

Programmation Web – Avancé

JavaScript & Node.js

Introduction à JSON



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

*Presentation template
by [SlidesCarnival](#)*



Introduction à JSON

Comment structurer ses messages ? Comment structurer de l'info que l'on souhaite enregistrer ?



Introduction à JSON

- JavaScript **O**bject **N**otation = JSON
- Syntaxe pour échanger et faire persister des données
- JSON : texte en notation JS



Introduction à JSON

- Types de données valides :
 - string
 - number
 - object
 - array
 - boolean
 - null
- Donc pas de **function**, **date** et **undefined**



Exemple de données JSON

```
[  
  {  
    "email": "raphael@voila.com",  
    "fullname": "Raphael Baroni"  
  },  
  {  
    "email": "jkj@herenqn.com",  
    "fullname": "JK Roling"  
  },  
  {  
    "email": "serena@gmail.com",  
    "fullname": "Serena Here"  
  }  
]
```



Enregistrement de fichiers JSON côté serveur

- Type de fichiers : **.json**
- MIME : **“application/json”**



Exemple de données JSON

```
[  
  {  
    "email": "raphael@voila.com",  
    "fullname": "Raphael Baroni"  
  },  
  {  
    "email": "jkj@herenqn.com",  
    "fullname": "JK Roling"  
  },  
  {  
    "email": "serena@gmail.com",  
    "fullname": "Serena Here"  
  }  
]
```



Sérialisation et désérialisation de données

- Conversion d'un objet JS en JSON pour envoi vers une application : **JSON.stringify(myObj)**

```
fetch(API_URL + "users/login", {  
  method: "POST", // *GET, POST, PUT, DELETE, etc.  
  body: JSON.stringify(user), // body data  
  headers: {"Content-Type": "application/json"},  
}).then((response) => {  
  if (!response.ok) throw new Error("Error...");  
  return response.json();  
})  
.then((data) => onUserLogin(data))  
.catch((err) => onError(err));
```

Côté client



Sérialisation et désérialisation de données

- Conversion d'un objet JS en JSON pour envoi vers une application : **res.json()** (Express)

```
// GET /pizzas : read all the pizzas from the menu
router.get("/", function (req, res) {
  console.log("GET /pizzas");
  return res.json(menu);
});
```

Côté serveur



Sérialisation et désérialisation de données

- Conversion des données JSON reçues d'une application en un objet JS :
JSON.parse(myJSON) ou **json()**

```
fetch(API_URL + "users/login", {method: "POST",  
  body: JSON.stringify(user), // body data  
  headers: {"Content-Type": "application/json"},},  
).then((response) => {  
  if (response.ok) return response.json();  
  .then((data) => onUserLogin(data))  
  .catch((err) => onError(err));
```

Côté client

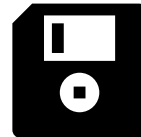


Sérialisation et désérialisation de données

- Conversion des données JSON reçues d'une application en un objet JS : utilisation d'un middleware côté serveur (**express.json()**)

```
app.use(express.json()); /app.js  
// POST /pizzas : create a pizza to be added to the menu.  
router.post("/", function (req, res) {  
  // check body parameters...  
  const newPizza = {  
    id: nextId, title: req.body.title, content: /routes/pizzas.js req.body.content,  
  };  
  // add the pizza and return it to the client
```

Côté serveur



Sérialisation et désérialisation de données

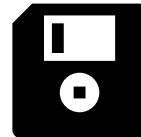
- Persistance de données grâce au format JSON :
JSON.stringify(myObj)

```
function saveUserListToFile(filePath, userList) {  
  const fs = require("fs");  
  let data = JSON.stringify(userList); // userList is an array of objects  
  fs.writeFileSync(filePath, data);  
}
```

Côté serveur

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

Côté client

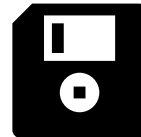


Sérialisation et désérialisation de données

- Lecture de données sauvegardées au format JSON : **JSON.parse(myJSON)**

```
function getUserListFromFile(filePath) {  
  const fs = require("fs");  
  if (!fs.existsSync(filePath)) return [];  
  let userListRawData = fs.readFileSync(filePath);  
  let userList;  
  if (userListRawData) userList = JSON.parse(userListRawData);  
  else userList = [];  
  return userList;  
}
```

Côté serveur



Sérialisation et désérialisation de données

- Lecture de données sauvegardées au format JSON : **JSON.parse(myJSON)**

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

Côté client



Création d'une RESTful API sous Express : JSON

- DEMO : Création d'une RESTfull API pour une pizzeria : Part 2 –Gestion de la persistance des données dans un fichier .json



Redémarrage automatique du serveur lors d'une modification de l'application : **nodemon**

- **npm install -g nodemon**

```
"scripts": {  
  "debug": "nodemon ./bin/www",  
  "start": "node ./bin/www"},
```

/package.json

- **npm run debug**



Création d'une RESTful API sous Express : JSON

🕒 DEMO : Création d'une RESTfull API pour une pizzeria : Part 2 – CRUD pizzas : Persistance des données dans un fichier .json



- Exclure des fichiers du redémarrage automatique de **nodemon** (/data) : **nodemon**

```
"nodemonConfig": {  
  "ignore": [  
    "data/*"  
  ]  
},
```

/package.json