

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

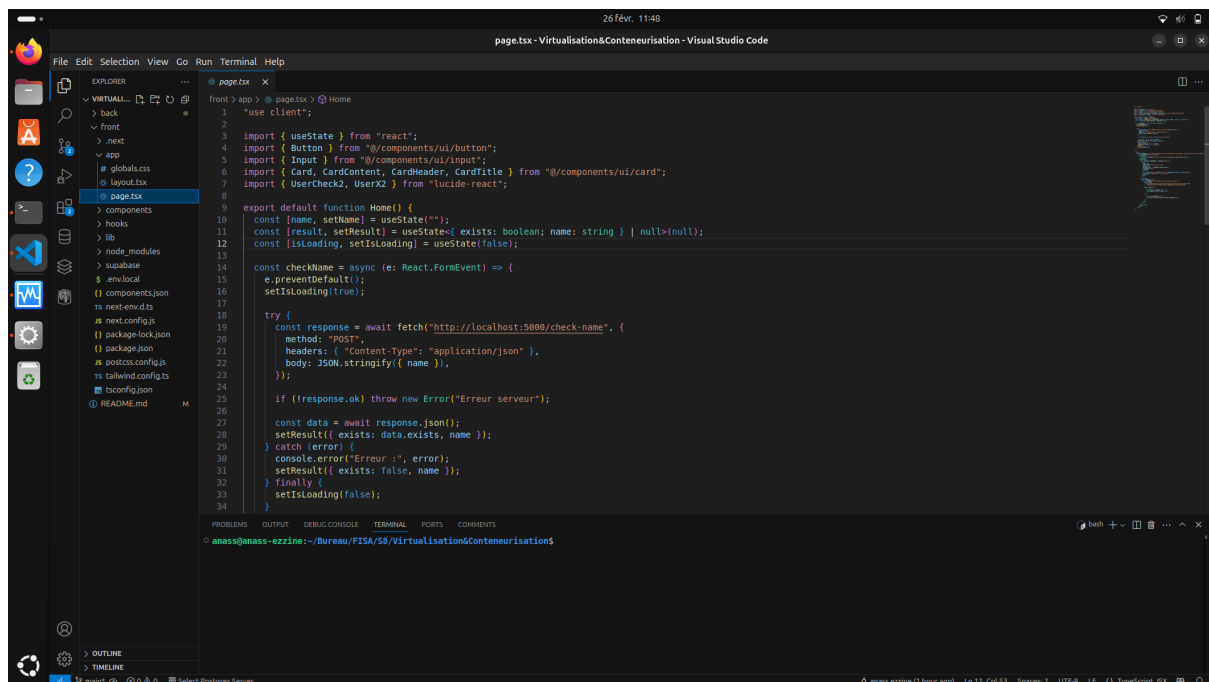
- **Front-end** : Next.js (React)
- **Back-end** : Node.js avec Express
- **Base de données** : PostgreSQL
- **Hébergement** : Machine virtuelle en cours de création

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) :

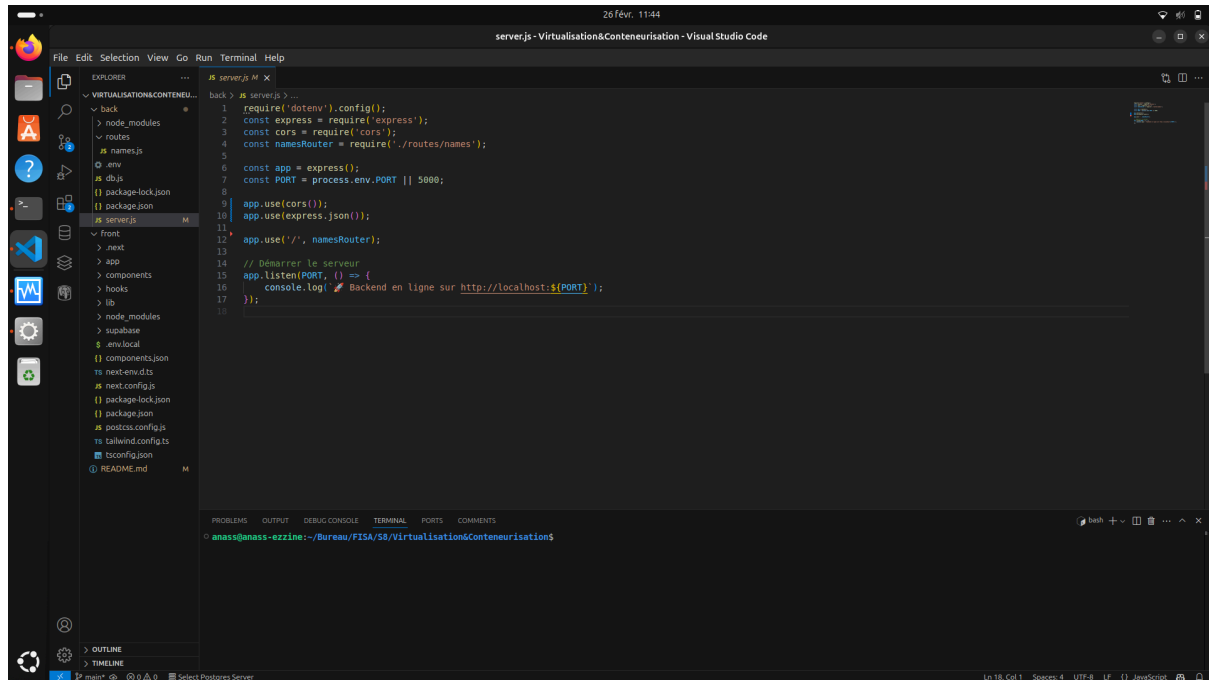


```
1  "use client";
2
3  import { useState } from "react";
4  import { Button } from "@components/ui/button";
5  import { Input } from "@components/ui/input";
6  import { Card, CardContent, CardHeader, CardTitle } from "@components/ui/card";
7  import { UserCheck2, UserX2 } from "lucide-react";
8
9  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) :



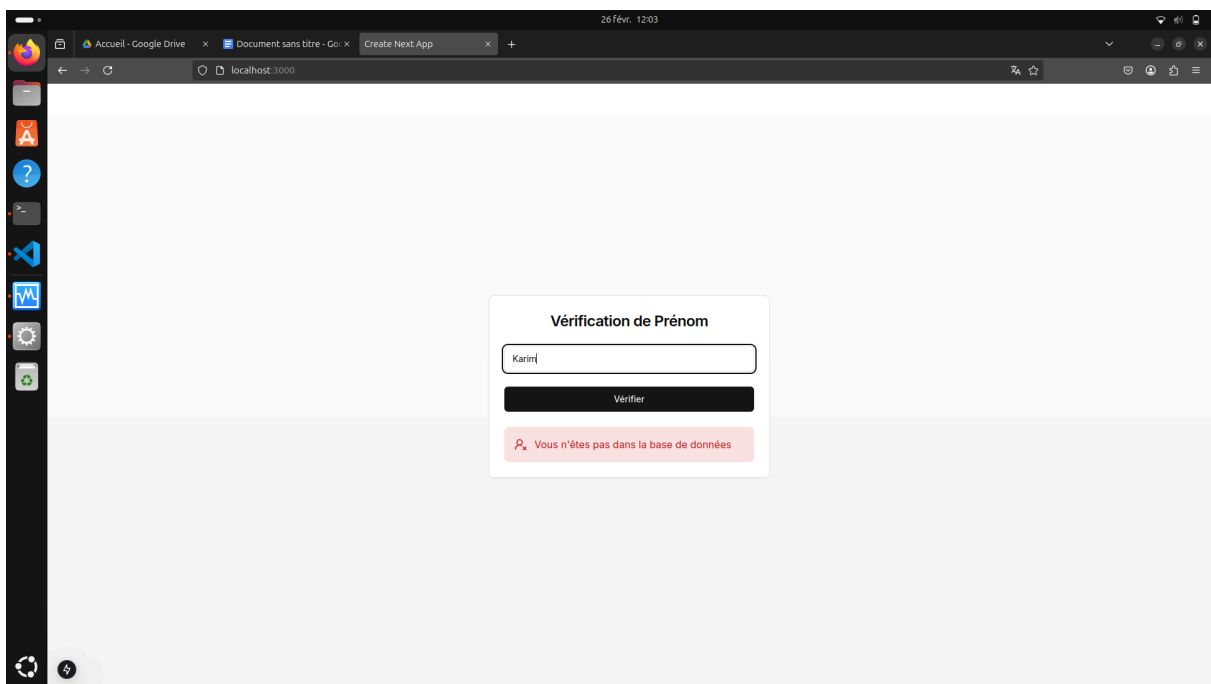
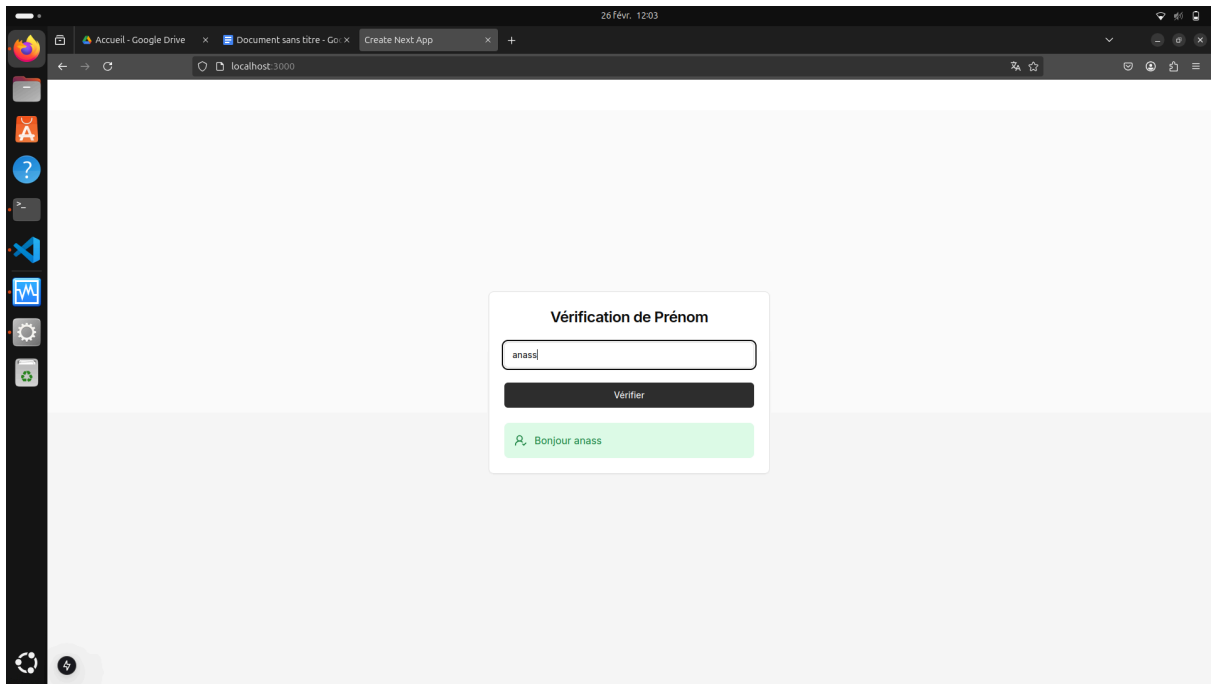
```
server.js - Virtualisation&Conteneurisation - Visual Studio Code
26 févr. 11:44

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 });
```

```
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 :



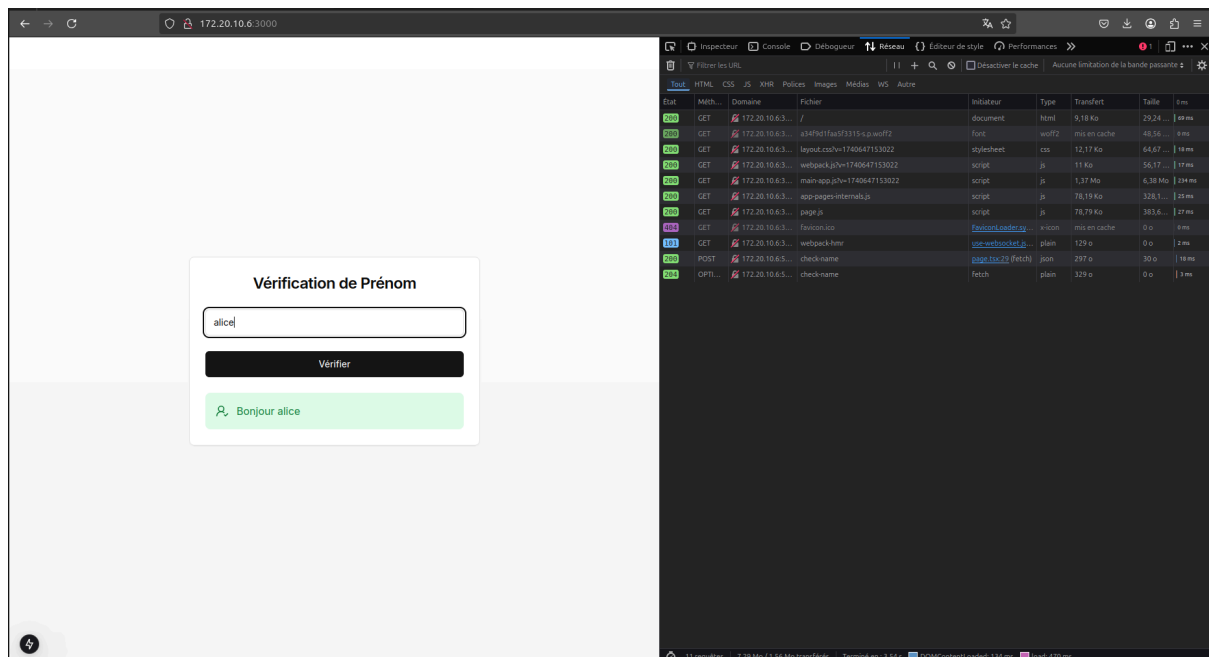
Hébergement sur une Machine Virtuelle

Je suis en train de configurer une machine virtuelle pour héberger mon application. Voici les étapes réalisées jusqu'à présent :

- **Installation de Linux (Ubuntu)**
- **Configuration et installation des outils nécessaires :**
 - VS Code
 - Node.js et npm
 - PostgreSQL (avec insertion des données)

- **Déploiement de l'application :**
 - Le frontend est accessible sur **172.20.10.6:3000**
 - Le backend est également accessible sur **172.20.10.6:5000**
(probablement une erreur de frappe dans ton message où tu as écrit 3000 deux fois)
- **Modification de la configuration PostgreSQL :**
 - PostgreSQL a été configuré pour écouter sur **172.20.10.6** au lieu de **localhost**, afin de permettre un accès distant à la base de données.

🚀 **L'environnement est maintenant prêt pour l'hébergement de l'application sur la machine virtuelle !**



Hébergement sur trois Machines Virtuelles

L'architecture repose sur :

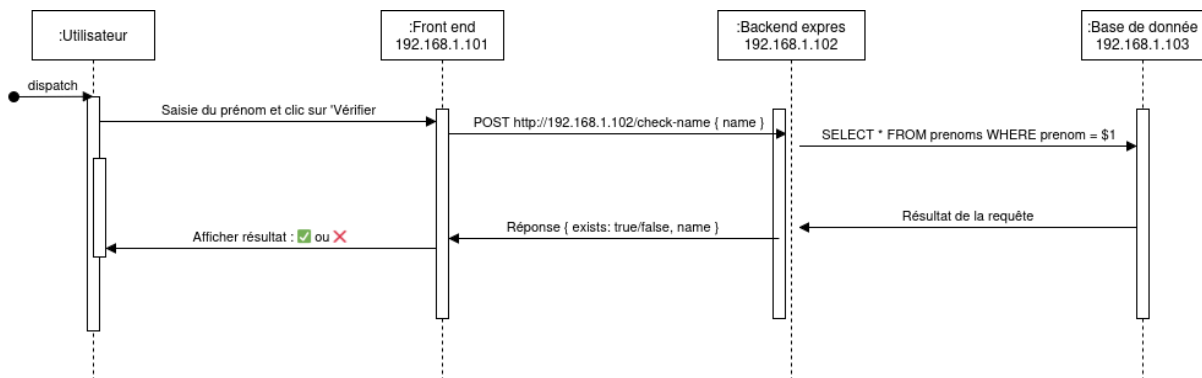
- Une machine pour le front-end développé en **Next.js**
- Une machine pour le back-end basé sur **Node.js** avec **Express**
- Une machine pour la base de données **PostgreSQL**

1.1 Architecture Réseau

Chaque machine virtuelle est configurée avec une adresse IP attribuée dynamiquement via DHCP et utilise le mode réseau ponté (Bridge Adapter), permettant ainsi leur accès depuis le réseau local.

Machine Virtuelle	Rôle	Adresse IP	Services Hébergés
frontend-vm	Frontend	statique	Next.js
backend-vm	Backend	statique	Node.js + Express
database-vm	Base de données	statique	PostgreSQL

Diagrammes de séquence:



Rapport d'installation et configuration

♦ 1. Installation du serveur Ubuntu

- Une **machine virtuelle Ubuntu Server** a été installée.
- Une **configuration par pont** a été mise en place pour permettre à la machine d'avoir une adresse IP fixe et accessible sur le réseau.

♦ 2. Installation et configuration du serveur SSH

- Le serveur **SSH** a été installé pour permettre l'accès à distance :
`sudo apt install openssh-server -y`

♦ 3. Copie du projet vers la machine distante

- Le dossier **front** a été copié depuis la machine hôte vers la machine serveur (172.20.10.9) via `scp` : `scp -r /home/anass/Bureau/FISA/vc/front front@172.20.10.11:`
- Le dossier **back** a été copié depuis la machine hôte vers la machine serveur (172.20.10.9) via `scp` : `scp -r /home/anass/Bureau/FISA/vc/back front@172.20.10.12:`

♦ 4. Installation de Node.js et NPM

Node.js (version 18) a été installé avec la commande :

- `curl -fsSL https://deb.nodesource.com/setup_18.x | sudo -E bash -`
- `sudo apt install -y nodejs`

♦ 5. Installation de l'interface graphique

Ajout de l'interface Ubuntu Desktop pour une utilisation graphique :

- `sudo apt install ubuntu-desktop -y`
- `sudo apt install gdm3 -y`
- `sudo reboot`

♦ 6. Configuration des adresses IP statiques

Configuration du réseau avec Netplan pour attribuer des adresses IP statiques :

- Adresse Frontend : 172.20.10.11
- Adresse Backend : 172.20.10.12
- Adresse BDD : 172.20.10.13

Modification du fichier Netplan et application des changements.

♦ 7. Installation et configuration de PostgreSQL

PostgreSQL a été installé avec la commande : `sudo apt install postgresql postgresql-contrib -y`

Modification du fichier **postgresql.conf** pour permettre l'écoute sur toutes les interfaces : `sudo nano /etc/postgresql/16/main/postgresql.conf`

- `listen_addresses = '*'`

Modification du fichier **pg_hba.conf** pour autoriser les connexions distantes : `sudo nano /etc/postgresql/14/main/pg_hba.conf`

- Ajout de la ligne :`host all all 0.0.0.0/0 md5`
- Redémarrage du service PostgreSQL : `sudo systemctl restart postgresql`

♦ 8. Création et configuration de la base de données

- Connexion à PostgreSQL : `sudo -u postgres psql`
- Création de la base de données `ma_base` : `CREATE DATABASE ma_base;`
- Connexion à la base : `\c ma_base`
- Création de la table `prenoms` : `CREATE TABLE prenoms (
 id SERIAL PRIMARY KEY,
 prenom VARCHAR(50) NOT NULL
);`
- Insertion de données dans la table : `INSERT INTO prenoms (prenom)VALUES(('Anass'),('Sofia'),('Mohamed'),('Amina'),('Youssef'));`

♦ 10. Lancement des machines et services

Pour lancer les machines et exécuter les services, **suivez ces étapes**:

① Activer le partage de connexion avec un téléphone

- Activez le **partage de connexion mobile** via **USB** ou **Wi-Fi**.
- Assurez-vous que les machines **Frontend** (**172.20.10.11**) et **Backend** (**172.20.10.12**) ont bien accès au réseau.

② Lancer le serveur de base de données PostgreSQL

Sur la machine **Backend** (**172.20.10.12**), démarrez PostgreSQL : `sudo systemctl restart postgresql`

③ Lancer le Backend

Sur la machine **Backend** (**172.20.10.12**), exécutez :

- `cd /home/front/back`
- `node server.js`

4 Lancer le Frontend

Sur la machine **Frontend** (**172.20.10.11**), exécutez :

- `cd /home/front/front`
- `npm run dev`