FRONT-END SOFTBINATOR-LABS

# CURS 6: HOOKS

## and Context

# CUSTOM HOOKS

useAnything

# Idei principale

- Custom Hooks sunt un mod simplu prin care putem refolosi logica
- Prin convenție încep cu use
- Sunt definite ca funcții, dar:
  - Se comporta ca niște componente funcționale React ce returnează date (nu JSX, ar fi antipattern)
  - Un hook poate folosi alte hook-uri în definiție (precum useState)
  - Fiecare instanță este individuală (are datele proprii)

# useGreeting

```
export const useGreetings = (name) => {
  return `Hello ${name}! How's it going?`
};
```

```
function App() {
  const greeting = useGreetings( name: 'Greg');

  return (
    <div className="App">
      <h1>{greeting}</h1>
    </div>
  );
}
```

# useGreeting

- hooks se apelează în cadrul componentei functionale
- Valoarea întoarsă se poate folosi direct pentru render

```
export const useGreetings = (name) => {
  return `Hello ${name}! How's it going?`
};
```

```
function App() {
  const greeting = useGreetings( name: 'Greg');

  return (
    <div className="App">
      <h1>{greeting}</h1>
    </div>
  );
}
```

**Hello Greg! How's it going?**

# Instanțe individuale

- Putem avea un hook ce ține și updatează datele unei entități

```
export const useSomeData = (someData :{createDate: number, name: string} = {
  name: 'Default',
  createDate: new Date().getDate()
}) => {
  const [data, setData] = useState(someData);

  1 usage   Ionut Catalin Tatuta *
  const updateName = (newName) => {
    setData( value: {...data, name: newName})
  };

  return ({data, updateName});
};
```

# Instanțe individuale

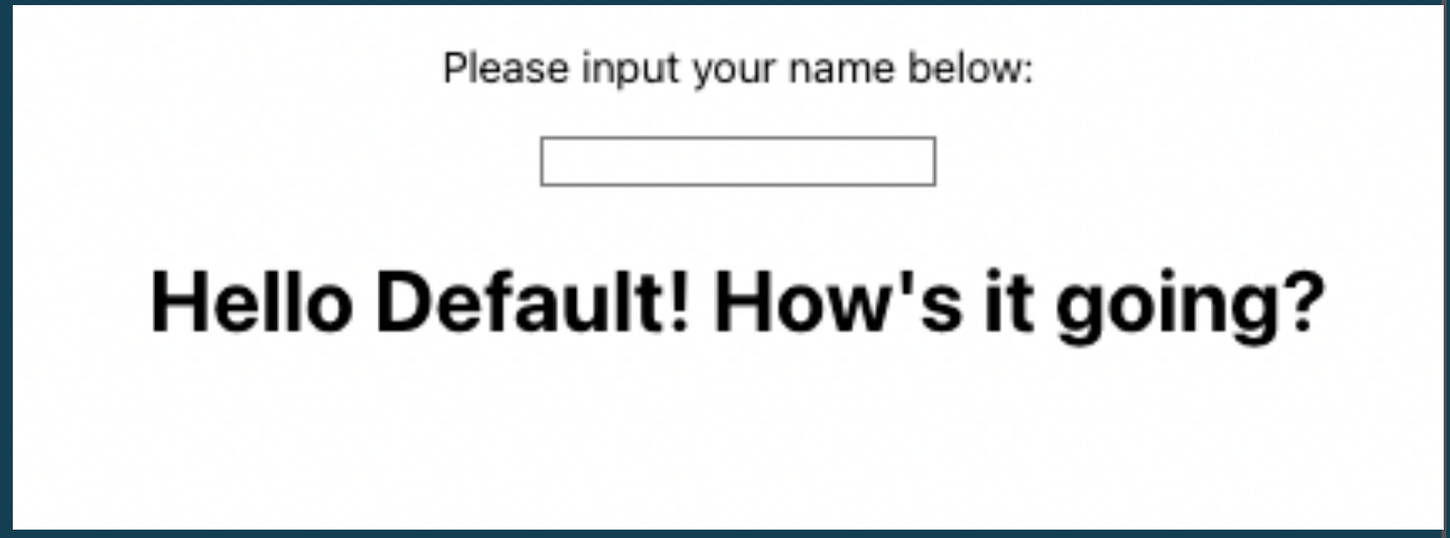
- Și două componente ce îl folosesc:

```
const InputComponent = () => {
  const {data, updateName} = useSomeData();

  return (
    <div>
      <p>
        Please input your name below:
      </p>
      <input
        type="text"
        onInput={(e : FormEvent<HTMLInputElement> ) => {
          updateName(e.target.value);
          console.log(data.name);
        }}
      />
    </div>
  )
}
```

```
const GreetingComponent = () => {
  const {data} = useSomeData();

  const greeting = useGreetings(data.name);

  return (
    <h1>{greeting}</h1>
  )
}
```

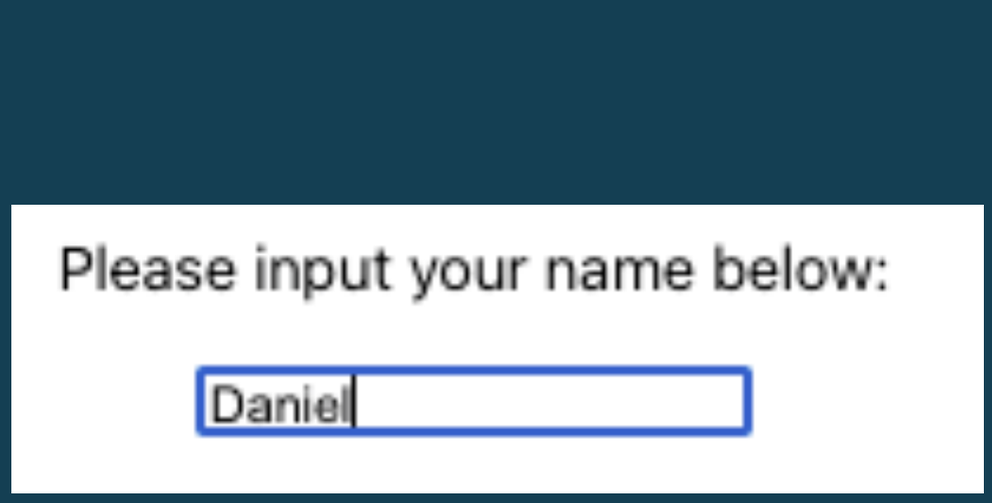
# Instanțe individuale

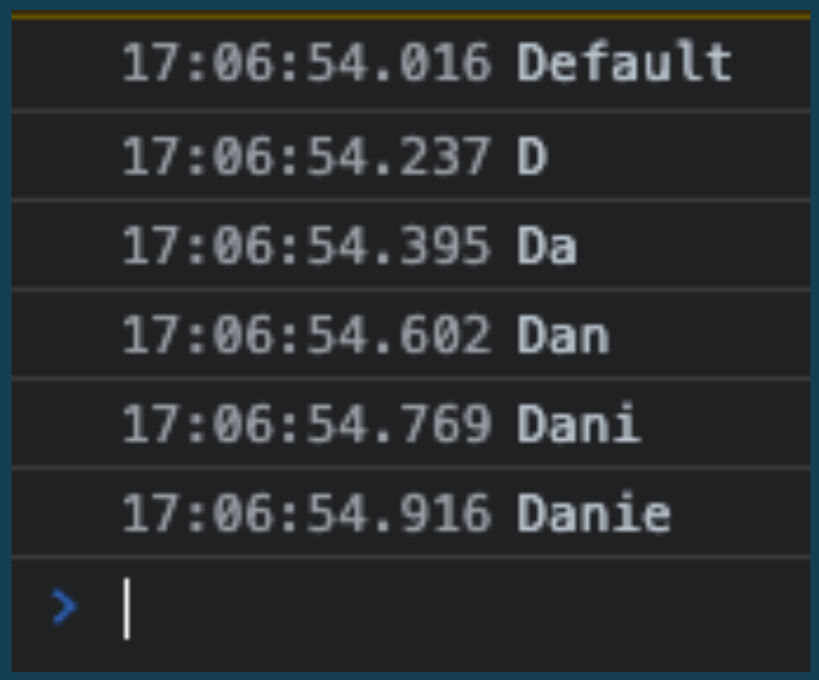
```
function App() {

  return (
    <div className="App">
      <InputComponent />
      <GreetingComponent />
    </div>
  );
}
```



# Instanțe individuale

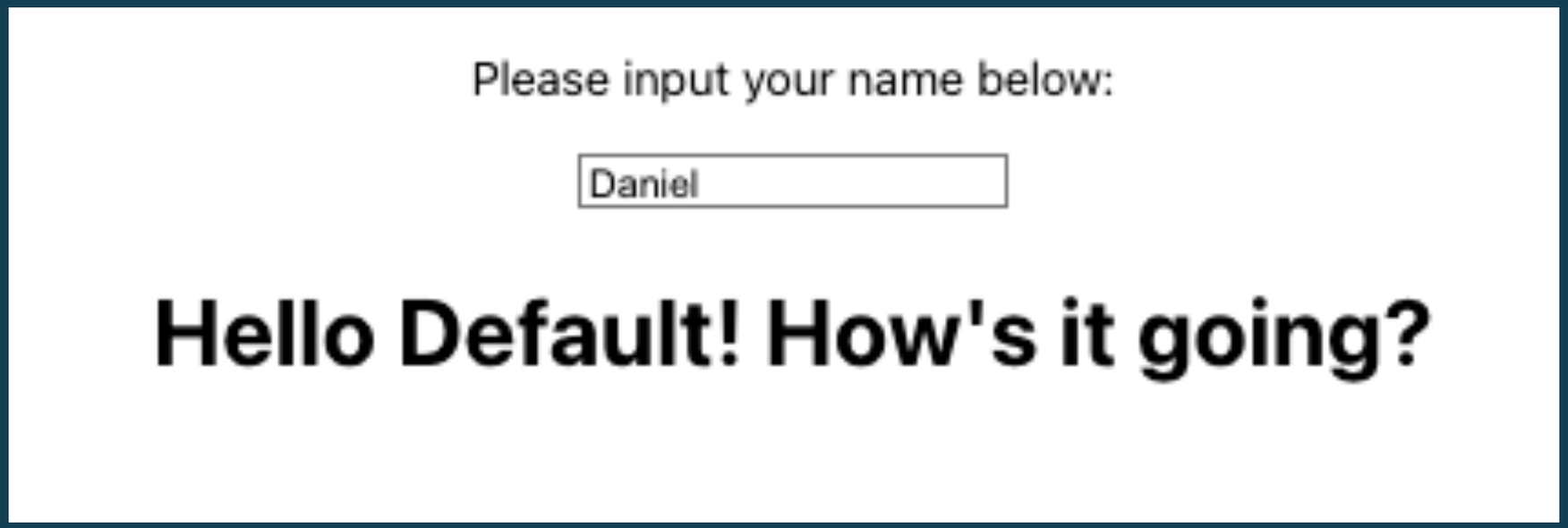
- Dacă introducem numele nou în InputComponent, putem observa că datele se updatează în cadrul componentei:

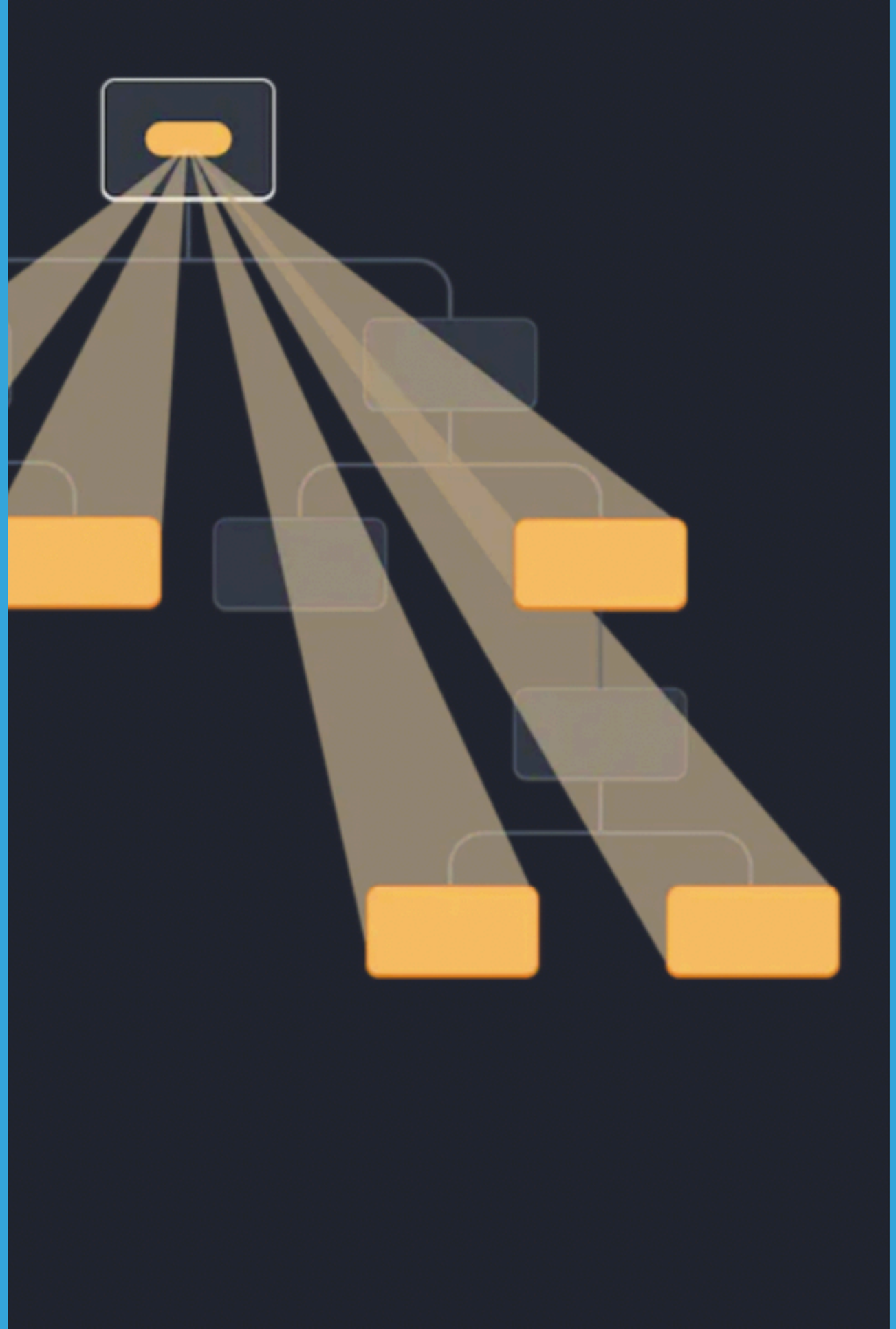


# Instanțe individuale

- Însă nu și în cadrul GreetingComponent





# CONTEXT

## everyone's in on it

# Idei principale

- Context permite unei componente părinte să dea acces tuturor copiilor la un set de date
- Folosind context putem evita prop drilling (trimiterea explicită a datelor si metodelor prin props pe fiecare nivel)
- Pentru a-l folosi, avem nevoie de două părți:
  - Context
  - Provider

# createContext

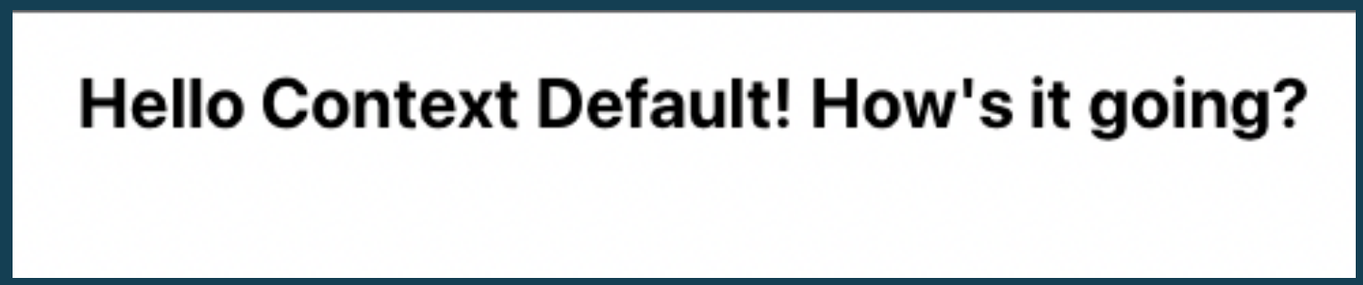
- Pentru a crea un Context-ul folosim createContext

```
import {createContext} from "react";
2 usages    Ionut Catalin Tatuta *
export const DataContext = createContext( defaultValue: {
  name: 'Context Default',
  createDate: new Date().getDate()
})
```

# createContext

- Pentru a citi Context-ul folosim useContext
- Dacă ne oprim aici, mereu va fi folosită valoarea default

```
const GreetingComponent = () => {
  const {name} = useContext(DataContext);

  const greeting = useGreetings(name);

  return (
    <h1>{greeting}</h1>
  )
}
```



# Provider

- Provider-ul se obține din Context-ul creat prin ContextName.Provider
- Acesta e o Componentă cu un singur prop: value, ce va determina valoarea datelor văzute de copii

```
function App() {

  return (
    <div className="App">
      <DataContext.Provider value={{name: 'Johnny'}} >
        <GreetingComponent />
      </DataContext.Provider>
    </div>
  );
}
```

# Custom Provider

- Pentru a putea schimba valorile, putem defini un Provider în care folosim useState

```
export const DataProvider = ({value, children}) => {
  const [data, setData] = useState(value);
  no usages   new *
  const updateName = (newName) => {
    setData( value: {...data, name: newName})
  };

  return (
    <DataContext.Provider
      value={{
        ...data,
        updateName
      }}
    >
      {children}
    </DataContext.Provider>
  )
}
```

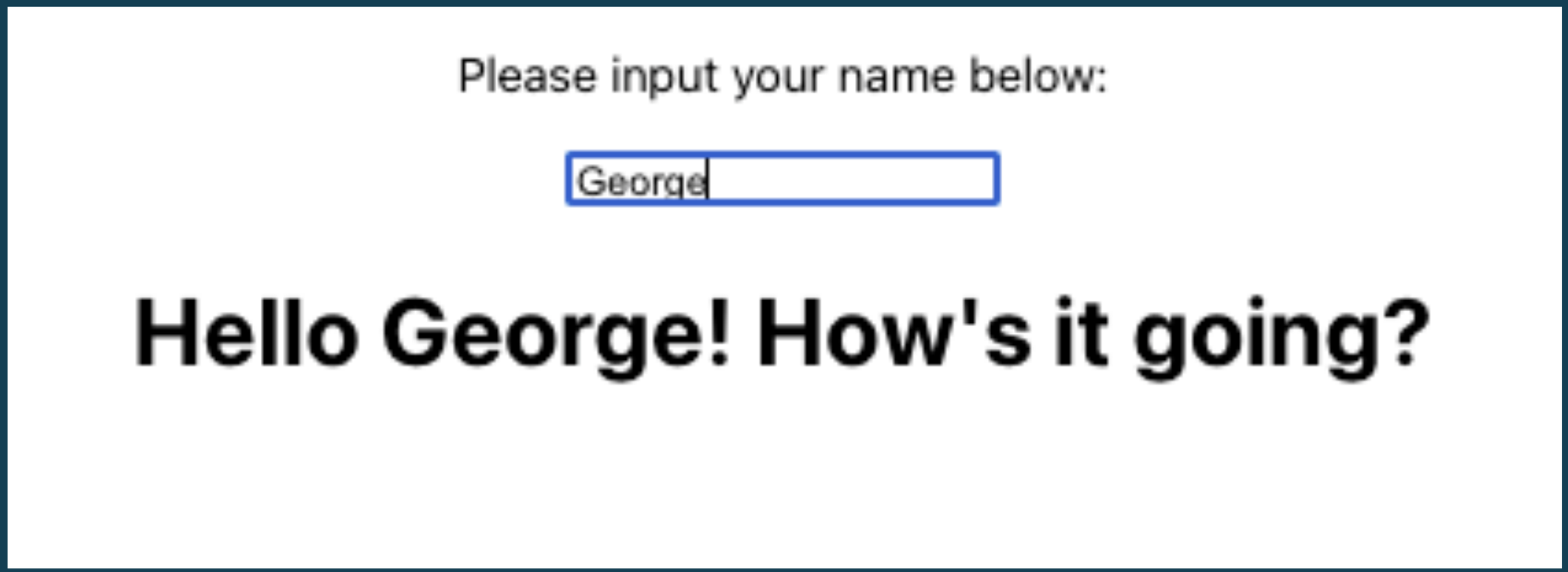
# Custom Provider

- Putem updata InputComponent să folosească Context
- Cele două componente vor folosi acum aceleași date

```
const InputComponent = () => {
  const {updateName, name} = useContext(DataContext);
  return (
    <div>
      <p>
        Please input your name below:
      </p>
      <input
        type="text"
        onInput={(e : FormEvent<HTMLInputElement> ) => {
          updateName(e.target.value);
          console.log(name);
        }}
      />
    </div>
  )
}
```

```
function App() {

  return (
    <div className="App">
      <DataProvider value={{name: 'Johnny'}} >
        <InputComponent />
        <GreetingComponent />
      </DataProvider>
    </div>
  );
}
```

# Custom Provider



Link-uri utile

- Custom Hooks
  https://react.dev/learn/reusing-logic-with-custom-hooks
- Context
  https://react.dev/learn/passing-data-deeply-with-context
  https://react.dev/reference/react/useContext