

TD4

MERN Application



M



E



R



N



<https://www.jordansablons.fr/enseignement>





https://github.com/jsablonsEnseignement/td4_sources



- Objectif du TD
- Récupération du code source & installation des dépendances
- Démarrage du backend
- Installation de la librairie Axios côté frontend
- Démarrage du frontend
- Récupération et affichage de la liste des étudiants
- Formulaire de création d'un nouvel étudiant
- Edition d'un étudiant
- Suppression d'un étudiant

Objectif du TD : créer une application qui permet de gérer des étudiants

L'application permettra d'afficher la liste des étudiants, d'en créer, modifier et supprimer :

Liste des étudiants :

Nom : DOE Prénom : Jane Age : 20  

Nom : DOE Prénom : Jane Age : 19  

Ajouter un nouvel étudiant :

* Nom:



* Prénom:



* Age:

Objectif du TD : créer une application qui permet de gérer des étudiants


Au clic sur l'icône d'édition, les valeurs de la ligne correspondante seront modifiables :



Liste des étudiants :

Nom : DOE Prénom : Jane Age : 20  

Nom : DOE Prénom : Jane Age : 19  

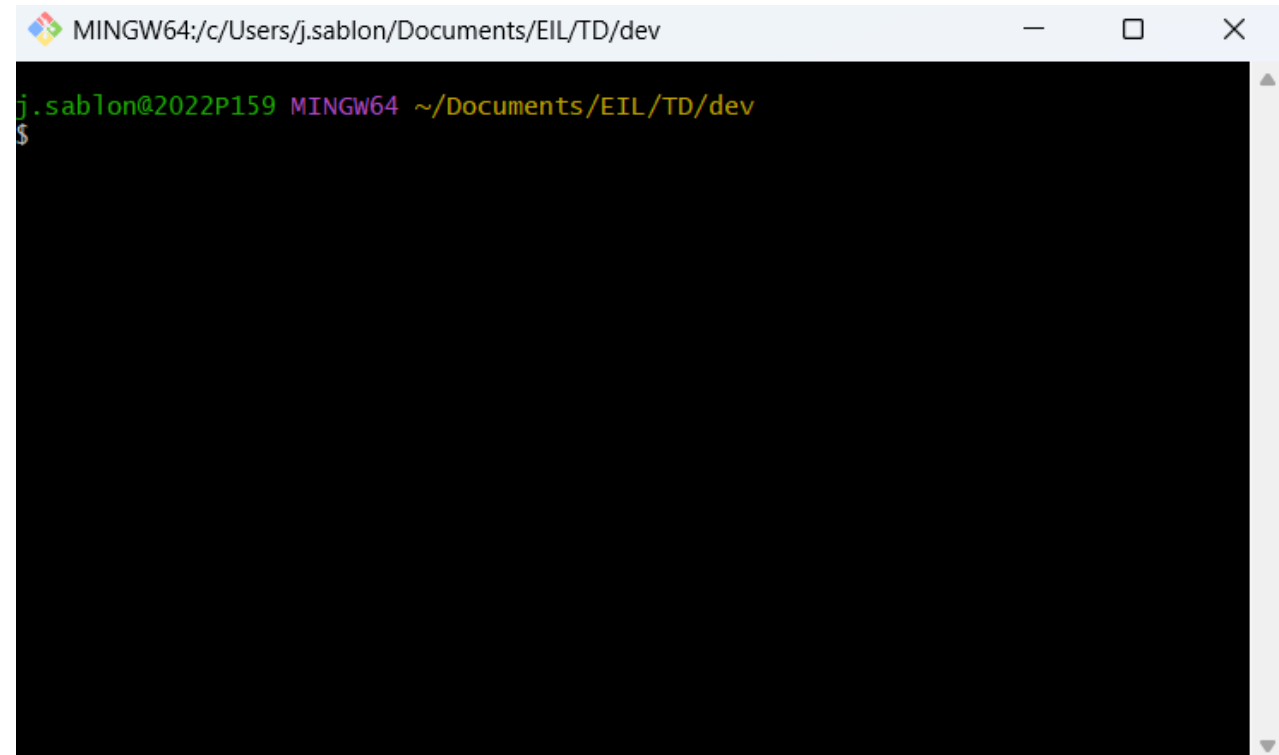
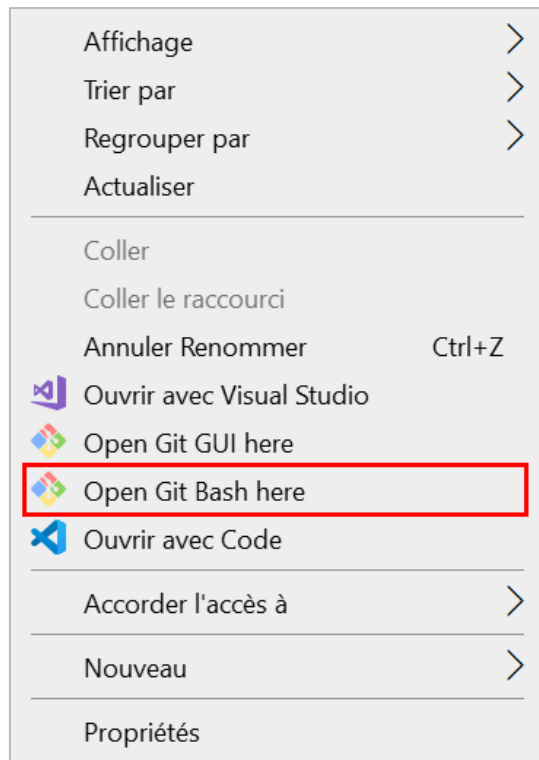
Liste des étudiants :

Nom : Prénom : Age :  

Nom : DOE Prénom : Jane Age : 19  

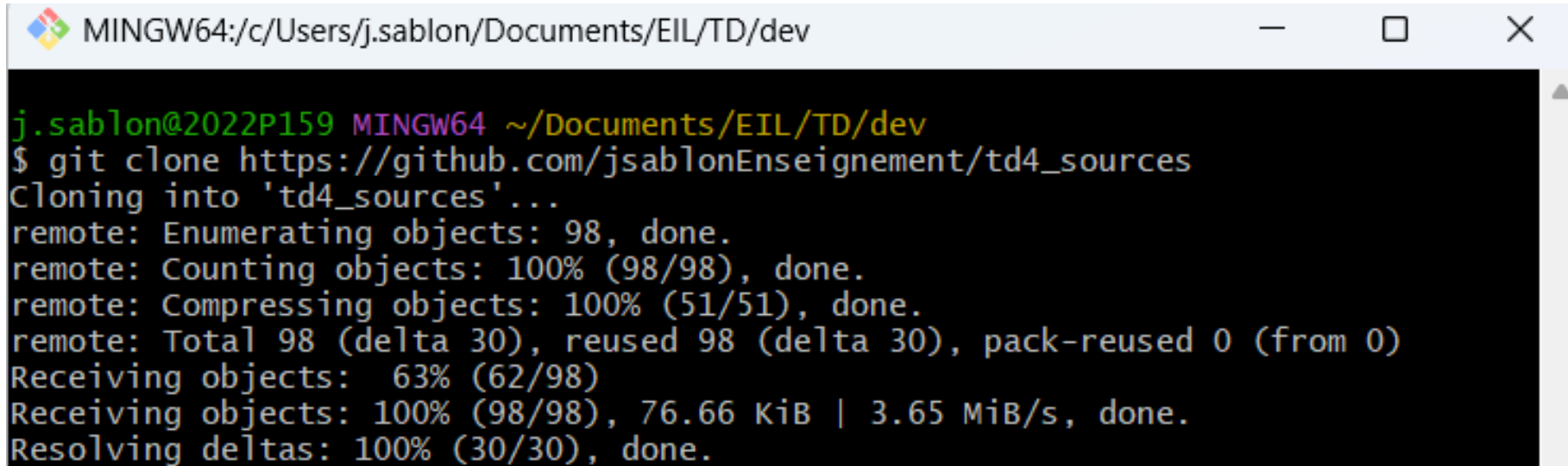
Récupération du code source du projet & installation des dépendances

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 :



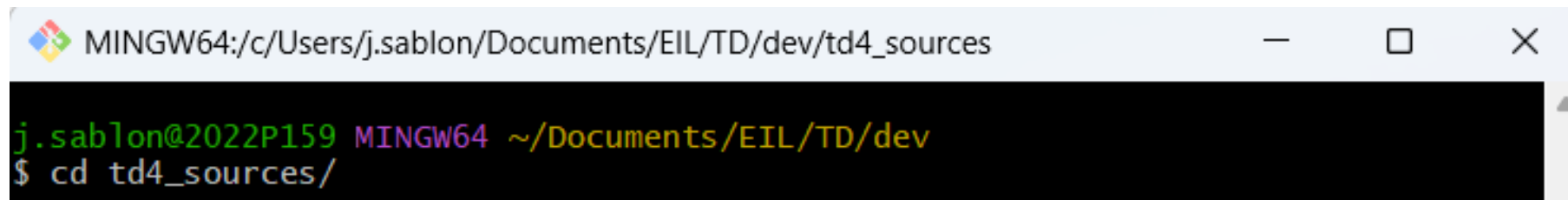
Récupération du code source du projet & installation des dépendances

3. Cloner le répertoire Git : https://github.com/jsablonEnseignement/td4_sources



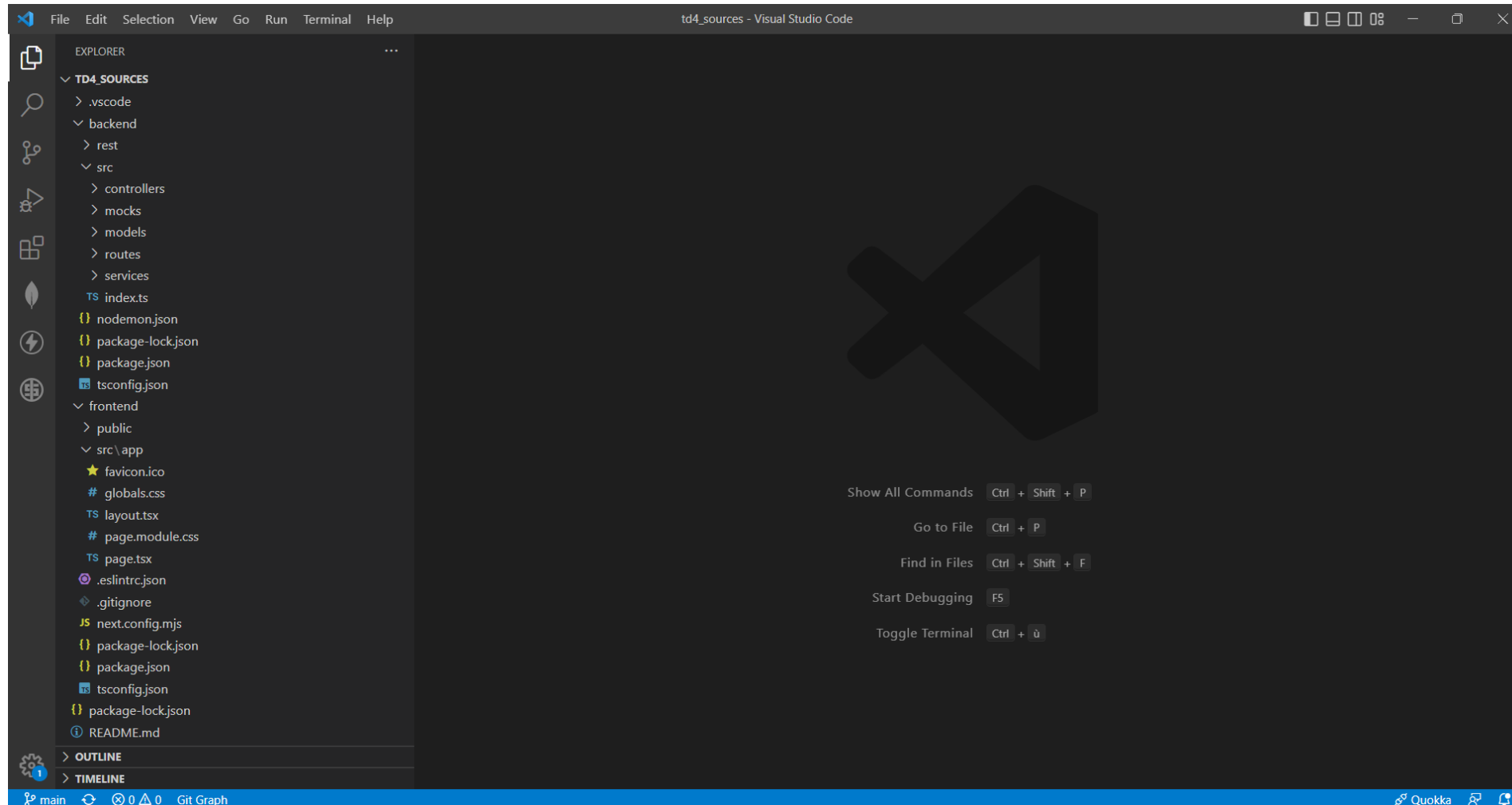
```
MINGW64:/c/Users/j.sablon/Documents/EIL/TD/dev
j.sablon@2022P159 MINGW64 ~/Documents/EIL/TD/dev
$ git clone https://github.com/jsablonEnseignement/td4_sources
Cloning into 'td4_sources'...
remote: Enumerating objects: 98, done.
remote: Counting objects: 100% (98/98), done.
remote: Compressing objects: 100% (51/51), done.
remote: Total 98 (delta 30), reused 98 (delta 30), pack-reused 0 (from 0)
Receiving objects: 63% (62/98)
Receiving objects: 100% (98/98), 76.66 KiB | 3.65 MiB/s, done.
Resolving deltas: 100% (30/30), done.
```

4. Se rendre dans le dossier précédemment cloné :

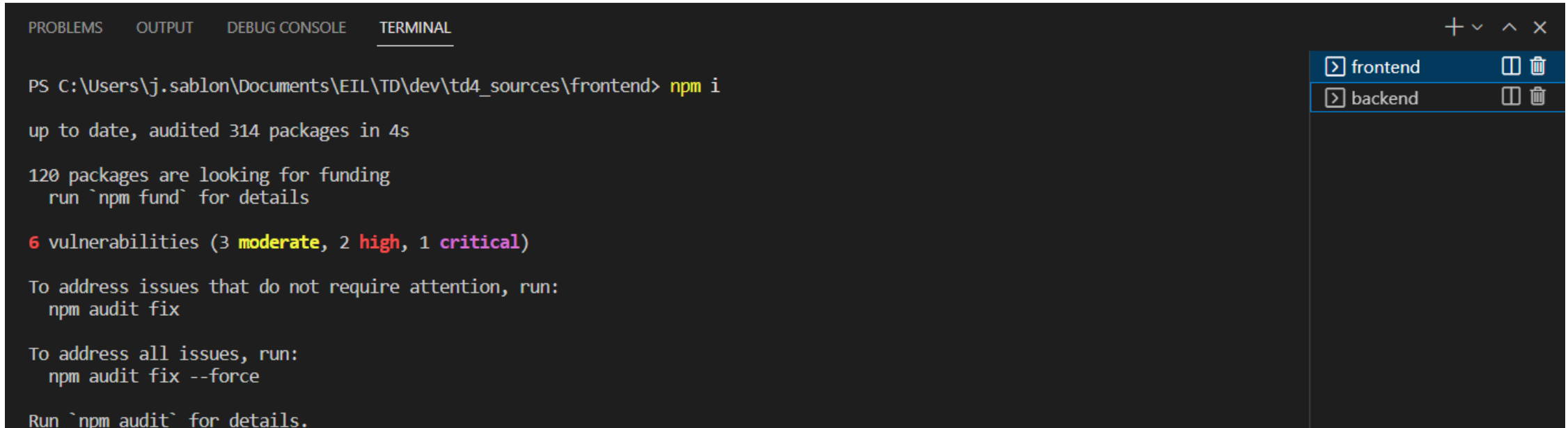


```
MINGW64:/c/Users/j.sablon/Documents/EIL/TD/dev/td4_sources
j.sablon@2022P159 MINGW64 ~/Documents/EIL/TD/dev
$ cd td4_sources/
```

5. Ouvrir le projet dans VSCode :

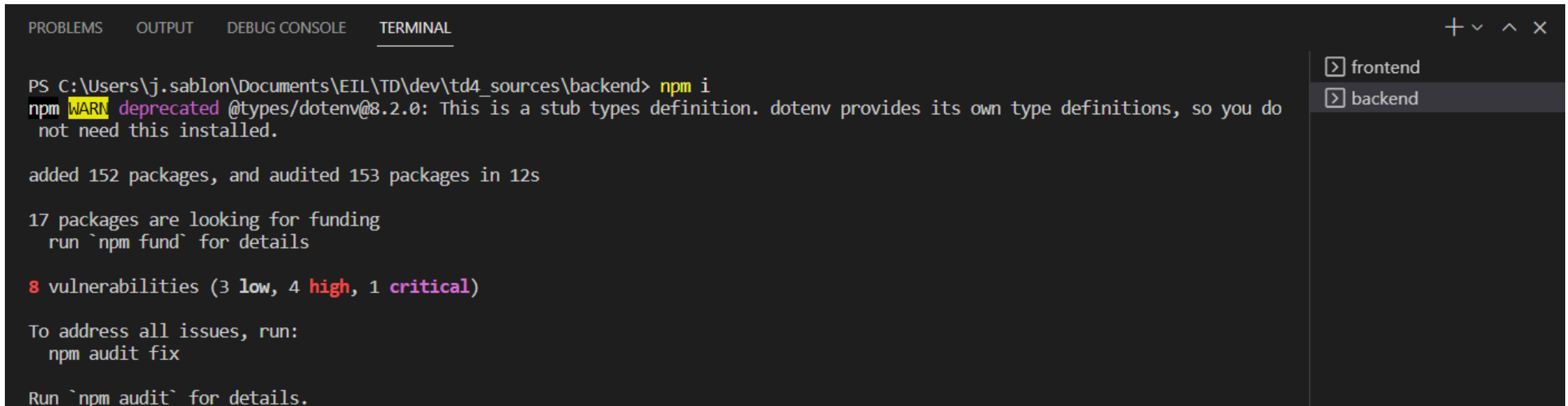


6. Depuis le répertoire frontend, installez les dépendances via la commande **npm i**



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\Users\j.sablon\Documents\EIL\TD\dev\td4_sources\frontend> npm i
up to date, audited 314 packages in 4s
120 packages are looking for funding
  run `npm fund` for details
6 vulnerabilities (3 moderate, 2 high, 1 critical)
To address issues that do not require attention, run:
  npm audit fix
To address all issues, run:
  npm audit fix --force
Run `npm audit` for details.
```

7. Depuis le répertoire backend, installez les dépendances via la commande **npm i**



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\Users\j.sablon\Documents\EIL\TD\dev\td4_sources\backend> npm i
npm WARN deprecated @types/dotenv@8.2.0: This is a stub types definition. dotenv provides its own type definitions, so you do not need this installed.

added 152 packages, and audited 153 packages in 12s

17 packages are looking for funding
  run `npm fund` for details

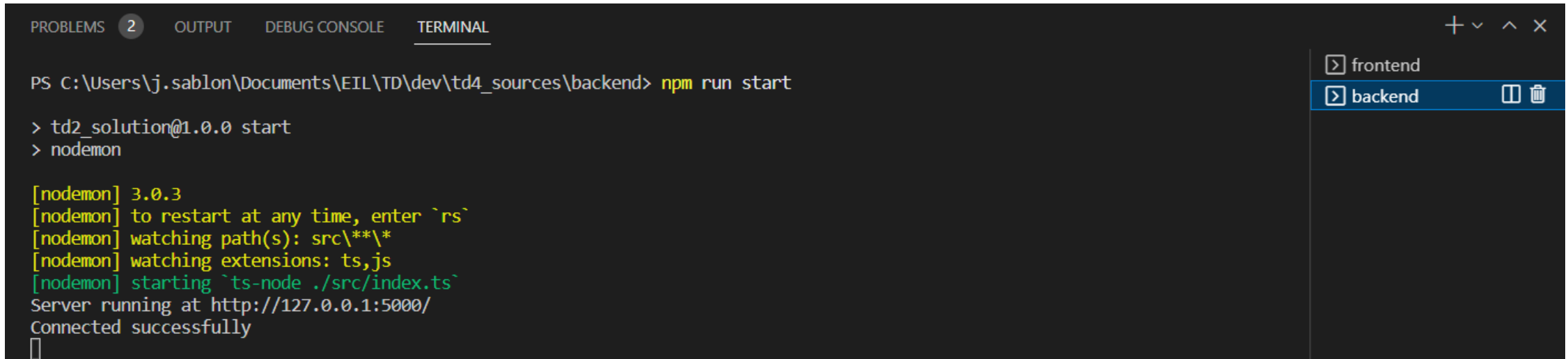
8 vulnerabilities (3 low, 4 high, 1 critical)

To address all issues, run:
  npm audit fix

Run `npm audit` for details.
```

Démarrer le backend

Depuis le dossier backend, exécuter la commande **npm run start** afin de démarrer l'application Node.js :



```
PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL
PS C:\Users\j.sablon\Documents\EIL\TD\dev\td4_sources\backend> npm run start

> td2_solution@1.0.0 start
> nodemon

[nodemon] 3.0.3
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): src\**\*
[nodemon] watching extensions: ts,js
[nodemon] starting `ts-node ./src/index.ts`
Server running at http://127.0.0.1:5000/
Connected successfully
█
```

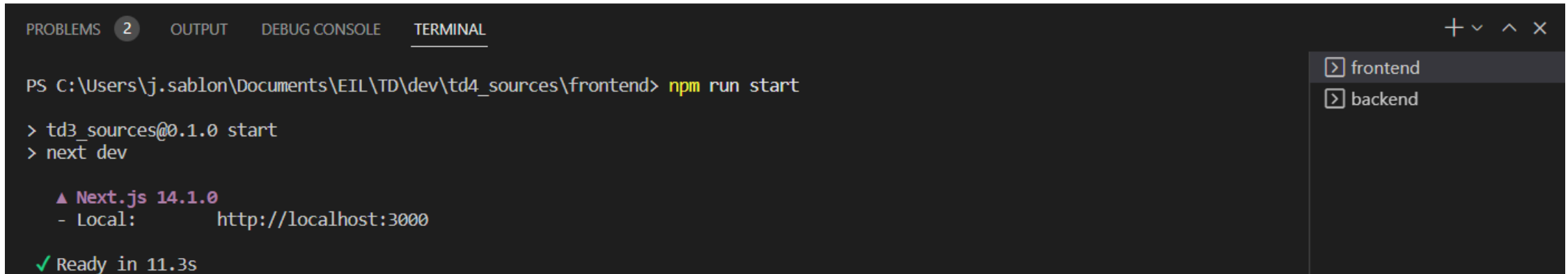
Depuis le dossier frontend, installer la dépendance **axios**

```
npm install axios
```

- ① Cette dépendance nous permettra de réaliser des requêtes HTTP au backend afin de pouvoir communiquer avec celui-ci (émission et récupération de données)

Démarrer le frontend

Depuis le dossier frontend, exécuter la commande **npm run start** afin de démarrer l'application React :



```
PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL
PS C:\Users\j.sablon\Documents\EIL\TD\dev\td4_sources\frontend> npm run start

> td3_sources@0.1.0 start
> next dev

▲ Next.js 14.1.0
- Local:      http://localhost:3000

✓ Ready in 11.3s
```

Récupération et affichage de la liste des étudiants

Côté frontend, créer un dossier **/components** dans **/src/app** qui regroupera l'ensemble de nos composants

Créer un dossier **/Students** dans le dossier **/components**

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

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

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 />;  
}
```

Récupération et affichage de la liste des étudiants

Dans le composant `<Students />`, ajouter un `useEffect` qui fera appel à une fonction `fetchStudents` qui fait une requête HTTP **GET** sur `http://localhost:5000/students` :

```
export const Students = () => {  
  
  useEffect(() => {  
    const fetchStudents = async () => {  
      const { data } = await axios.get(`http://localhost:5000/students`);  
    };  
    fetchStudents();  
  }, []);  
  
  return <>Students</>  
}
```

- ① Cette fonction sera asynchrone (`async`) avec attente du retour de l'appel HTTP (`await`) afin de pouvoir exploiter les données dès le chargement du composant
- ① On récupérera uniquement l'attribut **data** de la requête (`const { data }`), qui contient les données concernant les étudiants
- ① Pour réaliser un appel asynchrone depuis un **useEffect**, on y définit la fonction puis on y fait appel

Récupération et affichage de la liste des étudiants

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 StudentType = {  
  _id: string;  
  name: string;  
  firstname: string;  
  age: number;  
};
```

Récupération et affichage de la liste des étudiants

Dans le composant `<Students />`, ajouter (avant le `useEffect`) un `useState` qui permettra de stocker les étudiants précédemment récupérés :

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

- ① Le `useState` sera typé via le type `StudentType` précédemment créé
- ① On initialise notre variable d'état `students` avec un tableau vide par défaut

Récupération et affichage de la liste des étudiants

Dans le **useEffect** du composant **<Students />**, on vient stocker les données récupérées depuis la requête HTTP dans notre variable de state créée précédemment :

```
export const Students = () => {  
  
  useEffect(() => {  
    const fetchStudents = async () => {  
      const { data } = await axios.get(`http://localhost:5000/students`);  
      setStudents(data);  
    };  
    fetchStudents();  
  }, []);  
  
  return <>Students</>  
}
```

- ① On pourra désormais utiliser notre variable **students** pour manipuler les données dans notre composant (afficher les étudiants par exemple)

Récupération et affichage de la liste des étudiants

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

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

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

type StudentProps = {
  student: StudentType;
};

export const Student = ({ student }: 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>
  );
};
```

Récupération et affichage de la liste des étudiants

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>
  )}
</>
```

Récupération et affichage de la liste des étudiants

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

```
import axios from "axios";
import { useEffect, useState } from "react";
import { Student } from "../Student/Student";
import { 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>
      )}
    </>
  );
};
```

On obtient le rendu suivant :

Liste des étudiants :

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

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

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

Créer dans ce dossier un composant **AddStudent.tsx**

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

Formulaire de création d'un nouvel étudiant

Exemple (à compléter avec les différents attributs d'un étudiant) :

```
<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>
```

- ① On la fonction native en JavaScript **parseInt()** pour convertir la valeur de l'âge en nombre

Formulaire de création d'un nouvel étudiant

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;  
};
```

Formulaire de création d'un nouvel étudiant

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'une fonction addStudent :

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

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

On obtient le rendu suivant :

Liste des étudiants :

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

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

Ajouter un nouvel étudiant :

* Nom:

* Prénom:

* Age:

Ajouter

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>
```

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  
)  
}
```

```
<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>
```

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);  
};
```

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

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

  setStudents(students);
};
```

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 :

Liste des étudiants :

Nom : Prénom : Age :

Nom : DOE Prénom : Jane Age : 19

Ajouter un nouvel étudiant :

* Nom:

* Prénom:

* Age:

Ajouter

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 = async (id: string) => {
  const students = (
    await axios.delete(`http://localhost:5000/students/${id}`)
  ).data;

  setStudents(students);
};
```





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

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

/>
```

On obtient le rendu suivant :

Liste des étudiants :

Nom : DOE Prénom : Jane Age : 20  
Nom : DOE Prénom : Jane Age : 19  

Ajouter un nouvel étudiant :

* Nom:

* Prénom:

* Age:

Ajouter