Rapport Technique JavaScript

Application TrackWatch - Système de Suivi Cinématographique

Réalisé par :

Chahid Fatima Ezzahraa

Jilal Saad

Hajoub Anas

Encadré par : Mme Khadija Tlemcani

Table des matières

Problématique

Comment créer une application web monopage qui permet aux utilisateurs de gérer leur consommation cinématographique de manière organisée et personnalisée, avec les fonctionnalités suivantes :

- Navigation fluide entre différentes sections sans rechargement de page
- Intégration avec l'API TMDB pour accéder à une vaste base de données
- Persistance des données localement (listes de favoris, films visionnés)
- Interface utilisateur réactive avec mode clair/sombre
- Gestion de contenu personnalisé (ajout manuel de films)
- Visualisation statistique des préférences

Introduction du Projet

TrackWatch est une application web développée en JavaScript vanilla qui permet aux utilisateurs de suivre, organiser et découvrir des films et séries. L'application se connecte à l'API TMDB pour récupérer les données et utilise le stockage local du navigateur pour sauvegarder les préférences utilisateur. L'architecture est basée sur une approche monopage (SPA) où différentes sections de l'application sont affichées/masquées dynamiquement.

Parties Essentielles du Code JavaScript

Configuration Initiale et Variables Globales

```
// Configuration de l'API TMDB
const options = {
  method: 'GET',
  headers: {
    accept: 'application/json',
    Authorization: 'Bearer [TOKEN_API]'
  }
};

// S lection des    lments    DOM principaux
const hamburgerPress = document.querySelector('.hamburger-pess');
const homepage = document.querySelector('.homepage');
const movies = document.querySelector(".movies");
const mode = document.querySelector(".mode");
const navbar = document.querySelector(".navbar");

// Variables pour les diff rentes pages
const actorspage = document.querySelector(".actorspage");
const searchthing = document.querySelector(".searchthing");
const moviedeats = document.querySelector(".movie-details");
const watchedpage = document.querySelector(".watched-page");
const favpage = document.querySelector(".fav-page");
const statscontainer = document.querySelector('.stats-container');
const addmoviestuff = document.querySelector("#add-movie-modal");

// Variables pour les fonctionnalit s
let myChart = null; // Pour le graphique Chart.js
```

Explication : Cette section initialise les configurations essentielles, notamment les paramètres pour l'API TMDB et la sélection des principaux éléments DOM. La structure utilise des variables globales pour gérer l'état de l'application.

Gestion de la Navigation et Changement de Pages

```
// Gestion du menu hamburger
hamburger.addEventListener("mouseover", function(){
    hamburgerPress.style.display='block';
});
hamburgerPress.addEventListener("mouseout", function(){
    hamburgerPress.style.display='none';
});

// Navigation entre les pages
homebutton.forEach(el => {
    el.addEventListener("click",function(){
    pageload(1);
    homepage.style.display="block";
    mainstuff.style.display="block";
    mainpage.style.display="none";
    actorspage.style.display="none";
    searchthing.style.display="none";
    // ... masquer les autres pages
});
});

// Mode clair/sombre
mode.addEventListener("click",function(){
    document.body.classList.toggle("othermode");
    navbar.classList.toggle("othermode");
    hamburgerPress.classList.toggle("othermode");
});
```

Explication : Ce code gère la navigation principale de l'application. Il utilise des événements pour afficher/masquer différentes sections et implémente un système de mode clair/sombre via le toggle de classes CSS.

Chargement des Films avec Pagination

```
const pageload = function(p){
    let currentpage = p;
    movies.innerHTML = "";
    pages.innerHTML = "";

    // Chargement des films personnalis s depuis localStorage
    const addedmv = JSON.parse(localStorage.getItem("addedmovies")) ||
        [];
    if (p == 1) {
        addedmv.forEach(movie => {
            movies.innerHTML += `
                <div class="mvelement custom-item" data-id="${movie.id
                    }">
                    <button class="del-btn" data-id="${movie.id}">X</
                        button>
```

```
                    <div class="mvpic">
                        <img src="${movie.poster_path}">
                    </div>
                    <div class="mvname"><p>${movie.title}</p></div>
                </div>`;
        });
    }

    // Récupération des films populaires depuis l'API
    fetch(`https://api.themoviedb.org/3/movie/popular?language=en-US&page=${currentpage}`, options)
    .then(response => response.json())
    .then(d => {
        d.results.forEach(movie => {
            movies.innerHTML += `
            <div class="mvelement apistuff">
                <div class="mvpic">
                    <img src="https://image.tmdb.org/t/p/w500${movie.poster_path}">
                </div>
                <div class="mvname"><p>${movie.title}</p></div>
            </div>`;
        });

        // Gestion des événements de clic sur les films
        const actorsstuff = document.querySelectorAll(".apistuff");
        actorsstuff.forEach((el, index) => {
            el.addEventListener("click", function(){
                d.results[index].media_type = "movie";
                movieinfo(d.results[index]);
            });
        });

        // Pagination
        const lastpage = d.total_pages;
        for(let i = currentpage - 3; i < currentpage; ++i){
            if (i < 1) continue;
            pages.innerHTML += `<button class="page-number">${i}</button>`;
        }
        for(let i = currentpage; i < currentpage + 4; ++i){
            if (i == lastpage) continue;
            pages.innerHTML += `<button class="page-number">${i}</button>`;
        }
    })
    .catch(error => console.error(error));
};
```

Explication : Cette fonction gère le chargement des films avec pagination. Elle combine à la fois les films personnalisés (stockés en local) et les films de l'API TMDB. La pagination est calculée dynamiquement en fonction du nombre total de pages.

Gestion des Détails d'un Film

```
const movieinfo = function(p){
    mediaside.innerHTML = "";
```

```
    // Configuration de l'affichage de la page de d tails
    homepage.style.display = "none";
    moviedeats.style.display = "block";
    // ... masquer les autres pages

    const type = p.media_type || (p.title ? "movie" : "tv");

    // R cup ration des d tails complets du film
    fetch(`https://api.themoviedb.org/3/${type}/${p.id}?
        append_to_response=videos,credits&language=en-US`, options)
    .then(res => res.json())
    .then(d => {
        // Recherche de la bande-annonce
        const trailer = d.videos.results.find(v => v.type === "Trailer")
            ;
        const trailerKey = trailer ? trailer.key : "";

        // Construction de l'interface de d tails
        mediaside.innerHTML += `
            <div class="media-side">
                <div class="detail-pic">
                    <img id="info-img" src="https://image.tmdb.org/t/p/
                        w500${d.poster_path}">
                </div>
                <div class="detail-trailer">
                    <a href="https://www.youtube.com/watch?v=${
                        trailerKey}">Trailer</a>
                </div>
            </div>
            <!-- ... reste du HTML ... -->
        `;

        // Gestion des boutons "Marquer comme vu" et "Ajouter aux
            favoris"
        const markwatched = document.querySelector("#mark-watched");
        const markfav = document.querySelector("#addfav");

        markwatched.addEventListener("click", function(){
            const watchedlist = localStorage.getItem("mywatched");
            let watchedlisttab = watchedlist ? JSON.parse(watchedlist) :
                [];
            if(!watchedlisttab.some(m => m.id == p.id)){
                watchedlisttab.push({
                    id: p.id,
                    type: p.media_type,
                    genres: d.genres.map(g => g.name)
                });
                localStorage.setItem("mywatched", JSON.stringify(
                    watchedlisttab));
                alert("saved to your watchedlist");
            }
        });

        markfav.addEventListener("click", function(){
            // Logique similaire pour les favoris
        });
    })
    .catch(error => console.error(error));
```

```
};
```

Explication : Cette fonction affiche les détails complets d'un film, incluant sa bande-annonce, sa description, les acteurs principaux, et permet de l'ajouter aux listes "Vus" ou "Favoris". Elle gère également la persistance des données dans le localStorage.

Gestion des Films Visionnés et Favoris

```
const showmoviepage = function(){
    watchedmoviescontainer.innerHTML = "";
    const storage = localStorage.getItem("mywatched");

    if (!storage){
        watchedtitle.innerHTML = "no shows watched yet";
    } else {
        let moviecount = 0;
        JSON.parse(storage).forEach(el => {
            if (el.isCustom) {
                // Affichage des films personnalis s
                watchedmoviescontainer.innerHTML += `
                    <div class="mvelement">
                        <button class="delete-watch-btn" data-id="${el.
                            id}">X</button>
                        <div class="mvpic">
                            <img src="${el.poster_path || ''}">
                        </div>
                        <div class="mvname"><p>${el.title}</p></div>
                    </div>`;
                moviecount++;
            } else {
                // R cup ration des d tails depuis l'API pour les
                    films TMDB
                fetch(`https://api.themoviedb.org/3/${el.type}/${el.id}?
                    language=en-US`, options)
                .then(res => res.json())
                .then(d => {
                    watchedmoviescontainer.innerHTML += `
                        <div class="mvelement">
                            <button class="delete-watch-btn" data-id="${
                                el.id}">X</button>
                            <div class="mvpic">
                                <img src="https://image.tmdb.org/t/p/
                                    w500${d.poster_path}">
                            </div>
                            <div class="mvname"><p>${d.title || d.name
                                }</p></div>
                        </div>`;
                    moviecount++;

                    // Ajout des vnements de suppression
                    if(moviecount == JSON.parse(localStorage.getItem("
                        mywatched")).length){
                        const dels = document.querySelectorAll(`.delete-
                            watch-btn`);
                        dels.forEach(del => {
                            del.addEventListener("click", function(){
```

```
                                const idToDel = this.getAttribute("data-id");
                                let currentwatched = JSON.parse(localStorage.getItem("mywatched"));
                                currentwatched = currentwatched.filter(m => m.id != idToDel);
                                localStorage.setItem("mywatched", JSON.stringify(currentwatched));
                                showmoviepage();
                            });
                        });
                    }
                });
            }
        });
    }
};
```

Explication : Cette fonction gère l'affichage des films visionnés, en distinguant les films personnalisés de ceux provenant de l'API TMDB. Elle inclut également la fonction- nalité de suppression individuelle avec mise à jour en temps réel du localStorage.

Système de Recherche Avancée

```
const search = function(searchname, pagenmber, type){
    let currentResults = [];
    if (!type) { type = "multi" }
    let currentpage = pagenmber;
    searchpagination.innerHTML = " ";

    fetch(`https://api.themoviedb.org/3/search/${type}?query=${searchname}&include_adult=true&language=en-US&page=${currentpage}`, options)
    .then(res => res.json())
    .then(d => {
        currentResults = d.results;
        display(currentResults);

        // Pagination pour la recherche
        const lastpage = d.total_pages;
        for(let i = currentpage - 3; i < currentpage; ++i) {
            if (i < 1) { continue; }
            searchpagination.innerHTML += `<button class="searchpagination-number">${i}</button>`;
        }

        //    Tri    des    r    s u l t a t s
        sortrating.addEventListener("click", function(){
            const sorted = [...currentResults].sort((a, b) =>
                (b.vote_average || 0) - (a.vote_average || 0)
            );
            display(sorted);
        });

        sortdate.addEventListener("click", function(){
            const sorted = [...currentResults].sort((a, b) => {
```

```
                const dateA = new Date(a.release_date || a.
                    first_air_date || 0);
                const dateB = new Date(b.release_date || b.
                    first_air_date || 0);
                return dateB - dateA;
            });
            display(sorted);
        });
    })
    .catch(error => console.error(error));
};
```

Explication : Le système de recherche permet de chercher des films, séries et personnes avec pagination et options de tri (par note, date, ordre alphabétique). Il utilise l'endpoint "multi" de l'API TMDB pour rechercher dans différents types de contenu.

Ajout et Gestion de Films Personnalisés

```
const saveNewMovie = function() {
    let custmovies = JSON.parse(localStorage.getItem("addedmovies")) ||
        [];

    const newMovie = {
        id: Date.now().toString(),
        title: document.querySelector("#custom-name").value.trim(),
        overview: document.querySelector("#custom-desc").value,
        genres: [document.querySelector("#custom-genre").value],
        release_date: document.querySelector("#custom-date").value,
        vote_average: document.querySelector("#custom-rating").value ||
            0,
        vote_count: document.querySelector("#custom-votes").value || 0,
        poster_path: null,
        isCustom: true
    };

    custmovies.push(newMovie);
    localStorage.setItem("addedmovies", JSON.stringify(custmovies));
    alert("Movie added!");
    pageload(1);
};

const deleteMe = function(id) {
    let movies = JSON.parse(localStorage.getItem("addedmovies")) || [];
    movies = movies.filter(m => m.id !== id);
    localStorage.setItem("addedmovies", JSON.stringify(movies));
    pageload(1);
    homepage.style.display = "block";
    mainstuff.style.display = "block";
};
```

Explication : Cette section permet aux utilisateurs d'ajouter leurs propres films à la base de données locale. Chaque film personnalisé reçoit un ID unique basé sur le timestamp et est stocké dans le localStorage avec un flag isCustom: true pour le distinguer des films de l'API.

Visualisation des Statistiques avec Chart.js

```
const kpithing = function(){
    const favlist = JSON.parse(localStorage.getItem("myfav")) || [];
    if (favlist.length == 0) { return; }

    count   =   {};
    favlist.forEach(el => {
        el.genres.forEach(g => {
            count[g] = (count[g] || 0) + 1;
        });
    });

    const genres = Object.keys(count);
    const genresocc = Object.values(count);

    const doughnut = document.querySelector('#genreChart').getContext('2
       d');
    if (myChart) { myChart.destroy(); }

    myChart = new Chart(doughnut, {
        type: 'doughnut',
        data: {
            labels: genres,
            datasets: [{
                data: genresocc,
                backgroundColor: ['#8205ef', '#9b6dca', '#42146a',
                   '#7440a2', '#bf73d6'],
                hoverOffset: 15,
                borderWidth: 1,
                borderColor: '#000'
            }]
        },
        options: {
            responsive: true,
            plugins: {
                legend: {
                    position: 'right',
                    labels: { color: 'white' }
                }
            }
        }
    });
};
```

Explication : Cette fonction génère un graphique en beignet (doughnut chart) qui montre la répartition des genres dans les films favoris de l'utilisateur. Elle analyse les données du localStorage et utilise Chart.js pour la visualisation.

Conclusion

L'application TrackWatch démontre une utilisation avancée de JavaScript vanilla pour créer une application web complète sans frameworks externes. Les principaux enseignements techniques sont :

— **Gestion d'état complexe** : L'application maintient plusieurs états (page active, données utilisateur, filtres) sans bibliothèque dédiée
— **Communication asynchrone** : Utilisation intensive de l'API Fetch pour interagir avec TMDB
— **Persistance des données** : Implémentation robuste du localStorage pour sauvegarder les préférences
— **Architecture monopage** : Navigation fluide via manipulation du DOM plutôt que rechargements de page
— **UI réactive** : Gestion des événements utilisateur et mise à jour dynamique de l'interface

Le code présente une bonne séparation des préoccupations avec des fonctions dédiées à chaque fonctionnalité. Les améliorations potentielles pourraient inclure l'ajout d'un système de routage plus structuré, une meilleure gestion des erreurs et l'optimisation des performances pour les grandes collections de films.