TradUI

Une application mobile crossplatform de traduction
créée par **Maurine Jouault** et **Gautier Laisné** et conçue sous **ReactNative**

https://github.com/ensc-mobi/projet-laisnejouault

Sommaire

I. Présentation de l'application

1. Objectifs

TradUI a pour but d'appréhender à la fois l'API de traduction conçue par IBM et les connexions à des bases de données MySQL.

L'application se scinde en deux parties : la première correspond à un système de traduction mettant en avant ces API et le fonctionnement asynchrone que propose ReactNative (permettant ainsi à l'utilisateur de traduire des termes en très peu d'étapes). La seconde partie se concentre sur la connexion de l'application avec une base de données hébergée en ligne et apporte une dimension ludique sous forme de jeu où l'utilisateur tente de faire mieux que les API d'IBM.

2. Ressources externes

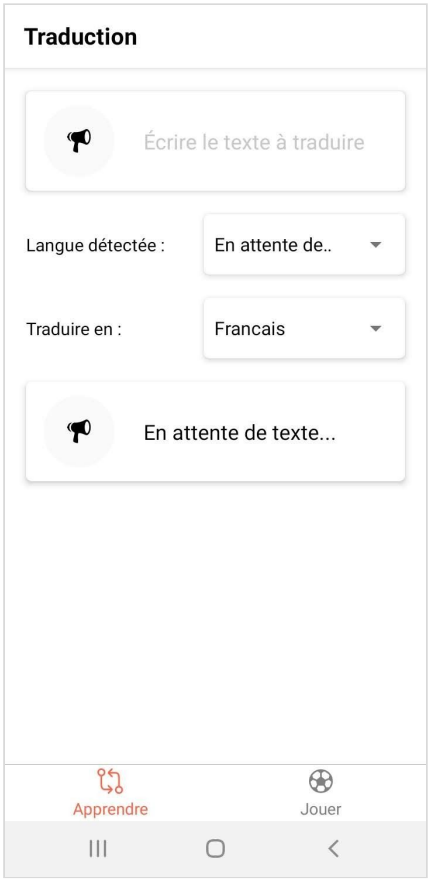
Librairies nécessaires au lancement de TradUI	Ionicons
	`expo install ionicons`
	Expo Speech
	`expo install expo-speech`
	ReactNative Chart Kit
	`expo install react-native-chart-kit`
	`expo install react-native-svg`
	ReactNative Navigation

Bases de données	Les BDD utilisées dans TradUI sont hébergées gratuitement par **000webhost** et sont disponibles sur http://projet-dev-mobile-laisnejouault.000webhostapp.com/.

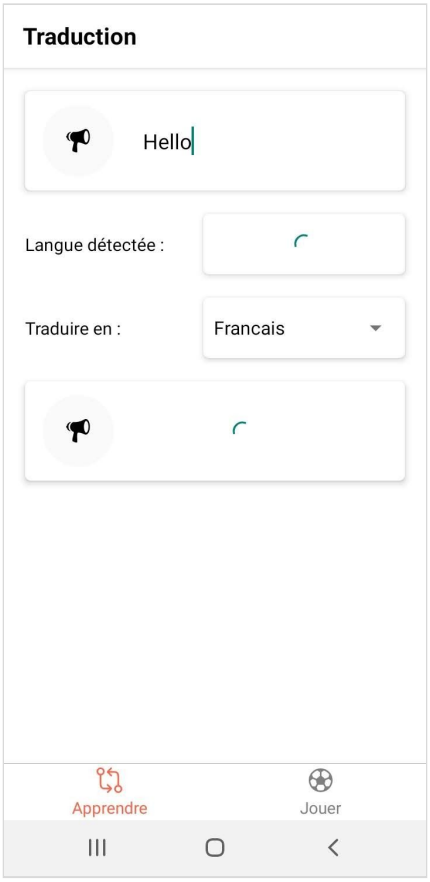
3. Captures d'écran

Stack "Apprendre"

Parcours basique :



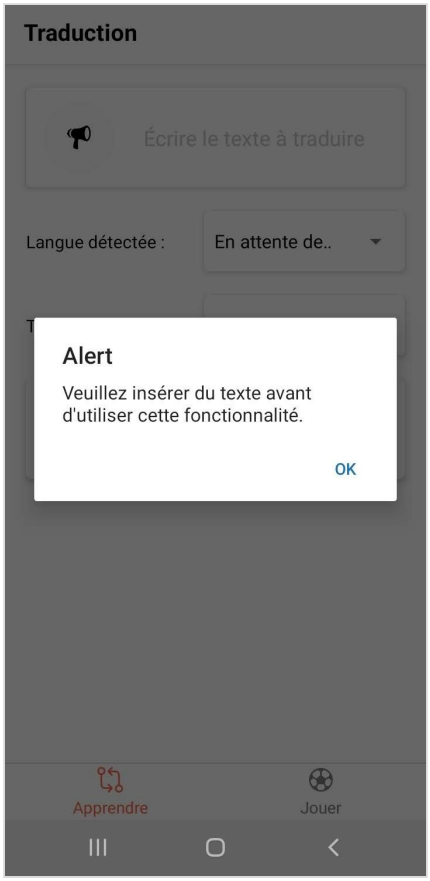
*C.1 Écran de traduction vide. Rien ne se passe tant que l'utilisateur ne saisit pas de texte dans le premier encart. Lorsque du texte est inséré, les appels à l'API de **détection de langage** puis de **traduction** sont lancés.*



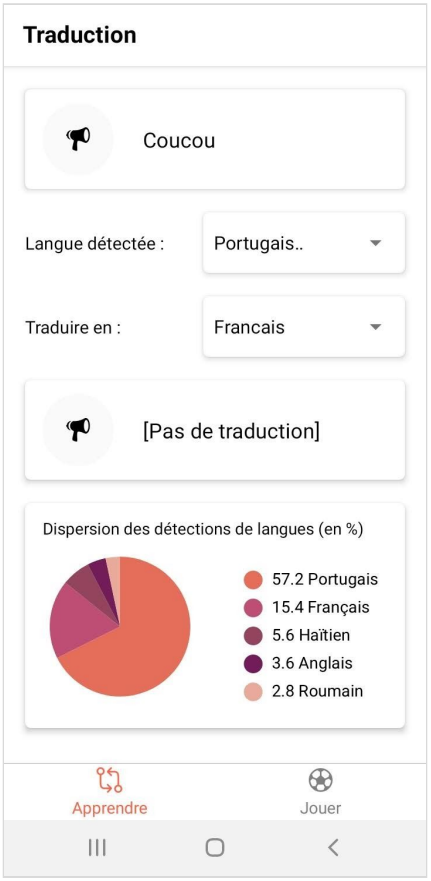
C.2 Écran de traduction en charge. Les requêtes se lancent ½ seconde après la fin de l'insertion du texte. Lorsque le texte est vide, ces requêtes sont annulées.

C.3 Écran de traduction avec traduction finale et graphe. Il est possible d'écouter une synthèse vocale de chaque texte dans leur langue respective en appuyant sur le bouton microphone.

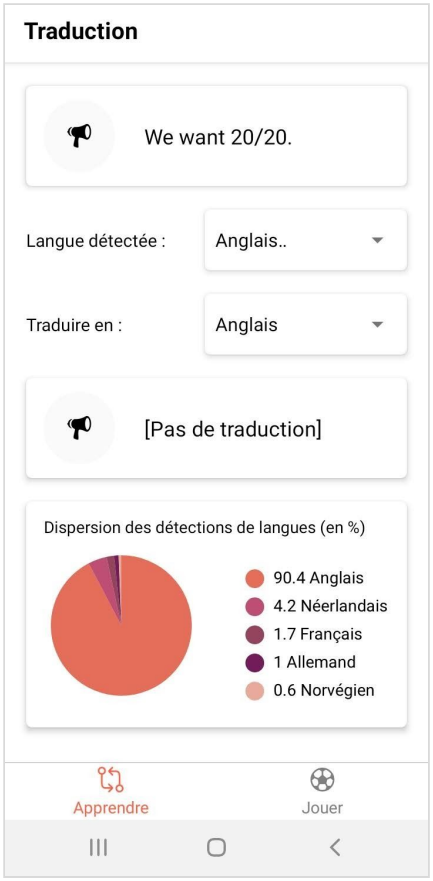
Gestion des erreurs :



C.4 Alerte lorsque l'utilisateur souhaite synthétiser le texte mais que rien n'a été inséré.



C.5 Le terme [Pas de traduction] apparaît lorsque l'API d'IBM n'a pas trouvé de traduction. NB : cela arrive souvent lorsqu'une traduction demandée ne part pas de l'anglais.



C.6 De même, ce terme apparaît lorsque la langue détectée est identique à celle de traduction. L'API ne gère pas cela.

Stack "Jouer"

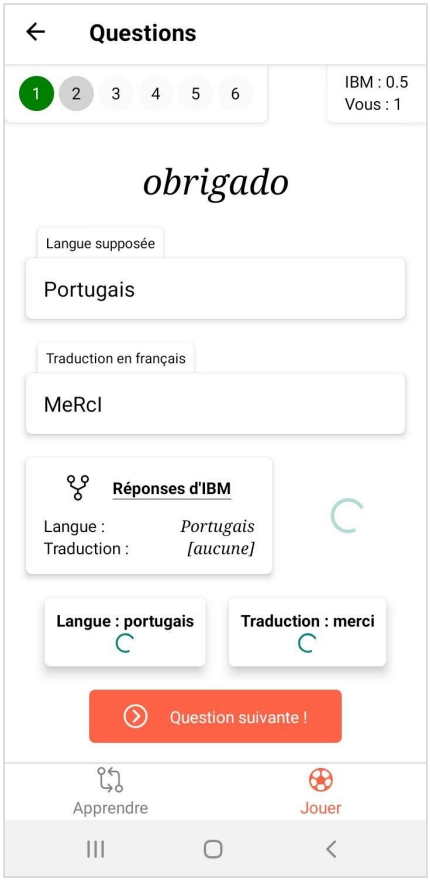
Parcours basique :

C.7 Les règles du jeu s'affichent lorsqu'on décider d'aller dans le menu Jouer.

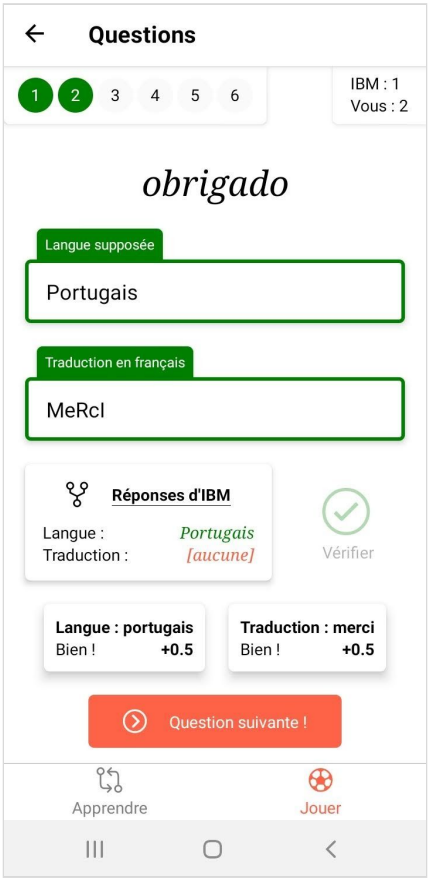
C.8 Le jeu nécessite de faire appel à des questions qui sont stockées dans des BDD en ligne. Toutes les questions sont récupérées lors du lancement de la partie.

C.9 Écran d'ajout d'une question au jeu. Cette question s'ajoute à la suite des autres.

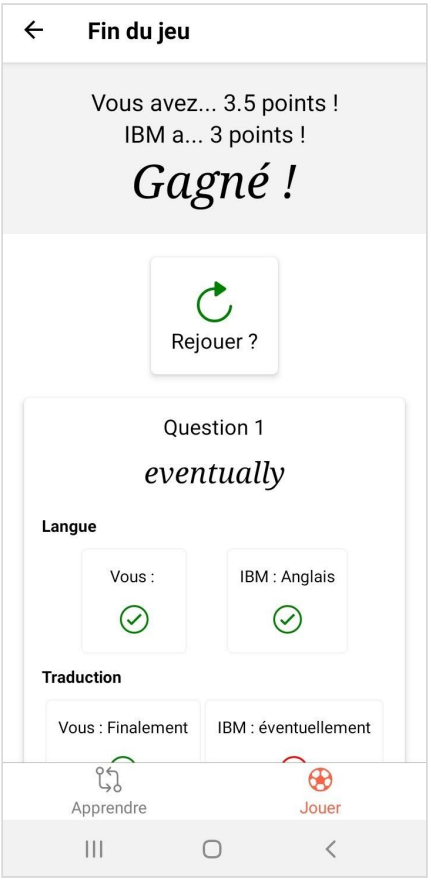
C.8 Exemple de question demandée durant la partie. IBM répond également aux questions. L'objectif est d'avoir plus de points à la fin de la partie qu'IBM. 6 questions sont initialement prévues, mais l'utilisateur peut en ajouter.



C.9 Lorsque la vérification est lancée, des requêtes sont envoyées pour savoir si les réponses sont correctes d'une manière ou d'une autre. Ces autres traductions peuvent être ajoutées par l'utilisateur s'il considère qu'une de ses réponses est tout de même correcte et qu'il a eu faux.

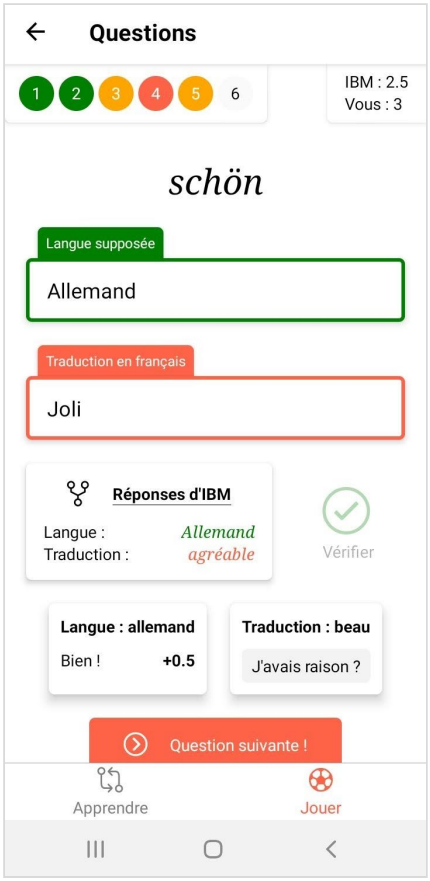


*C.10 Ecran du jeu une fois toutes les vérifications faites. Les **réponses sont formatées** afin d'avoir une meilleure correspondance avec les mots attendues.*

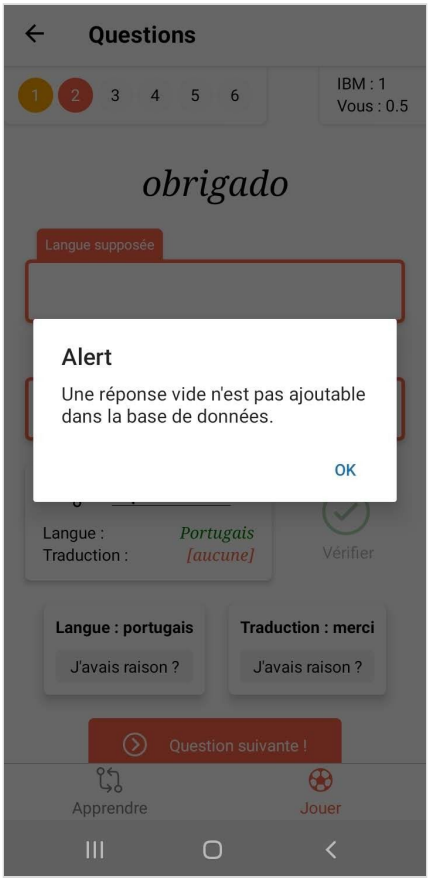


C.11 Une fois la dernière étape du jeu franchie, un écran de fin du jeu récapitule toutes les réponses et propose de rejouer. En cas d'égalité, l'utilisateur gagne.

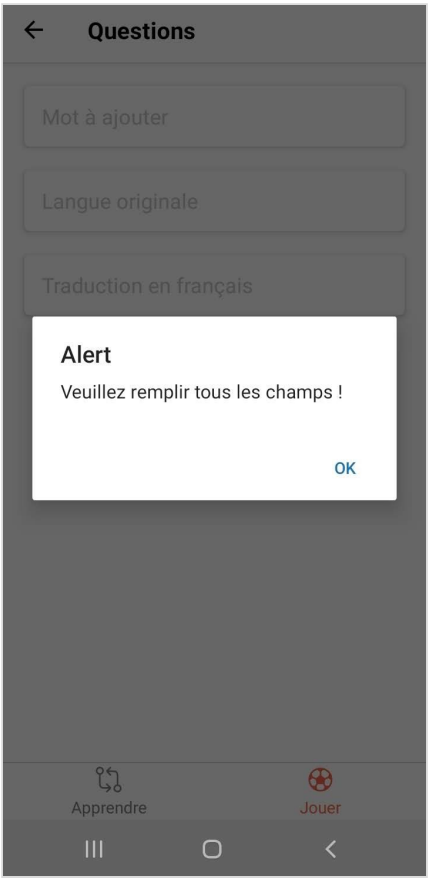
Gestion des erreurs :



*C.12 Possibilité de pouvoir dire qu'une réponse est tout de même correcte. Ici, le terme "beau" est attendu mais "joli" peut également être accepté dans certains contexte. Appuyer sur **J'avais raison ?** ajoute une réponse dans les BDD et "joli" deviendra un terme correcte pour une prochaine partie.*



*C.13 Si le bouton **J'avais raison ?** est cliquée alors qu'il n'y avait pas de réponse alors on interdit l'ajout d'élément vide dans la BDD.*



C.14 L'insertion d'une question dans les BDD fait également attention aux insertions vides.

II. Fonctionnalités disponibles

Référence	Description
Stack "Apprendre"	
F_1	En tant qu'utilisateur, je peux demander à traduire un mot/une phrase
F_2	En tant qu'utilisateur, je peux écouter le mot/la phrase à traduire
F_3	En tant qu'utilisateur, je peux visualiser graphiquement la dispersion des détections de langue par l'API *IBM Translator*
F_4	En tant qu'utilisateur, je peux modifier la langue source initialement détectée par l'API *IBM Translator*
F_5	En tant qu'utilisateur, je peux choisir la langue de traduction parmi Français/Anglais/Espagnol/Italien/Arabe/Allemand/Polonais/Chinois/Japonais
F_6	En tant qu'utilisateur, je peux écouter le mot/la phrase traduit(e) par l'API *IBM Translator*
Stack "Jouer"	
F_7	En tant qu'utilisateur, je peux "affronter" IBM Watson dans un jeu visant à trouver la langue et/ou la traduction d'une série de mots. Le but étant de gagner plus de points qu'*IBM Translator*.
F_8	En tant qu'utilisateur, je peux indiquer que ma réponse était bonne s'il y a ambiguïté sur la langue du mot. La réponse est alors enregistrée parmi les réponses possibles pour ce mot dans la BDD
F_9	En tant qu'utilisateur, je peux indiquer que ma traduction d'un mot était correcte (il peut s'agir d'un synonyme). La réponse est alors enregistrée parmi les réponses possibles pour ce mot dans la BDD.
F_10	En tant qu'utilisateur, je peux ajouter une question au jeu
F_11	En tant qu'utilisateur, je peux visualiser mon score mis à jour à chaque question ainsi que celui d'IBM.

III. Choix de conception

1. Architecture du code

Nous avons choisi de rédiger l'entièreté du code en anglais dans l'optique de pouvoir le présenter sur GitHub à des entreprises internationales. *Voir Annexe 1 : Carte mentale de l'architecture du code.*

Notre code essaye de suivre le principe **DRY** et la **séparation des responsabilités**. Pour cela nous avons scindé notre code en fonctions ayant des responsabilités uniques. Par exemple certaines sont dédiées uniquement aux appels API et se charge de mettre à jour l'état d'un composant, d'autres sont vouées à gérer les transmissions de données avec les composants enfants ou d'autres encore sont attitrées au formatage des données JSON.

Pour plus de détails, voir Annexe 2 : Tableau récapitulatif des méthodes utilisées.

2. Lifting State Up

Pour scinder proprement le code et obtenir un bel arbre de composants, nous avons pratiqué le *Lifting State Up*. Pour cela, nous avons transmis dès que nécessaire une fonction callback aux composants enfants; le composant parent faisant office de sur-ensemble de fonctionnalités regroupées par un même thème. *Voir Annexe 3 : Tableau récapitulatif des callbacks.*

```
<View style={styles.container}>
    <TextToTranslate
        handleInsertedTextChange={this.handleInsertedTextChange}
        chosenInitialLanguage={chosenInitialLanguage}
    />
    <DetectedLanguages
        insertedText={insertedText}
        handleDetectedLanguagesChange={this.handleDetectedLanguagesChange}
        detectedLanguages={detectedLanguages}
        chosenInitialLanguage={chosenInitialLanguage}
        handleChosenInitialLanguageChange={
            this.handleChosenInitialLanguageChange
        }
    />
    <TranslationLanguage
        chosenInitialLanguage={chosenInitialLanguage}
        chosenTranslationLanguage={chosenTranslationLanguage}
        handleChosenTranslationLanguageChange={
            this.handleChosenTranslationLanguageChange
        }
    />
    <TranslatedText
        insertedText={insertedText}
        chosenInitialLanguage={chosenInitialLanguage}
        chosenTranslationLanguage={chosenTranslationLanguage}
    />

    {detectedLanguages.length > 0 ? <GraphLanguages detectedLanguages={dete
</View>
```

Ceci a permi, par exemple, d'obtenir un code rigoureusement séparé pour la page de traduction de l'application qui est, nous pensons, sémantiquement bien plus lisible.

À noter : nous n'avons jamais eu la nécessité de transmettre des props à plus d'un niveau de profondeur. Si ça avait été le cas, nous aurions sûrement envisagé d'utiliser un gestionnaire d'état tel que MobX.

3. Fonctions asynchrones

Bien que notre projet ne **nécessite pas** forcément l'utilisation de fonctions asynchrones, nous avons jugé intéressant de les mettre en place principalement pour les appels aux API. Celles-ci permettent de rendre l'ensemble plus lisible. En effet, jouer avec des `await` est; selon notre perspective, un peu moins lourd que l'ancienne syntaxe basée sur des `.then()`, `.catch()`…

Pour donner un exemple concret : faire une traduction avec l'API de traduction d'IBM nécessite une langue de départ au préalable. Cette langue est, dans notre cas, récupérée par un autre appel à une API. Nous devions donc avoir dans notre code une imbrication de `.then()` que nous ne jugions pas forcément des plus pratiques, notamment pour gérer les erreurs.

Voici un exemple de notre code :

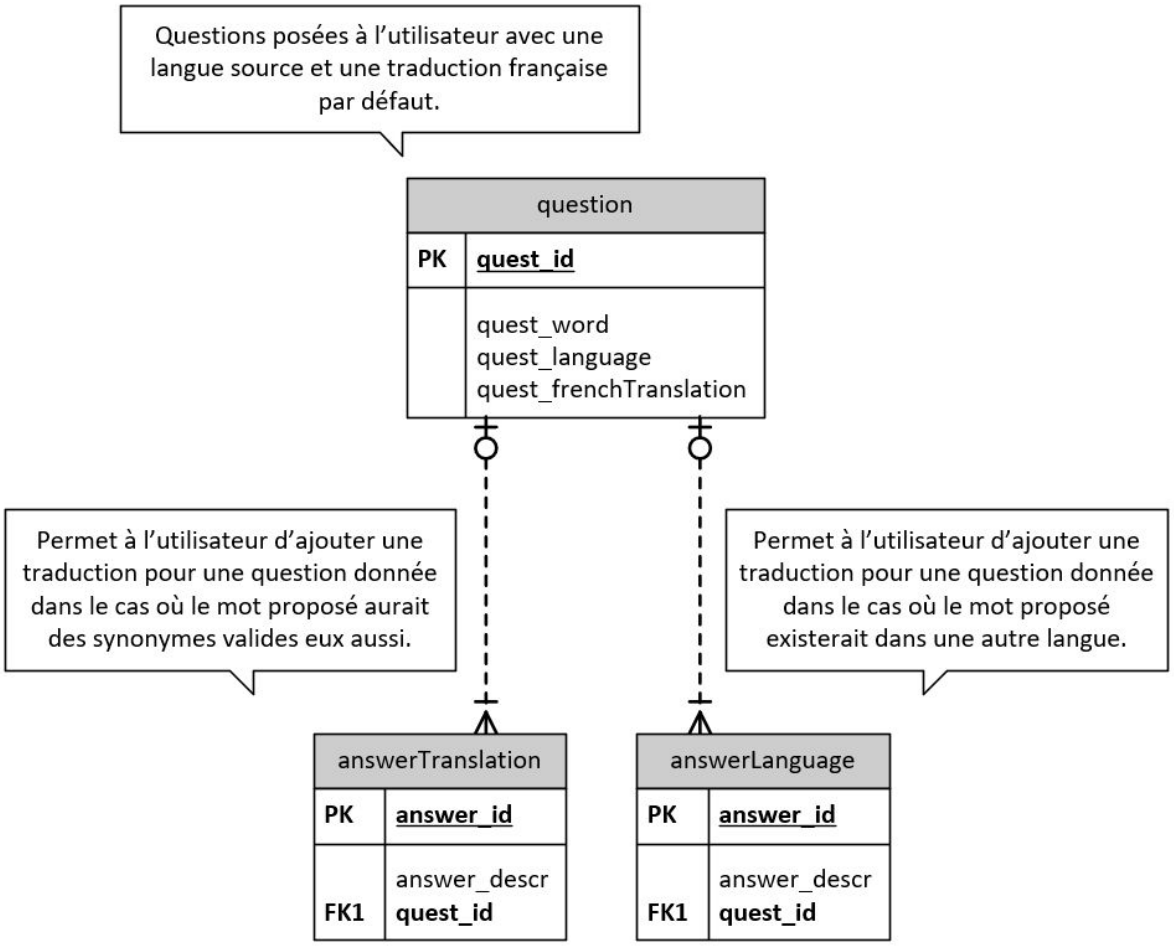
```
searchAnswers = async word => {
  this.setState({ isDetectingLanguages: true });

  try {
    const response = await getLanguageDetection(word);
    const data = (await response.json()).languages;
    this.setState(
      { detectedLanguages: data, detectedLanguage: data
      () => this.translate(data[0].language, word)
    );
  } catch (e) {
    console.log(e.message);
    this.setState({ detectedLanguage: "[aucune]" });
  } finally {
    this.setState({ isDetectingLanguages: false });
  }
};
```

Ici, la gestion des erreurs de l'API est gérée avec un `try...catch`.

Ce que nous avons trouvé intéressant en utilisant la fonction `searchAnswers` en version asynchrone est la **lecture quasi linéaire de son contenu** : on tente de faire une détection de langue. Si cela réussit, alors on récupère les données. Une fois cela fait, on modifie l'état de notre composant, puis la traduction est lancée. Finalement, on annonce que la détection est terminée.

4. Stockage des informations de jeu

Les informations de jeu sont stockées dans une base de données hébergée par 000webhost. Une copie de tous les fichiers présent sur la plateforme est disponible dans le dossier `./src/services/php`



Modèle Conceptuel de Données de nos bases de données

5. Design

Le design très épuré de l'application se veut consistant sur l'ensemble des pages. C'est un choix de conception car il se base sur les tendances web apparues vers fin 2019. Ce design met en valeur les ombrages et les nuances de blanc/gris. Nous avons également insisté sur les feedbacks visuels pour l'utilisateur : à chaque fois qu'un temps d'attente est à prévoir, un indicateur d'activité s'affiche. Cela arrive surtout avec les appels aux API.

IV. Organisation du travail en équipe

Ayant des niveaux différents en programmation, nous avons choisi de beaucoup travailler en pair-programming pour ce projet. Cela a permis un gain en compétence pour la personne la moins expérimentée d'entre nous tandis que l'autre a pu approfondir le cours notamment en matière de factorisation du code et de documentation sur la technologie ReactNative.

Dans un second temps (#coronavirus), nous nous sommes répartis les fonctionnalités, l'un travaillant sur les requêtes relatives au stack "Apprendre" et l'autre sur les requêtes relatives au stack "Jeu".

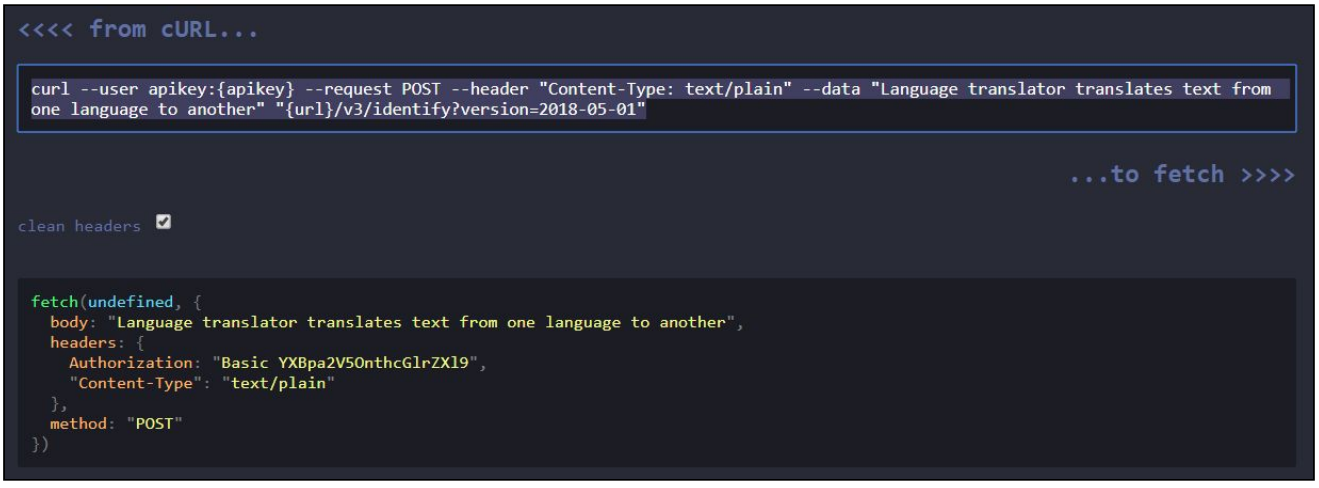
Nous avons également créé un répertoire Git de notre projet afin de faciliter le travail en équipe mais également la gestion de versions du code.

V. Difficultés rencontrées

1. Authentification d'accès à l'API

L'authentification requise pour se connecter à l'API d'IBM a été très chronophage. Il y a deux manières de pouvoir s'y connecter : soit en utilisant un système d'authentification de notre application avec des *tokens*, soit en utilisant une clé d'api unique associée à notre compte IBM. La première solution semblait la plus prometteuse, moderne et simple d'utilisation. Cependant, nous nous sommes heurtés à notre manque de connaissance et nous avons été dans l'incapacité de la mettre en place, après beaucoup d'heures consacrées à cela.

La seconde solution a fonctionné pour nous. Cependant, il nous a été difficile d'adapter la documentation fournie par IBM à ReactNative. Celle-ci se base sur des requêtes cURL ou sur Node.js. Or, la différence entre Node.js, ReactNative et JavaScript ne nous apparaissait pas de manière claire et évidente. Nous ne pouvions pas adapter le code fourni par la documentation sans faire d'erreurs. Finalement, nous avons trouvé un site permettant de convertir une requête cURL en requête JavaScript.
Cela nous a permis d'obtenir une clé d'autorisation produite automatiquement que nous ne parvenions pas à obtenir sinon.



2. Qualité de l'API d'IBM

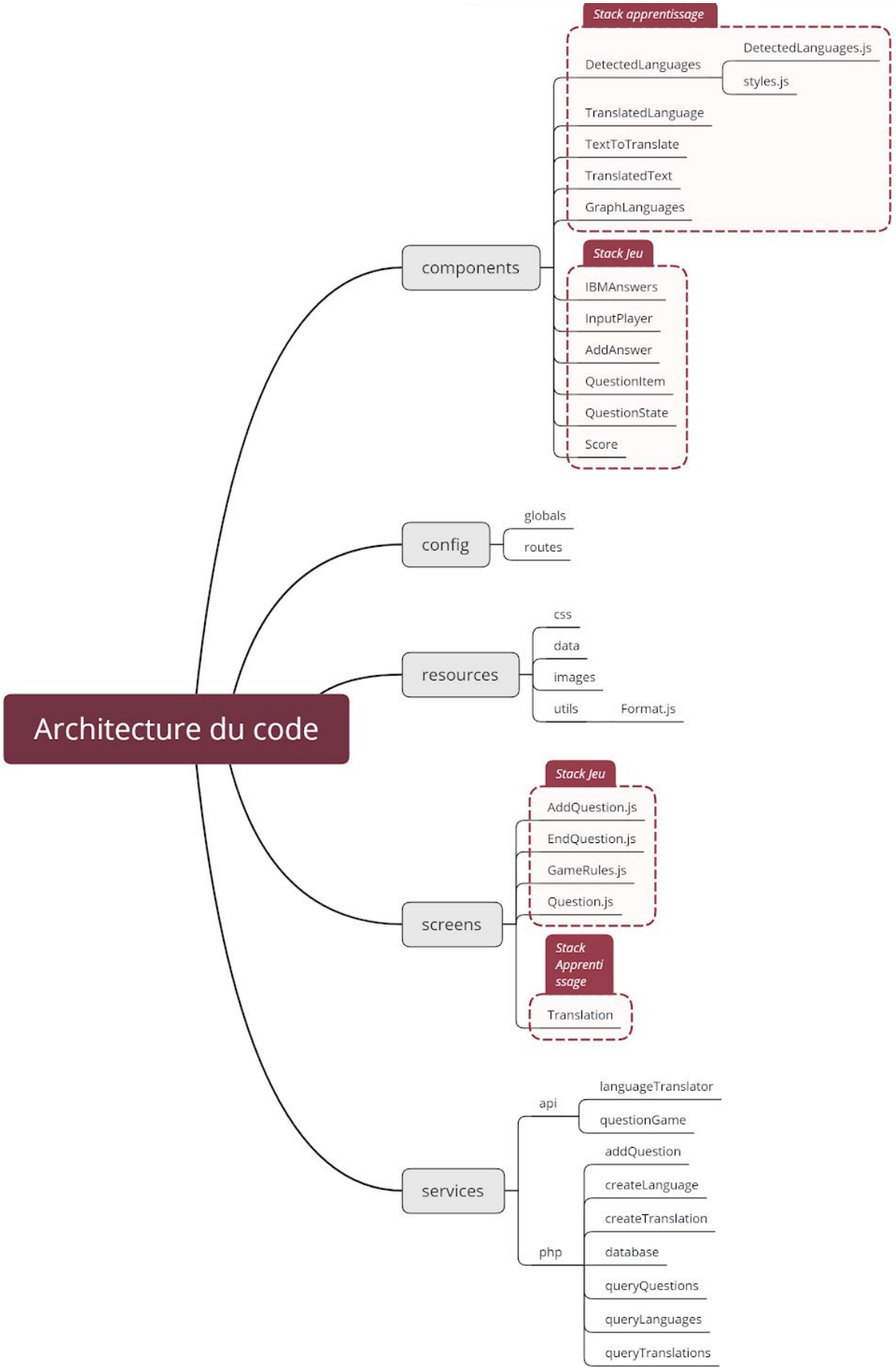
Nous nous sommes rendus compte en milieu de projet que l'API *IBM Translator* était peu performante lorsqu'il s'agissait de détecter et de traduire des mots issus d'une autre langue que l'anglais. Cela nous a empêché d'implémenter certaines fonctionnalités auxquelles nous avions initialement pensées. Nous souhaitions permettre à l'utilisateur de consulter des mises en contextes du mot traduit à la manière de Reverso par exemple. Le fait que l'API se "trompe" souvent de langue de détection ou de traduction a rendu cette fonctionnalité peu pertinente.

Également, l'une des fonctionnalités envisagées dans le stack "Jeu" était de générer aléatoirement des mots en français au moyen d'une autre API puis de se servir d'*IBM Translator* pour traduire ces mots. L'objectif de l'utilisateur aurait été de trouver la langue du mot et/ou sa traduction. Pour pallier à ce problème et pour tout de même proposer une fonctionnalité ludique à l'utilisateur, nous avons choisi de créer notre propre base de données. Nous y avons entré des mots de différentes langues, avec la traduction en français qui nous semblait la plus pertinente. Ainsi, nous n'avions plus besoin de l'API pour détecter la langue source ni pour traduire le mot.

L'enjeu pour l'utilisateur est donc de battre IBM Watson dans la détection de la langue ainsi que dans la traduction des mots proposés. S'il obtient plus de points que l'IA à la fin de la partie, il l'emporte.

ANNEXES

Annexe 1 : Carte mentale de l'architecture du code (tout est dans le dossier ./src)



Annexe 2 : Tableau récapitulatif des méthodes utilisées

Emplacement	Nom de la méthode	Données d'entrée	mise à jour de l'état (setState)/sortie	But
COMPONENTS				
DetectedLanguage	detectLanguage	texte	detectedLanguages isDetectingLanguages : false	Appel requête API à getLanguageDetection() *Cf tableau services >>api*
	pickerChange	index	chosenInitialLanguage pickerValue	Affichage des langues détectées sous la forme d'une liste de sélection déroulante. Permet à l'utilisateur de choisir une autre langue en cas d'erreur de détection
GraphLanguage	findCountry	codeIso	languageFound.French_name	Afficher le nom complet de la langue source
TextToTranslate	listenToInsertedText	⌀	⌀	Ecouter le texte entré par l'utilisateur
	listenToTranslatedText	⌀	⌀	Ecouter la traduction
	translate		transaltedText isTranslating	Appel à requête API getTraduction() *Cf tableau services >>api*
AddAnswer	createTranslation	questionId	isLoading disabled	Appel à requête addAnswer() *Cf tableau services >>php*
InputPlayer	⌀	⌀	⌀	⌀
IBMAnswers	searchAnswers	texte	isdetectingLanguage translatedText isTranslating	Récupérer les réponses (langue source et traduction) proposées par IBM pour une question donnée
QuestionItem	getQuestions	⌀	questionsData isLoeading disabled compteur	Afficher une par une les questions suivant un ordre
QuestionState	computScore	who quest_id	scoreQuestion	Mettre à jour le score du joueur à chaque question
Score	computScore	who	score	Calculer et Afficher le score final du joueur
SCREENS				
AddQuestion	createQuestion	quest_word quest_language	isLoading disabled	appel à requête php addQuestion() *Cf tableau services >>php*

		quest_frenchTranslation		
EndGame	questions	⌀	isLoading questionsData disabled	Appel à requête API getQuestions () *Cf tableau services >>api*
GameRules	questions	⌀	isLoading questionsData disabled	Appel à requête API getQuestions () *Cf tableau services >>api*
Questions	checkEveryAnswer	expectedAnswer answerInput allAnswers	true/false	Si la réponse de l'utilisateur/d'IBM est différente de la réponse par défaut, comparaison par rapport à toutes les autres réponses possibles proposées par les utilisateurs
	check	expectedLanguage expectedTranslation	hasAlreadyChecked	Vérification des réponses du joueur et d'IBM Translator
	updateGS (=gameState)	who array valueToAdd	who aray	Mise à jour de l'état de jeu d'IBM/de l'utilisateur. L'état de jeu d'un joueur est composé de : ● l'historique de ses réponses pour les langues sources et celui de ses réponses pour les traductions ● Ses points gagnés pour les langues sources et ceux gagnés pour les traductions
	updateAnswers	player IBM languageInput translationInput languageIBM translationIBM		
	updateScore	who array expectedAnswer AllAnswers		Mise à jour des points d'IBM et du joueur à chaque question
	nextQuestion	questions	languageInput translationInput count hasAlreadyCkecked disabledAddLanguage	Passage à la question suivante du jeu

			disabledAddTranslation	
Translation				
SERVICES				
api				
languageTranslator	getLanguageDetection	texte	⌀	Requête API pour récupérer les langues détectées comme potentielle langue source du texte
	getTraduction	texte initialLanguage finalLanguage	⌀	requête API pour récupérer la traduction du texte
questionGame	getQuestions	⌀	⌀	appel asynchrone à requête php *cf tableau services>>php*
	getTranslations	questionId	⌀	
	getLanguages	questionId	⌀	
	addQuestion	data	⌀	appel asynchrone à requête php *cf tableau services>>php*
	addLanguage	input questionId	⌀	appel asynchrone à requête php *cf tableau services>>php*
	addTraduction	input quesitonId	⌀	appel asynchrone à requête php *cf tableau services>>php*
php				
createTraduction		answerT_descr quest_id	⌀	requête php pour ajouter une nouvelle langue source possible pour une question donnée
createLanguage		answerL_descr quest_id	⌀	requête php pour ajouter une nouvelle traduction possible pour une question donnée
addQuestion		quest_word quest_language quest_frenchTranslation	⌀	requête php pour ajouter une question au jeu
queryLanguages		quest_id	⌀	requête php pour récupérer les réponses possibles de langue source ajoutées par

			les utilisateurs pour une question donnée
queryQuestions	∅	∅	requête php pour récupérer toutes les informations relatives aux questions du jeu
queryTranslations	quest_id		requête php pour récupérer les réponses possibles de traduction ajoutées par les utilisateurs pour une question donnée

Annexe 3 : Tableau récapitulatif des callbacks

Emplacement	Nom	Entrées	MAJ state	But
Translation	handleInsertedTextChange	newInsertedText	insertedText	Gérer immédiatement la modification du texte à traduire par l'utilisateur
	handleDtetectedLanguagesChange	newDetectedLanguages	detectedLanguages	Gérer immédiatement la modification des langues détectées comme potentielles langues sources par l'API
	handleChosenInitialLanguageChange	newChosenInitialLanguage	chosenInitialLanguage	Gérer immédiatement la modification de la langue source (en cas d'erreur de détection par IBM Translator) par l'utilisateur
	handleChosenTraductionLangageChange	newChosenTranslationLanguage	chosenTranslationLanguage	Gérer immédiatement la modification de la langue de traduction par l'utilisateur
Questions	handleIBMAnswers	language translation	languageIBM tranlsationIBM	Gérer immédiatement la modification par l'API sa réponse pour la langue source pour une question donnée
	handleLanguageAnswer	languageAnswer	languageInput	Gérer immédiatement la modification par l'utilisateur de sa réponse pour la langue source pour une question donnée
	handleTranslationAnswer	translationAnswer	translationInput	Gérer immédiatement la modification par l'utilisateur de sa réponse pour la traduction pour une question donnée