# Miletus : utiliser les applications Web hors ligne

## accès aux fonctions de l'appareil et du système

**Veikko Krypczyk (Allemagne)**

De nos jours, les applications web constituent un standard dans de nombreux domaines. Elles sont exécutées dans un navigateur et peuvent donc être déployées sur presque tous les appareils, y compris un Raspberry Pi. Le framework Miletus offre la possibilité de transformer ces applications web peuvent en applications natives, permettant ainsi leur utilisation hors ligne. Il vous permet également d'accéder aux interfaces du système local, par exemple pour lire et émettre des signaux via les broches GPIO.

Les applications de contrôle des appareils ou composants connectés à un PC ou à un Raspberry Pi doivent être adaptées au système d'exploitation utilisé et nécessitent l'accès aux interfaces du système. Elles sont appelées "applications natives" et sont spécialement créées et exécutées sur le système cible. D'autre part, il existe des applications web, qui, s'exécutant dans un navigateur, peuvent être lancées sur presque tous les systèmes. Cependant, ces applications présentent certaines limites par rapport aux applications natives. L'utilisation hors ligne et l'accès aux interfaces du système ne sont pas facilement possibles. Le **tableau 1** présente une comparaison des principales caractéristiques des applications natives et des applications web.

Pour les applications destinées aux électroniciens, des avantages tels que l'utilisation hors ligne, l'accès aux interfaces du système et l'intégration de pilotes individuels rendent les applications de bureau natives attrayantes. Si vous souhaitez développer une application qui fonctionne sur différents systèmes, tels que Windows, macOS et

**Tableau 1. Caractéristiques importantes des applications natives par rapport aux applications web.**

| Caractéristiques | Application native | Application Web |
|---|---|---|
| Interface utilisateur | native | basée sur HTML, CSS et JavaScript |
| Installation | nécessaire - les fichiers exécutables doivent être copiés sur le système | non nécessaire - les fichiers sont chargés par le navigateur via l'internet |
| Mises à jour | doivent être installées localement sur chaque appareil | gérées de manière centralisée sur le serveur |
| Exécution | directement sur le système | dans le navigateurin the browser |
| Multiplateforme | non — une application distincte doit être créée pour chaque système | oui |
| Fonctionnement hors ligne | oui | non |
| Accès aux fonctions de l'appareil et du système | oui | non |
| Accès aux bases de données installées localement, par exemple SQLite | oui | non |
| Contrôle de matériel spécifique via des pilotes | oui | non |
| Accès au système de fichiers | oui | limité |

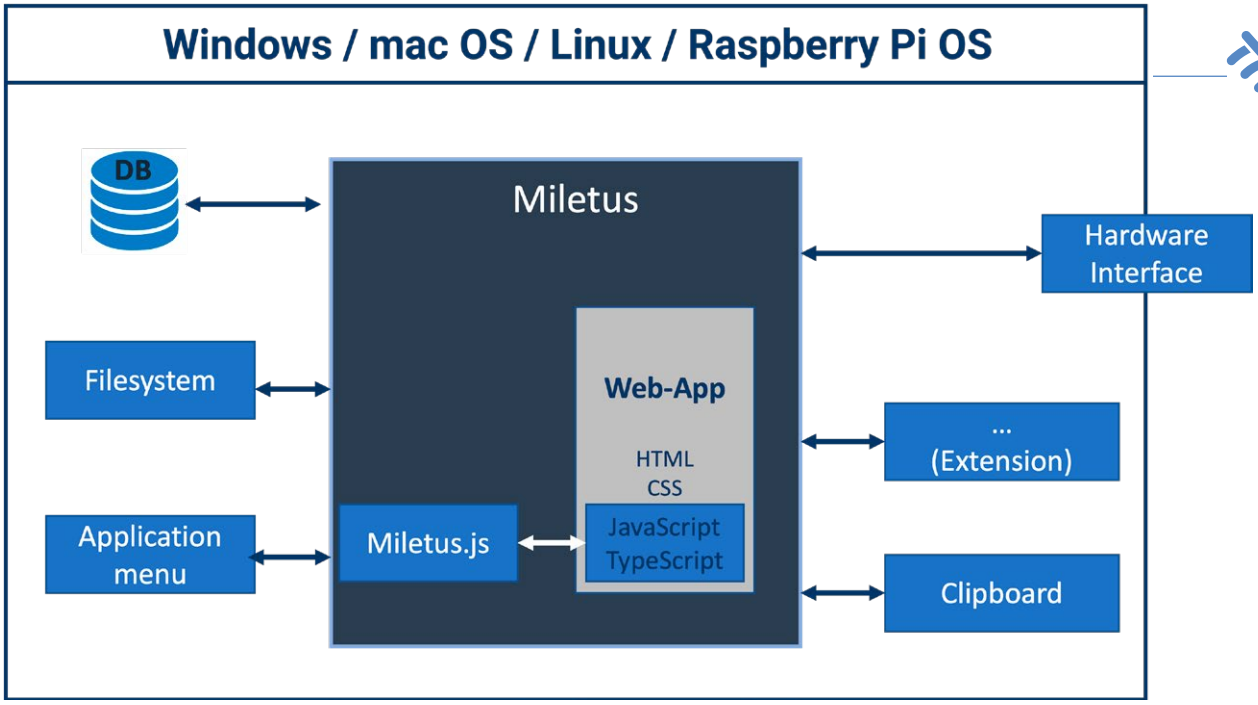


*Figure 1. Architecture d'une application Miletus.*

Raspberry Pi, les choses peuvent rapidement devenir compliquées. Chaque système a ses propres règles pour la création de l'interface utilisateur par exemple. Cela signifie que pour trois systèmes cibles, il faut développer trois applications, triplant ainsi les efforts de développement. De plus, le savoir-faire technique requis varie, car la programmation d'une application Windows native diffère de celle d'une application pour le Raspberry Pi ou pour macOS. Une application web, en revanche, est créée en HTML, CSS et JavaScript et s'exécute dans un navigateur ; en outre, de puissantes bibliothèques telles que *Bootstrap* simplifient la conception de l'interface.

Dans cet article, nous présentons un cadre intéressant appelé Miletus [1]. Avec Miletus, vous pouvez empaqueter une application web, nouvelle ou existante, dans un paquetage d'application pour différents systèmes cibles (y compris le Raspberry Pi) et l'exécuter en tant qu'application native. Ainsi, il est possible d'utiliser l'application hors ligne et d'avoir accès aux fonctions système.

## Le cadre Web Miletus

Avant de nous pencher sur l'utilisation pratique, nous allons présenter les fonctionnalités les plus importantes de Miletus. Les applications créées fonctionneront sur les systèmes d'exploitation Windows, macOS et Linux, y compris le Raspberry Pi OS. L'API de Miletus permet d'accéder aux fonctions système et de l'appareil, au système de fichiers et au matériel externe connecté à l'appareil via des pilotes.

Pour un électronicien, une fonctionnalité particulièrement intéressante est la possibilité d'accéder aux interfaces matérielles du Raspberry Pi, notamment aux ports GPIO, I²C, SPI et UART, ainsi qu'au tampon de mémoire. Ainsi, presque toutes les limitations de l'application web sont éliminées, tout en conservant sa compatibilité multiplateforme. Pour illustrer cela, la **figure 1** présente l'architecture d'une application basée sur le cadre Miletus.

L'intégration de l'application web dans un package d'applications natives pour le système cible correspondant ouvre des horizons pour de nouvelles utilisations. Il s'agit notamment d'applications pour le contrôle de machines ou d'applications de l'Internet des objets (IoT) où il est nécessaire d'envoyer et de recevoir des signaux via les interfaces matérielles du Raspberry Pi. Une autre particularité du cadre est l'extensibilité de l'API. Cela est possible grâce à des "bibliothèques partagées" - sous Windows, par exemple, sous la forme de fichiers DLL (*dynamic linked library*). Si une telle bibliothèque existe pour le système cible, par exemple un pilote pour un périphérique spécifique, il est possible d'intégrer cette bibliothèque dans Miletus et d'accéder a ses fonctions depuis l'application web. Miletus utilise le moteur de navigation standard fourni par le système cible :

- **Windows** : *WebView2*
- **Linux/Raspberry Pi OS** : *WebKitGTK*
- **macOS** : *WebKit* (*Safari*)

On peut généralement supposer que les moteurs de navigation mentionnés sont déjà installés sur les systèmes cibles, de sorte qu'il n'est pas nécessaire que l'application en fournisse d'autres. Le package d'applications à distribuer reste léger, ce qui accélère la distribution de l'application tout en utilisant les ressources du système avec modération. Ceci est particulièrement important pour l'exécution sur le Raspberry Pi. Comme le navigateur existant du système est utilisé, les applications bénéficient aussi automatiquement des mises à jour fonctionnelles et de sécurité du navigateur.

Cependant, vous devez vérifier la configuration requise sur le système cible et vous devrez éventuellement installer des bibliothèques système supplémentaires. Les bibliothèques suivantes doivent être disponibles sur les systèmes cibles :

- **Windows 32/64-bit** : La dernière version du navigateur Edge Chromium doit être installée. En outre, les fichiers *WebView2Loader_x86.dll* (32 bits) ou *WebView2Loader_x64.dll* (64 bits) doivent être copiés dans le dossier *System32* ou *SysWow64*, respectivement. Ces fichiers sont fournis avec l'installation de Miletus.
- **Linux/Raspberry Pi OS** : L'interface graphique *GTK3* est utilisée. Vous pouvez l'installer en ligne de commande avec la commande `sudo apt install libwebkit2gtk-4.0-dev`.
- **macOS** : Aucune autre condition préalable n'est requise.

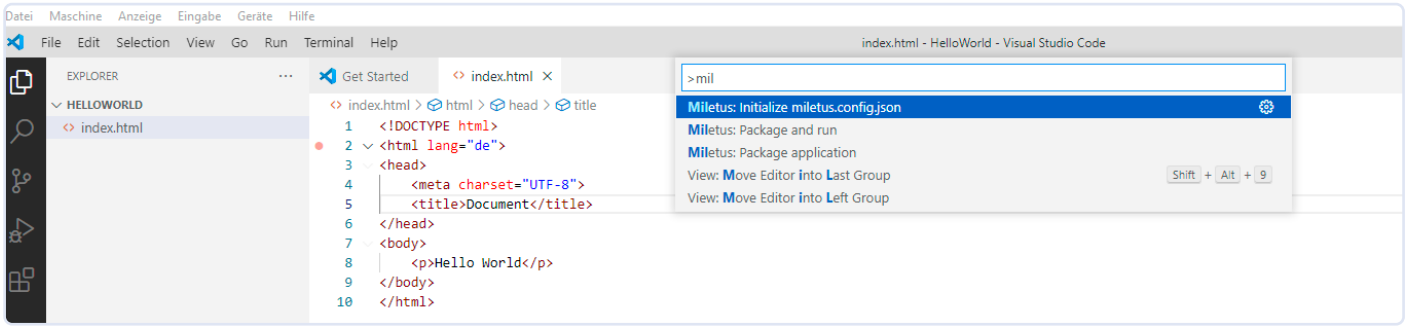


*Figure 2. Commandes pour le Packager Miletus dans Visual Studio Code.*

## Alternative : Electron

Le cadre Electron [2] est également utilisé pour convertir une application web en une application de bureau. Par exemple, l'éditeur Visual Studio Code, utilisé ici, a été développé avec ce framework (c'est-à-dire qu'il s'agit en fait d'une application web). Contrairement à Miletus, Electron fournit son propre moteur de navigation, ce qui signifie que les packages d'applications sont beaucoup plus volumineux. En outre, les applications Electron ne sont pas compatibles avec le Raspberry Pi et l'API ne permet pas d'extensions.

## Installation et premiers essais

Ce dont nous avons besoin :

- **Application Web :** Pour les premiers essais, un simple fichier HTML (*index.html*) qui ne produit que le texte "Hello, world" est suffisant. Nous ajoutons également un bouton pour démontrer le fonctionnement de l'API Miletus (voir ci-dessous).
- **Visual Studio Code** : Cet éditeur est téléchargeable à partir de [3] pour votre propre système d'exploitation.
- **Extension pour Visual Studio Code** : Vous pouvez également la télécharger depuis la page web de Miletus [1] et l'installer manuellement dans Visual Studio Code (fichier *vsix*).
- **Packager (optionnel)** : Vous pouvez le télécharger pour Windows, macOS ou Linux à partir de [4] et l'installer. Il permet d'utiliser le Packager en ligne de commande.

Remarque pour les développeurs web expérimentés : si le gestionnaire de paquets *npm* est installé sur votre système, vous pouvez également installer Miletus avec la commande `npm install miletus`. L'API est alors installée sur le système de développement et vous pouvez l'intégrer dans vos propres projets web.
Voici le fichier *index.html* de notre exemple "Hello, world". Le code source des exemples présentés ici est disponible sur le site web de l'auteur [5].

```
<!DOCTYPE html>
<html lang="de">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <script src="miletus.js"></script>
    <script>
        function save() {
            miletus.dialogs.showSaveDialog();
        }
    </script>
</head>

<body>
    <p>Hello World</p>
     <button onclick="save()">Save to File</button>
</body>
</html>
```

Tout est désormais en place. Si vous avez installé l'extension pour Visual Studio Code, le packager sera configuré automatiquement. Les commandes supplémentaires suivantes seront alors disponibles dans Visual Studio Code (**figure 2**) :

- *Miletus — Package application* : crée les packages d'application pour les systèmes cibles Windows, macOS et Linux / Raspberry Pi OS.
- *Miletus — Initialize config* : crée un modèle pour le fichier de configuration *miletus.config.json*.
- *Miletus — Package and run*: crée les packages pour les systèmes cibles sélectionnés et démarre l'application sur le système de développement. Un Raspberry Pi ne peut pas être utilisé comme environnement de développement ; l'application ne peut y être exécutée qu'après le packaging.

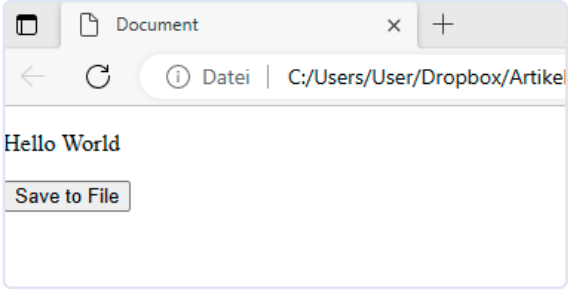


*Figure 3. Une application web avec un bouton.*

Ouvrez le dossier contenant le fichier *index.html*. Vous pouvez ouvrir ce fichier HTML dans votre navigateur pour le tester (**figure 3**).
Nous envisageons maintenant packager cette appli web en une appli de bureau. Le Packager est contrôlé par le fichier de configuration *miletus.config.json*. Le nom et la version de l'application, les systèmes cibles, le point d'entrée de l'application, et bien d'autres détails sont spécifiés ici. La description détaillée des variables de configuration est disponible dans la documentation [6]. Vous pouvez créer le fichier manuellement ou le générer automatiquement avec Visual Studio Code. Pour ce faire, vous pouvez utiliser le raccourci (*[Ctrl + Shift + P]*) et la commande *Miletus: Initialize config*. Complétez les paramètres, c'est-à-dire attribuez un nom à l'application, attribuez le fichier de démarrage (*index.html*) et sélectionnez les systèmes cibles. Le fichier *miletus.config.json* généré ressemble à ceci :

```
{
    "name": "Dialogs",
    "output": "output",
    "debug": true,
    "main": {
        "html": "index.html"
    },
    "target": [
        "win_ia32"
    ],
    "include": [
        "index.html",
        "miletus.js"
    ]
}
```

Les packages d'application destinés aux systèmes cibles sont générés à partir des informations contenues dans le fichier de configuration, par exemple *win_ia32* pour Windows 32 bits. Ceci est réalisé grâce à la commande *Miletus: Package application*. Pour Windows, il s'agit d'un fichier *exe*, pour macOS d'un dossier *.app* et du fichier *entitlements* nécessaire pour gérer les autorisations et pour Linux d'un fichier exécutable et d'un fichier *.desktop* (**figure 4**).
Passons maintenant à la première exécution de l'application web en tant qu'application de bureau sur le système cible. Pour ce faire, nous devons copier les fichiers exécutables sur le système cible, c'est-à-dire :

- **Windows** : Exécutez le fichier *.exe*. Si vous travaillez avec Visual Studio Code sous Windows, vous pouvez exécuter l'application directement avec la commande *Miletus: Package and run.*
- **Linux** : Copiez le dossier *Output* sur le système Linux. Attribuez les droits aux fichiers avec la commande `sudo chmod -R 755 /[Path of the Application]`. Lancez ensuite l'application.
- **macOS** : Vous devez copier les fichiers de l'application depuis le dossier *Output*, attribuer les droits, puis vous pouvez lancer l'application. Remarque : sous un système macOS basé sur ARM, la signature du code de l'application est nécessaire.

Vous pouvez voir l'application "Hello, world" en cours d'exécution dans la **figure 5**.
Remarque : si vous recevez un message d'erreur ou si une fenêtre vide s'affiche, vérifiez les conditions requises pour votre système.

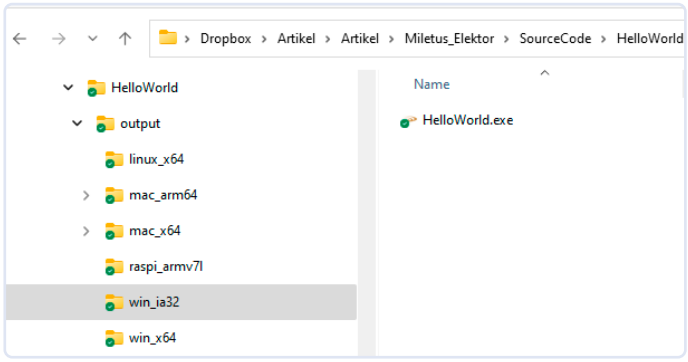


*Figure 4. Packages d'applications pour les différents systèmes cibles.*

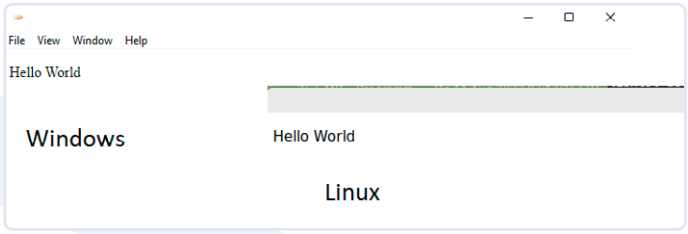


*Figure 5. Application "Hello, world" sur Windows et Linux.*

## Utiliser l'API Miletus

Une application "Hello World" ne suffit pas pour démontrer l'utilité du cadre Miletus. Nous allons donc explorer ses fonctionnalités avec un exemple simple. L'interface utilisateur de l'API Miletus offre, entre autres, les fonctions suivantes :

- **Dialogues** : L'API Miletus fournit des fonctions pour afficher des boîtes de dialogue pour ouvrir et enregistrer des fichiers (`showOpenDialog(…)`, `showSaveDialog(…)`), signaler des erreurs (`showErrorDialog(…)`) et afficher des messages (`showMessageBox(…)`). Les fonctions sont accessibles dans le code JavaScript ou dans le code source *TypeScript*. Vous devez d'abord inclure la bibliothèque correspondante *miletus.js*. Ensuite, il est possible d'appeler, par exemple, la boîte de dialogue *Save as…* avec la fonction :

```
miletus.dialogs.showSaveDialog()
```

Pour ce faire, cette ligne de code est intégrée dans une fonction nommée `save()`, qui est à son tour déclenchée par l'événement `onclick()` d'un bouton. La **figure 6** montre l'appel de la boîte de dialogue du fichier.

- **Accès aux fichiers :** Miletus propose des méthodes pour charger et écrire des fichiers texte et binaires `loadTextFile(…)`, `saveTextFile(…)`, `loadBinaryFile(…)`, et `saveBinaryFile(…)`. Il existe des méthodes pour le suivi des modifications des fichiers, à savoir `watchFile(…)`, `removeWatch(…)` et `removeAllWatches(…)`, ainsi que la méthode `startDrag(…)` pour les opérations de glisser-déposer. Des méthodes permettant d'afficher tous les éléments d'un dossier de fichiers (`openFile(…)`) et de supprimer des fichiers (`moveToBin(…)`) sont également disponibles.

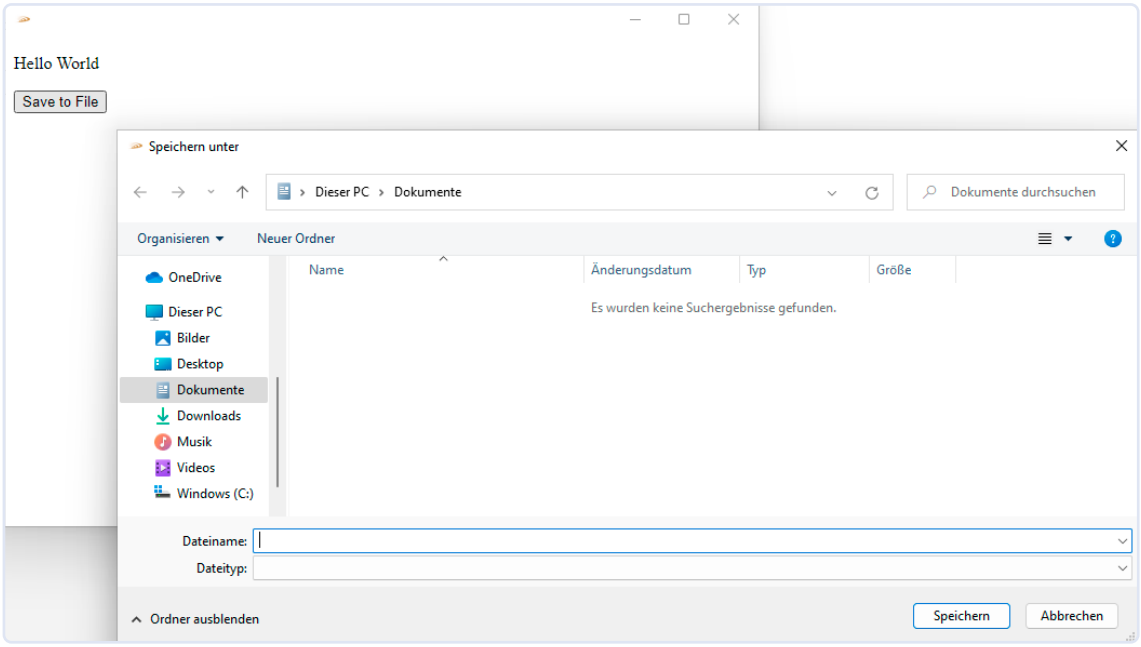


*Figure 6. Dialogue de fichier dans Windows - appelé via l'API Miletus.*

- **Menus de l'application :** Vous pouvez ajouter des éléments de menu à la fenêtre de l'application. Un menu peut comporter des sous-menus. Une réaction peut être déclenchée lorsque l'utilisateur interagit avec ces éléments. Un menu précédemment défini est attribué à l'application via `application.setMenu(menu)`.
- **Fonctions système :** Vous pouvez charger un URL externe à afficher dans le navigateur via `shell.openURL(…)`. Vous pouvez executer une commande système via `execute(…)`.
- **Accès aux bases de données :** Les bases de données suivantes sont prises en charge : SQLite, MySQL, MSSQL, PostgreSQL, MS Access (Windows), Firebird et Interbase. La connexion à la base de données peut simplement être établie via une "chaîne de connexion" sous la forme suivante :

```
let db = new DataBase('sqlite', {database: 'path_to_my/sqlitedb.db'})
```

- **Registre Windows :** Sous Windows, vous pouvez accéder au registre, c'est-à-dire lire, écrire et vérifier les valeurs clés.

D'autres fonctions de l'API Miletus (fenêtre d'application, lecture et écriture de fichiers *ini*, messages système) sont décrites dans la documentation. Une fonctionnalité remarquable est l'accès aux interfaces matérielles du Raspberry Pi que nous aborderons dans la section suivante.
ntation. A special feature is access to the hardware interfaces of the Raspberry Pi. We will look at how this works in the next section.

## Une application pour le Raspberry Pi

Nous souhaitons maintenant créer une application capable de lire les capteurs connectés au Raspberry Pi. L'application est développée initialement en tant qu'application web puis sera convertie en un package d'application pour le Raspberry Pi avec Miletus. Nous utilisons la configuration de test suivante :

- Raspberry Pi version 2 ou supérieure
- Capteur environnemental BME280 de Bosch Sensortec (voir le **tableau 2** pour les spécifications techniques) ; pour cet exemple, nous utilisons une carte d'expansion de Waveshare.
- Un capteur connecté au Raspberry Pi via l'interface I²C
- Un écran, une souris et un clavier connectés au Raspberry Pi.

Le capteur BME280 mesure la température, l'humidité et la pression atmosphérique. Nous souhaitons lire ces valeurs avec JavaScript et les afficher dans l'application. Le capteur est connecté directement aux broches de la carte Raspberry Pi. Seuls quatre fils sont nécessaires :

**Tableau 2. Caractéristiques techniques du capteur BME280 [7].**

| Caractéristiques | Valeurs |
|---|---|
| Tension de fonctionnement | 5 V/3 V |
| Interface | I2C/SPI |
| Plage de température | -40 à 85 °C, résolution 0.01 °C, précision ±1 °C |
| Humidité de l'air | 0 à 100 RH, résolution 0,008 % RH, précision ±3 % RH, temps de réponse 1 s, retard ≤2 % RH |
| Pression atmosphérique | 300 à 1 100 hPa, résolution 0.18 Pa, précision ±1 hPa |
| Dimensions | 27 mm × 20 mm × 2 mm |

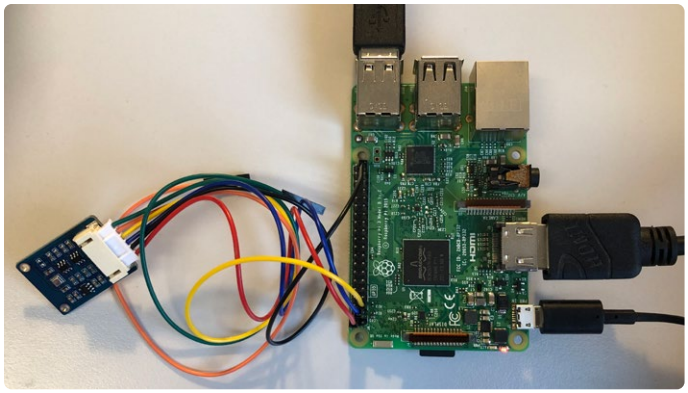

*Figure 7. Montage expérimental avec un Raspberry Pi et un capteur BME280.*

### Listage 1. index.html of the sensor app.

```
<!DOCTYPE html>
<html>
    <head>
        <title></title>
        <script src="../js/miletus.js"></script>
        <script src="../js/bme280.js" defer></script>
        <link rel="stylesheet" type="text/css" href="../css/style.css" />
    </head>
    <body>
        <h1>Weather Station</h1>
        <div class="container">
            <div class="child">
                <img class="image" width="100" src="../img/temp.png">
                <p class="label">Temperature</p>
                <div class="value" id="temperature"></div>

            </div>
            <div class="child">
                <img class="image" width="100" src="../img/air.png">
                <p class="label">Pressure</p>
                <p class="value" id="pressure"></p>

            </div>
            <div class="child">
                <img class="image" width="100" src="../img/humidity.png">
                <p class="label">Humidity  </p>
                <p class="value" id="humidity"></p>

            </div>
           <div class="buttonscontainer">
               <button id="start-reading" class="btn">Start reading</reading></button>
               <button id="stop-reading" disabled class="btn">Stopp reading </button>
           </div>
        </div>
        <img class="logo" src="../img/logo.png">
    </body>
</html>
```

VCC (rouge) et GND (noir) du module BME280 sont connectés aux broches 2 et 39 du Raspberry Pi. SDA (bleu) et SCL (jaune) sont connectés aux broches 3 et 5 du Raspberry Pi. La **figure 7** montre le montage de test.
Nous installons la dernière version de Raspberry Pi OS sur le Raspberry Pi sur une carte SD depuis un PC. La communication entre les deux composants s'effectue via l'interface I²C, qui doit être activée dans le Raspberry Pi OS. Pour ce faire, ouvrez un terminal sur le Raspberry Pi et entrez la commande :

```
sudo raspi-config
```

Dans l'étape suivante, sélectionnez l'option :

*Interfacing Options -> I²C*

pour activer le pilote I²C. Après avoir sauvegardé les paramètres et redémarré le Raspberry Pi en utilisant :

```
sudo reboot
```

Il est maintenant possible d'échanger des données via l'interface I²C. Les étapes préliminaires sont terminées. Passons maintenant à la programmation. Le projet se compose des fichiers suivants :

- *index.html* : Ce fichier est le point de départ de l'application web (**listage 1**). Le fichier *index.html* contient le fichier CSS (*style.css*) et les deux fichiers JavaScript *miletus.js* et *bme280.js*.
- *style.css* : La mise en page et la conception de l'application web sont spécifiées en CSS.
- *miletus.js* : contient l'API Miletus.
- *bme280.js* : définit toutes les fonctions de communication avec le capteur.

Tout d'abord, nous avons créé un design initial minimaliste (**figure 8**). Vous pouvez utiliser n'importe quel outil pour ce faire. Une simple esquisse dessinée à la main peut également suffire. Ensuite, nous avons implémenté l'application web avec HTML (structure), CSS (mise en page, conception) et JavaScript (fonctions).
Vous pouvez lancer l'application dans un navigateur sur n'importe quel système (**figure 9**) afin de tester son design. Cependant, la fonction-

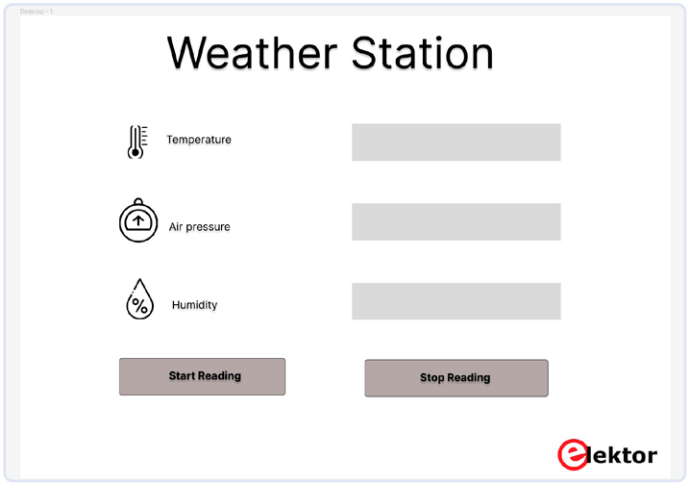


*Figure 8. Prototype de l'application (layout et design).*

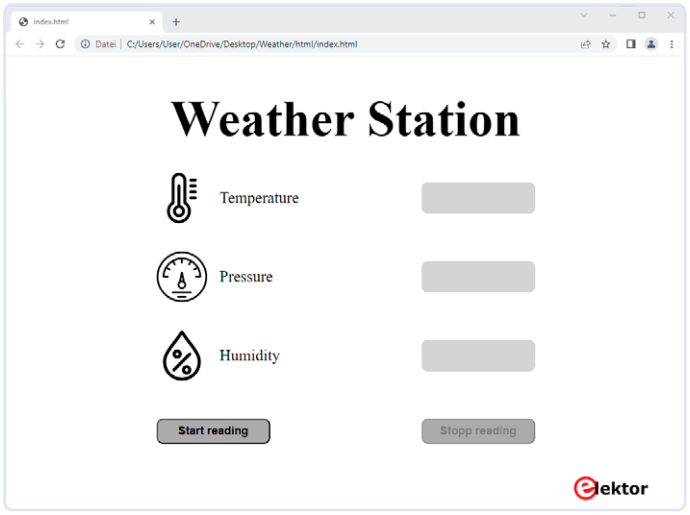


*Figure 9. Application météo fonctionnant dans le navigateur.*

nalité clé, c'est-à-dire la lecture des données du capteur, ne fonctionne que sur le Raspberry Pi.
Examinons de plus près le code source. Un bouton intitulé *Start reading* est défini dans le fichier HTML :

```
<button class="start" id="start-reading">Start reading</
button>
```

Lorsque vous cliquez sur ce bouton, les données du capteur sont censées être lues. L'événement `onclick` est défini dans le fichier *bme280.js* à cette fin :

```
document.getElementById("start-reading").onclick =
startReading;
```

Il s'agit de la fonction JavaScript `startReading()` du **listage 2**.
Les données du capteur sont calibrées, le bouton *Start* est désactivé, le bouton *Stop* est activé, les relevées sont lus en continu. Cette opération est conforme aux spécifications du capteur BME280, détaillées dans la documentation et dans les exemples [7] et [8]. Par exemple, en examinant la fonction `readSensorData()`, nous trouvons la ligne de code suivante :

```
let rawValues = await i2c.readBuffer(0xFA, 3)
```

Elle est utilisée pour lire les données de température. L'objet `i2c` est défini comme suit :

```
i2c = new miletus.I2C()
```

L'API Miletus permet d'accéder à l'interface I²C du Raspberry Pi. Cela nous permet de lire les données du capteur dans l'application web avec JavaScript. Créons maintenant le fichier de configuration *miletus.config.json* pour notre projet dans Visual Studio Code comme décrit ci-dessus (commande : *Miletus: Initialize config*). Saisissez également les chemins aux fichiers d'images qui doivent apparaître dans l'interface utilisateur (par exemple, les icônes).
Nous pouvons alors générer le package d'application pour le Raspberry Pi (commande : `Miletus: Package application`). Nous copions ensuite les fichiers d'application générés sur le Raspberry Pi, par exemple en utilisant une clé USB. L'attribut "exécutable" doit être défini pour le fichier d'application. Ensuite, démarrez l'application sur le Raspberry Pi. Elle devrait s'ouvrir dans une fenêtre séparée. Cliquez sur le bouton pour lire et afficher les données du capteur (**figure 10**).

## Autres possibilités avec le Raspberry Pi

Nous avons étudié la communication via l'interface I²C. Avec Miletus, nous pouvons également accéder aux interfaces suivantes sur le Raspberry Pi [9] :

- *GPIO* : Nous pouvons lire et écrire des données via les 26 broches GPIO.
- *SPI* : Ce bus série nécessite trois fils pour la communication et repose sur le principe maître/esclave.
- *UART* : Ce bus est notamment utilisé pour la communication avec les interfaces RS-232 ou RS-485, par exemple pour le transfert de données avec des microcontrôleurs.

Il est également possible d'accéder à la mémoire tampon. Cela permet de lire et d'écrire la mémoire tampon au niveau du shell de l'application. Voyons un exemple d'échange de données via des GPIO. Importez la bibliothèque *gpio* en utilisant :

```
import { gpio } from 'miletus'
```

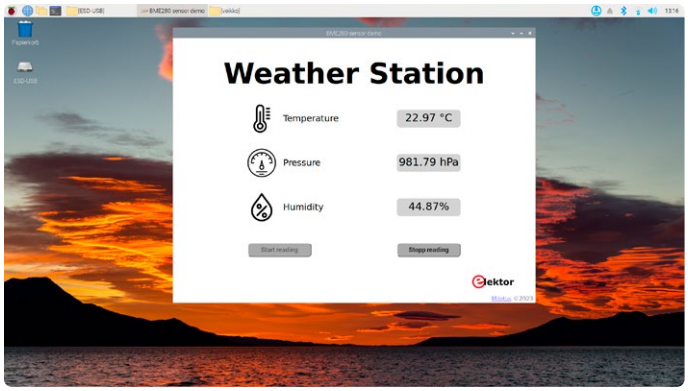


*Figure 10. Application météo sur le Raspberry Pi.*

### Listage 2. Function for reading out the sensor data.

```
async function startReading() {
    await readCalibrationData();
    document.getElementById("start-reading").disabled = true;
    document.getElementById("stop-reading").disabled = false;
    doRead = true;
    while (doRead) {
        await readSensorData();
        await new Promise(resolve => setTimeout(resolve, 1000));
    }
}
```

### Produits

- **Raspberry Pi 4 B (2 GB RAM)**
  www.elektor.fr/18965
- **Raspberry Pi 5 (4 GB RAM)**
  www.elektor.fr/20598

Nous pouvons maintenant utiliser les méthodes `setPin(…)` pour définir le mode de la broche (entrée ou sortie), utiliser `write(…)` pour activer ou désactiver une broche (signal haut ou bas) et utiliser `read(…)` pour lire une broche. Pour garantir que l'interface utilisateur de l'application web reste réactive pendant l'échange de données avec l'interface matérielle, l'accès doit être asynchrone. Par exemple, l'accès doit être asynchrone :

```
await gpio.setPin(1, 'read')
```

Ceci met la broche du port GPIO en mode lecture.

## Extensibilité

Le cadre Miletus offre une grande flexibilité d'extension. Vous pouvez charger n'importe quelle bibliothèque de Miletus et ensuite accéder aux fonctions qu'elle exporte depuis l'application web. Cela inclut l'accès aux fichiers *.dll* (Windows), *.dylib* (macOS) et *.so* (Linux/Raspberry Pi). Ainsi, les fonctions système des systèmes cibles peuvent être utilisées dans une application web packagée avec Miletus, par exemple pour contrôler le matériel connecté au PC via des pilotes système. Vous pouvez trouver plus d'informations sur ce sujet à l'adresse suivante : [10].

## Réduction de l'effort de programmation

La méthode, qui peut sembler inhabituelle, et qui consiste à transformer une application web en une application native offre de nombreux avantages. Elle permet à l'application de s'exécuter sur tous les systèmes (multiplateforme), de se comporter comme une application native, d'être utilisée hors ligne et d'avoir accès aux fonctions du système. Sur le Raspberry Pi, cela permet à l'application d'envoyer et de recevoir des données via les interfaces. Cela signifie que vous pouvez utiliser une application web pour contrôler des circuits électroniques. Cette approche peut considérablement simplifier et réduire l'effort de programmation pour de nombreuses applications, surtout si vous souhaitez couvrir plusieurs systèmes simultanément.

220616-04


### À propos de l'auteur

Veikko Krypczyk est développeur de logiciels, formateur et auteur spécialisé. Il partage sa passion et son expertise lors de séances de coaching, de séminaires, de cours de formation et des ateliers. Vous pouvez demander des ateliers personnalisés ou obtenir du soutien en visitant *https://larinet.com* et consulter l'agenda des événements à venir.


### Questions ou commentaires ?

Contactez elektor (redaction@elektor.fr).

### LIENS


[1] Miletus : https://miletus.org
[2] Electron : https://electronjs.org
[3] Visual Studio Code : https://code.visualstudio.com
[4] Miletus : téléchargement du Packager : https://miletus.org/download.html
[5] Téléchargement du code d'exemple : https://wp.larinet.com/?page_id=663
[6] Miletus : variables de configuration du Packager : https://miletus.org/doc/gettingstarted/packager
[7] BME280 Fiche technique : https://waveshare.com/w/upload/9/91/BME280_datasheet.pdf
[8] BME280 Bibliothèque du pilote : https://github.com/BoschSensortec/BME280_driver/blob/master/bme280.c
[9] Miletus : Accès aux GPIO: https://miletus.org/doc/reference/gpio
[10] Extensions de Miletus : https://miletus.org/doc/gettingstarted/extensibility