Pametna utičnica interoperabilna s Google Home ekosustavom

Dumančić, Hrvoje

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Hrvoje Dumančić

PAMETNA UTIČNICA INTEROPERABILNA S GOOGLE HOME EKOSUSTAVOM

DIPLOMSKI RAD

Varaždin, 2020.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Hrvoje Dumančić

Studij: Informacijsko i programsko inženjerstvo

PAMETNA UTIČNICA INTEROPERABILNA S GOOGLE HOME EKOSUSTAVOM

DIPLOMSKI RAD

Mentor:

Doc. dr. sc. Boris Tomaš

Varaždin, rujan 2020.

Hrvoje Dumančić

Izjava o izvornosti

Izjavljujem da je moj diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

_____

Sažetak

Diplomski rad je izrađen na temu izrade pametne utičnice direktno interoperabilne s Google Home ekosustavom. Za realizaciju projekta bilo je potrebno ispuniti par aspekata projekta, hardverski dio koji se odnosio na izradu same utičnice te softverski dio koji se odnosio na izradu aplikacija kojom će se upravljati utičnicom. Sama pametna utičnica je izrađena koristeći ESP32 mikrokontroler s izrađenom skriptom koji je spojen na relej koji propušta električnu energiju po potrebi. Sa strane softvera, bilo je potrebno implementirati određene načine upravljanja tom utičnicom. Za tu svrhu je izrađena Google pametna akcija integrirana s Google Home mobilnom aplikacijom, putem koje je moguće uključivati i isključivati utičnicu. U slučaju da osobe nemaju Google račun za upravljanje aplikacijom ili da preferiraju drugi način, izrađena je i Flutter aplikacija namijenjena Android i iOS mobilnim uređajima koja djeluje na isti način kao i pametna akcija s dodatnim prikazom trenutnog stanja pametne utičnice.

Ključne riječi: ESP32, Google Home, Flutter, Firebase, relej, mikrokontroler

Sadržaj

1. Uvod

Današnje prosječno kućanstvo sastoji se od velikog broja pametnih uređaja, svakodnevnih kućanskih objekata kojima su dodane određene „moderne“ osobine, kao što su povezivanje na internet, upravljanje putem mobitela i slično. Iako je većini ljudi prva asocijacija upravo mobitel ili televizor, postoji još mnoštvo stvari koje su u mogućnosti da se povežu na internet i da se njima na taj način upravlja. U novije vrijeme, praktički za svaki dio pokućstva i uređaj u domu, postoji njegov „smart“ ekvivalent, koji može obavljati sve stvari na identičan način, ali također može biti i upravljan putem interneta, što mu daje dodatne opcije rada. Iako je ta opcija poprilično atraktivna ljudima, postoji mala vjerojatnost ako imaju već pojedine stvari da će ih odbaciti i potrošiti veliku količinu novca na pametnu kuću. No za postizanje pametnog doma u određenim situacijama postoji i druga opcija, a to je izrada pametne utičnice. Ona bi se upravljala putem mobitela i uvjetovala prosljeđivanje struje, odnosno uključivanje i isključivanje uređaja koji su spojeni na nju i time bi se ostvarile pojedine funkcionalnosti pametnog doma. Upravo to je tema ovog diplomskog rada, izrada pametne utičnice koju je moguće upravljati putem Google Home-a, ali i putem izrađene aplikacije u programskom jeziku Flutter.

2. Metode i tehnike rada

U ovom radu istraživanje je u potpunosti obavljeno preko interneta. Zbog toga što su se tehnologije koje se koriste u ovom diplomskom radu pojavile unazad par godina, ne postoji mnoštvo knjiga koje ih opisuju na dovoljno specifičan način koji bio potreban kako bi se rad ostvario. Istraživanje se sastojalo od čitanja raznih članaka na internetu koji objašnjavaju pojedine funkcionalnosti rada elemenata i tehnologija, kloniranja postojećih repozitorija na Githubu i njihovog razumijevanja, nakon čega bi ih se pokušalo prilagoditi i osposobiti za rad na postojećim komponentama. Diplomski rad se sastoji od tri glavna dijela: izrade same pametne utičnice putem ESP32 mikrokontrolera i releja, izrada Google Home akcije koja će slati naredbe mikrokontroleru za uključivanje/isključivanje pametne utičnice i izrada mobilne aplikacije koja će također slati zahtjeve mikrokontroleru čime se pokriva i varijanta upravljanja pametnom utičnicom ne koristeći Google Home ekosustav.

Sam ESP32 mikrokontroler, a time i relej na koji je on povezan, upravljani su putem skripti koje se na njih uploadaju koristeći Arduino IDE razvojno okruženje. Te skripte su pisane u C++ programskom jeziku i jednom uploadane, ostaju zapisane u mikrokontroleru cijelo vrijeme, čak i nakon ponovnog uključivanja.

Izrada Google Home akcija se odvija putem Google korisničkog računa, prilikom kojeg se kreira i Firebase baza podataka. Iako je proces jednostavan jer Google obavi velik dio posla oko povezivanja i sinkroniziranja pojedinih elemenata, najvažniji dio aplikacije, samu logiku rada, potrebno je implementirati samostalno putem JavaScripta i uploadati na Firebase bazu podataka.

Mobilna aplikacija predstavlja zadnji element diplomskog rada, a ona je kreirana koristeći Flutter programski jezik. On omogućava da se aplikacije razvijaju na jednom mjestu, a pokreću na Android i iOS operacijskim sustavima. Ova aplikacija predstavlja alternativu korištenja Google Home akcija, odnosno obavlja istu funkciju upravljanja pametnom utičnicom, samo što nije direktno integrirana u Google home ekosustav.

3. Pametna utičnica

Ovo poglavlje diplomskog rada sastojat će se od upoznavanja i opisivanja pojedinih komponenti i programa koji su nam potrebni za realizaciju pametne utičnice. Opisat će se i objasniti određene stavke mikrokontrolera i svih komponenti potrebnih za realizaciju projekta, kao što su principi rada i načini spajanja u cjelinu.

3.1. Mikrokontroler

Prva i najvažnija komponenta koja je potrebna za realizaciju projekta je mikrokontroler. Već samo ime govori dosta o komponenti. Imenica mikro naglašava njegovu veličinu te naglašava kategoriju kojoj pripada, dok riječ kontroler govori koja mu je glavna funkcionalnost, da upravlja, odnosno kontrolira radom drugim komponenti i uređaja. To je kompaktni, integrirani strujni krug koji se na najvećoj razini može podijeliti na hadrver i softver. Softver je većinom zasnivan na Linux operativnom sustavu, gdje podržava određene funkcionalnosti koje se razvijaju u jezicima visoke razine, kao što je C ili C++, koji se koristi u ovom projektu u sklopu Arduino razvojnog okruženja. Hadrver, s druge strane predstavlja sklopovlje mikrokontrolera, odnosno dijelove od kojih je on sagrađen. Svaki model mikrokontrolera je građen drukčije, sa specifičnom namjerom olakšavanja i rada s drugim komponentama, ali na svakome možemo prepoznati određene dijelove koji su karakteristični za svaka računala. [9]

Tri su glavna dijela od kojih je sačinjen svaki mikrokontroler:

- Procesor
- Memorija
- Ulazno izlazna periferija

3.1.1. Procesor

Procesor je središnja jedinica za obradu podataka koja prima, obrađuje i prosljeđuje podatke dalje po potrebi. Glavna svrha samog procesora je izvršavanje instrukcija i njihovim izvršavanjem omogućiti funkcionalnost samog mikrokontrolera. Prethodno navedeno izvršavanje instrukcija uključuje izvršavanje osnovnih aritmetičkih, logičkih i ulazno-izlaznih operacija. Procesor je također zadužen za prebacivanje podataka između različitih komponenti unutar većeg ugrađenog sustava. Moderni procesori uz pomoć integriranih razvojnih okruženja i viših programskih jezika, u odnosu na sam Assembler, kao što je programski jezik C, omogućuju znatno lakše pisanje koda za sam mikrokontroler. [8]

3.1.2. Memorija

Sljedeća komponenta mikrokontrolera je memorija. Glavna funkcionalnost memorije je skladištiti podatke koje procesor prima i koristi kao naputke za obradu ili koje koristi kao ulazni set za samu obradu podataka. U slučaju mikrokontrolera postoje dva glavna tipa memorije.

Prvi tip memorije je programska memorija. Programska se memorija koristi za dugoročno skladištenje informacija vezanih za instrukcije koje procesor izvršava. Radi se o memoriji koja sadržava dugi niz mogućih strojnih instrukcija koje točno nalažu procesoru što mu je činiti. Programska memorija je nepromjenjiva memorija, što znači da se radi o memoriji koja zadržava podatke tijekom dužeg perioda i nije joj potreban stalan izvor struje kako bi zadržala podatke koje sadržava.

Drugi tip memorije koji se nalazi unutar mikrokontrolera naziva se podatkovna memorija i koristi se za skladištenje privremenih podataka koji se upotrebljavaju za vrijeme izvršavanja instrukcija. Za razliku od programske memorije, podatkovna memorija je promjenjiva memorija i potreban joj je stalan izvor struje kako bi zadržala podatke koje sadržava u određenom trenutku. U slučaju gubitka struje svi podaci koji se trenutno nalaze u podatkovnoj memoriji su izgubljeni.

Još jedan oblik memorije su unutarnji registri koji također pružaju svojevrsno privremeno skladištenje podatak za procesor, ali ih se ne navodi kao zasebnu jedinicu zato što su sami registri integrirani u procesor. [8]

3.1.3. Ulazno izlazna periferija

Ulazno izlaznu periferiju se smatramo sučeljem prema vanjskom svijetu. Radi se o sklopovlju koje omogućuje interakciju mikrokontrolera s vanjskim sustavima. Ulazni portovi primaju informacije i prosljeđuju ih do procesora u obliku seta podataka razumljivom procesoru. Izlazni portovi su zaduženi za slanje instrukcija vanjskim uređajima koji isti naknadno izvršavaju u obliku njima specifične akcije.

Ulazno izlazni pinovi opće namjene su pinovi koji se nalaze na samom mikrokontroleru i svaki od pinova ima svoju svrhu. Ulazno izlazni pinovi opće namjene moguće je konfigurirati kao ulazne ili izlazne. U slučaju ulaznih pinova, pinovi signaliziraju mikrokontroleru koja je voltaža prisutna na pojedinom pinu. Primjer korištenja jednog ulaznog pina je signaliziranje mikrokontroleru je li određeni gumb trenutno pritisnut. U slučaju izlaznih pinova sam mikrokontroler određuje hoće li vrijednost pojedinog izlaznog pina postaviti na visoku ili nisku voltažu. Primjer korištenja izlaznog pina je paljenje, odnosno gašenje LED žaruljice na temelju prethodno navedenih stanja pojedinog pina. Za kontroliranje pinova opće namjene postoje dio memorije koji se naziva periferni memorijski prostor ili periferni adresni prostor. [8]

3.1.4. Primjena mikrokontrolera

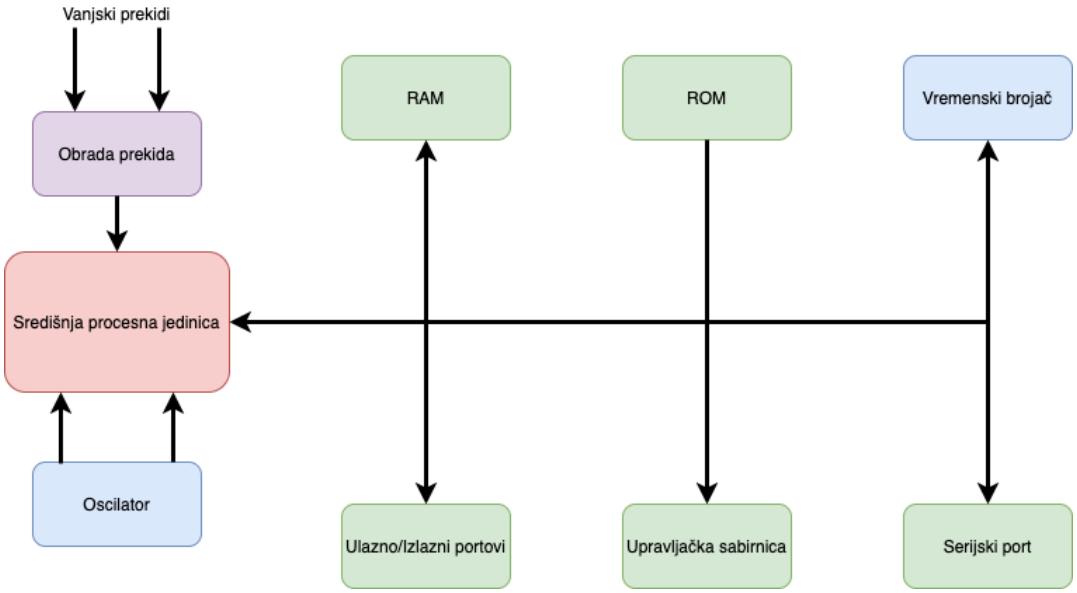
Primjena mikrokontrolera je iznimno široka i možemo pronaći u različitim područjima života. Koriste se u privatnim kućama, u automatizaciji, robotici, auto industriji, osvjetljenju, obnovljivoj energiji, komunikacijama, proizvodnji i još u mnoštvu drugih područja. Potpuna mogućnost prilagodbe i programiranja mikrokontrolera za raznolike situacije omogućava istima pojavljivanju u širokom spektru područja primjene.

Jedna od konkretnih primjena mikrokontrolera je njihovo korištenje kao procesora digitalnih signala. Ovdje govorimo o konverziji analognog signala, koji dolazi s određenom razinom šuma, u digitalni signal koji može poprimiti samo dvije vrijednosti, visoku ili nisku, odnosno jedan ili nula.

3.1.5. Razlika između mikrokontrolera i mikroprocesora

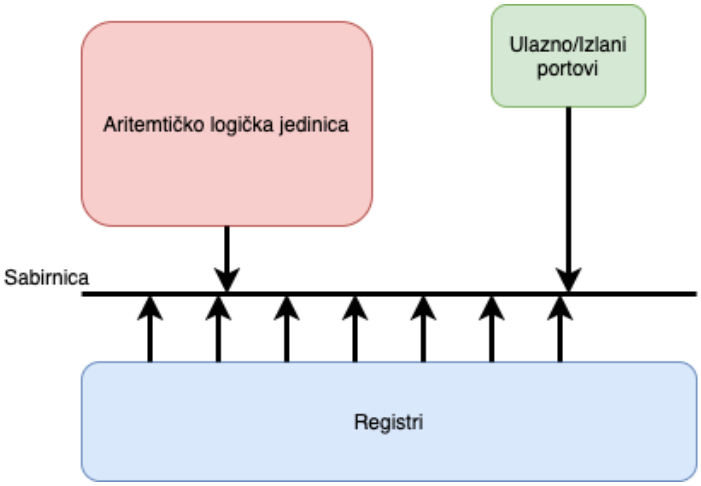
Važno je razlikovati mikrokontroler u odnosu na mikroprocesor i razumjeti njihove međusobne razlike.

Mikrokontroler možemo smatrati malim računalom koje je u cijelosti sadržano na jednom integriranom strujnom krugu. Mikrokontroler se sastoji od procesorske jezgre, RAM i ROM memorije i ulazno izlaznih pinova koji imaju široku namjenu. Sve navedene komponente su ujedno i jedine potrebne komponente za pojedinom čipu i iz toga razloga se često koriste u ugrađenim sustavima.



Slika 1. Pojednostavljena arhitektura mikrokontrolera

Mikroprocesori, za razliku od mikrokontrolera, sadrže samo središnju procesnu jedinicu. Zbog toga što ne sadrži RAM, ROM i ulazno izlazne portove namijenjene za periferiju potrebni su mu vanjski strujni krugovi periferije kako bi njegov rad bio značajan. Primjena mikroprocesora se očituje najviše u kompleksnim zadacima kao što su razvoj softvera, kompleksni izračuni u stvarnom vremenu i primjena u situacijama gdje ulazni set podataka kao i izlazni set podataka nisu unaprijed definirani i potrebno je prilagoditi rad procesora istim.



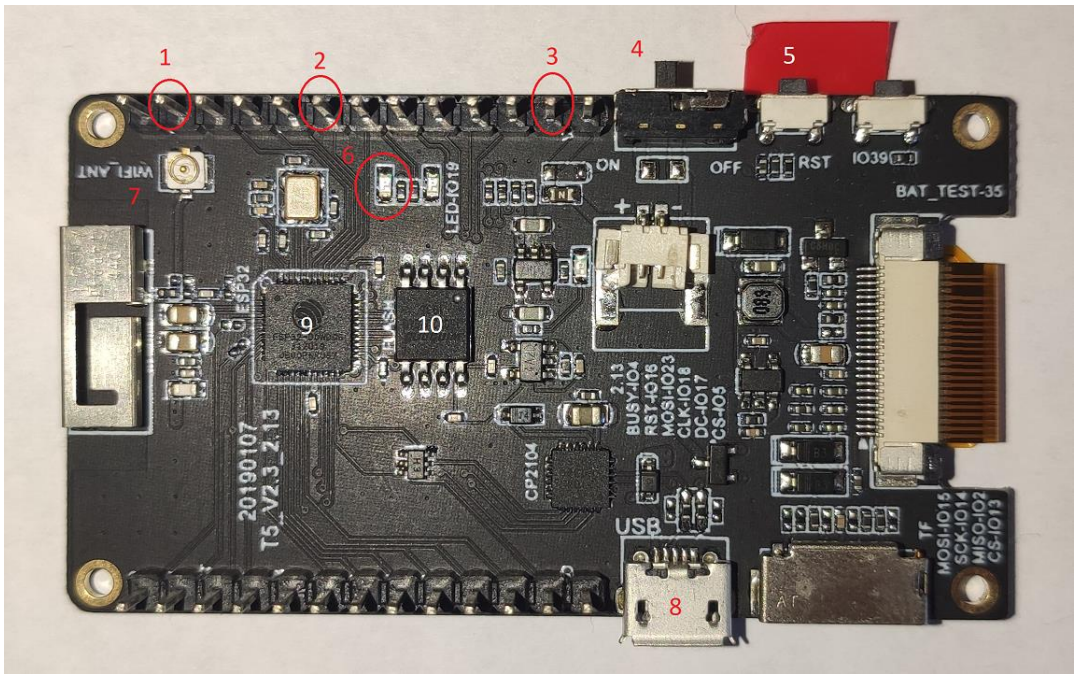
Slika 2. Pojednostavljena arhitektura mikroprocesora

Kao glavne razlike mikrokontrolera u odnosu mikroprocesora se navodi nedostatak RAM i ROM modula kao i nedostatak ulazno izlaznih portova za periferiju kod mikroprocesora. Još jedna od značajnih razlika je cijena izrade. Mikrokontroleri se izrađuju koristeći tehnologiju

komplementarnih poluvodiča od metalnog oksida i iz navedenog razloga su znatno jeftiniji od izrade mikroprocesora koji se izrađuju. Razlika u cijeni se također očituje i u performansama mikrokontrolera u odnosu na mikroprocesore. Taktovi mikrokontrolera se mjere u desecima MHz, dok se taktovi mikroprocesora očituju u vrijednostima od 1 GHz na više. Brzina obrade instrukcija je proporcionalna kompleksnosti zadataka, što znači da su mikrokontroleri u pravilu zaduženi za obradu jednostavnijih zadataka, dok su mikroprocesori zaduženi za obradu nešto više kompleksnih zadataka gdje veće brzine obrade omogućavaju znatnu uštedu vremena i brže rezultate rada. Zadnja značajna razlika između prethodno navedenih mikrokontrolera i mikroprocesora je arhitektura na kojoj se pojedini sustav bazira. U slučaju mikrokontrolera radi se o sustavu baziranom na Harvard arhitekturi gdje su programska memorija i podatkovna memorija međusobno odvojene, dok se u slučaju mikroprocesora navedene memorije nalaze unutar jednog zajedničkog memorijskog modula i takav tip arhitekture se bazira na von Neumannovom modelu. [19]

3.1.6. ESP32 mikrokontroler

ESP32 mikrokontroler je model mikrokontrolera proizvođača Espresiff Systems iz Šangaja, Kine. Predstavljen je 2016. godine kao nasljednik ESP8266 mikrokontrolera od istog proizvođača.



Slika 3. Korišteni ESP32 mikrokontroler

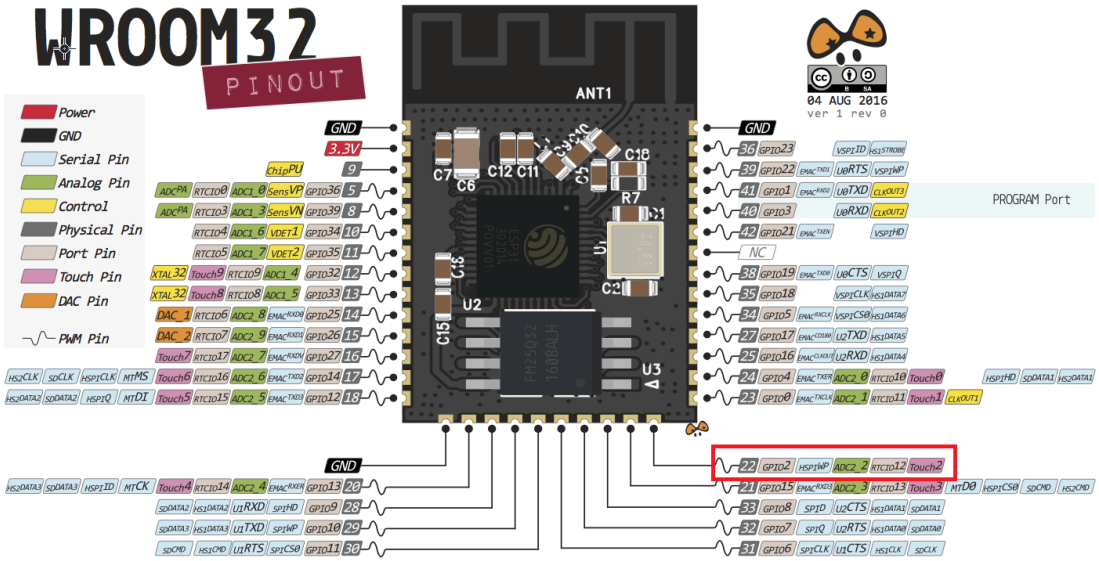
Prikazana slika predstavlja ESP32 mikrokontroler koji je korišten u realizacija projekta. Na slici su označeni brojevi koji su referencirani u tekstu koji opisuje uređaj.

Kao i svaki mikrokontroler, ESP32 se sastoji od pločice na kojoj su postavljene razne komponente, kao što je središnja procesna jedinica koja se kod ovog modela sastoji od dvije 32-bitne jezgre (broj 9 na slici), za razliku od njegovog prethodnika koji je imao samo jednu. Određeni modeli podržavaju do 4 MB radne memorije te duplo više pinova od svog prethodnika, čak 34, iako korišteni model u ovom projektu ima 26 pinova. Svi pinovi služe različitoj svrsi te se mogu tako spajati na različite komponente. U našem primjeru projekta bit će nam potrebna samo nekolicina pinova. Od toga su dva GND pina koji služe za uzemljenje releja gdje će biti potreban samo jedan (broj 1 na slici), tri pina izvora struje od koji su dva 3.3V i jedan je 5V koji služi za napajanje releja (broj 3), te mnogi drugi koji mogu služiti kao ulazno/izlazni pinovi ili samo ulazni i samo izlazni pinovi putem kojih releju dajemo informacije o propuštanju ili nepropuštanju napona. U realizaciji projekta će se koristiti pin 22 kao izvor električne energije (broj 2), odnosno na njega će se prosljeđivati logičku jedinicu ili nulu, s obzirom što se u projektu želi postići. O trenutnom stanju mikrokontrolera govori plava dioda koja svijetli ukoliko je uključen (na slici broj 6), koja se također može i kontrolirati putem skripti, iako to u opisanom projektu nije bilo potrebno. Na svakom mikrokontroleru se može pronaći klizni prekidač koji služi za uključivanje ili isključivanje kontrolera (pod brojem 4). Na korištenom ESP32 mikrokontroleru taj se prekidač nalazi odmah pored jednog jako korisnog gumba, RST (reset) tipke kojom se cijeli mikrokontroler resetira i automatski se pokreće program koji je pohranjen na njemu (broj 5).

Jedna od velikih prednosti mikrokontrolera iz serije ESP u odnosu na Arduino Uno i slične mikrokontrolere jest što svaki model dolazi s ugrađenim Wi-Fi modulom te nije potrebno nikakvo dodatno podešavanje prije korištenja (na slici u lijevom gornjem kutu, pod brojem 7). ESP32, kao nasljednik ESP8266, ide još jedan korak dalje te dolazi s ugrađenom Bluetooth 4.2 tehnologijom, kao i Bluetooth Low Energy verzijom koja u pojedinim slučajima koristi i do 100 puta manje energije nego uobičajeno zbog toga što se podaci prenose samo u određenim trenucima, a ne kontinuirano kao kod uobičajenog bluetootha.

3.1.6.1. Korišteni pinovi

ESP32 mikrokontroler koji se koristi u realizaciji projekt sadrži 26 pinova, koji mogu biti podijeljeni u određene skupine. Kao što je prethodno opisano, postoje pinovi koji služe kao izvori električne energije pomoću kojih napajamo druge elemente potrebne u projektu. Takvih je pinova nekoliko, kao i pinova za uzemljene, no preostaje velik broj pinova koji ima drugu ulogu, za primanje ili slanje podataka putem električnih signala. Kako je u ovom projektu potrebno putem jednog pina slati logičke nule ili jedinice, potrebno je bilo pronaći pin koji će biti u mogućnosti odrađivati tu zadaću. Putem jednog od mnogih internet članaka na tu temu [15] odabran je pin 22, koji osim za prosljeđivanje, može služiti i za primanje signala te je na slici označen crvenom bojom.

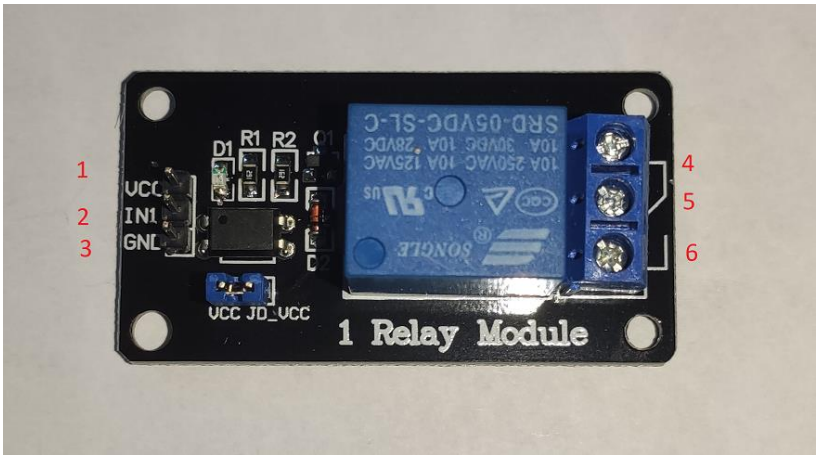


Slika 4. ESP32 Developer board pinovi (izvor: Random Nerd Tutorials)

Korišteni mikrokontroler također sadrži ugrađeni i jednostavni e-paper ekran, koji pomoću prosljeđivanja napona zacrni određene piksele i time prikazuje jednostavnu sliku, no to u ovom projektu nije bilo od važnosti, stoga nije niti korišteno.

3.2. Relej

Sljedeći jako bitan modul je relej, koji propušta električnu energiju uređaju koji je na njega povezan. Kako bi propuštao električnu energiju, potrebno je prvo da on sam dobiva električnu energiju iz određenog izvora, što će mu u ovom projektu pružati ESP32 mikrokontroler. Prije svakog rada s relejom potrebno je naglasiti kako njegovu opasnost i mogućnost uzrokovanja ozljeda ukoliko se osobe s njim ne ponašaju oprezno.



Slika 5. Korišteni relej s označenim ključnim dijelovima

Slika predstavlja relej koji je korišten u realizaciji ovog projekta. Brojevi pored određenih dijelova predstavljaju elemente releja koje se opisuju u poglavlju.

Postoje tri pina putem kojih se relej povezuje na mikrokontroler, a to su VCC, punog naziva voltage common collector (označen brojem 1 na slici) koji predstavlja pin putem kojeg se šalju naredbe releju o obliku logičkih jedinica ili nula, IN1, odnosno Input1 (broj 2 na slici) koji predstavlja pin za napajanje releja gdje se u primjeru napaja relej sa snagom od 5V. Znamenka jedan označava o kojem releju se radi, jer je moguće pronaći modul i s osam releja. Zadnji pin GND, ground (označen brojem 3 na slici) pin, predstavlja uzemljenje. Input1 i GND pinovi su samoobjašnjivi, jer se pinovi istih oznaka nalaze i na mikrokontroleru, dok je za VCC pin potrebno opisati njegovo djelovanje. On na mikrokontroleru mora predstavljati izlazni pin, odnosno pin koji šalje električnu energiju releju i daje mu naredbu u obliku logičke nule ili jedinice, koja mu na taj mu način govori o potrebi za prosljeđivanjem električne energije. Na drugom kraju releja nalaze se tri kontakta, NO (normally open) (na slici broj 6), CO (common) (broj 5) i NC (normally closed) kontakt (broj 4 na slici). Kako bismo upravljali uređajem koji je povezan, potrebno ga je svakako povezati na common kontakt te na NO ili NC kontakt.

Ukoliko uređaj povežemo na normally closed kontakt, strujni krug će se zatvoriti i struja proslijediti na povezani uređaj svaki put kad je relej aktivan, odnosno kada se na VCC pin proslijedi logička jedinica. U suprotnom slučaju, kada se proslijedi logička nula, strujni krug nije zatvoren i električna energija se ne prosljeđuje.

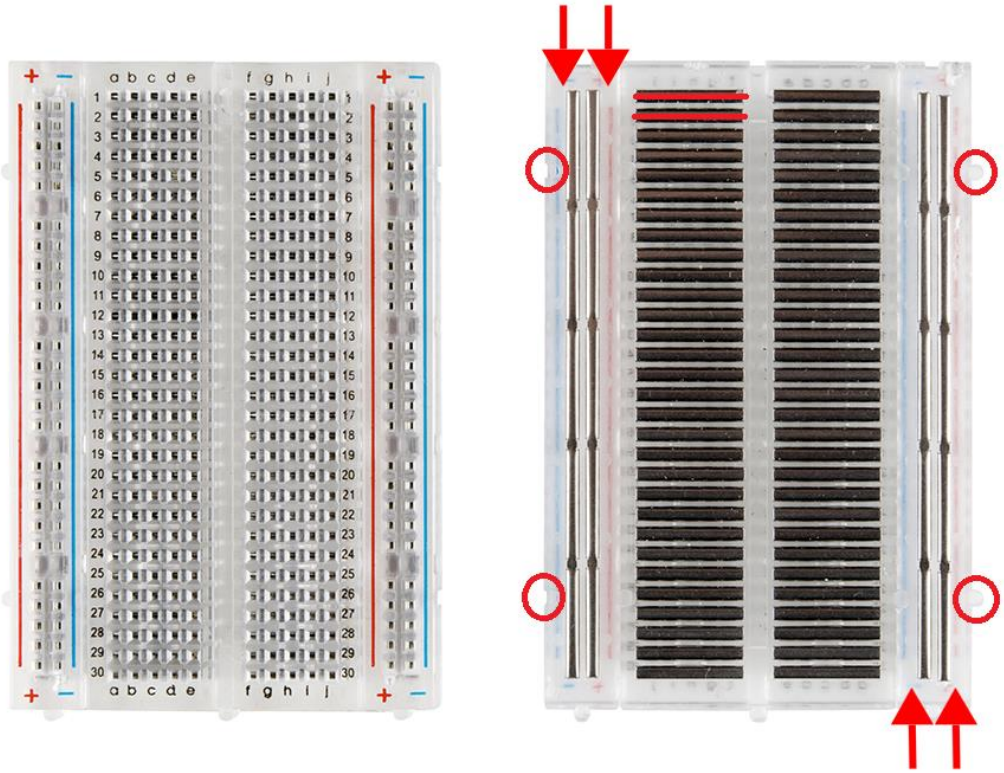
Kod normally open kontakta situacija je suprotna. U njegovom slučaju se strujni krug zatvara u slučaju kad je relej neaktivan, odnosno prosljeđuje se logička nula putem mikrokontrolera. U slučaju kad je proslijeđena logička jedinica, strujni krug nije zatvoren i priključenu uređaj ne dobiva električnu energiju.

Postoje određeni principi povezivanja uređaja na relej, odnosno na koji način ga je najbolje povezati na relej, ovisi o učestalosti njegovog korištenja. Ukoliko će povezani uređaj većinu svog vremena biti uključen, odnosno trebat će mu izvor električne energije, pravilo je da ga se povezuje na normally closed kontakt. U slučaju kad je potrebno samo povremeno prosljeđivanje električne energije najbolja je praksa spojiti uređaj na normally open kontakt zbog toga što će uređaji koji su povezani na pametnu utičnicu samo po potrebni biti uključeni. Upravo taj način korišten je u ovom projektu, uređaj je povezan na common i normally open kontakte čime se prosljeđivanjem logičke nule putem mikrokontrolera zatvara strujni krug, a prosljeđivanjem jedinice strujni krug ne zatvara.

3.3. Breadboard

Jedna od komponenti koja nam je također potrebna u realizaciji projekta je breadboard. To je pločica raznih dimenzija i veličina s mnoštvom isprepletenih električnih žica koja služi za

spajanje određenih pinova mikrokontrolera na razne druge komponente. Naravno, moguće je izravno spojiti pin za izvor struje ili uzemljenje u komponentu kao što je relej, ali zbog toga što će možda kasnije biti potrebno spojiti još neke dodatne komponente, a pinova na mikrokontroleru ima samo određeni broj, bolja je praksa prvotno ih spojiti na breadboard i onda iz breadboard na pojedine komponente.



Slika 6. Prikaz breadobarda (izvor: Sparkfun)

Breadboard je građen kao niz utora koji su međusobno povezani kako bi se što više optimiziralo spajanje pinova. Po svojoj dužini, sa svake strane, ima dvije linije pune utora, označenih s plus i minus (na slici označene sa strelicom). Te vertikalne linije su međusobno sve povezane, odnosno ukoliko umetnemo pin u jednu od njih, možemo smatrati da su sve ostale u toj dužini iste. Ovo je dobra stvar kad imamo više uređaja za koje nam je potrebno uzemljenje ili napajanje. Pomoću samo jednog pina za pojedinu stvar iz mikrokontrolera možemo dobiti uzemljenje ili napajanje za velik broj uređaja koji su povezani na breadboard. [16]

Osim okomitih linija, u sredini breadboarda protežu horizontalne linije cijelom njegovom dužinom (na slici označene crvenom linijom). Svaka linija sadržava utore koje su međusobno povezani, odnosno isto kao i kod vertikalnih, umetanjem pina na jednu možemo dobiti istu električnu energiju i na drugim pinovima u toj dužini. [16]

Dodatne karakteristike koje breadboard u većini slučajeva sadrži su i dodatni elementi s pojedinih strana kako bi se mogli serijski povezati da čine veću cjelinu (na slici označeni

krugom). U tom slučaju se ne prosljeđuje napajanje, stoga je na dodani breadboard potrebno opet spojiti pinove kako bi bio funkcionalan. Određeni modeli na poleđini sadrže ljepljiv dio kako bi se mogli postaviti na bilo koju površinu.

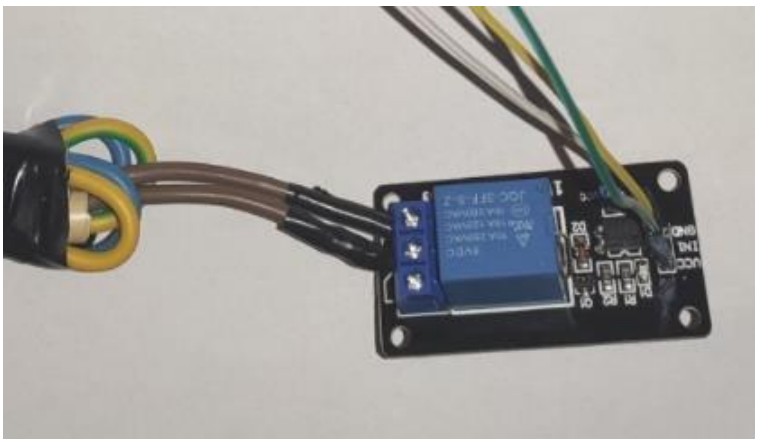
3.4. Povezivanje na utičnicu

Zadnja od hardverskih komponenti potrebnih za projekt je utičnica koja će se kontrolirati putem mikrokontrolera i releja. Za te potrebe, korišten je produžetak utičnice čiji kabel je odvojen i žice su mu izolirane. Kabel se sastoji od tri žice različitih, ali definiranih boja, tako da bilo koja osoba koja mora raditi s žicama zna točno koja što predstavlja:

- Žuto-zelena – Uzemljenje
- Plava – Radna nula
- Smeđa – Fazni vodič

Žice u korištenom produžetku utičnice su izolirane kako bi se mogla izdvojiti smeđa žica, koja je jedina potrebna u realizaciji ovog projekta. Ostale žice su spojene nazad na isti način kako bi kabel mogao normalno nastaviti funkcionirati.

Smeđu žicu s izoliranim krajevima je potrebno spojiti na već opisane hardverske komponente na način da će se kroz nju propuštati električna energija. Kako je prethodno rečeno, na releju se koriste common i normally open kontakti zbog čega je utičnicu potrebno povezati na njih kako bi pri radu releja mogli primati električnu energiju. Potrebno je odvijačem opustiti kontakte te umetnuti krajeve utičnica u njih. Sve bakrene žice moraju biti izolirane i moraju se nalaziti unutar kontakta, jer postoji velika opasnost za ozljedu prilikom rada, ukoliko žica ostane neizolirana. U ovom projektu žice broj bakrenih žica je smanjen, odnosno neke su odrezane kako bi sve mogle stati u potrebne kontakte.



Slika 7. Spajanje releja na utičnicu

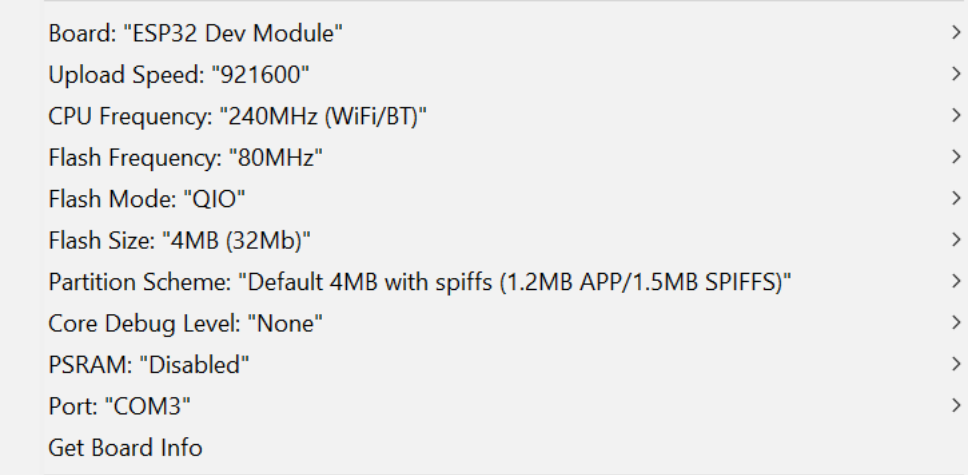
3.5. Arduino IDE

Kao što je u prethodnim poglavljima opisano, program koji se zapiše na ESP32 mikrokontroler ostaje trajno zapisan čak i nakon njegovog ponovnog pokretanja. Za razvijanje skripte koju je potrebno zapisati na mikrokontroler, koristi se Arduino IDE razvojno okruženje. Iako je to razvojno okruženje koje je prvotno nastalo za potrebe Arduino mikrokontrolera, zbog raznih biblioteka koje se za njega mogu preuzeti predstavlja odličan IDE za bilo koje IoT poslove jer podržava raznoliku vrstu mikrokontrolera od velikog broja proizvođača. U ovom projektu jedino će se koristiti ESP32 mikrokontroler zbog čega je potrebno prvotno namjestiti par stvari kako bi se moglo uopće početi raditi i razvijati skripte.

Prva, osnovna i najbitnija stvar koju treba napraviti ukoliko želimo razvijati s ESP32 mikrokontrolerom u Arduino IDE razvojnom okruženju je preuzimanje i instaliranje službene ESP32 biblioteke od Espressif Systems-a. Kako je sam IDE relativno jednostavan, ne posjeduje nikakav početni izbornik ili čak početni zaslon koji se prikazuje prilikom njegovog otvaranja, nego se odmah prikaže zadnja učitana skripta gdje se pri vrhu nalazi niz opcija samog razvojnog okruženja. Nazivi opcija su: „File“, „Edit“, „Sketch“, „Tools“, „Help“. Upravo tu, pod opcijom „Sketch“, odnosno na hrvatskom jeziku „skica“, postoji opcija da se doda biblioteka po želji. Klikom na opciju „Library manager“ otvara se novi prozor u kojemu je moguće preko jednostavne tražilice pronaći željenu biblioteku i jednim klikom instalirati ju u IDE. Nakon instalacije nije potrebno ponovno pokretati okruženje, nego je sve spremno na rad. S time je okruženje dobilo sve potrebne dijelove za razvoj novih skripti za ESP32 mikrokontroler, ali ih je potrebno i namjestiti kako bi se omogućio rad.

Sljedeća kategorija u izborniku, „Tools“, sadržava velik broj opcija za pojedine mikrokontrolere i samo okruženje. Kako bismo bili sigurni da se sve stvari odvijaju kako treba između mikrokontrolera i okruženja, potrebno je odabrati pravilnu ploču u izborniku pod „Board", što predstavlja sami model mikrokontrolera koji ćemo koristiti u razvoju. Iako Arduino IDE sam po sebi ima prethodno instaliran velik broj mogućih Arduino modela, zbog toga što je u ovom projektu korišten ESP32 model ta nam opcija nije bitna te je samo potrebno da odaberemo iz druge kategorije, prvi model s liste, odnosno „ESP32 Dev Module“. U toj listi postoji popriličan broj modela i još se drugi modeli mogu dodati te je početnicima jako zbunjujuće odabrati ispravnu stavku, jer izgledaju dosta slično. Zbog toga je na većini pločica zapisan naziv samog modela na vrlo vidljivo mjesto, a također takvu informaciju je lagano naći i na internetu. ESP32 model koji je korišten u realizaciji ovog projekta imao je dodatan e-paper ekran zalijepljen s jedne strane pločice, zbog čega nije bilo moguće pročitati naziv modela, ali u razgovoru s osobom koja mi je prodala pločicu zaključeno je da se radi o standardnom ESP32 Dev Module-u.

Sljedeća opcija koje se može namjestiti je „Upload speed“ koja predstavlja brzinu prijenosa podataka na sami mikrokontroler. Postoji par opcija i nije od prevelike važnosti koja opcija se namjesti, jer će se u konačnici skripta svakako uploadati na pločicu. Ispod toga, nalaze se opcije kao što su frekvencija središnje procesne jedinice, frekvencija radne memorije, veličina radne memorije, metode dijeljenja podataka, razine javljanja greške i mnoge druge. Ove opcije za ovaj projekt nisu bile mijenjanje, nego su ostavljene na iste postavke kao što su bile prilikom instalacije samog programskog okruženja, te se nije pojavila potreba prilikom razvijanja skripte za njihovim modifikacijama.



Slika 8. Opcije postavljanja mikrokontrolera

Nakon podešavanja tih opcija može se krenuti s razvojem skripte koja se razvija u C++ programskom jeziku. Svaka skripta prilikom kreiranja ima dvije metode, setup() i loop(). Prva metoda, setup se prva poziva prilikom pokretanja programa. To je metoda koja većinom služi za postavljanje određenih stvari koje će nam biti od koristi kasnije u skripti, kao što je povezivanje na internet, postavljanje frekvencije ispisa podataka na ekran i definiranje ulaznih i izlaznih pinova, zbog toga što se odvija samo jednom. Druga metoda, loop, se poziva odmah pri završetku metode setup i cijelo vrijeme se izvršava dokle god se izvodi skripta na mikrokontroleru. U nju je praktičnije postaviti određene stavke kao što su čitanje ulaznih informacija s pinova, ispisivanja trenutnih informacija na ekran zbog same prirode njenog izvođenja, odnosno bez posebnih petlji dobit ćemo kontinuirane informacije o radu mikrokontrolera i pojedinih komponenti.

Na dnu programskog okruženja nalazi se „Log“, odnosno dnevnik zapisa informacija o uspješnosti izvršavanja programa, pogreškama koje se odvijaju prilikom kompiliranja programa i drugo. Na njemu ćemo naći ispis svih grešaka zbog kojih se program ne može uploadati na mikrokontroler, ali i status uploadanja skripte. Same informacije koje se ispisuju na ekran na temelju skripte ne nalaze se u tom zapisu informacija, nego u „Serial Monitor-u“,

za koji postoji posebna tipka u desnom dijelu izbornika, koja se nalazi pod ikonom povećala, ali se može također i uključiti kroz „Tools“ opciju izbornika pod istim imenom. Prvotno Serial Monitor neće ispisivati nikakve podatke, ako nismo odredili frekvenciju s kojom ih on treba ispisivati. Kao što je navedeno, praksa je da se to obavlja najčešće u setup metodi skripte pomoću komande Serial.begin(), gdje metoda kao argument prima frekvenciju ispisa na ekran. Tu istu frekvenciju potrebno je namjestiti i u Serial Monitoru te ukoliko je sve u redu, trebali bismo vidjeti kako se rečenice ispisuju na ekran kad uploadamo određeni program.

Verifikacija i upload programa je jednostavan proces, za koji postoje tipke u gornjem lijevom dijelu ekrana. Prva tipka, verificiraj će samo kompilirati skriptu, no neće ju uploadati na mikrokontroler. Ova opcija se najčešće koristi ukoliko ne želimo da nam se trenutna skripta prestane izvršavati. Druga opcija, uploadanja, radi oba posla, verificiranje i uploadanje skripte na mikrokontroler. Prilikom klika na bilo koju gumb informacije i trenutno stanje skripte i procesa dobivaju se povratne informacije u dnevniku informacija na dnu Arduino IDE-ja te ukoliko imamo nekakvu semantičku grešku u programu bit će ispisana informacija.

Postoji mnogo načina razvijanja skripti za ESP32 mikrokontroler i velik broj biblioteka koje nude širok izbor mogućnosti, zbog čega postoje određene skripte koje mogu poslužiti kao primjer rada ili kako se određeni elementi biblioteke najbolje mogu iskoristiti. Te skripte se nalaze u izborniku pod imenom „File“ i nadalje „Examples“ te su grupirane po bibliotekama koje su ih kreirale radi jednostavnijeg pronalaska. Kako je u jednom od prethodnih koraka objašnjeno preuzimanje potrebnih biblioteka za ESP32 mikrokontroler, s uspješnim instaliranjem svih potrebnih stavki instalirani su i mnogobrojne skripte primjera rada pojedinih komponenti. Ovdje se mogu pronaći razni primjeri kao što su ispisivanje svih dostupnih Wi-Fi mreža putem Wi-Fi modula, korištenje bluetootha 4.2 i povezivanja s određenim elementima, povezivanje na internetsku mrežu, izrada web servera i mnoge druge. Jedan od takvih primjera će biti od velike koristi prilikom povezivanja mikrokontrolera na Firebase bazu podataka, što će biti kasnije objašnjeno. Takve skripte se na svojim izvornim lokacijama programskog okruženja ne mogu izmijeniti, zbog čega se moraju isprva spremiti u posebnu datoteku, nakon čega ih možemo izmjenjivati po volji i prilagođavati našem projektu.

Kako je za realizaciju ovog projekta potrebna Firebase baza podataka na koju se pohranjuju željena stanja pametne utičnice, potrebno je preuzeti i implementirati i tu biblioteku u Arduino IDE. To se događa na identičan način kao i kod preuzimanja biblioteke za ESP32 mikrokontroler, samo što je potrebno u tražilici pronaći „Firebase ESP32 Client“ biblioteku i preuzeti ju. Ta biblioteka dolazi s mnoštvom primjera kodova koji mogu uvelike pomoći u realizaciji ovog projekta. Skripta koja će nam poslužiti je „Stream_callback“, u kojoj se povezuje na Firebase bazu podataka s potrebnim podacima, povezuje na Wi-Fi mrežu s potrebnim podacima i kontinuirano šalju i primaju podaci iz baze. Kako u ovom projektu slanje podataka nije potrebno, nego samo primanje, taj dio se može u potpunosti zanemariti.

3.5.1. Skripta za pametnu utičnicu

Kao što je prije rečeno, svaka se skripta počinje izvršavati pozivom metodi setup, koja se izvršava samo jednom. Kako je za realizaciju ovog projekta potrebno spojiti mikrokontroler na internet te zatim na Firebase bazu podataka, to su prve dvije stvari koje se odvijaju u metodi setup, odmah nakon definiranja frekvencije ispisa podataka na serial monitor. Za povezivanje ESP32 na Wi-Fi potrebno je također koristiti klasu naziva WiFi, što je klasa napisana u programskom jeziku C modificirana za upotrebu u Arduino IDE razvojnom okruženju. Ta klasa sadrži metode spajanja na internet pomoću metode begin u koju je potrebno proslijediti naziv mreže na koju se želimo spojiti i njenu šifru za spajanje. Spajanje se ne događa u istom trenutku, nego je potrebno pričekati određeno vrijeme, zbog čega se koristi while petlja u kojoj se promatra status povezanosti putem ugrađene metode status. Ona vraća element enum niza koji govori o stanju povezanosti na internet, te ukoliko stanje govori da smo povezani, možemo nastaviti dalje s izvršavanjem koda.

```
WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
Serial.print("Connecting to Wi-Fi");
while (WiFi.status() != WL_CONNECTED)
{
  Serial.print(".");
  delay(300);
}
Serial.println();
Serial.print("Connected with IP: ");
Serial.println(WiFi.localIP());
```

Slika 9. Kod povezivanja mikrokontrolera na internet

```
Connecting to Wi-Fi..................
Connected with IP: 192.168.1.10
```

Slika 10. Povezivanje mikrokontrolera na internet

Nakon što se mikrokontroler uspješno spojio na internet, sljedeći korak je spajanje na Firebase bazu podataka. Za to je potrebno koristiti klasu, koja je dobivena iz preuzete Firebase biblioteke za ESP32, naziva FirebaseESP32. Ta klasa sadržava brojne metode od kojih je za povezivanje najpotrebnija metoda begin. Ona zahtjeva dva parametra, Firebase host i Firebase auth. Prvi parametar govori na kojoj internetskoj adresi se nalazi baza podataka, kako bi joj se moglo pristupiti, dok je drugi parametar autorizacijski token pomoću kojeg će se pristupiti bazi podataka. On se nalazi na web stranici od baze podataka u postavkama projekta, gdje u svakom trenutku baza podataka mora imati najmanje jedan kreirani token za pristup.

Za potrebe ovog projekta korišten je taj isti generiran od strane baze podataka, no moguće je kreirati i dodatne ukoliko se za to ukaže potreba.

Nakon što se mikrokontroler uspješno poveže na bazu podataka, može se početi pratiti stream podataka putem metode beginStream. Kod te metode je potrebno kao parametar navesti putanju u bazi podataka, odnosno granu na kojoj će se promatrati stream podataka. Ta putanja je deklarirana kao globalna varijabla u projektu kako bi joj se mogli pristupiti po potrebi. BeginStream metoda vraća boolean vrijednost o uspješnosti upostavljanja streama, odnosno ukoliko se vrati true, stream je uspješno postavljen, a ukoliko vrati false, dogodila se određena pogreška koja se u ovom projektu ispisuje na serial monitor.

```
Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);
Firebase.reconnectWiFi(true);

if (!Firebase.beginStream(firebaseData1, streamPathString))
{
  Serial.println("------------------------------------");
  Serial.println("Can't begin stream connection...");
  Serial.println("REASON: " + firebaseData1.errorReason());
  Serial.println("------------------------------------");
  Serial.println();
}
```

Slika 11. Kod povezivanja mikrokontrolera na Firebase bazu podataka

Sljedeća stavka skripte je Firebase metoda setStreamCallback. U njoj se prosljeđuju nazivi dviju metoda koje će se pozivati u slučaju određenih stream događaja. Prva metoda je streamCallback, koja će se pozivati svaki put kada se na definiranoj putanji dogodi promjena vrijednosti podataka, a druga streamTimeoutCallback, koja će se pozivati svaki put kada se dogodi timeout kod streama. Ta metoda u ovom projektu nije od prevelike važnosti, zbog toga što nije potreban nikakav poseban način rukovanja timeoutom, zbog čega se informacija samo ispisuje na serial monitor i stream se nastavlja. Metoda streamCallback, s druge strane, daje informacije o promjeni stanja baze podataka, zbog čega je ona iznimno potrebna i predstavlja glavni dio napisane skripte. Parametar metode je StreamData, objekt koji sadržava sve podatke o promjeni u bazi podataka, potpunoj putanji, koji tip podataka je nastao i njegovu vrijednost. Zbog toga što putem Google Home akcije u Firebase bazu podataka postavimo cjelokupan JSON zapis, a putem Flutter aplikacije samo postavljamo boolean varijablu, potrebno je prvotno unutar metode postaviti grananje kako bi se znalo koji tip podataka treba obraditi. To se obavlja koristeći metodu dataType koja za povratnu informaciju vraća tip podataka koji je postavljen u bazu podataka u obliku stringa. Na taj se način zna hoće li se pozvati metode controlBooleanData ili controlJsonData. To su dvije metode kreirane u ovoj skripti koje upravljaju podacima koji dolaze, jedna radi s boolean, a druga s JSON tipovima

podataka. Ako je tip podataka druge vrste, ispisuje se poruka o tome kako takav tip podataka u kreiranoj skripti nije podržan i metoda završava s radom, te se čekaju novi podaci.

```
void streamCallback(StreamData data)
{
  Serial.println(data.dataType());

  if (data.dataType() == "boolean") {
      controlBooleanData(data);
  } else if (data.dataType() == "json") {
      controlJsonData(data);
  } else {
    Serial.print("Unsupported data format: ");
    Serial.println(data.dataType());
  }
}
```

Slika 12. Kod za obradu dobivenih podataka iz Firebase baze podataka

ControlBooleanData je metoda u kojoj je potrebno preuzeti vrijednost podataka iz baze podataka koja je dobivena kroz objekt StreamData i postaviti ju na vrijednost pametne utičnice. Iako se sama vrijednost može odmah pročitati, u skripti je napravljeno par provjera pomoću if uvjetovanja kako bi se sa sigurnošću moglo reći da se radi o željenom ažuriranju podataka u bazi. Ukoliko su svi uvjeti zadovoljeni, poziva se metoda controlSwitch.

ControlJsonData je druga vrsta metode, koja služi za rukovođenje JSON promjena u bazi podataka. Metoda ima isto uvjetovanje kao i controlBooleanData, te se zatim poziva decodeSwitchControl metoda. Ta metoda kreira FirebaseJson objekt iz navedenog StreamData objekta, kako bi se pomoću petlje moglo proći kroz cjelokupan objekt i djelovati s jednim po jednim podatkom. Pomoću metode iteratorBegin klase FirebaseJson dobivamo broj zapisanih JSON vrijednosti, koji koristimo kako bi se pomoću for petlje prošlo kroz sve njih. Počevši od nule, koristimo metodu iteratorGet te dobivamo vrijednosti JSON atributa u obliku „ključ: vrijednost“, te ukoliko je dobiveni ključ isti kao i onaj koji trebamo, prosljeđuje se vrijednost kao parametar metodi controlSwitch.

```
void decodeSwitchControl(StreamData &data) {
  FirebaseJson *json = data.jsonObjectPtr();
  size_t len = json->iteratorBegin();

  String key, value = "";
  int type = 0;
  for (size_t i = 0; i < len; i++)
  {
    json->iteratorGet(i, type, key, value);

    if (key == switchControlKeyString) {
        bool boolValue = checkBooleanFirebaseValue(value);
        controlSwitch(boolValue);
    } else {
        printOutUnknownKey(key);
    }
  }
  json->iteratorEnd();
}
```

Slika 14. Kod za ažuriranje mikrokontrolera s podatacima pristiglim putem Google Home aplikacije

ControlSwitch metoda je jednostavna metoda koja služi kako bi se pomoću proslijeđene vrijednosti promijenilo trenutno stanje pina 22, pina koji je postavljen kao izlazni pin. Ukoliko je proslijeđena vrijednost true, tada postavljamo logičku jedinicu na izlazni pin te ukoliko je false, postavljamo logičku nulu. S time uvjetujemo zatvaranje strujnog kruga u releju i uključujemo, odnosno isključujemo pametnu utičnicu.

```
void controlSwitch(bool value) {
  if (value) {
      digitalWrite(SWITCH_PIN, LOW);
  } else {
      digitalWrite(SWITCH_PIN, HIGH);
  }
}
```

Slika 13. Kod za slanje željenog stanja pametne utičnice mikrokontroleru

Verificiranjem i uploadanjem ove skripte na ESP32 mikrokontroler uspješno je napravljena skripta za uključivanje i isključivanje pametne utičnice, te je osim toga potrebno još kreirati Google Home akciju i Flutter mobilnu aplikaciju koje će postavljati vrijednosti na Firebase bazu podataka.

3.6. Spajanje komponenti

Nakon što su prethodna poglavlja opisala sve potrebne komponente pametne utičnice, ovo poglavlje će opisati na koji točno način je sve spojeno. Prvotna stvar koju je potrebno

napraviti je uključiti ESP32 mikrokontroler u struju, a to radimo putem micro USB kabela koji je u realizaciji ovog projekta spojen. Odmah pri dobivanju električne energije, plava dioda počinje svijetliti i na taj način se prepoznaje da ESP32 radi. Kako korišteni model ima 24 pina, potrebno je znati točno mjesto na koje pojedini pin treba spojiti kako bi se uspješno mogli napajati druge komponente projekta, što činimo putem breadboarda. Iako se projekt može realizirati i bez toga, zbog jednostavnijeg ponovnog spajanja i preglednosti spajamo prvo ESP32 na breadboard, a onda breadboard relejom. Samo spajanje se odvija pomoću kratkih bakrenih žica, koje su s jedne strane imaju utore, a s druge strane priključke. Stranu s utorima spajamo u mikrokontroler i pojedine priključke na slobodne vertikalne linije u breadboard. Kao što je i prije opisano, na taj način spajanje jednog pina, dobijemo taj isti pin cijelom dužinom u breadboard, što nam olakšava spajanje i prespajanje komponenata. Za realizaciju ovog projekta su potrebna tri pina, GND pin, koji služi za uzemljenje releja, pin koji služi za 5V izvor električne energije za napajanje releja i jedan izlazni pin kojim će se slati naredbe releju u obliku logičke nule ili jedinice, gdje je za potrebe ovog projekta korišten pin 22. Kao što je prethodno opisano, te pinove spajamo redom na relej, na pinove GND, IN1 i VCC. Relej također sadržava jednu diodu kojom označava da ima napona i koja, ako se sve pravilno spoji, svijetli. Na drugom kraju releja, potrebno je spojiti smeđi kabel utičnice na kontakte CO i NO. Spajanjem utičnice na te kontakte, relej će proslijediti električnu energiju kada se pošalje putem pina 22 logička nula, zbog toga što će se tek onda strujni krug zatvoriti. U slučaju kad se šalje logička jedinica, uređaj koji se nalazi na utičnici izgubit će izvor električne energije. Zbog toga što se događa zatvaranje i otvaranje strujnog kruga, relej također napravi i određeni zvuk prilikom promjene stanje, što može poslužiti kao još jedna indikacija o radu releja.

4. Flutter aplikacija

U novije vrijeme, pojavljuje se sve veća potreba za razvojem mobilnih aplikacija koje će olakšavati, pojednostavljivati i automatizirati svakodnevne životne procese ljudi koji ih koriste. Te aplikacije bi u idealnoj situaciji trebale biti dostupne što većem broju ljudi, no to u određenim slučajevima, što zbog dostupnih resursa ili znanja nije moguće napraviti. Glavni razlog tome se nalazi u problemu raznovrsnosti operacijskih sustava, gdje se u prvom planu nalazi Google sa svojim operacijskim sustavom Android te Apple sa svojim operacijskim sustavom iOS. Oni pokrivaju veliku većinu sustava koje možemo pronaći na današnjim

mobilnim uređajima, zbog čega je najbitnije razvijati aplikacije za oba sustava, kako bi ju što više ljudi imalo priliku koristiti.

Slika 15. Programski jezik Flutter (izvor: Venturebeat)

Iako su u to vrijeme već postojala određena rješenja na tržištu, poput alata Xamarin ili React Native, Google je 2015. godine na svojoj godišnjoj programerskoj konferenciji objavio kako razvija novi alat za razvoj softvera, nazvan Flutter. Objavljen 2017. godine, Flutter je open-source alat za razvoj programa koji je zbog potpore Googlea stekao veliku popularnost i odmah postao jedan od vodećih alata za razvoj aplikacija na više platformi.

4.1. Karakteristike Fluttera

Mnoge su karakteristike alata koje je vrijedno spomenuti i uzeti u obzir prije samog razvoja aplikacija. Prva i najbitnija je mogućnost razvoja aplikacije na jednom mjestu koja će biti u mogućnosti izvršavati se na oba mobilna operacijska sustava, što je i prvotan razlog razvoja samog alata. Sva potrebna logika oko operacijskih sustava i posebnih karakteristika na koje bi razvojni programeri obraćali pozornost da razvijaju zasebne aplikacije obavljaju se u pozadinskom dijelu Fluttera te na njih nije potrebno obraćati pozornost prilikom razvoja i pokretanja aplikacija.

Sljedeća bitna funkcionalnost je upotreba widgeta. U drugim jezicima i razvojnim alatima, postoje različiti principi razvoja koji se koriste i različite strukture, kao što su pogledi, kontroleri, layouti i mnoge druge stvari. U Flutteru se sve pojednostavljuje na razinu widgeta gdje se pomoću njega može definirati [1]:

- Strukturni element, poput gumba ili menua
- Stilski element, poput fonta ili boje
- Određeni element layouta, poput margina

Putem widgeta se također kreira i hijerarhija bazirana na kompoziciji. Svaki widget može sadržavati druge, na koje prosljuđuje svoja određena svojstva. Na taj način nije potrebno implementiranje svojstava za svaki widget, što pomaže pri kreiranju boljeg korisničkog sučelja.

Widgeti mogu biti sa stanjem ili bez stanja, odnosno mogu sadržavati određene informacije i svojstva koje im govore o potrebi za njihovom promjenom kroz životni ciklus aplikacije. Svaki zaslon ili element ekrana se ažurira zamjenom starog widgeta koji se prisutan na ekranu s novim. Widgeti s čuvanjem stanja (stateful) mogu to samostalno učiniti u određenom trenutku, odnosno postaviti novi widget po potrebni, dok oni koji ne čuvaju stanje (stateless) to ne čine bez vanjskog utjecaja [2].

Velik broj widgeta već postoji unutar alata za razvoj programa, koji omogućavaju raznolik odabir. Postoje dvije glavne skupine ikona, a to su Material widgeti koji su skloniji ikonama koje se mogu pronaći na Android operacijskim sustavima te Cupertino widgeti, koji su pretežno za iOS sustave zbog čega su dobile ime, prema gradu u kojem se nalazi Appleov kampus. Osim postojećih widgeta, zbog toga što je Flutter open-source alat, uvijek postoji mogućnost za razvojem novih rješenja po potrebi, koja se mogu implementirati u projekt bez ikakvih problema, ukoliko prate konvenciju rada alata.

Važno je također naglasiti kako Flutter omogućava izvrsne performanse kako kod procesa razvoja aplikacije tako i u samom izvršavanju iste. Prvi razlog tome je taj što se za razvoj koristi programski jezik Dart, o kojemu će biti nešto više riječi u nastavku ovog rada. Dart programski kod se kompajlira izravno u nativni kod. Kao drugi razlog možemo navesti korištenje prethodno navedenih widgeta, tako da ne postoji potreba za korištenje widgeta izvornog proizvođača pojedinog operativnog sustava.

Još jedna od prednosti Fluttera, s obzirom na korištenje vlastitih widgeta, je da se kao rezultat korištenja istih javlja znatno manji broj grešaka vezanih uz kompatibilnost. To znači da razvojni programeri neće imati probleme s različitim verzijama pojedinih operativnih sustava, uključujući najnovije verzije istih. Google uvelike koristi Flutter u vlastitom razvoju aplikacija i kao takav razvojni alat ima veliki prioritet za ažuriranje i praćenje najnovijih verzija operativnih sustava za koje se koristi pri razvoju aplikacija.

4.2. Dart

Okosnica Flutter alata za razvoj aplikacija je programski jezik Dart. To je objekto orijentirani, klasno baziran jezik predstavljen 2011. godine te objavljen 2012. godine. Razvijen od strane Googlea, Dart predstavlja programski jezik s kojim se moguće izrađivati aplikacije za mobitele, web preglednike, samostalne aplikacije i mnogo drugih stvari. Iako je prvotno bio zamišljen kao programski jezik koji će biti direktno integriran unutar Googleovog web preglednika, Google Chromea, ta ideja nikada nije zaživjela zbog toga što ostali web

preglednici na tržištu nikada nisu prihvatili tu ideju [4]. Stoga je Google za Darta razvio kompiler, imena dart2js, koji omogućava pretvorbu koda zapisanog u Dart programskom jeziku u JavaScript, kako bi se mogao izvršavati na svim web preglednicima.

S novijim verzijama jezika razvijen je dart2native kompiler koji služi za izradu aplikacija na velikom spektru uređaja osim web preglednika, uključujući i samostalne aplikacije na Linux, Windows i drugim operativnim sustavima. Kako bi se izrada samostalnih aplikacija mogla realizirati, bilo je potrebno ugraditi i virtualni stroj (eng. *virtual machine*) putem kojeg će se moći izvršavati naredbe, izvoditi skripte i programi i drugo. [5]

Na ovaj se način izrađuju samostalne izvršne datoteke, no zbog kompleksnosti pokretanja nisu multi platformske, nego se na svakoj platformi moraju zasebno kreirati putem kompilera.

4.2.1. Just-in-time i Ahead-of-time kompiliranje

Osim dart2js i dart2native kompilera, programeri imaju opciju i odabira dvije vrste kompiliranja koje utječu na samu brzinu izrade i izvođenja aplikacije. Ove dvije vrste razlikuju se u načinu rukovođenja predmemorije unutar kompilera, odnosno vremenom zapisivanja podataka u nju.

Prva vrsta je just-in-time kompiliranje, gdje se kao što ime govori, samo kompiliranje napisanog koda odvija za vrijeme izvođenja aplikacije. Ovaj princip može rezultirati bržim pokretanjem aplikacija, no prilikom izvođenja može dovesti do opadanja brzine zbog potrebe za pretvorbom napisanog koda u izvršnu datoteku. Kompiler mora svaku liniju koda prevoriti u izvršni kod, za što je potreban određeni dio vremena. Ukoliko se aplikacija razvija na objekto orijentirani način, tada će postojati određene metode koje se pozivaju više puta, a djeluju s jednakom svrhom, bez ikakve promjene. Kompiler će nakon određenog broja puta shvatiti kako nije potrebno ponovno kompilirati taj dio koda, nego ga je moguće kompiliranog zapisati u predmemoriju odakle se onda po potrebi poziva i time ubrzava program. S time dobivamo da aplikacija kroz duže vrijeme izvođenja dobiva na brzini, odnosno pojedini dijelovi koda se mogu odmah koristiti, bez potrebe za dodatnim kompiliranjem. Ova metoda kompiliranja se koristi prilikom razvoja aplikacija i programa jer se na taj način ubrzava razvoj, zbog konstantnog pisanja koda i testiranja napisanog. [6]

S druge strane, postoji ahead-of-time kompiliranje napisanog koda, koje djeluje na način da se određeni dijelovi koda kompiliraju prije samog početka izvođenja aplikacije. Taj izvršni kod se zatim zapisuje direktno u predmemoriju, zbog čega je potrebno određeno vrijeme da se obavi kompiliranje, ali rezultira s bržim pokretanjem same aplikacije. Također, brzina njenog izvođenja je konzistentnija, bez pojedinih opadanja u brzini prilikom pristupanja određenim dijelovima aplikacije. Ova metoda kompiliranja se koristi prilikom izdavanja aplikacije u Google Play Storeu ili Apple Storeu zbog svoje konzistentnosti u brzini i stabilnih performansi prilikom izvođenja aplikacija i programa. [6]

Slika 16. Programski jezik Dart (izvor: Dart.dev)

4.2.2. Hot reload

Jedna od funkcionalnosti koja odvaja Dart od ostalih programskih jezika, a time i alat Flutter koji ju koristi je mogućnost izvođenja hot reloada, odnosno brze promjene i ažuriranja aplikacije koja se razvija. Kako je u prethodnom poglavlju rečeno, dart2native kompiler sadrži virtualni stroj putem kojeg se izvršavaju pojedine operacije. Taj virtualni stroj je prisutan u Flutteru i prilikom pokretanja aplikacije, ona se zapisuje u njega te se pomoću njega i izvodi. Pri svakoj sljedećoj izmjeni aplikacije, ne događa se cjelokupna njena zamjena u virtualnom stroju, nego samo promijenjenih dijelova, pri čemu se također ne gubi ni trenutno stanje rada aplikacije. Ta mogućnost uvelike ubrzava rad razvojnim programera zbog toga što su promjene vidljive u najkraćem mogućem vremenu i trenutno stanje aplikacije nije izgubljeno, nego se jednostavno može nastaviti testiranje i rad nad njom. Druga opcija koju je potrebno spomenuti je hot restart. Ona djeluje na sličan način kao i hot reload, ali s razlikom da se ne čuva trenutno stanje pokrenute aplikacije, nego se ona ponovno pokreće. Trenutno Flutter Web podržava samo hot restart princip, dok hot reload još uvijek nije implementiran. Zadnja opcija koja je dostupna je full restart, ponovno pokretanje aplikacije, gdje se ona učitava u virtualni stroj i nakon toga se događa njeno pokretanje. [7]

4.2.3. Postavljanje konekcije na Firebase

S obzirom da je za realizaciju ove aplikacije potreban pristup Firebase bazi podataka, potrebno je putem Firebase konzole i internet preglednika registrirati aplikaciju koju ćemo koristiti. To je proces u par koraka koji počinje s klikom na dodavanje nove Android ili iOS aplikacije na Firebase stranici na razini projekta. Za svaku aplikaciju potrebno je kreirati jedinstveni ID paketa koji će djelovati kao identifikacijska oznaka aplikacije. Taj ID će biti vidljiv i na početnom zaslonu projekta u Firebaseu zbog čega je najbolja praksa kreirati određeni smisleni naziv koji će odražavati smisao aplikacije. Taj ID mora sadržavati više riječi, koje su međusobno odvojene točkom.

Nakon klika na gumb da se aplikacija registrira, Google Firebase kreira JSON datoteku naziva google-services.json koja se mora preuzeti i postaviti u Flutter projekt na razini aplikacije. Ta JSON datoteka sadrži brojne podatke pomoću kojih se pristupa Firebase bazi podataka u pozadini aplikacije, bez potrebe za dodatnim pisanjem koda. U prošlim poglavljima je opisano kako se za potrebe pristupa bazi podataka pomoću ESP32 mikrokontrolera i ESP32 Firebase Client biblioteke treba iz postavki baze preuzeti autorizacijski token. Upravo ti podaci su pristupni u preuzetoj JSON datoteci, kao i mnogobrojni drugi koji su potrebni za nesmetano povezivanje mobilne aplikacije i Firebasea.

Sljedeća stavka koja je potrebna za realizaciju jest uključivanje Firebase SDK u postojeći Flutter projekt. Tu je potrebno dodatni određene parametre u datoteke postavke aplikacije, ali i cjelokupnog projekta. Na razini projekt potrebno je navesti kako će se koristi plugin Google servisa, dok se na razini aplikacije navodi koja verzija će se koristiti. Nakon toga je potrebno sinkronizirati podatke te ukoliko su svi koraci ispravno praćeni, prema dokumentaciji se ne bi trebala dogoditi nikakva pogreška. Potrebno je napomenuti kako se nakon ovog koraka u pokušaju realizacije projekta događala pogreška u pčitavanju plugina, koja je riješena implementiranje sljedećeg bloka koda u datoteci postavke projekta koji je pronađen na StackOverflow stranici [22]

```
def flutterRoot = localProperties.getProperty('flutter.sdk')
if (flutterRoot == null) {
   throw new GradleException("Flutter SDK not found. Define location
   with flutter.sdk in the local.properties file.")
}
def flutterVersionCode =
localProperties.getProperty('flutter.versionCode')
if (flutterVersionCode == null) {
flutterVersionCode = '1'
}
def flutterVersionName =
localProperties.getProperty('flutter.versionName')
if (flutterVersionName == null) {
flutterVersionName = '1.0'
}
```

Slika 17. Kod za omogućivanje rada s Firebaseom u Flutter aplikaciji

Nakon što je ovaj korak realiziran, zajedno s navedenim problemom i njegovim rješenjem, pokretanjem aplikacije trebali bismo dobiti jednostavan zaslon koji prikazuje poruku „Hello World“. Čitajući „Log“ poruke, odnosno dnevnik zapisa informacija o samom izvođenju aplikacije i programa, ne bi se trebale prikazivati nikakve poruke greške u vezi Firebase baze podataka i aplikacije, što označava da je s izvođenjem sve u redu.

4.2.4. Kreiranje zaslona

Nakon što je u projektu uspješno uspostavljena sva pozadinska logika u vezi rada s Firebase bazom podataka, može se početi raditi i sama aplikacija. Za potrebe rada ove aplikacije, zbog toga što trebamo imati stalan pristup podacima iz baze podataka, potrebno je uključiti dodatne biblioteke koji sadržavaju Firebase objekt. Potrebno ih je implementirati u pubspec.yaml datoteci pod dependencies:

```
dependencies:
  flutter:
    sdk: flutter
  firebase_database: ^3.1.3
  firebase_core: ^0.4.0+9
```

Slika 18. Kod za postavljanje Firebase SDK u Flutter aplikaciji

Također, potrebno je i u glavnu datoteku naziva main.dart navesti:

Aplikacija je zamišljena na jednostavan način, te se sastoji od dva widgeta, jednog zapisa koji prikazuje trenutno stanje pametne utičnice, odnosno je li upaljena ili nije te drugi widget, gumb pomoću kojeg će se mijenjati trenutno stanje utičnice. Zbog toga što se ovdje

```
import 'package:firebase_database/firebase_database.dart';
```

Slika 20. Kod za korištenje Firebase paketa u Flutter aplikaciji

radi o kontinuiranom primitku podataka, kako bi se moglo ispisivati stanje na ekran, potrebno je implementirati određenu vrstu widgeta koja će to omogućavati, a to je u ovom slučaju StreamBuilder. Kao što i ime govori, radi se o widgetu koji se sastoji od dva dijela, streama i buildera. Prvi dio, stream, odnosi se na izvor podataka na kojem će se promatrati promjena, što je u našem slučaju Firebase baza podataka te točnije grana pametne utičnice. Tu lokaciju referenciramo pomoću Firebase objekta koji je Singleton i navodimo punu putanju do mjesta u bazi gdje se treba promatrati promjena.

```
      stream:
FirebaseDatabase.instance.reference().child("1").child("OnOff").ch
ild("on").onValue
```

Slika 19. Kod za dobivanje novih vrijednosti iz baze podataka u Flutter aplikaciji

Drugi dio widgeta, builder, zadužen je za samo kreiranje widgeta koji će se prikazivati na ekran. Builder metoda se poziva svaki put kada se dogodi određena promjena u bazi podataka, stoga djeluje na sličan način kao i callback metoda u skripti namijenjenoj za ESP32 mikrokontroler. U tom dijelu se događa i glavna logika postupanja s podacima koji stignu. Kako se u ovom projektu radi o jednostavnijem tipu aplikacije, ta logika nastoji samo preuzeti novo zapisanu vrijednost iz baze podataka i na temelju nje kreirati widgete za prikaz na zaslonu. Prvi widget je samo jednostavni ispis podataka koji je dobiven u callback metodi. Ukoliko se putem callback metode dobije podatak da je pametna utičnica uključena, tada se na ekran ispisuje da radi, odnosno ukoliko se dobije podatak da je isključena tada na ekran ispisuje suprotna tvrdnja. Taj se zapis kreira koristeći Center widget, gdje je socketCurrentState string s tvrdnjama „on“ i „off“:

```
      RaisedButton(
            onPressed: postSocketValue,
            child: const Text(
            'Change socket state',
            style: TextStyle(fontSize: 30)
            )
      )
```

Slika 21. Kod za prikazivanje trenutnog stanja pametne utičnice

Drugi widget koji se koristi je RaisedButton. Taj widget označava običan gumb koji ima jednostavan string „Change socket state“ i na kojem je postavljeno da se prilikom pritiskanja pozove metoda koja će zapisati promjenu u bazu podataka:

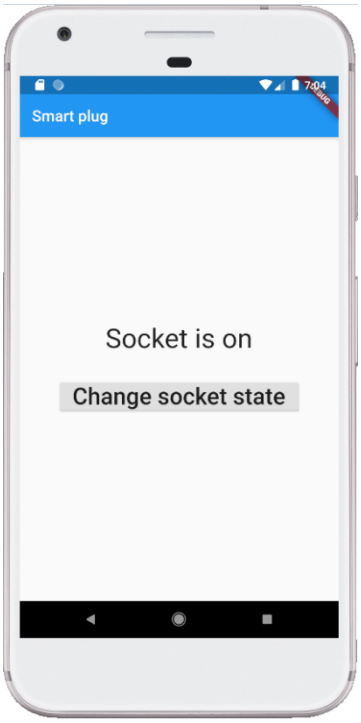
```
      Center(
            child:
            Text("Socket is " +
socketCurrentState,
            style: TextStyle(fontSize: 35))
      )
```

Slika 22. Kod za gumb mijenjanja stanja pametne utičnice

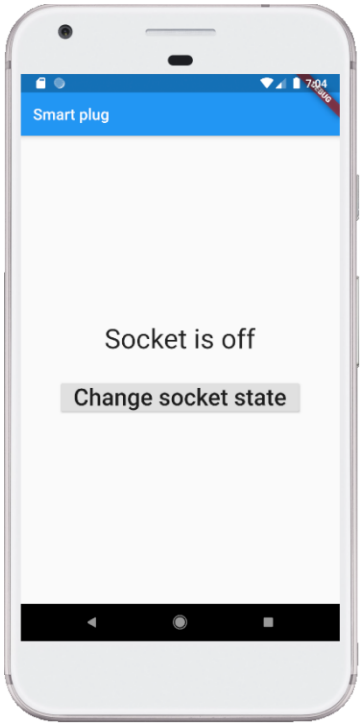
Metoda koja se poziva je postSocketValue i ona sadržava postavljanje varijable na true ili false i njeno zapisivanje u bazu podataka:

```
    void postSocketValue() {
          bool value = true;
          if (socketOnState) {
                value = false;
          }
          FirebaseDatabase.instance.reference().chil
    d("1").child("OnOff
          ").child("on").set(value);
      }
```

Slika 23. Kod za metodu mijenjanja stanja pametne utičnice



Slika 24. Uključena pametna utičnica u Flutter aplikaciji



Slika 25. Isključena pametna utičnica u Flutter aplikaciji

Prilikom ažuriranja stavki u Firebase bazi podataka potrebno je navesti punu putanju do lokacije koju želimo ažurirati. Zbog toga što je baza strukturirana kao stablo, gdje svaki čvor može sadržavati neograničen broj djece, potrebno je referencirati pojedinu granu s metodom child gdje u parametru navodimo naziv grane. Nadalje, metodom set kreiramo i ažuriramo vrijednosti u bazi. Ukoliko se ne navede ispravna putanja, Firebase baza podataka će spremiti traženu vrijednost na novu lokaciju umjesto da ažurira trenutnu vrijednost, te će vratiti poruku kako je operacija prošla u redu, zbog čega je teže shvatiti i ispraviti pogrešku, ukoliko dođe do nje. Također, zbog toga što u bazi nije potrebno prvotno definirati koji tip podataka će se spremati na koje mjesto, kao što je slučaj kod drugih baza, na primjer MySQL, vrlo je bitno da se ažurira uvijek s boolean tipom podataka jer se upravo taj tip očekuje u mikrokontroleru, Nakon što se ta promjena zapiše u bazu podataka, događa se callback streamBuildera te se ažurira trenutno stanje widgeta na glavnom zaslonu.

5. Google

Od početka interneta, postoji iznimno malo kompanija koje su imale tako velik utjecaj na svakodnevni život pojedinca, gdje je na vrhu te liste zasigurno Google. To je jedna od najvećih tehnoloških tvrtki, koja spada u „Big Four“ zajedno s Appleom, Amazonom i

Microsoftom. Postoji mnoštvo stvari koje Google razvija, bio to pretraživač putem kojeg možemo pretražiti bilo koji kutak interneta ili cjelokupan cloud servis koji koriste milijuni za sigurno pohranjivanje svojih datoteka, Google oduvijek ima jedan cilj, a to je poboljšati i pojednostaviti život pojedinca na što više načina.

2016. godine u upotrebu je pušena Google Assistant aplikacija, koja kao što ime govori, predstavlja virtualnog asistenta koji se može upravljati glasom ili unosom naredbi putem ekrana, a u stanju je prilagoditi se svakom korisniku po potrebi. U početku je asistent mogao obavljati osnovne funkcije, kao što su otvaranje aplikacija putem glasovnih naredbi i odgovaranja na jednostavne upite kao zamjena za Google tražilicu. Sa širenjem na sve više klijenata i na sve veći broj uređaja, kao što su pametni satovi pa čak i osobna računala, pojavila se potreba za odrađivanjem kompleksnijih aktivnosti, kao što je vođenje razgovora s pamćenjem prethodnih upita i odgovora, upravljanje većim brojem aplikacija na detaljnije i interaktivnije načine. Također, sve veća integracija svakodnevnih uređaja s internetom predstavljala je još jedan spektar koji je Google htio pokriti sa svojim asistentom, zbog čega je pred kraj 2016. godine objavljena „Actions on Google“, platforma za razvoj osobnih naredbi za Google asistenta. Na taj način bilo koja tvrtka ili osoba može razviti svoju pametnu akciju koju će prilagoditi svojim potrebama i intergrirati ju s raznim uređaja ili mikokontrolerima po potrebi. Tako kreirana akcija se može proslijediti Googleu na provjeru te ukoliko zadovolji sve njihove definirane standarde, može biti integrirana direktno u Google Assistanta, kako bi ju moglo koristiti bilo koje osobe koje imaju tražene uređaje. [10]

Od početka uporabe „Actions on Google“ platforme za razvoj osobnih naredbi, razne tvrtke i poduzeća su razvile više od 175 naredbi, uključujući Uber, Bolt, tvrtke za narudžbu hrane i mnoge druge. [11]

5.1. Actions konzola

Prva stvar koja je potrebna imati za rad na Google Actionsu je Googleov korisnični račun, pomoću kojeg je potrebno napraviti prijavu u njihov sustav. Google u svom sustavu ima i mnoštvo drugih servisa koji se mogu koristiti te automatski ih u pozadini sve sinkronizira, tako da pomoću te jedne prijave svi resursi su odmah na raspolaganju osobi za razvoj i integraciju aplikacija. Klikom na kreiranje novog projekta prvo se pojavljuje opcija koji tip projekta se želi kreirati. Dvije početne opcije su izrada komponente za pametni dom, što se koristi u ovom projektu te izrada igre koja može biti jednostavna interakcija s virtualnim asistentom do cjelokupne online igre putem više uređaja. Postoji i treća opcija, koja daje najviše slobode u kreaciji iz koje može nastati kombinacija prvotne dvije opcije ili nešto sasvim novo. Nakon odabira prve opcije, sljedeći zaslon koji se prikazuje jest upis naziva akcije koja se kreira. Taj naziv mora biti jedinstven zbog toga što on služi za aktiviranje akcije glasovnim putem unutar

virtualnog asistenta. Također, pod tim nazivom je akcija dostupna unutar popisa dostupnih akcija raznih uređaja i proizvođača, odakle se aktivira za korištenje na mobilnim uređajima i registrira na Googleovim servisima.

5.1.1. Fulfillment URL

Sljedeći zaslon koji je potrebno ispuniti u realiziranju Google akcije sadržava jedno bitno polje, a to je fulfillment URL. Kako bi se razumjelo što je potrebno, prvo je potrebno razumjeti pojam webhook.

Prilikom razvijanja aplikacija postoji potreba za dobivanjem svakakvih vrsta informacija iz velikog broja izvora. Ponekad te informacije dolaze u jednakim intervalima, a ponekad su asinkrone te se ne poznaju trenuci u kojima će se promijeniti. Moguće je raditi konstante upite na servere od kojih se dobivaju informacije, ali to je loša praksa iz višestrukih razloga, kao što je opterećivanje propusnosti mreže zbog paketa koji se konstantno šalju, ali i sa strane servera od kojih dobivamo informacije, zakrčivanje njihovog prometa. Poneki serveri kao što je Github imaju ograničenje na broj upita koji primaju iz pojedinog izvora, što predstavlja još jedan razlog zbog kojih konstantni upiti nisu dobro rješenje. S vremenom se razvilo rješenje u obliku webhooka. Djeluju na način da postavljeni na određenu web stranicu primaju podatke ukoliko dođe do neke promjene na stranici. Nije potrebno svaki određeni vremenski interval slati podatke jer su oni nepromijenjeni, nego samo nakon njihovog ažuriranja, kako bi strana koja je kreirala webhook dobila najnovije informacije. Na taj način je uvelike smanjena količina podataka koja se šalje putem mreže, a može poslužiti i kao izvor drugih vrsta informacije, naprimjer kada se dogodilo zadnje ažuriranje podataka.

U realizaciji projekta potrebno je za webhook postaviti adresu na kojoj će se nalaziti cjelokupni backend projekt razvijen u JavaScriptu, koji će obrađivati podatke koje prosljeđujemo. Ukoliko adresa projekta nije još poznata, moguće je ostaviti prazno mjesto koje će kasnije biti promijenjeno kad se budu znali podaci.

Zadnji pogled koji se vidi u izborniku je „Account Linking“, odnosno pogled na kojemu se moraju postaviti autorizacijski podaci koji će biti korišteni u projektu. Razlog ovome je što Google koristi OAuth 2.0 protokol za autorizaciju, koji je industrijski standard. Protokol prema dokumentaciji zahtjeva određene sigurnosne postavke servera kao što je HTTPS protokol, što nadalje povlači potrebu za priznatim certifikatom o sigurnosti stranice i mnoge druge stvari. Zbog toga što je projekt izrađen putem Firebase baze podataka, na Googleovom serveru će biti postavljena cjelokupna aplikacija koja obrađuje podatke te on automatski implementira OAuth 2.0 protokol. Na taj način se razvojni programer ne mora previše brinuti oko autorizacije i sigurnosti na jednostavnijim projektima kao što je projekt koji se realizira, jer Google obavlja većinu pozadinskog posla i olakšava sam proces programiranje pametne akcije. Iz tog razloga se mogu odabrati bilo koji primjeri identifikacijske oznake klijenta i klijentove tajne koje se mogu

upisati u dostupna polja, dok u preostala dva polja je potrebno navesti putanju prvotno do stranice pomoću koje se prijavljujemo na servis te također i endpointa za dobivanje tokena za omogućavanje rada na stranici.

5.1.2. HomeGraph API

Iako kreirani projekt sam po sebi sadržava velik broj funkcionalnosti, zbog složenosti i zahtjeva ponekih projekata, ponekad je potrebno implementirati dodatne funkcionalnosti i mogućnosti koje nisu otpočetka dostupne. Kako se sve radi na Googleu, postoji velik broj već osposobljenih i optimiziranih servisa koji se koriste putem API-ja i koje je jedino potrebno uključiti unutar projekta kako bi se mogli koristiti. Jedan primjer takvog API-ja koji je potreban u realizaciji ove aplikacije je HomeGraph API. To je API koji integriran u projekt uvelike pojednostavljuje autorizaciju između mobilnog uređaja na kojem će se koristiti kreirana akcija i same akcije na Googleovom serveru koja će ju obrađivati koristeći generirani JSON token. Integracijom bilo kojeg Googleovom API-ja u projekt dobiva se cjelokupna stranica na kojoj se mogu pratiti razni podaci vezani uz API. U ovom slučaju, moguće je kreiranje metrika, određenih kvota koje će se promatrati na dnevnoj bazi, raznih grafova koji prate latentnost i broj grešaka u radu u odabranom razdoblju. U realizaciji ovog projekta potrebno je odabrati posljednju kategoriju u izborniku, naziva „Credentials“. Klikom na „Create Credentials“ potrebno je odabrati „Service account“, koji će poslužiti za autentifikaciju aplikacije i kreirane akcije. Sljedeći ekran koji je vidljiv je odabir imena računa koji se kreira i njegove email adrese. Odabir imena nije pretjerano bitan zbog toga što će se on sadržavati unutar kreiranog JSON tokena i služit će samo za autorizaciju te neće biti vidljiv nigdje u aplikaciji koja se kreira. Google ovdje opet automatizira određene poslove umjesto razvojnog programera na način da email adresa sadržava njihovu domenu, te je na korisniku da samo odabere prvi dio adrese, koji se čak i može generirati. Klikom na sljedeći korak dolazi se do opcionalnog odabira uloge servisnog računa koji se kreira. Ukoliko se ne odabere ništa, račun se može koristiti u bilo kojem slučaju rada aplikacije, no odabirom određene uloge može se ograničiti njegovo djelovanje na pojedine aspekte aplikacije. Ovdje se odabire pod kategorijom „Service accounts“ naziv „Service Account Token Creator“ i kreira se račun. Nakon kreiranja računa potrebno je kreirati ključ koji će se koristiti u mobilnoj aplikaciji radi autentifikacije. Kreiranje je jednostavan proces u kojem se mora odabrati koji format ključa se želi preuzeti, a na izboru su JSON i P12 format. Odabirom JSON formata i preuzimanjem dobiva se ključ u formatu:

```
      {
        "type": "service_account",
        "project_id": "exampleProject-136e3",
        "private_key_id": "608acc4c162e75ad9efd6311b7d2c98c92b6e179",
        "private_key":
            "-----BEGIN PRIVATE KEY-----
            \nMIIEvAIBADANBgkqhkiG9w0BAQEFAASCBKYwggSiAgEAAoIBAQCHEnFSK41
            T4uJH\ni8xBwfDBCWHHLZC5KZF+8tnf1SkNR8iAO5B/FVhqenmc3HSm8Tuf1W
            tkjAyUPx5N\nBo8b/sFgEe+wjXVeCXbB0M5WDdGk2S9Vxqfz+zrXK3Jkw3qm/
            qDN6GTdYEkzpjMH\n1r+ZCJBlK4+JDflbNcFQRAirDJpSF+zE+Eh84FhPzJ1u
            3fUPTmEWBEQlyGtDvjqa\nCKEyjU8GghQfXy2CuEBJWd+
            ...
            +dyzJlVRsrmEMuzzZ\nj9IdWPhabyFziGpflbnEkFg1veRM0GS7IO086J98Mp
            +b+PL36OryITod9Q0XbRXB\nW2IvaDlgtz8rCtRdnhYuPAgITi4hjRCLmvK6N
            ZTbj9qH+OPXAmMxWnQBi6F3Csb+\nZhuf4NfYu3RD4KVMoCZ+Zg==\n
            -----END PRIVATE KEY-----\n",
        "client_email": "exampleServiceAccount@explanation-
136e3.iam.gserviceaccount.com",
        "client_id": "115757977456046988017",
        "auth_uri": "https://accounts.google.com/o/oauth2/auth",
        "token_uri": "https://oauth2.googleapis.com/token",
        "auth_provider_x509_cert_url":
"https://www.googleapis.com/oauth2/v1/certs",
        "client_x509_cert_url":
"https://www.googleapis.com/robot/v1/metadata/x509/id-3-
303%40explanation-136e3.iam.gserviceaccount.com"
      }
```

Slika 26. Prikaz strukture Google generiranog ključa

Ključ sadrži brojne podatke, od kojih su neki već spomenuti u ovom poglavlju. Potrebno je napomenuti kako se u preuzetoj JSON datoteci nalazi i identifikacijska oznaka privatnog ključa, kao i cjelokupni privatni ključ.

5.2. Firebase

Firebase se može definirati kao sveobuhvatnu razvojnu platformu za razne aplikacije čiji je vlasnik Google. Detaljnije se firebase sastoji od seta raznih alata koji pomažu pri izgradnji, poboljšavanju i kasnije samom skaliranju aplikacije. Unaprijed pripremljeni alati koje firebase pruža omogućuju razvojnim programerima da preskoče određeni dio razvoja s obzirom da već navedeni alati omogućuju određene značajke koje bi u suprotnom razvojni programeri morali sami implementirati. Na ovaj način se otvara veći dio razvojnog ciklusa za rad na ključnim značajkama same aplikacije, a ne na razvoj potpornih servisa za istu. Neki od navedenih alata su analitika korištenja, autentifikacija, baza podataka, razne konfiguracije,

sustav za pohranu podataka, push notifikacije i slično. Sve usluge su sadržane na serveru uz mogućnost skaliranja po potrebi. Ovdje govorimo o backend komponentama koje u potpunosti održava sam Google. Kompleti za razvoj softvera, koji se koriste na klijentskog strani, imaju izravnu interakciju s prethodno navedenim backend komponentama. Ovdje se pojavljuje značajna razlika u odnosu na do sada uobičajen način programiranja, a to je razvoj frontend kao i backend dijela aplikacije, dok firebase omogućava prebacivanje znatno većeg naglaska na frontend dio aplikacije gdje klijentska strana izravno poziva aplikacijska programska sučelja samih firebase komponenata. Za detaljno podešavanje i rad s firebase komponentama služi Firebase konzola koja nudi mnoštvo funkcionalnosti. [18]

Slika 27. Firebase logo (izvor: World Vector Logo)

5.2.1. Firebase CLI

Pozadina rada pametne utičnice, odnosno backend aplikacija će biti postavljena na Firebase, zbog čega je potrebno postaviti Firebase Command Line Interface kako bi se to omogućilo. Prije postavljanja Firebasea, potrebno je preuzeti i instalirati NodeJS framework. On služi kako bi se JavaScript projekti uspješno pokretali na serveru, što je potrebno u realizaciji ovog projekta. Instalacija je jednostavna, gdje treba slijediti propisane upute te naposljetku, ukoliko je sve napravljeno kako treba, ključna riječ npm treba postati prepoznata komanda u terminalu.

Npm predstavlja najveći svjetski registar softvera s više od 800 tisuća paketa koda koji se mogu instalirati, ali također može služiti i za kreiranje privatnih komanda za izvršavanje u traženom projektu. Open-source je, što znači da svako može dodavati svoje pakete koda i komande po želji, ukoliko su ključne riječi pod kojim se paketi trebaju preuzeti slobodne. Uključuje također i CLI, odnosno Command Line Client, putem kojeg se unose komande i naredbe sustavu. Nakon što je npm dostupan, potrebno je instalirati Firebase paket pomoću naredbe [12]:

```
npm install -g firebase-tools
```

Slika 28. Naredba potrebna za instalaciju Firebase CLI u terminalu

S instalacijom tog paketa firebase naredba postaje dostupna, ali ju je potrebno povezati s Googleovim korisničkim računom. Korisnički račun koji je potrebno koristiti treba biti isti kao i za kreiranje akcije na Googleovoj platformi. Prijava se izvršava koristeći naredbu

```
firebase login
```

Slika 29. Naredba za prijavu u Firebase u terminalu

nakon čega korisnik ima cjelokupni pristup Firebase bazi podataka. Traženi projekt se odabire pomoću naredbe

```
firebase use project-name
```

Slika 30. Naredba za odabir traženog Firebase projekta

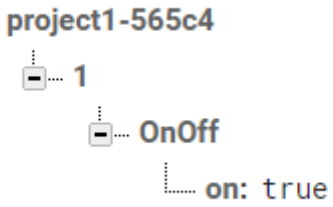
Google je ovdje također odradio jedan korak automatizacije procesa umjesto progamera, kreirajući Firebase bazu podataka prilikom kreiranja nove akcije za virtualnog asistenta. Akcija i baza podataka su kreirane pod istom identifikacijskom oznakom, zbog čega je potrebno navesti oznaku akcije pod „project-name“ u napisanoj komandi, kako bi se u terminalu počela koristiti ta baza podataka. Zadnja komanda koju je potrebno napomenuti jest

```
firebase deploy
```

Slika 31. Naredba za postavljanje projekta na Firebase

Pomoću te komande se postavlja projekt na Firebase bazu podataka te onda mogu slijediti sljedeći koraci sinkronizacije i autorizacije projekta.

Postavljeni projekt treba imati jednostavnu strukturu u bazi podataka, što je i vidljivo putem slike



Slika 32. Zapis u Firebase bazi podataka

5.3. Server aplikacija

Nakon objašnjavanja rada mobilne aplikacije u programskom jeziku Flutter i rada ESP32 mikrokontrolera, potrebno je objasniti na koji način je implementirana JavaScript aplikacija koja se pokreće na Firebaseu i na koju je povezana kreirana Google Home akcija. [21]

Prilikom pokretanja projekta prvotno se radi povezivanje aplikacije na Firebase, odnosno otvaranje veze prema bazi podataka. U ovom dijelu aplikacije se događa referenciranje na kreirani ključ putem „Service account“ funkcionalnosti HomeGraph API-ja koji je nazvan „smart-home-key“. On je zaslužan za autentifikaciju aplikacije u Firebaseu te bez njega ne bi bilo moguće raditi s bazom podataka ni na koji način. Sadržaj samog ključa se ne mijenja unutar kreirane aplikacije, niti se gledaju njegovi pojedini atributi, već se prosljeđuje u potpunosti na module koji obavljaju sve potrebne poslove.

U prethodnim poglavljima, kod kreiranja Google akcije opisano je kako mora postojati putanja za autorizaciju te putanja za dobivanje tokena. Za autorizaciju korisnika koristi se putanja projekta s nastavkom „fakeauth“. To je jednostavna metoda u kojoj nove razine provjere autorizacije nisu pretjerano potrebne zbog već odrađenih provjera prilikom povezivanja na Firebase. HTTPS zahtjev se dekodira i na temelju njega se kreira putanja za preusmjeravanje u slučaju uspješne autorizacije.

5.3.1. Lista uređaja

Putem jednog projekta moguće je postaviti veći broj uređaja na Google, koji mogu imati različite, nepovezane svrhe. Prilikom pokretanja aplikacije i sinkronizacije, događa se provjera popisa uređaja, koji se u aplikaciji nalaze u listi naziva „devices.json“. Prema ekstenziji same datoteke može se prepoznati da su podaci u njoj sadržani u obliku JSON datoteke u niz

objekata, koji sadrže velik broj podataka. Prvi podatak je identifikacijska oznaka objekta, koja mora biti jedinstvena i koja je vidljiva u Firebaseu. Sljedeći atribut je tip uređaja, kojeg je potrebno postaviti kako bi Google mogao prepoznati da je uređaj koji se želi dodati utičnica i kako bi za nju bila postavljena prikladna ikona i akcija. Iz potencijalnih uređaja koji se mogu pronaći na Googleu potrebno je odabrati naziv [17]:

```
action.devices.types.SWITCH
```

Slika 33. Kod za odabir prikaza pametne utičnice u Google Home aplikaciji

Putem liste uređaja određuje se i naziv koji će glasiti u Google Home aplikaciji. Ovdje postoji mogućnost zapisivanja pravog imena što je najbitnije za postavili, ali i nadimka akcije, što je u ovom slučaju ostavljeno nepromijenjeno.

```
"name": {
  "defaultNames": [
    "Smart socket"
  ],
  "name": "Smart socket",
  "nicknames": [
    "Smart socket"
  ]
}
```

Slika 34. Kod za postavljanje imena kreirane akcije

Na kraju JSON objekta su postavljene informacije o samom uređaju, što nije od pretjerane važnosti zbog toga što će se uređaj upotrebljavati samo u privatne svrhe, ali je svejedno definirano. Ovdje je definiran proizvođač, model pametne utičnice i hardver i softver verzija uređaja.

```
"deviceInfo": {
  "manufacturer": "Hrvoje Dumancic FOI",
  "model": "Smart socket v1.0",
  "hwVersion": "6.0",
  "swVersion": "7.0.1"
}
```

Slika 35. Kod za postavljanje tehničkih specifikacija kreirane akcije

5.3.2. Refresh i access tokeni

Sljedeća metoda, za dobivanje tokena, naziva „fakeauth“, radi na način da se prvo provjerava koji tip tokena je potrebno poslati. Postoje dvije vrste tokena koje se koriste, a to su refresh token i access token. Iako izgledom tokeni mogu biti slični, svrha, struktura i informacije koje sadržavaju su im vrlo različite.

Access tokeni su punog kraćeg trajanja, u većini slučajeva sat vremena te služe kao način autentifikacije klijenta koji zahtjeva ili šalje određene podatke prema serveru. Ukoliko je taj token još uvijek valjan, server može zaključiti kako se radi točno o tom korisniku, bez potrebe korisničkog slanja dodatnih atributa koji će pomoći pri autentifikaciji zbog toga što su ti podaci sadržani unutar tokena, u većini slučajeva obrađeni pomoću određenog algoritma zbog kojeg na prvi pogled ne izgledaju čitljivo.

Za dobivanje access tokena, prilikom prvog korištenja aplikacije ili za dobivanje novog tokena nakon njegovog isteka koristi se refresh token. Za razliku od access tokena, refresh tokeni su u pravilu dugotrajniji, gdje ih pojedini serveri znaju postaviti da traju godinu dana, kao što je Bitbucket, dok poneki nemaju ni tok trajanja, odnosno traju zauvijek. Zbog svoje važnosti, puno su bitniji od access tokena i potrebno je na njih obraćati više pozornosti kako ne bi došli u krive ruke i prouzročili potencijalne probleme korisnicima.

Za dobivanje tokena se šalje zahtjev koji može pripadati jednom od dva slučaja korištenja, kada korisnik koji šalje zahtjev posjeduje, odnosno ne posjeduje refresh token. Ukoliko korisnik posjeduje refresh token, tada je dovoljno u aplikaciji generirati novi access token, napisati iznos njegovog trajanja i poslati ga nazad kao odgovor korisniku. S druge strane, ukoliko korisnik ne šalje refresh token, tada je potrebno zajedno s ovim procesom generiranja access tokena, također generirati i refresh token te ga poslati nazad korisniku. Primjer strukture objekta koji se šalje korisniku kao odgovor u obliku JSON-a je [21]:

```
obj = {
      token_type: 'bearer',
      access_token: '123access',
      refresh_token: '123refresh',
      expires_in: secondsInDay,
    };
```

Slika 36. Prikaz JSON objekta generiranog refresh tokena

5.3.3. Kontrola pametne utičnice

Glavni ekran, koji se koristi za kontrolu pametne utičnice, sastoji se od samo jedne tipke, okruglog oblika koja ima simbol za uključivanje, odnosno isključivanje pametne utičnice. Klikom na njega se događa zapisivanje novog stanja utičnice u Firebase bazu podataka, na način da se upisuje cijela JSON struktura iz aplikacije koja je postavljena na Firebase. Primjer koda koji se postavlja je [21]:

```
if (Object.prototype.hasOwnProperty.call(snapshot, 'OnOff')) {
    syncvalue = Object.assign(syncvalue, {on: snapshot.OnOff.on});
  }

  const postData = {
    requestId: 'requestId',
    agentUserId: 'agentUserId',
    payload: {
      devices: {
        states: {
          [context.params.deviceId]: syncvalue,
        },
      },
    },
  };
const data = await app.reportState(postData);
```

Slika 37. Kod za ažuriranje stanja pametne utičnice

Pod atributom requestId potrebno je postaviti jedinstvenu identifikacijsku oznaku, koja ne mora imati ikakvo značenje. Pomoću metode reportState šalje se trenutno stanje objekta u bazu podataka. Ovdje se također koristi ključna riječ await kako bi se pričekao odgovor baze podataka na poslani zahtjev, nakon čega možemo nastaviti s korištenjem aplikacije.

5.4. Google Home aplikacija

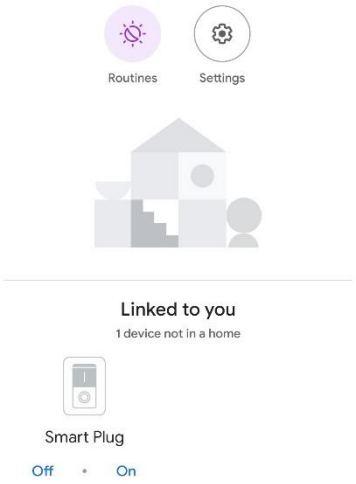
Nakon svih postavljenih elemenata na Firebase bazu podataka, potrebno je sinkronizirati kreiranu aplikacija s Google Home aplikacijom, preko koje će se odvijati upravljanje pametnom utičnicom. Aplikacija je dostupna na Google Play trgovini te ju je potrebno preuzeti i prijaviti se s istim računom s kojem je napravljena akcija na Googleovoj stranici, odnosno računom na kojem je kreirana Firebase baza podataka.

U gornjem lijevom kutu se nalazi ikona plus, ikona dodavanja novog pametnog uređaja. Sljedeće je potrebno odabrati postavljanje novog uređaja nakon čega se odabire opcija pametnog uređaja koji radi zajedno s Googleom, kao što je slučaj u ovom projektu.

Potrebno je pronaći uređaj istog naziva kao što je naziv akcije kreirane putem Googleovog servisa „Actions on Google“. Ukoliko na popisu opcija pametnih uređaja za povezivanje nije moguće pronaći traženu akciju, potrebno je na Googleovom servisu odabrati projekt u kojem je kreirana akcija te odabrati opciju „Test“, pokrenuti je jednom, nakon čega će akcija biti omogućena za korištenje na mobilnom uređaju.

Slika 38. Povezivanje s kreiranim projektom

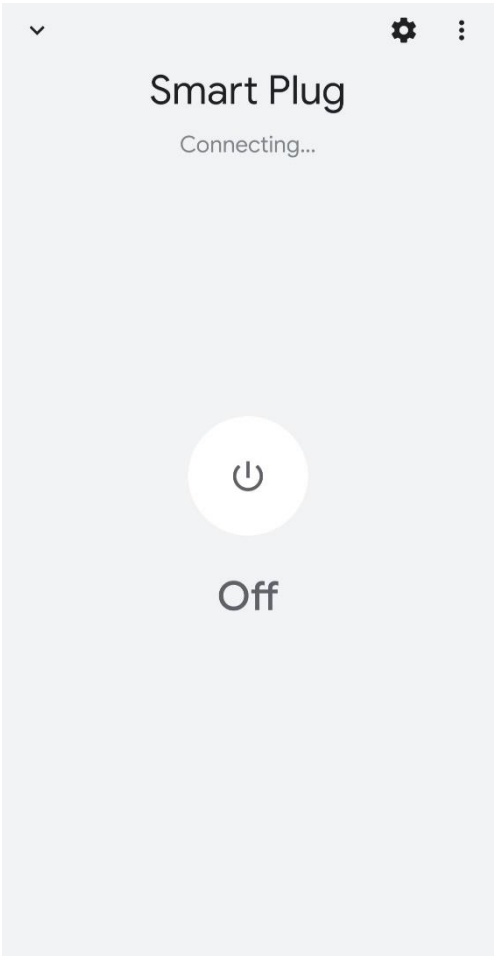
Nakon što se pronađe kreirani uređaj, klikom na njega se događa sinkronizacija i postavljanje pametne akcije s Firebase baze podataka. Ukoliko proces prođe bez problema, na početnom zaslonu Google Home aplikacije trebala bi postati dostupna kreirana pametna akcija. Odmah prilikom



Slika 39. Google Home početni zaslon

Pametnu akciju je moguće upravljati i na ovom glavnom ekranu, ali je moguće kliknuti na ikonu čime se dobije upravljanje akcije preko cijelog ekrana. Jednostavnog je prikaza, koji se sastoji od samo jednog gumba preko kojeg se uključuje i isključuje i šalje podatke na

Firebase bazu podataka. Potrebno je napomenuti kako aplikacije ne sadrži sinkronizaciju s trenutnim stanjem baze podataka prilikom pokretanja, zbog čega je početno stanje pametne utičnice koje se prikazuje na mobilnom uređaju uvijek isključeno.



Slika 41. Isključena pametna utičnica u Google Home aplikaciji

Slika 40. Uključena pametna utičnica u Google Home aplikaciji

6. Zaključak

Izrada ovog projekta zapravo obuhvaća izradu tri zasebna dijela koja su povezana u funkcionalnu cjelinu i omogućuju automatizaciju rada pametne utičnice. Prvi i najvažniji dio je bila izrada same utičnice koristeći ESP32 mikrokontroler i relej. S obzirom na veliku mogućnost prouzročavanja ozljeda zbog toga što se radi s električnom energijom, ovaj dio je potrebno odraditi s dodatnom razinom opreznosti kako bi se osigurala maksimalna sigurnost i spriječila mogućnost nastajanja potencijalnih ozljeda. Zbog toga je za projekt bio potreban duži vremenski period nego što je zamišljeno, ali naposljetku je skripta koja se povezuje na Firebase uspješno razvijena, hardverske komponente spojene i pametna utičnica realizirana.

Nakon razvijene utičnice, bilo je potrebno implementirati način njenog upravljanja. Tu su razvijene dvije aplikacije, prva u programskom jeziku Dart koristeći razvojni alat Flutter. Flutter je relativno nova tehnologija koja je isto predstavljala novitet i prepreku u realizaciji projekta, zbog čega je prvotno bilo potrebno pročitati dokumentaciju i pogledati pokoji video objašnjenja da se shvati princip djelovanja i na koji način se komponente povezuju u cjelinu. Naposljetku je razvijena aplikacija koja se povezuje na Firebase bazu podataka i iz nje povlači podatke po potrebi, kako bi stanje pametne utičnice u svakom trenutku bilo vidljivo na mobilnom uređaju.

Zadnja komponenta, zahtijevala je izradu aplikacije koja je integrirana s Googleovim servisima i koja omogućuje da se preko Googleovog virtualnog asistenta izvršava kontrola nad utičnicom. Prvotna implementacija je koristila IFTTT servis, poznati i popularan servis koji predstavlja vezu između poslanih naredbi i izvršenih akcija na asistentu, no zbog potrebe za direktnim i samostalnim obrađivanjem informacija, ovakva implementacija je u potpunosti odbačena. Naposljetku, nakon mnogo istraživanja razvijena je aplikacija koristeći JavaScript programski jezik koja se postavlja na Firebase i sinkronizira s kreiranom Google pametnom akcijom. Izazov prilikom realiziranja ove komponente je predstavljao i način sinkroniziranja pojedinih dijelova, u prvom planu Google kreirane akcije i Firebase baze podataka kako bi se ostvarilo što bolje funkcioniranje finalnog proizvoda.

Iako je realizacija projekta bila izazovna i komplicirana, konačan rezultat predstavlja veliki uspjeh i veliki korak u implementaciji pametnih akcija u svakodnevnom kućanstvu. Neke od najtežih stvari vezanih uz razvoj pametnih akcija su savladane u projektu, stoga daljna implementacija uređaja i razvoj aplikacija za potrebe kućanstva su realan i logičan nastavak rada s mikrokontrolerima.

Popis literature

[1] „Flutter architectural overview“, *Flutter.dev*, 2020. [Online]. Dostupno s: https://flutter.dev/docs/resources/technical-overview. (Pristupljeno: 18-8-2020).

[2] „StatefulWidget class - widgets library - Dart API“, *Api.flutter.dev*, 2020. [Online]. Dostupno s: https://api.flutter.dev/flutter/widgets/StatefulWidget-class.html. (Pristupljeno: 18-8- 2020).

[3] S. Balaganur, „How Google Flutter Became The Frontrunner In Mobile App Development“, *Analytics India Magazine*, 2020. [Online]. Dostupno s: https://analyticsindiamag.com/google-flutter-mobile-app-development/. (Pristupljeno: 18-8-2020).

[4] F. Lardinois, „Google will not integrate its Dart programming language into Chrome“, *Techcrunch.com*, 2020. [Online]. Available:https://techcrunch.com/2015/03/25/google-will-not-integrate-its-dart-programming-language-into-chrome/. (Pristupljeno: 18-8-2020).

[5] „Dart (Dart VM)“, *Dart.dev*, 2020. [Online]. Dostupno s: https://dart.dev/tools/dart-vm. (Pristupljeno: 18-8-2020).

[6] O. Obinna, „How does JIT and AOT work in Dart?“, *Medium*, 2020. [Online]. Dostupno s: https://medium.com/@onuohasilver9/how-does-jit-and-aot-work-in-dart-cab2f31d9cb5. (Pristupljeno: 18-8-2020].

[7] „Hot reload“, *Flutter.dev*, 2020. [Online]. Dostupno s: https://flutter.dev/docs/development/tools/hot-reload. (Pristupljeno s: 18-8-2020).

[8] M. Rouse, „What is a Microcontroller and How Does it Work?“, *IoT Agenda*, 2020. [Online]. Dostupno s: https://internetofthingsagenda.techtarget.com/definition/microcontroller. (Pristupljeno: 18-8-2020).

[9] R. Keim, „What Is a Microcontroller? The Defining Characteristics and Architecture of a Common Component - Technical Articles“, *Allaboutcircuits.com*, 2020. [Online]. Dostupno s: https://www.allaboutcircuits.com/technical-articles/what-is-a-microcontroller-introduction-component-characteristics-component/. (Pristupljeno: 18-8-2020).

[10] „Google Assistant“, *En.wikipedia.org*, 2020. [Online]. Dostupno s: https://en.wikipedia.org/wiki/Google_Assistant. [Pristupljeno: 18-8-2020).

[11] „Actions on Google“, *En.wikipedia.org*, 2020. [Online]. Dostupno s: https://en.wikipedia.org/wiki/Actions_on_Google. (Pristupljeno: 18-8-2020).

[12] „What is npm“, *W3schools.com*, 2020. [Online]. Dostupno s: https://www.w3schools.com/whatis/whatis_npm.asp. (Pristupljeno: 18-8-2020).

[13] S. Peyrott, „Refresh Tokens: When to Use Them and How They Interact with JWTs“, *Auth0 - Blog*, 2020. [Online]. Dostupno s: https://auth0.com/blog/refresh-tokens-what-are-they-and-when-to-use-them/. (Pristupljeno: 18-8-2020).

[14] A. Parecki, „What is the OAuth 2.0 Authorization Code Grant Type?“, *Okta Developer*, 2020. [Online]. Dostupno s: https://developer.okta.com/blog/2018/04/10/oauth-authorization-code-grant-type. (Pristupljeno: 18-8-2020).

[15] S. Santos, „ESP32 Pinout Reference: Which GPIO pins should you use?“, *Random Nerd Tutorials*, 2020. [Online]. Dostupno s: https://randomnerdtutorials.com/esp32-pinout-reference-gpios/. (Pristupljeno: 18-8-2020).

[16] „How to use a breadboard“, *Sparkfun – Start something*, 2020. [Online]. Dostupno s: https://learn.sparkfun.com/tutorials/how-to-use-a-breadboard/all (Pristupljeno: 18-8-2020).

[17] „Smart Home Device Types“, *Google Assistant*, 2020. [Online] Dostupno s: https://developers.google.com/assistant/smarthome/guides (Pristupljeno: 18-8-2020)

[18] D. Stevenson, „What is Firebase? The complete story, abridged.“, *Medium*, 2020. [Online]. Dostupno s: https://medium.com/firebase-developers/what-is-firebase-the-complete-story-abridged-bcc730c5f2c0 (Pristupljeno: 18-8-2020)

[19] „What is the difference between microprocessor and microcontroller?“, *Circuit Digest,* 2020. [Online]. Dostupno s: https://circuitdigest.com/article/what-is-the-difference-between-microprocessor-and-microcontroller (Pristupljeno: 18-8-2020).

[20] C. Svec, „ESE101: Microcontroller peripherals, gpios and blinking lights: part 1“, *Embedded*, 2020. [Online]. Dostupno s: https://embedded.fm/blog/2016/5/16/ese101-peripherals-part-1 (Pristupljeno: 18-8-2020)

[21] Siddharth, S. Google-Actions-Smarthome, *GitHub*. 2020. [online] Dostupno s: https://github.com/shivasiddharth/google-actions-smarthome (Pristupljeno: 20-8-2020].

[22] Plugin Project - Firebase_Core_Web Not Found. *Stack Overflow*. Dostupno s: https://stackoverflow.com/questions/61732409/plugin-project-firebase-core-web-not-found (Pristupljeno: 21-8-2020).

[23] H. Dumancic, "hrvojedumancic/SmartSocket", *GitHub*, 2020. [Online]. Dostupno s: https://github.com/hrvojedumancic/SmartSocket. [Pristupljeno: 23-8-2020].

Popis slika