

Programmation Web – Avancé

JavaScript & Node.js

Partie 24 : Gestion de session côté client

Version 2020



Attribution –
Partage dans les
Mêmes Conditions
4.0 International
(CC BY-SA 4.0)

*Presentation template
by [SlidesCarnival](#)*



Gestion de session côté client

Et si le client se souvenait de ses données de session ?



Déploiement

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



Les cookies

- Envoi automatique des cookies pour chaque requête vers le domaine
- Actuellement :
 - Protection contre les attaques XSS : utilisation de **HttpOnly** pour rendre les cookies inaccessible au JS
 - Ou utilisation du **localStorage** mis à disposition par les browsers modernes plutôt que les cookies



Gestion de session côté client avec Express

- Session middleware : **cookie-session** [\[86.\]](#)
- Enregistrement de données de session dans le cookie
- Installation : **npm install cookie-session**



Gestion de session côté client avec Express

● Utilisation du middleware :

```
var cookieSession = require("cookie-session");
let expiryDate = new Date(Date.now() + 60 * 60 * 1000); // 1h;
app.use(
  cookieSession({
    name: "user",
    keys: ["689HiHoveryDi79*"],
    cookie: {
      httpOnly: true,
      expires: expiryDate,
    },
  })
);
```



Gestion de session côté client avec Express

Données de sessions attachées aux cookies :

- Lecture & modification
de données :

req.session (comme
pour **express-session**)

```
// initialize the session data
app.use(function (req, res, next) {
  if (!req.session.isAuthenticated) {
    req.session.isAuthenticated = false;
    req.session.user = "";
  }
  next();
});
```

```
router.post("/", function (req, res, next){
  // ...
  req.session.isAuthenticated = true;
  req.session.user = req.body.email;
  return res.json({ username: req.body.email });
});

router.get("/list", function (req, res, next){
  if (req.session.isAuthenticated)
    if (User.isUser(req.session.user))
      return res.json(User.list);
    else return res.status(401).send("You are not
      authenticated.");
});
```



Gestion de session côté client avec Express

- Données de sessions attachées aux cookies :
 - Effacer une session : **req.session = null**

```
/* GET logout */
router.get("/logout", function (req, res, next) {
  console.log("GET users/logout");
  if (req.session.isAuthenticated) {
    req.session = null;
    return res.status(200).end();
  }
});
```




Visualisation des cookies au sein du browser

● cookieName

- Base64 encoded
- Décodeur base64 [\[87.\]](#)

● cookieName.sig : prévention contre le « tampering »

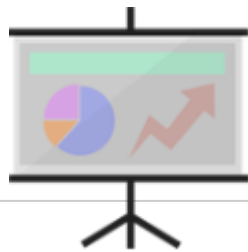


< DECODE >

Decodes your data into the textarea below.

```
{"isAuthenticated":false,"user":""}
```

**Vue du cookie au chargement
de MyCMS**



Cookies et session côté client

🕒 DEMO : SPA monolithique avec session côté client via des cookies

Garder toutes les infos de session côté client :
username et isAuthenticated

- Pas d'info de session sauvegardées côté serveur
- Problème : « secured » routes seulement si utilisation des formulaires du site / seulement si cookie avec isAuthenticated
- Garder l'info de password côté client ? Pas une bonne idée...



Cookies et session côté client

🕒 DEMO : SPA monolithique avec session côté client via des cookies

Garder toutes les infos de session côté client :
username et isAuthenticated

- Pas d'info de session sauvegardées côté serveur
- Prévoir des "secured" routes



Web storage : `localStorage` & `sessionStorage`

- « Key-value store » : toujours des strings
- `localStorage` [\[88.\]](#) : pas d'expiration
- `sessionStorage` : effacé à la fin d'une session



Web storage : localStorage & sessionStorage

🕒 getItem()

```
const STORE_NAME = "user";
const getUserSessionData = () => {
  const retrievedUser = localStorage.getItem(STORE_NAME);
  if (!retrievedUser) return;
  return JSON.parse(retrievedUser);
};
```



Web storage : localStorage & sessionStorage

● `setItem()`

```
const setUserSessionData = (user) => {  
  const storageValue = JSON.stringify(user);  
  localStorage.setItem(STORE_NAME, storageValue);  
};
```



Web storage : localStorage & sessionStorage

● removeItem()

```
const removeSessionData = () => {  
  localStorage.removeItem(STORE_NAME);  
};
```

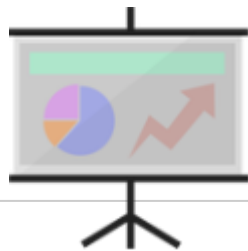
● localStorage.clear()



Web storage : localStorage & sessionStorage

🕒 DEMO : SPA monolithique avec session côté client via le localStorage

- Pas d'info de session côté serveur
- Trou de sécurité : plus de « secured » routes sans avoir l'info de password côté client ou autre mécanisme...
- Petite « sécurité » pour afficher la liste des utilisateurs :
 - attente du username
 - attente du password ? Ou password crypté ?



Web storage : localStorage & sessionStorage

- ◎ DEMO : SPA monolithique avec session côté client via le localStorage
 - Semble « secured » :
<http://localhost:3000/api/users>
 - Néanmoins :
<http://localhost:3000/api/users?username=teacher@vinci.be>
 - Est-ce OK de garder les passwords en clair côté serveur ?



Gestion de sessions

- Côté serveur ou côté client ?
- Côté serveur : **stateful server**
- Côté client : **stateless server**
- Avantages & inconvénients ?