307- Réaliser des pages Web interactives

Rapport personnel

Date de création : 15.05.2023
Version 1 du 15.06.2023

Simon Gendre

Module du 15.05.2023 au 13.06.2023

EMF – Fribourg / Freiburg
Ecole des Métiers / Berufsfachschule
Technique / Technik

Table des matières

1. Introduction

Le module 307 est la suite du module 101. Nous y verrons comment créer des pages interactives à l'aide de JavaScript et d'API.

1.1. Objectifs du module

Développer le caractère fonctionnel des pages Web interactives conformément aux données du problème.

Développer une maquette pour la saisie et la présentation des données compte tenu des aspects ergonomiques.

Choisir les éléments de formulaire appropriés pour la réalisation des données du problème, et garantir la validation des données entrées.

Programmer l'application de manière modulaire et conformément aux directives de codification.

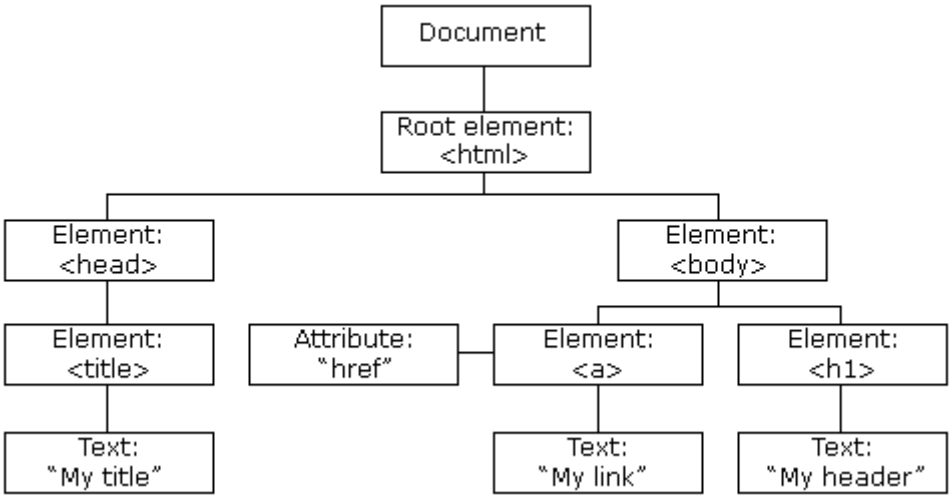
Définir et en mettre œuvre des cas de tests appropriés pour des pages Web interactives et documenter dans le procès-verbal de tests.

2. JavaScript

2.1. Listener

2.1.1. DOM

DOM ou Document Object Model permet de charger tous les éléments de la page



2.1.2. GetID

Dans le javascript, il est possible d'interfacer avec l'HTML grâce aux ID. cela permet d'exécuter du code et de rendre le tout dynamique.

```
document.getElementById("<ID>")
```

2.1.3. EventListener

Les écouteurs, comme en java, permettent d'exécuter une fonction lorsqu'une action précise est effectuée (clic sur un bouton, texte entré dans une textbox …)

Pour en ajouter, c'est comme ça :

```
document.getElementById("<ID>").addEventListener("click",<fonction>);
```

2.1.4. InnerHTML

InnerHTML permet de change le code HTML depuis le javascript. Cela permet d'avoir un retour comme un texte ou une image qui change.

```
document.getElementById("<ID>").innerHTML = "nouveau texte";
```

2.1.5. Ecouteur JS

```
<button id="testez">Teste-moi</button>
<p id="info"> </p>
```

```
function initCtrl() {
  // Ecouteur du bouton "Testez-moi"
  document.getElementById("testez").addEventListener("click", testez);
}

function testez() {
   document.getElementById("info").innerHTML =
      "C'est <b>Simon Gendre</b> qui a pressé le bouton !";
}
```

2.1.6. Ecouteur HTML

```
<button id="testez" onclick="testez()">Teste-moi, James</button>
<p id="info"> </p>
```

```
function testez() {
   document.getElementById("info").innerHTML = "C'est <b>Simon Gendre</b> qui a pressé le
bouton !";
}
```

2.2. Tests logiques

En JS, il existe deux types de tests. Étant donné que les variables sont typées automatiquement, il est possible de comparer soit la valeur ou soit la valeur et le type.

Cela permet de vérifier que (float) 3.5 et (long) 3.5 sont égaux mais pas de même type

```
“NaN” == NaN --> true
“NaN” === NaN --> false

“true” == true --> true
“false” === false --> false
```

2.3. Boucles

Comme en Java, le Javascript possède plusieurs types de boucle : for, while et do.

2.3.1. For

```
let out = "for (let i = 0; i > 5; i++){...}";

  for (let i = 0; i < 5; i++) {
    out += `<br>${i}`;
  }
  out += `<br> --> À utiliser si on sait que l'on veut itérer x fois (x connu avant de
commencer la boucle)`;
```

2.3.2. While

```
let out = " while (i < 5) {...}";
  let i = 0;
  while (i < 5) {
    out += `<br>${i}`;
    i++;
  }
  out += `<br> --> À utiliser si on ne sait pas le nombre d'itérations au démarrage de la
boucle`;
```

2.3.3. Do while

```
let out = "  do {...} while (i < 5)";
  let i = 0;
  do {
    out += `<br>${i}`;
    i++;
  } while (i < 5);
  out += `<br> --> À utiliser si on ne sait pas le nombre d'itérations au démarrage de la
boucle mais avec un passage obligatoire`;
```

2.4. Fonctions

2.4.1. Basique

```
  function a() {
    let val = 1;
    console.log(val);
  }
  a(); // => exécution de la méthode et affichage de 1 dans la console
```

2.4.2. Anonyme

```
  let b = function () {
    let val = 2;
    console.log(val);
  };
  b(); // => exécution de la méthode et affichage de 2 dans la console
```

2.4.3. Arrow

```
let c = (a, b) => a + b;
  console.log(c(2, 5)); // = 7
```

```
  let d = (val) => val * val; // avec un paramètre
  console.log(d(5)); // = 25
```

```
  let e = (a, b) => {
    // avec plusieurs instructions
    let somme = a + b;
    return somme;
  };
  console.log(e(5, 7)); // = 12
```

2.4.4. IIFE

```
(function() {
    let val = 3;
    console.log(val) ;
 })() ;    // => exécution et affichage de 3 dans la console
```

```
 (() => {
    let val = 4;
    console.log(val) ;
 })() ;  // => exécution et affichage de 4 dans la console via une fonction flèche
```

3. JSON

3.1. Fonctionnement

JSON, ou de son acronyme JavaScript Object Notation, est un format de données léger et largement utilisé pour représenter et échanger des informations structurées entre des applications. Les données JSON sont basées sur une syntaxe d'objet JavaScript et peuvent représenter des tableaux, des objets, des chaînes, des nombres et des booléens. JSON est facile à lire et à écrire pour les humains, tout en étant facilement analysable et générable par les machines.

Les données JSON sont généralement composées de paires clé-valeur, où les clés sont des chaînes de caractères et les valeurs peuvent être de différents types

3.2. Fonctions

3.2.1. JSON.parse()

La fonction JSON.parse() est une méthode JavaScript intégrée qui permet de convertir une chaîne de caractère JSON en un objet JavaScript correspondant. Lorsque des données JSON sont reçus depuis un serveur ou un fichier, il est possible d'utiliser JSON.parse() pour les analyser et les transformer en objets JavaScript qui peuvent ensuite être manipuler et utiliser dans du code.

3.2.2. JSON.stringify()

La fonction JSON.stringify() est une méthode JavaScript qui permet de convertir un objet JavaScript en une chaîne de caractères. Lorsque l'on aimerait envoyer un objet JavaScript à un serveur ou le stocker dans un fichier, il est possible utiliser JSON.stringify() pour le sérialiser en une représentation JSON.

3.3. Accéder aux valeurs

3.3.1. Accéder aux propriétés d'un objet JSON

Lorsque l'on travaille avec un objet JSON, on peut accéder à ses propriétés et valeurs en utilisant la notation point. Par exemple, si on a un objet JSON appelé "json" avec une propriété "key", on peut accéder à la valeur de cette propriété en utilisant la syntaxe "json.key".

```
json.key = valeur;
Console-log(json.key);
```

3.3.2. Accéder aux propriétés dynamiquement avec la notation entre crochets

Parfois, le nom de la propriété que l'on souhaite accéder est dynamique ou stocké dans une variable. Dans de tels cas, la notation entre crochets est utile. En l'utilisant, on peut accéder à une propriété en utilisant une valeur dynamique ou une variable comme clé. Par exemple, si on a une variable appelée "dynamicKey" contenant la valeur "name", on peut accéder à la propriété correspondante dans l'objet JSON en utilisant la syntaxe "json[dynamicKey]".

```
json[dynamicKey] = value ;
console.log(json[dynamicKey]) ;
```

4. JQuery

JQuery est une bibliothèque JavaScript populaire pour la manipulation du DOM, offrant une syntaxe concise et simplifiée. Il facilite les interactions avec le DOM, les animations, les effets visuels et les requêtes AJAX.

Avec une communauté active et une documentation complète, il reste une option solide pour les projets web. Bien que de nouvelles technologies aient émergé, jQuery reste utilisé pour les projets plus simples ou nécessitant une compatibilité avec d'anciens navigateurs. En somme, jQuery est une bibliothèque légère, facile à utiliser et largement adoptée pour développer des interfaces interactives sur le web.

4.1. Comment reconnaitre

JQuery est assez à reconnaitre de JS. Pour commencer, un signe « $ » (l'objet JQuery) précède les lignes de codes.

Voici quelques exemples qui permettent de bien faire la différence entre les deux.

Pour définir le texte d'un élément avec JavaScript, on utilise la propriété textContent :

```
// JavaScript
info.textContent = "Une info JavaScript!";
```

En revanche, avec jQuery, on utilise la méthode text() :

```
// jQuery
$("#info").text("C'est une info jQuery");
```

Pour ajouter un élément à l'intérieur de notre élément info avec JavaScript, on crée d'abord un élément et on l'ajoute à l'élément info en utilisant la méthode appendChild() :

```
// JavaScript
info.appendChild(document.createElement("div"));
```

En utilisant jQuery, on peut simplement utiliser la méthode append() :

```
// jQuery
$("#info").append("<div>C'est un div</div>");
```

Ces exemples mettent en évidence les différences entre JavaScript et jQuery en termes de manipulation du contenu d'un élément et d'ajout d'éléments supplémentaires.

4.2. Retrouver des éléments

4.2.1. Par balise

Pour retrouver des éléments en utilisant la balise, on utilise la syntaxe suivante en jQuery :

```
$("button")
$("body")
$("div")
```

Ces sélecteurs jQuery permettent de récupérer tous les éléments correspondant à la balise spécifiée. Par exemple, $("button") retournera tous les éléments <button> de la page.

4.2.2. Par ID

Pour retrouver des éléments en utilisant l'ID, on utilise la syntaxe suivante en jQuery :

```
$("#liste")
$("#accueil")
$("#logo")
```

En utilisant le préfixe #, on peut cibler les éléments ayant un ID spécifique. Par exemple, $("#liste") retournera l'élément ayant l'ID "liste".

4.2.3. Par classe

Pour retrouver des éléments en utilisant la classe, on utilise la syntaxe suivante en jQuery :

```
$(".data")
$(".hidden")
$(".menu")
```

En utilisant le préfixe '.', on peut cibler les éléments ayant une classe spécifique. Par exemple, $(".data") retournera tous les éléments ayant la classe "data".

4.2.4. Retrouver parent, enfant, frère, suivant ou précédent

Pour retrouver le parent d'un élément, on utilise la méthode .parent() :

```
$("#element").parent();
```

Pour retrouver les enfants d'un élément, on utilise la méthode .children() :

```
$("#element").children();
```

Pour retrouver le frère suivant d'un élément, on utilise la méthode .next() :

```
$("#element").next();
```

Pour retrouver le frère précédent d'un élément, on utilise la méthode .prev() :

```
$("#element").prev();
```

Enfin, pour retrouver tous les frères d'un élément (à l'exception de l'élément lui-même), on utilise la méthode .siblings() :

```
$("#element").siblings();
```

4.3. CSS

Avec jQuery, il est possible d'ajouter du CSS aux éléments sélectionnés en utilisant la méthode .css(propriété, valeur).

Voici un exemple :

```
$("button").css("border", "3px solid red");
```

```
$("sélecteur").css({
  "propriété1": "valeur1",
  "propriété2": "valeur2",
  // ...
});
```

4.4. Fonction au chargement

Cette fonction peut être comparée à l'attribut 'onload'. Elle permet d'exécuter du code au moment où la page à finit de charger.

```
$(document).ready(function(){
    // Votre code
```

```
});
```

4.5. Imbriquer des fichier HTML

JQuery permet d'imbriquer des fichiers avec la fonction 'load' dans un élément du DOM.

```
$("#view").load("fichier.html", function () {
          //faire qqch apres le chargement
}
```

4.6. Intégration de JQuery

JQuery peut être ajouté à un projet en important ce script. Il faudra faire attention à le charger avant les scripts qui utilisent JQuery pour ajouter les erreurs

```
<script src="https://code.jquery.com/jquery-3.7.0.min.js"></script>
<script src="scriptAvecJQ.js"></script>
```

4.7. Exemples

4.7.1. Modifier le CSS

Pour modifier le CSS d'un élément avec jQuery, on utilise la méthode .css() en spécifiant la propriété et la valeur souhaitées. Par exemple :

```
$("#element").css("color", "red");
```

Cela permet de changer la couleur du texte de l'élément ayant l'ID "element" en rouge.

4.7.2. Ajouter des éléments

Pour ajouter de nouveaux éléments à une page HTML avec jQuery, on utilise des méthodes telles que .append(), .prepend() ou .after(). Par exemple :

```
$("#container").append("<p>Nouveau paragraphe</p>");
```

Cela ajoute un nouveau paragraphe à l'intérieur de l'élément ayant l'ID "container".

4.7.3. Supprimer des éléments

Pour supprimer des éléments de la page avec jQuery, on utilise la méthode .remove(). Par exemple :

```
$("#element").remove();
```

Cela supprime complètement l'élément ayant l'ID "element" de la page.

4.7.4. Gérer des événements

JQuery facilite la gestion des événements tels que le clic (click) et le survol (hover). Par exemple :

```
$("#element").click(function() {
  // Code à exécuter lors du clic sur l'élément
});
```

```
$("#element").hover(function() {
  // Code à exécuter lorsque l'élément est survolé
});
```

Cela permet de définir des actions spécifiques à effectuer lorsque l'élément est cliqué ou survolé.

4.7.5. Cacher/afficher des éléments

Pour masquer ou afficher des éléments de la page avec jQuery, on utilise les méthodes .hide() et .show(). Par exemple :

```
$("#element").hide(); // Masquer l'élément
```

```
$("#element").show(); // Afficher l'élément
```

```
$("#element").toggle(); // alterne entre les deux
```

Cela permet de contrôler la visibilité des éléments de la page.

5. WebServices (API)

Les WebServices, également connus sous le nom d'API (Application Programming Interface), permettent aux applications de communiquer entre elles et d'échanger des données. Voici quelques aspects importants liés aux WebServices :

5.1. Postman

Postman est un outil populaire utilisé pour tester et développer des API. Il offre une interface conviviale pour envoyer des requêtes HTTP et visualiser les réponses.

5.2. SOAP

SOAP (Simple Object Access Protocol) est un protocole de communication basé sur XML utilisé pour échanger des données entre des applications via des WebServices.

Il est utilisé dans les applications d'entreprise pour assurer la fiabilité, la sécurité et la compatibilité inter plateforme des échanges de données.

5.3. REST

REST (Representational State Transfer) est un style architectural utilisé pour concevoir des services Web. Il est basé sur les principes du protocole HTTP et utilise des verbes HTTP tels que GET, POST, PUT et DELETE pour effectuer des opérations sur les ressources.

5.3.1. GET

La méthode GET est utilisée pour récupérer des données à partir d'une ressource spécifique. Par exemple, on peut utiliser une requête GET pour récupérer les détails d'un utilisateur à partir d'une API.

5.3.2. POST

La méthode POST est utilisée pour envoyer des données à une ressource spécifique pour la création de nouvelles entités. Par exemple, on peut utiliser une requête POST pour créer un nouvel utilisateur en soumettant les détails de l'utilisateur à une API.

5.3.3. PUT

La méthode PUT est utilisée pour mettre à jour une ressource spécifique ou la créer si elle n'existe pas. Si la ressource existe déjà, la méthode PUT la mettra à jour avec les nouvelles données. Si la ressource n'existe pas, la méthode PUT la créera. Cependant, si une ressource est mise à jour 10 fois avec la même requête PUT, cela ne créera qu'une seule entrée.

5.3.4. DELETE

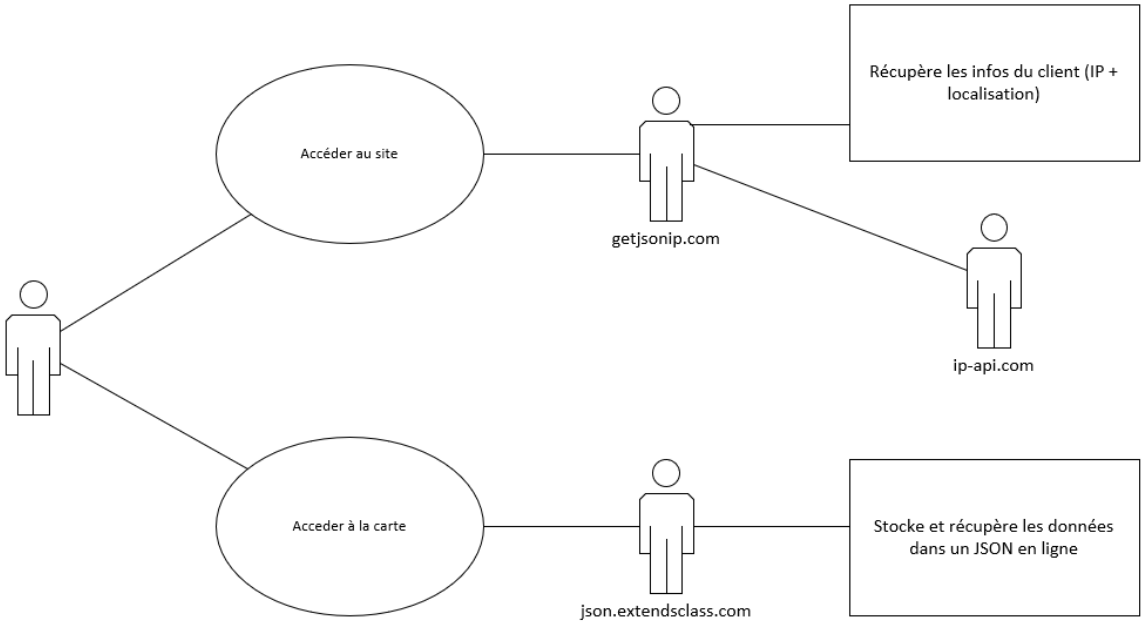
La méthode DELETE est utilisée pour supprimer une ressource spécifique. Par exemple, on peut utiliser une requête DELETE pour supprimer un utilisateur d'une API.

6. Projet

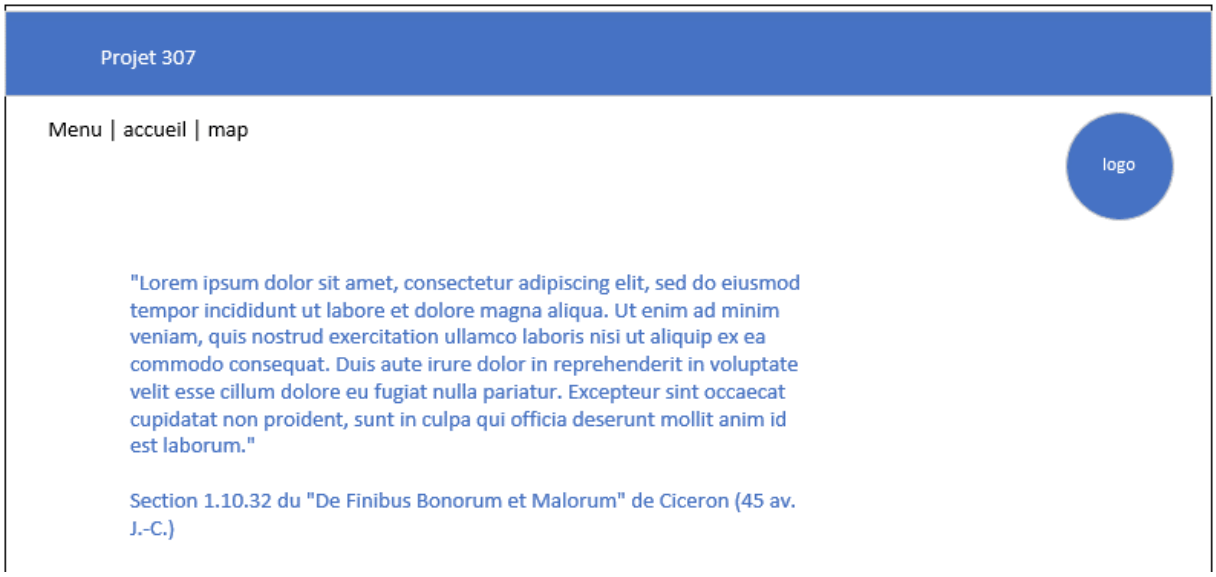
Ce projet consiste à faire une carte affichant un point pour chaque IP qui s'est connectée au site Web.

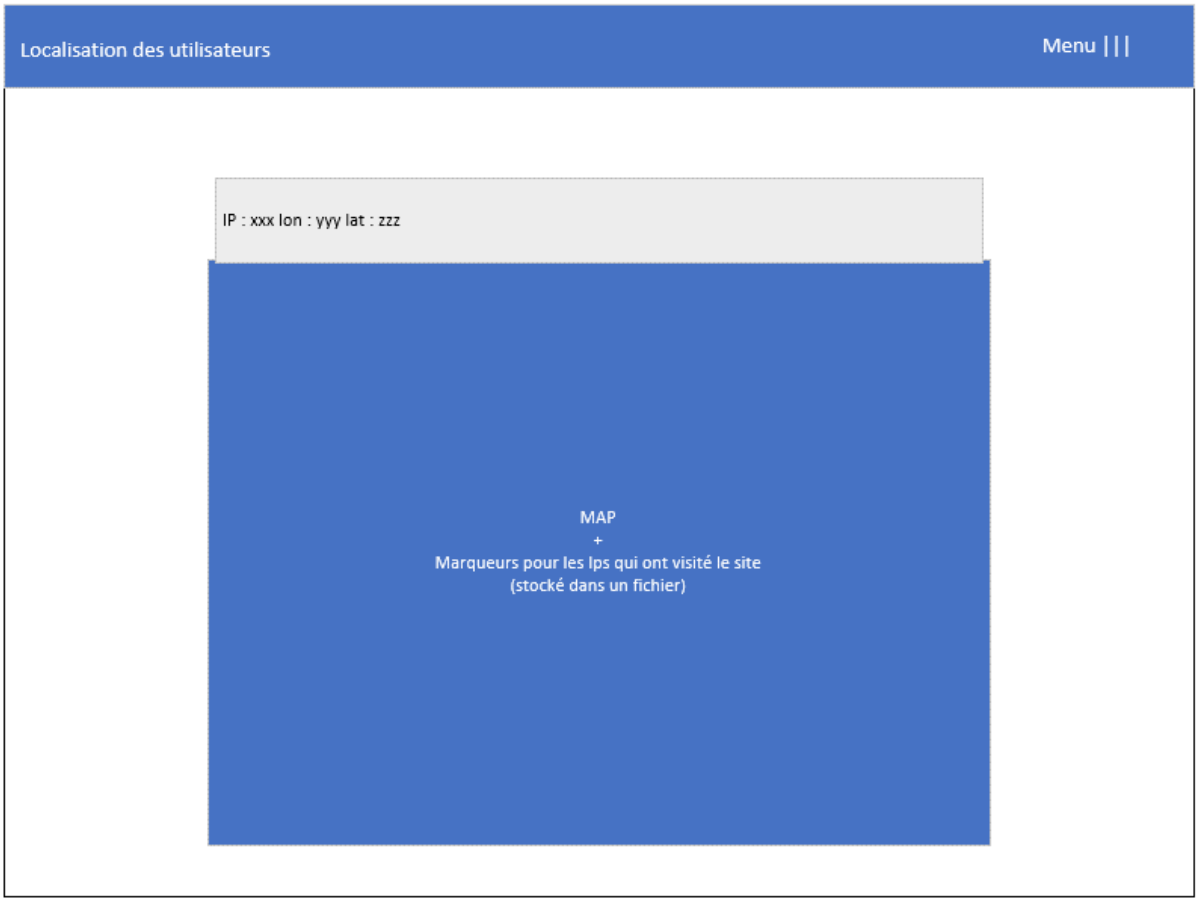
6.1. Analyse

6.1.1. Use case



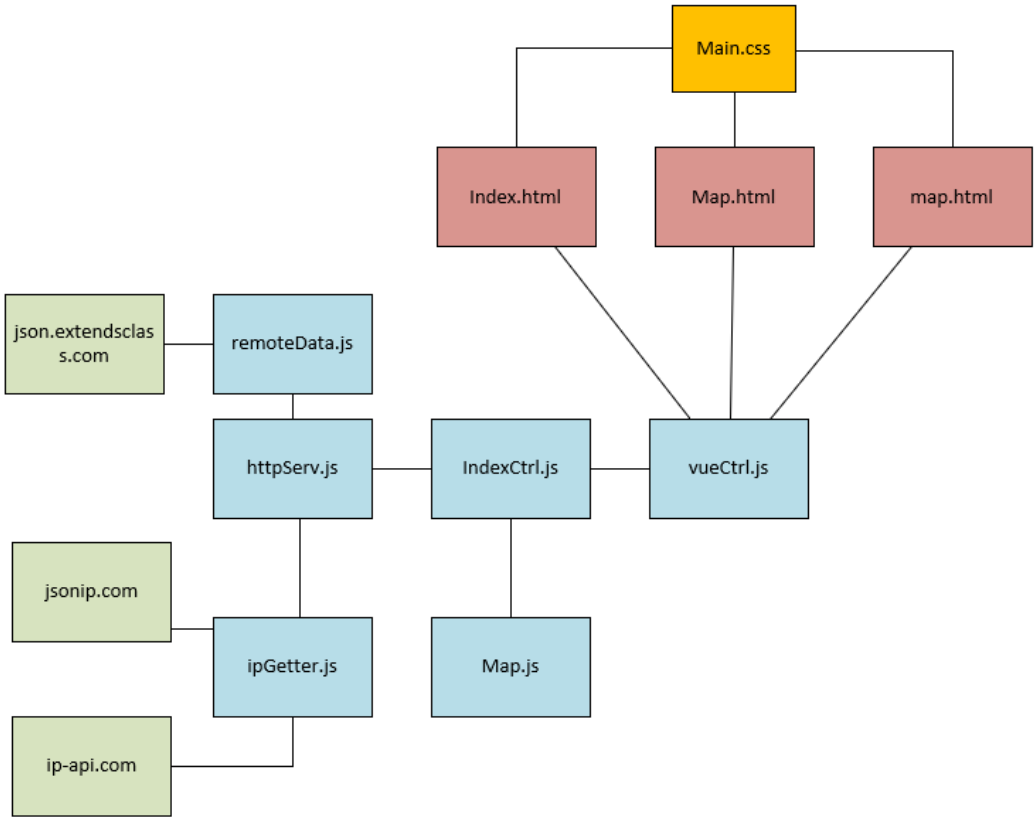
6.1.2. Maquette




Localisation des utilisateurs
Menu |||
IP : xxx lon : yyy lat : zzz
MAP
+
Marqueurs pour les Ips qui ont visité le site
(stocké dans un fichier)

6.2. Conception

6.2.1. Diagramme de navigation



6.3. Implémentation

6.3.1. Extraits de code

Voici le code servant à ajouter la carte et les marqueurs :

```
/**
   * cette méthode affiche la carte
   * @param {*} mapElement l'element contenant la map ($("#carte"))
   */
afficherCarte(mapElement) {
    // Utilisez l'élément "map" pour afficher la carte Leaflet ou effectuer d'autres
opérations
    this.mymap = L.map(mapElement[0]).setView([46.802, 7.146], 13);

    // Ajouter une couche de tuiles (carte de base)
    L.tileLayer("https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png", {
      attribution: "© OpenStreetMap contributors",
    }).addTo(this.mymap);
  }
  /**
   *
   * @param {*} lon  coordonnées du marqueur
   * @param {*} lat coordonnées du marqueur
   * @param {*} label texte du marqueur
   * @param {*} color couleur du marqueur (pas encore implémenté)
   */
  addMarker(lon, lat, label, color) {
    // Ajouter un marqueur simple
    let marker = L.marker([lon, lat]).addTo(this.mymap);
```

```
    // centre la carte sur le point ajouté
    this.mymap.setView([lon, lat], 10);
    // Ajouter une info-bulle au marqueur
    marker.bindPopup(label, { offset: [0, -30] }).openPopup();
  }
```

6.4. Tests

Test	Attendu	Réel	OK/NOK
Aller sur le site (http)	Afficher Accueil	Afficher accueil	OK
Aller sur le site (https)	Afficher Accueil	Afficher accueil	OK
Aller sur la carte (http)	Les points s'affichent (vous : IP)	Les points s'affichent (vous : IP)	OK
Aller sur la carte (https)	Les points s'affichent (vous : IP)	Le point (vous : IP) n'est pas affiché mais les autre oui	~

6.5. Hébergement et fonctionnement

Le site est disponible sur http://307.gendres.emf-informatique.ch et le repo GitHub ici. La documentation se trouve dans le README.md

6.5.1. Explications API

Ce projet utilise 3 APIs :

- https://jsonip.com/
- https://ip-api.com/docs/api:json
- https://extendsclass.com/json-storage.html

Ces APIs permettent respectivement de :

- Récupérer l'IP du client
- Récupérer les coordonnées GPS d'une IP
- Stocker et récupérer un JSON en ligne

À cause de IP-api, le site ne peut pas être en HTTPS sans payer un abonnement chez eux.

6.6. Problèmes

6.6.1. HTTPS non fonctionnel

À cause d'un API (http://ip-api.com/json/) qui ne fournit pas de SSL à moins d'utiliser la partie payante, le site est obligé à tourner sous http (qui n'est pas optimale) sous peine d'avoir la requête bloquée par le navigateur.

7. Conclusion

Le module 307 est un module très amusant et enrichissant. J'avais déjà essayé de faire du JS de mon côté mais sans trop réussir. Ce module m'a beaucoup aidé et je risque de refaire des applications de mon côté.

7.1. Amélioration / proposition

Afin de rendre le module plus agréable, je proposerais de faire une semaine d'exercices en moins et de commencer le projet plus tôt. (2 semaines exercices et 3 pour le projet).