SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Luka Vučetić

Zagreb, 2021.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Mentor:
Prof. dr. sc. Željko Šitum

Student:
Luka Vučetić

Zagreb, 2021.

Izjavljujem da sam ovaj rad izradio samostalno koristeći znanja stečena tijekom studija i navedenu literaturu.

Zahvaljujem se mentoru prof. dr. sc. Željku Šitumu što mi je omogućio izradu ovog diplomskog rada te na odvojenom vremenu za savjete i pomoć.

Također, zahvaljujem se komentoru Juraju Beniću, mag. ing. mech. na ukazanom povjerenju i odvojenom vremenu za pomoć i savjete tijekom izrade ovog rada.

Na kraju, posebno se zahvaljujem svojim roditeljima Sanji i Davoru, bratu Lovri i sestri Luciji te djevojci Tei na pruženoj potpori i razumijevanju tijekom studiranja i izrade ovog rada.

Luka Vučetić

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za diplomske radove studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala te mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum:	Prilog:
Klasa: 602-04/21-6/1	
Ur. broj: 15-1703-21	

DIPLOMSKI ZADATAK

Student: **LUKA VUČETIĆ** Mat. br.: 0035203033

Naslov rada na hrvatskom jeziku: **Izravno pogonjeni elektrohidraulički sustav upravljan putem web-a**

Naslov rada na engleskom jeziku: **Web controlled direct driven electrohydraulic system**

Opis zadatka:

U Laboratoriju za automatiku i robotiku izrađen je eksperimentalni postav izravno pogonjenog elektro-hidrauličkog sustava, koji ima mogućnost spajanja na internet preko usmjerivača (eng. router) ugrađenog na upravljački ormar. Sustavom upravlja PLC koji ujedno prikuplja sve mjerene podatke. Unosom različitih podataka u registre PLC-a moguće je pokrenuti i zaustaviti sustav, odabrati između klasičnog i izravno pogonjenog sustava te odabrati referentni signal. Svi prikupljeni podatci spremaju se u bazu podataka i prikazuju u web aplikaciji. U cilju proširenja mogućnosti postojeće web aplikacije potrebno je osmisliti način na koji se korisniku može prikazati slika postava u realnom vremenu te povijesni pregled snimljenih podataka. Radi lakšeg upravljanja sustavom potrebno je osmisliti način spremanja podataka iz web aplikacije u registre PLC-a. Za izradu web aplikacije koristit će se program Python i mrežna tehnologija Web2Py.

U radu je potrebno:
- opisati pojam interneta stvari (eng. Internet of Things, IoT) i njegovu primjenu u industriji,
- opisati eksperimentalni postav, shematski prikazati način odvijanja komunikacije između korisnika, servera i sustava, a zatim opisati korišteni komunikacijski protokol za ostvarivanje komunikacije između servera i PLC-a,
- konstruirati nosač za Raspberry Pi 4 i USB kameru te izraditi aplikaciju za prikaz slike na udaljenom računalu,
- proširiti postojeću aplikaciju tako da se omogući povijesni pregled svih snimljenih rezultata, prikaz slike u realnom vremenu te upisivanje željene vrijednosti u registar PLC-a.

U radu je potrebno navesti korištenu literaturu i eventualno dobivenu pomoć.

Zadatak zadan: 4. ožujka 2021.

Rok predaje rada: 6. svibnja 2021.

Predviđeni datum obrane: 10. svibnja do 14. svibnja 2021.

Zadatak zadao: prof. dr. sc. Željko Šitum

Predsjednica Povjerenstva: prof. dr. sc. Biserka Runje

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

POPIS PROGRAMSKIH KÔDOVA

POPIS TEHNIČKE DOKUMENTACIJE

BROJ CRTEŽA	Naziv iz sastavnice
PR-001-000	Gornji držač kamere
PR-002-000	Donji držač kamere
PR-003-000	Konusna prirubnica
PR-001-001	Nosač za Raspberry Pi

SAŽETAK

U ovom radu opisan je Internet stvari i njegova primjena, Modbus TCP/IP protokol i Web2Py platforma za izradu web aplikacija. Također, dan je i uvid u eksperimentalni hidraulički postav koji je povezan sa serverom pomoću Modbus TCP/IP protokola u svrhu obrade podataka i njihove interpretacije u obliku pristupačnom korisniku putem web aplikacije izrađene na Web2Py platformi. Koristeći opisane tehnologije unaprijeđena je postojeća web aplikacija te su izrađeni nosači za web kameru i Raspberry Pi uz korištenje suvremene aditivne tehnologije polimernog printanja.

Ključne riječi: Internet stvari, Modbus TCP/IP, Web2Py, hidraulika, Raspberry Pi, aditivna proizvodnja

SUMMARY

This work describes the Internet of Things and its applications, the Modbus TCP / IP protocol and the Web2Py platform for web application development. Also, an insight into the experimental hydraulic system connected to the server using the Modbus TCP / IP protocol for the purpose of data processing and their interpretation in an user-accessible form via a web application developed on the Web2Py platform. Using the described technologies, the existing web application was improved and webcam and Raspberry Pi mounts were made using modern additive polymer printing technologies.

Key words: Internet of Things, Modbus TCP/IP, Web2Py, hydraulics, Raspberry Pi, additive manufacturing

1. UVOD

U zadnjem desetljeću nastala je velika eksplozija količine prikupljenih podataka. Današnji život nezamisliv je bez korištenja neke naprave, uređaja ili senzora koji nam očitava neke podatke. Nama ljudima su ti podaci sami po sebi od malog značaja. Uzmimo za primjer fitnes narukvice koje mjere broj prijeđenih koraka. Nama ljudima bi čisti podaci iz senzora bili samo šuma brojeva. Upravo je za takav problem osmišljeno rješenje u vidu baza podataka i njihovog interpretiranja. Uz pomoć pametnih programa, mikroračunala mogu decentralizirano i samostalno donositi odluke na temelju prikupljenih podataka. Davanjem takve inteligencije običnoj narukvici postižemo mogućnosti interpretacije podataka senzora u brojenje koraka, praćenje otkucaja srca ili čak praćenje kvalitete spavanja. Odnosno, prikupljeni podaci se strukturiraju te kroz analizu postaju baze znanja za samostalne odluke autonomnih strojeva. Slika 1 prikazuje konceptualni prikaz strojnog učenja. Daljnji razvoj tehnologije Interneta stvari nam nudi nove načine upotrebe svakodnevnih objekata, mogućnost da sve oko nas postane pametno, odnosno, sveprisutna inteligencija na našem dlanu.

Slika 1. Konceptualni prikaz strojnog učenja [1]

2. OPIS EKSPERIMENTALNOG POSTAVA

U sklopu izrade ovog diplomskog rada, koristio se izrađeni eksperimentalni postav u Laboratoriju za automatiku i robotiku na Fakultetu strojarstva i brodogradnje. Svrha ovog eksperimentalnog postava je testiranje raznih upravljačkih algoritama, usporedbe energetske učinkovitosti i dinamike sustava te implementacije Interneta stvari. Na slici 2 prikazan je eksperimentalni postav koji se sastoji od elektrohidrauličkog, mjernog i upravljačkog sustava.

Slika 2. Eksperimentalni postav

2.1. Elektrohidraulički sustav

Ovaj sustav bio je zamišljen u svrhu usporedbe direktno pogonjene hidraulike i standardne proporcionalne hidraulike. Iz tog razloga na postavu se nalaze izravno pogonjeni elektrohidraulički podsustav i klasični proporcionalno upravljani elektrohidraulički podsustav koji su paralelno spojeni na zajednički aktuator te ih opskrbljuje pogonski dio sa zajedničkim spremnikom ulja. Slika 3 lijevo prikazuje servomotor koji pogoni dvije reverzibilne zupčaste pumpe međusobno spojene vratilom koje služe za izravnu kontrolu protoka ulja prema cilindru u izravno pogonjenom elektrohidrauličkom sustavu. Na slici 3 desno prikazan je jednofazni izmjenični motor koji pogoni pumpu koja osigurava konstantni protok u proporcionalnom elektrohidrauličkom sustavu.

Slika 3. Izravno pogonjeni elektrohidraulički sustav (lijevo); proporcionalni elektrohidraulički sustav (desno)

2.2. Mjerni sustav

Mjerni sustav eksperimentalnog postava sastoji se od senzora tlaka, volumnog protoka, pozicije cilindra i struje. Mjerenje tlaka ulja u cilindru i u spremniku ulja ostvaruje se pomoću Hydac HDA 7446 senzora tlaka prikazanog na slici 4 lijevo. Dok je na slici 4 desno prikazan senzor tlaka ESI Protran PR3103 kojim se mjeri tlak napajanja klasičnog elektrohidrauličkog sustava. Mjerenje volumnog protoka ulja u komore cilindra ostvaruje se pomoću Hydac EVS 3106 senzora protoka prikazanog na slici 5.

Slika 4. Senzori tlaka: HDA 7446 [2] (lijevo); ESI Protran PR3103 [3] (desno)

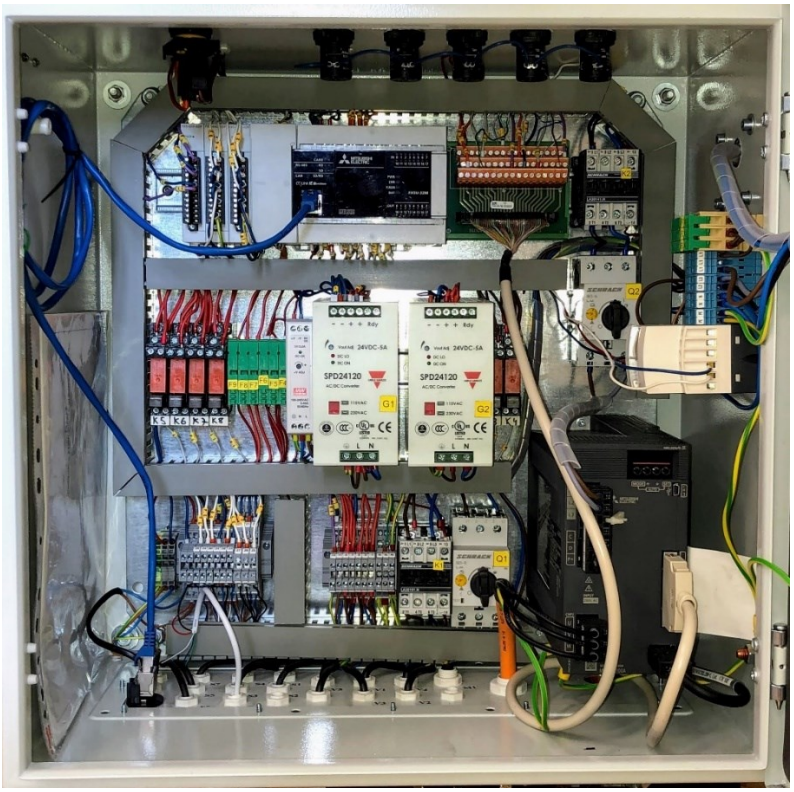
Slika 5. Senzor protoka [4]

Pozicija hidrauličkog cilindra mjeri se Micro-Epsilon WDS industrijskim senzorom prikazanim na slici 6 lijevo. Korištenjem podataka iz mjerenja pozicije, brzina gibanja cilindra se estimira u realnom vremenu. Translacijsko gibanje žice pričvršćene jednim dijelom na cilindar, a drugim dijelom namotane oko bubnja, mehanički se pretvara u rotacijsko gibanje koje se unutar senzora pretvara u signal u obliku napona. Mjerenje struje jednofaznog izmjeničnog motora koji pogoni pumpu proporcionalnog elektrohidrauličkog sustava vrši se pomoću Schneider Electric RMCA senzora struje prikazanog na slici 6 desno. Brzina cilindra je jedina veličina koja se ne mjeri, već se estimira. Estimacija se odvija u stvarnom vremenu pomoću algebarskog pristupa za filtriranje i estimiranje derivacije signala dizajniranog u Matlabu.

Slika 6. Senzor pomaka [5] (lijevo); senzor struje [6] (desno)

2.3. Upravljački sustav

U upravljačkom ormaru smještena je sva elektronika za upravljanje eksperimentalnim postavom. Cijelim sustavom upravlja Mitsubishi FX5U-32MT/ESS PLC na kojeg su spojene dvije 14-bitne A/D kartice i jedna 14-bitna D/A kartica. Svrha A/D (analogno-digitalna) kartice je prikupljanje analognih signala s mjernih senzora, dok se D/A (digitalno-analogna) kartica koristi za generiranje analognog upravljačkog signala na proporcionalnom ventilu i servo drive-u. Digitalni izlazi PLC-a koriste se za upravljanje DC relejima koji uključuju i isključuju ventile za zaobilaženje senzora protoka i ventil za uključivanje izravno pogonjenog elektrohidrauličkog sustava, ali i za paljenje i gašenje signalnih lampica. Upravljanje brzinom vrtnje servomotora vrši se pomoću servo pogona koji PLC-u preko serijske komunikacije prenosi podatke o brzini vrtnje servomotora, momentu i trenutnoj snazi. Na upravljačkom ormaru postavljen je usmjerivač (engl. *Router*) preko kojeg se odvija komunikacija između PLC-a i HMI-a (engl. *Human Machine Interface*) te PLC-a i serverskog računala. Na HMI-u se prikazuju sve procesne veličine i upravlja se eksperimentalnim sustavom. Na slici 7 prikazan je upravljački ormar.



Slika 7. Upravljački ormar

3. KOMUNIKACIJA SUSTAVA I KORISNIKA

3.1. Internet stvari

Tijekom posljednjeg desetljeća često se pojavljuje kratica IoT koja označava Internet of Things, odnosno Internet stvari. Taj je izraz po prvi puta korišten kao naslov prezentacije Kevina Ashtona, 1999 godine, u kojoj je predstavio ideju korištenja RFID-a (Radio frequency indentification) u nabavnom lancu. Općenito govoreći, IoT se odnosi na mrežno povezivanje svakodnevnih predmeta, koji su često opremljeni sveprisutnom inteligencijom. Na slici 8 prikazani su primjeri područja primjene. Implementacijom IoT tehnologije će se povećati sveprisutnost Interneta integrirajući svaki objekt za interakciju putem ugrađenih sustava, što dovodi do visoko distribuirane mreže uređaja koji komuniciraju s ljudima kao i s drugim uređajima [7].

Slika 8. Internet stvari (IoT) [8]

3.2. Potreba za IoT

Danas su računala i prema tome Internet gotovo u potpunosti na usluzi ljudima da dobiju informacije. Gotovo sve podatke dostupne na Internetu ljudi su prvo prikupili, a onda i pohranili: tipkanjem, pritiskom na tipku za snimanje, digitalnom fotografijom ili skeniranjem barkôda. Mnogi klasični dijagrami Interneta uključuju poslužitelje i usmjerivače i tako dalje, no izostavljaju jednu od najbitnijih stavki, čovjeka. Problem je što ljudi imaju ograničenu pažnju, vrijeme i točnost, što znači da nisu baš dobri u prikupljanju podataka o stvarima u stvarnom svijetu. Ljudi su fizička bića, isto kao i naša okolina. Naša ekonomija, društvo i

opstanak se ne temelje na idejama ili informacijama, nego na stvarima. Današnja informatička tehnologija toliko ovisi o podacima koje su stvorili ljudi, da računala znaju više o idejama nego o stvarima. Da imamo računala koja koriste podatke koje su prikupili bez ljudske pomoći mogli bismo sve pratiti i prebrojati te uvelike smanjiti otpad, gubitke i troškove, znali bismo kada stvari treba zamijeniti, popraviti ili opozvati. Cilj IoT tehnologija je omogućiti računalima samostalno prikupljanje informacija koristeći RFID i senzorsku tehnologiju te da u svom tom mnoštvu podataka zaključuju i prepoznaju svijet bez ograničenja koja uvodi čovjek kao posrednik.

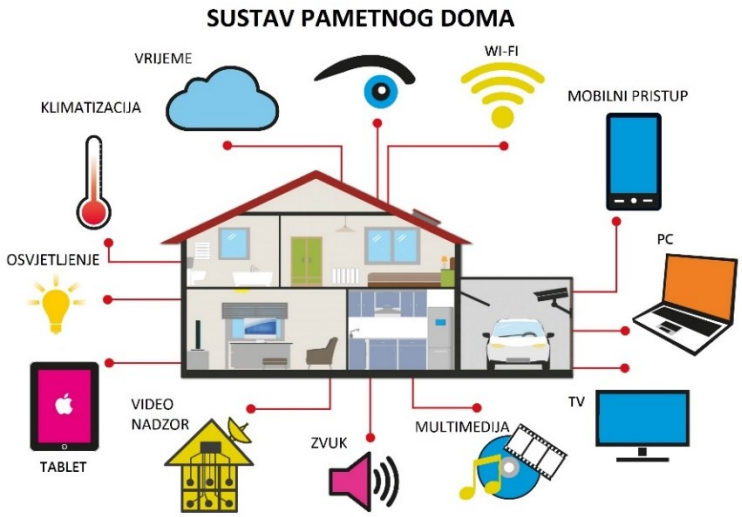
3.3. Primjena IoT tehnologija

Mogućnosti primjene IoT tehnologija gotovo su beskonačne, mogućnost da se svakodnevnim objektima integrira inteligencija dopušta nam nove načine primjene i već postojećih objekata i tehnologija. U svim tim mogućnostima bitno je istaknuti najvažnije [9].

3.3.1. Pametan dom

Zasigurno najpoznatija primjena IoT tehnologije očituje se u konceptu pametnih domova. Pametan dom možemo definirati kao koncept povezanih uređaja koji se nalaze u prostoru prebivanja, a imaju mogućnost spajanja na Internet, čime se omogućuje udaljena kontrola nad uređajima te mogućnost za međusobnu komunikaciju između uređaja. Takvu definiciju najbolje je prikazati uz primjer. Korisnik u bilo kojem trenutku pristupa centralnoj jedinici doma, serveru, putem Interneta. Na centralnu jedinicu spojeni su svi pametni uređaji, putem lokalnog NET kabela, WiFi mreže, energetskih kablova, Bluetootha i raznih drugih metoda. Razlog korištenja centralne jedinice je povećana pouzdanost rada sustava te bolja sigurnost od hakiranja. Najčešće korišteni pametni uređaji su pametna svjetla, termostati, klima uređaji, grijači, uređaji za multimediju te ulazna vrata. Tipični scenarij korištenja pametnih uređaja je mogućnost automatizacije procesa. Na primjer, pri otvaranju ulaznih vrata, korisnik biva prepoznat senzorima autentifikacije ugrađenim u vratima te se pokreće „rutina“ koju je korisnik prethodno definirao. Ta rutina može uključivati automatsko paljenje svjetla u domu, postavljanje grijanja/hlađenja prostorije, uključivanje uređaja za multimediju ili raznih kućanskih aparata te automatsko isključivanje alarmnog sustava. Tu možemo primijetiti najveću prednost IoT tehnologije, a to je komunikacija uređaja s uređajem, čime se omogućuje da korisnik jednom zada svoje osobne postavke, a pametni uređaji će samostalno,

komunicirajući jedan s drugim kako bi sakupili potrebne podatke za odluku, upravljati mikroklimom doma. Također, bitno je naglasiti i korištenje umjetne inteligencije u pametnom domu. U svrhu daljnjeg olakšavanja korisniku, uređaji mogu, uz međusobnu komunikaciju s centralnom jedinicom, naučiti ponašanje korisnika i samostalno prilagođavati sami sebe kako bi osigurali što veću uštedu energije, a da pritom izvrše sve zadane naredbe. Prikaz mogućnosti pametnog doma dan je na slici 9.



Slika 9. Prikaz pametnog doma [10]

3.3.2. Nosiva tehnologija

Kontinuirani razvoj mikro-kontrolera i senzora omogućio je u zadnjih desetak godina eksploziju razvoja nosive tehnologije. Pod pojmom nosiva tehnologija smatraju se svi pametni elektronički uređaji koji se nose blizu ili na površini same kože sa zadatkom da mjere, analiziraju i prenose informacije o korisniku, poput vitalnih znakova i podataka o okolini. Najraširenija nosiva tehnologija zasigurno su pametni satovi i fitnes narukvice, koji su odličan primjer primjene Interneta stvari u svakodnevnom životu, i to na način da „stvari“ poput elektronike, senzora i softvera, omogućuju automatsku razmjenu do sada teško mjerljivih podataka te njihovu analizu, s raznim fitnes aplikacijama, drugim povezanim uređajima i samim korisnikom [11]. Primjer fitnes narukvice dan je na slici 10 lijevo. Budućnost nosive tehnologije očituje se u raznim medicinskim implantatima, pametnim tkaninama te raznim pomagalima vida i sluha, od kojih su neki prikazani na slici 10 desno.

Slika 10. Fitnes narukvica [12] (lijevo); nosive tehnologije [13] (desno)

3.3.3. Pametni gradovi

Idući korak integracije Interneta stvari u svakodnevni život se manifestira unutar urbanih gradova kao pojam pametni grad. Ideja pametnih gradova je koristeći informacijske i komunikacijske tehnologije (engl. *Information and Communication Technologies* – ICT) implementirati inteligentnu mrežu povezanih objekata i strojeva koji prenose podatke koristeći bežičnu tehnologiju. Svi prikupljeni podaci se analiziraju i obrađuju u realnom vremenu te omogućuju bolju kvalitetu života na način da se poboljša distribucija energije, smanje prometne gužve, poveća sigurnost građana ili čak poboljša kvaliteta zraka [14]. Ideja pametnog grada prikazana je na slici 11.

Neki od načina integracije IoT tehnologije u gradove:

- Pametni semafori primaju podatke od senzora na raskrižjima i od prolazećih automobila te sukladno situaciji u stvarnom vremenu prilagođavaju vrijeme trajanja zelenog svjetla [15].
- Sve većim brojem električnih automobila, a lošijom infrastrukturom električnih punionica, povezani automobili mogu međusobno komunicirati te automatski preusmjeriti vozača na slobodnu punionicu. Također, postoji i mogućnost javljanja slobodnih parkirnih mjesta između povezanih automobila [16].

- Pametne otpadne kante u stvarnom vremenu prikazuju tvrtkama za gospodarenje otpada količinu smeća te automatski reguliraju vozne rute prikupnih kamiona [17].
- Implementacijom nove generacije mobilne mreže, koja će imati daleko gušći broj tornjeva, omogućuje se pouzdano povezivanje IoT uređaja koje će dodatno poboljšati mogućnosti pametnih gradova [18].

Slika 11. Pametni grad [19]

3.3.4. Industrijski Internet stvari

Poznatiji pod kraticom IIoT (engl. *Industrial Internet of Things*), Industrijski Internet stvari odnosi se na senzore, instrumente, uređaje i strojeve u proizvodnji koji su međusobno povezani na industrijska računala, čime se omogućuje prikupljanje, razmjena i analiza podataka. Ideja pametne industrije prikazana je na slici 12. Ti podaci su nam bitni jer omogućuju fino podešavanje proizvodnih sustava i parametara kako bi se skratilo vrijeme proizvodnje, smanjio škart, povećala efikasnost i smanjila potrošnja energije. Također, implementacijom RFID oznaka na proizvode, otvara se mogućnost praćenja svakog koraka proizvodnje, odnosno svaki proizvod će imati podatke o svakom procesu kroz koji je prošao te informacije o trajanju procesa i njegovoj kvaliteti. Osim prednosti koje RFID pruža samom proizvođaču kako bi dobio detaljan uvid u kvalitetu proizvodnog procesa, i sam kupac ima mogućnost uvida u kojem je procesu izrade naručeni proizvod. Uz sve navedeno, ta tehnologija omogućuje i izmjene "*on the fly*", odnosno kupac se može tijekom proizvodnog procesa odlučiti za neku promjenu, što bi u

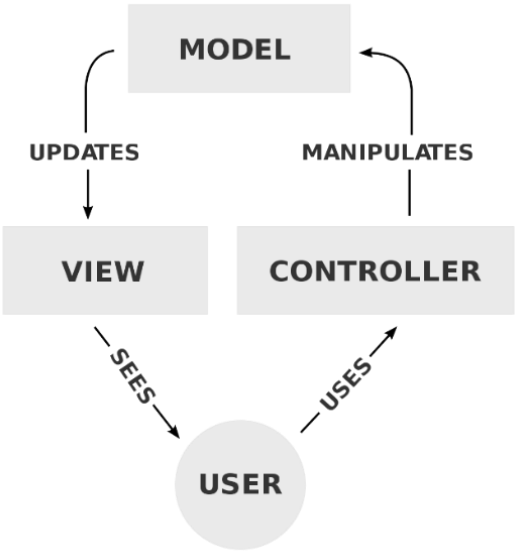
klasičnoj proizvodnji značilo kašnjenja i ponovne izrade proizvoda. Uporabom RFID tehnologije, takav se podatak može lagano upisati u sam proizvod tijekom proizvodnog procesa te omogućiti neometani nastavak proizvodnje uz željene izmjene.

Pored navedenih primjena Interneta stvari u industriji sve više do izražaja dolazi prediktivno održavanje, a uporabom IoT-a omogućuju se bolji rezultati. Razlog za sve češću implementaciju prediktivnog održavanja je mogućnost predviđanja kvarova i prestanka rada opreme. Strategija koja se koristi kod prediktivnog održavanja uključuje prikupljanje podataka o stanju sustava preko senzora ili bežičnih mrežnih sustava, odnosno strojevi se opremaju raznim senzorima koji omogućuju mjerenje ključnih varijabli iz kojih se analizom prikupljenih podataka predviđa „zdravlje“ samoga stroja. Osim senzora, strojevima se omogućuje i međusobna komunikacija, odnosno M2M (engl. *Machine to machine*), stroj-stroj komunikacija.

Slika 12. Industrijski Internet stvari (IIoT) [20]

3.4. Aplikacija Web2Py

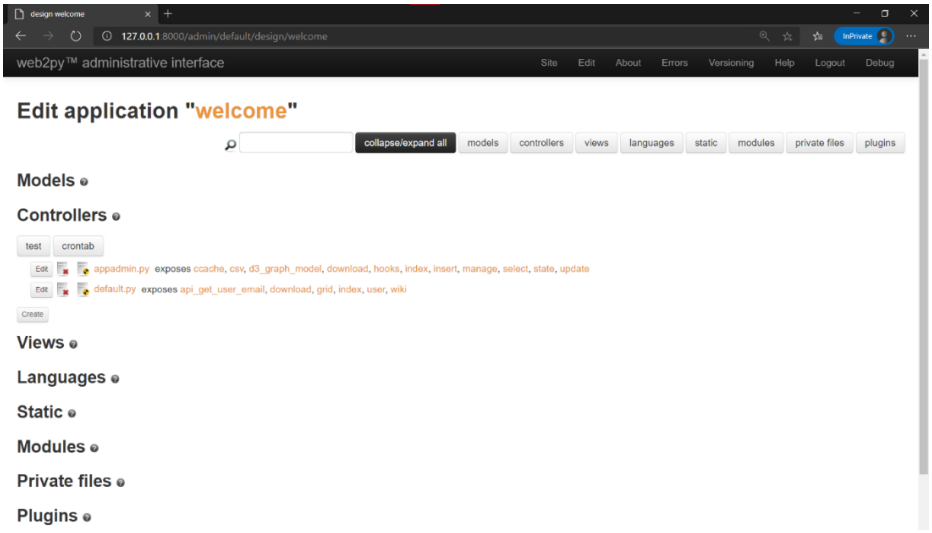
Web2Py je besplatni mrežni okvir otvorenog kôda za brzi razvoj sigurnih aplikacija vođenih bazama podataka, a napisan je u Python-u te se u njemu i programira. Web2Py je *full-stack* okvir, što znači da sadržava sve komponente potrebne za izgradnju potpuno funkcionalnih web aplikacija. Konstruiran je s idejom da vodi programere u smjeru dobrih programerskih praksi, poput korištenja MVC (engl. *Model View Controller*) obrasca, odnosno model-pogled-kontroler obrazac. To postiže odvajanjem predstavljanja podataka (model/*Model*) od prezentacije podataka (pogled/*View*), a također i od logike aplikacije i tijeka rada (kontroler/*Controller*). Web2Py pruža biblioteke koje pomažu programerima u konstruiranju, implementiranju i testiranju svakog od ova tri dijela zasebno te ih potiče da zajedno rade [22]. Na slici 13 prikazan je izgled MVC obrasca te način međudjelovanja s korisnikom.



Slika 13. Prikaz MVC obrasca [21]

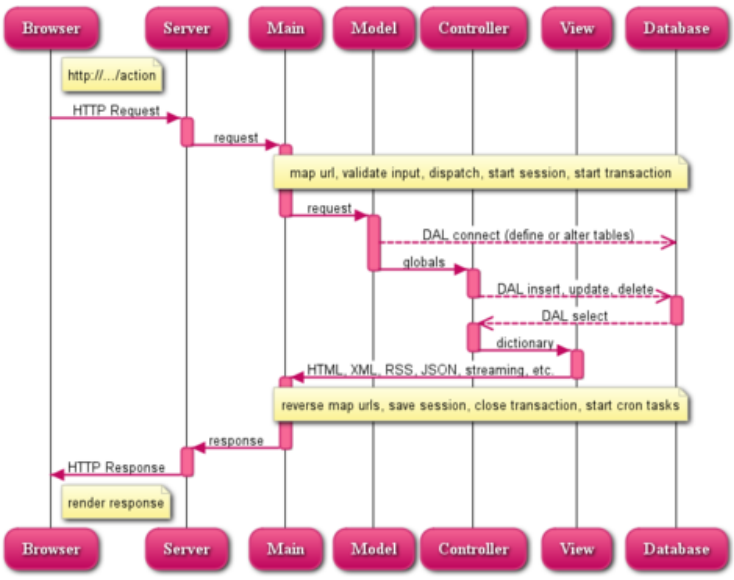
Opis MVC obrasca:

- Model je centralna komponenta web aplikacije te izražava ponašanje aplikacije u domeni problema, a objekti u modelu se koriste za dohvaćanje i spremanje podataka iz baze podataka.
- Pogled je dio aplikacije koji pomaže u prikazu informacija iz modela krajnjim korisnicima.
- Kontroler je dio aplikacije koji upravlja interakcijom korisnika, može čitati podatke s pogleda, kontrolirati unos korisnika i slati ulazne podatke određenom modelu.



Slika 14. Web2Py sučelje

Web2Py pruža korisniku pregledno sučelje bazirano na MVC obrascu prikazanom na slici 14. Osim modela, kontrolera i pogleda, Web2Py također nudi i podršku za jezike, gdje korisnik može implementirati prijevode na različite jezike za svoju aplikaciju, statičke podatke koji podrazumijevaju slike, video zapise, odnosno podatke koji se ne mijenjaju. Također nudi mogućnost pohranjivanja Python modula u koji spadaju moduli koje korisnik sam kreira te bilo koji Python paket ili modul koji korisnik želi izolirati od ostatka Web2Py okoliša. Uza sve navedeno, Web2Py pruža korisniku privatni folder u koji može spremiti informacije koje koristi, ali ne želi prikazati ostalim korisnicima.



Slika 15. Web2Py tok podataka [22]

Na slici 15 prikazan je Web2Py tok podataka MVC obrasca:

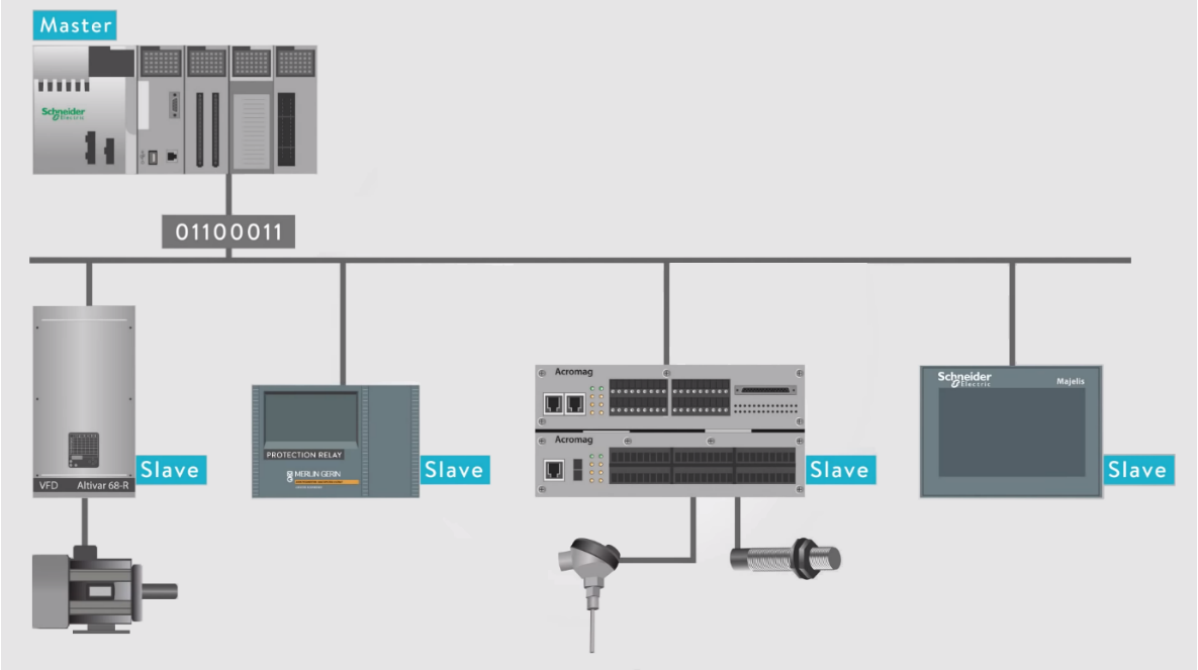
- *Server* može biti ugrađeni Web2Py web server ili bilo koji poslužitelj treće strane, poput Apache-a.
- *Main* je glavna WSGI (engl. *Web Server Gateway Interface*) aplikacija, odnosno ona obavlja sve uobičajene zadatke i obavija korisničke aplikacije. Bavi se kolačićima, sesijama, transakcijama, URL usmjeravanjem i obrnutim usmjeravanjem i slanjem.
- Komponente *Models*, *Views* i *Controller* čine korisničku aplikaciju.
- Više aplikacija može biti smješteno u istoj Web2Py instanci.
- Isprekidane strelice predstavljaju komunikaciju s bazom podataka. Upiti za bazu podataka mogu se napisati u čistom SQL (engl. *Structured Query Language*) formatu, što se ne preporuča, ili pomoću DAL (engl. *Database Abstraction Layer*) Web2Py formata, što se preporuča iz razloga da aplikacija ne ovisi o određenom tipu baze podataka.
- *Model* uspostavlja vezu s bazom podataka i komunicira s *Controller-om*. S druge strane *Controller* komunicira s *View-om* kako bi prikazao podatke.
- Dispečer preslikava traženi URL kako je naveden u HTTP odgovoru na poziv funkcije u kontroleru, a izlaz funkcije može biti niz ili *hash* tablica. *Hash* tablica je vrsta podatkovne strukture gdje se podaci spremaju u formatu polja, gdje svaka vrijednost podataka ima svoju jedinstvenu vrijednost indeksa, a pristup podacima postaje vrlo brz ako znamo indeks željenih podataka.
- Podatke generira *View*. Ako posjetitelj zatraži HTML stranicu (zadano), podaci se generiraju u HTML stranicu, no posjetitelj može istu stranicu zatražiti i u formatima poput XML, JSON, RSS, CSV, RTF koji su podržani ili u bilo kojem prilagođenom protokolu kojeg korisnik mora sam implementirati.
- Web2Py također automatski obrađuje sesije i kolačiće sesija, a kada je transakcija izvršena, sesija se također pohranjuje, osim ako nije drukčije naznačeno.

Bitno je objasniti što su sesije i kolačići unutar Web2Py aplikacije. Pod terminom sesija (engl. *session*) misli se na instancu klase *Storage* koja se nalazi unutar glavnog Web2Py modula *gluon*, u kojem su definirani svi Web2Py-evi objekti poput *request*, *response*, *session* i *class helpers*, *validators*, DAL i mnogi drugi. Varijable pohranjene u jednom *session* objektu sadrže

informacije samo jednog korisnika, a dostupne su na svim stranicama unutar aplikacije. Poslužitelj radi novi *session* objekt za svakog novog korisnika, a briše se nakon što sesija istekne (npr. zbog predugog perioda neaktivnosti). Kolačići, odnosno engl. *cookies* spadaju pod objekt *response* te označuju mali dio podataka koji web preglednik pohranjuje na računalo korisnika tijekom pregledavanja web aplikacije. Dizajnirani su da budu pouzdan mehanizam za pamćenje podataka sa statusom (npr. stavki dodanih u košaricu u Internetskoj trgovini) ili za bilježenje korisnikova pregledavanja (uključujući klikanje određenih gumba, prijavu ili bilježenje stranica koje su posjećene u prošlosti). Također, mogu se koristiti i za pamćenje podataka koje je korisnik prethodno unio u polja obrasca, poput imena, adresa, lozinki i brojeva platnih kartica.

3.5. Modbus protokol

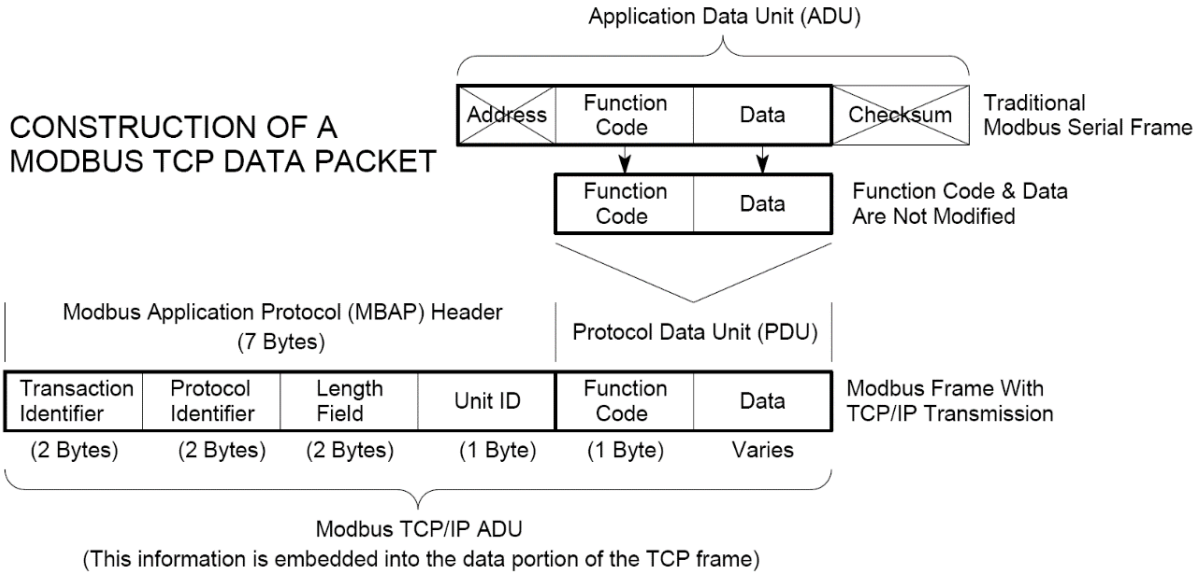
Kako bi se ostvarila valjana komunikacija između uređaja različitih proizvođača, uvedeni su komunikacijski protokoli. Kompanija Modicon je 1979. godine za vlastite potrebe razvila serijski komunikacijski protokol Modbus. Ubrzo se protokol proširio te je postao industrijski standard za prijenos analogno-digitalnih podataka između kontrolnih i mjernih uređaja. Velika prednost Modbus protokola leži u tome da je *open source* [23]. Modbus protokol temelji se na *Master-Slave* (u slučaju Modbus RTU) i *Client-Server* (za slučaj Modbus TCP/IP) komunikaciji u kojoj *Master*/*Client* uređaj pokreće prijenos podataka. Ostali spojeni uređaji (*Slaves*/*Servers*) odgovaraju dostavljanjem traženih podataka glavnom računalu ili poduzimanjem radnje tražene u upitu. *Slave* uređaj je bilo koji periferni uređaj koji obrađuje informacije i šalje svoj izlaz *Masteru* pomoću Modbusa. Tipični *Slave* uređaji su I/O pretvarač, ventil, mrežni pogon ili drugi mjerni uređaj. *Master* uređaj je tipično *host* računalo na kojem se izvodi odgovarajući program. Na slici 16 prikazan je primjer spojenih uređaja.



Slika 16. Primjer Modbus protokola [24]

Modbus protokol je u početku koristio ASCII kôd za započinjanje i završavanje poruka te se i danas mogu pronaći uređaji koji koriste taj standard. Razvojem SCADA sustava (engl. *Supervisory Control And Data Acquisition*) sve se više koristio Modbus RTU protokol (engl. *Remote Terminal Unit*) koji se temelji na binarnom kôdiranju i CRC (engl. *Cyclic Redundancy Check*) provjeri grešaka te vremenskim razmacima tišine za uokvirivanje poruka. Danas je Modbus RTU i dalje najkorišteniji Modbus protokol, no sve više njegovu ulogu preuzima moderna inačica koja se naziva Modbus TCP/IP te se temelji na komunikaciji *client-server* umjesto *master-slave*. U osnovi, to je Modbus RTU protokol s TCP sučeljem koje radi na Internetu. Struktura poruke je aplikacijski protokol koji definira pravila za organiziranje i tumačenje podataka neovisno o mediju za prijenos podataka. TCP/IP označuje engl. *Transmission Control Protocol/Internet Protocol* te predstavlja medij za prijenos Modbus poruka. Jednostavno rečeno, on omogućuje razmjenu blokova binarnih podataka između računala. To je također svjetski standard koji služi kao temelj za *World Wide Web*. Primarna funkcija TCP-a je osigurati da su svi paketi podataka pravilno primljeni, dok IP osigurava da su poruke ispravno adresirane i usmjerene. Bitno je napomenuti da je kombinacija TCP/IP samo transportni protokol i ne definira značenje podataka niti kako ih treba interpretirati, što je zadatak aplikacijskog protokola, Modbus-a.

Dakle, Modbus TCP/IP koristi TCP/IP i Internet za prijenos podataka Modbus-ove strukture poruka između kompatibilnih uređaja. Odnosno, Modbus TCP/IP kombinira fizičku mrežu (Internet), s mrežnim standardom (TCP/IP) i standardnom metodom predstavljanja podataka (Modbus kao aplikacijski protokol). U osnovi, Modbus TCP/IP poruka je jednostavno Modbus komunikacija uvrštena u TCP/IP zaglavlje. U odnosu na Modbus RTU koji se sastoji od adrese, funkcijskog kôda, podataka te CRC-a (engl. *Cyclic Redundancy Check*), Modbus TCP/IP zadržava samo funkcijski kôd i podatke, te dodaje na početak poruke MBAP zaglavlje (engl. *Modbus Application Header*) kao što možemo vidjeti na slici 17.



Slika 17. Konstrukcija Modbus TCP/IP paketa [23]

Pod *Protocol Data Unit* nalaze se funkcijski kôdovi te podaci koji se šalju. Modbus koristi memorijske registre kako bi konfigurirao, nadgledao i upravljao ulazno/izlaznim uređajem. Svaki Modbus uređaj ima svoju kartu registara te ju je potrebno proučiti za pravilno razumijevanje njegovog rada. Model podataka Modbus-a veoma je jednostavne strukture i razlikuje samo četiri osnovne vrste podataka prikazane u tablici 1.

Tablica 1. Tipovi Modbus objekta

Tip objekta	Dopuštenje	Veličina	Adresni prostor	Adresa podataka
Diskretni izlaz	Čitanje-pisanje	1 bit	00001 - 09999	0000 – 270E
Diskretni ulaz	Samo za čitanje	1 bit	10001 - 19999	0000 – 270E
Ulazni registar	Samo za čitanje	16 bita	30001 - 39999	0000 – 270E
Izlazni registar	Čitanje-pisanje	16 bita	40001 - 49999	0000 – 270E

Osim podataka u Modbus protokolu definirani su i funkcijski kôdovi. To su jednostavni numerički kôdovi koji govore *Slave*-u kojoj tablici podataka treba pristupiti (diskretni ulaz ili registar) te treba li taj podatak upisati ili samo pročitati. Najčešći funkcijski kôdovi prikazani su u tablici 2.

Tablica 2. Najčešći funkcijski kôdovi

Kôd	Funkcija
01 (01 hex)	Pročitaj diskretne izlaze
05 (05 hex)	Zapiši jedan diskretni izlaz
15 (0F hex)	Zapiši više diskretnih izlaza
02 (02 hex)	Pročitaj diskretne ulazne kontakte
04 (04 hex)	Pročitaj analogne ulazne registre
03 (03 hex)	Pročitaj analogne izlazne registre
06 (06 hex)	Zapiši jedan analogni izlazni registar
16 (10 hex)	Zapiši više analognih izlaznih registara

Bitno je napomenuti da svi tipovi podataka imaju iste podatkovne adrese, na primjer prvi izlazni registar, broj 40001, ima podatkovnu adresu 0000, stoga trebamo prije podataka postaviti funkcijski kôd 03 kako bi uređaj znao da se radi o lokaciji registra 40001.

Na početku svake Modbus TCP/IP poruke nalazi se MBAP zaglavlje (engl. *Modbus Application Header*) u kojem su svi podaci za identificiranje [25].

MBAP zaglavlje dugo je 7 bajta i sastoji se od:

- **Identifikatora prijenosa (2 bajta)** – Ovo identifikacijsko polje koristi se za uparivanje transakcija kada klijent šalje više poruka duž iste TCP veze bez čekanja na prethodni odgovor.
- **Identifikatora protokola (2 bajta)** – Ovo polje je uvijek 00 00 za Modbus usluge, a ostale vrijednosti su rezervirane za buduća proširenja.
- **Duljina (2 bajta)** – Ovo polje predstavlja broj bajtova preostalih polja i uključuje bajt identifikatora jedinice, bajt funkcionalnog kôda i podatkovna polja.
- **Identifikator jedinice (1 bajt)** – Ovo polje se koristi za identificiranje udaljenog poslužitelja koji se nalazi na mreži koja nije TCP/IP (za serijsko premošćivanje). U tipičnoj aplikaciji poslužitelja Modbus TCP/IP, ID jedinica se postavlja na 00 ili FF, poslužitelj ga ignorira i jednostavno odjekuje u odgovoru.

Kako bi se lakše shvatila Modbus TCP/IP poruka, bit će objašnjena na jednostavnom primjeru poruke: *0001 0000 0006 11 03 006B 0003*

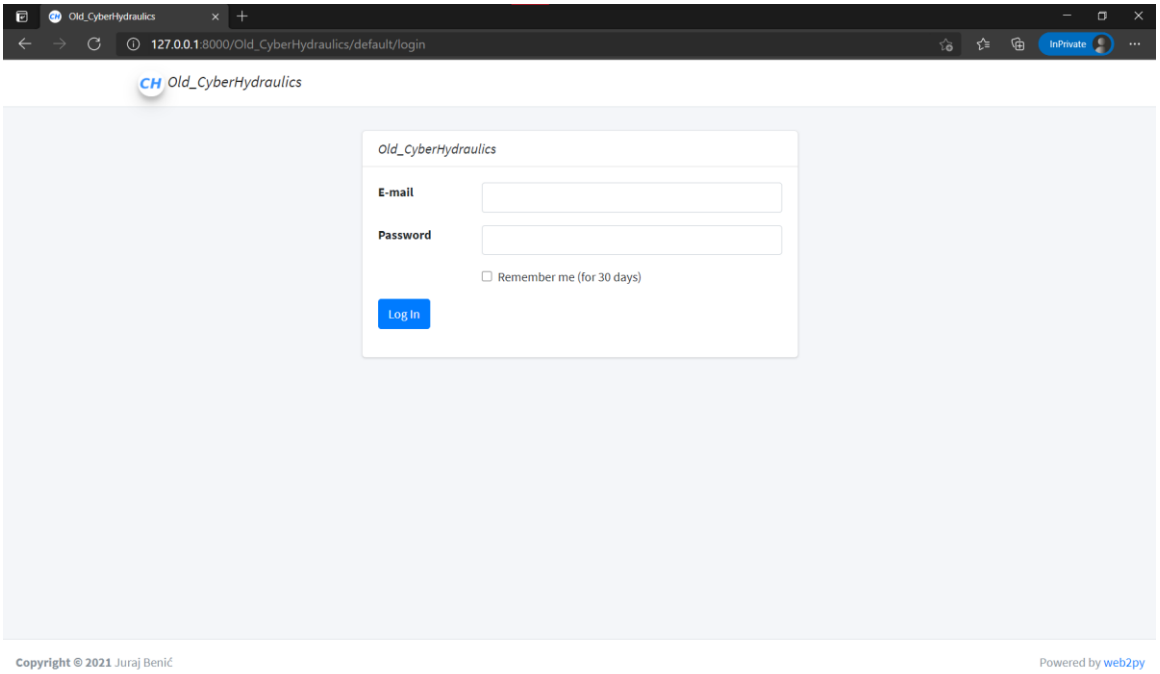
- 0001: Identifikator prijenosa
- 0000: Identifikator protokola
- 0006: Duljina poruke (slijedi šest bajtova)
- 11: Identifikator jedinice (17 = 11 hex)
- 03: Funkcijski kôd (pročitaj analogne izlazne registre)
- 006B: Podatkovna adresa prvog traženog registra (40108-40001 = 107 =6B hex)
- 0003: Ukupan broj traženih registara (pročitaj 3 registra od 40108 do 40110)

4. PROGRAMSKO RJEŠENJE APLIKACIJE SUSTAVA

Postojeća aplikacija je platforma stvorena od strane asistenta J. Benića kako bi se koristila za nastavne i eksperimentalne svrhe. U ovom poglavlju biti će prikazane dvije aplikacije, za opis postojeće aplikacije korištena je aplikacija *Old_CyberHydraulics*, dok se u ostatku poglavlja koristi nadograđena aplikacija *CyberHydraulics.*

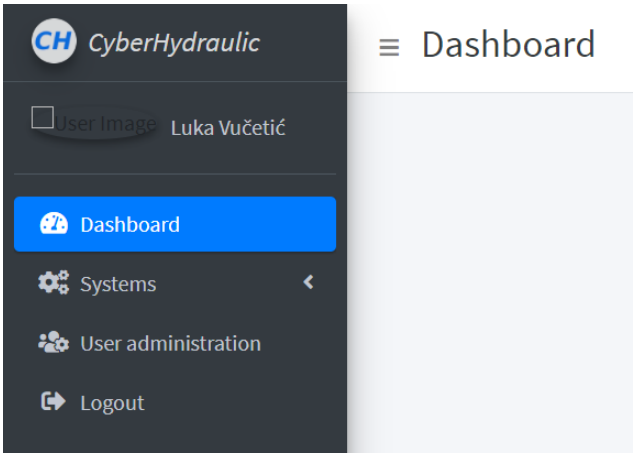
4.1. Opis postojeće aplikacije

Svrha aplikacije je platforma za pregled i upravljanje pneumatskim i hidrauličkim sustavima putem web preglednika te prikupljanje podataka u bazu podataka. Na slici 18 prikazan je izgled pri prvom pokretanju aplikacije. Korisnik aplikaciji može pristupiti tek nakon uspješnog unošenja e-maila i lozinke kako bi potvrdio svoj identitet. Ukoliko su uneseni e-mail i/ili šifra neispravni, korisnika se na to upozorava.



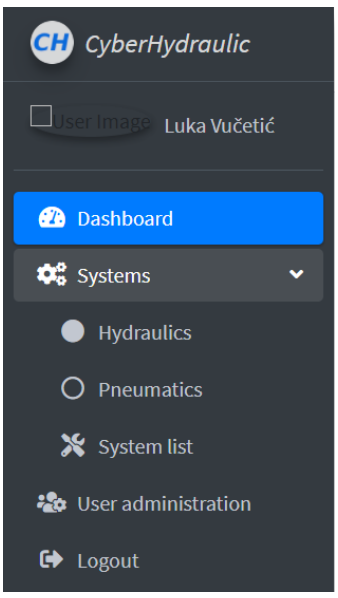
Slika 18. Prikaz stare aplikacije

Nakon uspješne prijave dolazimo na početnu stranicu *Dashboard* koja služi kao rezervirano mjesto za buduće primjene. Navigacija kroz aplikaciju vrši se pomoću bočnog izbornika prikazanog na slici 19, koji je u svakom trenutku dostupan.



Slika 19. Prikaz bočnog izbornika

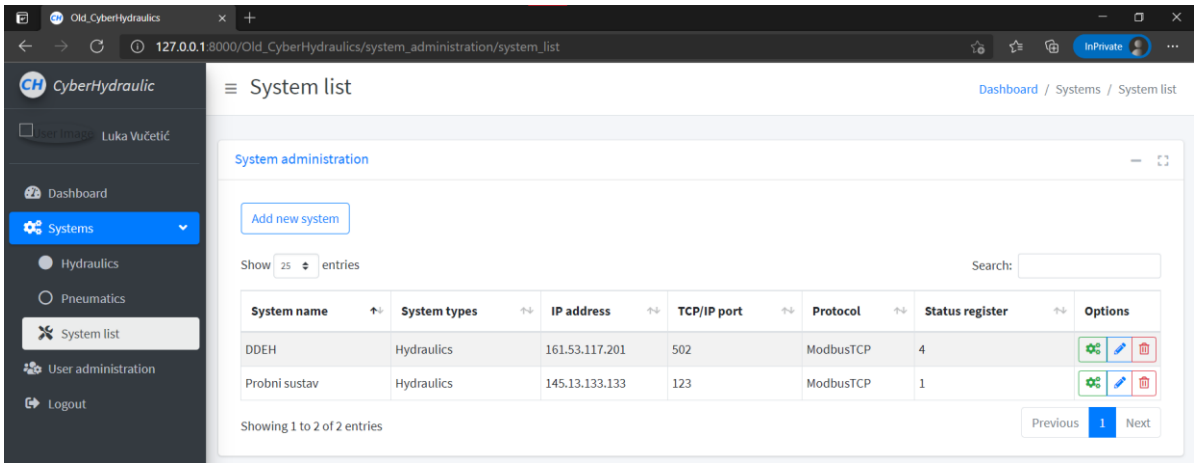
Prikazane su informacije o trenutnom korisniku putem korisničke slike te korisničkog imena, ispod čega se nalaze dostupne stranice *Dashboard*, *Systems*, *User administration* te *Logout*. Pritiskom na *Systems* otvara se padajući izbornik, prikazan na slici 20, koji nudi odabir sustava *Hydraulics* ili *Pneumatics* te *System list* pod kojim se nalazi popis sustava. Ispod *Systems* se nalazi stranica *User administration* na kojoj se mogu dodavati i uređivati korisnici i grupe te se naposljetku nalazi *Logout* čime se korisnik odjavljuje s aplikacije.



Slika 20. Padajući izbornik *Systems*

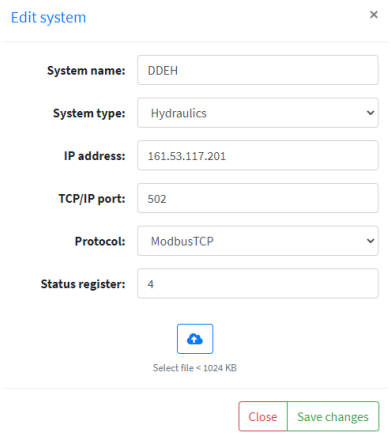
4.1.1. Izborna kategorija System list

U aplikaciji je zamišljeno povezivanje raznih hidrauličkih i pneumatskih sustava. Kako bi se sustav povezao na aplikaciju potrebno ga je definirati na stranici *System list* prikazanoj na slici 21.



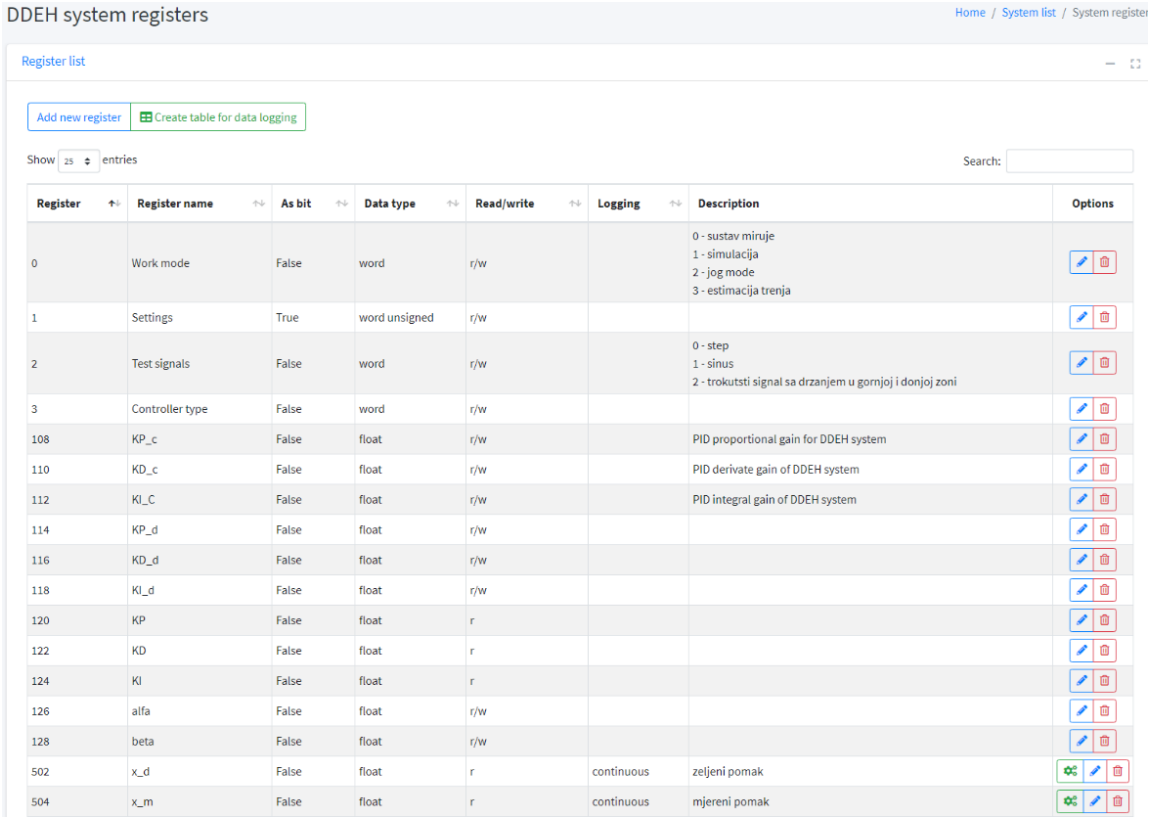
Slika 21. Izborna kategorija *System list*

Sa slike 21 vidi se da su definirana dva sustava, „DDEH“ i „Probni sustav“ te je za svaki prikazano njegovo ime, tip sustava, IP adresa, TCP/IP port, protokol komunikacije te broj statusnog registra. Također se za svaki sustav nude opcije poput uređivanja liste registara, uređivanje osnovnih podataka sustava te brisanje sustava. Odabirom na *Add new system* otvara se prozor za unos podataka bitnih za stvaranje novog sustava prikazanog na slici 22. Potrebno je dati sustavu ime, odabrati tip sustava između *Hydraulics* i *Pneumatics*, unesti *IP adresu*, *TCP/IP port*, odabrati kao protokol *ModbusTCP*, unijeti statusni registar te učitati sliku sustava.



Slika 22. Definiranje novog sustava

Za tako definirani sustav potrebno je unijeti listu registara kojima se upravlja putem ModbusTCP protokola. Na slici 23 prikazana je već definirana lista registara za DDEH sustav. Kako bi registar bio u potpunosti definiran potrebno je unijeti broj registra, njegovo proizvoljno ime za lakše snalaženje, odabrati *True/False* ako je registar kao bit ili nije, o kojem tipu podatka se radi (*word, word unsigned, double word, float*), može li se u registar vrijednost upisivati i čitati ili samo čitati (*r, r/w*), zapisuju li se podaci u bazu podataka te kratki opis njegove funkcije.



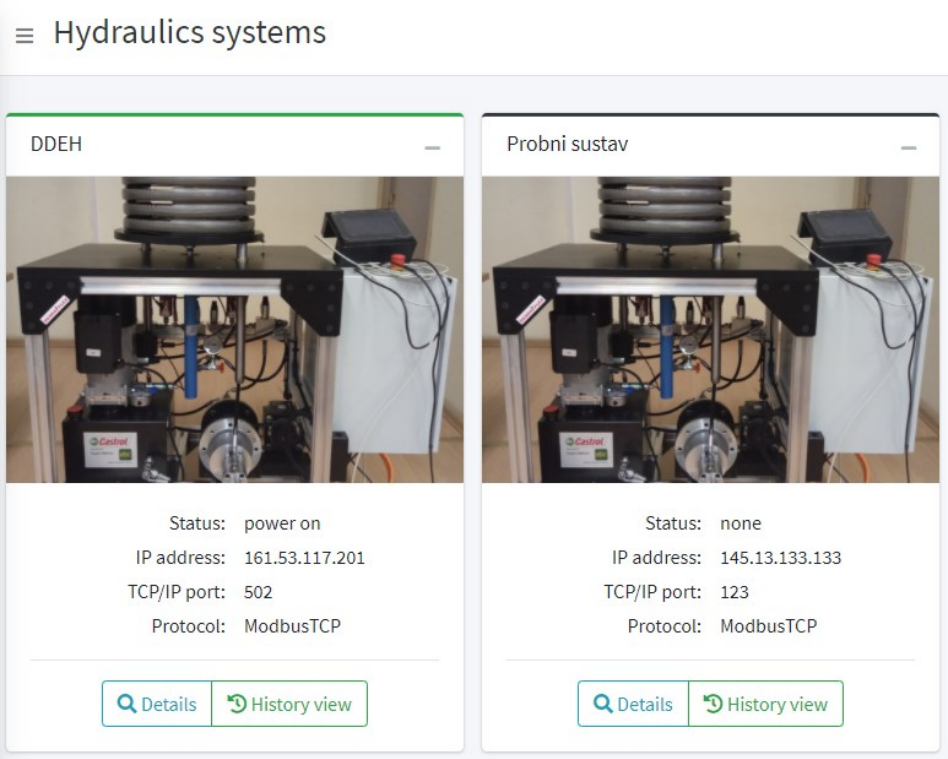
Register	Register name	As bit	Data type	Read/write	Logging	Description	Options
0	Work mode	False	word	r/w		0 - sustav miruje 1 - simulacija 2 - jog mode 3 - estimacija trenja	
1	Settings	True	word unsigned	r/w			
2	Test signals	False	word	r/w		0 - step 1 - sinus 2 - trokutsti signal sa drzanjem u gornjoj i donjoj zoni	
3	Controller type	False	word	r/w			
108	KP_c	False	float	r/w		PID proportional gain for DDEH system	
110	KD_c	False	float	r/w		PID derivate gain of DDEH system	
112	KI_C	False	float	r/w		PID integral gain of DDEH system	
114	KP_d	False	float	r/w			
116	KD_d	False	float	r/w			
118	KI_d	False	float	r/w			
120	KP	False	float	r			
122	KD	False	float	r			
124	KI	False	float	r			
126	alfa	False	float	r/w			
128	beta	False	float	r/w			
502	x_d	False	float	r	continuous	zeljeni pomak	
504	x_m	False	float	r	continuous	mjereni pomak	

Slika 23. Popis registara

4.1.2. Izborna kategorija Hydraulics systems

Prethodno definirani sustavi s odgovarajućim registrima prikazani su pod kategorijama *Hydraulics* i *Pneumatics*. Eksperimentalni sustav "DDEH" definiran u 2. poglavlju sastoji se od dva hidraulička motora te se iz tog razloga nalazi u kategoriji *Hydraulics*, isto kao i virtualni sustav stvoren za potrebe testiranja "Probni sustav". Kategorija *Pneumatics* je prazna te služi kao *placeholder* za buduće projekte i radove koji će koristiti pneumatske sustave. Na slici 24 prikazan je izgled kategorije *Hydraulics* s definiranim sustavima. Svaki sustav prikazan je na

svojoj dinamičnoj kartici čiji gornji rub mijenja boju ovisno u stanju sustava, a na slici se može vidjeti da je "DDEH" sustav upaljen te svijetli zelenom bojom, dok za "Probni sustav" ne postoji podatak o njegovom stanju pa svijetli sivom bojom. Boje su važan aspekt uspješnog i ugodnog korisničkog iskustva kada se koriste sukladno s tekstualnim informacijama.



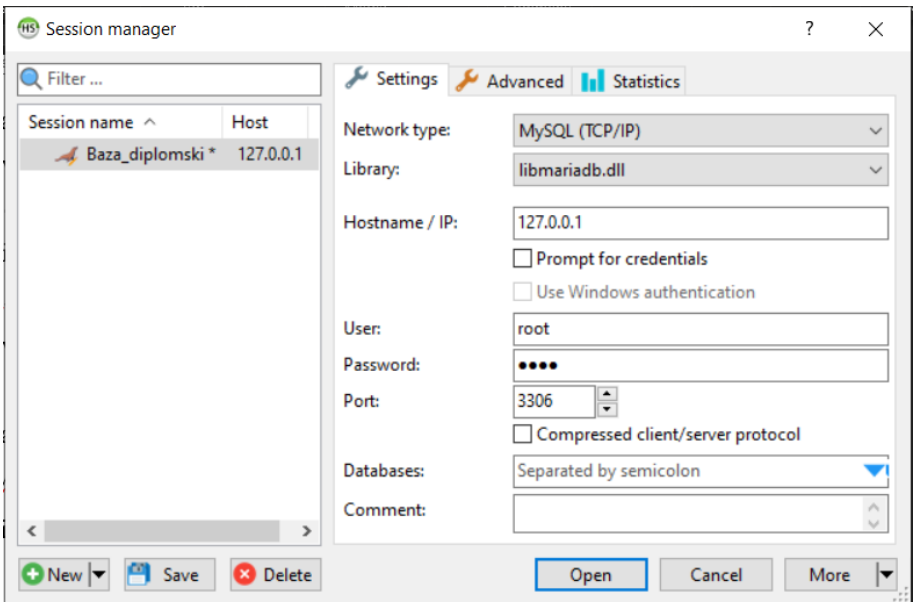
Slika 24. Izborna kategorija *Hydraulics systems*

Za svaki je sustav moguće vidjeti njegovo trenutno stanje odabirom na *Details* gdje su prikazani podaci sustava u realnom vremenu te povijesni pregled zabilježenih podataka odabirom na *History view*. Obje mogućnosti će se detaljnije objasniti u narednim potpoglavljima.

4.1.3. Baza podataka

Svi prikupljeni podaci o sustavu moraju se negdje pohraniti. U tu svrhu koristi se baza podataka, zbirka podataka koja je organizirana tako da joj se lako može pristupiti, upravljati i ažurirati. Korištena je jedna od najpopularnijih baza otvorenog kôda, MariaDB, koja je razvijena od zajednice, komercijalno podržana grana MySQL relacijskog sustava upravljanja bazom podataka (engl. RDBMS – *Relational Database Management System*). Razvoj MariaDB

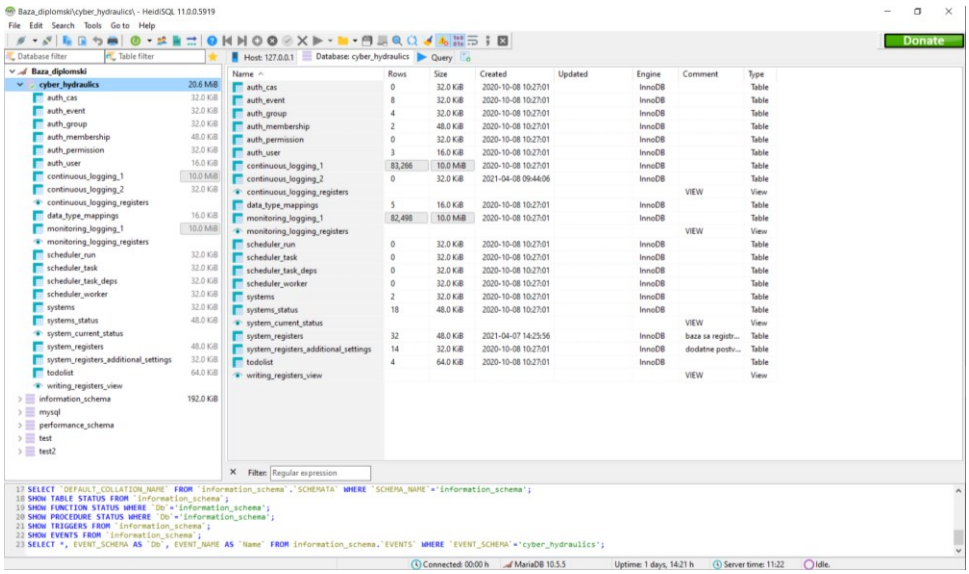
sustava vode neki od izvornih programera MySQL sustava za upravljanje bazama podataka te se na njemu i temelji. Glavna prednost korištenja MariaDB sustava je što dolazi s brojnim ugrađenim značajkama i mogućnostima te poboljšane sigurnosti i performansi u odnosu na MySQL [26]. Za upravljanje MariaDB sustavom korišten je besplatni program HeidiSQL kojem je svrha olakšati pregled i uređivanje podataka i strukture s računala. Također, podržava razne baze podataka poput MariaDB, MySQL, MS SQL, PostgreSQL i SQLite, a spada u najpopularniji program za MariaDB i MySQL u svijetu [27].



Slika 25. Početni zaslon HeidiSQL

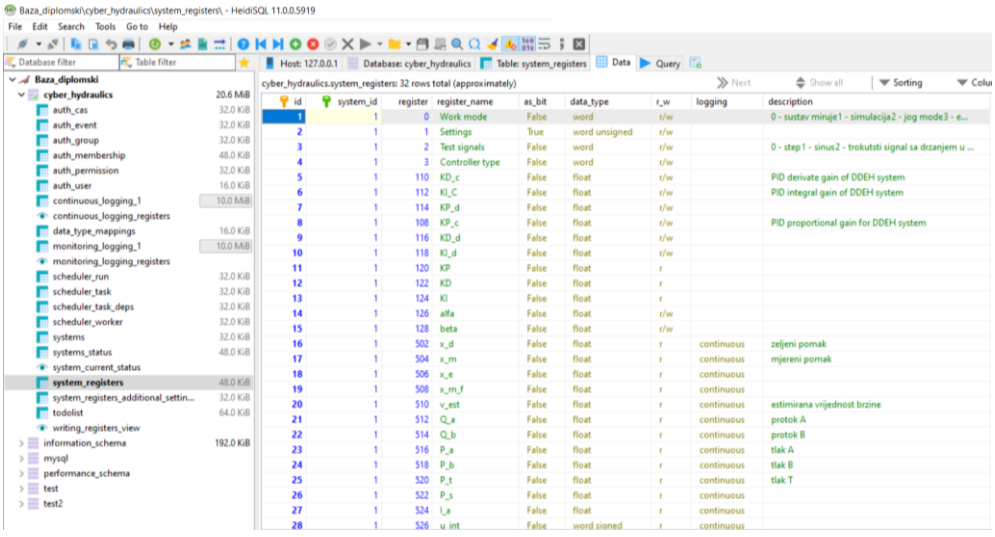
Na slici 25 prikazan je početni zaslon aplikacije HeidiSQL. Potrebno je definirati postavke sesije odnosno servera baze podataka odabirom na tip mreže MySQL (TCP/IP) te biblioteke *libmariadb.dll*. Također, bitno je postaviti Hostname / IP te zadati ime korisnika i šifru s kojom će se pristupati sesiji.

U samoj sesiji može biti više baza podataka, te u svakoj bazi niz tablica. Prikaz baze podataka *cyber_hydraulics* korištene za pohranu svih podataka dan je na slici 26. Baza se sastoji od raznih tablica poput podataka za identifikaciju, popisa sustava, popisa registara te kontinuirano prikupljenih podataka senzora.



Slika 26. Prikaz baze podataka

Primjer tablice, na slici 27, koja prikazuje popis svih registara od kojih svaki ima svoj jedinstveni ID i ID sustava kojem pripada (system_id: 1 odgovara sustavu "DDEH", dok system_id: 2 odgovara sustavu "Probni sustav").



Slika 27. Prikaz tablice registara

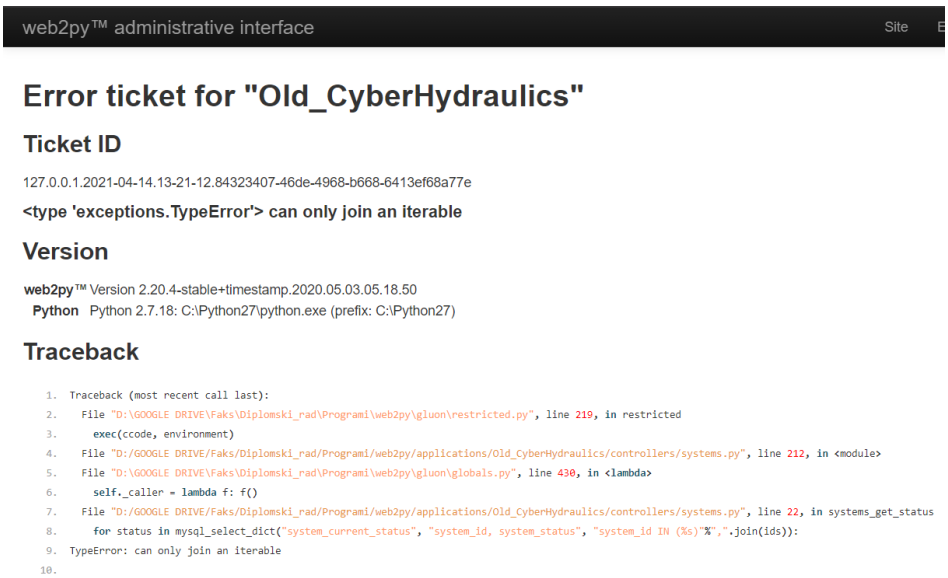
Navedena tablica registara rijetko se mijenja te je možemo smatrati relativno statičnom, dok s druge strane tablica *continuous_logging_1* ima konstantan dotok podataka sa ModbusTCP komunikacije s registrima PLC-a, gdje "1" na kraju imena definira sustav na koji se ta tablica odnosi. Prikaz tablice *continuous_logging_1* dan je na slici 28.

id	created	r_502	r_504	r_506	r_508	r_510	r_512	r_514	r_516	r_518	r_520	r_522	r_524	r_526	r_527
91,448	2021-04-08 13:05:13.303	0	-0.5	0	-0.420456	0	0	0	0.16	3.81	0.06	0.12	0.03	0	24
91,447	2021-04-08 13:05:13.157	0	-0.4	0	-0.387411	0	0	0	0.19	3.76	0.07	0.1	0.03	0	24
91,446	2021-04-08 13:05:13.023	0	-0.4	0	-0.381188	-0.533313	0	0	0.21	3.81	0.08	0.1	0.03	0	24
91,445	2021-04-08 13:05:12.882	0	-0.4	0	-0.360418	0	0	0	0.19	3.76	0.06	0.1	0.03	0	24
91,444	2021-04-08 13:05:12.748	0	-0.3	0	-0.375232	1.92549	0	0	0.17	3.83	0.05	0.11	0.03	0	24
91,443	2021-04-08 13:05:12.613	0	-0.4	0	-0.384264	-0.433224	0	0	0.17	3.74	0.05	0.11	0.03	0	24
91,442	2021-04-08 13:05:12.470	0	-0.4	0	-0.399524	0	0	0	0.17	3.83	0.05	0.11	0.03	0	24
91,441	2021-04-08 13:05:12.325	0	-0.3	0	-0.328724	0	0	0	0.16	3.84	0.04	0.1	0.03	0	24
91,440	2021-04-08 13:05:12.196	0	-0.4	0	-0.389185	0	0	0	0.19	3.88	0.07	0.12	0.03	0	24
91,439	2021-04-08 13:05:12.056	0	-0.3	0	-0.377412	0	0	0	0.19	3.81	0.07	0.11	0.03	0	24
91,438	2021-04-08 13:05:11.917	0	-0.4	0	-0.3461	0	0	0	0.18	3.79	0.07	0.09	0.03	0	24
91,437	2021-04-08 13:05:11.783	0	-0.4	0	-0.373281	0	0	0	0.16	3.76	0.08	0.12	0.03	0	24
91,436	2021-04-08 13:05:11.645	0	-0.3	0	-0.387074	0	0	0	0.21	3.81	0.07	0.09	0.03	0	24
91,435	2021-04-08 13:05:11.498	0	-0.4	0	-0.379623	0	0	0	0.21	3.79	0.07	0.11	0.03	0	24
91,434	2021-04-08 13:05:11.358	0	-0.3	0	-0.373645	0	0	0	0.18	3.83	0.07	0.11	0.03	0	24
91,433	2021-04-08 13:05:11.219	0	-0.3	0	-0.354029	0	0	0	0.13	3.8	0.04	0.1	0.03	0	24
91,432	2021-04-08 13:05:11.083	0	-0.4	0	-0.387213	-0.332826	0	0	0.18	3.8	0.07	0.12	0.03	0	24
91,431	2021-04-08 13:05:10.939	0	-0.4	0	-0.379386	0	0	0	0.21	3.81	0.07	0.11	0.03	0	24
91,430	2021-04-08 13:05:10.801	0	-0.4	0	-0.365627	-0.764865	0	0	0.18	3.77	0.06	0.11	0.03	0	24
91,429	2021-04-08 13:05:10.663	0	-0.3	0	-0.374026	0	0	0	0.16	3.8	0.08	0.09	0.03	0	24

Slika 28. Prikaz tablice stanja registara *continuous_logging_1*

4.2. Primijećeni nedostaci i prijedlog unaprjeđenja

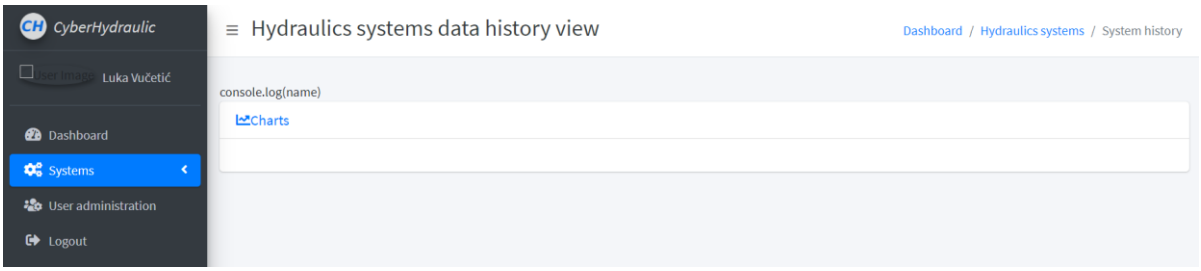
Glavni problem aplikacije je nedovršena *Pneumatics* kategorija stranice *Systems* te *Details* i *History view* pogledi. Odabirom na *Pneumatics*, aplikacija bi kontinuirano javljala internu grešku. Detaljnim pregledom kôda ustanovljeno je da će interna greška nestati ukoliko se definira pneumatski sustav, što nije zadovoljavajuće za rad aplikacije ukoliko ne postoji pneumatski sustav na kojem se radi, stoga je tu grešku potrebno otkloniti. Izgled interne greške prikazan je na slici 29.



Slika 29. Interna greška izborne kategorije *Pneumatics*

Kao što je ranije navedeno, pod kategorijom *Hydraulics* se za svaki sustav nudi mogućnost pregleda *Details* gdje su prikazani svi podaci o sustavu u realnom vremenu, video prikaz sustava

te komunikacija s PLC-om slanjem podataka u registre putem ModbusTCP veze. Također se, osim *Details*, nudi i povijesni pregled *History view* gdje bi trebao biti prikazan dijagram sa svim kontinuirano mjerenim podacima sa senzora sustava.



Slika 30. Prazna stranica *History view*

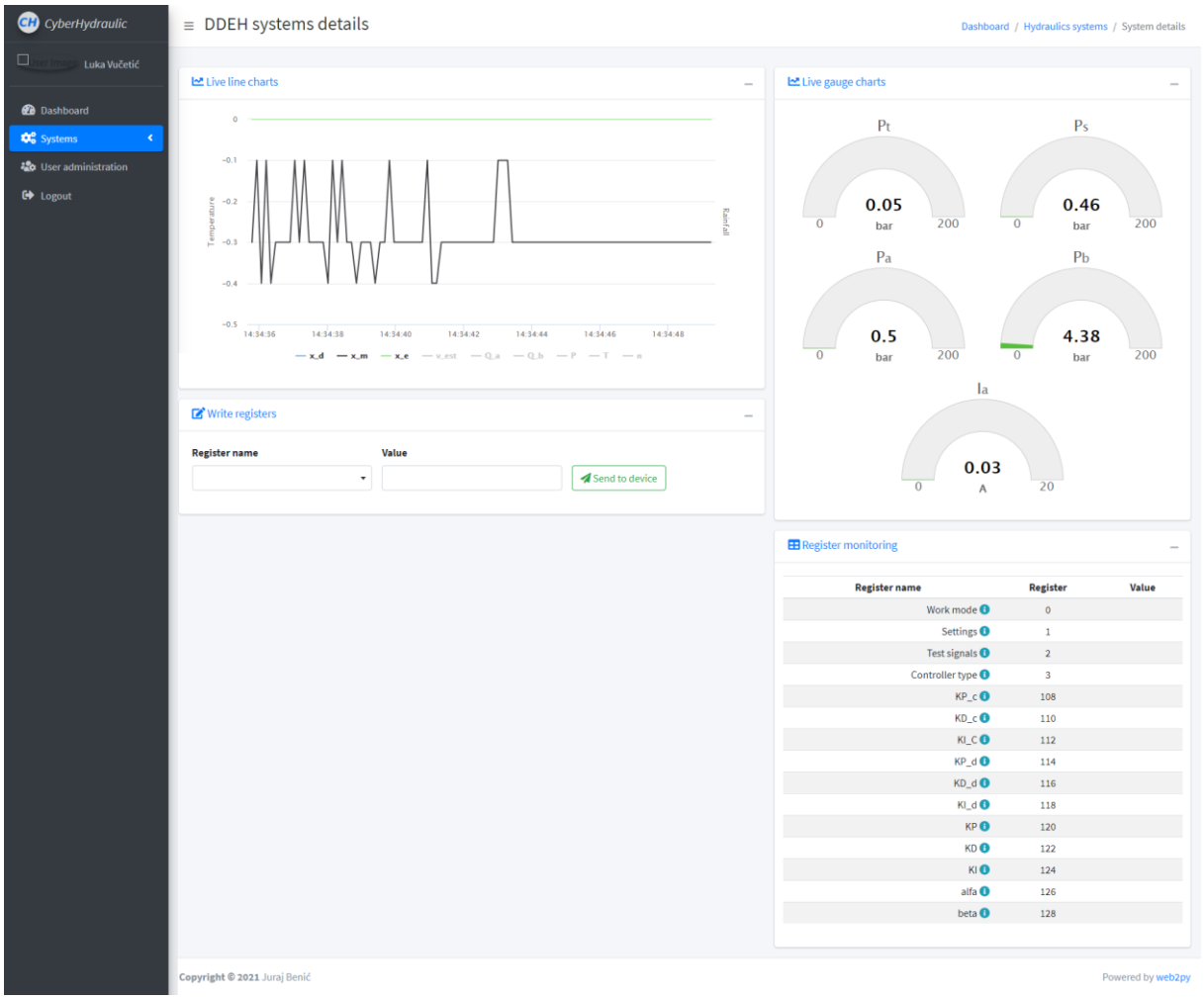
Stranica *History view* sadrži samo rezervirano mjesto za dijagram, prikazano na slici 30. Potrebno je izraditi prikaz dijagrama sa svim podacima registara koji se kontinuirano mjere s mogućnošću prikaza samo onih registara koje korisnik želi, omogućiti uvećanje određenog dijela dijagrama, odnosno prikaz podataka u određenom vremenskom periodu te mogućnost ispisa dijagrama kao pdf dokumenta ili kao slike.

Stranica *Details*, prikazana na slici 31, sastoji se od:

- Prikaza zadnje dobivene vrijednosti stanja registara mjernih vrijednosti iz baze podataka putem linijskog dijagrama *Live line charts*, koji nema definirane odgovarajuće mjerne jedinice na y-osi. Potrebno je popraviti mjerne jedinice da se dinamički prikazuju na y-osi ovisno o odabranim veličinama.
- Prikaza vrijednosti struje i tlaka sustava u obliku mjernih grafikona *Live gauge charts*
- Prikaza tablice stanja registara sustava, *Register monitoring,* poput vrijednosti pojačanja PID regulatora, tipa regulatora, vrste pobudnog signala i vrste rada sustava. Potrebno je uspostaviti prikupljanje podataka s PLC-a za navedene registre i te vrijednosti prikazati i osvježavati u realnom vremenu u stupcu *Value*.
- Polja *Write registers* za upis vrijednosti u registre PLC-a putem *ModbusTCP* komunikacije, korisnik odabire željeni registar u padajućem izborniku pod *Register*

name, a pod *Value* unosi vrijednost koju želi upisati u odabrani registar. Potrebno je osposobiti slanje podataka putem ModbusTCP protokola PLC-u.

- Video prikaza postava u stvarnom vremenu *Live camera*. Potrebno je konstruirati nosač za web kameru i Raspberry Pi te uspostaviti video prikaz s web kamere putem Raspberry Pi računala.



Slika 31. Prikaz stranice *Details*

4.3. Unaprjeđenje izborne kategorije Pneumatics

Pregledom postojećeg kôda ustanovljeno je da se greška pojavljuje zbog *ajax* zahtjeva o podacima sustava, prikazanog žutom bojom na liniji 12 u programskom kôdu 1, koji se nalazi pod stranicom *systems.html* koja je zadužena za prikaz svih hidrauličkih i pneumatskih sustava.

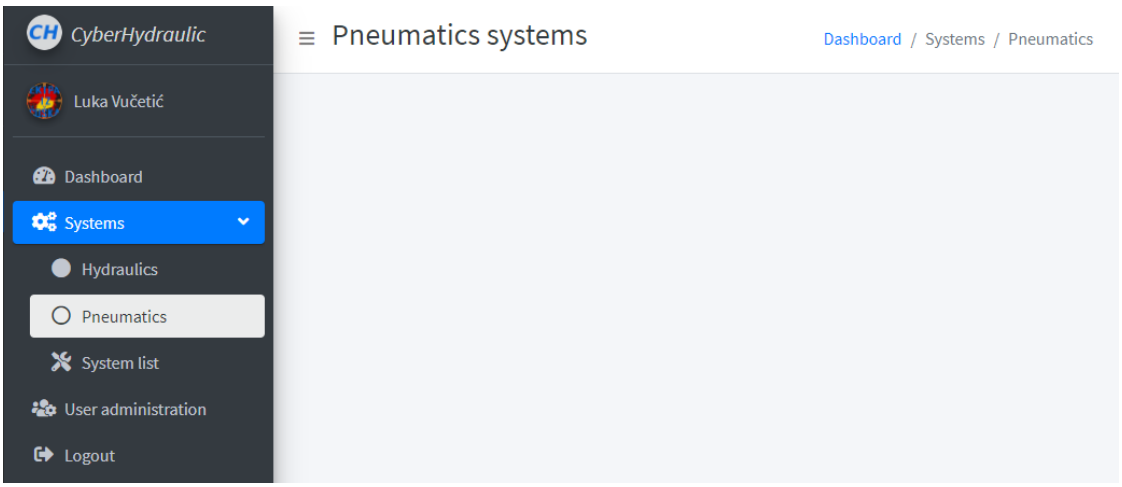
Programski kôd 1. Greška *ajax* zahtjeva

```
1. // id-evi svih divova sa postavom
2. var ids = $('.system').map(function(){
3.     return $(this).attr('id').split("-")[1]; }).get();
4. setInterval(function() {
5.   $.ajax({
6.         type: "POST",
7.         async: true,
8.         dataType: "json",
9.         url: "{{=URL('systems','systems_get_status')}}",
10.        data: {"ids":ids},
11.        success: function(d){
12.           $.each(d, function(i,v){
13.               $("#system-"+i+"-system_status").text(v["status"]);
14.               $("#system-"+i).removeClass( "card-gray card-success card-
    danger card-primary card-gray-dark" );
15.               $("#system-"+i).addClass("card-"+v["color"]);
16.           });}});}, 100);
```

Kako nema definiranog pneumatskog sustava, ne postoji definirani *ids* sustava te se javlja problem pri pozivanju tablice iz baze podataka za sustav koji ne postoji. Greška je uklonjena postavljanjem *ajax* zahtjeva pod uvjetnom izjavom *if* prikazanom na programskom kôdu 2. Postavljanjem tog uvjeta, *ajax* zahtjev će se izvršiti samo ako postoji *ids* sustava, odnosno ako duljina *ids* varijable nije nula, što je točno samo ako postoji definirani sustav, dok u slučaju da je duljina *ids* varijable jednaka nuli, *ajax* zahtjev se preskače te se prikazuje prazna kategorija *Pneumatics* prikazana na slici 32.

Programski kôd 2. Dodani *if* uvjet *ajax* zahtjeva

```
1. if (ids.length != 0){
2.     setInterval(function() {
3.         $.ajax({...}) }}
```



Slika 32. Prikaz stranice izborne kategorije *Pneumatics*

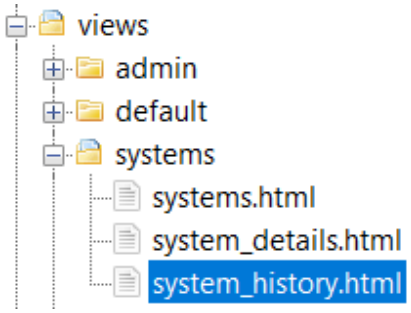
4.4. Povijesni pregled

Kao što je ranije navedeno, *History view* stranica služi prikazu svih pohranjenih vrijednosti sustava. Podatke je potrebno iščitati iz baze podataka *cyber_hydraulics* te ih prikazati na linijskom grafu. Podatke dohvaćamo uz pomoć kontrolera Web2Py aplikacije koji koristi mySQL funkciju zapisanu u modelu aplikacije *mySQLfunkcije.py* kao Python skripta. Pri pozivanju funkcije potrebno je odabrati tablicu iz baze podataka, definirati koja polja, odnosno stupce trebamo te zadati uvjet odabira. Prikaz poziva funkcije za dohvaćanje baze prikazan je pod programskim kôdom 3. Ova funkcija nalazi se u kontroleru aplikacije *systems.py* te se poziva svaki puta kada korisnik pristupi *History view* stranici. Potrebno je dohvatiti podatke svakog registra iz tablice *continuous_logging_1* te je također potrebno dohvatiti podatke o registrima iz tablice *system_registers*.

Programski kôd 3. Dohvaćanje potrebnih baza za *History view*

```
1.  def system_history():
2.
3.      id = request.vars.id
4.
5.      #========================================================================
6.      # dohvacanje podataka o kontinuiranom logiranju podataka
7.      #========================================================================
8.
9.      data = mysql_select_dict("continuous_logging_%s"%id, "*","id > 0")
10.
11.     # dohvacanje variabli koje se prikazuju kao krivulja
12.     line_graph = mysql_select_dict("""system_registers_additional_settings
13.                                       LEFT JOIN system_registers ON
    system_registers.id=system_registers_additional_settings.system_register_id""
    ",
14.                                       "system_register_id, register,
    graph_type, data_unit, min_value , max_value, register_name",
15.                                       "system_id = %s"%id )
16.
17.     return dict(
18.                 line_graph = json.dumps(line_graph),
19.                 data = json.dumps(data, default=to_unix_time),
20.                )
```

U pogledu (engl. *View*) aplikacije pod datotekom *systems* nalaze se html dokumenti, prikazani na slici 33, vezane za stranicu *Systems*, odnosno za *Details* i *History view* neovisno radi li se o hidrauličkom ili pneumatskom sustavu. Dokument *systems.html* odnosi se na stranicu koja se prikazuje nakon odabira pneumatskog ili hidrauličkog sustava, dok se *system_details.html* i *system_history.html* odnose na *Details* i *History view* stranicu.



Slika 33. Prikaz html dokumenata za *systems* pogled

Unutar *systems_history.html* dokumenta potrebno je definirati inicijalne podatke za kreiranje dijagrama te podatke o registrima. Ti podaci se lako dohvaćaju, u XML obliku, pomoću Python naredbe unutar dvije vitičaste zagrade prikazano na programskom kôdu 4.

Programski kôd 4. Podaci za *History view* dijagram

```
/*=========================================================
      inicijalni podatci za kreiranje grafa
  =========================================================*/
  var data = {{=XML(data)}};
  /*=========================================================
      dodavanje linijskih dijagrama
  =========================================================*/
  var line_graph = {{=XML(line_graph)}},
      line_data = [];
```

Za crtanje dijagrama korištena je *JavaScript* biblioteka grafikona *Highcharts JS* koja se temelji na promjenjivoj vektorskoj grafici (engl. SVG - *Scalable Vector Graphics*). Prednost *Highcharts* biblioteke je u slobodnom korištenju za nekomercijalne uporabe te u jednostavnosti i skalabilnosti stvaranja grafikona [29]. Za prikaz povijesnog pregleda odabran je linijski dijagram, jer pregledno prikazuje promjenu vrijednosti podataka u vremenu.

Programski kôd 5. Prilagodba podataka za *History view* dijagram

```
$.each(line_graph, function(i,v){
          var pomocna = { name:v["register_name"], data:[], yAxis:0};
          if (i>2){ pomocna.visible = false, pomocna.yAxis = 1 }
          if (i>6){pomocna.yAxis = 2}
          if (i>7){pomocna.yAxis = 3}
          if (i>9){pomocna.yAxis = 4}
          if (i>10){pomocna.yAxis = 5}
          if (i>11){pomocna.yAxis = 6}
          if (i>12){pomocna.yAxis = 7}
  $.each(data,function(ii,vv){
      pomocna.data.push( [vv["created"], vv[ "r_"+v["register"]] ] )
  });
  line_data.push(pomocna)
});
```

Kako bi *Highcharts* mogao prikazati ispravno dijagram, potrebno je podatke prvo urediti kako je prikazano na programskom kôdu 5. Da bi se olakšala manipulacija podacima, stvoren je rječnik *pomocna* koji se sastoji od imena registara koji se uzimaju iz varijable *line_graph*, podataka *data* koji su niže u kôdu ispunjeni vrijednostima iz varijable *data* te vrijednosti y-osi (*yAxis)* koju je bitno definirati za ispravan prikaz mjernih jedinica. Neki od podataka dijele iste mjerne jedinice te stoga spadaju pod istu y-os (*yAxis*). Naposljetku, varijablu *pomocna* pripisujemo u *line_data* varijablu koja se koristi kao ulaz podataka za crtanje dijagrama. Pozivanjem objekta *Highcharts* potrebno je definirati parametre poput naslova, imena osi, ulaznih podataka te raznih opcija, prikazanih u programskom kôdu 6. Prvi bitan parametar za dijagram *History view* je *chart* u kojem definiramo tip dijagrama te postavke za njegovo uvećanje.

Programski kôd 6. Definiranje *Highcharts* objekta za *History view*

```
1. Highcharts.chart('line', {
2.       chart: {
3.           type: 'line',
4.           zoomType: 'x',
5.           panning: true,
6.           panKey: 'shift'
7.       },
8.
9.       title: { text: "" },
10.
11.      time: { useUTC: true },
12.
13.      yAxis: [{ // nulta os
14.          labels: {
15.              format: '{value} mm',
16.              style: { color: Highcharts.getOptions().colors[0] }
17.          },
18.          title: { text: '' }
19.
20.      }, { // prva os
21.          labels: {
22.              format: '{value} bar',
23.              style: { color: Highcharts.getOptions().colors[3] }
24.          },
25.          title: { text: '' },
26.          opposite: true
27.      }, ...
28.
29.      ],
30.
31.    xAxis: { type: 'datetime' }
32.
33.    series: line_data,
34.
35.    plotOptions:{ line:{ marker:{ enabled:false }}},
36.
37.    exporting: { enabled: true }
38.  });
```

Odabrano je povećanje samo po x-osi, jer se na njoj nalazi vrijeme. Razlog odabira leži u svrsi povećanja, jer korisnika će zanimati stanje registara u određenom vremenskom periodu pa mu je potrebno omogućiti, pomoću uvećanja, odabir određenog vremenskog intervala. Također je bitno omogućiti i pomicanje tako uvećanog dijagrama. Nakon testiranja odabrana je tipka *shift* kao akcijska tipka koja mora biti pritisnuta da bi se omogućilo pomicanje dijagrama. Idući bitan parametar je *yAxis* koji služi za definiranje imena i oznaka na toj osi. Odabrano je sedam osi za prikaz četrnaest varijabli. Razlog odabira manje osi je što neke varijable dijele mjerne jedinice, pa ih možemo prikazati na istoj mjernoj osi. Nadalje, prikazano je definiranje prve dvije osi, nulta os u milimetrima i prva os u barima. Osim definiranja mjerne jedinice, definirana je i različita boja kako bi se olakšalo snalaženje po različitim mjernim osima. Naslov mjerne osi ostavljen je prazan jer bi uveo nepotreban šum pri iščitavanju vrijednosti. Definiranje x-osi mnogo je jednostavnije, jer je potrebno prikazati samo vrijeme. Sadržaj objekta *line_data* može se lako provjeriti pozivom kôda *console.log(line_data)*, čime dobivamo ispis u konzoli web preglednika prikazanog na slici 34, a sastoji se od imena varijable, njenih vrijednosti, kojoj osi pripada te vidljivosti pri otvaranju stranice.

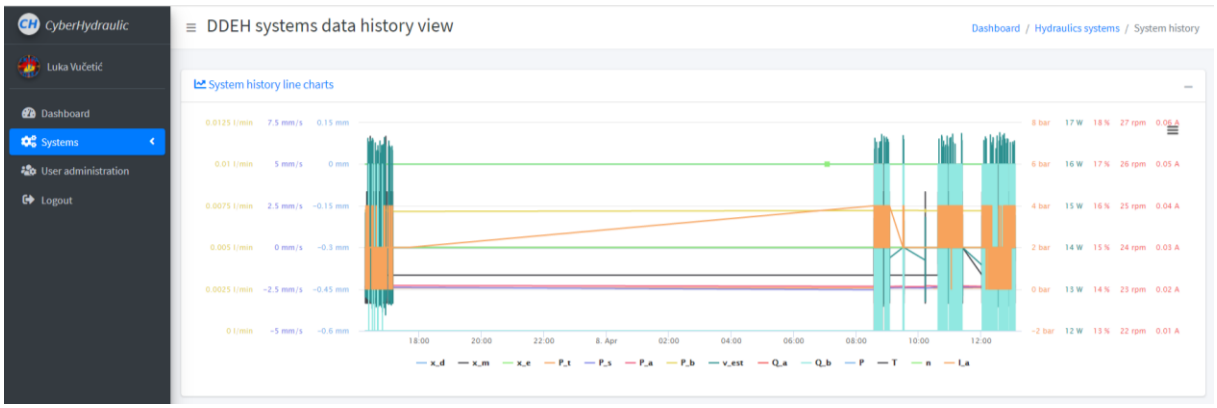
Slika 34. Sadržaj objekta *line_data*

Posljednji parametar kojega je potrebno definirati je mogućnost ispisa dijagrama u pdf dokument ili sliku. Tako definirani dijagram potrebno je prikazati na stranici pomoću <*div*> elementa prikazanog na programskom kôdu 7. Naziv dijagrama je *System history line charts*, a sam dijagram poziva se pomoću *id="line"*, definiranog u prvom retku u programskom kôdu 6, gdje *id* označava jedinstveno svojstvo identifikacije HTML elementa.

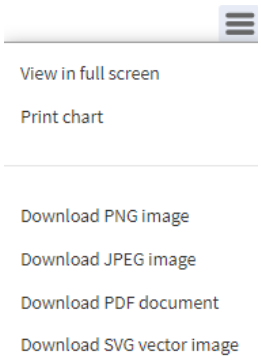
Programski kôd 7. Definiranje prikaza dijagrama

```
1. <div class="card card-outline" id="card-line-graph">
2.   <div class="card-header">
3.           <h3 class="card-title text-blue"><i class="fas fa-chart-line"></i>
   System history line charts</h3>
4.           <div class="card-tools">
5.                   <!-- Collapse Button -->
6.                   <button type="button" class="btn btn-tool" data-card-
   widget="collapse"><i class="fas fa-minus"></i></button>
7.           </div>
8.   </div>
9.   <div class="card-body">
10.          <div class="container-fluid">
11.                  <div id="line"></div>
12.          </div>
13.  </div>
14.</div>
```

Na slici 35 dan je pregled cijele stranice *History view* s uključenim svim varijablama, korisnik po želji treba kliknuti na varijablu kako se ona ne bi prikazivala. Odgovarajuće mjerne jedinice nalaze se s lijeve i desne strane dijagrama kako bi se olakšao pregled. Prikaz izbornika za ispis dijagrama u pdf ili u sliku nalazi se na slici 36.



Slika 35. Unaprijeđena stranica *History view*



Slika 36. Prikaz ispisa dijagrama

4.5. Trenutni pregled

Na stranici *Details* prikazani su svi podaci u realnom vremenu, putem linijskog dijagrama, mjernih grafikona te tablice stanja registara. Također je omogućeno upisivanje vrijednosti u registre te je cijeli sustav prikazan putem video kamere. Kao i za *History view* stranicu, podatke dohvaćamo uz pomoć kontrolera Web2Py aplikacije koja koristi mySQL funkciju zapisanu u modelu aplikacije *mySQLfunkcije.py*. Također, potrebno je zadati limit redaka koje želimo učitati prikazano na programskom kôdu 8, a odabrano je da učitava zadnjih 100 unosa kako bi se prikazale realne vrijednosti u zadnjih desetak sekundi.

Programski kôd 8. Funkcije za *system_details* stranicu

```
1.  def system_details():
2.      id = request.vars.id
3.      #============================================================================
4.      # dohvacanje podataka o kontinuiranom logiranju podataka
5.      #============================================================================
6.      rows = db.executesql("show table status like 'continuous_logging_%s'"%id)[0][10]
7.      limit = "limit %s, 100"%(rows-100) if rows > 100 else ""
8.      data = mysql_select_dict("continuous_logging_%s"%id, "*", "id > 0 order by created
    asc %s"%limit)
9.      # dohvacanje varijabli koje se prikazuju kao solid gauge
10.     solid_gauge = mysql_select_dict("""system_registers_additional_settings LEFT JOIN
    system_registers ON
    system_registers.id=system_registers_additional_settings.system_register_id""","system_re
    gister_id, register, graph_type, data_unit, min_value , max_value, register_name",
    "system_id = %s and graph_type = 'solid gauge'"%id )
11.     divs_solid_gauge = []
12.     for i in solid_gauge:
13.         divs_solid_gauge.append( DIV(_class="solid-gauge-container", _id="solid-gauge-
    %s"%i["system_register_id"]) )
14.     # dohvacanje variabli koje se prikazuju kao krivulja
15.     line_graph = mysql_select_dict("""system_registers_additional_settings LEFT JOIN
    system_registers ON
    system_registers.id=system_registers_additional_settings.system_register_id""","system_re
    gister_id, register, graph_type, data_unit, min_value , max_value, register_name",
    "system_id = %s and graph_type = 'line'"%id )
16.     line_graph_yAxis = {}
17.     #============================================================================
18.     # dohvacanje podataka za monitoring registara
19.     #============================================================================
20.     data_monitoring = mysql_select_dict("monitoring_logging_%s"%id, "*", "id > 0 order by
    created desc limit 1")
21.     systems_info = mysql_select_dict("systems", "*", "id=%s"%id)
22.     registers = mysql_select_dict("system_registers", "*", "system_id=%s and logging=\"\"
    order by register asc"%id)
23.     return dict(solid_gauge=json.dumps(solid_gauge),
24.                 divs_solid_gauge = divs_solid_gauge,
25.                 line_graph = json.dumps(line_graph),
26.                 systems_info = json.dumps(systems_info),
27.                 registers = registers,
28.                 registers_json = json.dumps(registers),
29.                 data = json.dumps(data, default=to_unix_time),
30.                 data_monitoring = json.dumps(data_monitoring, default=to_unix_time)
31.                 )
```

Kako bi prikazali mjerne grafikone potrebno je koristeći funkciju *mysql_select_dict()* odabrati varijable definirane kao *solid gauge* u tablici *system_registers_additional_settings* prikazanoj na slici 37.

cyber_hydraulics.system_registers_additional_settings: 14 rows total (approximately)

id	system_register_id	graph_type	data_unit	min_value	max_value
1	16	line	mm	-10	310
2	17	line	mm	-10	300
3	18	line	mm	(NULL)	(NULL)
4	25	solid gauge	bar	0	200
5	26	solid gauge	bar	0	200
6	23	solid gauge	bar	0	200
7	24	solid gauge	bar	0	200
8	20	line	mm/s	(NULL)	(NULL)
9	21	line	l/min	(NULL)	(NULL)
10	22	line	l/min	(NULL)	(NULL)
11	31	line	W	(NULL)	(NULL)
12	30	line	%	(NULL)	(NULL)
13	29	line	rpm	(NULL)	(NULL)
14	27	solid gauge	A	0	20

Slika 37. Prikaz dodatnih postavki registara

Nadalje, kao i za *History view*, potrebno je dohvatiti podatke za prikaz linijskog dijagrama. Bitna razlika u odnosu na poziv funkcije za *History view* je što sad želimo odabrati samo one registre koji su u tablici prikazanoj na slici 37 označeni s *line*. Naposljetku, potrebno je dohvatiti podatke za osvježavanje tablice stanja registara koji su pohranjeni u tablici *monitoring_logging_1*, podatke za upisivanje vrijednosti u registre koji se nalaze u tablici *system_registers* te informacije o sustavu bitne za uspostavu komunikacije s Raspberry Pi. Sve prethodno navedene podatke vraćamo kao rječnike na kraju *system_details()* funkcije.

Unutar *systems_details.html* skripte potrebno je definirati inicijalne podatke za kreiranje dijagrama te podatke o sustavu. Ti podaci se lako dohvaćaju, u XML obliku, pomoću Python naredbe unutar dvije vitičaste zagrade. Identično kao i kod *History view* dijagrama, i ovdje je potrebno učitane podatke urediti, prikazano na programskom kôdu 9. Ponovno je stvoren rječnik *pomocna* koji se sastoji od imena registara, podataka *data* te podataka o y-osi (*yAxis*) kojih ovdje imamo pet u odnosu na sedam kod *History view* iz razloga što su varijable tlaka i struje prikazane pomoću mjernih grafikona.

Programski kôd 9. Prilagodba podataka za *Details* dijagram

```
var data = {{=XML(data)}};
var data_last_id = data[data.length-1]["id"];
var systems_info = {{=XML(systems_info)}};
var line_graph = {{=XML(line_graph)}},
line_data = [],
line_data_reg_dict = [];
$.each(line_graph, function(i,v){
        var pomocna = { name:v["register_name"], data:[], yAxis:0};
        line_data_reg_dict.push([ "r_"+v["register"] ] );
        if (i>2){ pomocna.visible = false, pomocna.yAxis = 1 }
        if (i>3){pomocna.yAxis = 2}
        if (i>5){pomocna.yAxis = 3}
        if (i>6){pomocna.yAxis = 4}
        if (i>7){pomocna.yAxis = 5}
        $.each(data,function(ii,vv){
             pomocna.data.push( [vv["created"], vv[ "r_"+v["register"]] ] )
        });
        line_data.push(pomocna); });
```

U već definiranom kôdu linijskog dijagrama stranice *Details*, uređen je parametar *yAxis* kako bi mjerne jedinice odabranih varijabli bile prikazane dinamično, odnosno prikazuju se mjerne jedinice samo odabranih varijabli. Na programskom kôdu 10 prikazane su treća i četvrta os. Također su definirane različite boje kako bi se olakšalo snalaženje po mjernim osima te je naslov mjerne osi ostavljen prazan, jer bi uveo nepotreban šum pri iščitavanju vrijednosti.

Programski kôd 10. Definiranje mjernih jedinica na y-osi

```
    }, { // treca os
        labels: {
            format: '{value} W',
            style: { color: Highcharts.getOptions().colors[6] }
        },
        title: { text: '' },
        opposite: true
    }, { // cetvrta os
        labels: {
            format: '{value} %',
            style: { color: Highcharts.getOptions().colors[7] }
        },
        title: { text: '' },
        opposite: true
    }, ...
```

Potrebno je nadograditi postojeću skriptu *ModbusTCP.py* s funkcijama sličnim kao i za osvježavanje podataka linijskih dijagrama da bi se ostvarilo osvježavanje podataka u tablici stanja registara sustava *Register monitoring*. Prvo se definira funkcija za spremanje rječnika u bazu podataka *mysql_save_dict1()* koja ulazne podatke *data* sprema u odabranu tablicu prikazanu na programskom kôdu 11.

Programski kôd 11. Prikaz funkcije *mysql_save_dict1()*

```
1.      def mysql_save_dict1(self, tablica, data):
2.          """Funkcija za spremanje rjecnika u bazu podataka"""
3.          placeholders = ', '.join(['%s'] * len(data))
4.          columns = ', '.join(data.keys())
5.          sql = "INSERT INTO %s ( %s ) VALUES ( %s )" % (tablica, columns,
    placeholders)
6.          con = MySQLdb.connect(host=self._host, user=self._user,
    passwd=self._passwd, db=self.db, charset='utf8')
7.          cur = con.cursor()
8.          id = cur.execute(sql, data.values())
9.          cur.close()
10.         con.commit()
11.         con.close()
12.         return id
```

Nadalje, potrebno je dohvatiti procesne varijable iz postojeće tablice *monitoring_logging_registers* prikazane na slici 38 te ih pohraniti u rječnik *proces_vars1* koji se nalazi unutar funkcije *get_proces_variables1()* na programskom kôdu 12, koji će nam olakšati obradu podataka pristiglih putem ModbusTCP protokola. Bitno je napomenuti ključeve "*bytes1*" i "*bytes2*" koji se sastoje od zbroja količine registara svakog registra (1 ili 2) prikazane u zadnjem stupcu tablice *monitoring_logging_registers*.

Programski kôd 12. Prikaz funkcije *get_proces_variables1()*

```
1.      def get_proces_variables1(self,system_id):
2.          """za monitoring varijabli registara""",
3.
4.          proces_vars1 = {"id":[], "number_of_registre":[], "pack":[],
    "unpack":[], "bytes1":[], "bytes2":[] }
5.
6.          #za system registre
7.          for i in self.mysql_select_dict("monitoring_logging_registers", "*",
    "system_id=%s"%system_id):
8.
9.              proces_vars1["id"].append(i["r_name"])
10.             proces_vars1["number_of_registre"].append(i["number_of_registre"])
11.             proces_vars1["pack"].append(i["pack"])
12.             proces_vars1["unpack"].append(i["unpack"])
13.
14.             if i["number_of_registre"] == 1:
15.                 proces_vars1["bytes1"].append(i["number_of_registre"])
16.             elif i["number_of_registre"] == 2:
17.                 proces_vars1["bytes2"].append(i["number_of_registre"])
18.
19.         proces_vars1["bytes1"] = sum(proces_vars1["bytes1"])
20.         proces_vars1["bytes2"] = sum(proces_vars1["bytes2"])
21.
22.         return proces_vars1
```

cyber_hydraulics.monitoring_logging_registers

system_id	register	r_name	pack	unpack	number_of_registre
1	0	r_0	H	h	1
1	1	r_1	H	H	1
1	2	r_2	H	h	1
1	3	r_3	H	h	1
1	108	r_108	HH	f	2
1	110	r_110	HH	f	2
1	112	r_112	HH	f	2
1	114	r_114	HH	f	2
1	116	r_116	HH	f	2
1	118	r_118	HH	f	2
1	120	r_120	HH	f	2
1	122	r_122	HH	f	2
1	124	r_124	HH	f	2
1	126	r_126	HH	f	2
1	128	r_128	HH	f	2

Slika 38. Prikaz tablice *monitoring_logging_registers*

Tako definirani rječnik *proces_vars1* koristi se u novoj funkciji *save_continuous_modbus_data1()* prikazane na programskom kôdu 13, kojoj je zadatak raspakirati pristigle podatke te ih pohraniti u bazu podataka pozivanjem ranije definirane funkcije *mysql_save_dict1*.

Programski kôd 13. Prikaz funkcije *save_continuous_modbus_data1()*

```
1.      def save_continuous_modbus_data1(self, proces_vars1, data, system_id):
2.          """Funkcija koja u bazu sprema podatke pristigle preko TCP
    protokola"""
3.          first1 = 0
4.
5.          save_data1 = {}
6.
7.          #za system registre
8.          for i in range( len(proces_vars1["id"]) ):
9.
10.             save_data1[ proces_vars1["id"][i] ] = struct.unpack(
    proces_vars1["unpack"][i], struct.pack( proces_vars1["pack"][i],
    *data[first1:first1+proces_vars1["number_of_registre"][i]]) )[0]
11.
12.             first1 += proces_vars1["number_of_registre"][i]
13.
14.         self.mysql_save_dict1("monitoring_logging_%s"%system_id, save_data1)
```

Naposljetku, potrebno je proširiti postojeću funkciju *polling_thread(self, system_id)*, kojoj je zadatak unutar beskonačne petlje čitati stanje registara PLC-a pomoću ModbusTCP konekcije. Dodaju se pomoćne varijable *reg_list2* i *reg_list3* koje se pridružuju konačnoj varijabli *reg_list1* koja sadrži podatke svih registara prikazane na programskom kôdu 14. Razlog za postojanje dviju pomoćnih varijabli je rascjepkanost registra, odnosno, registri od 0 do 3 se sastoje od jednog registra, dok se registri od 108 do 128 sastoje od dva registara. Za čitanje registara koristi se ModbusTCP funkcija *modbus.read_holding_registers(reg_addr, reg_nb)*, gdje *reg_addr* označava adresu registra od kojeg počinje čitanje (cijeli broj od 0 do 65535), a *reg_nb* označava količinski broj registara koje treba pročitati (cijeli broj od 1 do 125). Pomoću *reg_list2* čitaju se registri počevši od *reg_addr=0*, za *reg_nb="bytes1"* koji je u ovom slučaju 4, što odgovara broju registara koji se sastoje od jednog registra iz tablice *monitoring_logging_registers*. Isto tako, *reg_list3* čita registre počevši od *reg_addr=108*, za *reg_nb="bytes2"* koji je u ovom slučaju 22, što odgovara zbroju registara koji se sastoje od dva registra iz tablice *monitoring_logging_registers*. U ovom obliku nam podaci nisu korisni jer su zapakirani, stoga ih je potrebno raspakirati te pridružiti odgovarajućim registrima pomoću funkcije *save_continuous_modbus_data1()* definirane pod programskim kôdom 13. Time je definirano prikupljanje novih podataka s PLC-a pomoću ModbusTCP protokola te njihovo pohranjivanje u bazu podataka.

Programski kôd 14. Čitanje registara PLC-a

```
1.  if modbus.read_holding_registers(0, 1):
2.      # citaj registre za continuous_logging
3.      reg_list = modbus.read_holding_registers(502, proces_variables["bytes"])
4.
5.      #citaj registre za register_monitoring
6.      reg_list2 = modbus.read_holding_registers(0, proces_variables1["bytes1"])
7.      reg_list3 = modbus.read_holding_registers(108,
        proces_variables1["bytes2"])
8.      reg_list1 = reg_list2 + reg_list3
9.
10.     # ako su uspjesno procitani podatci, spremi ih
11.     if reg_list:
12.         self.save_continuous_modbus_data(proces_variables, reg_list,
        system_id)
13.     if reg_list1:
14.         self.save_continuous_modbus_data1(proces_variables1, reg_list1,
        system_id)
```

Idući korak je omogućiti prikaz vrijednosti za tablicu stanja registara sustava. Prvo je potrebno stvoriti varijable *registers* i *data_monitoring* unutar *system_details.html* iz istoimenih podataka definiranih u funkciji na programskom kôdu 8. Koristeći Python *for* petlju prolazi se kroz sve registre sa slike 39, prikazana na programskom kôdu 15 te se pomoću *document.getElementById("{{='r_%s'%i['register']}}")* pretražuje kroz html dokument. Svakim prolazom *register* poprima drugu vrijednost iz tablice, i traži elemente koji posjeduju ID *r_0* do *r_128* te im dodjeljuje zadnju vrijednost iz varijable *data_monitoring*.

Programski kôd 15. Prikaz registara za *for* petlju

```
1.    {{for i in registers:}}
2.
3.    document.getElementById("{{='r_%s'%i['register']}}").innerHTML =
    data_monitoring[0].{{='r_%s'%i['register']}};
4.
5.    {{pass}}
```

cyber_hydraulics.system_registers: 32 rows total (approximately)

id	system_id	register	register_name	as_bit	data_type	r_w	logging
1	1	0	Work mode	False	word	r/w	
2	1	1	Settings	True	word unsigned	r/w	
3	1	2	Test signals	False	word	r/w	
4	1	3	Controller type	False	word	r/w	
5	1	110	KD_c	False	float	r/w	
6	1	112	KI_C	False	float	r/w	
7	1	114	KP_d	False	float	r/w	
8	1	108	KP_c	False	float	r/w	
9	1	116	KD_d	False	float	r/w	
10	1	118	KI_d	False	float	r/w	
11	1	120	KP	False	float	r	
12	1	122	KD	False	float	r	
13	1	124	KI	False	float	r	
14	1	126	alfa	False	float	r/w	
15	1	128	beta	False	float	r/w	

Slika 39. Prikaz registara za *for* petlju

Programski kôd 16 prikazuje izgled tijela tablice stanja registara sustava te se u 10. retku može vidjeti Python kôd koji pridjeljuje svakom retku traženi ID, odnosno od *r_0* do *r_128*. Konačan izgled tablice stanja registara sustava s vrijednostima *value* koje se osvježuju trenutnim vrijednostima s PLC-a prikazan je na slici 40.

Programski kôd 16. Tijelo tablice stanja registara

```
1. <tbody>
2.   {{for i in registers:}}
3.           <tr>
4.                   <td class="text-center">
5.                           <span id>{{=i["register_name"]}}</span>
6.                           i class="fas fa-info-circle text-info" data-
   toggle="tooltip"  data-html="true" data-placement="top"
   title="{{=i['description'].replace('\r\n','<br>')}}"></i>
7.                   </td>
8.                   <td class="text-center">{{=i["register"]}}</td>
9.                   <td class="text-center">
10.                          <span id="{{='r_%s'%i['register']}}"></span>
11.                  </td>
12.          </tr>
13.  {{pass}}
14.</tbody>
```

Register monitoring

Register name	Register	Value
Work mode	0	0
Settings	1	0
Test signals	2	0
Controller type	3	1
KP_c	108	260
KD_c	110	12
KI_C	112	250
KP_d	114	260
KD_d	116	10
KI_d	118	200
KP	120	0
KD	122	0
KI	124	0
alfa	126	1
beta	128	1

Slika 40. Konačan izgled tablice stanja registara sustava

Naposljetku, potrebno je osposobiti slanje podataka putem ModbusTCP protokola sa web stranice na PLC. Na programskom kôdu 17 prikazana je već postojeća *ajax* forma za slanje podataka, koja se nalazi unutar html dokumenta *system_details.html*. Iz URL-a forme se može iščitati da se poziva funkcija *system_details_write_data()* koja se nalazi u Python skripti *systems.py*. Funkcija nije bila u potpunosti definirana te prijenos podataka nije bilo moguće uspostaviti.

Programski kôd 17. Prikaz *ajax* forme za slanje podataka na PLC

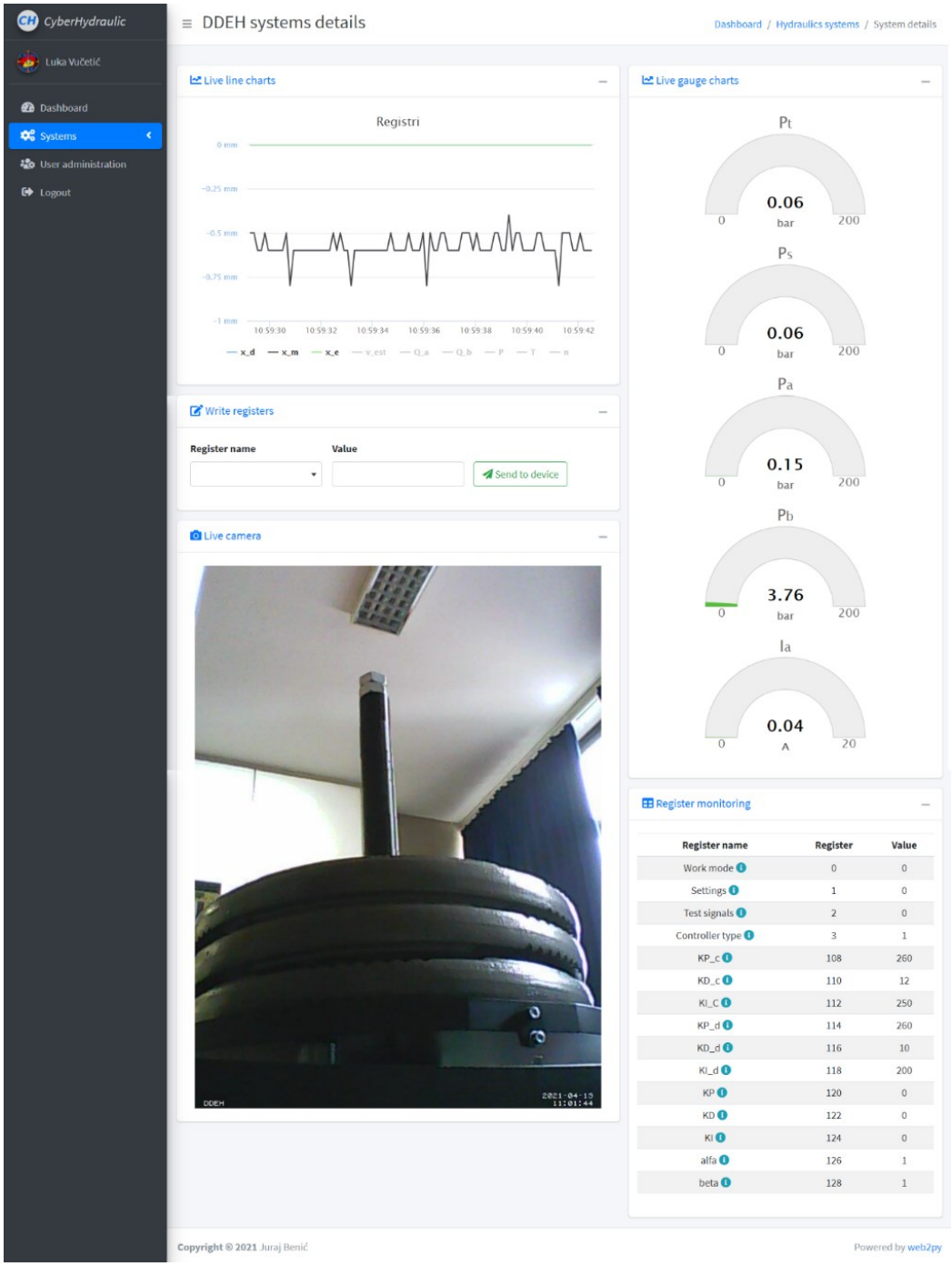
```
1.    $("#form-write-data").submit(function(e) {
2.              e.preventDefault();
3.              var form_data = $(this).serializeArray();
4.              form_data[form_data.length] = { name: "id", value:
    {{=request.vars.id}} };
5.
6.              $.ajax({
7.                  url: "{{=URL('systems' , 'system_details_write_data')}}",
8.                  type: "post",
9.                  data: form_data,
10.                 dataType: "json",
11.                 beforeSend: function() {
12.                     $.loadingBlockShow()
13.                     },
14.                 success: function(data) {
15.                     $.loadingBlockHide();
16.                     if (data.status == 200){
17.                             info(data.msg);
18.                     }
19.                     else if (data.status == 400){ error(data.msg); }
20.                 },
21.             });
22.   });
```

Programski kôd 18. Funkcija za pakiranje i slanje podataka na PLC

```
1.      data = dict(request.vars)
2.      registar = mysql_select_dict("writing_registers_view", "*", "system_id =
    %s and register = %s"%(_id,data["register"]) )
3.
4.      #pakiraj unesene podatke
5.      for i in registar:
6.
7.          if i["data_type"] == "word":
8.              #varijable koje su word
9.              bw = struct.unpack( i["pack"], struct.pack(i["unpack"],
    int(data["value"])))
10.             a = con.write_multiple_registers(i["register"], bw[::-1] )
11.
12.         elif i["data_type"] == "float":
13.             #varijable koje su float
14.             b16_l1 = utils.long_list_to_word( [utils.encode_ieee(
    float(data["value"])) ] )
15.             a = con.write_multiple_registers(i["register"], b16_l1[::-1] )
```

Na programskom kôdu 18 prikazan je kôd koji podatke prije slanja pretvara u format razumljiv PLC-u te ih upisuje u registar koji je odabran iz padajućeg izbornika *Write registers*. Koristeći *for* petlju prolazi se kroz sve vrijednosti rječnika *registar* te se pomoću *if* usporedbe traži par ključa i vrijednosti *"data_type" == "word"* te se koristi funkcija *struct.unpack()* koja pretvara binarne podatke u bajtove koji se upisuju u registar PLC-a funkcijom *write_multiple_registers()*. Na isti način provjerava se *elif* usporedbom par ključa i vrijednosti

"data_type" == "float" te se koriste funkcije *float()*, *utils.encode_ieee()* i *utils.long_list_to_word()* kako bi se uneseni podatak pretvorio u bajtove koji se upisuju u registar PLC-a funkcijom *write_multiple_registers()*. Dodatkom tog kôda omogućeno je slanje podataka na PLC. Prikaz funkcionalne *Details* stranice dan je na slici 41.



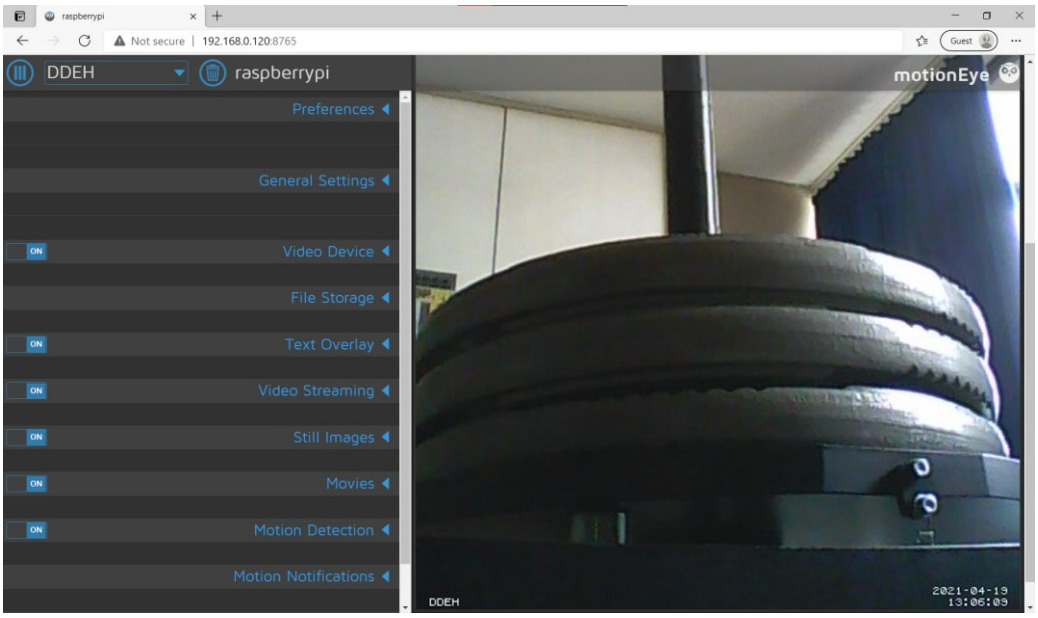
Slika 41. Funkcionalna *Details* stranica

4.6. Video prikaz postava u stvarnom vremenu

Kako bi se ostvario video prikaz postava tijekom rada, koristi se web kamera Logitech C170 i Raspberry Pi 4 Model B prikazani na slici 42.

Slika 42. Logitech C170 web kamera (lijevo); Raspberry Pi 4 (desno)

Kamera se putem USB priključka spaja na Raspberry Pi koji je spojen na usmjerivač putem Internet kabela RJ45. Na Raspberry Pi bio je postavljen operativni sustav *MotionEyeOS*, Linux distribucije, kojem je svrha biti sustav video nadzora. Video prikazu se pristupa putem web preglednika spajanjem na lokalnu IP adresu Raspberry Pi-a te dodavanjem porta za pristup *MotionEyeOS* postavkama http://192.168.0.120:8765/ prikazanim na slici 43.



Slika 43. Postavke za video prikaz

Zbog različitog osvjetljenja namještene su optimalne postavke za uvjete u laboratoriju te je podešen prikaz datuma i vremena snimke u donjem desnom kutu video prikaza. Kako bi se prikazao samo video prikaz potrebno je pristupiti linku prikazanom na programskom kôdu 19 unutar *system_details.html*. Varijabla *systems_info* dohvaća sve podatke o odabranom sustavu, što znači i njegovu globalnu IP adresu koja je potrebna za pristup Raspberry Pi-u.

Programski kôd 19. Pristup video prikazu

```
1.   var systems_info = {{=XML(systems_info)}};
2.   document.getElementById("imageid").src =
   "http://"+systems_info[0].ip+":8081/videostream.cgi?loginuse=user&loginpas="
```

Kôd *document.getElementById("imageid").src* pretražuje kroz html dokument i traži element koji posjeduje ID *imageid* prikazan u retku 18 na programskom kôdu 20 te upisuje u njega link za pristup video prikazu s programskog kôda 19. Konačan izgled video prikaza s web kamere na *Details* stranici prikazan je na slici 41.

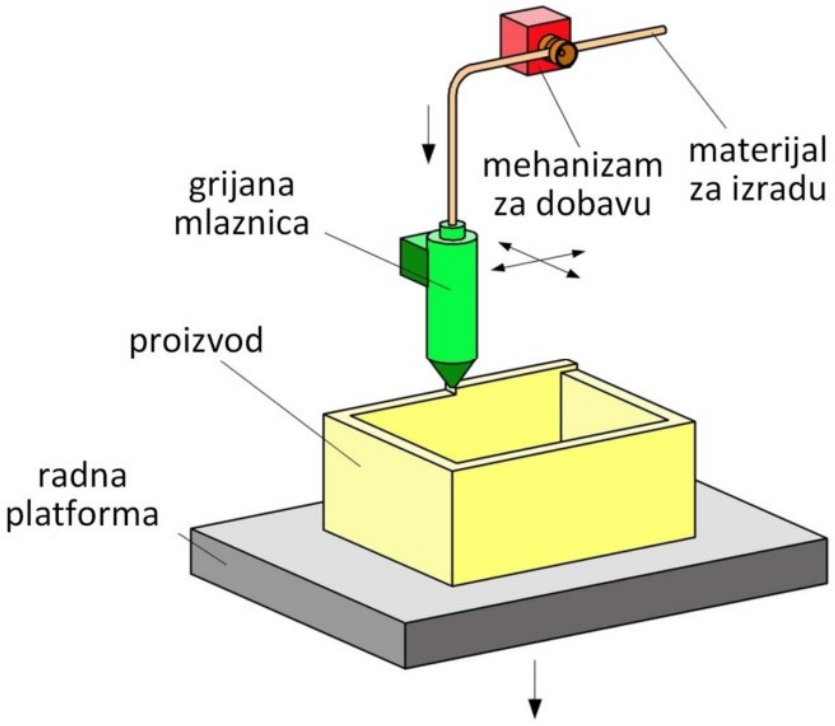
Programski kôd 20. Element za prikaz slike s kamere

```
1.          <!-- Camera -->
2.
3.         <div class="card card-outline" id="card-line-graph">
4.             <div class="card-header">
5.
6.                 <h3 class="card-title text-blue"><i class="fas fa-
   camera"></i> Live camera </h3>
7.
8.                 <div class="card-tools">
9.                     <!-- Collapse Button -->
10.                    <button type="button" class="btn btn-tool" data-card-
   widget="collapse"><i class="fas fa-minus"></i></button>
11.                </div>
12.            </div>
13.
14.            <div class="card-body">
15.
16.                <div class="container-fluid">
17.
18.                    <img id="imageid" name="main"  border="0" width="100%"
   src="">
19.
20.                </div>
21.
22.            </div>
23.
24.        </div>
```

5. PRIMJENA ADITIVNE TEHNOLOGIJE ZA RAZVOJ KONSTRUKCIJE NOSAČA

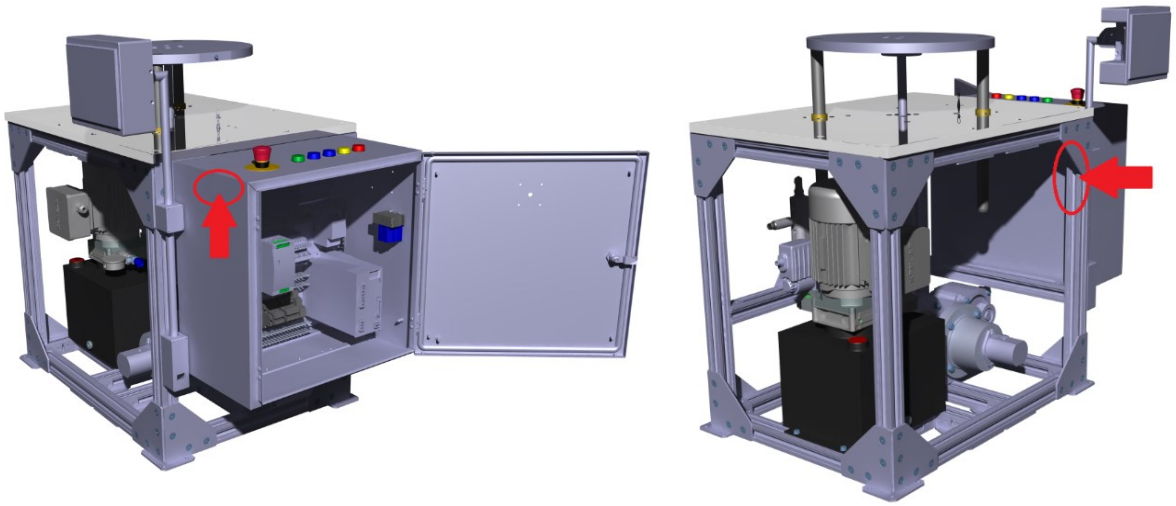
Kako bi se ostvario video prikaz sustava u stvarnom vremenu, bilo je potrebno postaviti web kameru i Raspberry Pi na sam eksperimentalni postav. Da bi se olakšala izrada nosača za web kameru i Raspberry Pi, korišten je CAD program CATIA V5 pri vizualizaciji sustava te smještaju nosača na njega.

Za izradu oba nosača korištena je FFF tehnologija (engl. *Fused Filament Fabrication*) 3D printanja, poznatija pod imenom FDM (engl. *Fused Deposition Modeling*). Radi se o postupku aditivne proizvodnje ekstrudiranjem taljene polimerne žice. Na slici 44 prikazan je shematski princip rada FDM metode. Rastaljeni polimer se nanosi sloj po sloj na X-Y (horizontalnoj ravnini), čime je ograničena mogućnost printanja kompleksnijih oblika te je potrebna suportna struktura ukoliko bridovi nisu samonosivi. [31]



Slika 44. Shematski princip rada FDM metode [31]

Na eksperimentalnom postavu razmotrilo se nekoliko mjesta za postavljanje Raspberry Pi-a i web kamere, ali se tek uz pregled već postojećeg 3D modela postava došlo do odabira. Na slici 45 prikazani su 3D modeli postava s odabranim mjestima postavljanja Raspberry Pi-a i web kamere.

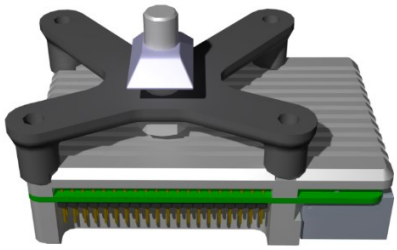


Slika 45. Odabrana lokacija za web kameru (lijevo); odabrana lokacija za Raspberry Pi (desno)

5.1. Nosač za Raspberry Pi 4

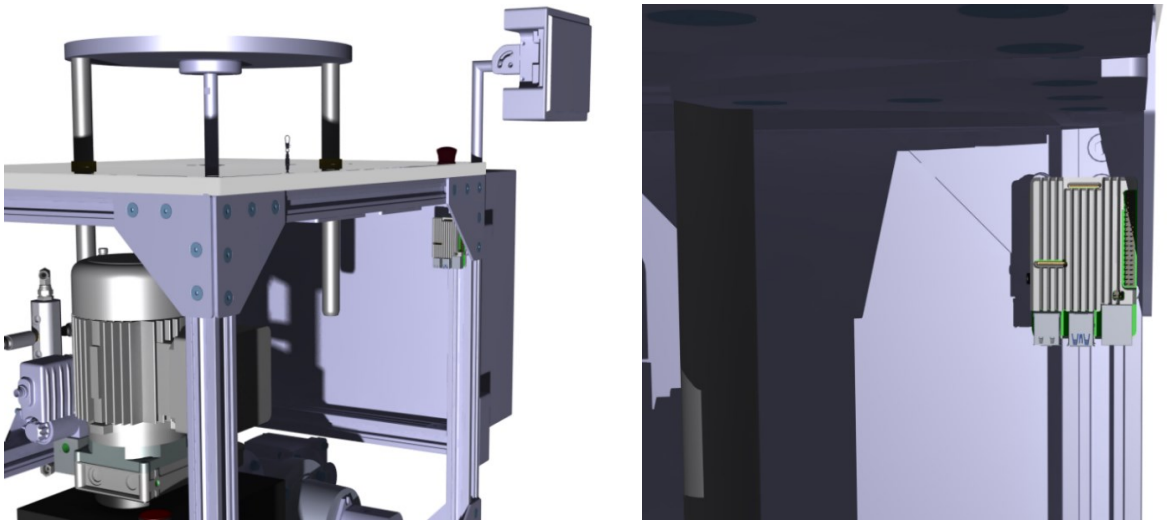
Razlog odabranom mjestu postavljanja je što Raspberry Pi ne sudjeluje aktivno u funkciji sustava, već samo procesira video prikaz, zbog čega ga je poželjno prikriti pri postavljanju. Također, razmotreno je postavljanje u upravljački ormar, no od te ideje se odustalo, jer u tako zatvorenom prostoru, zbog pojačanog grijanja prilikom obrade video toka, hlađenje Raspberry Pi-a ne bi bilo dovoljno. Kako bi se Raspberry Pi pričvrstio potrebno je konstruirati nosač koji će se pomoću četiri vijka pritegnuti na hladnjak Raspberry Pi-a te s dodatnim vijkom pritegnuti na noseću nogu postava. Nakon nekoliko iteracija odabran je nosač u obliku slova X, čiji je 3D model prikazan na slici 46. Konstruirani nosač zatim je postavljen na 3D model Raspberry Pi-a prikazan na slici 47.

Slika 46. 3D model nosača za Raspberry Pi



Slika 47. 3D model Raspberry Pi-a i nosača

Dodavanjem 3D modela Raspberry Pi-a i nosača modelu postava, odabrano je konačno mjesto postavljanja na nozi stola prikazano na slici 48. Razlog samo jednoosne simetrije nosača je mogućnost postavljanja nosača još više na nosećoj nozi sustava, kako bi se mogao sakriti iza kutnog spojnika aluminijskih profila.



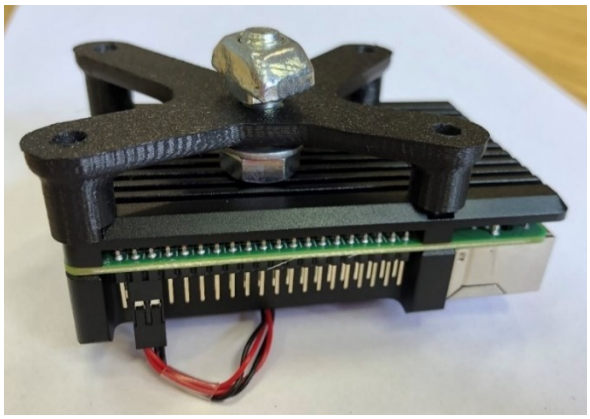
Slika 48. Prikaz modela nosača Raspberry Pi-a na sustavu (lijevo); približeni prikaz na sustavu (desno)

Nosač je konstruiran s minimalnim utroškom materijala kako bi se dodatno uštedilo pri 3D printanju. Također, na dvije nogice nosača napravljen je profilirani izrez kako bi hladnjak nasjeo te se smanjilo opterećenje na vijke. Prikaz hladnjaka sa donje strane, na koju se spaja s nosačem, dan je na slici 49.

Slika 49. Donja strana hladnjaka Raspberry Pi-a

Zadovoljavajući sve provjere kao 3D model, isprintani nosač prikazan je na slici 50. Konstruirani nosač zatim je postavljen na Raspberry Pi te pričvršćen na postav, što je prikazano na slici 51.

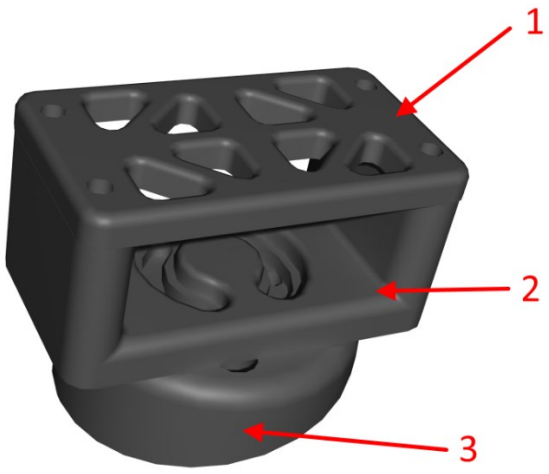
Slika 50. Isprintani nosač za Raspberry Pi



Slika 51. Raspberry Pi s nosačem (lijevo); prikaz pričvršćenog Raspberry Pi-a na postavu (desno)

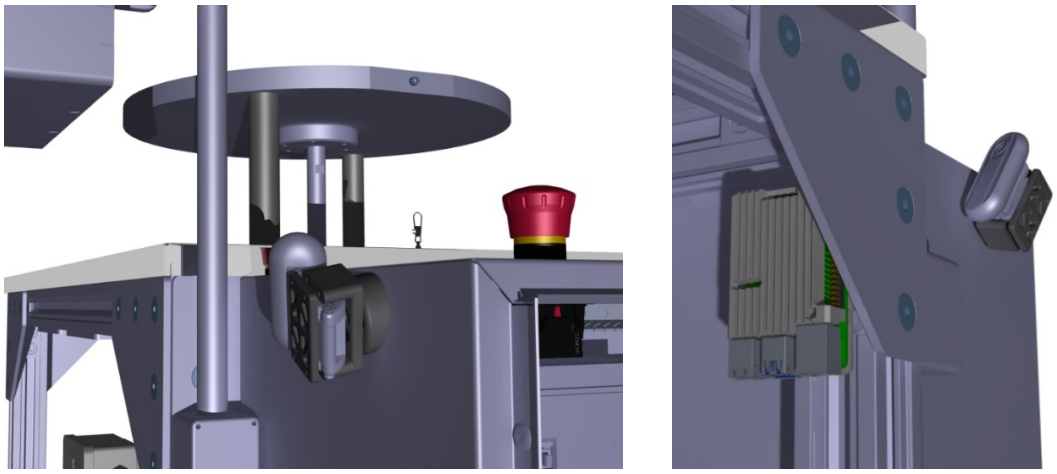
5.2. Nosač USB kamere

Odabir lokacije postavljanja kamere pokazao se problematičnim na početku. Ideja je bila montirati je u razini ljudskih očiju pomoću vertikalnog nosača koji bi se pričvrstio uz upravljački ormarić. Ta se ideja pokazala nezgodnom radi utjecaja vibracija na kameru koje postav proizvodi pri radu. Kroz nekoliko iteracija odlučeno je minimizirati udaljenost kamere od upravljačkog ormarića te je odabrano mjesto prikazano na slici 45. Budući da lokacija nije idealna za snimanje kamerom uskog vidnog polja, konstruiran je poseban nosač koji se sastoji od tri dijela: gornjeg (1) i donjeg (2) držača kamere te konusne prirubnice (3), čiji su 3D modeli prikazani u sklopu na slici 52.

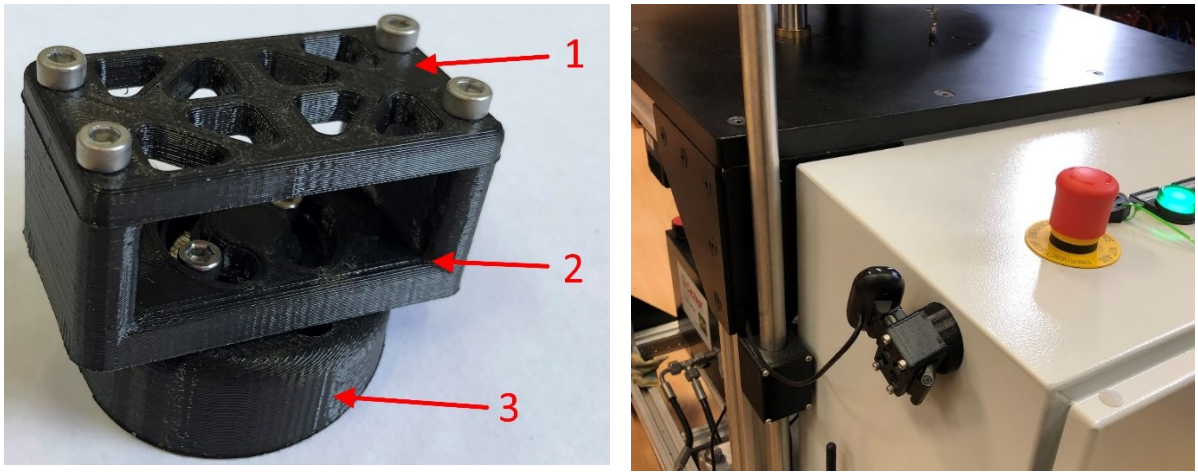


Slika 52. 3D model nosača kamere

U prvoj iteraciji prirubnica je bila paralelnih strana, no testiranjem se ispostavilo da je video prikaz kamere ukošen zbog kuta iz kojeg snima. Nakon nekoliko pokušaja odabran je pristup konusne prirubnice čime se ispravlja kut snimanja, a u kombinaciji s donjim držačem omogućeno je dodatno fino namještanje kamere u vertikalnom smjeru. Gornji držač kamere konstruiran je s namjerom smanjenja korištenog materijala pri printanju te se pomoću četiri dugačka vijka spaja s donjim držačem. Spoj donjeg držača i konusne prirubnice izveden je s dva vijka koji kroz donji nosač prolaze kroz žljebove kako bi se omogućila rotacija oko osi prirubnice za namještanje vertikalnog kuta kamere. U konačnici, spoj prirubnice s upravljačkim ormarom izveden je s dva vijka. Na slici 53 prikazan je model nosača kamere postavljen na postavu. Zadovoljavajući sve provjere kao 3D model, isprintani dijelovi nosača prikazani su u sklopu na slici 54 lijevo, a na slici 54 desno prikazana je montirana web kamera na eksperimentalnom postavu.



Slika 53. Prikaz modela nosača kamere na postavu (lijevo); prikaz modela nosača kamere na postavu iz drugog kuta (desno)



Slika 54. Sklopljeni nosač kamere (lijevo); montirana web kamera iz drugog kuta (desno)

6. ZAKLJUČAK

U ovome radu prikazan je proces unaprjeđenja postojeće web aplikacije koristeći Web2Py, baze podataka MariaDB te ModbusTCP/IP protokola za komunikaciju PLC-a sa serverom. Otklonjene su glavne greške te je osposobljena i unaprijeđena funkcija web aplikacije s eksperimentalnim hidrauličkim postavom. Objašnjena je važnost prikupljanja podataka u baze podataka te su istražene mogućnosti primjene tih baza u užurbanom razvoju tehnologija Interneta stvari.

Svrha unaprjeđene aplikacije je prikazati sakupljene podatke iz baze u korisnički pristupačnom obliku putem web aplikacije postavljene na serveru, kojoj se pristupa bilo gdje u svijetu koristeći web preglednik. Osim pregleda podataka putem dijagrama i tablica, omogućeno je i upravljanje samim sustavom na daljinu, koristeći ModbusTCP/IP protokol. Također, implementiran je i video prikaz postava koristeći Raspberry Pi i web kameru, za koje je bilo potrebno odabrati lokaciju postavljanja na sustav te konstruirati i primijeniti nosače kojima će biti pričvršćeni za sustav.

Mogućnosti web aplikacija eksponencijalno se povećavaju te se svakim danom pronalaze inovativne ideje za prikazom podataka i upravljanjem na daljinu. Prijedlozi za buduća poboljšanja su proširenje povijesnog pregleda novim mogućnostima, poput tražilice te dijagrama stanja sustava i prikaza vremena i identifikacije korisnika koji su pristupili sustavu. Isto tako, još jedan prijedlog za buduća poboljšanja je dodavanje sadržaja na početnu stranicu *Dashboard* poput korisnih informacija o vremenskom stanju unutar laboratorija te pregleda glavnih informacija o sustavu.

LITERATURA

[1] Data Preprocessing : Concepts, https://towardsdatascience.com/data-preprocessing-concepts-fa946d11c825, pristupljeno 23.4.2021.

[2] Senzor tlaka, https://www.hydac.com/de-en/products/sensors/pressure-sensors/pressure-transmitters/hda-7400.html, pristupljeno 7.9.2020.

[3] Senzor tlaka, https://www.esi-tec.com/datasheet/standard-pressure-sensor.pdf, pristupljeno 7.9.2020.

[4] Senzor protoka, https://m.hydac.com/jp-ja/products/sensors/flow-rate-sensors/flow-rate-transmitters/evs-3100.html, pristupljeno 7.9.2020.

[5] Senzor pomaka, https://www.sankyointernational.co.jp/products/wire_type_displacement_meter/item_108, pristupljeno 7.9.2020.

[6] Senzor struje, https://www.rexelusa.com/usr/Root-Category/Control%2C-Automation/Measuring%2C-Monitoring-%26-Logic-Devices/Programmable-Logic-Controllers/Field-Conversion-Modules/Square-D-RMCA61BD-Relay%2C-Analog-Converter%2C-24VDC-Supply%2C-0-5A-Input%2C-0---20mA-Output/p/894027, pristupljeno 7.9.2020.

[7] K. Ashton, „That Internet of things thing“, http://www.itrco.jp/libraries/RFIDjournal-That%20Internet%20of%20Things%20Thing.pdf, pristupljeno 6.9.2020.

[8] Categorising IoT devices - endpoints, enablers and controllers, https://deviceatlas.com/blog/categorising-iot-devices-endpoints-enablers-and-controllers

[9] IoT Applications, https://data-flair.training/blogs/iot-applications/, pristupljeno 6.9.2020.

[10] Smart Home, http://visioforce.com/smarthome.html, pristupljeno 7.9.2020.

[11] Wearable technology, https://en.wikipedia.org/wiki/Wearable_technology, pristupljeno 6.9.2020.

[12] Mi Smart Band 4, https://www.mi.com/in/mi-smart-band-4, pristupljeno 7.9.2020.

[13] Enabling Technologies for the Internet of Health Things, https://www.researchgate.net/figure/Different-types-of-wearable-technology_fig5_322261039, pristupljeno 7.9.2020.

[14] Smart cities, https://www.thalesgroup.com/en/markets/digital-identity-and-security/iot/inspired/smart-cities, pristupljeno 6.9.2020.

[15] V. Miz and V. Hahanov, "Smart traffic light in terms of the cognitive road traffic management system (CTMS) based on the Internet of Things", Kiev, 2014.

[16] M. Caliskan, A. Barthels, B. Scheuermann and M. Mauve, "Predicting Parking Lot Occupancy in Vehicular Ad Hoc Networks", Dublin, 2007.

[17] N. S. Kumar, B. Vuayalakshmi, R. J. Prarthana and A. Shankar, "IOT based smart garbage alert system using Arduino UNO", Singapore, 2016.

[18] K. E. Skouby and P. Lynggaard, "Smart home and smart city solutions enabled by 5G, IoT, AAI and CoT services", Mysore, 2014.

[19] Cities Pngs, https://pngio.com/images/png-a951645.html, pristupljeno 7.9.2020.

[20] Potential of Industry 4.0 to accelerate transition towards sustainable energy, https://www.unido.org/news/new-unido-report-explores-potential-industry-40-accelerate-transition-towards-sustainable-energy, pristupljeno 7.9.2020.

[21] Diagram of interactions within the MVC pattern, https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller#/media/File:MVC-Process.svg, pristupljeno 7.9.2020.

[22] Web2Py, http://web2py.com/books/default/chapter/34/01/introduction, pristupljeno 6.9.2020.

[23] Introduction to Modbus TCP/IP, https://www.prosoft-technology.com/kb/assets/intro_modbustcp.pdf, pristupljeno 6.9.2020.

[24] How does Modbus communication protocol work?, https://realpars.com/modbus-protocol/, pristupljeno 7.9.2020.

[25] Modbus TCP/IP, http://www.simplymodbus.ca/TCP.htm, pristupljeno 6.9.2020.

[26] MariaDB Platform, https://mariadb.com/kb/en/documentation/, pristupljeno 14.4.2021.

[27] HeidiSQL, https://www.heidisql.com/, pristupljeno 14.4.2021.

[28] What is highcharts?, https://software-sources.com/solutions/highsoft/, pristupljeno 16.4.2021.

[29] Highcharts, https://www.highcharts.com/blog/products/highcharts/, pristupljeno: 16.4.2021.

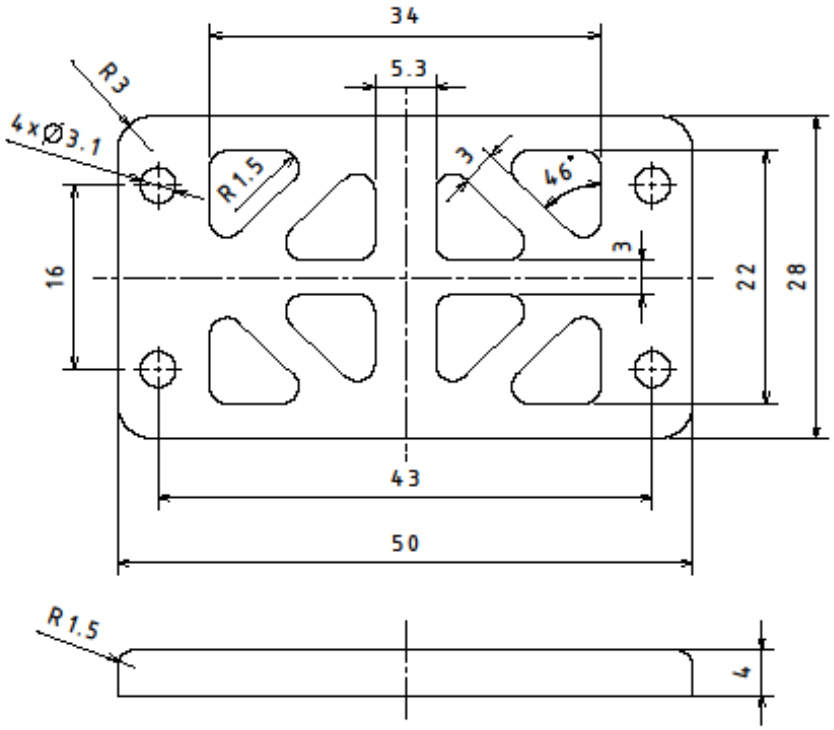
[30] Highcharts logo, https://www.highcharts.com/, pristupljeno 22.4.2021.

[31] M. Šercer, D. Godec, A. Pilipović, M. Katalenić, "Aditivna proizvodnja s polimerima", Zagreb: Fakultet strojarstva i brodogradnje; 2020.

PRILOZI

I. CD-R disc

II. Tehnička dokumentacija



	Datum	Ime i prezime	Potpis
Projektirao	27.4.2021.	Luka Vučetić	
Razradio	27.4.2021.	Luka Vučetić	
Crtao	27.4.2021.	Luka Vučetić	
Pregledao			

FSB Zagreb

Objekt:

Objekt broj:

R. N. broj:

Napomena:

Kopija

Materijal: PLA

Masa:

Naziv: Gornji držač kamere

Pozicija:

Format: A4

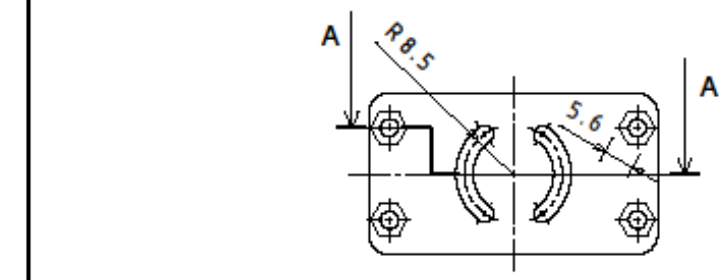
Mjerilo orginala M2:1

Listova: 4

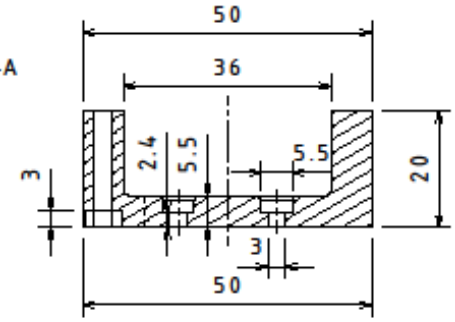
Crtež broj: PR-001-000

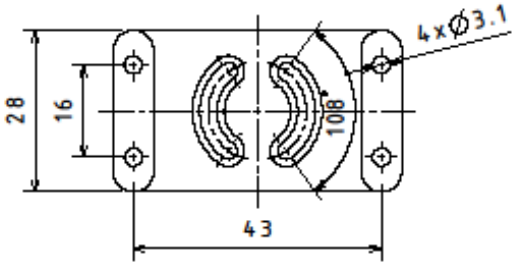
List: 1

Design by CADLab



Presjek A-A





	Datum	Ime i prezime	Potpis
Projektirao	27.4.2021.	Luka Vučetić	
Razradio	27.4.2021.	Luka Vučetić	
Crtao	27.4.2021.	Luka Vučetić	
Pregledao			

FSB Zagreb

Objekt:

Objekt broj:

R. N. broj:

Napomena:

Kopija

Materijal: PLA

Masa:

Mjerilo orginala M1:1

Naziv: Donji držač kamere

Pozicija:

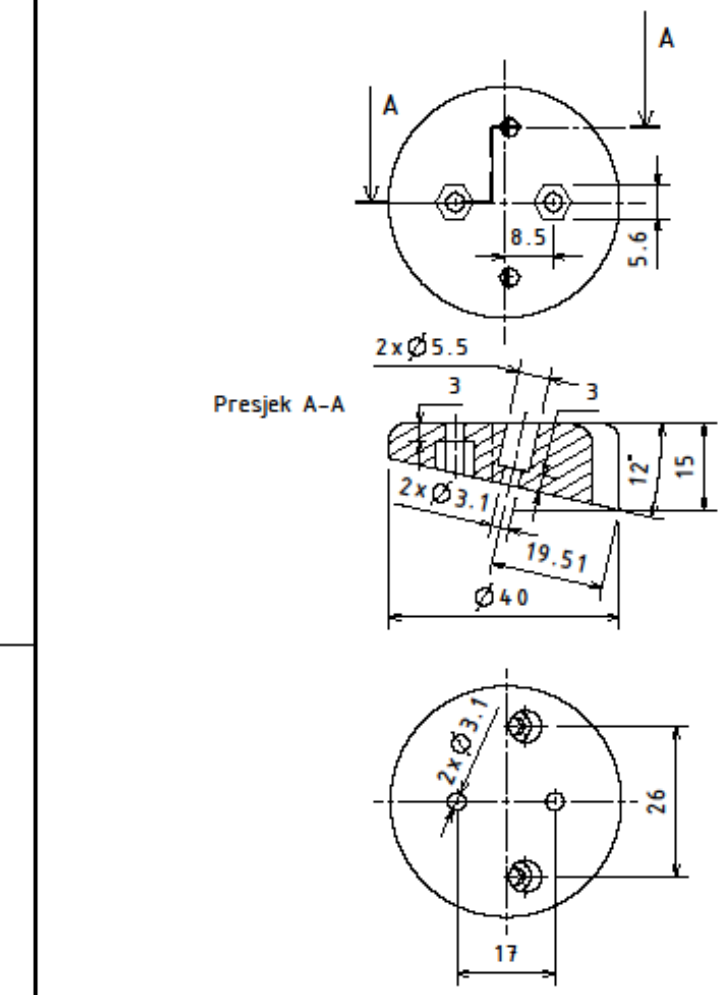
Format: A4

Listova: 4

Crtež broj: PR-002-000

List: 2

Design by CADLab



	Datum	Ime i prezime	Potpis
Projektirao	27.4.2021.	Luka Vučetić	
Razradio	27.4.2021.	Luka Vučetić	
Crtao	27.4.2021.	Luka Vučetić	
Pregledao			

FSB Zagreb

Objekt:

Objekt broj:

R. N. broj:

Napomena:

Kopija

Materijal: PLA

Masa:

Naziv: Konusna prirubnica

Pozicija:

Format: A4

Mjerilo orginala M1:1

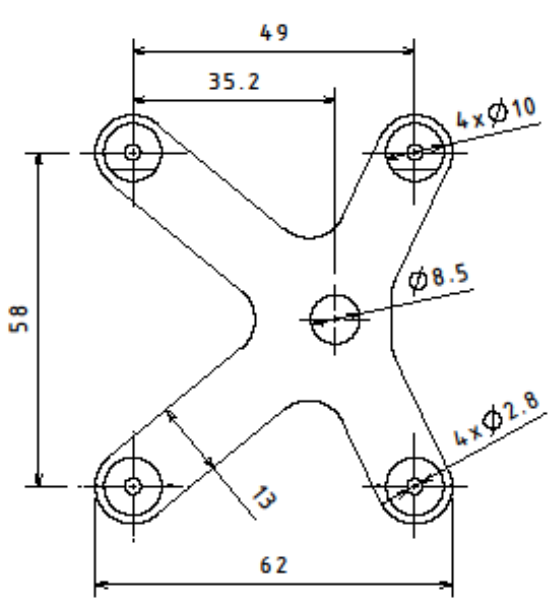
Listova: 4

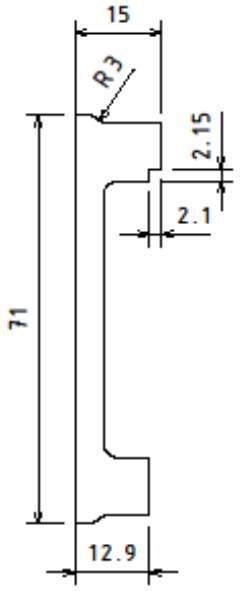
Crtež broj: PR-003-000

List: 3

Design by CADLab

NAPOMENA: Sve vanjske rubove zaobliti za 6.5mm





	Datum	Ime i prezime	Potpis
Projektirao	27.4.2021.	Luka Vučetić	
Razradio	27.4.2021.	Luka Vučetić	
Crtao	27.4.2021.	Luka Vučetić	
Pregledao			

FSB Zagreb

Objekt:	Objekt broj:
	R. N. broj:
Napomena:	Kopija
Materijal: PLA	Masa:
Mjerilo orginala: M1:1	Naziv: Nosač Raspberry Pi
Pozicija:	Format: A4
Listova: 4	List: 4
Crtež broj: PR-001-001	

Design by CADLab