Interaktivni nadzorni paneli

Đurin, Adrijan

Master's thesis / Diplomski rad

2016

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:333296

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-29**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

zir.nsk.hr

Image not found or type unknown

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

INTERAKTVINI NADZORNI PANELI

Diplomski rad

Adrijan Đurin

Osijek, 2016.

ETFOS
ELEKTROTEHNIČKI FAKULTET OSIJEK
Sveučilište Josipa Jurja Strossmayera u Osijeku

Obrazac D1: Obrazac za imenovanje Povjerenstva za obranu diplomskog rada

Osijek, 06.07.2016.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za obranu diplomskog rada

Ime i prezime studenta:	Adrijan Đurin
Studij, smjer:	Diplomski sveučilišni studij Računarstvo, smjer Procesno računarstvo
Mat. br. studenta, godina upisa:	D-610R, 23.10.2013.
Mentor:	Doc.dr.sc. Josip Job
Sumentor:	
Predsjednik Povjerenstva:	Doc.dr.sc. Ratko Grbić
Član Povjerenstva:	Doc.dr.sc. Emmanuel-Karlo Nyarko
Naslov diplomskog rada:	Interaktivni nadzorni paneli
Znanstvena grana rada:	**Informacijski sustavi (zn. polje računarstvo)**
Zadatak diplomskog rada:	Izraditi web aplikaciju za izradu i upravljanje interaktivnim nadzornim panelima primjenom AngularJS, D3.JS, HTML5, CSS3 i JavaScript.
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 2 Postignuti rezultati u odnosu na složenost zadatka: 2 Jasnoća pismenog izražavanja: 2 Razina samostalnosti: 3
Datum prijedloga ocjene mentora:	06.07.2016.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

ELEKTROTEHNIČKI FAKULTET OSIJEK
Sveučilište Josipa Jurja Strossmayera u Osijeku

IZJAVA O ORIGINALNOSTI RADA

Osijek, 11.07.2016.

Ime i prezime studenta:	Adrijan Đurin
Studij:	Diplomski sveučilišni studij Računarstvo, smjer Procesno računarstvo
Mat. br. studenta, godina upisa:	D-610R, 23.10.2013.
Ephorus podudaranje [%]:	3

Ovom izjavom izjavljujem da je rad pod nazivom: **Interaktivni nadzorni paneli**

izrađen pod vodstvom mentora Doc.dr.sc. Josip Job

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD 1

2. TEORIJSKA POZADINA RAZVOJA VIZUALNOG PANELA 3

2.1. Digitalni interaktivni vizualni paneli 4

2.2. Internet tehnologije 7

2.2.1. Google Cloud Datastore 9

2.2.2. AngularJS 10

2.2.3. D3.js 12

3. REALIZACIJA RAZVOJA VIZUALNOG PANELA 13

3.1. Početna stranica 14

3.1.1. Krajnje točke oblaka i info predložak 16

3.2. Linijski grafovi 20

3.3. Upravljanje navodnjavanjem 28

4. ZAKLJUČAK 30

LITERATURA 31

SAŽETAK 32

ABSTRACT – WEB DASHBOARDS 32

ŽIVOTOPIS 33

1. UVOD

Današnji svijet je okarakteriziran ubrzanim razvojem informacijskih tehnologija koje prodiru u sve pore čovjekovog djelovanja. Teško je nabrojati ljudske djelatnosti koje ne uključuju uporabu računala i Interneta. Pristup raznovrsnim informacijama je nadohvat ruke, a njihova pretraga je olakšana i ubrzana. Sve to omogućuje da korisnici određene radnje i zadatke povjere specijaliziranim programima koji će odrađivati dio posla, a sve kako bi uštedjeli vrijeme. Time uloga korisnika postaje nadgledanje rezultata koje prikazuju specijalizirani programi. Nerijetko su ti izlazi prikazi vrijednosti u stvarnom vremenu, pa ih je stoga potrebno formatirati kako bi korisnik imao lakši uvid u stanje nadgledanog sustava. U ovom radu biti će riječi o vizualizaciji podataka koje je potrebno što točnije i vjernije prikazati krajnjem korisniku jer korisnikove buduće akcije ovise o istinitosti podataka i njihovom prikazu. Podaci i njihove vrijednosti se prikupljaju senzorima, obrađuju i zatim prosljeđuju na uslugu spremanja podataka na računalnom oblaku. Korisnik u svakom trenutku zahtjeva točne i ažurirane podatke. Jedan od zahtjeva je i da se uvid u podatke može dobiti i pristupom s udaljenih lokacija kao i s različitih uređaja. Svi navedeni zahtjevi spadaju u domenu Interneta objekata (engl. Internet of Things) kojem je općeprihvaćena definicija da je to skup fizičkih objekata, uređaja, vozila i zgrada, u koje je ugrađena elektronika, senzori i programi. Temeljna odlika je i mogućnost povezivanja na mrežu koja prikupljene i djelomično obrađene podatke prosljeđuje drugim dijelovima sustava.

U daljnjem tekstu bit će prikazan jedan dio takvog sustava. Točnije, bit će riječi o interaktivnom vizualnom panelu. To je dio sustava s kojim će korisnik biti svakodnevno u doticaju. Sustav se sastoji od nekoliko dijelova: elektronike sa senzorima i aktuatorima, mreže koja prosljeđuje podatke na računalni oblak i aplikacije koja te podatke koristi i daje im vrijednost i značenje pri vizualizaciji. Aplikacijom će se moći i upravljati funkcijama kojima raspolaže plastenik. Postojanje navedenih dijelova sustava, pored aplikacije, se podrazumijeva i oni neće biti opširno pokriveni u ovom radu.

Za razvoj aplikacije koristit će se standardne Internet tehnologije, HTML i CSS. Za razvoj vizualizacije bit će korišten D3.js (engl. Data-Driven Documents). Pristup razvoju aplikacije će biti MVC (engl. model-view-controller) model (u daljnjem tekstu: MVC) gdje će biti korišten AngularJS, Internet aplikacijski okvir kojeg održava Google i zajednica individualaca jer je kod otvorenog tipa (engl. open-source).

Cilj je kroz ovaj rad razviti aplikaciju korištenjem gore navedenih tehnologija koja će davati

trenutni uvid u stanje sustava, upozoriti korisnika ukoliko neka vrijednost prekorači alarmantnu vrijednost i omogućiti korisniku minimalno upravljanje.

U prvom poglavlju bit će ukratko objašnjen cijeli sustav i način njegova rada kako bi se imao lakši uvid u kasniji rad aplikacije. Ukratko će biti objašnjene i tehnologije koje će se koristiti za daljnji razvoj aplikacije. U drugom poglavlju fokus će biti samo na aplikaciji i njezinom razvoju. Detaljno će se prikazati način rada aplikacije upotpunjene s dijelovima koda.

2. TEORIJSKA POZADINA RAZVOJA VIZUALNOG PANELA

Živimo u svijetu u kojem informacijska tehnologija igra značajnu ulogu. Teško je nabrojati društvene djelatnosti koje u nekom dijelu proizvodnog procesa ili sustava ne koriste računala, razne programe i aplikacije za pametne telefone i pristup Internetu. Problem koji se javlja jest kako izdvojiti korisne informacije iz velike količine prikupljenih podataka, koji ponekad sežu i do nekoliko terabajta, te kako ih prikazati na prihvatljiv način. Bitnu ulogu u procesu zaključivanja imaju i vizualni paneli (engl. dashboards).

U informacijskoj tehnologiji, paneli predstavljaju vizualno korisničko sučelje koje svrstava i prikazuje informacije korisniku na jednostavan i prihvatljiv način. Drugim riječima predstavljaju vizualna sučelja koja su jednostavna za razumjeti, često jednostranična i u stvarnom vremenu prikazuju stanje sustava. Svrha im je korisniku olakšati praćenje ključnih veličina vezanih za određene usluge, poslove ili ciljeve. Primjerice, broj prodanih automobila kroz tjedan, mjesec ili godinu dana na jednoj razini. Na menadžerskoj razini, dobiva se uvid u broj prodanih automobila po poslovnicama ili regijama. Sve to olakšava donošenje daljnjih odluka. Područje upotrebe i korištenja vizualnih panela se proširilo na sve ljudske djelatnosti. Odlike dobrog vizualnog panela su:

-jednostavnost i razumljivost
-prikaz bitnih informacija, minimum distrakcija
-informacije i podaci svrstani i uređeni po značenju i korisnosti
-naglasak na vizualnim elementima.

S obzirom na ulogu, panele možemo svrstati u četiri kategorije: strateški, analitički, operativni i informacijski. Strateški paneli pomažu menadžerima na svim razinama u nekoj organizaciji. Daju na uvid sve bitne informacije koje pomažu pri donošenju odluka vezanih za predviđanje poslovanja. Prikaz informacija je statički i mijenja se jednom u određenim intervalima, dnevno, tjedno ili mjesečno, ovisno o djelatnosti koja se nadgleda.

Analitički paneli pružaju više informacija, veći uvid u prošlost kretanja ključnih vrijednosti i usporedbe među informacijama (Sl. 2.1.). Operativni paneli su slični strateškima. Također daju uvid u bitne i relevantne informacije, ali s naglaskom na veličine koje se konstantno mijenjaju i koje zahtijevaju pažnju i moguću reakciju ukoliko ozbiljnost situacije to zahtjeva [1]. Informacijski paneli predstavljaju vrstu panela gdje se promjena informacija i njezin prikaz ne mijenjaju često i gdje ako i dođe do promijene, posljedice nisu velike.



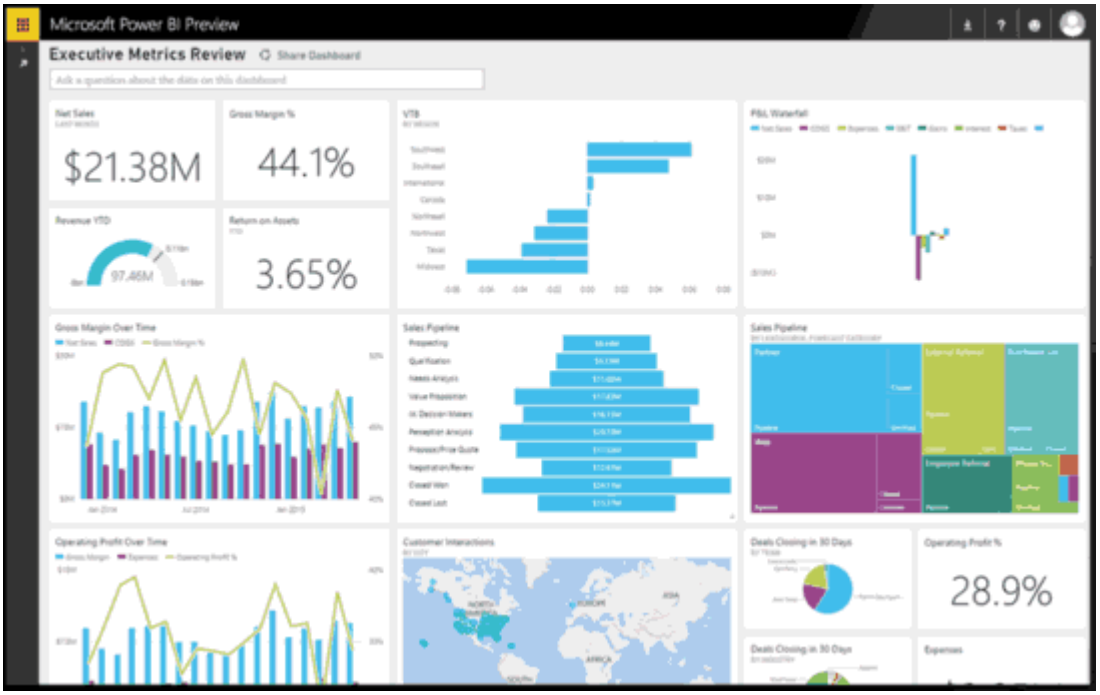
Sl. 2.1. *Primjer analitičkog vizualnog panela*

Paneli su u velikom broj slučaja jedinstveni jer uvelike ovise o osnovnoj djelatnosti na koju se nadograđuju. Njihov dizajn predstavlja niz grafova, dijagrama i tablica te drugih vizualnih indikatora koji se mogu pratiti i na lagan način interpretirati [2].

2.1. Digitalni interaktivni vizualni paneli

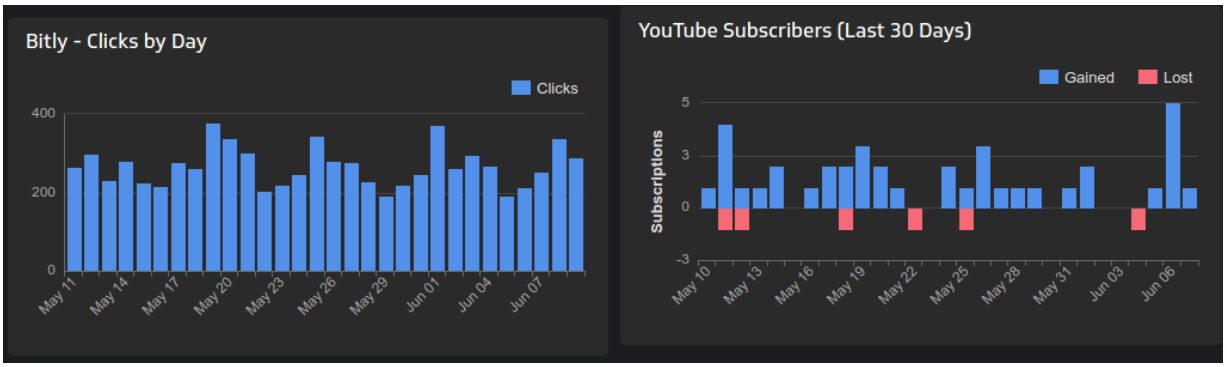
Pojam digitalnih interaktivnih vizualnih panela vezuje se uz menadžment informacijskih sustava čiji je zadatak upravljanje informacijama i informacijskim sustavima radi učinkovitijeg donošenja odluka. Svrha korištenja panela je olakšano praćenje svih ključnih vrijednosti vezanih za djelatnost koja se nadgleda, a cilj je olakšati i pospješiti donošenje bitnih strateških odluka. U osnovi, paneli su specijalizirani programi i aplikacije kojima se olakšava uvid u stanje sustava ili djelatnosti. To se može odnositi kako na cijele korporacije koje prate ukupnu prodaju i zadovoljstvo korisnika, tako i na male poduzetnike koji prate stanje na društvenim mrežama [3].
Razlikuju se tri vrste digitalnih panela: samostalne aplikacije za specifične sustave, aplikacije temeljene na Internet tehnologijama i jednostavne aplikacije dostupne za prijenosna i stolna računala i pametne telefone (engl. Widgets).
Specijalizirani paneli predstavljaju aplikacije koje su alat za vizualizaciju podataka trenutnog stanja ključnih indikatora sustava. Bitno svojstvo im je prilagodljivo sučelje i mogućnost prikaza

podataka u stvarnom vremenu. Oracle i Microsoft su među najpoznatijim dobavljačima poslovnih panela (engl. business intelligence dashboards). Slika 2.2. prikazuje primjer sučelja poslovnog panela. Interaktivnost se postiže mogućnošću detaljnog uvida u pojedine stavke kao i u povijest kretanja mjerenih vrijednosti.



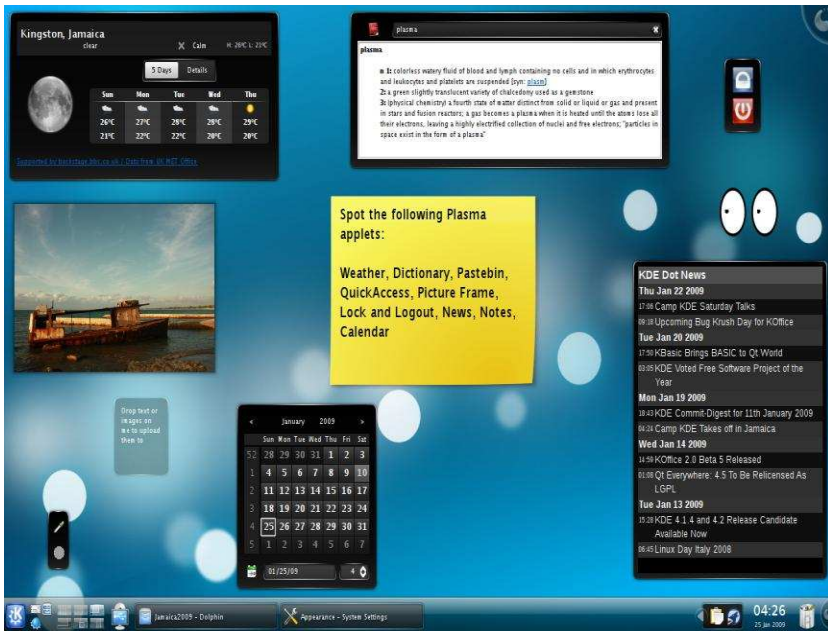
Sl. 2.2. *MS Power BI sučelje*

Paneli zasnovani na Internet tehnologijama su aplikacije temeljene na tehnologijama koje se koriste za Internet programiranje (HTML, JS, CSS). Samim time prilagodljivi su za sve uređaje koji podržavaju Internet preglednike i njihova je uporaba široko rasprostranjena. Na Internetu postoje mnogi servisi koji nude uslugu prikaza gotovih panela. Potrebno je samo pružiti podatke koji se žele prikazati u nekom od oblika koji su podržani, a zatim programska pozadina sama određuje vizualizaciju veličina. Ti servisi većinom nisu besplatni, niti se ima uvid u sam programski kod. Slika 2.3. prikazuje primjer jednog dijela vizualnog panela koji je nastao korištenjem servisa Klipfolio.



Sl. 2.3. Interaktivni grafovi servisa Klipfolio

Dodatne informacije je moguće dobiti prelaskom miša preko vrijednosti na grafu. Interaktivnost se postiže i mogućnošću klika mišem na legendu i odabirom koje se vrijednosti žele prikazati. Time se korisniku prikazuju pozitivni i negativni trendovi, što olakšava zaključivanje i donošenje budućih odluka.



Sl. 2.4. Widgeti u GNU/Linux Kubuntu distribuciji

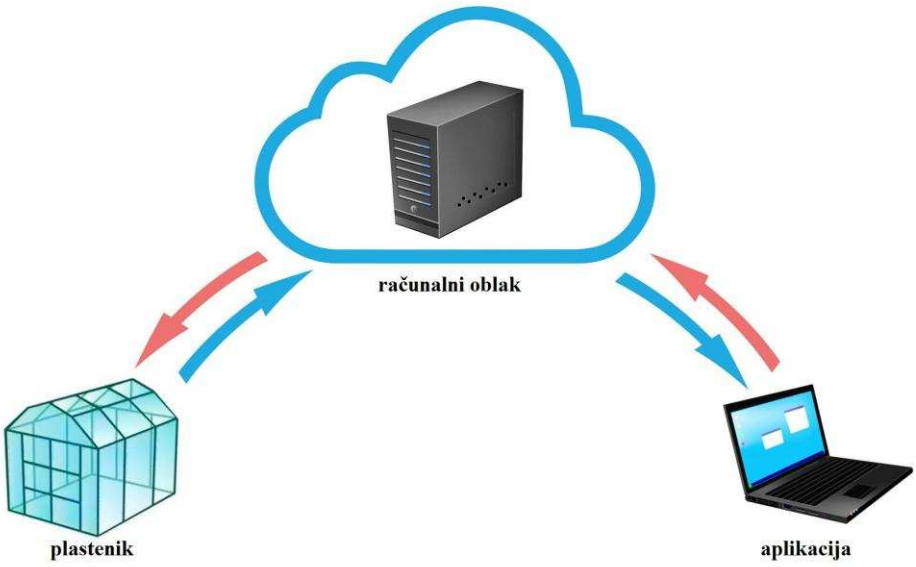
Widget predstavlja element grafičkog korisničkog sučelja koji prikazuje informacije i omogućuje korisniku specifičan način interakcije s operacijskim sustavom ili aplikacijom. To je jednostavan i interaktivni program koji često ima samo jednu namjenu, npr. prikaz kalendara, kalkulatora, bilješki, količine potrošnje procesora i radne memorije. Widgeti su razvijeni za sve računalne i mobilne operacijske sustave (Sl. 2.4.). Smješteni su na pozadini grafičkog sučelja i koriste male

količine resursa. Svrha im je prikazati relevantne informacije koje korisnik zahtjeva na dnevnoj bazi na nenametljiv način.

Primarna zadaća razvoja panela u sustavu nadzora plastenika bit će trenutni prikaz izmjerenih vrijednosti unutar plastenika. Ukoliko neka vrijednost prekorači unaprijed postavljene granice, panel će vizualno alarmirati korisnika promjenom boje. Dodatna razina interaktivnosti bit će prikaz detaljnijeg kretanja mjerenih vrijednosti. Klikom na graf prikazat će se trenutna izmjerena vrijednost u danom trenutku.

2.2. Internet tehnologije

U ovom će potpoglavlju biti ukratko riječi o pojedinim dijelovima cijelog sustava, kako bi se imao cjelovitiji uvid u razvoj aplikacije. Sustav je podijeljen na tri dijela, kao što je vidljivo na slici 2.5.



Sl. 2.5. Shema sustava

U dijelu sustava u kojem se nalazi plastenik nalaze se potrebni senzori koji provode mjerenja u određenim vremenskim intervalima. To su senzori za vanjsku i unutarnju temperaturu, senzori za vlažnost zemlje i zraka te senzor koji mjeri protok vode. Aktuator je ventil za vodu. Postoje dva načina kojima se upravlja ventilom. Jedan je taj da se u Raspberry skripti definira kritična

vlažnost zemlje nakon koje se ventil sam pali i propušta vodu dok se vlažnost ne poveća. Drugi je način da ga korisnik sam iz aplikacije proizvoljno upali, odnosno ugasi. Zadaća Arduina je da upravlja senzorima i aktuatorima i da prikuplja podatke koje senzori očitaju. Te podatke prosljeđuje Raspberryu koji žičanom vezom ima pristup Internetu i Google Cloud usluzi gdje se podaci pohranjuju. Također, Raspberry pri zapisu podataka u oblak ujedno provjerava postoje li kakve promjene u načinu navodnjavanja koje je zadao korisnik. Sustav kroz stalnu provjeru senzora u plasteniku nadzire stanje i usklađuje ga prema unaprijed definiranim korisničkim postavkama [4]. Pored samog prikaza i vizualizacije podataka potrebno je nekoliko riječi posvetit i tehnologijama koje će se koristit pri razvoju aplikacije.

O dijelu sustava u kojem se koriste Arduino sa senzorima i aktuatorima te Raspberry, neće biti previše riječi. Za razvoj aplikacije bitno je da su funkcionalni i da obavljaju zadaću prosljeđivanja podataka. Segment koji je od veće važnosti je pohrana dobivenih podataka na Google Cloud uslugu koji je centralni dio sustava. Preko njega se vrši komunikacija između plastenika na jednom kraju i korisnika, putem aplikacije, na drugom kraju. Aplikacija će redovno vršiti komunikaciju s oblakom kako bi uvijek bila ažurirana i kako bi korisnik uvijek imao osvježene podatke. Kao podlogu za razvoj bit će korišten AngularJS okvir za razvoj Internet aplikacija. AngularJS se koristi za razvoj aplikacija na klijentskoj strani. Za prikaz trenutnih razina temperature i vlažnosti, kao i količine protečene vode koristit će se D3.js, JavaScript biblioteka za razvoj dinamičkih, interaktivnih vizualizacija.

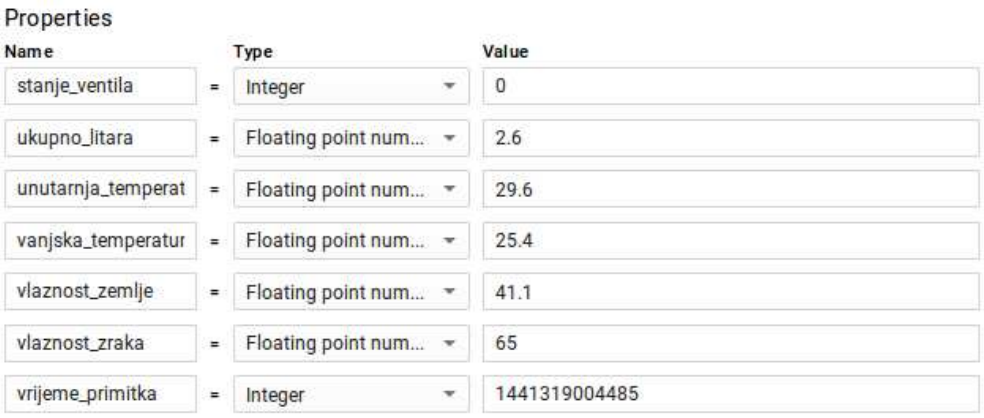
Zamisao je da se kroz pozadinske procese aplikacije redovno vrši ažuriranje podataka Google Cloud uslugom. AngularJS bi zatim te podatke dalje proslijedio dijelu koda koji je zadužen za vizualizaciju i prikaz. Na početnoj stranici korisnik će imati trenutni uvid u stanje sustava gdje će biti prikazani najsvježiji podaci. Ukoliko neka od vrijednosti odstupa od dopuštene, alarm će se oglasiti i korisnik će biti upozoren. Na sljedećoj stranici kroz linijske grafove bit će prikazane sve promijene i oscilacije mjerenih vrijednosti kroz duže vremensko razdoblje. Bit je omogućiti korisniku uvid u stanje sustava preko noći. Briga za plastenik iziskuje svakodnevni posjet i po nekoliko puta radi zalijevanja, provjere stanja vlažnosti i temperature. Korištenjem aplikacije taj broj posjeta bi se smanjio i time bi se znatno uštedjelo na vremenu koje korisnik može utrošiti na druge aktivnosti. Na zadnjoj stranici postojat će kratki opis svrhe, načina i rada aplikacije.

U sljedećim odlomcima kratko će biti opisane specifične tehnologije koje će se koristiti za razvoj aplikacije. U sklopu AngularJS spomenut će se i HTML tehnologija, no neće biti opširno predstavljena.

2.2.1. Google Cloud Datastore

Google Cloud platforma je usluga koju pruža Google koja nudi usluge poslužitelja svim zainteresiranim korisnicima i programerima na jednakoj infrastrukturi koja pokreće i same Googleove aplikacije (https://cloud.google.com/why-google/future-proof/). Nude se poslužiteljske usluge, usluge pohrane podataka, kao i pohrane podataka na računalni oblak. Platforma se sastoji od niza proizvoda. Primjerice Google App Engine predstavlja PaaS (engl. Platform as a Service) za Internet aplikacije. Nudi automatsko skaliranje resursa s obzirom na poslužiteljsko opterećenje. Google Compute Engine je IaaS (engl. Infrastructure as a Service) koja omogućuje pokretanje virtualnih uređaja na zahtjev. Google Cloud Storage nudi mogućnost pohrane dokumenata i pristup njima s udaljenih lokacija. To nisu sve usluge jer Google nudi široku lepezu usluga. Jedna od njih i Google Cloud Datastore (u daljnjem tekstu: računalni oblak), potpuno upravljiva i visoko dostupna NoSQL baza podataka za nerelacijske podatke. Jedna od prednosti pohrane podataka na računalni oblak je što se usluga nalazi unutar Googleovih podatkovnih centara što minimizira mogućnost pojave grešaka i kvarova. Usluga također vrši enkripciju svih podataka prije pohrane i automatski ih dekriptira ukoliko ih zahtjeva autorizirani korisnik [5].

Podaci poslani iz plastenika se spremaju u entitete čija svojstva su vrijednosti izmjerene u plasteniku. Pri spremanju u bazu podaci dobivaju vremensku oznaku pristizanja. Na slici 2.6. prikazan je primjer jednoga entiteta.



Sl. 2.6. Primjer entiteta

Za dohvat i upis podataka postoje dva programska sučelja za izradu programa (engl. Application Programming Interface); `get` i `post` (u daljnjem tekstu: API). Prvi služi za dohvat podataka iz

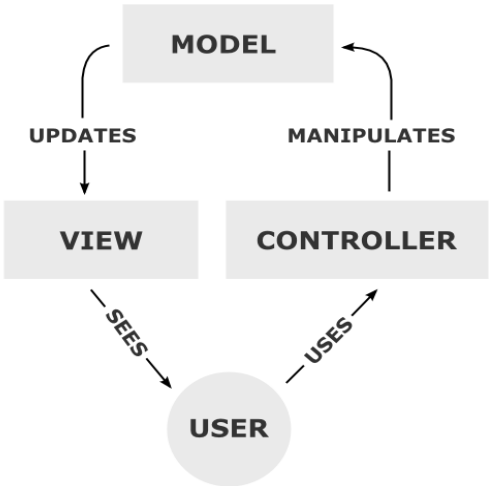
baze, dok drugi služi za upis podataka u bazu.

Prva metoda se inicijalizira kada korisnik otvori stranicu aplikacije koja zatim zahtjeva posljednje upisane podatke na računalnom oblaku, čime korisnik dobiva povratnu informaciju o stanju u plasteniku. Uvidom u vrijeme zapisa zadnjeg podataka korisnik može odrediti je li sve u redu ili se javio problem na relaciji plastenik – računalni oblak.

Druga metoda igra ulogu u Raspberry skripti, koja zapisuje podatke na računalni oblak. Kada korisnik želi upaliti navodnjavanje, stanje se zapisuje na računalni oblak, Raspberry čita stanje i prosljeđuje ga Arduinu koji otvara ventil i drži ga otvorenim dok korisnik ne odredi drukčije.

2.2.2. AngularJS

AngularJS je Internet aplikacijski okvir otvorenog koda koji održava Google zajedno sa zajednicom individualaca čiji je cilj odgovoriti na izazove koji se pojavljuju pri razvoju jednostraničnih (engl. single-page) aplikacija. Okvir je za klijentsku stranu i inkorporira MVC arhitekturu (Sl. 2.7.). Stabilna verzija se pojavila prvi put 2012., a stabilna verzija AngularJS 2 izlazi negdje sredinom 2016. godine.



Sl. 2.7. MVC arhitektura

Korisnik interakcijom na stranici, pomoću miša (klik miša) i tipkovnice, koristi kontrolere. Oni zapisuju akciju u `model` koji zatim generira određeni `view`. Tako definirana određena akcija ima svoju reakciju u obliku generiranja novog izgleda. Filozofija AngularJS-a može se sažeti u

načelo da se deklarativno programiranje treba koristiti pri izradi korisničkog sučelja [6].

Da bismo razumjeli kako AngularJS radi, potrebno je razumjeti objektni model dokumenta (engl. Document Object Model) i njegovu interakciju s objektima (u daljnjem tekstu: DOM) u HTML, XHTML, XML dokumentima. AngularJS učita kompletan objektni model, zatim kreira ubrizgavač (engl. injector) i sastavi listu direktiva koje uređuju DOM. Neke od najčešće korištenih direktiva su prikazane u tablici 2.1.

Tab. 2.1. *Najčešće korištene AngularJS direktive*

ng-app	Predstavlja korijensku direktivu, nakon koje se mogu koristit sve ostale direktive
ng-bind	Postavlja tekst DOM elementa na vrijednost vezanu uz izraz
ng-controller	Specificira JavaScript kontroler
ng-show i **ng-hide**	Uvjetno prikazivanje ili prikrivanje elementa, ovisno o izrazu
ng-view	Direktiva odgovorna za rukovanje putanjama

Najprimjetnije svojstvo AngularJS je dvostrano podatkovno povezivanje pri čemu se olakšava posao na poslužiteljskoj strani. Za to je odgovoran AngularJS `$scope` servis. On otkriva promjene u `model` dijelu i prema njima mijenja HTML, odnosno `view`. Također, i reverzibilna akcija je podržana, što znači da se bilo kakva promjena u `view` dijelu reflektira na `model`.

Od navedenih direktiva bit će potrebno koristiti korijensku direktivu. Izostavljanjem te direktive ostajemo bez mogućnosti korištenja AngularJS-a i njegovih posebnosti. Nju je potrebno naznačiti općenito na početku HTML dokumenta, najčešće u zaglavlju. Bitno je i napomenuti da je potrebno navesti i putanju do AngularJS skripte, bila ona lokalno pohranjena ili preko hiperveze. Nakon tih preliminarnih koraka, AngularJS je spreman za korištenje.

Pomoću `ng-controller` direktive rukovat će se raznim događajima u pozadini stranice. Tu će se nalazit sva logika potrebna za dohvat i upis podataka. Dohvat će se vršiti na određene akcije kako bi uvijek imali ažurirane podatke.

Korištenjem `ng-view` direktive najlakše se postiže privid višestranične aplikacije. Uz dobro definirane putanje i kontroler, HTML stranica se samo jednom učitava, a direktiva briše i dodaje DOM objekte. Dakle, korisnikovim klikom miša na hipervezu za drugu stranicu ne učitava se novi HTML dokument, nego direktiva odradi umetanje zatražene stranice u postojeću. Time se

dobiva na brzini i preglednosti stranice.

AngularJS podržava direktive i HTML elemente koji su prilagođeni po volji korisnika. Takve direktive sam korisnik razvija po potrebi. U slučaju ove aplikacije, bit će potrebno razviti posebnu direktivu koje će se pozivati svaki put kada bude potrebno vizualizirati linijski graf.

2.2.3. D3.js

D3.js je JavaScript biblioteka za razvoj dinamičkih, interaktivnih podatkovnih vizualizacija za Internet preglednike. Koristi široko rasprostranjene SVG, HTML5 i CSS standarde. Prva stabilna verzija izišla je 2011. godine. Koristi već ugrađene JavaScript funkcije za selekciju elemenata, kreiranje SVG objekata, uređivanje i dodavanje tranzicija, dinamičkih efekata ili skočnih opisa (engl. tooltips). Novostvoreni objekti mogu se urediti i primjenom CSS-a. Velike skupine podatak mogu se vrlo jednostavno vezati uz SVG objekte primjenom D3.js funkcija kako bi se generirali grafovi i dijagrami. Podaci mogu biti u raznim formatima (JSON, CSV, geoJSON) [7].

```
d3.selectAll("p")              //selektiraj sve <p> elemente
  .style("color","blue")       //obojaj u plavo
  .attr("class","kvadrat")     //dodaj klasu
  .attr("x",50);               //hor. pozicija na 50px
```

U gornjem dijelu koda uočavamo funkciju kojom se odabiru svi elementi unutar jednog dokumenta. Na primjeru može se primijetiti i ulančavanje metoda, gdje se više funkcija poziva na isti objekt. Moguće je odabrati i jedan element unutar objektnog modela dokumenta. Selekciju je moguće vršiti po HTML oznaci (engl. tag), klasi, atributu ili mjestu u hijerarhiji. Kada je element odabran njime se može lako manipulirati, primjerice promijeniti boju, tekst ili dodati tranziciju. Posebnost je povezivanje elemenata s grupom podataka koju se želi vizualizirati. Tako svaka vrijednost unutar podatkovnog niza dobiva svoj element s kojim je dalje moguće jednostavno upravljati.

U aplikaciji, nakon procesa dohvata podataka, potrebno ih je vizualizirati u obliku linijskog grafa. Vrijednosti na x-osi, vremenske vrijednosti, bit će iste na svim grafovima. Mijenjat će se samo vrijednosti mjerenih podataka na osi y. D3.js kod za vizualizaciju nalazit će se unutar AngularJS direktive.

3. REALIZACIJA RAZVOJA VIZUALNOG PANELA

Ovo će poglavlje biti posvećeno razvoju aplikacije za koju je teoretska osnova dana u prošlom poglavlju. Postepen razvoj upotpunit će se dijelovima koda, uz popratne komentare. Aplikacija će posjedovati tri stranice.

Na prvoj, početnoj stranici, koja će se prikazivati kada korisnik posjeti aplikaciju, bit će prikazani svi relevantni podaci o trenutnom stanju unutar plastenika. Ukoliko određene vrijednosti prekoračuju početne korisnički zadane granice, korisnik će vizualno biti upozoren. Promjena boje je upečatljiv alarm koji korisnik najlakše detektira. Također, početna stranica će prikazivati i vrijeme kada je zadnji podatak upisan u bazu podataka na računalnom oblaku. Ukoliko se pojavi alarm u tom segmentu, a aplikacija je u funkciji, korisnik može zaključiti kako se greška pojavila u prvom dijelu sustava, tj. u samom plasteniku ili u komunikaciji plastenika s oblakom.

Na drugoj stranici korisniku će biti pružen detaljniji uvid u mjerene vrijednosti tokom vremena. Pomoću linijskih grafova mjerene vrijednosti senzora u plasteniku bit će prikazane kroz vremenski period od deset sati. Unutarnja i vanjska temperatura će biti na istom grafu, protok vode i rad ventila također, dok će vlažnost zemlje i zraka biti prikazani na zasebnim grafovima. Točke na grafu povezat će se linearnom interpolacijom i klikom na njih pokazat će vrijednost koju je odgovarajući senzor zabilježio u danom trenutku. Pored grafova bit će prikazano i vrijeme zadnjeg primljenog podatka. Time korisnik, ukoliko je došlo do nekakve greške u prvom dijelu sustava, dobiva uvid u kretanje pojedinih mjerenih vrijednosti do pojave greške, odnosno kvara.

Na trećoj stranici bit će opisan rad aplikacije, način korištenja i detalji na koje korisnik treba obratiti pozornost. Pojašnjeni će biti alarmi i način na koji se manifestiraju. Pored opisa i uputa nalazit će se i trenutno stanje ventila kao i prekidač za ručno paljenje i gašenje ventila. Kada se ventil ručno upali, on ostaje upaljen sve dok ga korisnik sam ne ugasi. Ova opcija je ponuđena kada korisnik želi veću vlažnost zemlje, a sustav zbog svojih ograničenja to nije u mogućnosti ispuniti.

Ovo poglavlje je podijeljeno u tri potpoglavlja. U svakome je obrađena po jedna stranica aplikacije. Dijelovi koda bit će prikazani i komentirani.

3.1. Početna stranica

Razvoj aplikacije započinje uključivanjem korijenske direktive AngularJS-a u zaglavlje HTML dokumenta. U njoj se referencira ime koje će AngularJS koristiti kao varijablu u kojoj će se definirati AngularJS moduli.

index.html

```
<html ng-app="angApp">
```

Ovim kodom govorimo AngularJS-u koji element će biti smatran korijenom aplikacije. Sve unutar tog elementa će biti pod kontrolom AngularJS. Uobičajeno je da bude jedna korijenska direktiva po stranici. Međutim, samim definiranjem direktive AngularJS neće funkcionirati. AngularJS je proširena JavaScript biblioteka i kao takvu potrebno ju je referencirati u HTML dokumentu s odgovarajućim elementom.

index.html

```
<script type="text/javascript"
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.3/angul
ar.min.js"></script>
```

To su preliminarni koraci koji su potrebni kako bi se AngularJS okvir mogao uspješno koristiti. Sva logika AngularJS-a nalazi se u posebnoj datoteci u zasebnim zapisima. Razlog je lakše snalaženje kada se kod želi izmijeniti, doraditi ili se želi pronaći greška. Takve globalne varijable i funkcije imaju zasebnu skriptu. Kontroleri i direktive također.

index.html

```
<script type="text/javascript" src="jsDipl/angApp.js">
</script>
<script type="text/javascript"
src="jsDipl/controllers.js"></script>
<script type="text/javascript" src="jsDipl/directive.js">
</script>
```

Spomenuto je kako će biti tri stranice na kojima će korisnik moći dobiti informacije o stanju u plasteniku. Te stranice bilo bi potrebno povezati hipervezom, da se ne koristi AngularJS. Tako bi klikom na pojedinu stranicu dobivali novi kompletan HTML dokument, što bi za posljedicu imalo spor odaziv, jer treba proći određeni vremenski period dok Internet preglednik ne prevede HTML elemente, JavaScript biblioteke i CSS stilove u korisniku prihvatljiv izgled. Samim time aplikacija više ne bi bila jednostranična, nego višestranična, a to se želi izbjeći. To je jedan od izazova koji je nagnao programere zaposlene u Googleu da napišu JavaScript biblioteku koja će spremno odgovoriti na postavljene prepreke. AngularJS posjeduje direktivu koja omogućuje privid višestranične aplikacije. To je *ngRoute* modul koji je potrebno zasebno uključiti u HTML dokument.

index.html

```
<script type="text/javascript"
src="https://code.angularjs.org/1.5.3/angular-
route.min.js"></script>
```

Bitno je napomenuti da verzije moraju biti iste kako bi radile, u ovom slučaju 1.5.3. Kako je riječ o zasebnom modulu, potrebno je u globalnu varijablu AngularJS-a ubrizgati i ovisnost o ovome modulu.

jsDipl/angApp.js

```
var angApp = angular.module("angApp",["ngRoute"]);
```

Ovaj modul se sastoji od direktive *ngView* i opskrbljivača (engl. provider) *$routeProvider*. Direktivu je potrebno navesti u tijelu dokumenta gdje želimo da se određeni predložak prikaže. Većinom se deklarira u tijelu početne stranice. U kodu se navodi bez reference jer je jedina takva direktiva i nije potrebno posebno navoditi nikakvo ime.

index.html

```
<div ng-view=""></div>
```

Njezina zadaća je prikazati predloške na temelju putanje koja je određena u opskrbljivaču. Modul će imati svoju zasebnu skriptu.

jsDipl/route.js

```
angApp.config(["$routeProvider", function($routeProvider)
{
     $routeProvider
          .when("/", {
               templateUrl: "partials/info.html",
               controller: "infoCtrl"
          })
          .when("/grafovi", {
               templateUrl: "partials/grafovi.html",
               controller: "infoCtrl"
          })
          .when("/about", {
               templateUrl: "partials/about.html",
               controller: "infoCtrl"
          })
          .otherwise({
               redirectTo: "/"
          });
}]);
```

Iz navedenog koda vidimo da *$routeProvider* provjerava adresu stranice i na temelju adrese određuje koji će se predložak učitati. Taj predložak će i naslijediti određeni kontroler. Za sve adrese za koje *$routeProvider* ne može naći putanju, učitat će se početna stranica sa svojim predloškom. Promjena stranica je brža i doprinosi preglednosti.

3.1.1. Krajnje točke oblaka i info predložak

Krajnje točke oblaka (engl. cloud endpoints) predstavljaju razvojne blokove za izradu API-a s poslužiteljske strane. Slika 3.1. prikazuje primjer korištenja krajnjih točaka oblaka. One predstavljaju poveznicu između usluge pohrane podataka na računalni oblak i aplikacije, odnosno krajnjeg korisnika. Pripajanje JavaScript klijentske biblioteke s krajnjih točaka oblaka u

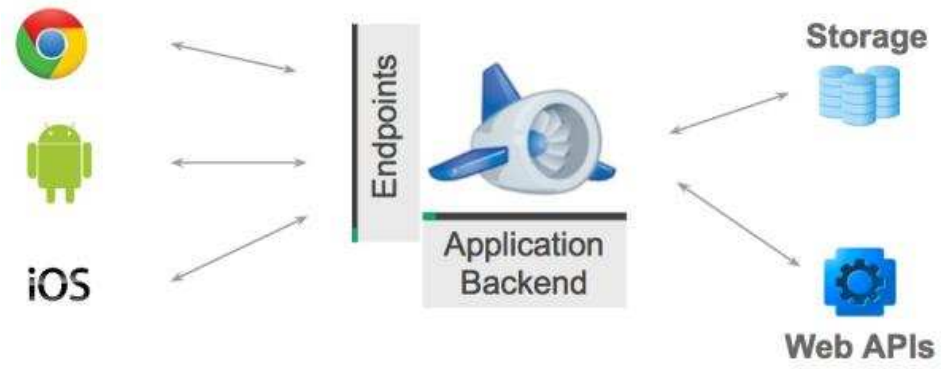
AngularJS je pojednostavljeno. Jednom linijom naredbi se može pozvati poslužiteljska logika i kroz povratnu funkciju, *model* se vrlo lako ažurira dobivenim rezultatima [8].

Pozornost treba obratiti na slijed inicijalizacije skripti. Redoslijed koji omogućava funkcionalnost je:

1. AngularJS,
2. skripte vezane za aplikaciju (angApp.js, controllers.js, directive.js, route.js),
3. Google API skripta, koja sadrži funkcionalnost krajnjih točaka oblaka.

index.html

```
<script
src="https://apis.google.com/js/client.js?onload=init"></script>
```



Sl. 3.1. *Krajnje točke oblaka*

Bitno je da se klijentska skripta učitava zadnja, jer ona poziva inicijalizacijsku funkciju koja je specificirana *onload* parametrom, u ovom slučaju *init()*. Zadaća funkcije *init()* jest da pozove drugu funkciju *window.initGapi()* koja je smještena unutar kontrolera. To je najučinkovitiji način jer omogućava AngularJS-u da se učita prije ili za vrijeme trajanja slanja zahtjeva prema poslužitelju. Ukoliko se zbog Internet preglednika dogodi da poslužitelj odgovori prije nego AngularJS postavi okolinu do kraja, javlja se greška da funkcija *window.initGapi()* nije definirana. Kako bi se to izbjeglo, potrebno je odgoditi zahtjev prema poslužitelju.

jsDipl/angApp.js

```
function init() {
     if ( typeof(window.initGapi) != 'function') {
          setTimeout(init, 500);
     } else {
          window.initGapi();
     }
};
```

Dostupnost Googleovih poslužitelja je preko 99.95%, pa je mogućnost pojave petlje koja bi konstantno slala zahtjev prema poslužitelju veoma mala. Pozivanjem *window.initGapi()* funkcije podrazumijeva se da je AngularJS uspješno učitan sa svim svojim modulima.

jsDipl/controllers.js

```
angApp.controller("infoCtrl", function($scope, $window,
timerServis) {
     $window.initGapi = function() {
          gapi.client.load(
"izvjestaj", "v1", postInit, restURL); }
});
```

AngularJS kontroler se definira na gore prikazan način. Pod navodnicima se navodi ime koje će se u HTML dokumentu vezivati uz direktivu potrebnu za poziv kontrolera. Nakon toga slijedi definiranje ovisnosti koje ubrizgavamo (engl. dependency injection). AngularJS globalnom *window* objektu se pristupa preko *$window* notacije. Prizivanjem ove funkcije pozivamo Google API metodu za učitavanje klijentske biblioteke. Njezina sintaksa je da se prvo navodi ime API metode, a zatim njezina verzija. Nakon toga se definira povratna funkcija koja će se pozvati kada se API metoda učita. Zadnji argument je URL adresa API metode.

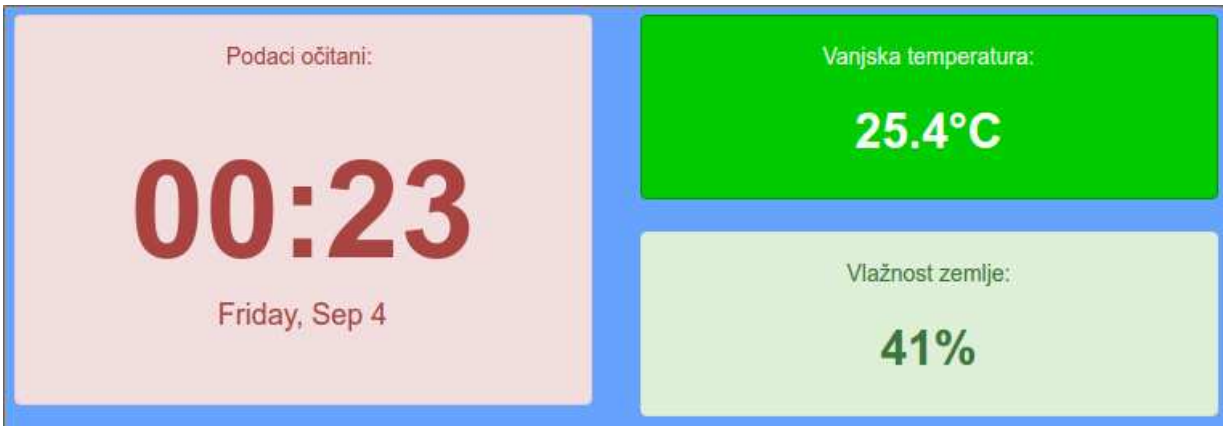
Po učitavanju API metode, poziva se povratna funkcija koja šalje zahtjev za dohvatom podataka koji se nalaze na računalom oblaku. Podaci su spremljeni kao niz objekata gdje jedno očitanje svih senzora u plasteniku predstavlja jedan objekt. Objekte je potrebno po primitku sortirati prema vremenu dolaska na oblak. Nakon sortiranja vrijednosti se svrstavaju u zasebne nizove s

obzirom na mjerenu veličinu. Tako će sve dohvaćene vrijednosti unutarnje temperature biti u jednom nizu, vanjske u drugom. Sve ostale mjerene veličine će slijediti taj princip. Povratna funkcija *postInit()* dohvaća određen broj objekata.

jsDipl/controllers.js

```
var postInit = function() {
          gapi.client.izvjestaj.getIzvjestaji({"number":
30}).execute(function(resp) {
                podaci = resp.items;
                Spodaci = podaci.sort(compare);
                $scope.dataTempV = Spodaci.map(function(i)
{ return i.vanjska_temperatura; });
                $scope.dataTempU = Spodaci.map(function(i)
{ return i.unutarnja_temperatura; });
                $scope.dataTime = Spodaci.map(function(i)
{ return i.vrijeme_primitka; });
```

Povezivanjem niza vrijednosti sa *$scope* objektom omogućavamo jednostavno dvostrano podatkovno vezivanje između *model* i *view* dijela. Svaka promjena vrijednosti u *model* dijelu će inicirati promjenu u *view* dijelu. Unutar istog kontrolera definiraju se i alarmne vrijednosti koje će korisnika na početnoj stranici vizualno upozoriti da su određene izmjerene vrijednosti iznad dopuštenih. Primjerice, ako je vlažnost zemlje ispod 40%, aktivirat će se vizualni alarm. Tako niska vrijednost će ujedno aktivirati i sustav za navodnjavanje. Također, ako je prošlo više od sat vremena od zadnje upisane vrijednosti u oblak, opet će se aktivirati vizualni alarm koji sugerira korisniku kako postoje određene poteškoće u radu dijela sustava koji je odgovoran za slanje izmjerenih podataka na oblak. Slika 3.2. prikazuje pojavu alarma vezanu uz zadnji upisani podataka na računalni oblak. Također, slika prikazuje i trenutno stanje pripadajućih temperatura i vlažnosti. Fokus miša je na vanjskoj temperaturi.



Sl. 3.2. Alarm koji vizualno upozorava korisnika

partials/info.html

```
<div class="col-md-2" ng-show="normalStaV">
    Stanje ventila: {{ dataStaV[0] === 1 ? "Upaljen" :
"Ugašen" }} </div>
<div class="col-md-12" ng-show="alertTime">
    Podaci očitani:{{ dataTime[0] | date:"HH:mm
dd/MM/yyyy" }} </div>
```

Gornji kod prikazuje kako se pomoću AngularJS izraza vrlo lako mogu prikazati podaci koji su ranije dohvaćeni. Unutar izraza se mogu koristiti metode za formatiranje podataka prije nego što se prikažu. Primjer je prikaz vremena. Vrijeme koje aplikacija dobiva sa računalnog oblaka je vrijeme u milisekundama. Kako ne bi bili primorani pisati dodatne linije koda za pretvaranje, AngularJS izrazi imaju poseban *date* filtar kojim se formatira vrijeme prema željama i potrebama. Pored formatiranja krajnjih izraza, moguće je koristiti i jednostavne ternarne operatore. Primjer je prikaz stanja ventila.

3.2. Linijski grafovi

D3.js biblioteka nije kolekcija gotovih čestih i uobičajenih grafova. Uobičajeno je da se koristi za prikaz linijskih ili kružnih grafova, ali prava moć leži u njegovoj fleksibilnosti i kontroli nad konačnim rezultatom. Ukoliko želimo koristiti D3.js biblioteku potrebno ju je učitati unutar dijela HTML dokumenta u kojem će se pojaviti određena vizualizacija.

index.html

```
<body>
<script type="text/javascript"
src="https://d3js.org/d3.v3.min.js">
</script>
</body>
```

Dosta je razloga zašto kombinirati AngularJS i D3.js kada su u pitanju vizualizacije, a jedan od njih su i direktive. Direktive uvelike pomažu pri stvaranju vizualizacija koje će se više puta upotrijebiti. One su način na koji kreiramo zasebne HTML elemente koji se ponašaju kao ugrađene HTML oznake. Posebnost direktiva je što su u mogućnosti i primati određene argumente.

partials/grafovi.html

```
<line-Graph
data1="dataTempV" data2="dataTempU" data4="[]"
vrijeme="dataTime" color="{{ 'blue' }}" color2="{{ 'red'
}}" mjerna="{{ '°C' }}" name="{{ 'Odnos vanjske i
unutarnje temperature' }}" tleg1="{{ 'Vanjska' }}"
tleg2="{{ 'Unutarnja' }}">
</line-Graph>
```

Argumenti koji se prosljeđuju su nizovi podataka ranije dohvaćenih iz oblaka, unutarnje i vanjske temperature te vremena u kojima su ta mjerenja obavljena. Prosljeđuje se i boja grafa, u ovom slučaju dvije boje jer će isti graf prikazivati i unutarnju i vanjsku temperaturu. Sljedeći argument je mjerna jedinica, koja se za svaki graf mijenja, kao i naslov grafa. Zadnji argument je tekst koji se pojavljuje u legendi na grafu. Ako se želi graf sa samo jednom prikazanom vrijednosti, tada *data2* argument mora biti prazan niz. Argument *data4* je prazan niz jer je rezerviran za prikaz rada ventila koji nije standardni linijski graf.

jsDipl/directive.js

```
angApp.directive("lineGraph",function(){
     function link(scope, element, attr){
//D3.js kod
//...
}
     return{
          restrict: "E",
          scope:    { data1:"=", data2:"=", data4:"=",
vrijeme:"=", color:"@", color2:"@", mjerna:"@", name:"@",
tleg1:"@", tleg2:"@" },
          link: link
     } });
```

U gornjem dijelu koda prikazano je definiranje direktive za linijske grafove. Uočava se razlika između imena direktiva. U HTML dokumentu se navode povlakom, a u skripti velikim početnim slovima. AngularJS sam vrši pretvaranje direktive iz velikih slova u HTML oznake s povlakom. Direktive sadrže i funkciju *link()* koja ima ulogu konstruktora iz objektnog programiranja. U tijelu funkcije se nalazi sve ono što se treba prikazati svaki put kada se direktiva navede u HTML dokumentu. Opcija `restrict:"E"` govori AngularJS-u da se direktiva koristi kao element. Moguće opcije su još `"C"` za klasu (`<div class="moja-direktiva"></div>`), `"A"` kao argument (`<div moja-direktiva></div>`) ili `"M"` kao komentar (`<!--directive: moja-direktiva -->`).

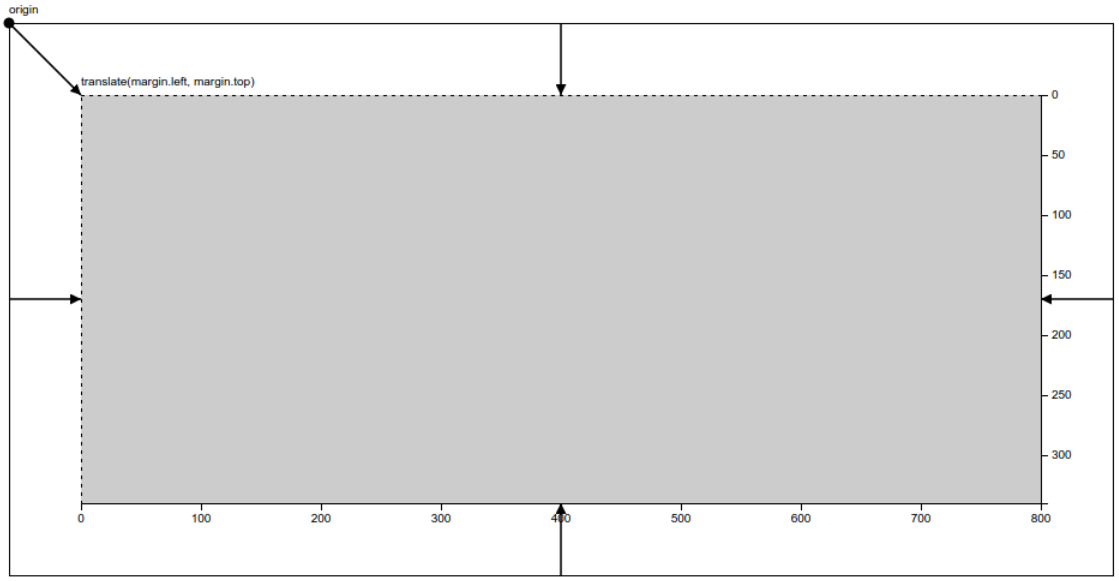
Kako bismo iskoristili podatke koje prosljeđujemo kao argumente potrebno je iskoristiti svojstvo pod nazivom izolirani *scope* (engl. isolate scope) koje govori AngularJS-u što učiniti s dobivenim podacima. Potrebno je samo dodati *scope* svojstvo u objekt koji vraćamo iz deklaracije direktive, kao što je navedeno gore u kodu. Dobivenim vrijednostima se pristupa preko *scope* objekta.

jsDipl/directive.js

```
//kod unutar direktive
```

```
var data1 = scope.data1,
    data2 = scope.data2,
    data4 = scope.data4,
    mjerna_jedinica = scope.mjerna,
    naslov = scope.name;
```

Podaci su unutar direktive i sljedeći korak je njihova vizualizacija korištenjem D3.js biblioteke. Prvotno se definiraju visina i širina polja, a zatim margine (Sl. 3.3.). Margine predstavljaju udaljenost od granica polja te su okvir unutar kojega će biti smješteni x i y os, mjerne jedinice, legenda i naziv grafa.



Sl. 3.3. Definiranje margina

Zatim se kreira varijabla `svg` u kojoj se pomoću D3.js ugradbene naredbe selektira DOM objekt relativan u odnosu na direktivu. Kreira se novi HTML element i dodaje unutar DOM-a. Atribut `viewBox` određuje do kojih granica se objekt širi prilikom uvećanja stranice. Dodaje se novi element koji se translatira za lijevu i gornju marginu, a koji će služiti kao polazišna točka prikazivanja grafa.

`jsDipl/directive.js`

```
var  width = 800,
```

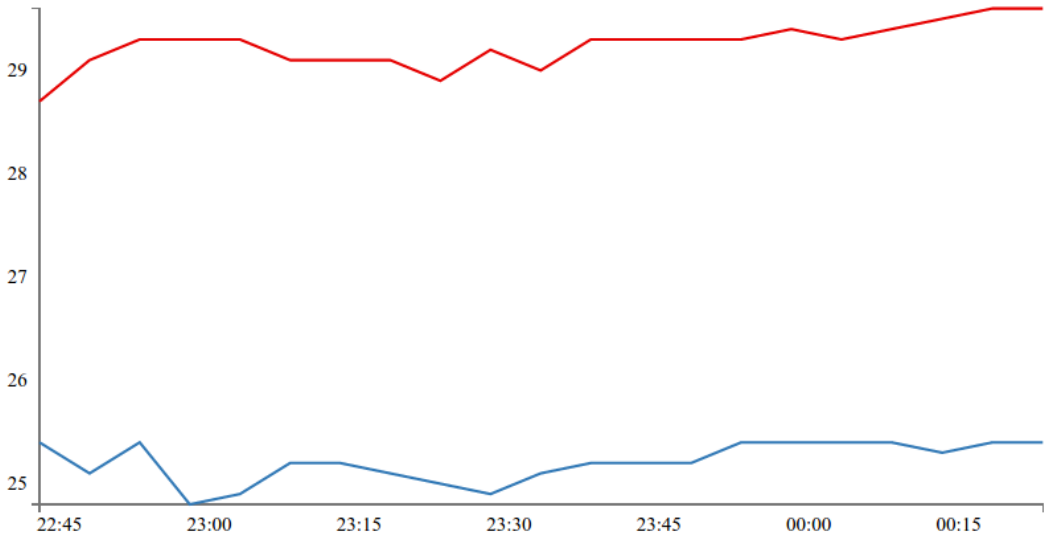
```
     height = 400,
     margins = { top: 20, right: 20, bottom: 20, left: 50
     },
svg = d3.select(element[0]).append("svg")
          .attr("viewBox", "0 0 900 450")
     .append("g")
          .attr("transform", "translate(" + margins.left +
"," + margins.top + ")"),
```

Za prikaz linijskih grafova potrebne su osi x i y na kojima su prikazane vrijednosti koje su izmjerene u plasteniku. Te vrijednosti je potrebno skalirati, odrediti maksimalnu i minimalnu vrijednost te pružiti domet u kojem moraju biti prikazane. D3.js posjeduje ugrađene funkcije koje sve to navedeno omogućavaju. Dva linijska grafa zahtijevaju povezivanje vrijednosti u jedan niz kako bi skala bila pravilno određena. U funkciji `xOs`, vidljivoj u kodu niže, definira se sve što je potrebno za prikazivanje x osi na linijskom grafu. Određuje se skala osi, okviran broj oznaka, format vremena i orijentacija prikazanih vrijednosti u odnosu na x-os. Funkcija uzima u obzir broj oznaka, ali sama određuje, na temelju skale, njihov konačan broj. Prikaz iziskuje dodavanje novog elementa i pozivanje funkcije `xOs`. Slično vrijedi i za prikaz y-osi.

jsDipl/directive.js

```
//skale
var  xScale = d3.time.scale().range([margins.left, width -
margins.right]).domain([d3.min(vrijeme),d3.max(vrijeme)]),
yScale = d3.scale.linear().range([height - margins.top,
margins.bottom]).domain([d3.min(data3), d3.max(data3)]),
//xOs funkcija
xOs = d3.svg.axis()
     .scale(xScale)
     .ticks(5)
     .tickFormat(d3.time.format("%H:%M"))
     .orient("bottom");
```

```
//prikaz
svg.append("svg:g")
      .attr("class", "axis")
      .attr("transform", "translate(0," + (height -
margins.bottom) + ")")
      .call(xOs);
```



Sl. 3.4. Linearno povezivanje temperaturnih vrijednosti

Za iscrtavanje linije koja povezuje vrijednosti princip je sličan kao i sa osima. Definira se funkcija koja računa koordinate koristeći anonimne funkcije koje iteracijom, za sve točke danog niza, računaju skalirane pripadajuće vrijednosti. Za interpolaciju se odabire linearna. Prikaz linije zahtjeva dodavanje elementa i poziv funkcije. Ostale opcije se sastoje od odabira širine linije i boje. Za prikaz još jedne vrijednosti u obliku linije na istom grafu, poziva se druga funkcija, ali s različitim nizom podataka (Sl. 3.4.).

jsDipl/directive.js

```
var line = d3.svg.line()
      .x(function(d, i) {
```

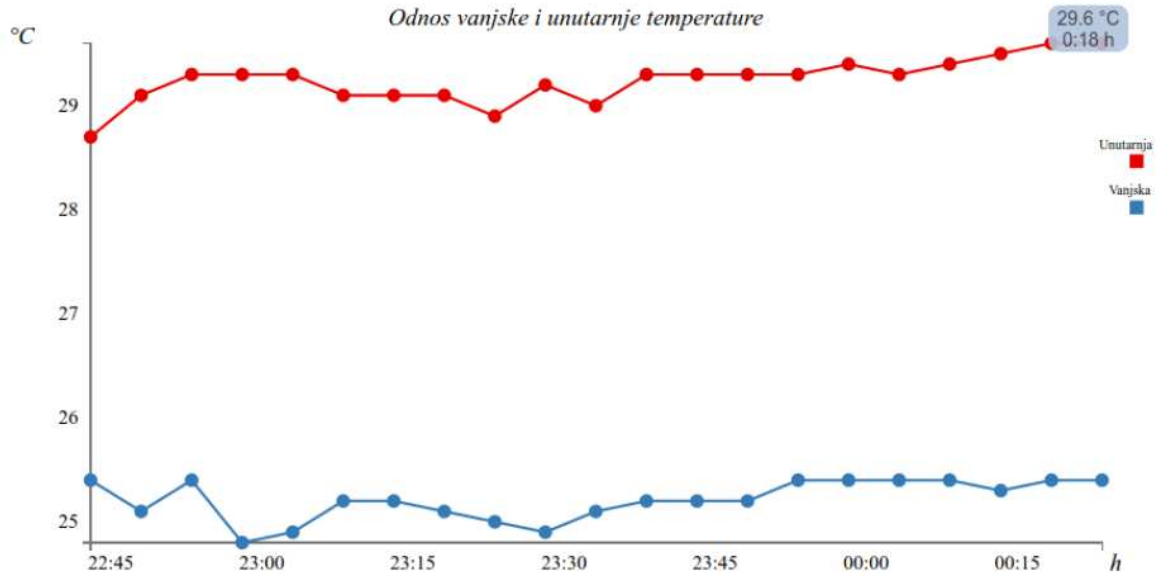
```
            return xScale(vrijeme[i]);
        })
        .y(function(d, i) {
            return yScale(data1[i]);
        })
        .interpolate("linear");
svg.append("svg:path")
        .attr("d", line(data1))
        .attr("stroke", color)
        .attr("stroke-width", 2)
        .attr("fill", "none");
```

Kako bi korisnik imao uvid u stanja mjerenja tokom vremena prikazanog na grafu, na linijama grafa postavljeni su krugovi koji reagiraju na klik miša. Klik miša omogućava prikaz stanja mjerene vrijednosti u određenom vremenskom trenutku pojavom skočnog opisa. Za definiranje kruga potrebna su tri elementa: polumjer te x i y koordinate. Polumjer je proizvoljan, a x i y koordinate se dobiju pozivom funkcija za skaliranje nad podatkovnim nizom. Princip je to koji je omogućio prikaz linije grafa. Pri kreiranju krugova koristi se posebna funkcija *enter()* koja za svaku vrijednost iz podatkovnog niza koji joj se proslijedi kreira novi element, odnosno novi krug (Sl. 3.5.). Skočni opis je varijabla dodana u DOM objekt čija je pojava definirana klikom miša. Pomicanje miša briše skočni opis.

jsDipl/directive.js

```
svg.selectAll("dot")
        .data(data1)
        .enter().append("circle")
        .attr("r", 5)
        .attr("cx", function(d, i) { return
xScale(vrijeme[i]); })
        .attr("cy", function(d, i) { return yScale(data1[i]);
})
        .attr("fill", color)
```

```
.on("click", function(d, i) { div.html(data1[i] + "" +
mjerna_jedinica + "</br>" + vrijeme[i].getHours() + ":" +
vrijeme[i].getMinutes() + " h")
     .on("mouseleave", function(d) {
          div.transition()
               .duration(400)
               .style("opacity", 0);
     });
```



Sl. 3.5. Graf odnosa vanjske i unutarnje temperature

Za prikaz teksta dovoljno je na postojeći `svg` element dodati element `text`. Zatim je potrebno odrediti klasu, radi CSS uređivanja, poziciju u polju i tekst koji će biti prikazan. Princip je isti i za prikaz mjernih jedinica, naslova grafa i teksta u legendi. Primjer je postavljanje teksta za naslov grafa (Sl. 3.5.).

`jsDipl/directive.js`

```
svg.append("text")
     .attr("class", "text")
     .attr("transform", "translate(" + (width/ 2) + ","
```

```
+(margins.top / 3) + ")")
      .attr("text-anchor", "middle")
      .text("" + naslov + "");
```

Legenda se sastoji od teksta, koji se dobije na gore navedeni način, i dva pravokutnika. Bitni atributi za prikaz pravokutnika su x i y pozicija i širina i visina. Boja pravokutnika se prosljeđuje iz direktive.

3.3. Upravljanje navodnjavanjem

Zadnji predložak koji ubrizgavač učitava u početnu stranicu sastoji se od dva dijela. Prvi dio definira trenutno stanje ventila i prekidač koji omogućava slanje naredbe za paljenje na računalni oblak. Raspberry u određenim vremenskim intervalima provjerava bazu podataka. Ukoliko se u njoj nalazi korisnikova naredba za paljenje, Raspberry trenutno prosljeđuje signal za puštanje vode Arduinu. Navodnjavanje se pali i ukoliko je vlažnost veća od korisnički definirane granice. Ono ostaje upaljeno sve dok Raspberry na oblaku ne očita naredbu za gašenje. Naravno, moguća je i reverzibilna akcija kojom korisnik trenutno gasi navodnjavanje, sve do idućeg ponovnog očitanja senzora. Kod za slanje naredbe na oblak se nalazi u zasebnom kontroleru.

jsDipl/controllers.js

```
angApp.controller("upisCtrl", function($scope) {
      $scope.postFunction = function() {
            var postObject = {};
```

Objektom `$scope` povezujemo ime funkcije u kontroleru s nazivom akcije klika miša u HTML predlošku. Kontroler provjerava trenutno stanje ventila i s obzirom na to stanje prikazuje odgovarajući prekidač za paljenje, odnosno gašenje ventila. Trenutni protok može se nadgledati na početnoj stranici. Klikom na prekidač poziva se funkcija koja kreira prazan objekt. S obzirom na trenutno stanje ventila, vrijednost unutar objekta se postavlja u odgovarajuće stanje. Ako je ventil bio ugašen, moguće ga je jedino upaliti, to jest svojstvo vrijednosti ventila unutar objekta se postavlja na paljenje. Potom se poziva Google API metoda za upis i podatak se šalje i upisuje u bazu na oblaku. Poslije upisa, vrijednost funkcije prekidača se postavlja na gašenje jer se

trenutno stanje ventila promijenilo.

jsDipl/controllers.js

```
gapi.client.izvjestaj.postIzvjestaj(postObject).execute
(function(resp) {});
```

Drugi dio predloška predstavlja opis i način rada aplikacije. Također su ukratko objašnjene stranice aplikacije i dana je shema sustava. Na kraju, korisniku je prikazan sustav vizualnih alarma.

4. ZAKLJUČAK

Cilj rada je bio omogućiti korisniku trenutni uvid u stanje sustava, koji je u ovom slučaju bio plastenik. Pretpostavka na kojoj je zasnivan ovaj rad jest postojanje određene elektroničke arhitekture u korisnikovom sustavu. Ta elektronika je potrebna ukoliko se želi nadzirati sustav i s udaljenih lokacija. Koliko se upravljačke logike postavi u taj dio sustava, tolika će biti mogućnost aplikacije koja će nadzirati sustav. Razmatrani sustav u ovom radu opskrbljen je s više senzora i jednim aktuatorom. Sukladno tome je i razvijana aplikacija. Odabrane tehnologije su sveprisutne na Internetu i podržavaju ih uređaji kojima je dovoljan pristup Internetu i koji posjeduju Internet preglednik.

Na početku rada dan je pregled korištenih tehnologija čije je korištenje i upotreba slobodna. HTML i CSS predstavljaju standarde koji su već dugi niz godina dostupni i rašireni među web programerima. Pojavom zahtjeva za jednostraničnim aplikacijama razvijaju se Internet okviri koji udovoljavaju tom zahtjevu. AngularJS biblioteka razvijena je na temeljima JavaScript-a te predstavlja i takav okvir. U radu su korištene sve bitne posebnosti AngularJS-a, od specifičnih direktiva, preko upotrebe kontrolera, kako bi se ažurirao `view`, i dvostranog povezivanja podataka pa sve do korištenje direktive i modula za privid višestranične aplikacije.

D3.js biblioteka nastala je s razlogom da se olakša prikaz dinamičkih vizualizacija korištenjem već postojećih elemenata. Visoko je fleksibilna i intuitivna za korištenje. Kroz rad pokazane su sve važnije ugradbene funkcije koje služe za manipulaciju HTML elementima i njihovo povezivanje s podacima. Selekcija elementa se vrši jednom linijom koda, a zatim se pomoću metode ulančavanja svojstvo elementa mijenja.

Google Cloud Datastore usluga podržava zapis podataka u oblak. Usluga je besplatna ukoliko se ne prijeđe određeni dnevni limit poslanih zahtjeva na poslužitelj. Pogodnosti usluge su brz odziv, visoka poslužiteljska dostupnost i pouzdanost.

Područje primjene aplikacije ovakvog tipa nije ograničeno samo na plastenik. Aplikacija se vrlo lako integrira u bilo koji sličan sustav. Dovoljno je samo dohvatiti podatke i predati ih direktivi za prikaz linijskih grafova. Dodatne linije koda senzora ili aktuatora lako se integriraju u postojeće kontrolere. Ograničenje aplikacije je što pretpostavlja postojanje podataka na računalnom oblaku. Za daljnji razvoj aplikacije korištenjem navedenih tehnologija bilo bi potrebno proširiti aspekt dohvata podataka aplikacije.

LITERATURA

[1] Klipfolio, Operational dashboards vs. Analytical dashboard, https://www.klipfolio.com/resources/articles/operational-analytical-bi-dashboards, pristup 10. lipnja 2016.

[2] Klipfolio, Social media dashboard, https://app.klipfolio.com/published/0fb58406c23c646ed329255b5abfb920/social-media, pristup 10. lipnja 2016.

[3] Dashboard, Dashboard (management information systems), https://en.wikipedia.org/wiki/Dashboard_(management_information_systems), pristup 30. svibnja 2016.

[4] Sustav za nadzor i upravljanje plastenikom, SNUP, http://vcg.etfos.hr/iot/2015/13/, pristup 3. travnja 2016.

[5] Google Cloud, Datastore https://cloud.google.com/datastore/docs/concepts/overview, pristup 4. travnja 2016.

[6] Grant, A., Beginning AngularJS, Apress, New York City, 2014.

[7] Lerner, A., Powell, V., D3 on AngularJS, Leanpub, Victoria, 2014.

[8] Google Cloud, AngularJS cloud endpoints, https://cloud.google.com/solutions/angularjscloud-endpoints-recipe-for-building-modern-web-applications, pristup 5. travnja 2016.

SAŽETAK

Cilj rada je pomoću Internet aplikacije prikazati stanje korisnikovog sustava, u ovom slučaju plastenika. Podaci prikazani aplikacijom moraju biti ažurirani kako bi korisnik imao uvid u trenutno stanje unutar plastenika. Korisnik pristupom aplikaciji aktivira pozadinsku funkciju koja šalje zahtjev poslužitelju za ažuriranim podacima. Ukoliko određene izmjerene vrijednosti prekoračuju korisnički zadane granice, javlja se vizualni alarm koji upozorava korisnika. Aplikacija daje korisniku i prikaz mjerenih podataka kroz određeno vremensko razdoblje pomoću linijskog grafa, koji je rađen uporabom D3.js biblioteke. U plasteniku se pored senzora za mjerenje temperature, vlažnosti i protoka vode nalazi i aktuator ventil koji regulira protok vode. Navodnjavanje se uvijek pali ukoliko vlažnost zemlje padne ispod određene razine. Korisnik pomoću aplikacije može i sam proizvoljno upaliti i ugasiti navodnjavanje. Aplikacija s plastenikom komunicira preko Google Cloud Datastore usluge, na koju se spremaju mjerene vrijednosti iz plastenika.

Ključne riječi: AngularJS, D3.js, MVC, IoT

ABSTRACT – WEB DASHBOARDS

The aim of this study is to show the state of the user's system, in this case a greenhouse, using the Internet application. The data displayed in the application have to be updated to give the user an insight into the current situation in the greenhouse. By accessing the application, the user triggers a background function that sends a request to the server for the updated data. If certain measured values exceed the preset limits, there will be a visual alarm that alerts the user. The application presents the user with the display of the measured data over a period of time using the line chart, which was made using the D3.js library. In the greenhouse, along with the temperature, humidity and water flow sensors there is an actuator valve which regulates the water flow. The irrigation is always turned on if the soil moisture value drops below a certain level. The user can arbitrarily switch on and off the irrigation via the application. The application communicates with the greenhouse via Google Cloud Datastore service, onto which the measured values form the greenhouse are stores.

Key words: AngularJS, D3.js, MVC, IoT

ŽIVOTOPIS

Adrijan Đurin rođen je 9. veljače 1990. godine u Vinkovcima. Od rođenja živi u Vinkovcima gdje je pohađao osnovnu školu „Antun Gustav Matoš“. Nakon završetka osnovne škole 2005. godine upisuje 1. razred prirodoslovno-matematičke gimnazije „M. A. Reljković“ u Vinkovcima. 2009. godine završava srednju školu i upisuje Elektrotehnički fakultet u Osijeku, sveučilišni preddiplomski studij računarstva. Preddiplomski studij završava 2013. godine i iste godine upisuje diplomski studij, smjer procesno računarstvo.

_____

Adrijan Đurin