute un fichier audio
* add\_video(file\_path) : Ajoute un fichier vidéo
* add\_link(file\_path) : Ajoute un lien
* remove\_media(file\_path) : Supprime un média
* get\_all\_media\_files() : Retourne tous les fichiers
* validate() : Valide la chanson
* is\_valid() : Vérifie la validité
* to\_dict() : Convertit en dictionnaire
* from\_dict(data) : Crée depuis un dictionnaire
* from\_json(json\_str) : Crée depuis JSON
* to\_json() : Convertit en JSON

**🔧 Fonction utilitaire**

* create\_song\_from\_folder(folder\_path, title, artist) : Crée une chanson depuis un dossier

**🔗 Dépendances UTILISÉES**

* pathlib.Path
* json
* dataclasses

**📤 Méthodes APPELÉES par d'autres fichiers**

* **main\_window.py** : Song(), display\_name, is\_valid(), has\_documents, has\_audio, has\_video, primary\_document, validate()
* **song\_dialog.py** : Song(), to\_dict(), from\_dict(), tous les add\_\*(), is\_valid()
* **library.py** : Song(), is\_valid(), to\_dict(), from\_dict(), display\_name
* **main.py via create\_song\_from\_folder** : create\_song\_from\_folder()

**📚 models/library.py**

**Rôle** : Gestionnaire de la bibliothèque musicale

**🔧 Classes et méthodes**

**LibraryConfig (dataclass)**

* Configuration de la bibliothèque

**Library**

* \_\_init\_\_(config) : Initialise la bibliothèque
* add\_observer(callback) : Ajoute un observateur
* remove\_observer(callback) : Supprime un observateur
* \_notify\_observers(event\_type, song) : Notifie les observateurs
* song\_count : Propriété - nombre de chansons
* artists, styles : Propriétés - listes uniques
* add\_song(song) : Ajoute une chanson
* remove\_song(song) : Supprime une chanson
* update\_song(song) : Met à jour une chanson
* find\_song\_by\_id(song\_id) : Trouve par ID
* find\_duplicate(song) : Cherche les doublons
* search\_songs(query, search\_in) : Recherche des chansons
* filter\_by\_artist(artist) : Filtre par artiste
* filter\_by\_style(style) : Filtre par style
* get\_songs\_sorted(sort\_by, reverse) : Retourne les chansons triées
* save\_library() : Sauvegarde sur disque
* load\_library() : Charge depuis le disque
* export\_library(export\_path, format\_type) : Exporte la bibliothèque
* get\_statistics() : Statistiques de la bibliothèque

**🔗 Dépendances UTILISÉES**

* models.song.Song → Song(), is\_valid(), to\_dict(), from\_dict()
* json, pathlib.Path, datetime

**📤 Méthodes APPELÉES par d'autres fichiers**

* **main\_window.py** : Library(), add\_song(), remove\_song(), update\_song(), search\_songs(), get\_songs\_sorted(), song\_count, get\_statistics(), export\_library(), add\_observer()

**🖥️ ui/main\_window.py**

**Rôle** : Fenêtre principale de l'application

**🔧 Classes et méthodes**

**ImportWorker (QThread)**

* \_\_init\_\_(folder\_paths) : Initialise le worker
* stop() : Arrête l'import
* run() : Effectue l'import en arrière-plan

**LibraryTreeWidget (QTreeWidget)**

* on\_item\_clicked(item, column) : Gère les clics
* show\_context\_menu(position) : Menu contextuel
* load\_library(library) : Charge la bibliothèque dans l'arbre

**SearchWidget (QWidget)**

* setup\_ui() : Configure l'interface de recherche
* on\_search\_changed() : Gère les changements de recherche
* clear\_search() : Efface la recherche

**MainWindow (QMainWindow)**

* \_\_init\_\_() : Initialise la fenêtre principale
* setup\_ui() : Configure l'interface utilisateur
* create\_library\_panel() : Crée le panel de bibliothèque
* create\_content\_panel() : Crée le panel de contenu
* setup\_menu\_bar() : Configure la barre de menu
* setup\_toolbar() : Configure la barre d'outils
* setup\_connections() : Configure les connexions
* load\_library() : Charge la bibliothèque
* on\_song\_selected(song) : Chanson sélectionnée
* on\_media\_selected(media\_type, file\_path) : Média sélectionné
* on\_search\_requested(query, filter\_type) : Recherche demandée
* new\_song() : Nouvelle chanson
* edit\_current\_song() : Édite la chanson
* delete\_current\_song() : Supprime la chanson
* import\_songs() : Importe des chansons
* export\_library() : Exporte la bibliothèque
* show\_statistics() : Affiche les statistiques

**🔗 Dépendances UTILISÉES**

* models.song.Song → Song(), display\_name, primary\_document
* models.library.Library → Library(), add\_song(), search\_songs(), etc.
* utils.config.config\_manager → get\_library\_path(), config
* utils.file\_utils → scan\_folder\_for\_media(), get\_folder\_statistics()
* ui.document\_viewer.DocumentViewer → load\_document()
* ui.media\_player.MediaPlayer → load\_media(), load\_youtube\_url()
* ui.song\_dialog.SongDialog → SongDialog()

**📤 Méthodes APPELÉES par d'autres fichiers**

* **main.py** : MainWindow()

**📄 ui/document\_viewer.py**

**Rôle** : Widget pour afficher les documents

**🔧 Classes et méthodes**

**DocumentLoadWorker (QThread)**

* \_\_init\_\_(file\_path) : Initialise le worker
* stop() : Arrête le chargement
* run() : Charge le document en arrière-plan
* load\_pdf() : Charge un PDF
* load\_text() : Charge un fichier texte
* load\_docx() : Charge un fichier Word
* load\_odt() : Charge un fichier ODT
* load\_image() : Charge une image

**DocumentViewer (QWidget)**

* setup\_ui() : Configure l'interface
* create\_toolbar(parent\_layout) : Crée la barre d'outils
* load\_document(file\_path) : Charge et affiche un document
* on\_document\_loaded(pages) : Document chargé
* display\_current\_page() : Affiche la page courante
* display\_image(pixmap) : Affiche une image
* display\_text(text, images) : Affiche du texte
* display\_html(html\_content) : Affiche du HTML
* show\_error(message) : Affiche une erreur
* previous\_page(), next\_page() : Navigation
* zoom\_in(), zoom\_out(), fit\_to\_window() : Zoom

**🔗 Dépendances UTILISÉES**

* PyMuPDF (fitz) : Pour les PDF
* python-docx : Pour les fichiers Word
* mammoth : Pour meilleur rendu DOCX
* odfpy : Pour les fichiers ODT
* Pillow : Pour les images

**📤 Méthodes APPELÉES par d'autres fichiers**

* **main\_window.py** : load\_document(), show\_welcome\_message()

**🎵 ui/media\_player.py**

**Rôle** : Widget pour la lecture multimédia

**🔧 Méthodes**

* \_\_init\_\_() : Initialise le lecteur
* setup\_full\_player() : Configure le lecteur complet
* setup\_fallback\_player() : Configure le lecteur de fallback
* setup\_full\_ui() : Interface complète
* setup\_connections() : Connexions des signaux
* load\_media(file\_path) : Charge un média (PRINCIPALE)
* load\_media\_fallback(file\_path) : Mode fallback
* setup\_video\_display() : Configure l'affichage vidéo
* hide\_video\_display() : Masque la vidéo
* open\_external() : Ouvre avec lecteur externe
* play\_pause() : Lecture/pause
* stop() : Arrête la lecture
* set\_position(position) : Change la position
* set\_volume(volume) : Change le volume
* clear\_media() : Vide le lecteur
* load\_youtube\_url(url, title) : Charge une URL YouTube
* open\_youtube\_url() : Ouvre YouTube dans navigateur

**🔗 Dépendances UTILISÉES**

* PySide6.QtMultimedia : QMediaPlayer, QAudioOutput
* PySide6.QtMultimediaWidgets : QVideoWidget
* webbrowser, subprocess : Pour ouverture externe

**📤 Méthodes APPELÉES par d'autres fichiers**

* **main\_window.py** : load\_media(), load\_youtube\_url(), clear\_media()

**✏️ ui/song\_dialog.py**

**Rôle** : Dialog pour créer/éditer les chansons

**🔧 Classes et méthodes**

**LinkDialog (QDialog)**

* \_\_init\_\_(parent) : Initialise le dialog
* get\_link\_data() : Retourne les données saisies

**MediaListWidget (QListWidget)**

* \_\_init\_\_(media\_type) : Initialise la liste
* update\_placeholder() : Met à jour le placeholder
* add\_file(file\_path) : Ajoute un fichier
* add\_link(file\_path) : Ajoute un lien
* remove\_selected() : Supprime la sélection
* get\_files() : Retourne la liste des fichiers
* get\_links() : Retourne la liste des liens
* clear\_files() : Vide la liste

**SongDialog (QDialog)**

* \_\_init\_\_(song, parent) : Initialise le dialog
* setup\_ui() : Configure l'interface
* create\_info\_tab() : Onglet informations
* create\_media\_tab() : Onglet médias
* create\_metadata\_tab() : Onglet métadonnées
* load\_song\_data() : Charge les données de la chanson
* validate\_form() : Valide le formulaire
* open\_link\_dialog() : Ouvre le dialog de lien
* add\_files() : Ajoute des fichiers
* import\_folder() : Importe un dossier
* process\_files(files) : Traite une liste de fichiers
* save\_song() : Sauvegarde la chanson
* collect\_metadata() : Collecte les métadonnées

**🔗 Dépendances UTILISÉES**

* models.song.Song → Song(), tous les attributs et méthodes
* utils.config.config\_manager → is\_supported\_media()
* utils.file\_utils → scan\_folder\_for\_media(), get\_file\_size\_human()

**📤 Méthodes APPELÉES par d'autres fichiers**

* **main\_window.py** : SongDialog(), exec()

**⚙️ utils/config.py**

**Rôle** : Configuration globale de l'application

**🔧 Classes et méthodes**

**AppConfig (dataclass)**

* Configuration avec tous les paramètres
* \_\_post\_init\_\_() : Initialise les listes par défaut

**ConfigManager**

* \_\_init\_\_(config\_path) : Initialise le gestionnaire
* load\_config() : Charge la configuration
* save\_config() : Sauvegarde la configuration
* get\_data\_dir() : Retourne le dossier de données
* get\_library\_path() : Chemin vers la bibliothèque
* get\_supported\_formats() : Formats supportés
* get\_cache\_dir() : Dossier de cache
* is\_supported\_document(file\_path) : Vérifie si document supporté
* is\_supported\_audio(file\_path) : Vérifie si audio supporté
* is\_supported\_video(file\_path) : Vérifie si vidéo supporté
* is\_supported\_media(file\_path) : Type de média supporté

**Fonctions utilitaires**

* get\_config() : Configuration actuelle
* save\_config() : Sauvegarde
* get\_supported\_formats() : Tous les formats
* get\_all\_supported\_extensions() : Toutes les extensions

**🔗 Dépendances UTILISÉES**

* json, pathlib.Path, dataclasses

**📤 Méthodes APPELÉES par d'autres fichiers**

* **main\_window.py** : config\_manager.get\_library\_path(), config\_manager.config
* **song\_dialog.py** : config\_manager.is\_supported\_media(), get\_supported\_formats()
* **library.py** : Via LibraryConfig

**📁 utils/file\_utils.py**

**Rôle** : Utilitaires pour la gestion des fichiers

**🔧 Fonctions utilitaires**

* get\_file\_size\_human(file\_path) : Taille lisible
* get\_file\_hash(file\_path, algorithm) : Hash du fichier
* is\_file\_accessible(file\_path) : Vérifie l'accessibilité
* safe\_filename(filename, replacement) : Nettoie le nom
* create\_song\_folder\_structure(base\_path, artist, title) : Structure dossier
* scan\_folder\_for\_media(folder\_path, extensions) : Scanne les médias
* copy\_file\_to\_song\_folder() : Copie fichier
* move\_file\_to\_song\_folder() : Déplace fichier
* get\_mime\_type(file\_path) : Type MIME
* is\_image\_file(), is\_audio\_file(), is\_video\_file(), is\_document\_file() : Tests de type
* cleanup\_empty\_folders() : Nettoie dossiers vides
* find\_duplicate\_files() : Trouve doublons
* validate\_file\_structure() : Valide structure
* get\_folder\_statistics() : Stats dossier
* create\_media\_symlinks() : Liens symboliques

**🔗 Dépendances UTILISÉES**

* pathlib.Path, os, shutil, mimetypes, hashlib

**📤 Méthodes APPELÉES par d'autres fichiers**

* **main\_window.py** : scan\_folder\_for\_media(), get\_folder\_statistics()
* **song\_dialog.py** : scan\_folder\_for\_media(), get\_file\_size\_human()
* **song.py (create\_song\_from\_folder)** : scan\_folder\_for\_media() (indirectement)

**🔄 Flux de données principal**

**1. Démarrage de l'application**

main.py → MainWindow → Library → config\_manager

**2. Création d'une nouvelle chanson**

MainWindow.new\_song() → SongDialog → Song → config\_manager/file\_utils

**3. Sélection d'une chanson**

LibraryTreeWidget.on\_item\_clicked() → MainWindow.on\_song\_selected() → DocumentViewer.load\_document()

**4. Import de chansons**

MainWindow.import\_songs() → ImportWorker → create\_song\_from\_folder() → Library.add\_song()

**5. Lecture de média**

LibraryTreeWidget.on\_item\_clicked() → MainWindow.on\_media\_selected() → MediaPlayer.load\_media()

Cette architecture montre une séparation claire des responsabilités avec des modèles de données robustes, une interface utilisateur modulaire, et des utilitaires réutilisables.

Une image contenant texte, capture d’écran, Police, Rectangle

Le contenu généré par l’IA peut être incorrect.