

**Profesionālās izglītības kompetences centrs
„RĪGAS VALSTS TEHNİKUMS”**

DATORIKAS NODAĻA

Izglītības programma: Programmēšana

KVALIFIKĀCIJAS DARBS

“Anticarium” viedā terārija sistēma

Audzēknis:

Mārtiņš Smirnovs

Skolotājs:

Ilona Demčenko

2021./2022. m.g.

Rīga

ANOTĀCIJA

Kvalifikācijas darbā ir aprakstīts “Anticarium” viedā terārija sistēmas izstrādes process. Sistēma ļauj aukstasiņu mājdzīvnieku īpašniekiem, attālināti, iestatīt mājdzīvnieka terārijā uzturamos apstākļus – gaisa siltumu, zemes mitrumu, gaisa un gaismas padevi. Papildus, sistēma piedāvā video tiešraidē sekot līdzi terārija mājdzīvniekam, kā arī veidot un pārvaldīt, lietotāja veidotos, terārija uzturamo apstākļu režīmus. Projekts tika izstrādāts izmantojot C++, Python, Bash, CMake programmēšanas valodas, SQLite relācijas datubāzes sistēmu, vairākas dažāda veida ierīces un procesorus.

Kvalifikācijas darbs ietver ievadu, uzdevumu nostādni, kvalifikācijas darbā lietotos terminus, prasību specifikāciju, datu struktūru aprakstu, programmatūras produkta modelēšanas un projektēšanas aprakstu, lietotāja ceļvedi, nobeigumu, izmantotos avotus un pielikumus.

Kvalifikācijas darba ievadā ir aprakstīta aukstasiņu dzīvnieku terāriju kopšanas aktuālā problēma un tās risinājumus. Uzdevumu nostādnē ir norādīti uzdevumi, kurus sistēmai būs nepieciešams veikt. Terminu sadaļā tiek aprakstīti visi svarīgkie darbā lietotie termini. Prasību specifikācija sastāv no iecjas un izejas informācijas, kā arī no sistēmas funkcionālajām un nefunkcionālajām prasībām, kādi līdzekļi tiks izmantoti izstrādāšanai un kādiem nolūkiem tie tiek izmantoti. Datu struktūru aprakstā tiek aprakstīta datu bāzes struktūra, kā arī svarīgākās datu struktūras, kuras tiek lietotas datu apmaiņā starp dažādām sistēmas daļām. Programmatūras produkta modelēšanas un projektēšanas apraksts sastāv no sistēmas struktūras modeļa, kas ietver sistēmas kopējo arhitektūru un katras sistēmas komponentes arhitektūru. Lietotāja ceļvedī ir norādītas nepieciešamās sistēmas prasības aparātūrai un programmatūrai, izstrādātāja izstrādes vides instalācija un palaišana, programmas uzstādīšanas lietotājam, kā arī programmas apraksts, kas paskaidro kā lietot sistēmu. Papildus lietotāja ceļvedī ir arī norādīti divi sistēmas testa piemēri, kur viens no testiem pārbauda terārija darbību un otrs tests pārbauda režīmu pārvaldīšanas pareizu darbību.

Kvalifikācijas darbs sastāv no 118 lappusēm, kurā ietilpst 65 pielikumi, kas satur visus attēlus un tabulas, kas tiek pieminēti un izmantoti kvalifikācijas darbā.

ANNOTATION

The final project describes development process of “Anticarium” smart terrarium system. The system allows owners of cold-blooded pets to control their terrarium’s physical environment – air temperature, soil moisture, light brightness and air blowing intensity. Additionally the system gives the possibility to create and manage regimes by saving and editing environment parameters, and watch for the pet remotely using live video output from terrarium. The project was made using C++, Python, Bash, CMake programming languages, SQLite relational database management system, different devices and processors.

The final project includes introduction, assignment, common terms used in project, the specification of requirements, description of data structures, description of the software product modeling and design, user guide, conclusions, used sources and attachments.

In the introduction part of the project you can find description of current problems and solutions regarding care of cold-blooded animals terrariums. In the assignment part you can find tasks that the system will have to do. Common terms used part contains most important and common terms used here. The specification of requirements contains system’s input and output information, functional and non-functional requirements, tools used along with their description. Description of data structures section contains database structure and most important data structures that are used in data exchange between different parts of the system. In the description of the software product modeling and design part you can find an architecture of the whole system and also architecture of each component in the system. In user guide there are minimal system requirements for software and hardware, development environment setup guide, user environment setup guide, description on how to use the system. In user guide section you can also find two test cases for the system which can test physical input, output of “Anticarium” system and also correct working of regime management.

The final project consists of 118 pages including 65 attachments which contain all pictures and tables mentioned and used in the final project.

SATURS

IEVADS.....	5
1. UZDEVUMA NOSTĀDNE	6
2. PRASĪBU SPECIFIKĀCIJA	7
2.1. Ieejas informācijas apraksts	7
2.2. Izvades informācijas apraksts.....	7
2.3. Funkcionālās prasības	8
2.4. Nefunkcionālās prasības.....	11
2.5. UZDEVUMA RISINĀŠANAS LĪDZEKĻU IZVĒLES PAMATOJUMS	13
4. PROGRAMMATŪRAS PRODUKTA MODELĒŠANA UN PROJEKTĒŠANA	21
4.1. Anticarium elementu struktūras apraksts	21
4.2. Sistēmas struktūras apraksts.....	24
4.3. Sistēmas projektēšanas procesa apraksts.....	25
5. DATU STRUKTŪRU APRAKSTS	37
5.1. shared_types	37
5.2. Datu uzglabāšana datubāzē	41
5.3. Datu apmaiņas formāts izmantojot I ² C	42
5.4. Video nosūtīšanas formāts	44
6. LIETOTĀJA CEĻVEDIS	46
6.1. Izstrādātājam	46
6.2. Lietotājam	58
6.3. Anticarium Desktop apraksts	59
6.4. Testa piemērs	60
NOBEIGUMS	63
TERMINI.....	64
IZMANTOTIE AVOTI.....	65
PIELIKUMI.....	68

IEVADS

Dažiem cilvēkiem kā mājdzīvnieks pieder kaķis, citam pieder suns. Kādam citam mēdz būt kāds neierastāks mājdzīvnieks, kā, piemēram, putns vai pele. Taču ir arī sabiedrības daļa, kas sev kā mājdzīvnieku tur kādu aukstasiņu dzīvnieku. Tas var būt gan rāpulis, gan kukainis, gan dažāda veida zirnekļi. Bieži vien šāda veida dzīvnieki dzīvo noteiktos apstākļos un šos apstākļus nodrošina terārijs.

Parasts terārijs pamatā ir stikla kaste, un lai tur mākslīgi izveidotu dzīvniekam nepieciešamos apstākļos, tad īpašniekam vajag salīdzinoši bieži ar to darboties. Terāriju ir nepieciešams aplaistīt, ieslēgt apsildīšanu, izvēdināt, kā arī nedrīkst aizmirst par paša mājdzīvnieka kopšanu. Tā ir problēma, jo tās ir ikdienišķas darbības, kas aizņem laiku.

Kā labāks risinājums būtu iepriekšminēto darbību automatizācija izmantojot viedo terāriju, kas pats būtu spējīgs nodrošināt aplaistīšanu, siltuma padašanu, vēdināšanu un gaismu. Šīs funkcijas ir jāspēj darbināt caur lietotni gan manuāli, gan iestatot automātiskus režīmus, kā arī nepieciešams izveidot video novērošanu, ko lietotne atrādītu. Terārija fiziskajai daļai ir jābūt viegli noņemamai un izjaucamai, tādējādi ļaujot to ērti iztīrīt nepieciešamības gadījumā.

Viedais terārijs kopā ar datora lietotni tiek paredzēts darbam izmantojot interneta tīklu. Katram terārijam pastāvēs tikai viens lietotājs, kas atvieglo sistēmas izstrādi. Terārija veiksmīgai darbībai caur internetu būs nepieciešams veikt portu pāradresāciju rūterim terārija UDP un TCP ieejām.

1. UZDEVUMA NOSTĀDNE

Šī kvalifikācijas darba mērķis ir izveidot viedā terārija sistēmu. No lietotāja saskarnes datora ekrānā, līdz pat paša terārija automātiskai laistīšanai. Sistēma ļaus lietotājam iestatīt savam mīlulim labākos dzīves apstākļus tam pat neatrodoties blakus, attālināti.

Kvalifikācijas darba izveides rezultātā tiek plānots izveidot ne tikai programmatisko daļu, lietotni un terārija vadību, bet arī fizisko terāriju. Ar automatizētu terāriju lietotājam vairs nebūs jāuztraucas par aplaistīšanas biežumu vai arī par to ka mājdzīvniekam varētu būt pārlieku auksts vai karsts.

Terārija galvenās funkcionalitātes:

- lietotāja iestatīto apstākļu automātiska uzturēšana terārijā;
- videonovērošana, atrādot video lietotnē;
- lietotne ļauj lietotājam vadīt terāriju attālināti;
- iespēja lietotājam izveidot, saglabāt, rediģēt uzturamos apstākļus, kā režīmus;
- esošo apstākļu atrādīšana lietotnē (temperatūra, mitrums);
- modulārs terārija dizains, kas ļauj to viegli izjaukt un veikt apkopi;
- ērta piekļuve pie mājdzīvnieka, terārijā, izmantojot durtiņas/lūkas.

2. PRASĪBU SPECIFIKĀCIJA

2.1. Ieejas informācijas apraksts

Anticarium sastāvēs no vairākām daļām. Papildus lietotāja ievadītajiem datiem iekš Anticarium Desktop, būs arī dati ko ievāc pats terārijs, tos apstrādās un pados tālāk. Dati tiks saņemti un padoti vairākos posmos. Turpmākajās sadaļās tiek aprakstīti tikai tie dati, kas tiek ievadīti pirmo reizi un nav apstrādāti/saņemti no citas sistēmas daļas.

2.1.1. *Anticarium Desktop ieejas dati*

Dati ienāk no lietotnes saskarnes.

1. Saglabātā režīma izvēle [Režīma nosaukums].
2. Gaismas spilgtums [0%-100%].
3. Uzturamais zemes mitrums [0%-100%].
4. Uzturamā gaisa temperatūra [15°C-40°C].
5. Ventilatora griešanās ātrums [0%-100%].

2.1.2. *Anticarium PCB ieejas dati*

Visi ieejas dati iekš Anticarium PCB notiks izmantojot Input IC. Dati tiek nolasīti izmantojot sensorus

1. Gaisa mitrums [0%-100%].
2. Gaisa temperatūra [-40°C -80°C].
3. Zemes mitrums [0-1023].

2.1.3. *Anticarium Camera ieejas dati*

1. RGB attēls [1280px x 960px].

2.2. Izvades informācijas apraksts

Šajā sadaļā tiek aprakstīti izejas dati no visām Anticarium daļām (izņemot Anticarium WEB).

2.2.1. *Anticarium Desktop izvades dati*

1. Anticarium Desktop atrāda terārija gaisa temperatūru.
2. Anticarium Desktop atrāda terārija gaisa mitrumu.
3. Anticarium Desktop atrāda terārija zemes mitrumu.
4. Anticarium Desktop atrāda video tiešraidi no terārija.
5. Anticarium Desktop atrāda iepriekš saglabātos režīmus.

2.2.2. *Anticarium Camera izvades dati*

1. Katrs uzņemtais attēls no kameras tiek sadalīts sīkākās vienībās un nosūtīts uz Anticarium Desktop.

Anticarium Pi izvades dati

Visi izvades dati no Anticarium Pi notiks uz Anticarium PCB, kur tos tālāk saņem Output IC.

1. Atbilstoši PID algoritmam, terārija gaisa temperatūrai un ievadītajai uzturamajai temperatūrai, automātiski tiek pieņemts lēmums par sildāmā elementa iedarbināšanu.
2. Atbilstoši PID algoritmam, terārija zemes mitrumam un ievadītajam uzturamajam zemes mitrumam, automātiski tiek pieņemts lēmums par ūdens padeves iedarbināšanu.
3. Ūdens padeve tiek apstādināta, ja ievadītais zemes mitrums ir 0%.
4. LED apgaismojuma spilgtums pārveidots no skalas 0-100 uz 0-255 tiek padots uz Output IC.
5. Ventilatora griešanās ātrums pārveidots no skalas 0-100 uz 255-0, tiek padots uz Output IC.
6. Ventilatora izslēgšana tiek padota uz Output IC, ja tā ievadītais ātrums ir 0%.

Output IC izvades dati

Output IC pārveidos padotās vērtības digitālos signālos, kas atbilstoši darbinās kādu no pieslēgtajām elektroniskajām komponentēm izmantojot Anticarium PCB.

1. Saņemot vērtību, kas lielāka par 0 priekš kājiņas 8, tiek pieslēgta barošana ventilatoram.
2. Saņemot vērtību, kas lielāka par 0 priekš kājiņas 9, tiek iedarbināts sildošais elements.
3. Saņemot vērtību, kas lielāka par 0 priekš kājiņas 10, tiek iedarbināta ūdens padeve;
4. Saņemot vērtību uz kājiņu 3, atbilstoši vērtībai tiek iestatīts PWM signāla ilgums ventilatora ātruma regulēšanai.
5. Saņemot vērtību uz kājiņu 2, atbilstoši vērtībai tiek iestatīts PWM signāla ilgums apgaismojuma spilgtuma regulēšanai.

2.3. Funkcionālās prasības

Nākamajās sadalās tiek aprakstītas ne tikai programmatūras funkcionālās prasības, bet arī elektroniskās un fiziskās funkcionālās prasības priekš paša terārija.

2.3.1. Anticarium Desktop

1. Lietotne ļauj iestatīt LED apgaismojuma vērtību.
2. Lietotne ļauj iestatīt ventilatora griešanās ātruma vērtību.
3. Lietotne ļauj iestatīt uzturamo zemes mitrumu.
4. Lietotne ļauj iestatīt uzturamo gaisa temperatūru.
5. Lietotne ļauj saglabāt esošās iestatītās vērtības, kā režīmu:
 - 5.1. saglabājot vērtības kā režīmu ir nepieciešams jaunajam režīmam dot nosaukumu;
 - 5.2. tiek saglabāts iepriekš iestatītais uzturāmais zemes mitrums;
 - 5.3. tiek saglabāta iepriekš iestatītā uzturamā gaisa temperatūra.
6. Lietotne ļauj izvēlēties un iestatīt kādu no esošajiem saglabātajiem režīmiem.

7. Lietotne ļauj atrādīt visus saglabātos režīmus un to vērtības.
8. Lietotne ļauj mainīt saglabāto režīmu parametrus:
 - 8.1. režīma nosaukumu;
 - 8.2. uzturamo temperatūru;
 - 8.3. uzturamo zemes mitrumu.
9. Lietotne ļauj dzēst saglabātos režīmus.
10. Lietotne atrāda video tiešraidi no terārija:
 - 10.1. atrāda melnu ekrānu ja video nav pieejams.
11. Lietotne komunicē ar serveri izmantojot tīkla savienojumu.
12. Lietotne atrāda gaisa mitruma vērtību.
13. Lietotne atrāda zemes mitruma vērtību.
14. Lietotne atrāda gaisa temperatūras vērtību.
15. Ieejot lietotnē tiek atrādītas pēdējās iestatītās vērtības, ieskaitot režīmu, ja tas bija iepriekš iestatīts.
16. Ja lietotājs pamaina uzturamās temperatūras vai uzturamā zemes mitruma parametrus galvenajā skatā, tad tiek parādīts, ka pašlaik neviens režīms nav iestatīts.
17. Saglabājot jaunu režīmu, tas tiek iestatīts kā esošais režīms.
18. Dzēsot pašlaik iestatīto režīmu, tiek parādīts, ka pašlaik neviens režīms nav iestatīts.

2.3.2. *Anticarium WEB*

1. Pēc pieprasījuma saglabā padotā režīma vērtības datubāzē.
2. Pēc pieprasījuma redīgē padotā režīma vērtības datubāzē.
3. Pēc pieprasījuma dzēš padoto režīmu no datubāzes.
4. Lokāli uzglabā visas ienākošas pašlaik iestatītās vērtības:
 - 4.1. uzturamo gaisa temperatūru;
 - 4.2. uzturamo zemes mitrumu;
 - 4.3. ventilatora griešanās ātrumu;
 - 4.4. gaismas spilgtumu.
5. Restartējot serveri, pašlaik iestatītās vērtības netiek zaudētas.
6. Pēc pieprasījuma atgriež pašlaik iestatītās vērtības.
7. Lokāli uzglabā ienākošos sensoru datus:
 - 7.1. gaisa temperatūru;
 - 7.2. zemes mitrumu;
 - 7.3. gaisa mitrumu.
8. Restartējot serveri lokāli uzglabātie sensoru dati drīkst būt pazaudēti.
9. Pēc pieprasījuma saglabā esošā režīma ID numuru.
10. Pēc pieprasījuma atgriež esošā režīma ID numuru.
11. Pēc pieprasījuma atgriež sensoru datus.
12. Pēc pieprasījuma atgriež esošās pašlaik iestatītās vērtības.
13. Pēc pieprasījuma atgriež visu režīmu nosaukumus.
14. Pēc pieprasījuma atgriež visus saglabātos režīmus.
15. Pēc pieprasījuma atgriež esošo iestatīto režīmu:
 - 15.1. ja pašlaik nav iestatīts neviens režīms, atgriež esošās iestatītās vērtības, tukšu režīma nosaukumu, “-1” kā režīma ID numuru;
 - 15.2. ja pašlaik ir iestatīts kāds režīms, atgriež pašlaik iestatītā režīma vērtības.

2.3.3. *Anticarium Pi*

1. Pieprasa gaisa temperatūras datus no Input IC.
2. Pieprasa gaisa mitruma datus no Input IC.
3. Pieprasa zemes mitruma datus no Input IC.
4. Pieprasa esošās iestatītās vērtības no Anticarium WEB.

5. Automātiski uztur iestatīto gaisa temperatūru iekš terārija.
6. Automātiski uztur iestatīto zemes mitrumu iekš terārija.
7. Darbina ventilatoru atbilstoši iestatītajam ventilatora griešanās ātrumam.
8. Darbina LED apgaismojumu atbilstoši iestatītajai gaismas spilgtuma vērtībai.
9. Pārveido izvadāmās vērtības uz Output IC, lai atbilstoši ieslēgtu vai izslēgtu katru komponenti.
10. Apstrādā un nosūta korektus datus, kas tika saņemti no Input IC, lasot sensoru vērtības.

2.3.4. *Anticarium Camera*

1. Ielas attēlus no kameras.
2. Pēc nepieciešamības apstrādā attēlu.
3. Nosūta attēlu pēc pieprasījuma.

2.3.5. *Input IC*

1. Nolasa gaisa mitruma datus no sensora.
2. Nolasa gaisa temperatūras datus no sensora.
3. Nolasa zemes mitruma datus no sensora.
4. Pēc pieprasījuma nosūta ielasītos sensoru datus.

2.3.6. *Output IC*

1. Pēc pieprasījuma darbina LED apgaismojuma spilgtumu.
2. Pēc pieprasījuma darbina ventilatora ātrumu.
3. Pēc pieprasījuma darbina siltuma padevi.
4. Pēc pieprasījuma darbina ūdens padevi.

2.3.7. *Anticarium PCB*

Galvenie elektroniskie parametri, kuriem būs jāpiemīt Anticarium galvenajai vadības platei.

1. Ľauj vadīt 12V 1A LED diožu lenu PWM režīmā.
2. Ľauj vadīt 12V 0.25A terārija ventilatoru.
3. Ľauj vadīt 12V 0.25A terārija vakuumu ūdens sūkni.
4. Ľauj vadīt 220V 75W sildošo elementu, lampu.
5. Nodrošina komunikāciju starp Input IC, Output IC ierīcēm un Anticarium Pi.
6. Katrai komponentei nodrošina atbilstošus, atsevišķus kontaktus ērtai pieslēgšanai.

2.3.8. *Anticarium terārijs*

Galvenās īpašības, kurām būs jāpiemīt fiziskajam terārijam tā ērtai lietošanai.

1. Terārijam jābūt pietiekoši noslēgtam, lai caur to nevarētu izklūt kukaiņi kas ir lielāki par 0.5mm.
2. Terārijs spēj novadīt lieko ūdeni, lai nepieļautu tā nostāvēšanos.
3. Terārijam ir atveres gaisa plūsmai, lai nodrošinātu terārija ventilāciju.
4. Terārija sildošā lampa spēj uzturēt gaisa temperatūra, kas ir vismaz par 5°C lielāka nekā istabas temperatūra.
5. Terārijs nodrošina vienmērīgu ūdens padevi.

6. Terārija vadības bloka augšējā un sānu daļa ir hermētiski noslēgtas, gadījumam, ja tiek uzliets ūdens.
7. Terārijs nodrošina konteineri ūdens uzglabāšanai, laistīšanai.
8. Terārijs nodrošina ērtu piekļuvi ūdens konteinerim.
9. Terārijs ļauj ērti un viegli piekļūt pie tajā dzīvojoša dzīvnieka.
10. Terārija elektroniskās komponentes, kuras tiek pakļautas mitruma riskam, tiek attiecīgi noizolētas.
11. Terārijs ļauj ērti atvērt vadības bloku, ja to ir nepieciešams apkopt.
12. Terārijam tiek lietoti tikai divi kabeļi:
 - 12.1. barošanas kabelis 220V;
 - 12.2. CAT5e interneta kabelis.
13. Terārijs nodrošina vietu kameras iestatīšanai.
14. Terārijs lauj ērti atvienot kameru no terārija un nomainīt tās korpusa komponentes labākam kameras leņķim.
15. Detaļas, kas ir tiešā iedarbībā ar mitrumu terārijā tiek atbilstoši noizolētas.
16. Detaļas, kas ir tiešā iedarbībā ar mitrumu terārijā ir viegli noņemamas un nomaināmas.
17. Terārijā ir nodrošināta gaisa padeve terārija vadības blokam.

2.4. Nefunkcionālās prasības

2.4.1. Sistēmas izstrāde

1. Tā kā Anticarium sastāv no vairākām atsevišķām daļām, dažādām daļām ir nepieciešams veidot savu atsevišķu Github repozitoriju ērtākai versionēšanai, komponenšu atkārtotai izmantošanai, vieglākai uzturēšanai.
2. C++ koda formatešanai tiek izmantots clang-format rīks:
 - 2.1. koda formatešanai jābūt vienādai visos C++ failos;
3. Visas Github repozitorijas tiek versionētas izmantojot CHANGELOG.md un VERSION failus, katrā Github repozitorijā.

2.4.2. Anticarium Desktop, Anticarium Pi, Anticarium Camera

1. Papildus programmas darbības regulācijai tiek izmantots atsevišķs iestatījumu fails settings.ini.
2. Lietotnes galvenie notikumi tiek izvadīti log failos.
3. Log failos ierakstāmos datus var regulēt pēc to prioritātes (critical, error, warning, info, utt.), iestatījumu failā.
4. Servera(-ru) adrese ar kuriem komunicē lietotne var tikt iestatītas iestatījumu failā.

2.4.3. Anticarium Desktop

1. Saskarnes valoda ir angļu valoda.
2. Lietotnes sensoru informācijas pieprasījumu biežums var tikt iestatīts iestatījumu failā.
3. Lietotnes skices skatīt no 1. līdz 4. pielikumam.

2.4.4. Anticarium Pi

1. Sensoru datu nosūtīšanas biežumu var regulēt no iestatījumu faila.
2. Iestatīto vērtību pieprasīšanas biežumu no servera var regulēt no iestatījumu faila.

3. Sensoru datu pieprasīšanas biežumu no Input IC var regulēt no iestatījumu faila.

2.4.5. *Anticarium Camera*

1. Attēlu uzņemšanas biežumu sekundē (FPS) ir iespējams regulēt no iestatījumu faila.
2. Papildus ātrumam, attēls tiek sadalīts vairākās daļās atsevišķās procesu vītnēs (threads), pirms nosūtīšanas. Vītnu daudzumu ir iespējams regulēt no iestatījumu faila.

2.4.6. *Anticarium WEB*

1. Serveris tiek uzstādīts uz paša terārija galvenās vadības ierīces – Raspberry Pi [1]. Tas novērš nepieciešamību izmantot ārēju serveri.

2.4.7. *Raspberry Pi*

1. Galvenā Anticarium viedā terārija vadības ierīce.
2. Kā operētājsistēma tiek izmantota Raspbian minimālā versija bez grafiskā interfeisa.

2.5. UZDEVUMA RISINĀŠANAS LĪDZEKĻU IZVĒLES PAMATOJUMS

Anticarium izstrādes procesā tika izmantotas vairākas tehnoloģijas, metodes, bibliotēkas, lietotnes, programmas. Turpmākās sadaļās tiks izklāstīts katra izvēlētā rīka izvēles iemeslus katrai Anticarium daļai.

2.6. Programmēšanas valodas

3.1.1 C++

C++ tiek izmantota kā galvenā programmēšanas valoda šajā projektā. Izmantojot C++ ir saprogrammētas Anticarium Desktop, Anticarium Camera, Anticarium Pi, Input IC, Output IC programmas. C++ ir programmēšanas valoda, kas tiek kompilēta, kā arī liek izstrādātājam pašam atbildēt par programmas atmiņas izmantošanu. Šie un citi faktori padara C++ programmas ļoti ātras darbības laikā, skaitlošanas ziņā. C++ valoda tika izvēlēta šīm visām Anticarium daļām, jo, personīgi, ir plaša profesionāla pieredze šīs programmēšanas valodas izmantošanā, kā arī rezultējošās programmas ātrdarbības un efektivitātes iemeslu dēļ.

C++ priekš Anticarium Desktop tiek izmantots, jo ir profesionāla pieredze Desktop lietotņu izstrādē C++ valodā.

C++ priekš Anticarium Pi un Anticarium Camera tiek izmantots, jo tas uzlabo ātrdarbību uz Raspberry Pi, uz kura šīs programmas tiek izpildītas.

C++ priekš Input IC un Output IC tiek izmantots, jo darbs notiek ar nelielām mikroshēmām ar limitētu skaitlošanas jaudu un atmiņas daudzumu. Izmantojot C/C++ mūsdienās lielākoties tiek programmētas visas mikroshēmas.

2.6.1. CMake [2]

CMake ir Open-Source skriptu programmēšanas valoda, kas ļauj kontrolēt C++ programmas kompilācijas gaitu. Izmantojot CMake C++ projektam var piesaistīt papildus bibliotēkas, pakotnes, veikt instalāciju, testus utt. neatkarīgi no izvēlētās operētājsistēmas. CMake tika izvēlēts, jo tas ir C++ projektu izveides standarta rīks. Tieki izmantots visās Anticarium C++ daļās izņemot Input IC un Output IC.

2.6.2. Python [3]

Python ir programmēšanas valoda, kas bieži vien tiek izmantota ātrai programmu izstrādei. Python piedāvā lielu daudzumu iebūvētas funkcionalitātes. Anticarium projekta ietvaros šī valoda tika lietota Anticarium WEB izstrādei.

2.7. Ietvari (Frameworks)

2.7.1. *Qt* [4]

Qt ir Open-Source C++ ietvars lietotņu un parastu C++ programmu izstrādei. Qt ir C++ lietotņu izstrādes standarta izvēle jau vairāk nekā 20 gadus. Šis ietvars ļauj izmantot standarta grafiskās komponentes lietotnēs, kā arī izmantojot C++, rada lielāku priekšroku ātrdarbībā salīdzinot ar citām programmēšanas valodām. Qt ietvaru var izmantot gan grafisku lietotņu izstrādei, gan arī parastām C++ programmām, jo Qt piedāvā plašu bibliotēku ar dažādām noderīgām komponentēm. Qt tiek izmantots iekš Anticarium Desktop, Anticarium Pi, Anticarium Camera.

Qt tika izvēlēts, jo atvieglo C++ izstrādi, rada programmas struktūru, atbild par datora atmiņas satīrīšanu izmantojot parent-child arhitektūru. Tieki izmantots visās Anticarium C++ daļās izņemot Input IC un Output IC.

2.7.2. *Arduino* [5]

Arduino ir hardware un software uzņēmums, kas ir izveidojis Arduino C++ ietvaru. Arduino ietvars ļauj viegli saprogrammēt mikroshēmas, izmantojot jau gatavas funkcijas. Šis ietvars tiek izmantotots Input IC un Output IC programmēšanā.

2.7.3. *Flask* [6]

Flask ir Python programmēšanas valodas ietvars, kas tiek izmantots WEB serveru izstrādei. Flask ļauj ātri izveidot WEB serveri Python vidē un to atklūdot. Tieki izmantots, jo ir pieredze Flask izstrādē.

2.8. Bibliotēkas

2.8.1. *C++ Standard Template Library* [7]

STL ir C++ bibliotēku kopums, kas ietver sevī visbiežāk izmantotās datu struktūras, funkcijas, algoritmus. Tieki izmantota visās Anticarium projekta daļās izņemot Input IC un Output IC, jo dotajām mikroshēmām ir pārāk mazs atmiņas daudzums šādām bibliotēkām.

2.8.2. *google/googletest* [8]

GoogleTest ir C++ programmas pirmkoda testu izveides bibliotēka. Šī bibliotēka tika izvēlēta, jo ir nepieciešama kāda ērta testu rakstīšanas bibliotēka. Fakts, ka šī bibliotēka nāk no uzņēmuma “Google” ir pietiekami iedrošinošs, lai izvēlētos tieši šo kā testēšanas rīku. Tieki izmantota visās Anticarium C++ daļās izņemot Input IC un Output IC.

2.8.3. nicholastmosher/PID [9]

PID ir algoritms, kas tiek plaši izmantots industriālajā nozarē, lai regulētu kādu fizisku parametru. Šajā projektā ir divi fiziskie parametri terārijā, kas ir jāvada – zemes mitrums un gaisa temperatūra. Dotā C++ bibliotēka sastāv tikai no diviem failiem un ir ērti implementējama iekš Anticarium Pi.

2.8.4. nlohmann/json [10]

JSON ir standarta datu formāts un datu apmaiņas formāts. Tas tiek izmantots datu apmaiņā starp Anticarium Desktop un Anticarium Pi. Dotā bibliotēka ir populārākais C++ JSON rīks.

2.8.5. cedricve/raspicam [11]

Tā kā Anticarium Camera ir C++ programma, ērtākais risinājums ir izmantot C++ bibliotēku Raspberry pi kameras vadīšanai. Šim mērķim tiek izmantota dotā raspicam bibliotēka.

2.8.6. gabime/spdlog [12]

Vieglai programmas klūdu atrašanai un darbības pētīšanai ir nepieciešams ieviest log failu ierakstus, kas ļauj programmai failā ierakstīt tās svarīgākos notikumus. Šim mērķim tiek izmantota spdlog C++ bibliotēka, kas ir ātrākā log failu rakstīšanas C++ bibliotēka, kas minimāli ietekmē C++ programmas darbību. Šī bibliotēka tiek izmantota Anticarium Desktop, Anticarium Pi, Anticarium Camera.

2.9. Pakotņu menedžeri (Package managers)

2.9.1. VCPKG [13]

VCPKG ir bezmaksas C/C++ cross-platform pakešu menedžeris, ko izstrādāja Windows. Izmantojot VCPKG, var ērti lejuplādēt C/C++ bibliotēkas un, izmantojot CMake, tās savietot ar savu projektu. Tieki izmantots Anticarium Desktop.

2.9.2. Hunter [14]

Hunter ir bezmaksas C/C++ cross-platform pakešu menedžeris. Tā pat kā ar VCPKG, izmantojot Hunter var ērti lejuplādēt C/C++ bibliotēkas un izmantojot CMake tās savietot ar savu projektu. Hunter tiek izmantots uz galvenās terārija vadības ierīces – Raspberry Pi, jo tam nepietiek procesēšanas jaudas veiksmīgai VCPKG kompilācijai. Tieki izmantots Anticarium Camera un Anticarium Pi.

2.9.3. Pip3 [15]

Pip3 ir pakešu menedžeris priekš Python programmēšanas valodas. Tieka izmantots kā standarta Python pakešu menedžeris.

2.10. Papildus rīki

2.10.1. clang-format [16]

Clang-format ir C/C++ un citu valodu pirmkoda formatēšanas rīks. Izmantojot clang-format, ir iespējams vienādi noformatēt visu sarakstīto kodu, kas atvieglo koda izskatīšanu, uzturēšanu, lasīšanu. Tieka izmantots visās Anticarium C++ daļās.

2.10.2. Git [17]

Git ir plaši izmantota versionēšanas sistēma. Git tiek izmantots koda rediģēšanai, izmaiņu veikšanai, koda vēstures skatīšanai. Salīdzinot ar citām versionēšanas sistēmām, kā, piemēram, Subversion, Git ļauj saglabāt koda izmaiņas lokāli pat mainot zarus.

2.11. Terārija vadības ierīces

2.11.1. Raspberry Pi

Raspberry Pi ir miniatūrs dators, kas tika izveidots uz nelielas, dažādu nozaru izstrādātājiem draudzīgas iespiedplates. Raspberry Pi tiek izmantots kā galvenā Anticarium terārija vadības ierīce. Tas tika izvēlēts, jo tas atbalsta video uzņemšu un apstrādi, tīkla komunikāciju, Linux operētājsistēmu, kā arī tam ir arī izejas kājiņas, kas ļauj vadīt ārēji pievienotas elektroniskas komponentes.

Atbalsta I²C protokolu, kas ir nepieciešams, lai komunicētu ar Input IC un Output IC.

2.11.2. ATtiny85 [18]

ATtiny85 ir 8 kājiņu, 8 bitu AVR¹ mikroshēma ar 8KB Flash atmiņas ietilpību. Šī mikroshēma tika izvēlēta, lai veiktu lasījumus no dažādiem sensoriem. Anticarium projektā tiek izmantoti divi sensori.

Zemes mitruma sensors ļauj viegli veikt lasījumus, jo tas izvada zemes mitrumu kā analogu informāciju, spriegumu.

¹ AVR ir 8 bitu mikrokontrolieris, kas piedero pie RISC (Reduced Instruction Set Computer) procesoru veida [46]

Gaisa temperatūras un gaisa mitruma noteikšanai tiek izmantots viens kopīgs sensors, kas datus izvada digitālā veidā. Datu nolasīšanai ir nepieciešama atsevišķa DHT sensoru lasīšanas bibliotēka. Visērtāk lietojamā šāda veida bibliotēka ir “Adafruit DHT sensor library [19]”, taču tās izmēra dēļ ir nepieciešams lielāks atmiņas daudzums. Tādēļ tika izvēlēta šī mikroshēma ar lielāko atmiņas daudzumu starp AttinyX5² sērijas mikroshēmām.

Dotā mikroshēma atbalsta I²C datu apmaiņas protokolu, kas ir nepieciešams, lai komunicētu ar citām ierīcēm izmantojot Anticarium PCB. Astoņu kājiņu daudzums nodrošina, ka mikroshēma neaizņem lieku vietu, jo tiek aktīvi izmantotas tikai sešas kājiņas. Divas kājiņas I²C komunikācijai, divas kājiņas barošanai, divas kājiņas sensoru lasījumiem. Kopā tiek lietotas sešas no astoņām kājiņām.

2.11.3. ATtiny24 [20]

ATtiny24 ir 14 kājiņu, 8 bitu AVR mikroshēma ar 2KB Flash atmiņas ietilpību. Šī mikroshēma tika izvēlēta, lai vadītu pie Anticarium PCB pieslēgtās elektroniskās komponentes.

Dotā mikroshēma atbalsta I²C datu apmaiņas protokolu, kas ir nepieciešams, lai komunicētu ar citām ierīcēm, izmantojot Anticarium PCB. Atšķirībā no iepriekš minētā ATtiny85, šai mikroshēmai nav nepieciešama papildus bibliotēka izņemot Arduino, lai veiksmīgi novadītu visas pie šīs mikroshēmas pieslēgtās elektroniskās komponentes, tāpēc var izmantot 2KB modeli. Taču komponenšu daudzuma dēļ, ir nepieciešams 14 kājiņu modelis atšķirībā no ATtiny85. Divas kājiņas I²C komunikācijai, divas kājiņas barošanai, divas kājiņas PWM lietošanai, trīs kājiņas I/O lietošanai. Kopā tiek lietotas deviņas no četrpadsmit kājiņām.

2.12. Datu pārraides protokoli

2.12.1. I²C [21]

“Philips Semiconductors (pašlaik NXP Semiconductors) ir izstrādājuši vienkāršu abpusēju divu kanālu komunikācijas protokolu efektīvai IC vadībai, ko nosauca par I²C. Komunikācijai ir nepieciešami tikai divi kanāli: serial data line (SDA) un serial clock line (SCL) [22]”.

² X nozīmē modeļa numuru, kas simbolizē atmiņas daudzumu dotajā mikroshēmā. ATtinyX5 sērija sastāv no ATtiny25, ATtiny45, ATtiny85 ar 2KB, 4KB, 8KB Flash atmiņas ietilpību. Tas pats attiecas arī uz ATtinyX4 mikroshēmu sēriju

Dotais protokols tiek izmantots, jo notiek komunikācija starp Raspberry Pi un Input IC, Output IC. I²C protokola priekšrocība ir tāda, ka visas ierīces, kas savā starpā komunicē ar šo protokolu, var pieslēgt paralēli pie diviem vadiem.

2.12.2. HTTP

HTTP ir protokols, ko visbiežāk izmanto komunikācijā starp mājaslapām un serveriem. Šo protokolu izmanto Anticarium Desktop un Anticarium Pi komunicējot ar Anticarium WEB.

Tika izvēlēts, jo šis ir standarta protokols datu apmaiņai starp klientu un serveri.

2.12.3. UDP

UDP ir datu apmaiņas protokols, kas negarantē par visu nosūtīto datu paku veiksmīgu piegādāšanu. Šādu protokolu ir izdevīgi lietot gadījumos, kad ir jāpārraida lielu daudzumu datu, nemot vērā, ka dati var tikt arī zaudēti pārraides laikā.

Šis protokols tiek lietots starp Anticarium Camera un Anticarium Desktop tiešsaistes video pārraidīšanai. Pārraidot tiešsaistes video, nav būtiski, ja dažas attēla daļas uz mirkli var būt nepilnvērtīgas.

2.13. Galvenās elektroniskās komponentes

2.13.1. DHT22 [23]

DHT22 ir gaisa mitruma un gaisa temperatūras sensors. Tā priekšrocības ir tā nelielā cena, precizitāte un ērts interfeiss (komunikācijai izmanto tikai vienu vadu, kanālu). Precizitāte: $<\pm 0.5^{\circ}\text{C}$ un $\pm 2\%$ [24].

2.13.2. Zemes mitruma sensors [25]

Zemes mitruma sensors tiek izmantots, lai mērītu zemes mitrumu procentos. Tika izvēlēts tā nelielās cenas un analoga rezultāta izvadīšanas dēļ.

2.13.3. LED diožu lenta

Anticarium tiek izmantota 12V IP65 diožu lenta. Krāsa – neitrāli balta.

2.13.4. Ventilators

Anticarium tiek izmantots 12V ventilators, lai veiktu kontrolētu gaisa apmaiņu.

2.13.5. Terārija lampas spuldze

Tika izvēlēta atbilstoši aprēķinam, ko dod terārija lampas spuldzes jaudas kalkulators [26]. Terārijam, kas tiek izmantots šī projekta ietvaros (izmērs 910x460x320mm), ir nepieciešama vismaz 65W spuldze, lai spētu pacelt gaisa temperatūru terārijā par 11°C. Pieņemot, ka terārijs atrodas istabas temperatūrā (~22°C), šāda spuldze spēj pacelt gaisa temperatūru terārijā līdz 33°C, kas ir vairāk par prasībās noteikto minimumu. Kā spuldze tika izvēlēta ReptiPlanet Daylight Neodymium 75W [27].

2.13.6. Vakuumā ūdens sūknis

Ūdens padošanai tiek izmantots 12V vakuma ūdens sūknis. Padotais ūdens daudzums <65ml/min [28].

3.8.9. Raspberry Pi kamera

Raspberry Pi piedāvā portu specifiski savām video kamerām. Šim projektam tika izvēlēta Raspberry Pi Camera V2. Kamera piedāvā 8MP 1080P attēlu kvalitāti.

3.9. WEB Serveris

3.9.1. Apache2 [29]

Terārija serveris atrodas uz Raspberry Pi datora Raspbian operētājsistēmas, kas ir bāzēta uz Linux operētājsistēmas. Apache2 ir populārākā HTTP serveru programma Linux serveriem, Apache2 lieto vairāk nekā 30% visu Linux serveru pasaulei. Apache2 izmanto Anticarium WEB.

3.9.1. SQLite [30]

SQLite ir pasaulei populārākais datubāzes dzinis [31]. Tas uzglabā visus datubāzes datus vienā failā un ir izcils risinājums nelielām sistēmām. Tā kā Anticarium izmanto vienu serveri priekš katras lietotāja, kas atrodas uz paša terārija vadības ierīces (Raspberry Pi), efektīvākais risinājums ir izmantot šo nelielo datubāzes dzini lietotāja datu glabāšanai.

3.10. Rasēšana

3.10.1. AutoCAD [32]

AutoCAD ir rasēšanas programma, ar kuru ir iespējams rāsēt gan 2D, gan 3D modeļus. Šī ir programma, ko izmanto arhitektūrā, inženierzinātnēs, grafiskajā dizainā. Izmantojot AutoCAD, ir iespējams precīzi uzrāsēt nepieciešamās detļas.

AutoCAD tiek lietots šī projekta ietvaros, lai izveidotu 3D modeļu rasējumus 3D printēšnai.

Tiek arī lietots šī projekta ietvaros 2D terārija vāka rasējumam, kura lāzērgriešanu veica SIA Posmaster [33].

3.10.2. Cura

Cura ir programma, kas pārstrādā 3D failus (STL, OBJ, 3MF) 3D printerim saprotamā formātā. Papildus 3D failu pārstrādei ir iespējams iestatīt dažāda veida 3D printēšanas parametrus, lai tiktu izprintēta maksimāli kvalitatīva detaļa. Uzrasēto modeļu printēšanas sagatavošanai tiek izmantots AutoCAD.

3.10.3. EasyEDA [34]

EasyEDA ir elektronisku shēmu un iespiedplašu rasēšanas rīks, ko izstrādāja JLCPCB COM [35]. Dotais rīks ir pieejams internetā bezmaksas. Izmantojot EasyEDA ir iespējams izveidot elektronisko shēmu un iespiedplašu rasējumus.

4. PROGRAMMATŪRAS PRODUKTA MODELĒŠANA UN PROJEKTĒŠANA

Anticarium sastāv no vairākām daļām. Katra no tām tika veidota neatkarīgi no citām, šādi saglabājot modularitāti. Modularitāte ir svarīga, jo ļauj veikt izmaiņas katrā no daļām, neietekmējot citas projekta daļas.

Anticarium izstrādes laikā tika izveidota sekojoša projekta struktūra, kas redzama pielikumā 5. Svarīgi piebilst, ka viss, kas ir ārpus Desktop PC jeb datora, uz kura darbojas Anticarium Desktop, atrodas paša terārija ietvaros.

4.1. Anticarium elementu struktūras apraksts

Pirms Anticarium struktūras apskatīšanas ir nepieciešams iepazīties ar katra Anticarium elementa struktūru un darbību.

4.1.1. Anticarium Desktop

Anticarium Desktop ir lietotāja interfeiss ar terāriju. Izmantojot Anticarium Desktop, ir iespējams iestatīt nepieciešamos terārija parametrus, mainīt režīmus, apskatīties video un datus, kas tiek atsūtīti no terārija. Šī ir galvenā lietotne, kas ir paredzēta lietotājam.

Anticarium Desktop funkcionālās dekompozīcijas diagramma ir redzama 6. pielikumā. Programma sastāv no trīs daļām. Grafiskais interfeiss, HTTP komunikācijas daļa, video apstrādes un atrādīšanas daļa. Video atrādīšanas daļa ir lielākoties neatkarīga, toties grafiskais interfeiss un HTTP komunikācijas daļa savā starpā ir cieši saistītas, jo lietotājam, veicot darbības grafiskajā interfeisā, bieži vien notiek datu apmaiņa ar serveri.

4.1.2. Anticarium WEB

Anticarium WEB ir HTTP serveris, kas ir atbildīgs par datu apmaiņu starp terāriju un galveno lietotni. Anticarium WEB uzglabā visus datus, kas tam tiek nosūtīti, un tālāk tos nosūta pēc pieprasījuma. Anticarium WEB arī atbild par režīmu uzglabāšanu un redīģēšanu datubāzē.

Anticarium WEB funkcionālās dekompozīcijas diagramma ir redzama 7. pielikumā. Kā redzams diagrammā, ir izveidota nepieciešamā funkcionalitāte, lai veiktu manipulācijas ar saglabātajiem elementiem. Ir arī redzams, ka katra HTTP pieprasījuma veidam ir siks konkrēts uzdevums, kas ir jāveic.

4.1.3. *Anticarium Pi*

Anticarium Pi ir programma, kas darbojas uz Raspberry Pi datora un atbild par lietotāja ievadīto vērtību apstrādi un padošanu tālāk fiziskai izvadei. Šī programma atbild arī par galveno terārija funkcionalitāti – terārija gaisa temperatūras un zemes mitruma uzturēšanu, ko pieprasa lietotājs.

Anticarium Pi funkcionālās dekompozīcijas diagramma ir apskatāma 8. pielikumā. Kā redzams diagrammā, šī programma ir sadalīta trīs daļās. Divas komunikācijas daļas, kas atbild par diviem dažādu veidu komunikācijas protokoliem, un viena daļa, kas veic aprēķinus un apstrādā datus, kas ienāk no lietotāja un terārija.

4.1.4. *Anticarium Camera*

Anticarium Camera ir programma, kas tāpat kā Anticarium Pi darbojas uz Raspberry Pi datora. Šī programma ielasa datus no kameras, kas ir pievienota pie Raspberry Pi, un pēc pieprasījuma nosūta attēlus uz Anticarium Desktop lietotni.

Šī programma ir pilnīgi nodalīta no Anticarium Pi, neskatoties uz to, ka abas programmas darbojas uz viena un tā paša Raspberry Pi, un abas programmas ir izveidotas C++ programmēšanas valodā. No sistēmas arhitektūras viedokļa šāda pieeja padara programmas vieglāk uzturamas un novērš nevajadzīga koda izveidi, kas būtu nepieciešams, lai šīs abas programmas sasaistītu vienā. Divas atsevišķas programmas garantē arī to, ka, ja viena vai otra programma negaidīti beidz savu darbību, tad joprojām ir otra, kas saglabā vismaz daļu funkcionalitātes iekš Anticarium Desktop. Ja Anticarium Camera un Anticarium Pi tikt apvienots vienā programmā, tad programmai, negaidīti apstājoties, tiek zaudēta visa terārija funkcionalitāte.

Anticarium Camera funkcionālās dekompozīcijas diagramma ir redzama 9. pielikumā. Kā redzams, programmas struktūra nav sarežģīta un katru no redzamajām daļām dara kādu vienu specifisku uzdevumu. Galvenais mērķis ir iegūt attēlu, to apstradāt, sadalīt mazākās daļās un nosūtīt.

4.1.5. *Anticarium IC*

Output IC vada visas četras terārija izejas - gaisa padevi, gaismas padevi, ūdens un siltuma padevi. Vadīšana notiek izmantojot PWM signālus vai arī ieslēdzot/izslēdzot kādu konkrētu izeju.

Input IC ielasa datus no abiem sensoriem – mitruma sensora un DHT22. Datu nolasīšana no mitruma sensora notiek, izmantojot analogu ievadi, kas nozīmē, ka atkarībā no mitruma mainīsies sensora izvadītais sprieguma līmenis. DHT22 datu nolasīšana notiek digitāli, izmantojot tam paredzētu protokolu.

Output IC funkcionālās dekompozīcijas diagramma ir apskatāma 10. pielikumā.

Input IC funkcionālās dekompozīcijas diagramma ir apskatāma 11. pielikumā.

4.1.6. *Anticarium PCB*

Anticarium izstrādes laikā radās nepieciešamība pēc paša veidotas elektroniskās iespiedplates jeb PCB³. Nemot vērā nelielo vietas daudzumu, kas tika ieplānots, tapa skaidrs, ka lodēt lielās THT⁴ elektroniskās komponentes nav efektīvs risinājums, tāpēc nācās pievērsties mazajām SMD⁵ elektroniskajām komponentēm, kas tiktu lodētas uz PCB.

Anticarium PCB uzdevums ir atbildēt par visu elektronisko komponenšu darbību un veiksmīgu I²C komunikāciju starp Raspberry Pi un Anticarium IC. Anticarium PCB ir iebūvēta arī analoga dzesēšanas sistēma, kas atbild par to, lai visas elektroniskās komponentes, kas atrodas Anticarium korpusā, nepārkarstu, mainot dzesēšanas stiprumu attiecībā no temperatūras pašā korpusā. Anticarium PCB rasejumus, izskatu var apskatīties pielikumos no 12. līdz 15.

4.1.7. *Terārijs*

Kā galvenais šī projekta galaproducts ir terārijs. Tika nolemts, ka ātrākais un kvalitatīvākais veids, kā izveidot terāriju, ir nopirkst jau gatavu terāriju un izveidot tam vāku, kas sevī ietvertu visu nepieciešamo terārija funkcionalitāti. Šāda pieeja ļauj pašam neveidot terārija korpusu no jauna, kā arī visas funkcionalitātes iestrādāšana vākā ļauj terāriju padarīt modulāru, un nepieciešamības brīdī vāku ir iespējams noņemt nost un veikt tā apkopi. Kā pats terārijs tika izvēlēts ReptiPlanet terārijs [42]. Tas, kopā ar vāku un elementu skaidrojumiem, ir redzams 16. pielikumā.

Terārija vākā kopumā tika iebūvētas četras lūkas. Divas mazās lūkas, kas kalpo tikai kā lūkas, un divas lielākas lūkas - katru ar savu īpašību. Vienā no lielajām lūkām tika iebūvēts ventilators (17. pielikums), bet otrajā lielajā lūkā tika iebūvēts tīkls, kas ļauj no augšas sildīt sildlampai (18. pielikums). Vāka apakšā ir iebūvētas caurules ūdens padevei, kā arī LED

³ Printed Circuit Board

⁴ Through Hole Technology

⁵ Surface Mount Device

apgaismojums (19. pielikums). Ūdens padeve notiek no ūdens tvertnes. Ūdens tvertne ir atverama un aizverama ar vāku. No tvertnes var izņemt ieliktni, kur ieliet iekšā ūdeni un to novietot atpakaļ kopā ar cauruli (20. pielikums). Terārija sānā ir stikla siena, kurā ir izgriezts apalš caurums. Šajā vietā tika izvietota terārija kamera (21. pielikums), kas filmē terāriju no iekšpuses. Terārija kamera atrodas korpusā, lai to pasargātu no mitruma un mehānikiem bojājumiem. Korpuss ir sadalāms uz pusēm un kameru vajadzības gadījumā var ērti noskrūvēt, tādējādi terārija vāks nav piesaistīts pie paša terārija.

Terārija vadība notiek no galvenā vadības bloka. Vadības bloks ir nosegts ar vāku un atrodas pacelts uz kājiņām no terārija vāka apakšas. Šāds dizains tika izveidots, lai pasargātu elektroniku no iespējamas saskares ar ūdeni, kā arī no apakšas tika atstāta vieta gaisa padevei, dzesēšanai. Terārija vadības bloks no iekšas ir apskatāms 22. pielikumā.

4.2. Sistēmas struktūras apraksts

Apskatot Anticarium struktūru (5. pielikums) var noprast, ka kopumā sistēmas darbība nav sarežģīta. Lielākoties sistēmai ir nepieciešams saņemt datus no lietotāja un padot tos tālāk no vienas daļas uz citu, līdz dati, ko lietotājs ievadīja datorā, pārvērsas par reālu, fizisku izeju. Turpmākajās apakšnodaļās tiks izklāstīts katras Anticarium daļas uzdevums un saistība ar citām daļām. Atsevišķu daļu saistība ar citām komponentēm ir skaidri attēlotā 5. pielikumā.

4.2.1. *Anticarium Desktop* un *Anticarium WEB* saistība

Galvenā datu apmaiņa iekš Anticarium notiek starp Anticarium Desktop un Anticarium WEB. Šī ir svarīgākā datu apmaiņa, jo Anticarium Desktop ir lietotāja interfeiss ar terāriju.

Datu apmaiņa starp Anticarium Desktop un Anticarium WEB notiek izmantojot HTTP protokolu. Starp Anticarium Desktop un Anticarium WEB notiek JSON datu apmaiņa, kuru saturu nosaka `shared_types` datu struktūras (šīs datu struktūras ir sīkāk aprakstītas sadaļā 5.1). Anticarium Desktop no servera, jeb Anticarium WEB, pieprasā visa veida `shared_types` datus. Uz serveri Anticarium Desktop nosūta tikai darbības ar režīmiem un pagaidu vērtības, jeb `shared_types::Control`.

4.2.2. *Anticarium Pi* un *Anticarium WEB* saistība

Anticarium Pi konstanti pieprasā datus no Anticarium WEB. Anticarium Pi interesē tikai viena veida dati, tie, kas ir pašlaik jāiestata terārijā. Anticarium Pi nav nozīmīgs esošais iestatītais režīms, jo visas darbības ar režīmiem ir Anticarium WEB atbildība. Šāds uzdevumu sadalījums novērš koda atkārtošanos un atvieglo sistēmas izstrādi. Anticarium Pi ir tikai

jāpieprasā serverim kādas vērtības ir pašlaik jāiestata. Pašlaik iestatāmās vērtības tiek atgrieztas `shared_types::Control` formā, jo dotais modelis satur visu nepieciešamo informāciju, kas ir jāizvada terārijā, neskatoties uz to, ka iekš Anticarium Desktop tas tiek lietots kā pagaidu vērtību struktūra.

Anticarium Pi veic ne tikai datu izvadīšanu, bet arī datu ievadīšanu. Uz serveri Anticarium Pi nosūta datus par terārijā esošajiem apstākļiem, kas tika nolasīti no sensoriem. Šie dati tiek nosūtīti `shared_types::SensorData` formā. Pēcāk šie dati, pēc pieprasījuma no Anticarium WEB, tiek nosūtīti uz Anticarium Desktop.

4.2.3. Anticarium Camera un tās saistība ar Anticarium Desktop

Anticarium Desktop ik pa laikam veic datu pieprasījumus uz Anticarium Camera izmantojot UDP protokolu. Šāda veida pieprasījumi notiek nepārtraukti ik pēc dažām sekundēm un to sauc par heartbeat savienojumu [43]. Anticarium Camera, saņemot UDP heartbeat pieprasījumu, uzsāk attēlu nosūtīšanu uz Anticarium Desktop.

4.2.4. Input IC un Output IC

Katra no mikroshēmām pilda tikai vienu savu uzdevumu – datu ievadi vai izvadi. Šāda modulāra pieeja padara sistēmu vienkāršāku, jo ir skaidri nodalītas katras daļas loma.

Raspberry Pi komunikācijā ar abām mikroshēmām tiek lietots I²C protokols. Abas mikroshēmas vada Anticarium Pi programma, kur arī notiek visa datu apstrāde. Pašas mikroshēmas tikai izpilda tām padotās vērtības un nekādas papildus darbības neveic, vērtības pēc pieprasījuma uzreiz tiek nodotas Anticarium Pi, tādas kādas tās ir. Ir svarīgi, lai pēc iespējas lielāku darba apjomu spētu izdarīt Anticarium Pi, jo veikt mikroshēmu atklūdošanu problēmu gadījumā ir sarežģīti.

4.3. Sistēmas projektēšanas procesa apraksts

Lielākā daļa Anticarium projekta tika izveidota izmantojot C++ programmēšanas valodu. Kā tika minēts iepriekš, projekts tika sadalīts vairākās daļās, git repozitorijās. Sekojoši, C++ projektu struktūra bieži vien ir līdzīga vai tāda pati starp visām Anticarium C++ projekta daļām. Izņēmuma gadījumi ir aprakstīti daļās, kur tie ir sastopami.

Bieži sastopami faili, direktorijas un to skaidrojumi:

- `cmake/` (direktorija) – C++ projekto, kur tiek izmantota CMake build sistēma, šajā direktorijā glabājas faili, kuros atrodas CMake funkcijas, makrofunkcijas;

- `include/` (direktorija) – bieži vien veidojot C++ bibliotēkas, direktorijās ar šādu nosaukumu tiek likti iekšā visi C++ header faili;
- `src/` (direktorija) – C++ bibliotēku gadījumā šajā direktorijā tiek likti visi source faili. Ja netiek veidota C++ bibliotēka, tad šajā direktorijā atrodas viss projekta pirmkods;
- `tests/` (direktorija) – šajā direktorijā atrodas unit testing⁶ C++ kods;
- `apps/` (direktorija) – glabā sevī programmas sākumpunkta(-u) failu(-us);
- `libs/` (direktorija) – glabā sevī papildus ārejo bibliotēku pirmkodu kopijas lietošanai projektā;
- `CHANGELOG.md` (fails) – šajā failā tiek veikti ieraksti par svarīgākajām izmaiņām projekta attīstības laikā;
- `README.md` (fails) – šajā failā tiek ierakstīta svarīgākā informācija git repositorijas lietotājam;
- `VERSION` (fails) – šajā failā glabājas projekta esošais versijas numurs;
- `*.in` (faili) – šos failus izmanto CMake, veicot failā nepieciešamās izmaiņas un noņemot paplašinājumu `.in`;
- `*.cmake` (faili) – faili, kuros glabājas CMake funkcijas, makrofunkcijas;
- `settings.ini.in` (fails) – projektos, kuros tiek izmantots Qt ietvars, šajā failā atrodas noklusējuma iestatījumu vērtības. CMake attiecīgi apstrādā šo failu un novieto to tajā pašā vietā, kur atrodas arī kompilētā programma.
- `TestDataDirectory.h.in` (fails) – šis fails atrodas zem `tests/` direktorijas. Šo failu apstrādā CMake un ņauj to iekļaut citos C++ pirmkoda failos. Satur makro ar pilnu ceļu (PATH) līdz projekta `tests/` direktorijai, kas ņauj ērti šajā direktorijā ievietot papildus testēšanas failus, kas nav pirmkods.
- `main.cpp` (fails) – standarta C++ sākumpunkta fails;
- `CMakeLists.txt` (fails) – galvenais CMake fails(-i), kas tiek izpildīts(-i);
- `.gitignore` (fails) – satur sevī failus, kuriem git versionēšanas sistēmai nav jāseko līdz;
- `.clang-format` (fails) – clang-format koda formatētāja fails, kas satur sevī nepieciešamos formatēšanas iestatījumus.

⁶ Unit testing, jeb vienību testēšana, ir programmas izstrādes daļa, kur mazākās, testējamās programmas daļas tiek individuāli un neatkarīgi testētas un pārbaudītas darbībā attiecībā pret sagaidāmajām vērtībām

4.3.1. CMake struktūra

Lielākā daļa Anticarium C++ komponenšu lieto CMake build sistēmu. Sekojoši, struktūras šiem projektiem tika veidotas līdzīgas, vai tādas pašas. Direktorijas augšējā daļā atrodas galvenais `CMakeLists.txt` fails, kas ir pirmais fails, kas tiek izpildīts, kad tiek palaista `cmake` komanda un iesākta programmas kompilācijas failu izveide. CMake ir izveidota pēc principa, ka tā darbojas ar augsta līmeņa logiskajiem objektiem, kur katrs objekts ir bibliotēka vai arī programma. Anticarium gadījumā visas augstākā līmeņa direktorijas, kas satur programmas pirmkodu tika izveidotas kā CMake objekti, kur vienīgie objekti-programmas atrodas zem `apps/` un `tests/` direktorijs. Pārējās direktorijas tika pārveidotas par CMake objektiem-bibliotēkām.

CMake ļauj automatizēt pirmkoda izgūšanu no git repozitorijām balstoties uz padoto versiju vai git commit id. Šāds paņēmiens tika izmantots, lai dalītos ar `shared_types` saturu starp Anticarium Desktop un Anticarium Pi. Šāda pieeja nepieļauj koda dublikāciju, ļauj nekopēt bibliotēkas pirmkodu uz visiem projektiem kas izmanto doto bibliotēku, kā arī ļauj izgūt konkrētu bibliotēkas versiju.

4.3.2. Unit testing

Visas Anticarium daļas, kas izmanto CMake build sistēmu arī izmanto Google test Unit testing ietvaru. Testu izveide notiek dažādos source, jeb `.cpp`, failos. 31. pielikumā ir apskatāms Unit Testing piemērs no Anticarium Desktop. Kā redzams, Google Unit testi tiek izveidoti izmantojot `TEST` makrofunkciju. Makrofunkcija par argumentiem nēm testa grupas nosaukumu un konkrētā testa nosaukumu. Šim seko scope kurā tiek rakstīts testēšanas pirmkods. Vērtību testēšana arī notiek ar speciālām testēšanas makrokomandām. Piemērā ir redzama makrokomanda `EXPECT_EQ()`. Šī komanda sagaida, ka abas padotās vērtības ir vienādas. Ja abas padotās vērtības nav vienādas, tad testēšanas programma izprintēs kļūdu aprakstot arī vietu no kuras tā nāk.

4.3.3. Log ierakstu veikšana

Visas Anticarium programmas, kuras tika izveidotas izmantojot CMake build sistēmu, arī veic log ierakstus. Log ieraksti ir automātiski izveidoti ieraksti, kas atzīmē ierakstīšanas laiku un kādu notikumu, kas tajā brīdī ir noticis konkrētai sistēmai. Anticarium projektā katrai programmai log faili atrodas zem `log/` direktorijs, un pati direktorija atrodas tur pat kur atrodas arī pati programma.

Log failu nosaukumi satur datumu, kad fails tika izveidots, kā arī programmas nosaukumu. Katrā failā tiek veidoti ieraksti līdz fails sasniedz 10MB izmēru, kad tas notiek, fails tiek pārsaukts ievietojot .<faila numurs> starp esošo faila nosaukumu un faila paplašinājumu.

Pastāv dažādu līmeņu log ieraksti. Log līmenis nosaka to kāda veida ieraksti tiks veikti log failā. Log līmeņus var mainīt settings.ini failā, kas atrodas tajā pašā vietā kur arī kompilētā programma, vai arī veikt attiecīgas izmaiņas settings.ini.in failā un palaist CMake, lai tas veiktu nepieciešamās izmaiņas. Pēc noklusējuma visos settings.ini.in failos ir iestatīts Log_Level=2 kā log līmenis. Sekojoši, tiek veikti kritiskie, kļūdu, brīdinošie ieraksti. Log_Level iestatot uz 3, tiks veikti arī informatīvie ieraksti, kas Anticarium sistēmā skaitās kā zemākā līmeņa ieraksti un tiek lietoti atklūdošanai.

4.3.4. *Anticarium Desktop arhitektūra*

Anticarium Desktop datu plūsmu diagramma ir apskatāma 25. pielikumā. Tā kā Anticarium Desktop ir grafiska lietotne, tās arhitektūras pamatā tika izvēlēts model-view princips. Model-view princips ir arī tas pēc kā vadās Qt ietvars. Model-view princips liek sadalīt visas grafiskās komponentes divās daļās, viena, kas veic tikai grafiskā elementa datu apstrādi, jeb model, un otra, kas veic tikai graifksā elementa grafiskos aspektus, jeb view.

Qt ietvars sevī iekļauj signal-slot mehānismu. Signal-slot mehānisms ļauj savietot speciālu funkciju izsaukumus, izmantojot emit makrokomandu, ar jebkura cita objekta funkciju, šādi ļaujot ērti sasaistīt vairākus objektus un veikt darbības radot signālus no dažādām programmas daļām uz citām programmas daļām. Anticarium Desktop arhitektūras pamatā ir HTTP pieprasījumi, kuri atgriež JSON datus, kas tiek tālāk serializēti par attiecīgiem objektiem. Katrs HTTP pieprasījums atjauno lietotnes skatu dažādos veidos izmantojot attiecīgus signālus. Šādas pieejas spēks un trūkums ir tāds, ka ir nepieciešami vairāki neatkarīgi HTTP pieprasījumi, lai veiktu kādu specifisku darbību, kā piemēram lietotnes pirmās reizes ielādēšanās pieprasījumi, kas ir redzami 32. pielikumā. Šādas pieejas trūkums ir tāds, ka tas var aizņemt vairāk laika, lai sagaidītu katru pieprasījuma atbildi, taču šādas pieejas priekšrocība ir robusta koda rakstīšana, kas ļauj ērti atjaunot tikai nepieciešamās lietotnes daļas ienākot HTTP atbildei.

Galvenais Anticarium Desktop lietotnes skats ir MainWindow skats, kas darbībā ir redzams 33. pielikumā. Skata implementācija atrodas iekš Anticarium/Desktop/src/anticarium_desktop/widgets/. MainWindow

datu apstrādes daļa ir MainWindowManager, kas ir atrodama iekš Anticarium/Desktop/src/anticarium_desktop/. MainWindowManager atbild par lietotāja datu ievades nodošanu uz HTTP komunikācijas daļu, HTTP atbildes nodošanu uz MainWindow skatu, kā arī konstantu sensor_data pieprasīšanu no servera.

MainWindowManager satur arī MainWindow video skata modeli, tādējādi ļaujot attēlot video datus iekš MainWindow, un MainWindow veikt tikai kosmētiskas izmaiņas video skatam. Video saņemšana un apstrāde darbojas paralēli programmai, lai neradītu problēmas ar lietotāja saskarni. Kā galvenais video saņemšanas un apstrādes objekts kalpo VideoManager. Tas, nepārtraukti, ik pēc dažām sekundēm, nosūta heartbeat UDP paketes uz Anticarium Camera atpakaļ saņemot video, kuru tālāk padot apstrādei un izvadīšanai.

Pēc tāda paša principa kā MainWindow un MainWindowManager tika izveidots arī režīmu attēlošanas logs. Sekojoši, tika izveidots DisplayRegimes view un DisplayRegimesManager model, kur DisplayRegimes tikai veic datu attēlošanu un DisplayRegimesManager veic datu pieprasīšanu un apstrādi.

Anticarium Desktop ir sekojoši iestatījumi iekš settings.ini:

- Anticarium_Server_URL – HTTP servera domēns;
- Anticarium_UDP_URL - UDP servera IP;
- Sensor_Data_fetch_timeout – laiks milisekundēs cik bieži ir jāpieprasa sensoru datus no servera;
- Server_UDP_Port – UDP servera ports;
- Client_UDP_PORT – ports, kuru lieto Anticarium Desktop UDP datu saņemšanai;
- Image_Width – video platums. Tam ir jābūt par vienu pikseli mazākam, jo pirmais pikselis tiek izmantots rindas id noteikšanai;
- Image_Height – video augstums;
- Log_Level – log ierakstu līmenis.

4.3.5. *Anticarium WEB arhitektūra*

Anticarium WEB ir HTTP serveris, kas izmanto Apache2 kā HTTP servera dzini. Savā sirdī, Anticarium funkcionalitāte šajā serverī tika izveidota izmantojot Python skriptus. Šāda pieeja ļāva ātrā laika sprīdī izveidot nepieciešamo backend dotajam projektam. Galvenais skripts ir anticarium_web.py, kas satur visu pieprasījumu adreses, datu apstrādi. Anticarium WEB datu plūsmu diagramma ir skatāma

26. pielikumā. Kā redzams diagrammā, Anticarium WEB darbība nav vienota, bet gan sadrumstalota. Šī ir Flask ietvara īpatnība, ka katrs pieprasījums izsauc atsevišķu funkciju.

Tā kā serveris atrodas uz paša terārija, tas var tikt izslēgts jebkurā brīdī. Ir svarīgi saglabāt visus svarīgākos datus par esošo stāvokli, lai nākamo reizi, kad terārijs tiktu pieslēgts pie strāvas, lietotājs varētu turpināt lietot Anticarium Desktop no tās vietas pie kuras pēdejo reizi bija palicis. Tāpēc, Anticarium WEB glabā datus trīs dažādos veidos:

- datubāzē;
- mainīgajos;
- failos.

Datubāzē tiek uzglabāti tikai režīmi un to pamatinformācija. Failos tiek uzglabāti tikai svarīgākie dati, kas nav pieejami datubāzē un lietotājam ir nepieciešami uzreiz atverot Anticarium Desktop:

- control;
- sensor_data;
- regime_id.

control tiek glabāts failā, jo tas papildus sevī glabā apgaismojuma spilgtumu un ventilatora griešanās ātruma vērtības, kuras nepieciešams attēlot iekš Anticarium Desktop. Šīs vērtības atrodas tikai iekš control datu struktūras. sensor_data tiek saglabāts failā, jo tas ļauj attēlot pēdējos sensoru lasījumus. regime_id tiek glabāts failā, jo balstoties pēc tā vērtības var noskaidrot pēdējo lietotāja iestatīto režīmu ko attēlot lietotnē un iestatīt pašā Anticarium WEB. Šie faili tiek atjaunināti katru reizi, kad atbilstošās datu struktūras vērtība tiek mainīta. Palaižot anticarium_web.py doto failu saturs tiek ielasīts lokālajos mainīgajos, ātrākai datu piekļuvei un apstrādei.

Visām datu struktūrām ir arī savi lokālie mainīgie, kas servera darbības laikā nosaka esošo terārija stāvokli.

4.3.6. *Anticarium Pi un Anticarium Camera kopējā arhitektūra*

Anticarium_Pi git repozitorija satur gan Anticarium Pi, gan Anticarium Camera projektus. Šāda pieeja vienkāršo projekta infrastruktūru samazinot koda dublikāciju. Sekojoši, src/ direktorija satur divas apakšdirektorijas un vienu papildus, kopējo direktoriju kopīgiem moduļiem:

- anticarium_camera/;

- `anticarium_pi/`;
- `config/`.

Katra no dotajām direktorijām satur attiecīgā projekta pirmkodu (izņemot `config/`, kur glabājas kods, kas tiek izmantots abos projektos). Šāda pati pieeja tika izmantota arī `apps/` direktorijai, kas satur divas sekojošas apakšdirektorijas:

- `server_app/`;
- `camera_app/`.

Šie abi projekti arī satur vienu kopīgu unit test programmu, sekojoši, `tests/` direktorija netiek dalīta sīkāk un testi tiek palaisti abiem projektiem vienlaicīgi.

4.3.7. *Anticarium Camera arhitektūra*

Anticarium Camera datu plūsmu diagramma ir apskatāma 27. pielikumā. Anticarium Camera izmanto Qt ietvaru, bet pati programma nepiedāvā grafisko vidi. Anticarium Camera darbojas kā CLI⁷ programma. Grafiskajai videi nepastāvot nav iespējams realizēt model-view programmas arhitektūru, tāpēc tika pieņemts lēmums, ka šai programmai pastāvēs hierarhiskā struktūra, kur kā galvenais objekts ir menedžeris, kas apmainīs datus starp pārejiem objektiem, tādējādi vadot visu programmu. Galvenais objekts Anticarium Camera programmā ir `StreamManager` klases instance, kuras implementācija ir atrodama `Anticarium_Pi/src/anticarium_camera/anticarium_camera/StreamManager.cpp`. Kā redzams datu plūsmas diagrammā, šīs klases instance gaida līdz tiek signalizēts par ienākušu UDP heartbeat paketi. Brīdī, kad tas notiek, tiek restartēts attēlu uzņemšanas taimeris un uz kādu laiku tiek dota atļauja nosūtīt datus tam kas tos ir pieprasījis. Ja nākamais UDP heartbeat signāls pienāk ātrāk nekā taimerim ir iztečējis laiks, tad taimeris tiek restartēts un UDP pakešu nosūtīšana turpinās.

Neskatoties uz to, ka dotā programma gaida uz UDP heartbeat paketi, attēlu uzņemšana notiek nepārtraukti, jo ir nepieciešams laiks lai kamera sāktu savu darbu.

UDP datu apmaiņai servera pusē ir nepieciešama portu pāradresācija, kas nodotu UDP datus no rūtera uz Raspberry Pi. Portu pāradresācija ir nepieciešama tikai servera pusē, jo klienta pusē rūteris redzot atpakaļ atnākušu UDP paketi, uz to pašu portu no kura tika sūtīts

⁷ CLI (Command Line Interface), čaulas interfeiss.

UDP heartbeat, veic automātisku paketes tālāku nodošanu uz attiecīgo ierīci. Šo procesu rūterī sauc par “UDP hole punching”.

Pirms nosūtīšanas attēls netiek nekādā veidā saspiests, lai nezaudētu kvalitāti. Taču, ātruma palielināšanai, attēla nosūtīšana notiek vairākos paralēlos procesos, sadalot attēlu vairākās daļās un katram procesam iedodot konkrētu attēla daļu kuru nosūtīt.

Anticarium Camera lieto raspicam [11] bibliotēku, kuru ir nepieciešams nokompilēt un uzinstalēt uz Raspberry Pi.

`settings.ini.in` fails tiek dalīts starp Anticarium Camera un Anticarium Pi projektiem. Visi liekie parametri, kuri netiek lietoti vienā vai otrā projektā paliek failā arī pēc CMake apstrādes. Saraksts ar parametriem un skaidrojumiem ko lieto Anticarium Camera:

- `Anticarium_UDP_Port` – UDP ports uz kura gaidīt ienākošo heartbeat UDP paketi;
- `FPS` – maksimālais attēlu skaits sekundē ko uzņemt;
- `Log_Level` – log ierakstu līmenis;
- `UDP_Threads_Amount` – paralēlo procesu skaits, kas nosūta attēlu.

4.3.8. *Anticarium Pi arhitektūra*

Anticarium Pi datu plūsmu diagramma ir apskatāma 2827. pielikumā. Tā pat kā Anticarium Camera, Anticarium Pi nepiedāvā grafisku lietotni, bet tikai CLI. Tika izmantota tāda pati programmas arhitektūra – hierarhiskā, kur kā galvenais objekts ir `AnticariumManager` klases instance, kuras implementācija ir atrodama `Anticarium_Pi/src/anticarium_pi/anticarium_pi/AnticariumManager.cpp`.

HTTP datu apmaiņas modulis tika nokopēts no Anticarium Desktop izņemot liekās pieprasījuma apstrādes un atstājot tikai `control` pieprasīšanu un `sensor_data` nosūtīšanu. HTTP datu apmaiņas modulis ir sasaistīts ar `AnticariumManager`.

Galvenais Anticarium Pi uzdevums ir uzturēt lietotāja iestatītos apstāklus terārijā. Tam tiek izmantots `WeatherManager` klases instance, kuras implementācija ir atrodama `Anticarium_Pi/src/anticarium_pi/anticarium_pi/WeatherManager.cpp`. Šis objekts atbild par I²C datu apmaiņu ar mikroshēmām, un mikroshēmām iestatāmo vērtbu apstrādi. Izvadāmo vērtību aprēķināšana gaisa temperatūras un zemes mitruma uzturēšanai notiek izmantojot ārējo PID bibliotēku, kas atrodas iekš `libs/pid`. No šīs bibliotēkas repozitorijas [9] tika izņemti visi liekie faili un atstāta tikai PID implementācija, kā

arī izveidots CMakeLists.txt fails, šīs bibliotēkas iekļaušanai projektā. Atbilstoši datu plūsmu diagrammai, uzturamo vērtību reķināšana, jeb PID algoritms, izmantojot esošo apstākļu terārijā vērtības, kā arī vērtības kuras lietotājs ir pašlaik iestatījis, veic aprēķinus, lai attiecīgi darbinātu elektroniskās komponentes.

Anticarium Pi dala vienu un to pašu settings.ini.in failu. Tā izmantotie iestatījumi un to skaidrojumi:

- Anticarium_Server_URL – HTTP servera domēns;
- Sensor_Data_send_timeout – laika intervāls cik bieži nosūtīt sensor_data vērtības uz serveri;
- Control_Data_fetch_timeout – laika intervāls cik bieži pieprasīt control vērtības no servera;
- PID_sample_timeout – laika intervāls cik bieži pārrēķināt iestatāmās terārija vērtības izmantojot PID algoritmu;
- I2C_fetch_timeout – laika intervāls cik bieži pieprasīt datus no Input IC;
- Log_Level – log ierakstu līmenis.

4.3.9. *Input IC un Output IC arhitektūra*

Input IC datu plūsmu diagramma ir redzama 29. pielikumā. Output IC datu plūsmu diagramma ir redzama 30. pielikumā. Neskatoties uz to, ka Raspberry Pi ir vairāk kā 40 I/O kājiņas, šajā projektā, terārija elektronisko komponenšu vadībai, tiek lietotas atsevišķas mikroshēmas. Par iemeslu kalpo tas, ka Raspberry Pi ir pieejama tikai viena kājiņa, kas atbalsta PWM⁸ funkciju. Ja šāda funkcija ir iebūvēta, tad tā mikroshēmā var notikt automātiski un neprasīt nekādu skaitļošanas jaudu, taču, bieži vien, iebūvēto PWM izeju ir neliels daudzums, ja ir nepieciešamas papildus PWM izejas, tad tās ir jāveido programmatoriski, konstanti ieslēdzot un izslēdzot elektrības padevi uz kājiņu. Šāds risinājums prasa lieki tērēt skaitļošanas jaudu. Anticarium ir nepieciešamas divas PWM izejas – LED apgaismojumam un ventilatora vadībai. Raspberry Pi ir tikai viena šāda izeja. Lai lieki nenoslogotu Raspberry Pi ar papildus, programmatoriski izveidotu PWM risinājumu, tika pieņemts lēmums visu elektronikas vadību uzticēt atsevišķiem mikrokontrolieriem.

Kā papildus priekšrocība Input IC un Output IC mikrokontrolieriem salīdzinājumā ar Raspberry Pi ir tas, ka tie darbojas 5V diapazonā, kas ļauj vadīt citas ierīces izmantojot mazāku

⁸ PWM (Pulse Width Modulation) ir elektriskā signāla modulēšanas tehnika, kas rada maināmas amplitūdas taisnstūrveida signālu

komponenšu daudzumu, jo šajā projektā tiek izmantotas arī detaļas, kas darbojas tikai 5V diapazonā un veidot pārveidošanu no 5V uz Raspberry Pi 3.3V, vai otrādi, būtu sarežģītāk.

Šī projekta ietvaros tika izmēģināta jauna pieeja elektronisko komponenšu vadībai. Vienas, lielas, kopīgas mikroshēmas vietā, terārija elektronisko komponenšu vadībai, tiek izmantotas divas mazākas mikroshēmas – Input IC un Output IC. Tā pat kā Anticarium Camera un Anticarium Pi sadalīšanā, arī šoreiz šādai pieejai ir priekšrocības no sistēmas arhitektūras puses. Abām mikroshēmām ir vienkāršots kods, jo nav nepieciešamības apvienot abu daļu kodus vienā daļā, kā arī, ja rodas problēmas ar vienu mikroshēmu, tad ir lielāka iespēja, ka otra darbosies un vismaz daļa terārija turpinās funkcionēt.

Input IC un Output IC projekti neizmanto CMake, bet izmanto PlatformIO sistēmu. Kā arī neskatoties uz to, ka abi projekti neveido bibliotēkas, bet gan programmas, `include`/ direktorijas satur visus header failus un `src`/ direktorijas satur visus source failus.

Input IC pirmkods ir atrodams `Anticarium_IC/src/attiny_input/`.

Output IC pirmkods ir atrodams `Anticarium_IC/src/attiny_output/`.

Kā tika minēts iepriekš, mikroshēmām ir nepieciešams izpildīt tikai absolūtu minimumu un ļaut visai pārējai datu apstrādei darboties jaudīgakā vidē, šoreiz uz Raspberry Pi. Šāda pieeja atvieglo izstrādes un atklūdošanas procesus, kā arī ļauj efektīvāk izmantot mikroshēmu jau tā mazos resursus.

Atbilstoši diagrammai, kas redzama 29. pielikumā, Input IC konstanti veic mērījumus un tos saglabā un, sagaidot I²C datu pieprasījumu, tos nosūta.

Atbilstoši diagrammai, kas redzama 30. pielikumā, Output IC sagaida I²C datus un uzreiz izvada tos uz atbilstošās izejas, tādējādi iedarbinot kādu no pieslēgtajām komponentēm.

4.3.10. Anticarium PCB arhitektūra

Anticarium PCB ir svarīga Anticarium projekta sastāvdaļa jo ir tā, kas ļauj darboties elektronikai. Papildus tam, ka tā nodrošina savienojumu starp svarīgākajām elektroniskajām komponentēm, tā nodrošina arī citus, ne tik ļoti pamānāmas funkcijas.

Raspberry Pi darbojas 3.3V diapazonā, bet Anticarium IC darbojas 5V diapazonā. Lai notiku veiksmīga I²C komunikācija starp šīm daļām nesabojājot Raspberry Pi ieejas ar manāmi augstāku sprieguma līmeni, ir nepieciešama I²C līmeņu pāreja, kas darbojas abos virzienos. Šī problēma un risinājums, kas tika arī implementēts Anticarium PCB, ir aprakstīts Philips

Semiconductors dokumentā [38]. Dotais risinājums ir apskatāms elektroniskajā shēmā (12. pielikums) ap komponentēm Q1 un Q9.

Lai nodrošinātu komponenšu dzesēšanu korpusā, un samazinātu trokšņa līmeni no ventilatoriem, kas tiek izmantoti dzesēšanas procesā, tika izveidota analoga dzesēšanas sistēma, kas darbina ventilatorus atkarībā no gaisa temperatūras korpusā. Analogs risinājums nozīmē, ka netiek lietota atsevišķa, saprogrammēta mikroshēma. Šāds risinājums aizņem vairāk vietas uz iespiedplates, bet komponenšu ziņā ir lētāks nekā programmējama mikroshēma. Dzesēšanas temperatūras diapazons ir maināms izmantojot platē iebūvētu potenciometru. Rasējumā šis risinājums ir redzams ap komponentēm U1 un U4.1.

Sildalampa, kas tiek lietota terārija apsildei tiek barota pa taisno no 220V barošanas. Tās darbināšana notiek ieslēdzot un atslēdzot barošanu. Šim mērķim ir paredzēts relejs, kuru darbina Output IC. Tomēr, tas nozīmē, ka šim augstajam spriegumam, kas tiek paredzēts relatīvi lielajai jaudai, ir jāaplūst caur Anticarium PCB. Tāpēc, drošības apsvērumu dēļ, 220V celiņi tika izveidot maksimāli plati, kā arī visi līdzsprieguma ceļi tika aizvākti maksimāli tālu. Risinājums ir apskatāms 13. pielikumā pie 220V brīdinājuma markējuma.

Salodējot plati un liekot to darbībā tika novērots, ka uz 5V barošanas līnijas ir taisnstūrveida signāla traucējumi. Tika novēroti divu veidu traucējumi, zemas frekvences $<10\text{Hz}$ un augstas frekvences $>1\text{kHz}$. Šādi traucējumi bija 0.8V diapazonā un nebojāja elektroniku, kā arī ierīču, kas barojās no 5V līnijas, darbība šķietami netika traucēta. Ar laiku tika novērots, ka šī tomēr ir manāma problēma Raspberry Pi darbībai, jo neskatoties uz to, ka Raspberry Pi nezaudēja barošanu, tas strādāja lēnāk un rādīja nepietiekama sprieguma paziņojumus, kā arī augstās frekvences radīja pīkstēšanu. Sīkāk izpētot šo problēmu tapa skaidrs, ka zemās frekvences traucējumi nāk no dzesēšanas ventilatoriem, jo tie tādā veidā tiek darbināti un tajā brīdī, kad tie ieslēdzas, ietekmē arī kopējo 5V barošanas līniju. Bet augstās frekvences traucējumi nāk darbinot LED lentu no 12V barošanas. 5V barošanas līnijas problēmas bija viegli identificējamas, jo ar to tiek barots gan Raspberry Pi, gan 5V ventilatori, tomēr, 12V līnijas ietekme uz 5V līniju bija neskaidra. Tika pieņemts, ka šāda parādība veidojas tadēļ, ka lielos ātrumos tiek ieslēgta un izslēgta LED lentas barošana, kas ietekmē 5V barošanas līniju uz Anticarium PCB, jo vienā no tās daļām 12V un 5V celiņi pārklājas augšējā un apakšējā slāņos. Kā risinājums šīm problēmām kļuva 5V barošanas nodalīšana Anticarium PCB un Anticarium Pi, Anticarium PCB izmantojot sprieguma stabilizatoru no 12V uz 5V, un Raspberry Pi barojot no 5V barošanas bloka. Atjauninātā elektroniskā shēma ir redzama 34. pielikumā, bet

iespiedplate ar šo labojumu netika pasūtīta. Risinājums tika realizēts izmantojot atsevišķu komponenti.

4.3.11. Terārija arhitektūra

Kā terārija vāka materiāls tika izvēlēts glancēts, 4mm biezs, melns organiskais stikls. Tika izvēlēts šāds materiāls, jo plastmasa netiek bojāta mitruma ietekmē, kā arī to ir viegli lāzerēt. Visas citas detaļas, kuras nebija iespējams nopirkt veikalā, tika printētas ar 3D printeri. Kā 3D printerā materiāls šoreiz tika izvēlēts PLA⁹ filaments.

Zemes mitruma sensors ir komponente, kurai ir jābūt tiešam kontaktam ar terārija apakšējo daļu, zemi, neskaitoties uz to, ka pati terārija vadība atrodas augšā uz vāka. Zemes mitruma sensors var tikt arī ātri sabojāts mitruma ietekmē, tāpēc, tas tika izveidots modulārs, atvienojams un viegli nomaināms pret citu.

Vāka apakšā atrodas divu veidu alumīnija profili, PVA ūdens caurulēm, kā arī LED apgaismojumam. Terārija LED lenta ir mitrumizturīga, bet lodēšanas un montēšanas procesā liela daļa tās kontaktu tika atkailināti. Visi šādi kontakti tika nolakoti ar laku vairākās kārtās ūdensdrošībai.

Tā kā Anticarium terārijs ir paredzēts visāda veida aukstasiņu dzīvniekiem, visas plašās atveres vākā tika nosegtas ar tērauda sietu, kura caurumu izmērs ir 0.4mm.

Vadības blokā ir iestrādāti divi barošanas bloki, 12V un 5V barošanai. Terārija komponenšu rasējumi un modeļi ir pieejami apskatīšanai no 35. līdz 42. pielikumam.

⁹ PLA (Polylactic acid) – Polipienskābe, plastmasa, kas parasti tiek izmantota kā 3D printēšanas izejmateriāls

5. DATU STRUKTŪRU APRAKSTS

5.1.shared_types

Anticarium ir dažāda veida dati, ko ir nepieciešams saņemt, padot, apstrādāt, atrādīt. Datu formātam ir jābūt tādam, ko varētu izmantot programmas ietvaros, gan arī spēt ērti nosūtīt izmantojot HTTP protokolu. Kā viens no labākajiem datu formātiem šādiem mērķiem ir JSON datu formāts. JSON ir datu formāts, kas ir viegli saprotams cilvēkam, kā arī viegli konvertējams uz citiem datu formātiem programmās.

Svarīgākā datu apmaiņa notiek starp Anticarium Desktop un Anticarium Pi, izmantojot Anticarium WEB starpniecību. Gan Anticarium Desktop, gan Anticarium Pi ir C++ programmas. Šie faktori mūs noved līdz secinājumam, ka labākais risinājums šādā situācijā būtu izmantot vienas un tās pašas datu struktūras abās programmās. Šim nolūkam tika izveidota Anticarium/shared_types [39] bibliotēka.

5.1.1. *shared_types* darbība

shared_types ir datu struktūru bibliotēka, kas tika izveidota specifiski Anticarium projekta nolūkiem. *shared_types* izmanto nlohmann/json C++ JSON bibliotēku, lai ielasītu datus JSON teksta formātā un ielasītās vērtības pārveidotu par C++ objektiem vai arī pārveidotu C++ objektus atpakaļ uz nlohmann::json tipa objektiem, kas spēj rast tādu pašu JSON teksta formāta rezultātu kāds bija pirms, izmantojot nlohmann::json::dump() metodi. Objektu pārveidošanu uz un no JSON formātu sauc par objektu serializāciju un deserializāciju.

Kā alternatīva serializācijai un deserializācija ir JSON atslēgas vērtības ievadīšana katru reizi pieprasot kādu vērtību no objekta, piemēram `json["key"] = 2;`. Šāds variants ir bīstams un grūti uzturams, jo izmaiņu gadījumā nāktos pētīt katru objektu kurš tiek manipulēts. Ja šādas teksta vai cita veida atslēgas nosaukumā tiek ievadīta kļūdaina vērtība, tad šādu kļūdu ir grūti atrast, kas palielina izstrādes laiku. Tāpēc, kā labākais risinājums ir šo JSON vērtību serializācija un deserializācija par pilnvērtīgiem objektiem, kuru vērtību izgūšanu jau pārbauda kompilators. Uzturot šādu praksi, JSON atslēga tiek ievadīta tikai vienu reizi un izmaiņas ir viegli ieviešamas.

Lai attēlotu *shared_types* darbību, par piemēru tiks ņemts viens no *shared_types* modeļiem, jeb datu struktūrām. Par piemēru tiks ņemts pagaidu vērtību modelis `shared_types::Control`, kura deklarācija atrodas iekš `Shared_Types/include/shared_types/Control.h`. Skatīt 52. pielikumu.

Katra modeļa klase sastāv no private uzglabājamajiem mainīgajiem. Katram mainīgajam ir sava publiskā setter un getter metode, kas attiecīgi iestata vai atgriež dotā mainīgā vērību.

53. pielikumā var redzēt funkcijas, kas tiek izmantotas objektu serializācijā un deserializācijā. Dotois kods nāk no Shared_Types/include/shared_types/ControlSerializer.hpp faila. Šīs funkcijas izmanto nlohmann::json bibliotēka, kas ļauj veikt objektu pārveidošanu “implicitly”, jeb nenorādot objekta datu tipu uz kuru notiek pārveidošana.

54. pielikumā ir iespēja redzēt objektu deserializāciju darbībā, kur sākotnēji vērtība tiek nolasīta no faila filePath uz nlohmann::json jsonIn objektu un vēlāk tiek pārveidots uz mums nepieciešamo pagaidu vērtību modeli shared_types::Control. Koda otrajā daļā ir iespēja redzēt kā dotois modelis tiek izmantots testu makrokomandās.

Ir svarīgi piebilst, ka shared_types neatbild par kļūdainu vērtību apstrādi. Tas nozīmē, ka ja ir kādas kritiskas problēmas ar objekta serializāciju vai deserializāciju, tad nlohmann::json izmet json::exception tipa objektu. Par exception apstrādi atbild pats kods, kas lieto shared_types. Tas pats ir arī attiecināms uz kļūdainām vērtībām. shared_types ļauj sevī uzglabāt jebkāda diapazona vērtības atbilstoši datu tipam. Šādā veidā šī bibliotēka pilda tikai vienu savu uzdevumu – veikt objektu serializāciju un deserializāciju.

5.1.2. shared_types JSON modeļi un to apraksti

shared_types sevī kopā iekļauj septiņus modeļus. Lai novērstu koda atkārtošanos, daži modeļi iekļauj sevī arī citus modeļus. Visi modeļi ir serializējami un deserializējami un tie lietoti komunikācijā starp Anticarium Desktop un Anticarium Pi. Turpmāk tiks apskatīti visi datu modeļi kopā ar to elementu skaidrojumiem, moduļi tiks attēloti JSON formātā. Piemēru vērtības ņemtas no Shared_Types/tests/json_files, kur katrs fails ir nosaukts attiecīgā modeļa nosaukumā ar faila paplašinājumu .json.

RegimeId

55. pielikumā ir redzams RegimeId modeļa JSON piemērs.

Lauki:

- “id” – režīma ID.

Apraksts:

`shared_types::RegimeId` modelis sastāv tikai no vienas vērtības “id”. Šis modelis tika izveidots tikai no vienas vērtības, lai to būtu iespējams serializēt un deserializēt, kā arī izmantot citos, lielākos modeļos nedublējot kodu. Šis modelis tiek izmantots, lai identificētu režīmu. Identifikācija ar skaitli ir efektīvāka nekā identificešana ar vārdu, jo aizņem mazāk vietas, kā arī ir ērti lietojama matemātiskās izteiksmēs.

RegimeValue

56. pielikumā ir redzams `RegimeValue` modeļa JSON piemērs.

Lauki:

- “temperature” – gaisa temperatūra celsija grādos;
- “moisture” – zemes mitrums procentos.

Apraksts:

`Anticarium` galvenā funkcionalitāte ir gaisa temperatūras un zemes mitruma uzturēšana. No šiem diviem parametriem sastāv arī citi modeļi. Šo divu vērtību svarīguma un neparastās atsevišķas izmantošanas vairākās vietās dēļ, tie tika pārveidoti par atsevišķu modeli.

SensorData

57. pielikumā ir redzams `SensorData` modeļa JSON piemērs.

Lauki:

- “temperature” – gaisa temperatūra celsija grādos (nāk no `RegimeValue` modeļa);
- “moisture” – zemes mitrums procentos (nāk no `RegimeValue` modeļa);
- “humidity” – gaisa mitrums procentos.

Apraksts:

`shared_types::SensorData` ir modelis, kas uztur sevī datus no visiem terārija sensoriem. Programmatoriski `shared_types::SensorData` manto elementus no `shared_types::RegimeValue`, lai novērstu koda dublikāciju.

Control

58. pielikumā ir redzams `Control` modeļa JSON piemērs.

Lauki:

- “light_percentage” – apgaismojuma spilgtums procentos;
- “wind_percentage” – ventilatora griešanās ātrums procentos;
- “regime_value” – `shared_types::RegimeValue` modelis.

Apraksts:

`shared_types::Control` ir modelis, kas sevī iekļauj esošās vērtības, jeb parametrus, kas pašlaik ir jāuzturt terārijā, ieskaitot apgaismojuma spilgtumu un gaisa padeves stiprumu. Šis modelis arī iekļauj sevī `shared_types::RegimeValue` modeli, tādējādi uzglabājot arī tekošo uzturamo gaisa temperatūru un zemes mitrumu.

Regimes

59. pielikumā ir redzams Regimes modeļa JSON piemērs.

Lauki:

- “regimes” – saraksts no režīmu nosaukumiem.

Apraksts:

Šis modelis sastāv no N daudzuma režīmu nosaukumu saraksta tādā pašā secībā kā režīmi tiek uzglabāti datubāzē. Lai katru reizi nepārsūtītu režīmus kopā ar visiem tā datiem, kad ir tikai nepieciešami režīmu nosaukumi, tiek izmantots šis modelis.

Regime

60. pielikumā ir redzams Regime modeļa JSON piemērs.

Lauki:

- “name” – režīma nosaukums;
- “regime_id” – `shared_types::RegimeId` modelis;
- “regime_value” - `shared_types::RegimeValue` modelis.

Apraksts:

`shared_types::Regime` ir galvenais modelis, kas uztur sevī visu informāciju par vienu režīmu. Visi dati kas atrodas iekš šī modeļa tiek arī uzglabāti datubāzē.

SavedRegimes

61. pielikumā ir redzams SavedRegimes modeļa JSON piemērs.

Lauki:

- “saved_regimes” – saraksts no `shared_types::Regime` modeļiem.

Apraksts:

Šis modelis ir tikai saraksts no `shared_types::Regime`. Iekš C++ koda tas tiek deserializēts par `std::vector<shared_types::Regime>` datu tipa mainīgo. Šis

modelis tiek izmantots, lai nosūtītu pilnu sarakstu ar visiem modeļiem un to vērtībām, attēlošanai.

5.1.3. *shared_types* Python JSON serializācija un deserializācija

Visi JSON modeļi netiek padoti pa taisno starp Anticarium Desktop un Anticarium Pi. Kā jau tika minēts iepriekš. Starp Anticarium Desktop un Anticarium Pi atrodas HTTP serveris kā Anticarium WEB projekts. Šis HTTP serveris izmanto Python Flask ietvaru, lai veiktu ienākošo datu apstrādi, padošanu tālāk, uzglabāšanu. Tā pat kā iekš C++, arī Python ir nepieciešama objektu serializācija un deserializācija drošākam kodam.

Atšķirībā no C++ koda, iekš Python šoreiz netiek izmantota atsevišķa JSON bibliotēka objektu serializācijai un deserializācijai, bet gan iebūvētās Python vārdnīcas, jeb dictionaries. 62. pielikumā ir redzama `control` modeļa serializācijas un deserializācijas Python implementācija.

Galvenā doma dotajā Python `control` modeļa koda piemērā ir tāda pati kā iekš C++ `shared_types::Control` – ir galvenā modeļa klase un divas funkcijas serializācijai un deserializācijai.

5.2. Datu uzglabāšana datubāzē

Galvenā Anticarium terārija funkcionalitāte ir gaisa temperatūras un zemes mitruma uzturēšana. Taču, dažādos dienas laikos vai pat gadalaikos mēdz rasties nepieciešamība nomainīt iestatītās vērtības uz citām. Piemēram, atslēgt terārija automātisko darbību uz uzkopšanas brīdi, vai arī nomainīt galvenos uzturēšanas parametrus uz ko tādu, kas ir atbilstošs gadalaikam. Abos gadījumos būtu ērti iestatīt viena veida parametrus un tad, pēc kāda laika, atgriezties pie vecajiem. Šim mērķim Anticarium Desktop ļauj izveidot režīmus. Režīmi tiek uzglabāti datubāzē.

Datubāzes izstrāde

Datubāzes projektēšanas rezultātā tika izveidota viena tabula, kurā arī tiek glabāti visi lietotāja izveidotie režīmi, jeb `shared_types::Regime` modeļi. Tabula ir redzama 63. pielikumā.

Ir svarīgi piebilst, ka, kā datubāzes dzinīs, izmantotais SQLite izmanto tikai parastākos SQL datu tipus. Ierakstu datu tipu izmēri netiek iestatīti, bet gan tiek automātiski noteikti. Izņēmums ir REAL datu tips, kas vienmēr aizņem 8 baitus.

Tā kā SQLite izmanto vienu failu datu uzglabāšanai, šis fails ir jāizveido. Tam ir paredzēts skripts `Anticarium_WEB/sqlite.py`.

Python skripts, kas attēlots 64. pielikumā izveido `anticarium.db` SQLite datubāzes failu un tur izveido tabulu, kura tika iepriekš aprakstīta. Iespējamais datubāzes tabulas aizpildījums ir attēlots 65. pielikumā.

5.3. Datu apmaiņas formāts izmantojot I²C

Neskatoties uz to, ka I²C ir pilntiesīgs datu apmaiņas protokols, tā pat kā arī HTTP datu apmaiņas protokolā, dati ir jānosūta tā, ka tos būtu ērti izmantot tai pusei uz kuru dati tiek sūtīti. Runājot konkrēti par Input IC un Output IC, uz šīm ierīcēm ir nepieciešams sūtīt komandas un saņemt atbildes. Pieprasījumu un atbilžu formātiem ir jābūt ērti izveidojamiem, saprotamiem.

JSON, kā datu formāts šoreiz neder, tāpēc, ka būtu jāveic JSON serializācija, deserializācija mikroshēmu pusē. Šādām darbībām ir nepieciešama attiecīga bibliotēka, kas aizņemtu daudz atmiņas, kuras uz dotajām mikroshēmām nav daudz. Sekojoši, ir nepieciešams kāds vienkāršs, specifisks datu formāts. Kā variants šādam datu formātam ir piemeklēt jau gatavu bibliotēku, jo šī problēma nav reti sastopama un noteikti jau ir jābūt gatavam risinājumam, taču Anticarium gadījumā ir neliels datu daudzums, kas ir jānosūta, un šādu datu formātu var izveidot arī pats. Liels pluss šādai pieejai ir tāds, ka tas ļautu izveidot datu formātu specifiski šī projekta vajadzībām, tādējādi tas būtu maksimāli efektīvs atmiņas izmēra ziņā.

5.3.1. *Datu tipu savietojamība starp dažādām platformām*

C++ programmas, kas darbojas uz Raspberry Pi atšķiras no tām, kas darbojas uz AVR mikroshēmām. Viena no svarīgākajām atšķirībām ir tas cik fiziski daudz atmiņas aizņem primitīvie mainīgie¹⁰. Atmiņas daudzumu, ko aizņem primitīvie mainīgie nosaka kompilators. Šoreiz, kompilejot C++ programmas priekš Input IC un Output IC tiek izmantots AVR-GCC [37] kompilators, bet C++ programmas priekš Raspberry Pi – GCC [38].

Kā otra tik pat svarīga atšķirība ir ierīču endianitāte. Endianitāti nosaka ierīces procesora arhitektūra. Mikroshēmu gadījumā, pati mikroshēma ir procesors. Endianitāte ir veids kādā procesors nolasa atmiņu. Lielās endianitātes gadījumā no atmiņas kā pirmais tiek ielasīts baits, kas sevī glabā lielākās binārās skalas vērtības. Mazās endianitātes gadījumā, kā pirmais tiek ielasīts baits, kas sevī glabā mazākās binārās skalas vērtības. I²C protokols nosūta datus tādā

¹⁰ Primitīvais datu tips ir datu tips, kuru var attēlot ļoti vienkāršā veidā, piemēram, skaitli, simbolu, patiesības vērtību [47]

veidā kādā tie tiek padoti un neveic endianitātes korekcijas sistēmai, kas saņem datus. Šī klūst par problēmu gadījumā, ja savā starpā komunicē divas ierīces ar dažādām endianitātēm, jo tās saņemtu datus apgrestā veidā.

Kā pēdējā svarīgākā atšķirība ir skaitļu ar peldošo komatu uzbūve. Atšķirībā no citiem primitīvajiem datu tipiem, kuriem vērtība tiek noteikta attiecīgi pēc binārās skalas, kur to uzbūve ir viegli saprotama, jo katrs nākamais bits ir iepriekšējā bita kāpinājums otrajā pakāpē, skaitļiem ar peldošo komatu ir specifiska uzbūve. Ja šī uzbūve atšķiras, tad ir nepieciešama relatīvi sarežģīta vērtības pārveidošana no viena tipa uz citu. Skaitļu ar peldošo komatu standartu nosaka IEEE 754 standarts [39].

5.3.2. *ARM un AVR platformu savietojamība*

Atbilstoši ARM-GCC [37] un GCC [40] kompilatoru dokumentācijām, var secināt, ka atšķiras datu tipu izmēri starp ARM un AVR platformām. Lai izveidotu datu formātu ko sūtīt starp mikroshēmām un Raspberry Pi, ir nepieciešams ņemt vērā, ka uz AVR platformas int primitīvā datu tipa izmērs aizņem 2 baitus, bet uz ARM – 4 baitus. float, jeb skaitļu ar peldošo komatu datu tipi ir vienādi starp abām dotajām platformām. float datu tips abās platformās aizņem 4 baitus, sekojoši, šis datu tips ir vienāds starp abām platformām

Gan ARM, gan AVR sistēmas izmanto mazo endianitāti, sekojoši, endianitātes pārveidošana nav nepieciešama.

5.3.3. *Input IC datu formāts*

Tā kā I²C protokols ir master-slave tipa protokols, Input IC nevar pats par sevi nosūtīt datus uz Raspberry Pi, jo Input IC ir slave un Raspberry Pi ir master. I²C komunikācijā viss notiek pēc master pieprasījumiem. Anticarium sistēmā Raspberry Pi ir master, sekojoši, Raspberry Pi veiks datu pieprasījumus no Input IC un Input IC atgriezīs sensoru lasījumus. Atgrieztie dati ir izveidota pēc sekojoša formāta:

<identifier><value>

Kur “identifier” var būt viens no sekojošiem simboliem:

- h – gaisa mitrums [%];
- m – zemes mitrums [%];
- t - temperatūra [°C].

“identifier” ir `int8_t`¹¹ tipa vērtība.

“value” ir `float` tipa vērtība, kas arī ir lasījuma vērtība.

5.3.4. *Output IC datu formāts*

Dotās mikroshēmas programma tika izveidota tā, ka tā neatgriež datus pēc I²C pieprasījuma, jo dotajai mikroshēmai vērtības ir nepieciešams tikai iestatīt. No Raspberry Pi nosūtītās ziņas, datu struktūras formāts ir sekojošs:

`<pin><value>`

Kur:

- mazāks-lielāks un lielāks-mazāks iekavas netiek ņemtas vērā, jo tās tiek izmantotas tikai piemēram;
- “pin” ir kājiņas numurs, kurai tiek iestatīta vērtība;
- “pin” datu tips ir `uint8_t`¹²;
- “value” ir vērtība skalā no 0 līdz 255, kas ir jāiestata kājiņai;
- “value” datu tips ir `uint8_t`.

5.4. Video nosūtīšanas formāts

Video datu nosūtīšana pamatā ir vairāku attēlu nosūtīšana. Anticarium gadījumā tas tiek darīts caur tīklu izmantojot UDP protokolu. Pilna izmēra attēlus nav iespējams nosūtīt caur tīklu izmantojot UDP, jo lielākais UDP paketes izmērs ir 65,507B [41]. Anticarium Camera ielasa RGB attēlus 1280x960 izmērā, sekojoši, viena attēla izmērs ir 3,686,400B. Tā kā tiek straumēta video tiešraide, attēla sadalīšana mazākās daļās un pārsūtīšana ir logisks risinājums, taču, UDP protokola īpatnība ir tāda ka UDP protokols nenodrošina datu secīgu pienākšanu. Tas nozīmē, ka attēla daļas var pienākt jauktā secībā un attēls tiks atradīts kropļots. Kā risinājums ir Anticarium Camera katra attēla sadalīšana sīkākās daļās, rindiņās. Lai nodrošinātu to, ka katra rindiņa tiek secīgi attēloti iekš Anticarium Desktop izmantojot UDP protokolu, katras rindas pirmie trīs baiti satur rindas kārtas skaitli. Katras nosūtītās attēla rindiņas struktūra ir redzama 24. pielikumā.

¹¹ 8 bitu izmēra mainīgais, viens bits nosaka skaitļa zīmi

¹² 8 bitu izmēra mainīgais, tikai pozitīvs skaitlis

Ātrdarbības iemeslu dēļ, šie trīs baiti tiek ierakstīti paša attēla rindiņas pirmajos trīs baitos, nevis attēla rinda tiek papildināta ar papildus trīs baitiem. Sekojoši, no 1280 rindiņas pikseļiem, tikai 1279 patiesi tiek arī atrādīti uz ekrāna.

6. LIETOTĀJA CELVEDIS

Anticarium sastāv no vairākām dažāda veida vidēm un sistēmām. Tik pat svarīgs cik lietotāja ceļvedis ir arī izstrādātāja ceļvedis, jo izstrādes vides uzstādīšana ir ļoti atšķirīga dažādām Anticarium daļām un sastāv no vairākiem soļiem, kā arī vidēm un pat ierīcēm.

6.1. Izstrādātājam

6.1.1. Versiju kontrole

Visas Anticarium projekta daļas vieno vienots versiju kontroles algoritms. Katrai izmaiņai tiek veidots atsevišķs git zars, un priekš tā apvienošanas ar galveno zaru tiek veidots ieraksts iekš CHANGELOG.md faila, par svarīgākajām izmaiņām zarā, un “Pull Request” iekš attiecīgās daļas Github repozitorijas, atkārtotai izmaiņu izskatīšanai.

Versijas numurā parasti tiek palielināts tikai vidējais skaitlis. Relīze, kur tiek mainīts trešais skaitlis tiek mainīts tikai tad, ja veiktās izmaiņas ir bijušas minimālas, un satur labojumus nelielām klūdām, kas tika pielaistas iepriekšējā relīzē. Versijas labējais skaitlis vienmēr paliek par nulli, ja tam skaitlis pa kreisi ir mainījis savu vērtību. Pirmais cipars versijas numurā netiek mainīts.

Ik pēc dažām izmaiņām ir nepieciešams veikt relīzi. Izmaiņu apjoms ik pēc kura tiek veidota relīze ir subjektīvs un var būt neregulārs. Veicot relīzi ir nepieciešams:

1. Veikt jaunās versijas ierakstu iekš CHANGELOG.md faila.
2. Atjaunot VERSION failu (ja tāds pastāv), ierakstot tur tādu pašu versiju kāda tika ierakstīta pirmajā solī.
3. Veikt git commit ar “Bump version” ierakstu uz galveno zaru.
4. Iekš attiecīgās Github repozitorijas izveidot jaunu “release” kopā ar jaunu, versijai atbilstošu “tag”, kura nosaukums sastāv no versijas numura ar v burtu sākumā, piemēram, v1.0.0.
5. Relīzes Github ierakstā tiek veikti visi attiecīgās versijas CHANGELOG.md faila ieraksti.

6.1.2. Anticarium Desktop un Shared_Types

Shared_Types un Anticarium Desktop tiek izstrādāti vienā un tajā pašā vidē, tāpēc to konfigurācija ir identiska.

Izstrādes vides prasības aparātūrai

- x64 bitu procesora arhitektūra;

- Windows 10 operētājsistēma;
- Administratora tiesības;

Izstrādes vides instalācija un palaišana

1. Visual Studio instalācija:

- 1.1. lejuplādēt Visual Studio installeri - <https://visualstudio.microsoft.com/thank-you-downloading-visual-studio/?sku=Community&channel=Release&version=VS2022&source=VS Landing Page&cid=2030&passive=false;>

- 1.2. atvērt lejuplādēto failu;
- 1.3. veikt instalācijas soļus līdz “Workloads” skatam;
- 1.4. izvēlēties “Desktop development with C++”;
- 1.5. “Install”;
- 1.6. Restartēt datoru.

2. Qt Creator un Qt instalācija:

- 2.1. lejuplādēt Qt installeri - <https://www.qt.io/download-thank-you?hsLang=en>;

- 2.2. atvērt lejuplādēto failu;
- 2.3. iziet cauri visiem instalācijas soļiem līdz “Installation Folder”;
- 2.4. izvēlēties tikai “Custom installation”, atķeļsēt visus citus ķekšus;
- 2.5. “Next”
- 2.6. ieķeļsēt “Qt 5.15.2” un atvērt tā apakšizvēlni, kur jābūt ieķeļsētiem sekojošiem punktiem (pārējos punktus nepieciešams atķeļsēt):
 - MSVC 2019 64-bit;
 - Sources;
 - Qt Network Authorization;
 - Qt Debug Information Files;
 - Qt Quick Timeline.

- 2.7. atvērt “Developer and Designer Tools” apakšizvēlni;
- 2.8. ieķeļsēt “CMake”, “OpenSSL”, “Ninja” ja tie vēl nav ieķeļsēti (pārējos punktus var atstāt ieķeļsētu);
- 2.9. turpināt instalāciju līdz galam.

3. git instalācija:

- 3.1. lejuplādēt git installeri: <https://github.com/git-for-windows/git/releases/download/v2.36.1.windows.1/Git-2.36.1-64-bit.exe>;

- 3.2. atvērt lejuplādēto failu;

3.3. veikt instalāciju akceptējot visas noklusējuma opcijas.

4. Github Desktop instalācija:

4.1. lejuplādēt Github Desktop installeri:
<https://central.github.com/deployments/desktop/desktop/latest/win32>;

4.2. atvērt lejuplādēto failu;

4.3. veikt instalāciju;

4.4. ielogoties savā Github profilā.

5. Klonēt Anticarium/Desktop repozitoriju izmantojot Github Desktop ar linku:
[https://github.com/Anticarium/Desktop](https://github.com/Anticarium/Anticarium/Desktop).

6. Klonēt Shared_Types repozitoriju izmantojot Github Desktop ar linku:
https://github.com/Anticarium/Shared_Types.

7. VCPKG instalācija:

7.1. klonēt VCPKG repozitoriju izmantojot Github Desktop un linku:
<https://github.com/microsoft/vcpkg.git>;

7.2. atvērt VCPKG direktoriju izmantojot komandu čaulu;

7.3. > .\bootstrap-vcpkg.bat

7.4. > vcpkg install nlohmann-json --triplet=x64-windows

7.5. > vcpkg install spdlog --triplet=x64-windows

7.6. > vcpkg install gtest --triplet=x64-windows

7.7. > vcpkg integrate install

7.8. pēc pēdējās komandas izpildes tiks uzrādīts parametrs ar tā vērtību, kuru būs jāizmanto projekta kompilācijā. Šo parametru ar vērtību nepieciešams uzbrīdi uzglabāt turpmākajiem soliem.

8. clang-format iestatīšana priekš Qt creator:

8.1. atvērt Qt creator;

8.2. “Help”->”About Plugins”;

8.3. ieķeksēt “Beautifier”;

8.4. restartēt Qt creator;

8.5. “Tools”->”Options”->”Beautifier”;

8.6. ieķeksēt “Enable auto format on file save”;

8.7. pie “Tool:” izvēlēties “ClangFormat”;

8.8. doties uz “Clang Format” sadaļu;

8.9. “Browse”, lai atrastu clang-format programmu. Tā atrodas C:\Program Files\Microsoft Visual Studio\2022\Community\VC\Tools\LLVM\bin\clang-format.exe;

- 8.10. pie “Use predefined style” izvēlēties “File”;
- 8.11. “Ok”;
- 8.12. Darba direktorijā iekopēt .clang-format failu no https://github.com/Anticarium/Anticarium_Doc/blob/main/.clang-format-12.
9. Qt creator konfigurēšana priekš Anticarium Desktop:
- 9.1. atvērt Qt creator;
 - 9.2. “Open project”;
 - 9.3. atrast Anticarium/Desktop direktoriju un veikt dubultklikšķi uz CMakeLists.txt;
 - 9.4. izvēlēties nepieciešamo izstrādes komplektu un ieķeksēt tikai “Debug” un “Release” konfigurācijas;
 - 9.5. iestatīt “Debug” kompilācijas direktoriju uz: C:\Users\<Lietotājs>\Documents\GitHub\Anticarium/Desktop\build\Debug;
 - 9.6. iestatīt “Release” kompilācijas direktoriju uz: C:\Users\<Lietotājs>\Documents\GitHub\Anticarium/Desktop\build\Release;
 - 9.7. “Configure project”;
 - 9.8. būs redzams, ka projektam neizdosies nokonfigurēties. Tas notiek tāpēc, ka ir nepieciešams padot VCPKG solī saglabāto CMake parametru:
 - 9.8.1. Doties uz “Projects”;
 - 9.8.2. Iekopēt saglabāto CMake parametru (bez pēdiņām);
 - 9.8.3. “Re-configure with Initial Parameters”.
 - 9.9. 9.8. punktā tika veikti soli, lai konfigurētu “Debug” vidi, ko pēc noklusējuma piedāvā Qt creator kā pirmo konfigurācijas variantu. Izstrāde pārsvarā notiek izmantojot “Release” konfigurāciju, jo “Debug” tiek izmantota tikai tad kad ir nepieciešams atrast problēmu cēloņus. Lai nokonfigurētu arī “Release” vidi ir nepieciešams izvēlēties “Release” kā “Build type:” un veikt visus iepriekšminētos solus punktā 9.8.;
 - 9.10. Anticarium Desktop izstrāde izmantojot Qt Creator ir gatava. Uzspiežot pogu “Build” projekts tiks nokompilēts un uzspiežot pogu “Run” Anticarium Desktop programma tiks palaista.
10. Shared_Types Qt creator konfigurēšana notiek tā pat kā priekš Anticarium Desktop 9. solī.
11. Unit testing pirmkoda redīgēšana pēc noklusējuma nav pieejama. Lai to iedarbinātu, papildus ir nepieciešams padot sekojošus CMake parametrus:
- 11.1. -DADD_anticarium_desktop_TESTS=ON priekš Anticarium Desktop;
 - 11.2. -DADD_shared_types_TESTS=ON priekš Shared_Types;

11.3. No jauna palaist CMake. Tagad parādīsies arī tests direktorija direktorija. Kamēr šie parametri ir pieejami tiks kompilējoties tiks izveidotas arī Unit Testing programmas, kuras ir iespējams izvēlēties un palaist no Qt creator.

Anticarium Desktop palaišana:

Nokompilējot Anticarium Desktop programmu, to ir iespējams palaist izmantojot “Run” pogu iekš Qt creator. Tiks automātiski veiktas nepieciešamās darbības, lai palaistu kompilēto programmu, kas atrodas iekš build\apps\ ar nosaukumu anticarium_desktop_app.exe. Taču, šo programmu nav iespējams uzreiz palaist izmantojot dubultklikšķi uz ikonas, jo radīsies vairākas klūdas par bibliotēkām, kuras nav iespējams atrast. Lai sagatavotu vidi programmas palaišanai izmantojot ikonu ir nepieciešams:

1. Izveidot jaunu direktoriju ārpus Anticarium Desktop direktorijas.
2. No iepriekšminētās build\apps\ direktorijas nokopēt visus .dll failus, settings.ini, pašu programmu, un iekopēt jaunajā direktorijā.
3. Izveidot platforms un styles direktorijas.
4. No <Qt instalācijas direktorija>\5.15.2\msvc2019_64\bin iekopēt:
 - 4.1. Qt5Core.dll;
 - 4.2. Qt5Gui.dll;
 - 4.3. Qt5Network.dll;
 - 4.4. Qt5Widgets.dll.
5. No <Qt instalācijas direktorija>\5.15.2\msvc2019_64\plugins\platforms nokopēt qwindows.dll un iekopēt to iekš platforms\ direktorijas.
6. No <Qt instalācijas direktorija>\5.15.2\msvc2019_64\plugins\styles\ nokopēt qwindowsvistastyle.dll un iekopēt to iekš platforms\ direktorijas.

Anticarium Desktop darbībai ir nepieciešama arī servera IP adrese pie kura pieslēgties. Tas ir jānorāda iekš settings.ini.in pirms kompilācijas darba direktorijā, vai arī iekš settings.ini faila, kas atrodas pie Anticarium Desktop programmas.

Shared_Types palaišana

Shared_Types kompilācijas rezultāts pēc noklusējuma neveido programmu, jo Shared_Types ir C++ bibliotēka. Tomēr, lai pārbaudītu bibliotēkas darbību, vienīgais Shared_Types palaišanas veids ir kompilējot Unit Testing programmu, kas tam ir izveidots.

Versiju kontrole priekš Anticarium Desktop

Anticarium Desktop reļīze tiek veidota tā pat kā tika aprakstīts sadaļā 6.1.1., bet papildus pievienojot arī .zip arhīvu, kas satur visus nepieciešamos failus jaunākās Anticarium Desktop lietotnes palaišanai. Piemērs .zip arhīva nosaukumam: anticarium_desktop_v1.5.0_windows_x64.zip

Versiju kontrole priekš Shared_Types

Shared_Types reļīze tiek veidota tā pat kā tika aprakstīts sadaļā 6.1.1, taču Shared_Types reļīzes netiek veidotas pēc subjektivitātes principa. Shared_Types reļīžu numurus lieto Anticarium Desktop un Anticarium Camera (lai attiecīgo reļīzi lejuplādētu), tāpēc pēc Shared_Types izmaiņām ir svarīgi veikt arī reļīzi, lai šīs izmaiņas varētu iekļaut arī iekš Anticarium Camera un Anticarium Desktop.

6.1.3. Anticarium Pi un Anticarium Camera

Anticarium Pi un Anticarium Camera izstrāde notiek uz Linux operētājsistēmas. Kopējai Anticarium projekta izstrādei ir ieteicams lietot Windows operētājsistēmu kā galveno operētājsistēmu un Linux operētājsistēmu izmantojot virtuālo mašīnu.

Izstrādes vides prasības aparatūrai

- Ubuntu 20.04 operētājsistēma.
- Superuser tiesības.

Izstrādes vides instalācija un palaišana

Galveno rīku un pakotņu automātiskai uzstādīšanai tika izveidots Anticarium_Pi/setup/setup_vm.sh [45] skripts. Skriptam ir iespējams padot trīs sekojošus argumentus:

- -h skripts izvada palīginfo un īsu aprakstu par tā darbību un iespējamiem argumentiem;
- -a <ip_address> šis arguments sagaida, ka tam tiks padota Anticarium Rasberry Pi lokālā IP adrese, kas tiek saglabāta RPI_IP vides mainīgajā. Tieka uzsākta sistēmas pirmās reizes automātiskā instalācija;
- -k tiek veikta host mašīnas SSH atslēgas reģistrēšana uz Raspberry Pi. Šīs darbības izpildei Raspberry Pi ir jābūt ieslēgtam, pieslēgtam pie lokālā tīkla, RPI_IP vides mainīgajam jābūt iestatītam;
- Bez padotiem argumentiem šis skripts uzsāk pirmās reizes automātisko instalāciju taču uzrādīs kļūdu, ja tiks noskaidrots, ka RPI_IP vides mainīgais nav iestatīts.

`setup_vm.sh` skripts spēj instalēt tikai tos elementus, kurus ir iespējams instalēt automātiski. IDE instalēšanu un konfigurēšanu jāveic manuāli. Visi izstrādes vides instalācijas soļi:

1. Automātiskās instalācijas soļi:

- 1.1. `$ cd $HOME`
- 1.2. `$ wget raw.githubusercontent.com/Anticarium/Anticarium_Pi/master/setup/setup_vm.sh`
- 1.3. `$ chmod 755 setup_vm.sh`
- 1.4. `$./setup_vm.sh -a <Raspberry_Pi_IP_adrese>`
- 1.5. CMake automātiskās instalācijas laikā būs nepieciešamība akceptēt CMake autortiesību politiku izmantojot “y”, un brīdī, kad CMake instalācija jautās par CMake instalācijas vietu būs jāatbild ar “n”;
- 1.6. automātiskās instalācijas beigās notiks sistēmas restarts.

2. Qt Creator un Qt instalācija:

- 2.1. lejuplādēt Qt installeri: <https://www.qt.io/download-thank-you?hsLang=en>;
- 2.2. `$ chmod 755 $HOME/Downloads/<Qt instalācijas faila nosaukums>`
- 2.3. atvērt lejuplādēto failu;
- 2.4. iziet cauri visiem instalācijas soļiem līdz “Installation Folder”;
- 2.5. izvēlēties tikai “Custom installation”, atķeeksēt visus citus ķekšus;
- 2.6. “Next”;
- 2.7. labajā pusē ieķeeksēt tikai “Archive”;
- 2.8. “Filter”;
- 2.9. ieķeeksēt “Qt 5.11.2” un atvērt tā apakšizvēlni, kur jābūt ieķeeksētiem sekojošiem punktiem (pārējos punktus nepieciešams atķeeksēt):
 - Desktop GCC 64-bit;
 - Sources;
 - Qt Network Authorization;
 - Qt Debug Information Files.
- 2.10. atvērt “Developer and Designer Tools” apakšizvēlni;
- 2.11. ieķeeksēt “OpenSSL”, “Ninja” ja tie vēl nav ieķeeksēti (pārējos punktus var atstāt ieķeeksētus);
- 2.12. turpināt instalāciju līdz galam.

3. Qt creator konfigurēšana priekš Anticarium_Pi:

- 3.1. “Open Project”;
 - 3.2. Atvērt `Anticarium_Pi/CMakeLists.txt` (Automātiskās konfigurācijas skripts jau klonēja `Anticarium_Pi`);
 - 3.3. Izvēlēties nepieciešamo izstrādes komplektu un ieķeksēt tikai “Debug” un “Release” konfigurācijas;
 - 3.4. Kompilācijas direktorijas atstāt pēc noklusējuma;
 - 3.5. “Configure project”;
 - 3.6. Projekts veiksmīgi, automātiski, tiks nokonfigurēts;.
4. clang-format tiek iestatīšana tiek darīta tā pat kā iekš Windows vides ar diviem izņēmumiem:
 - 4.1. clang-format programmas atrašanās vietu var uzzināt izpildot komandu `$ which clang-format`;
 - 4.2. `.clang-format` fails tiek ņemts no šī linka: https://github.com/Anticarium/Anticarium_Doc/blob/main/.clang-format-9.
 5. Pēc noklusējuma Anticarium WEB rezultātu netiek klonēta. Tās izstrāde notiek reti un parasti uz paša Raspberry Pi, lai uzreiz redzētu konfigurācijas izmaiņu rezultātu. Nepieciešamības gadījumā to ir iespējams noklonēt izmantojot: `$ git clone https://github.com/Anticarium/Anticarium_Web.git`.

Anticarium Pi un Anticarium Camera palaišana

Iekš Qt creator uzspiežot “Build” pogu programma tiks nokompilēta, bet tā joprojām atradīsies uz Ubuntu 20.04 mašīnas, nevis uz Raspberry Pi. Programmas arhitektūra arī nebūs saderīga ar ARM procesoru, kas atrodas uz Raspberry Pi. Ir nepieciešams veikt “cross compile”¹³. Šāda pieeja ļautu izmantot daudz jaudīgāku datoru par Raspberry Pi, tādējādi kompilējot programmas stipri ātrāk nekā tas ir iespējams uz paša Raspberry Pi. Tomēr, problēma ar šādu pieju ir tāda, ka krusta kompilācijas vidi ir grūti iestatīt. Anticarium izveides laikā noskaidrojās, ka krusta kompilācijas vides iestatīšana aizņem vairāk laika nekā tā varētu jebkādā veidā ietaupīt, tāpēc tika izveidots `pi.py` [46] skripts, kas veic kompilāciju vienkāršākā, bet lēnākā veidā. `pi.py` skripts ietver sevī dažādas funkcionalitātes, bet tā galvenā funkcija ir programmas koda kompilācija priekš Raspberry Pi. Dotais skripts augšuplādē programmas kodu no Ubuntu 20.04 mašīnas uz lokālajā tīklā pieslēgto Raspberry Pi un palaiž uz tā koda kompilāciju.

¹³ Cross compiling, jeb krusta kompilēšana ir programmas koda kompilēšana, lai izveidotu programmu, kas ir saderīga ar citu procesora arhitektūru nekā tā uz kurās notiek kompilācija

pi.py skripta komandas un argumenti:

- build palaiž koda kompilāciju uz Raspberry Pi;
- upload augšuplādē visus failus, kas atrodas iekš Anticarium_Pi direktorijas uz Raspberry Pi;
- install secīgi izpilda build and upload komandas;
- cmd [args] izpilda padoto komandu uz Raspberry Pi;
- shell atver Raspberry Pi komandas čaulu;
- run --app palaiž padoto programmu uz Raspberry Pi. Pieejamas trīs opcijas:
 - pi palaiž Anticarium Pi;
 - cam palaiž Anticarium Camera;
 - tests palaiž Anticarium Unit Testing programmu.
- -h, --help izprintē visu komandu aprakstus un īsu aprakstu par pašu skriptu.

Versiju kontrole priekš Anticarium Pi un Anticarium Camera

Relīzes, ieraksti iekš CHANGELOG.md faila abām projekta daļām notiek vienlaicīgi vienā un tajā pašā failā.

6.1.4. Raspberry Pi iestatīšana priekš Anticarium

Izstrādes vides prasības aparatūrai

- >8GB desmitās klases MicroSD atmiņas karte.
- microSD karšu lasītājs.

Izstrādes vides instalācija un palaišana

Galveno rīku un pakotņu automātiskai uzstādīšanai tika izveidots Anticarium_Pi/setup/setup_rpi.sh [47] skripts. Skriptam ir iespējams padot divus sekojošus argumentus:

- -h skripts izvadīs palīginfo un īsu aprakstu par tā darbību un iespējamiem argumentiem;
- -a <ip_address> šis arguments sagaida, ka tam tiks padota rūtera publiskā IP adrese, kas tiks saglabāta ANTICARIUM_SERVER_IP vides mainīgajā. Tieks uzsākta sistēmas pirmās reizes automātiskā instalācija;
- bez padotiem argumentiem šis skripts uzsāks pirmās reizes automātisko instalāciju taču uzrādīs kļūdu, ja tiks noskaidrots, ka ANTICARIUM_SERVER_IP vides mainīgais nav iestatīts.

Pirmās reizes vides iestatīšanas soļi:

1. Raspbian Lite operētājsistēmas instalācija:

- 1.1. lejuplādēt Raspberry Pi Imager installeri:
https://downloads.raspberrypi.org/imager/imager_latest.exe;
 - 1.2. atvērt lejuplādēto failu;
 - 1.3. beikt Raspberry Pi Imager instalāciju;
 - 1.4. iespraust datorā MicroSD atmiņas karšu lasītāju ar microSD karti;
 - 1.5. atvērt Raspberry Pi Imager;
 - 1.6. “CHOOSE OS”->”Raspberry Pi OS (Other)”->”Raspberry Pi OS Lite (32-bit)”;
 - 1.7. ”CHOOSE STORAGE”->”IESPRAUSTĀ SD KARTE”;
 - 1.8. ”WRITE”;
 - 1.9. sagaidīt līdz rakstīšanas process tiks pabeigts;
 - 1.10. izņemt MicroSD atmiņas karti.
2. Raspbian konfigurēšana:
 - 2.1. iespraust MicroSD karti ar Raspbian operētājsistēmu Raspberry Pi MicroSD karšu lasītājā;
 - 2.2. ieslēgt Raspberry Pi;
 - 2.3. izvēlēties tastatūras valodu;
 - 2.4. iestatīt lietotāja vārdu kā “pi”;
 - 2.5. iestatīt paroli;
 - 2.6. \$ wget
https://raw.githubusercontent.com/Anticarium/Anticarium_Pi/master/setup/setup_rpi.sh
 - 2.7. \$ chmod +x setup_rpi.sh
 - 2.8. \$./setup_rpi.sh -a <ip_address>
 - 2.9. automātiskās instalācijas beigās notiks sistēmas restarts.

Pirmās reizes iestatīšana priekš Raspberry Pi Anticarium sistēmas var aizņemt vairāk kā stundu, jo notiks arī CMake programmas kompilācija no pirmkoda, lai tiktu iegūta viena no jaunākajām CMake versijām. `setup_rpi.sh` arī iestatīs WEB serveri izmantojot Anticarium WEB `setup_web.sh` [48] skriptu.

`setup_rpi.sh` skripts veic tikai vides instalāciju un konfigurāciju. Tas neveic Anticarium_Pi Github repozitorijas klonēšanu un pirmkoda kompilāciju. Tā ir papildus darbība, kas ir jāveic izstrādātājam.

6.1.5. Anticarium IC

Izstrādes vides prasības aparatūrai

- Windows 10 operētājsistēma/
- Administratora tiesības.
- Arduino UNO.

Izstrādes vides instalācija un palaišana

1. Visual Studio Code instalācija:

1.1. lejuplādēt Visual Stidop Code installeri:
<https://code.visualstudio.com/docs/?dv=win;>

- 1.2. atvērt lejuplādēto failu;
- 1.3. beikt instalāciju.

2. Visual Studio Code konfigurēšana:

- 2.1. atvērt Visual Studio Code;
- 2.2. “Extensions”;
- 2.3. meklētājā atrast un instalēt:

- C/C++;
- PlatformIO.

- 2.4. restartēt Visual Studio Code.

3. Arduino IDE instalācija:

- 3.1. lejuplādēt Arduino IDE installeri: <https://www.arduino.cc/en/donate/>;
- 3.2. atvērt lejuplādēto failu;
- 3.3. veikt instalāciju.

4. Klonēt Anticarium_IC repozitoriju izmantojot Github Desktop un linku:
https://github.com/Anticarium/Anticarium_IC.git.

Anticarium IC instalācija un palaišana

Anticarium IC mikroshēmu programmēšana notiek izmantojot Arduino UNO un, Anticarium projekta ietvaros izveidotas, plates starpniecību, kas redzama 43. pielikumā un tās elektronisko shēmu 44. pielikumā. Dotās plates divas priekšējās mikroshēmu novietnes ļauj veikt to programmēšanu, bet abas apakšējās novietnes ļauj veikt augstsrieguma programmēšanu. Augšējās un apakšējās novietnes ir izveidotas divu dažādu izmēru mikroshēmām.

Output IC programmēšanas piemēra soļi:

1. atvērt Visual Studio Code;

2. “PlatformIO”->”Open”->”Open Project”;
3. atrast Anticarium_IC repositorijas mapi un doties uz `src\attiny_output`
4. “Open attiny_output”;
5. mikroshēmas programmēšanas sagatavošanas soli:
 - 5.1. savienot programmatora plati ar Arduino UNO;
 - 5.2. iesprauzt Output IC programmatora priekšējās daļas attiecīgajā spraudnī;
 - 5.3. pieslēgt programmatoru pie datora izmantojot USB portu;
 - 5.4. atvērt Arduino IDE;
 - 5.5. “Tools”->”Port”->Ports kur ir pieslēgts Arduino UNO;
 - 5.6. “File”->”Examples”->”ArduinoISP” ->”ArduinoISP”;
 - 5.7. “Upload”.
6. Visual Studio Code doties uz `platformio.ini` failu un iestatīt `upload_port` vērtību uz to portu pie kura ir pieslēgts programmators;
7. “PlatformIO”;
8. “Upload” un “Set Fuses”.

Input IC tiek programmēts tādā pašā veidā, bet ir nepieciešams Input IC mikroshēmu novietot attiecīgajā spraudnī, kā arī atvērt `src\attiny_input` mapi izmantojot PlatformIO;

6.1.6. Servera vides sagatavošana

Anticarium izstrāde var notikt tikai ja izstrādes dators un Raspberry Pi atrodas vienā tīklā. Šāds slēgums ļauj veikt SSH savienojumu ar Raspberry Pi, bet manāmi traucē Anticarium Camera darbību. Anticarium Pi darbojas kā UDP un TCP serveris. Tā optimālai darbībai rūterim ir nepieciešams veikt portu pāradresāciju. Nepieciešams veikt:

- 5011 porta pāradresāciju UDP pakām uz Raspberry Pi lokālo IP adresi;
- 80 porta pāradresāciju TCP pakām uz Raspberry Pi lokālo IP adresi.

Anticarium serverim un izstrādes datoram atrodoties vienā lokālajā tīklā, rūterim rodas problēmas ar Anticarium Camera attēlu UDP pakotņu nosūtīšanu uz Anticarium Desktop - pakas netiek pārsūtītas no rūtera uz izstrādes datoru. Tāpēc, kā risinājums ir veikt papildus porta pāradresāciju rūterim:

- 55622 ports UDP pakām uz izstrādes datora lokālo IP adresi;

Lai Anticarium Desktop darbotos ar šādu konfigurāciju, pirms kompilācijas, iekš `settings.ini.in`, `Client_UDP_Port` ir nepieciešams iestatīt uz 55622. Kad šis iestatījums ir izveidots, pēc kompilācijas Anticarium Desktop gaidīs ienākošās UDP paketes uz

iepriekšminētā porta. Papildus, Anticarium servera publiskā IP adrese ir jāievada settings.ini.in failā gan priekš Anticarium Desktop, gan priekš Anticarium Pi, attiecīgajos laukos.

6.2. Lietotājam

Tiek paredzēts, ka lietotājs saņem jau gatavu terāriju, taču tam ir jāveic daži labojumi, lai nokonfigurētu lietotni un Anticarium serveri.

Prasības aparātūrai un tiesībām

- Dators ar x64 bitu procesora arhitektūru.
- Windows 10 operētājsistēma.
- Administratora tiesības datoram.
- Dators pieslēgts pie lokālā tīkla.
- Piekļuve rūtera iestatījumiem.
- Atsevišķs viena līmeņa tīkls, pie kura pieslēgt terāriju, un pie kura netiks pieslēgts dators, uz kura tiks darbināts Anticarium Desktop.

6.2.1. Terārija sagatavošana

1. Pieslēgt terārija interneta kabeli pie tā paša tīkla pie kura ir pieslēgts dators.
2. Iespāraut barošanas kabeli rozetē.

6.2.2. Servera vides sagatavošana

1. Ieiet rūtera vadības panelī.
2. Iegūt publisko rūtera IP adresi tīklam, pie kura tiks pieslēgts Anticarium terārijs.
3. Veikt portu pāradresāciju tīklam, pie kura tiks pieslēgts Anticarium terārijs. LAN un WAN portu adreses vienādas:
 - 5011 porta pāradresāciju UDP pakām uz terārija lokālo IP adresi;
 - 80 porta pāradresāciju TCP pakām uz uz terārija lokālo IP adresi.

6.2.3. Servera konfigurešana

1. Atvērt komandu čaulu Windows datorā.
2. > ssh pi@<terārija lokālā IP adrese>
3. Parole: 0000.
4. Veikt servera konfigurāciju atrodot un labojot rindiņas ar terārija rūtera publisko IP adresi:
 - 4.1. \$ nano .profile
 - 4.2. \$ export ANTICARIUM_SERVER_IP=<terārija rūtera publiskā IP adrese>

- 4.3. Ctrl+x;
 - 4.4. \$ sudo nano /etc/apache2/envvars
 - 4.5. \$ export ANTICARIUM_SERVER_IP=<terārija rūtera publiskā IP adrese>
 - 4.6. Ctrl+x;
 - 4.7. \$ nano Anticarium_Pi/build/apps/server_app/settings.ini
 - 4.8. ierakstīt terārija rūtera publisko IP adresi atbilstošajos laukos;
 - 4.9. Ctrl+x;
 - 4.10. \$ nano Anticarium_Pi/build/apps/camera_app/settings.ini
 - 4.11. ierakstīt terārija rūtera publisko IP adresi atbilstošajos laukos;
 - 4.12. Ctrl+x;
 - 4.13. \$ sudo reboot now
 - 4.14. nogaidīt vismaz divas minūtes.
5. Aizvērt komandu čaulu.
 6. Atslēgt terārija barošanu.
 7. Pārsprauzt terārija interneta kabeli uz atsevišķu tīklu.
 8. Pieslēgt terārija barošanu.
 9. Terārijs automātiski iedarbinās visas nepieciešamās programmas pie katras ieslēgšanās.

6.2.4. *Anticarium Desktop instalācija*

1. No <https://github.com/Anticarium/Anticarium/Desktop/releases> lejuplādēt jaunāko anticarium_desktop..._x64.zip arhīvu.
2. Ekstraktēt lejuplādēto .zip arhīvu;
3. Ieiet ekstraktētajā mapē.
4. Atvērt settings.ini.
5. Rediģēt Anticarium_Server_URL un Anticarium_UDP_URL atbilstoši terārija rūtera publiskajai IP adresei.

6.3. Anticarium Desktop apraksts

Anticarium Desktop logs ar tā elementu īsiem aprakstiem ir redzams 45. pielikumā. Kreisajā pusē ir redzams panelis ar trīs lieliem cipariem. Tie ir terārija sensoru lasījumi. Svarīgākie ir temperatūras un zemes mitruma lasījumi, jo tās ir vērtības ko lietotājs spēj mainīt iestatot dažādus apstākļus terārijā. Gaisa mitrums tiek tikai nolasīts, bet netiek izmantots.

Zem sensoru datu paneļiem atrodas esošā režīma nosaukums un tā izvēlne. Attēlā neviens režīms vēl netika izveidots, tāpēc izvēlne ir tukša. Atverot izvēlni var izvēlēties kādu

no saglabātajiem režīmiem pēc nosaukuma. Uzklikšķinot, šī režīma uzturamās vērtības lietotnē tiek automātiski iestatītas par esošajām.

Ekrāna vidū atrodas terārija tiešsaistes video.

Zem video ekrāna atrodas četri paneļi ar slīdbīdņiem. No kreisās puces lasot tie atbild par terārija uzturamo temperatūru, terārija uzturamo zemes mitrumu, gaismas spilgtumu, ventilatora griešanās ātrumu. Vērtības var mainīt izmantojot slīdbīžņus.

Zem apstākļu paneļiem atrodas “Save”, jeb saglabāšanas poga. Tā ļauj saglabāt esošās, iestatītās, uzturamās vērtības par režīmu. Poga aktivizējas tikai tad, ja pašlaik nav iestatīts neviens cits saglabātais režīms.

46. pielikumā ir iespēja redzēt atvērtu izvēlni, caur izvēlni var atvērt divus logus – lietotnes versijas informācijas logu un saglabāto režīmu skata logu.

47. pielikumā ir apskatāms lietotnes versijas informācijas logs, kas tika atvērts izmantojot "About" pogu no izvēlnes.

48. pielikumā var redzēt, ka terārijam tika mainīta uzturamā temperatūra, ka arī tiek veidots jauns režīms izmantojot “Save” pogu. Jauna režīma izveides logs ļauj iestatīt jaunā režīma nosaukumu.

49. pielikumā ir redzams, ka ir iedarbināts kāds režīms. “Save” poga ir aizbloķēta. Ir mainījies režīma nosaukums un režīmu izvēlne.

50. pielikumā ir redzami visi saglabātie režīmi saglabāto režīmu logā. Šis logs tika atvērts izmantojot “Edit regimes” izvēlnes pogu. Dotais logs ļauj izvēlēties jebkuru no saglabātajiem režīmiem un to rediģēt vai dzēst. Ja tiek dzēsts režīms, kas pašlaik ir iestatīts, tad režīma stāvoklis mainīsies uz “Custom”.

51. pielikumā ir redzams režīma rediģēšanas logs, kas tika atvērts izmantojot “Edit” pogu saglabāto režīmu logā. Rediģējot režīmu ir iespēja mainīt jebkuru no režīma parametriem. Ja tiek rediģēts esošais, iestatītais režīms, tā vērtība uzreiz stājas spēkā un ir redzama lietotnē.

6.4. Testa piemērs

Sekojošais tests tiek veikts pie nosacījuma, ka pirms tam tika veikti visi nepieciešamie terārija pirmās reizes iestatīšanas soļi:

6.4.1. Režīmu manipulācijas tests

1. Ieslēgt terāriju.

2. Pabīdīt temperatūras slīdbīdni uz kādu citu vērtību.
 - Esošajam režīmam jāmainās uz “Custom”. Režīma izvēlnes poga tiek atrādīta bez uzraksta.
3. “Save”.
 - Atveras režīma saglabāšanas logs.
4. Saglabāt režīmu ar nosakumu “Regime 1” un uzspiežot “Save”.
 - Režīma saglabāšanas logs tiek aizvērts. “Custom” režīma nosaukums kreisajā pusē nomainās uz jauno režīma nosaukumu.
 - Režīma izvēlnes pogas uzraksts mainās uz jaunā režīma nosaukumu.
5. Atkārtot soļus no 2. līdz 4., izveidojot jaunu režīmu ar “Regime 2” nosaukumu.
6. “Home”->”Edit regimes”.
 - Tieki atvērts visu saglabāto režīmu saraksts.
7. Izvēlēties “Regime 2” režīmu no saraksta un uzspiest “Delete”.
 - Izvēlētais režīms pazūd no saraksta. Esošais režīms tiek iestatīts uz “Custom”.
8. No saraksta izvēlēties “Regime 1” un uzspiest “Edit”.
 - Tieki atvērts režīma redīģēšanas logs ar šī režīma saglabātajām vērtībām.
9. Veikt izmaiņas jebkurā no ievades laukiem un uzspiest “Save”.
 - Režīmu sarakstā “Regime 1” vērtības tiek atjaunotas.
10. Aizvērt saglabāto režīmu logu.
11. Režīmu izvēlnes izvēlēties “Regime 1”.
 - Tieki iestatītas visas “Regime 1” režīma vērtības.
12. Aizvērt Anticarium Desktop lietotni.
13. Atvērt Anticarium Desktop lietotni.
 - Anticarium Desktop tiek atrādīts ar esošo režīmu “Regime 1” un tā iestatītajām vērtībām.

6.4.2. Terārija funkcionalitātes tests

1. Ieslēgt terāriju.
2. Atvērt Anticarium Desktop lietotni.
 - Šajā brīdī Anticarium Desktop lietotnei ir jāatveras un pēc ūsa brīža kreisajā pusē jāparādās sensoru lasījumiem no terārija. Ekrāna vidū tiek attēlots tiešsaistes video. Režīms un visas apstākļu vērtības tiek iestatītas uz tām, kas tika iestatītas pēdējo reizi.
3. Pabīdīt temperatūras slīdbīdni uz vērtību, kas ir augstāka par esošo temperatūru terārijā.

- Pēc ūdens slīdbīdņa iestāšanās.
4. Pabīdīt temperatūras slīdbīdņi uz vērtību, kas ir zemāka par esošo temperatūru terārijā.
 - Pēc ūdens slīdbīdņa iestāšanās.
 5. Pabīdīt ūdens slīdbīdņi uz vērtību, kas ir augstāka par esošo terārijā zemes mitruma vērtību.
 - Pēc ūdens slīdbīdņa iestāšanās.
 6. Pabīdīt ūdens slīdbīdņi uz 0%.
 - Pēc ūdens slīdbīdņa iestāšanās.
 7. Pabīdīt gaismas slīdbīdņi uz 50%.
 - Terārijā tiek izslēgta gaisma, kas spīd uz 50% no tās maksimālā spilgtuma;
 8. Pabīdīt gaismas slīdbīdņi uz 0%.
 - Terārijā tiek izslēgta gaisma.
 9. Pabīdīt ventilatora slīdbīdņi uz 80%.
 - Terārijā tiek iedarbināts ventilators, kas darbojas uz 50% no sava maksimālā apgriezienu skaita.
 10. Pabīdīt ventilatora slīdbīdņi uz 0%.
 - Terārijā tiek izslēgta ventilators.
 11. Iestatīt uzturamo temperatūru par 0.3°C augstāku nekā ir pašlaik terārijā.
 12. Sagaidīt līdz sildlampa iestāšanai un uzsilda gaisu terārijā līdz uzturamajai temperatūrai.
 - Šis process var aizņemt ilgu laiku, jo gaisa uzsildīšana nenotiek momentāli. Sildlampa izslēgsies sasniedzot uzturamo temperatūru un izslēgsies atpakaļ, kad gaišs atkal atdzīs zem uzturamās temperatūras.
 13. Iebērt terārijā zemi vai ieliet ūdeni.
 14. Iespaušt terārijā zemes mitruma sensoru zemē vai arī ielikt to ūdenī.
 15. Terārijā ūdens tvertnē ieliet ūdeni.
 16. Iestatīt uzturamo zemes mitruma vērtību uz 10%.
 - Sāksies ūdens iestāšanās terārijā. Tā beigsies, līdz zemes mitrums apzemes mitruma sensoru sasniegts 10% vai vairāk. Ja sensors tika ielikts ūdenī, tad zemes mitruma izmaiņas būs redzamas uzreiz. Tā pat kā ar temperatūras uzturēšanu, ūdens padeves motors atsāks darbību, kad zemes mitrums nokritīs zem iestatītās uzturamās vērtības.

NOBEIGUMS

Diplomdarba izstrādes laikā tika izveidota “Anticarium” viedā terārija sistēma atbilstoši izvirzītajām prasībām. Tika izveidota gan lietotne, gan terārijs, gan terārija vadības modulis. Izstrādes laikā tika apgūtas jaunas tehnoloģijas un metodes, kā arī atkārtotas un uzlabotas tās kurās bija jau zināmas.

Esošā Anticarium sistēma kā galaprojekts nav paredzēts lietotājam bez IT zināšanām, jo satur lielu skaitu sarežģītu, specifisku uzstādīšanas soļu. Anticarium sistēma satur arī vērā ņemamus drošības riskus, jo esošais risinājums neparedz lietotāja(-u) izveidi un jebkurš izmantojot internetu var pieslēgties pie viena un tā paša terārija izmantojot publisko rūtera IP adresi, pie kura ir pieslēgts terārijs. Neskatoties uz drošības riskiem un ietilpīgo uzstādīšanu, terārijs pilda savu uzdevumu un spēj automātiski uzturēt lietotāja uzstādītos apstākļus, kā arī ļauj vadīt terāriju attālināti un apskatīties video tiešsaisti.

Terārija fiziskais dizains tika izveidots ērts un parocīgs lietošanai domājot par ūdensdrošību un spēju terāriju tīrīt un kopt.

Github repozitorija: <https://github.com/Anticarium>

TERMINI

“Anticarium” viedā terārija sistēma sastāv no vairākām daļām, kas darbojas un sadarbojas kopā. Viena daļa var sastāvēt no vairākām sīkākām vienībām, kur katra vienība pilda savu uzdevumu. Vienības var būt arī apvienotas kopā pēc nozīmes, vienā terminā. Lai skaidri nodotu domu, samazinātu teksta apjomu, un ļautu lasītājam vieglāk uztvert informāciju - šajā diplomdarbā tiek lietoti sekojoši termini:

- Anticarium – “Anticarium” viedā terārija sistēma;
- Anticarium WEB – Anticarium HTTP serveris un datubāze, kas saņem un padod vērtības;
- Anticarium Pi – Anticarium programma, kas atbild par Anticarium terārija fizisku vadību;
- Anticarium Camera – Anticarium programma, kas atbild par video ielasīšanu un padošanu tālāk;
- Anticarium Desktop – Anticarium lietotne Windows vidē, kas ļauj lietotājam vadīt Anticarium Pi;
- Input IC – Attiny85 mikrokontrolieris, kas ir atbildīgs par datu ievadi terārijā;
- Output IC – Attiny24 mikrokontrolieris, kas ir atbildīgs par datu izvadi terārijā;
- Anticarium PCB – Anticarium iespiedshēmas plate, kas atbild par terārija datu ievadi un izvadi. Iekļauj sevīt Input IC un Output IC;
- Anticarium IC - Input IC un Output IC koda ziņā atrodas zem vienas Github repozitorijas un tiek uzskatītas kā viens projekts;

IZMANTOTIE AVOTI

- [1] Raspberry Pi Foundation, "Raspberry Pi mājaslapa," Raspberry Pi Foundation, [Online]. Available: <https://www.raspberrypi.com/>. [Accessed 6. marts 2022].
- [2] A. Cedilnik, "CMake mājaslapa," [Online]. Available: <https://cmake.org/>. [Accessed 6. marts 2022].
- [3] The Python Software Foundation, "Python mājaslapa," The Python Software Foundation, [Online]. Available: <https://www.python.org/>. [Accessed 6. marts 2022].
- [4] Nokia, "Qt mājaslapa," Nokia, [Online]. Available: <https://www.qt.io/>. [Accessed 6. marts 2022].
- [5] BCMI, "Arduino mājaslapa," BCMI, [Online]. Available: <https://www.arduino.cc/>. [Accessed 6. marts 2022].
- [6] A. Ronacher, "Flask mājaslapa," [Online]. Available: <https://palletsprojects.com/p/flask/>. [Accessed 6. marts 2022].
- [7] A. Stepanov, "STL mājaslapa," [Online]. Available: <https://en.cppreference.com/w/>. [Accessed 6. marts 2022].
- [8] Google, "google/googletest Github repozitorijs," Google, [Online]. Available: <https://github.com/google/googletest>. [Accessed 6. marts 2022].
- [9] nicholastmosher, "nicholastmosher/PID Github repozitorijs," [Online]. Available: <https://github.com/nicholastmosher/PID>. [Accessed 6. marts 2022].
- [10] nlohmann, "nlohmann/json Github repozitorijs," [Online]. Available: <https://github.com/nlohmann/json>. [Accessed 6. marts 2022].
- [11] cedricve, "cedricve/raspicam Github repozitorijs," [Online]. Available: <https://github.com/cedricve/raspicam>. [Accessed 6. marts 2022].
- [12] gabime, "gabime spdlog Github repozitorijs," [Online]. Available: <https://github.com/gabime/spdlog>. [Accessed 6. marts 2022].
- [13] Microsoft, "VCPKG mājaslapa," Microsoft, [Online]. Available: <https://vcpkg.io/en/index.html>. [Accessed 6. marts 2022].
- [14] R. Baratov, "Hunter mājaslapa," [Online]. Available: <https://hunter.readthedocs.io/en/latest/>. [Accessed 6. marts 2022].
- [15] I. Bicking, "PIP dokumentācija," [Online]. Available: <https://pip.pypa.io/en/stable/>. [Accessed 6. marts 2022].
- [16] C. Lattner, "clang-format koda formatēšanas rīka dokumentācija," LLVM Developer Group, [Online]. Available: <https://clang.llvm.org/docs/ClangFormatStyleOptions.html>. [Accessed 6. marts 2022].
- [17] L. Torvalds, "Git mājaslapa," [Online]. Available: <https://git-scm.com/>. [Accessed 6. marts 2022].
- [18] Atmel, "ATtiny25/45/85 datulapa, dokumentācija," Atmel, [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2586-AVR-8-bit-Microcontroller-ATtiny25-ATtiny45-ATtiny85_Datasheet.pdf. [Accessed 6. marts 2022].
- [19] Adafruit Industries, "adafruit/DHT-sensor-library Github repozitorijs," [Online]. Available: <https://github.com/adafruit/DHT-sensor-library>. [Accessed 6. marts 2022].
- [20] Atmel, "ATtiny24/44/84 datulapa, dokumentācija," Atmel, [Online]. Available: http://ww1.microchip.com/downloads/en/devicedoc/Atmel-7701_Automotive-Microcontrollers-ATtiny24-44-84_Datasheet.pdf. [Accessed 6. marts 2022].

- [21] NXP Semiconductors, "I2C dokumentācija," [Online]. Available: <https://www.nxp.com/docs/en/user-guide/UM10204.pdf>. [Accessed 6. marts 2022].
- [22] NXP Semiconductors, "I2C-bus specification and user manual," in *I2C-bus specification and user manual*, 2021, pp. 1-1.
- [23] SparkFun Electronics, "DHT22 dokumentācija," [Online]. Available: <https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf>. [Accessed 6. marts 2022].
- [24] Aosong Electronics Co.,Ltd, "Digital-output relative humidity & temperature sensor/module DHT22 (DHT22 also named as AM2302)," in *Digital-output relative humidity & temperature sensor/module DHT22 (DHT22 also named as AM2302)* , pp. 2-2.
- [25] Zhiwei Robotics Corp, "Zemes mitruma sensora dokumentācija," [Online]. Available: https://media.digikey.com/pdf/Data%20Sheets/DFRobot%20PDFs/SEN0114_Web.pdf . [Accessed 6. marts 2022].
- [26] O.E.M. Heaters, "Terārija lampas spuldzes jaudas kalkulators," O.E.M. Heaters, [Online]. Available: <https://www.oemheaters.com/topic/enclosure-heating>. [Accessed 6. marts 2022].
- [27] Plaček Pet Product s.r.o., "ReptiPlanet 75W spuldzes dokumentācija," Agentura WhyNOT, [Online]. Available: <http://reptiplanet.pet/portfolio-items/repti-planet-daylight-neodymium/>. [Accessed 6. marts 2022].
- [28] Hyuduo Store, "Vakuuma sūknis, B veids, parametri C veida sūkņa aprakstā," Hyuduo Store, [Online]. Available: https://www.amazon.de/gp/product/B087NSVDFY/ref=ppx_yo_dt_b_asin_title_o03_s00?ie=UTF8&th=1. [Accessed 6. marts 2022].
- [29] R. McCool, "Apache mājaslapa," [Online]. Available: <https://httpd.apache.org/>. [Accessed 6. marts 2022].
- [30] D. R. Hipp, "SQLite mājaslapa," [Online]. Available: <https://www.sqlite.org/index.html>. [Accessed 6. marts 2022].
- [31] D. R. Hipp, "SQLite citāts," [Online]. Available: <https://www.sqlite.org/mostdeployed.html> . [Accessed 6. marts 2022].
- [32] Autodesk, Inc., "Autodesk mājaslapa," Autodesk, Inc., [Online]. Available: <https://www.autodesk.com/>. [Accessed 6. marts 2022].
- [33] POSMASTER, SIA, "Posmaster mājaslapa," POSMASTER, SIA, [Online]. Available: <https://posmaster.lv/>. [Accessed 6. marts 2022].
- [34] EasyEDA , "EasyEDA mājaslapa," EasyEDA , [Online]. Available: <https://easyeda.com/>. [Accessed 6. marts 2022].
- [35] JLCPCB, "JLCPCB mājaslapa," JLCPCB, [Online]. Available: https://jlcpcb.com/VGB?gclid=CjwKCAiA1JGRBhBSEiwAxXblwaw5k_Up_0jkrdk-AOxGfdUn21fQ9HuEKGjXnNgBUMbhJBUsH0BDDRoCayQQAvD_BwE. [Accessed 6. marts 2022].
- [36] Plaček Pet Product s.r.o., "ReptiPlanet terārija mājaslapa," Agentura WhyNOT, [Online]. Available: <http://reptiplanet.pet/portfolio-items/repti-planet-buildable-terrarium-914x457x32cm/?portfolioCats=84>. [Accessed 12. maijs 2022].
- [37] V. Pai, "UDP Heartbeat apraksts," 4. maijs 2004. [Online]. Available: https://www.usenix.org/legacy/publications/library/proceedings/usenix04/tech/general/full_papers/wang/wang_html/node5.html. [Accessed 2022 maijs 13.].

- [38] Philips Semiconductors, "Bi-directional level shifter for I²C-bus," [Online]. Available: <https://cdn-shop.adafruit.com/datasheets/an97055.pdf>. [Accessed 17. maijs 2022].
- [39] MartinsSmirnovs, "Anticarium/Shared_Types Github repozitorijss," [Online]. Available: https://github.com/Anticarium/Shared_Types. [Accessed 6. marts 2022].
- [40] GJLay, "AVR-GCC dokumentācijas mājaslapa," [Online]. Available: <https://gcc.gnu.org/wiki/avr-gcc>. [Accessed 10. maijs 2022].
- [41] CAMCEM, S.A. de C.V., "GCC mājaslapa," CAMCEM, S.A. de C.V., [Online]. Available: <https://gcc.gnu.org/>. [Accessed 10. maijs 2022].
- [42] I. o. E. a. E. Engineers, "IEEE 754 standarts," [Online]. Available: <https://irem.univ-reunion.fr/IMG/pdf/ieee-754-2008.pdf>. [Accessed 10. maijs 2022].
- [43] CAMCEM, S.A. de C.V., "GCC datu tipu izmēri," [Online]. Available: <https://gcc.gnu.org/onlinedocs/gccint/Type-Layout.html>. [Accessed 10. maijs 2022].
- [44] G. Fairhurst, "UDP apraksts," 19. novembris 2008. [Online]. Available: <https://erg.abdn.ac.uk/users/gorry/course/inet-pages/udp.html#:~:text=A%20UDP%20datagram%20is%20carried,packets%20usually%20requires%20IP%20fragmentation..> [Accessed 13 maijs 2022].
- [45] MartinsSmirnovs, "Anticarium_Pi setup_vm.sh skripts," [Online]. Available: https://github.com/Anticarium/Anticarium_Pi/blob/master/setup/setup_vm.sh. [Accessed 26. maijs 2022].
- [46] MartinsSmirnovs, "Anticariu_Pi/pi.py skripts," [Online]. Available: https://github.com/Anticarium/Anticarium_Pi/blob/master/pi.py. [Accessed 26. maijs 2022].
- [47] MartinsSmirnovs, "Anticarium_Pi/setup/setup_rpi.sh skripts," [Online]. Available: https://github.com/Anticarium/Anticarium_Pi/blob/master/setup/setup_rpi.sh. [Accessed 26. maijs 2022].
- [48] MartinsSmirnovs, "Anticarium WEB setup_web.sh skripts," [Online]. Available: https://github.com/Anticarium/Anticarium_Web/blob/main/setup_web.sh. [Accessed 26. maijs 2022].
- [49] A. Daga, "AVR mikrokontrolieru apskats," [Online]. Available: <https://www.engineersgarage.com/avr-microcontroller-all-you-need-to-know-part-1-46/>. [Accessed 10. maijs 2022].
- [50] F. John, "Raksts "Kas ir primitīvie mainīgie?"," [Online]. Available: <https://www.tutorialspoint.com/What-are-primitive-data-type-in-Cplusplus#:~:text=A%20primitive%20type%20is%20a,the%20following%20primitive%20data%20types%20E2%88%92>. [Accessed 10. maijs 2022].

PIELIKUMI

1. pielikums

Anticarium Desktop lietotnes galvenā skata skice

Home

Temperature:
23.4 °C

Moisture:
87 %

Humidity:
50 %

Custom:

Save



23.4°C



85%



70%

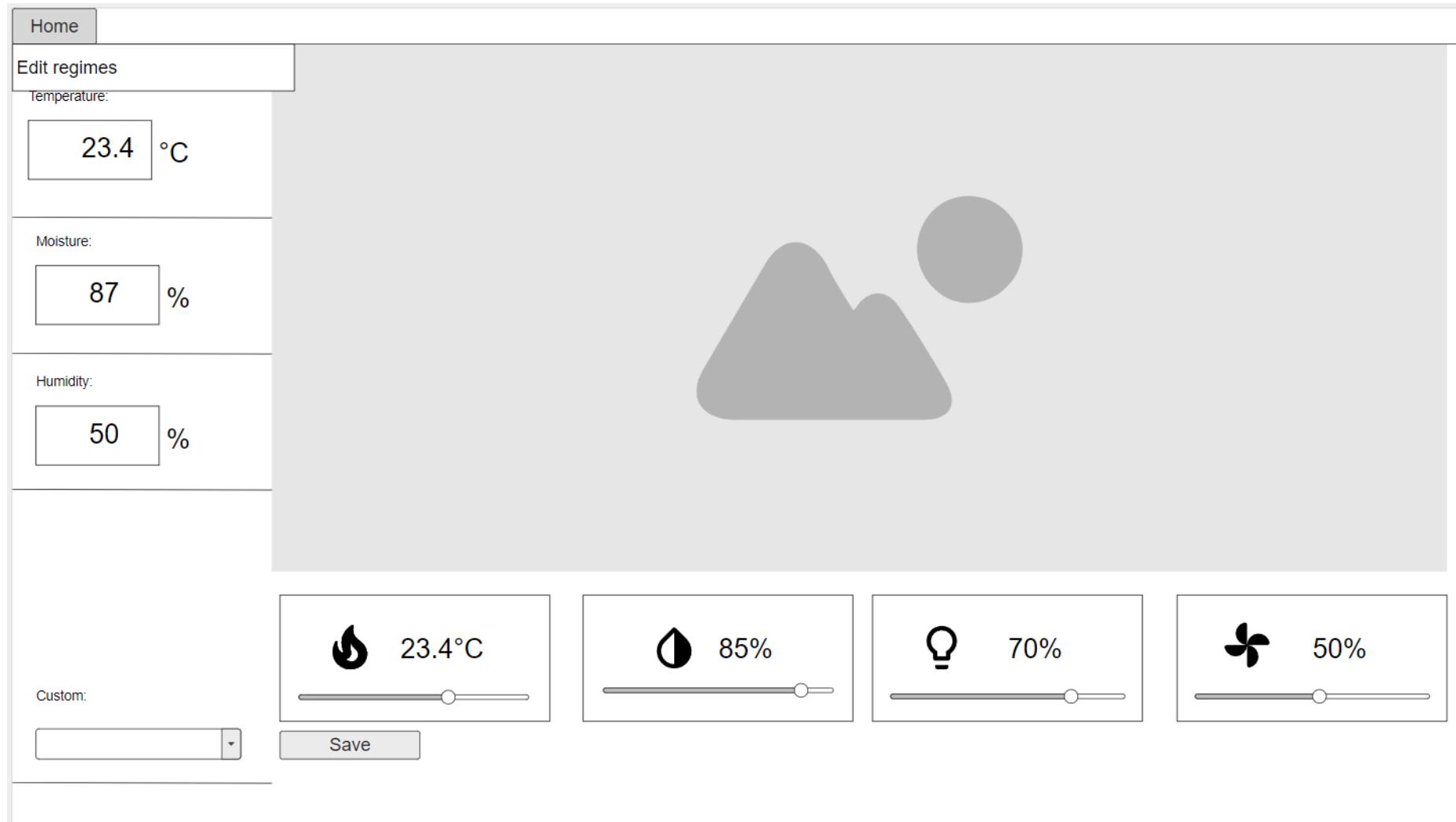


50%



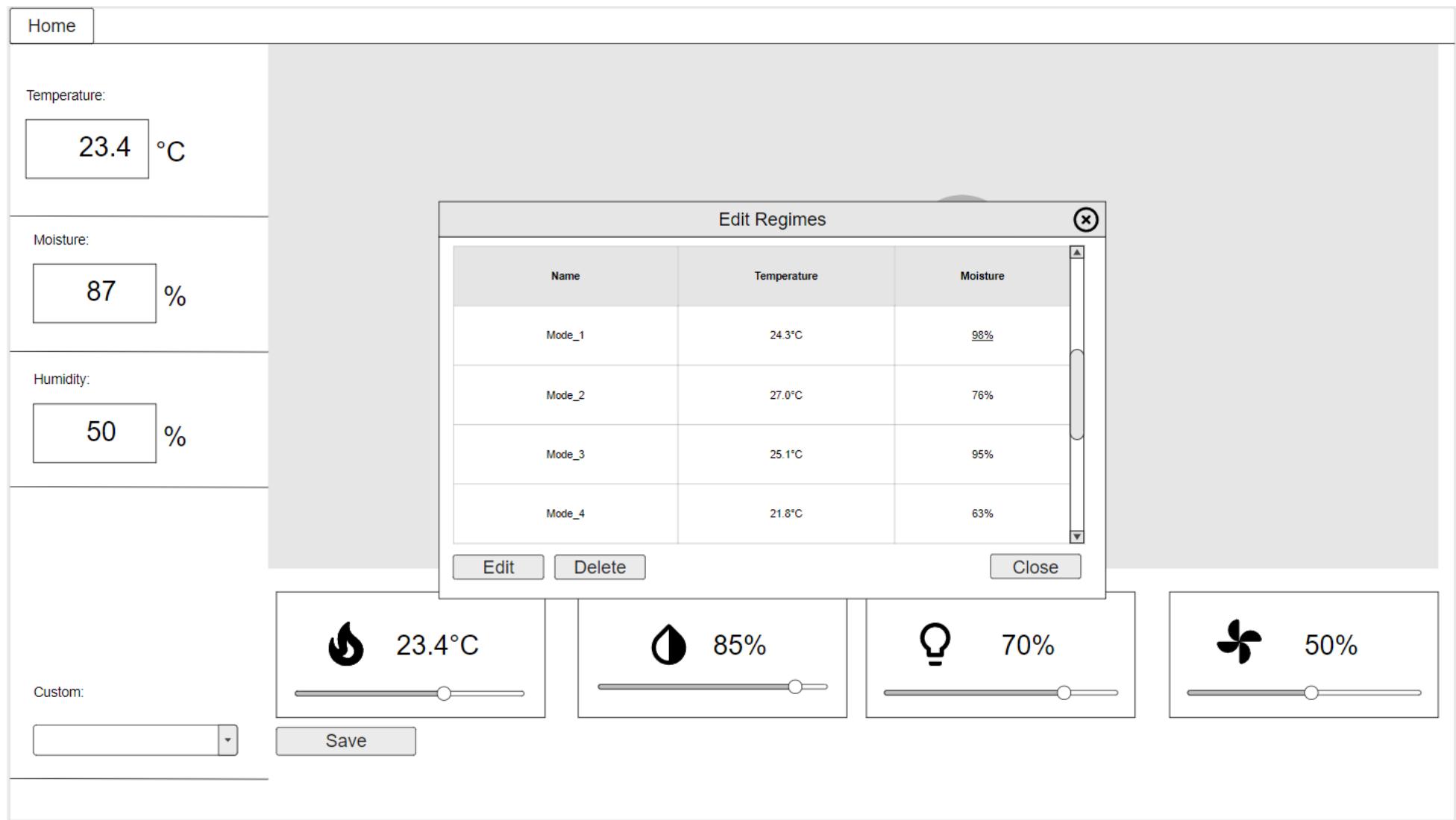
2. pielikums

Anticarium Desktop lietotnes galvenā skata skice ar uzklikšķinātu “Home” sadaļu



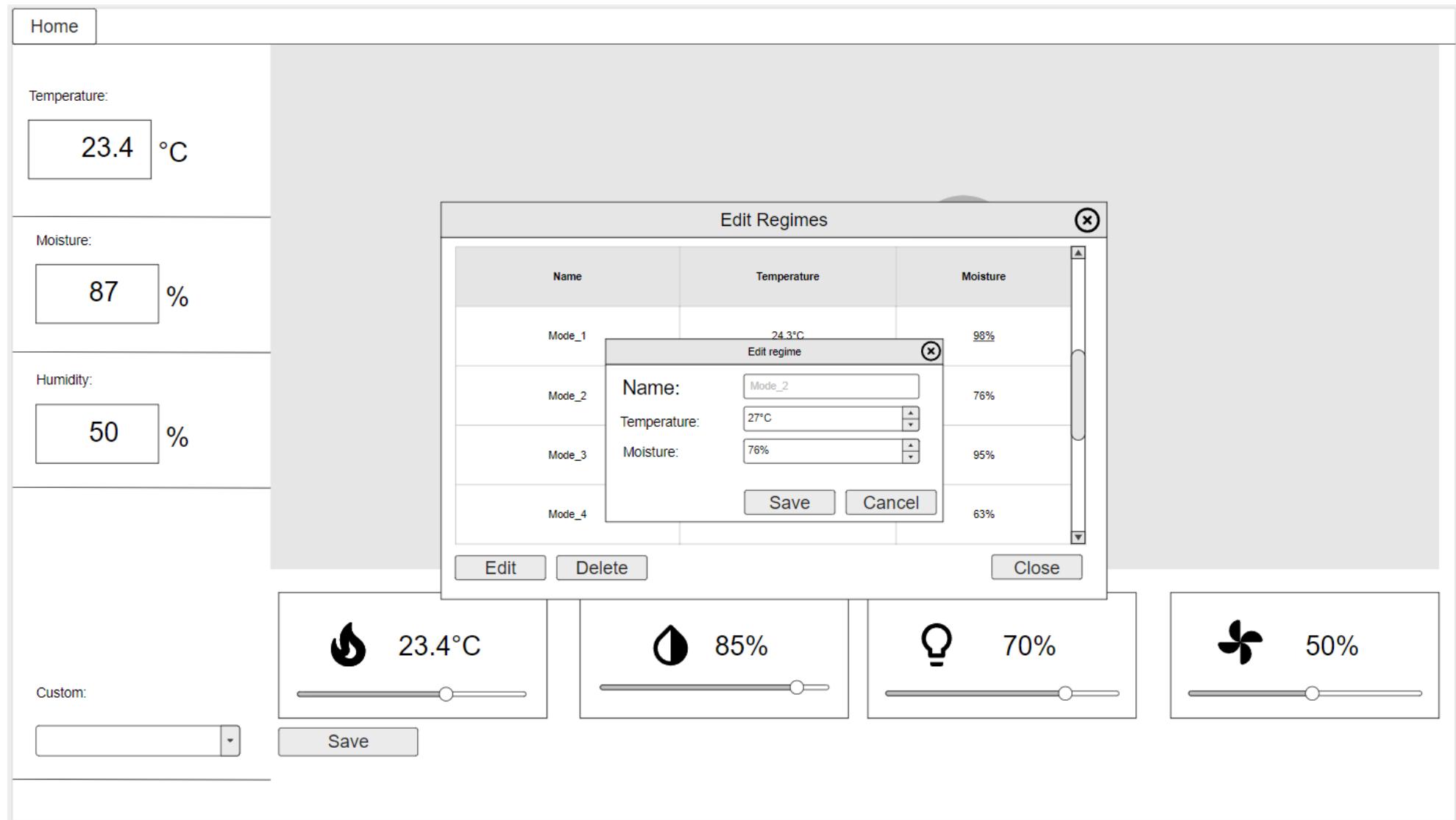
3. pielikums

Režīmu pārvaldīšanas logs, kas atvērās, kad tika izvēlēta "Edit Regimes" apakšizvēlnē no "Home" sadaļas

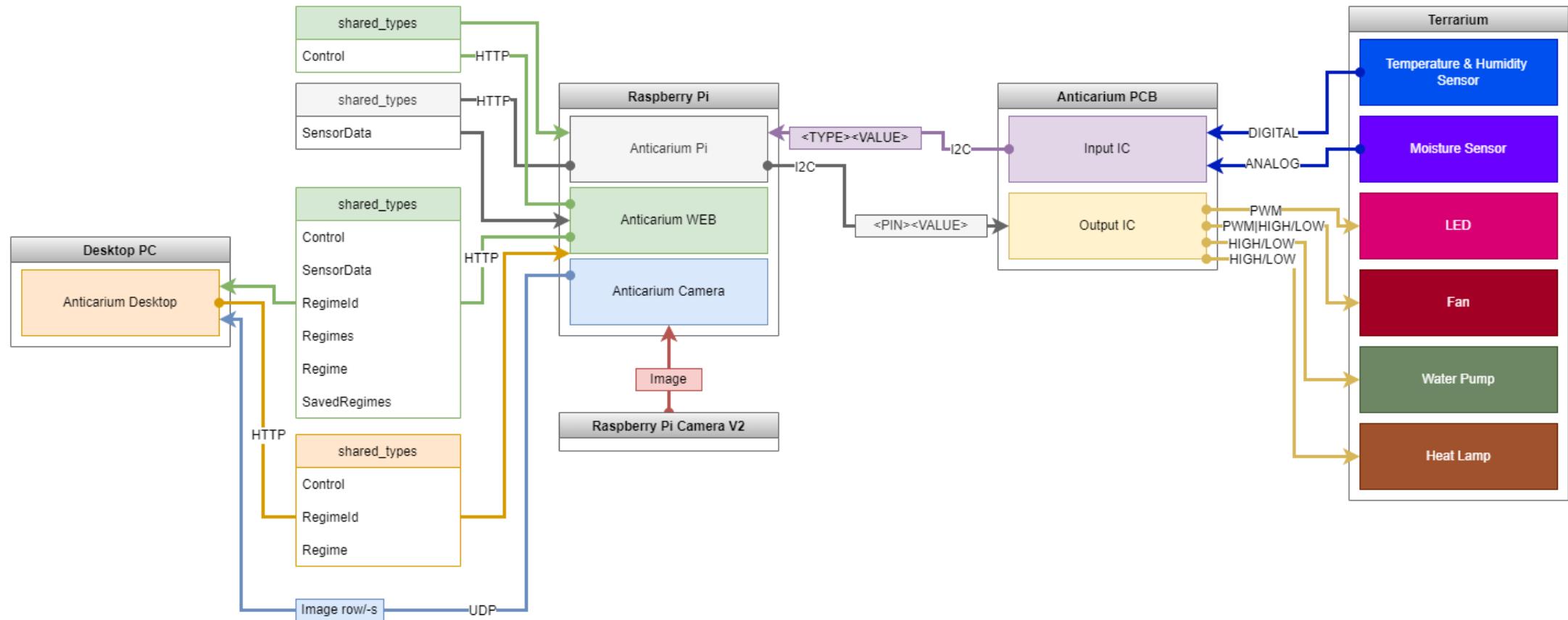


4. pielikums

Režīma redīgēšanas logs, kas atvērās, kad tika uzklikšķināts uz “Edit” pogas iekš režīmu pārvaldīšanas loga

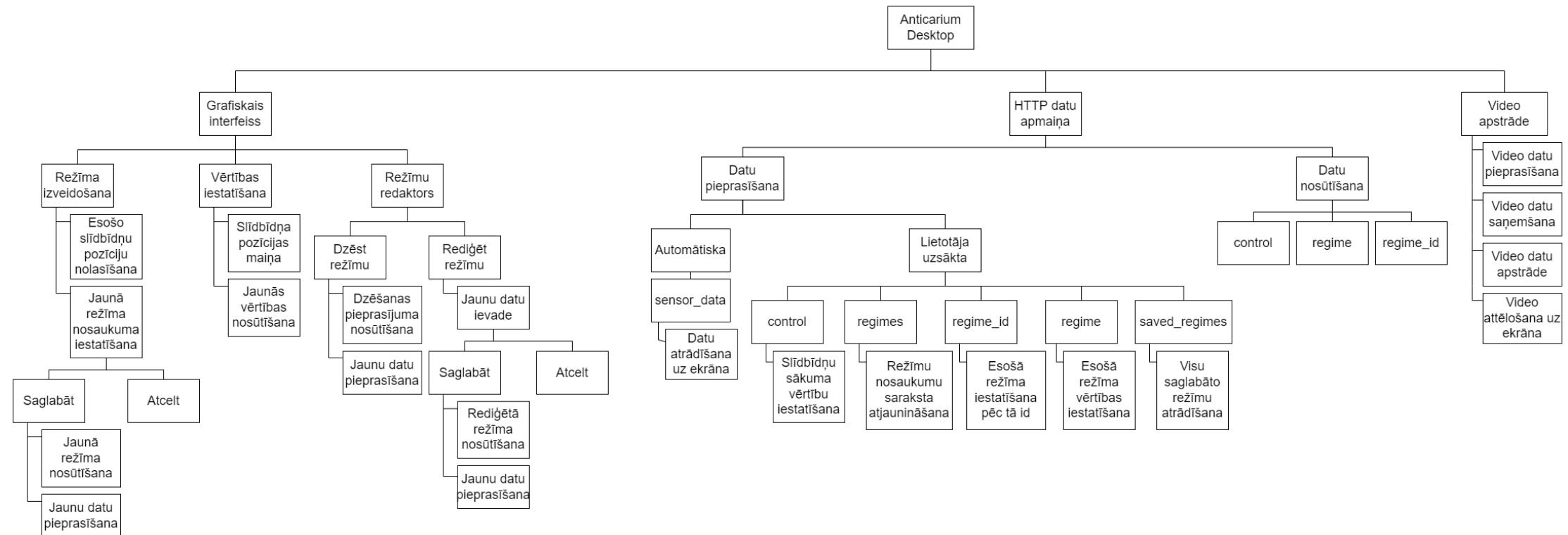


Anticarium projekta arhitektūras shēma

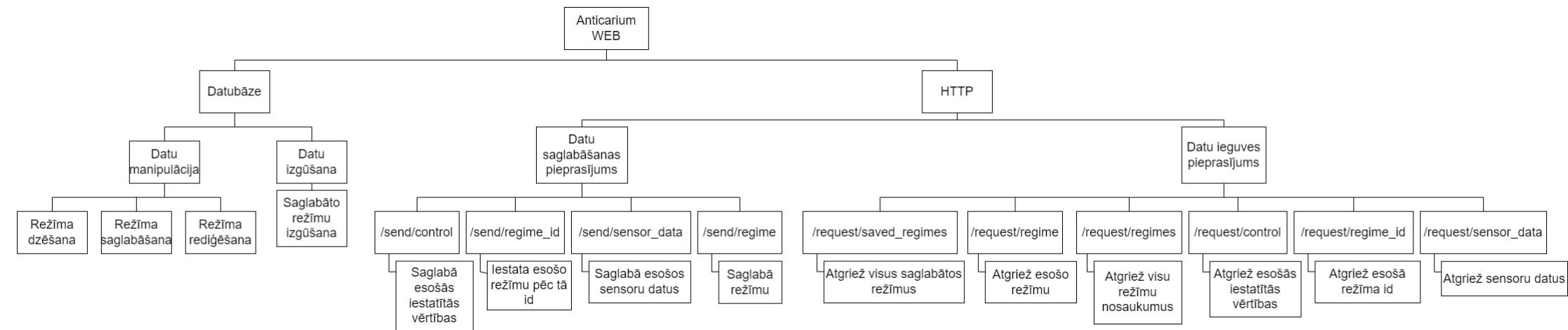


6. pielikums

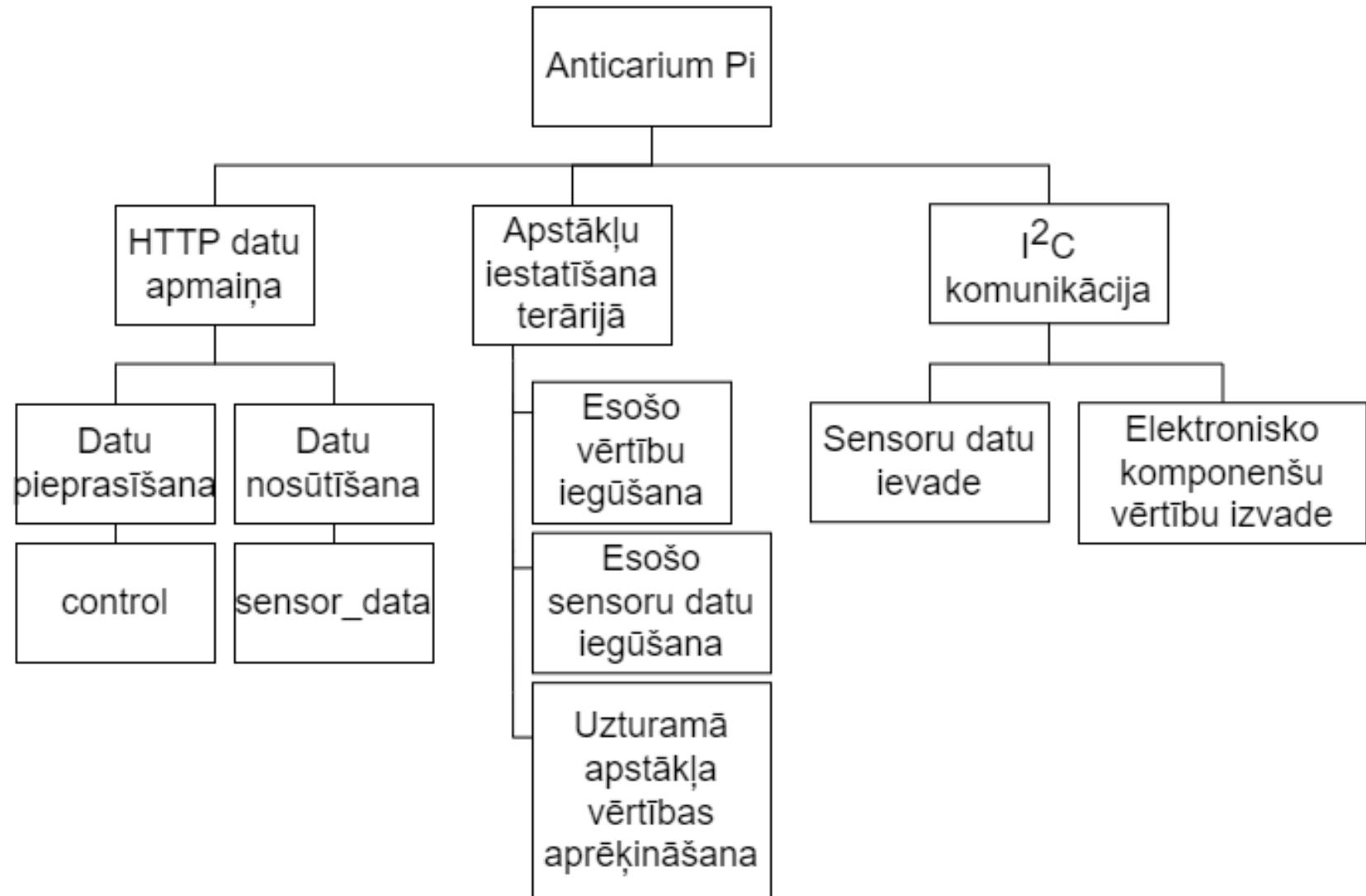
Anticarium Desktop funkcionālās dekompozīcijas diagramma



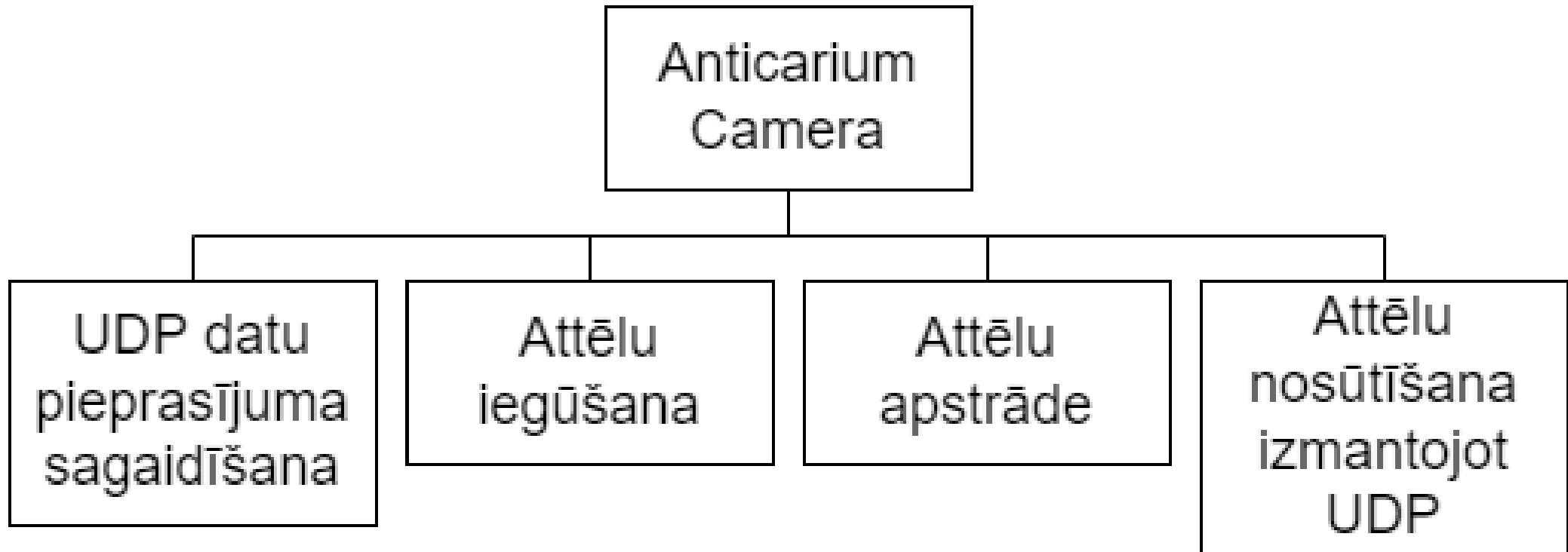
Anticarium WEB funkcionalās dekompozīcijas diagramma



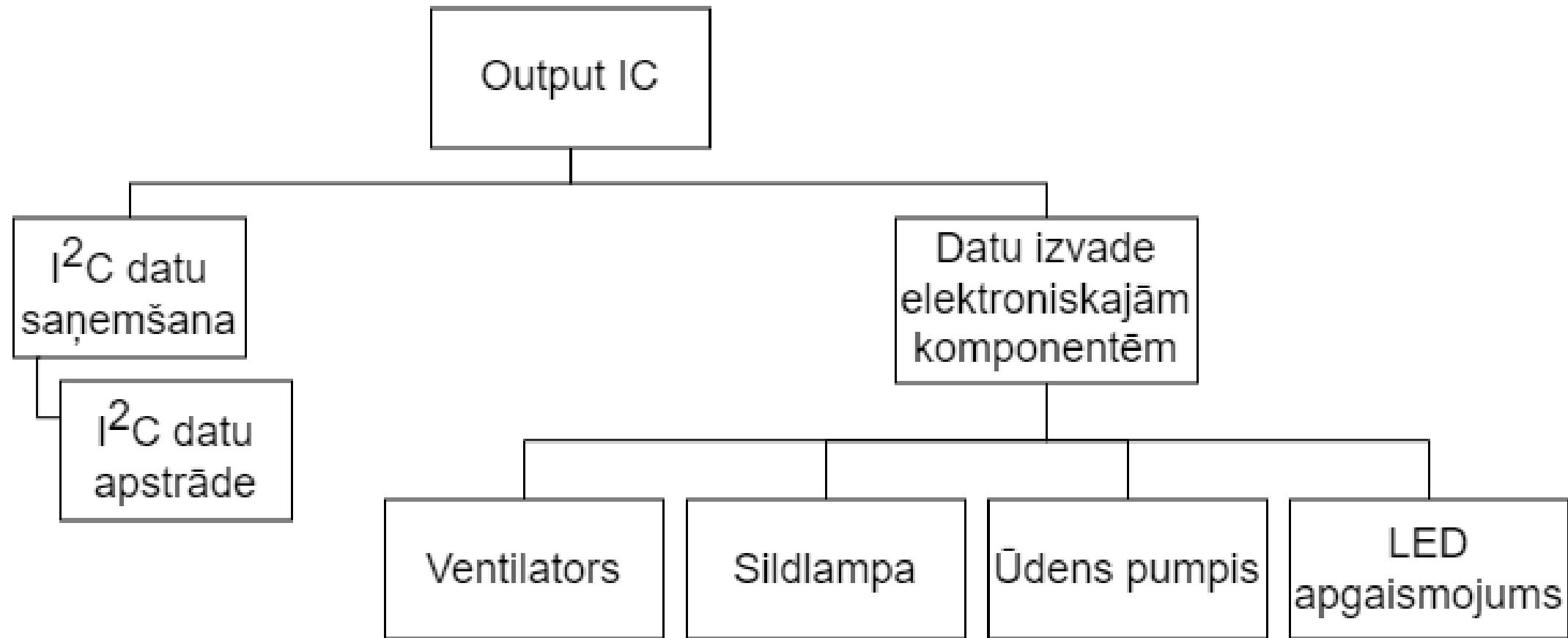
Anticarium Pi funkcionālās dekompozīcijas diagramma



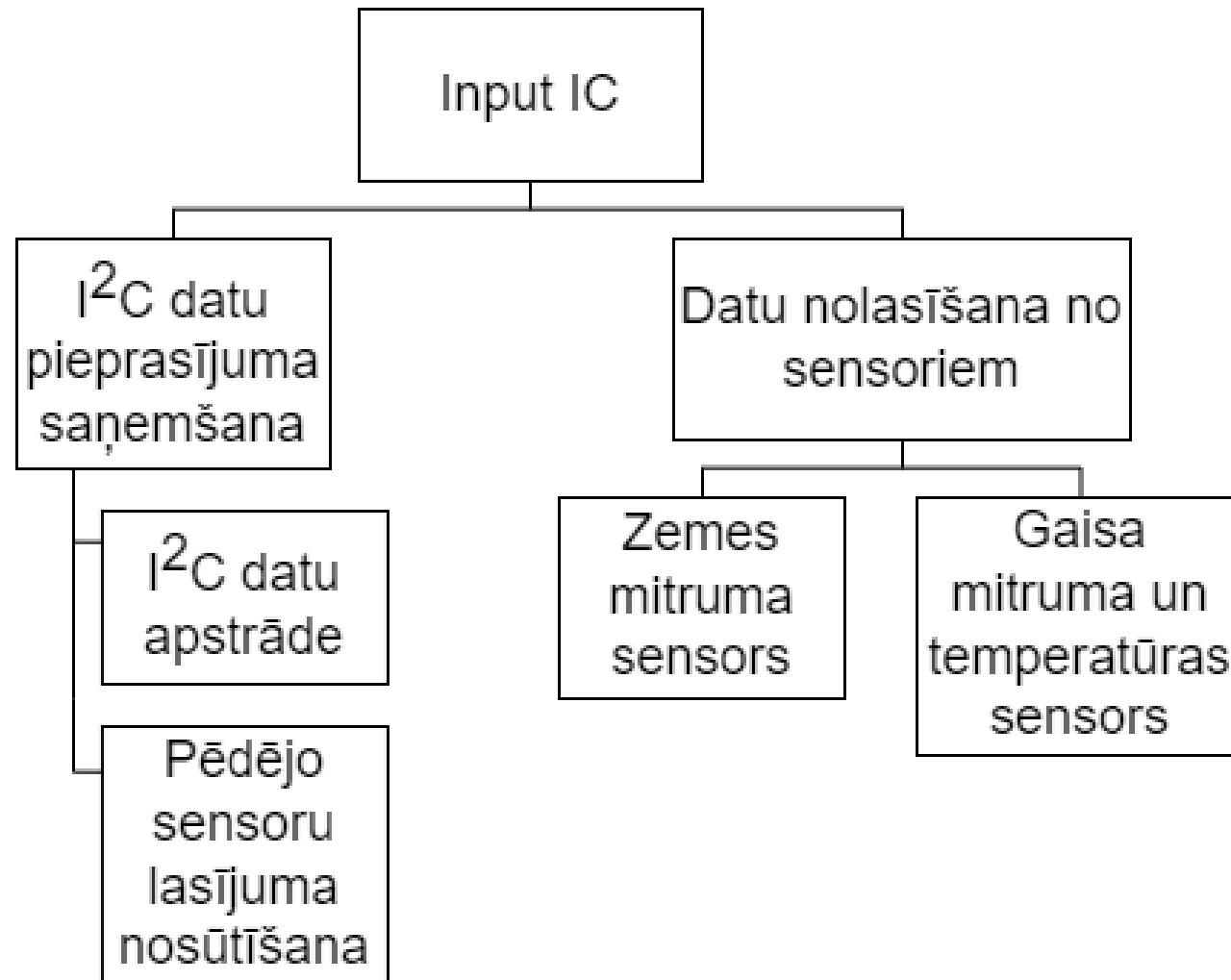
Anticarium Camera funkcionālās dekompozīcijas diagramma



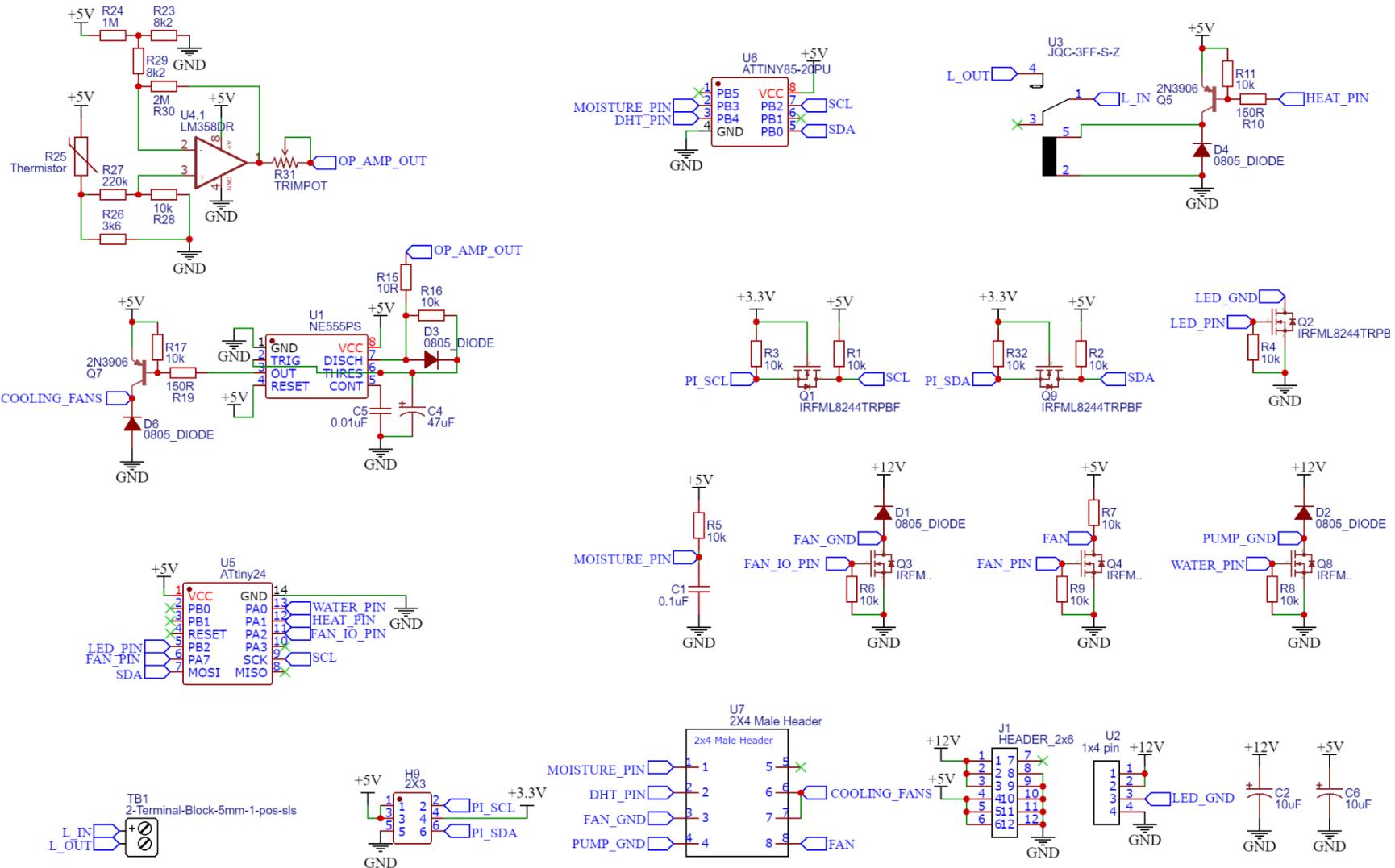
Output IC funkcionālās dekompozīcijas diagramma



Input IC funkcionalās dekompozīcijas diagramma

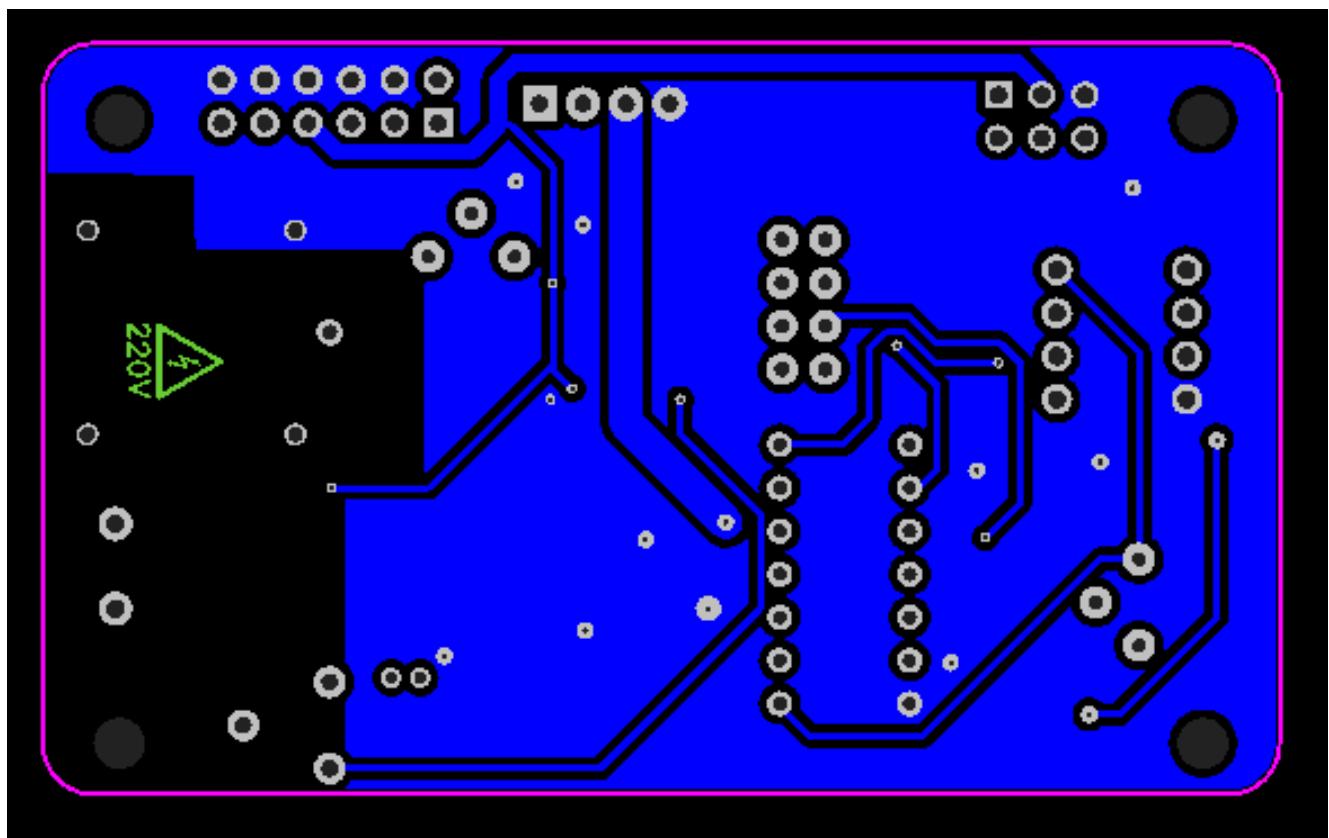
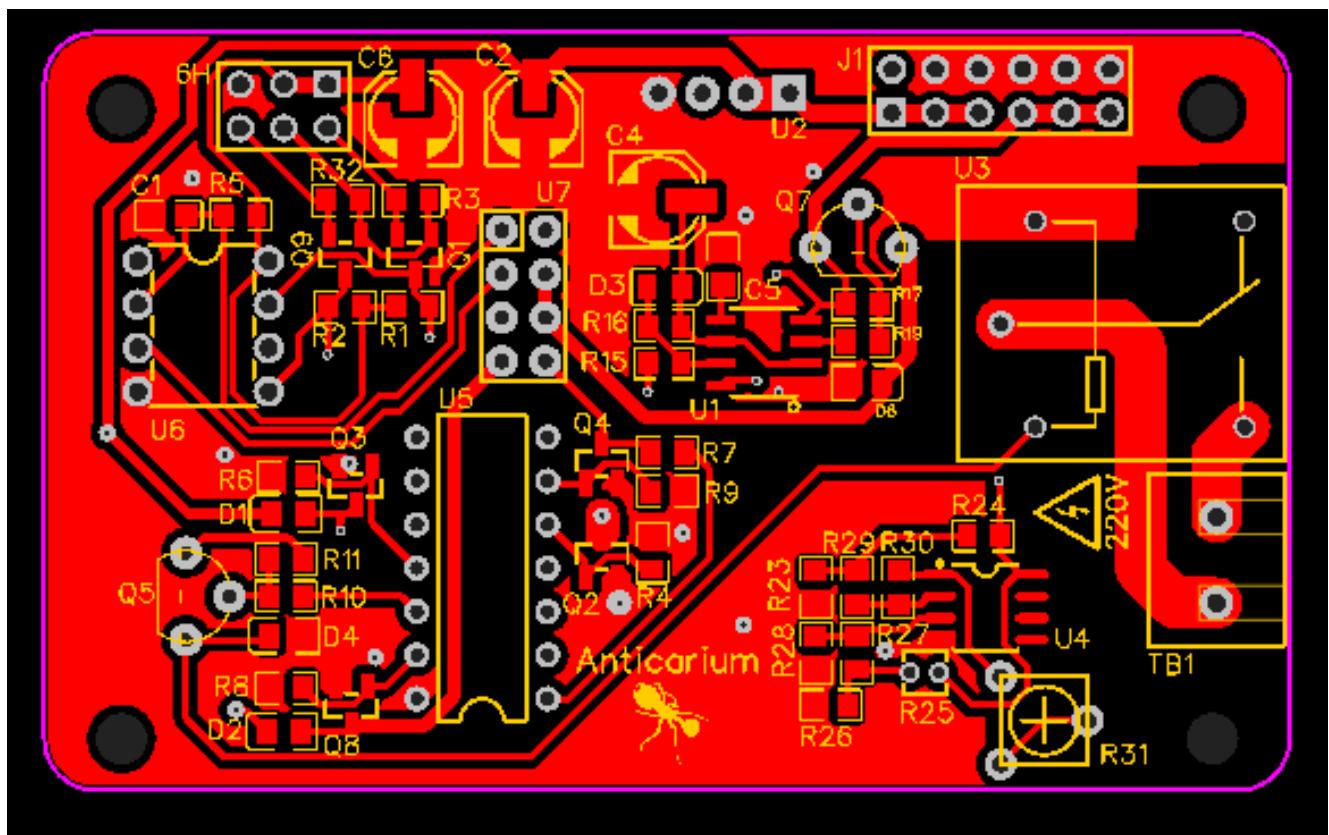


Anticarium PCB elektroniskās shēmas rasējums



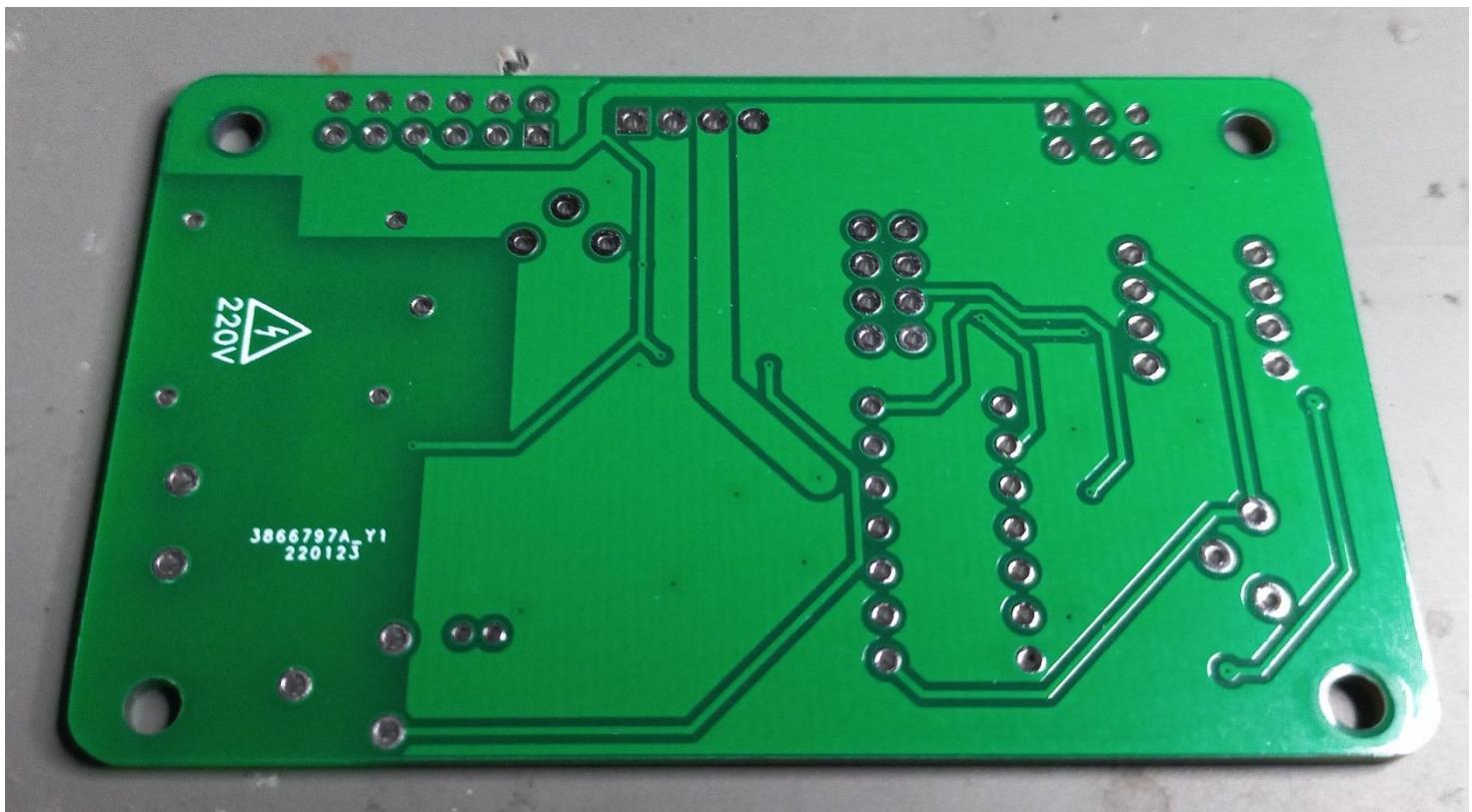
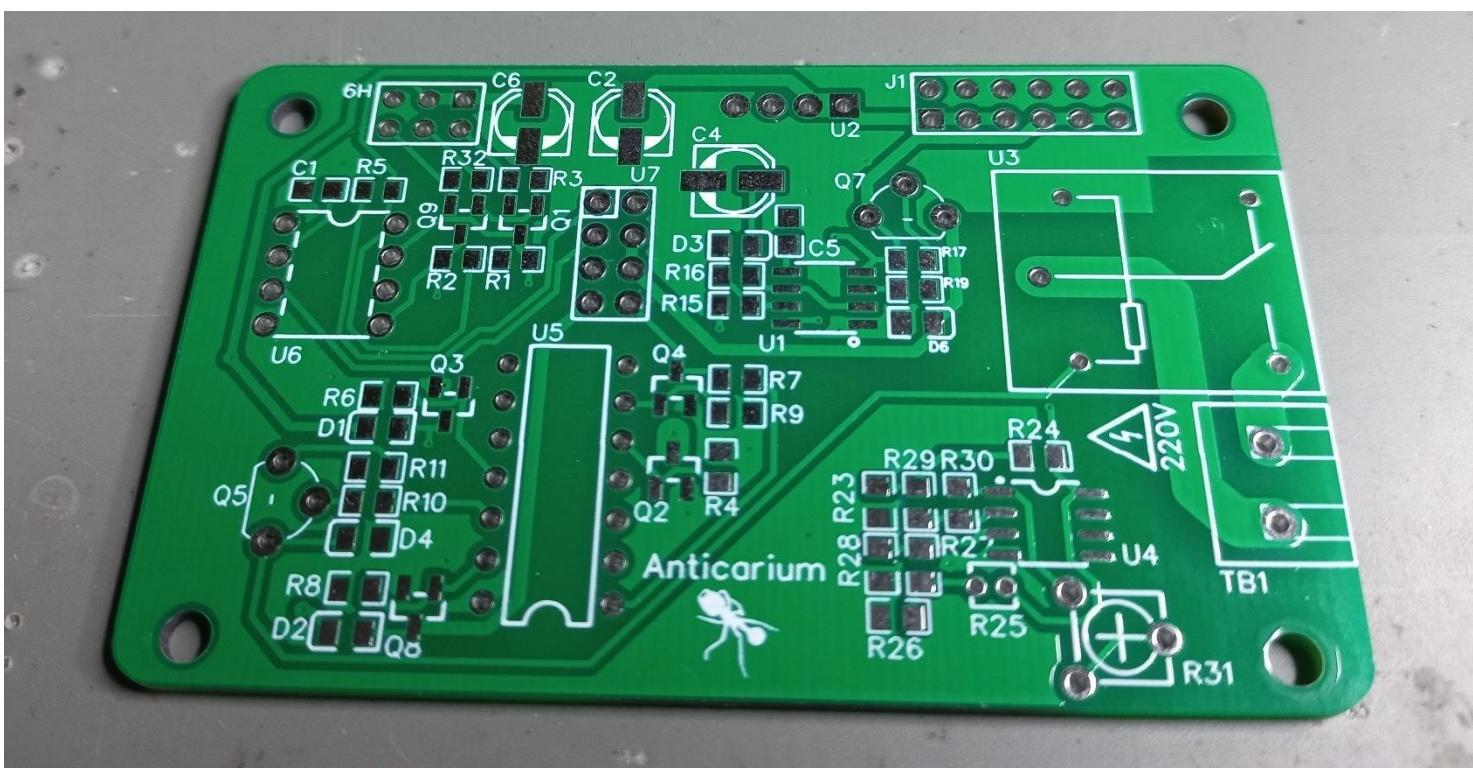
13. pielikums

Anticarium PCB rasējumi no abām plates pusēm



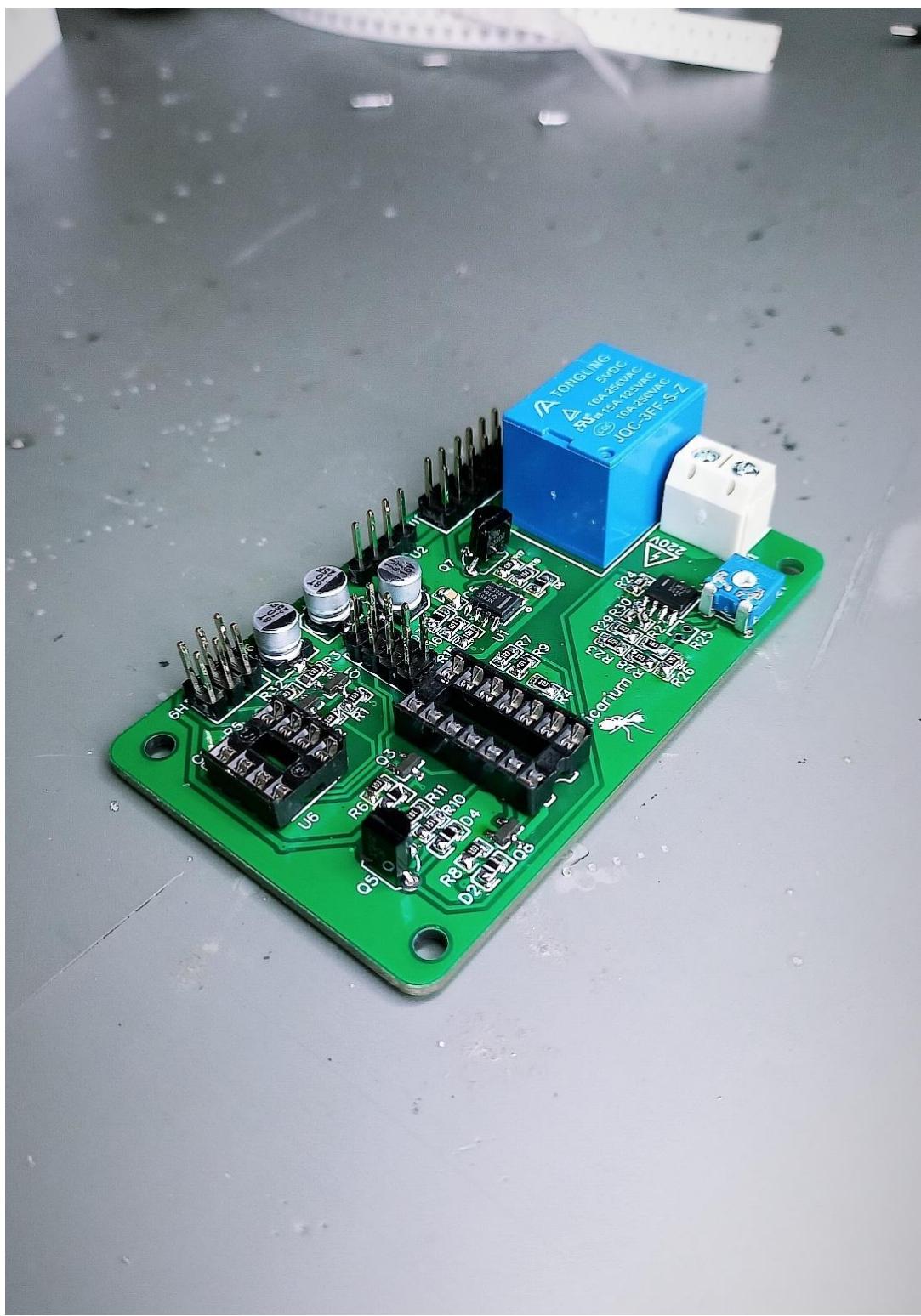
14. pielikums

Anticarium PCB no abām pusēm



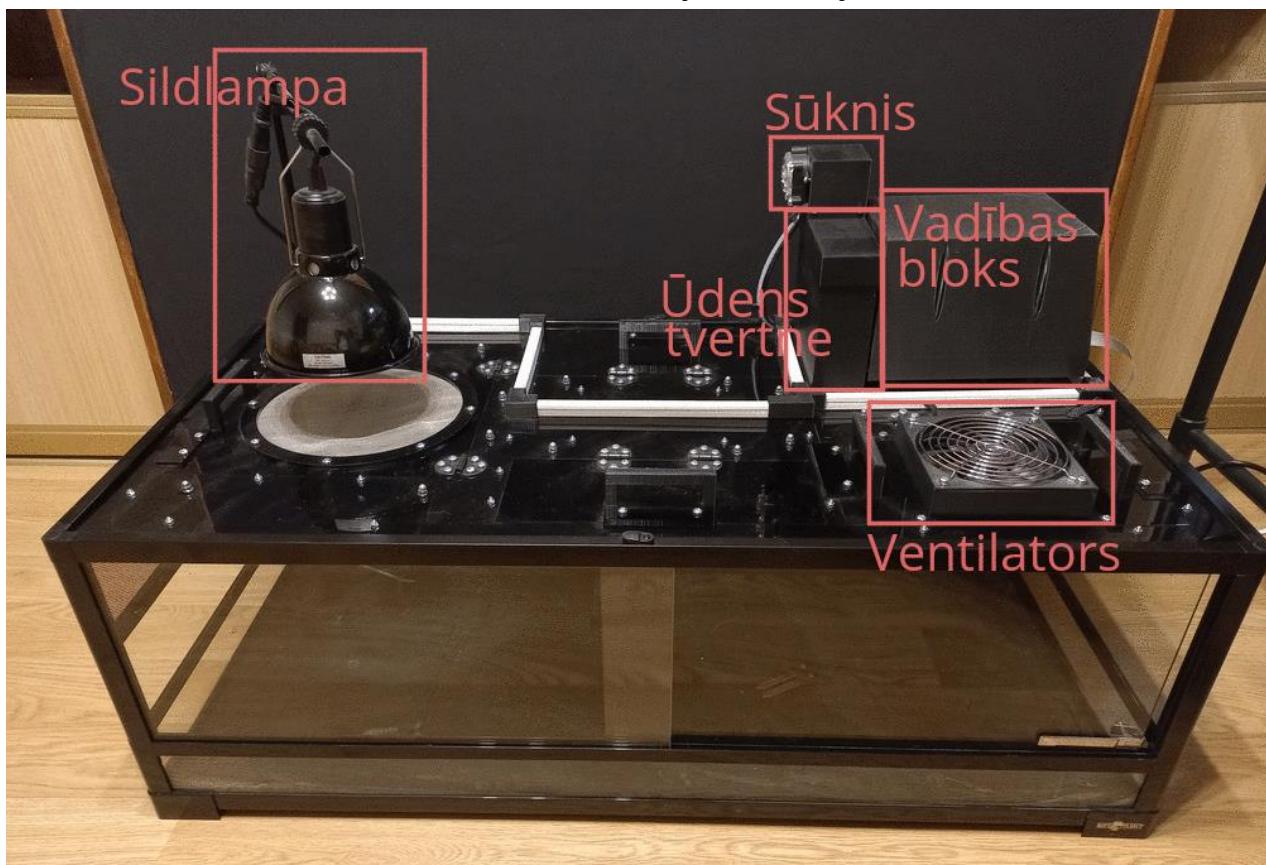
15. pielikums

Anticarium PCB salodētā veidā



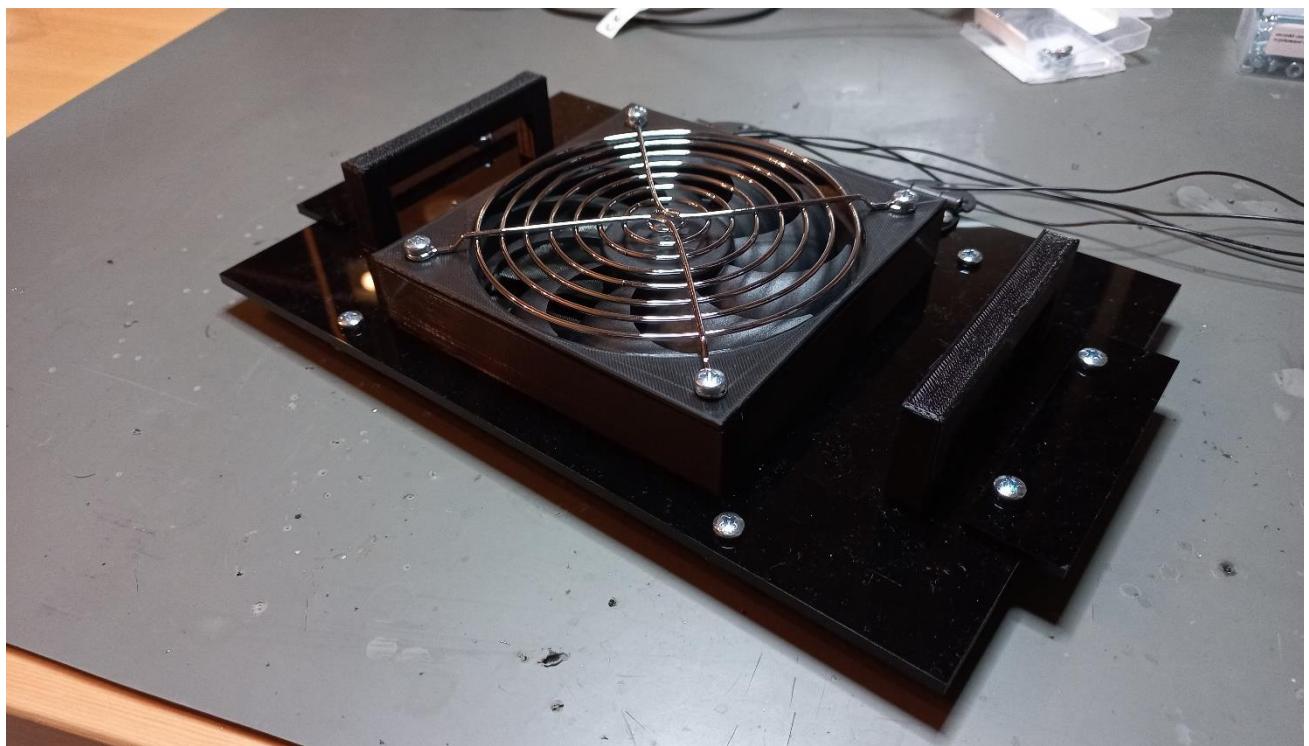
16. pielikums

Anticarium terārijs ar skaidrojumiem



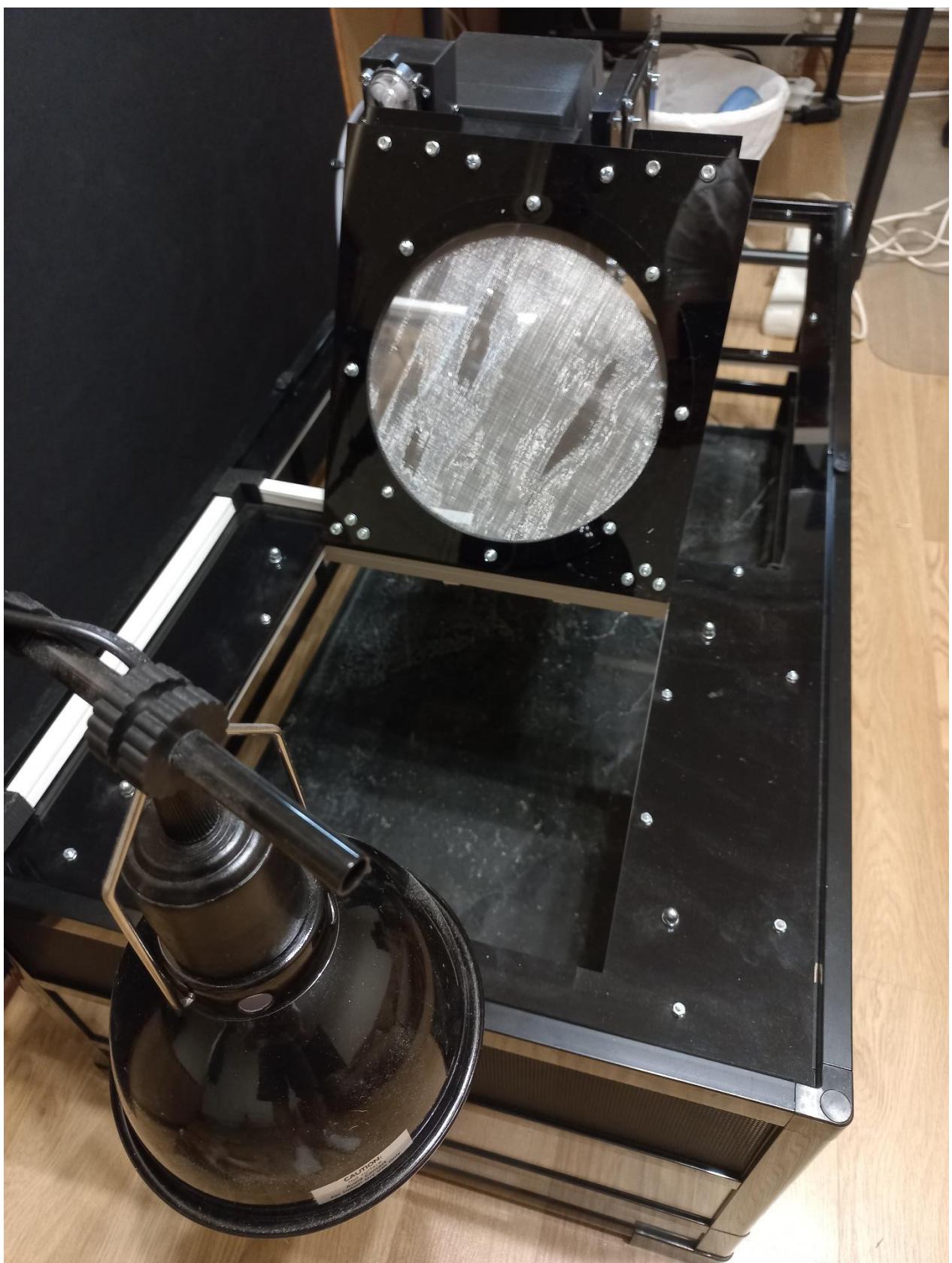
17. pielikums

Lūka ar iebūvētu ventilatoru



18. pielikums

Sildlampas lūka atvērta

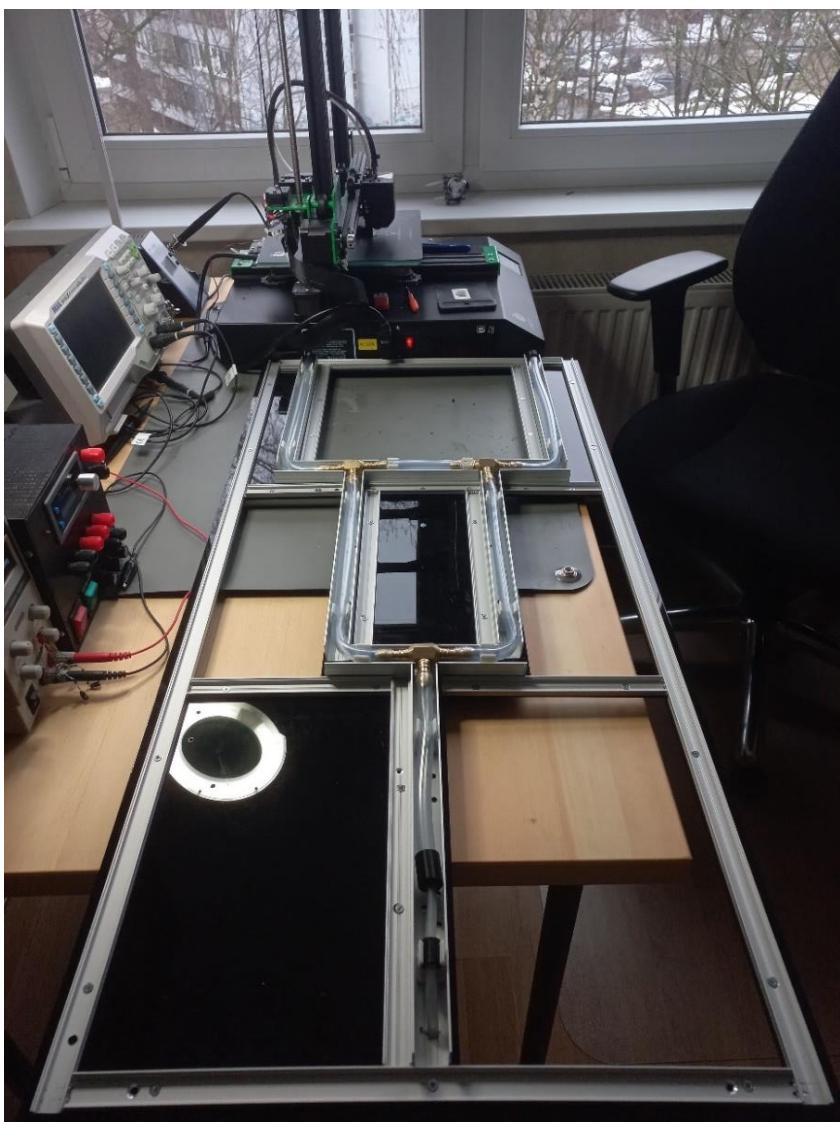


19. pielikums

Vāka apakša ar LED apgaismojumu

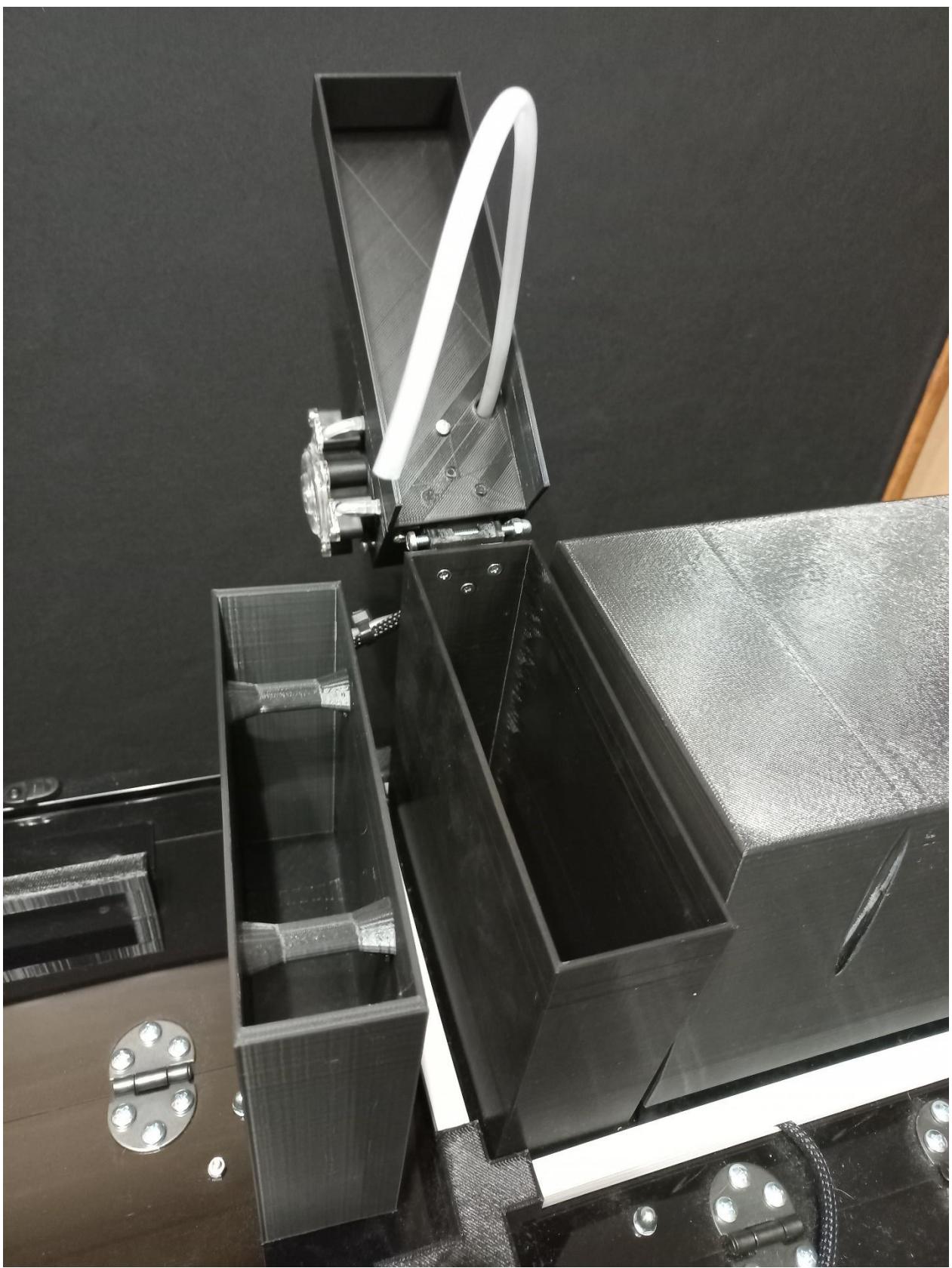


Vāka apakša ar ūdens caurulēm



20. pielikums

Atvērta ūdens tvertne ar izņemtu ieliktni



21. pielikums

Terārija kameras korpusa daļa izņemta kopā ar kameru

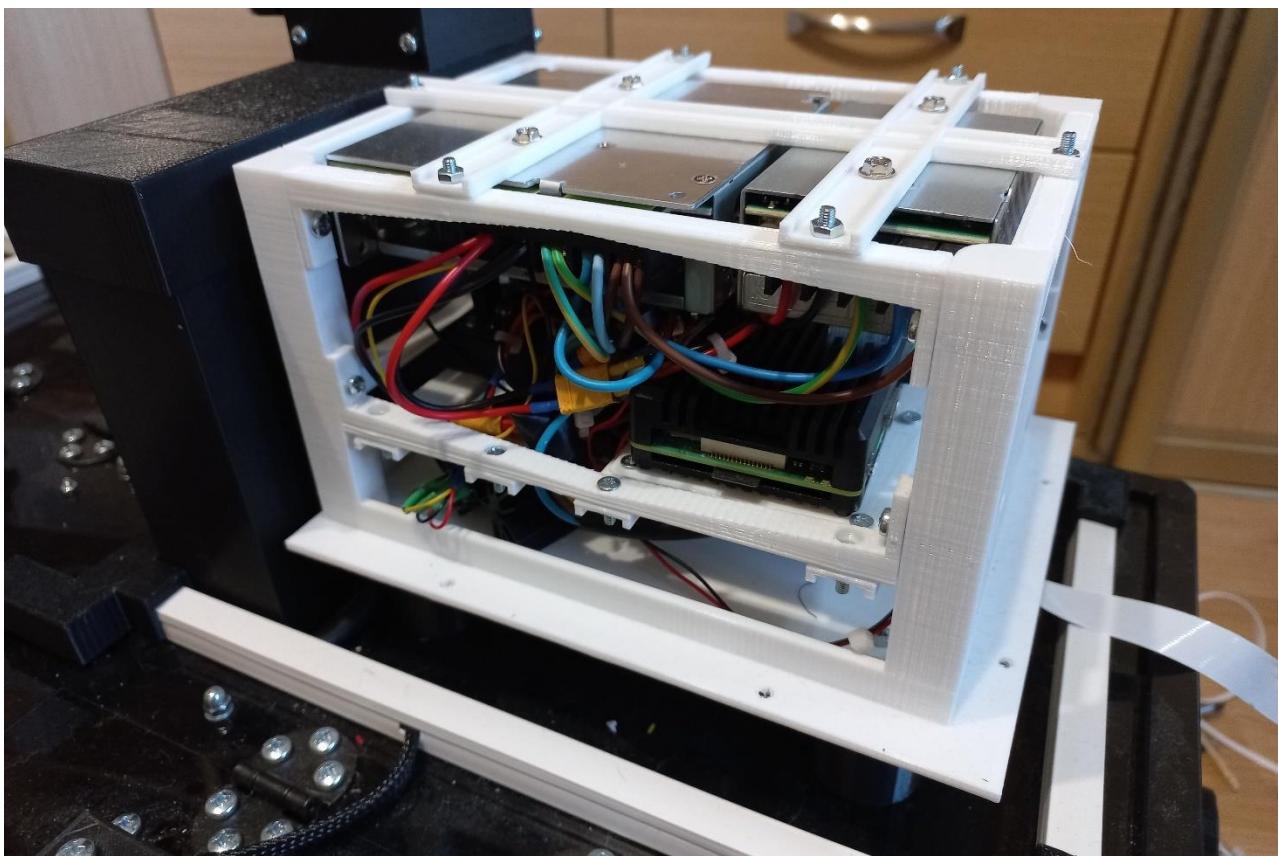


Terārija kamera tās korpusā



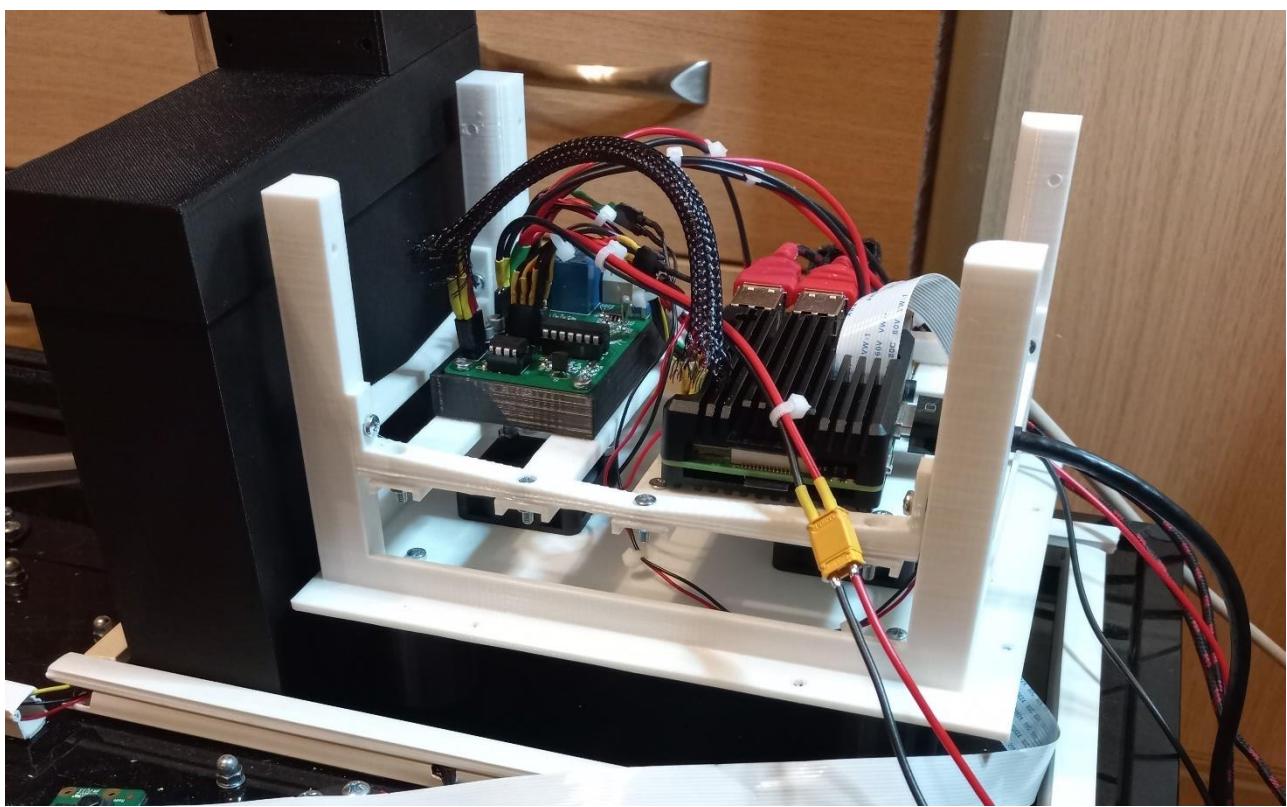
22. pielikums

Terārija vadības bloks no iekšas

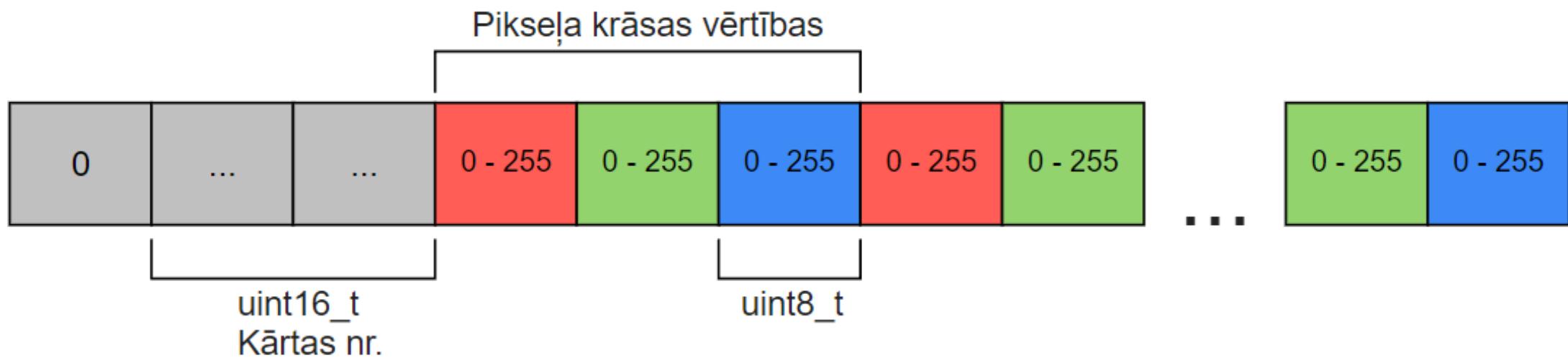


23. pielikums

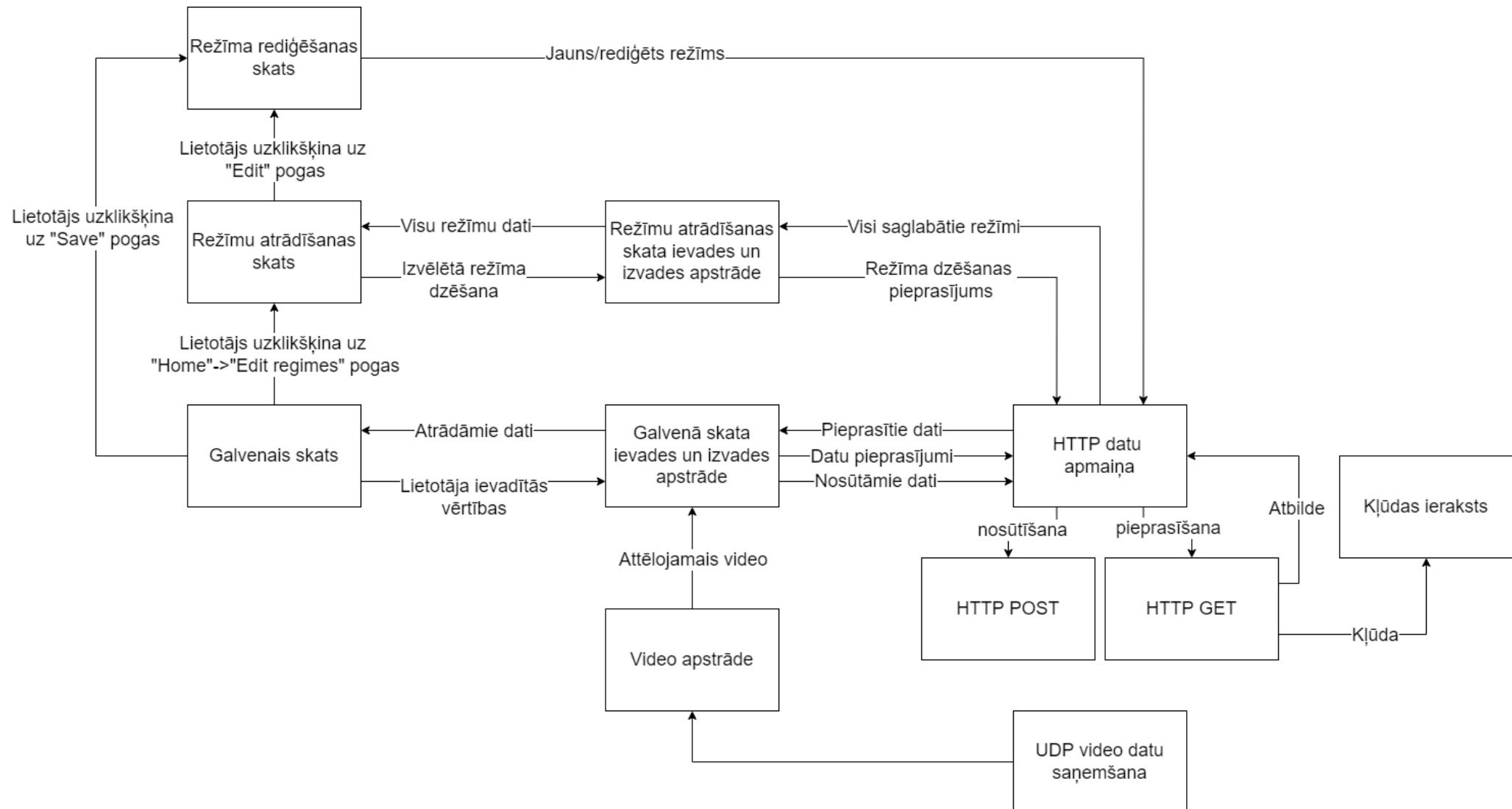
Terārija vadības bloks no iekšas, bez pievienota barošanas bloka



Nosūtāmās attēla rindīņas struktūra

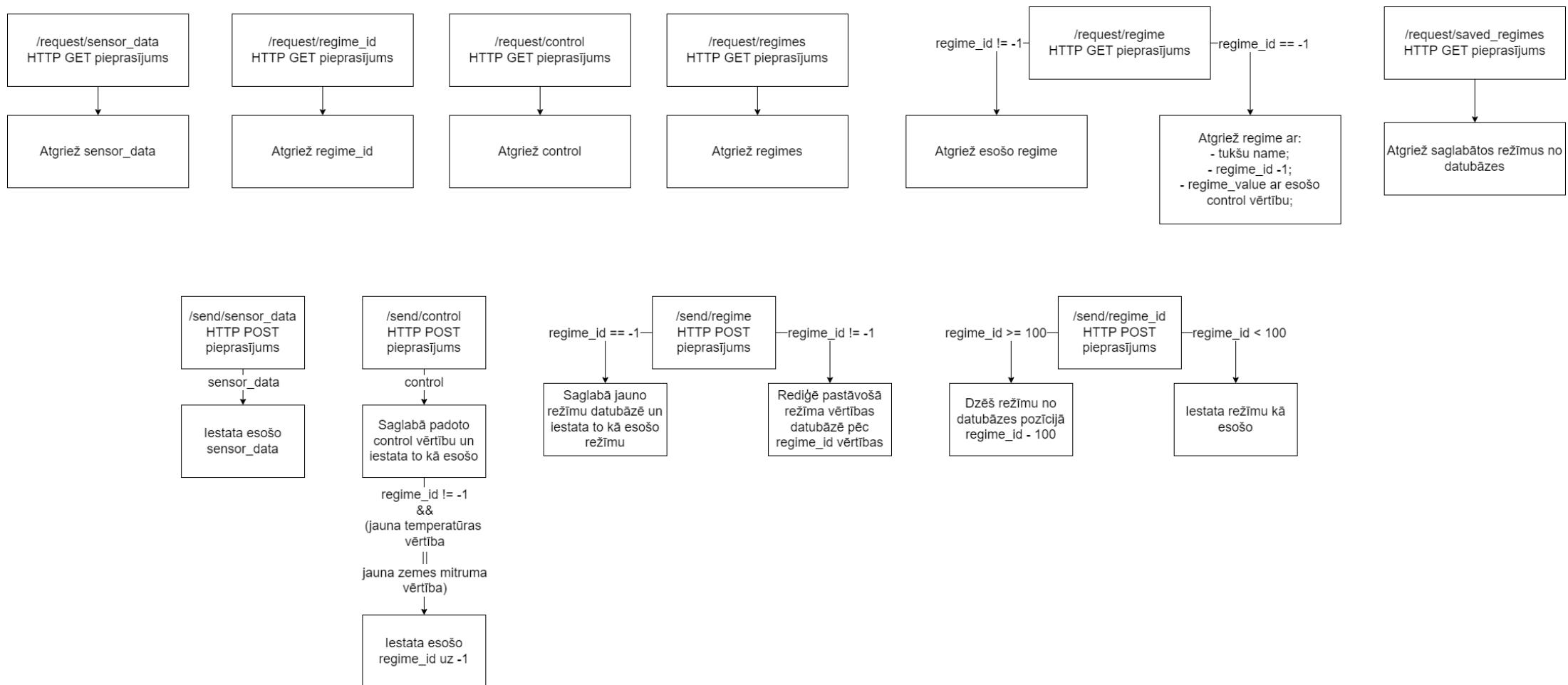


Anticarium Desktop datu plūsmu diagramma

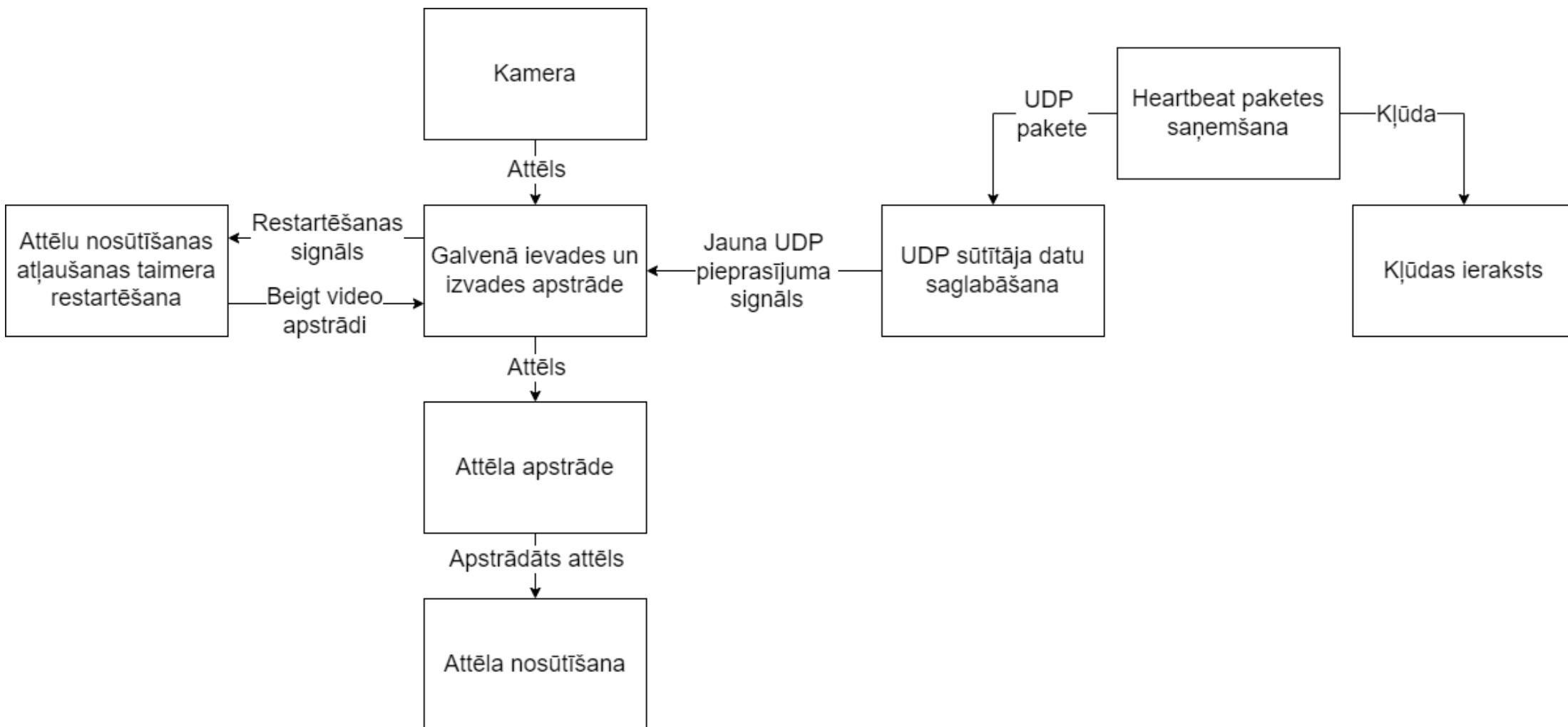


26. pielikums

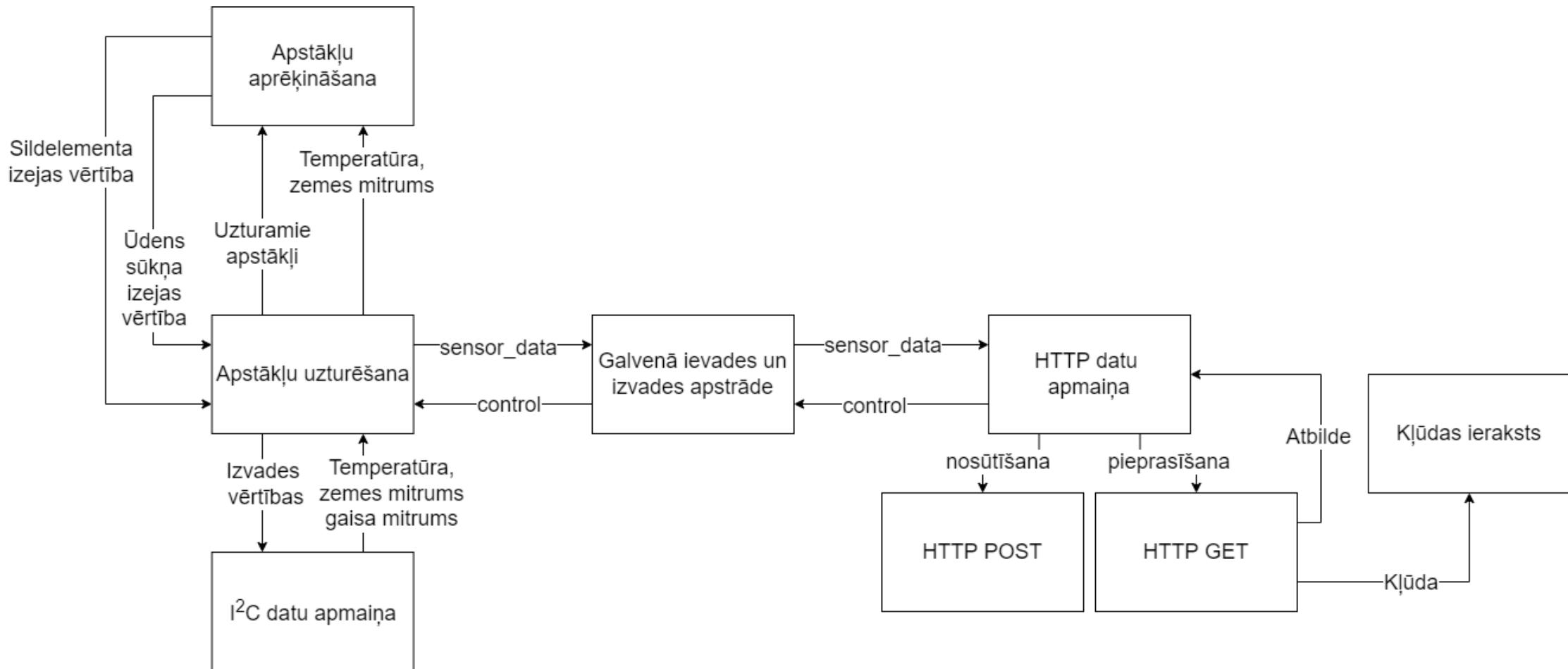
Anticarium WEB datu plūsmu diagramma



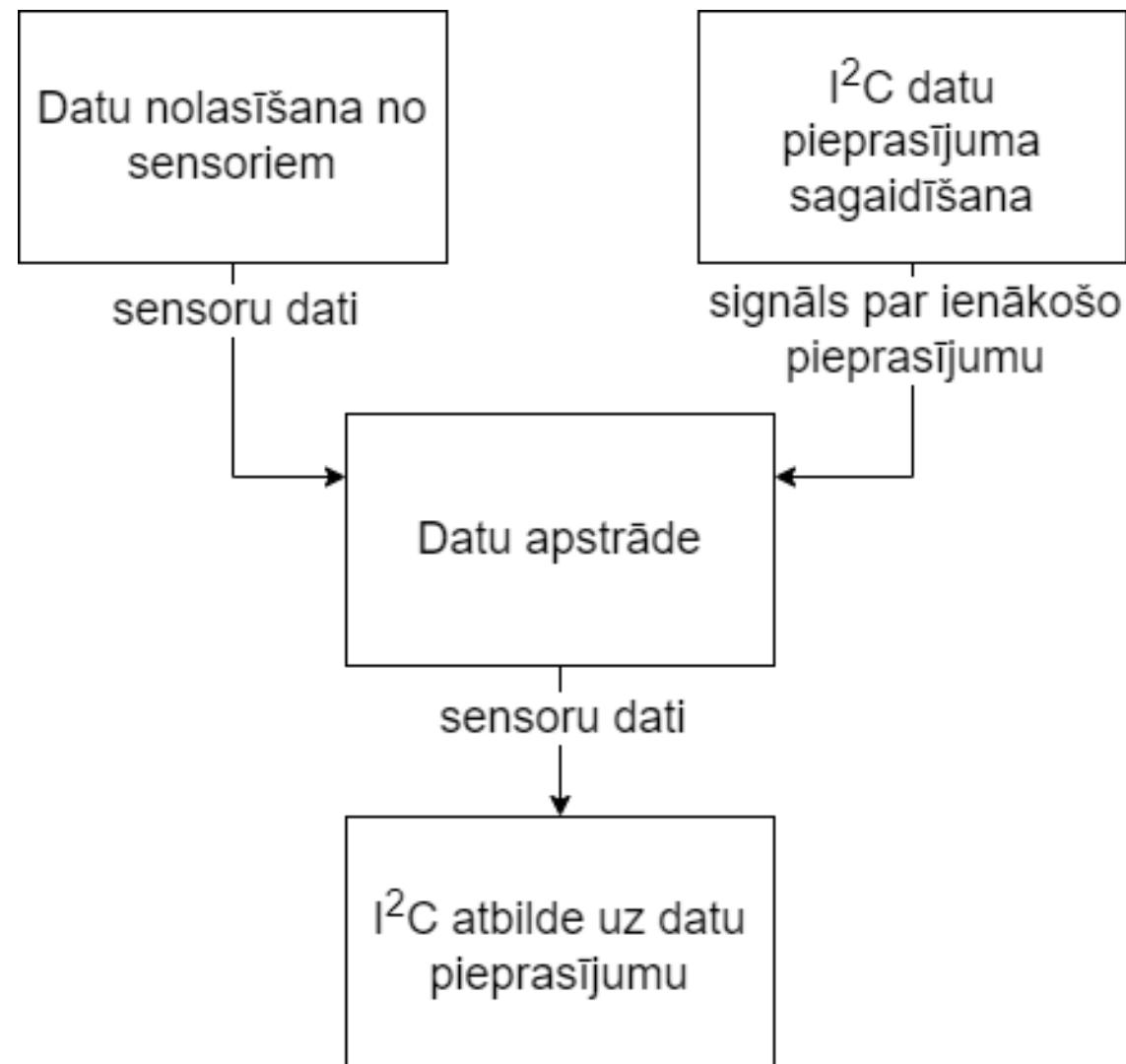
Anticarium Camera datu plūsmu diagramma



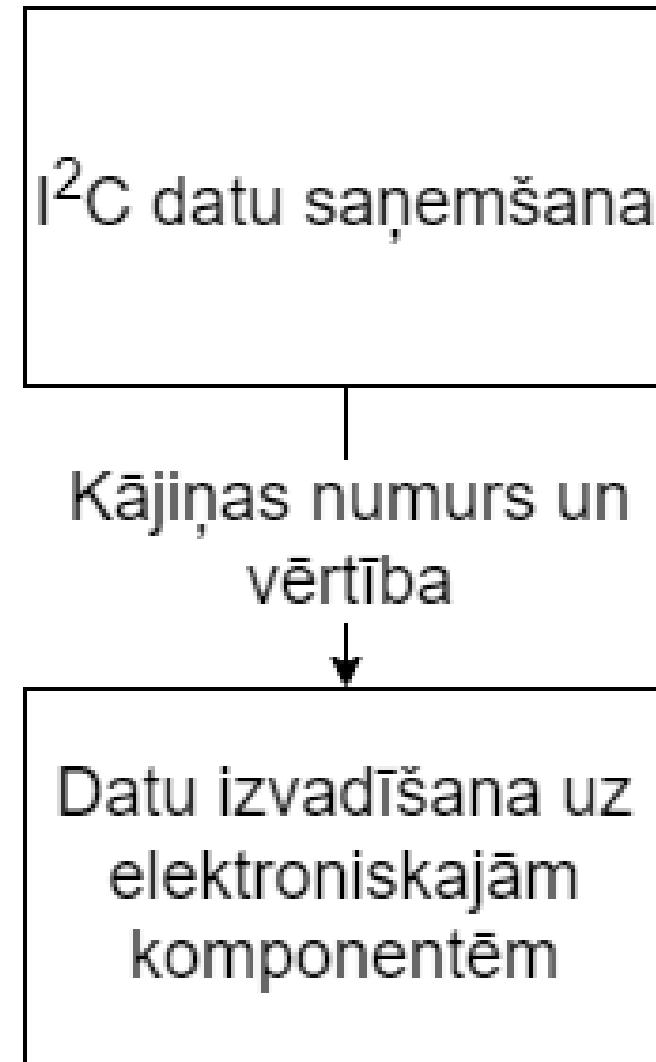
Anticarium Pi datu plūsmu diagramma



Input IC datu plūsmu diagramma



Output IC datu plūsmu diagramma



31. pielikums

Google Test Unit Testing piemērs no
Anticarium/Desktop/tests/config/ApplicationSettingsTests.cpp

```
TEST(TestApplicationSettings, TestRead) {
    QObject parent;

    QString testSettingsPath(TEST_DATA_DIR);
    testSettingsPath.append("/config/TestSettings.ini");
    ApplicationSettings* applicationSettings = ApplicationSettings::instance(testSettingsPath, &parent);

    EXPECT_EQ(applicationSettings->getAnticariumUDPUrl(), "127.0.0.1");
    EXPECT_EQ(applicationSettings->getAnticariumUrl(), "http://127.0.0.1:5000");
    EXPECT_EQ(applicationSettings->getSensorDataFetchTimeout(), 1000);
    EXPECT_EQ(applicationSettings->getServerUDPPort(), 10432);
    EXPECT_EQ(applicationSettings->getImageWidth(), 500);
    EXPECT_EQ(applicationSettings->getImageHeight(), 200);
    EXPECT_EQ(applicationSettings->getLogLevel(), 2);
    EXPECT_EQ(applicationSettings->getClientUDPPort(), 10000);
}
```

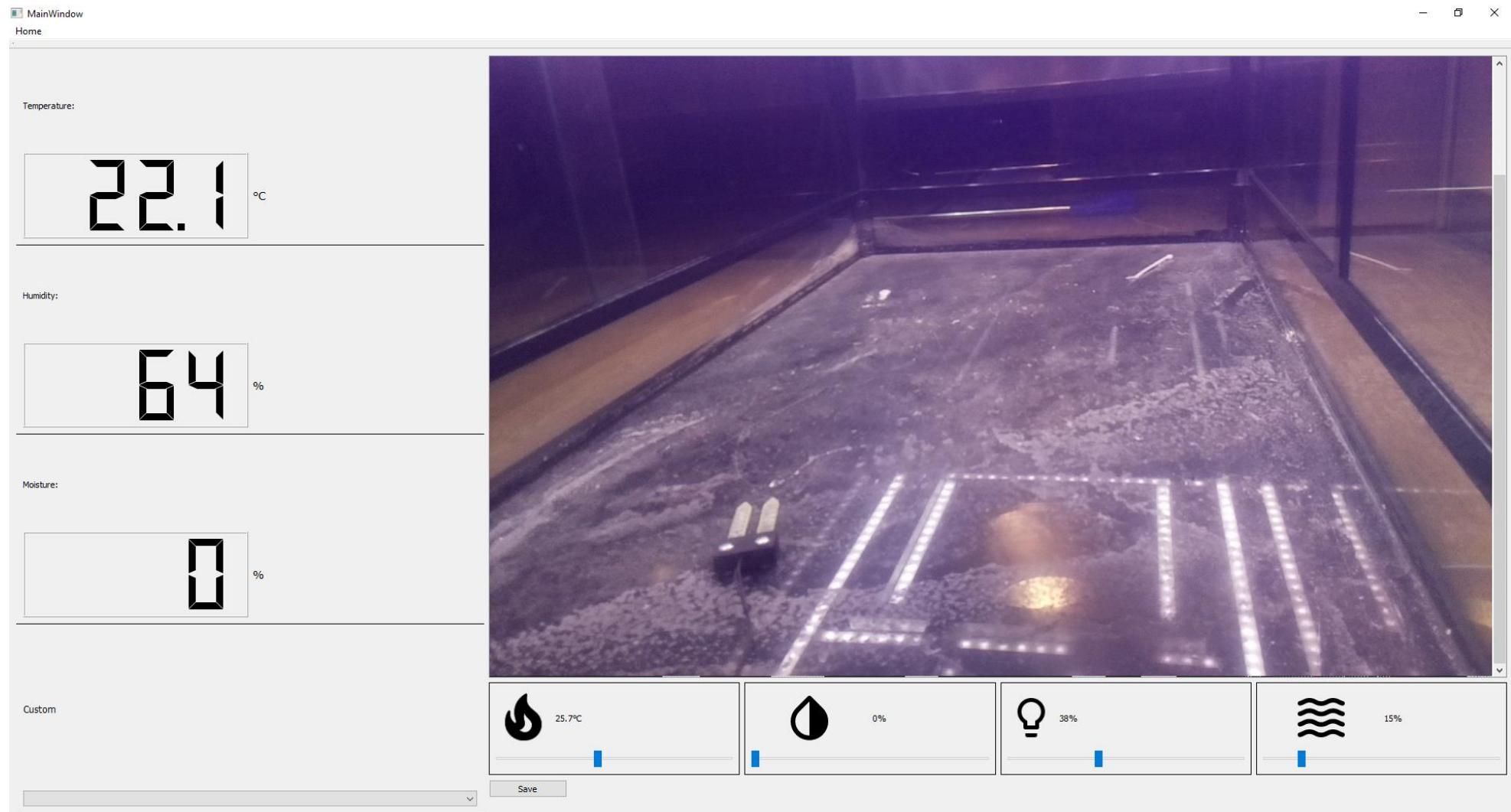
32. pielikums

Lietotnes sākotnējie ielādēšanās pieprasījumi no
Anticarium/Desktop/src/anticarium_desktop/ MainWindowManager.cpp

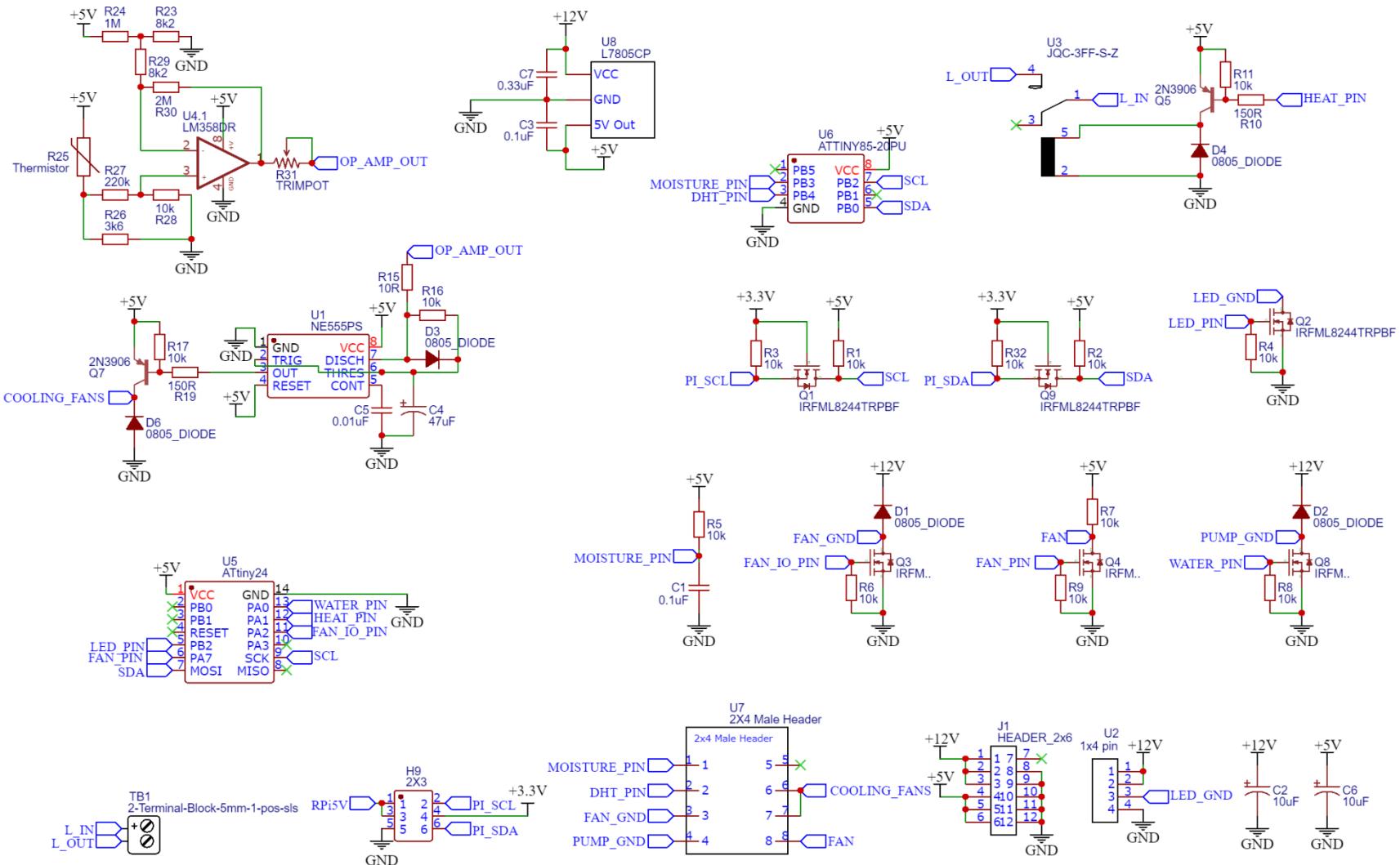
```
// Request data for the first time on first time loading
emit requestDataEvent(JTTP::REQUEST_DATA::SENSOR_DATA);
emit requestDataEvent(JTTP::REQUEST_DATA::REGIMES);
emit requestDataEvent(JTTP::REQUEST_DATA::CONTROL_DATA);
emit requestDataEvent(JTTP::REQUEST_DATA::REGIME_ID);
```

33. pielikums

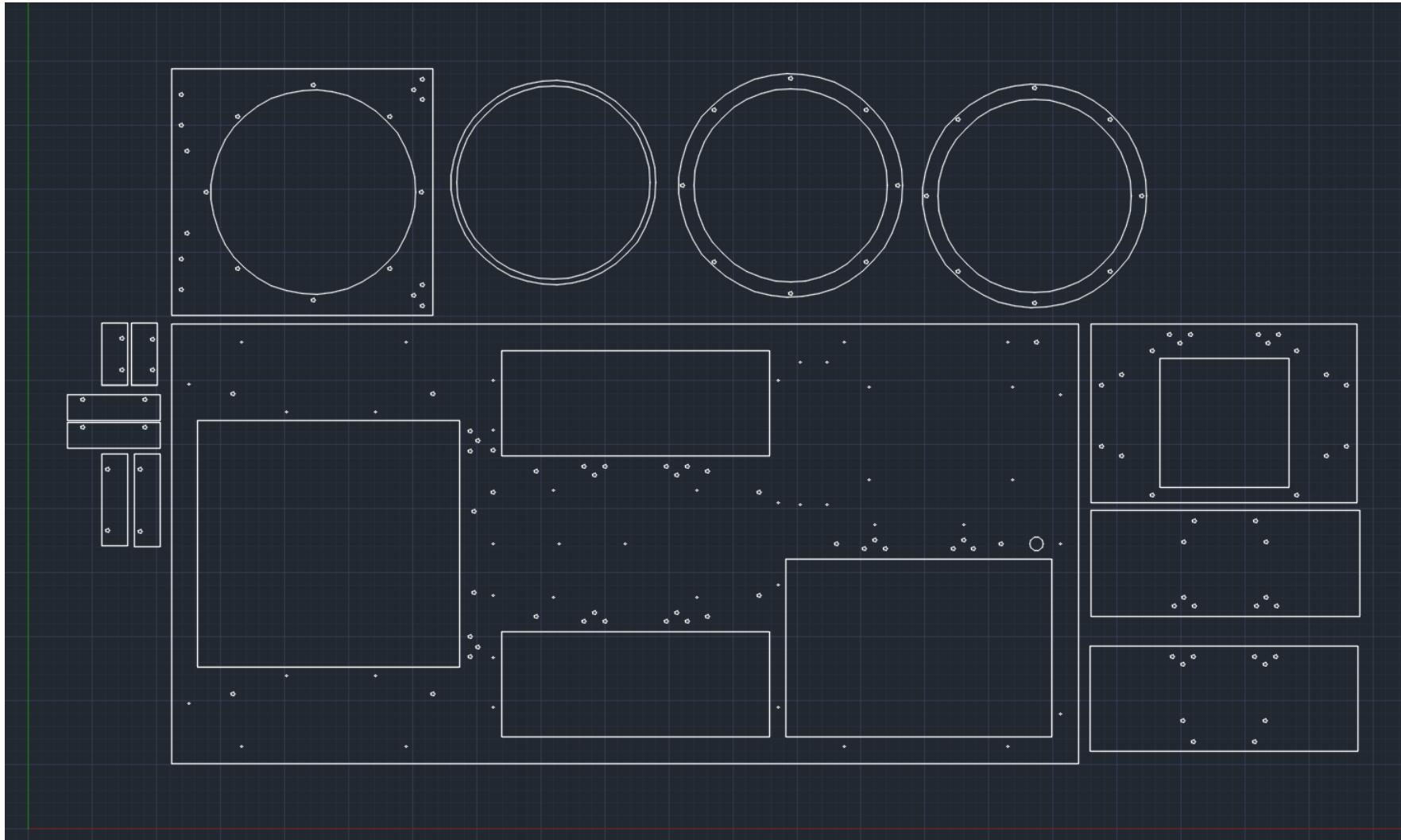
Anticarium Desktop lietotnes galvenais skats



Anticarium PCB labotā elektroniskā shēma

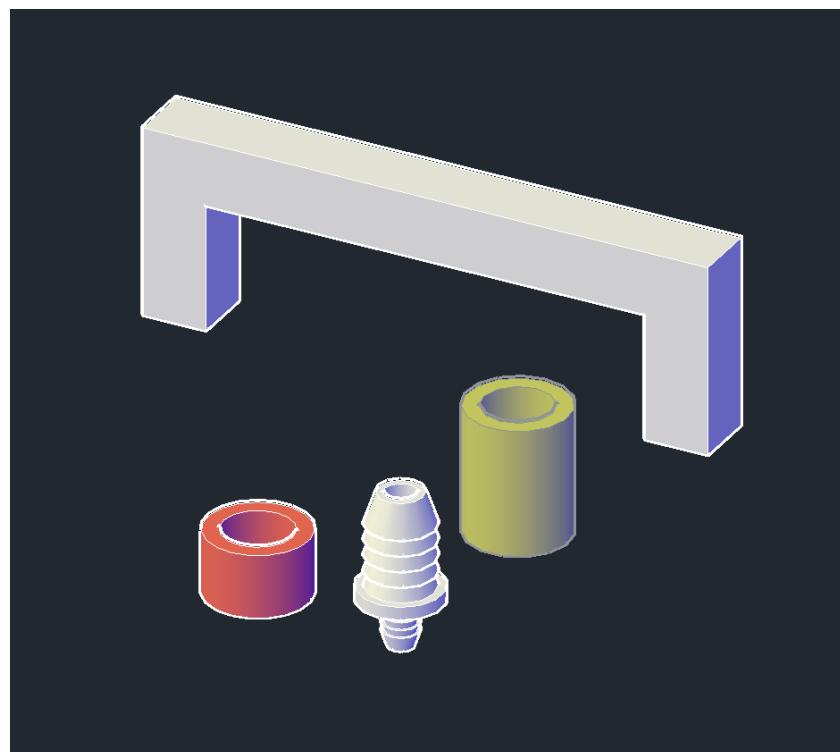


Terārija vāka rasējums



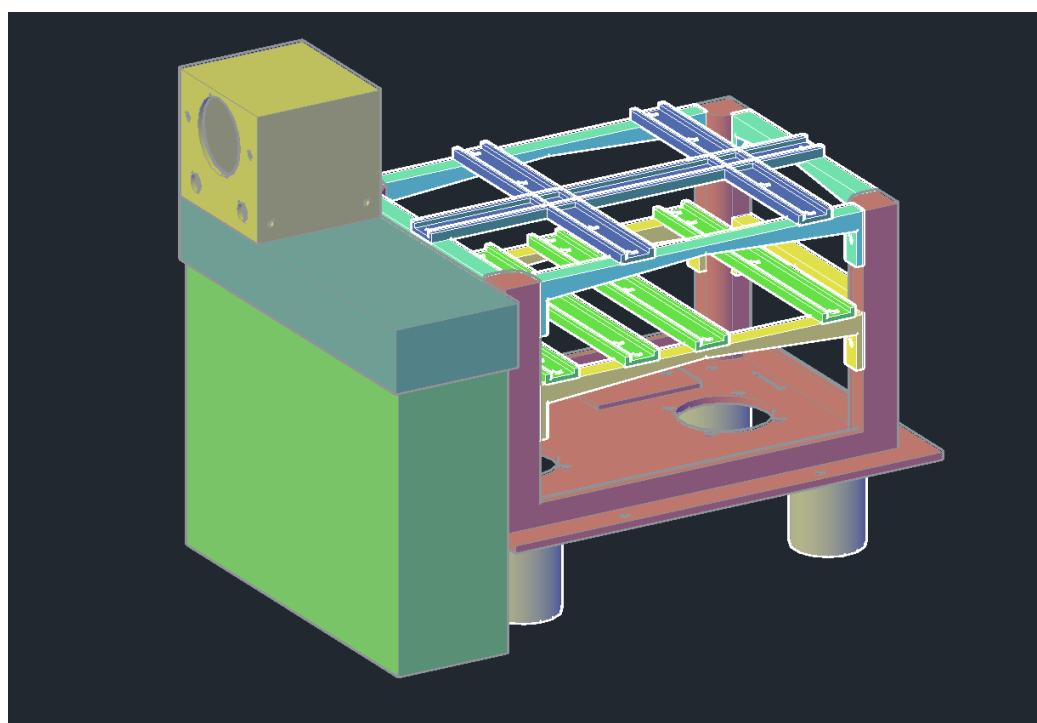
36. pielikums

Terārija lūku roktura un dažādu cauruļu savienotāju 3D modeļi



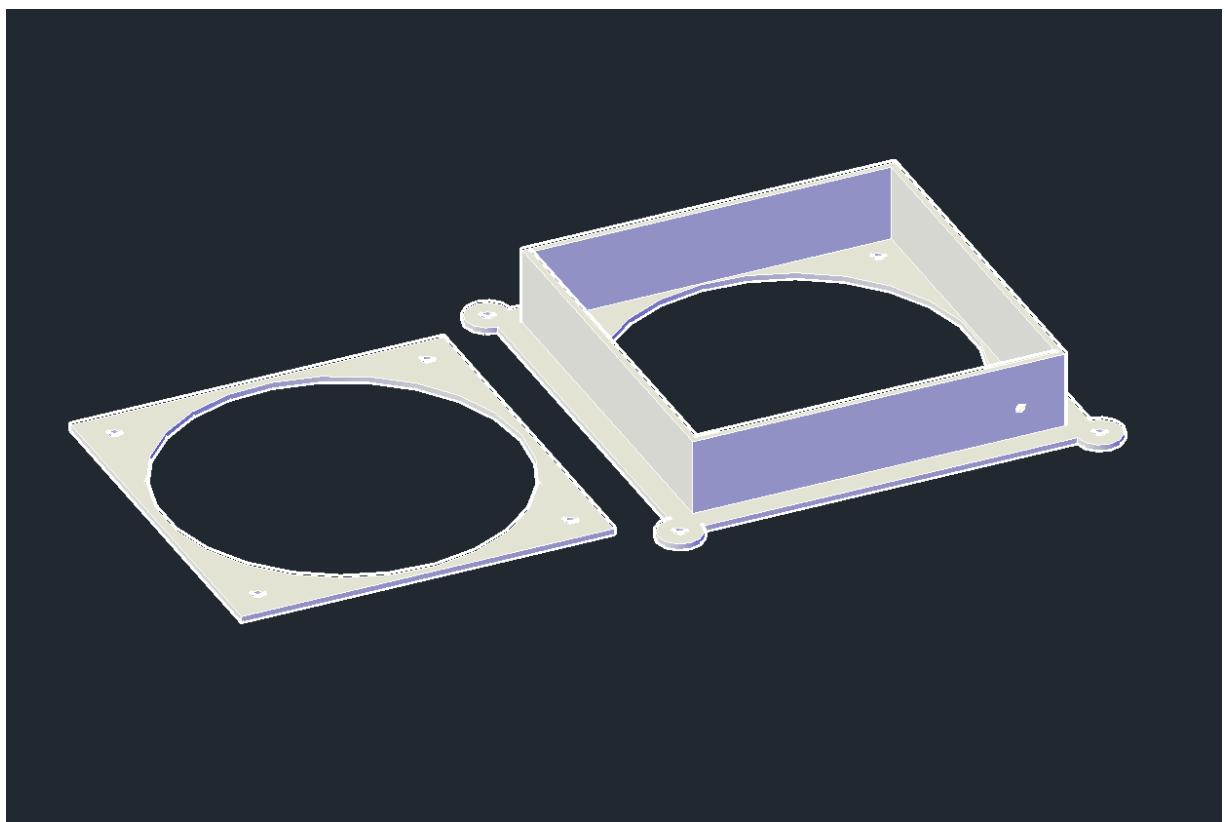
37. pielikums

Terārija vadības bloka un ūdens tvertnes korpusa 3D modeļi



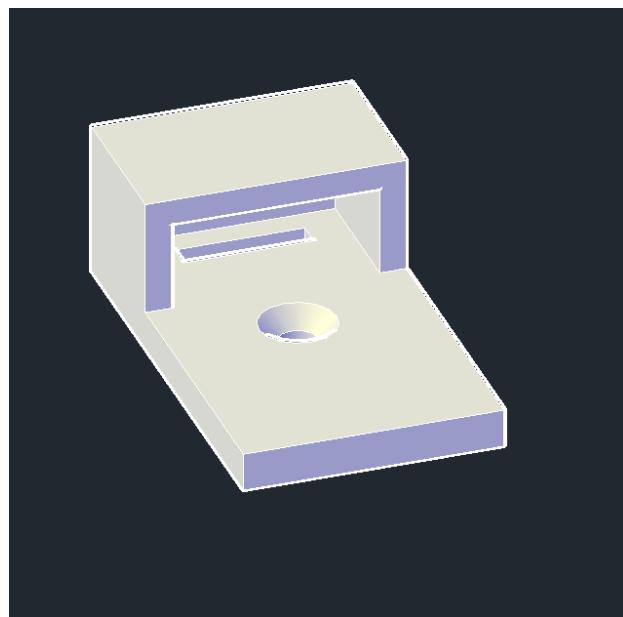
38. pielikums

Ventilatora korpusa 3D modeļi



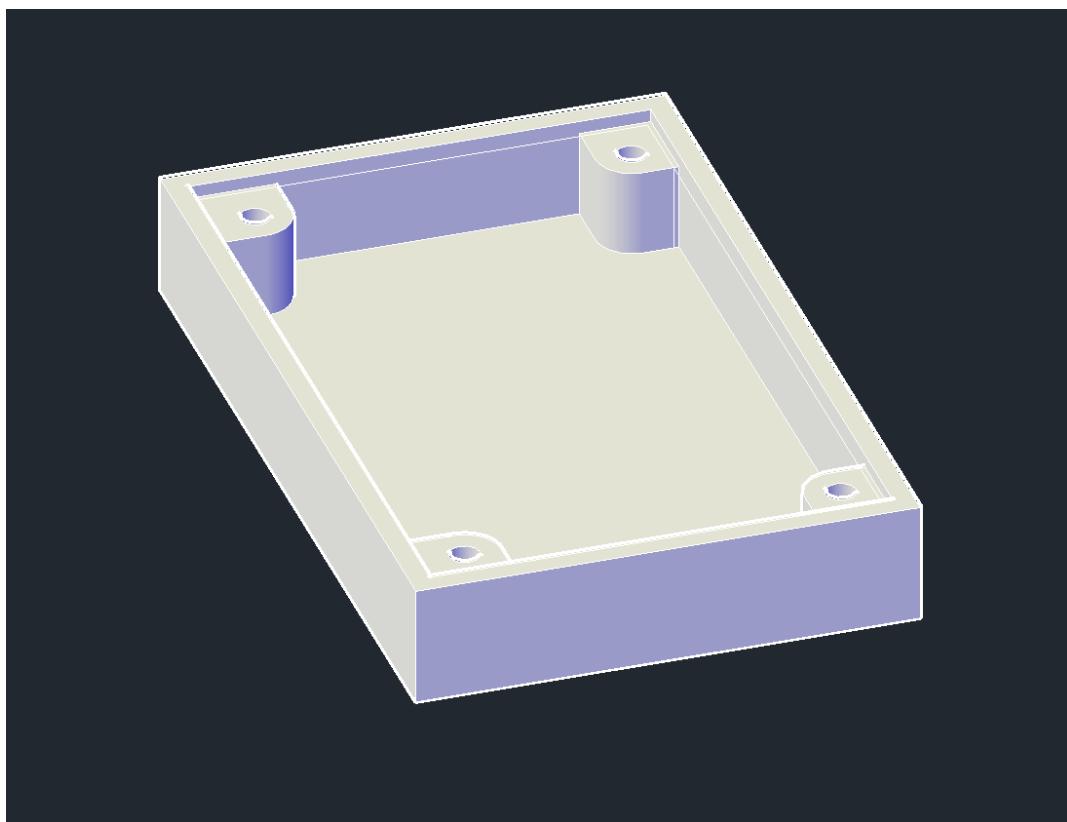
39. pielikums

Mitruma un temperatūras sensora korpusa 3D modelis



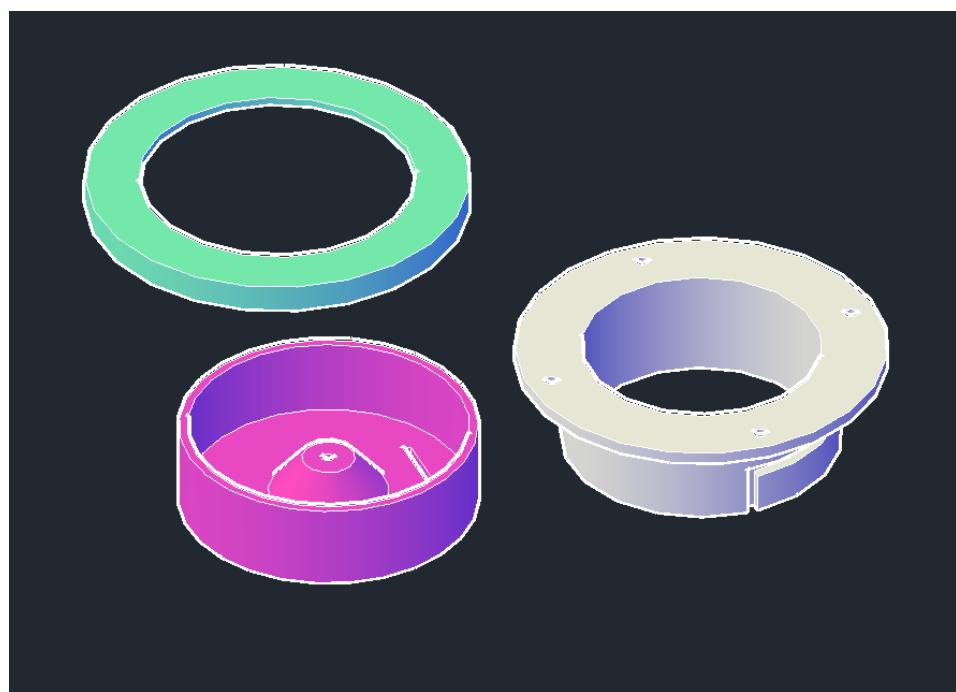
40. pielikums

Anticarium PCB korpusa 3D modelis



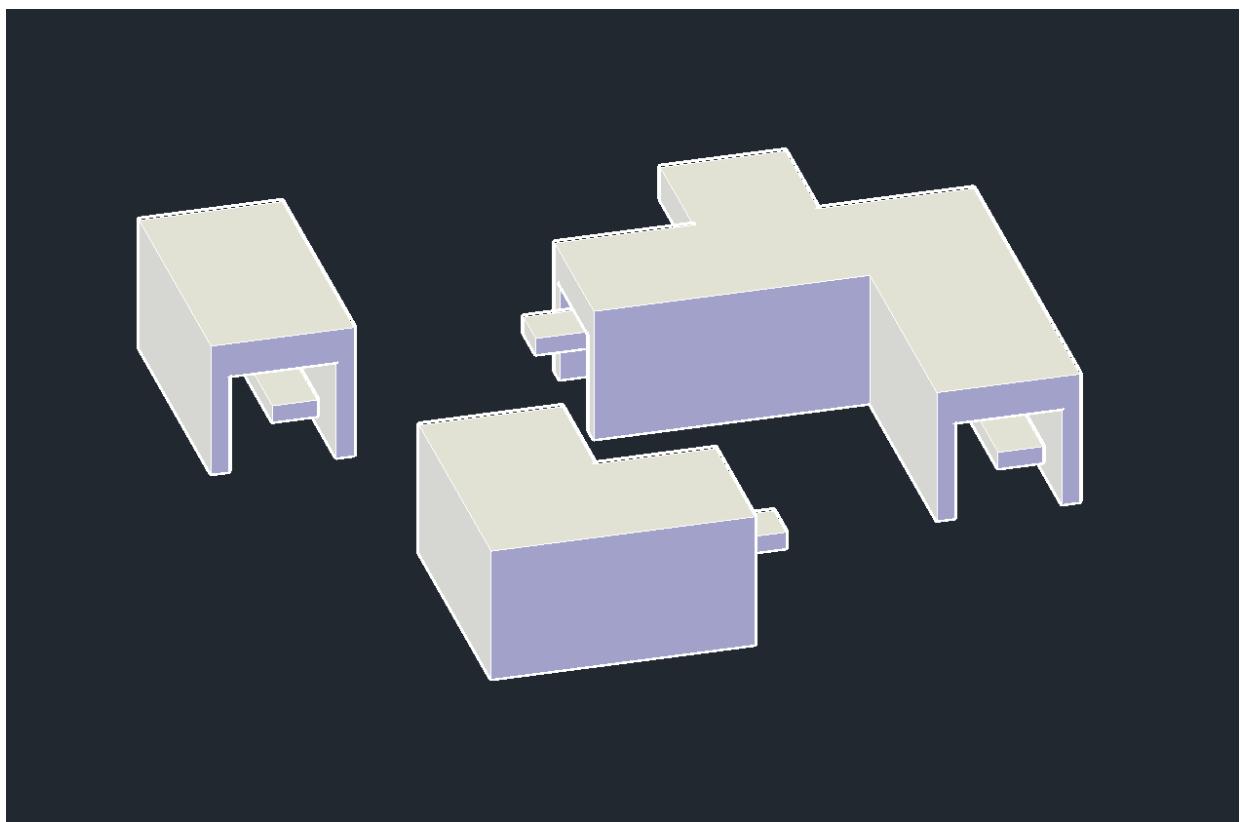
41. pielikums

Terārija vadības bloka un ūdens tvertnes korpusa 3D modeļi



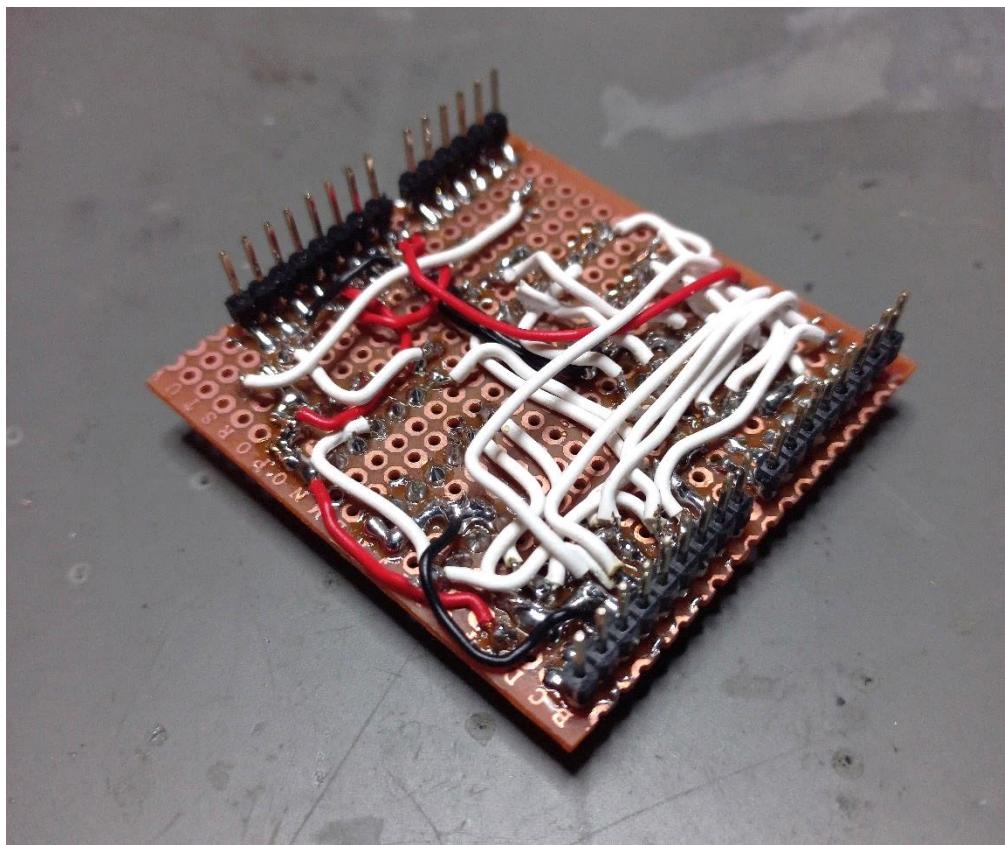
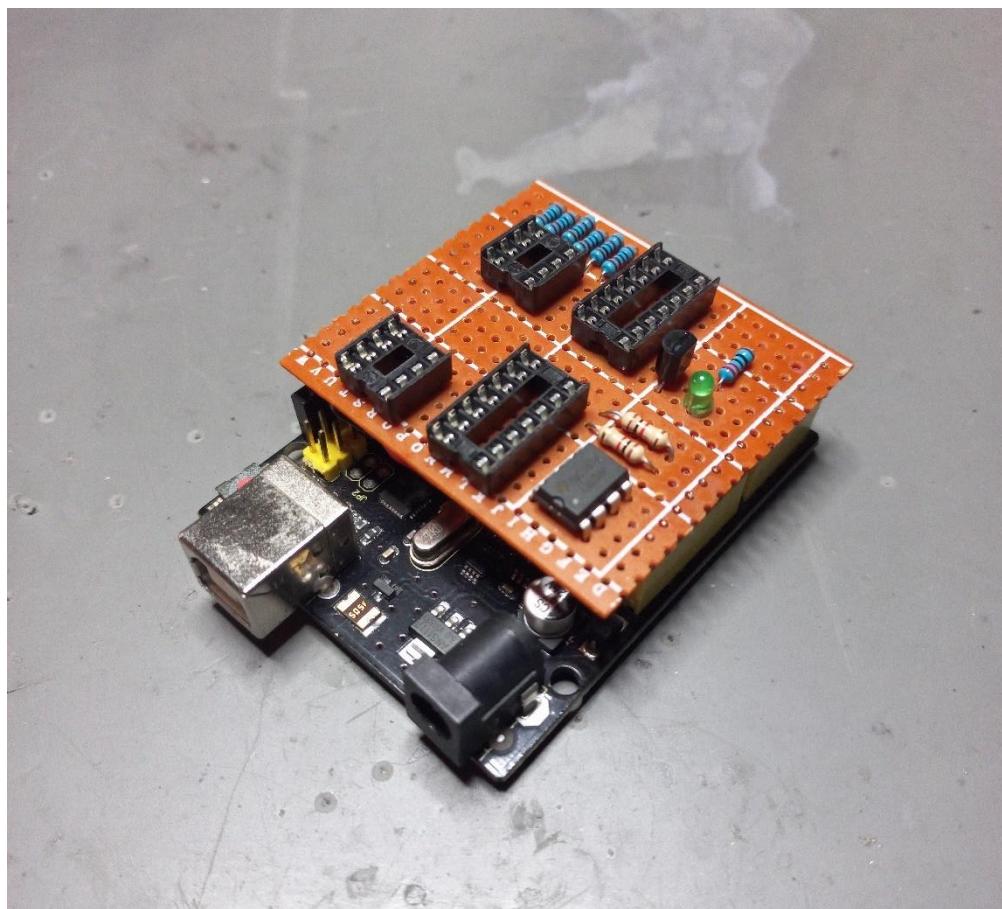
42. pielikums

Kabeļu profilu stūru 3D modeļi

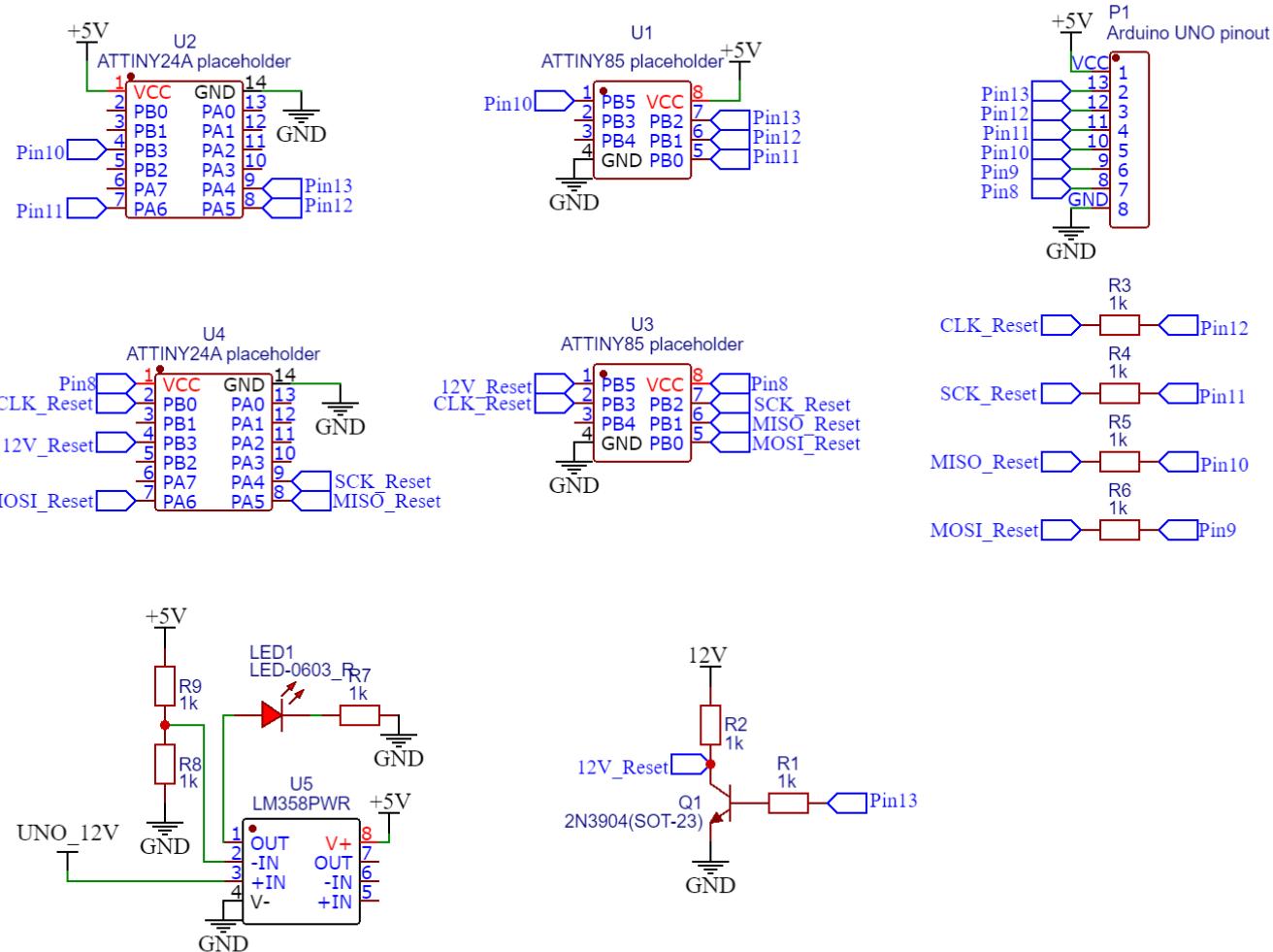


43. pielikums

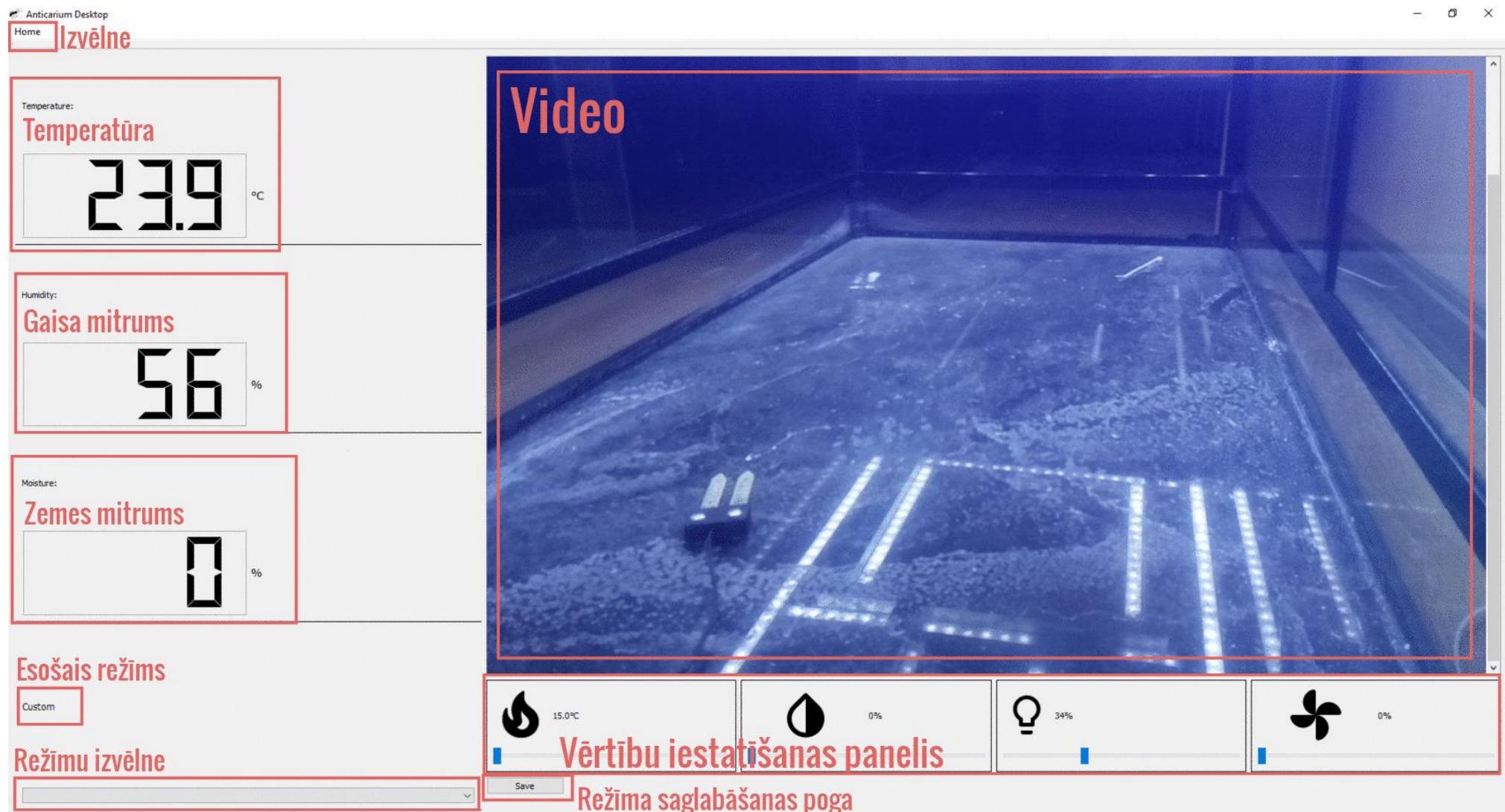
ATtiny mikroshēmu programmaturs



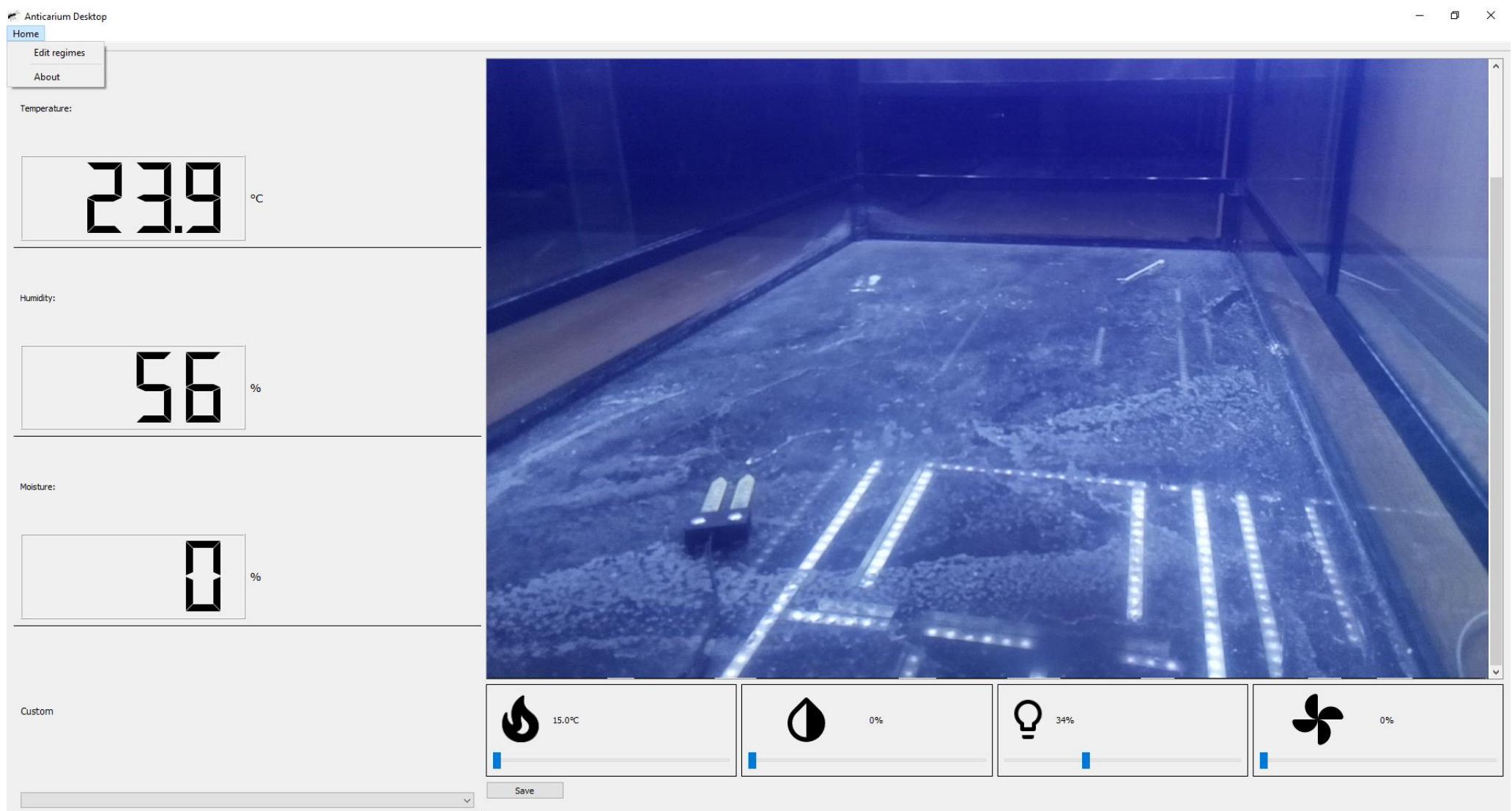
ATTiny mikroshēmu programmatora shēma



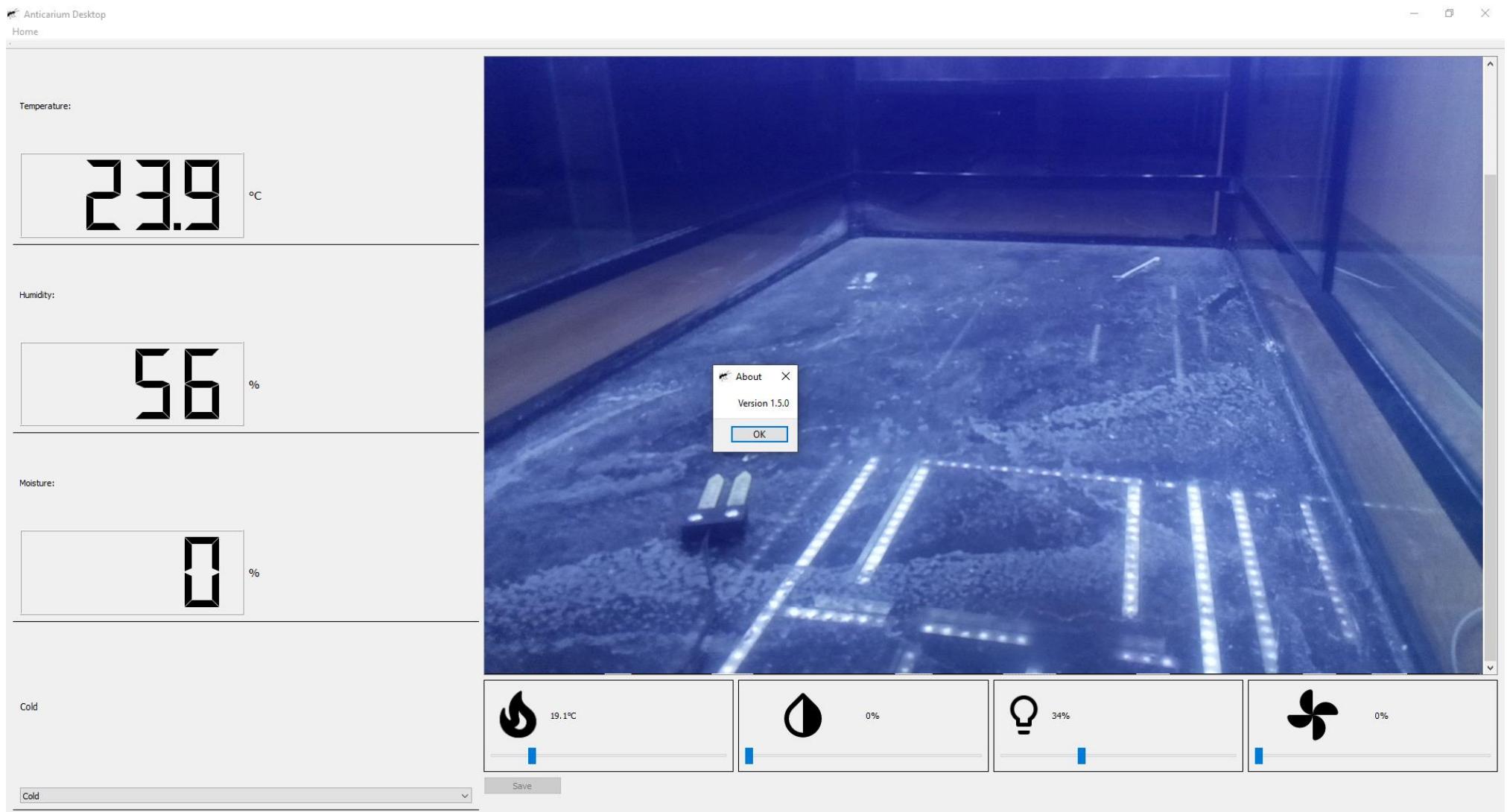
Anticarium Desktop ar skaidrojumu



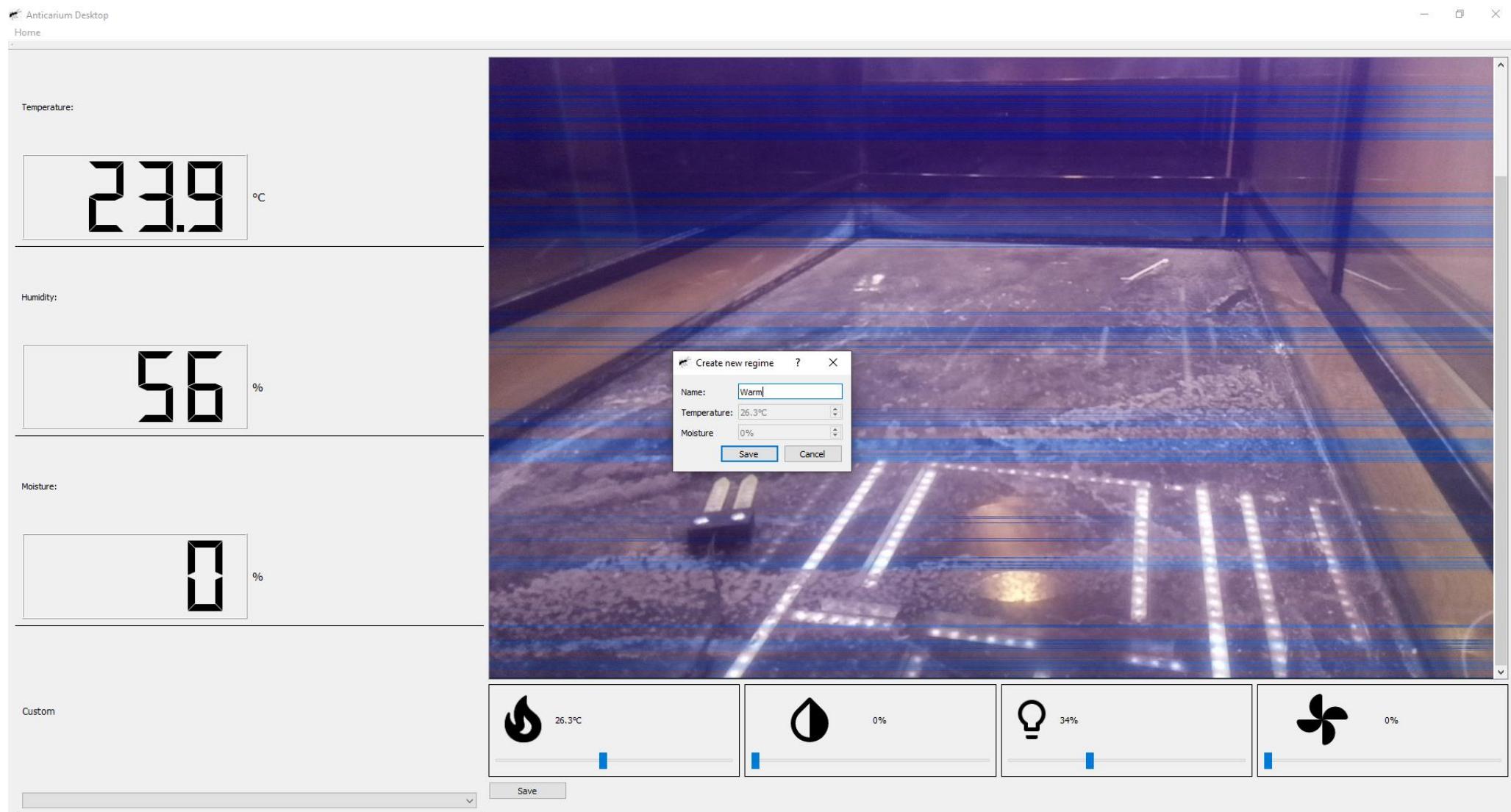
Anticarium Desktop ar atvērtu izvēlni



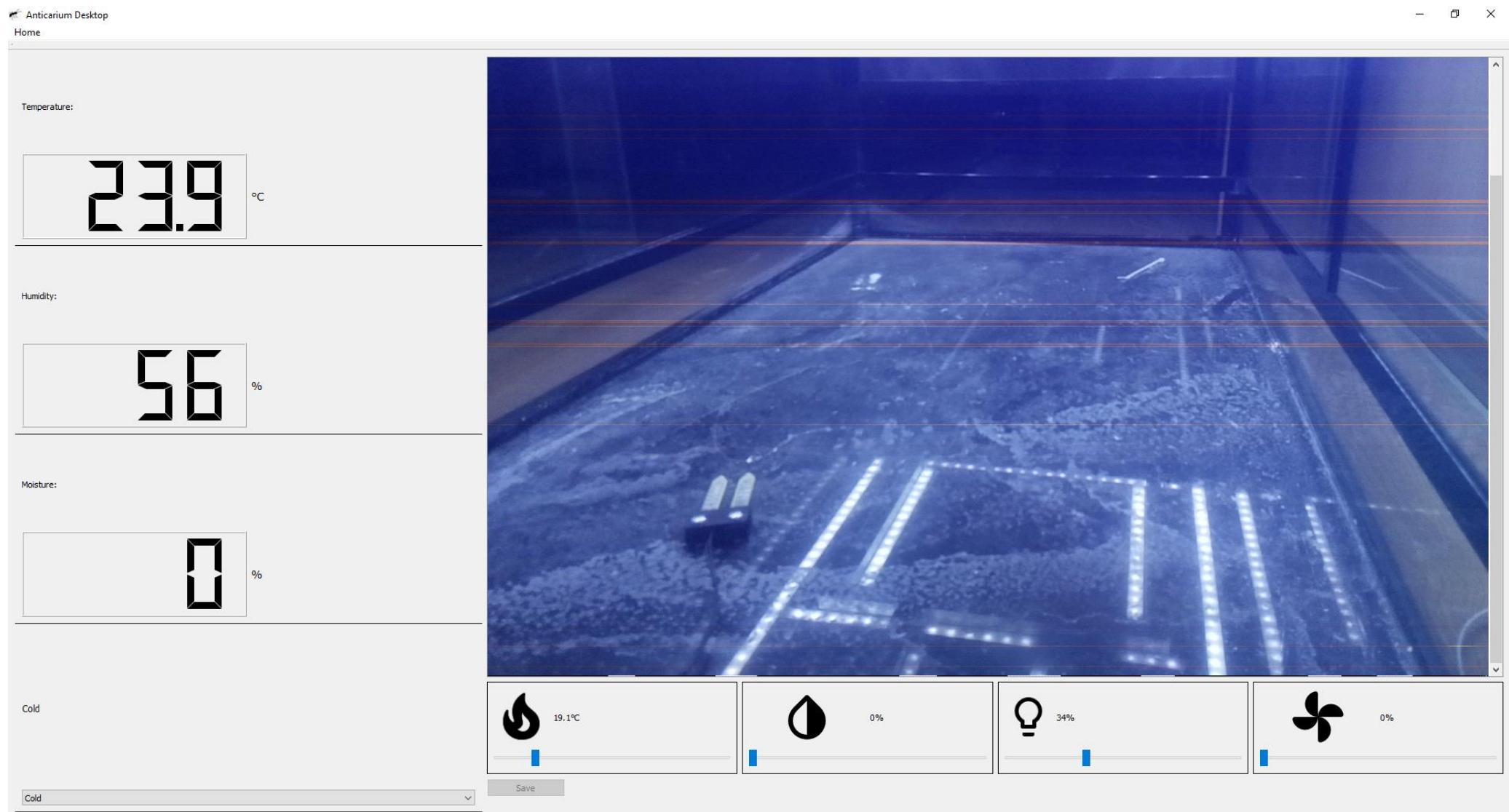
Anticarium Desktop ar atvērtu versijas logu



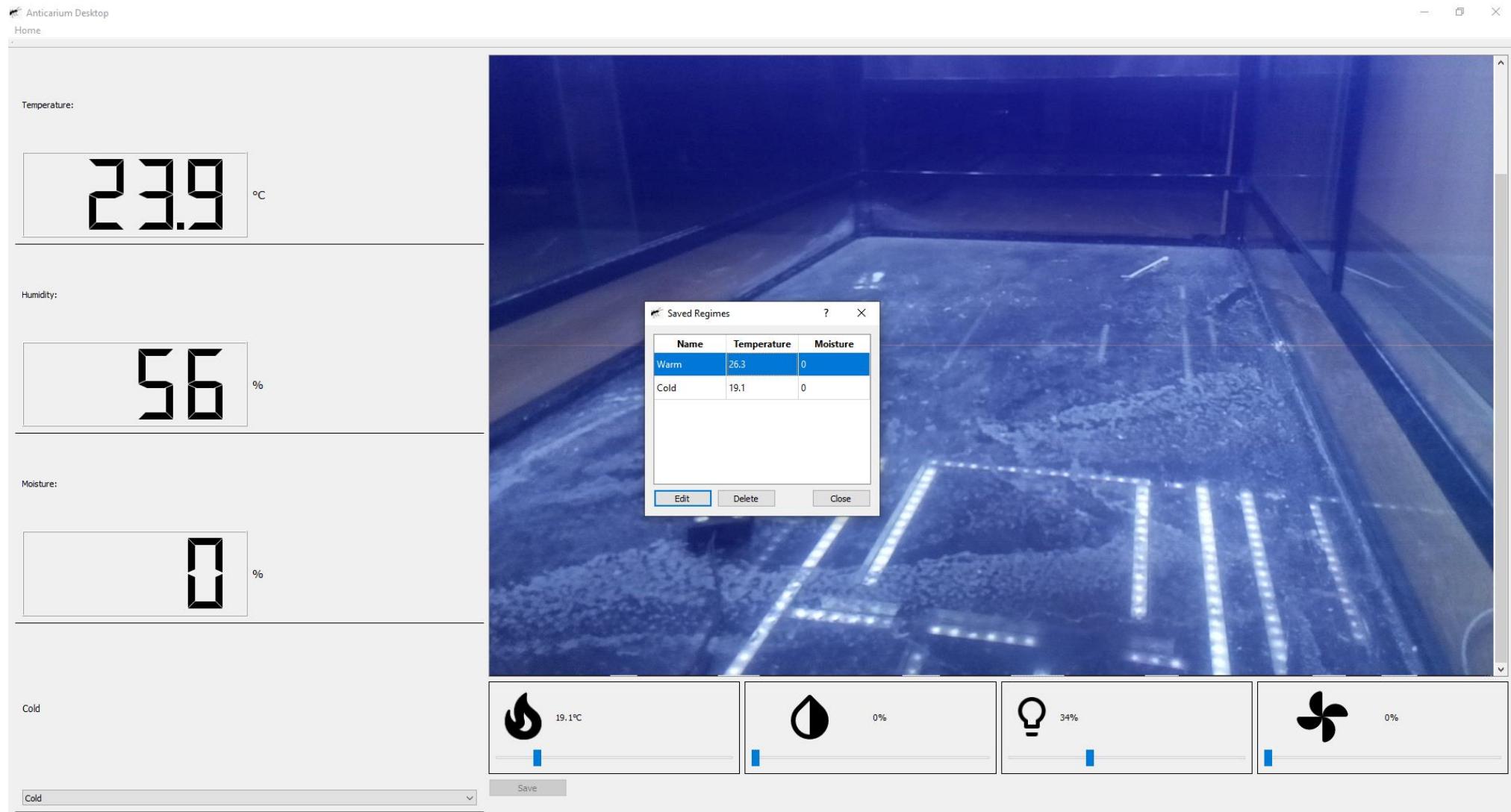
Anticarium Desktop jauna režīma saglabāšanas logs



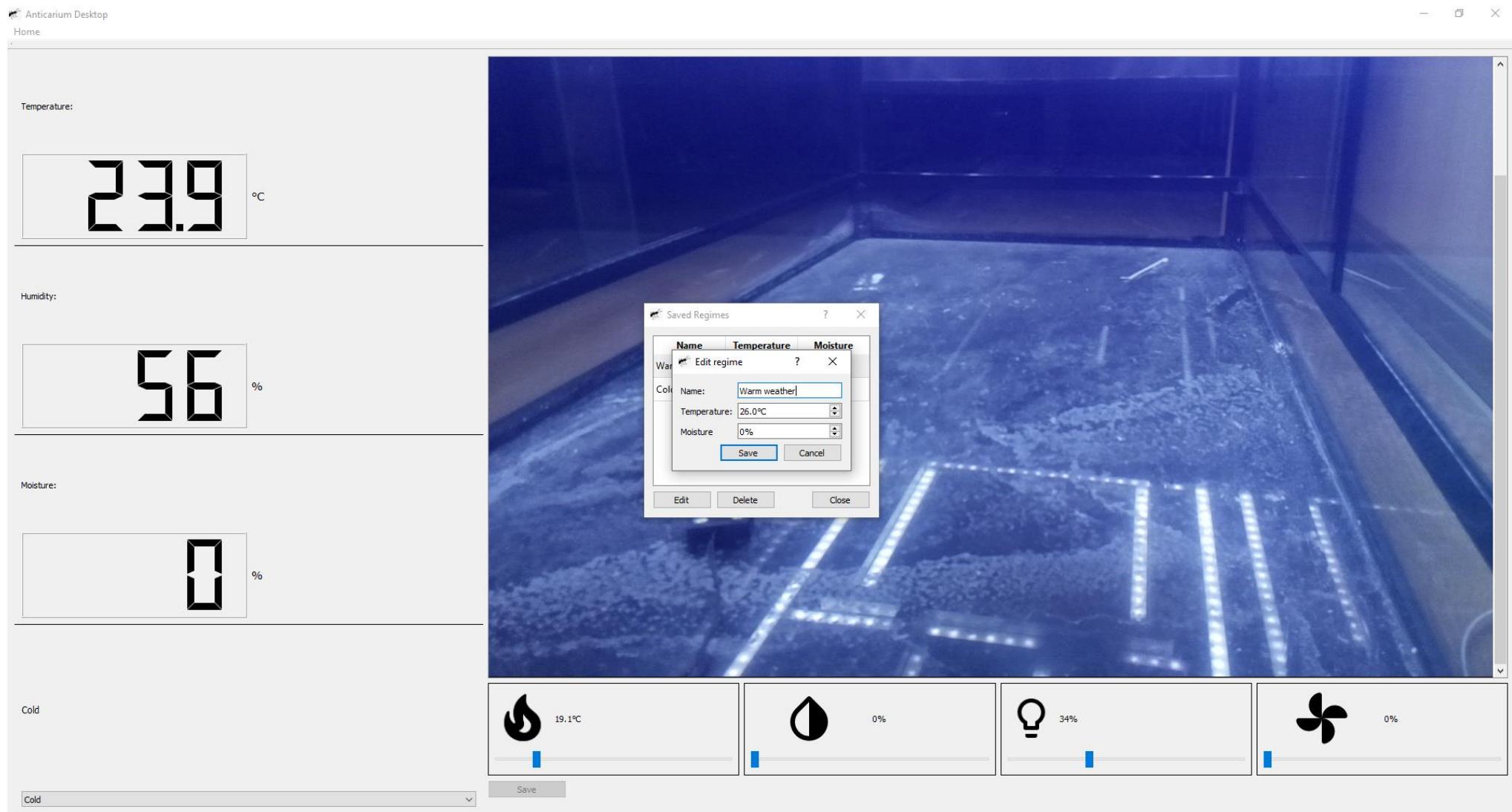
Anticarium Desktop ar mainītu esošo režīmu



Anticarium Desktop saglabāto režīmu pārskata logs



Anticarium Desktop režīma redīģēšanas logs



52. pielikums

shared_types::Control klases deklarācija

```
namespace shared_types {
class Control {
public:
    int getLightPercentage() const;
    void setLightPercentage(int newLightPercentage);
    int getWindPercentage() const;
    void setWindPercentage(int newWindPercentage);
    shared_types::RegimeValue getRegimeValue() const;
    void setRegimeValue(const shared_types::RegimeValue& regimeValue);

private:
    int lightPercentage_ = 0;
    int windPercentage_ = 0;
    shared_types::RegimeValue regimeValue_;
};

} // namespace shared_types
```

53. pielikums

shared_types::Control objektu serializācijas un deserializācijas funkcijas

```
namespace nlohmann {
inline void from_json(const json& j, shared_types::Control& x) {
    x.setLightPercentage(j.at("light_percentage").get<int>());
    x.setWindPercentage(j.at("wind_percentage").get<int>());
    x.setRegimeValue(j.at("regime_value").get<shared_types::RegimeValue>());
}

inline void to_json(json& j, const shared_types::Control& x) {
    j["light_percentage"] = x.getLightPercentage();
    j["wind_percentage"] = x.getWindPercentage();
    j["regime_value"] = x.getRegimeValue();
}
} // namespace nlohmann
```

54. pielikums

nlohmann::json deserializācijas piemērs uz shared_types::Control tipa objektu “implicitly”

```
nlohmann::json jsonIn          = getJsonFromPath(filePath);
shared_types::Control control = jsonIn;

EXPECT_EQ(control.getLightPercentage(), 23);
EXPECT_EQ(control.getWindPercentage(), 12);
EXPECT_EQ(control.getRegimeValue().getMoisture(), 76);
EXPECT_EQ(control.getRegimeValue().getTemperature(), 32.4f);
```

55. pielikums

RegimeId modeļa JSON piemērs

```
{  
    "id": 3  
}
```

56. pielikums

RegimeValue modeļa JSON piemērs

```
{  
    "temperature": 23.1,  
    "moisture": 87  
}
```

57. pielikums

SensorData modeļa JSON piemērs

```
{  
    "temperature": 23.1,  
    "humidity": 23,  
    "moisture": 87  
}
```

58. pielikums

Control modeļa JSON piemērs

```
{  
    "light_percentage":23,  
    "wind_percentage":12,  
    "regime_value":  
    {  
        "moisture": 76,  
        "temperature": 32.4  
    }  
}
```

59. pielikums

Regimes modeļa JSON piemērs

```
{  
    "regimes": [  
        "mode_1",  
        "mode_2",  
        "mode_3"  
    ]  
}
```

60. pielikums

Regime modeļa JSON piemērs

```
{  
  
    "name": "mode_1",  
    "regime_id":  
    {  
        "id": 3  
    },  
    "regime_value":  
    {  
        "temperature": 23.1,  
        "moisture": 77  
    }  
}
```

61. pielikums

SavedRegimes modeļa JSON piemērs

```
{  
    "saved_regimes": [  
        {  
            "name": "mode_1",  
            "regime_id":  
            {  
                "id": 3  
            },  
            "regime_value": {  
                "temperature": 23.1,  
                "moisture": 50  
            }  
        },  
        {  
            "name": "mode_2",  
            "regime_id":  
            {  
                "id": 4  
            },  
            "regime_value": {  
                "temperature": 15.5,  
                "moisture": 78  
            }  
        },  
        {  
            "name": "mode_3",  
            "regime_id":  
            {  
                "id": 5  
            },  
            "regime_value": {  
                "temperature": 35.8,  
                "moisture": 10  
            }  
        }  
    ]  
}
```

62. pielikums

Control modeļa serializācijas un deserializācijas Python implementācija

```
class Control():  
    def __init__(self, lightPercentage, windPercentage, regimeValue):  
        self.lightPercentage = lightPercentage  
        self.windPercentage = windPercentage  
        self.regimeValue = regimeValue  
  
    def toControl(dct):  
        return Control(dct['light_percentage'], dct['wind_percentage'], toRegimeValue(dct['regime_value']))  
  
    def fromControl(control):  
        dct = {  
            'light_percentage': control.lightPercentage,  
            'wind_percentage': control.windPercentage,  
            'regime_value': fromRegimeValue(control.regimeValue)  
        }  
        return dct
```

63. pielikums

Tabulas “SavedRegimes” struktūra

Nr.	Nosaukums	Tips	Izmērs [B]	Piezīme
1.	ID	INT	0 - 8	Režīma ID
2.	NAME	TEXT	-	Režīma nosaukums
3.	TEMPERATURE	REAL	8	Uzturāmā temperatūra
4.	MOISTURE	INT	0 - 8	Uzturamais mitrums

64. pielikums

sqlite.py skripts

```
import sqlite3

# Creates anticarium database if it does not exist
conn = sqlite3.connect('anticarium.db')

# Adds Regimes table to anticarium database
conn.execute('''CREATE TABLE REGIMES (
    ID INT PRIMARY KEY NOT NULL,
    NAME TEXT NOT NULL,
    TEMPERATURE REAL NOT NULL,
    MOISTURE INT NOT NULL
);''')

conn.close()
```

65. pielikums

Tabulas “SavedRegimes” iespējamais saturs

ID	NAME	TEMPERATURE	MOISTURE
0	Regime 1	21.3	100
1	Regime 2	32.1	24
2	Regime 3	15.0	0