Dvosmjerna komunikacija temeljena na MQTT protokolu unutar virtualnog i stvarnog okruženja

Kolar, Luka

S V E U Č I L I Š T E U Z A G R E B U

FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Luka Kolar

Zagreb, 2024.

S V E U Č I L I Š T E U Z A G R E B U

FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Mentor:

Izv. prof. dr. sc. Tomislav Stipančić, dipl. ing.

Student:

Luka Kolar

Zagreb, 2024.

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

Zahvaljujem mentoru izv. prof. dr. sc. Tomislavu Stipančiću, dipl. ing. te asistentu Leonu Korenu, mag. ing. na dostupnosti, pristupačnosti, korisnim savjetima i pomoći prilikom izrade ovog diplomskog rada.

Zahvaljujem bratu Ivanu, djevojci Mariji i svim prijateljima na podršci, pomoći i divnim trenutcima tijekom studiranja.

Posebno zahvaljujem svojim roditeljima Jadranki i Zlatku bez čije potpore, savjeta, strpljenja i ljubavi ne bih uspio.

Luka Kolar

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za diplomske ispite studija strojarstva za smjerove:
Proizvodno inženjerstvo, inženjerstvo materijala, industrijsko inženjerstvo i menadžment, mehatronika i robotika, autonomni sustavi i računalna inteligencija

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa: **602 - 04 / 24 - 06 / 1**	
Ur.broj: **15 - 24 -**	

DIPLOMSKI ZADATAK

Student: **Luka Kolar** JMBAG: 0035215538

Naslov rada na hrvatskom jeziku: **Dvosmjerna komunikacija temeljena na MQTT protokolu unutar virtualnog i stvarnog okruženja**

Naslov rada na engleskom jeziku: **Two-way communication based on the MQTT protocol within the virtual and real environment**

Opis zadatka:

Događaje iz stvarne i virtualne okoline moguće je povezati koristeći MQTT komunikacijski protokol koristeći različite senzore. Na taj je način moguće pratiti promjene unutar jednog prostora te dostavljati te informacije relevantnim uređajima ili njihovim virtualnim inačicama unutar druge okoline.

U radu je potrebno:

- proučiti Unreal Engine 5 razvojnu okolinu za stvaranje interaktivnih digitalnih okruženja te implementirati model digitalnog blizanca laboratorija
- proučiti Blueprint sustav u Unreal Engine 5 s ciljem razumijevanja implementacije MQTT komunikacije unutar digitalnog prostora
- izraditi komponente za primanje i slanje MQTT poruka unutar virtualnog prostora, uključujući kontrolu osvjetljenja i otvaranje vrata hladnjaka
- integrirati postojeće rješenje s Home Assistant platformom za ostvarivanje pametnih prostora radi daljinskog upravljanja uređajima
- povezati senzore iz stvarnog okruženja s digitalnim prostorom putem MQTT protokola kako bi se osigurale interakcije te omogućile povratne informacije kroz različite prostore
- eksperimentalno evaluirati razvijena rješenja u sklopu Laboratorija za projektiranje izradbenih i montažnih sustava.

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

Zadatak zadan:	Datum predaje rada:	Predviđeni datumi obrane:
9. svibnja 2024.	11. srpnja 2024.	15. – 19. srpnja 2024.
Zadatak zadao:		Predsjednik Povjerenstva:
Izv. prof. dr. sc. Tomislav Stipančić		Prof. dr. sc. Ivica Garašić

Sadržaj

Sadržaj I

Popis slika II

Popis tablica V

SAŽETAK VI

SUMMARY VII

1. UVOD 1

2. DIGITALNI BLIZANAC 2

2.1. Tehnologija digitalnog blizanca 2
2.2. Stadiji razvitka digitalnih blizanaca 4
2.3. Tipovi digitalnih blizanaca 6
2.4. Laboratorij za projektiranje izradbenih i montažnih sustava 7

3. MQTT PROTOKOL 9

3.1. Princip rada 9
3.2. Karakteristike 10
3.3. Primjena MQTT protokola 11

4. UNREAL ENGINE 5 12

4.1. Stvaranje 3D objekta 12
4.2. Interakcija s modelom 14
4.3. Blueprint programiranje 16
4.4. Unreal Engine 5 i MQTT protokol 19

5. KONCEPT INTERNETA STVARI LABORATORIJA 21

5.1. Zigbee 21
5.2. Sonoff SNZB-04, kontaktni senzor 23
5.3. Aqara Smart Wall Switch H1, pametni prekidač 24
5.4. Sonoff ZBDONGLE-P, Zigbee koordinator 25
5.5. Home Assistant 26
5.6. Povezivanje i konfiguracija 27

6. VIRTUALNI MODEL LABORATORIJA I UREĐAJA 32

6.1. Stvaranje modela 33
6.2. Logika interakcije i animacija 37

7. OSTVARIVANJE DVOSMJERNE KOMUNIKACIJE 45

7.1. Definiranje klijenta i povezivanje s brokerom 46
7.2. Komunikacija rasvjete 47
7.3. Komunikacija hladnjaka 55

8. KRITIČKI OSVRT 66

9. ZAKLJUČAK 67

LITERATURA: 68

Popis slika

Slika 1. Gartnerov model evolucije digitalnih blizanaca [1] 5
Slika 2. Stadiji evolucije digitalnih blizanaca [1] 6
Slika 3. Digitalni blizanac sustava [2] 7
Slika 4. Struktura MQTT protokola [3] 9
Slika 5. Početni zaslon Unreal Engine-a 5 12
Slika 6. Uređivanje osnovnog oblika *Box* 13
Slika 7. Početni modeli unutar *StarterContent* mape 13
Slika 8. Quixel Bridge knjižnica 13
Slika 9. Kreiranje C++ klase 15
Slika 10. Osnovni *template* za kreiranu klasu 15
Slika 11. Kreiranje *Blueprint* klase 16
Slika 12. Dodavanje *StaticMesh* komponente u *Blueprint* 17
Slika 13. *Event Graph* 17
Slika 14. Primjer jednostavne interakcije 18
Slika 15. Instaliranje MQTT *plugin*-a 19
Slika 16. Osnovni MQTT *Blueprint node*-ovi 20
Slika 17. Zigbee *mesh* struktura [4] 23
Slika 18. Sonoff SNZB-04 [5] 24
Slika 19. Aqara Smart Wall Switch H1 [6] 25
Slika 20. Sonoff ZBDONGLE-P [7] 26
Slika 21. Home Assistant laboratorija 27
Slika 22. *Addon*-ovi 29
Slika 23. Integracije 29
Slika 24. Uređaji koji koriste Zigbee2MQTT 29
Slika 25. Primjer automatizacije 30
Slika 26. Laboratorij za projektiranje izradbenih i montažnih sustava, pogled 1. 32
Slika 27. Laboratorij za projektiranje izradbenih i montažnih sustava, pogled 2. 33
Slika 28. *Modeling Mode* i konstrukcija laboratorija 34
Slika 29. Modeli hladnjaka i prekidača 35
Slika 30. *Blueprint*-i objekta i vanjski izgleda laboratorija 35
Slika 31. Unutrašnjost laboratorija, pogled 1 36

Slika 32. Unutrašnjost laboratorija, pogled 2....36
Slika 33. Stvaranje *Interface*-a....38
Slika 34. Dodavanje funkcije i varijabli u *Interface*-u....38
Slika 35. Akcija „*Interact“*....39
Slika 36. Logika unutar *"BP_FirstPersonCharacter"*....40
Slika 37. *"Interact"* funkcija u *"BP_LightSwitch"*....41
Slika 38. *EventGraph "BP_LightSwitch"*....42
Slika 39. *"Interact"* funkcija u *"BP_Fridge"*....43
Slika 40. *EventGraph "BP_Fridge"*....43
Slika 41. Paljenje svjetla....44
Slika 42. Otvaranje hladnjaka....44
Slika 43. Definiranje klijenta i povezivanje s brokerom....46
Slika 44. *Payload* JSON *LightSwitch* uređaja....47
Slika 45. *Light Switch Subscriber*....49
Slika 46. Promjena *EventGraph*-a u *"BP_LightSwitch"*....50
Slika 47. *Light Switch Publisher*....51
Slika 48. *source* datoteka....53
Slika 49. *header* datoteka....53
Slika 50. Upaljeno svjetlo u oba okruženja....54
Slika 51. Ugašeno svjetlo u oba okruženja....54
Slika 52. Prikaz upaljenog stanja u Home Assistant-u....55
Slika 53. Prikaz ugašenog stanja u Home Assistant-u....55
Slika 54. Dodavanje "virtualnog" senzora....56
Slika 55. Kartica za objavu poruke virtualnog hladnjaka na preglednoj ploči....57
Slika 56. *Payload* JSON kontaktnog senzora hladnjaka....57
Slika 57. Izmijenjena funkcija „*Interact“* u *"BP_Fridge"*....59
Slika 58. Izmjena *"BP_Fridge" Event Graph*-a....60
Slika 59. *Fridge Publisher*....61
Slika 60. *Fridge Publisher* početnog i konačnog stanja simulacije....62
Slika 61. *Fridge Subscriber*....62
Slika 62. Hladnjak otvoren u stvarnom okruženju....63
Slika 63. Hladnjak zatvoren u stvarnom okruženju....64

Slika 64. Hladnjak otvoren u virtualnom okruženju 64
Slika 65. Hladnjak zatvoren u virtualnom okruženju 65

Popis tablica

Tablica 1. Temeljne tehnologije digitalnog blizanca 4

Tablica 2. Specifikacije Zigbee standarda 22

SAŽETAK

U ovom diplomskom radu se ostvaruje dvosmjerna komunikacija temeljena na MQTT protokolu kojom se povezuju događaji iz stvarnog i virtualnog okruženja. Pomoću alata koje posjeduje program Unreal Engine 5, postavlja se virtualna inačica Laboratorija za projektiranje izradbenih i montažnih sustava te se modeliraju uređaji unutar prostora koji posjeduju senzore za praćenje promjena unutar istog, prvenstveno osvjetljenje i hladnjak. Nadalje, koristeći MQTT *plugin* i *Blueprint* sustav programiranja unutar Unreal Engine-a 5, definira se logika odvijanja animacija uređaja u virtualnom okruženju i razmjene MQTT poruka između uređaja i njihovih virtualnih inačica. Tako se dobiva potpuno funkcionalan model digitalnog blizanca laboratorija koji omogućava praćenje i kontrolu stanja unutar stvarnog okruženja. Također, integrira se i postojeće rješenje interneta stvari s Home Assistant platformom u kojoj je definirana komunikacija uređaja u stvarnom okruženju radi daljinskog upravljanja uređajima.

Ključne riječi: MQTT, MQTT komunikacijski protokol, Unreal Engine 5, virtualna stvarnost, VR, digitalni blizanac, internet stvari, IoT

SUMMARY

In this masters thesis, two-way communication based on the MQTT protocol is implemented to connect events from both the real and virtual environments. Using tools provided by the Unreal Engine 5 program, a virtual version of the Laboratory for Manufacturing and Assembly Systems Planning is set up and devices within the space equipped with sensors for monitoring changes, primarily lighting and a refrigerator, are modeled. Furthermore, using the MQTT plugin and the Blueprint programming system within Unreal Engine 5, the logic for device animation in the virtual environment and the exchange of MQTT messages between the devices and their virtual counterparts is defined. This results in a fully functional digital twin model of the laboratory, allowing monitoring and control of the real environment's state. Additionally, the existing Internet of Things solution is integrated with the Home Assistant platform, where the communication of devices in the real environment for remote control is defined.

Keywords: MQTT, MQTT communication protocol, Unreal Engine 5, virtual reality, VR, digital twin, Internet of Things, IoT

1. UVOD

U današnjem digitalnom dobu, industrija sve više koristi napredne tehnologije kako bi unaprijedila svoje procese i optimizirala performanse. Jedan od ključnih koncepta koji postaje sve prisutniji je digitalni blizanac.

Jedna od temeljnih karakteristika digitalnog blizanca je reprezentacija. Digitalni blizanac je virtualni model koji točno odražava stvarni objekt, proces ili sistem. To može biti stroj, postrojenje, tvornica ili cijeli proizvodni lanac. Međutim, kako bi se korisnost digitalnog blizanca podigla na višu razinu, uvodi se komunikacija između virtualnog modela i stvarnog objekta. Komunikacija se postiže implementacijom *Internet of Things* uređaja te razmjenom podataka uređaja putem mreže pomoću komunikacijskih protokola.

Unreal Engine 5 je moćan alat koji omogućava izradu visokokvalitetnih virtualnih modela. Njegove napredne grafičke mogućnosti koje omogućuju detaljnu vizualizaciju te mogućnost definiranja smjera komunikacije između uređaja u stvarnom okruženju i virtualnom modelu pomoću *Blueprint* sustava ili C++ programskog jezika ga čine idealnim za navedenu upotrebu.

MQTT protokol omogućava pouzdanu i efikasnu komunikaciju između različitih uređaja putem mreže. Pomoću MQTT protokola, virtualni modeli izrađeni u Unreal Engine-u 5 mogu komunicirati s fizičkim objektima i suprotno. Tako se podaci o stanju, performansama ili okolini objekata mogu neprekidno razmjenjivati između dva modela.

Ova kombinacija tehnologija otvara vrata za brojne primjene, uključujući simulaciju, upravljanje, testiranje ili kontinuirano praćenje i optimizaciju performansi stvarnih objekata ili sistema. Osim u industriji, ove tehnologije se mogu primijeniti i u kućnim okruženjima.

Cilj ovoga rada je približiti koncept rada MQTT protokola, objasniti značajke Unreal Engine 5 softvera, prikazati mogućnost implementacije protokola u isti te time izraditi digitalnog blizanca koji posjeduje virtualne modele potrebnih uređaja kako bi se kombinacijom ovih tehnologija povezali događaji iz stvarne i virtualne okoline. Također, integrira se i postojeće rješenje s Home Assistant platformom radi daljinskog upravljanja uređajima.

2. DIGITALNI BLIZANAC

Digitalni blizanac je digitalna reprezentacija fizičkog objekta, sustava ili procesa koja koristi stvarne podatke za testiranje, simulaciju, analizu i optimizaciju njegovog ponašanja. Digitalni blizanci omogućuju korisnicima da prate, modeliraju i kontroliraju fizičke entitete u virtualnom okruženju, čime se poboljšava učinkovitost, smanjuju troškovi i povećava sigurnost. Stoga, digitalni blizanac je skup različitih tehnologija kao što su vizualizacija i modeliranje visoke rezolucije, simulacije, senzorika, analiza podataka, internet stvari i strojno učenje.

2.1. Tehnologija digitalnog blizanca

Digitalni blizanac može se definirati kao tehnološka platforma za sinkronizaciju fizičkih objekata i digitalnih objekata koji ih imitiraju u realnom vremenu, analiziranje situacija prema različitim namjenama i optimiziranje fizičkih objekata predviđanjem na temelju analiziranih rezultata. Kako bi navedeno bilo moguće, koncept sadrži sljedeće ključne komponente:

- **Fizički objekt:** Stvarni entitet koji se replicira digitalno.
- **Senzori i uređaji za prikupljanje podataka:** Senzori prikupljaju podatke o stanju, performansama i okruženju fizičkog objekta.
- **Digitalna platforma:** Softverska platforma koja prima, pohranjuje i obrađuje prikupljene podatke senzora.
- **Analitički alati i modeli:** Algoritmi i modeli koji koriste prikupljene podatke za simulaciju i analizu ponašanja fizičkog objekta.
- **Sučelja za vizualizaciju:** Alati za vizualizaciju koji omogućuju korisnicima interakciju s digitalnim blizancem.

Korištenjem ključnih komponenti te mnoštvom tehnologija koje digitalni blizanac posjeduje, ostvaruju se opće namjene korištenja tehnologije digitalnog blizanca koje su navedene u nastavku.

Optimizacija procesa. „*What-if*“ simulacije temeljene na modelima ponašanja digitalnog blizanca mogu pomoći u pronalaženju poboljšanih operativnih procesa prema bilo kojoj promjeni povezanog osoblja, opreme, proizvodnih postupaka, komponenti itd.

Sprječavanje problema u stvarnom svijetu unaprijed. Analiza prošlih i sadašnjih informacija prikupljenih iz stvarnog svijeta u virtualnom svijetu omogućuje identifikaciju rizičnih faktora.

Učinkovit dizajn proizvoda. Simulacije dizajna proizvoda putem digitalnog blizanca, koristeći podatke iz stvarnog svijeta naučene iz performansi postojećih uređaja, procesa i proizvoda tijekom vremena, mogu podržati eksperimente s iteracijama dizajna, informiranije dizajnerske i inženjerske odluke te ukupno poboljšanje plana proizvoda.

Analiza uzroka. Modeli ponašanja digitalnog blizanca mogu reproducirati događaje koji se događaju njegovom fizičkom entitetu. Rezultati simulacije temeljeni na prošlim i zapisničkim podacima mogu pomoći u analizi zašto su se ti događaji dogodili.

Multidisciplinarno donošenje odluka. Federativno međudjelovanje digitalnih blizanaca može olakšati identifikaciju uzroka povezane i složene problematike, analizu međusobnih odnosa i nuspojava koje se događaju između industrijskih domena te suradnju među dionicima kroz industrijski ekosustav.

Navedene namjene digitalnih blizanaca se ostvaruju mnoštvom tehnologija. Te tehnologije su prikazane u tablici 1. i mogu se podijeliti u 6 glavnih kategorija:

- Vizualizacija i operacije,
- Analiza,
- Tehnologije modeliranja i simulacije,
- Tehnologije povezivanja,
- Podaci i sigurnost,
- Sinkronizacija.

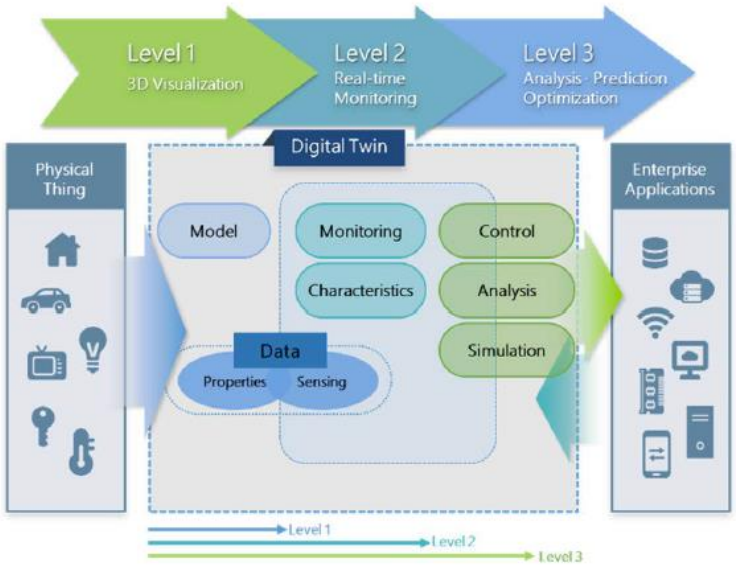
Međutim, trenutne mogućnosti digitalnih blizanaca višestruko se razlikuju od svojih početnih inačica. Redovite implementacije novih, navedenih tehnologija su unaprjeđivale koncept digitalnog blizanca kroz vrijeme. Stoga su u nastavku prikazani stadiji razvitka na temelju kojih se svaki digitalni blizanac može kategorizirati po mogućnosti i kompleksnosti.

Tablica 1. Temeljne tehnologije digitalnog blizanca

Vizualizacija i operacije	AR/VR/XR	HMI	Pretraživanje podataka i operacija	Raspolaganje resursima
Analiza	Dijeljenje podataka	Korelacijska analiza	Obavještenje	Analiza nezgoda
Modeliranje i simulacije	Ažuriranje u stvarnom vremenu	Integracija podataka	Automatizacija uvjetovana pravilima	Prepoznavanje 3D modela
Povezivanje	Komunikacijski protokoli	IoT	Povezivanje virtualnih objekata	Žičano/bežično povezivanje
Podaci i sigurnost	Enkripcija podataka	Optimizacija podataka	Brisanje podataka	Vjerodostojnost automatizacije
Sinkronizacija	Navedenih tehnologija			

2.2. Stadiji razvitka digitalnih blizanaca

Gartner, vodeća svjetska istraživačka i savjetodavna tvrtka koja pruža uvid i savjete u području informacijske tehnologije, uključujući razvoj i primjenu digitalnih blizanaca, pruža model evolucije koncepta digitalnih blizanaca u tri faze koji je široko prihvaćen. Prema tom modelu, u prvoj fazi se duplicira stvarni svijet te su digitalni blizanci te faze isključivo reprezentativni. U drugoj fazi, digitalni blizanci omogućavaju kontroliranje objekata iz stvarnog okruženja, dok se u trećoj fazi uvodi mogućnost optimizacije stvarnog svijeta. Stoga, u većini postojećih istraživanja, nakon dupliciranja pojedinačnog proizvoda ili sustava u virtualnom svijetu, isti se može optimizirati na temelju rezultata simulacije dupliciranog modela. Takav model evolucije je prikazan na slici 1.

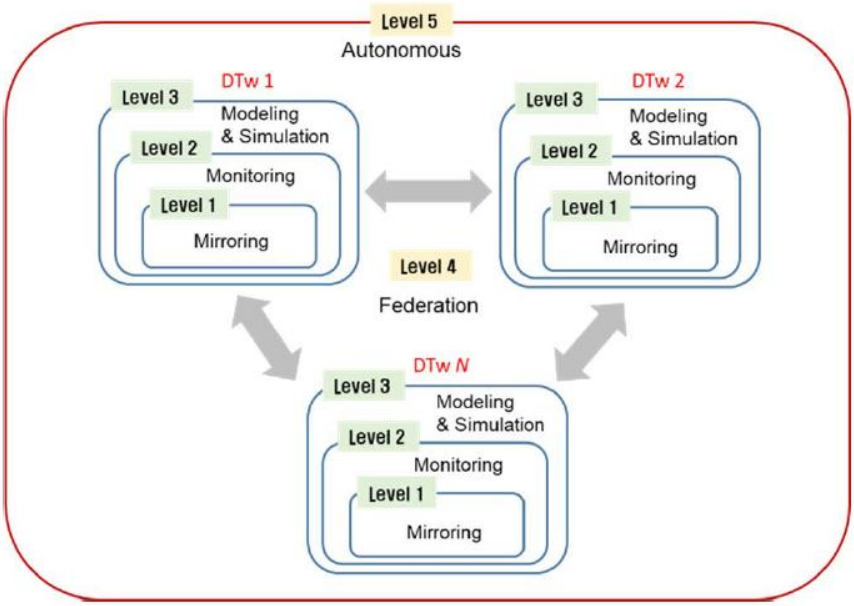


Slika 1. Gartnerov model evolucije digitalnih blizanaca [1]

Međutim, zbog kompleksnosti objekata, procesa i sustava u današnjem svijetu, digitalni blizanac za jedan sustav ne može pružiti sveobuhvatno, optimalno rješenje. Stoga, pruža se novi model evolucije digitalnih blizanaca koji sadrži 5 stadija razvitka. Ti stadiji su:

- **Stadij 1 – Zrcaljenje:** Dupliciranje fizičkog objekta u digitalnog blizanca.
- **Stadij 2 – Praćenje:** Praćenje i kontroliranje fizičkog objekta na temelju analize digitalnog blizanca.
- **Stadij 3 – Modeliranje i simulacija:** Optimiziranje fizičkog objekta kroz rezultate simulacije digitalnog blizanca.
- **Stadij 4 – Federacija:** Konfiguriranje federalnih digitalnih blizanaca, optimizacija složenih fizičkih objekata i međusobno djelovanje federalnih digitalnih blizanaca i složenih fizičkih objekata.
- **Stadij 5 – Automatizacija:** Autonomno prepoznavanje i rješavanje problema u federalnim digitalnim blizancima te optimizacija fizičkih objekata prema rješenju federalnog digitalnog blizanca.

Temeljna razlika između Gartnerovog i novog modela evolucije je ta što se u Gartnerovom modelu razmatra stvaranje digitalnog blizanca samo za jedan sustav, dok novi model predstavlja povezivanje nekolicine digitalnih blizanaca različitih sustava koji čine federaciju.



Slika 2. Stadiji evolucije digitalnih blizanaca [1]

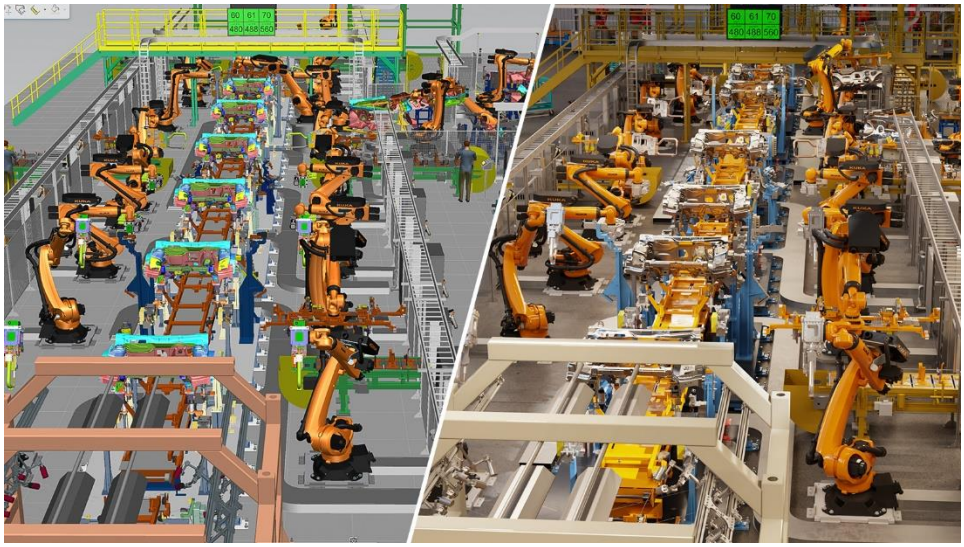
2.3. Tipovi digitalnih blizanaca

Osim po kompleksnosti i mogućnostima koje su se povećavale tijekom vremena, tipovi digitalnih blizanaca mogu se sistematizirati i po veličini proizvoda i području primjene. Također, kako je navedeno, moguće je da različite vrste digitalnih blizanaca koegzistiraju unutar sustava ili procesa. To su:

- **Komponentni blizanci ili blizanci dijelova:** Komponentni blizanci su osnovna jedinica digitalnog blizanca, najmanji primjer funkcionalne komponente. Blizanci dijelova su otprilike isto, ali se odnose na komponente nešto manje važnosti.
- **Blizanci skupova:** Kada dvije ili više komponenti rade zajedno, one tvore ono što je poznato kao skup. Blizanci skupova omogućuju proučavanje interakcije tih komponenti, stvarajući bogatstvo podataka o performansama koje se mogu obraditi i pretvoriti u korisne uvide.
- **Blizanci sustava ili jedinica:** Omogućuju uvid u ukolnosti kako se različita sredstva spajaju da bi tvorila cjelokupni funkcionalni sustav. Sustavni blizanci pružaju uvid u interakciju sredstava i mogu predložiti poboljšanja performansi.

- **Procesni blizanci:** Procesni blizanci, na makro razini povećanja, otkrivaju kako sustavi rade zajedno kako bi stvorili cjelokupni proizvodni pogon. Daju uvid u to jesu li svi sustavi sinkronizirani za rad na vrhunskoj učinkovitosti ili će kašnjenja u jednom sustavu utjecati na druge. Procesni blizanci mogu pomoći u određivanju preciznih vremenskih shema koje u konačnici utječu na ukupnu učinkovitost.

Na slici 3. je prikazan digitalni blizanac sustava koji čine industrijski roboti, pokretna traka i ostali elementi jednog dijela postrojenja pomoću kojeg se nadgledaju radnje u stvarnom okruženju. Ako bi taj digitalni blizanac bio povezan s modelom jednog ili više modela ostalih dijelova postrojenja i tako tvorili cjelokupni proizvodni pogon, skup bi činio procesnog digitalnog blizanca.



Slika 3. Digitalni blizanac sustava [2]

2.4. Laboratorij za projektiranje izradbenih i montažnih sustava

U ovom radu, naglasak je na eksperimentalnoj evaluaciji ostvarivanja rješenja dvosmjerne komunikacije temeljene na MQTT protokolu unutar stvarnog i virtualnog okruženja u sklopu Laboratorija za projektiranje izradbenih i montažnih sustava. Stoga, potrebno je detaljno sagledati cijeli laboratorij i uređaje koji koriste senzore. Laboratorij sadrži nekolicinu takvih uređaja (hladnjak, prozori i vrata s kontaktnim senzorima, pametni sustav osvjetljenja, pametni televizor i sl.). Također, povezivanje i međusobna komunikacija između uređaja i brokera koji

pohranjuje informacije o stanju uređaja u laboratoriju su ostvareni te je razvijen koncept interneta stvari pomoću Home Assistant platforme. Dakle, već postoji određen stupanj kontrole prostora laboratorija.

Međutim, idući logičan korak unaprjeđenja laboratorija je razvijanje njegove virtualne inačice koja uz sve navedeno ostvaruje i mogućnost vizualizacije sustava i događaja unutar njega. Definiranjem virtualne inačice sustava i njegovih objekata te naposljetku ostvarivanjem komunikacije, odnosno razmjene informacija o stanju unutar stvarnog i virtualnog okruženja pomoću MQTT komunikacijskog protokola u realnom vremenu, dobiva se vjerodostojan digitalni blizanac laboratorija kao produkt.

Analizom i pojašnjenjem osnovnih koncepta digitalnih blizanaca te njihovom sistematizacijom u pogledu kompleksnosti, mogućnosti i evolucije kroz vrijeme, može se zaključiti da dobiveni produkt ima sljedeće karakteristike:

- **Digitalni blizanac sustava:** Budući da se ostvaruje vizualna reprezentacija i povezivanje događaja nekolicine uređaja unutar stvarnog i virtualnog laboratorija, može se zaključiti da digitalan blizanac laboratorija pripada tom tipu.
- **Digitalni blizanac drugog stadija:** Praćenje i kontroliranje fizičkog objekta pomoću virtualne inačice i obratno čine ovog digitalnog blizanca pripadnikom drugog stadija evolucije po Gartnerovom i novom modelu evolucije.
- **Analiza uzroka:** Modeli ponašanja digitalnog blizanca reproduciraju događaje koji se događaju u njegovom fizičkom entitetu te je analiza uzroka uz kontrolu i vizualizaciju njegova primarna namjena.

Time je svrha kreiranja virtualnog modela i ostvarivanja komunikacije kompletna. U nastavku je detaljno objašnjen komunikacijski protokol zaslužan za izmjenu informacija, alati za kreiranje virtualnog modela, uređaji i rješenje interneta stvari laboratorija, integracija svega navedenog u cjelinu te naposljetku definiranje logike i smjera komunikacija u određenim situacijama.

3. MQTT PROTOKOL

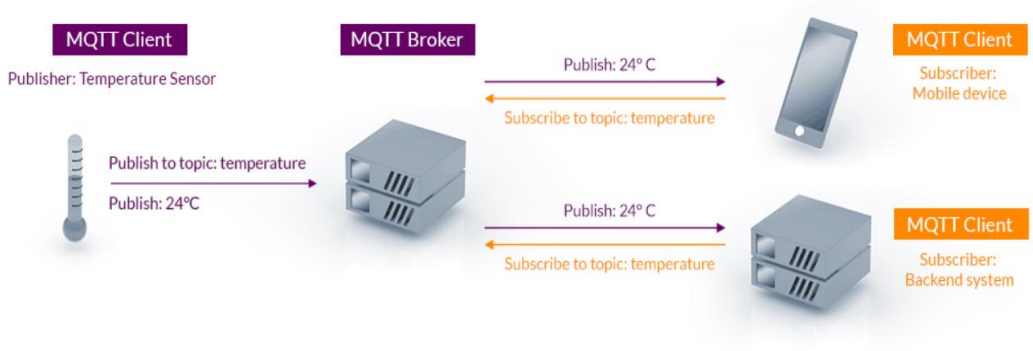
MQTT je najčešće korišteni protokol za komunikaciju u internetu stvari (IoT) zbog svoje jednostavnosti, lakoće implementacije, efikasnosti u upotrebi resursa i minimalnom opterećenju mreže. MQTT označava *MQ Telemetry Transport*. Protokol je skup pravila koji definiraju kako IoT uređaji mogu objavljivati i pretplatiti se na podatke preko interneta. MQTT se koristi za razmjenu poruka i podataka između IoT i industrijskih IoT (IIoT) uređaja, poput ugrađenih uređaja, senzora, industrijskih PLC-a, itd.

3.1. Princip rada

MQTT je baziran na modelu objavljivanja i pretplate, što znači da se uređaji mogu povezati putem tematskih kanala. Osnovni elementi su:

- **Objavljivači (*eng. Publishers*):** Uređaji koji šalju poruke na MQTT teme.
- **Pretplatnici (*eng. Subscribers*):** Uređaji koji primaju poruke s MQTT tema.
- **Teme (*eng. Topics*):** Nazivi kanala na koje se objavljuju poruke. Teme mogu biti hijerarhijski organizirane.
- **Posrednik (*eng. Broker*):** MQTT posrednik je server koji upravlja komunikacijom između objavljivača i pretplatnika.

Kada objavljivač pošalje poruku na određenu temu, MQTT posrednik (broker) distribuira tu poruku svim pretplatnicima koji su se pretplatili na tu temu. To omogućuje efikasnu, asinkronu komunikaciju između uređaja, a sve se odvija uz minimalnu mrežnu propusnost.



Slika 4. Struktura MQTT protokola [3]

3.2. Karakteristike

Kao što je već navedeno, MQTT je postao standardni protokol za razmjenu podataka IoT uređaja zbog svojih mnogih pozitivnih karakteristika. Te karakteristike su opisane u nastavku.

3.2.1. Lakoća implementacije

MQTT je dizajniran da bude jednostavan za implementaciju na različitim platformama i uređajima.

3.2.2. Efikasnost resursa

Protokol je optimiziran za korištenje malih resursa, što ga čini idealnim za uređaje s ograničenim resursima poput senzora ili mikrokontrolera. Na primjer, minimalna MQTT kontrolna poruka može biti samo dva podatkovna *byte*-a. MQTT *header*-i poruka su također mali kako bi se mogla optimizirati propusnost mreže.

3.2.3. Skalabilnost

Implementacija MQTT protokola zahtijeva minimalnu količinu koda koji troši vrlo malo energije prilikom operacija. Protokol također ima ugrađene značajke za podršku komunikaciji s velikim brojem IoT uređaja. Stoga, MQTT protokol se može implementirati za povezivanje milijunima uređaja.

3.2.4. Dvosmjerna komunikacija

MQTT omogućuje slanje poruka između uređaja i servera te obrnuto, što olakšava emitiranje poruka grupama uređaja.

3.2.5. Pouzdanost

Mnogi IoT uređaji se povezuju putem nepouzdanih mobilnih mreža s niskom propusnošću i visokom latencijom. MQTT ima ugrađene značajke koje smanjuju vrijeme ponovnog povezivanja uređaja sa serverom. Također, protokol definira tri različite razine pouzdanosti usluge, *eng. Quality of Service (QoS),* kako bi se osigurala pouzdanost isporuke poruka u upotrebi IoT-a. Te razine su:

- **QoS 0 (najviše jednom):** U ovoj razini se poruke šalju samo jednom, ali se ne jamči njihova isporuka. Šalju se bez potvrde i nema dodatnih pokušaja ako poruka nije primljena. Ta razina je najmanje pouzdana, ali pruža najmanje opterećenje na mreži.
- **QoS 1 (barem jednom):** Ova razina jamči da će poruka biti primljena, ali može biti primljena više puta. Objavljivač šalje poruku dok posrednik potvrđuje primanje. Ako se primanje ne potvrdi, poruka se ponovno šalje što može rezultirati duplim porukama, ali se jamči da poruka neće biti izgubljena.
- **QoS 2 (točno jednom):** Ova razina jamči da će poruka biti primljena samo jednom. Objavljivač i posrednik se razmjenjuju i potvrdama kako bi osigurali da je poruka primljena samo jednom. To je najpouzdanija razina, no također stvara najveći promet na mreži.

Razina QoS-a odabrana za poruku ovisi o važnosti podataka i riziku gubitka ili dupliranja poruka. Na primjer, za podatke koji se često mijenjaju, ali nisu kritični, QoS 0 može biti prikladan. Za kritične podatke, poput upravljačkih naredbi ili alarmnih stanja, preporučuje se korištenje QoS 2 radi osiguranja točne isporuke bez dupliranja.

Nadalje, MQTT sadrži podršku za *„last will and testament“* (LWT) poruke. To znači da uređaji mogu postaviti automatsko slanje poruke na određenu temu u slučaju prekida veze i time osigurati pouzdanost komunikacije u nepouzdanim uvjetima.

3.2.6. Sigurnost

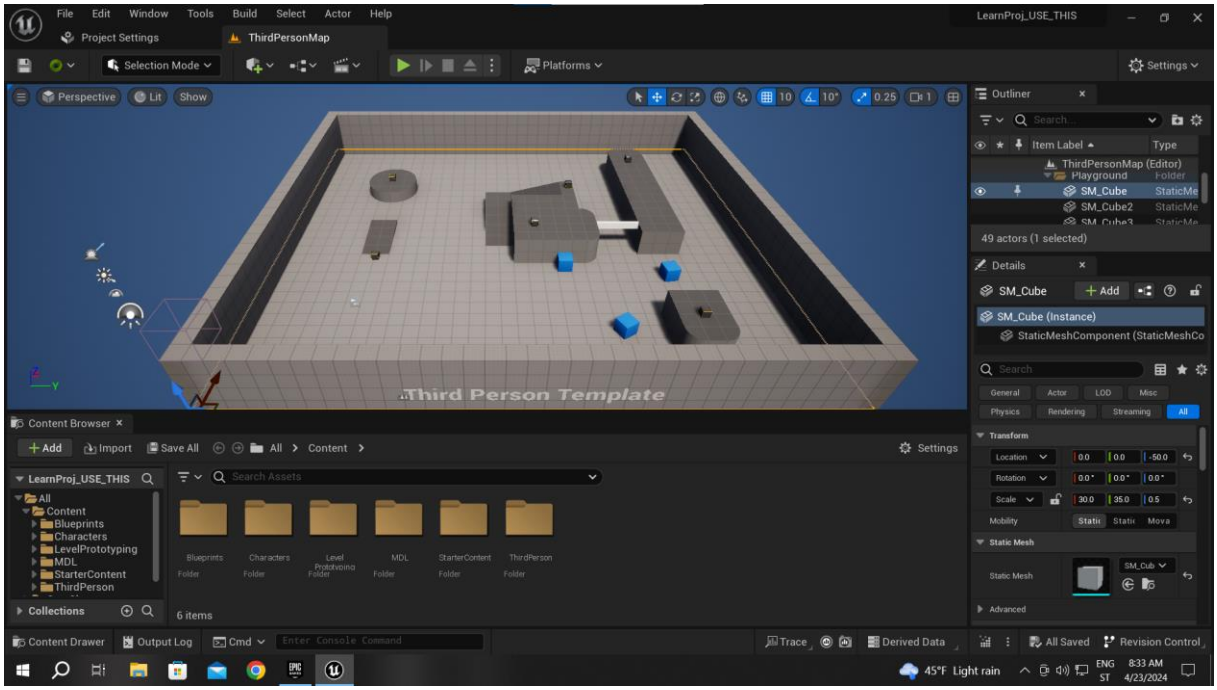
MQTT olakšava programerima enkripciju poruka i autentifikaciju uređaja i korisnika korištenjem modernih autentifikacijskih protokola, kao što su OAuth, TLS1.3, upravljani certifikati korisnika i drugi.

3.3. Primjena MQTT protokola

MQTT protokol se može koristiti u automobilskoj industriji gdje se u pametnim sustavima za nadzor vozila dijele podaci o vozilima. U industriji za kontrolu i nadzor industrijskih procesa, poput praćenja stanja strojeva i senzora. Za pametne kuće, gdje je moguće upravljanje uređajima u kući putem IoT platformi poput Home Assistant-a. Također u zdravstvu, za praćenje zdravstvenog stanja pacijenata pomoću pametnih senzora. I na kraju, za ostvarivanje komunikacije između stvarnih i virtualnih okruženja što je i tema ovoga rada.

4. UNREAL ENGINE 5

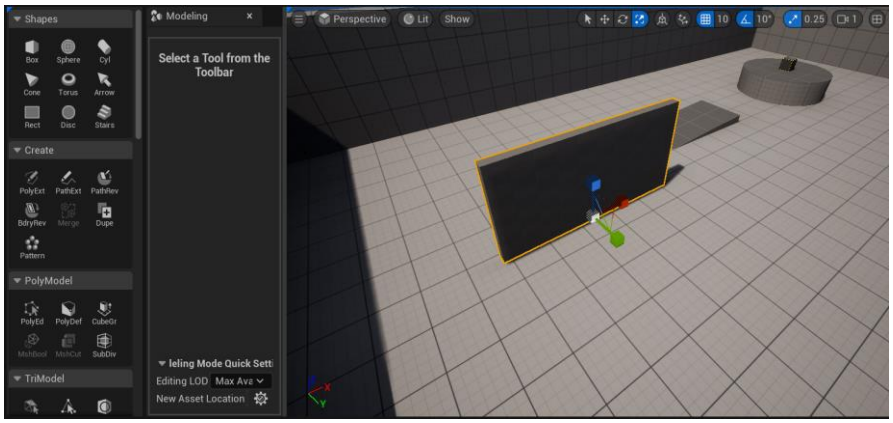
Unreal Engine 5 (UE5) je skup moćnih alata za stvaranje i interakciju s 3D objektima. Primarno razvijen od tvrtke Epic Games za razvoj video igara, ali i za širok spektar drugih primjena poput filmske produkcije, arhitektonskog dizajna, vizualizacije proizvoda, razvoja digitalnih blizanaca i još mnogo toga. U ovom poglavlju, prikazat će se detaljan opis Unreal Engine-a 5, proces stvaranja 3D objekta i mogućnosti interakcije s njima, uključujući korištenje C++ programiranja i *Blueprint* sustava.



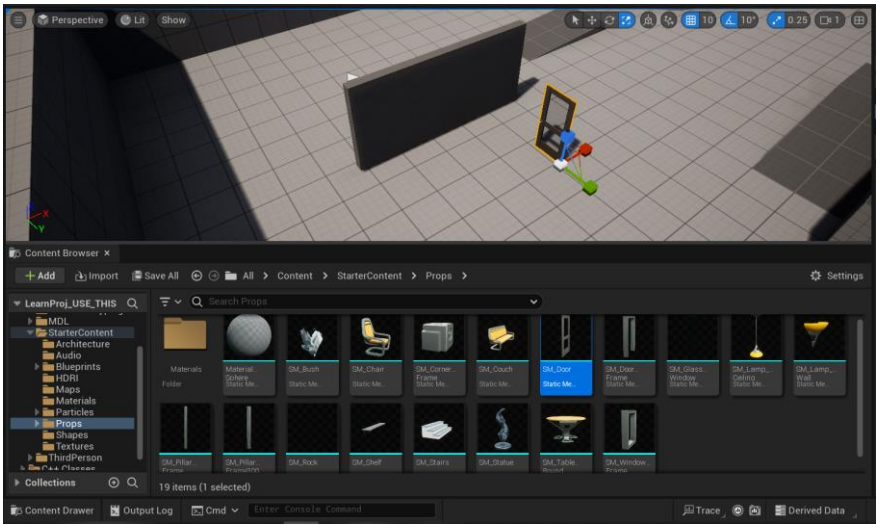
Slika 5. Početni zaslon Unreal Engine-a 5

4.1. Stvaranje 3D objekta

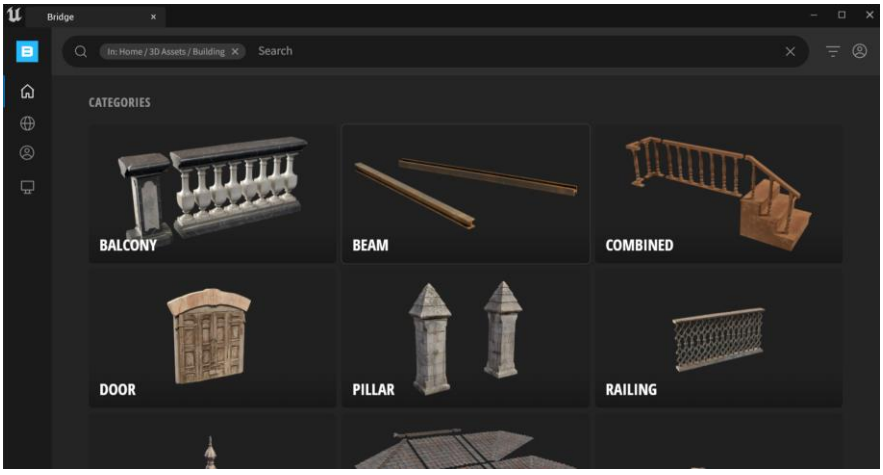
Unreal Engine 5 omogućava uvoz 3D objekta iz različitih vanjskih alata za modeliranje poput Blendera, Maya-e, 3ds Max-a ili drugih. Nakon što je model kreiran, može se uzvesti u UE5 koristeći podržane formate poput *.fbx* ili *.obj*. Međutim, sam alat nudi osnovne oblike koji se mogu uređivati te se tako mogu kreirati željeni objekti. Također, Unreal Engine 5 pruža i skup osnovnih, gotovih modela poput stolova, rasvjete i sl. koji se nalaze u *StarterContent* mapi. Nadalje, integracija s Quixel Bridge knjižnicom pruža pristup velikom broju visokokvalitetnih 3D modela, tekstura i materijala.



Slika 6. Uređivanje osnovnog oblika *Box*



Slika 7. Početni modeli unutar *StarterContent* mape



Slika 8. Quixel Bridge knjižnica

4.1.1. Tipovi objekta

Nadalje, potrebno je navesti i objasniti osnovne tipove koji će služiti kao grafički prikaz ili s kojima će se vršiti interakcija. Objekti čije je postavljanje prikazano u poglavlju 4.1. su definirani kao *StaticMesh*. To je osnovni tip koji može djelovati samostalno i tako predstavljati statičan model koji se koristi za grafičko prikazivanje objekata (zid zgrade, stol i sl.) te u tom slučaju s njime nema interakcije. Međutim, ako se s predviđenim modelom želi obavljati interakcija, tada taj osnovni tip postaje jedinica hijerarhijski višeg objekta. Ti osnovni, interaktivni objekti su:

- **Glumac (*eng. Actor*):** Tip objekta koji može biti postavljen ili stvoren u određenom trenutku u projektu te biti programiran da odrađuje željenu funkciju.
- **Pijun (*eng. Pawn*):** Tip Glumca koji može biti zaposjednut te primati naredbe pomoću kontrolera.
- **Lik (*eng. Character*):** Tip Pijuna koji ima mogućnost hodanja kroz prostor.

4.2. Interakcija s modelom

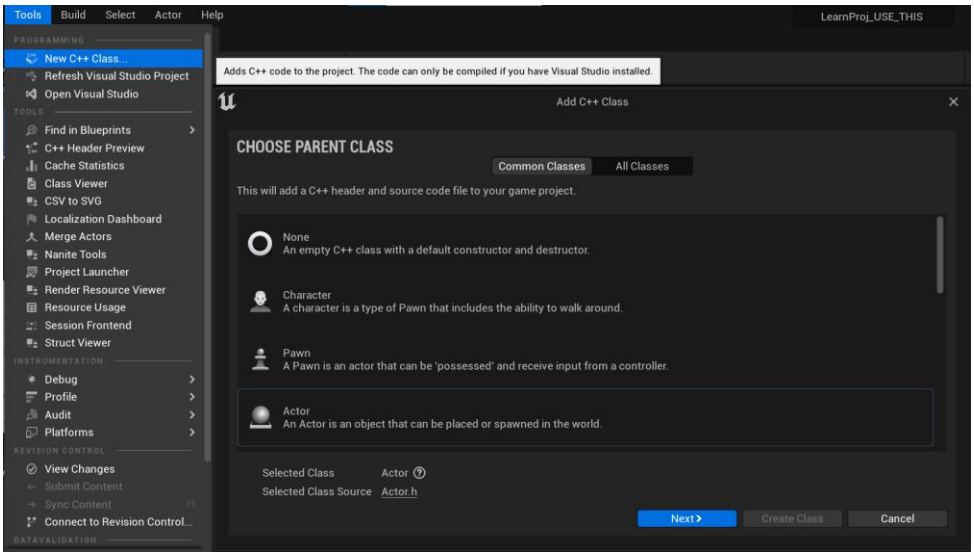
Kako bi digitalni blizanac bio vjerodostojan prikaz stvarnog objekta te kako bi se radnje unutar modela mogle odražavati na radnje u fizičkom prostoru i suprotno, potrebna je mogućnost interakcije s modelom. Unreal Engine 5 pruža različite načine interakcije s 3D modelima, prilagođene potrebama i razinama vještina korisnika. Dva glavna načina interakcije su putem:

- C++ programiranja,
- *Blueprint* sustava.

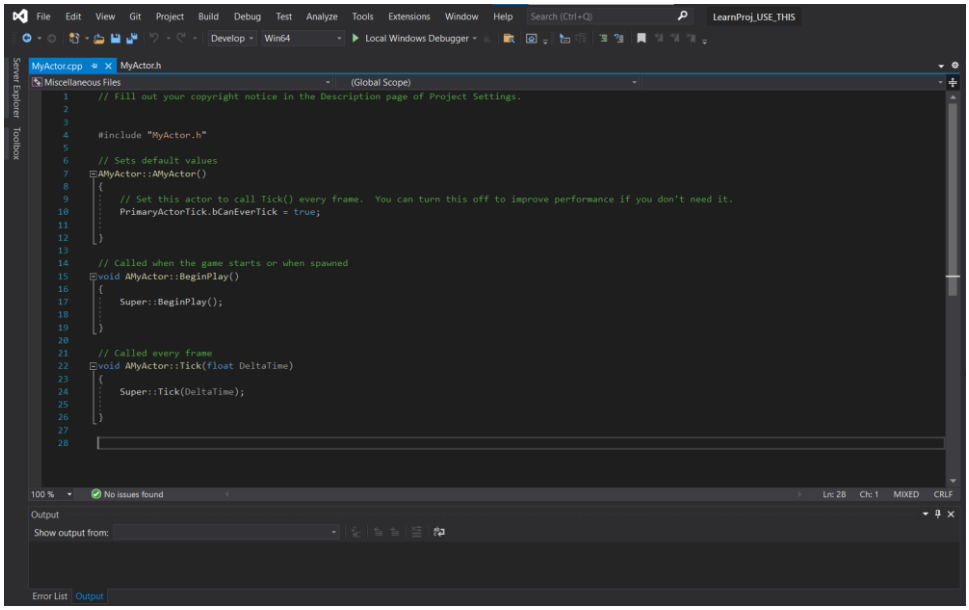
4.2.1. C++ programiranje

C++ omogućava razvoj naprednih funkcionalnosti i kompleksnih interakcija s modelima. Razvijanje u C++-u pruža potpunu kontrolu nad ponašanjem modela, omogućavajući implementiranje složenih animacija, dodavanje fizike, optimiziranje performansi ili čak integriranje umjetne inteligencije za interaktivne objekte. Programiranje pomoću C++ jezika unutar Unreal Engine-a 5 je slično standardnom C++-u koristeći klase, funkcije i varijable koje se definirane standardnim C++ sintaksama. Prilikom kreiranja C++ klase, stvara se specifičan *template* za svaki tip objekta te se na taj načina postavlja polazna točka programiranja zajedno

s nužnim elementima. Također, potrebno je napomenuti da UE5 zahtijeva prethodno instaliran Visual Studio.



Slika 9. Kreiranje C++ klase



Slika 10. Osnovni *template* za kreiranu klasu

4.2.2. Blueprint programiranje

Blueprint je vizualni sustav programiranja u Unreal Engine-u 5 koji omogućava korisnicima da dodaju funkcionalnost, stvaraju logiku i interakcije između različitih elemenata aplikacije bez pisanja koda. To je moćan alat koji omogućava brzo prototipiranje, iteraciju i implementaciju funkcionalnosti na intuitivan način. Budući da će se u ovome radu interakcije s 3D modelima programirati isključivo putem *Blueprint* sustava, isti je detaljno opisan u nastavku.

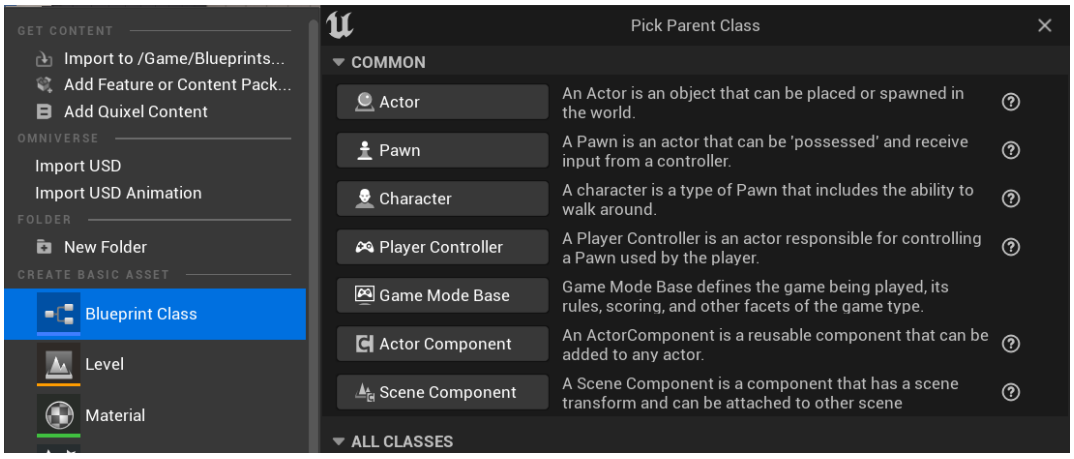
4.3. Blueprint programiranje

Blueprint sustav je alat za vizualno programiranje koji stvara klase, funkcije i varijable u Unreal Engine-u 5. Te se klase mogu zatim izvršiti povezivanjem različitih *node*-a. C++ klase koriste se kao baza za *Blueprint* klase. Odnosno, iza svakog *node*-a u *Blueprint* sustavu je definiran C++ kod koji je različit za svaki tip *node*-a. Objekti definirani za interakciju pomoću *Blueprint*-a generalno se nazivaju samo *„Blueprint*-i*“*.

4.3.1. Kreiranje Blueprint-a

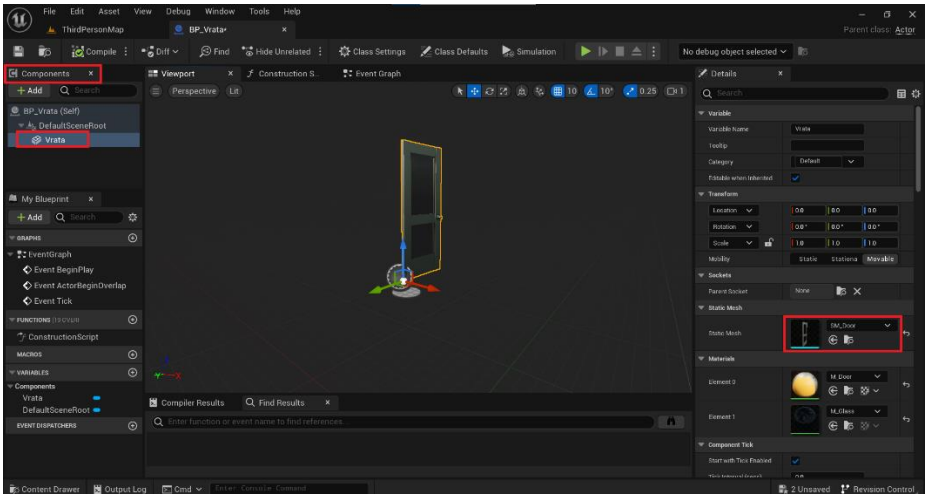
U nastavku su objašnjene osnovne komponente i izgled *Blueprint*-a te koraci potrebni za stvaranje interakcije sa željenim, njime definiranim predmetom kako isti ne bi predstavljao samo statičan model koji se koristi isključivo za grafičko prikazivanje.

Actor Blueprint. Isto kao i kod C++ programiranja interakcija, u početku se definira tip *Blueprint*-a, odnosno predmeta koji će se nalaziti u *Blueprint*-u. Najčešće je to *Actor Blueprint* budući da taj tip može biti stvoren ili postavljen u okoliš te biti programiran.



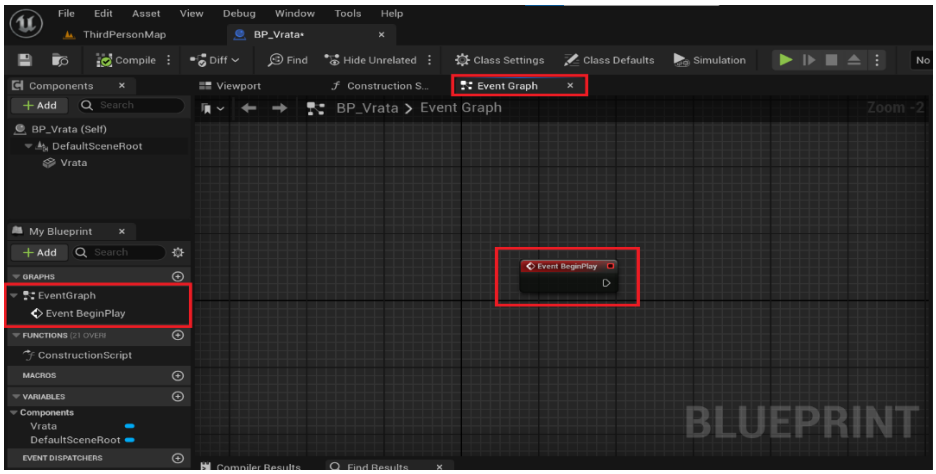
Slika 11. Kreiranje *Blueprint* klase

Components. Dio *Blueprint*-a su njegove komponente, odnosno željeni, interaktivni predmeti koji su definirani unutar njega. To može biti prethodno spomenuti *StaticMesh* koji je skup geometrije kojemu se može jednostavno dodati izgled. Također, to mogu biti komponente poput *Collision*-a koje uvjetuju radnju preklapanjem objekata ili *Camera-e,* koja definira pogled na željeni predmet.



Slika 12. Dodavanje *StaticMesh* komponente u *Blueprint*

EventGraph. Dio *Blueprint*-a koji sadrži logiku koja se pokreće na temelju događaja ili akcija koje se događaju u simulaciji. S *node*-ovima koji su dio te logike i predstavljaju događaj (npr. početak simulacije, pritisak tipke, sudar s objektom) se povezuju akcije koje treba izvršiti (npr. pokretanje animacije, promjena boje objekta). Na slici 13. je prikazan *Event BeginPlay* na koji se povezuju funkcionalnosti koje se žele obaviti pri početku simulacije.

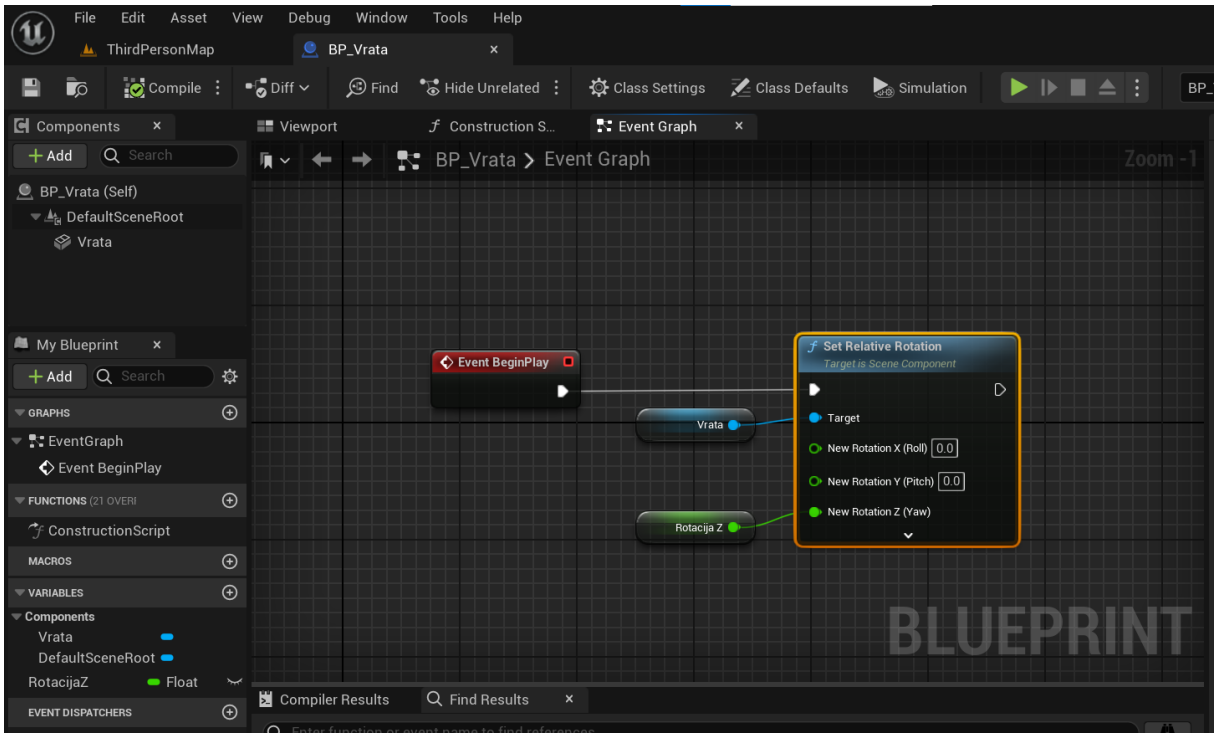


Slika 13. *Event Graph*

Functions. Funkcije su skupovi operacija ili akcija koje se mogu izvršiti ili pozvati preko drugog *node*-a unutar *Blueprint*-a. Omogućuju ponovnu upotrebu koda i organizaciju logike.

Variables. Varijable čuvaju vrijednost ili referenciraju objekt ili glumca u okolišu. Ta svojstva mogu biti dostupna samo *Blueprint*-u koji ih sadrži iznutra ili ih se može učiniti dostupnima izvana tako da se njihove vrijednosti mogu mijenjati radeći s drugim *Blueprint*-ima.

Na slici 14. je prikazan jednostavan primjer programiranja interakcije s objektom. Interakcija je povezana tako da kada se pokrene simulacija, pokreće se događaj *Event BeginPlay* koji poziva funkciju *SetRelativeRotation* koja služi za rotaciju objekta i cilja *StaticMesh* komponentu „Vrata“ te koristi „RotacijaZ“ varijablu koja sadrži *float* vrijednost 90. Drugim riječima, pokretanjem simulacije se vrata zakreću za 90°.



Slika 14. Primjer jednostavne interakcije

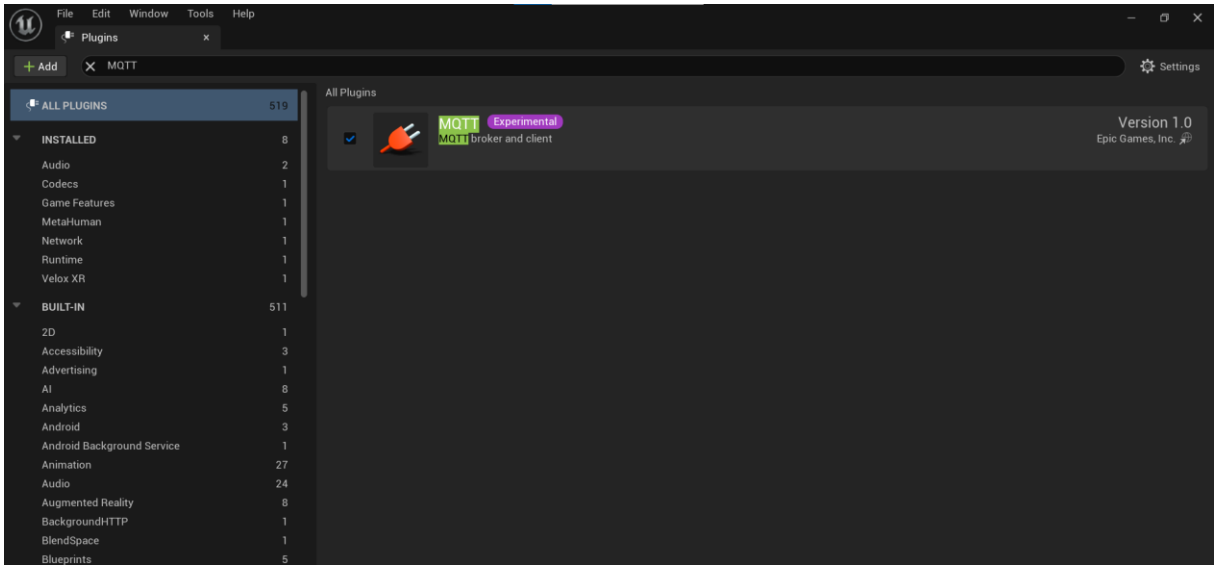
Level Blueprint. Na kraju, treba napomenuti kako postoji i *Level Blueprint*. To je poseban tip *Blueprint*-a koji služi za kontroliranje cijele razine, a ne samo pojedinih klasa. U njemu se definiraju radnje poput međusobne interakcije više *Actor*-a, postavljanje početnih uvjeta koji se odnose na razinu, reakcije na određene događaje i sl.

Nakon detaljnog opisa *Blueprint* programiranja, slijedi opis implementacije MQTT protokola unutar Unreal Engine-a 5 te njegovih komponenti unutar *Blueprint* sustava.

4.4. Unreal Engine 5 i MQTT protokol

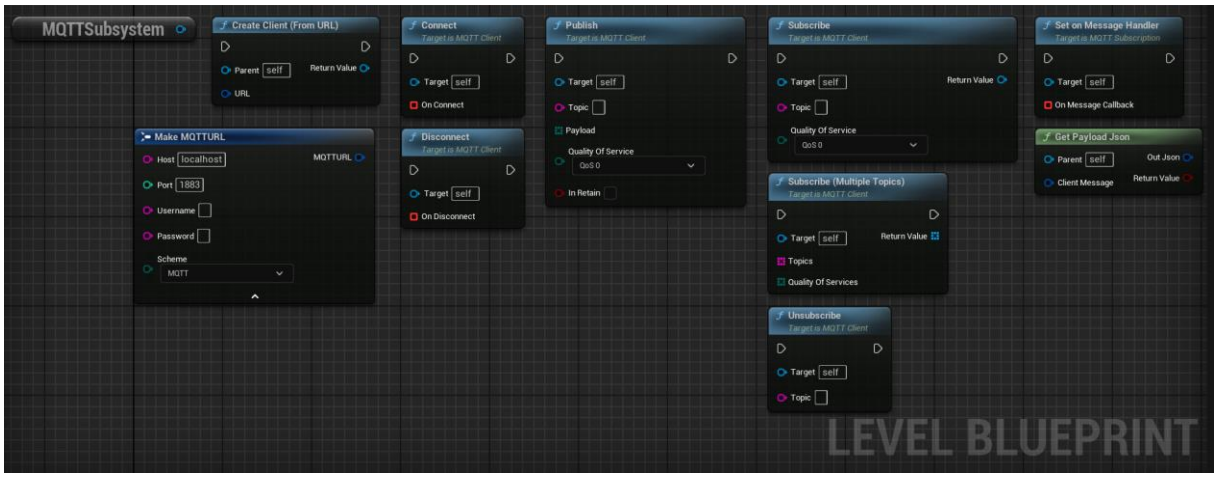
Tijekom vremena, Unreal Engine se razvio u softver s raznolikim mogućnostima. Od razvoja video igara do filmske produkcije, arhitektonskog dizajna, vizualizacije proizvoda te razvoja digitalnih blizanaca. Međutim, mnogi alati potrebni za navedene primjene nisu obuhvaćeni u osnovnim instalacijskim postavkama. Stoga, potrebno ih je postaviti u obliku *plugin*-a.

Dakle, kako bi se omogućilo dohvaćanje podataka sa servera na koji IoT uređaji šalju informacije i slanje podataka na isti te time ostvarila komunikacija i uzročno-posljedična veza između elemenata u digitalnom blizancu i elemenata u stvarnom okruženju, potrebno je preuzeti i instalirati MQTT *plugin* za Unreal Engine 5.



Slika 15. Instaliranje MQTT *plugin*-a

Nakon preuzimanja i instalacije potrebnog *plugin*-a, u *Blueprint* sustavu se pojavljuje mnoštvo *node*-ova za definiranje smjera komunikacije, ali i za upravljanje različitim tipovima podataka koji se koriste pri MQTT komunikaciji. Neki osnovni *node*-ovi su prikazani na slici 16. te su ukratko objašnjeni u nastavku.



Slika 16. Osnovni MQTT *Blueprint node*-ovi

Prikazani *node*-ovi su:

- ***MQTTSubsystem*:** Najviši hijerarhijski objekt koji sadrži sve potrebne informacije za MQTT protokol unutar Unreal Engine-a 5.
- ***Create Client (From URL)*:** Definira klijenta koji će služiti kao objavljivač ili pretplatnik servera.
- ***Make MQTTURL*:** Definira sve potrebne podatke (adresa servera, korisničko ime i lozinka za pristup i sl.) za pristup serveru te se spaja na funkciju *Create Client* kako bi klijent bio potpuno definiran.
- ***Connect*:** Povezuje klijenta na server te može pokrenuti radnju prilikom povezivanja.
- ***Disconnect:*** Prekida vezu klijenta sa serverom.
- ***Publish:*** Definira klijenta kao objavljivača te određuje na koju temu se šalju informacije i kojom kvalitetom usluge.
- ***Subscribe:*** Definira klijenta kao pretplatnika te određuje temu servera s koje prima podatke i kojom kvalitetom usluge.
- ***Subscribe (Multiple topics):*** Isto kao *Subscribe*, ali s određenim većim brojem tema.
- ***Unsubscribe:*** Prekida pretplatu na temu.
- ***Set on Message Handler:*** Upravlja i obrađuje primljenu poruku.
- ***Get Payload JSON:*** Izvlači JSON tip podatka iz poruke u kojemu se najčešće nalaze potrebne informacije koje se koriste u daljnjem određivanju interakcije.

5. KONCEPT INTERNETA STVARI LABORATORIJA

U nastavku je opisan koncept interneta stvari Laboratorija za projektiranje izradbenih i montažnih sustava. To se odnosi na uređaje koje laboratorij sadrži i njihove specifikacije, komunikacijske protokole koje koriste za razmjenu informacija o njihovom stanju te konfiguracija upravljanja uređaja unutar Home Assistant platforme kao rješenje ostvarivanja pametnoga prostora. Kako je navedeno, unutar laboratorija se nalazi mnoštvo pametnih uređaja. Kontaktni senzori implementirani na prozore, hladnjak i vrata koji prikupljaju informacije jesu li navedeni objekti otvoreni ili zatvoreni, sustav pametnog osvjetljenja koji se može kontrolirati na daljinu, pametni televizor, senzori topline i vlage i mnogi drugi. Međutim, u sklopu ovoga rada će se razmatrati samo uređaji poput hladnjaka i osvjetljenja u svrhu ostvarivanja komunikacije putem MQTT protokola s virtualnim modelom. Budući da navedeni uređaji primarno koriste Zigbee standard, na početku je pružen njegov kratki opis.

5.1. Zigbee

Zigbee je bežični standard za mrežno povezivanje koji je dizajniran za aplikacije uređaja s niskom potrošnjom energije (uporaba baterija) i kratkim do srednjim dometom. Omogućava robusnu i pouzdanu komunikaciju što ga čini idealnim izborom za mnoge IoT i M2M (*machine-to-machine*) scenarije. Kao otvoren standard, teoretski također omogućava miješanje implementacija različitih proizvođača, ali u praksi su Zigbee proizvodi prošireni i prilagođeni od strane proizvođača, što rezultira problemima s interoperabilnošću. Za razliku od Wi-Fi mreža koje se koriste za povezivanje krajnjih točaka s mrežama velike brzine, Zigbee podržava puno niže brzine prijenosa podataka i *mesh* umrežavanje. *Mesh* umrežavanje je mrežna topologija u kojoj svaki čvor (uređaj) u mreži može komunicirati s bilo kojim drugim čvorom direktno ili preko jednog ili više međupovezanih čvorova, što povećava domet i pouzdanost mreže kako bi se izbjegli središnji uređaji i kako bi se stvorila samopopravljajuća arhitektura. Tri tipa uređaja čine Zigbee mrežu. To su:

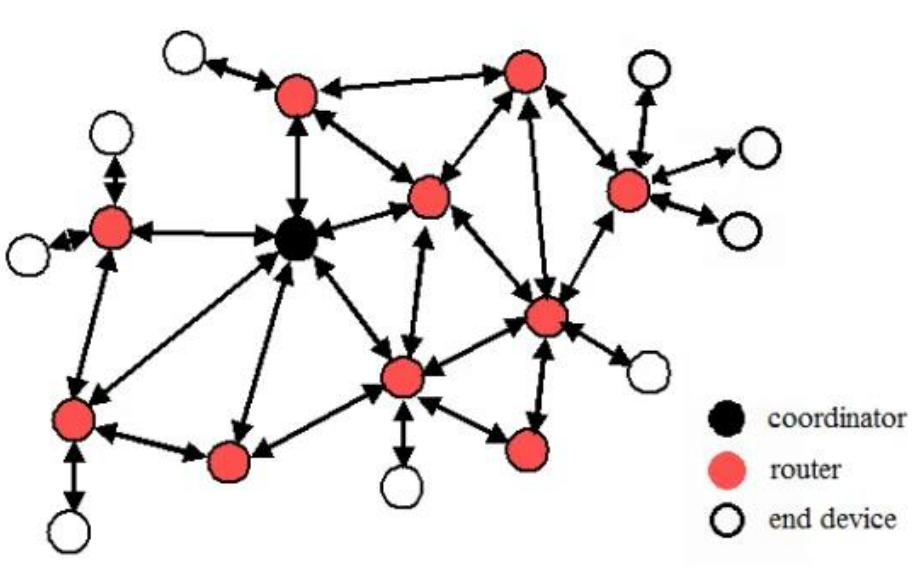
- **Koordinator:** Samo jedan centralni uređaj u Zigbee mreži čija je glavna uloga da inicira mrežu, upravlja njom i održava informacije o mrežnoj strukturi. Inicijacijom mreže se svi uređaji spajaju na koordinatora čime se formira mreža. Nakon formacije, sve informacije s iste dolaze do njega te se šalju kontrolnom sustavu.

- **Usmjerivač:** Usmjerivači proširuju domet mreže i omogućavaju komunikaciju između uređaja koji su izvan dometa direktne komunikacije, odnosno prosljeđuju podatke od krajnjih uređaja do koordinatora. Spojeni su direktno na izvor napajanja. Krajnji uređaji mogu biti i usmjerivači.
- **Krajnji uređaj:** Senzori, aktuatori ili prekidači koji prikupljaju/šalju podatke ili izvršavaju radnje. Većinom ugašeni te se aktiviraju samo pri izvršavanju funkcije.

Detaljan opis specifikacija Zigbee standarda je prikazana u tablici 2.

Tablica 2. Specifikacije Zigbee standarda

Standard	IEEE 802.15.4
Frekvencijski opseg	2.4 GHz (globalno), 915 MHz (Sjeverna Amerika), 868 MHz (Europa)
Maksimalna brzina prijenosa podataka	250 kbps (2.4 GHz), 40 kbps (915 MHz), 20 kbps (868 MHz)
Domet	10-100 metara (ovisno o uvjetima i okruženju)
Topologija mreže	Mrežna (*mesh*), zvjezdasta (star), stablasto (*tree*)
Broj čvorova u mreži	Do 65.000 čvorova (teoretski)
Sigurnost	AES-128 enkripcija
Maksimalna potrošnja energije	Vrlo niska (pogodno za uređaje na baterije)
Latencija	Niska
Protokoli	Aplikacijski protokoli kao što su Zigbee PRO, Zigbee RF4CE
Tipični uređaji	Pametni senzori, aktuatori, pametna rasvjeta, uređaji za praćenje zdravlja
Primjena	Automatizacija doma, industrijska automatizacija, praćenje imovine, zdravstvo
Interoperabilnost	Teoretski visoka, ali može biti ograničena zbog proizvođačkih prilagodbi



Slika 17. Zigbee *mesh* struktura [4]

5.2. Sonoff SNZB-04, kontaktni senzor

Sonoff SNZB-04 je bežični, kontaktni senzor koji koristi Zigbee 3.0 protokol za povezivanje. Dizajniran je za jednostavnu instalaciju pomoću dvostrane ljepljive trake (opcionalno vijci) i integraciju u pametne kućne sisteme, posebno one bazirane na Sonoff Zigbee mostu ili bilo kojem drugom Zigbee centralnom uređaju. Senzor je koristan za praćenje stanja otvoreno/zatvoreno, pružajući obavještenja ili automatizaciju kada se stanje promijeni. Zbog niske potrošnje energije, senzor koristi CR2032 3V bateriju čiji životni vijek može biti i do dvije godine. Senzor je funkcionalan na radnoj temperaturi od -10°C do 40°C te na 10% do 90% relativne vlažnosti (bez kondenzacije), a domet mu je do 80 metara na otvorenom prostoru. Konstrukcija senzora je dvodijelna. Prvi dio je plastično kućište dimenzije 47 x 27 x 13.5 mm koje sadrži elektroniku, Zigbee modul za komunikaciju te prostor za bateriju. Drugi dio je plastično kućište magneta dimenzija 32 x 15.6 x 13 mm. Kada su vrata, hladnjak ili prozori na kojima je instaliran senzor zatvoreni, oba dijela se nalaze u blizini čime se *reed* relej magnetizira. S druge strane, udaljavanjem tih dijelova se stanje senzora prepoznaje kao otvoreno. Promjenom tih stanja se pomoću Zigbee modula šalje informacija. Sonoff SNZB-04 je prikazan na slici 18.

Slika 18. Sonoff SNZB-04 [5]

5.3. Aqara Smart Wall Switch H1, pametni prekidač

Aqara Smart Wall Switch H1 je pametni zidni prekidač koji se integrira s pametnim kućnim sustavima kako bi omogućio daljinsko upravljanje rasvjetom. Prekidač koristi Zigbee 3.0 protokol za komunikaciju s kompatibilnim Zigbee uređajima, omogućavajući korisnicima upravljanje rasvjetom putem mobilne aplikacije, glasovnih asistenata (npr. Google Assistant ili Amazon Alexa) ili automatiziranih scena. Dostupan je u verzijama s jednim i dvostrukim prekidačem (*single* i *double rocker*) te u oba slučaja tipke mogu služiti za paljenje i gašenje ili se konfigurirati za dodatne funkcije poput dugog držanja i dvostrukog pritiska. Također, dostupan je u verzijama sa i bez neutralne žice za osiguravanje konstantnog napajanja prekidača. Funkcionira pri radnim temperaturama od -10°C do 40°C te na 0% do 95% relativne vlažnosti (bez kondenzacije). Maksimalno opterećenje mu je 10A pri izvedbi s jednim prekidačem čije su dimenzije 86 x 86 x 42.9 mm ili 5A po kanalu u izvedbi s dva prekidača čije su dimenzije 86 x 86 x 44.5 mm. Navedeni pametni prekidač koristi AC napajanje iz mreže (100-250V, 50/60Hz). Prikazan je na slici 19.

Slika 19. Aqara Smart Wall Switch H1 [6]

5.4. Sonoff ZBDONGLE-P, Zigbee koordinator

Sonoff ZBDONGLE-P je Zigbee koordinator koji omogućuje komunikaciju između Zigbee uređaja i pametnih kućnih sustava. Ovaj uređaj je ključan za integraciju Zigbee uređaja u sustav pametnog prostora, omogućavajući kontrolu i nadzor nad različitim uređajima kao što su pametne žarulje, utičnice, senzori i mnogi drugi. Primarno osmišljen za Home Assistant, ovaj koordinator također služi kao poveznica između Zigbee uređaja i navedenog sustava koji omogućuje korisniku upravljanje istima tako da se povezuje na računalo na kojemu je instaliran Home Assistant putem USB priključka te konfigurira za navedenu funkciju. Osim za Home Assistant, koordinator se može koristiti i za druge sustave pametnih prostora putem Zigbee2MQTT protokola. Navedeni koordinator podržava upravljanje do 40 uređaja, od kojih 21 može biti direktno upravljano. Međutim, instalacijom drugog *firmware-a* putem USB-a, teoretski se može podržati upravljanje i do 200 uređaja. Koordinator je prikazan na slici 20.

Slika 20. Sonoff ZBDONGLE-P [7]

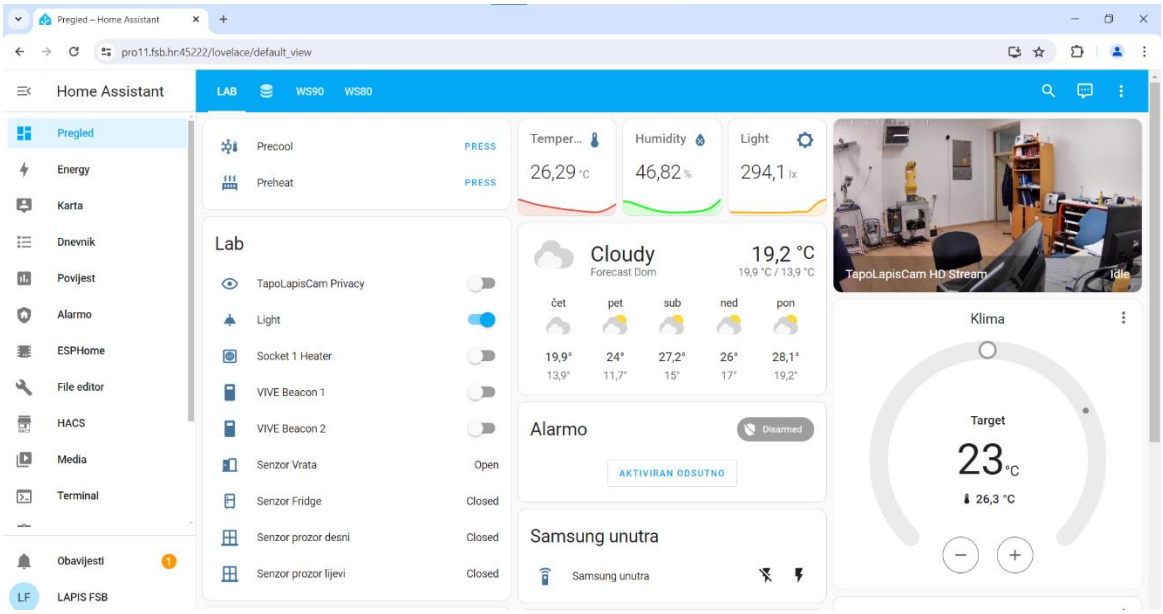
5.5. Home Assistant

Home Assistant je *open-source* platforma za pametne prostore koja omogućuje centraliziranu kontrolu i automatizaciju širokog spektra uređaja i sustava. Platforma je dizajnirana da bude fleksibilna, prilagodljiva i korisna za napredne korisnike koji žele visoku razinu kontrole nad svojim pametnim prostorom.

Ključne značajke Home Assistant platforme su:

- ***Open-Source*, lokalna kontrola i sigurnost:** Potpuno *open-source* platforma što znači da je izvorni kod dostupan i podložan uređivanju i promjeni. Tako korisnici mogu potpuno kontrolirati svoj sustav. Također, izvršavanje je lokalno što predstavlja povećanu sigurnost i privatnost, a korisnici mogu postaviti i enkripciju i autentifikaciju za dodatan sloj sigurnosti.
- **Podrška raznih protokola i uređaja:** Podržava mnoge protokole poput MQTT, Zigbee, Zigbee2MQTT, Z-Wave, Wi-Fi, Bluetooth i sl. Također, podržava više od 1000 marki uređaja i integraciju istih unutar jednog sustava.

- **Grafičko sučelje:** Home Assistant posjeduje intuitivno, visoko prilagodljivo sučelje koje omogućuje stvaranje vlastitog izgleda nadzornih ploča pomoću kojih se kontroliraju povezani uređaji.
- **Automatizacija:** Omogućuje napredne automatizacije koristeći YAML skripte, Node-RED ili *editor* unutar platforme. Na temelju raznih uvjeta poput stanja uređaja, vremena ili drugih parametara se automatski odvijaju definirane radnje.
- **Podrška zajednice i redovita ažuriranja:** Budući da je *open-source* platforma, Home Assistant posjeduje široku, aktivnu zajednicu korisnika koja pruža redovna ažuriranja, nove značajke i dodatke.



Slika 21. Home Assistant laboratorija

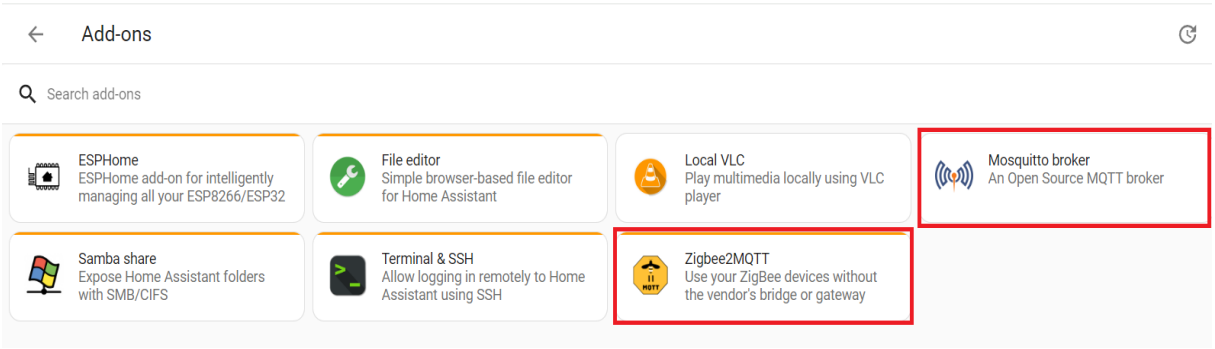
5.6. Povezivanje i konfiguracija

Nakon postavljanja kontaktnih senzora na željene uređaje i postavljanja pametnog prekidača, idući korak je instalirati Home Assistant platformu. Odabran uređaj za instalaciju navedene platforme je HP stolno računalo unutar laboratorija. Time se osigurava snažan procesor, velika količina radne memorije i veliki kapacitet za spremanje podataka uz mogućnost obavljanja više radnji istovremeno. Računalo koristi Debian operativni sustav te koristi *ethernet* kabel za sigurnu i kvalitetnu vezu. Detaljna instalacija Home Assistant-a se može pronaći na službenoj web stranici [8].

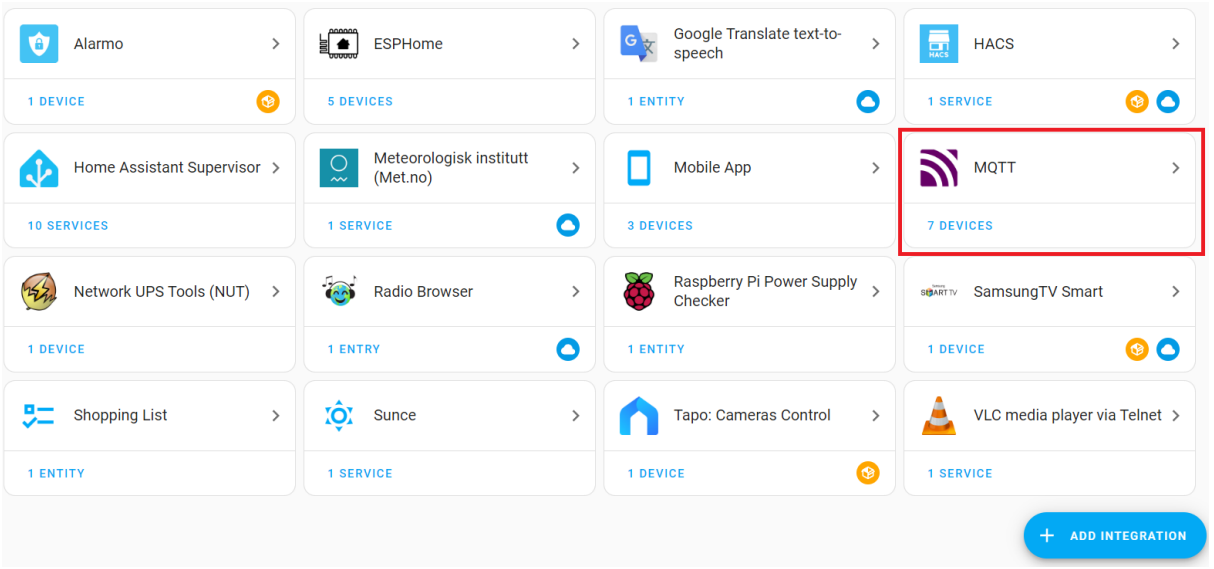
Nakon toga je potrebno povezati i konfigurirati Zigbee koordinatora i uređaje. To se postiže navedenim redom:

- Pokretanje Home Assistant sustava i instaliranje potrebnog Zigbee2MQTT *addon*-a i MQTT integracije kako bi Home Assistant mogao komunicirati sa Zigbee koordinatorom i uređajima.
- Preuzimanje Mosquitto broker *addon*-a. Njegovom instalacijom se postavlja MQTT broker kao dio Home Assistant sustava. Tako Home Assistant posjeduje broker (objašnjen u poglavlju 3.1.) i može distribuirati MQTT poruke. Također, tim dodatkom se omogućuje postavljanje autentifikacije i autorizacije određene korisničkim imenom i lozinkom za pristup brokeru.
- Postavljanje Sonoff ZBDONGLE-P koordinatora. Navedeni koordinator se uključuje u računalo, povezuje s Home Assistant platformom i konfigurira da inicijalizira Zigbee mrežu. Zigbee2MQTT *addon* se koristi za upravljanje koordinatorom.
- Povezivanje Zigbee uređaja. Nakon inicijalizacije mreže, Zigbee uređaji se postavljaju u način uparivanja. Ovisno o uređaju, to se postiže pritiskom određene tipke ili sekvenci tipki. Nadalje, pomoću navedenih integracija se inicira proces dodavanja novih uređaja nakon čega koordinator prepoznaje uređaje i dodaje ih u Zigbee mrežu.
- Detaljna konfiguracija. Nakon povezivanja, odvija se konfiguracija pomoću navedenih integracija i *addon*-ova te se omogućuje komunikacija i upravljanje uređaja unutar Home Assistant sustava.

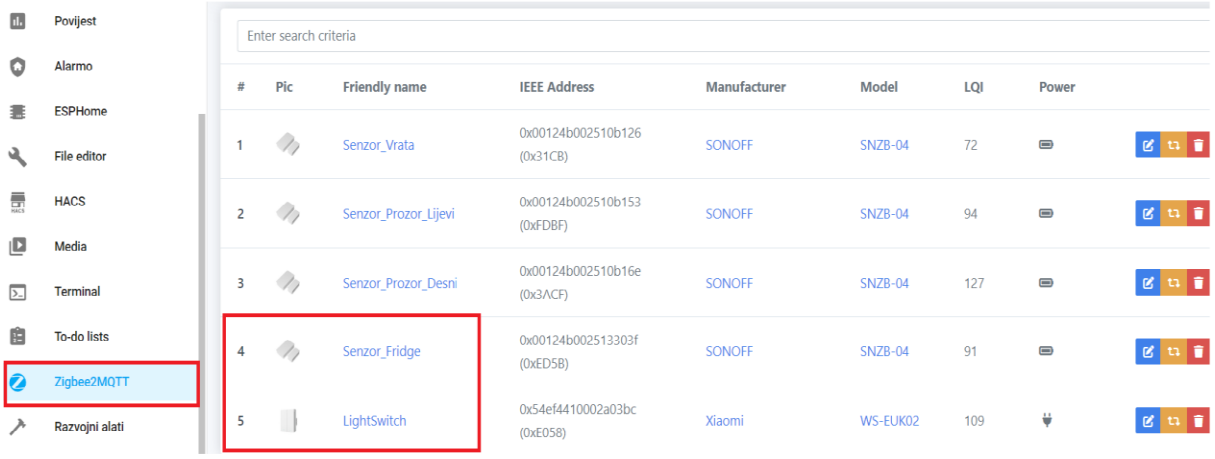
Nakon povezivanja i konfiguracije Zigbee koordinatora i uređaja te time omogućivanja njihove integracije unutar Home Assistant sustava, idući korak je postavljanje automatizacija. Definiranjem različitih okidača se pokreću druge radnje i/ili šalju poruke brokeru. Na kraju, posljednji korak je uređenje pregledne ploče, prikazane na slici 21., radi lakšeg upravljanja i pregleda stanja uređaja. Tako je ostvaren koncept interneta stvari laboratorija.



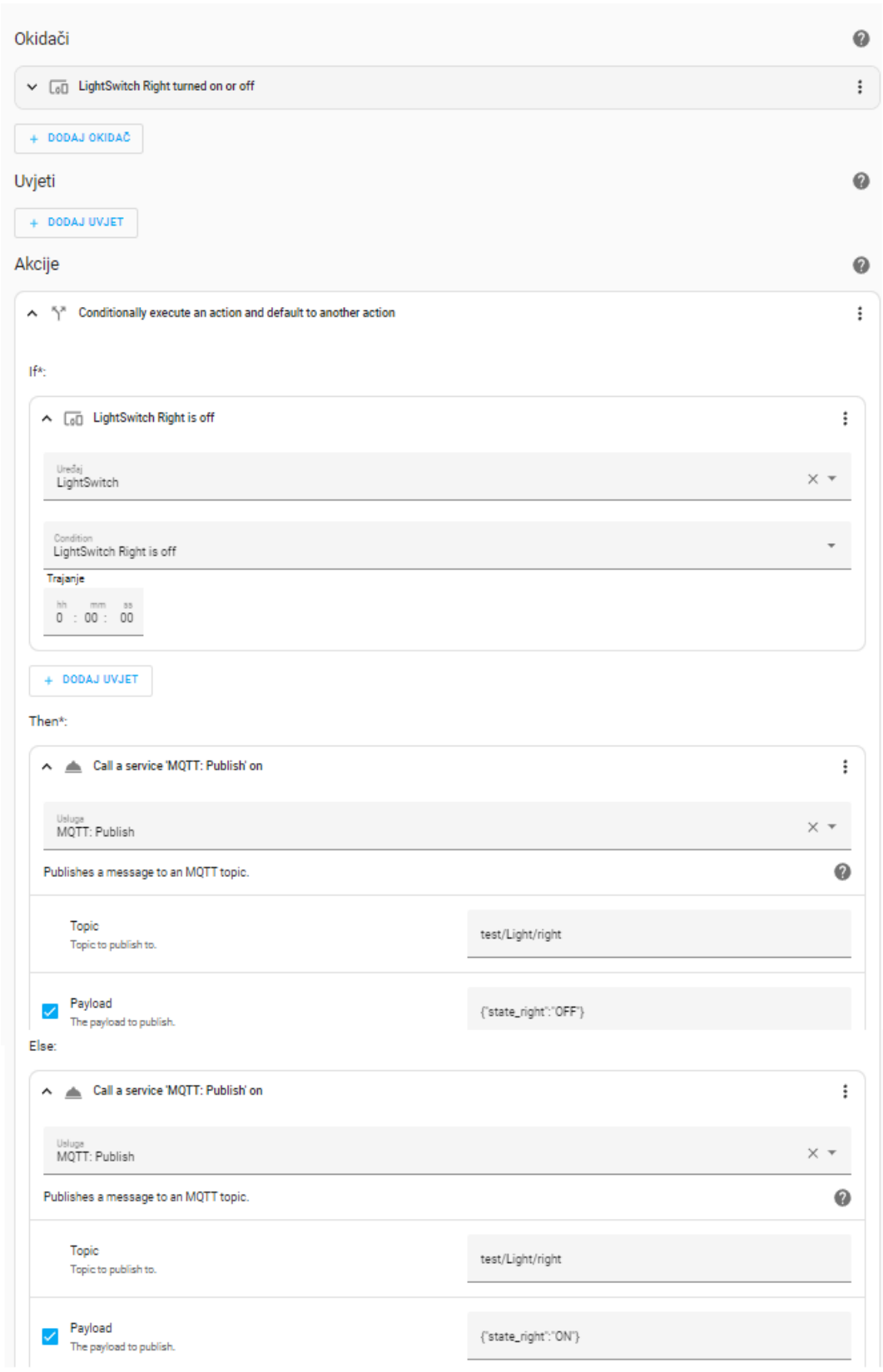
Slika 22. *Addon*-ovi



Slika 23. Integracije



Slika 24. Uređaji koji koriste Zigbee2MQTT



Slika 25. Primjer automatizacije

Tijek komunikacije je određen na sljedeći način. Podaci sa Zigbee uređaja (npr. stanje senzora) se šalju Zigbee koordinatoru koji prima te podatke i prosljeđuje ih Home Assistant sustavu. Pomoću Zigbee2MQTT *addon*-a, ti podaci se preuzimaju i pretvaraju u MQTT poruku koja se objavljuje na određenu temu Mosquitto brokera unutar sustava. Kada Mosquitto broker primi MQTT poruku, istu distribuira pretplaćenim klijentima na tu temu (Home Assistant i/ili Unreal Engine 5 što će biti slučaj u nastavku) koji je koriste za ažuriranje stanja uređaja ili pokretanje automatizacije. U obratnoj situaciji, promjenom stanja uređaja preko pregledne ploče u Home Assistant-u, isti šalje poruku na odgovarajuću temu nakon čega se pomoću Zigbee2MQTT ta tema preuzima, tumači i šalje Zigbee koordinatoru, a on je dalje prosljeđuje Zigbee uređaju.

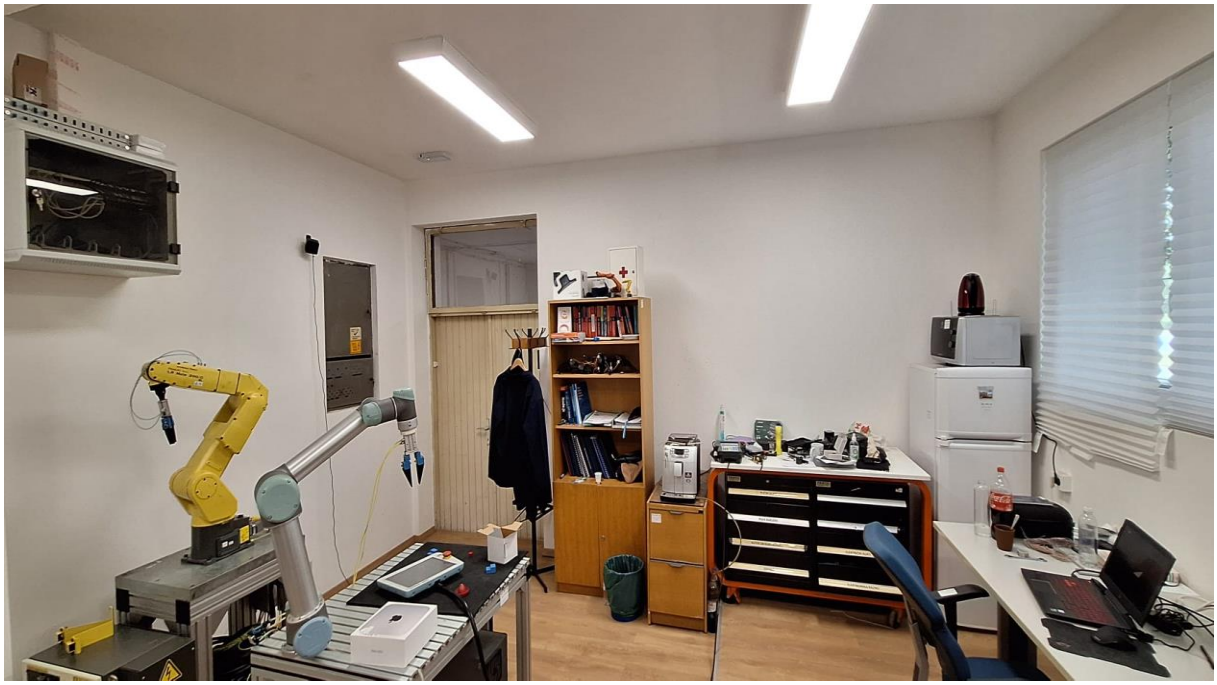
6. VIRTUALNI MODEL LABORATORIJA I UREĐAJA

U prethodnim poglavljima razmatrane su teorijske osnove digitalnih blizanaca, MQTT protokola, Unreal Engine 5 programa i kako je ostvaren koncept interneta stvari laboratorija pomoću Zigbee uređaja i Home Assistant sustava. Stoga, u nastavku se usredotočuje na praktičnu realizaciju ostvarivanja dvosmjerne komunikacije između stvarnog i virtualnog okruženja putem MQTT protokola čiji je prvi korak posjedovanje virtualnog modela laboratorija i uređaja.

Cilj ovog poglavlja je prikazati izgled Laboratorija za projektiranje izradbenih i montažnih sustava i razmještaj uređaja unutar njega kako bi se mogla kreirati što vjerodostojnija njegova inačica. Također, koji se alati koriste unutar Unreal Engine-a 5 i koji procesi za pojedini uređaj te proces izrade animacija koje će simulirati promjenu stanja istih u realnom vremenu. Dakle, kroz ovo poglavlje će se prikazati kako je moguće stvoriti funkcionalan i realističan digitalni blizanac koji služi za prikaz događaja i upravljanje laboratorijem.

Izgled laboratorija je prikazan na slikama 26. i 27.

Slika 26. Laboratorij za projektiranje izradbenih i montažnih sustava, pogled 1.

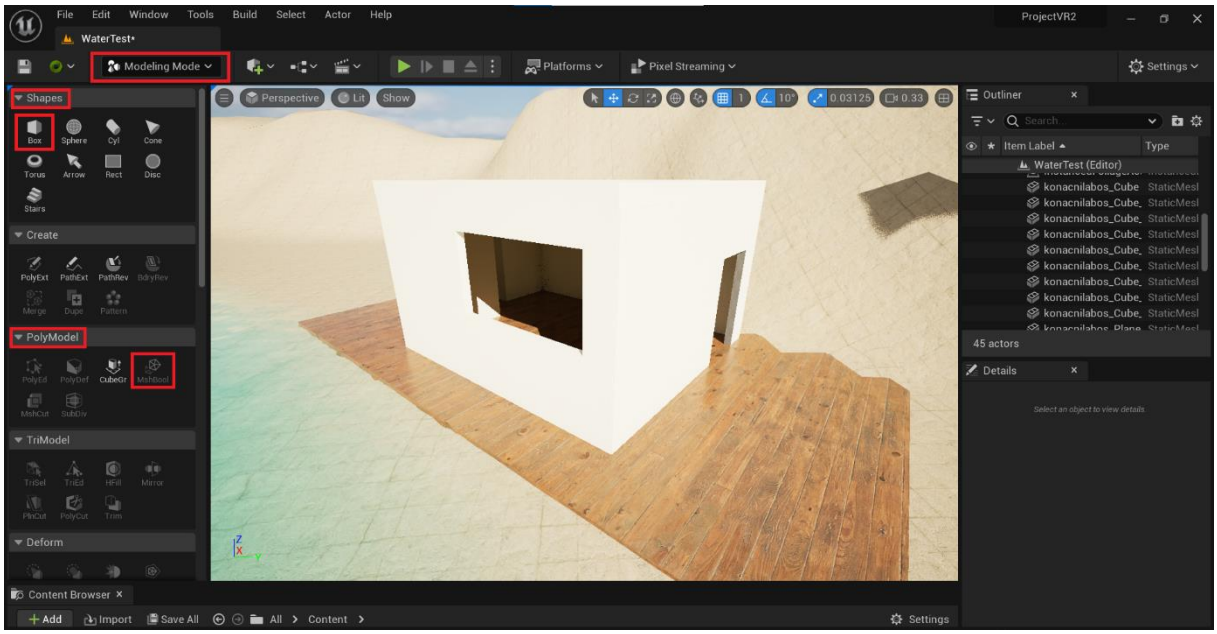


Slika 27. Laboratorij za projektiranje izradbenih i montažnih sustava, pogled 2.

6.1. Stvaranje modela

Nakon detaljnog proučavanja laboratorija, u postojeći Unreal Engine 5 projekt se implementira njegova konstrukcija modela. Tu konstrukciju čine *StaticMesh* objekti zidova, poda te stropa laboratorija. Kako je objašnjeno u poglavlju 4., to su samostalni objekti koji se koriste isključivo za grafičko prikazivanje. Objektima su promijenjene dimenzije sukladno s dimenzijama laboratorija, određeni materijali u *Materials* odjelu radi bolje vizualizacije te su postavljeni na pozicije kako bi činili osnovnu konstrukciju virtualnog modela. Budući da laboratorij posjeduje dva prozora i ulazna vrata, bilo je potrebno napraviti izreze u modelima zidovima za njihovu kasniju implementaciju.

Modeling Mode nudi mnoštvo alata i operacija za kreiranje i uređivanje 3D modela. *Shapes, Create, PolyModel* i *Transform* su samo neki od njih. Pomoću *Shapes* odjela su kreirani *Box* objekti veličine potrebnih izreza te su postavljeni na predviđena mjesta prozora i vrata. Na kraju, označavanjem zidova i spomenutih *Box* objekta i odabiranjem opcije *MshBool* unutar *PolyModel* odjela se dva *StaticMesh* objekta pretvaraju u jedan *Boolean*. Time je kostur modela upotpunjen, a prikaz je na slici 28.

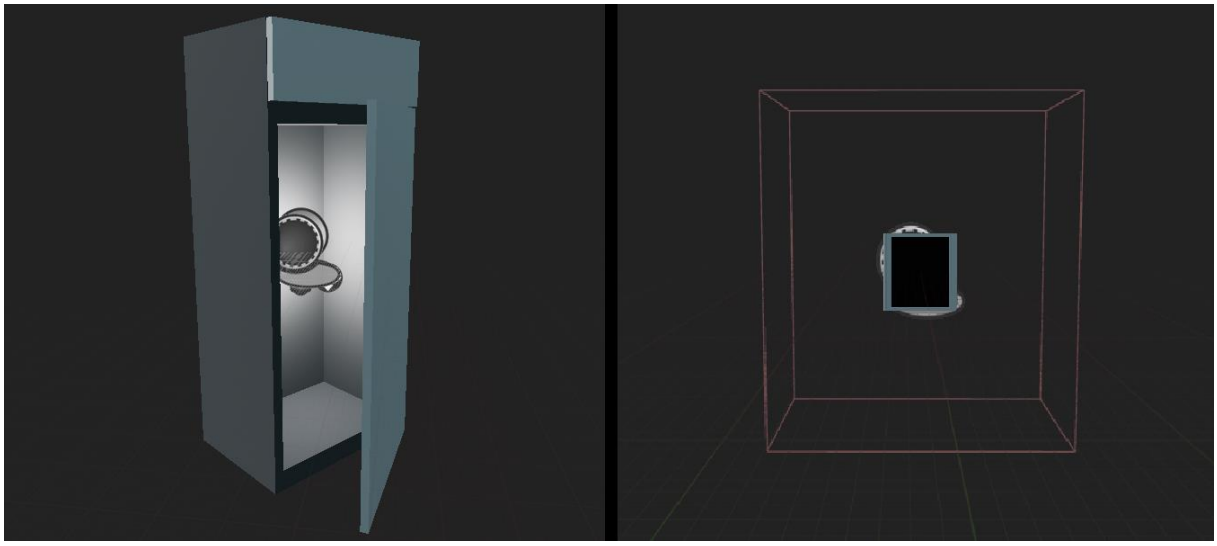


Slika 28. *Modeling Mode* i konstrukcija laboratorija

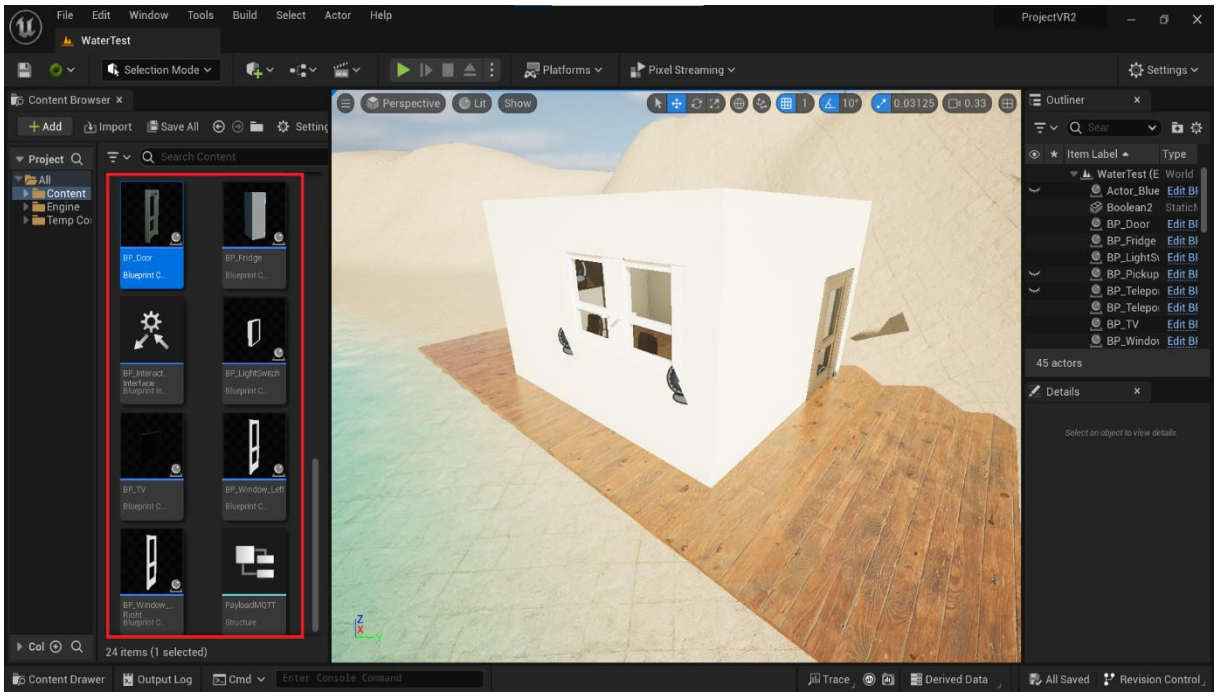
Vrata, prozori, prekidač za paljenje i gašenje svjetla, hladnjak i televizor su neki od predmeta i uređaja koji su pametni ili na sebi imaju postavljen senzor. Ti objekti posjeduju mogućnost identifikacije promjene stanja uzrokovane ljudskom interakcijom. Stoga, interakcija s njihovim virtualnim inačicama također mora biti moguća. Međutim, interakcija neće biti moguća ako su isti kreirani kao *StaticMesh* objekti. Zbog toga, prateći korake objašnjene u poglavlju 4.3., modeli se stvaraju kao *Blueprint* objekti koji sadrže *StaticMesh*-ove izmodelirane unutar programa (za televizor preuzeti s interneta u obliku *.fbx* datoteke) u *Components* odjelu kojima se definira izgled željenih predmeta. Tako se može definirati logika odvijanja animacija pomoću *Blueprint node*-ova unutar *EventGraph* odjela koja će simulirati događaje u laboratoriju ili ih pokretati nakon interakcije. Iako su svi navedeni predmeti kreirani kao *Blueprint* objekti zbog njihovih mogućnosti, samo će se s hladnjakom i prekidačem za paljenje i gašenje svjetla definirati animacije i uspostavljati komunikacija, kako je spomenuto u prethodnim poglavljima.

Nadalje, dodaju se još objekti poput stolova i stolica koji su samo neki od mnoštva predmeta u laboratoriju. Razlog njihovog dodavanja je približavanje virtualnog modela stvarnom izgledu laboratorija te s njima neće biti nikakve interakcije. Stolovi su preuzeti iz integrirane Quixel Bridge knjižnice, a stolice su preuzete u obliku *.fbx* datoteke s interneta. Na kraju, osim *StaticMesh* komponenti postoje i *PointLight* komponente. One predstavljaju osvjetljenje unutar

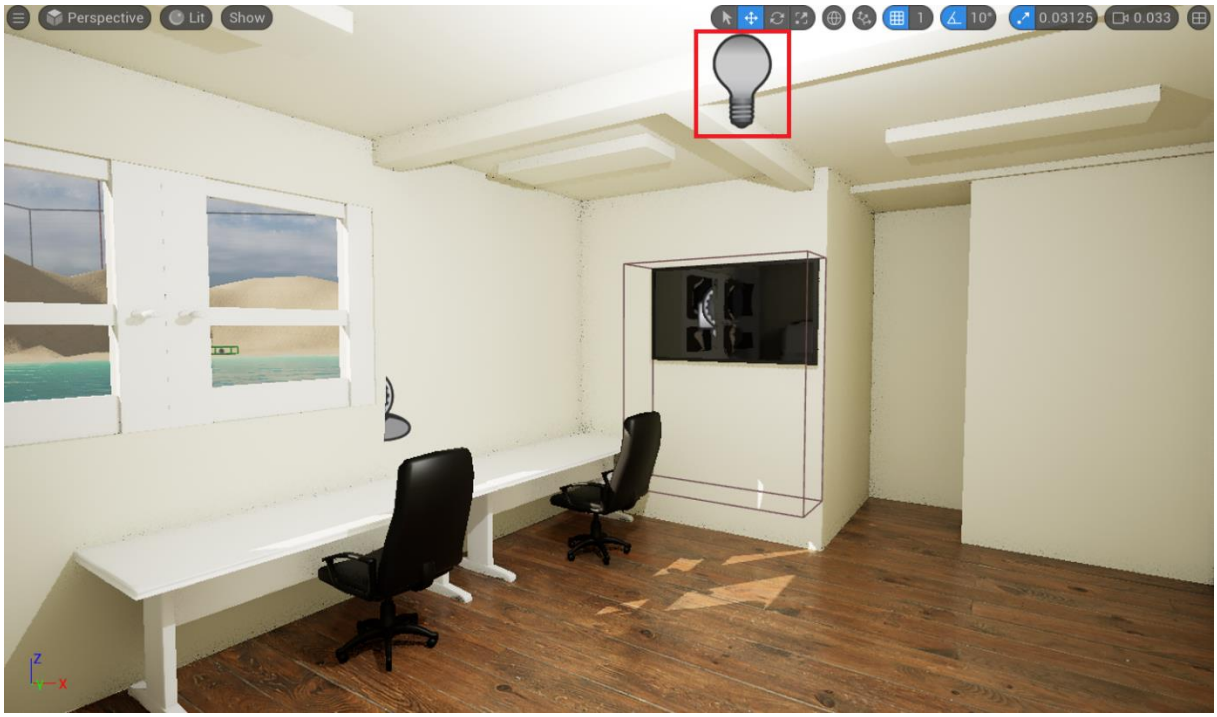
Unreal Engine 5 programa. Stoga, laboratorij posjeduje tu samostalnu komponentu na sredini stropa s kojom će *Blueprint* prekidača komunicirati nakon interakcije s njim. Finalni virtualni modeli potrebnih uređaja su prikazani na slici 29., a model laboratorija na slikama 30., 31. i 32.



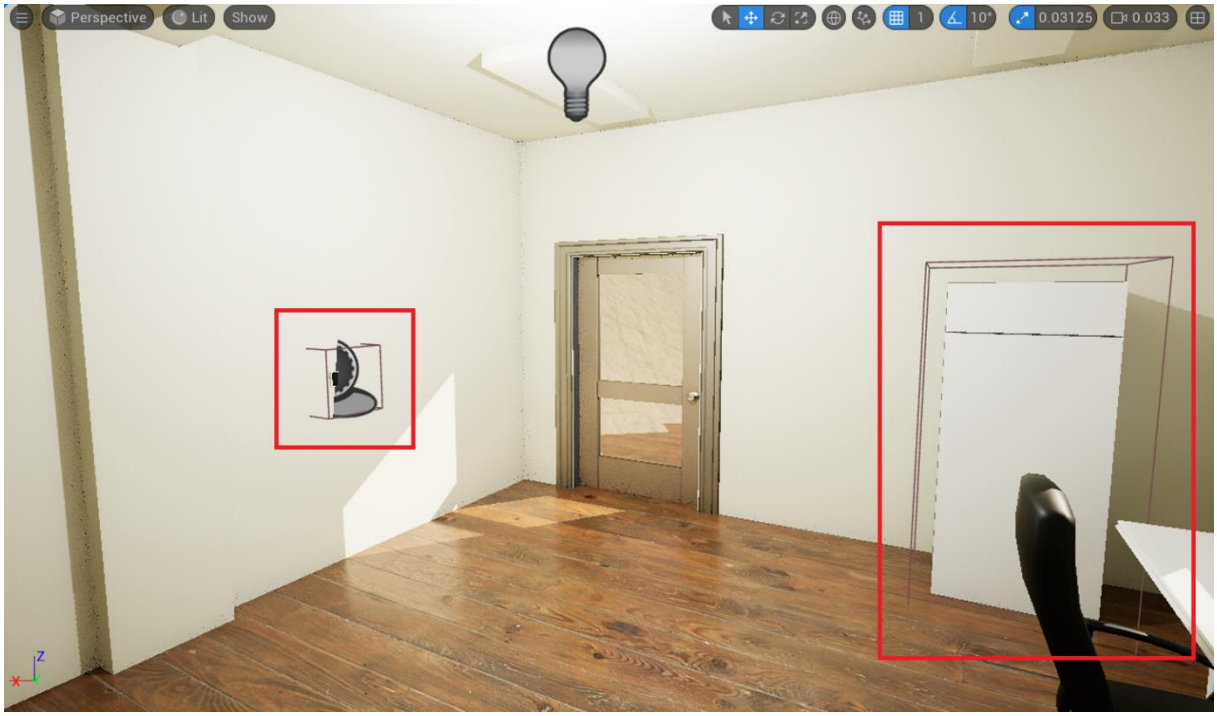
Slika 29. Modeli hladnjaka i prekidača



Slika 30. *Blueprint*-i objekta i vanjski izgleda laboratorija



Slika 31. Unutrašnjost laboratorija, pogled 1.



Slika 32. Unutrašnjost laboratorija, pogled 2.

6.2. Logika interakcije i animacija

Prije pregleda procesa stvaranja interakcije i animacija uređaja, potrebno je naglasiti da projekt posjeduje *Blueprint „BP_FirstPersonCharacter“*. Taj *Blueprint* predstavlja osobu unutar virtualnog modela te posjeduje osnovne kontrole i funkcionalnosti za kretanje kroz virtualni model. Najčešće se te kontrole sastoje u tome da se pritiscima tipki strelica tipkovnice ili WASD tipki odvija kretanje lijevo, desno, naprijed i nazad, a pomicanjem miša se upravlja pogledom, odnosno usmjerenjem kamere. Tako se osoba koja upravlja modelom može jednostavno kretati kroz model, približavati objektima i vršiti interakciju.

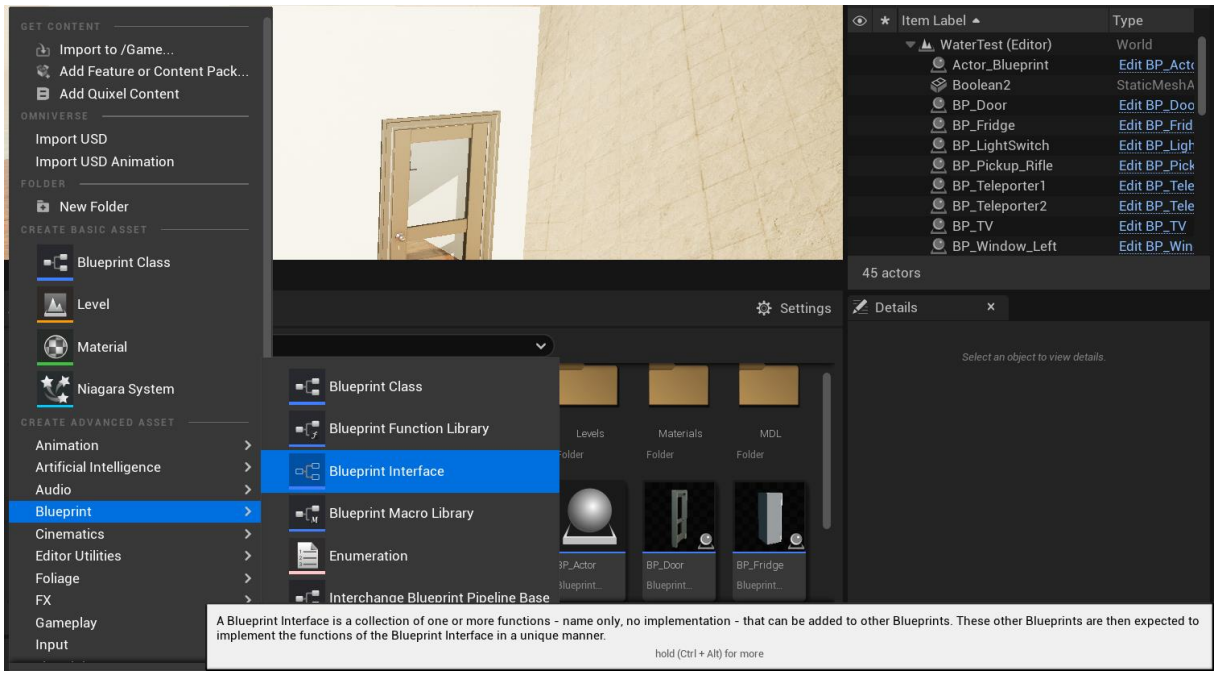
S obzirom na to, interakcije i animacije će se odvijati na sljedeći način. Kreirat će se *Blueprint Interface* koji omogućava komunikaciju između različitih *Blueprint*-a. Kada se model osobe, definiran pomoću „*BP_FirstPersonCharacter“ Blueprint*-a, približi nekom objektu i pritisne tipku *„E“* (što će se definirati u postavkama projekta) odvit će se interakcija koja će pokrenuti animacije u modelu. Međutim, interakcija će se odviti samo između modela osobe i onih *Blueprint*-a uređaja koji u svojoj strukturi implementiraju *Blueprint Interface* i posjeduju njegove funkcije kako bi ih mogli koristiti. Detaljan postupak je prikazan u nastavku.

6.2.1. Blueprint Interface

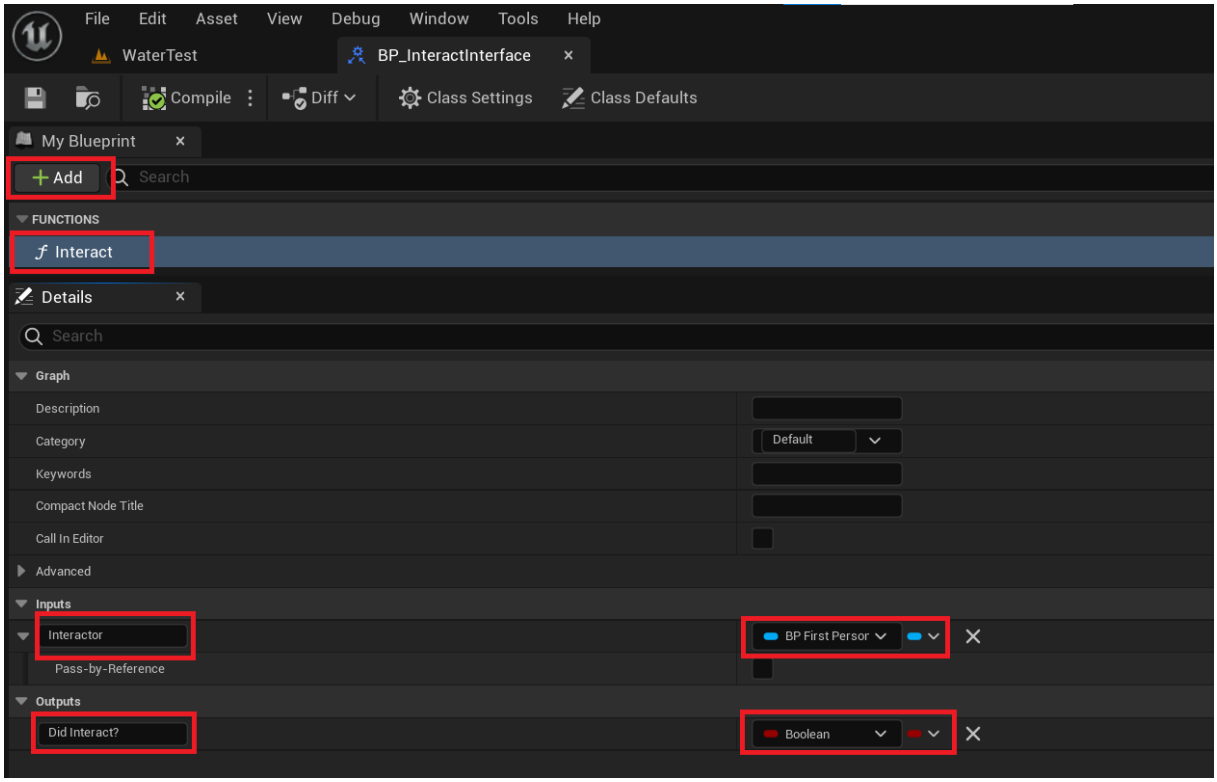
Kreiranje *Blueprint Interface*-a se obavlja desnim klikom na *Content Browser* i odabirom *Blueprints* → *Blueprint Interface* (prikaz na slici 33.). Kako nudi opis, taj *Blueprint* je skup jedne ili više funkcija, ali samo njihovih imena, koje mogu biti dodane drugim *Blueprint*-ima nakon čega se očekuje njihovo uređivanje i korištenje unutar jednih, a pozivanje unutar drugih. Navedeni *Blueprint* se naziva *„BP_InteractInterface“*. Unutar *Blueprint*-a se dodaje funkcija *„Interact“* i ulazna varijabla *„Interactor“* čija je referenca *„BP_FirstPersonCharacter“* te izlazna varijabla *„DidInteract?“* koja je *boolean* tip podatka i koja označava je li se interakcija dogodila. Prikaz je na slici 34. Time je *Blueprint Interface* kreiran.

6.2.2. Input akcija

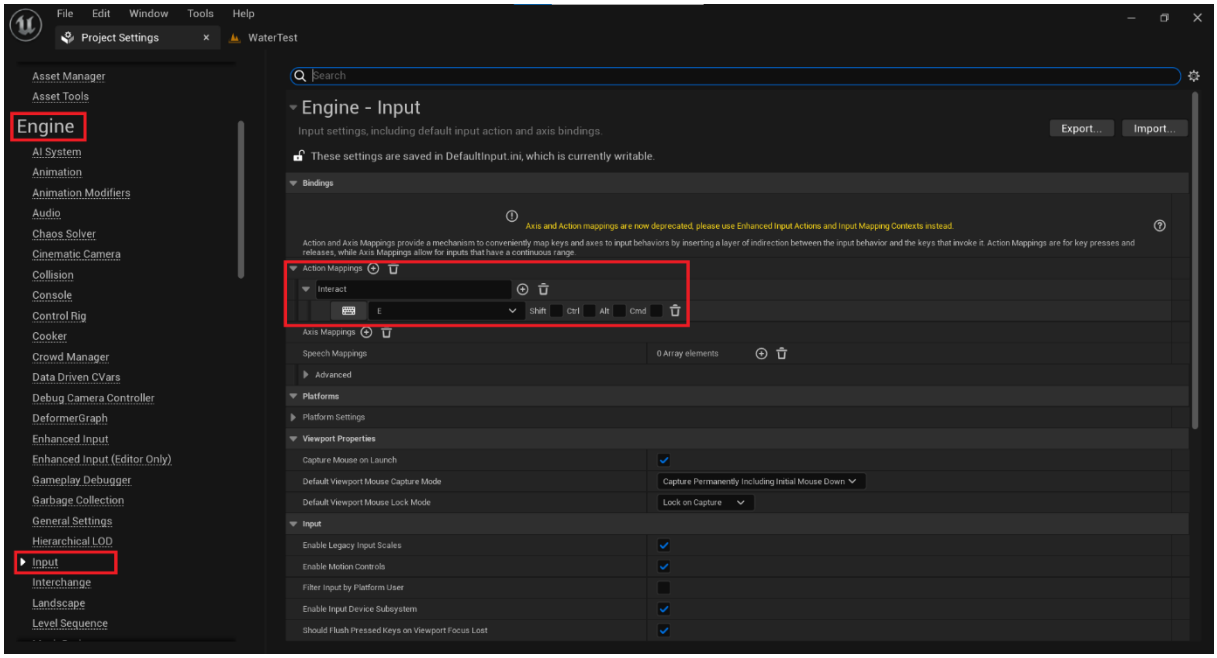
Idući korak je otvaranje postavki projekta. Unutar *Engine* odjela se odabiru *Input* postavke kako bi se definirala akcija koja se pokreće kao odgovor na unos korisnika. Drugim riječima, kreira se akcija „*Interact“* i definira se njeno pokretanje pritiskom na tipku *„E“*. Akcija će se koristiti u daljnjem definiranju logike u *Blueprint*-u *„BP_FirstPersonCharacter“*. Prikaz je na slici 35.



Slika 33. Stvaranje *Interface*-a



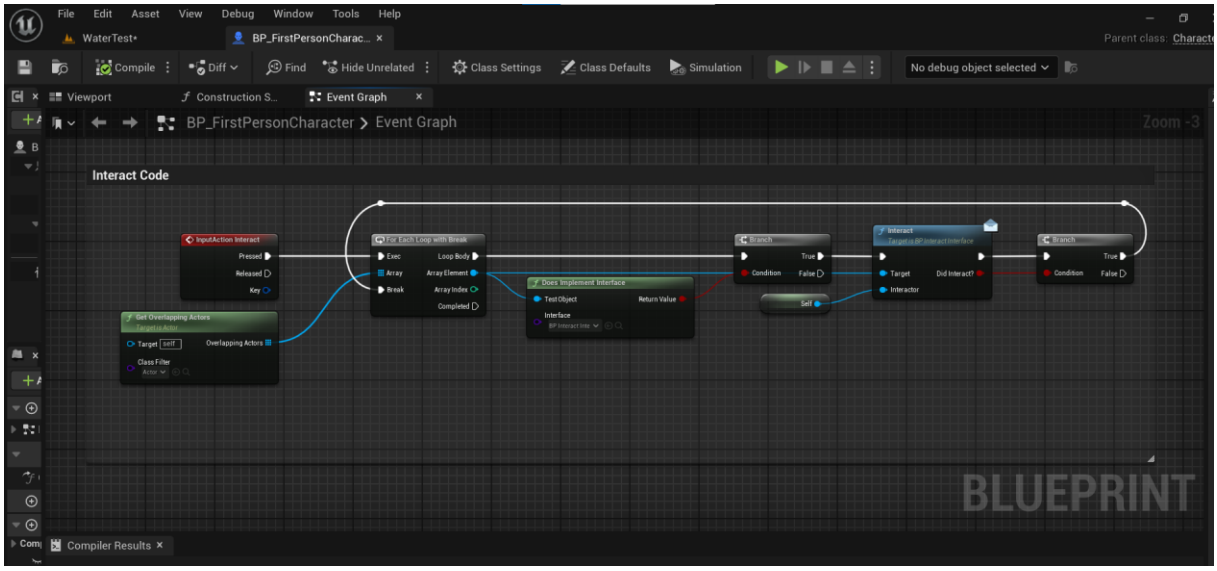
Slika 34. Dodavanje funkcije i varijabli u *Interface*-u



Slika 35. Akcija „*Interact*“

6.2.3. Blueprint „BP_FirstPersonCharacter“

S prethodno definiranim, određuje se logika unutar *„BP_FirstPersonCharacter“* za upravljanje modelom osobe (slika 36.). *InputAction „Interact“* je *node* događaja vezan za prethodno određenu akciju pritiska na tipku *„E“*. Pokretanjem simulacije i pritiskom na tipku se pokreće niz radnji vezanih na taj *node* koje će dalje pokretati interakciju s drugim objektima. *Get Overlapping Actors* je *node* funkcije koji vraća listu *Blueprint*-a uređaja s kojima model osobe vrši preklapanje, odnosno dolazi u doticaj. Pomoću *node*-a funkcije *DoesImplementInterface* se pregledava imaju li ti *Blueprint*-i uređaja u listi implementiran *„BP_InteractInterface“*. Ista funkcija vraća izlaznu vrijednost kao *true/false* što je ulazni parametar za *node Branch* koji određuje tijek daljnjeg odvijanja radnje ovisno o dobivenoj vrijednosti. Ako je vrijednost *true*, tada se poziva funkcija *interface*-a *„Interact“* u *Blueprint*-ima uređaja koji je posjeduju te ista određuje daljnji tijek radnje za te uređaje. Pomoću varijable *self* se definira da je *Blueprint* osobe taj koji vrši interakciju, a pomoću povratne vrijednosti „*DidInteract?“* se pregledava je li se interakcija izvršila. Ako je vrijednost *true*, *node For Each Loop with Break* pokreće novu sekvencu interakcija (samo jedan uređaj u zadano vrijeme da se ne pokrenu sve interakcije odjednom). Dakle, model osobe se približava uređaju nakon čega se pregledava ima li uređaj implementiranu funkciju *interface*-a. Ako da, poziva se ta funkcija u *Blueprint*-u uređaja koja definira daljnji tijek radnje te se ponavlja sekvenca u drugim slučajevima.



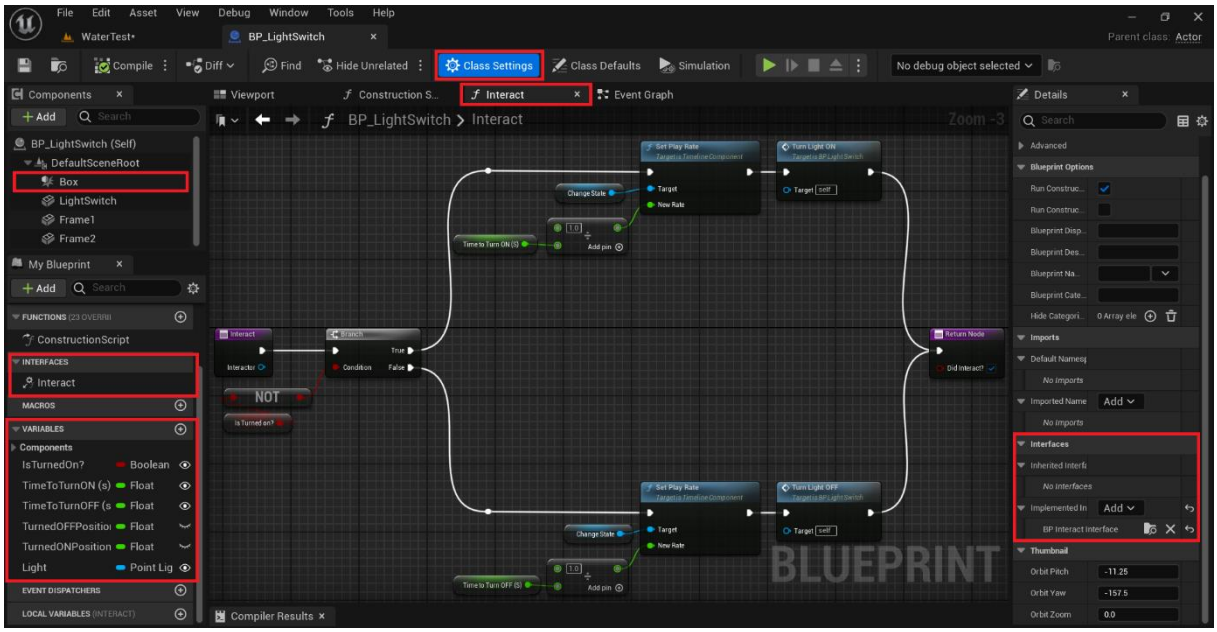
Slika 36. Logika unutar *"BP_FirstPersonCharacter"*

6.2.4. Blueprint-i *„BP_LightSwitch“* i *„BP_Fridge“*

Prvo, kako bi *Get Overlapping Actors* funkcija u *Blueprint*-u modela osobe mogla registrirati preklapanje, odnosno doticaj s drugim uređajima, unutar *Components* odjela u *Blueprint*-ima potrebnih uređaja se mora nalaziti element *BoxCollision.* To je nevidljiva komponenta koja definira prostor u kojemu može doći do interakcije između navedenog objekta i objekta koji se nađe u njemu. Time su modeli uređaja potpuno definirani te se logika može određivati. Izgled te komponente je prikazan na slici 32. oko uređaja u obliku kocke, a implementacija na slici 37. Idući korak je implementacija *„BP_InteractInterface“* u postavkama *Blueprint*-a u odjelu *Interfaces*. Time se pojavljuje funkcija *„Interact“* koja je spremna za uređivanje i definiranje slijeda radnji i animacija.

Funkcija *„Interact“ Blueprint*-a *„BP_LightSwitch“* je prikazana na slici 37. te je postavljena na sljedeći način. Ulaskom modela osobe u prostor blizu prekidača i pritiskom tipke *„E“* se pokreće funkcija *„Interact“*. Pomoću *Branch node*-a se očitava uvjet stanja prekidača pri kojem je svjetlo upaljeno ili ugašeno. To stanje je definirano *boolean* varijablom *„IsTurnedOn?“* čija je zadana vrijednost *false* i predstavlja ugašeno svjetlo, što je početno stanje. *Node*-om *NOT* se koristi suprotna vrijednost te varijable, odnosno *true* u početnom slučaju kako bi korisnik lakše mogao čitati: „Ako svjetlo NIJE upaljeno – šalji *true* i pokreni radnje za paljenje svjetla“. Te radnje su definirane unutar *EventGraph*-a pokretanjem događaja *„TurnLightON“*, koji se aktivira pozivanjem *node*-a istog imena u „*Interact“* funkciji povezanog na *true* ishod. Isto

tako, ako je svjetlo upaljeno, ponovnom interakcijom modela osobe i prekidača se odvijaju radnje za gašenje svjetla definirane unutar *EventGraph*-a pokretanjem događaja *„TurnLightOFF“*, koji se također aktivira pozivanjem *node*-a istog imena u *„Interact“* funkciji povezanog na *false* ishod. Između *node*-ova za aktivaciju radnji se nalazi *SetPlayRate node* koji služi za upravljanje brzinom animacije, u ovom slučaju rotacije prekidača pri promjenama stanja. Svoje podatke o brzini izvedbe šalje varijabli *ChangeState* koja predstavlja *node* postavljen u *EventGraph*-u za definiranje animacije. Brzina animacije ovisi o postavljenim vrijednostima *float* varijabli *„TimeToTurnON (s)“* i *„TimeToTurnOFF (s)“* te dijeljenju jedinice s istom. Na kraju, u oba slučaja se ostvaruje poveznica s *ReturnNode*-om koji vraća izlaznu varijablu koja govori da se interakcija dogodila.



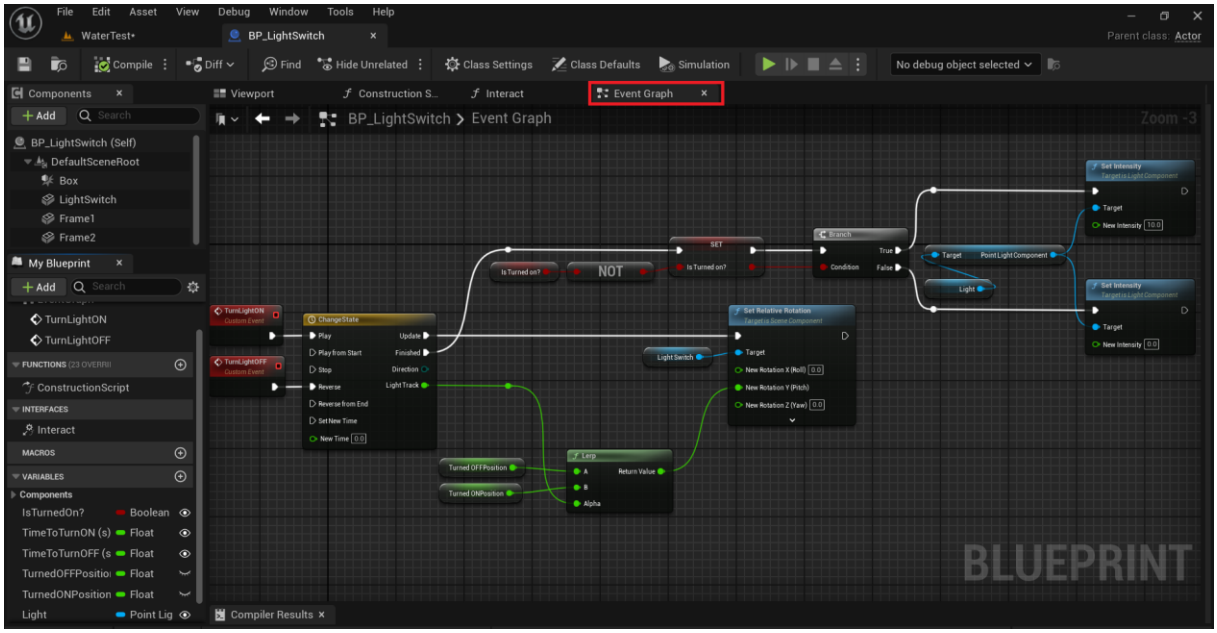
Slika 37. *"Interact"* funkcija u *"BP_LightSwitch"*

Budući da je funkcija koja pokreće odvijanje radnji i animacija određena, isti su razrađeni u nastavku. Pokretanjem *„TurnLightON“* događaja, cilj je zarotirati prekidač oko *y* osi iz stanja ugašenog u stanje upaljenog svjetla te nakon izvršene rotacije svjetlo i upaliti, a pokretanjem *„TurnLightOFF“* događaja dobiti suprotno. Prikaz se nalazi na slici 38.

Rotacija se izvršava pomoću *SetRelativeRotation node*-a koji se odnosi na *StaticMesh* objekt *„LightSwitch“*. Međutim, prije nego se rotacija može odviti, potrebno je kontrolirati vrijednost i smjer promjene stanja tijekom vremena. Tome služe *Timeline node „ChangeState“*, *float* varijable *„TurnedOFFPosition“ i „TurnedONPosition“* te *Lerp node*. Pomoću *„ChangeState“*

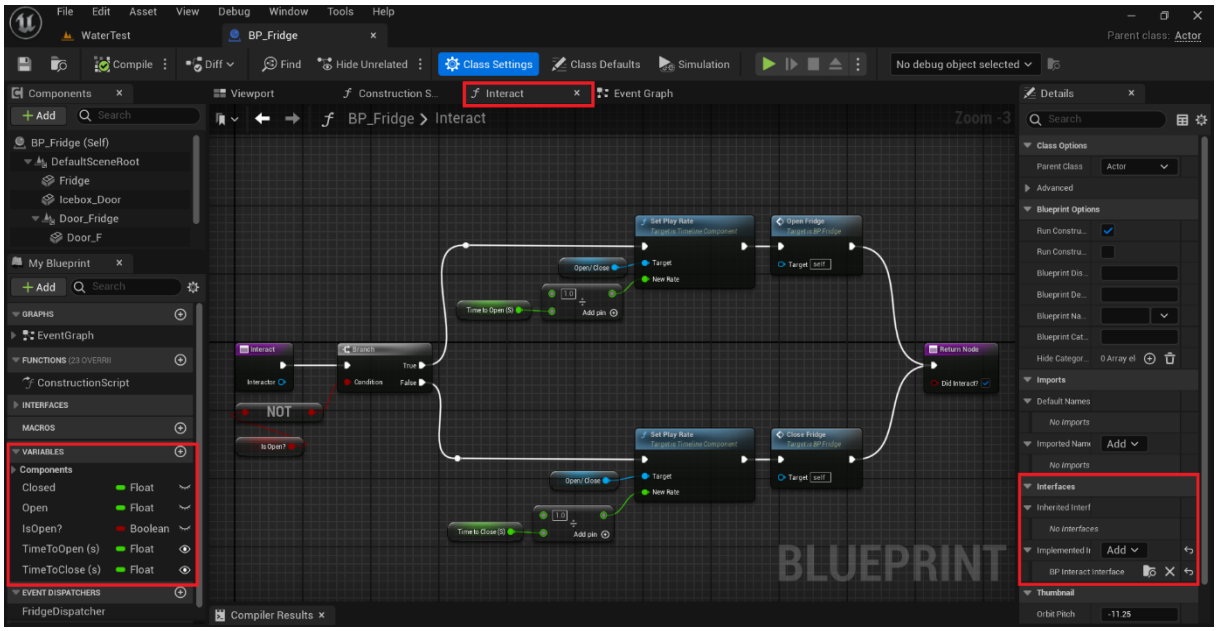
node-a je definiran linearan slijed promjene od 0 do 1 imena *„LightTrack“* te smjer iste. *Float* varijabla *„TurnedOFFPosition“* predstavlja vrijednost rotacije prekidača u stanju ugašenog svjetla čija je zadana vrijednost 0, a varijabla *„TurnedONPosition“* predstavlja vrijednost rotacije u stanju upaljenog svjetla čija je zadana vrijednost -10. *Lerp node* interpolira vrijednosti između *A* i *B* na temelju *Alpha-e* pri čemu 0 označava 100%-tnu vrijednost *A*, a 1 označava 100%-tnu vrijednost B.

Dakle, logika je sljedeća. Aktivacijom *„TurnLightON“* događaja se pomoću *„ChangeState“* *node*-a i preko *Lerp*-a definira linearna promjena stanja prekidača iz ugašenog stanja *A* (*„TurnedOFFPosition“)* u upaljeno stanje *B* (*„TurnedONPosition“*). Te vrijednosti se postupno ažuriraju i postaju ulazne vrijednosti rotacije oko *y* osi *SetRelativeRotation node*-a (od 0 do 10 stupnjeva u negativnom smjeru) budući da je *node* spojen na *Update* komponentu. S druge strane, aktivacijom *„TurnLightOFF“* događaja se pokreće suprotna animacija budući da je povezan na *Reverse* komponentu *„ChangeState“ node*-a. Pri završetku animacije (poveznica iz *Finished* komponente istog *node*-a), pomoću *Set node*-a se sprema nova vrijednost u varijablu *„IsTurnedOn?“* suprotna onoj očitanoj na početku funkcije *„Interact“*. Ako je prije bila *false*, sada je *true* i obratno te se tako označava promjena stanja. Na temelju toga se preko *Branch node*-a, a pomoću *SetIntensity node*-a postavlja jačina svjetla *„Light“* varijable koja je definirana kao *PointLightComponent* na stropu modela, odnosno pali i gasi svjetlo. Također, nova vrijednost varijable *„IsTurnedOn?“* se zadržava kako bi se ponovnom interakcijom i pokretanjem funkcije *„Interact“* radnja mogla odvijati u drugom smjeru.

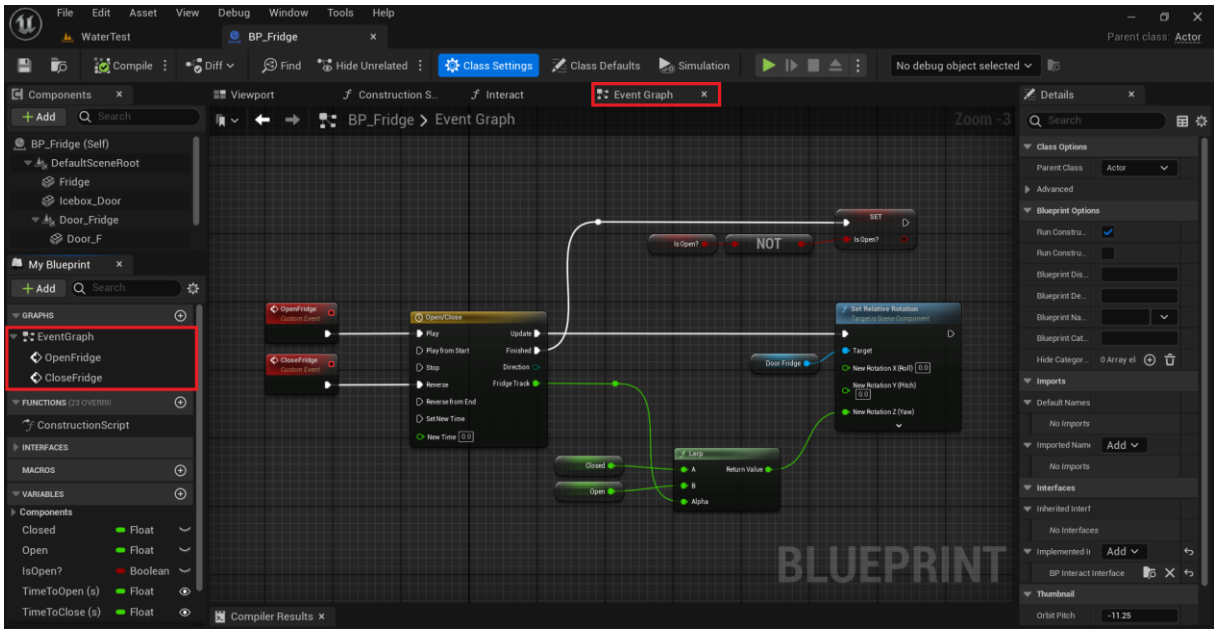


Slika 38. *EventGraph "BP_LightSwitch"*

Na vrlo sličan način su definirani funkcija *„Interact“* i *EventGraph Blueprint*-a hladnjaka *„BP_Fridge“*. Razlika je u imenima i vrijednosti varijabli, događaja i drugih *node*-ova i u tome što nakon postavljanja nove vrijednosti u *boolean*-u *„IsOpen?“* nema daljnje radnje. Prikaz *„Interact“* funkcije je na slici 39., a *Event Graph*-a na slici 40.



Slika 39. *"Interact"* funkcija u *"BP_Fridge"*



Slika 40. *EventGraph "BP_Fridge"*

Na kraju, na slikama 41. i 42. su prikazane animacije promjena stanja svjetla i hladnjaka. Time je virtualni model laboratorija i uređaja kreiran te je sljedeći korak uspostava komunikacije sa stvarnim okruženjem.

Slika 41. Paljenje svjetla

Slika 42. Otvaranje hladnjaka

7. OSTVARIVANJE DVOSMJERNE KOMUNIKACIJE

Nakon ostvarivanja interneta stvari laboratorija, implementacije njegovog digitalnog blizanca u Unreal Engine 5 programu, kreiranja objekata unutar njega i definiranja njihovog ponašanja, posljednji korak je ostvarivanje dvosmjerne komunikacije unutar stvarnog i virtualnog okruženja. Instaliranjem MQTT *plugin*-a, objašnjenog u poglavlju 4.4., omogućuje se pristup potrebnim *Blueprint node*-ama pomoću kojih se ostvaruje povezivanje na mrežu, stvaranje klijenta, primanje ili slanje poruke sa servera i na server te obrada iste poruke, a time i definiranje daljnje logike kojima se povezuju određene radnje unutar virtualnog modela sa stvarnim događajima i obratno.

Koncept ostvarivanja dvosmjerne komunikacije je sljedeći. U ostvarivanju interneta stvari laboratorija, svakom uređaju su definirane teme preko kojih se razmjenjuju poruke za specifične radnje ili informacije isključivo toga uređaja. Stoga je za svaku virtualnu inačicu uređaja definirana logika odvijanja animacije koja će nakon ostvarenog povezivanja Unreal Engine 5 programa s Home Assistant-om (pomoću kojeg je cijeli koncept interneta stvari definiran) pokrenuti tijek radnji slanja točno određene poruke na temu točno tog fizičkog uređaja. Tada se poruka s te teme čita, a rezultat je ostvarivanje radnje u stvarnom okruženju. Isti princip je i u obratnom smjeru. U stvarnom okruženju će se pokrenuti radnja zbog koje će senzor očitati promjenu stanja, a rezultat toga će biti slanje poruke na definiranu temu za taj uređaj. Zbog prethodno ostvarene komunikacije između Unreal Engine-a 5 i Home Assistanta-a te *Blueprint node*-a u kojemu je definirano čitanje poruke s te teme, pokreće se logika pomoću koje se odvijaju animacije virtualnih inačica koje odražavaju radnje u stvarnom okruženju.

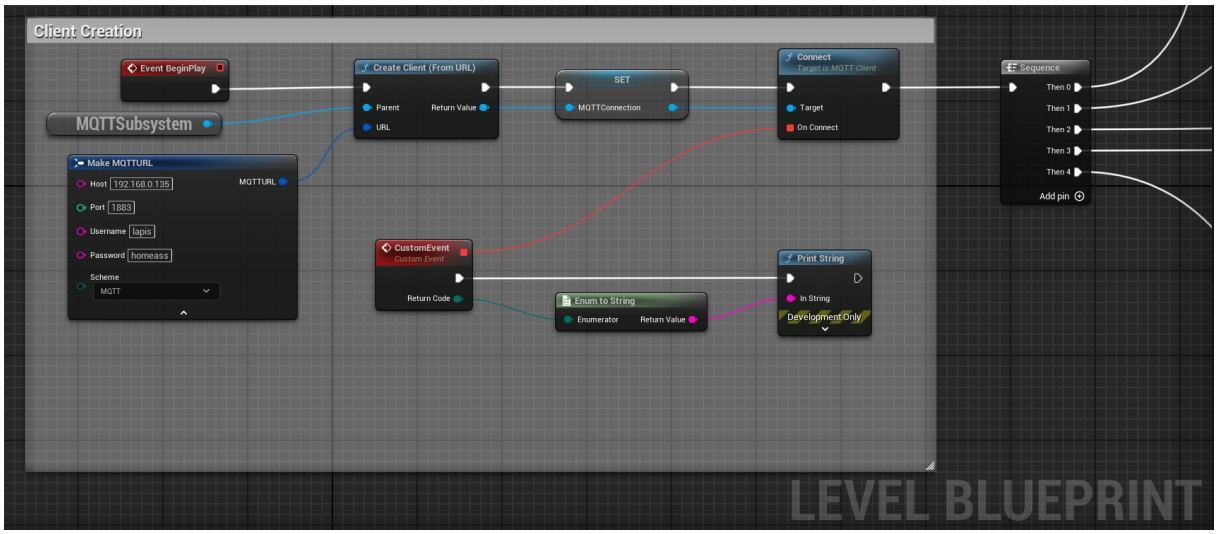
U poglavlju 4.3. se spominje *Level Blueprint*. Kako je navedeno, *Level Blueprint* je tip *Blueprint*-a koji služi za upravljanje cijele razine Unreal Engine 5 projekta, a time i svim *Blueprint* objektima u razini. Ako se neke cjeline u projektu žele potpuno odvojiti od drugih radi postavljanja drugačijih pravila ili početnih uvjeta (npr. jedan tijek logike koji potiče niz radnji treba utjecati na veliku količinu objekata, ali na pojedine ne), tada je pametno te objekte odvojiti u dvije različite razine koje su potpuno neovisne jedna o drugoj. Specifičan primjer je posjedovanje dva digitalna blizanca unutar istog projekta na dvije različite razine te time pravila za jednog blizanca nikako ne mogu utjecati na drugog. U ovom slučaju, projekt koji posjeduje digitalnog blizanca laboratorija i virtualne inačice uređaja posjeduje samo jednu razinu. Stoga, budući da su virtualni uređaji definirani u zasebnim *Blueprint* klasama, a oba uređaja trebaju biti upravljana istom logikom za stvaranje MQTT klijenta i ostvarivanje povezivanja, cijela

MQTT logika se definira u *Level Blueprint*-u jer isti to i omogućuje. S druge strane, pregled cijele logike na istom mjestu koja pokreće niz radnji u više klasa je poželjan i logičan izbor.

U nastavku će biti objašnjeno kako se projekt definira kao klijent i povezuje na mrežu, koje funkcionalnosti preuzima kao klijent u određenim slučajevima, koja je logika u pozadini te na koji način *Blueprint*-i virtualnih uređaja komuniciraju s *Level Blueprint*-om što u cjelini omogućuje ostvarivanje dvosmjerne komunikacije putem MQTT protokola unutar stvarnog i virtualnog okruženja.

7.1. Definiranje klijenta i povezivanje s brokerom

Na slici 43. je prikazano definiranje postavljanja programa kao klijenta i ostvarivanje povezivanja s brokerom u *Level Blueprint*-u.



Slika 43. Definiranje klijenta i povezivanje s brokerom

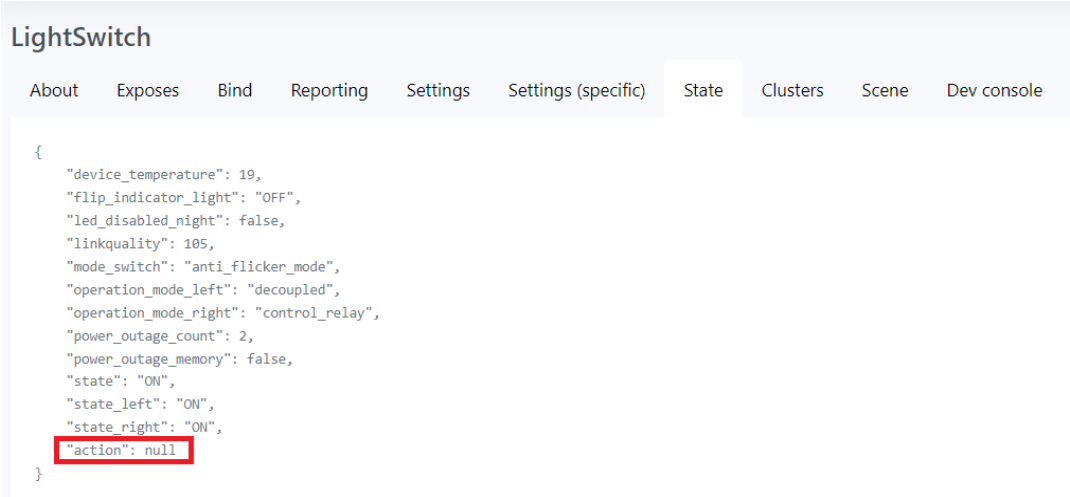
Blueprint node Event BeginPlay služi za inicijalizaciju radnji pri samom pokretanju simulacije. Budući da je potrebno ostvariti povezivanje s brokerom na početku radi trenutne sinkronizacije virtualnog i stvarnog modela, na njega se spaja *node Create Client (From URL)*. Ta *Blueprint* funkcija definira Unreal Engine 5 program kao klijenta MQTT mreže. Određivanje koje mreže će program biti klijent se ostvaruje pomoću *Make MQTTURL node*-a. Kako je navedeno u poglavlju 5.6., internet stvari laboratorija je određen Home Assistant platformom koja je instalirana na HP stolnom računalu, a broker je definiran unutar platforme kao naknadno instalirani *addon*. Zbog toga se u navedeni *node* upisuje IP adresa računala, te korisničko ime i

lozinka platforme. Na *Create Client* funkciju je također povezan i *MQTTSubsystem node* koji je najviši hijerarhijski objekt te potpuno definira i implementira MQTT protokol u Unreal Engine 5 programu. Nadalje, pomoću *Set node*-a se izlazni podaci stvorenog klijenta spremaju u posebnu varijablu „*MQTTConnection*“ kako bi se potrebne informacije mogle naknadno koristit u drugim *node*-ovima bez potrebe direktnog povezivanja. Funkcija *Connect* koristi sve potrebne informacije iz stvorenog klijenta kako bi se napokon ostvarilo povezivanje s brokerom. Također, sama funkcija pokreće *Custom Event*, događaj definiran od strane korisnika, koji će obaviti ispis u svrhu potvrde povezivanja. Na kraju, budući da će klijent u različitim situacijama za različite uređaje biti postavljen kao *Publisher* i/ili *Subscriber*, funkcija *Connect* se povezuje na *Sequence* koji omogućuje grananje i višestruko definiranje klijenta ovisno o trenutnoj razmjeni informacija i pokrenutim događajima.

Na ovaj način je ostvareno postavljanje programa kao klijenta MQTT mreže i povezivanje s brokerom te je sljedeći korak postavljenje logike za sinkronizaciju i komunikaciju pametnog sustava rasvjete i hladnjaka.

7.2. Komunikacija rasvjete

Prije svega, potrebno je naglasiti sljedeće. Prethodno definirana poruka koja se razmjenjuje u MQTT mreži preko postavljene teme sustava pametnog osvjetljenja, ali i za druge uređaje, sadrži nekolicinu podataka. Jedan tip podatka unutar te poruke je *Payload* JSON. Tu se nalaze sve informacije stanja uređaja koje su prikazane na slici 44.



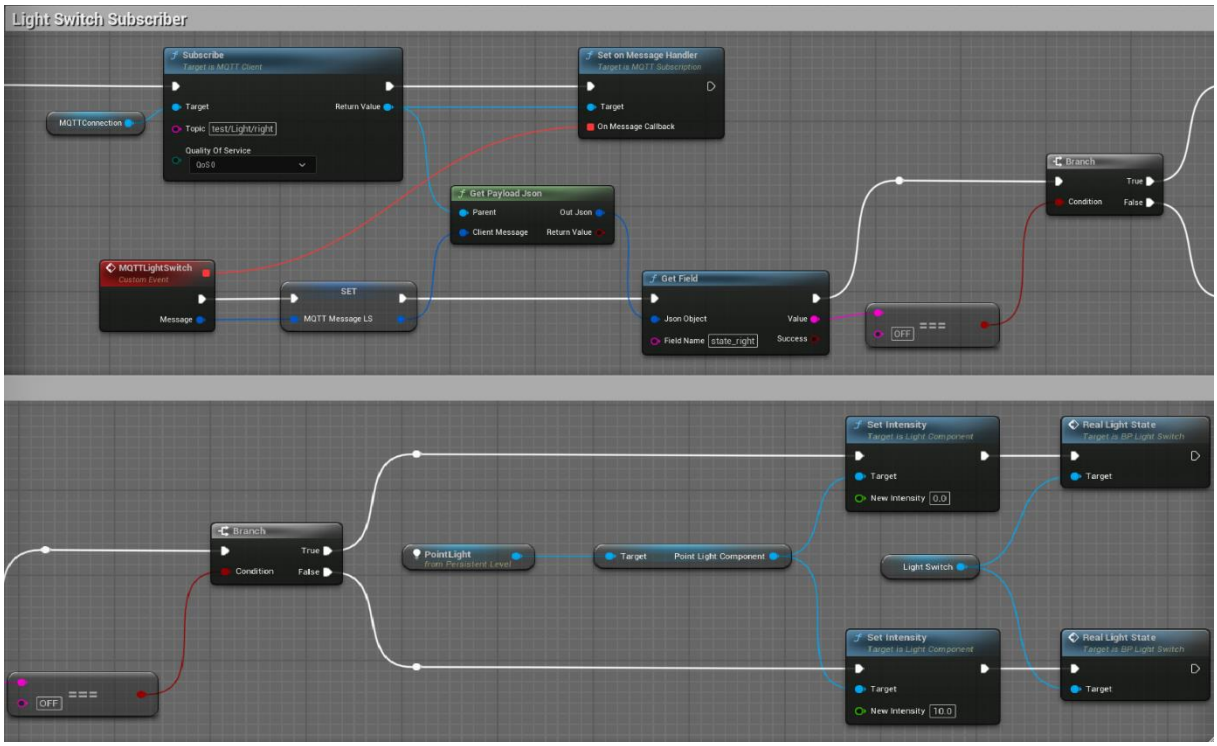
Slika 44. *Payload* JSON *LightSwitch* uređaja

Međutim, budući da je MQTT *plugin* za Unreal Engine 5 program još uvijek u razvoju i nije optimalno podešen, eksperimentalno je dokazano da pretplata na potrebnu temu koja razmjenjuje *null* vrijednosti u JSON-u (slika 44.) uzrokuje pad programa. Zbog toga je stvorena automatizacija prikazana na slici 25. u poglavlju 5.6. koja paljenjem i gašenjem istog uređaja razmjenjuje samo vrijednosti polja „*state_right*“ na novo definiranu temu, budući da je paljenje i gašenje definirano na desnu tipku fizičkog uređaja.

Koncept ostvarivanja dvosmjerne komunikacije rasvjete je sljedeći. Isti je podijeljen u 3 logičke cjeline: interaktivna i logika animacije virtualnog uređaja, logika određivanja programa kao klijenta objavljivača poruke na temu rasvjete brokera i logika određivanja programa kao klijenta pretplatnika na temu brokera za istu. Cilj je postići isto stanje virtualnog i stvarnog uređaja neovisno u kojem smjeru se radnja provodi. Pretpostavlja se slučaj u kojemu se prva akcija pokreće u virtualnom modelu u stanju ugašenog svjetla u oba okruženja. To stanje je određeno *boolean* varijablom *„IsTurnedON?“* u konceptu čija je početna vrijednost zadana kao *false*. Zbog vrijednosti te varijable se interakcijom pokreće logika za paljenje svjetla u virtualnom modelu. Paljenjem svjetla u istome se poziva pokretanje radnji u logičkoj cjelini za određivanja programa kao klijenta objavljivača. Nakon objavljene poruke na temu rasvjete brokera za promjenu stanja iz ugašenog u upaljeno, stanje se mijenja i u stvarnom modelu, odnosno svjetlo u laboratoriju se pali. Time se automatski aktivira logička cjelina određivanja programa kao klijenta pretplatnika pomoću koje se čita novo stanje s teme brokera i potvrđuje isto u modelu. Pri završetku te cjeline se mijenja vrijednost varijable *„IsTurnedON?“* iz *false* u *true* što označava promjenu stanja. Ponovnom interakcijom u virtualnom modelu i čitanjem nove vrijednosti varijable, pokreće se logika za gašenje svjetla što završetkom poziva objavljivanje ugašenog stanja koje će potaknuti gašenje svjetla u stvarnosti. Tako se ponovno automatski aktivira logička cjelina za pretplatu koja čita novu poruku (ugašeno) na temi i potvrđuje isto stanje u modelu. Na kraju te cjeline, ponovno se mijenja vrijednost varijable *„IsTurnedON?“*, no ovoga puta iz *true* u *false*. S druge strane, ako se radnja odvija u smjeru stvarnog prema virtualnom okruženju, paljenjem ili gašenjem svjetla u laboratoriju se pokreće samo logika za definiranje programa kao klijenta pretplatnika koji s teme brokera čita poruku i odražava trenutno stanje rasvjete laboratorija u virtualnom modelu. Također, vrijednost varijable *„IsTurnedON?“* se sukladno mijenja na kraju te cjeline u *true* ako se svjetlo pali ili u *false* ako se gasi. Takvom logikom se osim dvosmjerne komunikacije osigurava i sinkronizacija oba okruženja. To znači da neovisno u kojem okruženju promjena stanja započinje, sljedeća promjena može biti u bilo kojem od dva okruženja koja će postaviti jednaka stanja u istima.

7.2.1. Subscriber rasvjete

Na slici 45. je prikazana logika pretplate Unreal Engine 5 programa na novu temu rasvjete koja će ovisno o vrijednosti primljenog podatka utjecati na virtualni model iste.



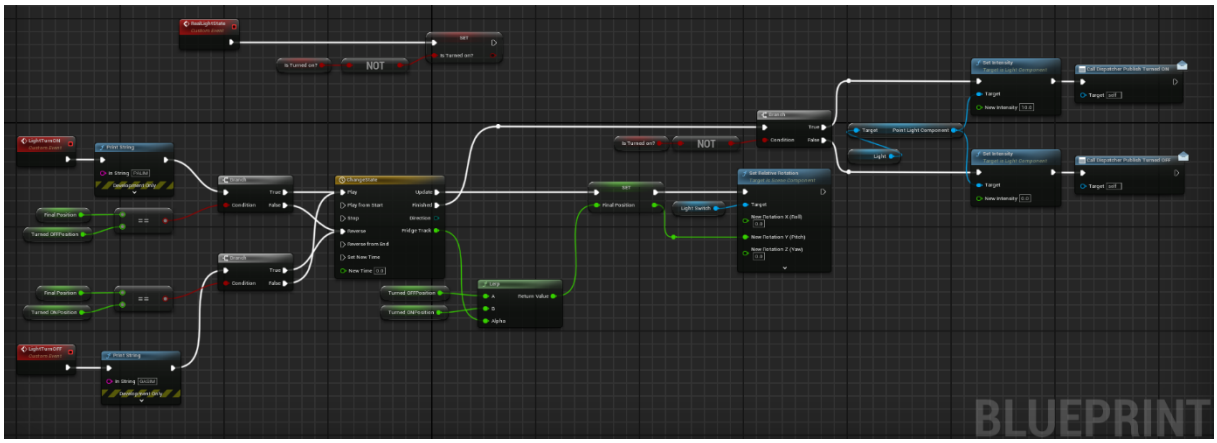
Slika 45. *Light Switch Subscriber*

Na jedan od *Sequence pin*-ova je povezana funkcija *Subscribe*. Ta funkcija definira program kao klijenta pretplatnika teme MQTT brokera. U područje *Target* se povezuje izlazna vrijednost funkcije *Create Client* kao varijabla kako bi se znalo da je program klijent koji se pretplaćuje, a u *Topic* se upisuje tema za sustav rasvjete definirane na slici 25. Pomoću funkcije *Set on Message Handler* se obrađuje poruka i poziva događaj imena „*MQTTLightSwitch"* koji je definiran od strane korisnika. Pokretanjem događaja se pomoću *Blueprint node*-a *Set* cijela poruka sprema u zasebnu varijablu, a funkcijom *Get Payload Json* se iz cijele poruke izdvaja samo dio u kojemu se nalaze informacije stanja uređaja. Funkcija *Get Field* izdvaja potrebno polje „*state_right*" u kojemu se nalazi vrijednost koja pokazuje je li rasvjeta upaljena ili ugašena. Nadalje, pomoću funkcije *Equal Exactly (String)* se definira vrijednost s kojom se želi usporediti dobivena vrijednost „*state_right*" polja i time se *Branch Blueprint*-om radnja usmjerava dalje. Ako su obje vrijednosti „*OFF"*, izlazna vrijednost je *true* te se funkcijom *Set Intensity* postavlja jačina *Point Light Component*-e na nulu što gasi svjetlo u virtualnom

modelu. S druge strane, ako se paljenjem svjetla u laboratoriju primi vrijednost „*ON*“, izlazna vrijednost usporedne funkcije će biti *false* te će se jačina *Point Light Component*-e postaviti na vrijednost deset. Tako je svjetlo virtualnog modela upaljeno. Na kraju, u oba slučaja se pokreće funkcija *„Real Light State“* koja poziva istoimeni događaj definiran u *Blueprint*-u objekta rasvjete *„BP_LightSwitch“*. Varijabla *„LightSwitch“*, koja je ulazni podatak u navedenu funkciju, je definirana kao sam *Blueprint* objekta rasvjete kako bi se taj *Blueprint* povezao s *Level Blueprint*-om te da bi tako program znao gdje se potreban događaj nalazi. Tim događajem se pokreće mijenjanje vrijednosti *boolean* varijable *„IsTurnedON?“* koja pokazuje promjenu stanja stvarnog i virtualnog uređaja iz upaljenog u ugašeno i obratno kako bi se ponovnom interakcijom pokretao suprotan tijek radnji.

7.2.2. *Publisher* rasvjete

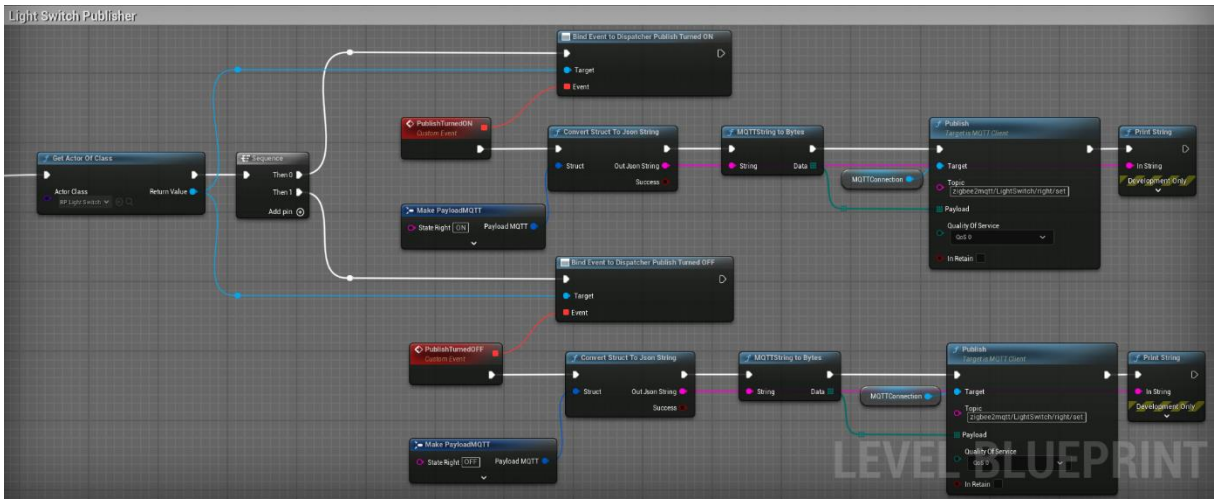
Prije nego se objasni objavljivački dio istog uređaja, potrebno je prikazati neke izmjene u logici animacije virtualnog prekidača. Na slici 46. je prikazana promjena *EventGraph*-a *Blueprint*-a *„BP_LightSwitch“*.



Slika 46. Promjena *EventGraph*-a u *"BP_LightSwitch"*

Prva promjena je ta da se pomoću nove *float* varijable *„FinalPosition“* pamti konačna rotacija prekidača nakon odvijene animacije kako bi se idućom interakcijom s prekidačem u virtualnom modelu mogla usporediti sa stanjem rotacije prethodne interakcije. Time se osigurava da neovisno u kojem položaju se prekidač nalazi, iduća animacija može i paliti ili gasiti svjetlo. Tako se simulira prekidač kojemu oba položaja služe i za paljenje i za gašenje, ovisno o tome koja je radnja sljedeća. Druga promjena je ta da se nakon odvijanja animacije koristi samo trenutna promjena vrijednosti varijable „*IsTurnedON?“* koja određuje u kojem smjeru se logika

kreće, umjesto spremanja te vrijednosti u istu. Razlog tome je što se na temelju očitane trenutne vrijednosti te varijable pokreće objavljivanje upaljenog ili ugašenog stanja virtualnog modela, a nakon toga i automatska pretplata i očitavanje promjene u stvarnom modelu koji na kraju svoje radnje završava tijek cjelokupnog događaja i sprema novu vrijednost u varijablu, kako je objašnjeno u konceptu. Treća razlika je dodavanje *Event Dispatcher*-a na kraju svakog postavljanja jačine svjetla u virtualnom modelu. *Event Dispatcher* je značajka pomoću koje je moguće definirati nastavak radnje iz jednog *Blueprint*-a u drugome. Stoga, paljenje svjetla interakcijom korisnika s virtualnom uređajem definirano u *Blueprint*-u *„BP_LightSwitch“* pokreće *Event Dispatcher „DispatcherPublishTurnedON“* koji će pokrenuti nastavak radnje za objavljivanje upaljenog stanja brokeru određene u *Level Blueprint*-u jer se tamo nalazi MQTT logika. S druge strane, gašenje svjetla u virtualnom modelu će se pokrenuti drugi *Event Dispatcher „DispatcherPublishTurnedOFF“* koji također pokreće nastavak radnje u *Level Blueprint*-u za objavljivanje ugašenog stanja brokeru. S time, objašnjenje objavljivača stanja svjetla virtualnog modela stvarnom uređaju je u nastavku, a logika je prikazana na slici 47.



Slika 47. *Light Switch Publisher*

Na jedan od *Sequence pin*-ova je povezana funkcija *Get Actor of Class*. Tom funkcijom se pronalazi *Blueprint* klasa uređaja, u ovom slučaju *„BP_LightSwitch“*, i implementira u *Level Blueprint* kako bi se mogle koristiti funkcije vezane za isti. Pomoću novog *Sequence Blueprint node*-a se odvaja tijek logike za objavljivanje različitih stanja, ovisno o tome koji je *Event Dispatcher* pozvan. Ako je interakcijom svjetlo u virtualnom modelu upaljeno, poziva se *node Bind Event to „DispatcherPublishTurnedON“*. Taj *node* služi kako bi se označio nastavak radnje definirane u *Blueprint*-u *„BP_LightSwitch“*. Nadalje, njegovom aktivacijom se pokreće

događaj definiran od strane korisnika imena *„PublishTurnedON“.* Potrebno je naglasiti da se u logici nalazi *Make „PayloadMQTT“ Blueprint node* koji koristi strukturni tip podatka istoga imena. Ta struktura služi za definiranje sličnog *Payload*-a u Unreal Engine-u 5 poput onoga koji je definiran u Home Assistant-u za razmjenu informacija uređaja pomoću određenih tema. Stoga, u navedenoj strukturi je definirano polje „*state_right*“ neodređene *string* vrijednosti. Kako je prikazano na slici 47., pomoću *Blueprint node*-a *Make „PayloadMQTT“* se definira vrijednost „*ON“* unutar polja „*state_right*“. Ta kompletna struktura se pretvara u JSON *string* pomoću funkcije *Convert Struct to JSON String,* a zatim se navedeni *string* pretvara u niz *byte*-ova koji će se slati brokeru na temu uređaja. Na kraju se nalazi funkcija *Publish* koja definira program kao klijenta objavljivača. Ulazni podaci *Publish* funkcije su tema rasvjete *„zigbee2mqtt/LightSwitch/right/set“* (definirana za razmjenu podataka o paljenju i gašenju svjetla), prethodno određen *Payload* izražen u nizu *byte-ov*a te kvaliteta slanja mreže. Ukratko, pokretanjem događaja *„PublishTurnedON“* se čita postavljena vrijednost *„ON“* u polju „*state_right*“ spomenute strukture i objavljuje na zadanu temu pomoću *Publish* funkcije. Budući da su polje i vrijednost polja u strukturi točno određeni kao i *Payload* JSON unutar Home Assistant-a za temu tog uređaja, broker lako prepoznaje koju vrijednost kojeg polja treba mijenjati te se svjetlo pali u stvarnome laboratoriju.

Ista logika se poziva ako se ugasi svjetlo interakcijom u virtualnom modelu. Aktivacijom funkcije *Bind Event to „DispatcherPublishTurnedOFF“* se pokreće događaj definiran od strane korisnika imena *„PublishTurnedOFF“*. U ovom slučaju, čita se definirana vrijednost „*OFF*“ polja „*state_right*“ te šalje pomoću funkcije *Publish* na zadanu temu čime se gasi svjetlo u stvarnome laboratoriju.

Međutim, kako bi objavljivanje podataka na temu iz Unreal Engine-a 5 bilo moguće, potrebno je naglasiti sljedeće. Na slici 47. je prikazano da ulazni podatak *Payload* u *Publish* funkciji mora biti izražen kao niz *byte*-ova, a potrebno polje i njegova vrijednost objavljivanja su definirani kao *Blueprint* struktura izraza {„*state_right*“:“vrijednost u obliku *string*-a“.} koja se kasnije pretvara u JSON *string.* Nakon te operacije je potrebno isti JSON *string* pretvoriti u niz *byte*-ova. No, u cijelom *Blueprint* sustavu programiranja (ni u MQTT *plugin*-u) ne postoji funkcija koja obavlja tu operaciju. Stoga, potrebno je samostalno napraviti C++ klasu kako je objašnjeno u poglavlju 4.2.1. Slika 48. prikazuje *.cpp* datoteku koja sadrži kod koji omogućuje pretvorbu, a slika 49. prikazuje *.h* datoteku pomoću koje se implementira funkcija *„MQTTStringtoBytes“* u *Blueprint* knjižnicu kako bi se mogla koristiti. Time je *Publish* rasvjete u potpunosti definiran.

```cpp
// Fill out your copyright notice in the Description page of Project Settings.


#include "MQTTaddons.h"

void UMQTTaddons::MQTTStringToBytes(const FString& String, TArray<uint8>& data){
    uint32 size = String.Len();
    data.AddUninitialized(size);
    uint8* dataPtr = data.GetData();
    StringToBytes(String, dataPtr, size);
    for (uint32 i = 0; i < size; i++) {
        *(dataPtr + i) = *(dataPtr + i) + 1;
    }
}
```

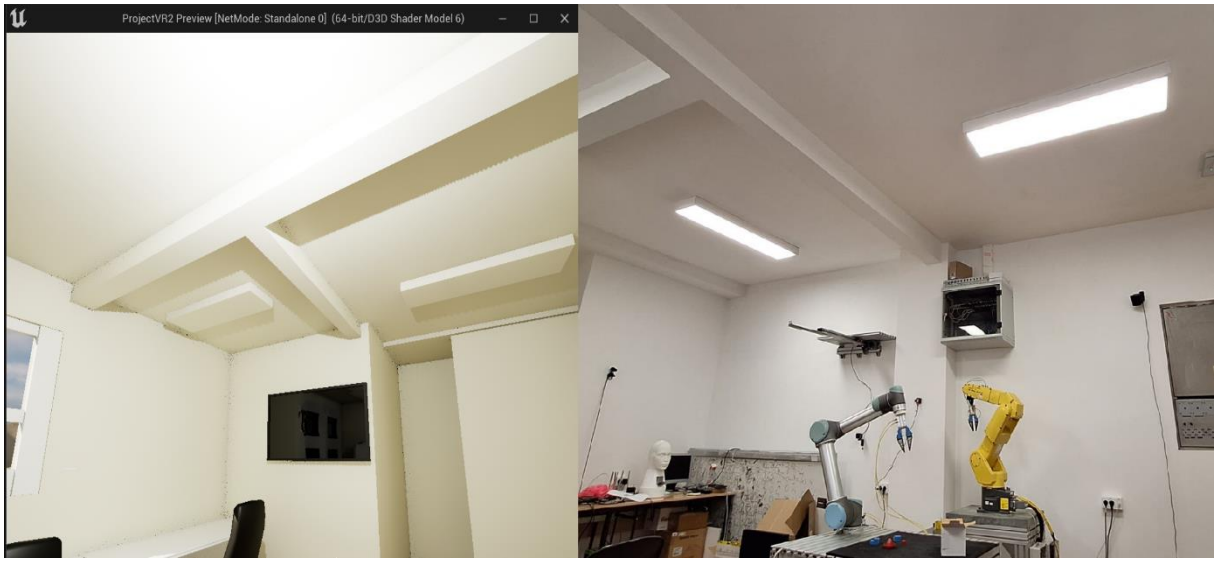
Slika 48. *source* datoteka

```cpp
// Fill out your copyright notice in the Description page of Project Settings.

#pragma once

#include "CoreMinimal.h"
#include "Kismet/BlueprintFunctionLibrary.h"
#include "MQTTaddons.generated.h"

/**
 *
 */
UCLASS()
class PROJECTVR2_API UMQTTaddons : public UBlueprintFunctionLibrary
{
    GENERATED_BODY()

public:
    UFUNCTION(BlueprintCallable)
        static void MQTTStringToBytes(const FString& String, TArray<uint8>& data);
};

```

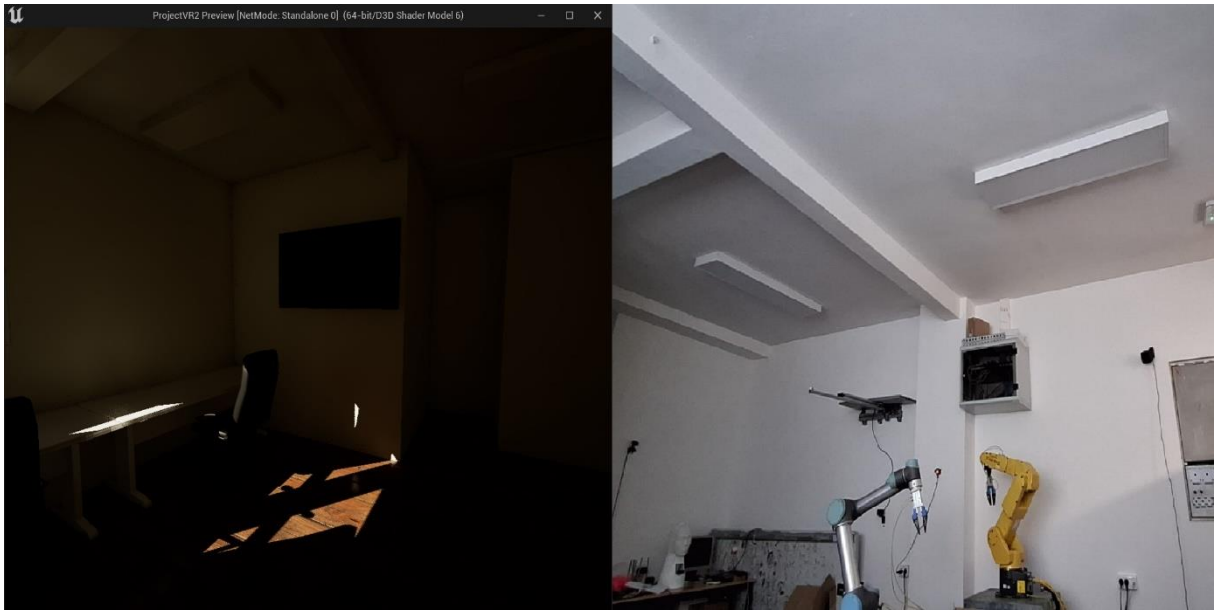
Slika 49. *header* datoteka

7.2.3. Rezultati

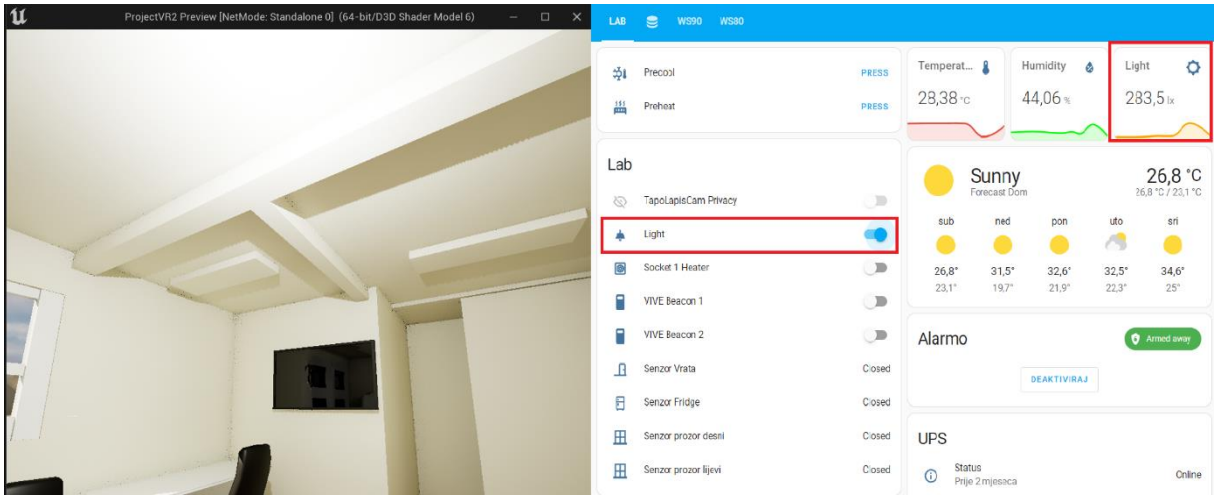
Prethodnim je ostvarena dvosmjerna komunikacija rasvjete unutar stvarnog i virtualnog okruženja. No osim komunikacije, pravilnom logikom je ostvarena i sinkronizacija. Odnosno, paljenje i gašenje svjetla u laboratoriju kontinuirano pali i gasi svjetlo u virtualnom modelu i obratno. Međutim, također je moguće upaliti svjetlo u virtualnom modelu (što će upaliti svjetlo i u laboratoriju) te nakon toga ugasiti svjetlo u laboratoriju (što će ugasiti svjetlo i u virtualnom modelu) i obratno.



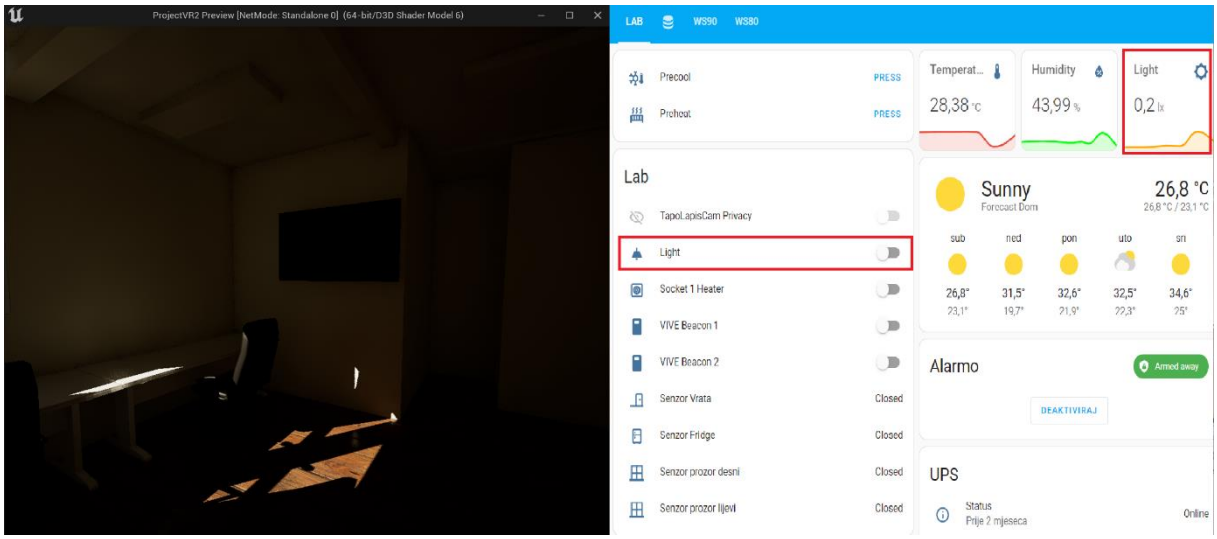
Slika 50. Upaljeno svjetlo u oba okruženja



Slika 51. Ugašeno svjetlo u oba okruženja



Slika 52. Prikaz upaljenog stanja u Home Assistant-u



Slika 53. Prikaz ugašenog stanja u Home Assistant-u

7.3. Komunikacija hladnjaka

Temeljna razlika hladnjaka i pametnog sustava rasvjete je ta što hladnjak posjeduje samo kontaktni senzor. Zbog toga, otvaranje i zatvaranje stvarnog hladnjaka će moći poticati odražavanje tih radnji u virtualnom okruženju, no istim radnjama virtualnog uređaja se ne može utjecati na otvaranje i zatvaranje stvarnog hladnjaka. Međutim, kako bi dvosmjerna komunikacija ipak postojala, zamišljeno je da se promjenom stanja virtualnog hladnjaka pojavljuje tekstualna poruka na preglednoj ploči Home Assistant-a. Tu poruku osoba u

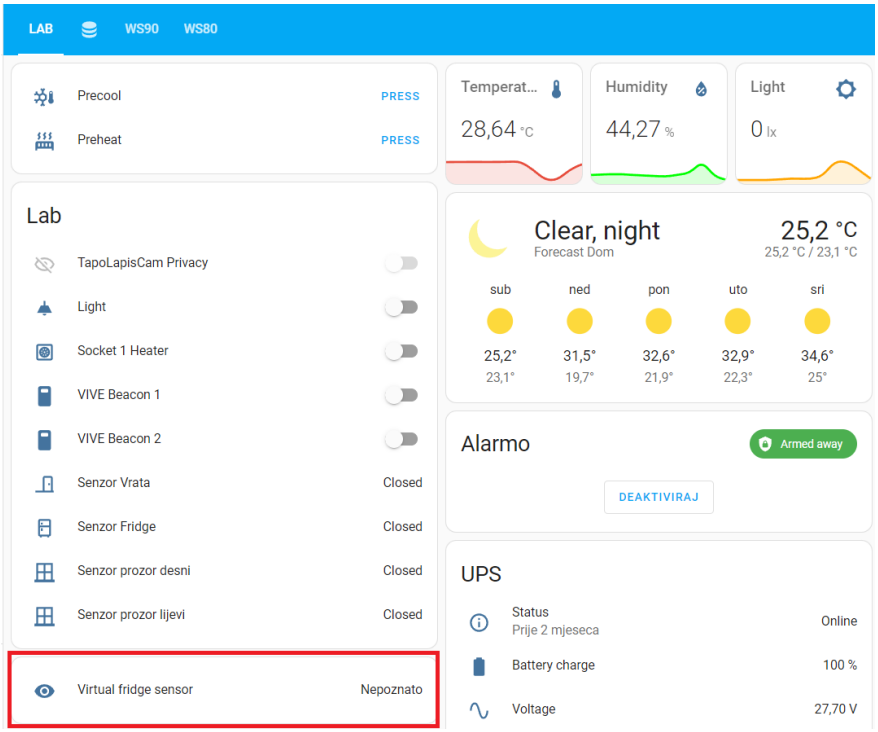
laboratoriju može pročitati i djelovati kao posrednik zatvaranjem ili otvaranjem hladnjaka, ovisno o uputi poruke. Kako bi navedeno bilo moguće, u konfiguraciji Home Assistant-a (*configuration.yaml* datoteka) je potrebno definirati postojanje novog senzora za razmjenu MQTT poruka. Naravno, taj senzor ne postoji nego je reprezentacija virtualnog uređaja. No, isti se definira kao fizički uređaj kako bi mu se mogla dodijeliti tema za objavu tekstualne poruke i kako bi se na preglednu ploču Home Assistant-a mogla dodati njegova kartica na kojoj će poruka biti uočljiva i lako se čitati. Na slici 54. je prikazana izmjena konfiguracijske datoteke dodavanjem senzora imena *„Virtual Fridge Sensor“* i određivanjem njegove teme *„test/Fridge/notification“* pod mqtt/sensor odjelom.

/homeassistant/configuration.yaml

```
# Loads default set of integrations. Do not remove.
default_config:

http:
  trusted_proxies: 192.168.0.1
  use_x_forwarded_for: true

logger:
  default: warning

# Load frontend themes from the themes folder
frontend:
  themes: !include_dir_merge_named themes

homeassistant:
    allowlist_external_dirs:
      - "/tmp"

mqtt:
  sensor:
    - name: "WS80 online"
      state_topic: "homeassistant/sensor/Workstation80/availability"
    - name: "WS90 online"
      state_topic: "homeassistant/sensor/Workstation90/availability"
    - name: "Virtual fridge sensor"
      state_topic: "test/Fridge/notification"

wake_on_lan:

automation: !include automations.yaml
script: !include scripts.yaml
scene: !include scenes.yaml
shell_command: !include shell_commands.yaml

smartir:

climate:
  - platform: smartir
    name: Office AC
    unique_id: office_ac
    device_code: 1502
    controller_data: esp_wroom_send_raw_command
    temperature_sensor: sensor.esp_wroom_lab_temperature
    humidity_sensor: sensor.esp_wroom_lab_humidity
```

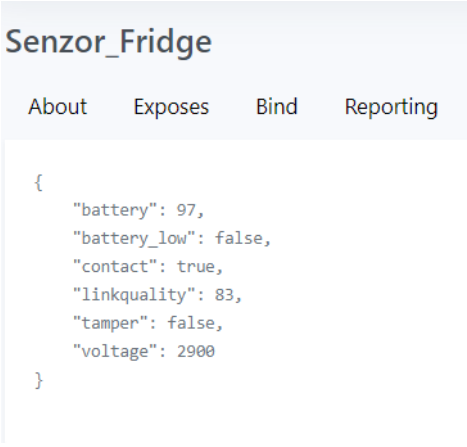
Slika 54. Dodavanje "virtualnog" senzora

Nakon toga, uređivanjem pregledne ploče Home Assistant-a i dodavanjem kartice senzora, na istoj se pojavljuje odjeljak gdje će se prikazivati objavljena poruka. Prikaz dodane kartice na preglednoj ploči se nalazi na slici 55.



Slika 55. Kartica za objavu poruke virtualnog hladnjaka na preglednoj ploči

Također, prije objašnjenja koncepta dvosmjerne komunikacije hladnjaka, potrebno je utvrditi koje se sve informacije stanja uređaja razmjenjuju u definiranom *Payload* JSON-u preko teme za kontaktni senzor stvarnog hladnjaka u Home Assistant-u.

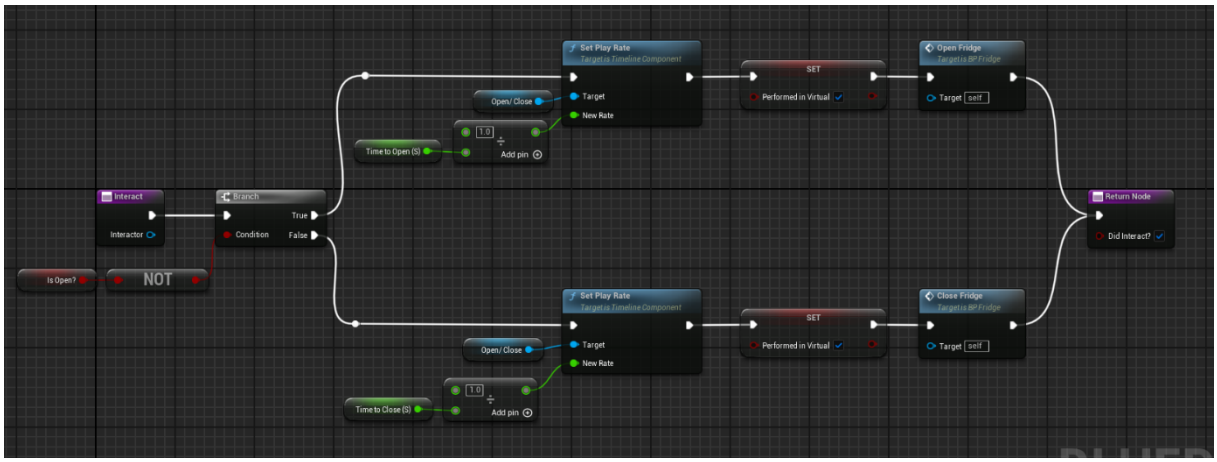


Slika 56. *Payload* JSON kontaktnog senzora hladnjaka

Koncept ostvarivanja dvosmjerne komunikacije hladnjaka je sličan onome za rasvjetu, no postoji nekoliko razlika. Isti je također podijeljen u 3 logičke cjeline: interaktivna i logika animacije virtualnog uređaja, logika određivanja programa kao klijenta objavljivača poruke (obavijesti) na temu novopostavljenog virtualnog senzora hladnjaka i logika određivanja programa kao klijenta pretplatnika na temu brokera stvarnog hladnjaka. Cilj je postići sinkronizaciju radnji stvarnog i virtualnog hladnjaka kada se radnja potiče u stvarnome okruženju, a ispisati obavijest u Home Assistant-u kada se radnja potiče u virtualnom okruženju kako bi osoba u laboratoriju mogla obaviti istu u stvarnosti. Zbog toga se osim *boolean* varijable *„IsOpen?“* koja kontrolira stanje otvorenosti virtualnog hladnjaka dodaju i *boolean* varijable *„PerformedInVirtual“* i *„IsRealFridgeOpen?“*. Varijablom *„PerformedInVirtual“* se prati pokreće li se radnja u virtualnom ili stvarnom okruženju. Ista poprima vrijednost *true* ako se radnja pokreće u virtualnom, a vrijednost *false* u stvarnom okruženju. Nadalje, varijablom *„IsRealFridgeOpen?“* se kontrolira je li stvarni hladnjak otvoren (*true*) ili zatvoren (*false*). Pretpostavlja se slučaj u kojemu se prva akcija pokreće u virtualnom modelu u stanju zatvorenog hladnjaka u oba okruženja. To stanje je određeno vrijednostima *false* svih *boolean* varijabli. Prvom interakcijom s virtualnim hladnjakom se pokreće logika otvaranja istoga te se trenutno sprema nova vrijednost *true* u varijablu *„PerformedInVirtual“*. Nakon toga se odvija animacija otvaranja hladnjaka, a zbog nove vrijednosti spomenute varijable (koja je uvjet za određivanje idućeg smjera logike) se također mijenja i sprema nova vrijednost *true* u varijablu *„IsOpen?“* te poziva pokretanje radnji u logičkoj cjelini za određivanje programa kao klijenta objavljivača. Time se na temu virtualnog senzora definirane prema slici 54. objavljuje uputa „OTVORI HLADNJAK“ i pojavljuje na preglednoj ploči Home Assistant-a. Ako je virtualni model hladnjaka prethodno bio otvoren te je vrijednost varijable *„IsOpen?“* bila *true*, ponovnom interakcijom s virtualnim hladnjakom se opet postavlja vrijednost *true* u varijablu *„PerformedInVirtual“*. Odvija se animacija zatvaranja hladnjaka nakon čega logika ponovno ide istim smjerom, no ovaj put se sprema vrijednost *false* u varijablu *„IsOpen?“* te objavljuje poruka „ZATVORI HLADNJAK“. U oba slučaja se nakon toga očekuje reakcija osobe u laboratoriju koja otvara ili zatvara stvarni hladnjak. S druge strane, ako se radnja odvija u smjeru stvarnoga prema virtualnom okruženju, otvaranjem ili zatvaranjem hladnjaka u laboratoriju se pokreće logika definiranja programa kao klijenta pretplatnika na određenu temu istoga. Nakon toga se čita poruka s teme, a zatim varijabla *„PerformedInVirtual“* poprima vrijednost *false*. Time se stanje hladnjaka u laboratoriju animacijom odražava na njegov virtualni model, no radnja još nije završena. Zbog *false* vrijednosti „*PerformedInVirtual“* varijable se logika usmjerava suprotno od slučaja kada se radnja potiče u virtualnom okruženju.

Taj smjer logike mijenja i sprema nove, identične vrijednosti u varijable *„IsRealFridgeOpen?“* i *„IsOpen?“* (*true* ako se hladnjak otvorio i *false* ako se zatvorio) kako bi se automatski mogla objaviti poruka na temu virtualnog senzora hladnjaka „OTVOREN“ ili „ZATVOREN“. Tako se na još jedan način potvrđuje je li osoba u laboratoriju obavila potrebnu radnju nakon odvijanja interakcije s hladnjakom u virtualnom okruženju ili jednostavno uz vizualni prikaz u modelu pokazuje trenutno stanje otvorenosti stvarnog hladnjaka. Također, promjenom vrijednosti varijabli *„IsRealFridgeOpen?“* i *„IsOpen?“* se omogućuje suprotna radnja sljedećom interakcijom u virtualnom okruženju. Na kraju, neovisno o interakciji s hladnjakom u stvarnome ili virtualnom okruženju, dodana su dva objavljivača od kojih jedan pri početku simulacije objavljuje poruku „SIMULACIJA POKRENUTA“, a drugi pri završetku iste objavljuje poruku „SIMULACIJA GOTOVA“ na temu virtualnog senzora. Time se sprječava zadržavanje stare poruke na istoj temi i na preglednoj ploči prekidom simulacije.

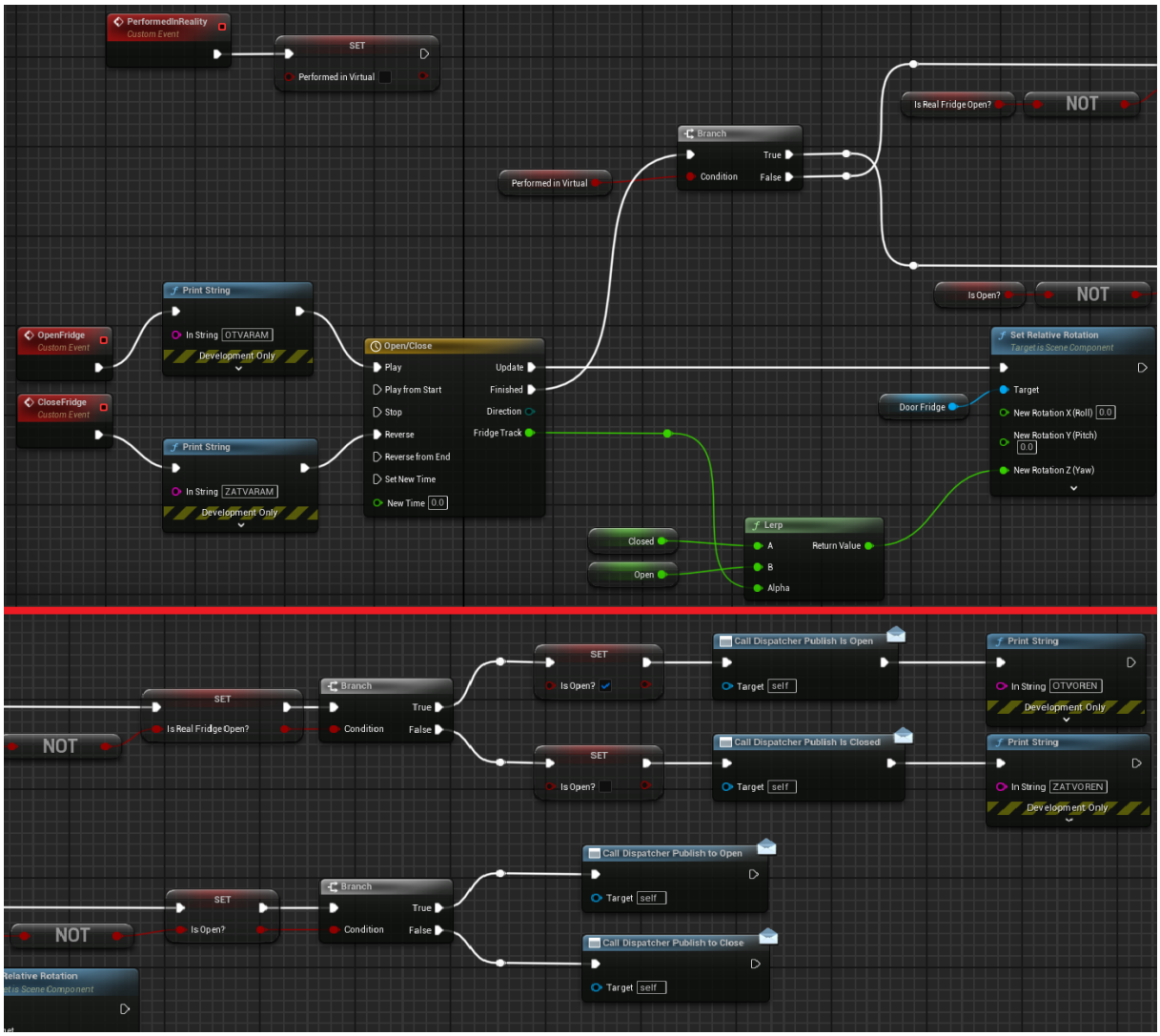
Prije nego se prikažu objavljivačke i pretplatničke logike hladnjaka, potrebno je prikazati neke izmjene u logici funkcije „*Interact“* (slika 57.) i *EventGraph*-a (slika 58.) *Blueprint*-a *„BP_Fridge“*. U funkciji „*Interact“* je dodano samo definiranje vrijednosti *true* varijable *„PerformedInVirtual“* pri svakoj interakciji.



Slika 57. Izmijenjena funkcija „*Interact*“ u *"BP_Fridge"*

S druge strane, *Event Graph* istog *Blueprint*-a sadrži nekoliko promjena. Prva je grananje smjera logika nakon odvijanja animacije ovisno o tome je li radnja pokrenuta u virtualnom ili stvarnom okruženju, odnosno o vrijednosti varijable *„PerformedInVirtual“*. Nakon toga, svaki od ta dva slučaja se dijele na još dva smjera, ovisno o vrijednosti potrebnih varijabli. Kada se radnja pokreće u virtualnom okruženju, pregledava se vrijednost varijable *„IsOpen?“* na

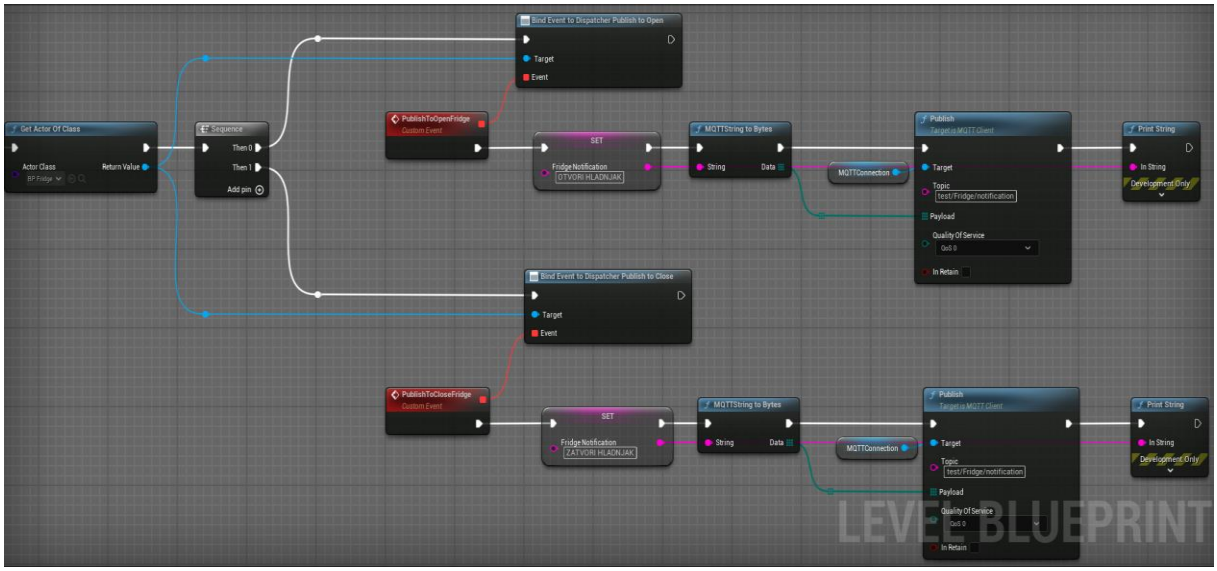
temelju koje se pozivaju *Event Dispatcher*-i za povezivanje *Bluepint*-a *„BP_Fridge“* i *Level Blueprint*-a te za objavu obavijesti brokeru na temu virtualnog senzora. Suprotno, kada je radnja pokrenuta u stvarnosti, gleda se vrijednost varijable *„IsRealFridgeOpen?“* na temelju koje se postavljaju nove vrijednosti varijable *„IsOpen?“* radi sinkronizacije dvaju okruženja te pozivaju *Event Dispatcher*-i za objavu potvrde stanja hladnjaka brokeru na temu virtualnog senzora. Na kraju, u *Event Graph*-u se nalazi i događaj stvoren od strane korisnika imena *„PerformedInReality“* koji se poziva iz *Level Blueprint*-a pri završetku logičke cjeline određivanja programa kao klijenta pretplatnika na temu brokera stvarnog hladnjaka te koji određuje vrijednost *false* unutar varijable *„PerformedInVirtual“*.



Slika 58. Izmjena *"BP_Fridge" Event Graph*-a

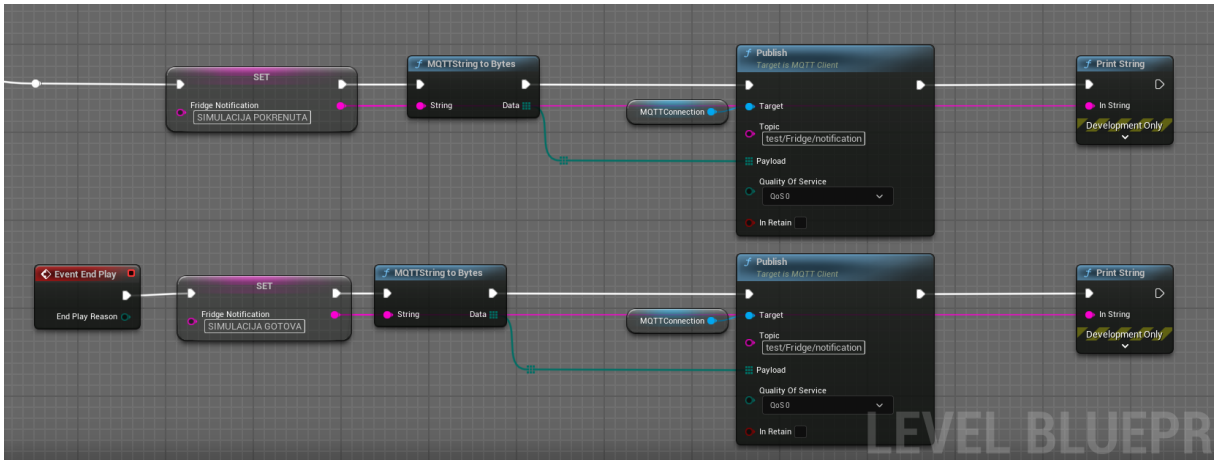
7.3.1. Publisher hladnjaka

Na slici 59. je prikazana logička cjelina određivanja Unreal Engine 5 programa kao klijenta objavljivača poruke na temu *„test/Fridge/notification“* virtualnog senzora. Pronalaženjem *Blueprint* klase *„BP_Fridge“*, njenom implementacijom u *Level Blueprint* te nastavkom radnje pri pozivu odgovarajućeg *Event Dispatcher*-a pokreću se događaji *„PublishToOpenFridge“* i *„PublishToCloseFridge“* na temelju vrijednosti varijable *„IsOpen?“*. Budući da virtualni senzor definiran u Home Assistant-u nije stvaran, već postoji kao reprezentacija istoga za prikaz tekstualne obavijesti, *Payload* JSON za razmjenu informacija preko spomenute teme ne postoji i nije definiran kao što je slučaj stvarnog hladnjaka prikazanog na slici 56. Stoga, nije potrebno stvaranje nove *Blueprint* strukture kao što je slučaj kod objave podataka na temu pametne rasvjete. Suprotno, definira se *string* varijabla *„FridgeNotification“* u koju se sprema tekst koji se želi objaviti u pojedinim slučajevima. Konačno, pomoću funkcije *„MQTTStringToBytes“* se željeni *string* pretvara u niz *byte*-ova te objavljuje na temu funkcijom *Publish.*



Slika 59. *Fridge Publisher*

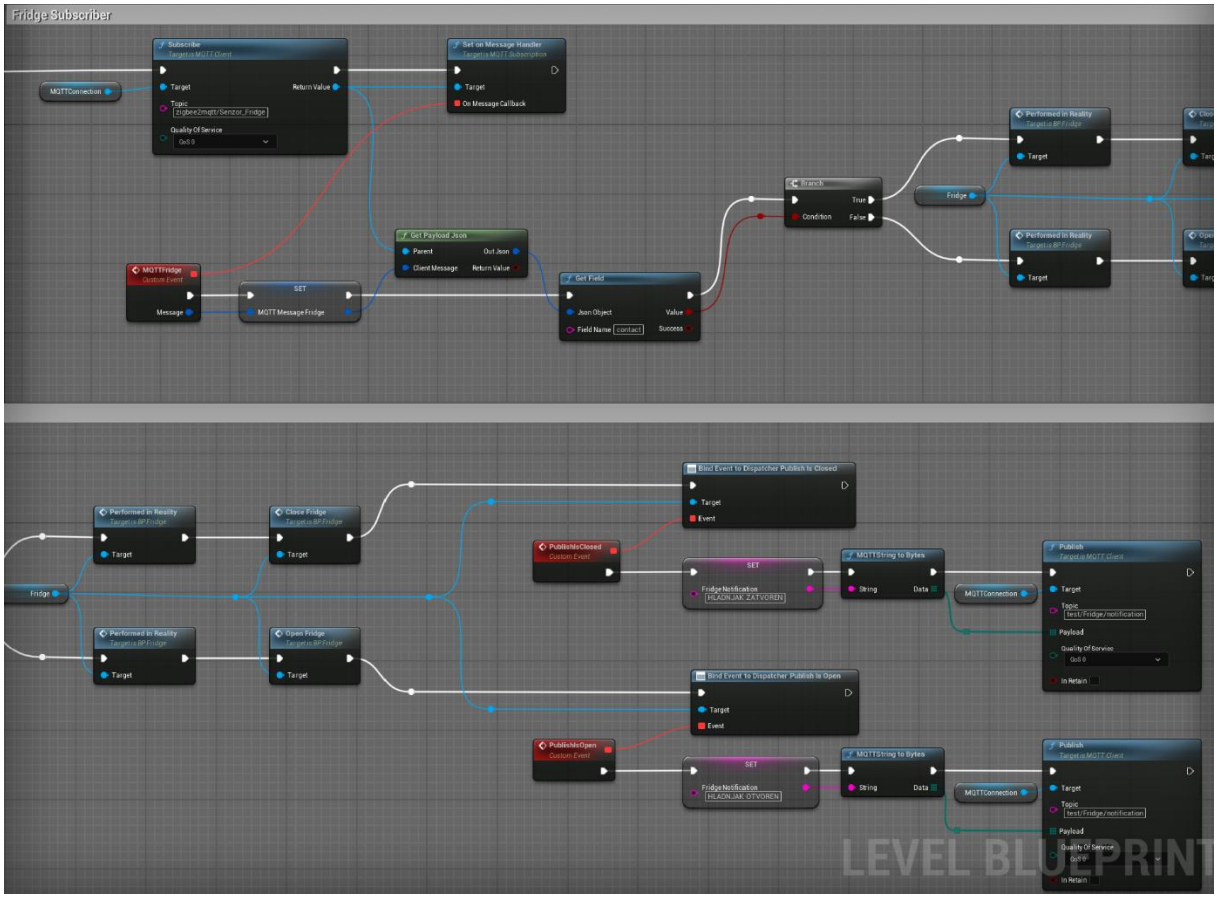
Također, u konceptu komunikacije hladnjaka je spomenuto i postojanje objave obavijesti na istu temu pri početku i završeku simulacije. Ta logika objavljivanja je prikazana na slici 60., a identična je prethodnom slučaju s malim razlikama. Tijek objave „SIMULACIJA POKRENUTA“ je spojen direktno na jedan od *Sequence pin*-ova nakon ostvarivanja povezivanja s brokerom čime je posredno povezan na *Event BeginPlay*, a tijek objave „SIMULACIJA GOTOVA“ je povezan na *Blueprint* događaj *Event EndPlay.*



Slika 60. *Fridge Publisher* početnog i konačnog stanja simulacije

7.3.2. Subscriber hladnjaka

Na slici 61. prikazana je logika pretplate Unreal Engine 5 programa na temu stvarnog hladnjaka koja će ovisno o vrijednosti primljenog podatka utjecati na virtualni model iste.

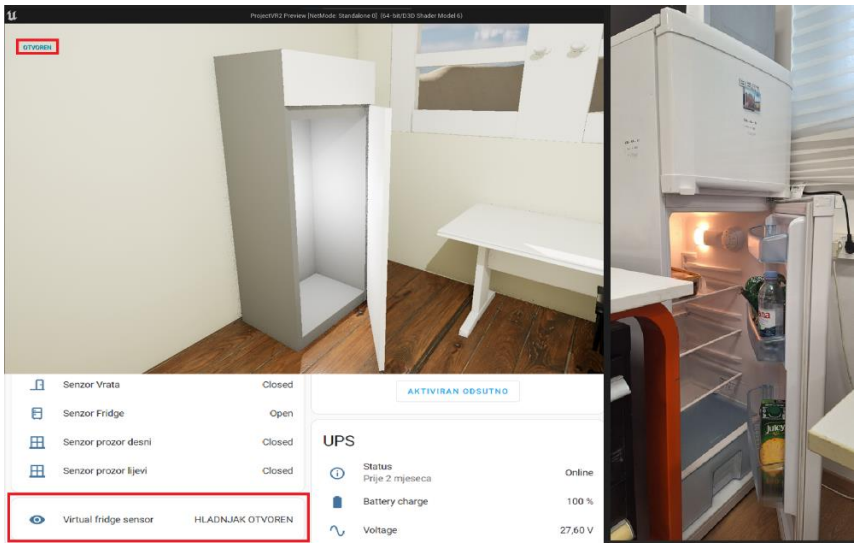


Slika 61. *Fridge Subscriber*

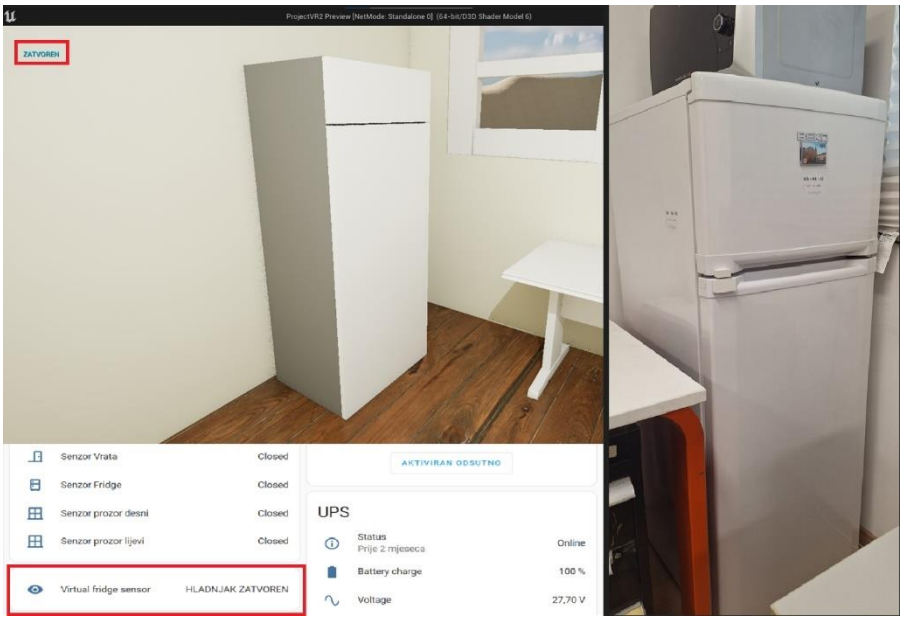
Slično kao i kod pretplate pametne rasvjete, definiranjem programa kao klijenta pretplatnika na temu *„zigbee2mqtt/Senzor_Fridge“* prati se trenutno stanje kontaktnog senzora stvarnog hladnjaka. U ovom slučaju se prati vrijednost polja „*contact*“ *Payload* JSON-a prikazanog na slici 56. koja je *true* ako je hladnjak zatvoren, a *false* ako je isti otvoren. Otvaranjem ili zatvaranjem stvarnog hladnjaka se čita ta vrijednost te se pozivanjem događaja *„PerformedInReality“* u *Blueprint*-u *„BP_Fridge“* pokreće postavljanje *false* vrijednosti u varijablu *„PerformedInVirtual“* kako bi se definiralo pokretanje radnje u stvarnom okruženju u oba slučaja. Tada se odvija animacija u virtualnom okruženju koja odražava promjenu stanja stvarnog hladnjaka. Pri završetku animacije se tijek logike u *„BP_Fridge“* usmjerava sukladno s vrijednosti varijable *„PerformedInVirtual“* te se pomoću *Event Dispatcher*-a *„DispatcherPublishIsOpen“* i *„DispatcherPublishIsClosed“* ponovno nastavlja u *Level Blueprint*-u. Ovisno o vrijednosti varijable *„IsRealFridgeOpen?“* pokreću se događaji definirani od strane korisnika imena *„PublishIsOpen“* i *„PublishIsClosed“* koji objavljuju obavijesti „OTVOREN“ ili „ZATVOREN“ na temu virtualnog senzora.

7.3.3. Rezultati

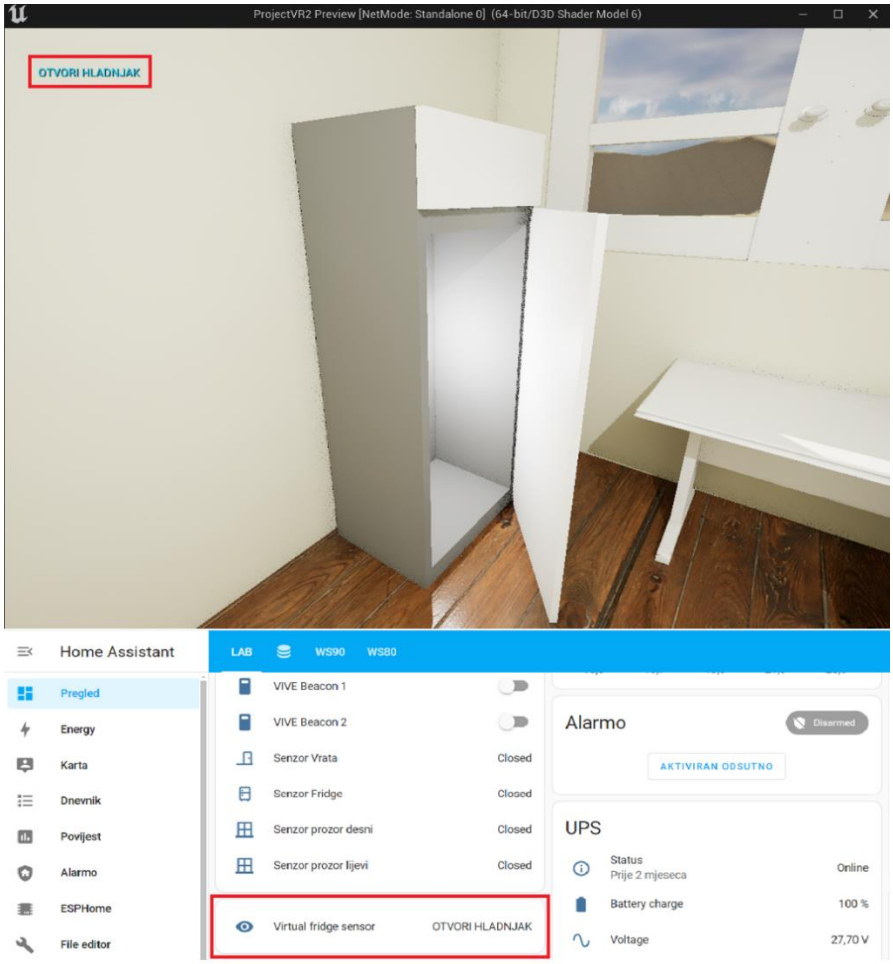
Prethodnim je ostvarena dvosmjerna komunikacija hladnjaka unutar stvarnog i virtualnom okruženja. Otvaranje i zatvaranje hladnjaka u laboratoriju kontinuirano otvara i zatvara hladnjak u virtualnom modelu i ispisuje trenutnu obavijest stanja na preglednoj ploči Home Assistant-a. Međutim, otvaranje i zatvaranje istog u virtualnom modelu zahtijeva posrednika koji će pročitati ispisanu uputu i obaviti radnju u stvarnom okruženju.



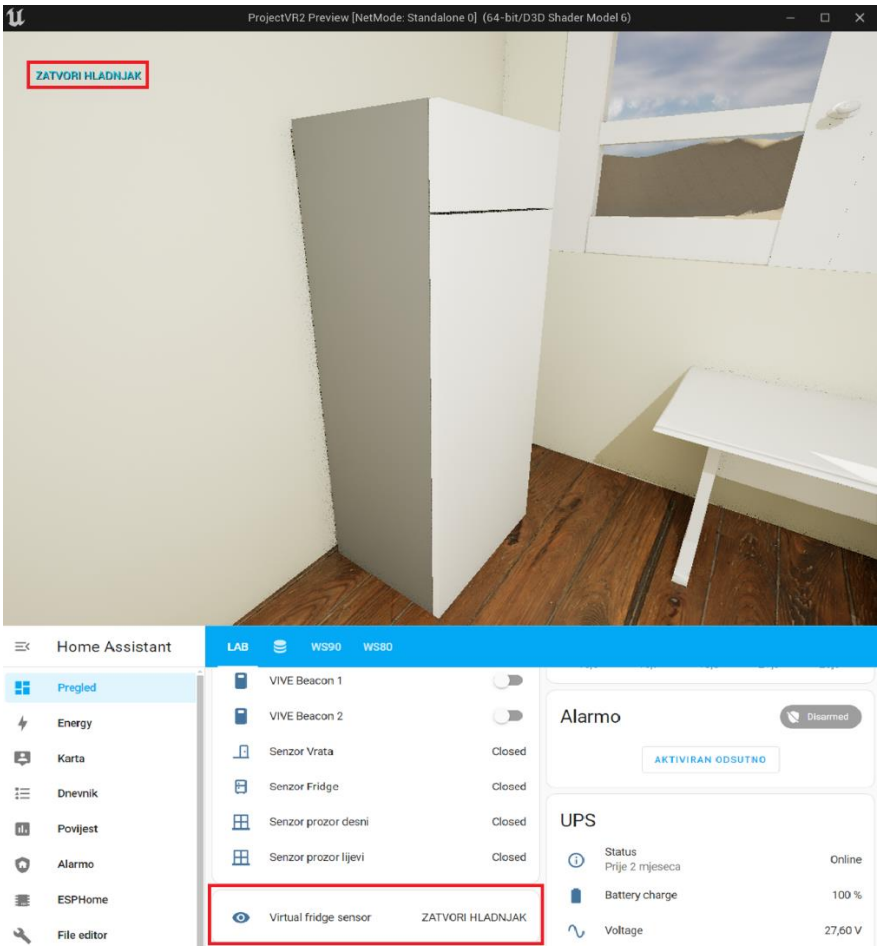
Slika 62. Hladnjak otvoren u stvarnom okruženju



Slika 63. Hladnjak zatvoren u stvarnom okruženju



Slika 64. Hladnjak otvoren u virtualnom okruženju



Slika 65. Hladnjak zatvoren u virtualnom okruženju

8. KRITIČKI OSVRT

Ostvarivanjem dvosmjerne komunikacije temeljenje na MQTT protokolu unutar virtualnog i stvarnog okruženja te povezivanjem događaja unutar istih uz integraciju postojećeg rješenja pametnog prostora laboratorija s Home Assistant platformom radi daljinskog upravljanja postignut je cilj ovog diplomskog zadatka. Nadalje, u procesu postizanja istoga i proučavanja tehnologija koje su korištene može se zaključiti sljedeće.

Home Assistant je moćna i prilagodljiva platforma za upravljanje pametnim prostorima koja podržava širok spektar uređaja i integracija. Njena je glavna prednost što je platforma otvorenog koda koja osim visoke razine prilagodbe posjeduje široku zajednicu korisnika koja neprestano pridonosi stalnom razvoju i poboljšanju sustava. S druge strane, također omogućuje kontrolu privatnosti podataka.

Unreal Engine 5 nudi napredne grafičke mogućnosti što omogućuje visoku razinu detalja i stvaranje preciznih modela stvarnog svijeta što je posebno korisno za stvaranje digitalnih blizanaca. Osim stvaranja vizualno vjerodostojnih modela, mogućnost programiranja i razvoja kompleksnih funkcionalnosti i interakcija s modelom čini ga potpuno legitimnim izborom za ostvarivanje dvosmjerne komunikacije i povezivanje događaja stvarne i virtualne okoline iako u početcima razvoja nije bio zamišljen za tu svrhu. Uz C++ programski jezik, posebno se ističe *Blueprint* sustav programiranja koji bez potrebe pisanja koda omogućuje implementaciju funkcionalnosti na intuitivan način. Međutim, UE5 je zahtjevan u smislu računalne snage, što može predstavljati problem za kontinuiranu i glatku simulaciju kod složenih i brojnih operacija. Zbog toga, optimizacija resursa i posjedovanje naprednih tehnika programiranja ključne su za optimalne performanse.

MQTT je lagani protokol za razmjenu poruka po principu objavljivanja i pretplate na temu brokera, idealan i najrasprostranjeniji za aplikacije interneta stvari zbog svoje jednostavnosti i efikasnosti. Također, implementacijom MQTT *plugin*-a u Unreal Engine-u 5 proširuje se njegova primjena na ostvarivanje komunikacije i razmjenu informacija između uređaja u stvarnom svijetu i njihovih virtualnih ekvivalenata čime se može razviti interaktivni digitalni blizanac. No, zbog eksperimentalnog stadija razvoja *plugin*-a, još uvijek postoje neke poteškoće (poput nemogućnosti čitanja *null* vrijednosti s pretplaćene teme bez pada programa). Najizraženija poteškoća je ta što iako postoji logika za trenutnu pretplatu i čitanje poruke s određene teme, stanje u virtualnom modelu se odražava na stvarno okruženje tek nakon prvog poticaja promjene istoga. S vremenom to neće biti slučaj i poteškoće će se ukloniti, a unatoč njima kompletni sustav za povezivanje događaja obje okoline više je nego zadovoljavajući.

9. ZAKLJUČAK

Kreiranje digitalnog blizanca u Unreal Engine 5 programu i ostvarivanje dvosmjerne komunikacije putem MQTT protokola između stvarnog i virtualnog okruženja kako bi se moglo povezati događaje u istima, uz integraciju Home Assistant-a za daljinsko upravljanje, predstavlja značajan korak u modernizaciji i unapređenju inteligentnih sustava za automatizaciju. Kroz ovaj proces, dobiva se detaljan i dinamičan model stvarnog svijeta koji omogućuje preciznu simulaciju i kontrolu te vizualno primanje informacije različitih scenarija i operacija u realnom vremenu. Takav koncept pokazuje svrhu na temelju koje ova tehnologija nalazi široku primjenu u raznim industrijama gdje se može značajno unaprijediti efikasnost i pouzdanost proizvodnih procesa.

Praktični primjeri su industrijska postrojenja gdje se digitalni blizanac koristi za nadzor i optimizaciju proizvodnih linija. Izvršavanjem određenih radnji u stvarnom svijetu, poput prekida rada stroja, isti se trenutno replicira u virtualnom modelu omogućujući inženjerima da brzo reagiraju, analiziraju uzroke i implementiraju rješenja. Korištenjem prediktivnog održavanja, digitalni blizanac može pratiti rad strojeva u stvarnom vremenu, prikupljajući podatke o vibracijama, temperaturi, tlaku i drugim ključnim parametrima te ih vizualno prikazati, a na temelju istih sustav može predvidjeti potencijalne kvarove prije nego što se dogode, smanjujući troškove održavanja i povećavajući dostupnost strojeva. S druge strane, povezivanje događaja u stvarnom i virtualnom okruženju može dramatično poboljšati učinkovitost i sigurnost. Na primjer, digitalni blizanac može automatski prilagoditi postavke strojeva u stvarnom vremenu kako bi optimizirao proizvodni proces. Također, utjecaj virtualnog modela na određene scenarije stvarnog okruženja je posebno korisno za obuku novih operatera, koji mogu vježbati u virtualnom okruženju prije nego što počnu raditi sa stvarnim strojevima ili se jednostavno obrazovati na daljinu.

Optimizacija rada strojeva i proizvodnih procesa, povećanje produktivnosti i smanjenje troškova, kontrola podataka i upravljanje sustavom, a i fleksibilnost radnog mjesta (što s vremenom postaje sve bitniji aspekt zaposlenja) samo su neke od prednosti ovakvih rješenja zbog kojih imaju veliki potencijal u industriji.

LITERATURA:

[1] Jeong, D. -Y. et al. (2022) „Digital Twin: Technology Evolution Stages and Implementation Layers With Technology Elements" *IEEE Access* (Pristupljeno 2024-5-15)

[2] How to build a digital twin. URL: https://www.sekai.io/blog/how-to-build-a-digital-twin (Pristupljeno 2024-5-15)

[3] MQTT.org. URL: https://mqtt.org/ (Pristupljeno 2024-5-20)

[4] Research Gate. URL: https://www.researchgate.net/figure/Structure-of-typical-ZigBee-mesh-network_fig2_337444074 (Pristupljeno 2024-6-1)

[5] Sonoff. URL: https://sonoff.tech/product-document/gateway-and-sensors-doc/snzb-04-doc/ (Pristupljeno 2024-6-6)

[6] Aqara. URL: https://www.aqara.com/en/product/smart-wall-switch-h1-eu-with-neutral/ (Pristupljeno 2024-6-6)

[7] Sonoff. URL: https://sonoff.tech/product-document/gateway-and-sensors-doc/zigbee-dongle-plus-cc2652p-doc/ (Pristupljeno 2024-6-6)

[8] Home Assistant. URL: https://www.home-assistant.io/ (Pristupljeno 2024-6-10)

[9] Blueprint visual scriptnig overview. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/overview-of-blueprints-visual-scripting-in-unreal-engine (Pristupljeno 2024-5-27)

[10] Programming with C++ in Unreal Engine. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/programming-with-cplusplus-in-unreal-engine?application_version=5.3 (Pristupljeno 2024-5-27)