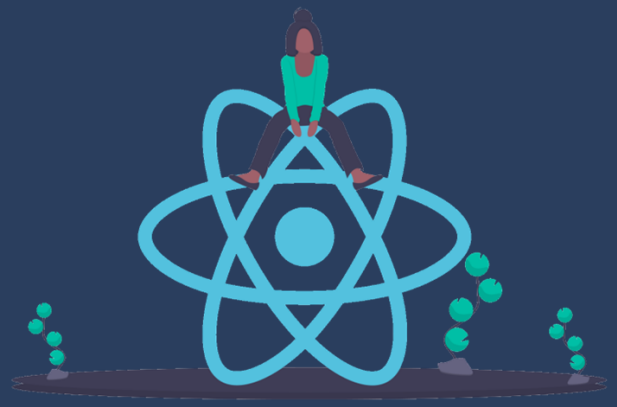
TD3

Frontend with React



https://www.jordansablon.fr/enseignement

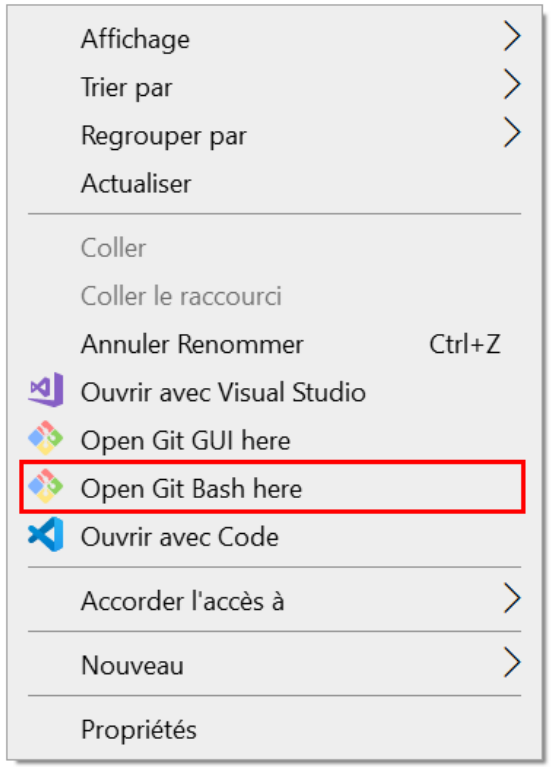
https://github.com/jsablonEnseignement/td3_sources

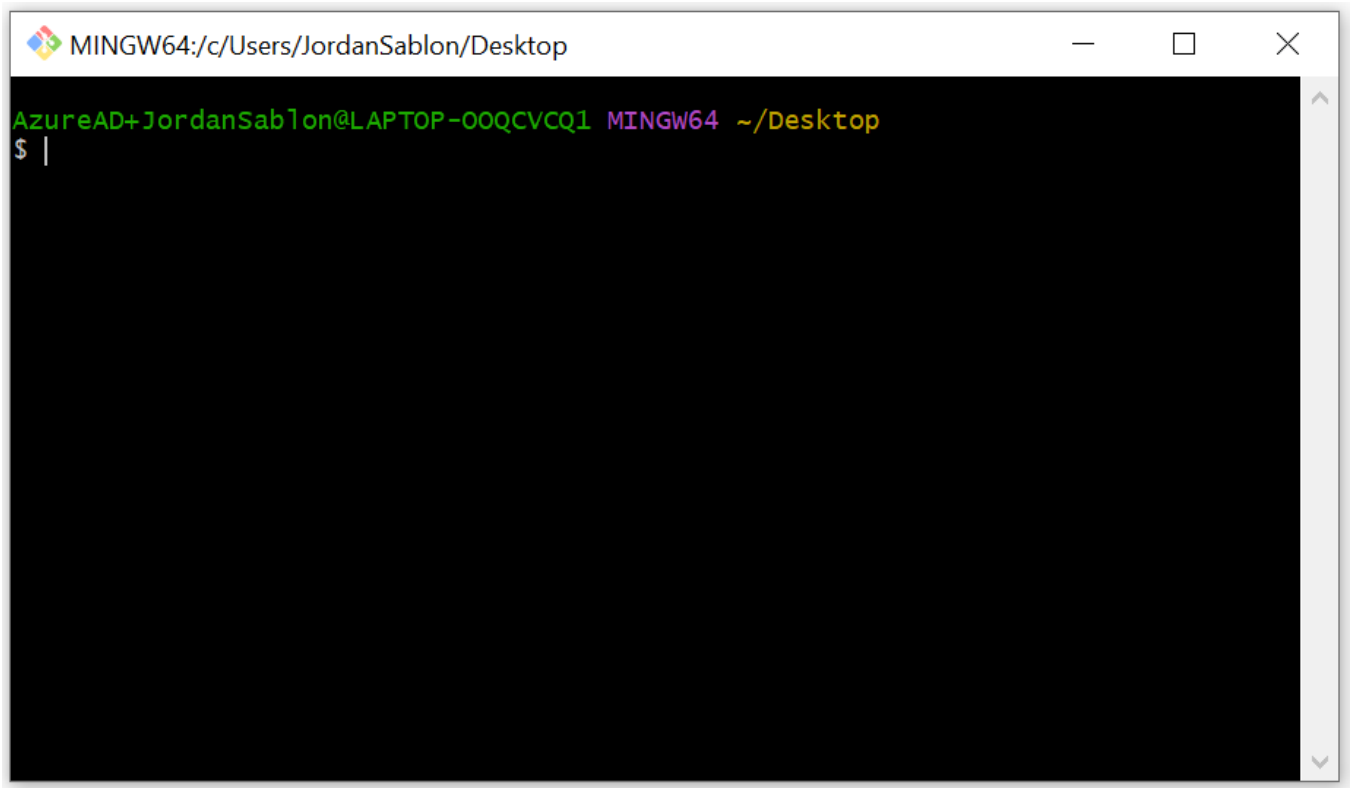
Sommaire - Partie 1 : prise en main de React

Récupération du code source du projet & installation des dépendances

1. Se positionner à l'emplacement de votre choix (où se trouvera le futur dossier du projet)

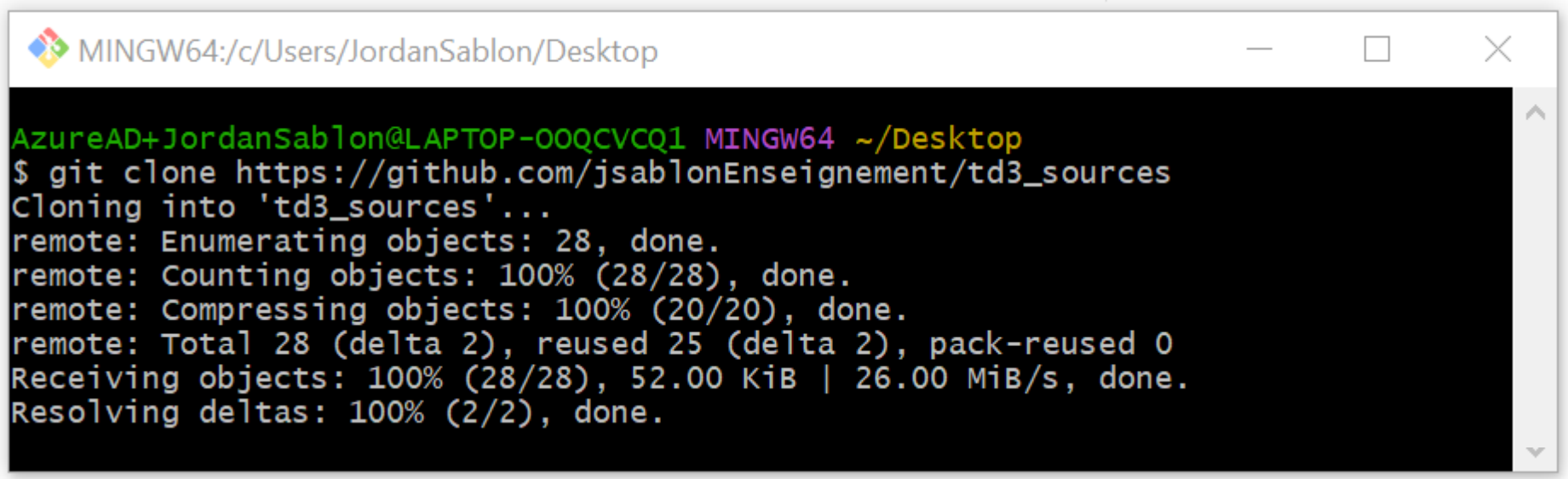
2. Ouvrir un invité de commande à l'emplacement précédent :



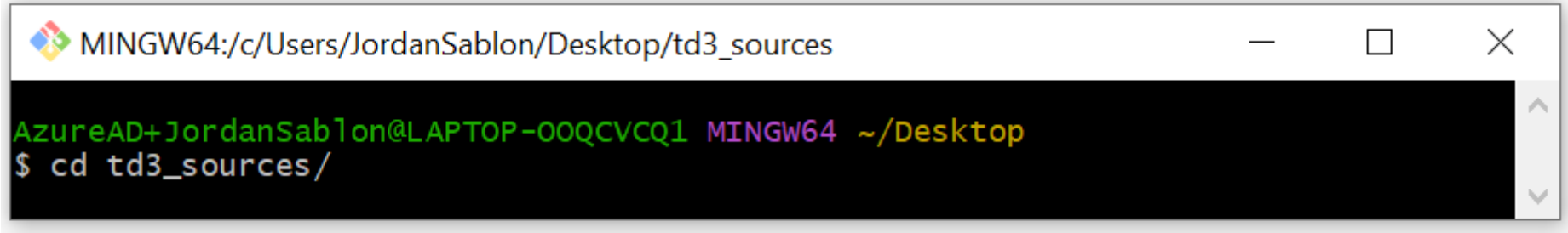


Récupération du code source du projet & installation des dépendances

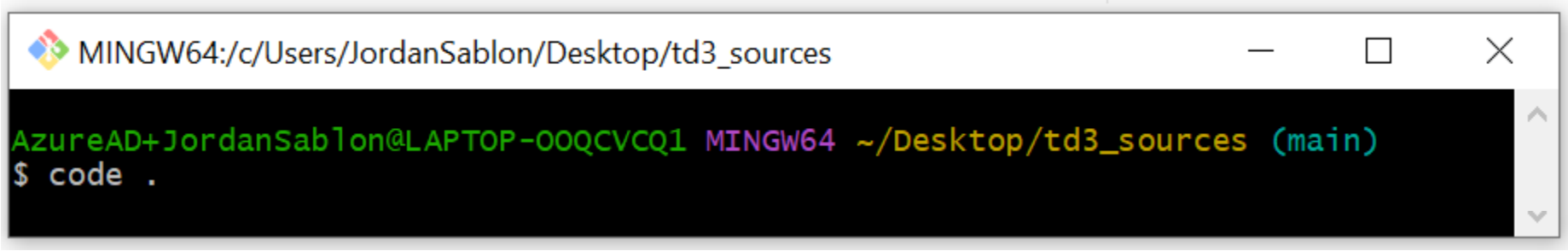
3. Cloner le répertoire Git : https://github.com/jsablonEnseignement/td3_sources

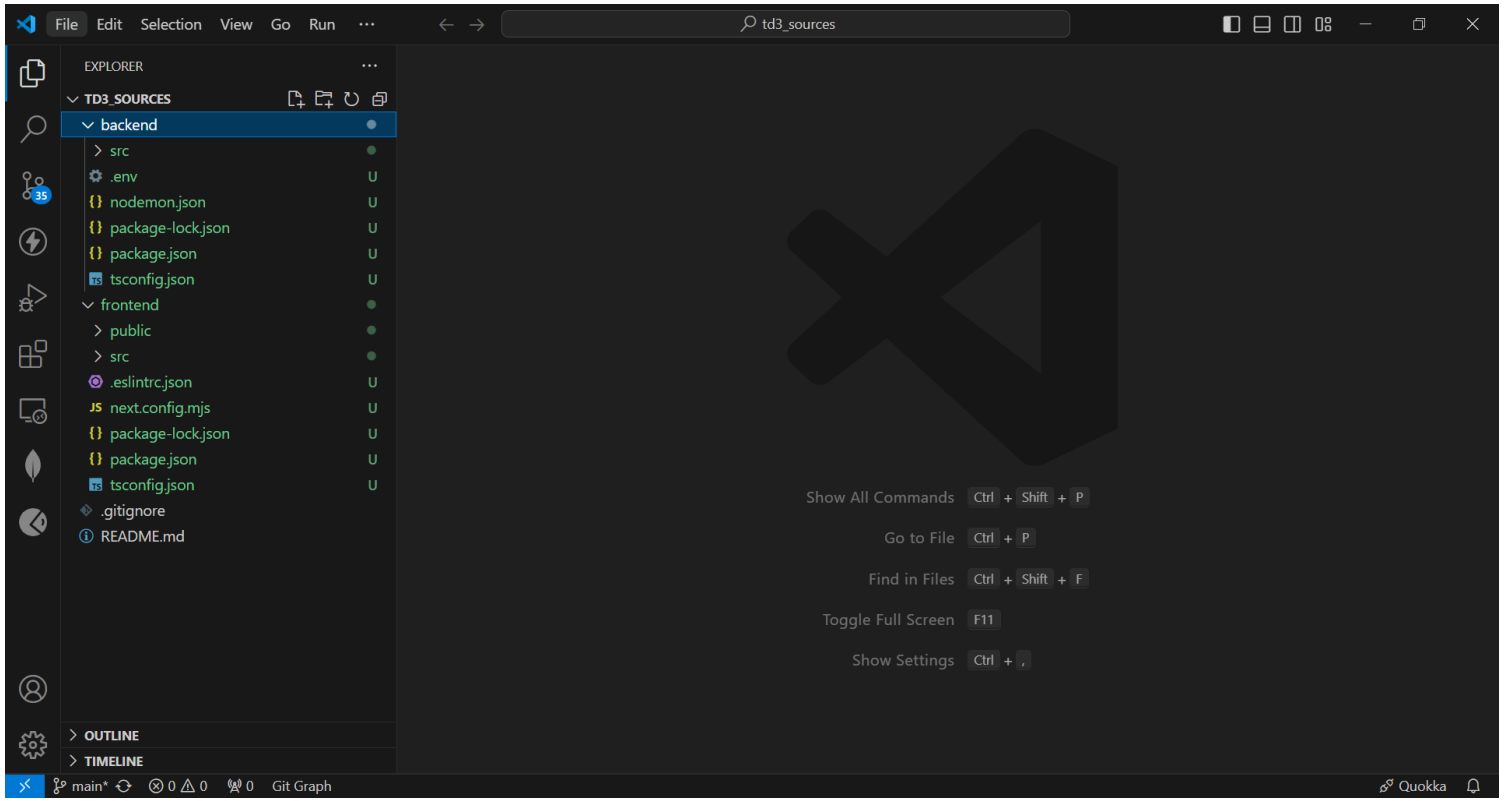


4. Se rendre dans le dossier précédemment cloné :



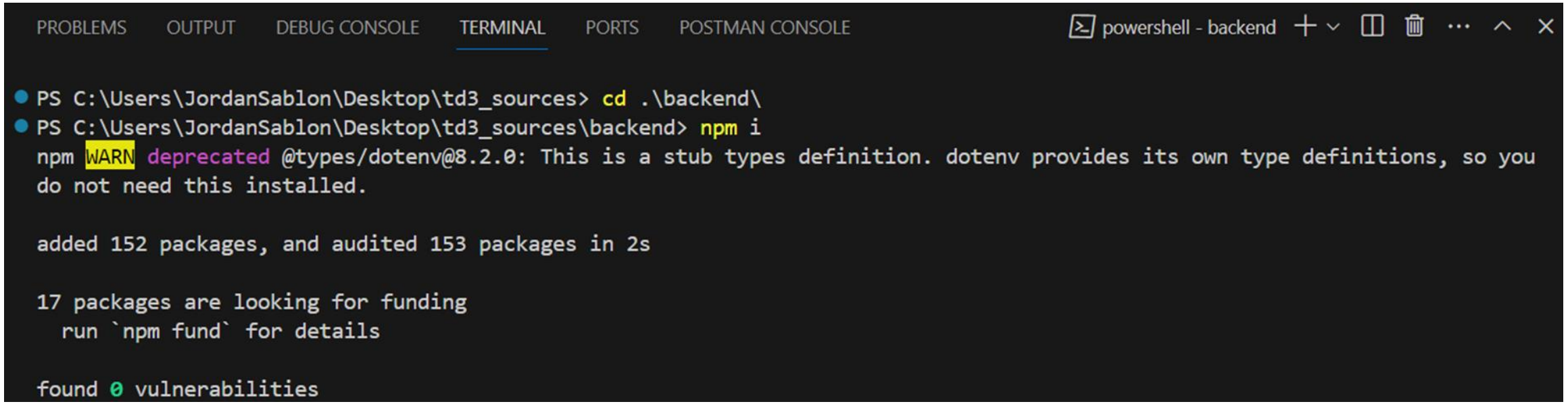
5. Ouvrir le projet dans VSCode :





Récupération du code source du projet & installation des dépendances

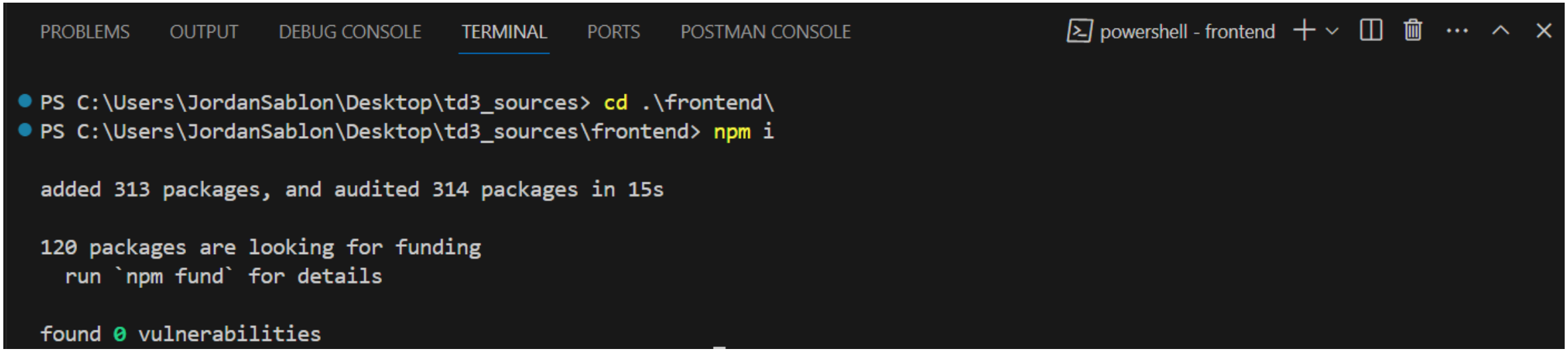
Depuis le terminal de VSCode dans le dossier backend, exécuter la commande suivante :



Les dépendances nécessaires au bon fonctionnement de l'application viennent d'être installées

Récupération du code source du projet & installation des dépendances

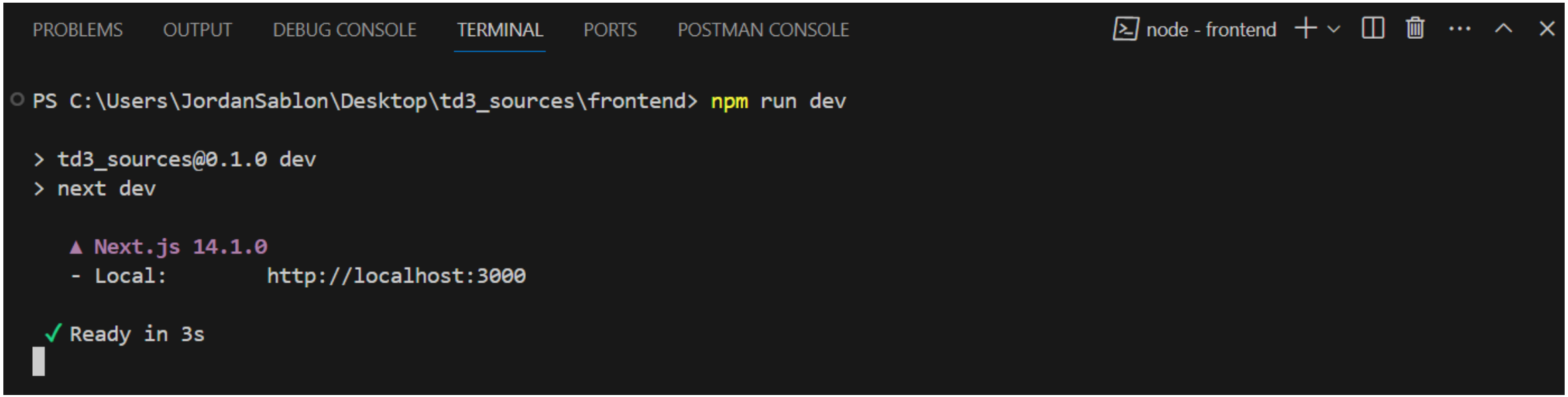
Depuis le terminal de VSCode dans le dossier frontend, exécuter la commande suivante :



Les dépendances nécessaires au bon fonctionnement de l'application viennent d'être installées

Récupération du code source du projet & installation des dépendances

Depuis le terminal de VSCode dans le dossier frontend, exécuter la commande suivante afin de démarrer l'application React :



L'application est maintenant accessible via l'url http://127.0.0.1:3000/

Création d'un premier composant

Créer un dossier **/components** dans **/src/app** qui regroupera l'ensemble de nos composants

Créer un dossier **Demo** dans **/components**

Créer un fichier **Demo.tsx** qui contiendra le code du composant :

```
export const Demo = () => {
  return <>Hello from our first component!</>;
};
```

Importer et faire appel à ce composant depuis le fichier **page.tsx**

```
"use client";
import { Demo } from "./components/Demo/Demo";

export default function Home() {
  return (
    <Demo />
  );
}
```

Utilisation d'une variable d'état (*state*) et du hook *useState*

Depuis notre composant **Demo.tsx**, importer le hook ***useState***

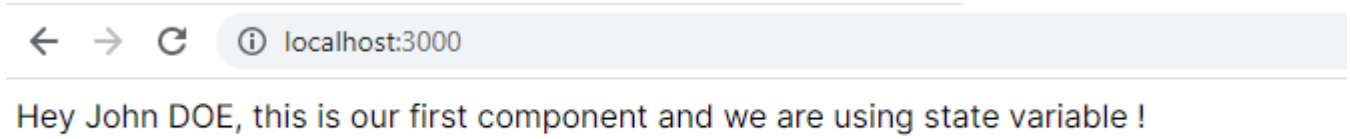
```
import { useState } from "react";
```

Initialisation du hook ***useState***

```
const [name, setName] = useState("John DOE");
```

Utilisation de la variable :

```
Hey {name}, this is our first component and we are using state variable!
```

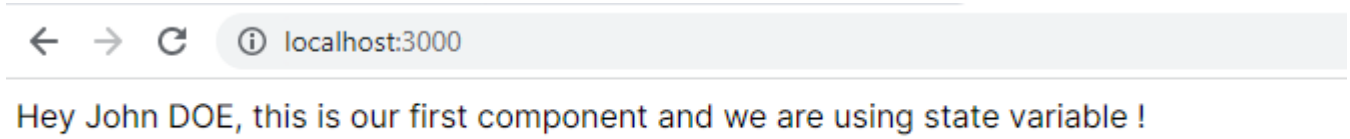


Utilisation d'une variable d'état (*state*) et du hook *useState*

Le composant **Demo.tsx** ressemble désormais à ceci :

```
import { useState } from "react";

export const Demo = () => {
  const [name, setName] = useState("John DOE");

  return <>Hey {name}, this is our first component and we are using state variable!
</>;
};
```

Il nous donne le rendu suivant :



Utilisation d'une variable d'état (*state*) et du hook *useState*

Ajout d'un champ de saisie ***input*** de type ***text***

```
<input type="text" defaultValue={name} />
```

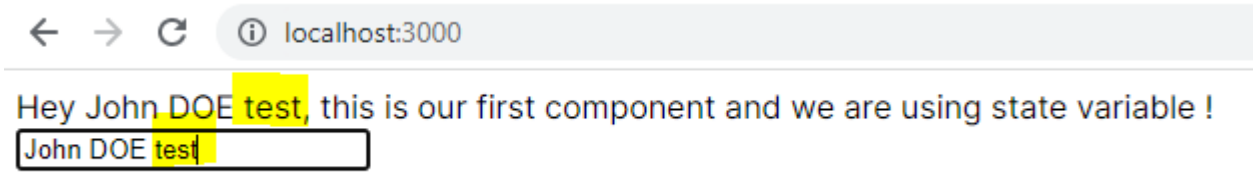
Ajout d'un évènement ***onChange*** afin de modifier la valeur

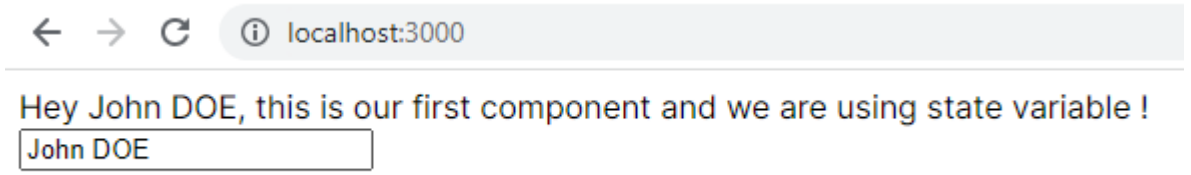
```
<input
    type="text"
    defaultValue={name}
    onChange={(e) => setName(e.target.value)}
/>
```

Utilisation d'une variable d'état (*state*) et du hook *useState*

Voici le code du composant et le rendu associé

```
import { useState } from "react";

export const Demo = () => {
  const [name, setName] = useState("John DOE");

  return (
    <>
      Hey {name}, this is our first component and we are using state variable!
      <input
        type="text"
        defaultValue={name}
        onChange={(e) => setName(e.target.value)}
      />

    </>;
};
```





Création d'un second composant & import / export

Création d'un fichier ***Hello.tsx*** qui contiendra le texte affiché du précédent composant :

```
export const Hello = () => {
  return (
    <>
      Hey John DOE, this is our first component and we are using state
variable
      !
    </>
  );
};
```

Création d'un second composant & import / export

Import du composant ***Hello.tsx*** depuis ***Demo.tsx***

```tsx
import { useState } from "react";
import { Hello } from "./Hello";

export const Demo = () => {
  const [name, setName] = useState("John DOE");
  return (
    <>
      <Hello />
      <br />
      <input
        type="text"
        defaultValue={name}
        onChange={(e) => setName(e.target.value)}
      />
    </>
  );
};
```

Utilisation des propriétés (*props*)

Ajout d'un paramètre ***name*** pour le composant ***Hello.tsx***

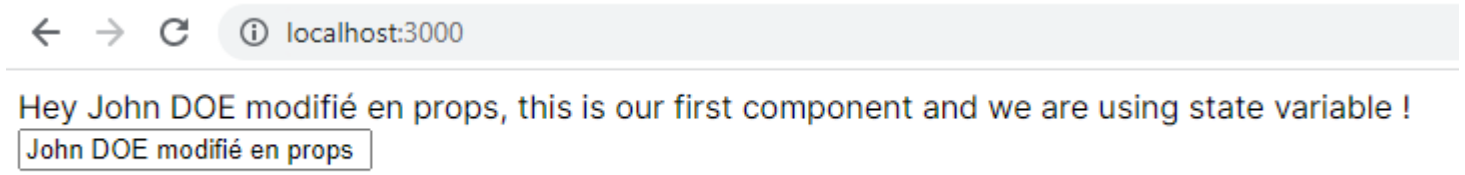
```
type HelloProps = {
  name: string;
};

export const Hello = ({ name }: HelloProps) => {
  return (
    <>
      Hey {name}, this is our first component and we are using state variable !
    </>
  );
};
```

→ On note la définition d'un type qui regroupe l'ensemble des paramètres d'entrée du composant ***Hello***

Utilisation des propriétés (*props*)

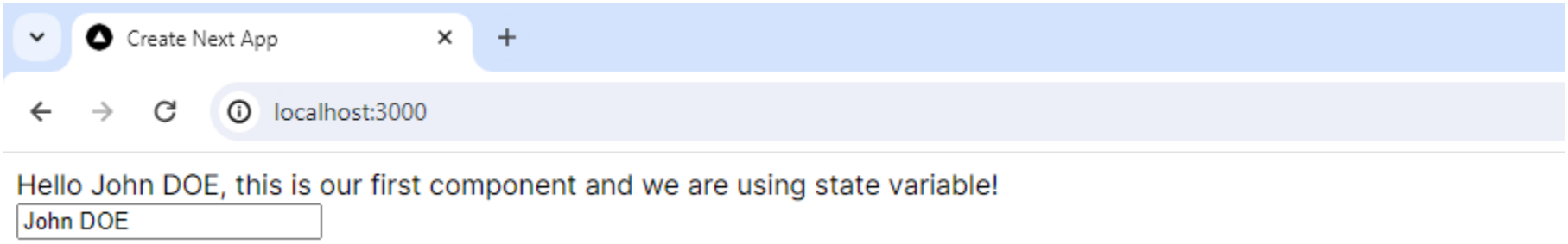
On passe la variable de state name comme paramètre du composant ***Hello***

```
<Hello name={name} />
```



Affichage du composant **<Hello />** seulement si *name* existe

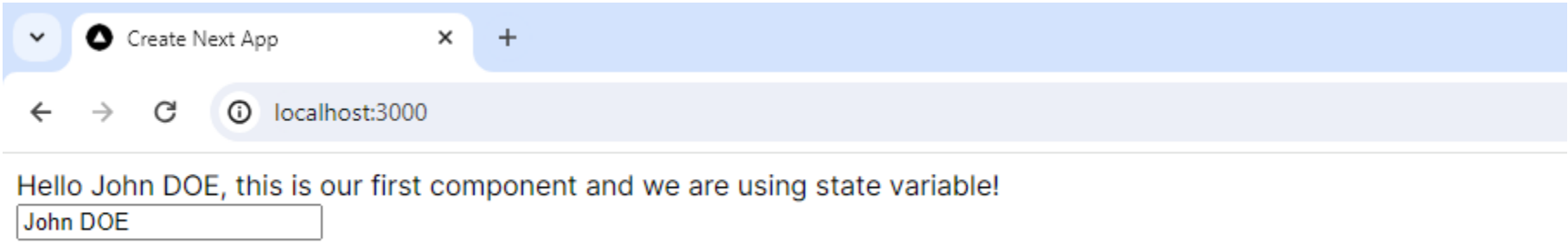
```
{name && <Hello name={name} />}
```



Conditionner l'affichage d'un composant

Il est également possible d'afficher un texte alternatif :

```
{name ? <Hello name={name} /> : "Veuillez renseigner un nom d'utilisateur"}
```



Utilisation du hook *useEffect*

Deux usages :

→ Exécution d'une action au chargement du composant (avant son rendu)

→ Exécution d'une action lors du changement d'état d'une ou plusieurs variable(s)

Syntaxe :

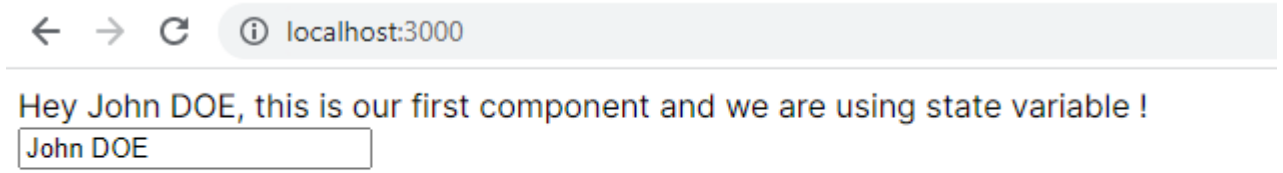
```
useEffect(() => {
        // some code here
}, []);
```

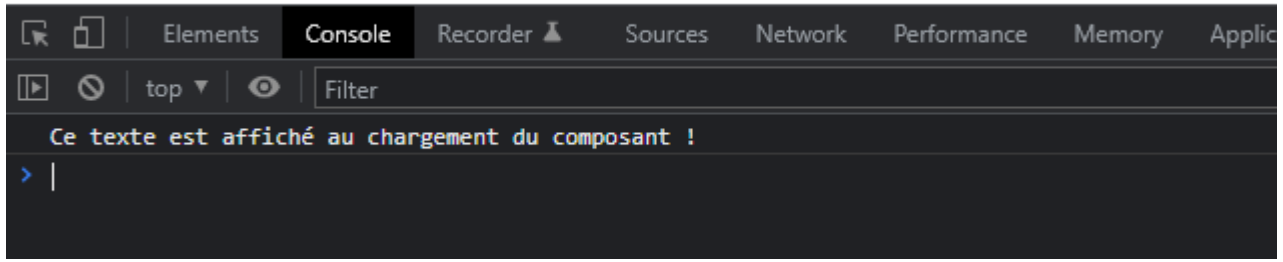
Utilisation du hook *useEffect*

Ajout d'un ***useEffect*** après la déclaration du ***useState*** dans le composant **Demo** afin d'écrire dans la console lors du chargement du composant.

Ouvrir les outils de développement du navigateur (F12) pour afficher la console.

```
useEffect(() => {
    console.log("Ce texte est affiché au chargement du composant !");
  }, []);
```

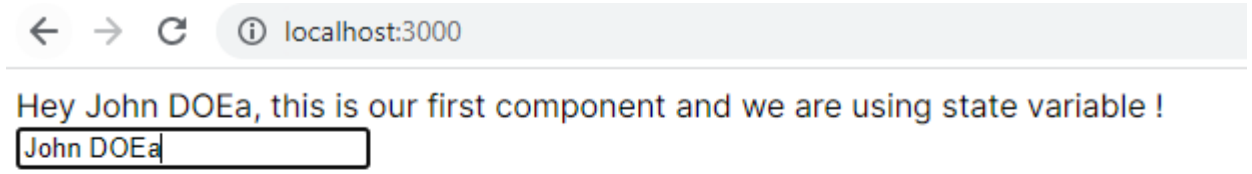


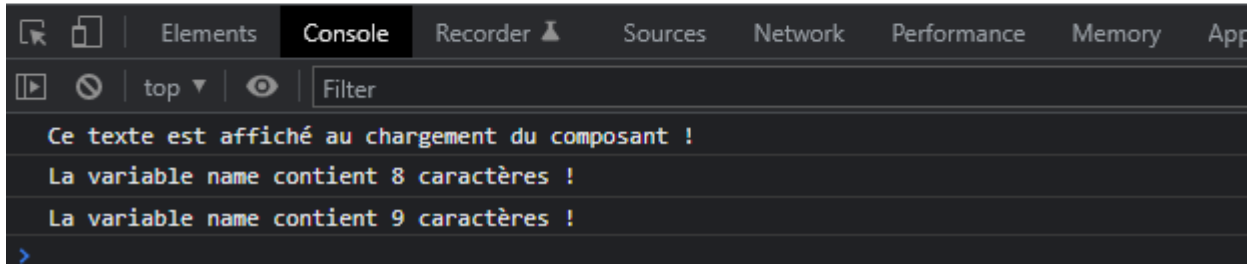


Utilisation du hook *useEffect*

Ajout d'un second ***useEffect*** d'afficher le nombre de caractère de la variable ***name***

```
useEffect(() => {
    console.log(`La variable name contient ${name.length} caractères !`);
  }, [name]);
```





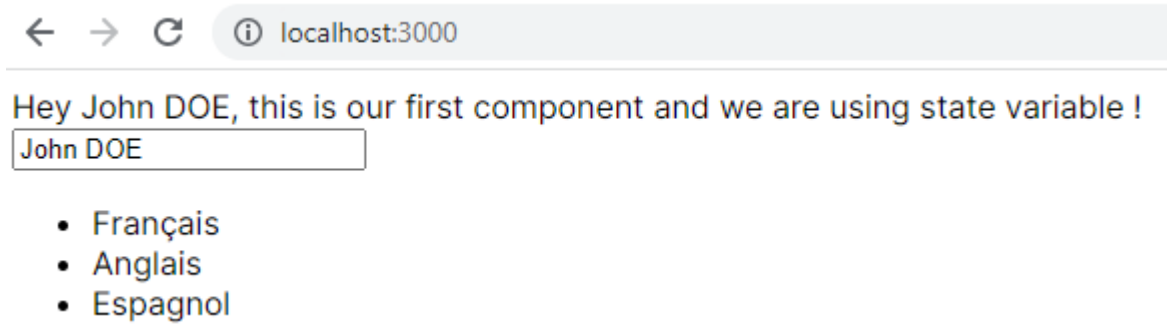
Affichage d'un composant au sein d'une itération

Ajout d'un tableau de valeur dans le composant **Demo.tsx**

```
  const languages = ["Français", "Anglais", "Espagnol"];
```

Affichage des valeurs sous forme de liste :

```
<ul>
  {languages.map((language) => (
    <li key={language}>{language}</li>
  ))}
</ul>
```



Affichage d'un composant au sein d'une itération

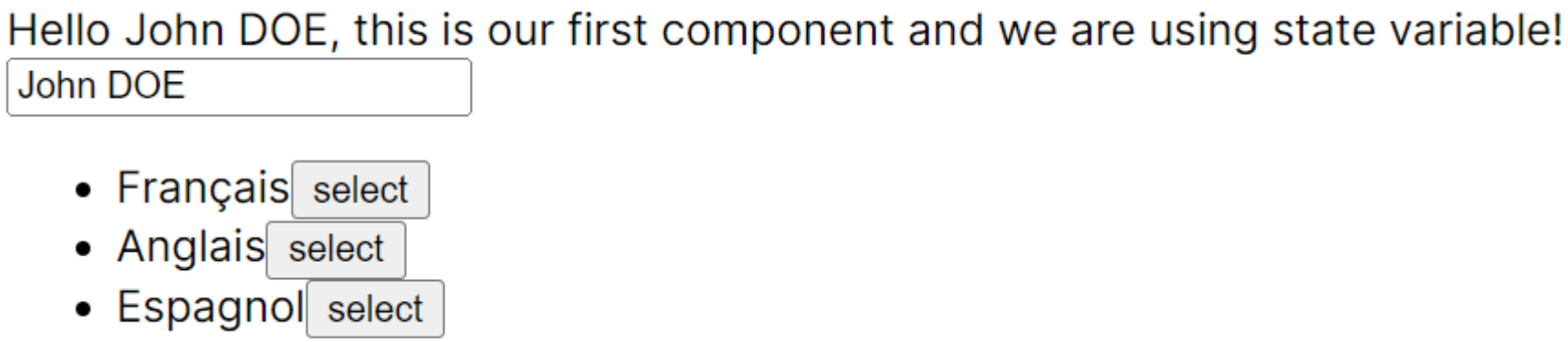
Voici désormais le code du composant :

```
import { useEffect, useState } from "react";
import { Hello } from "./Hello";

const languages = ["Français", "Anglais", "Espagnol"];

export const Demo = () => {
    const [name, setName] = useState("John DOE");

    useEffect(() => {
        console.log("Ce texte est affiché au chargement du composant !");
    }, []);

    useEffect(() => {
        console.log(`La variable name contient ${name.length} caractères !`);
    }, [name]);


    return <>
        {name ? <Hello name={name} /> : "Veuillez renseigner un nom d'utilisateur"}
        <br />
        <input type="text" defaultValue={name} onChange={(e) => setName(e.target.value)} />
        <ul>
            {languages.map((language) => (
                <li key={language}>{language}</li>
            ))}
        </ul>

    </>;
};
```

Affichage d'un composant au sein d'une itération

Ajout d'un bouton pour chaque élément :

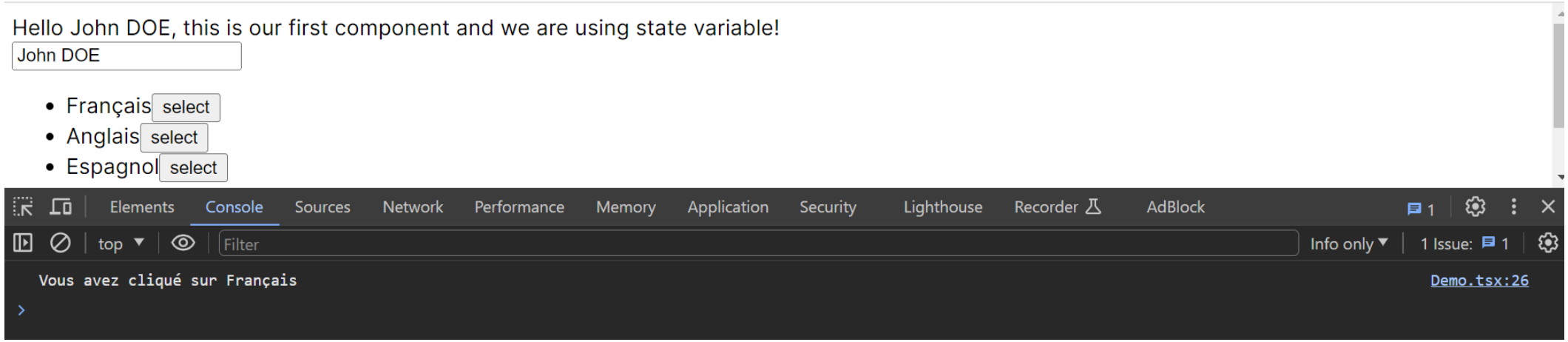
```
<ul>
    {languages.map((language) => (
        <li key={language}>{language}
           <button>select</button>
        </li>
    ))}
</ul>
```



Affichage d'un composant au sein d'une itération

Au clic sur ce bouton, on souhaite afficher la valeur de la ligne correspondante

```
<button onClick={() => console.log("Vous avez cliqué sur " + language)}>select</button>
```



Installation de la librairie

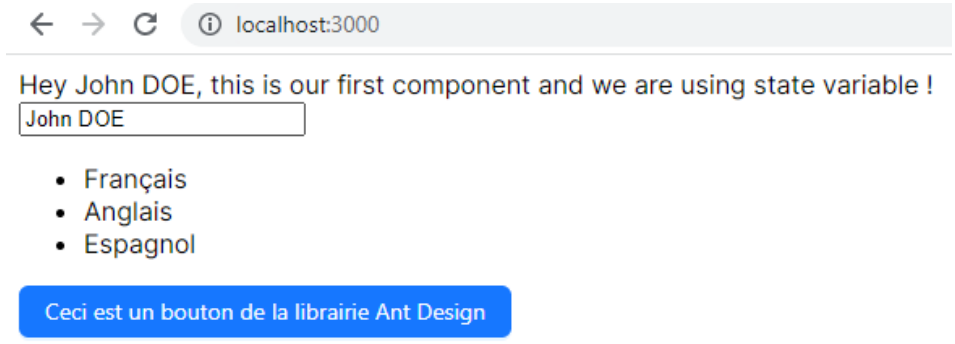
```
npm install antd
```

Import d'un composant (***Button***)

```
import { Button } from "antd";
```

Utilisation :

```
<Button type="primary">
        Ceci est un bouton de la librairie Ant Design
      </Button>
    </>
```



Création d'une application React permettant de gérer des étudiants

Sommaire - Partie 2 : gestion des étudiants

→ Démarrer le backend

→ Installation de la libraire Axios

→ Récupération et affichage de la liste des étudiants

→ Formulaire de création d'un nouvel étudiant

→ Edition d'un étudiant

→ Suppression d'un étudiant

Démarrer le backend

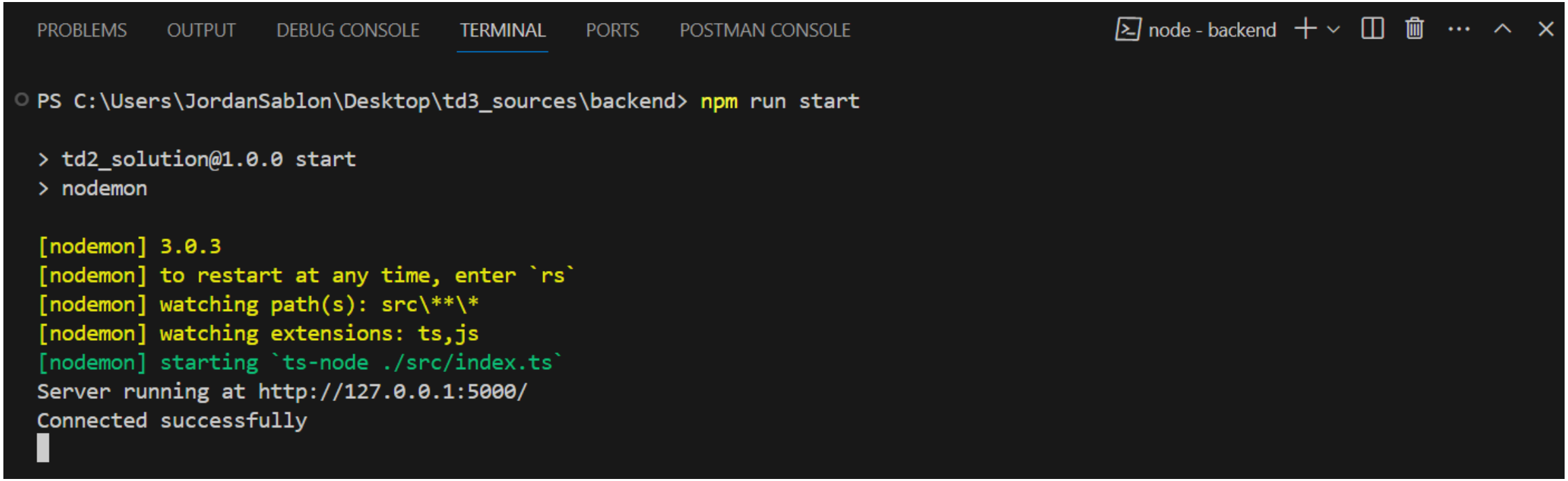
Dans le fichier `index.ts`, mettre à jour l'URI de connexion à la base de données mongodb en renseignant vos informations de connexion :

- username
- password
- domain
- db_name

```
mongoose.connect(
    `mongodb+srv://<username>:<password>@<domain>.mongodb.net/<db_name>?retryWrites=true&w=majority`
);
```

Démarrer le backend

Depuis le terminal de VSCode dans le dossier backend, exécuter la commande suivante afin de démarrer l'application React :



Installation de la dépendance Axios

Depuis le dossier frontend, installer la dépendance **axios**

```
npm install axios
```

Récupération et affichage de la liste des étudiants

Création d'un dossier **/Students** dans le dossier **/components**

Créer un fichier **Students.tsx** et y insérer le contenu suivant :

```
export const Students = () => {
  return <>Hello!</>
};
```

Modifier le fichier **page.tsx** pour faire appel à notre composant <Students />

```
"use client";
import { Students } from "./components/Students/Students";

export default function Home() {
  return <Students />;
}
```

Première requête HTTP depuis React

Ajouter un bouton dans le composant **<Students />**

```
export const Students = () => {
   return <button onClick={() => fetchStudents()}>Récupérer les étudiants</button>;
};
```

Créer une fonction **fetchStudents** qui fait une requête HTTP **GET** sur **http://localhost:5000/students**

```
const fetchStudents = async () => {
   const { data } = await axios.get(`http://localhost:5000/students`);
   console.log(data);
};
```

Utilisation du hook **useEffect**

```
import axios from "axios";
import { useEffect } from "react";

export const Students = () => {
  useEffect(() => {
    (async () => {
      const students = await fetchStudents();
      console.log(students);
    })();
  });

  const fetchStudents = async () => {
    return (await axios.get(`http://localhost:5000/students`)).data;
  };

  return <></>;
};
```

Affichage des étudiants via un composant dédié <Student />

Créer un dossier **/types** dans **/src/app** qui regroupera l'ensemble de nos types

Créer un fichier **Student.ts** qui contiendra le code suivant :

```
export type Student = {
    _id: string;
    name: string;
    firstname: string;
    age: number;
  };
```

Affichage des étudiants via un composant dédié <Student />

Créer un dossier **Student** dans **/components**

Créer un fichier **Student.tsx** qui contiendra le code suivant :

```tsx
import { Space } from "antd";
import { Student as StudentType } from "../../types/Student";

type StudentProps = {
  student: StudentType;
};

export const Student = ({ student, onEdit, onDelete }: StudentProps) => {
  return (
    <div key={student._id}>
      <Space>
          <label>
            <b>Nom :</b>
          </label>
          {student.name}
          <label>
            <b>Prénom :</b>
          </label>
          {student.firstname}
          <label>
            <b>Age :</b>
          </label>
          {student.age}
        </Space>
    </div>
  );
};
```

Affichage des étudiants via un composant dédié <Student />

Dans le composant **<Students />**, créer une variable de state afin d'y stocker les étudiants récupérés depuis le **useEffect** :

```
const [students, setStudents] = useState<StudentType[]>([]);
```

```
useEffect(() => {
    const fetchStudents = async () => {
        const { data } = await axios.get(`http://localhost:5000/students`);
        setStudents(data);
    };
    fetchStudents();
});
```

Affichage des étudiants via un composant dédié <Student />

Pour chaque étudiant, on affiche le composant **<Student />**

S'il n'y a aucun étudiant à afficher, on affiche le texte « Aucun étudiant à afficher »

```
<>
  <h3>Liste des étudiants :</h3>
  {students.length > 0 ? (
    students.map((student) => (
      <Student
        key={student._id}
        student={student}
      />
    ))
  ) : (
      <div>Aucun étudiant à afficher</div>
  )}
</>
```

Affichage des étudiants via un composant dédié <Student />

Voici désormais le code du composant **<Students />**

```
import axios from "axios";
import { useEffect, useState } from "react";
import { Student } from "../Student/Student";
import { Student as StudentType } from "../../types/Student";

export const Students = () => {
    const [students, setStudents] = useState<StudentType[]>([]);

    useEffect(() => {
        const fetchStudents = async () => {
            const { data } = await axios.get(`http://localhost:5000/students`);
            setStudents(data);
        };
        fetchStudents();
    });

    return (
        <>
            <h3>Liste des étudiants :</h3>
            {students.length > 0 ? (
            students.map((student) => (
                <Student
                key={student._id}
                student={student}
                />
            ))
            ) : (
            <div>Aucun étudiant à afficher</div>
            )}
        </>
    );
};
```

Affichage des étudiants via un composant dédié <Student />

On obtient le rendu suivant :

Liste des étudiants :

Nom : DOE **Prénom :** Jane **Age :** 20

Nom : DOE **Prénom :** Jane **Age :** 19

Création d'un étudiant via un composant <AddStudent />

Création d'un dossier **AddStudent** dans le dossier **/components**

Créer un fichier **AddStudent.tsx**

Ce composant contiendra un formulaire **<Form />** avec un **<Form.Item />** par attribut

Création d'un étudiant via un composant <AddStudent />

Exemple :

```
<Form
    name="basic"
    labelCol={{ span: 8 }}
    wrapperCol={{ span: 16 }}
    style={{ maxWidth: 600 }}
    initialValues={{ remember: true }}
    onFinish={() => onAdd(name, firstname, age!)}
    autoComplete="off"
>
    <Form.Item
      label="Nom"
      name="name"
      rules={[{ required: true, message: "Le nom est obligatoire !" }]}
    >
      <Input onChange={(e) => setName(e.target.value)} />
    </Form.Item>
</Form>
```

Création d'un étudiant via un composant <AddStudent />

On créera une variable de state pour stocker la valeur saisie de chaque attribut :

```
  const [name, setName] = useState<string>("");
  const [firstname, setFirstname] = useState<string>("");
  const [age, setAge] = useState<number>();
```

Le composant aura les props suivantes :

```
type AddStudentProps = {
  onAdd: (name: string, firstname: string, age: number) => void;
};
```

Création d'un étudiant via un composant <AddStudent />

On ajoutera enfin un bouton permettant la création d'un étudiant :

```
<Form.Item wrapperCol={{ offset: 8, span: 16 }}>
  <Button type="primary" htmlType="submit">
    Ajouter
  </Button>
</Form.Item>
```

Ce composant sera appelé dans le composant **<Students />**

```
<>
    <h3>Liste des étudiants :</h3>
    {students.length > 0 ? (
      students.map((student) => (
        <Student
          key={student._id}
          student={student}
        />
      ))
    ) : (
      <div>Aucun étudiant à afficher</div>
      )}
      <Divider />
      <AddStudent onAdd={addStudent} />
</>
```

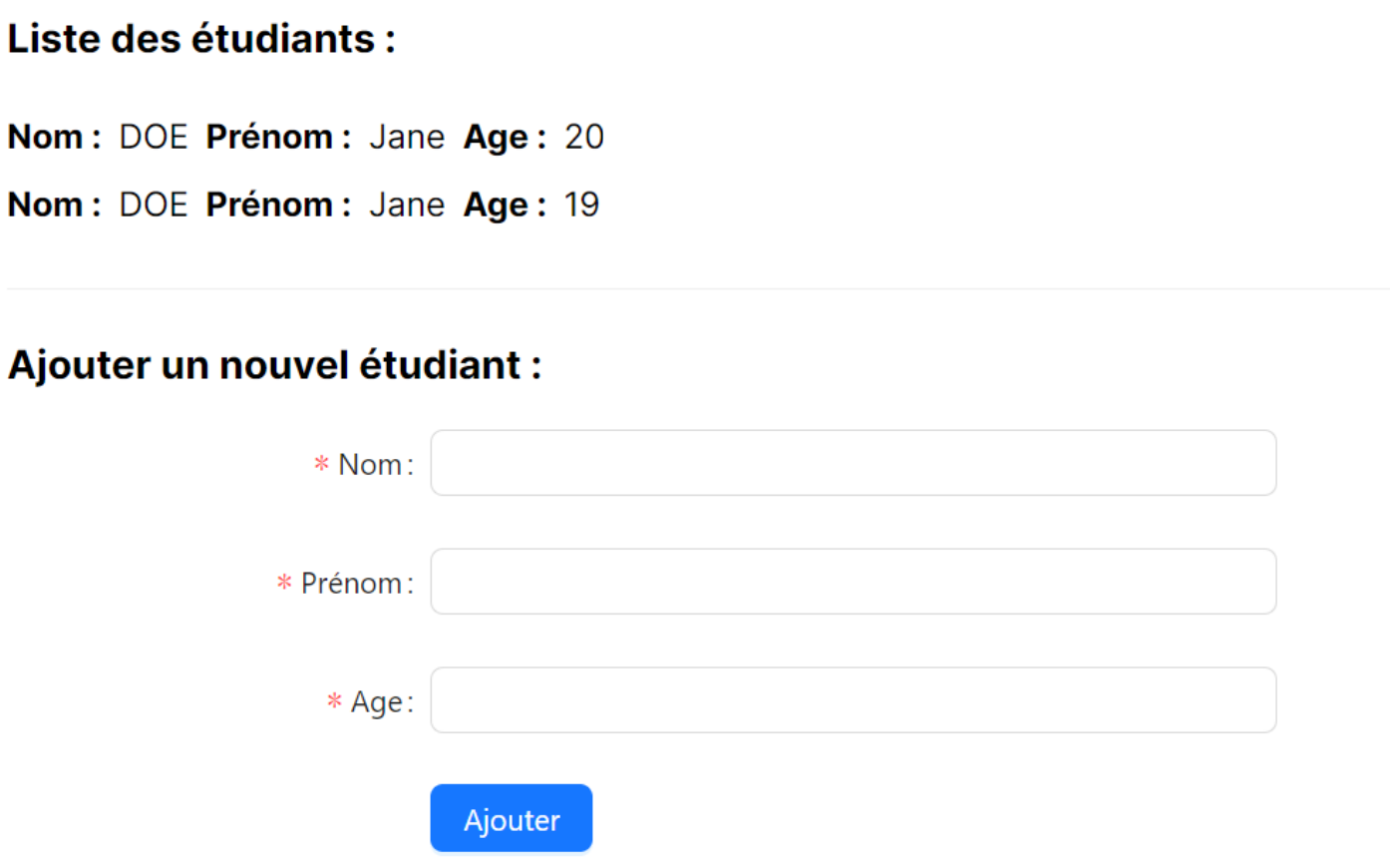
Création d'un étudiant via un composant <AddStudent />

Création d'une fonction addStudent :

```
const addStudent = async (name: string, firstname: string, age: number) => {
  const student = {
    name: name,
    firstname: firstname,
    age: age,
  };

  const newStudent = (
    await axios.post(`http://localhost:5000/students`, student)
  ).data;
  setStudents([...students, newStudent]);
};
```

Création d'un étudiant via un composant <AddStudent />

On obtient le rendu suivant :



Edition d'un étudiant

Dans le composant **<Student />**, créer une variable de state permettant de déterminer si l'on est en mode édition ou non :

```
const [editing, setEditing] = useState<boolean>(false);
```

Ajouter un bouton d'édition qui passera notre variable de state à ***true*** :

```
<Button
  type="primary"
  icon={<EditOutlined />}
  onClick={() => setEditing(true)}
/>
```

On obtient le code suivant pour l'affichage d'un étudiant :

```
<Space>
    <label>
      <b>Nom :</b>
    </label>
    {student.name}
    <label>
      <b>Prénom :</b>
    </label>
    {student.firstname}
    <label>
      <b>Age :</b>
    </label>
    {student.age}
    <Button
      type="primary"
      icon={<EditOutlined />}
      onClick={() => setEditing(true)}
    />
 </Space>
```

Edition d’un étudiant

Si l’on est en mode édition, alors on transforme l’affichage des attributs en input afin de pouvoir les modifier

On remplace également l’icône d’édition par deux boutons de confirmation et d’annulation

```
{editing ? (
    // Code du formulaire avec input avec boutons de confirmation et d'annulation
  ) : (
    // Code d'affichage des attributs d'un étudiant avec bouton d'édition
  )
}
```

Edition d'un étudiant

```
<Space>
  <label>
    <b>Nom :</b>
  </label>
  <Input
    defaultValue={student.name}
    onChange={(e) => setName(e.target.value)}
  />
  <label>
    <b>Prénom :</b>
  </label>
  <Input
    defaultValue={student.firstname}
    onChange={(e) => setFirstname(e.target.value)}
  />
  <label>
    <b>Age :</b>
  </label>
  <Input
    defaultValue={student.age}
    onChange={(e) => setAge(parseInt(e.target.value))}
  />
  <Button
    type="primary"
    icon={<CheckOutlined />}
    onClick={() => edit(student._id)}
  />
  <Button
    type="primary"
    danger={true}
    icon={<CloseOutlined />}
    onClick={() => setEditing(false)}
  />
</Space>
```

Edition d'un étudiant

Ajout d'une props au composant **<Student />**

```
type StudentProps = {
  student: StudentType;
  onEdit: (data: Partial<StudentType>) => void;
};
```

```
export const Student = ({ student, onEdit}: StudentProps) => {
```

Création d'une fonction **edit** dans le composant **<Student />**

```
const edit = (_id: string) => {
  onEdit({ _id, name, firstname, age });
  setEditing(false);
};
```

Edition d'un étudiant

Dans le composant **<Students />**, créer une fonction editStudent

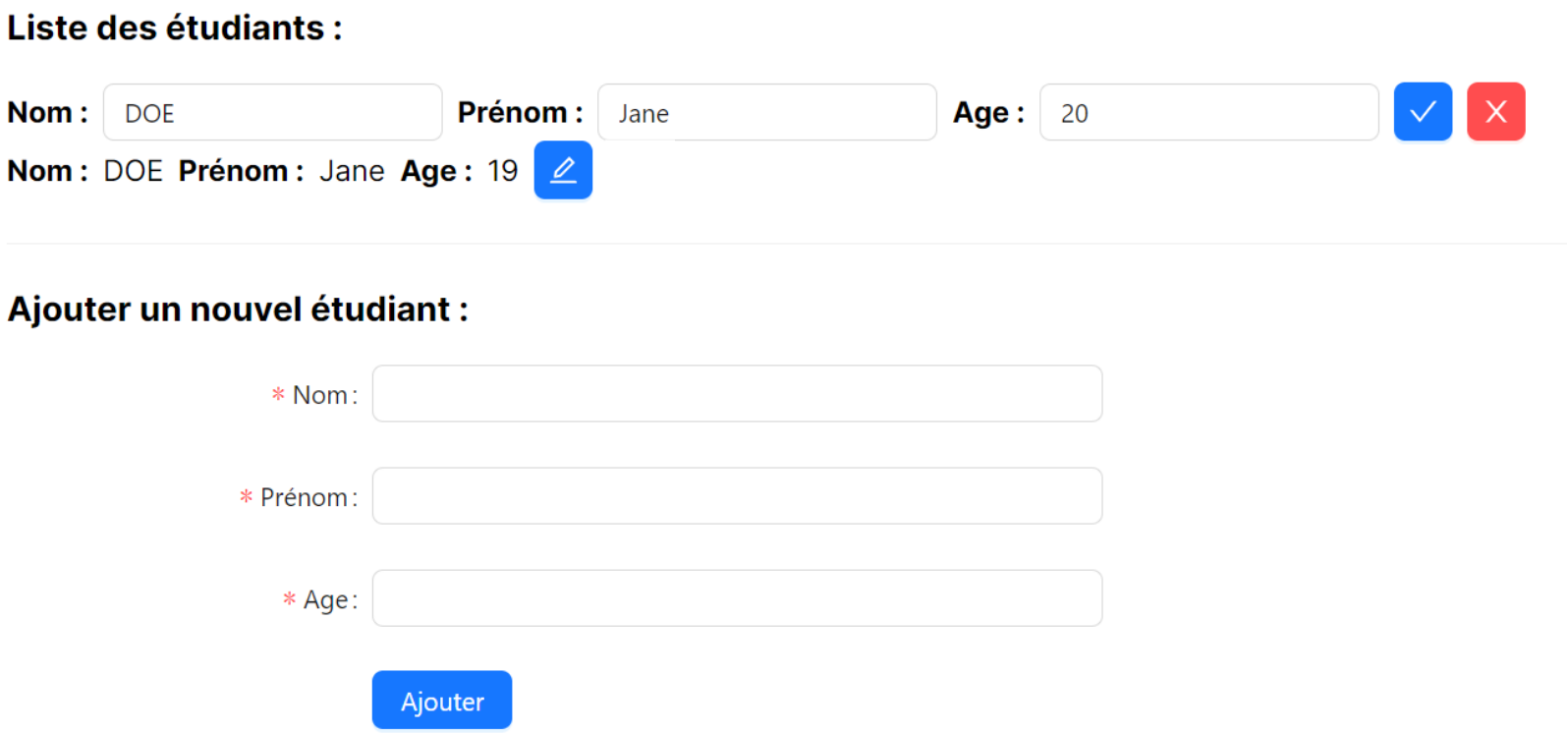
```
const editStudent = async (data: Partial<StudentType>) => {
  const updatedStudent = (
    await axios.put(`http://localhost:5000/students/${data._id}`, data)
  ).data;

  const studentsUpdated = students.map((student) => {
    if (student._id === updatedStudent._id) {
      student = updatedStudent;
    }
    return student;
  });

  setStudents(studentsUpdated);
};
```

Edition d'un étudiant

Dans le composant **<Students />**, passer cette fonction en props du composant **<Student />**

```
<Student
  key={student._id}
  student={student}
  onEdit={editStudent}
/>
```

On obtient le rendu suivant :



Suppression d'un étudiant

Dans le composant **<Student />**, ajouter un bouton de suppression après le bouton d'édition :

```
<Button
  type="primary"
  danger={true}
  icon={<DeleteOutlined />}
  onClick={() => onDelete(student._id)}
/>
```

Ajout d'une props au composant **<Student />** :

```
type StudentProps = {
  student: StudentType;
  onEdit: (data: Partial<StudentType>) => void;
  onDelete: (id: string) => void;
};
```

```
export const Student = ({ student, onEdit, onDelete }: StudentProps) => {
```

Suppression d'un étudiant

Dans le composant **<Students />**, créer une fonction deleteStudent

```
const deleteStudent = (id: string) => {
  axios.delete(`http://localhost:5000/students/${id}`);
  setStudents(students.filter((student) => student._id !== id));
};
```

Passer cette fonction en props du composant **<Student />**

```
<Student
  key={student._id}
  student={student}
  onEdit={editStudent}
  onDelete={deleteStudent}

/>
```

Suppression d'un étudiant

On obtient le rendu suivant :