# Architecture domotique

L’application d’interface fait un certain nombre d’hypothèses sur la manière dont l’ensemble du système domotique sera installé. Cette première partie décrit cette architecture

## Les composants



### Les capteurs / actionneurs

Les capteurs / actionneurs (appelés ensuite seulement capteurs) sont disposés partout dans la maison. Il remontent de l’information et peuvent être commandés à distance.

Exemples de capteurs:

* Thermomètre: lit la température
* Thermostat :
  + Lit la température
  + Actionne une température de consigne
  + Poussoir : bouton poussoir que l’on peut appuyer. Ne remonte aucune information
* Variateur, par exemple ouverture de volets roulants ou variateur lumineux :
  + Lit une valeur entre 0% et 100%
  + Commande la variation entre 0% et 10%
* Caméra:
  + Lit un flux vidéo
  + Commande la rotation de la caméra

### Le serveur domotique

Le serveur domotique a trois composants principaux:

* Le **contrôleur** qui communique avec les capteurs. Il transforme les données physiques des capteurs en données informatiques qui peuvent être appelées à partir d’un programme. La transformation se fait dans les deux sens : lecture et action.
* Le **serveur PHP** qui peut lire les informations issues des capteurs (via le contrôleur) pour les envoyer sur l’ordinateur de l’utilisateur. Réciproquement, il interprète les clics de l’utilisateur pour envoyer des commandes aux capteurs
* La **base SQL** qui permet de stocker les données de configuration

### L’application IHM

L’application IHM est « poussée » par le serveur PHP. Mais comme elle fait appel à pas mal de javascript, elle est considérée comme un composant à part entière.

Lorsque la page principale de l’application de contrôle est appelée, on ne fait plus de lien url vers d’autres pages, on reste sur cette page en la modifiant dynamiquement via le javascript.

On aurait pu utiliser des composants standards comme jquery pour simplifier la programmation, mais pour éviter d’introduire un nouveau macro langage, on reste en javascript natif.

## Architecture logicielle de la maquette domotique

L’application IHM a été développée sans connaître le fonctionnement du contrôleur ni le modèle de données du système domotique. Ce fonctionnement côté serveur a été simulé de manière simplifiée. En particulier, on n’utilise pas de base MySql alors que ce sera indispensable dans l’application finale.

Attention: Aucune sécurité n’a été mise en place !

### Modèle de données

Le modèle de données suppose que l’utilisateur, une fois identifié, peut accéder à ses maisons.

Chaque maison a plusieurs étages

Chaque étage à plusieurs pièces

Les capteurs sont dans les pièces



### Simulation de la base de données

Dans la maquette, on n’utilise pas de base SQL mais une structure avec des tableaux statiques non modifiables. Pour changer la configuration, il faut changer le code source !

La syntaxe de déclaration statique permet de voir tout de suite la structure des tables. La syntaxe de déclaration PHP est la suivante :

$table = [

id => [

"nom" => "Dupont",

"age" => 12

],

];

Se traduira facilement en une table SQL de 3 colonnes :

* id : index
* nom : string
* age: int

Les table de la base sont déclarée dans le fichier **base.php**.

### Les maisons

Il y a deux maisons qui appartiennent toutes les deux à l’utilisateur 1.

Il n’y a pas de table des utilisateurs et aucun contrôle n’est fait sur l’authentification de l’utilisateur

$maisons = [

1 => [

"nom" => "Appartement",

"user" => 1

],

2 => [

"nom" => "Résidence secondaire",

"user" => 1

]

];

### Les étages

La propriété *maison* donne l’index de la maison auquel appartient l’étage

Le *niveau* permet de trier les étages de haut en bas

Le *plan* est le nom de l’image qui contient le plan de l’étage.

Les plans sont placés dans le répertoire **plans**.

$etages = [

1 => [

"maison" => 1,

"niveau" => 0,

"nom" => "Appartement",

"plan" => "plan0.jpg"

],

2 => [

"maison" => 2,

"niveau" => 0,

"nom" => "Rez de chaussée",

"plan" => "plan1.jpg"

],

3 => [

"maison" => 2,

"niveau" => 1,

"nom" => "Etage",

"plan" => "plan2.jpg"

],

4 => [

"maison" => 2,

"niveau" => 2,

"nom" => "Combles",

"plan" => "plan3.jpg"

]

];

### Les pièces

La propriété *etage* donne l’index de l’étage auquel appartient la pièce

La propriété *coord* contient les coordonnées du polygone qui dessine la pièce sur le plan

Commentaire

* Les coordonnées sont dans l’ordre x0,y0, x1, y1…
* Ce n’est pas la peine de répéter le premier point, le polygone est fermé tout seul
* Les coordonnées sont calculées pour un plan de 600 x 400. Pas de fonction de redimensionnement
* La page **dessin\_pieces.html** permet de dessiner les pièces et de récupérer les coordonnées du polygone

$pieces = [

1 => [

"etage" => 1,

"nom" => "Palier",

"coord" => "382,348, 382,252, 276,252, 277,173, 380,175, 380,136, 221,134, 185,160, 186,196, 221,226, 218,345"

],

2 => [

"etage" => 1,

"nom" => "Chambre 2",

"coord" => "220,348, 223,202, 26,201, 26,346"

],

3 => [

"etage" => 1,

"nom" => "Chambre 3",

"coord" => "219,155, 219,28, 31,26, 31,158"

],

( ... )

### Les capteurs

La propriété *piece* donne l’index de la pièce à laquelle appartient le capteur

Le *numero* est le numéro du capteur tel que le contrôleur le connaît. Lorsqu’il y a plusieurs capteurs du même type, le numéro permet au contrôleur de savoir auquel il doit s’adresser.

La *categorie* indique le type de capteur. Pour ajouter un nouveau type de capteur, il faut lui donner un nom de catégorie et développer une page php spécifique. La mécanique est décrite plus tard.

*icone* et *coord* permettent de placer une icône de capteur sur la plan. Cette possibilité n’a pas été développée dans cette maquette.

$capteurs = [

1 => [

"piece" => 3,

"numero" => "adr\_1",

"nom" => "Eclairage principal",

"categorie" => "poussoir",

"icone" => "none",

"coord" => "50, 50"

],

2 => [

"piece" => 3,

"numero" => "adr\_2",

"nom" => "Température",

"categorie" => "thermometre",

"icone" => "none",

"coord" => "100, 50"

],

3 => [

"piece" => 3,

"numero" => "adr\_3",

"nom" => "Thermostat",

"categorie" => "thermostat",

"icone" => "none",

"coord" => "150, 50"

],

( ... )

## Consultation de la base

La base est consultée en appelée avec la page **serveur.php** en ajoutant :

* Le nom de la table que l’on veut
* Avec l’identifiant du parent

Quelques exemples :

* Maisons de l’utilisateur 1 : <http://localhost/domo/serveur.php?maisons=1>
* Etages de la maison 1 : <http://localhost/domo/serveur.php?etages=1>
* Pièces de l’étage 1 : <http://localhost/domo/serveur.php?pieces=1>
* Capteurs de la pièce 1 : <http://localhost/domo/serveur.php?capteurs=1>

La première requête retourne:

{"1":{

"nom": "Appartement",

"user": 1},

"2":{

"nom": "Résidence secondaire",

"user": 1}}

Les appels sont faits en javascript à l’intérieur de la page principale mais il est possible d’entrer directement ces adresses dans le navigateur pour débeuguer.

## Fonctionnement de la page principales

La page principale est **index.php**. Elle permet de sélectionner une maison puis un étage.

Lorsqu’une maison est sélectionnée, le premier étage est automatiquement sélectionné

Lorsqu’un étage est sélectionné, l’utilisateur peut sélectionner graphiquement une pièce sur le plan

Lorsqu’il sélectionne une pièce, les informations et actions de tous les capteurs sont affichés dans une troisième zone.

Les trois zones principales sont appelées en javascript:

* **leftZone** : le menu de sélection maison > étage
* **centerZone**: la zone d’affichage des plans
* **rightZone**: la zone d’affichage des capteurs

Il faut créer trois éléments HTML avec les identifiants menu\_bar, middle et sidebar. Le code d’initialisation de la page fait correspondre ces zones aux trois variables. La zone de navigation vient s’ajouter pour afficher là où on en est.

leftZone = document.getElementById("menu\_bar");

centerZone = document.getElementById("middle");

rightZone = document.getElementById("sidebar");

navZone = document.getElementById("navigation");

Dans la maquette, le code HTML est le suivant (les 4 sections sont mises en évidence):

<div id="pagewrap">

<header>

<h1>Maquette domotique</h1>

</header>

<section id="navigation"></section>

<section id="menu\_bar">

<h2>Domotique</h2>

<ul id="menu">

</ul>

</section>

<section id="middle">

<img id="plan"/>

</section>

<aside id="sidebar"> &nbsp </aside>

</div>

## Principe de navigation

La navigation classique charge une nouvelle page lorsque l’utilisateur clique sur un lien :

<a href="page.php">Le lien vers une page</a>

L’inconvénient est que la page est rechargée complétement ce qui peut être lent et lourd. Par ailleurs, d’un point de vue javascript, il faut « repartir de zéro » en initialisant toutes les variables de contexte.

C’est pourquoi un clic va déclencher une fonction javascript qui va juste changer ce qui est nécessaire sur la page. Le principe consiste d’abord à indiquer quelle fonction appeler lorsque l’utilisateur clique :

<a onclick=toto()>Appeler toto</a>

Lorsque l’utilisateur va cliquer, la fonction javascript sera appélée.

Pour changer dynamiquement, javascript doit récupérer une zone précise de la page. Pour cela il utiliser principalement la fonction **document.getElementById(id)**. Id est l’identifiant de l’élément qui est indiqué dans le code html. Dans l’exemple ci-dessous, **document.getElementById("main")** va retourner toute la division :

<div id=main>

<h1>Exemple de page dynamique</h1>

</div>

Javascript disposant de l’élément, il peut changer son code HMTL en écrivant dans la propriété innerHTML. Le fonctionnement type est donné dans l’exemple suivant :

<html>

<script>

function toto() {

element = document.getElementById("variable");

element.innerHTML = "CHANGEMENT EFFECTUE"

}

</script>

<div id=main>

<h1>Exemple de page dynamique</h1>

<p>Le code <b><span id="variable">ICI</span></b> va être changé avec le clic.</p>

<a onclick=toto()>Changer la zone variable</a>

</div>

</html>

## Récupération des données en javascript

Les valeurs à afficher dynamiquement sur la page dépendent du choix de l’utilisateur. Par exemple, il faut afficher le plan de l’étage qu’il a choisi. Pour ce faire, la fonction javascript doit récupérer des informations auprès du serveur en appelant la page **serveur.php**.

Récupérer des données en javascript n’est pas intuitif car si javascript peut appeler une adresse url, la fonction d’appel n’attend pas le résultat et revient à l’appelant sans rien retourner d’intéressant.

La lecture des données se fait donc avec deux fonctions :

* **Appel**: appel l’adresse url qui retournera les données voulues plus tard
* **Retour**: traitement du résultat lorsque les données arrivent

L’exemple suivant lit la liste des maisons et affiche le résultat. La page utilise le fichier **request.js** qui inclut la fonction **makeHttpRequest**.

<html>

<head>

<script type="text/javascript" src="js/request.js" ></script>

</head>

<body>

<script>

*// Fonction appelée lorsque les résultats arrivent*

*// Se contente d'imprimer le résultat*

function traitementRetour(text) {

document.write(text);

}

*// Lance la requête de consultation des données*

function litDonnees(url, fonctionRetour) {

var requete = makeHttpObject();

requete.open("GET", url, true);

requete.send(null);

requete.onreadystatechange = function() {

if (requete.readyState == 4) {

if (requete.status == 200)

return fonctionRetour(requete.responseText);

}

};

}

*// Consulte les données*

litDonnees("serveur.php?maisons=1", traitementRetour);

</script>

</body>

</html>

La fonction **litDonnees** a deux paramètres :

* **url**: l’url qui retournera des données
* **fonctionRetour**: la fonction à appeler lorsque le résultat sera arrivé

La fonction retour se contente d’écrire le résultat sur la page.

## Interprétation du json

La bibliothèque request.js inclut la fonction appelServeur qui appelle l’url serveur.php avec le paramètre key=value. Cette fonction s’occupe en plus s’occupe de transformer le json en tableau javascript.

function appelServeur(key, value, retour)

* **key**: le nom du paramètre get
* **value**: la valeur du paramètre
* **retour**: la fonction à appeler lorsque le résultat est là. Cette fonction retour accepte seul paramètre qui est un tableau javascript

L’exemple suivant montre comment on lit en javascript le contenu de la table des maisons.

<html>

<head>

<script type="text/javascript" src="js/request.js" ></script>

</head>

<body>

<script>

*// Fonction appelée lorsque les résultats arrivent*

*// Se contente d'imprimer le résultat*

function traitementTableauRetour(table) {

for (id in table) {

document.write(id + ' --&gt; ' + table[id]["nom"] + '<br/>');

}

}

*// Consulte les données*

appelServeur("maisons", "1", traitementTableauRetour)

</script>

</body>

</html>

**traitementTableauRetour** reçoit le tableau de toutes les maisons. Dans cet exemple, la fonction se contente de lister le contenu.

La fonction appelServeur lance la requête **serveur.php ?maisons=1**. Lorsque le résultat arrive, elle transforme le json reçu en tableau javascript qu’elle utilise dans l’appel de **traitementTableauRetour**.

## Navigation dans la maquette

La sélection d’une maison, d’un étage, d’une pièce se fait avec à chaque fois un couple de fonctions :

* selectMaison -> modeSelectionMaison
* selectEtage -> modeSelectionEtage
* selectPiece -> modeSelectionPiece

La première fonction est appelée lorsque l’utilisateur clique sur un lien. La fonction va lire les données en indiquant que le retour des données doit être traité par la seconde fonction.

La seconde fonction met à jour l’affichage avec les données reçue.

### modeSelectionMaison

Dans ce mode, les maisons disponibles sont affichées dans le menu de gauche. Si l’utilisateur clique sur le nom d’une maison, la fonction **selectEtage** est appelée.

**selectMaison** est appelée au chargement de la page ce qui fait que **modeSelectionMaison** est appelée dans la foulée.

**modeSelectionEtage**

Dans ce mode, les étages de la maison choisie sont affichés dans le menu de gauche. Par défaut le premier étage est sélectionné.

Le plan de l’étage est affiché au centre et l’utilisateur peut cliquer sur la pièce dont il veut afficher les capteurs.

### modeSelectionPiece

Dans ce mode, une pièce de l’étage a été sélectionnée. Les capteurs de la pièce sont affichés dans la zone de droite.

## Les capteurs

Les capteurs d’une pièce sont affichés dans un tableau dans la zone de droite.

L’affichage dépend du type de capteur (sa catégorie). Les deux dernières sections expliquent comment ajouter un nouveau capteur.

# Les fichiers de la maquette

## Répertoire racine

|  |  |
| --- | --- |
| **Fichier** | **Description** |
| index.html | Le fichier principal appelé par défaut |
| domo.css | La feuille de style de index.html |
| dessin\_pieces.html | Utilitaire pour dessiner les pièces sur un plan |
| base.php | Déclaration des tables simulant une base SQL |
| serveur.php | Accède aux tables et retourne le résultat au format json |

## Fichiers javascript

Les fichiers javascript sont placés dans le sous-repertoire **js** :

|  |  |
| --- | --- |
| **Fichier** | **Description** |
| erreurs.js | Affiche un message d’erreur sous forme de bandeau rouge en haut de la page. A utiliser pour afficher un message à l’utilisateur. |
| debug\_dump.js | Permet d’afficher des informations dans une zone de la page. La fonction writeDebug(text) ajoute du texte de debugage. Si la zone n’est pas précisée à l’initialisation, la fonction writeDebug ne fait rien. Cela permet d’avoir facilement une application propre tout en gardant la possibilité de rapasser rapidement en mode debug. |
| request.js | Gestion de la mécanique lecture des données 🡪 traitement du retour. |
| capteurs.js | Les fonctions utilisées pour l’affichage des capteurs et le rafraichissement des valeurs affichées par les capteurs. |

## Les plans

Les plans sont des images placées dans le répertoire **plans**. Dans la maquette, les plans sont affichés dans un rectangle de 600x400 quelle que soit leur taille.

## Les capteurs

Pour chaque catégorie de capteur, il faut créer un fichier PHP dans le sous-répertoire **capteurs**. Le fonctionnement est indiqué dans les sections suivantes.

# Création d’un nouveau capteur

## Objectif

Imaginons qu’on veuille gérer un interrupteur à deux états : ouvert et fermé. Lorsqu’on appuie sur **Ouvrir**, la lumière s’allume, lorsqu’on appuie sur **Fermer**, elle s’éteint. Le capteur nous indique dans quel état il se trouve.

Le capteur **interrupteur** est différent d’un capteur **poussoir** qui ne retourne aucun état et sur lequel on appuie « à l’aveugle » pour changer l’état (allumé -> éteint -> allumé…).

## Description globale

Le fonctionnement d’un capteur est défini par sa « catégorie ». Pour gérer un nouveau capteur, il faut déclarer une nouvelle catégorie. L’application utilise le nom de la catégorie pour savoir quelle page afficher. Il y a donc deux choses à faire :

1. Dans le fichier **base.php**, déclarer le capteur dans la table des capteurs. Comme nom de catégorie on utilise le nouveau nom « *interrupteur* »
2. Créer la page **interrupteur.php** que l’application va appeler pour afficher les informations et les commandes du capteur. Cet affichage sera inséré dans l’emplacement prévu pour les capteurs dans la page principale.

Le capteur pouvant afficher des informations changeantes, il faut que son affichage soit mis régulièrement à jour. Ce n’est pas tout l’affichage qui change (ça serait très laid) mais seulement les valeurs. L’application principale rafraîchit la valeur des capteurs toutes les secondes. Il faut que la page **interrupteur.php** déclare quelles zones rafraîchir et comment. La page contient donc du code javascript qui paramètre le rafraîchissement des valeurs du capteur.

Enfin, l’affichage du capteur contient des commandes sous forme de fonctions javascript.

Le fichier **capteurs.js** contient les fonctions javascript utilisées pour cette mécanique.

## Déclaration du capteur dans une nouvelle catégorie

Pour créer le nouveau capteur, il faut lui donner un nom de catégorie qui n’existe pas déjà : **interrupteur**. On place l’interrupteur dans une pièce dans la base serveur (ici la pièce 4) :

7 => [

"piece" => 4,

"numero" => "adr\_99",

"nom" => "Lumière",

"categorie" => "interrupteur",

"icone" => "none",

"coord" => "50, 100"

]

Le nom *Lumière* est le nom que l’utilisateur décide de donner au capteur : ici c’est l’interrupteur commandant l’éclairage de la pièce.

Désormais, lorsqu’on cliquera sur la pièce N° 4, le capteur sera affiché avec le titre *Lumière*.

Pour afficher les informations et actions du capteur, il faut créer le code html dans le fichier **interrupteur.php**. Ce fichier doit être dans le dossier **capteurs/** qui regroupe tous les affichages de capteurs.

### Hello world !

On commence par vérifier que ça marche en utilisant tout simplement du code html statique. Le capteur va afficher « Je suis un ***interrupteur*** » :

Je suis un <i><b>interrupteur</b></i>

Commentaire

* Le code html de la page est partiel car il est destiné à être inclus dans la page principale. Il ne faut pas commencer par la balise **<html>** comme dans une page complète.

### Les variables

Ce qu’on veut c’est un affichage dynamique qui affiche vraiment l’état du capteur ouvert ou fermé. Cet état peut changer avec le temps parce que quelqu’un entre dans la pièce et allume la lumière. Normalement, le serveur doit lire l’état du capteur via le contrôleur. Ici on simule en PHP l’état de l’interrupteur en utilisant les fonctions PHP **litVariable** et **ecritVariable** qui permettent de lire et stocker une variable. On appelle la variable « **interrupteur »**. Tant qu’on n’a pas appelé **ecritVariable**, la variable n’existe pas encore et donc **litVariable** retourne la valeur par défaut.

*<?PHP*

include "variables.php";

$etat = litVariable("interrupteur", "Ouvert");

echo "Etat : <i><b>$etat</i></b>"

*?>*

Désormais, le capteur s’affiche :

Etat : **Ouvert**

Commentaires

* Les fonctions sont déclarées dans le fichier **variables.php**. Il faut donc inclure ce fichier dans la page pour rendre ces deux fonctions disponibles.
* La fonction PHP **echo** écrit le code html qui est retourné. Lorsqu’il y a une variable à l’intérieur du chaîne, le PHP la remplace par sa valeur. Le code ci-dessous est équivalent :

*<?PHP*

include "variables.php";

$etat = litVariable("interrupteur", "Ouvert");

*?>*

Etat : <i><b>*<?php* echo $etat; *?>* </i></b>

### Différenciation des capteurs

Mais il peut y avoir plusieurs interrupteurs. Avec ce programme, tous les interrupteurs afficheront la même chose. C’est pour ça que les capteurs ont chacun un **numéro** qui permet de les différencier entre eux. Ce numéro est passé dans l’appel de la page **interrupteur.php** avec le paramètre get : **interrupteur.php?numero=xxx**. On appelle la variable PHP **interrupteur\_xxx** ce qui permet bien de distinguer les interrupteurs entre eux.

Le code suivant est le même mais il récupère d’abord le numéro d’interrupteur passé en paramètre d’appel pour définir le nom de la variable.

*<?PHP*

include "variables.php";

$numero = $\_GET["numero"];

$nomVariable = "interrupteur\_".$numero;

$etat = litVariable($nomVariable, "Ouvert");

*?>*

Etat : <i><b>*<?*=$etat=*?>*</i></b>

Commentaires

* En PHP le point est l’opérateur de concaténation entre deux chaînes de caractères. Si la variable **$numero** vaut 12, **$nomVariable** vaudra « *interrupteur\_12* ».
* Dans cet extrait on voit comment on peut écrire dans le fichier html une variable PHP. On a vu tout à l’heure la syntaxe longue <?php echo $etat; ?>. Comme ce n’est pas très beau, on utilise plutôt la syntaxe courte <?=$etat?> qui fait exactement la même chose en plus joli.

### Affichage dynamique

L’affichage est toujours statique. Si quelqu’un allume dans la pièce, cela ne va pas se voir dynamiquement sur la page. Ce qu’il faut c’est rafraichir régulièrement l’affichage du capteur en allant récupérer son état. L’application offre un mécanisme qui permet d’aller lire une certaine valeur toutes les secondes et d’aller l’écrire à un endroit particulier.

Comme la valeur sera lue régulièrement, ce n’est plus la peine de la mettre dans la page d’affichage au départ. Il suffit de prévoir l’endroit où cette valeur devra être placée ultérieurement lors du rafraîchissement.

On identifie cet endroit avec l’id : **etat\_interrupteur**.

Etat : <i><b><span id="etat\_interrupteur">valeur</span></i></b>

Commentaires

* La zone variable est un élément **span** dans cet exemple. On peut prévoir un affichage plus sophistiqué avec une mise en page tableau et utiliser une cellule de tableau comme zone de rafraîchissement.
* On peut (on doit ☺) faire appel à du CSS. La feuille de style est celle de la page principale puise que le code sera inséré dans cette page.
* L’identifiant de la zone doit être unique dans la page, même lorsqu’il y a plusieurs interrupteurs, c’est pour ça que l’identifiant utilisé dans le code suivant n’est pas **etat\_interrupteur** mais **etat\_interrupteur\_xxx**.

*<?PHP*

$numero = $\_GET["numero"];

*?>*

Etat : <span id="etat\_interrupteur\_*<?*=$numero*?>*">valeur</span>

Il faut maintenant indiquer à l’application d’aller chercher l’information toutes les secondes. La bibliothèque **capteurs.js** inclut la fonction **capteursRaf\_register** qui a pour but de passer cette information à l’application.

function capteursRaf\_register(elementId, url, interval=1, traitement=null)

* **elementId**: l’identifiant de l’élément html qu’il faut utiliser pour afficher la valeur du capteur
* **url** : l’url à appeler pour récupérer la valeur
* Les deux autres paramètres seront décrits plus tard

Toutes les secondes l’application va appeler l’url et le résultat retourné par l’url sera placé dans l’élément **etat\_interrupteur\_xxx**.

L’appel de la fonction **capteursRaf\_register** se fait dans la page **interrupteur.php** dans une section <**script>…</script>** normale. Ce code javascript est exécuté une fois lors de l’insertion de l’affichage du capteur dans la page principale.

*<?PHP*

$numero = $\_GET["numero"];

$elementId = "etat\_interrupteur\_".$numero;

*?>*

Etat : <i><b><span id="*<?*=$elementId*?>*">valeur</span></i></b>

<script>

capteursRaf\_register("*<?*=$elementId*?>*", url);

</script>

### url de lecture

Il reste maintenant à donner l’url à utiliser pour obtenir la valeur du capteur. Cette url peut être n’importe quoi. On pourrait par exemple créer une nouvelle page **interrupteur\_lecture.php** qui serait appelée avec le paramètre get **numero** contenant le numéro de l’interrupteur dont on veut lire l’état :

*<?PHP*

include "variables.php";

$numero = $\_GET["numero"];

$nomVariable = "interrupteur\_".$numero;

$etat = litVariable($nomVariable, "Ouvert");

echo $etat;

*?>*

En tapant **http://localhost/domo/capteurs/interrupteur\_lecture.php?numero=xxx**, on obtient bien l’état de l’interrupteur xxx à afficher.

On peut aussi utiliser la même page **interrupteur.php** en gérant le paramètre get **lecture** : **interrupteur.php?lecture=xxx**

*<?PHP*

include "variables.php";

if (isset($\_GET["lecture"])) {

$nomVariable = "interrupteur\_".$\_GET["lecture"];

echo litVariable($nomVariable, "Ouvert");

exit();

}

$numero = $\_GET["numero"];

$elementId = "etat\_interrupteur\_".$numero;

$etat = litVariable($nomVariable, "Ouvert");

*?>*

Etat : <i><b><span id="*<?*=$elementId*?>*">valeur</span></i></b>

<script>

capteursRaf\_register(  
 "<?=$elementId?>",  
 "capteurs/interrupteur.php?lecture=<?=$numero?>");

</script>

Commentaires

* La fonction PHP **isset** détermine si le paramètre **lecture** a été passé en paramètre get. Si c’est le cas, le nom de la variable est construit à partir de la valeur de **lecture**. La fonction **exit** indique qu’il faut sortir de la page.

Lorsque le paramètre **lecture** est présent, la page PHP retourne simplement la valeur lue et non pas le html d’affichage. On note qu’on utilise cette url dans l’appel de la fonction **capteursRaf\_register**.

L’affichage de l’interrupteur peut maintenant être testé. Il affiche bien son état ouvert.

### Commandes de l’interrupteur

Il faut maintenant que l’utilisateur pour allumer ou éteindre. On va lui afficher deux boutons d’action **Allumer** et **Eteindre**.

Une commande est là encore une url. Contrairement à la lecture, la commande n’attend pas de retour. Le changement d’état sera obtenu lors du prochain rafraichissement.

La bibliothèque **capteurs.js** inclut la fonction **commandeCapteur** dédiée à l’envoi de commandes : elle soumet une requête et ne prévoit aucun retour.

Comme pour la lecture, la commande peut être n’importe quelle url. Nous allons encore une fois utiliser la page **interrupteur.php** pour exécuter les deux commandes. L’allumage sera fait avec **interrupteur.php?allumer=xxx** et l’extinction avec **interrupteur.php?eteindre=xxx**.

if (isset($\_GET["allumer"])) {

$nomVariable = "interrupteur\_".$\_GET["allumer"];

ecritVariable($nomVariable, "Ouvert");

exit();

}

if (isset($\_GET["eteindre"])) {

$nomVariable = "interrupteur\_".$\_GET["eteindre"];

ecritVariable($nomVariable, "Fermé");

exit();

}

On peut mettre à jour le code html avec les deux boutons utilisant ces deux url.

<a onclick=commandeCapteur(

"capteurs/interrupteur.php?allumer=*<?*=$numero*?>*")>Allumer</a>

<a onclick=commandeCapteur(

"capteurs/interrupteur.php?eteindre=*<?*=$numero*?>*")>Eteindre</a>

Commentaires

* Noter que l’url est relative au serveur principal. Il faut indiquer le répertoire **capteurs/** dans l’url.
* Il n’y a aucune sécurité : il suffit d’entre l’url **interrupteur.php?eteindre=xxx** dans un navigateur pour commander l’interrupteur de n’importe quelle maison, même une dont on n’est pas propriétaire !

## Page complète

Le code complet de l’afficheur de l’interrupteur est le suivant.

*<?PHP*

include "variables.php";

*// Lecture de l'état de l'interrupteur*

*// Retourne la valeur et termine l'exécution de la page*

if (isset($\_GET["lecture"])) {

$nomVariable = "interrupteur\_".$\_GET["lecture"];

echo litVariable($nomVariable, "Ouvert");

exit();

}

*// Commande d'allumage : change la valeur de la variable*

*// et quitte la page sans rien retourner*

if (isset($\_GET["allumer"])) {

$nomVariable = "interrupteur\_".$\_GET["allumer"];

ecritVariable($nomVariable, "Ouvert");

exit();

}

*// Idem avec l'extinction*

if (isset($\_GET["eteindre"])) {

$nomVariable = "interrupteur\_".$\_GET["eteindre"];

ecritVariable($nomVariable, "Fermé");

exit();

}

*// Si on est ici c'est que l'on veut le code HTML*

*// de l'affichage du capteur*

*// On ne fait aucun contrôle....*

*// Numéro de l'interrupteur*

$numero = $\_GET["numero"];

*// On construit l'identifiant de l'élément html*

*// qui contiendra la valeur de l'état*

$elementId = "etat\_interrupteur\_".$numero;

*// S'en est fini du PHP, c'est plus facile de continuer en HTML*

*?>*

Etat : <i><b><span id="*<?*=$elementId*?>*">valeur</span></i></b>

<br/>

<a onclick=commandeCapteur("capteurs/interrupteur.php?allumer=*<?*=$numero*?>*")>Allumer</a>

&nbsp;

<a onclick=commandeCapteur("capteurs/interrupteur.php?eteindre=*<?*=$numero*?>*")>Eteindre</a>

<script>

*// Déclare l'url qu'il faut utiliser pour mettre à jour*

*// l'état du capteur*

*// Toutes les secondes, l'url sera appelée et le résultat sera*

*// Placé au bon endroit dans l'affichage du capteur*

capteursRaf\_register("*<?*=$elementId*?>*", "capteurs/interrupteur.php?lecture=*<?*=$numero*?>*");

</script>

Et voilà : l’interrupteur est opérationnel dans la maquette !

En cliquant sur Allumer ou Eteindre, l’interrupteur change d’état au bout d’une seconde environ.

# Exemple de capteur complexe : capteur météo

## Objectif

Un capteur peut également procéder à la collecte de données externe, la météo par exemple. Le site **openweathermap.org** permet de récupérer des informations météo. Le retour n’est pas une valeur simple mais un fichier json.

Il faut s’inscrire sur le site et récupérer une clef d’API.

L’objectif du capteur météo est d’afficher la température, la pression et le temps autour de la maison.

## Particularités

Le rafraichissement toutes les secondes n’a pas de sens. Il faut un rafraîchissement moins fréquent, toutes les 30 minutes par exemple.

Par ailleurs, l’url de lecture ne récupère pas une valeur unique mais un ensemble de valeurs au format json. Il faut d’abord analyser le retour pour procéder à l’affichage des valeurs. On pourrait demander au serveur PHP de le faire avec le principe suivant :

* Consultation de la météo
* Analyse des données reçues et stockage des valeurs intéressantes
* Création d’une url de lecture pour chaque valeur

L’autre solution consiste à analyser des données du côté de l’application. La fonction **capteursRaf\_register** offre deux paramètres supplémentaires (en plus de l’identifiant de l’élément à rafraîchir et de l’url) :

function capteursRaf\_register(

elementId,

url,

interval=1,

traitement=null) {

* **interval** : intervalle en secondes entre deux rafraichissements (1 par défaut)
* **traitement** : fonction javascript à appeler lors du retour de la réponse (null par défaut)

On a vu dans l’exemple de l’interrupteur que lorsque la fonction **traitement** n’est pas précisée, l’application écrivait le retour dans l’élément cible.

Lorsque la fonction **traitement** est donnée, l’application ne traite pas le retour elle-même mais appelle la fonction avec le retour et le numéro de capteur: **traitement(numero, retour)**.

capteursRaf\_register(

"*<?*=$numero*?>*",

urlMeteoVille("Paris, fr"),

interval=1800,

traitement=traitement\_meteo);

Commentaires

* Cette fois-ci l’identifiant d’élément html n’est plus le nom d’une zone d’affichage mais le numéro du capteur. La fonction traitement sait comment la zone est structurée et comment utiliser ce numéro pour mettre à jour les valeurs. C’est donc le numéro de capteur qu’il faut utiliser dans l’appel de la fonction **capteursRaf\_register**.
* **traitement\_meteo** est le nom de la fonction javascript à appeler lorsque le résultat est reçu du site météo. Cette fonction peut être écrite dans le fichier meteo.php mais elle peut aussi être incluse dans l’application principale.

Dans l’exemple météo, le json retourné est du format :

{

"coord":{"lon":2.35,"lat":48.85},

"weather":[

{"id":803,

"main":"Clouds",

"description":"broken clouds",

"icon":"04d"}

],

"base":"stations",

"main":{

"temp":284.15,

"pressure":1023,

"humidity":76,

"temp\_min":284.15,

"temp\_max":284.15},

"visibility":10000,

"wind":{"speed":2.6,"deg":30},

"clouds":{"all":75},

"dt":1478082600,

"sys":{

"type":1,

"id":5615,

"message":0.1812,

"country":"FR",

"sunrise":1478068816,

"sunset":1478104038

},

"id":2988507,

"name":"Paris",

"cod":200

}

La fonction **traitement\_meteo** récupère les trois valeurs qui nous intéressent ici :

var meteo = JSON.parse(json);

var temp = meteo["main"]["temp"] - 273; *// kelvins*

var pressure = meteo["main"]["pressure"];

var temps = meteo["weather"][0]["description"];

## Page complète

La page complète du capteur météo est la suivante :

*<?PHP*

$numero = $\_GET["numero"];

$id\_tempe = "meteo\_temperature\_".$numero;

$id\_press = "meteo\_pression\_".$numero;

$id\_temps = "meteo\_temps\_".$numero;

*?>*

<!-- Le code principal du capteur sous forme de tableau de 3 lignes -->

<table>

<tr>

<td>Température : </td><td><b>&nbsp;<span id=*<?*=$id\_tempe*?>*>&nbsp</span>°C</td>

</tr><tr>

<td>Pression : </td><td><b>&nbsp;<span id=*<?*=$id\_press*?>*>&nbsp</span> hPa</td>

</tr><tr>

<td>Temps : </td><td><b>&nbsp;<span id=*<?*=$id\_temps*?>*>&nbsp</span></td>

</tr>

</table>

<!-- Le script -->

<script>

*// Il faut récupérer une clef gratuite sur le site* openweathermap.org

var weather\_key = 'toto';

*// Retourne l'url de consultation de la météo par nom de ville*

*// Il existe des version qui prenent le code de la ville,*

*// une autre les coordonnées lat / lon ...*

function urlMeteoVille(nom) {

return "http://api.openweathermap.org/data/2.5/weather?q=" +

nom + "&appid=" + weather\_key;

}

*// Traite le json retour de la consultation météo*

*// Le numéro du capteur météo est passé en paramètre*

function traitement\_meteo(numero, json) {

var meteo = JSON.parse(json);

var temp = meteo["main"]["temp"] - 273; *// Température en kelvins*

var pressure = meteo["main"]["pressure"];

var temps = meteo["weather"][0]["description"];

*// writeDebug("json Méteo: " + json);*

var id\_tempe = "meteo\_temperature\_" + numero;

var id\_press = "meteo\_pression\_" + numero;

var id\_temps = "meteo\_temps\_" + numero;

document.getElementById(id\_tempe).innerHTML = temp.toFixed(1);

document.getElementById(id\_press).innerHTML = pressure;

document.getElementById(id\_temps).innerHTML = temps;

}

*// Raffraichissement toutes les 30 minutes*

*// Comme on fait appel à la fonction traitement\_meteo,*

*// le premier paramètre n'est pas un élément id*

*// mais le numéro du capteur*

*// L'url de rafraichissement est un lien extérieur*

capteursRaf\_register("*<?*=$numero*?>*", urlMeteoVille("Paris, fr"), interval=1800, traitement=traitement\_meteo);

</script>