VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Daniel Hjerppe

PIENJÄNNITEKESKUSTEN LÄMPENEMÄTESTAUS

Prosessi ja sen automatisointi

Tekniikka
2024

VAASAN AMMATTIKORKEAKOULU
Tekniikka

TIIVISTELMÄ

Tekijä	Daniel Hjerppe
Opinnäytetyön nimi	Pienjännitekeskusten lämpenemätestaus: Prosessi ja sen automatisointi
Vuosi	2024
Kieli	suomi
Sivumäärä	65
Ohjaaja	Osku Hirvonen

Opinnäytetyön aiheena on etsiä keinoja ja ratkaisuja automaation avulla kahteen pienjännitekeskusten lämpenemätestauksessa aikaa vievään ongelmaan: raportointiin ja testikriteerien täyttymisen todentamiseen. Työssä keskitytään erityisesti pienjännitekeskusten standardin mukaisiin määritelmiin lämpenemätestaukselle, sekä nykyisten sovellusten ja laitteistojen ominaisuuksien ja laajentamismahdollisuuksien vertailuun.

Teoreettisena pohjana on pienjännitekeskusten standardin mukaiset määritelmät lämpenemätestaukselle, sekä nykyisten sovellusten ja laitteistojen ominaisuudet ja laajentamismahdollisuudet. Lisäksi teoriaosuudessa tutustutaan valitun toteutusratkaisun kannalta keskeisiin ohjelmointiparadigmoihin.

Opinnäytetyön tuloksena on kehitetty erityisesti pienjännitekeskusten lämpenemätesteihin räätälöity sovellus, joka vähentää manuaalisten työvaiheiden määrää n. 2/3. Tässä raportissa käydään läpi kehitetyn sovelluksen rakenne ja toimintalogiikka, sekä opastetaan uuden sovelluksen käyttöönotossa ja jatkokehityksessä.

Avainsanat ohjelmistokehitys, automaatio, testaus, dokumentointi

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Tekniikka

ABSTRACT

Author	Daniel Hjerppe
Title	The Process and Automation of Temperature Rise Tests for Low-voltage Switchgear
Year	2024
Language	Finnish
Pages	65
Name of Supervisor	Osku Hirvonen

This thesis explores methods and solutions to automate two time-consuming issues in the temperature rise testing of low-voltage switchgear: reporting and verifying the fulfillment of test criteria. A particular focus is placed on definitions according to the standard for low-voltage switchgear temperature rise testing and comparing the features and expansion possibilities of current applications and hardware.

The theoretical framework is based on the methods described in the standard for low-voltage switchgear, properties and possibilities for expansion of software and equipment currently in use and on programming paradigms most related to the chosen solution. Several methods of expanding the functions of the current measuring software were identified, analyzed and compared to the tools available for programming a completely new measuring software for existing equipment.

As a result of this thesis a completely new software was developed and tailored specifically to the needs of low-voltage switchgear temperature rise tests. This newly developed software reduces the number of manual tasks for temperature rise tests by around 2/3. This thesis describes the development and inner workings of the software and ends with a guide on how to use and develop it further.

Keywords: Software development, automation, testing, documentation

SANASTO JA LYHENTEET

Aikaleima	Ohjelmoinnissa dataelementti, joka tallentaa tapahtuman ajankohdan
API	Application Programming Interface, ohjelmointirajapinta
Attribuutti	Luokan tai olion sisällä määritelty muuttuja
CJC	Cold Junction Compensation, termoparien kylmäliitoskompensaatio
Dataloggeri	Datankeräin, eli laite, joka kerää mittatietoa antureilta
Funktio	Ohjelmoinnissa nimetty koodilohko, joka ottaa vastaan syötteitä ja palauttaa tuloksen
HTTP	Hypertext Transfer Protocol, tiedonsiirtoprotokolla
JSON	JavaScript Object Notation, tiedonvaihtoformaatti
Luokka	Ohjelmoinnissa suunnittelumalli, joka määrittelee joukon attribuutteja ja metodeja
Metodi	Ohjelmoinnissa luokan tai olion sisällä määritetty funktio
Muuttuja	Ohjelmoinnissa nimetty tietorakenne, joka varaa muistista tilaa arvon säilyttämiseen
Olio	Ohjelmoinnissa instanssi luokasta, joka kapseloi attribuutteja ja metodeja
SDK	Software Development Kit, ohjelmistokehitystyökalupaketti
Termopari	Lämpötilaa mittaava anturi
UTC	Universal Coordinated Time, maailmanlaajuisesti sovittu ajanlaskun standardi

SISÄLLYS

KUVIO- JA TAULUKKOLUETTELO

1 JOHDANTO

Pienjännitekeskusten lämpenemätestaus on tärkeä osa laitteiston turvallisuuden ja vaatimustenmukaisuuden varmistamista. Keskuksen täytyy pystyä johtamaan mitoitusvirtaansa ylittämättä SFS-EN 61439-1 -standardissa määriteltyjä lämpenemän raja-arvoja, sillä liiallinen lämpeneminen voi esimerkiksi saattaa keskuksen käyttäjät vaaraan, tai heikentää johtimien ja laitteiden toiminta- tai suorituskykyä.

Lämpenemä todennetaan, kun lämpötila on vakiintunut keskusta ympäröivää ilmaa korkeampaan arvoon, eli saavuttanut termisen tasapainotilan. Lämpenemätestin kannalta terminen tasapainotila katsotaan saavutetuksi, kun lämpenemä on alle 1 K/h. [1].

Lämpenemätestit ovat usein verrattaen työläitä, sekä pitkäkestoisia ja olisikin resurssien kannalta eduksi, jos termisen tasapainotilan ilmaisu, mittausdatan prosessointi ja raportointi voitaisiin automatisoida mahdollisimman pitkälle.

Tämä opinnäytetyö on tehty VEOn toimeksiantona ja tavoitteena on kehittää nykyisiä menetelmiä paremmin lämpenemätestaukseen tarpeisiin räätälöity ratkaisu mittausdatan keräykseen ja raportointiin. Lisäksi tämä teksti toimii kehitettävän ohjelmiston käyttöoppaana, sekä dokumentaationa tukemassa ohjelmistoon tehtäviä muutoksia tulevaisuudessa.

2 LÄHTÖTIEDOT

Opinnäytetyön aloituspalaverissa nostettiin esille kaksi ongelmakohtaa nykyisessä prosessissa, joihin haluttiin löytää ratkaisu.

Ensimmäinen ongelma oli pienjännitekeskusten standardissa määritettyjen ehtojen täyttymisen toteaminen [1]. Käytössä olevasta sovelluksesta puuttui ominaisuus, jolla termisen tasapainotilan saavuttamista olisi voitu seurata reaaliajassa. Tasapainotilan saavuttamisen todentaminen täytyi tehdä selaamalla kaikkien kanavien mittaushistoriaa ja silmämääräisesti arvioida ehtojen täyttyneen, joka tarkoitti käytännössä, että testausta jatkettiin usein varmuuden vuoksi, vaikka todellisuudessa testin ehdot olisivat jo täyttyneet.

Toinen esiin nostettu ongelma liittyi lämpenemätestistä tehtävään mittausraporttiin. Nykyisestä sovelluksesta saatiin tulostettua mittaushistoria ja sen kuvaaja, mutta raportin laatiminen ja edellisten liittäminen raporttiin oli täysin manuaalista ja aikaa vievää. [2]

2.1 Lämpenemätestauksen prosessikuvaus

Tyypillistä lämpenemätestauksen prosessia voidaan kuvata esimerkiksi seuraavanlaisena manuaalisten työvaiheiden sarjana:

1. Testin määrittely
2. Testattavan laitteiston ja antureiden valmistelu
3. Antureiden nimeäminen ja konfiguroiminen mittaussovellukseen
4. Testin aloitus
5. Testin ehtojen seuranta*
6. Testin päättäminen*
7. Mittaushistorian prosessointi*
8. Testin hyväksyminen / hylkääminen*
9. Mittaushistorian tulostaminen*
10. Kaavion tulostaminen*

11. Lämpenemien laskenta*
12. Raportin kirjoittaminen*
13. Mittaushistorian lisääminen raporttiin*
14. Kaavion lisääminen raporttiin*
15. Kuvien ja piirustusten lisääminen raporttiin.

Näistä kohtien 5–14 tunnistettiin olevan automatisoitavissa, joka vähentäisi työvaiheiden määrää jo noin 2/3. Myöhemmissä keskusteluissa testin päättäminen haluttiin kuitenkin jättää manuaaliseksi, mutta toisaalta huomattiin mahdollisuus yhdistää testin määrittely ja antureiden konfigurointi yhdeksi prosessiksi.

2.2 Laitteisto

Ongelmanratkaisun kannalta oli tärkeää lisäksi tutustua nykyiseen lämpenemätestiprosessiin käytetyn sovelluksen käyttöön, sekä erityisesti suoraan mittaukseen liittyviin laitteisiin.

VEOn tekemät lämpenemätestit tehdään Pico Technologyn tuotteilla, tarkemmin PicoLog6-mittaussovelluksella ja TC-08-dataloggereilla. Mittausvirta pienjännitekeskukseen syötetään tarkoitukseen valmistetulta kolmivaihemuuntajalta, jossa mitoitusvirtaa seurataan Celsan TNM96-ETN:llä.

2.2.1 PicoLog 6

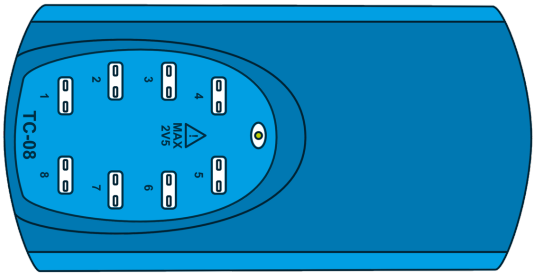
PicoLog6 on datankeräykseen ja -seurantaan erikoistunut sovellus, jolla voidaan reaaliaikaisesti seurata siihen liitettävien antureiden syöttämää dataa. Mittausdataa voidaan seurata kaavioiden ja taulukoiden muodossa ja mittauskanaville voidaan asettaa erityyppisiä hälytyksiä yksinkertaisten ehtolauseiden avulla. Lisäksi mittaukseen voidaan luoda virtuaalisia matematiikkakanavia, joilla antureiden dataa voidaan jalostaa erilaisilla funktioilla. Matematiikkakanavilla voitaisiin ilmaista anturin lämpenemä, mutta ei esimerkiksi termisen tasapainotilan ehtojen täyttymistä, sillä vain reaaliaikainen data on funktioiden käytettävissä. [3]

2.2.2 PicoLog Cloud

PicoLog 6:n versiosta 6.2.0 (julkaistu v. 2005) lähtien mittausdataa on voinut lähettää suoraan PicoLogista Pico Techin palvelimille, josta dataa voi edelleen lähettää muille laitteille, tai seurata mittaustuloksia etänä. PicoLog Cloudin avulla voidaan myös PicoLog-mittausprosesseja käynnistää etäyhteydellä. [4]

2.2.3 TC-08

TC-08 on USB-väylään liitettävä 8-kanavainen jännitteen- ja lämpötilanmittauslaite, joka tukee useita erilaisia termopareja (Kuvio 1.). Lisäksi siinä on yksi sisäinen kanava kylmäliitoskorvausta varten (CJC), jolla parannetaan termoparien mittatarkkuutta. PicoLogiin voidaan liittää maksimissaan kaksikymmentä TC-08:aa yhtäaikaisesti. [5]. VEOn käyttämiin dataloggereihin on oikean dataloggerin tunnistamisen vuoksi merkattu sarjanumeron loppu laitteen päälle. Sarjanumero on näkyvillä myös PicoLogin kanava-asetusten yhteydessä.



Kuvio 1. TC-08-dataloggeri.

2.2.4 Termoparit

VEOn lämpenemätesteissä käytetään pääsääntöisesti kahta termoparityyppiä: J- ja K-tyyppiä. K-tyyppi soveltuu jatkuviin mittauksiin 0–1100 °C:n alueella ja J-tyyppi vastaavasti 0–750 °C. Molempien mittatarkkuus on noin ±1 °C [6].

3 TEORIATAUSTAA

3.1 Lämpenemätestaus

Pienjännitekeskusten lämpenemätestauksen suoritusmenetelmät ja raja-arvot on määritelty standardissa SFS-EN IEC 61439-1:2022. Lämpötilojen mittaus määrätään tehtäväksi termopareja tai lämpötilamittareita käyttäen ja anturit on suojattava ilmavirtauksilta, sekä lämpötilasäteilyltä. Mittauspisteeet ja niiden raja-arvot ovat voimassa, kun ympäristön päivittäinen keskilämpötila on korkeintaan 35 °C.

Ympäristön lämpötila tulee mitata vähintään kahdella keskuksen vastakkaisille puolille asetetuilla antureilla, joista lasketaan keskiarvo.

$$T_{ympäristö} = \frac{T_{y1} + T_{y2}}{2}$$

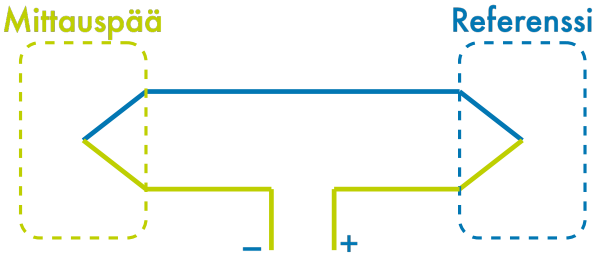
Mittausjakso voidaan päättää, kun mittauspisteen lämpötilan nousu on vakiintunut alle 1 K / h, jolloin lämpenemä lasketaan mittauspisteen ja keskuksen ulkopuolisen ilman lämpötilojen erotuksena:

$$\Delta T = T_{Laite} - T_{Ympäristö}$$

Lopuksi mittaustuloksista koostetaan taulukko. Testi katsotaan läpäistyksi, jos kaikkien mittauspisteiden lämpenemä on niille annettujen rajojen puitteissa.

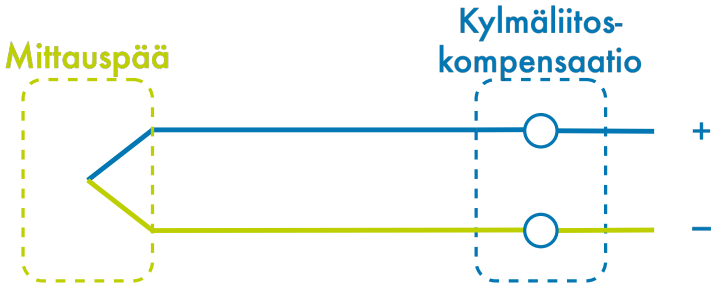
3.2 Termoparit

Termopari on lämpösähköiseen ilmiöön perustuva lämpötila-anturi, jossa kahdesta erilaisesta metallista, tai metalliseoksesta yhteen liitetyn langan välille muodostuu jännite-ero, joka on kyseiselle parille ominaisten rajojen sisäpuolella suoraan verrannollinen lämpötilaan. Jännite-ero on yleensä mikrovolttiluokassa.



Kuvio 2. Termoparin toimintaperiaate.

Jos langat on hitsattu yhteen molemmista päistä ja toista päätä, eli kylmää päätä pidetään vertailulämpötilassa 0 °C, voidaan kuuman pään lämpötila selvittää jännitemittarilla yllä olevan kuvan mukaisesti.



Kuvio 3. Termoparin kylmäliitoskompensaatio.

Käytännön mittauksissa ei ole kuitenkaan käytännöllistä pitää esimerkiksi jäävesiastiaa mukana referenssilämpötilan lähteenä, joten kylmäpään referenssijännite tehdään sähköisin menetelmin. [7, s. 22]

3.3 Python ohjelmointikielenä

Python on tulkattava, alustariippumaton ja monipuolinen ohjelmakieli, jota on kehitetty jo 1980-luvun lopulta lähtien. Sen syntaksi on suunniteltu helpoksi lukea ja kirjoittaa, jonka ansiosta ohjelman toimintoja voi usein toteuttaa vähemmillä koodiriveillä verrattuna moniin muihin ohjelmointikieliin. Esimerkiksi C++:lla toteutettu ohjelma on helposti 5–10 kertaa pidempi, kuin vastaava Pythonilla toteutettu ohjelma [8]. Kääntöpuolena muun muassa tulkattavuudesta johtuen Python ei ole tietokoneen resurssienkäytön kannalta kaikkein suorituskykyisin kieli. Ohjelma käännetään konekielelle lennosta vasta ohjelman ajon aikana, toisin kuin

esimerkiksi C-kielessä, jossa ohjelma on käännettävä ennen ajoa. Tulkattava kieli on myös hyvässä ja pahassa erittäin vikasietoinen, sillä ohjelma käynnistyy ja suorittaa koodia, mutta voi kaatua yllättäen, jos virheelliseen koodiriviin törmätään ajon aikana.

Voidaan siis sanoa, että Python vähentää sovelluksen kehitykseen käytettävää aikaa suorituskyvyn kustannuksella ja Pythonia käytettäessä onkin tärkeää käyttää sen hyötyjä edukseen. Esimerkiksi suorituskykyä vaativien tehtävien kohdalla voidaan ulkoistaa ne prosessit C-kielellä tehtyjen kirjastojen tehtäväksi.

Pythonille ominainen piirre on dynaaminen tyypitys, eli muuttujien tietotyypit määritellään vasta suorituksen aikana, toisin kuin monissa muissa kielissä, joissa tietotyyppi täytyy määritellä jo muuttujan luonnin aikana. Alla oleva esimerkki (kuvio 4) on Pythonissa täysin hyväksyttyä koodia.

```
numero = 7
numero = "VEO"
```

Kuvio 4. Esimerkki Pythonin dynaamisesta tyypityksestä.

Python tukee kuitenkin vapaaehtoisia tyyppiannotaatioita (kuvio 5), joita on suositeltavaa käyttää varsinkin monimutkaisemmissa ohjelmissa, joissa ei ole heti selvää minkä tyyppisiä muuttuja esimerkiksi funktio tarvitsee toimiakseen. [9]

```
numero:int = 7
numero:str = "VEO"
```

Kuvio 5. Esimerkki tyyppiannotaatioista.

Tyylisäännöt, kuten muuttujien, funktioiden ja muiden elementtien nimeäminen on periaatteessa ohjelmoijan vapaasti päätettävissä, mutta käytännössä olisi hyvä pitäytyä PEP 8:ssa määritellyissä tyylisäännöissä. Muuttujien ja funktioiden ni-

meämiseen suositellaan käytettäväksi pieniä kirjaimia ja erottamaan sanat alaviivalla, kuten esimerkiksi *testi_funktio*. Luokat puolestaan tulisi nimetä isoin kirjaimin ja ilman välimerkkejä, kuten *TestiLuokka*. [10]

Pythonissa on sisäänrakennettuna erittäin kattava standardikirjasto yleisimpiin käyttötarkoituksiin ja sen suosiosta johtuen sille on tarjolla myös erittäin laaja kokoelma kolmansien osapuolien kirjastoja.[11]

3.4 Proseduraalinen ohjelmointi

Proseduraalisessa ohjelmoinnissa ohjelman sen hetkinen tila on tallennettuna sen muuttujiin. Ohjelmaa voidaan selkeyden ja uusiokäytön vuoksi jakaa aliohjelmiin, kuten funktioihin, jotka käsittelevät niille annettuja arvoja ja joiden ulostulo voidaan tallentaa muuttujaan esimerkiksi käsittelyä varten jossakin muussa ohjelman kohdassa. Tämä ohjelmointiparadigma soveltuu hyvin yksinkertaisempiin ohjelmiin, jossa ohjelman suoritus voidaan mieltää lineaariseksi sarjaksi toimintoja ja käskyjä. [12] Monimutkaisemmissa ohjelmissa proseduraalisen ohjelmoinnin ylläpidon ja muokkauksen vaikeus kuitenkin kasvaa helposti eksponentiaalisesti, sillä ohjelmoijan on jatkuvasti huolehdittava tiedon kulkemisesta ja säilymisestä oikeissa muuttujissa.

3.5 Olio-ohjelmointi

Olio-ohjelmoinnin rakentava ajatus on tehdä ohjelmoinnista ihmislähtöistä, eli heijastella sitä tapaa miten me jäsennämme maailman. Olioilla voidaan ikään kuin paketoida proseduraalisesta ohjelmoinnista tutut käsitteet omiin ihmisystävällisempiin paketteihin. Oliot määritellään *luokissa*, jotka ovat olioiden rakennuspiirustuksia. Funktioita kutsutaan olioissa *metodeiksi* ja muuttujia *attribuuteiksi*.

Esimerkkinä oliosta voidaan ottaa esimerkiksi auto. Ensin luodaan *luokka* nimeltä Auto, jolla voi olla attribuuttina esimerkiksi väri, kiihtyvyys ja hetkellinen nopeus. Auton metodina voi olla esimerkiksi kiihdyttäminen, joka muuttaa auton hetkellistä nopeutta kiihtyvyysattribuutin perusteella.

Autoluokasta voidaan luoda FerrariAuto ja VolvoAuto. FerrariAutolle annetaan väriksi punainen ja suuri kiihtyvyys, VolvoAutolle taas sininen väri ja hidas kiihtyvyys

Kun jommankumman auton kiihdytysmetodia kutsutaan, kasvaa kyseisen autoolion hetkellinen nopeus sille oliolle tyypillisellä tavalla, vaikka molemmat ovat luotu samasta luokasta. [13]

Vastaava yksinkertainen esimerkki proseduraalisen ohjelmoinnin keinoin olisi huomattavasti vaikeaselkoisempi, sillä tilaa kuvaavat muuttujat ja toimintaa ohjaavat funktiot ovat erillään

3.6 Käyttöliittymän suunnitteluperiaatteet

Hyvän graafisen käyttöliittymän suunnittelu on parhaimmillaankin hyvin hankalaa. Kun asioita voidaan piirtää ruudulle mielivaltaisiin kohtiin ja kaikilla sateenkaaren väreillä, on syytä pysähtyä miettimään mitä eri graafisilla elementeillä halutaan viestiä. Vaikka visuaalinen miellyttävyys onkin hyvän käyttöliittymän tunnuspiirre, on se enemmän seurausta taustalla tehdyistä loogisista päätöksistä tiedon jäsentelyn suhteen.

Käyttöliittymän tulisi ensisijaisesti olla helppokäyttöinen ja intuitiivinen. Elementtien koolla ja kontrastilla voidaan tietoa ja toimintoja järjestää hierarkkisesti loogisiin kokonaisuuksiin, jossa esimerkiksi käyttäjälle tärkeimmät elementit esitetään suurempina tai korkeakontrastisina, kuin vähemmän tärkeät elementit. Tieto ja niihin liittyvät toiminnot on myös hyvä pitää mahdollisimman lähellä toisiaan.

Toiminnallisesti samankaltaisten elementtien esittämistä on hyvä tukea myös visuaalisesti ja johdonmukaisesti koko käyttöliittymässä. Esimerkiksi käyttäjältä syötettävä odottavat tekstikentät ja puhtaasti tietoa ilmaisevat tekstikentät on syytä visuaalisesti erottaa toisistaan, eikä niitä tule mielellään sekoittaa sovelluksen sisällä. [14]

4 TOTEUTUSTAPOJEN TARKASTELU

Toimeksiannon tavoitteena oli siis löytää mahdollisimman hyvä ratkaisu lämpenemätestiprosessin ongelmiin. PicoLogin kehittyneempiä ominaisuuksia tutkimalla löydettiin lupaavia menetelmiä, joilla testikriteerien seurantaa olisi voitu selkiyttää ohjelman sisällä, mutta ratkaisu ei kuitenkaan olisi täyttänyt standardin kriteerejä täysin. Lisäksi raportoinnin osalta PicoLogin mahdollisuudet jäivät mittaushistorian ja kuvaajan tulostukseen.

PicoLog oli kuitenkin muilta osin todettu toimivaksi sovellukseksi lämpenemätestauksen tarpeisiin, joten ratkaisuvaihtoehdoksi mietittiin ensin niin sanotun apuohjelman rakentamista, jota olisi ajettu PicoLogin rinnalla. Apuohjelmalle olisi ulkoistettu ainoastaan testikriteerien seuranta ja raportin tulostaminen. PicoLogin ja apuohjelman välille oli kuitenkin löydettävä ensin tiedonsiirtoprotokolla.

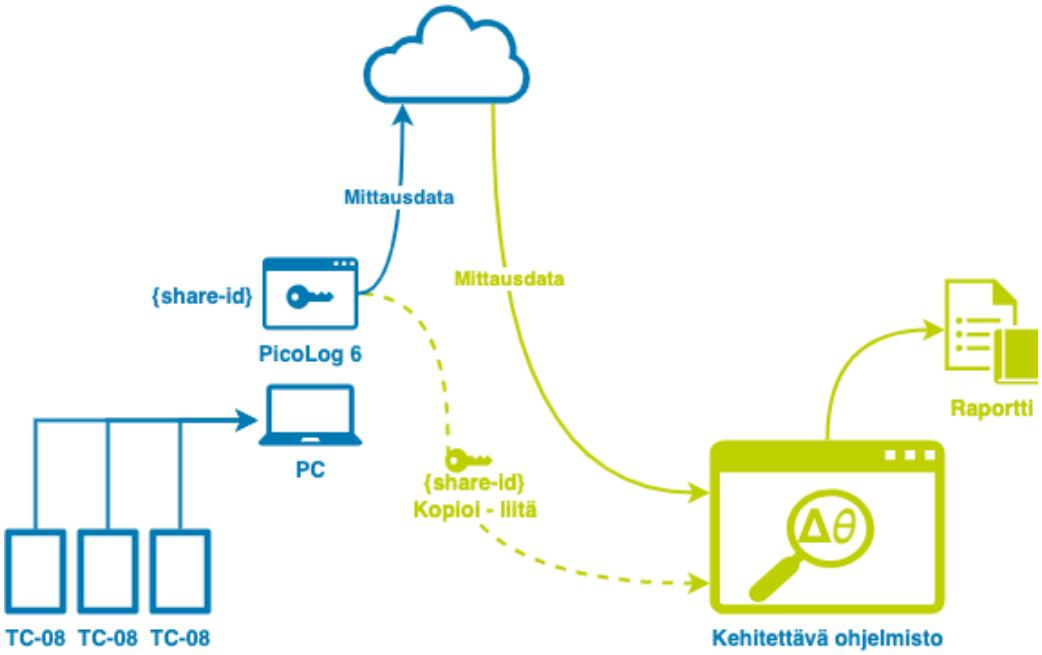
Apuohjelman mahdollisia tiedonsiirtoprotokollia tunnistettiin kolme erilaista:

- PicoCloud API
- PicoCloud Web API
- PicoLog 5 Microsoft DDE:n avulla.

Ratkaisua etsittiin ensin näiden protokollien avulla, sillä siten olisi kaikki sovelluskehitykseen käytettävissä ollut aika voitu kohdistaa täysin ongelmakohtiin. Mahdollisimman laadukkaan ja helppokäyttöisen ratkaisun vuoksi päädyttiin kuitenkin lopulta täysin itsenäisen ja VEO:n tarpeisiin räätälöityyn ohjelmiston kehittämiseen PicoSDK:n avulla.

4.1 PicoCloud API

Versiosta 6.2.0 eteenpäin, PicoLogilla voidaan syöttää mittausdataa suoraan PicoCloud-pilvipalveluun, jonka käyttöönotto vaatii ainoastaan käyttäjätilin luomisen. Mittausdataa voidaan seurata PicoCloudin web-palvelun kautta, tai pyytää datapaketteja HTTP-protokollaa käyttäen omiin räätälöityihin sovelluksiin. [15].



Kuvio 6. Virallisen PicoCloud API:n hyödyntäminen.

Apuohjelma ajateltiin rakennettavan yllä olevan kuvion mukaan, jossa vihreällä on kuvattu prosessia varten kehitetyt uudet elementit. Lämpötilatiedot olisi kopiotu apuohjelmaan lähes reaaliajassa PicoTechin pilvipalvelusta.

PicoCloud API on erittäin pelkistetty ja koostuu vain kahdesta erilaisesta HTTP GET -pyynnöstä:

1. Kaikkien kanavien tämänhetkinen tilanne
2. Tietyn kanavan historia.

Kaikkien kanavien hetkellinen tilanne saadaan tekemällä HTTP GET -pyyntö osoitteeseen: https://api.picolog.app/v1/channels/{share-id}, jossa share-id korvataan PicoLogissa generoidulla mittaussession avaimella.

Paluuna saadaan esimerkiksi seuraavanlainen json-vastaus (kuvio 7). Id on kanavakohtainen uniikki tunnus, jota tarvitaan, jos halutaan tarkastella yhden kanavan mittaustuloksia aikaan sidottuna.


```
[
	{
		"id": "ida3691dcc-1a42-472e-8e7c-76da82c67173",
		"deviceSerial": "AO023/869",
		"channelIdentifier": "ch1",
		"name": "Channel 1",
		"interval": 1,
		"value": null,
		"multiplier": 1
	},
	{
		"id": "idc349d396-abcb-431d-81b9-3325874b7150",
		"deviceSerial": "AO023/869",
		"channelIdentifier": "ch2",
		"name": "Testaus",
		"interval": 1,
		"value": null,
		"multiplier": 1
	}
]
```


Kuvio 7. PicoCloud API:n testivastaus, kaikki kanavat.

Jos halutaan tarkastella ainoastaan yhden kanavan tuloksia tietyllä aikavälillä, tehdään GET-pyyntö osoitteeseen: https://api.picolog.app/v1/samples/{share-id}/{channel-id}/{start-time}/{end-time}, jossa share-id on sama kuin yllä, channel-id on yllä saatu kanavakohtainen uniikki tunnus, sekä start- ja end-time puolestaan haluttu aikaväli UTC unix epoch -muodossa [16]. Esimerkki pyynnön paluuviestistä alla (kuvio 8).


```
[
	["2021-05-04T05:55:46Z": 24.6],
	["2021-05-04T05:55:47Z": 24.7],
	["2021-05-04T05:55:48Z": 24.8],
	["2021-05-04T05:55:49Z": 24.8]
]
```


Kuvio 8. PicoCloud API:n mallivastaus, yksittäinen kanava.

Unix epoch time on monissa sovelluksissa käytetty ajan tallennustapa, jossa aika esitetään kuluneina sekunteina ajanhetkestä 1.1.1970 klo 00:00 lähtien [17]. Esimerkiksi tätä tekstiä kirjoitettaessa aika unix-muodossa on 1709496976 (UTC).

Virallista API:a käytettäessä datapyyntöjä ei autentikoida missään vaiheessa, joten share-id-avain on pidettävä turvassa, sillä sen avulla ketä tahansa voi yksinkertaisesti pyytää mittaussession koko tiedot.

Jostain syystä ensimmäisten datapakettien saannissa serveriltä kesti jokaisella testikerralla kauan, useista kymmenistä sekunneista muutamiin minuutteihin. Tämä olisi vaikuttanut negatiivisesti lämpenemätestauksen apuohjelman käyttäjäkokemukseen. Lisäksi kaikkien kanavien datapakettiin ei sisälly aikaleimoja, joten mittaustulosten sijoitus aikajanaan perustuisi lähinnä hyvään arvaukseen. Yksittäisten kanavien datapakettien käyttö taas aiheuttaisi merkittävää dataliikennettä, sillä pienjännitekeskusten lämpenemätestauksissa käytetään usein jopa yli kolmeakymmentä kanavaa kerrallaan ja testit ovat useiden tuntien mittaisia [18].

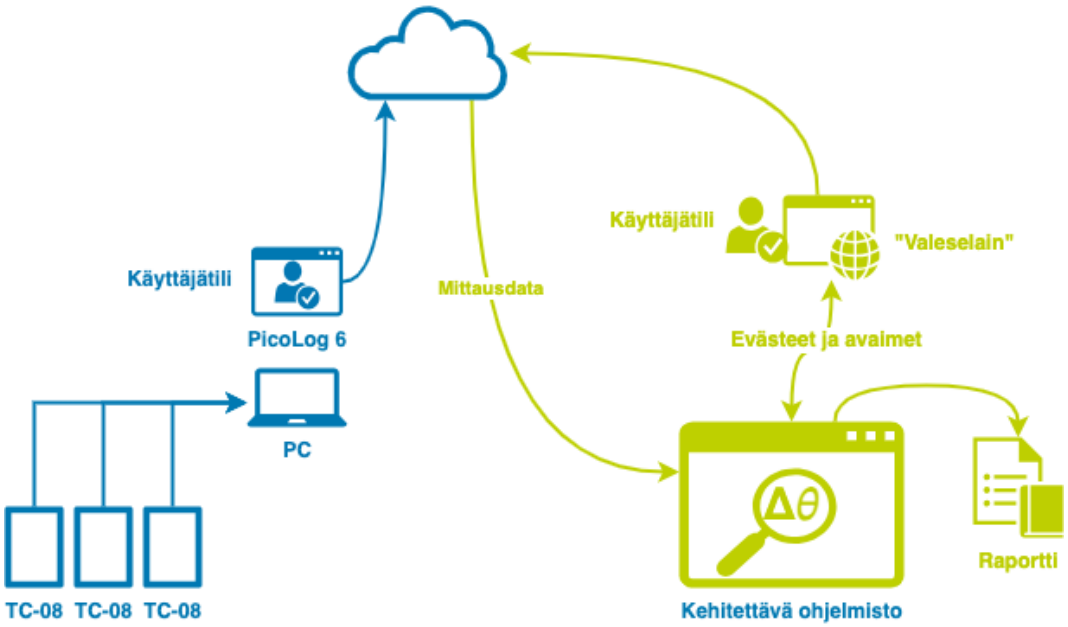
PicoCloud API:lla toteutettu apuohjelma olisi suositeltava valinta, mikäli yhtäaikaisia kanavia olisi vain muutama, ja jos ohjelman kehityskustannukset täytyisi huomioida. Tällä tekniikalla toteutettu ratkaisu vaatisi kahden ohjelman yhtäaikaista käyttöä, minkä arvioitiin olevan ei-optimaalinen käyttäjäkokemuksen kannalta.

Taulukko 1. PicoCloud API:n ominaisuudet.

PicoCloud API	
+ Nopea sovelluksen kehittäminen	- Hitaus ensimmäisten mittaustulosten kohdalla
+ Yksinkertainen	- Vaatii internetyhteyden
	- Kahden ohjelmaikkunan yhtäaikainen käyttö
	- Share-id:n siirto
	- Aikaleimatun datan siirto vain kanava kerrallaan

4.2 PicoCloud Web-API

PicoCloudin web-käyttöliittymän dataliikennettä seuraamalla löytyi toinenkin, joskin epävirallinen tapa päästä pilvidataan käsiksi [19]. Webkäyttöliittymä on toteutettu omalla suljetulla API:lla, joka tarjoaa huomattavasti kattavammat datapaketit kuin virallinen. Esimerkiksi yhdellä GET-pyynnöllä on mahdollista saada kaikkien laitteiden ja kanavien yksityiskohtaiset tiedot aikaleimoilla. Lisäksi mittaussessioita ei tarvitse erikseen jakaa share-id:llä, vaan kaikki mittaussessiot ovat heti saatavilla.



Kuvio 9. PicoCloud Web-API:n käyttö.

Testeissä tämä toimi erittäin nopeasti ja tarjosi kaiken tarvittavan datan yhdellä kutsulla. Kääntöpuolena datapakettien yksityiskohtaisuus johti jopa runsaudenpulaan ja halutun datan suodattaminen paketeista olisi ollut tarpeen. Tämä API ei vastaa ilman aktiivisia sisäänkirjautumisevästeitä, joten apuohjelman tapauksessa nämä pitäisi mittaussession käynnistyksen yhteydessä kalastaa valeselaimella.

Koska tämä menetelmä on tarkoitettu PicoCloudin webkäyttöliittymän tarpeisiin, ei Pico Techillä ole velvollisuuksia pitää sitä muuttumattomana, tai ilmoittaa muu-

toksista käyttäjilleen. Pienetkin muutokset API:in johtaisivat todennäköisesti apuohjelman rikkoutumiseen. Näistä syistä sitä ei voi suositella apuohjelman datankeruumenetelmäksi.

Taulukko 2. PicoCloud Web-API:n ominaisuudet.

PicoCloud Web-API	
+ Kaikki data heti saatavilla	- Muutokset API:ssa johtaisivat apuohjelman rikkoutumiseen
+ Nopea	- Vaatii internetyhteyden
+ Aikaleimat mittaustuloksissa	

4.3 Dynamic Data Exchange

PicoLogin dokumentaatiosta löytyi myös maininta DDE:n käytöstä datasiirtoon. [20] Apuhojelman kehitys tätä protokollaa käyttäen olisi ollut kaikista tutkituista menetelmistä ylivoimaisesti yksinkertaisin. Tämän menetelmän etuna olisi myös, että datasiirto on täysin paikallista, eikä riippuvaista verkkoyhteyksistä.

Alla olevassa kuviossa on kuvattu DDE:llä toteutettavan apuohjelman toimintaperiaate, jossa vihreällä värillä on kuvattu prosessia varten kehitettävät uudet elementit.

Kuvio 10. Microsoft Dynamic Data Exchange.

Datapaketteja voi noutaa määrittelemällä ainoastaan kolme parametria: sovellus, aihe ja kohta (kuvio 11). DDE on Microsoftin kehittämä datansiirtoprotokolla, joka on ollut käytössä jo Windows 2.0:sta lähtien [21]. DDE:tä tukevien sovellusten dataa voi näppärästi siirtää esimerkiksi Exceliin reaaliajassa.

```
=PLW|Current!Value
```

Kuvio 11. Esimerkki PicoLog 5 DDE -kutsusta Excelissä.

Vaikka DDE:n käyttö onkin hyvin yksinkertaista, vaatii se tässä tapauksessa PicoLog versio 5:n käyttöä, joka on tätä kirjoitettaessa jo lähes 30 vuotta vanha sovellus. PicoLog 5:n käyttöliittymä on myös hyvin monimutkainen ja vanhentunut verrattuna versioon 6. Tämä yhdistelmä johtaisi vieläkin heikompaan käyttäjäkokemukseen verrattuna kahteen edelliseen tekniikkaan. Lisäksi vanhojen ohjelmien ja protokollien hyödyntäminen on kyseenalaista yhteensopivuuksien ja tietoturvallisuuden kannalta.

Taulukko 3. DDE:n ominaisuudet.

Dynamic Data Exchange	
+ Helppo protokolla käyttää	- Vaatii vanhan isäntäohjelmiston
+ Nopea	- Yhteensopivuusongelmat tulevaisuudessa

4.4 PicoSDK

Pico Tech ylläpitää erittäin kattavaa SDK:ta tuotteilleen, joka helpottaa räätälöityjen sovellusten kehitystä. PicoSDK on ohjelmistokehityspaketti, jonka avulla voidaan tehdä räätälöityjä mittaussovelluksia Pico Technologyn eri datakeruulaitteille, kuten TC-08:lle. Vaihtoehtoisesti SDK:n avulla datakeruulaitteet voidaan yhdistää esimerkiksi Excel-, tai Mathlab-sovelluksiin.

Itse SDK:n käyttö sellaisenaan vaatisi C-ohjelmointikielen käyttöä, mutta SDK:n ympärille on myös tarjolla niin kutsutut *wrapperit*, jolla C-kieliset kutsut saadaan käyttöön myös muita ohjelmointikieliä käytettäessä. [22]

Tällä tavalla kehitetty sovellus on viimeistä piirtoa myöten täysin kehittäjän ja tilaajan räätälöitävissä, joskin on huomioitavaa, että tällöin koko sovellus on tehtävä alusta asti laitteistotasolta aina graafiseen käyttöliittymään, eikä PicoLogin valmiita ominaisuuksia voida hyödyntää miltään osin. Ylläpitoon ja jatkokehitykseen tulee myös varata resursseja ja erityisosaamista.

SDK:ta hyödyntäen päästään silti saumattomimpaan käyttäjäkokemukseen, sillä koko prosessi datankeruusta raportointiin on yhden ja saman ohjelman sisällä.

Taulukko 4. PicoSDK:n ominaisuudet.

PicoSDK	
+ Koko prosessi saman ohjelman sisällä	- Vaatii paljon kehitysresursseja
+ Saumtaon käyttäjäkokemus	- Ylläpito vaatii erityisosaamista
+ Täysin räätälöitävissä	

4.5 Toteutustapojen vertailu

Kaikilla tutkituilla lähestymistavoilla voidaan teoriassa toteuttaa ratkaisu, joka vastaa toimeksiannon ongelmakohtiin ja automaattisen raportoinnin osalta toteutus olisikin identtinen eri lähestymismallien välillä. Käytännön toteutuksessa ja käyttäjäkokemuksessa voidaan kuitenkin havaita suuria eroja erityisesti datankeräyksen osalta, eli siinä miten lämpötilamittaukset saadaan siirrettyä laitteistolta eteenpäin. Kaikissa vaihtoehdoissa, poislukien SDK, täytyisi käyttää kahta eri ohjelmaa rinnakkain, joka voisi helposti johtaa lämpenemätestauksen epäonnistumiseen tahattoman väärinkäytön seurauksena.

Jos toimeksiantona olisi ainoastaan ollut automaattinen raportointi, olisi esimerkiksi DDE:tä tai virallista API:a käyttäen päästy tyydyttävään lopputulokseen, mutta toive, tai vaatimus testin päättymisestä automaattisesti standardin kriteerien täyttyessä kallistaa valinnan selvästi SDK:n käytön puolelle.

Eri lähestymistavoista tehtiin lisäksi painotettu pisteytys eri ominaisuuksien perusteella tukemaan valittua kehitysmenetelmää. Kehitysajalle ei pisteytyksessä annettu lainkaan painoarvoa, sillä sovelluksen kehitys tehdään osana tätä opinnäytetyötä.

Pisteytys on skaalalla 1–5, jossa 1 on heikko ja 5 erinomainen. Ominaisuuksia vertailtiin muun muassa seuraavin kriteerein:

Nopea kehitysaika: Työtuntien määrä suhteutettuna valmiiseen sovellukseen. Esimerkiksi PicoCloud API:a käyttämällä kaikki kehitysaika voidaan suoraan käyttää ongelmanratkaisun pääkohtiin, eli esimerkiksi automaattisen raportoinnin ohjelmointiin.

Toimintavarmuus: Käytetystä protokollasta johtuvat epävarmuustekijät. Esimerkiksi API:a käyttämällä ohjelman toiminta on aina riippuvainen tietoliikenneyhteyksistä.

Tietoturva: Jos käytetty datansiirtoprotokolla on tietoliikenneyhteyksien varassa, kuinka se on suojattu? Jos käytetään paikallisia siirtoprotokollia, kuinka moderneihin käytäntöihin ne perustuvat?

Muokattavuus: Kuinka pitkälle ohjelma on räätälöitävissä? Suoraan verrannollinen siihen, miten yksityiskohtaisia datapaketteja on mahdollista noutaa

Nopeus: Datansiirto antureilta ohjelmaan, sekä käytön nopeus mittaustilanteen alusta valmiiseen raporttiin.

Riippuvuudet: Kolmannen osapuolen rajapinnoista riippuvuudet alentavat pisteitä, jos ne eivät ole korvattavissa rikkomatta ohjelmaa.

Taulukko 5. Toteutustapojen vertailu, painoarvot.

		Kpl	Painoarvo
K	Nopea kehitysaika	0	0,00
TV	Toimintavarmuus	10	0,24
TT	Tietoturva	12	0,29
H	Helppokäyttöisyys	6	0,14
M	Muokattavuus	8	0,19
N	Nopeus	4	0,10
R	Riippuvuudet	2	0,05
YHT		**42**	**1**

	K	TV	TT	H	M	N	R
K		TV	TT	H	M	N	R
TV	TV		TT	TV	TV	TV	TV
TT	TT	TT		TT	TT	TT	TT
H	H	TV	TT		M	H	H
M	M	TV	TT	M		M	M
N	N	TV	TT	H	M		N
R	R	TV	TT	H	M	N	

Taulukon oikealla puolella ominaisuudet aseteltiin vastakkain, jossa voittajaominaisuus kirjattiin taulukkoon. Voittajaominaisuuksien määrä laskettiin yhteen, josta saatiin ominaisuuksille suhteellinen painoarvo.

Taulukko 6. Toteutustapojen vertailu, pisteet.

	PicoCloud API		PicoCloud Web API		DDE		PicoSDK	
	1-5	P	1-5	P	1-5	P	1-5	P
Nopea kehitysaika	4	0,0	3	0,0	5	0,0	2	0,0
Toimintavarmuus	3	0,7	2	0,5	4	1,0	5	1,2
Tietoturva	3	0,9	3	0,9	4	1,1	5	1,4
Helppokäyttöisyys	3	0,4	4	0,6	2	0,3	5	0,7
Muokattavuus	3	0,6	3	0,6	3	0,6	5	1,0
Nopeus	2	0,2	3	0,3	4	0,4	5	0,5
Riippuvuudet	3	0,1	2	0,1	3	0,1	5	0,2
YHT		2,8		2,8		3,3		**4,8**

Antamalla eri toteutusvaihtoehdoille painotetut pisteet kallistui suunta entistäkin varmemmin PicoSDK:n kannalle. Hieman yllättäen DDE pärjäsi tässä vertailussa myös hyvin, vaikka todellisuudessa PicoCloud API:in perustuva ratkaisu olisi ollut käytännössä ainoa toinen mahdollinen vaihtoehto.

4.6 Sovelluskehityksen rajaus

PicoSDK:n valinta toteutusvaihtoehdoksi tarkoitti käytännössä sovelluksen kehittämistä täysin nollasta, eli aina laitteistotasolta graafiseen käyttöliittymään asti. Koska alkuperäisessä ongelmankuvauksessa etsittiin ratkaisua ainoastaan PicoLogista puuttuviin ominaisuuksiin, oli projektin laajuuden hallitsimiseksi tärkeää ensin tunnistaa mitkä PicoLogin olemassa olevista ominaisuuksista oli välttämätöntä luoda uuteen kehitettävään sovellukseen. Näihin katsottiin kuuluvan:

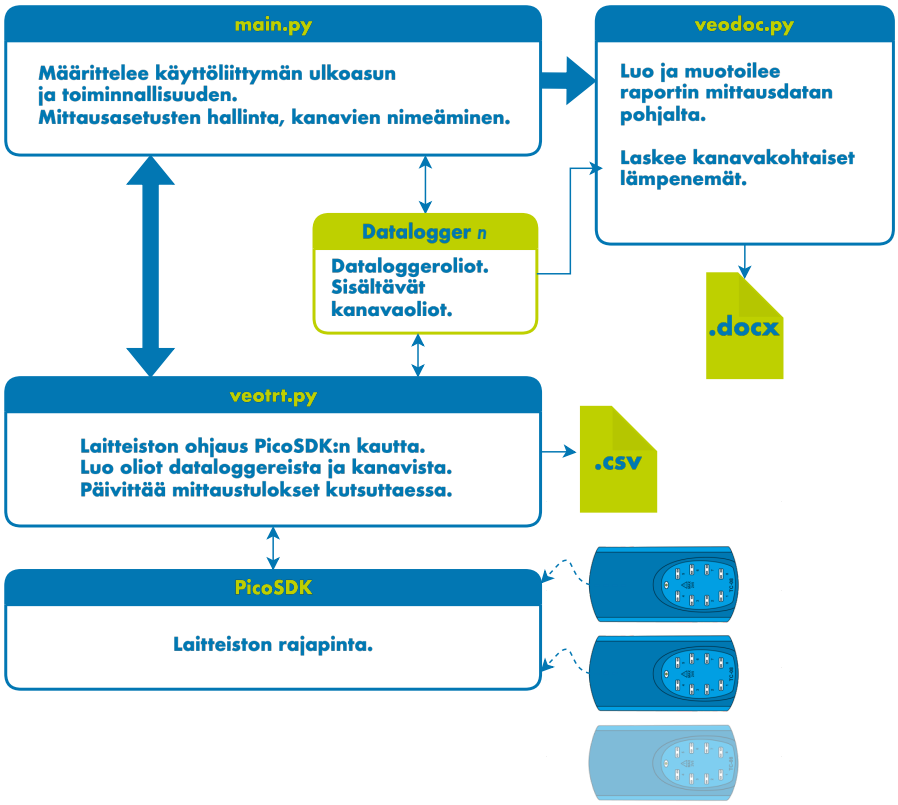
1. Laitteiston kommunikointi
2. Kanavien asetukset ja nimeäminen
3. Lämpötilojen seuranta numeerisesti ja graafisesti
4. Mittaushistorian tallennus ja tulostus.

Tämän lisäksi alkuperäisen ongelmakuvauksen mukaiset lämpenemän seuranta ja raportointi kuuluivat rajauksen piiriin. Muut kehitysehdotukset ja ideat kerättiin muutoslistalle opinnäytetyön jälkeistä jatkokehitystä varten.

5 SOVELLUKSEN RAKENNE

Sovellus on jaettu kolmeen osaan ohjelmoinnin ja ylläpidon helpottamiseksi. Ensimmäinen osa *veotrt.py*, sisältää mittaustulosten keräykseen ja käsittelyyn liittyvät funktiot. Toinen osa *main.py*, sisältää kaikki graafiseen käyttöliittymään liittyvät toiminnot ja kolmas osa veodoc.py huolehtii raportointiin liittyvästä datan käsittelystä ja muotoilusta, sekä tulostaa lopuksi valmiin mittausraportin.

Veotrt.py ja veodoc.py tuodaan moduuleina käyttöliittymään, josta niiden funktioita kutsutaan tarvittaessa. Se kutsuu vuorollaan PicoSDK:n funktioita, joilla tehdään laitteistoon liittyvät käskyt ja asetukset. Veotrt.py myös määrittelee luokat, joiden pohjalta muun muassa kanavista ja datakerääjistä tehdään olioita. Olioita käsitellään luonnin jälkeen instansseina käyttöliittymän eri osissa.



Kuvio 12. Sovelluksen rakennekaavio.

Sovellus pohjautuu proseduraalisen ja olio-ohjelmoinnin yhdistelmään. Veotrt.py ja veodoc.py on käytännössä toteutettu täysin proseduraalisen ohjelmoinnin periaatteiden mukaan, sillä niiden toiminta on helposti jäsenneltävissä sarjana yksittäisiä komentoja.

5.1 Kirjastot

Sovelluksessa on hyödynnetty seuraavia Pythoniin sisäänrakennettuja, sekä *ulkoisia kirjastoja. Kirjastojen tarkat versiot on määritelty lähdekoodin mukana kulkevassa *requirements.txt* tiedostossa.

Taulukko 7. Käytetyt ohjelmointikirjastot.

Nimi	Selite ja käyttö sovelluksessa
ctypes	C-yhteensopivien tietotyyppien käyttö ja funktioiden kutsuminen DLL-tiedostoista, tai jaetuista kirjastoista. [23] Tässä sovelluksessa sitä käytetään muuntamaan pythonin tietotyypit PicoSDK:n tarvitsemiin C-yhteensopiviin tietotyyppeihin.
enum	Tarjoaa tuen luetelmille (enumerations). Luetelmat ovat joukko symbolisia nimiä, jotka on sidottu yksilöllisiin arvoihin. [24] Tässä sovelluksessa sitä on käytetty rajaamaan esimerkiksi termoparin mahdolliset nimet ja arvot SDK:n ymmärtämiin vastineisiin.
datetime	Päivämäärien ja aikojen käsittelyyn erikoistunut kirjasto. Tässä sovelluksessa hyödynnetään kirjaston *datetime*- ja *timedelta*-luokkia. *Datetime* -luokalla määritetään mittauksen aloituksen aikaleima ja *timedelta* -luokalla määritetään muun muassa mittaustulosten haun aikaväli.
itertools	Toimintoja, jotka luovat iteraattoreita tehokkaaseen silmukointiin. Tässä sovelluksessa on käytetty *chain*-funktiota, jolla voidaan yhdistää useita iteraattoreita yhdeksi pitkäksi iteraattoriksi. Esimerkiksi lista listoista, voidaan yhdistää näin yhdeksi pitkäksi listaksi.

Nimi	Selite ja käyttö sovelluksessa
***numpy**	Yksi- tai moniulotteisten taulukkojen ja matriisien käsittelyyn erikoistunut kirjasto. Tässä sovelluksessa kirjastoa käytetään muun muassa muuntamaan PicoSDK:n palauttamat yksiulotteiset taulukot helpommin käsiteltävään muotoon ja siistimään tyhjät rivit kyseisistä taulukoista.
***pandas**	Tehokkaat ja helppokäyttöiset tietorakenteet, joilla voi käsitellä suuria määriä monimutkaista dataa. Tässä sovelluksessa kanavakohtaiset mittaustulokset tallennetaan *Series*-tietorakenteena, eli yksiulotteisena taulukkona, jossa riveillä on määritettävä indeksi. Kaikkien kanavien mittaustulokset yhdistetään lopuksi *DataFrame*-tietorakenteeksi.
***flet**	Vuorovaikutteisten sovellusten rakentamiseen tarkoitettu käyttöliittymäkehikko. Työpöytä-, web- ja mobiilisovellukset yhdellä lähdekoodilla. Perustuu Googlen Flutter-käyttöliittymäkehikkoon. Tämän sovelluksen graafisen käyttöliittymän perusta.
***docxtpl**	Word-dokumenttien luontiin ja muokkaukseen erikoistunut kirjasto. Wordilla luodun pohjadokumentin tietoja voidaan korvata jinja2 tageilla. Tämän sovelluksen raporttiautomaatin selkäranka. Docxtpl:n avulla mm. täytetään raportin tekniset tiedot, luodaan mittaustuloksista taulukko, sekä liitetään mittaustulosten viivakaavio raporttiin.
***matplotlb**	Datan graafiseen esittämiseen erikoistunut kirjasto, jolla voidaan luoda sekä staattisia, että interaktiivisia kaavioita. Luo mittausraportin viivakaavion.

5.2 PicoSDK

PicoSDK mahdollistaa C-kielisen laitteistoajurin komentamisen suoraan python-ohjelmasta [25]. C-kielisiä funktioita kutsuttaessa Pythonilla ohjelmoidusta ohjelmasta on kuitenkin otettava huomioon näiden kielien perusteelliset erot datatyyppien ja muuttujien luomisen suhteen. Pythonissa datatyyppejä voi muuttaa lennosta ja datatyypit ovat hyvin joustavia; esimerkiksi kokonaisluvuille on vain yksi datatyyppi (*int*), joka voi sisältää sekä positiivisia että negatiivisia lukuja, eikä luvun koolle bitteinä tarvitse määrittää rajaa. [26]. Esimerkiksi C:llä ohjelmoitaessa on muuttujalle aina määriteltävä tarkasti, kuinka monta bittiä arvon tallentamiseen tarvitaan, sekä onko syytä ottaa huomioon mahdolliset negatiiviset luvut. PicoSDK:n funktioita kutsuttaessa onkin aina muutettava tarvittavat muuttujat sopiviksi *ctypes*-kirjaston avulla. Hyväksytyt datatyypit on määritelty PicoSDK:n manuaalissa funktiokohtaisesti [27].

TC-08-datalogger tukee kolmea eri toimintamuotoa; streaming, single ja legacy, joita kaikkia voidaan hyödyntää ajurin avulla.

Streaming tarkoittaa, että dataloggeri tekee mittauksia itsenäisesti oman sisäisen kellonsa mukaan ja tallentaa ne omaan puskurimuistiin, josta ajurilla voidaan tarpeen mukaan hakea katkeamaton sekvenssi mittaustuloksia. Puskurimuisti on kuitenkin hyvin rajallinen, joten on pidettävä huoli siitä, että ohjelmisto ehtii noutaa tulokset riittävän usein. Manuaalissa suositellaan, että tuloksien hakuväli ei ylitä näytteenottotaajuutta kolminkertaisesti.

Single-moodissa mittaustulokset tuotetaan ainoastaan pyydettäessä. Näytteenoton hetki on täysin riippuvainen mittausta kutsuvan ohjelmiston kellosta ja muiden prosessien kuormasta. Jos kaikki kanavat ovat käytössä, tämän funktion kutsumiseen kuluu aina 900 millisekuntia.

Legacy-moodia tarvitaan, jos halutaan käyttää vanhempia sarjaporttiin kytkettäviä TC-08 dataloggereita.

Lämpenemätestit eivät aseta korkeita vaatimuksia näytteenottotaajuuden osalta, joten single-moodi näytti ensin luonnolliselta vaihtoehdolta. Tämän ohjelmiston osalta päädyttiin kuitenkin käyttämään **streaming**-moodia, sillä lämpenemätesteissä käytetään suurta määrää kanavia dataloggeria kohden ja käyttöliittymän responsiivisuus koettiin tärkeäksi. Näin mittaustulosten aikaavievä prosessi saadaan ulkoistettua dataloggereille ilman, että ohjelmistoa tarvitsee monimutkaistaa asynkronisen tai rinnakkaisprosessien myötä.

PicoSDK:n kattavasta funktiolistasta hyödynnettiin vain tämän sovelluksen kehityksen kannalta olennaisia toimintoja. Ne on listattu alla, suluissa merkittynä funktion tarvitsemat sisääntulot.

usb_tc08_open_unit()
Etsii kaikki kytketyt TC-08 dataloggerit.

usb_tc08_get_unit_info(kahva, devinfo)
Palauttaa dataloggerin sisäiset tiedot, kuten sarjanumeron.

usb_tc08_set_mains(kahva, taajuus)
Asettaa häiriöpoistosuodatuksen halutulle taajuudelle.

usb_tc08_set_channel(kahva, kanavan numero, termopari)
Määrittää kanavakohtaiset asetukset.

usb_tc08_run(kahva, näytteenottotaajuuden minimi)
Asettaa dataloggerin streaming-tilaan.

usb_tc08_stop(kahva)
Pysäyttää streaming-tilan.

usb_tc08_close_unit(kahva)
Sulkee laitteistoyhteyden.

5.3 veotrt.py

Tämä sovelluksen osa vastaa kaikesta laitteistoon liittyvästä ja paketoi PicoSDK:n laitteistokutsut helpommin hallittaviin kokonaisuuksiin. Lisäksi se luo sovelluksen toiminnan kannalta tärkeimmät oliot ja tallentaa lopuksi mittaushistorian csv-tiedostoon.

5.3.1 Luokat

Tc08package

Jokaisesta kytketystä TC-08-dataloggerista tehdään olio funktiossa *format_usb_info* tämän luokan perusteella. Näin dataloggereita voidaan käsitellä omina kokonaisuuksinaan koko ohjelmassa. Datalogger-olioon liitetään lista kanavaolioista ja lisäksi se pitää sisällään muun muassa kyseisen dataloggerin sarjanumeron ja kalibrointipäivämäärän.

```
@dataclass
class Tc08package:
    handle: int
    driver_version: str
    hardware_version: int
    picoapp_version: int
    variant: int
    size: int
    calibration_date: str
    serial_number: str
    channels_in_use: list = field(default_factory=list)
    reject_frequency_hz: int = 50
    running: bool = False
    start_time: datetime = None
```

Kuvio 13. Dataluokka, Tc08package.

Tärkeimmät attribuutit ja niiden selitteet:

handle: Kokonaisluku (> 0), jolla määritetään, mihin dataloggeriin ollaan yhteydessä.

serial_number: Merkkijono, joka kertoo dataloggerin sarjanumeron, jonka avulla identtiset kanavat voidaan sitoa tiettyyn dataloggeriin käyttöliittymässä ja raportissa.

channels_in_use: Lista, joka alustetaan tyhjänä. Kanavien käyttöönoton jälkeen se pitää listaa kaikista dataloggerin kanava-olioista.

reject_frequency: Kokonaisluku, jolla asetetaan häiriösuodatuksen taajuus. Helppolukuisuuden vuoksi tässä käytetään suoraan häiriötaajuutta vastaavaa kokonaislukua, mutta se muutetaan ajurin ymmärtämään muotoon (1 | 0) *set_mains_rejection* -funktiossa.

running: Boolean, asetaan True-tilaan, kun dataloggerit keräävät dataa, muuten False.

start_time: Datetime, alustetaan tyhjänä. Kun mittausprosessi käynnistetään, tähän luodaan alkuajankohta datetime-oliona funktiolla *timestamp_dataloggers*.

Channel

Tämän luokan perusteella jokaisesta dataloggerin kanavasta tehdään olio, joka liitetään dataloggerin olioon listana järjestyksessä. Channel-luokalla määritellään muun muassa kanavan nimi ja termopari, sekä ylläpidetään kanavakohtaista mittaushistoriaa (Kuvio 14.).

```
@dataclass
class Channel:
    number: int
    name: str
    thermocouple: Thermocouple
    measurements: pd.Series = field(default_factory=pd.Series)
    value_now: float = 0.0
    unit: int = 0 # 0: CENTIGRADE, 1: FAHRENHEIT, 2: KELVIN, 3:
RANKINE
    limit: int = 0
```

Kuvio 14. Dataluokka, Channel.

Tärkeimmät attribuutit ja niiden selitteet:

number: Kokonaisluku (1–8), jolla määritetään kanavan järjestysluku. Tämän avulla voidaan esimerkiksi määrittää asetuksia ja hakea mittausdataa tietyltä kanavalta. Lisäksi sitä hyödynnetään graafisessa käyttöliittymässä. Jos kanavan numeroksi määritellään 0, luetaan tällöin TC-08-dataloggerin sisäistä CJC-anturia. CJC-anturia käytetään kompensoimaan termoparin ja dataloggerin lämpenemisestä aiheutuvia mittausvääristymiä. [7]

name: Merkkijono, käyttäjän kanalle antama nimi. Käytetään ainoastaan tekemään mittaustulosten seurannasta ja raportoinnista helppolukuisempia.

thermocouple: Luetelmamuuttuja, joka on määritelty luokassa Thermocouple. Tällä sekä varmistetaan, että kanava-oliolle voidaan antaa ainoastaan ennaltamääritettyjä termoparin arvoja, sekä tehdään tiedon määrittämisestä ja lukemisesta yksiselitteisempää.

measurements: pd.Series, johon mittaustulokset tallennetaan kanavakohtaisesti ennalta määritetyn ajan välein. Sarjan indeksinä on mittaushetken aika millisekunteina ja arvona senhetkinen lämpötila desimaalilukuna.

value_now: desimaaliluku, joka alustetaan arvolla 0,0. Tähän attribuuttiin tallennetaan aina measurements-sarjan viimeisin arvo, jota käytetään graafisessa käyttöliittymässä esittämään senhetkinen lämpötila.

unit: kokonaisluku (0–3), jolla määritetään lämpötilamittausten yksikkö.

limit: kokonaisluku, jolla määritetään suurin sallittu lämpenemä. Tätä muuttujaa hallitaan graafisesta käyttöliittymästä.

Thermocouple

Tämä luokka on tyyppiä luetelmamuuttuja, jolla rajataan ja asetetaan kanavan termopari TC-08-ajurin hyväksymään muotoon. Oletusasetuksena ohjelmassa käytetään J-tyypin termoparia. Jos kanava ei tarvita mittaustuloksissa, tulee tämä asettaa arvoon DISABLED.

```
class Thermocouple(Enum):
    TYPE_B = 66
    TYPE_E = 69
    TYPE_J = 74
    TYPE_K = 75
    TYPE_N = 78
    TYPE_R = 82
    TYPE_S = 83
    TYPE_T = 84
    DISABLED = 32
    TYPE_X = 88 # For voltage readings
```

Kuvio 15. Dataluokka, Thermocouple.

CurrentLogger

Tämän luokan perusteella ohjelmassa luodaan yksi olio virtamittausten tarpeisiin. Kirjoitushetkellä virta-arvot syötetään ohjelmassa vielä käsin, mutta seuraavaan versioon on mahdollista kytkeä esimerkiksi VEOlla käytössä oleva Celsa TNM96 -virtamittari, jolloin myös virtamittaukset saadaan automaattisesti raporttiin.

```
@dataclass
class CurrentLogger:
    name: str
    measurements_all: pd.Series = field(default_factory=pd.Series)
    measurements_l1: pd.Series = field(default_factory=pd.Series)
    measurements_l2: pd.Series = field(default_factory=pd.Series)
    measurements_l3: pd.Series = field(default_factory=pd.Series)
    value_now: float = 0.0
    index_now: int = 0
```

Kuvio 16. CurrentLogger-luokka.

Tärkeimmät attribuutit ja niiden selitteet:

measurements_all: pd.Series, johon virta syötetään ampeereina, kaikkien vaiheiden keskiarvona.

measurements_l1, _l2 ja _l3: pd.Series, ei käytössä tällä hetkellä.

5.3.2 Funktiot

Veotrt.py:n pääasiallinen tarkoitus on hoitaa laitteiston ja ohjelmiston välinen dataliikenne. Tällä moduulilla muun muassa konfiguroidaan mittalaitteet, käynnistetään ja pysäytetään mittaus, sekä kerätään ja paketoidaan mittausdata datakeräimiltä. Vaikka tätä moduulia voi ajaa itsenäisesti, on se ensisijaisesti tarkoitettu käytettäväksi main.py:n käyttöliittymän kautta.

Laitteistoon liittyvät funktiot kutsuvat vuorollaan sisäisesti PicoSDK:n tarjoamia funktioita. Koska dataloggerien ajurit ovat vuorostaan ohjelmoitu C-kielellä, täytyy PicoSDK:n funktioita kutsuttaessa muuntaa tietotyypit Ctypes-kirjastolla sopivaan muotoon. Ohjelman yksinkertaisuuden nimissä ohjelman sisällä muuttujat pidetään aina Pythonille ominaisissa tietotyypeissä ja muunnos tehdään ainoastaan kutsuttaessa PicoSDK:n tietotyyppejä. [27]

Funktioiden ajojärjestys on kriittinen, eikä alla esitetystä järjestyksestä tule poiketa. Kohtia 2 ja 3 voi tarpeen vaatiessa kutsua rajattomasti uudestaan, jos kohtaa 4 ei ole vielä käynnistetty.

1. Initialize_tc08s
2. Set_mains_rejection
3. Set_channels
4. Start_devices
5. Stop_and_close_devices.

initialize_tc08s

Etsii kaikki tietokoneen USB-väylään kytketyt TC-08-dataloggerit PicoSDK:n funktiolla *usb_tc08_open_unit* ja palauttaa niiden kahvat, eli laitteen kytkemisjärjestystä kuvaavat kokonaisluvut listassa. Etsintäprosessi pyörii silmukassa, josta poistutaan vasta kun laitteita ei enää löydy. Tällöin *usb_tc08_open_unit* palauttaa kokonaisluvun 0. [27, s. 15]

get_unit_info_loop

Tämä funktio tarvitsee sisääntulona listan kahvoista ja palauttaa listan kaikista kytketyistä dataloggereista Tc08package-olioina. Jokaista sisääntulolistan kahvaa kohti kutsutaan funktiota get_unit_info, jonka ulostulo suodatetaan funktiolla *format_usb_info* sellaiseen muotoon, joka täyttää luokan Tc08packages määritelmät.

Funktio *get_unit_info* tarvitsee sisääntulona yhden dataloggeria kuvaavan kokonaisluvun ja palauttaa tietueen. Se kutsuu PicoSDK:n *usb_tc08_get_unit_info* -funktiota, jolla tarkasteltavasta dataloggerista saadaan sen sisäiset tiedot talteen. Näistä ohjelman kannalta tärkein ja käytetyin on dataloggerin sarjanumero. [27, s. 21]

Funktio format_usb_info tarvitsee sisääntulona edellisen funktion tietueen, sekä dataloggerin kahvan ja palauttaa näistä dataloggeria kuvaavan olion;

Tc08package. Tämän funktion päätarkoitus on muokata sisääntulon tietue paremmin oliolle sopivaan muotoon.

set_mains_rejection

Tällä funktiolla määritellään dataloggerin häiriönpoistosuodatus. Sisääntulona se tarvitsee listan Tc08package-olioista. Koska häiriönpoistosuodatuksen taajuus on helppolukuisuuden nimissä tallennettu dataloggerin olioon taajuutta vastaavassa muodossa, täytyy se muuttaa PicoSDK:n usb_tc08_set_mains funktiolle sopivaan muotoon. 50 Hz muutetaan kokonaisluvuksi 0 ja 60 Hz kokonaisluvuksi 1.[27, s. 19]

set_channels

Tämä funktio tarvitsee sisääntulona listan Tc08package-olioista ja se määrittelee kaikkien kanavien termoparin asetukset. Sisäisesti se kutsuu PicoSDK:n usb_tc08_set_channel funktiota, joka tarvitsee dataloggerin kahvan, kanavan numeron ja termoparin asetuksen joko kirjaimena, tai kirjainta vastaavana ASCII merkkikoodausstandardin mukaisena kokonaislukuna. Jos termoparin arvoksi asetetaan välilyönti, tai ASCII-merkistön vastaava numero 32, ei kyseistä kanavaa käytetä mittaustuloksissa. Kaikki mahdolliset termoparit on määritelty luokassa Thermocouple.

start_devices

Tällä funktiolla käynnistetään dataloggerien mittaustoiminto ja se tarvitsee sisääntulona listan dataloggereista, sekä halutun mittausintervallin. TC-08 pystyy yhtä kanavaa käytettäessä kymmeneen näytteeseen sekunnissa, mutta kaikkien kanavien hakuun CJC mukaan luettuna on varattava 900 millisekuntia. [27, s. 27] Koska lämpenemätestauksissa on normaalisti suurin osa kanavista käytössä, eikä tiheälle näytteenottotaajuudelle ole tarvetta, on mittausintervalli tässä ohjelmassa määritelty kiinteästi 1 000 millisekuntiin. Tämä tarjoaa riittävän nopeuden käyttöliittymän tarpeisiin ja tekee mittaustulosten aikaleimaamisesta yksinkertaisempaa.

Kutsuu sisäisesti PicoSDK:n usb_tc08_run -funktiota jokaista määriteltyä kanavaa kohden.

stop_and_close_devices

Yksinkertaisuuden vuoksi, tämän funktion sisään on sisällytetty sekä dataloggerien pysäyttäminen, että sulkeminen. Funktio tarvitsee sisääntulona listan Tc08package-olioista.

Kutsuu sisäisesti PicoSDK:n *usb_tc08_stop* ja *usb_tc08_close_unit* -funktioita.

collect_all_data

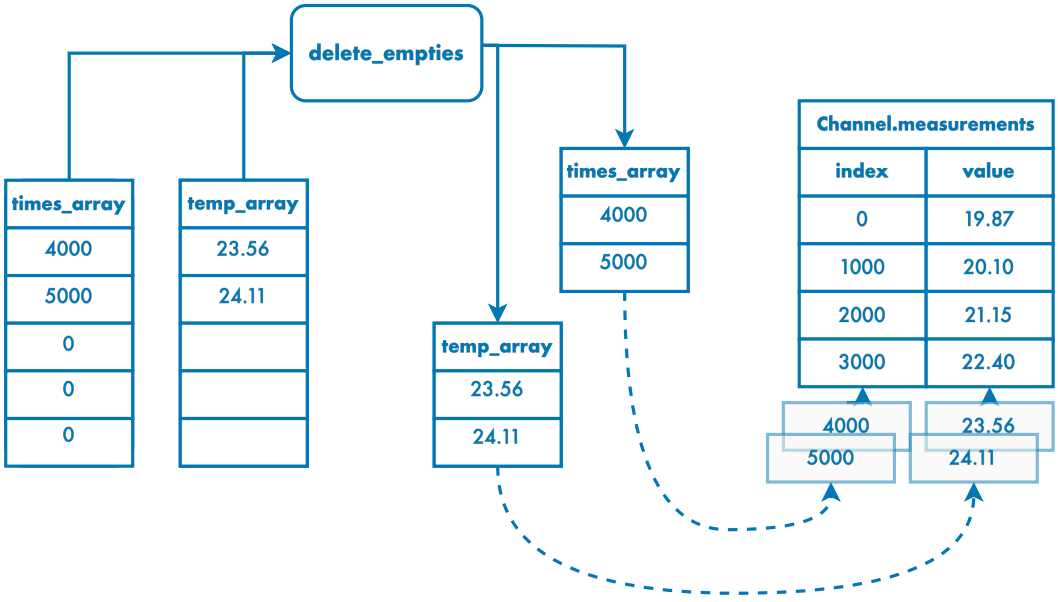
Logiikkafunktio, joka tarvitsee sisääntulona listan Tc08package-olioista. Käy läpi kyseisen dataloggerin kaikki kanavat ja kutsuu funktiota *collect_data_single*, jos kanava on käytössä.

collect_data_single

Tämä funktio kerää mittaustulokset dataloggerin ajurin bufferista ja muuntaa ne taulukkomuotoon. Sisääntulona se tarvitsee Tc08package -olion ja tarkasteltavan kanavan Channel-olion. Dataloggerilta tulevaa mittausdataa käsitellään tämän funktion sisällä monella tapaa, jotta se saadaan helpommin tallennettavaan ja luettavaan muotoon. Kutsuu sisäisesti PicoSDK:n funktiota *usb_tc08_get_temp*, joka palauttaa lämpötilat ja aikaleiman omina yksiulotteisina taulukoinaan.

Koska dataloggerin ajurilta tulevan palautustaulukon koko on aina 15 riippumatta siitä kuinka monta näytettä näytteenottohetkien välissä on ehditty ottaa, täytyy palautustaulukoista ensin poistaa epäkelpo data. Datan siivous tehdään funktiolla *delete_empties*, joka poistaa molemmista ne sarakkeet, joiden aikaleima on 0, tehden poikkeuksen aivan ensimmäisen näytteen kohdalla. Tämän jälkeen aikaleima ja lämpötila siirretään osaksi kanavaolion taulukkoa, joka pitää yllä mittaustulosten historiaa aikajärjestyksessä. Lopuksi tämän taulukon viimeisin arvo siirretään

kanavaolion tämänhetkisen muuttujan arvoksi, joka esitetään graafisessa käyttöliittymässä.



Kuvio 17. Logiikkakaavio, delete_empties.

store_all_as_dataframe

Tämä funktio ajetaan lämpenemämittauksen päätteeksi. Funktion tehtävänä on kerätä kaikkien kanavien mittaustiedot ja järjestää ne taulukkoon, jossa niitä voidaan jatkokäsitellä. Funktio tarvitsee sisääntulona listan Tc08package-olioista ja sen ulostulo on tyypiltään Pandas dataframeolio.

Mittaustulokset ovat aikajärjestyksessä, jonka indeksi on mittausajankohta millisekunteina mittauksen aloitushetkestä. Taulukon sarakkeet nimetään muotoon:

[Kahva]-[Sarjanumero]_[Kanavan numero]-[Kanavan nimi]

add_amperes_to_dataframe

Apufunktio, jolla edellisen kohdan dataframeolioon lisätään vielä virtamittauksen tulokset viimeiseen sarakkeeseen. Käytettäessä manuaalista tapaa syöttää virtamittauksia on niin sanottu näytteenottotaajuus huomattavasti harvempi ja satunnaisempi, kuin automaattisten lämpötilamittausten kohdalla. Tällöin funktio täyttää puuttuvat arvot kopioimalla edeltävät arvot varmistaen lämpötila- ja virtamittauksen yhtäläisen datatiheyden.

save_data_to_file

Tallentaa lämpenemätestauksen mittaushistorian csv-tiedostoon, joka toimii varmuuskopiona ja mahdollistaa mittausdatan erillisen jälkikäsittelyn tarpeen vaatiessa. Funktio tarvitsee sisääntulona listan Tc08package-olioista, sekä edellisen funktion Pandas dataframen.

Csv-tiedoston tallennusnimi on aina mittauksen aloitusaika ja muotoa:

[VVVV-KK-PP_TT-MM-SS]

5.4 veodoc.py

Kun mittausprosessi on saatu onnistuneesti päätökseen, voidaan mittausdata tulostaa automaattisesti valmiiksi raportiksi. Käyttöliittymä paketoi tällöin dataloggerien tiedot DocInfo-olioon ja siirtää olion ja mittaushistorian veodoc.py:n käsittelyyn.

Tämän aliohjelman sisällä täytetään raporttipohja, lasketaan lämpenemät ja tehdään niistä taulukko, sekä tulostetaan mittaushistorian kuvaaja. Lopuksi kaikki elementit liitetään raporttipohjaan ja raporttipohja tallennetaan Word-tiedostona.

5.4.1 Luokat

DocInfo

Raporttigenerointia varten mittaustilanteesta luodaan DocInfo-olio, johon tallennetaan muun muassa käytetyt termoparit, mittauksen aloitus- ja lopetusaika, sekä kaikki raportin taustatietoihin liittyvät tekstikentät.

```
@dataclass
class DocInfo:
    product: str = "Product name"
    tested_assembly: str = "Tested assembly"
    result: str = "XXXX A"
    firstname_lastname: str = "First name Last name"
    meas_software: str = "Software name + version"
    sampling_rate: str = "1000 ms"
    thermocouples: list = field(default_factory=list)
    today: datetime = None
    test_start_time: datetime = None
    test_end_time: datetime = None
    test_duration: datetime = None
    passed_failed: str = "N/A"
```

Kuvio 18. DocInfo-luokka.

5.4.2 Funktiot

Raportti luodaan funktioketjulla ennalta määritetyssä järjestyksessä ja sen pohjana on valmiiksi luotu Word-dokumentti, johon täytettävät kohdat on määritelty jinja2-tageilla.

make_report

Raporttigeneroinnin pääfunktio, joka tarvitsee sisääntulona listan kanavaolioista, koko mittaushistorian dataframena, sekä DocInfo-olion. Se avaa raporttityyppiin sopivan pohjan, täyttää merkityt kohdat, kutsuu alifunktioita taulukon ja kuvaajan luontiin, sekä lopuksi tallentaa raportin uutena dokumenttina docx-muotoon.

table_data

Tämä funktio täyttää raporttipohjassa olevan taulukon, laskee lämpenemät sekä muotoilee tulokset sallittujen raja-arvojen mukaan. Se tarvitsee sisääntuloina edellisessä funktiossa luodun raporttiolion, listan kanavaolioista, sekä DocInfo-olion. Funktio olettaa, että täytettävä taulukko on mallidokumentin viimeinen taulukko.

Kaikki kanavat, joiden nimessä esiintyy sana *ambient,* käsitellään ympäristön lämpötila-antureina. Ne lisätään listaan, jonka keskiarvo lasketaan ja tallennetaan muuttujaan *ambient_average*.

$$ambient\_average = \frac{sum(ambient\_channels)}{len(ambient\_channels)}$$

Muiden kanavien kohdalla lämpenemä lasketaan kanavan lämpötilan ja ympäristön lämpötilan erotuksena. [1, s. 51]

make_graph

Tämä funktio luo raporttiin kuvaajan koko mittaushistoriasta. Sen sisääntuloina ovat mittaushistorian dataframe, sekä dokumenttiolio. Kuvaaja luodaan png kuvatiedostona ja funktio korvaa mallidokumentissa olevan kuvaajan paikkamerkin. Lopuksi dokumenttiolio palautetaan make_report -funktioon, jossa se tallennetaan.

5.5 main.py

Main.py on ohjelman pääkeskus, joka luo graafisen käyttöliittymän ja toimii linkkinä käyttäjän ja mittalaitteiden välillä. Se muuntaa käyttäjän komennot laitteiston ymmärtämään muotoon ja esittää mittaustulokset reaaliajassa ja mahdollisimman helppolukuisesti.

Käyttöliittymän toistuvat elementit, kuten kaikki kanava- ja dataloggerkohtaiset osat on määritelty omissa luokissaan, joiden perusteella luodaan tarvittava määrä olioita kytkettyjen dataloggerien määrän perusteella.

Elementtien keskinäistä hierarkiaa voidaan kuvata seuraavalla tavalla, jossa tähdellä merkityt ovat toistuvia:

- channel_setup_main
    - SetupChannelsTop
    - *SetupChannels
        - *ChannelRow
- measurement_main
    - MeasurementTop
    - Measurement
        - *MeasurementBox
        - LineChart
            - *Dataline
- ReportMaster.

Ohjelma on sisäisesti jaettu kolmeen loogiseen lohkoon: kanavien asetusten määrittelyyn, mittauksen seurantaan ja raportin luontiin. Kaikkiin näihin pääolioihin tuodaan dataloggereiden oliot instansseina, jolla mahdollistetaan tiedonsiirto eri käyttöliittymäelementtien välillä. Käyttöliittymäoliot luodaan vasta, kun kaikille dataloggereille on ensin alustettu olio veotrt.py:n *get_unit_info_loop* -funktiossa.

5.5.1 channel_setup_main

Tässä osiossa määritellään dataloggerien ja niiden kanavien asetukset, kuten kanavan nimi, käytettävän termoparin tyyppi ja lämpenemän raja-arvo. Lisäksi kaikille dataloggereille määritetään yhteinen häiriösuodatuksen taajuus.

Olioita tähän osioon luodaan kolmen luokan perusteella: SetupChannelsTop, SetupChannels ja ChannelRow.

SetupChannelsTop pitää sisällään kaikkia dataloggereita yhteisesti koskevat asetukset ja sen sisälle tuodaan kaikki dataloggeroliot instansseina. Sovelluksen tämänhetkisessä versiossa ainoa dataloggerien yhteinen säätö on häiriösuodatuksen taajuus.

SetupChannels on dataloggerkohtainen luokka, jonka sisälle luodaan aina kahdeksan kappaletta kanavakohtaisia olioita ChannelRow -luokan perusteella. SetupChannels-luokkaan kuuluu myös dataloggerin sarjanumeroa esittävä tekstikenttä ja lisäksi se asettaa kanavaoliot riviin.

5.5.2 measurement_main

Tämä osio vastaa mittaustussession käynnistämisestä ja pysäyttämisestä, sekä päivittää mittausdatan kanavakohtaisesti luettavaan muotoon numeerisina arvoina, sekä kuvaajan muodossa.

Tähän osioon luodaan olioita viiden luokan perusteella: MeasurementTop, Measurement, MeasurementBox, LineChart ja Dataline.

MeasurementTop

MeasurementTop pitää sisällään käynnistys- ja pysäytysnapit, sekä mittauksen aikakestoa kuvaavan tekstikentän. Instansseina siihen tuodaan dataloggeroliot, sekä Measurement-luokasta luotu olio. Käynnistys- ja pysäytysnapit ohjaavat vuorollaan tämän luokan ja koko sovelluksen kahta tärkeintä metodia: start_recording ja stop_recording.

Metodi *start_recording* hoitaa ensin kaikki mittaussession käynnistämistä edeltävät alustukset, kuten syöttää veotrt.py:n ohjausfunktioille käyttäjän määrittelemät datalogger- ja kanavakohtaiset asetukset (kuvio 19). Kun dataloggerien käynnistysfunktiota on kutsuttu, otetaan käynnistysnappi pois käytöstä, jotta mittaussession käynnistämisprosessia ei voida tahattomasti kutsua toistamiseen.

```
def start_recording(self, e):
        veo.set_mains_rejection(self.tc08instances)
        veo.set_channels(self.tc08instances)
        minimum_interval_ms = 1000
        time_now = datetime.now()
        time_last_measured = time_now
        veo.start_devices(self.tc08instances, minimum_interval_ms)
        veo.timestamp_dataloggers(self.tc08instances)
        self.recording = True
        self.start_recording_btn.disabled = True
        self.stop_recording_btn.disabled = False
        sampling_interval = timedelta(seconds=2)
```

Kuvio 19. Ote MeasurementTop-luokan start_recording-metodista.

Mittaussession käynnistysprosessin jälkeen ohjelman ajo siirtyy silmukkaan, jossa kahden sekunnin välein kutsutaan veotrt.py:n collect_all_data -funktiota, sekä Measurement-luokan refresh-metodia, jolla senhetkiset lämpötilatiedot siirretään graafiseen käyttöliittymään. Silmukasta poistutaan vasta, kun mittaussessio halutaan päättää.

```
        try:
            while self.recording:
                time_now = datetime.now()
                if time_now >= (time_last_measured + sampling_interval):
                    veo.collect_all_data(self.tc08instances)
                    self.measurement_content.refresh(e)
                    self.update_elapsed_time()
                    time_last_measured = datetime.now()
```

Kuvio 20. Ote MeasurementTop-luokan start_recording-silmukasta.

Pysäytysnapin painaminen kutsuu metodia stop_recording, joka pysäyttää yllä mainitun silmukan suorittamisen ja sulkee dataloggerien yhteyden. Lopuksi metodi kutsuu veotrt.py:n funktioita, joilla dataloggerolioiden mittaushistoria kootaan yhteen datakehykseen ja sen jälkeen tallennetaan kovalevylle csv-tiedostona.

```
def stop_recording(self, e):
    self.recording = False
    veo.stop_and_close_devices(self.tc08instances)
    veo.timestamp_dataloggers(self.tc08instances)
    self.start_recording_btn.disabled = False
    self.stop_recording_btn.disabled = True
    self.update()
    self.df_main =
veo.store_all_as_dataframe(self.tc08instances, self.df_main)
    self.df_main =
veo.add_amperes_to_dataframe(self.df_main,
self.measurement_content.amperes_object)
    veo.save_data_to_file(self.tc08instances, self.df_main)
```

Kuvio 21. Ote MeasurementTop-luokan stop_recording-metodista.

Measurement

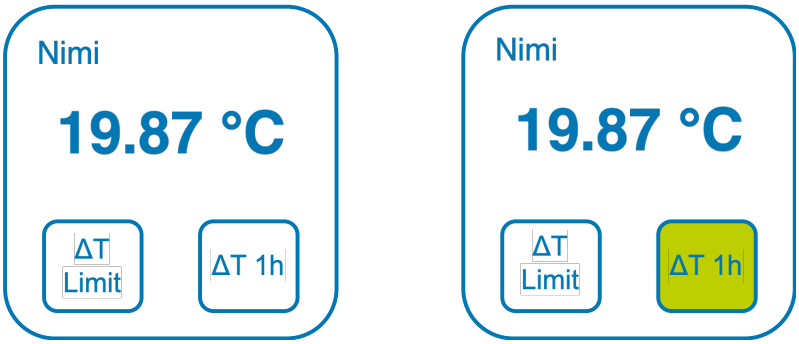
Measurement-luokan tehtävänä on jäsennellä sen sisäiset oliot helposti esitettävään muotoon. Lisäksi sillä on graafisen käyttöliittymän päivittämisestä huolehtiva metodi *refresh* ja lämpenemäehdon toteava metodi *delta_t_one_hour* (kuvio 22).

```
def delta_t_one_hour(self, container, channel: veo.Channel):
    one_hour_ms = 3600 * 1000
    if len(channel.measurements) > 0:
        try:
            if channel.index_now > one_hour_ms:
                earlier_temperature =
channel.measurements.loc[channel.index_now - one_hour_ms]
                temperature_delta =
round((channel.value_now - earlier_temperature), 1)
                container.trt.value = temperature_delta
                if temperature_delta < 1:
                    container.trt.bgcolor = veo_green
```

Kuvio 22. Measurement-luokan metodi delta_t_one_hour.

MeasurementBox

Tämän luokan perusteella jokaisesta kanavasta luodaan olio graafiseen käyttöliittymään, joka ilmaisee kanavan nimen, hetkellisen lämpötilan, lämpenemän, sekä lämpenemälle asetetun rajan. Kun lämpenemän ehto on täytetty, muuttuu sen ilmaisimen väri vihreäksi.



Kuvio 23. MeasurementBox-elementti

LineChart ja Dataline

LineChart on luokka, jonka pohjalta graafiseen käyttöliittymään tehdään kuvaaja, jonka x-akselilla on näytteenottohetki ja y-akselilla jokaisen kanavan lämpötilan hetkellinen arvo. X-akseli skaalautuu automaattisesti mittausprosessin edetessä näyttäen aina koko mittaushistorian alusta loppuun. Jokaisen kanavan viiva kuvaajaan syötetään Dataline-oliona.

Dataline on yhden kanavan mittaushistoriaa kuvaava luokka, jossa kanavan lämpötilatietoja kuvaavat x- ja y-arvot on listattu Flet-kirjaston LineChartDataPoint-olioina. Dataline pitää sisällään new_data -metodin, jolla listaan lisätään uusi datapaketti jokaisella mittaussilmukan kierroksella. Koska monta tuntia kestävän lämpenemätestin aikana kerääntyisi helposti kymmeniä tuhansia datapaketteja jokaista kanavaa kohti, harventaa new_data -metodi listaa tasaisin väliajoin.

5.5.3 ReportMaster

ReportMaster-luokalla rakennetaan raportointiosion graafinen käyttöliittymä, joka koostuu suurimmaksi osaksi tekstikentistä. Lisäksi luokkaan kuuluu nappi, jolla raportointiprosessi lopulta käynnistetään kutsumalla metodia generate_report.

```
def generate_report(self, e) -> None:
    dataframe = self.measurement_top_instance.df_main
    list_of_channels = self.generate_list_of_channels()
    self.thermocouples = self.unique_thermocouples(list_of_channels)
    docinfo = veodoc.DocInfo
    docinfo = self.populate_docinfo(docinfo)
    veodoc.make_report(list_of_channels, dataframe, docinfo)
```

Kuvio 24. ReportMaster-luokan metodi generate_report.

Generate_report -metodi luo veodoc.py:ssä määritellyn DocInfo-olion, johon raportointia varten tarvittavat tiedot kerätään kanavaolioista, sekä käyttäjän tekstikenttiin täyttämistä syötteistä (kuvio 24).

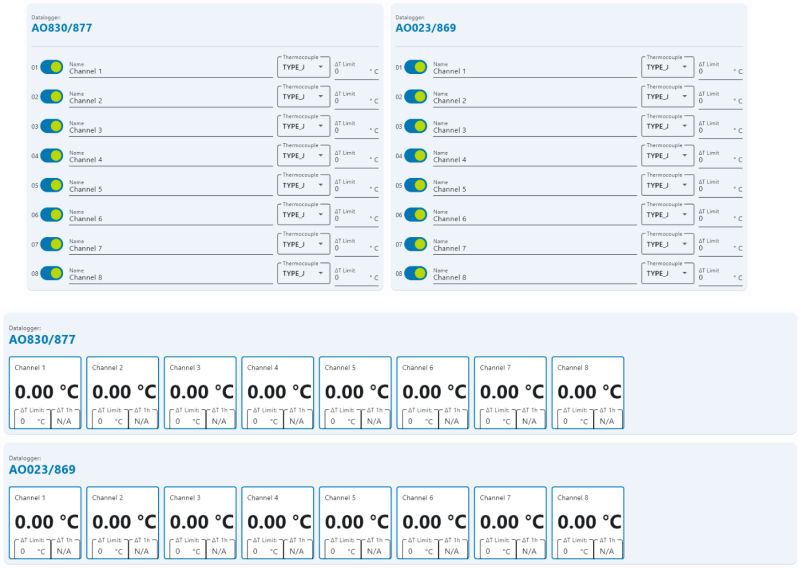
5.6 Graafisen käyttöliittymän suunnittelu

Käyttöliittymä on pyritty suunnittelemaan mahdollisimman yksiselitteiseksi ja helpoksi käyttää noudattaen teoriaosuudessa tutkittuja käyttöliittymäsuunnittelun periaatteita. Esimerkiksi kaikki käyttäjän syöttöä odottavat tekstikentät on luotu käyttäen samaa muotokieltä (kuvio 25), joka eroaa informaatiolähteenä toimivien tekstikenttien muotoilusta. Lisäksi erityistä huomiota vaativan datan kokoa ja kontrastia on nostettu (kuvio 26).

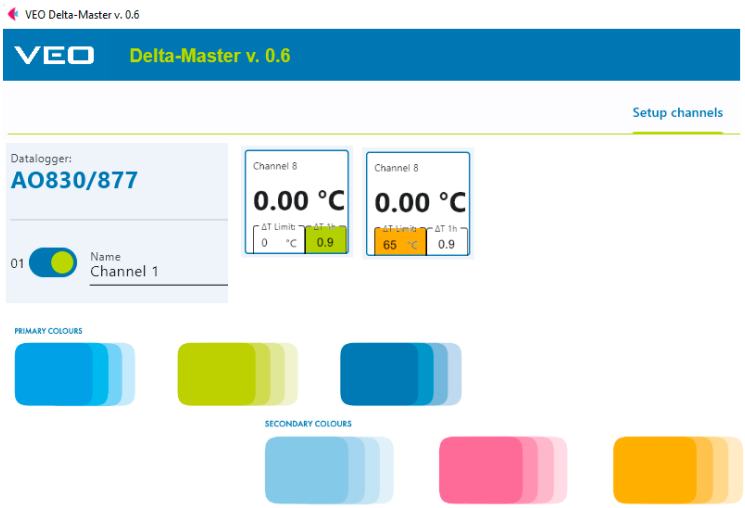
Kuvio 25. Esimerkki datansyöttökentästä.

Kuvio 26. Esimerkki informaatiodatakentästä.

Värejä ja kontrastia on käytetty säästeliäästi ja mahdollisimman johdonmukaisesti. Jokaista loogista kokonaisuutta kuvataan aina vaaleansinisellä taustalla, jotta kokonaisuudet eroavat toisistaan (kuvio 27). Lisäksi väriskaala on rajattu tarkoituksella VEOn graafisessa manuaalissa määriteltyihin pääväreihin, siniseen ja vihreään. Ainoastaan poikkeustilanteita ilmaisemaan väriskaalaa laajennettiin lainaamalla vanhemmassa graafisessa manuaalissa määriteltyjä muita värejä.



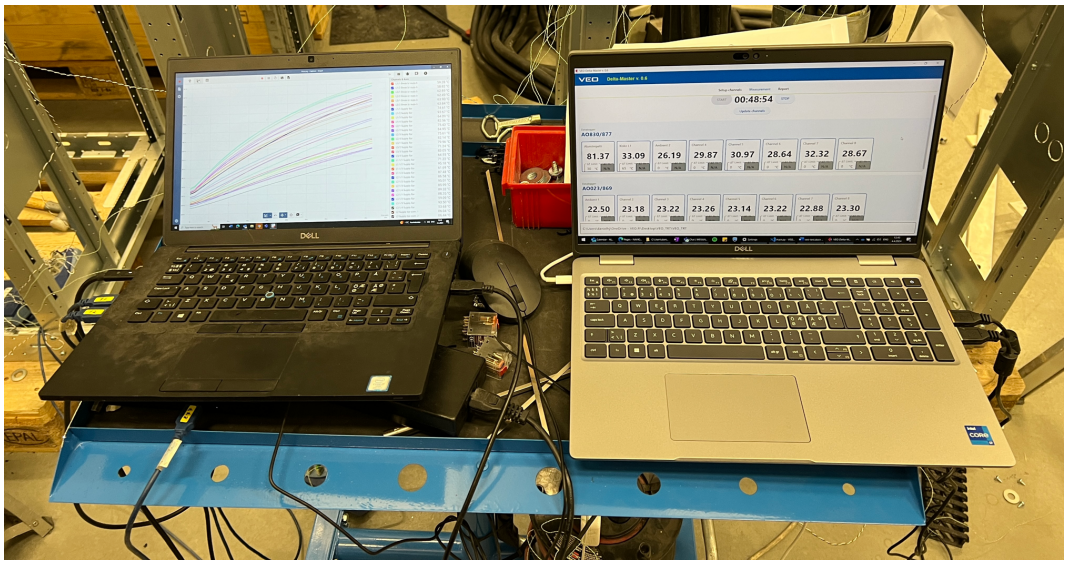
Kuvio 27. Kokonaisuuksien jäsentely värien ja kontrastin avulla.



Kuvio 28. VEOn väriskaala käyttöliittymäsuunnittelun pohjana

5.7 Sovelluksen koeajo

Sovelluksen ensimmäiset koeajot tehtiin VEOn protopajassa käynnissä olevan lämpenemätestin yhteydessä 4.4.2024. Tietokoneeseen kytkettiin kaksi TC-08-dataloggeria ja neljä J- ja K-tyypin termoparia. Ensimmäisessä lyhyessä koeajossa kokeiltiin kanavien asetusten säätämistä, lyhyttä testiajoa, sekä raportin generointia, jotka kaikki toimivat odotetusti. Ensimmäinen ohjelmavirhe löytyi reaaliaikaisesta kuvaajasta, joka ei päivittynyt jos kaikki kanavat eivät olleet käytössä.



Kuvio 29. Koeajojärjestelyt. PicoLog 6 vasemmalla, Delta-Master oikealla.

Lisäksi tehtiin yksi yli tunnin mittainen lämpenemätestin koeajo, jossa havaittiin kaksi ohjelmointivirhettä lisää, joista yksi voidaan laskea kriittiseksi. Kriittinen virhe tapahtui lämpenemätestin päätteeksi, kun testiraporttia yritettiin tulostaa. Ohjelmointi- ja käyttäjävirheen seurauksena ohjelma yritti tulostaa saman nimistä tiedostoa, joka oli jäänyt edellisen koeajon jäljiltä auki. Ohjelma oli kuitenkin suunnitellusti onnistunut tallentamaan koko mittaushistorian ensin uniikkiin csv-tiedostoon, joten mittaustuloksia ei epäonnistuneesta raporttigeneroinnista huolimatta kadotettu.

```
# Virheellinen rivi funktiossa make_report
document_name = "veo-test" + ".docx"
doc.save(Path(__file__).parent / "Measurements" / document_name)

# Korjattu rivi
start_time = f"{docinfo.test_start_time:%Y-%m-%d_%H-%M-%S}"
document_name = f"{start_time}.docx"

try:
    doc.save(Path(__file__).parent / "Measurements" / document_name)
except:
    ...
```

Kuvio 30. Raportintallennuksen ohjelmointivirhe ja sen korjaus.

Virhe löytyi veodoc.py:n make_report-funktiosta johon oli jäänyt väliaikainen ratkaisu tiedoston nimeämiseen. Virhe korjattiin muuttamalla nimeämisperiaate funktioon alun perin suunnitellulla tavalla, eli testin ensimmäiseen aikaleimaan perustuvana nimenä, joka on aina uniikki. Lisäksi tallennusprosessi sijoitettiin try-except-virheenkäsittelylogiikan sisälle, jotta virhe tallennuksessa ei kaataisi ohjelmaa.

Toinen ohjelmointivirhe liittyy termisen tasapainotilan ilmaisun värikoodaukseen, jossa virheellisen logiikan seurauksena ehtolause oli aina tosi. Virhe löytyi main.py:n Measurement-luokan metodista delta_t_one_hour ja se korjattiin kuviossa 31 esitetyllä tavalla.

```
# Virheellinen rivi metodissa delta_t_one_hour_
if temperature_delta < 1 or temperature_delta > -1:
    container.trt.bgcolor = veo_green

# Korjattu rivi
if 1 > temperature_delta >= 0:
    container.trt.bgcolor = veo_green
```

Kuvio 31. Värikoodauksen ohjelmointivirhe ja sen korjaus.

5.8 Sovellukseen tehtävät muutokset

Sovelluskehityksen aikana eri kokouksissa ja keskusteluissa on noussut lukuisia hyviä kehitysideoita sovelluksen ominaisuuksiin ja toimintaperiaatteisiin. Ne rajattiin muutamaa poikkeusta lukuun ottamatta tämän opinnäytetyön ulkopuolelle, mutta ne listataan tässä alla ohjaamaan ja jäsentämään sovelluksen jatkokehitystä.

- **Censa-virtamittari**
  Virtamittarin liittäminen suoraan sovellukseen. Näin raporttiin saataisiin automaattisesti lämpenemätestin aikana syötetty virta. Censan virtamittarissa on ethernet-pistoke ja manuaalin mukaan se tottelee http-protokollaa.
- **Käyttäjäystävällisyys ja tilamuisti**
  Käyttäjäystävällisyyttä voisi parantaa, jos ohjelma muistaisi tiettyjä asetuksia edeltävästä lämpenemätestistä. Lisäksi tiettyihin kriittisiin toimintoihin, kuten testin pysäyttämiseen tulisi lisätä varmistusdialogi.
- **CSV-raporttiautomaatti**
  Nykyisellään valmiin raportin voi tulostaa vain juuri suoritetusta lämpenemätestistä. Lisätään mahdollisuus tulostaa raportti myös aikaisemman csv-tiedoston pohjalta. Koska csv-tiedosto ei sisällä kaikkea raportinluontiin tarvittavaa tietoa, täytyy myös DocInfo-olio tulostaa esimerkiksi json-tiedostoon.
- **Refaktorointi**
  Jatkokehitystä helpottamaan olisi lähdekoodi hyvä jäsennellä uudestaan ja poistaa luokista ja funktioista päällekkäisiä toimintoja.
- **Kanavat numerojärjestykseen 1-30+**
  Tällä hetkellä kanavanumerointi heijastelee laitteiston logiikkaa, eli niitä käsitellään aina kahdeksan kanavan ryhminä. Muutetaan kanavanumerointi jatkuvaksi.

- **Hetkellinen lämpenemä**

  Lisätään Measurement-välilehdelle informaatiotekstikenttä, joka näyttää myös hetkellisen lämpötilan. $\Delta T_{hetkellinen} = T_{kanava} - T_{ambient(KA)}$
- **Asetusten tuominen testimäärittelystä**

  Tehdään Excel-pohja testimäärittelylle, jonka perusteella sovellus voi tehdä kanava-asetukset ja nimeämiset automaattisesti.

5.9 Sovelluksen saattaminen käyttövalmiiksi

Sovelluksen käyttö edellyttää, että tietokoneeseen on asennettu TC-08-ajurit, jotka voi ladata PicoTechin websivuilta. Sovelluksen voi saattaa käyttövalmiiksi kahdella tavalla: ajamalla suoraan lähdekoodista, tai paketoimalla se .exe-muotoon.

5.9.1 Ohjelman käyttö suoraan lähdekoodista

Lähdekoodista ajaminen edellyttää, että tietokoneelle on asennettu Python 3.12 ja sovelluksen juurihakemistosta on löydyttävä seuraavat elementit:

- assets-kansio
- picosdk-kansio
- main.py
- veotrt.py
- veodoc.py
- requirements.txt.

Juurihakemistoon luodaan venv-kehitysympäristö, johon aktivoinnin jälkeen asennetaan sovelluksen määrittelemät kirjastot komennolla *pip install -r requirements.txt*. Sovellus voidaan nyt käynnistää kutsulla *python main.py.*

5.9.2 Ohjelman paketointi .exe-muotoon

Sovelluksen käyttäjän kannalta on helpointa, jos sovellus on pakattu .exe-muotoon, kuten suurin osa muistakin Windows-sovelluksista. Paketointi on erittäin hidas prosessi ja tulisi tehdä vasta, kun ohjelmaan tehdyt muutokset on todettu toimivaksi lähdekoodista ajettuna.

Tämä vaihe edellyttää, että kaikki edellisen kohdan ehdot on täytetty ja venv on aktiivisena. Venv-kehitysympäristöön tulee tämän lisäksi asentaa vielä pyinstaller-kirjasto komennolla *pip install pyinstaller.*

Paketointi tapahtuu komennolla *flet pack main.py --name DeltaMaster*.[28] Valmis exe-tiedosto löytyy nyt juurihakemiston dist-kansiosta.

6 SOVELLUKSEN KÄYTTÖ

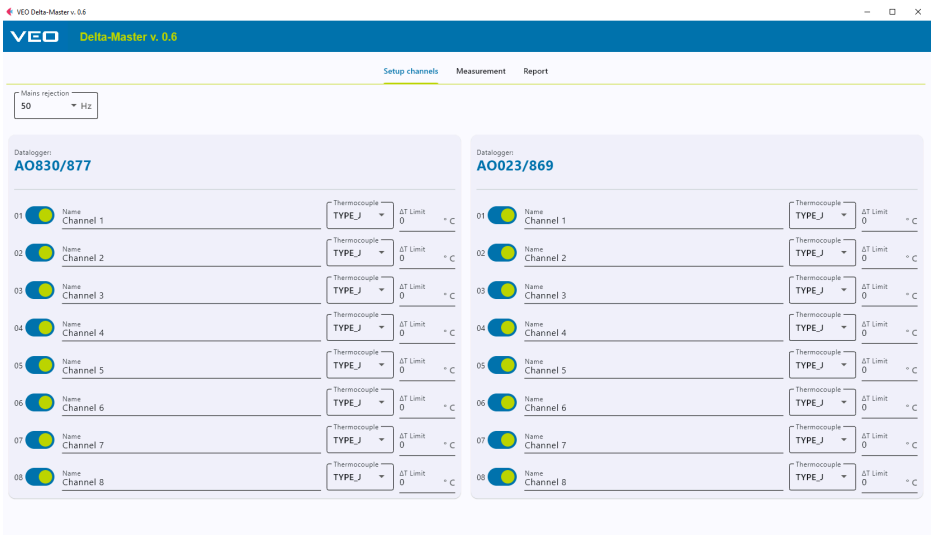
Käyttäjälle mittausohjelmisto näyttäytyy mahdollisimman yksinkertaisena ja helppokäyttöisenä graafisena käyttöliittymänä, joka on jaettu kolmeen osaan lämpenemätestauksen kannalta loogisessa järjestyksessä. Jos aloitusnäkymässä ei näy dataloggereita, tarkista että vähintään yksi TC-08 on kytketty tietokoneen USB-väylään ja että TC-08-ajurit on asennettu.

6.1 Kanavien asetukset

Ohjelmiston käynnistyttyä, käyttäjälle esitetään kaikki kytketyt TC-08-dataloggerit. Koska dataloggerit ovat kaikki identtisiä, niiden tunnistamiseksi sarjanumero on erityisen tärkeä tieto. Kaikkia kanavia koskevat asetukset löytyvät yläpalkista, kuten verkkovirran suodatus (oletuksena 50 Hz).

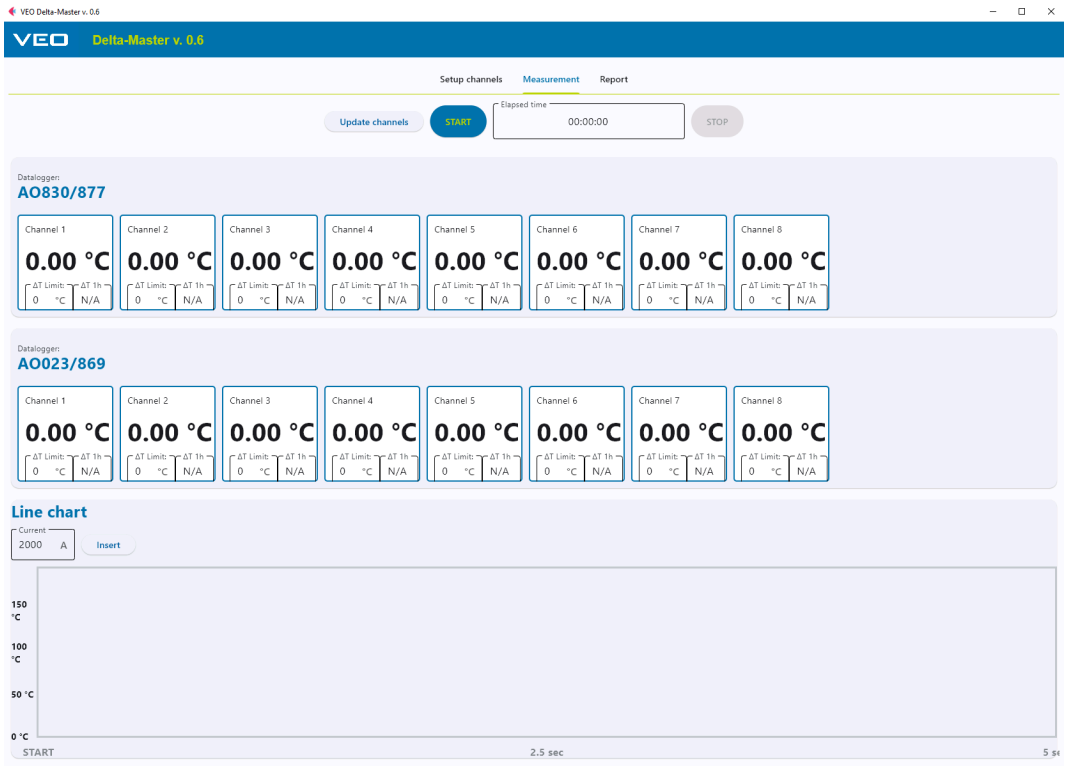
Ensimmäisellä välilehdellä tehdään mittauksen kanavakohtaiset asetukset:

- Kanavan käyttöönotto (On/Off, oletuksena On)
- Kanavan nimi (**Huom!** Käytä ympäristöantureissa sanaa **ambient**)
- Termopari (oletuksena Type J)
- ΔT raja (oletuksena 0; ei käytössä).



Kuvio 32. VEO Delta-Masterin käyttöliittymä, kanavien asetukset.

6.2 Mittauksen käynnistys ja seuranta



Kuvio 33. VEO Delta-Masterin käyttöliittymä, mittaus

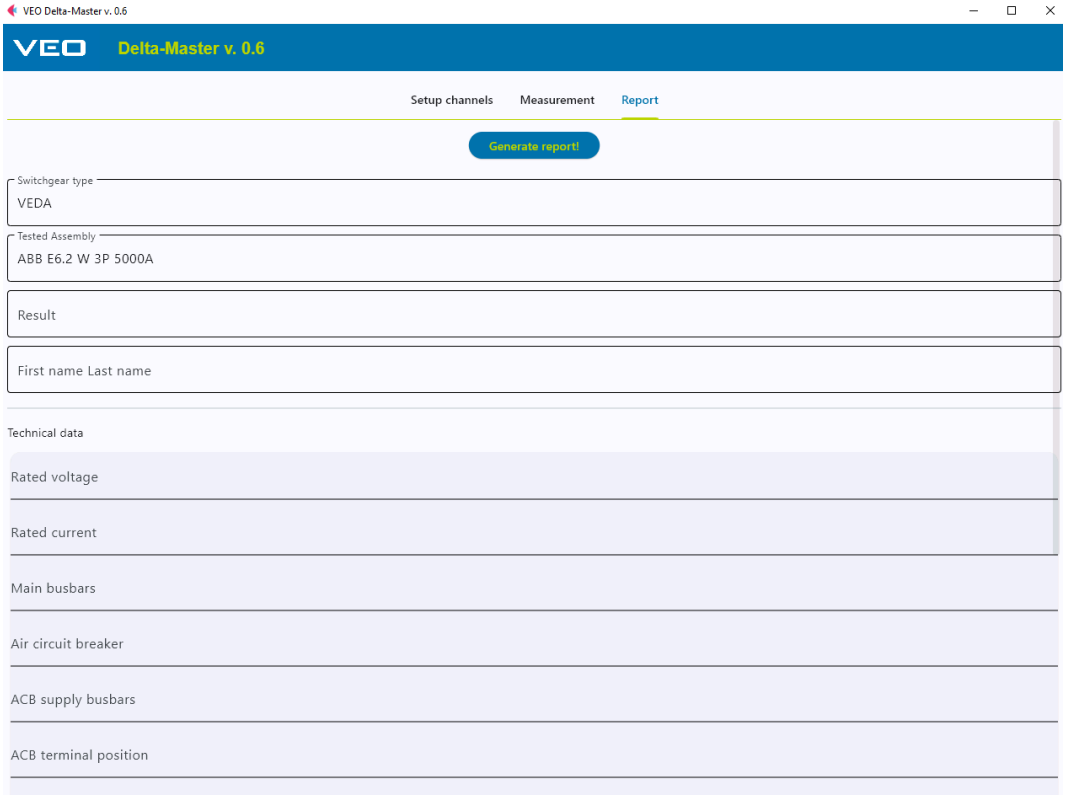
Tältä välilehdeltä voidaan käynnistää mittausprosessi ja seurata sen edistymistä. Yläosa koostuu kanavakohtaisista laatikoista:

- Datakeräimen sarjanumero
- Kanavakohtainen nimi
- Tämänhetkinen lämpötila
- Lämpenemän raja (ΔT Limit)
- Terminen tasapaino (ΔT 1h).

Alaosassa mittaustulokset esitetään jatkuvasti päivittyvänä kuvaajana. Viemällä hiiren kuvaajan päälle, saadaan sen hetkiset mittaustulokset kanavakohtaisesti.

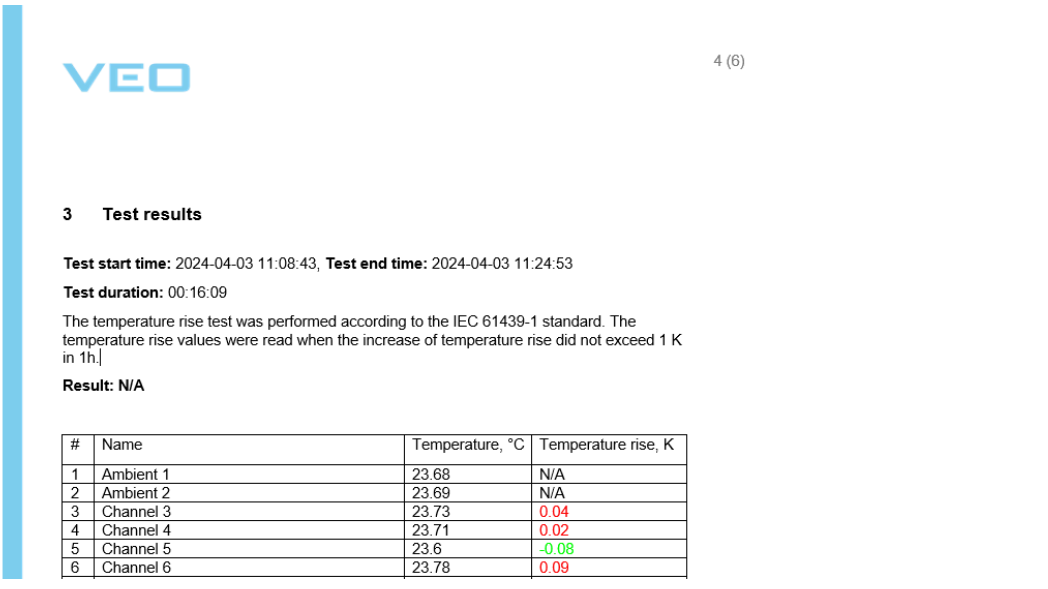
Käytetty mitoitusvirta voidaan myös syöttää kuvaajaan manuaalisesti.

6.3 Raportointi



Kuvio 34. VEO Delta-Masterin käyttöliittymä, raportointi

Kun lämpenemätestin ehdot on todettu täyttyneeksi ja testausprosessi on pysäytetty, voidaan tältä välilehdeltä tulostaa raportti mittaustuloksista.



VEO

4 (6)

3 Test results

Test start time: 2024-04-03 11:08:43, **Test end time:** 2024-04-03 11:24:53

Test duration: 00:16:09

The temperature rise test was performed according to the IEC 61439-1 standard. The temperature rise values were read when the increase of temperature rise did not exceed 1 K in 1h.

Result: N/A

#	Name	Temperature, °C	Temperature rise, K
1	Ambient 1	23.68	N/A
2	Ambient 2	23.69	N/A
3	Channel 3	23.73	0.04
4	Channel 4	23.71	0.02
5	Channel 5	23.6	-0.08
6	Channel 6	23.78	0.09

7 YHTEENVETO JA REFLEKTOINTI

Opinnäytetyön lähtöpiste, eli ongelma, johon lähdettiin etsimään ja toteuttamaan ratkaisua oli ensisilmäyksellä hyvin yksinkertainen; kuinka ilmaista testiehtojen täyttyminen ja tulostaa automaattinen raportti lämpenemätestauksesta?

Eri lähestymistapoja vertailemalla löydettiinkin muutama mahdollinen ratkaisutekniikka, jotka olisivat vastanneet lähtöpisteen ongelmiin ja rakentuneet jo olemassa olevien sovellusten päälle. Lopulta päädyttiin kuitenkin aloittamaan täysin puhtaalta pöydältä ja rakentamaan oma pienjännitekeskusten lämpenemätestauksen mittaussovellus, sillä olemassa olevien sovelluksien liitoskohdat katsottiin olevan rajoittavia ja teknisesti huteralla pohjalla.

Sovelluksen rakentaminen alusta alkaen on ollut erittäin palkitseva oppimiskokemus ja jokainen askel sillä polulla on ollut jatkuvasti oman sen hetkisen ymmärryksen ja osaamisen reunalla. Ennen tätä opinnäytetyötä kokemukseni ohjelmoinnista on ollut suurimmaksi osaksi funktioista koostuvien skriptien tekemistä, jolloin ohjelman sisäisen ja ulkoisen tiedonsiirron on voinut toteuttaa yksinkertaisesti eri muuttujatyypeillä ja tiedostojen luku- ja kirjoituskäskyillä. Tämän opinnäytetyön onnistumisen kannalta kriittinen hetki olikin, kun ymmärsin että minun on otettava haltuun oliopohjaisen ohjelmoinnin periaatteet. Ilman oliopohjaisen ohjelmoinnin käytäntöjä olisi koko projekti todennäköisesti kaatunut omaan monimutkaisuuteensa ja se olisi tehnyt graafisen käyttöliittymän toteuttamisesta erittäin hankalaa, jollei täysin mahdotonta.

Vaikka tämän projektin laajuus oli selvillä ratkaisuvaihtoehtojen tutkimisen jälkeen, yllätti huomioonotettavien yksityiskohtien ja toimintojen riippuvuuksien määrä silti, varsinkin mitä enemmän ominaisuuksia ja toimintoja ohjelmistoon saatiin valmiiksi.

Projektinhallinnallisesti suurien kokonaisuuksien toteuttamisjärjestys oli selvää alusta asti, mutta tämän projektin kannalta lineaarinen toteuttamistapa yrityksen

ja erehdyksen kautta ei ollut optimaalinen. Parempi tapa olisi ollut hajauttaa ohjelmistonkehitys alussa muutamiin rinnakkaishaaroihin, erityisesti niiltä osin, jotka olivat ennestään tuntemattomia konsepteja, kuten esimerkiksi oliopohjainen ohjelmointi ja käyttöliittymän ohjelmointi. Oliopohjaisen ohjelmoinnin hallitseminen jo suunnitteluvaiheessa olisi virtaviivaistanut ohjelmiston kehitystä huomattavasti ja tehnyt kokonaisuudesta helpommin hallittavan.

Haastavin osuus tämän opinnäytetyön tekemisessä oli sen jakautuminen kahteen osaan; tähän kirjoitettuun raporttiin, sekä itse sovelluksen ohjelmointiin, johon tässä vaiheessa sisältyy jo yli 1 500 riviä koodia. Jälkiviisaana työn olisi voinut rajata esimerkiksi vain sovelluksen yksityiskohtaiseen suunnitteluun ja jättää toteutuksen myöhemmälle. Muuten opinnäytetyön laajuuden rajauksessa onnistuttiin hyvin ja siitä pidettiin myös kiinni, vaikka keskustelujen ja kokousten tuloksena syntyikin laaja kirjo uusia toivottuja ominaisuuksia. Näistä tärkeimmät ja niiden toteutustavat on kirjattu ohjelmiston muutokset-otsikon alle.

Näen ohjelmistokehityksen tuntemuksen erinomaisena työkaluna myös koneinsinöörin työkalupakissa. Siitä ei ole ainoastaan hyötyä vain omien ja yrityksen sisäisten työkalujen rakentamisessa ja ylläpidossa, vaan se on myös erittäin tärkeää, kun yritykselle tilataan sovelluksia tai sovellusten räätälöintejä kolmansilta osapuolilta. Tietokoneohjelmien ominaisuuksien ja rajapintojen tarkka määrittely ja ymmärtäminen on kriittinen osa onnistuneita ohjelmien kehitysprojekteja.

Osana tätä opinnäytetyötä VEOlle on toimitettu projektille asetetut kriteerit täyttävä ensimmäinen versio lämpenemätestauksen sovelluksesta. Haluan ojentaa suuret kiitokset tämän opinnäytetyön ideasta ja sen tiimoilta käydyistä keskusteluista VEOn Martti Mattilalle, Jussi Anttilalle, Janne Saarenpäälle ja Jan Rönnholmille, sekä VAMKin Osku Hirvoselle ohjauksesta ja monista kiinnostavista luennoista.

LÄHTEET

1. SFS-EN IEC 61439-1. Pienjännitekeskukset. Osa 1: Yleisvaatimukset. Suomen standardisoimisliitto SFS ry.; 2022.

2. Rönnholm J, Saarenpää J, Anttila J. [Kokous]. 2023.

3. PicoLog 6 data logging software - The Data Logging A to Z [Internet]. [viitattu 1. maaliskuuta 2024]. Saatavissa: https://www.picotech.com/library/data-loggers/picolog-6-data-logger-software

4. PicoLog Cloud 6.2.0 Beta now open! - Pico Technology [Internet]. [viitattu 1. maaliskuuta 2024]. Saatavissa: https://www.picotech.com/support/topic41153.html

5. TC-08 Thermocouple data logger | Pico Technology [Internet]. [viitattu 1. maaliskuuta 2024]. Saatavissa: https://www.picotech.com/data-logger/tc-08/thermocouple-data-logger

6. Gill L. Difference Between J vs K Type Thermocouples: Which to Buy? [Internet]. Process Parameters Ltd. 2023 [viitattu 4. huhtikuuta 2024]. Saatavissa: https://www.processparameters.co.uk/j-vs-k-type-thermocouples/

7. Fisher-Cripps AC. Newnes Interfacing Companion - Computers, Transducers, Instrumentation and Signal Processing. Newnes; 2002.

8. Comparing Python to Other Languages [Internet]. Python.org. [viitattu 3. huhtikuuta 2024]. Saatavissa: https://www.python.org/doc/essays/comparisons/

9. Lutz M. Learning Python. 5th Edition. O'Reilly Media, Inc.; 2013.

10. PEP 8 – Style Guide for Python Code | peps.python.org [Internet]. Python Enhancement Proposals (PEPs). [viitattu 5. huhtikuuta 2024]. Saatavissa: https://peps.python.org/pep-0008/

11. Marsh K, Downes A. Python, The Complete Manual. Future PLC; 2022.

12. Ohjelmoinnin perusteet ja Ohjelmoinnin jatkokurssi, syksy 2018 | Osa 7 [Internet]. [viitattu 3. huhtikuuta 2024]. Saatavissa: https://ohjelmointis18.mooc.fi/part7/

13. Luku 2.1: Olio-ohjelmointi | O1 | A+ [Internet]. [viitattu 3. huhtikuuta 2024]. Saatavissa: https://plus.cs.aalto.fi/o1/2022/w02/ch01/

14. Dannaway A. Practical UI. 2024.

15. Introducing PicoLog Cloud - a free upgrade for PicoLog users [Internet]. [viitattu 3. maaliskuuta 2024]. Saatavissa: https://www.picotech.com/library/data-loggers/introducing-picolog-cloud

16. PicoTech. PicoLog Cloud Public API. PicoTech;

17. Bhargava K. What is Epoch time ? Also know as UNIX time or POSIX time [Internet]. Medium. 2023 [viitattu 3. maaliskuuta 2024]. Saatavissa: https://medium.com/@kushal.bhargava01/what-is-epoch-time-also-know-as-unix-time-or-posix-time-bd8efd1a491

18. Marttila M. [Keskustelu]. 2024.

19. PicoLog Cloud - Login [Internet]. [viitattu 4. maaliskuuta 2024]. Saatavissa: https://picolog.app/#/login

20. PicoLog Users Guide. PicoTech; 2014.

21. alvinashcraft. About Dynamic Data Exchange - Win32 apps [Internet]. 2020 [viitattu 9. helmikuuta 2024]. Saatavissa: https://learn.microsoft.com/en-us/windows/win32/dataxchg/about-dynamic-data-exchange

22. Pico Technology [Internet]. [viitattu 4. maaliskuuta 2024]. Saatavissa: https://github.com/picotech

23. ctypes — A foreign function library for Python [Internet]. Python documentation. [viitattu 25. maaliskuuta 2024]. Saatavissa: https://docs.python.org/3/library/ctypes.html

24. enum — Support for enumerations [Internet]. Python documentation. [viitattu 25. maaliskuuta 2024]. Saatavissa: https://docs.python.org/3/library/enum.html

25. picotech/picosdk-python-wrappers [Internet]. Pico Technology; 2024 [viitattu 25. maaliskuuta 2024]. Saatavissa: https://github.com/picotech/picosdk-python-wrappers

26. Python Data Types [Internet]. GeeksforGeeks. 2019 [viitattu 25. maaliskuuta 2024]. Saatavissa: https://www.geeksforgeeks.org/python-data-types/

27. PicoTech. USB TC-08 Programmer's Guide. PicoTech; 2017.

28. Packaging desktop app | Flet [Internet]. [viitattu 5. huhtikuuta 2024]. Saatavissa: https://flet.dev/docs/guides/python/packaging-desktop-app