// ========================================================================

// SERVICE WORKER POUR SUPPORT HORS LIGNE - PNGDI

// Création d'organisation - Support offline basic

// ========================================================================

const CACHE\_NAME = 'pngdi-v1.0.0';

const STATIC\_CACHE = 'pngdi-static-v1.0.0';

const DYNAMIC\_CACHE = 'pngdi-dynamic-v1.0.0';

// Ressources critiques à mettre en cache

const STATIC\_ASSETS = [

'/',

'/operator/dashboard',

'/operator/dossiers/create',

'/css/app.css',

'/js/app.js',

'https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css',

'https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js',

'https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0/css/all.min.css',

'https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js'

];

// URLs d'API à gérer en mode offline

const API\_URLS = [

'/api/v1/save-draft',

'/api/v1/load-draft',

'/api/v1/form-analytics'

];

// Installation du Service Worker

self.addEventListener('install', event => {

console.log('🔧 Service Worker: Installation');

event.waitUntil(

caches.open(STATIC\_CACHE)

.then(cache => {

console.log('📦 Service Worker: Mise en cache des ressources statiques');

return cache.addAll(STATIC\_ASSETS.map(url => {

return new Request(url, {

cache: 'reload'

});

}));

})

.catch(error => {

console.warn('⚠️ Service Worker: Erreur lors de la mise en cache:', error);

})

);

// Forcer l'activation immédiate

self.skipWaiting();

});

// Activation du Service Worker

self.addEventListener('activate', event => {

console.log('🚀 Service Worker: Activation');

event.waitUntil(

caches.keys()

.then(cacheNames => {

return Promise.all(

cacheNames.map(cacheName => {

// Supprimer les anciens caches

if (cacheName !== STATIC\_CACHE &&

cacheName !== DYNAMIC\_CACHE &&

cacheName !== CACHE\_NAME) {

console.log('🗑️ Service Worker: Suppression ancien cache:', cacheName);

return caches.delete(cacheName);

}

})

);

})

.then(() => {

console.log('✅ Service Worker: Nettoyage terminé');

return self.clients.claim();

})

);

});

// Interception des requêtes

self.addEventListener('fetch', event => {

const { request } = event;

const url = new URL(request.url);

// Ignorer les requêtes non-HTTP

if (!request.url.startsWith('http')) {

return;

}

// Stratégie selon le type de requête

if (isStaticAsset(request)) {

// Cache First pour les ressources statiques

event.respondWith(cacheFirst(request));

} else if (isApiRequest(request)) {

// Network First avec fallback pour les API

event.respondWith(networkFirstWithOfflineSupport(request));

} else {

// Stale While Revalidate pour les pages

event.respondWith(staleWhileRevalidate(request));

}

});

// Gestion de la synchronisation en arrière-plan

self.addEventListener('sync', event => {

console.log('🔄 Service Worker: Synchronisation arrière-plan:', event.tag);

if (event.tag === 'draft-sync') {

event.waitUntil(syncDrafts());

} else if (event.tag === 'analytics-sync') {

event.waitUntil(syncAnalytics());

}

});

// Gestion des messages du client

self.addEventListener('message', event => {

const { data } = event;

switch (data.type) {

case 'SKIP\_WAITING':

self.skipWaiting();

break;

case 'SAVE\_OFFLINE\_DRAFT':

saveOfflineDraft(data.payload);

break;

case 'GET\_OFFLINE\_DRAFTS':

getOfflineDrafts().then(drafts => {

event.ports[0].postMessage({ drafts });

});

break;

case 'CLEAR\_CACHE':

clearAllCaches().then(() => {

event.ports[0].postMessage({ success: true });

});

break;

}

});

// ========================================

// STRATÉGIES DE CACHE

// ========================================

/\*\*

\* Cache First - Priorité au cache

\*/

async function cacheFirst(request) {

try {

const cachedResponse = await caches.match(request);

if (cachedResponse) {

return cachedResponse;

}

const networkResponse = await fetch(request);

if (networkResponse.ok) {

const cache = await caches.open(STATIC\_CACHE);

cache.put(request, networkResponse.clone());

}

return networkResponse;

} catch (error) {

console.warn('❌ Cache First failed:', error);

return new Response('Ressource non disponible hors ligne', {

status: 503,

statusText: 'Service Unavailable'

});

}

}

/\*\*

\* Network First avec support offline

\*/

async function networkFirstWithOfflineSupport(request) {

try {

const networkResponse = await fetch(request);

// Mettre en cache les réponses API réussies

if (networkResponse.ok && request.method === 'GET') {

const cache = await caches.open(DYNAMIC\_CACHE);

cache.put(request, networkResponse.clone());

}

return networkResponse;

} catch (error) {

console.log('🔌 Mode hors ligne pour:', request.url);

// Gérer les requêtes offline spécifiques

if (request.method === 'POST' && isApiRequest(request)) {

return handleOfflineApiRequest(request);

}

// Essayer le cache pour les GET

const cachedResponse = await caches.match(request);

if (cachedResponse) {

return cachedResponse;

}

// Réponse par défaut pour mode offline

return new Response(JSON.stringify({

error: 'Mode hors ligne',

message: 'Cette action sera synchronisée lorsque vous serez en ligne',

offline: true

}), {

status: 200,

headers: {

'Content-Type': 'application/json'

}

});

}

}

/\*\*

\* Stale While Revalidate

\*/

async function staleWhileRevalidate(request) {

const cache = await caches.open(DYNAMIC\_CACHE);

const cachedResponse = await cache.match(request);

const fetchPromise = fetch(request).then(networkResponse => {

if (networkResponse.ok) {

cache.put(request, networkResponse.clone());

}

return networkResponse;

}).catch(() => cachedResponse);

return cachedResponse || fetchPromise;

}

// ========================================

// GESTION HORS LIGNE SPÉCIALISÉE

// ========================================

/\*\*

\* Gérer les requêtes API en mode hors ligne

\*/

async function handleOfflineApiRequest(request) {

const url = new URL(request.url);

const body = await request.text();

// Sauvegarder la requête pour sync ultérieure

if (url.pathname.includes('save-draft')) {

await saveOfflineData('drafts', {

url: request.url,

method: request.method,

body: body,

headers: Object.fromEntries(request.headers.entries()),

timestamp: Date.now()

});

return new Response(JSON.stringify({

success: true,

message: 'Brouillon sauvegardé localement',

offline: true,

draft\_id: 'offline\_' + Date.now()

}), {

status: 200,

headers: { 'Content-Type': 'application/json' }

});

}

if (url.pathname.includes('form-analytics')) {

await saveOfflineData('analytics', {

url: request.url,

method: request.method,

body: body,

headers: Object.fromEntries(request.headers.entries()),

timestamp: Date.now()

});

return new Response(JSON.stringify({

success: true,

message: 'Analytics enregistrées localement',

offline: true

}), {

status: 200,

headers: { 'Content-Type': 'application/json' }

});

}

// Réponse générique

return new Response(JSON.stringify({

error: 'Action non disponible hors ligne',

offline: true

}), {

status: 503,

headers: { 'Content-Type': 'application/json' }

});

}

/\*\*

\* Sauvegarder des données hors ligne

\*/

async function saveOfflineData(store, data) {

try {

const cache = await caches.open('offline-data');

const key = `${store}\_${Date.now()}\_${Math.random()}`;

await cache.put(

new Request(key),

new Response(JSON.stringify(data), {

headers: { 'Content-Type': 'application/json' }

})

);

console.log('💾 Données sauvegardées hors ligne:', store);

} catch (error) {

console.error('❌ Erreur sauvegarde hors ligne:', error);

}

}

/\*\*

\* Synchroniser les brouillons

\*/

async function syncDrafts() {

try {

const cache = await caches.open('offline-data');

const requests = await cache.keys();

for (const request of requests) {

if (request.url.includes('drafts\_')) {

const response = await cache.match(request);

const data = await response.json();

try {

// Tenter de renvoyer la requête

await fetch(data.url, {

method: data.method,

body: data.body,

headers: data.headers

});

// Supprimer du cache après succès

await cache.delete(request);

console.log('✅ Brouillon synchronisé et supprimé du cache');

} catch (error) {

console.warn('⚠️ Échec synchronisation brouillon:', error);

}

}

}

} catch (error) {

console.error('❌ Erreur synchronisation brouillons:', error);

}

}

/\*\*

\* Synchroniser les analytics

\*/

async function syncAnalytics() {

try {

const cache = await caches.open('offline-data');

const requests = await cache.keys();

for (const request of requests) {

if (request.url.includes('analytics\_')) {

const response = await cache.match(request);

const data = await response.json();

try {

await fetch(data.url, {

method: data.method,

body: data.body,

headers: data.headers

});

await cache.delete(request);

console.log('✅ Analytics synchronisées');

} catch (error) {

console.warn('⚠️ Échec synchronisation analytics:', error);

}

}

}

} catch (error) {

console.error('❌ Erreur synchronisation analytics:', error);

}

}

// ========================================

// UTILITAIRES

// ========================================

/\*\*

\* Vérifier si la requête concerne un asset statique

\*/

function isStaticAsset(request) {

const url = new URL(request.url);

return url.pathname.match(/\.(css|js|png|jpg|jpeg|gif|svg|ico|woff|woff2|ttf)$/i) ||

STATIC\_ASSETS.includes(request.url);

}

/\*\*

\* Vérifier si la requête concerne une API

\*/

function isApiRequest(request) {

const url = new URL(request.url);

return url.pathname.startsWith('/api/') ||

API\_URLS.some(apiUrl => url.pathname.includes(apiUrl));

}

/\*\*

\* Obtenir les brouillons hors ligne

\*/

async function getOfflineDrafts() {

try {

const cache = await caches.open('offline-data');

const requests = await cache.keys();

const drafts = [];

for (const request of requests) {

if (request.url.includes('drafts\_')) {

const response = await cache.match(request);

const data = await response.json();

drafts.push({

id: request.url,

data: data,

timestamp: data.timestamp

});

}

}

return drafts;

} catch (error) {

console.error('❌ Erreur récupération brouillons:', error);

return [];

}

}

/\*\*

\* Sauvegarder un brouillon hors ligne

\*/

async function saveOfflineDraft(payload) {

await saveOfflineData('drafts', {

...payload,

timestamp: Date.now()

});

}

/\*\*

\* Nettoyer tous les caches

\*/

async function clearAllCaches() {

try {

const cacheNames = await caches.keys();

await Promise.all(

cacheNames.map(cacheName => caches.delete(cacheName))

);

console.log('🗑️ Tous les caches supprimés');

} catch (error) {

console.error('❌ Erreur nettoyage caches:', error);

}

}

console.log('🔧 Service Worker PNGDI chargé et configuré');