TP : Initiation à Node.js - Création d'une API REST

Objectifs pédagogiques

À la fin de ce TP, vous serez capable de :

- Comprendre et utiliser le système de modules Node.js (require/module.exports)
- Manipuler des fichiers avec le module `fs`
- Effectuer des requêtes HTTP avec le module `https`
- Créer un serveur web avec le module `http`
- Développer une API REST simple
- Consommer une API depuis une page web

Prérequis

- Node.js installé (version 14+)
- Un éditeur de code (VS Code recommandé)
- Extension REST Client pour VS Code

Structure du projet

```
tp-nodejs-api/
├── seed.js           # Script pour récupérer les données
├── posts.json        # Fichier de données (généré)
├── server.js         # Serveur API REST
├── postManager.js    # Module de gestion des posts
├── client/
│   └── index.html    # Interface web
└── package.json      # Configuration du projet
```

Partie 1 : Initialisation du projet

Étape 1.1 : Créer le projet

```
mkdir tp-nodejs-api
cd tp-nodejs-api
npm init -y
```

Étape 1.2 : Créer la structure des dossiers

Partie 2 : Script de récupération des données (seed.js)

Objectif

Créer un script qui récupère des posts depuis l'API JSONPlaceholder et les sauvegarde localement.
URL de l'API : https://jsonplaceholder.typicode.com/posts

Étape 2.1 : Créer le fichier seed.js

```
// seed.js
const https = require('https');
const fs = require('fs');

// URL de l'API
const API_URL = 'https://jsonplaceholder.typicode.com/posts';

// Fonction pour récupérer les posts
function fetchPosts() {
    // Acompléter
}

// Fonction pour sauvegarder les posts
function savePosts(posts) {
    // Acompléter
}

// Exécution du script
fetchPosts();
```

Étape 2.2 : Tester le script

```
node seed.js
```

Vérifiez que le fichier `posts.json` a été créé avec les données.

Partie 3 : Module de gestion des posts (postManager.js)

Objectif

Créer un module réutilisable pour gérer les opérations CRUD sur les posts.

Étape 3.1 : Créer le module postManager.js

```
// postManager.js
const fs = require('fs');
const path = require('path');

const POSTS_FILE = path.join(__dirname, 'posts.json');

// Lire tous les posts
function getAllPosts(callback) {
    fs.readFile(POSTS_FILE, 'utf8', (err, data) => {
        if (err) {
            // A Compléter
        }
        try {
            // A Compléter
        } catch (parseError) {
            // A Compléter
        }
    });
}

// Obtenir un post par ID
function getPost(id, callback) {
    // A Compléter
}

// Insérer un nouveau post
function insertPost(newPost, callback) {
    // A Compléter
}

// Mettre à jour un post
function updatePost(id, updatedPost, callback) {
    // A Compléter
}

// Supprimer un post
function deletePost(id, callback) {
    // A Compléter
}

// Fonction utilitaire pour sauvegarder les posts
function savePosts(posts, callback) {
    // A Compléter
}
```

```
// Exporter les fonctions
module.exports = {
    getAllPosts,
    getPost,
    insertPost,
    updatePost,
    deletePost
};
```

Partie 4 : Serveur API REST (server.js)

Objectif

Créer un serveur HTTP qui expose une API REST utilisant le module postManager.

Étape 4.1 : Créer le serveur

```
// server.js
const http = require('http');
const url = require('url');
const postManager = require('./postManager');

const PORT = 3000;

// Fonction pour parser le body JSON
function parseJsonBody(request, callback) {
    let body = '';
    request.on('data', chunk => {
        body += chunk.toString();
    });
    request.on('end', () => {
        try {
            const data = JSON.parse(body);
            callback(null, data);
        } catch (error) {
            callback(error, null);
        }
    });
}

// Créer le serveur en testant sur la méthode HTTP methid et le pathname
const server = http.createServer((request, response) => {
    const parsedUrl = url.parse(request.url, true);
    const pathname = parsedUrl.pathname;
    const method = request.method;

    // Headers CORS pour permettre l'accès depuis le client web
    response.setHeader('Access-Control-Allow-Origin', '*');
    response.setHeader('Access-Control-Allow-Methods', 'GET, POST, PUT, DELETE');
    response.setHeader('Access-Control-Allow-Headers', 'Content-Type');
    response.setHeader('Content-Type', 'application/json');

    // Routes API

    // GET /api/posts - Obtenir tous les posts
    if (pathname === '/api/posts' && method === 'GET') {
        postManager.getAllPosts((err, posts) => {
            if (err) {
                response.writeHead(500);
                response.end(JSON.stringify({ error: 'Erreur serveur' }));
            } else {
```

```
                response.writeHead(200);
                response.end(JSON.stringify(posts));
            }
        });
    }

    // GET /api/posts/:id - Obtenir un post par ID


    // POST /api/posts - Créer un nouveau post


    // PUT /api/posts/:id - Mettre à jour un post


    // DELETE /api/posts/:id - Supprimer un post


    // Route non trouvée
    else {
        response.writeHead(404);
        response.end(JSON.stringify({ error: 'Route non trouvée' }));
    }
});

// Démarrer le serveur
server.listen(PORT, () => {
    console.log(`Serveur API démarré sur http://localhost:${PORT}`);
    console.log('Routes disponibles:');
    console.log('  GET    /api/posts');
    console.log('  GET    /api/posts/:id');
    console.log('  POST   /api/posts');
    console.log('  PUT    /api/posts/:id');
    console.log('  DELETE /api/posts/:id');
});
```

Partie 5 : Tests avec REST Client

Étape 5.1 : Créer un fichier de tests

Créez un fichier `api-tests.http` pour tester votre API avec l'extension REST Client de VS Code :

```
### Obtenir tous les posts
GET http://localhost:3000/api/posts

### Obtenir un post spécifique
GET http://localhost:3000/api/posts/1

### Créer un nouveau post
POST http://localhost:3000/api/posts
Content-Type: application/json

{
    "userId": 1,
    "title": "Mon nouveau post",
    "body": "Ceci est le contenu de mon nouveau post créé via l'API"
}

### Mettre à jour un post
PUT http://localhost:3000/api/posts/1
Content-Type: application/json

{
    "userId": 1,
    "title": "Post modifié",
    "body": "Le contenu a été mis à jour"
}

### Supprimer un post
DELETE http://localhost:3000/api/posts/101
```

Étape 5.2 : Tester l'API

1. Démarrez le serveur : `node server.js`
2. Ouvrez le fichier `api-tests.http` dans VS Code
3. Cliquez sur "Send Request" pour chaque test

Partie 6 : Client Web (client/index.html)

Objectif

Créer une interface web pour consommer l'API REST.

Étape 6.1 : Créer l'interface HTML