المعهد الوطني للبريد والمواصلات
ⵍⵎⵄⵀⴰⴷ ⴰⵡⴰⵟⵏⵉ ⵏ ⵜⵓⵙⵙⵓⵔⵜ ⴰⴷ ⵜⵙⵙⵉⵡⵍ
Institut National des Postes et Télécommunications

Compte rendu

SUD341 : Systèmes temps réel et embarqués

(Le temps réel sur le web : NodeJs)

Realisé par :
HAJJAJI Ayyoub
ELAMMARI Souhail
El MEKKAOUI Fayssal

Encadré par :
EN-NOUAARY Abdeslam

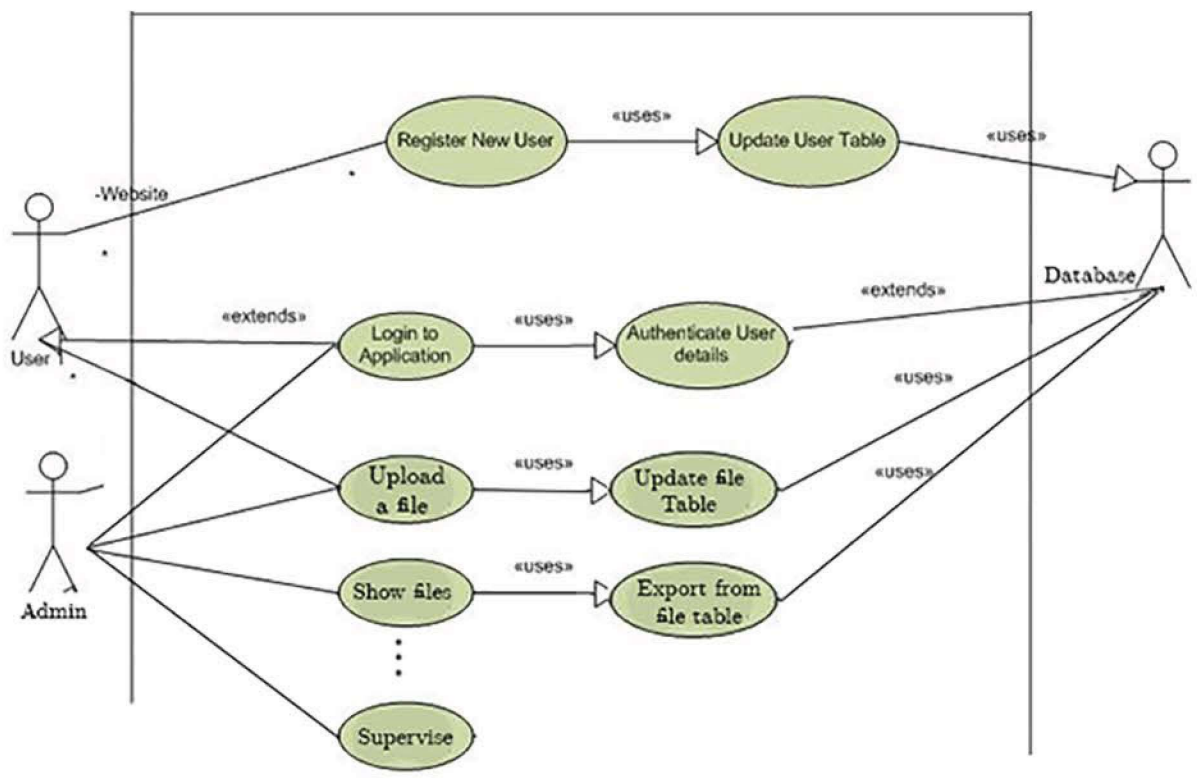
express socket.io React mongoDB

16 Octobre 2020

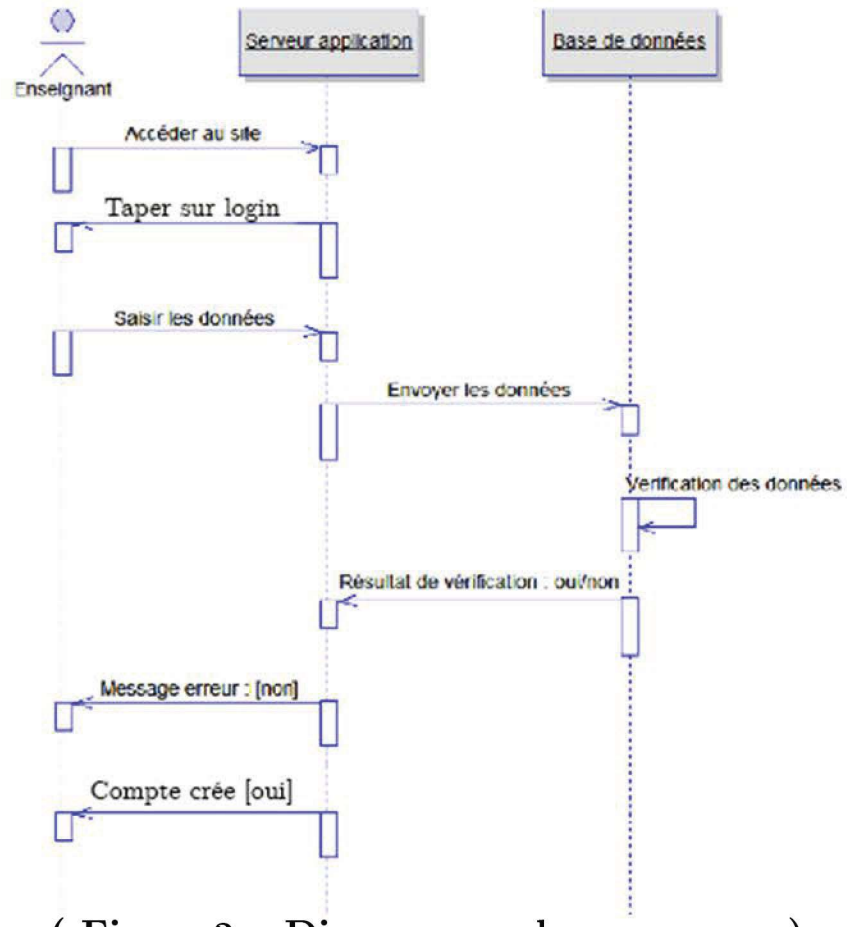
Introduction .

L'objectif de ce TP est de mettre en pratique certains des concepts discutés dans le cours et ce à travers le développement d'une application temps réel sur web en se basant sur les technologies HTML, HTTP et NODEJS.

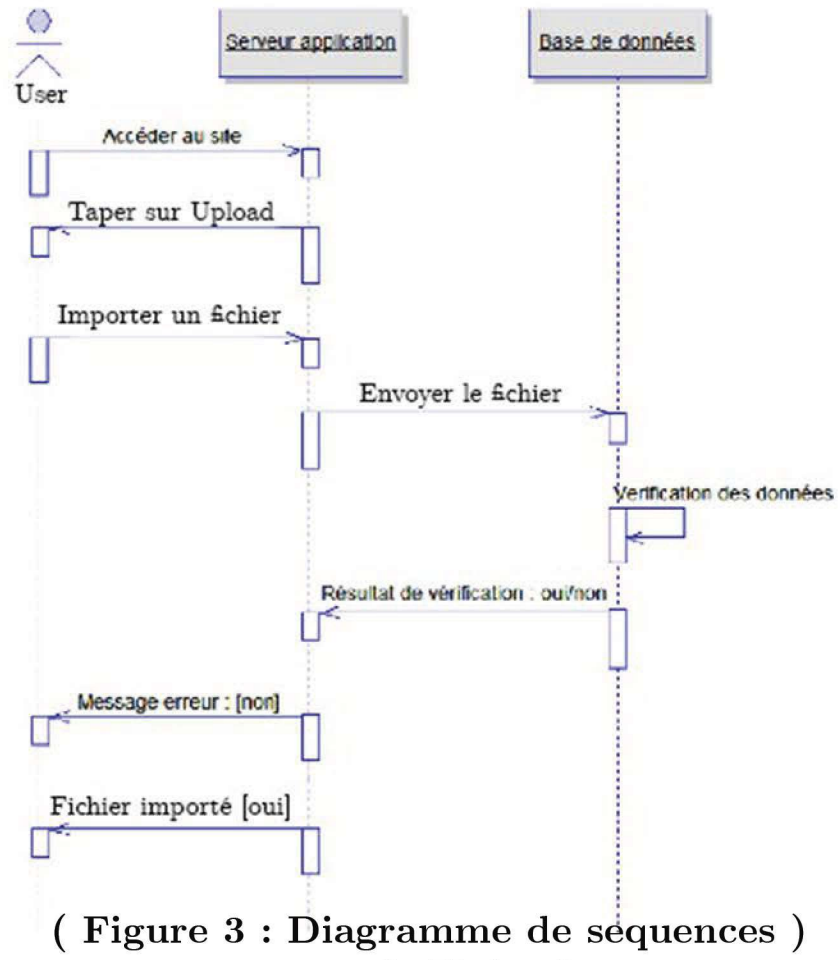
Le but de l'application est de permettre une interaction entre le client et le serveur, pour cela nous allons créer une interface dans laquelle l'utilisateur peut se registrer et s'authentifier, en insèrant son nom, prenom, mot de passe... et puis commander quelques implémentations (Upload, Show ...).
La conception UML suivante résume en quelques sorts la logique de notre application.



(Figure 1 : Diagramme Use Case)



(Figure 2 : Diagramme de sequences)
partie Login



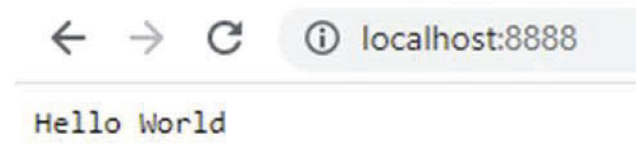
(Figure 3 : Diagramme de sequences)
partie Upload

1er Partie : Implémentation de la solution

Dans cette partie du TP nous allons exécuter une suite des implémentation, en discutant les imperfections de chaque version, afin d'aboutir à une version qui répond aux besoins de l'utilisateur.

1er Version :

```
C:\Users\fayssal2\Desktop>node index.js
```



Cette première version de l'application affiche le message ¨ Hello world ¨¨ mais on ne recoit aucune information dans la console .

2ème version :

```
C:\Users\fayssal2\Desktop>node index.js
Server has started.
Request received.
Request received.
```

Cette Version nous renvoie la demande du client pour acceder au site mais sans savoir l'url composé

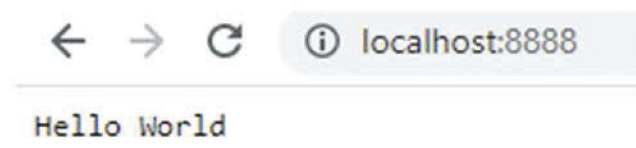
3ème version :

```
C:\Users\fayssal2\Desktop>node index.js
Server has started.
Request for / received.
About to route a request for /
```

Le problème précédent est fixé ,on est maintenat capable de savoir le ¨Pathname¨ tappé par le client

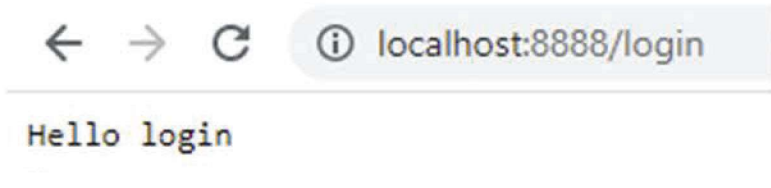
4ème Version :

```
C:\Users\fayssal2\Desktop>node index.js
Server has started.
Request for / received.
About to route a request for /
Request handler 'start' was called.
Request for /favicon.ico received.
```



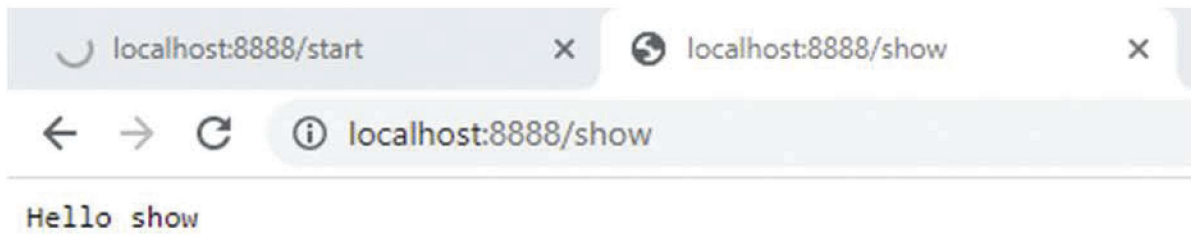
Dans cette version le client peut accéder au plusieurs pages mais il affiche toujours le meme message ¨ Hello world ¨.

5ème version :



Dans cette version le client peut voir des messages diffèrents dans chaque page , mais la version ne le permet pas d'exécuter plusieurs service en **même temps .**

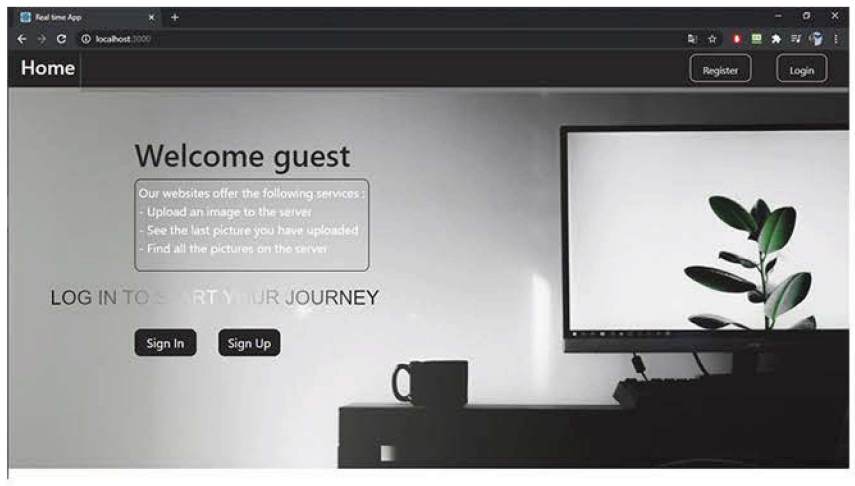
6ème version :



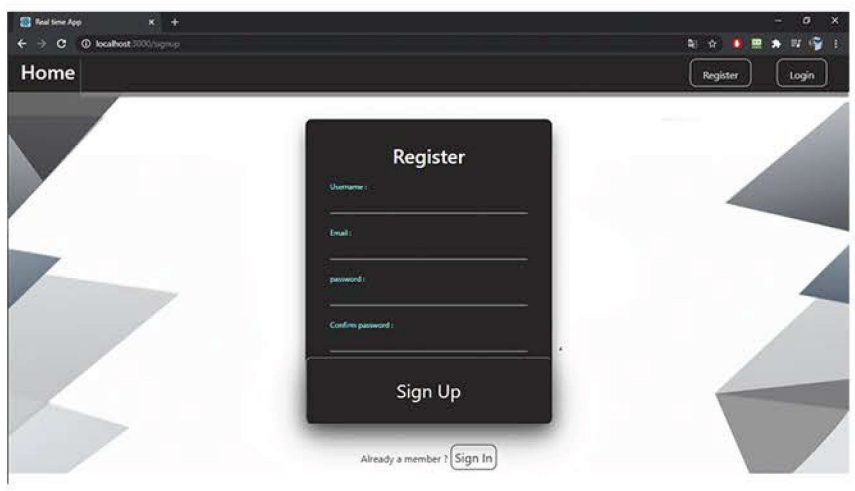
La version Finale à soulever le problème précédent, le client peut maintenant demander plusieurs services en même temps.

2éme Partie : Extensions .

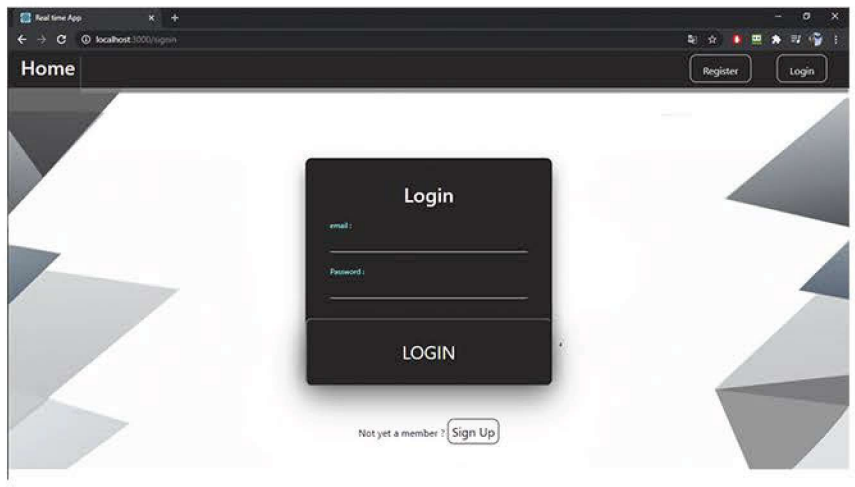
Dans cette partie du TP nous allons créer un site web en utilisant Nodejs , mongodb, express et socket.io .en complètant le code précédent par la logique-métier de chacun des services .



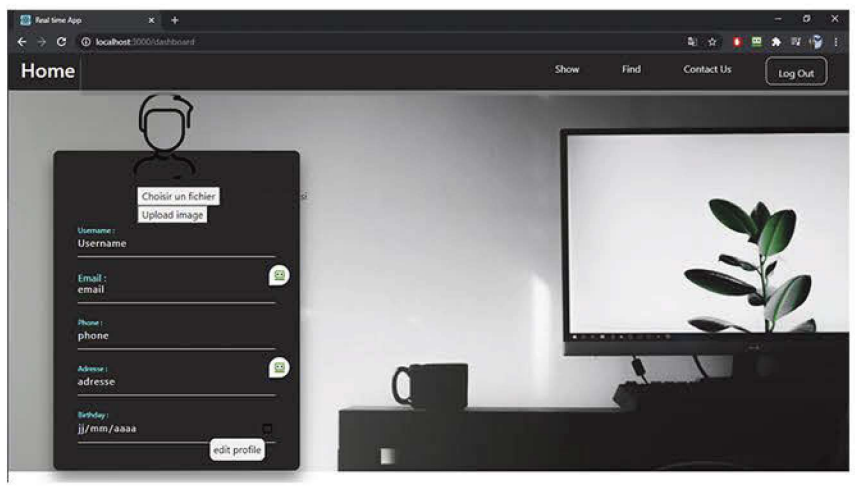
Url **: http://localhot:3000/**
Cette page représente la service ./start accessible par tout le monde, elle affiche un message d'accueil avec l'ensemble des services disponibles, et demande de l'utilisateur de s'authentifier pour tester ses services.



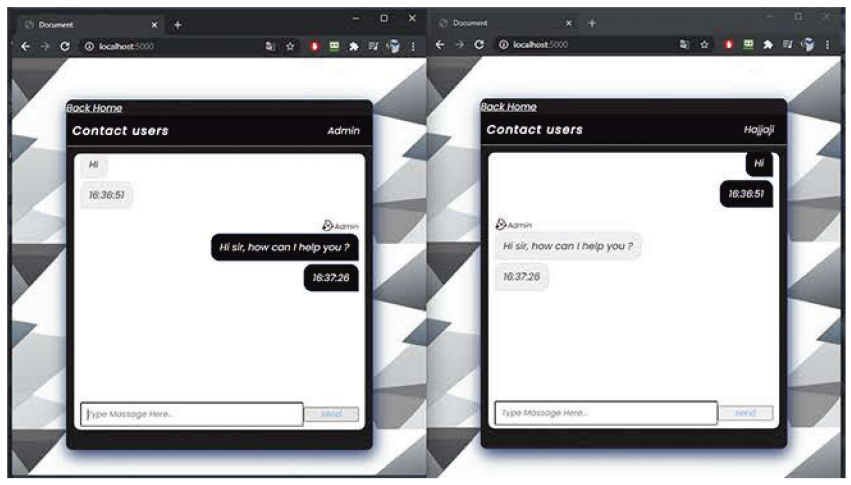
Url : **http://Localhost:3000/register**
Dans cette page l'utilisateur peut créer un compte afin d'accéder aux autres services après l'authentification .
Les informations reçus seromt stocker dans La base de données **MongoDB .**



Url : **http://localhost:3000/login**
Cette page représente le service **./login** qui permet au client de s'authentifier en comparant ses informations avec la base de données .



Url : **http://localhost:3000/dashboard**
Cette page représente le profil de l'utilisateur, elle lui permet de voir ses information stocker dans la base de données, ajouter des informations ou bien les modifier, en plus il peut choisir une image pour son profil et la sauvegarder sur le disque du serveur, ce qui représente le service Upload



Url : **http://localhost:3000/contact**

Si l'utilisateur a une réclamation, ce service lui permet d'ouvrir une conversation en temps réel avec l'administration du site.

Modules utilisés :

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

const signUpTemplate = new mongoose.Schema({
    userName:{
        type: String,
        required:true,
        unique: 1
    },
    email:{
        type: String,
        required:true,
        unique:1
```

MongoDB
On utilisant le framework Mongoose, on a pu créer une table (Schema) qui représente les informations qu'on veut collecter de l'utilisateur, puis on a utilisé la commande mongoose.connect() pour se connecter à notre base de données MongoDB, et finallement on utilisant les méthodes post et get on a pu echanger de l'information avec la base de donné

```socket
const socket = io();
const msgText = document.querySelector('#msg')
const btnSend = document.querySelector('#btn-send')
const chatBox = document.querySelector('.chat-content')
const displayMsg = document.querySelector('.message')

let name;
do{
    name = prompt('What is your name ?')
}while(!name)

document.querySelector('#your-name').textContent = name
msgText.focus()

btnSend.addEventListener('click', (e)=>{
    e.preventDefault
    sendMsg(msgText.value)
    msgText.value = '';
    msgText.focus();
    chatBox.scrollTop = chatBox.scrollHeight;
```

Socket.io
Socket.io est un module de Node. js qui permet de créer des Web Sockets, c'est-à-dire des connections bi-directionnelles entre clients et serveur, cette module nous a aidé à crée la page de chat.

```react
onSubmit(event){
    event.preventDefault()

    const registred = {
        userName: this.state.userName,
        email: this.state.email,
        password: this.state.password,
        confirmPassword: this.state.confirmPassword,

    }

    axios.post('http://localhost:4000/app/signup', registred)
    .then(response => console.log(response.data))
    window.location = '/dashboard'
}
render() {
  return (
    <div>
        <img className="home__image" src={require('./backgroun.jpeg')} alt="" />
    <div className="box">
        <h2>Register</h2>
        <form method="post" onSubmit={this.onSubmit}>
        <div className="inputbox">
                <input type="text"  required="" value={this.state.userName} onChange={this.changeUserName} />
                <label for="name">  Username :</label>
```

React
Le but principal de cette bibliothèque est de faciliter la création d'application web, via la création de composants dépendant d'un état et générant une page (ou portion) HTML à chaque changement d'état.
On l'a utilisé pour représenter la partie frontend du site, et pour prendre les informations de l'utilisateur, et on utilisant le package axios on a pu les envoyer au backend pour les stocker en base de donnée.

```express
app.use(fileUpload())
app.use(express.json())
app.use(cors())
app.use(session({secret:"jkhfjdsgf51fgh1h15hr5fd1ghr", resave
app.use('/app', routesUrls)
app.listen(4000, () => console.log("server is up and runing")
```

Express
C'est un framework pour construire des applications web basées sur Node.js2. C'est de fait le framework standard pour le développement de serveur en Node.js, et on a eu l'occasion pour l'intégrer à notre application dans le but d'implémenter l'aspect routage.

Visiter ce lien pour voir le site web à son état de fonctionnement :
https://drive.google.com/file/d/1IFgw6qufKc9o9rw7-USl7oM9IGffT2A5/view?usp=sharing

Pour voir le code complet visitez ce lien :
https://github.com/ayy-oub/tp1-NodeJS

Récapitulatif :

Tous les navigateurs sont équipés d'un moteur javascript qui permet de traduire le code javascript en code machine, la brillante idée derrière Node.-js c'est d'utiliser ce moteur (v8) en dehors du navigateur.

Dans ce TP on a pu constater que le Node.js est un système 'Non Block-ing', c'est-à-dire, si un client fait une requête de fichier au serveur, alors ce dernier lance cette requête mais il n'attende pas son résultat, et si un autre client fait une autre requête, alors le serveur est capable de la traiter également, ce qui rend Node.js super rapide pour les applications qui font énormément de requêtes de fichiers parce que node.js est capable de gérer toutes ces requêtes en parallèle, c'est grâce a cela qu'il est particulièrement bien adapté aux RTA (Real Time Applications) parce que les RTA elles font énormément de requêtes pour, sans cesses, mettre à jour les données de l'application.

On a constaté également que Node.js est un système 'Flexible', c'est l'util-isateur qui choisit quels sont les modules qu'il veut lui greffer pour l'utilis-er, ce qui vous offre une marge de manœuvre pour avancer petit à petit dans la direction que vous pensez être la meilleure au fur et à mesure que vous avancez dans votre projet, et pour ce faire, vous disposez d'un écosys-tème très large de librairie 'open source' grâce au 'npm' : Node Package Manager. Et en termes d'organisation et de facilité, Node.js vous offre la possibilité de garder le même langage (javascript) pour le Front end comme pour le Back end, ce qu'est plus pratique.

Limites et perspectives :

Node.js est Single-thread, alors que la plus part des systèmes classiques sont multithreads, ce qui pourrais être un incontinent pour certains pro-jets qui demandent des taches longes à traité du coté serveur, des taches qui demande beaucoup de ressources et beaucoup de calculs, par exemple : l'encodage vidéo, alors Node.js n'est plus adapté, alors comment peut-on dépasser ce problème ?