O FIL DE L'Ô

Yahia LAMRI

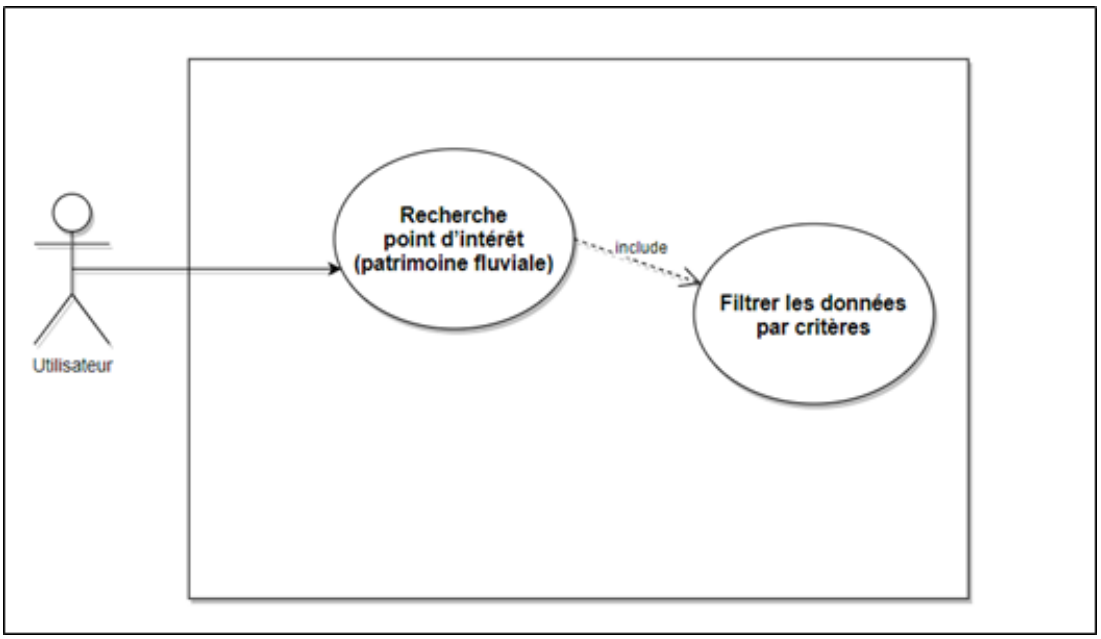
Stanislas RACZYNSKI

SIO22

PARTIE I : ETUDE

O fil de l'ô est une application exploitant une api concernant les patrimoines fluviale longeant la Seine et la Marne

Les données ouvertes, ou open datas sont des données numériques dont l'accès et l'usage sont laissés libres aux usagers. Une donnée est ouverte si chacun peut librement y accéder, l'utiliser, la modifier et la diffuser, quel que soit son but. Le format JSON est une notation objet issue de Javascript, facile à lire et à analyser il utilise des conventions compréhensibles par tous développeurs. Il est aussi très scalable et moins verbeux que le XML qui lui est un format de fichier généralement lourd, il sert à conserver les données. Et dans la majorité des cas il est un la notation qu'utilise les APIs pour communiquer avec le serveur.



PARTIE II : API REST

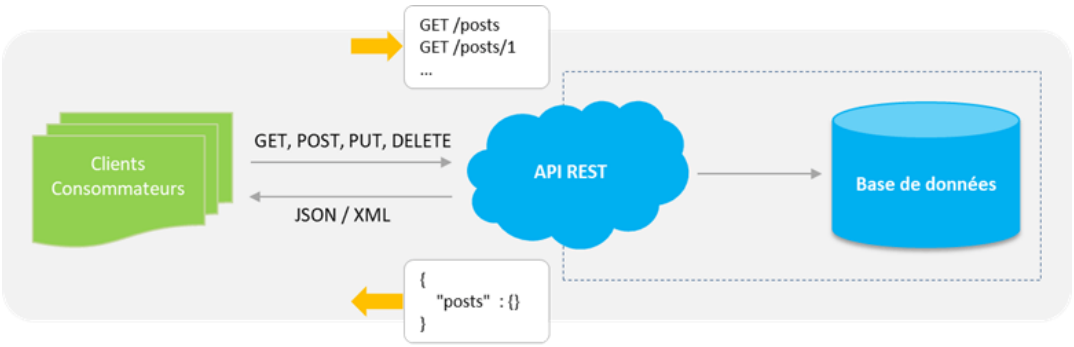
Définition

Une API (Application Programming Interface) ou interface de programmation applicative est une solution informatique qui permet à des applications de communiquer entre elles et de s'échanger des services (classes, méthodes, fonctions) ou des données.

Il existe différent type d'APIs (REST, SOAP) mais pour ce projet nous utiliserons une architecture de type REST. Une API REST ou RESTful n'est pas une technologie mais plutôt un ensemble de règles et de conventions concernant la structure et l'utilisation des données.

Représentées en JSON ou XML, les données qui transitent entre le serveurs et l'API proviennent souvent d'un système de persistance ou base de données. L'API utilise un ensemble de méthodes de requête HTTP (GET, POST, PUT, DELETE) qui indiquent l'action que l'on souhaite réaliser sur la ressource indiquée.

Pour implémenter un API Rest il est nécessaire de respecter certaines règles, notamment utiliser les verbes HTTP (GET, POST, PUT, …) comme identifiant des opérations., également l'URI comme identifiant des ressources utilisées.



PRINCIPE D'ETL :

L'API, s'appuie sur un fichier d'opendatas. Nous utilisons le principe ETL (Extract Transform & Load) pour extraire les données afin de les utiliser. Pour coder l'API il nous faut d'abord extraire les données du fichier opendata. L'API est codée sous Symfony 4, l'extraction et l'insertion des données se fait grâce à un contrôleur, les données extraites grâce à la fonction `$ponts = json_decode("../public/assets/opendata.json", true);`. Ces données sont ensuite envoyées dans un tableau intermédiaire pour valider l'existence de celles-ci. Ayant remarqué que certaine description n'était pas définie dans le fichier json, l'expression ternaire suivante permet de remplacer tout champ vide par "Aucune description" : `(isset($item['elem_princ'])) ? $item['elem_princ'] : "Aucune description.";`

```php
foreach ($ponts as $pont) {
    $tabPonts[] = [
        "identifiant" => $pont['fields']['identifian'],
        "commune" => $pont['fields']['commune'],
        "elem_princ" => $elem_princ = (isset($pont['fields']['elem_princ']))
                        ? $pont['fields']['elem_princ'] : "Aucune description.",
        "elem_patri" => $pont['fields']['elem_patri'],
    ];
}
```

Ces données sont ensuite envoyées grâce à Doctrine à la base de données.

```php
foreach ($tabPonts as $pont) {
    $patrimoine = new Patrimoine();
    if (empty($r)) {
        $patrimoine->setIdentifiant($pont['identifiant']);
        $patrimoine->setCommune($pont['commune']);
        $patrimoine->setElemPatri($pont['elem_patri']);
        $elem_princ->setElemPrinc($pont['elem_princ']);
        $em->persist($patrimoine);
        $em->flush();
    }
}
```

Maintenant que les données sont accessibles il suffit de les faire interagir avec l'API. Pour ce faire, le framework Symfony dispose d'une classe *JsonResponse* qui permet d'envoyer des données en un contenu au format json avec les bonnes en-têtes HTTP.

```php
$response = new JsonResponse();
$em = $this->getDoctrine()->getManager();
$ponts = $em->getRepository( className: Patrimoine::class)->findAll();
```

Nous commençons par récupérer toutes les données depuis la base de données. L'initialisation avec les bonnes en-tête va permettre au serveur de comprendre les données qu'il reçoit. La création d'un tableau qui contiendra les données qui seront envoyées permet de définir la structure de ce dernier. Les données envoyé grâce à la ligne suivante : `$response->setData($tabPonts);`

```php
$tabPonts = [];
foreach ($ponts as $pont) {
    $tabPonts[] = [
        "id" => $pont->getId(),
        "identifiant" => $pont->getIdentifiant(),
        "commune" => $pont->getCommune(),
        "elem_patri" => $pont->getElemPatri(),
        "elem_princ" => $pont->getElemPrinc(),
    ];
};
```


```json
{
    "id": 12,
    "identifiant": "91174150",
    "commune": "Corbeil-Essonnes",
    "elem_patri": "Viaduc de la N104",
    "elem_princ": "Pont-poutres double en béton."
},
{
    "id": 13,
    "identifiant": "77305151",
    "commune": "Montereau-Fault-Yonne",
    "elem_patri": "Pont Saint-Martin",
    "elem_princ": "Pont-poutres en béton"
},
```


De cette manière les données reçues au format json sont prêtes à être exploitées côté client.

PARTIE III : EXPLOITATION DE L'API

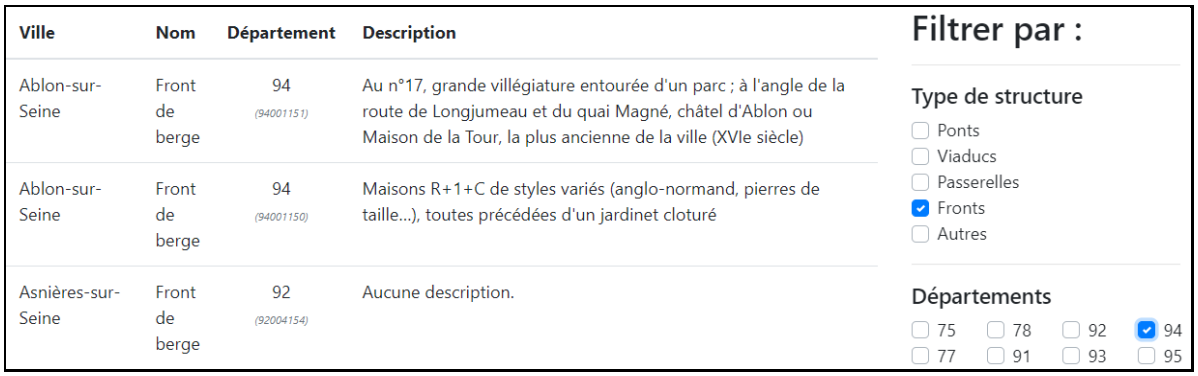
Pour récupérer les informations que nous renvoie l'API, il faut au préalable avoir lancé un serveur web, de cette manière nous pouvons récupérer les informations grâce à l'url `localhost:8000/ofildelo/api/ponts` de cette manière et grâce à la fonction suivante :

```
$.getJSON("http://127.0.0.1:8000/api/ponts", function (datas) {
    let lesPonts = JSON.parse(JSON.stringify(datas)); //add to array
    //ranger les ponts par ordre alphabetique de commune
    lesPonts.sort((a, b) => (a.commune > b.commune) ? 1 : ((b.commune > a.commune)
? -1 : 0));
    lesPonts.forEach(unPont => {
        let pontViaducOuPasserelle = (typePont) => {
            return typePont === 'pont' ? 'pont' : typePont === 'passerelle' ?
'passerelle' : typePont === 'viaduc' ? 'viaduc' : typePont === 'front' ? 'front' :
'autre';
 })
```

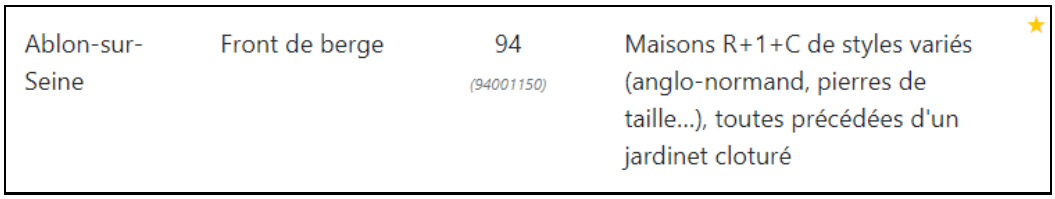
Nous insérons les informations récupérées dans un tableau que nous trions puis grâce à la fonction `$('.listePonts').append(` [contenu html] `);` nous ajoutons le contenu suivant :

```
<tr>
    <td class="commune" id="${unPont.identifiant}">
        ${unPont.commune}
    </td>
    <td>
        ${unPont.elem_patri}
    </td>
    <td class="departement" style=" ... ">
        ${getDepartement(unPont.identifiant.substring(0, 2))}
    </td>
    <td>
        ${unPont.elem_princ}
    </td>
    <td class="typePont" hidden>
        ${pontViaducOuPasserelle(unPont.elem_patri.split(' ')[0].toLowerCase())}
    </td>
</tr>
```

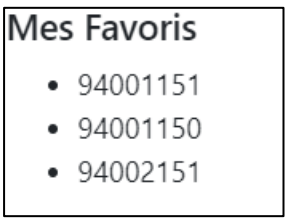
Pour ce projet, nous avons choisis de faire appel à notre API via une interface client écrite en Javascript et JQuery. Une fois l'API démarrée, l'interface client peut alors afficher tous les ponts contenus dans la base de données. L'interface propose alors au client plusieurs critères pour affiner ses recherches qui sont : les types de structures que l'on peut trouver dans les points d'intérêts, et le département où ils se trouvent.



L'interface renvoie alors un résultat contenu dans un tableau lorsque le critère Département 94 est sélectionné. Un système de favoris est aussi implanté pour que l'utilisateur puisse sauvegarder les monuments par lesquels il est intéressé :



Lorsque l'utilisateur passe sa souris au-dessus d'une case du tableau (donc d'un point d'intérêt), une étoile jaune apparaît sur la droite. Lorsque l'utilisateur clique sur le nom de la ville (dans notre exemple Albon sur Seine), l'ID de ce monument est sauvegardé dans la catégorie sur le côté nommée Mes Favoris. Les favoris ont stocké localement grâce à la librairie *localStorage*, elles sont aussi récupérées de la même manière.



La fonction de tri des données en fonction des checkbox se base sur les éléments du DOM et fait relier la valeur de la checkbox aux différents éléments (lignes) du tableau. Nous cachons les éléments qui ne contiennent pas la valeurs (attribut value) de la checkbox. Pour pouvoir identifier les éléments nous avons rajouté une colonne cachée (hidden) qui a pour valeurs le types de ponts ou l'identifiant en fonction du type de tri.

```
$(document).ready(function () {
    $("input:checkbox").on("change", function () {
        const a = $("input:checkbox:checked").val();
        console.log(a);
        if ($(this).prop("checked")) {
            $(".typePont:not(:contains(" + a + "))").parent().hide();
        } else {
            $(".typePont").parent().show();
        }
    });
});
```

```
<td class="typePont" hidden> ${pontViaducOuPasserelle(unPont.elem_patri.split('
')[0].toLowerCase())}</td>
```

La fonction `pontViaducOuPasserelle()` retourne le type de ponts de chaque élément et l'affiche dans la colonne cachée.

```
let pontViaducOuPasserelle = (typePont) => {
      return typePont === 'pont' ? 'pont' : typePont === 'passerelle' ?
'passerelle' : typePont === 'viaduc' ? 'viaduc' : typePont === 'front' ? 'front' :
'autre';
}
```

```
if (typePont === 'pont') {
    typePont = 'pont';
} else if (typePont === 'passerelle') {
    typePont = 'passerelle';
} else if (typePont === 'viaduc') {
    typePont = 'viaduc';
} else {
    typePont = 'autres';
}
```

L'expression du return remplace le code suivant pour faciliter l'écriture. Une autre fonction ayant la même logique est utilisée pour filtrer les données par départements.