TELECOM Nancy

Coding Week

TelecomNancy FlashCard

CAPELLI Louis-Vincent
CHENEVIÈRE Thibault
CLÉRIOT Louis
SERRAND Coralie

Responsable de module :
Gerald Oster


UNIVERSITÉ
DE LORRAINE

Table des matières

Chapitre 1

Gestion de projet

Nous avons commencé la semaine de projet le lundi soir par la lecture du sujet et par une réflexion personnelle sur la façon d'implémenter le projet.
Une réunion à été programmé le mardi matin pour faire part de nos réflexion et pour commencer l'élaboration du diagramme de classe UML.
Pour la suite du projet, nous nous sommes organisés de la façon suivante : deux réunions par jour, une le matin à 8h et une le midi à 14h. Entre ces réunions, une session de programmation personnelle était effectuée le matin, et une en groupe pour l'après-midi. Ces réunions nous ont permis de faire des points régulièrement, d'élaborer des to-do listes et de se recentrer sur ce qui est important pour ne rien oublier et éviter l'effet tunnel.
Vous pouvez retrouver ci dessous les TODO liste réalisées :

TODO Mardi			
LV	Louis	Coralie	Thibault
rédiger l'UML debut mvc	réaliser card réaliser deck	Création de deck view accueil et pop up creation	mise en place du git configuration de maven

TODO Mercredi			
LV	Louis	Coralie	Thibault
Back mvc strategie	stats réaliser deck	Création de carte continuer les v	mise en place du git configuratiiewon de maven

Chapitre 2

Gestion du dépôt gitlab

Pour ce projet nous n'avions pas d'architecture pré-faite comme la plupart de nos projet. Nous avons donc du configurer le dépôt gitlab pour avoir un environement de travail identique.
Nous avons d'abord choisi une architecture Maven que nous avons générer directement à partir de l'archetype javafx donnée par Maven. Cependant, à la fin du projet nous avons trouvé cela plus utile pour la génération de notre archive executable d'utiliser Gradle. C'est pourquoi nous avons convertit notre projet Maven en Gradle (fait automatiquement par la fonction *gradle init*).
Au cours de ce projet nous avons eu à gerer différentes librairies et leur version ce qui passe par une bonne gestion des dépendances et plugins Maven.

Chapitre 3

Fonctionnalités

Toutes les pages de l'application sont composées d'une barre verticale gauche visible lorsqu'on clique sur le bouton Menu. Ce menu permet de naviguer entre la page d'accueil, la page des statistiques et la page des paramètres.

3.1 Accueil

La page d'accueil permet de visualiser les différents paquets de cartes avec leur noms descriptions et tags si présents. Il est aussi possible de supprimer un paquet en cliquant sur le bouton "Supprimer" puis une des petite croix placée à côté d'un paquet. Deux boutons "Import" et "Export" permettent d'importer et d'exporter des paquets sous des formats ".zip". Notons aussi qu'il est possible d'importer des paquets de cartes sous le format .apkg de Anki. L'application se charge de rendre les paquets sous un format accepté.

3.2 Paquets de cartes

Le bouton "Ajouter" de la page d'accueil ouvre une Pop-Up permettant la création d'un nouveau paquet. Des champs nom et description sont requis pour valider . Des tags peuvent aussi être ajoutés.
En double-cliquant sur un des paquets de la page d'accueil, on peut accéder à une page dédiée à l'édition du paquet de cartes. Cette page permet une vue globale des cartes du paquet. Le bouton "Éditer" permet de passer la page dans un mode d'édition. Il est alors possible pour l'utilisateur de modifier le nom et la description de son paquet ainsi que d'éditer ses cartes. Plus précisément, il peut modifier la question et la réponse d'une carte mais il peut aussi ajouter un média si celle-ci n'en possédait pas encore. Si la carte possède déjà un média, on peut le supprimer ou bien le modifier. Lorsqu'on visualise le paquet de cartes, on peut voir les cartes possédant un média via une petite icône. En cliquant dessus, le média est visualisé. L'utilisateur peut aussi supprimer des cartes de son paquet dans le mode édition.

3.3 Cartes

Le bouton "Ajouter" ouvre une pop-up permettant la création d'une carte. Une question et une réponse sont demandées. Il est aussi possible d'ajouter un média. Celui-ci peut être une image, une vidéo ou un audio. Nous avons aussi implémenter des fonctionnalités supplémentaires telles que la génération de réponse. Il suffit de rentrer une question dans le champ dédié puis de cliquer sur "Generate Answer". Pour la générer, on envoie une requête POST a l'api d'openai pour le model Davinci avec la question, l'api nous renvoie alors la réponse. Il est également possible d'éditer la réponse avant de valider la création de la carte. On peut aussi choisir de générer des cartes entières à partir d'un deck. En effet En se basant sur les questions déjà poséés, nous pouvons en générer de nouvelles. Cependant, les requêtes à l'api sont limité à un certain nombre de token. Si les question sont trop nombreuses, le nombre de token est alors dépasé.

3.4 Paramètres

Notre application permet à l'utilisateur de configurer deux choses différentes. Le style de l'application et l'algorithme de tirage de cartes utilisé dans le mode apprentissage de l'application.
Pour le style de l'application nous avons le choix entre deux thèmes : le thème classique composé de carte blanche et le thème TelecomNancy avec des cartes oranges et le texte violet.
Nous pouvons également choisir trois types de tirages de cartes différents.

- Le mode Classique : les cartes s'affichent dans l'ordre de création. Le mode Aléatoire : les cartes se suivent de manière aléatoire.
- Le mode Poids : On pondère la probabilité d'apparition d'une carte en fonction des bonnes et mauvaises réponses de l'utilisateur.

3.5 Apprentissage

Depuis la page d'un paquet de carte, un bouton Apprentissage permet à l'utilisateur d'ouvrir un menu pour sélectionner son mode d'apprentissage. Trois modes sont disponibles :

- Apprendre tout : c'est un mode qui permet à l'utilisateur d'apprendre toutes les cartes d'un paquet. Une carte est considérée apprise lorsque l'utilisateur répond correctement 3 fois à la question.
- Limité par le temps : L'utilisateur définie à l'avance dans un champ, le nombre de minutes qu'il veut que sa session dure. La session s'arrête lorsque le temps est écoulé.
- Limité par un nombre de cartes. L'utilisateur choisit le nombre de cartes sur lequel il veut se tester.

Au début d'une session d'apprentissage, l'utilisateur est confronté au question d'une carte. Lorsqu'il pense avoir la réponse, il peut retourner la carte via un bouton "Retourner. Il voit alors la bonne réponse. Il peut s'auto-évaluer grâce à différents boutons. Notons qu'il peut aussi choisir de revoir la question. L'auto-évaluation d'une carte implique le tirage d'une nouvelle carte suivant un algorithme choisi dans les paramètres. A la fin d'une session, l'utilisateur est renvoyé sur la page du paquet de carte. De plus une pop-up affiche les statistiques de la session.

3.6 Statistiques

On peut retrouver des statistiques à différents endroits de l'application. D'une part dans l'onglet qui leur est dédié accessible via le Menu à gauche. D'autre part, à a fin de chaque apprentissage, une pop-up regroupant les statisiques apparaît pour résumer la session. Pour représenter les différents graphiques utilisés dans le projet, nous avons utilisé la librairie de javafx chart.

3.6.1 Menu dédié

On retrouvera l'ensemble des statiques gloables et par paquet de cartes. Pour les statistiques globales, quatres graphes sont affichés, 1 bubble chart, 1 bar chart, 1 line chart et 1 pie chart. On représente donc aussi bien des données numériques, que temporelles ou catégoriques. On représente ainsi, l'évolution du nombre de paquet présent dans l'application dans le temps, le pourcentage de bonnes réponses par paquet et le pourcentage de temps passé sur chacun des decks. Finalement le graphique le plus compliqué est en trois dimensions. Sur l'axe des abcsisse, on retrouve le temps passé sur chaque paquet, en en ordonnée, le pourcentage de bonnes réponses du paquet et enfin l'ai représente le nombre de carte du deck. On retrouve en suite en descendant la pages, une

liste des decks présent dans l'application avec associé a chacun, 2 graphique différents. Le diagramme camembert représente le pourcentage de bonne et mauvaise réponse au sein du deck, le diagramme bâton, représente lui le pourcentage de bonne réponse pour chaque tag de carte présente dans le deck.

3.6.2 Fin d'un apprentissage

A la fin d'une session, un petit récapitulatif est donné. On pourra retrouvé le temps de la session, le nombre de carte vue, le nombre de bonne réponse données. On représente également sous forme de graphique camembert le pourcentage de bonne et mauvaise réponse. On peut également retrouver l'évolution du temps passé par carte tout au long de l'apprentissage.

Chapitre 4

Conception

4.1 Modèles

4.1.1 DeckList

L'application possède un unique DeckListModel. Il contient à l'initialisation un paquet de démonstration et on peut en ajouter d'autres via l'interface utilisateur. Il est aussi possible d'en importer et d'en exporter (cf. Gestion des fichiers).

4.1.2 Deck

Un DeckModel est constitué d'une liste de CardModel qui représentent chacun une carte. Il a un nom et une description et on peut aussi lui ajouter des tags, afin de faire des statistiques sur les taux de bonnes réponses en fonction des tags, etc.

4.2 Card

Un CardModel représente une carte. Elle a un id, une question et une réponse et peut contenir un média. Elle possède aussi une probabilité, qui réflète sont niveau relatif d'apprentissage au sein de son paquet et donc sa probabilité relative d'être soumise à l'utilisateur.

4.3 Tags

Les tags sont des mots-clés qui peuvent être attribués aux cartes et aux paquets. CardTag et DeckTag sont des classes séparées, bien qu'identiques, afin d'avoir un attribut statique séparés qui sauvegarde la liste des tags déjà attribuées à des paquets et à des cartes. Le but était de pouvoir afficher des suggestions à l'utilisateur mais cette fonctionnalité n'a pas été implémentée par manque de temps.

4.4 Stats

DeckListModel, DeckModel et CardModel sont la base de l'application : ils représentent les cartes crées et importées par l'utilisateur. Chacun des modèles est associé à un objet "Stat" qui suit la progression de l'utilisateur et ses habitudes. Elles sont mises à jour lors de l'apprentissage, à chaque fois qu'une nouvelle carte est tirée (cf. Apprentissage). Les objets Stats mesurent le nombre de fois qu'une carte est vue, le taux de bonnes réponses sur celle-ci, le temps mis par l'utilisateur à y répondre ainsi que la date de création et de dernière ouverture d'un paquet et le temps passé à l'étudier. Ces statistiques peuvent être consultée dans la vue dédiée sous forme de graphiques. (cf. Statistiques)

4.5 Learning

L'apprentissage s'occupe de soumettre les cartes successives à l'utilisateur lors d'une session. Il est initialisé selon les réglages choisi par l'utilisateur avec une thème visuel et une manière de lui soumettre les cartes (cf.Paramètres). Un apprentissage peut avoir plusieurs format qui implémentent une même classe abstraite Learning, tout ce qui diffère est la condition d'arrêt de l'apprentissage dont le choix est laissé à l'utilisateur (cf. Apprentissage). Il met à jour les statistiques de la cartes tirées et tiens à jour un objet StatLearning qui suit la progression de l'utilisateur lors d'une session.

4.6 Gestion des fichiers

Nous avons implémenté la fonctionnalité d'import et d'export de paquets de cartes. Pour ce faire on utilise un format propre à l'application mais nous avons aussi une fonctionnalité pour importer des fichiers au format .apkg (format Anki).

4.6.1 Format *.tlf*

Notre application utilise le format *.tlf* pour importer et exporter des decks. Ce format n'est autre qu'un *.zip* contenant les informations nécessaires au deck. On retrouve dans le sous-dossier *decks* un fichier *.json* qui contient l'architecture de la pile de carte, c'est a dire ses cartes et les différentes statistiques sur cette pile. Si il y a des médias liés à certaines cartes (image, son ou vidéo) ces médias sont rangés dans un sous-dossier adéquat (images, sounds ou videos).
Pour ce qui est de l'exportation, on rassemble les différents fichiers nécessaire dans un *.tlf* que l'on peut charger après.

4.6.2 Prise en charge du format *.apkg*

Notre application implémente la prise en charge de piles de cartes au format *.apkg*. C'est à dire que l'on peut importer des piles de cartes sous ce format et les formater au format *.tlf*. Étant donné l'implémentation de notre application, notre format de cartes et de piles de cartes ne nous permet pas d'avoir autant de détails que les cartes d'Anki, c'est pourquoi on a fait le choix de ne garder qu'un média sur le recto d'une carte au format Anki. De plus, notre application ne permettant pas d'afficher de média sur le verso d'une carte, certaines questions importés d'un fichier *.apkg* ne possède pas de réponse étant donnée que celle-ci peut être sous forme d'une image.

Chapitre 5

Tests

Dans ce projet nous avons implémentés et utilisé des tests unitaires. Nous avons utilisés les test unitaires pour pouvoir vérifier les fonctionnalités implémentées au fur et a mesure. En effet, avec la répartition du travail nous ne pouvions pas toujours tester directement nos nouvelles fonctionnalités sur l'application. C'est pourquoi nous sommes passé écrit des test unitaires, en particulier pour les modèles et la gestion des fichiers.
Pour ces test, nous avons utilisé la librairie *junit.jupiter* qui permet la création de test unitaire facilement et permet de les executer directement avec gradle.