Word Quest

Documentation technique v1.1

Abel HALLER

Hippolyte PANKUTZ

Charte graphique 🗺️

HEADING — *Banger -36px - Regular*

Subheading — *League Spartan - 28px - Bold*

Title — *League Spartan - 20px - Bold*

Text — *League Spartan -16px - Regular*

Slogan :
Apprendre l'anglais en s'amusant !

Les icônes -> Icons8 : iOS 17 Glyph

Les illustrations -> Icons8 : 3D Business

Structure du site 🦴

Cette partie a pour but d'expliquer la structure principale du site internet, c'est-à-dire, son squelette, sa sécurité et son système de gestion des données.

Application Flask 🌐

En savoir plus sur Flask : Flask

Flask est un micro framework open-source de développement web en Python. Il est conçu pour faciliter le développement d'applications web. Ce module nous permet de gérer les différentes "routes" de notre site internet, il définit la structure globale de notre site.

L'application suit une structure de répertoires standard pour un projet Flask :

- ➔ **templates/ :** Contient toutes les pages html à afficher par le serveur
  - ↳ **auth/ :** Contient les pages d'authentification (login, register, 2fa, password-recovery)
  - ↳ **dashboard/ :** Continent les pages du dashboard (discover, create, profile, etc.)
  - ↳ **games/ :** Contient les pages html des jeux (nom_du_jeu.html)
  - ↳ **emails/ :** Contient les templates d'email
- ➔ **static/ :** Contient tous les éléments annexes (images, css, javascript)
  - ↳ icons/
  - ↳ imgs/
  - ↳ scripts/
  - ↳ styles/
    - ↳ css
    - ↳ js
    - ↳ **games/ :** Continent les fichiers de style des jeux
      - ↳ **nom_du_jeu/ :** Chaque jeu a un dossier
        - ↳ main.css
        - ↳ main.js
- ➔ **games/ :** Continent les fichiers python des mini-jeux
- ➔ **.env :** Fichier contenant toutes les variables d'environnement (clef API, connexion à la DB)
- ➔ **requirements.txt :** Fichier contenant tout les modules utilisés
- ➔ **main.py :** Fichier principal pour lancer le serveur
- ➔ **root.py :** Contient la fonction (`create_connection()` qui renvoi un objet de connexion à MySql)
  - ↳ Il est important de renseigner le bon fuseau horaire dans cette fonction. (`conn.time_zone = '+01:00'`)
    Si celui-ci diffère de celui de votre machine, vous pourriez avoir des problèmes d'affichages et notamment avec le système de gestion des coeurs qui pourrait vous faire attendre 75 minutes (contre 15) pour avoir une nouvelle vie.
- ➔ **sendmail.py :** Contient la fonction send_mail(`send_mail(to, subject, body)`)
- ➔ *Autres fichiers python*

Connexion à la base de données (create_connection(), close_connection()) 💾

Nous avons créé deux fonctions ci-dessous pour permettre une meilleure utilisation de SQL et pour éviter la redondance.

```python
# Create a connection to the database
def create_connection() -> mysql.connector.connection.MySQLConnection:
    conn = mysql.connector.connect(host=_dbserver, database=_dbname, user=_dbuser, password=_dbpass)
    return conn

# Close a connection to the database
def close_connection(conn: mysql.connector.connection.MySQLConnection) -> None:
    if conn:
        conn.close()
```

```python
conn = None
cursor = None
try:
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM my_table")
    cursor.fetchall()
except Exception as e:
    # Gerer l'erreur
finally:
    if cursor:
        cursor.close()
    if conn:
        close_connection(conn)
```

Ainsi, pour exécuter du SQL de manière sécurisé dans notre interface web, il faudra utiliser la structure de code ci-contre.

Cette dernière permet de gérer les erreurs serveurs qui pourrait mettre à l'arrêt l'application, mais aussi permet de fermer la connexion SQL.

Notez que les données de la BD sont ici des variables pythons. Ces dernières récupèrent les informations des variables d'environnement.

Flask Blueprint 🔧

Au vu de la diversité des routes et de la taille du projet, nous avons choisi Flask Blueprint pour la suite. Contrairement à Flask, Flask Blueprint permet la création de route sur d'autres fichiers pythons.

Ainsi chaque page du dashboard a droit à sa propre page python pour créer des fichiers plus "souple" et lisible.

Il faut cependant préciser dans la page main.py que nous avons ajouté un fichier python. À la manière d'un sommaire, main.py répertorie tous les blueprint qui constituent le site.

```python
from games.hangman import hangman_bp
app.register_blueprint(hangman_bp)
```

Ci-contre, un exemple d'ajout du fichier python pour le jeu du pendu.

Sécurité Web 🛟

Aujourd'hui, créer une application web signifie créer une infrastructure sécurisée.

Content Security Policy (CSP)

La Content Security Policy (CSP) est un mécanisme de sécurité web qui permet aux propriétaires de sites web de contrôler les sources de contenu autorisées à être chargées sur leurs pages web. En définissant et en appliquant une CSP, les administrateurs de sites web peuvent réduire le risque d'attaques telles que les injections de script malveillant (comme les attaques XSS - Cross-Site Scripting) en limitant les origines autorisées à partir desquelles le navigateur peut charger des ressources telles que des scripts JavaScript, des feuilles de style CSS, des images, etc.

Pour Word Quest, nous avons opté pour une CSP stricte :

- ➔ **default-src :** seul google font, notre site web est autorisé par défaut
- ➔ **script-src :** seuls les scripts provenant de notre domaine fonctionnent
- ➔ **style-src :** seul google font, les feuilles de style provenant du domaine et les style dans l'html sont autorisés
- ➔ **img-src :** toutes les images sont autorisées (Notamment pour le Quiz)
- ➔ **form-action:** les formulaires ne peuvent qu'être envoyé sur notre site

HTTP Strict Transport Security (HSTP)

HSTS, ou HTTP Strict Transport Security, est un mécanisme de sécurité qui permet à un site web de déclarer aux navigateurs web qu'ils doivent toujours utiliser une connexion sécurisée HTTPS pour interagir avec le site.

Permission Policy

La Permission Policy est définie à l'aide d'en-têtes HTTP envoyés par le serveur web aux navigateurs. Ces en-têtes spécifient les directives concernant les fonctionnalités autorisées (microphone, fullscreen) ou interdites (tout autre accès), ainsi que d'autres paramètres de sécurité.

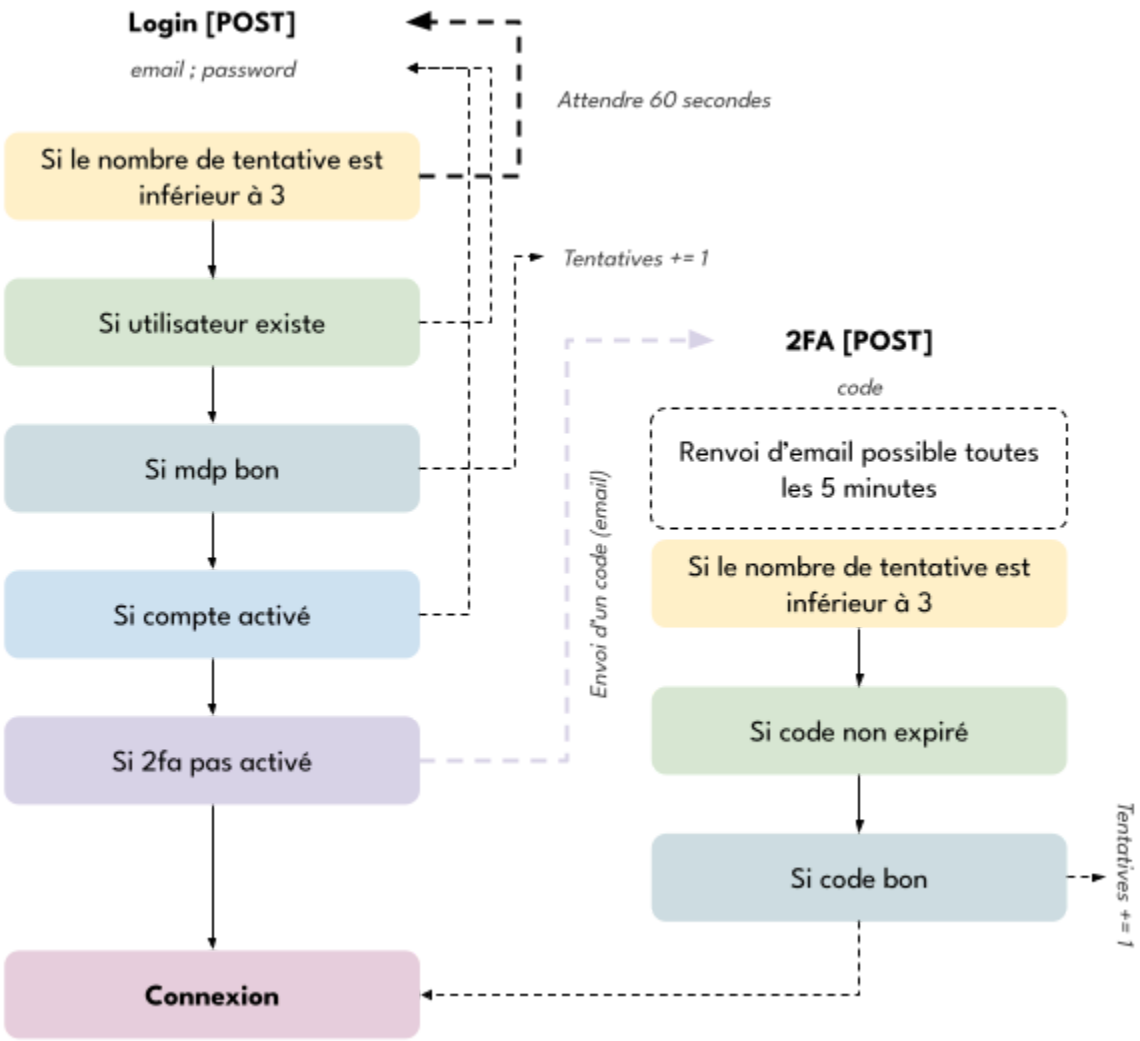
CSRF protection

La protection CSRF (Cross-Site Request Forgery) est une mesure de sécurité utilisée pour protéger les utilisateurs d'un site web contre les attaques ou l'attaquant exploite la confiance établie entre le navigateur et le serveur. Pour ce faire, lorsqu'un utilisateur est connecté à un site, l'attaquant pour l'inciter à cliquer sur un lien qui déclenche une requête HTTP vers le site ciblé, en utilisant les informations de session stockées dans les cookies. Pour faire face à ce genre d'attaques, nos formulaires (de type POST) sont équipés de jeton CSRF qui permet d'identifier la requête comme provenant bien de la page HTML généré par le serveur.

Voici comment ce système fonctionne. Un jeton est généré grâce à la clef de sécurité de l'application. Lorsqu'une requête POST est faite, le serveur vérifie automatiquement ce jeton grâce à la clef secrète de l'application. Cela permet d'authentifier la requête. *(Utilisation de flask wtf -> CSRFProtect)*

```html
<form method="POST" action="">
    <!-- Autres champs -->
    <input type="hidden" name="csrf_token" value="{{ csrf_token() }}">
    <!-- Bouton -->
</form>
```

Authentification 🔐



Une fois toutes ces étapes passées, le système connecté l'utilisateur en utilisant le module Flask Login qui gère les sessions.

L'utilisation de la vérification par email empêche alors les bots de pénétrer notre site internet, ce qui, de fait, augmente sa sécurité.

Enfin, lorsque nous voulons qu'une page/route soit accessible uniquement à un utilisateur connecté, nous utilisons le décorateur python `@login_required` qui permet de vérifier si un utilisateur est connecté et qui, le cas échéant, le redirige vers la page de connexion.

Chiffrement des mots de passes 🧮

Pour ce qui est des mots de passe, ils sont chiffrés lors de leur inscription pour qu'ils ne soient pas affichés clairement dans notre base de données. Pour chiffrer les mots de passe de manière sécurisé, nous utilisons le module bcrypt de python. Voici les étapes du hachage :

1. **Génération d'un sel :** le sel aléatoire est une valeur unique qui est ajoutée au mot de passe avant le hachage. Il rend impossible de pré-calculer les hachages de mots de passe courants.
2. **Le hachage :** le mot de passe et le sel sont ensuite hachés à l'aide de l'algorithme de Blowfish. Comme cet algorithme est assez lent et utilise un grand nombre d'itérations, cela rend le craquage par bruteforce très difficile.
3. **Vérification :** bcrypt récupère le sel du mot de passe haché en base de données, utilise le même nombre d'itérations et la même fonction de hachage et le compare avec le mdp stocké en base de données.

Réinitialisation d'un mot de passe 🔑

La réinitialisation d'un mot de passe est une étape délicate, car elle pourrait mettre en péril les données d'un utilisateur ainsi que son accès à son espace membre.

Pour faciliter cette étape, nous avons préféré un lien sur lequel cliquer plutôt que l'envoi d'un code de vérification. Voici donc les étapes lors du changement de mot de passe.

- **Récupération de l'e-mail :** Si l'email renseigné existe dans notre base de données, nous générons un token valide durant 1 heure, et nous précisons en base de données, qu'une session de récupération de mot de passe est activée.

  Utilisation d'un token de type JWT (Json Web Token)

```
token = jwt.encode({
                'email': email,
                'exp': datetime.datetime.utcnow() + datetime.timedelta(hours=1)
            }, current_app.config['SECRET_KEY'], algorithm="HS256")
             # Generate the password recovery link
            url = request.host_url + "auth/pass-recovery/reset-password/" + token
```

Ici, l'email et la date d'expiration sont stockés dans le token qui est ensuite encodé grâce à la clef secrète de l'application.

- **Vérification :** Lorsque l'utilisateur reçoit l'email, il n'a qu'à cliquer sur le bouton qui le redirigera vers une page d'où il pourrait modifier son mot de passe. Ensuite, il enverra le formulaire et sa demande sera traitée.

```python
@auth_bp.route('/pass-recovery/reset-password/<token>', methods=['POST'])
def pass_recovery_reset_password_post(token):
    try:
        payload = jwt.decode(token, current_app.config['SECRET_KEY'], algorithms=["HS256"])
        # Check if the token has expired
        if datetime.datetime.fromtimestamp(payload["exp"]) > datetime.datetime.utcnow():
            conn = create_connection()
            cursor = conn.cursor()
            cursor.execute("SELECT * FROM users WHERE email=%s", (payload["email"],))
            data = cursor.fetchone()
            if data:
                password = password.encode('utf-8')
                hashed = bcrypt.hashpw(password, bcrypt.gensalt())
                cursor.execute("UPDATE users SET password=%s,\
                    password_recovery_session=NULL WHERE email=%s",
                    (hashed, payload["email"])) # Update the user's password
                conn.commit()
                flash("Mot de passe modifié avec succès")
                return redirect(url_for('auth.login'))
            else:
                flash("Email introuvable")
                return redirect(url_for('auth.pass_recovery'))

    # Etc ...
```

Ainsi, l'utilisateur peut réinitialiser son mot de passe assez aisément, le tout de manière sécurisée.

Profil utilisateurs 👲

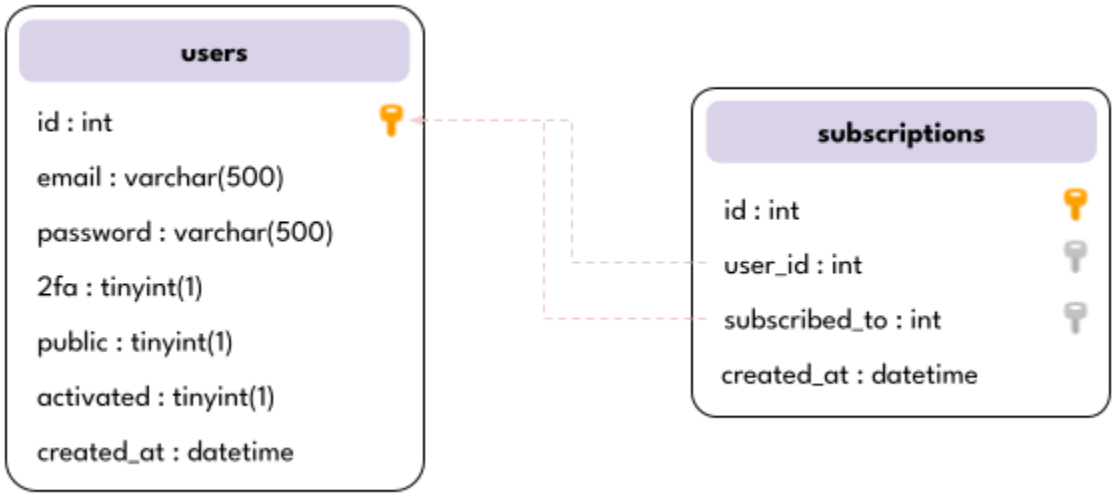
Après son inscription sur notre site internet, chaque utilisateur dispose de sa propre page profile. Nous avons donc mis en place un système de profil public et de profil privé pour être en accord avec le consentement de l'utilisateur.

Ainsi, seule deux possibilités permettent de voir le profil d'un autre utilisateur :

➔ Soit son profil est public
➔ Soit cet utilisateur est abonné à votre compte

Pour vérifier si un profil est public, nous avons un attribut booléen dans notre `user` de notre BD qui nous indique alors la visibilité choisie.

Pour ce qui est du système d'abonnement, nous avons créé une table `subscription` qui enregistre les abonnements de chaque utilisateur. Ci-dessous voici la composition de ces deux tables.



Ici la table `users` n'est pas présentée dans son intégralité

Ainsi, pour vérifier si une page profile est accessible, on doit vérifier si son profil est public en faisant une requête simple. Si ce n'est pas le cas, on récupère les abonnements de l'utilisateur et on vérifie si l'identifiant de l'utilisateur actuel est dedans :

```python
subscriptions = []
cursor.execute("SELECT users.id, users.name, users.picture, \
        SUM(CASE WHEN user_statements.transaction_type = 'xp' THEN user_statements.transaction ELSE 0 END) \
        AS sum_xp FROM subscriptions JOIN users ON users.id = subscriptions.subscribed_to \
        JOIN user_statements ON users.id = user_statements.user_id WHERE \
        subscriptions.user_id = %s GROUP BY users.id, users.name, users.picture;", (user_id,))
result = cursor.fetchall()

for row in result:
    if row[0] is None:
        continue
    subscriptions.append(row)

is_subscribed = False
if not is_public and not is_current_user:
    for sub in subscriptions:
        if int(sub[0]) == int(current_user.id):
            is_subscribed = True
            break
```

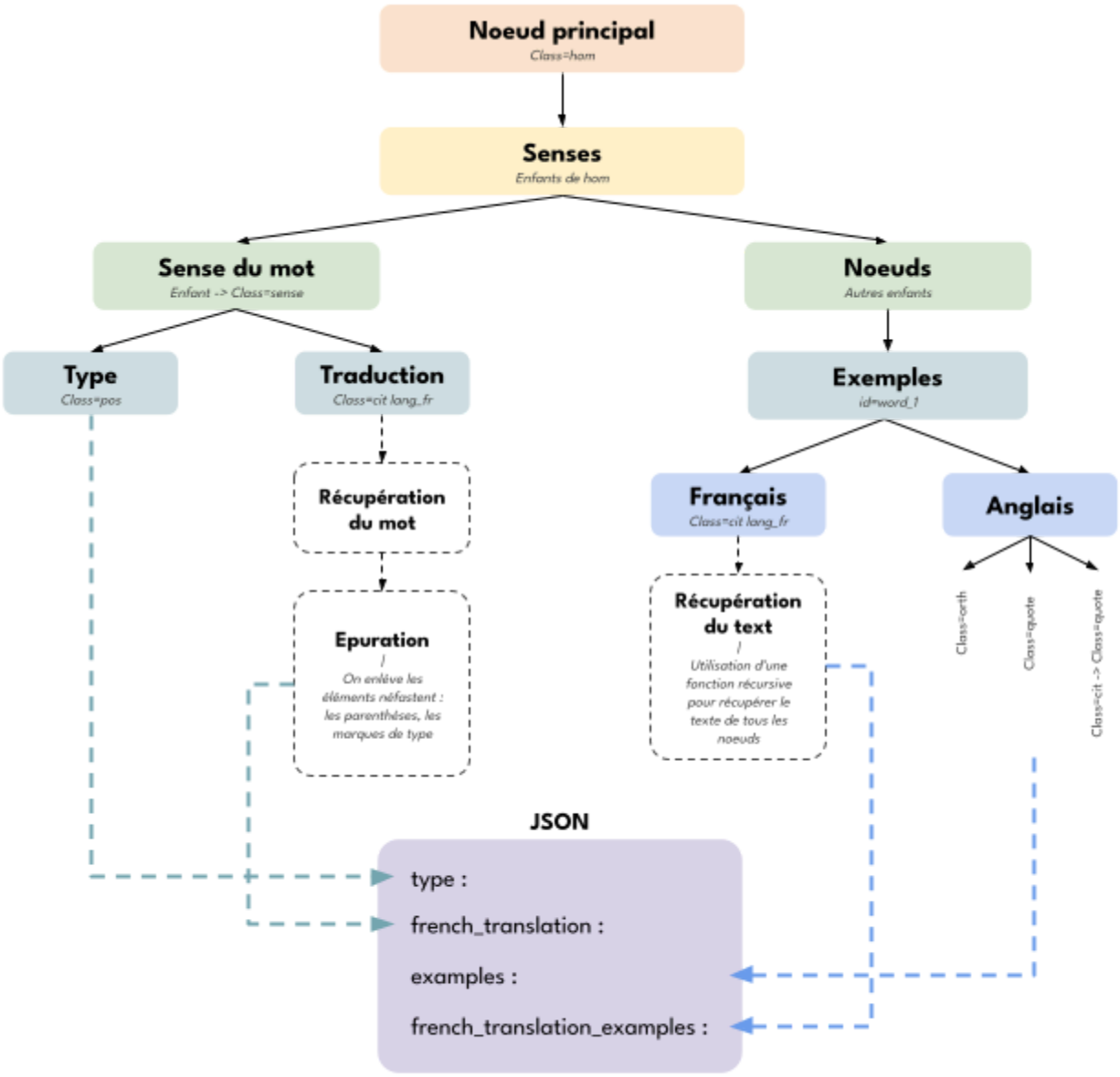
La grande requête SQL permet de non seulement récupérer les abonnements du profil recherché, mais aussi, pour chaque utilisateur auquel il est abonné, sa photo de profil, son nom, et son nombre d'XP total obtenu.

Création de listes ➕

Extraction des données 🤖

Pour la création de liste, nous avons eu besoin des données d'un dictionnaire franco-anglais. Nous avons ainsi utilisé l'API de Collins Dictionary (5000 requêtes par mois), qui nous permet, pour un mot anglais en entrée, de recevoir toutes ses traductions en français, ainsi que des exemples d'utilisation.

Cependant, et bien malheureusement, l'API proposé par Collins ne renvoie les données du dictionnaire qu'en xml ou html, ce qui ne facilite pas alors la récupération de données précise. Pour y faire face, nous avons dû créer un algorithme de récupération des données. Après plusieurs analyses du corps HTML reçu, nous avons créé ceci :



Ainsi, lorsqu'un utilisateur recherche un mot avec le champ recherche de la page create, une requête est envoyée au serveur. Ce dernier envoie alors une requête à l'API de Collins Dictionary, réceptionne la réponse et lance l'algorithme d'extraction des données.

Envoi des données au client 💻

Pour envoyer des requêtes au serveur depuis le côté "client", nous avons utilisé Fetch() de Javascript. Notre serveur doit lui renvoyer les données extraites du dictionnaire de Collins pour ensuite les afficher côté client.

```
{
    "code": "integer",
    "message": "string",
    "result": "dict||list"
}
```

Forme fixe des réponses du serveur

Pour un meilleur travail d'équipe, nous avons défini à l'avance la forme des données renvoyées par le serveur. Voici sa structure :

Enregistrement de la liste en cours de création 🗒️

Lorsqu'un utilisateur, après avoir cherché un mot, veut l'ajouter à sa liste de vocabulaire, il clique dessus, et une requête est envoyée au serveur. Coté "serveur", la liste est enregistrée dans un objet classe qui contient :

- ➔ **Attributs**
  - ↳ **_list :** Contient la liste de vocabulaire en cours de création
  - ↳ **_lasts_earched :** Contient le mot et les traductions du mot qui a été cherché
- ➔ **Méthodes**
  - ↳ **add(id) :** Ajoute les données d'un mot à la liste
  - ↳ **remove(id) :** Enlève un mot à la liste
  - ↳ **get_all() :** Recupère tous les mots de la liste
  - ↳ **length() :** Récupère la taille de la liste
  - ↳ **search(searched) :** Ajoute la recherche en cours à _last_searched

Cependant, dans une application web, pour enregistrer des données pour une courte durée, nous utilisons généralement les sessions. Malheureusement, les objets de type classe ne peuvent pas être enregistrés dans des sessions.

Enregistrement d'une classe dans une variable de session 🗒️

L'idée principale de cette étape est de convertir l'objet de type classe en un objet accepté dans une variable de session. Ainsi, à chaque fois qu'une méthode est appelée par une quelconque route du serveur, voici les étapes à suivre :

1. **Transformer** l'objet contenu dans la variable de session en classe
2. **Appeler la méthode** souhaitée
3. **Transformer** la classe en objet (de type dictionnaire) et mettre à jour la variable de session

Pour ce faire, voici les deux méthodes intégrées à notre classe :

```python
class WordList:
    def to_json(self):
        return json.dumps({
            "_list": self._list,
            "_last_searched": self._last_searched
        })

    @classmethod
    def from_json(cls, json_string):
        data = json.loads(json_string)
        word_list = cls()
        word_list._list = data["_list"]
        word_list._last_searched = data["_last_searched"]
        return word_list
```

L'utilisation de `@classmethod` permet de prendre la classe elle-même en premier argument, plutôt qu'un instance de la classe.

Ainsi, dans une quelconque route :

```python
wordList = WordList.from_json(session['list_under_creation']) # Retrieve the class
wordList.search(senses) # Apply the method
session['list_under_creation'] = wordList.to_json() # Save the class
```

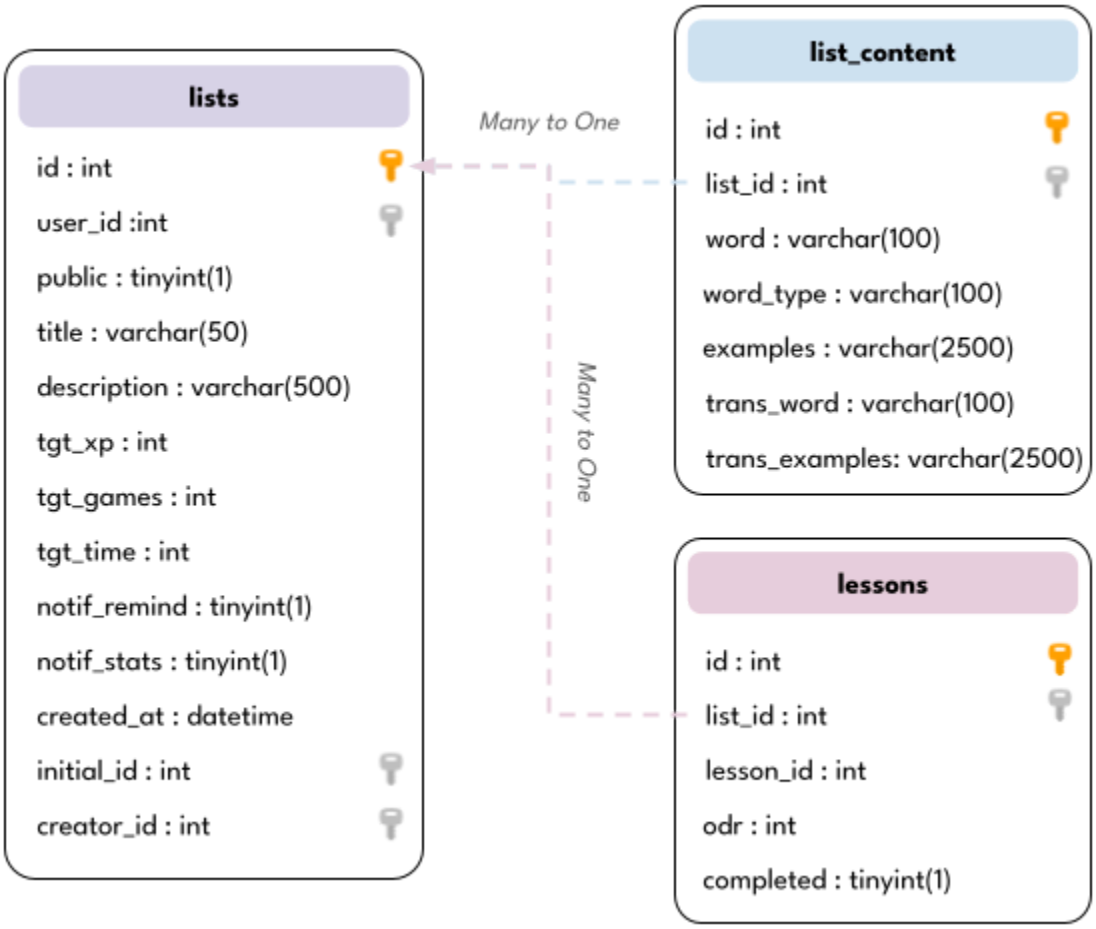
Finalisation de la liste 🔚

Pour créer sa liste, l'utilisateur clique sur le bouton "créer". Ce bouton va envoyer les données du formulaire contenant le nom, la description, les objectifs et les préférences de la liste de l'utilisateur, au serveur. Les données vont être vérifiées et normées, et vont être insérées en base de données.

Ensuite, en récupérant les données contenues dans la classe `WordList`, le système va pouvoir insérer chaque mot, un a un dans la base de données.

Pour finir, le backend génère un parcours à l'utilisateur pour apprendre sa liste. Le parcours est composé de 2 jeux de niveau 1, de 2 jeux de niveau 2, et d'un jeu de niveau 3, soit le niveau le plus complexe.

Voici la structure des tables :



La grande taille des attributs `examples` et `trans_examples` et dû au fait qu'il contienne des objets JSON.

Gestion des listes (visibilité, partage, etc.) 📖

Comme pour le profil des utilisateurs, nous avons ajouté un système de partage de liste pour renforcer l'entraide entre les utilisateurs et pour améliorer leur expérience sur le site internet.

Il est possible de copier une liste par plusieurs moyens :

- ➔ **Depuis la page profil de l'utilisateur :** Lorsque le profil d'un utilisateur est public ou lorsqu'il est abonné à vous, vous pouvez voir ses listes de vocabulaire et les copier pour pouvoir en profiter tout autant.
- ➔ **Depuis la page "Découvertes" :** Lorsqu'un utilisateur crée sa liste de vocabulaire, il peut choisir de la rendre visible pour les autres utilisateurs. Depuis la page "découverte", vous accéderez à toutes les listes publiques et pourrez ainsi les copier.

➔ **Avec un lien de partage :** Lorsqu'un utilisateur crée une liste de vocabulaire, il a la possibilité de la partager à ses amis, en leur envoyant un lien. En cliquant sur ce lien, vous copierez cette liste sur votre compte personnel.

Algorithme de recommandation (Page "Découvertes")

L'algorithme créé de toute pièce est un algorithme très simple qui permet de personnaliser l'expérience utilisateurs. Tout d'abord, nous récupérons les données nécessaires pour toutes les listes de vocabulaire présentes dans la base de données avec cette requête SQL :

```sql
SELECT
    lists.id AS list_id,
    lists.initial_id AS list_initial,
    lists.title AS list_title,
    lists.public AS list_visibility,
    lists.user_id AS list_user,
    JSON_ARRAYAGG(JSON_OBJECT('word', list_content.word, 'type', list_content.word_type)) AS words,
    COALESCE(like_counts.like_count, 0) AS like_count,
    COALESCE(COUNT(DISTINCT routes_log.id), 0) AS page_views,
    CASE WHEN list_likes.user_id IS NULL THEN FALSE ELSE TRUE END AS user_in_like_list
FROM
    lists
LEFT JOIN
    list_content ON lists.id = list_content.list_id
LEFT JOIN
    (SELECT route, COUNT(DISTINCT id) AS id FROM routes_log GROUP BY route) AS routes_log
    ON routes_log.route = CONCAT('/dashboard/profile/list/', lists.id)
LEFT JOIN
    (SELECT list_id, COUNT(DISTINCT id) AS like_count FROM list_likes GROUP BY list_id) AS like_counts
    ON lists.id = like_counts.list_id
LEFT JOIN
    list_likes ON lists.id = list_likes.list_id AND list_likes.user_id = %s
WHERE
    lists.user_id != %s
GROUP BY
    lists.id, lists.title;
```

Cette requête récupère, l'identifiant de chaque liste, son nom, sa visibilité, son propriétaire, les données de la liste (mots et types de chaque mot), le nombre de likes, le nombre de vues, ainsi qu'une valeur booléenne pour savoir si l'utilisateur a liké ou nom la liste.

Explication des spécificités :

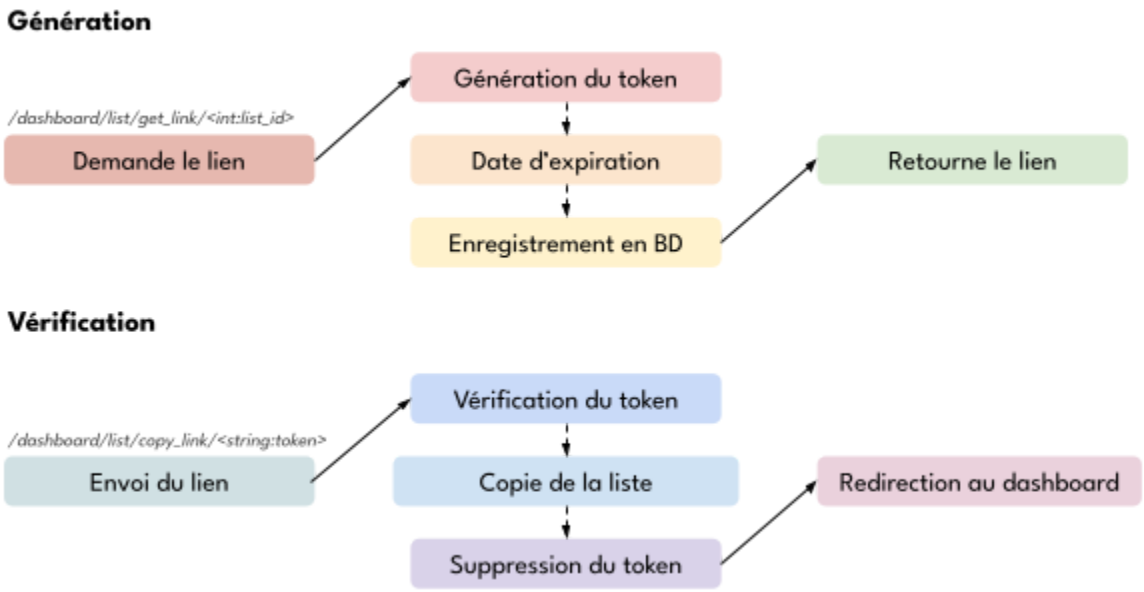
➔ **Ligne 7 :** Cette ligne permet de créer un objet JSON contenant tous les mots de la liste associés à leur type.
➔ **Ligne 8 et 9 :** COALESCE -> Utilisé pour gérer les valeurs NULL et les remplacer par 0
➔ **Ligne 16-17 :** Permet de sélectionner le nombre de vues de manière unique
➔ **Ligne 19-20 :** Permet de sélectionner le nombre de likes de manière unique

Une fois toutes les données récupérées, voici comment sont triées les listes pour s'afficher par ordre de recommandation chez chaque utilisateur :

1. **Visibilité de la liste :** Si la liste n'est pas publique, elle n'est pas affichée
2. **Calcul du coefficient :**
    a. Chaque mot correspondant à un des mots de l'utilisateur augmente le coefficient de 2
    b. Utilisation de `SequenceMatcher` pour vérifier la similarité du titre avec ceux de l'utilisateur
    c. Chaque like compte pour 0,1
    d. Chaque vue compte pour 0,05
3. **Tri en fonction du coefficient :** Les coefficients les plus hauts seront affichés en premier
4. **Affichage dans 2 parties :** Les listes créées par Word Quest sont affichés dans un onglet à part

Algorithme disponible lignes 50 à 124 dans le fichier discover.py

Lien de partage 📎



Chaque utilisateur peut facilement partager ses listes. Le lien est valable une heure et n'est utilisable qu'une seule fois. Le token est généré avec UUID (Universal Unique Identifier).

`/dashboard/list/get_link/<int:list_id>` : se trouve dans le fichier dashboard.py

`/dashboard/list/copy_link/<string:token>` : se trouve dans le fichier create.py

Gestion des statistiques 📈

Les statistiques des utilisateurs sont présentes sur Word Quest pour étoffer encore plus l'expérience utilisateur, pour l'inciter à en faire davantage pour mémoriser son vocabulaire.

Tout d'abord, lorsqu'un utilisateur crée une liste de vocabulaire, il se fixe des objectifs précis. Dans la page "Quêtes", il pourra alors découvrir ses objectifs en cours de réalisation. Ceux-ci correspondent à l'addition des objectifs en cours. Les parcours d'apprentissages complétés ne comptent pas dans l'addition des objectifs.

Ensuite, un indicateur de progression est calculé et est affiché à l'utilisateur. Pour ce faire, le système prend les objectifs actuels de l'utilisateur et les multiplie par 7. Ensuite, on calcul les statistiques de l'utilisateur des 7 derniers jours. Pour finir, on calcule le ratio.

Voici l'algorithme complet :

```
cursor.execute("SELECT DATE(created_at) as day, COUNT(*) as lesson_count, SUM(xp) as total_xp, \
SUM(time) as total_time FROM lessons_log \
WHERE user_id = %s AND DATE(created_at) >= %s GROUP BY day", (current_user.id,seven_days_ago))
result = cursor.fetchall()

results = []
progress = 0
for i in range(7):
    # Get the user's stats for each day
    day = datetime.now().date() - timedelta(days=i)
    day_result = next((res for res in result if res[0] == day), None)
    if day_result:
        progress += day_result[1] + day_result[2] + day_result[3]//60
        target_achieved = day_result[2] >= targets["xp"] and day_result[3] >= targets["time"]\
            and day_result[1] >= targets["games"]
    else:
        target_achieved = False

# Set the progress for the progress bar
if progress != 0 and targets["games"] != 0 and targets["xp"] != 0 and targets["time"] != 0:
    progress = round(progress/(targets["games"]*7 + targets["xp"]*7 + targets["time"]*7)*100, 0)

```

Système de récompense 🏆

Pour mieux fidéliser les joueurs, nous avons mis en place un système de récompense. Si un utilisateur réalise l'ensemble de ses objectifs journaliers (objectifs de temps, de nombre de mini-jeux réalisés, et d'XP), il reçoit un certain nombre de gemmes. Ce nombre de gemmes correspond au ratio entre le nombre d'XP obtenu et le nombre d'XP à atteindre, il est multiplié par 10.

Une fois la récompense attribuée, il est enregistré dans la table `rewards`, que l'utilisateur a bien reçu sa récompense.

Classement 🏅

Pour motiver les joueurs et pour ajouter une certaine compétitivité, nous avons mis en place un système de classement mondial. Celui est parfaitement simple, les joueurs sont classés en fonction du nombre d'XP total gagné.

```sql
SELECT
    u.id as user_id,
    u.picture as user_picture,
    u.name as username,
    SUM(ll.xp) as total_xp,
    RANK() OVER (ORDER BY SUM(ll.xp) DESC) as user_rank
FROM
    users u
JOIN
    lessons_log ll ON u.id = ll.user_id
GROUP BY
    u.id,
    u.name
ORDER BY total_xp DESC;
```

Mini-jeux 🃏

Les mini-jeux sont le cœur même du projet. Ils permettent un apprentissage ludique et personnalisé. Ils permettent aux utilisateurs de gagner des points d'expérience et peuvent leur faire perdre des vies. C'est pour ces raisons que, pour éviter la triche, nous avons mis en place une architecture CLIENT-SERVEUR pour nos jeux.

Gestion des vies ❤️

Lorsqu'un utilisateur termine un jeu, il peut perdre une vie si son nombre d'erreurs est jugé trop grand. Nous avons donc mis un système de 5 vies pour chaque utilisateur. Lorsqu'une vie est perdue, elle met 15 minutes avant de revenir, ou alors, elle peut être récupérée contre 200 gemmes. Ainsi, nous avons donc créé un système qui, automatiquement, rajoute la vie nécessaire à chaque joueur.

Coté client

Pour améliorer l'expérience utilisateur, lorsque l'utilisateur manque d'une vie, un compteur s'affiche pour lui indiquer le temps restant. Et lorsque ce compteur est terminé, une vie lui est rajoutée au niveau de l'interface.

Coté serveur

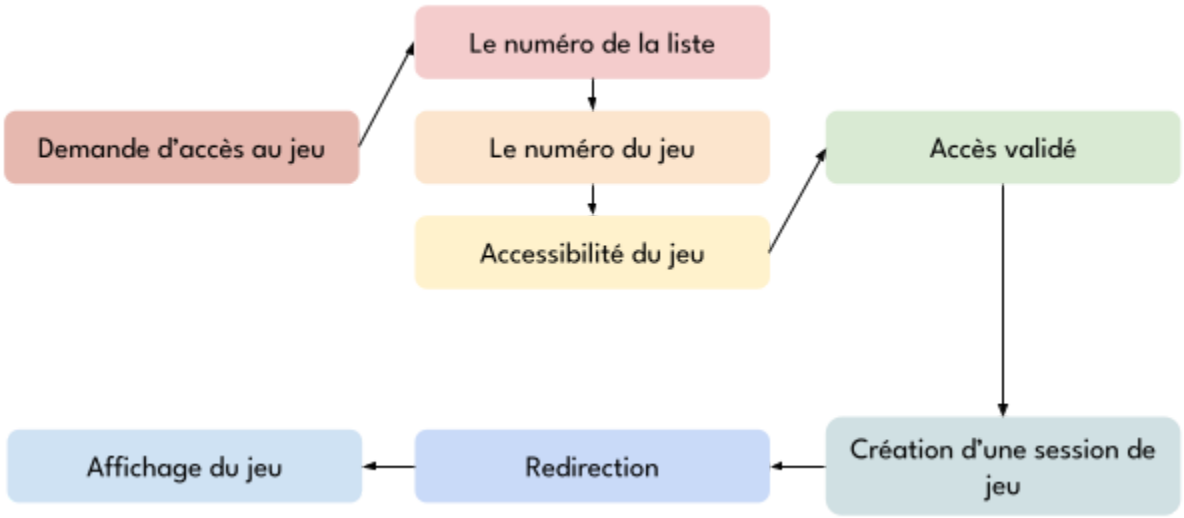
Lorsque l'utilisateur charge la page dashboard (seule page où le nombre de vies apparaît), le système calcul le temps depuis la dernière vie perdue ou gagnée. Si ce dernier dépasse 15 minutes, une vie est ajoutée.

```python
if lives != 5:
            life_time = datetime.strptime(str(lives_time), "%Y-%m-%d %H:%M:%S")
            life_time = life_time + timedelta(minutes=15)
            print(life_time, datetime.now())
            while life_time < datetime.now() and lives < 5:
                cursor.execute("INSERT INTO user_statements (user_id, transaction_type, transaction)\
                    VALUES (%s, 'lives', 1);", (current_user.id,))
                lives += 1
                life_time = life_time + timedelta(minutes=15)
                lives_time = datetime.now()
```

Ici, un défini le moment à partir duquel l'utilisateur est censé récupérer une vie, si ce temps est inférieur au temps actuel est que l'utilisateur n'a pas ses 5 vies, alors on lui en rajoute une, et ainsi de suite.

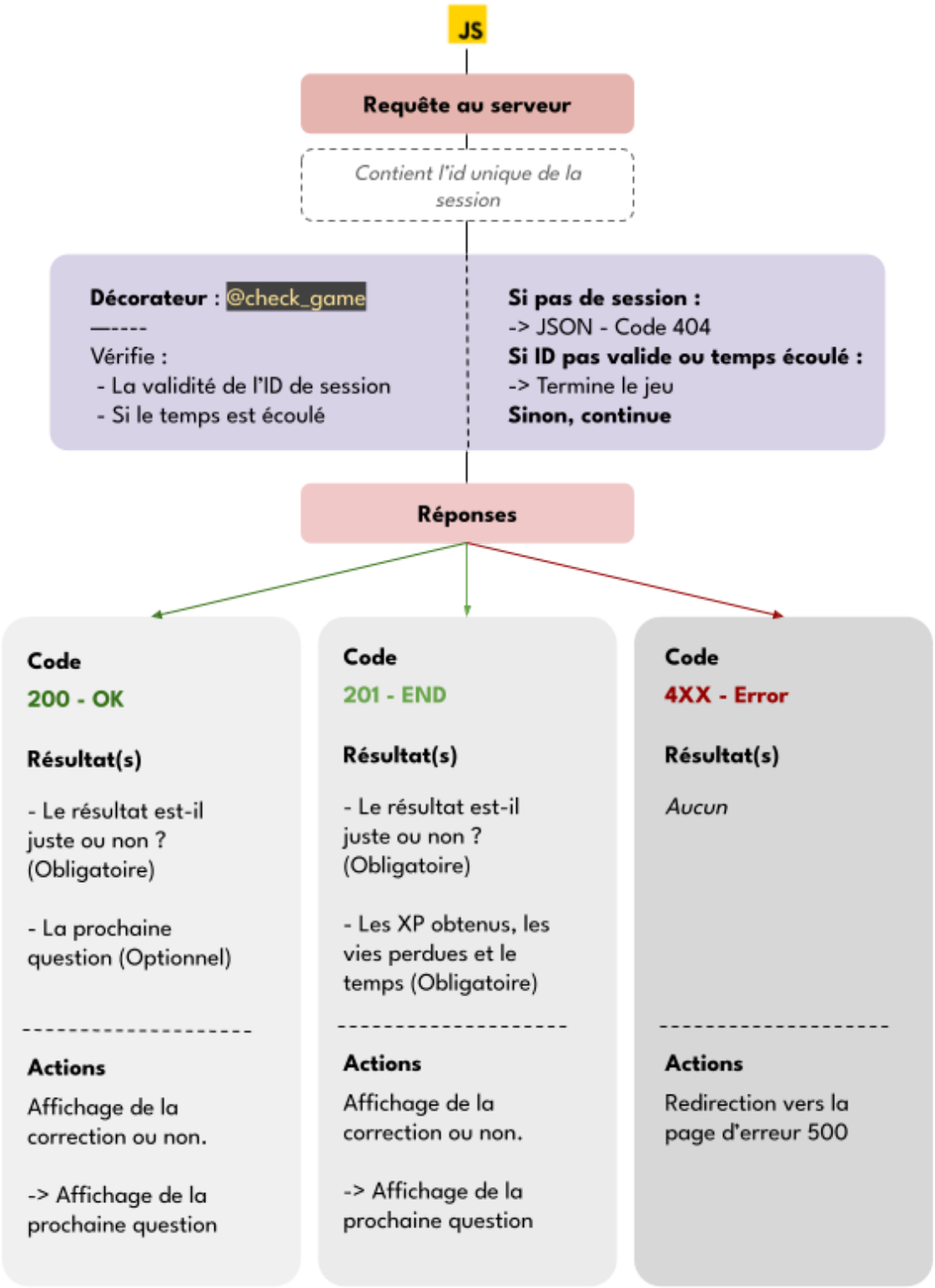
Globalité 🌍

Création d'une session de jeu



Nous fonctionnons avec la programmation orientée objet pour la création des jeux. L'idée est que la session de jeu soit enregistrée côté serveur afin de limiter au maximum la triche. Ainsi, un identifiant unique est attribué à chaque utilisateur, lui permettant d'accéder à sa session de jeu. Dans la plupart des cas, si l'utilisateur recharge par inadvertance la page de son jeu, comme les données sont stockées côté serveur, il pourra retrouver sa session intacte.

Lors d'une session de jeu



Un décorateur en Python est une fonction qui prend une autre fonction comme argument et qui renvoie une nouvelle fonction. Il est utilisé pour modifier ou étendre le comportement de la fonction originale sans la modifier directement. Dans notre cas, le décorateur `@check_game` est utilisé pour vérifier si le jeu est toujours en cours avant d'exécuter certaines fonctions associées aux routes du jeu de quiz.

```python
def check_game(func):
    @wraps(func)
    def wrapper_function(session_id, *args, **kwargs):
        if 'game' in session:
            game = game_name.from_json(session["game"])
            # Check if the game is still in progress
            if session_id != game.id or game.time < str(datetime.datetime.now()):
                response = game._end_game()
                session["game"] = game.to_json()
                return response
            else:
                return func(session_id, *args, **kwargs)
        return jsonify({
            "code": 404,
            "message": "Le jeu n'a pas été trouvé!",
            "result": []
        })
    # Return the wrapper function
    return wrapper_function
```

Ainsi, lors de la création d'une nouvelle route pour un jeu, il est impératif de spécifier le décorateur pour garantir la sécurité du jeu.

```python
@quiz_bp.route('/dashboard/games/quiz/<string:session_id>/audio')
@check_game
def audio(session_id):
    pass
```

Lorsque l'on crée une interface CLIENT-SERVEUR, il faut impérativement veiller à limiter au maximum le nombre de requêtes au serveur. Premièrement, les requêtes au serveur ralentissent le jeu. En effet, lorsque le serveur n'est pas parfaitement performant, les requêtes peuvent mettre de 200 à 1000 ms avant de recevoir une réponse. Deuxièmement, il faut préserver les ressources du serveur. Un nombre trop important de requêtes pourrait surcharger le serveur et ralentir tout le site internet.

—-----

De la même manière que pour la création de la liste, les données du jeu sont enregistrées dans une classe. Cette classe contient des méthodes spécifiques pour chaque jeu, mais aussi des méthodes et attributs universels dans notre système :

Artributs :

- ➔ id : L'identifiant unique de la session de jeu en cours
- ➔ list_id : L'identifiant unique de la liste sur lequel l'utilisateur s'entraîne
- ➔ lesson_id : L'identifiant unique de la leçon en cours
- ➔ words : Les mots de l'utilisateur sous la forme :

```
{
    "word": "dance",
    "type": "verb",
    "examples": [
        "I love to dance.",
        "She danced with him."
    ],
    "trans_word": "danser",
    "trans_examples": [
        "J'adore danser.",
        "Elle a dansé avec lui."
    ]
}
```

- ➔ words_to_check : (Optionnel) Les mots restants à faire (présents dans le quiz, le typeFast et le hangman)
- ➔ time : Le temps maximal d'une session de jeu (généralement 1 ou 2 minutes)
- ➔ start : La date de commencement du jeu

Méthodes :

- ➔ get_remaning_time() : Donne le temps restant, en soustrayant time avec la date actuelle.
- ➔ get_words_checked() : (Optionnel) : Donne la liste des mots qui n'ont pas été faits.
- ➔ _lose_life() : Permet de faire perdre une vie à l'utilisateur en cas de trop grosses fautes.
- ➔ _end_game() : Termine le jeu :
    - ↳ Marque la leçon comme complétée
    - ↳ Calcul le temps passé (Date actuelle moins start)
    - ↳ Calcul les points d'expérience gagnés (si la leçon est déjà complété alors, on fait x0.66)
    - ↳ Fait perdre des vies (Optionnel)
    - ↳ Insère les données dans la base de données
        - → Met l'attribut *completed* de la leçon à 1 dans `lessons`
        - → Ajoute l'xp gagné, le temps passé, etc. dans `lessons_log`
    - ↳ Renvoi un JSON avec un code 201 (Voir plus haut)

Ces attributs et méthodes permettent d'harmoniser nos jeux et nous permettent d'en créer d'autres plus aisément.

Spécificités 🔎

Jeu du quiz

Une des spécificités de ce jeu est le fait que les 4 possibilités de réponses proposées à l'utilisateur contiennent, le bon mot, mais aussi 3 mots similaires.

Cependant, pour réaliser ceci, il nous fallait calculer la distance de Levenshtein *(distance, au sens mathématique du terme, donnant une mesure de la différence entre deux chaînes de caractères)* pour les 370 000 mots anglais et ensuite, triées par les 3 plus ressemblants. Cette opération pouvait prendre jusqu'à 1 seconde en local et jusqu'à plus de 5 secondes sur un serveur.

Nous avons donc créé un programme qui a calculé, pour chaque mot anglais, les 10 mots qui lui ressemblaient le plus, et nous avons ensuite créé un fichier texte comme celui-ci (15 heures d'exécution ont été nécessaires) :

cuve: cave, cive, couve, cove, cube, cue, cuke, cure, curve
cuvee: cuve, cadee, cavae, cave, cavea, caveae, caved, cavey, cavel

Comme ce fichier faisait plus de 40 Mo, nous l'avons divisé en deux, et, pour ne pas avoir à charger le fichier de 20 Mo à chaque fois, nous avons utilisé un algorithme de recherche dichotomique. En utilisant linecache de python, cela nous a permis de n'ouvrir qu'une ligne spécifique du fichier. *(Lignes 229 à 264 fichier /games/quiz.py)*

Si malheureusement, aucun mot n'est trouvé, alors la question change et devient plus globale. On demandera alors le type du mot à l'utilisateur.

MemoWordRize

Plusieurs spécificités existent dans ce jeu. Tout d'abord, le parcours généré se compose de 7 cases. La première case est pris au hasard (Nombre entre 1 et 5) et les 6 cases suivantes sont choisies en fonction :

```python
positions = [starting_position]
self.game_count += 1
for i in range(1, 6):
    # Add the next position
    next_positions = [positions[-1]]
    # 3 possibilities: go to the right, go to the left, stay at the same position
    # If the position is at the edge, the player can only go to the opposite direction
    next_positions.append(positions[-1] + 1 if positions[-1] < 5 else 5)
    next_positions.append(positions[-1] - 1 if positions[-1] > 1 else 1)
    positions.append(random.choice(next_positions)) # Add the next position
```

- Si la case précédente est toute à gauche ou tout à droite, alors 2 possibilités existent
- Sinon trois possibilités existent

Après avoir choisi le parcours complet, un mot est attribué à chaque case. Le mot est choisi au hasard parmi la liste de mot de l'utilisateur, et un identifiant unique est donné au mot :

```python
path = [] # The path of the game
for index, position in enumerate(positions):
    # Select a random word to place in the path
    word_choice = random.choice(self.words_to_check)
    dict_position = {
        "id": str(uuid.uuid4()),
        "line": index + 1,
        "position": position + index * 5, # Calculate the position of the case in html
        "word": word_choice,
        "checked": False
    }
    path.append(dict_position) # Add the word to the path
```

Et enfin, lorsque le parcours est montré à l'utilisateur, le mot doit être prononcé par un outil de synthèse vocale. La spécificité de ce dernier réside dans le système de sécurité mis en place.

Lorsque le JavaScript fait entendre l'audio à l'utilisateur, il appelle la route suivante : `/dashboard/games/memowordrize/<string:session_id>/audio/<string:id>`

Cette route utilise l'id pour récupérer le mot et renvoyer l'audio correspondant. Sans aucune sécurité, l'utilisateur pourrait se servir de l'URL pour réécouter le mot plusieurs fois. Nous avons donc contré ceci en modifiant l'identifiant unique qui permet au système de lire un audio. Après cette modification, l'URL devient obsolète et aucune triche ne peut être engagée par l'utilisateur.

TypeFast

Aucune spécificité (Code entièrement documenté)

Hangman

Pour le jeu du Pendu, nous n'avons pas eu d'autres choix que de stocker le mot côté serveur. Sans ça, un utilisateur malicieux pourrait accéder à la variable contenant le mot et ainsi tricher. Ainsi, à chaque lettre entrée par l'utilisateur, une requête est envoyée au serveur.

Le serveur vérifie si la lettre est d'ores et déjà utilisée, dans le cas contraire, si elle est dans le mot, il l'ajoute à une liste `good_letters` et sinon à une liste `bad_letters`.

Pour ce qui est de la partie affichage, nous avons utilisé un SVG en HTML pour les différentes parties du corps du pendu. Ces dernières s'affichent au fur et à mesure des erreurs de l'utilisateur.

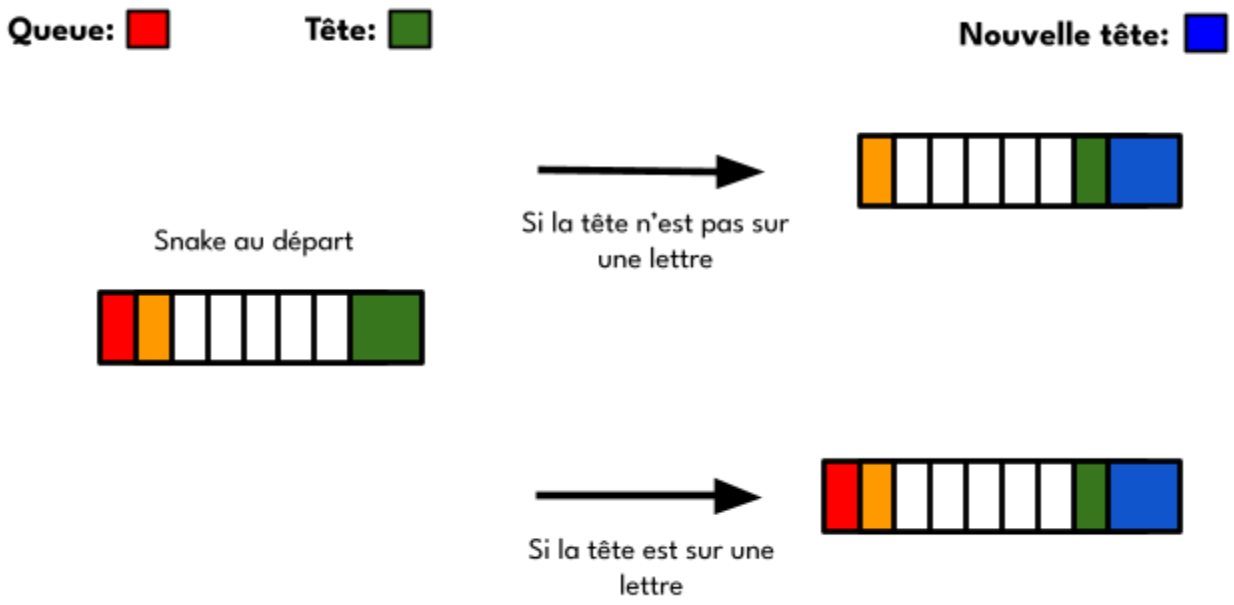
```svg
<svg height="500" width="600" class=figure-content id="svg">
    <!-- Support-->
    <line x1="30%" y1="10%" x2="75%" y2="10%" />
    <line x1="75%" y1="10%" x2="75%" y2="25%" />
    <line x1="30%" y1="10%" x2="30%" y2="85%" />
    <line x1="5%" y1="85%" x2="55%" y2="85%" />

    <!--stickman-->
    <!--head-->
    <circle cx="75%" cy="34%" r="6%" class="figure-part"/>
    <!--body-->
    <line x1="75%" y1="42%" x2="75%" y2="62%" class="figure-part" />
    <!--arms-->

    <line x1="75%" y1="50%" x2="65%" y2="45%" class="figure-part" />
    <line x1="75%" y1="50%" x2="85%" y2="45%" class="figure-part" />

    <!--legs-->
    <line x1="75%" y1="62%" x2="70%" y2="75%" class="figure-part" />
    <line x1="75%" y1="62%" x2="80%" y2="75%" class="figure-part" />
</svg>
```

Snake

Pour programmer le Snake, nous avons dû utiliser le composant canvas. Ce dernier permet de dessiner des formes, insérer du texte ou encore des images dans une zone donnée. Ces éléments sont placés grâce à des coordonnées. À la manière d'un film ou d'un jeu vidéo, le canvas se rafraîchit toutes les 20 millisecondes (50 fps) de manière à créer un rendu fluide.

Le snake est donc un ensemble de carrés dessiné sur le canvas. Sur le canvas, les cases ont pour taille 32x32 et le snake avance de 2px toutes les 20 millisecondes, il doit donc avancer 16 fois pour parcourir une case. Ainsi, le snake est, au départ, composé de 16 carrés auxquels on ajoute 16 autres carrés à chaque fois que la bonne lettre est mangée.

Pour le faire avancer, on ne déplace pas les carrés du snake, mais on crée à chaque fois un nouveau carré (Nouvelle tête) et on supprime le carré qui constitue sa queue comme présenté ci-dessous :



Pour la génération des lettres du mot et de leurs coordonnées, C'est le Python qui s'en occupe, il renvoie donc au JavaScript la liste des lettres et de leurs coordonnées. Ainsi, lorsque l'utilisateur joue, les coordonnées des lettres sont récupérées et traitées par le JavaScript. Elles ne sont pas directement renvoyées au Python afin de rendre le jeu fluide. Ce n'est que quand l'utilisateur finit son mot ou qu'il perd que les coordonnées des lettres sont renvoyées au Python. Le Python s'occupe alors de vérifier si les coordonnées renvoyées par le JavaScript ne sont pas erronées, empêchant ainsi l'utilisateur de tricher.

Memory

Ici aussi, une requête est envoyée au serveur lorsque l'utilisateur retourne une carte. Cela permet de "cacher" le mot qui se trouve en dessous. Ainsi, aucun utilisateur ne peut utiliser l'outil inspecter et voir ainsi le contenu de toutes les cartes.

Il faut savoir que, les mots qui composent la liste de vocabulaire du jeu en cours ont la même structure qu'annoncé juste avant *(page 22 de cette documentation)*. Ainsi, nous récupérer les mots anglais et français dans le même ordre comme ceci :

```
1  self.french_words = [[word['trans_word'], str(i), "french"] for i, word in enumerate(self.words)]
2  self.english_words = [[word['word'], str(i), "english"] for i, word in enumerate(self.words)]
```

Ainsi, l'identifiant du mot anglais et le même que l'identifiant du mot français. Par la suite, les mots sont mélangés.

Pour vérifier si les deux cases retournées sont les bonnes, il suffit de vérifier si l'identifiant du mot anglais correspond avec l'identifiant du mot français.

Falling Word

Ce jeu est un jeu de survie qui dure 30 secondes dans lesquelles des duos de mot anglais/français tombent peu à peu. Pour assurer une fluidité au niveau de l'apparition des duos de mots, il a fallu créer à l'avance 30 duos de mots en Python. Ces duos de mots sont renvoyés au JavaScript avec un id correspondant et la réponse (si le duo est juste ou faux). Ensuite, de la même manière que pour le snake, les mots cliqués par l'utilisateur pendant sa partie sont d'abord traités dans le JavaScript où l'on crée une liste des id de duo cliqué par l'utilisateur. Cette liste est envoyée dans une requête Python qui va vérifier si les id des boîtes cliqués correspondent bien aux réponses enregistrées dans le Python.

Annexes 📁

Animations sur le dashboard 🎇

Pour augmenter et pousser l'expérience utilisateur au maximum, nous avons créé diverses animations. En plus d'avoir mis un loader sur chaque bouton pour prévenir du temps de réponse du serveur, nous avons ajouté une animation de changement de page sur le dashboard.

Nous avons remarqué que lorsque nous cliquons sur un lien, la page sur laquelle nous sommes reste affichée jusqu'à ce que la prochaine soit chargée. Ainsi, comme notre dashboard possède une partie fixe (la barre de navigation à gauche), à chaque fois qu'un lien est cliqué, une animation se lance puis la redirection s'effectue.

L'animation retire les éléments changeant de manière dynamique, et après cela, l'utilisateur est redirigé. Sur la page sur laquelle l'utilisateur arrive, une animation en css est mise en place, et les éléments changeants reviennent.

Fonction redirect(el, event) disponible lignes 15 à 43, fichier /sources/static/styles/js/main.js

Bibliothèques utilisées 📚

Pour assurer le bon fonctionnement de notre projet, nous utilisons plusieurs bibliothèques Python tierces. Ces bibliothèques offrent une variété de fonctionnalités allant de la gestion des sessions à l'accès à la base de données, en passant par l'authentification et bien plus encore. Voici une liste détaillée des bibliothèques utilisées, accompagnée de leurs versions respectives :

Flask (Version 3.0.0)

Flask est un micro-framework Web pour Python. Il offre une structure de base pour le développement d'applications Web, y compris la gestion des routes, des requêtes et des réponses HTTP. Flask est au cœur de notre application, fournissant l'infrastructure nécessaire pour gérer les requêtes utilisateur et renvoyer les réponses appropriées.

Flask-Login (Version 0.6.3)

Flask-Login fournit un système de gestion de session utilisateur pour Flask. Il permet aux utilisateurs de s'authentifier, de se connecter et de gérer leur session sur notre application Web de manière sécurisée. Flask-Login est souvent utilisé en conjonction avec Flask-Session pour stocker les informations de session côté serveur.

Flask-Session (Version 0.5.0)

Flask-Session étend les fonctionnalités de session de Flask en permettant le stockage des informations de session côté serveur. Il est souvent utilisé avec Flask-Login pour fournir une gestion de session robuste et sécurisée.

Flask-Talisman (Version 1.1.0) et Flask-WTF (Version 1.2.1)

Flask-Talisman renforce la sécurité des applications Flask en configurant des en-têtes de sécurité HTTP. Il aide à protéger contre les attaques telles que l'injection de contenu malveillant, la falsification de requête inter-sites (CSRF), et d'autres vulnérabilités liées à la sécurité Web.

bcrypt (Version 4.1.2)

bcrypt est une bibliothèque de hachage de mot de passe utilisée pour stocker les mots de passe de manière sécurisée. Il est souvent utilisé en combinaison avec Flask-Login pour sécuriser l'authentification des utilisateurs.

pyotp (Version 2.9.0)

pyotp permet de générer des jetons à usage unique (OTP) conformes au standard Time-Based One-Time Password Algorithm (TOTP). Il est souvent utilisé pour l'authentification à deux facteurs (2FA) ou pour renforcer la sécurité des comptes utilisateur.

python-dotenv (Version 1.0.0)

python-dotenv charge les variables d'environnement à partir d'un fichier .env dans les applications Python. Cela permet de séparer les paramètres de configuration sensibles du code source, améliorant ainsi la sécurité et la portabilité de l'application.

PyJWT (Version 2.8.0)

PyJWT est une bibliothèque Python pour travailler avec JSON Web Tokens (JWT). Il est souvent utilisé pour l'authentification stateless, où les informations d'identification de l'utilisateur sont encapsulées dans un token JWT.

mysql-connector (Version 2.2.9)

mysql-connector fournit une interface Python pour se connecter à une base de données MySQL. Il est souvent utilisé avec Flask pour accéder et manipuler des données stockées dans une base de données MySQL.

requests (Version 2.31.0)

Requests est une bibliothèque HTTP pour Python, permettant d'envoyer des requêtes HTTP de manière simple. Bien qu'elle ne soit pas spécifique à Flask, elle est couramment utilisée pour effectuer des requêtes HTTP vers des API externes dans les applications Flask.

lxml (Version 5.1.0)

lxml est une bibliothèque Python utilisée dans notre application pour convertir les données HTML provenant d'API externes en une structure de données exploitable. Cette conversion facilite le traitement et l'analyse des données, améliorant ainsi la fonctionnalité de notre application.

Google-Images-Search (Version 1.4.6)

Google-Images-Search est une API Python pour rechercher des images sur Google. Nous l'utilisons dans le jeu du quiz pour rechercher des illustrations sur le web.

python-Levenshtein (Version 0.25.0)

python-Levenshtein est une extension C pour calculer la distance de Levenshtein entre deux chaînes. Elle nous a permis de construire notre fichier contenant, pour chaque mot du dictionnaire, ses 10 plus semblables.

gTTS (Version 2.5.1)

gTTS (Google Text-to-Speech) est une interface Python pour le service de synthèse vocale de Google. Elle nous permet dans différents jeux, de générer des audios contenant des phrases ou des mots.

Conclusion 🔚

Dans cette documentation technique, nous avons parcouru avec minutie chaque aspect de notre projet, de son architecture à ses composants logiciels, en passant par ses fonctionnalités essentielles. C'est le résultat de centaines d'heures de travail acharné, investies par nous deux, et nous sommes fiers de ce que nous avons accompli ensemble.

Nous avons consacré un temps considérable pour parvenir à ce résultat. Cet engagement envers l'excellence reflète notre détermination à créer une application robuste, fiable et innovante. Nous sommes particulièrement fiers d'avoir développé un projet qui non seulement répond à un besoin spécifique, mais qui contribue également à promouvoir l'inclusion et à faciliter l'apprentissage de l'anglais, une compétence cruciale dans la société moderne.

Ce projet dépasse les simples lignes de code et les fonctionnalités techniques. Il incarne notre vision commune de créer quelque chose d'utile, de significatif et de durable.

Alors que nous terminons cette documentation, nous regardons vers l'avenir avec optimisme et détermination. Nous sommes impatients de voir comment notre application évoluera et continuera à avoir un impact positif sur la vie de ses utilisateurs.

Quelques infos importantes 👊

Pour ce projet, nous avons délibérément choisi de ne pas recourir à des bibliothèques tierces en JavaScript ou en CSS. Tous les codes CSS, HTML, JavaScript et Python ont été entièrement développés par notre équipe. Bien que nous ayons consulté un tutoriel pour le jeu du snake (Tutoriel de la chaîne AKDEV), le code final est significativement différent de sa version initiale, et l'accent a été mis sur la valeur ajoutée à travers nos propres modifications et améliorations.

De plus, les fichiers contenant les mots les plus similaires ainsi que le fichier texte contenant les termes censurés pour notre système anti-profanité ont été créés en interne. Bien entendu, quelques bibliothèques Python ont été utilisées, mais leur utilisation était nécessaire pour faciliter le développement sans compromettre l'intégrité du projet.

En somme, notre volonté première était véritablement de créer quelque chose par nous-mêmes, et le simple copier-coller de code n'était pas une option pour nous.

Quelques statistiques 📊

Le projet a nécessité un travail intensif, comprenant environ 30 000 lignes de code réparties comme suit : 27,2% en CSS, 18% en HTML, 20,2% en JavaScript et 34,6% en Python. Durant une période de six mois, nous avons consacré plus de 500 heures au développement, effectuant plus de 200 commits sur GitHub pour assurer un suivi continu du projet.

Pendant la période de tests, qui s'étend du 25 février au 26 mars 2024, notre application a enregistré plus de 100 000 requêtes, incluant des demandes de pages HTML, des appels serveur, des chargements de logos, d'images et de fichiers annexes. Nous avons accueilli 28 utilisateurs différents, lesquels ont contribué à la création de 75 listes de vocabulaires. Ces utilisateurs ont cumulé près de 12 382 XP au total à ce jour, ce qui témoigne d'un fort engouement pour notre application. Les retours des utilisateurs ont été très positifs, démontrant un intérêt significatif pour notre projet.

Cette ouverture à la communauté a permis la correction de nombreux bugs et a également inspiré la mise en place de nouvelles fonctionnalités telles que les badges d'utilisateurs et des modifications dans le système de classement, renforçant ainsi l'attrait et l'utilité de notre application.

Créé avec ❤️ par Abel et Hippolyte à Colmar