Institut National des Postes et Télécommunications

Module : Communication P2P en temps réel sur le web

Rapport de TP2 : La Plateforme NodeJS

Développement d'une application temps réel avec la stack MEAN

Réalisé par :
Aymane ASKRI

Encadré par :
Asmae ELHAMZAOUI

Année Universitaire : 2025-2026

Table des matières

Introduction

L'objectif de ce TP est de maîtriser les concepts du développement web moderne axé sur la performance et le temps réel, en utilisant la plateforme **Node.js**. Contrairement aux modèles bloquants traditionnels (comme vu précédemment avec les Servlets Java), Node.js utilise un modèle d'entrées/sorties non bloquant piloté par les événements.

Dans ce travail pratique, nous avons développé une application web complète permettant l'authentification, l'upload de fichiers, l'exploration du système de fichiers serveur et la communication temps réel.

Pour répondre aux exigences professionnelles actuelles et aux extensions demandées, nous avons directement adopté une architecture modulaire basée sur des frameworks robustes : **Express** pour le routage, **Mongoose** pour la base de données, et **Socket.io** pour le temps réel.

Chapitre 1

Environnement et Architecture

1.1 La Plateforme Node.js

Node.js est un environnement d'exécution JavaScript construit sur le moteur V8 de Chrome. Il permet d'utiliser JavaScript côté serveur. Son gestionnaire de paquets, **NPM** (Node Package Manager), nous a permis d'installer les dépendances nécessaires à notre projet modulaire.

1.2 Modules Utilisés (Dépendances)

Pour éviter de réinventer la roue et pour assurer la maintenabilité du code (Extension 4 et 6), nous avons utilisé les bibliothèques suivantes :

- **express** : Framework web minimaliste pour gérer les routes HTTP et les middlewares.
- **mongoose** : Bibliothèque de modélisation d'objets (ODM) pour MongoDB.
- **socket.io** : Bibliothèque permettant la communication bidirectionnelle temps réel (WebSockets).
- **multer** : Middleware pour la gestion des données `multipart/form-data`, utilisé ici pour l'upload de fichiers.
- **cors** : Pour gérer les autorisations d'accès entre différentes origines (Cross-Origin Resource Sharing).

Chapitre 2

Implémentation de la Solution

2.1 Configuration du Serveur et Base de Données

Nous avons mis en place un serveur HTTP unique qui délègue le routage à Express et la gestion temps réel à Socket.io. La connexion à la base de données NoSQL MongoDB est assurée par Mongoose.

```
const mongoose = require('mongoose');

async function connectDB() {
    try {
        await
mongoose.connect('mongodb://localhost:27017/tp_node_db');
        console.log("MongoDB connecté");
    } catch (err) {
        console.error("Erreur MongoDB :", err);
        process.exit(1);
    }
}

module.exports = connectDB;
```

FIGURE 2.1 – Initialisation de MongoDB et définition du Schema User

Le schéma de données `UserSchema` définit la structure des documents utilisateurs (login, password, nom, prénom, dernière image uploadée).

2.2 Gestion du Routage (Express)

Au lieu d'utiliser les modules natifs `http` et `url` qui nécessitent un parsing manuel complexe des URLs, nous avons intégré le Framework **Express** (Extension 4). Cela permet de définir des routes claires pour chaque verbe HTTP (GET, POST).

2.2.1 Service d'Authentification (Login)

Le service `/login` vérifie l'existence de l'utilisateur dans MongoDB. Si l'utilisateur n'existe pas, il est créé à la volée (simplification pour le TP).

2.2.2 Service Upload et Show

Le service d'upload utilise `Multer` pour stocker l'image sur le disque serveur et met à jour le champ `lastImage` de l'utilisateur en base de données.

FIGURE 2.2 – Implémentation des routes Express (SystemRoutes.js)

```
// In uploadRoutes.js
const express = require('express');
const multer = require('multer');

const uploadController =
require('../controllers/uploadController');

const router = express.Router();
const upload = multer({ dest: 'uploads/' });

router.post('/upload', upload.single('image'),
uploadController.uploadImage);
router.get('/show/:login', uploadController.showImage);

module.exports = router;
```

FIGURE 2.3 – Implémentation des routes Express (UploadRoutes.js)

```
// In userRoutes.js
const express = require('express');
const router = express.Router();
const userController =
require('../controllers/userController');

router.post('/login', userController.login);

module.exports = router;
```

FIGURE 2.4 – Implémentation des routes Express (userRoutes.js)

2.3 Communication Temps Réel (Socket.io)

Pour l'extension 2, nous avons migré une partie de la communication vers **Socket.io**. Contrairement aux requêtes HTTP classiques (Request-Response), le WebSocket permet au serveur de pousser ("push") des données vers le client instantanément.

```
io.on('connection', (socket) => {
    console.log('Client connecté via WebSocket');

    socket.on('chat_message', (msg) => {
        // Broadcast du message a tous les clients connectes
        io.emit('chat_message', msg);
    });
});
```

Listing 2.1 – Logique Serveur Socket.io

Chapitre 3

Extensions et Conception (UML & MEAN)

3.1 Architecture MEAN Stack

L'extension 3 demande l'architecture d'une application MEAN. C'est une solution "Full JavaScript" composée de :

- **M**ongoDB : Base de données NoSQL (Stockage JSON).
- **E**xpress : Framework Backend (API REST).
- **A**ngular : Framework Frontend (Single Page Application).
- **N**ode.js : Serveur d'application.

[Image of MEAN stack architecture diagram]

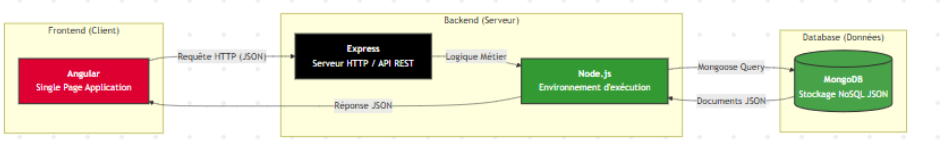

FIGURE 3.1 – Architecture MEAN Stack

3.2 Diagrammes UML

3.2.1 Diagramme de Cas d'Utilisation

L'utilisateur interagit avec le système pour s'authentifier, uploader des fichiers et visualiser le contenu.

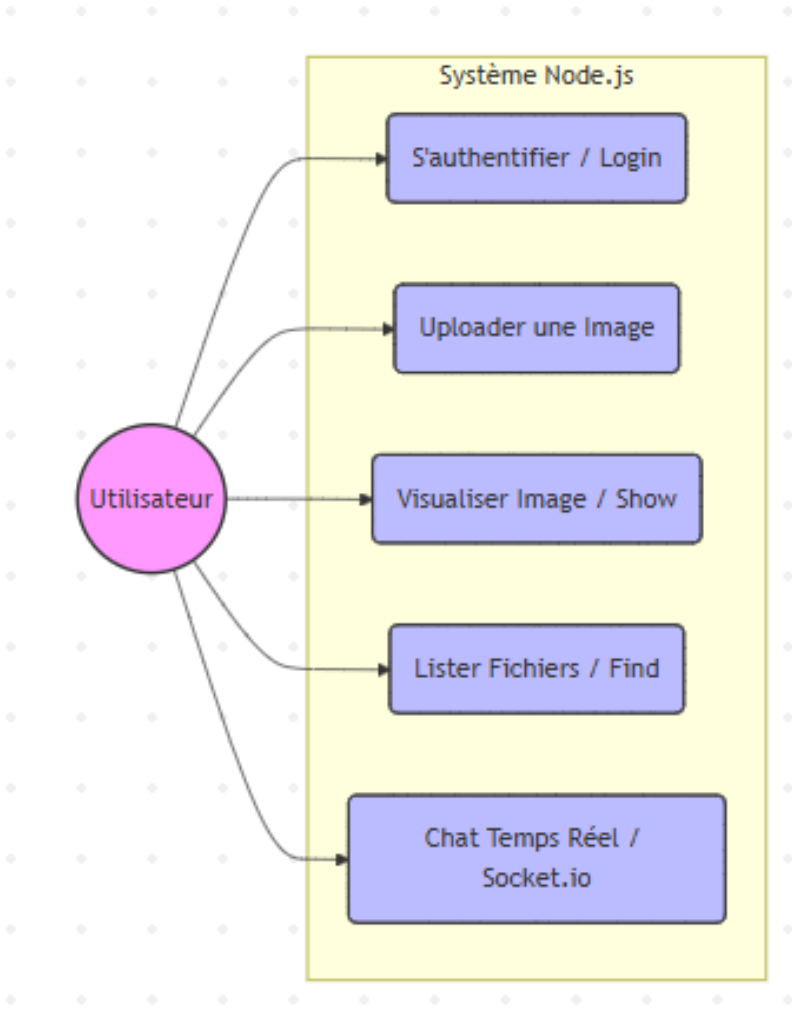

FIGURE 3.2 – Diagramme de Cas d'Utilisation

3.2.2 Diagramme de Séquence (Upload)

Le diagramme suivant illustre le flux de données lors d'un upload d'image, impliquant le Client, le Serveur (Express) et la Base de Données (MongoDB).

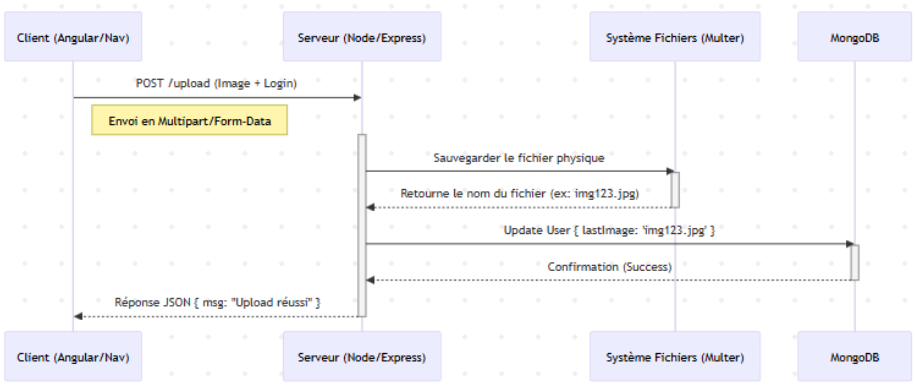

FIGURE 3.3 – Diagramme de Séquence : Scénario Upload

Chapitre 4

Synthèse des Acquis

Ce TP a permis de consolider notre compréhension des architectures web asynchrones. Voici les points clés retenus :

- **Programmation Non-Bloquante :** L'avantage majeur de Node.js est sa capacité à gérer des milliers de connexions simultanées grâce à l'Event Loop, sans créer un thread par utilisateur (contrairement aux modèles classiques).
- **Full Stack JavaScript :** L'utilisation du même langage (JS) pour le frontend, le backend et la manipulation de base de données (objets JSON dans MongoDB) simplifie énormément le développement et réduit la charge cognitive ("Context Switching").
- **WebSockets vs HTTP :** Nous avons distingué le rôle de HTTP (chargement de ressources, actions statiques) de celui des WebSockets (chats, notifications temps réel), comprenant que les applications modernes sont souvent hybrides.
- **Frameworks Modernes :** L'intégration d'Express et Mongoose a démontré comment l'écosystème Node.js permet de développer rapidement des APIs RESTful robustes et maintenables.