



CS230 - DISTRIBUIRANI SISTEMI

Java API za WebSocket

Lekcija 12

PRIRUČNIK ZA STUDENTE

CS230 - DISTRIBUIRANI SISTEMI

Lekcija 12

JAVA API ZA WEBSOCKET

- ✓ Java API za WebSocket
- ✓ Poglavlje 1: Analiza WebSocket koda iz NetBeans IDE primera
- ✓ Poglavlje 2: WebSocket krajnja tačka
- ✓ Poglavlje 3: Kreiranje vlastite WebSocket aplikacije
- ✓ Poglavlje 4: Studija slučaja za Java EE, WebSocket, JS i HTML5
- ✓ Poglavlje 5: Pokazni primer - Java API za WebSocket
- ✓ Poglavlje 6: Individualna vežba 12
- ✓ Poglavlje 7: Domaći zadatak 12
- ✓ Zaključak

Copyright © 2017 – UNIVERZITET METROPOLITAN, Beograd. Sva prava zadržana. Bez prethodne pismene dozvole od strane Univerziteta METROPOLITAN zabranjena je reprodukcija, transfer, distribucija ili memorisanje nekog dela ili čitavih sadržaja ovog dokumenta., kopiranjem, snimanjem, elektronskim putem, skeniranjem ili na bilo koji drugi način.

Copyright © 2017 BELGRADE METROPOLITAN UNIVERSITY. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, without the prior written permission of Belgrade Metropolitan University.

▼ Uvod

UVOD

Tradicionalno, veb aplikacije se grade nad modelom request / response.

U ovoj lekciji, poseban akcenat će biti na analizi i diskusiji koja je u vezi sa problematikom *Java API za WebSocket*. Na samom početku bi trebalo istaći da, tradicionalno posmatrano, veb aplikacije se grade nad modelom zahtev / odgovor (*request / response*). Uprošćeno rečeno, veb pregledač, koji je učitao izvesnu klijent stranicu, šalje *HTTP zahtev* serveru koji, nakon procesiranja zahteva, vraća *HTTP odgovor*.

WebSocket je nova *HTML5* tehnologija koja omogućava dvosmernu, punu dupleks (*full duplex*) komunikaciju između klijenta (obično se radi o veb pregledaču) i servera. Drugim rečima, primenom ove tehnologije je omogućeno je da serveri šalju podatke klijentima (u savremenim aplikacijama dominantno veb pregledačima) bez potrebe za čekanjem *HTTP zahteva*. Java EE 7 obezbeđuje potpunu podršku za razvoj *WebSocket* distribuiranih aplikacija. Ova podrška značajno dobija jednu kvalitetniju dimenziju kada se koristi u sinergiji sa savremenim Java razvojnim okruženjima. Tako, *NetBeans IDE* čini razvoj Java *WebSocket* aplikacija značajno olakšanim.

Imajući u vidu navedeno, lekcija će se bazirati na pažljivo biranim temama koje obrađuju razvoj Java veb aplikacija sa *WebSocket* podrškom. Posebno, svako od izlaganja će biti pokriveno pažljivo odabranim primerima koji služe svrsi lakšeg i kompletnijeg razumevanja i savladavanja aktuelne problematike.

Lekcija će voditi izlaganje u dva smera:

- Analiza i diskusija *WebSocket* koda primenom primera koji je ugrađen u *NetBeans IDE* dokumentaciju;
- Razvoj vlastite Java EE aplikacije primenom *WebSocket* tehnologije.

Savladavanjem lekcije studenti će biti u potpunosti osposobljeni da kreiraju vlastite Java EE aplikacije primenom *WebSocket* tehnologije.

▼ Poglavlje 1

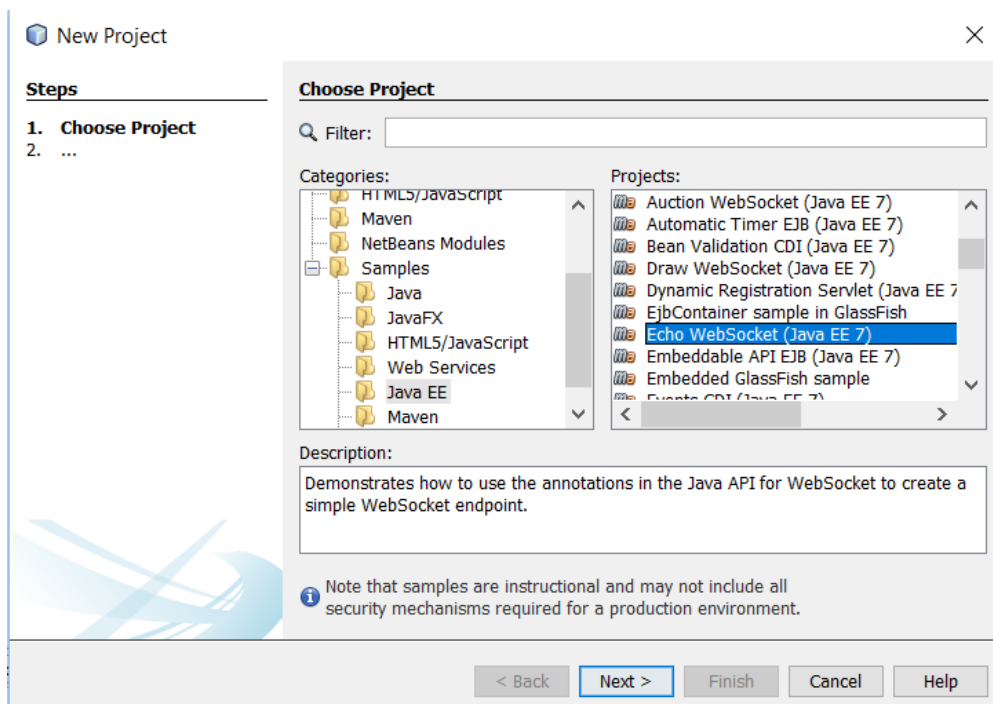
Analiza WebSocket koda iz NetBeans IDE primera

PROJEKAT NETBEANS IDE PRIMERA

Biće iskorišćen NetBeans IDE projekat "Echo" za demonstraciju WebSocket aplikacije.

Dokumentacija razvojnog okruženja [NetBeans IDE](#) sadrži veliki broj projekata koji mogu da posluže kao osnov programerima za započinjanje vlastitih projekata. Jedan posebno koristan primer, sadržan u pratećoj [NetBeans IDE](#) dokumentaciji, jeste [Echo](#) aplikacija koja koristi [WebSocket](#) tehnologiju za plasiranje određenih serverskih podataka ka veb pregledaču.

U svrhu navedenog, biće pristupljeno kreiranju projekta primera ([Sample Project](#)) u razvojnem okruženju [NetBeans IDE](#). Za početak počinje se od izbora [File | New Project](#), a zatim se pod kategorijama ([Categories](#)) bira podopcija [Java EE](#) opcije [Samples](#). Kao nov projekat, u nastavku, bira se [Echo WebSocket \(Java EE 7\)](#) iz liste ponuđenih projekata. Navedeno je prikazano sledećom slikom.



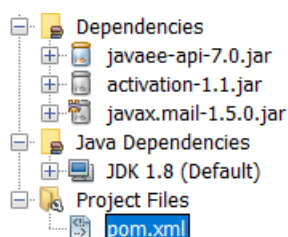
Slika 1.1 Kreiranje novog projekta iz NetBeans IDE primera [izvor: autor]

U sledećem prozoru, prihvataju se ponuđena podešavanja i vrši se klik na dugme [Finish](#). U kreirani projekat dodat je primer iz [NetBeans IDE](#) prateće dokumentacije.

ANGAŽOVANJE "ECHO" APLIKACIJE

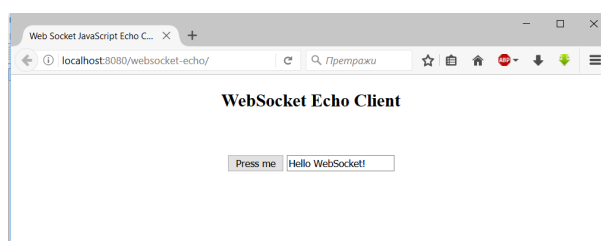
U nastavku biće testiran učitani primer u NetBeans IDE projekat.

Kada se pogleda *NetBeans IDE* primer *Echo*, koji je učitani u kreirani primer projekta, moguće je primetiti da on koristi *Maven* alat za kreiranje (**build tool**) preko kojeg primenjuje odgovarajuće zavisnosti, koristeći dobro poznati Maven konfiguracioni fajl *pom.xml*. Sledećom slikom su prikazane zavisnosti ugrađene u projekat *Echo*. Nakon slike sledi listing konfiguracione Maven datoteke.



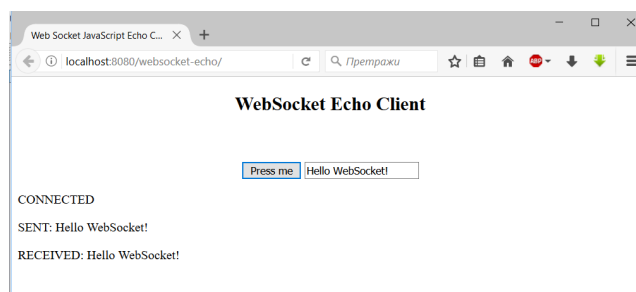
Slika 1.2 Zavisnosti ugrađene u Maven projekat Echo [izvor: autor]

Klikom na dugme *Run Project*, u razvojnom okruženju *Net Beans IDE*, vrši se prevođenje aplikacije i njeno pokretanje. U veb pregledaču se ubrzo prikazuje sledeća stranica.



Slika 1.3 Početna stranica aplikacije "Echo" [izvor: autor]

Klikom na dugme *Press me* na stranici se pojavljuje nov sadržaj, upravo na način prikazan sledećom slikom.



Slika 1.4 Reagovanje aplikacije na klik na dugme "Press me" [izvor: autor]

Tekst *Hello WebSocket!*, koji se nalazi u polju za unos teksta na stranici, unapred je određen i automatski ubačen u navedenu kontrolu. Klikom na dugme *Press me*, tekst se šalje u krajnju tačku *WebSocket* servera (**WebSocket server endpoint**) koji, u nastavku, jednostavno šalje nazad primljeni **String** klijentu. Upravo je to ilustrovano Slikom 5.

Iz perspektive analize i diskusije, od posebnog značaja su dve datoteke:

- *index.html* - datoteka sadrži JavaScript funkcije koje generišu *WebSocket* događaje;
- *EchoEndpoint.java* - odgovarajuća Java klasa koja obrađuje ove događaje i o kojoj će prvo biti reči.

▼ Poglavlje 2

WebSocket krajnja tačka

KLASA KRAJNJE TAČKE WEBSOCKET SERVERA

Kod koji će prvo bit analiziran i diskutovan pripada klasi krajnje tačke WebSocket servera.

Nakon demonstracije *NetBeans IDE* ugrađenog projekta *Echo* sledi analiza koda kojim su snabdevene njegove datoteke koje su pomenute u prethodnom izlaganju. Kod koji će prvo bit analiziran i diskutovan pripada klasi krajnje tačke *WebSocket* servera. U konkretnom slučaju klasa se naziva *EchoEndpoint.java* i sledećim listingom je priložen njen kod:

```
package org.glassfish.samples.websocket.echo;

import javax.websocket.OnMessage;
import javax.websocket.server.ServerEndpoint;

@ServerEndpoint("/echo")
public class EchoEndpoint {

    @OnMessage
    public String echo(String message) {
        return message;
    }
}
```

Klasa koja obrađuje WebSocket zahtev (WebSocket requests) na serverskoj strani naziva se krajnja tačka servera (server endpoint).

Iz priloženog listinga je, nesporno, moguće utvrditi da je za razvoj klase krajnje tačke *WebSocket servera* (WebSocket server endpoint), primenom Java API za WebSocket, neophodno primeniti veoma malo konkretnog Java koda.

Java klasu krajnje tačke WebSocket servera neophodno je, kao što je moguće primetiti u priloženom kodu, obeležiti anotacijom **@ServerEndpoint**. Vrednost koja je istaknuta, u okviru anotacije **@ServerEndpoint**, ukazuje na uniformni identifikator resursa (URI - **Uniform Resource Identifier**) krajnje tačke servera i klijente koji pristupaju krajnjoj tački servera preko navedenog URI.

Svaka metoda koja je obeležena anotacijom `@OnMessage` poziva se automatski svaki puta kada klijent pošalje poruku krajnjoj tački `WebSocket` servera. Metoda, koja je obeležena ovom anotacijom, preuzima `String` argument, koji sadrži kontekst poruke koju je klijent poslao. Sadržaj poruke može biti bilo šta. Međutim, ovo predstavlja opšte prihvaćen način da klijent pošalje podatke u `JSON` formatu. U ovom konkretnom primeru, metoda `echo()`, koja je obeležena anotacijom `@OnMessage`, prihvata jednostavan `String` podatak.

Povratna vrednost metode, koja je obeležena anotacijom `@OnMessage`, prosleđuje se nazad klijentu. Iako je najčešće slučaj da se podatak u `JSON` formatu šalje, u ovom jednostavnom primeru `String` objekat je primljen kao argument metode i vraćen nazad klijentu.

ANALIZA PRISUTNOG JAVASCRIPT KODA

Klijent strana je reprezentovana HTML stranicom `index.html` sa dodatnim JavaScript kodom.

Klijent strana je reprezentovana `HTML` stranicom `index.html`. Datoteka `index.html` sadrži `JavaScript` funkcije koje generišu `WebSocket` događaje za komunikaciju sa krajnjom tačkom `WebSocket` servera. Sledećim listingom je priložen kod ove stranice.

```
<html>
<head>
  <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
</head>

<body>
<meta charset="utf-8">
<title>Web Socket JavaScript Echo Client</title>
<script language="javascript" type="text/javascript">
  var wsUri = getRootUri() + "/websocket-echo/echo";

  function getRootUri() {
    return "ws://" + (document.location.hostname == "" ? "localhost" :
document.location.hostname) + ":" +
      (document.location.port == "" ? "8080" : document.location.port);
  }

  function init() {
    output = document.getElementById("output");
  }

  function send_echo() {

    websocket = new WebSocket(wsUri);
    websocket.onopen = function (evt) {
      onOpen(evt)
    };
    websocket.onmessage = function (evt) {
      onMessage(evt)
    };
  }
};
```



```

        websocket.onerror = function (evt) {
            onError(evt)
        };

    }

    function onOpen(evt) {
        writeToScreen("CONNECTED");
        doSend(textID.value);
    }

    function onMessage(evt) {
        writeToScreen("RECEIVED: " + evt.data);
    }

    function onError(evt) {
        writeToScreen('<span style="color: red;">ERROR:</span> ' + evt.data);
    }

    function doSend(message) {
        writeToScreen("SENT: " + message);
        websocket.send(message);
    }

    function writeToScreen(message) {
        var pre = document.createElement("p");
        pre.style.wordWrap = "break-word";
        pre.innerHTML = message;
        //alert(output);
        output.appendChild(pre);
    }

    window.addEventListener("load", init, false);

</script>

<h2 style="text-align: center;">WebSocket Echo Client</h2>

<br/></br>

<div style="text-align: center;">
    <form action="">
        <input onclick="send_echo()" value="Press me" type="button">
        <input id="textID" name="message" value="Hello WebSocket!" type="text"><br/>
    </form>
</div>
<div id="output"></div>
</body>
</html>

```

U nastavku sledi analiza i diskusija elemenata priloženog klijent koda. Od posebnog značaja, za ovaj deo izlaganja, jeste deo koda koji se nalazi pod tagom **<script>**. Ako se pogleda

detaljno istaknuti deo koda, moguće je primetiti da *JavaScript* `send_echo()` funkcija kreira nov *JavaScript* `WebSocket` objekat i nakon toga pridružuje `onopen`, `onmessage` i `onerror` funkcije `WebSocket` objekta funkcijama `onOpen()`, `onMessage()` i `onError()` koje su ugrađene u kod. U nastavku, funkcija `onOpen()` će biti automatski pozvana u svakoj situaciji kada je `WebSocket` konekcija (`WebSocket connection`) otvorena, metoda `onMessage()` će biti pozvana uvek kada `WebSocket` poruka (`WebSocket message`) primljena od servera i, na samo kraju, `onError()` metoda će biti pozvana svaki puta kada dođe do pojavljivanja `WebSocket` greške (`WebSocket error`).

Na klijent strani posmatrane *Echo aplikacije* vrši se jednostavno ažuriranje `<div>` taga, pogledati nastavak priloženog koda, primenom izraza `id="output"` porukom primljenom sa servera. Navedeno je obavljeno pomoću `writeToScreen()` funkcije koja je pozvana funkcijom `onMessage()` svaki puta kada se `WebSocket` poruka primi od servera.

Navedeni primer je odličan način da studenti steknu predstavu o načinima pisanja vlastitih aplikacija koje koriste *Java API za WebSocket*. Oslanjajući se na demonstrirani primer, u nastavku izlaganja će biti demonstriran postupak razvoja vlastite `WebSocket` Java EE aplikacije.

▼ Poglavlje 3

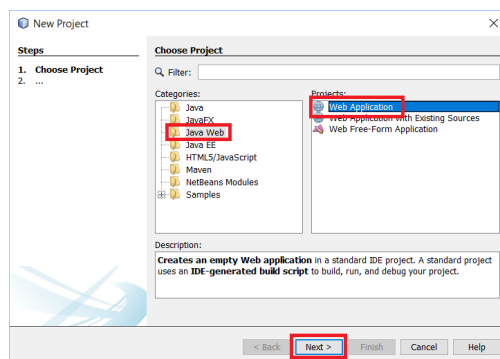
Kreiranje vlastite WebSocket aplikacije

KREIRANJE WEBSOCKET PROJEKTA

U prvom koraku biće kreiran projekat WebSocket Java EE aplikacije.

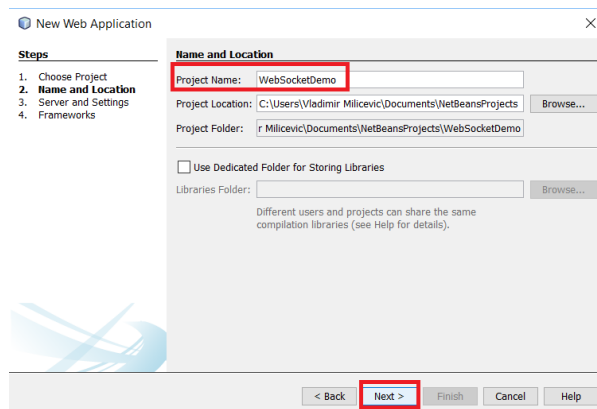
U prethodnom izlaganju je urađena detaljna analiza urađenog projekta koji koristi *Java API za WebSocket* i koji bi trebalo da posluži kao odlična osnova za kreiranje vlastitih *WebSocket* aplikacija. U nastavku sva analiza i diskusija će biti usmerene ka kreiranju Java EE *WebSocket* aplikacije koja sadrži *krajnju tačku WebSocket servera* (*WebSocket server endpoint*) i koja će izvršiti popunjavanje konkretne forme odgovarajućim podacima.

Za kreiranje *WebSocket* aplikacije neophodno je pokrenuti razvojno okruženje *NetBeans IDE* i izabrati opciju *File | New Project*. U nastavku, u otvorenom prozoru *New Project* neophodno je iz liste kategorija aplikacija izabrati opciju *Java Web*. Za izabranu kategoriju aplikacija, u nastavku, neophodno je izabrati tip aplikacije (projekta) *Web Application*. Navedeno je prikazano sledećom slikom.



Slika 3.1 Kreiranje WebSocket projekta [izvor: autor]

U nastavku je neophodno zadati naziv aplikaciji, odnosno projektu. U konkretnom slučaju će to biti *WebSocketDemo*. Sledeća slika ilustruje navedeni korak.



Slika 3.2 Dodeljivanje naziva projektu [izvor: autor]

Ostala su još samo dva koraka do kreiranja inicijalne strukture *WebSocket projekta*. Klikom na *Next* (prethodna slika) biće otvoren prozor za izbor aplikativnog servera. Ovde će biti prihvaćen *GlassFish* kao aplikativni server za ovu aplikaciju. I na ovom prozoru, neophodno je kliknuti na dugme *Next*.

Konačno, otvara se poslednji prozor čarobnjaka u kojem je neophodno, na dobro poznati način, izabrati *JSF 2.2* okvir (*framework*) koji će biti osnov za kreiranje JSF stranica koje će biti sastavni deo ovog projekta. Klikom na *Finish* inicijalna struktura *WebSocket* projekta je kreirana.

RAZVOJ KORISNIČKOG INTERFEJSA ZA WEBSOCKET PROJEKAT

U prvom koraku razvoja WebSocket projekta biće kreiran GUI aplikacije.

Pošto je kreirana inicijalna struktura *WebSocket* projekta neophodno je pristupiti kodiranu odgovarajućih datoteka aplikacije. Na samom početku će biti kreiran grafički korisnički interfejs *WebSocket* projekta. Fokus je na početnoj JSF stranici *index.xhtml*. Stranica ima sledeći inicijalni kod:

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://xmlns.jcp.org/jsf/html">
    <h:head>
        <title>Facelet Title</title>
    </h:head>
    <h:body>
        Hello from Facelets
    </h:body>
</html>
```

Sada će ovaj kod biti modifikovan na sledeći način da postane **HTML5 - friendly**. Korekcije se odnose na dodavanjem korekcija specifičnih **JSF** tagova: `<head jsf:id="head">` i `<body jsf:id="body">`. Sledi korišćeni listing:

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:jsf="http://xmlns.jcp.org/jsf">
    <head jsf:id="head">
        <title>Facelet Title</title>
    </head>
    <body jsf:id="body">
        Hello from Facelets
    </body>
</html>
```

Glava izmena je učinjena na polju angažovanja prostora imena (**name spaces**). Zamenjen je prostor imena `xmlns:h=http://xmlns.jcp.org/jsf/html` prostorom `xhmlns:jsf=http://xmlns.jsp.org/jsf`. Prethodni prostor imena se odnosi na specifične JSF tagove koji neće biti korišćeni u ovoj aplikaciji, a novi prostor imena podržava primenu specifičnih JSF atributa koji će aktivno biti korišćeni u ovoj **WebSocket** aplikaciji. U navedenom svetlu, zamenjeni su specifični JSF `<h:head>` i `<h:body>` tagovi odgovarajućim HTML tagovima u kojima je, potom, izvršeno dodavanje specifičnih JSF atributa `jsf:id`.

U nastavku, na početnu stranicu `index.xhtml`, biće dodata forma sa nekoliko jednostavnih kontrola za unos teksta. Kasnije, ova polja će biti popunjena podrazumevanim tekstom uz korišćenje **Java API za WebSocket**. Sledi konačan listing datoteke `index.xhtml`.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:jsf="http://xmlns.jcp.org/jsf">
    <head jsf:id="head">
        <title>WebSocket i Java EE</title>
    </head>
    <body jsf:id="body">
        <form method="POST" jsf:prependId="false">
            <table>
                <tr>
                    <td>Ime</td>
                    <td>
                        <input type="text" jsf:id="ime"
                            jsf:value="#{osoba.ime}"/>
                    </td>
                </tr>
                <tr>
                    <td>Prezime</td>
                    <td>
                        <input type="text" jsf:id="prezime"
                            jsf:value="#{osoba.prezime}"/>
                    </td>
                </tr>
            </table>
        </form>
    </body>
</html>
```

```
        </td>
      </tr>
      <tr>
        <td></td>
        <td>
          <input type="submit" value="Posalji"
            jsf:action="potvrda"/>
        </td>
      </tr>
    </table>
  </form>
</body>
</html>
```

WEBSOCKET PROJEKAT - KREIRANJE CDI ZRNA

U nastavku je neophodno kreirati odgovarajuće Java klase.

Ako se pogleda kod priložene JSF stranice *index.xhtml*, a posebno polja odgovarajuće forme sa stranice, moguće je primetiti da su polja direktno povezana sa osobinama izvesnog CDI (**Context and Dependency Injection**) zrna. U konkretnom slučaju zrna je predstavljeno klasom *Osoba.java*, a njemu, u kodu stranice, odgovara instanca *osoba*.

Kreiranjem klase CDI zrna i JSF stranice *index.xhtml* dobijena je jednostavna JSF *HTML 5 - friendly* aplikacija. ovu aplikaciju je, u nastavku, neophodno dodatno modifikovati kako bi bile iskorišćene sve prednosti primene *Java API za WebSocket*.

U nastavku sledi listing CDI klase *Osoba.java*. Listing sadrži sve privatne osobine klase zajedno sa odgovarajućim **setter** i **getter** metodama.

Klasa je kreirana kao standardna Java klasa kojoj je dodeljen navedeni naziv i odgovarajući paket kojem pripada.

```
package com.metropolitan.websocketdemo;

import javax.enterprise.context.RequestScoped;
import javax.inject.Named;

/**
 *
 * @author Vladimir Milicevic
 */
@Named
@RequestScoped
public class Osoba {
    private String ime;
    private String prezime;

    public String getIme() {
        return ime;
    }
}
```

```
}

public void setIme(String ime) {
    this.ime = ime;
}

public String getPrezime() {
    return prezime;
}

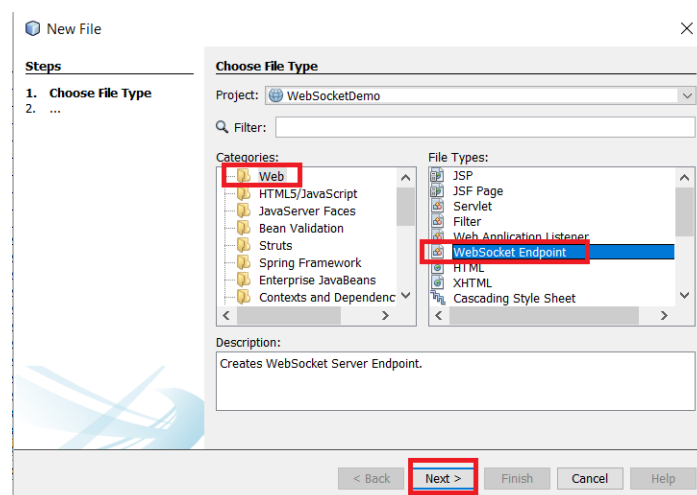
public void setPrezime(String prezime) {
    this.prezime = prezime;
}

}
```

RAZVOJ KRAJNJE TAČKE WEBSOCKET SERVERA

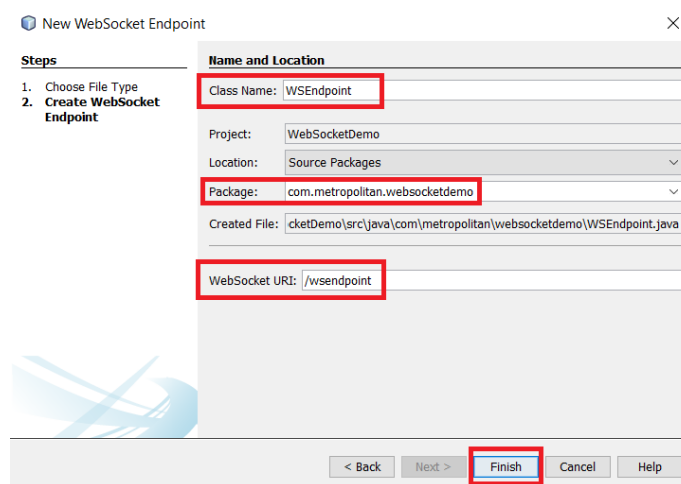
U nastavku je moguće pozabaviti se razvojem koda krajnje tačke WebSocket servera.

Kada je prethodno kreirani kod zauzeo svoje mesto u projektu *WebSocket* aplikacije, moguće je pristupiti kreiranju datoteke *krajnje tačke WebSocket servera*. U razvojnom okruženju NetBeans IDE, bira se opcija, za aktuelni projekat, *File | New File*. Nakon toga se iz kategorije *Web* bira tip datoteke *WebSocket Endpoint*. Navedeno je ilustrovano sledećom slikom.



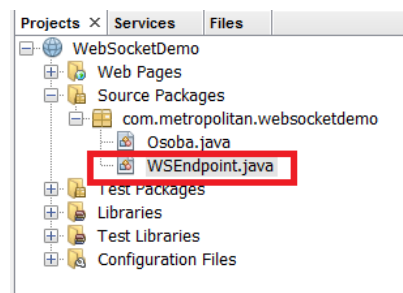
Slika 3.3 Kreiranje datoteke krajnje tačke WebSocket servera [izvor: autor]

U nastavku, klikom na dugme *Next* prozora *New File*, sa prethodne slike, otvara se nov prozor u kojem je neophodno zadati naziv datoteke i paket kojem će datoteka pripadati. Datoteka će dobiti naziv *WSEndpoint.java* i pripadaće paketu *com.metropolitan.websocketdemo*. Navedeno je ilustrovano sledećom slikom.



Slika 3.4 Definisanje naziva datotele i paketa [izvor: autor]

Klikom na *Finish* datoteka je zauzela svoje mesto u *WebSocket* projektu na način prikazan sledećom slikom.



Slika 3.5 Klasa krajnje tačke WebSocket servera u projektu [izvor: autor]

KOD KLASE KRAJNJE TAČKE WEBSOCKET SERVERA

Nakon generisanja i podešavanja datoteke sledi modifikacija inicijalnog koda.

Kao što je bilo moguće primetiti, u prethodnom izlaganju je kreirana klasa *krajnje tačke WebSocket servera*, izvršeno je podešavanje njenog naziva i njena dodela odgovarajućem paketu. Klikom na dugme *Finish* (videti Sliku 4) izvršeno je generisanje datoteke ove klase koja je dobila izvesni inicijalni kod. Početni kod datoteke *WSEndpoint.java* (klase krajnje tačke WebSocket servera) priložen je sledećim listingom.

```
package com.metropolitan.websocketdemo;

import javax.websocket.OnMessage;
import javax.websocket.server.ServerEndpoint;

/**
 *
 * @author Vladimir Milicevic
 */
```



```
@ServerEndpoint("/wsendpoint")
public class WSEndpoint {

    @OnMessage
    public String onMessage(String message) {
        return null;
    }

}
```

Ono što je moguće odmah primetiti, iz priloženog listinga, da se vrednost obuhvaćena anotacijom **@ServerEndpoint** podudara sa vrednošću koja je uneta u prozoru prikazanim Slikom 4 za *WebSocket URI*.

Takođe, generisana je i početna verzija metode **onMessage()**, obeležene anotacijom **@OnMessage**, koja u narednim koracima mora da bude modifikovana da bi mogla da obavlja zadatke koji se od nje očekuju. Metoda će biti modifikovana tako da vraća **JSON String** koji će biti parsiran na klijentovoj strani. Deo koda klase *krajnje tačke WebSocket servera* dobija sledeći oblik nakon navedenih modifikacija.

```
@OnMessage
public String onMessage(String message) {
    String retVal;
    if (message.equals("get_defaults")) {
        retVal = new StringBuilder("{").
            append("\ime\":"Auto",").
            append("\prezime\":"Generated").
            append("}").toString();
    } else {
        retVal = "";
    }
    return retVal;
}
```

DODATNO RAZMATRANJE

Neophodno je izvesti dodatnu analizu nad kodom klase krajnje tačke WebSocket servera.

Nakon izvršenih modifikacija, neophodno je još jednom sagledati u celini kreiranu klasu *krajnje tačke WebSocket servera*. Sledi kod u celosti:

```
package com.metropolitan.websocketdemo;

import javax.websocket.OnMessage;
import javax.websocket.server.ServerEndpoint;

/**
 *
```

```
* @author Vladimir Milicevic
*/
@ServerEndpoint("/wsendpoint")
public class WSEndpoint {

    @OnMessage
    public String onMessage(String message) {
        String retVal;
        if (message.equals("get_defaults")) {
            retVal = new StringBuilder("{").
                append("\"ime\": \"Auto\",").
                append("\"prezime\": \"Generated\").
                append("}").toString();
        } else {
            retVal = "";
        }
        return retVal;
    }
}
```

Parametar *message*, metode *onMessage()*, predstavlja vrednost poruke koja je poslata sa klijentove strane. U konkretnom slučaju metoda prima *get_defaults String* od klijenta, generiše *JSON String* sa podrazumevanim vrednostima koje će biti upotrebljene da popune formu sa stranice *index.xhtml*.

U nastavku *JSON String* će morati da bude parsiran *JavaScript* kodom na klijentovoj strani. Upravo će ovaj kod biti u fokusu u narednom izlaganju lekcije.

U konkretnom primeru JSON String je kreiran ručno. Obradeno je kako može biti generisan pomoću Java API za JSON-P u prethodnim lekcijama.

Poruke poslate sa klijentove strane obično imaju oblik JSON String - a.

KREIRANJE WEBSOCKET FUNKCIONALNOSTI NA KLIJENTU

Neophodno je dodati klijent kod koji će interagovati sa krajnjom tačkom WebSocket servera.

Po uzoru na ugrađeni demo primer razvojnog okruženja *NetBeans IDE* moguće je realizovati klijent stranu aktuelnog primera tako što će joj biti pridružen klijent kod koji će interagovati sa krajnjom tačkom *WebSocket* servera. I ovde može biti upotrebljen *JavaScript* kod. Sledi prošireni kod datoteke *index.xhtml*.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/
TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:jsf="http://xmlns.jcp.org/jsf">
  <head jsf:id="head">
    <title>WebSocket i Java EE</title>
    <script language="javascript" type="text/javascript">
      var wsUri = getRootUri() + "/WebSocketDemo/wsendpoint";

      function getRootUri() {
        return "ws://" + (document.location.hostname == "" ? "localhost" :
document.location.hostname) + ":" +
          (document.location.port == "" ? "8080" :
document.location.port);
      }

      function init() {
        websocket = new WebSocket(wsUri);
        websocket.onopen = function (evt) {
          onOpen(evt)
        };
        websocket.onmessage = function (evt) {
          onMessage(evt)
        };
        websocket.onerror = function (evt) {
          onError(evt)
        };
      }

      function onOpen(evt) {
        console.log("CONNECTED");
      }

      function onMessage(evt) {
        console.log("RECEIVED: " + evt.data);

        var json = JSON.parse(evt.data);

        document.getElementById('ime').value=json.ime;
        document.getElementById('prezime').value=json.prezime;
      }

      function onError(evt) {
        console.log('ERROR: ' + evt.data);
      }

      function doSend(message) {
        console.log("SENT: " + message);
        websocket.send(message);
      }

      window.addEventListener("load", init, false);
```

```
</script>

</head>
<body jsf:id="body">
  <form method="POST" jsf:prependId="false">
    <input type="button" value="Get Defaults"
onclick="doSend('get_defaults')"/>
    <table>
      <tr>
        <td>Ime</td>
        <td>
          <input type="text" jsf:id="ime"
            jsf:value="#{osoba.ime}"/>
        </td>
      </tr>
      <tr>
        <td>Prezime</td>
        <td>
          <input type="text" jsf:id="prezime"
            jsf:value="#{osoba.prezime}"/>
        </td>
      </tr>
      <tr>
        <td></td>
        <td><input type="submit" value="Potvrđi"
jsf:action="potvrda"/></td>
      </tr>
    </table>
  </form>
</body>
</html>
```

Kao što je već pomenuto, *JavaScript* kod u ovom primeru bazira se na primeru *Echo* koji demonstriran u prethodnom izlaganju i predstavlja sastavni deo *NetBeans IDE* prateće dokumentacije. U ovom konkretnom primeru su učinjene brojne izmene. Prava korekcija se odnosi na promenu vrednosti varijable *wsUri* koja sada predstavlja *URI* kreirane krajnje tačke *WebSocket* servera. *URI* krajnje tačke *WebSocket* servera koja je kreirana u ovom poslednjem primeru uvek će se sastojati od korena konteksta (*context root*) aplikacije praćenje vrednošću atributa anotacije *@ServerEndpoint* (u konkretnom slučaju */wsendpoint*).

Metoda *onMessage()*, koja je kreirana u ovom primeru, parsira *JSON String* koji je prosleđen sa krajnje tačke *WebSocket* servera i popunjava kreiranu formu odgovarajućim vrednostima.

U nastavku, na formu je dodato i dugme *Get Defaults* koje poziva *doSend()* metodu kreiranu u okviru *JavaScript* koda, prosleđujući *get_defaults String* kao parametar. Metoda *doSend()*, potom, prosleđuje *String* ka krajnjoj tački *WebSocket* servera pomoću metode *send()* *JavaScript WebSocket* objekta. Krajnja tačka *WebSocket* servera vraća *JSON String* sa podrazumevanim vrednostima kada dobije pomenuti *String*.

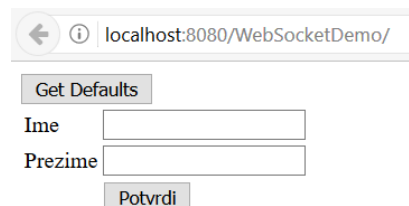
PODRAZUMEVANI PODACI

Neophodno je pokrenuti aplikaciju i testirati kodirane datoteke.

Pre nego što se pokrene aplikacija, u cilju njenog testiranja, neophodno je primetiti da na formi početne stranice postoji dugme *Potvrdi*. Klikom na ovo dugme vrši se slanje podataka, u vezi sa objektom *osoba*, u JSF stranicu *potvrda.xhtml*, da bi krajnji korisnik mogao da pregleda navedene rezultate procesiranja u *WebSocket* aplikaciji. Sledećim listingom je priložen kod *JSF* stranice sadržan u datoteci *potvrda.xhtml*:

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/
TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:jsf="http://xmlns.jcp.org/jsf">
  <head jsf:id="head">
    <title>Potvrda</title>
  </head>
  <body jsf:id="body">
    #{osoba.ime} #{osoba.prezime}
  </body>
</html>
```

Nakon prevođenja i pokretanja kreirane aplikacije u veb pregledaču se otvara početna stranica kao na sledećoj slici.



localhost:8080/WebSocketDemo/

Get Defaults

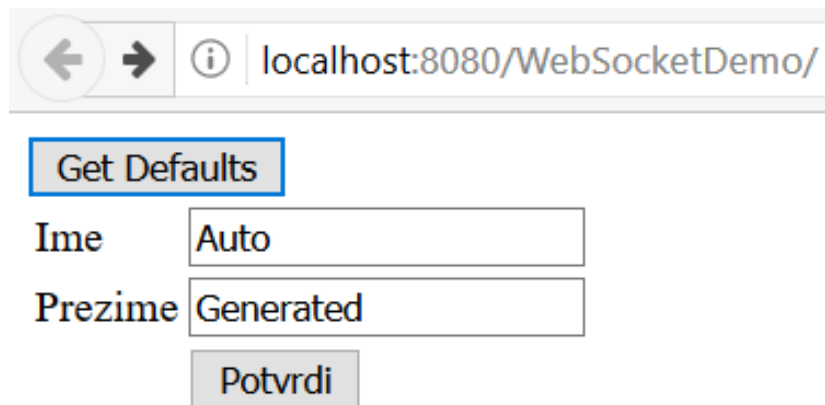
Ime

Prezime

Potvrdi

Slika 3.6 Početna stranica sa formom [izvor: autor]

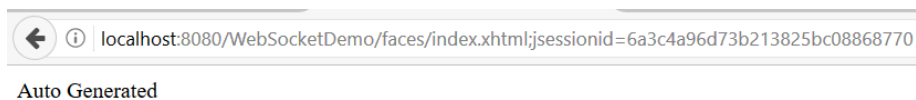
Klikom na dugme *Get Defaults* iz krajnje tačke *WebSocket* servera dobijaju se podrazumevane vrednosti za polja *ime* i *prezime*. Navedeno je prikazano sledećom slikom.



A screenshot of a web browser window. The address bar shows 'localhost:8080/WebSocketDemo/'. Below the address bar is a 'Get Defaults' button. Underneath this button are two text input fields: 'Ime' with the value 'Auto' and 'Prezime' with the value 'Generated'. At the bottom of the form is a 'Potvrdi' button.

Slika 3.7 Forma popunjena podrazumevanim vrednostima [izvor: autor]

U nastavku neophodno je testirati slučajeve kada korisnik popunjava sadržaj forme i to će biti obavljeno u narednom izlaganju. Klikom na dugme *Potvrđi* ovi podaci mogu biti prosleđeni na stranicu za prikazivanje podataka *potvrda.xhtml*.



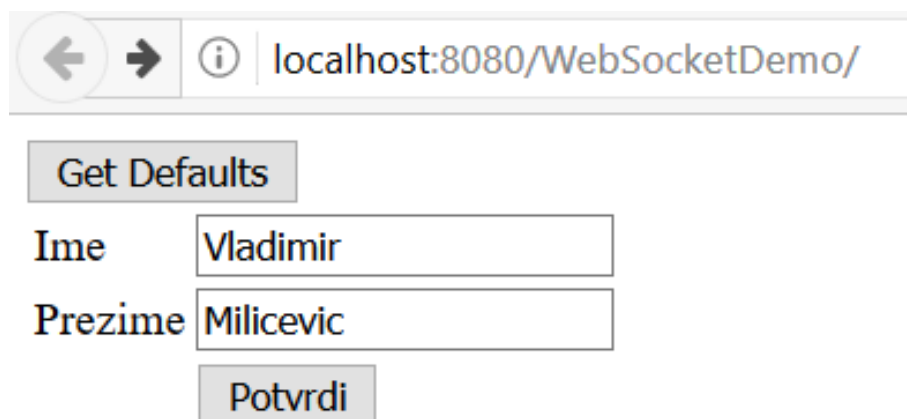
A screenshot of a web browser window. The address bar shows 'localhost:8080/WebSocketDemo/faces/index.xhtml;jsessionid=6a3c4a96d73b213825bc08868770'. Below the address bar, the text 'Auto Generated' is displayed.

Slika 3.8 Podrazumevani podaci na stranici za prikazivanje unetih podataka [izvor: autor]

KORISNIČKI PODACI

Neophodno je pokrenuti aplikaciju i testirati kodirane datoteke za vrednosti koje je uneo korisnik.

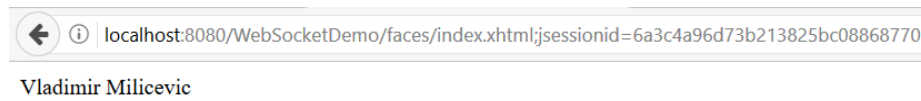
Kao što je istaknuto u prethodnom izlaganju, u nastavku je neophodno testirati slučajeve kada korisnik popunjava sadržaj forme. Na početnoj stranici *index.xhtml*, korisnik unosi podatke u polja za unos teksta koja su u direktnoj vezi sa osobinama *ime* i *prezime* objekta klase *Osoba.java*. Sledećom slikom je prikazan unos korisničkih podataka u formu početne stranice.



A screenshot of a web browser window. The address bar shows 'localhost:8080/WebSocketDemo/'. Below the address bar is a 'Get Defaults' button. Underneath this button are two text input fields: 'Ime' with the value 'Vladimir' and 'Prezime' with the value 'Milicevic'. At the bottom of the form is a 'Potvrđi' button.

Slika 3.9 Unos korisnikovih podataka na formu početne stranice [izvor: autor]

Klikom na dugme potvrdi podaci se šalju na prikazivanje u stranicu *potvrda.xhtml* na način obrađen u prethodnim izlaganjima.



Slika 3.10 Prikaz korisničkih podataka [izvor: autor]

Nakon obavljenog testiranja moguće je zaključiti da je celokupan posao oko razvoja vlastite *WebSocket* aplikacije urađen na pravi način.

▼ Poglavlje 4

Studija slučaja za Java EE, WebSocket, JS i HTML5

DEFINISANJE ZAHTEVA

Definisanje zahteva po kojima će funkcionisati demo aplikacija je prvi zadatak.

Vodeći se teorijskim izlaganjem iz prethodnog dela lekcija, kao i demonstriranim primerima, u ovom delu izlaganja će prvo biti istaknute smernice u okviru kojih će teći analiza i diskusija konkretnog Java EE projekta koji koristi *WebSocket API*. Izlaganje će se bazirati na sledećim tezama:

- Kreiranje Java EE 7 aplikacije koja koristi *WebSocket API*;
- Primena metoda *OnOpen()* i *OnMessage()* *WebSocket* životnog ciklusa za izvođenje određenih akcija u Java EE 7 aplikaciji;
- Definisanje klijent strane *WebSocket krajnje tačke* primenom *JavaScript* koda;
- Kreiranje *POJO* (*Plain Old Java Objects*), u realnom vremenu, sa akcijama koje se pozivaju iz veb pregledača.

U lekciji je već pomenuto da standardni *HTTP* pristup komunikaciji između klijenta i servera ima izvesne nedostatke koji impliciraju primenu drugačijeg pristupa njihovom komuniciranju. *WebSocket* je upravo taj alternativni pristup klijent - server komunikaciji koji omogućava dvosmernu, *full-duplex* klijent - server komunikaciju u realnom vremenu. Serve može da šalje podatke klijentu u bilo kojem trenutku.

SCENARIO: Kreira se Java aplikacija pod nazivom *WebSocket Home*, kontrolna aplikacija "pametne kuće" podignuta nad *Java EE 7* platformom. Aplikacija koristi korisnički interfejs za kontrolisanje uređaja u domaćinstvu iz veb pregledača. Sva ažuriranja događaja za klijente prikazane na *WebSocket Home* server dešavaju se u realnom vremenu.

SISTEMSKI ZAHTEVI: Za realizaciju *WebSocket Home* aplikacije neophodni su sledeći zahtevi:

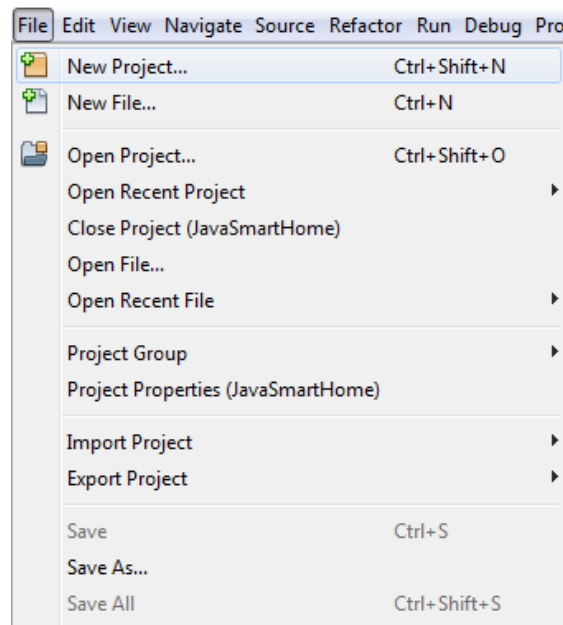
- Poznavanje Java programskog jezika;
- Osnovno znanje platforme *Java EE 7*;
- Osnovno znanje *HTML 5*, *JavaScript* i *CSS*.

Java API za WebSocket (JSR-356) pojednostavljuje integraciju WebSocket - a u Java EE 7 aplikaciju.

JAVA EE 7 PROJEKAT WEBSOCKETHOME

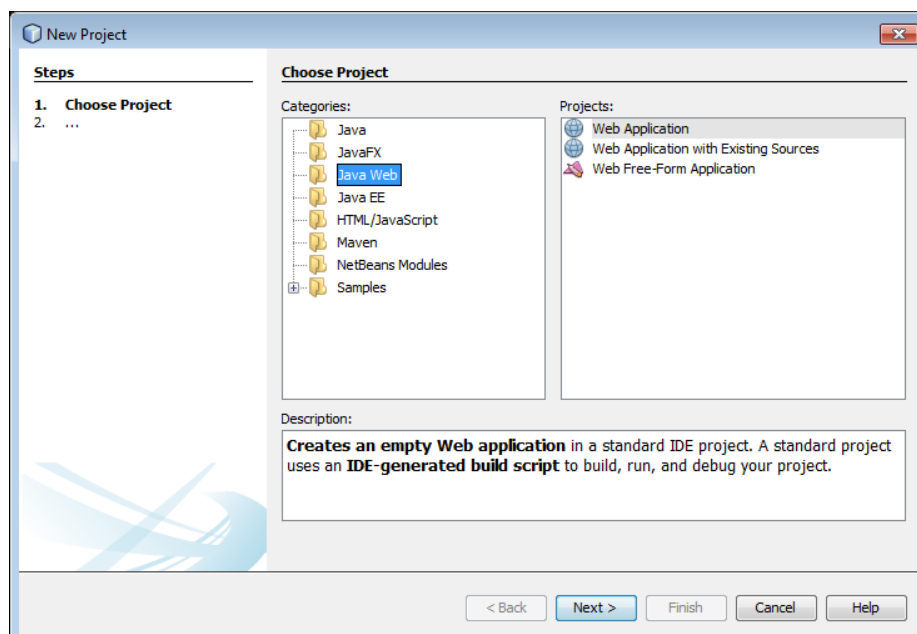
U prvom koraku će biti kreiran konkretna Java EE veb projekat.

Za kreiranje *WebSocket* projekta "*pametne kuće*" neophodno je pokrenuti razvojno okruženje NetBeans IDE i u odgovarajućem meniju izabrati opciju *New Project*. Navedeno je prikazano sledećom slikom.



Slika 4.1 Kreiranje novog NetBeans projekta [izvor: autor]

Nakon obavljene akcije otvoriće se *New Project* prozor iz kojeg je neophodno izabrati kategoriju projekta *Java Web*, odnosno tip aplikacije koja se kreira trebalo bi da bude *Web Application*. Navedeno je prikazano sledećom slikom.



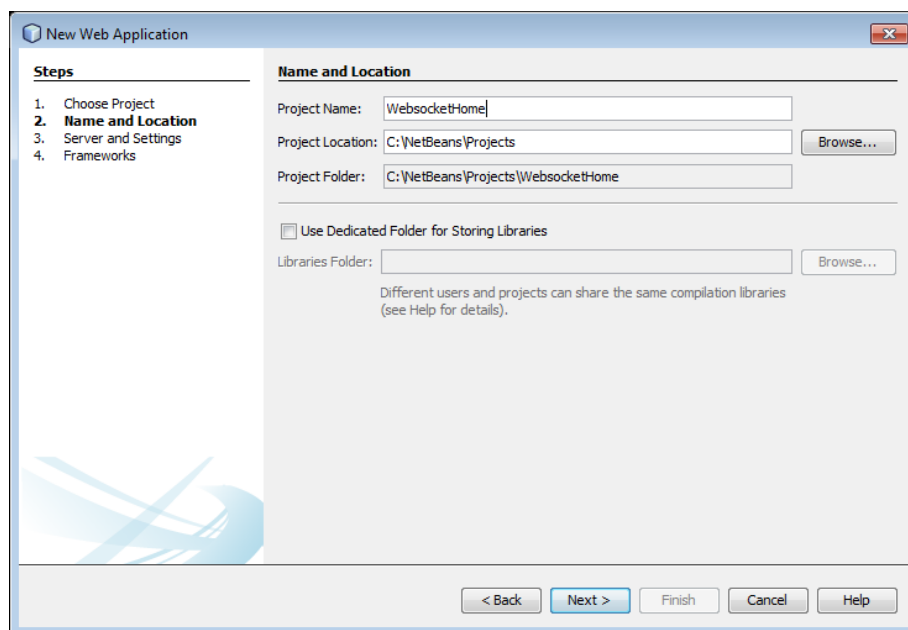
Slika 4.2 Kreiranje Java EE veb aplikacije [izvor: autor]

Klikom na dugme *Next* na prethodnom prozoru započinje inicijalno podešavanje Java veb projekta koji se upravo kreira.

INICIJALNA PODEŠAVANJA

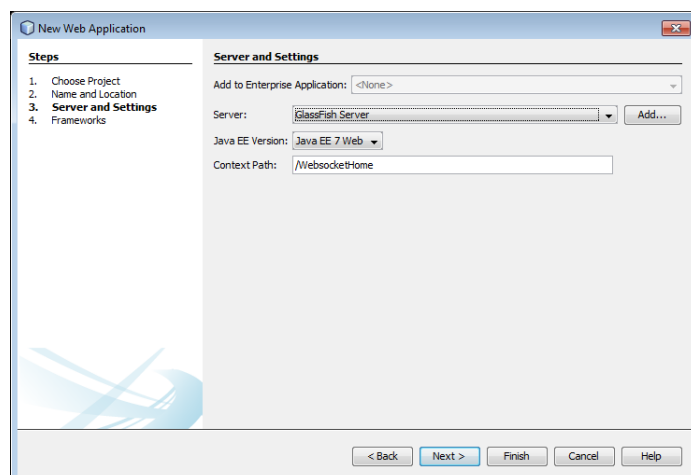
Za projekat koji se kreira razvojnim okruženjem NetBeans IDE obavljaju se početna podešavanja.

Kao što je istaknuto u prethodnom izlaganju, klikom na dugme *Next* na prethodnom prozoru započinje inicijalno podešavanje *Java veb* projekta koji se upravo kreira, a naziv mu je *WebSocketHome*. U sledećem prozoru, neophodno je dodati pomenuti naziv projektu. Navedeno je prikazano sledećom slikom.



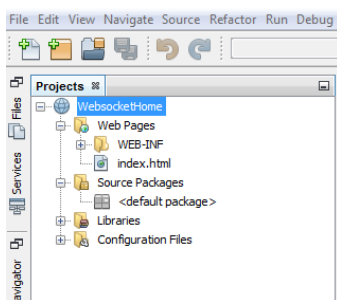
Slika 4.3 Podešavanje naziva WebSocket projekta [izvor: autor]

Nakon obavljenog zadatka dodele naziva projektu, neophodno je pristupiti dodatnim podešavanjima za izbor servera i Java EE verzije (sledeća slika).



Slika 4.4 Izbor servera i Java verzije [izvor: autor]

Konačno klikom na dugme Finish, projekat sa inicijalnim podešavanjima je kreiran, a to je moguće sagledati sa sledeće slike.

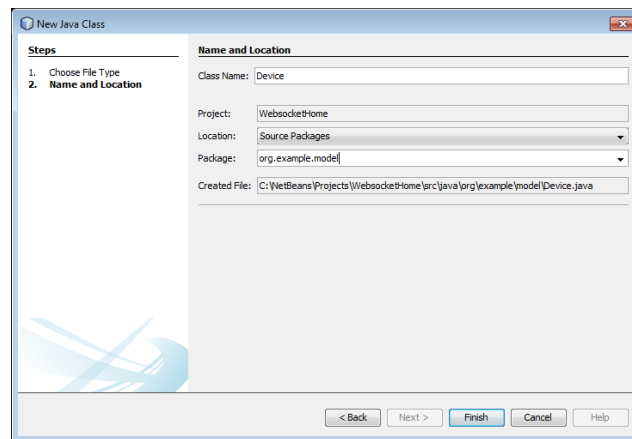


Slika 4.5 Početna struktura WebSocketHome projekta [izvor: autor]

KREIRANJE KLASA MODELA DEVICE.JAVA

U nastavku kreira se klasa koja određuje pojedinačne uređaje u domaćinstvu.

Nakon što je kreiran projekat, sa osnovnim podešavanjima, neophodno je izvršiti dodavanje odgovarajućih programskih datoteka u skladu sa definisanim zahtevima aplikacije. U prvom koraku će biti kreirana modelska Java klasa pod nazivom *Device.java*. Klasa će morati da sadrži sve neophodne osobine uređaja kao i odgovarajuće *setter* i *getter* metode. Kreiranje datoteke započinje u razvojnom okruženju izborom opcije, za aktuelni projekat, *File > New File*. Potom, na dobro poznat način iz kategorije *Java*, neophodno je izabrati tip datoteke *Java class*. Po obavljenom zadatku, otvara se prozor u kojem je neophodno dodeliti naziv klasi koja se kreira, *Device.java* u konkretnom slučaju, i naziv paketa kojem će klasa pripadati, na primer *org.example.model* u konkretnom slučaju. Poslednje navedeno je ilustrovano sledećom slikom.



Slika 4.6 Podešavanje naziva i paketa klase modela [izvor: autor]

Klikom na *Finish* datoteka zauzima svoje mesto u strukturi projekta. Sledeće što je važno jeste dodeliti kreiranoj klasi osobine i njima odgovarajuće *setter* i *getter* metode. Sledećim listingom priložene je kompletirana klasa *Device.java*.

```
package org.example.model;

public class Device {

    private int id;
    private String name;
    private String status;
    private String type;
    private String description;

    public Device() {
    }

    public int getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    public String getStatus() {
        return status;
    }

    public String getType() {
        return type;
    }

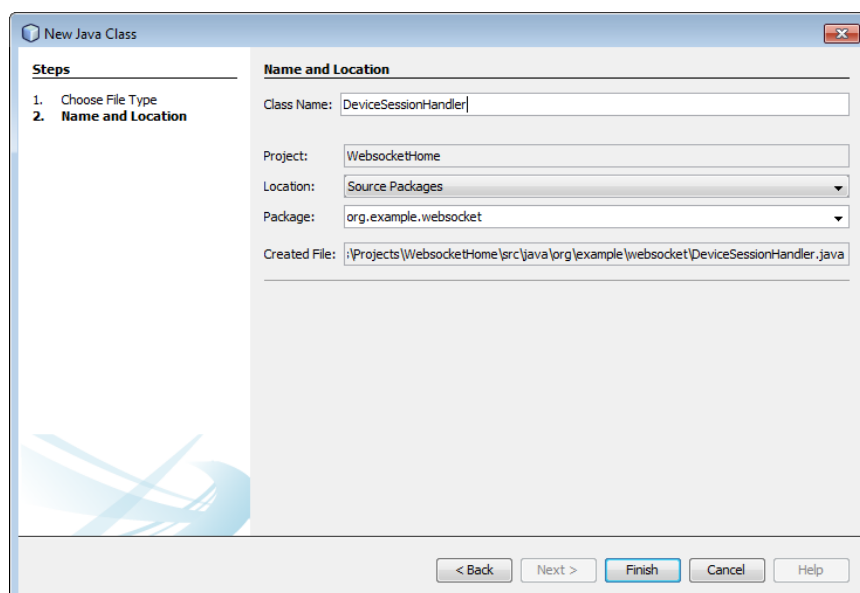
    public String getDescription() {
        return description;
    }
}
```

```
public void setId(int id) {  
    this.id = id;  
}  
  
public void setName(String name) {  
    this.name = name;  
}  
  
public void setStatus(String status) {  
    this.status = status;  
}  
  
public void setType(String type) {  
    this.type = type;  
}  
  
public void setDescription(String description) {  
    this.description = description;  
}  
}
```

KREIRANJE KLASJE ZA UPRAVLJANJE SESIJOM

Nakon kreiranja modela sledi klasa za upravljanje sesijom.

Nakon kreiranja modela sledi klasa za upravljanje sesijom. Kreiranje datoteke započinje u razvojnom okruženju izborom opcije, za aktuelni projekat, *File > New File*. Potom, na dobro poznat način iz kategorije *Java*, neophodno je izabrati tip datoteke *Java class*. Po obavljenom zadatku, otvara se prozor u kojem je neophodno dodeliti naziv klasi koja se kreira, *DeviceSessionHandler.java* u konkretnom slučaju, i naziv paketa kojem će klasa pripadati, na primer *org.example.websocket* u konkretnom slučaju. Poslednje navedeno je ilustrovano sledećom slikom.



Slika 4.7 Klasa za upravljanje sesijom [izvor: autor]

Za početak, klasa će biti obeležena anotacijom **@ApplicationScoped** koju će pratiti odgovarajuća **import** instrukcija (sledeći listing).

```
package org.example.websocket;

import javax.enterprise.context.ApplicationScoped;

@ApplicationScoped
public class DeviceSessionHandler {

}
```

U kod klase za upravljanje sesijom će, potom, biti ugrađene dve **HashSet** instance za čuvanje dodatih uređaja i aktivnih sesija. Navedeno je ilustrovano sledećim izolovanim listingom.

```
package org.example.websocket;

import javax.enterprise.context.ApplicationScoped;
import java.util.HashSet;
import java.util.Set;
import javax.websocket.Session;
import org.example.model.Device;

@ApplicationScoped
public class DeviceSessionHandler {
    private final Set<Session> sessions = new HashSet<>();
    private final Set<Device> devices = new HashSet<>();
}
```

Svakom klijentu konektovanom na aplikaciju odgovara vlastita sesija.

UKLOP VANJSKE FUNKCIONALNOSTI

U klasu za upravljanje sesijom neophodno je dodati dopunske funkcionalnosti.

U nastavku, u kreiranu klasu za upravljanje sesijom neophodno je dodati metode kojima je omogućeno dodavanje i uklanjanje sesija sa servera. Navedeno je prikazano sledećim izolovanim listingom klase za upravljanje sesijom.

```
package org.example.websocket;

...

@ApplicationScoped
public class DeviceSessionHandler {
```

```

...

public void addSession(Session session) {
    sessions.add(session);
}

public void removeSession(Session session) {
    sessions.remove(session);
}
}

```

Dalje, klasa za upravljanje sesijom mora da poseduje mogućnost rukovanja objektima uređaja. U tu svrhu je neophodno dodati nekoliko novih metoda:

- [addDevice\(\)](#) - dodaje novi uređaj u aplikaciji;
- [removeDevice\(\)](#) - uklanja postojeći uređaj iz aplikacije;
- [toggleDevice\(\)](#) - Isključuje status uređaja;
- [getDevices\(\)](#) - Vraća listu uređaja i njihovih atributa;
- [getDeviceById\(\)](#) - Vraća uređaj sa specifičnim identifikatorom;
- [createAddMessage\(\)](#) - Kreira JSON poruku za dodavanje uređaja u aplikaciju;
- [sendToSession\(\)](#) - Šalje poruku događaja klijentu;
- [sendToAllConnectedSessions\(\)](#) - Šalje poruku događaja svim konektovanim klijentima.

Sledi trenutni listing klase za upravljanje sesijom koji se odnosi na gore pomenute metode.

```

package org.example.websocket;

...
public class DeviceSessionHandler {

    ...

    public List<Device> getDevices() {
        return new ArrayList<>(devices);
    }

    public void addDevice(Device device) {
    }

    public void removeDevice(int id) {
    }

    public void toggleDevice(int id) {
    }

    private Device getDeviceById(int id) {
        return null;
    }

    private JsonObject createAddMessage(Device device) {

```

```
        return null;
    }

    private void sendToAllConnectedSessions(JsonObject message) {
    }

    private void sendToSession(Session session, JsonObject message) {
    }
}
```

KOMPLETIRANJE LOGIKE METODA

Kreirane metode za rukovanje uređajima neophodno je redefinisati.

U prethodnom izlaganju su priložene metode za upravljanje *Device* objektima. U nastavku je neophodno konkretizovati ove metode dodajući im konkretne linije koda za ispunjavanje tačno određene programske logike.

Za početak neophodno je dodati varijablu identifikatora uređaja kao na sledeći način:

```
...

public class DeviceSessionHandler {

    private int deviceId = 0;
    private final Set<Session> sessions = new HashSet<>();
    private final Set<Device> devices = new HashSet<>();

    ...

}
```

Dalje, u metodu *addSession()* neophodno je dodati petlju za slanje liste uređaja povezanog klijenta.

```
public void addSession(Session session) {
    sessions.add(session);
    for (Device device : devices) {
        JsonObject addMessage = createAddMessage(device);
        sendToSession(session, addMessage);
    }
}
```

U nastavku, implemetira se metoda za dodavanje uređaja koja kreira novi uređaj sa odgovarajućim identifikatorom i šalje *JSON* poruku svim sesijama ili aktivnim klijentima *WebSocket* servera.

```
public void addDevice(Device device) {
    device.setId(deviceId);
}
```



```

        devices.add(device);
        deviceId++;
        JsonObject addMessage = createAddMessage(device);
        sendToAllConnectedSessions(addMessage);
    }

```

Kao komplement metodi za dodavanje uređaja neophodno je definisati metodu za uklanjanje uređaja kreiranih prethodno prikazanom metodom. Sledi kod metode *removeDevice()*.

```

public void removeDevice(int id) {
    Device device = getDeviceById(id);
    if (device != null) {
        devices.remove(device);
        JsonProvider provider = JsonProvider.provider();
        JsonObject removeMessage = provider.createObjectBuilder()
            .add("action", "remove")
            .add("id", id)
            .build();
        sendToAllConnectedSessions(removeMessage);
    }
}

```

KLASA ZA UPRAVLJANJE SESIJOM

Preostale metode klase za upravljanje sesijom moraju biti dopunjene odgovarajućim kodom.

Metoda *toggleDevice()* je sledeća koja mora biti redefinisana. Ova metoda rukuje statusom uređaja i šalje poruku svim sesijama koje su još uvek aktivne u *WebSocket* serveru. Metoda je priložena sledećim listingom:

```

public void toggleDevice(int id) {
    JsonProvider provider = JsonProvider.provider();
    Device device = getDeviceById(id);
    if (device != null) {
        if ("On".equals(device.getStatus())) {
            device.setStatus("Off");
        } else {
            device.setStatus("On");
        }
        JsonObject updateDevMessage = provider.createObjectBuilder()
            .add("action", "toggle")
            .add("id", device.getId())
            .add("status", device.getStatus())
            .build();
        sendToAllConnectedSessions(updateDevMessage);
    }
}

```

Za kraj je neophodno pozabaviti se metodama `getDeviceById()`, `createAddMessage()`, `sendToAllConnectedSessions` i `sendToSession()`. Konačnim definisanjem ovih metoda, završava se kodiranje *klase za upravljanje sesijom*. Kod ovih metoda je priložen narednim listingom:

```
private Device getDeviceById(int id) {
    for (Device device : devices) {
        if (device.getId() == id) {
            return device;
        }
    }
    return null;
}

private JsonObject createAddMessage(Device device) {
    JsonProvider provider = JsonProvider.provider();
    JsonObject addMessage = provider.createObjectBuilder()
        .add("action", "add")
        .add("id", device.getId())
        .add("name", device.getName())
        .add("type", device.getType())
        .add("status", device.getStatus())
        .add("description", device.getDescription())
        .build();
    return addMessage;
}

private void sendToAllConnectedSessions(JsonObject message) {
    for (Session session : sessions) {
        sendToSession(session, message);
    }
}

private void sendToSession(Session session, JsonObject message) {
    try {
        session.getBasicRemote().sendText(message.toString());
    } catch (IOException ex) {
        sessions.remove(session);
    }
}

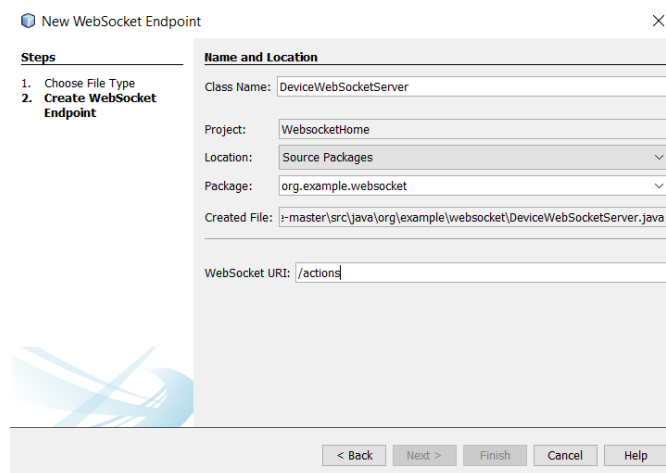
Logger.getLogger(DeviceSessionHandler.class.getName()).log(Level.SEVERE, null, ex);
}
```

KREIRANJE KRAJNJE TAČKE WEBSOCKET SERVERA

Nakom kreirane klase modela, kreira se datoteka krajnje tačke WebSocket servera.

U nastavku na red dolazi zadatak kreiranja krajnje tačke *WebSocket* servera. Kreiranje datoteke započinje u razvojnom okruženju izborom opcije, za aktuelni projekat, *File > New*

File. Potom, na dobro poznat način iz kategorije *Web*, neophodno je izabrati tip datoteke *WebSocket Endpoint*. Po obavljenom zadatku, otvara se prozor u kojem je neophodno dodeliti naziv klasi koja se kreira, *DeviceWebSocketServer.java* u konkretnom slučaju, i naziv paketa kojem će klasa pripadati, na primer *org.example.websocket* u konkretnom slučaju. Još jedno podešavanje je neophodno izvesti u ovom prozoru, a ono se odnosi na zadavanje vrednosti parametra anotacije *@ServerEndpoint* krajnje tačke *WebSocket* servera. Poslednje navedeno je ilustrovano sledećom slikom.



Slika 4.8 Definisanje naziva, paketa i URI krajnje tačke WebServera [izvor: autor]

Klikom na dugme *Finish*, kreirana je datoteka klase krajnje tačke *WebSocket* servera čiji je inicijalni kod priložen sledećim listingom.

```
import javax.websocket.OnClose;
import javax.websocket.OnError;
import javax.websocket.OnMessage;
import javax.websocket.OnOpen;
import javax.websocket.Session;
import javax.websocket.server.ServerEndpoint;

@ServerEndpoint("/actions")
public class DeviceWebSocketServer {

    @OnOpen
    public void open(Session session) {
    }

    @OnClose
    public void close(Session session) {
    }

    @OnError
    public void onError(Throwable error) {
    }

    @OnMessage
    public void handleMessage(String message, Session session) {
```

```
}
}
```

DODATNO KODIRANJE KRAJNJE TAČKE WEBSOCKET SERVERA

Neophodno je dodati dopunski kod u datoteku krajnje tačke WebSocket servera.

Prva korekcija koja će biti načinjena jeste dodavanje oblasti aplikacije klasi krajnje tačke *WebSocket* servera. To će biti učinjeno na način što će klasa, dodatno, biti obeležena anotacijom *@ApplicationScoped*, pri čemu je u kod klase neophodno dodati i odgovarajuću *import* instrukciju. Navedeno je ilustrovano sledećim izolovanim delom listinga klase krajnje tačke *WebSocket* servera.

```
...
import javax.websocket.OnOpen;
import javax.websocket.Session;
import javax.enterprise.context.ApplicationScoped;

@ApplicationScoped
@ServerEndpoint("/actions")
public class DeviceWebSocketServer {
    ...
}
```

U nastavku je neophodno izvršiti umetanje (*inject*) objekta prethodno kreirane *klase za upravljanje sesijom*. Ovim objektom biće obrađeni događaji koji odgovaraju fazama *WebSocket* životnog ciklusa.

```
package org.example.websocket;

...
import javax.websocket.server.ServerEndpoint;
import javax.inject.Inject;

@ApplicationScoped
@ServerEndpoint("/actions")
public class DeviceWebSocketServer {

    @Inject
    private DeviceSessionHandler sessionHandler;

    @OnOpen
    public void open(Session session) {
    }

    ...
}
```

Dalje, kodira se metoda **open()** koja prihvata objekat sesije i obeležena je anotacijom **@OnOpen**.

```
@OnOpen
public void open(Session session) {
    sessionHandler.addSession(session);
}
```

Metoda koja je obeležena anotacijom **@OnMessage** čita akcije i atribute uređaja poslate sa klijenta i poziva rukovaoca uređajima za izvođenje specifičnih operacija nad **Device** objektom. Kompletiran kod metode je priložen sledećim listingom. Dodate su i **import** instrukcije.

```
package org.example.websocket;

...
import java.io.StringReader;
import javax.json.Json;
import javax.json.JsonObject;
import javax.json.JsonReader;
import org.example.model.Device;
...

@OnMessage
public void handleMessage(String message, Session session) {

    try (JsonReader reader = Json.createReader(new StringReader(message))) {
        JsonObject jsonMessage = reader.readObject();

        if ("add".equals(jsonMessage.getString("action"))) {
            Device device = new Device();
            device.setName(jsonMessage.getString("name"));
            device.setDescription(jsonMessage.getString("description"));
            device.setType(jsonMessage.getString("type"));
            device.setStatus("Off");
            sessionHandler.addDevice(device);
        }

        if ("remove".equals(jsonMessage.getString("action"))) {
            int id = (int) jsonMessage.getInt("id");
            sessionHandler.removeDevice(id);
        }

        if ("toggle".equals(jsonMessage.getString("action"))) {
            int id = (int) jsonMessage.getInt("id");
            sessionHandler.toggleDevice(id);
        }
    }
}
```

METODE CLOSE() I ONERROR()

Neophodno je kodirati i preostale dve metode klase.

Na samom kraju izlaganja u vezi sa krajnjom tačkom *WebSocket* servera konkretnog primera, neophodno je dodati konkretan kod u metode *close()* i *onError()*. Metode su obeležene odgovarajućim anotacijama *@OnClose* i *@OnError*, respektivno. Takođe, u skladu sa novim kodom, priložene su i odgovarajuće *import* instrukcije. Sledi listing kojim je završeno definisanje klase krajnje tačke *WebSocket* servera.

```
package org.example.websocket;

...
import java.util.logging.Level;
import java.util.logging.Logger;

...

@OnClose
public void close(Session session) {
    sessionHandler.removeSession(session);
}

@OnError
public void onError(Throwable error) {
    Logger.getLogger(DeviceWebSocketServer.class.getName()).log(Level.SEVERE,
null, error);
}
```

KREIRANJE KORISNIČKOG INTERFEJSA

*Neophodno je redefinisati datoteku *index.html* za kvalitetno prikazivanje funkcionalnosti primera.*

Da bi primer bio kompletiran u skladu sa postavljenim zahtevima, neophodno je kreirati i programske komponente preko kojih korisnik može da komunicira sa aplikacijom. Prva od njih će biti veb stranica *index.html*. Ova stranica sadrži formu za unos podataka u vezi sa uređajima, a ima mogućnost i prikazivanja uređaja koji su aktivni u sistemu. Sledi listing ove stranice.

```
<!DOCTYPE html>
<html>
  <head>
    <title></title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <script src="websocket.js"></script>
    <link rel="stylesheet" type="text/css" href="style.css">
  </head>
```

```

<body>

    <div id="wrapper">
        <h1>Java WebSocket Home</h1>
        <p>Welcome to the Java WebSocket Home. Click the Add a device button to
start adding devices.</p>
        <br/>
        <div id="addDevice">
            <div class="button"> <a href="#" OnClick="showForm()">Add a
device</a> </div>
            <form id="addDeviceForm">
                <h3>Add a new device</h3>
                <span>Name: <input type="text" name="device_name"
id="device_name"></span>
                <span>Type:
                    <select id="device_type">
                        <option name="type" value="Appliance">Appliance</option>
                        <option name="type"
value="Electronics">Electronics</option>
                        <option name="type" value="Lights">Lights</option>
                        <option name="type" value="Other">Other</option>
                    </select></span>

                <span>Description:<br/>
                    <textarea name="description" id="device_description"
rows="2" cols="50"></textarea>
                </span>

                <input type="button" class="button" value="Add"
onclick=formSubmit()>
                <input type="reset" class="button" value="Cancel"
onclick=hideForm()>
            </form>
        </div>
        <br/>
        <h3>Currently connected devices:</h3>
        <div id="content">
        </div>
    </div>

</body>
</html>

```

Stranica je dekorisana primenom CSS stilova kreiranih u datoteci style.css koja je ilustrovana sledećim listingom.

```

body {
    font-family: Arial, Helvetica, sans-serif;
    font-size: 80%;
    background-color: #1f1f1f;
}

#wrapper {

```

```
width: 960px;
margin: auto;
text-align: left;
color: #d9d9d9;
}

p {
    text-align: left;
}

.button {
    display: inline;
    color: #fff;
    background-color: #f2791d;
    padding: 8px;
    margin: auto;
    border-radius: 8px;
    -moz-border-radius: 8px;
    -webkit-border-radius: 8px;
    box-shadow: none;
    border: none;
}

.button:hover {
    background-color: #ffb15e;
}

.button a, a:visited, a:hover, a:active {
    color: #fff;
    text-decoration: none;
}

#addDevice {
    text-align: center;
    width: 960px;
    margin: auto;
    margin-bottom: 10px;
}

#addDeviceForm {
    text-align: left;
    width: 400px;
    margin: auto;
    padding: 10px;
}

#addDeviceForm span {
    display: block;
}

#content {
    margin: auto;
    width: 960px;
}
```



```
.device {
    width: 180px;
    height: 110px;
    margin: 10px;
    padding: 16px;
    color: #fff;
    vertical-align: top;
    border-radius: 8px;
    -moz-border-radius: 8px;
    -webkit-border-radius: 8px;
    display: inline-block;
}

.device.off {
    background-color: #c8cccf;
}

.device span {
    display: block;
}

.deviceName {
    text-align: center;
    font-weight: bold;
    margin-bottom: 12px;
}

.removeDevice {
    margin-top: 12px;
    text-align: center;
}

.device.Appliance {
    background-color: #5eb85e;
}

.device.Appliance a:hover {
    color: #aled82;
}

.device.Electronics {
    background-color: #0f90d1;
}

.device.Electronics a:hover {
    color: #4badd1;
}

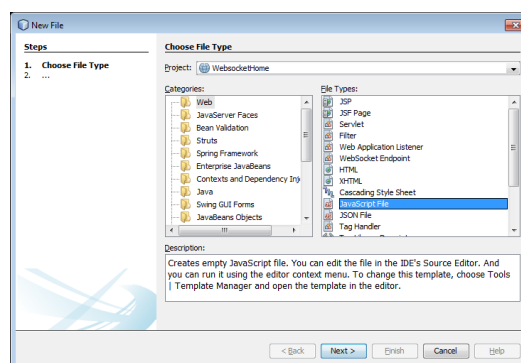
.device.Lights {
    background-color: #c2a00c;
}
```

```
.device.Lights a:hover {  
    color: #fad232;  
}  
  
.device.Other {  
    background-color: #db524d;  
}  
  
.device.Other a:hover {  
    color: #ff907d;  
}  
  
.device a {  
    text-decoration: none;  
}  
  
.device a:visited, a:active, a:hover {  
    color: #fff;  
}  
  
.device a:hover {  
    text-decoration: underline;  
}
```

KLIJENT KRAJNJE TAČKE WEBSOCKET SERVERA

Sledi poslednja datoteka primera.

U prethodnim primerima bio je aktuelan slučaj da je *JavaScript* kod dodavan u veb stranice. U ovom slučaju stranica poseduje referencu na datoteku *websocket.js* koja predstavlja datoteku klijenta krajnje tačke *WebSocket* servera. Datoteka se kreira izborom opcije *File > New File*, a zatim se u prozoru za kreiranje nove datoteke iz kategorije *Web*, bira tip datoteke *JavaScript File*.



Slika 4.9 Kreiranje JS datoteke klijenta krajnje tačke WebSocket servera [izvor: autor]

U nastavku se otvara prozor u okviru koje se definiše naziv datoteke, u konkretnom slučaju *websocket.js*. Ova datoteka ima sledeće zadatke:

- mapira krajnju tačku *WebSocket* servera i odgovarajući *URI*;
- prihvata *JavaScript* događaje za dodavanje, uklanjanje i promenu statusa uređaja i prosleđuje ove događaje na *WebSocket* server;
- definiše povratnu metodu za *WebSocket onMessage* događaj;
- upravlja vidljivošću HTML forme za dodavanje novih uređaja.

Sledi listing datoteke *websocket.js*.

```

window.onload = init;
var socket = new WebSocket("ws://localhost:8080/WebsocketHome/actions");
socket.onmessage = onMessage;

function onMessage(event) {
    var device = JSON.parse(event.data);
    if (device.action === "add") {
        printDeviceElement(device);
    }
    if (device.action === "remove") {
        document.getElementById(device.id).remove();
        //device.parentNode.removeChild(device);
    }
    if (device.action === "toggle") {
        var node = document.getElementById(device.id);
        var statusText = node.children[2];
        if (device.status === "On") {
            statusText.innerHTML = "Status: " + device.status + " (<a href=\"#\"
OnClick=toggleDevice(\" + device.id + \")>Turn off</a>");
        } else if (device.status === "Off") {
            statusText.innerHTML = "Status: " + device.status + " (<a href=\"#\"
OnClick=toggleDevice(\" + device.id + \")>Turn on</a>");
        }
    }
}

function addDevice(name, type, description) {
    var DeviceAction = {
        action: "add",
        name: name,
        type: type,
        description: description
    };
    socket.send(JSON.stringify(DeviceAction));
}

function removeDevice(element) {
    var id = element;
    var DeviceAction = {
        action: "remove",
        id: id
    };
    socket.send(JSON.stringify(DeviceAction));
}

```

```
function toggleDevice(element) {
    var id = element;
    var DeviceAction = {
        action: "toggle",
        id: id
    };
    socket.send(JSON.stringify(DeviceAction));
}

function printDeviceElement(device) {
    var content = document.getElementById("content");

    var deviceDiv = document.createElement("div");
    deviceDiv.setAttribute("id", device.id);
    deviceDiv.setAttribute("class", "device " + device.type);
    content.appendChild(deviceDiv);

    var deviceName = document.createElement("span");
    deviceName.setAttribute("class", "deviceName");
    deviceName.innerHTML = device.name;
    deviceDiv.appendChild(deviceName);

    var deviceType = document.createElement("span");
    deviceType.innerHTML = "<b>Type:</b> " + device.type;
    deviceDiv.appendChild(deviceType);

    var deviceStatus = document.createElement("span");
    if (device.status === "On") {
        deviceStatus.innerHTML = "<b>Status:</b> " + device.status + " (<a href='#\" OnClick=toggleDevice(\" + device.id + \")>Turn off</a>\"");
    } else if (device.status === "Off") {
        deviceStatus.innerHTML = "<b>Status:</b> " + device.status + " (<a href='#\" OnClick=toggleDevice(\" + device.id + \")>Turn on</a>\"");
        //deviceDiv.setAttribute("class", "device off");
    }
    deviceDiv.appendChild(deviceStatus);

    var deviceDescription = document.createElement("span");
    deviceDescription.innerHTML = "<b>Comments:</b> " + device.description;
    deviceDiv.appendChild(deviceDescription);

    var removeDevice = document.createElement("span");
    removeDevice.setAttribute("class", "removeDevice");
    removeDevice.innerHTML = "<a href='#\" OnClick=removeDevice(\" + device.id + \")>Remove device</a>\"";
    deviceDiv.appendChild(removeDevice);
}

function showForm() {
    document.getElementById("addDeviceForm").style.display = '';
}

function hideForm() {
```

```
document.getElementById("addDeviceForm").style.display = "none";
}

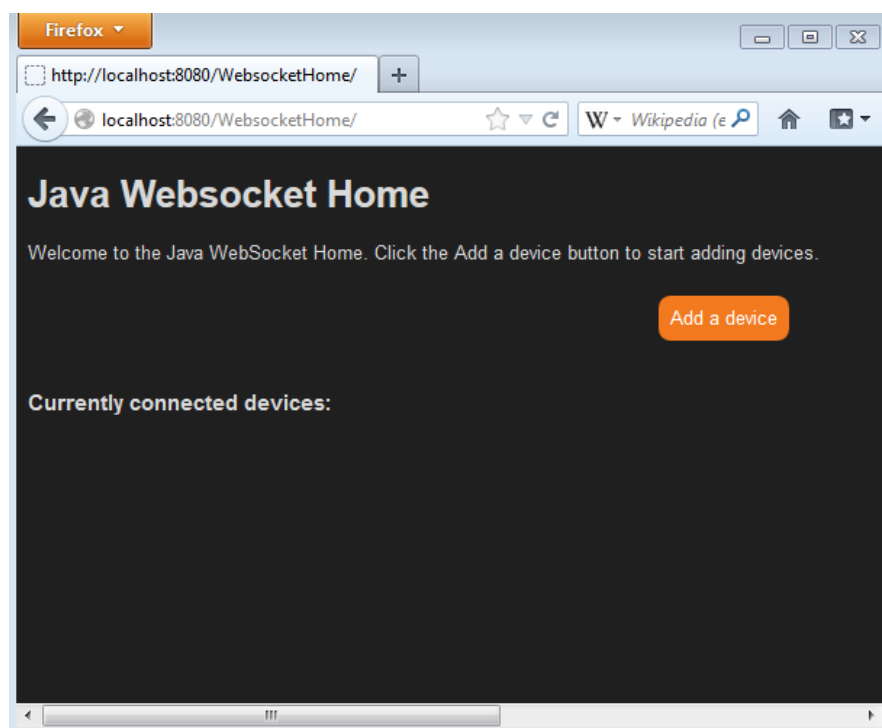
function formSubmit() {
    var form = document.getElementById("addDeviceForm");
    var name = form.elements["device_name"].value;
    var type = form.elements["device_type"].value;
    var description = form.elements["device_description"].value;
    hideForm();
    document.getElementById("addDeviceForm").reset();
    addDevice(name, type, description);
}

function init() {
    hideForm();
}
```

TESTIRANJE KREIRANE APLIKACIJE

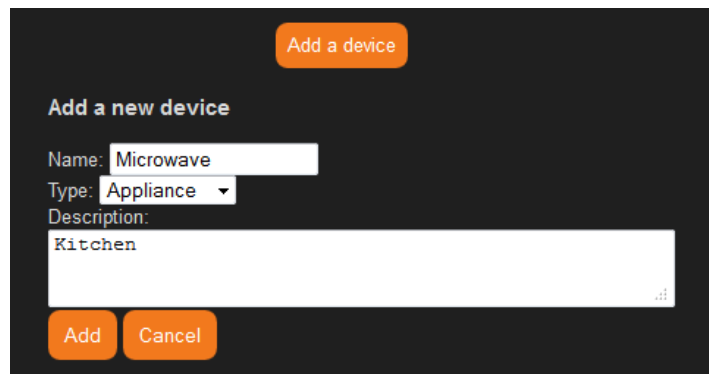
Na kraju je neophodno proveriti funkcionalnost kreiranog koda.

Klikom na opciju *Run Project*, u razvojnom okruženju *NetBeans IDE*, pokreće se aplikacija i učitava se početna stranica *index.html* u veb pregledač. Navedeno je prikazano sledećom slikom.



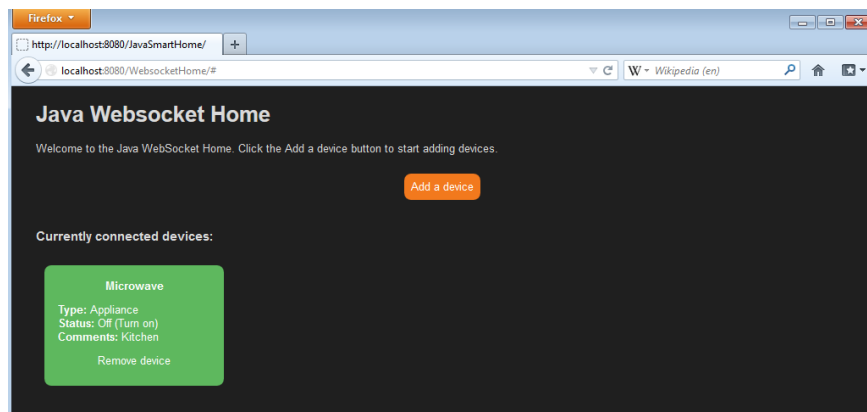
Slika 4.10 Početna stranica [izvor: autor]

Klikom na *Add a device*, moguće je početi dodavanje uređaja putem odgovarajuće forme kao na sledećoj slici.



Slika 4.11 Forma za dodavanje uređaja [izvor: autor]

Klikom na *Add* uređaj je registrovan od strane servera i biće aktivan sve dok ne bude uklonjen.

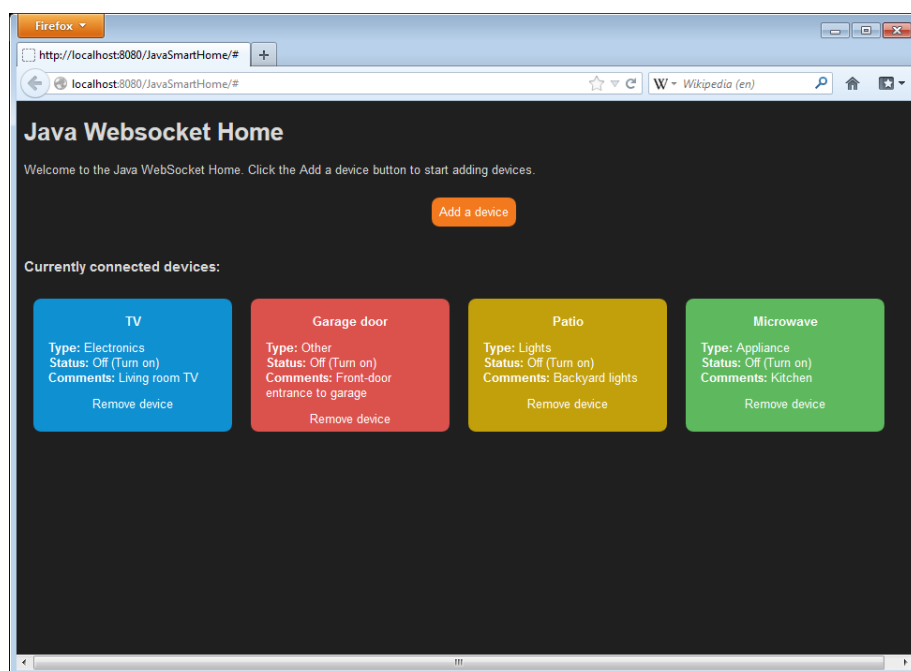


Slika 4.12 Dodati uređaj u aplikaciju [izvor: autor]

BRISANJE KREIRANOG UREĐAJA

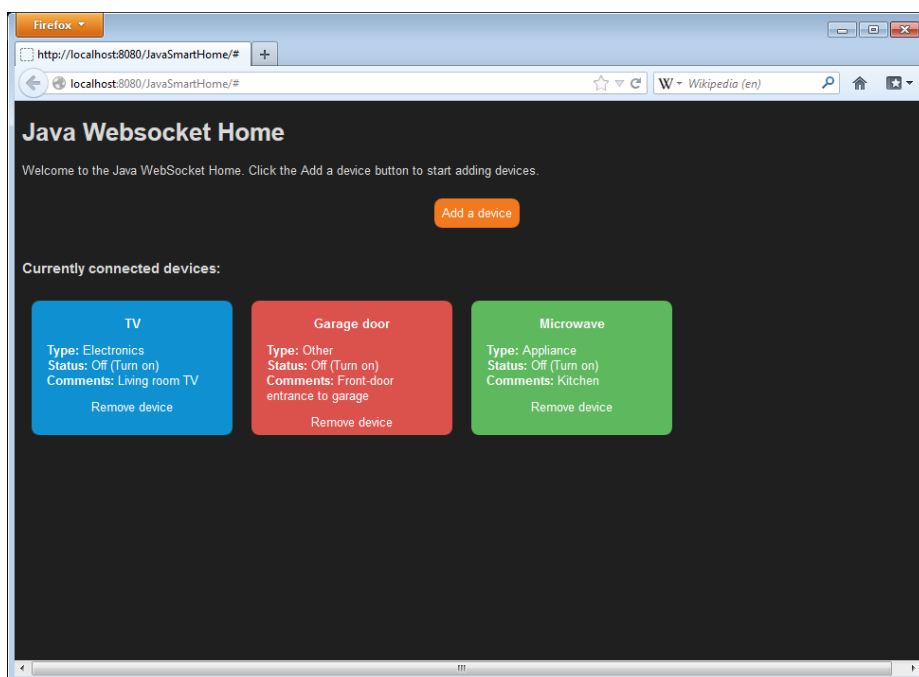
Još samo par slučajeva i testiranje je završeno.

Korisniku aplikacije su dostupni svi uređaji koji su dodati na prethodno opisani način. To je moguće ilustrovati sledećom slikom.



Slika 4.13 Lista dodatih uređaja u aplikaciju [izvor: autor]

Aplikacija dozvoljava i brisanje uređaja. Kod koji je zadužen za operacije brisanja i dodavanja *Device* objekata je detaljno analiziran u prethodnom izlaganju. Sledećom slikom je demonstrirana prethodna stranica iz koje je uklonjen konkretan uređaj klikom na link *Remove device*.



Slika 4.14 Brisanje postojećeg uređaja [izvor: autor]

VIDEO MATERIJALI

Izlaganje će biti dopunjeno prilaganjem odgovarajućih video materijala.

Java EE 7's Java API for WebSocket

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Java EE 7 & WebSocket API with Oracle's Arun Gupta

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

▼ Poglavlje 5

Pokazni primer - Java API za WebSocket

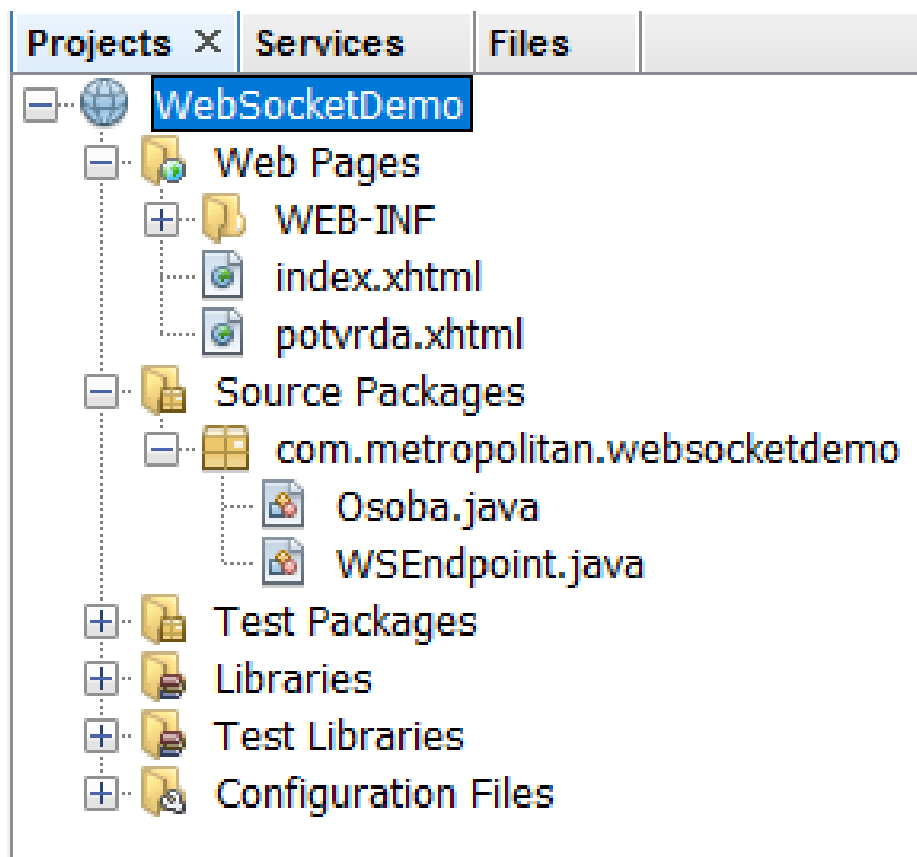
POSTAVKA (25 MIN)

Kreira se WebSocket Java aplikacija.

ZADATAK:

1. Primenom NetBeans IDE razvojnog okruženja kreirati aplikaciju koja poseduje funkcionalnost komuniciranja klijent i server strane pomoću WebSocket - a;
2. Projekat može da se zove Vezba12;
3. Kreirati JSF stranicu sa formom za unos podataka koji se šalju ka serveru: ime, prezime i grad studenta;
4. Krajnu tačku klijenta realizovati pomoću JavaScript -a;
5. JavaScript kod može biti u posebnoj datoteci ili integrisan u JSF stranicu;
6. Kreirati klasu krajnje tačke WebSocket servera;
7. Kreirati model klasu Osoba.java sa osobinama i odgovarajućim setter i getter metodama koje odgovaraju: imenu, prezimenu i gradu.
8. Pokrenuti program i obaviti testiranje.

Struktura projekta bi mogla da bude kao na sledećoj slici:



Slika 5.1 Predlog strukture projekta [izvor: autor]

KLIJENT STRANA JSF STRANICE

Prema uslovima prvo će biti kreiranje JSF stranice projekta.

Prva datoteka koja će u projektu biti kreirana je nastala redefinisanjem podrazumevane stranice index.xhtml. Stranica će pored forma za unos korisničkih podataka sadržati i JavaScript kod koji odgovara krajnoj tački klijenta WebSocket aplikacije. Kod ove stranice je priložen sledećim listingom:

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/
TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:jsf="http://xmlns.jcp.org/jsf">
  <head jsf:id="head">
    <title>WebSocket i Java EE</title>
    <script language="javascript" type="text/javascript">
      var wsUri = getRootUri() + "/WebSocketDemo/wsendpoint";

      function getRootUri() {
        return "ws://" + (document.location.hostname == "" ? "localhost" :
document.location.hostname) + ":" +
          (document.location.port == "" ? "8080" :
document.location.port);
      }
    </script>
  </head>
</html>
```

```

    }

    function init() {
        websocket = new WebSocket(wsUri);
        websocket.onopen = function (evt) {
            onOpen(evt)
        };
        websocket.onmessage = function (evt) {
            onMessage(evt)
        };
        websocket.onerror = function (evt) {
            onError(evt)
        };
    }

    function onOpen(evt) {
        console.log("CONNECTED");
    }

    function onMessage(evt) {
        console.log("RECEIVED: " + evt.data);

        var json = JSON.parse(evt.data);

        document.getElementById('ime').value=json.ime;
        document.getElementById('prezime').value=json.prezime;
        document.getElementById('grad').value=json.grad;
    }

    function onError(evt) {
        console.log('ERROR: ' + evt.data);
    }

    function doSend(message) {
        console.log("SENT: " + message);
        websocket.send(message);
    }

    window.addEventListener("load", init, false);

</script>

</head>
<body jsf:id="body">
    <form method="POST" jsf:prependId="false">
        <input type="button" value="Dodaj podrazumevano"
onclick="doSend('get_defaults')"/>
        <table>
            <tr>
                <td>Ime</td>
                <td>
                    <input type="text" jsf:id="ime"
jsf:value="#{osoba.ime}"/>

```

```

        </td>
    </tr>
    <tr>
        <td>Prezime</td>
        <td>
            <input type="text" jsf:id="prezime"
                jsf:value="#{osoba.prezime}"/>
        </td>
    </tr>
    <tr>
        <td>Grad</td>
        <td>
            <input type="text" jsf:id="grad"
                jsf:value="#{osoba.grad}"/>
        </td>
    </tr>
    <tr>
        <td></td>
        <td><input type="submit" value="Potvrdi"
jsf:action="potvrda"/></td>
    </tr>
</table>
</form>
</body>
</html>

```

U nastavku, biće kreirana i stranica pod nazivom potvrda.xhtml, čiji će zadatak biti prikazivanje unetih korisničkih podataka. Sledi listing ove stranice:

```

<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/
TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:jsf="http://xmlns.jcp.org/jsf">
    <head jsf:id="head">
        <title>Potvrda</title>
    </head>
    <body jsf:id="body">
        #{osoba.ime} #{osoba.prezime} #{osoba.grad}
    </body>
</html>

```

KREIRANJE MODELSKE KLASKE

Kreira se klasa instanci koje odgovaraju studentima (osobama) čiji se podaci unose.

U nastavku, akcenat se stavlja na kreiranje Java klasa projekta. Prva od njih će biti klasa Osoba.java koja će sadržati tražene osobine, koje se na početnoj formi unose, zajedno sa

seter i geter metodama. Naziv klase, odgovarajući paket, osobine i metode priloženi su sledećim listingom:

```
package com.metropolitan.websocketdemo;

import javax.enterprise.context.RequestScoped;
import javax.inject.Named;
/**
 *
 * @author Vladimir Milicevic
 */
@Named
@RequestScoped
public class Osoba {
    private String ime;
    private String prezime;
    private String grad;

    public String getGrad() {
        return grad;
    }

    public void setGrad(String grad) {
        this.grad = grad;
    }

    public String getIme() {
        return ime;
    }

    public void setIme(String ime) {
        this.ime = ime;
    }

    public String getPrezime() {
        return prezime;
    }

    public void setPrezime(String prezime) {
        this.prezime = prezime;
    }
}
```

KLASA KRAJNE TAČKE WEBSOCKET SERVERA

Konačno, kreira se i klasa krajne tačke WebSocket servera.

Izrada ove aplikacije biće zaokružena kreiranjem klase krajnje tačke WebSocket servera. Klasa se kreira na način opisan u predavanjima i nakon redefinisanja `onMessage()` metode ona dobija oblik priložen sledećim listingom:

```
package com.metropolitan.websocketdemo;

import javax.websocket.OnMessage;
import javax.websocket.server.ServerEndpoint;

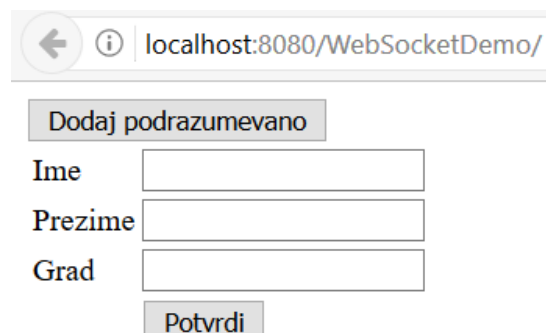
/**
 *
 * @author Vladimir Milicevic
 */
@ServerEndpoint("/wsendpoint")
public class WSEndpoint {

    @OnMessage
    public String onMessage(String message) {
        String retVal;
        if (message.equals("get_defaults")) {
            retVal = new StringBuilder("{").
                append("\ime\":"Automatski\").
                append("\prezime\":"Generisano\").
                append("\grad\":"Nepoznat\").
                append("}").toString();
        } else {
            retVal = "";
        }
        return retVal;
    }
}
```

DEMONSTRACIJA

Kreirana aplikacija se pokreće i testira.

Ako je sve dobro obavljeno, nakon prevođenja i pokretanja, pojavljuje se sledeća početna stranica:



← ⓘ localhost:8080/WebSocketDemo/

Dodaj podrazumevano

Ime

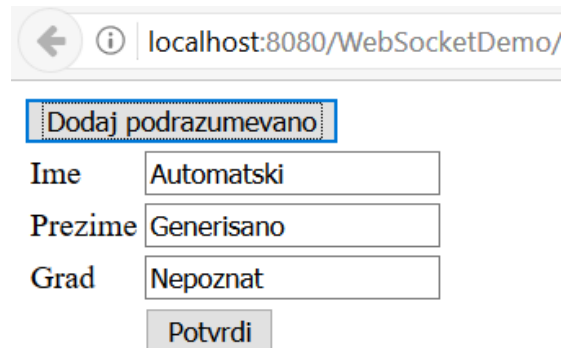
Prezime

Grad

Potvrdi

Slika 5.2 Početna stranica [izvor: autor]

Klikom na dugme *Dodaj podrazumevano*, krajnja tačka WebSocket servera vraća sledeći rezultat:



localhost:8080/WebSocketDemo/

Dodaj podrazumevano

Ime Automatski

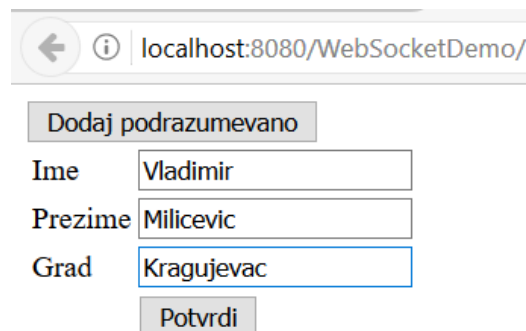
Prezime Generisano

Grad Nepoznat

Potvrdi

Slika 5.3 Server vraća podrazumevane vrednosti [izvor: autor]

Ako se u formu početne stranice unesu konkretni podaci, to može da bude ilustrovano na sledeći način:



localhost:8080/WebSocketDemo/

Dodaj podrazumevano

Ime Vladimir

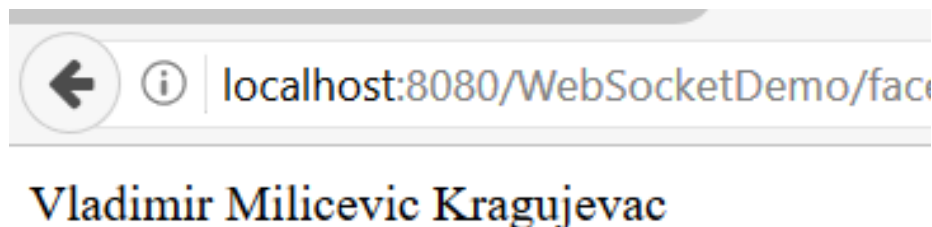
Prezime Milicevic

Grad Kragujevac

Potvrdi

Slika 5.4 Unos podataka na formu početne stranice [izvor: autor]

Klikom na dugme potvrdi, obavlja se klijent - server komunikacija preko WebSoceta i, ako je sve u redu, rezultat je sledeći (klik na dugme *Potvrdi*):



localhost:8080/WebSocketDemo/fac

Vladimir Milicevic Kragujevac

Slika 5.5 Podaci uneti na početnoj formi [izvor: autor]

▼ Poglavlje 6

Individualna vežba 12

INDIVIDUALNA VEŽBA

Pokušajte sami.

1. Pronađite WebSocket primer za demonstraciju rada bele table kao na sledećem linku: <https://netbeans.org/kb/docs/javaee/maven-websocketapi.html>;
2. Obavite analizu primera;
3. Pokušajte da dodate nove objekte koje je moguće postaviti na tablu;
4. Testirajte modifikovani primer.
5. Prilikom izrade individualne vežbe konsultujte se sa predmetnim asistentom

▼ Poglavlje 7

Domaći zadatak 12

ZADATAK 1

Uradite domaći zadatak

Uradite sledeće:

1. Osnova domaćeg zadatka je zadatak urađen na vežbama - unose se osobe (studenici);
2. Obezbediti još jedno polje za unos teksta - fakultet koji student studira;
3. Po uzoru na zadatka sa predavanja ([WebSocketHome](#)) obezbediti na početnoj stranici prikazivanje unetih studenata (dodavanje) kao i njihovo brisanje;
4. Testirati aplikaciju;
5. Dokumentovati u pdf formatu sve korake u izradi ove aplikacije.

Nakon izrade obaveznog zadatka studenti dobijaju dodatne različite zadatke na mail od predmetnog asistenta,

▼ Poglavlje 8

Zaključak

ZAKLJUČAK

Lekcija se bavila analizom i diskusijom u vezi sa problematikom Java API za WebSocket.

Lekcija je stavila detaljan akcenat na analizi i diskusiju koja je u direktnoj vezi sa problematikom *Java API za WebSocket*.

Posebno je istaknuto da je *WebSocket* nova *HTML5* tehnologija koja omogućava dvosmernu, punu dupleks (**full duplex**) komunikaciju između klijenta (obično se radi o veb pregledaču) i servera. Drugim rečima, primenom ove tehnologije je omogućeno je da serveri šalju podatke klijentima (u savremenim aplikacijama dominantno veb pregledačima) bez potrebe za čekanjem *HTTP zahteva*. Java EE 7 obezbeđuje potpunu podršku za razvoj *WebSocket* distribuiranih aplikacija. Ova podrška značajno dobija jednu kvalitetniju dimenziju kada se koristi u sinergiji sa savremenim Java razvojnim okruženjima. Tako, *NetBeans IDE* čini razvoj Java *WebSocket* aplikacija značajno olakšanim.

Imajući u vidu navedeno, lekcija se oslanjala na pažljivo birane teme koje obrađuju razvoj Java veb aplikacija sa *WebSocket* podrškom. Posebno, svako od izlaganja je bilo pokriveno pažljivo odabranim primerima koji služe svrsi lakšeg i kompletnijeg razumevanja i savladavanja aktuelne problematike.

Lekcija se kretala u dva smera:

- Analiza i diskusija *WebSocket* koda primenom primera koji je ugrađen u *NetBeans IDE* dokumentaciju;
- Razvoj vlastite Java EE aplikacije primenom *WebSocket* tehnologije.

Savladavanjem lekcije studenti su u potpunosti osposobljeni da kreiraju vlastite Java EE aplikacije primenom *WebSocket* tehnologije.

LITERATURA

Za pripremu lekcije korišćena je najnovija literatura.

1. Eric Jendrock, Ricardo Cervera-Navarro, Ian Evans, Kim Haase, William Markito. 2014. Java Platform, Enterprise Edition The Java EE Tutorial, Release 7, ORACLE
2. David R. Heffelfinger. 2015. Java EE7 Development With NetBeans 8, PACK Publishing

3. Yakov Fain. 2015. Java 8 programiranje, Kombib (Wiley)
4. Josh JUneau. 2015. Java EE7 Recipes, Apress
5. <http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/HomeWebsocket/WebsocketHome.html>