INSTITUT NATIONAL SUPERIEUR
EN INFORMATIQUE

RAPPORT COMPLET

Developpement Web

HTML • CSS • JavaScript

Analyse des projets pedagogiques

Presente par

Muckaya Florida
RAMALALAHASITERA

L1B — N°331

Table des matières

Introduction

Ce rapport presente une synthese de ma comprehension des technologies web fondamentales a travers l'analyse de six projets pratiques. Durant cette formation, j'ai eu l'opportunite d'explorer les interactions entre HTML, CSS et JavaScript en developpant des applications variees.

Objectifs du rapport

L'objectif principal de ce document est de demontrer ma maitrise progressive des concepts suivants :

- **HTML** : Structuration semantique du contenu
- **CSS** : Mise en forme et design responsive
- **JavaScript** : Interactivite et logique applicative

Methodologie

Chaque projet a ete aborde selon une demarche structuree en trois phases :

1. Conception de la structure HTML
2. Stylisation avec CSS
3. Implementation de la logique JavaScript

Les projets analyses couvrent un large spectre d'applications web, allant d'un simple quiz interactif a des applications plus complexes comme un Pokedex ou une application meteo consommant des API externes.

Chapitre 1

Quiz Interactif

1.1 Vue d'ensemble du projet

Le quiz interactif represente mon premier projet d'application dynamique complete. Il s'agit d'un questionnaire a choix multiples qui evalue les reponses de l'utilisateur et fournit un retour immediat avec des animations visuelles.

1.2 Structure HTML

La structure HTML du quiz est organisee de maniere modulaire. Chaque question est encapsulee dans un bloc `div` avec la classe `question-block`, ce qui facilite la manipulation en JavaScript.

Points cles HTML

- Utilisation de `<form>` pour regrouper les questions
- Inputs de type `radio` pour les choix multiples
- Labels associes aux inputs pour ameliorer l'accessibilite
- Section dediee pour l'affichage des resultats

1.3 Mise en forme CSS

Le CSS apporte une dimension visuelle professionnelle au quiz. J'ai particulierement apprecie l'utilisation des pseudo-classes et des animations.

1.3.1 Techniques CSS notables

Variables CSS (Custom Properties) : J'ai decouvert l'utilisation du selecteur `:root` pour definir des variables de couleurs, ce qui facilite grandement la maintenance du code.

```
:root {
    --lcd: #fff;
    --light: #f1f1f1;
    --dark: #333;
    --dark2: #000022;
```

```
}
```

Listing 1.1 – Utilisation des variables CSS

Animations avec @keyframes : La classe `.echec` utilise une animation de tremblement pour indiquer les reponses incorrectes, ce qui ameliore l'experience utilisateur.

```
@keyframes echec {
    0% { transform: translateX(0px); }
    33% { transform: translateX(5px); }
    66% { transform: translateX(-5px); }
    100% { transform: translateX(0px); }
}
```

Listing 1.2 – Animation de tremblement

1.4 Logique JavaScript

C'est dans la partie JavaScript que reside toute la complexite du projet. J'ai structure le code en plusieurs fonctions pour une meilleure lisibilite.

1.4.1 Gestion des evenements

L'evenement `submit` du formulaire est intercepte pour eviter le rechargement de la page et traiter les reponses en JavaScript.

```
form.addEventListener('submit', (e) => {
    e.preventDefault();
    for(i = 1; i < 6; i++){
        tableauResultats.push(
            document.querySelectorAll(
                `input[name="q${i}"]:checked`
            ).value
        );
    }
    verifFunc(tableauResultats);
    tableauResultats = [];
})
```

Listing 1.3 – Gestion de la soumission

1.4.2 Verification des reponses

La fonction `verifFunc()` compare les reponses de l'utilisateur avec les bonnes reponses et genere un tableau de booleens.

Ma comprehension

J'ai compris que cette approche modulaire permet de separer les responsabilites : une fonction pour verifier, une autre pour afficher les resultats, et une derniere pour gerer les couleurs. Cela rend le code plus maintenable et evolutif.

1.4.3 Retour visuel dynamique

La fonction `couleursFonction()` change la couleur de fond des questions selon leur validite. L'utilisation de `setTimeout()` pour retirer la classe d'animation est une technique que j'ai trouvee particulierement astucieuse.

```
function couleursFonction(tabValBool){
    for(let j = 0; j < tabValBool.length; j++){
        if(tabValBool[j] === true){
            toutesLesQuestions[j].style.background
                = 'lightgreen';
        } else {
            toutesLesQuestions[j].style.background
                = '#ffb8b8';
            toutesLesQuestions[j].classList.add('echec');

            setTimeout(() => {
                toutesLesQuestions[j].classList
                    .remove('echec');
            }, 500);
        }
    }
}
```

Listing 1.4 – Gestion des couleurs et animations

1.5 Apprentissages cles

1. **Manipulation du DOM** : J'ai appris a selectionner et modifier des elements HTML dynamiquement
2. **Gestion des formulaires** : La methode `preventDefault()` est essentielle pour controler le comportement des formulaires
3. **Tableaux et iterations** : L'utilisation de `filter()` pour compter les erreurs est elegante
4. **Switch case** : Cette structure de controle est ideale pour gerer les differents cas de resultats

Chapitre 2

Application Meteo

2.1 Presentation du projet

L'application meteo constitue mon premier contact avec les APIs externes. Elle recupere la geolocalisation de l'utilisateur et affiche les previsions meteorologiques en temps reel.

2.2 Architecture HTML

La structure HTML suit un modele en grille avec trois sections principales : le titre, les informations actuelles, et les previsions.

Structure organisee

- `container` : Conteneur principal en display grid
- `bloc-logo-info` : Affichage des conditions actuelles
- `heure-bloc-prevision` : Previsions par intervalle de 3 heures
- `jour-prevision-bloc` : Previsions pour la semaine

2.3 Design avec CSS Grid

C'est dans ce projet que j'ai vraiment compris la puissance de CSS Grid. La propriete `grid-template` permet de definir une mise en page complexe en une seule ligne.

```
.container {
    display: grid;
    grid-template: 100px 300px 100px 100px /
                   repeat(8, 100px);
    background: linear-gradient(45deg, #1968b6,
                #ec60a6);
}
```

Listing 2.1 – Configuration de la grille

2.3.1 Positionnement avec grid-area

La propriete `grid-area` simplifie le positionnement des elements dans la grille. Par exemple, pour faire occuper toute la premiere ligne au titre :

```
h1 {
    grid-area: 1 / 1 / 2 / -1;
}
```

Listing 2.2 – Positionnement du titre

2.4 JavaScript et APIs

2.4.1 Geolocalisation

La geolocalisation utilise l'API native du navigateur. J'ai appris a gerer les cas de succes et d'echec avec des fonctions callback.

```
navigator.geolocation.getCurrentPosition(
    position => {
        let long = position.coords.longitude;
        let lat = position.coords.latitude;
        AppelAPI(long, lat);
    },
    () => {
        alert("Vous avez refuse la geolocalisation,
               veuillez l'activer");
    }
);
```

Listing 2.3 – Recuperation de la position

2.4.2 Appels API avec Fetch

L'utilisation de `fetch()` m'a permis de comprendre les Promises et le chainage avec `.then()`.

```
function AppelAPI(long, lat){
    fetch(`https://api.openweathermap.org/data/2.5/
           onecall?lat=${lat}&lon=${long}
           &exclude=minutely&units=metric&lang=fr
           &appid=${CLEFAPI}`)
        .then(response => response.json())
        .then((data) => {
            resultatAPI = data;

            temps.innerText =
                resultatAPI.current.weather[0].description;
            temperature.innerText =
                `${Math.trunc(resultatAPI.current.temp)}  `;
            localisation.innerText =
                resultatAPI.timezone;
        })
}
```

Listing 2.4 – Appel a l'API OpenWeather

2.4.3 Manipulation des donnees temporelles

La gestion des heures et des jours m'a appris a travailler avec l'objet `Date` et a manipuler les tableaux.

Astuce decouverte

Pour creer un tableau des jours de la semaine commencant par aujourd'hui, j'ai utilise `slice()` et `concat()` :

```
tabJourEnOrdre = joursSemaine.slice(indexOf(jourActuel)).concat(joursSemaine.slice(
indexOf(jourActuel)))
```

Cette technique reorganise le tableau sans modifier l'original.

2.4.4 Gestion des images conditionnelles

Le choix entre icones de jour et de nuit se fait selon l'heure actuelle :

```
if(heureActuelle >= 6 && heureActuelle < 21){
    imIcon.src = `res/jour/${resultatAPI.current
                 .weather[0].icon}.svg`;
} else {
    imIcon.src = `res/nuit/${resultatAPI.current
                 .weather[0].icon}.svg`;
}
```

Listing 2.5 – Selection des icones

2.5 Bonus : Animation de chargement

L'ajout d'une animation de chargement ameliore l'experience utilisateur pendant la recuperation des donnees.

```
.overlay img.char {
    position: absolute;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
}
```

Listing 2.6 – Positionnement centre de l'animation

2.6 Competences acquises

1. **Requetes HTTP** : Utilisation de fetch() et gestion des Promises
2. **APIs externes** : Interaction avec OpenWeatherMap API
3. **CSS Grid** : Creation de mises en page complexes
4. **Manipulation de dates** : Travail avec l'objet Date en JavaScript
5. **UX** : Amelioration de l'experience avec des animations de chargement

Chapitre 3

To-Do List

3.1 Introduction au projet

La To-Do List est une application de gestion de taches qui permet d'ajouter, cocher et supprimer des elements. Ce projet m'a permis de consolider mes connaissances en manipulation du DOM.

3.2 Structure HTML minimaliste

Le HTML de ce projet est volontairement simple car la majorite du contenu est genere dynamiquement.

```
<form>
    <label for="item">Enter une action</label>
    <input id="item" type='text'
           placeholder="nouvelle action"/>
    <button type="submit">Ajouter la tache</button>
</form>

<ul class="liste"></ul>
```

Listing 3.1 – Structure de base

3.3 CSS et experience utilisateur

3.3.1 Effet de focus personnalise

J'ai appris a utiliser la pseudo-classe `:focus` pour ameliorer l'experience utilisateur lors de la saisie.

```
form input:focus {
    box-shadow: 0 0 2px 2px rgba(255, 255, 255, .4);
}
```

Listing 3.2 – Effet de halo sur focus

3.3.2 Hover et transitions

Les effets de survol sur les boutons utilisent des transitions fluides :

```
form button {
    transition: .5s;
}

form button:hover {
    background: transparent;
    color: #f1f1f1;
    cursor: pointer;
    border: 1px solid #f1f1f1;
    box-shadow: 0 0 2px 2px rgba(255, 255, 255, .4);
}
```

Listing 3.3 – Transition sur hover

3.3.3 Style des taches terminees

La classe `.finDeTache` applique un style barre aux taches completees :

```
li.finDeTache span {
    text-decoration: line-through;
}
```

Listing 3.4 – Style des taches terminees

3.4 JavaScript : Gestion dynamique des taches

3.4.1 Structure des donnees

Chaque tache est un objet avec deux proprietes :

```
const todo = {
    text,
    id: Date.now()
}
```

Listing 3.5 – Structure d'une tache

Point important

L'utilisation de `Date.now()` comme identifiant unique est une solution simple et efficace pour de petites applications.

3.4.2 Ajout de taches

La fonction `rajouterTache()` cree l'objet et actualise l'affichage :

```
function rajouterTache(text){
    const todo = {
        text,
        id: Date.now()
    }
    afficherListe(todo);
```

```
}
```

Listing 3.6 – Ajout d'une tache

3.4.3 Creation dynamique d'elements HTML

La fonction `afficherListe()` cree tous les elements necessaires pour chaque tache :

```
function afficherListe(todo){
    const item = document.createElement('li');
    item.setAttribute('data-key', todo.id);

    const input = document.createElement('input');
    input.setAttribute('type', 'checkbox');
    input.addEventListener('click', tacheFaite);
    item.appendChild(input);

    const txt = document.createElement('span');
    txt.innerText = todo.text;
    item.appendChild(txt);

    const btn = document.createElement('button');
    btn.addEventListener('click', supprimerTache);
    const img = document.createElement('img');
    img.setAttribute('src', 'fermer.svg');
    btn.appendChild(img);
    item.appendChild(btn);

    liste.appendChild(item);
    toutesLesTaches.push(item);
}
```

Listing 3.7 – Creation d'elements DOM

3.4.4 Gestion des etats

La fonction `tacheFaite()` utilise `toggle()` pour ajouter ou retirer la classe :

```
function tacheFaite(e){
    e.target.parentNode.classList.toggle('finDeTache');
}
```

Listing 3.8 – Basculement d'etat

3.4.5 Suppression de taches

La suppression necessite deux operations : retirer l'element du DOM et du tableau :

```
function supprimerTache(e){
    toutesLesTaches.forEach(el => {
        if(e.target.parentNode.getAttribute('data-key')
           === el.getAttribute('data-key')){
            el.remove();
        }
    })

    toutesLesTaches = toutesLesTaches.filter(li =>
```

```
10          li.dataset.key !== el.dataset.key);
11  }
```

Listing 3.9 – Suppression d'une tache

3.5 Lecons apprises

1. **Attributs data-*** : Utiles pour stocker des metadonnees
2. **createElement()** : Creation d'elements HTML
3. **toggle()** : Basculement de classe
4. **filter()** : Filtrage de tableaux
5. **forEach()** : Iteration sur les tableaux

Chapitre 4

Pokedex

4.1 Contexte du projet

Le Pokedex est une application qui affiche des cartes de Pokemon avec leurs images et informations. Les donnees proviennent de l'API PokeAPI, et les cartes s'affichent progressivement lors du defilement.

4.2 Structure HTML

Le HTML contient un formulaire de recherche et un conteneur pour les cartes :

```
<h1>Bienvenue dans le pokedex</h1>

<form class="recherche-poke">
    <label for="recherche">Chercher un pokemon</label>
    <input type="text" name="recherche"
           id="recherche">
    <button type="submit">Rechercher</button>
</form>

<div class="container">
    <ul class="liste-poke"></ul>
</div>
```

Listing 4.1 – Structure du Pokedex

4.3 CSS : Label integre dans l'input

Une technique CSS interessante consiste a positionner le label a l'interieur du champ de saisie.

```
.recherche-poke label {
    position: absolute;
    color: #333;
    top: 50%;
    transform: translateY(-50%);
    padding-left: 8px;
    transition: all 0.3s ease-in;
```

```
}
```

Listing 4.2 – Positionnement du label

```
.recherche-poke:focus-within label,
.recherche-poke.active-input label {
    top: -15px;
    padding-left: 5px;
    color: #f1f1f1;
}
```

Listing 4.3 – Animation du label au focus

Decouverte CSS

La pseudo-classe `:focus-within` s'applique a un element parent lorsqu'un de ses enfants a le focus.

4.4 JavaScript : Appels API multiples

4.4.1 Gestion des Promises

Ce projet m'a fait decouvrir `Promise.all()` pour gerer plusieurs appels API simultanes.

```
function fetchPokemonBase(){
    const promises = [];

    fetch(`https://pokeapi.co/api/v2/pokemon?limit=${limite}`)
        .then(response => response.json())
        .then((allPoke) => {
            allPoke.results.forEach((pokemon) => {
                promises.push(
                    fetchPokemonComplet(pokemon)
                        .catch(error =>
                            console.error(error))
                );
            })
        })
        .then(() => {
            Promise.all(promises).then(() => {
                tableauFin = allPokemon.sort((a, b) => {
                    return a.id - b.id;
                }).slice(0, PokeNombreDebut);

                createCard(tableauFin);
                chargement.style.display= 'none';
            })
        })
}
```

Listing 4.4 – Recuperation des Pokemon

4.4.2 Appels API en cascade

Pour chaque Pokemon, trois appels API sont necessaires.

```
function fetchPokemonComplet(pokemon){
    let objPokemonFull = {};
    let url = pokemon.url;

    return fetch(url)
        .then(response => response.json())
        .then((pokeData) => {
            counter++;
            objPokemonFull.pic =
                pokeData.sprites.front_default;
            objPokemonFull.type =
                pokeData.types[0].type.name;
            objPokemonFull.id = pokeData.id;

            return fetch(`https://pokeapi.co/api/v2/
                        pokemon-species/${nameP}`)
                .then(response => response.json())
                .then((pokeData) => {
                    objPokemonFull.name =
                        pokeData.names[4].name;
                    allPokemon.push(objPokemonFull);
                })
        });
}
```

Listing 4.5 – Recuperation des details

4.4.3 Creation dynamique des cartes

La fonction `createCard()` genere les elements HTML pour chaque Pokemon.

```
function createCard(arr){
    for(let i = 0; i < arr.length; i++){
        const carte = document.createElement("li");
        carte.classList.add('hoverableCarte');
        let couleur = types[arr[i].type];
        carte.style.background = couleur;

        const txtCarte = document.createElement('h5');
        txtCarte.innerText = arr[i].name;

        const idCarte = document.createElement('p');
        idCarte.innerText = `ID# ${arr[i].id}`;

        const imgCarte = document.createElement('img');
        imgCarte.src = arr[i].pic;

        carte.appendChild(imgCarte);
        carte.appendChild(txtCarte);
        carte.appendChild(idCarte);

        listePoke.appendChild(carte);
    }
}
```

Listing 4.6 – Creation des cartes

4.4.4 Scroll infini

Le chargement progressif des cartes utilise l'evenement `scroll`.

```
window.addEventListener('scroll', () => {
    const {scrollTop, scrollHeight, clientHeight} =
        document.documentElement;

    if(clientHeight + scrollTop >= scrollHeight - 20){
        addPoke(6);
    }
})
```

Listing 4.7 – Detection du scroll

4.4.5 Fonction de recherche

La recherche utilise `indexOf()` pour trouver les correspondances.

```
function recherche(){
    let filter, allLi, titleValue, allTitles;
    filter = searchInput.value.toUpperCase();
    allLi = document.querySelectorAll('li');
    allTitles = document.querySelectorAll
                ("li > h5");

    for(i= 0; i< allLi.length; i++){
        titleValue = allTitles[i].innerText;
        if(titleValue.toUpperCase()
                    .indexOf(filter) > -1){
            allLi[i].style.display= "flex";
        } else {
            allLi[i].style.display= "none";
        }
    }
}
```

Listing 4.8 – Recherche de Pokemon

4.5 Animation de chargement

L'ajout d'une animation CSS ameliore l'experience pendant le chargement.

```
.point {
    width: 20px;
    height: 20px;
    border-radius: 50%;
    background: #f1f1f1;
    animation: monte 0.3s alternate infinite;
}

.point:nth-child(1){ animation-delay: 0.1s; }
.point:nth-child(2){ animation-delay: 0.2s; }
.point:nth-child(3){ animation-delay: 0.3s; }

@keyframes monte{
    100% { transform: translateY(-20px); }
```

```
15 }
```

Listing 4.9 – Animation des points

4.6 Connaissances acquises

1. **Promise.all()** : Gestion de multiples requetes asynchrones
2. **Scroll event** : Detection de la position de defilement
3. **Destructuring** : Extraction de proprietes d'objets
4. **indexOf()** : Recherche dans des chaines de caracteres
5. **Animation-delay** : Decalage d'animations CSS

Chapitre 5

Timer Pomodoro

5.1 Principe de l'application

Le Pomodoro est une technique de gestion du temps qui alterne des periodes de travail de 25 minutes avec des pauses de 5 minutes.

5.2 HTML simplifie

La structure HTML est minimaliste car les contenus sont geres en JavaScript.

```
<h1>Pomodoro</h1>

<div class="affichage">
    <div class="bloc_travail">
        <p>Travail</p>
        <p class="affichageT"></p>
    </div>
    <div class="bloc_pause">
        <p>Repos</p>
        <p class="affichageP"></p>
    </div>
</div>

<div class="container-btns">
    <button class="btn b1">Commencer</button>
    <button class="btn b2">Pause</button>
    <button class="btn b3">Reinitialiser</button>
</div>

<h2></h2>
```

Listing 5.1 – Structure du Pomodoro

5.3 CSS : Police LCD personnalisee

Une police externe `Digital-7` donne un aspect d'ecran LCD au timer.

```
@font-face {
    font-family: 'Digital-7';
    src: url('fonts/Digital-7Italic.woff2')
```

```
         format('woff2'),
         url('fonts/Digital-7Italic.woff')
         format('woff');
    font-weight: normal;
    font-style: italic;
    font-display: swap;
}

.affichageT, .affichageP {
    font-family: Digital-7;
    text-shadow: 0px 0px 8px var(--lcd);
}
```

Listing 5.2 – Import de police personnalisee

5.4 JavaScript : Gestion du temps

5.4.1 Variables de configuration

Les durees sont definies en secondes.

```
const T_INIT = 1500;  // 25 minutes
const T_REP = 300;    // 5 minutes

let tempsInitial = T_INIT;
let tempsDeRepos = T_REP;
let pause = false;
let nbDeCycles = 0;
let ChronoEnMarche = false;
```

Listing 5.3 – Configuration des durees

5.4.2 Affichage du temps

La conversion secondes → minutes :secondes utilise `Math.trunc()` et l'operateur ternaire.

```
affichageTravail.innerText =
    `${Math.trunc(tempsInitial/60)} : \
    ${tempsInitial % 60 < 10 ? '0' +
      (tempsInitial % 60) : tempsInitial % 60}`;
```

Listing 5.4 – Formatage du temps

5.4.3 setInterval() : Le cœur du timer

`setInterval()` execute une fonction a intervalles reguliers.

```
btnGo.addEventListener('click', () => {
    if(!ChronoEnMarche){
        ChronoEnMarche = true;

        if(tempsInitial > 0){
            tempsInitial--;
        }
        affichageTravail.innerText =
```

```
            `${Math.trunc(tempsInitial/60)} : \
            ${tempsInitial % 60 < 10 ? '0' +
              (tempsInitial % 60) :
              tempsInitial % 60}`;

        let timer = setInterval(() => {
            if(!pause && tempsInitial > 0){
                tempsInitial--;
                affichageTravail.innerText =
                    `${Math.trunc(tempsInitial/60)} : \
                    ${tempsInitial % 60 < 10 ? '0' +
                      (tempsInitial % 60) :
                      tempsInitial % 60}`;
            }
        }, 1000);
    }
})
```

Listing 5.5 – Demarrage du chronometre

5.4.4 Gestion des cycles

Le timer gere automatiquement l'alternance entre travail et repos.

```
if(!pause && tempsDeRepos === 0 &&
   tempsInitial === 0){
    tempsInitial = T_INIT;
    tempsDeRepos = T_REP;
    nbDeCycles++;
    cycles.innerText =
        `Nombre de cycles : ${nbDeCycles}`;

    affichageTravail.innerText =
        `${Math.trunc(tempsInitial/60)} : \
        ${tempsInitial % 60 < 10 ? '0' +
          (tempsInitial % 60) :
          tempsInitial % 60}`;
    affichagePause.innerText =
        `${Math.trunc(tempsDeRepos/60)} : \
        ${tempsDeRepos % 60 < 10 ? '0' +
          (tempsDeRepos % 60) :
          tempsDeRepos % 60}`;
}
```

Listing 5.6 – Alternance travail/repos

5.4.5 Fonction pause

La pause utilise une variable booleenne qui inverse sa valeur.

```
btnPause.addEventListener('click', () => {
    if(ChronoEnMarche){
        if(pause){
            btnPause.innerText = "Pause";
        } else {
            btnPause.innerText = "Reprendre";
        }
```

```
        pause = !pause;
    }
})
```

Listing 5.7 – Gestion de la pause

5.4.6 Reinitialisation

Le bouton reset arrete le timer et reinitialise toutes les valeurs.

```
btnReset.addEventListener('click', () => {
    clearInterval(timer);
    ChronoEnMarche = false;
    pause = false;
    tempsInitial = T_INIT;
    tempsDeRepos = T_REP;
    btnPause.innerText = "Pause";
    nbDeCycles = 0;
    cycles.innerText = `Nombre de cycles :
                        ${nbDeCycles}`;

    affichageTravail.innerText =
        `${Math.trunc(tempsInitial/60)} : \
        ${tempsInitial % 60 < 10 ? '0' +
          (tempsInitial % 60) :
          tempsInitial % 60}`;
})
```

Listing 5.8 – Reinitialisation

5.5 Points cles appris

1. **setInterval()** : Execution periodique d'une fonction
2. **clearInterval()** : Arret d'un intervalle
3. **Math.trunc()** : Arrondi a l'unite
4. **Operateur ternaire** : Condition concise
5. **Variables booleennes** : Gestion d'etats

Chapitre 6

Gestion des Cookies

6.1 Presentation du projet

Le projet *Cookies Creator* est une application web interactive permettant de creer, afficher et supprimer des cookies navigateur. Ce projet m'a permis de comprendre en profondeur le fonctionnement des cookies HTTP, leur manipulation en JavaScript et l'importance de la gestion du temps et des encodages.

6.2 Structure HTML

Le HTML est simple et fonctionnel, avec trois champs de saisie et deux boutons d'action.

```
<h1>Le cr ateur de cookies</h1>
<input type="text" name="cookieName" value="cookie-01" />
<input type="text" name="cookieValue" value="50" />
<input type="date" name="cookieExpire" />

<div class="container-btns">
    <button data-cookie="creer" class="btn1">Cr er</button>
    <button data-cookie="toutAfficher" class="btn2">Afficher</button>
</div>

<p class="info-txt"></p>
<ul class="affichage"></ul>
```

Listing 6.1 – Structure HTML de l'application Cookies

Points cles HTML

- Utilisation d'attributs `data-cookie` pour identifier les actions
- Champ `date` pour la date d'expiration
- Zone d'affichage dynamique (`<ul>`) pour les resultats
- Zone de texte pour les messages d'information

6.3 Mise en forme CSS

Le CSS de ce projet est particulierement soigne, avec des animations subtiles et des effets de focus.

6.3.1 Effet de survol et animations

Le premier bouton possede une animation *idle* qui le fait trembler legerement, attirant l'oeil de l'utilisateur.

```
button:nth-child(1) {
    animation: idle 2.5s alternate infinite;
}

@keyframes idle {
    0% { transform: matrix(1, 0, 0, 1, 0, 0); }
    93% { transform: matrix(1, 0, 0, 1, 0, 0); }
    95% { transform: matrix(1.03, -0.03, 0.03, 1.03, 0, 0); }
    97% { transform: matrix(1.03, 0.03, -0.03, 1.03, 0, 0); }
}
```

Listing 6.2 – Animation idle sur le bouton Creer

6.3.2 Effet de rebond au survol

J'ai decouvert l'utilisation de `transform: scale()` pour creer un effet de rebond subtil.

```
@keyframes bounce {
    50% { transform: scale(1.03); }
    100% { transform: scale(1.01); }
}

button:hover {
    animation: bounce .3s;
}
```

Listing 6.3 – Animation bounce au survol

6.3.3 Style des inputs

Les champs de saisie ont un style sombre coherent avec le theme, et un effet de halo au focus.

```
input:focus {
    box-shadow: 0px 0px 2px 2px rgba(255, 255, 255, .4);
}
```

Listing 6.4 – Effet de focus

6.4 JavaScript : Cœur de l'application

C'est dans ce projet que j'ai vraiment appris a manipuler les cookies en JavaScript.

6.4.1 Initialisation de la date

Une astuce interessante : la date d'expiration est automatiquement initialisee a J+7.

```
const today = new Date();
const nextWeek = new Date(today.getTime() + 7 * 24 * 60 * 60 * 1000);

let day = ('0' + nextWeek).slice(9, 11);
let month = (('0' + (today.getMonth() + 1)).slice(-2));
let year = today.getFullYear();

document.querySelector('input[type=date]').value =
`${year}-${month}-${day}`;
```

Listing 6.5 – Initialisation de la date par defaut

6.4.2 Gestion evenementielle

Un seul gestionnaire d'evenement pour les deux boutons, grace a l'attribut `data-cookie`.

```
btns.forEach(btn => {
    btn.addEventListener('click', btnAction);
})

function btnAction(e) {
    let nvObj = {};
    input.forEach(input => {
        let attrName = input.getAttribute('name');
        let attrValeur =
        attrName !== "cookieExpire" ? input.value : input.valueAsDate;
        nvObj[attrName] = attrValeur;
    })

    let description = e.target.getAttribute('data-cookie');

    if(description === 'creer') {
        creerCookie(nvObj.cookieName,
            nvObj.cookieValue, nvObj.cookieExpire)
    }else if(description === "toutAfficher") {
        listeCookies();
    }
}
```

Listing 6.6 – Gestionnaire d'evenement unique

Ma comprehension

J'ai compris l'interet de separer la collecte des donnees de l'action a effectuer. L'objet `nvObj` centralise les valeurs des inputs, ce qui rend le code plus lisible et facile a maintenir.

6.4.3 Creation d'un cookie

La fonction `creerCookie()` verifie d'abord si le nom existe deja, puis cree le cookie.

```
function creerCookie(name, value, exp) {
    // Verification de l'unicite du nom
```

```
    let cookies = document.cookie.split(';');
    cookies.forEach(cookie => {
        cookie = cookie.trim();
        let formatCookie = cookie.split('=');
        if(formatCookie[0] === encodeURIComponent(name)) {
            dejaFait = true;
        }
    })

    if(dejaFait) {
        infoTxt.innerText = "Le nom est deja utilise";
        dejaFait = false;
        return;
    }

    if(name.length === 0) {
        infoTxt.innerText = "Impossible de creer un cookie sans nom.";
        return;
    }

    // Creation du cookie
    document.cookie = `${encodeURIComponent(name)}=\
    ${encodeURIComponent(value)};expires=${exp.toUTCString()}`;

    // Feedback utilisateur
    let info = document.createElement('li');
    info.innerText = `Cookie ${name} cree.`;
    affichage.appendChild(info);
    setTimeout(() => {
        info.remove();
    }, 1500);
}
```

Listing 6.7 – Creation d'un cookie avec verification

6.4.4 Affichage et suppression

La fonction `listeCookies()` affiche tous les cookies et attache un gestionnaire d'evenement a chaque element pour permettre la suppression au clic.

```
function listeCookies() {
    let cookies = document.cookie.split(';');
    if(cookies.join() === '') {
        infoTxt.innerText = 'La liste de cookies est vide';
        return;
    }
    affichage.innerText = "";

    cookies.forEach(cookie => {
        cookie = cookie.trim();
        let formatCookie = cookie.split('=');

        infoTxt.innerText = 'Cliquer sur le nom pour supprimer.'

        let item = document.createElement('li');
        item.innerText = `Nom : ${decodeURIComponent(formatCookie[0])},\
        Valeur : ${decodeURIComponent(formatCookie[1])}`;
        affichage.appendChild(item);
```

```
19
20          item.addEventListener('click', ()=> {
21              document.cookie = `${formatCookie[0]}=; expires=${new Date
    (0)}`;
22              item.innerText = `Cookie ${formatCookie[0]} supprime.`;
23              setTimeout(() => {
24                  item.remove();
25              }, 1000);
26          })
27      })
28  }
```

Listing 6.8 – Affichage et suppression interactive

6.4.5 Encodage et decodage

Une decouverte importante : l'utilisation de `encodeURIComponent()` et `decodeURIComponent()`.

Astuce importante

Les cookies ne peuvent pas contenir de caracteres speciaux (espaces, accents, symboles). `encodeURIComponent()` convertit ces caracteres en un format compatible, et `decodeURIComponent()` les retraduit lors de l'affichage.

6.5 Apprentissages cles

1. **Cookies HTTP** : Compréhension du stockage cote client et du format `nom=valeur; expires=date`
2. **Encodage** : Necessite d'encoder les noms et valeurs avec `encodeURIComponent()`
3. **Gestion du temps** : Manipulation de l'objet `Date` et formatage pour l'expiration
4. **Attributs data-*** : Utilisation pour differencier les actions sans creer plusieurs gestionnaires
5. **Feedback utilisateur** : Messages temporaires avec `setTimeout()` et suppression automatique
6. **Suppression** : Invalidation d'un cookie en fixant une date d'expiration passee
7. **Animations CSS** : Creation d'effets subtils (idle, bounce) avec `@keyframes`

6.6 Difficultes rencontrees

6.6.1 Gestion du format de date

J'ai eu du mal a formater correctement la date pour l'input `type=date`. La solution a ete d'utiliser `slice()` et de construire manuellement la chaine `YYYY-MM-DD`.

6.6.2 Verification de l'unicite

Au debut, je ne comprenais pas pourquoi la verification echouait. J'ai realise qu'il fallait encoder le nom avant de le comparer avec les cookies existants.

6.6.3 Suppression des cookies

J'ai appris qu'il ne suffit pas de *supprimer* un cookie, il faut le *remplacer* avec une date d'expiration passee.

6.7 Conclusion sur le projet

Ce projet a ete une excellente introduction a la gestion d'etat cote client. Les cookies sont un mecanisme simple mais puissant, et leur manipulation m'a fait comprendre des concepts fondamentaux du web : les en-tetes HTTP, la gestion des dates, et l'importance de l'encodage.

Conclusion

Synthese des apprentissages

Au terme de cette formation et de l'analyse de ces six projets, j'ai acquis une comprehension solide des technologies web fondamentales.

Competences HTML

J'ai appris a structurer des documents de maniere semantique, utiliser les formulaires, organiser le contenu et lier des ressources externes.

Maitrise du CSS

Les projets m'ont permis de decouvrir CSS Grid, Flexbox, les transitions, animations, pseudo-classes, variables CSS et l'import de polices.

JavaScript

Ma progression a ete significative en manipulation du DOM, gestion des evenements, programmation asynchrone (fetch, Promises, Promise.all()), manipulation de tableaux et objets, et gestion des cookies.

Methodologie acquise

J'ai developpe une methodologie de travail : planification, incrementalite, separation des preoccupations, debugging et consultation de documentation.

Perspectives

Je souhaite approfondir LocalStorage, les frameworks (React, Vue), le backend avec Node.js, les fonctionnalites ES6+ et le testing.

Muckaya Florida RAMALALAHASITERA
L1B N°331
13 février 2026

Annexe A

Ressources et references

A.1 Documentation consultee

— **MDN Web Docs** : https://developer.mozilla.org
— **CSS-Tricks** : https://css-tricks.com
— **OpenWeatherMap API** : https://openweathermap.org/api
— **PokeAPI** : https://pokeapi.co
— **Documentation Cookies** : https://developer.mozilla.org/fr/docs/Web/HTTP/Cookies

A.2 Outils utilises

— **Editeur de code** : Visual Studio Code
— **Navigateur** : Chrome avec DevTools
— **Controle de version** : Git
— **Design** : Figma

Annexe B

Glossaire

API Application Programming Interface

Asynchrone Code qui s'execute sans bloquer le reste du programme

Cookie Petit fichier stocke cote client contenant des informations d'etat

DOM Document Object Model

Fetch Fonction JavaScript pour les requetes HTTP

Grid Systeme de mise en page CSS bidimensionnel

Promise Objet representant une valeur future asynchrone

Responsive Design adaptable

URI Encoding Encodage des caracteres speciaux dans une URL