

RAPPORT TECHNIQUE
PRÉSENTÉ À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
DANS LE CADRE DU COURS GTI795 PROJET DE FIN D'ÉTUDE EN GÉNIE DES TI ET
LOG795 PROJET DE FIN D'ÉTUDES EN GÉNIE LOGICIEL

PROJET DE FIN D'ÉTUDE 008
Reconnaissance visuelle de partitions musicales avec OMR

LAFLÈCHE CHEVRETTE
CHEL70100001
JÉRÉMY DA COSTA
DACJ03029800
PHILIPPE A. LANGEVIN
LANP1302806
CHARLIE PONCSAK
PONC06109907
XAVIER JEANSON
JEAX04039709

DÉPARTEMENT DE GÉNIE LOGICIEL ET DES TI

Professeur-superviseur

Camille Coti

Pascal Giard

MONTREAL, 15 AOÛT 2024
ÉTÉ 2024

REMERCIEMENTS

Nous aimerions remercier nos professeurs Camille Coti et Pascal Giard pour leur soutien durant l'entièreté du projet. Ils nous ont été d'une grande aide lors de certaines décisions et pour la rédaction de notre plan de projet et de ce rapport. Nous aimerions aussi remercier Adrien Misandeau et le club MusiquÉTS de nous avoir permis de travailler sur cet excitant projet.

RÉSUMÉ

Le club étudiant MusiquÉTS de l'école de technologie supérieure a mandaté notre équipe de réaliser le projet de fin d'études 008 - Reconnaissance visuelle de partitions musicales avec OMR. Celui-ci consiste en une application implémentant un système de reconnaissance optique de musique. Cette application doit pouvoir transformer un fichier (pdf, jpg, png, etc) représentant une partition musicale en sa version MIDI. Le fichier d'entrée peut être une partition faite par ordinateur ou une partition manuscrite. Le fichier MIDI doit, quant à lui, être une représentation sonore fidèle de la partition originale.

TABLE DES MATIÈRES

RÉSUMÉ	3
TABLE DES MATIÈRES	4
LISTE DES TABLEAUX	6
LISTE DES FIGURES	7
LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES	8
INTRODUCTION	9
PROBLÉMATIQUE ET CONTEXTE	10
OBJECTIFS DU PROJET	11
Objectifs principaux	11
Objectifs secondaires (non obligatoires)	12
Retombées du projet	12
MÉTHODOLOGIE	13
Tableau des échéanciers	15
Composition de l'équipe et rôle	17
Retour sur les rencontres suite au projet	18
LIVRABLES ET PLANIFICATION	19
Description des artefacts	19
Planification	19
RISQUES	20
TECHNIQUES ET OUTILS	21
Outils et techniques de gestion de projet	21
Outils de communication	22
Technologies et outils de développement.	22
OBJECTIFS DE RÉUSSITE	23
CONTRAINTES	24
Contraintes fonctionnelles	24
Contraintes non fonctionnelles	25
Aptitude fonctionnelle	25
Efficacité de performance	25
Compatibilité	25
Utilisabilité	25
Fiabilité	25
Maintenabilité	26
TESTS	27

Applications Maestro	27
Application Audiveris	28
DÉPLOIEMENT	29
Déployer l'application en local	29
Déployer l'image dans le Docker hub	30
FONCTIONNEMENT	31
PROBLÈMES CONNUS	33
Tempos	33
Tempos irréguliers et uniques	33
Symboles de tempo mal annotée	34
Symboles de tempo non reconnus	34
Outil UI de modification de tempo non intuitif	34
Partitions avec paroles	35
Partitions manuscrites	36
Audiveris	37
Signes naturels dans l'armure	37
Changement d'armure	37
Tiges opposées	37
RECOMMANDATIONS	38
Améliorations de l'interface utilisateur	38
Révision des champs d'entrée pour la modification de tempo	38
Ajout d'une étape de pré-conversion	38
Barre de progression de conversion	38
Ajout de message d'erreurs plus significatif	39
Fonctionnalité de prise de photo	39
Améliorations de l'application serveur	39
Adopter un seul langage pour l'application serveur	39
Extraction de l'Engin Audiveris pour Optimiser la Conversion	39
Ajout d'un Pipeline de Développement et d'Intégration Continue	40
CONCLUSION	41
LISTE DE RÉFÉRENCES	42
ANNEXE A : PLAN DE TRAVAIL	43
ANNEXE B : LISTE DES ÉLÉMENTS D'UNE PARTITION	44

LISTE DES TABLEAUX

Tableau 1 : Échéanciers	16
Tableau 2 : Composition de l'Équipe	17
Tableau 3 : Artéfacts	19
Tableau 4 : Risques	20
Tableau 5 : Plan de travail	44

LISTE DES FIGURES

Figure 1 : Schématisation des étapes et tâches d'un sprint de deux semaines	14
Figure 2 : Configuration des tests dans Gradle	27
Figure 3 : Fonctionnement de l'image Docker	30
Figure 4 : Fonctionnement de l'application Maestro	32
Figure 5 : Exemple d'un tempo et humeur textuels uniques	33
Figure 6 : Partition de Hey Jude avec paroles sous la portée	35
Figure 7 : Fonctionnement du classificateur IA de symboles musicaux	36

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

OMR : Optical Music Recognition

MIDI : Musical Instrument Digital Interface

PDF : Portable Document Format

API : Application Programming Interface

IDE : Integrated Development Environment

GUI : Graphical User Interface

MXL : Fichier MusicXML

XML : eXtensible Markup Language

Consulter l'[annexe B](#) pour une description des symboles musicaux.

INTRODUCTION

La musique a toujours été un des meilleurs moyens pour l'humain de s'exprimer culturellement et de partager ses émotions. Dans les dernières années, les ordinateurs ont commencé à prendre une place de plus en plus grande dans la création musicale, que ce soit dans la création de musique électronique autant que dans l'enregistrement et dans le mixage sonore de performances musicales humaines. Le projet que nous présentons aujourd'hui permettrait aux machines de performer des œuvres musicales connues sans l'intervention d'humains dans le processus. En effet, le projet d'envergure du club étudiant MusiquÉTS est de construire des instruments de musique mécaniques et automatiques.

Le premier instrument envisagé par le club est un piano qui joue tout seul. Le moyen le plus simple de fournir les instructions des notes à jouer pour cet instrument automatisé est de lui fournir un fichier MIDI. Ceux-ci sont couramment utilisés en composition musicale pour représenter les notes musicales sur une ligne du temps. Plusieurs attributs sont associés à ces notes, comme la vélocité, la vitesse, la longueur et bien d'autres. Le club nous a donc contactés pour les aider sur ce projet, plus précisément sur la transcription des partitions afin qu'elles soient lisibles par les instruments mécaniques.

Dans ce rapport, nous décrivons le problème auquel le club faisait face et comment nous avons répondu à ce problème. Nous expliquons les différentes pistes envisagées et celle retenue. Les méthodes utilisées pour le développement et la gestion du code sont ensuite présentées. Les différents enjeux, les résultats et les améliorations possibles sont finalement expliqués en détail.

PROBLÉMATIQUE ET CONTEXTE

Le club étudiant MusiquÉTS travaille actuellement sur le projet MAESTRO, qui consiste à développer des instruments capables de jouer des partitions musicales de manière autonome. L'un des défis majeurs du projet est de concevoir et d'implémenter un système de reconnaissance optique de musique (OMR) capable de générer automatiquement les fichiers MIDI nécessaires pour MAESTRO.

Actuellement, MAESTRO ne dispose d'aucune solution automatisée pour la reconnaissance des partitions. Les tests se limitent à l'utilisation de fichiers MIDI pour essayer de faire jouer les instruments automatiquement, d'où la nécessité de notre solution.

Pour surmonter ces obstacles, deux approches s'offrent à nous: soit la réutilisation et l'amélioration d'un projet open-source existant, tel que cadencV, oemer, OMReader, Mozart, etc., soit développer une solution entièrement nouvelle. Bien que partir de zéro puisse offrir plus de flexibilité, cela risque d'être plus complexe et chronophage. Nous avons donc décidé de nous concentrer initialement sur un type de partition spécifique, en l'occurrence les partitions de piano à deux portées (une en clé de sol et une en clé de fa), afin de maximiser l'efficacité de notre développement et d'assurer une reconnaissance précise et fiable des partitions.

En ce qui concerne les technologies actuelles de reconnaissance optique de musique (OMR) présentent des limitations importantes pour répondre pleinement aux besoins spécifiques du projet MAESTRO. Plusieurs solutions open-source existent, mais aucune ne propose une solution complète et parfaitement adaptée aux exigences de notre projet.

OBJECTIFS DU PROJET

Dans ce projet, nous devons concevoir et implémenter une application qui permettra de créer un fichier MIDI à partir d'une partition musicale numérique ou manuscrite. Le fichier MIDI doit respecter au maximum la partition afin d'être le plus fidèle possible à la musique d'origine.

Objectifs principaux

Conception et implémentation d'une application dorsale :

- *Lecture de partition musicale* : Développer une application dorsale capable de lire des partitions musicales sous format PDF, en utilisant des techniques de reconnaissance optique de musique (OMR).
- *Génération de fichier MIDI* : Implémenter un système qui génère un (ou plusieurs) fichier MIDI à partir de la partition lue, garantissant la fidélité à la partition originale en termes de notes, rythmes et dynamiques.
- *Interface API* : Mettre en place une interface (API) permettant d'accéder à l'application dorsale pour envoyer des partitions et recevoir les fichiers MIDI générés. Cette interface API permettra de découpler l'interface utilisateur et l'algorithme de reconnaissance optique de musique afin d'offrir au club MusiquÉTS plus de flexibilité dans l'évolution de l'application.

Développement d'une Application frontale :

- *Interface utilisateur simple* : Concevoir et implémenter une application frontale simple permettant aux utilisateurs de téléverser des partitions (format PDF ou images) et de récupérer les fichiers MIDI générés par l'application dorsale.

Validation et Tests :

- *Séries de tests exhaustives* : Développer des séries de tests exhaustives pour valider la robustesse de l'application, vérifier la précision de la reconnaissance optique de musique et déterminer les limites de la solution proposée.

Documentations et guide d'utilisation :

- *Documentation technique* : Rédiger une documentation technique comprenant différents artefacts utiles pour la prise en main du système, tels que des diagrammes d'architecture, des diagrammes de classes et des diagrammes de séquence.
- *Guide utilisateur* : Produire un guide utilisateur détaillé expliquant comment utiliser l'application, incluant les forces et les limites de l'application, afin de faciliter l'adoption par les utilisateurs finaux.

Objectifs secondaires (non obligatoires)

Ces objectifs seraient intéressants à ajouter au projet, mais ne sont pas obligatoires dans le contexte limité en temps pour le projet.

Support de Multi-Instruments :

- *Génération de fichiers MIDI pour plusieurs instruments* : Étendre la capacité de l'application à générer des fichiers MIDI capables de jouer plusieurs instruments (au moins deux). **[Abandonné par manque de temps]**
- *Reconnaissance multi-instruments* : Améliorer la lecture de partitions pour reconnaître plusieurs instruments sur une même feuille, en identifiant correctement les différentes portées et leurs attributs. **[Abandonné par manque de temps]**

Amélioration de l'Interface utilisateur :

- *Interface utilisateur avancée* : Apporter des améliorations à l'interface utilisateur pour rendre l'application plus intuitive et agréable à utiliser, facilitant l'interaction avec le système pour les utilisateurs. Ajouter une fonctionnalité pour effectuer la lecture des fichiers MIDI générés. **[Abandonné par manque de temps]**

Conception et Implémentation d'une Base de Données :

- *Base de données pour stockage et gestion* : Concevoir et implémenter une base de données connectée à notre application dorsale pour stocker les partitions téléchargées, les fichiers MIDI générés, et éventuellement d'autres métadonnées pertinentes. **[Abandonné par manque de temps]**

Retombées du projet

Nous espérons que suite à la complétion de ce projet, plusieurs bénéfices tangibles et intangibles vont se manifester pour le club étudiant MusiquÉTS :

- *Automatisation de la collecte de partitions* : Le projet permettra au club de collecter et numériser des partitions musicales de manière automatisée, réduisant ainsi le besoin de saisie manuelle et de traitements fastidieux.
- *Base solide pour futurs développements* : La réalisation de ce projet fournira une base technologique solide que le club pourra étendre et améliorer à l'avenir. Cela ouvrira des possibilités pour des fonctionnalités supplémentaires et des améliorations continues.
- *Visibilité et attractivité du club* : La réussite de ce projet pourrait renforcer la visibilité du club MusiquÉTS au sein de l'École de technologie supérieure et pourrait attirer de nouveaux membres intéressés par la musique et les technologies de type OMR

MÉTHODOLOGIE

Comme la plupart des projets en informatique, des modifications et des changements seront demandés par le client au cours de ce projet. Nous allons utiliser la méthode Agile afin de capturer les demandes de changement le plus rapidement possible et ainsi nous adapter pour éviter tout retard potentiel.

Nous effectuerons des “sprints” de deux semaines. À l’exception des deux premiers sprints, chaque période de deux semaines respectera le schéma suivant :

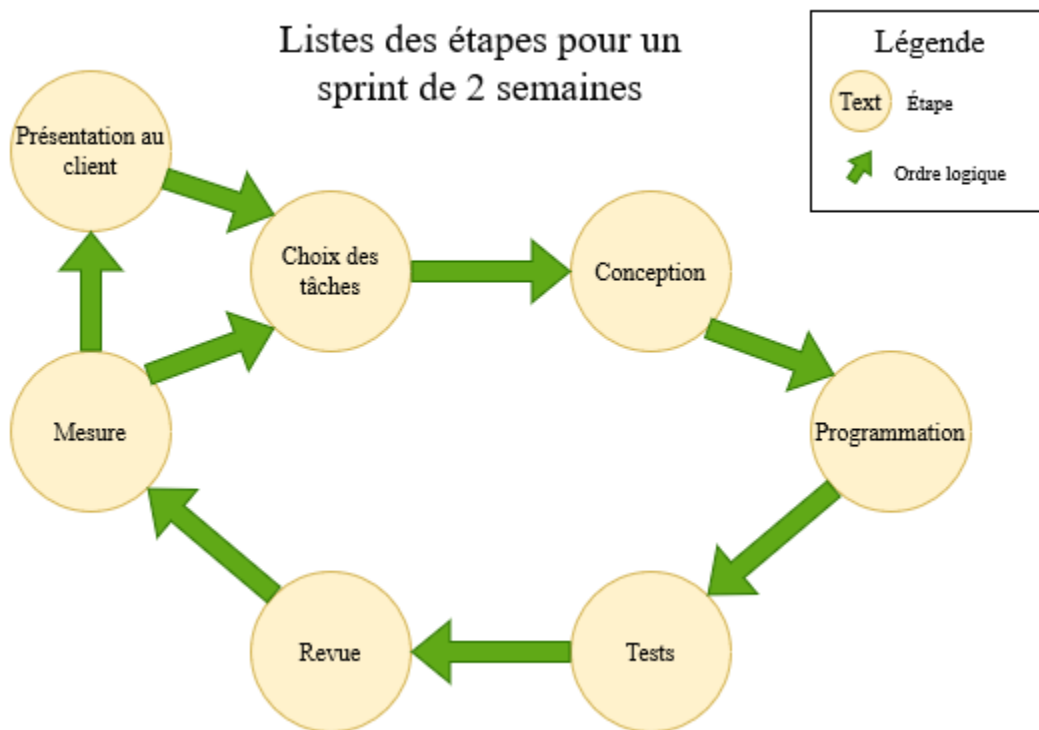


Figure 1 : Schématisation des étapes et tâches d'un sprint de deux semaines

- Choix des tâches** : Avec ou sans le client, l'équipe choisira les tâches sur lesquelles elle devra se concentrer lors du prochain sprint. Il sera aussi possible de créer de nouvelles tâches à la demande du client lors de cette étape.
- Conception** : L'équipe concevra la solution pour une tâche donnée. La plupart de la documentation technique sera produite lors de cette étape. La documentation sera amenée à être modifiée lors des étapes suivantes. C'est aussi à cette étape que les choix seront effectués.

3. **Programmation** : L'équipe implémentera les solutions trouvées lors de la phase de conception.
4. **Tests** : L'équipe implémentera la suite de tests suivant les directives du plan de tests pour les solutions programmées.
5. **Revu** : Les membres de l'équipe devront effectuer une revue du code des autres membres pour s'assurer qu'ils aient une compréhension complète du projet en plus de détecter les erreurs possibles. Cette phase permettra de s'assurer que le code respecte les plus hauts standards de qualité en termes de programmation.
6. **Mesure** : L'équipe effectuera les mesures pour vérifier et prouver que les solutions et les tests respectent les contraintes et les exigences. Toutes les mesures seront documentées.
7. **Présentation au client** : Au cours du projet, deux prototypes seront officiellement présentés au client, cependant à la demande de celui-ci, l'équipe pourra présenter l'état de l'application à la fin du sprint.

Puisque nous utilisons une méthode agile, les étapes précédentes pourront se faire en parallèle (ex : programmation et tests).

De plus, comme spécifiés plus haut, les deux premiers sprints ne respecteront pas le schéma. Étant au début du projet, des étapes préliminaires devront être faites proprement débiter le projet. Les étapes préliminaires sont :

- Rédaction du plan de projet
- Rédaction du document d'analyse des besoins
- Rédaction de l'analyse de risque
- Rédaction du plan de test
- Découpage des tâches

Il est à noter que la plupart des livrables seront modifiés et améliorés au cours du projet.

En temps normal, dans les principaux cadriceis agiles, il y a beaucoup de rencontres entre l'équipe (Stand-Up, Planification de sprint, Sprint review, post-mortem, etc). Puisqu'aucun des membres du projet ne travaille sur celui-ci à temps plein et qu'en plus il ne dure qu'environ 3 mois, nous avons décidé que durant les sprints, il y aurait deux rencontres officielles (planification de sprint et rencontre avec les professeurs), et des rencontres non officielles (rencontre entre les membres).

Tableau des échéanciers

Nom de la tâche ou du livrable	Début	Fin	Durée
Rédaction du plan de projet	15 mai 2024	22 mai 2024	7 jours
Rédaction du plan d'analyse des besoins	15 mai 2024	29 mai 2024	14 jours
Rédaction de l'analyse de risque	15 mai 2024	29 mai 2024	14 jours
Rédaction du plan de test	15 mai 2024	29 mai 2024	14 jours
Découpage des tâches	15 mai 2024	29 mai 2024	14 jours
Analyses et tests des solutions déjà existantes	22 mai 2024	29 mai 2024	7 jours
Rédaction compte rendu des analyses et tests des solutions existantes	22 mai 2024	29 mai 2024	7 jours
Planification du sprint #1	29 mai 2024	29 mai 2024	1 jour
Exécution du sprint #1	29 mai 2024	12 juin 2024	14 jours
Rétroaction (sprint #1)	12 juin 2024	12 juin 2024	1 jour
Planification du sprint #2	12 juin 2024	12 juin 2024	1 jour
Exécution du sprint #2	12 juin 2024	26 juin 2024	14 jours
Rétroaction (sprint #2)*	26 juin 2024	26 juin 2024	1 jour
Démonstration du prototype #1*	26 juin 2024	26 juin 2024	1 jour
Planification du sprint #3*	26 juin 2024	26 juin 2024	1 jour
Exécution du sprint #3	26 juin 2024	10 juillet 2024	14 jours

Rétroaction (sprint #3)	10 juillet 2024	10 juillet 2024	1 jour
Planification du sprint #4	10 juillet 2024	10 juillet 2024	1 jour
Exécution du sprint #4	10 juillet 2024	24 juillet 2024	14 jours
Rétroaction (sprint #4)	24 juillet 2024	24 juillet 2024	1 jour
Démonstration du prototype #2	24 juillet 2024	24 juillet 2024	1 jour
Planification du sprint #5	24 juillet 2024	24 juillet 2024	1 jour
Exécution du sprint #5	24 juillet 2024	7 août 2024	14 jours
Rétroaction (sprint #5)	7 août 2024	7 août 2024	1 jour
Démonstration finale	7 août 2024	7 août 2024	1 jour
Fin du projet, remise du projet	8 août 2024	8 août 2024	1 jour

Tableau 1 : Échéanciers

Composition de l'équipe et rôle

Prénom	Rôle(s) (voir annexe A)	Responsabilités
1. Philippe Langevin	Chef d'équipe/Développeur	Rédige les documents administratifs, organise les réunions, prise de notes pendant les réunions Rédige la documentation en lien avec le rapport Écris le code
2. Charlie Poncsak	Développeur	Rédige la documentation en lien avec le rapport Écris le code
3. Laflèche Chevrette	Animateur/Développeur	Anime les rencontres Rédige la documentation en lien avec le rapport Écris le code
4. Jérémy Da Costa	Développeur	Rédige la documentation en lien avec le rapport Écris le code
5. Xavier Jeanson	Développeur	Rédige la documentation en lien avec le rapport Écris le code

Tableau 2 : Composition de l'équipe

Retour sur les rencontres suite au projet

Au cours des treize rencontres hebdomadaires de notre équipe, qui se sont échelonnées du 15 mai 2024 jusqu'à la dernière démonstration du 31 juillet 2024, nous avons travaillé de manière intensive pour développer une solution robuste de reconnaissance musicale automatisée. Dès le départ, nous avons défini des objectifs clairs, axés sur la conversion précise des partitions musicales en fichiers MIDI, en utilisant Audiveris comme base technologique. Au fil des semaines, nous avons adapté et perfectionné notre approche, en tenant compte des retours constants de nos clients pour répondre au mieux à leurs besoins.

Lors des premières réunions, nous avons posé les fondations du projet, en nous concentrant sur l'intégration d'Audiveris et en explorant les défis liés à la reconnaissance des partitions manuscrites et numériques. Nos discussions ont souvent tourné autour des limitations techniques, comme la difficulté d'Audiveris à gérer certains termes de tempo ou la conversion des fichiers en formats MIDI valides. Nous avons travaillé de manière itérative, en améliorant continuellement notre prototype, tout en veillant à ce que chaque version soit testée rigoureusement.

Les démonstrations ont joué un rôle crucial dans notre processus de développement. Chaque présentation a servi de point de référence pour évaluer notre progression et recueillir des commentaires précieux de la part du client. Par exemple, lors de la démonstration du 28 juin, nous avons présenté un premier prototype qui, bien qu'efficace, a révélé des limitations dans la conversion des dynamiques ou du tempo en MIDI. Le client a apprécié notre travail, mais a souligné l'importance de garantir la précision du tempo et des vélocités manquantes. Cela nous a conduits à collaborer plus étroitement pour affiner les configurations et améliorer la fiabilité de la conversion.

Lors de la dernière démonstration du 31 juillet, nous avons présenté une version quasiment finale du projet. Nous avons alors intégré des fonctionnalités supplémentaires demandées par le client, comme une notification indiquant l'absence de tempo détecté, et avons documenté les limitations de notre solution dans le rapport final ainsi que sur GitHub. Les retours ont été globalement positifs, avec des suggestions pour rendre l'interface plus intuitive et pour fournir un retour d'information plus clair aux utilisateurs.

En somme, cette série de rencontres et de démonstrations a non seulement permis de développer une solution technique solide, mais a également renforcé notre capacité à répondre de manière agile et réactive aux attentes du client, tout en laissant un projet bien documenté et prêt à être transmis pour une future évolution.

LIVRABLES ET PLANIFICATION

Description des artefacts

Nom de l'artefact	Description
Plan de projet	Document décrivant le plan et la direction du projet dans son ensemble.
Document d'analyse des besoins	Document de vision qui englobe les contraintes et les exigences du client, fonctionnelles, non fonctionnelles et autres.
Analyse de risque	Document permettant d'analyser les risques possibles du projet ainsi que les solutions à chacun des risques
Plan de tests	Document planifiant les tests utilisés lors du développement.
Prototype 1	Le premier prototype qui sera présenté au client
Prototype 2	Le second prototype qui sera présenté au client
Prototype final	L'application finale
Rapport final	Le rapport final ne sert qu'à l'équipe dans le cadre du PFE

Tableau 3 : Artefacts

Planification

Voir Annexe A

RISQUES

L'échelle d'impact et de probabilité est décomposée sur trois tiers: Faible - Moyen - Fort

Risque	Impact	Probabilité	Mitigation / atténuation
Technologie nécessaire trop récente ou jeune	Moyen	Moyen	Effectuer une analyse préliminaire des solutions existantes.
Technologies existantes ne permettant pas d'atteindre le niveau de robustesse voulu	Fort	Moyen	Chercher des papiers de recherche sur l'OMR, plus précisément sur la robustesse des modèles. Tester en profondeur les prototypes.
Difficulté à trouver des données d'entraînement ou de test	Fort	Faible	Prendre note des données utilisées pour les solutions existantes.
Difficulté à tester les prototypes	Moyen	Faible	Prendre note des tests effectués pour les solutions existantes.
Manque de ressources (base de données, serveurs, etc.)	Faible	Fort	Mettre en place les attentes et exigences du client quant au livrable final à remettre.
Projets existants difficiles à modifier	Fort	Faible	Observer l'architecture et la conception des solutions existantes lors de leur analyse.
Équipe démotivée ou pas à son affaire	Moyen	Faible	Bien définir les rôles des membres. Bien définir la séparation des tâches et en faire un suivi régulier et ouvert aux changements.

Tableau 4 : Risques

TECHNIQUES ET OUTILS

Pour le développement du projet, nos choix d'outils, de technologies et de techniques ont été guidés par plusieurs facteurs clés : la durée limitée du projet (4 mois), les compétences et points faibles des membres de l'équipe, ainsi que les besoins et critères spécifiques du client. Nous avons sélectionné des outils et technologies qui nous permettent de travailler efficacement, de manière collaborative et qui répondent aux exigences techniques du projet.

En plus de cela, nous avons effectué plusieurs tests de technologies déjà existantes pour évaluer leurs capacités et leur adéquation avec les exigences du projet. Nous avons construit une matrice décisionnelle pour comparer ces technologies sur divers critères importants pour le client et le projet. Ce processus nous a conduits à choisir Audiveris comme base technologique pour notre solution de reconnaissance optique de musique.

Il est à noter que ces choix technologiques peuvent évoluer au cours du projet ou après la présentation de notre premier prototype. Certaines technologies, comme les bibliothèques de développement spécifiques, ne sont pas mentionnées ici, car leur impact est minime en comparaison des outils principaux.

Outils et techniques de gestion de projet

- *Gestion de projet agile* : Méthodologie de gestion de projet qui privilégie l'adaptation aux changements, les livrables fréquents et la collaboration étroite avec le client. Nous l'utiliserons pour mieux nous adapter aux exigences changeantes et livrer des résultats de qualité régulièrement.
- *Github* : Une plateforme de développement collaboratif, permettant de gérer le code source, suivre les modifications, et collaborer efficacement avec l'équipe. Il sera utilisé pour héberger le code du projet et gérer les contributions.
- *Github projects* : Un outil de gestion de projet intégré à GitHub, facilitant la planification, le suivi des tâches et la gestion des flux de travail. Il nous aidera à organiser et prioriser les tâches de développement de manière structurée.
- *Google Drive* : Une solution de stockage en ligne et de partage de fichiers qui nous permettra de centraliser et partager facilement les documents, rapports et autres ressources du projet avec l'équipe.

Outils de communication

- *Discord* : Une plateforme de communication vocale et textuelle. Nous l'utiliserons pour les discussions informelles, les sessions de travail collaboratif et les échanges rapides entre les membres de l'équipe, les professeurs et le client.
- *Microsoft Teams* : Outil de collaboration qui intègre des fonctionnalités de messagerie, de visioconférence et de partage de fichiers. Il sera utilisé pour les réunions formelles, les présentations de projet et la communication avec les professeurs, le client et toutes autres parties prenantes extérieures.
- *Outlook* : Application de messagerie électronique qui nous permettra de gérer les courriels et les calendriers. Elle sera utilisée pour la communication officielle et la gestion des rendez-vous et échéances.

Technologies et outils de développement.

- *Java* : Langage de programmation orienté objet robuste et portable, choisi pour le développement de l'algorithme de reconnaissance optique de musique.
- *Spring boot* : Cadriceiel choisi pour faire la gestion d'appel API pour l'application dorsale (*back-end*). Le cadriceiel en langage Java fera aussi la gestion d'appels avec le programme Audiveris qui sera utilisé.
- *Audiveris* : Logiciel open-source de reconnaissance optique de musique qui sera modifié, adapté et amélioré pour répondre aux besoins spécifiques du projet. L'engin OMR d'Audiveris fournira la base technologique pour la conversion des partitions en fichiers MIDI.
- *Git* : Système de contrôle de version décentralisé qui permettra de suivre les modifications du code source, de gérer les branches de développement et de faciliter la collaboration entre les membres de notre équipe. Git assurera la gestion efficace du code tout au long du projet.
- *IDE (Environnement de Développement Intégré)* : Logiciel qui fournit des outils complets pour l'édition, la compilation, le débogage et le test du code. Nous utiliserons des IDEs pour le développement de l'application dorsale et frontale. L'IDE final n'est pas choisi.
- *JavaScript* : Un langage de "scripting" utilisé pour programmer les éléments interactifs d'une page Web.
- *NPM* : Un gestionnaire de paquet pour JavaScript utilisé pour installer les bibliothèques utilisées ainsi que leurs dépendances.
- *Node.js* : Une application multiplateforme permettant de développer, de tester et d'exécuter des environnements de développement et de production pour le langage de programmation JavaScript.
- *React* : Une bibliothèque JavaScript pour la construction d'interfaces utilisateur, utilisée pour développer l'application frontale. React sera notre technologie frontale principale. Elle

permettra de créer une interface utilisateur réactive et dynamique pour téléverser les partitions et récupérer les fichiers MIDI générés

- *music21*: La librairie Python *music21* est un outil puissant développé par le MIT (licence BSD (3-clause)), utilisé pour analyser, manipuler et convertir des données musicales. Dans notre application, *music21* joue un rôle crucial en permettant la conversion des fichiers MXL reçus d'Audiveris en fichiers MIDI. *Music21* offre une panoplie de fonctions simples à utiliser pour lire et écrire en différents formats de musique, facilitant ainsi la transformation des données MusicXML en MIDI pour notre application. La librairie permet de manipuler les éléments musicaux, tels que les notes, les dynamiques et les tempos, garantissant une conversion précise et fidèle des partitions originales. Si d'autres modifications devaient être apportées aux partitions suite à la conversion d'Audiveris, *music21* a probablement les outils nécessaires pour ce faire.
- *Python* : Un langage de programmation orienté objet, interprété et multiplateforme puissant avec une communauté active. Le langage Python est principalement utilisé pour la génération des fichiers MIDI à partir des fichiers MXL.
- *Docker* : Une application puissante permettant de déployer rapidement de petites machines virtuelles, nommées conteneurs, peu coûteuses en ressources. Nous utilisons ces conteneurs pour déployer notre application. Puisque les images Docker sont basées sur le système d'exploitation Linux, l'utilisation de langages multiplateforme comme Java et Python est importante.
- *Junit* : Une bibliothèque Java permettant de faciliter la programmation de tests unitaires pour notre application en implémentant des fonctionnalités pour gérer les cas de tests et les suites de tests.
- *Mockito* : Une bibliothèque Java permettant d'implémenter le principe de "mock" dans nos tests unitaires. Un "mock" est un objet permettant de simuler des classes afin de vérifier que le logiciel fonctionne sans lire ou créer des ressources externes qui peuvent être lourdes ou complexes.
- *Unittest* : Une bibliothèque de tests pour Python inspirée par Junit. Elle implémente la plupart des fonctionnalités de tests unitaires connues comme les cas de tests, les suites de tests, les "mocks", etc. Cette bibliothèque est utilisée pour tester notre script Python.
- *Gradle* : Un outil de moteur de production permettant de construire, de compiler, de tester et de gérer des projets Java de façon simple et scriptable.

OBJECTIFS DE RÉUSSITE

Voici une liste d'objectifs que nous devons atteindre dans le projet afin de considérer celui-ci comme réalisé :

- Les limites de l'application sont claires, précises, testées et documentées.
- Chaque fonctionnalité est accompagnée de documentation (architecture, use case, etc.)
- L'application dorsale fonctionne indépendamment. Elle doit être capable de traiter les entrées et de produire les sorties sans dépendance externes.
- Le logiciel détecte et reproduit correctement :
 - Clés (Sol et Fa)
 - Notes (Do, ré, mi, fa, sol, la, si)
 - Accidents (dièse, bémol, bécarre)
 - Silences (pause, demi-pause, soupir, demi-soupir, quart de soupir)
 - Figures de note (ronde, blanche, noire, croche, double croche)
 - Armure
 - Indication de mesure
 - Articulations (Staccato, accents)
 - Barres de mesure
- Le logiciel s'exécute dans un temps raisonnable, c'est-à-dire en moins d'une heure pour les plus grosses partitions.
- Chaque fonctionnalité est testée par des tests unitaires et des tests d'intégrations.

CONTRAINTES

Contraintes fonctionnelles

- **Reconnaissance des notes** : Le logiciel doit détecter et identifier les notes d'une partition musicale pour la clé de sol et de fa.
- **Détection des accidents** : Le logiciel doit détecter et identifier les accidents dans une mesure.
- **Reconnaissance des silences** : Le logiciel doit détecter et identifier les silences.
- **Identification des figures de notes** : Le logiciel doit détecter et identifier les figures de notes.
- **Détection des clés** : Le logiciel doit détecter la clé de sol et la clé de Fa en plus d'adapter les notes en conséquence de celles-ci.
- **Lecture de l'armure** : Le logiciel doit être capable de lire l'armure et d'appliquer les accidents provenant de celle-ci.
- **Reconnaissance des indications de mesure** : Le logiciel doit détecter et comprendre les indications de mesure.
- **Identification des articulations** : Le logiciel doit détecter, identifier jouer certaines articulations (staccato et accents).
- **Détection des barres de mesure** : Le logiciel doit être capable de détecter et comprendre les barres de mesure, notamment lorsqu'il s'agit de la barre de reprise.
- **Utilisation du GUI** : L'interface graphique doit pouvoir servir de guide simplifié de l'utilisation de l'API du backend. Si une route existe dans l'application dorsale, elle doit être accessible sur l'application frontale.
- **Lecture des fichiers PDF** : Le logiciel doit être capable de lire les partitions sous le format PDF. Les partitions peuvent être une image ou une partition numérique.
- **Création d'un fichier MIDI** : Le logiciel doit produire un fichier MIDI à partir de la partition reçue en entrée. Cela ne l'empêche pas de pouvoir produire des sorties intermédiaires (ex : musicxml).

Contraintes non fonctionnelles

Aptitude fonctionnelle

Exhaustivité fonctionnelle

- Le système doit être capable de lire des partitions musicales sous format PDF, détecter les notes, les silences, les clés, les armures, et autres symboles musicaux, et générer un fichier MIDI conforme à la partition d'origine.

Exactitude fonctionnelle

- La reconnaissance des éléments musicaux doit être précise, avec un taux d'exactitude respectant le taux de réussite établi.

Pertinence fonctionnelle

- Le système doit produire des fichiers MIDI qui respectent fidèlement la partition musicale originale en termes de notes, rythmes et dynamiques.

Efficacité de performance

Utilisation des ressources

- L'application doit utiliser les ressources système de manière efficiente, sans surcharger la mémoire ou le processeur pour éviter des pannes potentielles du système.

Capacité

- Le système doit pouvoir traiter des partitions musicales de différentes tailles et complexités.

Compatibilité

Coexistence

- L'application doit pouvoir fonctionner en parallèle avec d'autres applications sur le même système sans conflit.

Utilisabilité

Reconnaissance de l'adéquation

- L'interface utilisateur doit être intuitive, permettant aux utilisateurs de comprendre facilement comment utiliser l'application.

Apprentissage

- Le système doit être facile à apprendre, avec une courbe d'apprentissage minimale pour les nouveaux utilisateurs du club MusiquÉTS.

Exploitabilité

- L'interface utilisateur doit permettre une interaction facile et efficace avec le système.

Fiabilité

Maturité

- Le système doit être stable et fonctionner correctement dans des conditions normales d'utilisation.

Tolérance aux pannes

- Le système doit être capable de gérer les erreurs et les défaillances de manière gracieuse, sans perte de données.

Maintenabilité

Modularité

- Le système doit être conçu de manière modulaire pour faciliter les mises à jour et les améliorations futures voulant être apportées par le club une fois le projet remis.

Analysabilité

- Le code source doit être bien documenté pour permettre une analyse facile et rapide.

Modifiabilité

- Le système doit être facilement modifiable pour permettre des ajustements et des améliorations.

Testabilité

- Le système doit être conçu de manière à faciliter les tests unitaires et d'intégration

TESTS

Applications Maestro

Les tests de l'application Maestro se trouvent dans le dossier :

backend/src/test/java/PFE008/backend

Il existe une suite de tests par fichier .java de l'application dorsale. Les tests pour le script Python s'y trouvent aussi. Lorsque les tests doivent utiliser des ressources externes comme des fichiers, ceux-ci se trouvent dans :

backend/src/test/java/PFE008/backend/resources/tests_java

Les tests sont exécutés automatiquement par Gradle lorsque l'application est compilée. Cependant, il est aussi possible d'exécuter seulement les tests avec la commande :

gradle test

Les tests Python sont aussi exécutés par la commande Gradle.

Toutes les configurations liées aux tests se trouvent dans le fichier :

backend/build.gradle

```
tasks.named('test') {
    useJUnitPlatform()
    testLogging {
        events "passed", "skipped", "failed"
        showStandardStreams = false // True pour déboguer
        exceptionFormat = 'full'
    }
    dependsOn runPythonTests
}

task runPythonTests(type: Exec) {
    commandLine 'python', 'src/test/java/PFE008/backend/MxlToMidi_tests.py'
}
```

Figure 2 : Configuration des tests dans Gradle

Pour simplifier le débogage des tests, il est possible de définir « showStandardStreams » à « true ». Par défaut, nous avons choisi de désactiver cette option puisque ça cause du bruit inutile lors du fonctionnement normal.

Application Audiveris

Afin de déterminer les limites du logiciel Audiveris en termes de reconnaissance des partitions musicales, nous avons effectué une série de tests manuelle.

Afin d'effectuer les tests, nous avons créé une série de 10 partitions testant chacune un ou plusieurs éléments musicaux

1. Les accidents
2. Les figures de note et les silences
3. Les clés
4. Les armures et les indications de mesure
5. Les indications de tempo et les nuances
6. Les hauteurs
7. Les articulations
8. Les barres de mesure et de reprises et les signes de reprise (Al fine, Al Coda, etc)
9. Les éléments autres (accords, ghost note, etc)
10. Les notes

Nous avons aussi plusieurs partitions musicales offertes par le club MusiquÉTS ou téléchargées à partir de Free-scores.com afin de tester Audiveris pour simuler des cas d'utilisation réels et possibles.

Chacun des fichiers utilisés se trouve dans :

backend/src/test/java/PFE0008/backend/resources/manual_tests

Nous avons utilisé l'application libre de droit MuseScore Studio 4 pour créer nos partitions de tests.

DÉPLOIEMENT

Déployer l'application en local

Afin de faciliter le déploiement de l'application, nous avons décidé d'utiliser la technologie de conteneurisation Docker.

Notre conteneur se base sur la dernière version d'Ubuntu. Cette image ne contient que le strict minimum afin de ne pas alourdir inutilement l'image Docker. Présentement, l'image Docker a une taille de 5,35 GB et 2,22 Gb lorsqu'elle est compressée.

Pour ne pas alourdir inutilement le conteneur dans l'image, tous les fichiers reliés aux tests tels que les séries de tests et leurs ressources ne sont pas copiés dans le conteneur. Si jamais vous utilisez une version locale d'Audiveris, celle-ci ne sera pas non plus copiée afin de garder l'image intègre. Il a aussi été décidé de compiler l'application lorsque l'image docker est compilée afin que son fonctionnement et sa maintenance soit le plus simple possible.

Le point d'entrée de l'image est le script "entrypoint.sh". Celui exécuté

Il permet d'exécuter simultanément le backend et le frontend avec les commandes :

```
java -jar backend/build/libs/backend-0.0.1-SNAPSHOT.jar &  
npm --prefix maestro-app/ run start
```

Afin d'exécuter l'image Docker sur un serveur grâce, il faut commencer par avoir l'image locale. Il est possible de compiler l'image avec la commande :

```
docker build -t musiquets/maestro:<tag> .
```

Cela dure environ 5 minutes pendant lesquelles l'image installe toutes les dépendances dont elle a besoin.

Il est aussi possible d'obtenir l'image via [Docker Hub](#) officiel de MusiquÉTS avec la commande :

```
docker pull musiquets/maestro
```

Pour démarrer le conteneur, il faut utiliser la commande :

```
docker run -p <port1>:3000 -p <port2>:8080 musiquets/maestro
```

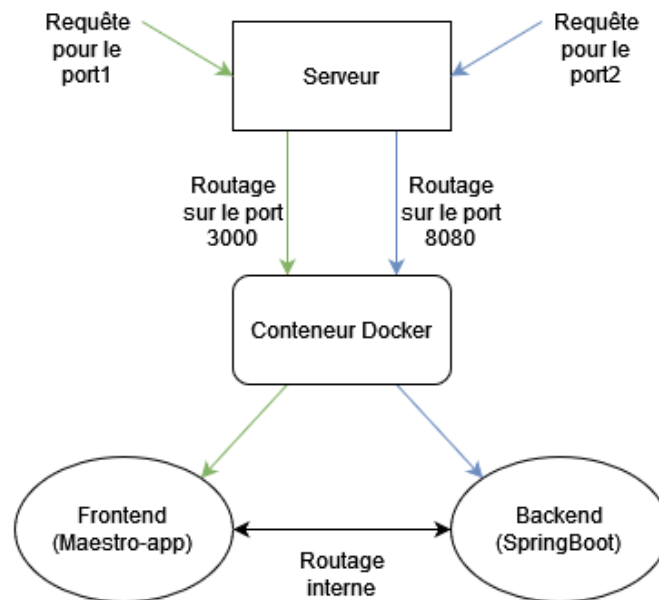


Figure 3 : Fonctionnement de l'image Docker

Notre application frontend a besoin du port 3000 et notre application backend du port 8080 pour fonctionner correctement. Pour ne pas avoir à modifier la configuration des deux applications, il est possible de mapper les ports du serveur hôtes avec les ports utilisés par l'image Docker. Ainsi, les administrateurs pourront choisir les deux ports sans avoir à se soucier de ce que Maestro utilise réellement.

Déployer l'image dans le Docker hub

Si vous avez effectué des modifications au code et que vous souhaitez déployer une nouvelle version de l'image dans le hub officiel de MusiquÉTS, il faut effectuer les commandes suivantes :

```
docker login
```

```
docker build -t musiquets/maestro:<tag> .
```

```
docker push musiquets/maestro:<tag>
```

Il est important de noter que si vous utilisez un tag déjà existant, l'image sera alors écrasée par la dernière image poussée, c'est-à-dire la plus récente.

FONCTIONNEMENT

L'application de conversion de partitions en fichiers MIDI permet aux utilisateurs de transformer des partitions en PDF ou en image en fichiers MIDI exploitables, en passant par plusieurs étapes de traitement et de reconnaissance. Voici une explication détaillée du processus de fonctionnement de l'application.

1. Sélection de la Partition

L'utilisateur accède à la page frontale de l'application, où il peut sélectionner une partition au format PDF ou image supportée. Sur cette page, l'utilisateur dispose de plusieurs options :

- *Changer le nom de la partition* : par défaut, le nom du fichier PDF sera utilisé comme nom pour le fichier MIDI retourné, sauf si l'utilisateur choisit de le modifier.
- *Ajouter des tempos en BPM à certaines mesures* : cette fonctionnalité permet de spécifier des tempos personnalisés pour différentes sections de la partition. Toutefois, cette option présente certaines limitations, comme mentionnées dans la section sur les problèmes connus.

2. Transmission et Traitement par l'Application Dorsale

Une fois que l'utilisateur a configuré ses options et activé la conversion, le fichier est transmis à l'application dorsale avec les paramètres sélectionnés. L'application dorsale initie alors un processus qui transmet la partition au modèle de reconnaissance Audiveris. Audiveris est chargé de la reconnaissance des éléments de la partition, incluant les notes, les dynamiques et les tempos.

3. Analyse et Exportation par Audiveris

Après avoir analysé la partition, Audiveris exporte un fichier MXL (MusicXML) contenant la partition convertie. Ce format est utilisé pour sa flexibilité et sa compatibilité avec diverses applications de traitement de musique.

4. Conversion du Fichier MXL en MIDI

Le client ayant demandé une sortie au format MIDI, l'application dorsale utilise un script Python pour convertir le fichier MXL en MIDI. Ce script s'appuie sur la librairie music21 du MIT pour effectuer cette conversion. Le processus suit les étapes suivantes :

- Le fichier MXL est reconverti en XML, puis transformé en fichier MIDI.
- Les dynamiques musicales (pianissimo, mezzo piano, forte, etc.), qui ne sont pas automatiquement converties, sont sauvegardées en mémoire lors de la conversion initiale et réappliquées au fichier MIDI.

4.1 Application des Tempos

Le script Python analyse ensuite les tempos présents dans le fichier MXL original et les applique au fichier MIDI si ces informations existent. Si l'utilisateur avait spécifié des tempos particuliers lors de l'envoi du fichier, ces options sont également intégrées au fichier MIDI à ce stade.

5. Retour du Fichier MIDI à l'Application frontale

Enfin, le fichier MIDI généré est retourné à l'application frontale. L'utilisateur peut alors télécharger le fichier MIDI finalisé depuis la page web.

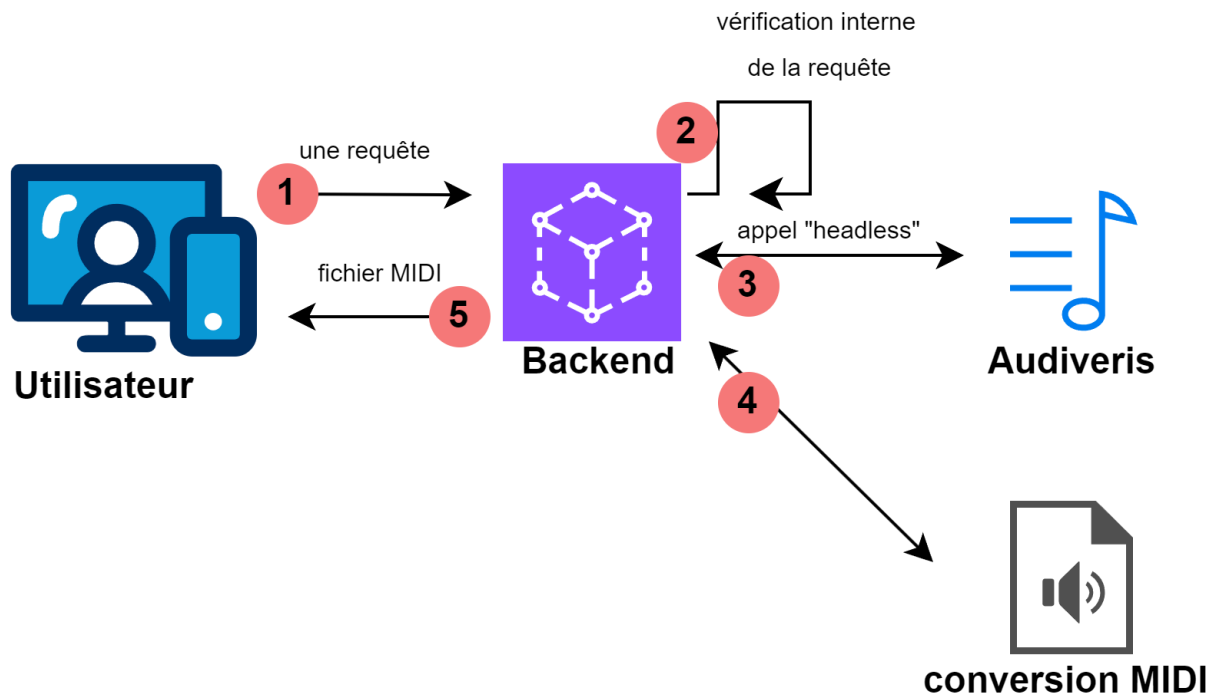


Figure 4 : Fonctionnement de l'application Maestro

PROBLÈMES CONNUS

Tempos

Tempos irréguliers et uniques

L'application présente un problème notable lorsqu'elle traite des partitions de musique comportant des tempos et les humeurs textuelles irrégulières ou uniques (voir la figure ci-dessous), donc autre que ceux qui par exemple annoncent leur tempo comme $J = 90$. Actuellement, l'application dispose d'une table de termes de tempo contenant environ 100 entrées, couvrant les langues les plus couramment utilisées dans les partitions occidentales : français, anglais, allemand et italien. Cette table se trouve dans le fichier *AudiverisController* et est utilisée par la fonction *applyTempo* pour déterminer les tempos des partitions.

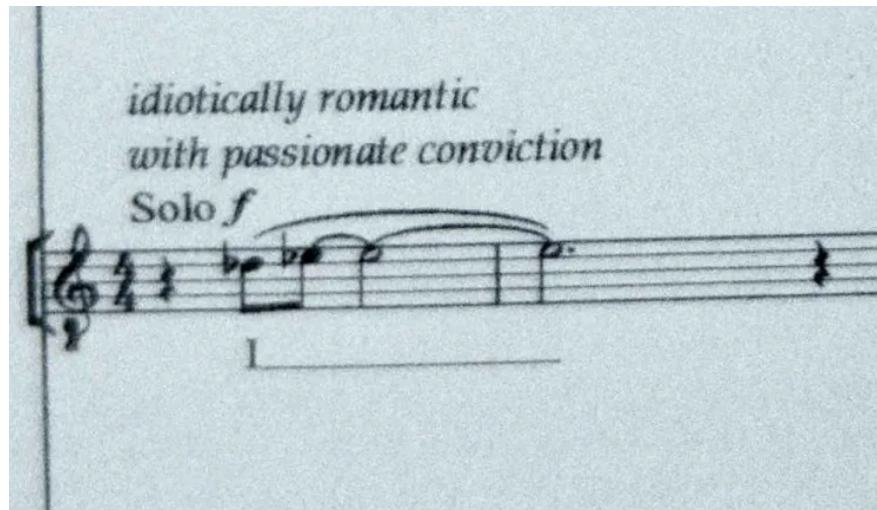


Figure 5 : Exemple d'un tempo et humeur textuels uniques

Cependant, lorsque l'utilisateur importe une partition avec des termes de tempo qui ne figurent pas dans cette table, ou dans une langue différente de celles supportées, l'application attribue automatiquement un tempo par défaut de 120 BPM dans le fichier MIDI généré. Cela signifie que pour obtenir le tempo correct, l'utilisateur doit utiliser l'outil UI de l'application frontale (le site web) pour ajuster manuellement le tempo dans la partition MIDI, ce qui peut s'avérer fastidieux et chronophage.

Pour améliorer cette situation, il est possible d'élargir la table de termes de tempo en y ajoutant de nouveaux termes fréquemment rencontrés. Cette tâche peut être effectuée par l'équipe de développement ou par le club musiquÉTS, qui prendra en charge le développement futur de l'application. Une solution potentielle serait d'externaliser cette table dans un fichier séparé, facilitant ainsi les mises à jour et les modifications sans avoir à modifier directement le code source de l'application.

En outre, il est crucial de s'assurer que le *language pack* utilisé par Audiveris, qui repose sur les packs de langues de Tesseract, soit installé pour toutes les langues pertinentes concernant des tempos textuels que l'on veut ajouter. Cela garantirait le bon fonctionnement de l'application dans des langues actuellement non testées et potentiellement non supportées. L'ajout de ces packs de langues

permettrait une reconnaissance plus précise des termes de tempo et réduirait la dépendance à la table de termes intégrée.

Symboles de tempo mal annotée

Lors de la conversion d'une partition, le tempo peut être détecté par la notation suivante : $J = 90$. Cependant, nous avons rencontré un problème qui affecte la majorité des partitions convertie par Audiveris. Lors de l'analyse du fichier, Audiveris note le tempo en changeant le symbole de clé par le caractère J. De plus, il ne l'écrit pas à la place affectée au tempo dans le fichier de sortie musicXML.

Pour contourner ce problème qui n'est pas réglable sans modifier l'algorithme du moteur de classement de Audiveris. Nous avons créé une solution de traitement secondaire afin de récupérer le tempo détecté et l'affecter aux bonnes mesures. Cette solution prend avantage que Audiveris écrit tous les mots qu'il détecte en notant l'emplacement de la détection. Nous avons alors pu trouver la zone normalement attribuée au tempo afin de récupérer celui détecté.

Cependant, cette solution n'est pas sans failles. Afin d'améliorer la détection du tempo, il faudrait rentrer dans l'algorithme de l'intelligence artificielle de Audiveris. Ainsi, il serait possible de modifier la façon de noter le tempo dans le fichier de sortie.

Symboles de tempo non reconnus

Un autre problème fréquent rencontré lors de la transformation des partitions en fichiers MIDI est lié à la taille des symboles de tempo, tels que $J = 90$. Lorsque ces symboles sont trop petits ou mal positionnés sur la partition, Audiveris peut échouer à les reconnaître correctement. En conséquence, le logiciel ne parvient pas à extraire le tempo réel de la partition, ce qui conduit à l'attribution automatique du tempo par défaut de 120 BPM dans le fichier MIDI généré.

Ce défaut de reconnaissance du tempo peut résulter de plusieurs facteurs, notamment la qualité de l'image de la partition, la clarté des symboles de tempo, ou encore des limitations inhérentes à l'algorithme de reconnaissance optique de caractères utilisé par Audiveris. Pour mitiger ce problème, il serait bénéfique d'améliorer le prétraitement des images de partitions pour s'assurer que les symboles de tempo soient suffisamment visibles et contrastés. De plus, l'affinement des paramètres de reconnaissance d'Audiveris pour mieux détecter les symboles de petite taille pourrait significativement améliorer la précision de la conversion des tempos. Enfin, des vérifications manuelles et des ajustements via l'outil UI de l'application restent nécessaires pour garantir que le tempo correct soit appliqué au fichier MIDI, surtout dans les cas où les symboles de tempo sont peu lisibles.

Outil UI de modification de tempo non intuitif

Il s'agit plus d'une suggestion d'amélioration qu'une solution à un vrai problème pour l'outil de tempo de l'interface utilisateur. Cependant, en adoptant une syntaxe plus intuitive et familière semblable à celle utilisée dans les logiciels d'édition musicale, la compréhension et l'utilisation par les utilisateurs seraient grandement facilitées. Cette modification de l'interface contribuerait à une expérience utilisateur plus fluide et intuitive, permettant aux utilisateurs de corriger et d'ajuster les tempos plus efficacement.

Partitions avec paroles

Un autre problème existant concerne les partitions contenant des paroles de chant. Lorsque ces partitions présentent plusieurs lignes de texte superposées (voir la figure ci-dessous), Audiveris peut commencer à interpréter ces lignes de texte comme des lignes de partitions, et les caractères du texte comme des notes. Cette confusion peut entraîner l'ajout de fausses notes dans les fichiers MIDI générés après la conversion.

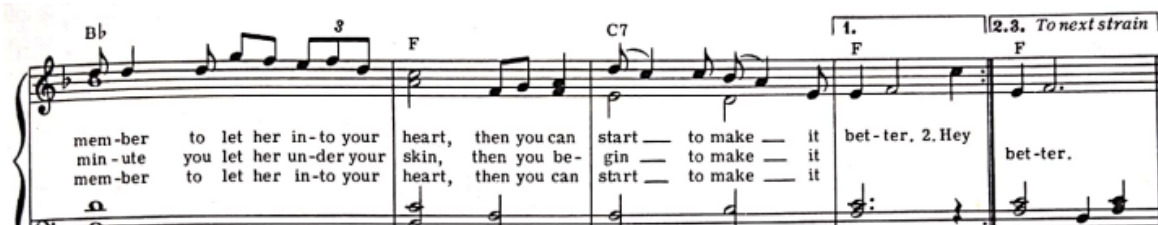


Figure 6 : Partition de *Hey Jude* avec paroles sous la portée

Ce problème survient principalement parce que, même si Audiveris parvient à identifier le texte dans des balises "lyric" ou "text", ces balises ne sont pas bien supportées par le protocole MIDI. En conséquence, le texte est parfois interprété à tort comme des éléments musicaux, sur le même instrument, ici l'instrument "voix", perturbant ainsi la précision de la partition convertie.

Actuellement, la capacité d'Audiveris à détecter et à ignorer automatiquement le texte lors de l'exportation après l'analyse n'est pas suffisamment développée. Cela signifie que les paroles et autres textes présents sur la partition peuvent encore être mal interprétés et affecter négativement la conversion en MIDI. Une solution temporaire, mais peu élégante consiste à effacer ou à cacher le texte problématique sur la partition avant de procéder à l'analyse. Cependant, cette approche n'est pas idéale, car elle modifie la partition originale et peut être laborieuse.

Pour une solution plus durable, il serait nécessaire de développer des algorithmes plus sophistiqués au sein d'Audiveris pour améliorer la détection et la gestion du texte. Cela pourrait inclure une meilleure différenciation entre le texte des paroles et les notes de musique, ainsi qu'une capacité accrue à filtrer automatiquement le texte lors de l'exportation. Jusqu'à ce que ces améliorations soient mises en œuvre, il est important de vérifier manuellement les fichiers MIDI générés et de corriger toute note incorrecte due à une mauvaise interprétation des paroles.

Partitions manuscrites

Un des défis rencontrés par l'application concerne la reconnaissance des partitions manuscrites. Audiveris est capable de reconnaître partiellement les partitions faites à la main. Cependant, la grande variation dans la manière dont les différents composants musicaux sont dessinés entraîne des résultats inconsistants. Ces résultats ne reflètent souvent pas fidèlement la partition originale. La variabilité des écritures manuscrites, notamment dans la forme des notes, des clés et des symboles dynamiques, complique la tâche de reconnaissance automatique.

L'équipe de développement a tenté de surmonter ce défi en travaillant sur son propre modèle de reconnaissance à intégrer avec l'outil Audiveris. Cependant, plusieurs problèmes persistent. Les notes sont reconnues par une méthode de correspondance de motifs (pattern matching), tandis que les symboles tels que les clés et les dynamiques sont identifiées par des algorithmes d'apprentissage machine. Cette approche hybride pose des difficultés, car la précision de la reconnaissance varie en fonction de la qualité et du style de l'écriture manuscrite.

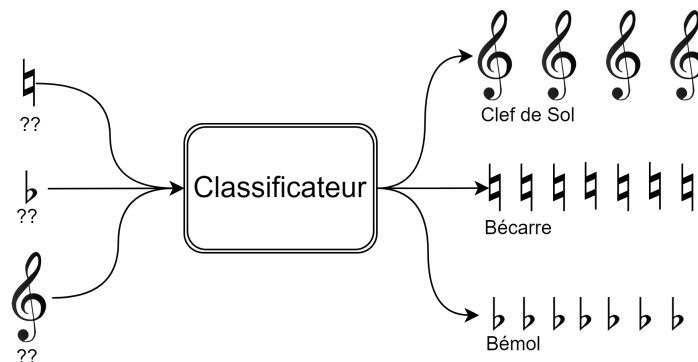


Figure 7 : Fonctionnement du classificateur IA de symboles musicaux

La première tentative pour faire fonctionner la reconnaissance de partitions manuscrites a été de créer un nouveau modèle d'IA en utilisant les technologies *Tensorflow*⁶ et *Keras*⁷. Cette tentative a bloqué lors de l'intégration du modèle dans l'architecture de l'outil Audiveris. Le modèle IA de Audiveris est complètement fait sur mesure. De plus, la représentation des partitions dans le code de Audiveris faisait en sorte qu'il était très difficile d'intégrer notre propre classificateur à l'outil. Il était difficile d'identifier quelles variables envoyer vers notre modèle et il aurait fallu les convertir vers un format accepté par celui-ci.

La deuxième tentative était de réentraîner le modèle existant. Audiveris offre un outil d'entraînement pour leur classificateur intégré à leur interface graphique. Il a donc été entraîné avec l'ensemble de données HOMUS², qui consiste en une série de milliers d'images de symboles musicaux manuscrits. Le modèle a ensuite été testé avec des partitions manuscrites de l'ensemble de partitions Musicma¹. Les résultats n'étaient malheureusement pas concluants.

Bien que certaines options d'Audiveris permettent d'augmenter le taux de reconnaissance, les résultats obtenus ne sont jamais entièrement satisfaisants. Pour atteindre un niveau de précision acceptable, il serait nécessaire qu'une autre équipe se consacre spécifiquement à l'amélioration de la reconnaissance des partitions manuscrites. Cela pourrait impliquer le développement de modèles d'apprentissage machine plus sophistiquée et une meilleure intégration avec Audiveris, afin de gérer efficacement les divers styles d'écriture manuscrite et d'assurer une conversion fidèle des partitions en fichiers MIDI.

Audiveris

Le projet que nous utilisons pour nous occuper de la reconnaissance de partition de musique a quelque limitation dans son fonctionnement.

Signes naturels dans l'armure

Dans le modèle de données présent de Audiveris, les seuls signes étant attendus dans l'armature sont les dièses et les bémols. Les signes naturels ne sont donc pas supportés. Ces signes occupent normalement le rôle d'indicateur pour le lecteur et ne devraient donc pas affecter le travail de notre projet. Cependant, leur seule présence pourrait décaler le reste de la mesure empêchant ainsi la reconnaissance correcte de l'armature. Il est donc préférable, dans la mesure du possible, présenter des partitions sans signes naturels ⁴.

Changement d'armure

Dans une partition, l'armature apparaît habituellement au début de la portée. Cependant, elle peut aussi être présente à la fin. Ce placement ne change pas la partition en elle-même, car une armature en fin sert uniquement à avertir le lecteur d'un changement d'armature à la prochaine portée. En revanche, puisque le moteur de Audiveris ne supporte pas encore ce placement, il est préférable de l'éviter afin de garantir une bonne lecture de la partition.

Tiges opposées

Par moment, il est possible de rencontrer deux notes dos à dos partageant la même tige. Dans cette situation, Audiveris détecte les deux notes, mais il est troublé, car il n'y a pas deux tiges. Cette confusion, va amener Audiveris à supprimer la tige et les deux notes puisqu'ils sont maintenant isolés.

RECOMMANDATIONS

Améliorations de l'interface utilisateur

Lors du développement, et même après celui-ci, nous avons reçu beaucoup de retours de la part des professeurs et du client concernant des améliorations à apporter à l'interface graphique. Certaines de ces améliorations concernaient l'interface utilisateur, tandis que d'autres relevaient plus de l'expérience utilisateur.

En suivant une approche agile, nous avons tenté d'implémenter ces changements et améliorations au fur et à mesure des sprints. Cependant, en raison de la durée limitée du projet, nous n'avons pas pu appliquer toutes ces améliorations. Voici donc une liste regroupant toutes les améliorations possibles que nous avons prises en note au cours du projet ainsi que nos suggestions pour les implémenter :

Révision des champs d'entrée pour la modification de tempo

Comme mentionné dans la section des problèmes connus, l'outil actuel de modification de tempo n'est pas assez intuitif pour les nouveaux utilisateurs.

Notre recommandation est donc de réviser la façon d'entrer les tempos. Nous suggérons d'adopter une méthode similaire à celle utilisée pour la rédaction de pages dans un service d'impression. Cela rendrait l'outil plus accessible et compréhensible, facilitant ainsi la tâche des utilisateurs pour ajouter ou modifier les tempos sur leurs partitions.

Ajout d'une étape de pré-conversion

Lors de nos différentes démonstrations, le client a mentionné à plusieurs reprises qu'il serait intéressant d'avoir un étape de pré-conversion permettant d'obtenir de l'information sur la conversion à venir. Puisque la conversion peut prendre un certain temps non négligeable, il serait en effet intéressant d'avoir, par exemple, un message qui alerte l'utilisateur lorsqu'aucun tempo n'a été détecté avant même d'effectuer la conversion. Cette notification pourrait également informer l'utilisateur des tempos spécifiques qui ont été détectés et appliqués. Ce type de retour d'information aiderait l'utilisateur à comprendre les actions automatiques effectuées par l'application et à effectuer les ajustements nécessaires plus facilement.

Pour cette amélioration, nous suggérons premièrement de faire une analyse de faisabilité avant d'entreprendre le travail. Nous ne sommes pas certains qu'avec Audiveris il soit possible d'effectuer seulement la détection du tempo sans effectuer le processus de conversion au complet.

Barre de progression de conversion

La conversion d'une partition peut être longue, laissant l'utilisateur se demander ce qui se passe. Lors des démonstrations, nous avons reçu à plusieurs reprises le retour qu'il serait pertinent d'offrir un retour plus informatif à l'utilisateur.

Notre recommandation pour améliorer le retour pendant la conversion du fichier est d'ajouter une barre de progression. Cette fonctionnalité permettrait aux utilisateurs de suivre l'avancement du processus en temps réel, réduisant ainsi l'incertitude et améliorant l'expérience utilisateur. Une barre de progression visuelle indiquerait le temps restant pour compléter la conversion, rendant l'application plus agréable à utiliser.

En complément, nous suggérons d'afficher des messages explicatifs indiquant l'étape actuelle de la conversion. Pour cela, il serait nécessaire de trouver un moyen de renvoyer les messages d'Audiveris vers l'interface utilisateur et d'implémenter un système permettant d'estimer le temps restant pour chaque étape ou message reçu et exécuté.

Ajout de message d'erreurs plus significatif

Actuellement, lorsque survient une erreur lors de la conversion de la partition, notre application frontale se contente d'afficher un message d'erreur générique. Cela n'est pas idéal pour l'utilisateur, car ce type de message ne fournit aucune indication sur la nature du problème ni sur la manière de le résoudre.

Nous recommandons donc d'afficher des messages d'erreur plus spécifiques et adaptés à la situation. Ces messages devraient inclure des détails sur l'erreur rencontrée ainsi que des suggestions pour aider l'utilisateur à corriger le problème survenu. En offrant des messages d'erreurs plus détaillés et utiles, l'expérience utilisateur serait améliorée, rendant l'application plus intuitive et réduisant la frustration liée aux erreurs imprévues.

Fonctionnalité de prise de photo

Une amélioration que nous jugeons particulièrement importante serait d'ajouter une fonctionnalité de prise de photo pour les utilisateurs sur appareils mobiles. Cette fonctionnalité permettrait aux utilisateurs de capturer directement une image d'une partition musicale via l'interface de l'application, sans avoir à passer par l'étape intermédiaire de sauvegarder le fichier dans leurs documents.

Améliorations de l'application serveur

Plusieurs améliorations concernant l'application serveur (backend) ont été identifiées au cours du développement. Ces améliorations visent à optimiser la conversion, à améliorer la qualité de celle-ci, et à rendre le projet plus maintenable. Nous recommandons fortement d'entreprendre ces améliorations.

Adopter un seul langage pour l'application serveur

Nous avons utilisé plusieurs technologies et langages de programmation pour concevoir notre solution. Ces choix ont été en partie dictés par la contrainte de temps. Cependant, pour faciliter le développement futur de l'application et améliorer sa maintenabilité, nous recommandons d'utiliser un seul langage de programmation pour l'application serveur, à savoir Java. Cela implique de retranscrire le code open source de Music21, actuellement en Python, vers Java.

L'utilisation d'un seul langage réduit la complexité du projet en unifiant le code, ce qui simplifie sa compréhension et son maintien.

Extraction de l'Engin Audiveris pour Optimiser la Conversion

Actuellement, nous utilisons une distribution complète d'Audiveris pour réaliser la conversion, en appelant l'application via la ligne de commande depuis notre code. Cependant, cette approche n'est pas nécessairement la plus efficace. Nous pensons qu'il est possible d'obtenir une conversion plus rapide et de meilleure qualité en intégrant directement le code d'Audiveris dans notre application.

Notre suggestion est d'extraire le moteur de conversion d'Audiveris et de supprimer le code superflu, tel que l'interface utilisateur, la reconnaissance des paroles, et les options inutiles. En procédant ainsi, nous pourrions intégrer directement le moteur d'Audiveris dans notre application serveur. Cette amélioration permettrait de développer une solution plus autonome, adaptée aux besoins spécifiques du club musiqueÉTS, et indépendante des futures évolutions d'Audiveris.

Ajout d'un Pipeline de Développement et d'Intégration Continue

Nous recommandons également de mettre en place un pipeline CI/CD dans le dépôt GitHub. Ce pipeline permettra d'automatiser l'exécution des nombreux tests que nous avons développés, garantissant ainsi la qualité du code à chaque mise à jour.

En intégrant ce pipeline, le club étudiant pourra facilement déployer de nouvelles versions de l'application sur DockerHub. Cela garantira que la version la plus récente soit toujours disponible pour le déploiement, simplifiant ainsi la maintenance et les mises à jour continues de l'application. De plus, ce pipeline permettra de valider le bon fonctionnement des contributions extérieures.

CONCLUSION

Pour conclure, notre solution logicielle devrait permettre au club étudiant musiquÉTS de convertir leurs partitions en fichiers MIDI lisibles par leurs instruments mécaniques automatisés. Pour en venir à cette solution, nous avons envisagé soit de partir d'un projet existant ou bien de coder un nouvel outil OMR. Grâce au projet Audiveris, nous avons pu développer une solution complète au club de manière agile pour nous assurer de satisfaire leurs exigences. Nous avons expliqué les différents enjeux et problèmes que nous avons rencontré lors du développement et les solutions trouvées. Nous avons ensuite décrit la suite de tests conçue pour vérifier l'intégrité de notre application et la méthode utilisée pour déployer la solution.

En perspective, cette solution offre la possibilité au club musiquÉTS d'ajouter des fonctionnalités. Le support de plusieurs formats de fichiers, l'ajout d'un outil de capture photo d'une partition papier et l'export d'autres composants comme les paroles sont tous envisageables. Cet outil pourrait même servir hors du contexte du club. La licence du code est d'ailleurs ouverte au public, ce qui permet à n'importe qui d'y faire des ajouts. Les interactions entre humains et ordinateurs sont de plus en plus communes aujourd'hui. Une performance musicale entièrement dirigée par des robots mécaniques et des ordinateurs fait perdurer cette tendance. Il faut donc continuer à se questionner sur les effets que la coexistence des humains et des robots apporte à l'humanité.

LISTE DE RÉFÉRENCES

1. Musicma dataset - <https://ufal.mff.cuni.cz/muscima>
2. HOMUS dataset - <https://grfia.dlsi.ua.es/homus/>
3. Docker - <https://docs.docker.com/reference/>
4. Audiveris - <https://github.com/Audiveris/audiveris>
5. Maestro - <https://github.com/MaestroETS/PFE008>
6. Tensorflow - <https://www.tensorflow.org/>
7. Keras - <https://keras.io/>

ANNEXE A : PLAN DE TRAVAIL

Le tableau suivant présente la planification pour la réalisation des tâches ou artefacts décrits précédemment.

#	Commence	Termine	Tâches/Jalon	Livrable(s)/Artéfacts	Responsable(s)
1	24 avril 2024	24 avril 2024	entrer en contact avec les superviseur		Chef d'équipe
2	8 mai 2024	9 mai 2024	organiser l'horaire des rencontre		Chef d'équipe
2.1	8 mai 2024	8 mai 2024	Rencontre – professeur superviseur		Animateur
2.2	15 mai 2024	15 mai 2024	Rencontre – client		Animateur
3	15 mai 2024	29 mai 2024	Remise du plan de projet	Plan de projet	Chef d'équipe
3.1	15 mai 2024	29 mai 2024	Remise du Document d'analyse de besoins	Document d'analyse de besoins	Chef d'équipe
3.2	15 mai 2024	29 mai 2024	Remise de l'analyse de risque	analyse de risque	Chef d'équipe
3.3	15 mai 2024	29 mai 2024	Remise du plan de test	plan de test	Chef d'équipe
4	29 mai 2024	26 juin 2024	Réalisation des sprint 1 et 2		toutes l'équipe
5	26 juin 2024	26 juin 2024	Démo / Rencontre – professeur superviseur		Animateur
6	26 juin 2024	10 juillet 2024	Réalisation du sprint 3		toutes l'équipe
6.1	10 juillet 2024	10 juillet 2024	Remise du rapport d'étape (<i>facultatif</i>)	Rapport d'étape	
6.2	10 juillet 2024	24 juillet 2024	Réalisation du sprint 4		toutes l'équipe
7	24 juillet 2024	24 juillet 2024	Rencontre – professeur superviseur		Animateur
8	24 juillet 2024	31 juillet 2024	Réalisation du sprint 5		toutes l'équipe
	31 juillet 2024	31 juillet 2024	Présentation finale au client		toutes l'équipe
9.1	12 août 2024	12 août 2024	Soutenance	Présentation	toutes l'équipe
9.2	8 août 2024	8 août 2024	Remise du projet	Projet	Chef d'équipe
9.3	19 août 2024	19 août 2024	Remise du rapport final	Rapport de projet	toutes l'équipe

Tableau 5 : Plan de travail




ANNEXE B : LISTE DES ÉLÉMENTS D'UNE PARTITION

Notes : Do, Ré, Mi, Fa, Sol, La, Si

Accidents : Dièse (#), Double dièse (x), Bémol (b), Double bémol (bb), Bécarré (⌢)

Silences : Pause, Demi-pause, Soupir, Demi-soupir, Quart de soupir, Huitième de soupir, Seizième de soupir

Figures de Note : Ronde, Blanche, Noire, Croche, Double croche, Triple croche, Quadruple croche

Clés : Clé de sol () , Clé de fa () , Clé d'ut ()

Armures : Armure de dièses (jusqu'à 7 dièses), Armure de bémols (jusqu'à 7 bémols)

Indications de Mesure

Indications de Tempo : Marquage du tempo (Andante, Allegro, ...), BPM (beats per minute)

Ligatures : Liaison, Liaison de phrasé

Articulations : Staccato (point), Accent (>), Tenuto (—), Marcato (^)

Ornements : Trille, Mordant, Grupetto

Dynamique : Pianissimo (pp), Piano (p), Mezzo-piano (mp), Mezzo-forte (mf), Forte (f), Fortissimo (ff), Crescendo (crescendo), Decrescendo (decrescendo)

Barres de Mesure : Barre simple, Double barre, Barre de reprise (ouverte et fermée)

Autres : Clefs de lecture, Indications de répétition, Signes de coda et de segno, ghost note