ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR
Membre de HONORIS UNITED UNIVERSITIES

HONORIS UNITED UNIVERSITIES

Rapport de Projet

CineTech Dashboard

Smart Backoffice Dashboard

Elaboré par :

-Doubla Wissam
-Choukri Khadija
-Mohammed Amine Naiem

Encadré par :

-Mme.Tlemçani

Filière 3IIR
Module : Développement Web

Année universitaire 2025 – 2026

Remerciements

Je tiens à remercier tout particulièrement mon encadrant , Mme.Tlemçani, pour son soutien et ses précieux conseils tout au long de ce projet.
Je remercie également mes camarades , pour leur collaboration et leur aide durant le développement du CineTech Dashboard. Enfin, je remercie l'ensemble du corps enseignant et l'institution pour m'avoir fourni les ressources et l'encadrement nécessaires à la réalisation de ce projet.

Table des matières

1 Introduction

Dans le cadre de notre projet, nous avons conçu une application web intitulée **CineTech Dashboard**. Ce projet avait pour objectif de créer une plateforme de type backoffice permettant de gérer efficacement les données liées au cinéma, notamment les films et les réalisateurs, tout en offrant une interface moderne pour les utilisateurs.

Ce travail nous a permis de mettre en pratique les notions acquises durant notre formation, notamment en HTML5, CSS3 et JavaScript Vanilla. Nous avons appris à structurer une application web, à gérer les interactions dynamiques via JavaScript, et à organiser différentes sections telles que la gestion des films, la gestion des réalisateurs et l'affichage des indicateurs de performance (KPI).

Ce projet nous a également permis de développer notre sens de la créativité, notre capacité à collaborer en équipe. Grâce à CineTech Dashboard, nous avons pu consolider nos compétences techniques tout en concevant une interface professionnelle et fonctionnelle.

2 Analyse du sujet CineTech

Le sujet CineTech vise à concevoir un système de gestion cinématographique similaire à ceux utilisés par les plateformes de streaming et les bases de données culturelles.

2.1 Objectifs du projet

— Gérer une liste de films (CRUD complet)
— Gérer une liste de réalisateurs (CRUD simplifié)
— Afficher des KPI cinéma
— Visualiser des statistiques sous forme de graphiques
— Récupérer des données via une API externe

2.2 Contraintes techniques

— Utilisation exclusive de JavaScript Vanilla
— Architecture Single Page Application
— Persistance des données via LocalStorage

3 Conception de l'application

3.1 Architecture SPA

L'application **CineTech Dashboard** est développée sous forme de **Single Page Application (SPA)**, ce qui permet une navigation fluide. Les caractéristiques principales de cette architecture sont :

— Un seul fichier HTML contenant la structure globale de l'application.
— Plusieurs sections affichées ou masquées dynamiquement via JavaScript.
— Navigation interactive permettant de passer d'une section à l'autre sans recharger la page.

3.2 Structure générale de l'interface

L'interface de l'application est organisée de manière à offrir une expérience utilisateur claire :

— **Sidebar verticale :** Permet la navigation entre les différentes sections de l'application.
— **Navbar supérieure :** Affiche le titre de la section active et des informations générales.
— **Dashboard :** Vue analytique offrant des indicateurs clés de performance (KPI).
— **Module Films :** Gestion complète des films.
— **Module Réalisateurs :** Gestion simplifiée des réalisateurs.
— **Section Statistiques & API :** Visualisation de données et exploitation d'une API externe.

3.3 Technologies utilisées

Pour développer CineTech Dashboard, nous avons utilisé les technologies suivantes :

— **HTML5** : utilisé pour structurer le contenu de l'application (titres, paragraphes, images, formulaires, etc.).Grâce à HTML, l'ensemble de la structure du site est organisé de manière hiérarchique et sémantique.
— **CSS3 / Bootstrap** :CSS est utilisé pour la mise en forme et le design du site. Il permet de personnaliser l'apparence des éléments HTML : couleurs, polices, tailles, marges, etc. Le CSS rend l'interface plus attractive et agréable à utiliser pour les visiteurs.

— **JavaScript Vanilla** : utilisé pour rendre l'application interactive, gérer le CRUD des films et réalisateurs, et afficher/masquer les sections de la SPA.
— **Chart.js** : bibliothèque pour créer des graphiques et visualiser les statistiques et KPI du tableau de bord.
— **FontAwesome** : bibliothèque d'icônes pour améliorer la lisibilité et l'interface utilisateur.
— **LocalStorage** : API du navigateur pour stocker localement les données des films et réalisateurs côté client.

4 Extraits De Code

4.1 init() – Point d'entrée de l'application

```
// ===== FONCTION D'INITIALISATION =====
// Point d'entrée principal, appelé au chargement du DOM
function init() {
    // Récupération des éléments DOM dans le cache
    cacheElements();

    // Chargement des données d'exemple si la base est vide
    if (films.length === 0) {
        initializeSampleData();
    }

    // Configuration des événements utilisateur
    initEventListeners();

    // Initialisation des graphiques Chart.js
    initCharts();

    // Premier rendu de l'interface
    updateDashboard();
    renderFilms();
    renderDirectors();

    // Message de bienvenue avec délai pour meilleure UX
    setTimeout(() => {
        showToast('Tableau de bord CineTech chargé avec succès !', 'success');
    }, 1000);
}
```

FIGURE 1 – Initialisation complète de l'application et affichage des données

La fonction init() démarre l'application dès que le DOM est prêt. Elle commence par récupérer les éléments du DOM avec cacheElements(), puis charge des données d'exemple avec initializeSampleData() si nécessaire. Elle initialise les écouteurs d'événements via initEventListeners(), crée les graphiques avec initCharts(), met à jour le tableau de bord avec updateDashboard(), et affiche les listes de films et de réalisateurs avec renderFilms() et renderDirectors(). Enfin, après une courte pause, une notification confirme que l'application est prête.

4.2 cacheElements() – Optimisation des accès DOM

```
// ===== CACHE DES ÉLÉMENTS DOM =====
// Stocke les références aux éléments DOM pour éviter les requêtes répétées
function cacheElements() {
    elements.sidebarLinks = document.querySelectorAll('#sidebar a');
    elements.contentSections = document.querySelectorAll('.content-section');
    elements.navbarToggle = document.getElementById('navbar-toggle');
    elements.currentSection = document.getElementById('current-section');
    elements.breadcrumb = document.querySelector('.breadcrumb');
    elements.notificationBtn = document.getElementById('notification-btn');
    elements.notificationDropdown = document.getElementById('notification-dropdown');
    elements.loadingOverlay = document.getElementById('loading-overlay');
    elements.toastContainer = document.getElementById('toast-container');
}
```

FIGURE 2 – Stocke les références DOM pour rendre l'application plus rapide

Cette fonction enregistre les éléments importants de l'écran (boutons, menus, sections) pour éviter de les chercher plusieurs fois. Ça rend l'application plus rapide et fluide, surtout sur mobile.

4.3 initEventListeners() – Gestion des interactions utilisateur

```
// Attache tous les écouteurs d'événements aux éléments interactifs
function initEventListeners() {
    // Navigation dans la sidebar
    elements.sidebarLinks.forEach(link => {
        link.addEventListener('click', handleNavigation);
    });

    // Toggle de la sidebar sur mobile
    elements.navbarToggle.addEventListener('click', toggleSidebar);

    // Gestion des notifications
    if (elements.notificationBtn) {
        elements.notificationBtn.addEventListener('click', toggleNotifications);
    }

    // ÉVÉNEMENTS DES FILMS (Module 1 - CRUD Complet)
    document.getElementById('add-film-btn')?.addEventListener('click', () => openFilmModal());
    document.getElementById('film-search-btn')?.addEventListener('click', renderFilms);
    document.getElementById('film-search')?.addEventListener('input', renderFilms);
    document.getElementById('film-genre-filter')?.addEventListener('change', renderFilms);
    document.getElementById('film-sort')?.addEventListener('change', renderFilms);

    // Changement de vue (tableau/cartes/compact)
    document.getElementById('view-table-btn')?.addEventListener('click', () => switchView('table'));
    document.getElementById('view-cards-btn')?.addEventListener('click', () => switchView('cards'));
    document.getElementById('view-compact-btn')?.addEventListener('click', () => switchView('compact'));

    // ÉVÉNEMENTS DES RÉALISATEURS (Module 2 - CRUD Light)
    document.getElementById('add-director-btn')?.addEventListener('click', () => openDirectorModal());

    // ÉVÉNEMENTS API (Module 3 - Intégration externe)
    document.getElementById('api-search-btn')?.addEventListener('click', searchFilmsAPI);
    document.getElementById('api-search-input')?.addEventListener('keypress', (e) => {
        if (e.key === 'Enter') searchFilmsAPI();
    });
```

```
// ÉVÉNEMENTS DES MODALES
document.getElementById('save-film-btn')?.addEventListener('click', saveFilm);
document.getElementById('save-director-btn')?.addEventListener('click', saveDirector);
document.getElementById('confirm-delete-btn')?.addEventListener('click', () => {
    if (deleteCallback) deleteCallback();
});
document.getElementById('confirm-clear-btn')?.addEventListener('click', clearAllData);
document.getElementById('edit-film-from-detail')?.addEventListener('click', editFilmFromDetail);

// GESTION DES DONNÉES (Import/Export)
document.getElementById('export-data-btn')?.addEventListener('click', exportData);
document.getElementById('import-data-btn')?.addEventListener('click', triggerImport);

// Écouteur pour l'input fichier d'import
const importInput = document.getElementById('import-data-file');
if (importInput) {
    importInput.addEventListener('change', handleFileImport);
}

// RECHERCHE GLOBALE
document.getElementById('global-search')?.addEventListener('input', handleGlobalSearch);

// DASHBOARD
document.getElementById('refresh-dashboard')?.addEventListener('click', refreshDashboard);

// SÉLECTION MULTIPLE DE FILMS
document.getElementById('select-all-films')?.addEventListener('change', toggleSelectAllFilms);
document.getElementById('delete-selected-films')?.addEventListener('click', deleteSelectedFilms);

// PARAMÈTRES D'APPARENCE
document.querySelectorAll('.theme-option').forEach(option => {
    option.addEventListener('click', handleThemeChange);
});
```

FIGURE 3 – Configure tous les événements pour rendre l'application interactive.

La fonction initEventListeners() configure tous les événements de l'interface. Elle gère la navigation (sidebar, bouton mobile), les actions sur les films (ajout, recherche, filtre, tri, changement de vue) et sur les réalisateurs (ajout et interactions). Elle configure aussi les modales, l'intégration API pour l'import et la recherche de films, ainsi que les fonctionnalités avancées comme l'import/export, la recherche globale et les paramètres d'apparence. Cette centralisation rend l'application interactive et facilite la maintenance.

4.4 handleNavigation(e) – Gestion de la navigation SPA

```
// ===== FONCTIONS DE NAVIGATION =====

// Gère le clic sur les liens de navigation de la sidebar
function handleNavigation(e) {
    e.preventDefault();

    const link = e.currentTarget;
    const targetId = link.id.replace('-link', '-section');
    const sectionName = link.querySelector('span').textContent;

    // Met à jour l'élément actif dans la sidebar
    elements.sidebarLinks.forEach(l => l.classList.remove('active'));
    link.classList.add('active');

    // Cache toutes les sections de contenu
    elements.contentSections.forEach(section => {
        section.classList.remove('active');
    });
```

```
    // Affiche la section cible avec animation
    const targetSection = document.getElementById(targetId);
    if (targetSection) {
        targetSection.classList.add('active');
        elements.currentSection.textContent = sectionName;

        // Met à jour le fil d'Ariane
        updateBreadcrumb(sectionName);

        // Actions spécifiques selon la section
        switch(targetId) {
            case 'dashboard-section':
                updateDashboard();      // Rafraîchit les KPI et graphiques
                break;
            case 'stats-section':
                updateStatsCharts();    // Met à jour les graphiques statistiques
                break;
            case 'films-section':
                renderFilms();          // Affiche la liste des films
                break;
            case 'directors-section':
                renderDirectors();      // Affiche la liste des réalisateurs
                break;
        }
    }

    // Ferme la sidebar sur mobile après sélection
    if (window.innerWidth < 992) {
        document.getElementById('sidebar').classList.remove('active');
    }
}
```

FIGURE 4 – Gère la navigation et l'affichage des sections dans l'application.

Cette fonction gère les clics sur la sidebar et met à jour l'interface en SPA. Elle marque le lien actif, cache les sections inutiles, affiche la section cible avec animation, met à jour le titre et le fil d'Ariane, et déclenche les actions spécifiques à chaque section. Sur mobile, elle ferme la sidebar pour optimiser l'affichage.

4.5 renderFilms() – Affichage des films

```
// ===== MODULE 1 : GESTION DES FILMS (CRUD COMPLET) =====

// Fonction principale de rendu des films avec filtrage et tri
function renderFilms() {
    // Récupère les critères de filtrage depuis l'interface
    const searchTerm = document.getElementById('film-search')?.value.toLowerCase() || '';
    const genreFilter = document.getElementById('film-genre-filter')?.value || '';
    const sortOption = document.getElementById('film-sort')?.value || 'title-asc';

    // Filtre les films selon les critères
    let filteredFilms = films.filter(film => {
        const filmTitle = film.title.toLowerCase();
        const filmDirector = getDirectorById(film.directorId)?.name.toLowerCase() || '';

        // Vérifie la correspondance avec la recherche
        const matchesSearch = !searchTerm ||
            filmTitle.includes(searchTerm) ||
            filmDirector.includes(searchTerm) ||
            film.year.toString().includes(searchTerm) ||
            film.genre.toLowerCase().includes(searchTerm);

        // Vérifie le filtre de genre
        const matchesGenre = !genreFilter || film.genre === genreFilter;

        return matchesSearch && matchesGenre;
    });
```

```
    // Trie les films selon l'option sélectionnée
    filteredFilms.sort((a, b) => {
        switch (sortOption) {
            case 'title-asc': return a.title.localeCompare(b.title);
            case 'title-desc': return b.title.localeCompare(a.title);
            case 'year-asc': return a.year - b.year;
            case 'year-desc': return b.year - a.year;
            case 'rating-desc': return b.rating - a.rating;
            case 'rating-asc': return a.rating - b.rating;
            default: return 0;
        }
    });

    // Met à jour le compteur de résultats
    const filmsCount = document.getElementById('films-count');
    if (filmsCount) {
        filmsCount.textContent = filteredFilms.length;
    }

    // Affiche selon la vue active
    switch(currentView) {
        case 'table':
            renderFilmsTableView(filteredFilms);
            break;
        case 'cards':
            renderFilmsCardsView(filteredFilms);
            break;
        case 'compact':
            renderFilmsCompactView(filteredFilms);
            break;
    }
```

FIGURE 5 – Affiche les films avec filtrage, tri et différents modes d'affichage.

Cette fonction affiche les films selon les critères de recherche, de filtre et de tri. Elle applique d'abord la recherche texte puis le filtre de genre, trie les résultats selon l'option choisie, met à jour le compteur,

et rend la liste selon le mode sélectionné (tableau, cartes, compact).

4.6 saveFilm() – Création et modification des films

```
// Sauvegarde un film (ajout ou mise à jour)
function saveFilm() {
    const form = document.getElementById('film-form');

    // Validation du formulaire
    if (!form.checkValidity()) {
        form.classList.add('was-validated');
        return;
    }

    // Récupération des données du formulaire
    const filmId = parseInt(document.getElementById('film-id').value);
    const posterUrl = document.getElementById('film-poster').value.trim();

    const filmData = {
        id: filmId || getNextId(films),
        title: document.getElementById('film-title').value.trim(),
        directorId: parseInt(document.getElementById('film-director').value),
        year: parseInt(document.getElementById('film-year').value),
        genre: document.getElementById('film-genre').value,
        duration: parseInt(document.getElementById('film-duration').value),
        rating: parseFloat(document.getElementById('film-rating').value),
        poster: posterUrl || 'https://via.placeholder.com/300x450?text=Poster+non+disponible',
        synopsis: document.getElementById('film-synopsis').value.trim(),
        createdAt: new Date().toISOString()
    };

    // Affiche l'indicateur de chargement
    const saveBtn = document.getElementById('save-film-btn');
    const spinner = document.getElementById('film-saving-spinner');
    const saveText = document.getElementById('save-film-text');

    saveBtn.disabled = true;
    spinner.classList.remove('d-none');
    saveText.textContent = 'Enregistrement...';
```

```
// Simule un délai pour l'UX (simulation d'appel API)
setTimeout(() => {
    if (filmId) {
        // Mise à jour d'un film existant
        const index = films.findIndex(f => f.id === filmId);
        if (index !== -1) {
            filmData.createdAt = films[index].createdAt; // Garde la date de création originale
            films[index] = filmData;
        }
    } else {
        // Ajout d'un nouveau film
        films.push(filmData);
    }

    // Persiste les données et met à jour l'interface
    saveDataToLocalStorage();
    renderFilms();
    updateDashboard();

    // Ferme la modal
    bootstrap.Modal.getInstance(document.getElementById('filmModal')).hide();

    // Réinitialise le formulaire
    form.reset();
    form.classList.remove('was-validated');

    // Réinitialise le bouton
    saveBtn.disabled = false;
    spinner.classList.add('d-none');
    saveText.textContent = 'Enregistrer';

    // Affiche une notification
    showToast(`Film "${filmData.title}" ${filmId ? 'modifié' : 'ajouté'} avec succès !`, 'success');
}, 800);
```

FIGURE 6 – Crée ou modifie un film avec validation et mise à jour de l'interface.

Cette fonction gère l'ajout ou la modification d'un film. Elle valide le formulaire, crée ou met à jour l'objet film, applique des transformations, affiche un indicateur de chargement, met à jour le tableau et le localStorage, réinitialise le formulaire, ferme la modal et notifie l'utilisateur du succès.

4.7 deleteSelectedFilms() – Suppression de films

```
// Supprime les films sélectionnés
function deleteSelectedFilms() {
    if (selectedFilms.size === 0) {
        showToast('Aucun film sélectionné', 'warning');
        return;
    }

    confirmDelete('films', Array.from(selectedFilms), `${selectedFilms.size} films`, () => {
        films = films.filter(film => !selectedFilms.has(film.id));
        selectedFilms.clear();
        saveDataToLocalStorage();
        renderFilms();
        updateDashboard();
        showToast(`${selectedFilms.size} film(s) supprimé(s) avec succès !`, 'success');
    });
}
```

FIGURE 7 – Supprime un ou plusieurs films et met à jour l'interface.

Ces fonctions permettent de supprimer un ou plusieurs films de la base locale. deleteFilm(filmId) supprime un film précis, met à jour le localStorage, l'affichage des films et le tableau de bord, et affiche une notification de succès. deleteSelectedFilms() supprime tous les films sélectionnés, réinitialise la sélection, actualise l'interface et notifie l'utilisateur du nombre de films supprimés.

4.8 confirmDelete(type, id, name, callback) – Confirmation de suppression

```javascript
// Affiche une modal de confirmation avant suppression
function confirmDelete(type, id, name, callback) {
    const modal = new bootstrap.Modal(document.getElementById('deleteConfirmModal'));
    const textElement = document.getElementById('delete-confirm-text');

    let message = '';
    if (type === 'film') {
        message = `Êtes-vous sûr de vouloir supprimer le film "${name}" ? Cette action est irréversible.`;
    } else if (type === 'director') {
        const directorFilms = films.filter(f => f.directorId === id);
        if (directorFilms.length > 0) {
            message = `Attention : ce réalisateur a ${directorFilms.length} film(s) associé(s). Êtes-vous sûr de voulo
        } else {
            message = `Êtes-vous sûr de vouloir supprimer le réalisateur "${name}" ? Cette action est irréversible.`;
        }
    } else if (type === 'films') {
        message = `Êtes-vous sûr de vouloir supprimer ${name} ? Cette action est irréversible.`;
    }

    textElement.textContent = message;

    // Stocke le callback à exécuter après confirmation
    deleteCallback = () => {
        callback();
        modal.hide();
        deleteCallback = null;
    };

    modal.show();
}
```

FIGURE 8 – Affiche une confirmation avant suppression et exécute un callback.

Cette fonction affiche une modal de confirmation avant toute suppression, avec un message adapté au type d'élément et aux dépendances (ex. films liés à un réalisateur). Elle exécute un callback uniquement si l'utilisateur confirme, séparant interface et logique.

4.9 searchFilmsAPI() – Recherche dans l'API OMDB

```javascript
// Recherche des films via l'API OMDB
async function searchFilmsAPI() {
    const searchInput = document.getElementById('api-search-input').value.trim();
    const resultsContainer = document.getElementById('api-results-container');
    const resultsCount = document.getElementById('api-results-count');

    if (!searchInput) {
        showToast('Veuillez saisir un terme de recherche', 'warning');
        return;
    }

    // Affiche l'indicateur de chargement
    showLoading();
    resultsContainer.innerHTML = `
        <div class="text-center py-5">
            <div class="spinner-border text-primary mb-3"></div>
            <p>Recherche en cours...</p>
        </div>
    `;

    try {
        // Utilisation de l'API OMDB avec clé publique de test
        const apiKey = 'apikey=thewdb'; // Clé publique pour démonstration
        const response = await fetch(`https://www.omdbapi.com/?s=${encodeURIComponent(searchInput)}&${apiKey}`);

        if (!response.ok) throw new Error('Erreur réseau');

        const data = await response.json();

        if (data.Response === 'True') {
            // Récupère les détails complets pour chaque film
            const filmDetailsPromises = data.Search.slice(0, 8).map(async (item) => {
                const detailResponse = await fetch(`https://www.omdbapi.com/?i=${item.imdbID}&${apiKey}`);
                return await detailResponse.json();
            });
```

```
            const filmDetails = await Promise.all(filmDetailsPromises);
            displayAPIResults(filmDetails);
            resultsCount.textContent = `${filmDetails.length} résultat(s)`;
        } else {
            // Aucun résultat
            resultsContainer.innerHTML = `
                <div class="empty-state">
                    <i class="fas fa-film fa-3x text-muted mb-3"></i>
                    <h5>Aucun résultat</h5>
                    <p class="text-muted">Aucun film trouvé pour "${searchInput}"</p>
                </div>
            `;
            resultsCount.textContent = '0 résultat(s)';
        }
    } catch (error) {
        console.error('Erreur API:', error);
        resultsContainer.innerHTML = `
            <div class="empty-state">
                <i class="fas fa-exclamation-triangle fa-3x text-danger mb-3"></i>
                <h5>Erreur de connexion</h5>
                <p class="text-muted">Impossible de se connecter à l'API. Vérifiez votre connexion Internet.</p>
            </div>
        `;
        resultsCount.textContent = 'Erreur';
        showToast('Erreur lors de la recherche API', 'error');
    } finally {
        hideLoading();
    }
}
```

FIGURE 9 – Recherche des films via l'API OMDB et affiche les résultats.

Cette fonction interroge l'API OMDB de manière asynchrone. Elle valide le terme de recherche, affiche un loader, récupère les résultats (limités à 8), traite les détails de chaque film en parallèle, et les affiche avec un bouton d'ajout. Elle gère aussi les erreurs réseau ou API et les résultats vides.

4.10 updateDashboard() – Mise à jour du dashboard

```
// Met à jour toutes les données du dashboard
function updateDashboard() {
    // Met à jour les KPI (Key Performance Indicators)
    document.getElementById('kpi-films').textContent = films.length;
    document.getElementById('kpi-directors').textContent = directors.length;

    // Calcule la note moyenne des films
    const avgRating = films.length > 0
        ? (films.reduce((sum, film) => sum + film.rating, 0) / films.length).toFixed(1)
        : '0.0';
    document.getElementById('kpi-avg-rating').textContent = avgRating;

    // Calcule la durée totale en heures
    const totalMinutes = films.reduce((sum, film) => sum + film.duration, 0);
    const totalHours = Math.round(totalMinutes / 60);
    document.getElementById('kpi-total-duration').textContent = totalHours;

    // Met à jour la liste des films récents
    updateRecentFilms();

    // Met à jour les graphiques du dashboard
    updateDashboardCharts();
}
```

FIGURE 10 – Met à jour les indicateurs et les composants du dashboard.

Cette fonction calcule et affiche les KPI (nombre de films, réalisateurs, note moyenne, durée totale), met à jour la liste des films récents et actualise les graphiques pour refléter les dernières données. Elle assure que le tableau de bord reste synchronisé après chaque modification.

4.11 saveDataToLocalStorage() – Persistance des données

```
// ===== GESTION DE LA PERSISTANCE DES DONNÉES =====

// Sauvegarde les données dans le localStorage du navigateur
function saveDataToLocalStorage() {
    localStorage.setItem('cineTechFilms', JSON.stringify(films));
    localStorage.setItem('cineTechDirectors', JSON.stringify(directors));

    // Met à jour les statistiques d'utilisation du stockage
    updateStorageStats();
}
```

FIGURE 11 – Sauvegarde l'état de l'application dans le stockage local.

Cette fonction sauvegarde les tableaux films et directors dans le localStorage en JSON, avec gestion d'erreurs (quota dépassé) et mise à jour visuelle de l'utilisation du stockage, pour conserver les données entre les sessions.

4.12 showToast(message, type) – Notifications utilisateur

```
// Affiche une notification toast (popup temporaire)
function showToast(message, type = 'info') {
    const toast = document.createElement('div');
    toast.className = `toast ${type}`;

    const icons = {
        success: 'fa-check-circle',
        error: 'fa-exclamation-circle',
        warning: 'fa-exclamation-triangle',
        info: 'fa-info-circle'
    };

    toast.innerHTML = `
        <i class="fas ${icons[type] || icons.info}"></i>
        <div class="toast-content">
            <div class="toast-title">${type.charAt(0).toUpperCase() + type.slice(1)}</div>
            <div class="toast-message">${message}</div>
        </div>
    `;

    elements.toastContainer.appendChild(toast);

    // Animation d'entrée
    setTimeout(() => toast.classList.add('show'), 10);

    // Auto-destruction après 5 secondes
    setTimeout(() => {
        toast.classList.remove('show');
        setTimeout(() => {
            if (toast.parentNode) {
                toast.parentNode.removeChild(toast);
            }
        }, 300);
    }, 5000);
}
```

FIGURE 12 – Affiche des notifications temporaires pour l'utilisateur.

Cette fonction crée une notification toast avec icône et couleur selon le type (succès, erreur, info, avertissement), l'affiche avec animation, la supprime automatiquement après 5 secondes et permet une fermeture manuelle.

4.13 openFilmModal(filmId) – Modal de création/édition de film

```
// Ouvre la modal d'ajout/édition de film
function openFilmModal(filmId = null) {
    const modalTitle = document.getElementById('filmModalTitle');
    const form = document.getElementById('film-form');
    const directorSelect = document.getElementById('film-director');

    // Réinitialise le formulaire
    form.reset();
    form.classList.remove('was-validated');
    document.getElementById('film-id').value = '';

    // Remplit la liste déroulante des réalisateurs
    directorSelect.innerHTML = '<option value="">Sélectionner un réalisateur...</option>';
    directors.forEach(director => {
        const option = document.createElement('option');
        option.value = director.id;
        option.textContent = director.name;
        directorSelect.appendChild(option);
    });

    if (filmId) {
        // Mode édition - préremplit le formulaire
        modalTitle.textContent = 'Modifier le film';
        const film = films.find(f => f.id === filmId);
```

```
if (filmId) {
    // Mode édition - préremplit le formulaire
    modalTitle.textContent = 'Modifier le film';
    const film = films.find(f => f.id === filmId);

    if (film) {
        document.getElementById('film-id').value = film.id;
        document.getElementById('film-title').value = film.title;
        document.getElementById('film-director').value = film.directorId;
        document.getElementById('film-year').value = film.year;
        document.getElementById('film-genre').value = film.genre;
        document.getElementById('film-duration').value = film.duration;
        document.getElementById('film-rating').value = film.rating;
        document.getElementById('film-poster').value = film.poster !== 'https://via.placeholder.co
        document.getElementById('film-synopsis').value = film.synopsis || '';
    }
} else {
    // Mode ajout - formulaire vide
    modalTitle.textContent = 'Ajouter un film';
}

// Affiche la modal avec Bootstrap
const modal = new bootstrap.Modal(document.getElementById('filmModal'));
modal.show();
```

FIGURE 13 – Ouvre la modal pour ajouter ou éditer un film.

Cette fonction prépare et ouvre la modal pour ajouter ou modifier un film. Elle réinitialise le formulaire, remplit les options de réalisateurs, préremplit les champs si le film existe, adapte le titre et affiche la modal Bootstrap.

4.14 addFilmFromAPI(apiFilm) – Ajout d'un film depuis l'API

```javascript
// Ajoute un film depuis l'API à la base locale
function addFilmFromAPI(apiFilm) {
    // Cherche ou crée le réalisateur
    let director = directors.find(d => d.name === apiFilm.Director);
    let directorId;

    if (!director && apiFilm.Director !== 'N/A') {
        // Crée un nouveau réalisateur
        directorId = getNextId(directors);
        const newDirector = {
            id: directorId,
            name: apiFilm.Director,
            nationality: apiFilm.Country !== 'N/A' ? apiFilm.Country.split(', ')[0] : '',
            birthdate: '',
            bio: `Réalisateur de "${apiFilm.Title}"`
        };
        directors.push(newDirector);
        director = newDirector;
    } else if (director) {
        directorId = director.id;
    } else {
        directorId = 0; // Réalisateur inconnu
    }

    // Crée l'objet film pour la base locale
    const newFilm = {
        id: getNextId(films),
        title: apiFilm.Title,
        directorId: directorId,
        year: parseInt(apiFilm.Year) || new Date().getFullYear(),
        genre: apiFilm.Genre !== 'N/A' ? apiFilm.Genre.split(', ')[0] : 'Non spécifié',
        duration: apiFilm.Runtime !== 'N/A' ? parseInt(apiFilm.Runtime.split(' ')[0]) : 120,
        rating: apiFilm.imdbRating !== 'N/A' ? parseFloat(apiFilm.imdbRating) : 7.0,
        poster: apiFilm.Poster !== 'N/A' ? apiFilm.Poster : 'https://via.placeholder.com/300x450?text=Poster
        synopsis: apiFilm.Plot !== 'N/A' ? apiFilm.Plot : '',
        createdAt: new Date().toISOString()
    };

    // Ajoute à la base locale
    films.push(newFilm);
    saveDataToLocalStorage();

    // Met à jour l'interface
    renderFilms();
    renderDirectors();
    updateDashboard();

    showToast(`Film "${newFilm.title}" ajouté avec succès depuis l'API !`, 'success');

    // Redirige vers la section des films
    document.getElementById('films-link').click();
}
```

FIGURE 14 – Ajoute et normalise un film importé depuis l'API OMDB.

Cette fonction transforme un film provenant de l'API OMDB en format interne et l'ajoute à la base locale. Elle crée un nouveau réalisateur si nécessaire, normalise les données (année, genre, durée, note, affiche), met à jour le tableau local et toutes les vues, puis redirige l'utilisateur vers la section des films pour voir le résultat.

4.15 exportData() – Export des données

```javascript
// Exporte toutes les données au format JSON
function exportData() {
    const data = {
        films: films,
        directors: directors,
        exportedAt: new Date().toISOString(),
        version: '1.0'
    };

    const dataStr = JSON.stringify(data, null, 2);
    const dataBlob = new Blob([dataStr], { type: 'application/json' });

    // Crée un lien de téléchargement
    const link = document.createElement('a');
    link.href = URL.createObjectURL(dataBlob);
    link.download = `cineTech-backup-${new Date().toISOString().slice(0, 10)}.json`;
    document.body.appendChild(link);
    link.click();
    document.body.removeChild(link);

    showToast('Données exportées avec succès !', 'success');
}
```

FIGURE 15 – Exporte toutes les données de l'application en JSON

Cette fonction exporte toutes les données de l'application dans un fichier JSON téléchargeable. Elle inclut films, réalisateurs et métadonnées (date, version, taille), crée un fichier lisible, déclenche le téléchargement via un lien temporaire et affiche une notification de succès.

4.16 switchView(view) – Changement du mode d'affichage

```javascript
// Change le mode d'affichage des films
function switchView(view) {
    currentView = view;

    // Met à jour l'état des boutons de vue
    document.getElementById('view-table-btn')?.classList.toggle('active', view === 'table');
    document.getElementById('view-cards-btn')?.classList.toggle('active', view === 'cards');
    document.getElementById('view-compact-btn')?.classList.toggle('active', view === 'compact');

    // Affiche/masque les conteneurs de vue
    document.getElementById('films-table-view').style.display = view === 'table' ? 'block' : 'none';
    document.getElementById('films-cards-view').style.display = view === 'cards' ? 'block' : 'none';
    document.getElementById('films-compact-view').style.display = view === 'compact' ? 'block' : 'none';

    // Re-rend la vue si nécessaire (pas pour le tableau car déjà rendu)
    if (view !== 'table') {
        renderFilms();
    }
}
```

FIGURE 16 – Permet de changer le mode d'affichage des films.

Cette fonction change le mode d'affichage des films (tableau, cartes, compact). Elle met à jour l'état currentView, ajuste les boutons et conteneurs visibles, et déclenche un rendu complet seulement si nécessaire, assurant ainsi une interface fluide et performante.

5 Persistance et gestion des données

La persistance des données dans l'application **CineTech Dashboard** est assurée grâce à l'API **LocalStorage** du navigateur. Cette solution permet de conserver les données même après le rechargement

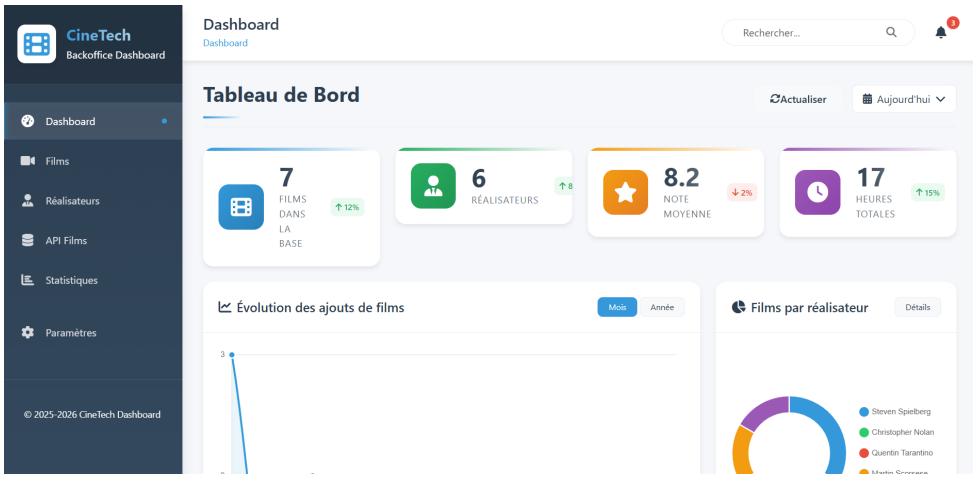

FIGURE 17 – interface utilisateur

ou la fermeture de la page.

Les données principales stockées sont :

— La liste des films

— La liste des réalisateurs

Ces données sont sérialisées au format JSON afin d'être facilement stockées et restaurées. À chaque modification (ajout, suppression ou mise à jour), le stockage local est automatiquement mis à jour, garantissant la cohérence de l'état de l'application.

Cette approche permet une gestion simple, rapide et efficace des données sans dépendre d'un serveur backend.

6 Interface utilisateur et expérience UX

L'interface utilisateur du CineTech Dashboard est conçue pour être simple,et facile à utiliser. L'application dispose d'une sidebar latérale permettant de naviguer rapidement entre les différentes sections : Dashboard, Films, Réalisateurs, API Films, Statistiques et Paramètres. Cette navigation se fait sans rechargement de page, offrant une expérience fluide.

7 Sécurité

Même si l'application fonctionne seulement sur le navigateur, elle a été conçue pour éviter les erreurs.

— Validation des formulaires avant toute sauvegarde

— Gestion des erreurs lors des appels API

— Confirmation obligatoire avant toute suppression

Ces mécanismes réduisent les risques d'erreurs utilisateur et assurent une meilleure stabilité de l'application.

8 Évolutions futures

Plusieurs améliorations pourraient être ajoutées pour rendre CineTech Dashboard encore meilleur :

— Ajout d'une authentification utilisateur

— Mise en place d'un backend avec une base de données réelle

— Synchronisation multi-utilisateurs

Ces évolutions permettraient de transformer l'application en une solution encore plus complète et professionnelle.

9 Conclusion et bilan

Le projet **CineTech Dashboard** nous a permis de developper une application web complète en JavaScript Vanilla, en appliquant les concepts de SPA, de gestion des données et d'expérience utilisateur.

Ce projet a renforcé nos compétences techniques en développement web. Il constitue une expérience enrichissante et représentative des problématiques rencontrées dans des projets web professionnels.

En conclusion, CineTech Dashboard répond pleinement aux objectifs fixés et constitue une base solide pour des évolutions futures.