Razvoj spletnih aplikacij z uporabo spletnih komponent

Andrej Krajnc, Jani Pulko, Andrej Korošec, Bojan Štok

IZUM – Institut informacijskih znanosti, Maribor, Slovenija
andrej.krajnc@izum.si, jani,pulko@izum.si, andrej.korosec@izum.si, bojan.stok@izum.si

Sinopsis Razvoj informacijskih rešitev na osnovi komponent omogoča ponovno uporabo vnaprej pripravljenih gradnikov, ki so dostopni le preko vmesnikov, ni pa vidna njihova notranjost. Razvoj na osnovi komponent je že dolgo uveljavljen pri uporabi objektnih jezikov, ki so zastavljeni tako, da je razvoj komponent precej olajšan. V zadnjem času prevladujejo predvsem spletne aplikacije, kjer pa pri uporabi jezikov HTML in JavaScript razvoj na osnovi komponent ni zaživel v pravi meri. Velik korak naprej v tej smeri predstavljajo spletne komponente (angl. Web Components), ki omogočajo razvoj HTML komponent za spletne aplikacije. Notranja struktura spletnih komponent je skrita za druge, preko vmesnikov pa je omogočena interoperabilnost z drugimi HTML elementi. Implementacija spletnih komponent temelji na uporabi spletnih tehnologij, ki so dobro podprte v sodobnih brskalnikih. Spletne komponente lahko uporabljamo v obstoječih pristopih za razvoj spletnih aplikacij. Tako jih lahko kombiniramo s poljubnimi JavaScript knjižnicami in ogrodji, lahko se uporabljajo tudi v javanskih ogrodjih (npr. JSF), poseben pristop pa predstavlja ogrodje Vaadin, kjer s spletnimi komponentami delamo kot z drugimi javanskimi razredi.

Ključne besede:
splet
komponenta
HTML
DOM
Java

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-639-6
DOI https://doi.org/10.18690/um.feri.10.2022.4

1 Uvod

Že od nekdaj se pri razvoju programske opreme teži k uporabi komponent. Komponente so ponovno uporabne enote, ki jih razvijamo z namenom, da bodo večkrat uporabljene. Uporaba komponente temelji na konceptu ograjevanja (angl. encapsulation), kjer se za druge skrivajo interne lastnosti in operacije. Komponente so tako dostopne le preko vmesnikov, ni pa vidna njihova notranjost.

Razvoj na osnovi komponent je pridobil na pomenu ob uveljavitvi objektnih jezikov (npr. Java, C+, C#), ki so zastavljeni tako, da je razvoj komponent precej olajšan. Še posebej velik napredek pri uporabi komponent se je zgodil ob uporabi programskega jezika Java. Obstaja namreč veliko javanskih komponent, knjižnic razredov in ogrodij.

V zadnjem času prevladujejo predvsem spletne aplikacije, kjer se objektni jeziki uporabljajo predvsem na strežniku, na odjemalski strani (v brskalniku) pa programska koda temelji na predvsem označevalnem jeziku HTML in programskem jeziku JavaScript. Čeprav se je jezik HTML od pojava leta 1993 veliko razvijal in nadgrajeval, pa dolgo ni omogočal razvoja na osnovi komponent kot ga poznamo v objektnih jezikih.

Pri razvoju na osnovi komponent želimo uporabljati komponente, ki bodo uporabne še čez mnogo let. Žal mnogi pristopi pri razvoju spletnih aplikacij temeljijo na uporabi pristopov, ki se sčasoma precej spreminjajo in z leti komponente sčasoma niso več ponovno uporabne v takšni meri kot bi si želeli. Veliko JavaScript ogrodij ima življenjsko dobo le nekaj let, na drugi strani pa želimo, da naše aplikacije delajo veliko dlje. Zato moramo pri gradnji spletnih aplikacij uporabljati pristope, kjer bomo uporabljali komponente z daljšo življenjsko dobo. Velik korak naprej glede preglednosti JavaScript kode je bil narejen z EcmaScript 6 (ES6).

2 Spletni standardi za spletne komponente

Precejšen korak naprej pri uporabi komponent v spletnih aplikacijah predstavljajo spletne komponente (angl. Web Components). Spletne komponente omogočajo razvoj HTML komponent za spletne aplikacije. Notranja struktura spletnih komponent je skrita za druge, preko vmesnikov pa je omogočena interoperabilnost z drugimi HTML elementi. Implementacija spletnih komponent temelji na uporabi spletnih tehnologij, ki so dobro podprte v sodobnih brskalnikih.

Koncept spletnih komponent je prvi predstavil Alex Russell leta 2011 na konferenci Fronteers Conference. Spletne aplikacije temeljijo na uporabi spletnih standardov, vendar ne obstaja poseben standard samo za spletne komponente, saj spletne komponente povezujejo več drugih spletnih standardov. Veliko spletnih standardov je nastalo znotraj konzorcija za splet (W3C - World Wide Web Consortium) [1], dodatno so različne standarde in specifikacije definirali še Mozilla, Google Web Fundamentals, WHATWG itd.

Trenutno najbolj aktualni standardi so WHATWG standardi [2]. WHATWG (Web Hypertext Application Technology Working Group) je skupnost ljudi, zainteresiranih za razvoj HTML in povezanih tehnologij. Ustanovljena je bila leta 2004, člani pa so Apple, Mozilla, Google, Microsoft itd. Leta 2018 je prišlo do nestrinjanja med W3C in člani WHATWG glede prihodnosti razvoja standardov HTML, leta 2019 pa je bil sklenjen dogovor med W3C in WHATWG, da se bo večina najpomembnejših spletnih standardov razvijala znotraj WHATWG.

Za področje spletnih komponent sta najpomembnejša standarda HTML Living Standard (nadaljevanje W3C HTML standarda) in DOM Living Standard (nadaljevanje W3C DOM standarda). Znotraj HTML Living Standard sta aktualna predvsem standarda HTML custom elements (HTML elementi po meri) in HTML templates (HTML predloge). Znotraj DOM Living Standard je aktualen predvsem standard HTML shadow DOM (HTML senčni DOM).

Spletni standardi so pomembni predvsem zaradi podpore v spletnih brskalnikih. Tipično je potreben določen čas, da brskalniki podprejo nove standarde, stari brskalniki pa imajo pogosto težave z novimi standardi. Vendar pa, ko

so novosti podprte v brskalnikih, pa imajo brskalniki dobro podporo za nazaj. Za razliko od mnogih JavaScript ogrodij se programski vmesniki v brskalnikih ohranjajo skozi mnoga leta.

3 Ključne specifikacije/tehnologije za spletne komponente

Spletne komponente zaobsegajo 4 tehnologije / specifikacije.

Senčni DOM (shadow DOM) je množica JavaScript programskih vmesnikov (API) za pripenjanje ločenega DOM drevesa v glavno DOM drevo oz. element.

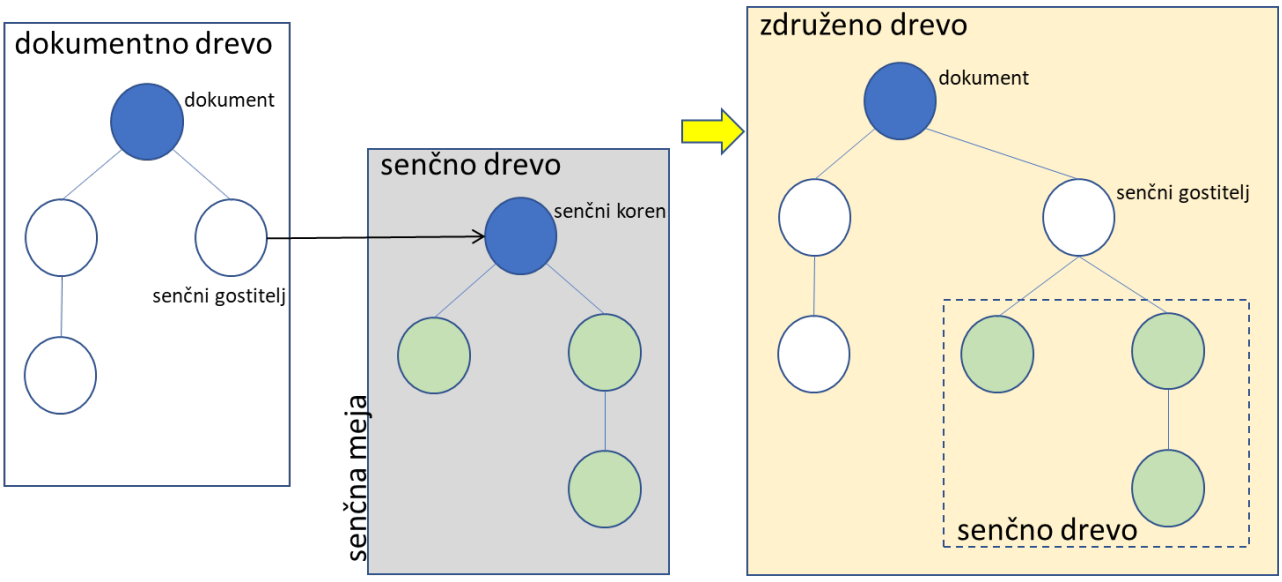
Elementi po meri (custom elements) predstavljajo množico JavaScript programskih vmesnikov (API) za definiranje novih HTML elementov.

HTML predloge (templates) prinašajo HTML elementa <template> in <slot>, ki omogočata izdelavo označevalnih predlog, ki se ne prikažejo na osnovni strani. Predloge so lahko večkrat ponovno uporabljene in predstavljajo osnovo za elemente po meri.

ECMAScript moduli (nodules) omogočajo definiranje modulov v ECMAScript. Ta trenutek moduli še niso tako zaživeli, znajo pa biti zelo pomembni v prihodnosti tudi za spletne komponente.

3.1 Senčni DOM (shadow DOM)

Programski vmesnik, ki ga ponuja senčni DOM, omogoča vpenjanje senčnega drevesa (shadow tree) v glavno drevo. Senčni koren (shadow root) iz senčnega drevesa se pripne v senčnega gostitelja (shadow host), ki je navadno DOM vozlišče (node) v glavnem drevesu. Rezultat je združeno drevo in takšno združeno drevo je upodobljeno (rendered) v brskalniku. Senčni DOM je omejen s senčno mejo (shadow boundary), ki predstavlja mesto, kjer se konča senčni DOM in začne glavni DOM.



Slika 1: Senčni DOM.
Vir: lasten.

V senčnem DOMu je drevo upodobljeno ločeno od glavnega DOM drevesa, zaradi česar so lastnosti elementa zasebne in neodvisne od drugih delov HTML dokumenta. Ta koncept ograjevanja sodi med glavne prednosti uporabe senčnega DOMa. Pri ograjevanju gre za več stvari: ograjevanje kode, ograjevanje stilov in ograjevanje obnašanja. Delo s senčnim DOMom je enako delu z navadnim DOMom, saj lahko dodajamo nova vozlišče v

DOM, spreminjamo atribute, nastavljamo stile itd. Glavna razlika je v tem, da koda, stili in obnašanje v senčnem DOMu nimajo vpliva na obnašanje izven senčnega DOMa.

Med slabosti uporabe senčnega DOMa sodi to, da ne delujejo vse knjižnice s senčnim DOMom. Včasih tudi ne želimo izolacije stilov, temveč želimo globalne stile. V senčni DOM je možno prilepiti zunanji stil, npr.

```
const linkElem = document.createElement('link');
linkElem.setAttribute('rel', 'stylesheet');
linkElem.setAttribute('href', 'style.css');
shadow.appendChild(linkElem);
```

Senčni DOM lahko uporablja na dva načina:

- **odprti način** - iz glavne strani lahko dostopamo do senčnega DOMa
  - senčni DOM dodamo z **attachShadow({mode: 'open'})**, nato lahko senčni DOM pridobimo z **let myShadowDom = myCustomElem.shadowRoot;**
- **zaprti način** - iz glavne strani ne moremo dostopati do senčnega DOMa
  - senčni DOM dodamo z **attachShadow({mode: 'closed'})**, ne moremo pa dostopati do njega, saj **myCustomElem.shadowRoot** vrne **null**

Senčni DOM ni čisto nov koncept v brskalnikih. Že dolgo so v brskalnikih nahajajo elementi, ki skrivajo svojo notranjost. Primer takšnega elementa je implementacija označbe <video>, kjer vsak brskalnik na svoj način implementira predvajalnik videoposnetkov in vsak tak predvajalnik ima svoje elemente (gumbe) za upravljanje predvajanja. V HTML kodi je vidna le označba <video>, elementi predvajalnika pa so navzven skriti.

3.2 HTML elementi po meri

Pri HTML elementih po meri gre za to, da ne uporabljamo standardnih HTML elementov, temveč si ustvarimo lastne HTML elemente po meri. Tako v HTML kodi ne uporabljamo le standardnih HTML označb, temveč tudi naše lastne označbe.

Ločimo dve vrsti HTML elementov po meri

- **avtonomni elementi po meri** so popolnoma ločeni od obstoječih HTML elementov
- **prilagojeni elementi po meri** razširjajo enega od HTML elementov

3.2.1 Avtonomni elementi po meri

Avtonomni elementi po meri so neodvisni od standardnih HTML elementov. Registriramo jih z uporabo **customElements.define**, npr.

```
customElements  .define('my-paragraph1',
  class extends HTMLElement {
    constructor() {
      super();
     // dodatna inicializacija elementa po meri
    }
  }
);
```

V HTML kodi jih uporabljamo kot HTML elemente s HTML oznako, npr.

```
<my-paragraph1></my-paragraph1>
```

ali

```
document.createElement("my-paragraph1")
```

3.2.2 Prilagojeni elementi po meri

Prilagojeni elementi po meri razširjajo enega od standardnih HTML elementov. Registriramo jih z uporabo **customElements.define**, pri čemer se navede oznako razširjenega HTML elementa, npr.

```
class MyParagraph2 extends HTMLParagraphElement {
    constructor() {
      super();
     // dodatna inicializacija elementa po meri
    }
  }
customElements.define('my-paragraph2', MyParagraph2, { extends: 'p' });
```

V HTML kodi jih uporabljamo tako, da uporabimo oznako razširjenega HTML elementa, ime elementa pa navedemo pri lastnosti "is", npr.

```
<p is="my-paragraph2">
```

ali

```
document.createElement("p", { is: "my-paragraph2" })
```

3.3 HTML predloge in reže

Pri spletnih komponentah se pogosto uporabljajo HTML predloge (angl. template) in reže (angl. slot).

HTML označba <template> predstavlja predlogo za HTML element in je na voljo že od ECMAScript 2015 (ES6) dalje. Gre za mehanizem, ki omogoča, da se del HTML kode ne uporabi za prikaz takoj po nalaganju strani (se ne aktivira). Ob nalaganju strani se le validira HTML koda v predlogi. Vse dokler se predloga ne uporabi, se skripte ne izvedejo, slike se ne naložijo, zvok se ne predvaja itd. Pri HTML predlogah gre poudariti, da dostop do elementov v predlogi ni mogoč (npr. preko getElementById ali querySelector).

Predloga se za prikaz uporabi v času izvajanja z JavaScript kodo. Osnovna ideja pri HTML predlogah je večkratna ponovna uporaba predloge. Pogosto želimo doseči ponovno uporabo posameznih delov strani in to lahko naredimo tudi z uporabo HTML predlog.

Primer definiranja in uporabe HTML predloge v spletni komponenti, ki se nahaja v datoteki my-paragraph1.js:

```
customElements.define('my-paragraph1',
  class extends HTMLElement {
    constructor() {
      super();
      let template = document.getElementById('my-paragraph-template');
      let templateContent = template.content;
      const shadowRoot = this.attachShadow({mode: 'open'})
```

```
        .appendChild(templateContent.cloneNode(true));
    }
  }
);

<!DOCTYPE html>
<html>
<head>
  <script src="my-paragraph1.js" defer></script>
</head>
<body>
  <p>Demo Web Components 1</p>
  <template id="my-paragraph-template">
    <style>
      p {
        color: white;
        background-color: rgb(117, 17, 121);
        padding: 15px;
      }
    </style>
    <p>My paragraph</p>
  </template>
  <my-paragraph1></my-paragraph1>
</body>
</html>
```

Ob označbi za predlogo se lahko uporablja HTML označba za režo <slot>. Ideja pri uporabi rež je ta, da se predloga razdeli na manjše dele (reže). Vsak del (reža) ima lahko svoj stil. Primer definiranja in uporabe HTML reže in zgornje spletne komponente.

```
<!DOCTYPE html>
<html>
  <head>
    <script src=my-paragraph1.js" defer></script>
  </head>
<body>
  <p>Demo Web Components 2</p>
  <template id="my-paragraph-template">
    <style>
      p {
        color: white;
        background-color: rgb(117, 17, 121);
        padding: 15px;
      }
```

```
    </style>
    <p><slot name="my-text">My default text</slot></p>
  </template>
  <my-paragraph1></my-paragraph1>
  <my-paragraph1>
    <span slot="my-text">Let's have some different text!</span>
  </my-paragraph1>
  <my-paragraph1>
    <ul slot="my-text">
      <li>Let's have some different text!</li>
      <li>In a list!</li>
    </ul>
  </my-paragraph1>
</body>
</html>
```

4 Podpora za spletne komponente

4.1 Podpora za spletne komponente v brskalnikih

Spletne komponente so že nekaj let dobro podprte v brskalnikih, kot so Chrome, Firefox, Microsoft Edge, Opera itd. Dobra podpora je predvsem v brskalnikih, ki temeljijo na Chromium.

Brskalnik Safari je v zadnjem času zelo izboljšal podporo za spletne komponente, vendar ne podpira spletnih komponent čisto v celoti. Brskalnik Safari ne podpira prilagojenih elementov, ki razširjajo enega od HTML elementov

V primeru brskalnikov, ki ne podpirajo spletnih komponent, je zagotovljena kompatibilnost za nazaj z uporabo kode polyfills [4].

Podporo za spletne komponente v brskalnikih se da preveriti tudi na spletni strani Can I use [4].

4.2 Podpora za spletne komponente v knjižnicah in ogrodjih

Obstaja kar nekaj knjižnic za spletne komponente, kot so FASTElement, snuggsi, X-Tag, Slim.js, Lit, Smart, Stencil, hyperHTML-Element, DataFormsJS, Polymer, Bosonic, Riot.js.

Skupnost za spletne komponente je vedno večja, kar je razvidno tudi na ponudbi spletnih komponent na WebComponents.org [5].

Vedno boljše je podprta tudi uporaba spletnih komponent v navezavi s spletnimi ogrodji, kot so Angular, React, Vue.js itd.

Obstajajo celo ogrodja, ki v celoti temeljijo na uporabi spletnih komponent. Eno takšnih ogrodij je Vaadin [6].

5 Uporaba spletnih komponent v Izumovih spletnih aplikacijah

Institut informacijskih znanosti (IZUM) razvija informacijski servis slovenske znanosti, kulture in izobraževanja. Javnosti najbolj poznan je sistem COBISS, ki predstavlja temelj knjižničnega informacijskega sistema Slovenije in nekaterih drugih držav, ki so povezani v mrežo COBISS.net.

Dosedaj so bile spletne komponente uporabljene v treh spletnih aplikacijah:

- Digitalni repozitorij COBISS (dCOBISS) [7]
- Informacijski sistem o raziskovalni dejavnosti v Sloveniji (SICRIS) [8]
- v razvoju je COBISS4 (nova generacija programske opreme za knjižničarje, nadgrajuje COBISS3) [9]

Za razvoj dCOBISS in SICRIS se uporablja programski jezik Java. Ključna ogrodja v dCOBISS in SICRIS so JavaServer Faces (JSF), Krazo (javansko ogrodje za MVC Model-View-Controller) in Bootstrap (CSS ogrodje za razvoj odzivnih spletnih aplikacij). Čeprav gre pri dCOBISS in SICRIS za spletne aplikacije, je klasičnega JavaScripta-a relativno malo. Velik poudarek pa se daje na spletne komponente, na katerih temelji velik del aplikacij.

Pri razvoju COBISS4 se spletne komponente uporabljajo skozi ogrodje Vaadin.

5.1 Uporaba spletnih komponent v dCOBISS

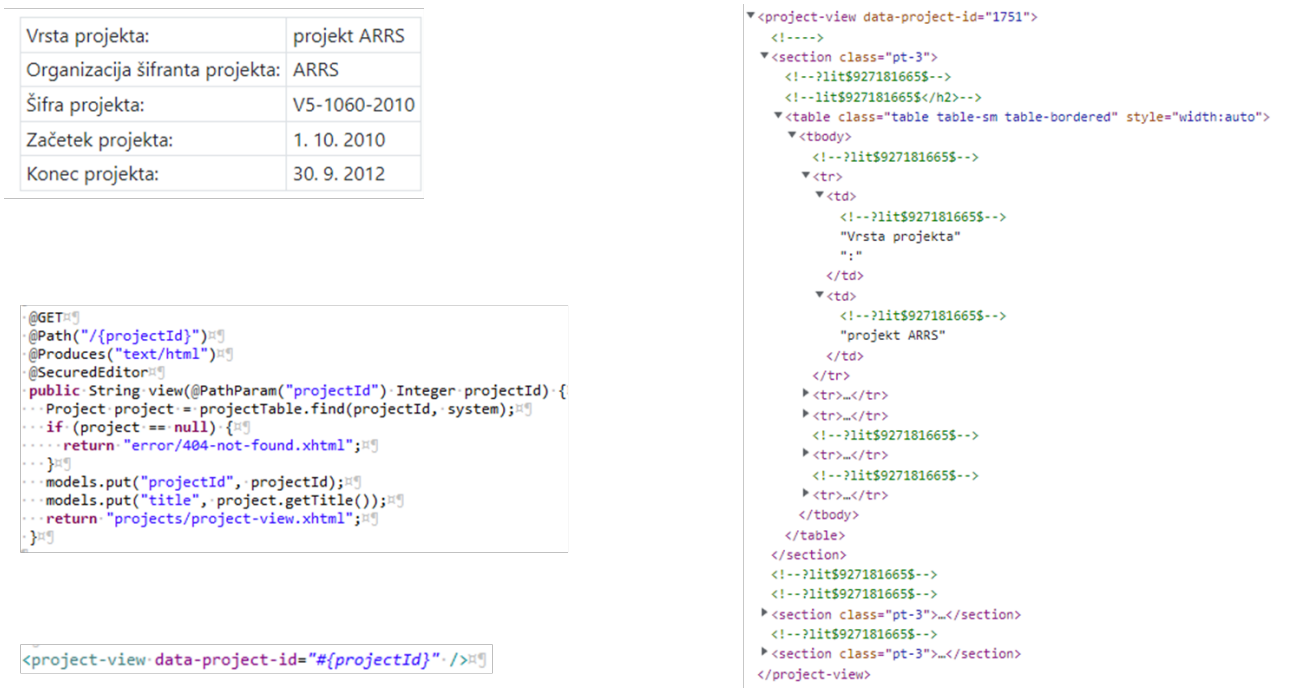
Spletne komponente v dCOBISS se uporabljajo za večino uporabniškega vmesnika dCOBISS (prikaz, urejanje itd.).

Slika 2: Digitalni repozitorij COBISS (dCOBISS).
Vir: [7].

dCOBISS je aplikacija za knjižničarje, ne za končne uporabnike. Zato izgled ni tako zelo pomemben kot pri aplikacijah za končne uporabnike.

HTML struktura je definirana večinoma v sami spletni komponenti, na strežniški strani (Controller, JSF XHTML) se zgolj uporabi ta spletna komponenta.

Primer uporabe spletnih komponent pri prikazu podatkov o projektu v dCOBISS je prikazan na spodnjih slikah.



Vrsta projekta:	projekt ARRS
Organizacija šifranta projekta:	ARRS
Šifra projekta:	V5-1060-2010
Začetek projekta:	1. 10. 2010
Konec projekta:	30. 9. 2012

```
@GET
@Path("/{projectId}")
@Produces("text/html")
@SecuredEditor
public String view(@PathParam("projectId") Integer projectId) {
    Project project = projectTable.find(projectId, system);
    if (project == null) {
        return "error/404-not-found.xhtml";
    }
    models.put("projectId", projectId);
    models.put("title", project.getTitle());
    return "projects/project-view.xhtml";
}
```

```
<project-view data-project-id="#{projectId}" />
```

```
<project-view data-project-id="1751">
  <!---->
  <section class="pt-3">
    <!--?lit$927181665$-->
    <!--lit$927181665$</h2>-->
    <table class="table table-sm table-bordered" style="width:auto">
      <tbody>
        <!--?lit$927181665$-->
        <tr>
          <td>
            <!--?lit$927181665$-->
            "Vrsta projekta"
            ":"
          </td>
          <td>
            <!--?lit$927181665$-->
            "projekt ARRS"
          </td>
        </tr>
        <tr>…</tr>
        <tr>…</tr>
        <!--?lit$927181665$-->
        <tr>…</tr>
        <!--?lit$927181665$-->
        <tr>…</tr>
      </tbody>
    </table>
  </section>
  <!--?lit$927181665$-->
  <!--?lit$927181665$-->
  <section class="pt-3">…</section>
  <!--?lit$927181665$-->
  <section class="pt-3">…</section>
</project-view>
```

Slika 3: Prikaz podatkov o projektu v dCOBISS.
Vir: [7].

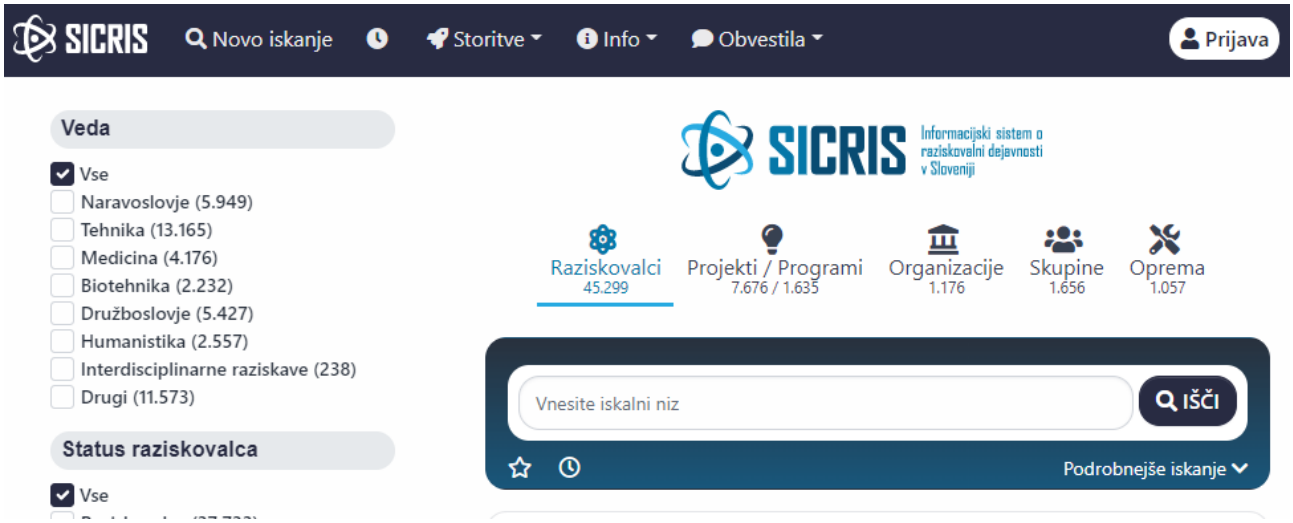
```
class ProjectView extends HTMLElement {
    async connectedCallback() {
        let projectId = parseInt(this.dataset.projectId);
        if (!projectId) {
            logger.warn("project-view.js data-project-id attribute not set");
            return;
        }
        this.project = await getProject(projectId);
        if (!this.project) {
            logger.warn(`project-view.js project ${projectId} not found`);
            return;
        }

        this.render();

customElements.define("project-view", ProjectView);
```

```
render() {
    render(
        html`
            <section class="pt-3">${this.baseDataHtml}</section>
        `,
        this
    );
}
get baseDataHtml() {
    return html`
        <!--${translate("erepo.project.base-data.title")}</h2>-->
        <table class="table table-sm table-bordered" style="width:auto">
            <tbody>
                <tr>
                    <td>${translate("erepo.project.projectType")}:</td>
                    <td>
                        ${translate("erepo.project.type." + this.project.projectType)
                    </td>
                </tr>
```

Slika 4: Spletna komponenta za prikaz podatkov o projektu v dCOBISS.
Vir: [7].

5.2 Uporaba spletnih komponent v SICRIS

Spletne komponente v SICRIS se uporabljajo za večino uporabniškega vmesnika SICRIS. Tako so bile izdelane komponente za prikaz posameznih rezultatov, komponente za prikaz iskalnih rezultatov, vizualizacijo podatkov in filtre, komponente za samodokončanje (autocomplete) in komponente za urejanje podatkov.



Slika 5: Informacijski sistem o raziskovalni dejavnosti v Sloveniji (SICRIS).
Vir: [8].

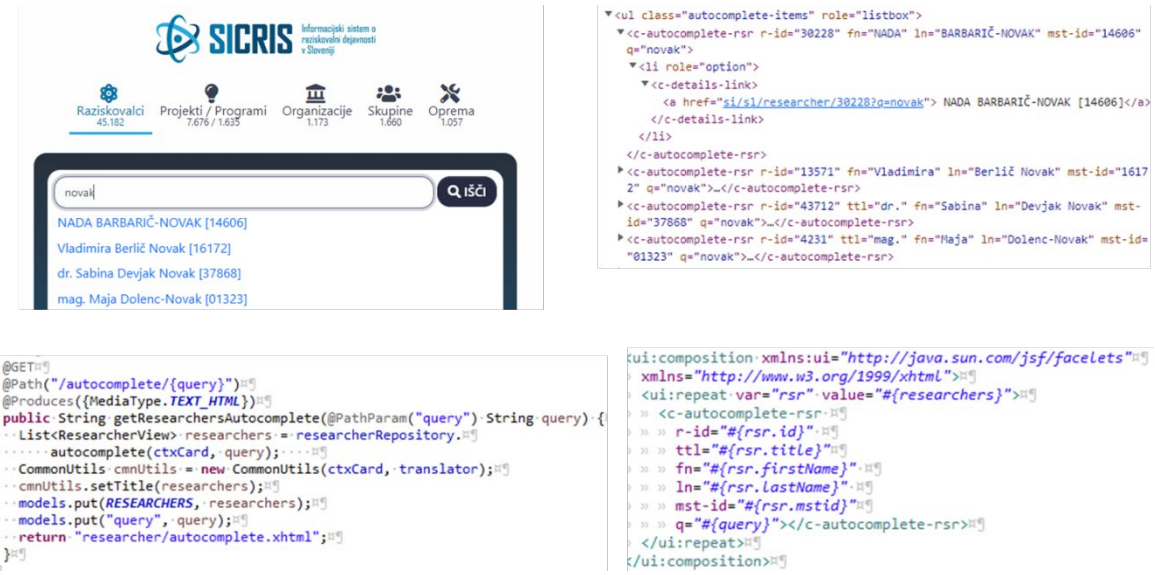
SICRIS je aplikacija za končne uporabnike, zato je izgled zelo pomemben. Pomembno je, da ima oblikovalec aplikacij čimbolj olajšan način oblikovanja aplikacij.

HTML struktura je definirana večinoma na strežniški strani v JSF XHTML. Na odjemalski strani v spletni komponenti ni definirana HTML struktura in tak pristop prijaznejši za oblikovalce uporabniškega vmesnika, ki so navajeni na XHTML datoteke

SICRIS je »single page« aplikacija, ki vsebine pridobiva na asinhroni način preko spletnih komponent. Komponente med seboj komunicirajo preko dogodkov.

Pogosto so onemogočeni privzeti načini pošiljanja podatkov iz obrazcev. V takšnih primerih se proži globalni dogodek, na katerega se odzovejo spletne komponente, ki so poslušalci tega dogodka. Ti tipično pošljejo asinhrone zahteve na strežnik in na ta način se napolnijo posamezni fragmenti strani (rezultati iskanja, filtri, navigacijska vrstica, zgodovina iskanja itd.). Kot fragment se iz strežnika pošljejo kar HTML fragmenti in ne zgolj podatki npr. v JSON obliki.

Primer uporabe spletnih komponent pri prikazu rezultatov samodokončanja (autocomplete) raziskovalcev v SICRIS je prikazan na spodnjih slikah.



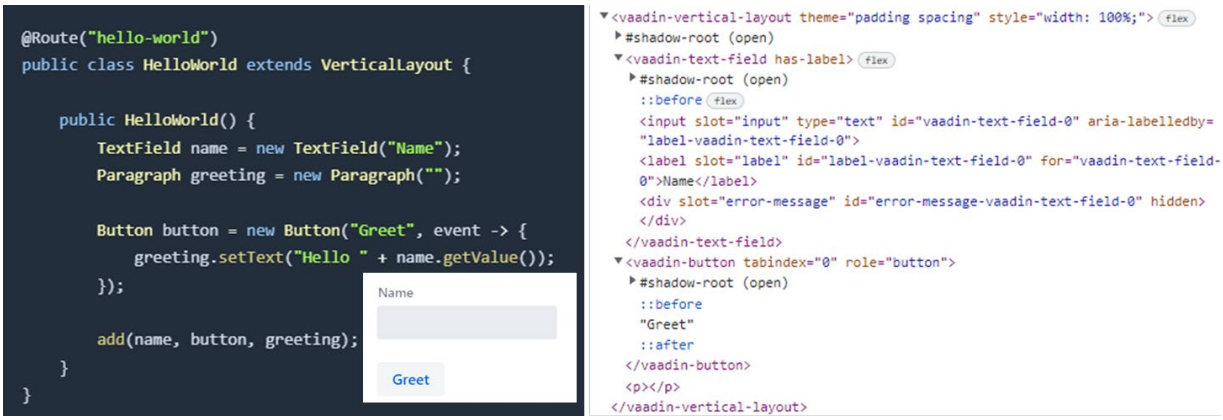
Slika 6: Prikaz rezultatov iskanja raziskovalcev v SICRIS.
Vir:[8].

```
class CAutocompleteRsr extends HTMLElement {
  constructor() {
    super();
  }

  connectedCallback() {
    const id = this.getAttribute("r-id");
    const title = this.getAttribute("ttl") || "";
    const firstName = this.getAttribute("fn");
    const lastName = this.getAttribute("ln");
    const mstid = this.getAttribute("mst-id");
    const q = this.getAttribute("q");
    if (!id || !q) {
      return;
    }
    const li = document.createElement("LI");
    li.setAttribute("role", "option");
    const detailsLink = document.createElement("c-details-link");
    const link = document.createElement("link");
    link.href = `${window.appData.url}/researcher/${id}?q=${q}`;
    link.textContent = `${title} ${boldStart(firstName, q)} ${boldStart(
      lastName,
      q
    )} [${boldStart(mstid, q)}]`;

    detailsLink.appendChild(link);
    li.appendChild(detailsLink);
    li.onclick = () => {
      li.querySelector("a").click();
    };
    this.appendChild(li);
  }
}
```

Slika 7: Spletna komponenta za prikaz rezultatov iskanja raziskovalcev v SICRIS.
Vir:[8].

5.3 Uporaba spletnih komponent v ogrodju Vaadin

Vaadin je odprtokodna platforma za razvoj spletnih aplikacij. Za razvoj spletnih aplikacij v Javi je na voljo ogrodje Vaadin Flow, ki omogoča izgradnjo spletnih aplikacij 100% v Javi. Vaadin Flow omogoča izdelavo spletnih aplikacije brez programiranja v JavaScript in HTML., saj se javanski razredi preslikajo v spletne komponente.



Slika 8: Programiranje v ogrodju Vaadin Flow.
Vir: [6].

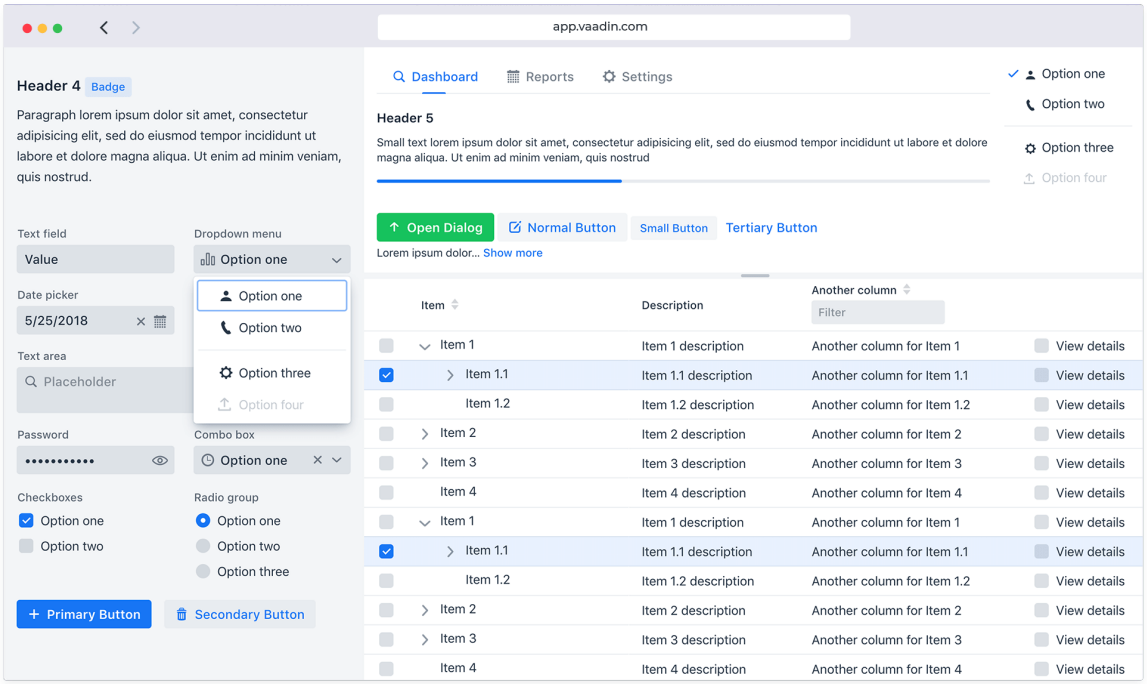
Pri programiranju v Vaadin Flow gre za drugačen pristop k izgradnji spletnih aplikacij. Razvijalci ne razmišljajo o HTTP zahtevah in odgovorih, temveč se podobno kot pri tradicionalnih namiznih aplikacijah (npr. Swing) razmišlja predvsem o izgradnji grafičnega vmesnika na osnovi komponent.

Javanske grafične komponente se v brskalniku prikazujejo v HTML kodi kot spletne komponente v standardni HTML obliki in podobno kot druge spletne komponente delujejo v vseh brskalnikih. Večina logike za grafični vmesnik se izvede na strežniku, s čimer imamo manjšo varnostno izpostavljenost. Komunikacija med odjemalcem

in strežnikom poteka avtomatsko preko ogrodja Vaadin z uporabo WebSocket ali HTTP. Pošiljajo se JSON sporočila, ki posodabljajo tako grafični vmesnik v brskalniku in stanje grafičnega vmesnika na strežniku

Vaadin ponuja veliko spletnih komponent za grafični vmesnik:

- Različna polja za obrazce: Text Field, Number Field, Text Area, Checkbox, Combo Box, Radio Button, Date Picker, Time Picker itd.
- Različni elementi za prikaz: Menu Bar, Context Menu, Button, Dialog, Notification, Progress Bar, Grid, Tabs, Scroller itd.



Slika 9: Komponente v ogrodju Vaadin Flow.
Vir: [6].

Vaadinove spletne komponente temeljijo na modernih konceptih, saj temeljijo na spletnih standardih (W3C/WHATG). V komponentah se uporablja senčni DOM, možno jih je prilagajati z različnimi temami (CSS). Vaadinove komponente je možno uporabljati v praktično vseh sodobnih ogrodjih za front-end. Ponujajo programski vmesnik za Javo in TypeScript. Prijazne so za mobilne naprave, saj so odzivne in optimizirane za različne velikosti zaslonov.

Pomembno je tudi, da podpirajo spletno dostopnost v skladu s standardom WCAG. Še posebej je to pomembno za uporabnike bralnika zaslona in tudi druge ranljive skupine, ki imajo težave pri uporabi spleta.

6 Zaključek

Spletne komponente omogočajo gradnjo ponovno uporabnih enot, do katerih se dostopa preko programskih vmesnikov, notranja implementacija pa je skrita. Na ta način so spletne komponente velik korak naprej k razvoju na osnovi komponent, kar pri HTML prej ni bilo tako prisotno.

Spletne komponente poenostavijo razvoj, omogočajo lažje vzdrževanje na dolgi rok in prispevajo k večji modularizaciji spletnih aplikacij. Temeljijo na spletnih standardih, ki imajo dobro podporo v brskalnikih. Imajo tudi daljšo življenjsko dobo kot marsikatere druge alternativne rešitve.

Spletne komponente lahko uporabljamo na različne načine. Možen je klasični pristop h gradnji aplikacij z generiranjem HTML vsebine na strežniku oz. uporabo predlog (npr. z uporabo JSF), javanski pristop (Vaadin) ali pa z generiranjem HTML na odjemalcu (JavaScript).

Naše izkušnje so pozitivne in z uporabo spletnih komponent nameravamo nadaljevati tudi v prihodnosti.

Literatura

[1] World Wide Web Consortium (W3C), https://www.w3.org/, obiskano 28. 7. 2022

[2] Web Hypertext Application Technology Working Group (WHATWG), https://whatwg.org/, obiskano 28. 7. 2022

[3] polyfills, https://github.com/webcomponents/polyfills/tree/master/packages/webcomponentsjs, obiskano 28. 7. 2022

[4] Can I use web components?, https://caniuse.com/?search=web%20components, obiskano 28. 7. 2022

[5] WebComponents.org, https://www.webcomponents.org/, obiskano 28. 7. 2022

[6] Vaadin, https://vaadin.com/, obiskano 28. 7. 2022

[7] Digitalni repozitorij COBISS (dCOBISS), https://d.cobiss.net/repository/, obiskano 28. 7. 2022

[8] Informacijski sistem o raziskovalni dejavnosti v Sloveniji (SICRIS), https://www.sicris.si/, obiskano 28. 7. 2022

[9] Programska oprema COBISS3, https://www.cobiss.si/c3/, obiskano 28. 7. 2022