ÉCOLE NATIONALE DE LA STATISTIQUE
ET DE L'ANALYSE DE L'INFORMATION

PROJET INFORMATIQUE

2A

Rapport final

Mathis DAVID-QUILLOT
Linh-Da DINH
Apolline GUERINEAU
Oussama MARHFOUL
Mathieu THOMASSIN

Chargée de projet :
Armelle KOEHL

Jeux de mots

2022-2023

Table des matières

Table des figures

1 Introduction

L'application *Kata* est un jeu de lettres inspiré de jeux tels que *Wordle* ou *Sutom*. Le but du jeu est de deviner un mot spécifique caché en un nombre de tentatives limité. Le joueur gagne s'il parvient à trouver le mot avant que son nombre de tentatives ne soit épuisé. Lorsque le joueur fait une proposition, celle-ci lui est retournée en couleur : vert si la lettre est bien placée, jaune si la lettre est dans le mot mais mal placée et sans couleur si la lettre n'y est pas.

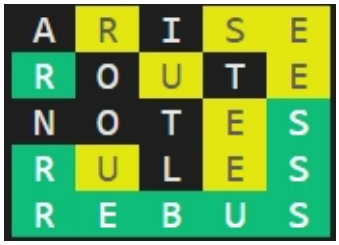


FIGURE 1 – Exemple de partie

Le jeu Kata offre différents niveaux de difficulté, et d'autres fonctionnalités avancées comme la création de compte, la mise sur pause d'une partie ou la possibilité de jouer avec des listes de mots personnalisées.

2 Organisation du projet

2.1 Phase d'analyse

Notre projet a débuté en septembre par une phase d'analyse qui s'est déroulée en trois temps : une première étape durant laquelle nous avons réfléchi aux fonctionnalités et cas d'utilisation de l'application, une seconde étape d'élaboration des différents diagrammes, puis une dernière étape de rédaction du rapport d'analyse.

La première étape sur les cas d'utilisation et les fonctionnalités ne nous a pas posé de problème particulier. En revanche, nous avons eu plus de difficultés lors de la seconde étape. En effet, au premier abord, dans le cadre de notre sujet, la création et l'utilisation d'un webservice interne ne nous a pas paru intuitive. De ce fait, nous avons eu un peu de mal à établir notre diagramme de classes. Cependant, l'élaboration de plusieurs diagrammes de séquences pour quelques fonctionnalités nous a aidé à mieux cerner le rôle du webservice dans notre application.

Pour cette phase d'analyse, en plus des séances banalisées dans l'emploi du temps, nous nous sommes réunis quatre fois à l'école. Nous avons réfléchi tous ensemble sur les fonctionnalités et les différents diagrammes. Puis nous nous sommes partagé le travail pour la mise au propre des diagrammes et la rédaction du rapport d'analyse.

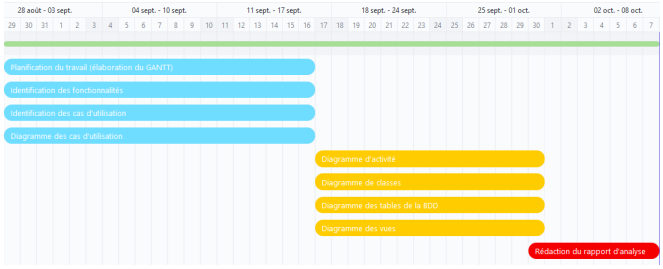


FIGURE 2 – Planning de la phase d'analyse

2.2 Phase de développement

La phase de développement a débuté en octobre. La première étape a consisté à lister les différentes classes à coder dans chaque couche. Cela a été fait lors de la séance de suivi de projet du 14 octobre. Lors de cette séance, nous avons également décidé de la répartition du code. Nous avons fait le choix d'une division du travail par couche. Ainsi, chaque membre du groupe s'est occupé prioritairement d'une couche en particulier. Cette répartition s'est faite naturellement car nous avions chacun des appétences différentes.

Ensuite, entre cette séance et la période du 26 au 28 octobre, chaque membre du groupe a relu le TP afférent à la couche dont il était responsable, et a débuté l'écriture du code de quelques classes.

Les trois journées banalisées du 26 au 28 octobre ont été consacrées à l'écriture du code. Pendant ces trois jours, nous avons eu quelques problèmes de configuration de Visual Studio Code qui nous ont légèrement ralentis. Concernant le code en lui-même, la principale difficulté que nous avons rencontrée est inhérente au fait de coder toutes les couches en parallèle alors qu'elles sont dépendantes entre elles. Par exemple, pour vérifier le fonctionnement d'une vue, il est nécessaire que les parties de code correspondantes dans les couches DAO et services soient également fonctionnelles. Or, comme tout était codé en parallèle, ce n'était pas toujours le cas. Ainsi, certaines vues ont d'abord été codées en utilisant directement les méthodes de la DAO, puis ont ensuite dû être réécrites en utilisant les méthodes de la couche services. Nous aurions peut-être pu limiter ce type de ralentissement si nous avions partagé notre code avant les 3 journées banalisées. Mais du fait du laps de temps réduit et de nos emplois du temps, cela n'a pas été possible. Cependant, malgré ces quelques ralentissements, nous avons réussi à mettre à profit ces trois journées puisque l'essentiel du code a été écrit

pendant cette période.

Par la suite, durant la dernière phase du projet en novembre, les personnes les plus à l'aise en informatique ont poursuivi et terminé l'écriture du code. En parallèle, les autres membres du groupe ont effectué des travaux de documentation, de tests unitaires du code, et initié l'écriture du rapport final. Enfin, chacun a contribué à l'écriture du rapport final.

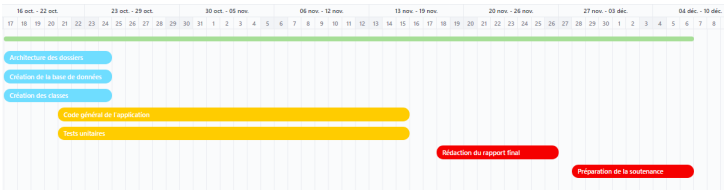


FIGURE 3 – Planning de la phase de développement

2.3 Outils utilisés

Pour faciliter le travail en groupe, nous avons utilisé plusieurs outils.

Nous avons d'abord créé un Drive pour partager les documents importants au projet tels que le planning pour organiser notre travail ou les différents diagrammes. Ainsi, tous les documents utiles étaient centralisés au même endroit et pouvaient être consultés et modifiés au fil de l'eau par tous les membres du groupe. De la même façon, nous avons utilisé Overleaf pour la rédaction des deux rapports.

Pour la phase de développement en Python, nous avons utilisé GitHub. Bien que cela nous ait demandé un coût d'entrée, nous nous sommes rapidement rendu compte de l'utilité de cet outil pour le versionnage et le partage du code. Étant déjà un peu familier avec GIT, Mathieu, le chef de projet, nous a aidés à mieux utiliser cet outil et nous a présenté le système de branches. Mis à part un problème initial avec les fichiers cache, nous avons eu très peu de conflits. Nous avons également utilisé l'outil de vérification syntaxique Pylint pour tendre vers un code "standard".

3 Fonctionnalités et fonctionnement général du jeu

3.1 Fonctionnalités du jeu

Le sujet imposait les fonctionnalités de base suivantes : jouer une partie, sauvegarder et reprendre une partie en cours, visualiser un tableau des scores général.
En plus de ces fonctionnalités de base, notre application offre la possibilité aux joueurs de créer et jouer avec des listes personnalisées de mots, de consulter leur tableau de scores personnels, ainsi qu'un choix de difficultés lors du lancement d'une partie.

Pour créer une liste de mots personnalisée, le joueur peut importer un fichier contenant sa liste de mots au format CSV ou JSON. La création d'une liste peut aussi se faire manuellement en saisissant les mots un à un directement dans l'interface.

Concernant les difficultés, le joueur peut modifier les options suivantes : le nombre de lettres du mot à deviner (6 par défaut), le nombre de tentatives pour trouver le mot à deviner (6 par défaut), le temps maximum pour faire une proposition (10 secondes par défaut), l'utilisation d'un indice (la première lettre du mot à deviner est alors dévoilée ; option activée par défaut).

3.2 Fonctionnement général du jeu et diagrammes d'activité

Le diagramme d'activité présente le fonctionnement général de l'application Kata. Par souci de lisibilité, ce diagramme a été découpé en trois parties représentant les différents écrans d'accueils, le déroulement d'une

partie de jeu et la gestion des listes personnalisées.

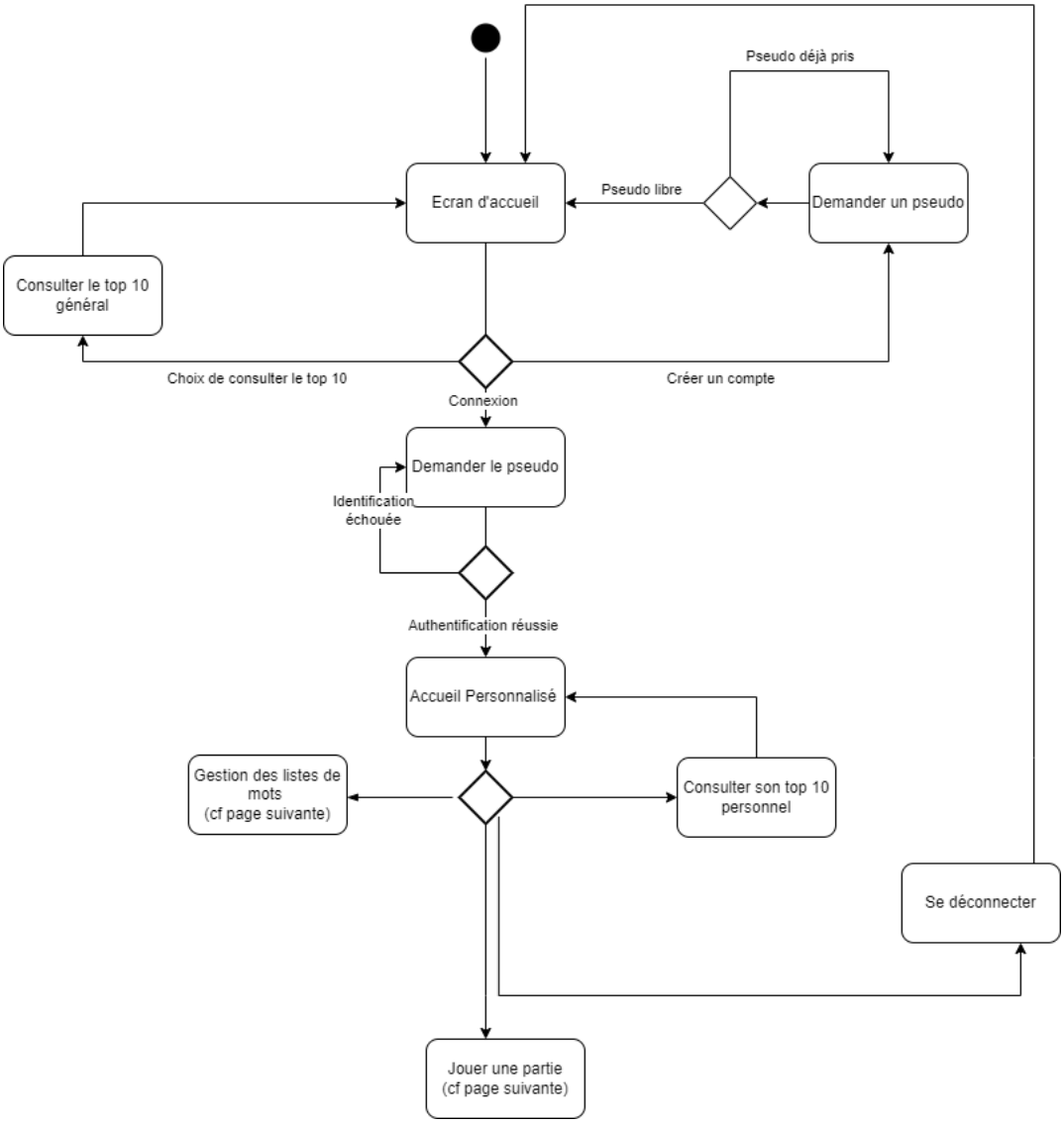


FIGURE 4 – Diagramme d'activité montrant les fonctionnalités sur les deux écrans d'accueil du jeu

Pour commencer, le jeu démarre sur un écran d'accueil général de l'application qui offre plusieurs possibilités à l'utilisateur. Il peut tout d'abord consulter les dix meilleurs scores jamais réalisés par tous les joueurs confondus : une liste des dix meilleurs scores est alors affichée puis le joueur retourne sur l'écran d'accueil. Ensuite, il peut créer un compte auquel cas il lui sera demandé un pseudo (qu'aucun autre joueur ne devra avoir choisi précédemment) pour pouvoir s'identifier. Une fois le compte créé, le joueur retourne sur l'écran d'accueil principal du jeu. La dernière possibilité sur cet écran est de pouvoir se connecter : le joueur doit rentrer son pseudo et si celui-ci existe, le joueur sera redirigé vers un écran d'accueil personnalisé. Sur ce nouvel écran, le joueur a, à nouveau, plusieurs possibilités. Le joueur peut, à l'instar de l'écran d'accueil général, consulter ses meilleurs scores. Comme précédemment, le joueur peut se connecter à sa session et c'est donc naturellement qu'il peut également se déconnecter de sa session auquel cas le joueur retourne sur l'écran d'accueil principal du jeu.

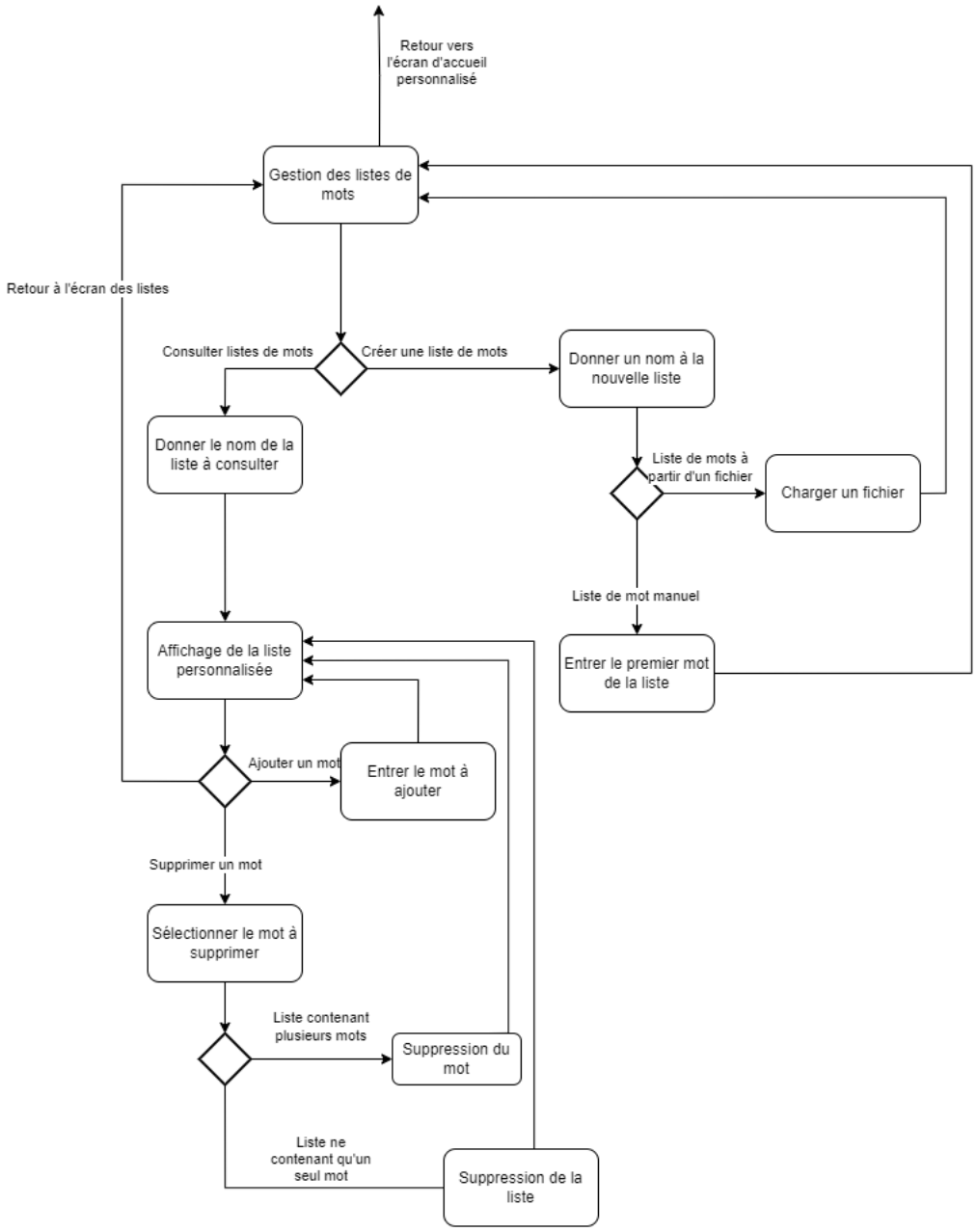


FIGURE 5 – Diagramme d'activité de la consultation, création et modification des listes personnalisées

Ensuite, chaque joueur a la possibilité de créer et consulter ses listes personnalisées. En décidant à partir du menu d'accueil personnalisé de se diriger vers les listes de mots, le joueur a alors de nouveau deux possibilités. Il peut soit créer une liste, auquel cas, il est demandé au joueur de donner un nom de liste et un premier mot dans la liste, soit de consulter ses listes. Pour consulter une liste, le joueur doit sélectionner le nom de la liste qu'il veut consulter. En sélectionnant la liste, le joueur voit s'afficher tous les mots de sa liste et trois propositions lui sont faites. Il peut retourner au menu des listes, ou il peut décider de modifier cette liste en ajoutant un mot ou en en supprimant un. Si le joueur supprime le dernier mot d'une liste de mots, cette liste est supprimée également : une liste de mots ne peut pas être vide.

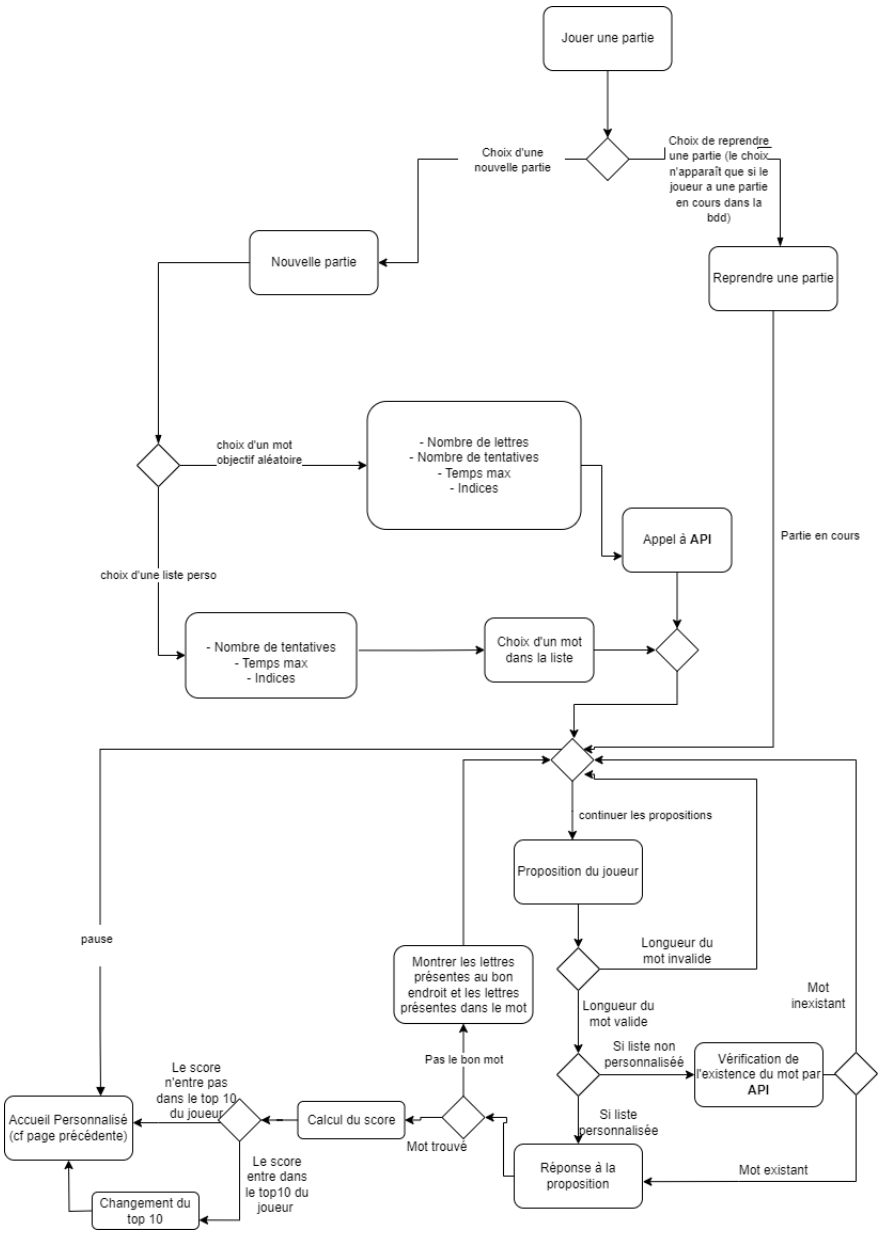


FIGURE 6 – Diagramme d'activité du déroulement d'une partie

Le joueur peut enfin évidemment jouer en lançant une partie. Il a alors le choix de reprendre sa dernière partie en cours (si elle existe) ou de démarrer une nouvelle partie. Dans le cas où le joueur lance une nouvelle partie, il doit remplir plusieurs paramètres de difficulté (longueur du mot, langue, nombre de tentatives maximales, etc) et doit choisir s'il veut jouer avec un mot d'une de ses listes personnalisées ou s'il préfère jouer avec un mot au hasard. Dans le second cas, un appel à une API extérieure est effectué pour donner un mot objectif à la partie. Une fois ces informations rentrées, la partie peut enfin commencer. Le joueur est alors invité à faire une proposition de mot. Les mots de longueur différente du mot cible ne sont pas acceptés et le joueur doit faire une autre proposition. Au contraire si le mot proposé a la même longueur que le mot à trouver, deux nouveaux cas sont possibles : soit le mot est issu d'une liste personnalisée et aucune vérification supplémentaire n'est effectuée, soit ce n'est pas le cas et notre programme fait un autre appel à une API externe pour vérifier que le mot proposé existe. Après ces vérifications, le joueur se voit retourner une réponse à sa proposition qui lui indique les lettres bien placées, les lettres en commun mais mal placées et les lettres qui ne sont pas dans le mot objectif. Si le joueur n'a pas trouvé le mot objectif, il a la possibilité de refaire une proposition dans le cas où le nombre de propositions maximal n'a pas été atteint. Au contraire,

si le joueur a découvert le bon mot, un score est alors calculé en fonction des paramètres de difficulté. Le score ainsi calculé permet de mettre à jour les scores du joueur puis le joueur retourne vers l'écran d'accueil personnalisé.

4 L'architecture de l'application

4.1 L'architecture "macro" de l'application

Le schéma ci-dessous décrit les relations entre les différents éléments de notre application.

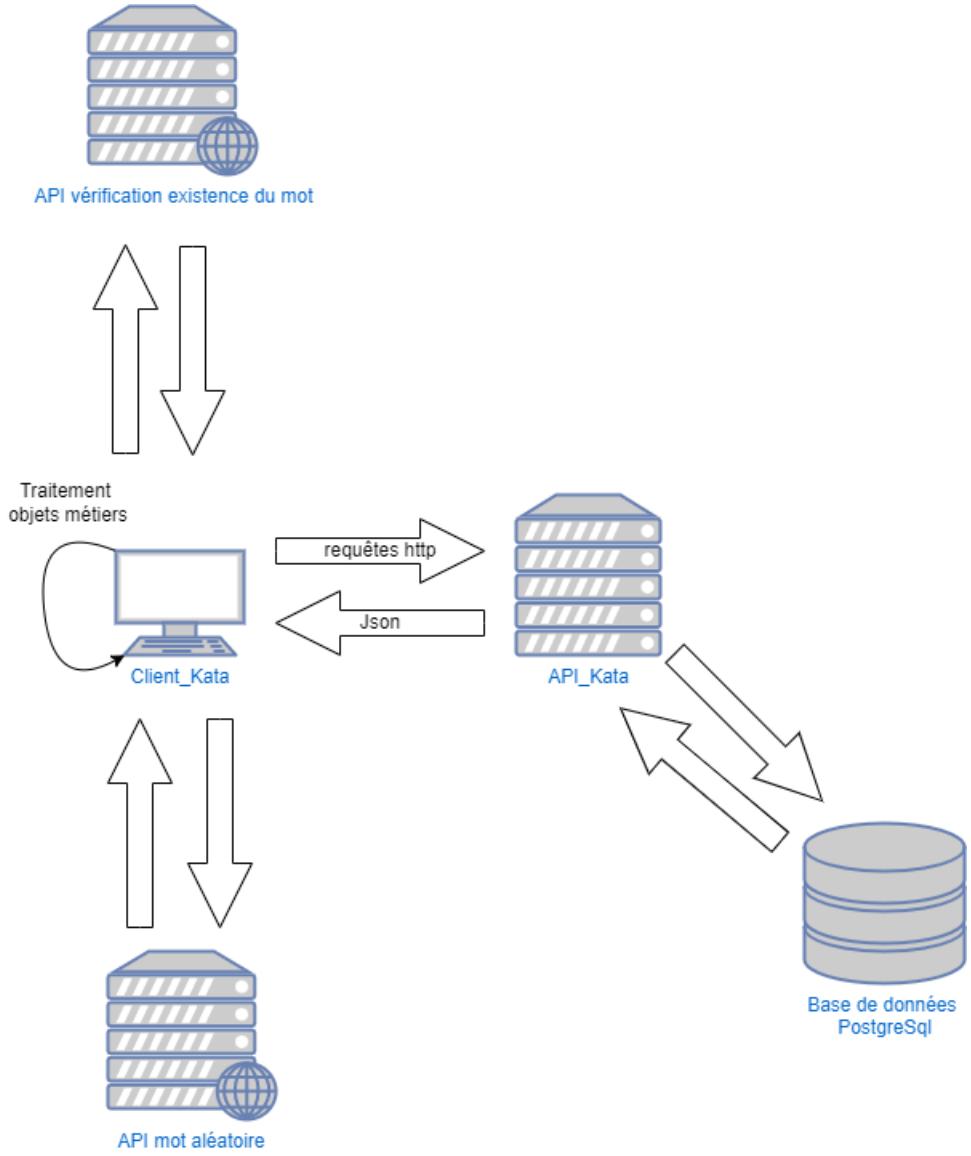
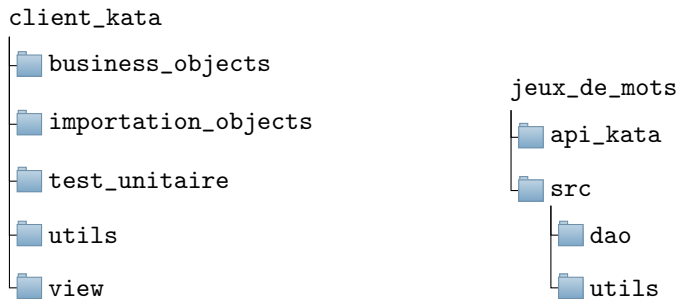


FIGURE 7 – Schéma d'architecture

Notre application est construite sur un modèle client-serveur, avec une persistance des données assurée par une base Postgresql. Nous avons recours à deux API externes pour générer des mots aléatoirement (random-word-api) et vérifier l'existence des mots proposés par le joueur (dictionaryapi). Le serveur, ou API Kata, fait office de passe-plat entre la base de données et le client. Ce dernier est du type "client lourd" et assure lui-même les traitements métier aux informations récupérées depuis l'API Kata ainsi que l'interaction avec les API externes. Ainsi, le jeu est affiché dans le client : le joueur interagit avec l'application via une

succession de menus (views) dans la console. Le client contient également du code métier avec lequel les views communiquent. Ce code métier communique avec notre webservice interne par des requêtes HTTP. Ce dernier interagit avec la base de données grâce à la couche DAO.

4.2 Packages



L'application Kata est structuré en deux dossiers de code, matérialisés par deux dépôts git différents. Le répertoire client_kata contient le code métier dans business_objects et le code pour importer des listes de mots dans importation_objects. Il contient également le dossier view qui gère l'affichage en console pour le joueur. Le dossier utils contient la classe Singleton qui est utilisée pour s'assurer qu'un objet n'a qu'une unisque instance. Le répertoire jeux_de_mots contient le code assurant le déploiement de l'API dans api_kata ainsi que l'accès à la base de données dans src. On y retrouve le dossier utils et sa classe Singleton.

4.3 L'architecture du code

4.3.1 Business Objects

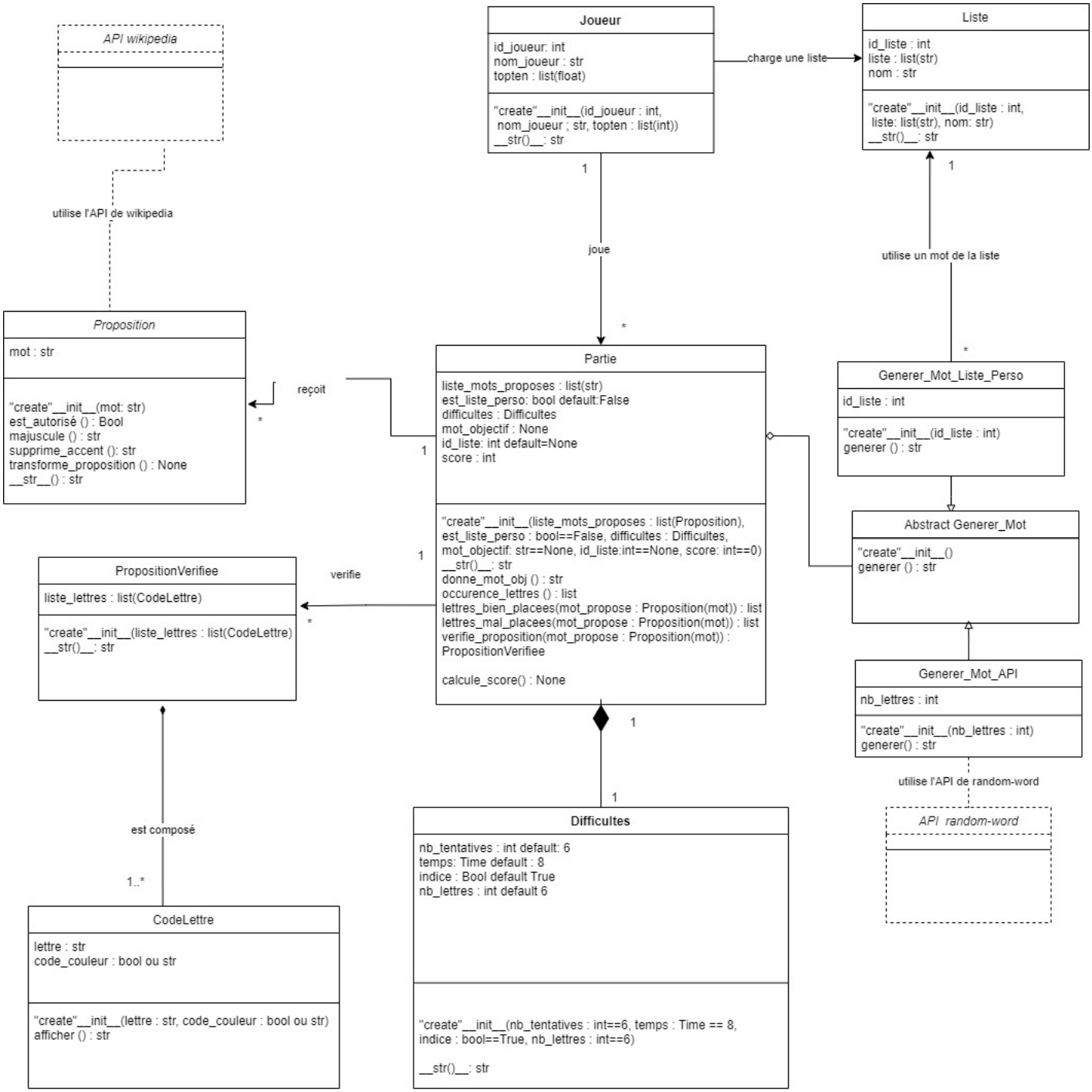


FIGURE 8 – Diagramme de classes

Partie et Difficultes Le joueur a la possibilité de jouer une partie représentée par un objet Partie. Chaque partie est caractérisée :

- le mot objectif : déterminé par la méthode generermot() (voir ci-dessous)
- la liste des mots proposés par le joueur
- par son score : nul jusqu'à ce que la partie soit terminée, il est alors calculé par la méthode calculerscore(),
- par le fait de savoir si le joueur joue avec des listes personnalisées ou pas à travers l'attribut estlisteperso et donc par la même occasion l'identifiant de liste avec laquelle le joueur joue (cette valeur vaut None si le joueur ne joue pas avec une liste personnalisée).
- La partie est également composée par un objet Difficultes qui permet de connaître les paramètres de la partie. Cet objet contient la longueur du mot (qui est de 6 par défaut), le nombre de tentatives maximum (de 6 par défaut également), la présence ou non d'un indice (connaître ou non la et le temps 1ère lettre du mot objectif), et le temps maximal autorisé pour faire une proposition (si le joueur dépasse ce temps alors il perd la possibilité de faire sa tentative et passe à la suivante).

A partir de ces informations, la méthode generermot() permet de définir un mot objectif à la partie. Cette méthode de la classe abstraite GenererMot peut avoir deux formes différentes en fonction de la valeur de l'attribut estlisteperso ce qui donne lieu à deux classes filles de GenererMot. Si estlisteperso vaut True, le mot est généré à partir de la classe GenererMotListePerso qui prend en fonction des paramètres de difficulté et de la liste sélectionnée un mot dans la liste personnalisée souhaitée. Dans le cas contraire, notre programme fait appel à l'API random-word à travers la classe GenererMotAPI. De plus, l'utilisation de cette classe abstraite permettrait de gérer l'appel à une API externe pour d'autres langues.
Cet objet Partie permet de gérer tous les paramètres choisis par le joueur quand il fait une partie.

AbstractGenererMot Pour initialiser notre jeu, on a besoin du mot objectif que le joueur cherche à touver. Ce mot peut être générer selon deux manières : la première est de le générer à partir d'une liste personnelle du joueur, la deuxième est d'utiliser l'API de random-word. Pour ce faire on crée les deux classes GenererMotListePerso et GenererMotAPI. Ces deux classes héritent de la classe AbstractGenererMot qui est une classe abstraite. La classe AbstraiteGenererMot dispose de la méthode abstraite generer() qui sera bien définie dans les classes filles. Dans le 1er cas la méthode generer() retourne un mot à partir d'une liste personnelle du joueur. Dans le 2ème cas, elle retourne un mot en faisant l'appel à l'API de random-word.

Proposition, PropositionVerifiee, CodeLettre Lorsque le joueur fait une proposition, celle ci est instanciée en objet Proposition possédant un attribut : le mot. Cela permet de normaliser automatiquement le mot en enlevant les accents éventuels, et le mettant en majuscule. La comparaison avec le mot objectif est alors facilitée car il est lui aussi en majuscule et sans accent.
Chaque proposition est d'abord vérifiée à partir de la méthode estautorisee de la classe Proposition. Cette méthode fait appel à l'API dictionaryapi pour garantir l'existence du mot proposé dans le cas où le joueur ne joue pas avec ses listes de mots personnalisés (voir partie 4.2 pour plus de détail sur cette API). La Proposition est ensuite comparée au mot objectif grâce à la fonction verifieproposition de la classe Partie, qui renvoie alors un objet de type PropositionVerifiee. La proposition du joueur est ainsi représentée par une liste d'objets CodeLettre ceux-ci ayant pour attribut une lettre et un code couleur (vert si la lettre est bien placée ; jaune si la lettre est présente mais mal placée).
L'affichage en couleur de sa proposition au joueur consiste alors simplement à afficher l'objet PropositionVerifiee. Enfin, l'objet Partie stocke la liste des propositions (en type str) du joueur pour cette partie.

Joueur Un Joueur a pour attribut :

- son identifiant
- son pseudo
- son top 10 des scores

Lorsqu'un utilisateur se connecte grâce à son pseudo à l'application, un objet Joueur est instancié et placé dans la session. Les attributs choisis pour cet objet Joueur correspond ainsi aux informations pertinentes à garder dans la session.

Liste Un joueur peut avoir ses propres listes de mots personnalisées. Une liste est caractérisée par un identifiant, un nom et une liste de mots (str). La création de ses listes peut s'opérer de deux manières

différentes : soit une importation manuelle (le joueur rentre les mots de la liste un à un), soit une importation de fichier, csv ou json.

AbstractImportationListe Chaque joueur a la possibilité d'importer une liste de mots personnalisées sous forme d'un fichier csv ou json. On gére ces importations à travers les deux classes ImportationCsv et ImportationJson qui héritent de la classe abstraite AbstractImportationListe et se basent sur la méthode creer() pour transformer un fichier csv ou json en une liste de mots.

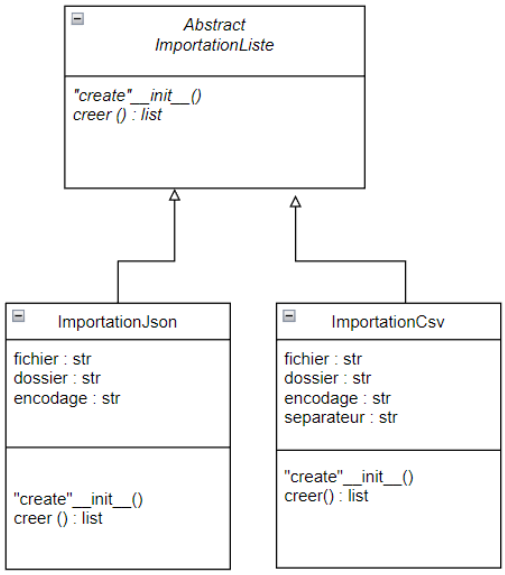


FIGURE 9 – La classe AbstractImportationListe

4.3.2 Persistance des données : base de données et couche DAO

Plusieurs fonctionnalités, notamment la sauvegarde d'une partie et la création de listes de mots personnalisées, imposent de persister des données.

Pour ce faire, nous utilisons une base de données SQL. Dans notre base de données, chaque table correspond à un type d'élément à persister : joueur, liste, mot, partie, proposition, et score. Ces différentes tables sont mises en relation via des clés étrangères signalées par des # dans le diagramme. Par exemple, une liste personnalisée doit être liée à un et un seul joueur. L'utilisation de la clef étrangère id_joueur dans la table liste permet de respecter cette contrainte.

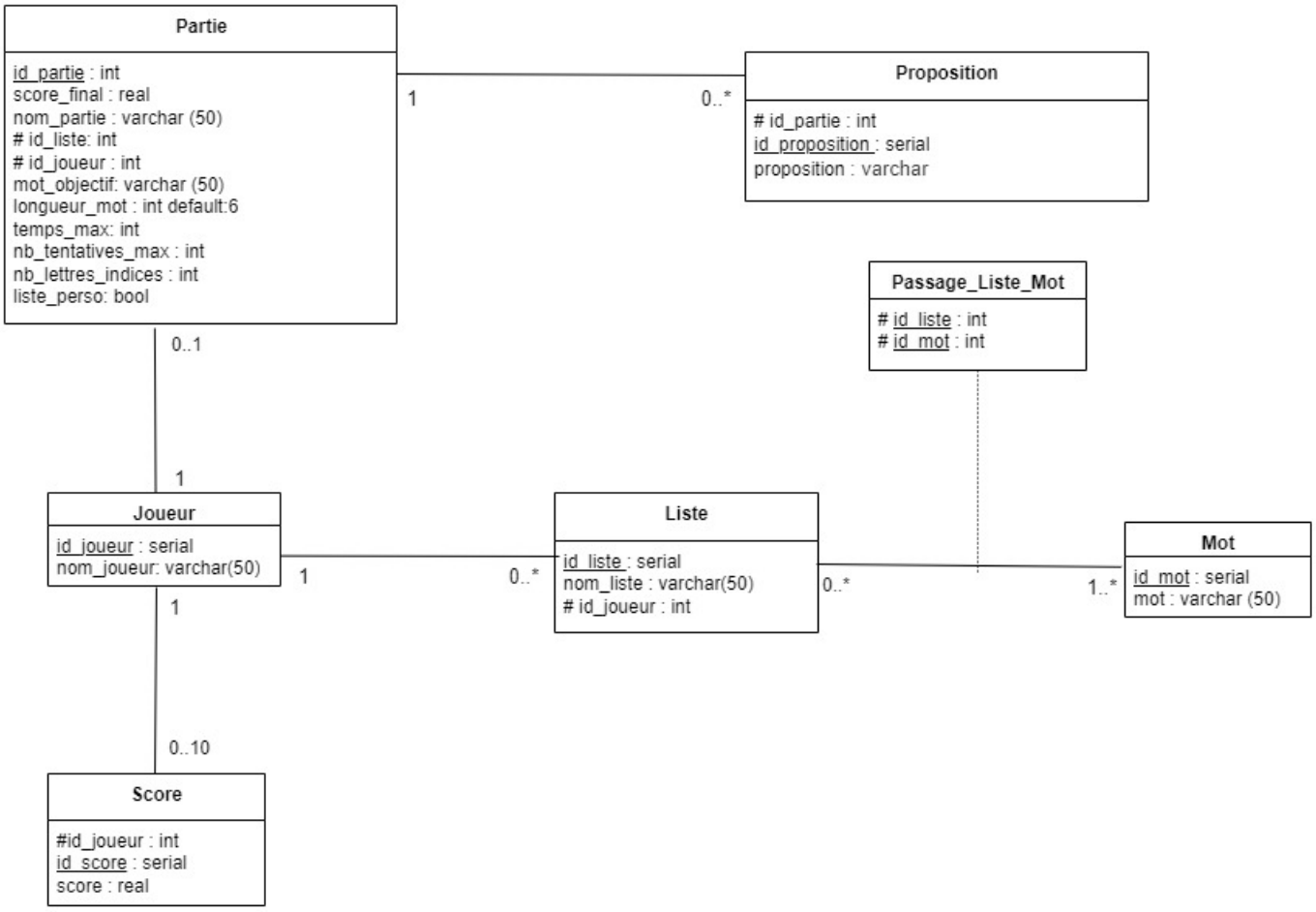


FIGURE 10 – Diagramme des tables

Nous utilisons la table d'association *passage_liste_mot* entre les tables *liste* et *mot*. Elle permet de gérer l'appartenance d'un mot à plusieurs listes. En effet, comme un mot peut appartenir à plusieurs listes, nous ne souhaitons pas que sa suppression d'une liste entraîne sa suppression de la base de données. Ainsi, grâce à l'utilisation de la table d'association, lorsque un mot est supprimé d'une liste, seul le lien entre le mot et la liste dans la table *passage_liste_mot* est supprimé, et le mot est conservé dans la base de données.

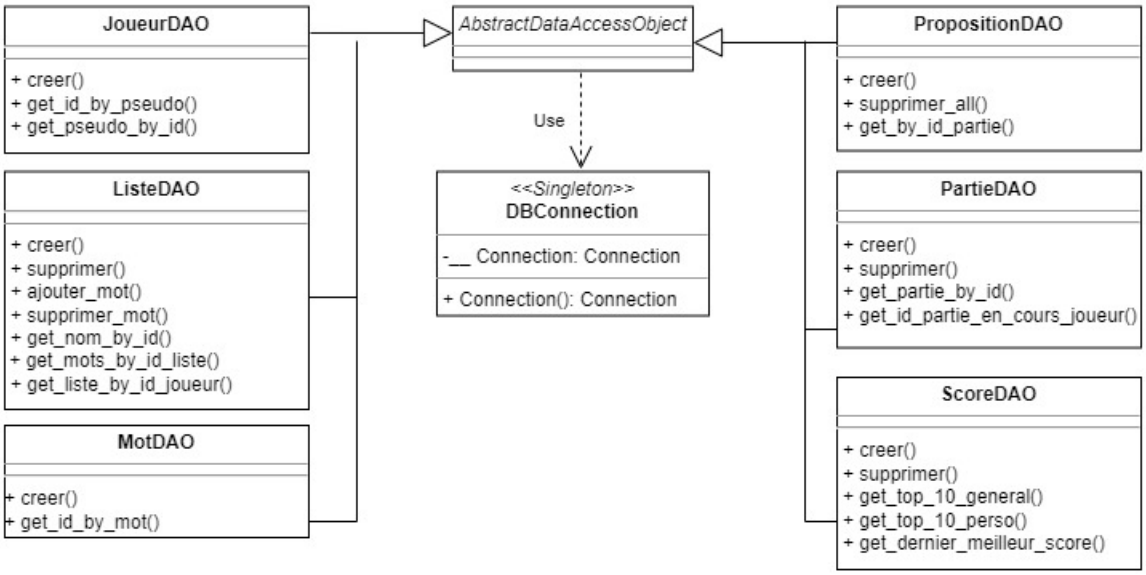


FIGURE 11 – Diagramme simplifié des classes DAO

Le lien entre la base de données et le code métier se fait grâce à la couche DAO.

DBConnexion est la classe qui permet la connexion avec la base de données. Elle est donc utilisée par toutes les autres classes. Cette classe utilise des paramètres de connexion qui sont stockés dans le fichier .env à la racine.

Les autres classes DAO contiennent des méthodes permettant les quatre opérations CRUD (Create Read Update Delete) de manipulation de données. Par exemple, les méthodes creer() persistent les données en base, les méthodes get() permettent de lire et retourner des données, et les méthodes supprimer() suppriment les données de la base.

4.3.3 Le webservice

Nous avons implémenté le webservice avec FastAPI. Pour chaque objet métier, des endpoints ont été créés. Les méthodes CRUD ont été implémentées selon les besoins du client : création d'un nouveau joueur dans la base de données lorsqu'un utilisateur veut créer un compte, récupération d'une partie en cours, suppression d'un mot dans une liste personnelle, visualisation des 10 meilleurs scores, etc.
Ainsi, le client communique par requête HTTP avec notre webservice, qui lui retourne les informations dont il a besoin. Les endpoints et méthodes utilisées sont listés ci-dessous :

FIGURE 12 – Endpoints joueur

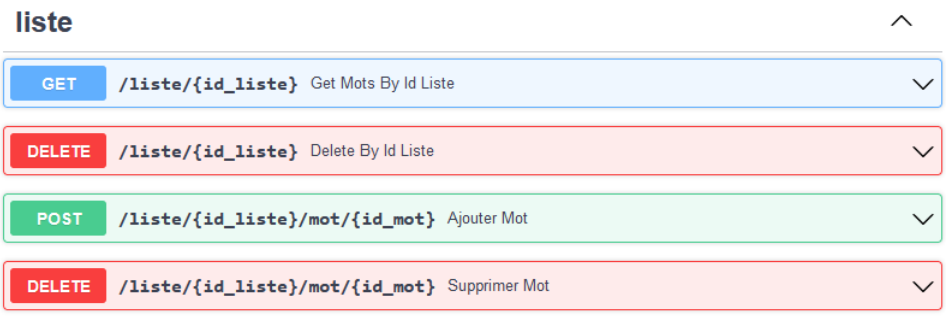


FIGURE 13 – Endpoints liste

4.3.4 Le client

Le client est la partie du code qui communique avec le webservice. Nous utilisons le package request de Python pour communiquer par des requêtes HTTP. Le client implémente toute les méthodes nécessaires au fonctionnement des vues : récupérer une partie de la bdd, création d'un compte, suppression d'un mot dans une liste, etc.

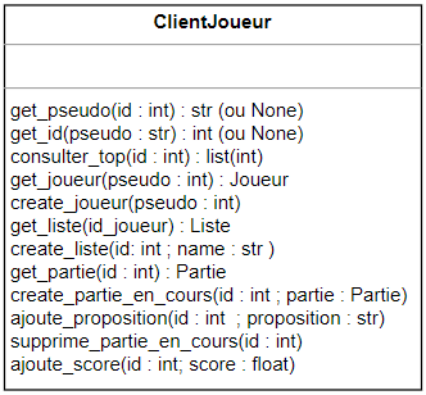


FIGURE 14 – Classe ClientJoueur

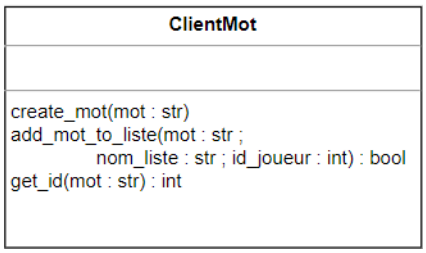


FIGURE 15 – Classe ClientMot

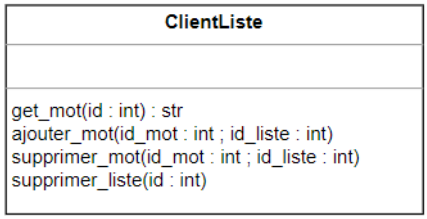


FIGURE 16 – Classe ClientListe

4.3.5 Interface de l'application : les Views

L'interaction entre l'utilisateur et notre application se fait via des vues qui s'affichent dans le terminal Python. Chaque vue correspond à un écran et constitue un module différent. Le package "view" est constitué de 20 modules. Toutes les vues héritent de la vue abstraite *AbstractView* qui contient deux méthodes abstraites : la méthode *display_info* pour la gestion de l'affichage et la méthode *make_choice* pour gérer les choix de l'utilisateur et l'envoyer sur une autre vue. Sur chaque vue, l'utilisateur doit faire un choix dans un menu ou saisir une information. Ainsi, chaque vue amène sur une autre vue. Lors du passage d'une vue à l'autre, il est nécessaire de conserver certains attributs (notamment les identifiants du joueur et de la partie) : ceux-ci sont stockés dans l'objet *session*.

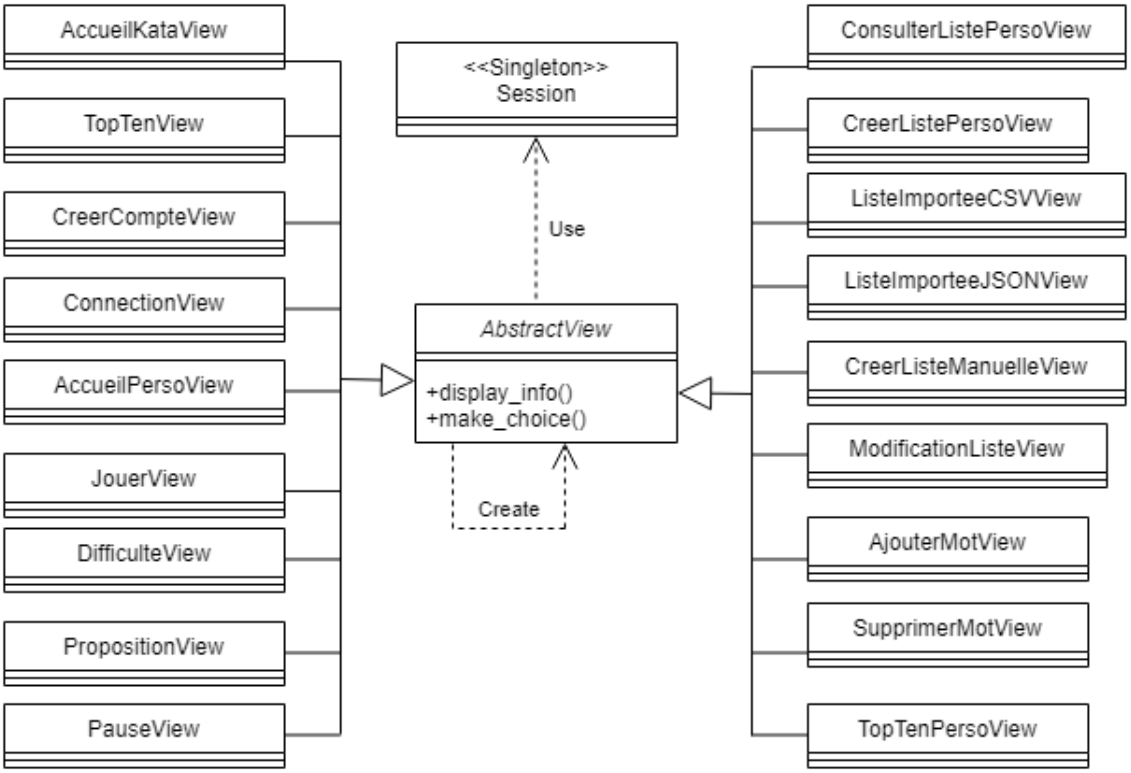


FIGURE 17 – Diagramme des vues

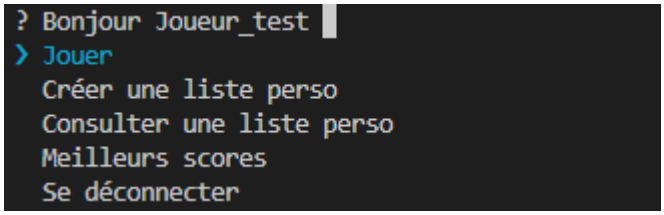
Par exemple, si un joueur veut lancer une partie de puis l'écran d'accueil principal du jeu, l'enchaînement des menus sera le suivant :

```
? Bonjour
❯ Se connecter
  Créer un compte
  Consulter les 10 meilleurs scores
  Quitter le jeu
```

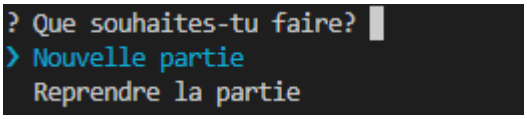
Lorsque l'utilisateur ouvre le jeu, il se trouve tout d'abord sur l'écran principal du jeu où plusieurs propositions lui sont faites comme créer un compte, se connecter ou encore consulter les dix meilleurs scores. Le joueur, en supposant qu'il ait déjà un compte choisit donc de de connecter.

```
Connexion au compte
? Quel est ton pseudo? Joueur_test
```

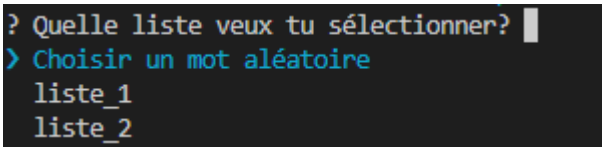
Il est donc demandé au joueur de rentrer son pseudo. Le pseudo entré ne peut contenir que des chiffres, des lettres éventuellement accentuées et des tirets underscores. La gestion de la vérification de la forme du pseudo sera expliquée dans une section prévue à cet effet. L'application va alors chercher une correspondance dans la base de données. Si la correspondance est établie, le joueur accède à son écran d'accueil personnalisé. Sinon, le joueur retourne sur l'écran d'accueil principal du jeu. En supposant que le joueur ait entré un pseudo valide :



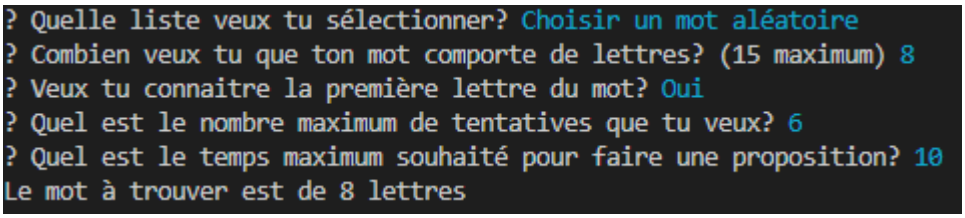
Sur ce nouvel écran, plusieurs choix sont à nouveau proposés. Le joueur peut consulter ses meilleurs scores, gérer ses listes, se déconnecter évidemment et enfin, il peut décider de jouer. Pour lancer une partie, le joueur sélectionne donc le choix "Jouer".



Le joueur fait face ensuite à un autre choix (qui n'est toutefois pas automatique). Si le joueur n'a pas terminée sa dernière partie et a décidé de la mettre en pause, le joueur a le choix entre reprendre cette partie et en commencer une nouvelle. En décidant de reprendre la partie en cours qu'il avait, le joueur est automatiquement emmené sur la View Proposition qui lui permet de jouer en faisant des propositions de mots. Si le joueur n'a pas de partie en cours, il est immédiatement redirigé vers l'écran suivant sans passer par cet écran transitoire.



Une fois que le joueur a décidé de commencer une partie, il a désormais le choix sur la provenance du mot à trouver. En effet, il peut décider de jouer avec un mot aléatoire d'un nombre de lettres fixé ou il peut décider de choisir un mot dans l'une de ses listes qu'il a la possibilité de sélectionner pour jouer. Dans ce cas, l'application choisit un mot aléatoire dans la liste sélectionnée et indique au joueur la longueur du mot à trouver.



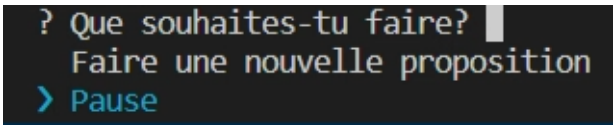
Pour finir, avant le lancement de la partie, le joueur doit choisir ses préférences pour jouer. Il peut choisir la longueur du mot (si le mot n'est pas issu d'une liste de mots personnalisée), le nombre de tentatives, le temps par proposition et de connaître la première lettre du mot à trouver. Si les informations rentrées ne sont pas conformes aux exigences, l'application fixe arbitrairement ces paramètres (6 pour la longueur du mot, 6 pour le nombre de tentatives et 10 secondes pour le temps par proposition). Une fois que toutes ces informations sont rentrées, la partie peut se lancer et le joueur peut faire des propositions.

5 Le fonctionnement du code

5.1 Zoom sur un processus central : mettre sur pause une partie et la reprendre

Une des fonctionnalités principales de notre application est la possibilité de mettre en pause et de reprendre plus tard une partie. Un joueur peut choisir à tout moment de mettre en pause une partie en cours. Celle-ci est stockée en base pour pouvoir ensuite être reprise au même endroit si le joueur le décide. Un joueur ne peut avoir qu'une seule partie en cours. Ainsi à chaque fois qu'il commence une nouvelle partie ou qu'il reprend une partie, la partie en cours dans la base de données (si elle existe) est supprimée.

5.1.1 Mise en pause d'une partie



A chaque fois que le joueur a fait une proposition (et qu'il n'a pas atteint le nombre maximum de tentatives autorisées), il lui est proposé de faire une nouvelle proposition ou bien de mettre en pause sa partie.
La mise en pause d'une partie nécessite la participation de plusieurs couches de l'application.

- Vue : la vue appelle le client.
- Client : Le client appelle la requête POST du end point joueur/id_joueur/partie du webservice : l'objet Partie de la session est envoyée au webservice sous la forme d'un json.
- Webservice : La classe BaseModel PartieCreation du webservice permet ensuite de traiter le json reçu. Le webservice appelle alors la DAO
- La DAO utilise une requête SQL INSERT INTO pour créer la partie dans la base de données.

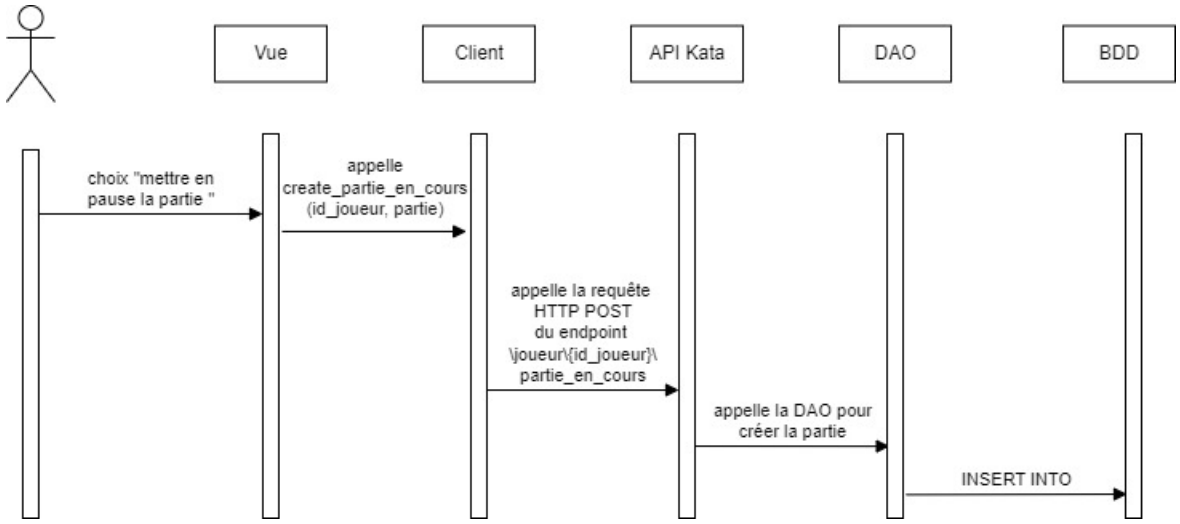
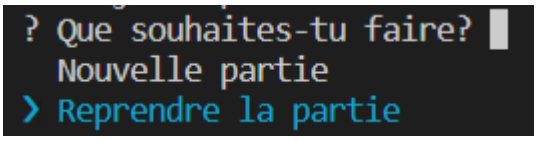


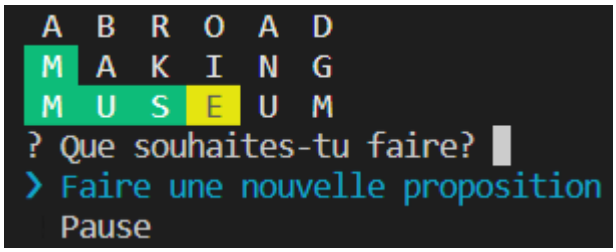
FIGURE 18 – Diagramme de séquence de la sauvegarde d'une partie

5.1.2 Reprendre une partie en cours



Pour jouer une partie deux choix s'imposent à l'utilisateur : commencer une nouvelle partie ou continuer une

partie en cours (s'il en a une).
La reprise d'une partie nécessite ici encore la participation des différentes couches. Le client appelle la requête GET du endpoint du webservice, avec comme paramètre l'identifiant du joueur (pour rappel, un joueur a au plus une partie en cours stockée en base). Le webservice contacte alors la DAO qui lui renvoie les paramètres de la partie (mot objectif, difficultés (nombre de tentatives, temps maximal par tentative), propositions déjà faites par le joueur) après une requête SQL SELECT. Ces paramètres sont alors récupérés par le client qui crée un objet Partie. La vue affiche alors au joueur les propositions qu'il a déjà faites et lui permet de faire une nouvelle proposition :



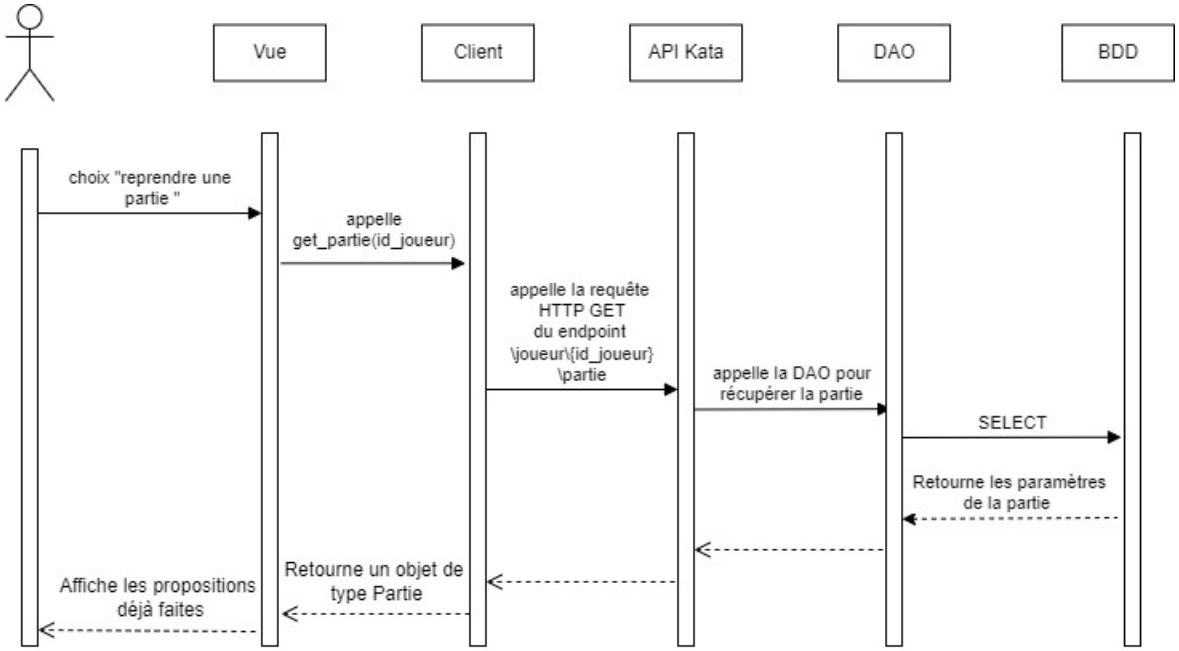


FIGURE 19 – Diagramme de séquence de la reprise d'une partie en cours

5.2 Autres processus importants

Utilisations d'API externes : Notre application utilise par deux fois des API externes par l'intermédiaire de requêtes HTTP.
Nous utilisons ainsi d'abord l'API random-word-api. A l'instanciation de la Partie, plusieurs choix de difficultés sont proposés au joueur, dont le choix de la provenance du mot objectif (tiré d'une liste personnelle ou choisi aléatoirement). Ainsi, l'API nous permet de renvoyer un mot d'une certaine longueur (en anglais) lorsque le joueur choisit de faire une partie avec un mot objectif aléatoire.
La deuxième API que nous utilisons est API dictionaryapi. Cette API permet de vérifier qu'un mot existe dans le dictionnaire. Nous nous servons de cette API à deux reprises : pour vérifier si le mot objectif, dans le cas où il est aléatoire existe (si ce n'est pas le cas on tire un nouveau mot objectif), et pour vérifier si les

propositions du joueur existent (ainsi si le joueur propose un mot qui n'est pas autorisé, la proposition ne sera pas vérifiée).

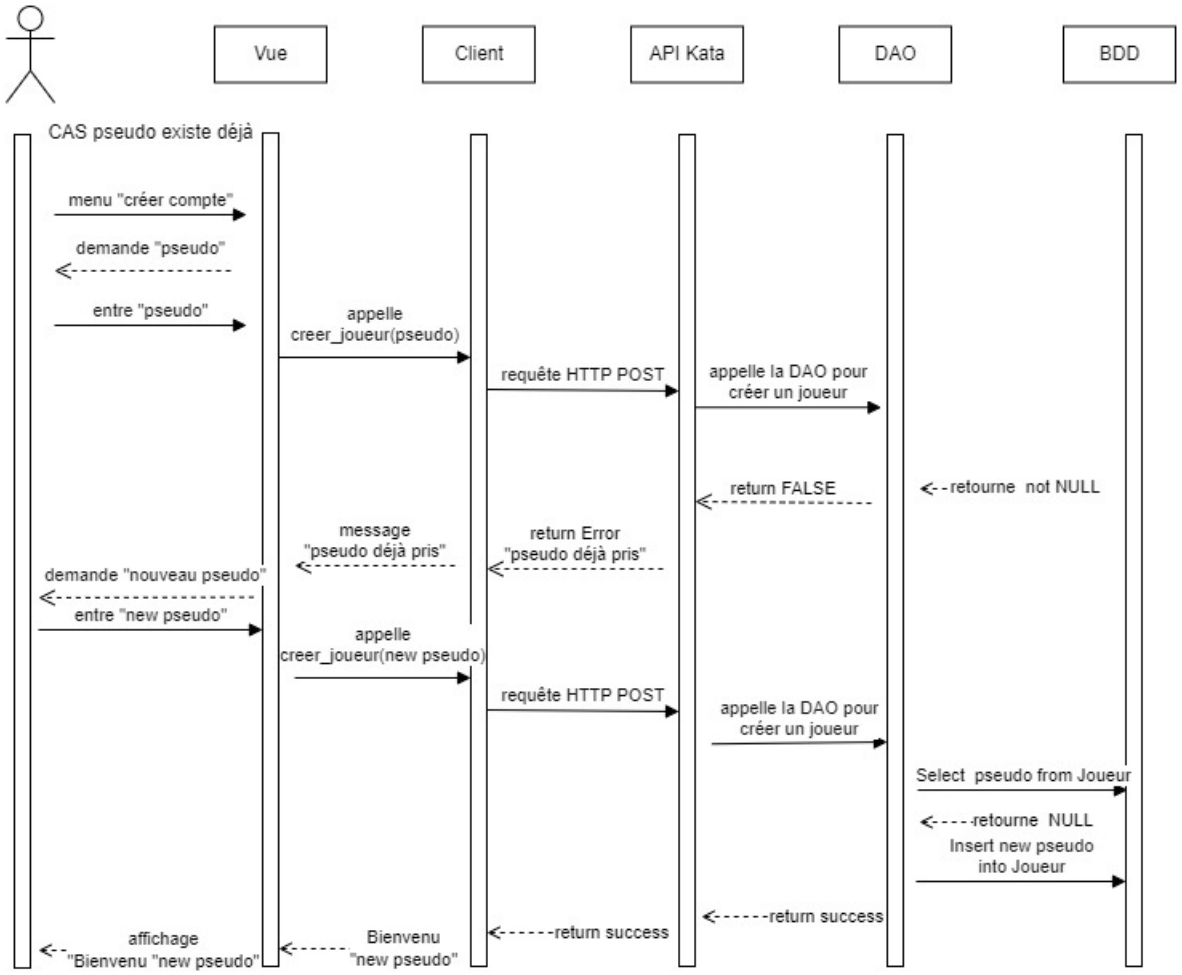


FIGURE 20 – Diagramme de séquence de création d'un compte

Création d'un compte Lorsqu'un utilisateur souhaite créer un compte, il lui est demandé d'entrer un pseudo. Ce pseudo doit être unique dans la base de données. Nous allons nous placer dans le cas où le pseudo initialement choisi par l'utilisateur existe déjà dans la table Joueurs de la BDD. Après avoir été invité à entrer son pseudo dans le terminal, le client lance une première requête POST à l'API. La couche DAO demande à la BDD une requête de vérification de l'existence de ce pseudo dans la base. Puisque ce pseudo existe déjà, le résultat de cette requête n'est pas NULL, ce qui déclenche un retour en cascade jusqu'au client qui affiche à l'utilisateur un message lui indiquant que le pseudo est déjà pris et qu'il lui en faut entrer un autre. Le nouveau pseudo retraverse ainsi les différentes couches jusqu'à la BDD qui vérifie si le pseudo existe ou non. Si ce nouveau pseudo n'existe pas, la DAO va déclencher une requête pour créer une nouvelle ligne dans la table Joueur.

Jouer avec une liste personnalisée : Le joueur peut créer ses propres listes de mots avec lesquelles jouer. La création d'une liste peut se faire manuellement (il entre les mots un par un) ou en important directement un fichier CSV ou JSON. Cela lui est proposé directement après qu'il se soit connecté. La liste créée est alors sauvegardée en base, en passant par les mêmes étapes que pour la création d'une partie. Si l'utilisateur veut ajouter une liste de mots à partie d'un fichier, il lui sera demandé d'entrer le chemin de son fichier dans le terminal. Le client va envoyer une requête HTTP POST à l'API Kata pour déclencher la méthode de création d'une liste. Dans l'API Kata, la couche DAO a la responsabilité de contacter la BDD pour créer une nouvelle

liste. La BDD renvoie l'id_liste à la DAO au moment de la création de la liste.
La DAO retourne cet id_liste à l'API Kata qui s'en sert pour appeler la DAO pour chaque mot de la liste. Un appel à la BDD est fait pour associer un id_mot à l'id_liste dans la table de passage et éventuellement créer au préalable un nouveau mot dans la table de mots. Aucun retour n'est fait au client. L'affichage d'une liste personnalisée se fait lors d'une autre interaction.

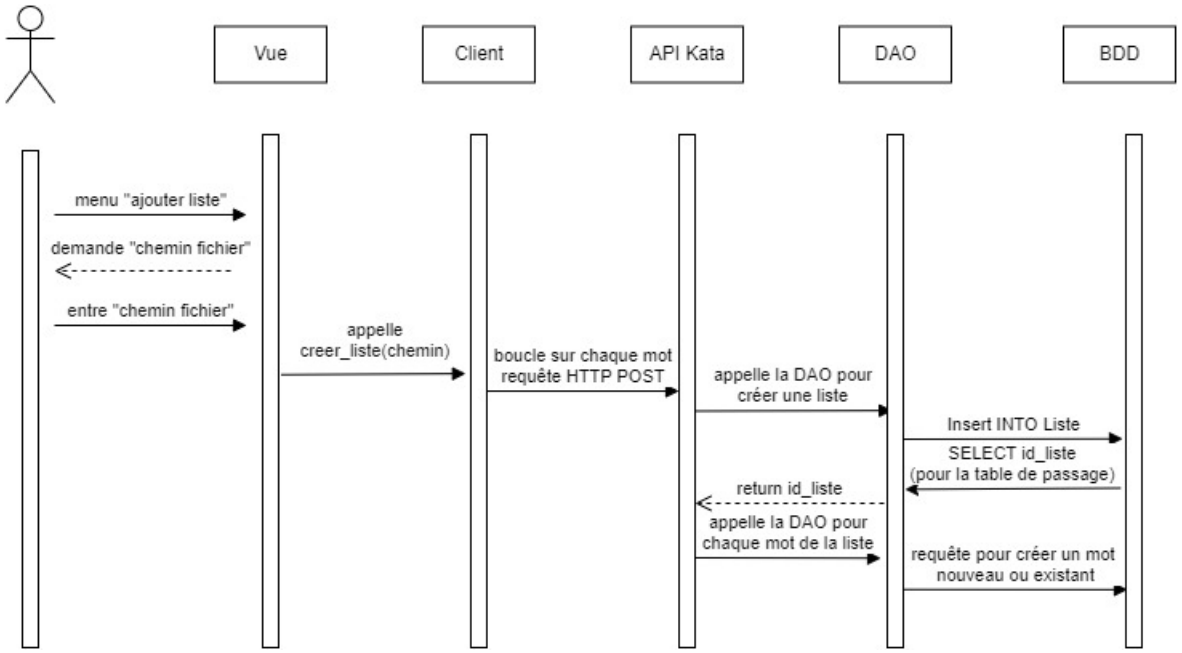


FIGURE 21 – Diagramme de séquence d'ajout d'une liste personnalisée

Vérification des caractères dans les inputs Pour éviter de faire planter l'application, nous avons décidé de vérifier chaque expression que le joueur peut entrer dans la console et qui peut ensuite être testée par l'application ou envoyée dans la base de données. Cela permet d'éviter des joueurs mal intentionnés de faire des injections SQL dans le but de s'attribuer de meilleurs scores ou supprimer des tables.

Pour l'application, le pseudo du joueur et le nom des listes ne doivent contenir que des lettres (éventuellement accentuées), des chiffres et des undescores. Les propositions, quant à elles, ne peuvent contenir que des lettres éventuellement accentuées. et pour finir le choix des difficultés (nombre de tentatives, temps par proposition et longueur du mot) ne peuvent contenir que des entiers.

La vérification des inputs se fait avec des expressions régulières. Si l'input n'est pas validé par l'expression régulière, l'input n'est pas traitée dans l'application et un message d'erreur est affiché à l'écran pour prévenir l'utilisateur.

5.3 Tests unitaires

Pour s'assurer du bon fonctionnement de certaines parties du code, nous avons réalisé des tests unitaires en parallèle de l'écriture du code sur les classes suivantes : Joueur, Import, Difficultes, Liste, Partie et Proposition.

Pour ces tests, nous avons utilisé le package *unittest*. Le dossier contenant les modules de test est placé à la racine du client et nous avons créé un module pour chaque classe testée. Toutes les classes de test héritent de la classe TestCase du module unittest. Et tous les tests suivent la structure GIVEN/WHEN/THEN.

```python
def test_est_autorise(self) :
    """ Test de la méthode : est_autorise() qui vérifie si la proposition existe dans le dictionnaire
    par l'intermédiaire de l'API dictionaryapi
    """

    #GIVEN
    mot1 = "uhujhbj"
    mot2 = "Sky"

    #WHEN
    prop1 = Proposition(mot1)
    prop2 = Proposition(mot2)

    #WTHEN
    self.assertEqual(False, prop1.est_autorise())
    self.assertEqual(True, prop2.est_autorise())
```

FIGURE 22 – Exemple : test de la méthode est_autorise de la classe Proposition

6 Conclusion et pistes d'amélioration

Ce projet a été l'occasion d'appliquer nos connaissances dans la réalisation d'un projet concret. L'application Kata a ainsi pu voir le jour en trois mois. Elle répond au cahier des charges, en présentant quelques fonctionnalités supplémentaires. On peut toutefois noter quelques pistes d'amélioration.

-Les mots proposés de manières aléatoire sont pour l'instant en anglais par l'API. Il serait possible d'utiliser la même API pour obtenir d'autres langues. Cela prendrait alors la forme d'un nouvel endpoint /langue et d'un menu de choix de la langue pour l'utilisateur. Cependant, le français ne fait pas partie des langues fournies par cette API. Si nous voulions utiliser le français, il faudrait alors trouver une autre API de génération de mot aléatoire et réfléchir à l'abstraction des requêtes utilisées afin de ne pas dupliquer le code.

-Kata est un jeu très simple, et ne conserve dans les top10 que les scores les plus élevés. Il n'est pas possible de faire disparaître le score d'un joueur autrement qu'en jouant et en ayant un score plus élevé. Cependant, un joueur malicieux pourrait utiliser le pseudo d'un autre pour le faire. Il s'agirait alors d'introduire un système d'authentification par mot de passe.

-La manière dont nous gérons le SQL peut être améliorée à l'aide du package sqlmodel. Ce dernier a été créé par le même auteur que FastAPI dans le but de faciliter la gestion des bases de données dans FastAPI. Cependant, sa découverte tardive ne nous a pas laissé le temps de l'implémenter en refactorant le code.

-Lors du jeu, nous appliquons des traitements aux propositions du joueur, notamment pour enlever les accents. Nous faisons cela à l'aide d'un dictionnaire pré-rempli. Cependant, il serait possible de rendre le dictionnaire des accents "unicode-ready".

-Nous pourrions gérer, par des blocs TRY-EXCEPT, les appels du client à l'API. Cela nous permettrait de traiter d'éventuels messages d'erreur de la part de l'API. Ce point est complémentaire d'une meilleure documentation de l'API lors de la définition des endpoints. Cet axe serait vraiment à privilégier pour un déploiement de l'application sur internet afin qu'un développeur puisse plus facilement comprendre les services fournis.

-Cela nous amène à un dernier point d'architecture. Nous avons réalisé une application de type client lourd / serveur. En effet, les objets métiers sont traités par le client et l'API renvoie les données nécessaires à la construction de ces objets métiers, et non des objets métiers eux-mêmes. Cela alourdit notre client, sans pour autant nuire à l'application. Nous avions pourtant décidé de le faire, mais cela nous a échappé durant le stade de réalisation.

7 Notes individuelles

7.1 Mathis

Pour ce projet, nous avons décidé de répartir les tâches assez rapidement entre les 5 membres du groupe pour que les choses soient très claires dès le début.

Pour ma part, je me suis occupé principalement de faire le diagramme d'activité pendant la phase d'analyse du sujet et de compléter quelques sections du rapport d'analyse. Ensuite, une fois que nous avons commencé à coder, j'ai codé toutes les vues du jeu pour permettre un bel affichage dans la console. J'ai aussi géré toute la partie de vérification des entrées du joueur dans la console à l'aide d'expressions régulières pour éviter que le joueur puisse faire planter notre application. Cela permet de plus d'éviter qu'un joueur décide de faire des modifications de notre code ou de notre base de données.

Nous avons essayé de travailler un maximum sur les plages horaires dédiées au projet mais il a également fallu qu'on travaille sur ce projet en dehors de ces horaires et nous avons donc essayé un maximum de travailler ensemble car cela permet de résoudre les problèmes rapidement. En effet, pour la construction des vues, j'avais constamment besoin d'échanger avec les autres membres du groupe pour connaître le nom des fonctions dont j'avais besoin, pour savoir ce que renvoie chaque fonction, etc. Il était donc plus facile de travailler ensemble qu'être chacun de son côté sur le sujet. Les trois jours dédiés au code avant les vacances de la Toussaint ont été très bénéfiques pour construire l'architecture globale du code. J'ai eu l'impression de découvrir le travail en équipe comme je pourrais travailler ensuite dans une entreprise dans le monde du travail. La majeure partie du code a donc pu être faite avant les vacances et c'était très pratique si on voulait coder un peu après car la structure était présente.

En ce qui concerne le rapport, je ne suis pas celui qui en a fait le plus mais j'ai écrit le descriptif du diagramme d'activité ainsi que la gestion des entrées et le déroulement des vues. C'était plus simple que je l'écrive étant donné que c'est moi qui ait fait ces choses.

Ce projet m'a permis évidemment de développer mes compétences de code en Python mais aussi des compétences plus relatives à l'organisation et au travail de groupe. J'ai trouvé le travail en groupe très enrichissant et aussi très efficace même si comme souvent, nous avons perdu pas mal de temps sur des petits problèmes qui nous empêcher de pouvoir avancer le code comme des packages que l'on n'arrivait pas à installer ou des problèmes avec Git. Je trouve cependant que malgré des débuts un peu hasardeux qui sont dus pour beaucoup au fait que nous n'avions jamais utilisé Git, l'utilisation de Git nous a été au final bénéfique car elle a largement contribué à la transmission du code d'une façon simple entre les membres du groupe.

7.2 Oussama Marhfoul

Personnellement, j'avais toujours une ambition de découvrir le processus de création d'un jeu. Et grâce à ce projet, j'ai eu la chance de participer à la conception et à l'implémentation du jeu intitulé ‘’jeu de mots”.

La première phase était la modélisation en UML des diagrammes principaux qui constituent le corps et la base de notre travail, à savoir : le diagramme d'activité, le diagramme de classes, etc. C'était un travail collectif qui m'a permis d'améliorer la manière dont je dois approcher une problématique et aussi de découvrir les idées de mes collègues afin de les biens discutées et en sortir une modélisation qui résume notre vision en tant que groupe à propos de la structure du jeu.

Durant la deuxième phase, j'ai effectué plusieurs tâches afin de constituer le code global du jeu. J'ai implémenté une partie des classes figurant dans le diagramme de classes. Je me suis occupé de réaliser les tests unitaires nécessaires pour s'assurer du bon fonctionnement des méthodes des classes principales selon la structure GIVEN WHEN THEN. Finalement, j'ai documenté un ensemble de classes pour faciliter l'identification des différents attributs et méthodes, et alors de faciliter la compréhension des différentes parties de code.

En ce qui concerne les connaissances acquises le long de ce projet, je peux citer la maîtrise de l'outil GIT qui renforce et facilite le travail en groupe. De plus, j'ai découvert le pouvoir des branches dans GIT à travers

leurs utilisations pour gérer l'implémentation du code. D'une autre part, j'arrive à comprendre l'importance et la puissance des APIs, d'avoir des idées sur la génération et la gestion des vues sur le terminal, et aussi de découvrir la DAO comme un outil permettant l'accès à notre base de données et d'extraire les informations nécessaires à travers des requêtes SQL.

En résumé, ce projet fut très enrichissant en me permettant de découvrir plus sur le monde de l'informatique et aussi de travailler au sein d'une équipe motivée.

7.3 Apolline Guérineau

Rôle dans l'équipe et tâches réalisées Je me suis surtout consacrée à l'implémentation du code, notamment les parties de création du webservice, l'appel à ce webservice par le client et la création des objets métiers. J'ai eu une bonne vision générale sur le projet, j'ai bien compris comment les différentes couches et structures s'emboitaient. J'ai ainsi pu aider mon groupe à se répartir les différentes tâches et à maintenir un ensemble cohérent et efficace. J'ai travaillé en parallèle surtout avec Mathis qui a lui implémenté les vues, car nos rôles se rejoignaient.

- Création du webservice : Nous avions défini au préalable les principaux endpoints nécessaires. J'ai ainsi complété ces endpoints en fonction de nos besoins et je les ai implementés. J'aurais souhaité que notre webservice puisse fonctionner en ligne et pas seulement en local mais je n'ai pas trouvé de solution.
- Appel au webservice : J'ai d'abord implémenté les objets métiers. Cela m'a demandé une certaine réflexion car nous n'avions pas réfléchi à tout, comme par exemple comment traiter en pratique l'affichage en couleur des propositions du joueur ou encore le fait que certains attributs ou méthodes étaient manquants dans le dossier d'analyse ou bien superflus.
  Dans les fichiers clients j'ai ensuite pu faire appel au webservice créé en retournant des objets métiers : objet de type Partie, Liste, etc.
  Les fonctions du client pouvait ainsi être réutilisées directement dans les vues, selon le besoin.

Retour personnel sur le projet J'ai beaucoup aimé participer à ce projet qui à été très enrichissant et instructif pour ma part. J'ai aussi beaucoup aimé le sujet, qui était formateur tout en restant ludique.
J'ai cependant trouvé le travail en équipe de 5 un peu difficile. L'organisation est plus lourde, plus compliquée ce qui ralentit beaucoup l'avancement du projet. Se mettre d'accord est aussi plus difficile. J'ai eu un temps d'adaptation à cette façon de travailler, même si j'avais déjà travaillé en équipe. Malgré la division en couche indépendante il s'est avéré relativement difficile de travailler à plusieurs. En effet, certaines couches sont très liées les unes avec les autres et ont demandé un travail conjoint, comme la création du webservice et la dao. Travailler à plusieurs sur les mêmes fichiers, surtout lorsque nous étions à distance étaient alors plutôt compliqué et chronophage.
Je retiens quand même du travail en groupe de 5 un enrichissement dans ma manière de travailler et d'aborder le projet. Il m'a fallu par exemple apprendre à mieux prioriser les tâches importantes, en commençant par la réalisation des tâches dont les autres membres de l'équipe avaient besoin pour eux-mêmes pouvoir avancer dans leur travail. J'ai aussi tendance à vouloir tout prendre en main en ayant du mal à déléguer, ce qui est impossible dans un groupe de 5. Le projet m'a donc aussi fait travailler sur ce point là.

Conclusion Avec le recul, je pense que notre organisation au début du projet était très efficace. Nous nous voyions assez régulièrement, en dehors des heures dédiées au projet, ce qui nous a permis d'avancer assez vite sur plein de problématiques, comme les diagrammes de classes, d'activité ou les principales fonctions que nous souhaitions implémentées dans le jeu. Cela nous a ensuite beaucoup servi après le rendu du rapport d'analyse pour implémenter le code, c'était une base solide qui a été d'une grande aide.
Je pense cependant que pour la suite nous aurions dû mieux nous répartir les tâches et mieux anticiper la rédaction du rapport, en le rédigeant petit à petit et en mettant à jour les diagrammes au fur et à mesure de nos modifications dans le code.

7.4 Linh-Da Dinh

N'ayant pas de fortes compétences en informatique, j'appréhendais un peu ce projet. Finalement, je l'ai personnellement bien vécu pour plusieurs raisons. Tout d'abord, en tant que fan de feu Motus, j'étais tout

particulièrement intéressée par le sujet. Ensuite, la bonne entente et organisation au sein de notre groupe nous ont permis de mener ce projet sans accroc majeur. Enfin, ce projet m'a permis d'approfondir mes connaissances sur deux choses que j'avais déjà utilisées dans ma vie professionnelle sans vraiment en comprendre le fonctionnement : GIT et les webservices.

Comme indiqué dans le rapport, nous avons réfléchi collectivement lors de la phase d'analyse puis la répartition des tâches pour la phase de développement s'est faite assez naturellement selon les appétences et compétences de chaque membre du groupe. Ainsi, même si par exemple, certains membres du groupe (Apolline et Mathis) ont plus codé que les autres, chaque membre du groupe a participé de manière effective à l'ensemble des travaux (analyse, code et rédaction). Ayant vécu des projets où ce n'était pas forcément le cas, j'ai trouvé ce point particulièrement positif.

Pour ma part, en plus de ma participation aux travaux d'analyse et de rédaction, je me suis concentrée sur la couche DAO pour la partie développement. Je me suis proposée pour travailler sur cette couche car c'est la couche qui me semblait la plus simple à réaliser. Globalement, je n'ai pas rencontré trop de difficultés à coder cette couche en suivant ce qui avait été fait lors du TP. Et lorsque j'ai eu des difficultés, soit de configuration, soit de code, j'ai pu compter sur l'aide rapide des autres membres du groupe plus compétents sur ces sujets car nous avons souvent travaillé en présentiel tous ensemble à l'école.

Selon moi, un autre point positif de notre organisation a été notre capacité à prioriser les travaux. Concrètement, lors de phase de développement, plutôt que de se lancer dans le développement de toutes les fonctionnalités que nous avions listées dans notre rapport d'analyse, nous avons décidé dans un premier temps de nous concentrer sur celles qui nous semblaient les plus essentielles. Dans ce genre de projet, ne pas perdre de vue ce qui est exigé par le cahier des charges me semble primordial car on peut vite tomber dans l'écueil de vouloir trop en faire alors que l'objectif principal reste de répondre au cahier des charges.

Concernant les difficultés, j'ai trouvé que la phase d'analyse n'était pas aisée, notamment parce que cela demandait d'utiliser des concepts dont nous avions peu, voire jamais, entendu parler. Peut-être que cette phase de conception aurait été plus simple si elle avait débuté plus tardivement, après qu'on ait suivi plusieurs cours et TP de compléments d'informatique. Enfin, de mon point de vue, une autre difficulté a été de gérer le conflit entre les examens (quatre sur cette période) avec la phase de développement et de rédaction du rapport final.

7.5 Mathieu

Ce projet a été l'occasion d'y vivre beaucoup de surprises et s'est révélé enrichissant. Avec la phase d'analyse, j'ai pu me rendre compte qu'aucun d'entre nous n'était significativement plus avancé que les autres, sans pour autant que quelqu'un soit complètement derrière. Cela s'est révélé d'une très grande force tout au long du projet. En effet, la pression des deadlines et des autres matières, les contraintes de temps nous ont amené à travailler très régulièrement sur une ou deux séances par semaine. Sans vraiment savoir où aller, nos réflexions devant un tableau blanc nous ont permis de réfléchir collectivement et d'aboutir à un plan raisonnable pour la phase de développement.

Durant cette période, chacun a eu à coeur de prendre à charge un aspect particulier du projet : test, DAO, affichage et code métier. 4 parties pour 5 développeurs ? En effet, j'ai assuré une forme de "liant". Il m'a fallu permettre, le plus rapidement possible, de transmettre aux autres l'utilisation de git à plusieurs et mettre en place une ébauche d'organisation du travail. Je note que l'utilisation des projets Git pour éviter de tous travailler en même temps sur un fichier a été vite abandonnée pour plusieurs raisons. La première est que chacun s'est vite spécialisé dans une couche de l'application. La deuxième est que la pression du temps n'est pas propice à un tel outil. Nous n'avions littéralement pas le temps de nous coordonner avec cet outil, alors qu'il est une base pour le développement dans le monde du logiciel libre. Les premières journées d'immersion ont été intenses. Cependant, très vite l'outil a été collectivement maîtrisé et les bugs de configuration/conflits à résoudre de moins en moins nombreux. Puisque les rôles avaient été distribués, j'ai cherché à conserver une vision globale du projet en aidant par-ci par-là.

Une chose que je voulais faire mais que j'ai abandonné en voyant le temps qu'il m'aurait fallu était de dockeriser l'application pour la déployer sur le datalab. Cependant, il aurait été impossible de l'utiliser à plusieurs, notamment à cause de la frontière entre espaces personnels.

J'ai également assuré la relecture du code à travers la notation Pylint. Ceci a été bien plus long que je ne le pensais. En effet, au-delà d'y avoir découvert de meilleures manières de coder certains points, j'y ai passé entre 15 et 20h. Il aurait été plus efficace, à mon avis, de s'organiser autrement. Par exemple, au lieu de se lancer dans sa propre partie de code et de s'y spécialiser, il aurait mieux valu documenter petit à petit, que ce soit en préparant de la docstring, utile pour les tests unitaires, qu'en lançant pylint au fur et à mesure. Ainsi, je n'aurais pas passé autant de temps sur pylint et j'aurais pu mettre en place une découverte de chaque technologie, où chaque spécialiste aurait supervisé le développement d'une petite classe ou méthode par les autres. Cela aurait assuré une meilleure montée en compétences pour tous, et donc une meilleure vision globale de l'application par tout le groupe.

Néanmoins, j'ai eu beaucoup de plaisir à venir réfléchir et coder avec le groupe. Chacun s'est montré très efficace dans ses tâches. Et je fus très heureux de voir les réactions lors de la résolution de bugs compliqués, ou lorsque l'application s'est mise à fonctionner. Tout en restant bien à sa place, ce projet restera donc un bon souvenir.