Documentation technique SkiWeather

Table des matières :

---

1. Présentation de l’app

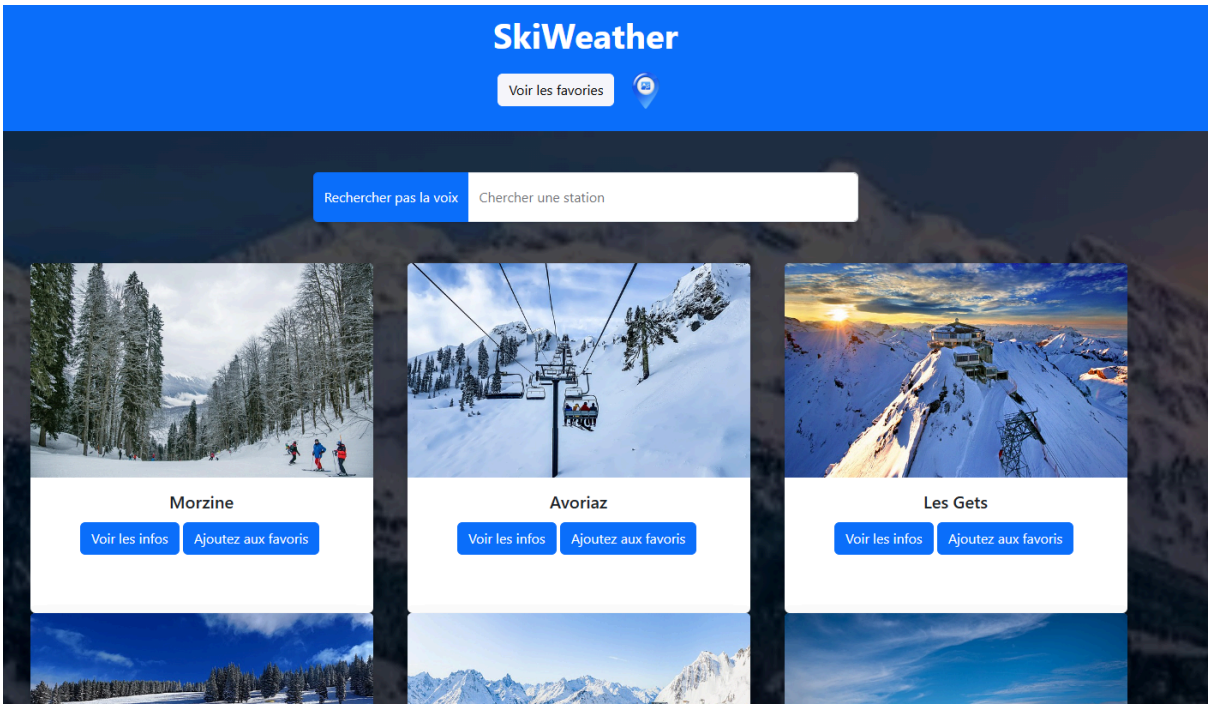
SkiWeather est une application web permettant de consulter différentes informations pour 35 stations de ski françaises. Elle permet de consulter les prévisions météo, les chutes de neige passées et à venir ainsi que d'autres informations utiles.

SkiWeather embarque aussi la possibilité de consulter les stations de ski directement depuis une map.

2. Présentation des fonctionnalitées

Comme expliqué dans la partie 1, SkiWeather permet de consulter certaines informations concernant les stations de ski françaises.

La page d’accueil affiche toutes les stations disponibles, on a la possibilité de les parcourir directement, de faire une recherche par leur nom ou encore une recherche vocale



Après avoir sélectionné une station on accède à ses informations :

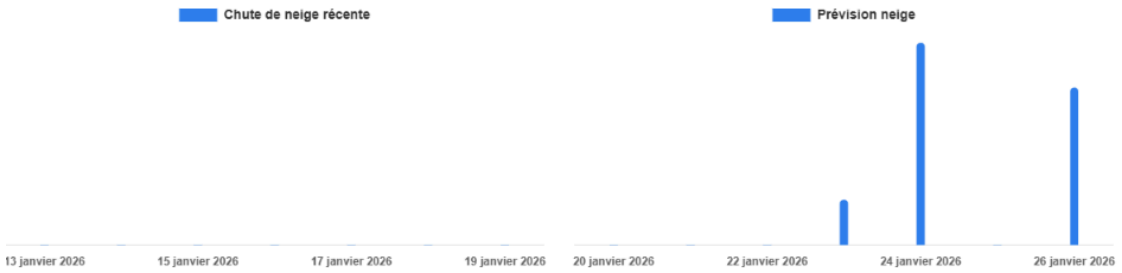
- L’altitude du village et l’altitude max du domaine
- Le massif
- Les chutes de neige sur les 7 derniers jours ainsi que sur les 7 prochains

- Les prévisions météo pour les 7 prochains jours

AVORIAZ

Altitude du village : 1800
Altitude max du domaine : 2466
Massif : Chablais

Obtenir le BRA | Plus d'info sur la météo | Voir sur la map | Ajoutez aux favori

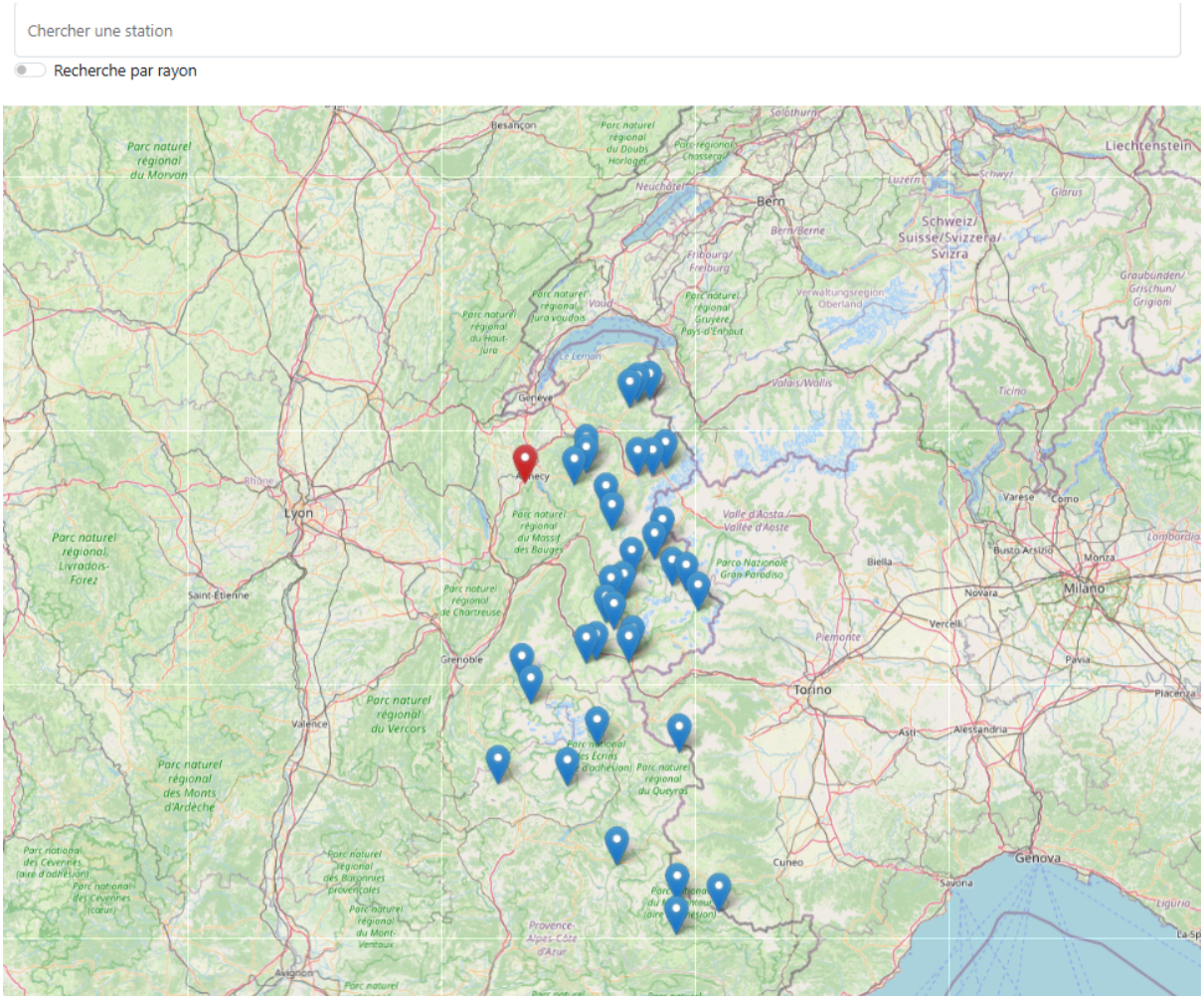


Jour	Temps	Min/Max	Vent	Précipitation
20 janvier 2026	Nuages	-1.4/3.3 °C	10.3km/h	0 cm
21 janvier 2026	Nuages	-0.7/2.9 °C	8.6km/h	0 cm
22 janvier 2026	Nuages	-2/2.6 °C	10.4km/h	0 cm
23 janvier 2026	Neige légère	-2.7/2.2 °C	8.4km/h	0.42 cm
24 janvier 2026	Neige modérée	-4.7/0.1 °C	10km/h	1.89 cm
25 janvier 2026	Brouillard	-4.4/-1.6 °C	6.8km/h	0 cm
26 janvier 2026	Grains de neige	-5.2/-2.9 °C	6.6km/h	1.47 cm

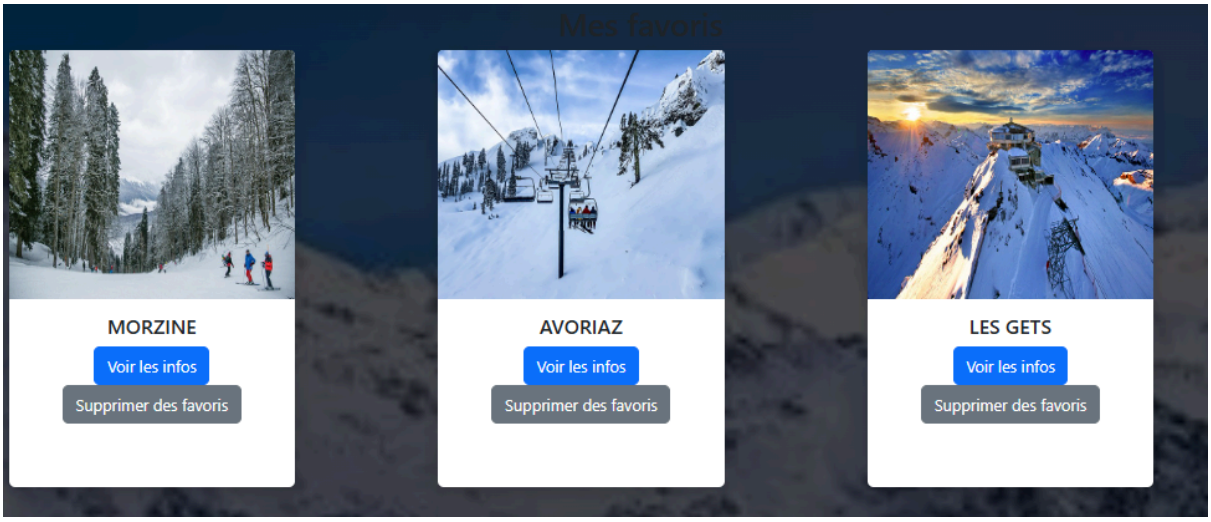
Comme on peut le voir, certaines informations supplémentaires son accessibles à l'aide de boutons, on peut :

- Obtenir le BRA (Bulletin d'estimation du Risque d'Avalanche)
- Obtenir plus d'information sur la météo ce qui nous donnera la météo heure par heure sur 3 jours
- Voir la station directement sur la map
- Ajouter la station aux favoris

SkiWeather offre aussi la possibilité de consulter une carte pour voir où sont situées les stations. Une fois dans la map on peut rechercher des stations par leur nom ou alors par un rayon par rapport à notre position.



L’application offre aussi la possibilité d’ajouter ou de supprimer des stations à votre liste de favoris



3. Explication du fonctionnement

3.1 Langage, librairie, API

Langages de développement utilisés : HTML/CSS, JavaScript
Librairie utilisée : Leaflet (map), pdf.js, chart.js, Bootstrap (css)
APIs utilisées : Météo France, OpenMeteo, Pixabai (banque d'image), Web Speech API, Geolocation API MDN

3.2. Architecture de l'application

L'architecture de la pwa se présente ainsi :

```
pwa/
├── css/
├── favicon/
│   └── site.webmanifest
├── img/
├── js/
├── lib/
├── index.html
└── service_worker.js
```

3.3 Fonctionnement

3.3.1 Lancement

Lorsque vous consultez l'application sur le web et qu'elle n'est pas installée, une boîte de dialogue s'ouvre pour vous proposer de l'installer. La boîte de dialogue est appelé par la fonction **onBeforeInstallPrompt(event)** à la ligne 50 du fichier /sae302/js/script.js

Le fonctionnement reste le même que l'app soit installer ou non

Quand vous arrivez sur la page d'accueil, le fichier /sae302/js/pwa.js est chargé et exécute automatiquement les fonctions suivante :

- **setInterval(is_offline, 10000)** qui permet de vérifier si le client est toujours connecté à internet toutes les 10 secondes

- **update_massifs()** qui permet de récupérer la liste des différents massifs à l'aide d'une requête vers l'API **Bulletin Avalanche** de météo france.

- **update_stations()** qui permet de récupérer la liste de toutes les stations. Dans notre cas, aucune API ne donne une liste de stations de ski française, les stations de ski

sont donc stockéeslocalement dans le fichier **/sae302/js/stations_ski_alpes_française.json**

- **update_images()** qui permet de récupérer une image par station à l'aide d'une requête vers l'API de **pixabay.** Les images sont ensuite stockées dans une base de donnée IndexedDB créée par la fonction **update_images()** à l'aide de la fonction **get_sans_clef(url, id, option)**.

- **localisation()** qui permet de récupérer la position du client afin de centrer la map sur sa position. En cas de succès cette fonction appelle la fonction success() et met à jour les variables globales **latitude** et **longitude** pour stocker la position du client.

3.3.2 Accueil et informations

Une fois dans l'accueil pour consulter les informations d'une station de ski vous pouvez parcourir directement les stations puis cliquer sur le bouton "voir les infos".

Ce bouton possède comme ID le nom de la station et appel la fonction **info_station(event)** à l'aide d'un écouteur d'évenement de type **"click"** qui récupère le nom de la station choisi à l'aide de la variable **event**, elle cherche ensuite parmi la liste des stations stocké dans la variable globale **stations** la station qui a été choisie. Le but est surtout de récupérer le nom de son massif ainsi que la latitude et la longitude de la station pour pouvoir obtenir le BERA à l'aide de l'API de Météo France et les prévisions météo qui la concernent.

Le BERA est récupéré à l'aide d'une nouvelle requête vers l'API de Météo France avec l'ID du massif concerné. Il est ensuite traité par la fonction **affiche_pdf_bra(donnee)** à l'aide de la librairie **pdf.js** pour pouvoir l'afficher proprement sans barre d'outil.

Les prévisions météo sont récupérées par une requête à l'API de OpenMeteo à l'aide de la latitude et de la longitude de la station. Les données sont ensuite traité par les fonctions :

- **affiche_meteo(donnee)** qui va afficher un histogramme des chutes de neige des 7 derniers jours, créé à l'aide de la librairie **chart.js**

- **affiche_meteo_prevision(donnee)** qui elle va afficher l'histogramme des prévisions de chute de neige pour les 7 prochains jours, ainsi qu'un tableau qui représente les prévisions météorologiques pour les 7 prochains jours.

- **more_meteo(donnee)** qui est lancé à l'aide d'un écouteur d'événement de type **"click"** sur le bouton **"plus d'info sur la météo"** qui affiche la météo heure par heure pour les 3 prochains jours.

Enfin sur la page d'information de la station, vous pouvez, à l'aide du bouton **"Voir sur la map"** afficher directement la station sur la map. *(Le détail du fonctionnement de la map est détaillé en 3.3.3)*

3.3.3 Map

Pour accéder à la map il y a plusieurs possibilités :

- Cliquer sur le logo de la map présent dans le **<header>** qui appellera la fonction **initMap(event)**

- Cliquer sur le bouton **"Voir sur la map"** présent dans la page d'information d'une station qui appelle aussi la fonction **initMap(event)**

Une fois dans la fonction **initMap(event)** à l'aide de la variable event on pourra savoir ce qu'on doit afficher. Si la map est appelée en cliquant sur le logo de la map alors l'event remplira la condition du **if** ligne 1322 de /sae302/js/pwa.js

```
if (event.target && event.target.alt == "logo_map" || event == "reset")
```

Dans ce cas, on créera un **marker** par stations en fonction de leur latitude et leur longitude qu'on affichera sur la map.

Dans le deuxième cas, si le client clique sur le bouton **"Voir sur la map",** alors on rentre dans le **else** ligne 1351 et on récupère la latitude et la longitude de la station demandée qui ont été placé dans la **value** du bouton puis on créer un **marker.**

Dans les deux cas, si la géolocalisation du client est disponible alors on crée aussi un **marker** sur la position du client. A la suite de tout cela, on affiche le groupe de marker sur la map.

Lorsque que la géolocalisation du client est disponible, on peut aussi en plus de la recherche par nom disponible dans tous les cas, faire une recherche par rayon.

Pour ce faire, la fonction **active_rayon(event)** est appelée quand l'input "**active_rayon"** est activé, et aussi à chaque fois que le rayon change. La fonction **active_rayon(event)** trace un cercle du rayon choisi sur la map centré sur la position du client. Ensuite à l'aide de la boucle **for** ligne 1403 on regarde si la distance obtenu entre la position du client et la station est inférieur à la valeur du rayon choisi multiplié par 1000 pour l'avoir en km à l'aide de la condition ligne 1407 :

```
if (centre.distanceTo(position_station) <= CHOIX_RAYON.value*1000)
```

*(Ici **centre** représente la position du client)*

3.3.4 Gestion des favoris

L'application offre la possibilité de mettre des stations en favoris. Pour ce faire, on utilise le local storage pour que le client garde ses favoris d'une utilisation à l'autre.

Lorsque le client souhaite ajouter une station favorite, la fonction **ajout_favori(event)** est appelée à l'aide d'un écouteur d'événement de type **"click"** sur le bouton **"Ajouter aux favoris".** Cette fonction va vérifier si la liste des favoris est présente dans le localstorage, puis la récupérer et la stocker dans une liste temporaire et ajouter le nom station à ajouter dans cette liste. Enfin la fonction met à jour le localstorage avec les données de la nouvelle liste.

Pour supprimer des favoris le principe est le même, la fonction **supp_favori(event)** récupère la liste des favoris dans le localstorage puis recrée une liste temporaire en vérifiant pour chaque nom si il correspond au nom de la station que le client veut supprimer, si c'est le cas alors il ne l'ajoute pas. Et enfin il met à jour le localstorage avec les données de la nouvelle liste.

3.3.5 Gestion de l'affichage

Tout l'affichage de l'application repose sur la classe **"pas_affiche"** définie dans **/sae302/css/default.css**

```
.pas_affiche
{
  display: none;
}
```

Cette classe est ajoutée ou supprimée au **div** que l'on souhaite afficher ou non.
Les **div** principales sont :

- la div **main_container** qui représente la totalité de la page excepté l'affichage de la map
- la div **page_default** qui représente le contenu affiché lorsque que l'on accède aux informations d'une station
- la div **menu_map** qui représente la totalité du contenu relié à la map
- la div **menu_favori** qui représente le contenu relié à l'affichage des favoris

3.3.6 Gestion du mode hors ligne

Comme expliqué au debut, toutes les 10 secondes la fonction **is_offline()** vérifie si le client est encore connecté à internet.

Si il ne l'est plus, il peut quand même accéder aux données qui sont stockées dans le localstorage ou dans la base de donnée IndexedDB :

- La liste des stations.

- Toutes les informations météorologiques des stations qu'il a déjà consultées à l'exception des prévisions météos heure par heure sur 3 jours.

- Consulter le BERA, cependant la consultation du BERA n'est là qu'à titre de secours, en effet le BERA étant une donnée sensible, le BERA sera mis à jour dès que l'utilisateur sera de nouveau connecté à internet.

  Toutes les images de stations et le BERA ne pouvant être stocké qu'en binaire (blob) ne sont pas stockés dans le localstorage mais dans la BDD IndexedDB. De plus le localstorage étant limité à 5 Mo, le moyen le plus efficace pour que l'utilisateur ait accès aux images et au BERA en mode hors ligne est de les stocker dans IndexedDB.

3.3.6 CSS

La plus grande partie du CSS a été réalisée à l'aide de la librairie **Bootstrap CSS** mais certaines choses on été ajouté dans **/sae302/css/default.css** pour le bon fonctionnement de l'app.

L'élément le plus important du default.css est l'instruction sur la classe **"map"** :

```
#map
{
  height: 100vh;
  width: 100vw;
}
```

La librairie leaflet à besoin qu'on fixe une taille à la map pour l'afficher, sans cela la map ne s'affiche pas.

Le reste du default.css représente seulement des ajustements graphiques comme, la taille des histogrammes ou encore des tableaux mais rien n'affecte le bon fonctionnement de l'application.

4. Fonctionnement des API

4.1 Météo France

L'API de Météo France fonctionne avec une clef API, elle permet de récupérer dans un premier temps la liste des massifs dans un JSON, les principales informations utiles sont l'ID correspondant ainsi que son nom.
La requête pour obtenir la liste des massifs est la suivante :

```
https://public-api.meteofrance.fr/public/DPBRA/v1/liste-massifs
```

*(fonction **update_massifs()** dans /sae302/js/pwa.js ligne 371)*

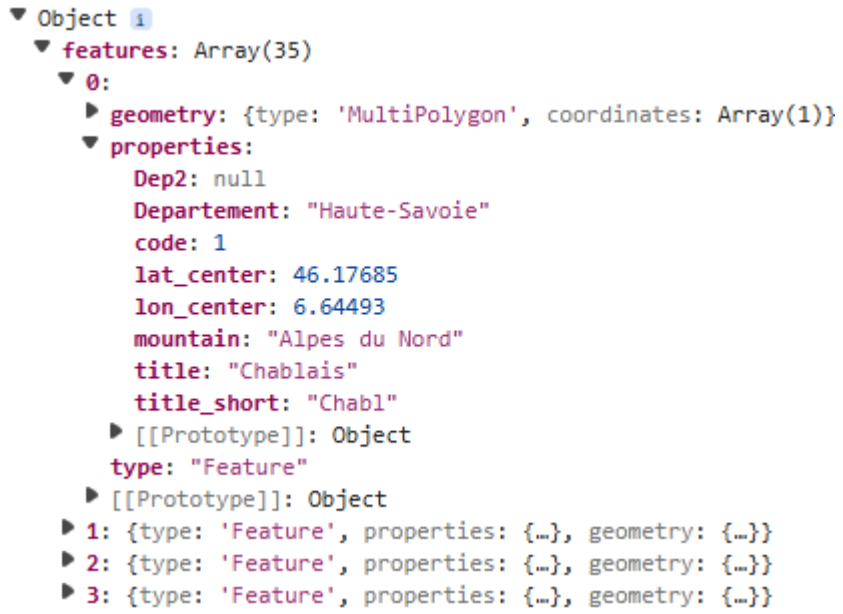
update_massifs() appel la fonction **get(url, id)**, c'est dans cette fonction que la clef API est renseignée.

Par la suite, c'est avec l'ID d'un massif qu'on peut récupérer le BERA correspondant, la requête pour obtenir le BERA est la suivante.

```
https://public-api.meteofrance.fr/public/DPBRA/v1/massif/BRA?id-massif="+ID_MASSIF+"&format=pdf
```

Cette url est créé dans la fonction **affiche_bra(event)** dans /sae302/js/pwa.js ligne 760. Comme pour la liste des massifs, l'url est passé en paramètre dans la fonction **get(url, id)**. *(La clef API est la même pour toutes les requêtes vers l'api Météo France)*

Exemple de réponse de l'api Météo France pour la liste des massifs



4.2 OpenMeteo

L'API OpenMeteo fonctionne elle sans clef, les paramètres principaux pour obtenir les informations souhaitées sont la latitude et la longitude de la station dont on veut les données météorologiques. Ici aussi les données sont récupérées au format JSON.

Voici un exemple d'url pour une requete vers l'API OpenMeteo :

```
https://api.open-meteo.com/v1/forecast?latitude="+stations[index].latitude+"&longitude="+stations[index].longitude+"&daily=weather_code,temperature_2m_max,temperature_2m_min,snowfall_sum,wind_speed_10m_max&past_days=7
```

*(fonction **info_station(event)** dans /sae302/js/pwa.js ligne 718)*

Cette url est créé en fonction de la station sélectionnée, elle permet de récupérer les prévisions météorologique *(température max/min, chute de neige, vent et le weather_code)* pour les 7 prochains jours *(valeur par défaut pour le paramètre forecast)* ainsi que les 7 derniers *(paramètre past_days=7).* Le weather_code correspond à une valeur qui identifie une condition météo *(ensoleillé, nuageux…)*, La signification de chaque code est renseignée dans la constante **WEATHER_CODE** ligne 41 dans /sae302/js/pwa.js.

Exemple de réponse de l'API OpenMeteo pour la requête ci-dessus

{latitude: 46.18, longitude: 6.6999993, generationtime_ms: 2.0667314529418945, utc_offset_seconds: 0, timezone: 'GMT', …}
daily:
snowfall_sum: (14) [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.47, 5.81]
temperature_2m_max: (14) [7.6, 9.1, 9.1, 7.5, 6.8, 9.9, 6.9, 6.3, 5.5, 5.5, 5.4, 5.3, 1.1, -0.5]
temperature_2m_min: (14) [-2.8, -2.6, 0, -2.3, 1.9, 0.1, -0.4, -1.5, -2, -1.9, 0.5, -1.3, -1.6, -2.2]
time: (14) ['2026-01-13', '2026-01-14', '2026-01-15', '2026-01-16', '2026-01-17', '2026-01-18', '2026-01-19', '2026-01-20', '2026
weather_code: (14) [3, 3, 3, 3, 3, 3, 3, 3, 3, 61, 61, 77, 77]
wind_speed_10m_max: (14) [2.9, 3.7, 3.4, 4.6, 5.2, 5.5, 5.7, 5.8, 5.2, 5.7, 6.2, 6.1, 5, 6.5]
[[Prototype]]: Object
daily_units: {time: 'iso8601', weather_code: 'wmo code', temperature_2m_max: '°C', temperature_2m_min: '°C', snowfall_sum: 'cm', …}
elevation: 988
generationtime_ms: 2.0667314529418945
latitude: 46.18
longitude: 6.6999993
timezone: "GMT"
timezone_abbreviation: "GMT"
utc_offset_seconds: 0
[[Prototype]]: Object

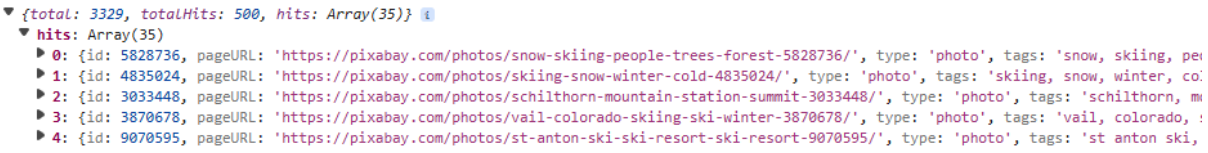
4.3 API Pixabai

L'API Pixabai est elle un petit peu différente. Elle nécessite une clef API pour fonctionner mais elle est passée directement dans l'url. Les requêtes de l'API Pixabai sont donc faites à l'aide de fonction **get_sans_clef(url, id, option).** Voici l'exemple de la seule requête vers l'API Pixabai :

```
    https://pixabay.com/api/?key=54187326-38043289379cc9208ae4f89cd&q=ski%20resort&min_width=1920&min_height=1080&per_page="+n+"&image_type=photo
```

Ici la variable **n** représente le nombre d'images que l'on souhaite. Dans notre cas **n = nombre de stations**. Le paramètres **q** représente les mots clefs de la recherche. Avec la réponse, on refait appel à la fonction **get_sans_clef(url, id, option)** avec comme url le lien pour télécharger l'image. Comme expliqué dans 3.3.1 les images sont après stockées en binaire dans la BDD IndexedDB.

Exemple de réponse de l'API Pixabai



4.4 Fonctionnement de get() et get_sans_clef()

Les fonctions **get()** et **get_sans_clef(),** fonctionne de la même manière à l'exception que la fonction **get()** intègre une clé API dans le **headers** de la requête.

```
const RESPONSE = await fetch(url, {
  method: 'GET',
  headers: {
      "apikey": "clef_API",
      "accept": "*/*",
  },
});
```

L'élément le plus important de ces fonctions excepté l'url, est l'ID qui est passé en paramètre. C'est en fonction de cet ID que les réponses seront traitées différemment selon les besoins à l'aide d'un **switch**.

Voici un court extrait

```
            switch(id)
            {
              case "massifs":
                data = await RESPONSE.text(); // ou await
RESPONSE.text()
                localStorage.setItem("data_massifs", data);

console.log(JSON.parse(localStorage.getItem("data_massifs")));
                break;
```

*Voir ligne 173 et 271 pour la fonction **get_sans_clef()** et **get()***

Quant au paramètre **"option"**, il sert surtout à définir le nom par lequel la donnée sera stocké dans le localstorage ou dans la BDD

```
localStorage.setItem("meteo_station_"+option, JSON.stringify(data2));
```

5. Installation et mise à jour

5.1 Installation

A chaque visite du site la fonction **main()** dans /sae302/js/script.js est appelé, si l'application n'est pas déjà installée alors la fonction **onBeforeInstallPrompt(event)** est à son tour appelé par l'écouteur d'événement :

window.addEventListener("beforeinstallprompt", onBeforeInstallPrompt)
ligne 41 /sae302/js/script.js

Cette fonction va afficher une boîte de dialogue à l'aide de la librairie Bootstrap pour permettre à l'utilisateur d'installer la PWA. S' il choisit de l'installer alors la fonction **installPwa()** est appelé par l'écouteur d'événement :

INSTALL_BUTTON.addEventListener("click", installPwa)
Ligne 12 /sae302/js/script.js

La fonction **installPwa()** va installer l'application et fermer la boîte de dialogue grâce à l'instruction : **DIALOG.hide()** *Ligne 68 /sae302/js/script.js*

5.2 Mise à jour

Si l'application est installé, alors à chaque fois que l'application est chargé, comme expliqué au dessus la fonction **main()** dans /sae302/js/script.js est appelé, mais cette fois si elle est lancé en tant qu'application elle lancera directement la fonction **registerServiceWorker()** ou alors si elle est installé mais lancé depuis le navigateur c'est la fonction **onAppInstalled()** qui sera appelé et qui appellera la fonction **registerServiceWorker()** à son tour.

La fonction **registerServiceWorker()** va vérifier si l'application possède un **"ServiceWorker"** *(fichier détaillé dans le 4.3),* si le fichier est présent et bien configuré alors la fonction **caching()** de **/sae302/service_worker.js** comparera les numéros de versions du **"ServiceWorker"** qu'il a récupéré avec celui qui est dans le cache, si le numéro de version n'est pas identique alors la fonction **onStateChange(event)** est appelé et affichera une notification à l'aide de la librairie Bootstrap JS pour dire à l'utilisateur de recharger la PWA.

const toastBootstrap =
bootstrap.Toast.getOrCreateInstance(SNACKBARCONTAINER)
toastBootstrap.show()

Ligne 145 /sae302/js/script.js

5.3 Service Worker

Le Service Worker contient dans la constante **“RESSOURCES”** tout ce que l’application doit stocker en cache :

```
const RESSOURCES = [

"./",
"./index.html",
"./service_worker.js",

"./css/default.css",
"./css/bootstrap-5.3.8-dist/css/bootstrap.min.css",

"./favicon/apple-touch-icon.png",
"./favicon/favicon.ico",
"./favicon/favicon.svg",
"./favicon/favicon-96x96.png",
"./favicon/site.webmanifest",
"./favicon/web-app-manifest-192x192.png",
"./favicon/web-app-manifest-512x512.png",
"./img/logo_app.png",
"./img/logo_map.png",
"./img/poubelles.png",
"./img/etoile.png",
"./img/attention.jpg",
"./img/background.jpg",

"./js/js/bootstrap.bundle.min.js",
"./js/pwa.js"
];
```

A chaque lancement si l’app est installé le **service_worker** va comparer les numéros de versions qu’il à dans son cache et qu’il vient de récupérer. Comme expliqué précédemment, si les numéros de version ne sont pas identiques alors il faut recharger l’application sinon il prend directement les ressources qu’il a dans le cache.

6. Exemple de structure de code

6.1 Structure des notifications

Les notifications sont définis dans le code HTML à l'aide de la librairie Bootstrap avec la structure suivante :

```
<div class="toast-container position-fixed bottom-0 end-0 p-3">
  <div id="notif_hors_ligne" class="toast" role="alert" aria-live="assertive" aria-atomic="true">
    <div class="toast-header">
      <img src="./img/attention.jpg" width="20" height="20" class="rounded me-2" alt="logo attention">
      <strong class="me-auto">Notification</strong>
      <button type="button" class="btn-close" data-bs-dismiss="toast" aria-label="Close"></button>
    </div>
    <div class="toast-body">
      Attention vous êtes actuellement hors ligne.
      Certaines fonctionnalités peuvent ne pas fonctionner ou alors être imprécises.
    </div>
  </div>
</div>
```

Pour les afficher grâce à la librairie Bootstrap il suffit de donner l'instruction suivante en JavaScript :

```
bootstrap.Toast.getOrCreateInstance(NOTIF_OFFLINE).show()
```

(ici la constante NOTIF_OFFLINE correspond à document.getElementById("notif_hors_ligne"), voir l'exemple dans /sae302/js/pwa.js ligne 136)

6.2 Structure des boîtes de dialogues

Comme les notifications, les boîtes de dialogue sont définis dans le code HTML à l'aide de la librairie Bootstrap avec la structure suivante :

```
<div class="modal fade" id="dialog_install" data-bs-backdrop="static" data-bs-keyboard="false" tabindex="-1" aria-labelledby="staticBackdropLabel" aria-hidden="true">
```

```
      <div class="modal-dialog">
        <div class="modal-content">
          <div class="modal-header">
            <h1 class="modal-title fs-5" id="staticBackdropLabel">Installe vite
SkiWeather</h1>
            <button type="button" class="btn-close" data-bs-dismiss="modal"
aria-label="Close"></button>
          </div>
          <div class="modal-body">
            Installez SkiWeather pour être au courant de la météo dans les stations de ski
françaises !
          </div>
          <div class="modal-footer">
            <button type="button" id="dialog_close" class="btn btn-secondary"
data-bs-dismiss="modal">Revenir à la version Web</button>
            <button type="button" id="install_button" class="btn
btn-primary">Installer</button>
          </div>
        </div>
      </div>
    </div>
```

Pour les afficher grâce à la librairie Bootstrap il suffit de donner l'instruction suivante en JavaScript :

DIALOG.show()

et pour les fermer :

DIALOG.hide()

(Ici la constante DIALOG correspond à new bootstrap.Modal(document.getElementById('dialog_install'), voir exemple dans /sae302/js/script.js ligne 55 et 60)

7. Informations supplémentaires

Pour toutes informations supplémentaires, vous pouvez me contacter par mail à l'adresse précisée dans **Contact.**

De plus l'entièreté du code est commenté et trouvable aux l'url suivante :

https://srv-peda2.iut-acy.univ-smb.fr/girodoar/sae302/js/pwa.js
(code pour les fonctionnalitées de l'app)

https://srv-peda2.iut-acy.univ-smb.fr/girodoar/sae302/js/script.js
(code principalement pour les fonctionnalitées lié à l'installation de la PWA)

https://srv-peda2.iut-acy.univ-smb.fr/girodoar/sae302/service_worker.js

8. Contact

Créateur : Arthur GIRODON
Mail : Arthur.Girodon@etu.univ-smb.fr

9. Documentations utiles

Voici la liste des principales documentations utilisées pour la réalisation de ce projet.

Web Speech API - Web APIs | MDN
Geolocation - Les API Web | MDN
Bar Chart | Chart.js
PDF-LIB · Create and modify PDF documents in any JavaScript environment.
Get started with Bootstrap · Bootstrap v5.3
METEO FRANCE – L'API DonneesPubliquesBRA
IndexedDB API - Web APIs | MDN
Plus d'un million d'images libres de droits - Pixabay
🌦️ Docs | Open-Meteo.com