

Guide d'utilisation de Vite PWA avec Vue 3

Installation

1. Installer le plugin Vite PWA dans votre projet Vue 3 :

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

Configuration de base

Configuration du vite.config.ts

```
// vite.config.ts
import path from 'node:path'
import { VitePWA } from 'vite-plugin-pwa';
import { defineConfig } from 'vite'
import vue from '@vitejs/plugin-vue'

import tailwind from 'tailwindcss'
import autoprefixer from 'autoprefixer'

// https://vitejs.dev/config/
export default defineConfig({

  css: {
    postcss: {
      plugins: [tailwind(), autoprefixer()],
    },
  },

  plugins: [vue(), VitePWA({
    registerType: 'prompt',
    injectRegister: false,

    pwaAssets: {
      disabled: false,
      config: true,
    },

    manifest: {
      name: 'elan_app',
      short_name: 'elan_app',
      description: 'Application Elan maintenance',
      theme_color: '#ffffff',
    },

    workbox: {
      globPatterns: ['**/*.{js,css,html,svg,png,ico}'],
      cleanupOutdatedCaches: true,
      clientsClaim: true,
    },

    devOptions: {
      enabled: false,
      navigateFallback: 'index.html',
      suppressWarnings: true,
      type: 'module',
    },
  })],
  // base: '/elan_app/',
  resolve: {
    alias: {
      '@': path.resolve(__dirname, './src'),
    },
  },
})
```

Fonctionnalités principales

1. Service Worker

Le Service Worker est automatiquement généré par Vite PWA. Voici les principales configurations : (toujours basé sur notre application elan_app)

```
VitePWA({
  registerType: 'prompt',
  injectRegister: false,

  pwaAssets: {
    disabled: false,
    config: true,
  },

  manifest: {
    name: 'elan_app',
    short_name: 'elan_app',
    description: 'Application Elan maintenance',
    theme_color: '#ffffff',
  },
})
```

```

workbox: {
  globPatterns: ['**/*.{js,css,html,svg,png,ico}'],
  cleanupOutdatedCaches: true,
  clientsClaim: true,
},

devOptions: {
  enabled: false,
  navigateFallback: 'index.html',
  suppressWarnings: true,
  type: 'module',
},
},
}))

```

2. Installation de l'application

Pour gérer l'installation de la PWA :

```

// PWABadge.vue
<script setup lang="ts">
import { useRegisterSW } from 'virtual:pwa-register/vue'

const { needRefresh, updateServiceWorker } = useRegisterSW({
  immediate: true,
  onRegistered(r) {
    console.log('SW Registered:', r)
  },
  onRegisterError(error) {
    console.log('SW registration error', error)
  },
})

const updateApp = async () => {
  await updateServiceWorker()
}

const closeToast = () => {
  needRefresh.value = false
}
</script>

<template>
<div v-if="needRefresh" class="pwa-toast">
  <div class="message">
    Nouvelle version disponible. Voulez-vous mettre à jour ?
  </div>
  <div class="buttons">
    <button @click="updateApp" class="update-button">Mettre à jour</button>
    <button @click="closeToast" class="close-button">Fermer</button>
  </div>
</div>
</template>

```

3. Installation de Pinia

Pour gérer l'installation de Pinia :

```
npm install pinia
```

Gestion de l'état avec Pinia

Pinia est une bibliothèque de gestion d'état pour Vue.js, offrant une alternative plus légère et intuitive à Vuex. Elle permet de créer des stores pour gérer l'état global de l'application de manière efficace et typesafe.

Installation et configuration

Dans notre projet, Pinia est installé et configuré comme suit :

```

import { createPinia } from 'pinia'
import './style.css'
import App from './App.vue'
import router from './router.ts'
import '@fortawesome/fontawesome-free/css/all.css'
import piniaPluginPersistedstate from 'pinia-plugin-persistedstate'
import { useAuthStore } from './stores/auth.ts'
const pinia = createPinia()

pinia.use(piniaPluginPersistedstate)

```

Nous utilisons également le plugin `pinia-plugin-persistedstate` pour persister l'état de certains stores entre les rechargements de page.

Création d'un store

Un store Pinia est généralement défini dans un fichier séparé. Par exemple, voici comment pourrait être défini notre store d'authentification :

```

import { defineStore } from 'pinia'
import { config } from '@config';
import axios from 'axios'

```

```

// Création d'une instance axios avec une configuration de base
const apiClient = axios.create({
  baseURL: config.apiUrl,
  withCredentials: true,
  headers: {
    'Accept': 'application/json',
    'Content-Type': 'application/json'
  }
});

// Définition des interfaces pour le typage
interface User {
  id: number;
  email: string;
}

interface AuthState {
  user: User | null;
  error: string | null;
  clientLogo: string | null;
}

// Définition du store d'authentification
export const useAuthStore = defineStore({
  id: "auth", // Identifiant unique du store

  // État initial du store
  state: (): AuthState => ({
    user: null,
    error: null,
    clientLogo: null
  } as AuthState),

  // Getters pour accéder à l'état de manière calculée
  getters: {
    // Vérifie si un utilisateur est authentifié
    isAuthenticated: (state): boolean => !!state.user
  },

  // Actions pour modifier l'état du store
  actions: {
    // Récupère le logo du client depuis l'API
    async fetchClientLogo(): Promise<void> {
      try {
        const response = await apiClient.get('/client-logo');
        this.clientLogo = response.data.logo_url;
      } catch (error) {
        console.error('Erreur lors de la récupération du logo client:', error);
        this.clientLogo = null;
      }
    },

    // Gère la connexion de l'utilisateur
    async login(email: string, password: string): Promise<boolean> {
      try {
        const response = await apiClient.post('/vue/login', { email, password });
        const { access_token, user } = response.data;

        // Stocke le token dans le localStorage
        localStorage.setItem('token', access_token);

        // Met à jour l'état du store
        this.user = user;
        this.error = null;

        return true;
      } catch (error: any) {
        this.error = error.response?.data?.message || 'Une erreur est survenue lors de la connexion';
        console.error('Erreur de connexion:', this.error);
        return false;
      }
    },

    // Vérifie l'authentification de l'utilisateur
    async checkAuth(): Promise<void> {
      const token = localStorage.getItem('token');
      if (token) {
        try {
          const response = await apiClient.get('/verifyToken', {
            headers: { Authorization: `Bearer ${token}` }
          });

          this.user = response.data.user;
        } catch (error) {
          // Si le token n'est pas valide, réinitialise l'état
          this.user = null;
          localStorage.removeItem('token');
        }
      } else {
        this.user = null;
      }
    },

    // Déconnecte l'utilisateur
    logout(): void {
      this.user = null;
      this.error = null;
    }
  }
});

```

```

    localStorage.removeItem('token');
  },

  // Efface les erreurs
  clearError() {
    this.error = null;
  },
},

// Configuration de la persistance du store
persist: {
  key: 'auth', // Clé utilisée pour le stockage
  storage: localStorage, // Utilise le localStorage pour la persistance
}
})

```

Utilisation dans les composants

Dans nos composants Vue, nous pouvons utiliser les stores Pinia comme suit :

```

<script setup>
import { useAuthStore } from '@/stores/auth'
const authStore = useAuthStore()

// Utilisation de l'état
console.log(authStore.isAuthenticated)

// Appel d'une action
authStore.login(userData)
</script>

```

Exemple d'utilisation dans notre projet

Dans notre application, nous utilisons Pinia pour gérer l'état d'authentification et les notifications. Par exemple, dans `App.vue`, nous observons les changements d'authentification et gérons les notifications :

```

// App.vue
<script setup>
// Gestion des notifications
onMounted(() => {
  if (authStore.isAuthenticated) {
    notificationsStore.initSSE();
  }
});

onUnmounted(() => {
  notificationsStore.closeSSE();
});

// Surveillance des changements d'authentification
watch(() => authStore.isAuthenticated, (isAuthenticated) => {
  isAuthenticated ? notificationsStore.initSSE() : notificationsStore.closeSSE();
});

// Surveillance des nouvelles notifications
watch(() => notificationsStore.lastNotification, (newNotification) => {
  if (newNotification?.data?.message && newNotification.isNew) {
    toast({
      title: "Nouvelle notification",
      description: newNotification.data.message,
      duration: 5000,
    });
  }
});
</script>

```

Cette approche nous permet de réagir aux changements d'état de manière réactive.

Bonnes pratiques

1. Assets

- Fournir des icônes pour toutes les tailles requises

```

// manifest.json
"icons": [
  {
    "src": "src/assets/elan_192.png",
    "sizes": "192x192",
    "type": "image/png"
  },
  {
    "src": "src/assets/elan_512.png",
    "sizes": "512x512",
    "type": "image/png"
  },
  {
    "src": "src/assets/elan_180.png",
    "sizes": "180x180",
    "type": "image/png",
    "purpose": "any maskable"
  }
]

```

```

    }
  ],
  ○ Optimiser les images pour réduire la taille
  ○ Inclure un favicon et une apple-touch-icon

<!-- index.html -->
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0, viewport-fit=cover, maximum-scale=1.0, user-sca
  <title>Elan Maintenance Application</title>
  <link rel="manifest" href="/manifest.json">
  <meta name="apple-mobile-web-app-capable" content="yes">
  <meta name="apple-mobile-web-app-status-bar-style" content="black-translucent">
  <link rel="apple-touch-icon" href="/src/assets/elan_192.png">
  <link rel="apple-touch-icon" sizes="152x152" href="/src/assets/elan_152.png">
  <link rel="apple-touch-icon" sizes="180x180" href="/src/assets/elan_180.png">
  <link rel="apple-touch-icon" sizes="167x167" href="/src/assets/elan_167.png">
  <link rel="apple-touch-startup-image" href="/src/assets/splash.png">
</head>

```

2. Performance

- Mettre en cache les ressources statiques

```

// vite.config.ts
workbox: {
  globPatterns: ['**/*.{js,css,html,svg,png,ico}'],
  cleanupOutdatedCaches: true,
  clientsClaim: true,
},

```

- Utiliser la stratégie de cache appropriée pour chaque type de ressource, par exemple :
 - Pour les fichiers CSS et JS, utiliser une stratégie de cache "Cache First" afin d'assurer un chargement rapide.
 - Pour les images, utiliser une stratégie "Network First" pour garantir que les utilisateurs voient toujours les dernières versions.
 - Pour les fichiers HTML, une stratégie "Stale While Revalidate" peut être bénéfique pour améliorer l'expérience utilisateur tout en gardant le contenu à jour.
- Implémenter une logique de fallback pour le contenu hors ligne en fournissant une page d'accueil ou un message d'erreur personnalisé lorsque l'utilisateur n'est pas connecté à Internet. Cela peut inclure l'utilisation d'un fichier HTML statique qui sera servi lorsque le réseau n'est pas disponible.

Dans notre application nous n'avons pas encore les 2 derniers points.

3. Mise à jour

- Gérer les mises à jour de l'application
- Notifier l'utilisateur des nouvelles versions
- Prévoir une stratégie de rechargement

```

// PWABadge.vue
<script setup lang="ts">
import { useRegisterSW } from 'virtual:pwa-register/vue'

const { needRefresh, updateServiceWorker } = useRegisterSW({
  immediate: true,
  onRegistered(r) {
    console.log('SW Registered:', r)
  },
  onRegisterError(error) {
    console.log('SW registration error', error)
  },
})

const updateApp = async () => {
  await updateServiceWorker()
}

const closeToast = () => {
  needRefresh.value = false
}
</script>

<template>
  <div v-if="needRefresh" class="pwa-toast">
    <div class="message">
      Nouvelle version disponible. Voulez-vous mettre à jour ?
    </div>
    <div class="buttons">
      <button @click="updateApp" class="update-button">Mettre à jour</button>
      <button @click="closeToast" class="close-button">Fermer</button>
    </div>
  </div>
</template>

```

Spécificités du projet elan_app

Manifest personnalisé

Notre application utilise un `manifest.json` personnalisé qui inclut des configurations pour les icônes, les raccourcis, et les paramètres de partage. Voir le fichier `manifest.json` pour plus de détails.

Gestion des mises à jour et du mode hors ligne

Nous utilisons deux composants principaux pour gérer les mises à jour de l'application et le mode hors ligne :

- **Prompt.vue** : Gère les notifications pour le mode hors ligne et les mises à jour disponibles.
- **PWABadge.vue** : Affiche une notification lorsqu'une mise à jour est disponible et permet à l'utilisateur de l'installer.

Configuration de l'environnement

Notre application utilise une configuration dynamique pour l'URL de l'API en fonction de l'environnement de déploiement. Voir le fichier `src/config.js` pour plus de détails.

Intégration avec l'authentification

L'initialisation de notre PWA est liée à l'authentification de l'utilisateur. Nous vérifions l'authentification avant de monter l'application.

Gestion des notifications

Notre application inclut une logique pour gérer les notifications en temps réel via SSE (Server-Sent Events). Cette logique est implémentée dans le composant principal `App.vue`.

Dépendances spécifiques

Notre projet utilise plusieurs dépendances spécifiques pour la PWA, notamment :

- `@vite-pwa/assets-generator` : Pour la génération d'assets PWA
- `vite-plugin-pwa` : Pour la configuration de la PWA avec Vite
- `workbox-window` est une bibliothèque qui fait partie de l'écosystème Workbox, spécialement conçue pour faciliter l'interaction entre votre application web et le service worker. (Ensemble de modules destinés à s'exécuter dans le contexte de fenêtre, c'est-à-dire à l'intérieur de vos pages Web PWA.)

Ces ajouts permettront de mieux refléter les spécificités de votre projet dans la documentation.

Commandes et Bibliothèques du Projet

Vue.js et Vite

- Installation du projet : `npm create vite@latest nom-du-projet -- --template vue-ts`
- Lancement du serveur de développement : `npm run dev`
- Build du projet : `npm run build`
- Preview du build : `npm run preview`

Pinia

- Installation : `npm install pinia`
- Utilisation dans le fichier principal :

```
// main.ts
import { createPinia } from 'pinia'
import piniaPluginPersistedstate from 'pinia-plugin-persistedstate'

const pinia = createPinia()
pinia.use(piniaPluginPersistedstate)

const app = createApp(App)
app.use(pinia)
```

Vue Router

- Installation : `npm install vue-router@4`
- Configuration dans le fichier principal :

```
// router.ts
import { createRouter, createWebHistory } from 'vue-router'
import { useAuthStore } from './stores/auth.ts'

const router = createRouter({
  history: createWebHistory(),
  routes
})

app.use(router)
```

Axios

- Installation : `npm install axios`
- Configuration d'une instance :

```
// store/auth.ts store/interventions.js store/notifications.js
import axios from 'axios'
const apiClient = axios.create({
  baseURL: config.apiUrl,
  withCredentials: true,
  headers: {
    'Accept': 'application/json',
    'Content-Type': 'application/json'
  }
});
```

PWA (Progressive Web App)

- Installation : `npm install vite-plugin-pwa -D`
- Configuration dans `vite.config.ts` :

```
// vite.config.ts
import path from 'node:path'
import { VitePWA } from 'vite-plugin-pwa';
import { defineConfig } from 'vite'
import vue from '@vitejs/plugin-vue'

import tailwind from 'tailwindcss'
import autoprefixer from 'autoprefixer'

// https://vitejs.dev/config/
export default defineConfig({

  css: {
    postcss: {
      plugins: [tailwind(), autoprefixer()],
    },
  },

  plugins: [vue(), VitePWA({
    registerType: 'prompt',
    injectRegister: false,

    pwaAssets: {
      disabled: false,
      config: true,
    },

    manifest: {
      name: 'elan_app',
      short name: 'elan app',
      description: 'Application Elan maintenance',
      theme_color: '#ffffff',
    },

    workbox: {
      globPatterns: ['**/*.{js,css,html,svg,png,ico}'],
      cleanupOutdatedCaches: true,
      clientsClaim: true,
    },

    devOptions: {
      enabled: false,
      navigateFallback: 'index.html',
      suppressWarnings: true,
      type: 'module',
    },
  })],
  // base: '/elan_app/',
  resolve: {
    alias: {
      '@': path.resolve(__dirname, './src'),
    },
  },
})
```

Tailwind CSS

- Installation :
- ```
npm install -D tailwindcss@latest postcss@latest autoprefixer@latest
npx tailwindcss init -p
```
- Configuration dans `vite.config.ts` :

```
// vite.config.ts
import tailwind from 'tailwindcss'
import autoprefixer from 'autoprefixer'
export default defineConfig({
 css: {
 postcss: {
 plugins: [tailwind(), autoprefixer()],
 },
 },
})
```

## FontAwesome

- Installation : `npm install @fortawesome/fontawesome-free`
- Importation dans le fichier principal :

```
// main.ts
import '@fortawesome/fontawesome-free/css/all.css'
```

## Persistance des données Pinia

- Installation : `npm install pinia-plugin-persistedstate`
- Configuration dans le fichier principal :

```
// main.ts
import piniaPluginPersistedstate from 'pinia-plugin-persistedstate'
const pinia = createPinia()
pinia.use(piniaPluginPersistedstate)
```

## Géolocalisation

- Utilisation de l'API native du navigateur dans le store de localisation :

```
// store/location.ts
import { defineStore } from 'pinia';

export const useLocationStore = defineStore('location', {
 state: () => ({
 currentPosition: null,
 watchId: null,
 error: null,
 }),
 actions: {
 startTracking() {
 if ("geolocation" in navigator) {
 this.watchId = navigator.geolocation.watchPosition(
 this.updatePosition,
 this.handleError,
 { enableHighAccuracy: true, timeout: 10000, maximumAge: 0 }
);
 } else {
 this.error = "La géolocalisation n'est pas supportée par ce navigateur.";
 }
 },
 stopTracking() {
 if (this.watchId !== null) {
 navigator.geolocation.clearWatch(this.watchId);
 this.watchId = null;
 }
 },
 updatePosition(position) {
 this.currentPosition = {
 latitude: position.coords.latitude,
 longitude: position.coords.longitude,
 };
 this.error = null;
 },
 handleError(error) {
 console.warn("Erreur de géolocalisation:", error.message);
 this.error = error.message;
 },
 calculateDistance(lat1, lon1, lat2, lon2) {
 const R = 6371; // Rayon de la Terre en km
 const dLat = this.deg2rad(lat2 - lat1);
 const dLon = this.deg2rad(lon2 - lon1);
 const a =
 Math.sin(dLat/2) * Math.sin(dLat/2) +
 Math.cos(this.deg2rad(lat1)) * Math.cos(this.deg2rad(lat2)) *
 Math.sin(dLon/2) * Math.sin(dLon/2);
 const c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
 return R * c; // Distance en km
 },
 deg2rad(deg) {
 return deg * (Math.PI/180);
 },
 getDistanceToInterventions(interventions) {
 if (!this.currentPosition) {
 // Retourner les interventions sans distance si la position n'est pas disponible
 return interventions.map(intervention => ({
 ...intervention,
 distance: null
 }));
 }
 return interventions.map(intervention => ({
 ...intervention,
 distance: this.calculateDistance(
 this.currentPosition.latitude,
 this.currentPosition.longitude,
 intervention.sit_lat,
 intervention.sit_lng
)
 }));
 },
 },
});
```



Ces commandes et configurations représentent les principales bibliothèques et outils utilisés dans notre projet Vue.js avec Vite.

## Commandes utiles

```
Build de production
npm run build

Build de développement
npm run dev

Test de la PWA en local
npm run preview

Générer les icônes (avec sharp)
npm run generate-pwa-icons
```