Uporaba spletnih komponent v Javi

Andrej Krajnc, Bojan Štok

COBISS

IZUM

Spletne komponente (Web Components) na eni strani

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


customElements.define('my-paragraph1',
  class extends HTMLElement {
    constructor() {
      super();
      let template = document.getElementById('my-paragraph');
      let templateContent = template.content;

      const shadowRoot =  this.attachShadow({mode: 'open'})
        .appendChild( templateContent.cloneNode(true));
    }
  }
);

   <template id="my-paragraph">
        <style>
        p {
            color: white;
            background-color: rgb(117, 17, 121);
            padding: 15px;
        }
        </style>
        <p>My paragraph</p>
    </template>
```

Razvoj na osnovi komponent

- od nekdaj želja po uporabi komponent
  - ponovno uporabne enote
  - ograjevanje (skrivanje internih lastnosti in privatnih metod)
- razvoj na osnovi komponent je pridobil na pomenu ob uveljavitvi objektnih jezikov (Java, C+, C# itd.)
  - precej olajšan razvoj komponent
- velik napredek ob uporabi programskega jezika Java
  - veliko javanskih komponent, knjižnic razredov in ogrodij
- pri javanskih aplikacijah se komponente uporabljajo predvsem na strežniku
- spletne aplikacije se na odjemalski strani izvajajo v brskalniku
  - ključni tehnologiji so HTML in JavaScript
    - veliko JavaScript ogrodij in komponent
    - jezik HTML se je od pojava 1993 veliko razvijal in nadgrajeval, vendar dolgo ni omogočal razvoja na osnovi komponent kot ga poznamo v objektnih jezikih
    - želimo izdelovati komponente, ki bodo uporabne še čez leta

Spletne komponente - standardi

- Web Components - ponovno uporabne HTML komponente
  - predstavitev na Fronteers Conference 2011 (Alex Russell)
  - ne obstaja standard samo za spletne komponente
    - veliko standardov nastalo znotraj W3C - World Wide Web Consortium
    - različni standardi/specifikacije: Mozilla, Google Web Fundamentals, WHATWG
  - trenutno najbolj aktualni standardi so WHATWG standardi
- WHATWG - Web Hypertext Application Technology Working Group
  - skupnost ljudi, zainteresiranih za razvoj HTML in povezanih tehnologij
    - ustanovljena leta 2004, člani Apple, Mozilla, Google, Microsoft itd.
    - leta 2019 dogovor med W3C in WHATWG
  - WHATWG standardi (https://whatwg.org/)
    - HTML Living Standard (nadaljevanje W3C HTML standarda)
      - HTML custom elements (HTML elementi po meri)
      - HTML templates (HTML predloge)
    - DOM Living Standard (nadaljevanje W3C DOM standarda)
      - HTML shadow DOM (HTML senčni DOM)

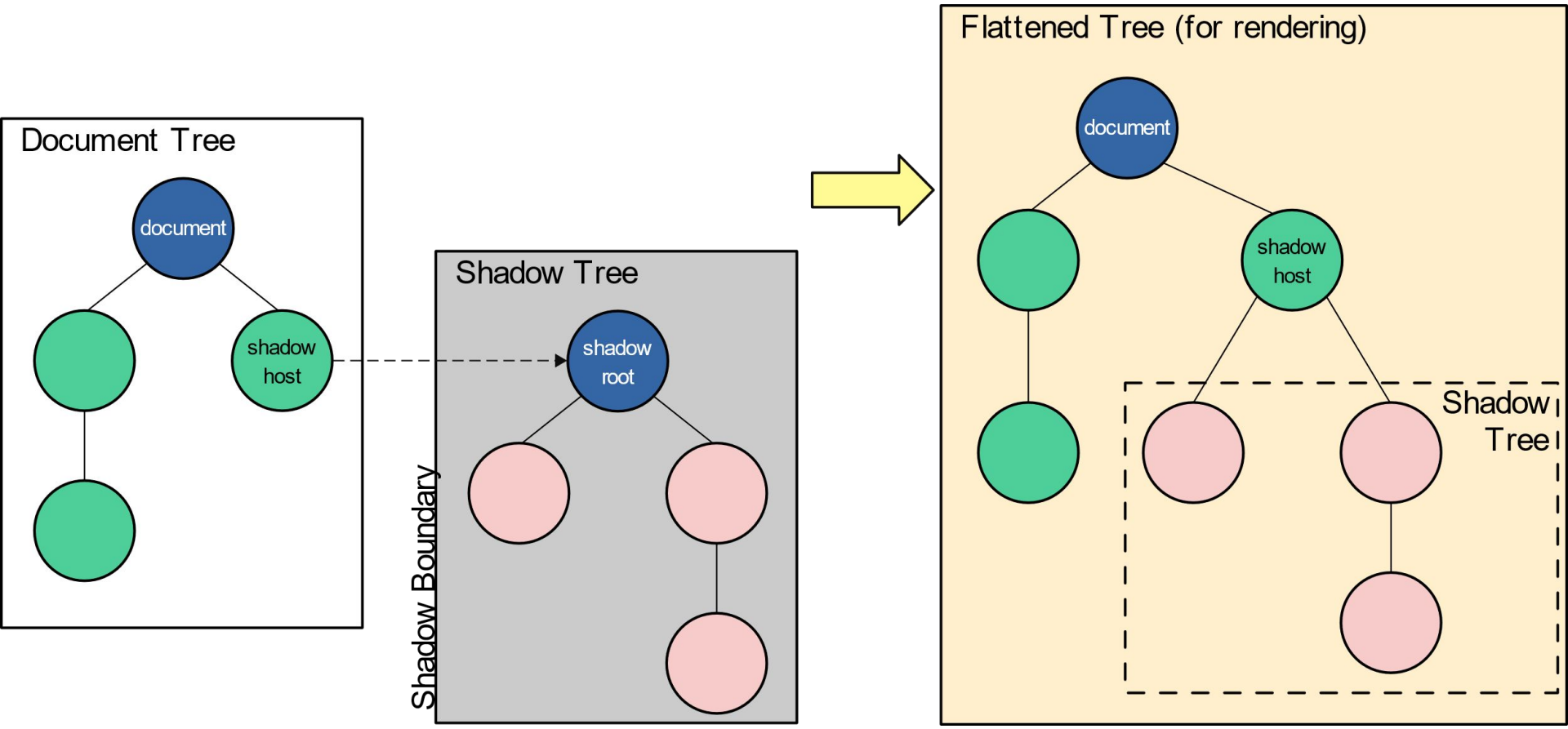
Spletne komponente - standardi

- potreben je čas, da brskalniki podprejo nove standarde
  - stari brskalniki imajo pogosto težave z novimi standardi
- ko pa so novosti podprte v brskalnikih, pa imajo brskalniki dobro podporo za nazaj
  - za razliko od mnogih JavaScript ogrodij se programski vmesniki ohranijo

Spletne komponente - osnove

- zaobsegajo 4 tehnologije / specifikacije
  - senčni DOM (shadow DOM) - množica JavaScript programskih vmesnikov (API) za pripenjanje ločenega DOM drevesa v glavno DOM drevo oz. element
    - drevo je upodobljeno (rendered) ločeno od glavnega DOM drevesa, zaradi česar so lastnosti elementa zasebne in neodvisne od drugih delov HTML dokumenta
  - elementi po meri (custom elements) - množica JavaScript programskih vmesnikov (API) za definiranje novih HTML elementov
  - HTML predloge - elementa <template> in <slot> omogočata izdelavo označevalnih predlog, ki se ne prikažejo na osnovni strani
    - predloge so lahko večkrat ponovno uporabljene
    - predstavljajo osnovo za elemente po meri
    - so le shranjene (se ne uporabijo/prikažejo takoj)
  - ECMAScript moduli (Modules) - definiranje modulov v ECMAScript

Senčni DOM (Shadow DOM)



Senčni DOM (Shadow DOM)

- prednosti uporabe senčnega DOMa
  - ograjevanje kode
  - ograjevanje stilov
  - ograjevanje obnašanja
- slabosti uporabe senčnega DOMa
  - ne delujejo vse knjižnice s senčnim DOMom
  - včasih ne želimo izolacije stilov, temveč želimo globalne stile

- v senčni DOM je možno prilepiti zunanji stil

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

Senčni DOM (Shadow DOM)

- načina uporabe
  - odprti način - iz glavne strani lahko dostopamo do senčnega DOMa
    - attachShadow({mode: 'open'})
    - let myShadowDom = myCustomElem.shadowRoot;
  - zaprti način - iz glavne strani ne moremo dostopati do senčnega DOMa
    - attachShadow({mode: 'closed'})
    - myCustomElem.shadowRoot vrne null

HTML elementi po meri

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

HTML elementi po meri

- avtonomni elementi po meri
  - registracija
    - customElements.define(' my-paragraph1', MyParagraph1);
  - uporabljamo jih kot HTML elemente s HTML oznako, npr.
    - <my-paragraph1></my-paragraph1> ali
    - document.createElement("my-paragraph1")
- prilagojeni elementi po meri
  - registracija
    - customElements.define('my-paragraph2', MyParagraph2, { extends: 'p' });
  - uporabljamo jih 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" })

HTML predloge elementov - oznaka template

- HTML označba <template> je predloga za HTML element
  - od ECMAScript 2015 (ES6) dalje
- 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 template ne uporabi, se skripte ne izvedejo, slike se ne naložijo, audio se ne predvaja itd.
  - dostop do elementov v predlogi ni mogoče npr. preko getElementById() ali querySelector
- predloga se za prikaz uporabi v času izvajanja z JavaScript kodo
  - večkratna ponovna uporaba predloge
    - pogosto želimo doseči ponovno uporabo posameznih delov strani

HTML predloge elementov - primer

```
<template id="my-paragraph">
    <style>
    p {
        color: white;
        background-color: rgb(117, 17, 121);
        padding: 15px;
    }
    </style>
    <p>My paragraph</p>
</template>

  let template = document.getElementById('my-paragraph');
  let templateContent = template.content;

  const shadowRoot = this.attachShadow({mode: 'open'})
    .appendChild(templateContent.cloneNode(true));
```

HTML predloge elementov - oznaka slot

- HTML označba <slot> - reža
- predloga se razdeli na manjše dele (reže)
- vsak del (reža) ima lahko svoj stil

```
<template id="my-paragraph">
  <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-paragraph3>
   <span slot="my-text">Let's have some different text!</span>
 </my-paragraph3>
```

Avtonomni element po meri - primer

```
customElements.define('my-paragraph1',
  class extends HTMLElement {
    constructor() {
      super();
      let template = document.getElementById('my-paragraph');
      let templateContent = template.content;

      const shadowRoot = this.attachShadow({mode: 'open'})
        .appendChild(templateContent.cloneNode(true));
    }
  }
);
```

Avtonomni element po meri – primer uporabe 1

```
<!DOCTYPE html>
<html>
    <head>
        <script src="DemoWebComponents1.js" defer></script>
    </head>
<body>
    <p>Demo Web Components 1</p>

    <template id="my-paragraph">
        <style>
        p {
            color: white;
            background-color: rgb(117, 17, 121);
            padding: 15px;
        }
        </style>
        <p>My paragraph</p>
    </template>

  <my-paragraph1></my-paragraph1>

</body>
</html>
```

Avtonomni element po meri – primer uporabe 2

```
<!DOCTYPE html>
<html>
    <head>
        <script src="DemoWebComponents1.js" defer></script>
    </head>
<body>
    <p>Demo Web Components 2</p>

    <template id="my-paragraph">
      <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>
```

Prilagojeni element po meri - primer

```
class MyParagraph2 extends HTMLParagraphElement {
    constructor() {
      super();
      let template = document.getElementById('my-paragraph');
      let templateContent = template.content;

      const shadowRoot = this.attachShadow({mode: 'open'})
        .appendChild(templateContent.cloneNode(true));
    }
  }

  customElements.define('my-paragraph2', MyParagraph2, {
extends: 'p' });
```

Prilagojeni element po meri - primer

```
<!DOCTYPE html>
<html>
    <head>
        <script src="DemoWebComponents2.js" defer></script>
    </head>
<body>

    <p>Demo Web Components 2</p>

    <template id="my-paragraph">
        <style>
        p {
            color: white;
            background-color: rgb(117, 17, 121);
            padding: 15px;
        }
        </style>
        <p>My paragraph</p>
    </template>

    <p is="my-paragraph2">
</body>
</html>
```

Implementacije spletnih komponent

- v splošnem izdelava spletne komponente poteka po teh korakih
  1. ustvarjanje razreda za element po meri - razred specificira funkcionalnost spletne komponente z uporabo ECMAScript 2015 sintakse)
  2. registriranje novega elementa po meri
  3. po potrebi dodajanje senčnega DOMa v element po meri z uporabo metode Element.attachShadow()
     - v senčni DOM se z uporabo DOM metod dodajo vgnezdeni elementi (child elements), poslušalci dogodkov (event listeners)
  4. po potrebi definiranje HTML predloge z uporabo <template> in <slot>
     - z uporabom DOM metod kloniramo HTML predlogo in jo pripnemo v senčni DOM
  5. uporaba novega elementa po meri kjerkoli na spletni strani na enak način kot se uporabi drug HTML dokument

Spletne komponente - podpora v brskalnikih

- spletne komponente so podprte v brskalnikih Chrome, Firefox, Microsoft Edge, Opera
  - dobra podpora v brskalnikih, ki temeljijo na Chromium
- Safari je v zadnjem času zelo izboljšal podporo za spletne komponente, vendar ne podpira spletnih komponent čisto 100%
  - 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
- https://caniuse.com/?search=web%20components

Ponudba spletnih komponent

- obstaja kar nekaj knjižnic za spletne komponente
  - FASTElement, snuggsi, X-Tag, Slim.js, Lit, Smart, Stencil, hyperHTML-Element, DataFormsJS, Polymer, Bosonic, Riot.js
- vedno večja skupnost
  - WebComponents.org
- uporaba spletnih komponent v navezavi s spletnimi ogrodji
  - Angular, React, Vue.js itd.
  - Vaadin Web Components

Razvoj spletnih aplikacij v Javi

- standardi Java EE (Java Enterprise Edition) / Jakarta EE
  - leta 2019 preimenovanje Java EE -> Jakarta EE
    - preimenovanje v kodi javax -> jakarta
  - Java/Jakarta EE
    - 1999: J2EE 1.2, 2017: Java EE 8.0, 2022: Jakarta EE 10
  - Java/Jakarta Servlet
    - 1996: Java Servlet 1.0, 2017: Java Servlet 4.0, 2021: Jakarta Servlet 6.0
  - JavaServer Pages / Jakarta Server Pages (JSP)
    - 1999: JSP 1.1, 2003: JSP 2.0, 2020: Jakarta Server Pages 3.0
  - JavaServer Faces / Jakarta Server Faces (JSF)
    - 2004: JSF 1.0, 2017: JSF 2.3, 2022: Jakarta Server Faces 4.0
  - Java/Jakarta Expression Language (EL)
    - 2005: JSTL 1.0, 2013: Expression Language 3.0, 2020: Jakarta Expression Language 4.0
  - Java/Jakarta WebSocket API
    - 2014: JavaAPI for WebSocket 1.1, 2022: Jakarta WebSocket 2.1

Ogrodja za razvoj spletnih aplikacij v Javi

- skozi leta se je pojavilo veliko ogrodij
  - JavaServer Faces
  - Spring
  - Struts
  - Google Web Toolkit
  - Grails
  - Apache Wicket
  - Tapestry
  - PrimeFaces
  - IceFaces
  - Vaadin
  - itd.

Uporaba spletnih komponent na IZUMu

- Institut informacijskih znanosti (IZUM) - informacijski servis slovenske znanosti, kulture in izobraževanja
  - razvoj sistema in servisov COBISS (temelj knjižničnega informacijskega sistema Slovenije in nekaterih drugih držav, ki so povezani v mrežo COBISS.net)
- spletne komponente v 3 aplikacijah
  - Digitalni repozitorij COBISS (dCOBISS)
    - https://d.cobiss.net/repository/
  - Informacijski sistem o raziskovalni dejavnosti v Sloveniji (SICRIS)
    - https://www.sicris.si/
  - COBISS4 (nova generacija programske opreme za knjižničarje, nadgrajuje COBISS3)
    - v razvoju

Izbrana ogrodja v dCOBISS in SICRIS

- ključna ogrodja v dCOBISS in SICRIS so:
  - JavaServer Faces (JSF)
  - Krazo
    - javansko ogrodje za MVC (Model-View-Controller)
  - Bootstrap
    - CSS ogrodje za razvoj odzivnih spletnih aplikacij

- + spletne komponente

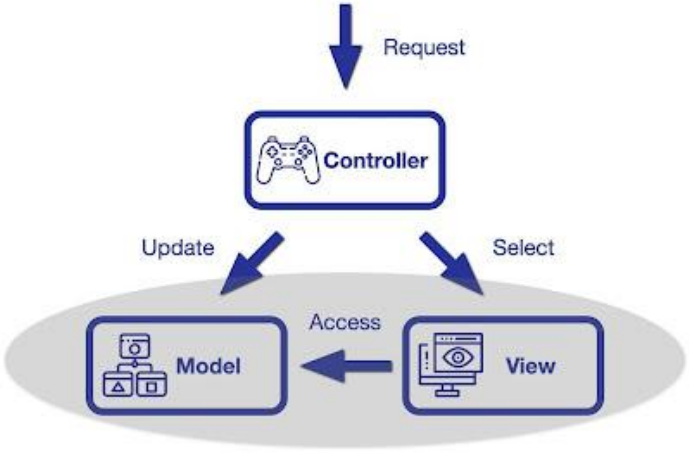
- klasičnega JavaScripta-a relativno malo

Krazo

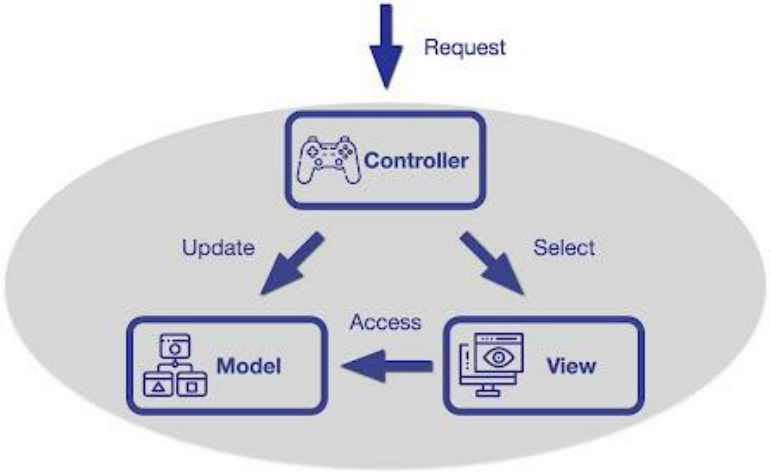
- MVC (Model-View-Controller)
  - začetki v Java Community Process (JSR 371: Model-View-Controller (MVC 1.0) Specification), dolgo časa planirano za Java EE 8
  - trenutna zadnja verzija Jakarta MVC 2.1 (Eclipse Krazo)
- Krazo implementira programski vmesnik MVC, ki je manjkajoči del v JSF (JavaServer Faces)
  - podobne MVC rešitve ponujajo tudi druga ogrodja, npr. Spring
- MVC
  - Model – aplikacijski podatki
  - View – predstavitev aplikacijskih podatkov
  - Controller – del sistema, odgovoren za upravljanje vnosa, posodobitve modela in produciranje rezultatov

JSF MVC vs. Krazo MVC

JSF MVC



Krazo MVC



Krazo/MVC Controller

- Controller definiran s pomočjo JAX-RS anotacij

```java
@Controller
@Path("hello")
public class HelloController {

    @GET @Path("void")
    @View("hello.jsp")
    public void helloVoid() {
    }

    @GET @Path("string")
    public String helloString() {
        return "hello.jsp";
    }

    @GET @Path("response")
    public Response helloResponse() {
        return Response.status(Response.Status.OK)
            .entity("hello.jsp")
            .build();
    }
}
```

```java
@GET
@Controller
public String redirect() {
    return "redirect:see/here";
}
```

Krazo/MVC Model

- uporaba CDI anotacij @Named in @Inject

```java
@Named("greeting")
@RequestScoped
public class Greeting {

    private String message;

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }
    //...
}
```

```java
@Path("hello")
public class HelloController {

    @Inject
    private Greeting greeting;

    @GET
    @Controller
    public String hello() {
        greeting.setMessage("Hello there!");
        return "hello.jsp";
    }
}
```

```java
@Path("hello")
public class HelloController {

    @Inject
    private Models models;

    @GET
    @Controller
    public String hello() {
        models.put("greeting", new Greeting("Hello there!"));
        return "hello.jsp";
    }
}
```

Krazo/MVC View

- uporaba predlog (JSP, Facelets/JSF itd.), uporabljajo se podatki iz modela

```
<!DOCTYPE html>
<html>
    <head>
        <title>Hello</title>
    </head>
    <body>
        <h1>${greeting.message}</h1>
    </body>
</html>
```

```
<!DOCTYPE html>
<html lang="en" xmlns:h="http://xmlns.jcp.org/jsf/html">
    <h:head>
        <title>Hello</title>
    </h:head>
    <h:body>
        <h:outputText value="#{greeting.message}" />
    </h:body>
</html>
```

Uporaba spletnih komponent v dCOBISS

- spletne komponente se uporabljajo za večino uporabniškega vmesnika dCOBISS (prikaz, urejanje itd.)
- dCOBISS je aplikacija za knjižničarje, ne za končne uporabnike
  - 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

Uporaba spletnih komponent v dCOBISS - primer

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>
```

Uporaba spletnih komponent v dCOBISS - primer

project-view.js

```
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>
```

Uporaba spletnih komponent v SICRIS

- spletne komponente se uporabljajo za večino uporabniškega vmesnika SICRIS (prikaz, urejanje itd.)
- SICRIS je aplikacija za končne uporabnike
  - izgled je zelo pomemben
- HTML struktura je definirana večinoma na strežniški strani v JSF XHTML
  - na odjemalski strani v spletni komponenti ni definirana HTML struktura
  - pristop prijaznajši za oblikovalce uporabniškega vmesnika, ki so navajeni na XHTML datoteke
  - SICRIS je single page aplikacija in vsebine se pridobivajo na asinhroni način preko spletnih komponent

Uporaba spletnih komponent v SICRIS - primer


```
<ul class="autocomplete-items" role="listbox">
  <c-autocomplete-rsr r-id="30228" fn="NADA" ln="BARBARIČ-NOVAK" mst-id="14606" q="novak">
    <li role="option">
      <c-details-link>
        <a href="si/sl/researcher/30228?q=novak"> NADA BARBARIČ-NOVAK [14606]</a>
      </c-details-link>
    </li>
  </c-autocomplete-rsr>
  <c-autocomplete-rsr r-id="13571" fn="Vladimira" ln="Berlič Novak" mst-id="16172" q="novak">…</c-autocomplete-rsr>
  <c-autocomplete-rsr r-id="43712" ttl="dr." fn="Sabina" ln="Devjak Novak" mst-id="37868" q="novak">…</c-autocomplete-rsr>
  <c-autocomplete-rsr r-id="4231" ttl="mag." fn="Maja" ln="Dolenc-Novak" mst-id="01323" q="novak">…</c-autocomplete-rsr>
```

```
@GET
@Path("/autocomplete/{query}")
@Produces({MediaType.TEXT_HTML})
public String getResearchersAutocomplete(@PathParam("query") String query) {
    List<ResearcherView> researchers = researcherRepository.
            autocomplete(ctxCard, query);
    CommonUtils cmnUtils = new CommonUtils(ctxCard, translator);
    cmnUtils.setTitle(researchers);
    models.put(RESEARCHERS, researchers);
    models.put("query", query);
    return "researcher/autocomplete.xhtml";
}
```

```
<ui:composition xmlns:ui="http://java.sun.com/jsf/facelets"
 xmlns="http://www.w3.org/1999/xhtml">
 <ui:repeat var="rsr" value="#{researchers}">
  <c-autocomplete-rsr
   r-id="#{rsr.id}"
   ttl="#{rsr.title}"
   fn="#{rsr.firstName}"
   ln="#{rsr.lastName}"
   mst-id="#{rsr.mstid}"
   q="#{query}"></c-autocomplete-rsr>
 </ui:repeat>
</ui:composition>
```

Uporaba spletnih komponent v SICRIS - primer

SICRIS Informacijski sistem o raziskovalni dejavnosti v Sloveniji

```
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;¤¶
    }¤¶
    let li = document.createElement("LI");¤¶
    li.setAttribute("role", "option");¤¶
    li.innerHTML = `<c-details-link><a href="${¤¶
      window.appData.url¤¶
    }/researcher/${id}?q=${q}"> ${title} ${boldStart(firstName, q)} ${boldStart(
      lastName,¤¶
      q¤¶
    )} [${boldStart(mstid, q)}]</a></c-details-link>`;¤¶
    li.onclick = () => {¤¶
      li.querySelector("a").click();¤¶
    };¤¶
    this.appendChild(li);¤¶
  }¤¶
}¤¶
¤¶
window.customElements.define("c-autocomplete-rsr", CAutocompleteRsr);¤¶
```

Uporaba spletnih komponent v SICRIS - primer

- inicializacija spletne komponente
  - HTML elementi po meri: constructor() ali connectedCallback()?
  - constructor() se pokliče, ko je spletna komponenta kreirana
  - connectedCallback() se pokliče potem, ko je spletna komponenta pripeta v DOM
- v SICRIS in dCOBISS se uporablja predvsem connectedCallback()

```
class MyWebComponent extends HTMLElement {
  constructor() {
    super()
    console.info( 'constructed' )
  }
  connectedCallback() {
    console.info( 'connected' )
    this.innerHTML = 'Hello'  //can't be set in constructor()
  }
}
customElements.define('my-component', MyWebComponent);
```

Uporaba spletnih komponent v ogrodju Vaadin

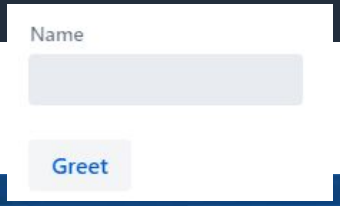
- Vaadin je odprtokodna platforma za razvoj spletnih aplikacij
- Vaadin Flow je ogrodje za razvoj spletnih aplikacij v Javi
  - omogoča izgradnjo spletnih aplikacij 100% v Javi
    - brez programiranja v JavaScript, HTML itd.
  - javanski razredi se preslikajo v spletne komponente

```
@Route("hello-world")
public class HelloWorld extends VerticalLayout {

    public HelloWorld() {
        TextField name = new TextField("Name");
        Paragraph greeting = new Paragraph("");

        Button button = new Button("Greet", event -> {
            greeting.setText("Hello " + name.getValue());
        });

        add(name, button, greeting);
    }
}
```



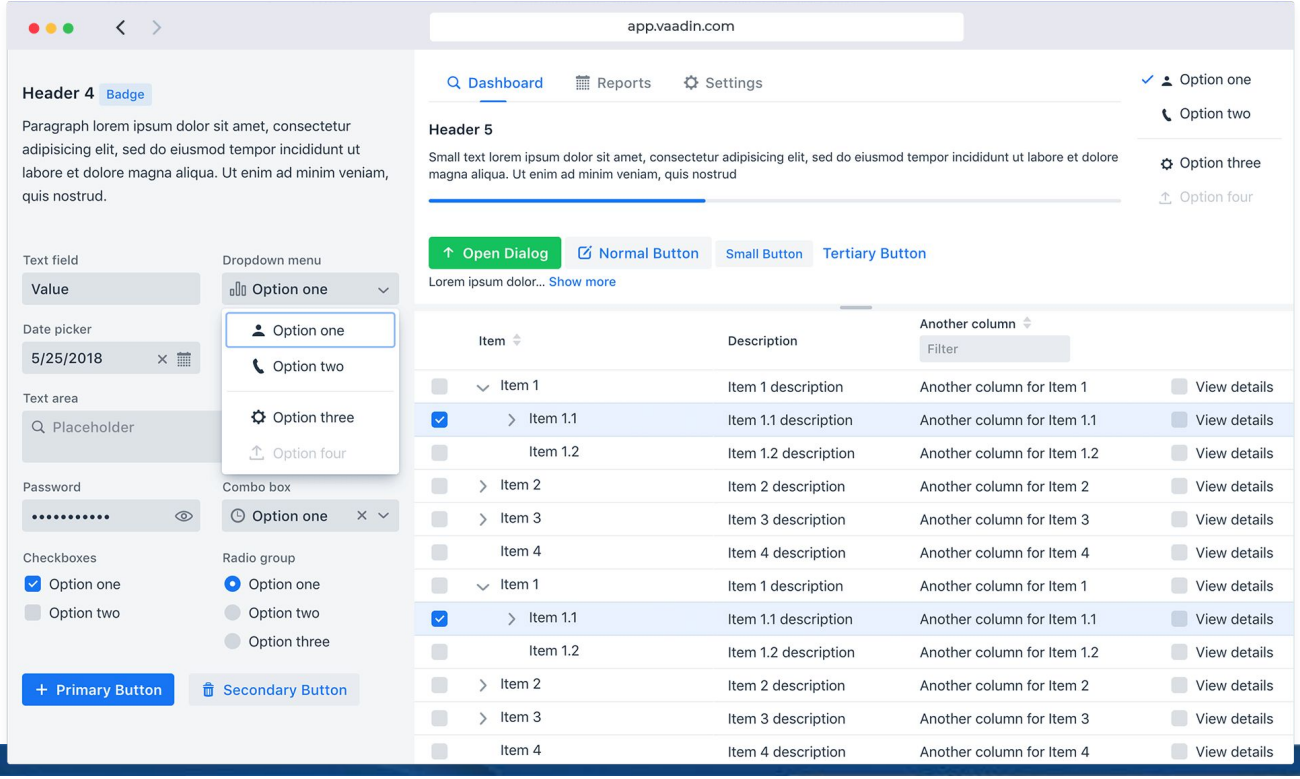

```
<vaadin-vertical-layout theme="padding spacing" style="width: 100%;"> flex
  #shadow-root (open)
  <vaadin-text-field has-label> flex
    #shadow-root (open)
      ::before flex
      <input slot="input" type="text" id="vaadin-text-field-0" aria-labelledby=
      "label-vaadin-text-field-0">
      <label slot="label" id="label-vaadin-text-field-0" for="vaadin-text-field-
      0">Name</label>
      <div slot="error-message" id="error-message-vaadin-text-field-0" hidden>
      </div>
    </vaadin-text-field>
  <vaadin-button tabindex="0" role="button">
    #shadow-root (open)
      ::before
      "Greet"
      ::after
    </vaadin-button>
    <p></p>
</vaadin-vertical-layout>
```

Uporaba spletnih komponent v ogrodju Vaadin

- drugačen pristop k izgradnji spletnih aplikacij
  - razvijalci ne razmišljajo o HTTP requestih in responsih
  - podobno kot pri tradicionalnih namiznih aplikacijah (npr. Swing) se 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
  - podobno kot druge spletne komponente delujejo v vseh brskalnikih
  - večina logike za grafični vmesnik se izvede na strežniku (manjša varnostna 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

Uporaba spletnih komponent v ogrodju Vaadin

- 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.



Uporaba spletnih komponent v ogrodju Vaadin

- Vaadinove spletne komponente temeljijo na modernih temeljih
  - temeljijo na spletnih standardih (W3C/WHATG)
    - v komponentah se uporablja senčni DOM
  - možno jih je prilagajati z različnimi temami (CSS)
  - možno jih je uporabljati v praktično vseh sodobnih ogrodjih za front-end
  - ponujajo programski vmesnik za Javo in TypeScript
  - prijazne za mobilne naprave (odzivne in optimizirane za različne velikosti zaslonov)
  - podpirajo spletno dostopnost v skladu s standardom WCAG (bralnik zaslona itd.)

Uporaba Vaadin v drugih spletnih aplikacijah

- Vaadinove spletne komponente je možno uporabljati direktno v HTML

```
<!DOCTYPE html>
<html>
<head>
<title>Layout</title>
<base
href="https://raw-dot-custom-elements.appspot.com/vaadin/vaadin-ordered-layout/v1.4.0/vaadin-order
ed-layout/">
<script src="../webcomponentsjs/webcomponents-lite.js"></script>
<link rel="import" href="vaadin-horizontal-layout.html">
<link rel="import" href="vaadin-vertical-layout.html">
</head>
<body>

<vaadin-horizontal-layout theme="spacing padding" class="height-5xl">
  <div>Horizontally</div>
  <div>Aligned</div>
  <layout-item>Item 2</layout-item>
  <layout-item theme="inactive">Item 3</layout-item>
</vaadin-horizontal-layout>
<br/>
<vaadin-vertical-layout>
  <div>Vertically</div>
  <div>Aligned</div>
</vaadin-vertical-layout>
</body>
</html>
```

Uporaba Vaadin v COBISS4

- obstoječa aplikacija za knjižničarje COBISS3 je Swing aplikacija
- Vaadin omogoča lažji prehod iz Swing na splet
  - programski vmesnik Vaadin zelo spominja na Swing
  - obstoječe ogrodje za izgradnjo grafičnega vmesnika je bilo lažje prilagoditi v primeru uporabe Vaadin
  - pri izgradnji spletne aplikacije želimo pouporabiti čim večji del obstoječe kode, ki ni direktno vezana na grafični vmesnik

```
final JTextField textField = new JTextField();
JButton button = new JButton();
button.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        System.out.println("You typed: " + textField.getText());
    }
});
add(textField);
add(button);
```

```
final TextField textField = new TextField();
Button button = new Button();
button.addClickListener(new Button.ClickListener() {
    @Override
    public void buttonClick(Button.ClickEvent event) {
        System.out.println("You typed: " + textField.getValue());
    }
});
addComponent(textField);
addComponent(button);
```

Zaključek

- spletne komponente so bodočnost razvoja spletnih aplikacij
  - temeljijo na spletnih standardih, ki imajo dobro podporo v brskalnikih
  - imajo daljšo življenjsko dobo kot marsikatere druge alternativne rešitve
  - omogočajo gradnjo ponovno uporabnih enot, do katerih se dostopa preko programskih vmesnikov, notranja implementacija pa je skrita
  - možen je klasični pristop h gradnji aplikacij (npr. z uporabo JSF) ali pa javanski pristop (Vaadin)
- naše izkušnje so pozitivne
  - z uporabo spletnih komponent nameravamo nadaljevati tudi v bodočnosti