Projet Langages Web M1

Théo SALLES, LEITAO MOREIRA Thomas

Janvier 2025

UFR Sciences et Techniques

Université de Rouen

Année 2024-2025

Contents

1 Introduction

L'objectif de ce projet est de réaliser un site proposant à un utilisateur de résoudre des grilles de mots croisés proposés. Le site doit proposer un système de compte, permettant ainsi à un utilisateur de sauvegarder les grilles auxquels il a joué dans le passé, et créer des grilles qui pourront être joués par d'autres utilisateurs.

Ce projet a été réalisé sans cadriciels, en n'utilisant que des technologies de base, à savoir HTML, CSS, JavaScript et PHP. Une revue détaillée des technologies sera disponible plus tard dans ce document

2 Installation

Cette section sert avant tout de préambule, et de guide de secours au cas où le script automatisée d'installation ne fonctionne pas, par exemple si jamais l'environnement d'exécution ne contient pas les programmes requis (ex° : `sudo`, ...).

Tout d'abord, l'environnement doit contenir les logiciels suivants. Nous assumons que nous utilisons une base Ubuntu (ie. Debian) de version minimum 22.04 :

- LAMP (Guide d'installation)
- MySQL (ou MariaDB)

Une fois ces paquets installés, le script d'installation, présent à la racine du projet, peut être lancé. **Attention : le script doit être exécuté par l'administrateur. Dans le cas contraire, le script affichera un message d'erreur sur la sortie et s'arrêtera immédiatement, sans aucune autre modification.**

```
chmod +x "./script.sh"
sudo "./script.sh"
# ou
sudo bash "./script.sh"
# ou
su root -c "./script.sh"
```

Au cas où ce script viendrait à planter, où à envoyer une erreur, nous devons alors réaliser ces étapes manuellement.

Pour que ce projet fonctionne, il faut qu'il se trouve à l'emplacement `http://[root]/cwJS/` (ici, `root` désigne `localhost` ou bien l'adresse URL ou IP de la machine sur lequel est déployé le projet. Pour cela, en admettant que nous utilisons Apache, le projet doit être déplacé dans `/var/www/html/cwJS` (la casse doit être respecté).

```
mkdir -p /var/www/html/cwJS

mv ./* /var/www/html/cwJS
# ou
cp ./* /var/www/html/cwJS
```

Ensuite, nous devons initialiser la base de données utilisée par le projet. Un script SQL est disponible dans le dossier `./_setup/setup.sql`. En admettant que MySQL soit installé et configuré au moins pour fonctionner avec l'utilisateur `root`, vous pourrez renseigner la commande suivante :

```
sudo mysql < "./_setup/setup.sql"
```

Il est à noter que ce script crée :

- un utilisateur `cwjs`, ayant tous les droits sur...
- une base de données `CWJS` contentant...
- deux tables, `USERS` et `GRIDS`
- de plus, une entrée sera ajoutée à `USERS`, il s'agit de l'administrateur du projet.

Il est à noter que, si pour une quelconque raison, vous souhaitez vous authentifier en tant que cet utilisateur `cwjs`, le mot de passe est disponible dans le script.

Le projet devrait dés maintenant être opérationnel. Pour tester son bon fonctionnement, il suffira d'aller sur `http://localhost/cwJS`. La page d'accueil devrait s'afficher. Dans le cas contraire, si une **erreur 500** s'affiche, cela signifie qu'il s'est passé quelque chose durant la configuration de la base de données. Dans ce cas, veuillez renseigner ces commandes dans l'invite MySQL (en tant que super utilisateur), puis retentez d'exécuter le script.

```
DROP DATABASE CWJS;
DROP USER cwjs;
EXIT;
```

3 Visite

Avant de détailler la structure du site et sa programmation, nous vous proposons de faire une visite guidée en tant que :

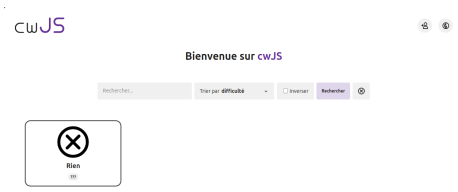

Figure 1: Page d'accueil

3.1 En tant qu'utilisateur sans compte

En entrant l'adresse, nous nous retrouvons sur la page d'accueil. Elle se compose :

- une barre d'en-tête - proposant de se connecter (donc menant vers la page d'inscription/connexion, voir ci-dessous), et de changer de thème -,
- une barre de recherche permettant de rechercher des grilles selon leur nom, puis de les trier selon leur dimension, difficulté ou date d'ajout,
- une liste de toutes les grilles enregistrées dans la base de données. Chaque grille dispose de plusieurs informations sur la grille, comme son nom, sa dimension, le nombre de mots, et sa difficulté. Un bouton "Jouer" permettra à l'utilisateur de jouer à cette grille en particulier.

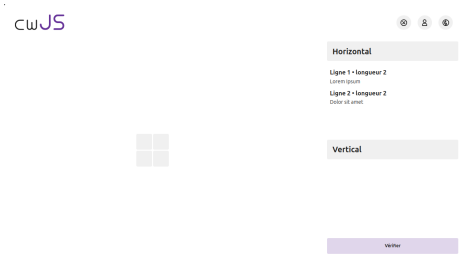

Figure 2: Page de jeu

Une fois sur la page de jeu, l'utilisateur fera face à 3 éléments :

- La grille de jeu, composée de cellules vides ou grisées (donc ne contenant aucune lettre). Cliquer sur une cellule mettra en surbrillance toutes les cellules sur la même colonne et ligne.
- La liste des mots, divisés en deux catégories : horizontal et vertical. Chaque mot possède une longueur, une ligne de départ, et une définition. Cliquer sur le mot mettra en surbrillance les cellules devant être remplies.

- Un bouton de vérification, mettant en surbrillance les cellules contenant la bonne lettre, faisant gagner la partie à l'utilisateur si toutes les cellules remplissables contiennent la même lettre.

Figure 3: Écran de victoire

Une fois la partie finie, l'utilisateur se retrouve sur la page de victoire, lui permettant, entre autres, de rejouer la grille, de revenir à la page d'accueil ou, si l'utilisateur est connecté, de revenir à sa page "Compte".

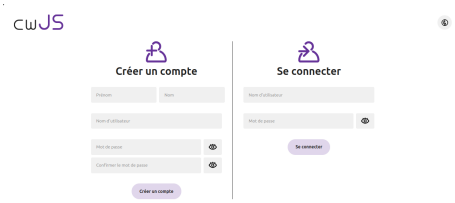

Figure 4: Page de connexion

Si le joueur veut créer un compte ou se connecter, il devra se rendre sur la page de connexion. Ici, deux formulaires sont disponibles, un permettant de créer un compte, un autre permettant de se connecter. Il est à noter que chacun de ces formulaires doivent être remplies en suivant des conditions précises. Par exemple, le nom d'utilisateur doit contenir au moins 5 caractères, ou encore le mot de passe doit suivre les recommandations de base sur les mots de passe (plus de 8 caractères, lettres majuscules, minuscules, chiffres et caractères spéciaux).

Si une erreur se produit durant la création ou la connexion au compte, celui-ci sera re-déplacé sur la page de connexion, avec une erreur en bas à gauche.

3.2 En tant qu'utilisateur connecté

Une fois l'utilisateur connecté, il sera déplacé sur sa page de compte. Celle-ci lui permettra de faire les actions suivantes :

Figure 5: Page de compte

- Créer une nouvelle grille, avec le bouton dans l'en-tête,
- Accéder à ses grilles en cours de jeu, a.k.a les grilles que l'utilisateur a quitté avant d'avoir atteint la page de victoire. Il est à noter que cette fonctionnalité est permise par l'utilisation de cookies (voir plus bas),
- Accéder aux grilles que l'utilisateur a créé, les jouer, les modifier et les supprimer à l'aide des boutons associés,
- Modifier les informations liés au compte (nom/prénom, nom d'utilisateur, mot de passe)

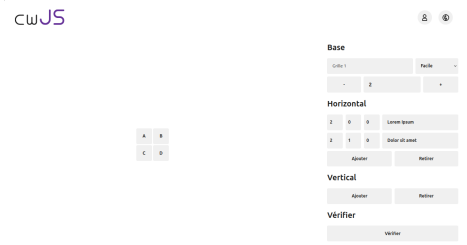

Figure 6: Page de création de grille

Lors de la création ou édition d'une grille, l'utilisateur se retrouvera face à une interface similaire à l'interface de jeu, à savoir avec une grille de jeu sur la gauche, et une liste de mots sur la droite. Plusieurs différences sont évidemment à remarquer. Par exemple, l'utilisateur pourra renseigner :

- Le nom de la grille, ainsi que sa difficulté, parmi facile, moyen et difficile.
- La dimension de la grille, que l'utilisateur pourra faire changer de 1 dimension. Deux remarques sont à faire :
  - La grille ne peut avoir une dimension inférieure à 1. Il n'y a pas de limite maximale.

  - La dimension de la grille sera toujours égale en longueur et en largeur. Un grille sera toujours en 1x1, 2x2, 3x3...

- Les mots associés à la grille, que l'utilisateur pourra ajouter avec les boutons associés dans chaque catégorie. Dés la création d'un mot (que l'on nommera "entrée" à partir de maintenant) avec le bouton "Ajouter", l'utilisateur pourra renseigner la longueur du mot, ses coordonnées de départ (commençant par 0, donc un mot commençant à la ligne 1, colonne 1 aura comme coordonnées $(0, 0)$), ainsi que sa définition. Appuyer sur le bouton "Retirer" retirera la dernière entrée.

- La grille de jeu. L'utilisateur devra donc remplir la grille comme pourra faire les joueurs pendant leur partie. Les cellules laissées vides seront considérées comme grisées pendant le jeu.

Lorsque l'utilisateur aura fini, il pourra cliquer sur le bouton vérifier, situé en bas à droite. En cas d'erreur, un message s'affichera à côté du bouton, annonçant la raison de l'erreur. Cela peut être causé par :

- Un mot dépassant la grille
- Un mot avec des lettres manquantes
- Un mot avec une taille invalide (≤ 0)
- Un mot avec une définition vide

Si aucune erreur est détectée, la grille sera enregistrée, et l'utilisateur sera immédiatement déplacé vers la page de jeu pour tester sa grille.

À part cela, les capacités de l'utilisateur resteront les mêmes qu'il soit connecté ou non, donc ayant toujours la capacité de jouer aux grille, avec la différence que l'utilisateur non connecté ne pourra pas accéder à une liste de grilles en cours.

3.3 En tant qu'administrateur

Par défaut, le projet ne dispose que d'un seul compte administrateur, son nom d'utilisateur et mot de passe étant défini à l'avance. Celui-ci pourra s'identifier comme n'importe quel utilisateur, en passant par la page de connexion.

Il est à noter que les identifiants sont disponibles dans le fichier `setup.sql`, en tant que commentaire. Pour des raisons de sécurité, ces informations ne seront pas répétés ici.

L'administrateur n'a pas la capacité de créer des grilles, toute tentative d'aller sur la page de création de grilles le renverra vers sa page de compte.

Sur cette page, toutes liste de grilles (donc "Grilles en cours" et "Vos grilles") seront vides. Par contre, une liste d'utilisateurs sera disponible, permettant à

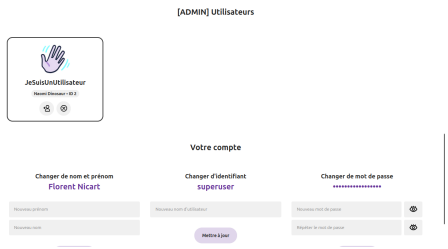

Figure 7: Vue de la page compte en tant qu'administrateur

l'utilisateur de se connecter en tant qu'un utilisateur spécifique sans avoir besoin d'utiliser un mot de passe (un mot sur les risques de confidentialité qu'ajoute une telle fonctionnalité sera ajouté dans la section **Améliorations** de ce document), ou de le supprimer.

Hormis cela, l'administrateur pourra toujours jouer aux grilles des autres joueurs, avec des fonctionnalités similaires aux joueurs non connectés.

4 Structure et programmation

Ce projet se structure en différentes parties, codées en utilisant différents langages de programmation. Dans cette partie, nous verrons à quoi servent ces parties, les raisons derrière leur programmation, ainsi que certaines notes et objections sur certaines technologies utilisées (le peuple demande des cadriciels ¿:( ) .

Pour rappel, le projet possède l'arborescence suivante (en partant du principe que celui-ci soit récupéré directement depuis un dépôt Git :

```
    .
>-- account
    >- index.php
>-- congrats
    >- index.php
>-- create
    >- index.php
>-- _credentials.php
>-- index.php
>-- LICENSE.md
>-- login
    >- index.php
>-- play
    >- index.php
>-- _public
```

```
    >-- code
        >-- create.js
        >-- ...
    >-- resources
        >-- icons
            >- ...
        >- images
            >-- ...
    >- style
        >-- main.css
        >- src
            >-- ...
>-- README.md
>-- _server
    >-- components
        >-- Button.php
        >-- Component.php
        >-- ...
    >-- data
        >-- AccessHandler.php
        >-- CookieHandler.php
        >-- DatabaseHandler.php
        >- patterns.php
    >- handlers
        >-- account_edit.php
        >-- ...
>-- _setup
    >- setup.sql
>- setup.sh
```

Une dernière remarque est à destination du fichier `_credentials.php`, contenant les identifiants pour la base de données. Celui-ci ne doit **SURTOUT PAS** être modifié, sauf si la configuration de la base de données préalablement faite ne se soit pas passé comme prévu, forçant l'utilisation d'identifiants différents.

4.1 Pages utilisateur

Tout dossier situé dans la racine du projet ne commençant pas par un "." ou un "_" est un dossier accessible par l'utilisateur, en admettant certaines conditions variant selon les pages (par exemple, la page de compte n'est disponible seulement si l'utilisateur est connecté).

4.2 Le dossier serveur

Le dossier "`_server`" contient tous les composants utiles au serveur, incluant par exemple les scripts permettant la gestion de la base de données, la gestion

d'accès aux utilisateurs, ... Ces fichiers sont donc utilisés durant la création de la page sur le serveur, avant l'envoi des données à l'utilisateur. Le projet suit donc un principe SSR (Server Side Rendering).

Il paraîtra donc évident que ces scripts soient réalisés en PHP.

Celui-ci ne devrait pas être accédé par l'utilisateur, bien que, la plupart du temps, ces fichiers une fois demandés par l'utilisateur, renverront une page vide, la plupart de ces fichiers étant des définitions de classe.

4.2.1 Components

Le dossier "`components`" contient l'intégralité des "composants", à savoir des objets voulant représenter des *snippets* de code HTML, dont leur apparence et contenu peuvent être décidé dés la création de l'objet, avec le constructeur.

La volonté derrière cette implémentation vient d'une volonté de factoriser le code, permettant d'éviter des répétitions, et/ou des "*echos*" sauvages de code HTML en plein milieu d'un script. Nous pouvons aussi ajouter à cela une volonté de "copier" le système de composants vus dans la plupart des cadriciels JavaScript.

Chacun de ces fichiers contiennent une unique classe, toutes implémentant la classe abstraite `Component`, contenue dans le fichier `Component.php`. Ces classes disposeront donc toutes de deux fonctions publiques, une permettant de récupérer le code HTML associé, nommée `getCode()`, et une permettant d'afficher/"render" ledit code avec un `echo`, `render()`.

4.2.2 Data

Le dossier (très mal nommé) "`data`" contient une série de services et fonctions/variables utiles au fonctionnement de ce projet. Nous pouvons citer :

- `AccessHandler`, fournissant une classe permettant de vérifier si un utilisateur est connecté, le rediriger sinon avec `check()`, si un utilisateur est un administrateur, ... Aussi inclus est une fonction permettant de rediriger l'utilisateur vers une page et de stopper le script si une condition est vraie.
- `CookieHandler`, permettant d'interagir avec les cookies enregistrés par le biais de PHP (donc non accessible avec JS). Ce module est surtout utilisé pour l'enregistrement des parties non finies/en cours.
- `DatabaseHandler`, permettant un raccourci pour l'initialisation de l'interface vers la base de données PDO. Il est à noter que de part le fonctionnement de PHP vis-à-vis des imports, chaque page utilisant la base de données devra importer le fichier contenant les identifiants de manière différente.

Chaque module est programmé sous forme de classe, hormis certaines fonctions servant principalement à factoriser du code réutilisé partout dans le projet.

4.2.3 Handlers

Le dossier "`handlers`" contient tous les scripts devant réaliser une action pour l'utilisateur, comme la connexion, la création de compte, la création et/ou modification d'une grille, ...

- `account_edit` (**POST**) : permet de mettre à jour les informations d'un utilisateur.
  - **fn** *str* : le nouveau prénom de l'utilisateur
  - **ln** *str* : le nouveau nom de l'utilisateur
  - **id** *str* : le nouveau nom d'utilisateur
  - **pwd** *str* : le nouveau mot de passe de l'utilisateur
  - **pwd2** *str* : copie du mot de passe, pour vérification
- `delgrid` **GET** : permet la suppression d'une grille. L'utilisateur doit être l'auteur de la grille pour la supprimer.
  - **id** *int* : l'ID de la grille
- `deluser` **GET** : permet de supprimer un utilisateur. L'utilisateur doit être un administrateur.
  - **id** *int* : l'ID de l'utilisateur à supprimer
- `grid` **POST** : permet l'ajout ou la modification d'une grille. L'utilisateur doit être connecté, et doit être l'auteur de la grille si il s'agit d'une modification.
  - **id** (optionnel) *int* : l'ID de la grille si elle existe
  - **grid** *str* : le contenu de la grille au format JSON.
  - **words** *str* : la liste de mots au format JSON.
  - **title** *str* : le titre de la grille
  - **dim** *int* : la dimension de la grille
  - **diff** *int* : la difficulté de la grille
  - **count** *int* : le nombre de mots dans la grille
- `logas` **GET** : permet de s'identifier en tant qu'un utilisateur précis sans mot de passe. L'utilisateur doit être administrateur.
  - **id** *int* : l'ID de l'utilisateur
- `login` **POST** : permet de se connecter.
  - **username** *str* : le nom d'utilisateur
  - **passwd** *str* : le mot de passe (en clair, on va revenir sur cela)

- `logout` **GET** : permet de se déconnecter. Aucune condition n'est requise pour accéder à cette page, celle-ci supprimant toute information sur l'utilisateur dans la session courante, hormis les cookies.
- `signup` **POST** : permet de créer un compte.
  - **fn** *str* : le prénom du nouvel utilisateur
  - **ln** *str* : le nom du nouvel utilisateur
  - **username** *str* : le nom d'utilisateur
  - **passwd1** *str* : le mot de passe du nouvel utilisateur
  - **passwd2** *str* : copie du mot de passe, pour vérification

Ce dossier est ce qui se rapproche le plus d'une API. En effet, il est facile de remarquer qu'aucune requête n'est faite à quelconque *endpoint* dans ce projet. La raison de l'utilisation de cette méthode relativement datée est liée à comment sont gérés les sessions en PHP. Pour s'assurer de la présence d'une session et d'accéder aux informations associés sans avoir à envoyer le token d'identification à chaque requête, et plus simplement pour sa facilité d'utilisation, nous avons préféré utiliser des *handlers* auxquels l'utilisateur devra accéder afin de réaliser certaines actions. Un autre avantage réside aussi dans la gestion des erreurs, nous permettant ainsi de directement rediriger l'utilisateur en cas d'erreur ou de réussite.

4.3 Le dossier public

Le dossier "`_public`" contient tous les fichiers utiles à la navigation de l'utilisateur, à savoir les images et icônes, les scripts JavaScript et les fichiers CSS. Ceux-ci sont donc directement accédés par le navigateur lors du chargement d'une page.

4.3.1 CSS

Nous passerons très rapidement sur cette partie, mais nous devons faire une remarque sur la technologie utilisée ici. Pour simplifier la réalisation du fichier CSS et permettre une meilleure relecture, nous avons utilisé le pré-compilateur **Sass**. Il ne s'agit pas d'un cadriciel en lui-même, son objectif étant de "compiler" les fichiers en format `.sass` situés dans `.../style/src` en un seul et unique fichier `main.css`, accessible depuis le dossier `.../style/`. Cette compilation nous a été permise grâce au compilateur intégré aux IDE JetBrains (ici **PhpStorm**), nous évitant d'avoir à configurer un projet `npm` pour utiliser `sass-dart`.

Nous avons préféré utiliser Sass, une de ses fonctions étant la possibilité de "nicher" des classes CSS dans d'autres, nous permettant de passer d'un code similaire à :

```
a {
    //...
}

a > b {
    //...
}

a > b c {
    //...
}
```

... à ...

```
a
    //...
    > b
        //...
        c
            //...
```

4.3.2 Scripts JavaScript

Tout script JavaScript se trouve dans le dossier `_public/scripts/`. Dans ce projet, nous avons principalement utilisé JavaScript pour trois rasions, ayant pour point commun la nécessité d'interagir avec un élément sans avoir à interroger le serveur en permanence :

- L'interaction avec la grille de jeu et la liste de mots pendant une partie, que ce soit pour identifier des mots, remplir la grille, ou encore vérifier sa réponse.
- La création d'une grille, avec la modification dynamique de la grille, la création dynamique d'entrée dans la liste, et la vérification de la validité de la grille.
- La gestion du schéma de couleur de la page.

L'importation des scripts se fait à l'aide du composant PHP `Header`, permettant de renseigner les scripts à importer, leur emplacement changeant selon la "profondeur" à laquelle l'utilisateur se trouve durant la navigation du site.

Une dernière remarque à faire réside dans l'un des grands désavantages de JavaScript, à savoir son typage dynamique essentiellement basée sur le "feeling", pouvant créer des situations où une erreur se déclenche trop tard car un objet passé de manière incorrecte en paramètre ne possède pas un attribut recherché. Pour palier à cela, plusieurs solutions sont disponibles, la première étant TypeScript.

Pour rappel, TypeScript est un sur-ensemble syntaxique permettant de typer plus durement les fonctions et variables, tout en gardant un typage dynamique, par exemple lors de l'initialisation d'une constante. Toutefois, bien que ce ne soit par un cadriciel, et que le produit fini sera toujours en JavaScript, TS requiert l'utilisation de `npm`, alourdissant le projet. De plus, la "compilation" vers un unique fichier JavaScript est compliqué (les sur-compilateurs comme `gulp` ou `webpack` produisant des résultats non concluants), et peut mener à des incompatibilités, notamment lors de l'utilisation de la syntaxe ECMA $\geq$ 2016.

Une autre solution possible est JSDoc, permettant d'indiquer à l'IDE - et donc n'ayant aucun impact sur l'exécution - les types de certaines variables, permettant d'utiliser l'IntelliSense, et de notifier au codeur les possibles erreurs liés aux types. Il s'agira donc de la solution utilisée, étant un outil fourni avec tout IDE récent, et ne nécessitant pas d'ajouter de paquets et/ou programmes supplémentaires.

Interactivité pendant la partie

Pour permettre l'interactivité lors d'une partie, le script du nom de `grid.js` propose 3 classes principales servant de modèles :

- `Grid` : il s'agit du modèle permettant de représenter la grille de jeu. Elle peut être initialisée avec lune matrice carré de caractères, lui permettant de remplir la grille aux bonnes dimensions, et de placer les cellules grisées au bon endroit. Cette classe sert principalement à contenir le modèle associé à la grille de jeu, à savoir une matrice d'objets `Cells`. Elle permet aussi d'intéragir avec la grille, en surlignant certaines cellules à l'aide des méthodes `selectCell`, `selectRow` et `unselect`.
- `WordSet` : il s'agit du modèle permettant de représenter les mots à renseigner dans la grille, divisée en deux sections (horizontal et vertical). Celle-ci est initialisée à l'aide d'une liste de mots. L'objectif de cette grille est en priorité d'indiquer à l'utilisateur, outre l'affichage des mots, l'emplacement de ceux-ci, en rendant la liste interactive et en permettant de surligner certaines parties de la grille lors du clic sur une définition.

- `Solver` : il s'agit de la classe qui va permettre d'indiquer à l'utilisateur si la grille a été résolue et est correcte. Celle-ci s'initialise avec l'objet contenant une instance de modèle de grille préalablement initialisé, ainsi qu'une matrice de caractères représentant la grille finie. Cette classe, une fois initialisée, permettra à l'utilisateur de vérifier de manière dynamique et interactive, de vérifier sa grille, dés la pression d'un bouton, qui exécutera la méthode `check()`.

Il est à noter que ces classes n'ont pour objectif que de permettre à l'utilisateur d'interagir avec la partie, et ne permet pas en lui-même de créer dynamiquement des éléments HTML. En effet, ceci est permis grâce aux classes suivantes :

- `Cell` : représente une case/cellule dans la grille, et se veut être le code sous-jacent de chaque élément `<input>` composant la grille dans le fichier HTML.

- `Word` : représente une entrée pour un mot, il se veut être le code sous-jacent de chaque élément `<div>` contenant les informations sur chaque mot composant la liste de mots dans le fichier HTML.

Il s'agit de ces éléments en particulier qui permettent en réalité d'interagir avec la partie, les classes citées plus haut servant au mieux d'*initialiseurs* et/ou de conteneurs, en leur qualité de modèle.

Il est aussi à noter que `grid.js` mets à disposition une variable globale du nom d'`overseer`. Celui-ci permet de rendre la grille accessible à toutes les instances de `WordSet`, `Cell` et `Word`, permettant entre autres de faciliter la programmation de fonctions lié à l'intéraction entre la grille et la liste de mots. Bien que l'utilisation de variables globales soit controversé et peut mener à des situations où celle-ci peut se retrouver inutilisable, la relative simplicité des scripts n'apporte donc en soit aucun risque trop grave.

Interactivité pendant la création d'une grille

Pour permettre l'interactivité lors d'une partie, le script du nom de `create.js` propose trois classes supplémentaires. Il doit être notifié que ce script nécessite le script mentionné précédemment pour ne pas obtenir d'erreurs, la raison pourquoi deviendra évidente.

- `EditableGrid` : Il s'agit d'une classe inhéritant de `Grid`. Celui-ci propose deux fonctionnalités en plus :
  - Le changement de sa taille, ajoutant et supprimant des objets `Cell` de manière dynamique, de par la méthode `resizeGrid()`
  - Son "export" sous forme de matrice de caractères, utile pour son stockage dans la base de données.

- `EditableWordSet` : Il s'agit d'une classe indépendante de `WordSet`, permettant la création dynamique d'entrées que l'utilisateur pourra modifier, créer et supprimer à souhait. Tout comme `EditableGrid`, sa fonctionnalité principale est d'exporter la liste ainsi remplie sous forme d'une liste de mots.

- `Verificator` : Cette classe prends en paramètre des instances des deux classes ci-dessus, et permet de vérifier que la grille créée par le biais de ces classes est valide, envoie une requête au serveur par le biais d'un formulaire caché et informant l'utilisateur sinon.

Scripts complémentaires
Les scripts `login.js` et `theme.js` proposent respectivement :

- la fonction permettant d'afficher le mot de passe en clair après avoir cliqué sur le bouton d'affichage, fonctionnalité permise par le composant PHP `Field`

- la classe permettant la gestion du schéma de couleur, ou "thème", de la page, permettant le stockage de l'information dans le stockage local du navigateur de l'utilisateur. Le changement de thème est permis à l'aide du champ `dataset` de la racine du document (`<body>`), et grâce à certaines règles CSS.

4.4 Un mot sur le motif MVC

Pour rappel, le motif MVC se caractérise très très simplement de la manière suivante :

> Toute **vue** accessible à l'utilisateur dialogue avec son **modèle** associé par le biais de **controlleurs**.

En pratique, cela se manifeste dans notre code de la manière suivante :

- Chaque élément située dans notre **vue**, préalablement modifiée par l'exécution d'un script PHP, est accessible depuis le document HTML et modifiable dynamiquement de par l'utilisation d'`id` uniques.

- Lors de l'initialisation de chaque **modèle**, de par les `id` mentionnés plus haut, le constructeur "lie" son **modèle** à son élément dans la **vue**.

- De plus, le constructeur ajoute aussi les **controlleurs** nécessaires au fonctionnement de l'application, et à la communication entre la **vue** et leurs **modèles**. Cela peut être visible dans le code JS par la redéfinition des propriété `onclick`, `onfocus` et `onblur`, permettant l'ajout de fonctions *callback* aux **écouteurs d'événements** des éléments de la vue.

5 Base de données et Sécurité

Chaque interaction avec la base de données se fait au travers d'une interface PDO, permettant entre autres d'éviter toute attaque de type "injection SQL".

Une dernière remarque sur la sécurité peut se faire vis-à-vis des *handlers*. En effet, de part l'utilisation des `AccessHandler` et des `manageRedirect`, redirigeant l'utilisateur le plus vite possible en cas d'accès non autorisé, ou d'erreur, que ce soit avec les données de l'utilisateur ou avec la base de données.

De plus, lors de la création de compte et/ou de la connexion avec les *handlers* `login.php` et `signup.php`, le renouvellement du token de session est effectué pour éviter toute conséquence vis-à-vis d'un possible vol de token.

Lors de l'initialisation de la base de données, le script SQL fait les actions suivantes :

- Création d'un utilisateur indépendant, permettant une gestion précise des autorisations
- Une base de données indépendante, sur lequel seul l'utilisateur créé et mentionné ci-dessus peut avoir accès.
- Deux tables.

Ces deux tables sont les suivantes :

5.1 USERS

La table `USERS` permet de stocker les informations en lien avec les utilisateurs. Il est initialisé avec un seul utilisateur, le seul compte administrateur renseigné par défaut.

Dans cette table, deux attributs sont à remarquer :

- L'attribut `pwd_hash` contient non pas le mot de passe en clair de l'utilisateur, mais un hachage dudit mot de passe, passé à travers l'algorithme de hachage par défaut de PHP. Cela évite toute fuite de donnée si la base de données se retrouve compromise
- L'attribut `role` contient le "rôle" de l'utilisateur, il s'agit d'un nombre positif. "0" signifie que l'utilisateur est un utilisateur lambda, "1" signifie qu'il s'agit d'un administrateur. Il est à noter qu'actuellement, il n'existe qu'un seul administrateur, et il est impossible d'en créer un autre directement depuis les pages du projet.

5.2 GRIDS

La table `GRIDS` permet de stocker les grilles produites par les utilisateurs. À l'initialisation, il est vide.

Dans cette table, trois attributs sont à remarquer :

- L'attribut `author` est une clé étrangère, en lien avec la table `USERS`. Cela signifie qu'il y a une condition permettant d'assurer qu'un utilisateur doit exister avant de se dirte "auteur" d'une grille.
- Les attributs `grid` et `words` sont au format `TEXT`, il est à noter qu'ils servent à stocker respectivement la matrice de caractères et la liste de mots préalablement "exportés" des instances de `EditableGrid` et `EditableWordSet`, comme vu ci-dessus, pendant la création d'une grille. Ces deux données sont stockés au format JSON. De part le support plus que douteux des données JSON avec le type `JSON` par MySQL, nous avons préféré utiliser le type `TEXT`.

6 Améliorations possibles

De par le court délai laissé pour la réalisation de ce projet et la charge de travail en parallèle, plusieurs améliorations sont possibles pour ce projet :

- De part le fonctionnement de PHP et son import basé sur le chemin d'accès aux scripts, il a été impossible de créer un système de modules similaire à JS sans l'utilisation d'un cadriciel PHP, nous forçant donc à importer sans arrêt des scripts avec `include_once`, chose surtout remarquable avec l'utilisation de `DatabaseHandler`, nécessitant l'import constant de `_Credentials.php`. Nous avons tout de même tenté de palier à cela avec la fonction `spl_autoload_register()`.
- L'initialisation des grilles et listes de mots lors d'une partie et/ou de l'édition d'une grille se fait en injectant directement le code au format JSON dans les constructeurs des classes. Bien qu'une vérification sommaire soit faite par les constructeurs, ce système espère avoir des données valides de la part de la base de données. Or...
- La vérification des données de la grille n'est pas effectuée de nouveau par le *handler* `grid.php`, ce qui peut entraîner l'ajout de données incorrectes à la table `GRIDS` par un utilisateur mal intentionné. Bien que cela ne pose pas beaucoup de préoccupations d'un point de vue sécurité, cela permettrait quand même de casser le site.
- La création d'une grille se fait de manière peu ergonomique et prône aux erreurs. Une approche plus intéressante serait de rendre la grille non interactive, l'intégralité de l'édition de celle-ci se faisant depuis la liste de mots.

- L'interface manque de clarté et d'information. Cela peut être surtout remarqué avec les boutons "Vérifier", ne précisant pas ce que cliquer dessus va faire, ou encore avec les conditions requises lors de la création d'un utilisateur et d'un mot de passe. En effet, les données doivent être validés par un *RegEx*, mais les conditions précises ne sont pas indiqués

- L'utilisation de *handlers* en lieu et place d'API rend l'application fermé (pas d'*endpoint* à destination de tiers) , et se révèle être une approche relativement datée de l'interaction avec une base de données. De plus l'entièreté des communication avec ces *handlers*, surtout lorsque ceux-ci attendent des requêtes `POST`, se font à travers des formulaires, sans utilisation de requêtes donc.

- La possibilité pour l'administrateur de se connecter en tant que n'importe quel utilisateur avec le *handler* `logas.php` est beaucoup trop dangereux et pourrait être bridé, en permettant à l'administrateur de directement modifier les informations liés à un utilisateur directement depuis une page d'administration sans avoir à usurper son identité par exemple.

7 Conclusion et auteurs

En conclusion, ce projet nous a permis de (re)découvrir toutes les technologies entourant la création d'une application Web, de se familiariser avec certains principes (comme MVC ou encore SSR), et de tester nos capacités sur des sujets sensibles, comme les choix d'implémentation ou encore la sécurité des données utilisateur. Enfin, ce projet nous a permis de découvrir de nouvelles pratiques (PDO, gestion de session, requêtes AJAX), et des technologies plus complexes (Sass, TypeScript, JSDoc, ...).

Projet réalisé par :

- **SALLES Théo**
- **LEITAO MOREIRA Thomas**

8 Annexe

Figure 8: Entendez nos pleurs