

Programmation Web – Avancé

JavaScript & Node.js

Partie 28 : Gestion de la sécurité du browser

Version 2020



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

*Presentation template
by [SlidesCarnival](#)*



Introduction aux procédures de sécurité appliquées par votre browser

Pourquoi mon application frontend ne peut pas communiquer avec
l'application backend ?



Sécurité des communications : SOP

- ◎ SOP = Same Origin Policy [\[93.\]](#)
- ◎ Appliquée par le browser
- ◎ But :
 - Restriction des interactions entre un document ou script chargé par une origine avec une ressource d'une autre origine
 - Isolation des documents ou scripts malicieux, réduction des attaques



Sécurité des communications : SOP

- Même origine entre deux URLs si même :
 - Protocole
 - Port
 - Host



Sécurité des communications : SOP

Type d'interaction	SOP Permission	Ressources
Cross-origin writes Requêtes vers une autre origine	Permis	Liens, redirection, soumission de formulaires
Cross-origin embedding	Permis	JavaScript via <code><script src="..."></code> <code></script></code> , CSS via <code><link rel="stylesheet" href="..."></code> , Images, Media, iframes...
Cross-origin reads Réponse d'une autre origine	Interdit	



Relaxer la sécurité via des CORS

- ◎ CORS = Cross Origin Resource Sharing [\[94.\]](#)
- ◎ Spécification par le serveur :
 - des origines pouvant lire ses ressources via un web browser / pouvant accéder à ses réponses
 - via des « HTTP headers »



Problème de sécurité : CORS trop large





Sécurité des communications : SOP

● DEMO :

Trouver schéma et explications ...



NB : attaque XSS





Option A : Relaxer la sécurité via des CORS

- Installation du package cors [\[95.\]](#): `npm install cors`
- Headers configurés au niveau du backend via Middleware
- Configuration et utilisation

```
var cors = require('cors');  
let corsOptions = {  
  origin: 'http://localhost',  
}  
// enable CORS for all routes in the given router  
app.use("/api/users", cors(corsOptions), usersRouter);
```



Option A : Relaxer la sécurité via des CORS

- DEMO : SPA avec frontend indépendant du backend : gestion de CORS

Relaxer la sécurité en ajoutant les CORS au niveau du backend

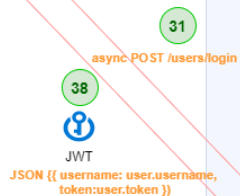
- OK, mais quid si le backend est non modifiable ?



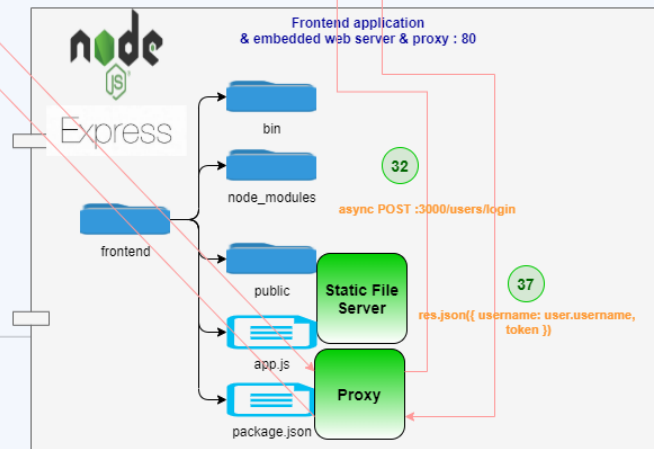
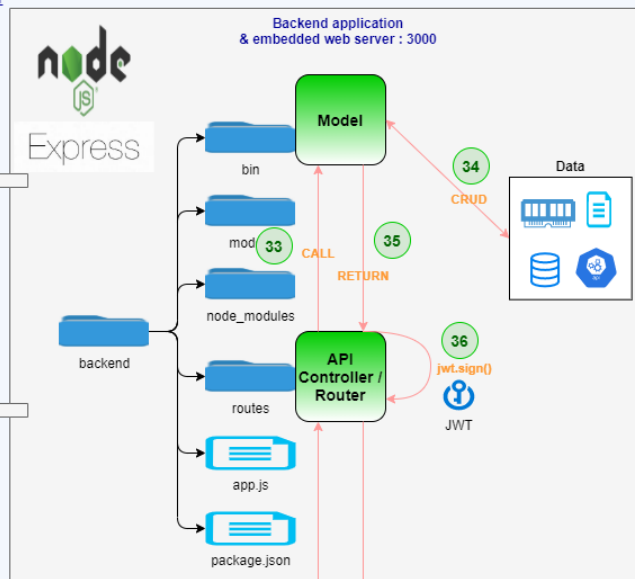
Option B : Contourner le SOP via un proxy au niveau du frontend

- Transfert des requêtes du frontend vers vos Web API

Redirection des requêtes via un proxy



Server





Option B : Contourner le SOP via un proxy au niveau du frontend

- Proxy pour développement
 - Existence d'une multitude de proxys
 - Proxy complet sous Node : [http-proxy-middleware](#)
[\[96.\]](#)



Option B : Contourner le SOP via un proxy au niveau du frontend

☉ Dev proxy : [http-proxy-middleware](#) [\[96.\]](#)

```
const { createProxyMiddleware } = require("http-proxy-middleware");
app.use("/api",
  createProxyMiddleware({target: config.API_URL,
    changeOrigin: true,
    logLevel: "debug",
    /* If we wanted that the call to http://localhost/api were transformed to
       API_URL/ (instead of API_URL/api/)
    */
    /*pathRewrite: {
      '^/api/': '/' // remove base path
    },*/
  })
);
```



Option B : Contourner le SOP via un proxy au niveau du frontend

- Dev proxy
 - Autre option : Configuration du serveur de développement de Node (**package.json**), e.g. :

```
"proxy": "http://localhost:3000",
```




Option B : Contourner le SOP via un proxy au niveau du frontend

- Proxy pour la production : voir les instructions de votre provider



Option B : Contourner le SOP via un proxy

🕒 DEMO : SPA avec frontend indépendant du backend : gestion de proxy

Gérer la sécurité en ajoutant un proxy au
niveau du frontend via http-proxy-
middleware