Signali v programskem jeziku JavaScript

Gregor Jošt, Viktor Taneski

Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko,
Maribor, Slovenija
gregor.jost@um.si, viktor.taneski@um.si

V sodobnem spletnem razvoju je asinhrona komunikacija ključna za zagotavljanje odzivne uporabniške izkušnje. Tradicionalne metode, kot so povratne funkcije in obljube, lahko postanejo kompleksne. Medtem ko sintaksa async/await poenostavi asinhrono programiranje, še vedno ne rešuje vseh težav. Signali predstavljajo sodobno rešitev za te izzive, saj omogočajo modularno in prilagodljivo obvladovanje dogodkov. Signali omogočajo preprosto registracijo in sprožanje dogodkov ter boljšo modularnost v aplikacijah. Prispevek se osredotoča na preučitev signalov v JavaScriptu, njihovih osnovnih konceptov, primerov uporabe in prednosti, kot so zmanjšanje odvisnosti med komponentami, izboljšana asinhrona komunikacija in poenostavljeno upravljanje stanja. Poudarjena je tudi integracija signalov v priljubljena ogrodja JavaScript, kot so React.js, Vue.js in Angular.

Ključne besede:
signali
JavaScript
reaktivnost
obljube
asinhrona komunikacija

Univerzitetna založba Univerze v Mariboru

ISBN 978-961-286-894-9
DOI https://doi.org/10.18690/um.feri.4.2024.18

1 Uvod

1.1. Ozadje

V sodobnem spletnem razvoju je asinhrona komunikacija ključnega pomena za zagotavljanje tekoče in odzivne uporabniške izkušnje. Tradicionalne metode za obvladovanje asinhronih operacij, kot so povratne funkcije (angl. callback functions) in obljube (angl. promises), lahko hitro postanejo kompleksne in težavne za vzdrževanje, zlasti pri večjih projektih.

Povratne funkcije pogosto vodijo v tako imenovani pekel povratnih klicev (angl. callback hell), kjer je koda globoko vgnezdena in posledično težko berljiva [1]. Obljube so bile uvedene za rešitev tega problema in so omogočile bolj linearen način obvladovanja asinhronih operacij. Z njimi je mogoče verižno povezovati asinhrone operacije, kar poenostavi obvladovanje napak in izboljša berljivost kode. Vendar pa tudi obljube niso popolna rešitev, saj kompleksne asinhrone tokove podatkov še vedno težko upravljamo in sledimo njihovemu stanju [1].

Z uvedbo sintakse »async/await« je asinhrono programiranje postalo še bolj intuitivno in podobno sinhronemu programiranju. Kljub temu pa async/await ni univerzalna rešitev za vse izzive, saj ne ponuja preprostega načina za obvladovanje več asinhronih dogodkov, ki se zgodijo v različnih delih aplikacije. Tukaj pridejo v poštev signali, ki omogočajo bolj modularno in prilagodljivo obvladovanje dogodkov.

Signali so vzorec (in implementacija), ki omogoča preprosto registracijo in sprožanje dogodkov, kar omogoča bolj modularno in vzdržljivo kodo. Ta pristop je še posebej koristen v večjih aplikacijah, kjer je treba upravljati komunikacijo med različnimi moduli ali komponentami. Pogosto se primerjajo z vzorcem »opazovalec« (angl. Observable pattern), vendar dejstvo je, da se signali razlikujejo od vzorca in ponujajo preprostost in takojšno reaktivnost [2].

V zadnjih letih so signali pridobili na priljubljenosti saj poenostavljajo in izboljšujejo asinhrono komunikacijo med komponentami. Poleg tega signali omogočajo boljšo modularnost, saj komponente med seboj komunicirajo prek signalov, ne da bi morale neposredno vedeti ena za drugo. To vodi do bolj pregledno programsko kodo, ki jo je lažje testirati in razširjati.

1.2. Cilji prispevka

Cilj prispevka je raziskati uporabo signalov v JavaScriptu in prikazati, kako lahko ta vzorec izboljša asinhrono komunikacijo v spletnih aplikacijah. Prispevek se osredotoča na naslednje cilje.

Pregled trenutnih metod za asinhrono komunikacijo v JavaScriptu, vključno s povratnimi funkcijami, obljubami in sintakso »async/await«. Ta pregled bo vključeval zgodovinski razvoj teh metod, njihove prednosti in slabosti ter primere uporabe v različnih scenarijih.

Predstavitev osnovnih konceptov signalov in njihove uporabe v JavaScriptu. Pojasnili bomo, kaj so signali, kako delujejo in zakaj so koristni pri asinhronem programiranju. Poleg tega bomo podali osnovne primere implementacije signalov.

Analiza prednosti uporabe signalov, vključno z izboljšano modularnostjo in poenostavljenim upravljanjem stanja. Poudarili bomo, kako signali omogočajo bolj jasno programsko kodo in kako olajšajo upravljanje kompleksnih asinhronih tokov podatkov.

Namen prispevka je predstaviti, kako lahko signali poenostavijo asinhrono komunikacijo in izboljšajo strukturo in vzdrževanje kode v spletnih aplikacijah. Raziskali bomo primere iz prakse in prikazali, kako signali omogočajo učinkovito obravnavo dogodkov in izboljšajo uporabniško izkušnjo. Cilj je tudi spodbuditi nadaljnje raziskave na tem področju ter predstaviti nove možnosti uporabe signalov v prihodnosti.

1.3. Struktura prispevka

Prispevek je organiziran na naslednji način. V drugem poglavju bomo predstavili osnove signalov v JavaScriptu, vključno z njihovimi definicijami, osnovnimi implementacijami ter integracijami z različnimi ogrodji. Tretje poglavje bo namenjeno analizi prednosti uporabe signalov, četrto pa bo prikazalo praktični primer uporabe signalov s pomočjo knjižnice @preact/signals-core. V zadnjem, petem poglavju bomo predstavili povzetke, ugotovitve in predloge za prihodnje raziskave.

2 Osnove signalov v JavaScriptu

2.1. Kaj so Signali?

Signali so pomemben koncept reaktivnosti v jeziku JavaScript. Lahko vsebujejo vrednost in omogočajo, da se določene komponente ali deli kode odzovejo, ko se ta vrednost spremeni. Prav tako omogočajo bolj nadzorovano uporabljanje stanja in natančno reaktivnost, kjer se komponente posodabljajo selektivno glede specifične spremembe stanja, namesto da bi se posodabljale v celoti [3].

2.2. Osnovni koncepti Signalov

- *Registracija poslušalcev:* Poslušalci (ali obravnavalci dogodkov) so funkcije, ki se registrirajo pri signalu, da prejmejo vrednosti, ko se signal sproži. To je mogoče storiti z dodajanjem funkcije na seznam poslušalcev signala. Ta registracija omogoča, da so različni deli aplikacije obveščeni o določenem dogodku, ne da bi morali neposredno komunicirati z drugimi komponentami. Na primer, več komponent se lahko registrira za prejemanje obvestil o spremembah stanja uporabnika ali o novih sporočilih v aplikaciji za klepet.
- *Sprožanje signalov:* Ko pride do določenega dogodka (t.j. spremembe vrednosti signala), se vsi registrirani poslušalci obvestijo in izvedejo z določenimi podatki. To omogoča asinhrono komunikacijo med različnimi deli aplikacije. Sprožanje signalov je ključnega pomena za odzivnost aplikacije, saj omogoča takojšnjo obravnavo dogodkov, kot so uporabniške interakcije, spremembe podatkov ali omrežne aktivnosti. Na primer, ko uporabnik pošlje sporočilo v klepetu, se signal sproži, kar obvesti vse registrirane poslušalce o novem sporočilu.
- *Odstranjevanje poslušalcev:* Poslušalce je mogoče tudi odstraniti s seznama, kar pomeni, da ne bodo več prejemali obvestil, ko se signal sproži. To je uporabno v primerih, ko komponenta ne potrebuje več obvestil o določenih dogodkih ali ko se komponenta uniči, da se preprečijo morebitni uhajanja pomnilnika (angl. memory leaks). Na primer, ko uporabnik zapre okno za klepet, se lahko poslušalec odstrani, da ne prejema več obvestil o novih sporočilih.

Registracija in odstranjevanje poslušalcev omogočata fleksibilno in dinamično upravljanje dogodkov, saj se lahko komponente enostavno prijavijo ali odjavijo glede na svoje potrebe. To zmanjšuje povezanost med komponentami in omogoča bolj modularno in vzdržljivo zasnovo aplikacij.

2.3. Primer osnovnega Signala

V svoji osnovni obliki signali delujejo kot posredniki med različnimi deli aplikacije. To izboljšuje modularnost in ponovno uporabo kode, saj komponente komunicirajo preko signalov namesto neposrednih klicev funkcij.
Signali torej služijo kot posredniki med različnimi deli aplikacije. To izboljšuje modularnost in omogoča ponovno uporabo kode, saj komponente komunicirajo preko signalov namesto preko neposrednih klicev funkcij.

Spodnji izsek kode prikazuje primer implementacije osnovnega signala:

```
const [2] = require('signals');
let count = signal(0); // začetna vrednost signala
count.observe(value => console.log(value)); // obdelovalec dogodkov, ki reagira na spremembe signala
count(10); // nova vrednost, dodeljena signalu, kar sproži opazovalca
```

V zgornjem primeru je prikazana uporaba signala za reaktivno spremljanje sprememb vrednosti. Najprej se ustvari signal `count` z začetno vrednostjo 0. Nato se registrira opazovalec z metodo `observe`, ki se sproži vsakič, ko se spremeni vrednost signala. Opazovalec prejme novo vrednost signala in izvede funkcijo, ki jo podamo, v tem primeru izpiše novo vrednost v konzolo. Na koncu se spremeni vrednost signala `count` na 10, kar sproži opazovalca in povzroči, da se nova vrednost izpiše v konzolo.

To deluje tako, da signal ob spremembi vrednosti obvesti vse registrirane opazovalce (poslušalce), kar omogoča reaktivno posodabljanje stanja in izvajanje povezanih funkcij.

2.4. Signali kot del jezika JavaScript

V času pisanja tega prispevka je predlog za vključitev signalov v standard JavaScript v 1. fazi (Stage 1) v postopku TC39 [4], kar je obetaven znak, da bi lahko v prihodnosti postali del standarda. 1. faza pomeni, da je bil koncept uradno predstavljen in da odbor prepoznava potrebo po rešitvi, vendar je predlog še vedno v zgodnji fazi razvoja. Če bodo signali napredovali v poznejše faze in postali del standarda, bi to lahko prineslo poenoteno in standardizirano upravljanje stanja in reaktivnosti v JavaScriptu. To bi zmanjšalo potrebo po različnih knjižnicah in ogrodjih z različnimi implementacijami ter olajšalo delo razvijalcem pri razvoju sodobnih aplikacij. Vključitev signalov v standard bi lahko pripeljala do bolj konsistentnega razvoja in izboljšanja učinkovitosti, kar bi imelo dolgoročne pozitivne učinke na celotno razvojno skupnost.

Predlog signala je skupno delo, ki ga vodita Rob Eisenberg in Daniel Ehrenberg [4]. Vključuje prispevke pri oblikovanju avtorjev in vzdrževalcev glavnih orodij JavaScript, kot so Angular, React in Vue. Predlog si prizadeva vzpostaviti skupno osnovo za signale, ki jo lahko sprejmejo različna JavaScript ogrodja, kar spodbuja interoperabilnost in zmanjšuje razdrobljenost ekosistema. S poudarkom na osnovni semantiki signalnega grafa, želi predlog zagotoviti osnovo, na kateri lahko ogrodja gradijo, namesto da bi določal API, usmerjen na razvijalce.

2.5. Vzorec Opazovalec vs. Signali: v čem je razlika?

Vzorec Opazovalec se bistveno razlikuje od vrednosti oz. signala. Opazovalec je podoben cevi, ki lahko kadar koli dostavi novo vrednost. To prinaša več pomembnih posledic. Prvič, v cev ne moremo preprosto pogledati in prebrati »trenutno« vrednost. Namesto tega je treba registrirati povratni klic in počakati, da se povratni klic izvede, ko se pojavi nova vrednost. Drugič, opazovalci dostavljajo vrednosti čez čas, kar pomeni, da je koncept »časa« osrednji del opazovalcev. Signali, nasprotno, nimajo koncepta časa, saj vedno lahko pridobimo »trenutno« vrednost signala. Ta razlika – dostop do »trenutne« vrednosti v primerjavi s cevjo, ki dostavlja vrednosti čez čas – je ključna razlika med signali in opazovalci. Signali so torej kot posoda, ki vedno vsebuje najnovejšo vrednost. Kadar koli jo pogledamo, vidimo trenutno stanje. Opazovalci pa so kot cev, po kateri tečejo vrednosti čez čas. Potrebujemo način (povratni klic), da »poberemo« nove vrednosti, ko pridejo.

To pomeni, da lahko pri signalih vedno preprosto dostopamo do trenutne vrednosti, medtem ko moramo pri opazovalcih nastaviti mehanizem za prejemanje vrednosti, ko se te pojavijo.

2.6. Integracija Signalov v različnih ogrodij

2.6.1. Integracija z React.js

Trenutno je za upravljanje stanja med komponentami v Reactu potrebno to stanje prenašati prek lastnosti (angl. props), ustvariti kontekst ali uporabiti nekaj, kot je Redux, ki pa ima svoje lastne težave in veliko predpripravljene kode. Ogrodje Preact, ki velja za lahko in hitro različico Reacta, kaže pot z zgledom tako, da najprej uvaja signale.

V ogrodjih, kot je Preact, so signali implementirani kot objekti z lastnostjo *.value*, ki hrani stanje. Komponente lahko berejo in spreminjajo to vrednost, pri čemer Preact učinkovito upravlja ponovne upodobe s posodabljanjem samo tistih komponent, ki so neposredno naročene na signal [6]. Tukaj je osnovni primer v Preactu:

```
import { signal, computed } from "@preact/signals";

const count = signal(0);

// Preberi vrednost signala
console.log(count.value); // 0

// Posodobi vrednost signala
count.value += 1;
console.log(count.value); // 1

// Določi komponento, ki reagira na spremembe signala
function Counter() {
 const value = count.value;

 const increment = () => {
  count.value++;
 }

 return (
  <div>
   <p>Število: {value}</p>
   <button onClick={increment}>klikni me</button>
  </div>
 );
}
```

Podrobnejši primer integracije signalov v ekosistem Preact bomo predstavili v poglavju 4.

V Reactu signali nudijo robusten mehanizem za komunikacijo med komponentami, ki presega tradicionalno upravljanje stanj. Ta pristop, ki temelji na dogodkih, omogoča komponentam, da oddajajo in poslušajo signale, kar spodbuja bolj ohlapno povezanost in odzivno arhitekturo. Integracija signalov v React je mogoča s pomočjo knjižnice *@peact/signals-react* in prinaša nekaj ključnih prednosti, in sicer [5]:

- Ločevanje komponent – signali omogočajo komunikacijo med komponentami brez neposrednih odvisnosti, kar pripomore k modularnemu in vzdržljivemu kodnemu načrtu.
- Arhitektura, ki temelji na dogodkih – ta arhitektura omogoča fleksibilno, razširljivo zasnovo, kjer komponente reagirajo na signale glede na specifične skrbi, kar izboljšuje odzivnost in intuitivnost uporabniškega vmesnika.

- Poenostavljeno upravljanje stanj – medtem ko je upravljanje stanj v Reactu zelo zmogljivo, signali nudijo alternativo za določene scenarije, saj omogočajo upravljanje specifičnih vidikov stanja aplikacije brez centraliziranega sistema za upravljanje stanj.

Oglejmo si primer implementacije signala za upravljanje opravil, shranjenih v lokalni shrambi. Začetna vrednost signala se pridobi iz lokalne shrambe:

```
const LOCAL_STORAGE_KEY = "todos";

export const getTodosFromLocalStorage = () => {
  const todos = localStorage.getItem(LOCAL_STORAGE_KEY);
  return todos ? JSON.parse(todos) : [];
};
export const todosSignal = signal(getTodosFromLocalStorage());
```

Ko se vrednost signala spremeni, se učinek (angl. effect) shrani nazaj v lokalno shrambo:

```
export const setTodosToLocalStorage = (todos) => {
  localStorage.setItem(LOCAL_STORAGE_KEY, JSON.stringify(todos));
};

effect(() => {
  setTodosToLocalStorage(todosSignal.value);
});
```

Poleg tega se dinamično preverjanje izvede, kadar se kateri koli signal znotraj funkcije spremeni, kot je signal opravil:

```
export const completedTodoCount = computed(
    () => todosSignal.value.filter((todo) => todo.isDone).length,
 );
```

Ta nastavitev se sproži pri dodajanju novega opravila:

```
const onAddTodo = () => {
    const todo = {
      id: uuidV4(),
      text,
      isDone: false,
    };

    todosSignal.value = [todo, ...todosSignal.value];

    setText("");
  };
```

2.6.2. Integracija z Vue.js

Vue.js, znan po svojem robustnem sistemu reaktivnosti, implementira signale na podoben način. Vue.js uporablja sistem Proxy za prestrezanje dostopa do lastnosti in sprememb, kar zagotavlja učinkovito sledenje in širjenje posodobitev [6, 7]. Tukaj je primer, kako deluje reaktivnost v Vue.js:

```
import { reactive, computed } from "vue";

const todos = reactive([
  { text: "Kupiti živila", completed: false },
  { text: "Sprehoditi psa", completed: false },
]);

const completed = computed(() => {
  return todos.filter(todo => todo.completed).length;
});

todos[0].completed = true;

console.log(completed.value); // Izhod: 1
```

V tem primeru je *todos* reaktiven seznam, *completed* pa je izračunana lastnost, ki se samodejno posodobi, ko se *todos* spremeni. Zgornja koda je zelo podobna kodi iz poglavja 4, ki predstavlja primer implementacije signalov s pomočjo knjižnice *@preact/signals-core*.

2.6.3. Integracija z Angular

V Angularju signali služijo kot ovoji (angl. wrappers) okoli vrednosti, ki obveščajo zainteresirane porabnike, kadar se vrednost spremeni. Ti signali lahko vključujejo katero koli vrsto vrednosti, od preprostih primitivnih vrednosti do zapletenih podatkovnih struktur. Za dostop do vrednosti signala je treba poklicati njegovo funkcijo za pridobivanje vrednosti, kar Angularju omogoča sledenje uporabe [8].

Pisni signali (angl. writable signals) omogočajo neposredne posodobitve njihovih vrednosti prek API-ja. Ustvarijo se z uporabo funkcije *signal* z začetno vrednostjo. Na primer:

```
const count = signal(0);
console.log('Število je: ' + count());
```

Za spremembo vrednosti pisnega signala lahko uporabimo metodo *.set()*:

```
count.set(3);
```

ali metodo *.update()* za izračun nove vrednosti iz prejšnje:

```
count.update(value => value + 1);
```

Računski signali (angl. computed signals) so samo za branje in izpeljujejo svoje vrednosti iz drugih signalov. Definirajo se z uporabo funkcije *computed*. Na primer:

```
const count: WritableSignal<number> = signal(0);
const doubleCount: Signal<number> = computed(() => count() * 2);
```

Tukaj *doubleCount* temelji na *count* in se posodobi, kadar se *count* spremeni. Izračunani signali se leno vrednotijo in shranijo v predpomnilnik, kar pomeni, da se vrednost izračuna le, ko je prvič dostopana, in ostane v predpomnilniku za kasnejša branja, dokler ni neveljavna zaradi spremembe odvisnosti. Izračunanim signalom ni mogoče neposredno dodeliti novih vrednosti in spremljajo samo signale, ki so bili prebrani med njihovo izpeljavo. Takšno dinamično sledenje odvisnosti omogoča učinkovito ponovno izračunavanje samo, ko je to potrebno.

Učinki (angl. effects) v Angularju so operacije, ki se sprožijo s spremembami ene ali več vrednosti signalov. Ustvarijo se z uporabo funkcije *effect* in vedno tečejo vsaj enkrat ter dinamično sledijo odvisnostim:

```
effect(() => {
  console.log(`Trenutno število je: ${count()}`);
});
```

Učinki se izvajajo asinhrono med postopkom zaznavanja sprememb. Običajno se uporabljajo za beleženje, sinhronizacijo podatkov z lokalno shrambo, prilagojeno obnašanje DOM-a ali prilagojeno upodabljanje (angl. rendering).

3 Prednosti uporabe Signalov

Signali predstavljajo napreden pristop k obvladovanju reaktivnega stanja v aplikacijah in ponujajo številne prednosti v primerjavi z drugimi metodami za asinhrono komunikacijo in upravljanje stanja. V tem poglavju bomo predstavili poglavitne prednosti uporabe signalov.

3.1. Zmanjševanje odvisnosti med komponentami

Ena glavnih prednosti signalov je, da zmanjšajo odvisnosti med različnimi komponentami aplikacije. Tradicionalno se v JavaScriptu komunikacija med komponentami pogosto izvaja prek neposrednih klicev funkcij ali prek deljenja skupnega stanja. To lahko vodi do tesne povezanosti, kar pomeni, da so spremembe v eni komponenti lahko nepredvidljivo vplivale na druge dele aplikacije.

Signali omogočajo obveščanje drugih komponent o dogodkih brez neposredne povezave med njimi. Ko ena komponenta sproži signal, se vse prijavljene komponente obvestijo o dogodku. To pomeni, da lahko komponente reagirajo na dogodke, ne da bi bile neposredno odvisne od drugih komponent, kar zmanjšuje kompleksnost in povečuje fleksibilnost.

Signali torej prispevajo tudi k večji modularnosti aplikacij, saj omogočajo, da so posamezne komponente bolj neodvisne. Z uporabo signalov lahko dosežemo, da vsaka komponenta deluje kot samostojna enota, ki se osredotoča na svojo nalogo in ne skrbi za to, kako bodo druge komponente vplivale nanjo.

3.2. Izboljšana asinhrona komunikacija

Signali bistveno izboljšajo asinhrono komunikacijo med različnimi deli aplikacije. Tradicionalni pristopi k obvladovanju asinhronih dogodkov pogosto vključujejo kompleksne strukture, kot so ugnezdene povratne funkcije ali dolge in nepregledne verige JavaScript obljub, kar lahko vodi do zapletene kode, ki jo je prav tako težko vzdrževati.

Signali omogočajo preprosto spremljanje sprememb in samodejno obveščanje vseh komponent, ki so odvisne od teh sprememb. Ta pristop poenostavi obvladovanje dogodkov, saj odpravlja potrebo po ročnem upravljanju asinhronih operacij in povratnih funkcij. Namesto tega signali omogočajo, da se spremembe v podatkih takoj odražajo v uporabniškem vmesniku, kar pripomore k bolj pregledni in manj zapleteni kodi.

3.3. Poenostavljeno upravljanje stanja

Signali prav tako poenostavijo upravljanje stanja v aplikacijah. Namesto da bi uporabljali različne metode za sinhronizacijo stanj, signali omogočajo centralizirano obdelavo in spremljanje sprememb. To pomeni, da lahko z uporabo signalov enostavno sledimo spremembam v podatkih in samodejno posodabljamo uporabniški vmesnik brez potrebe po ročnem upravljanju stanja.

Ko se stanje spremeni, signali samodejno sprožijo posodobitve v vseh komponentah, ki spremljajo to stanje. To poenostavi razvoj in vzdrževanje aplikacij, saj zmanjša potrebo po dodatnem kodiranju in usklajevanju različnih delov aplikacije, ki so odvisni od enakih podatkov.

4 Primeri uporabe

Zavoljo prikaza uporabe signalov v jeziku JavaScript bomo uporabili knjižnico @preact/signals-core, ki je del ekosistema Preact. Preact je preprosta knjižnica za gradnjo uporabniških vmesnikov, ki ponuja podobne funkcionalnosti kot React, vendar z manjšimi zahtevami po pomnilniku in boljšimi zmogljivostmi. Več o Preact smo povedali v predhodnem poglavju.

Knjižnica @preact/signals-core je specializiran paket za obvladovanje reaktivnega stanja v aplikacijah. Omogoča enostavno spremljanje in obvladovanje sprememb v podatkih in samodejno posodabljanje uporabniškega vmesnika, ko se ti podatki spremenijo. Knjižnica vključuje osnovne funkcionalnosti, kot so:

- signal: Za ustvarjanje reaktivnih podatkovnih točk.
- computed: Za ustvarjanje vrednosti, ki so odvisne od drugih reaktivnih podatkov.
- effect: Za izvajanje stranskih učinkov, ko se reaktivni podatki spremenijo.

Na praktičnem primeru bomo pokazali, kako te funkcionalnosti uporabljamo za obvladovanje stanja aplikacije in samodejno posodabljanje uporabniškega vmesnika brez potrebe po ročnem obvladovanju sprememb stanja. V aplikaciji bomo prikazali seznam artiklov, ki vsebujejo informacije o imenu, kategoriji in ceni. Uporabnik lahko išče po imenu artiklov in filtrira rezultate po kategoriji.

Reaktivno obvladovanje teh funkcij je tipičen primer za signale, ker omogoča samodejno posodabljanje uporabniškega vmesnika brez ročnega posredovanja. Ko uporabnik spremeni iskalno besedilo ali izbiro filtra, signali samodejno poskrbijo za posodobitev prikazanih podatkov v realnem času. To pomeni, da so spremembe v iskalnem polju in filtrih takoj vidne v tabeli. Primer implementacije je prikazan na spodnjem programskem kodu.

```
import { signal, computed, effect } from "@preact/signals-core";
import { IArtikel, podatki } from "./podatki";
import {
  dodajZapis,
  filtriranjeElement,
  iskanjeElement,
  teloTabeleElement,
} from "./dom";

// Signali
```

```
const iskanjeBesedila = signal("");
const filtriranjeVrednosti = signal("");

const filtriraniPodatki = computed(() => {
  const iskalniNiz = iskanjeBesedila.value.toLowerCase();
  const filtriranje = filtriranjeVrednosti.value;

  return podatki.filter(
    (artikel: IArtikel) =>
      artikel.ime.toLowerCase().includes(iskalniNiz) &&
      (filtriranje === "" || artikel.kategorija === filtriranje)
  );
});

// Funkcija za prikaz tabele
const prikaziTabela = () => {
  if (!teloTabeleElement) return;

  teloTabeleElement.innerHTML = "";
  filtriraniPodatki.value.forEach(dodajZapis);
};

// Učinki za obvladovanje reaktivnosti
effect(() => {
  prikaziTabela();
});

// Funkcija za obdelavo sprememb v iskalnem polju
const obdelajIskanje = (e: Event) => {
  const vhod = e.target as HTMLInputElement;
  iskanjeBesedila.value = vhod.value;
};

// Funkcija za obdelavo sprememb v filtrirnem elementu
const obdelajFiltriranje = (e: Event) => {
  const izbirnik = e.target as HTMLSelectElement;
  filtriranjeVrednosti.value = izbirnik.value;
};

// Nastavi obdelovalce dogodkov
if (iskanjeElement) {
  iskanjeElement.addEventListener("input", obdelajIskanje);
}

if (filtriranjeElement) {
  filtriranjeElement.addEventListener("change", obdelajFiltriranje);
}

// Začetni prikaz
```

```
prikaziTabela();
```

Kot je razvidno iz programske kode, v tej aplikaciji uporabljamo signale za obvladovanje in spremljanje stanja iskanja in filtriranja.

Implementacija vključuje dva signala, in sicer *iskanjiBesedila* in *filtriranjeVrednosti*, ki shranjujeta trenutno stanje iskanja in filtriranja. Signala omogočata, da se spremembe v teh vrednostih samodejno spremlja in posodobllja. Ko se te vrednosti spremenijo, se vsi, ki spremljajo te signale, samodejno obvestijo in ustrezno odzovejo na spremembe.

Računska vrednost (angl. computed value) *filtriraniPodatki* je odvisna od omenjenih signalov *iskanjiBesedila* in *filtriranjeVrednosti*. Ko se ti signali spremenijo, se funkcija *computed* znova izračuna. Ko se torej iskalno besedilo ali filtrirna vrednost spremenita, se znova izračuna in vrne posodobljen seznam podatkov. Signali omogočajo, da se filtriranje podatkov zgodi v realnem času brez potrebe po ročni obdelavi sprememb.

Funkcija *prikaziTabela* uporablja rezultate iz *filtriraniPodatki* in jih prikaže v tabeli. Se posodobi, ko se filtrirani podatki spremenijo, kar je posledica sprememb signalov. Funkcija izbriše obstoječe vrstice in ponovno naloži nove, kar zagotavlja, da tabela vedno prikazuje najnovejše rezultate. Signali omogočajo, da se ta posodobitev zgodi samodejno, brez potrebe po ročni spremembi stanja.

Funkcija *effect* zagotavlja, da se funkcija *prikaziTabela* vedno sproži, kadar se signali *iskanjiBesedila* ali *filtriranjeVrednosti* spremenijo. Tako je tabela vedno posodobljena z najnovejšimi podatki brez potrebe po dodatnem nadzoru.

Definirali smo tudi dve funkciji za obdelavo dogodkov (angl. event handlers), in sicer *obdelajIskanje* in *obdelajFiltriranje* za obdelavo sprememb v iskalnem polju in filtrirnem elementu:

- *obdelajIskanje* posodobi signal *iskanjiBesedila* z novim iskalnim besedilom, kar povzroči, da se filtrirani podatki samodejno posodobijo. Signali tukaj omogočajo, da se spremembe v uporabniškem vnosu nemudoma odražajo v podatkih.
- *obdelajFiltriranje* posodobi signal *filtriranjeVrednosti* z izbrano vrednostjo filtra, kar prav tako sproži posodobitev filtriranih podatkov. Signali omogočajo, da se spremembe v filtriranju takoj aplicirajo na podatke.

V tej kodi signali igrajo ključno vlogo pri zagotavljanju reaktivnosti aplikacije. Ko se spremeni stanje iskanja ali filtriranja, signali sprožijo ponovno izračunavanje filtriranih podatkov, kar povzroči, da se tabela avtomatsko posodobi. To omogoča preprosto in učinkovito obvladovanje stanja in posodabljanje uporabniškega vmesnika brez potrebe po kompleksnem ročnem upravljanju stanja.

5 Zaključek

V članku smo predstavili koncept signalov in raziskali njihove številne prednosti pri uporabi v aplikacijah JavaScript. Signali omogočajo učinkovito obvladovanje dogodkov in komunikacijo med komponentami, kar pripomore k boljši organizaciji kode, povečanju modularnosti ter izboljšanju reaktivnosti aplikacij. Sodobna ogrodja, kot so Svelte, Vue.js, Angular in drugi, že vključujejo podporo za signale, kar kaže na njihovo široko sprejetost v razvojni skupnosti. Ta podpora pomeni, da so signali že postali pomemben del sodobnega razvoja spletnih aplikacij in ponujajo številne izboljšave v primerjavi s tradicionalnimi pristopi.

V članku smo predstavili tudi konkretne primere uporabe signalov, kjer se jasno pokaže njihova prednost pred drugimi pristopi, kot so neposredni klici funkcij ali upravljanje stanja preko globalnih spremenljivk. Signali omogočajo bolj strukturiran in jasen način obvladovanja kompleksnih interakcij in asinhronih dogodkov, kar vodi do boljšega razumevanja in vzdrževanja programske kode.

Obetavno je tudi, da bi lahko Signali v prihodnosti postali del standarda JavaScript, kar bi še povečalo njihovo prisotnost in pomembnost v ekosistemu. Če oz. ko se to zgodi, bo njihova uporaba verjetno postala še bolj razširjena, saj bodo razvijalci lahko izkoristili prednosti signalov na standardiziran in združljiv način v različnih okoljih in ogrodjih. Tako lahko pričakujemo, da bodo signali igrali vse pomembnejšo vlogo v prihodnjem razvoju spletnih aplikacij in tehnologij.

Literatura

[1] Contributors M. Promise (JavaScript reference). Available from: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise.

[2] Miško Heveri. Signals vs. Observables, what's all the fuss about? 2023 [cited 2024 July 30]; Available from: https://www.builder.io/blog/signals-vs-observables.

[3] Gupta R. Learn JavaScript Reactivity: How to Build Signals from Scratch. 2024 [cited 2024 July 28]; Available from: https://www.freecodecamp.org/news/learn-javascript-reactivity-build-signals-from-scratch/#:~:text=Signals%20are%20an%20important%20concept,in%20a%20more%20controlled%20manner.

[4] Proposal Signals. [cited 2024 July 27]; Available from: https://github.com/tc39/proposal-signals.

[5] Marian Babić. Unlocking Communication in React: A Journey into the Power of Signals. 2023 [cited 2024 July 28]; Available from: https://medium.com/comsystoreply/unlocking-communication-in-react-a-journey-into-the-power-of-signals-f013d3a1ad7d.

[6] Fotis Adamakis React + Signals = Vue 3. 2023 [cited 2024 July 30]; Available from: https://fadamakis.com/react-signals-vue-3-463fefc51129.

[7] Vue Guide - Reactivity in Depth. [cited 2024 July 29]; Available from: https://vuejs.org/guide/extras/reactivity-in-depth.html.

[8] Angular Signals. Available from: https://angular.dev/guide/signals#writable-signals.