

Travaux pratiques Domotique

-

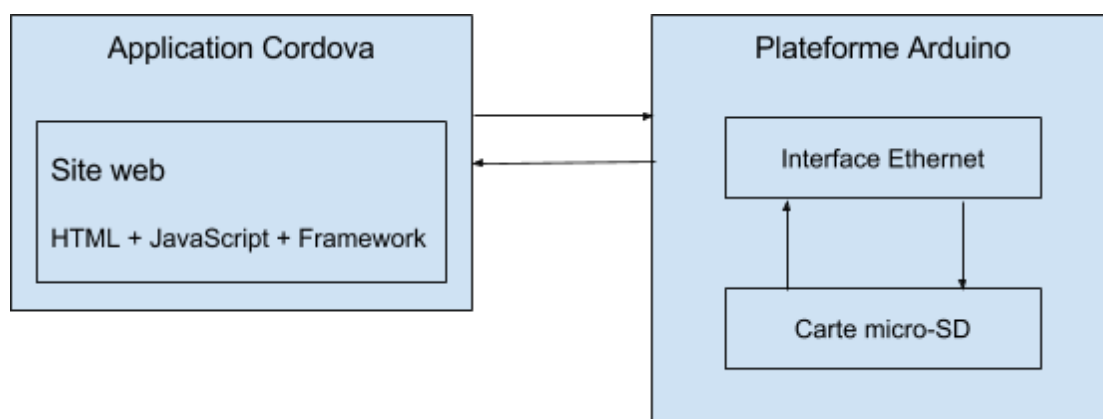
*Sébastien LAPORTE
Guillaume MORAND*

- [1. Architecture de l'application](#)
- [2. Matériel utilisé](#)
- [3. API du serveur](#)
 - [3.1. Demandes d'état et informations : Requêtes GET](#)
 - [3.1.1. État des entrées analogiques](#)
 - [3.1.1.1. URL d'accès](#)
 - [3.1.1.2. Données renvoyées](#)
 - [3.1.2. Liste des périphériques i2c](#)
 - [3.1.2.1. URL d'accès](#)
 - [3.1.2.2. Données renvoyées](#)
 - [3.1.3. État d'un périphérique i2c](#)
 - [3.1.3.1. URL d'accès](#)
 - [3.1.3.2. Données renvoyées](#)
 - [3.1.4. Liste des sorties digitales](#)
 - [3.1.4.1. URL d'accès](#)
 - [3.1.4.2. Données renvoyées](#)
 - [3.2. Demande de changement d'état : Requête PUT](#)
 - [3.2.1. Modification de l'état d'une sortie digitale](#)
 - [3.2.1.1. Format de l'URL](#)
 - [3.2.1.2. Données renvoyées](#)
 - [3.2.2. Modification de l'état d'un périphérique i2c](#)
 - [3.2.2.1. Format de l'URL](#)
 - [3.2.2.2. Données renvoyées](#)
- [4. Programme serveur du microcontrôleur](#)
 - [4.1. Fonctionnement du programme](#)
 - [4.1.1. Initialisation](#)
 - [4.1.2. Gestion d'une requête](#)
 - [4.1.2.1. Requêtes de type GET](#)
 - [4.1.2.2. Requêtes de type PUT](#)
 - [4.2. Optimisation de la mémoire](#)
 - [4.3. Gestion des erreurs](#)
- [5. Partie cliente](#)
 - [5.1. Cordova](#)
 - [5.2. Framework CSS](#)
 - [5.3. jQuery](#)
 - [5.4. Architecture du projet](#)
 - [5.5. Interface - Entrées analogiques](#)
 - [5.6. Interface - Sorties digitales](#)
 - [5.7. Interface - Périphériques I2C](#)
- [6. Reprise de projet](#)

1. Architecture de l'application

Pour concevoir l'architecture de notre application, nous avons dû prendre en compte les contraintes associées à l'utilisation d'un microcontrôleur. En effet, la carte Arduino Uno que nous utilisons possède des ressources mémoires très limitées par rapport à celle d'un ordinateur personnel. De plus, un tel système ne possède pas d'un véritable serveur web comme Apache2 que l'on pourrait utiliser sur une carte tel que le Raspberry Pi.

Nous avons donc fait le choix de déporter au maximum les traitements à effectuer côté application client. En effet, aujourd'hui les smartphones, tablettes et ordinateur possèdent suffisamment de ressource et des navigateurs web très avancés pour exploiter cette ressource. Nous nous sommes orienté sur une application web réalisée en HTML et JavaScript et JQuery. Pour avoir une interface intuitive et ergonomique, nous avons également utilisé un framework CSS nommé Framework7.



Nous sommes donc sur une architecture de type client - serveur. La plateforme Arduino héberge une API qui permet d'exposer des ressources à un client web. Aujourd'hui la majorité des API fonctionnent avec le format de donnée JSON. L'utilisation de ce format de fichier permet de se baser sur un standard couramment utilisé dans le monde du web, ce qui permet de sérialiser / dé-sérialiser facilement.

2. Matériel utilisé

Pour la réalisation de ce TP nous avons utilisé une carte Arduino Uno basée sur microcontrôleur Atmega328 ainsi qu'un Shield Ethernet Arduino V1 basé sur une puce W5100. Nous avons connecté un ensemble de périphérique :

Type de matériel	Pin Arduino	Type de matériel	Pin Arduino
LED	D2	LED	D8
LED	D3	Écran LCD	A4 - A5 (I2C)
LED	D5	Potentiomètre	A0
LED	D6	Potentiomètre	A1
LED	D7	Bouton poussoir	A2

3. API du serveur

Le serveur est représenté par le microcontrôleur Arduino Uno + Shield Ethernet V1.

3.1. Demandes d'état et informations : Requêtes GET

Pour demander des ressources au serveur, nous utilisons les requêtes HTTP GET.

3.1.1. État des entrées analogiques

Cette ressource permet d'envoyer au client l'état des entrées analogiques de la carte Arduino. Les données sont sérialisées au format JSON par couple : identifiant d'entrée analogique – valeur. Les valeurs varient entre 0 et 1023 qui est la résolution des entrées analogique de l'Arduino Uno.

3.1.1.1. URL d'accès

<http://<adresse-ip>/analog>

3.1.1.2. Données renvoyées

```
[
{
  "id": 0,
  "value": 756
},
{
  "id": 1,
  "value": 1022
},
{
  "id": 2,
  "value": 0
},
{
  "id": 3,
  "value": 555
}
]
```

3.1.2. Liste des périphériques i2c

Cette ressource permet au client d'obtenir la liste des périphériques i2c connectés à la carte. Ce fichier au format JSON est stocké sur la carte micro-SD du Shield Ethernet. Grâce à cette ressource, le client peut construire l'interface cliente et par la suite effectuer d'autres requêtes pour connaître l'état des périphériques et également modifier cet état.

3.1.2.1. URL d'accès

<http://<adresse-ip>/i2c>

3.1.2.2. Données renvoyées

Chaque élément du tableau JSON constitue 1 périphérique i2c. Chaque périphérique possède un « type » (écran, capteur, ...), une adresse utilisée comme identifiant et des options. Chaque type de périphérique i2c possède ses propres options. Dans l'exemple ci-dessous, nous avons mis en œuvre un écran LCD qui possède deux lignes.

```
[
{
  "type": "screen",
  "address": "69",
  "option_1": {
    "human": "Ligne 1",
    "short": "line1"
  },
}
```

```

    "option_2": {
      "human": "Ligne 2",
      "short": "line2"
    }
  }
]

```

3.1.3. État d'un périphérique i2c

Cette ressource permet de récupérer l'état d'un périphérique i2c. Pour accéder à cette ressource, le client utilise les informations récupérées grâce à la liste des périphériques i2c.

3.1.3.1. URL d'accès

<http://<adresse-ip>/i2c/<adresse-du-périphérique-i2c>/<ressource>>

3.1.3.2. Données renvoyées

Exemple : <http://192.168.0.100/i2c/69/line1>

Dans cet exemple, la requête permet de récupérer l'option 1 du périphérique i2c avec l'identifiant 69.

```

{
  "name": "line1",
  "value": "Hello"
}

```

3.1.4. Liste des sorties digitales

Cette ressource renvoie la liste des sorties digitales avec leur état. On distingue deux type de sortie digitale : Les sortie tout ou rien (TOR) ou les sorties de type PWM.

Remarque : Le Shield Ethernet utilise les entrées / sorties 10, 11, 12 et 13 pour la communication SPI avec la ship W5100. Le lecteur de carte micro-SD utilise également l'entrée / sortie 4. Ces sorties ne doivent donc pas être utilisées.

3.1.4.1. URL d'accès

<http://<adresse-ip>/digital>

3.1.4.2. Données renvoyées

```

[
  {
    "id": 2,
    "type": "TOR",
    "value": 0
  },
  {
    "id": 3,
    "type": "PWM",
    "value": 109
  },
  {
    "id": 5,
    "type": "PWM",
    "value": 96
  },
  {
    "id": 6,
    "type": "PWM",
    "value": 0
  },
  {
    "id": 7,
    "type": "TOR",

```

```
    "value": 0
  },
  {
    "id": 8,
    "type": "TOR",
    "value": 0
  },
  {
    "id": 9,
    "type": "PWM",
    "value": 0
  }
]
```

3.2. Demande de changement d'état : Requête PUT

Pour modifier les sorties digitales ou modifier les paramètres des périphériques i2c, nous utilisons les requêtes HTTP de type PUT.

3.2.1. Modification de l'état d'une sortie digitale

3.2.1.1. Format de l'URL

<http://<adresse-ip>/pwm/<numéro-de-sortie>/<valeur>>

Exemple : <http://192.168.0.100/digital/2/255>

Cet exemple demande la modification de la sortie digitale numéro 2 à la valeur 255.

3.2.1.2. Données renvoyées

Pour informer le client de la prise en compte de sa requête, nous renvoyons le code HTTP 200 OK.

3.2.2. Modification de l'état d'un périphérique i2c

3.2.2.1. Format de l'URL

<http://<adresse-ip>/<type-de-périphérique>/<numéro-d'option>/<Valeur>>

Exemple : <http://192.168.0.100/i2c/69/line1/Hello>

Cet exemple demande la modification de la ressource « line1 » du périphérique i2c avec l'adresse 0x69 à la valeur « Hello ».

3.2.2.2. Données renvoyées

Pour informer le client de la prise en compte de sa requête, nous renvoyons le code HTTP 200 OK.

4. Programme serveur du microcontrôleur

4.1. Fonctionnement du programme

4.1.1. Initialisation

La phase d'initialisation consiste à configurer les ressources du microcontrôleur :

- Configuration de l'écran LCD
- Configuration du port série (pour le debug)
- Initialisation du lecteur de carte micro-SD
- Configuration des pins analogiques en sorties
- Configuration et initialisation du serveur Ethernet (Adresse IP, adresse MAC)
- Restauration de l'état des sorties digitales (sauvegardées sur carte micro-SD)

À l'issue de cette phase de configuration, on passe en fonctionnement normal.

4.1.2. Gestion d'une requête

La réception d'une requête du client est gérée par la fonction **listen_web_client**. Pour extraire les données utiles de la requête, on utilise la fonction **search_request** qui permet de stocker dans un buffer les données à analyser. Ensuite, la fonction **extract_request** permet d'extraire les données en récupérant l'ensemble des paramètres de la requête. Dans notre API, nous utilisons au maximum 4 paramètres.

4.1.2.1. Requêtes de type GET

Pour rappel, les requêtes de type GET permettent au client de récupérer les informations sur les ressources. Nous effectuons le traitement des requêtes dans le programme principal. En fonction de la requête transmise par le client, nous effectuons un traitement pour envoyer la ressource demandée (Voir partir API du rapport).

- **Demande de l'état des entrées analogiques** : La fonction **readAndSerializeAnalog** permet de lire l'état des entrées analogiques, de sérialiser les données au format JSON et de les envoyer au client.
- **Demande de l'état des sorties digitales** : Lecture de l'état des sorties digitales stockées en carte SD avec la fonction **readPwm** puis sérialisation au format JSON et envoi au client avec la fonction **serializePwm**.
- **Demande la liste des périphériques i2c** : Lecture du fichier stocké en carte SD et envoi des données au client web avec la fonction **send_data**.
- **Demande de l'état d'un périphérique i2c** : Lecture de l'état de la ressource demandée en carte SD avec la fonction **readStrFromSdCard** puis sérialisation et envoi des données au client avec la fonction **serializeDeviceInfo**.

4.1.2.2. Requêtes de type PUT

Les requêtes PUT permettent de modifier l'état d'une ressource. Le traitement est effectué dans le programme principal. En fonction de la requête transmise par le client, nous effectuons un traitement pour envoyer la ressource demandée (Voir partir API du rapport).

- **Modification de l'état d'une sortie digitale** : Mise à jour de l'état demandé par le client sur la carte SD avec la fonction **updateFile** puis mise à jour de l'état de la sortie en fonction du numéro de broche avec les fonctions **analogWrite** pour les sorties type PWM et **digitalWrite** pour les sorties tout ou rien.
- **Modification de l'état d'un périphérique i2c** : Mise à jour de l'état du périphérique en carte SD avec la fonction **updateFile** puis mise à jour de l'afficheur avec les fonctions **setCursor** pour sélectionner la ligne et **print** pour écrire sur l'afficheur.

4.2. Optimisation de la mémoire

Étant sur un microcontrôleur avec peu de mémoire, nous avons dû mettre en oeuvre des mécanismes permettant d'économiser l'emprunte mémoire de notre programme. Pour cela, nous avons limité au maximum la taille de nos buffers en les dimensionnant au plus juste. De plus, nous avons stocké les chaînes de caractères en mémoire flash pour économiser la mémoire RAM du microcontrôleur. Pour effectuer cela, nous avons utilisé les directives préprocesseur **PROGMEM**. Sur l'environnement Arduino, il y a deux manières d'utiliser cette directive :

- **Pour les méthodes des bibliothèques Arduino**

Pour stocker en flash les chaînes de caractères utilisées avec les méthodes suivantes, il est possible de précéder les chaînes de caractère de la façon suivante avec **F("String")** :

```
Serial.println(F("Carte SD non presente"));          /* Pour le terminal série */
```

```
client.println(F("Access-Control-Allow-Origin: *")); /* Pour le client web */
```

- Pour les autres fonctions

Les fonctions natives du C ne permettent pas l'utilisation précédente. Pour cela il est nécessaire de déclarer les chaînes de caractères de la façon suivante et d'utiliser un pointeur pour y accéder. Nous utilisons ensuite un buffer au moment d'utiliser cette variable stockée en mémoire flash.

```
char buffer_progm[64]; /* Buffer utilisé pour charger la variable en RAM */

const char APP_JSON[] PROGMEM = "application/json"; /* Déclaration de la variable */
const char * const _APP_JSON PROGMEM = APP_JSON; /* Pointeur sur la variable en flash */

strcpy_P(buffer_progm, (char*)pgm_read_word(&(_APP_JSON))); /* Chargement */
send_special_header(client, buffer_progm); /* Utilisation de la variable */
```

4.3. Gestion des erreurs

Afin de gérer les cas où le client nous envoie des informations erronées, nous avons protégé le programme serveur pour éviter les erreurs et éventuellement un overflow.

- Dans le cas d'une requête erronée transmise par le client, on renvoie un erreur HTTP 404 pour informer le client que la ressource est introuvable.
- Dans le cas où le client transmet une requête trop volumineuse pour les buffers, celle-ci est ignorée pour éviter l'overflow.

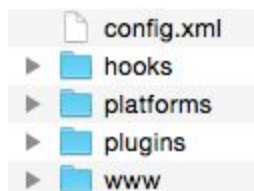
5. Partie cliente

5.1. Cordova

Lors de ce TP la partie cliente développée a été intégrée dans un dossier d'application Cordova.

Cordova permet de créer des applications multi-plateforme en utilisant les langages web tel que HTML, CSS et JavaScript.

Le développement de la partie cliente est donc entièrement réalisé grâce au langage web et est donc utilisable à la fois sur un périphérique mobile iOS ou Android une fois l'application compilée grâce à Cordova mais aussi à travers un navigateur web en exécutant le fichier "index.html".



L'architecture du dossier ci-dessus correspond à l'architecture d'un dossier Cordova.

Le dossier **/platforms** contient l'ensemble des éléments de configuration, de fonctionnement et produits finaux pour une plateforme, nous avons créé les plateformes iOS et Android.

Le dossier **/plugins** regroupe l'ensemble des plug-ins utilisés et les fichiers nécessaires à leur fonctionnement. En effet, Cordova permet d'intégrer des plugins afin d'accéder aux fonctionnalités natives du périphérique cible. Cette partie de nous ne concerne pas lors de ce TP.

Le dossier **/www** est le dossier contenant l'ensemble des éléments à modifier afin d'apporter des modifications à l'application. C'est dans ce dossier que l'on trouve l'ensemble des fichiers HTML, CSS et JavaScript que nous avons développés.

5.2. Framework CSS

Afin de réaliser un gain de temps non négligeable lors du développement de l'application, nous avons utilisé un framework CSS/JS nous mettant à disposition de nombre de classes afin de mettre en forme visuellement notre application. Le framework que nous avons utilisé est Framework7, l'ensemble de la documentation sur ce Framework est disponible via le lien suivant :

<http://www.idangero.us/framework7/docs/#.VrDZ9nqhksl>

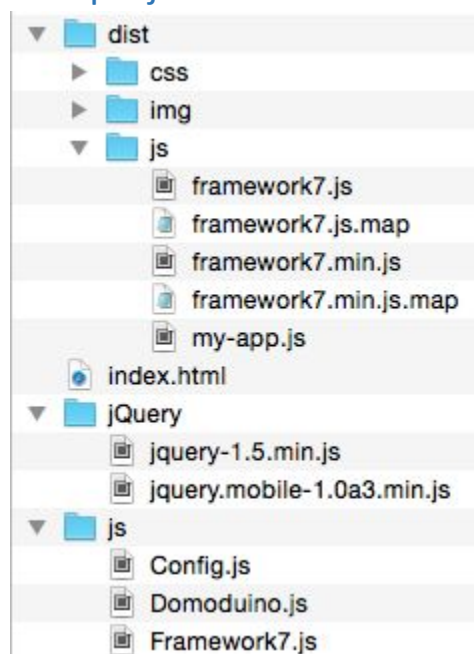
5.3. jQuery

Toujours afin de procéder à un gain de temps lors du développement de notre partie client afin de nous concentrer sur les fonctionnalités permettant le pilotage de périphériques et la visualisation d'informations remontées par les différents périphériques nous avons utilisé le framework JavaScript jQuery.

Ce framework nous permet d'utiliser de façon simplifiée et bien plus rapide l'ensemble des sélecteurs et événements JavaScript. La documentation est disponible via le lien suivant :

<http://api.jquery.com>

5.4. Architecture du projet



Ci-dessus l'architecture du dossier **/www** décrit dans la partie 4.1 du présent document.

Dans le dossier **/dist** nous retrouvons l'ensemble du framework "Framework7", il est composé de fichiers CSS pour la mise en page et JavaScript pour le dynamisme des animations.

Le fichier **index.html** contient l'ensemble du code statique de notre page web.





Dans le dossier **/jQuery** l'on retrouve les sources du framework "jQuery".

Dans le dossier **/js** on trouve 3 fichiers JavaScript :

- Le fichier **Config.js** contient la configuration de l'adresse IP où se trouve notre API.
- Le fichier **Domoduino.js** contient l'ensemble des fonctions nous permettant de récupérer via l'API, mettre en forme les données dans du code HTML, afin de les visualiser dans l'interface du client web. Les fonctions permettant de mettre à jour des données se trouvent aussi dans ce fichier. Elles récupèrent les informations issues de l'interface web afin de les soumettre à l'API.
- Le fichier **Framework7.js** contient l'ensemble des initialisations et gestions d'éléments du framework.





5.5. Interface - Entrées analogiques

Rendu sur navigateur web d'ordinateur :

Domoduino		
Analog		
Digital		
I2C		
	AN0	1023
	AN1	0
	AN2	0
	AN3	516

Remarque : La navigation dans les différentes interface de l'application s'effectue par le menu en haut de la page.

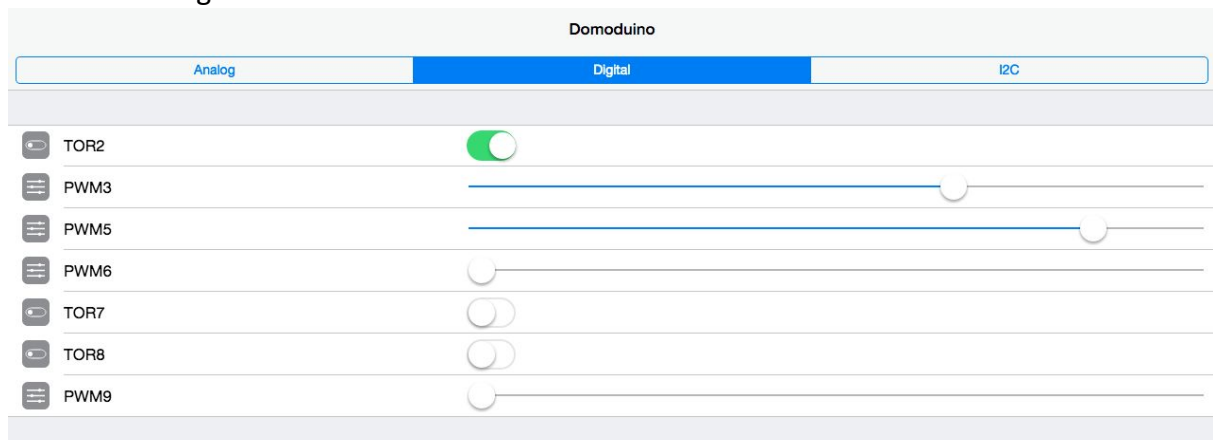
Rendu sur un smartphone avec l'application mobile :

Orange F 17:27 100 %		
Domoduino		
Analog		
Digital		
I2C		
	AN0	1023
	AN1	0
	AN2	0
	AN3	568

- **Fonction de la page** : Permet d'afficher les valeurs issues des potentiomètres et du bouton poussoir raccordé sur nos entrées analogiques. Aucune modification de valeur n'est possible à travers cette interface. C'est cette page qui est affichée lors de l'ouverture du fichier "index.html" dans un navigateur web.
- **Initialisation de la page** : Lorsque l'ensemble du contenu du DOM est chargé, les éléments suivants sont effectués :
 - Fonction **get_analog()** :
 - Requête GET sur l'API pour récupérer la ressource "/analog"
 - Ajout des différentes entrées analogiques avec leur identifiant et leur valeur dans la balise HTML identifié : ***list-analog***
 - En cas d'erreur sur la récupération de la ressource, l'utilisateur est informé d'une erreur par une popup
- **Event** : Lorsque l'utilisateur clique sur le bouton "Analog" présent dans le bandeau de navigation situé en haut de l'interface, la fonction **get_analog()** est appelée de nouveau.

5.6. Interface - Sorties digitales

Rendu sur navigateur web d'ordinateur :



Rendu sur un smartphone avec l'application mobile :



- **Fonction de la page** : Permet d'afficher et modifier les valeurs appliquées sur les sorties de notre Arduino. On trouve de type de sortie, les sorties TOR (Tout Ou Rien) et les sorties PWM.
- **Initialisation de la page** : Lors de l'ouverture du fichier "index.html" aucune fonction concernant cette interface n'est appelée, en effet, l'interface d'ouverture est l'interface "Analog"
- **Event** :
 - Lorsque l'utilisateur clique sur le bouton "Digital" présent dans le bandeau de navigation situé en haut de l'interface, la fonction **get_digital()** est appelé.
Fonction **get_digital()** :
 - Requête GET sur l'API pour récupérer la ressource "/digital"
 - Ajout des différentes sorties TOR et PWM avec leur identifiant et leur valeur dans la balise HTML identifiée : **list-pwm**. En fonction, de leur type TOR ou PWM la représentation graphique est différente, un simple SWITCH pour les sorties TOR et un SLIDER pour les sorties PWM
 - En cas d'erreur sur la récupération de la ressource, l'utilisateur est informé d'une erreur par une popup
 - Lorsque l'utilisateur modifie une valeur à l'aide d'un SWITCH ou d'un SLIDER la fonction **update_digital()** est appelée.
Fonction **update_digital()** :
 - Récupération de l'identifiant et de la valeur modifiée
 - Requête PUT sur l'API pour modifier la ressource "/digital/<id>/<value>"

- En cas d'erreur lors de la modification de la ressource, l'utilisateur est informé d'une erreur par une popup

5.7. Interface - Périphériques I2C

Rendu sur navigateur web d'ordinateur :



Rendu sur un smartphone avec l'application mobile :



- **Fonction de la page** : Permet d'afficher et modifier les valeurs issues de périphériques I2C. Dans le cadre de ce TP nous avons mis en oeuvre un écran LCD.
- **Initialisation de la page** : Lors de l'ouverture du fichier "index.html" aucune fonction concernant cette interface n'est appelée, en effet, l'interface d'ouverture est l'interface "Analog"
- **Event** :
 - Lorsque l'utilisateur clique sur le bouton "I2C" présent dans le bandeau de navigation situé en haut de l'interface, la fonction **get_i2c()** est appelée.

Fonction **get_i2c()** :

- Requête GET sur l'API pour récupérer la ressource "/i2c"
- Ajout des différents périphériques I2C avec leur identifiant et leur valeur dans la balise HTML identifiée : ***list-i2c***. Dans le cas de l'écran LCD, nous ajoutons 2 champs à l'interface utilisateur, afin que celui-ci puisse visualiser et modifier chacune des deux lignes de l'écran indépendamment
- En cas d'erreur sur la récupération de la ressource, l'utilisateur est informé d'une erreur par une popup

Pour chaque périphérique I2C obtenu la fonction **I2C_state()** appelée :

Requête GET sur l'API pour récupérer la ressource
"/i2c/<adresse>/<ressource>"

- Ajout de la valeur obtenue via l'API dans la propriété "value" de l'élément HTML auquel correspond la valeur
- En cas d'erreur sur la récupération de la ressource, l'utilisateur est informé d'une erreur par une popup
 - Lorsque l'utilisateur modifie la valeur d'un champ de texte, la fonction **update_I2C()** est appelée.

Fonction **update_I2C()** :

- Récupération de l'identifiant, de l'option et de la valeur modifiée
- Requête PUT sur l'API pour modifier la ressource
"/i2c/<address>/<option>/<value>"
- En cas d'erreur lors de la modification de la ressource, l'utilisateur est informé d'une erreur par une popup

NB : Sur chacune des interfaces présentées, il est possible d'effectuer un rafraîchissement des valeurs en plaçant sa souris sous le bandeau haut et en effectuant un clic droit maintenu et glissé vers le bas.

6. Reprise de projet

Pour mettre en oeuvre ce projet il faut disposer du matériel présenté en première partie de ce rapport. Le programme Arduino est en annexe de ce rapport, disponible via un lien Github. Dans le dossier du projet se trouve un répertoire "Carte SD". L'ensemble de ce répertoire doit être copié sur une carte micro-SD qui doit être insérée sur le Shield Ethernet.

Il peut être éventuellement nécessaire de modifier l'adresse IP du serveur pour s'adapter aux contraintes du réseau local. Dans le cadre de ce projet, nous avons mis en place notre propre réseau avec un adressage : 192.168.0.255/24. L'adresse IP du Shield Ethernet est fixée à 192.168.0.100.

Lien Github vers le projet : <https://github.com/bibi03331/TP-domotique-5A>