S

ssss

307 - Find your Hero

Lopes Monteiro Miguel Tiago

Version 1 du 20.06.2021  
Module du 17.05.2021 au 15.06.2021

Table des matières

[1 Introduction module et projet 2](#_Toc75119765)

[1.1 Définition du module 2](#_Toc75119766)

[1.2 Définition du projet personnel 2](#_Toc75119767)

[2 Analyse à faire complètement avec EA 6](#_Toc75119768)

[2.1 Use case avec les explications de l'application 6](#_Toc75119769)

[2.2 Activity Diagram d'un cas complet navigant dans l'application avec les explications 7](#_Toc75119770)

[3 Conception à faire complétement avec EA 7](#_Toc75119771)

[3.1 Conception MVC du client 8](#_Toc75119772)

[4 Implémentation de l'application 8](#_Toc75119773)

[4.1 Partie HTML et CSS 8](#_Toc75119774)

[4.1.1 Index 9](#_Toc75119775)

[4.1.2 Login 9](#_Toc75119776)

[4.1.3 Accueil 9](#_Toc75119777)

[4.1.4 CSS 9](#_Toc75119778)

[4.2 Partie JS et les commentaires 10](#_Toc75119779)

[4.2.1 Index.js 10](#_Toc75119780)

[4.2.2 service.js 10](#_Toc75119781)

[4.2.3 login.js 11](#_Toc75119782)

[4.2.4 requete.js 12](#_Toc75119783)

[5 Tests de l'application 13](#_Toc75119784)

[5.1 Client fonctions internes 13](#_Toc75119785)

[5.2 Client fonctions extérieures 13](#_Toc75119786)

[6 Hébergement et installation du projet complet 13](#_Toc75119787)

[7 Fonctionnement du projet hébergé 14](#_Toc75119788)

[8 Auto-évaluation et conclusion 15](#_Toc75119789)

[9 Tests technologiques selon les exercices 15](#_Toc75119790)

[9.1 Exercice 1 - base HTML5 15](#_Toc75119791)

[9.2 Exercice 2 - login 16](#_Toc75119792)

[9.3 Exercice 3 - login avec PHP 16](#_Toc75119793)

[9.4 Exercice 4 – utilisation de jsfiddle 18](#_Toc75119794)

[9.5 Exercice 5 – variables en JavaScript 18](#_Toc75119795)

[9.6 Exercice 6 – la structure de contrôle SWITCH 18](#_Toc75119796)

[9.7 Exercice 7 – les boucles en JavaScript 19](#_Toc75119797)

[9.8 Exercice 8 – tableau associatif ou tableau d’objets JSON 20](#_Toc75119798)

[9.8.1 Déclaration du tableau 20](#_Toc75119799)

[9.8.2 Parcours du tableau 20](#_Toc75119800)

[9.9 Exercice 9 – démo sur les objets en JavaScript 21](#_Toc75119801)

[9.9.1 JSON 21](#_Toc75119802)

[9.9.2 Objet 21](#_Toc75119803)

[9.9.3 Object fonction 21](#_Toc75119804)

[9.9.4 Fonction prototype 22](#_Toc75119805)

[9.9.5 Classe 22](#_Toc75119806)

[9.10 Exercice 10 – programmation orientée *« objets »* en JS 22](#_Toc75119807)

[9.11 Exercice 11 – programmation orientée *« classe »* en JS 23](#_Toc75119808)

[9.12 Exercice 12 – les closures – compteur 23](#_Toc75119809)

[9.13 Exercice 13 – les IIFE en JavaScript 24](#_Toc75119810)

[9.14 Exercice 14 - bases de jQuery 24](#_Toc75119811)

[9.15 Exercice 15 – première utilisation de jQuery 25](#_Toc75119812)

[9.15.1 JavaScript 25](#_Toc75119813)

[9.15.2 JQuery 26](#_Toc75119814)

[Exercice 16 – lecture jQuery & Bootstrap 26](#_Toc75119815)

[9.16 Exercice 17 – test de web services 27](#_Toc75119816)

[9.16.1 CORS 27](#_Toc75119817)

[9.17 Exercice 18 – application HTML5 d’appel à un web service 27](#_Toc75119818)

[9.18 Exercice 19 – application HTML5 avec double service web 28](#_Toc75119819)

[9.19 Exercice 20 – application *« login »* sur un serveur PHP 29](#_Toc75119820)

[10 Conclusion 30](#_Toc75119821)

[Ce que j’ai appris 30](#_Toc75119822)

[Ce que j’ai aimé 30](#_Toc75119823)

[Ce que j’ai moins aimé 30](#_Toc75119824)

# Introduction module et projet

## Définition du module

* Rappel sur les bases des techniques du Web côté client (HTML, CSS et autres)
* Tester un environnement de travail efficace pour le développement web
* Comprendre les bases du développement web côté client (intelligence locale) en étudiant un langage de script et une ou l’autre bibliothèque adaptée
* Comprendre les bases de la consommation de services web existants et tester en pratique par des requêtes HTTP
* Comprendre les bases d’un découpage MVC d’une application web côté client et tester en pratique
* Mettre en forme des données JSON ou XML provenant d’un serveur.
* Comprendre la problématique de l’adaptation des applications Web interactives à des écrans de diverses natures et mettre en pratique
* Créer et améliorer un projet global « accompagné », basé sur les techniques vues pendant le module

## Définition du projet personnel

Pour le projet de ce module j’ai décidé de réaliser un projet qui s’intitule “Find your Hero”

L’idée est d’utiliser l’api mise à disposition par MARVEL, qui est la responsable de la création d’une énorme quantité de super héros, à travers les comics, dessins animés films, pour trouver des informations sur ces mêmes super héros. On trouverait par exemple son histoire ou un bref résumé du personnage en entant le nom de l’héros dans une barre de recherche et en recevant en JSON une réponse avec tout ce contenu. De plus on peut avoir des images avec

L’idée est donc de faire une grande bibliothèque avec tous ces personnages et ces caractéristiques. De plus nous auront un système de login avec un système captcha au début pour nous connecter.

* Site de marvel : <https://developer.marvel.com/>
* Site avec la documentation pour réaliser des tests : <https://developer.marvel.com/docs>

Exemple avec le JSON comportent toutes les informations sur le personnage « iron man » :

L’exemple a été raccourci afin de ne pas remplir trop d’espace

{

    "code": 200,

    "status": "Ok",

    "copyright": "© 2021 MARVEL",

    "attributionText": "Data provided by Marvel. © 2021 MARVEL",

    "attributionHTML": "<a href=\"http://marvel.com\">Data provided by Marvel. © 2021 MARVEL</a>",

    "etag": "d6e325636d3760be4aa7193a13d70ef4eb405635",

    "data": {

      "offset": 0,

      "limit": 20,

      "total": 1,

      "count": 1,

      "results": [

        {

          "id": 1009368,

          "name": "Iron Man",

          "description": "Wounded, captured and forced to build a weapon by his enemies, billionaire industrialist Tony Stark instead created an advanced suit of armor to save his life and escape captivity. Now with a new outlook on life, Tony uses his money and intelligence to make the world a safer, better place as Iron Man.",

          "modified": "2016-09-28T12:08:19-0400",

          "thumbnail": {

            "path": "http://i.annihil.us/u/prod/marvel/i/mg/9/c0/527bb7b37ff55",

            "extension": "jpg"

          },

          "resourceURI": "http://gateway.marvel.com/v1/public/characters/1009368",

          "comics": {

            "available": 2576,

            "collectionURI": "http://gateway.marvel.com/v1/public/characters/1009368/comics",

            "items": [

              {

                "resourceURI": "http://gateway.marvel.com/v1/public/comics/43495",

                "name": "A+X (2012) #2"

              },

              {

                "resourceURI": "http://gateway.marvel.com/v1/public/comics/12668",

                "name": "Alpha Flight (1983) #127"

              },

              {

                "resourceURI": "http://gateway.marvel.com/v1/public/comics/55311",

                "name": "The Amazing Spider-Man (2015) #13"

              }

            ],

            "returned": 20

          },

          "series": {

            "available": 630,

            "collectionURI": "http://gateway.marvel.com/v1/public/characters/1009368/series",

            "items": [

              {

                "resourceURI": "http://gateway.marvel.com/v1/public/series/16450",

                "name": "A+X (2012 - 2014)"

              },

              {

                "resourceURI": "http://gateway.marvel.com/v1/public/series/23446",

                "name": "Amazing Spider-Man: Worldwide Vol. 2 (2017)"

              }

            ],

            "returned": 20

          },

          "stories": {

            "available": 3902,

            "collectionURI": "http://gateway.marvel.com/v1/public/characters/1009368/stories",

            "items": [

              {

                "resourceURI": "http://gateway.marvel.com/v1/public/stories/670",

                "name": "X-MEN (2004) #186",

                "type": "cover"

              },

              {

                "resourceURI": "http://gateway.marvel.com/v1/public/stories/892",

                "name": "THOR (1998) #81",

                "type": "cover"

              }

            ],

            "returned": 20

          },

          "urls": [

            {

              "type": "detail",

              "url": "http://marvel.com/comics/characters/1009368/iron\_man?utm\_campaign=apiRef&utm\_source=ceb9db9f7dab22d3e21c5f48ce074034"

            },

            {

              "type": "wiki",

              "url": "http://marvel.com/universe/Iron\_Man\_(Anthony\_Stark)?utm\_campaign=apiRef&utm\_source=ceb9db9f7dab22d3e21c5f48ce074034"

            }

          ]

        }

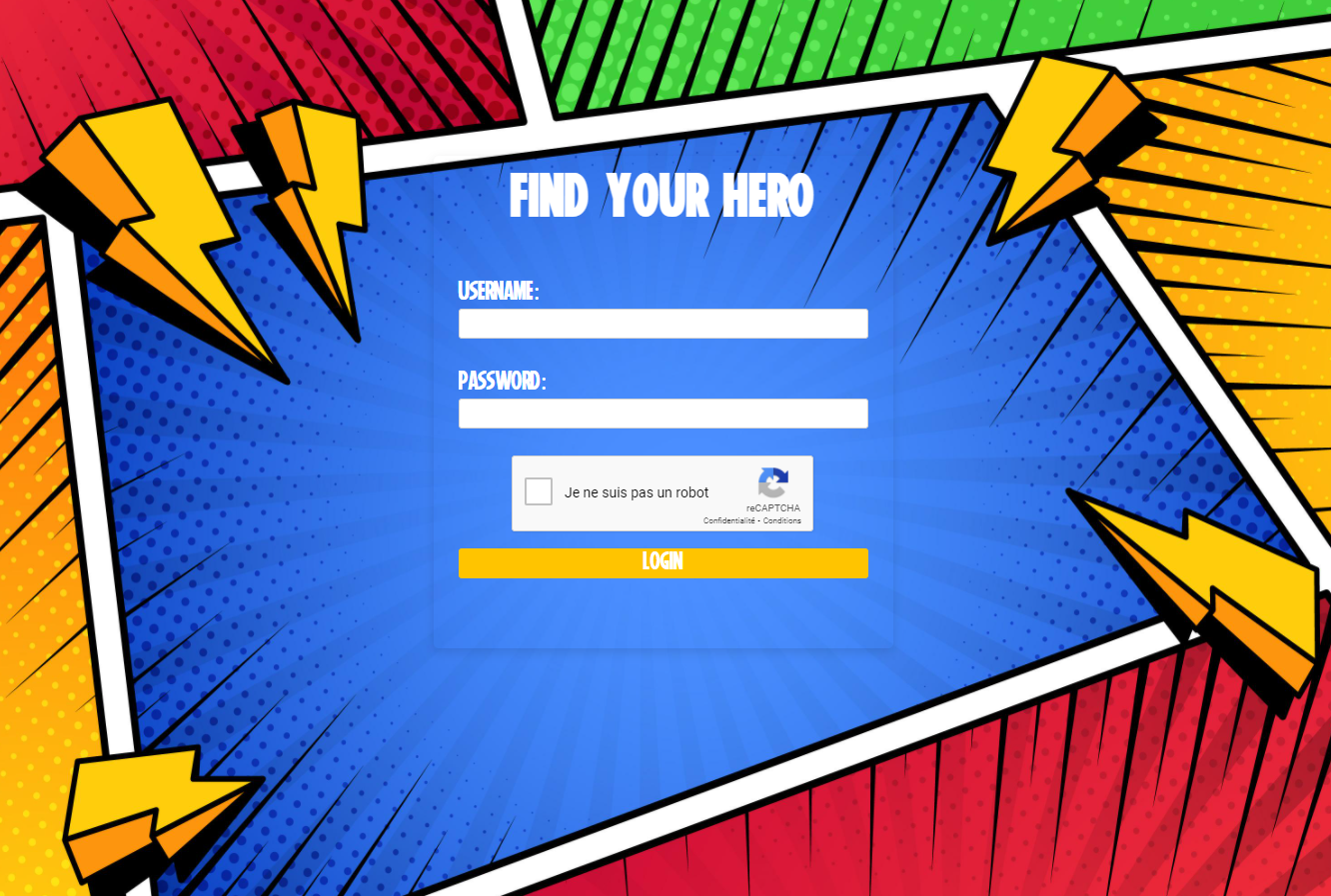
      ]

    }

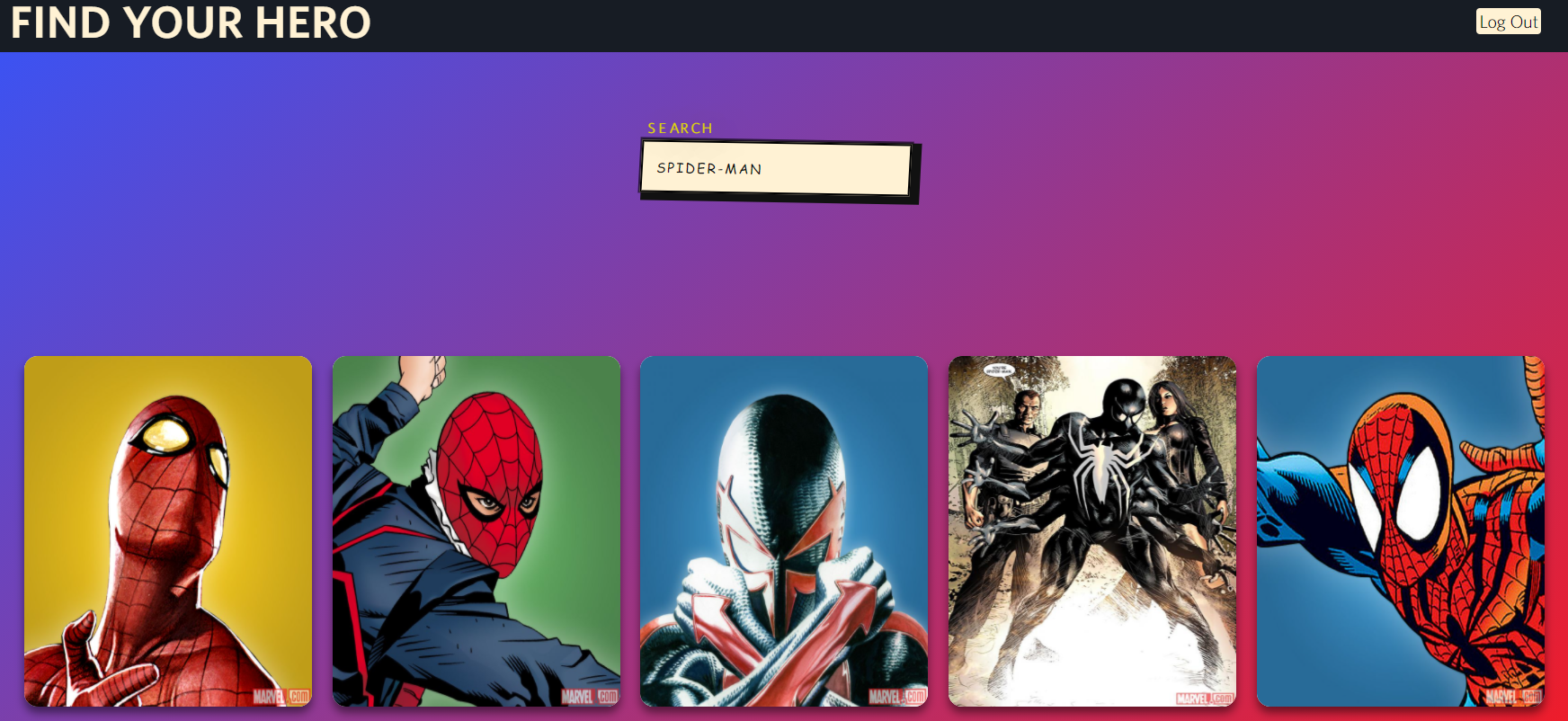
  }

Première idée visuelle du site :

Page d’accueil pour se loger

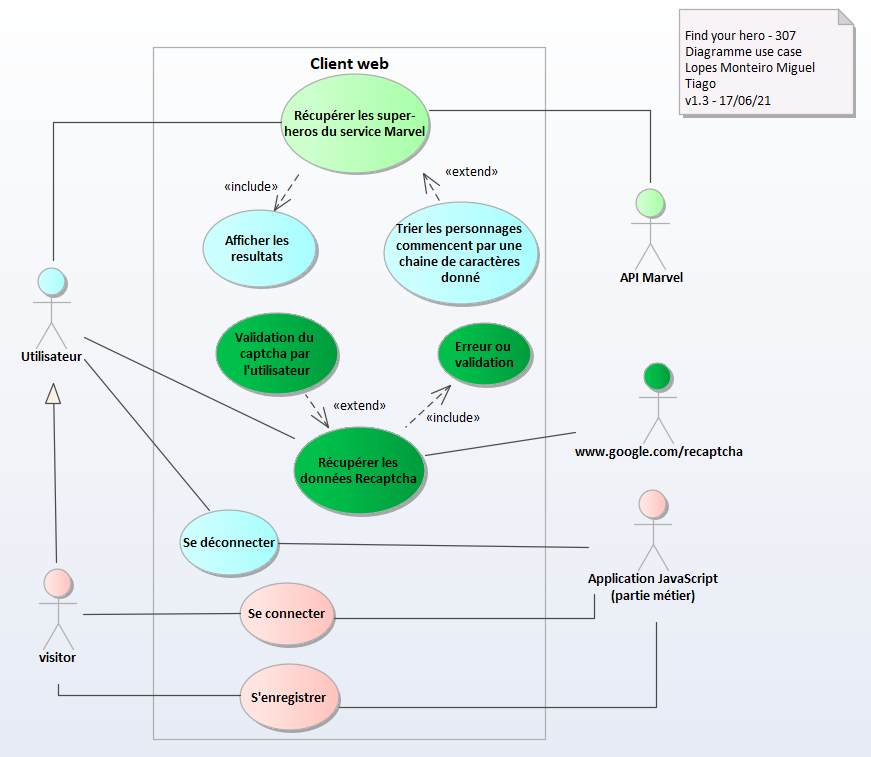


Page qui affichera les héros



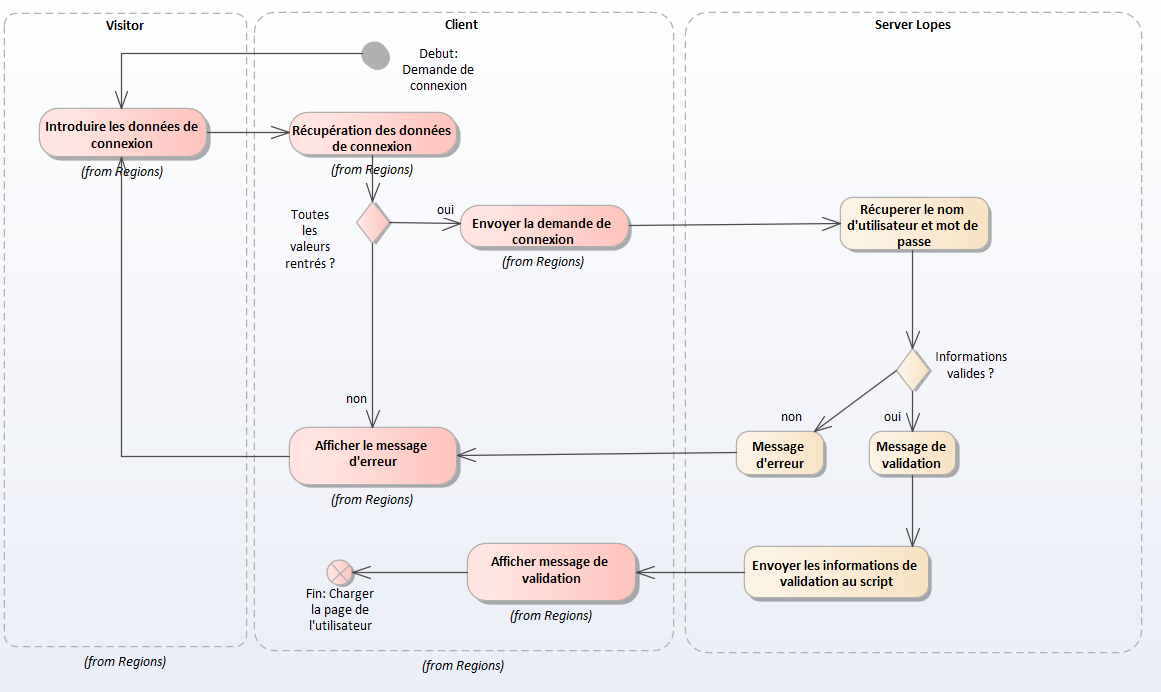
# Analyse à faire complètement avec EA

## Use case avec les explications de l'application



Voici le schéma « use case » du site. Premièrement le visiteur va arriver sur le site et il aura 2 possibilités, soit se connecter soit s’enregistrer. Quand le visiteur se connecte avec un login valable celui-ci devient un utilisateur et c’est à partir de ce moment qu’il peut faire les autres possibilités du site. Il pourra alors chercher dans L’api de Marvel les plusieurs héros qui existent dans le monde Marvel, selon une chaine de caractères donnés par l’utilisateur.

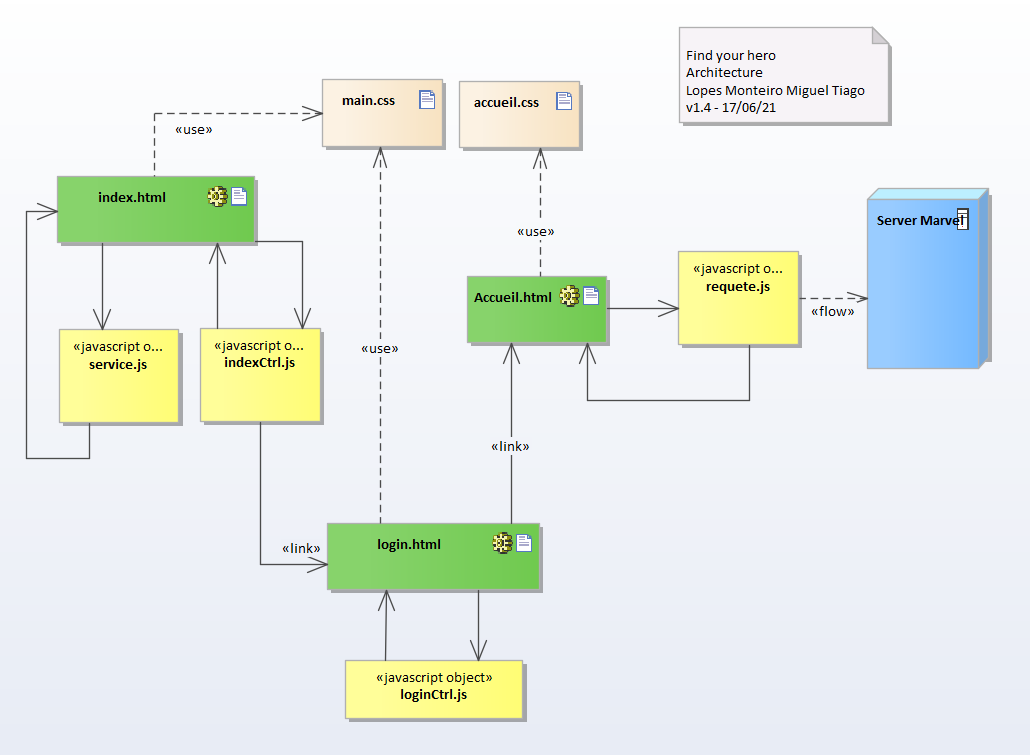
## Activity Diagram d'un cas complet navigant dans l'application avec les explications



Voici le diagramme d’activité pour le login de mon application web. Ici on commence par le visiteur qui va introduire les données de connexion, ensuite l’application web va récupérer ces informations et vérifier dans un premier temps si l’utilisateur les a bien mises. Il testera ensuite si les informations rentrées sont correctes, si une erreur se produit, un message d’erreur d’inscription sera envoyé et affiché. Si l’inscription s’est bien effectuée, les informations de l’utilisateur inscrit seront envoyées au client. Le client va ensuite les récupérer et afficher la validation de l’inscription. Pour terminer la page de l’utilisateur connecté sera chargé.

# Conception à faire complétement avec EA

## Conception MVC du client



Voici la conception MVC de mon d’un client. Ici on peut voir en vert tous les fichiers html qui possèdent un style CSS, selon la page c’est soit « main.css » ou alors « accueil.css ». On peut aussi voir les différents liens entre les fichiers html et JS. Ici le fichier « login.html » possède un fichier JS qui nous permet de gérer le login de la page. Quand la connexion est validée on redirectionne l’utilisateur vers la page « Accueil.html » ce qu’il lui permettra d’accéder au fichier « requête.js » qui pourra faire des requêtes au serveur de Marvel pour avoir les informations des héros en retour.

# Implémentation de l'application

Lors de cette partie je vais montrer plus en détails chacun des fichiers de mon projet, et expliquer les différents problèmes rencontrés.

## Partie HTML et CSS

Pour ce qui est de la partie html, j’ai utilisé 3 pages différentes. Et pour la partie CSS j’ai utilisé 2 afin de mieux organiser le site même si tout serait faisable avec un seul fichier.

### Index

Dans cette page on est confronté directement avec la page d’accueil. Celle-ci a une base qui s’appelle « index.html », elle appellera ensuite une view qui contiendra le html nécessaire pour le login. On importe aussi les fichiers javascript et toutes les librairies comme captcha par exemple.

### Login

Ici on a un username avec un mot de passe et un captcha avant de pourvoir valider le login. Le plus difficile à réaliser dans cette page était le captcha. Pour pouvoir avoir un captach il faut se rendre dans la page de google developers : <https://developers.google.com/recaptcha> se connecter avec un compte google et suivre les étapes. Par après dans la documentation on peut trouver la partie de code qu’on doit ajouter dans notre html :

*<!-- Captcha -->*

<form action="valideCaptcha" method="POST" id="captcha">

   <div class="g-recaptcha" data-callback="valideCaptcha" data-sitekey="6Ldb1RkbAAAAAEKSvT8NRY9BMPUPV5atQOXibb7I">

   </div>

Un des problèmes que j’ai rencontrés c’était que je n’arrivais pas à avoir le captcha qui fonctionnais sur ma page. C’est là que j’ai appris qu’il fallait ajouter les domaines dans le site de google, personnellement j’utilisais Visual studio code qui me permettais grâce à live serveur (qui est une extension) de travailler avec l’adresse local host. J’ai mis alors l’adresse « 127.0.0.1 » et aussi le domaine de mon site.

### Accueil

L’accueil est le cœur de notre page, c’est ici que la principale fonctionnalité va être réalisé, pour ce qui concerne la partie html j’ai repris sur le site <https://codepen.io/> quelques idées pour la partie visuel du site. J’ai alors pris et compris un bout de code qui comportais des cartes avec des images et une description par-dessous. J’ai également pris pour la search bar qui nous permet d’écrire quel héro on veut trouver des informations.

### CSS

Dans le css j’ai utilisé 2 pages différentes afin de mieux organiser le projet, il s’agit de « main.css » et « accueil.css ». On les importe les deux avec la link style et en mettant la source du ficher.

<link rel="stylesheet" href="../css/accueill.css">

<link rel="stylesheet" href="css/main.css">

C’est grâce à tous les éléments css que le site devient plus agréable à regarder et a utiliser. Ici je n’ai pas rencontré beaucoup de problèmes avec le css, j’ai repris des exemples sur codepen.io et j’ai fait quelques modifications sur ces exemples.

## Partie JS et les commentaires

### Index.js

Cette page est le cœur de l’application, c’est ici qu’on crée les différents constructeurs pour les autres fichiers js et ensuite les donnes les instructions de quoi il faut faire.

$().ready(function () {

  service = new Service();

  indexCtrl = new IndexCtrl();

  loginCtrl = new LoginCtrl();

  service.centraliserErreurHttp(indexCtrl.afficherErreurHttp);

*// indexCtrl.loadLogin();*

});

Ici on utilise le « ready »pour attendre que le document soit prêt pour commencer a créer les constructeurs, de plus on a encore l’appelle de la méthode centraliser Erreur qui nous permet d’identifier plusieurs erreurs.

C’est aussi a ce niveau qu’on demande au fichiers service de charger la première vue pour le login.

loadLogin(){

  service.chargerVue('login', () => new LoginCtrl());

}

### service.js

Avec cette page on a 2 méthodes, une qui s’appelle « centraliserErreurHttp » qui comme son nom l’indique nous permet de centraliser les erreurs http, ce qui rends plus facile de savoir de quel type d’erreur nous avons en cas problème.

centraliserErreurHttp(httpErrorCallbackFn) {

  $.ajaxSetup({

    error: function (xhr, exception) {

      let msg;

      if (xhr.status === 0) {

        msg = "Pas d'accès à la ressource serveur demandée !";

      } else if (xhr.status === 404) {

        msg = "Page demandée non trouvée [404] !";

      } else if (xhr.status === 500) {

        msg = "Erreur interne sur le serveur [500] !";

      } else if (exception === "parsererror") {

        msg = "Erreur de parcours dans le JSON !";

      } else if (exception === "timeout") {

        msg = "Erreur de délai dépassé [Time out] !";

      } else if (exception === "abort") {

        msg = "Requête Ajax stoppée !";

      } else {

        msg = "Erreur inconnue : \n" + xhr.responseText;

      }

      httpErrorCallbackFn(msg);

    },

  });

}

La deuxième c’est la méthode responsable pour le changement de vue dans notre application est le méthode « chargerVue ». On reçoit quel vue on veut charger et grâce à « .load » la vue sera chargé.

chargerVue(vue, callback) {

*// charger la vue demandee*

  $("#view").load("views/" + vue + ".html", function () {

*// si une fonction de callback est spécifiée, on l'appelle ici*

    if (typeof callback !== "undefined") {

      callback();

    }

   });

}

### login.js

Vu qu’on ne sera pas évaluez sur le PHP pour réaliser le login je n’ai pas réalisé de PHP. Avoir un base de données en MySQL serais la meilleurs des solutions sauf que moi j’ai tout simplement utilisé une méthode qui est appelé avec une condition qui vérifie si les informations correspondent au valeurs entrées.

validerUtilisateur() {

*// récupère les objets InputText du formulaire HTML*

  let user = document.getElementById("username").value;

  let psw = document.getElementById("password").value;

  console.log("le username est : " + user);

  console.log("le password est : " + psw);

  if (user.toLowerCase() === "admin" && psw === "admin" || user.toLowerCase() === "emf123" && psw === "emf123") {

      window.alert("Utilisateur validé");

      this.redirect();

   } else {

   window.alert("Vos identifications sont incorrects !!!\nVeuillez réessayer...")

  }

}

Si le login est validé alors on redirectionne a la page d’accueil.

redirect() {

  window.location.href = "../300222\_307\_Projet\_Lopes\_Tiago/views/accueil.html"

}

### requete.js

C’est grâce à ce fichier que nous pouvons faire la requête sur le serveur Marvel afin de retourner les informations qu’on désire. Cela ce fait au moyen de la méthode « requete ».

function requete() {

*//mise a 0 des cards*

    let herosSection = document.querySelector(".wrapper");

    herosSection.innerHTML = "";

*//Valeur dans la searche box*

    let input = document.getElementById("searchBarInput").value;

    fetch(baseUrl + input + "&ts=1&apikey=ceb9db9f7dab22d3e21c5f48ce074034&hash=ddd431394cf1912ff23a7cd59be524fd").then(function (response) {

        if (response.status !== 200) {

            console.log("Looks like there was a problem. Status Code: " + response.status);

            return;

        }

        response.json().then(function (data) {

            afficheHeros(data);

        });

    }).catch(function (err) { console.log("Fetch Error :-S", err); });

}

Pour qu’on puisse faire des requêtes vers le server Marvel il nous demande la chose suivante, un hashmd5 avec le timestamp, le clé privé et aussi la clé publique, le résultat de cet hash vient être mis a la fin de l’url.

Si on regarde étape par étape, tout s’abords con commence par remettre a zéro tous les cartes qu’on a créées si jamais on a fait une autre requête avant, ensuite on garde la chaine de caractères de notre barre de recherche. Après on fait notre requête avec toutes les info nécessaires et on les envois a la prochaine méthode qui va organiser la réponse de la requête.

function afficheHeros(data) {

    let herosArray = data.data.results;

    for (let hero of herosArray) {

        let heroName = hero.name;

        let heroDescription = hero.description;

        let heroImage = hero.thumbnail.path + "." + hero.thumbnail.extension;

        document.createElement('div');

        let heroCard = `<div class='card'><img src="${heroImage}"/><div class='info'><h1>${heroName}</h1><p>${heroDescription}</p></div></div>`;

        document.querySelector(".wrapper").innerHTML += heroCard;

    }

}

Ici on créer une arraylist avec les informations reçues, et on parcourt cette liste pour chaque héro. Pour chaque héro alors on prend son nom, ça description et son image. Ensuite on crée notre div card avec les informations qu’on veut renvoyer.

# Tests de l'application

## Client fonctions internes

|  |  |  |
| --- | --- | --- |
| Test | Réponse | Résultat |
| Lien de l’html et le css |  | Les fichiers sont chargées, on peut voir que le fichier css applique ces styles |
| Lien de l’html et les javascripts |  | Les fichiers sont chargées car les vues sont chargées, cela veut dire que les différents contrôleurs sont appelés |

## Client fonctions extérieures

|  |  |  |
| --- | --- | --- |
| Test | Réponse | Résultat |
| Appelle du service Web |  | La requête vers l’api de Marvel fonctionne et retournes toutes les informations désirés |
| Appelle du service reCAPTCHA |  | Le captcha apparait a la page de login et est fonctionnel |

# Hébergement et installation du projet complet

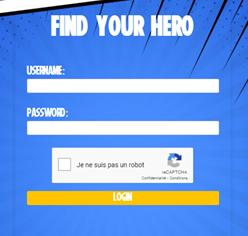
Personnellement j’utilise core FTP pour l’hébergement de mon site, mais il y existe d’autres alternative comme Cpanel par exemple. Mon projet se trouve sur :   
<http://lopesmonteirom.emf-informatique.ch/307/300222_307_Projet_Lopes_Tiago/>

Pour se connecter on utilise le login suivant :

|  |  |
| --- | --- |
| Username | Password |
| admin | admin |
| emf123 | emf123 |

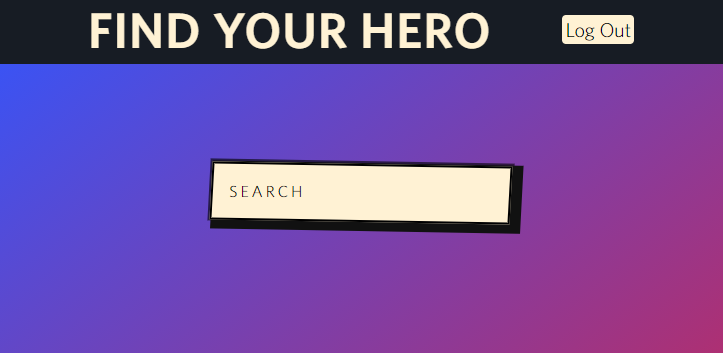
# Fonctionnement du projet hébergé

Voici comment fonctionne mon site, pour commencer le visiteur se rends directement dans une page de login, c’est-à-dire celle de base dans le site et se connecte avec les logins donnés (voir [point 6](#_Exercice_6_–) pour avoir le lien et les logins). Ensuite il faut valider le captcha, si celui-ci n’est pas validé on ne peut pas se loguer, il faut alors le faire avant d’appuyer sur le bouton login.

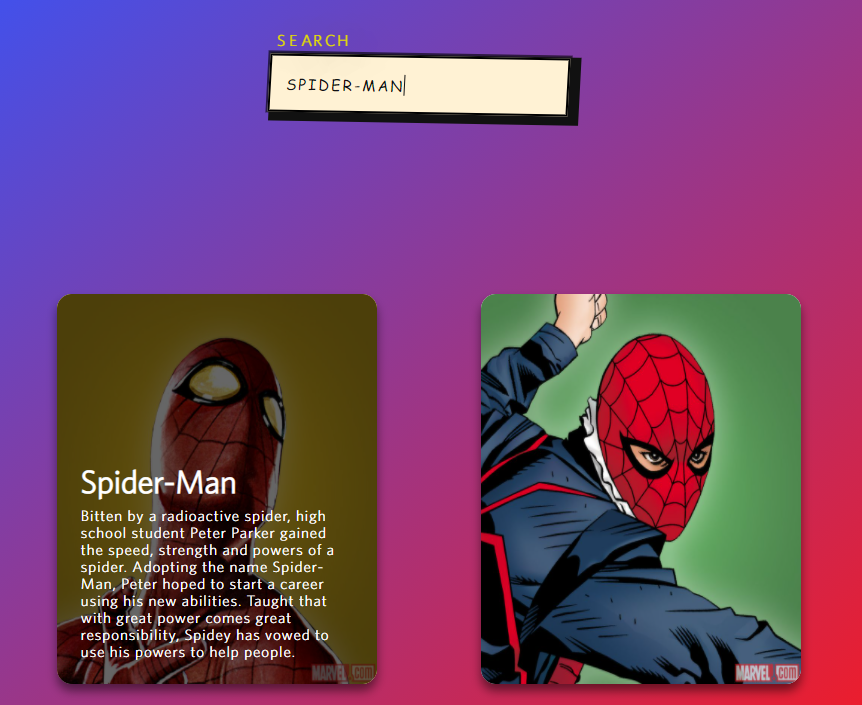


L’idée de base était aussi d’avoir une autre view pour créer des utilisateurs mais cela ne serait pas possible car il nous faudrait avoir une base de données.

Une fois connecté l’utilisateur est renvoyé à la page « accueil » et est confronté avec une barre de recherche qui sera la façon de choisir quels personnages on veut afficher. On retrouve aussi un bouton « log out » si on veut revenir à la page d’accueil.



En écrivant sur la barre de recherche un nom de héro par exemple « spider » on peut retrouver tous les personnages qui contiennent « spider » dans leurs noms. En passant la souris par-dessus les cartes on peut retrouver pour certains des personnages une description. A chaque nouveau changement réalisé dans la barre de recherche les cartes sont mises à jour avec les résultats.

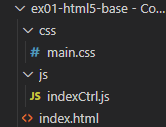


# Auto-évaluation et conclusion

# Tests technologiques selon les exercices

## Exercice 1 - base HTML5

Le but de cet exercice était de se rappeler la structure qu’on doit utiliser lorsqu’on code notre page Web. Structure de l’exercice :



Il est important de bien structuré notre projet afin de bien nous structurer et surtout ne pas mélanger les différents langages par exemples ne pas mettre de CSS dans notre page HTML. Pour ce qui est des points les plus importants il faut faire attention dans cet exercice nous avons :

Dans la première manière de faire l’écouteur se fessait au niveau de la méthode javascript.

<body onload="initCtrl()">

Dans la deuxième façon il fallait ajouter l’écouteur au niveau du html :

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

<http://lopesmonteirom.emf-informatique.ch/307/ex01-html5-base/>

## Exercice 2 - login

Le point important à retenir dans cet exercice c’est comment reprendre des valeurs en HTML, les utiliser en JS et vice versa. Pour ça on utilise l’attribut, écrire dans la console en document.getElementById("username") pour reprendre une valeur du DOM.

let userComp = document.getElementById("username");

Nous avons aussi vu le console.log("") qui nous permet d’afficher quelque chose dans la console du navigateur.

console.log("utilisateur:" + username + " mot de passe: " + password);

Pour conclure on a utilisé window.alert() qui nous permet d’afficher quelque chose dans une pop-up.

window.alert("Utilisateur ou mot de passe incorecte !");

Ensuite on a regardé que voulais dire les 2 attributs « placeholder» et « autofocus». Pour ce qu’il est du placeholder il nous permet de définir un bref indice qui décrit la valeur attendue d'un champ d'entrée. Le autofocus est un attribut booléen qui lorsqu'il est présent, il spécifie qu'un <input> élément doit automatiquement obtenir le focus lors du chargement de la page.

<http://lopesmonteirom.emf-informatique.ch/307/ex02-html5-form-login/>

## Exercice 3 - login avec PHP

Ici j’ai rencontré plusieurs problèmes, dans un premier temps on devait adapter le CSS comme demandé à la consigne. Pour ce qui est de la partie PHP ici le point clé était que pour avoir notre PHP qui contienne il faut avant tout que notre projet soit hébergé via « FTP côre » par exemple car le PHP fait partie du coté serveur. Il est aussi important de savoir la différence entre le GET et le POST, le GET nous donne des informations en clair dans l’url ce qui veut dire, pas sécurisé. Alors que le POST nous crypte ces informations.

Voici le code donné et expliqué :

<?php

*// test si on a reçu une donnée de formulaire nommée "username"*

if (isset($\_POST['username'])) {

*// récupération des données transmises dans des variables locales*

  $username = strtolower($\_POST['username']);

  $password = $\_POST['password'];

*// affichage des infos reçues*

  echo "username: ".$username."</br>";

  echo "password: ".$password."</br>";

*// test username et mot de passe*

  if (($username == "admin") && ($password == "emf123")) {

    echo "<script>alert('Validation OK');</script>";

  } else {

    echo "<script>".

            "alert('Utilisateur ou mot de passe incorrect !!!');".

         "</script>";

  }

}

?>

Pour que ce code puisse fonctionner il nous faut changer l’attribut input dans notre HTML en ajoutant name="…", sinon notre code ne fonctionnera pas. Il est aussi important de se rappeler que si les chemins ne sont pas corrects alors notre page ne fonctionnera pas.

Encore un élément important ce sont les différents types de boutons. Il y existe en quatre versions :

* type="submit": le principal bouton d'envoi de formulaire. C'est celui que vous utiliserez le plus souvent. Le visiteur sera conduit à la page indiquée dans l'attribut action du formulaire ;
* type="reset": remise à zéro du formulaire ;
* type="image": équivalent du bouton submit , présenté cette fois sous forme d'image. Rajoutez l'attribut src pour indiquer l'URL de l'image ;
* type="button": bouton générique, qui n'aura (par défaut) aucun effet. En général, ce bouton est géré en JavaScript pour exécuter des actions sur la page. Nous ne l'utiliserons pas ici.

Ici nous avons donc utilisé submit pour envoyer notre formulaire. <https://openclassrooms.com/fr/courses/1603881-apprenez-a-creer-votre-site-web-avec-html5-et-css3/1607171-creez-des-formulaires>

<http://lopesmonteirom.emf-informatique.ch/307/ex03-html5-form-php/>

## Exercice 4 – utilisation de jsfiddle

Dans cet exercice nous avons dû utiliser un site intitulé <https://jsfiddle.net/> , qui nous permettais de tester du code JavaScript à partir de code HTML + CSS. Pour que le code puisse fonctionner il faut aller sur « Ressources » et coller ce lien : <https://rawgit.com/eu81273/jsfiddle-console/master/console.js>

## Exercice 5 – variables en JavaScript

Avec cet exercice nous avons appris les différentes façons de déclarer une variable avec JavaScript, il y a ces 4 possibilités :

myVar;

var myVar;

let myVar;

const myVar=1;

Le meilleur a utilisé est le mot-clé « let », la première et la deuxième façon de faire sont trop libres (possibilité de déclarer 2x la même variable). Ensuite nous avons dû faire différents tests sur la console d’un navigateur, voici les bouts de code les plus intéressants :

console.log(`${a} + ${b} = ${a + b}`);

console.log(`${a.toLowerCase()} ${b}`);

console.log(`${a} AND ${b} = ${a && b}`);

console.log(`${a} OR ${b} = ${a || b}`);

let date = new Date();

date.setDate(date.getDate() - 61);

console.log(date.toLocaleString());

console.log(`${a} est de type ${typeof a}`);

## Exercice 6 – la structure de contrôle SWITCH

Dans cet exercice on découvre comment fonctionne un switch en java script. Il est identique en java dans sa formulation. On a repris la base de l’exercice 1 mais au lieu d’afficher notre nom on a fait un switch case pour afficher les jours de la semaine. Voici comment j’ai fait pour le switch :

switch (day) {

    case 0:

      jourSemaine = "dimanche";

      break;

    case 1:

      jourSemaine = "lundi";

      break;

    case 2:

      jourSemaine = "mardi";

      break;

    case 3:

      jourSemaine = "mercredi";

      break;

    case 4:

      jourSemaine = "jeudi";

      break;

    case 5:

      jourSemaine = "vendredi";

      break;

    case 6:

      jourSemaine = "samedi";

      break;

  }

<http://lopesmonteirom.emf-informatique.ch/307/ex06-js-switch-test/>

## Exercice 7 – les boucles en JavaScript

Dans l’exercice nous avons appris le fonctionnement des boucles en JavaScript. L’idée reste la même que dans les autres langages de programmation, nous avons 3 types de boucle, for, while et do while. Voici un exemple de leur utilisation, ici ou voulais à chaque fois afficher le nom de la boucle avec toutes les itérations :

function testerFor() {

  let valeur = "for (let i=0; i<5; i++){...}<br>";

  for (let i = 0; i < 5; i++) {

    valeur += "i = " + i + "<br>";

  }

  valeur += "À utiliser si on sait que l'on veut itérer x fois (x connu avant de commencer la boucle)";

  document.getElementById("info").innerHTML = valeur

}

function testerWhile() {

  let valeur = "while (i<5){...}<br>";

  let i = 0;

  while (i < 5) {

    valeur += "i = " + i + "<br>";

    i++;

  }

  valeur += "À utiliser si on ne sait pas le nombre d'itérations au démarrage de la boucle";

  document.getElementById("info").innerHTML = valeur

}

function testerDoWhile() {

  let valeur = "do {...} while(i<5);<br>";

  let i = 0;

  while (i < 5) {

    valeur += "i = " + i + "<br>";

    i++;

  }

  valeur += "À utiliser si on ne sait pas le nombre d'itérations au démarrage de la boucle mais avec un passage obligatoire";

  document.getElementById("info").innerHTML = valeur

}

<http://lopesmonteirom.emf-informatique.ch/307/ex07-js-loopes-test/>

## Exercice 8 – tableau associatif ou tableau d’objets JSON

Dans cet exercice nous avons vu comment utiliser des objets JSON afin dès les lire depuis un tableau. La structure dans l’idée reste simple mais il y a quelques points de syntaxe auquel il faut faire attention, ce site contient des explications supplémentaires sur le JSON : <http://blog.idleman.fr/le-json-un-pont-entre-tous-les-langages/>

Dans l’exercice, le but étant de parcourir un tableau JSON, j’ai utilisé une boucle for qui va me permettre de parcourir chaque élément.

<http://lopesmonteirom.emf-informatique.ch/307/ex08-js-json-array-loop/>

### Déclaration du tableau

Ici on déclare un nouveau tableau de personnes avec 4 personnes a l’intérieur.

const json = {

    personnes: [

      { prenom: "John", nom: "Doe", age: 44 },

      { prenom: "Anna", nom: "Smith", age: 32 },

      { prenom: "Peter", nom: "Jones", age: 29 }

    ]

  };

### Parcours du tableau

On parcourt le tableau pour ensuite retourner les informations

function parcourirUnTableauJSON() {

    let personne;

    let valeur = "";

    for (let i = 0; i < json.personnes.length; i++) {

      personne = json.personnes[i];

      valeur += i + ". " + personne.prenom + " " + personne.nom + " " + personne.age + "<br>";

    }

    document.getElementById("info").innerHTML = valeur

}

## Exercice 9 – démo sur les objets en JavaScript

L’exercice 9 s’agit d’une démonstration de l’utilisation des objets en JavaScript, on va voir 5 façons de les utiliser.

<http://lopesmonteirom.emf-informatique.ch/307/ex09-js-poo-demo-objet/>

### JSON

La première consiste à utiliser JSON pour créer notre objet on peut mettre les attributs qu’on souhaite à l’intérieur et leurs donner des valeurs, ici par exemple on a créé une personne qui a comme attributs un nom un prénom et un Age. On peut donc le créer de cette façon :

*// créer un objet en JSON*

 let p1 = {

    prenom: "Jules",

    nom: "Tartampion",

    age: 25,

  };

### Objet

La deuxième méthode de faire c’est en utilisant un Objet, on peut lui donner également des attributs mais cette fois c’est en appelant l’objet. Voici un exemple :

*// créer un objet avec la fonction Object*

  const p1 = new Object();

  p1.prenom = "Juliette";

  p1.nom = "Tartampion";

  p1.age = 23;

### Object fonction

Il y existe aussi encore une autre méthode c’est avec un Object fonction. Voici un exemple d’utilisation :

function Personne(prenom, nom, age) {

  this.prenom = prenom;

  this.nom = nom;

  this.age = age;

  this.toString = function () {

    return this.prenom + " " + this.nom + " (" + this.age + ")";

  };

}

### Fonction prototype

Une autre manière c’est en utilisant une fonction prototype, par exemple :

function Professeur(prenom, nom, age) {

  this.prenom = prenom;

  this.nom = nom;

  this.age = age;

}

Professeur.prototype.toString = function () {

  return this.prenom + " " + this.nom + " (" + this.age + ")";

};

### Classe

La dernière façon de faire consiste à utiliser une classe pour créer un objet. Cette méthode est très utilisée en Java et consiste à utiliser un constructeur pour qu’à chaque fois qu’un objet soit créé il faut seulement que les paramètres soient remplis. Example:

class Eleve {

  constructor(prenom, nom, age) {

    this.prenom = prenom;

    this.nom = nom;

    this.age = age;

  }

## Exercice 10 – programmation orientée *« objets »* en JS

L’exercice 10 nous a permis d’utiliser les fonctions privées, une fonction est privée lorsqu’elle a un « \_ » avant le nom de la fonction. Une autre notion qui a été utilisé est le démarrage de la page à la fin du chargement et pour ça on utilise :

document.onreadystatechange = function () {

    if (document.readyState === "complete") {

      \_afficherPersonnes();

    }

  };

<http://lopesmonteirom.emf-informatique.ch/307/ex10-js-poo/>

## Exercice 11 – programmation orientée *« classe »* en JS

Dans l’exercice 11 nous avons appris à comment utiliser une classe en JavaScript. Pour voir comment on déclare une classe cliquez [ici](#_Classe). Le plus important avec les classes en JS, c’est qu’il faut toujours utiliser « this.objet » quand on fait référence à quelque chose de la même classe. Par example:

class Personne {

    constructor(prenom, nom, age){

        this.prenom = prenom ;

        this.nom = nom;

        this.age = age;

    };

    toString() {

        return this.nom + " " + this.prenom + " (" + this.age + ")";

    };

}

On a aussi vu comment utiliser un ctrl dans notre page. Pour instancier la classe Ctrl, il faut la stocker dans l’objet Windows de JS. Ainsi, l’html pourra y accéder :

document.onreadystatechange = function () {

  if (document.readyState === "complete") {

    window.ctrl = new Ctrl();

    ctrl.\_afficherPersonnes();

  }

};

<http://lopesmonteirom.emf-informatique.ch/307/ex11/>

## Exercice 12 – les closures – compteur

Dans cet exercice on a appris la notion des closures. Une closure c’est une fermeture qui donne accès à la portée d'une fonction externe à partir d'une fonction interne (on dit aussi que la fonction « capture son environnement »). Dans l’exercice nous avons dû faire une closure pour un compteur de cliques :

function c() {

  let compteur = 1

  return function () {

    return compteur++;

  }

}

Ici on peut voir qu’on a une fonction interner dans la fonction « c() » ce qui nous indique que c’est une closure.

<http://lopesmonteirom.emf-informatique.ch/307/ex12-js-closure/>

## Exercice 13 – les IIFE en JavaScript

Dans cet exercice on voit le concepte d’une IIFE, « Immediately-Invoked Function Expression », qui nous permet d'exécuter une fonction juste après sa création. Voici un exemple:

(function () {

  Let myVal = 2;

  Console.log(myVal);

})()

Et voici la syntaxe avec des arguments :

(function (a) {

  Let myVal = 2;

  Console.log(myVal + a);

})(2)

<http://lopesmonteirom.emf-informatique.ch/307/ex13-js-poo-class-iife/>

## Exercice 14 - bases de jQuery

Dans cet exercice nous avons à disposition un site avec les bases et quelques exemples de l’utilisations de JQuery. Voici le site <https://www.jcsinfo.ch/demo/jquery/> . En résument ce que dit le site JQuery est utilisé car c’est un langage qui est bien documenté, qui a une grande communauté et qui est compatible avec la plupart de navigateur.

$("div").addClass("special");

* $ est l'objet JQuery (aussi nommé jQuery).
* C'est en réalité une fonction JavaScript !
* Le "div" --> la recherche se fait à l'aide d'un sélecteur CSS.
* $("div") retourne un set jQuery de 0 à N noeuds du DOM.
* addClass(...) ajoute une classe HTML à tous les noeuds trouvés (définit du CSS sous-jacent).

Il y a aussi quelques évènements qui sont intéressants comme le « ready »

$(document).ready(function(){

*// Votre code jQuery doit se trouver ici*

});

* Dans le but de traverser et de manipuler une page HTML, nous devons attendre jusqu'à ce que celle-ci soit prête (ready) pour être utilisée.
* jQuery contient un événement “ready” qui part à l'instant même où le DOM est prêt et que l'on peut travailler avec (même si des images sont encore en train de se charger).
* Mettez votre code jQuery dans un bloc “ready” et le chargement du code se passera sans blocage.

Pour plus d’informations le site de W3Schools est très complet et nous permet de revoir tout ce qu’on peut faire avec JQuery. <https://www.w3schools.com/jquery/default.asp>

## Exercice 15 – première utilisation de jQuery

Dans cet exercice il faut réaliser une même action avec JS et JQ, ici on un a menu déroulant qui nous permet de choisir une couleur qui sera changé une fois cliquer.

<http://lopesmonteirom.emf-informatique.ch/307/ex15-jquery-premiers-pas/>

### JavaScript

En java script je suis arrivé au résultat le suivant :

(() => {

  document.onreadystatechange = function () {

    if (document.readyState === "complete") {

      window.ctrl = new Ctrl();

      let couleur = document.getElementById("couleurs").value;

*//sélection de la couleur dans la liste*

      console.log(couleur);

      ctrl.changerCouleur(couleur);*//appel de la méthode dans le ctrl*

Dans un premier lieu on vérifiait si on pouvait lancer la page avec le « readyState ». Si c’est validé alors on crée notre ctrl. Ensuite ont stock la valeur de notre menu dans un variable « couleur » et on appel notre méthode pour changer de couleur dans le ctrl « ctrl.changerCouleur(couleur) ».

Dans notre Ctrl on a la méthode changerCouleur qui sera responsable par le changement de couleur.

  changerCouleur(*couleur*) {

    let conteneur = document.getElementById("container");

*//crée une variable avec la valeur du div "container"*

    conteneur.style.background = *couleur*;

*//change la couleur selon la valeur dans le paramètre*

*//changement de la couleur du texte selon ces 3 couleurs*

    if (*couleur* === "red" || *couleur* === "blue" || *couleur* === "green") {

      conteneur.style.color = "white";

    } else {

      conteneur.style.color = "black";

    }

  }

### JQuery

En JQuery pour ce qui est de la structure on va rester avec la même, on utilise le ctrl pour appeler une méthode qui change la couleur. La différence est comment on déclare la fonction et on vérifie si notre page peut être charger :

$().ready(function () {

}

Pour la méthode de changer couleur la différence est la manière donc on fait les changements visuels.

  changerCouleur(*couleur*) {

    $("#container").fadeOut(1500, function () {*/\*effect du fade out, premièr paramètre la duré du fade,*

*deuxiéme la function exécuté par apres.\*/*

      $("#container").css("background-color", *couleur*);*//changement de la couleur le "css" nous permet d'utiliser des*

      if (*couleur* === "red" || *couleur* === "blue" || *couleur* === "green") {

        $("#container").css("color", "white");

      } else {

        $("#container").css("color", "black");

      }

*//$("#container").slideToggle(1000);*

      $("#container").fadeIn(1000);

    });

  }

On veut voir qu’on change tout suite la partie IHM de notre page, de plus le résultat est beaucoup plus charmant qu’on utilisant seulement que JS.

## Exercice 16 – lecture jQuery & Bootstrap

L’exercice 16 était un exercice facultatif ici on voulait apprendre l’existence des Framework qui peuvent être très puissants. L’exemple que l’enseignant nous avais donné il s’agissait d’un Framework qui va gérer la taille des écrans dans un environnement fragmenté entre les desktops, les laptops, les tablettes et les mobiles de toutes tailles Bootstrap va nous aider à faire des sites "responsive" qui signifient que l'organisation des pages va changer en fonction de la taille de l'écran. Ce Framework nous propose de nouveaux composants ainsi que des validations évoluées de formulaire. Il est la propriété de Twitter.

Voici des liens mis à disposition pour nous documenter :

<https://www.pierre-giraud.com/bootstrap-apprendre-cours>

## Exercice 17 – test de web services

Dans cette exercice nous avons découvert ce que c’était une API, une API c’est une interface utilisateur qui est destinée à être utilisée par d’autres programmes, contrairement aux interfaces traditionnelles qui sont destinées à un End user.

Ces dernières sont mises à disposition par des entreprises parfois gratuitement, parfois contre de doux deniers. Elles permettent à un site d’accéder aux données d’autres services. Par exemple, l’API Transport permet d’accéder aux horaires des transports publics suisses. On pourrait donc grâce à cette API reproduire le fonctionnement site des CFF.

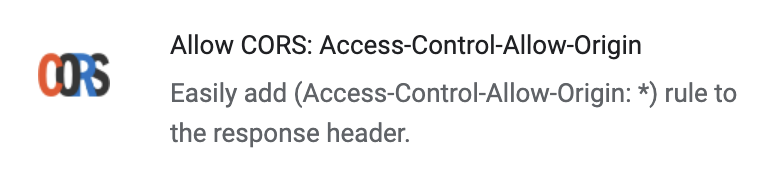
### CORS

Tout ceci est génial, mais cela pose quelques problèmes de sécurité. En effet, côté serveur, si on laisse n’importe-qui utiliser nos données en ayant juste une url, ce n’est pas très sécurisé. C’est pourquoi, par défaut, PhP et les principaux browsers bloquent l’accès depuis un domaine autre que le leur. Pour créer ou accéder à des APIs, il faudra donc changer ce paramètre.

Côté serveur (PhP), il faut modifier l’entête du fichier php :

header("Access-Control-Allow-Origin: \*");

Côté client, on peut installer l’extension Allow CORS qui permet de désactiver cette règle :



## Exercice 18 – application HTML5 d’appel à un web service

Dans cet exercice nous avions dû créer une page qui va transformer les degrés C° en F°. Une bonne partie du code était déjà donnée, ici on voulait surtout voir la notion de l’appel à un service web. Pour que cela puisse fonctionner il faut changer l’url du php, il est important de faire attention orthographe car c’est une éventuelle source de problèmes.

  celcius2Fahrenheit(degres, successCallback) {

    let url = "http://lopesmonteirom.emf-informatique.ch/307/ex18-ws-simple/php/convert\_temp\_p\_xml.php";

    let param = "Temperature=" + degres + "&FromUnit=C&ToUnit=F";

Si les changements ne sont pas faits il faut supprimer le cache du navigateur et éventuellement tester en 4G ou avec un autre réseau.

On a aussi ajouté quelques conditions afin de mieux identifier les éventuels problèmes qu’on peut avoir. Ici on peut gérer toutes ces erreurs, pas d’accès au serveur demandé, erreur dans le parcours dans le JSON, etc…

  centraliserErreurHttp(httpErrorCallbackFn) {

    $.ajaxSetup({

      error: function (xhr, exception) {

        let msg;

        if (xhr.status === 0) {

          msg = "Pas d'accès à la ressource serveur demandée !";

        } else if (xhr.status === 404) {

          msg = "Page demandée non trouvée [404] !";

        } else if (xhr.status === 500) {

          msg = "Erreur interne sur le serveur [500] !";

        } else if (exception === 'parsererror') {

          msg = "Erreur de parcours dans le JSON !";

        } else if (exception === 'timeout') {

          msg = "Erreur de délai dépassé [Time out] !";

        } else if (exception === 'abort') {

          msg = "Requête Ajax stoppée !";

        } else {

          msg = "Erreur inconnue : \n" + xhr.responseText;

        }

        httpErrorCallbackFn(msg);

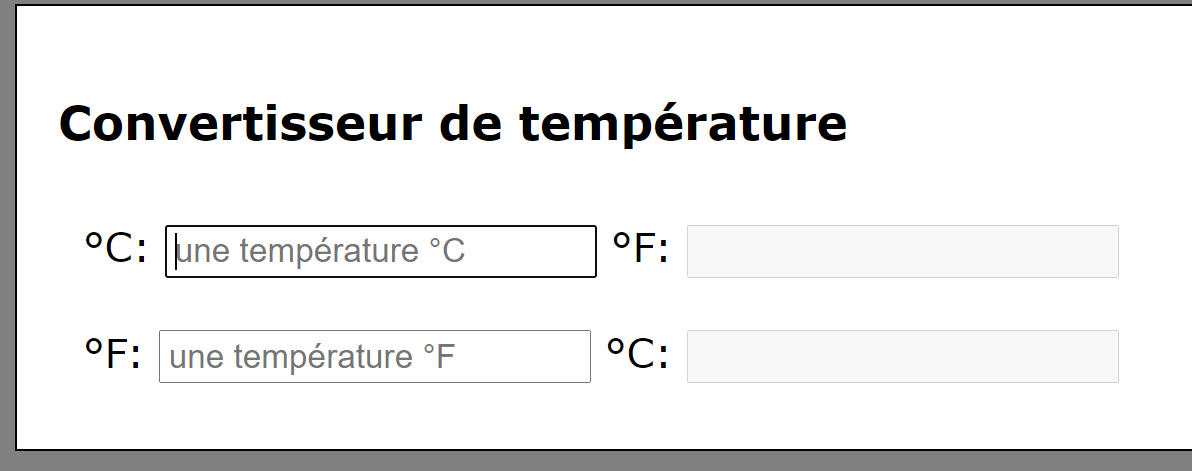
      }

    });

  }

<http://lopesmonteirom.emf-informatique.ch/307/ex18-ws-simple/>

## Exercice 19 – application HTML5 avec double service web

Cet exercice était identique à celui d’avant dans l’idée sauf que cette fois au lieu d’avoir un seul service web on en a deux. Ici on fait une conversion de C° en F° comme avant mais aussi de F° en C°. L’idées de cet exercice était donc de copier le html et le javascript afin d’avoir un résultat comme ceci :  


Ensuite la différence pour ce qui est du PHP c’est que cette fois on utilise du GET aux lieux du POST. Et pour ce qui est du JavaScript il faut copier le html des 2 méthodes dans ctrl et la méthode méthode httpserv.

<http://lopesmonteirom.emf-informatique.ch/307/ex19-ws-double-jquery/>

## Exercice 20 – application *« login »* sur un serveur PHP

Dans cet exercice nous avion comme objectif de créer une page web qui nous permet de gérer un login avec des utilisateurs et de créer des utilisateurs. La structure de cet exercice nous sera également utile pour notre projet, ici on a plusieurs views qui sont appelés selon ce qu’on veut par exemple pour créer un compte ou pour le login, cela fait en sorte que l’url ne change pas selon la vue. Les points les plus importants dans cet exercice étaient de bien s’avoir ce qu’on appelle et quand. IL fallait bien vérifier que les constructeurs soient toujours créés. Les parties les plus importantes dans ces exercices sont :  
l’appelle des views :

  loadLogin() {

    service.chargerVue("login", () => new LoginCtrl());

  }

  loadAccueil() {

    service.chargerVue("accueil", () => new AccueilCtrl());

  }

  loadCompte() {

    service.chargerVue("compte", () => new CompteCtrl());

  }

}

La création des views au début du chargement :

$(document).ready(function () {

  service = new Service();

  indexCtrl = new IndexCtrl();

  loginCtrl = new LoginCtrl();

  compteCtrl = new CompteCtrl();

  accueilCtrl = new AccueilCtrl();

  service.centraliserErreurHttp(indexCtrl.afficherErreurHttp);

  indexCtrl.loadLogin();

});

<http://lopesmonteirom.emf-informatique.ch/307/ex20-ws-login/>

# Conclusion

J’ai beaucoup apprécié ce module, je trouve que réaliser un projet depuis le zéro a été une très bonne expérience pour moi car cela m’a appris diverses choses. Je trouve que l’enseignant a su me motiver pendant tout le module et je trouve cela excellent comme démarche. J’ai appris pleins de nouveaux concepts et technologies que je n’avais jamais utilisé avant.