Rapport projet de formation 3WA

SOMMAIRE

Introduction

Le but de ce projet est de proposer un mini site internet au sein duquel il est possible de générer des suites de mélodies grâce à une intelligence artificielle. L'idée est venue d'une discussion avec un professeur de piano, sur la capacité créative des intelligences artificielles dans le domaine musical et il en est ressorti cette idée, qui pourrait permettre de libérer de la créativité chez des musiciens, afin de trouver une inspiration grâce à l'intelligence artificielle.

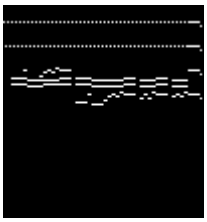
Pour arriver à ce résultat, ce projet est composé d'une intelligence artificielle, qui à été conçue et entraînée sur mesure par mes soins à partir de données rigoureusement sélectionnées et traitées. D'une partie Backend au sein de laquelle on retrouve l'import et l'utilisation de l'IA au sein d'une API communiquant avec une base de données NoSql. Et enfin d'un frontEnd, réalisé avec le framework React pour donner une interface visuelle à cet outil. Bien entendu, chaque partie de ce projet à été réalisée en partant de zéro. Ainsi nous verrons dans un premier temps tout ce qui découle de l'intelligence artificielle dans ce projet (du choix des données initial à l'entraînement et sauvegarde du modèle). Nous verrons ensuite comment a été pensé et réalisé le backend, et enfin nous finirons par le frontEnd.

1] L'intelligence artificielle.

1.a] Quantification et Qualification des données

Comme pour tout projet ou il faut créer et entraîner une intelligence artificielle, il faut des données à consommer pour cette dernière. Dans le cadre de ce projet, les données en question sont des données audios. La première réflexion à donc été de savoir quels fichiers audio utiliser parmis tout ceux existants. Etant donné que le but du projet est de réaliser seulement des mélodies, le but était d'avoir uniquement des mélodies, sans accompagnement. Ensuite il fallait choisir les formats audio souhaités.

Pour cela il fallait tout d'abord savoir comment la musique allait être traitée ensuite par l'IA.



La première idée fût de récupérer des fichiers mp3, et de transformer le spectre audio en image. Mais cette idée à très vite été abandonnée : Les fichiers sont lourds, le traitement de l'image aussi et il n'est en plus pas précis.

La deuxième idée fut donc de récupérer au format MIDI (Musical Instrument Digital Interface) des mélodies. Ce format, utilisé par exemple par les instruments électroniques, est bien plus permissif. Ensuite, la découverte du format krn à été faite, et c'est ce format qui sera utilisé par la suite, avec un gros dataset de mélodies trouvé sur une banque de données de fichier krn. Ce format est tout simplement un format MIDI compressé, plus léger donc et plus rapide à manipuler.

1.b] Préparation des données

Une fois les données utilisables définies et en quantité suffisante pour être consommées par une IA est venue l'étape de la préparation de ces dernières. Le son en temps que tel n'est pas utilisable par une intelligence artificielle et il faut donc trouver un moyen de le décomposer et de l'encoder en un format consommable par l'IA. La première étape à donc été de trouver un moyen de "transformer" ce son. C'est là qu'intervient une librairie python : Music21. Cette dernière permet de décomposer les fichiers MIDI (et Krn par extension) afin d'en récupérer les notes, les durées en temps de ces dernières et les pauses, soit les temps sans notes jouées. Avec toutes ces données, réaliser une série temporelle semble être la solution idéale : un vecteur, au sein duquel la musique y est représentée dans une suite logique ou chacune des unités représente un temps et ou les notes sont notées de 1 à 88 inclus, les notes maintenues sont notées "_" et les temps de pause sont notées "r".

exemple : `["r", "_", "49", "_", "_", "_", "58" "_"]`

Appliqué sur l'ensemble du dataset composé de 1700 mélodies, il apparaît très vite problématique la diversité énorme de notes, ainsi que les soucis de mélodies non encodables. Pour pallier ces problèmes, deux optimisations sont réalisées sur les sons avant de les encoder :

Premièrement, une vérification de la temporalité de la mélodie est faite : Il est défini les temps de notes acceptés (ACCEPTABLE DURATIONS) qui correspondent à la durée d'une note sur un temps de la mélodie. Ces durées acceptables correspondent ni plus ni moins qu'à une note jouée en adéquation avec le tempo de la musique. Ainsi, une fonction de tri est créer, pour ne retourner que les musiques dont l'ensemble des notes est compris dans ces temps acceptables, afin d'écarter toutes mélodie complexe (mélodie contenant par exemple des notes jouées à contretemps, ou encore des mélodies dont le tempo change de manière imprévisible).

Deuxièmement, afin d'éviter d'avoir les 88 notes dans la série temporelle totale, qui ne sont en fait que 7 notes réparties sur des octaves (Do (C), Ré (D), Mi(E), Fa(F), Sol(G), La(A) et Si(B)) une décision de transposer les sons est prise. Cette transposition consiste à détecter la gamme sur laquelle la mélodie est jouée, et de la transposer en Do Majeur lorsque c'est une gamme majeure ou en La mineur lorsque c'est une gamme mineure. Grâce à cela, la quantité totale de notes est grandement réduite (on passe de 88 à 8 notes). Cela permet donc de grandement réduire les futures possibilités de prédiction, et donc d'améliorer les chances de réussite de l'entraînement de L'IA. Cependant, il faut noter que cette méthode rend l'ensemble des mélodies plus monotone, étant donné que toutes leurs notes seront jouées sur la même gamme.

Une fois ces étapes réalisées, on se retrouve donc avec les sons ayant ses notes dans les durées acceptables, et encodés sous forme de séries temporelles. L'étape suivante est de concaténer tous ces fichiers en un, afin de se préparer pour les futurs générations de séquences pour entraîner l'intelligence artificielle. Puis, de générer un fichier mapping, correspondant à un json contenant l'ensemble des notes de toutes les mélodies afin de donner un "poids" à chacune des notes (au plus une note est présente, au plus son poids est important) et donc d'exclure les exceptions, comme les notes qui apparaissent très rarement sur l'ensemble des mélodies.

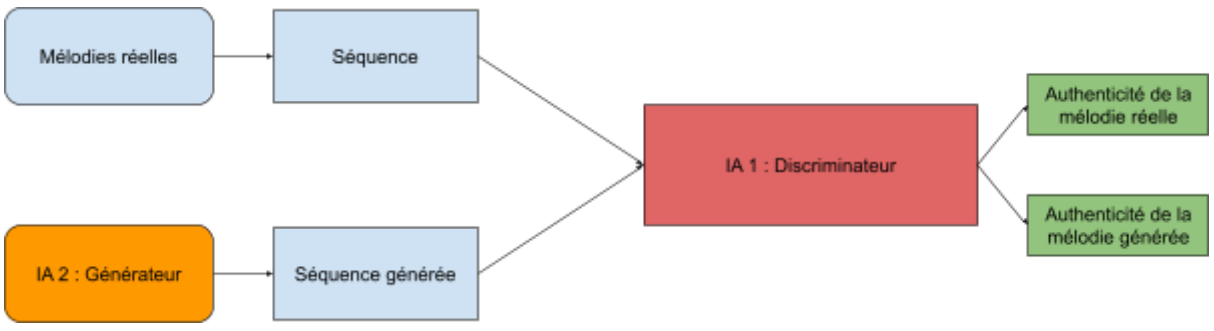
Il ne reste plus qu'à préparer les séquences d'entraînement pour le futur modèle, ces séquences ont été définies d'une longueur de 64 unités, un choix arbitraire certes discutable mais qui semble assez optimal. Les séquences représentent donc une série temporelle de 64 unités de longueur en input, pour un output attendu d'une note, qui correspond à la

note suivant la 64eme note de l'input. Pour encoder l'input, est utilisée la fonction de Keras to_categorical pour transformer la séquence en matrice binaire lisible par l'IA. Il existe un nombre de séquences (inputs) et d'outputs équivalents à l'ensemble des séries temporelles de toutes nos mélodies / 64 (la longueur souhaitées des séquences) - 64 (une longueur de séquence, qui correspond à la séparation entre les mélodies dans le SINGLE_FILE_DATASET

1.c] Conception et développement de l'intelligence artificielle

La première idée lors de la conception de l'intelligence artificielle pour générer des mélodie était de réaliser un GAN (Generative adversarial Network). La GAN est une technique de machine learning mettant en jeu deux intelligence artificielle : un générateur et un discriminateur « . Le générateur aurait pour rôle de générer la mélodie. tandis que le discriminateur aurait pour rôle de déterminer si la mélodie paraît authentique et réaliste ou non.

Pendant le processus d'entraînement, ces deux entités sont donc en compétition et c'est ce qui leur permet d'améliorer leurs comportements respectifs. L'objectif du générateur aurait donc été de produire des mélodies sans que l'on puisse déterminer s'ils elles sont fausses, tandis que le discriminateur aurait cherché à détecter si oui ou non la mélodie est une génération. Ainsi, au fil du processus, le générateur produit des outputs de meilleure qualité tandis que le discriminateur détecte de mieux en mieux les faux. Ainsi, le résultat n'aurait été que meilleur avec du temps.

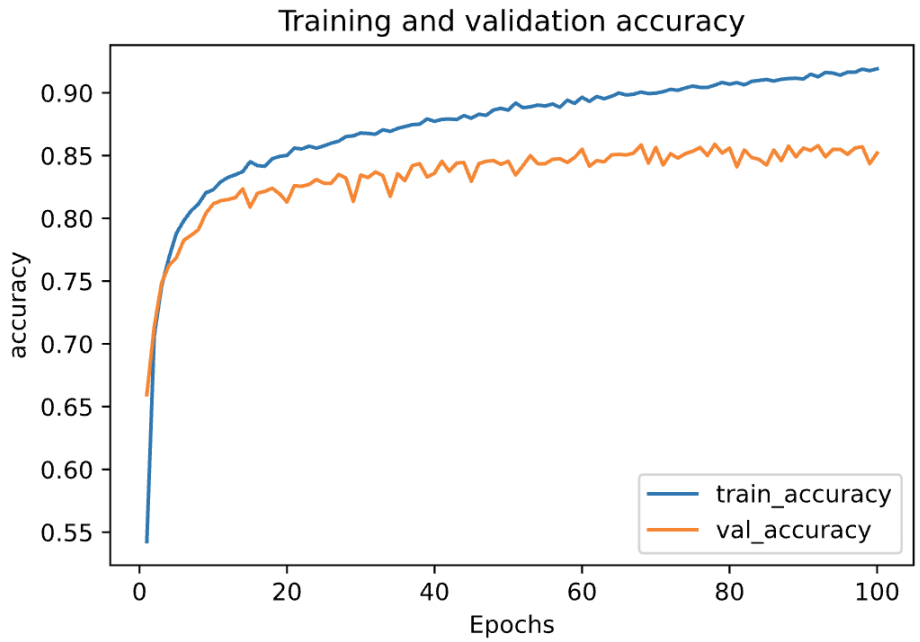


Si cette méthode est la plus efficace, elle n'est cependant pas la plus optimale compte tenu du rapport entre le temps donné pour ce projet et la complexité pour réaliser un GAN de qualité. C'est pourquoi après réflexion, la solution optimale semblait être un RNN (Recurrent Neural Network), qui de par son concept de récurrence, prédit la prochaine note en se servant des informations antérieures fournies (les autres notes qui précèdent encore). Mais comme une mélodie est quelque chose qu'on peut visualiser sous forme "d'étapes" (couplet/refrain) la décision d'utiliser un modèle LSTM (Long short term Memory) a été prise. Ce dernier, grâce à sa cellule de "mémoire" va permettre d'enregistrer et d'apprendre des séquences sur le long terme, comme les refrains répétitifs sur les mélodies par exemple.

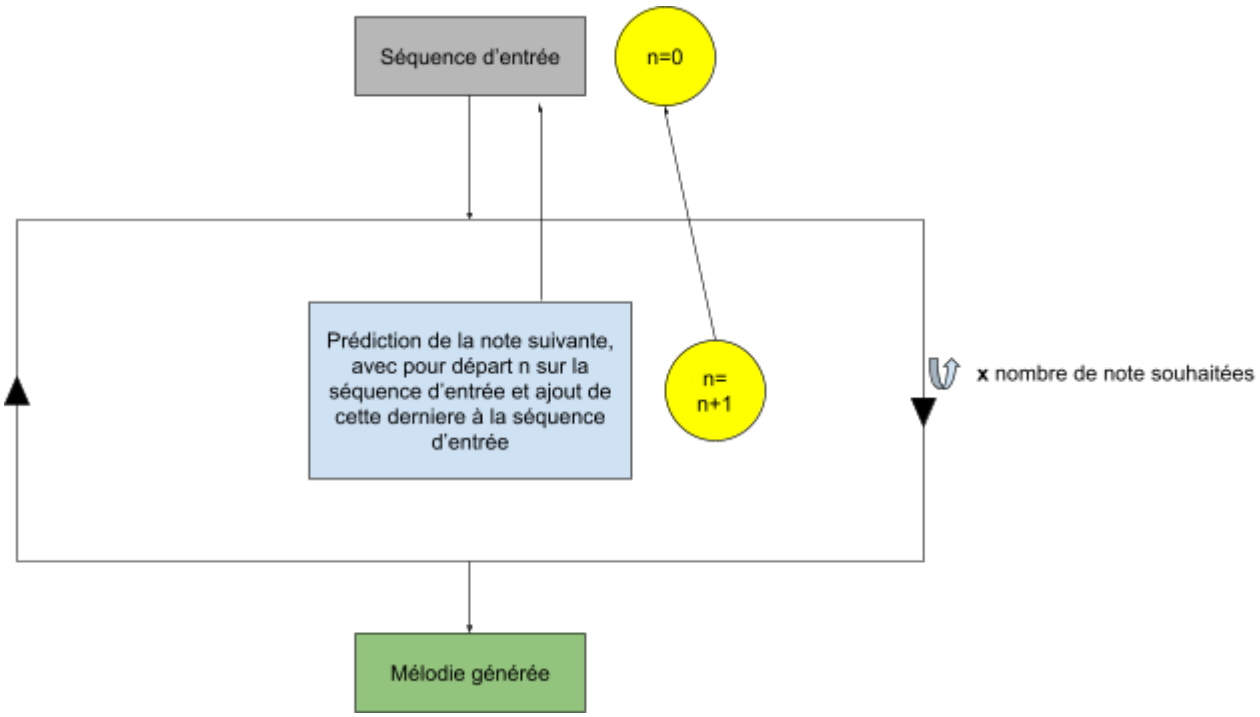
Concernant la conception du modèle, une première couche input est ajoutée, contenant 256 unités (neurones) , puis la couche LSTM contenant elle aussi 256 neurones. Un dropout de 0.2 est ajouté, donnant ainsi à chaque neurone 20% de chance d'être désactivé pendant l'entraînement et évitant ainsi un surentraînement du modèle. Enfin est ajoutée une couche de base, Dense elle aussi de 256 neurones à activation softmax permettant d'avoir un ensemble de probabilité de note en output, et retournant le plus proche de 1 (la note la plus susceptible d'être jouée).

Concernant les hyperparamètres, une méthode empirique à été utilisée afin de définir plus ou moins les bons paramètres concernant le nombre de neurones, le nombre de couches et le nombre d'époques d'entrainements. En se servant notamment des résultats lors des entraînements sur les valeurs de la loss, ou encore l'accuracy du modèle et la vitesse d'apprentissage.

La fonction loss elle à été directement définie comme "sparse_categorical_crossentropy" car après de multiples recherches sur les réseaux LSTM dans la génération de musique, cette fonction est apparue comme la plus récurrente. De même pour l'optimiseur Adam, qui d'après plusieurs recherches apparaît comme un excellent optimiseur par défaut pour la plupart des applications.



Une fois l'entraînement terminé et des résultats satisfaisants (plus de 95% de précision sur la prédiction de l'output), une classe MelodyGenerator a été réalisée afin d'appliquer ce modèle a une mélodie donnée afin d'en générer la suite. l'idée est de récupérer la musique, et comme pour les mélodies du dataset, vérifier si l'ensemble de ses notes est acceptable, la transposer si nécessaire puis l'encoder au format timeSeries comme vu plus haut. Enfin, le modèle nécessitant un input long de 64 unités de temps, le choix de prendre uniquement les 64 premières unités de temps de la musique passée en input est pris (l'idée est d'avoir ainsi le début de la vraie mélodie, qui est suivi par une suite générée par l'IA). A partir de cette séquence, on peut définir combien de notes l'on souhaite prédire, puis tout simplement boucler sur le nombre de notes souhaitées en ajoutant à chaque fois la note retournée par l'IA et en décalant donc le début de la séquence de 1 vers la fin.



Une fois la mélodie générée, la librairie Music 21 est à nouveau utilisée pour faire chemin inverse et encoder la mélodie actuellement au format timeSeries au format MIDI, et l'écrire dans un fichier, en ajoutant pour plus de personnalisation la possibilité d'augmenter le facteur de vitesse de la musique (son tempo)

2] Le backend

2.a] Implémentation de l'intelligence artificielle

Une fois l'IA entraînée est prête à être utilisée, la création du backend de l'application a pu commencer. l'idée était de faire un backend simple, en python afin de permettre l'import et l'utilisation à la fois du modèle mais aussi de la classe de génération de mélodie qui y fait appel, ainsi que les fonctions permettant de modeler et rendre compatible au traitement par l'IA tout ce que l'on enverra ensuite depuis le frontEnd. Ainsi l'implémentation coté backend de l'intelligence artificielle s'est fait comme un import de librairie classique pour les classes/fonctions et le modèle d'IA.

2.b] Base de données

Concernant la base de données, la décision de partir sur une petite base de données NoSQL (MongoDB) a été prise. Le choix de cette technologie repose principalement sur les affinités avec cette dernière. Habitué à travailler avec des bases de données documents, le choix est apparu comme une évidence. Cette base de données contient deux collections :

-**Users** qui contient toutes les informations des utilisateurs de l'application (identifiant unique, emails avec clé primaire unique, mot de passe)

-**Melodies** qui contient les midi des mélodies générées par l'IA depuis le site, encodées en base64 pour des transferts plus simples entre front et backend ainsi que pour pouvoir les stocker directement en base de données.

Les principales interactions avec la base (création de celle-ci, des collections, ajout des clés) ont été réalisées directement depuis l'interface visuelle de MongoDb : Compass (toujours pour raisons d'affinité car utilisé en entreprise).

2.c]API

Le cœur du backend, et ce qui lui permet de communiquer par la suite avec le frontend se trouve dans l'api. Pour réaliser cette api, un framework python déjà utilisé en entreprise a été utilisé : FastAPI. Bien qu'encore assez méconnu, il est pourtant très performant et assez simple dans la conception. D'autant plus qu'il délivre une documentation automatique à la création d'une API.

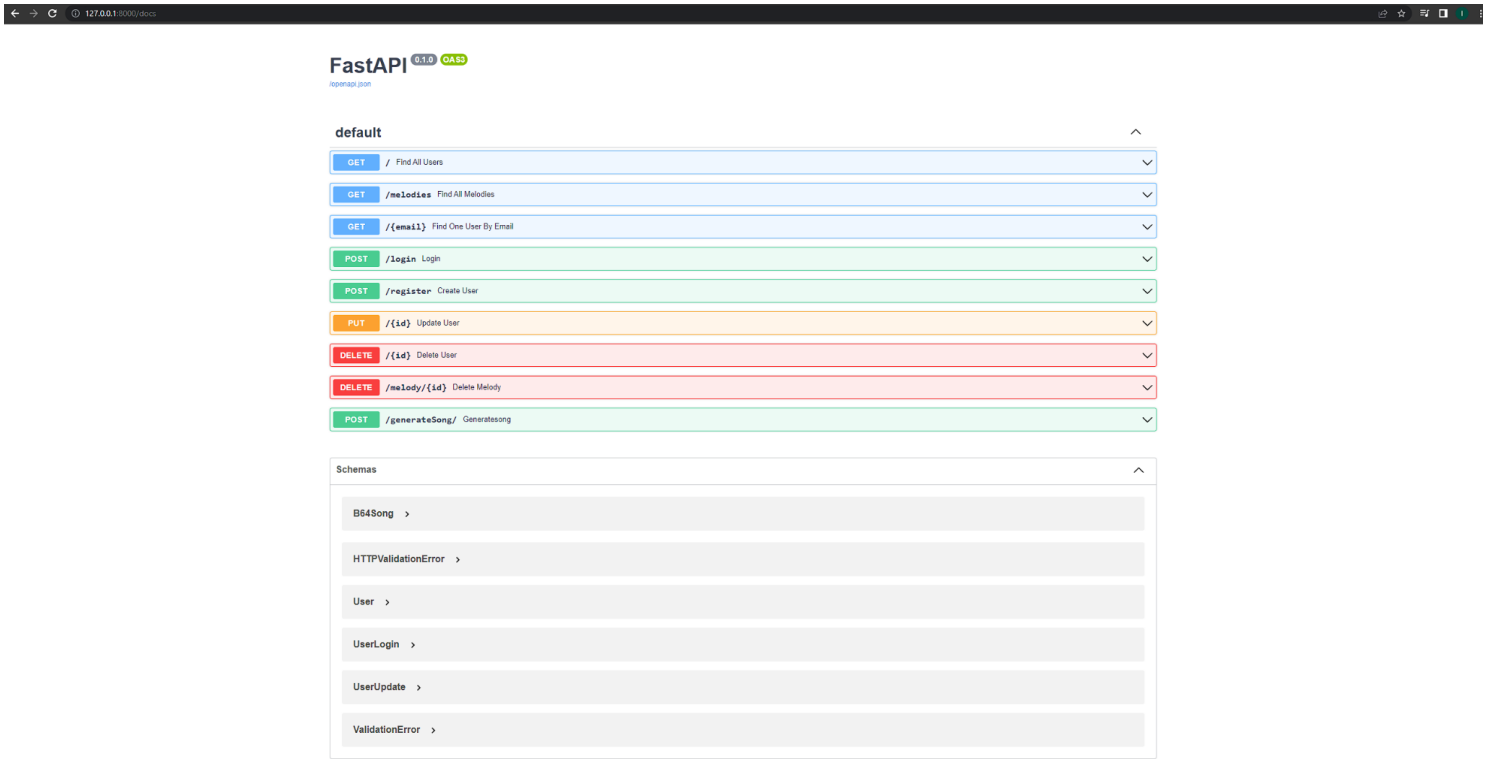
La création de l'api avec ce framework s'est déroulée en 3 étapes clefs qui ont été successivement la configuration de l'api (connection avec la base de données), la création des routes (chemins auxquels le front passera ensuite les requêtes pour obtenir des réponses et/ou effectuer des actions sur la base de données), et enfin la sécurisation de l'API au travers de règles de sécurité simples mais efficaces : Cryptage des mots de passe, système d'authentification avec token unique, et accès à l'API depuis et vers l'IP du serveur sur lequel le frontend sera hébergé.

La configuration de l'api est assez simple avec fastAPI. En effet, en utilisant le package motor_asyncio la connexion à la base de données se fait en quelques lignes, et permet ainsi par la suite d'interagir avec cette dernière grâce aux fonctions délivrées (find, update, insert, delete).

Une fois cette configuration opérationnelle, la création des routes commence. Un total de 8 routes est créé, et chacune d'elle est liée à une méthode de requête spécifique (GET, POST, PUT, DELETE). Également, lorsqu'une route attend un paramètre, un modèle de ce paramètre est implémenté afin d'empêcher toute requête nécessitant un paramètre d'avoir lieu si le paramètre en question est manquant ou a un modèle différent de celui attendu. Voici les routes réalisées :

- "**/** ": Route de base, elle attend une requête GET sur l'url de l'API et retourne la liste de tous les utilisateurs.
- "**/{email}**" : Route pour requête GET, qui attend comme paramètre un email et retourne dans le cas où ce mail est en base de données, les données de l'utilisateur lié à ce dernier.
- "**/login**" : Route en Post, attend comme paramètre une requête correspondant au modèle UserLogin, contenant email et mot de passe. Si l'email et le mot de passe correspondent à un existant en BDD retourne l'identifiant de l'utilisateur ainsi que token d'authentification unique

- “**/register**” : Route en POST, attend comme paramètre une liste contenant email et mot de passe (avec modèle assurant la bonne forme). Si l’email n’existe pas déjà, créé un utilisateur en base de données et retourne une réponse valide.
- “**/{id]”** : Requête PUT avec comme paramètre attendu un identifiant utilisateur, ainsi que les informations à modifier. Si tout correspond, modifie l’utilisateur avec les nouvelles informations reçues.
- “**/{id}**” : Requete DELETE : Attend un identifiant utilisateur afin de supprimer ce dernier de la base de données.
- “**/generateSong**”: Requête POST. Attends un son encodé en base64, afin de faire appel à l’IA pour générer la suite de la mélodie, puis insert en base de données le résultat et retourne la mélodie en base64 vers le frontend.
- “**/melodies**” : Requête GET : Retourne toutes les mélodies présentes en base de données
- “**/melody/{id}**” : Requête DELETE : Supprime la mélodie correspondant à l’id passé en paramètre de la base de données.



Une fois toutes ces routes réalisées et testées (Les routes ont été doublement testées, à la fois depuis la documentation générée par FastAPI, mais aussi grâce à POSTMAN, un outil permettant d'exécuter des requêtes), la troisième étape de la conception de l'API est la sécurisation des données.

Ainsi, en premier lieu, l'import de la librairie CryptContext permet de hacher au format bcrypt les mots de passes entrés en base de données. Ce hachage étant impossible à décrypter, on peut seulement vérifier que les mots de passe entrés ensuite correspondent. Il est impossible de traduire une version hachée pour obtenir le mot de passe, rendant ainsi impossible la récupération de mots de passe depuis la base de données.

Ensuite, la conception d'un système d'authentification a été faite pour les routes liées à ce dernier. Ce système permet de délivrer un token d'accès, valable 2h lors de la connexion et ce dernier est demandé pour chaque requête nécessitant un utilisateur connecté. Ainsi, il est impossible de requêter les routes de l'API nécessitant un utilisateur si l'on est pas réellement connecté. Également, étant donné que chaque token délivré est unique, il est aussi impossible de récupérer un token existant afin de l'utiliser ultérieurement et/ou le partager.

3] Le frontend

3.a] Interface

Pour le frontend de ce projet, la décision d'utiliser le framework React à été prise. Cette décision s'appuie sur deux éléments qui sont premièrement l'affinité avec ce framework, utilisé quotidiennement en entreprise et apprécié, et également le fait qu'il correspond parfaitement aux besoins pour le frontend : Une interface one page simple et rapide. L'idée est d'avoir un site simple protégé par authentification obligatoire, et une fois connecté donnant accès à un générateur de mélodie, auquel on doit passer un fichier au format midi d'une mélodie, choisir le facteur de vitesse voulu, puis laisser le backend récupérer les informations et faire appel à l'IA pour générer la suite de cette mélodie. Enfin, une page listant toutes les mélodies générées est disponible, avec possibilité de supprimer les mélodies.

3.b] Fonctionnalitées

Les fonctionnalités côté front peuvent être séparées en trois catégories : L'authentification, La génération de musique et l'accès à ces dernières.

Il y a donc tout d'abord l'authentification, via deux composantes qui sont la connexion et l'inscription, qui font appel à l'API afin d'authentifier l'utilisateur. Une fois l'authentification effectuée, le Token retourné par l'API est enregistré en tant que Cookie et un provider de React (Auth) est défini comme Vrai. Ainsi, une logique autour de l'authentification est définie, afin d'avoir un affichage différent en fonction de l'état (Authentifié ou non) dans lequel l'utilisateur se trouve.

Ensuite concernant la génération, il s'agit d'un formulaire délivrant une requête POST sur la route GenerateSong de l'API, avec pour paramètre le fichier midi encodé en base64 depuis React.

Enfin la récupération et lecture des mélodies créées par l'IA sont sur une page dédiée, au sein de laquelle React requête l'API en GET pour obtenir les sons, puis boucle la réponse dans le render de la page.

Toutes les requêtes effectuées côté frontEnd sont gérées avec une librairie, Axios qui est assez similaire sur l'utilisation au Fetch natif de Javascript mais à tout de même plus de possibilité, étant donné que les réponses

sont des promesses. On peut ainsi intercepter les réponses et gérer les erreurs plus facilement qu'avec une requête Fetch classique.

Enfin, les sons reçus étant des base64 de fichier MIDI, une librairie Jazz est utilisée pour lire ces fichiers depuis le frontend.

4] Relation frontend - backend - IA

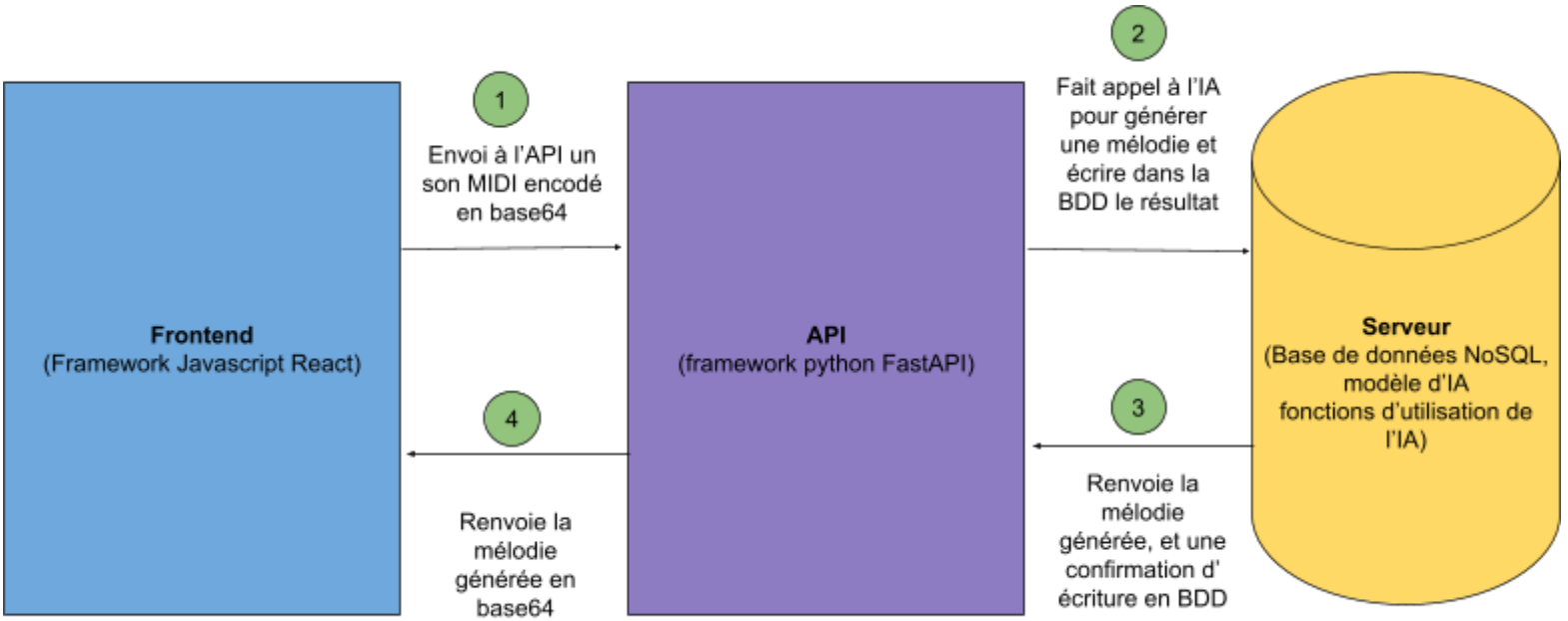


Schéma des relations lors de la génération d'une mélodie

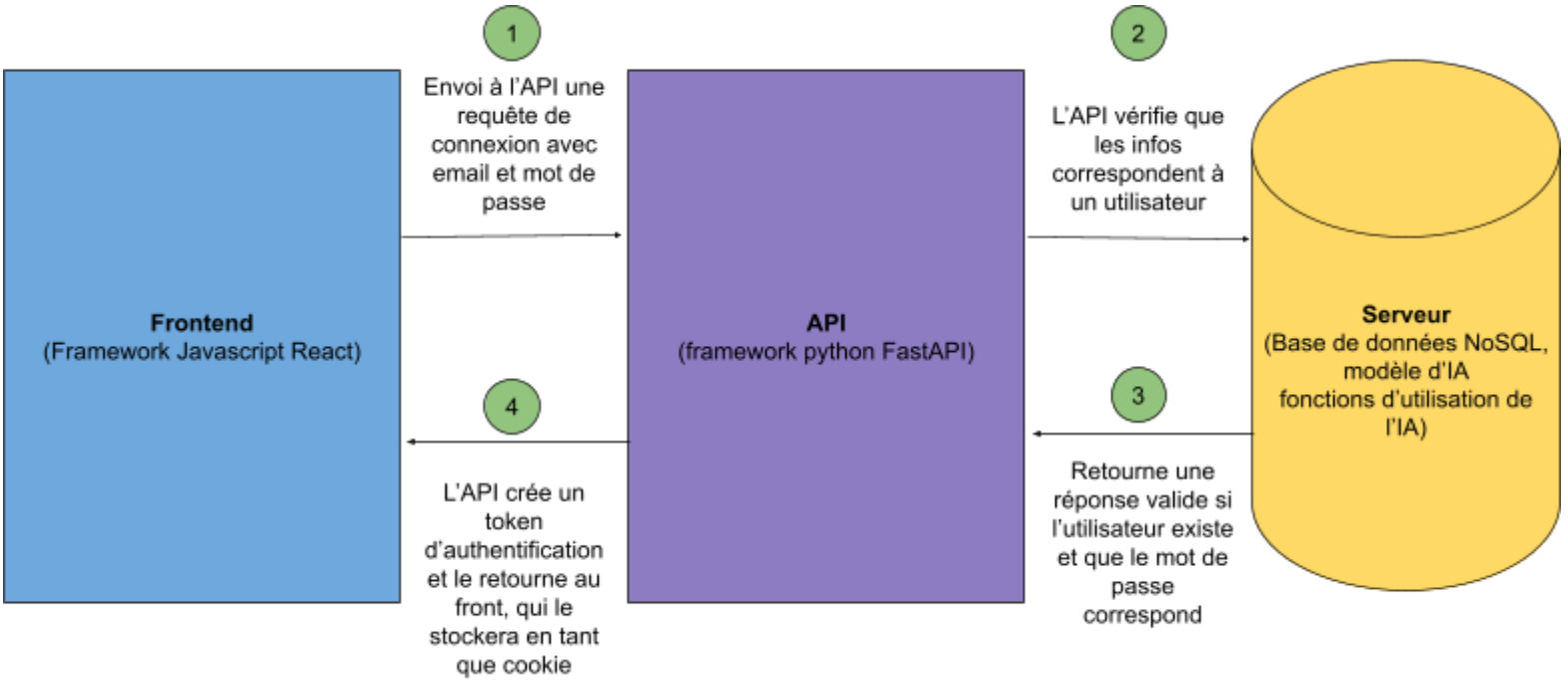


Schéma des relations lors de la connexion d'un utilisateur

5] Conclusion

Ce projet, à été intégralement réalisé avec une méthode Agile (Chaque partie du projet a été divisé en sous parties, soit des tâches à réaliser en un court terme c'est-à-dire des sprints). Il couvre les deux grandes facettes du développement web (frontend et backend) ainsi que le domaine de l'intelligence artificielle. Un long travail de recherche sur l'audio dans l'IA fût nécessaire, couplé à des connaissances personnelles dans le domaine musical ayant permis de prendre des choix décisifs dans la préparation des données. Le travail pour le frontend ainsi que pour le backend étant bien plus proche du travail réalisé en entreprise, cela à nécessité moins de temps. Le résultat final correspond globalement à ce qui était attendu, mais un laps de temps supplémentaire aurait pu être bénéfique, afin d'étoffer d'une part le front-End et d'autre part améliorer encore le modèle d'IA afin de le rendre compatible non plus avec seulement des mélodies, mais aussi avec les accompagnements de ces dernières (afin de générer à la fois la mélodie et la rythmique)