Livrable_SharingFile

Partage de fichiers sur serveur HTTP

A. Description du système du point de vue de l'utilisateur (spécification fonctionnelle):

Le serveur HTTP de partage de fichiers présente l'accessibilité directe des données depuis le navigateur du client.
La particularité de ce type de serveur réside sur le fait qu'aucune installation n'est nécessaire pour le client.
N'importe qui peut donc envoyer et télécharger des fichiers *sans être obliger de posséder un logiciel client comme c'est le cas de serveur FTP*.

1. **Les spécifications fonctionnelles sont comme suit :**
   - permettre à l'utilisateur de pouvoir ajouter des fichiers sur le serveur.
   - permettre le téléchargement des ressources partagées telles que des fichiers images, vidéo et autres.

Le modèle pour le stockage et la représentation des ressources

- Les données ajoutées sont stockées dans un répertoire de l'application à savoir **sharingFile**.
- Chaque fichier ajouté est indexé comme un fichier uploads , suivi d'une clé de hachage aléatoire et son nom d'origine:
  1. Exemple : **uploads**-AyHzziB7W4QNHn8ER-**exec_dechiffre_pict1.png**
- Les fichiers sont copiés selon ce modèle dans le répertoire courant de l'application.
- La représentation de la structure du site est la suivante :

```
\.
├──/.meteor
├──/.idea
├── sharingFile.css
├──sharingFile.html
├── sharingFile.js
├── package.json
├── uploads-AyHzziB7W4QNHn8ER-exec_dechiffre_pict1.png
├── uploads-geShvqendo5zZARgo-2013-07-21-Blackhole-EK-traffic.pcap
├── uploads-gFxDk2YExXydBPyon-trace3.pcapng
├── uploads-nSAwxZxvz7LcsasRC-team06.pcap
└── uploads-y4vqpATbwxYdNnsuq-test.bro
```

2. **Réalisation de l'application qui va gérer l'interaction avec l'utilisateur** :

Pour réaliser l'application, nous avons passé par **Meteor** (voir https://www.meteor.com/). **Meteor** englobe complétement Node.js, si bien que c'est Meteor qui est exécuté, et non l'exécutable de Node.js. Il s'agit donc d'un système qui a besoin de Node.js pour exister.

Les sept principes de Meteor tirés de sa documentation:

1. **Data on the Wire** : Meteor, n'envoie pas de code HTML, il envoie des données et laisse au client le soin de faire le rendu de l'application.
2. **One Language** : javascript
3. **Database Everywhere:** vous pouvez autant accéder à la base de données depuis le client que depuis le serveur
4. **Latency Compensation** : Meteor étant basé sur le temps réel,
5. **Full Stack Reactivity** : Le temps réel dans Meteor n'est pas une exagération
6. **Embrace the Ecosystem** : Meteor est un framework complétement open-source, de même que ses composants et ses nombreux plugins. Si vous devenez assez bons, vous pourriez donc vous même créer vos propres plugins.
7. **Simplicity equals Productivity** : une fois pris en main meteor n'est pas trés compliqué.

Installation de Meteor

curl https://install.meteor.com/ | sh

Une fois Meteor installé, créé le projet avec la commande

```
$ meteor create sharingFile
```

Modules nécessaire pour cette application

- root@:~/ sharingFile# meteor add **cfs:standard-packages**
  cfs:standard-packages: Filesystem for Meteor, collectionFS

- root@:~/ sharingFile# meteor add **cfs:filesystem**

- root@:~/ sharingFile# meteor add **mrt:bootstrap-3**

Module non implementer

- **meteor add iron:router** : module indispensable pour configurer les routes de l'application:
  - Cependant notre application ne comporte qu'une seule page d'où l'omission volontaire des routes dans «**sharingFile**».

Démarrage de l'application :

- Décompresser le fichier **«sharingFile.zip»**
- $ cd sharingFile
- $ meteor

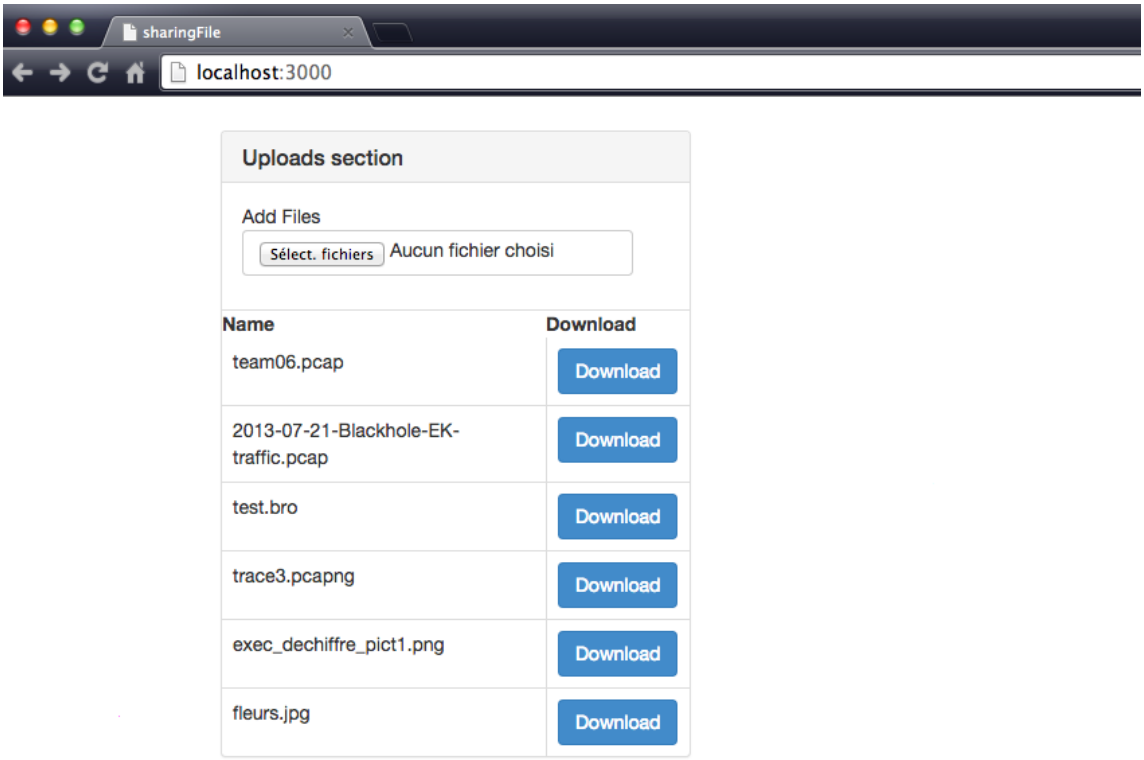
```
[[[[[ ~/sharingFile ]]]]]

=> Started proxy.
=> Meteor 1.1.0.2 is available. Update this project with 'meteor update'.
=> Started MongoDB.
=> Started your app.

=> App running at: http://localhost:3000/
```

Et voilà notre serveur est bien en marche et n'attend qu'à être exécuter via un browser!

Ajout des fichiers à partager:



3. Considérations sur la sécurité

Les éléments à considérer au niveau de la sécurisation de l'application sont les suivantes:

- Obliger les utilisateurs à être authentifier par un nom d'utilisateur et mot de passe pour pouvoir accéder aux ressources partagées.
    - Crypter les données partagés par une clé RSA/DSA ou autres types de hachages.
- socket.io non implémenté :
    - cette fonctionnalité permet de savoir les utilisateurs connectés.
- **Les routes** permettent de réinventer les répertoires en plus vous pouvez joindre le code pour le chemin "/ **de contact**", par exemple, qui ne permet qu'aux utilisateurs enregistrés de voir un formulaire, etc
- Faille de sécurité du script d'installation:

  La propriété des scripts du paquet NPM permet au propriétaire de spécifier des scripts qui devrait être exécuté à différents moments au cours de l'installation ou l'utilisation du module.

  Par exemple

```
{
  "name": "malicious",
  "version": "0.1.0",
  "dependencies": {},
  // Run the script file before installation
  "scripts" : { "preinstall" : "node script.js" },
  "engines": { "node": ">= 0.1" }
}
```

Figure 35 - Malicious Node.js package JSON

```
// Require filesystem library
var fs = require('fs');

// Write a file in the root users directory
fs.writeFile("/root/test", "Malicious", function(err) {
    fs.unlink("script.js"); // Delete this script
});
```

Figure 36 - Malicious payload attached to test module

Commentaire : aucune indication n'a été donnée à l'utilisateur sur le script écrit dans le répertoire privilégié, ce qui montre que, en principe, un tel script ne pouvait rien faire avec le système - créer un backdoor (porte dérobée), infecter ou quoi que l'attaquant souhaite accomplir.