[Date]

Zenidoc

[Nom de la société]

[Adresse de la société]

[Titre du document]

[Sous-titre du document]

Les informations contenues dans ce document pourront faire l’objet de modifications.

Aucune partie de ce manuel ne peut être reproduite ou transmise à quelque fin ou par quelque moyen que ce soit, électronique ou mécanique, sans la permission expresse de **Zenidoc** S.A.

*© 2022* ***Zenidoc*** *S.A. Tous droits réservés.*

*Zenidoc® est une marque déposée de Zenidoc S.A.*

Table des matières

I - Description du travail proposé1

A – Le besoin3

B – La solution proposée3

Tapez le titre du chapitre (niveau 3)3

II - Description du travail réalisé4

A – API Fetch 4

B – Description intégration des commands5

1. Fonctionnement et intégration des commands 6
2. La base de données7

C – Diagramme d’activité8

III - Bibliographie9

**I – Description du travail proposé**

**A – Le besoin**

La société Zenidoc édite un service de reconnaissance vocale. De ce fait, elle possède un dictionnaire de données permettant de reconnaitre l’ensemble des mots pouvant être retranscrits correctement. Cependant, du fait qu’elle cible une clientèle assez spécifique dans le domaine médical, tous les mots ne peuvent pas être tous immédiatement reconnus par le système car n’étant pas présents dans le dictionnaire de données. La reconnaissance de ces mots est ainsi faite manuellement par des auto-entrepreneuses via un fichier Google Sheet partagé en ligne par la personne encadrante du projet. Ce fichier partagé permet la catégorisation, le traitement, la correction de ces mots avant leur validation par la personne encadrante du projet, permettant ainsi l’ajout de ces mots dans le dictionnaire de données.

Cependant, l’usage de ce fichier partagé n’est pas la meilleure solution car :

* Il est actuellement hébergé par Google, qui est un service commercial
* Il ne permet pas la vérification directe de la validité des champs saisis
* Il ne permet pas de garantir l’intégrité des données saisies

Il faudrait ainsi pouvoir fournir une solution permettant la saisie, la récupération et la vérification de ces données sans passer par ce fichier partagé.

**B – La solution proposée**

Réalisation d’un formulaire permettant de récupérer et envoyer à partir et vers une base de données MySQL, les informations saisies par les auto-entrepreneuses. Il intègre un système d’authentification, empêchant l’accès à toutes les personnes qui n’y seraient pas autorisées et permettant également d’afficher une liste de mots attribuées à l’utilisateur connecté. Ce formulaire pourrait ainsi inclure de contraintes sur la saisie qui n’étaient pas possible jusqu’à maintenant et assurer la sécurité et l’intégrité des données qui transitent par le site.

**II – Description du travail réalisé**

**A – API Fetch**

L’**API Fetch** permet la récupération et l’envoi de données à travers un réseau grâce à une requête qui correspond à une ressource souhaitée.

Dans notre cas, elle contient divers paramètres d’url dont :

* Le **logon** qui sert d’identifiant, de token pour se connecter en général sur le serveur
* La **command** qui permet d’identifier l’action que l’on souhaite réaliser, par exemple récupérer la liste des mots. Elles correspondent à des requêtes SQL
* Les **paramètres P1/P2/P3** dont le contenu varie en fonction de la command saisie. Seul le contenu de P1 est variable et n’est pas à saisir.



*Format de la requête*

Il existe actuellement cinq command permettant de réaliser une action prédéfinie dont :

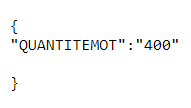
* Récupérer le nombre de mots alloués à un utilisateur : **customVERIFLISTE**
* Attribuer un nombre de mots et des mots si ce nombre est égal à 0 : **customAttributionListe**
* Afficher la liste des mots attribués à un utilisateur : **customListeMots**
* Récupérer un contexte par rapport à l’identifiant d’un mot : **customgetcontexte**
* Envoyer la saisie de l’utilisateur vers la base de données : **customvalidationcorrection**

**B – Intégration des commands et des données**

1. **Fonctionnement et intégration des commands**

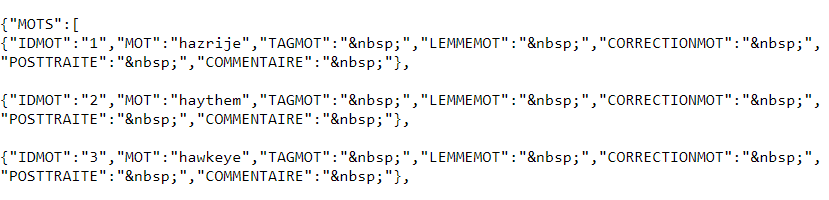
Les diverses commands énoncées dans la partie précédente sont exécutées de manière **asynchrone** les unes par rapport aux autres pour assurer une cohérence entre tous les points de vérification (comme vérifier qu’un utilisateur a des mots attribués ou pas). Les retours de ces fonctions étaient au **format JSON**, qu’il a fallu convertir pour qu’ils soient exploitables par le site. Si ces command renvoient une valeur à afficher sur le site, les fonctions JavaScript relatives à ces affichages sont appelées directement dans la méthode **fetch** JavaScript permettant la récupération des données globale. Ils ont pu être intégrées dans la réalisation du formulaire de la manière suivante :

* **customVERIFLISTE :** Suite à la connexion avec succès de l’utilisateur (en cas d’erreur ou de tentative d’accès direct à la page d’affichage des mots, on redirige l’utilisateur vers la page de connexion), l’utilisateur est automatiquement redirigé vers la page d’affichage des mots, avec en paramètre d’url l’identifiant qu’il a renseigné sur la page de connexion. Au chargement de la page, la fonction permettant l’envoi de la command **customVERIFLISTE**est lancé automatiquement et permet d’envoyer une requête pour récupérer le nombre de mots d’un utilisateur.



*Retour de la requête correspondant à l’usage de customVERIFLISTE*

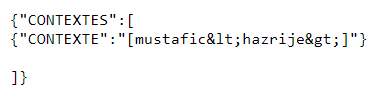
* **customAttributionListe :** Après l’exécution de la command précédente, la valeur de retour (contenant le nombre de mots alloués à l’utilisateur connecté) est directement intégrée en tant que paramètre de la fonction faisant appel à la command **customAttributionListe.** Si le nombre de mots est égal à 0, alors on envoie une requête permettant l’exécution de la command **customAttributionListe** qui permet l’attribution d’un nombre de mots à l’utilisateur. Dans les autres cas, il n’y a pas besoin d’exécuter cette command et on passe directement à la récupération de la liste des mots.
* **customListeMots :** Cette command permet la récupération de la liste des mots d’un utilisateur au format JSON. La fonction JavaScript qui lui est associée permet l’affichage de ces données dans un tableau possédant des champs de saisies.



*Extrait retour de la requête correspondant à l’usage de customListeMots*

* **customgetcontexte :** Elle permet de récupérer le contexte du mot de la ligne concernée, grâce à un bouton qui fait appel à une fonction JavaScript avec pour paramètre l’id du mot, ce qui permet de faire usage de la command. Également, le contexte récupéré est au format JSON. La fonction qui lui est associée permet l’affichage de ces données dans une fenêtre popup, qui possède une valeur par défaut si le mot sélectionné ne possède aucun contexte.

Elle est exécutée à la demande de l’utilisateur lorsqu’il appuie sur un bouton pour pouvoir économiser les ressources.



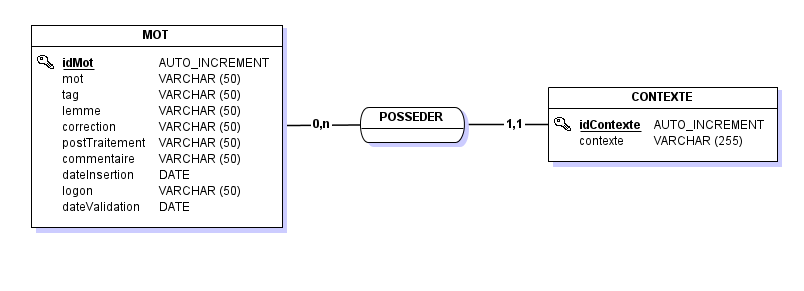
*Retour de la requête correspondant à l’usage de customgetcontexte*

* **customvalidationcorrection :** Cette command permet l’envoi des données vers la base de données MySQL. Elle se fait individuellement pour chaque mot et envoi en paramètre p3, l’ensemble des champs qui ont été saisis sous le format suivant :

tag + ";" + lemme + ";" + correction + ";" + commentaire + ";" + postTraitement

1. **La base de données**

La base de données a déjà été réalisé par un membre de l’équipe et permet de récupérer les données à afficher sur le site ou d’envoyer des données saisies lorsque l’utilisateur le demande.

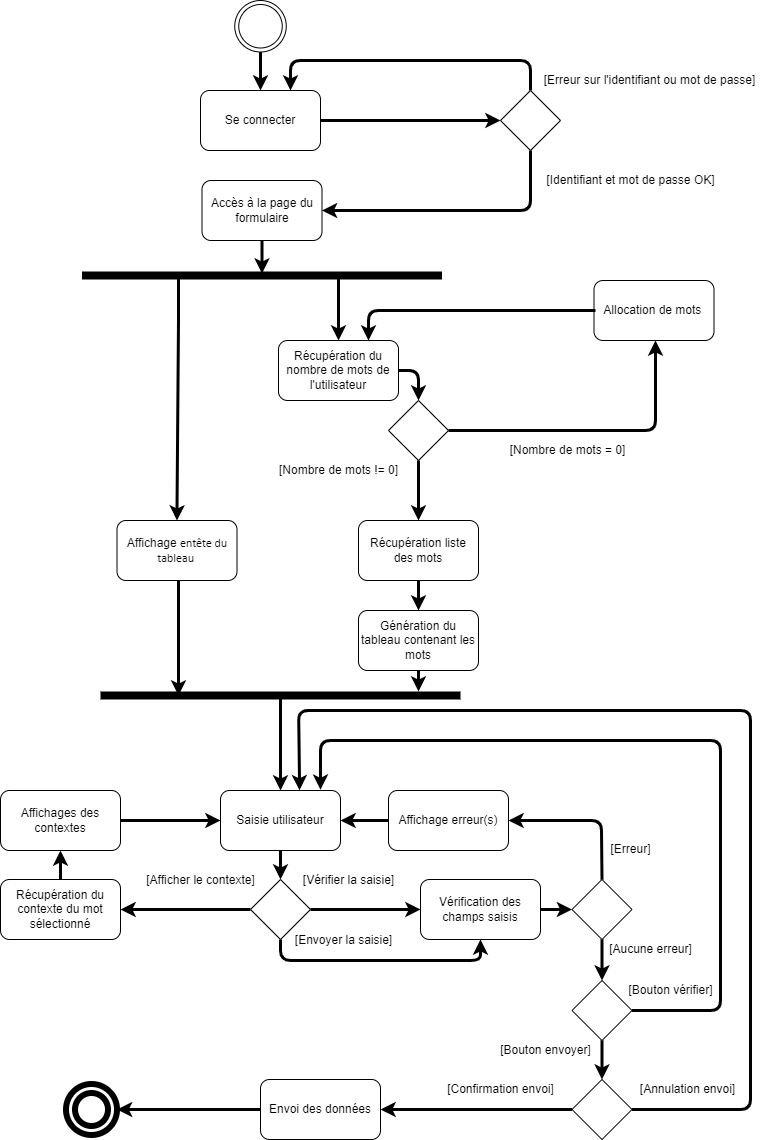


*Modèle Conceptuel de la base de données*

Le système de gestion de base de données utilisé est **MariaDB**, une version **Open Source** de **MySQL** d’Oracle utilisant aussi le langage SQL (Structured Query Language).

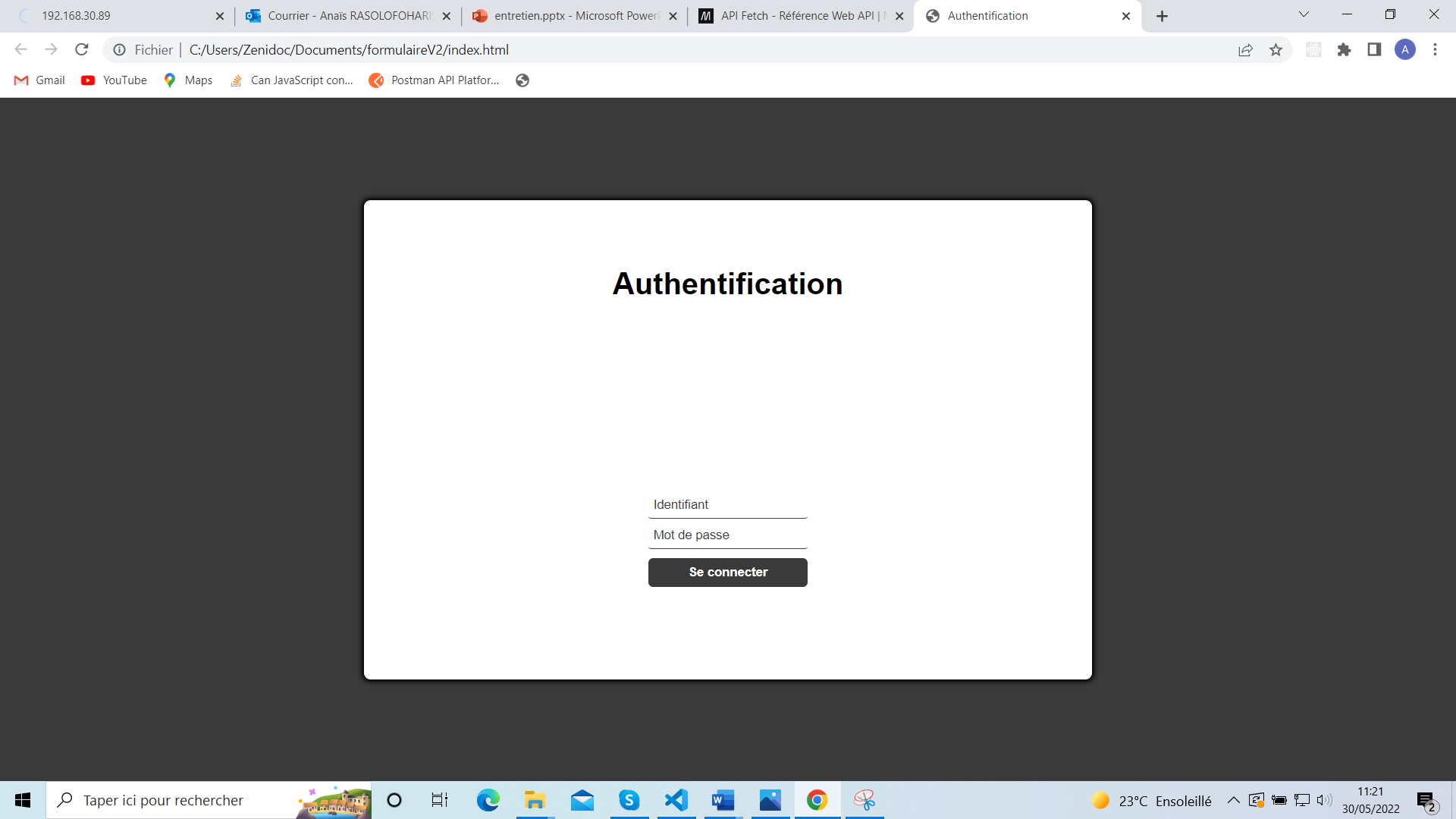
**C – Diagramme d’activité**

Le diagramme d’activité suivant permet de décrire le fonctionnement global du site lorsqu’il est utilisé par un utilisateur.

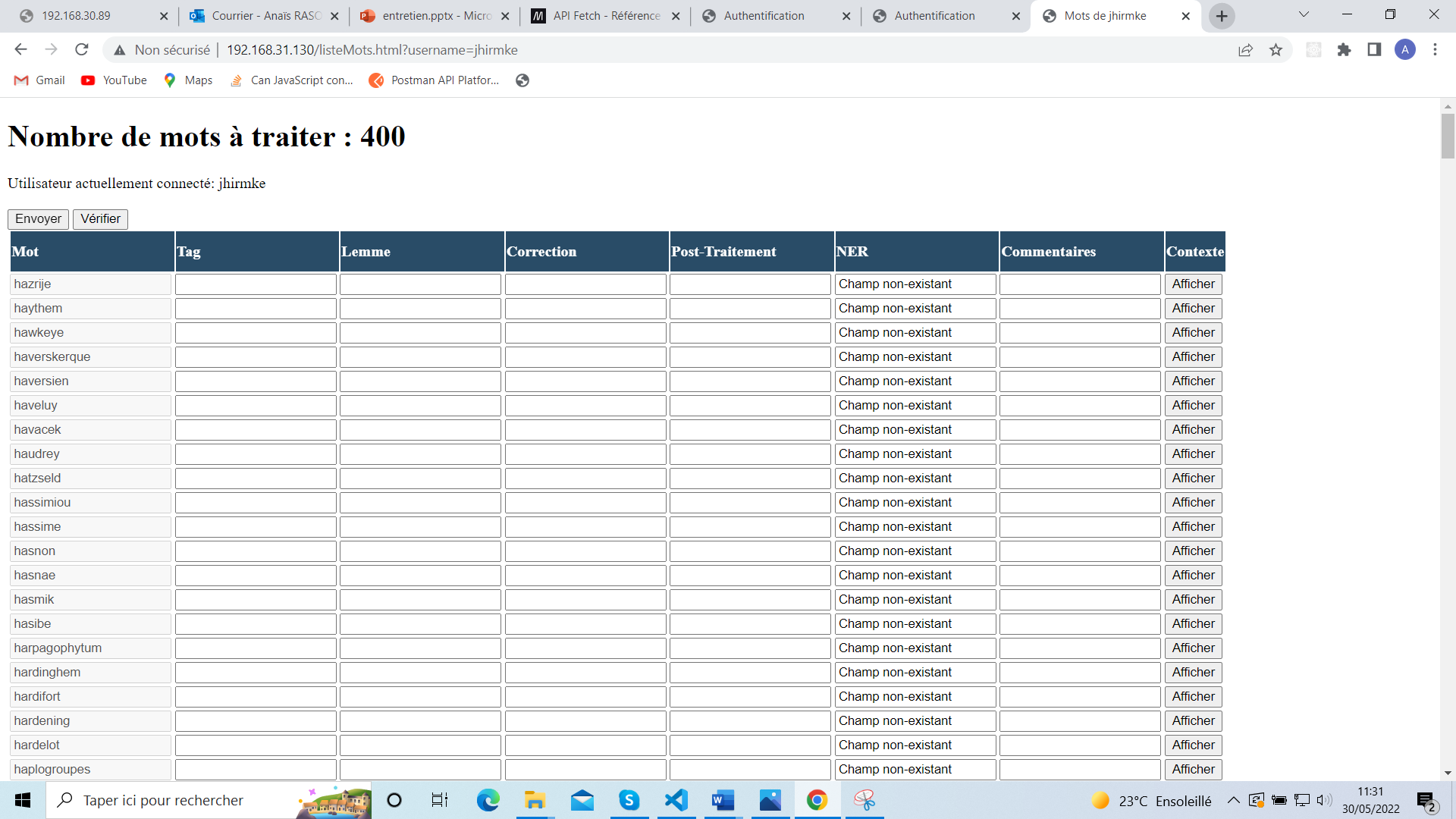


**IV – Annexes**

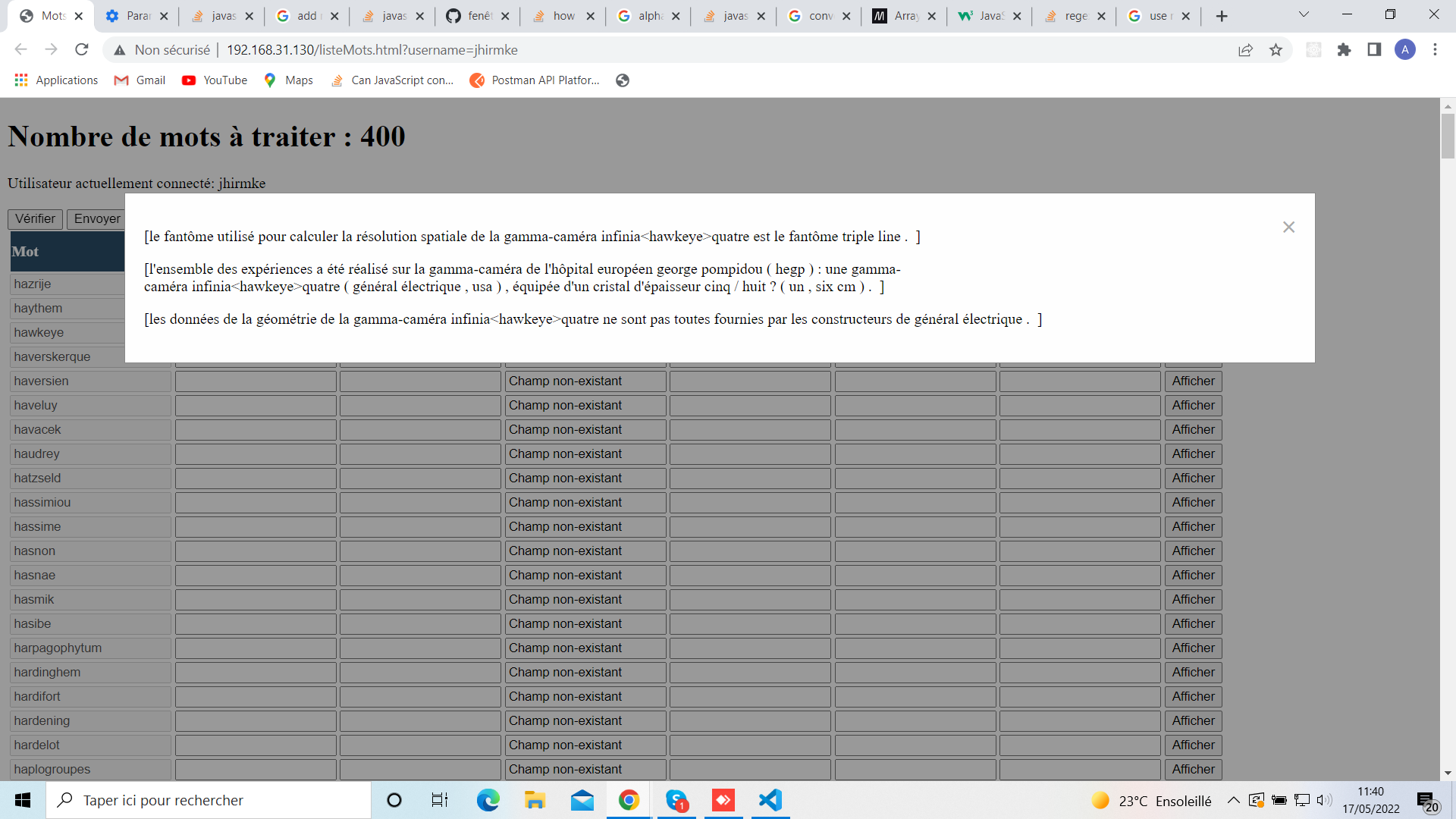
**Annexe 1 : Interface de connexion**



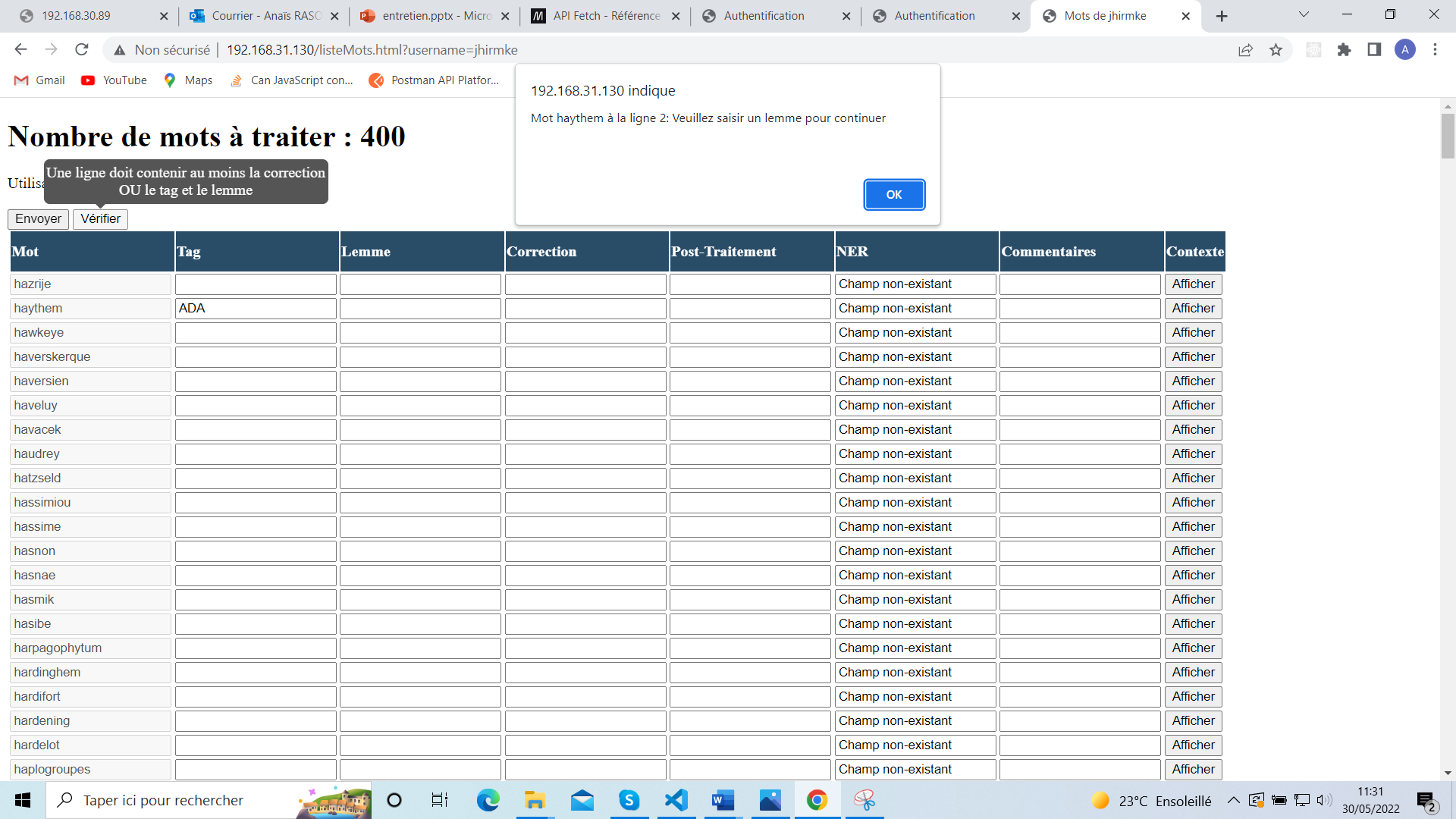
**Annexe 2 : Exemple du formulaire**

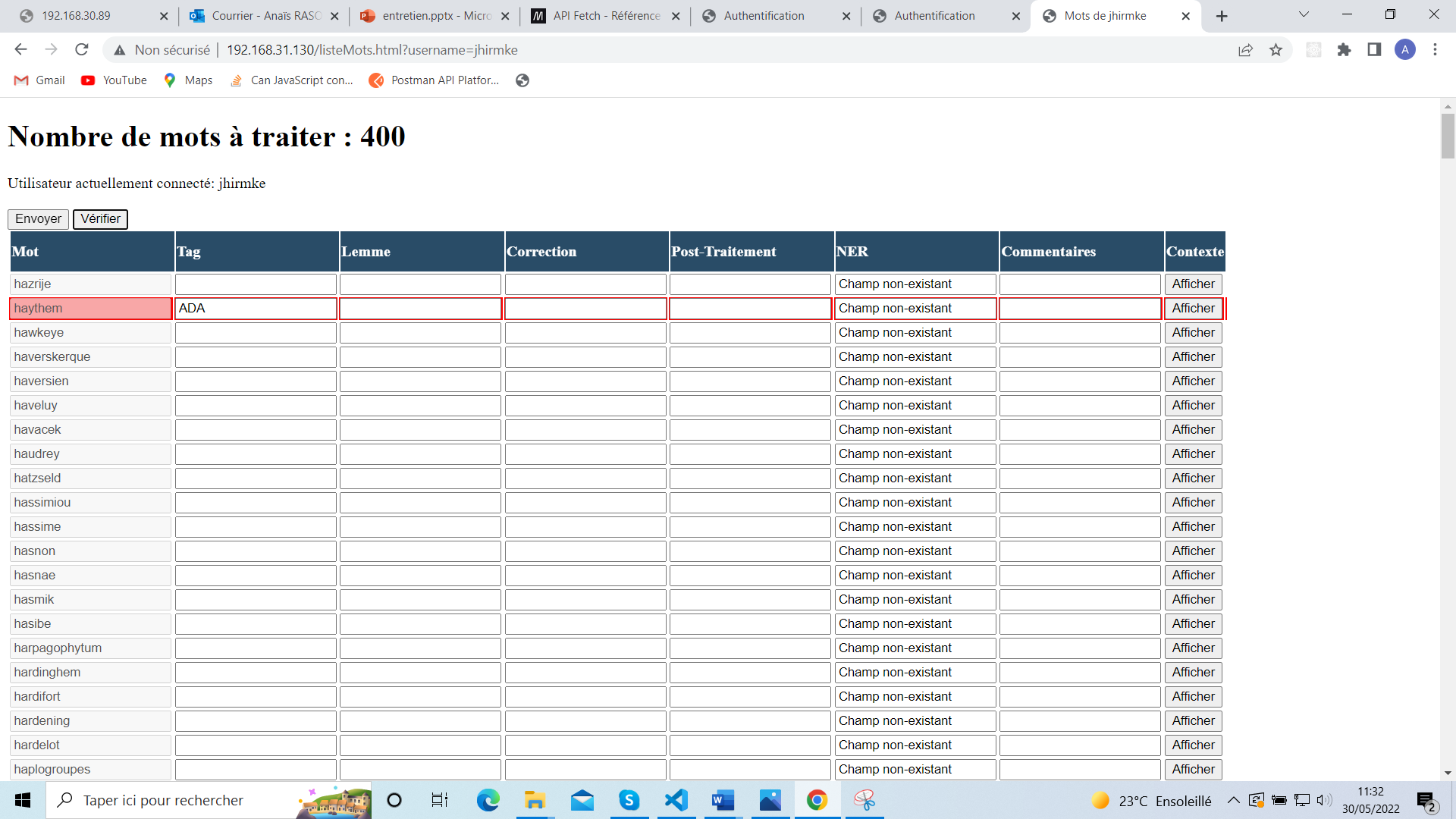


**Annexe 3 : Exemple de contexte**

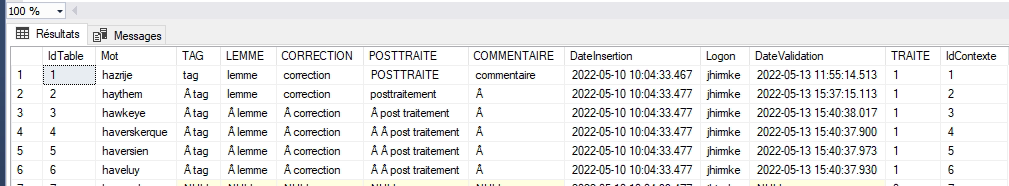
****

**Annexe 4 : Exemple d’erreurs**





**Annexe 5 : Champs dans la base de données**

****

**Annexe 6 : Fetch**

function getDatafromZenidoc(

    portal = "",

    command,

    p1 = "",

    p2,

    p3) {

    portal = "http://192.168.31.130";

    const url = portal + "/portail.iz?DoCommand?logon=14fd95745de35f254ca958c73e546a11&command=" + command + "&p1=" + p1 + "&p2=" + p2 + "&p3=" + p3;

    fetch(url, {

        method: "GET",

    }).then(response => {

        return response.text();

    }).then(function (data) {

        var body = data

        if (body.toString().startsWith("{")) {

            result = body.replace('},\n\n]}', '}]}');

            result = JSON.parse(result)

            //identification du point d’entrée dans la fonction

            if (command == "customVERIFLISTE" || command == "customAttributionListe"){

                //on affiche le nb de mots à l'utilisateur

                document.getElementsByTagName("h1")[0].getElementsByTagName("span")[0].innerHTML = result.QUANTITEMOT;

                allocationDeMots(result.QUANTITEMOT);}

            else if (command == "customListeMots"){

                genererTableauDeMots(result);}

            else if (command == "customgetcontexte"){

                //on définit le contexte

                setContexte(result); }

        } else {

            var getFullURl = RegExp(

                '(?<=var target=").\*?(?=";)',

            );

            var getP1 = RegExp(

                '(?<=&p1=).\*?(?=&p2)',

            );

            p1 = body.match(getFullURl).toString();

            p1 = body.match(getP1).toString() ?? "";

            return getDatafromZenidoc(

                portal, command, p1, p2, p3);

        } }).catch(err => {

        console.log(err); })}

API Fetch

<https://developer.mozilla.org/fr/docs/Web/API/Fetch_API>