TP 2: Création d'un Serveur HTTP de Streaming Vidéo

Remise: 11 Novembre avant minuit sous Léa

Objectif : L'objectif de cet exercice est de créer un serveur HTTP simple en Node.js qui permet de diffuser une vidéo en répondant aux requêtes des clients et en gérant les réponses correctement.

1. **Initialiser le serveur :**
   - Créez un fichier `server.js` en utilisant le module `http`. Le serveur doit écouter sur le port 8080 et être prêt à traiter les requêtes.
2. **Fonctions Utilisées :**
   - `http.createServer(callback)`: Crée un nouveau serveur HTTP qui exécute la fonction de rappel chaque fois qu'une requête est reçue.
   - `server.listen(port, callback)`: Fait en sorte que le serveur écoute sur le port spécifié. Le deuxième argument est une fonction de rappel qui s'exécute lorsque le serveur est prêt.
3. **Définir le type de contenu :**
   - Utilisez `setHeader` pour définir le type de contenu de la réponse comme `video/mp4`, indiquant que le contenu renvoyé est une vidéo.
4. **Fonctions Utilisées :**
   - `response.writeHead(statusCode, headers)`: Définit le code de statut HTTP et les en-têtes de la réponse. Par exemple, un code de statut 200 pour indiquer que la requête a été traitée avec succès.
5. **Gérer les requêtes :**
   - Lorsque le serveur reçoit une requête, utilisez `fs.statSync` pour obtenir la taille du fichier vidéo afin de préparer le flux.
6. **Fonctions Utilisées :**
   - `fs.statSync(path)`: Renvoie les statistiques d'un fichier, y compris sa taille. Utilisé pour obtenir la taille du fichier vidéo.
   - `path.join(...)`: Combine plusieurs segments de chemin en un seul chemin pour gérer les chemins de manière sécurisée.
7. **Créer un flux de lecture :**
   - Utilisez `fs.createReadStream` pour créer un flux de lecture à partir du fichier vidéo. Ce flux permet d'envoyer le contenu du fichier en continu.
8. **Fonctions Utilisées :**

- `fs.createReadStream(path)`: Crée un flux de lecture à partir du fichier spécifié. Ce flux peut être pipé directement dans la réponse HTTP pour envoyer les données au client.

9. **Gérer les événements du flux :**
   - Gérez les événements du flux pour contrôler le comportement lors de l'ouverture du flux et lors de la gestion des erreurs.
10. **Fonctions Utilisées :**
    - `stream.on(event, callback)`: Écoute un événement sur le flux, tel que `'open'` ou `'error'`, et exécute la fonction de rappel lorsque l'événement se produit.
    - `stream.pipe(destination)`: Connecte le flux de lecture à un flux d'écriture (dans ce cas, la réponse HTTP), permettant l'envoi direct des données au client.
11. **Envoyer la réponse :**
    - Vous n'avez pas besoin d'appeler `response.end()` explicitement ici, car `pipe()` termine automatiquement la réponse lorsque le flux est complet.
12. **Lancer le serveur :**
    - Faites en sorte que le serveur écoute sur le port 8080 et affiche un message dans la console pour confirmer son démarrage.

Barème de Notation (Total : 100 points)

1. **Initialisation du serveur (20 points)**
   - Création du fichier `server.js` (5 points)
   - Utilisation correcte du module `http` pour créer le serveur (10 points)
   - Le serveur écoute sur le port 8080 et est prêt à traiter les requêtes (5 points)
2. **Définition du type de contenu (15 points)**
   - Utilisation de `setHeader` pour définir le type de contenu comme `video/mp4` (10 points)
   - Envoi d'un code de statut HTTP approprié (5 points)
3. **Gestion des requêtes (20 points)**
   - Utilisation de `fs.statSync` pour obtenir la taille du fichier vidéo (10 points)
   - Utilisation de `path.join` pour gérer les chemins de manière sécurisée (10 points)
4. **Création d'un flux de lecture (20 points)**
   - Utilisation de `fs.createReadStream` pour créer le flux de lecture (10 points)
   - Intégration correcte du flux dans la réponse HTTP (10 points)
5. **Gestion des événements du flux (15 points)**
   - Écoute des événements du flux (`'open'`, `'error'`, etc.) (10 points)
   - Utilisation de `stream.pipe` pour envoyer les données au client (5 points)
6. **Finalisation et exécution (10 points)**

- L'appel de `response.end()` n'est pas nécessaire, et le comportement du serveur est vérifié (5 points)
- Affichage d'un message de confirmation dans la console lors du démarrage du serveur (5 points)