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

React JS
useEffect Hook

Δαμιανός Μαρινάτος

useEffect (1)

- Το **useEffect** είναι ένα Hook της React που μας επιτρέπει να τρέχουμε κώδικα όταν αλλάζει κάτι στο component – π.χ. όταν κάνει render ή όταν αλλάζει κάποιο state.
- **Πού χρησιμοποιείται το useEffect;**
  - Για **fetch δεδομένων** από API
  - Για **σύνδεση / αποσύνδεση** listeners (π.χ. event listeners)
  - Για **τοπική αποθήκευση** (localStorage)
  - Για **καθαρισμό πόρων** όταν φεύγει το component
  - Για **επικοινωνία με εξωτερικά συστήματα** (timers, websockets, κ.λπ.)

useEffect (2)

Κλασικό lifecycle	Lifecycle με useEffect
componentDidMount	useEffect(fn, [])
componentDidUpdate	useEffect(fn, [deps])
componentWillUnmount	return cleanup function

useEffect - Parameters


```
useEffect(setup, [dependencies]);
```

- Η συνάρτηση **setup** εκτελείται όταν το component προστεθεί στο DOM (mount) και κάθε φορά που αλλάζει κάποιο dependency.
- Τα **dependencies** (πίνακας) καθορίζει πότε θα ξανατρέξει ξανά το side effect.

useEffect - Depedencies (1)


```
useEffect(setup, [depOne, depTwo]);
```

- **Τρέχει το effect στο πρώτο render ΚΑΙ κάθε φορά που αλλάζει κάτι στο array των dependencies.**
- Δηλαδή:
  - Στο mount (πρώτη φορά)
  - Και κάθε φορά που κάποια από τις τιμές στο [dependencies] αλλάξει
  - **Ιδανικό** όταν θες το effect να τρέχει όταν αλλάζει συγκεκριμένο state ή prop
    (π.χ. το όνομα του χρήστη, ένα φίλτρο, ένα query κ.λπ.)

useEffect - Depedencies (2)


```
useEffect(setup, []);
```

- **Τρέχει το effect ΜΟΝΟ στο πρώτο render!**
- Δηλαδή όταν φορτώσει το component για πρώτη φορά (mount) και ποτέ ξανά.
- **Ιδανικό** για fetch δεδομένων, εγγραφές σε event listeners, αρχικές ρυθμίσεις (σαν το componentDidMount).

useEffect - Depedencies (3)


```
useEffect(setup);
```

- **Τρέχει το effect σε κάθε render!**
- Δηλαδή κάθε φορά που γίνεται render το component (λόγω state/props/force render), εκτελείται η συνάρτηση setup.
- **Σπάνια χρήση** – Συνήθως δεν το θέλουμε, γιατί μπορεί να κάνει αχρείαστες κλήσεις ή side effects.

useEffect - Cleanup (1)


```
useEffect(() => {
  // setup (runs side-effect)
  return () => {
    // cleanup (clean up side-effect)
  };
}, [deps]);
```

- Όταν η setup function του useEffect επιστρέφει μια συνάρτηση, αυτή η συνάρτηση ονομάζεται **cleanup**.
- Το **setup** τρέχει αμέσως μετά το **mount**.
- Το **cleanup** τρέχει λίγο πριν το component **unmount** (ή πριν ξανατρέξει το effect λόγω αλλαγής dependencies).

useEffect - Cleanup (2)


```
useEffect(() => {
  // setup (runs side-effect)
  return () => {
    // cleanup (clean up side-effect)
  };
}, [deps]);
```

- **Εκτελείται:**
  - **Πριν ξαναεκτελεστεί το effect:** το cleanup τρέχει όταν αλλάξει κάποιο από τα dependencies.
  - **Όταν γίνει unmount το component:** το cleanup τρέχει μόλις το component βγει από το DOM

useEffect - Cleanup (3)


```
useEffect(() : () => void => {
  const id : number = setInterval(() : void => console.log( ...data: "tick"), timeout: 1000);
  return () : void => clearInterval(id);
}, []);
```

- **useEffect(() => { … }, []);**
  Το κενό dependency array [] σημαίνει ότι το effect θα εκτελεστεί **μόνο μία φορά** – στο αρχικό mount του component.
- **const id = setInterval(() => console.log("tick"), 1000);**
  Ξεκινάει ένα setInterval που γράφει “tick” στη κονσόλα κάθε 1000 ms (1s). Κρατά το numeric ID του interval για μελλοντική χρήση.
- **return () => clearInterval(id);**
  Επιστρεφει μια cleanup συναρτηση όπου, όταν το component γίνει unmount) *ή* όταν, θεωρητικά, θα ξανατρέξει το effect (δεν θα συμβεί εδώ γιατί έχουμε []), εκτελείται το clearInterval(id) και τερματίζει τον timer.

useEffect – Key Points (1)

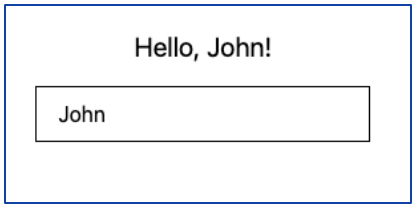
- Η function που επιστρέφουμε από το useEffect είναι **cleanup function** και εκτελείται όταν το component απο-φορτώνεται ή πριν ξανατρέξει το effect.
- Το useEffect είναι το αντίστοιχο των lifecycle methods componentDidMount, componentDidUpdate και componentWillUnmount από τις class components.
- **Προσοχή στα dependencies.** Αν δεν τα δηλωθούν σωστά, το effect μπορεί να τρέχει συνέχεια ή καθόλου.
- Αν το **dependency array** είναι κενό ([]), το effect τρέχει μόνο στο αρχικό render (σαν το componentDidMount)
- Τα dependencies μπορούν να είναι **props ή state** που θέλουμε να παρακολουθούμε.

useEffect – Key Points (2)

- Μπορούμε να έχουμε **περισσότερα από ένα useEffect** στο ίδιο component για να χωρίζουμε διαφορετικές λειτουργίες.
- Αν αλλάζουμε state μέσα στο effect, βεβαιωνόμαστε ότι το dependency array δεν περιέχει το ίδιο state που αλλάζουμε μέσα στο effect (εκτός αν αυτό είναι πραγματικά αυτό που θέλουμε).
- Όταν αλλάζουμε το state μέσα σε ένα useEffect, το component **κάνει re-render**.
- Αν το effect τρέχει κάθε φορά που γίνεται render ή κάθε φορά που αλλάζει το state που ενημερώνεται μέσα στο ίδιο το effect, μπορεί να δημιουργηθεί ένας “ατέρμονος κύκλος” (infinite loop) από συνεχόμενα re-renders.

Παράδειγμα NameChanger(1)

- Φτιάχνουμε ένα component όπου ο χρήστης πληκτρολογεί το όνομά του σε ένα input.
- Το όνομα αποθηκεύεται στο state και εμφανίζεται δυναμικά στην οθόνη και **ενημερώνει τον τίτλο της σελίδας** (document.title).
- Αν δεν έχει δοθεί όνομα, εμφανίζεται το "Hello, stranger!".



Παράδειγμα NameChanger(2)


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

const NameChangerWithEffect : () => Element = () : Element => { Show usages  new *
  const [name, setName] = useState( initialState: "");

  useEffect(() : void => {
    document.title = name ? `Hello, ${name}!` : "Hello, stranger!";
  }, [name]);

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

  return (
    <>
      <h1 className="text-center text-xl">Hello, {name || "stranger"}!</h1>
      <div className="text-center mt-4">
        <input
          type="text"
          value={name}
          onChange={handleChange}
          placeholder="Enter your name"
          className="border px-4 py-2"
        />
      </div>
    </>
  );
};

export default NameChangerWithEffect; Show usages  new *
```

Παράδειγμα NameChanger(3)

- Εισάγουμε τα Hooks **useState** και **useEffect** από την React.
- Ορίζουμε το arrow functional component.
- Δημιουργούμε μια state μεταβλητή **name** με αρχική τιμή κενό string και συνάρτηση **setName** για να την αλλάζουμε.
- Χρησιμοποιούμε το **useEffect** για να τρέξει μια ενέργεια κάθε φορά που αλλάζει το name.
- Ενημερώνουμε τον τίτλο της σελίδας: αν υπάρχει όνομα, το βάζουμε στον τίτλο· αλλιώς γράφει "Hello, stranger!".
- Ορίζουμε την συνάρτηση **handleChange** που διαβάζει το value από το input και ενημερώνει το state name κάθε φορά που αλλάζει το input.

Παράδειγμα OnlineStatus (1)


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

const  OnlineStatus : () => Element  = () : Element  => {  Show usages   new *
  const [isOnline, setIsOnline] = useState(navigator.onLine);

  useEffect(() : () => void  => {
    const handler : () => void  = () : void  => setIsOnline(navigator.onLine);
    const pollingId : number  = setInterval(handler,  timeout: 5000);

    window.addEventListener( type: "online",  handler);
    window.addEventListener( type: "offline", handler);

    return () : void  => {
      clearInterval(pollingId);
      window.removeEventListener( type: "online",  handler);
      window.removeEventListener( type: "offline", handler);
    };
  }, []);

  return (
    <div
      className={`p-4 rounded text-white text-center ${
        isOnline ? "bg-green-600" : "bg-cf-dark-red"
      }`}
    >
      {isOnline ? "You are online" : "You are offline"}
    </div>
  );
}

export default OnlineStatus;  Show usages   new *
```

- **useState(navigator.onLine)**
  Αρχικοποιεί το state isOnline με την τρέχουσα τιμή του navigator.onLine. Το navigator.onLine είναι true αν ο browser «βλέπει» δίκτυο.
- Η ιδιότητα **onLine** του αντικειμένου **Navigator** επιστρέφει αν η συσκευή είναι συνδεδεμένη στο δίκτυο. Η τιμή της αλλάζει αφού ο browser ελέγξει τη σύνδεσή του, συνήθως όταν ο χρήστης ακολουθεί κάποιον σύνδεσμο ή όταν ένα script ζητά μια απομακρυσμένη σελίδα.
- **handler**
  Μία συνάρτηση που διαβάζει ξανά το navigator.onLinen και ενημερώνει το state καλώντας setIsOnline.

Παράδειγμα OnlineStatus (2)


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

const  OnlineStatus : () => Element  = () : Element  => {  Show usages   new *
  const [isOnline, setIsOnline] = useState(navigator.onLine);

  useEffect(() : () => void  => {
    const handler : () => void  = () : void  => setIsOnline(navigator.onLine);
    const pollingId : number  = setInterval(handler,  timeout: 5000);

    window.addEventListener( type: "online",  handler);
    window.addEventListener( type: "offline", handler);

    return () : void  => {
      clearInterval(pollingId);
      window.removeEventListener( type: "online",  handler);
      window.removeEventListener( type: "offline", handler);
    };
  }, []);

  return (
    <div
      className={`p-4 rounded text-white text-center ${
        isOnline ? "bg-green-600" : "bg-cf-dark-red"
      }`}
    >
      {isOnline ? "You are online" : "You are offline"}
    </div>
  );
}

export default OnlineStatus;  Show usages   new *
```

- Όταν ο browser στείλει το online ή offline, τρέχει το handler μία φορά αμέσως, χωρίς να περιμένουμε το επόμενο polling.
- **return () => {};**
  Η Cleanup function σταματάει το timer polling όταν το component γίνει unmount μέσω της clearInterval() και αφαιρεί τα event listener χρησιμοποιώντας την μέθοδο removeEventListener().
- **[]**
  Το effect τρέχει **μόνο μία φορά** στο mount. Δεν χρειάζεται να ξανατρέξει, γιατί οι listeners και το polling ελέγχουν όλες τις μελλοντικές αλλαγές.
- **Polling**
  Κάποιοι browsersδεν στέλνουν πάντα τα events online/offline. Οπότε, εκτελούμε το handler χειροκίνητα κάθε 5s ώστε να εντοπίσουμε αλλαγές.