Node 2

Toute documentation autorisée.

Consignes générales

Une partie de la note est basée sur la propreté du code rendu, selon les critères suivants :

- La consistence dans les choix de style de code (coding style).
- Une indentation correcte.
- Pas de "cochonnerie", comme du code mort (inutilisé) ou de l'ancien code en commentaire (*commented out*, les commentaires normaux sont bien entendu autorisés). Supprimez ce qui ne sert à rien, utilisez git à bon escient si vous souhaitez faire des tentatives.

Dans chacun des exercices, on vous donne le nom du fichier à créer pour l'exercice. Vous avez la liberté de créer ou non des fichiers supplémentaires si vous le souhaitez.

Rendu

Le rendu de votre travail s'effectuera sous la forme d'un repository git, par exemple sur Github. Envoyer un lien vers le repository à : eric@rixo.fr.

Les noms de chacun des participants du groupe doivent apparaître dans les commits git (ou un pseudo, mais il me faut la correspondance nom <=> pseudo).

Seul le travail sur la branche master sera pris en compte (y compris pour les commits, et donc les noms des gens). Pensez à merger vos contributions dans master régulièrement (conseil : n'attendez surtout pas la dernière minute pour tout merger).

1. Mise en route

À partir d'un repository git vierge, initialiser un projet Node.js en installant les dépendances suivantes pour le développement :

- nodemon

Installer également les packages suivants à partir de la registry npm :

- `node-fetch`
- `fs-extra`

Commit.

2. Time server

Dans le fichier `time-server.js` :

- Créez un serveur HTTP qui écoutera sur `localhost:4000`.
- Le serveur doit envoyer l'heure courante aux requêtes GET sur '/', au format suivant `00:00:00` (heures, minutes, secondes).
- Cette URL doit être accessible aux formats suivants selon les spécifications de la requête :
    - simple string
    - JSON (Attention, une simple string n'est pas du JSON valide -- vous devrez wrapper ça dans un objet)
- Si on accède à l'URL `http://localhost:4000` avec un navigateur, cela doit afficher l'heure courante (au moment du chargement de la page) en bleu.

3. Secret Server

Dans le fichier `secret-server.js` :

- Créez un serveur HTTP qui écoutera sur `localhost:4001`.
- Sur l'URL `/secret`, le serveur doit renvoyer le mot secret.
- Le mot secret doit être stocké dans le fichier `data/secret.txt` du projet, mais **attention** : le contenu de ce fichier doit être encrypter.
    - Pour encrypter le fichier, vous pouvez par exemple inverser l'ordre des lettres. Vous pouvez également trouver une autre solution mais (1) elle doit être au moins aussi puissante que celle proposée et (2) votre programme doit être capable de la décrypter !
    - Vous devez utiliser le module `fs-extra` pour toutes les opérations sur le filesystem.
- Le serveur doit également pouvoir accepter une requête pour modifier le mot secret actuel.

4. Secret History Server

Dans le fichier `sechisrv.js` :

- Créez un serveur HTTP qui écoutera sur `localhost:4002`.
- Toutes les 5s, ce serveur doit effectuer **en parallèle** (1) une requête sur l'URL `http://localhost:4000/` et (2) une requête sur l'URL `http://localhost:4001/secret`.
  - Cf. fonction JS `setInterval`
- Le programme doit garder en mémoire les résultats des 10 dernières paires de requests.
- Le serveur doit pouvoir envoyer les `n` dernières paires de résultats au format JSON lors des requêtes sur les URLs `/1`, `/2`, etc., où `n` est le chemin de l'URL.
- Si n est inférieur à 1, supérieur à 10, ou non numérique, le serveur doit répondre une erreur. Votre erreur doit respecter la sémantique HTTP.
- La réponse (de succès) doit respecter le format suivant :

```
[
  {time: '00:00:00', secret: 'abc'},
  ...
]
```

- Les paires de résultats doivent être ordonnées par ordre antéchronologique (plus récentes en premier) dans la réponse.
- Si un des serveurs retourne une erreur à une requête donnée, enregistrez "*ERROR*" à la place du résultat.
- Vous devez utiliser `node-fetch` pour effectuer les requêtes HTTP sortantes.

5. App Server

- Créez un serveur HTTP qui servira des fichiers statiques.

6. Servers App

- Créez une application Vue.js accessible sur l'URL racine (`/`) du serveur de l'exercice 5.
- L'application ne doit jamais provoquer un rechargement de la page du navigateur (attention si vous mettez un

).
- L'application doit récupérer toutes les secondes les résultats des appels aux serveurs des exercices 2, 3 et 4.
  - Cf. fonction JS `setInterval`
- L'application doit conserver en mémoire les 100 derniers résultats de ces serveurs, ainsi que la date (et heure) à laquelle la requête est partie du client.
  - L'application client n'est pas obligée de contacter directement les autres serveurs. Si vous envoyez une requête vers un autre domaine (y compris un autre port sur le même domaine, vous allez devoir configurer tous vos serveurs pour accepter les requêtes CORS…).
  - Si un serveur renvoie une erreur à une requête donnée ou bien n'est pas en ligne pour répondre à une requête donnée, enregistrez "DOWN" dans la liste de résultats.
- On doit pouvoir choisir un des 3 serveurs dans une liste et l'application affiche uniquement les résultats pour le serveur sélectionné (mais garde les autres en mémoire).
- Pour chaque résultat, afficher également le *temps relatif* (e.g. "Il y a 3s", ou format équivalent) depuis le départ de la requête en question de l'application client.
- On doit pouvoir trier les résultats du plus récent au plus ancien, ou bien l'inverse.
- Les résultats doivent être présentés par pages de 10 résultats (et on doit bien entendu pouvoir changer de page).
- Sur la même page, avec Vue.js (mais pas nécessairement la même instance Vue -- i.e. vous pouvez avoir 2 `new Vue(...)` -- si ça vous arrange pour collaborer), on doit trouver un

formulaire qui permet de changer le secret du serveur de l'exercice 3.

- Là encore, CORS n'est pas une obligation. (La requête utilisée doit marcher pour changer le mot secret par contre !)