SAVONIA


OPINNÄYTETYÖ - AMMATTIKORKEAKOULUTUTKINTO
TEKNIIKAN ALA

# LÄMMITYSKATTILAN INTEGROINTI KOTIAUTOMAATIOJÄRJESTELMÄÄN

Opinnäytetyö

TEKIJÄ Oskari Toivanen

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tiivistelmä

| | |
|---|---|
| Koulutusala<br>Tekniikan ja liikenteen ala | |
| Tutkinto-ohjelma<br>Tietotekniikan tutkinto-ohjelma | |
| Työn tekijä<br>Oskari Toivanen | |
| Työn nimi<br>Lämmityskattilan integrointi kotiautomaatiojärjestelmään | |
| Päiväys 23.3.2025 | Sivumäärä/Liitteet 69/0 |
| Yhteistyötaho<br>Savonia-ammattikorkeakoulu | |


Opinnäytetyössä tutkittiin vesikiertoisen keskuslämmityksen sähkökattilan integrointia kotiautomaatiojärjestelmään. Työn tavoitteena oli kehittää IoT-pohjainen laite, joka mahdollistaa lämmitysjärjestelmän hallinnan ja energiatehokkuuden parantamisen automaation avulla. Erityisenä haasteena oli integroida iäkäs järjestelmä, joka ei tarjonnut valmiita liitäntämahdollisuuksia ulkoisiin järjestelmiin. Ratkaisun avulla järjestelmän ohjaus ja seuranta voitiin toteuttaa etänä sekä hyödyntää pörssisähkön hintavaihteluita ja tukilämmitysmuotojen tuottamaa dataa energiankulutuksen optimointiin.

Työ toteutettiin suunnittelemalla ja rakentamalla mikrokontrolleripohjainen laite, joka kykenee kommunikoimaan ulkoisten järjestelmien kanssa MQTT- ja HTTP-protokollien avulla. Vaikka laite on suunniteltu yleiskäyttöiseksi, opinnäytetyön kohteessa se konfiguroitiin toimimaan Home Assistant -kotiautomaatioalustan kanssa. Kehitysprosessi sisälsi laitteiston valinnan, koteloinnin, ohjelmistokehityksen ja testauksen. Ohjelmointi toteutettiin C++-kielellä, ja laitteelle kehitettiin selainpohjainen käyttöliittymä, joka mahdollistaa sen konfiguroinnin ja hallinnan suoraan eri päätelaitteilla ilman erillistä kotiautomaatiojärjestelmää. Laitteen toimintaa testattiin muun muassa verkkoyhteyden vakauden, sensoriarvojen tarkkuuden ja ohjaustoimintojen luotettavuuden osalta.

Tulokset osoittivat, että laite parantaa lämmitysjärjestelmän hallittavuutta ja mahdollistaa energiankulutuksen optimoinnin. Se toimi luotettavasti ja täytti sille asetetut vaatimukset. Jatkokehitysmahdollisuuksia ovat muun muassa tekoälypohjaisen analytiikan hyödyntäminen, tietoturvan kehittäminen sekä laajennettavuus muihin mitattaviin suureisiin ja ohjattaviin toimilaitteisiin.



Avainsanat
ESP8266, Mikrokontrolleri, Kotiautomaatio, Home Assistant

SISÄLTÖ

1 JOHDANTO ..... 7

2 PIENTALOJEN SÄHKÖLÄMMITYS ..... 8

2.1 Sähkölämmityksen perusteet ..... 8

2.2 Vesikiertoinen lattialämmitys sähkökattilalla ..... 8

2.3 Energiatehokkuus ja kustannusnäkökulmat ..... 8

3 HOME ASSISTANT -KOTIAUTOMAATIOALUSTA ..... 10

3.1 Home Assistantin yleisesittely ..... 10

3.2 Integraatiot ..... 11

3.3 Automaatiot ..... 11

4 MIKROKONTROLLERIT JA NIIDEN KÄYTTÖ IOT-RATKAISUISSA ..... 13

4.1 Mikrokontrollerien perusteet ..... 13

4.2 ESP8266 ja sen soveltuvuus kotiautomaatioprojekteihin ..... 13

4.3 Mikrokontrollerien viestintäprotokollat ..... 14

4.4 MQTT-protokolla ..... 16

5 TYÖN SUUNNITTELU ..... 18

5.1 Vaatimusmäärittely ..... 18

5.2 Projektinhallinta ..... 20

5.3 Mikrokontrollerin ja kehitysalustan valinta ..... 20

5.4 Komponenttien valinta ..... 22

5.5 Laitteistoarkkitehtuuri ..... 24

6 SÄHKÖISET KYTKENNÄT ..... 25

7 KOTELO JA LAITTEISTON SUOJAUS ..... 27

8 OHJELMOINTI ..... 30

8.1 Ohjelmointikielen valinta ..... 30

8.2 Ohjelmointiympäristö ..... 30

8.3 Ohjelman toiminnallinen kuvaus ..... 32

8.4 Ohjelman arkkitehtuuri ..... 38

8.4.1 Modulaarisuus ja kapselointi ..... 38

8.4.2 Luokkien välinen kommunikointi ..... 39

8.5 Ohjelman suorituskaavio ..... 40

8.6 Selainkäyttöliittymä ..... 42

8.6.1 Yleistä ..... 42

8.6.2 Käyttöliittymän rakenne ..... 43
8.6.3 Status-sivu ..... 45
8.6.4 Sensor Config -sivu ..... 47
8.6.5 WiFi Config ja MQTT Config -sivut ..... 49
8.6.6 Tyylitiedosto ja responsiivisuus ..... 52
8.7 Muut luokat ja toiminnot ..... 53
8.7.1 Konfiguraation ja tiedostojen hallinta ..... 53
8.7.2 Sensoreiden hallinta ..... 54
8.7.3 Fyysiset indikaattorit ..... 55
8.7.4 Relettä ohjaava luokka ..... 57
8.8 MQTT-viestintä ..... 58
8.9 Home Assistant konfigurointi ..... 61
9 TESTAUS JA TULOKSET ..... 63
10 YHTEENVETO JA JATKOKEHITYS ..... 64
11 POHDINTA ..... 66
12 LÄHTEET ..... 67

## KUVALUETTELO

Kuva 1. Esimerkki Home Assistantin koontinäytöstä ..... 10
Kuva 2. Home Assistant integraatiota (Home Assistant 2025) ..... 11
Kuva 3. Esimerkki Telegram -integraatioon perustuvan automaation konfiguroinnista käyttöliittymän kautta ..... 12
Kuva 4. Ote saman automaation YAML-määrityksestä ..... 12
Kuva 5. Lolin Wemos D1 Mini (WEMOS n.d.) ..... 13
Kuva 6. IoT-keskeinen OSI-malli (Cisco Press 2019) ..... 14
Kuva 7. MQTT julkaisu/tilaus arkkitehtuuri (HiveMQ n.d.) ..... 16
Kuva 8. Dallas DS18B20 lämpötilasensori (JemRF 2025) ..... 22
Kuva 9. 3.3V relemoduuli (Amazon 2025) ..... 22
Kuva 10. I²C-liitäntäinen OLED-näyttö (Amazon 2025) ..... 23
Kuva 11. WS2812B RGB-LED (Triopak Oy 2025) ..... 23
Kuva 12. Laitteistoarkkitehtuuri ja fyysinen kuvaus ..... 24
Kuva 13. Mikrokontrollerin kytkennät ..... 25
Kuva 14. Jäspi TehoWatti relekortin piirikaavio (Jäspi 2016) ..... 25
Kuva 15. Kattilan piirikortin fyysiset liittimet ..... 26
Kuva 16. TehoWatin ohjaussignaalin kytkentä IoT-laitteeseen ..... 26

Kuva 17. 3D-mallinnettu kokoonpano ..... 27
Kuva 18. Kotelon runko ilman johdotusta ..... 28
Kuva 19. Kotelon runko ..... 28
Kuva 20. Komponentit johdotettuna ..... 29
Kuva 21. Valmis kokoonpano ..... 29
Kuva 22. Arduino IDE ..... 30
Kuva 23. Visual Studio Code PlatformIO -lisäosa ..... 31
Kuva 24. PlatformIO Inspect-toiminto ..... 31
Kuva 25. PlatformIO-projektin rakenne ..... 32
Kuva 26. Laitteen muodostama WiFi-tukiasema ..... 33
Kuva 27. Laitteen tilaindikaattorit ennen konfigurointia ..... 33
Kuva 28. Laitteen web-käyttöliittymä ..... 33
Kuva 29. MQTT-konfiguraatiolomake ..... 34
Kuva 30. Sensoriosoitteiden skannaus ja asettelu ..... 34
Kuva 31. Fyysiset indikaattorit valmiustilassa ..... 35
Kuva 32. Käyttöliittymän status-sivu jatkuvan toiminnan tilassa ..... 35
Kuva 33. Tulo- ja menoveden lämpötilojen historiatiedot ..... 36
Kuva 34. Laitteen tilatiedot ja ohjauspainike automaatiojärjestelmässä ..... 36
Kuva 35. MQTT-yhteys katkennut ..... 37
Kuva 36. WiFiManager-luokka laajentaa ESP8266WiFiClass luokkaa ..... 38
Kuva 37. Luokkien välinen kommunikaatio, periaatekuva ..... 39
Kuva 38. Globaali getStatesAsJson() -funktio ..... 40
Kuva 39. Ohjelman yksinkertaistettu suorituskaavio ..... 41
Kuva 40. MQTT-yhteyden muodostava funktio ..... 42
Kuva 41. Käyttöliittymän rakenne ..... 43
Kuva 42. HTTP-pyynnöt käyttöliittymää ladattaessa ..... 44
Kuva 43. Staattisia tiedostoja palauttavat reitit ..... 44
Kuva 44. WebServerManager-luokan _streamFile() -metodi ..... 45
Kuva 45. /states -reitin palauttama JSON-objekti ..... 45
Kuva 46. Selaimessa tilatiedot täydentävä JavaScript-funktio ..... 46
Kuva 47. IoT-laitteen tilatiedot palauttava globaali funktio pääohjelmassa ..... 46
Kuva 48. Status-sivun ohjauspainikkeet ..... 47
Kuva 49. ”Find available sensors” painettu ..... 47
Kuva 50. sensorconfig.html -fragmentti ..... 48
Kuva 51. Palvelimen "/sensors" -reitti joka palauttaa osoitteet ja lämpötilat ..... 48
Kuva 52. Sensoritiedot konfigurointisivulle täydentävä JavaScript-funktio ..... 49

Kuva 53. Konfiguraatiolomake ..... 50
Kuva 54. Lomaketietojen tallentaminen JSON-objektista laitteessa ..... 51
Kuva 55. ConfigManager -luokan settereitä ..... 51
Kuva 56. Mediakyselyyn pohjautuva responsiivisuus ..... 52
Kuva 57. Mobiilinäkymä ..... 52
Kuva 58. ConfigManager-luokan otsikkotiedosto ..... 53
Kuva 59. Oliot konfiguroidaan pääohjelman setup-lohkossa ConfigManagerin ..... 54
Kuva 60. SensorManager-luokan otsikkotiedosto ..... 55
Kuva 61. readSensors()-metodi lämpötilojen lukemiseksi ..... 55
Kuva 62. LEDManager-luokan otsikkotiedosto ..... 56
Kuva 63. LED-yksikköä ohjaava logiikka ..... 56
Kuva 64. OLEDManager-luokan otsikkotiedosto ..... 56
Kuva 65. Näytön päivitys rivikohtaisesti ..... 57
Kuva 66. Relay-luokan otsikkotiedosto ..... 57
Kuva 67. Luokka kapseloi I/O-komennot ..... 58
Kuva 68. MQTT-viestinnän työnkulku ..... 59
Kuva 69. Viestintä relettä ohjattaessa ..... 59
Kuva 70. MQTT-asiakkaan määritys setup-lohkossa ..... 60
Kuva 71. MQTT-asiakkaan takaisinkutsufunktio ..... 60
Kuva 72. Pääohjelman globaali setRelayState()-funktio ..... 61
Kuva 73. configuration.yaml sisältö ..... 61
Kuva 74. Home Assistantin laitenäkymä koostaa entiteetit yhteen näkymään ..... 62
Kuva 75. Lämpötilan pudotusreleen tilatiedon historia ..... 62
Kuva 76. Valmis laite ennen käyttöpaikkaan asentamista ..... 64
Kuva 77. MQTT-kommunikaatio Windows- ja Android-sovelluksilla ..... 64

# 1 JOHDANTO

Tämän opinnäytetyön tavoitteena oli integroida omakotitalon vesikiertoisen lattialämmitysjärjestelmän lämmönlähteenä toimiva sähkökattila kotiautomaatiojärjestelmään mahdollistaen kiertoveden lämpötilojen seurannan ja karkean säädön automatisoidusti. Opinnäytetyön pääpaino oli ESP8266 pohjaisen mikrokontrolleriin perustuvan laitteen ohjelmoimisessa ja konfiguroimisessa olemassa olevaan Home Assistant järjestelmään. Kotiautomaatiojärjestelmän käyttöönotto rajattiin pois opinnäytetyöstä.

Kohteen sähkökattila oli vuonna 2012 käyttöönotettu Jäspi TehoWatti, ja siitä puuttui ajankohtaan nähden tyypillisesti liitäntämahdollisuudet ulkoisiin järjestelmiin. Kattilan omien sensoreiden oloarvoja ei pysty lukemaan ohjelmallisesti, eikä kattila tarjoa rajapintoja menoveden asetusarvon säätämiseksi ohjelmallisesti. Kattilan ohjauskorttiin on kuitenkin mahdollisuus kytkeä ns. kotona-poissa-kytkin joka aktivoituessaan alentaa menoveden lämpötilaa. Lämpötilan alentamisen suuruutta voidaan säätää kattilan ohjauspaneelista.

Opinnäytetyössä lattialämmitysjärjestelmään asennettiin erilliset sensorit mittaamaan tulo- ja menoveden lämpötiloja sekä rele, jota ohjaamalla voidaan vaihtaa kattilan kotona-poissa-tilaa ja näin säätää menoveden lämpötilaa kahdessa portaassa. Sensorit ja rele kytkettiin fyysisesti ESP8266-pohjaiseen mikrokontrolleriin, joka välittää dataa ja ohjauskomentoja langattomasti kotiautomaatiojärjestelmälle MQTT-protokollaa hyödyntäen.

Integraatio palvelee taloudellisten näkökohtien lisäksi myös turvallisuus- ja ympäristönäkökohtia. Ohjaamalla lämmitystä pörssisähkön edullisimmille tunneille kulutus sijoittuu automaattisesti ajankohtiin, jolloin sähköverkon kuormitus on alimmillaan ja näin kuormitus tasapainottuu. Tämän lisäksi ulkolämpötilan, tulo- ja menoveden sekä lisälämmönlähteiden kuten ilmalämpöpumpun ja tulisijan reaaliaikainen seuranta mahdollistaa lämmönlähteiden yhteiskäytön optimoinnin. Turvallisuusnäkökohtana järjestelmä pystyy tuottamaan etähälytyksiä viestintälaitteisiin, mikäli verkoston lämpötila laskee äkillisesti ja tilanteeseen liittyy esimerkiksi jäätymisvaara.

Järjestelmä tuottaa myös historiatietoa lukuisista eri lämmityskokonaisuuteen vaikuttavista parametreista, jota voidaan tulevaisuudessa hyödyntää esimerkiksi optimoimalla lämmitysjärjestelmien toimintaa koneoppimisen tai tekoälyn avulla.

# 2 PIENTALOJEN SÄHKÖLÄMMITYS

## 2.1 Sähkölämmityksen perusteet

Sähkölämmitys oli vuonna 2020 Suomen yleisin pientalojen lämmitysmuoto myös vanhat kiinteistöt huomioiden (Tilastokeskus 2021). Sähkölämmityksen suosioon on vaikuttanut muita lämmitysmuotoja matalammat käyttöönottokustannukset, nopea käyttöönotto ja järjestelmän helppokäyttöisyys (OptiWatti 2019). Lämmitysenergiamuotona sähkö on käytönoton suhteen nopea ja edullinen, sillä erillisiä lämpökaivoja, öljy- ja pellettisäiliöitä tai muita energialogistiikkaan liittyviä ratkaisuja ei tarvita.

Sähkölämmitys perustuu sähköenergian muuntamiseen lämmöksi, joko suoraan sähkövastuksien avulla esimerkiksi pattereissa ja lattialämmityksessä, tai välillisesti muun muassa lämpöpumppujen avulla. Lämmitysvastukset (patterit, lattialämmitys) voivat sijaita huonekohtaisesti kiinteistön eri osissa, tai lämmitysenergiaa voidaan siirtää ns. keskuslämmitysjärjestelmässä väliainetta käyttäen eri tiloihin.

Sähkölämmityksen negatiivisiin puoliin kuuluu suhteellisen voimakas hinnan vaihtelu ja heikko ennustettavuus, jotka perustuvat sähkön tuotanto- ja kulutusmääriin (Fingrid n.d.)

## 2.2 Vesikiertoinen lattialämmitys sähkökattilalla

Opinnäytetyön kohteessa on käytössä vesikiertoinen lattialämmitys sähkökattilalla. Tällaisessa järjestelmässä lämmitysenergia tuotetaan keskitetysti, yleensä teknisessä tilassa sijaitsevalla lämmityskattilalla. Kattilan tuottama sähköenergia siirretään kiinteistön eri osiin lattialaattaan sijoitettujen kiertovesiputkien avulla. Putkissa kiertävä vesi lämmittää kiinteistön lattialaattaa, joka puolestaan lämmittää huonetiloja. Lattialaatta toimii myös osittain varaavana lämmityselementtinä. Vesikiertoinen lattialämmitys on yleisesti jaettu erillisiin nk. lämmityspiireihin – kiinteistön eri tiloissa on toisistaan riippumattomat kiertovesiputket, jolloin yksittäisen tilan lämmityspiiri voidaan sulkea lämmityksen katkaisemiseksi kyseisessä tilassa. (Aurelia n.d.)

Vesikiertoisen lattialämmityksen lämmitystehoa säädetään yleisesti korottamalla tai laskemalla putkistossa kiertävän veden lämpötilaa ulkolämpötilaan perustuen sekä huonekohtaisilla termostaateilla. Huonetermostaatit tyypillisesti avaavat ja sulkevat kyseistä tilaa lämmittävää lämmityspiiriä. (Aurelia n.d.)

Vesikiertoisessa lattialämmityksessä kiertoveden lämpötila on tyypillisesti pienempi kuin huonekohtaisiin lattialämmitysvastuksiin perustuvissa ratkaisuissa. Matalan kiertoveden lämpötilan ja varaavana elementtinä toimivan lattialaatan vuoksi lämmitysjärjestelmä reagoi hitaasti muutoksiin.

Vesikiertoisen lattialämmityksen etuna voidaan pitää muunneltavuutta – lämmitykseen käytettävä energiamuoto voidaan päivittää energiatehokkaammaksi vaihtamalla sähkökattila toisenlaiseen lämmönlähteeseen. (Aurelia n.d.)

## 2.3 Energiatehokkuus ja kustannusnäkökulmat

Suurimpia sähkölämmitteisen pientalon energiatehokkuuteen vaikuttavia tekijöitä ovat muun muassa rakennuksen eristystaso sekä ilmanvaihdon ja lämmitysjärjestelmän oikeanlaiset säädöt. Pientalojen energiankulutuksesta 68 % käytetään rakennuksen lämmittämiseen. (Tilastokeskus 2016.)

Energiatehokkuus tarkoittaa rakennusten lämmityksessä haluttujen olosuhteiden, esimerkiksi huonelämpötilan, saavuttamista mahdollisimman pienellä energiankulutuksella. Huonelämpötilan ylläpitämiseksi vaadittu lämmitysenergia määräytyy muun muassa edellä mainittujen eristystason, ilmanvaihdon sekä lämmitysjärjestelmän säätöjen ja valitun lämmitysmuodon mukaan (EnergiaYkkönen, n.d.). Energiatehokkuuden parantaminen vaatii usein rakennusteknisiä lisäinvestointeja kuten lisälämmöneristämistä, lämmön talteenoton asentamista, ilmavuotojen sulkemista tai lämpöpumpputeknologiaa hyödyntävään lämmitysjärjestelmään investoimista (Motiva 2024). Energiatehokkuutta voidaan parantaa rajallisesti järjestelmän perussäädöillä. Kun järjestelmä toimii muutoin teknisesti oikein, pelkästään huonelämpötilaa laskemalla 1 asteen verran voidaan saavuttaa jopa 5 % säästö käytetyssä lämmitysenergiassa (Motiva 2024). Rakennuksen lämpöhäviöt ovat suoraan verrannollisia sisä- ja ulkolämpötilojen väliseen erotukseen, minkä vuoksi huonelämpötilan alentaminen pienentää lämpöhäviöitä (Ympäristöministeriö 2018).

Kustannustehokkuus muodostuu käytetyn lämmitysenergian kustannusten optimoinnista. Sähkön hinta- ja sopimustyyppikehityksen vuoksi yhä useampi kuluttaja valitsee kiinteähintaisen sähkösopimuksen sijaan pörssisähkösopimuksen. Pohjoismaissa energian sähkön tuotantomääriä säädetään jatkuvasti kysynnän mukaan sähköverkon balansoimiseksi. Energiantuotannon siirtyminen lämpövoimasta säätilojen mukaan vaihteleviin uusiutuvan energian lähteisiin kuten tuuli- ja vesivoimaan hankaloittaa tasapainottamista. Muutokset säätövoiman tuotannossa johtavat tilanteeseen, jossa kysyntä kasvaa sähkön tuotantokapasiteettia suuremmaksi (Energiateollisuus ry 2024). Kuluttajasähkön hinta määräytyy sähköpörssin mukaan. Pörssisähkön hintavaihtelut vaikuttavat pitkällä aikajänteellä myös määräaikaisiin sähkösopimuksiin, mutta erityisen suuri vaikutus hintavaihteluilla on pörssisähkösopimusta käyttävillä kuluttajilla. Pörssisähkösopimuksissa kuluttajalta laskutettavan sähkön hinta määräytyy tuntikohtaisesti vallitsevan pörssisähkön hinnan mukaan (Fingrid n.d.). Pörssisähkön hinnat julkaistaan päivittäin seuraavan vuorokauden ajalle – tämä mahdollistaa kulutuksen säätämisen käyttöpaikalla tuleviin pörssisähköhintoihin perustuen. Tällä tavoin kuluttaja voi säästää energiakustannuksissa ja tasapainottaa sähköverkon kuormitusta.

Pientalon vesikiertoisen lattialämmityksen sähkökattilan integroiminen kotiautomaatiojärjestelmään parantaa sekä energia- että kustannustehokkuutta. Järjestelmä voi automaattisesti alentaa huonetilojen ja lattialaatan lämpötiloja kalliiden pörssisähkötuntien sekä asukkaiden poissaolon ajaksi. Lisäksi sähkökattilaa voidaan ohjata varaamaan lämpöä lattialaattaan erityisen halpojen sähkötuntien aikana.

Tehokkaampi seuranta ja ohjaus mahdollistavat myös tukilämmitysmuotojen, kuten tulisijan ja ilmalämpöpumpun, käytön optimoinnin. Automaatiojärjestelmä hyödyntää pörssisähkön tuntihintoja ja tuntikohtaisia paikallisia ulkolämpötiloja, jolloin lattialämmitystä voidaan ohjata ennakoivasti. Ennustettavan ohjauksen avulla hitaasti reagoivaa lattialämmitystä voidaan säätää ulkolämpötilan muutosten mukaan, mikä parantaa asumismukavuutta ja vähentää turhaa energiankulutusta.

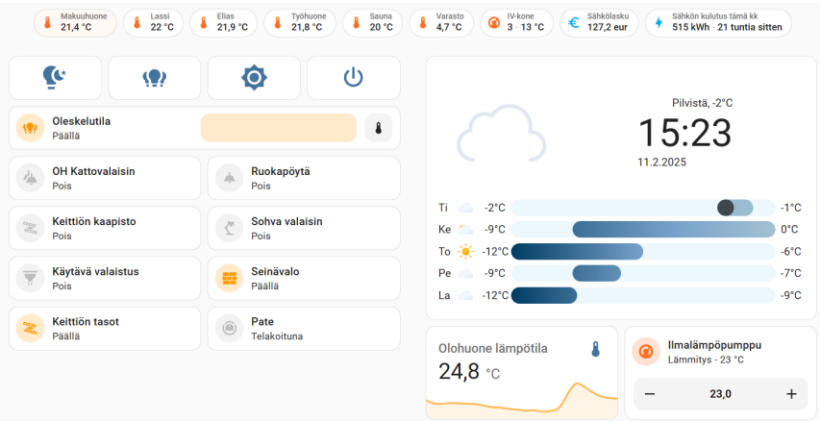
# 3 HOME ASSISTANT -KOTIAUTOMAATIOALUSTA

## 3.1 Home Assistantin yleisesittely

Home Assistant on avoimeen lähdekoodiin perustuva ilmainen, harrastajille ja edistyneille käyttäjille suunnattu kotiautomaatioalusta. Järjestelmän keskiössä on yksityisyys ja riippumattomuus kaupallisista toimijoista yhdessä laajan ilmaisen integraatiokirjaston kanssa. Valmiiden integraatioiden avulla järjestelmään on helppo yhdistää useiden eri toimijoiden laitteistoja ja palveluita ilman toimittajakohtaisia pilvipalveluriippuvuuksia (Home Assistant 2025). Tämän lisäksi järjestelmä mahdollistaa omien integraatioiden luomisen useita eri protokollia ja ohjelmointikieliä hyödyntäen. Home Assistantin avulla käyttäjä voi esimerkiksi yhdistää älyvalaisimet, lämmitysjärjestelmän ja turvakamerat samaan käyttöliittymään ilman erillisiä pilvipalveluita.

Home Assistantista on saatavissa useita versioita eri asennustapoja- ja alustoja varten. Yhteistä näille on se, että tyypillisessä asennuksessa Home Assistant asentuu palvelimeksi paikallisverkkoon. Home Assistant palvelimen tarkoitus on toimia eräänlaisena solmukohtana, joka sisältää tuen protokollille kuten ZigBee, Bluetooth, Z-Wave ja MQTT. Tämän lisäksi palvelimen ydintehtäviä ovat muun muassa sensoritietojen kerääminen erilaisista integraatioista, tietojen tallentaminen ja mahdollinen analytiikka sekä erilaisten automaatioiden suorittaminen. (Home Assistant 2025.)

Taustapalvelintoimintojen lisäksi Home Assistant järjestelmä sisältää kustomoitavan käyttöliittymän, josta järjestelmää voi konfiguroida ja käyttää. Järjestelmän ylläpitäjä voi luoda erilaisia koontinäyttöjä (dashboard), joita voi tarkastella ja käyttää mobiililaitteilla ja tietokoneilla.




Kuva 1. Esimerkki Home Assistantin koontinäytöstä

Vaikka tarvittavat toiminnallisuudet ja logiikka olisi voitu toteuttaa myös suoraan sähkökattilaan kytkettävällä IoT-laitteella, Home Assistant oli luonnollinen valinta. Järjestelmä on ollut käytössä kiinteistössä jo pidemmän aikaa, ja siihen on jo integroitu muun muassa tulisija, ilmalämpöpumppu, ilmanvaihtokone, pörssisähkön tuntihintojen seuranta sekä kattava huonelämpötilan seuranta. Seuraavissa kappaleissa on lyhyesti esitelty opinnäytetyön kannalta tärkeimmät toiminnallisuudet.

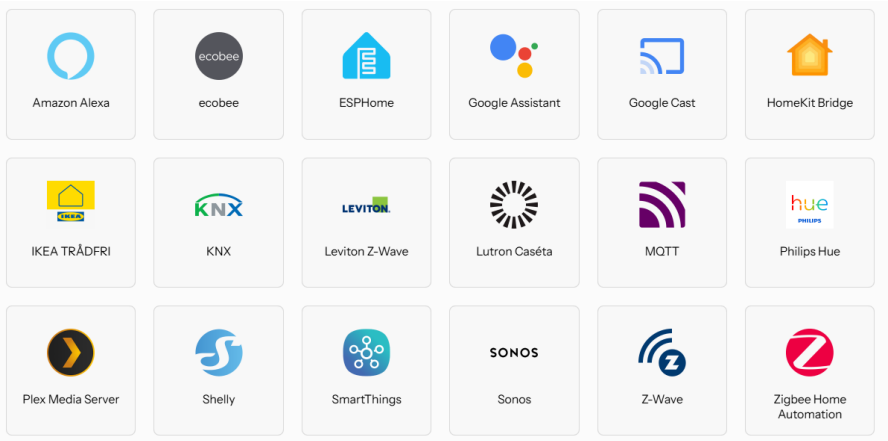
## 3.2 Integraatiot

Home Assistant tukee yli 3000 integraatiota eri valmistajien laitteisiin ja palveluihin (Home Assistant 2025). Integraatiot voivat hyödyntää joko kolmannen osapuolen julkisia rajapintoja, kommunikoida valmistajan omien siltalaitteiden kanssa paikallisesti tai ohittaa nämä täysin keskustelemalla suoraan yksittäisten laitteiden kanssa.

Esimerkiksi Philips Hue ja IKEA Trådfri -järjestelmät edellyttävät normaalisti valmistajan omaa Zigbee-siltalaitetta, jotta niiden älykotituotteet toimivat. Home Assistant voi kuitenkin yhdistää eri ekosysteemejä halliten niiden siltalaitteita samanaikaisesti tai käyttämällä kolmannen osapuolen sovitinta ja riippumatonta Zigbee2MQTT- tai ZHA-integraatiota, mikä mahdollistaa laitteiden suoran liittämisen ilman valmistajan pilvipalveluita.

Keskitetty kotiautomaatiojärjestelmä mahdollistaa eri valmistajien laitteiden yhdistämisen, jolloin käyttäjä voi valita kustannustehokkaimmat ja parhaiten tarpeisiinsa sopivat ratkaisut. Riippumattoman sovittimen avulla järjestelmä toimii paikallisesti ilman, että käyttäjätietoja lähetetään valmistajille, eikä sen toiminta ole riippuvainen verkkoyhteydestä.

Home Assistantin integraatioihin kuuluvat myös lisäosat, jotka mahdollistavat omavalmisteisten laitteiden liittämisen järjestelmään esimerkiksi MQTT- tai HTTP-protokollan avulla. Käyttäjä voi määritellä, miten näiden protokollien mukaiset viestit tulkitaan, jolloin laitteet voidaan esittää järjestelmässä sensoreina, kytkiminä, valaisimina tai muina entiteetteinä.




Kuva 2. Home Assistant integraatiota (Home Assistant 2025)

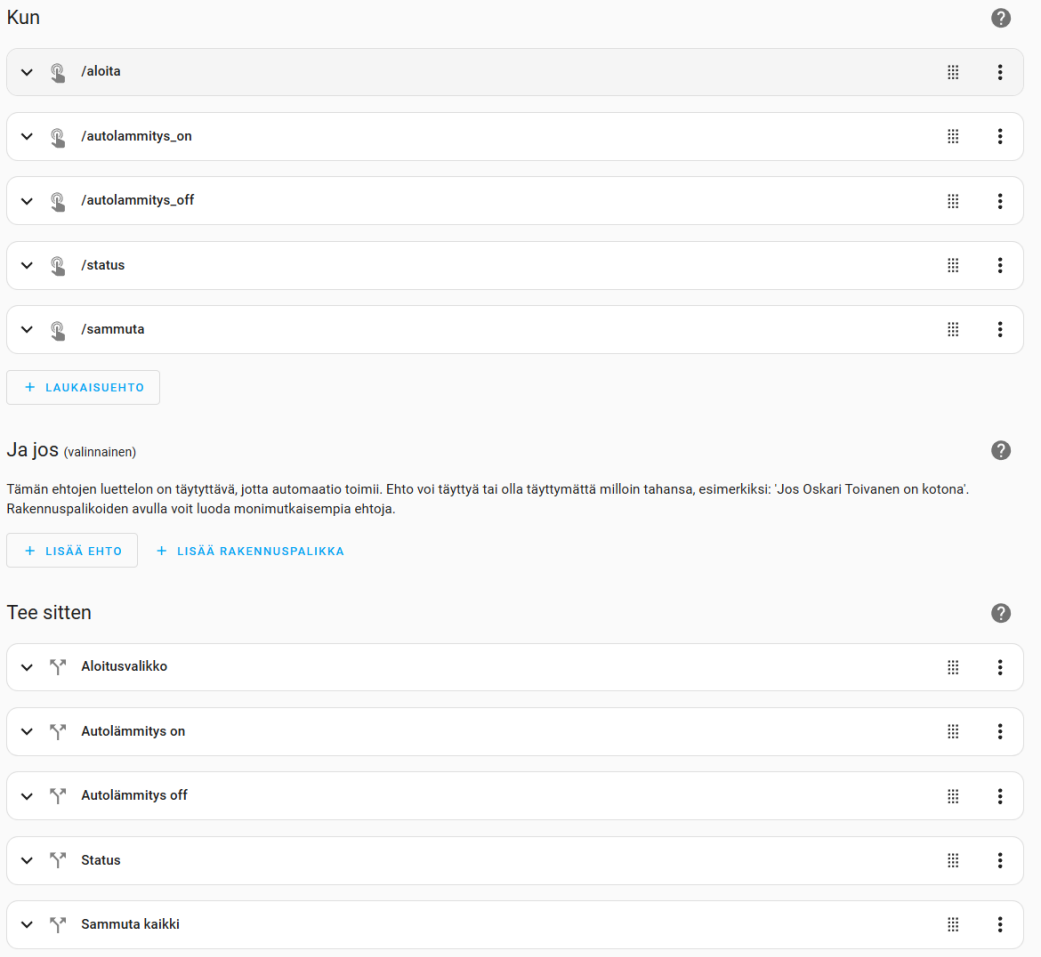
## 3.3 Automaatiot

Home Assistantin perusajatus on paitsi ohjata, myös automatisoida kodin laitteita ja niihin liitettyjä palveluita. Keskeinen osa tätä ovat *automaatiot.* Automaatiot ovat Home Assistantiin ohjelmoitavia työnkulkuja.

Automaatiolla on aina yksi tai useampi laukaisin (trigger), joka käynnistää työnkulun. Automaatio voidaan laukaista esimerkiksi vuorokaudenaikaan, sensoriarvoon, sensoriarvon muutosnopeuteen tai painikkeen painallukseen perustuen.

Laukaisimen lisäksi automaatiolle määritetään suoritettavat toiminnot (action) – esimerkiksi viive, termostaatin asetusarvon muuttaminen, ilmastointilaitteen nopeuden muuttaminen tai valojen sammuttaminen.

Automaatioita voidaan luoda ja hallinnoida lukuisilla eri tavoilla. Yleisimmin automaatioita luodaan ja ylläpidetään joko Home Assistantin oman käyttöliittymän kautta, mikä sisältää käyttäjäystävällisen, visuaalisen työkalun tähän tarkoitukseen tai muokkaamalla Home Assistantin YAML -muodossa olevia konfigurointitiedostoja. Automaatioita on mahdollista kirjoittaa myös esimerkiksi JavaScript- tai Python -ohjelmointikielillä.




Kuva 3. Esimerkki Telegram -integraatioon perustuvan automaation konfiguroinnista käyttöliittymän kautta

```yaml
alias: Telegram valikko
description: >-
  Automaatio triggeröityy kun telegram botille lähetetään komento. Automaatio
  lähettää vastauksena inline painikkeet toimintoja varten
triggers:
  - event_type: telegram_command
    event_data:
      command: /aloita
    id: aloita
    alias: /aloita
    trigger: event
  - alias: /autolammitys_on
    event_type: telegram_callback
    event_data:
      command: /autolammitys_on
    id: autolammitys_on
    trigger: event
  - alias: /autolammitys_off
    event_type: telegram_callback
    event_data:
      command: /autolammitys_off
    id: autolammitys_off
    trigger: event
  - alias: /status
```

Kuva 4. Ote saman automaation YAML-määrityksestä

# 4 MIKROKONTROLLERIT JA NIIDEN KÄYTTÖ IOT-RATKAISUISSA

## 4.1 Mikrokontrollerien perusteet

Mikrokontrolleri on pohjimmiltaan yksittäisellä mikropiirillä toimiva tietokone. Se on suunniteltu tiettyä tehtävää varten sulautetuissa järjestelmissä ilman monimutkaista käyttöjärjestelmää. Toisin kuin perinteinen mikroprosessori, mikrokontrolleri yhdistää samalle alustalle prosessorin, muistin ja I/O-toiminnot. (IBM n.d.)

Pelkän mikrokontrollerin käyttö vaatii ympärilleen lisäelektroniikkaa, jotta sitä voi ohjelmoida ja käyttää tulo- ja lähtösignaalien hallintaan. Tämän vuoksi mikrokontrollereiden ympärille on kehitetty erilaisia kehitysalustoja, joissa varsinaisen mikrokontrollerin ympärille on lisätty tarvittavat komponentit ohjelmointia ja käyttöä varten.

Mikrokontrollereita käytetään laajasti erilaisissa sulautetuissa järjestelmissä, kuten kodin automaatiossa, teollisuusautomaatiossa, lääketieteellisissä laitteissa, kulkuneuvojen ohjausjärjestelmissä ja IoT (Internet of Things) -ratkaisuissa. Ne soveltuvat erityisesti sovelluksiin, joissa tarvitaan reaaliaikaista ohjausta, alhaista tehonkulutusta ja kompaktia rakennetta. (GeeksForGeeks n.d.)

Kehitysalustat jaetaan usein erilaisiin tuotemerkkeihin ja tuoteperheisiin, kuten Arduino ja NodeMCU. Yksittäinen tuoteperhe voi sisältää lukuisia malleja erilaisin ominaisuuksin. Mallien välillä ominaisuudet vaihtelevat esimerkiksi laskentatehon, muistin määrän ja langattomien yhteyksien suhteen. Mikäli käyttökohteessa ei tarvita esimerkiksi WiFi -yhteyttä, voidaan valita edullisempi kehitysalusta.


Kuva 5. Lolin Wemos D1 Mini (WEMOS n.d.)

Mikrokontrollereiden negatiivisina huomioitavina puolina voidaan nähdä muun muassa vaatimaton laskentateho, heikko laajennettavuus, jossain määrin rajoittuneet kehitysympäristöt sekä tarvittava elektroniikkaosaaminen erityisesti I/O-toimintoja hyödyntäviä IoT-ratkaisuja tuotettaessa. (GeeksForGeeks n.d.)

## 4.2 ESP8266 ja sen soveltuvuus kotiautomaatioprojekteihin

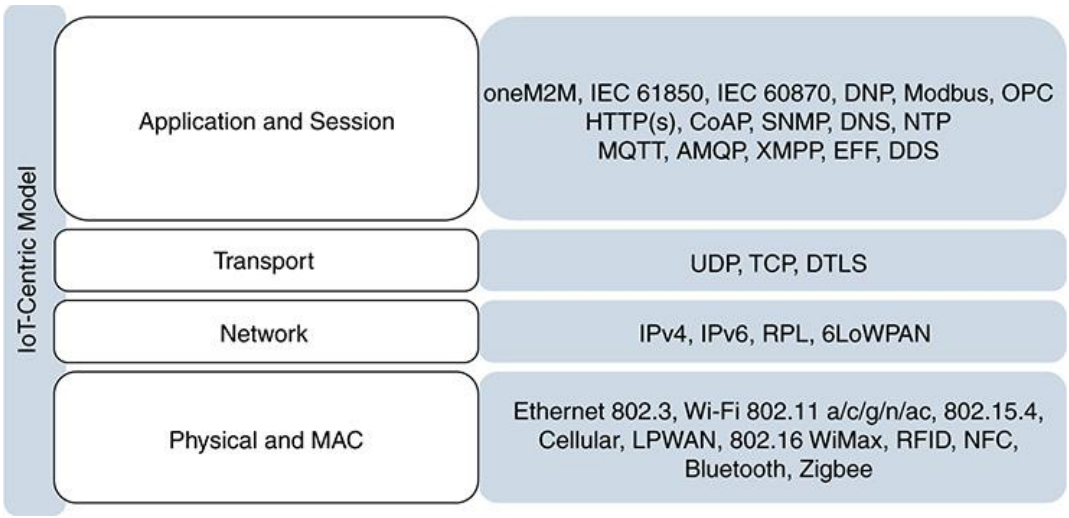
ESP8266 on SoC-piiri, jossa on sisäänrakennettu 32-bittinen mikrokontrolleri, Wi-Fi-radio ja tarvittavat oheispiirit langattomien IoT-ratkaisujen toteuttamiseen. ESP8266-piiriin perustuvat kehitysalustat sopivat erityisen hyvin kotiautomaatioprojekteihin verrattuna moneen muuhun mikrokontrolleriin, koska I/O -toimintojen lisäksi siitä löytyy usein välttämätön langaton verkkoyhteys. (Nabto n.d.)

WiFi -yhteydellä varustettu kehitysalusta pystyy kommunikoimaan muiden kotiautomaatiolaitteiden kanssa käyttäen tarpeeseen sopivaa protokollaa kuten HTTP tai MQTT.

ESP8266 -pohjaiset kehitysalustat ovat suosittuja kotiautomaatioyhteisöissä. Kehitysalustoille on saatavilla runsaasti dokumentaatiota ja valmiita esimerkkiratkaisuja erilaisiin IoT -tarpeisiin.

## 4.3 Mikrokontrollerien viestintäprotokollat

Lähes kaikissa käyttötapauksissa, erityisesti IoT-ratkaisuissa osa mikrokontrollerin ydintehtävää on muiden järjestelmien ja komponenttien välinen kommunikaatio. Mikrokontrollerit mahdollistavat lukuisten eri protokollien hyödyntämisen OSI-mallin eri kerroksissa. (Cisco Press 2019.)




Kuva 6. IoT-keskeinen OSI-malli (Cisco Press 2019)

Tärkeitä fyysisen- ja siirtokerroksen protokollia ovat esimerkiksi seuraavat:

- **UART (Universal Asynchronous Receiver-Transmitter)** sarjamuotoinen tiedonsiirtoprotokolla, mahdollistaa kahden laitteen välisen viestinnän ilman kellosignaalia. Se on laajalti käytössä mikrokontrollerien ja muiden sulautettujen järjestelmien välisessä kommunikaatiossa. (Arduino 2024.)
- **I²C (Inter-Integrated Circuit)** sarjamuotoinen tiedonsiirtoprotokolla, joka mahdollistaa useiden laitteiden (esim. anturit ja mikrokontrollerit) liittämisen samaan väylään käyttäen vain kahta johtoa (SDA ja SCL). Se on hyödyllinen erityisesti tilanteissa, joissa tarvitaan vähäistä kaapelointia. (Zambetti, et al., 2024.)
- **SPI (Serial Peripheral Interface)** on sarjaliitäntä, jota käytetään nopeaan tiedonsiirtoon lyhyillä etäisyyksillä esimerkiksi näyttöjen, muistikorttien ja antureiden kanssa (Arduino 2023).
- **LoRa (Long Range)** Pitkän kantaman ja vähäisen virrankulutuksen langaton viestintäprotokolla, jota käytetään esimerkiksi kaupunkien IoT-ratkaisuissa ja maataloudessa (Arduino 2025).
- **Bluetooth** Lyhyen kantaman langaton tiedonsiirtoteknologia, jota käytetään esimerkiksi puettavissa laitteissa ja kotiautomaatiossa (Siebeneicher, 2024).
- **Ethernet** Kaapelipohjainen verkkoteknologia, joka mahdollistaa luotettavan ja nopean tiedonsiirron. Käytetään laajalti tietoverkoissa ja IoT-laitteissa, joissa tarvitaan vakaata yhteyttä. (CISCO, n.d.)

- **WiFi (Wireless Fidelity)** Yleisesti myös kodeissa käytettävä langaton tiedonsiirtoprotokolla, joka mahdollistaa nopean tiedonsiirron ja on yhteensopiva olemassa olevan verkkoinfrastruktuurin kanssa (CISCO n.d.).
- **1-Wire** Sarjamuotoinen tiedonsiirtoprotokolla, jota käytetään erityisesti yksinkertaisten anturei-den, kuten lämpötila- ja kosteustunnistimien, liittämiseen mikrokontrolleriin tai muuhun laitteis-toon. Protokollan erityispiirre on, että se käyttää vain yhtä johdinta tiedonsiirtoon, mikä vähentää kaapelointitarvetta ja tekee siitä erityisen kätevän pienissä ja kapeissa tiloissa. (Community 2024.)

Sovellustason viestintäprotokollista opinnäytetyön kannalta merkityksellisiä ovat muun muassa seu-raavat:

- **MQTT (Message Queuing Telemetry Transport)** on kevyt ja tehokas, erityisesti IoT-laitteille suunniteltu viestinvälitysprotokolla. Perustuu julkaisu- ja tilausmalliin (publish/subscribe), jossa viestinvälittäjänä toimii välityspalvelin (broker). Laitteet voivat tilata (subscribe) ja julkaista (pub-lish) viestejä eri aihepiireihin (topics). (AWS n.d.)
- **CoAP (Constrained Application Protocol)** on kevyt vaihtoehto HTTP:lle, suunniteltu pienite-hoisille IoT-laitteille ja rajallisiin verkkoihin. Perustuu UDP-protokollaan, minkä vuoksi se on TCP-pohjaista HTTP:tä nopeampi. Tukee REST-tyyppistä viestintää kuten HTTP, mutta kuluttaa vähemmän resursseja. (GeeksForGeeks n.d.)
- **WebSockets** mahdollistaa kahdensuuntaisen, jatkuvan yhteyden asiakkaan ja palvelimen vä-lillä. Toimii TCP-pohjaisesti, mikä mahdollistaa reaaliaikaisen viestinnän. Soveltuu erityisesti re-aaliaikaisiin IoT-sovelluksiin, kuten sensoridatan jatkuvaan lähetykseen. (The Curve, 2024.)

Käytettävät viestintäprotokollat päätettiin ennen työn tarkemman suunnittelun aloittamista, sillä ne ohjaavat myös laite- ja järjestelmäarkkitehtuurivalintoja. Fyysisen kerroksen protokolliin liittyvät rat-kaisut on kuvattu komponenttivalinnat -kappaleessa. Sovellustason viestintäprotokollista muodostet-tiin taulukon 1 vertailu päätöksenteon tueksi. (Nabto n.d.)

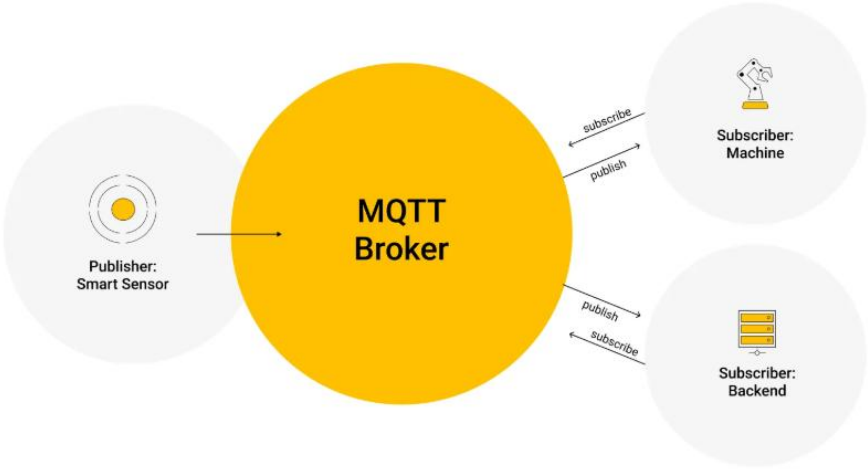
Taulukko 1. Viestintäprotokollien vertailutaulukko IoT-ympäristössä

| **Protokolla** | **Yhteystapa** | **Kulutus & tehok-kuus** | **Käyttötapaukset** | **Haittapuolet** |
|---|---|---|---|---|
| MQTT | Publish/Subscribe | Kevyt, pieni kaistanle-veys | IoT, hajautetut sensoriverkot, Home Assistant | Tarvitsee MQTT-välitys-palvelimen |
| CoAP | Rest over UPD | Kevyt, vähäinen re-surssinkulutus | Pienitehoiset IoT-laitteet, verkon rajatut resurssit | Ei luotettavaa yhteyttä ilman li-sämekanismeja |
| WebSockets | Jatkuva kahden-suuntainen yhteys (TCP) | Nopea, mutta kuluttaa enemmän resursseja kuin MQTT | Reaaliaikainen viestintä, jatkuvat yhteydet | Kuluttaa enem-män muistia ja kaistaa, jatkuva yhteys kuormit-taa palvelinta |

Opinnäytetyön viestintäprotokollaksi valittiin vertailun perusteella MQTT. Olemassa olevaan Home Assistant -toteutukseen sisältyi MQTT-välityspalvelin, sen konfiguroimiseen oli omakohtaista koke-musta ja standardoidun MQTT-protokollan yleisyys mahdollistaa laitteen integroimisen helposti myös muihin sovelluksiin ja alustoihin.

## 4.4 MQTT-protokolla

MQTT (Message Queuing Telemetry Transport) on kevyt ja tehokas viestinvälitysprotokolla, joka on suunniteltu erityisesti IoT-laitteille ja epävakaille verkkoyhteyksille. Se perustuu julkaisu/tilaus (publish/subscribe) -malliin, jossa laitteet voivat lähettää ja vastaanottaa viestejä välityspalvelimen (brokerin) kautta. (HiveMQ n.d.)




Kuva 7. MQTT julkaisu/tilaus arkkitehtuuri (HiveMQ n.d.)

MQTT-välityspalvelimen (brokerin) toiminta perustuu publish/subscribe-malliin, jossa viestit välitetään asiakkailta toisille niiden tilaamien otsikoiden (topic) perusteella. Välityspalvelin toimii keskeisenä solmukohtana, joka hallinnoi viestiliikennettä ja varmistaa, että viestit päätyvät oikeille vastaanottajille. Välityspalvelin käsittelee esimerkiksi asiakkaiden lähettämien viestien metatietojen sisältämät viestin uudelleen lähettämiseen ja säilyttämiseen liittyvät ohjaukset. (HiveMQ n.d.)

MQTT-asiakkaat julkaisevat viestejä välityspalvelimen kautta. Lähetetty viesti sisältää aina otsikkotiedon, minkä perusteella välityspalvelin jakaa viestin halutuille asiakkaille sen mukaan, mitä otsikoita asiakkaat ovat tilanneet. Käytännössä yksi tai useampi asiakas voi vastaanottaa saman viestin, mikäli asiakkaat ovat tilanneet saman otsikon. MQTT-viestejä julkaistaessa viestiin liitetään tarvittaessa myös ohjeet viestin säilyttämiseen ja uudelleenlähettämiseen liittyen, esimerkiksi viestin kriittisyyden mukaan. (HiveMQ n.d.)

Viestinvälityksen luotettavuusvaatimusten mukaan vastaanottajan ja asiakkaan välille voidaan määrittää kolmetasoinen QoS (Quality of Service) -asetus. Standardin mukaisia Quality of Service -viestinvälitysasetuksia ovat

1) Korkeintaan kerran (0)
2) Vähintään kerran(1)
3) Tarkalleen yhden kerran (3)

QoS 0 -tason viestit ovat niin sanottuja best-effort-viestejä. Viesti lähetetään vain kerran, eikä vastaanottajalta odoteta kuittausta sen vastaanottamisesta. Viestiä ei myöskään säilytetä eikä lähetetä uudelleen, joten sen välitys voi epäonnistua esimerkiksi verkkoyhteysongelmien vuoksi. (HiveMQ n.d.)

QoS 1 -tason viestit toimitetaan vastaanottajalle vähintään kerran (at least once). Lähettäjä säilyttää kopion viestistä, kunnes vastaanottaja lähettää PUBACK-kuittauksen. Mikäli kuittausta ei tapahdu määräajassa, viesti lähetetään uudelleen. Tämä voi johtaa siihen, että sama viesti vastaanotetaan useammin kuin kerran. (HiveMQ n.d.)

QoS 2 -tason viestit toimitetaan vastaanottajalle täsmälleen kerran (exactly once). Tämä saavutetaan lisäkättelyiden avulla, jotka varmistavat, että viesti on sekä vastaanotettu että käsitelty, mutta ei duplikoitu. QoS-tason nostaminen lisää lähettäjän ja vastaanottajan välisten kättelyiden määrää, mikä kuormittaa välityspalvelinta ja verkkoa enemmän. (HiveMQ n.d.)

Tässä opinnäytetyössä käsiteltävässä käyttökohteessa QoS 0 on riittävä taso, sillä viestien mahdollinen häviäminen ei aiheuta merkittäviä ongelmia.

# 5 TYÖN SUUNNITTELU

## 5.1 Vaatimusmäärittely

Projektia suunniteltaessa ja vaatimusmäärittelyä koostaessa noudatettiin löyhästi ketterän kehityksen menetelmiä. Laitteen aiotusta käyttötarkoituksesta muodostettiin nk. käyttäjätarinoita, joiden avulla pystyttiin tunnistamaan loppukäyttäjälle lisäarvoa tuovat toiminnallisuudet ja ohjaamaan toteutusjärjestystä sen mukaisesti. Vaatimusmäärittelystä tuotettiin yksityiskohtaisempia, ohjelmointiin liittyviä tehtäviä toteutusvaiheessa. Laitteelle määritellyt vaatimukset on esitetty taulukossa 2.

Prioriteetit:

1 = Välttämätön jotta laite tuottaa lainkaan lisäarvoa käyttäjälleen tai sisältää turvallisuusriskin

2 = Toivottu ja suunniteltu ominaisuus, joka tuottaa lisäarvoa mutta puuttuessaan integraation hyödyntäminen on edelleen mahdollista

3 = Käyttökokemusta parantava ominaisuus, joka ei vaikuta perustoiminnallisuuteen

Taulukko 2. Vaatimusmäärittelytaulukko

| Vaatimus | Prioriteetti (1–3) | Kuvaus |
|---|---|---|
| Laitteen on toivuttava sähkökatkoista ja verkkoyhteyden menettämisestä | 1 | Sähkökatkon- tai verkkoyhteyden menetyksen jälkeen laitteen on toivuttava itsenäisesti toimintakykyiseksi. |
| Laite ei saa häiritä lämmitysjärjestelmän itsenäistä toimintaa | 1 | Laite on kytkettävä ja ohjelmoitava niin, ettei se voi keskeyttää kattilan toimintaa tai alentaa menoveden lämpötilaa tahattomasti |
| Laitteen on raportoitava sensoriarvoja MQTT-protokollaa käyttäen | 1 | Sensoridatan tuottaminen kotiautomaatio- tai muuhun sovellukseen |
| Laiteessa olevaa relettä voidaan ohjata MQTT-protokollan yli, ja releen tila raportoidaan takaisin | 2 | IoT-laite ohjaa ulkoista laitetta releen avulla. Tämä on suuri osa suunniteltua toiminnallisuutta, mutta pelkkä sensoridatan lukeminenkin tuottaa hyötyjä |
| Laitteessa on oma Web-käyttöliittymä josta se voidaan uudelleenkonfiguroida | 2 | Esimerkiksi WLAN -yhteyden muuttuessa laite voidaan konfiguroida yhdistämään toiseen verkkoon ilman uudelleenohjelmointia |
| Laite on koteloitu | 1 | IoT-laite on koteloitava, jotta se voidaan asentaa loppusijoituspaikkaan. Koteloinnissa on huomioitava mm. vedonpoistot kaapeleille |
| Laitteessa on näyttö, josta oloarvoja ja yhteyden tilaa voidaan seurata paikallisesti | 3 | Laitteen käyttökelpoisuutta lisää merkittävästi, mikäli lämpötilojen ja releen oloarvot sekä IP-osoitteet ja yhteyden tilat voidaan esittää käyttäjälle näytöltä |
| Laitteessa on LED-indikaattori, josta laitteen tilan näkee nopeasti | 3 | Näkyvä LED-indikaattori laitteen koteloon, joka esimerkiksi valmius- ja häiriötilat värikoodein. Tämän avulla häiriötilanteet voidaan havainnoida nopeasti teknisessä tilassa asioidessa. |
| Laitteen on toimittava täysin paikallisesti ilman ulkoisia riippuvuuksia | 1 | Laite ei saa olla riippuvainen mistään sisäverkon ulkopuolisesta palvelusta tai osoitteesta. Laiteen on säilyttävä toimintakykyisenä, vaikka pääsy internetiin estettäisiin tietoturva- tai stabiiliussyistä. |

## 5.2 Projektinhallinta

Ennen opinnäytetyön aloittamista työn valmiiksi saattamiseen liittyvät riskit kartoitettiin ja niiden hallintakeinot määritettiin.

Taulukko 3. Projektin riskit taulukoituna

| Riski | Vakavuus (1–5) | Todennäköisyys (1–5) | Hallintakeino |
|---|---|---|---|
| Mikrokontrollerin resurssit eivät riitä ohjelman suoritukseen | 4 | 3 | Perehdytään lähteisiin ja pyritään optimoimaan resurssien käyttö alusta alkaen. Toteutetaan toiminnallisuus kerrallaan tärkeimmästä alkaen |
| Ohjelma muodostuu liian kompleksiseksi hallittavaksi | 4 | 4 | Jaetaan ohjelma loogisiin kokonaisuuksiin, hyödynnetään luokkia ja kirjastoja, kommentoidaan koodia kattavasti |
| Laitteet eivät toimi | 5 | 2 | Laitteiden ollessa fyysisesti rikki, hankitaan korvaavat (huom. toimitusajat). |
| Aikaa ei riitä opinnäytetyölle | 4 | 3 | Pyritään määrittämään tehtävät mahdollisimman tarkasti, jotta työmäärä on arvioitavissa. Keskitytään kriittisiin toiminnallisuuksiin ensisijaisesti. Allokoidaan esimerkiksi X määrä tunteja viikossa etukäteen opinnäytetyön tekemiselle. |

Projektin- ja tehtävien hallintatyökaluksi valittiin Microsoft Planner. Ennen projektin aloitusta tehtävät aikataulutettiin ja lisättiin Planneriin, jotta eri toiminnallisuuksien edistymistä pystyttiin seuraamaan.

## 5.3 Mikrokontrollerin ja kehitysalustan valinta

Kehitysalustaa valittaessa painotettiin kustannustehokkuutta, saatavuutta, liitettävyyttä ja fyysistä kokoa. Vertailuun otettiin mukaan myös Raspberry Pi yleisyytensä vuoksi, vaikka sitä ei lueta mikrokontrolleriksi. Kehitysalustat on otettu vertailuun mukaan ensisijaisesti saatavuutensa perusteella – tuotteet ovat yleisimmin kotimaisista tai ulkomaisista verkkokaupoista kohtuullisilla toimitusajoilla toimitettavissa. Myös Flash-muistin määrä oli tässä projektissa erityisen merkityksellinen, sillä web-käyttöliittymään liittyvät tiedostot tallennetaan laitteen Flash-muistiin.

Tiedot vertailutaulukkoon on kerätty valmistajilta ja jälleenmyyjiltä.

Taulukko 4. Vaihtoehtoisten laitteiden ominaisuuksien vertailutaulukko

| **Ominaisuus** | **Wemos D1 Mini** | **NodeMCU** | **Arduino Nano** | **Raspberry Pi 3B+** |
|---|---|---|---|---|
| Mikrokontrolleri / SoC | ESP8266 (32-bit) | ESP8266 (32-bit) | ATmega328P (8-bit) | Broadcom BCM2837 (64-bit) |
| Kellotaajuus | 80 / 160 MHz | 80 / 160 MHz | 16 MHz | 1.2 GHz |
| RAM | 80 KB | 80 KB | 2 KB | 1 GB |
| Flash-muisti | 4 MB (vaihtelee) | 4 MB (vaihtelee) | 32 KB | microSD, max 256 GB |
| WiFi | Kyllä | Kyllä | Ei | Kyllä |
| GPIO-pinnit | 11 | 17 | 14 | 40 |
| I/O protokollat | UART, I²C, SPI, PWM | UART, I²C, SPI, PWM | UART, I²C, SPI, PWM, ADC | UART, I²C, SPI, PWM |
| Analogiset tulot | 1 | 1 | 8 | 0 |
| Käyttöjärjes-telmä | Sulautettu | Sulautettu | Sulautettu | Esim. Linux |
| Virrankulutus | Noin 70–200 mA | Noin 70–200 mA | Noin 15–20 mA | Noin 400–800 mA |
| Käyttöjännite ja virtalähde | 3.3V (USB tai 5V pinni) | 3.3V (USB tai 5V pinni) | 5V (USB tai 5V pinni) | 5V (USB tai 5V pinni) |
| Ohjelmointi | C++ (Arduino) MicroPython | C++ (Arduino) MicroPython | C++ (Arduino) | Python, C, C++, JavaScript ym. |
| Fyysinen koko | 25.6 x 34.2 | 26 x 48 | 18 x 45 | 49 x 85 |
| Hinta (noin) | 2–6 € | 5–10 € | 3–6 € | 40–50 € |

Vertailutaulukon 4 perusteella valinta tehtiin Wemos D1 Minin ja NodeMCU:n välillä. Raspberry Pi variaatiot ovat merkittävästi kalliimpia, tarpeettoman tehokkaita ja vaativat erillisen käyttöjärjestelmän ylläpitämistä – tuote ei sovellu opinnäytetyön käyttötarkoitukseen. Arduino Nano ja sen eri variaatiot ovat kustannuksiltaan samaa tasoa ESP8266-pohjaisten kehitysalustojen kanssa, mutta siitä puuttuu WiFi-yhteys ja muistin määrä on vähäinen.

## 5.4 Komponenttien valinta

Kehitysalustan lisäksi laitteeseen tarvittiin lämpötilasensoreita, relemoduuli, LED-indikaattori ja näyttö. Komponenttien valintaa ohjasivat hinta, saatavuus ja johdotuksen yksinkertaisuus. Sarjaliitäntäisiä komponentteja priorisoitiin, koska ne vähentävät mikrokontrollerin I/O-pinnien tarvetta ja mahdollistavat laajennettavuuden.

Lämpötilasensoreina käytettiin valmiiksi koteloituja, roiskevedeltä suojattuja Dallas DS18B20 -sensoreita. Sensorin mittausalue on -55°C – 125°C ja käyttöjännite 3.0V – 5.5V, joten se soveltuu hyvin mikrokontrollerikäyttöön. DS18B20 käyttää sarjamuotoista 1-Wire-protokollaa, mikä mahdollistaa useiden antureiden kytkemisen samaan dataväylään käyttäen vain yhtä tiedonsiirtolinjaa. Jokaisella väylään liitetyllä sensorilla on yksilöllinen 64-bittinen osoite, minkä ansiosta mikrokontrolleri voi lukea niiden lämpötila-arvot erikseen. (Maxim integrated 2019.)

Kuva 8. Dallas DS18B20 lämpötilasensori (JemRF 2025)

Releeksi valittiin yksikanavainen moduuli, joka sisältää mikrokontrollerikäyttöön tarvittavat komponentit, liitännät ja kiinnityskohdat. Relemoduulin valinnassa tärkeimpiä tekijöitä olivat releen käyttöjännite, kelajännite, koskettimien toiminta (NO/NC) sekä jännitteen- ja kuormankesto. Valittu, yleisesti saatavilla oleva relemoduuli sisältää vaihtokoskettimet, joten se voi toimia sekä normaalisti auki (NO) että normaalisti kiinni (NC) -tilassa. Releen käyttö- ja kelajännite on 3,3 V, joten sitä voidaan ohjata suoraan mikrokontrollerin lähtöpinnistä ilman lisäkomponentteja. Koskettimien maksimijännite on 240 V AC / 30 V DC, ja niiden maksimivirta on 10 A sekä vaihto- että tasajännitteellä. Releen ohjaaminen vaatii yhden mikrokontrollerin lähtöpinnin. (Amazon 2025.)


Kuva 9. 3.3V relemoduuli (Amazon 2025)

Näyttönä käytettiin 0,96” OLED-näyttöä, jonka 128 × 64 pikselin resoluutio mahdollistaa tarvittavien tietojen esittämisen selkeästi. Näyttöä valittaessa priorisoitiin sarjamuotoisesti mikrokontrollerin kanssa kommunikoivia malleja johdotuksen yksinkertaistamiseksi ja tulo-/lähtöpinnien säästä-

miseksi. Valittu näyttö käyttää I²C-protokollaa, joka mahdollistaa useiden laitteiden liittämisen samaan väylään käyttäen vain kahta tulo-/lähtöpinniä (SDA ja SCL) virransyötön lisäksi. Väylän laitteet tunnistetaan yksilöllisen osoitteen avulla. Näyttö toimii 3,3 V – 5 V käyttöjännitteellä. (Amazon 2025.)


Kuva 10. I²C-liitäntäinen OLED-näyttö (Amazon 2025)

LED-indikaattorina käytettiin digitaalisesti ohjattavaa WS2812B RGB-LED:iä, joka käyttää yksijohtoista sarjamuotoista viestintäprotokollaa. Tarvittaessa useita LED-komponentteja voidaan kytkeä sarjaan, jolloin ne voidaan ohjata yksitellen lähetettävän signaaliketjun avulla. LED:in käyttöjännite on 5 V, ja sen värisyvyys on 24 bittiä (8 bittiä per väri: punainen, vihreä ja sininen). RGB-LED:in avulla laitteen tiloja voidaan indikoida eri värein. LED tarvitsee ainoastaan yhden lähtöpinnin. (Triopak Oy 2025.)

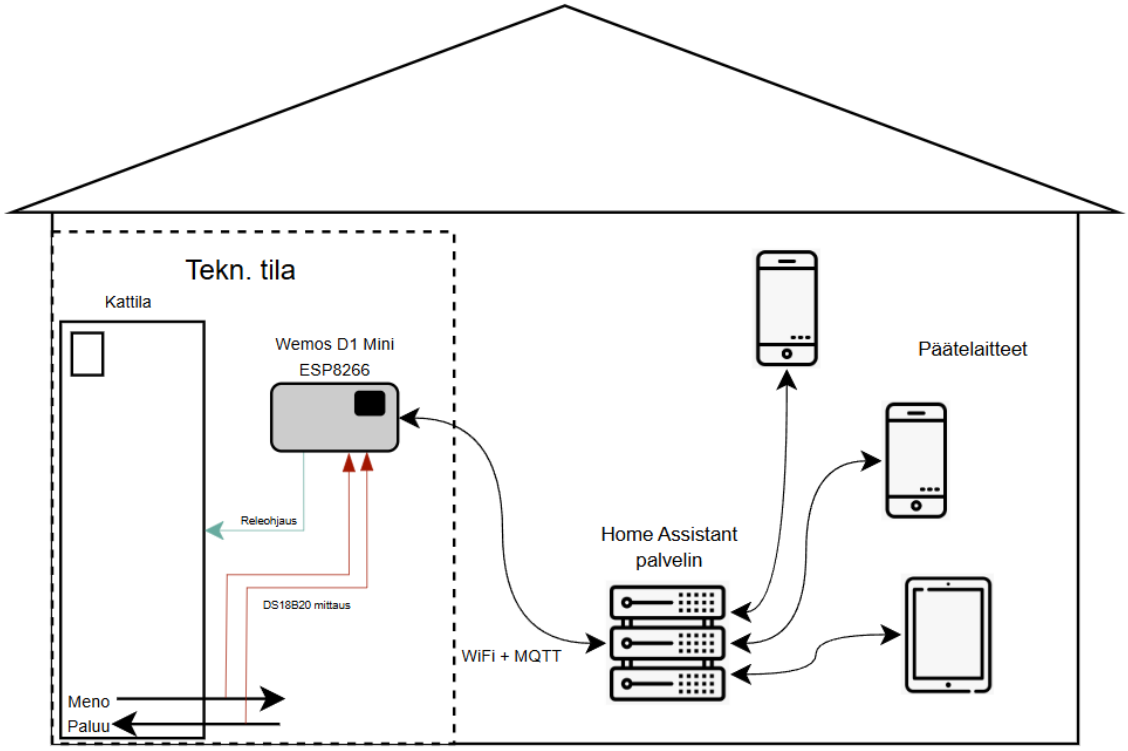
Kuva 11. WS2812B RGB-LED (Triopak Oy 2025)

Taulukko 5. Osaluettelo ja kustannusarvio

| Tuote | Määrä | Yksikköhinta | Summa |
|---|---|---|---|
| Wemos D1 Mini mikrokontrolleri | 1 | 3.50 € | 3.50 € |
| DS18B20 lämpötilasensori | 2 | 1,80 € | 3,60 € |
| WAGO -liittimet | 5 | 1.00 € | 5.00 € |
| 0.96” OLED I2C näyttö | 1 | 2,50 € | 2,50 € |
| WS2812B RGB-LED | 1 | 2,50 € | 2,50 € |
| 3D-tulostimen filamentit, ruuvit ja muut oheistarvikkeet | 1 | 5 € | 5 € |
| | | | **22,10 €** |

## 5.5 Laitteistoarkkitehtuuri

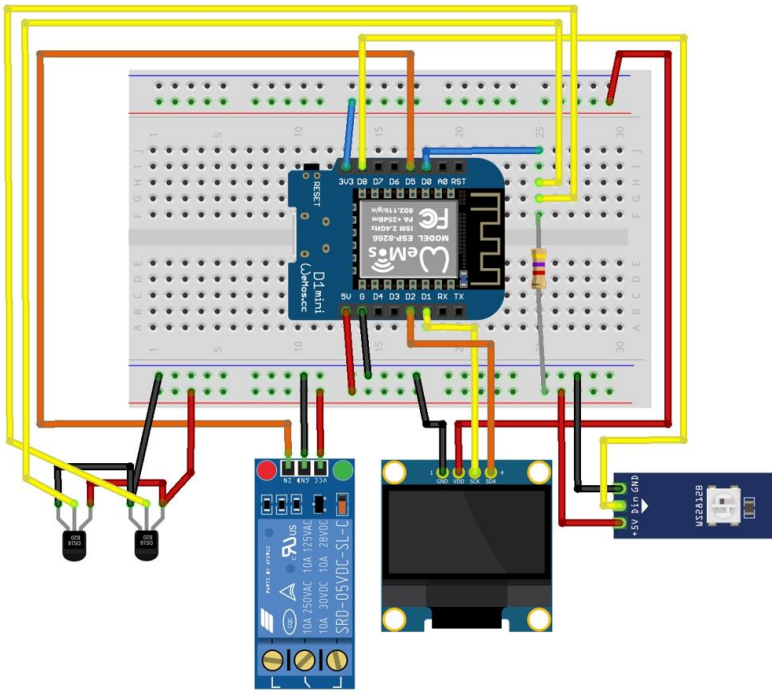
Ennen työn aloittamista suunniteltiin karkea laitteistoarkkitehtuuri, jota päivitettiin teknologiavalintojen tarkentuessa. Laitteistoarkkitehtuuriin piirrettiin fyysinen kuvaus järjestelmään liittyvistä laitteista ja niiden välisistä suhteista. IoT-laite sijoitetaan omakotitalon tekniseen tilaan, josta laite kommunikoi WiFi-yhteyden avulla sisäverkossa olevan MQTT-välityspalvelimen kanssa. Vastaavasti samassa verkossa olevat muut laitteet voivat kommunikoida IoT-laitteen kanssa MQTT-viestein.




Kuva 12. Laitteistoarkkitehtuuri ja fyysinen kuvaus

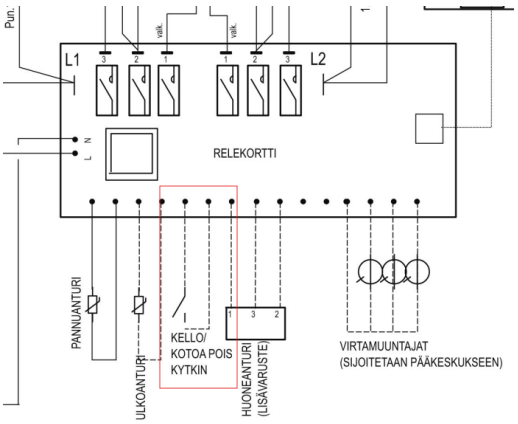
# 6 SÄHKÖISET KYTKENNÄT

IoT-laitteen sisäiset kytkennät suunniteltiin Fritzing-ohjelmalla koekytkentälevynäkymää hyödyntäen. Kytkennät testattiin koekytkentälevyllä ennen koteloon asentamista. DS18B20 lämpötila-antureille lisättiin 4,7 kΩ ylösvetovastus signaalin stabiloimiseksi.



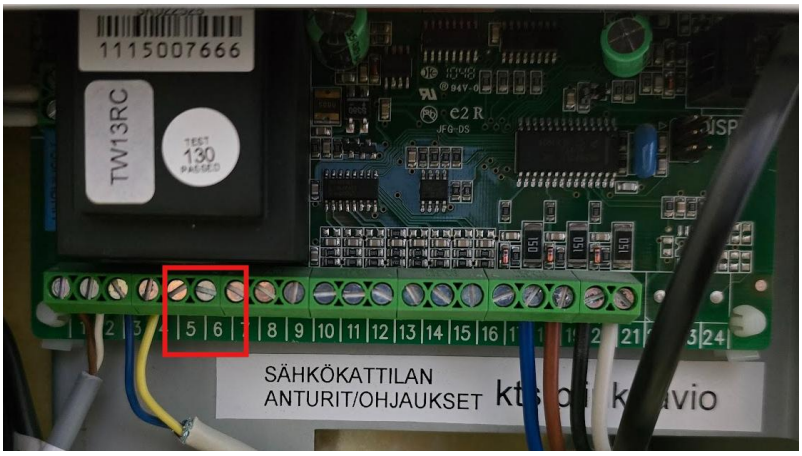

Kuva 13. Mikrokontrollerin kytkennät

Laitteen kytkemiseksi sähkökattilaan kytkentäpiste oli tarkastettava kattilan sähkökaaviosta. Sähkökattilan ohjauskortissa on liittimet, joihin ulkopuolinen potentiaalivapaa kosketin voidaan kytkeä. Usein käytetään fyysistä katkaisijaa, tässä tapauksessa IoT-laitteen rele kytketään kuvaan 14 merkittyyn signaalituloon. Ohjauskortin signaalijännite on 5 V, joten se ei aiheuta erityisvaatimuksia kaapeloinnille tai komponenteille.



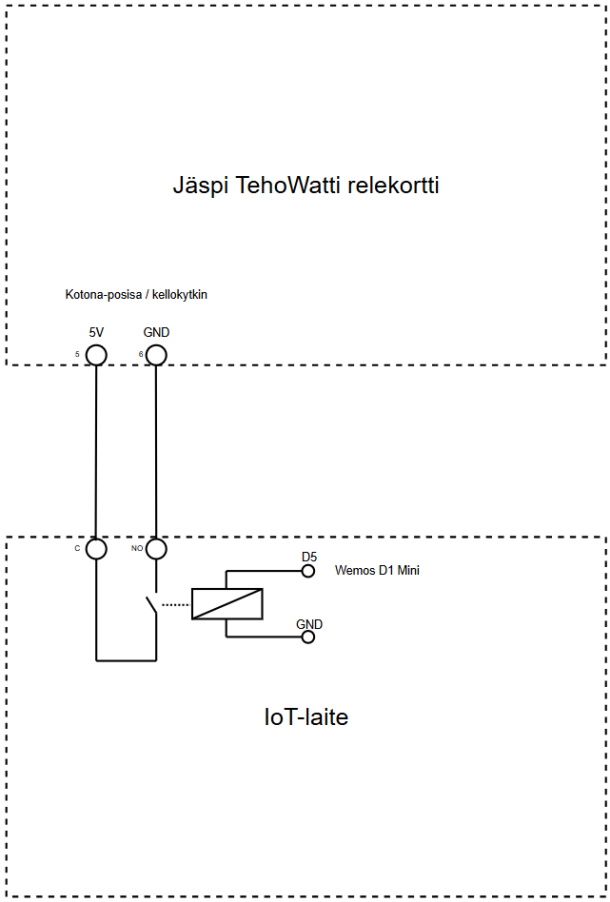

Kuva 14. Jäspi TehoWatti relekortin piirikaavio (Jäspi 2016)

Kuvissa 15 ja 16 on esitetty sähkökattilan relekortin fyysiset kytkentäpisteet sekä IoT-laitteen ja kattilan välinen kytkentä




Kuva 15. Kattilan piirikortin fyysiset liittimet

Sähkökattilan ja IoT-laitteen välisestä kytkennästä piirrettiin kytkentäkaavio käytettävissä olevin työkaluin.




Kuva 16. TehoWatin ohjaussignaalin kytkentä IoT-laitteeseen

# 7 KOTELO JA LAITTEISTON SUOJAUS

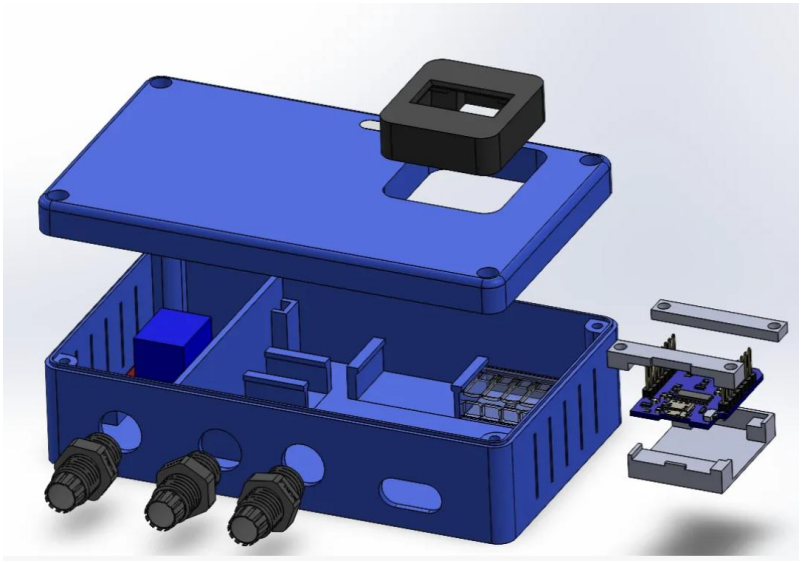
IoT-laitteet on koteloitava ja suojattava luotettavan toiminnan varmistamiseksi. Koteloa suunniteltaessa ja valittaessa on otettava huomioon muun muassa komponenttien vaatima jäähdytys, kotelomateriaalin radiosignaalien läpäisykyky, huollettavuus ja saavutettavuus, mahdollinen häiriösuojaus, kaapeleiden läpiviennit ja vedonpoistot sekä sijoituspaikan mukaan esimerkiksi korroosionkesto ja IP-luokitus. (Polycase 2020.)

Opinnäytetyössä rakennettu IoT-laite sijoitettiin omakotitalon tekniseen tilaan. Tila ei asettanut erityisvaatimuksia esimerkiksi kosteus- tai pölysuojaukselle. Suunnittelun lähtökohtina käytettiin seuraavia vaatimuksia

- Laite on oltava siististi koteloitu, ja kotelon on oltava mahdollisimman kompakti
- Kotelossa on oltava tarvittavat läpiviennit kaapeleille vedonpoistoin
- Kotelon on mahdollistettava kokoonpanon laajentaminen ja päivittäminen
- Kotelossa on oltava seinäkiinnitysmahdollisuus
- Kotelon on oltava avattavissa ja suljettavissa
- Rele on osastoitava, sillä releelle tuodaan vierasjännite ulkoisesta laitteesta
- Kotelon on mahdollistettava ilmavirtaus komponenttien jäähdyttämiseksi

Suunnittelu aloitettiin hahmottelemalla komponenttien sijainti toisiinsa nähden mahdollisimman tiiviisti, mutta kuitenkin niin että komponentit voidaan johdottaa. Suuntaa antavien ulkomittojen ja komponenttien sijainnin määrittämisen jälkeen varsinainen suunnittelutyö voitiin aloittaa SolidWorks 3D-mallinnusohjelmalla.

Suunnitteluohjelmaan kerättiin ensin joko valmiit 3D-mallit käytetyistä komponenteista, tai niiden puuttuessa mallinnettiin karkea vedos fyysisen komponentin mukaan. Komponenttien ympärille mallinnettiin kotelon runko, kansi ja komponenttien kiinnitysosat. Koko laitekokonaisuudesta tehtiin SolidWorks -mallinnusohjelmassa kokoonpano, jotta tarvittavat etäisyydet, välykset ja sijainnit voitiin mitoittaa.



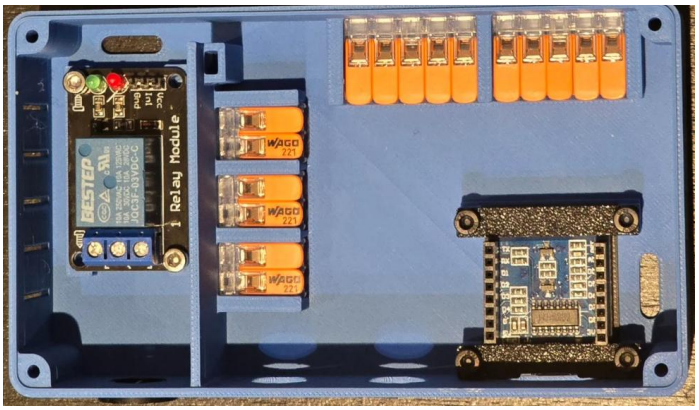
Kuva 17. 3D-mallinnettu kokoonpano

Koteloon tehtiin kolme läpivientiä tarvittaville sensori- ja relekaapeleille, jotka mitoitettiin teollisten M12/PG7 -vedonpoistoholkkien mukaan. Tämän lisäksi laitteen alareunaan mallinnettiin aukko mikrokontrollerin Micro-USB-liittimelle. Mikrokontrollerin USB-liitin jätettiin avoimeksi virransyöttöä ja ohjelmointia varten. Laitteen molemmilla sivuilla on jäähdytysaukot ja rele on erotettu muusta elektroniikasta väliseinän avulla.

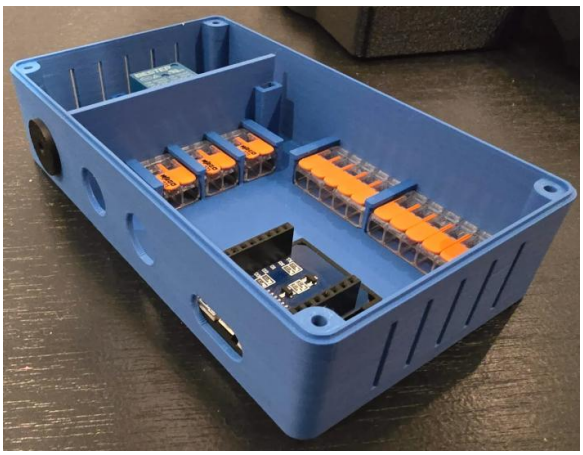
Kotelo tulostettiin FDM (Fused depositioning model) tulostimella. 3D-mallinnetut kotelon osat vietiin SolidWorksista viipalointiohjelmissa yleisesti käytettyyn 3MF-tiedostomuotoon. Viipalointiohjelman (slicer) tehtävä on määrittää tulostusasetukset, viipaloida tulostettava malli vaakasuuntaisesti ennalta määritetyn paksuisiksi kerroksiksi, ja lopulta muuntaa tulostustehtävä 3D-tulostimen ymmärtämään muotoon.

3D-tulostetun kappaleen aiottu loppukäyttö määrittää materiaalivalinnan. Kuluttajatulostimilla yleisimmin käytetyt materiaalit ovat PLA-, PETG- ja ABS-muovit. Muovilaatujen ominaisuudet eroavat toisistaan mekaanisen kestävyyden sekä lämmön- ja kemikaalien kestävyyden osalta. Vastaavasti materiaalien tulostettavuus riippuu näistä ominaisuuksista – yleisesti vaativampaan käyttöön soveltuvat materiaalit ovat haasteellisempaa tulostaa.

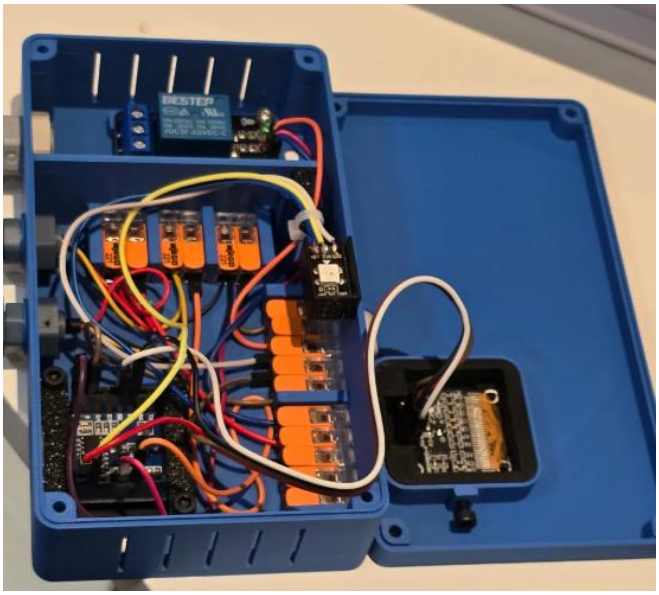
Opinnäyteyön laitteen ei todettu joutuvan poikkeuksellisen lämpö- tai kemikaalirasituksen kohteeksi eikä siihen kohdistu mekaanista kuormitusta. Tulostusmateriaaliksi valikoitui edullinen ja helposti tulostettava PLA. Kotelon tulostusaika oli noin kolme tuntia, ja PLA-materiaalia kului 120 grammaa.




Kuva 18. Kotelon runko ilman johdotusta



Kuva 19. Kotelon runko



Kuva 20. Komponentit johdotettuna


Kuva 21. Valmis kokoonpano

# 8 OHJELMOINTI

## 8.1 Ohjelmointikielen valinta

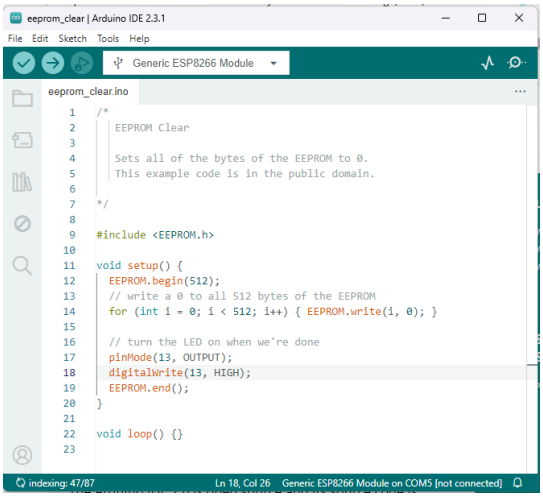
ESP8266-mikrokontrolleria voidaan ohjelmoida useilla eri kielillä, kuten C++, MicroPython, JavaScript ja Lua. Tässä projektissa vaihtoehdoiksi valikoituivat C++ (Arduino-ympäristössä) ja MicroPython, perustuen saatavilla olevaan osaamiseen ja kehitysympäristön vaatimuksiin. Käytetty C++ pohjautuu C++11-standardiin, mutta sisältää Arduino-alustan tarjoamia abstraktioita ja kirjastotukea, mikä helpottaa kehitystä. (Harshvardhan, 2025.)

Ohjelma päätettiin toteuttaa C++-kielellä, sillä se tarjoaa kattavan dokumentaation ja laajan yhteisötuen. MicroPython olisi ollut syntaksiltaan helppo vaihtoehto Pythonia tunteville, mutta sen standardikirjastot poikkeavat usein tavallisesta Pythonista joko syntaksiltaan tai käytettävissä olevien toimintojen osalta. Koska projektissa tarvittiin monipuolisia kirjastoja ja tehokasta resurssienhallintaa, C++ valikoitui paremmaksi vaihtoehdoksi.

## 8.2 Ohjelmointiympäristö

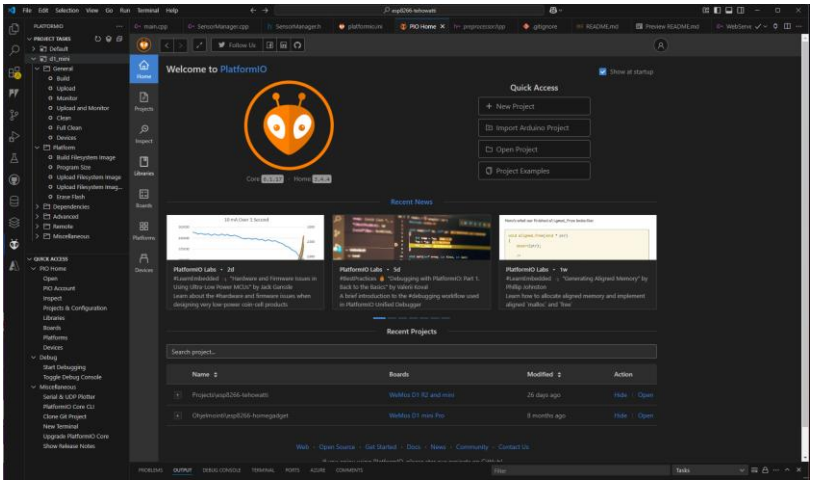
ESP8266 -pohjaisten kehitysalustojen ohjelmointiin on valittavissa useita eri kehitysympäristöjä (Integrated Development Environment, IDE). Kehitysympäristö on työkalu, joka tehostaa ohjelmistokehitysprosessia yhdistämällä samaan työkaluun muun muassa koodin kirjoittamisen, virheenkorjauksen ja mikrokontrollerien yhteydessä ohjelman kääntämisen ja laitteeseen lataamisen (The Codest n.d.)

Ohjelmointiympäristön valinta rajoittui Arduino IDE:n ja Visual Studio Code -ohjelmaan lisäosana asennettavaan PlatformIO -lisäosaan aiempaan kokemukseen perustuen. Arduino IDE on lähtökohtaisesti Arduino -kehitysalustojen ohjelmointiin tarkoitettu kehitysympäristö, mutta tarvittavat kirjastot asentamalla sillä voi ohjelmoida myös ESP8266 -pohjaisia kehitysalustoja. Arduino IDE sisältää kaikki tarvittavat perustoiminnallisuudet ohjelman kirjoittamiseksi, kääntämiseksi ja laitteeseen lataamiseksi. Muihin ohjelmoinnissa yleisesti käytettyihin kehitysympäristöihin tottuneelle sen käyttöliittymän tarjoamat mahdollisuudet voivat kuitenkin tuntua hyvin rajoittuneelle. Arduino IDE ei tarjoa esimerkiksi samassa käyttöliittymässä versionhallintaan liittyviä toiminnallisuuksia. Arduino IDE suunniteltu helppokäyttöiseksi matalan kynnyksen ohjelmointiympäristöksi, joten se on suosittu harrastajien keskuudessa.



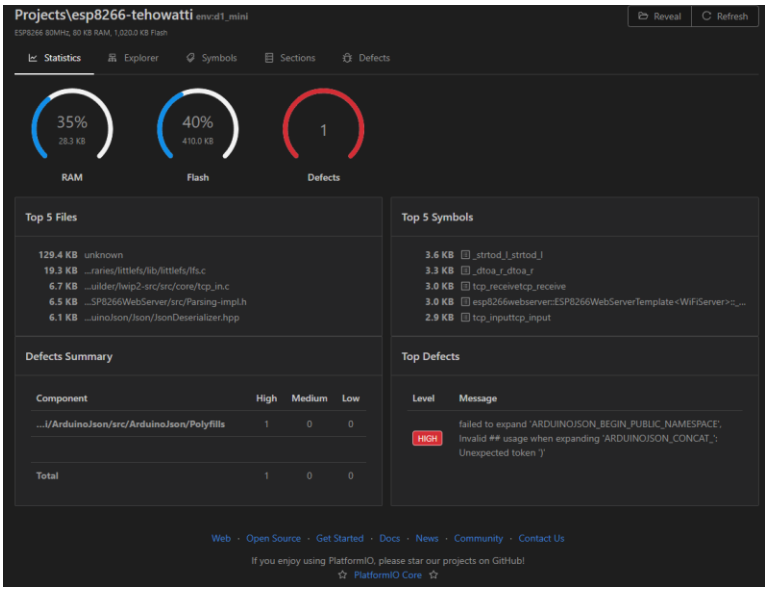

Kuva 22. Arduino IDE

PlatformIO-lisäosa puolestaan tarjoaa ohjelmointiin tottuneelle enemmän työkaluja ja toiminnallisuuksia, ja mahdollistaa kaikkien Visual Studio Code:n ominaisuuksien hyödyntämisen myös mikrokontrolleriohjelmoinnissa. PlatformIO tuo mikrokontrolleriohjelmointiin muun muassa edistyneen kirjastojen hallinnan, versionhallintatyökalut, yksikkötestit ja edistyneet virheen etsintätyökalut.




Kuva 23. Visual Studio Code PlatformIO -lisäosa

Visual Studio Codeen saatavilla olevien yleisten ohjelmointia helpottavien laajennusosien lisäksi PlatformIO tarjoaa työkaluja nimenomaan mikrokontrollerikehityksen helpottamiseksi. Esimerkkinä hyödyllisestä työkalusta on Inspect-toiminto, joka vertaa koodin vaatimia resursseja valitun mikrokontrollerin resursseihin.



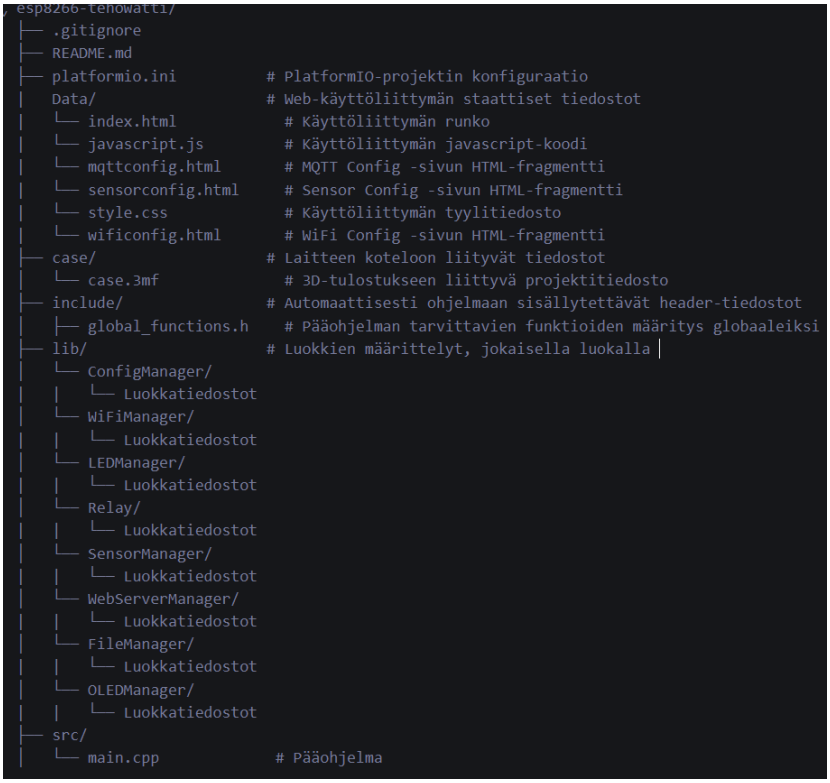

Kuva 24. PlatformIO Inspect-toiminto

Arduino IDE ja PlatformIO eroavat merkittävästi projektirakenteensa ja hallintotapojensa suhteen. Vaikka ohjelmat ovat siirrettävissä Arduino IDE:stä PlatformIO:n ja päinvastoin, tämä vaatii raken-

teellisia muutoksia ja mahdollisesti konfiguraatioiden uudelleenmäärittelyä. Kokonaisuutena PlatformIO tarjoaa laajemmat mahdollisuudet, mutta Arduino IDE:n yksinkertaisuus tekee siitä edelleen suositun aloittelijoille ja pieniin projekteihin.

Arduino IDE:n projektirakenne on hyvin yksinkertainen, ja koodi sijoitetaan yleensä .INO-tiedostoon, josta löytyvät pakolliset setup() ja loop() -funktiot. Kirjastot lisätään joko manuaalisesti tai kirjastonhallinnan kautta. Arduino IDE ei sisällä erillistä projektikonfiguraatiotiedostoa, vaan asetukset määritetään käyttöliittymästä. PlatformIO jakaa projektin useampiin kansioihin hallittavuuden parantamiseksi ja laajojen projektien selkeyttämiseksi.

Kuvassa 25 on esitetty opinnäytetyön PlatformIO-projektin rakenne ja ohjelmaan liittyvät tiedostot.

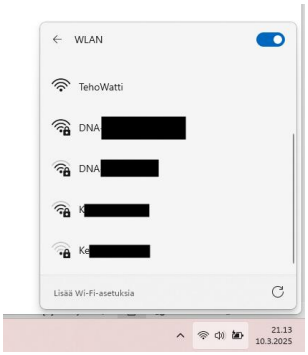



Kuva 25. PlatformIO-projektin rakenne

## 8.3 Ohjelman toiminnallinen kuvaus

Ohjelman pääasiallinen tarkoitus on mitata ja välittää sähkökattilan tulo- ja menoveden lämpötiloja sekä ohjata kattilan lämpötilan pudotustoimintoa manuaalisesti tai automatisoidusti. Edellä mainitut toiminnot ovat ohjattavissa MQTT-viestein, HTTP-pyynnöin sekä selaimella toimivan käyttöliittymän kautta. Ohjelman toiminta voidaan jakaa kahteen osa-alueeseen; käyttöönotto ja jatkuva toiminta. Käyttöönotto on suoritettava, kun ohjelma ladataan ensimmäisen kerran tietokoneelta laitteeseen. Ohjelman tekninen rakenne ja yksityiskohdat on kuvattu tarkemmin seuraavissa kappaleissa.

Käyttöönoton yhteydessä laitteessa ei ole WiFi- ja MQTT-yhteyksien muodostamiseen tai sensorien lukemiseen tarvittavia parametreja, vaan käyttäjän on määritettävä ne käyttöliittymän kautta ensimmäisellä käyttökerralla. Ilman konfigurointitietoja laite käynnistyy tukiasematilassa, eli laite perustaa itsenäisen langattoman lähiverkon johon käyttäjä voi yhdistää esimerkiksi mobiililaitteen tai tietokoneen.

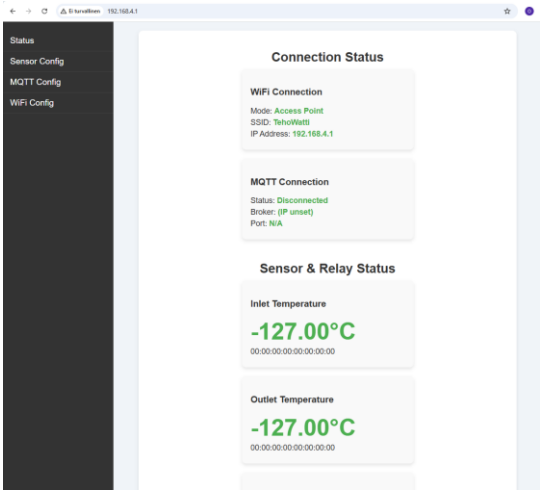



Kuva 26. Laitteen muodostama WiFi-tukiasema

Laitteen käynnistyessä tukiasematilaan ilman konfigurointeja, LED-indikaattori on purppuran värinen AP-tilan merkiksi ja näytössä näytetään kaikki laitteen tilat. Käyttöönottovaiheessa laitteesta puuttuu kytkettyjen lämpötilasensoreiden osoitteet sekä yhteyskonfiguraatiot, joten MQTT- ja WiFi-yhteyksiä ei voida muodostaa eikä lämpötilojen oloarvoja lukea. Lämpötilatietojen puuttuessa tai ollessa virheellisiä näytetään -127 °C oletuslämpötila. Kun laite on AP-tilassa ja web-palvelin on käynnistetty, käyttäjä voi siirtyä tukiasemaan yhdistetyllä laitteella näytössä näkyvään IP-osoitteeseen konfigurointikäyttöliittymän avaamiseksi.

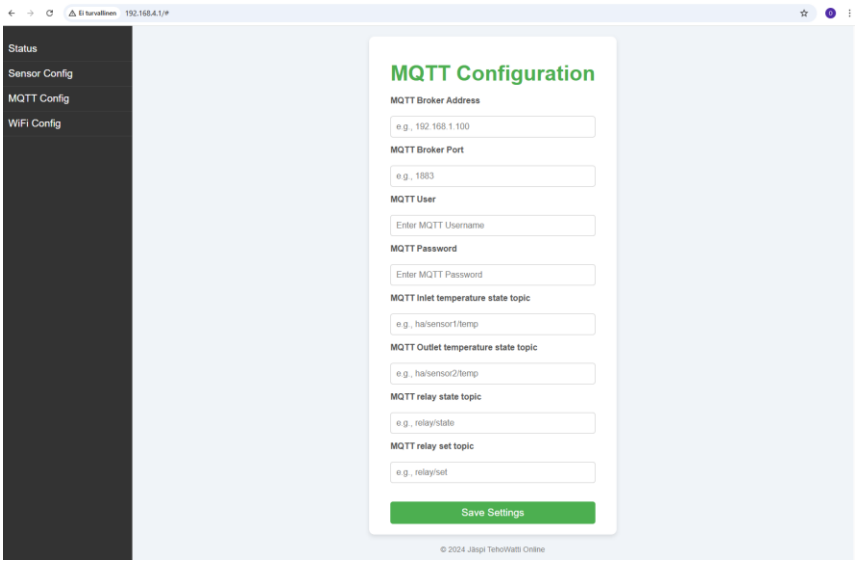

Kuva 27. Laitteen tilaindikaattorit ennen konfigurointia

Selaimella avautuva, kuvan 21 konfiguraatio- ja status-sivusto sisältää laitteen konfiguroimattomat tilatiedot sekä valikon, jonka kautta kukin osa-alue päästään asettelemaan erikseen.



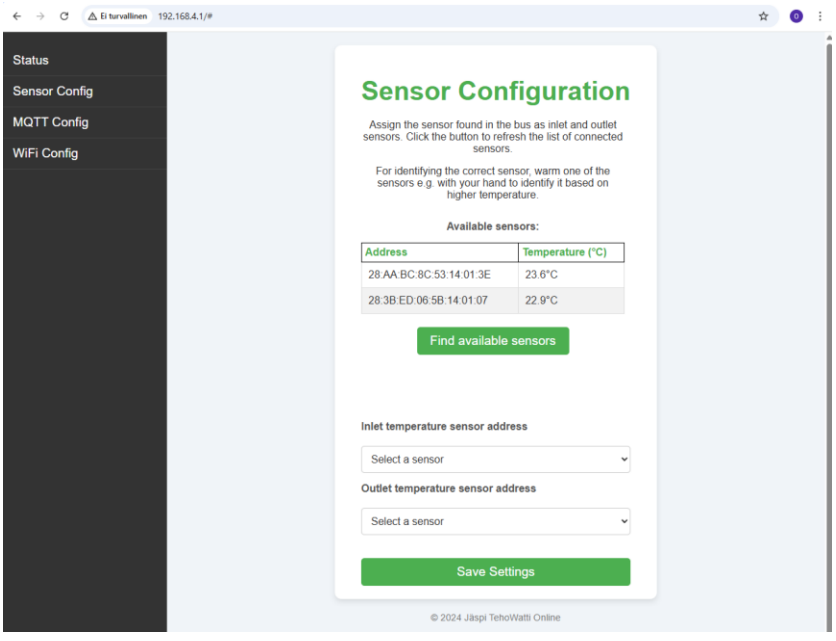

Kuva 28. Laitteen web-käyttöliittymä

Käyttäjä voi konfiguroida sensorien väyläosoitteet ja yhteyksiä varten tarvittavat parametrit vasemman laidan valikoiden kautta. WiFi- ja MQTT-konfiguraatiosivut sisältävät lomakkeet, joiden kautta syötetyt parametrit tallennetaan laitteen pysyväismuistiin. Kun MQTT- ja WiFi -yhteysparametrit on tallennettu ja laite käynnistetty uudelleen, laite yhdistää käyttäjän määrittämään langattomaan verkkoon ja on valmis MQTT-viestintää varten.




Kuva 29. MQTT-konfiguraatiolomake

Tulo- ja menoveden lämpötilaa mittaavien sensoreiden konfiguroimiseksi käyttäjän on navigoitava ”Sensor Config” -sivulle. Sivulla olevaa ”Find available devices” -painiketta painamalla laitteeseen kytketyt sensorit skannataan, ja niiden väyläosoitteet sekä lämpötilan oloarvot palautetaan käyttöliittymään. Käyttäjä voi valita alasvetovalikoista tulo- ja menovesiputkiin kytketyn sensorin lämpötilan perusteella.

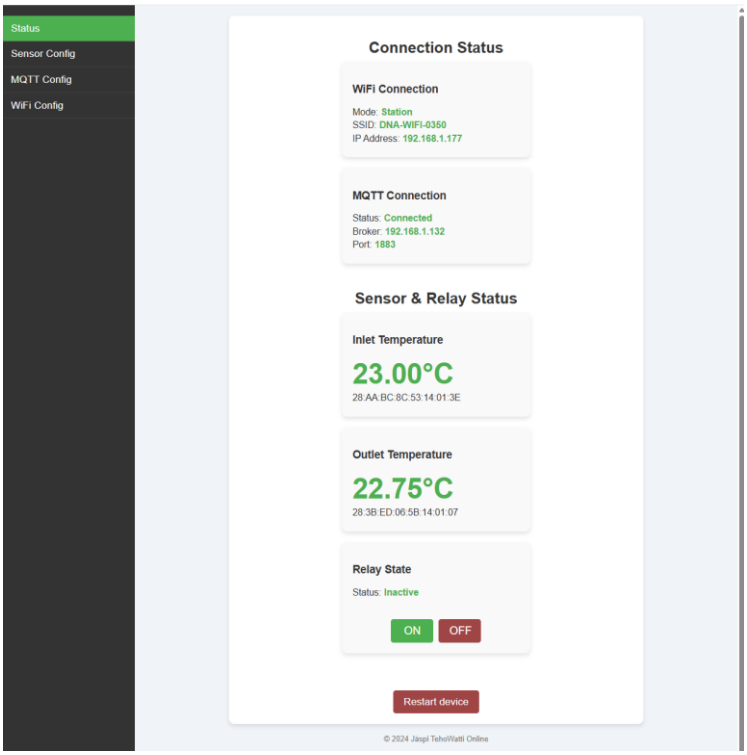



Kuva 30. Sensoriosoitteiden skannaus ja asettelu

Kun tarvittavat asetukset on tallennettu, käyttäjälle lähetetään selaimen kautta kehote laitteen uudelleenkäynnistämiseksi. Laite voidaan käynnistää uudelleen status-sivulta löytyvällä ”Restart device” -painikkeella. Laite käynnistyy uudelleen jatkuvan toiminnan tilaan. Mikäli WiFi- ja MQTT-yhteyksien muodostaminen onnistuu, laitteen LED-indikaattorin väri muuttuu vihreäksi ja näytössä esitetään jälleen yhteyksien tila sekä lämpötilojen oloarvot. Käyttäjä voi nyt yhdistää laitteen web-käyttöliittymään kuvassa näkyvän IP-osoitteen avulla millä tahansa samaan WiFi-verkkoon kytketyllä laitteella.


Kuva 31. Fyysiset indikaattorit valmiustilassa

Valmiustilassa käyttäjä pääsee muokkaamaan olemassa olevia konfiguraatioita, tarkastelemaan laitteen tilaa sekä ohjaamaan relelähtöä käyttöliittymän kautta. Kuvassa 32 on esitetty käyttöliittymän status-sivu kun tarvittavat parametrit on aseteltu. Status-sivulla esitetään sekä WiFi- että MQTT-yhteyksien tilojen lisäksi tulo- ja menovettä mittaavien sensoreiden oloarvot ja niiden väyläosoitteet. Käyttäjä voi myös pakko-ohjata laitteen relettä, eli sähkökattilan lämpötilan pudotusta ”ON”- ja ”OFF” -painikkeiden avulla tai käynnistää laitteen uudelleen ”Restart device” -painikkeella. Status-sivulla näkyvät tilat päivitetään viiden sekunnin välein, mikäli käyttäjä ei poistu näkymästä tai sulje selainta.

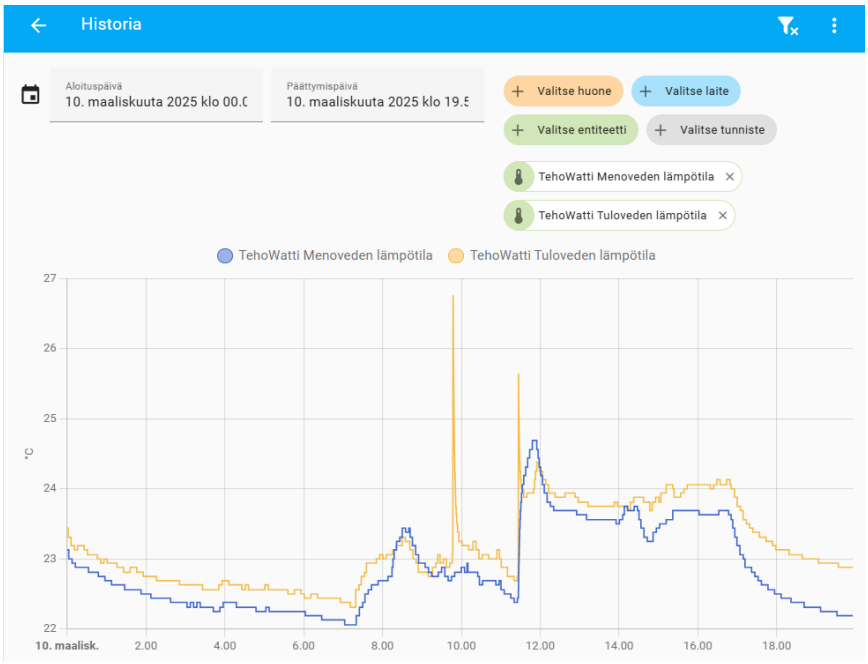



Kuva 32. Käyttöliittymän status-sivu jatkuvan toiminnan tilassa

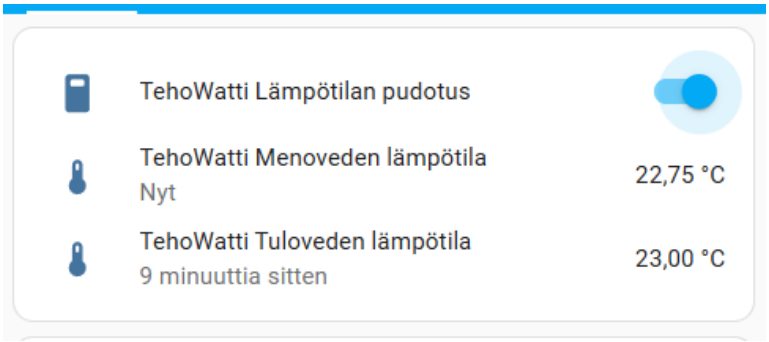
Käyttöliittymän lisäksi laite kommunikoi jatkuvasti MQTT-palvelimen kanssa lämpötilatietoja lähettäen ja releen ohjauskomentoja vastaanottaen. Mahdolliset taustalla tapahtuvat releen tilamuutokset päivitetään myös käyttöliittymään automaattisesti. Laite lähettää lämpötilojen oloarvot konfiguroituihin MQTT-otsikoihin aina, kun lämpötila muuttuu yli 0,1 °C ja edellisestä lähetyksestä on kulunut vähintään 30 sekuntia, tai arvot lähetetään aina kun viimeisimmästä lähetyksestä on kulunut 30 minuuttia.

Mikäli laite vastaanottaa MQTT- tai HTTP-komennon releen ohjaamiseksi, releen fyysinen tila muuttuu, tila päivitetään käyttöliittymään sekä laitteen fyysisiin indikaattoreihin. Kun relelähtö on aktiivinen, laitteen LED-indikaattori muuttuu siniseksi. Kun rele deaktivoidaan, palaa LED-indikaattori edelliseen tilaansa.

Kuvissa 33 ja 34 on esitetty Home Assistant -kotiautomaatiojärjestelmään konfiguroidut alustavat näkymät sensoriarvojen seuraamiselle ja kytkin kattilan lämpötilan pudotuksen aktivoimiseksi. Lämpötilan pudotustoiminto ja sensoriarvot ovat nyt käytettävissä erilaisiin automaatioihin.




Kuva 33. Tulo- ja menoveden lämpötilojen historiatiedot




Kuva 34. Laitteen tilatiedot ja ohjauspainike automaatiojärjestelmässä

Jatkuvan toiminnan tilassa laite tarkkailee sekä WiFi- että MQTT-yhteyden tilaa. Mikäli yhteys MQTT-palvelimeen katkeaa, laite koettaa muodostaa palvelinyhteyden uudelleen 10 sekunnin välein. Mikäli WiFi-yhteys menetetään, laite aloittaa yhteydenmuodostussyklin – AP-tila käynnistetään minuutiksi, jonka jälkeen yhteyttä koetetaan muodostaa tallennetuin asetuksin ja mikäli yhteyden muodostus jälleen epäonnistuu, palataan takaisin AP-tilaan. AP-tilan ja yhteyskokeilun välillä vuorottelu mahdollistaa tilapäisistä verkonmenetyksistä toipumisen automaattisesti, mutta toisaalta myös antaa mahdollisuuden siirtyä konfigurointitilaan esimerkiksi WiFi-asetusten muuttuessa yllättäen. Mikäli yksikin asiakas on yhdistänyt laitteen verkkoon AP-tilassa, vuorottelu pysähtyy, kunnes asiakasyhteydet on katkaistu.


Kuva 35. MQTT-yhteys katkennut

Mikäli MQTT- tai WiFi-yhteys menetetään, laite menee virhetilaan ja LED-indikaattorin väri muuttuu punaiseksi ja yhteyksien uudelleenmuodostussykli käynnistyy.

## 8.4 Ohjelman arkkitehtuuri

### 8.4.1 Modulaarisuus ja kapselointi

Ohjelman rakenne on suunniteltu modulaariseksi, jotta se olisi helposti laajennettavissa ja ylläpidettävissä. Koodi on jaettu osakokonaisuuksiin kapseloimalla toiminnallisuuksia erillisiin luokkiin, mikä parantaa pääohjelman luettavuutta ja selkeyttää kokonaisuuden hallintaa.

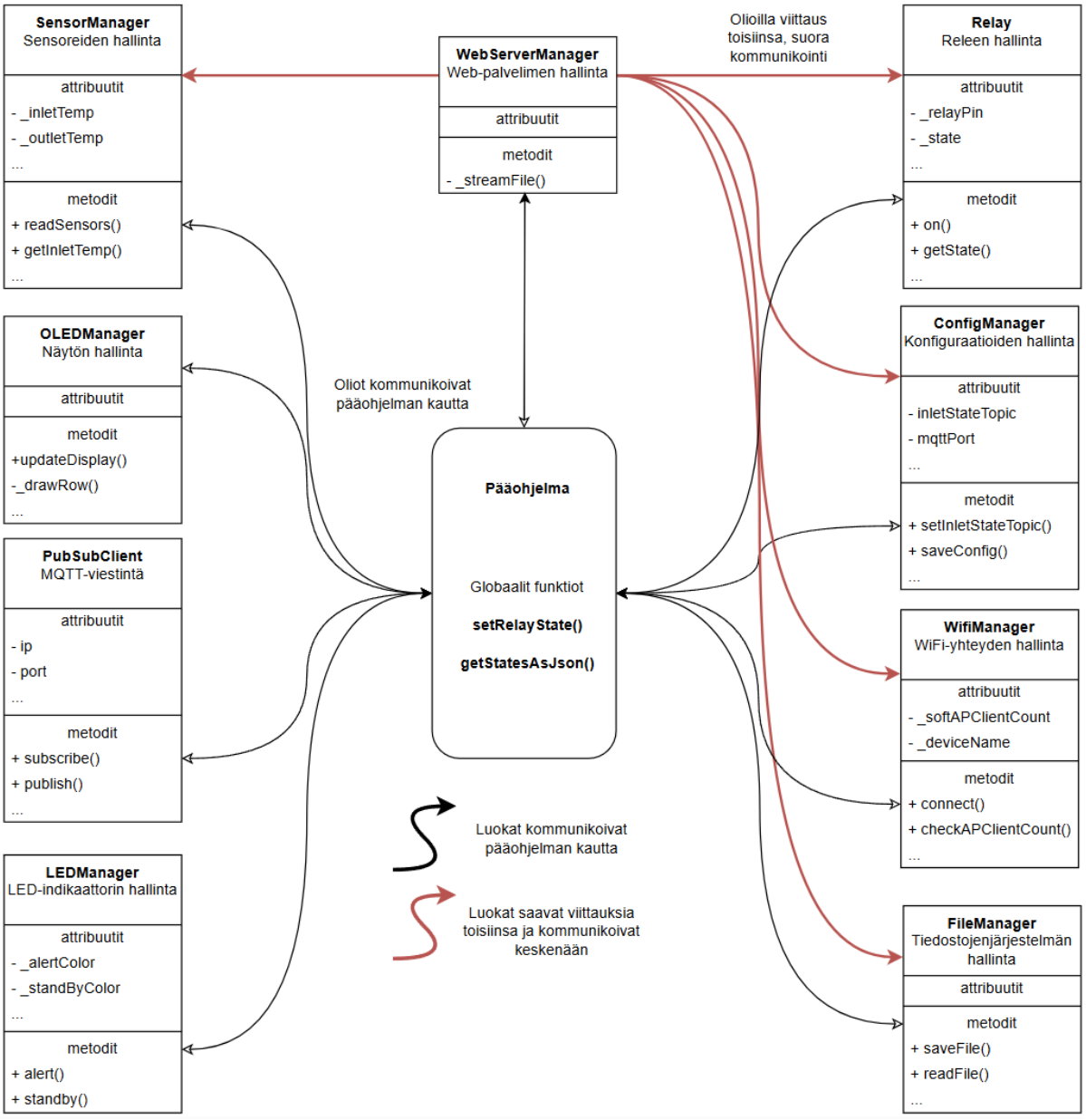
Ohjelmassa hyödynnetään itse kirjoitettuja sekä yleisesti tunnettuja ja käytettyjä Arduino- ja ESP8266-luokkia, joita on tarvittaessa laajennettu periyttämällä. Näin on voitu lisätä tarvittavia jäsenmuuttujia ja metodeja olemassa olevia toiminnallisuuksia säilyttäen. Ohjelman pääkomponentit ja niiden toiminnallisuudet on esitetty tarkemmin opinnäytetyön myöhemmissä kappaleissa.

```
#ifndef WiFiManager_h
#define WiFiManager_h

#include <ESP8266WiFi.h>

// Inherit the default Arduino WiFiClass and create custom methods to streamline operation
class WiFiManager : public ESP8266WiFiClass {

public:
    WiFiManager();
    bool connect();
    void startSoftAP();
    void checkWiFiStatus();
    void checkAPClientCount(); // Checks the softAP clientcount
    bool isAPOn(); // Helper method to return boolean value if softAP is active
    void setCredentials(const char* ssid, const char* password, const char* deviceName);
    const char* getIP(); // Returns ip address regardless of operating mode
    const char* getMode(); // Returns STA/AP mode

private:
    const char* _ssid;
    const char* _password;
    const char* _deviceName;
    int _softAPclientCount = 0;                                  // nbr of softAP clients
    unsigned long _softAPclientCountLastCheckMillis = millis();  // when clientcount was last time checked
    unsigned long _softAPStartMillis = 0;
};

#endif
```

Kuva 36. WiFiManager-luokka laajentaa ESP8266WiFiClass luokkaa

### 8.4.2 Luokkien välinen kommunikointi

Jokaista laitteen toiminnallisuutta tai komponenttia ohjaa omaan luokkaansa perustuva olio. Luokat kommunikoivat joko pääohjelman kanssa, toistensa kanssa suoria olioviittauksia hyödyntäen tai pääohjelman globaaleja funktioita käyttämällä.




Kuva 37. Luokkien välinen kommunikaatio, periaatekuva

Kuvan 37 yksinkertaistetussa esityksessä mustat yhdistinviivat kuvastavat pääohjelman kanssa kommunikointia. Kaikki oliot alustetaan pääohjelmassa, joten pääohjelma voi ohjata laitteen kaikkia toimintoja. Tämän lisäksi pääohjelmaan on kirjoitettu kaksi globaalia funktiota, joita kutsumalla mikä tahansa olio voi suorittaa sarjan toimintoja, joissa tarvitaan lähes kaikkien olioiden metodeja. Esimerkkinä tästä on pääohjelman getStatesAsJson()-funktio, joka kerää kaikkien olioiden tilatiedot ja palauttaa ne kutsujalle JSON-muodossa. Funktiota käytetään ensisijaisesti tilatietojen keräämiseksi käyttöliittymälle ilman, että jokaista oliota on välitettävä viittauksena WebServerManager-luokalle.

```
String getStatesAsJson() {
  JsonDocument doc;

  String currentSSID;

  if (wm.getMode() == "STA") {
    currentSSID = wm.SSID();
  } else {
    currentSSID = wm.softAPSSID();
  }

  // Populate JSON document with current config values
  doc["wifiMode"] = wm.getMode();
  doc["wifiIP"] = wm.getIP();
  doc["SSID"] = currentSSID;
  doc["MQTTConnected"] = mqttClient.connected();
  doc["MQTTBroker"] = config.getMqttServer().toString();
  doc["MQTTPort"] = config.getMqttPort();
  doc["InletTemp"] = sensors.getInletTemp();
  doc["OutletTemp"] = sensors.getOutletTemp();
  doc["RelayState"] = relay.getState();
  doc["inletSensorAddress"] = sensors.getInletSensorAddress();
  doc["outletSensorAddress"] = sensors.getOutletSensorAddress();

  // Serialize JSON document into a string
  String serializedStates;
  serializeJson(doc, serializedStates);

  return serializedStates;
}
```

Kuva 38. Globaali getStatesAsJson() -funktio

Kaavion punaiset yhdistinviivat kuvastavat olioiden sisältämiä viittauksia toisiinsa, jolloin oliot voivat kutsua toistensa metodeja suoraan. Tosiasiallisesti viittauksia on merkittyä enemmän, mutta erityisen tärkeitä ne ovat WebServerManager-olion yhteydessä. WebServerManager-luokkaan on kapseloitu HTTP-kutsujen käsittelyyn tarvittava logiikka mukaan lukien konfiguraatioiden tallennukseen, lämpötilatietojen palauttamiseen ja releen ohjaukseen liittyen, joten luokalle oli luonnollista viedä suora viittaus ko. toiminnallisuuksista vastaaviin olioihin ja metodikutsuihin. Esimerkiksi, kun käyttäjä tallentaa parametrit käyttöliittymän kautta, WebServerManager päivittää parametrit suoraan ConfigManager-olion jäsenmuuttujiin ja kutsuu saveConfig()-metodia. Näin pääohjelman luettavuutta ja nimiavaruuden hallittavuutta ei tarvinnut heikentää käyttämällä liiallisesti globaaleja funktioita.

WebServerManager -luokka vastaa siis pääasiallisesti käyttöliittymässä tarvittavista toiminnallisuuksista suorien olioviittausten avulla, mutta laitteen muut ydintoiminnallisuudet kuten MQTT-viestintä, fyysisten indikaattorien päivitys ja verkkoyhteyksien hallinta on toteutettu pääohjelmassa tarvittavien luokkien kanssa kommunikoimalla.

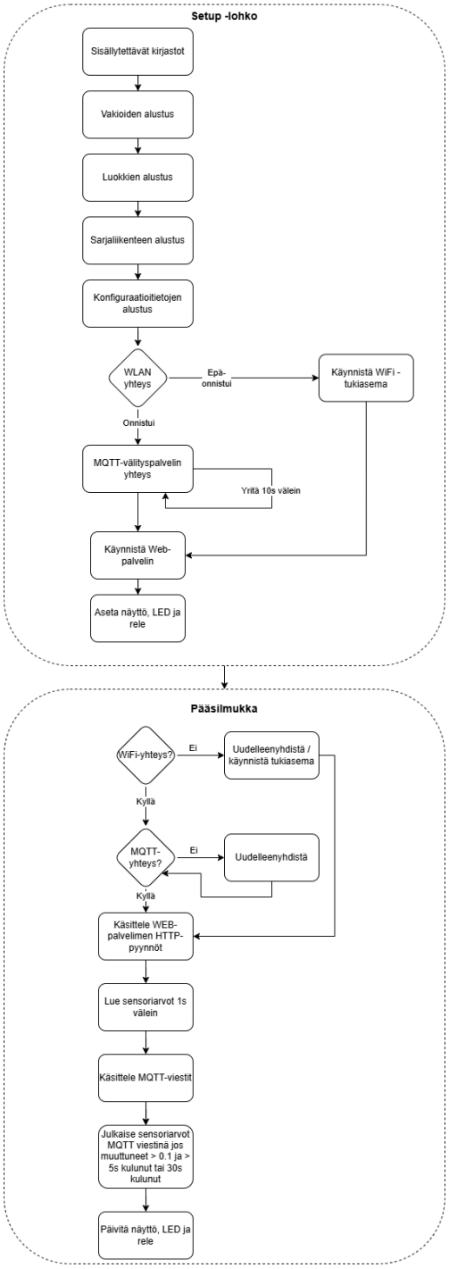
## 8.5 Ohjelman suorituskaavio

Kuvassa 39 on yksinkertaistettu esitys ohjelman suorituskaaviosta. Ohjelma noudattaa Arduino-ympäristöstä tuttua rakennetta, jossa ensin suoritetaan setup-lohko muuttujien ja olioiden alustamiseksi. Setup -lohko suoritetaan vain kerran järjestelmän käynnistyessä.

Setup-vaiheen jälkeen ohjelma siirtyy pääsilmukkaan, jota mikrokontrolleri suorittaa jatkuvasti, kunnes järjestelmä sammutetaan tai sen suorittaminen keskeytyy hallitsemattoman virheen vuoksi. ESP8266-mikrokontrolleri ei tue säikeistystä tai rinnakkaista suoritusta, joten kaikki pääsilmukassa ajettavat toiminnot suoritetaan peräkkäin. Tämä asettaa haasteita ohjelman rakenteelle, erityisesti silloin, kun jokin toiminto vaatii odottamista ennen valmistumistaan.

Vaikka mikrokontrolleri itse kykenee nopeaan suoritukseen, järjestelmään liitetyt komponentit, kuten sensorit ja näyttölaitteet, toimivat hitaammin. Tästä syystä niiden kanssa kommunikointia ei voida suorittaa joka silmukan kierroksella ilman, että ohjelman suoritus hidastuisi merkittävästi.

Ohjelman ei-toivottujen viiveiden välttämiseksi eri toiminnoille on toteutettu non-blocking (ei-suoritusta keskeyttäviä) ratkaisuja. Esimerkiksi sensoridatan lukeminen ja OLED-näytön päivitys suoritetaan vain tietyin aikavälein tai muuttujan arvon muuttuessa, eikä jokaisella silmukan kierroksella. Ajastuksia ei hallita perinteisellä delay()-metodilla, koska se pysäyttää ohjelman suorituksen määräajaksi. Sen sijaan kunkin komponentin edellinen päivitysaika tai tila tallennetaan, ja päivitys suoritaan vain, kun määritelty aikaväli on kulunut tai tila muuttunut. Tämä mahdollistaa järjestelmän tehokkaan toiminnan ilman tarpeettomia viiveitä ja estää muiden kriittisten toimintojen, kuten verkkoviestinnän, estymisen.




Kuva 39. Ohjelman yksinkertaistettu suorituskaavio

Esimerkkinä ei-suoritusta keskeyttävästä toiminnosta on kuvassa 40 esitetty MQTT-yhteyden muodostava funktio. Pääsilmukassa MQTT-yhteyden olemassaolo tarkastetaan jokaisella suorituskerralla, ja yhteyden puuttuessa uudelleenyhdistämistä yritetään MQTT_RECONNECT_INTERVAL-vakiossa määritetyn viiveen välein (10s). Jos yhteyttä yritettäisiin muodostaa jokaisella silmukan kierroksella, se kuluttaisi lähes kaiken suoritusajan ja heikentäisi web-käyttöliittymän responsiivisuutta.

```
/*
Connect to mqtt broker
*/
void connectMqtt() {
  static unsigned long lastConnectionAttempt = 0;

  if(wm.status() == WL_CONNECTED && millis() - lastConnectionAttempt > MQTT_RECONNECT_INTERVAL) {

    Serial.println("Trying to establish mqtt broker connection");

    if(mqttClient.connect(DEVICE_NAME, config.getMqttUser(), config.getMqttPassword())) {
      Serial.println("Connected to mqtt broker succesfully");
    } else {
      Serial.print("Failed to connect mqtt broker with errorcode: ");
      Serial.println(mqttClient.state());
    }

    lastConnectionAttempt = millis();
  }
}
```

Kuva 40. MQTT-yhteyden muodostava funktio

## 8.6 Selainkäyttöliittymä

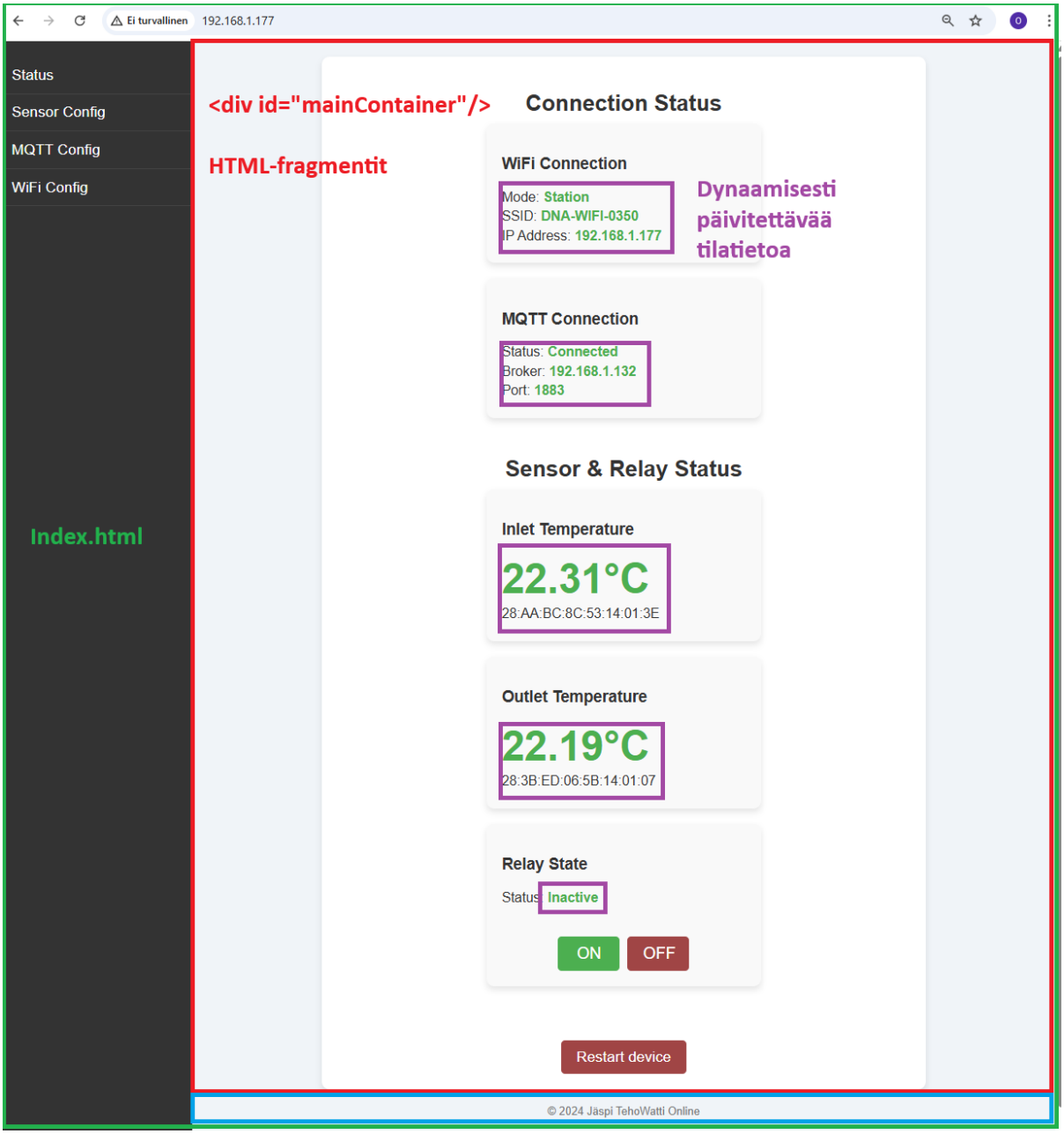
### 8.6.1 Yleistä

Laitteen selainkäyttöliittymästä vastaa WebServerManager-luokka, joka käynnistää laitteessa web-palvelimen, tarjoilee tarvittavat laitteen tiedostojärjestelmään tallennetut HTML-, JavaScript- ja CSS-tiedostot sekä sisältää käyttöliittymän toimintoihin liittyvän logiikan. Käyttöliittymän ohjelmoinnissa rajoittavina tekijöinä oli laitteen rajallinen suorituskyky, pieni tallennustila staattisille tiedostoille ja vaatimus, jonka mukaan käyttöliittymän on toimittava myös AP-tilassa ilman ulkoista internet-yhteyttä. Tämän vuoksi käyttöliittymän koodissa ei käytetä lainkaan tyyli- tai JavaScript-kirjastoja, jottei ylimääräisiä tiedostoja tarvitse ladata tai noutaa verkosta.

Laitteen web-palvelin sisältää useita reittejä, joiden avulla selain kommunikoi laitteen kanssa. Palvelimella on tarvittavat reitit tilatietojen ja staattisen sisällön palauttamiseksi sekä konfiguraatiotietojen tallentamiseksi ja laitteen toimintojen ohjaamiseksi. Aloitussivun lataamisen jälkeen navigointi perustuu dynaamiseen HTML-fragmenttien lataamiseen ja yksittäisten elementtien päivittämiseen JSON-objektien avulla.

### 8.6.2 Käyttöliittymän rakenne

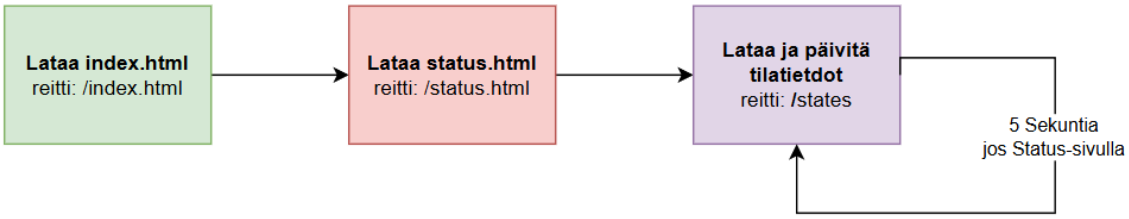
Käyttöliittymän runko on kirjoitettu laitteen pysyväismuistiin tallennettuun index.html -tiedostoon, jossa on tarvittavat navigointilinkit konfiguraatio- ja statussivuille sekä tilavaraus dynaamisesti ladattavalle sisällölle. Navigoimalla laitteen IP-osoitteeseen selainta käyttäen ensin ladataan index.html -tiedosto, minkä jälkeen selain lähettää erillisen HTTP-kutsun sisällön noutamiseksi.




Kuva 41. Käyttöliittymän rakenne

Kuvaan 41 on merkitty käyttöliittymän tärkeimmät elementit. Sivuston avaamisen konfiguraatiosivulta toiselle navigoitaessa ainoastaan ”mainContainer” -säilöä päivitetään noutamalla tarvittava pysyväismuistiin tallennettu HTML-fragmentti ja asettamalla sen sisältö näkyviin. Jokaiselle konfiguraatiosivulle on erillinen HTML-tiedosto tilavarauksen täydentämiseksi. Jokaisen HTML-fragmentin tarvittavat elementit on merkitty id-tunnuksin, jotta yksittäisten elementtien tietoja voidaan päivittää tai käsitellä JavaScript-koodissa. Kuvaan violetilla merkityt alueet ovat dynaamisesti päivitettävää tilatietoa fragmentin lataamisen jälkeen.

Kuvassa 42 on esitetty esimerkkinä selaimen lähettämät HTTP-pyynnöt kun käyttöliittymän status-sivu avataan. Status-sivu on käyttöliittymän oletusnäkymä. Selain lähettää HTTP-kutsun IP-osoitteeseen ja saa vastauksena index.html -tiedoston sisällön. Tässä vaiheessa selain on vastaanottanut ainoastaan valikkorakenteen, tyylit ja JavaScript-koodin. Kun DOM on kokonaan latautunut, JavaScript lähettää asynkronisen HTTP-kutsun status-fragmentin lataamiseksi ja näyttämiseksi tilavarauksen sisällä. Kun tarvittava fragmentti on noudettu, selain lähettää vielä kolmannen HTTP-kutsun reittiin, joka palauttaa tarvittavat tilatiedot JSON-objektina ja täydentää status-fragmentin elementit. Samalla käynnistyy JavaScript-ajastin, joka päivittää tilatiedot 5 sekunnin välein, kunnes käyttäjä sulkee selaimen tai navigoi toiselle sivulle. Kaavion värit ovat verrannollisia edellisen, kuvan 41 kuvan värimerkintöihin.




Kuva 42. HTTP-pyynnöt käyttöliittymää ladattaessa

Staattisia tiedostoja palautettaessa oli otettava huomioon mikrokontrolleriohjelmoinnin erityispiirteet. JavaScript- ja CSS-tiedostoja varten oli määritettävä erilliset reitit, sillä tiedostojärjestelmä ei sallinut suoraa viittausta index.html -tiedostosta samassa hakemistossa oleviin muihin tiedostoihin. Tämän lisäksi tiedostot oli lähetettävä sisältövirtana (stream) – tällöin tiedoston sisältöä ei tarvitse ladata ensin kokonaan laitteen RAM-muistiin, vaan se lähetetään jatkuvana virtana tiedostoa luettaessa resurssien riittävyyden takaamiseksi.

```
/*
Serve the root route html from SPIFFS memory
*/
on("/", HTTP_GET, [this]() {
    _streamFile("/index.html");
});

/*
Serve the javascript file as stream from filesystem
*/
on("/javascript", HTTP_GET, [this]() {
    _streamFile("/javascript.js");
});

/*
Serve the stylesheet
*/
on("/stylesheet", HTTP_GET, [this]() {
    _streamFile("/style.css", "text/css");
});

/*
Serve the HTML fragments for maincontainer in index.html
*/
on("/status", HTTP_GET, [this]() {
_streamFile("/status.html");
});
```

Kuva 43. Staattisia tiedostoja palauttavat reitit

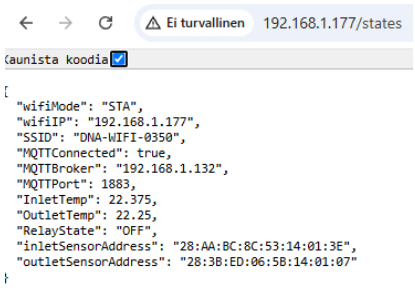
Kuvassa 44 on esitetty ohjelman WebServerManager-luokkaan kirjoitettu metodi, jonka avulla reitit palauttavat tiedostoja sisältövirtana. Virhetilanteessa metodi palauttaa selaimelle HTTP-koodin 404 sekä viestin tiedostovirheestä. Metodi lähettää myös parametrina vastaanotettavan ”Content-Type” -otsikkotiedon kulloinkin palautettavan tiedostotyypin mukaan, jotta selain osaa käsitellä tiedoston oikein.

```
/**
 * @brief Private method to stream files
 *
 * Streams the static files in chunks to avoid loading them into RAM and running out of memory.
 * Sends "text/plain" as default content type unless defined in parameter
 *
 * @param path The filepath to open and stream in SPIFFS memory e.g. /index.html
 * @param type The Content-Type of response, e.g. "text/plain" or "application/json"
 */
void WebServerManager::_streamFile(const char* path, const char* type) {
    File file = LittleFS.open(path, "r");
        if (!file) {
            send(404, "text/plain", "File not found");
            return;
        }

        // Set the Content-Type header
        sendHeader("Content-Type", type);

        size_t sent = streamFile(file, type);
    file.close();
};
```

Kuva 44. WebServerManager-luokan _streamFile() -metodi

### 8.6.3 Status-sivu

Kaikkien käyttöliittymäsivujen ja -elementtien lataus tapahtuu pääasiallisesti edellisessä kappaleessa kuvatun perusperiaatteen mukaisesti. Edellisessä kappaleessa kuvatulla Status-sivulla näytettävät tilatiedot palautetaan palvelimen ”/states” -reitin kautta JSON-objektina ja arvot asetetaan HTML-elementteihin.




Kuva 45. /states -reitin palauttama JSON-objekti

Kuvassa 46 on vastaavasti esitetty JavaScript -funktio, joka vastaa tilatietojen täydentämisestä status-sivun elementteihin. Tilatiedon puuttuessa tai ollessa alustamaton arvoksi asetetaan ”N/A”. Tilatiedot täydentävää funktiota kutsutaan viiden sekunnin välein, kunnes käyttäjä poistuu status-sivulta, näin näytettävät tiedot pysyvät ajantasaisina jatkuvasti. Kuvassa 47 on esitetty ESP8266-laitteen pääohjelmassa oleva tilatiedot keräävä ja palauttava globaali funktio.

```
async function populateStates() {
  // Load all the states with http request to global variable
  statesJson = await loadData(fullUrl+'states');

  // Set wifi states
  const wifiStates = {
    "STA": "Station",
    "AP": "Access Point"
  };

  document.getElementById("wifimode").innerHTML = wifiStates[statesJson.wifiMode] || "N/A";
  document.getElementById("ssid").innerHTML = statesJson.SSID || "N/A";
  document.getElementById("wifiIP").innerHTML = statesJson.wifiIP || "N/A";

  const mqttStates = {
    true: "Connected",
    false: "Disconnected"
  };

  // Set mqtt states
  document.getElementById("mqttstatus").innerHTML = mqttStates[statesJson.MQTTConnected] || "N/A";
  document.getElementById("mqttIP").innerHTML = statesJson.MQTTBroker || "N/A";
  document.getElementById("mqttPort").innerHTML = statesJson.MQTTPort || "N/A";

  // Set sensor states
  document.getElementById("inlet").innerHTML = parseFloat(statesJson.InletTemp).toFixed(2) + "°C" || "N/A";
  document.getElementById("inletSensorAddress").innerHTML = statesJson.inletSensorAddress || "N/A"
  document.getElementById("outlet").innerHTML = parseFloat(statesJson.OutletTemp).toFixed(2) + "°C" || "N/A";
  document.getElementById("outletSensorAddress").innerHTML = statesJson.outletSensorAddress || "N/A"

  // Set relay states
  const relayStates = {
    "ON": "Active",
    "OFF": "Inactive"
  };

  document.getElementById("relaystate").innerHTML = relayStates[statesJson.RelayState] || "N/A";
}
```

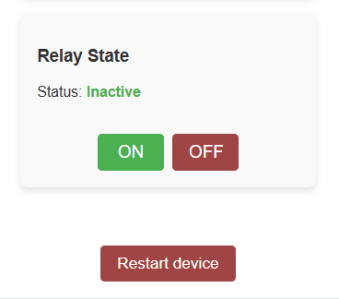
Kuva 46. Selaimessa tilatiedot täydentävä JavaScript-funktio

```
String getStatesAsJson() {
  JsonDocument doc;

  String currentSSID;

  if (wm.getMode() == "STA") {
    currentSSID = wm.SSID();
  } else {
    currentSSID = wm.softAPSSID();
  }

  // Populate JSON document with current config values
  doc["wifiMode"] = wm.getMode();
  doc["wifiIP"] = wm.getIP();
  doc["SSID"] = currentSSID;
  doc["MQTTConnected"] = mqttClient.connected();
  doc["MQTTBroker"] = config.getMqttServer().toString();
  doc["MQTTPort"] = config.getMqttPort();
  doc["InletTemp"] = sensors.getInletTemp();
  doc["OutletTemp"] = sensors.getOutletTemp();
  doc["RelayState"] = relay.getState();
  doc["inletSensorAddress"] = sensors.getInletSensorAddress();
  doc["outletSensorAddress"] = sensors.getOutletSensorAddress();

  // Serialize JSON document into a string
  String serializedStates;
  serializeJson(doc, serializedStates);

  return serializedStates;
}
```

Kuva 47. IoT-laitteen tilatiedot palauttava globaali funktio pääohjelmassa

Status-sivulla on myös laitteen ohjaamiseen liittyviä painikkeita. Releen ohjaamiseen liittyvät painikkeet kommunikoivat laitteen kanssa ”/relay” -reitin avulla GET-metodia käyttäen. Kun kyseiseen reittiin lähetetään GET-pyyntö ilman parametreja, laite palauttaa releen tämänhetkisen tilan (ON/OFF). GET-pyynnön yhteydessä laitteelle voidaan lähettää parametrina ohjauskomentoja releen tilan muuttamiseksi, esimerkiksi ”/relay?state=ON” tai ”/relay?state=OFF”. Käyttöliittymäpainikkeet lähettävät nämä GET-pyynnöt laitteelle ja saa vastauksena releen uuden tilan.

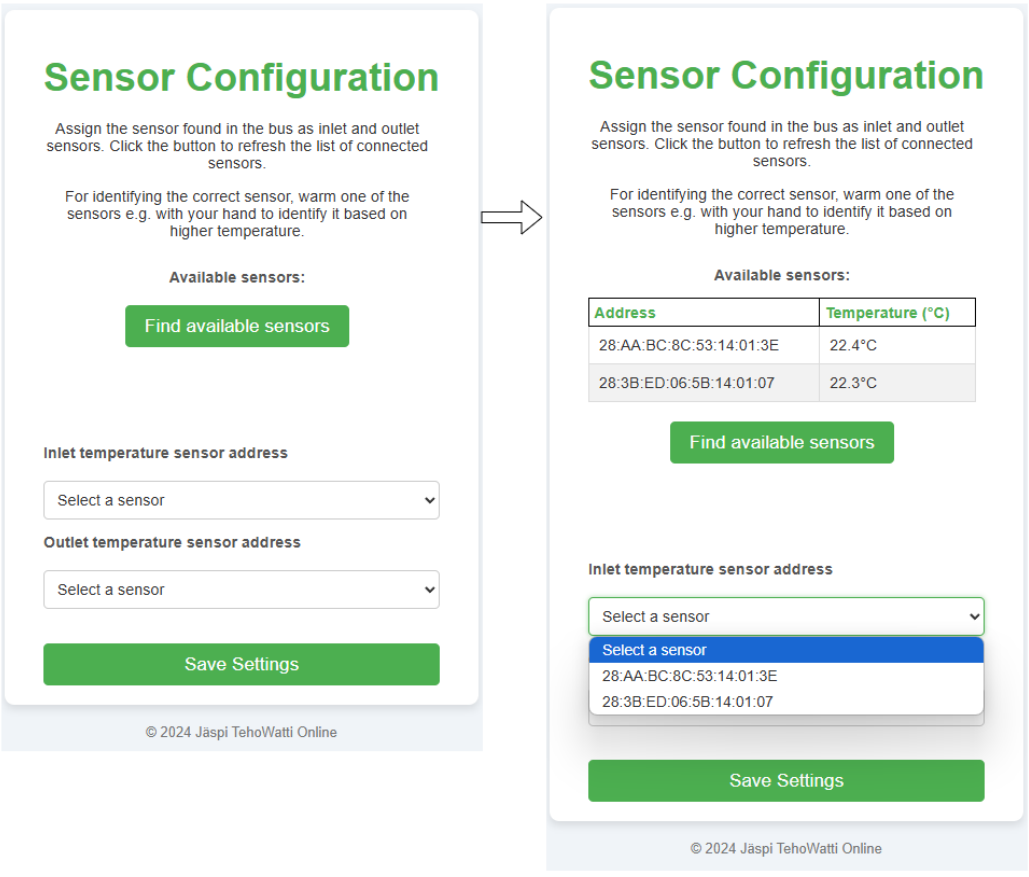
”Restart device” -painike lähettää GET-pyynnön reittiin ”/restart”, jolloin laite käynnistyy uudelleen.




Kuva 48. Status-sivun ohjauspainikkeet

### 8.6.4 Sensor Config -sivu

Sensoriosoitteiden konfigurointinäkymä koostuu skannauksen perusteella väylästä löytyneiden sensoreiden taulukkomuotoisesta esityksestä sekä alasvetovalikoista koostuvasta lomakkeesta. Kun SensorConfig.html-fragmentti on ladattu, sensoritaulukko ja valikot ovat tyhjiä.




Kuva 49. ”Find available sensors” painettu

Käyttäjän painaessa ”Find available sensors” -painiketta selain lähettää laitteelle HTTP-pyynnön reittiin ”/sensors”, joka puolestaan kutsuu SensorManager-olion metodia getSensorData(). Kyseinen metodi etsii väylään liitetyt sensorit ja palauttaa jokaisen sensorin väyläosoitteen sekä hetkellisen lämpötilan JSON-objektina. Löydetyt sensorit esitetään JavaScript-koodilla generoidussa taulukossa, ja alasvetovalikot täydennetään valittavissa olevilla väyläosoitteilla. Käyttäjä voi nyt määrittää mikä väylään kytketty sensori on liitetty tulo- ja menovesilinjaan. Mikäli sensoreita ei löydy, taulukon tilalla näytetään virheilmoitus ja kehote tarkastaa laitteen johdotus.

Kun halutut väyläosoitteet on liitetty oikeisiin muuttujiin, ”Save Settings”-painikketta painamalla JavaScript-funktio luo lomakkeesta JSON-objektin ja lähettää sen POST-kutsun body-lohkossa laitteen ”/settings” -reittiin. Reitti vastaanottaa objektin ja asettaa uudet arvot ConfigManager-olion jäsenmuuttujiin setter-metodeja käyttäen, tallentaa uuden konfiguraation pysyväismuistiin ConfigManager-luokan saveConfig() -metodin avulla ja palauttaa käyttäjälle kehotteen käynnistää laite uudelleen.

```
<h1>Sensor Configuration</h1>
<div class="sensorcontainer">
  <p>
    Assign the sensor found in the bus as inlet and outlet sensors. Click the button to refresh the list of connected sensors.
  </p>
  <p>
    For identifying the correct sensor, warm one of the sensors e.g. with your hand to identify it based on higher temperature.
  </p>
  <div style="margin-top: 30px" id="availableSensors">
    <label>Available sensors:</label>

    </table>
  </div>
  <button id="getSensorsButton">Find available sensors</button>
</div>

<form id="settingsForm">
  <label for="inletSensorAddress">Inlet temperature sensor address</label>
  <select id="inletSensorAddress" name="inletSensorAddress" required>
    <option value="">Select a sensor</option>
  </select>

  <label for="outletSensorAddress">Outlet temperature sensor address</label>
  <select id="outletSensorAddress" name="outletSensorAddress" required>
    <option value="">Select a sensor</option>
  </select>

  <button type="button" id="saveButton">Save Settings</button>
</form>
```

Kuva 50. sensorconfig.html -fragmentti

```
on("/sensors", HTTP_GET, [this, &sensors]() {

    sendHeader("Access-Control-Allow-Origin", "*");
    sendHeader("Access-Control-Allow-Methods", "GET, POST, OPTIONS");
    sendHeader("Access-Control-Allow-Headers", "Content-Type");

    std::vector<SensorData> sensorList = sensors.getSensorData();

    Serial.println(sensorList.size());

    JsonDocument  doc; // Adjust size if needed
    JsonArray sensorArray = doc.createNestedArray("sensors");

    for (const auto& sensor : sensorList) {
        JsonObject obj = sensorArray.createNestedObject();
        char addressStr[25];
        sprintf(addressStr, "%02X:%02X:%02X:%02X:%02X:%02X:%02X:%02X",
            sensor.address[0], sensor.address[1], sensor.address[2], sensor.address[3],
            sensor.address[4], sensor.address[5], sensor.address[6], sensor.address[7]);
        obj["address"] = addressStr;
        obj["temperature"] = sensor.temperature;
        Serial.println(addressStr);
        Serial.println(sensor.temperature);
    }

    String jsonResponse;
    serializeJson(doc, jsonResponse);
    send(200, "application/json", jsonResponse);
});

};
```

Kuva 51. Palvelimen "/sensors" -reitti joka palauttaa osoitteet ja lämpötilat

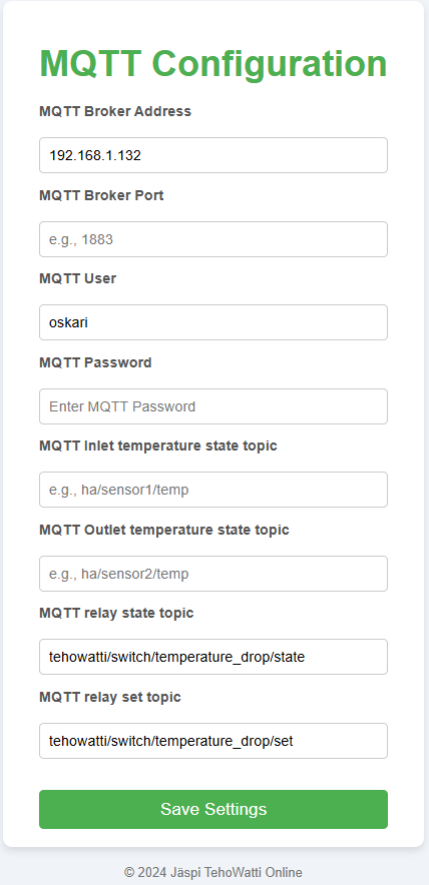
Kuvassa 52 on esitetty selaimen suorittama JavaScript-funktio, jota kutsutaan ”Find available sensors” -painikkeella. Funktio lähettää HTTP-kutsun ”/sensors” -reittiin, saa vastauksena listan löydetyistä sensoreista ja täydentää sensoritaulukon sekä alasvetovalikot tai esittää virheilmoituksen.

```
async function getAvailableSensors() {
  console.log("Fetchign available sensors");

  let sensorData = await loadData(fullUrl+"sensors")

  let sensorDiv = document.getElementById("availableSensors");
  let inletDropdown = document.getElementById("inletSensorAddress");
  let outletDropdown = document.getElementById("outletSensorAddress");


  // Clear the div
  sensorDiv.innerHTML = "<label>Available sensors:</label>";
  inletDropdown.innerHTML = '<option value="">Select a sensor</option>';
  outletDropdown.innerHTML = '<option value="">Select a sensor</option>';

  if (!sensorData.sensors || sensorData.sensors.length === 0) {
    sensorDiv.innerHTML += "<p>No sensors found, check the wiring ⚠️"
  } else {
    // Generate a table
    let table = document.createElement("table");
    table.classList.add("sensor-table");

    // Headers
    let thead = table.createTHead();
    let headerRow = thead.insertRow();
    ["Address", "Temperature (°C)"].forEach((text) => {
      let th = document.createElement("th");
      th.textContent = text;
      th.style.border = "1px solid black";
      th.style.padding = "5px";
      headerRow.appendChild(th);
    });

    // Create table body
    let tbody = table.createTBody();
    sensorData.sensors.forEach((sensor) => {
      // Add dropdown options
      let option1 = document.createElement("option");
      option1.value = sensor.address;
      option1.textContent = sensor.address;
      inletDropdown.appendChild(option1);

      let option2 = document.createElement("option");
      option2.value = sensor.address;
      option2.textContent = sensor.address;
      outletDropdown.appendChild(option2);

      // Add rows to table
      let row = tbody.insertRow();

      // Address column
      let addressCell = row.insertCell();
      addressCell.textContent = sensor.address;

      // Temperature column
      let tempCell = row.insertCell();
      tempCell.textContent = sensor.temperature.toFixed(1) + "°C"; // 1 decimal place
    });

    // Append table to div
    sensorDiv.appendChild(table);
  }
}
```

Kuva 52. Sensoritiedot konfigurointisivulle täydentävä JavaScript-funktio

### 8.6.5 WiFi Config ja MQTT Config -sivut

Yhteys- ja MQTT-konfiguraatiosivut ovat tekniseltä toteutukseltaan identtiset. Molemmat fragmentit sisältävät lomakkeen tarvittavine syötteineen. Tässä kappaleessa on esitetty vain MQTT-konfiguraatiolomake.




Kuva 53. Konfiguraatiolomake

Lomakkeilla on syötekentät kaikille laitteen toiminnan kannalta kriittisille pysyväismuistiin tallennettaville asetuksille. Mikäli laitteen ConfigurationManager-oliosta löytyy aiemmin tallennetut konfiguraatiotiedot, kentät täydennetään valmiiksi nykyisillä arvoilla. Selain kutsuu laitteen ”/settings”-reittiä GET-metodia käyttäen, jolloin nykyiset konfiguraatiotiedot salasanaa lukuun ottamatta palautetaan JSON-objektina lomakkeen täyttämistä varten. Salasanaa ei koskaan palauteta laitteelta.

Arvot syötettyään käyttäjä painaa ”Save Settings”-painiketta, jolloin lomakkeen syötteistä muodostetaan JSON-objekti ja se lähetetään ”/settings” -reittiin POST-metodia käyttäen. Lomaketiedot lähetetään viestin body-lohkossa.

Laite vastaanottaa JSON-objektin validoi sen. Mikäli JSON-objektin parsiminen epäonnistuu, tai syötteiden validoinnit palauttavat virheen, reitti palauttaa HTTP-koodin 500 ja viestin virheellisestä syötteestä – muutoin konfiguraatio päivitetään pysyväismuistiin ja palautetaan koodi 200 ja kehote käynnistää laite uudelleen.

```
on("/settings", HTTP_POST, [this, &config]() {

    sendHeader("Access-Control-Allow-Origin", "*");
    sendHeader("Access-Control-Allow-Methods", "GET, POST, OPTIONS");
    sendHeader("Access-Control-Allow-Headers", "Content-Type");

    if(hasArg("plain")) {
        String body = arg("plain"); // Get the json string from request body

        bool validInputs = true; // ConfigManager setters validate the inputs lightly

        JsonDocument doc;
        DeserializationError error = deserializeJson(doc, body);

        // If error in deserialization, send response
        if (error) {
            Serial.print("Failed to deserialize JSON: ");
            Serial.println(error.f_str());
            return send(500, "text/html", "Invalid JSON");
        }

        serializeJsonPretty(doc, Serial); // Print the received Json

        if (doc.containsKey("ssid")) { validInputs &= config.setSSID(doc["ssid"]); }
        if (doc.containsKey("password")) { validInputs &= config.setWiFiPassword(doc["password"]); }
        if (doc.containsKey("mqttServer")) { validInputs &= config.setMqttServer(doc["mqttServer"]); }
        if (doc.containsKey("mqttPort")) { validInputs &= config.setMqttPort(doc["mqttPort"]); }
        if (doc.containsKey("mqttUser")) { validInputs &= config.setMqttUser(doc["mqttUser"]); }
        if (doc.containsKey("mqttPassword")) { validInputs &= config.setMqttPassword(doc["mqttPassword"]); }
        if (doc.containsKey("inletTempStateTopic")) { validInputs &= config.setInletTempStateTopic(doc["inletTempStateTopic"]); }
        if (doc.containsKey("outletTempStateTopic")) { validInputs &= config.setOutletTempStateTopic(doc["outletTempStateTopic"]); }
        if (doc.containsKey("relayStateTopic")) { validInputs &= config.setRelayStateTopic(doc["relayStateTopic"]); }
        if (doc.containsKey("relaySetTopic")) { validInputs &= config.setRelaySetTopic(doc["relaySetTopic"]); }
        if (doc.containsKey("inletSensorAddress")) { validInputs &= config.setInletSensorAddress(doc["inletSensorAddress"]); }
        if (doc.containsKey("outletSensorAddress")) { validInputs &= config.setOutletSensorAddress(doc["outletSensorAddress"]); }

        if(!validInputs) {
            send(500, "text/html", "Some inputs were invalid, check empty inputs and formats");
            return;
        }

        config.saveConfig();

        // Send OK response
        send(200, "text/html", "Settings saved succesfully, after finalizing configuration restart device from status page or unplug!");

    } else {
        send(400, "text/plain", "No config received!");
    }
});
```

Kuva 54. Lomaketietojen tallentaminen JSON-objektista laitteessa

Syötteiden validointi tapahtuu serverin päässä ConfigManager-olion setter-metodeissa. Yleisesti käyttäjä ei voi lähettää liian pitkiä tai tyhjiä syötteitä, mutta tietyt konfiguraatiot validoidaan tarkemmin ennen tallentamista. Esimerkiksi MQTT-portin on oltava numeerinen arvo ja sallitulla porttialueella sekä MQTT-palvelimen IP-osoitteen on oltava rakenteeltaan oikeanlainen. Setterit palauttavat true/false -arvoja, joiden mukaan myös reitin vastaus selaimelle määräytyy.

```
/**
 * @brief Validates and stores MQTT broker address
 *
 * _mqttServer is type IPAddress, which contains method to parse IP-address from
 * string and validate it is proper. The received chars are parsed to IP and result is returned.
 * If IP address structure was it will fail.
 *
 * @param mqttServer The MQTT broker address as chars
 * @return true if possible to parse into IPAddress, else false.
 */
bool ConfigManager::setMqttServer(const char* mqttServer) {
    // Parse and validate IP address with method that returns false if string cannot be parsed to IPAddress object
    if(!_mqttServer.fromString(mqttServer)) {
        return false;
    };

    return true;
}

int ConfigManager::getMqttPort() {
    return _mqttPort;
}

/**
 * @brief Stores MQTT port as integer to member variable
 *
 * Converts the received mqttPort as chars into integer, and lightly validates that the port is > 0 and < 65535
 *
 * @param mqttPort as chars
 * @return true if was convertable and > 0 and < 65535 else false
 */
bool ConfigManager::setMqttPort(const char* mqttPort) {

    int port = atoi(mqttPort);
    if(port > 0 && port <= 65535) {
        _mqttPort = port;
        return true;
    }

    return false;
}
```

Kuva 55. ConfigManager -luokan settereitä

### 8.6.6 Tyylitiedosto ja responsiivisuus

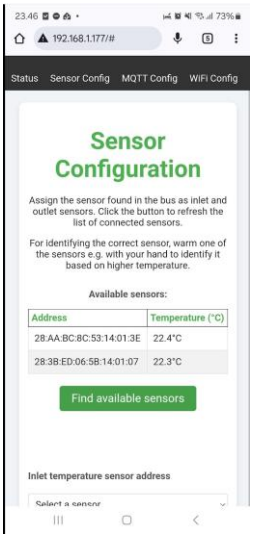
Vähäisen muistin vuoksi käyttöliittymässä ei käytetty lainkaan erillisiä tyylikirjastoja, vaan muotoilut tehtiin puhtaasti CSS-kielellä hyödyntämällä erilaisia CSS-valitsimia. CSS-tiedosto on perinteiseen tapaan linkitetty index.html -sivuun jotta se ladataan käyttöliittymän lataamisen yhteydessä.

Laitteen käyttöönoton yhteydessä on luonnollista käyttää erilaisia mobiililaitteita, minkä vuoksi näkymistä haluttiin tehdä responsiivisia. Responsiivisuus saavutettiin käyttämällä elementeille flex-tyylejä ja kuvan 56 mediakyselyyn pohjautuvaa määritystä hyödyntämällä.

```css
/* Responsive Design */
@media (max-width: 768px) {
  .sidebar {
    width: 100%;
    position: relative;
    height: auto;
    flex-direction: row;
    justify-content: flex-start;
  }

  .sidebar a {
    font-size: 1em;
    padding: 10px;
    text-align: center;
  }

  .main-content {
    margin-left: 0;
  }

  .container {
    width: 90%;
    padding: 15px;
  }

  .status-card {
    width: 90%;
    margin-right: 0;
    margin-bottom: 15px;
  }

  .btn-container {
    padding-top: 15px;
  }

  select {
    font-size: 1em;
    padding: 8px;
  }
}
```

Kuva 56. Mediakyselyyn pohjautuva responsiivisuus

Kun päätelaitteen näytön leveys on alle 768 pikseliä, HTML-elementtien ominaisuuksia muutetaan, jolloin esimerkiksi valikkorakenne siirtyy yläreunaan.




Kuva 57. Mobiilinäkymä

## 8.7 Muut luokat ja toiminnot

### 8.7.1 Konfiguraation ja tiedostojen hallinta

FileManager-luokka kapseloi tiedostojärjestelmään liittyvät operaatiot virheenkäsittelyineen yksittäisiin metodeihin, jotta tiedostojen käsittely on suoraviivaisempaa muissa luokissa ja pääohjelmassa.

ConfigManager-luokka puolestaan on tärkeässä roolissa, sillä se paitsi lukee ja kirjoittaa konfiguraatiotiedot pysyväismuistiin FileManager-luokkaa hyödyntämällä, myös ylläpitää ja välittää konfiguraatiotietoja muille luokille laitteen ollessa toiminnassa. ConfigManager-luokka kapseloi kaikki tarvittavat validoinnit ja konfiguraatiotietojen muotoilut eri käyttötarpeisiin.

Yksittäiset konfiguraatiotiedot ovat luokan yksityisiä jäsenmuuttujia, joita voidaan käsitellä ainoastaan getter- ja setter-metodien avulla. Näin jäsenmuuttujien muokkaamista ja lukemista voidaan kontrolloida luokan sisäisesti.

```
class ConfigManager {
    public:
        ConfigManager(FileManager &fm);
        void loadConfig();
        bool saveConfig();
        String getConfigAsJson();

        // Getters
        const char* getSSID();
        const char* getWiFiPassword();
        IPAddress getMqttServer();
        int getMqttPort();
        const char* getMqttUser();
        const char* getMqttPassword();
        const char* getInletTempStateTopic();
        const char* getOutletTempStateTopic();
        const char* getRelayStateTopic();
        const char* getRelaySetTopic();
        const char* getInletSensorAddress();
        const char* getOutletSensorAddress();

        // Setters
        bool setSSID(const char* ssid);
        bool setWiFiPassword(const char* password);
        bool setMqttServer(const char* mqttServer);
        bool setMqttPort(const char* mqttPort);
        bool setMqttUser(const char* mqttUser);
        bool setMqttPassword(const char* mqttPassword);
        bool setInletTempStateTopic(const char* inletStateTopic);
        bool setOutletTempStateTopic(const char* outletStateTopic);
        bool setRelayStateTopic(const char* relayStateTopic);
        bool setRelaySetTopic(const char* relaySetTopic);
        bool setInletSensorAddress(const char* inletSensorAddress);
        bool setOutletSensorAddress(const char* outletSensorAddress);

    private:
        FileManager &_fm;

        // WiFi configs
        char _SSID[32] = "";
        char _wifiPassword[64] = "";

        // MQTT configs
        IPAddress _mqttServer = IPAddress(0, 0, 0, 0); // PubSubClient needs IPAddress object
        int _mqttPort = 0;
        char _mqttUser[32] = "";
        char _mqttPassword[64] = "";

        // MQTT topics
        char _inletTempStateTopic[64] = "";
        char _outletTempStateTopic[64] = "";
        char _relayStateTopic[64] = "";
        char _relaySetTopic[64] = "";

        //Sensor configs
        char _inletSensorAddress[24] = "";
        char _outletSensorAddress[24] = "";

        // Validate lenght for strings in setters
        bool _validateLength(const char* data, unsigned long target);
};
```

Kuva 58. ConfigManager-luokan otsikkotiedosto

Ohjelman käynnistyessä luokasta alustettu olio lukee konfiguraatiot pysyväismuistista loadConfig()-metodia pääohjelmasta kutsumalla. Tämän jälkeen konfiguraatiot ovat globaalisti saatavilla kaikissa

luokissa, jotka ovat saaneet viittauksen tähän olioon. Pääohjelmassa olion jäsenmuuttujia käytetään lähes kaikkien muiden olioiden alustamiseen.

```
void setup() {
  // Initialize and start the display
  oled.begin(SSD1306_SWITCHCAPVCC, 0x3C);
  oled.showBootMsg("Booting..");

  // Initialize and start the led operation, show alert color until fully booted
  led.enable();
  led.alert();

  Serial.begin(9600); // Open the serial port
  fm.begin(); // Initialize the FS
  config.loadConfig(); // Load all configs from file

  // Set the sensor addresses from config
  sensors.setInletSensorAddress(config.getInletSensorAddress());
  sensors.setOutletSensorAddress(config.getOutletSensorAddress());

  wm.setCredentials(config.getSSID(), config.getWiFiPassword(), DEVICE_NAME);

  oled.showBootMsg("Trying to establish WiFi Connection");
  if(!wm.connect()) {
    oled.showBootMsg("WiFi failed, starting AP");
    wm.startSoftAP();
  }

  server.begin(); // Start the web server
  mqttClient.setServer(config.getMqttServer(), config.getMqttPort());
  mqttClient.setKeepAlive(15);
  mqttClient.setCallback(callback); // Assign callback function to execute when MQTT-msg is received
  connectMqtt();

  oled.clearDisplay();
  oled.display();
}
```

Kuva 59. Oliot konfiguroidaan pääohjelman setup-lohkossa ConfigManagerin

### 8.7.2 Sensoreiden hallinta

Lämpötilasensoreiden hallinnasta ja luennasta vastaa SensorManager-luokka. Luokka ei peri muista luokista, mutta hyödyntää tunnettua DallasTemperature-kirjastoa. SensorManager-luokka säilyttää DallasTemperature-luokan olion jäsenmuuttujanaan ja kapseloi sovelluskohtaiset lisätoiminnot sekä sovelluslogiikan omiin, kuvassa 60 esitettyihin jäsenmuuttujiinsa ja -metodeihinsa. DallasTemperature -luokka itsessään sisältää tarvittavat metodit DS18B20-lämpötilasensoreiden luentaan ja skannaukseen. SensorManager-luokka sisältää tarvittavat toiminnallisuudet lämpötila-arvojen ja osoitteiden ylläpitämiseen sekä erityisesti sensoriarvojen lukemiseen non-blocking-metodilla.

Tärkeimpänä lisänä DallasTemperature-luokan metodeihin SensorManager-luokan readSensors()-metodi mahdollistaa lämpötilojen hakemisen toistuvasti ohjelmaa pysäyttämättä, ilman että web-palvelin ja MQTT-kommunikaatio häiriintyvät. Sensoreiden lukuoperaatio on suhteellisen hidas, joten sitä ei voida suorittaa pääsilmukan jokaisella kierroksella. ReadSensors()-metodi kapseloi tarvittavan logiikan, jolla luenta suoritetaan vain määritetyn aikaviiveen välein vaikka metodia kutsuttaisiin pääsilmukan jokaisella kierroksella.

```
// A struct for holding sensor address + current temp values
struct SensorData {
    uint8_t address[8];  // Store sensor address (8 bytes)
    float temperature; // Store current temperature
};

class SensorManager {
public:
    SensorManager(const int onewire_pin);
    void readSensors(); // Update sensorvalues
    std::vector<SensorData> getSensorData(); // Search available sensor address

    float getInletTemp();
    float getOutletTemp();
    float getLastInletTemp();
    float getLastOutletTemp();

    const char* getInletSensorAddress();
    void setInletSensorAddress(const char* inletSensorAddress);
    const char* getOutletSensorAddress();
    void setOutletSensorAddress(const char* outletSensorAddress);

private:
    OneWire _oneWire;
    DallasTemperature _sensors;

    // Store the correct sensor addresses
    DeviceAddress _inletSensorAddress;
    DeviceAddress _outletSensorAddress;

    // Current measured temperatures
    float _inletTemp = 0.00;
    float _outletTemp = 0.00;

    // Last measured temperatures
    float _lastInletTemp = 0.00;
    float _lastOutletTemp = 0.00;

    // Utils
    void _parseAddressFromString(const char* addr, DeviceAddress &outAddress);
};

#endif
```

Kuva 60. SensorManager-luokan otsikkotiedosto

```
void SensorManager::readSensors() {

    static unsigned long lastSensorRead = 0;

    if (millis() - lastSensorRead > SENSOR_READ_INTERVAL){
        _sensors.requestTemperatures();
        _lastInletTemp = _inletTemp;
        _inletTemp = _sensors.getTempC(_inletSensorAddress);

        _lastOutletTemp = _outletTemp;
        _outletTemp = _sensors.getTempC(_outletSensorAddress);

        lastSensorRead = millis();

    }
};
```

Kuva 61. readSensors()-metodi lämpötilojen lukemiseksi

### 8.7.3 Fyysiset indikaattorit

Laitteen fyysiset indikaattorit, LED-merkkivalo ja OLED-näyttö, mahdollistavat tilan tarkastamisen nopeasti esimerkiksi teknisessä tilassa vierailtaessa.

LED-indikaattoria kontrolloiva luokka perii Adafruit_NeoPixel-luokan, joka mahdollistaa WS2812B-tyyppisten RGB-ledien ohjaamisen. Luokka on hyvin yksinkertainen – yksityisiksi jäsenmuuttujiksi alustetaan eri tiloja kuvastavat värit, ja luokalla on metodit, jotka vaihtavat tilaa ennalta määrättyihin väreihin esimerkiksi valmiustilasta virhetilaan siirryttäessä. Luokka kapseloi itseensä värien määrittämisen lisäksi laitteen tilan määrittämisen, minkä perusteella väriä vaihdetaan tarvittaessa.

```h
#ifndef LED_MANAGER_H
#define LED_MANAGER_H

#include <Adafruit_NeoPixel.h>
#include <Arduino.h>

class LEDManager : Adafruit_NeoPixel {
public:
    LEDManager();
    void standby();
    void alert();
    void relayActive();
    void apMode();
    void enable();
    void setStatus(bool apMode, bool mqttConnection, const char* relayState);
    const char* getStatus();

private:
    uint32_t _standByColor;
    uint32_t _alertColor;
    uint32_t _relayActivatedColor;
    uint32_t _apModeColor;
    const char* _status;
};

#endif
```

Kuva 62. LEDManager-luokan otsikkotiedosto

Luokan julkinen setStatus()-metodi ottaa parametrina vastaan muiden komponenttien tilat, ja määrittää näiden perusteella esitettävän värin. Metodi on non-blocking, sillä tila päivitetään ainoastaan, mikäli muut tarkasteltavat tilat ovat muuttuneet.

```h
void LEDManager::setStatus(bool apMode, bool mqttConnection, const char* relayState) {
    static bool lastApMode = false;
    static bool lastMqttConnection = true;
    static const char* lastRelayState = "OFF";
    static const char* lastState = "";

    if (apMode && apMode != lastApMode) {
        lastApMode = apMode;
        lastState = "apMode";
        this->apMode();
    } else if (!mqttConnection && mqttConnection != lastMqttConnection && !apMode) {
        lastMqttConnection = mqttConnection;
        lastState = "mqttConnection";
        this->alert();
    } else if (relayState == "ON" && relayState != lastRelayState && !apMode) {
        lastRelayState = relayState;
        lastState = "relayActivated";
        this->relayActive();
    } else if (lastState != "standby" && !apMode && mqttConnection && relayState != "ON") {
        lastState = "standby";
        this->standby();
    };
};
```

Kuva 63. LED-yksikköä ohjaava logiikka

OLEDManager-luokka puolestaan vastaa laitteen etupaneelissa sijaitsevan näytön päivittämisestä, ja se periytyy Adafruit_SSD1306-luokasta. Luokka ohjaa näyttöä riviperustaisesti – jokaiselle esitettävälle muuttujalle on varattu näytöstä yksi rivi, ja kyseinen rivi päivitetään vain, mikäli muuttujan arvo on päivittynyt. Tämä mahdollistaa näytön ohjaamisen non-blocking-metodina.

```h
#ifndef OLED_MANAGER_H
#define OLED_MANAGER_H

#include <Wire.h>
#include <Adafruit_SSD1306.h>

class OLEDManager : public Adafruit_SSD1306 {
public:
    OLEDManager();
    void updateDisplay(float inletTemp, float outletTemp, const char* relayState, const char* ip, const char* mode, boolean mqtt);
    void showBootMsg(const char* text);
private:
    void _drawRow(int row, const String& text);
};

#endif
```

Kuva 64. OLEDManager-luokan otsikkotiedosto

```
*/
void OLEDManager::updateDisplay(float inletTemp, float outletTemp,const char* relayState,const char* ip, const char* mode, boolean mqtt) {
    static float lastInletTemp = -999;
    static float lastOutletTemp = -999;
    static const char* lastRelayState = "";
    static const char* lastIp = "";
    static const char* lastMode = "";
    static bool lastMqtt = NULL;

    if (inletTemp != lastInletTemp) {
        lastInletTemp = inletTemp;
        _drawRow(1, "Inlet: " + String(inletTemp, 1) + " C");
    }

    if (outletTemp != lastOutletTemp) {
        lastOutletTemp = outletTemp;
        _drawRow(2, "Outlet: " + String(outletTemp, 1) + " C");
    }

    if (lastRelayState != relayState) {
        lastRelayState = relayState;
        _drawRow(3, "Relay: " + String(relayState));
    }

    if (lastIp != ip || lastMode != mode) {
        lastIp = ip;
        lastMode = mode;
        _drawRow(4, "IP-"+ String(mode) + ":" + String(ip));
    }

    if (lastMqtt != mqtt || lastMqtt == NULL) {
        lastMqtt = mqtt;
        _drawRow(5, mqtt ? "MQTT: Connected" : "MQTT: Disconnected");
    }
};

/**
 * @brief Private method to draw single row in the OLED display
 *
 * Method draws one row at a time and only when needed to avoid overhead of writing to OLED. It receives the
 * row number and content to write as parameter. First the existing contents are overwritten by drawing
 * black reactangle and then new content is written and display() is called to actually show the content.
 *
 * @param row The rownumber to rewrite
 * @param text The text content to write
 */
void OLEDManager::_drawRow(int row, const String& text) {
    setTextSize(1);
    setTextColor(SSD1306_WHITE);
    fillRect(0, row * 10, 128, 10, SSD1306_BLACK); // First erase contents from the row to be updated, otherwise pixels just add up
    setCursor(0, row * 10);
    print(text);
    display();
};
```

Kuva 65. Näytön päivitys rivikohtaisesti

### 8.7.4 Relettä ohjaava luokka

Relettä ohjaa Relay-luokan olio. Luokka kapseloi I/O-pinnien ohjaamiseen liittyvät komennot yksinkertaisiksi on()-, off()- ja toggle()-metodeiksi. Tämän lisäksi luokan tehtävä on ylläpitää releen tilaa, jotta se voidaan palauttaa tarvittaessa käyttöliittymään tai OLED-näyttöön.

```
#ifndef Relay_h
#define Relay_h

#include <Arduino.h>

class Relay {

public:
    Relay(int relayPin);
    void on();
    void off();
    void toggle();
    const char* getState(); // Return the current state of relay
private:
    int _relayPin;
    const char* _state = "OFF"; // Store the state of relay after toggling
    void setState(const char* state);
};

#endif
```

Kuva 66. Relay-luokan otsikkotiedosto

```
Relay::Relay (int relayPin) {
    _relayPin = relayPin;
    pinMode(_relayPin, OUTPUT_OPEN_DRAIN);
    digitalWrite(_relayPin, HIGH);
};

/**
 * @brief Sets the I/O pin state to LOW which activates the relay
 *
 * Sets the relayPin LOW using digitalWrite, prints the state change and sets the
 * relay object's internal state tracker to corresponding value.
 */
void Relay::on() {
    digitalWrite(_relayPin, LOW);
    Serial.println("Relay set ON");
    setState("ON");
};

/**
 * @brief Sets the I/O pin state to HIGH which deactivates the relay
 *
 * Sets the relayPin HIGH using digitalWrite, prints the state change and sets the
 * relay object's internal state tracker to corresponding value.
 */
void Relay::off() {
    digitalWrite(_relayPin, HIGH);
    Serial.println("Relay set OFF");
    setState("OFF");
};
```

Kuva 67. Luokka kapseloi I/O-komennot

## 8.8 MQTT-viestintä

MQTT-viestintä välityspalvelimen kanssa on toteutettu Arduinolle kirjoitetulla, yleisesti saatavilla olevalla PubSubClient-kirjastolla ilman muokkauksia. Kirjastoa käyttäen viestien vastaanottaminen ja lähettäminen on hyvin yksinkertaista. MQTT-viestien käsittelyyn liittyvä logiikka on sisällytetty pääohjelmaan.

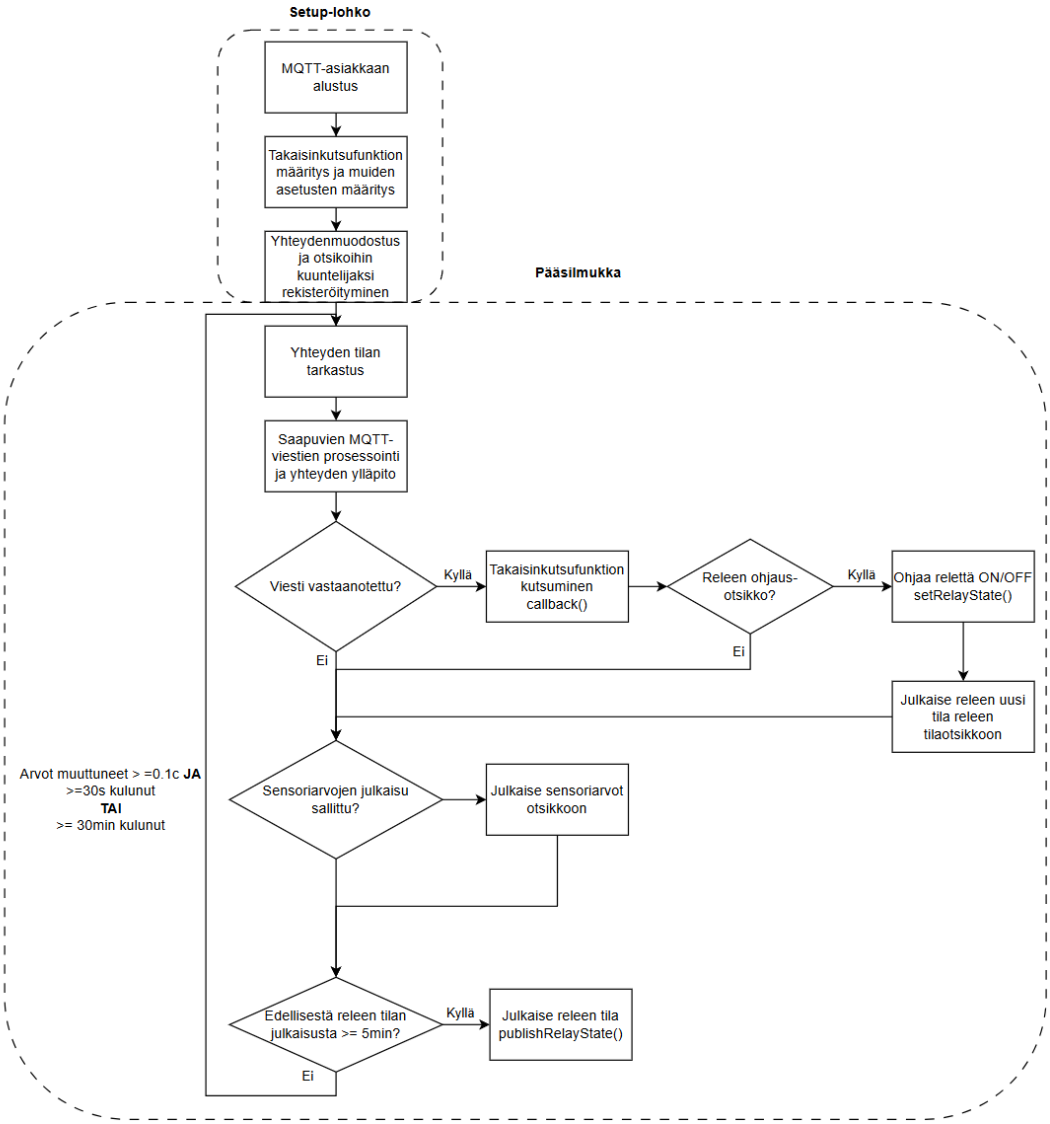
Kuvan 68 kaaviossa on esitetty ohjelman MQTT-viestinnän setup-lohkoon ja pääsilmukkaan sijoitettu työnkulku tarkemmalla tasolla. Setup-lohkossa alustetaan MQTT-asiakas, annetaan sille tarvittavat parametrit ja tehdään ensimmäinen yhteyskokeilu välityspalvelimeen sekä rekisteröidytään tarvittaviin otsikoihin kuuntelijaksi. Tämän jälkeen siirrytään pääsilmukkaan, jossa jokaisella suoituskerralla tarkastetaan yhteyden tila, tarkastetaan ja käsitellään mahdollisesti seurattaviin otsikoihin saapuneet MQTT-viestit ja julkaistaan sensoriarvot sekä releen tilatiedot.

Sensoriarvojen ja releen tilatiedon julkaisutiheyttä on rajoitettu ohjelmallisesti, sillä liian tiheä julkaiseminen ei tuota lisäarvoa käyttäjälle, aiheuttaa ylimääräistä liikennettä välityspalvelimelle ja hidastaa ohjelman suoritusta tarpeettomasti. Sensoriarvot julkaistaan ainoastaan, mikäli lämpötilan oloarvo on muuttunut >= 0.1 astetta ja edellisestä julkaisusta on kulunut yli 30 sekuntia, tai aina 30 minuutin välein. Releen tilatieto (ON/OFF) julkaistaan aina releen tilan muuttuessa ja viiden minuutin välein, jotta esimerkiksi sähkökatkostilanteissa releen tilatieto synkronoituu.

Taulukoon 6 on kirjattu opinnäytetyön käyttökohteessa konfiguroidut MQTT-otsikot.

Taulukko 6. Laitteeseen konfiguroidut MQTT-otsikot

| **Otsikko** | **Kuvaus** |
|---|---|
| tehowatti/sensor/inlet_temperature | Lähetetään tuloveden lämpötilat |
| tehowatti/sensor/outlet_temperature | Lähetetään menoveden lämpötilat |
| tehowatti/switch/temperature_drop/state | Lähetetään releen tilatiedot |
| tehowatti/switch/temperature_drop/set | Kuunnellaan releen ohjauskäskyjä |




Kuva 68. MQTT-viestinnän työnkulku

HomeAssistantin kanssa kommunikoitaessa releen ohjaus tapahtuu aina kahdensuuntaisesti. Home Assistant lähettää halutun releen tilan ”/set”-otsikkoon, ja jää kuuntelemaan ”/state”-otsikkoa johon IoT-laite lähettää tilapäivityksen, kun haluttu muutos on fyysisesti toteutettu. Home Assistant päivittää tilan käyttöliittymään vasta, kun se on saanut kuittauksen päätelaitteelta, että operaatio on onnistunut. Laite lähettää ”/state”-otsikkoon viestin myös aina, kun sitä ohjataan HTTP-pyynnön tai käyttöliittymän kautta, joten Home Assistantin käyttöliittymä pysyy ajan tasaisena, vaikka se ei osallistuisi ohjaukseen


Kuva 69. Viestintä relettä ohjattaessa

Ohjelman setup-lohkossa alustetaan PubSubClient-luokasta olio ”mqttClient”, jolla on tarvittavat metodit yhteyden muodostamiseksi ja viestien käsittelemiseksi. MQTT-asiakkaan määrityksessä tarvittavia parametreja ovat välityspalvelimen osoite, portti, käyttäjätunnus, salasana sekä vastaanotettujen viestien käsittelyyn tarvittava takaisinkutsufunktio.

```
server.begin(); // Start the web server
mqttClient.setServer(config.getMqttServer(), config.getMqttPort());
mqttClient.setKeepAlive(15);
mqttClient.setCallback(callback); // Assign callback function to execute when MQTT-msg is received
connectMqtt();
```

Kuva 70. MQTT-asiakkaan määritys setup-lohkossa

Kuvan 70 märittelyn yhteydessä asiakasoliolle annetaan tunnistetiedot ConfigManager-olion jäsenmuuttujista, ja asetetaan takaisinkutsufunktioksi callback()-funktio. Tämän jälkeen kutsutaan sivulla 18, kuvassa 35 esitettyä connectMqtt()-funktiota. Kyseinen funktio sisältää tarvittavan logiikan määrävälein tapahtuville yhteyskokeiluille ja releen ohjauskäskyjä vastaanottavaan otsikkoon kuuntelijaksi kirjautumisen. Kuvassa 71 esitetty takaisinkutsufunktio suoritetaan aina, kun MQTT-asiakas vastaanottaa viestin mistä tahansa seuratusta otsikosta. Opinnäytetyön tapauksessa kuunnellaan ainoastaan yhtä otsikkoa, joten siihen liittyvää päättelyä ei tarvita. Suoritettavat releen ohjauskomennot riippuvat viestin sisällöstä.

```
void callback(char* topic, byte* payload, unsigned int length) {

  char message[10]; // Buffer large enough for "ON", "OFF", and null terminator

  memcpy(message, payload, length); // Copy payload into buffer
  message[length] = '\0'; // Null-terminate the string

  Serial.println("Message received on topic: ");
  Serial.println(topic);
  Serial.println("Message: ");
  Serial.println(message);

  if (strcmp(message, "ON") == 0) {
    setRelayState(true);
  } else if (strcmp(message, "OFF") == 0) {
    setRelayState(false);
  }
}
```

Kuva 71. MQTT-asiakkaan takaisinkutsufunktio

Takaisinkutsufunktio kutsuu releen tilan vaihtamiseksi pääohjelman globaalia setRelayState()-funktiota joka suorittaa kaikki tarvittavat releen tilamuutoksiin liittyvät operaatiot. Funktio on kutsuttavissa myös WebServerManager-luokasta, jolloin samat operaatiot toistetaan riippumatta siitä, millä mekanismilla releelle välitetään ohjauskäskyjä. Funktio kapseloi yhdeksi toiminnoksi tilan muuttamisen, tilan lähettämisen ”/state”-otsikkoon ja LED-indikaattorin värin muuttamisen.

```
void setRelayState(bool state) {
  if (state) {
    relay.on();
    mqttClient.publish(config.getRelayStateTopic(), "ON", true);
    led.relayActive();
  } else {
    relay.off();
    mqttClient.publish(config.getRelayStateTopic(), "OFF", true);
    led.standby();
  }
}
```

Kuva 72. Pääohjelman globaali setRelayState()-funktio

## 8.9 Home Assistant konfigurointi

Laitteen konfigurointi automaatiojärjestelmään liitetyiksi sensoreiksi ja kytkimeksi oli suoraviivaista. MQTT-viesteihin perustuvien oheislaitteiden konfiguroimiseksi vaadittava MQTT-integraatio oli asennettu järjestelmään jo entuudestaan. MQTT-integraatioon on asennusvaiheessa sisällytetty myös MQTT-välityspalvelin, joten opinnäytetyön tapauksessa Home Assistant -palvelin toimii samalla myös MQTT-välityspalvelimena. Integraatio kykenee vastaanottamaan ja lähettämään MQTT-protokollan viestejä ja parsimaan niistä konfiguraation mukaisia sensoriarvoja tai vastaavasti muuntamaan käyttöliittymän ohjauskomennot MQTT-viesteiksi.

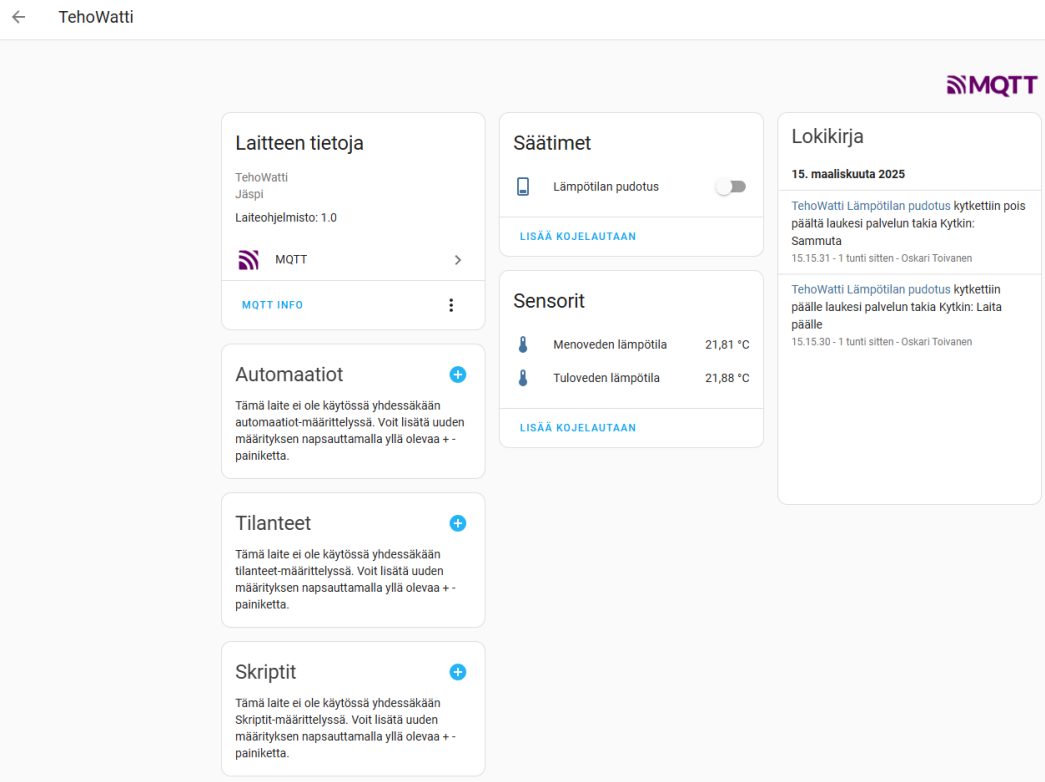
Home Assistantissa käyttäjän omia konfiguraatioita voi toisinaan lisätä käyttöliittymän kautta, mikäli integraation tekijä on mahdollistanut sen – tässä tapauksessa MQTT-entiteettejä voidaan määritellä ainoastaan YAML-kielellä järjestelmän tiedostojärjestelmään tallennetussa configuration.yaml -tiedostossa, ellei käytetä ns. MQTT Discovery -metodia jolloin laite voi lähettää tarvittavat konfiguraatiot suoraan järjestelmälle MQTT-viestinä. Opinnäytetyön tapauksessa päädyttiin tekemään määrittelyt manuaalisesti configuration.yaml -tiedostoon.

Laite konfiguroitiin näkymään Home Assistantissa yksittäisenä ”TehoWatti” -nimisenä laitteena, joka sisältää kaksi lämpötilasensoria ja switch-tyyppisenä ohjattavana kytkimenä. Konfiguraatiotiedostoa voi muokata millä tahansa tekstieditorilla, jolla on pääsy Home Assistantin tiedostojärjestelmään. Tämä konfiguraatio suoritettiin Home Assistantin omalla File editor -lisäosalla.

```
mqtt:
  sensor:
    - name: "Tuloveden lämpötila"
      unique_id: "tehowatti_inlet_temperature"
      state_topic: "tehowatti/sensor/inlet_temperature"
      unit_of_measurement: "°C"
      device_class: "temperature"
      device:
        name: "TehoWatti"
        identifiers: ["tehowatti"]
        manufacturer: "Jäspi"
        model: "TehoWatti"
        sw_version: "1.0"

    - name: "Menoveden lämpötila"
      unique_id: "tehowatti_outlet_temperature"
      state_topic: "tehowatti/sensor/outlet_temperature"
      unit_of_measurement: "°C"
      device_class: "temperature"
      device:
        name: "TehoWatti"
        identifiers: ["tehowatti"]
        manufacturer: "Jäspi"
        model: "TehoWatti"
        sw_version: "1.0"

  switch:
    - name: "Lämpötilan pudotus"
      unique_id: "tehowatti_temperature_drop_switch"
      state_topic: "tehowatti/switch/temperature_drop/state"
      command_topic: "tehowatti/switch/temperature_drop/set"
      device:
        name: "TehoWatti"
        identifiers: ["tehowatti"]
        manufacturer: "Jäspi"
        model: "TehoWatti"
        sw_version: "1.0"
```

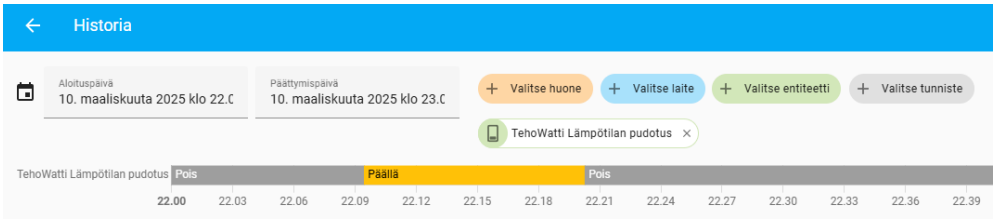
Kuva 73. configuration.yaml-tiedoston sisältö

Kuvassa 73 on esitetty IoT-laitteen tarvittavat konfiguraatiot. YAML-tiedostossa on ensin määritetty integraatio, johon entiteetit lisätään (mqtt). Tämän jälkeen määritellään sensor- ja switch -entiteetti-tyyppien alle sekä tulo- että menoveden lämpötilaa mittaavien sensoreiden tiedot että releeseen liittyvät tiedot. konfiguraatioissa on tärkeää huomata, että jokaisen entiteetin device-osiossa ne ovat määritetty samaan laitteeseen kuuluvaksi, jolloin Home Assistant koostaa entiteetit saman laitteen alle käyttöliittymässä. Tämän lisäksi konfiguraatiossa on määritetty jokaiselle entiteetille nimi, joka näytetään käyttöliittymässä sekä tärkeimpänä MQTT-otsikot, joita kuuntelemalla arvot vastaanotetaan. Switch-tyyppiselle entiteetille on määritetty command_topic ja state_topic ohjauskäskyjen lähettämiseksi ja tilan seuraamiseksi. Kun yllä esitetyt määritetyt on tehty, ja Home Assistant käynnistetty uudelleen, laite entiteetteineen alkaa näkyä Home Assistantissa.




Kuva 74. Home Assistantin laitenäkymä koostaa entiteetit yhteen näkymään

Home Assistant tallentaa sensoreiden ja releen tilatiedot automaattisesti tietokantaan säilyttäen dataa konfiguraation mukaan jopa vuosien ajalta. Arvoja voidaan verrata rinnakkain minkä tahansa toisen entiteetin kanssa. Lämpötilojen lisäksi voidaan seurata myös lämpötilan pudotuksen tilatiedon historiaa.




Kuva 75. Lämpötilan pudotusreleen tilatiedon historia

# 9 TESTAUS JA TULOKSET

Kehitystyön edetessä laitteen perustoimintoja testattiin jatkuvasti, mutta automaattisia tai ohjelmallisia testauksia ei käytetty. Käyttötestauksessa kiinnitettiin erityishuomiota alkuperäisiin käyttäjävaatimuksiin verkkoyhteyden ja käyttöjännitteen menetyksestä. Testitapaukset on kuvattu taulukossa 7.

Taulukko 7. Testitapaukset ja tulokset

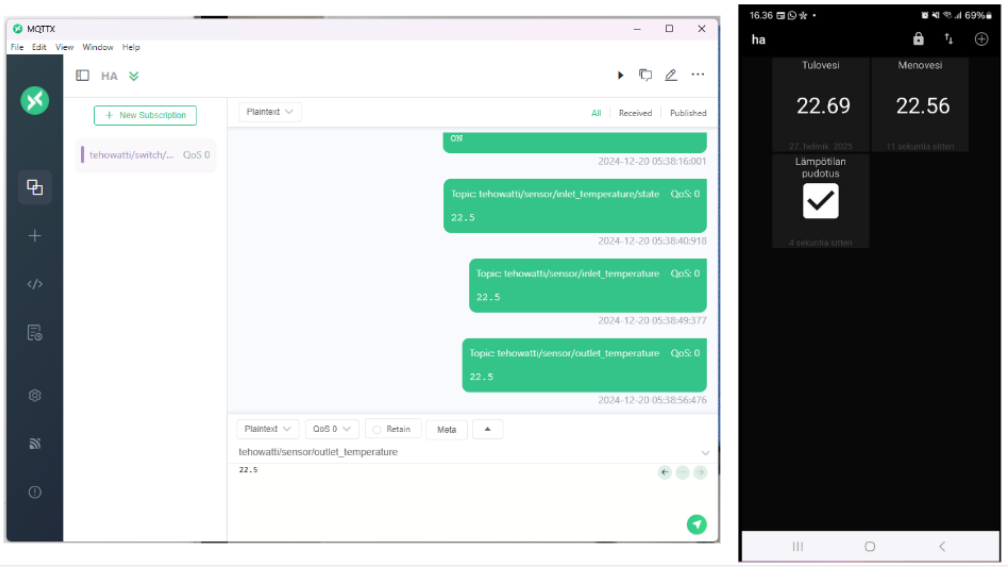
| **Tapaus** | **Testi** | **Tulos** |
|---|---|---|
| Verkkoyhteys katkeaa | WiFi-reititin sammutettiin laitteen ollessa toiminnassa ja käynnistettiin uudelleen | Laite siirtyi automaattisesti AP/uudelleenyhdistämissykliin ja päivitti LED-indikaattorin punaiseksi. Yhteyden palautuessa toiminta normalisoitui |
| MQTT-yhteys menetetään | MQTT-välityspalvelin sammutettiin | Laite meni virhetilaan, ja yhdisti uudelleen välityspalvelimeen tilanteen korjauduttua. Muut toiminnot eivät häiriintyneet |
| Sensori rikkoontuu tai irtoaa | Irrotettiin signaalijohdin | Sensoriarvot näyttivät -127 °C oletuslämpötilaa virheen merkiksi, toiminta ei häiriintynyt |
| IoT-laite, Home Assistant tai MQTT-välityspalvelin käynnistyy uudelleen | Sammutettiin osajärjestelmiä vuorotellen | Järjestelmä palautui toimintakykyiseksi kaikissa tilanteissa |
| Konfigurointi onnistuu käyttöliittymän kautta | Laitteeseen ladattiin puhdas ohjelma ilman konfigurointeja ja tehtiin määritykset alusta alkaen | Konfiguraatiot saatiin tallennettua ja laite siirtyi toimintatilaan odotetusti |
| Sensoriarvojen tarkastus | Verrattiin sensoriarvoja keskenään ja tunnettuun lämpötilamittariin | Todettiin että sensoreissa on 0,5 °C lämpötilapoikkeama toisiinsa nähden. Ei todettu ohjelmallista kalibrointitarvetta |
| Laitetta ohjataan ristiin MQTT-Käyttöliittymä- ja HTTP-pyynnöin | Lähetettiin ohjauskomentoja eri lähteistä ja varmistettiin että tilat päivittyvät kaikkialle | Tilat päivittyivät jokaiseen järjestelmään, vaikka ohjauskomento olisi tullut toisaalta |
| Laite säilyy toimintakykyisenä pitkiä aikoja | Laitteen annettiin olla yhtäjaksoisesti toiminnassa lähes 3 viikkoa esimerkiksi muistivuotojen havaitsemiseksi | Laite säilyi normaalisti toimintakykyisenä koko tämän ajan. |

# 10 YHTEENVETO JA JATKOKEHITYS

Laitteen suunnittelutyö ja tekninen toteutus onnistuivat jopa odotettua paremmin, eikä merkittäviä hidasteita tai ongelmia ilmennyt kehitystyön edetessä. Laite täyttää kaikki sille asetetut tavoitteet ja toimii luotettavasti. Lopputuloksena syntyi käyttökelpoinen tuote kotiautomaatiojärjestelmän oheislaitteeksi.


Kuva 76. Valmis laite ennen käyttöpaikkaan asentamista

Laite on ohjattavissa ja sen tilat luettavissa joko MQTT-asiakasohjelmaa käyttäen, selainpohjaisella käyttöliittymällä tai HTTP-pyynnöillä, joten se ei ole sidottu Home Assistant -järjestelmään vaikka integroitavuus olikin yksi suunnittelulähtökohta. Esimerkkinä laajennettavuudesta on kuvan 75 esimerkit lukuisista Windows- ja Android-sovelluksista joilla kommunikointi on mahdollista.




Kuva 77. MQTT-kommunikaatio Windows- ja Android-sovelluksilla

Jo nyt laitteen avulla onnistuu lämmityskattilan lämpötilojen reaaliaikainen seuranta ja lämpötilan pudotuksen ohjaus. Seuraavaksi Home Assistant -järjestelmään tehdään yksinkertaiset automaatiot lämpötilan pudotuksen ohjaamiseksi ja lämmitysjärjestelmien optimoimiseksi, mutta tämä osa-alue on rajattu opinnäytetyön ulkopuolelle. Vaikka automaatioiden tekeminen järjestelmään on yksinkertaista, säätäminen vienee merkittävän paljon aikaa.

Jatkokehitysmahdollisuuksia on paljon. IoT-laitteeseen jäi lukuisia vapaita I/O-pinnejä, joten siihen on helppo yhdistää tarpeen mukaan lisää toimilaitteita tai sensoreita. Tarvittaessa laitteen tuottamaa dataa voidaan viedä pienin muutoksin myös ulkoiseen tietokantaan tai pilvipalveluun. Lämmityskattilan tuottamaa dataa on mahdollista purkaa myös Home Assistantista, jolloin sen analysointiin voi hyödyntää esimerkiksi tekoälyä sisällyttämällä myös muiden entiteettien, kuten ulkolämpötila, huonelämpötilat, ilmalämpöpumpun tilat sekä tulisijan tilat ja etsimällä optimointikohteita tätä kautta.

Tietoturvakysymykset on tietoisesti rajattu opinnäytetyön ulkopuolelle aiheen laajuuden vuoksi. IoT-laitteiden mahdolliset tietoturvariskit on kuitenkin tärkeää huomioida. Opinnäytetyön tapauksessa laitteen kommunikointi estetään sisäverkon ulkopuolelle eikä laitteen kommunikointimetodit koskaan palauta pysyväismuistiin tallennettuja salasanoja. Jatkokehityksenä laitteeseen voisi kuitenkin ohjelmoida autentikoinnin ja vaihtaa kaikki kommunikointiprotokollat salatuiksi yhteyksiksi. Tämän lisäksi ohjelma sisältää lukuisan määrän vakioarvoja esimerkiksi MQTT-viestien julkaisutiheyteen liittyen – nämä voisi muuttaa käyttöliittymän tai MQTT-integraation kautta dynaamisesti parametroitavaksi.

# 11 POHDINTA

Opinnäytetyön tavoitteena oli kehittää IoT-laite, jonka avulla vanhanaikainen omakotitalon vesikiertoinen lattialämmitysjärjestelmä voidaan integroida moderniin kotiautomaatiojärjestelmään. Työn suunnittelua ja toteutusta helpotti aiempi työ- ja koulutustausta sähkö- ja konetekniikan alalta sekä harrastajakokemus mikrokontrolleriohjelmoinnista. Kuitenkaan ennen tätä projektia en ollut toteuttanut näin kokonaisvaltaista järjestelmää, jossa ohjelmisto koostuu useista moduuleista ja oheiskomponenteista. Tämän vuoksi suunnittelutyö, riskienhallinta ja työjärjestyksen priorisointi olivat erityisen tärkeitä.

Projektin aikana syvensin osaamistani erityisesti olio-ohjelmoinnin, IoT-järjestelmien, MQTT-protokollan ja web-ohjelmoinnin osalta. Työ yhdisti useita opintojen aikana käsiteltyjä aihealueita – koen, että opinnot antoivat hyvän pohjan ohjelmiston rakenteen ja käyttöliittymän suunnitteluun, mutta suurimmat haasteet liittyivät opitun soveltamiseen mikrokontrolleriympäristössä, jossa resurssien hallinta on kriittistä. Koska työn painopiste oli mikrokontrolleriohjelmoinnissa, pyrin alusta alkaen huomioimaan nämä erityispiirteet varmistaakseni järjestelmän tehokkuuden ja vakauden.

Tämä projekti tarjosi arvokkaan mahdollisuuden soveltaa teoreettista osaamista käytännön ongelmanratkaisuun. Se vahvisti ymmärrystäni tietotekniikan eri osa-alueista ja kehitti taitojani kokonaisvaltaisen ratkaisun suunnittelussa ja toteutuksessa. Kokonaisuudessaan projekti oli erittäin opettavainen ja antoi hyvät valmiudet tuleviin haasteisiin niin ohjelmistokehityksen kuin järjestelmäsuunnittelun parissa.

# 12 LÄHTEET


Amazon 2025. OLED Display for Arduino. Verkkojulkaisu. https://www.amazon.co.uk/Serial-Display-Arduino-Performance-Optional/dp/B0D94GPNGT.
Viitattu 15.03.2025.

Amazon 2025. Shanrya 3.3V Relay Module. Verkkojulkaisu. https://www.amazon.com/Relay-Module-Safety-Industrial-Control/dp/B09NQHV8GJ.
Viitattu 15.03.2025.

ANALOG DEVICES n.d. UART: A Hardware Communication Protocol. Verkkojulkaisu.
https://www.analog.com/en/resources/analog-dialogue/articles/uart-a-hardware-communication-protocol.html.
Viitattu 12.02.2025.

Arduino 2023. Arduino & Serial Peripheral Interface (SPI). Verkkojulkaisu.
https://docs.arduino.cc/learn/communication/spi/.
Viitattu 12.02.2025.

Aurelia n.d. Vesikiertoinen lattialämmitys on energiatehokas ratkaisu.
https://aurelialattialammitys.fi/vesikiertoinen-lattialammitys/.
Viitattu 15.03.2025.

AWS n.d. What is MQTT? - MQTT Protocol Explained - AWS. Verkkojulkaisu.
https://aws.amazon.com/what-is/mqtt/.
Viitattu 12.02.2025.

Cisco Press 2019. Communication Protocols for IoT. Verkkojulkaisu.
https://www.ciscopress.com/articles/article.asp?p=2923211&seqNum=6.
Viitattu 12.02.2025.

CISCO n.d. What Is Ethernet? - Cisco. Verkkojulkaisu.
https://www.cisco.com/c/en/us/solutions/enterprise-networks/what-is-ethernet.html.
Viitattu 12.02.2025.

CISCO n.d. What Is Wi-Fi? - Definition and types - Cisco. Verkkojulkaisu.
https://www.cisco.com/c/en/us/products/wireless/what-is-wifi.html
Viitattu 12.02.2025.

Community, A., 2024. 1-Wire Protocol | Arduino Documentation. Verkkojulkaisu.
https://docs.arduino.cc/learn/communication/one-wire/.
Viitattu 12.02.2025.

Energiateollisuus ry 2024. Säätövoima - Energiateollisuus. Verkkojulkaisu.
https://energia.fi/energiatietoa/energiantuotanto/sahkontuotanto/saatovoima/
Viitattu 15.12.2024.

EnergiaYkkönen n.d. Energiatehokkuus rakennuksissa. Verkkojulkaisu.
https://energiaykkonen.fi/energiatehokkuus-rakennuksissa/.
Viitattu 12.02.2025.

Fingrid, n.d. Miten sähkön hinta muodostuu? Verkkojulkaisu.
https://www.fingrid.fi/sahkomarkkinat/yleistietoa-sahkomarkkinoista/miten-sahkon-hinta-muodostuu/.
Viitattu 15.12.2024.

GeeksForGeeks n.d. Components Of Microcontrollers. Verkkojulkaisu.
https://www.geeksforgeeks.org/components-of-microcontroller/.
Viitattu 12.02.2025.

GeeksForGeeks n.d. Constrained Application Protocol (CoAP). Verkkojulkaisu.
https://www.geeksforgeeks.org/constrained-application-protocol-coap/.
Viitattu 12.02.2025.

GeeksForGeeks n.d. Microcontrollers and its types. Verkkojulkaisu.
http://geeksforgeeks.org/microcontroller-and-its-types/.
Viitattu 12.02.2025.

Harshvardhan, M. 2025. Programming Languages for Microcontrollers, Arduino, ESP, and Similar boards. Verkkojulkaisu.
https://iotbyhvm.ooo/programming-languages-for-microcontrollers-arduino-esp-and-similar-boards/.
Viitattu 15.03.2025.

HiveMQ n.d. MQTT Essentials - All Core Concepts Explained. Verkkojulkaisu.
https://www.hivemq.com/mqtt/.
Viitattu 23.02.2025.

Home Assistant 2025. Home Assistant. Verkkojulkaisu.
http://www.home-assistant.io
Viitattu 11.02.2025.

JemRF 2025. Waterproof DS18B20 Dallas 1 wire temperature sensor. Verkkojulkaisu.
https://www.jemrf.com/products/copy-of-ds18b20-dallas-1-wire-digital-temperature-sensor-and-resistor.
Viitattu 15.03.2025.

Jäspi 2016. TehoWatti käyttöohje. Verkkojulkaisu.
https://jaspi.fi/wp-content/uploads/2016/05/Tehowatti_manual_r3_1114.pdf#:~:text=J%C3%A4spi%20Tehowatti%20on%20moduulimitoitettu%20l%C3%A4mmityslaite%2C%20johon%20on%20integroitu,vesikiertoisiin%20lattia-%20ja%20patteril%C3%A4mmi-tysj%C3%A4rjestelmiin%20s.
Viitattu 15.03.2025.

Maxim integrated 2019. DS18B20 - Programmable Resolution 1-Wire Thermometer. Verkkojulkaisu.
https://www.analog.com/media/en/technical-documentation/data-sheets/DS18B20.pdf.
Viitattu 20.02.2025.

Motiva 2024. Energiatehokas pientalo. Verkkojulkaisu.
https://www.motiva.fi/koti_ja_asuminen/energiatehokas_pientalo.
Viitattu 12.02.2025.

Motiva 2024. Hallitse huonelämpötiloja. Verkkojulkaisu.
https://www.motiva.fi/koti_ja_asuminen/energiatehokas_arki/hallitse_huonelampotiloja.
Viitattu 12.02.2025.

Nabto n.d. ESP8266 for IoT: Complete Guide. Verkkojulkaisu.
https://www.nabto.com/esp8266-for-iot-complete-guide/.
Viitattu 12.02.2025.

Nabto n.d. WebSockets vs. MQTT vs. CoAP Which is the best protocol? Verkkojulkaisu.
https://www.nabto.com/websocket-vs-mqtt-vs-coap/.
Viitattu 23.02.2025.

OptiWatti 2019. Sähkölämmitysopas. Verkkojulkaisu.
https://www.optiwatti.fi/wp-content/uploads/2021/06/OptiWatti-Sa%CC%88hko%CC%88la%CC%88mmitysopas.pdf.
Viitattu 11.02.2025.

Polycase 2020. IoT Enclosures: Buyer's guide to IoT device housings. Verkkojulkaisu.
https://www.polycase.com/techtalk/plastic-electronic-enclosures/iot-enclosures-buyers-guide-to-iot-

device-housings.html.
Viitattu 13.02.2025.

Siebeneicher, H. 2024. Bluetooth Low Energy | Arduino. Verkkojulkaisu.
https://docs.arduino.cc/learn/communication/bluetooth/.
Viitattu 12.02.2025.

The Codest n.d. Integroitu kehitysympäristö Ide. Verkkojulkaisu.
https://thecodest.co/fi/sanakirja/integroitu-kehitysymparisto-ide/.
Viitattu 23.02.2025.

The Curve 2024. What are WebSockets, and why are they essential for IoT? Verkkojulkaisu.
https://thecurve.io/what-are-websockets-and-why-they-are-essential-for-iot/.
Viitattu 12.02.2025.

Tilastokeskus 2016. Asumisen energiankulutus 2016. Verkkojulkaisu.
https://stat.fi/til/asen/2016/asen_2016_2017-11-17_tie_001_fi.html.
Viitattu 2.02.2025.

Tilastokeskus 2021. Liitetaulukko 2. Rakennukset lämmitysaineen mukaan 1970-2020.
Verkkojulkaisu.
https://stat.fi/til/rakke/2020/rakke_2020_2021-05-27_tau_002_fi.html.
Viitattu 11.02.2025.

Triopak Oy 2025. RGB Led, 5050 digitaalisesti ohjattava 5V / WS2812B. Verkkojulkaisu.
https://www.triopak.fi/fi/tuote/WS2812B.
Viitattu 15.03.2025.

WEMOS n.d. LOLIN D1 mini - WEMOS Documentation Verkkojulkaisu.
https://www.wemos.cc/en/latest/d1/d1_mini.html.
Viitattu 15.03.2025.

Ympäristöministeriö 2018. Rakennuksen energiankulutuksen ja lämmitystehontarpeen laskenta.
Verkkojulkaisu.
https://www.edilex.fi/data/rakentamismaaraykset/Ohje_Rakennuksen_energiankulutuksen_ja_lammitystehontarpeen_laskenta_20122017_vain_korostukset.pdf.
Viitattu 12.02.2025.

Zambetti, N., Karl, S. & Jacob, H. 2024. Inter-Integrated Circuit (I2C) Protocol. Verkkojulkaisu.
https://docs.arduino.cc/learn/communication/wire/.
Viitattu 12.02.2024.