PLD SMART H4213

Dossier synthèse

Clémentine Coquio-Lebresne, Tifenn Floch, Anatolii Gasiuk, Matthieu Halunka, Christophe Hirt, Alan Paugois

RECITO

Motivation

Afin de faciliter l'apprentissage de texte, *Recito* a été réalisé. Cette application a pour objectif, de signaler les erreurs lors de la récitation d'un texte importé au préalable. L'application donnera un score par rapport aux erreurs réalisées lors de la performance. Ce dernier permettra à l'utilisateur de voir où se situaient ses erreurs, pour pouvoir ensuite s'en souvenir.

Afin d'avoir une application complète et intéressante pour tout le monde, l'étape finale de notre application serait de la faire évoluer afin de permettre la création de groupes de personnes travaillant sur un texte commun (ex : sur le texte d'une pièce de théâtre).

Cas d'utilisation

Cas apprentissage d'un texte

Roger est un étudiant en fac de lettres et dans sa discipline, il se doit d'apprendre des textes pour un partiel. Afin d'optimiser son temps de révision, Roger utilise l'application *Recito*. Pour ce faire, il lance l'application directement sur son smartphone et importe son texte. Une fois importé, il appuie sur le bouton "enregistrement", puis commence à réciter son cours en essayant de faire le moins de fautes possible. Une fois terminé, il clique sur "terminer l'enregistrement" pour pouvoir voir son score. Puis en cliquant sur "détails", il pourra voir les fautes qu'il a pu faire. Après une bonne séance de révision, Roger est prêt pour son partiel.

Cas écoute du texte

Roger souhaiterait écouter son cours préféré, cependant le texte est très long. Pour ce faire, il décide de lancer *Recito* et d'importer le texte complet. Puis, d'une simple pression du bouton "*play*". Son cours est lu, ce qui lui permet en parallèle d'écouter, comprendre et faire ses fiches.

Cas apprentissage d'une langue

Rodriguez, un étudiant chinois souhaiterait vraiment visiter Paris cet été. Mais malgré les nombreux cours, il n'arrive pas à se débarrasser de son accent qu'il trouve ridicule. Pour pallier cela, il lance son application *Recito*, importe un texte en langue française puis grâce à un clic sur le bouton "*start*", commence à écouter l'application qui récite le texte avec la bonne prononciation. En mettant sur pause, il peut reproduire la prononciation entendue plus tôt. Et voilà, Rodriguez est prêt à aller à Paris.

Choix technique

Azure

Pour le bon fonctionnement de notre application, nous avons décidé d'utiliser Azure qui est une plateforme en ligne située dans le *cloud*. Cette décision a été principalement motivée par deux points :

- l'ensemble des outils nécessaires à la réalisation de notre application étaient déjà disponibles sur cette plateforme ;
- les outils équivalents et leurs plateformes, notamment proposés par Google et Amazon, ne nous proposaient pas suffisamment de crédits gratuits pour pouvoir les utiliser et mener à bien le développement de *Recito*.

Un exemple d'outil que nous utilisons est l'API *Speech-To-Text* qui nous permet de récupérer et traiter ce que l'utilisateur récite.

Nous profitons aussi de l'outil base de données (Azure Cosmos DB) afin d'avoir une persistance des données. Par ailleurs, l'outil *App Services* nous permet de déployer simplement notre back-end sur un serveur Azure et ainsi d'avoir une URL, et une IP publique, pour accéder à notre back-end sans rencontrer de blocage entre notre application et notre base de données.

NB : Nous avons pu constater que les blocages rencontrés, avec la connexion vers nos bases de données, durant le développement de notre application étaient dû à la DSI qui bloque les ports de connexion à nos bases MongoDB sur le réseau de l'INSA.

Dans le but d'avoir des métriques sur la qualité de notre code et d'avoir la certitude que les modifications apportées ne provoquent pas de régression, nous avons décidé de mettre en place des machines virtuelles Azure pour utiliser les outils d'intégration continue tels que Sonar et Jenkins.

Front-end

Android Studio

La communauté de développement Android étant en croissance constante et étant donné que Eclipse est de moins en moins utilisé dans le développement mobile sur la plateforme Android, nous avons choisi d'utiliser Android Studio pour développer la partie front-end. Cet IDE est destiné à la création d'applications Android. Il est facile dans son installation et dérive de JetBrains - un outil que vous avons déjà utilisé et qui permet une assimilation plus rapide à l'environnement de programmation. De plus ce logiciel est très courant dans le monde professionnel et nous offre la possibilité d'acquérir de l'expérience pour les futures missions de stage dans ce domaine.

OkHttp3

Cette librairie nous permet de réaliser des requêtes HTTP en direction du back-end afin de réaliser les différentes interactions avec notre base de données. L'intérêt de cette librairie réside dans le fait qu'elle est disponible aussi bien pour l'environnement Android que l'environnement Java classique et que son utilisation reste intuitive et permet une prise en main rapide de ses fonctionnalités.

Microsoft Cognitive Services Speech (*Speech-To-Text*)

Nous avons choisi d'utiliser Microsoft Cognitive Services Speech, afin de transformer la voix de l'utilisateur en texte. Ce service étant le même que celui utilisé pour Cortana, il est extrêmement performant.

Java Speech API

N'ayant pas la possibilité d'utiliser Microsoft Cognitive Services Speech pour transformer du texte en voix, nous avons utilisé l'API Java Speech, qui nous permet de synthétiser une voix facilement. Malheureusement, la voix n'est pas très agréable à écouter.

Diff Utils library

Diff Utils est une librairie Java *open source*, qui nous permet de comparer deux textes et d'indiquer quels mots ont été ajoutés et supprimés, d'un texte à l'autre.

Back-end

Spring boot

Afin de faire la liaison entre front-end et base de données, un serveur a été mis en place. Ce dernier constitue la partie métier de l'application. C'est sur ce dernier que les processus de récupération et de traitement des données seront situés et exécutés en fonction de ce que l'IHM Android demande. Nous avons choisi cette technologie afin d'avoir d'une part, un *framework* qui nous donnait de nombreux outils clés en main dont notamment Jackson et JUnit, et d'autre part, d'avoir un langage commun à l'ensemble du projet.

Il aurait été possible d'utiliser NodeJS ou PHP pour réaliser le serveur avec notamment des *frameworks* tel que Laravel mais nous nous serions retrouvés avec une seule personne de l'hexanôme maîtrisant cette technologie.

Azure Cosmo DB

Sachant que nous avions prévu de stocker les textes sur la base de données pour éviter de devoir faire une importation à chaque fois, nous avons choisi de nous tourner vers une base NoSQL puisque nous avons besoin de stocker des grands volumes de textes ainsi que, possiblement, des données plus exotiques telles que des enregistrements audio. Pour ce faire via Azure, nous avons choisi d'utiliser une base mongoDB via le service Azure ayant pour nom Cosmos DB.

Détail d'implémentation

Organisation du projet

Nous avons décidé de fonctionner en Agile en établissant deux sprints.

Sprint 1

La première étape du sprint était d'établir un squelette de l'application, réaliser la connexion entre les différents composants, afin de pouvoir itérer par la suite pour ajouter des fonctionnalités en partant d'une base fonctionnelle.

Les tâches réalisées pendant ce dernier étaient :

- Réaliser le flux de données entre Front-end et Back-end
- Concevoir la base de données
- Concevoir les IHM
- Mettre en place les outils d'intégration continue Sonar, Github et Jenkins
- Avoir une version 0 de l'application utilisant l'API Microsoft pour le Speech-To-Text

Sprint 2

La seconde étape était de compléter l'application par l'ajout de fonctionnalités secondaires/non-essentielles vis-à-vis des fonctions principales de *Recito*. Par exemple, ces fonctionnalités secondaires correspondent à la présence d'un compte utilisateur, d'une librairie personnelle ou encore d'un score en fonction de la comparaison de texte.

Fonctionnalités

- Pouvoir comparer des paroles par rapport à un texte de référence
- Réciter un texte donné afin de mémoriser le texte et d'obtenir un score associé à la correspondance entre le texte récité et le texte donné.
- Sauvegarder les textes importés dans une bibliothèque propre à chaque utilisateur.
- Sauvegarder et mettre à jour le score obtenu.
- Choisir les parties de texte qu'on souhaite travailler. L'application omettra le reste du texte qui n'est pas surligné.

Difficultés rencontrées

Azure

Une des premières difficultés rencontrées pendant la réalisation du projet était l'utilisation d'Azure. En effet, malgré la présence d'un grand nombre de tutoriels sur Internet, le

déploiement du serveur pour le back-end nous a réservé de nombreuses difficultés dûes à un plugin Maven difficile à mettre en place proprement et une documentation floue sur la configuration des différents modules à utiliser.

De plus, l'explorateur de coûts nous a parfois réservé des surprises telles que la consommation non-notifiée de 160 € sur 170 € de crédit sur le compte hébergeant le serveur SonarQube et le back-end à la suite de trois requêtes à l'API *Detect Language* pour détecter la langue d'un texte de moins de 1000 caractères.

Cet évènement nous a imposé de redéployer une partie des solutions sur un autre compte *Azure* ce qui fût une perte de temps non-négligeable.

Malgré l'implémentation pratiquement finalisée de la fonction de détection de la langue du texte sélectionné, cet évènement nous a imposé d'abandonner l'idée de détecter la langue du texte extrait du PDF afin d'adapter la reconnaissance et la synthèse vocale en conséquence.

Spring boot

L'utilisation de Spring boot pour le serveur était une des tâches les plus compliquées. De nombreuses difficultés ont été rencontrées dont des erreurs de dépendances, conflits entre API et conflits de version de package (JaCoCo notamment) avec Maven entre autres. La mise en route était difficile mais une fois réalisé, il a été aisé de concevoir le serveur ainsi que de mettre en place la base mongoDB une fois la connexion établie avec cette dernière.

Cependant, nous n'avons pas réussi à implémenter dans le temps imparti les sessions ou les cookies avec Spring. Cela nous aurait permis de nous passer de l'id du client dans les transactions avec le serveur et de s'assurer de la bonne authentification du client vis-à-vis de l'application.

Sonar & Jenkins

La mise en place des serveurs Sonar et Jenkins fût difficile principalement à cause du manque d'expérience que nous avions au départ sur ces technologies.

Le serveur Sonar fût aisément mis en place en local et mis en relation avec la partie back-end du projet, même si nous n'avons pas réussi à le configurer correctement pour analyser la partie Java Android du front-end. Nous avons réussi cependant à le mettre en lien avec Jenkins afin de vérifier la qualité de notre application au fur et à mesure des ajouts sur la branche finale de développement.

Le serveur Jenkins fût cependant plus difficile à mettre en place avec des problèmes de permission d'accès et d'intégration à Git. Nous avons pu cependant le mettre en oeuvre et réaliser la compilation du code lors des différents Pull Request qui ont eu lieu dans notre projet afin d'éviter de casser la branche finale de développement.

Différentes API

Certaines personnes de notre groupe ayant déjà travaillé avec les *Cognitive Services* de Microsoft en stage, le principe d'installation de Speech a été facilement compris et les documentations et exemples de code fournis par Microsoft ont été particulièrement utiles et appréciés.

Nous aurions aimé que l'installation du *Text-To-Speech* se déroule de la même manière, avec le même service Azure, mais il n'existait pas de service *Text-To-Speech* en Java pour Android. Il n'y avait que du C# et C++. Nous avons donc dû chercher un autre moyen de synthétiser de la voix, et nous avons trouvé Java Speech API, qui est extrêmement facile d'utilisation.

Il nous a été difficile de trouver un outil de comparaison de texte, en Java, qui nous renvoyait des éléments comparatifs à partir de deux textes. Finalement, nous avons trouvé la librairie Java Diff Utils, disponible en *open source*, qui fonctionne exactement comme nous le souhaitions.

Utilisation thread

L'architecture de l'application Android nécessite de comprendre le fonctionnement des *threads*. Cependant, n'ayant pas les connaissances suffisantes sur l'ensemble des librairies et des classes destinées à ceux-ci, il a fallu, dans un délai très court, essayer de comprendre son fonctionnement. Malheureusement, celui-ci n'ayant pas totalement été compris, nous avons rencontré des nombreuses choses non triviales ce qui a posé des contraintes au

cours du développement, notamment dans la réalisation des requêtes HTTP et du rafraîchissement de la vue courante.

Bilan

Bilan Humain

Le projet s'est très bien passé sur le plan humain. Durant la dizaine de jours, l'ensemble des membres de l'équipe ont été très actifs sur le projet et ont produit un travail efficace, le tout dans une bonne ambiance. Le travail hors séance a été très important mais cela n'a pas dérangé l'équipe car le projet était très intéressant. Les technologies utilisées, étant choisies par l'équipe, ont fait que chaque personne y a trouvé un intérêt. De plus, le fait d'avoir un premier rendu très rapidement a encore plus motivé l'équipe. En tant que chef de projet, il m'a fallu répartir chaque personne sur des technologies avec lesquelles elle était familière.

Le seul bémol que nous avons à propos de ce projet est sa durée. En effet, nous aurions aimé avoir plus de temps pour le réaliser car il nous a semblé être très formateur dans notre cursus scolaire.

Bilan Technique

Nous sommes satisfaits du travail qui a été effectué jusqu'à présent. La prise en main a été plus ou moins difficile pour chaque membre. Les personnes affectées à la partie back-end ont rencontrées plus de difficultés que les membres sur la partie front-end. En effet, le déploiement de Jenkins et de Spring boot a été plus compliqué que prévu car l'ensemble des tutoriels trouvés étaient obsolètes.

Nous n'avons pas pu développer l'ensemble des fonctionnalités que nous avions prévues par manque de temps. Cependant, si nous l'avions eu, nous aurions pu avoir une application contenant l'ensemble de nos fonctionnalités.