

Programmation Web – Avancé

JavaScript & Node.js

Partie 22 : SPA monolithique, communications

AJAX avec RESTful API

Version 2020



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

*Presentation template
by [SlidesCarnival](#)*



Gestion des communications d'une SPA

Comment est-ce que nous allons gérer les données au sein d'une SPA ?



Quelles protocoles / techniques principales pour communiquer avec une SPA ?

- AJAX :
 - Asynchronous JavaScript and XML
 - Requête HTTP asynchrone
 - 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

	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.

Comparaison de bibliothèques AJAX/HTTP [73.]



Appel asynchrone d'une opération d'une Web API

- Quelle librairie AJAX (environ 20 ans d'âge) utiliser pour ce cours ?
- Nouveau standard : **Fetch API**
- Anciennement :
 - **XMLHttpRequest**
 - **\$.ajax()**



Fetch : requête asynchrone vers une API

🕒 Fetch API [\[74.\]](#)

```
fetch("/api/users")
  .then((response) => {
    if (!response.ok)
      throw new Error(
        "Error code : " + response.status + " : " + response.statusText
      );
    return response.json();
  })
  .then((data) => onUserList(data)) // build and render the user list
  .catch((err) => onError(err));
```



Fetch : requête asynchrone vers une API

🕒 Fetch API [\[74.\]](#)

```
fetch("/api/users/", {  
  method: "POST", // *GET, POST, PUT, DELETE, etc.  
  body: JSON.stringify(user), // body data type must match "Content-Type" header  
  headers: {  
    "Content-Type": "application/json",  
  },  
})  
  .then((response) => { // deal with errors  
    return response.json();  
  })  
  .then((data) => onUserRegistration(data)) // re-render navBar and go to /list  
  .catch((err) => onError(err));
```



JQuery : requête asynchrone vers une API

◎ \$.ajax() [\[75.\]](#)

```
$.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  
  },  
});
```




Gestion de l'opération sur les ressources et réponse au frontend

- Gestion des requêtes :
 - Soit Serveur de Fichiers Statiques
 - Soit Renvoi de index.html
 - Soit API Router
- Proposition : Gestion d'opération sur des ressources si les requêtes commencent par **/api/**

```
app.use((req, res, next) => {  
  if (!req.path.startsWith("/api/"))  
    return res.sendFile(`${__dirname}/public/index.html`);  
  next();  
});
```



Gestion de l'opération sur les ressources et réponse au frontend

- Récupération des données dans l'URL de la requête (**GET**, **DELETE**)

```
router.get("/:username", function (req, res, next) {  
  console.log("GET users/:username", req.params.username);  
  const userFound = User.getUserFromList(req.params.username);  
  if (userFound) {  
    return res.json(userFound);  
  } else {  
    return res.status(404).send("ressource not found");  
  }  
});
```



Gestion de l'opération sur les ressources et réponse au frontend

← → ↻ ⓘ localhost:3000/api/users/teacher@vinci.be

📁 Applications 📁 IT 📁 Recipes 📁 Education 📁 Hobbies 📁 Bank&Insurance&T...

```
{"username":"teacher@vinci.be","email":"teacher@vinci.be","password":"Teacher"}
```

← → ↻ ⓘ localhost:3000/api/users/student@vinci.be

📁 Applications 📁 IT 📁 Recipes 📁 Education 📁 Hobbies

ressource not found



Gestion de l'opération sur les ressources et réponse au frontend

- Récupération des données dans le body de la requête (**POST**, **PUT**...)

```
router.post("/", function (req, res, next) {  
  console.log("POST users/", User.list);  
  console.log("email:", req.body.email);  
  if (User.isUser(req.body.email))  
    return res.status(409).end();  
  let newUser = new User(req.body.email, req.body.email, req.body.password);  
  newUser.save();  
  req.session.isAuthenticated = true;  
  req.session.user = req.body.email;  
  return res.json({ username: req.body.email });  
});
```



Gestion de la réponse et éventuelle MàJ de l'affichage de la SPA

- Gestion des codes de status HTTP [\[76.\]](#)
 - Réponses informatives (100–199),
 - Réponses en cas de succès (200–299),
 - Redirections (300–399),
 - Erreurs du client (400–499),
 - Erreurs du serveur (500–599).



Gestion de la réponse et éventuelle MàJ de l'affichage de la SPA

● Gestion des codes de status HTTP [\[76.\]](#)

```
router.post("/login", function (req, res, next) {  
  let user = new User(req.body.email, req.body.email, req.body.password);  
  console.log("POST users/login:", User.list);  
  if (user.checkCredentials(req.body.email, req.body.password)) {  
    req.session.isAuthenticated = true;  
    req.session.user = req.body.email;  
    return res.json({ username: req.body.email });  
  } else {  
    return res.status(401).send("bad email/password");  
  }  
});
```



Gestion de la réponse et éventuelle MàJ de l'affichage de la SPA

● Gestion des codes de status HTTP [\[76.\]](#)

```
fetch("/api/users/login", {  
  // method, body, headers ...  
})  
  .then((response) => {  
    if (!response.ok)  
      throw new Error("Error code:" + response.status + " : " + response.statusText);  
    return response.json();  
  })  
  .then((data) => onUserLogin(data))  
  .catch((err) => onError(err));  
};
```



Gestion de la réponse et éventuelle MàJ de l'affichage de la SPA

● Gestion des codes de status HTTP [\[76.\]](#)

```
const onError = (err) => {  
  let messageBoard = document.querySelector("#messageBoard");  
  let errorMessage = "";  
  if (err.message.includes("401")) errorMessage = "Wrong username or password.";  
  else errorMessage = err.message;  
  messageBoard.innerText = errorMessage;  
  // show the messageBoard div (add relevant Bootstrap class)  
  messageBoard.classList.add("d-block");  
};
```




Consommer des web APIs rapidement



localhost:3000/api/users/login



Applications



IT



Recipes



Education

ressource not found



Consommer des web APIs rapidement

- Installation du REST Client [\[77.\]](#) dans VS Code
- Création de fichiers **.http**
- Ajout des requêtes vers vos web APIs

```
### POST /user/login for a user that does not exist  
Send Request  
POST http://localhost:3000/api/users/login
```

```
1 HTTP/1.1 401 Unauthorized  
2 X-Powered-By: Express  
3 Content-Type: text/html; charset=utf-8  
4 Content-Length: 18  
5 ETag: W/"12-1McGD5NQMPAKQrUcKcYA2woAoEw"  
6 Date: Thu, 30 Jul 2020 10:05:49 GMT  
7 Connection: close  
8  
9 bad email/password
```



Consommer des web APIs rapidement

```

### POST users : john@here.com / john
Send Request
POST http://localhost:3000/api/users/
Content-Type: application/json

{
  "email": "john@here.com",
  "password": "John"
}

1 HTTP/1.1 200 OK
2 X-Powered-By: Express
3 Content-Type: application/json; charset=utf-8
4 Content-Length: 28
5 ETag: W/"1c-46ddGT2wnHx0IFplr3gPHnL2CZc"
6 Set-Cookie: connect.sid=s%3A18YhgPtEL4R74B7ITUDEXMPK159
  KM-QA.mk%2BlZqKwvEFOIMPObMxiWc7a6n1d3eHpuP%2B%2FrnlVdh
  E; Path=/; HttpOnly
7 Date: Thu, 30 Jul 2020 10:28:17 GMT
8 Connection: close
9
10 {
11   "username": "john@here.com"
12 }

```



Consommer des web APIs rapidement

```
### POST/users/login for an existant user
Send Request
POST http://localhost:3000/api/users/login
Content-Type: application/x-www-form-urlencoded

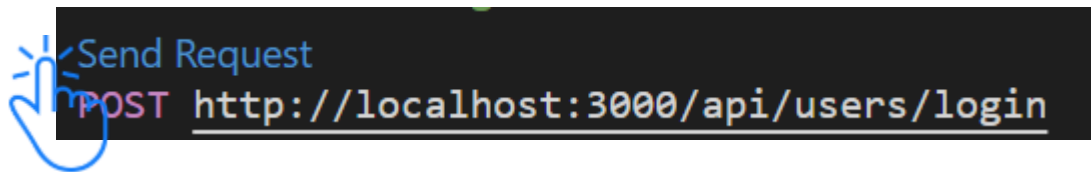
email=john@here.com
&password=John

1 HTTP/1.1 200 OK
2 X-Powered-By: Express
3 Content-Type: application/json; charset=utf-8
4 Content-Length: 28
5 ETag: W/"1c-46ddGT2wnHx0IFplr3gPHnL2CZc"
6 Date: Thu, 30 Jul 2020 10:35:18 GMT
7 Connection: close
8
9 {
10   "username": "john@here.com"
11 }
```



Consommer des web APIs rapidement

- REST Client :
 - Séparation des requêtes par ### (ou plus)
 - Exécution d'une requête :



- Existence de beaucoup d'autres options pour tester vos APIs, notamment Postman [\[78.\]](#)



Introduction à AJAX

🕒 DEMO : SPA monolithique & communications AJAX

- Màj de MyCMS sur base d'une RESTful API
- **GET /users** : envoi de la liste de tous les utilisateurs
- **POST /users** : ajout d'un utilisateur à la BD JSON
- **GET /users/login** : authentification d'un utilisateur
- **GET /users/logout** : logout d'un utilisateur