ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS
ΚΕΝΤΡΟ ΕΠΙΜΟΡΦΩΣΗΣ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗΣ

React JS
State Management Μέρος 3ο

Δαμιανός Μαρινάτος

useReducer (1)

- Το useReducer είναι **ένα React Hook για διαχείριση πιο σύνθετης κατάστασης (state)**, παρόμοια με το useState, αλλά πιο οργανωμένο και πιο προβλέψιμο όταν:
  - έχουμε **πολλά πεδία** κατάστασης (π.χ. count, lastAction, time),
  - η κατάσταση αλλάζει με **πολύπλοκους τρόπους**,
  - θέλουμε να **συγκεντρώσεις** τη λογική των αλλαγών σε **μία κεντρική συνάρτηση** (reducer).

useReducer (2)

Το useReducer χρειάζεται:

- **reducer**: Μια συνάρτηση που παίρνει το state και ένα action και επιστρέφει το νέο state.
- **initialState**: Η αρχική τιμή της κατάστασης.
- **dispatch**: Μια συνάρτηση για να "στείλουμε" εντολές (actions) που θα αλλάξουν την κατάσταση.

```
const [state, dispatch] = useReducer(reducer, initialState);
```

useReducer (3)

Το useReducer χρειάζεται:

- **reducer**: Μια συνάρτηση που παίρνει το state και ένα action και επιστρέφει το νέο state.
- **initialState**: Η αρχική τιμή της κατάστασης.
- **dispatch**: Μια συνάρτηση για να "στείλουμε" εντολές (actions) που θα αλλάξουν την κατάσταση.

```
const [state, dispatch] = useReducer(reducer, initialState);
```

Παράδειγμα Counter (1)


```tsx
import {useCounterWithReducer} from "../hooks/useCounterWithReducer.ts"
import CounterButton from "./CounterButton.tsx";

const CounterWithCustomHook : () => Element = () : Element => { Show usages new *
  const { count, lastAction, time, increase, decrease, reset } = useCounterWithReducer();

  return (
    <>
      <h1 className="text-center text-2xl">Count is {count}</h1>
      <div className="text-center space-x-4 pt-8">
        <CounterButton onClick={increase} label="Increase" />
        <CounterButton onClick={decrease} label="Decrease" disabled={count === 0} />
        <CounterButton onClick={reset} label="Reset" disabled={count === 0} />
      </div>
      <p className="text-center text-cf-gray pt-8">
        Last change: <strong>{lastAction || "-"}</strong> at{" "}
        <strong>{time || "-"}</strong>
      </p>
    </>
  );
};

export default CounterWithCustomHook; Show usages new *
```

- Από το παράδειγμα του προηγουμένου μαθήματος θα κρατήσουμε ίδιο όλο τον κώδικα με την διαφορά ότι αυτή την φορά θα καλέσουμε ένα νέο custom hook (useCounterWithReducer) που θα φτιάξουμε κάνοντας χρήση του **useReducer**.

Παράδειγμα Counter (2)


```tsx
import { useReducer } from "react";

type CounterState = {
  count: number;
  lastAction: string;
  time: string;
};

type Action =
  | { type: "INCREASE" }
  | { type: "DECREASE" }
  | { type: "RESET" };

const getCurrentTime : () => string  = () : string  => new Date().toLocaleTimeString();

const initialState: CounterState = {
  count: 0,
  lastAction: "",
  time: "",
};
```

- Δημιούργει έναν τύπο για το State του component όπου περιλαμβάνει 3 πεδία.
- Ο τύπος **actions** λέει τι **ενέργειες** μπορεί να δεχτεί ο reducer. Τα actions είναι JavaScript αντικείμενα με πεδίο type, που καθορίζει τι αλλαγή θα γίνει στο state.
- Το **initialState** είναι το **αρχικό state** που θα φορτωθεί όταν το component ή το hook αρχικοποιηθεί.

Παράδειγμα Counter (3)


```
function reducer(state: CounterState, action: Action): CounterState {
  switch (action.type) {
    case "INCREASE":
      return {
        count: state.count + 1,
        lastAction: "Increase",
        time: getCurrentTime(),
      };
    case "DECREASE":
      return state.count > 0
        ? {
          count: state.count - 1,
          lastAction: "Decrease",
          time: getCurrentTime(),
        }
        : state;
    case "RESET":
      return {
        count: 0,
        lastAction: "Reset",
        time: getCurrentTime(),
      };
    default:
      return state;
  }
}
```

- Η reducer είναι μια **καθαρή συνάρτηση** (pure function) που παίρνει το **τρέχον state** και το **action** και **επιστρέφει το νέο state** ανάλογα με το είδος της ενέργειας.
- Αν το **action.type** δεν ταιριάζει με καμία από τις γνωστές περιπτώσεις, **επιστρέφει το ίδιο state χωρίς αλλαγές**. Είναι απαραίτητο για να διασφαλιστεί ότι ο κώδικας δεν θα σπάσει αν σταλεί άγνωστη ενέργεια.

Παράδειγμα Counter (4)


```
export const useCounterWithReducer : () => { count: number; lastAction: string... = () : { count: number; lastAction: string; time... => {

  const [state, dispatch] = useReducer(reducer, initialState);

  const increase : () => void = () : void => dispatch({ type: "INCREASE" }); Show usages new *
  const decrease : () => void = () : void => dispatch({ type: "DECREASE" }); Show usages new *
  const reset : () => void = () : void => dispatch({ type: "RESET" }); Show usages new *

  return {
    count: state.count,
    lastAction: state.lastAction,
    time: state.time,
    increase,
    decrease,
    reset,
  };
};
```

- Η συνάρτηση αυτή χρησιμοποιεί το hook: **useReducer**.
- Κάθε μια από τις συναρτήσεις increase, decrease, reset **στέλνει μια action** στον reducer.
- Το hook επιστρέφει τις **τιμές** του state (count, lastAction, time) και τις **συναρτήσεις** για την αλλαγή του state (increase, decrease, reset

Παράδειγμα Todo List (1)


```tsx
import { useReducer } from "react";
import ToDoForm from "./ToDoForm";
import ToDoList from "./ToDoList";

type Todo = {
  id: number;
  text: string;
};

type Action =
  | { type: "ADD"; payload: string }
  | { type: "DELETE"; payload: number };

const todoReducer : (state: Todo[], action: Action) => Todo[]  = (state: Todo[], action: Action): Todo[] => {
  switch (action.type) {
    case "ADD":
      const newTodo: Todo = {
        id: Date.now(),
        text: action.payload,
      };
      return [...state, newTodo];
    case "DELETE":
      return state.filter(todo : Todo  => todo.id !== action.payload);
    default:
      return state;
  }
};
```

/Components/ToDo/ToDo.tsx - 1/2

- Εισάγουμε το Hook **useReducer** από την βιβλιοθήκη της React.
- Εισάγουμε τα component **ToDoForm** και **ToDoList.**
- Ορίζουμε τον **τύπο Todo**.
- Ορίζουμε τον **τύπο Action** (Discriminated Union) που καθορίζει τις επιτρεπόμενες ενέργειες (actions) και τα δεδομένα που τις συνοδεύουν (payload).
- Η συνάρτηση **todoReducer** δέχεται την τρέχουσα κατάσταση (state) και μια ενέργεια (action), επιστρέφει νέα κατάσταση.
- Η ενέργεια **ADD** δημιουργεί νέα εργασία και την προσθέτει στην υπάρχουσα λίστα.
- Η ενέργεια **DELETE** αφαιρεί μια εργασία βάσει του id που περνάει στο payload

Παράδειγμα Todo List (2)


```
const ToDo : () => Element = () : Element => { Show usages  new *
  const [todos, dispatch] = useReducer(todoReducer, []);

  return (
    <div className="p-6 max-w-md mx-auto">
      <h1 className="text-2xl font-bold mb-4 text-center">To-Do List</h1>
      <ToDoForm dispatch={dispatch} />
      <ToDoList todos={todos} dispatch={dispatch} />
    </div>
  );
};

export default ToDo; Show usages  new *
```

/Components/ToDo/ToDo.tsx - 2/2

- Ορίζουμε το component **ToDo**
- Χρησιμοποιούμε το Hook **useReduce** για διαχείριση του state.
- Δηλώνουμε την reducer function (**todoReducer**), την αρχική κατάσταση (**[]**), την τρέχουσα κατάσταση (**todos**) και την συνάρτηση για αποστολή ενεργειών(**dispatch**).
- Περνάει την τρέχουσα λίστα (**todos**) και την συνάρτηση **dispatch** μέσα στα components

Παράδειγμα Todo List (3)


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

type Action =
  | { type: "ADD"; payload: string }
  | { type: "DELETE"; payload: number };

type TodoFormProps = {
  dispatch: React.Dispatch<Action>;
};

const ToDoForm : ({ dispatch }: TodoFormProps) => Element  = ({ dispatch }: TodoFormProps) : Element  => {  Show usages  new *
  const [text, setText] = useState( initialState: "");

  const handleChange : (e: ChangeEvent<HTMLInputElement>) => voi...  = (e: React.ChangeEvent<HTMLInputElement>) : void  => {
    setText(e.target.value);
  };

  const handleSubmit : (e: FormEvent<Element>) => void  = (e: React.FormEvent) : void  => {  Show usages  new *
    e.preventDefault();
    if (text.trim() !== "") {
      dispatch({ type: "ADD", payload: text });
      setText( value: "");
    }
  };
```

/Components/ToDo/ToDoForm.tsx - 1/2

- Ορίζουμε τους τύπους για τις ενέργειες (**actions**) και τα props του component.
- Το Component δέχεται ως prop μόνο τη συνάρτηση **dispatch**.
- Διατηρούμε το κείμενο του input σε τοπικό state μέσω του Hook **useState**.
- Η συνάρτηση **handleChange** ενημερώνει το state (text) κάθε φορά που αλλάζει το input.
- Η συνάρτηση **handleSubmit** στέλνει την ενέργεια "ADD" όταν υποβάλλεται η φόρμα και καθαρίζει το input μετά την υποβολή.

Παράδειγμα Todo List (4)


```
  return (
    <form onSubmit={handleSubmit} className="flex gap-2 mb-4">
      <input
        type="text"
        value={text}
        onChange={handleChange}
        placeholder="New task..."
        className="flex-1 border p-2 rounded"
      />
      <button
        type="submit"
        disabled={text.trim() === ""}
        className="bg-cf-dark-gray text-white px-4 py-2 rounded"
      >
        Add
      </button>
    </form>
  );
};
```

/Components/ToDo/ToDoForm.tsx - 2/2

- Στη φόρμα δηλώνουμε τις συναρτήσεις **handleChange** και **handleSubmit** και απενεργοποιούμε το button όταν το κείμενο είναι κενό.

Παράδειγμα Todo List (5)


```tsx
type Todo = {
  id: number;
  text: string;
};

type TodoListProps = {
  todos: Todo[];
  dispatch: React.Dispatch<{ type: "DELETE"; payload: number }>;
};

const ToDoList : ({ todos, dispatch }: TodoListProps) => E... = ({ todos, dispatch }: TodoListProps) : Element => {

  const handleDelete : (id: number) => () => void = (id: number) : () => void => () : void => { Show usages  new *
    dispatch({ type: "DELETE", payload: id });
  };

  return (
    <ul className="space-y-2">
      {todos.map(todo : Todo => (
        <li key={todo.id} className="flex justify-between items-center bg-gray-100 p-2 rounded">
          <span>{todo.text}</span>
          <button
            onClick={handleDelete(todo.id)}
            className="text-cf-dark-red hover:underline"
          >
            Delete
          </button>
        </li>
      ))}
    </ul>
  );
};

export default ToDoList; Show usages  new *
```

/Components/ToDo/ToDoList.tsx

- Ορίζουμε τους τύπους για τις ενέργειες (**actions**) και τα props του component.
- Η συνάρτηση **handleDelete** δέχεται το **id** της λίστας και το εισάγει στην συνάρτηση dispatch που καλεί την ενέργεια **DELETE**.
- Στο JSX εμφανίζονται τα todos σε μορφή λίστας κάνοντας χρήση του map() της Javascript.