

**VELEUČILIŠTE U RIJECI**

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

**Projekt SVPP\_2015\_4 - Vodosprema  
(projektna dokumentacija)**

**Projektni tim:** Nives Miletić, Vedrana Ružić, Neven Frković

Rijeka, prosinac 2015.

## SAŽETAK

Prema zadanome projektnom zadatku, izrađen je projekt koji se sastoji od nadzorne i simulatorske aplikacije koje prikazuju upravljanje proizvodnim procesom sustava „Vodosprema“.

**Ključne riječi:** Aplikacija; Ventil; Senzor; Pumpa;

## SADRŽAJ

<b>1. UVOD.....</b>	<b>4</b>
<b>2. IZRADA APLIKACIJE .....</b>	<b>5</b>
<b>3. PROGRAMSKI DIO APLIKACIJE .....</b>	<b>7</b>
<b>3.1. Klase.....</b>	<b>7</b>
<b>3.2. Senzor .....</b>	<b>10</b>
<b>3.3. Ventil.....</b>	<b>13</b>
<b>3.4. Pumpa .....</b>	<b>14</b>
<b>4. GRAFIČKI DIO APLIKACIJE .....</b>	<b>15</b>

## 1. UVOD

Za rješenje projektnog zadatka bilo je potrebno izraditi dvije vrste aplikacija. One su navedene nadalje:

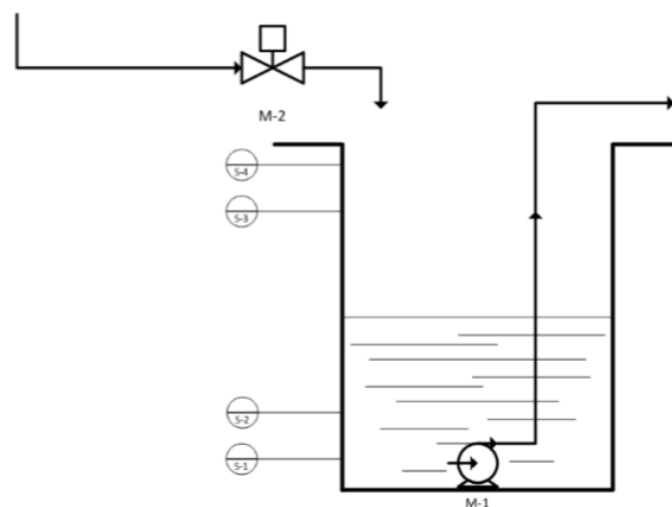
- 1) Simulatorska aplikacija – simulira proizvodni sustav
- 2) Nadzorna aplikacija – prima informacije iz proizvodnog sustava

Između navedenih aplikacija osposobljena je komunikacija kao što je i bilo zadano za projektni zadatak.

Simulatorska aplikacija je izgrađena web tehnologijama (HTML5, PHP AJAX, CSS, JAVASCRIPT). Iz simulatora se definira i dolazak vode u preko ventila na cijevi za dovod vode.

Nadzorna aplikacija izgrađena pomoću alata „ECLIPSE“ te programske platforme Java i njenih tehnika za razvoj. Komunikacija je omogućena online bazom podataka. Voda se crpi po aktivaciji iz nadzorne aplikacije. Prikazuje stanja na grafičkoj pozadini paljenjem lampica ili animacijom. **Omogućuje ručno aktiviranje motora i ventila kad je simulacija zaustavljena.**

Za projektni zadatak bilo je potrebno obraditi i prikazati proces dotoka tekućine u spremnik koji je kontroliran radom ventila, a istjecanje tekućine iz njega je slobodno regulacijom rada pumpe, kako je to prikazano na sljedećoj slici. Izrada ovog zadatka biti će prikazana po odabranim cjelinama razrade.

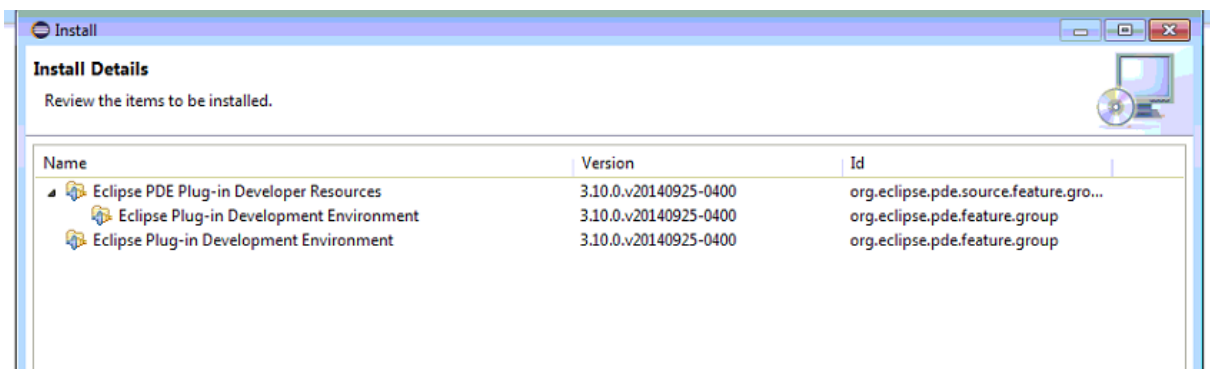


*Slika 1. Vodosprema*

## 2. IZRADA APLIKACIJE

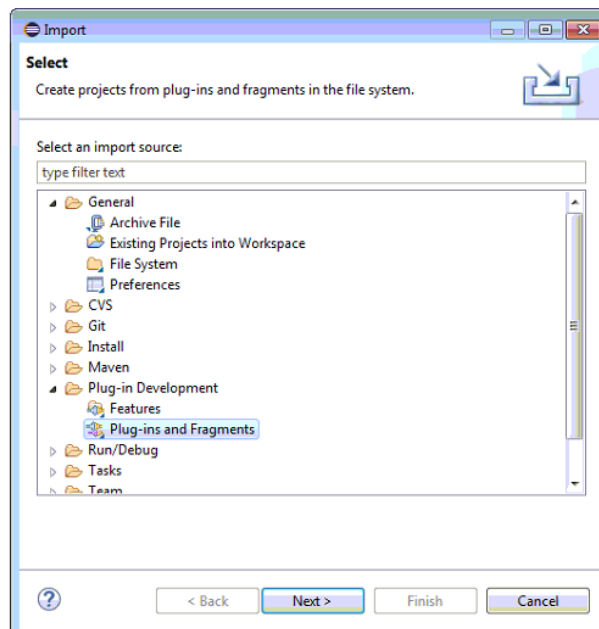
Prije početka izrade aplikacije bilo je potrebno postaviti i podesiti nekoliko dodatnih opcija u samom razvojnom sučelju platforme „Eclipse“.

Najprije je bilo potrebno omogućiti korištenje tzv. *plug-in* mogućnosti za nadogradnju razvojnog sučelja, a isto je moguće dobiti putem platforme. Sljedeća slika prikazuje instalirane dodatke za naše razvojno okruženje.



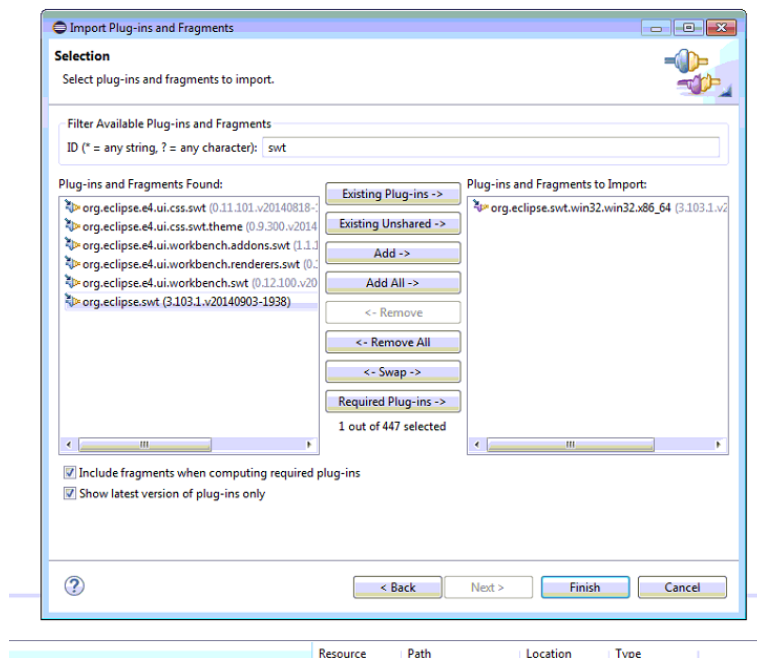
Slika 2. Prikaz korištenog softvera u razvojnom okruženju

Isti programskim paketom omogućeno je korištenje i uvoženje dodatnih fragmenata programa korištenih za razvoj aplikacije.



Slika 3. Uvoženje dodatnih alata

Nadalje je bilo potrebno odabrati odgovarajući SWT (eng. *Standard Widget Toolkit*) paket alata koji prilikom razvoja korisniku omogućuju korištenje i interakciju s grafičkim objektima aplikacije, a odgovarajući *toolkit* odabiremo u odnosu na operativni sustav za koji su namijenjeni.



Slika 4. Prikaz korištenog toolkit-a

Dodatno je bilo potrebno omogućiti i povezivanje s bazom podataka, a isto je



omogućeno s JAR (eng. Java ARchive) paketom „mysql-connector-java-5.1.34-bin.jar“.

Sve o konekciji s bazom podataka pisati će u daljnjem djelu dokumentacije. Naknadno se za sami razvojni prostor postavljaju putanje alata za nadogradnju razvojnog sučelja, a isto je moguće vidjeti sljedećom slikom. Podešavanjem istih postavka definiramo točne prostore smještaja nadogradnji. Isto je važno kako bi aplikacija mogla biti pravilno razvijana, a u slučaju da nam nešto od navedenog nedostaje, pokretanje aplikacije putem razvojnog sučelja neće biti moguće (tada se posljedično javlja greška o nedostatku potrebnih nadogradnji).

### 3. PROGRAMSKI DIO APLIKACIJE

#### 3.1. Klase

Izrada aplikacije započeta je u odnosu na zadani zadatak projekta „Vodosprema“. U početku su definirane klase „Senzor“, „Tank“, „Motor“, „Alarm“. Svaka od tih klasa u aplikaciji sastoji se od nekoliko funkcija koje su uglavnom i svojstvena obilježja funkcije klase.

U klasi „Tank“ s njenim privatnim objektom „razina“ omogućeno je generiranje *get* i *set* metoda za pozivanje funkcija „getRazina()“ i „setRazina“. Klasa se nalazi u određenom programskom paketu „model“. Iste funkcije omogućuju nam korištenje metoda 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“ nalazi se objekt „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 njena privatna varijabla kao objekt „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“ stvoren je objekt „aktiviran“ čija je početna vrijednost *false*. S funkcijom „setAktiviran“ definiran je tip objekta koji valja proslijediti kada prilikom korištenja metoda koje se odnose na razinu vode, odnosno, stanje u spremniku. 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 funkcija „pooling“ nudi mogućnost pozivanja metode „getRazina“ koja se odnosi na objekt „spremnik“. 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 datoteku
- 2) Spremanje vrijednosti u online bazu podataka – *online* pohrana u BP

Spremanje vrijednosti u datoteku omogućeno je iz razloga što je jedan dio aplikacije napravljen kao desktop aplikacija, dok je drugi dio aplikacije napravljen kao online aplikacija koja sprema isti podatak u bazu.



Nadalje se nalazi primjer koda strukture klase „Datotekica“ koja sadrži blokove naredba za funkciju pohrane vrijednosti.

```
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 klasa „UcitajDat“.

```
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 serveru u korištena je klasa „UcitajizBaze“.

```
public void UcitajizBaze(){
    try {
        Class.forName("com.mysql.jdbc.Driver");
        Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/vodosprema","root"," ");
        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();
    }
}

```

### 3.2. 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 našem slučaju napravljena je 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 tipa te pomoću objekta „razinaTek“ koji je svojstva klase „Tank“, u predviđenim petljama detektiraju slijedeće vrijednosti:

- 1) Senzor *vrloNisko* – detektira razinu vode u rasponu od  $\leq 50$  i  $\geq 10$
- 2) Senzor *nisko* – detektira razinu vode u rasponu od  $\leq 150$  i  $\geq 60$
- 3) Senzor *visoko* – detektira razinu vode u rasponu od  $\leq 440$  i  $\geq 350$
- 4) Senzor *vrloVisoko* – detektira razinu vode u rasponu od  $\geq 450$  i  $\leq 490$

Pomoću objekta „razinaTek“ u predviđenim *if* petljama postavljeni su i objekti „stanjeSpremnika“ i „stanje“, a oni ispisuju još i sljedeće razine vode u spremniku:

- „stanjeSpremnika“ očitava razinu „Srednje“, a tada se obj. „stanja“ ispisuje „f,f,f,f“

Primjer koda u kojemu su definirane varijable za ispis i provjeru s tanja prikazane su nadalje.

```

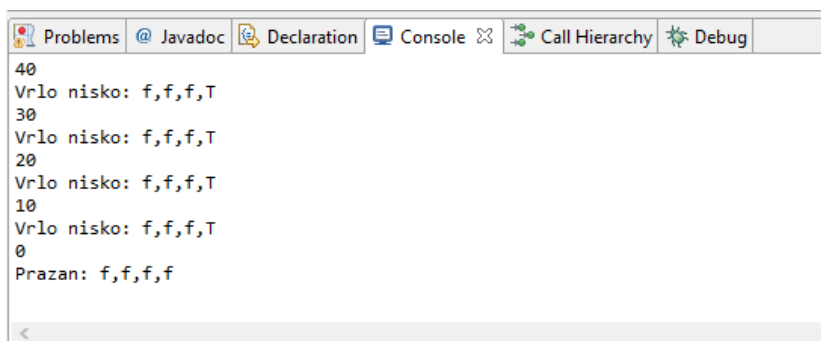
int razinaTek=spremnik.getRazina();
String stanja= " ";
String stanjeSpremnika= " ";

```

Napravljena je i dodatna *if* petlja koja pomoću objekta „razinaTek“ pod uvjetom da je njegova vrijednost  $\leq 0$ , aktivira alarm metodama *prazan.setAktiviran(true)* i *pun.setAktiviran(true)* ispisuju se slijedeće prilagođene vrijednosti:

- *prazan.setAktiviran(true)* - ispisuje „Prazan: f,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“

Nadalje je prikazana slika ispisa istih poruka u programskoj konzoli.



```

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

```

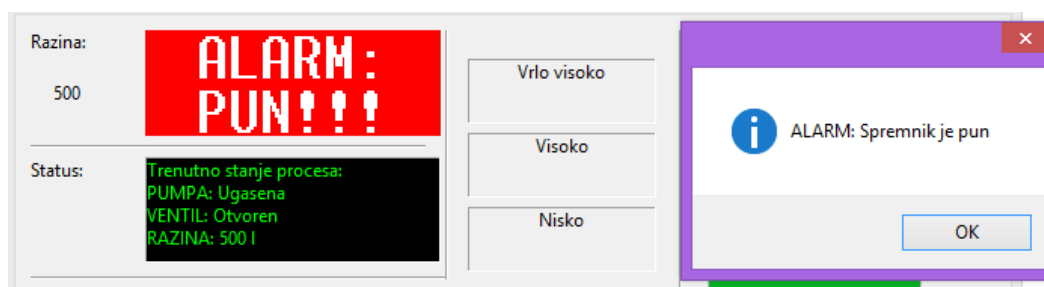
Slika 5. Ispis stanja tanka u konzoli aplikacije

Ovako to izgleda realno u aplikaciji:



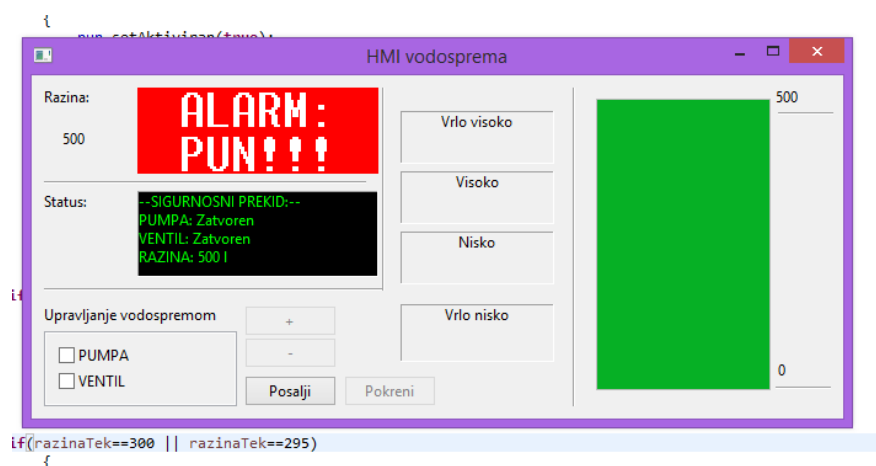
Slika 6. Prikaz alarma

Nadalje je prikazana slika aplikacije te prozora koji prikazuje informaciju o stanju tanka. Tada se i u konzoli aplikacije ispisuje stanje tanka putem *POP-UP* prozora.



Slika 7. Ispis stanja punog tanka

Na sljedećoj slici je prikazano stanje senzora u slučaju kada je tank pun. Alarm je u tom stanju upaljen, svi senzori ugašeni, a ventil za dovod vode je zatvoren.



Slika 8. Prikaz alarmantnog stanja



Slika 9. Ispis stanja tanka "Visoko"

Na posljednjoj prethodnoj slici nalaze se senzori pomoću kojih je vidljivo stanje razine vode u spremniku. Što se tiče grafičkog djela aplikacije za prikaz istog korišteni su sljedeći elementi:

- 1) Button btnPokreni – gumb za pokretanje aplikacije
- 2) final Label lblAlarm – labela za prikazivanje alarma. Za prikaz su korištena sljedeća svojstva:
- 3) Label lblRazina\_1 – grafički element za ispis labela „Razina“

**final** Label lblStatus – labela za prikazivanje statusa:

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

4) final Label lblVrloVisoko, lblVisoko, lblNisko, lblVrloNisko - grafički prikaz stanja u tanku. Njihov prikaz ovisi o postavljenim uvjetima za čitanje razine.

5) lblStatus – ispisuje trenutno stanje procesa, a primjer koda je:

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

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

### 3.3. Ventil

Ventil je u simulatorskoj 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 jednostavne *if* zadan je uvjet koji se odnosi na *true* stanje ventila, kao upaljeno stanje i tada je u kodu obrađeno povećavanje razine tekućine (razina se povećava za vrijednost 10). Primjer koda prikazan je nadalje.

```
public void otvoriVentil(boolean stanjeVentila)
{
    if(stanjeVentila == true)
    {
        int razinaTek=spremnik.getRazina();
        razinaTek=razinaTek+10;
        spremnik.setRazina(razinaTek);
        ventil.setOtvoren_upaljena(true);
        /*Metoda otvoriVentil argument je stanjeVentila on se provjerava
        ako je njegova vrijednost true(ako je ventil aktiviran) uzima se vrijednost
        razine objekta spremnik, poveca se za 10 i povecana vrijednost sprema se natrag
        atributu razina objektu spremnik i postavlja se atributu otvoren_upaljena objekta
        ventil vrijednost na true u smislu da je ventil otvoren
        */
    }
}
```

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 isto 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);
    /*Metoda upaljenoOtvoreno izvršava se kada je ventil otvoren a pumpa
    upaljena * uzima se vrijednost razine objekta spremnik poveća se za 6 i vraća
    natrag, stanje je postavljeno na true*/
}

```

### 3.4. Pumpa

Metoda „upaliPumpu“ radi pod uvjetom da je varijabla „stanjePumpe“ postavljena na *true*. Tada se pomoću objekta „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);
        /*Funkcija upaliPumpu argument je stanjePumpe on se provjerava ako je
        njegova vrijednost true(ako je pumpa aktivirana) uzima se vrijednost razine
        objekta spremnik umanjuje se za 10 i umanjena vrijednost se ponovno prema objektu
        spremnik i njegovom atributu razina atribut Otvoren_upaljena postavlja se na true
        kao dojava stanja da pumpa radi razina-10*/
    }
}

```

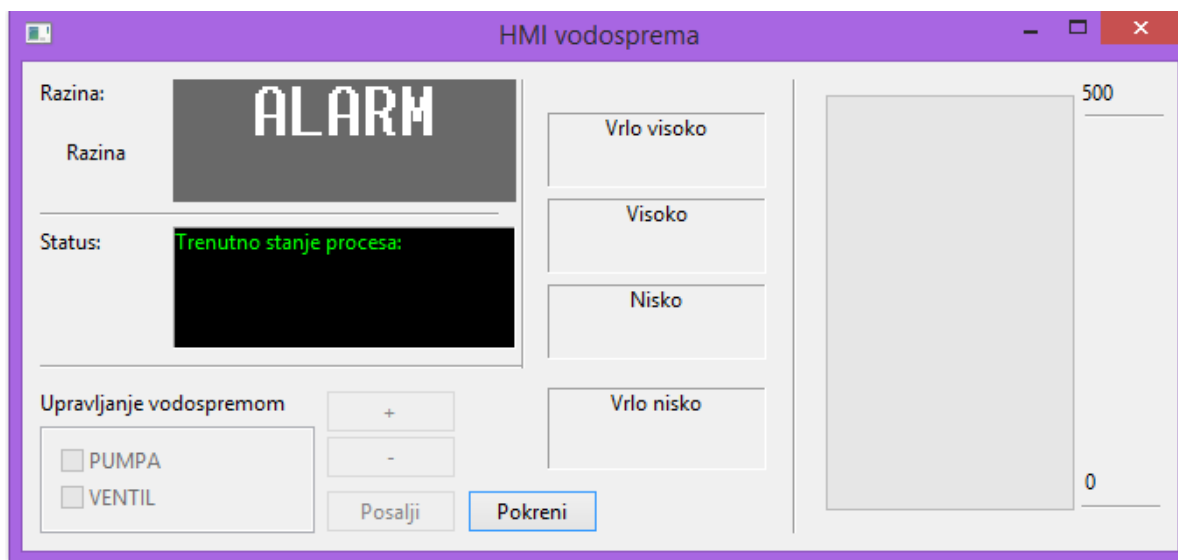
## 4. GRAFIČKI DIO APLIKACIJE

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 način da na *root* mapu projekta odaberemo opciju „*Debug As → Java Application*“. Slijedi slikoviti prikaz rezultata izvršenog koda aplikacije:



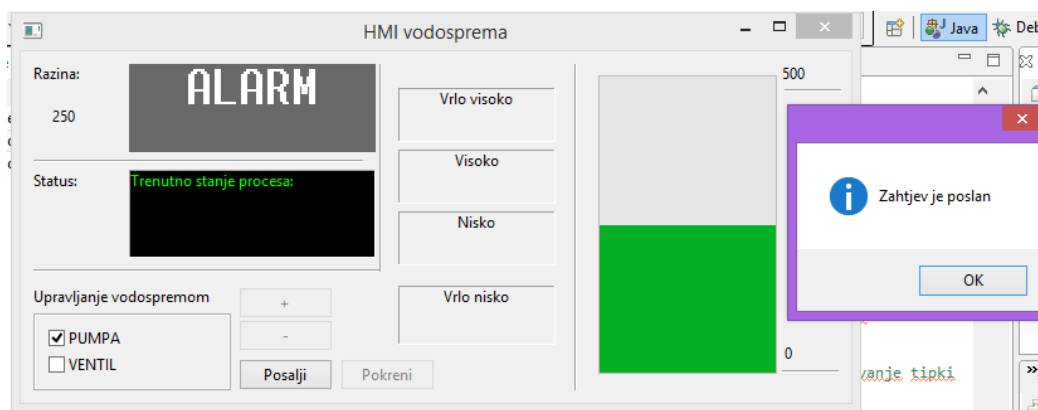
Slika 10. Izvršena aplikacija

Slika prikazuje izvršenu aplikaciju, ali ne i pokrenutu. Istu pokrećemo pomoću jednostavnog gumba „Pokreni“ koji je na slici jasno vidljiv. Isto tako valja spomenuti da je tek tada omogućena interakcija s ostalim elementima aplikacije.

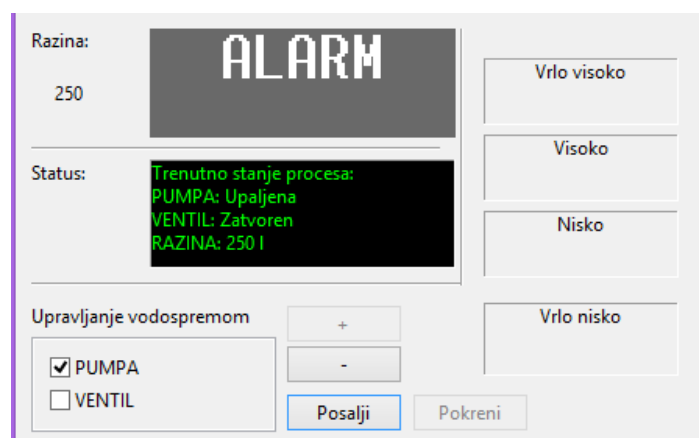
Za grafičke elemente, u odnosu na korištenje određenih mogućnosti aplikacije (npr. omogućavanje rada pumpe), postavljeni su neki od uvjeta za prikaz odgovora aplikacije prema upitu. Slijedi primjer koda koji definira način rada i korištenje pumpe u aplikaciji:

```
String razina=String.valueOf(razinaI);
pumpa=btnPumpa.getSelection();
//varijable stanja pumpe ventila i razine -> ispis statusa
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 11. Prikaz upita prejma aplikaciji



Slika 12. Prikaz početnog stanja pumpe



U ovom slučaju je moguće korištenje pumpe u aplikaciji. Tada je moguće koristiti i gumb „-“ koji u praksi odgovara radu pumpe. Na isti način je 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. On npr. Registrira i omogućuje paralelni rad pumpe i ventila u 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 čime je upravljati s razinom vode u spremniku, dok je pumpa upaljena, a ventil otvoren.

Valja spomenuti još i glavnu uporabu istog objekta u aplikaciji:

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

Takvom metodom omogućeno je očitavanje razine i stanja spremnika, što je već prije objašnjeno u ovoj projektnoj dokumentaciji.

## POPIS SLIKA

Slika 1. Vodosprema .....	4
Slika 2. Prikaz korištenog softvera u razvojnom okruženju.....	5
Slika 3. Uvoženje dodatnih alata.....	5
Slika 4. Prikaz korištenog toolkit-a .....	6
Slika 5. Ispis stanja tanka u konzoli aplikacije.....	11
Slika 6. Prikaz alarma.....	11
Slika 7. ispis stanja punog tanka .....	12
Slika 8. Prikaz alarmantnog stanja .....	12
Slika 10. Ispis stanja tanka "Visoko" .....	12
Slika 11. Izvršena aplikacija.....	15
Slika 12. Prikaz upita prejma aplikaciji .....	16
Slika 13. Prikaz početnog stanja pumpe.....	16