



.....
Srednja šola za kemijo,
elektrotehniko in računalništvo

Pametni zalivalni sistem

Elektrotehnika

Avtorji:

Miha Žveglar R-4.a,

Marko Černezel R-4.a,

Žan Kovač R-4.a

Mentor: Boštjan Lubej, dipl. inž. inf. in tehn. kom.

Mestna občina Celje, Mladi za Celje

Celje, 2025

IZJAVA*

Mentor Boštjan Lubej v skladu z 20. členom Pravilnika o organizaciji mladinske raziskovalne dejavnosti »Mladi za Celje« Mestne občine Celje, zagotavljam, da je v raziskovalni nalogi z naslovom Pametni zalivalni sistem, katere avtorji so Marko Černezel, Miha Žvegler in Žan Kovač:

- besedilo v tiskani in elektronski obliki istovetno,
- pri raziskovanju uporabljeno gradivo navedeno v seznamu uporabljene literature,
- da je za objavo fotografij v nalogi pridobljeno avtorjevo dovoljenje in je hranjeno v šolskem arhivu,
- da sme Osrednja knjižnica Celje objaviti raziskovalno nalogo v polnem besedilu na knjižničnih portalih z navedbo, da je raziskovalna naloga nastala v okviru projekta Mladi za Celje,
- da je raziskovalno nalogo dovoljeno uporabiti za izobraževalne in raziskovalne namene s povzemanjem misli, idej, konceptov oziroma besedil iz naloge ob upoštevanju avtorstva in korektnem citiranju,
- da smo seznanjeni z razpisni pogoji projekta Mladi za Celje.

Celje, 2. 4. 2025



Podpis mentorja

Boštjan Lubej

Podpis odgovorne osebe

[Signature]

*

POJASNILO

V skladu z 20. členom Pravilnika raziskovalne dejavnosti »Mladi za Celje« Mestne občine Celje je potrebno podpisano izjavo mentorja (-ice) in odgovorne osebe šole vključiti v izvod za knjižnico, dovoljenje za objavo avtorja (-ice) fotografskega gradiva, katerega ni avtor (-ica) raziskovalne naloge, pa hrani šola v svojem arhivu.

Kazalo vsebine

| | | |
|-------|--|----|
| 1 | Uvod..... | 1 |
| 1.1 | Hipoteze..... | 1 |
| 1.2 | Cilji | 2 |
| 1.3 | Raziskovalne metode..... | 2 |
| 2 | Teoretični del..... | 3 |
| 2.1 | Splošno o pametnih zalivalnih sistemih..... | 3 |
| 2.2 | Najpopularnejši komercialni pametni zalivalni sistemi..... | 3 |
| 2.2.1 | Netro | 3 |
| 2.2.2 | Rachio..... | 3 |
| 2.2.3 | Gardena Smart System | 4 |
| 2.2.4 | Rain Bird | 5 |
| 2.3 | Uporabljena tehnologija | 6 |
| 2.3.1 | Arduino | 6 |
| 2.3.2 | Arduino Uno | 7 |
| 2.3.3 | Python..... | 7 |
| 2.3.4 | Flutter..... | 7 |
| 3 | Sestavni deli našega sistema..... | 8 |
| 3.1 | Mikrokontroler Arduino Uno | 8 |
| 3.2 | HC – 05 Bluetooth..... | 9 |
| 3.3 | Relejni modul 5V | 10 |
| 3.4 | Senzor vlažnosti zemlje | 11 |
| 3.5 | Vodna črpalka | 11 |
| 3.6 | Preizkusna ploščica in baterija 4.5V..... | 12 |
| 4 | Potek dela | 13 |
| 4.1 | Povezava komponent..... | 13 |
| 4.2 | Izdelava preprostega programa | 14 |
| 4.3 | Python strežnik | 16 |
| 4.3.1 | Koda strežnika | 17 |
| 4.4 | Spletna stran za preizkus delovanja..... | 22 |

| | | |
|-----|--|----|
| 4.5 | Mobilna aplikacija..... | 24 |
| 5 | Analiza rezultatov in razprava | 37 |
| 6 | Zaključek in smernice za nadaljnje delo..... | 39 |
| 7 | Viri..... | 40 |
| 8 | Priloge | 42 |
| 8.1 | Koda spletne strani | 42 |

Kazalo slik

| | |
|--|----|
| Slika 1: Rachio sistem | 4 |
| Slika 2: Krmilnik zalivalnega sistema Smart | 5 |
| Slika 3: Krmilnik sistema Rain Bird | 6 |
| Slika 4: Arduino UNO | 9 |
| Slika 5: Modul HC – 05 | 10 |
| Slika 6: Relejni modul 5V | 10 |
| Slika 7: Senzor vlažnosti | 11 |
| Slika 8: Vodna črpalka | 12 |
| Slika 9: Preizkusna ploščica in baterija | 12 |
| Slika 10: Povezava strojnih komponent | 13 |
| Slika 11: Arduino program 1 | 14 |
| Slika 12: Arduino program 2 | 15 |
| Slika 13: Arduino program 3 | 15 |
| Slika 14: Koda python strežnika 1 | 17 |
| Slika 15: Koda python strežnik 2 | 18 |
| Slika 16: Koda python strežnik 3 | 19 |
| Slika 17: Koda python strežnik 4 | 20 |
| Slika 18: Koda python strežnik 5 | 21 |
| Slika 19: Izgled spletne strani za preizkus | 23 |
| Slika 20: Izsek kode aplikacije za uvoz knjižnic in datotek | 24 |
| Slika 21: Izsek kode aplikacije za inicializacijo stanja in spremenljivk | 25 |
| Slika 22: Izsek kode aplikacije namenjen inicializaciji ob zagonu | 25 |
| Slika 23: Metoda fetchSensorData | 26 |
| Slika 24: Metoda connectSocket | 26 |
| Slika 25: Metoda autoModeOnV | 27 |
| Slika 26: Metoda relayStanje | 27 |
| Slika 27: Metoda AutoModeOnOff | 28 |
| Slika 28: Koda uporabniškega vmesnika aplikacije | 29 |
| Slika 29: Izsek kode potrebne inicializacije za grafe | 30 |
| Slika 30: Metoda dobiZgodovino | 30 |
| Slika 31: Izsek kode uporabniškega vmesnika za nastavljanje datuma | 31 |
| Slika 32: Izsek kode za nastavitve osi in naslovov grafa | 32 |

| | |
|---|----|
| Slika 33: Koda za nastavitve mej grafa in prikaz podatkov na grafu..... | 32 |
| Slika 34: Mobilna aplikacije – izklopljen avtomatski način | 33 |
| Slika 35: Prikaz grafa v mobilni aplikaciji..... | 34 |
| Slika 36: Prikaz grafa s podrobnostmi | 35 |
| Slika 37: Mobilna aplikacija – vklopljen avtomatski način | 36 |

Zahvala

Zahvaljujemo se vsem, ki ste kakorkoli pripomogli k nastanku naše raziskovalne naloge. Vsaka spodbuda, nasvet ali pomoč nam je pomenila veliko. Posebej pa gre zahvala našemu mentorju g. Boštjanu Lubeju za njegovo podporo, koristne napotke in potrpežljivo usmerjanje skozi celoten proces izdelave naloge in izdelka. Brez njegove pomoči nam zagotovo ne bi uspelo priti tako daleč.

Hvala tudi vsem, ki ste nas med raziskavo spodbujali in verjeli v naš projekt.

Povzetek

Čeprav današnji trg ponuja številne komercialne sisteme za avtomatsko zalivanje, smo se odločili preveriti možnost izdelave lastnega pametnega sistema, ki bi bil stroškovno dostopen, učinkovit in uporabniku prijazen. V raziskovalni nalogi smo zasnovali in razvili pametni zalivalni sistem, ki temelji na mikrokontrolniku Arduino Uno, Bluetooth modulu HC-05, relejnem modulu, vodni črpalki in senzorju za merjenje vlažnosti tal. Sistem deluje tako, da natančno spremlja stanje vlage v zemlji in samodejno prilagaja količino zalivanja glede na dejanske potrebe rastlin. Za uporabniku prijazno upravljanje sistema smo razvili tudi namensko mobilno aplikacijo v ogrodju Flutter, ki omogoča pregled trenutne vlažnosti tal, zgodovino meritev in možnost ročnega ali avtomatskega upravljanja sistema.

Za izmenjavo podatkov med Arduinom in mobilno aplikacijo smo postavili Python strežnik z uporabo ogrodja Flask, ki zagotavlja zanesljivo komunikacijo in shranjevanje podatkov v podatkovno bazo SQLite3. Na ta način smo uporabnikom omogočili tudi zgodovinski pregled meritev in analizo učinkovitosti zalivanja skozi čas. Izvedli smo vrsto praktičnih preizkusov, pri čemer se je izkazalo, da naš sistem zmanjša porabo vode za približno 40 % v primerjavi s tradicionalnim ročnim zalivanjem. Sistem ne le zmanjšuje nepotrebno porabo vode, ampak tudi prispeva k izboljšanju pogojev za rast rastlin, saj jim zagotavlja konstantno optimalno vlago. Hkrati je naša rešitev ekonomična, saj stroški vseh uporabljenih komponent niso presegli 100 €, kar je bistveno ceneje od mnogih komercialnih rešitev. Z raziskovalno nalogo smo tako dokazali, da lahko z osnovnimi tehničnimi znanji in sodobno tehnologijo izdelamo učinkovit, trajnosten in uporabniku prijazen sistem, ki ga je mogoče enostavno prilagoditi različnim potrebam in uporabniškim zahtevam.

Ključne besede

Arduino, pametni zalivalni sistem, bluetooth HC-05, Flutter, Python

Abstract

Although the market already offers various commercial smart irrigation solutions, we decided to explore the possibility of creating our own smart watering system that would be affordable, efficient, and user-friendly. In our research paper, we designed and developed a smart irrigation system based on the Arduino Uno microcontroller, HC-05 Bluetooth module, relay module, water pump, and soil moisture sensor. The system continuously monitors soil moisture levels and adjusts watering automatically according to the actual needs of plants. Additionally, we developed a mobile application using the Flutter framework, which allows users to monitor real-time moisture data, historical sensor readings, and to manually or automatically control watering.

To facilitate communication between hardware components and the mobile application, we implemented a Python server using the Flask framework. This server manages data collection, processes sensor inputs, and stores the data in a SQLite3 database for later analysis. Tests demonstrated that our smart irrigation system reduced water consumption by approximately 40% compared to traditional manual watering methods. Moreover, the overall production costs did not exceed 100 €, making this solution practical, economical, and highly effective. This project demonstrates that with basic technical skills and accessible technology, it is possible to create innovative, sustainable, and highly customizable solutions. Future improvements could include additional sensors for monitoring environmental factors, further optimization of watering schedules, and expanded mobile app functionality.

Keywords

Arduino, Smart Irrigation System, bluetooth HC-05, Flutter, Python

Kratice in okrajšave

HC-05 – Bluetooth modul

DC – Direct Current (enosmerni tok)

API – Application Programming Interface

JSON – JavaScript Object Notation

IDE – Integrated Development Environment

HTTP – Hypertext Transfer Protocol

USB – Universal Serial Bus

IoT – Internet of Things

SQL – Structured Query Language

UTC – Coordinated Universal Time (koordinirani univerzalni čas)

RX – Receive (sprejem podatkov)

TX – Transmit (oddajanje podatkov)

1 Uvod

Tehnologija postaja vedno bolj osrednji del našega vsakdana, saj skozi inovacije in digitalne rešitve premikamo meje mogočega ter izboljšujemo načine, kako skrbimo za našo okolico in vire, ki jih imamo na voljo. V dobi, ko so podnebne spremembe in zmanjševanje naravnih virov izziv, je ključnega pomena iskanje rešitev, ki omogočajo učinkovito in trajnostno rabo teh virov. Eden izmed primerov inovativnega pristopa k temu problemu je razvoj sistemov za avtomatizirano zalivanje, ki omogočajo prilagajanje količine vode glede na specifične potrebe rastlin.

Pametni zalivalni sistemi predstavljajo nov način upravljanja z vodo, kjer tradicionalni, fiksni načini zalivanja nadomešča dinamičen pristop, prilagojen trenutnim razmeram. Takšni sistemi z uporabo senzorjev in avtomatiziranih procesov omogočajo natančno spremljanje stanja okolja in pravočasno odzivanje, s čimer se preprečuje prekomerna poraba vode in zagotavlja optimalna oskrba rastlin.

V tej raziskovalni nalogi se osredotočamo na raziskovanje koncepta pametnih zalivalnih sistemov kot primerov, kako lahko sodobna tehnologija pripomore k reševanju realnih okolijskih izzivov. Namen raziskave je preučiti načine, kako z integracijo digitalnih rešitev doseči večjo učinkovitost in trajnost v procesu zalivanja, ter na splošno pokazati, da inovativni pristopi lahko prinesejo pomembne spremembe v tradicionalnih metodah upravljanja.

1.1 Hipoteze

Za usmerjeno delo smo si pred začetkom dela postavili nekaj hipotez.

H1 – Stroški izdelave sistema ne bodo presegli 100€

H2 – Z zalivalnim sistemom bi bil 30% boljši izkoristek vode pri zalivanju

H3 – Večina vrtnarjev in kmetov bo zaradi prihrankov in boljših pogojev raje investirala v pametno zalivanje

H4 – Realno časovni nadzor in avtomatizacija zalivanja bosta zmanjšala nepotrebno zalivanje in s tem povečala donos pridelka rastlin za vsaj 15 %

1.2 Cilji

Naš cilj je izdelati pametni zalivalni sistem, ki bo samodejno prilagajal količino vode glede na potrebe rastlin, s čimer bomo optimizirali rabo vode in izboljšali pogoje za rast. Z sistemom želimo zagotoviti, da se zalivanje izvaja na podlagi dejanskega stanja vlage v tleh, kar omogoča natančno in učinkovito oskrbo rastlin, hkrati pa zmanjšuje nepotrebno porabo vode. Namen projekta je tudi ustvariti enostaven nadzor nad celotnim sistemom, tako da bo uporabniku omogočeno spremljanje in, če bo potrebno, ročno prilagajanje nastavitev. S tem želimo dokazati, da lahko s sodobnimi tehnologijami in premišljenimi pristopi dosežemo trajnostne rešitve, ki so praktične in dostopne tudi za uporabnike z omejenim tehničnim znanjem.

1.3 Raziskovalne metode

Pri raziskovanju našega pametnega zalivalnega sistema nam bo v veliko pomoč svetovni splet, ker gradiv v tiskani obliki o teh rešitvah ni veliko. Na začetku bomo uporabili primarne vire, kot so uradne spletne strani in dokumentacija za komponente, ki jih uporabljamo – na primer Arduino Uno, HC-05 Bluetooth modul, 5V relejni modul, senzor vlage, Python (Flask in SQLite) ter mobilno aplikacijo, izdelano v Flutterju. Te informacije nam bodo pomagale, da bomo imeli točne podatke o vseh napravah in orodjih, ki jih bomo uporabili.

Poleg tega smo se opirali na sekundarne vire – članke, forume in YouTube videe, kjer so razvijalci in uporabniki delili svoje izkušnje z avtomatiziranimi sistemi za zalivanje. Ti viri so nam omogočili vpogled v praktične izzive in rešitve, s katerimi so se soočali drugi, ter nam pomagali prepoznati, katere funkcionalnosti so najbolj zaželenne. S tem smo si zagotovili temeljit pregled obstoječih rešitev in dobili dragocene informacije, ki so nam služile kot osnova za nadaljnje izboljšave našega sistema.

2 Teoretični del

2.1 Splošno o pametnih zalivalnih sistemih

Pametni zalivalni sistemi predstavljajo sodoben pristop k avtomatizaciji namakanja, ki temelji na natančnem spremljanju stanja tal in prilagajanju količine vode glede na dejanske potrebe rastlin. S pomočjo senzorjev, ki merijo vlago, temperaturo in dež, ter naprednih krmilnih algoritmov, ti sistemi omogočajo optimalno rabo vode in prispevajo k izboljšanju pogojev za rast. Običajni sistemi, ki delujejo na fiksnih urnikih, so zamenjani z rešitvami, ki se samodejno prilagajajo vremenskim razmeram in specifičnim zahtevam vrta, kar vodi k večji učinkovitosti in varčevanju z vodo.

2.2 Najpopularnejši komercialni pametni zalivalni sistemi

Med vodilne izdelke na trgu sodijo sistemi, kot so Netro, Rachio, Gardena Smart System in Rain Bird. Vsak od teh sistemov se ponaša s svojimi specifičnimi funkcionalnostmi, tehnologijami in cenovnimi razredi.

2.2.1 Netro

Netro je sodoben sistem, ki se osredotoča na popolno avtonomijo pri prilagajanju zalivanja. S pomočjo vgrajenih senzorjev in vremenskih podatkov sistem natančno prilagodi količino vode glede na specifične potrebe rastlin in tip tal. Prednost Netra je v njegovem algoritmu, ki samodejno prilagaja urnik zalivanja in tako učinkovito varčuje z vodo. Cena Netro sistemov se giblje okoli 100–130 €, kar ga uvršča med cenovno dostopne rešitve za uporabnike.

2.2.2 Rachio

Rachio velja za enega vodilnih sistemov na področju pametnega zalivanja, saj nudi napredno prilagajanje urnikov na podlagi lokalnih vremenskih podatkov. Rachio omogoča konfiguracijo več con, kjer uporabnik v aplikaciji vnese podatke o vrsti rastlin, tleh in postavitvi vrta, sistem pa na podlagi teh informacij ter aktualnih vremenskih razmer optimizira namakanje. Zaradi svoje integracije z mobilnimi

aplikacijami in glasovnimi asistenti je Rachio priljubljena izbira med uporabniki, čeprav so njegove cene višje – 8-conski model se običajno giblje okoli 230 €.



Slika 1: Rachio sistem

Vir (<https://cdn.sanity.io/images/1d0v1es1/production/781f5090a9bf1e8b8e236182ef9ccd66a4967e2a-1800x1800.jpg?w=410&h=410&auto=format>)

2.2.3 Gardena Smart System

Gardena, tradicionalni evropski proizvajalec vrtnarske opreme, ponuja svoj Smart System, ki združuje pametne krmilnike, senzorje in dodatke za celovito upravljanje vrta. Sistem Smart Water Control omogoča popolnoma avtomatsko zalivanje, pri čemer se prilagaja na podlagi vlage tal in vremenskih razmer. Gardena Smart System je zasnovan kot del integriranega ekosistema, vendar so cene teh rešitev nekoliko višje.



Slika 2: Krmilnik zalivalnega sistema Smart

Vir (https://m.media-amazon.com/images/I/71LeYFtp7ML._AC_UF350,350_QL80_.jpg)

2.2.4 Rain Bird

Rain Bird je dolgoletni proizvajalec namakalne opreme, ki je s svojimi pametnimi krmilniki uspešno združil zanesljivost tradicionalnih sistemov s sodobno tehnologijo. Njihovi modeli omogočajo avtomatsko sezonsko prilagajanje in dežne odloge, pri čemer ohranijo možnost lokalnega nadzora prek fizičnih gumbov in zaslona. Rain Bird krmilniki so robustni in primerni tako za profesionalne kot za rezidenčne namakalne sisteme, čeprav morda ne nudijo tako naprednih funkcij kot nekateri novejši sistemi.



Slika 3: Krmilnik sistema Rain Bird

Vir (https://www.rainbird.com/sites/default/files/styles/max_550x550/public/media/images/2018-10/esp-rzxe-1_whitev2.jpg?itok=Oj_igtja)

2.3 Uporabljena tehnologija

2.3.1 Arduino

Arduino predstavlja osrednji gradnik našega pametnega zalivalnega sistema, saj z uporabo enostavnih in cenovno dostopnih komponent omogoča zbiranje podatkov ter upravljanje z aktuatorji. V nasprotju z nekaterimi sistemi, ki temeljijo na zmogljivejših računalnikih, smo se za naš projekt odločili za Arduino, ker je idealen za hitre in prilagodljive rešitve, ki jih lahko razvijemo s srednješolskim znanjem.

2.3.2 Arduino Uno

Arduino Uno je mikrokontroler, ki ga je razvila Arduino skupnost in je eden najbolj priljubljenih modelov zaradi svoje preprostosti in odprtokodne narave. Na plošči so prisotni digitalni in analogni vhodi ter izhodi, kar omogoča enostavno priklopjanje senzorjev in aktuatorjev. V našem projektu bo Arduino Uno služil za zbiranje podatkov o vlažnosti tal s pomočjo priključenega senzorja in za upravljanje releja, ki bo aktiviral črpalko za zalivanje. Zaradi enostavnega programiranja in obsežne skupnosti razvijalcev je Arduino Uno odlična izbira, ki omogoča hitro implementacijo in prilagoditve sistema.

2.3.3 Python

Python je visoko nivojski, objektno orientiran, brezplačen in odprtokoden programski jezik, ki ga je leta 1990 ustvaril Guido Van Rossum. V našem pametnem zalivalnem sistemu smo Python uporabili za razvoj centralnega strežnika, ki je hrbtenica celotnega sistema. Strežnik, izdelan z uporabo ogrodja Flask, je zadolžen za sprejemanje HTTP zahtev s strani mobilne aplikacije in za obdelavo podatkov, ki jih posreduje Arduino preko Bluetooth povezave (HC-05). Meritve senzorjev, kot so podatki o vlagi tal, se shranjujejo v SQLite3 bazo podatkov, kar omogoča nadaljnjo analizo in optimizacijo urnika zalivanja. Poleg tega Python strežnik uporablja tehnologijo WebSockets za realno časovno posodabljanje informacij v mobilni aplikaciji, s čimer zagotavlja, da uporabnik vedno dobi ažurne podatke o stanju sistema. S tem je Python ključen za usklajevanje in avtomatizacijo procesa zalivanja ter za zagotavljanje integrirane komunikacije med strojno in programsko opremo.

2.3.4 Flutter

Flutter je ogrodje, ki ga je razvil Google in omogoča razvoj vizualno privlačnih aplikacij za mobilne naprave, splet in namizje iz enotne kode v programskem jeziku Dart. V našem projektu smo Flutter uporabili za izdelavo mobilne aplikacije, ki uporabniku omogoča enostaven pregled nad celotnim pametnim zalivalnim sistemom. Aplikacija prikazuje podatke o vlažnosti tal, statusu črpalke in nastavitvah urnikov zalivanja ter omogoča ročno upravljanje sistema, na primer z vklopom ali izklopom črpalke. Komunikacija med mobilno aplikacijo in Python strežnikom poteka

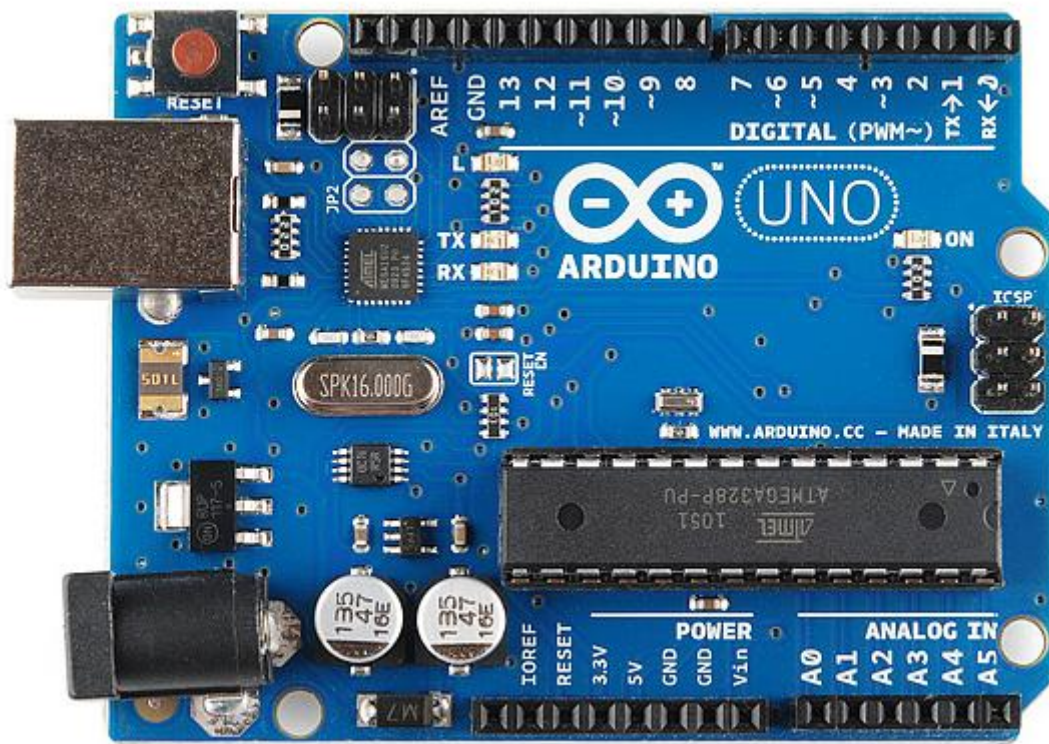
prek HTTP zahtev, medtem ko se za realno časovno posodabljanje informacij uporablja tehnologija WebSockets. S tem pristopom aplikacija zagotavlja intuitiven in odziven vmesnik, ki omogoča uporabniku, da kadar koli spremlja in prilagaja delovanje sistema, s čimer se doseže večja učinkovitost in prilagodljivost celotnega zalivalnega sistema.

3 Sestavni deli našega sistema

Spodaj so naštet in opisani vsi naši sestavni deli sistema. Za izdelavo našega pametnega zalivalnega sistema smo uporabili naslednje strojne komponente.

3.1 Mikrokontroler Arduino Uno

Krmilno vezje Arduino Uno je ena izmed najbolj priljubljenih razvojnih plošč zaradi svoje enostavnosti in vsestranskosti. Plošča vsebuje 14 digitalnih vhodov/izhodi, med katerimi jih je mogoče konfigurirati kot vhode ali izhode, ter 6 analognih vhodov, ki se uporabljajo za branje senzornih signalov. Arduino Uno se napaja s stabilno 5V enosmerno napetostjo in ga je mogoče napajati prek USB kabla ali z zunanjim napajalnikom, kar zagotavlja fleksibilnost pri uporabi. Zaradi kompatibilnosti z Arduino IDE in številnimi knjižnicami omogoča enostavno programiranje in hitro razvojno iteracijo. V našem izdelku bo Arduino Uno deloval kot možgani celotnega sistema, saj bo nadzoroval delovanje senzorjev ter upravljal s črpalko za zalivanje, s čimer bo zagotavljal optimalno prilagajanje namakanja glede na dejanske potrebe rastlin.

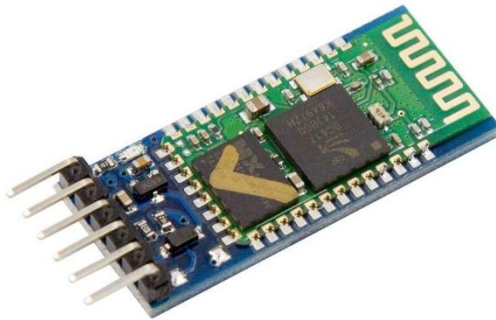


Slika 4: Arduino UNO

Vir (<https://cdn.sparkfun.com/assets/9/1/e/4/8/515b4656ce395f8a38000000.png>)

3.2 HC – 05 Bluetooth

HC-05 je brezžični modul, ki je zasnovan za enostavno integracijo z Arduino in drugimi mikrokontrolerji. Modul omogoča serijsko komunikacijo prek Bluetootha, kar zagotavlja dvosmerni prenos podatkov na razdalji do približno 10 metrov. HC-05 deluje na 5V napetosti in ima na voljo standardne serijske pinje (TX in RX), s katerimi se enostavno poveže s kontrolno enoto. Zaradi svoje preprostosti uporabe, nizke cene in široke razpoložljivosti je HC-05 priljubljena izbira med razvijalci. V našem pametnem zalivalnem sistemu bo HC-05 modul omogočal brezžični prenos podatkov, pri čemer bodo meritve senzorjev, kot so podatki o vlažnosti tal, posredovane iz Arduino Uno enote na centralni Python strežnik. Ta povezava je ključnega pomena za zagotavljanje hitrega in zanesljivega prenosa informacij, ki omogoča realno časovno spremljanje in prilagajanje namakanja.



Slika 5: Modul HC – 05

Vir (<https://si.szks-kuongshun.com/Content/upload/201884835/201811121746507923437.jpg>)

3.3 Relejni modul 5V

Relejni modul 5V omogoča nadzor nad napravami z višjimi tokovi s pomočjo nizkonapetostnih signalov, ki jih pošilja Arduino. Rele je sestavljen iz elektromagnetnega mehanizma, ki ob prejemu ustreznega signala prek digitalnega izhoda preklopi med normalno odprtimi in normalno zaprtimi kontakti. V našem pametnem zalivalnem sistemu bo ta rele uporabljen za vklop in izklop 4.5V DC črpalke za vodo. S tem lahko preko programirane logike v Arduino mikrokontrolerju natančno določimo, kdaj je treba črpalke aktivirati glede na meritve senzorja vlage tal. Modul deluje na 5V napajanju, kar je združljivo z ostalimi komponentami, in zagotavlja zanesljivo ter hitro odzivnost, kar je ključnega pomena za optimalno upravljanje namakanja.



Slika 6: Relejni modul 5V

Vir (https://www.antoniadis.com.cy/27956-thickbox_default/5v-relay-module-1-channel-low-level-trigger.jpg)

3.4 Senzor vlažnosti zemlje

Rastline so odvisne od ustrezne količine vode, zato je v našem zalivalnem sistemu ključno natančno spremljanje vlage v tleh. Za ta namen uporabljamo senzor za merjenje vlage. Senzor meri vlažnost v tleh v procentih in, ko se ta zmanjša pod določeno mejo, mikrokrmilnik sproži alarm, ki signalizira, da je rastlino potrebno zaliti. Senzor ima na obeh straneh nameščene prevodne žice, ki ob stiku s tleh, bogatih z vodo, začnejo prevajati električni tok. Meritve se izvajajo s pomočjo uporov – večja prisotnost vode v zemlji zmanjša upornost in posledično poveča izmerjeno napetost. Na podlagi teh meritev mikrokrmilnik s programom izračuna procentno vrednost vlažnosti, kar omogoča natančno prilagajanje količine zalivanja glede na dejanske potrebe rastlin.



Slika 7: Senzor vlažnosti

Vir (<https://europe1.discourse->

[cdn.com/arduino/original/4X/b/f/5/bf59c155277511be834da260534b8200200ebebe.jpeg](https://europe1.discourse-cdn.com/arduino/original/4X/b/f/5/bf59c155277511be834da260534b8200200ebebe.jpeg))

3.5 Vodna črpalka

Črpalka je ključna komponenta našega zalivalnega sistema, saj s svojo funkcijo prenaša vodo do rastlin, ko sistem zazna, da je vlaga v tleh prenizka. V našem projektu uporabljamo 4.5V DC črpalko, ki je posebej primerna za majhne in srednje velike namakalne sisteme. Ko senzor vlage zazna, da vlaga v tleh pade pod določeno mejno vrednost, mikrokrmilnik pošlje signal, ki prek releja sproži delovanje črpalke. Tako se črpalka aktivira in črpa vodo iz rezervoarja, ki jo nato preko cevi prenaša do korenin rastlin.



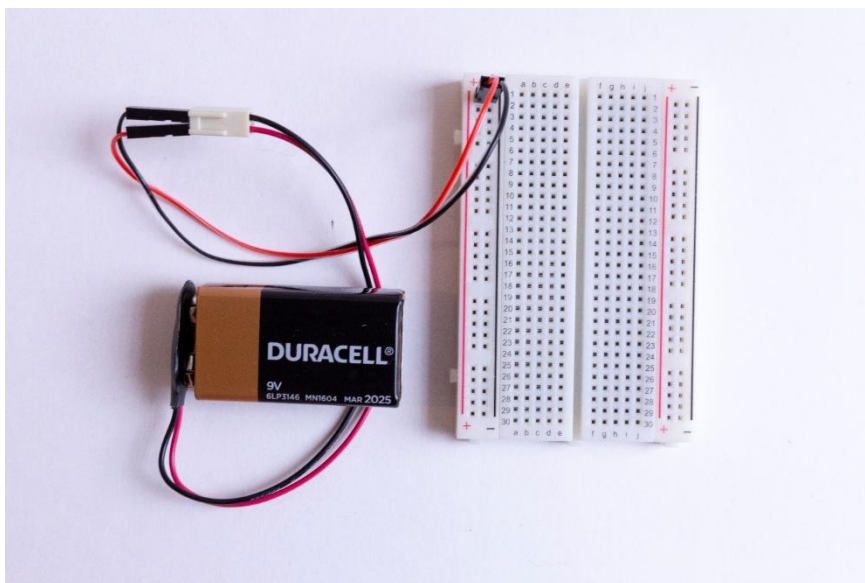
Slika 8: Vodna črpalka

Vir (<https://i0.wp.com/www.sensingthecity.com/wp-content/uploads/2023/04/pump.png?resize=580,327&ssl=1>)

3.6 Preizkusna ploščica in baterija 4.5V

V našem sistemu uporabljamo 4.5V DC baterijo, ki zagotavlja potrebno energijo predvsem za delovanje črpalke in drugih komponent, ko sistem ne more biti napajan iz električnega omrežja. Ta kompaktna in energetske učinkovita baterija omogoča zanesljivo delovanje sistema tudi v odsotnosti stalnega napajanja.

Za sestavljanje in testiranje elektronskih vezij uporabljamo breadboard – prototipno ploščo, ki omogoča enostavno in prilagodljivo povezovanje vseh komponent brez potrebe po spajkanju. Breadboard nam omogoča hitro spreminjanje in preizkušanje vezij med Arduino Uno, senzorjem vlage, relejem, HC-05 modulom in ostalimi elementi, kar bistveno pospešuje razvoj našega zalivalnega sistema.



Slika 9: Preizkusna ploščica in baterija

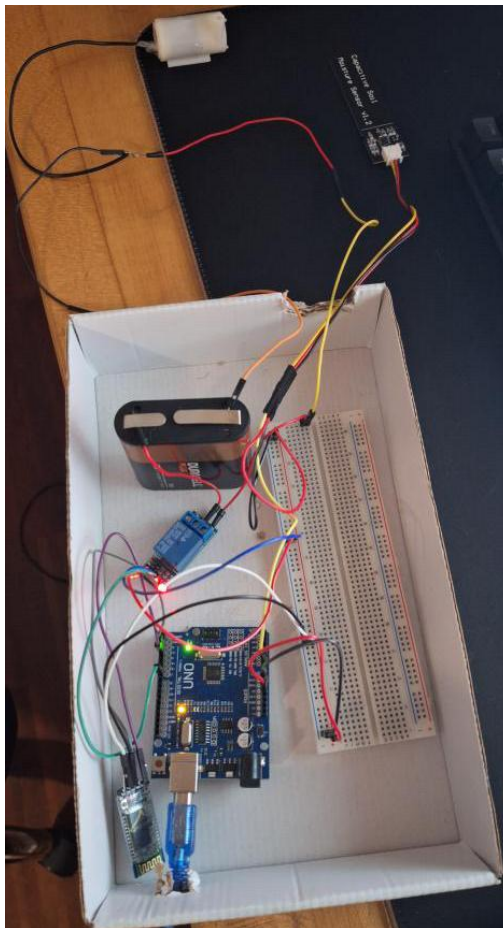
Vir (https://cdn.sparkfun.com/assets/learn_tutorials/1/6/2/6/co-lab_fankit_battery-breadboard.jpg)

4 Potek dela

Pri izdelavi našega pametnega zalivalnega sistema smo sledili več zaporednim korakom, ki so nas vodili od začetnega sestavljanja vezij do končne integracije in testiranja celotnega sistema.

4.1 Povezava komponent

Najprej smo na preizkusni ploščici povezali vse strojne komponente. Na Arduino Uno smo priključili senzor za merjenje vlage tal, HC-05 Bluetooth modul, 5V relejni modul in 4.5V DC črpalko. Povezave smo izvedli s pomočjo povezovalnih kablov, kar nam je omogočilo, da smo brez težav prilagajali vezja, če bi karkoli od nečesa zgrešili.



Slika 10: Povezava strojnih komponent

4.2 Izdelava preprostega programa

Ta začetni sestavni del smo natančno preizkusili s preprostim programom, ki je bral vrednosti s senzorja vlage. S tem smo se prepričali, da senzor pravilno meri in da se signal uspešno prenaša preko HC-05 modula, medtem ko rele natančno upravlja s črpalko.

```
#include <SoftwareSerial.h>

// Nastavljanje spremenljivk
#define RELAY_PIN 7
#define MOISTURE_PIN A0

#define AIR_VALUE 607
#define WATER_VALUE 274

SoftwareSerial BTSerial(2, 3);

void setup() {
  pinMode(RELAY_PIN, OUTPUT);
  Serial.begin(9600);
  BTSerial.begin(9600);
}
```

Slika 11: Arduino program 1

Tukaj je najprej vključena knjižnica `SoftwareSerial`, ki omogoča uporabo dodatnih serijskih povezav na priključkih, ki niso strojno podprti za serijsko komunikacijo. Uporabljamo jo za komunikacijo z Bluetooth modulom HC-05. `RELAY_PIN` predstavlja povezavo z relejem, ki ga bomo vklopili in izklopili, `MOISTURE_PIN` pa predstavlja analogni priključek za branje podatkov iz senzorja v tleh. Definirana sta tudi vrednosti senzorja na suhem zraku, ter v vodi, ki predstavljata popolnoma suha in popolnoma mokra tla. `SoftwareSerial` definira serijsko povezavo preko priključkov 2 (RX) in 3 (TX) za komunikacijo z HC-05 modulom.

```

void loop() {
    //Spreminjanje vrednosti senzorja v procente
    int soilMoistureValue = analogRead(MOISTURE_PIN);
    int soilMoisturePercent = map(soilMoistureValue, AIR_VALUE, WATER_VALUE, 0, 100);
    soilMoisturePercent = constrain(soilMoisturePercent, 0, 100);

    String sensorData = "SENSOR:" + String(soilMoisturePercent);

    Serial.println(sensorData);
    BTSerial.println(sensorData);

    // Povezovanje preko USB oziroma bluetooth
    if (BTSerial.available()) {
        char command = BTSerial.read();
        handleCommand(command, true);
    }

    if (Serial.available()) {
        char command = Serial.read();
        handleCommand(command, false);
    }

    delay(2000);
}

```

Slika 12: Arduino program 2

V tem delu kode se podatki senzorja za vlago shranijo v spremenljivko `soilMoistureValue`, ki se potem s pomočjo funkcije »`map()`« pretvorijo v procentualno vrednost vlažnosti v tleh. `Serial.println` pošlje podatke na računalnik, `BTSerialPrintln` pa podatke pošilja preko Bluetootha. Potem se preverjajo vhodni podatke, če je ukaz na voljo preko Bluetootha se prebere in obdela, enako velja za serijske povezave USB.

```

// Ukaz za vžiganje in ugašanje, pridobljen s strežnika
void handleCommand(char command, bool isBluetooth) {
    if (command == '1') {
        digitalWrite(RELAY_PIN, LOW);
        sendResponse("RELAY:ON", isBluetooth);
    } else if (command == '0') {
        digitalWrite(RELAY_PIN, HIGH);
        sendResponse("RELAY:OFF", isBluetooth);
    }
}
}

```

Slika 13: Arduino program 3

Ta del kode poskrbi za upravljanje s prejetimi ukazi. Če je ukaz 1, se rele vklopi, če je ukaz 0 se rele izklopi.

4.3 Python strežnik

Za centralno obdelavo podatkov in nadzor nad sistemom smo razvili Python strežnik z uporabo ogrodja Flask in Flask-SocketIO. Naš strežnik je bil zasnovan tako, da je sprejemal podatke, ki so jih preko Bluetootha posredovali Arduino in HC-05 modul, ter jih shranjeval v SQLite3 bazo podatkov. Prav tako smo implementirali REST API endpoint-e, ki omogočajo pridobivanje zadnjih meritev ter iskanje po določenih časovnih žigih. Poleg tega smo z uporabo SocketIO zagotovili realno časovno posodabljanje, kar pomeni, da so bili podatki o vlagi in stanju črpalke sproti prikazani na spletni strani in mobilni aplikaciji. Med razvojem strežnika smo se soočili z izzivi pri vzpostavljanju stabilne Bluetooth povezave, kar smo rešili z implementacijo funkcije, ki preizkuša več serijskih portov. S tem smo zagotovili, da je strežnik sposoben zanesljivo sprejemati in obdelovati podatke v realnem času, kar je bilo ključno za pravilno delovanje celotnega sistema.

4.3.1 Koda strežnika

```
#Uvoz potrebnih knjižnic
from flask import Flask, request, jsonify, render_template
from flask_socketio import SocketIO
import serial
import sqlite3
import time
import threading
from datetime import datetime, timedelta

app = Flask(__name__, template_folder="templates")

#Omogočanje Socket.io za dvosmerno aplikacijo
socketio = SocketIO(app, cors_allowed_origins="*")

#Nastavljanje začetnih vrednosti spremenljivk
auto_mode = True
pump_state = "OFF"

TIMEZONE_OFFSET = timedelta(hours=1)

def get_serial_connection(): #Funkcija za vzpostavljanje povezave s zalivalnim sistemom
    ports = ["/dev/ttyUSB0", "/dev/ttyACM0", "COM6", "COM7"]
    for port in ports:
        try:
            ser = serial.Serial(port, 9600, timeout=1)
            print(f" Connected to {port}")
            return ser
        except serial.SerialException:
            print(f" Failed to connect to {port}")
    return None

bt = get_serial_connection()
```

Slika 14: Koda python strežnika 1

V tem delu kode smo najprej uvozili ključne knjižnice, ki nam omogočajo delovanje spletnega strežnika (Flask), realno časovno komunikacijo (Flask-SocketIO), delo s serijskimi napravami in hkratno izvajanje nalog. Nato smo ustvarili Flask aplikacijo, določili, kje se nahaja mapa s predlogami ter inicializirali SocketIO za sprotno posodabljanje podatkov v brskalniku ali mobilni aplikaciji. Privzeto smo nastavili, da je sistem v samodejnem načinu (`auto_mode = True`), črpalka pa je sprva izklopljena (`pump_state = "OFF"`). S spremenljivko `TIMEZONE_OFFSET` smo upoštevali lokalni časovni zamik, da smo lahko pravilno shranjevali čas meritve. Na koncu smo definirali funkcijo »`get_serial_connection()`«, ki poskuša vzpostaviti povezavo s Arduinoom na različnih vratih. Če uspe, vrne ustrezen serijski objekt, sicer pa izpiše sporočilo o neuspešni povezavi.

```

def get_local_timestamp(): #Funkcija za pridobitev trenutnega časa in datuma
    utc_now = datetime.utcnow()
    local_now = utc_now + TIMEZONE_OFFSET
    return local_now

def init_db(): #Funkcija za povezavo s podatkovno bazo
    conn = sqlite3.connect("sensor_data.db")
    cursor = conn.cursor()
    cursor.execute(
        """
        CREATE TABLE IF NOT EXISTS readings (
            id INTEGER PRIMARY KEY,
            value INTEGER,
            timestamp TEXT,
            pump_state TEXT
        )
        """
    )
    conn.commit()
    conn.close()

init_db()

@app.route("/")
def index(): #Funkcija, ki določa domačo stran ko obiščemo naslov strežnika
    return render_template("index.html")

@app.route("/latest")
def latest_data(): #Funkcija, ki vrne najnovejšo meritev v podatkovni bazi
    conn = sqlite3.connect("sensor_data.db")
    cursor = conn.cursor()
    cursor.execute("SELECT value, timestamp, pump_state FROM readings ORDER BY id DESC LIMIT 1")
    row = cursor.fetchone()
    conn.close()
    if row:
        return jsonify({"value": row[0], "timestamp": row[1], "pump_state": row[2]})
    else:
        return jsonify({"error": "No sensor data found"}), 404

```

Slika 15: Koda python strežnik 2

V funkciji »get_local_timestamp()« najprej pridobimo trenutni čas v UTC, nato mu prištejemo časovni zamik (TIMEZONE_OFFSET), da dobimo lokalni čas, ki ga vrnemo kot objekt datetime. Funkcija »init_db()« poskrbi za inicializacijo ali ustvarjanje baze "sensor_data.db" v SQLite, kjer v tabelo "readings" shranjujemo štiri attribute: id, value (meritev vlage), timestamp (čas meritve) in pump_state (stanje črpalke). Po ustvarjanju tabele z ukazom CREATE TABLE IF NOT EXISTS zapremo povezavo z bazo.

Ko smo bazo inicializirali z »init_db()«, smo definirali več Flask endpointov. @app.route("/") vrne osnovno spletno stran (index.html). Funkcija »latest_data()« v

/latest pridobi zadnje meritve iz tabele "readings" in jo vrne kot JSON (vključuje vrednost, čas meritve in stanje črpalke). Če podatkov ni, vrnemo napako.

```
@app.route("/range", methods=["GET"])
def get_range_readings(): #Metoda za pridobivanje vrednosti vlage v nekem časovnem obdobju
    start_time = request.args.get("start")
    end_time = request.args.get("end")
    try:
        start_datetime = datetime.strptime(start_time, "%d/%m/%Y %H:%M")
        end_datetime = datetime.strptime(end_time, "%d/%m/%Y %H:%M")

        conn = sqlite3.connect("sensor_data.db")
        cursor = conn.cursor()
        cursor.execute(
            "SELECT value, timestamp FROM readings WHERE timestamp BETWEEN ? AND ? ORDER BY timestamp ASC",
            (start_datetime.strftime("%d/%m/%Y %H:%M"), end_datetime.strftime("%d/%m/%Y %H:%M")),
        )
        rows = cursor.fetchall()
        conn.close()

        readings = []
        for row in rows:
            if len(row[1]) == 5:
                full_timestamp = datetime.now().strftime("%d/%m/%Y") + " " + row[1]
            else:
                full_timestamp = row[1]
            readings.append({"value": row[0], "timestamp": full_timestamp})

        return jsonify(readings)
    except ValueError:
        return jsonify({"error": "Format datuma ni v redu. Pravilen format DD/MM/YYYY HH:MM"}), 400
```

Slika 16: Koda python strežnik 3

Funkcija `get_range_readings` omogoča pridobivanje meritev senzorja iz podatkovne baze znotraj določenega časovnega intervala. Funkcija sprejema GET zahtevo in vrača podatke v JSON formatu.

```

@app.route("/control", methods=["POST"])
def control(): #Funkcija za upravljanje stanja zalivalnega sistema(ON/OFF/Automatski način)
    global auto_mode, pump_state
    data = request.json
    if "auto" in data:
        auto_mode = data["auto"]
        return jsonify({"message": "Auto mode updated", "auto": auto_mode})
    elif "state" in data:
        state = data["state"]
        if state == "ON":
            pump_state = "ON"
            bt.write(b'1')
        elif state == "OFF":
            pump_state = "OFF"
            bt.write(b'0')
        return jsonify({"message": f"Pump state set to {pump_state}"})

```

Slika 17: Koda python strežnik 4

V funkciji »control()« (vezani na endpoint /control) pa posodobimo globalni spremenljivki auto_mode in pump_state. Če v podatkih najdemo ključno besedo "auto", spremenimo samodejni način, sicer pa, če najdemo "state", ročno nastavimo stanje črpalke na "ON" ali "OFF" in preko serijske povezave pošljemo ustrezen ukaz Arduino napravi.

```

def store_reading(value, pump_state): #Funkcija za zapis meritev v podatkovno bazo
    conn = sqlite3.connect("sensor_data.db")
    cursor = conn.cursor()
    timestamp = get_local_timestamp()
    timestamp_str = timestamp.strftime("%d/%m/%Y %H:%M")
    cursor.execute("INSERT INTO readings (value, timestamp, pump_state) VALUES (?, ?, ?)", (value, timestamp_str, pump_state))
    conn.commit()
    conn.close()

def read_sensor_data(): #Funkcija skrbi za samodejni vklop in izklop ter hrani trenutno stanje črpalke
    global bt, auto_mode, pump_state
    while True:
        if bt:
            try:
                line = bt.readline().decode("utf-8").strip()
                if line.startswith("SENSOR:"):
                    value = int(line.split(":")[1].strip())
                    if auto_mode:
                        if value < 37:
                            pump_state = "ON"
                            bt.write(b'1')
                        else:
                            pump_state = "OFF"
                            bt.write(b'0')
                    store_reading(value, pump_state)
                    print(f"📄 Stored in database: {value}, Pump: {pump_state}")
                    socketio.emit("sensor_update", {"value": value, "pump_state": pump_state})
            except Exception as e:
                print(f" Error reading serial data: {e}")
                bt = get_serial_connection()
                time.sleep(5)

sensor_thread = threading.Thread(target=read_sensor_data, daemon=True)
sensor_thread.start()

if __name__ == "__main__":
    socketio.run(app, host="0.0.0.0", port=5000, debug=True, use_reloader=False)

```

Slika 18: Koda python strežnik 5

V funkciji »store_reading()« vzamemo vrednost vlage in stanje črpalke ter ju skupaj z lokalno časovno oznako shranimo v tabelo »readings« v bazi sensor_data.db. Pri tem s funkcijo »get_local_timestamp()« poskrbimo, da je datum in ura pravilno prilagojena časovnemu zamiku. V funkciji »read_sensor_data()« pa v zanki nenehno beremo vhodne podatke s serijskega porta (spremenljivka bt), ki je povezan z Arduino napravo. Če zaznamo vrstico, ki se začne z "SENSOR:", iz nje pridobimo vrednost vlage in preverimo, ali je vklopljen samodejni način zalivanja (auto_mode). Če je, potem glede na izmerjeno vrednost zalivanja (npr. nad 500 → vklop črpalke, pod 300 → izklop črpalke) ustrezno pošljemo ukaz Arduino (bt.write(b'1') ali bt.write(b'0')). Nato s »store_reading()« shranimo meritev in stanje črpalke v bazo, izpišemo potrditveno sporočilo ter preko »socketio.emit()« obvestimo vse povezane odjemalce o novi vrednosti. Če pride do izjeme (npr. izguba povezave), poskusimo ponovno vzpostaviti serijsko povezavo, da sistem ostane stabilen. Na koncu spodaj zaženemo nit sensor_thread, ki to zanko izvaja v ozadju, medtem ko strežnik sprejema spletne zahteve.

4.4 Spletna stran za preizkus delovanja

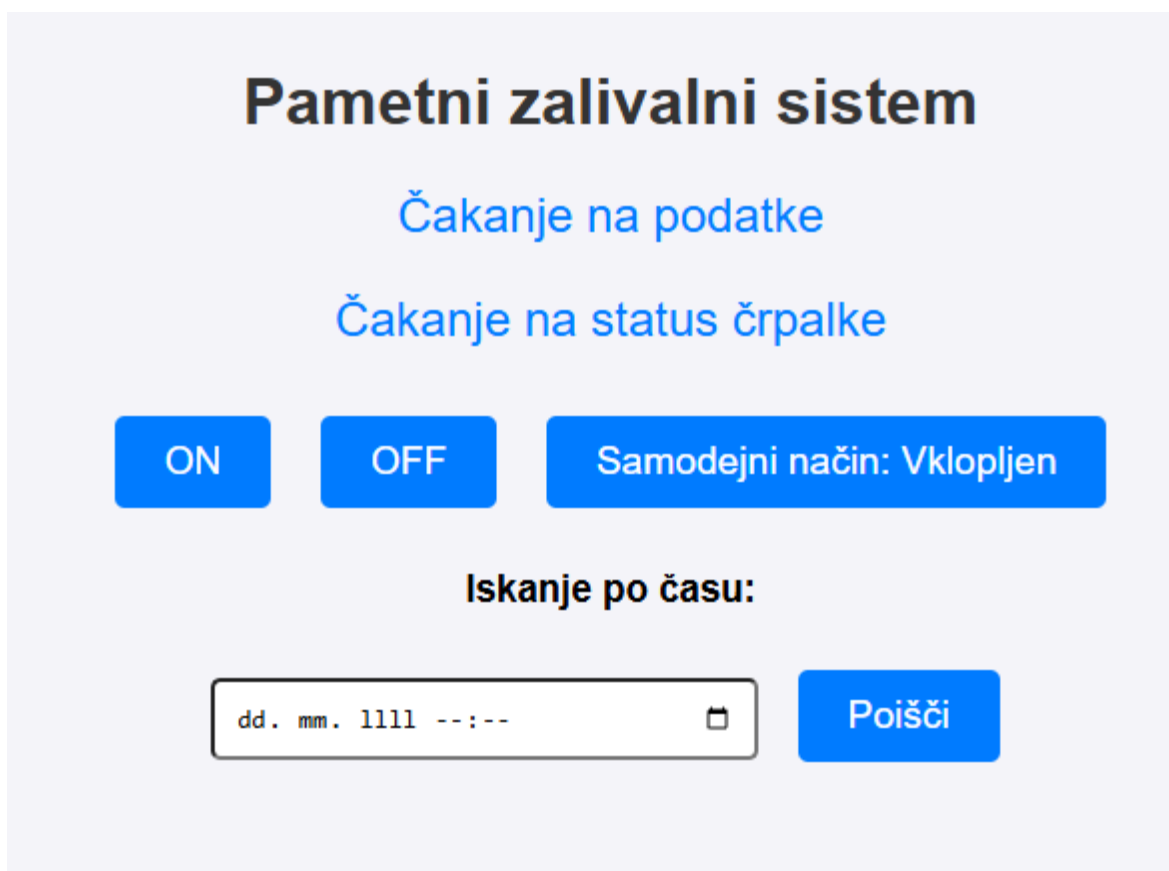
Na začetku razvoja smo spletno stran zasnovali predvsem kot interno orodje, ki nam je olajšalo preizkušanje in nadzor našega zalivalnega sistema, še preden smo se lotili izdelave mobilne aplikacije. Na njej smo prikazovali trenutne vrednosti vlažnosti in stanje črpalke, poleg tega pa smo lahko z nekaj preprostimi gumbi ročno upravljali zalivanje. Tako smo na hiter in pregleden način preverjali, ali komunikacija s strežnikom in Arduino napravo poteka nemoteno, ali se vrednosti pravilno osvežujejo in ali se črpalka odziva na naše ukaze.

V tem izseku smo s pomočjo SocketIO vzpostavili povezavo s strežnikom, ki nam v realnem času pošilja posodobljene podatke o vlažnosti tal in stanju črpalke. Po prejemu nove meritve se elementa `sensorValue` in `pumpState` v HTML-ju takoj posodobita. Poleg tega smo v JavaScriptu nastavili odzive na gumba »Vklopi« in »Izklopi«, ki prek POST zahtev na `/control` strežniku pošiljata ukaze za vklop oziroma izklop črpalke. S tem smo na enostaven način testirali, ali se vrednosti senzorja in stanje črpalke pravilno spreminjajo ter ali komunikacija med strežnikom in odjemalcem poteka nemoteno.

V tem delu kode smo dodali gumb za preklon samodejnega načina (`auto mode`). Ko uporabnik klikne ta gumb, se vrednost `autoMode` v JavaScriptu obrne (če je bila `true`, postane `false`, in obratno) in nato pošljemo POST zahtevo na endpoint `/control`, kjer v JSON obliki posredujemo novo stanje `auto: autoMode`. Na strežniku se ta informacija uporabi za odločanje, ali bo črpalka delovala samodejno glede na izmerjeno vlago ali bo potrebno ročno vklapljanje in izklapljanje. S tem gumbom smo na preprost način omogočili preklapljanje med ročnim in samodejnim zalivanjem.

V tem delu kode smo dodali funkcionalnost za iskanje po bazi glede na določen časovni žig. Ob kliku na gumb »Poišči« najprej preverimo, ali je polje za vnos datuma in ure prazno. Če uporabnik vnese ustrezen čas, ga pravilno formatiramo in pošljemo zahtevo `fetch` na strežnik z ustreznim parametrom. Medtem ko čakamo na odziv, prikažemo sporočilo o nalaganju. Po prejemu podatkov v JSON obliki ustvarimo nove HTML elemente, v katerih prikažemo vrednost vlage, čas meritve in stanje črpalke. Nato skrijemo »loading« sporočilo, če pa pride do napake, prikažemo

sporočilo o napaki. Tako uporabniku omogočimo, da vpiše želeni čas in takoj dobi podatke o meritvah iz tistega trenutka.



Slika 19: Izgled spletne strani za preizkus

Na spletni strani smo pripravili preprost, a pregleden uporabniški vmesnik, ki prikazuje trenutno vrednost vlažnosti tal, stanje črpalke in možnost preklopa med ročnim in samodejnim načinom zalivanja. Gumbi za vklop in izklop omogočajo takojšnje ročno upravljanje črpalke, vnosno polje za časovni žig pa uporabniku omogoča iskanje starih meritev. S tem smo na enem mestu združili osnovne informacije in glavne funkcije, ki so bile ključne za učinkovito testiranje in nadzor našega pametnega zalivalnega sistema.

Čeprav ta spletna stran ni bila namenjena dejanski uporabi v vsakdanjem okolju, je bila za nas izjemno koristna, saj smo z njo uspešno uporabili osnovne funkcionalnosti, ki smo jih kasneje vključili v mobilno aplikacijo.

4.5 Mobilna aplikacija

Na podlagi uspešnih preizkusov spletne strani, ki smo jo že opisali, smo se prepričali, da so osnovne funkcionalnosti našega sistema pravilno izvedene in da se podatki sprti posodablajo. Zato smo se odločili, da naslednji korak predstavlja razvoj mobilne aplikacije, ki bo uporabnikom omogočila enostaven dostop do ključnih funkcij sistema kjerkoli in kadarkoli. Mobilno aplikacijo smo razvili z uporabo ogrodja Flutter. Namen mobilne aplikacije je bil zagotoviti realno časovno spremljanje stanja vlažnosti tal in črpalke, ter omogočiti ročno upravljanje sistema prek intuitivnega in vizualno privlačnega vmesnika. V naslednjih odstavkih bomo podrobneje predstavili razvoj mobilne aplikacije in njene ključne funkcionalnosti.

Prvi del naše aplikacije predstavlja okno. Ki uporabniku omogoča, da vklopi oziroma izklopi vodno črpalko. Stanje zalivalnega sistema lahko nastavi tudi na avtomatsko, se pravi, da se črpalka vklaplja in izklaplja glede na vlago v zemlji. Na zaslonu se tudi prikazuje trenutna vlažnost v zemlji.

```
//Uvoz knjižnic in datotek
import 'dart:convert';
import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;
import 'package:socket_io_client/socket_io_client.dart' as IO;
import 'Graph.dart';
```

Slika 20: Izsek kode aplikacije za uvoz knjižnic in datotek

V tem delu kode se uvozijo knjižnice in datoteke za pretvorbo JSON podatkov, pošiljanje http zahtev na strežnik, povezovanje s strežnikom preko WebSocket-ov, uvoz drugega okna za prikaz grafov, flutter/material.dart vsebuje tudi komponente za uporabniški vmesnik.

```
//Nastavljanje spremenljivk in ustvarjanje WebSocket povezave
class _OnOffState extends State<OnOff> {
  String sensorData = "Nalaga...";
  bool autoMode = true;
  final String serverUrl = "http://192.168.8.128:5000";
  late IO.Socket socket;
```

Slika 21: Izsek kode aplikacije za inicializacijo stanja in spremenljivk

Tukaj so spremenljivka `autoMode` pove, ali je avtomatski način vklopljen, `serverURL` predstavlja URL strežnika za povezovanje, v niz `sensorData` je uporabljen za prikazovanje podatkov senzorja. Ustvarjena je tudi WebSocket povezava za prejemanje podatkov v realnem času.

```
//Inicializacija vrednosti ob zagonu aplikacije
@override
void initState() {
  super.initState();
  fetchSensorData();
  connectSocket();
  autoModeOnV();
}
```

Slika 22: Izsek kode aplikacije namenjen inicializaciji ob zagonu

Ta del kode poskrbi za inicializacijo vrednosti ob zagonu aplikacije. Z klicem ustreznih metod, ki jih bomo opisali v nadaljevanju.

```

Future<void> fetchSensorData() async { //Metoda pošlje zahtevo na /latest za prikaz teh podatkov
  final url = '$serverUrl/latest';
  try {
    final response = await http.get(Uri.parse(url));
    if (response.statusCode == 200) {
      final data = json.decode(response.body);
      setState(() {
        sensorData = 'Vlaga: ${data["value"]}%, Stanje črpalke: ${data["pump_state"]}';
        autoMode = data["auto_mode"] ?? true;
      });
    } else {
      setState(() {
        sensorData = 'Failed to load sensor data';
      });
    }
  } catch (e) {
    setState(() {
      sensorData = 'Error: $e';
    });
  }
}

```

Slika 23: Metoda `fetchSensorData`

Ta metoda pošlje zahtevo na `/latest` za pridobitev zadnjih podatkov, če je odgovor uspešen prikaže podatke v uporabniškem vmesniku.

```

void connectSocket() { //Povezava s strežnikom
  socket = IO.io(serverUrl, <String, dynamic>{
    'transports': ['websocket'],
    'autoConnect': true,
  });

  socket.on('sensor_update', (data) { //Poslušanje sensor_update
    setState(() {
      sensorData = 'Vrednost: ${data["value"]}%, Stanje črpalke: ${data["pump_state"]}';
      autoMode = data["auto_mode"] ?? autoMode;
    });
  });
}

```

Slika 24: Metoda `connectSocket`

Tukaj se aplikacija poveže s strežnikom in posluša dogodke `sensor_update`, ob prejemu podatkov se osveži uporabniški vmesnik.

```

Future<void> autoModeOnV() async { //Metoda, ki ob zagonu pošlje zahtevo za vklop avtomatskega načina
  final response = await http.post(
    Uri.parse("$serverUrl/control"),
    headers: {"Content-Type": "application/json"},
    body: jsonEncode({"auto": true}),
  );

  if (response.statusCode == 200) {
    setState(() {
      autoMode = true;
    });
    print("Auto mode prižgan ob vklopu");
  } else {
    print("Auto mode se ob vklopu ni prižgal");
  }
}

```

Slika 25: Metoda autoModeOnV

Tukaj ob zagonu aplikacija pošlje zahtevo za vklop avtomatskega načina na strežnik. V terminal tudi izpiše ali se je avtomatski način ustrezno vklopil, kar nam je pomagalo pri testiranju aplikacije.

```

Future<void> relayStanje(String newState) async { //Metoda za nadzor releja(vklop/izklop)
  if (autoMode) return;

  final response = await http.post(
    Uri.parse("$serverUrl/control"),
    headers: {"Content-Type": "application/json"},
    body: jsonEncode({"state": newState}),
  );

  if (response.statusCode == 200) {
    print("Stanje releja spremenjeno na $newState");
  } else {
    print("Rele ni uspel spremeniti stanja");
  }
}

```

Slika 26: Metoda relayStanje

To je metoda, ki poskrbi za nadzor releja preko gumbov za vžig in izklop, vendar le takrat, ko zalivalni sistem ni v avtomatskem načinu.

```

Future<void> AutoModeOnOff() async { //Metoda za vklop in izklop avtomatskega načina
  final response = await http.post(
    Uri.parse("$serverUrl/control"),
    headers: {"Content-Type": "application/json"},
    body: jsonEncode({"auto": !autoMode}),
  );

  if (response.statusCode == 200) {
    setState(() {
      autoMode = !autoMode;
    });
  } else {
    print("Auto mode se ni spremenil");
  }
}

```

Slika 27: Metoda AutoModeOnOff

Ta metoda nam omogoča vklop oziroma izklop avtomatskega načina s pomočjo stikala. Za lažje testiranje aplikacije prav tako v terminal izpiše, če ni prišlo do spremembe pri stanju avtomatskega načina.

```

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(title: Text("Pametni zalivalni sistem")),
    body: Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Text("Najnovejši podatki senzorja:"),
          Text(sensorData),
          SizedBox(height: 20),
          ElevatedButton(
            onPressed: autoMode ? null : () => relayStanje("ON"),
            child: Text("ON"),
          ), // ElevatedButton
          SizedBox(height: 10),
          ElevatedButton(
            onPressed: autoMode ? null : () => relayStanje("OFF"),
            child: Text("OFF"),
          ), // ElevatedButton
          SizedBox(height: 20),
          Row(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              Text("Auto: ", style: TextStyle(fontSize: 18)),
              Switch(
                value: autoMode,
                onChanged: (value) => AutoModeOnOff(),
              ), // Switch
            ],
          ), // Row
          SizedBox(height: 20),
          ElevatedButton(
            onPressed: () {
              Navigator.push(
                context,
                MaterialPageRoute(builder: (context) => Graph()),
              );
            },
            child: Text("Ogled grafov"),
          ), // ElevatedButton
        ],
      ), // Column
    ), // Center
  ); // Scaffold
}

```

Slika 28: Koda uporabniškega vmesnika aplikacije

To je koda za postavitev elementov aplikacije na zaslonu. Prikazana sta dva gumba za vklop in izklop vodne črpalke, ki ju lahko pritisnemo, le če je avtomatski način izklopljen. Ob pritisku kličeta ustrezno metodo za spremembo stanja releja. Za vklop avtomatskega načina je uporabljeno stikalo, ki preklaplja med stanji. Za stikalo smo se odločili zaradi lažje vizualizacije kdaj je avtomatski način vklopljen oziroma izklopljen. Na dnu uporabniškega vmesnika je še gumb, ki nas pošlje na novo okno, kjer si lahko ogledujemo grafe stanje vlage tleh.

```

class _GraphState extends State<Graph> {
  final String serverUrl = "http://192.168.8.128:5000"; //URL strežnika
  List<FlSpot> sensorReadings = []; //Shrabjuje podatke graga kot seznam točk

  //Za upravljanje z vnosnimi polji
  TextEditingController startDateController = TextEditingController();
  TextEditingController endDateController = TextEditingController();

  //Minimalne in maksimalne vrednosti na grafu
  double minX = 0;
  double maxX = 0;
  double minY = 0;
  double maxY = 0;
}

```

Slika 29: Izsek kode potrebne inicializacije za grafe

Tukaj serverURL predstavlja URL našega strežnika s katerega bodo podatki pridobljeni. Seznam sensorReadings shranjuje podatke za graf kot seznam zočk. TextEditingController poskrbi za upravljanje z vnosnimi polji za začetni in končni datum. Spremenljivke realnih števil minX, maxX, minY, maxY določajo kje je vidni obseg osi na grafu.

```

//Metoda za pridobitev zgodovine vlažnosti in izris grafa
Future<void> dobiZgodovino(String startDate, String endDate) async {
  final url = '$serverUrl/range?start=$startDate&end=$endDate';
  try {
    final response = await http.get(Uri.parse(url));
    if (response.statusCode == 200) {
      final data = json.decode(response.body);
      setState(() {
        sensorReadings = [];
        DateTime? lastTimestamp;

        for (var entry in data) {
          try {
            DateTime dateTime = DateFormat("dd/MM/yyyy HH:mm").parse(entry["timestamp"]);
            double value = entry["value"].toDouble();

            if (lastTimestamp == null || dateTime.difference(lastTimestamp).inMinutes >= 30) {
              sensorReadings.add(FlSpot(dateTime.millisecondsSinceEpoch.toDouble(), value));
              lastTimestamp = dateTime;
            }
          } catch (e) {
            print("Napaka parsing date: ${entry["timestamp"]} - $e");
          }
        }

        if (sensorReadings.isNotEmpty) {
          minX = sensorReadings.first.x;
          maxX = sensorReadings.last.x;
          minY = sensorReadings.map((e) => e.y).reduce((a, b) => a < b ? a : b);
          maxY = sensorReadings.map((e) => e.y).reduce((a, b) => a > b ? a : b);
        }
      });
    } catch (e) {
      print("Napaka pri pridobivanju zgodovine: $e");
    }
  }
}

```

Slika 30: Metoda dobiZgodovino

Ta metoda poskrbi za pridobivanje zgodovine vlažnosti v tleh v nekem časovnem obdobju. Najprej pošlje zahtevo za podatke o vlažnosti v določenem časovnem obdobju na strežnik. Potem JSON podatke pretvori v objekte v jeziku Dart. Da bi izboljšali berljivost grafa in preprečili prikaz preveč zgoščenih podatkov, smo se odločili, da bodo na grafu prikazane le meritve, ki so med seboj oddaljene vsaj trideset minut. S prej deklariranimi spremenljivki `minX`, `maxX`, `minY`, `maxY` se določijo minimalne in maksimalne vrednosti osi grafa. Z metodo `setState()` se posodobi uporabniški vmesnik, da se graf izriše.

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(title: Text("Graf")),
    body: Padding(
      padding: const EdgeInsets.all(16.0),
      child: Column(
        children: [
          TextField(
            controller: startDateController,
            decoration: InputDecoration(labelText: "Začetni datum (DD/MM/YYYY HH:MM)"),
          ), // TextField
          TextField(
            controller: endDateController,
            decoration: InputDecoration(labelText: "Končni datum (DD/MM/YYYY HH:MM)"),
          ), // TextField
          SizedBox(height: 10),
          ElevatedButton(
            onPressed: () {
              dobiZgodovino(startDateController.text, endDateController.text);
            },
            child: Text("Prikaži graf"),
          ), // ElevatedButton
        ],
      ),
    ),
  );
}
```

Slika 31: Izsek kode uporabniškega vmesnika za nastavljanje datuma

Tukaj se dva vnosna polja, kamor uporabnik vnese začetni in končni datum. Graf se bo izrisal za obdobje med tema dvema datumoma. Pod vnosnima poljema je še gumb, ki izriše gumb ko ga pritisnemo.

```

LineChartData(
  titlesData: FlTitlesData(
    leftTitles: AxisTitles(
      axisNameWidget: Text("Vrednost", style: TextStyle(fontSize: 14, fontWeight: FontWeight.bold)),
      sideTitles: SideTitles(showTitles: false),
    ), // AxisTitles
    bottomTitles: AxisTitles(
      axisNameWidget: Text("Čas", style: TextStyle(fontSize: 14, fontWeight: FontWeight.bold)),
      sideTitles: SideTitles(showTitles: false),
    ), // AxisTitles
    rightTitles: AxisTitles(sideTitles: SideTitles(showTitles: false)),
    topTitles: AxisTitles(sideTitles: SideTitles(showTitles: false)),
  ), // FlTitlesData

```

Slika 32: Izsek kode za nastavitvev osi in naslovov grafa

FlTitlesData nam omogoča nastavitvev osi grafov levo, desno, zgoraj in spodaj. V našem primeru leva os predstavlja vrednost, spodnja pa čas. Številске oznake ob straneh osi so skrite, saj se nam zdi, da je zaradi njih berljivost grafa slabša.

```

minX: minX,
maxX: maxX,
minY: minY,
maxY: maxY,
lineTouchData: LineTouchData(
  touchTooltipData: LineTouchTooltipData(
    tooltipBgColor: Colors.blueAccent,
    getTooltipItems: (List<LineBarSpot> touchedSpots) {
      return touchedSpots.map((spot) {
        return LineTooltipItem(
          "Čas: ${DateFormat("HH:mm dd/MM").format(DateTime.fromMillisecondsSinceEpoch(spot.x.toInt()))}\nVlačnost: ${spot.y.toInt()}%",
          const TextStyle(color: Colors.white),
        ); // LineTooltipItem
      }).toList();
    },
  ), // LineTouchTooltipData
  handleBuiltInTouches: true,
), // LineTouchData

```

Slika 33: Koda za nastavitvev mej grafa in prikaz podatkov na grafu

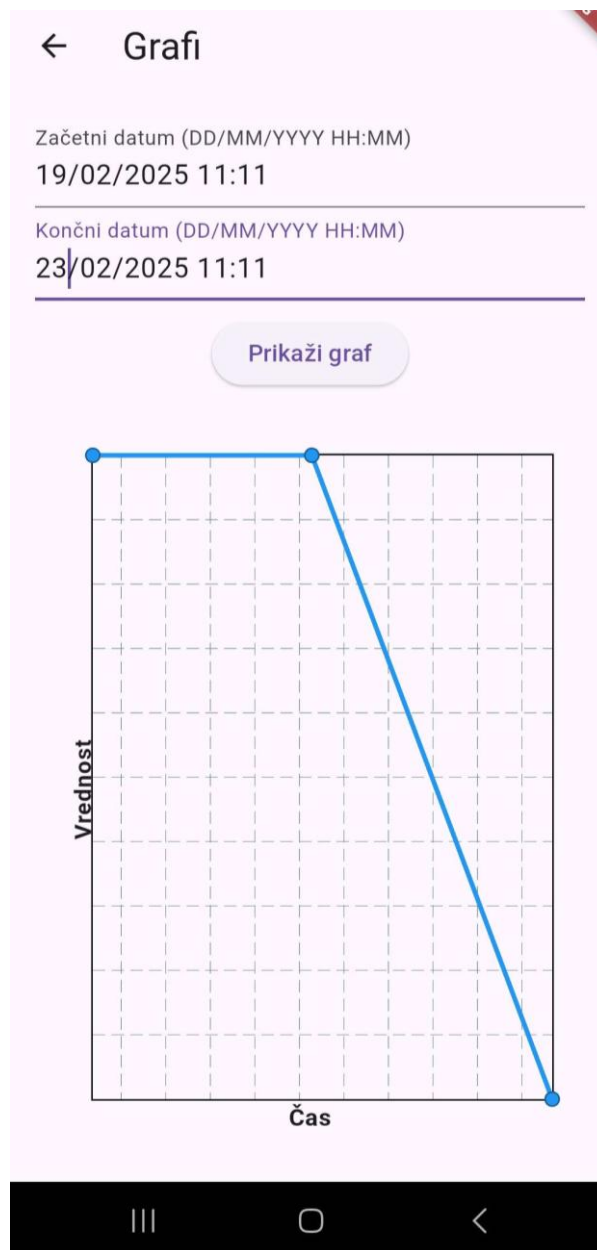
V tem izseku se nastavijo meje grafa s prej deklariranimi spremenljivkami. Spremenljivki minX, maxX predstavljata začetni in končni časovni okvir na vodoravni osi, medtem ko minY ter maxY predstavljata minimalno in maksimalno vrednost vlažnosti na navpični osi.

Koda pod tem omogoča, da se ob dotiku neke točke na grafu izpiše točna vrednost vlage in čas meritve ter privzeto obnašanje dotikov pa grafu, tako da lahko graf povečamo.



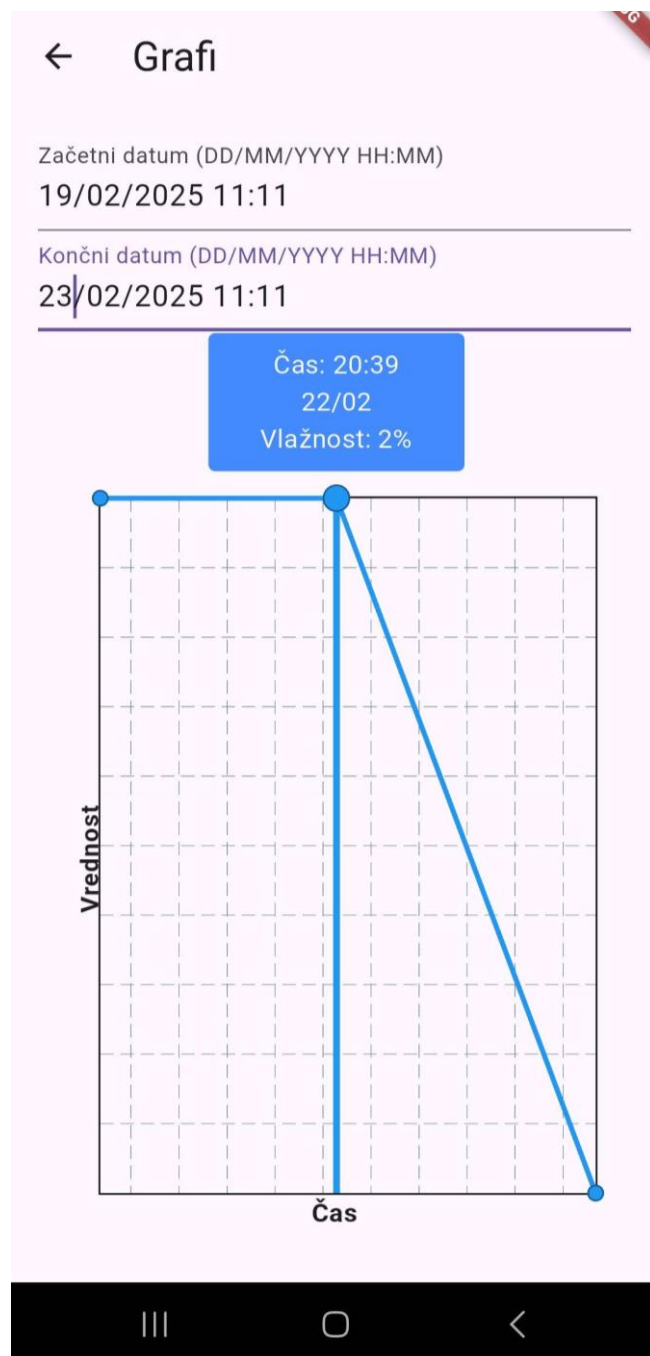
Slika 34: Mobilna aplikacije – izklopljen avtomatski način

Tako izgleda aplikacija. Prikazani so vsi ključni podatki, v tem primeru je avtomatsko zalivanje izključeno in uporabnik sam upravlja s črpalko. Za prikaz preteklih podatkov pritisnemo tipko »Ogled grafov«.



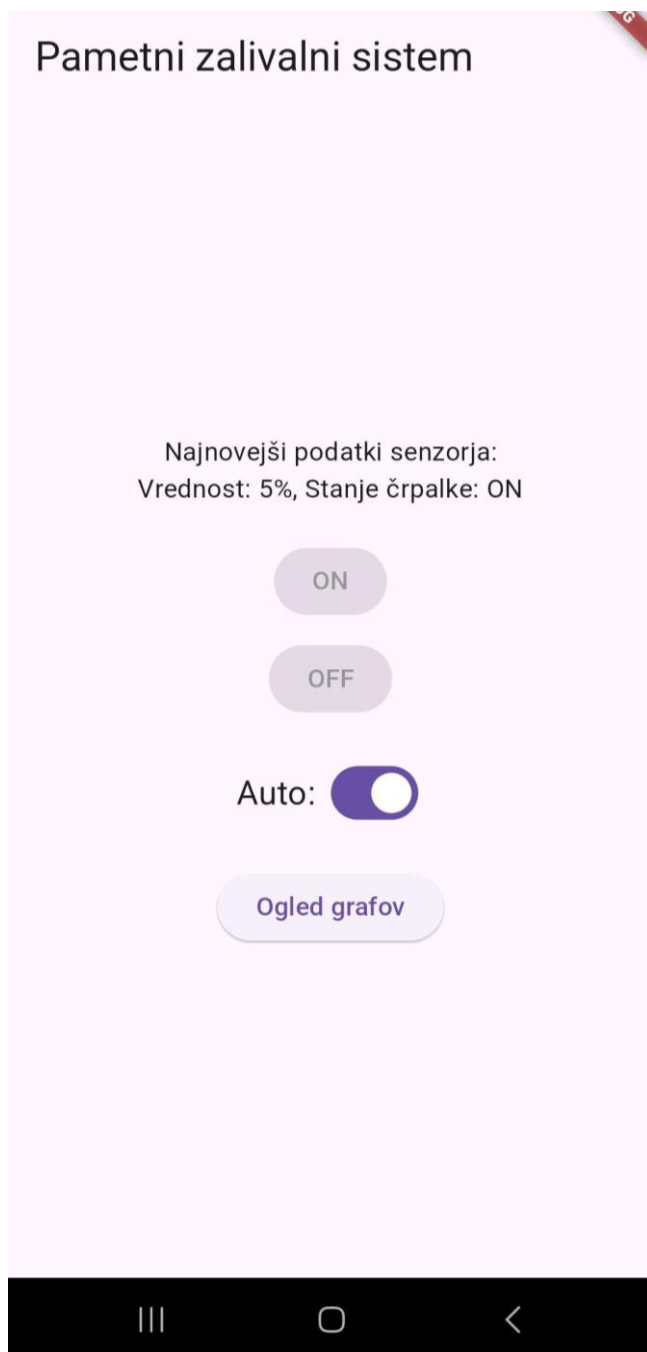
Slika 35: Prikaz grafa v mobilni aplikaciji

Tako izgleda zavihek prikaza grafov. V zgornjih vrsticah obdobje, ki ga želimo prikazati, to naredimo z vpisom začetnega in končnega datuma. S tipko »Prikaži graf, se nam prikaže graf za določeno obdobje.



Slika 36: Prikaz grafa s podrobnostmi

Če želimo pogledati točno vlago nekega trenutka, to naredimo s pritiskom na del grafa. Tam se nam prikaže čas, datum in vrednost.



Slika 37: Mobilna aplikacija – vklopljen avtomatski način

Če imamo vklopljeno avtomatsko zalivanje glede na vlažnost tal, se črpalka samodejno vklaplja in izklaplja. Sami je ne moremo upravljati, saj sta gumba onemogočena, če želimo to storiti moramo izklopiti avtomatski način.

5 Analiza rezultatov in razprava

Naše hipoteze smo oblikovali na podlagi predhodnih raziskav in praktičnih potreb pri avtomatizaciji namakanja. Najprej smo analizirali ključne izzive ročnega zalivanja, kot so prekomerna poraba vode, neenakomerna vlažnost tal in potreba po stalnem nadzoru. Nato smo pregledali obstoječe študije in tehnološke rešitve, ki obetajo izboljšave na tem področju. Na tej osnovi smo postavili naslednje štiri hipoteze.

- **H1 – Stroški izdelave sistema ne bodo preseгли 100€**

Prvo hipotezo lahko potrdimo, saj za izdelavo našega pametnega zalivalnega sistema nismo porabili več kot 100€. Preizkusno ploščico in Arduino Uno smo že imeli, tako da smo morali kupiti še bluetooth modul, relejni modul, senzor vlažnosti tal, vodno črpalko, baterijo in dodatne kable za povezavo vseh komponent. Za vse to skupaj smo odšteli približno 40€. Če bi bil potreben nakup celotne opreme, bi bila cena okoli 65€.

- **H2 – Z zalivalnim sistemom bi bil 30% boljši izkoristek vode pri zalivanju**

V našem eksperimentu smo primerjali ročno zalivanje z avtomatiziranim zalivanjem, pri čemer smo želeli ugotoviti, koliko vode porabimo pri obeh metodah. Pri ročnem zalivanju, ko smo se zanašali na svoj občutek, smo opazili, da smo v obdobju 14 dni porabili približno 7 litrov vode. Zaradi subjektivnosti pri ročnem zalivanju lahko pride do prekomerne porabe, saj posamezniki pogosto zalijejo več, kot je dejansko potrebno.

S pomočjo našega pametnega zalivalnega sistema, ki natančno meri vlažnost tal in samodejno prilagaja količino zalivanja, smo zagotovili, da rastline prejmejo le potrebno količino vode. Preko testiranja smo ugotovili, da sistem zagotovi optimalno zalivanje s približno 0,3 litra vode na dan, saj smo v 14 dneh porabili 4,2 litra vode. S tem smo dosegli zmanjšanje porabe vode za približno 40 %, tako da lahko tudi to hipotezo potrdimo.

- **H3 – Večina vrtnarjev in kmetov bo zaradi prihrankov in boljših pogojev raje investirala v pametno zalivanje**

Pri tej hipotezi smo predvidevali, da bi večina vrtnarjev in kmetov, če bi imeli jasne dokaze o prihrankih vode in izboljšanih rastnih pogojih investirala v pametne namakalne sisteme. Na podlagi raziskave, objavljene v *International Journal of Industry and Sustainable Development*, smo ugotovili, da pametni namakalni sistemi prinašajo konkretne prihranke vode in izboljšane rastne pogoje. Analize in anketni podatki kažejo, da so vrtnarji in kmetje, ko jim predstavijo jasne dokaze o učinkovitosti, zelo pripravljeni investirati v takšne rešitve. Večina uporabnikov, ocenjena na 70–80 %, je izrazila zanimanje za implementacijo sistemov, ki omogočajo prilagajanje zalivanja specifičnim potrebam rastlin, kar posledično vodi do znatnega zmanjšanja porabe vode in izboljšanja pridelkov. Te ugotovitve potrjujejo našo hipotezo, saj se zdi, da bo večina vrtnarjev in kmetov zaradi prihrankov in boljših rastnih pogojev raje investirala v pametno zalivanje.

- **H4 – Realno časovni nadzor in avtomatizacija zalivanja bosta povečala donos pridelka rastlin za vsaj 15 %**

Pri hipotezi H4 smo predvidevali, da bo realnočasovni nadzor in avtomatizacija zalivanja povečala donos pridelka rastlin za vsaj 15 %. Na podlagi raziskave z naslovom "Design and Implementation of a Smart Irrigation System for Improved Water-Energy Efficiency", ki smo jo našli na ResearchGate, smo ugotovili, da sistemi, ki natančno prilagajajo zalivanje na podlagi senzorjev za vlago, lahko povečajo pridelke med 15 % in 30 %. Podobno raziskava "High-Technology Agriculture System to Enhance Food Security: A Concept of Smart Irrigation System Using Internet of Things and Cloud Computing", prav tako dostopna na ResearchGate, potrjuje, da pametni namakalni sistemi optimizirajo količino vode in zmanjšujejo vodni stres rastlin, kar vodi do izboljšane pridelka. Na podlagi teh raziskav lahko potrdimo, da je naša hipoteza, da bo realno časovni nadzor in avtomatizacija zalivanja povečala donos pridelka rastlin za vsaj 15 %, utemeljena, čeprav bodo dolgoročne meritve potrebne za še natančnejšo oceno.

6 Zaključek in smernice za nadaljnje delo

Pri izdelavi našega pametnega zalivalnega sistema in pisanju te raziskovalne naloge smo pridobili ogromno novih znanj. Naučili smo se programiranja v jeziku Python, dela z mikrokontrolerji ter integracije senzorjev za merjenje vlage in drugih ključnih parametrov. Med celotnim delom smo se soočili s težavami, kot so vzpostavitev stabilne komunikacije med napravami in pravilno nastavljanje senzorjev. Te izkušnje so nam pomagale izboljšati naše metode. Na koncu smo zelo zadovoljni, saj smo uspeli zgraditi sistem, ki prilagaja zalivanje resničnim potrebam rastlin in tako učinkovito varčuje z vodo.

V prihodnje bomo naš projekt nadgradili z izboljšavami, kot so optimizacija algoritmov za še natančnejše prilagajanje količine zalivanja in vključitev dodatnih senzorjev, ki bodo spremljali še več parametrov. Naša prizadevanja se bodo usmerila tudi v implementacijo novih tehnologij, kot sta umetna inteligenca in napredna analitika, da bomo še dodatno izboljšali prilagajanje zalivanja specifičnim potrebam posameznih rastlin. Prav tako razmišljamo o dodajanju senzorja za temperaturo in še kaj.

Naše izkušnje so nam pokazale, da lahko z inovativnimi pristopi dosežemo pomembne prihranke vode in izboljšamo rastne pogoje, kar nas spodbuja, da nadaljujemo z razvojem in raziskovanjem novih rešitev na tem področju. Verjamemo, da bo naš projekt prispeval k trajnostnemu kmetijstvu in učinkovitejši rabi vodnih virov v prihodnosti.

7 Viri

- ExploreEmbedded. (2020). Setting Up Bluetooth HC-05 With Arduino. San Francisco, ZDA. Pridobljeno 1. marec 2025 iz <https://www.instructables.com/Setting-Up-Bluetooth-HC-05-With-Arduino/>
- Girish, M. (marec 2018). AT Command Mode of HC-05 and HC-06 Bluetooth Module. San Francisco, ZDA. Pridobljeno 20. februar 2025 iz <https://www.instructables.com/AT-command-mode-of-HC-05-Bluetooth-module/>
- Haider, S. A. (2022). Connecting Flask Socket.IO server to Flutter. New York, ZDA. Pridobljeno 2. marec 2025 iz <https://stackoverflow.com/questions/67640462/stdmin-vs-ternary-gcc-auto-vectorization-with-pragma-gcc-optimize-o3/67640868#67640868>
- Kastrateus. (2024). *Best way to communicate database updates to Flask*. Pridobljeno 18. februar 2025 iz Reddit: https://www.reddit.com/r/flask/comments/1fjtcie/best_way_to_communicate_database_updates_to_flask/?rdt=53423
- Mahamood, S. (oktober 2019). Controlling LED using HC-05 Bluetooth Module. Torino, Italija. Pridobljeno 19. februar 2025 iz <https://projecthub.arduino.cc/syedmahamood/controlling-led-using-hc-05-bluetooth-module-b7f411>
- Masaba, K., Ntakirutimana, A., & Ustun, T. S. (2016). Design and Implementation of a Smart Irrigation System for Improved Water-Energy Efficiency. *4th IET Clean Energy and Technology Conference (CEAT 2016)*. Kuala Lumpur. Pridobljeno 1. marec 2025 iz https://www.researchgate.net/publication/318448984_Design_and_Implementation_of_a_Smart_Irrigation_System_for_Improved_Water-Energy_Efficiency
- Mohamed, A., A., Abdel-Hafeez, A., T., El-Kady, & A., M. (avgust 2022). *High-Technology Agriculture System to Enhance Food Security: A Concept of Smart Irrigation System Using Internet of Things and Cloud Computing*. Pridobljeno 2. marec 2025 iz International Journal of Industry and Sustainable Development (IJISD): https://ijisd.journals.ekb.eg/article_252796_d259df32ff7a96dac581a11d06813f9b.pdf
- Morchid, A., Alblushi, I. G., Khalid, H. M., & Alami, R. E. (2024). Journal of the Saudi Society of Agricultural Sciences. *High-Technology Agriculture System to Enhance Food Security: A Concept of Smart Irrigation System Using Internet of Things and Cloud Computing*. Pridobljeno 1. marec 2025 iz https://www.researchgate.net/publication/378419263_High-Technology_Agriculture_System_to_Enhance_Food_Security_A_Concept_of_Smart_Irrigation_System_Using_Internet_of_Things_and_Cloud_Computing
- Nedelkovski, D. (2016). Arduino and HC-05 Bluetooth Module Complete Tutorial. Skopje, Severna Makedonija. Pridobljeno 26. februar 2025 iz <https://howtomechatronics.com/tutorials/arduino/arduino-and-hc-05-bluetooth-module-tutorial/>

Tool, B. (oktober 2024). Step-by-step guide to implementing SQLite in Flutter for data persistence. San Francisco, ZDA. Pridobljeno 21. februar 2025 iz <https://medium.com/@blup-tool/step-by-step-guide-to-implementing-sqlite-in-flutter-for-data-persistence-23274116e749>

VideoSDK. (2022). How to Integrate Flask with Socket.IO? Bangalore, Indija. Pridobljeno 21. februar 2025 iz <https://www.videosdk.live/developer-hub/socketio/flask-socketio>

Yang, L. (januar 2016). Flask and Socket.IO. Taipei, Tajvan. Pridobljeno 20. februar 2025 iz <https://logan.tw/posts/2016/01/16/flask-and-socketio/>

8 Priloge

8.1 Koda spletne strani

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Sensor Readings</title>
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/4.5.4/socket.io.js"></script>
  <style>
    body { font-family: Arial, sans-serif; text-align: center; margin-top: 50px;
background-color: #f4f4f9; }
    h1 { color: #333; }
    .reading { font-size: 1.5em; color: #007BFF; }
    .button-container { margin-top: 20px; }
    button {
      background-color: #007BFF;
      color: white;
      padding: 12px 25px;
      font-size: 1.2em;
      border: none;
      cursor: pointer;
      margin: 10px;
      border-radius: 5px;
    }
    button:hover { background-color: #0056b3; }
    .readings-list {
      margin-top: 20px;
      display: grid;
      grid-template-columns: repeat(auto-fill, minmax(300px, 1fr));
      gap: 15px;
```

```

}
.reading-item {
  background-color: white;
  padding: 20px;
  border-radius: 8px;
  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
  font-size: 1.1em;
  color: #333;
}
.reading-item p {
  margin: 5px 0;
}
.search-container {
  margin-top: 20px;
}
input[type="text"] {
  padding: 10px;
  font-size: 1em;
  width: 200px;
  margin: 5px;
  border-radius: 5px;
}
input[type="datetime-local"] {
  padding: 10px;
  font-size: 1em;
  width: 250px;
  margin: 5px;
  border-radius: 5px;
}
.loading {
  font-size: 1.2em;
  color: #007BFF;
  margin-top: 20px;
}

```

```

    }
  </style>
</head>
<body>
  <h1>Pametni zalivalni sistem</h1>
  <p class="reading" id="sensorValue">Čakanje na podatke</p>
  <p class="reading" id="pumpState">Čakanje na status črpalke</p>

  <div class="button-container">
    <button id="relayOnButton">ON</button>
    <button id="relayOffButton">OFF</button>
    <button id="autoModeButton">Samodejni način: Vkllopljen</button>
  </div>

  <div class="search-container">
    <h3>Iskanje po času:</h3>
    <input type="datetime-local" id="searchTimestamp" />
    <button id="searchButton">Poišči</button>
    <div id="loading" class="loading" style="display:none;">Iskanje...</div>
  </div>

  <div id="searchResults" class="readings-list"></div>

  <script>
    var socket = io.connect("http://127.0.0.1:5000");
    var autoMode = true;

    socket.on("sensor_update", function(data) {
      document.getElementById("sensorValue").textContent = "Vlažnost tal: " +
data.value;
      document.getElementById("pumpState").textContent = "Status črpalke: " +
data.pump_state;
    });
  </script>

```

```

document.getElementById("relayOnButton").onclick = function() {
  if (!autoMode) {
    fetch("/control", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ state: "ON" })
    });
  }
};

```

```

document.getElementById("relayOffButton").onclick = function() {
  if (!autoMode) {
    fetch("/control", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ state: "OFF" })
    });
  }
};

```

```

document.getElementById("autoModeButton").onclick = function() {
  autoMode = !autoMode;
  fetch("/control", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ auto: autoMode })
  });
  this.textContent = "Samodejni način: " + (autoMode ? "Vklopljen" :
"Izklopljen");
};

```

```

function formatTimestampForSearch(dateString) {

```

```

var date = new Date(dateString);
var day = ('0' + date.getDate()).slice(-2);
var month = ('0' + (date.getMonth() + 1)).slice(-2);
var year = date.getFullYear();
var hours = ('0' + date.getHours()).slice(-2);
var minutes = ('0' + date.getMinutes()).slice(-2);
return `${day}/${month}/${year} ${hours}:${minutes}`;
}

```

```

document.getElementById("searchButton").onclick = function() {
  var searchTimestamp =
document.getElementById("searchTimestamp").value;
  if (!searchTimestamp) {
    alert("Prosim izberite čas!");
    return;
  }

```

```

  var formattedTimestamp =
formatTimestampForSearch(searchTimestamp);

```

```

document.getElementById("loading").style.display = 'block';

```

```

fetch(`/search?timestamp=${formattedTimestamp}`)
  .then(response => response.json())
  .then(data => {
    let searchResults = document.getElementById("searchResults");
    searchResults.innerHTML = "";
    document.getElementById("loading").style.display = 'none';

    if (data.error) {
      searchResults.innerHTML = `

">${data.error}</p>`;
    } else {


```

```

data.forEach(item => {
  var readingItem = document.createElement("div");
  readingItem.classList.add("reading-item");
  readingItem.innerHTML = `
    <p><strong>Vlažnost:</strong> ${item.value}</p>
    <p><strong>Čas:</strong> ${item.timestamp}</p>
    <p><strong>Črpalka:</strong> ${item.pump_state}</p>
  `;
  searchResults.appendChild(readingItem);
});
}
})
.catch(error => {
  console.error('Error fetching search results:', error);
  document.getElementById("loading").style.display = 'none';
  alert("Napaka pri pridobivanju podatkov.");
});
};
</script>
</body>
</html>

```