Kućni informacijski sustav baziran na Raspberry PI

Barlović, Vanja

MEĐIMURSKO VELEUČILIŠTE U ČAKOVCU
STRUČNI PRIJEDIPLOMSKI STUDIJ RAČUNARSTVO

BARLOVIĆ VANJA

KUĆNI INFORMACIJSKI SUSTAV BAZIRAN NA RASPBERRY PI

ZAVRŠNI RAD

ČAKOVEC, 2023.

MEĐIMURSKO VELEUČILIŠTE U ČAKOVCU

STRUČNI PRIJEDIPLOMSKI STUDIJ RAČUNARSTVO

BARLOVIĆ VANJA

KUĆNI INFORMACIJSKI SUSTAV BAZIRAN NA RASPBERRY PI

HOME INFORMATION SYSTEM BASED ON RASPBERRY PI

ZAVRŠNI RAD

Mentor:

Jurica Trstenjak, dipl.ing.

Čakovec, 2023.

MEĐIMURSKO VELEUČILIŠTE U ČAKOVCU
ODBOR ZA ZAVRŠNI RAD

Čakovec, 26. siječnja 2023.

država: **Republika Hrvatska**
Predmet: **Računalne mreže**

ZAVRŠNI ZADATAK br. 2022-RAČ-R-26

Pristupnik: **Vanja Barlović (0336026740)**
Studij: Redoviti preddiplomski stručni studij Računarstvo
Smjer: Inženjerstvo računalnih sustava i mreža

Zadatak: **Kućni informacijski sustav baziran na Raspberry PI**

Opis zadatka:

Pomoću Raspberry Pi razvojnog okruženja potrebno je realizirati kućni informacijski sustav baziran na WIN10 Operativni sustav. Potrebno je instalirati WIN10 operativni sustav na SD karticu sa koje će se pokretati. Također je potrebno spojiti vanjski dodirni monitor, te osigurati potrebno napajanje za Raspberry PI razvojnu pločicu. Za primjer upotrebe IoT, koristiti informacije o npr. vremenskoj prognozi (temperatura, tlak i vlažnost), informacije o temperaturi iz prostorije, te kontrolirati rasvjetom u prostoriji.

Rok za predaju rada: 20. rujna 2023.

Mentor:

Jurica Trstenjak, v. pred.

Predsjednik povjerenstva za završni ispit:

ZAHVALA

Posebnu zahvalnost iskazujem svom mentoru Jurici Trstenjaku, na uloženom trudu i vremenu te na stručnim savjetima i prijedlozima kojima mi je pomogao pri izradi ovog završnog rada.

Najveću zahvalnost dugujem svojim roditeljima, sestri i prijateljima koji su uvijek bili uz mene te mi pružili podršku i razumijevanje tijekom studiranja i izrade završnog rada.

Vanja Barlović

Sažetak

Ovaj završni rad se temelji na izradi aplikacije kućnog informacijskog sustava koji je baziran na Raspberry Pi 3B mikroračunalu s instaliranim Windows 10 IoT Core operativnim sustavom. Glavni cilj aplikacije je pružati korisnicima informacije o vremenskoj prognozi, pratiti temperaturu i vlažnost prostorije pomoću DHT11 senzora te omogućiti upravljanje rasvjetom putem relejnog modula. Aplikacijska logika je potpuno izrađena pomoću C# jezika, dok je XAML[1] deklarativni jezik korišten za izradu korisničkog sučelja. Aplikacija se izrađuje u programu Visual Studio 2022. Uz pojavu Windows 10 operativnog sustava, pojavila se nova aplikacijska platforma – UWP[2], koja omogućuje pokretanje aplikacije na različitim Windows platformama. Kroz aplikaciju koristimo se GPIO pinovima koji se nalaze na Raspberry-u. Oni nam služe za povezivanje vanjskih komponenti kao što su DHT11 senzor i relejni modul. Kako bismo mogli koristiti DHT11 senzor na Raspberry-u, potrebno je instalirati posebnu biblioteku. Signalni pin na Raspberry-u na koji je spojen DHT11 senzor, potrebno je postaviti u „Input" način rada kako bismo mogli čitati podatke s njega. Koristeći se dodatnom bibliotekom, inicijaliziramo senzor te očitavamo podatke s njega i prikazujemo ih u korisničkom sučelju. Dodatnu biblioteku preuzimamo s „NuGet Packet Managera" koji je dio Visual Studia. Relejni modul također spajamo na odgovarajuće GPIO pinove te jednostavnim metodama omogućujemo upravljanje rasvjetom. Relej se upravlja pomoću jednostavnih manipulacija pin-a na koji je spojen relejni modul. Pin na Raspberry-u je potrebno postaviti na „Output" način rada kako bi mogao slati signal koji će se slati putem gumba. Na korisničkom sučelju nalazi se gumb koji služi za uključivanje i isključivanje rasvjete. Za dohvat vremenske prognoze, koristiti ćemo se OpenWeatherMap pružateljem usluge. On nam omogućuje korištenje njihovog API[3]-a pomoću kojeg dohvaćamo trenutnu i višednevnu vremensku prognozu. Za dohvat vremenske prognoze potrebno je napraviti pretvorbu podataka iz JSON[4] oblika u C# objekte.

Aplikacija će sadržavati tri izbornika, u svakom će se nalaziti zasebna stavka te će biti osjetljiva na dodir, a korisnik će uz korištenje odgovarajućeg ekrana moći upravljati njome samo prstima.

[1] XAML - (Extensible Application Markup Language) je deklarativni jezik koji je baziran na XML-u. Koristi se za izradu korisničkog sučelja.
[2] UWP- (Universal Windows Platform) - Programska platforma za sve Microsoftove proizvode.
[3] API - (Application Programming Interface) - https://www.pcmag.com/encyclopedia/term/api
[4] JSON - (JavaScript Object Notation) format za razmjenu podataka koji je lako čitljiv. Baziran je na sintaksi JavaScripta-a. Služi za prijenos podataka između klijentskih aplikacija i različitih servisa

Ključne riječi: *Kućni informacijski sustav, API, UWP, Raspberry Pi, GPIO, IoT*

Sadržaj

1. UVOD 4

2. INTERNET OF THINGS 5

3. RASPBERRY PI 6

4. WINDOWS 10 IOT 7

5. OPENWEATHERMAP 8

6. KOMPONENTE KORIŠTENE U RADU 9

6.1. Raspberry Pi 3b 9

6.2. DHT11 senzor temperature i vlage 11

6.3. 5V relejni modul 13

6.4. MicroSD kartica 14

6.5. Monitor 15

6.6. Napajanje 16

7. INSTALACIJA OPERATIVNOG SUSTAVA 17

7.1. Upravljanje Raspberry-em pomoću Open Device Portal-a 20

8. IZRADA APLIKACIJE 21

8.1. Korištene biblioteke 21

8.2. Vremenska Prognoza 23

8.2.1. Provjeravanje API-a pomoću programa „Postman“ 23

8.2.2. Dohvat vremenske prognoze 24

8.3. Čitanje podataka s DHT11 senzora 27

8.4. Upravljanje rasvjetom pomoću relejnog modula 31

8.5. Izrada korisničkog sučelja 35

9. GOTOV PROIZVOD 36

11. ZAKLJUČAK 39

12. Popis kratica 40

13. Popis slika 41

14. Popis tablica.....41

15. Popis kodova.....41

16. LITERATURA 43

PRILOG 1. Programski kôd klase „Vrijeme“.....45

PRILOG 2. Programski kôd klase „RelejKlasa“.....47

PRILOG 3. Programski kôd „KontrolaRasvjetomPage“ 48

PRILOG 4. Programski kôd „SenzorPage“ 50

PRILOG 5. Programski kôd „VremenskaPrognozaPage“.....53

PRILOG 6. XAML kôd „VremenskaPrognozaPage“.....56

PRILOG 7. XAML kôd „SenzorPage“ 58

PRILOG 8. XAML kôd „KontrolaRasvjetomPage“ 59

1. UVOD

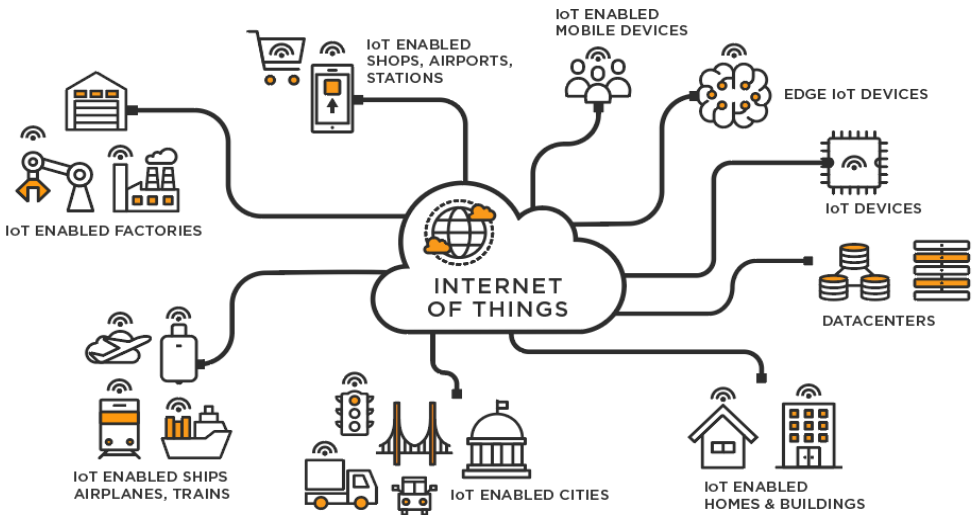
U današnje digitalno doba u kojem se automatizacija primjenjuje u gotovo svim sferama života, počevši od industrije gdje strojevi već odavno međusobno komuniciraju i razmjenjuju proizvodne podatke, zašto ne bismo dio te automatizacije uveli u naše domove? Automatizacija i digitalizacija u našem domu uvelike nam mogu poboljšati udobnost življenja. Zamislite sustav u kojem s jednog mjesta možete upravljati svjetlima u više prostorija, možete pratiti temperaturu soba, itd. Ovaj rad bavi se upravo takvom idejom.

U radu će se predstaviti inovativno rješenje koje omogućava integraciju različitih funkcionalnosti kao što su praćenje vremenske prognoze, praćenje temperature i vlažnosti sobe te upravljanje rasvjetom. Kućni informacijski sustav temeljit će se na popularnoj platformi Raspberry Pi. Raspberry Pi pruža iznimnu prilagodljivost, fleksibilnost, pouzdanost i niske troškove kod izrade takvog sustava. Korištenjem pravilnog operativnog sustava, Windows 10 IoT Core, omogućuje se jednostavna integracija komponenata koja će biti dio kućnog informacijskog sustava, odnosno pametne kuće. Projektirani sustav manje je kompleksan od komercijalnih proizvoda, ali je dobar pokazatelj kako komponente rade te ponavljanjem pojedinih metoda, sustav može postati "pametniji". Komponente koje će se koristiti, prvenstveno su namijenjene za Arduino razvojnu pločicu. Glavni koraci u izradi ovog rada su programiranje aplikacije te izrada samog sklopa. U nastavku će rada biti objašnjene sve komponente potrebne za rad kućnog informacijskog sustava, koja je njihova svrha i princip njihova rada. Vremenska prognoza je jedina komponenta koja nije fizička. Za dohvat vremenske prognoze koristi se popularni OpenWeatherMap, koji omogućuje dohvat vremenske prognoze preko njihovog API-a. Uz objašnjenje komponenti, u radu je objašnjen i programski kod pomoću kojeg je izrađena cijela aplikacija. Nakon svake implementacije koda, aplikacija je testirana u samom Raspberry-u kako bi se pravovremeno mogle otkriti bilo kakve nepravilnosti.

2. INTERNET OF THINGS

Internet of things, ili skraćeno IoT već je dobro poznat pojam. Prvi put ga je upotrijebio britanski tehnološki pionir, Kevin Ashton. On je opisao sustav u kojem bi se objekti u fizičkom obliku mogli spojiti na Internet pomoću senzora. Ashton je izmislio taj izraz kako bi prikazao „moć“ povezivanja RFID oznaka u korporativnim opskrbnim lancima s internetom kako bi pratili robu bez potrebe za ljudskom intervencijom.[1] Iako je IoT relativno nov pojam, koncept umrežavanja računala s mrežom postoji već od davnih 1970-ih godina, kada su se koristili sustavi za praćenje brojila na električnoj mreži putem telefonskih linija. U 1990-im godinama bežična tehnologija je omogućila „Machine-to-Machine[5]“ postrojenjima nadzor i udaljeno upravljanje strojevima. Povezivost je koristila namjenske zatvorene mreže industrijskih standarda, umjesto današnjeg Internet protokola (IP) i internetskim standardima.[1]

Prvi uređaj koji je bio IP adresom povezan na Internet, a da nije računalo, bio je toster koji se mogao ugasiti i upaliti preko Interneta. Sljedećih godina pojavile su se druge umrežene stvari, kao na primjer aparat za gazirane napitke i posuda za kavu.[1] Danas je „Internet stvari“ postao popularan pojam koji se koristi za opisivanje situacija u kojima se internetska povezivost i računalne mogućnosti primjenjuju na razne objekte, uređaje i senzore koje koristimo svakodnevno u različitim aspektima života, što je vidljivo na slici 1.[1]



Slika 1. Internet of things - ilustracija

Izvor: https://businesstech.bus.umich.edu/wp-content/uploads/2023/01/iot2.png (7.6.2023)

[5] Machine-to-Machine - Automatska komunikacija između uređaja s malo ili nikakvom ljudskom intervencijom. https://www.pcmag.com/encyclopedia/term/m2m

Internet stvari može se primijeniti na čovjeku u obliku uređaja koji se mogu nositi ili koji su ugrađeni u njega. Primjer ugradbenog IoT uređaja bi bio pacemaker ili uređaj za praćenje razine šećera u krvi, dok bi nosivi uređaji bili aktualni pametni satovi koji imaju mnoštvo funkcija za praćenje.[2]

3. RASPBERRY PI

Raspberry Pi je mala računalna pločica, slika 2., koja pripada u kategoriju razvojnih pločica koja je primarno bila namijenjena za svakodnevnu medijsko korištenje, no s vremenom se prepoznao njezin kapacitet te je postala idealna edukacijska platforma. Raspberry Pi je veličinom gotovo jednaka bankovnim karticama što je čini vrlo moćnim računalom s obzirom na njezine dimenzije.[3]

Slika 2. Raspberry Pi 3B

Izvor: https://www.reichelt.com/fr/fr/raspberry-pi-3-b-4-x-1-4-ghz-1-go-ram-wifi-bt-raspberry-pi-3b--p217696.html (7.6.2023)

Iako malena, pločica posjeduje više nego dovoljan broj I/O sučelja kojim korisnik može raspolagati. Za snagu pločice zadužen je procesor tvrtke Broadcom koji se bazira na ARM arhitekturi. ARM arhitektura nije tipična za računala. ARM arhitekturu pronalazimo u našim

mobitelima, no baš zbog toga je idealan izbor za Raspberry jer može obavljati zahtjevnije zadatke uz malu potrošnju struje i bez puno zagrijavanja. Također, to znači da se na njoj ne mogu pokretati računalni programi koji su rađeni za procesore arhitekture x86.[3]

Raspberry Pi nema jednu striktnu namjenu, može se koristiti na bezbroj načina i zbog toga je mnogi vole.[4] Namjena bi se mogla podijeliti u 3 veće kategorije:

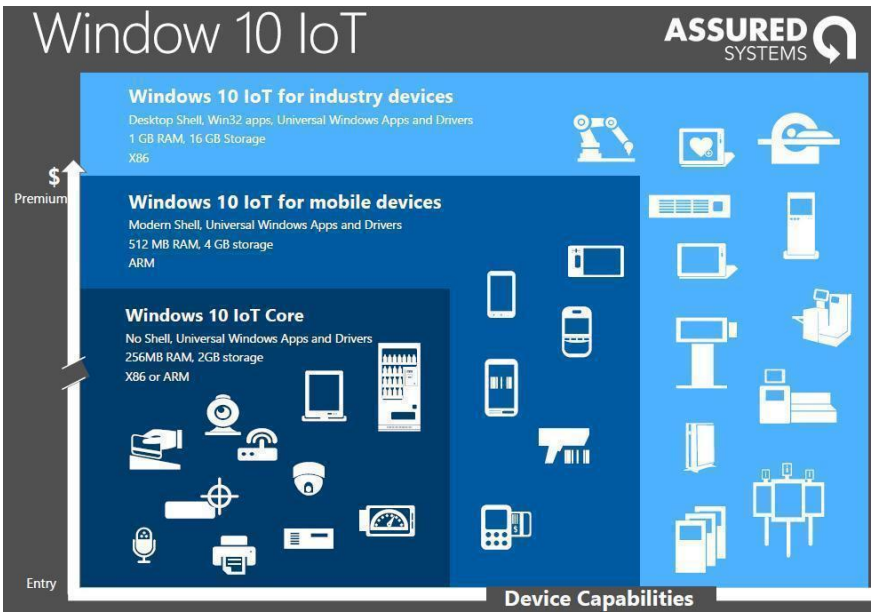
- Svakodnevno korištenje
    - Na Raspberry se instalacijom željenog operativnog sustava mogu „otključati“ mogućnosti koje inače možemo koristiti na stolnim računalima. Pa se tako mogu koristiti programi za uređivanje teksta, tablica, može se pretraživati po webu, pokretati razne multimedijske datoteke i još mnogo toga.
- Korištenje u edukacijske svrhe
    - Ako koristimo “Raspbian”, njihov službeni OS, on dolazi s predinstaliranim interpreterima i kompajlerima za različite programske jezike. Može se programirati u Javi, Pythonu, Rubyu itd.
- Projektna platforma
    - Zbog njezine mogućnosti integracije s drugim električnim uređajima putem GPIO sučelja, može se koristiti kod raznih projekata. Ovaj rad može ovdje poslužiti kao primjer.[4]

4. WINDOWS 10 IOT

Razvijanjem Windows 10 operativnog sustava, nastala je filozofija u kojoj se koristi jedan operativni sustav na više različitih uređaja. Plan je bio koristiti dva bazna operativna sustava, primjerice za pametne telefone i stolna računala. Cilj je razviti UWP aplikacije koje će biti podržane na svakoj inačici sustava, od slabije zahtjevnih IoT uređaja (u ovom slučaju Raspberry Pi), pa sve do igraćih konzola ili pametnih ploča.[5]

U radu ćemo se fokusirati na Windows IoT operativni sustav koji je prilagođen stvaranju rješenja u IoT-u. Na slici 3. prikazane su tri verzije Windows IoT-a i njihova primjena, odnosno:

- Windows 10 IoT Enterprise;
- Windows 10 IoT Mobile;
- Windows 10 IoT Core.[5]



Slika 3. Windows 10 IoT i njegova primjena

Izvor: https://www.assured-systems.com/uploads/media/windows-10-iot.jpg (7.6.2023)

Bitan nam je Windows 10 IoT Core jer je napravljen za procesore koji koriste ARM arhitekturu i potrebno mu je malo resursa za rad. To je ujedno i „najsiromašnija“ verzija, ali je više nego dovoljna za ovaj rad. Zbog specifičnih ograničenja, samo jedna aplikacija može biti pokrenuta kao shell, dok više njih može raditi u pozadini.[5]

Najbitnija značajka kod Windows IoT Core-a su novi APIi i upravljački programi za I/O mikroupravljača kao što su: GPIO, SPI, PWM, I2C.[5]

5. OPENWEATHERMAP

OpenWeatherMap je popularna platforma za pružanje informacija o vremenskoj prognozi diljem svijeta. Pruža brojnim korisnicima jednostavan pristup meteorološkim podacima putem njihovog web API-a.[6] OpenWeatherMap nudi širok spektar vremenskih podataka.[6] Ovi podaci uključuju trenutne vremenske uvjete, dugoročne i kratkoročne prognoze, interaktivne karte, pa čak i povijesne podatke.

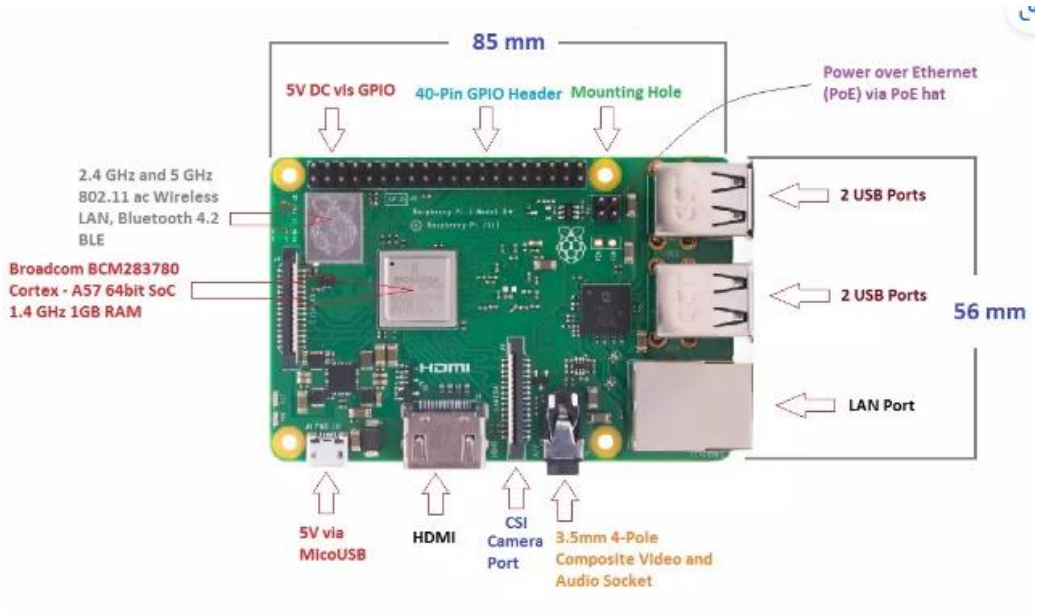
Najveća prednost OpenWeatherMap-a je jednostavna integracija. Korisnici pomoću njihovog API-ja mogu dohvatiti željene podatke pomoću raznih programskih jezika kao što su C#, Python, Java, PHP i drugi.[6] Također, OpenWeatherMap pruža i neke napredne funkcionalnosti poput

vremenskih upozorenja, geografske autentifikacije i druge.[6] No, treba naglasiti da u ovom projektu koristimo besplatni plan koji omogućuje određeni broj upita dnevno i koji je ipak nešto ograničeniji.

6. KOMPONENTE KORIŠTENE U RADU

6.1. Raspberry Pi 3b

U ovom radu koristit će se Raspberry Pi 3B. Raspberry Pi idealan je za ovaj rad zato što nudi dovoljnu snagu i dovoljan broj I/O priključaka. To je prvi model koji je stigao s trećom generacijom Raspberry-a. Slika 4. prikazuje komponente od kojih se sastoji Raspberry Pi 3B.



Slika 4. Detaljan prikaz Raspberry Pi 3B pločice

Izvor: https://images.theengineeringprojects.com/image/webp/2018/07/introduction-to-raspberry-pi-3-b-plus-1.jpg.webp?ssl=1 (7.6.2023)

Raspberry Pi 3B ima četverojezgreni Broadcom BCM2837 SoC[6] procesor od 64 bita te RAM memoriju od 1 gigabajta. Za njegov rad potrebno mu je osigurati SD karticu na kojoj će se nalaziti instalirani operativni sustav, u ovom slučaju Windows 10 IoT. Što se tiče ostalog povezivanja, na pločici se mogu naći bluetooth i wireless LAN moduli koji nam služe za bežično povezivanje, 4x USB 2.0 portova, HDMI izlaz, 1x 10/100 ethernet port, RCA VIDEO/audio izlaz te DSI konektor za monitor.[7] Glavne karakteristike Raspberry-a su procesor i njezine dimenzije. Detaljne specifikacije uređaja navedene su u tablici 1.

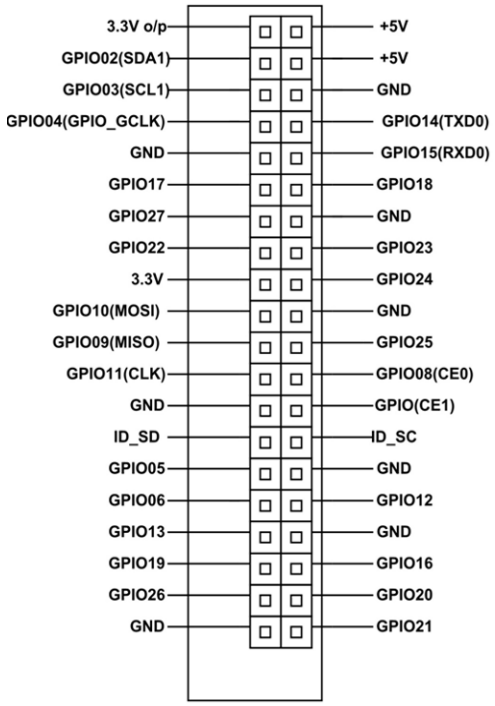
Tablica 1. Raspberry Pi 3B - specifikacije

System on Chip (SoC)	Broadcom BCM2837
CPU	Quad Cortex A53 @ 1.2GHz
GPU	VideoCore IV
Instruction set	ARM v8-A
RAM	1GB SDRAM
Pohrana	microSD
Ethernet	10/100 BaseT
Wireless	802.11n / Bluetooth 4.0
Podržana sučelja	Video izlaz: HDMI / Composite Audio izlaz: 3.5mm jack, HDMI, 4xUSB 2.0
GPIO	40
Priključak za kameru	15-pin MIPI CSI
Priključak za monitor	DSI
Dimenzije(D,Š,V)	85 x 56 x 17mm

Izvor: Autor

[6] SoC- (System-On-Chip) - predstavlja grupu upravljačkih jedinica na jednom čipu. Svaki čip je prije bio zaseban.

GPIO pinovi koji se nalaze na slici 5., služe nam za kontroliranje raznih sklopovlja, kao što su LED diode, motori, releji. Navedeno sklopovlje primjer je kako GPIO pinovi mogu slati podatke, odnosno mogu biti postavljeni na izlaz (engl. *Output*), dok primjerice senzori topline, svijetla, pokreta, šalju podatke prema Raspberry-u. Tada pinovi čitaju podatke i čine ulaz (engl. *input*). Svaki GPIO pin se može postaviti na ulaz ili izlaz s obzirom na vrstu zadataka koji obavljamo. [4]

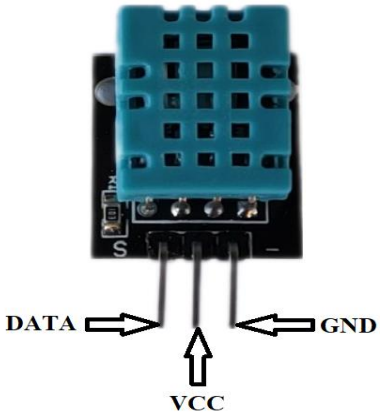


Slika 5. Prikaz pinova na Raspberry Pi 3B

Izvor: https://components101.com/microcontrollers/raspberry-pi-3-pinout-features-datasheet (7.6.2023)

6.2. DHT11 senzor temperature i vlage

U ovom radu koristit će se DHT11 senzor kojim će se pratiti temperatura i vlažnost prostorije. Koristimo kondenzatorski senzor vlage te termo senzor za očitavanje temperature. Komunikacija između senzora i Raspberry-a odvija se tako što šalje električne impulse preko podatkovnog "data" pin-a koji je povezan s "data" pin-om na DHT11 senzoru., slika 6.[8]



Slika 6. DHT11 senzor i njegovi pinovi

Izvor: Autor

Raspberry šalje zahtjev za čitanje podatka prema senzoru te po povratku signala čita pulsove koji predstavljaju dijelove informacije. Svaki puls predstavlja 1 bit poslanih podataka, 40 bitova čine kompletno očitanje temperature i vlažnosti. U tablici 2 objašnjeni su pinovi koji se nalaze na njemu.[8]

Tablica 2. Objašnjenje pinova na DHT11 senzoru

VCC	Osigurava napajanje za modul, koristi +3.3V
GND	Uzemljenje
DATA	Prenosi digitalni signal

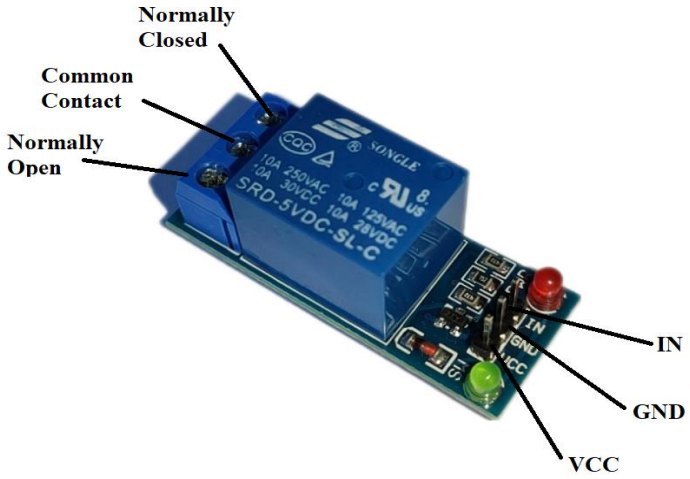
Izvor: Autor

Karakteristike DHT11 senzora navedene su u nastavku:

- Vlaga: očitavanje između 20% i 80% vlage s 5% točnošću;
- Temperatura: od 0° do 50°C s točnošću očitanja +-2%;
- Osjetno razdoblje: 1 Hz brzina uzorkovanja (jednom svake sekunde).[8]

6.3. 5V relejni modul

Relej je elektro-mehanička komponenta koja služi kao prekidač u automatskom kontroliranom krugu. Visoka struja upravlja malom strujom. Zavojnica releja napaja se istosmjernom strujom tako da se kontaktni prekidač može otvoriti ili zatvoriti. [9]



Slika 7. 5V relejni modul i njegovi pinovi i sučelje

Izvor: Autor

U radu će se koristiti jednokanalni 5V relejni modul, slika 7., koji ima zavojnicu, tri kontaktna izlaza NO, NC i COM, dva ulazna pina VCC i signalni pin te uzemljenje. Također ima i dvije kontrolne LED diode. Crvena dioda služi za prikaz stanja napajanja, dok zelena predstavlja stanje aktiviranjem releja i zavojnice preko „Input“ pina, što je detaljnije objašnjeno u tablici 3.[9]

Tablica 3. Objašnjenje pinova na relejnom modulu

Common Contact (COM)	Koristi se za povezivanje preko opterećenja koje želimo prenijeti preko modula
Normally Open (NO)	Otvoren je tako dugo dok signalni pin ne dobije signal s Raspberry–a, tada COM prekida kontakt sa NC i spaja se s NO
Normally Closed (NC)	Spojen je preko COM pin-a te čini zatvoreni strujni krug tako dugo dok ne stigne signal s Raspberry-a
IN	Signalni pin - služi za kontroliranje releja
VCC	Osigurava napajanje za modul, koristi +5V
GND	Uzemljenje

Izvor: Autor

6.4. MicroSD kartica

Za instalaciju operativnog sustava i aplikacije, potrebna nam je SD kartica, točnije za stabilan rad potrebna nam je SD kartica određenih karakteristika. Treba obratiti pažnju na sljedeće karakteristike:

- Speed class: najstariji sustav klasiranja. Najviša klasa je klasa 10 i predstavlja brzinu od 10MBps. Svaka novija SD kartica će zadovoljiti ovaj standard.
- UHS Speed class: može varirati između 1 i 3 unutar simbola „U“. Oznaka 1 označava brzinu od 10MBps, dok 3 označava 30MBps. U praksi, kartice označene s 1 mogu biti jednako brze kao i one s oznakom 3.
- Video speed class: označuje se sa simbolom „V“ i brojem pokraj. V10 označuje brzinu od 10MBps, V30 brzinu od 30MBps, itd. Ova karakteristika se odnosi na reprodukciju videozapisa te kao takva nama nije bitna.
- Application speed: novija je oznaka koja mjeri IOPS umjesto klasičnog mjerenja brzine prijenosa podataka u kontinuiranom i sekvencijalnom načinu. Označuje se sa slovom „A“ i odgovarajućim brojem. A1 označuje brzinu od 1500 IOPS čitanja, a A2 4000 IOPS čitanja. U praksi se pokazalo da nema velike razlike između A1 i A2.[10]

U ovom projektu koristit će se mikro SD kartica marke Goodram, slika 8., sljedećih specifikacija:

- 32GB
- Speed class: 10
- UHS Speed Class: U1
- Video Speed Class: V10
- Application Speed Class: A1

Slika 8. Korištena microSD kartica

Izvor: Autor

6.5. Monitor

Da bismo mogli koristiti aplikaciju, potreban nam je monitor. U radu je korišten monitor dijagonale 10.1“ te rezolucije 1280x800 piksela, koji je vidljiv na slici 9. Monitor je osjetljiv na dodir te za interakciju s aplikacijom nije potrebna ostala periferija. Monitor je naručen s Amazon.de stranice i cijena mu je bila €99. Dolazi sa stalkom i nije potrebno nikakvo dodatno kućište za njega. Monitor se povezuje s Raspberry-em preko USB kabla te je „Plug 'n play“ jer za rad touch screena nisu potrebni nikakvi upravljački programi. Za rad monitora potrebno je napajanje, odnosno može se koristiti običan punjač za mobitel ili čak USB port na samom Raspberry-u, no preporuča se vanjski izvor napajanja kako se ne bi opteretila sama pločica.[11]

Slika 9. Korišteni ekran osjetljiv na dodir

Izvor: https://m.media-amazon.com/images/I/61HfXh6+xiL._AC_SL1500_.jpg (11.6.2023)

6.6. Napajanje

Na slici 10. je prikazan službeni Raspberry Pi adapter koji se koristi za napajanje uređaja. Može se koristiti i punjač za mobitele, no u ovom je slučaju Raspberry bio nestabilan. Razlika između originalnog napajanja i npr. mobilnog punjača je u tome što Raspberry treba napajanje od 5V s konstantnom minimalnom strujom od 2.5A, dok kod mobilnog punjača struja varira i može biti manja od 2.5A. U tom slučaju dolazi do nestabilnog rada Raspberry-a što može izazvati oštećenje samog uređaja.

Slika 10. Raspberry Pi službeno napajanje

Izvor: Autor

Karakteristike napajanja dijele se na ulazne i izlazne, vidljivo u tablici 4., najbitnija karakteristika ovdje je nominalna struja opterećenja koja mora biti konstantnih 2.5A.

Tablica 4. Ulazne i izlazne karakteristike napajanja

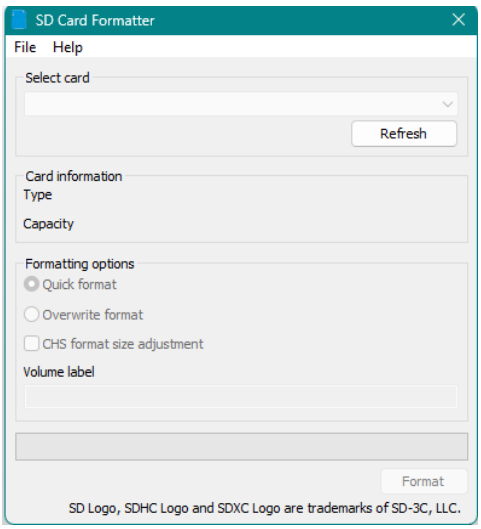
ULAZNE KARAKTERISTIKE	
Raspon napona	100-240Vac (rated) 96-264Vac (operating)
Frekvencija	50-60Hz +-3Hz
Struja	0.5A maksimum
Potrošnja energije (bez opterećenja)	0.075W maksimum
IZLAZNE KARAKTERISTIKE	
Izlazni napon	+5.1V DC
Minimalna struja opterećenja	0.0A
Nominalna struja opterećenja	2.5A
Maksimalna snaga	12.5W
Efikasnost	80.73%

Izvor: Autor

7. INSTALACIJA OPERATIVNOG SUSTAVA

Kao što je prije spomenuto, Raspberry dolazi bez pohrane te bez instaliranog operativnog sustava. Preporučeno je da se SD kartica prije daljnjeg korištenja formatira. Za formatiranje će se koristiti besplatan program „SD Card Formatter“.

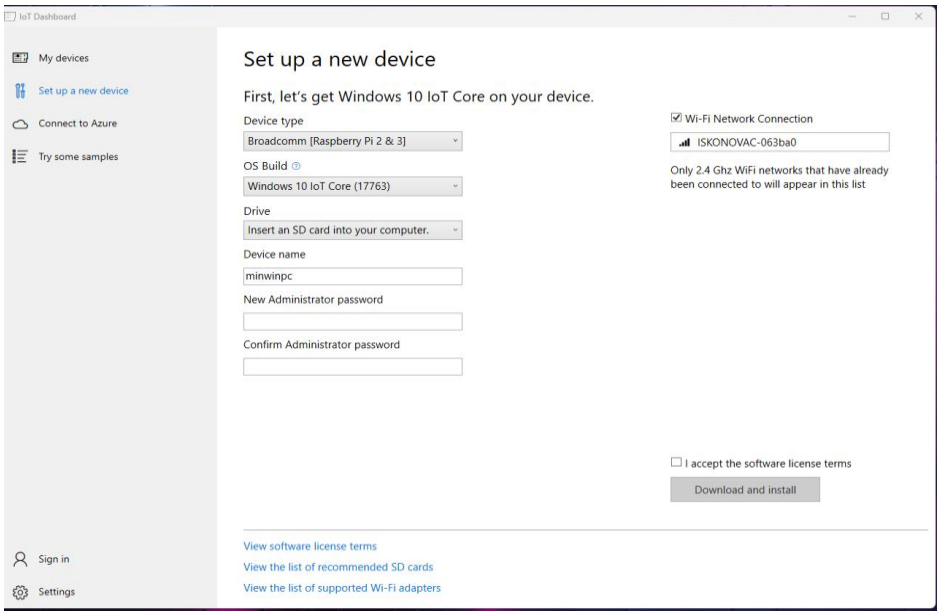
U okviru „Select card“ biramo željenu karticu i pritisnemo „Format“, slika 11.



Slika 11. SD Card Formatter program

Izvor: Autor

Nakon uspješnog formatiranja, možemo instalirati Windows 10 IoT Core putem “IoT Dashboard-a”. Slika 12. prikazuje početni izbornik, gdje počinje instalacija operativnog sustava na microSD karticu.



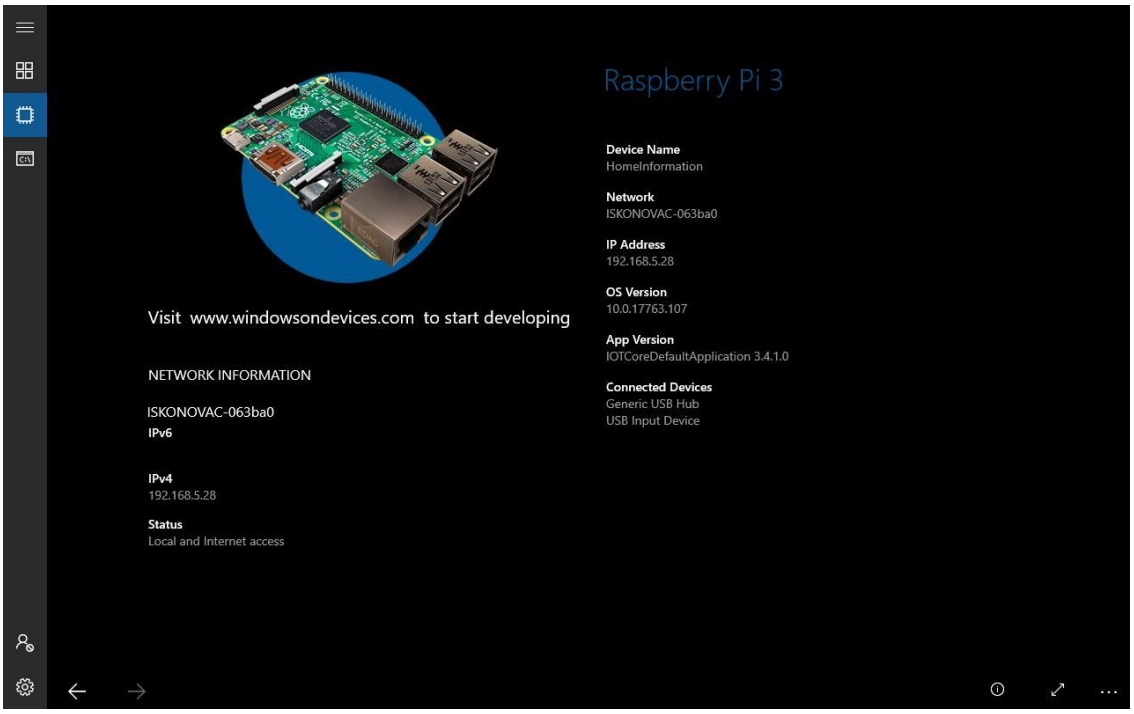
Slika 12. IoT Dashboard program

Izvor: Autor

Tijek instaliranja Windows 10 IoT na Raspberry Pi 3B:

1. Na lijevoj strani odaberemo karticu „Set up a new device“;
2. Kod „Device type“ odaberemo naš uređaj;
3. „OS Build“ je Windows 10 IoT Core (17763);
4. „Drive“ je microSD kartica na koju ćemo instalirati OS;
5. Zatim odaberemo „Device name“ i željenu lozinku;
6. Zatim “Download and install”.

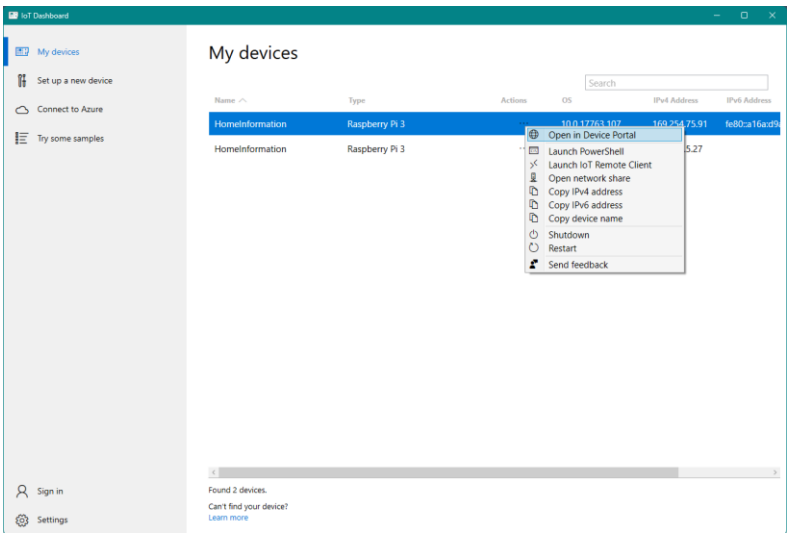
Nakon uspješne instalacije, SD kartica se stavlja u Raspberry te se uključuje priključivanjem napajanja u struju. Kod prvog pokretanja potrebno je imati spojenu osnovnu periferiju, kao što su miš, tipkovnica i monitor, u suprotnome se Raspberry neće pokrenuti. Nakon uspješnog pokretanja, dočekat će nas početni zaslon koji se može vidjeti na slici 13.



Slika 13. Prikaz Windows IoT sučelja instaliranog na Raspberry Pi

Izvor: Autor

Slika 14. prikazuje karticu „My Devices“ gdje se može vidjeti naš Raspberry koji je prethodno bio spojen na istu internetsku mrežu kao i računalo. Pritiskom na tri točkice otvara nam se padajući izbornik te se odabire opcija “Open Device Portal”.

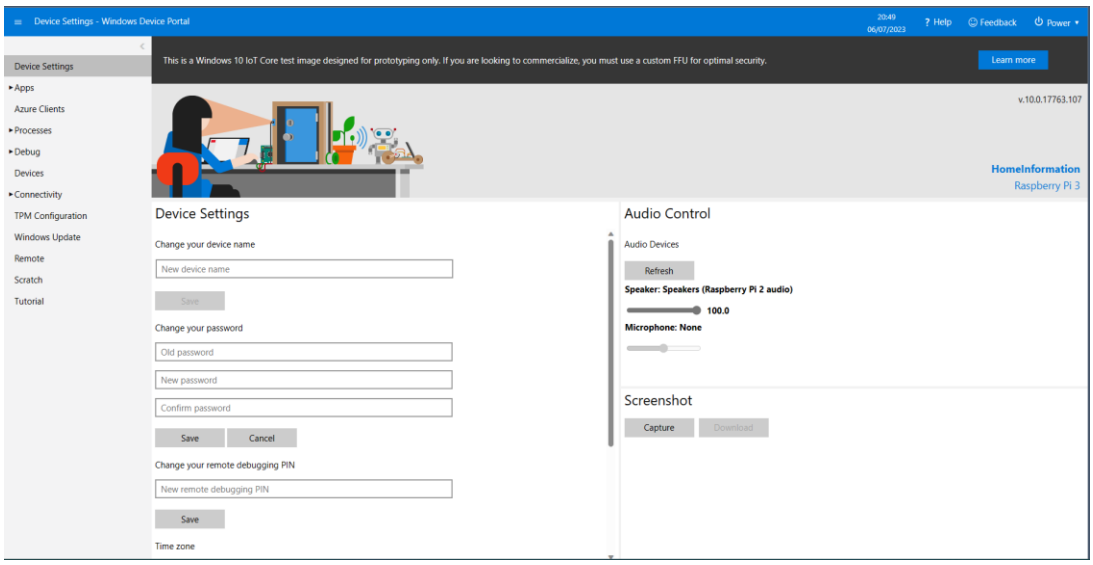


Slika 14. Prikaz izbornika za odabir instaliranih uređaja

Izvor: Autor

7.1. Upravljanje Raspberry-em pomoću Open Device Portal-a

Open Device Portal, slika 15., glavni je izbornik Windows 10 IoT-a. Ovdje se nalaze sve postavke koje možemo mijenjati na Raspberry Pi-u. Ovo je upravljačka ploča (engl. *dashboard*) uređaja. Preko njega se mogu instalirati i pokretati aplikacije, pratiti performanse uređaja, upravljati mrežama, isključiti ili ponovno pokrenuti uređaj. Upravljačkoj ploči se pristupa preko web preglednika ili preko aplikacije „IoT Dashboard“. „IoT Dashboard“ će automatski pronaći uređaj jer Raspberry ima ugrađeni prijemnik bežične mreže, no zbog stabilnosti preporuča se spajanje Raspberry-a s računalom pomoću mrežnog kabla.



Slika 15. Prikaz upravljačke ploče

Izvor: Autor

8. IZRADA APLIKACIJE

Dijelovi koda preuzeti su s interneta te su modificirani kako bi mogli raditi na željen način. [12] Kod vremenske prognoze nepotrebni objekti su potpuno izbačeni te se koristi drugačiji način i struktura serijalizacije kao i prikazivanje dobivenih podataka u XAML dijelu. Kod za upravljanje rasvjetom, odnosno modulnim relejom je nadograđen kako bi bio potpuno funkcionalan. U nastavku su opisani bitni dijelovi programskog koda kod izrade aplikacije.

8.1. Korištene biblioteke

Za rad komponenata, korištene su sljedeće biblioteke:

```
using Sensors.Dht

using System.Threading.Tasks;

using System.Runtime.Serialization.Json;

using System.Net.Http;

using Windows.Devices.Gpio;
```

Kôd 1. Prikaz korištenih biblioteka

Izvor: Autor

`Sensors.Dht`

DHT senzori su prvenstveno bili namijenjeni za Arduino razvojne pločice. Visual Studio kao takav nema službenu biblioteku koja bi nam omogućila rad s našim senzorom.

Postoje nekoliko biblioteka koje se mogu preuzeti putem NuGet packet manager-a koje će nam pomoći s upravljanjem. Jedna od tih je DHT biblioteka. DHT biblioteka je prilagođena za rad u UWP-u i u Windows 10 IoT Core okruženju. [13]

`System.Threading.Tasks`

Ova biblioteka nam služi za asinkrono programiranje, bez nje ne bismo mogli dohvaćati podatke s weba, omogućuje nam normalan rad aplikacije, dok se u pozadini izvršavaju zahtjevniji ili duži zadaci. S njom dobivamo korištenje try-catch sintakse koju koristimo za "hvatanje" grešaka.

`System.Runtime.Serialization.Json;`

Bez nje ne bismo mogli izvršavati serijalizaciju i deserijalizaciju, odnosno ne bismo mogli pretvoriti JSON datoteku u C# objekte. „DataContract" i „DataMember" pripadaju ovoj biblioteci

`System.Net.Http;`

Omogućuje nam rad sa HTTP baziranom komunikacijom. Također, nam omogućuje slanje HTTP zahtjeva te dobivanje odgovora kao i interakciju s web servisima i API-jem.

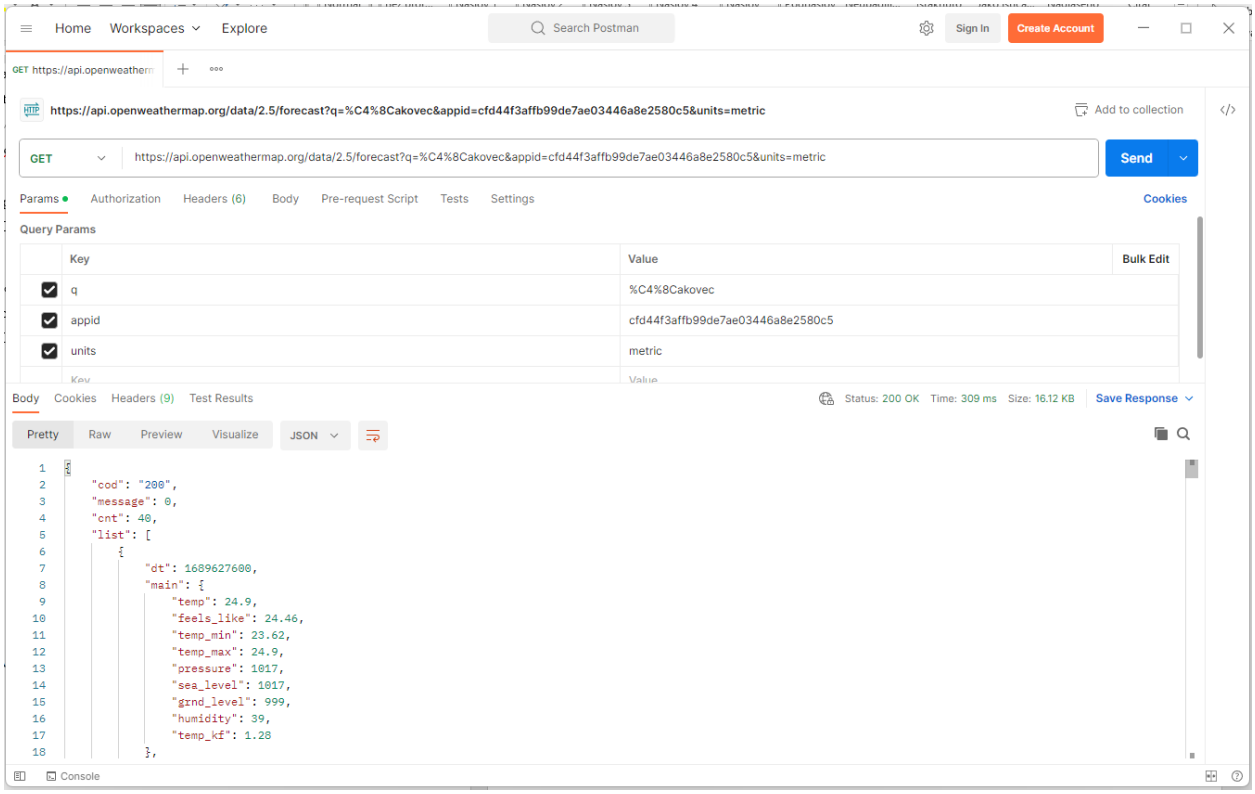
`Windows.Devices.Gpio;`

Omogućuje nam klase za interakciju s "General Purpose Input/Output" pinovima na uređajima koji koriste Windows 10 IoT Core.

8.2. Vremenska Prognoza

8.2.1. Provjeravanje API-a pomoću programa „Postman“

Postman je alat koji nam je u izradi vremenske prognoze jako pomogao. To je alat koji služi za izradu i korištenje API-a. U ovom slučaju, Postman nam pomoću našeg linka prikazuje datoteku u JSON formatu te pomoću njega izrađujemo C# objekte za vremensku prognozu. Izgled sučelja korištenog programa možete vidjeti na slici 16.



Slika 16. Prikaz programa za dohvat i provjeru API-a

Izvor: Autor

Pomoću njega možemo testirati radi li naš API zahtjev prije nego li ga implementiramo u aplikaciju. Prikazuje nam vrijednost statusnog koda, sadržaj zaglavlja te glavni sadržaj, odnosno „content“ koji mi koristimo za izradu C# objekata. Da bismo dohvatili podatke o vremenskoj prognozi, koristi se metoda koja će dohvatiti podatke sa zadanog URL-a. Za to će se pobrinuti zasebna klasa.

8.2.2. Dohvat vremenske prognoze

„HttpResponseMessage“ je klasa koja nam šalje odgovor koji smo zatražili. Sadrži dvije bitne stavke: statusni kod (engl. *StatusCode*), koji ako je uspješan može biti u rasponu od 200-299, te sadržaj (engl. *HttpContent*), koji je u ovom slučaju u JSON formatu. Ako je statusni kod uspješan tada pomoću naredbe „using(var stream=await response.Content.ReadAsStreamAsync())“ čitamo podatke s URL-a. Podaci su u JSON obliku te je potrebno napraviti pretvorbu iz JSON oblika u C# objekte. To se izvršava pomoću “DataContractJsonSerializer”, više o tome bit će objašnjeno kasnije. Metoda „DohvatiTrenutnuPrognozu“ koristi se za dohvat vremenske prognoze za zadani grad putem URL-a s našim API ključem.

```
     //Metoda za asinkroni dohvat trenutne vremenske prognoze
        public async Task<TrenutnaPrognoza>
DohvatiTrenutnuPrognozu(string cityName)
        {
            string url =
$"https://api.openweathermap.org/data/2.5/weather?q={cityName}&appid={
apiKey}&units=metric"; //Kreiranje URL-a za poziv prognoze
            HttpResponseMessage response = await
httpClient.GetAsync(url);  //Asinkrono slanje HTTP GET zahtjeva
            if (response.IsSuccessStatusCode)//Provjera statusnog koda

            {
                using (var stream = await
response.Content.ReadAsStreamAsync()) //Asinkrono čitanje podataka
                {
                    var serializer = new
DataContractJsonSerializer(typeof(TrenutnaPrognoza));
                    TrenutnaPrognoza weatherData =
(TrenutnaPrognoza)serializer.ReadObject(stream); //Deserijalizacija
podataka
                    return weatherData;
                }
            }
            else
            {

                return null;
            }
        }
```

Kôd 2. Dohvat podataka vremenske prognoze sa stranice OpenWeatherMap
Izvor: Autor

Ovo je prikaz koda za trenutnu prognozu. Jedina razlika za prikaz višednevne prognoze od trenutne prognoze je u URL-u na koji se pozivamo:

https://api.openweathermap.org/data/2.5/forecast?q={cityName}&appid={apiKey}&units=metric. Razlika je u tome da "weather" zamijenimo sa "forecast".

Sljedeći kod prikazuje C# objekte koji imaju atribute „DataContract" i „DataMember". Oni se koriste kako bi se mogla povezati JSON svojstva s C# objektima prilikom deserijalizacije.

```
[DataContract]
        public class TrenutnaPrognoza
        {
            [DataMember(Name = "main")] //Mapiranje JSON polja na
odgovarajući C# objekata
            public MainWeatherData Main { get; set; }
            [DataMember(Name = "weather")]
            public List<WeatherDescription> Weather { get; set; }
        }
```

Kôd 3. Prikaz C# objekata u vremenskoj prognozi
Izvor: Autor

„DataContract" atribut primjenjuje se na klasu "TrenutnaPrognoza" te je označava kao serijalizirajuću, odnosno to znači da će se njezina svojstva uključiti u serijalizaciju podataka. Atribut [DataMember(Name ="main")] predstavlja atribut čije će se svojstvo povezati s JSON atributom „main" prilikom serijalizacije i deserijalizacije.

Nakon što je klasa postavljena, potrebno je podatke prikazati u aplikaciji. Za prikazivanje vremenske prognoze u aplikaciji koristimo se XAML jezikom. Klasu povezujemo s logičkom (engl. *Code-behind*[7]) datotekom XAML datoteke.
Za prikaz trenutne vremenske prognoze, koristimo jednostavne elemente kao što je tekstualni blok te ih popunjavamo objektima koje smo prethodno zadali u klasi gdje se izvršava serijalizacija.

```
 // Popunjavanje elemenata na korisničkom sučelju sa dohvaćenim
podacima
                cityNameTextBlock.Text = cityName;
                temperatureTextBlock.Text =
$"{trenutnoVrijemePodaci.Main.Temperature}\u00B0C";
```

[7]Code-behind - Dio koda koji je povezan sa XAML kodom i koji je zaslužan za logiku i funkcionalnost prikaza

```
                pressureTextBlock.Text =
$"{trenutnoVrijemePodaci.Main.Pressure} hPa";
```

Kôd 4. Prosljeđivanje dohvaćenih podataka vremenske prognoze u XAML
Izvor: Autor

Za prikaz višednevne vremenske prognoze, moramo napraviti listu koja će se popunjavati pomoću "foreach" petlje koja prolazi kroz "VrijemeAPI.VremenskaPrognoza", odnosno uzima serijalizirane podatke koji su dohvaćeni putem URL-a s našim API ključem. "forecastDateTime.Hour==12" nam služi kako bismo prikazali vremensku prognozu za svaki dan točno u 12:00 sati, u suprotnome bismo dobili prognozu svakih 3 sata, što bi moglo biti nepregledno. Podaci se za svaki dan popunjavaju automatski tako što listu "ForecastItem" prosljeđujemo na XAML putem "ListView klase".

Da bi vremenska prognoza bila potpuna, potrebno je prikazati i ikone za svako vremensko stanje. Moguće je koristiti ikone koje se nalaze na "OpenWeatherMap" stranici, no zbog lošije kvalitete i čestog ne prikazivanja koristit ćemo vanjske (eng. Third-party) ikone koje pohranjujemo u mapu „Assets[8]" u aplikaciji. Da bi se te ikone mogle prikazati, potrebno je napraviti rječnik koji će povezati ime ikone s API-a s našim pohranjenim ikonama.

```
// Rječnik kojim nam je potreban da bismo mogli povezati ikone sa
lokalnog diska sa odgovarajućom prognozom
 weatherIcons = new Dictionary<string, Uri>
            {
                { "01d", new Uri("ms-appx:///Assets/Icons/sunny.png")
},
                { "02d", new Uri("ms-appx:///Assets/Icons/cloudy.png")
},
                { "03d", new Uri("ms-appx:///Assets/Icons/cloudy.png")
},
                { "04d", new Uri("ms-
appx:///Assets/Icons/overcast.png") },
                { "09d", new Uri("ms-appx:///Assets/Icons/rainy.png")
},
                { "10d", new Uri("ms-appx:///Assets/Icons/rainy.png")
},
                { "11d", new Uri("ms-
appx:///Assets/Icons/thunderstorm.png") },
                { "13d", new Uri("ms-appx:///Assets/Icons/snowy.png")
},
                { "50d", new Uri("ms-appx:///Assets/Icons/mist.png") }
            };
        }
```

Kôd 5. Rječnik korištenih ikona

Izvor: Autor

[8] Assets - mapa u kojoj se spremaju svi dodatni vanjski sadržaju koje koristimo u projektu.

Simbol „01d“ prikazuje sunčano vrijeme, „02d“ prikazuje oblačno vrijeme, itd. To su kodna imena koja dolaze s API-a te predstavljaju određeno vremensko stanje.

8.3. Čitanje podataka s DHT11 senzora

Za provjeru temperature i vlažnosti prostorije pobrinut će se DHT11 senzor kojeg smo prije opisali. Za njegov rad potrebno je instalirati biblioteku “Sensors.Dht” koja se preuzima putem NuGet packet manager-a.

```
        private GpioController gpio;
        private GpioPin dht11Pin;
        private Dht11 dht11Senzor;
        private bool CitanjePodataka;
        private void InitDht11Sensor()
        {
            var gpio = GpioController.GetDefault();//Dodavanje GPIO
kontrolera za Raspberry
            if (gpio != null) //Provjerava postoji li GPIO kontroler
            {
                dht11Pin = gpio.OpenPin(17); //Otvaranje
komunikacijskog pina
                if (dht11Pin != null)//Provjera je li pin otvoren
                {
                    dht11Senzor = new Dht11(dht11Pin,
GpioPinDriveMode.Input); //Inicijalizacija
                }
            }
   }
```

Kôd 6. Povezivanje senzora s Raspberry-em
Izvor: Autor

Na početku trebamo inicijalizirati “GpioController” klasu koja će nam omogućiti rad s pinovima na Raspberry Pi-u. “GpioPin” klasa predstavlja jedan pin pomoću kojeg ćemo konfigurirati pin, čitati ili zapisivati podatke. U našem slučaju podatke ćemo čitati, stoga na “GpioPinDriveMode”

postavljamo “Input”. Kada smo inicijalizirali senzor i konfigurirali pin, potrebno je očitati podatke s njega.

```
  DhtReading reading = await dht11Senzor.GetReadingAsync().AsTask();
//Čitanje podataka sa senzora
                if (reading.IsValid)
                {
                    double temperature = reading.Temperature; //Dohvat
temperature i vlažnosti iz objekta "reading"
                    double humidity = reading.Humidity;

                    await
Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal, ()
=> //Ažuriranje korisničkog sučelja
                    {
                        TemperaturaTextBlock.Text = $"Temperatura:
{temperature:F2}°C";  //Postavljanje podataka na odgovarajuća mjesta
                        VlažnostTextBlock.Text = $"Vlažnost:
{humidity:F2}%";
                        TemperaturaTextBlock.FontFamily = new
FontFamily("ms-appx:///Assets/Fonts/Montserrat-
Regular.ttf#Montserrat");//Promjena fonta
                        VlažnostTextBlock.FontFamily = new
FontFamily("ms-appx:///Assets/Fonts/Montserrat-
Regular.ttf#Montserrat");
                    });
```

Kôd 7. Čitanje podataka sa senzora te prosljeđivanje istih na XAML
Izvor: Autor

Prvo se poziva metoda “GetReadingAsync” kako bi se asinkrono očitali podaci sa senzora, zatim se pomoću naredbe “await” čeka da se čitanje završi prije nego se krene na ostatak koda.

“Dispatcher” klasa pobrinut će se da se kod izvrši na posebnoj dretvi (engl. *threads*), odnosno da se u ovom slučaju modifikacije korisničkog slučaja (Temperatura i Vlažnost) izvode u dretvi zaduženoj za korisničko sučelje (engl. *User Interface*). “CoreDispatcherPriority” nam samo služi za postavljanje prioriteta izvršavanja koda, u ovom slučaju je prioritet postavljen na “Normal”,

odnosno kod će se izvršavati kada dretva korisničkog sučelja (engl. *UI thread*) nije zauzeta s drugim visoko prioritetnim zadacima.

```
  private async void StartCitanjePodataka()
        {
            if (dht11Senzor != null && !CitanjePodataka) //Provjera
inicijalizacije
            {
                CitanjePodataka = true;  //Čitanje podataka

                while (CitanjePodataka) //Petlja za kontinuirano
čitanje podataka sa senzora
                {
                    await CitajDht11Podatke();
                    await Task.Delay(5000); //Čekanje idućeg čitanja 5
sekundi
                }
            }
            else
            {
                TemperaturaTextBlock.Text = "Senzor nije pronađen";
//Greška
                VlažnostTextBlock.Text = "Senzor nije pronađen";
                TemperaturaTextBlock.FontFamily = new FontFamily("ms-
appx:///Assets/Fonts/Montserrat-Regular.ttf#Montserrat");
                VlažnostTextBlock.FontFamily = new FontFamily("ms-
appx:///Assets/Fonts/Montserrat-Regular.ttf#Montserrat");
            }
        }
```

Kôd 8. Prikaz metode kontinuiranog čitanja podataka
Izvor: Autor

Metoda "StartCitanjePodataka()" nam omogućuje kontinuirano čitanje podataka sa senzora svakih 5 sekundi. Također koristimo „await" jer je metoda asinkrona.

Ako metoda ne može očitati nikakve podatke, javlja se greška "Senzor nije pronađen".

Kada senzor čita podatke, tada je pin zauzet. Kada bismo napustili stranicu u aplikaciji na kojoj se čitaju podaci i kada bismo se htjeli vratiti natrag, dobili bismo grešku da resursi nisu slobodni, odnosno da je pin zauzet.

Da bismo to izbjegli, koristimo metodu "StopCitanjePodataka()" u kojoj prekidamo čitanje sa senzora. Prekid se izvodi u metodi "Prostorija_Unloaded" kod koje provjeravamo je li pin slobodan ili zauzet, odnosno 0 ili 1, i ako je različit od nule tada metodom "Dispose()" oslobađamo prvo senzor, zatim pin. Pozivamo je na početku koda u "ProstorijaPage" klasi u kojoj se nalazi cijela logika korisničkog sučelja stranice za kontrolu temperature i vlažnosti.

```
private void Prostorija_Unloaded(object sender, RoutedEventArgs e)
//Metoda prestanka čitanja nakon napuštanja stranice
        {
            StopCitanjePodataka(); //Zaustavljanje čitanje podataka
            if (dht11Pin != null)
            {
                dht11Senzor.Dispose(); //Oslobađanje senzora
                dht11Pin.Dispose();    //Oslobađanje pina
            }
        }
```

Kôd 9. Prikaz prekida čitanja podataka i oslobađanja resursa
Izvor: Autor

"Prostorija_Loaded" ovdje nam služi za automatsko čitanje podataka svaki put kada otvorimo stranicu, isto tako se poziva u "ProstorijaPage" klasi.

```
private void Prostorija_Loaded(object sender, RoutedEventArgs e)
//Automatsko pokretanje čitanje podataka prilikom otvaranja stranice
        {
            InitDht11Sensor();      //Inicijalizacija senzora
            StartCitanjePodataka(); //Početak čitanja
        }
```

Kôd 10. Metoda čitanja podataka prilikom svakog pozicioniranja na stranicu
Izvor: Autor

8.4. Upravljanje rasvjetom pomoću relejnog modula

Kontrola releja zahtjeva sličnu logiku kao i čitanje podataka sa senzora. Razlika je u tome što se kod kontrole releja "gpio" pin mora postaviti na „Output" jer će se tada slati signal s pina na relej. Koristimo odvojenu klasu za upravljanje relejom kako bismo imali konstantno stanje releja prilikom pozicioniranja na bilo koju stranicu u aplikaciji.

```
   private void InitGPIO()
        {
            var gpio = GpioController.GetDefault();//Kreiranje
kontrolera za gpio

            if (gpio == null)  //Provjera je li gpio kontroler
slobodan
            {
                relejPin = null; //Ako je kontroler slobodan, relejPin
se postavlja na nulu
                return;
            }
            try
            {
                relejPin = gpio.OpenPin(RelejPinBroj); //Otvaranje
pina
                relejPin.SetDriveMode(GpioPinDriveMode.Output);
//postavljanje pina na izlazni režim rada
                relejPin.Write(GpioPinValue.High); //Postavljanje
vrijednosti relejPin-a na 1.
            }
            catch
            {
                relejPin = null;
            }
   }
```

Kôd 11. Povezivanje relejnog modula sa Raspberry-em
Izvor: Autor

U ovom slučaju, kada vrijednost pina postavimo na „High“, relej će tada biti ugašen. Sljedeće metode prikazuju paljenje i gašenje releja. Ovdje samo manipuliramo vrijednostima pina.

```
public void UkljuciRelej()
        {
            if (relejPin == null)
                return;
            relejPin.Write(GpioPinValue.Low); //Postavljanje
vrijednosti na 0
        }
        public void IskljuciRelej()
        {
            if (relejPin == null)
                return;
            relejPin.Write(GpioPinValue.High); //Postavljanje
vrijednosti
         }
```

Kôd 12. Uključivanje i isključivanje relejnog modula
Izvor: Autor

Za očuvanje stanja pobrinut će se klasa “DohvatiStanje()”. Ona zapisuje vrijednost samo ako je relej u upaljenom stanju.

```
  public bool DohvatiStanje() //Metoda kojom ćemo "spremiti" stanje
releja kako se ne bi ugasio
        {
            if(relejPin == null)
                return false;
            return relejPin.Read() == GpioPinValue.Low;
        }

        public void Dispose() //Metoda oslobađanja releja
        {
            if(relejPin != null)
            {
                relejPin.Dispose();
                relejPin = null;
```

```
            }
        }
```

Kôd 13. Zadržavanje uključenog stanja relejnog modula
Izvor: Autor

Kako bismo omogućili da relej bude uključen prilikom mijenjanja stranice, odnosno da svjetlo svijetli bez obzira na našu poziciju u aplikaciji, moramo napraviti metodu koja će se pokretati kad god se vratimo na našu stranicu s koje kontroliramo rasvjetom te koja će čitati stanje releja iz klase "DohvatiStanje()". Na taj način se oslobađaju resursi s pina pritom ne mijenjajući njegovo stanje, u suprotnome bi se relej vratio u prvobitno stanje – isključeno.

```
public sealed partial class KontrolaRasvjetomPage : Page,
INotifyPropertyChanged
    {
        private static RelejKlasa relejService;
        private static bool isRelejInitialized = false;
        private bool isRelejOn;
        public KontrolaRasvjetomPage()
        {
            this.InitializeComponent();
            if (!isRelejInitialized)
            {
                relejService = new RelejKlasa();
                relejService.IskljuciRelej(); // Isključen relej kao
početna vrijednost
                isRelejInitialized=true;
            }
        }
        protected override void OnNavigatedTo(NavigationEventArgs e)
//Metoda koja se poziva prilikom navigacije na tu stranicu
        {
            base.OnNavigatedTo(e);
            isRelejOn = relejService.DohvatiStanje(); //Provjera
trenutnog stanja releja i ažuriranje gumba
            AzurirajGumb();
```

```
        }
```

Kôd 14. Inicijalizacija relejnog modula prilikom pozicioniranja na stranicu za upravljanje rasvjetom
Izvor: Autor

Upravljanje izvodimo pomoću jednog gumba koji prati stanje releja te tako može sadržavati tekst „Uključi“ ili „Isključi“.

```
private void AzurirajGumb()
        {
            string gumbText = isRelejOn ? "Isključi" : "Uključi";
//Postavljanje sadržaja na osnovu trenutnog stanja gumba
            Gumb.Content = gumbText;
            Gumb.FontSize = 40;
            Gumb.FontFamily = new FontFamily("ms-
appx:///Assets/Fonts/Montserrat-Regular.ttf#Montserrat");

            if (gumbText == "Uključi")                          //Ako
je relej isključen, gumb je sivi i postavljena je slikla ugašene
žarulje
            {
                Gumb.Background = new SolidColorBrush(Colors.Gray);
                Gumb.Foreground = new SolidColorBrush(Colors.Black);
                Gumb.BorderBrush = new SolidColorBrush(Colors.Black);
                LightBulbImageOff.Visibility = Visibility.Visible;
                LightBulbImageON.Visibility = Visibility.Collapsed;
            }
            else                                                //ako
je gumb uključen, gumb poprima žutu boju i postavlja se slika upaljene
žarulje
            {
                Gumb.Background = new SolidColorBrush(Colors.Yellow);
                Gumb.Foreground = new SolidColorBrush(Colors.Black);
                Gumb.BorderBrush = new SolidColorBrush(Colors.Black);
                LightBulbImageON.Visibility = Visibility.Visible;
                LightBulbImageOff.Visibility = Visibility.Collapsed;
            }
```

```
        }
```

Kôd 15. Prikaz metode upravljanja modulom pomoću virtualnog gumba i animacija
Izvor: Autor

8.5. Izrada korisničkog sučelja

U poglavlju 8 spomenuli smo da kreiranje korisničkog sučelja koristimo XAML deklarativni jezik. Kod u nastavku prikazuje dio XAML koda kojim se prikazuje višednevna vremenska prognoza. Za povezivanje koristimo sintaksu "Binding". Tom metodom omogućuje nam se automatsko popunjavanje sučelja podacima sa samo jednim pozivanjem.

```
<StackPanel Grid.Column="1" Margin="10,0,0,0">

<TextBlock Text="{Binding Day}" FontSize="20"
VerticalAlignment="Center" FontFamily="/Assets/Fonts/Montserrat-
Bold.ttf#Montserrat" Foreground="Black"/>

<TextBlock Text="{Binding Temperature}" FontSize="16"
VerticalAlignment="Center" FontFamily="/Assets/Fonts/Montserrat-
Regular.ttf#Montserrat" F  oreground="Black"/>

<TextBlock Text="{Binding WeatherDescription}" FontSize="14"
VerticalAlignment="Center" FontFamily="/Assets/Fonts/Montserrat-
Regular.ttf#Montserrat" Foreground="Black"/>

</StackPanel>
```

Kôd 16. XAML prikaz trenutne vremenske prognoze
Izvor: Autor

9. GOTOV PROIZVOD

Kao gotov proizvod, kućni informacijski sustav bit će sastavljen u jednu cjelinu, slika 17. Raspberry Pi 3B instalirat će se na poleđinu monitora zajedno s kutijicom u kojoj se nalaze preostale komponente.

Slika 17. Stražnja i prednja strana kućnog informacijskog sustava

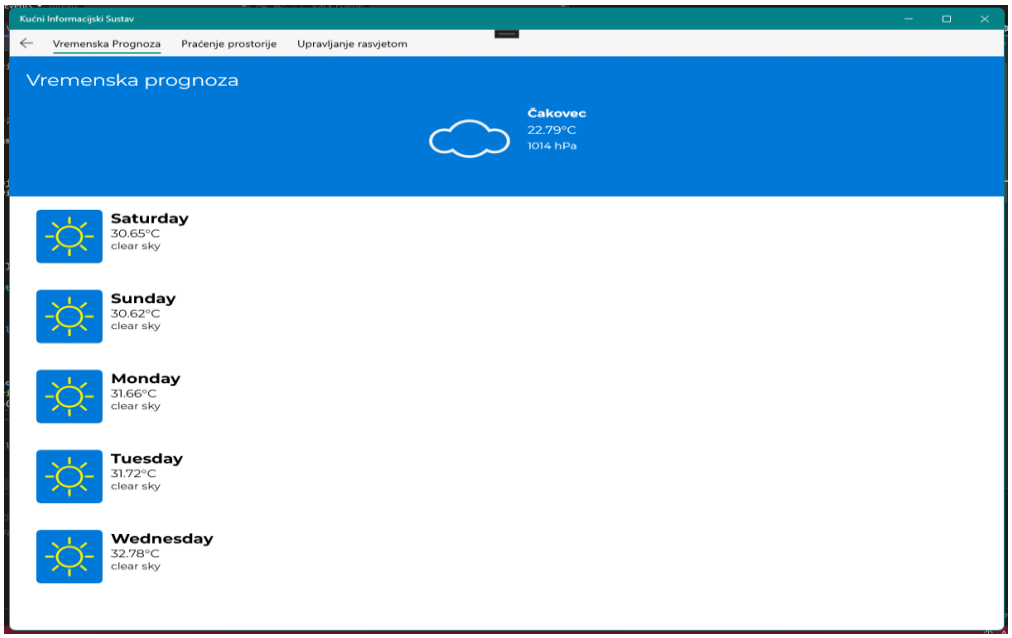
Izvor: Autor

U gornjem dijelu korisničkog sučelja nalaze se navigacijske tipke kojima se pozicioniramo na željeni prozor, slika 18.

Slika 18. Prikaz navigacijske trake

Izvor: Autor

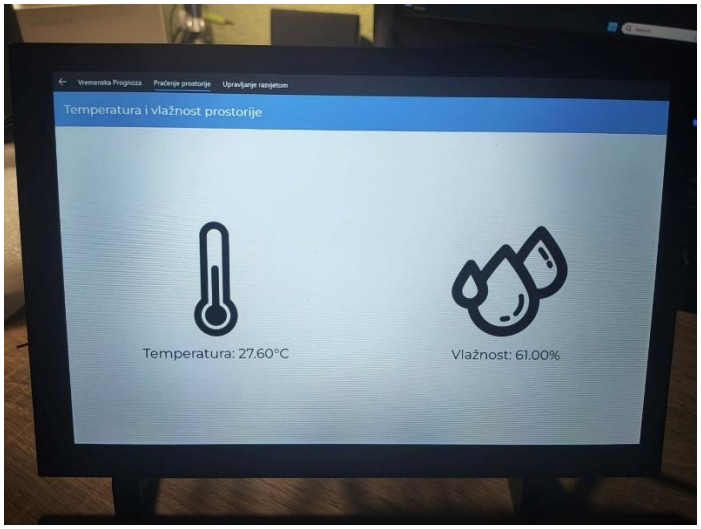
Odabirom „Vremenske prognoze“ prikazuje se vremenska prognoza, što je vidljivo na slici 19.



Slika 19. Prikaz sučelja vremenske prognoze

Izvor: Autor

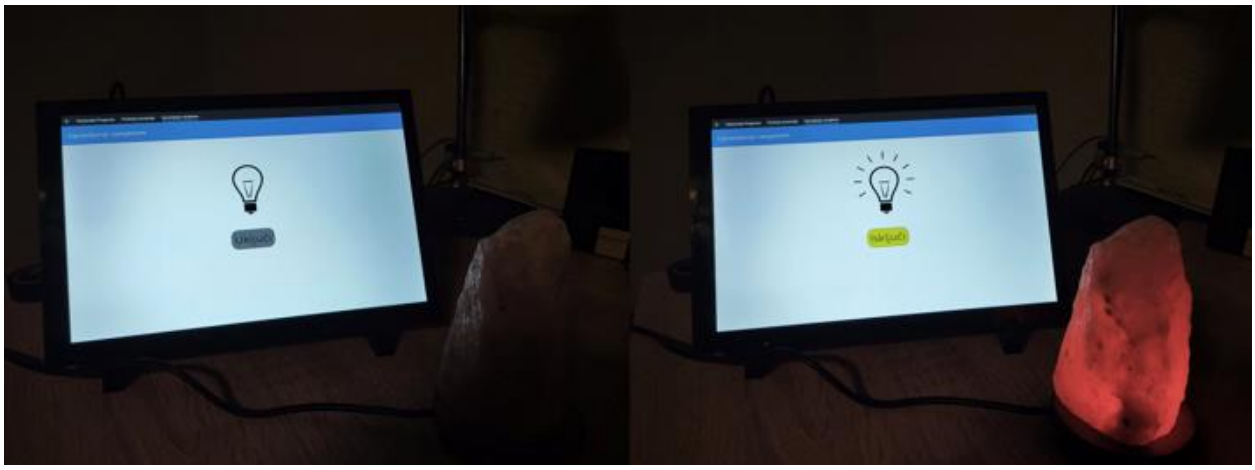
Navigiranjem na karticu „Praćenje prostorije“ prikazuje se trenutna temperatura i vlažnost prostorije, što se može vidjeti na slici 20.



Slika 20. Prikaz dijela aplikacije pokrenut na Raspberry Pi-u

Izvor: Autor

Pozicioniranjem na karticu „Upravljanje rasvjetom“ možemo uključivati i isključivati rasvjetu, slika 21. Za kontrolu rasvjete korišteno je ambijentalno svijetlo koje za napajanje koristi USB priključak. Kako bi cijeli proizvod estetski izgledao dobro, na plastičnu kutijicu montirat će se USB priključak, koji je spojen s relejnim modulom i koji struju dobiva izravno s Raspberry-a.



Slika 21. Isključena i uključena rasvjeta

Izvor: Autor

11. ZAKLJUČAK

Cilj ovog završnog rada bio je izrada i objašnjenje kućnog informacijskog sustava koji se pokreće s Raspberry Pi-a. Korisničko sučelje aplikacije prikazuje se na ekranu osjetljivom na dodir koji će s Raspberry-em biti povezan putem HDMI i USB kabla. Korisnik će se moći pozicionirati na jedan od tri izbornika. Prilikom pozicioniranja na izbornik „Vremenska prognoza“, trenutna i višednevna vremenska prognoza automatski će se učitati za grad Čakovec. U tom izborniku korisnik može vidjeti trenutnu temperaturu, tlak zraka i kratak opis prognoze. Pozicioniranjem na izbornik s podacima o temperaturi i vlažnosti prostorije, DHT11 se također automatski uključuje i počinje slati podatke. Prilikom napuštanja istog izbornika, senzor se oslobađa kako bi bio spreman za rad idući put kada odaberemo taj izbornik. Upravljanje rasvjetom izvodi se s trećeg izbornika na kojem se nalazi gumb koji uključuje, odnosno isključuje, relejni modul na koji je spojena stolna lampa. Kako bismo znali je li lampa upaljena ili ugašena, na ekranu će se prikazati animacija upaljene, odnosno ugašene žarulje.

Ovaj kućni informacijski sustav mogao bi se poboljšati tako što bismo u izradi koristili bolje komponente. Uz bolji senzor temperature i vlage, podatke bismo mogli prosljeđivati, npr. na termostat koji će uz određene parametre povećati ili smanjiti grijanje. Može se dodati više relejnih modula, te bi se time moglo upravljati rasvjetom u cijelome domu. Možemo mu dodati i funkciju digitalnog okvira za slike. Što se tiče fizičkih promjena hardvera, moguće je prilagodit cijeli sklop tako da se može montirati na zid.

12. Popis kratica

IP	*Internet Protocol*	Internet protokol
RFID	*Radio-Frequency identification*	Radiofrekvencijska identifikacija
ARM	*Advanced RISC Machines*	Napredni RISC strojevi
OS	*Operation System*	Operativni sustav
GPIO	*General Purpose Input/Output*	Opće namijenjeni ulaz/izlaz
I/O	*Input/Output*	Ulaz/izlaz
SPI	*Serial Peripheral Interfac*e	Serijsko periferno sučelje
PWM	*Pulse-Width Modulation*	Modulacija širine pulsa
I2C	*Inter-Integrated Circuit*	Među Integrirani krug
RCA	*Radio Corporation of America*	Radio korporacija Amerike
DSI	*Display Serial Interface*	Sučelje zaslona
LAN	*Local Area Network*	Lokalna mreža
RAM	*Random Access Memory*	Radna memorija
SDRAM	*Synchronous dynamic random-access memory*	
Sinkrona dinamička radna memorija		
HDMI	*High-Definition Multimedia Interfac*e	Sučelje za multimedijski prijenos visoke definicije
MIPI	*Mobile Industry Processor Interface*	Sučelje mobilne industrije za procesore
CSI	*Camera Serial Interface*	Sučelje kamere
LED	*Light Emitting Diode*	Svjetlosna dioda
USB	*Universal Serial Bus*	Univerzalna serijska sabirnica
DC	*Direct Curren*t	Istosmjerna struja
URL	*Uniform Resource Locators*	Usklađeni lokator sadržaja
XML	*Extensible Markup Language*	Jezik za označavanje podataka
IOPS	*Input/Output Operations per Second*	Ulazno/Izlazne operacije u sekundi

13. Popis slika

Slika 1. Internet of things - ilustracija 5
Slika 2. Raspberry Pi 3B 6
Slika 3. Windows 10 IoT i njegova primjena 8
Slika 4. Detaljan prikaz Raspberry Pi 3B pločice 9
Slika 5. Prikaz pinova na Raspberry Pi 3B 11
Slika 6. DHT11 senzor i njegovi pinovi 12
Slika 7. 5V relejni modul i njegovi pinovi i sučelje 13
Slika 8. Korištena microSD kartica 15
Slika 9. Korišteni ekran osjetljiv na dodir 16
Slika 10. Raspberry Pi službeno napajanje 16
Slika 11. SD Card Formatter program 18
Slika 12. IoT Dashboard program 18
Slika 13. Prikaz Windows IoT sučelja instaliranog na Raspberry Pi 19
Slika 14. Prikaz izbornika za odabir instaliranih uređaja 20
Slika 15. Prikaz upravljačke ploče 21
Slika 16. Prikaz programa za dohvat i provjeru API-a 23
Slika 17. Stražnja i prednja strana kućnog informacijskog sustava 36
Slika 18. Prikaz navigacijske trake 36
Slika 19. Prikaz sučelja vremenske prognoze 37
Slika 20. Prikaz dijela aplikacije pokrenut na Raspberry Pi-u 37
Slika 21. Isključena i uključena rasvjeta 38

14. Popis tablica

Tablica 1. Raspberry Pi 3B - specifikacije 10
Tablica 2. Objašnjenje pinova na DHT11 senzoru 12
Tablica 3. Objašnjenje pinova na relejnom modulu 14
Tablica 4. Ulazne i izlazne karakteristike napajanja 17

15. Popis kodova

Kôd 1. Prikaz korištenih biblioteka 21
Kôd 2. Dohvat podataka vremenske prognoze sa stranice OpenWeatherMap 24
Kôd 3. Prikaz C# objekata u vremenskoj prognozi 25

Kôd 4. Prosljeđivanje dohvaćenih podataka vremenske prognoze u XAML 26
Kôd 5. Rječnik korištenih ikona 26
Kôd 6. Povezivanje senzora sa Raspberry-em 27
Kôd 7. Čitanje podataka sa senzora te prosljeđivanje istih na XAML 28
Kôd 8. Prikaz metode kontinuiranog čitanja podataka 29
Kôd 9. Prikaz prekida čitanja podataka i oslobađanja resursa 30
Kôd 10. Metoda čitanja podataka prilikom svakog pozicioniranja na stranicu 30
Kôd 11. Povezivanje relejnog modula sa Raspberry-em 31
Kôd 12. Uključivanje i isključivanje relejnog modula 32
Kôd 13. Zadržavanje uključenog stanja relejnog modula 33
Kôd 14. Inicijalizacija relejnog modula prilikom pozicioniranja na stranicu za upravljanje rasvjetom 34
Kôd 15. Prikaz metode upravljanja modulom pomoću virtualnog gumba i animacija 35
Kôd 16. XAML prikaz trenutne vremenske prognoze 35

16. LITERATURA

[1] Rose K.; Eldridge S.; Chapin L. The Internet of things: an overview. Understanding th e Issues and Challenges of a More Connected World [Online]. Ženeva: The Internet Society; 2015. Dostupno na: https://www.internetsociety.org/wp-content/uploads/2017/08/ISOC-IoT-Overview-20151221-en.pdf (5.6.2023)

[2] Bell C. Windows 10 for the Internet of Things [Online]. SAD: Apress; 2016. Dostupno na: http://repo.darmajaya.ac.id/4525/1/Windows%2010%20for%20the%20Internet%20of%20Things%20%28%20PDFDrive%20%29.pdf (5.6.2023)

[3] Upton E.; Halfacree G. Raspberry Pi. User Guide [Online]. Ujedinjeno Kraljevstvo: John Wiley & Sons Ltd.; 2012. Dostupno na: https://www.cs.unca.edu/~bruce/Fall14/360/RPiUsersGuide.pdf (5.6.2023)

[4] Richardson M.; Wallace S. Getting Started with Raspberry Pi [Online]. SAD: O'Reilly Media; 2013. Dostupno na: http://www.multimedialab.be/doc/erg/2018-2019/Raspberry_Pi/Getting_Started_with_Raspberry_Pi.pdf (6.6.2023)

[5] Liming D. S.; Malin R. J.; Starter Guide for Windows 10 IoT Enterprise; Windows 10 IoT – The Big Reboot [Online]. SAD: Annabooks; 2015. Dostupno na: https://www.annabooks.com/Articles/Articles_IoT10/Windows-10-IoT%20Rev1.3.pdf (6.6.2023)

[6] Dewi C.; Chen R.C; International Journal of information technology and business: Integrating Real-Time Weather Forecasts Data Using OpenWeatherMap and Twitter [Online]. Indonezija: eJournal; 2019. Dostupno na: https://ejournal.uksw.edu/ijiteb/article/view/2302/1297 (7.6.2023)

[7] Upton E; Halfacree G. Raspberry Pi User Guide 4th Edition [Online]. Ujedinjeno Kraljevstvo: Wiley; 2016. Dostupno na: https://books.google.hr/books?hl=en&lr=&id=WHPhDAAAQBAJ&oi=fnd&pg=PA1&dq=raspberry+pi+user+guide+4th+edition&ots=cHfXXjz5jK&sig=XBkWjybsKcse1bpFUBvAXGqnqcw&redir_esc=y#v=onepage&q=raspberry%20pi%20user%20guide%204th%20edition&f=false (7.6.2023)

[8] Duet3D Documentation. Connecting Digital Humidity and Temperature (DHT) sensors [Online]. Dostupno na: https://docs.duet3d.com/User_manual/Connecting_hardware/Temperature_connecting_DHT#:~:text=DHT%20sensors%20are%20low%2Dcost,analog%20input%20pins%20are%20needed. (7.6.2023)

[9] ElProCus. What is a 5V Relay Module: Working & Its Applications [Online]. Dostupno na: https://www.elprocus.com/5v-relay-module/#:~:text=A%205v%20relay%20is%20an,ranges%20from%200%20to%205V (9.6.2023)

[10] tom's HARDWARE. Best microSD Cards for Raspberry Pi 2023 [Online]. Dostupno na: https://www.tomshardware.com/best-picks/raspberry-pi-microsd-cards (9.6.2023)

[11] Amazon.de [Online]. Dostupno na: https://www.amazon.de/-/en/Beumons-Raspberry-Touchscreen-Portable-Compatible/dp/B0B4VMNB42/ref=sr_1_5?keywords=raspberry+pi+10%22+display&qid=1690193411&sr=8-5 (10.6.2023)

[12] Borycki D. Programming for the Internet of Things: Using Windows 10 IoT Core and Azure IoT Suite [Online]. SAD: Microsoft Press; 2017. Dostupno na: https://www.google.hr/books/edition/Programming_for_the_Internet_of_Things/1_skDwAAQBAJ?hl=en&gbpv=1&dq=windows+10+IoT+and+c%23&pg=PT45&printsec=frontcover (13.6.2023)

[13] hackster.io. DHT11/DHT22 Temperature Sensor [Online]. Dostupno na: https://www.hackster.io/porrey/dht11-dht22-temperature-sensor-077790 (13.6.2023)

PRILOG 1. Programski kôd klase „Vrijeme“

```
namespace KucniInformacijskiSustav
{
    internal class Vrijeme
    {
        private readonly string apiKey;
        private readonly HttpClient httpClient;

        public Vrijeme(string apiKey)
        {
            this.apiKey = apiKey;
            this.httpClient = new HttpClient();
        }

        //Metoda za asinkroni dohvat trenutne vremenske prognoze
        public async Task<TrenutnaPrognoza>
DohvatiTrenutnuPrognozu(string cityName)
        {
            string url =
$"https://api.openweathermap.org/data/2.5/weather?q={cityName}&appid={
apiKey}&units=metric"; //Kreiranje URL-a za poziv prognoze
            HttpResponseMessage response = await
httpClient.GetAsync(url);  //Asinkrono slanje HTTP GET zahtjeva
            if (response.IsSuccessStatusCode)
//Provjera statusnog koda
            {
                using (var stream = await
response.Content.ReadAsStreamAsync()) //Asinkrono čitanje podataka
                {
                    var serializer = new
DataContractJsonSerializer(typeof(TrenutnaPrognoza));
                    TrenutnaPrognoza weatherData =
(TrenutnaPrognoza)serializer.ReadObject(stream); //Deserijalizacija
podataka
                    return weatherData;
                }
            }
            else
            {

                return null;
            }
        }

        public async Task<List<VremenskaPrognoza>>
DohvatiVisednevnuPrognozu(string cityName)
        {
            string url =
$"https://api.openweathermap.org/data/2.5/forecast?q={cityName}&appid=
{apiKey}&units=metric";
            HttpResponseMessage response = await
httpClient.GetAsync(url);
            if (response.IsSuccessStatusCode)
            {
```

```
                using (var stream = await
response.Content.ReadAsStreamAsync())
                {
                    var serializer = new
DataContractJsonSerializer(typeof(WeatherForecastResponse));
                    WeatherForecastResponse forecastResponse =
(WeatherForecastResponse)serializer.ReadObject(stream);
                    return forecastResponse.Forecast;
                }
            }
            else
            {
                // Handle error response
                return null;
            }
        }

        [DataContract]
        public class TrenutnaPrognoza
        {
            [DataMember(Name = "main")] //Mapiranje JSON polja na
odgovarajući C# objekat
            public MainWeatherData Main { get; set; }

            [DataMember(Name = "weather")]
            public List<WeatherDescription> Weather { get; set; }
        }



        [DataContract]
        public class MainWeatherData
        {
            [DataMember(Name = "temp")]
            public float Temperature { get; set; }

            [DataMember(Name = "pressure")]
            public float Pressure { get; set; }
        }

        [DataContract]
        public class WeatherDescription
        {
            [DataMember(Name = "description")]
            public string Description { get; set; }

            [DataMember(Name = "icon")]
            public string Icon { get; set; }
        }

        [DataContract]
        public class WeatherForecastResponse
        {
            [DataMember(Name = "list")]
            public List<VremenskaPrognoza> Forecast { get; set; }
        }
```

```
        [DataContract]
        public class VremenskaPrognoza
        {
            [DataMember(Name = "dt_txt")]
            public string DateTimeText { get; set; }

            [DataMember(Name = "main")]
            public MainWeatherData Main { get; set; }

            [DataMember(Name = "weather")]
            public List<WeatherDescription> Weather { get; set; }

            [DataMember(Name = "icon")]
            public string Icon { get; set; }

            public DateTime Date => DateTime.Parse(DateTimeText).Date;

        }

 }
}
```

PRILOG 2. Programski kôd klase „RelejKlasa“

```
namespace KucniInformacijskiSustav
{
    internal class RelejKlasa
    {
        private const int RelejPinBroj = 27;
        private GpioPin relejPin;

        public RelejKlasa()
        {
            InitGPIO();
        }

        private void InitGPIO()
        {
            var gpio = GpioController.GetDefault();//Kreiranje
kontrolera za gpio

            if (gpio == null)  //Provjera je li gpio kontroler
slobodan
            {
                relejPin = null; //Ako je kontroler slobodan, relejPin
se postavlja na nulu
                return;
            }
            try
            {
                relejPin = gpio.OpenPin(RelejPinBroj); //Otvaranje
pina
                relejPin.SetDriveMode(GpioPinDriveMode.Output);
//postavljanje pina na izlazni režim rada
```

```
                relejPin.Write(GpioPinValue.High); //Postavljanje
vrijednosti relejPin-a na 1.
            }
            catch
            {
                relejPin = null;
            }
        }

        public void UkljuciRelej()
        {
            if (relejPin == null)
                return;
            relejPin.Write(GpioPinValue.Low); //Postavljanje
vrijednosti na 0
        }

        public void IskljuciRelej()
        {
            if (relejPin == null)
                return;
            relejPin.Write(GpioPinValue.High); //Postavljanje
vrijednosti na 1
        }

        public bool DohvatiStanje() //Metoda kojom ćemo "spremiti"
stanje releja kako se ne bi ugasio
        {
            if (relejPin == null)
                return false;
            return relejPin.Read() == GpioPinValue.Low;
        }

        public void Dispose() //Metoda oslobađanja releja
        {
            if (relejPin != null)
            {
                relejPin.Dispose();
                relejPin = null;
            }
        }
    }
}
```

PRILOG 3. Programski kôd „KontrolaRasvjetomPage“

```
namespace KucniInformacijskiSustav.Views
{
    public sealed partial class KontrolaRasvjetomPage : Page,
INotifyPropertyChanged
    {

        private static RelejKlasa relejService;
        private static bool isRelejInitialized = false;
```

```
        private bool isRelejOn;

        public KontrolaRasvjetomPage()
        {
            this.InitializeComponent();

            if (!isRelejInitialized)
            {
                relejService = new RelejKlasa();
                relejService.IskljuciRelej(); // Isključen relej kao
početna vrijednost
                isRelejInitialized=true;
            }

        }

        protected override void OnNavigatedTo(NavigationEventArgs e)
//Metoda koja se poziva prilikom navigacije na tu stranicu
        {
            base.OnNavigatedTo(e);


            isRelejOn = relejService.DohvatiStanje(); //Provjera
trenutnog stanja releja i ažuriranje gumba
            AzurirajGumb();
        }

        private void Gumb_Click(object sender, RoutedEventArgs e)
//Postavljanje gumba na korisničko sučelje
        {
            isRelejOn = !isRelejOn; //Ako je relej uključen, poziva se
metoda za njegovo isključenje
            if (isRelejOn)
            {
                relejService.UkljuciRelej();
            }
            else
            {
                relejService.IskljuciRelej();
            }
            AzurirajGumb();      //Ažuriranje sadržaja gumba u
korisničkom sučelju
        }
        private void AzurirajGumb()
        {
            string gumbText = isRelejOn ? "Isključi" : "Uključi";
//Postavljanje sadržaja na osnovu trenutnog stanja gumba
            Gumb.Content = gumbText;
            Gumb.FontSize = 40;
            Gumb.FontFamily = new FontFamily("ms-
appx:///Assets/Fonts/Montserrat-Regular.ttf#Montserrat");

            if (gumbText == "Uključi")  //Ako je relej isključen, gumb
je sivi i postavljena je slikla ugašene žarulje
            {
                Gumb.Background = new SolidColorBrush(Colors.Gray);
                Gumb.Foreground = new SolidColorBrush(Colors.Black);
                Gumb.BorderBrush = new SolidColorBrush(Colors.Black);
```

```
                LightBulbImageOff.Visibility = Visibility.Visible;
                LightBulbImageON.Visibility = Visibility.Collapsed;
            }
            else  //ako je gumb uključen, gumb poprima žutu boju i
postavlja se slika upaljene žarulje
            {
                Gumb.Background = new SolidColorBrush(Colors.Yellow);
                Gumb.Foreground = new SolidColorBrush(Colors.Black);
                Gumb.BorderBrush = new SolidColorBrush(Colors.Black);
                LightBulbImageON.Visibility = Visibility.Visible;
                LightBulbImageOff.Visibility = Visibility.Collapsed;
            }
        }



        public event PropertyChangedEventHandler PropertyChanged;

        private void Set<T>(ref T storage, T value,
[CallerMemberName]string propertyName = null)
        {
            if (Equals(storage, value))
            {
                return;
            }

            storage = value;
            OnPropertyChanged(propertyName);
        }

        private void OnPropertyChanged(string propertyName) =>
PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
    }
}
```

PRILOG 4. Programski kôd „SenzorPage“

```
namespace KucniInformacijskiSustav.Views
{
    public sealed partial class SenzorPage : Page,
INotifyPropertyChanged
    {
        public SenzorPage()
        {
            InitializeComponent();
            Loaded += Prostorija_Loaded;
            Unloaded += Prostorija_Unloaded;
        }
        private GpioController gpio;
        private GpioPin dht11Pin;
        private Dht11 dht11Senzor;
        private bool CitanjePodataka;
```

```
        private void InitDht11Sensor()
        {
            var gpio = GpioController.GetDefault();
            if (gpio != null)
            {
                dht11Pin = gpio.OpenPin(17);
                if (dht11Pin != null)
                {
                    dht11Senzor = new Dht11(dht11Pin,
GpioPinDriveMode.Input);
                }

            }
        }

        private async Task CitajDht11Podatke()
        {
            try
            {
                DhtReading reading = await
dht11Senzor.GetReadingAsync().AsTask(); //Čitanje podataka sa senzora
                if (reading.IsValid)
                {
                    double temperature = reading.Temperature; //Dohvat
temperature i vlažnosti iz objekta "reading"
                    double humidity = reading.Humidity;

                    await
Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal, ()
=> //Ažuriranje korisničkog sučelja
                    {
                        TemperaturaTextBlock.Text = $"Temperatura:
{temperature:F2}°C";            //Postavljanje podataka na
odgovarajuća mjesta
                        VlažnostTextBlock.Text = $"Vlažnost:
{humidity:F2}%";
                        TemperaturaTextBlock.FontFamily = new
FontFamily("ms-appx:///Assets/Fonts/Montserrat-
Regular.ttf#Montserrat");//Promjena fonta
                        VlažnostTextBlock.FontFamily = new
FontFamily("ms-appx:///Assets/Fonts/Montserrat-
Regular.ttf#Montserrat");
                    });
                }
                else
                {
                    await
Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal, ()
=>
                    {
                        TemperaturaTextBlock.Text = "Pričekajte";
                        VlažnostTextBlock.Text = "Pričekajte";
                        TemperaturaTextBlock.FontFamily = new
FontFamily("ms-appx:///Assets/Fonts/Montserrat-
Regular.ttf#Montserrat");
```

```
                        VlažnostTextBlock.FontFamily = new
FontFamily("ms-appx:///Assets/Fonts/Montserrat-
Regular.ttf#Montserrat");
                    });
                }

            }
            catch (Exception ex)
            {
                await
Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal, ()
=>
                {
                    TemperaturaTextBlock.Text = "Greška u čitanju
podataka!";
                    VlažnostTextBlock.Text = "Greška u čitanju
podataka!";
                    TemperaturaTextBlock.FontFamily = new
FontFamily("ms-appx:///Assets/Fonts/Montserrat-
Regular.ttf#Montserrat");
                    VlažnostTextBlock.FontFamily = new FontFamily("ms-
appx:///Assets/Fonts/Montserrat-Regular.ttf#Montserrat");
                });
            }
        }

        private async void StartCitanjePodataka()
        {
            if (dht11Senzor != null && !CitanjePodataka) //Provjera
inicijalizacije
            {
                CitanjePodataka = true;  //Čitanje podataka

                while (CitanjePodataka) //Petlja za kontinuirano
čitanje podataka sa senzora
                {
                    await CitajDht11Podatke();
                    await Task.Delay(5000); //Čekanje idćeg čitanja 5
sekundi
                }
            }
            else
            {
                TemperaturaTextBlock.Text = "Senzor nije pronađen";
//Potškoće sa pronalaženjem senzora
                VlažnostTextBlock.Text = "Senzor nije pronađen";
                TemperaturaTextBlock.FontFamily = new FontFamily("ms-
appx:///Assets/Fonts/Montserrat-Regular.ttf#Montserrat");
                VlažnostTextBlock.FontFamily = new FontFamily("ms-
appx:///Assets/Fonts/Montserrat-Regular.ttf#Montserrat");
            }

        }

        private void StopCitanjePodataka()
        {
            CitanjePodataka = false;
        }
```

```
        private void Prostorija_Loaded(object sender, RoutedEventArgs
e) //Automatsko pokretanje čitanje podataka prilikom otvaranja
stranice
        {
            InitDht11Sensor();      //Inicijalizacija senzora
            StartCitanjePodataka(); //Početak čitanja
        }

        private void Prostorija_Unloaded(object sender,
RoutedEventArgs e) //Metoda prestanka čitanja nakon napuštanja
stranice
        {
            StopCitanjePodataka(); //Zaustavljanje čitanje podataka
            if (dht11Pin != null)
            {
                dht11Senzor.Dispose(); //Oslobađanje senzora
                dht11Pin.Dispose();    //Oslobađanje pina
               ;
            }

        }


        public event PropertyChangedEventHandler PropertyChanged;

        private void Set<T>(ref T storage, T value,
[CallerMemberName]string propertyName = null)
        {
            if (Equals(storage, value))
            {
                return;
            }

            storage = value;
            OnPropertyChanged(propertyName);
        }

        private void OnPropertyChanged(string propertyName) =>
PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
    }
}
```

PRILOG 5. Programski kôd „VremenskaPrognozaPage“

```
namespace KucniInformacijskiSustav.Views
{
    public sealed partial class VremenskaPrognozaPage : Page,
INotifyPropertyChanged
    {
        private Vrijeme vrijeme;
        private Dictionary<string, Uri> weatherIcons;
```

```
        public VremenskaPrognozaPage()
        {
            InitializeComponent();
            vrijeme = new Vrijeme("cfd44f3affb99de7ae03446a8e2580c5");
            Loaded += VremenskaPrognozaPage_Loaded;
            DohvatiVremenskeIkone();
        }

        private void DohvatiVremenskeIkone()
        {
            // Riječnik kojim nam je potreban da bismo mogli povezati
ikone sa lokalnog diska sa odgovarajućom prognozom
            weatherIcons = new Dictionary<string, Uri>
            {
                { "01d", new Uri("ms-appx:///Assets/Icons/sunny.png")
},
                { "02d", new Uri("ms-appx:///Assets/Icons/cloudy.png")
},
                { "03d", new Uri("ms-appx:///Assets/Icons/cloudy.png")
},
                { "04d", new Uri("ms-
appx:///Assets/Icons/overcast.png") },
                { "09d", new Uri("ms-appx:///Assets/Icons/rainy.png")
},
                { "10d", new Uri("ms-appx:///Assets/Icons/rainy.png")
},
                { "11d", new Uri("ms-
appx:///Assets/Icons/thunderstorm.png") },
                { "13d", new Uri("ms-appx:///Assets/Icons/snowy.png")
},
                { "50d", new Uri("ms-appx:///Assets/Icons/mist.png") }
            };
        }

        private async void VremenskaPrognozaPage_Loaded(object sender,
RoutedEventArgs e)
        {
            await DohvatiVremenskuPrognozu("Čakovec");
        }

        private async Task DohvatiVremenskuPrognozu(string cityName)
        {
            try
            {
                Vrijeme.TrenutnaPrognoza trenutnoVrijemePodaci = await
vrijeme.DohvatiTrenutnuPrognozu(cityName);
                List<Vrijeme.VremenskaPrognoza>
vremenskaPrognozaPodaci = await
vrijeme.DohvatiVisednevnuPrognozu(cityName);

                // Popunjavanje elemenata na korisničkom sučelju sa
dohvaćenim podacima
                cityNameTextBlock.Text = cityName;
                temperatureTextBlock.Text =
$"{trenutnoVrijemePodaci.Main.Temperature}\u00B0C";
                pressureTextBlock.Text =
$"{trenutnoVrijemePodaci.Main.Pressure} hPa";
```

```
                // Dohvati ikonu vremenskog stanja i postavi je na
Image element
                if
(weatherIcons.ContainsKey(trenutnoVrijemePodaci.Weather[0].Icon))
                {
                    Uri iconUri =
weatherIcons[trenutnoVrijemePodaci.Weather[0].Icon];
                    BitmapImage iconImage = new BitmapImage(iconUri);
                    weatherIconImage.Source = iconImage;
                }

                // Popuni ListView s podacima o vremenskoj prognozi
                List<ForecastItem> forecastItems = new
List<ForecastItem>();

                foreach (Vrijeme.VremenskaPrognoza prognozaPodaci in
vremenskaPrognozaPodaci)
                {
                    DateTime forecastDateTime =
DateTime.Parse(prognozaPodaci.DateTimeText);

                    // Prikazi samo prognozu za odabrani sat (npr.
12:00)
                    if (forecastDateTime.Hour == 12)
                    {
                        string dan =
forecastDateTime.ToString("dddd");
                        string temperatura =
$"{prognozaPodaci.Main.Temperature}\u00B0C";
                        string opis =
prognozaPodaci.Weather[0].Description;

                        // Dohvati ikonu vremenskog stanja za prognozu
                        Uri iconUriForecast;
                        if
(weatherIcons.ContainsKey(prognozaPodaci.Weather[0].Icon))
                        {
                            iconUriForecast =
weatherIcons[prognozaPodaci.Weather[0].Icon];
                        }
                        else
                        {
                            // Ako ikona nije pronađena, koristi
zadani placeholder
                            iconUriForecast = new Uri("ms-
appx:///Assets/Icons/placeholder.png");
                        }

                        forecastItems.Add(new ForecastItem(dan,
temperatura, opis, iconUriForecast));
                    }

                }
                forecastListView.ItemsSource = forecastItems;
            }
            catch (Exception ex)
            {
                errorMessageTextBlock.Text = $"Error: {ex.Message}";
```

```
            }
        }


        public class ForecastItem
        {
            public string Day { get; set; }
            public string Temperature { get; set; }
            public string WeatherDescription { get; set; }
            public Uri IconUrlForecast { get; set; }

            public ForecastItem(string dan, string temperatura, string
opis, Uri iconUrlForecast)
            {
                Day = dan;
                Temperature = temperatura;
                WeatherDescription = opis;
                IconUrlForecast = iconUrlForecast;
            }
        }


        public event PropertyChangedEventHandler PropertyChanged;

        private void Set<T>(ref T storage, T value,
[CallerMemberName]string propertyName = null)
        {
            if (Equals(storage, value))
            {
                return;
            }

            storage = value;
            OnPropertyChanged(propertyName);
        }

        private void OnPropertyChanged(string propertyName) =>
PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
    }
}
```

PRILOG 6. XAML kôd „VremenskaPrognozaPage“

```
 <Page
    x:Class="KucniInformacijskiSustav.Views.VremenskaPrognozaPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
    Style="{StaticResource PageStyle}"
    mc:Ignorable="d">
```

```
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>

        <!-- Current Weather Section -->
        <StackPanel Grid.Row="0" Background="#0078D7" Padding="20">
            <TextBlock x:Name="errorMessageTextBlock" Text="Vremenska
prognoza" FontSize="24" Foreground="White" VerticalAlignment="Center"
FontFamily="/Assets/Fonts/Montserrat-Regular.ttf#Montserrat"/>

            <Grid Margin="0,20,0,20" HorizontalAlignment="Center">
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="Auto" />
                    <ColumnDefinition Width="*" />
                </Grid.ColumnDefinitions>

                <StackPanel Orientation="Horizontal"
HorizontalAlignment="Right">
                    <Border Width="100" Height="100"
Background="#0078D7" CornerRadius="25">
                        <StackPanel Width="100" Height="100"
Background="#0078D7" HorizontalAlignment="Right">

                            <Image x:Name="weatherIconImage"
Stretch="UniformToFill" />
                        </StackPanel>
                    </Border>
                </StackPanel>

                <StackPanel Grid.Column="1" Margin="20,0,0,0">
                    <TextBlock x:Name="cityNameTextBlock" Text="Grad"
FontSize="16" Foreground="White" Margin="0,5,0,0"
FontFamily="/Assets/Fonts/Montserrat-Bold.ttf#Montserrat"/>
                    <TextBlock x:Name="temperatureTextBlock"
Text="Temperatura" FontSize="16" Foreground="White" Margin="0,5,0,0"
FontFamily="/Assets/Fonts/Montserrat-Regular.ttf#Montserrat"/>
                    <TextBlock x:Name="pressureTextBlock" Text="Tlak
zraka" FontSize="14" Foreground="White" Margin="0,5,0,0"
FontFamily="/Assets/Fonts/Montserrat-Regular.ttf#Montserrat"/>

                </StackPanel>

            </Grid>
        </StackPanel>

        <!-- Forecast Section -->
        <ListView x:Name="forecastListView" Grid.Row="1"
ItemsSource="{Binding ForecastItems}" Background="White">
            <ListView.ItemTemplate>
                <DataTemplate>
                    <Grid Margin="20">
                        <Grid.ColumnDefinitions>
                            <ColumnDefinition Width="Auto" />
                            <ColumnDefinition Width="*" />
                        </Grid.ColumnDefinitions>
```

```
                        <Grid Width="80" Height="80"
VerticalAlignment="Center" HorizontalAlignment="Center">
                            <Border Background="#0078D7" Padding="10"
CornerRadius="5">
                                <Image Source="{Binding
IconUrlForecast}" Stretch="Uniform" />
                            </Border>
                        </Grid>

                        <StackPanel Grid.Column="1" Margin="10,0,0,0">
                            <TextBlock Text="{Binding Day}"
FontSize="20" VerticalAlignment="Center"
FontFamily="/Assets/Fonts/Montserrat-Bold.ttf#Montserrat"
Foreground="Black"/>
                            <TextBlock Text="{Binding Temperature}"
FontSize="16" VerticalAlignment="Center"
FontFamily="/Assets/Fonts/Montserrat-Regular.ttf#Montserrat"
Foreground="Black"/>
                            <TextBlock Text="{Binding
WeatherDescription}" FontSize="14" VerticalAlignment="Center"
FontFamily="/Assets/Fonts/Montserrat-Regular.ttf#Montserrat"
Foreground="Black"/>
                        </StackPanel>
                    </Grid>
                </DataTemplate>
            </ListView.ItemTemplate>
        </ListView>
    </Grid>
</Page>
```

PRILOG 7. XAML kôd „SenzorPage“

```
<Page x:Class="KucniInformacijskiSustav.Views.SenzorPage"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
      mc:Ignorable="d"
      Background="White">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>
        <StackPanel Grid.Row="0" Background="#0078D7" Padding="20">
            <TextBlock Text="Temperatura i vlažnost prostorije"
FontSize="24" FontWeight="Bold" Foreground="White"
FontFamily="/Assets/Fonts/Montserrat-Regular.ttf#Montserrat"/>
        </StackPanel>
        <Grid Grid.Row="1">
            <Grid.ColumnDefinitions>
```

```
                <ColumnDefinition Width="*" />
                <ColumnDefinition Width="*" />
            </Grid.ColumnDefinitions>
            <Grid Grid.Column="0" HorizontalAlignment="Center"
VerticalAlignment="Center">
                <Grid.RowDefinitions>
                    <RowDefinition Height="Auto" />
                    <RowDefinition Height="Auto" />
                </Grid.RowDefinitions>
                <Image Source="/Assets/termometar.png" Width="250"
Height="250" />
                <TextBlock x:Name="TemperaturaTextBlock"
Text="Temperatura" Grid.Row="1" HorizontalAlignment="Center"
VerticalAlignment="Top" Margin="0,20,0,0" FontSize="30"
Foreground="Black" FontFamily="/Assets/Fonts/Montserrat-
Regular.ttf#Montserrat" />
            </Grid>
            <Grid Grid.Column="1" HorizontalAlignment="Center"
VerticalAlignment="Center">
                <Grid.RowDefinitions>
                    <RowDefinition Height="Auto" />
                    <RowDefinition Height="Auto" />
                </Grid.RowDefinitions>

                <Image Source="/Assets/humidity.png" Width="250"
Height="250" />
                <TextBlock x:Name="VlažnostTextBlock" Text="Vlažnost"
Grid.Row="1" HorizontalAlignment="Center" VerticalAlignment="Top"
Margin="0,20,0,0" FontSize="30" Foreground="Black"
FontFamily="/Assets/Fonts/Montserrat-Regular.ttf#Montserrat" />
            </Grid>
        </Grid>
    </Grid>
</Page>
```

PRILOG 8. XAML kôd „KontrolaRasvjetomPage“

```
<Page
    x:Class="KucniInformacijskiSustav.Views.KontrolaRasvjetomPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
    mc:Ignorable="d"
    Background="White">

    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>
```

```
        <StackPanel Grid.Row="0" Background="#0078D7" Padding="20">
            <TextBlock Text="Upravljanje rasvjetom" FontSize="24"
FontWeight="Bold" Foreground="White"
FontFamily="/Assets/Fonts/Montserrat-Regular.ttf#Montserrat"/>
        </StackPanel>
        <Grid Grid.Row="1">
            <Grid.RowDefinitions>
                <RowDefinition Height="3*"/>
                <RowDefinition Height="*"/>
                <RowDefinition Height="2*"/>
            </Grid.RowDefinitions>

            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="*"/>
                <ColumnDefinition Width="Auto"/>
                <ColumnDefinition Width="*"/>
            </Grid.ColumnDefinitions>
            <Image x:Name="LightBulbImageOff" Grid.Row="0"
Grid.Column="1" HorizontalAlignment="Center"
VerticalAlignment="Center" Width="250" Height="250" Stretch="Uniform">
                <Image.Source>
                    <BitmapImage
UriSource="/Assets/lightbulb_off.png"/>
                </Image.Source>
            </Image>
            <Image x:Name="LightBulbImageON" Grid.Row="0"
Grid.Column="1" HorizontalAlignment="Center"
VerticalAlignment="Center" Width="250" Height="250" Stretch="Uniform">
                <Image.Source>
                    <BitmapImage
UriSource="/Assets/lightbulb_on.png"/>
                </Image.Source>
            </Image>

        </Grid>
    </Grid>
</Page>
```