

VELEU ILIŠTE U RIJECI

Spec. dipl. str. Studij Informacijske tehnologije u poslovnim sustavima

**Projekt SVPP_2014_4 - Vodosprema
(projektna dokumentacija)**

Projektni tim: Nives Mileti , 2422000143/14
Anis Galijatovi , 2422000138/14
Vjekoslav Bari , 2422000112/14
Neven Frkovi , 2422000125/14

Rijeka, sije anj 2015.

SAŽETAK

Ključne riječi: Aplikacija; Ventil; Senzor; Pumpa;

SADRŽAJ

UVOD	1
PRIPREMA RAZVOJNOG OKRUŽENJA SUSTAVA	2
PROGRAMSKI DIO SUSTAVA	5
Nadzorna aplikacija.....	5
Klase.....	5
Senzor.....	9
Ventil	11
Pumpa.....	12
Simulatorska aplikacija	13
GRAFI KI DIO NADZORNE APLIKACIJE	17
GRAFI KI DIO SIMULATORSKE APLIKACIJE	20
POPIS SLIKA	22

UVOD

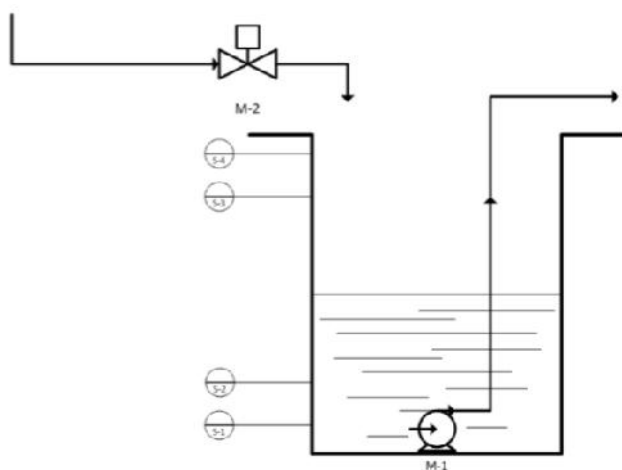
Za rješenje projektnog zadatka potrebno je razviti dvije aplikacije, a to se odnosi na:

- 1) Simulatorska aplikacija – simulira proizvodni sustav vodospreme
- 2) Nadzorna aplikacija – upravljanje ventilom i pumpom vodospreme simulatorske aplikacije

Simulatorska aplikacija zamišljena je kao web aplikacija troslojne arhitekture, razvijena pomoću tehnologija: *HTML5*, *CSS3*, *JavaScript*, *PHP* i *AJAX*-a. Logički sloj koji u ovom slučaju djeluje kao servis, razvijen pomoću *PHP*-a, povezan je s *MySQL* bazom podataka u kojoj je zapisana razina trenutnog stanja.

Nadzorna aplikacija je razvijena pomoću objektno orijentiranog programskog jezika *JAVA* unutar razvojnog okruženja *Eclipse*.

Za projektni zadatak bilo je potrebno obraditi i prikazati proces dotoka tekućine u spremnik koji je kontroliran radom ventila, a istjecanje tekućine iz njega omogućeno je regulacijom rada pumpe, što je prikazano na slici Slika 1. Izrada ovog zadatka bit će prikazana po odabranim cjelinama razrade.

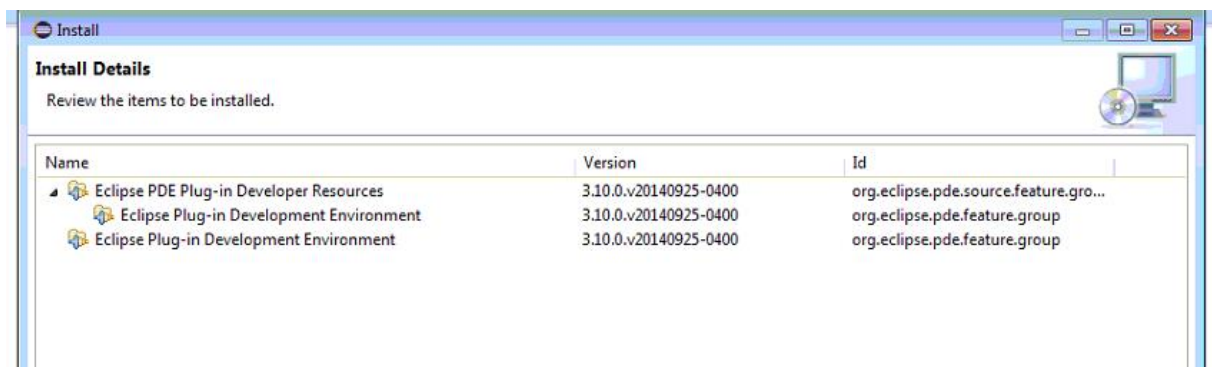


Slika 1. Vodosprema

PRIPREMA RAZVOJNOG OKRUŽENJA SUSTAVA

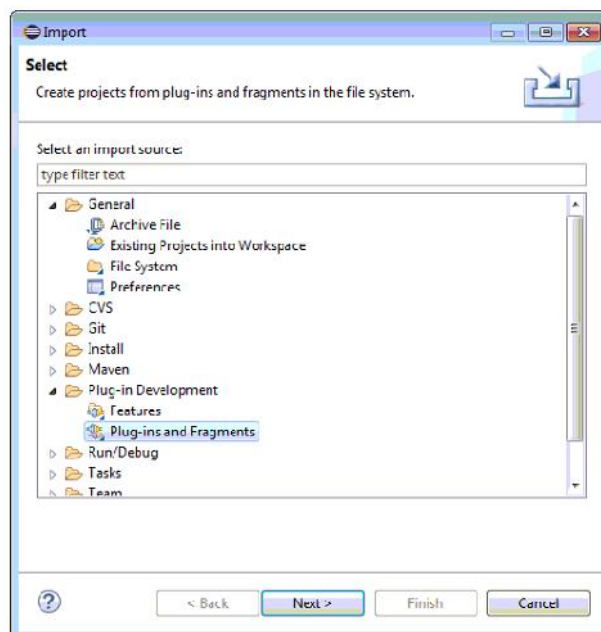
Prije po etka razvoja nadzorne aplikacije potrebno je postaviti i podesiti nekoliko dodatnih opcija u samom razvojnom su elju razvojnog okruženja *Eclipse*.

Najprije je potrebno omogu iti korištenje tzv. *plug-in* mogu nosti za nadogradnju razvojnog su elja, a isto je mogu e dobiti putem platforme. Slika 2. prikazuje instalirane dodatke razvojnog okruženja.



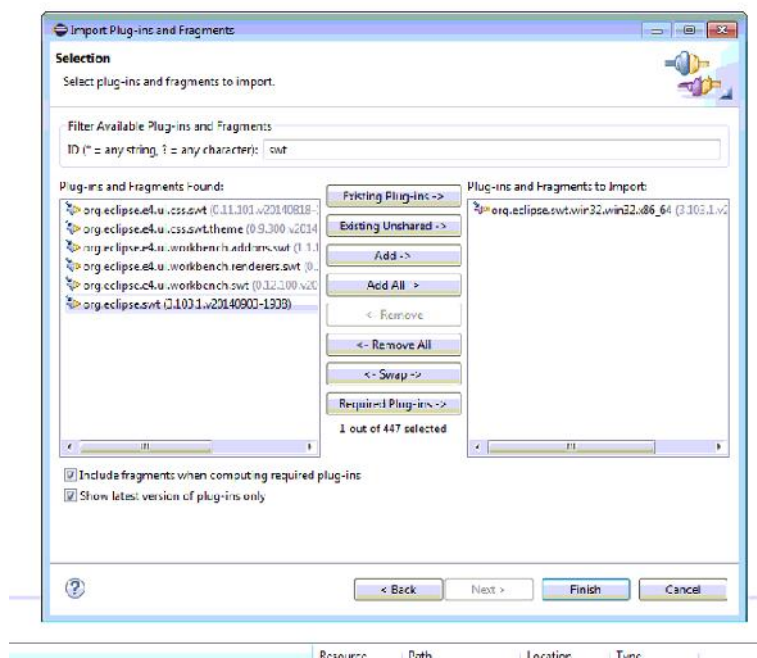
Slika 2. Prikaz korištenog softvera u razvojnom okruženju

Istim programskim paketom omogu eno je korištenje i uvoženje dodatnih fragmenata programa korištenih za razvoj aplikacije (Slika 3.).



Slika 3. Uvoženje dodatnih alata

Nadalje, potrebno je odabrati odgovaraju i SWT (eng. *Standard Widget Toolkit*) paket alata koji prilikom razvoja korisniku omogu avaju korištenje i interakciju s grafi kim objektima aplikacije.



Slika 4. Prikaz korištenog toolkit-a

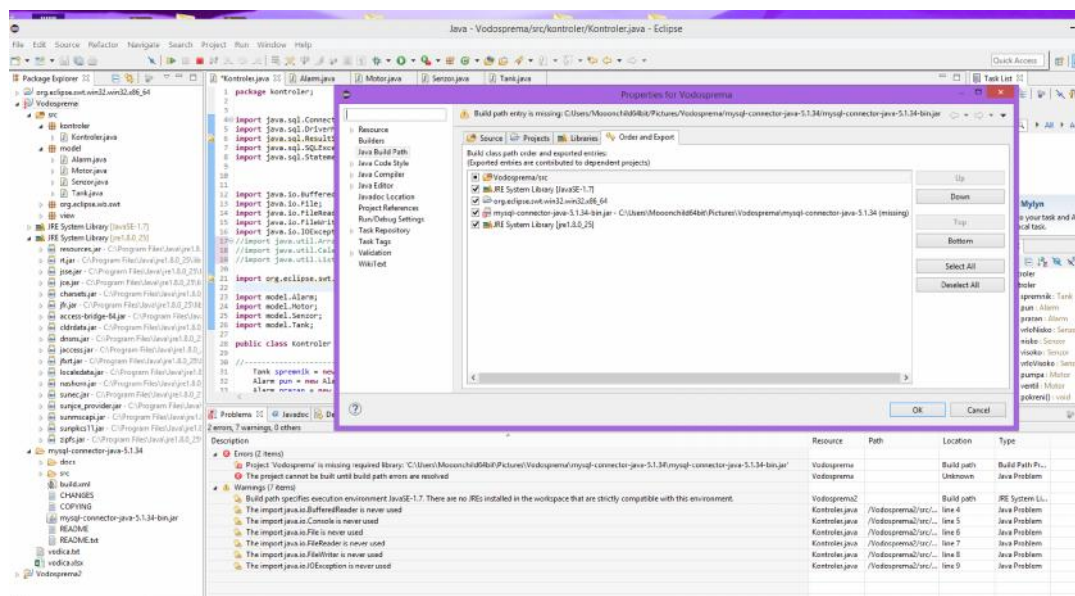
Dodatno tome, potrebno je omogu iti i povezivanje s bazom podataka, a isto je omogu eno s MySQL konektorom u obliku JAR datoteke (eng. *Java ARchive*), prikazanog slikom Slika 5.



mysql-connector-java-5.1.34-bin.jar

Slika 5. Prikaz korištenog toolkit-a

Tako er, za potpunu funkcionalnost razvojnog okruženja postavljaju se putanje alata za nadogradnju razvojnog su elja (Slika 6.). Te su nadogradnje od iznimne važnosti kako bi aplikacija mogla biti pravilno razvijena, jer u slu aju da nešto od navedenog nedostaje, pokretanje aplikacije putem razvojnog su elja ne e biti mogu e, što posljedi no uzrokuje javljanje grešaka o nedostatku potrebnih nadogradnji.



Slika 6. Nadogradnja

Što se ti e simulatorskog dijela sustava, odnosno web aplikacije, za njen je razvoj potreban *Notepad++* kao razvojno okruženje *HTML* i *PHP* datoteka korištenih za rad tog dijela. Osim toga potrebno je postaviti web poslužitelj, u ovom slučaju *Apache* s pristupom *phpmyadmin* panelu pomoću kojeg je izrađena *MySQL* baza podataka korištena za komunikaciju između desktop (nadzorne) i web (simulatorske) aplikacije.

PROGRAMSKI DIO SUSTAVA

Nadzorna aplikacija

Klase

Razvoj nadzorne aplikacije započet je u odnosu na zadani zadatak projekta „Vodosprema“. U početku su definirane klase „Senzor“, „Tank“, „Motor“, „Alarm“.

U klasi „Tank“ deklarirana je varijabla tipa *int* naziva „razina“ te su definirane metode „getRazina()“ tipa *int* i „setRazina()“ tipa *void*. Klasa se nalazi u određenom programskom paketu „model“. Te metode omogućuju korištenje funkcionalnosti po potrebi, kao što je to u slučaju kontrole i učitavanja razine u odnosu na količinu vode u spremniku. Slijedi primjer koda:

```
public class Tank {  
    private int razina = 0;  
    public int getRazina() {  
        return razina;  
    }  
    public void setRazina(int razina) {  
        this.razina = razina;  
    }  
}
```

U klasi „Senzor“ deklarirana je varijabla tipa *boolean* naziva „detektira“ te su i u ovoj klasi na isti način izgrađene funkcije za učitavanje objekta „detektira“, a isto omogućuje funkciju rada senzora. Slijedi primjer koda:

```
public class Senzor {  
    private boolean detektira=false;  
    public boolean isDetektira() {  
        return detektira;  
    }  
    public void setDetektira(boolean detektira) {  
        this.detektira = detektira;  
    }  
}
```

U klasi „Motor“ deklarirana je varijabla tipa *boolean* naziva „otvoren_upaljena“ te je njome posljedično moguće definirati stanje rada pumpe (*true* ili *false*). Slijedi primjer koda:

```
public class Motor {  
    private boolean otvoren_upaljena=false;  
    public boolean isOtvoren_upaljena() {  
        return otvoren_upaljena;  
    }  
    public void setOtvoren_upaljena(boolean otvoren_upaljena) {  
        this.otvoren_upaljena = otvoren_upaljena;  
    }  
}
```



```
}
```

U klasi „Alarm“ deklarirana je varijabla tipa *boolean* naziva „aktiviran“ ija je po etna vrijednost *false*. S metodom „setAktiviran“ definiran je tip objekta koji se odnosi na stanje spremnika. Slijedi primjer koda:

```
public class Alarm {  
    private boolean aktiviran=false;  
    public boolean isAktiviran() {  
        return aktiviran;  
    }  
    public void setAktiviran(boolean aktiviran) {  
        this.aktiviran = aktiviran;  
    }  
}
```

U sljedećem primjeru koda, definirana je metoda naziva „pooling“ tipa *int* koja varijabli „razina“ dodjeljuje vrijednost trenutne razine spremnika pozivanjem metode „getRazina“. Unutar nje postavljeni su i uvjeti koji definiraju vrijednost razine koja u postavljenim slučajevima ne smije prelaziti maksimalnu vrijednost 500 i minimalnu vrijednost 0. Na taj način definirane su konstantne granice razine vode u spremniku.

```
public int pooling() {  
    int razina;  
    razina=spremnik.getRazina();  
    if(razina<=0) {  
        razina=0; }  
    if(razina>=500)  
    {        razina=500; }  
    return razina; }  
}
```

U aplikaciji je omogućeno spremanje vrijednosti razine vode na dva načina.

- 1) Spremanje vrijednosti u datoteku – *offline* pohrana u *txt* datoteku
- 2) Spremanje vrijednosti u online bazu podataka – *online* pohrana u BP

Spremanje vrijednosti u datoteku omogućeno je inicijalizacijom objekta *file* pomoću kojeg se stvara tekstualna datoteka u koju se zapisuje svaka promjena razine uzrokovana radom nadzorne aplikacije. Također, osim tog lokalnog zapisa, omogućeno je i zapis unutar udaljene baze podataka koji služi za osvježavanja trenutne razine unutar web aplikacije koja predstavlja simulatorski dio sustava. Nadalje se nalazi primjer koda strukture metode „Datotekica“ koja sadrži blokove naredbi za funkcionalnost pohrane vrijednosti u tekstualnu datoteku.

```
File file = new File("vodica.txt");  
FileWriter writer=null;
```

```

try {
    writer = new FileWriter(file, true);
    writer.write(String.valueOf(razina));
    writer.write("\r\n");
    writer.flush();
    writer.close();
}

catch (IOException e1) {
    e1.printStackTrace();
} /*Stavranje i spremanje u datoteku; \r\n je za ispis
vrijednost jedno ispod drugog u txt datoteci*/
}

```

Za učitavanje vrijednosti iz datoteke pohranjene lokalno na disku korištena je metoda tipa *void* naziva „UcitajDat“, prikazana u nastavku.

```

public void UcitajDat() {
    File file = new File("vodica.txt");
    String strLine=" ";
    StringBuilder text = new StringBuilder();
    if(file.exists())
    {
        try {
            FileReader fReader = new FileReader(file);
            BufferedReader bReader = new BufferedReader(fReader);
            while( (strLine=bReader.readLine()) != null ){
                spremnik.setRazina(Integer.parseInt(strLine));
            }
        }
        catch (Exception e)
        {
            spremnik.setRazina(250);
        }
        else
        {
            spremnik.setRazina(250);
        }
    }
}

```

Za učitavanje vrijednosti iz baze podataka pohranjene na udaljenom poslužitelju korištena je metoda tipa *void* naziva „UcitajizBaze“.

```

public void UcitajizBaze(){
    try {
        Class.forName("com.mysql.jdbc.Driver");
        Connection con =
        DriverManager.getConnection("jdbc:mysql://headless.eur.hr:3306/headless_riteatar",
        "headless_razina", "razina");
        Statement stmt = (Statement) con.createStatement();
        String select = "SELECT razina FROM Razina ORDER BY id DESC LIMIT 1";
        stmt.executeUpdate(select);
    } catch (ClassNotFoundException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (SQLException e) {
        // TODO Auto-generated catch block
    }
}

```

```

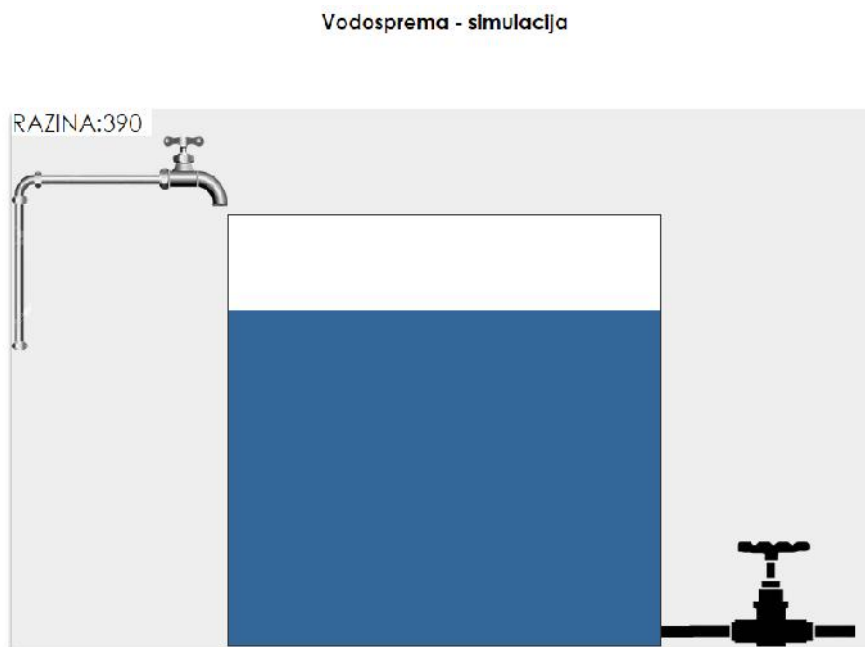
        e.printStackTrace();
    }
}

```

Baza podataka je strukturirana na jednostavan način u kojem je kreirana tablica s dva atributa (Slika 7.): 'id' tipa *int* i 'Razina' tipa *int* u kojoj se korištenjem nadzorne aplikacije ažurira vrijednost razine koju simulatorska aplikacija koristi kao referencu u prikazu trenutne razine (Slika 8.).



Slika 7. Prikaz strukture baze podataka u phpmyadmin-u



Slika 8. Prikaz simulatorske aplikacije

Senzor

Senzor ili pretvornik je uređaj koji mjeri fizikalnu veličinu (npr. temperature, vlažnosti zraka, tlaka, broj okretaja motora) i pretvara ju u signal pogodan za daljnju obradu. U ovome slučaju senzori su s tehničkog pogleda u aplikaciji korišteni za detekciju razine kao trenutnog stanja vode u spremniku.

Pri pokretanju aplikacije, senzori bi trebali očitavati vrijednost 0, a u ovom je slučaju napravljena klasa „Senzor“ čija je početna vrijednost *false* kao što je to navedeno ranije.

Ova klasa nalazi se u imenskom prostoru „model“ te se njeni atributi pozivaju po potrebi, a naredbom „**import** model.Senzor“ u javnoj klasi „Kontroler“ deklarirani su tipovi senzora.

Tipovi senzora dijele se na četiri stanja s obzirom na interval na koji se odnose:

- 1) Senzor *vrloNisko* – detektira razinu vode u rasponu od ≤ 50 i ≥ 10
 - 2) Senzor *nisko* – detektira razinu vode u rasponu od ≤ 150 i ≥ 60
 - 3) Senzor *visoko* – detektira razinu vode u rasponu od ≤ 440 i ≥ 350
 - 4) Senzor *vrloVisoko* – detektira razinu vode u rasponu od ≥ 450 i ≤ 490
- Pomoću pomoćne varijable „razinaTek“ u predviđenim *if* grananjima toka, deklarirane su i varijable tipa *String*, „stanjeSpremnika“ i „stanje“.

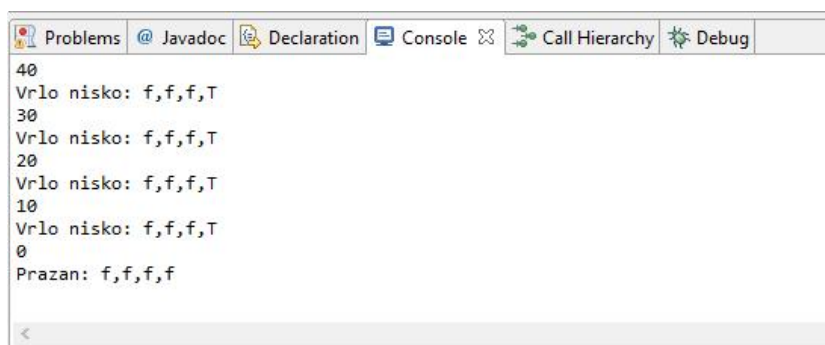
Primjer koda u kojemu su deklarirane varijable za ispis i provjeru stanja prikazan je u nastavku.

```
int razinaTek=spremnik.getRazina();
String stanja= " ";
String stanjeSpremnika= " ";
```

Napravljeno je i dodatno *if* grananje toka koje pomoću pomoćne varijable „razinaTek“ provjerava vrijednost, a ukoliko je veća ili jednaka nuli, aktivira se alarm metodama *prazan.setAktiviran(true)* i *pun.setAktiviran(true)* što uzrokuje ispisom sljedećih prilagođenih vrijednosti:

- *prazan.setAktiviran(true)* - ispisuje „Prazan: f,f,f“. U tom slučaju senzori su postavljeni na *false* (tada oni nisu aktivni) i aktiviran je alarm koji u praksi označava alarmantno stanje.
- *pun.setAktiviran(true)* - ispisuje stanje „Pun“

Na slici Slika 9. prikazan je ispis tih poruka unutar konzole.



```
40  
Vrlo nisko: f,f,f,T  
30  
Vrlo nisko: f,f,f,T  
20  
Vrlo nisko: f,f,f,T  
10  
Vrlo nisko: f,f,f,T  
0  
Prazan: f,f,f,f
```

Slika 9. Ispis stanja tanka u konzoli aplikacije

Na slici Slika 10. prikazan je alarm stanja praznog spremnika.



Slika 10. Prikaz alarma

Na slici Slika 11. prikazana je nadzorna aplikacija, a na slici Slika 12. prozor koji prikazuje informaciju o stanju tanka. Tada se i u konzoli aplikacije ispisuje stanje tanka putem *POP-UP* prozora.



Slika 11. ispis stanja punog tanka



Slika 12. Ispis stanja tanka "Visoko"

Na slici Slika 12. tako er su prikazani senzori pomo u kojih je vidljivo stanje razine vode u spremniku. Što se ti e grafi kog djela aplikacije za prikaz istog korišteni su sljede i elementi:

- 1) Button btnPokreni – gumb za pokretanje aplikacije
- 2) final Label lblAlarm – labela za prikazivanje alarma. Za prikaz su korištena sljede a svojstva:
- 3) Label lblRazina_1 – grafi ki element za ispis labele „Razina“

```
final Label lblStatus;
lblStatus.setBackground(SWTResourceManager.getColor(SWT.COLOR_BLACK));
    lblStatus.setForeground(SWTResourceManager.getColor(SWT.COLOR_GREEN));
lblStatus.setText("Trenutno stanje procesa: ");
lblStatus.setAlignment(SWT.LEFT);
```

- 4) Label lblVrloVisoko, lblVisoko, lblNisko, lblVrloNisko - grafi ki prikaz stanja u tanku. Njihov prikaz ovisi o postavljenim uvjetima za itanje razine.
- 5) lblStatus – ispisuje trenutno stanje procesa, a primjer koda je:

```
("Trenutno stanje procesa: \nPUMPA: "+stanjePumpa +
    "\nVENTIL: "+stanjeVentil+ "\nRAZINA: "+razina+" l");
```

Stanje razine uz prilago ene informacije i vrijednosti ispisuju se i u konzoli razvojnog okruženja.

Ventil

Ventil je u nadzornoj aplikaciji korišten u svrhu puštanja vode u spremnik. Pomoću nadalje prikazanih metoda definirana je struktura koda koja se odnosi na funkcije ventila u aplikaciji. U metodi „otvoriVentil“ putem *if* grananja toka zadan je uvjet koji se odnosi na *true* stanje ventila, kao upaljeno stanje i tada je u kodu obrađeno povećanje razine tekućine (razina se povećava za vrijednost 10). Primjer koda prikazan je u nastavku.

```
public void otvoriVentil(boolean stanjeVentila)
{
    if(stanjeVentila == true)
    {
        int razinaTek=spremnik.getRazina();
        razinaTek=razinaTek+10;
        spremnik.setRazina(razinaTek);
        ventil.setOtvoren_upaljena(true);
    }
}
```

Metoda „upaljenoOtvoreno()“ stvorena je iz razloga što je u aplikaciji moguće upaliti pumpu i ventil u isto vrijeme. Tako je u metodi definirano da se razina tekućine povećava za vrijednost 6, što nije jednako kao u prethodnoj metodi. U tom slučaju pumpa mora biti otvorena te je proslijeđena *true* vrijednost.

```
public void upaljenoOtvoreno()
{
    int razinaTek=spremnik.getRazina();
    razinaTek=razinaTek+6;
    spremnik.setRazina(razinaTek);
    pumpa.setOtvoren_upaljena(true);
}
```

Pumpa

Metoda „upaliPumpu()“ radi pod uvjetom da je varijabla „stanjePumpe“ postavljena na *true*. Tada se pomoću varijable „razinaTek“ njegova vrijednost uvijek umanjuje za vrijednost 10.

```
public void upaliPumpu(boolean stanjePumpe)
{
    if(stanjePumpe==true)
    {
        int razinaTek=spremnik.getRazina();
        razinaTek=razinaTek-10;
    }
}
```

```
        spremnik.setRazina(razinaTek);  
        pumpa.setOtvoren_upaljena(true);  
    }  
}
```

Simulatorska aplikacija

Simulatorska je aplikacija (Slika 8.) kao web aplikacija od dva osnovna logička sloja razvijena kao dvije datoteke: *HTML* i *PHP*.

Unutar *HTML* datoteke nalaze se četiri osnovna elementa za potpunu funkcionalnost simulacije: element `<progress>`, koji predstavlja razinu vode, element `<div id="ventil">`, koji u animacijskom smislu predstavlja aktivaciju ventila, element `<div id="pumpa">`, koji u animacijskom smislu predstavlja aktivaciju pumpe te `<div id="razina">`, koji služi kao tekstualni okvir prikaza vrijednosti trenutne razine dohvaćene iz baze podataka.

Osim *HTML* elemenata, od iznimne su važnosti *JavaScript* skriptni dijelovi datoteke, pomoću kojih su razvijeni potrebni algoritmi za provjeru i uspoređivanje stanja pa tako i dodavanje, odnosno oduzimanje potrebnih *CSS* klasa nad elementima koji su u tijeku animacije.

U tom su dijelu definirane dvije osnovne funkcije, `dataBase()`, koja služi za dohvaćanje podataka pomoću `get.php` datoteke, kao i `animacija()`, koja *if* grananjima toka, s obzirom na uvjete, mijenja stanja klasa nad *HTML* elementima te tako vrši animaciju.

U nastavku je prikazana funkcija `dataBase()`:

```
function dataBase() {  
    xmlhttp = new XMLHttpRequest();  
    xmlhttp.onreadystatechange = function() {
```



```

    if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
        document.getElementById("razina").innerHTML = xmlhttp.responseText;
        var razina = document.getElementById("lbl").innerHTML;
        document.getElementById("water").value = razina;
        setTimeout(function(){ animacija(); }, 100);
    }
    xmlhttp.open("GET","get.php",true);
    xmlhttp.send();
}

setTimeout(function(){ dataBase }, 1000);
}

```

Ova je funkcija je rekurzivna iz razloga stalnog pregleda promjene stanja i to svaku sekundu. Ujedno, osim što funkcija poziva samu sebe, poziva i drugu funkciju, odnosno animacija(), prikazanu u nastavku:

```

function animacija()
{
    var razina = document.getElementById("lbl").innerHTML;
    document.getElementById("water").value = razina;
    var raz = document.getElementById('txtHint2').innerHTML;
    if(razina<raz && razina%10==0)
    {
        document.getElementById('txtHint3').innerHTML = "pumpa";
        document.getElementById('ventil').classList.remove('ventil2');
        document.getElementById('ventil').classList.add('ventil');
        document.getElementById('pumpa').classList.remove('pumpa');
        document.getElementById('pumpa').classList.add('pumpa2');
        document.getElementById('pumpa').style.backgroundImage = "url('pipe2.png')";
        document.getElementById('ventil').style.backgroundImage = "url('ventil.png')";
    }
    else if(razina>raz && razina%10==0)
    {
        document.getElementById('txtHint3').innerHTML = "ventil";
        document.getElementById('pumpa').classList.remove('pumpa2');
    }
}

```

```

document.getElementById('pumpa').classList.add('pumpa');
document.getElementById('ventil').classList.remove('ventil');
document.getElementById('ventil').classList.add('ventil2');
document.getElementById('ventil').style.backgroundImage = "url('ventil2.png')";
}
else if(razina>raz || razina<raz && razina%10!=0)
{
document.getElementById('pumpa').classList.remove('pumpa');
document.getElementById('pumpa').classList.add('pumpa2');
document.getElementById('ventil').classList.remove('ventil');
document.getElementById('ventil').classList.add('ventil2');
document.getElementById('pumpa').style.backgroundImage = "url('pipe2.png')";
document.getElementById('ventil').style.backgroundImage = "url('ventil2.png')";
}
else
{
document.getElementById('txtHint3').innerHTML = "equal";
document.getElementById('pumpa').style.backgroundImage = "url('pipe.png')";
}
raz = document.getElementById("lbl").innerHTML;
document.getElementById('txtHint2').innerHTML = raz;
}

```

PHP datoteka koja se koristi u ovoj simulatorskoj aplikaciji definira povezivanje na *MySQL* bazu podataka i naredbom *echo* ispisuje `<label>` *HTML* element koji se kasnije pomoću *JavaScripta* smješta u `<div id="razina">` element kao rezultat vrijednosti razine.

PHP naredbe su:

```

<?php
$con = mysqli_connect('serverurl','baza','tablica','korisnik');
if (!$con) {
    die('Could not connect: ' . mysqli_error($con));
}
mysqli_select_db($con,"ajax_demo");

```

```
$sql="SELECT * FROM razina";  
$result = mysqli_query($con,$sql);  
while($row = mysqli_fetch_array($result)) {  
echo "<label id='lbl' value='" . $row['Razina'] . "' style='height:50px; height:200px; font-size:22px;'>" .  
$row['Razina'] . "</label>";  
}  
mysqli_close($con);  
?>
```

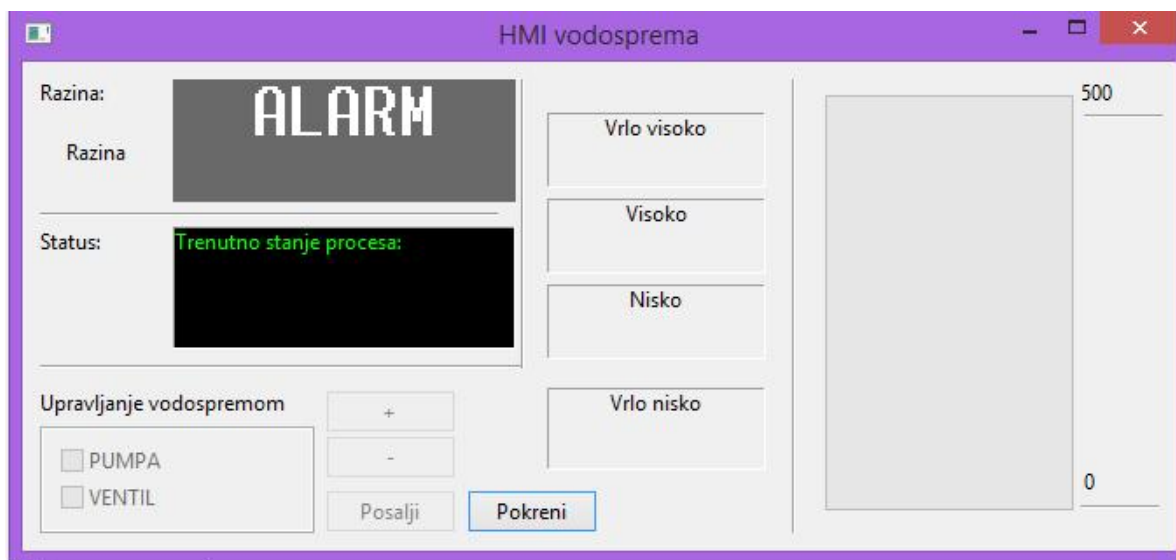
GRAFIČKI DIO NADZORNE APLIKACIJE

Kod za prikaz i definiranje ponašanja grafičkih elemenata aplikacije definiran je u zasebnoj Java datoteci „HMIVodosprema“. U ovoj datoteci bilo je potrebno izvršiti *import* različitih paketa, a oni ovise o grafičkim elementima koji su korišteni. Uglavnom su to „org.eclipse.swt.“ imenski prostori korišteni za pravilan prikaz elemenata.

Kod za deklaraciju glavne metode u izvršnoj klasi „HMIVodosprema“ je sljedeći:

```
public static void main(String[] args) {  
    try {  
        HMIVodosprema window = new HMIVodosprema();  
        window.open();  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```

Prilikom izvršavanja, traži se na navedenoj lokaciji objekt naziva „window“, te `main()` metoda u toj klasi. Ukoliko metoda nije nađena, javlja se greška. Izvršavanje programa interpreterom pokrećemo na in da na *root* mapu projekta odaberemo opciju „*Debug As → Java Application*“. U nastavku, na slici Slika 13. prikazan je izgled aplikacije nakon otvaranja.



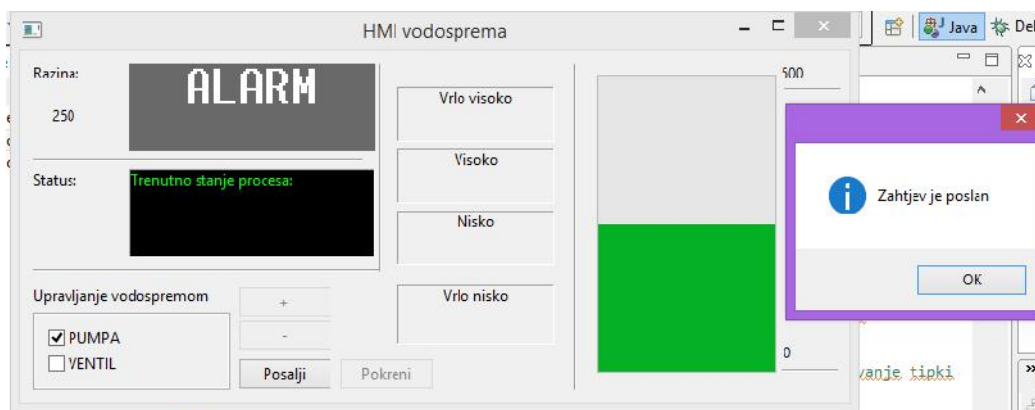
Slika 13. Prikaz nadzorne aplikacije nakon otvaranja

Slika prikazuje izvršenu, odnosno otvorenu aplikaciju, ali ne i pokrenutu. Istu se pokreće pomoću gumba „Pokreni“. Isto tako valja napomenuti da je tek tada moguća interakcija s ostalim elementima aplikacije.

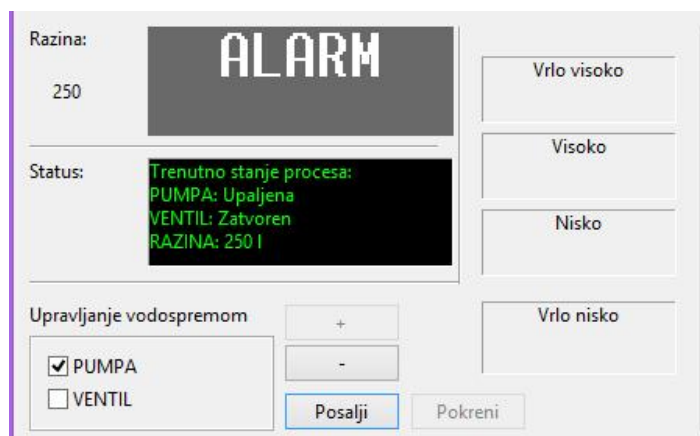
Za grafi ke elemente, u odnosu na korištenje odre enih mogu nosti nadzorne aplikacije (npr. omogu avanje rada pumpe), postavljeni su neki od uvjeta za prikaz odgovora aplikacije prema upitu. Slijedi primjer koda koji definira na in rada i korištenje pumpe u aplikaciji:

```
String razina=String.valueOf(razinaI);
pumpa=btnPumpa.getSelection();
MessageBox boks = new MessageBox(shlHmiVodosprema);
boks.setMessage("Zahtjev je poslan");
boks.open();

if (pumpa == true && razinaI > 0) {
    stanjePumpa="Upaljena";
    btnMinus.setEnabled(true);
}
else {
    stanjePumpa="Ugasena";
    btnMinus.setEnabled(false);
}
```



Slika 14. Prikaz upita prema aplikaciji



Slika 15. Prikaz po etnog stanja pumpe

U ovom je slučaju moguće korištenje pumpe u aplikaciji. Tada je moguće koristiti i gumb „-“ koji u odgovara radu pumpe, odnosno oduzimanju vode iz spremnika. Na jednak je način omogućen rad ventila, u slučaju kada pumpa ne smije raditi.

U ovom djelu važno je spomenuti još i rad objekta „arduino“ koji služi kao poveznica između programskog modela i grafičkih elemenata aplikacije. Taj objekt služi da, primjerice, registrira i omogućava paralelni rad pumpe i ventila u nadzornoj aplikaciji. Nadalje će, uz primjer deklaracije u kodu, biti objašnjena i njegova uporaba.

```
int razinaI = arduino.pooling(); {  
if(obje == true && razina <= 500)  
arduino.upaljenoOtvoreno();  
arduino.Datotekica(razina);  
}
```

U ovom slučaju pomoću objekta „arduino“ pozivamo metodu „upaljenoOtvoreno()“ s kojom se upravlja razinom vode u spremniku, dok je pumpa upaljena, a ventil otvoren.

Valja napomenuti i uporabu tog objekta u aplikaciji, a odnosi se na očitavanje razine i stanja spremnika:

```
arduino.kruzniProces();
```

GRAFIČKI DIO SIMULATORSKE APLIKACIJE

Svi grafi i dijelovi simulatorske aplikacije definirani su prethodno navedenim *HTML* elementima, odnosno tagovima, a stilska svojstva, koja obuhvaćaju dimenzije, boje i tranzicije postignuta su CSS-om (eng. *Cascade Style Sheet*).

CSS je prikazan u nastavku:

```
#water {
-webkit-appearance: none;
appearance: none;
border:1px solid #333;
}
#water::-webkit-progress-value
{
background-color:#336699;
transition: 1s ease;
}
#water::-webkit-progress-bar
{
background-color:#fff;
}
.pumpa
{
width:180px;
height:97px;
background-image:url('pipe.png');
background-size:100% 100%;
float:right;
margin-top:400px;
margin-left:620px;
transition:0.2s;
}
.pumpa2
{
width:180px;
```

```
height:97px;
background-image:url('pipe.png');
background-size:100% 100%;
float:right;
margin-top:400px;
margin-left:600px;
transition:0.2s;
}

.ventil
{
width:200px;
height:200px;
margin-left:0px;
background-image:url('ventil.png');
background-size:100% 100%; z-index:200;
transition:0.2s;
position:absolute;
}

.ventil2
{
width:200px;
height:200px;
margin-left:30px;
background-image:url('ventil.png');
background-size:100% 100%;
z-index:200;
transition:0.2s;
position:absolute;
}
```


POPIS SLIKA

Slika 1. Vodosprema	1
Slika 2. Prikaz korištenog softvera u razvojnom okruženju.....	2
Slika 3. Uvoženje dodatnih alata.....	3
Slika 4. Prikaz korištenog toolkit-a	3
Slika 5. Prikaz korištenog toolkit-a	3
Slika 6. Nadogradnja	4
Slika 7. Prikaz strukture baze podataka u phpmyadmin-u	8
Slika 8. Prikaz simulatorske aplikacije	8
Slika 9. Ispis stanja tanka u konzoli aplikacije.....	10
Slika 10. Prikaz alarma.....	10
Slika 11. ispis stanja punog tanka	11
Slika 12. Ispis stanja tanka "Visoko"	11
Slika 13. Prikaz nadzorne aplikacije nakon otvaranja.....	17
Slika 14. Prikaz upita prema aplikaciji.....	18
Slika 15. Prikaz po etnog stanja pumpe.....	18