Analyse d'apprentissage d'une nouvelle langue à l'aide d'un Eye Tracker

Antoine Pirrone
Pierre Celor

Chargé de TD :
Pascal Desbarats

Souha Guissouma
Bissen Hamdi

Clients :
Nicholas Journet
Boris Mansencal

3 avril 2017

Table des matières

Chapitre 1

Introduction et Description du projet

Le monde ne cesse de changer, Il n'a toutefois jamais autant évolué que ces dernières décennies, avec la révolution technologique, nous avons assisté à un véritable essor des utilisations des ordinateurs ainsi que d'autres types d'appareils tels que les smartphones, les tablettes et autres. Ces technologies deviennent de plus en plus abordables pour le consommateur qui ne cesse pas de chercher à chaque fois de nouvelles façons plus efficaces plus intuitives et plus faciles afin d'interagir avec ces équipements.

Aujourd'hui, les ordinateurs sont équipés de webcams, microphones, Haut-parleurs, écrans haute résolution et surfaces sensibles au toucher, toutes ces outils peuvent servir à créer de nouvelles interactions. La technologie d'Eye Tracking commence à se populariser et propose une toute nouvelle manière d'interagir avec l'ordinateur, simplement en regardant l'écran.

Ce document est un rapport de projet de programmation (PdP), effectué lors du deuxième semestre de Master 1 Informatique à l'Université de Bordeaux, il porte sur la création d'une application web d'apprentissage linguistique à l'aide d'un Eye Tracker.
L'objectif de notre projet est la création d'une application web d'apprentissage linguistique permettant d'analyser le vocabulaire lu par l'utilisateur, de générer des statistiques de lecture, et de proposer des nouveaux textes à l'utilisateur de manière adaptée à sa progression, afin de l'accompagner dans son apprentissage.

L'acquisition des données de lecture se fait à l'aide d'un Eye Tracker. Ce périphérique est capable de détecter la position du regard sur un écran, on peut ainsi en théorie savoir quel mot à été lu par l'utilisateur. Nous verrons dans la suite du rapport les difficultés et limitations liées à l'utilisation des Eye Trackers, et plus particulièrement celles du périphérique que nous avons utilisé pour notre projet (Steeseries Sentry, cf fig 1.1) [1]. Nous verrons aussi en quoi le choix de réaliser l'application en Web a pu être problématique, et quelles ont été nos solutions.

1. `http://fr.steelseries.com/gaming-controllers/sentry`

FIGURE 1.1 – Notre Eye Tracker, le Steelseries Sentry

Le client a insisté sur le fait que l'application devra plus être une "Proof of Concept" (étude preuve de concept) qu'un produit fini commercialisable. Ainsi, il nous a demandé de nous concentrer sur les aspects techniques du projet, et non sur l'apparence et l'ergonomie de l'application.

Le présent rapport est subdivisé en cinq chapitres, dont le premier est celui ci, le second chapitre "Étude de l'existant" portera sur les domaines d'utilisation de l'Eye Tracker, et une analyse des projets qui ont un rapport avec le domaine de l'apprentissage, Le troisième chapitre détaille le cahier des besoins que nous avons réalisés en amont du développement en tant que tel du projet. Nous y expliquons les besoins fonctionnels, non fonctionnels et les contraintes liées à notre projet. Une section du chapitre est aussi consacrée à la présentation du prototype que nous avons réalisé pour valider la faisabilité du projet.
Le chapitre quatre présente l'architecture complète du logiciel final. On y trouve différents diagrammes de conception (classe, séquence, cas d'utilisation...) ainsi qu'une description des technologies utilisées pour le développement de l'application.
Le chapitre cinq présente notre approche pour implémenter l'architecture décrite auparavant. On y détaille notamment les points techniques intéressants, en expliquant les raisons de nos choix, les difficultés rencontrées et les solutions que nous proposons.
Le dernier chapitre détaille les tests unitaires et de fonctionnements que nous avons mis en place pour valider le bon fonctionnement de l'application.

Nous tenons à remercier notre chargé de TD, Pascal Desbarats, pour ses explications et conseils toujours très utiles ainsi que pour sa disponibilité et son investissement à notre égard.
Nous tenons aussi à remercier nos clients, Nicholas Journet et Boris Mansencal, dans un premier temps pour nous avoir offert l'opportunité de travailler sur un projet aussi intéressant, ainsi que pour les rendez vous toujours très agréables et encourageants.

Chapitre 2

Étude de l'existant

Avec des évolutions technologiques de plus en plus rapides, de nombreux acteurs s'intéressent progressivement aux interactions homme-machine via des outils innovants. L'Eye Tracking ne fait pas exception, et est en train de devenir un outil précieux dans différents domaines tels que le marketing, la psychologie et les jeux vidéos.

L'idée n'est pas de remplacer les outils existants, mais bien de mettre à profit les avantages de l'Eye Tracking. En effet, l'Eye Tracker, en tant que périphérique d'acquisition de données, est capable de déterminer la position des mouvements oculaires d'un individu en apportant de nouvelles techniques d'interaction basées sur le regard.

Notre projet tombe dans la catégorie l'apprentissage, nous avons donc cherché en priorité des projets en rapport avec ce domaine, mais sans délaisser d'autres projets moins connexes, mais qui traitent de l'utilisation de l'Eye Tracking pour analyser les comportements d'un utilisateur. Les sections suivantes donnent un aperçu sur les différents domaines d'utilisation ainsi que les solutions existantes.

2.1 Etat de l'art

Dans cette partie, on présentera les différents articles de revues, de journaux et les sites web qui constituent la bibliographie de notre projet et qui sont en relation avec le domaine de l'apprentissage à l'aide d'un eye tracker.
Ces références visent a améliorer nos idées sur le sujet lui même et a enrichir nos connaissances sur certains points, prenons l'exemple de l'article [8], cet article nous donne beaucoup d'informations sur comment attirer l'œil d'un utilisateur où l'on veut ou à l'inverse éviter que l'utilisateur soit attiré ailleurs. Cela pourra nous servir pour mieux penser notre interface graphique afin que l'utilisateur puisse lire le texte sans que son regard ne soit attiré par autre chose, ce qui pourrait fausser l'eye tracking. Pour ce qui concerne l'article [12] celui là nous montre comment ça fonctionne la détection des données à l'aide d'eye tracker dans un site web afin de savoir les parties qui attire plus l'attention par l'utilisateur . De plus, il explique le fonctionnement d'un eye tracker, ce qui a été utile pour comprendre exactement ce qu'on doit faire pour gérer l'eye tracker au niveau de

[11] nous montre comment analyser la lecture d'une langue étrangère à l'aide d'un eye tracker avec des exemples de structures grammaticale qui posent plus de problèmes que d'autres et les conséquences visibles sur les données de l'eye tracker.

Dans l'article [3], des expériences sont menées pour analyser en partie grâce à un eye tracker les conditions d'un lecteurs en phase d'apprentissage d'une langue étrangère en s'intéressant notamment à la forme physique et la concentration.

[2] nous informe sur la "fenêtre" de vision d'un lecteur. Il est très utile pour comprendre comment nous analysons les caractères et comment nous déplaçons nos yeux en lisant.

[6] est très intéressant pour notre projet puisse qu'il s'agit de chercher à détecter lorsqu'une personne lit un texte si elle connaît ou non les mots qu'elle a lu grâce à un Eye Tracker et un EEG.

L'article [4] porte sur l'analyse de l'attention d'un lecteur lisant un texte en langue étrangère pour détecter les mots inconnus grâce à un eye tracker
Les articles [7] [13][9][5] présentent des solutions qui existent sur le marché.

La page web [1] explique les différences entre les procédés de "lemmatisation" et de "*Stemming*".

L'article [10] propose un algorithme pour enlever les suffixes des mots afin d'en extraire la racine, l'algorithme de Porter. Ce procédé, appelé *racinisation*, est très proche du procédé de *lemmatisation*, mais est plus simple, en ce qu'il ne prends pas en compte le contexte dans lequel le mot à traiter apparaît. Nous avons considéré que cette approche était suffisante pour notre utilisation, de plus, elle est bien plus rapide à exécuter.

L'article [14] décrit et compare différentes méthodes de filtrage de mouvements des yeux.

2.2 Domaines d'utilisation

L'eye tracker est l'outil idéal pour l'apprentissage en ligne, mise à part son utilisation dans le domaine de l'éducation il est utilisé dans une variété de domaines tels que la psychologie, la médecine, le marketing, l'ingénierie et le jeu ainsi que pour améliorer l'interaction ordinateur humain en utilisant les yeux pour la navigation et les commandes. Les domaines ci dessous sont les plus courants :

Marketing
Au cours des dernières années, l'eye tracking pour les études de marché est devenu de plus en plus important. De nombreuses marques utilisent cet outil pour évaluer leurs produits, leur conception, leur publicité ou même le comportement commercial de leurs clients afin d'optimiser l'expérience globale du client. Avec l'eye tracking, il est possible de mesurer l'attention aux marques, produits et leurs messages clés ainsi que la facilité ou la difficulté de la navigation de magasin.

FIGURE 2.1 – `http://www.thinkeyetracking.com`

Recherche en Psychologie

Dans ce domaine, l'eye tracking peut être mesurée et corrélée avec d'autres mesures telles que le fonctionnement du cerveau. L'eye tracking peut être utilisée pour des populations normales ainsi que pour des sous-populations spécifiques qui ont des schémas comportementaux remarquables ou différents types de troubles de santé mentale.

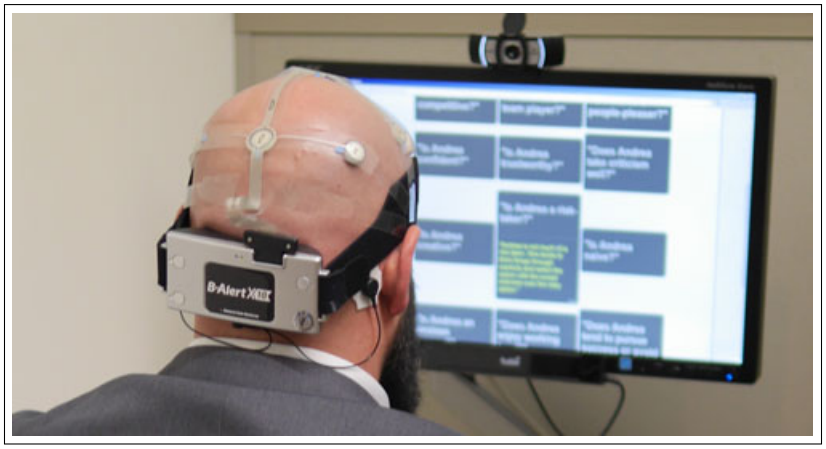


FIGURE 2.2 – `http://www.imotions.com/psychology`

Jeux Videos

On commence à voir apparaître des jeux proposant de modifier l'expérience de jeu en utilisant des Eye Trackers, par exemple pour contrôler la vue d'un personnage, ou pour mettre en avant certains détails à l'endroit ou le joueur regarde.
L'Eye Tracking est aussi utilisé par certains joueurs professionnels pour collecter des statis-

tiques sur leurs style de jeu, par exemple, dans Starcraft 2, il est essentiel de toujours avoir un œil sur la *MiniMap* afin de voir venir le plus tôt possible un assaut ennemi ; le joueur peut se rendre compte de la fréquence à laquelle il effectue certaines actions, et corriger son comportement afin d'améliorer son niveau, ou même mettre en place une alerte lors de ses séances d'entraînement qui lui indique qu'il faut qu'il regarde la *MiniMap* s'il ne l'a pas fait dans un intervalle de temps .

FIGURE 2.3 – `http://www.tobiipro.com/fr/domaine-dapplication/experience-utilisateur-et-interaction`

Recherche clinique

L'eye tracking est de plus en plus déployé en recherche clinique. En effet les mouvements oculaires permettent de diagnostiquer des états, de les comparer à différents stades d'une maladie ou d'un traitement afin de pouvoir quantifier les évolutions de manière normative.

FIGURE 2.4 – `http://www.tobiipro.com/fr/domaine-d'application/recherche-clinique`

Pour notre projet, nous allons nous intéresser au domaine de l'apprentissage, la section

suivante présente une analyse de quelques exemples des applications dans ce domaine.

2.3 Étude des applications existantes

À l'issue de nos recherches, nous avons trouvé une diversité des solutions dédiés à l'apprentissage à partir d'un Eye Tracker. Nous avons pris à titre d'exemple ces solutions :

AdeLE [8] , un framework dédié à l'apprentissage en ligne, son objectif principal est d'observer les activités d'apprentissage des utilisateurs en temps réel en surveillant un certain nombre d'aspects comportementaux et de traits personnels tels que les objets et les zones de concentration, le temps passé sur les objets et la séquence dans laquelle le contenu d'apprentissage est traité ainsi d'identifier les problèmes de compréhension et fournir des informations ou des explications supplémentaires sélectives. Ainsi,on trouve que certains résultats du projet AdeLE peuvent contribuer à trouver de nouvelles façons de créer des environnements adaptatifs avancés pour l'enseignement et l'apprentissage et abordables pour les institutions dans un proche avenir, aussi on peut dire que la structure architecturale de AdeLE facilite une grande variété d'applications tel que la notre.

Nous avons aussi trouvé un projet innovant, il s'agit de celui de Tristan Hume, un étudiant développeur au Canada, il a réalisé un projet [5] en 2012 sur un détecteur du centre de l'œil en OpenCV (librairie spécialisée dans le traitement d'images), avec comme seul matériel la webcam. Une vidéo reflétant bien son travail est disponible sur son site.

Dans [14], projet de thèse de maîtrise intitulé "Eye Gaze Tracking for reading progress". Le projet a été réalisé par des étudiants à l'Université d'Aalborg en 2013. Le but de ce projet est de suivre le processus de lecture des sujets grâce à un Eye Tracker . Pour ce faire, les tests ont été basé sur les saccades et les fixations ce qui va nous servir à enrichir nos informations sur celles ci.

L'article [10] parle d'une étude qui réalise des tests d'utilisabilité sur un cours en ligne intitulé «histoire de la littérature étrangère» en intégrant des techniques de suivi des yeux. Un eye tracker de type Tobii T60 a été utilisé dans cette étude à l'Université normale de l'est de la Chine-Université du Missouri. Ce cours est un cours de premier cycle obligatoire pour les étudiants majeurs de littérature chinoise et aussi un cours électif pour ceux qui sont d'autres domaines, tels que l'anglais, l'histoire et les majors de l'éducation. Ce cours utilise la méthode de l'interaction avec les étudiants en se basant sur des questionnaire afin de connaître la difficulté du cours, les points à améliorer..etc. Bien que cette approche soit assez complexe, elle s'avère très utile afin d'améliorer la qualité d'une application d'e-learning.

Chapitre 3

Cahier des besoins

Afin d'avoir une vision globale sur le projet, et de mettre au clair et de hiérarchiser les besoins du client, nous avons réalisé en amont un Cahier des Besoins. Ce document décrit les besoins fonctionnels (fonctionnalités du logiciel, et quelles sont les briques nécessaires pour chaque fonctionnalité), les besoins non fonctionnels (tout ce qui a trait à l'expérience utilisateur et à la qualité globale du logiciel), les différentes contraintes (imposées par le client, ou par les choix technologiques par exemple) et les différents tests mis en place pour contrôler la validité de ces besoins.
Le cahier des besoins contient aussi des diagrammes de conception tels qu'un diagramme de cas d'utilisation, des diagrammes de séquence et des maquettes d'interface.

3.1 Analyse des besoins

3.1.1 Besoins Fonctionnels

À l'ouverture de l'application, une interface s'affiche sur l'écran, proposant à l'utilisateur de lire un texte ou de consulter ses statistiques. Après avoir choisi de lire un texte, l'application sélectionne un texte depuis une base de données en se basant sur le niveau de maîtrise en Anglais de l'utilisateur et l'affiche à l'écran.
L'utilisateur peut ainsi lire le texte. L'application suit les déplacements de ses yeux en enregistrant les mots lus. L'application doit pouvoir reconnaître et analyser ces mots et les ajouter a la base de mots de l'utilisateurs s'ils n'y sont pas déjà. Notre application doit générer aux utilisateurs des statistiques sur l'évolution du nombre de nouveaux mots appris après chaque lecture.
L'application doit aussi permettre à l'utilisateur d'ajuster certains paramètres (tailles de la police et de l'interligne par exemple) et de recalibrer la détection du regard à la volée.

Note : Après recherches, garantir une précision au mot avec notre matériel est difficilement envisageable. En partant du principe que l'utilisateur "joue le jeu" et essaye de lire "normalement" le texte, on peut se contenter détecter quel paragraphe vient d'être lu, et considérer que tous les mots contenus dans ce paragraphe ont été lus. (C'est une idée du Client, elle est détaillée dans la partie implémentation)

Note : La priorité ou criticité des besoins est exprimée à l'aide de lettres de A à D entre parenthèses, (A) étant le plus prioritaire/critique.

Besoins liés à l'eye tracking :

- (A) Afficher un texte à l'écran pour que l'utilisateur le lise
- (A) Détecter les mots lus par l'utilisateur
  - (A) Récupérer les coordonnées de l'eye tracker pour savoir où l'utilisateur regarde sur l'écran
    - Test : Afficher à chaque instant un point aux coordonnées données par l'Eye Tracker et vérifier qu'il s'affiche bien à l'endroit où l'utilisateur regarde (calibrage)
  - (A) Connaître la position sur l'écran de chaque mot du texte (sans passer par un OCR [1] sur des captures d'écran par soucis de performance)
    - Test : Écrire un programme test qui affiche un texte et un rectangle autour de chaque mot pour être sur que l'on arrive à connaître les positions de chaque mots
- (B) Pouvoir modifier les paramètres de l'Eye Tracker en direct sur la page (ajuster le calibrage à la volée)

Besoins liés à la collecte des données de lecture :

- (B) Compter les mots lus par l'utilisateur
  - (B) Faire une liste des différents mots lus
  - (B) Compter pour chaque mot, le nombre de fois qu'il a été lu
  - (C) Regrouper les mots suivant certaines règles en fonction de leurs sens
    - (D) Détecter les mots conjugués et les compter tous comme un seul mot qui sera leur infinitif. par exemple si l'utilisateur lit les mots mangera, mange, manger et mangeait, on retiendra qu'il aura lu le mot manger quatre fois
    - (D) Détecter les mots conjugués et compter les différentes utilisations des temps de conjugaison
  - Test : Extraire les paramètres ci-dessus de plusieurs textes (manuellement ou à l'aide d'un script), et faire lire ces textes à des testeurs. Comparer les données générées par notre logiciel aux données extraites du texte lui même

Besoins liés à l'aide pour l'apprentissage d'une langue :

- (B) Choisir automatiquement un texte à présenter à l'utilisateur en fonction d'une estimation de son niveau
  - (B) Évaluer le niveau de l'utilisateur à partir des données de lectures collectées
  - (B) Être capable de stocker entre 200 et 300 textes classés en plusieurs niveaux de difficulté de lecture
  - (B) Choisir le texte le plus pertinent parmi tous les textes appartenant au même niveau de difficulté que celui de l'utilisateur
    Concernant la définition des niveaux de difficulté ainsi que les textes à stocker et classer, nous ne pouvons pas être plus précis pour le moment. En effet, nous

1. Optical Character Recognition

devons encore nous entretenir avec le client pour décider de la manière dont nous allons procéder pour cette partie des besoins fonctionnels.

— Tests : Essayer avec des "profils" d'utilisateurs de différents niveaux pour voir si les textes proposés ne sont ni trop faciles ni trop difficiles puis faire tester l'application à un panel d'au moins dix testeurs qui devront répondre à un questionnaire entre chaque texte pour savoir si celui-ci est bien choisi. Ces testeurs devront avoir des niveaux de maîtrise de l'anglais très variés afin de vérifier la pertinence de l'évaluation de leur niveau et du choix des textes sur l'ensemble des niveaux de difficultés disponibles.

Besoins liés à la gestion du profil de l'utilisateur :

- (B) Permettre à l'utilisateur de créer un compte
  - (B) Stocker dans une base de données les différents utilisateurs inscrits
  - (C) Assurer un chiffrage des mots de passes
    Le client ne place pas la sécurité dans les besoins les plus prioritaires.
- (B) Permettre à l'utilisateur de s'identifier
- (C) Calculer des statistiques à partir des données collectées lors de la lecture d'un texte
  - Nombre de mots lus
  - Nombre de mots "connus" (on peut considérer qu'à partir d'un certain nombre de fois que le mot a été lu, il est connu. A corréler avec les "fixations" : si l'utilisateur reste plus longtemps qu'en moyenne sur un mot, il est probable qu'il ne le connaisse pas encore bien - à corréler avec la taille du mot)
  - Vitesse moyenne de lecture
  - Courbe d'apprentissage : Représenter sous forme de graphique les différentes statistiques, évoluant en fonction du temps
- (C) Permettre à l'utilisateur de visualiser les statistiques

3.1.2 Besoins non fonctionnels

Facilité d'installation : Rendre l'installation du logiciel et du matériel simple pour le nouvel utilisateur. Test : Faire installer l'ensemble à au moins 10 personnes ayant des niveaux de connaissances en informatique variés. Nous vérifierons s'ils ont réussi l'installation, en combien de temps et nous leur demanderons de répondre à un questionnaire pour avoir leur ressenti sur le déroulement de l'installation.

Interface Intuitive : Un nouvel utilisateur doit être capable de naviguer entre les différentes fonctionnalités facilement, et sans formation initiale (tutoriel) . Test : Montrer l'interface à des testeurs, leur dire de chercher une fonctionnalité, mesurer le temps qu'il leur faut ainsi que le nombre d'opérations et proposer un questionnaire à la fin pour avoir un retour personnel.

Précision : Le logiciel doit être capable d'identifier à chaque fois quel paragraphe vient d'être lu .

Test : Faire lire a 10 testeurs 2 textes chacun, et vérifier que chaque paragraphe lu se surligne bien.

Pseudo temps réel de la détection des paragraphes lus : Dès que l'utilisateur à fini de lire un paragraphe, on doit pouvoir le détecter et afficher un retour visuel en moins d'une seconde.

Pertinence des suggestions de textes : Les textes proposés à l'utilisateur doivent être cohérents avec son niveau et son évolution, afin de l'accompagner dans son apprentissage.

3.2 Contraintes

Une des contraintes fortes est l'Eye Tracker en lui même. Le modèle fourni par le client, et avec lequel l'application doit fonctionner est un Steelseries Sentry, qui utilise la technologie TobiiGaming EyeX. Nous devons donc utiliser le SDK d'EyeX qui n'est disponible que sur Windows 7, 8.1 et 10.

Le client souhaite aussi que l'application se présente sous la forme d'un site web, nous avons donc étudié la faisabilité d'une telle application avec des technologies web en développant un prototype (cf section Prototype). Ce dernier prouvant bien que c'est possible, nous développerons l'application en web.

3.3 Prototype

Afin de prouver la faisabilité de l'application en utilisant des technologies web, nous avons réalisé un prototype permettant d'afficher, dans une page web contenant un texte, les positions du regard de l'utilisateur, et de détecter les mots que ce dernier lis.
Notre prototype utilise un serveur Node.js [2] **local**, qui génère une page web HTML contenant un texte dont chaque mot est dans une balise "span".
Nous avons utilisé la bibliothèque GazeJs [3] qui permets d'accéder au SDK de l'Eye Tracker [4] en Node.js.

On peut ainsi accéder aux positions du regard donnée par l'Eye Tracker côté serveur. Pour récupérer ces données côté client, nous avons utilisé Socket.io [5] qui permet une communication bidirectionnelle en temps réel entre client et serveur. Ainsi, nous sommes capables d'afficher en temps réel des curseurs sur la page web, un curseur pour l'œil droit ("R") et un curseur pour l'œil gauche ("L").

2. `http://nodejs.org`
3. `http://github.com/jiahansu/GazeJS`
4. `http://developer.tobii.com/eyex-sdk/`
5. `http://socket.io`

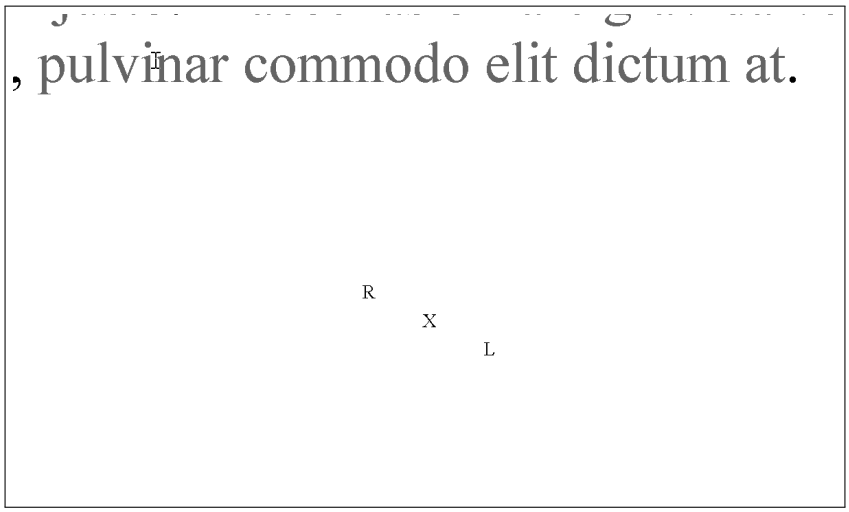


FIGURE 3.1 – Curseurs

Une difficulté intrinsèquement liée à la technologie utilisée par notre Eye Tracker (reflection d'infra rouges par la cornée dans une caméra infra-rouge) [6] est que les données brutes envoyées par le périphérique sont très bruitées. Nous avons donc choisi d'implémenter une simple algorithme qui moyenne les x dernières valeurs brutes afin d'avoir des coordonnées exploitables pour effectuer des tests de collisions avec les mots.
Cette manière de filtrer les données est très naïve mais elle est satisfaisante pour notre prototype.

Une autre difficulté découlant de l'utilisation des technologies web est l'incertitude concernant la façon dont va être affiché l'application dans la fenêtre du navigateur internet et même dans l'écran de l'utilisateur.
En effet, GazeJs produit des coordonnées absolues dans l'écran de l'utilisateur. Il faut prendre en compte (entre autres) la possibilité que la fenêtre du navigateur internet de l'utilisateur n'est pas forcement en plein écran.
Il se trouve qu'en Javascript, lorsqu'on détecte un mouvement de la souris, on peut connaître les coordonnées du curseur relatives à l'écran ainsi que les coordonnées du curseur relatives au viewport (la partie du navigateur qui affiche la page web). Il suffit alors de calculer la différence entre ces deux coordonnées pour connaître la position de la fenêtre dans l'écran, et prendre en compte le décalage lors des calculs de collisions avec les mots.

Nous avons choisi de changer la couleur des mots en fonction de la distance Euclidienne à laquelle ils sont du curseur moyenné ("X"). Au dessus de 200px, les mots sont gris. En dessous de 200px, les mots passent de bleu à rouge suivant un dégradé en fonction de leur distance au curseur.

6. `http://www.tobiipro.com/learn-and-support/learn/eye-tracking-essentials/how-do-tobii-eye-trackers-work/`

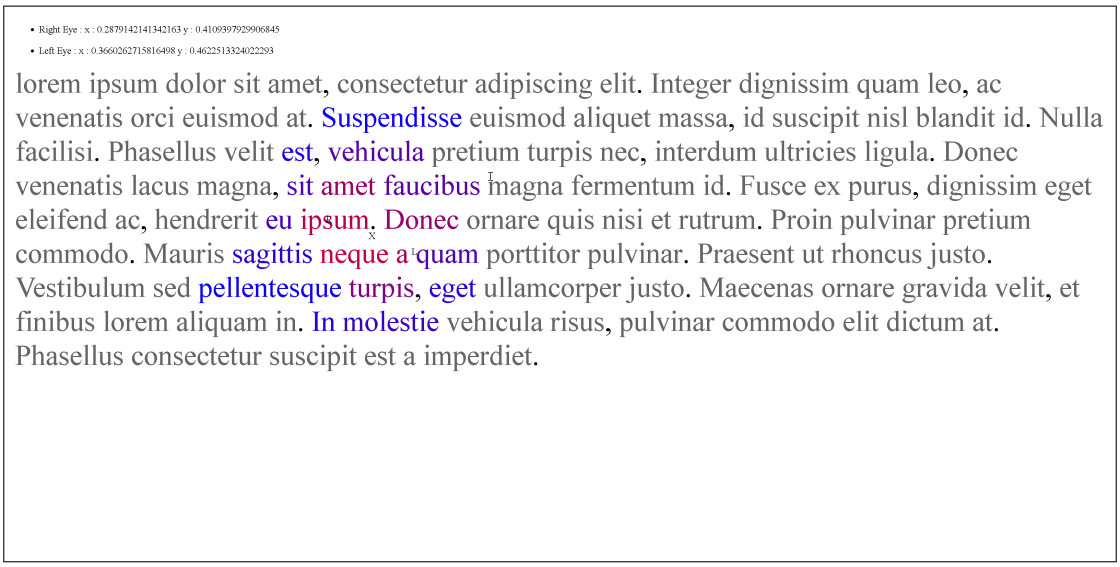


FIGURE 3.2 – Interface du prototype

La réalisation de ce prototype nous a permis de confirmer la faisabilité d'une telle application en web. Nous avons pu nous rendre compte des difficultés qu'engendrent le fait de travailler avec un périphérique externe non standard, et plus particulièrement avec un Eye Tracker grand public, dont la précision n'est pas toujours excellente, notamment à cause de sa grande sensibilité aux changements de conditions d'utilisation. Lorsque l'Eye Tracker est bien calibré, et que l'utilisateur ne bouge pas de la position dans laquelle il a effectué la procédure de calibration, la précision est tout à fait acceptable, et nous obtenons des résultats satisfaisants. Mais si l'utilisateur ou le périphérique bouge, on remarque des décalages importants, surtout sur l'axe vertical.

Ce problème de robustesse à la position semble avoir été corrigé dans la nouvelle version des Eye Trackers Tobii, les **Tobii Eye Tracker 4C**[7] grâce à un système de suivi de la tête, ou *head tracking*. Le pilote peut ainsi compenser les changements de position de l'utilisateur en sachant ou se trouve sa tête dans l'espace.

Connaissant ce problème, et sachant qu'il est très difficile à résoudre sans *head tracking*, nous avons pu revoir avec le client le besoin de précision que nous avions défini, et ainsi définir un objectif plus réaliste et atteignable pour la version finale du projet.

7. `http://tobiigaming.com/eye-tracker-4c/`

Chapitre 4

Architecture

Ce chapitre détaille l'architecture de l'application à travers des diagrammes de cas d'utilisation, de séquence et de classe. Une description des technologies utilisées est aussi incluse à la fin du chapitre.

4.1 Diagrammes

4.1.1 Diagramme de cas d'utilisation global

- **L'administrateur** qui est le web master de l'application, doit gérer les utilisateurs en ajoutant, modifiant ou supprimant un utilisateur. Il doit aussi gérer la base de données des textes en ajoutant, modifiant ou supprimant un texte.
- Pour accéder à l'application, un utilisateur doit créer un compte pour avoir un espace personnel sécurisé lui permettre de lire les textes proposés par le système. Après la création d'un compte sur l'application, **L'utilisateur** peut consulter le manuel d'installation , il peut aussi modifier les paramètres du eye tracker et commencer à lire les textes.Après chaque lecture, l'utilisateur peut consulter les statistiques fournis par le système afin de mettre à jour le niveau du lecteur.
- Notre **système** doit permettre aux utilisateur d'avoir un espace personnel sécurisé et leurs permettre de lire des textes en utilisant le Eye tracker, en premier lieu le système doit détecter les mots déjà lus par un utilisateur, en deuxième lieu arriver à calculer le score des mots lus par cet utilisateur et fournir les statistiques à travers lesquelles le niveau du lecteur va évoluer après chaque lecture.

Le diagramme de cas d'utilisation global présenté dans **la figure 4.1** modélise les rôles de chaque acteur dans l'application :

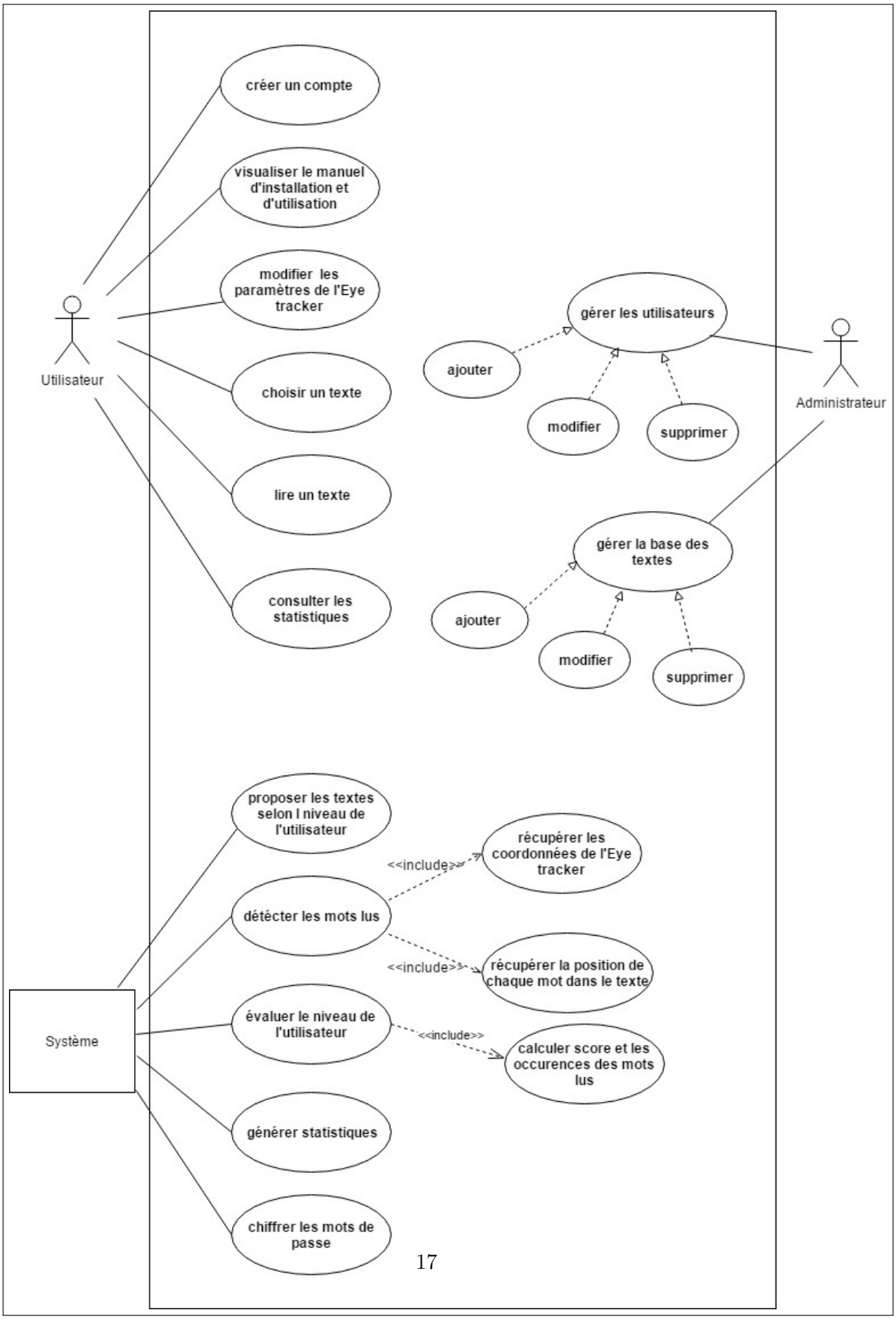


FIGURE 4.1 – Le diagramme de cas d'utilisation global

4.1.2 Diagrammes de classes

Voici un schéma représentant notre architecture dans sa globalité.

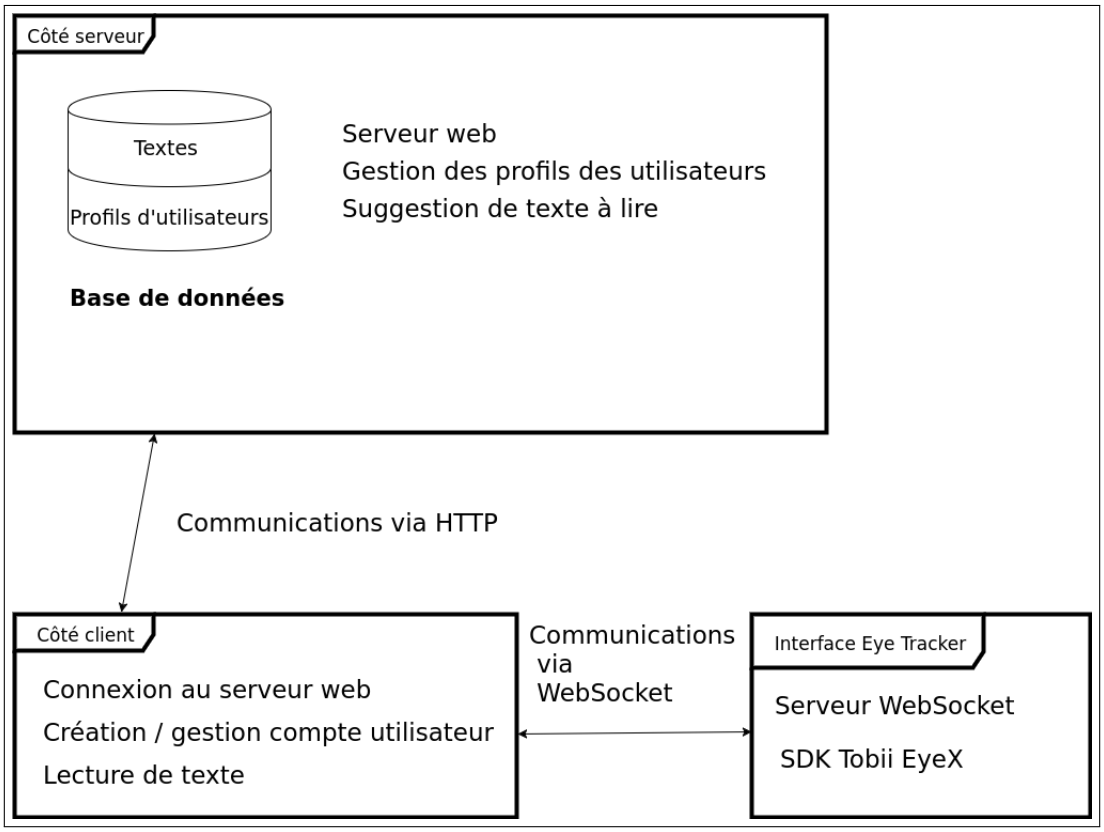


FIGURE 4.2 – Schéma global de notre architecture

Puisque nous avons décidé de faire une application web, nous avons une architecture client-serveur. En effet, nous avons un serveur distant qui s'occupe de gérer la base de données (profils utilisateurs et textes) et qui permet aux clients d'accéder au site.

La partie client permet de présenter notre application à l'utilisateur. Sur cette partie, l'utilisateur peut s'inscrire, se connecter, consulter son profil et lire des textes. Ces trois premières tâches consistent tout simplement à envoyer des requêtes à la partie serveur et afficher le résultat.

La partie "Interface Eye Tracker" sert à faciliter la communication entre l'eye tracker et l'application client. En effet, le SDK proposé par Tobii est très contraignant. Il n'est pas possible d'y accéder directement en javascript depuis un navigateur web. Notre client avait une préférence pour une application web et nous avions réussi à faire un prototype en utilisant un serveur websocket local qui lui pouvait utiliser le SDK EyeX et communiquer les coordonnées de l'eye tracker au navigateur par websocket. De plus, il est intéressant de préciser que l'utilisation d'une telle interface permet théoriquement d'adapter notre

application à n'importe quel type d'eye tracker. Il suffirait pour cela de l'adapter pour qu'elle puisse communiquer avec l'eye tracker voulu.

Partie client

Concernant le côté client, la lecture de texte est une des parties les plus délicates. Lorsque l'utilisateur veut lire un texte, le client doit l'afficher et communiquer avec l'Eye Tracker afin de détecter ce qui est lu par l'utilisateur. Il s'agit d'une partie complexe qui doit interagir avec l'utilisateur, l'Eye Tracker et réaliser un traitement en temps réel. Nous avons donc décidé de l'implémenter en suivant le design pattern MVC (cf. figure 4.3).

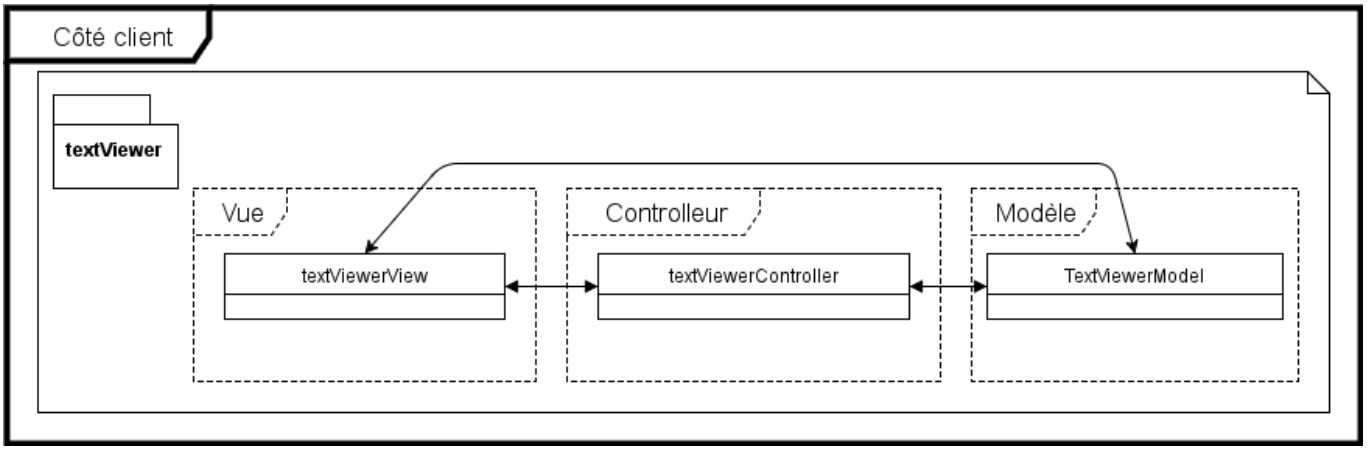


FIGURE 4.3 – Schéma global du MVC côté client

Dans la partie modèle, nous avons une structure d'arbre à trois niveau pour représenter le texte à lire : un objet de la classe TextModel, qui peut contenir plusieurs objets de la classe ParagraphModel et qui peuvent à leur tour contenir plusieurs objets de la classe WordModel (cf. figure 4.4).

Cette structure nous permet de parcourir facilement les différentes parties du texte et de marquer facilement les mots et les paragraphes qui ont été lus.

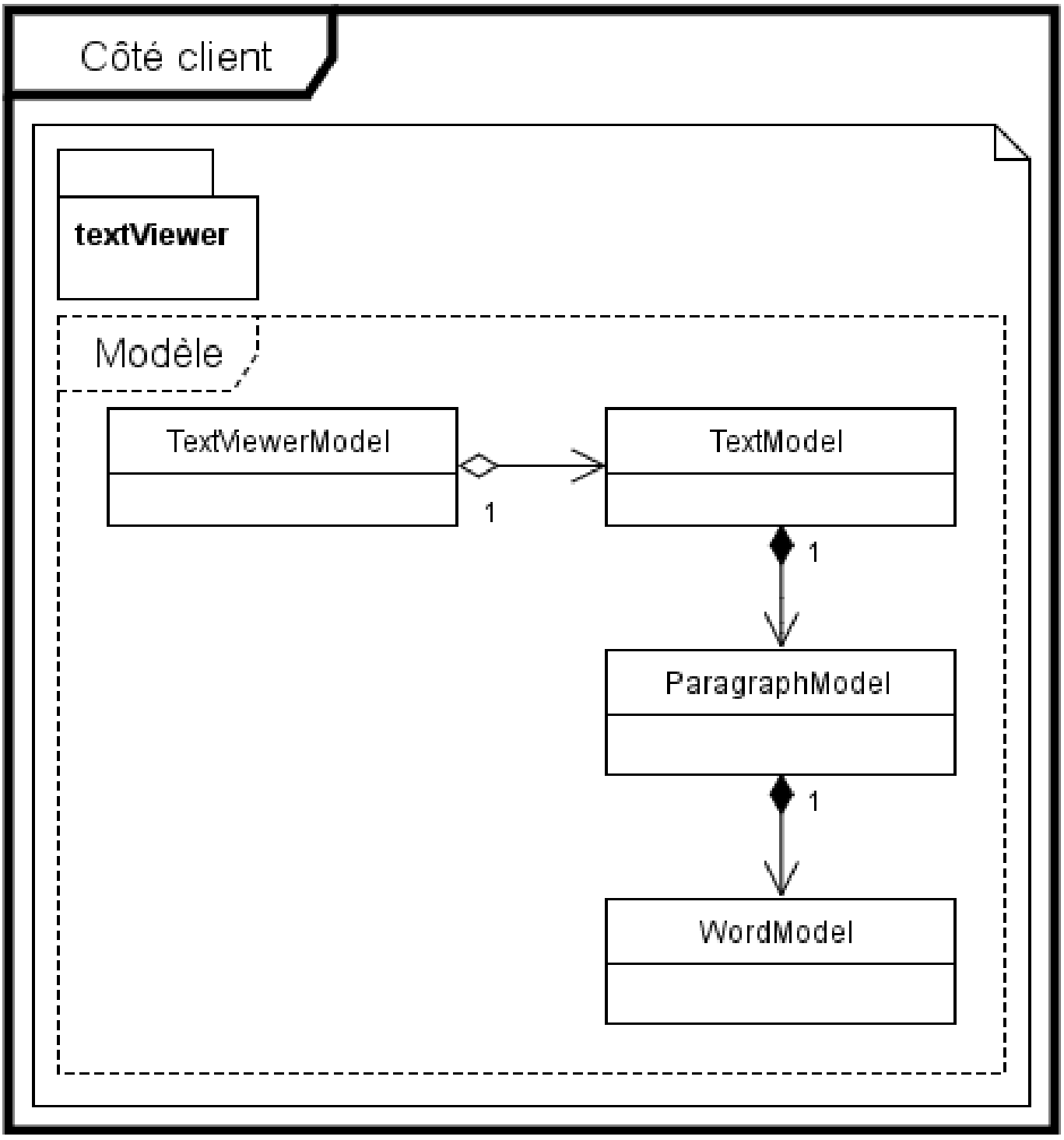


FIGURE 4.4 – Diagramme de classes du modèle

Dans la partie vue, nous avons une structure similaire avec les classes TextView, ParagraphView et WordView qui permettent d'afficher le texte à partir de notre structure de donnée du modèle (cf. figure 4.5). Chaque instance de WordModel, ParagraphModel ou TextModel sera affichée respectivement grâce à une instance de WordView, ParagrapheView ou TextView.

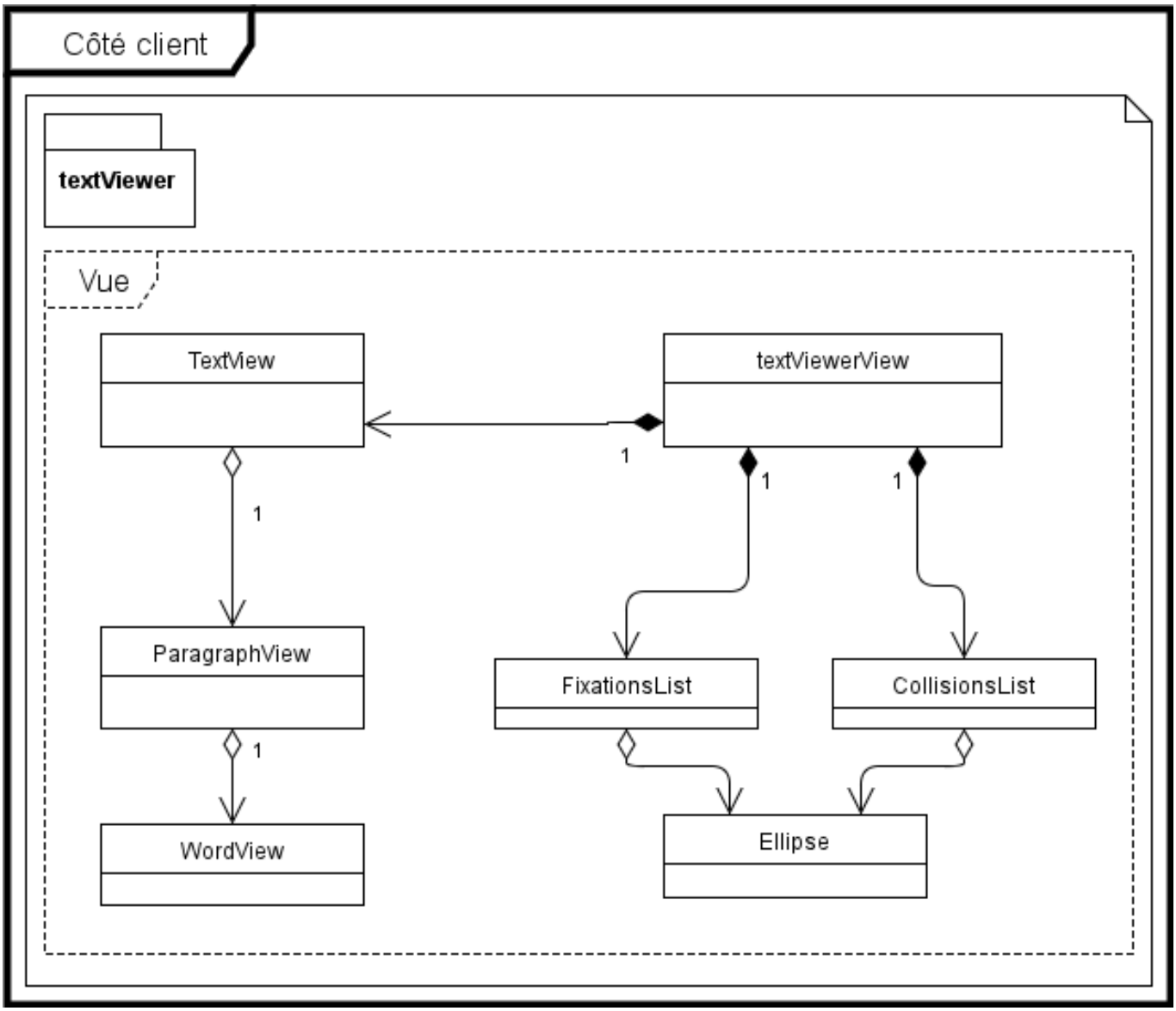


FIGURE 4.5 – Diagramme de classes de la vue

Le contrôleur (cf. figure 4.6) s'occupe des interactions avec l'utilisateur et l'Eye Tracker pour mettre à jour le modèle. Il contient une classe "Eye Tracker Interface" qui s'occupe de communiquer avec le serveur local afin de récupérer les coordonnées de l'Eye Tracker

et de les filtrer.

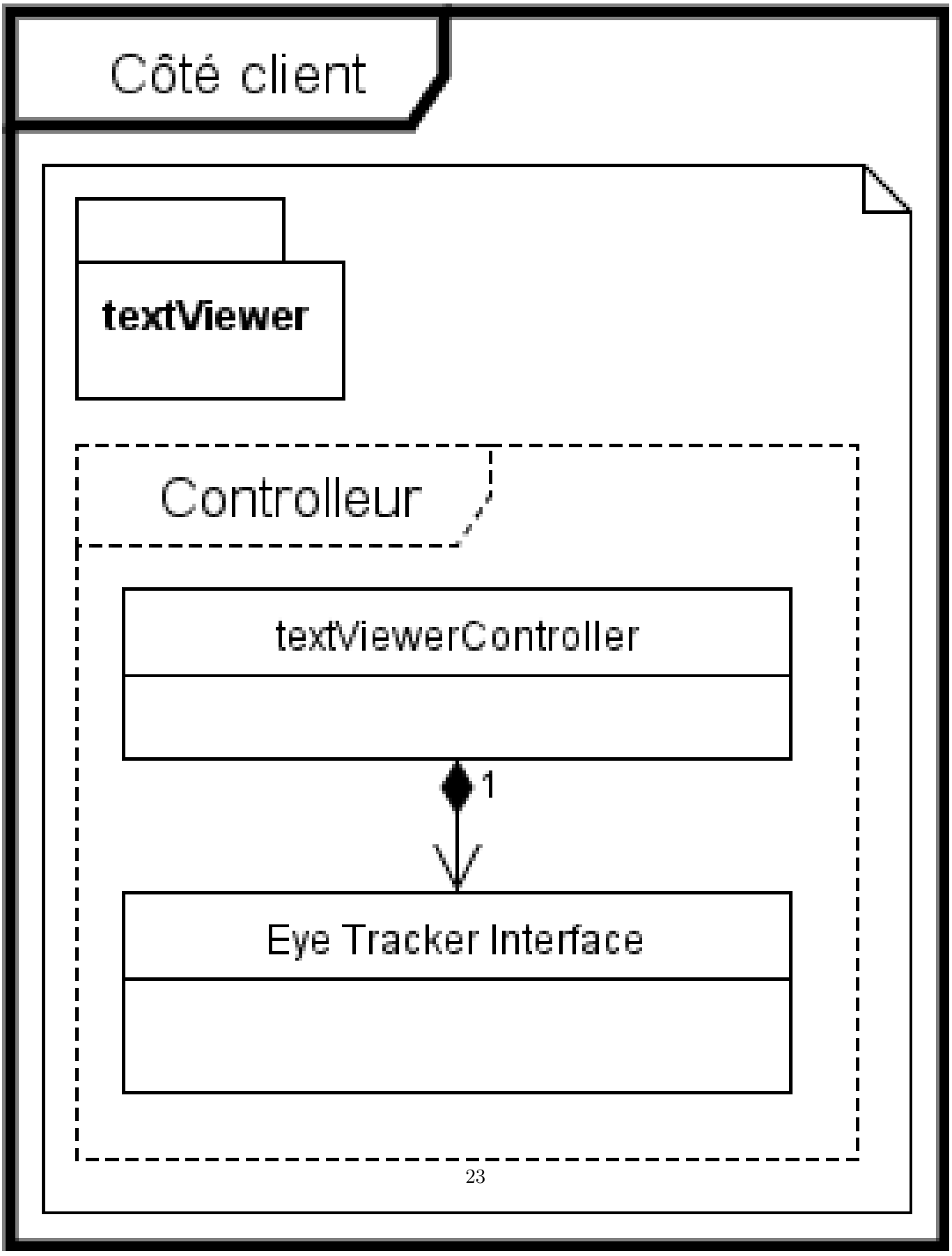


FIGURE 4.6 – Diagramme de classes du contrôleur

Partie serveur

La partie serveur se contente de répondre aux requêtes du client, lui fournir des données telles qu'une page web, un texte à lire, des statistiques ou encore récupérer des données envoyées en fin de lecture d'un texte.

Le serveur comporte donc un ensemble de procédures de génération de pages web, d'opérations sur la base de données et traitement de données tels que la modification du profil de lecture d'un utilisateur à partir des données de lecture d'un texte ou la sélection d'un texte à lire en fonction du niveau et du profil de l'utilisateur.

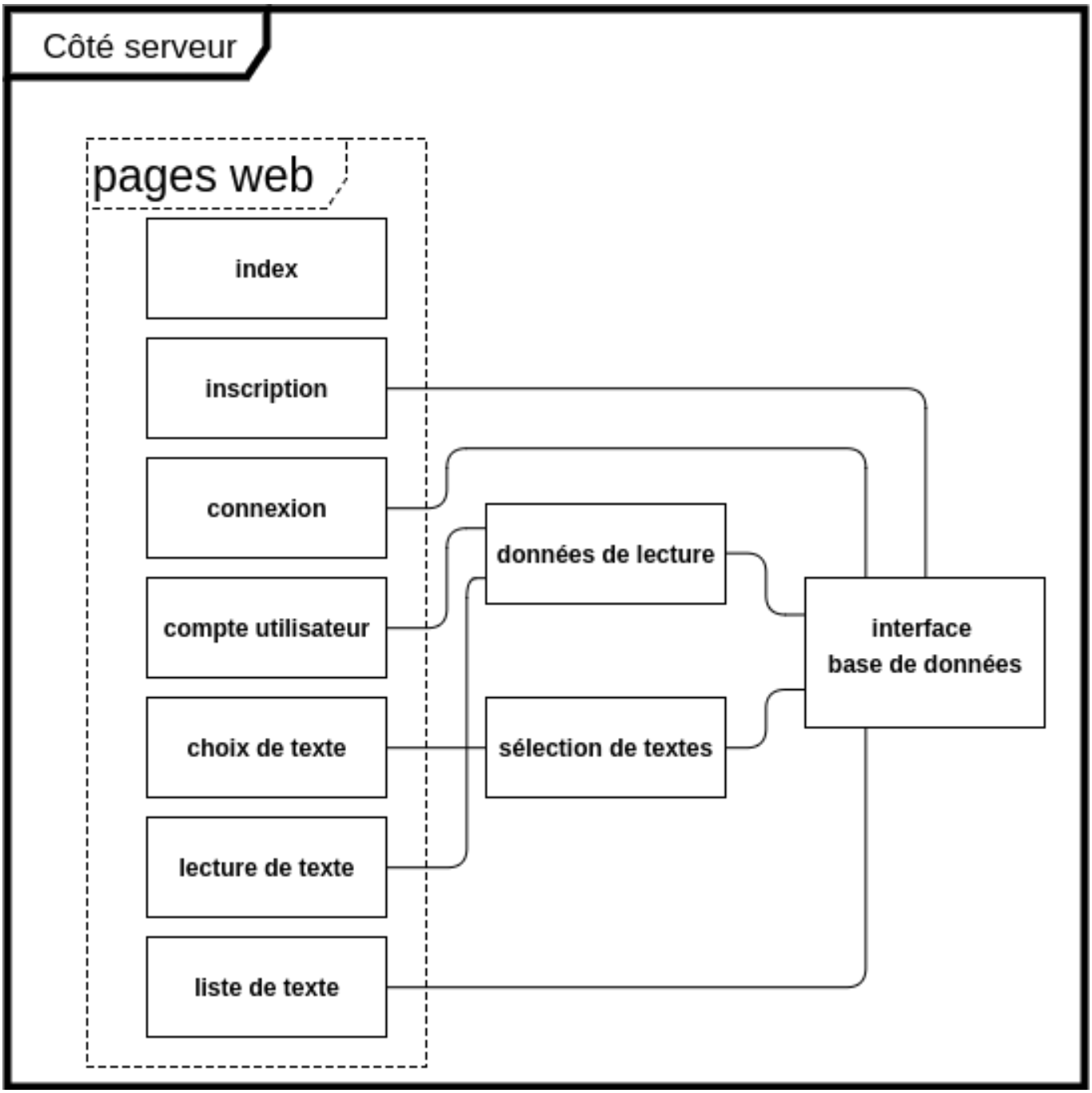


FIGURE 4.7 – Diagramme de classe de la partie serveur

Sur la figure 4.7, on peut voir que nous avons un ensemble de procédures de génération de pages web.

Nous avons également une interface pour la base de donnée qui contient les informations

de connexion à celle-ci ainsi que l'ensemble des requêtes.

On peut voir que nous avons un module "données de lecture" qui permet de récupérer des données brutes depuis la base de données et les transformer en statistiques à afficher sur la page du compte utilisateur et qui sert également dans l'autre sens à récupérer les données d'une session à la fin de la lecture d'un texte afin de les traiter avant de les ajouter à la base de données.

Le module "sélection de textes" va récupérer dans la base de données les informations sur l'utilisateur et la base de textes et les traiter afin de sélectionner les textes à proposer sur la page "choix de texte".

4.1.3 Diagrammes de séquences

Diagramme de séquences «Créer compte»

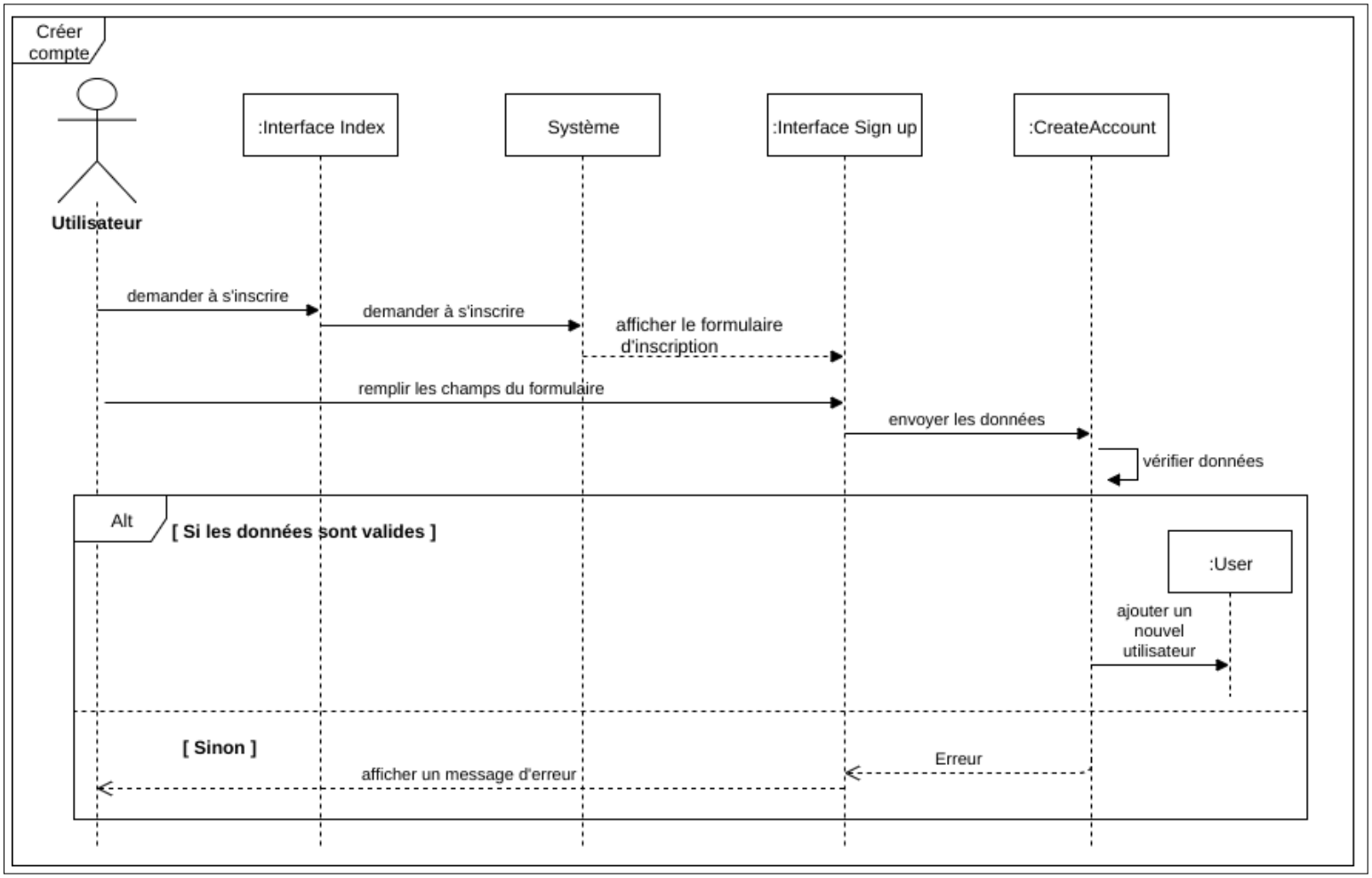


FIGURE 4.8 – Le diagramme de séquences «créer compte»

Dans le tableau ci dessous, nous présentons la description textuelle du cas d'utilisation «Créer compte» :

Cas d'utilisation	**Créer compte**
Acteur	Utilisateur
But	Création d'un compte sur l'application
Pré-condition	L'utilisateur n'a pas encore un compte sur l'application ou bien il veut créer un autre compte.
Post-condition	L'utilisateur peut accéder à l'application à travers son espace personnel.
Scénario principal	1-L'utilisateur demande de créer un compte. 2-Le système affiche le formulaire d'inscription. 3-L'utilisateur remplit les champs du formulaire. 4-Le système vérifie les champs. 5-Si les champs sont valides. 5.1- Le système ajoute un nouvel utilisateur.

Diagramme de séquences «s'authentifier»

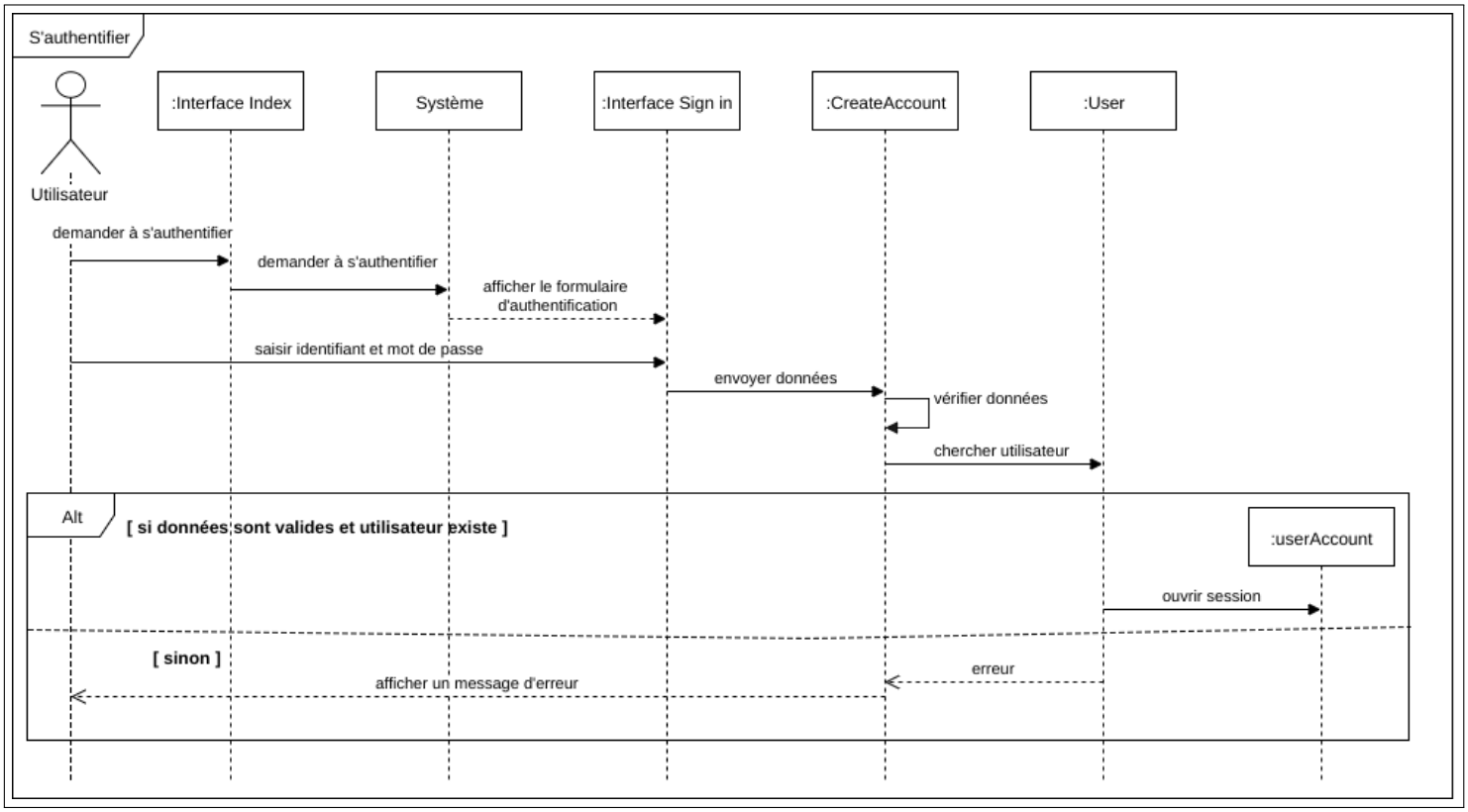


FIGURE 4.9 – Le diagramme de séquences «s'authentifier»

Dans le tableau ci dessous, nous présentons la description textuelle du cas d'utilisation «s'authentifier» :

Cas d'utilisation	**s'authentifier**
Acteur	utilisateur
But	Authentification d'un utilisateur
Pré-condition	L'utilisateur doit avoir un compte.
Post-condition	Accès à son espace privé.
Scénario principal	1- L'utilisateur demande l'accès au système. 2- Le système affiche le formulaire d'authentification. 3- L'utilisateur remplit les champs du formulaire. 4- Le système vérifie les champs et cherche l'utilisateur dans la base de données. 5- Si les champs sont valides. 5.1- Le système autorise l'utilisateur.

Diagramme de séquence «choisir un texte»

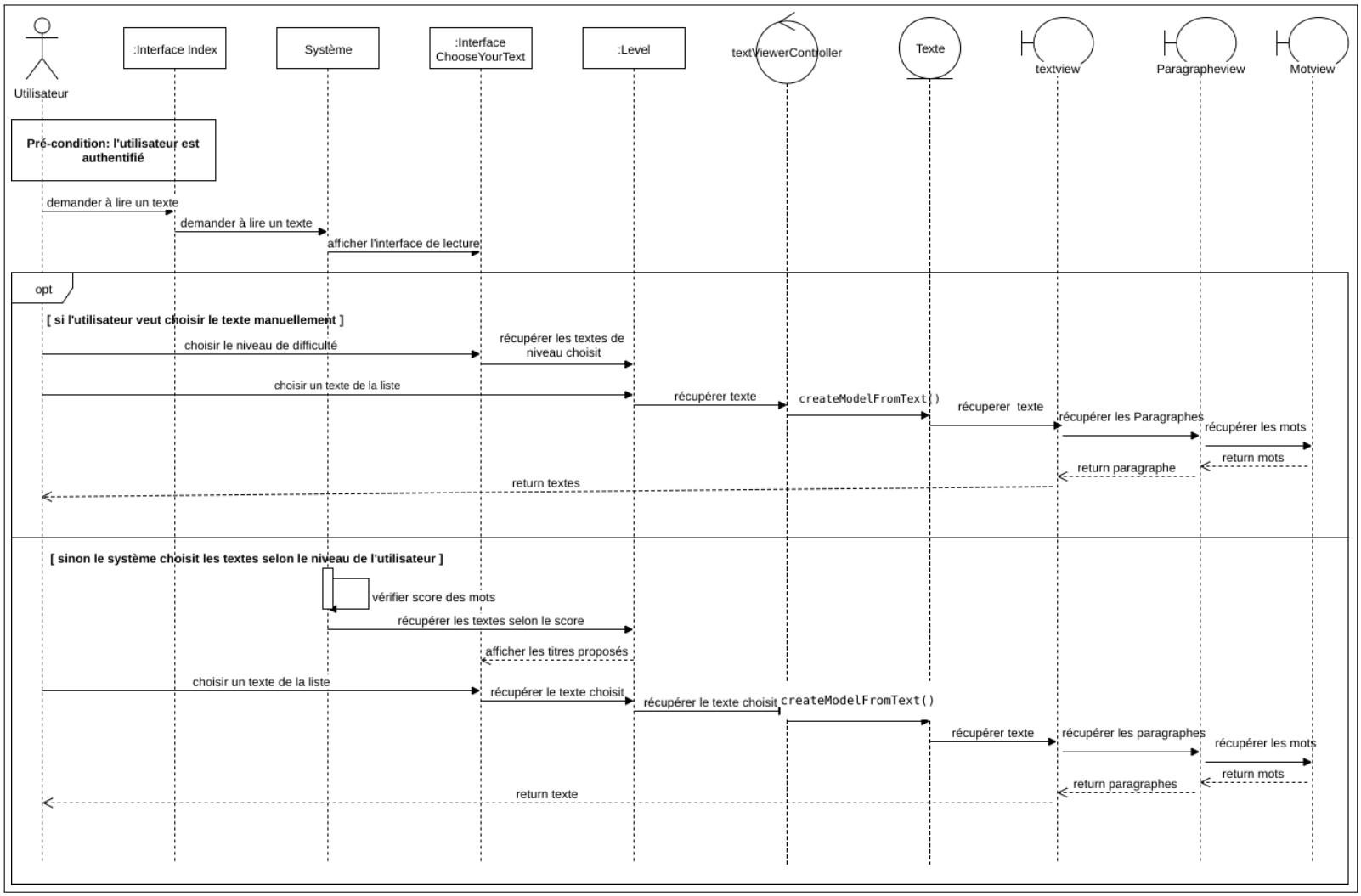


FIGURE 4.10 – Le diagramme de séquence «choisir un texte»

Dans le tableau ci dessous, nous présentons la description textuelle du cas d'utilisation «choisir un texte» :

Cas d'utilisation	**choisir un texte**
Acteur	utilisateur
But	choisir un texte pour le lire
Pré-condition	L'utilisateur doit être authentifié.
Post-condition	le système affiche le texte.
Scénario principal	1-L'utilisateur demande de lire un texte. 2-Le système affiche l'interface de lecture. 3-Si Le système va choisir automatiquement les textes selon le niveau de l'utilisateur. 3.1-Le système vérifie le score des mots de l'utilisateur. 3.2-Le système affiche les textes proposés selon le niveau de l'utilisateur. 3.3-L'utilisateur choisit un texte parmi les textes choisis. 3.4-Le système affiche le texte.

4.1.4 Architecture de la base de données

Notre base de données s'organise comme décrit sur la figure 4.11.

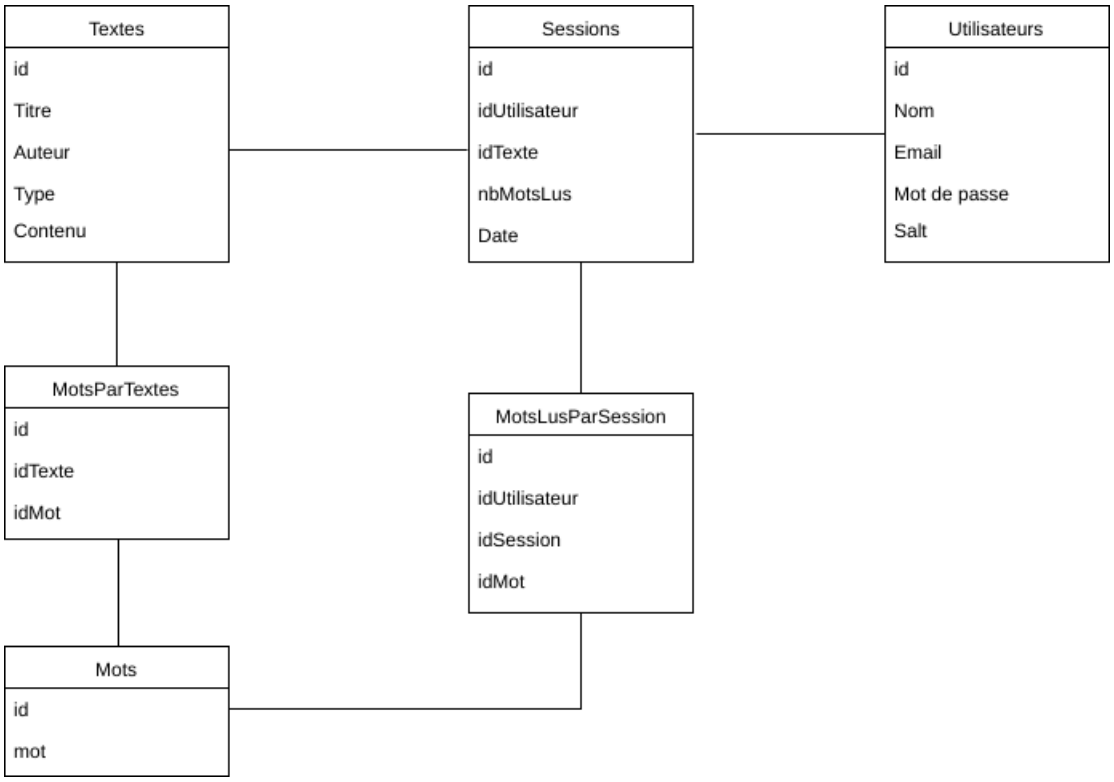


FIGURE 4.11 – Architecture de la base de données

A chaque fin de session (quand l'utilisateur a terminé de lire un texte), on enregistre les informations dans la table *Sessions*, qui est une table d'association entre les table *Textes* et *Utilisateurs*. Cette table nous permet de savoir quels textes ont été lu par l'utilisateur, et quand.

Dans la table *Textes*, nous stockons des informations à propos du texte, sa difficulté (champ Type), ainsi que le texte en lui même (champ Contenu).
La table *MotsLusParSession* est aussi une table d'association entre les tables *Sessions* et *Mots*. On peut ainsi connaître les identifiants des mots lus lors de chaque session.
La table *Mots* contient une liste de mots Anglais qui sont passés par l'algorithme de *racinisation* (*Stemming*) de Porter[10], qui permet d'extraire les racines des mots. Ce procédé facilitera la comparaison des mots des textes, qui passeront aussi par cet algorithme avant d'être comparés avec les mots de la liste.
La table *MotsParTextes* est une table d'association entre les tables *Mots* et *Textes*. Nous stockons dans cette table les identifiants des mots contenus dans chaque texte, sans dupli-

cation, afin de faciliter et d'accélérer notre algorithme de suggestion de textes. A chaque ajout de texte, chaque mot de ce ce dernier passe par le *Stemmer*, et sont stockés dans la table les identifiants des racines des mots sans doublons.

4.2 Technologies

Côté client

Afin de faire communiquer notre page web avec le SDK de l'Eye Tracker, nous avons choisi d'utiliser un serveur local NodeJS, qui fait tourner la bibliothèque GazeJS [1]. Cette bibliothèque permet d'avoir accès à certaines fonctions du SDK Tobii EyeX en Javascript. Ainsi, notre serveur local récupère les coordonnées du regard de l'utilisateur, et les transmets à la page web via un websocket Socket.IO [2].

Pour l'application web en elle même, nous avons choisi d'utiliser le *framework* Javascript AngularJS [3], qui permet d'implémenter "proprement" le design pattern MVC que nous avons choisi pour l'architecture de notre projet. Nous utilisons bien entendu aussi HTML5 et CSS3.

Côté serveur

Côté serveur, nous utilisons un serveur php et une base de données MySQL. Tous les accès à la base de donnée se font de manière sécurisé via des requêtes préparées PDO. Les mots de passe utilisateurs sont "hashées" avec "sha256" et un salt de 32 bits.

1. `http://github.com/jiahansu/GazeJS`
2. `http://socket.io`
3. `http://angularjs.org/`

Dans le tableau ci dessous, nous décrivons brièvement les langages, *frameworks* et bibliothèques utilisés

Technologies	**Description**
PHP5	Langage de programmation libre utilisé sur l'internet pour produire des pages web dynamiques via un serveur HTTP.
Mysql	Système de gestion de base de données (SGBD) relationnelles très populaire
Xampp	Environnement de développement web, facilitant la mise en place de serveurs web locaux
Javascript	Langage de programmation de scripts, principalement utilisé dans les pages web interactives. C'est un langage orienté objet à prototype.
HTML5	Langage de balisage pour la création de sites internet, il sert à structurer le document.
CSS3	Langage servant à décrire la présentation des documents HTML
Node.js	Plateforme logicielle libre et événementielle en Javascript orientée vers les applications réseau qui doivent pouvoir monter en charge
AngularJS	Framework Javascript fondé sur le principe d'extension du HTML et sur le design pattern MVC
GazeJS	Bibliothèque qui fait l'interface entre le SDK de Tobii EyeX et Javascript

Chapitre 5

Implémentation

Après avoir analysé la partie Architecture, il s'agit désormais de se focaliser sur l'aspect pratique de notre application. Nous spécifions aussi l'environnement de développement supportant notre projet, ainsi que nous allons présenter notre travail en décrivant les principales interfaces de notre application.

Dans ce chapitre, nous détaillons le travail effectué, en précisant les points d'implémentation intéressants.

5.1 Réalisation

Page d'accueil

La page d'accueil est la première page qui s'affiche. Elle comporte une barre de menu contant un lien vers l'accueil qui redirige vers la même page, elle comporte aussi un lien vers la page d'inscription, la page de connexion et la page contant les informations de l'application ainsi que la page contact.

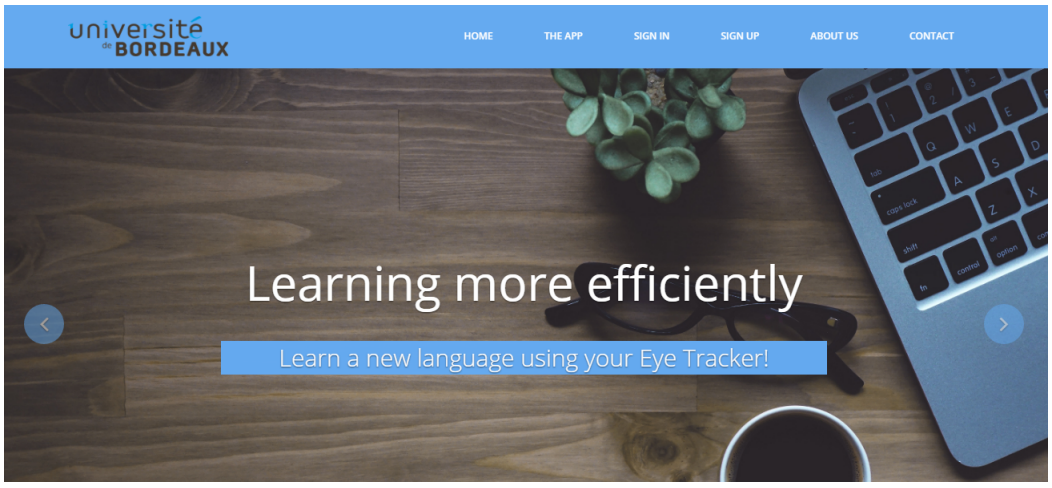


FIGURE 5.1 – La page d'accueil

Création d'un compte sur l'application

Pour créer un compte sur l'application, l'utilisateur doit remplir correctement le formulaire d'inscription et valider sa demande en tapant sur le bouton SUBMIT.

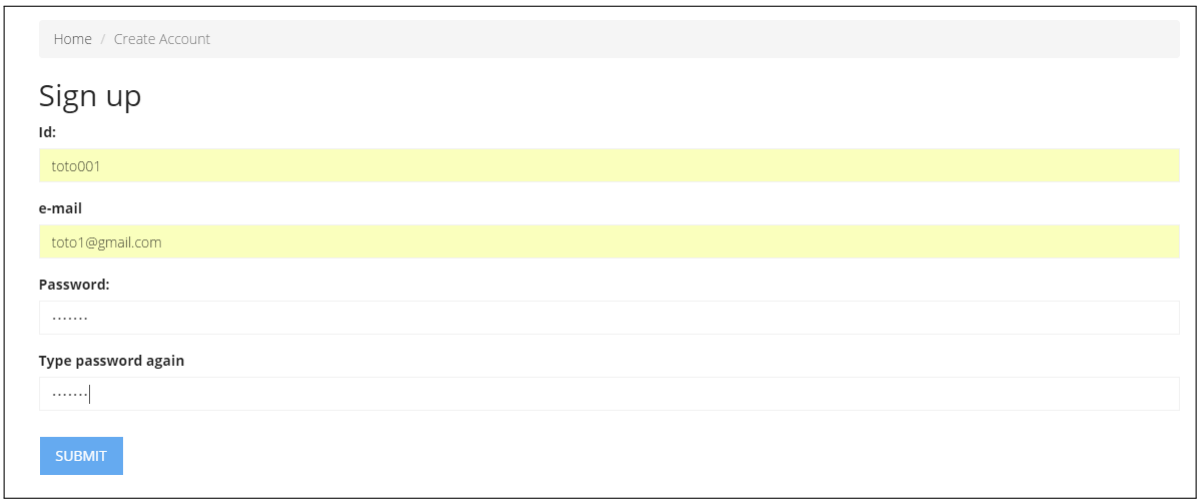


FIGURE 5.2 – Formulaire d'inscription

Authentification

Pour accéder à l'application, l'utilisateur devra d'abord entrer son login et son mot de passe.

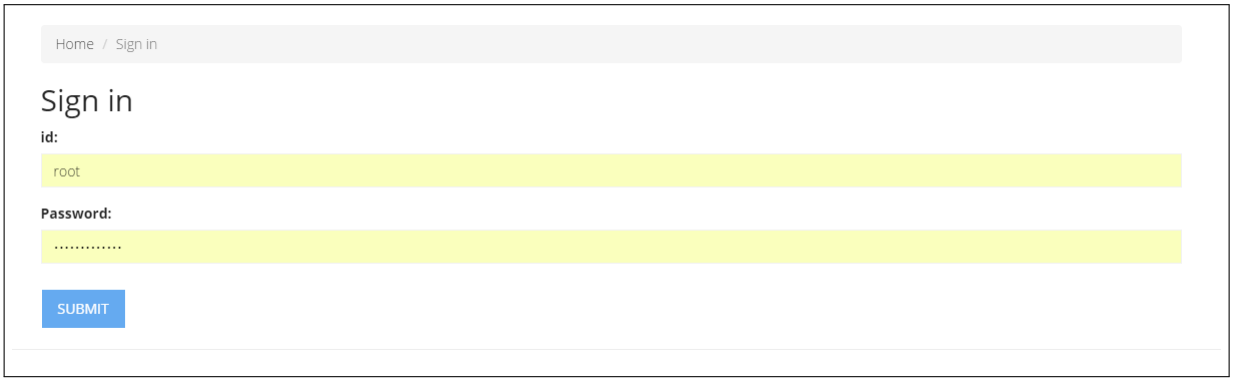


FIGURE 5.3 – Interface d'authentification

Choix du texte à lire

A travers cette interface, l'utilisateur peut soit choisir de lire un des textes suggérés par l'application en fonction des mots qu'il a déjà lu, ou sélectionner manuellement un niveau

de difficulté et choisir un texte parmi les textes affichés.

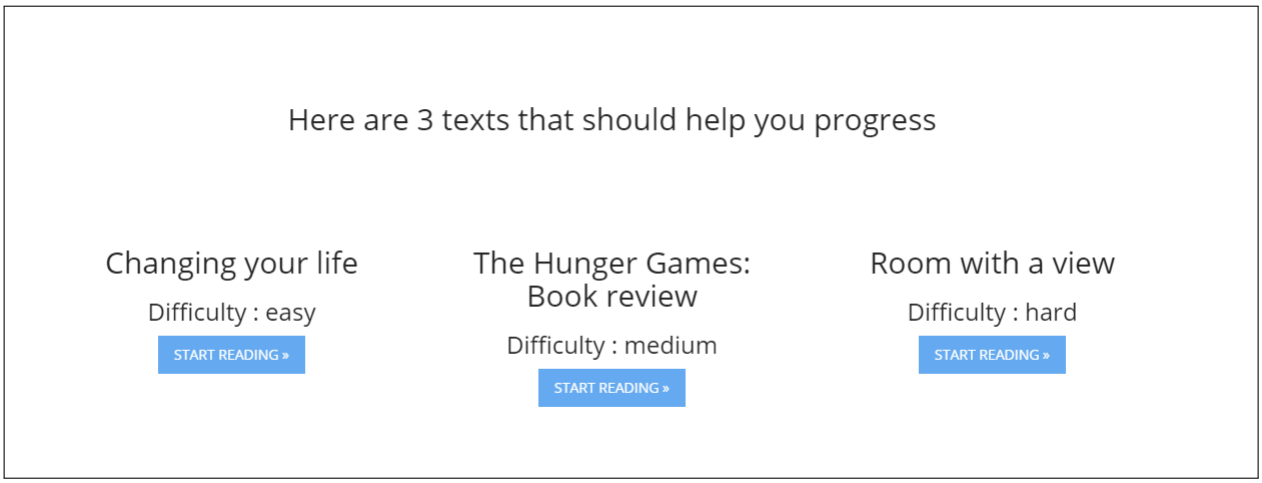


FIGURE 5.4 – interface de suggestion des textes

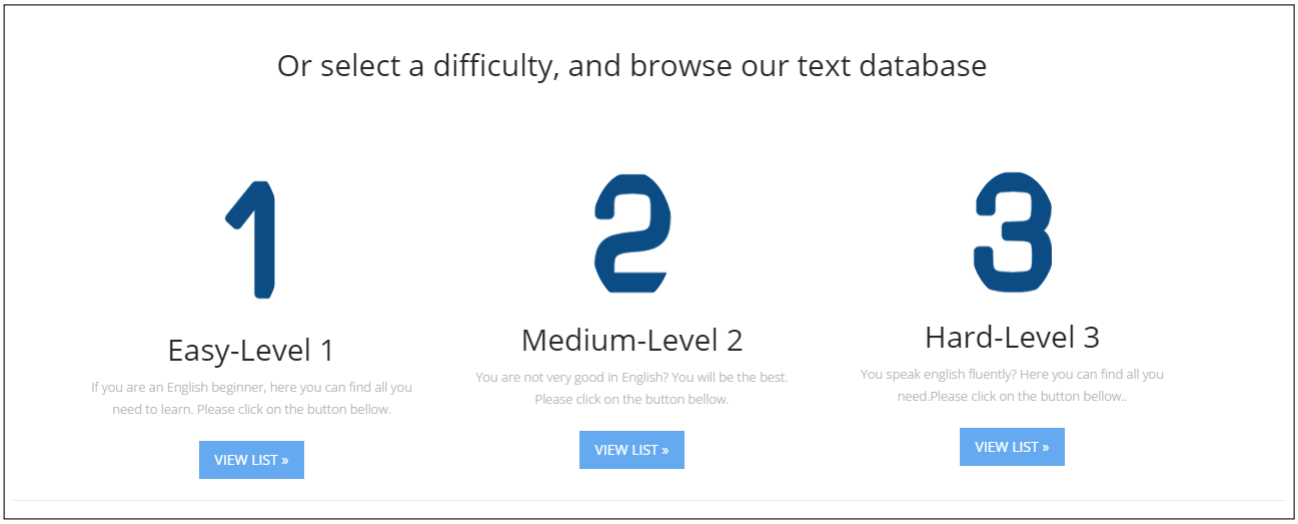


FIGURE 5.5 – Interface du choix du niveau de texte à lire

Liste des textes

Si l'utilisateur à choisi un niveau de difficulté manuellement, on affiche la liste des textes correspondants à cette difficulté (ci-dessous, une partie de la liste de textes de difficulté *facile*

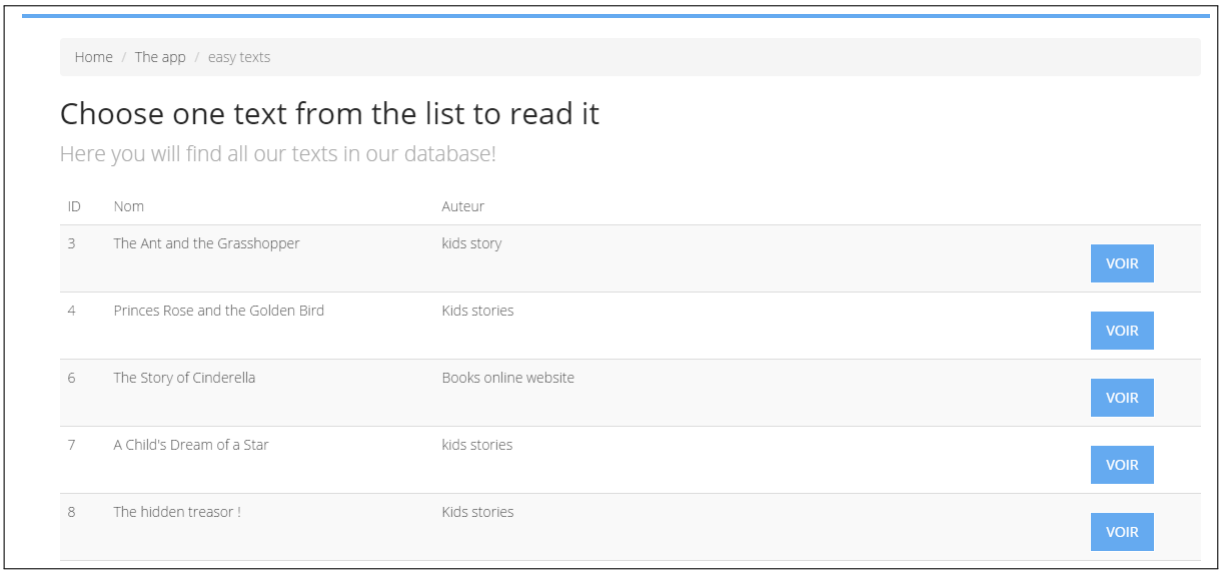


FIGURE 5.6 – Interface d'affichage de textes de difficulté *facile*

Modifier les paramètres

L'interface ci dessous présente les différents paramètres qu'il est possible de modifier

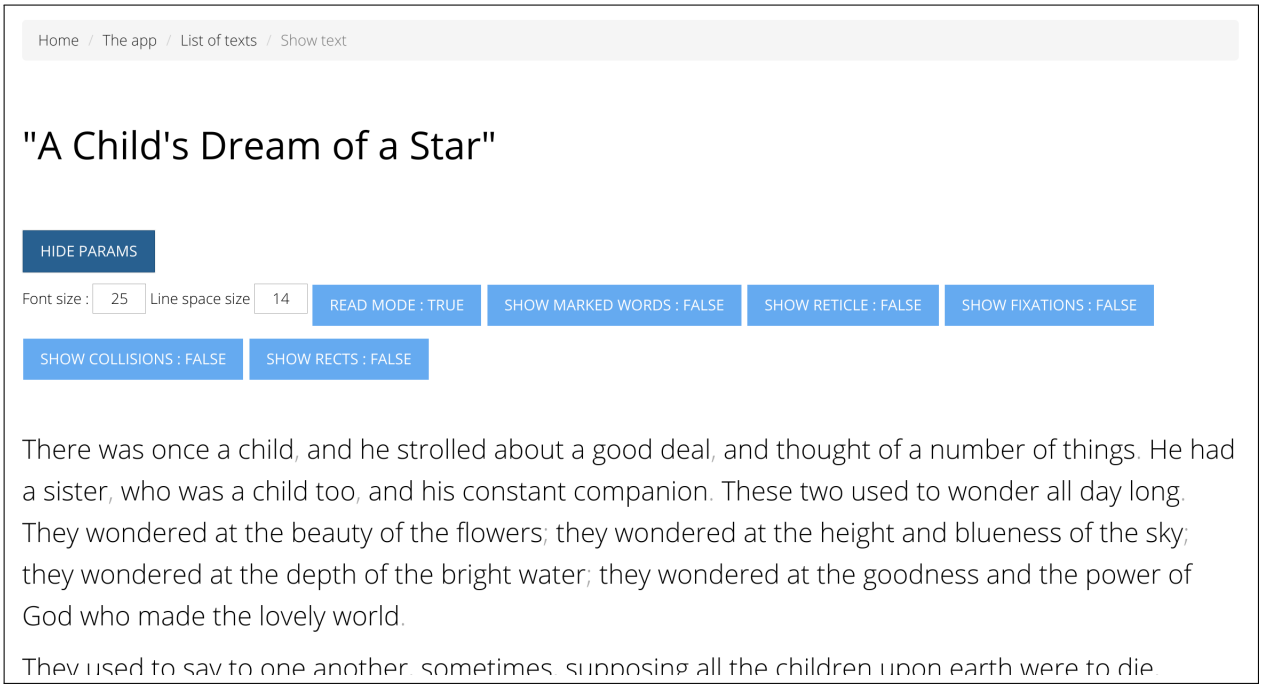


FIGURE 5.7 – Interface de modification des paramètres

Le paramètre **READ MODE** active ou désactive la détection des mots. **SHOW MARKED WORDS** affiche les mots marqués comme étant lus. **SHOW RETICLE** affiche le réticule (l'endroit ou l'utilisateur regarde actuellement). **SHOW FIXATIONS** affiche les fixations, c'est à dire les endroits où le regard de l'utilisateur s'est arrêté (cf partie Détection des mots lus par l'utilisateur pour plus de détails). **SHOW COLLISIONS** affiche les collisions entre le réticule et les boites englobantes des mots. **SHOW RECTS** affiche les boites englobantes des mots.

5.2 Implémentation côté client

5.2.1 Implémentation du MVC avec Angular

Nous avons choisi d'implémenter notre MVC en utilisant le framework AngularJS. Celui-ci nous permet de créer facilement un contrôleur et de lier des variables du modèle à notre vue en les utilisant directement dans le code HTML. De plus, il nous permet de créer des balises HTML personnalisées, ou *directives*, qui vont être modifiées en temps réel pendant l'exécution à chaque changement du modèle.

AngularJS nous offre également la possibilité d'implémenter plusieurs types de services, dont les *factory* qui permettent d'exécuter un code une seule fois et de retourner des valeurs ou des fonctions. Ce qui est très pratique pour définir des classes ou des modules par exemple.

Nous avons d'ailleurs utilisé des *factory* pour implémenter les classes *TextModel*, *ParagraphModel* et *WordModel* (cf. figure 4.4).

La partie vue (cf. figure 4.5) a été implémentée grâce à des directives *textView*, *paragraphView* et *wordView*.

Le contrôleur, quant à lui, est tout simplement une entité *controlleur* d'AngularJS. Il s'agit d'une fonction où l'on s'occupe d'initialiser le modèle et le module *Eye Tracker Interface* (cf. figure 4.6).

Notre modèle est donc sous la forme d'un arbre : le *textModel* n'est instancié qu'une fois à la création du modèle et contient une liste de *paragraphModel*, et chaque *paragraphModel* contient une liste de *wordModel* (cf figure 5.8).

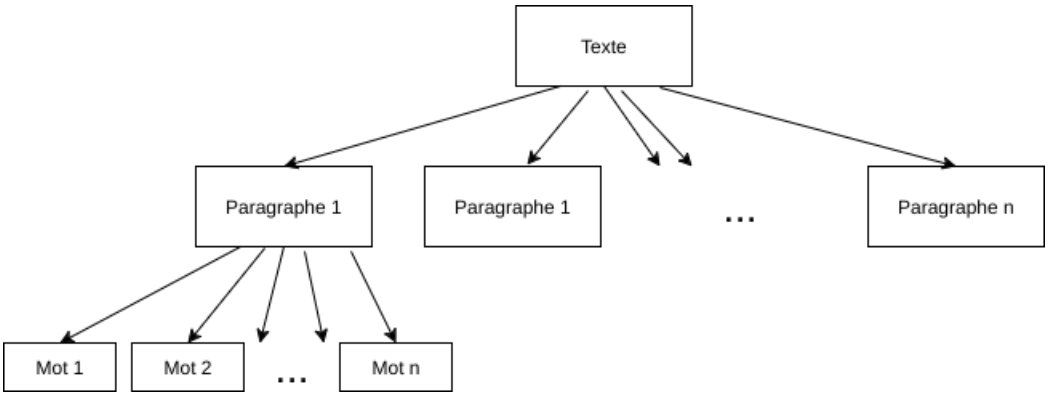


FIGURE 5.8 – Structure arborescente d'un texte

Cette structure arborescente est intéressante en terme de temps de calcul de collisions, en effet nous n'avons pas à tester à chaque moment les collisions avec tous les mots du texte si nous savons que les coordonnées se trouvent à l'intérieur d'un paragraphe, et même à l'intérieur du texte.
Ainsi, on regarde dans un premier temps si les coordonnées sont dans la boite englobante du texte, si c'est le cas, on regarde dans quel paragraphe du texte elles se trouvent, et enfin, on fait les tests de collision avec tous les mots du paragraphe.
Du point de vue de la complexité de l'algorithme, effectuer les collisions avec tous les mots du texte est en $O(n)$, n étant le nombre de mots dans le texte. Notre solution se rapproche d'une complexité en $O(\log n)$ dans les cas où le texte comporte plusieurs paragraphes, et que les paragraphes contiennent approximativement le même nombre de mots. Le cas limite étant un texte contenant un seul paragraphe : on effectue alors deux tests de collision de plus que sans la structure arborescente (test de collision avec le texte et le paragraphe).

Lorsque l'on initialise l'application, on découpe le texte à chaque retour à la ligne pour le découper en paragraphes puis on découpe chaque paragraphe en début et fin de mot. En faisant ainsi, nous obtenons deux chaînes de caractères pour chaque mot, la première contient le mot et la deuxième contient tout ce qu'il y a entre ce mot et le suivant.

Par exemple, en découpant la chaîne "Ceci est un exemple !" on obtient les chaînes suivantes : "Ceci", " ", "est", " ", "un", " ", "exemple", " !". Dans chaque instance de la classe WordModel nous stockons ses chaînes deux par deux, le premier WordModel contiendra "Ceci" et " ", le deuxième contiendra "est" et " " et ainsi de suite.

Afin de pouvoir connaître la position des mots affichés à l'écran, nous sommes resté sur l'idée que nous avions eu pour notre prototype : isoler chaque mot dans une balise span qui lui est propre. Ainsi, on peut récupérer en javascript chaque span et utiliser la méthode *getBoundingCllientRect()* pour récupérer sa position et sa taille à l'écran.

Nous avons utilisé les directives de la vue pour créer une arborescence similaire à celle du modèle du texte. Le code HTML de la page contient une instance de la directive *textView* qui est reliée à l'instance de *textModel*. Cette directive va créer une instance

de la directive *paragraphView* pour chaque *paragraphModel* présent dans l'instance de *textModel*. De manière récursive, chaque instance de la directive *paragraphView* va créer une balise * * qui contient une instance de la directive *wordView* pour chaque instance de *wordModel* contenu dans l'instance de *paragraphModel*. Enfin, chaque instance de la directive *wordView* va créer deux balises **, la première pour afficher la première chaîne stockée dans le *wordModel*, qui contient le mot, et la deuxième pour afficher la deuxième chaîne qui contient tous les caractères entre ce mot et le prochain.

Les directives d'AngularJS nous donnent la possibilité d'implémenter une fonction *link* qui est appelée après que le contenu de celle-ci ait été ajouté au document HTML. On utilise cette fonction pour récupérer une instance des objets HTML de la balise *textView*, de chaque balise * * des *paragraphView* et de chaque première balise ** des *wordView* que l'on stocke respectivement dans le *textModel*, *paragraphModel* ou *wordModel* correspondant.

On peut ainsi appeler la méthode *getBoundingClientRect()* de chaque élément du texte directement depuis notre modèle.

5.2.2 Détection des mots lus par l'utilisateur

Afin de détecter les mots lus par l'utilisateur il a fallu que nous nous occupions de plusieurs points :

- afficher le texte sur l'écran de l'utilisateur et connaître la position de chaque mot
- filtrer le flux de données de l'eye tracker afin d'obtenir des informations plus précises et adaptées à la détection de la lecture
- définir un algorithme qui à partir de la position des mots et des données de l'eye tracker filtrées nous donne les mots lus par l'utilisateur en temps réel

Nous avons vu dans la partie précédente comment nous affichons le texte tout en l'ayant découpé pour faire en sorte de pouvoir connaître la position de chaque mot individuellement. Il ne reste donc plus qu'à voir comment fonctionnent le filtrage et l'algorithme de détection.

Filtrage des données de l'Eye Tracker

Les coordonnées que nous envoie le SDK de l'Eye Tracker sont très bruitées, en l'état elles sont inexploitables. Il faut donc appliquer une couche de filtrage, pour réduire la dispersion des données.

A partir de l'eye tracker, nous obtenons à chaque itération deux couples de coordonnées : les coordonnées en X et Y de l'oeil gauche et les coordonnées en X et en Y de l'oeil droit. Nous procédons d'abord à un filtrage des données nulles. En effet, GazeJs nous envoie un couple de zéros si l'oeil correspondant n'est pas détecté, ce qui arrive souvent lorsque l'utilisateur n'est pas bien positionné, si quelque chose passe entre l'eye tracker et l'utilisateur ou si celui-ci ferme les yeux. Si les coordonnées d'un des yeux sont nulles, alors on utilise les coordonnées de l'autre oeil. Sinon, on fait la moyenne des coordonnées des deux yeux en X et en Y pour obtenir un seul couple de valeurs.

Une fois que l'on a un couple de valeurs non nulles, on ajoute ce couple à un tableau en écrasant la valeur la plus ancienne. On a ainsi un tableau contenant les N dernières coordonnées dont on fait la moyenne pour débruiter le signal et obtenir des coordonnées stables.

Après la première release, nous nous sommes rendu compte que nous n'avions pas une précision suffisante pour réaliser une détection au mot près. Nous avons donc cherché à mieux filtrer nos données afin d'avoir une représentation de la dispersion des données autour de la moyenne que nous calculons déjà.

Dans un premier temps, nous avons essayé de calculer la moyenne des N dernières coordonnées de l'eye tracker et leurs écarts types en distinguant les points à gauche, à droite , au dessus et au dessous de la moyenne. Le résultat que nous avons obtenu était satisfaisant visuellement mais ce n'était pas une bonne solution. En effet, nous obtenions une forme composée de quatres quartiers d'ellipses différentes. Nous aurions donc eu un algorithme de détection des collisions plus complexe et éventuellement trop coûteux pour avoir un rendu en temps réel. Voir figure 5.9 A

Nous sommes donc passés sur une forme d'ellipse simple, alignée sur les axes X et Y. Pour cela il suffisait de calculer les moyennes des N coordonnées et leurs écarts types sur les deux axes. Seulement, l'ellipse étant alignée sur les axes, elle devenait anormalement grande lorsque l'on avait un mouvement en diagonale. Voir figure 5.9 B

Nous avons donc cherché à obtenir une ellipse qui n'est pas alignée sur les axes. Pour cela, nous avons dû réaliser une analyse en composantes principales pour obtenir les vecteurs propres et les valeurs propres de notre nuage de points. On utilise ensuite ces vecteurs et la moyenne des coordonnées pour obtenir une ellipse. Le résultat obtenu était très satisfaisant, l'ellipse suivait parfaitement le mouvement des yeux. Voir figure 5.9

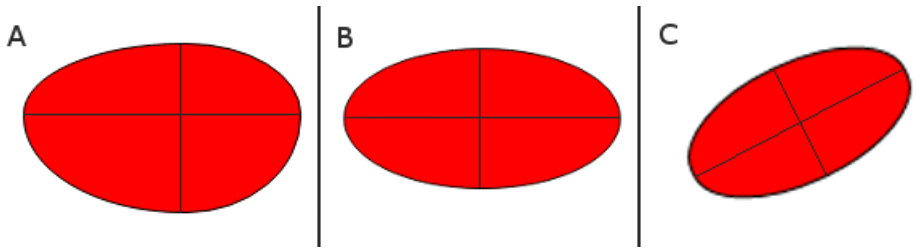


FIGURE 5.9 – Différents types d'ellipses :

— A forme composée de quatre quartiers d'ellipses différentes
— B ellipse alignée sur les axes
— C ellipse correspondant le mieux à notre nuage de point (ACP)

Cependant, le calcul de l'ACP était exécuté à chaque itération de notre boucle d'affichage. Afin de palier à ce problème nous avons revu notre manière de filtrer et traiter les coordonnées de l'eye tracker.

En effet, nous avons décidé d'implémenter un algorithme de détection des fixations et des saccades basé sur la dispersion. Lorsqu'un utilisateur lit un texte, on peut détecter des fixations qui apparaissent lorsque l'utilisateur est en train de lire un mot ou une partie d'un mot et que ses yeux fixent un point. Les saccades correspondent aux mouvements très rapides des yeux entre deux fixations. Nous nous interessons principalement aux fixations pour détecter les mots lus.

Pour détecter une fixation il y a deux contraintes :
— les critères de dispersion de notre nuage de points, en l'occurrence ici les écarts types, ne doivent pas dépasser un certain seuil,
— le temps minimum de durée de la fixation, en général aux alentours de 100 à 150 ms.

Puisque nous nous intéressons uniquement aux fixations, dont la dispersion est faible, nous n'avons plus de réel besoin d'avoir une ellipse non alignée sur les axes et donc nous calculons les écarts types sur les axes X et Y.

Lorsque l'on obtient une nouvelle coordonnée de l'eye tracker, on remplace dans notre tableau la plus ancienne par celle-ci et on recalcule les moyennes et les écarts types. Si la somme des écarts types des deux axes dépasse notre seuil de dispersion, alors la fixation en cours est terminée. On vide notre tableau de coordonnées et on regarde si la durée de cette fixation est plus grande que la durée minimum. Si c'est le cas, on a bien détecté un fixation. Vous pouvez voir un exemple des fixations que l'on obtient sur la figure 5.10 en rouge.

Algorithme de détection des mots

Afin de détecter les mots lus, il faut d'abord pouvoir détecter les mots qui se trouvent à la position obtenue après avoir filtré les coordonnées de l'Eye Tracker. Nous avons donc créé une fonction *isInRect(rect, x, y)* qui retourne *true* si le point *(x, y)* se trouve à l'intérieur du rectangle *rect* pour pouvoir détecter les mots lu avec notre première version du filtrage où nous utilisions uniquement la moyenne des N dernières coordonnées de l'Eye Tracker. Pour détecter les mots lus avec le filtrage basé sur la dispersion, nous avons utilisé une bibliothèque de géométrie 2D, développée par Kevin Lindsay [1] pour pouvoir détecter les collisions entre une ellipse, pour les fixations, et un rectangle, pour les élements du texte. Nous utilisons ces fonction dans les méthodes *isInRect()* et *ellipseIntersect()* des classes *textModel*, *paragrapheModel* et *wordModel* pour détecter les collisions entre leur élément HTML respectif et le point ou l'ellipse passée en paramètre. Nous avons également ajouté les méthodes *getParagraphsAt(x, y)* et *getParagraphsAtEllipse(ellipse)* dans *TextModel* ainsi que les méthodes *getWordsAt(x, y)* et *getWordsAtEllipse(ellipse)* dans *ParagraphsModel* afin de détecter les collisions plus facilement dans notre arborescence.

Pour la première release, notre algorithme de détection des mots lus utilisait la moyenne des coordonnées de l'Eye Tracker. À chaque itération de notre boucle d'affichage, nous cherchons si le point retourné par le filtrage des données se trouve sur un mot.

1. `http://www.kevlindev.com/geometry/2D/intersections/index.htm`

Si on trouve un mot, nous avions trois cas de figure :

— si nous n'avions aucun mot à la précédente itération, alors cela veut dire que l'on vient d'entrer dans ce mot. On le garde en mémoire et on enregistre le temps actuel.
— si nous avions ce même mot à l'itération précédente, alors on ne l'a pas quitté. On ne fait rien.
— si nous avions un mot différent à l'itération précédente, alors cela veut dire que l'on a quitté le mot précédent et que l'on est entré dans le nouveau mot. On garde donc en mémoire le nouveau mot et on enregistre le temps actuel comme dans le premier cas. On va également comparer le temps actuel et le temps que l'on a enregistré pour le mot précédent et calculer le temps pendant lequel on est resté dessus. Si on est resté plus de Xms (TODO expliquer le seuil) on marque le mot comme lu.

Si on ne trouve pas de mot et que l'on avait un mot à l'itération précédente, alors on va comparer le temps actuel et le temps enregistré pour celui-ci pour décider si on le marque comme lu de la même manière.

Cette technique marchait assez bien mais pas à tous les coups, nous n'étions pas assez précis pour détecter tous les mots lus. La position captée par l'eye tracker est souvent légèrement décalée, généralement juste en dessous de la ligne lue et les mots très petits étaient plus durs à détecter que les mots longs puisqu'ils sont moins larges.

Nous avons donc rajouté une détection au niveau des paragraphes. De la même manière que pour les mots, on regarde à chaque itération dans quel paragraphe on se trouve et on regarde le pourcentage de mots lus lorsque l'on en sort. Si le pourcentage de mots lus est au dessus d'un certains seuil, on considère que tous les mots du paragraphe sont lus.

On règle ainsi le problème des mots qui ne sont pas détectés. Cependant, cette solution n'était pas très satisfaisante. Il arrivait par moment que la position captée par l'eye tracker se trouve pile entre deux lignes rendant la détection impossible.

Pour la release finale, nous avons repensé notre manière de détecter les mots. Tout d'abord, nous utilisons le filtrage avec la détection des fixations. L'algorithme de filtrage appelle une fonction du contrôleur à chaque fois qu'il détecte une fixation. Ainsi, la détection des mots lus ne se fait plus à chaque itération de la boucle d'affichage mais uns seule fois pour chaque fixation trouvée.

Il ne reste plus qu'à détecter les collisions entre les mots du texte et l'ellipse correspondant à cette fixation. Tous les mots qui sont en collision avec l'ellipse sont marqués comme lus puis on fait une détection par paragraphe comme pour la première release en regardant le pourcentage de mots lus.

Nous avons remarqué cependant que les dispersions des fixations n'étaient pas très grandes et qu'il arrivait encore par moment que les ellipse des fixations se trouvent entre deux lignes. Nous avons alors décidé d'agrandir en hauteur l'ellipse de nos fixations avant de faire la détection des collisions avec les mots afin d'être sûr de détecter les mots lors de la lecture, quitte à rajouter une marge d'erreur. Voir les ellipses en cyan sur la figure 5.10.

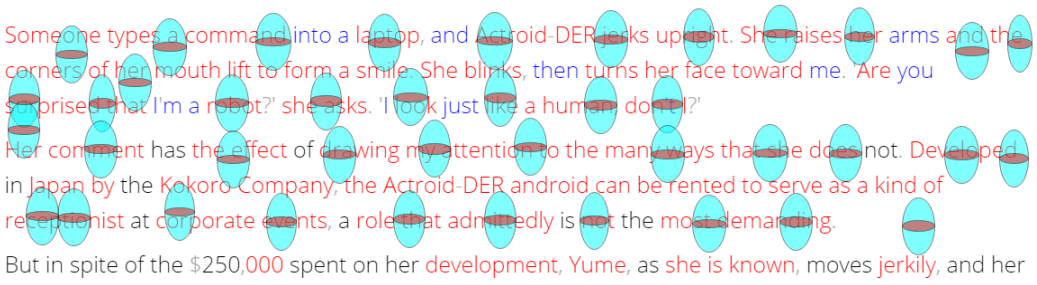


FIGURE 5.10 – Résultat obtenu en affichant les fixations (en rouge) et les ellipses utilisées pour la détection des mots (en cyan).

5.2.3 Affichage des statistiques utilisateur

Sur sa page, l'utilisateur peut consulter des statistiques concernant sa progression. Les statistiques que nous avons choisi d'afficher sont l'évolution du nombre de mots lus au cours du temps, le nombre de mots lus par session, et le pourcentage de mots de la langue qu'il a déjà vu.

Pour créer nos graphiques, nous avons utilisé la bibliothèque Plotly [2] dans sa version javascript. Cette bibliothèque est extrêmement simple à mettre en place et à utiliser, et propose une grande variété de types de graphiques.

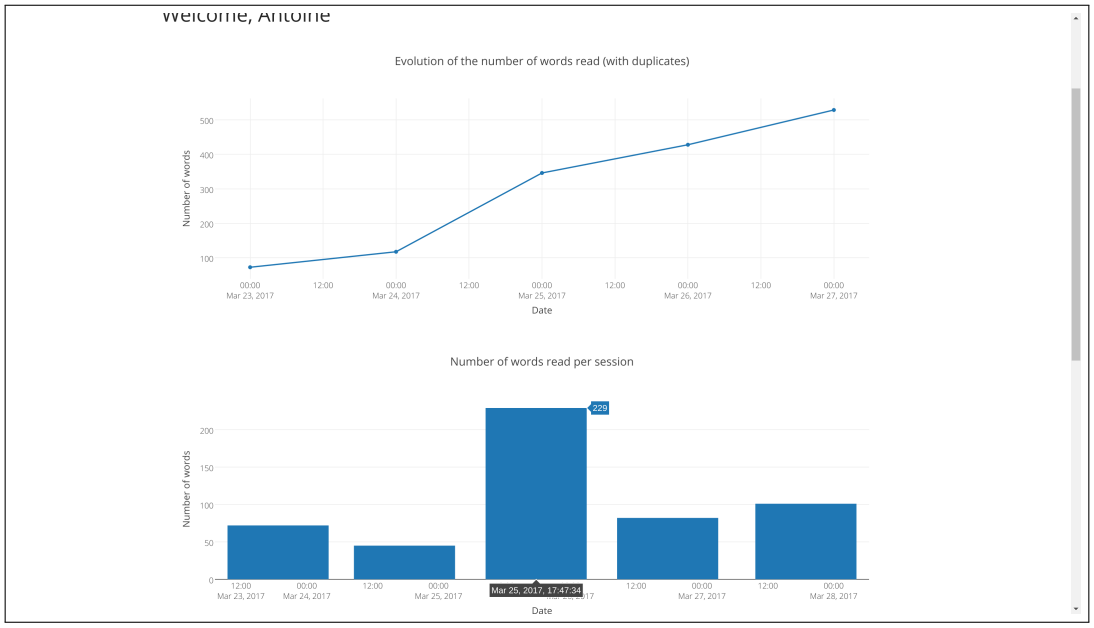


FIGURE 5.11 – Affichage des statistiques de l'utilisateur (1)

2. `http://plot.ly/javascript/`

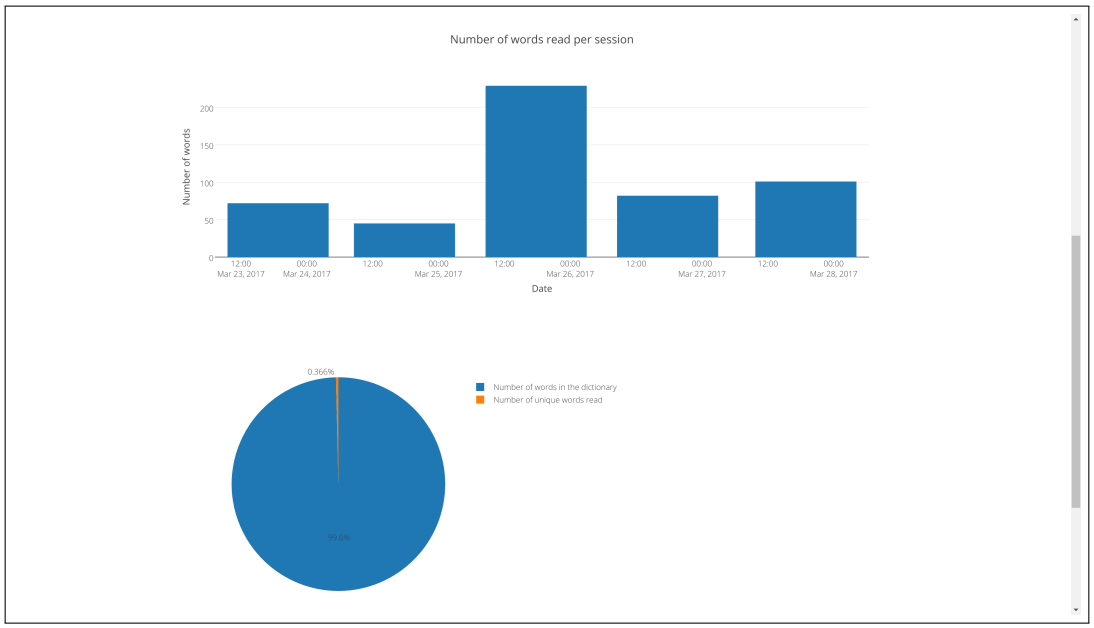


FIGURE 5.12 – Affichage des statistiques de l'utilisateur (2)

5.2.4 Implémentation de l'interface pour l'Eye Tracker

Afin de faire communiquer notre page web avec le SDK de l'Eye Tracker, nous avons choisi d'utiliser un serveur local NodeJS, qui fait tourner la bibliothèque GazeJS [3]. Cette bibliothèque permet d'avoir accès à certaines fonctions du SDK Tobii EyeX en Javascript. Ainsi, notre serveur local récupère les coordonnées du regard de l'utilisateur, et les transmets à la page web via un websocket Socket.IO [4].

5.3 Implémentation côté serveur

5.3.1 Suggestion de textes

Racinisation

Pour pouvoir savoir efficacement les différents mots lus par l'utilisateur et les stocker, il est important d'être capable d'extraire la racine de chaque mot. En effet, nous voulons éviter de considérer deux "formes" différentes du même mot (conjugaison différente, pluriel...) comme étant deux mots différents. Ainsi, à titre d'exemple, nous voulons considérer les mots "released" et "releasing" comme un même mot, de racine "release".

3. `http://github.com/jiahansu/GazeJS`
4. `http://socket.io`

Il existe plusieurs méthodes permettant d'extraire la racine d'un mot, notamment des algorithme de "lemmatisation" et de "*Stemming*". Un algorithme de "*Stemming*" va simplement chercher des chaînes de caractères connues comme étant des suffixes en fin de mot, sans se préoccuper du mot, ou du contexte, alors qu'un algorithme de "lemmatisation" s'appuie sur du vocabulaire, une analyse morphologique des mots et de leur contexte, pour identifier les cas particulier et extraire ainsi de manière plus fiable les racines des mots.

Ainsi, un algorithme de "lemmatisation" pourra identifier que les mots *am*, *are* et *is* possèdent tous la même racine : *be*, ce qui n'est pas possible avec un algorithme de "*Stemming*".

On voit facilement que le procédé de "lemmatisation" est bien meilleur que le "*Stemming*", mais il est aussi beaucoup plus lourd, et la précision supplémentaire offerte n'est finalement pas très utile dans notre cas, car nous ne cherchons pas à identifier les mots afin, par exemple, d'en déduire une définition, mais seulement à rassembler au maximum les mots proches afin de ne pas les compter plusieurs fois.

Un des algorithmes les plus populaires pour faire du "*Stemming*" est l'algorithme de Porter[10], nous avons pu en trouver facilement une implémentation libre en php [5] que nous avons importé dans notre projet (dans le dossier *libs*).

D'un point de vue plus pratique, nous avons passé cet algorithme sur notre base de mots anglais, et enlevé les doublons afin d'obtenir une base de mots "racinisés" (réduisant au passage de moitié la taille de la base). De même, nous avons passé l'algorithme sur tous les mots de chaque texte, créant à chaque nouveau mot (racinisé et existant dans la base de mots), une ligne dans la table d'association entre mots et texte, ce qui nous permet de connaître quels mots sont présents dans le texte (sans doublons).

Distance de Jaccard

Une partie importante de notre application est la suggestion automatique de textes à l'utilisateur. L'idée est de proposer à l'utilisateur une manière de toujours s'améliorer, en apprenant du nouveau vocabulaire.
Notre algorithme trie les textes par ordre croissant de leurs distance de Jaccard [6] avec l'ensemble des mots déjà lus par l'utilisateur. La distance de Jaccard permet de comparer la similarité entre des échantillons, elle s'exprime comme ceci :

$$\text{Distance(A, B)} = 1 - \frac{|A \cap B|}{|A \cup B|}$$

Avec A et B deux ensembles d'échantillons, dans notre cas, deux ensembles de mots.

Nous choisissons ensuite 3 textes, en sélectionnant aléatoirement le premier entre le premier tiers de l'ensemble de textes trié, le deuxième dans le deuxième tiers et le troisième

5. `http://tartarus.org/martin/PorterStemmer/php.txt`
6. `http://fr.wikipedia.org/wiki/Indice_et_distance_de_Jaccard`

dans le troisième tiers. Ainsi, le premier texte est plus susceptible de contenir un grand nombre de mots que l'utilisateur à déjà lu, et le troisième est plus susceptible d'en contenir peu (voire aucun, au début de l'utilisation de l'application).

5.3.2 Calcul des statistiques

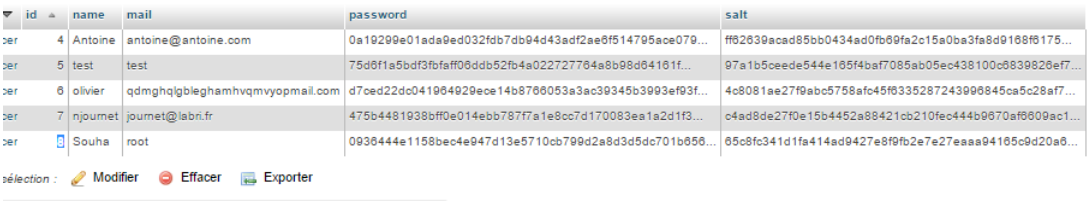
<à rédiger>

5.3.3 Sécurité

Chiffrement des mots de passes

Lors de l'inscription, il est demandé à l'utilisateur de renseigner un mot de passe. Il est important que ce mot de passe soit chiffré en base de données. Pour ce faire, nous utilisons la fonction de hachage **SHA256**, qui produit une chaîne de caractères hachée de 256bits à partir de la chaîne de caractères fournie. Nous effectuons aussi un **salage** dynamique, qui consiste à concaténer la chaîne de caractères à hacher avec une chaîne générée aléatoirement (puis stockée en base, associée à l'utilisateur). Cette technique sert à minimiser le risque de collisions, c'est à dire une situation dans laquelle deux entrées différentes d'une fonction de hachage produisent la même sortie.

Lors de l'authentification, l'opération inverse est effectué pour vérifier que les informations fournies par la personne qui tente de s'authentifier correspondent bien à un compte enregistré dans la base de données.



id	name	mail	password	salt
4	Antoine	antoine@antoine.com	0a19299e01ada9ed032fdb7db94d43adf2ae6f514795ace079...	ff62639acad85bb0434ad0fb69fa2c15a0ba3fa8d9168f6175...
5	test	test	75d6f1a5bdf3fbfaff06ddb52fb4a022727764a8b98d64161f...	97a1b5ceede544e165f4baf7085ab05ec438100c6839826ef7...
6	olivier	qdmghqlgbleghamhvqmvyopmail.com	d7ced22dc041964929ece14b8766053a3ac39345b3993ef93f...	4c8081ae27f9abc5758afc45f6335287243996845ca5c28af7...
7	njournet	journet@labri.fr	475b4481938bff0e014ebb787f7a1e8cc7d170083ea1a2d1f3...	c4ad8de27f0e15b4452a88421cb210fec444b9670af6609ac1...
8	Souha	root	0936444e1158bec4e947d13e5710cb799d2a8d3d5dc701b656...	65c8fc341d1fa414ad9427e8f9fb2e7e27eaaa94165c9d20a6...

Modifier Effacer Exporter

FIGURE 5.13 – *hash* et *salt* dans la base de données

Sécuriser la connexion avec le serveur

En plus de cette méthode, nous avons aussi utilisé les certificats SSL Let's Encript [7] afin que le site utilise HTTPS. Ce protocole chiffre les communications entre le client et le serveur, limitant, entre autres, les tentative de **network spoofing**, puisque le mot de passe est chiffré lorsqu'il est envoyé au serveur lors de la création de compte et de l'authentification.

7. `http://letsencrypt.org/`

Chapitre 6

Critiques, évolutions et perspectives

6.1 Facilité d'installation

Lors du rendu de la première release, notre module de communication avec l'Eye Tracker était le serveur NodeJS local décrit dans la section prototype. Nous avons voulu *packager* ce module (le transformer en exécutable), car en l'état, l'utilisateur devait installer NodeJS, un certain nombre de modules NodeJS, et nous n'avons pas réussi à tous les coups à le faire fonctionner correctement (4 réussites sur 5 essais, avec 5 ordinateurs différents). Ce procédé est laborieux, nous voudrions que l'utilisateur n'ait qu'a exécuter un programme, et rien d'autre.
Après de nombreux et divers essais, nous n'avons pas réussi à créer un exécutable à partir du module tel quel, et nous avons choisi de ne pas insister sur ce point pour nous concentrer sur les autres parties du projet.
Une solution serait de refaire cette partie du projet en C# .NET, car le SDK Tobii et socket.io existent pour ces langages. Il serait de plus beaucoup plus aisé de créer un exécutable de cette façon.

Une piste d'enrichissement aurait été d'implémenter la possibilité d'enregistrer les coordonnées d'une session, afin de pouvoir la répéter et ainsi avoir un outil plus fiable pour tester et classer nos différentes approches de détection de mots.

6.2 Pertinence de l'algorithme de suggestion de textes

L'algorithme tel quel est assez naïf et ne mesure l'évolution de l'utilisateur que sur la base des mots qu'il a déjà lu. On pourrait raffiner cette mesure en définissant l'importance de chaque mots avec un algorithme de pondération de type **TF-IDF** [1], qui donne des importances différentes aux mots en fonction du nombre de fois qu'ils apparaissent dans le corpus de textes. On pourrait aussi enregistrer à quelle date chaque mot a été lu pour la dernière fois, cette donnée nous permet d'imaginer un algorithme qui espace les révisions de

1. `http://fr.wikipedia.org/wiki/TF-IDF`

mots de manière à optimiser l'apprentissage, en s'inspirant des techniques d'apprentissage par répétitions espacées [2]

6.3 Statistiques plus avancées

Il serait intéressant pour l'utilisateur de connaître sa vitesse moyenne de lecture. On peut générer cette statistique en enregistrant le *timestamp* du début de session (au moment ou le premier mot est lu par exemple), et de la fin de session (quand l'utilisateur a cliqué sur *Done Reading*). On stocke alors ces données dans la table Sessions de la base de données, et on peut afficher la vitesse de lecture pour chaque session, et ainsi visualiser son évolution. On pourrait même se servir de ces données pour mieux estimer le niveau de l'utilisateur, afin de proposer une meilleure suggestion de texte.

6.4 Meilleure expérience utilisateur

Une autre idée d'amélioration est d'intégrer un défilement automatique de la page lorsque l'utilisateur lis le bas de la page. Cette fonctionnalité semble relativement simple à implémenter (détecter quand les coordonnées de regard en y sont au dessus d'un seuil, et faire défiler la page) mais il est important que cette fonctionnalité ne soit pas dérangeante pour l'utilisateur, il faut que le défilement soit fluide, et arrive au bon moment.

Nous avons aussi eu l'idée de rajouter une couche de calibration de l'Eye Tracker à la volée dans l'application elle même. Au début de chaque session, on demande à l'utilisateur de fixer un point au milieu de l'écran et d'appuyer sur *Entrée*. On peut ainsi quantifier le décalage effectif en x et en y et le corriger au moment de la détection des mots.

2. `http://en.wikipedia.org/wiki/Spaced_repetition`

Chapitre 7

Tests

Afin de valider le bon fonctionnement de l'application, ainsi que de s'assurer que nous avons bien respecté les besoins, nous avons mis en place différents types de tests.

Les tests d'ergonomie cherchent à valider le besoin non fonctionnel *Interface Intuitive* en faisant essayer l'application à des testeurs pour mesurer le nombre de clics et le temps qu'ils mettent à trouver différentes fonctionnalités du logiciel.
Les tests de fonctionnement visent à vérifier que le logiciel fonctionne comme prévu, et atteint les objectifs de performance fixés dans le cahier des besoins.
Les tests unitaires servent à vérifier que le code écrit se comporte comme attendu, et à mettre en évidence les cas particuliers afin de les prendre en compte.

7.1 Tests d'ergonomie

Pour tester l'ergonomie et l'intuitivité de notre application, nous avons mis en place différents tests, que nous avons soumis à un panel de 8 personnes.

7.1.1 Protocole

Contexte : Le sujet est assis devant un pc portable, l'Eye Tracker est branché, et le sujet a déjà effectué la procédure de calibration. L'application est ouverte sur la page d'accueil.
Mesures : Pour chaque test, nous mesurons le temps que mets le sujet à effectuer la tache, et le nombre de clics.
Déroulement : Au moment ou la tache est annoncée, nous lançons le chronomètre et nous commençons à compter les clics.

7.1.2 Création de compte

Nous regardons ici dans quelle mesure notre interface est intuitive dans le cas où l'utilisateur veut créer un compte.

Testeur	Temps	Clics	Notes
1	1m07	3	Passé par *Sign In* avant d'aller sur *Sign up*
2	1m00	5	A essayé de s'inscrire sur la page Sign In
3	30s	2	
4	43s	3	A cliqué sur le carousel
5	1m24	5	A cliqué sur *The app* (redirige sur *Sign In*), a essayé de s'inscrire sur cette page, puis a cliqué sur *Sign Up*
6	37s	2	
7	29s	2	
8	40s	3	Passé par Sign In

On peut voir qu'il existe une confusion entre ***Sign In*** et ***Sign Up***, respectivement **se connecter** et **s'inscrire**. Peut-être devrions nous proposer une version française du site, afin que les débutants en Anglais puissent naviguer plus facilement.

7.1.3 Connexion au compte

Testeur	Temps	Clics	Notes
1	22s	2	
2	15s	2	
3	41s	3	
4	14s	3	
5	50s	4	S'est trompé de mot de passe
6	12s	2	
7	16s	2	
8	20s	2	

On peut se poser la question de la pertinence de ce test, étant donné que les sujets venaient de créer leur compte au moment du test, et certains d'entre eux sont passé par la page de connexion pour s'inscrire. Un test plus représentatif serait de faire ces deux tests sur des personnes différentes.

7.1.4 Démarrer une session de lecture

Testeur	Temps	Clics	Notes
1	37s	3	A choisis une suggestion
2	50s	3	Sélection manuelle
3	30s	3	Suggestion
4	27s	2	Suggestion
5	46s	3	Manuelle
6	20s	2	Suggestion
7	23s	2	Suggestion
8	35s	3	Suggestion

7.2 Tests de fonctionnement

7.2.1 Détection des paragraphes lus

Contexte : Le sujet est assis devant un pc portable, l'Eye Tracker est branché, et le sujet a déjà effectué la procédure de calibration. L'application est ouverte sur la page d'accueil.
Mesures : On regarde ici le nombre de paragraphes détectés comme lus lors d'une session de lecture normale par rapport au nombre effectif de paragraphes lus par le sujet
Déroulement : Nous présentons l'interface de lecture avec un texte que nous avons choisi à l'utilisateur, on lui demande de lire le texte en entier. Les paragraphes lus s'affichent en bleu

Texte : The first Americans

Difficulté du texte : Facile
Nombre de paragraphes : 6

Testeur	Nombre de paragraphes détectés
1	1
2	2
3	1
4	2
5	3
6	2
7	2
8	1

A l'issue de cette première session de test, nous nous sommes rendu compte que notre seuil de détection de paragraphes était trop haut, nous l'avions modifié pour un test et oublié de le remettre à sa valeur originale. Nous l'avons donc ajusté et continué la session.

Texte : Nanotechnology : an introduction

Difficulté du texte : Moyen
Nombre de paragraphes : 6

Testeur	Nombre de paragraphes détectés
1	6
2	5
3	6
4	6
5	5
6	6
7	6
8	6

Texte : A Hacker's life

Difficulté du texte : Difficile
Nombre de paragraphes : 8

Testeur	Nombre de paragraphes détectés
1	8
2	8
3	8
4	7
5	7
6	8
7	6
8	8

Texte : Sea gypsies of Myanmar

Difficulté du texte : Difficile
Nombre de paragraphes : 4

Testeur	Nombre de paragraphes détectés
1	3
2	4
3	4
4	4
5	4
6	4
7	3
8	4

Chapitre 8

Conclusion

L'utilisation de l'Eye Tracking dans les différentes domaines présente de nos jours un atout considérable afin de comprendre le comportent de l'internaute sur un site, le suivre et l'orienter. La mission qui nous a été confié est de concevoir et réaliser une application d'apprentissage d'une langue à l'aide Eye Tracker.

TODO revoir la conclusion, Nous avons, à cet effet, essayé d'adopter les meilleures solutions techniques et méthodes de développement. Dans une première étape, nous avons commencé par introduire le cadre de l'élaboration du projet ainsi qu'une étude théorique sur les notions de base et les domaines d'utilisation, suivie d'une étude de l'existant. Puis, nous avons introduit la phase de conception avec les différents diagrammes, et on a mis en place l'architecture de base de notre projet, l'implémentation suivi des tests.

Nous avons préparé par la suite notre planning de travail en respectant les priorités de nos besoins suite à une discussion entre l'équipe du développement et le client du produit. La richesse de ce sujet c'était une occasion pour profiter tant dans l'acquisition des connaissances que dans l'initiation au travail de groupe. Cependant ce projet réalisé dans le cadre du projet de programmation, était aussi une opportunité de découvrir le travail en équipe du point de vue partage, organisation, adaptation et difficultés.

Comme d'autres applications Web, notre application peut être aisément améliorée. En effet, grâce à son aspect ouvert, le web offre l'opportunité de créer d'autres applications innovant révolutionnaires en encourageant les développeurs à avancer dans leur imagination et à mobiliser toutes leurs compétences pour le meilleur de cette application. Comme travail de futur, nous suggérons de faire compléter cette application, c'est-à-dire, trouver un SDK compatible avec divers systèmes d'exploitation, améliorer les interfaces web et régler le bruit lié au type du matériel.

Chapitre 9

Annexes

9.1 Questionnaires

9.1.1 Déterminer les niveaux des textes

- Numéro (identifiant) du texte
- Nombre de textes lus précédemment
- Quel niveau de langue estimez vous avoir ? :
  - Débutant : A commencé très récemment son apprentissage de la langue
  - Intermédiaire : Est capable de comprendre un texte simple
  - Avancé : Est capable de comprendre un texte avec des tournures de phrases et du vocabulaire complexe (expressions idiomatiques, vocabulaire technique etc)
  - Bilingue : Est complètement à l'aise avec la langue et ses subtilités
- Comment avez vous trouvé le texte ? :
  - Très facile
  - Facile, mais quelques difficultés par endroits
  - Moyennement facile : Comprends le sens du texte, sans comprendre la totalité
  - Difficile : Difficulté à comprendre le sens
  - Impossible : N'a rien compris au texte
- des précisions ?

9.1.2 Évaluation de la difficulté d'installation

- Quel est votre niveau de connaissance en Informatique ?
- l'installation a-t-elle été réussie ? [Oui][Non]
- Combien de temps est-ce que ça vous a pris ?
- Avez vous rencontré des difficultés lors de l'installation ? Si oui, précisez lesquelles
- Estimez vous que vous avez été assez guidé lors de cette installation ?
- Avez vous des suggestions pour faciliter le processus d'installation de ce logiciel ?

9.1.3 Évaluation de l'intuitivité du logiciel

- Avez vous eu des difficultés à trouver certaines fonctionnalités du logiciel ? Si oui lesquelles (difficultés et fonctionnalités) ?
- Les noms et localisations des menus vous ont ils semblé logiques ?
- Avez vous des suggestions pour améliorer l'intuitivité de l'interface ?

Bibliographie

[1] Stemming and lemmatization. https ://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html. Accessed : 2017-03-28.

[2] D. A. L. Y. Chi Yui Leung, Masatoshi Sugiura. The perceptual span in second language reading : An eye-tracking study using a gaze-contingent moving window paradigm [open journal of modern linguistics]. pages P 585–594, 2014. http ://file.scirp.org/pdf/OJML2014111314212831.pdf.

[3] E. Cintrón-Valentín. Salience in second language acquisition : Physical form, learner attention, and instructional focus. *Front. Psychol*, Vol 7, 2016. https ://www.ncbi.nlm.nih.gov/pmc/articles/PMC5002427/.

[4] E. Dolgunsoz. Measuring attention in second language reading using eye-tracking : The case of the noticing hypothesis. *Journal of Eye Movement Research*, pages P 1–18, 2015. https ://bop.unibe.ch/index.php/JEMR/article/view/2413.

[5] T. Hume. Accurate eye center tracking in opencv, 2012. http ://thume.ca/projects/2012/11/04/simple-accurate-eye-center-tracking-in-opencv/, [visite] 04/03/2017.

[6] J.-S. Kang, A. Ojha, and M. Lee. Development of intelligent learning tool for improving foreign language skills based on eeg and eye tracker[proceedings of the 3rd international conference on human-agent interaction]. pages P 121–126, 2015. http ://doi.acm.org/10.1145/2814940.2814951r.

[7] C. T. Maja Pivec and J. Pripfl. Eye-tracking adaptable e-learning and content authoring support. *Informatica*, Vol 152 :P 83–86, 2006. http ://www.eurodl.org/materials/contrib/2005/Christian_Gutl.htm.

[8] N. Z. Martina Manhartsberger. Eye tracking in usability research : What users really see. *How Can Usability Engineering Reach These Goals ?*, Vol 198 :P 141–152, 2005. http ://www.usability.at/ueberuns/images/EyetrackinginUsability.pdf.

[9] Z. Z. Mingzhuo Liu. A case study of using eye tracking techniques to evaluate the usability of e-learning courses. *Int. J. Learn. Technol.*, 7(2) :154–171, July 2012. http ://dx.doi.org/10.1504/IJLT.2012.047980.

[10] M. Porter. An algorithm for suffix stripping. Vol 14, 1980. http ://www.emeraldinsight.com/doi/pdfplus/10.1108/eb046814.

[11] S.-C. Roberts. Using eye-tracking to investigate topics in l2 acquisition and l2 processing [studies in second language acquisition]. 2013. https ://doi.org/10.1017/S0272263112000861.

[12] J. D. VeláSquez. Combining eye-tracking technologies with web usage mining for identifying website keyobjects. *Eng. Appl. Artif. Intell.*, Vol 26(N 5-6) :P 1469–1478, May 2013. https ://www.dii.uchile.cl/wp-content/uploads/2015/06/Combining-eye-tracking-technologies-with-web.pdf.

[13] J. A. Vigo. Eye gaze tracking for tracking reading progress. Vol 100, June 3, 2013. http ://projekter.aau.dk/projekter/files/77098782/master.pdf.

[14] O. Špakov. Comparaison of eye movement filters used in hci. *Int. J. Learn. Technol.*, 7(2) :154–171, July 2012. https ://pdfs.semanticscholar.org/ff1e/ff1aaf27998e6019f0d7ae4d664efe3c344d.pdf.