TOULOUSE INP N7

ENSEEIHT

Rapport 2 - Projet long TOB
Logiciel de montage vidéo

Equipe KL-08 :
Alexandre Lescot
Camille Meyer
Doryan Benoit
Iman-Norr Draou
Mathier Trahand
Oscar Mautin
Sophie Girardot
Yacine Meziani

Professeur :
G. Dupont

May 12, 2025

Contents

1 Objectif du projet

Ce projet vise à développer une application de montage vidéo intuitive et performante, accessible aussi bien aux professionnels qu'aux débutants. L'objectif est de proposer un outil ergonomique permettant d'importer, d'éditer et de manipuler facilement du contenu multimédia.

2 Préparation du projet

2.1 MVP

2.1.1 Timeline

La timeline constitue la base du logiciel de montage. Il s'agit d'une frise chronologique sur laquelle les utilisateurs peuvent organiser leurs fichiers multimédias (vidéos, audios, images, textes). L'objectif est d'intégrer un système de *drag and drop* permettant d'ajouter du contenu à la timeline et de le visualiser en temps réel dans la *preview*.

Il sera également possible de manipuler les éléments sur la timeline : déplacer une vidéo de quelques secondes, assembler plusieurs fichiers ou ajuster leur durée. Les pistes vidéo (vidéos, images, textes) seront distinctes des pistes audio pour une meilleure gestion du contenu.

2.1.2 Preview

Un espace de prévisualisation sera intégré dans l'interface, permettant aux utilisateurs de visualiser les modifications effectuées sur la timeline en temps réel. La prévisualisation sera située dans la partie supérieure droite de la fenêtre, selon la conception définie.

2.1.3 Importation de vidéos

L'application offrira la possibilité d'importer des vidéos à partir du stockage local. Les utilisateurs pourront parcourir leurs documents et sélectionner un fichier à intégrer dans leur projet.

2.1.4 Gestion des fichiers importés

Une bibliothèque de médias affichera les fichiers importés, facilitant leur accès et leur organisation. Cet espace sera situé en haut à gauche de l'interface, conformément au design prévu.

2.1.5 Ajout de vidéos à la timeline

Les vidéos importées pourront être ajoutées à la timeline pour être manipulées. L'utilisateur pourra sélectionner un point précis de la vidéo à afficher en *preview*, à l'aide d'un curseur dédié. D'autres actions de modification seront ajoutées ultérieurement.

2.1.6 Lecture et manipulation des fichiers audio et vidéo

Les fichiers vidéo et audio présents sur la timeline seront lisibles directement via la *preview*, permettant un contrôle précis du montage.

2.2 Fonctionnalités intermédiaires

2.2.1 Découpage de vidéos

L'application permettra de découper une vidéo en plusieurs segments directement sur la timeline, à l'aide d'un curseur de sélection. Chaque segment pourra être déplacé indépendamment, supprimé ou réorganisé.

2.2.2 Fusion de vidéos

Les utilisateurs pourront assembler plusieurs vidéos pour en faire un seul fichier, facilitant le montage et l'édition de contenus combinés.

2.2.3 Édition et lecture des fichiers audio

Un fichier audio pourra être manipulé dans la timeline de la même manière qu'une vidéo. L'utilisateur pourra écouter un fichier, ajuster son placement et effectuer des modifications basiques.

2.2.4 Ajout d'images à la timeline

Les images pourront être intégrées dans la timeline et manipulées comme des vidéos, avec la possibilité de les visualiser dans la *preview*.

2.2.5 Exportation du montage

L'application offrira une option d'export permettant de regrouper l'ensemble des éléments montés sur la timeline en un fichier final unique (ex. MP4).

2.3 Fonctionnalités supplémentaires

2.3.1 Ajout de texte à la timeline

L'application permettra d'ajouter du texte personnalisable sur la timeline (possibilité de personnalisation de la police, la taille, la couleur, etc). L'utilisateur pourra définir la durée d'affichage du texte et le manipuler comme une vidéo ou une image. Ce texte pourra être positionné et édité.

2.4 Croquis

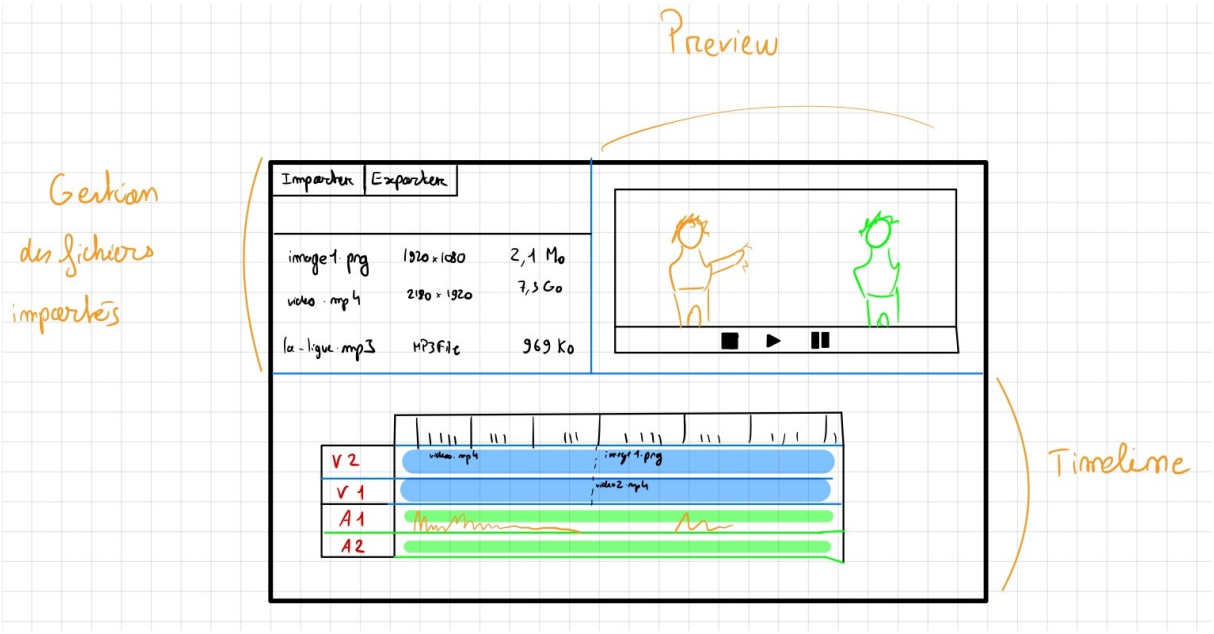

Figure 1: Premier croquis de l'application

3 Réalisation du projet

3.1 Conception [It. 0]

3.2 Mise en place de l'environnement [It. 0 et 1]

L'itération 1 a principalement porté sur la mise en place de l'environnement de développement, une phase qui a pris plus de temps que prévu en raison de la diversité des systèmes d'exploitation utilisés par les membres de l'équipe.

3.2.1 Spécification initiale et planification

Nous avons tout d'abord formalisé l'ensemble des fonctionnalités, rédigé en LaTeX. Le code source de cette spécification se trouve dans le répertoire `docs/fonctionnalites`. À partir de ce référentiel, un tableau Kanban a été mis en place pour visualiser l'avancement, et des *epics* ont été créés pour chacune des fonctionnalités définies, afin de faciliter le découpage et le suivi des tâches.

3.2.2 Choix et configuration des bibliothèques

Pour la partie interface utilisateur, nous avons retenu `JavaFX` (avec *Scene Builder*) et pour le traitement bas-niveau de la vidéo, `GStreamer`. L'installation de `GStreamer` s'effectue sans difficulté sous Linux et macOS, mais s'est avérée instable sur Windows pour des raisons indéterminées. Afin d'uniformiser l'environnement et d'éviter les incompatibilités, nous avons opté pour une solution de conteneurisation.

3.2.3 Conteneurisation via Docker

Le recours à Docker s'est imposé pour assurer la portabilité et la cohérence de notre environnement de développement, indépendamment du système hôte :

- **Isolation des dépendances** : chaque conteneur embarque les versions exactes de JavaFX, GStreamer et de leurs bibliothèques associées, évitant ainsi les conflits de versions ou de configuration entre Windows, Linux et macOS.
- **Reproductibilité** : grâce à un Dockerfile, tout membre de l'équipe peut reconstruire, à l'identique, l'image de développement avec un simple `docker build`, garantissant que le programme de test `FXPlayer` fonctionne de manière identique sur toutes les machines.
- **Simplicité de déploiement** : la conteneurisation évite l'installation manuelle de GStreamer sur l'hôte Windows, problème dont la cause exacte (versions de DLL manquantes, conflits de chemin, etc.) n'a pas pu être résolue en temps utile.

La création d'un Dockerfile pleinement opérationnel a néanmoins présenté plusieurs challenges :

1. **Choix de l'image de base** : après plusieurs itérations, nous avons retenu `openjdk:17-jdk-slim` pour sa légèreté et sa compatibilité JavaFX ; un premier prototype basé sur `ubuntu:22.04` était trop volumineux et nécessitait trop de traitements de nettoyage.
2. **Installation de GStreamer et plug-ins** : il a fallu identifier et installer les paquets `gstreamer1.0-plugins-base`, `-good`, `-bad` et `-ugly`, ainsi que leurs dépendances système (libc, libglib), puis configurer `LD_LIBRARY_PATH` pour que JavaFX puisse les localiser.
3. **Gestion des volumes et des permissions** : pour monter un répertoire de projet en lecture/écriture, nous avons dû ajuster les UID/GID dans le conteneur et ajouter un utilisateur non-root afin de respecter les bonnes pratiques de sécurité.
4. **Optimisation de l'image** : un build multi-étapes a été mis en place pour séparer la compilation Java et l'exécution, réduisant considérablement la taille finale de l'image (passée de 1,2 Go à 400 Mo).
5. **Validation automatique** : enfin, un script d'entrée a été développé pour lancer `FXPlayer` au démarrage du conteneur et signaler par code de sortie le succès ou l'échec de l'initialisation des bibliothèques.

3.2.4 Gestion du dépôt et workflow collaboratif

Compte tenu des contraintes d'accès au réseau INP (nécessité d'un VPN), le code source principal est hébergé sur GitHub. À intervalles réguliers, un mirroring automatique est effectué vers notre instance GitLab interne. Pour limiter les conflits de fusion, chaque membre utilise une branche dédiée pour ses développements. Une fois le travail terminé et validé localement, une demande de fusion (*merge request*) est émise vers la branche `main`, après revue du code et tests unitaires.

3.3 Développement des fonctionnalités principales (MVP) [It. 2]

Lors de cette deuxième itération, plusieurs fonctionnalités clés ont été développées ou finalisées, tout en amorçant une structuration plus claire de l'architecture logicielle. Le travail a été réparti autour de trois axes principaux : la gestion des imports et des clips, l'intégration de la timeline, et la mise en place d'une prévisualisation fonctionnelle.

3.3.1 Prévisualisation et gestion des flux

Une première version du système de prévisualisation a été implémentée. Elle repose sur une pipeline qui permet de récupérer une vidéo d'exemple en ligne. Les flux audio et vidéo sont désormais gérés dynamiquement, et connectés aux sorties (écran et haut-parleur) au moment de l'exécution.

3.3.2 Modélisation du modèle de données

Un diagramme de classes a été produit afin de modéliser la structure de la partie *model* du logiciel. Deux classes principales ont été introduites :

- `Clip`, définie sous forme d'interface, représentant une séquence vidéo ou audio.
- `VideoProject`, une classe centrale qui regroupe les informations relatives aux vidéos importées ainsi qu'à la timeline du projet.

3.3.3 Importation de vidéos

La fonctionnalité d'import a été finalisée. Elle comprend :

- Un bouton pour importer des vidéos via une fenêtre système.
- Une `TableView` listant les vidéos importées avec leurs métadonnées (miniature, nom, taille, dimensions, durée, date de création).
- Une barre de recherche permettant de filtrer les vidéos par nom.
- Un bouton pour supprimer une vidéo de la liste.
- Une fonctionnalité de glisser (*drag*) affichant la miniature de la vidéo (sans gestion du *drop* à ce stade).

3.3.4 Création de clips depuis les vidéos importées

Un travail spécifique a été mené sur la création d'un objet `Clip` à partir d'une vidéo :

- Implémentation de la classe `GStreamerVideoImporter`, avec la méthode `importVideo` qui crée un `ImportedClip` à partir d'une URI.
- Extraction d'informations locales (nom, type) sans passer par GStreamer.
- Utilisation de GStreamer pour récupérer la durée, la taille et une miniature (prise à 5% ou à 1 seconde).
- Redimensionnement de la miniature pour une taille standard.

3.3.5 Travail sur la timeline

Des éléments visuels ont été ajoutés pour la timeline :

- Réservation d'espaces pour l'échelle de temps, le chronomètre et les contrôles.
- Création des boutons `addTrack`, `supprimerTrack`, et `séparer` (seul le premier est fonctionnel).
- Ajout d'un champ texte pour le chronomètre (non encore lié aux données).

3.3.6 Implémentation de la classe `Track`

Les méthodes suivantes ont été développées :

- `addTimelineObject` : insertion ordonnée selon le début des clips.
- `getObjectAtTime` : identification de l'objet actif à un instant donné.
- `changeTimelineName` : renommage de la piste.
- `getTotalDuration` : durée totale de la piste.
- `enableDrop` : amorce de la gestion du glisser-déposer.

3.3.7 Ajout d'un curseur temporel

Un curseur permettant de naviguer dans la timeline a été créé :

- Ajout de l'affichage dans le fichier FXML dédié.
- Mise en place d'une nouvelle classe `Curseur` pour gérer ce composant.
- Le curseur est visible mais non encore synchronisé avec l'affichage de la prévisualisation.