Motle

« Une adaptation française du jeu Wordle, où les joueurs devinent un mot de 5 lettres en 6 tentatives. »

Par Vasco et Guillaume – En 2024 – Projet de Formation

Table des matières

1. Introduction

Motle est une application web interactive inspirée du célèbre jeu Wordle. L'objectif pour le joueur est de deviner un mot français de cinq lettres, généré aléatoirement par le système, en un maximum de six tentatives. Après chaque tentative, le jeu fournit des indices visuels sous forme de couleurs : les lettres correctement placées apparaissent en vert, les lettres présentes dans le mot mais mal placées en jaune, et les lettres absentes du mot en gris.

2. Objectifs du Projet

2.1 Contexte de Développement

Ce projet a été conçu et réalisé comme un exercice pratique durant la formation BTS SIO SLAM. Nous avons eu l'idée de développer une adaptation française du jeu populaire Wordle, pour appliquer et consolider nos compétences en développement web full-stack, notamment avec Node.js, Express.js et JavaScript vanilla pour la logique côté client.

2.2 Objectifs Principaux

- **Pédagogiques :** Maîtriser les fondamentaux du développement web avec Node.js et Express.js, ainsi que la manipulation du DOM avec JavaScript ;
- **Fonctionnels :** Créer un jeu de devinettes de mots qui soit à la fois interactif, intuitif et engageant pour l'utilisateur ;
- **Techniques :** Implémenter une logique de jeu robuste, incluant la validation des mots côté client par rapport à un dictionnaire défini ;
- **UX/UI (Expérience Utilisateur/Interface Utilisateur) :** Développer une interface utilisateur moderne, esthétique, responsive et agréable à utiliser, avec des animations et des retours visuels clairs.

3. Architecture de l'Application

3.1. Style Architectural et Structure

Le projet Motle adopte une **architecture Client-Serveur** avec une approche inspirée du pattern **MVC (Modèle-Vue-Contrôleur) simplifié** :

- **Serveur (Backend) :** Express.js est utilisé pour servir les fichiers statiques (HTML, CSS, JavaScript client, assets) et pour rendre les vues via le moteur de template EJS. Il gère également la sélection initiale du mot à deviner ;
- **Client (Frontend) :** JavaScript vanilla est employé pour toute la logique de jeu interactive exécutée dans le navigateur de l'utilisateur (gestion des saisies, validation des tentatives, mise à jour de l'interface) ;
- **Données (Modèle) :** Un fichier JSON statique (mots.json, récupéré avec le scraping) sert de dictionnaire de mots français. L'état du jeu (mot en cours, tentatives, etc.) est géré côté client pour la simplicité.

3.2. Fichiers Essentiels et Rôles

- **Points d'Entrée :**

    - index.js (à la racine) : Script principal du serveur Express.js, responsable de la configuration du serveur, du routage et du rendu de la page initiale ;
    - public/scripts/index.js : Contient toute la logique JavaScript exécutée côté client pour gérer les interactions du jeu ;
    - views/index.ejs : Fichier de template EJS qui définit la structure HTML de l'interface utilisateur du jeu.
- **Configuration :**
    - package.json : Définit les métadonnées du projet, les dépendances (Express, EJS, Nodemon) et les scripts npm (ex : start, dev) ;
    - public/assets/mots.json : Sert de base de données des mots français valides.
- **Interface Utilisateur :**
    - views/index.ejs : Structure HTML de la page ;
    - public/styles/index.css : Contient tous les styles CSS pour l'apparence, le design responsive et les animations.
- **Gestion des Données :**
    - public/assets/mots.json : Source principale des mots pour le jeu ;
    - Des variables JavaScript globales dans public/scripts/index.js sont utilisées pour maintenir l'état actuel du jeu (mot à deviner, tentatives restantes, lettres saisies, etc.).

3.4. Choix Technologiques

- **Langage(s) de Programmation :** JavaScript ES6+ (utilisé à la fois pour le backend avec Node.js et pour le frontend en vanilla JS), HTML5 (avec templating EJS), CSS3 (avec animations, Grid/Flexbox pour le responsive design).
- **Frameworks et Bibliothèques Majeures :**
    - Express.js : Framework web minimaliste pour Node.js, utilisé pour le serveur ;
    - EJS (Embedded JavaScript templates) : Moteur de template pour générer du HTML dynamiquement côté serveur ;
    - Nodemon : Outil de développement qui redémarre automatiquement le serveur Node.js lors de modifications de fichiers.
- **Base de Données :** Un fichier JSON statique (public/assets/mots.json) contenant une liste d'environ 600 mots français de 5 lettres ;
- **API Utilisées :**
    - API externe : 1mot.net est référencée pour obtenir les définitions des mots (lien externe fourni à l'utilisateur en fin de partie) ;
    - API de polices : Google Fonts (pour la police "Outfit").
- **Environnement de Développement et Outils Notables :** Node.js (environnement d'exécution), NPM (gestionnaire de paquets) ;
- **Système de Gestion de Version :** Git et Github.

4. Gestion et Structure des Données

4.1. Structures de Données Clés (Variables d'état, Dictionnaire)

Les données essentielles au fonctionnement du jeu sont gérées comme suit :

- **Variables d'État Globales (côté client, dans public/scripts/index.js) :**

```
let words = "ABATS,ABBES,ABCES,ABOIE,ABOIS..."; // Chaîne contenant tous les mots valides, séparés par des virgules
let word = "ABATS";                             // Le mot de 5 lettres à deviner pour la partie en cours
let attempts = 0;                               // Compteur du nombre de tentatives effectuées (0 à 5)
let currentLetter = 0;                          // Index de la case active pour la saisie de lettre dans la ligne actuelle (0 à 4)
let hasWon = false;                             // Booléen indiquant si le joueur a gagné la partie
```

- **Structure du Dictionnaire (public/assets/mots.json) :** Un simple tableau de chaînes de caractères.

```
// Extrait de mots.json
[
  "abats",
  "abbés",
  "abcès",
  "aboie",
  // ...
]
```

4.2. Mécanisme de Stockage et Normalisation

- **Côté Serveur :** Le fichier mots.json est lu au démarrage du serveur (via require()) pour sélectionner un mot aléatoire ;
- **Côté Client :** La liste complète des mots valides (words) et le mot à deviner (word) sont transmis au client lors du rendu initial de la page. L'état du jeu est maintenu en mémoire dans des variables JavaScript ;
- **Persistance :** Aucune persistance des données de jeu (score, mot en cours) n'est implémentée entre les sessions. Chaque rechargement de page démarre une nouvelle partie ;

- **Normalisation des Caractères :** Pour gérer correctement les accents français, les mots (à la fois ceux du dictionnaire et ceux saisis par l'utilisateur) sont normalisés. Cela implique une décomposition canonique (NFD) suivie de la suppression des accents et une conversion en majuscules pour assurer des comparaisons cohérentes ;

4.3. Flux de Données Principal

1. **Chargement Initial :**
    - Le serveur Express (index.js racine) lit le fichier mots.json ;
    - Un mot aléatoire de 5 lettres est sélectionné, normalisé (sans accents, en majuscules) et stocké ;
    - La liste complète des mots (également normalisée) et le mot à deviner sont injectés dans le template views/index.ejs lors de son rendu.
2. **Initialisation Côté Client :**
    - Le script public/scripts/index.js récupère le mot à deviner et la liste des mots valides ;
    - L'interface de jeu (grille, clavier virtuel) est initialisée.
3. **Déroulement du Jeu (Côté Client) :**
    - L'utilisateur saisit un mot de 5 lettres ;
    - Lors de la soumission (touche "Entrée"), le mot saisi est validé par rapport à la liste des mots (words) ;
    - Si valide, le mot est comparé lettre par lettre avec le mot cible (word) ;
    - L'interface est mise à jour avec des indicateurs colorés.
4. **Fin de Partie :**
    - Si le mot est deviné ou si les 6 tentatives sont épuisées, un écran modal de fin de partie s'affiche.

5. Fonctionnalités Principales et Implémentation

5.1. Saisie et Validation des Lettres

- **Description :** Permet au joueur de saisir des lettres, soit en cliquant sur le clavier virtuel AZERTY affiché à l'écran, soit en utilisant son clavier physique ;

- **Fichiers/Méthodes Impliqués :** public/scripts/index.js (fonction clickedLetter(letter) et gestionnaire d'événement keydown), views/index.ejs (pour les boutons du clavier virtuel avec onclick) ;
- **Logique Clé :**
    - La saisie est bloquée si un écran modal (pause/fin de jeu) est actif ;
    - La lettre cliquée ou tapée est affichée dans la case active de la ligne de tentative en cours ;
    - La case active change de style (selected -> has-content) ;
    - Le focus se déplace automatiquement à la case suivante, sauf si la ligne est pleine (currentLetter === 4) ;
    - La touche "Backspace" (clickedErase()) permet d'effacer la dernière lettre saisie et de reculer le focus.

```
function clickedLetter(letter) {
  if (pauseScreen.style.opacity == 1) return; // Jeu en pause ou terminé
  if (currentLetter > 4) return; // Ligne pleine

  currentRow[currentLetter].innerHTML = letter; // Afficher la lettre

  if (currentLetter === 4) { // Si c'est la dernière lettre de la ligne
    currentRow[currentLetter].classList.remove("selected");
    currentRow[currentLetter].classList.add("has-content");
    return;
  }

  // Progression vers la case suivante
  currentRow[currentLetter].classList.remove("selected");
  currentRow[currentLetter].classList.add("has-content");
  currentLetter++;
  currentRow[currentLetter].classList.add("selected");
}
```

5.2. Validation et Soumission des Mots

- **Description :** Après avoir saisi un mot de 5 lettres, le joueur appuie sur "Entrée" (ou clique sur un bouton "Entrée" virtuel). Le système vérifie si le mot est complet et s'il existe dans le dictionnaire français fourni ;
- **Fichiers/Méthodes Impliqués :** public/scripts/index.js (fonction clickedEnter()) ;
- **Logique Clé :**
    - Vérifie si la ligne de tentative actuelle est bien remplie (5 lettres) ;
    - Reconstruit le mot saisi à partir des cases de la grille ;
    - Normalise le mot saisi (suppression des accents, majuscules) pour la comparaison ;

- Vérifie si le mot normalisé est présent dans la liste des mots valides (words) ;
- Si l'une des conditions n'est pas remplie, une notification d'erreur temporaire est affichée ;
- Si tout est valide, la fonction setResultStyles() est appelée pour l'évaluation et le feedback coloré, suivie de la logique de victoire/défaite ou de passage à la tentative suivante.

- **Extrait de Code (public/scripts/index.js) :**

```
async function clickedEnter() {
  if (pauseScreen.style.opacity == 1) return;

  if (currentLetter !== 4 || currentRow[currentLetter].innerHTML === "") {
    notification.innerHTML = "Veuillez insérer un mot à 5 lettres.";
    // ... affichage notification ...
    return;
  }

  const currentWordAttempt = (currentRow[0].innerHTML + currentRow[1].innerHTML +
                              currentRow[2].innerHTML + currentRow[3].innerHTML +
                              currentRow[4].innerHTML).normalize("NFD").replace(/[\u0300-
\u036f]/g, "").toUpperCase();

  if (!words.includes(currentWordAttempt)) {
    notification.innerHTML = "Ce mot n'est pas dans la liste.";
    // ... affichage notification ...
    return;
  }

  setResultStyles(); // Appliquer les couleurs
  // Logique de temporisation pour animation avant de vérifier victoire/défaite
  setTimeout(() => {
    if (currentWordAttempt !== word && attempts === 5) return lose(); // Défaite
    if (currentWordAttempt === word) return win(); // Victoire
    // Passage à la ligne suivante...
    // ...
  }, 1000); // Délai pour l'animation de "flip" des lettres
}
```

5.3. Système de Feedback Coloré

- **Description :** Après la soumission d'un mot valide, chaque lettre de la tentative est colorée pour indiquer sa relation avec le mot à deviner, suivant la logique de Wordle ;
- **Fichiers/Méthodes Impliqués :** public/scripts/index.js (fonction setResultStyles()), public/styles/index.css (classes CSS .letter-ok, .letter-exists, .letter-not-exists) ;
- **Logique Clé :**
  - Itère sur les 5 lettres du mot soumis ;
  - **Vert (.letter-ok) :** La lettre est correcte et à la bonne position ;
  - **Jaune (.letter-exists) :** La lettre est présente dans le mot cible mais à une mauvaise position. Une gestion est faite pour ne pas colorer en jaune plus

d'occurrences d'une lettre qu'il n'y en a dans le mot cible (logique de comptage des lettres) ;

- **Gris (.letter-not-exists) :** La lettre n'est pas présente dans le mot cible ;
- Les touches correspondantes sur le clavier virtuel sont également mises à jour avec ces couleurs pour un feedback persistant ;
- Des animations CSS sont appliquées lors du changement de couleur des cases.

- **Extrait de Code (public/scripts/index.js) :**

```
function setResultStyles() {
  const attemptWord = [];
  for(let i=0; i<5; i++) attemptWord.push(currentRow[i].innerHTML);
  const targetWordLetters = word.split(''); // Mot à deviner
  const resultColors = ["", "", "", "", ""]; // Pour stocker la couleur de chaque case

  // 1. Trouver les lettres vertes (correctes et bien placées)
  for (let i = 0; i < 5; i++) {
    if (attemptWord[i] === targetWordLetters[i]) {
      resultColors[i] = "letter-ok";
      targetWordLetters[i] = null; // Marquer la lettre comme utilisée pour éviter double
comptage en jaune
    }
  }

  // 2. Trouver les lettres jaunes (correctes mais mal placées)
  for (let i = 0; i < 5; i++) {
    if (resultColors[i] === "") { // Si pas déjà verte
      const letterIndexInTarget = targetWordLetters.indexOf(attemptWord[i]);
      if (letterIndexInTarget !== -1) {
        resultColors[i] = "letter-exists";
        targetWordLetters[letterIndexInTarget] = null; // Marquer comme utilisée
      } else {
        resultColors[i] = "letter-not-exists";
      }
    }
  }

  // Appliquer les styles et mettre à jour le clavier virtuel
  for (let i = 0; i < 5; i++) {
    currentRow[i].classList.add(resultColors[i]);
    const keyElement = document.getElementById(attemptWord[i]);
    if (keyElement) { // Appliquer aussi au clavier virtuel
        // Logique pour ne pas remplacer vert par jaune, etc.
        if (resultColors[i] === "letter-ok") {
            keyElement.classList.remove("exists", "not-exists");
            keyElement.classList.add("ok");
        } else if (resultColors[i] === "letter-exists" && !
keyElement.classList.contains("ok")) {
            keyElement.classList.add("exists");
        } else if (resultColors[i] === "letter-not-exists" && !
keyElement.classList.contains("ok") && !keyElement.classList.contains("exists")) {
            keyElement.classList.add("not-exists");
        }
    }
  }
}
```

5.4. Gestion des États de Fin de Jeu (Victoire/Défaite)

- **Description :** Détecte si le joueur a gagné (mot deviné) ou perdu (6 tentatives épuisées sans trouver le mot) et affiche un écran modal approprié ;
- **Fichiers/Méthodes Impliqués :** public/scripts/index.js (fonctions win() et lose()) ;
- **Logique Clé :**
  - La variable hasWon est mise à true en cas de victoire ;
  - L'écran modal (pauseScreen) est rendu visible et opaque ;
  - Le contenu du modal (pauseContainer) est généré dynamiquement pour afficher un message ("GAGNÉ" ou "PERDU"), le mot qui était à deviner, et des boutons d'action ;
  - Un bouton permet de consulter la définition du mot sur 1mot.net ;
  - Un bouton "Rejouer" appelle la fonction resetGame().
- **Extrait de Code (public/scripts/index.js - fonction win()) :**

```
function win() {
  hasWon = true;
  pauseScreen.style.visibility = "visible";
  pauseScreen.style.opacity = 1;

  let title = document.createElement("h1");
  title.innerHTML = "GAGNÉ"; /* ... */ pauseContainer.appendChild(title);

  let text = document.createElement("p");
  text.innerHTML = "Le mot était"; pauseContainer.appendChild(text);

  title = document.createElement("h1");
  title.innerHTML = word; pauseContainer.appendChild(title);

  // Bouton pour la définition
  let link = document.createElement("a");
  link.setAttribute("href", `https://1mot.net/${word.toLowerCase()}`); /* ... */
  let button = document.createElement("button"); /* ... */ link.appendChild(button);
  pauseContainer.appendChild(link);

  // Bouton Rejouer
  let resetButton = document.createElement("button");
  resetButton.innerHTML = "Rejouer";
  resetButton.onclick = resetGame; /* ... */ pauseContainer.appendChild(resetButton);
}
// La fonction lose() est structurée de manière similaire.
```

5.5. Fonction de Réinitialisation du Jeu

- **Description :** Permet de démarrer une nouvelle partie en remettant à zéro l'état du jeu et en sélectionnant un nouveau mot aléatoire ;
- **Fichiers/Méthodes Impliqués :** public/scripts/index.js (fonction resetGame()) ;
- **Logique Clé :**

- Vide le contenu du conteneur modal (pauseContainer) ;
- Nettoie la grille de jeu : efface les lettres saisies et supprime toutes les classes CSS de feedback coloré des cases ;
- Nettoie les couleurs des touches du clavier virtuel ;
- Réinitialise les variables d'état globales (currentLetter, attempts, hasWon) ;
- Sélectionne un nouveau mot aléatoire à partir de la liste words (qui est une chaîne, donc elle est d'abord divisée en tableau) ;
- Masque l'écran modal ;
- Réinitialise la sélection de case à la première de la première ligne.

- **Extrait de Code (public/scripts/index.js) :**

```
function resetGame() {
  pauseContainer.innerHTML = ""; // Vide le modal

  // Nettoyage de la grille (6 lignes x 5 cases)
  for (let i = 0; i < 6; i++) {
    const currentResultRow = document.getElementsByClassName("result-row")[i];
    for (let j = 0; j < 5; j++) {
      const currentResultLetter = currentResultRow.getElementsByClassName("result-letter")[j];
      currentResultLetter.classList.remove("selected", "has-content", "letter-ok", "letter-exists",
"letter-not-exists");
      currentResultLetter.innerHTML = "";
    }
  }
  // Nettoyage du clavier virtuel
  // ...

  currentLetter = 0;
  attempts = 0;
  hasWon = false;

  const wordsArray = words.split(","); // Convertir la chaîne en tableau
  word = wordsArray[Math.floor(Math.random() * wordsArray.length)]; // Nouveau mot
  // Normaliser le nouveau mot (important si la liste initiale n'est pas déjà normalisée)
  word = word.normalize("NFD").replace(/[\u0300-\u036f]/g, "").toUpperCase();

  // Réinitialiser la sélection de la première case
  currentRow = document.getElementsByClassName("result-row")
[attempts].getElementsByClassName("result-letter");
  currentRow[currentLetter].classList.add("selected");

  pauseScreen.style.opacity = 0;
  setTimeout(() => pauseScreen.style.visibility = "hidden", 250); // Délai pour transition
}
```

6. Interface Utilisateur (UI)

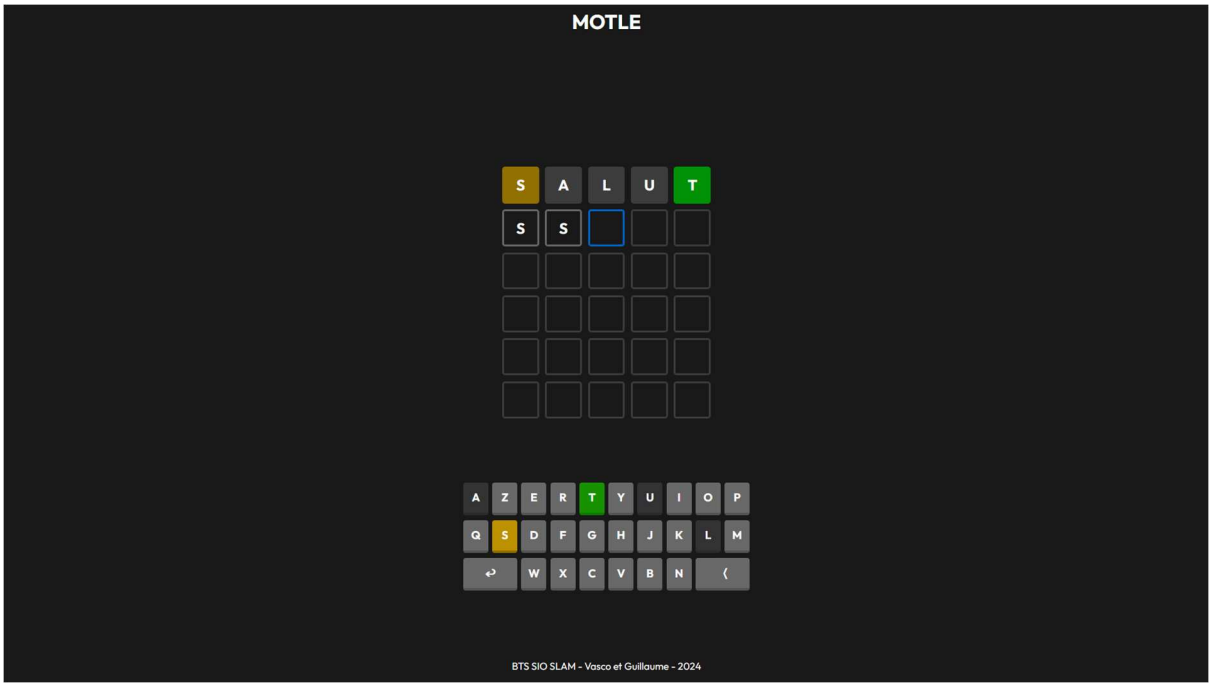
6.1. Description des Écrans/Vues Principales

L'application se présente sur un écran unique qui évolue en fonction de l'état du jeu.

- **Écran Principal de Jeu :**

- **Barre de Navigation (Navbar) :** Affiche le titre "MOTLE" de manière centrée ;
- **Zone de Notification :** Un espace discret (au-dessus de la grille) pour afficher des messages temporaires à l'utilisateur (ex: "Ce mot n'est pas dans la liste.", "Veuillez insérer un mot à 5 lettres.") ;
- **Grille de Résultats :** Une grille de 6 lignes par 5 colonnes. Chaque case représente une lettre d'une tentative. Les cases changent de couleur après chaque soumission. La case active pour la saisie est visuellement distincte (classe .selected) ;
- **Clavier Virtuel :** Un clavier AZERTY est affiché en bas de l'écran, permettant la saisie par clic. Les touches du clavier changent également de couleur pour refléter les indices (vert, jaune, gris) ;
- **Pied de Page (Footer) :** Affiche les crédits des développeurs.

- **Écran Modal de Fin de Partie :**
  - Apparaît en superposition (overlay sombre semi-transparent) lorsque la partie est gagnée ou perdue ;
  - Affiche un message clair ("GAGNÉ" ou "PERDU") ;
  - Révèle le mot qui était à deviner ;
  - Propose des boutons d'action : un lien pour voir la définition du mot sur 1mot.net et un bouton "Rejouer" pour lancer une nouvelle partie.



6.2. Flux d'Interaction Utilisateur Principaux

1. **Démarrage du Jeu :** La page se charge, un mot aléatoire est sélectionné par le serveur et transmis au client. La première case de la première ligne est sélectionnée pour la saisie ;

2. **Saisie d'une Lettre :**
    - L'utilisateur clique sur une touche du clavier virtuel OU tape une lettre sur son clavier physique ;
    - La lettre s'affiche dans la case active. Le focus passe à la case suivante.
3. **Correction :** L'utilisateur appuie sur "Retour" (clavier physique ou virtuel) pour effacer la dernière lettre :
4. **Soumission d'un Mot :**
    - Une fois 5 lettres saisies, l'utilisateur appuie sur "Entrée" (physique ou virtuel) :
    - Le mot est validé (longueur, existence dans le dictionnaire) ;
    - Les couleurs des cases et des touches du clavier sont mises à jour.
5. **Répétition :** Si le mot n'est pas trouvé et qu'il reste des tentatives, le focus passe à la première case de la ligne suivante ;
6. **Fin de Partie :**
    - Si le mot est trouvé : Modal "GAGNÉ" avec options ;
    - Si 6 tentatives sont écoulées sans succès : Modal "PERDU" avec options :
    - L'utilisateur clique sur "Rejouer" pour démarrer une nouvelle partie.

7. Conclusion

Le projet "Motle" a été une petite initiation à la maîtrise des technologies web full-stack fondamentales (Node.js, Express.js, EJS, JavaScript vanilla, HTML, CSS). Il nous a permis de développer des compétences pratiques en conception d'interfaces utilisateur, en logique de jeu, en gestion d'état côté client, et en interaction client-serveur simple. C’était une base solide et un projet d'études intéressant pour une première année d’études supérieures.