

Rapport TP1

Introduction

Dans le cadre de ce travail pratique, j'ai développé une application web complète en utilisant plusieurs technologies modernes. Ce projet comprend un front-end en **Next.js**, un back-end en **Node.js avec Express**, et une base de données **PostgreSQL**. De plus, je suis en train de créer une machine virtuelle pour héberger mon application.

Technologies utilisées

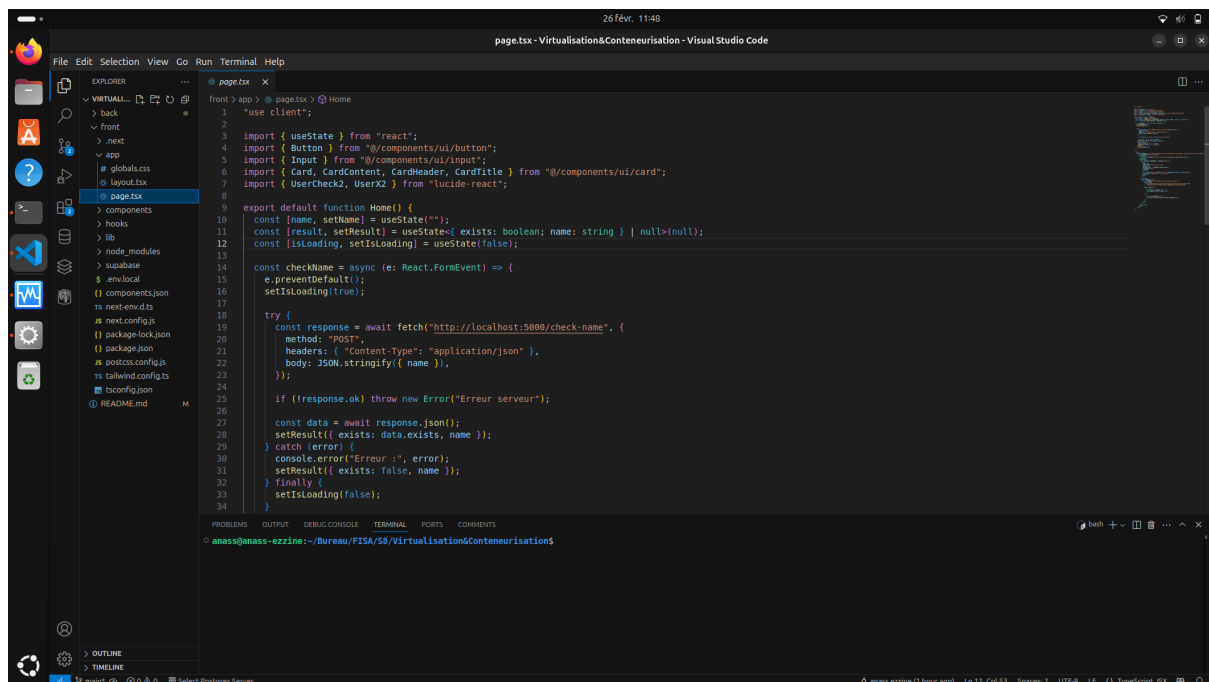
- **Front-end** : Next.js (React)
- **Back-end** : Node.js avec Express
- **Base de données** : PostgreSQL
- **Hébergement** : Machine virtuelle en cours de création

Développement de l'application

Front-end (Next.js)

J'ai utilisé **Next.js** pour construire l'interface utilisateur. Cette technologie permet un rendu côté serveur (SSR) et une meilleure performance.

Exemple de code (Next.js) :

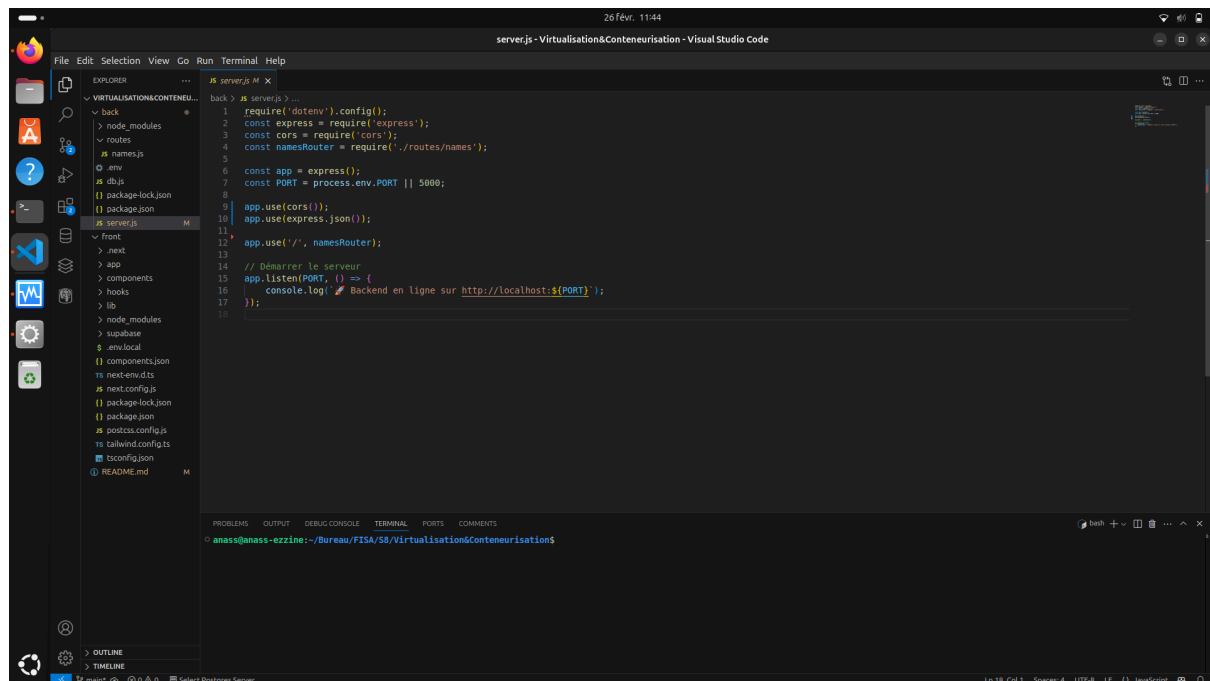


```
1  "use client";
2
3  import { useState } from "react";
4  import { Button } from "@components/ui/button";
5  import { Input } from "@components/ui/input";
6  import { Card, CardContent, CardHeader, CardTitle } from "@components/ui/card";
7  import { UserCheck2, UserX2 } from "lucide-react";
8
9  export default function Home() {
10   const [name, setName] = useState("");
11   const [result, setResult] = useState<{ exists: boolean; name: string } | null>(null);
12   const [isLoading, setIsLoading] = useState(false);
13
14   const checkName = async (e: React.FormEvent) => {
15     e.preventDefault();
16     setIsLoading(true);
17
18     try {
19       const response = await fetch("http://localhost:5000/check-name", {
20         method: "POST",
21         headers: { "Content-Type": "application/json" },
22         body: JSON.stringify({ name }),
23       });
24       if (!response.ok) throw new Error("Erreur serveur");
25
26       const data = await response.json();
27       setResult({ exists: data.exists, name });
28     } catch (error) {
29       console.error("Erreur :", error);
30       setResult({ exists: false, name });
31     } finally {
32       setIsLoading(false);
33     }
34   }
35 }
```

Back-end (Node.js avec Express)

Le serveur back-end est construit avec **Express.js**, qui permet de gérer les requêtes HTTP et d'interagir avec la base de données.

Exemple de code (Express) :



```
server.js - Virtualisation&Conteneurisation - Visual Studio Code
26 févr. 11:44

back > ls server.js
1  require('dotenv').config();
2  const express = require('express');
3  const cors = require('cors');
4  const namesRouter = require('./routes/names');
5
6  const app = express();
7  const PORT = process.env.PORT || 5000;
8
9  app.use(cors());
10 app.use(express.json());
11
12 app.use('/', namesRouter);
13
14 // Démarrer le serveur
15 app.listen(PORT, () => {
16   console.log(`Backend en ligne sur http://localhost:${PORT}`);
17 });
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

```
anass@anass-ezzine:~/Bureau/FISA/58/Virtualisation&Conteneurisation$
```

Base de données (PostgreSQL)

J'ai utilisé **PostgreSQL** pour stocker les données de l'application. Voici un exemple de création de table :

Hébergement sur une machine virtuelle

Actuellement, je suis en train de configurer une **machine virtuelle** pour héberger mon application. Cette étape comprend :

- L'installation de **Linux** (Ubuntu)

