Programmation Web – Avancé

BINV2150 A : JavaScript (& JAVA SERVLETS)

Week 6

R. Baroni / J.L. Collinet / C. Damas

0

Table des matières

Tous les sujets traités pendant ce cours...

Table des matières

Table des matières

4

Introduction aux communications (synchrones) client / serveur

Découvrons le backend Java...

Table des matières :
4. Introduction aux communications (synchrones) client / serveur

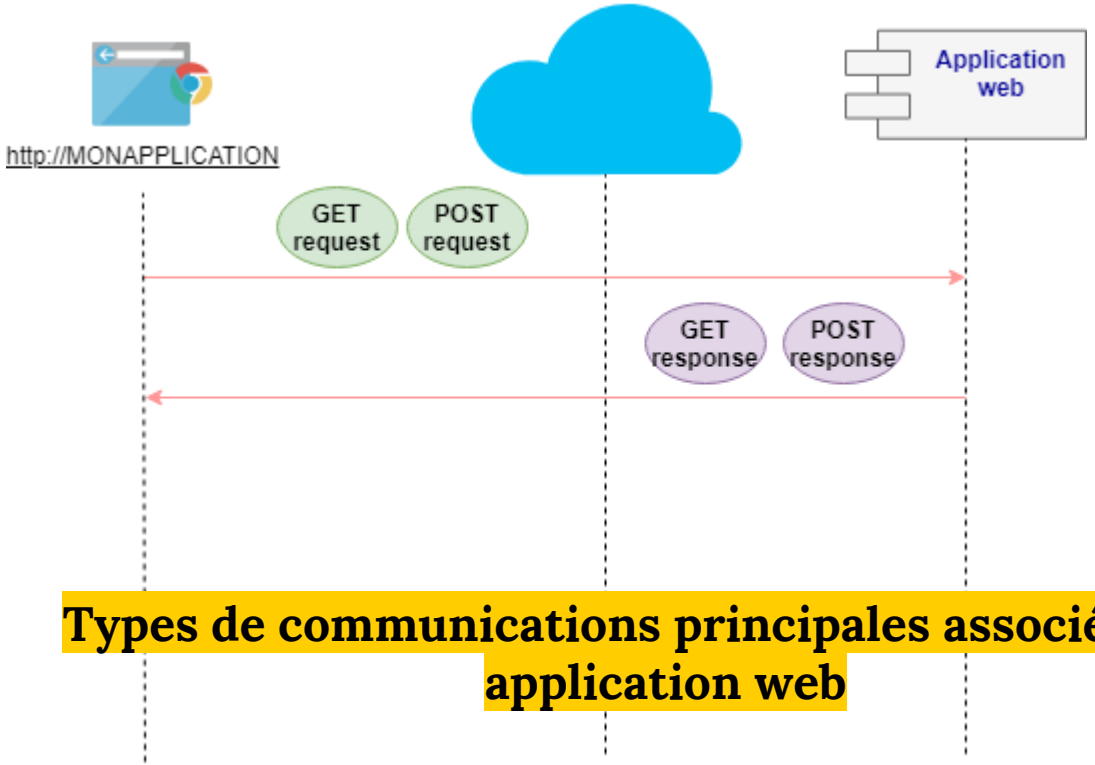
1. Quels types de communications ?
2. Quelles technologies ?
3. Introduction à un backend Java (Jetty)
4. Communications synchrones client /serveur
5. Introduction à la gestion de sessions côté serveur (Jetty)

Table des matières :

4. Introduction aux communications (synchrones) client / serveur

6. Introduction à JSON
7. Introduction à JSON côté Java : Genson

Quels types de communications ?



- Communications synchrones ou asynchrones ?
- Quelles types de données ?

Types de communications principales associés à une application web

Quels types de communications ?

- Basics of HTTP [7]: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview
- Requêtes :
  - Start line (Méthode HTTP GET / PUT / ..., URL
  - Headers (host, cookies...)
  - Body (HTML form parameters, JSON...)

Quels types de communications ?

- Réponses :
  - Status line (status code & text…)
  - Header (content type, Location, cookies…)
  - Body (ressource demandée, message d'erreur, JSON…)

Quels types de communications ?
Méthodes HTTP principales

- GET : obtenir une ressource.
  - L'URL entrée dans le navigateur effectue un GET sur cette adresse.
  - Les paramètres sont encodés dans cet URL :

```
?param1=valeur1&param2=valeur2
```

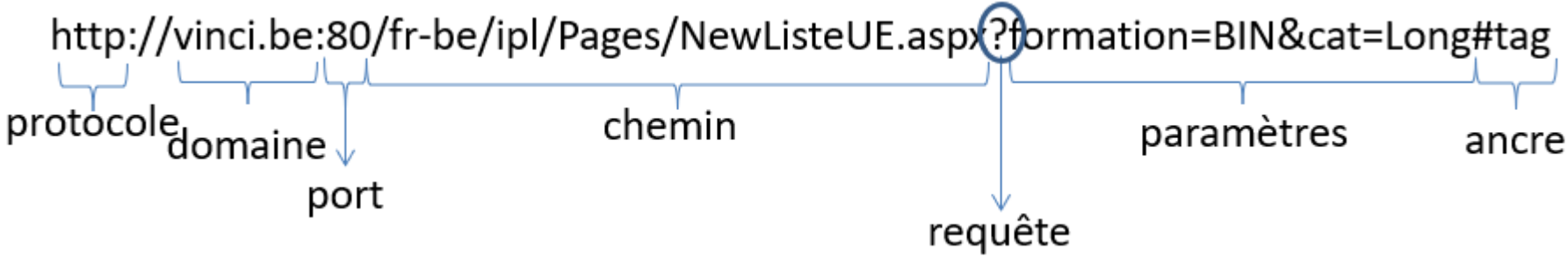
  - Bookmarkable.
  - En général peut être mis en cache.

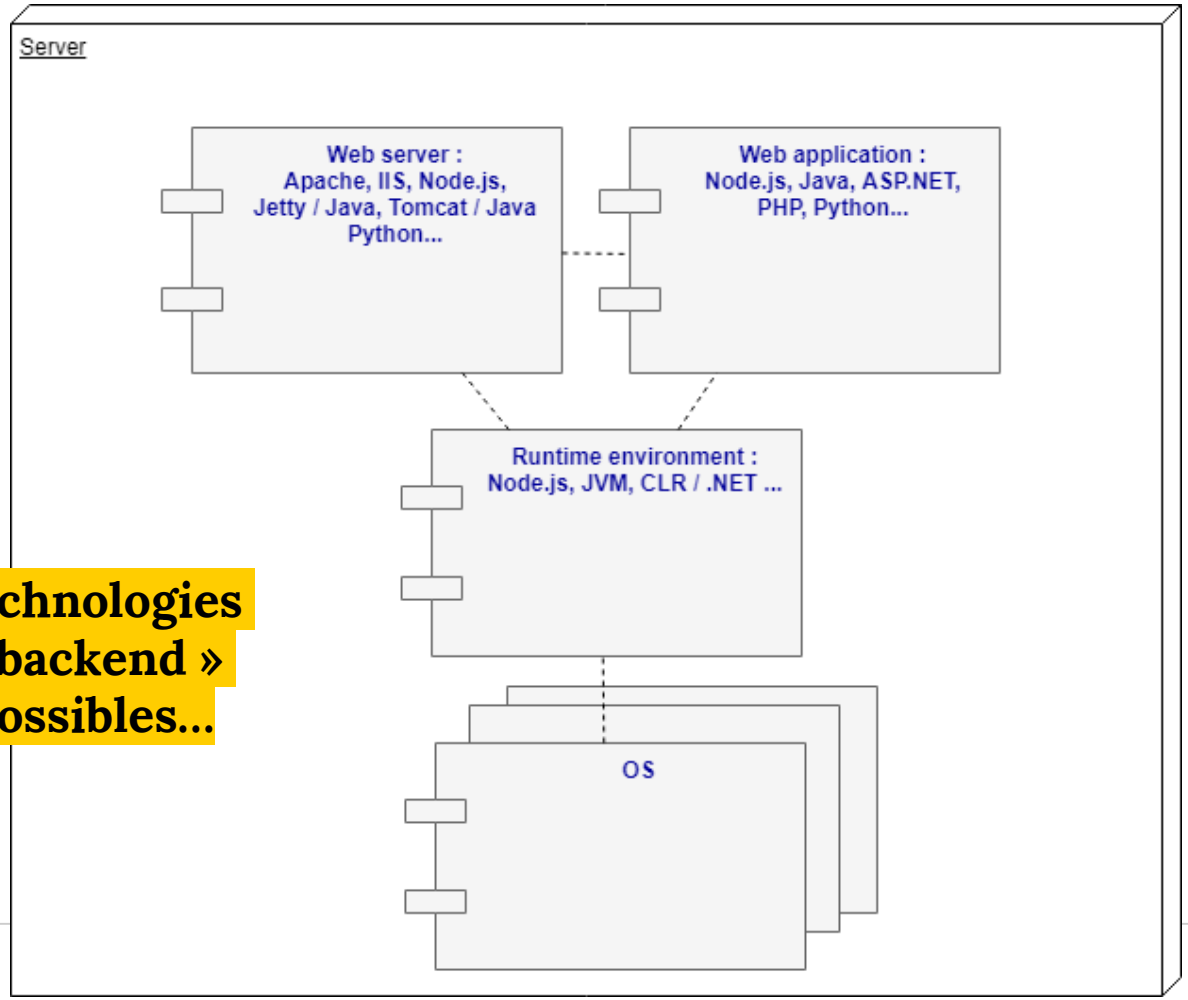
Quels types de communications ?
Méthodes HTTP principales

- POST : soumettre une ressource.
  - Soumission d'un formulaire : les données sont envoyées en POST.
  - Les paramètres sont encodés dans le 'payload' qui est distinct de l'URL.
  - Non bookmarkable.
  - En général ne peut pas être mis en cache : la réponse à la soumission d'un formulaire dépend de ce formulaire et change donc à chaque fois.

Quels types de communications ?
Méthodes HTTP principales

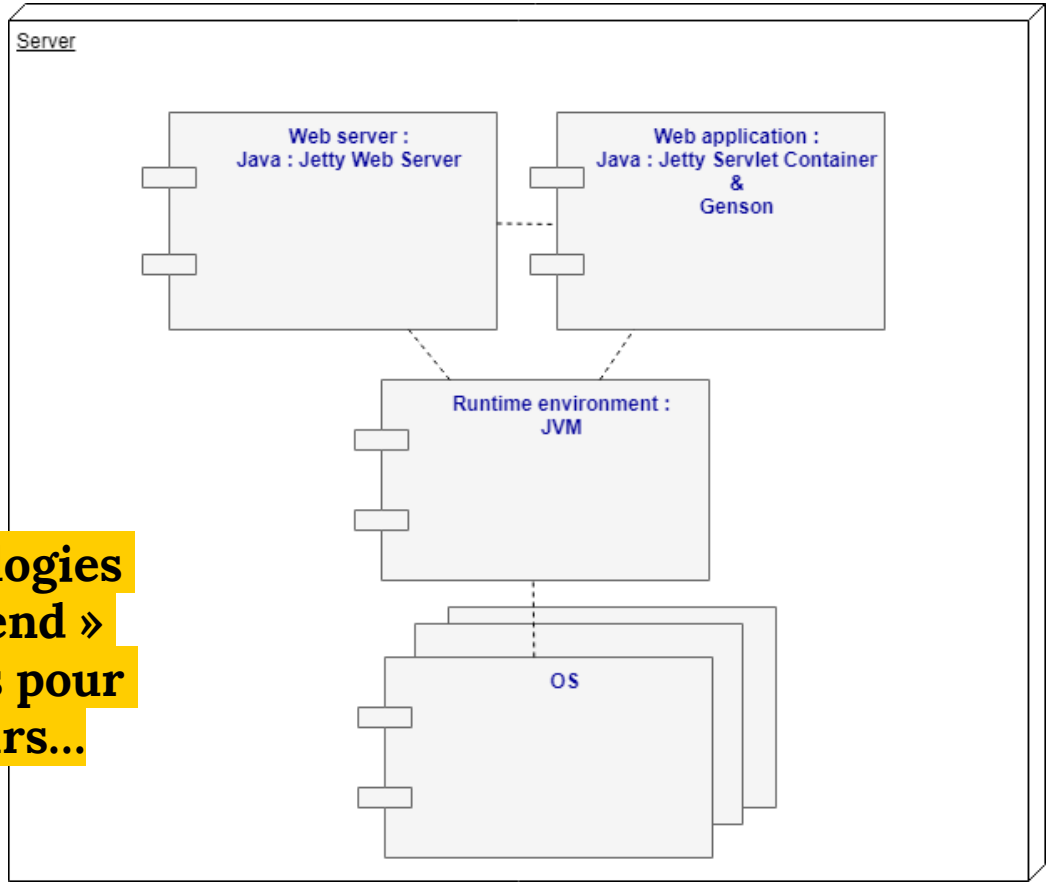
- CRUD Operations = POST, GET, PUT (& PATCH), DELETE methods
  - utilisés dans les API Webs, dans le standard REST;
  - peu utilisées à partir d'un navigateur.
- Exemple de requête GET vers un serveur :





Technologies « backend » possibles...

Technologies « backend » choisies pour ce cours...



Introduction à un backend Java (Jetty) : Mise en place d'un serveur web et de servlets en JAVA pour répondre aux requêtes clientes

- Serveur web pour écouter et répondre aux requêtes
  - Soit déployer les applications web dans un application serveur ; exécutions d'une application web selon le chemin de l'URL :
    - www.monsite.com/app1/...
    - www.monsite.com/app2/...
  - Soit « embedding Jetty » : intégration d'un module HTTP dans une application web (méthode choisie pour ce cours)

Introduction à un backend Java (Jetty) : Mise en place d'un serveur web et de servlets en JAVA pour répondre aux requêtes clientes

- Serveur web pour écouter et répondre aux requêtes
  - Ecoutant sur un port particulier
  - Routant la requête et la réponse en fonction de l'URL et des servlets associées au chemin (URL endpoint)
  - Identifiant l'emplacement des fichiers à publier
  - ...

Introduction à un backend Java (Jetty) : Mise en place d'un serveur web et de servlets en JAVA pour répondre aux requêtes clientes

- Servlet pour traiter les requêtes sur un chemin précis : **HttpServlet**
- Servlet utilisée par un serveur web / Multiples servlets par serveur (multiple chemins de requêtes)

Introduction à un backend Java : Jetty

- Open source
- Solution 100% Java
- Documentation et infos : http://www.eclipse.org/jetty/

Introduction à un backend Java : Jetty

- Nomenclature pour une application web embarquée :
  - **Server** : ce qui écoute sur un port TCP.
  - **WebAppContext** : ce qui configure l'application web embarquée, en spécifiant le routage des requêtes vers les servlet (router)
  - **ServletHolder** : retient le nom et la configuration d'une instance de servlet.
  - **HttpServlet** : classe à étendre pour répondre aux requêtes HTTP (application logic, controller)

Introduction à un backend Java : Jetty

- **DEMO-12** : Mise en place d'un serveur web et d'une application web statique
  Affichage d'une page « Hello world » en allant sur http://localhost:8080/hello ou hello/*.
  Sinon, faire en sorte que l'application web mette à disposition tous les fichiers repris dans le répertoire www du projet (rôle de serveur web de contenu statique).

Introduction à un backend Java : Jetty

- Installation d'Eclipse (Eclipse IDE for Java Developers) ?
- Installation du JDK ?

Introduction à un backend Java : Jetty

- Jetty HelloWord tutorial : https://www.eclipse.org/jetty/documentation/current/advanced-embedding.html#jetty-helloworld
- Embedding Jetty : https://www.eclipse.org/jetty/documentation/current/embedding-jetty.html

Introduction à un backend Java : Jetty

- **DEMO-13** : Communication synchrone client / serveur : mise en place d'une application web dynamique

  A) Reprenons notre formulaire de login (qqs petits changement : action, connect.hmtl, method), mettons le dans un répertoire non public (views), le JS dans un répertoire public (public/myLibrary.js) et rendons son contenu accessible à http://localhost/connect .

Introduction à un backend Java : Jetty

- **DEMO-13** :
  B) Copier ce même formulaire de login dans un répertoire public (autre nom : login.html, autre action) et prévoir une redirection vers http://localhost/login.html quand on accède à http://localhost/login.

Introduction à un backend Java : Jetty

- **DEMO-13** :
  C) Faire en sorte qu'à l'envoi des données du formulaire au serveur (http://localhost/connect), celui-ci affiche un message de bienvenue basé sur les données du formulaire.

Jetty & WebAppContext : quelques méthodes

- **Jcontext.setContextPath("/")** : chemin de l'URL pour lequel ce contexte s'applique
- **context.setWelcomeFiles(new String[] { "index.html" })** : Quel est le fichier à servir si l'utilisateur va à l'URL racine sans plus de précision. www.monsite.com affichera www.monsite.com/index.html

Jetty & WebAppContext : quelques méthodes

- **context.setInitParameter("cacheControl","no-store,no-cache,must-revalidate")** : Dans le protocole HTTP, le serveur dicte le comportement du cache du navigateur. Ici on dit que par défaut, il ne faut pas stocker ni retenir en cache les réponses aux requêtes.

Jetty & WebAppContext : quelques méthodes

- **context.setInitParameter("redirectWelcome", "true")** : Quand c'est 'true', la page de bienvenue est affichée par redirection plutôt que par suivi. En d'autres termes, l'URL change à la page de bienvenue. Dans tous les cas le contenu de cette page sera affichée.

Jetty & WebAppContext : quelques méthodes

- **context.setMaxFormContentSize(50000000)** : Spécifie la taille limite des données qu'un front-end peut soumettre à ce back-end.
- **context.addServlet(new ServletHolder(new MaServlet()), "/url1")** : Enregistre une servlet pour répondre à l'url **/url1** exactement.

Jetty & WebAppContext : quelques méthodes

- **context.addServlet(new ServletHolder(new MaServlet ()), "/url2/*")** :
  - Enregistre une servlet pour répondre à toute url commençant par **/url2**
  - Si plusieurs servlets sont ajoutées, chaque requête est traitée dans l'ordre de cet ajout, jusqu'à trouver une servlet qui accepte de la traiter.

Jetty & WebAppContext : quelques méthodes

- **context.addServlet(new ServletHolder(new DefaultServlet()), "/")** :
  - La DefaultServlet sert des fichiers.
  - Ici toutes les URLs seront traitées comme des fichiers à trouver puisque cela commence à / directement
  - C'est pour cela qu'on ajoute la DefaultServlet en dernier en général.
  - **context.setResourceBase("c:\\web")** spécifie dans quel répertoire se trouvent les fichiers.

Jetty & WebAppContext : quelques méthodes

- Détails : https://www.eclipse.org/jetty/javadoc/9.4.14.v20181114/org/eclipse/jetty/webapp/WebAppContext.html

HttpServlet

- Classe à étendre
- Pour chaque requête HTTP, selon la méthode HTTP (GET, POST…) , une de ces méthodes est appelée :
  - doGet(HttpServletRequest i,HttpServletResponse o)
  - doPost(HttpServletRequest i,HttpServletResponse o)
  - doPut(HttpServletRequest i,HttpServletResponse o)
  - doDelete(HttpServletRequest i,HttpServletResponse o)
  - …

HttpServlet

- **HttpServletRequest** : contient tout ce qui concerne la requête (son chemin, l'adresse de l'émetteur, les cookies, les données transmises, etc...)
- **HttpServletResponse** : possède les méthodes pour préparer la réponse à renvoyer au client.

La requête : HttpServletRequest

- Composition d'une requête (rappel) :
  - Une URL
  - Une méthode : Get/Post/Put/Delete/…
  - Une adresse d'origine
  - Des cookies
  - De données à envoyer
  - D'un mimetype pour les données à envoyer : http://www.sitepoint.com/web-foundations/mime-types-complete-list/
  - Des en-têtes, p.ex. le user agent=browser
  - ….

La requête : HttpServletRequest

- Quelques méthodes :
  - **getPathInfo()** : renvoie la partie d'URL supplémentaire à ce qui est configuré pour le servlet.
  - **getParameter(String)** : obtient un des paramètres de la requête, çad des données transmises.
  - **getCookies()** : retourne les cookies.
- Détails : http://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpServletRequest.html

La réponse : HttpServletResponse

- Composition d'une réponse (rappel) :
  - Un code d'état : 200 = ok, 404 = manquant etc, http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html
  - Des données
  - Un mimetype pour ces données
  - Des modification de cookies (ajout/suppression/modification)
  - Des en-têtes
  - ...

La réponse : HttpServletResponse

- Quelques méthodes :
  - **setStatus(int)** : définit le statut de la réponse
  - **setContentLength(int)** : définit la longueur en bytes de la réponse
  - **setCharacterEncoding(String)** : définit l'encodage de la réponse ("utf-8")
  - **setContentType(String)** : définit le mimetype de la réponse

La réponse : HttpServletResponse

- Quelques méthodes :
  - **getOutputStream()** : retourne l'OutputStream permettant d'écrire le body de la réponse (OK pour des données binaires)
  - **getWriter()** : retourne un PrintWriter pour envoyer des caractères au client (choisir **getWriter()** ou **getOutputStream()**, pas les deux) et donc pour écrire le body de la réponse

La réponse : HttpServletResponse

- Détails : http://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpServletResponse.html

Gestion des exceptions quand utilisation de Jetty

- Si une exception s'échappe, Jetty la gère :
  - Redirection vers une page d'erreur.
  - Ou envoi d'un message standard.
- Bonne pratique : évitez de laisser s'échapper les exceptions.

Gestion des exceptions quand utilisation de Jetty

```
@Override
protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
                        try {
                                ....
                        } catch (Exception e) {
                                e.printStackTrace(); // log the error
                                resp.setStatus(500); // error at the server level
                                resp.setContentType("text/html");
                                byte[] msgBytes=e.getMessage().getBytes("UTF-8");
                                resp.setContentLength(msgBytes.length);
                                resp.setCharacterEncoding("utf-8");
                                resp.getOutputStream().write(msgBytes);
                        }
        }
```

HttpServlet : syntaxe d'extension

```
public class MaServlet extends HttpServlet {
        @Override
        protected void doGet(HttpServletRequest req, HttpServletResponse
resp) throws ServletException, IOException {
                ...
        }
        @Override
        protected void doPost(HttpServletRequest req, HttpServletResponse
resp) throws ServletException, IOException {
                ...
        }
}
```

Exercices

A vous de jouer...

Communication synchrone client / serveur : mise en place d'une application web dynamique

- **EX-07 : Mission**
  Reprenez le formulaire de login. Faites en sorte qu'un utilisateur se logge via l'URL http://localhost/signin.
  Vérifiez que le password corresponde à « Logme » :
  - En cas de succès, n'affichez plus le formulaire de login mais une message dynamique accueillant l'utilisateur.
  - Sinon, affichez un message d'erreur en dessous du formulaire de login.

Communication synchrone client / serveur : mise en place d'une application web dynamique

EX-07 : Instructions

- Mettez en œuvre un concept de « view engine » ou de template HTML.
- Frontend :
  - Créez un fichier **/views/login.html** et **/views/home.html** contenant les views que qui seront à afficher. Au sein de ces fichiers, insérer une div que vous utiliserez côté backend pour insérer du contenu HTML dynamique.
  - Créez un fichier JS qui n'affiche votre div que si son contenu n'est pas vide (**/public/myLibrary.js**).

Communication synchrone client / serveur : mise en place d'une application web dynamique

EX-07 : Instructions

- Backend :
  - Utilisez la librairie **jsoup** pour charger, parser et modifier le contenu HTML de vos views :
    - Télécharger la librairie : https://jsoup.org/download
    - Charger : https://jsoup.org/cookbook/input/load-document-from-file
    - Modifier le contenu de la div (via CSS selector) d'une view : https://jsoup.org/cookbook/modifying-data/set-html
    - Renvoyer le contenu de vos views modifiées

Communication synchrone client / serveur : mise en place d'une application web dynamique

- **EX-07-B** : Expérimentez avec Jetty pour faire évoluer l'application précédente. Par exemple, en cas de connexion sur http://localhost, redirigez vers http://localhost/signin .

Références

[1]	MDN web docs, Introduction to web APIs. Lien : https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Introduction
[2]	MDN web docs, JavaScript Guide. Lien : https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide
[3]	w3schools.com, JavaScript Tutorial. Lien : https://www.w3schools.com/js/default.asp
[4]	tutorialspoints.com, Javascript Tutorial : Lien : https://www.tutorialspoint.com/javascript/index.htm

Références

[5]	Medium.com, Neal Burger, The end of life of IE11. Lien : https://medium.com/@burger.neal/the-end-of-life-of-internet-explorer-11-12736f9ff75f
[6]	w3schools.com, JS HTML DOM. Lien : http://www.w3schools.com/js/js_htmldom.asp
[7]	MDN web docus, Basics of HTTP. Lien : https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP