Programmation Web – Avancé

BINV2150 A : JavaScript (& JAVA SERVLETS)

Week 8

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

0

Table des matières

Tous les sujets traités pendant ce cours...

Table des matières

Table des matières

5

Introduction aux single-page web applications et aux communications asynchrones client / serveur

Frontend de luxe...

Qu’est-ce qu’une SPA ?

- Qu’en pensez-vous ?
  - Pas de rechargement de page
  - Réécriture dynamique
  - Expérience utilisateur augmentée

Quelles protocoles / techniques principales pour une SPA ?

- AJAX :
  - Asynchronous JavaScript and XML
  - Transport de données via XML (autrefois) ou JSON
  - Combinaison de technologies (HTML/CSS, DOM, JSON ou XML, XMLHttpRequest, JS) pour réaliser une application web asynchrone
- Websockets : technologie de communication temps-réel client/serveur bidirectionnelle

AJAX/HTTP Library Comparison [9]

	Support			Features				
	Chrome & Firefox[1]	All Browsers	Node	Concise Syntax	Promises	Native[2]	Single Purpose[3]	Formal Specification
XMLHttpRequest	✓	✓				✓	✓	✓
Node HTTP			✓			✓	✓	✓
fetch()	✓			✓	✓	✓	✓	✓
Fetch polyfill	✓	✓		✓	✓		✓	✓
node-fetch			✓	✓	✓		✓	✓
isomorphic-fetch	✓	✓	✓	✓	✓		✓	✓
superagent	✓	✓	✓	✓			✓	
axios	✓	✓	✓	✓	✓		✓	
request			✓	✓			✓	
jQuery	✓	✓		✓				
reqwest	✓	✓	✓	✓	✓		✓	

[1] **Chrome & Firefox** are listed separately because they support `fetch()`: caniuse.com/fetch [2] **Native:** Meaning you can just use it - no need to include a library. [3] **Single Purpose:** Meaning this library or technology is ONLY used for AJAX / HTTP communication, nothing else.

Introduction à AJAX

- Quelle librairie AJAX (environ 20 ans d'âge) utiliser ? **$.ajax()** ou utilisation de **Fetch API** (jeune « promising » API) ?
  - XMLHttpRequest vs the Fetch API: What's Best for Ajax in 2019? [8] : https://www.sitepoint.com/xmlhttprequest-vs-the-fetch-api-whats-best-for-ajax-in-2019/
  - Difficulté pour choisir votre librairie JS pour AJAX ? Lisez [9] : https://www.javascriptstuff.com/ajax-libraries/

Introduction à AJAX

- **$.ajax() :** http://api.jquery.com/jquery.ajax/

```
$.ajax({
        type: "post",
        url: "/login",
        data: { email: $("#email1").val(), password: $("#password1").val() },
        dataType: "json",
        success: function(response) {
            // Do something : if dataType was specified to “json”, response has already been parsed
            // to an Object. Else : reponse=JSON.parse(response);
},
        error: function name(err, status, message) {
          // Do something in case of error
        }
    });
```

Introduction à AJAX

- **Fetch API:** https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch

```
async function postData(url = '', data = {},token) {
  console.log("data:",data);
  const response = await fetch(url, {
    method: 'POST', // GET, POST, PUT, DELETE, etc.
    body: JSON.stringify(data), // body data type must match "Content-Type" header
    headers: {
      'Content-Type': 'application/json'
      }
  });
  return await response.json();
```

Introduction à AJAX

- **Fetch API:** https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch

```
postData("/login",data)
      .then(response => {
        // success, therefore do something
       })
      .catch(err =>{
        // error, do something
      });
```

Les paramètres dans doPost (HtttpServlet)

- Type de données envoyées par défaut via **$.ajax()** :
  contentType (default: 'application/x-www-form-urlencoded; charset=UTF-8')
- Récupération des paramètres côté serveur :

```
doPost(…){
String email = req.getParameter("email");
String password = req.getParameter("password");
}
```

Les paramètres dans doPost (HtttpServlet)

- Si type de données envoyées au format JSON par client via **$.ajax()** ou **fetch API**, récupération des paramètres côté serveur :

```
doPost(…){
StringBuffer jb = new StringBuffer();
String line = null;
try {
BufferedReader reader = req.getReader();
while ((line = reader.readLine()) != null)
   jb.append(line);
  System.out.println("READER:" + jb.toString());
} catch (Exception e) {    e.printStackTrace(); }
```

Les paramètres dans doPost (HtttpServlet)

- Si type de données envoyées au format JSON par client via **$.ajax()** ou **fetch API**, récupération des paramètres côté serveur :

```
doPost(…){
//deserialize the data
Genson genson = new Genson();
Map<String, Object> map = genson.deserialize(jb.toString(), Map.class);
String email = map.get("email").toString() ;
String password = map.get("password").toString();
…
}
```

Introduction à AJAX

- **DEMO-16** : Upgrade de notre application d'enregistrement d'utilisateurs en SPA. L'app se trouve dans **index.html**. Le frontend de l'application fait régulièrement des demandes de MàJ de la liste d'utilisateurs qui doit s'afficher sans action de l'utilisateur.

Introduction à AJAX

- **DEMO-16** :
  L'application web Jetty fait office d'API :
  - **GET /users** : envoi de la liste de tous les utilisateurs (format JSON) en cas de succès, ou sinon d'un message d'erreur
  - **POST /users** : ajout d'un utilisateur à la BD JSON et renvoi de l'info succès , ou sinon d'un message d'erreur (format JSON)

6

Introduction à l'authentification sécurisée d'un utilisateur, aux cookies et au localStorage

En route vers une authentification « stateless »...

Différents moyens d'authentification ?

- Stateful authentication (session management)
  - User session data : côté serveur (avec cookie géré par Jetty)
  - Inconvénients & Avantages ?
- Stateless authentication
  - User session data : côté client
  - Inconvénients & Avantages ?
  - Exemple : JWT

JWT

- JSON Web Token (JWT) : https://jwt.io/
- JWT : **xxxxx.yyyyy.zzzzz**
  - Header : encodé
  - Payload : encodé, pas crypté !
  - Signature : cryptée !

JWT

- JWT: https://github.com/auth0/java-jwt
- Création d'un token en cas d'authentification réussie :

```
Map<String, Object> claims = new HashMap<String, Object>();
claims.put("id", id);
claims.put("ip", req.getRemoteAddr());
String ltoken = new JWTSigner(JWTSECRET).sign(claims);
```

- **JWTSECRET** : string connue uniquement par un serveur permettant de signer/valider une signature.

JWT

- Vérification d'un token lors d'une requête (user authorization)

```
Object userID = null;
try {
  Map<String, Object> decodedPayload = new JWTVerifier(JWTSECRET).verify(token);
  userID = decodedPayload.get("id");
} catch (Exception exception) {
  // return some error
}
if (userID!=null)  { // user is valid, return some API data e.g.
} else {
 // return some message error, such as : "Unauthorized: this ressource can only be accessed with a valid token”
}
```

Les cookies

- Données envoyées par un serveur vers un client
- But :
  - **Gestion de session**
  - Personnalisation
  - Tracking
- Autrefois : « general client-storage »

Les cookies

- Cookies sont envoyés automatiquement pour chaque requête vers le domaine
- Actuellement :
  - Protection contre les attaques XSS : utiliser **HttpOnly** pour rendre les cookies inaccessible au JS
  - Ou utilisation du **localStorage** mis à disposition par les browsers modernes plutôt que les cookies

Les cookies

- Création d'un cookie nommé « token »

```
//the token has been generated, save it in the cookie, and give  the name token to your cookie
Cookie cookie = new Cookie("token", ltoken);
cookie.setPath("/");
//for security reason, don't allow JS to touch the cookie
cookie.setHttpOnly(true);
// expiration time of a cookie in seconds
cookie.setMaxAge(60 * 60 * 24 * 365);
resp.addCookie(cookie);
```

Les cookies

- Lors des requêtes ultérieurs, lecture du token parmis les cookies reçus

```
String token = null;
Cookie[] cookies=req.getCookies();
if (cookies != null) {
  for (Cookie c : cookies) {
    if ("token".equals(c.getName()) ) {
       token = c.getValue();
       Map<String, Object> decodedPayload = new JWTVerifier(JWTSECRET).verify(token);
       userID = decodedPayload.get("id");
       ip = decodedPayload.get("ip");
       }
    }
}
```

Introduction à l'authentification sécurisée d'un utilisateur et aux cookies

- **DEMO-17** : la SPA doit permettre de se logger via un username (email ici) et un password (« Logme ») en accédant à http://localhost:8080 . Si les credentials sont OK, fourniture d'un token. Une fois loggé, il est possible d'accéder à l'application précédente (vue de tous les utilisateurs enregistrés) SSI le token est donné par le client.

Introduction à l'authentification sécurisée d'un utilisateur et aux cookies

- **DEMO-17** :
  Si le token n'est pas donné, ou n'est pas valide, ne pas afficher l'application précédente.
  - **DEMO-17A** : Partial stateful SPA : 1er step, sans cookie, sans JWT, avec gestion de session (rien de neuf).
  - **DEMO-17B** : Partial stateless SPA : 2ème step, avec JWT, sans session, avec cookie.

localStorage & sessionStorage

- « Key-value store » : toujours des strings
- **localStorage** : pas d'expiration
- **sessionStorage** : effacé à la fin d'une session

- Détails : https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage

localStorage & sessionStorage

getItem()

```
const retrievedValue = localStorage.getItem('testObject');
const object = JSON.parse(retrievedValue);
```

localStorage & sessionStorage

setItem()

```
const testObject = { 'one': 1, 'two': 2, 'three': 3 };
const storageValue = JSON.stringify(testObject);

localStorage.setItem('testObject', storageValue);
```

localStorage & sessionStorage

- removeItem()

```
localStorage.removeItem('token');
```

- localStorage.clear()

Introduction à l'authentification sécurisée d'un utilisateur et aux cookies

- **DEMO-17C** : 3ème step : Full stateless SPA, avec JWT, sans session, sans cookie mais avec **localStorage** et **fetch API** (au lieu de **$.ajax()**). Ajout d'une nouvelle API **POST /login** qui retourne un token en cas d'authentification réussie (password=« Logme »).

7

Projet mettant en œuvre une SPA et des librairies JS

Faites ce qui vous tient à cœur et épatez vous en asynchrone ...

Projet mettant en œuvre une SPA et des librairies JS (rappel)

- Consignes détaillées : voir template du projet
- Développement :
  - Investissement proposé : 12h en classe + ~12h suppl.
  - Durée : 3 semaines (semaines 9 à 11)
  - Présentation et validation de vos objectifs (semaine 9)
  - Groupes de 2 étudiants, sauf exception
- Présentation (4h en classe) : semaine 12

Exercices

A vous de jouer...

Frontend asynchrone, API et JWT

- **EX-10** : **Mission** : SPA (http://localhost:8080) donnant accès à une API de films suite à l'authentification d'un utilisateur.

  Instructions :
  A) Réutilisez le composant de login (#login_component) ainsi que l'API **POST /login** pour authentifier un utilisateur et sauvegarder le JWT

Frontend asynchrone, API et JWT

- **EX-10** :
  B) Créez l'API films

Film function	Endpoint URL	HTTP method
Create	/films	POST
Read (all)	/films	GET

« BD » films sous forme de fichiers **.json**

```
[{id:1, title:'Smile', duration:120, producer:'Romeo', budget:1000000},{….},{…}]
```

Frontend asynchrone, API et JWT

- **EX-10** :
  C) Créez le frontend pour consommer votre API. Quand un utilisateur est authentifié :
  Affichez un menu permettant :
  - de voir la liste des films disponibles (consommer **GET /films**)
  - d'ajouter un film (formulaire consommant **POST /films**)
  - de se délogger (effacer le token et affichage du composant de login)

Frontend asynchrone, API et JWT

- **EX-11** : Si ça n'est pas déjà fait, sécurisez votre API pour toutes les opérations CRUD sur un film : celles-ci ne sont possibles que si le client fourni un token valide.
- **EX-12** : Mission : complétez votre SPA et films API (pour toutes les méthodes CRUD) afin de pouvoir mettre à jour ou effacer un film.

Film function	Endpoint URL	HTTP method
Update	/films/:id	PUT (or POST)
Delete	/films/:id	DELETE

Frontend asynchrone, API et JWT

- **EX-12 : Notions**
  - Récupérez l'id du chemin, par exemple , pour l' URL endpoint /films/*

```
doPut(…){
String pathInfo = req.getRequestURI(); // /films/:id
String[] pathParts = pathInfo.split("/");
String id;
if(pathInfo.length()>2) {
  id = pathParts[2]; // :id
 }
…
}
```

Frontend asynchrone, API et JWT

- **EX-13** (optionnel) : Au sein de votre SPA films, ajoutez la possibilité de faire des critiques de films, de voir les critiques sur un film.

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
[8]	sitepoint, Craig Buckler, XMLHttpRequest vs the Fetch API: What's Best for Ajax in 2019?. Lien : https://www.sitepoint.com/xmlhttprequest-vs-the-fetch-api-whats-best-for-ajax-in-2019/

Références

[9]	javascriptstuff.com, AJAX/HTTP Library Comparison. Lien https://www.javascriptstuff.com/ajax-libraries/
[10]	
[11]	
[12]	