

Activité - Moteur de recherche de jeux vidéo - Étape 5

Notre application est maintenant opérationnelle. Il ne nous reste plus qu'à en faire une « Progressive Web App » (PWA). Mais qu'est-ce qu'une PWA exactement ?

Comprendre les Progressive Web Apps

Une Progressive Web App est une application web qui peut être installée sur votre appareil (ordinateur, smartphone, tablette) et fonctionner comme une application native. Elle offre plusieurs avantages :

- **Installation** : L'application peut être installée directement depuis le navigateur
- **Hors-ligne** : Elle continue de fonctionner même sans connexion internet
- **Rapide** : Les ressources sont mises en cache pour un chargement plus rapide
- **Notifications** : Elle peut envoyer des notifications (si l'utilisateur l'autorise)
- **Mise à jour** : Elle se met à jour automatiquement

Pour transformer notre application React en PWA, nous avons besoin de trois éléments principaux :

1. Un fichier manifest qui décrit notre application
2. Un service worker qui gère le cache et le fonctionnement hors-ligne
3. Des icônes pour l'affichage sur l'écran d'accueil

Avec Vite, nous allons utiliser le plugin officiel vite-plugin-pwa qui simplifie grandement cette configuration.

Installation des dépendances

Commencez par ouvrir un terminal dans le dossier de votre projet et installez les dépendances nécessaires :

```
npm install -D vite-plugin-pwa workbox-window
```

💡 L'option -D (ou --save-dev) indique que ces packages sont des dépendances de développement, nécessaires uniquement pour construire l'application.

Configuration du fichier manifest

Le manifest est un fichier JSON qui décrit votre application au système d'exploitation. Il contient des informations comme :

- Le nom de l'application
- Les icônes à utiliser
- Les couleurs du thème
- Le mode d'affichage

Avec Vite, nous configurons le manifest directement dans le fichier `vite.config.js`. Voici comment le modifier :

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'
import tailwindcss from '@tailwindcss/vite'
import { VitePWA } from 'vite-plugin-pwa'

export default defineConfig({
  plugins: [
    react(),
    tailwindcss(),
    VitePWA({
      // Mise à jour automatique du service worker
      registerType: 'autoUpdate',

      // Fichiers à inclure dans le cache
      includeAssets: ['favicon.ico', 'apple-touch-icon.png', 'mask-icon.svg'],

      // Configuration du manifest
      manifest: {
        name: 'Annuaire des jeux', // Nom complet
        short_name: 'Jeux', // Nom court pour l'écran d'accueil
        description: 'Application de recherche de jeux vidéo',
        theme_color: '#ffffff', // Couleur du thème
        start_url: '/', // Page de démarrage
        display: 'standalone', // Mode d'affichage
        background_color: '#ffffff', // Couleur de fond au démarrage

        // Liste des icônes
        icons: [
          {
            src: 'pwa-192x192.png', // Chemin relatif au dossier public
```

```

      sizes: '192x192', // Taille de l'icône
      type: 'image/png' // Type de fichier
    },
    {
      src: 'pwa-512x512.png',
      sizes: '512x512',
      type: 'image/png',
      purpose: 'any maskable' // L'icône peut être masquée (Android)
    }
  ]
})
]
})

```

💡 Les commentaires dans le code ci-dessus expliquent chaque partie de la configuration.

⚠️ **Important :** Notez que nous conservons notre configuration existante de Tailwind CSS (`tailwindcss()`) dans les plugins. Vite permet d'utiliser plusieurs plugins simultanément, chacun ajoutant ses propres fonctionnalités à notre application.

Préparation des icônes

Pour qu'une PWA soit installable, elle doit fournir des icônes de différentes tailles. Ces icônes doivent être placées dans le dossier public de votre projet. Voici les fichiers nécessaires :

- `pwa-192x192.png` : Icône de 192x192 pixels
- `pwa-512x512.png` : Icône de 512x512 pixels
- `apple-touch-icon.png` : Icône pour iOS (180x180 pixels)
- `favicon.ico` : Favicon classique
- `mask-icon.svg` : Icône vectorielle pour la barre des tâches

Pour les tests, vous pouvez utiliser ces icônes :

```

https://formacitron.github.io/shopslist/pwa-512x512.png
https://formacitron.github.io/shopslist/pwa-192x192.png
https://formacitron.github.io/shopslist/apple-touch-icon.png
https://formacitron.github.io/shopslist/favicon.ico
https://formacitron.github.io/shopslist/mask-icon.svg

```

⚠ Pour une application en production, vous devrez créer vos propres icônes aux bonnes dimensions.

Configuration du service worker

Le service worker est un script qui s'exécute en arrière-plan et gère le cache de l'application. Il permet le fonctionnement hors-ligne et améliore les performances.

1. Créez un nouveau fichier `src/pwa.js` :

```
import { registerSW } from 'virtual:pwa-register'

// Configuration de la mise à jour du service worker
const updateSW = registerSW({
  // Appelé quand une mise à jour est disponible
  onNeedRefresh() {
    if (confirm('Une nouvelle version est disponible. Voulez-vous mettre à jour ?'))
      updateSW()
  },
  // Appelé quand l'application est prête pour le mode hors-ligne
  onOfflineReady() {
    console.log('Application prête pour une utilisation hors-ligne')
  },
})
```

2. Importez ce fichier dans votre `src/main.jsx` :

```
import React from 'react'
import ReactDOM from 'react-dom/client'
import App from './App'
import './index.css'
import './pwa' // Import du fichier de configuration PWA

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
)
```

```
</React.StrictMode>,  
)
```

Ajout du bouton d'installation

Par défaut, le navigateur propose d'installer la PWA via un petit bouton dans la barre d'adresse. Nous pouvons aussi ajouter notre propre bouton d'installation. Voici comment modifier votre `App.jsx` :

```
import { useState, useEffect, useRef } from 'react';  
  
function App() {  
  // État pour gérer l'affichage du bouton d'installation  
  const [canInstall, setCanInstall] = useState(false);  
  // Référence pour stocker l'événement d'installation  
  const deferredPrompt = useRef(null);  
  
  useEffect(() => {  
    // Fonction appelée quand l'application peut être installée  
    const handleBeforeInstallPrompt = (e) => {  
      // Empêche l'affichage automatique du prompt  
      e.preventDefault();  
      // Stocke l'événement pour une utilisation ultérieure  
      deferredPrompt.current = e;  
      // Affiche notre bouton d'installation  
      setCanInstall(true);  
    };  
  
    // Écoute l'événement d'installation  
    window.addEventListener('beforeinstallprompt', handleBeforeInstallPrompt);  
  
    // Nettoyage à la destruction du composant  
    return () => {  
      window.removeEventListener('beforeinstallprompt', handleBeforeInstallPrompt);  
    };  
  }, []);  
  
  // Fonction appelée quand l'utilisateur clique sur le bouton d'installation  
  const handleInstallClick = async () => {
```

```

    if (!deferredPrompt.current) {
      return;
    }

    // Affiche le prompt d'installation natif
    const result = await deferredPrompt.current.prompt();
    console.log(`Installation ${result.outcome}`);
    // Réinitialise l'état
    deferredPrompt.current = null;
    setCanInstall(false);
  };

  return (
    <BookmarksContext.Provider value={{ bookmarks, setBookmarks }}>
      {/* Affiche le bouton d'installation si disponible */}
      {canInstall && (
        <div className="bg-gray-300 shadow-gray-700 p-4 flex items-center">
          <div className="flex-grow text-center">
            Voulez-vous installer l'application sur votre appareil ?
          </div>
          <button
            className="px-4 py-2 rounded text-white bg-teal-600"
            onClick={handleInstallClick}
          >
            Installer
          </button>
        </div>
      )}
    <RouterProvider router={router} />
  </BookmarksContext.Provider>
);
}

export default App;

```

Configuration du cache pour l'API

Pour que l'application fonctionne hors-ligne, nous devons mettre en cache les réponses de l'API. Ajoutez cette configuration dans `vite.config.js`:

```
VitePWA({
  // ... configuration précédente ...
  workbox: {
    runtimeCaching: [{
      // Pattern pour les URLs de l'API RAWG
      urlPattern: /^https:\/\/api\.rawg\.io\/api/,
      // Stratégie de cache : sert le cache d'abord, puis met à jour
      handler: 'StaleWhileRevalidate',
      options: {
        cacheName: 'api-cache',
        // Configuration de l'expiration du cache
        expiration: {
          maxEntries: 100, // Nombre maximum d'entrées
          maxAgeSeconds: 86400 // Durée de vie (24h)
        },
        // Types de réponses à mettre en cache
        cacheableResponse: {
          statuses: [0, 200] // Codes HTTP acceptés
        }
      }
    }]
  }
})
```

Test de la PWA

Pour tester votre Progressive Web App :

1. Construisez la version de production :


```
npm run build
```

2. Lancez le serveur de prévisualisation :

```
npm run preview
```

3. Ouvrez les outils de développement (F12) et vérifiez :

- Onglet "Application" > "Service Workers" : Le service worker doit être actif
- Onglet "Application" > "Manifest" : Toutes les informations doivent être correctes
- Test hors-ligne : Désactivez le réseau dans les outils de développement et vérifiez que l'application continue de fonctionner

 **Important** : Pour que l'installation soit proposée, votre PWA doit :


- Être servie en HTTPS (sauf sur localhost)
- Avoir toutes les icônes requises
- Avoir un manifest valide
- Avoir un service worker fonctionnel

Gestion des mises à jour

Notre configuration permet une mise à jour automatique de l'application. Quand une nouvelle version est disponible :

1. Le service worker détecte la mise à jour
2. Un message est affiché à l'utilisateur
3. Si l'utilisateur accepte, l'application est mise à jour
4. La nouvelle version est chargée automatiquement

Cette approche garantit que les utilisateurs ont toujours accès aux dernières fonctionnalités et corrections de bugs.

 Félicitations ! Votre application est maintenant une Progressive Web App complète, capable de fonctionner hors-ligne et d'être installée sur les appareils des utilisateurs.