

# WebTerminal

« Création d'un portfolio interactif sous la forme d'un terminal Linux simulé dans un navigateur web. »

# Table des matières

|   |    |
|---|----|
| 1. Introduction .....   | 3  |
| 2. Objectifs du Projet .....                                    | 3  |
| 2.1 Contexte de Développement .....                             | 3  |
| 2.2 Objectifs Principaux .....                                  | 3  |
| 3. Architecture de l'Application.....                           | 3  |
| 3.1. Style Architectural et Structure.....                      | 3  |
| 3.2. Modules et Composants Clés.....                            | 4  |
| 3.3. Fichiers Essentiels et Rôles.....                          | 4  |
| 3.4. Choix Technologiques.....                                  | 4  |
| 4. Gestion et Structure des Données .....                       | 5  |
| 4.1. Structures de Données Clés .....                           | 5  |
| 4.2. Mécanisme de Stockage.....                                 | 6  |
| 4.3. Flux de Données Principal .....                            | 6  |
| 5. Fonctionnalités Principales et Implémentation .....          | 7  |
| 5.1. Système de Commandes Modulaire .....                       | 7  |
| 5.2. Navigation dans le Système de Fichiers Virtuel.....        | 8  |
| 5.3. Système de Thèmes Dynamiques .....                         | 8  |
| 5.4. Exécution de Scripts Bash Simulés .....                    | 9  |
| 6. Interface Utilisateur (UI) .....                             | 10 |
| 6.1. Description Générale et Composants Clés.....               | 10 |
| 6.2. Flux d'Interaction Utilisateur Principaux.....             | 10 |
| 7. Points Particuliers, Difficultés Notables et Solutions ..... | 10 |
| 7.1. Choix de Conception .....                                  | 10 |
| 7.2. Défis Techniques Surmontés .....                           | 11 |
| 8. Perspectives d'Évolution et Améliorations Futures .....      | 11 |
| 9. Conclusion.....  | 12 |

## 1. Introduction

Ce projet est un portfolio web interactif qui simule l'apparence et le comportement d'un terminal de type Linux directement dans le navigateur. Il permet aux visiteurs de découvrir mes informations professionnelles et personnelles, en utilisant des commandes Unix/Linux classiques.

## 2. Objectifs du Projet

### 2.1 Contexte de Développement

Inspiré par mon expérience professionnelle avec l'automatisation via des scripts Bash sous Linux chez SabSystem, et réalisé dans le cadre des études en BTS SIO SLAM, ce projet vise à créer un portfolio unique. Il a pour but de refléter ma passion pour les environnements de terminal tout en exploitant les technologies web modernes.

### 2.2 Objectifs Principaux

- **Apprentissage Approfondi** : Maîtriser JavaScript moderne (modules ES6, manipulation avancée du DOM, async/await) et les techniques CSS avancées ;
- **Portfolio Original** : Présenter mes compétences techniques et mon parcours de manière interactive, engageante et mémorable ;
- **Expérience Utilisateur Innovante** : Offrir une interface qui combine la familiarité d'un terminal avec l'accessibilité du web ;
- **Internationalisation** : Assurer un support bilingue (français/anglais) pour une portée plus large ;
- **Extensibilité** : Concevoir un système de commandes facilement extensible.

## 3. Architecture de l'Application

### 3.1. Style Architectural et Structure

L'application est une architecture modulaire côté client. Elle s'inspire des principes suivants :

- **Pattern MVC simplifié** :
  - **Modèle** : scripts/data.js (structure du système de fichiers virtuel, configuration des commandes, textes de localisation) ;
  - **Vue** : index.html (structure HTML) et les fichiers CSS dans styles/ (présentation) ;
  - **Contrôleur** : scripts/index.js (logique principale, gestion des événements, interprétation des commandes).
- **Command Pattern** : Chaque commande du terminal (ex: ls, cat) est implémentée comme un module ES6 indépendant dans le dossier « commands/ » ;
- **Architecture de Plugins (pour les commandes)** : Le système permet le chargement dynamique des modules de commandes, facilitant l'ajout de nouvelles commandes sans modifier le cœur de l'application.

## 3.2. Modules et Composants Clés

La structure du projet est organisée comme suit :

- **index.html** : Point d'entrée principal de l'application ;
- **commands/** : Contient les modules JavaScript pour chaque commande du terminal (12 modules identifiés, ex: help.js, ls.js, theme.js) ;
- **scripts/** : Renferme la logique métier et les utilitaires :
  - **index.js** : Contrôleur principal, gestion des entrées utilisateur, exécution des commandes ;
  - **functions.js** : Fonctions utilitaires (ex: vérification de l'existence de fichiers, gestion des erreurs) ;
  - **data.js** : Modèle de données (système de fichiers virtuel, liste des commandes, textes pour l'internationalisation).
- **styles/** : Comprend les fichiers de style CSS pour les différents thèmes visuels ;
- **assets/** : Contient les ressources statiques comme les polices (JetBrains Mono) et les images ;
- **tests/** : Contient une version expérimentale du projet.

## 3.3. Fichiers Essentiels et Rôles

- **Point d'Entrée** : index.html structure la page et charge les scripts nécessaires ;
- **Configuration Centrale** : scripts/data.js est crucial car il définit la structure du système de fichiers simulé, la liste des commandes disponibles, leurs descriptions, et les chaînes de caractères pour la localisation ;
- **Logique Principale** : scripts/index.js orchestre l'interaction, l'interprétation des saisies, et le chargement/exécution des modules de commande ;
- **Stylisation** : Les fichiers styles/theme\*.css définissent l'apparence du terminal. La police JetBrains Mono est utilisée.

## 3.4. Choix Technologiques

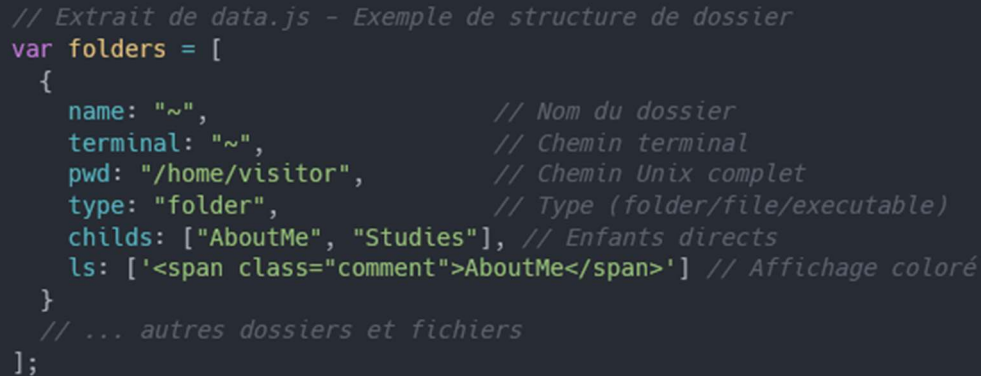
- **Langages** : JavaScript ES6+ (modules, async/await, manipulation du DOM), HTML5, CSS3 ;
- **Frameworks/Bibliothèques** : Aucun framework JavaScript externe majeur (Vanilla JavaScript) pour maximiser l'apprentissage ;
- **API Navigateur** : DOM API, Dynamic Import API (pour le chargement des modules de commande), XMLHttpRequest (utilisé pour vérifier l'existence des fichiers de commande avant import) ;
- **Outils de Développement** : VS Code, Live Server (pour le développement local), Git et Github.

## 4. Gestion et Structure des Données

### 4.1. Structures de Données Clés

Les données sont principalement structurées en JavaScript au sein de scripts/data.js.

- **Système de Fichiers Virtuel** : Un tableau d'objets (folders) simule une hiérarchie de dossiers et de fichiers. Chaque objet peut contenir :
  - **name** : Nom de l'élément ;
  - **terminal** : Chemin affiché dans le terminal ;
  - **pwd** : Chemin Unix complet ;
  - **type** : "folder", "file", ou "executable" ;
  - **childs** : Tableau des enfants directs (pour les dossiers) ;
  - **content (pour les fichiers)** : Le contenu textuel du fichier, potentiellement localisé ;
  - **bash (pour les exécutables)** : Code JavaScript à évaluer ;
  - **ls** : Chaîne HTML formatée pour la commande ls.



```
// Extrait de data.js - Exemple de structure de dossier
var folders = [
  {
    name: "~", // Nom du dossier
    terminal: "~", // Chemin terminal
    pwd: "/home/visitor", // Chemin Unix complet
    type: "folder", // Type (folder/file/executable)
    childs: ["AboutMe", "Studies"], // Enfants directs
    ls: ['<span class="comment">AboutMe</span>'] // Affichage coloré
  }
  // ... autres dossiers et fichiers
];
```

- **Configuration des Commandes** : Un tableau commandList liste les noms des commandes disponibles. Des objets peuvent stocker les descriptions et usages pour chaque langue ;

```
// Extrait de data.js - Liste des commandes
var commandList = [
  "help", "banner", "pwd", "ls", "cd", "cat",
  "echo", "clear", "bash", "project", "theme", "lang"
];
```

- **Localisation (Internationalisation)** : Un objet `textsLocalization` stocke les chaînes de caractères en français (fr) et en anglais (en) pour les messages de l'interface, les descriptions des commandes, etc.

```
// Extrait de data.js - Textes localisés
var textsLocalization = {
  welcomeMessage: {
    fr: "Bienvenue sur le terminal portfolio...",
    en: "Welcome to the terminal portfolio..."
  }
  // ... autres textes
};
```

## 4.2. Mécanisme de Stockage

- **État en Mémoire (Variables JavaScript)** : L'état actuel du terminal (dossier courant, langue sélectionnée, historique des commandes tapées) est maintenu dans des variables JavaScript globales ou au sein du scope du contrôleur principal (`index.js`) ;
- **Données Statiques (dans `data.js`)** : La structure du système de fichiers et le contenu des fichiers virtuels sont définis statiquement.

## 4.3. Flux de Données Principal

1. L'utilisateur saisit une commande dans l'input du terminal ;
2. `scripts/index.js` capture l'événement (touche "Entrée") ;

3. La chaîne saisie est analysée pour extraire le nom de la commande et ses arguments ;
4. Le système vérifie si le fichier du module de commande correspondant existe dans `commands/` (via `XMLHttpRequest` dans `functions.js`) ;
5. Si le module existe, il est chargé dynamiquement (`import("../commands/" + commandName + ".js")`) ;
6. La fonction `run` du module de commande est exécutée avec la commande et ses arguments ;
7. Le module de commande interagit avec les données (système de fichiers virtuel, état global) et génère une sortie ;
8. La sortie est formatée en HTML et ajoutée au DOM pour simuler l'affichage du terminal ;
9. Une nouvelle ligne de prompt est créée, prête pour la prochaine commande.

## 5. Fonctionnalités Principales et Implémentation

### 5.1. Système de Commandes Modulaire

- **Description** : Le cœur du terminal est son système de commandes extensible. Chaque commande est un module ES6 indépendant, ce qui facilite l'ajout, la modification ou la suppression de commandes ;
- **Fichiers Impliqués** : `scripts/index.js` (fonction `runCommand`), `scripts/functions.js` (fonction `fileExists`), et tous les fichiers `.js` dans le dossier `commands/` ;
- **Logique Clé** : Utilisation de l'import dynamique (`await import(...)`) pour charger le module de commande approprié seulement lorsqu'il est appelé. Une vérification préalable de l'existence du fichier de commande est effectuée ;

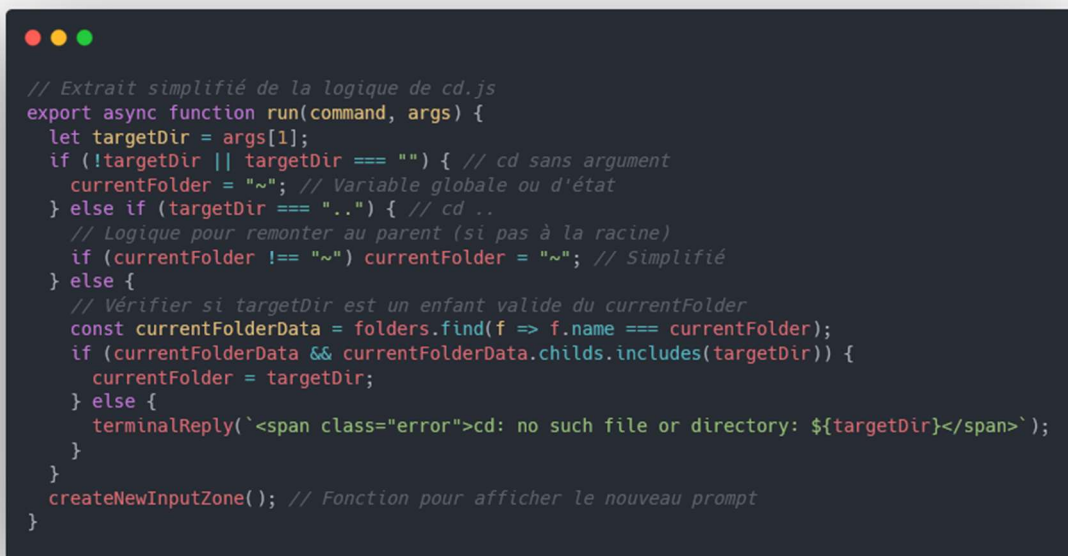
```
// Dans scripts/index.js (ou une fonction appelée par celui-ci)
async function runCommand(commandName, argsArray) { // Simplifié
  // Vérification de l'existence du fichier de commande
  let commandFileExists = await fileExists("../commands/" + commandName + ".js");

  if (!commandFileExists) {
    return runError("not-found", commandName); // Affiche une erreur
  }

  // Import dynamique et exécution
  try {
    var commandModule = await import("../commands/" + commandName + ".js");
    return commandModule.run(commandName, argsArray); // Appel de la fonction run du module
  } catch (error) {
    console.error("Error running command:", commandName, error);
    // Gérer l'erreur d'exécution du module
  }
}
```

## 5.2. Navigation dans le Système de Fichiers Virtuel

- **Description** : L'application simule un système de fichiers Unix, permettant à l'utilisateur de naviguer entre les dossiers (cd), de lister leur contenu (ls), d'afficher le chemin du répertoire courant (pwd), et de lire le contenu des fichiers (cat) ;
- **Fichiers Impliqués** : commands/cd.js, commands/ls.js, commands/pwd.js, commands/cat.js, et scripts/data.js (pour la définition de la structure) ;
- **Logique Clé** : Les commandes de navigation modifient une variable globale (ou d'état) qui représente le dossier courant. Elles valident les chemins par rapport à la structure définie dans data.js.



```
// Extrait simplifié de la logique de cd.js
export async function run(command, args) {
  let targetDir = args[1];
  if (!targetDir || targetDir === "") { // cd sans argument
    currentFolder = "~"; // Variable globale ou d'état
  } else if (targetDir === "..") { // cd ..
    // Logique pour remonter au parent (si pas à la racine)
    if (currentFolder !== "~") currentFolder = "~"; // Simplifié
  } else {
    // Vérifier si targetDir est un enfant valide du currentFolder
    const currentFolderData = folders.find(f => f.name === currentFolder);
    if (currentFolderData && currentFolderData.chlds.includes(targetDir)) {
      currentFolder = targetDir;
    } else {
      terminalReply(`<span class="error">cd: no such file or directory: ${targetDir}</span>`);
    }
  }
  createNewInputZone(); // Fonction pour afficher le nouveau prompt
}
```

## 5.3. Système de Thèmes Dynamiques

- **Description** : L'utilisateur peut changer l'apparence visuelle du terminal en temps réel en choisissant parmi trois thèmes prédéfinis (Dracula, Orange, Terminal Vert) ;
- **Fichiers Impliqués** : commands/theme.js, les fichiers CSS dans styles/ (ex: theme0.css, theme1.css, theme2.css), et index.html (où le lien vers la feuille de style est modifié) ;
- **Logique Clé** : La commande theme modifie dynamiquement l'attribut href de l'élément <link> qui charge la feuille de style CSS du thème.



```

// Dans commands/theme.js
export async function run(command, args) {
  const themeId = args[1]; // ex: "0", "1", "2"
  const usableArgs = ["0", "1", "2"]; // Thèmes valides
  if (args.length === 1 || !usableArgs.includes(themeId)) {
    terminalReply(`<span class="error">${usage[lang]}</span>`); // 'usage' est un objet de localisation
  } else {
    var styleLink = document.getElementById("styleLink"); // ID de la balise <link>
    styleLink.href = `styles/theme${themeId}.css`;
    terminalReply(`<span class="comment">${success[lang]}</span>`); // 'success' est
    // localisé
  }
  createNewInputZone();
}

```

## 5.4. Exécution de Scripts Bash Simulés

- **Description** : La commande bash permet de simuler l'exécution de fichiers .sh. Dans ce portfolio, ces scripts ouvrent des liens vers les réseaux sociaux ou d'autres profils de l'auteur ;
- **Fichiers Impliqués** : commands/bash.js, scripts/data.js (où les "scripts" sont définis avec leur code JavaScript associé dans la structure du système de fichiers virtuel) ;
- **Logique Clé** : La commande bash recherche le fichier .sh spécifié dans la structure de données. Si trouvé, elle exécute le code JavaScript stocké dans la propriété bash de l'objet fichier en utilisant eval().

```

// Dans commands/bash.js
export async function run(command, args) {
  const scriptName = args[1];
  const currentFolderData = folders.find(f => f.name === currentFolder);
  const scriptData = currentFolderData && currentFolderData.childs
    ? folders.find(f => f.name === scriptName && currentFolderData.childs.includes(scriptName) &&
      f.type === "executable" && scriptName.endsWith(".sh"))
    : null;

  if (scriptData && scriptData.bash) {
    try {
      eval(scriptData.bash); // Exécution du code JS associé au script .sh
      terminalReply(`<span class="normalized">${bashScripts[scriptName][lang]}</span>`); // Message de
      // succès localisé
    } catch (e) {
      terminalReply(`<span class="error">Error executing script: ${scriptName}</span>`);
      console.error(e);
    }
  } else {
    terminalReply(`<span class="error">bash: ${scriptName}: No such file or not an executable script</
    span>`);
  }
  createNewInputZone();
}

```

## 6. Interface Utilisateur (UI)

### 6.1. Description Générale et Composants Clés

L'interface utilisateur est conçue pour imiter fidèlement un terminal Linux :

- **Structure HTML (index.html) :**
  - Un conteneur principal `#terminal` ;
  - Des éléments pour le logo ASCII (`#logoMessage`), le message de bienvenue (`#welcomeMessage`), et les instructions d'aide initiales (`#helpMessage`) ;
  - Les lignes de commande et leurs sorties sont ajoutées dynamiquement dans ce conteneur. Chaque nouvelle ligne de saisie est une div avec la classe `currentLine` contenant le prompt et un input (ou un span éditable).
- **Stylisation (CSS) :**
  - Utilisation de la police mono-espace JetBrains Mono ;
  - Variables CSS pour les couleurs des thèmes, permettant un changement facile ;
  - Classes CSS pour styliser les différents types de messages : `.error`, `.success`, `.comment`, `.normalized`.
- **Composants Dynamiques (JavaScript) :**
  - Les lignes de prompt et les zones de saisie sont créées et gérées par JavaScript ;
  - L'historique des commandes est affiché en ajoutant des div au DOM.

### 6.2. Flux d'Interaction Utilisateur Principaux

1. **Focus Automatique** : Un clic n'importe où sur la page principale met le focus sur la zone de saisie active du terminal ;
2. **Saisie de Commande** : L'utilisateur tape une commande et ses arguments, puis appuie sur "Entrée" pour la valider ;
3. **Autocomplétion** : La touche "Tab" permet de compléter automatiquement les noms de commandes ou les arguments (fichiers/dossiers) en fonction du contexte ;
4. **Historique des Commandes** : Les flèches haut et bas du clavier permettent de naviguer dans l'historique des commandes précédemment saisies ;
5. **Commandes Spéciales** : Des commandes comme `theme [nom_theme]` ou `lang [fr|en]` modifient l'état de l'interface.

## 7. Points Particuliers, Difficultés Notables et Solutions

### 7.1. Choix de Conception

- **Modules ES6 vs. `<script>` tags** : L'adoption des modules ES6 a été un choix délibéré pour favoriser une architecture modulaire, extensible et moderne, malgré la nécessité d'un serveur HTTP local pour le développement (à cause des restrictions CORS) ;

- **Utilisation de eval() pour bash.js** : Ce choix permet une simulation flexible de l'exécution de scripts, mais introduit des considérations de sécurité si le contenu des "scripts" n'est pas entièrement statique, mais dans ce projet-là, le risque peut être ignoré ;
- **Système de Fichiers Virtuel Détaillé** : La création d'une structure de données complexe pour le système de fichiers virtuel contribue grandement au réalisme de la simulation.

## 7.2. Défis Techniques Surmontés

- **Import Dynamique avec Gestion d'Erreurs** : La vérification de l'existence d'un fichier de module de commande avant de tenter un import() dynamique (via XMLHttpRequest synchrone ou asynchrone dans fileExists) est une solution pour gérer les commandes inconnues proprement ;
- **Autocomplétion Contextuelle** : Implémenter un système d'autocomplétion qui suggère des commandes ou des chemins de fichiers/dossiers pertinents en fonction de la saisie actuelle ;
- **Gestion de l'État du DOM Dynamique** : Maintenir la cohérence de l'affichage (historique, prompt actuel) tout en ajoutant et modifiant dynamiquement des éléments du DOM.

## 8. Perspectives d'Évolution et Améliorations Futures

- **Finalisation du dossier tests/** : Intégrer et finaliser la version refactorisée du code pour une meilleure maintenabilité et propreté ;
- **Persistance des Préférences Utilisateur** : Utiliser localStorage pour sauvegarder le thème et la langue choisis par l'utilisateur entre les sessions ;
- **Complétion du Contenu** : Finaliser et enrichir le contenu des fichiers virtuels (ex: personal.txt, professional.txt, projects.txt) pour rendre le portfolio plus informatif ;
- **Documentation Utilisateur Intégrée** : Améliorer la commande help ou ajouter un manuel (man) pour fournir un guide détaillé de toutes les commandes disponibles et de leurs options ;
- **Ajout de Nouvelles Commandes Utiles/Ludiques** :
  - history : Afficher l'historique des commandes ;
  - date : Afficher la date et l'heure actuelles ;
  - clearhistory (ou option de history) : Effacer l'historique des commandes ;
  - Peut-être des commandes plus interactives ou des mini-jeux simples.
- **Amélioration de l'Autocomplétion** : Rendre l'autocomplétion encore plus intelligente, en suggérant par exemple les options des commandes ;
- **Accessibilité (a11y)** : Revoir l'accessibilité pour les utilisateurs de lecteurs d'écran et la navigation au clavier ;

- **Tests Automatisés** : Mettre en place des tests unitaires pour les modules de commande et des tests d'intégration pour le comportement global du terminal.

## 9. Conclusion

Avec le "Web Terminal Portfolio", j'ai cherché à créer une vitrine de mes compétences qui soit à la fois originale, interactive et techniquement intéressante. Ce projet m'a permis de plonger en profondeur dans JavaScript moderne, la manipulation du DOM, et les subtilités de CSS, tout en construisant une simulation authentique d'un terminal Linux. L'architecture modulaire des commandes, le système de fichiers virtuel, l'internationalisation et les thèmes dynamiques sont autant de facettes qui démontrent, je l'espère, mon souci du détail et ma capacité à concevoir des applications web complexes et soignées. C'était difficile de recréer l'expérience utilisateur d'un terminal en utilisant uniquement les technologies web standards, et je suis fier du résultat obtenu qui, je pense, reflète ma passion pour le développement et les environnements en ligne de commande.