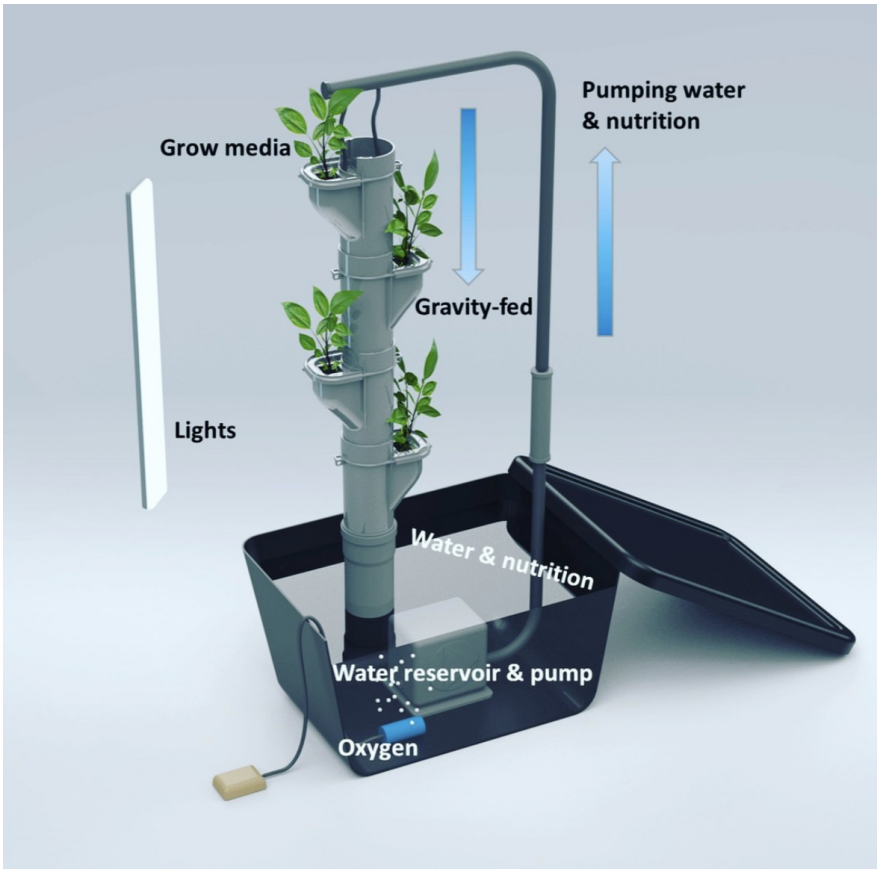
CHALMERS



Övervakningssystem för hydroponisk växt-odling med Zephyr RTOS

Examensarbete inom högskoleprogrammet Mekatronik

Alexander Olofsson

Julius Sjöstrand

INSTITUTIONEN FÖR ELEKTROTEKNIK

CHALMERS TEKNISKA HÖGSKOLA

Göteborg 2026

www.chalmers.se

Examensarbete 2026

Övervakningssystem för hydroponisk växtodling med Zephyr RTOS

Alexander Olofsson
Julius Sjöstrand

CHALMERS

Institutionen för Elektroteknik
Chalmers Tekniska Högskola
Göteborg 2026

Övervakningssystem för hydroponisk växtodling med Zephyr RTOS

Alexander Olofsson
Julius Sjöstrand

Handledare: Professor Bertil Thomas Institutionen för Elektroteknik
Examinator: Professor Bertil Thomas Institutionen för Elektroteknik
Handledare: Johan Ekberg Zellaco AB

Examensarbete 2026
Institutionen för Elektroteknik
Chalmers Tekniska Högskola
SE-412 96 Göteborg
Telefon +46 31 772 1000

Omslagsbild: GrowPipe Single Tower.

Skriven i LaTeX
Göteborg 2026

Sammanfattning

Hydroponisk odling är en odlingsmetod där växter odlas utan jord och där tillväxten påverkas av flera mätbara parametrar, exempelvis pH-värde, näringskoncentration, temperatur, luftfuktighet, ljusnivå och vattennivå. För att möjliggöra kontinuerlig övervakning av dessa parametrar har ett inbyggt sensorsystem utvecklats i detta examensarbete.

Syftet med arbetet var att utveckla, konfigurera och utvärdera ett prototypsystem för mätning och övervakning av ett hydroponiskt växtodlingssystem. Systemet baserades på en ESP32 med Zephyr RTOS och integrerades med sensorer för mätning av lufttemperatur, luftfuktighet, vattentemperatur, vattennivå, pH, konduktivitet och ljusnivå. Sensordata skickades via MQTT till ThingsBoard, där värdena kunde visualiseras i en dashboard och användas som grund för larmfunktioner.

Arbetet genomfördes iterativt där varje sensor först testades separat och därefter integrerades i ett gemensamt system. Systemet verifierades genom enhetstestning, integrationstestning och systemtestning i driftliknande miljö. Resultatet visar att ett hydroponiskt odlingssystem kan övervakas med hjälp av ett inbyggt sensorsystem och en IoT-plattform.

Projektet resulterade i en funktionell prototyp för datainsamling, visualisering och larmhantering. Systemet är dock inte långtidsverifierat över flera odlingscykler och bör därför ses som en teknisk prototyp snarare än en kommersiell färdig lösning. Vidare utveckling kan omfatta förbättrad kalibrering, längre drifttester och implementering av automatisk styrning av aktuatorer.

Abstract

Hydroponic cultivation is a cultivation method in which plants are grown without soil, and where plant growth is affected by several measurable parameters, such as pH, nutrient concentration, temperature, humidity, light level and water level. To enable continuous monitoring of these parameters, an embedded sensor system was developed in this thesis.

The purpose of the project was to develop, configure and evaluate a prototype system for measuring and monitoring a hydroponic cultivation system. The system was based on an ESP32 with Zephyr RTOS and was integrated with sensors for measuring air temperature, humidity, water temperature, water level, pH, electrical conductivity and light level. Sensor data was transmitted via MQTT to ThingsBoard, where the values could be visualized in a dashboard and used as a basis for alarm functions.

The work was carried out iteratively, where each sensor was first tested separately and then integrated into a complete system. The system was verified through unit testing, integration testing and system testing in an environment similar to the intended operating conditions. The results show that a hydroponic cultivation system can be monitored using an embedded sensor system and an IoT platform.

The project resulted in a functional prototype for data collection, visualization and alarm handling. However, the system has not been long-term verified over several growing cycles and should therefore be regarded as a technical prototype rather than a finished commercial product. Further development may include improved calibration, longer operational testing and implementation of automatic actuator control.

Förord

Detta examensarbete har utförts inom högskoleingenjörsprogrammet Mekatronik vid Chalmers tekniska högskola under vårterminen 2026. Utbildningen omfattar 180 högskolepoäng och examensarbetet omfattar 15 högskolepoäng. Arbetet har genomförts i samarbete med Zellaco AB.

Vi vill rikta ett stort tack till vår handledare på Zellaco, Johan Ekberg, för stöd, vägledning och teknisk återkoppling under arbetets gång. Vi vill även tacka Professor Bertil Thomas, vår examinator och handledare vid Chalmers tekniska högskola för värdefulla synpunkter, återkoppling och vägledning.

Ett särskilt tack riktas även till doktoranden Jelka Feldhusen, som hjälpte oss med utrustning och vägledning vid kalibrering av projektets sensorer.

Alexander Olofsson och Julius Sjöstrand
Göteborg, juni 2026

Terminologi och förkortningar

Nedan följer en lista över terminologi och förkortningar som används i rapporten, listade i alfabetisk ordning:

ADC	Analog-to-Digital Converter. Omvandlare som konverterar en analog spänning till ett digitalt värde som mikrokontrollern kan behandla.
DHCP	Dynamic Host Configuration Protocol. Nätverksprotokoll som automatiskt tilldelar IP-adress och andra nätverksinställningar till en enhet.
DNS	Domain Name System. System som översätter domännamn till IP-adresser.
EC	Electrical Conductivity. Elektrisk konduktivitet. Används för att uppskatta mängden lösta joner i näringslösningen.
ESP32	Mikrokontroller med inbyggt stöd för bland annat Wi-Fi och Bluetooth. Används som central styrenhet i systemet.
GPIO	General Purpose Input/Output. Digitala in- och utgångar på en mikrokontroller som kan användas för att läsa signaler eller styra externa komponenter.
I/O	Input/Output. In- och utgångar som används för att ta emot eller skicka signaler.
I2C	Inter-Integrated Circuit. Seriellt kommunikationsprotokoll som använder två signalledningar, SDA och SCL.
IoT	Internet of Things. Nätverk av fysiska enheter som kan samla in, skicka och ta emot data via internet.
IP	Internet Protocol. Protokoll som används för adressering och kommunikation i nätverk.
JSON	JavaScript Object Notation. Textbaserat dataformat som används för att strukturera och skicka data mellan system.
MQTT	Message Queuing Telemetry Transport. Kommunikationsprotokoll för IoT-system, baserat på publicering och prenumeration.
pH	Potential of Hydrogen. Mått på hur sur eller basisk en lösning är.
PPM	Parts Per Million. Enhet som anger koncentration, ofta använd för TDS-värden.
RH	Relative Humidity. Relativ luftfuktighet, angiven i procent.
RTOS	Real-Time Operating System. Realtidsoperativsystem som används för tidsstyrda och resursbegränsade inbyggda system.
TCP	Transmission Control Protocol. Transportprotokoll som säkerställer tillförlitlig dataöverföring mellan två enheter.

TDS	Total Dissolved Solids. Uppskattad mängd lösta fasta ämnen i vatten, ofta angiven i ppm eller mg/L.
VPD	Vapour Pressure Deficit. Skillnaden mellan luftens mättnadsångtryck och det aktuella vattenångtrycket. Används för att beskriva drivkraften för transpiration.
Wi-Fi	Trådlös nätverksteknik som används för att ansluta ESP32 till internet.
Zephyr RTOS	Ett öppet realtidsoperativsystem för inbyggda system.

Nomenklatur och symboler

Nedan följer en lista över symboler och variabler som används i rapportens ekvationer och tekniska beskrivningar.

Odlings- och mätparametrar

pH	Mått på hur sur eller basisk en lösning är.
EC	Elektrisk konduktivitet. Används för att uppskatta mängden lösta joner i näringslösningen.
$EC_{25°C}$	Elektrisk konduktivitet temperaturkompenserad till 25 °C.
TDS	Total Dissolved Solids. Uppskattad mängd lösta fasta ämnen i vatten.
VPD	Vapour Pressure Deficit. Ångtrycksunderskott som beskriver drivkraften för transpiration.
RH	Relativ luftfuktighet, angiven i procent.
T	Temperatur, angiven i grader Celsius.
LUX	Belysningsstyrka, angiven i lux.
ADC	Analog-to-Digital Converter. Digitalt råvärde från en analog insignal.

Variabler i ekvationer

k	Konversionsfaktor mellan EC och TDS.
ρ_{max}	Luftens mättnadsångtryck vid en given temperatur, angivet i kPa.
ρ_{luft}	Aktuellt vattenångtryck i luften, angivet i kPa.
$EC_{\text{lösning}}$	Uppskattad elektrisk konduktivitet efter utspädning.
EC_{stam}	Elektrisk konduktivitet hos stamlösningen före utspädning.
V_{stam}	Volym stamlösning som används vid utspädning.
V_{total}	Total volym efter utspädning.
t	Tid eller tidsintervall, beroende på sammanhang.

Innehåll

Figurer

Tabeller

1

Inledning

Detta kapitel presenterar bakgrunden till examensarbetet, arbetets syfte, avgränsningar samt de frågeställningar som ligger till grund för projektets genomförande.

1.1 Bakgrund

Hydroponisk odling är en växande teknik inom hållbart jordbruk där växter odlas utan jord, med rötterna direkt i en näringslösning som cirkuleras genom systemet. För att uppnå optimal tillväxt krävs noggrann kontroll av vattenflöde, pH-värde, näringsnivåer, temperatur och ljusförhållanden. Genom att integrera inbyggda system, givare och aktuatorer kan man skapa ett intelligent styrsystem som automatiskt reglerar dessa parametrar.

Zellaco utvecklar tekniska lösningar för styr- och övervakningssystem och har identifierat ett behov av ett flexibelt och skalbart inbyggt system för hydroponisk växtodling. Företaget efterfrågar därför ett system som kan samla in mätdata från sensorer samt visualisera och logga data via en molnbaserad plattform. Ett sådant system kan utgöra grund för vidare produktutveckling och kommersiella tillämpningar inom automatiserad odling.

1.2 Syfte

Syftet med examensarbetet är att utveckla, konfigurera och utvärdera ett inbyggt system för mätning och övervakning av ett hydroponiskt växtodlingssystem. Systemet ska samla in sensordata, logga data till ThingsBoard och kunna larma vid onormala förhållanden. Systemet ska kunna demonstreras på ett verkligt odlingssystem, exempelvis GrowPipe Single Tower.

1.3 Avgränsningar

Examensarbetet avgränsas till att omfatta grundläggande mätning och övervakning av ett hydroponiskt system. Följande ingår inte i arbetet:

- Kommersiell produktdesign eller certifiering
- Avancerad maskininlärning eller prediktiva modeller
- Långtidstester av systemets driftsäkerhet över flera odlingscykler
- Fullskalig industriell implementation

1.4 Precisering av frågeställningen

Utifrån syftet preciseras examensarbetet genom följande frågeställningar:

- Hur kan ett inbyggt system baserat på Zephyr RTOS utformas för att på ett tillförlitligt sätt mäta och övervaka en hydroponisk odling?
- Hur kan sensordata samlas in, behandlas och skickas vidare till en molnbaserad plattform för visualisering och loggning?
- Hur kan larmfunktioner implementeras för att indikera avvikelser från fördefinierade gränsvärden?
- Hur väl uppfyller det utvecklade systemet de funktionella krav som ställs av uppdragsgivaren?

2

Teori och teknisk bakgrund

I detta kapitel presenteras både odlingsrelaterad fakta och relevanta tekniska delar såsom sensorer, kommunikationsprotokoll, realtidsoperativsystem och IoT-plattform.

2.1 Hydroponisk odling

I detta avsnitt presenteras en översikt över hydroponisk odling, dess bakomliggande principer och de faktorer som påverkar växternas tillväxt.

2.1.1 Definition och princip

Hydroponik är en odlingsmetod där växter odlas utan jord. Till skillnad från jordodling där jorden fungerar som både stödstruktur och näringsreservoar använder hydroponiska system särskilda konstruktioner för att hålla växterna på plats [1]. I stället för att ta upp näring från jorden förses växterna med en vattenbaserad näringslösning som innehåller de makro- och mikronäringsämnen som krävs för deras tillväxt [1, 2]. Hydroponiska system medger en högre grad av kontroll över odlingsmiljön än traditionella jordodlingar [2]. Denna kontroll kan främja optimal tillväxt och därmed bidra till snabbare utveckling och högre avkastning [2].

2.1.2 Typer av hydroponiska system

Hydroponiska odlingssystem kan delas in i två bevattningsmetoder:

- Underbevattningssystem är system där växternas rötter är kontinuerligt nedsänkta i en näringslösning, exempelvis i system som Deep Water Culture (DWC) och Ebb-and-Flow [3]. I enklare system sker ingen aktiv cirkulation av näringslösningen utan den är statisk och byts ut eller justeras periodiskt. I mer avancerade varianter såsom recirkulerande djupvattensystem (RDWC), pumpas näringslösningen kontinuerligt mellan flera behållare. Detta skapar en cirkulation som bidrar till en jämn fördelning av näringsämnen, temperatur och syrehalt i hela systemet [3].

- Toppbevattningssystem är system där näringslösningen tillförs ovanifrån genom att pumpas från en reservoar genom rör till droppmunstycken som placerats vid varje växt. Därefter rinner lösningen tillbaka till reservoaren för återcirkulation. På så sätt får rötterna kontinuerligt tillgång till näring och syre [3]. Mer avancerade varianter bygger på samma grundprincip men innebär att flera behållare kopplas samman och att näringslösningen pumpas mellan dessa. På så sätt kan lösningen cirkulera genom systemet, vilket bidrar till en jämnare fördelning av näring och vatten mellan växterna [3].

2.1.3 Påverkande faktorer för tillväxt

Tillväxten i hydroponiska system styrs av flera fysikaliska och kemiska parametrar. Nedan beskrivs de mest centrala faktorerna.

2.1.3.1 Växtnäring

Växternas näringsbehov varierar beroende på utvecklingsstadium. Under den vegetativa fasen (fasen där utveckling av blad, stam och rötter utan att blomma) är behovet av kväve högre för att stödja blad- och stamtillväxt, medan behovet av fosfor och kalium är relativt lågt. I den reproduktiva fasen (fasen där växten blommar och bildar frön eller frukter) ökar i stället kraven på fosfor och kalium eftersom dessa näringsämnen har stor betydelse för energitransport och fruktbildning. Samtidigt minskar behovet av kväve, eftersom ett överskott kan leda till fortsatt bladproduktion i stället för blomning och fruktsättning. Anpassning av näringslösningens sammansättning efter dessa olika stadier är därför viktig för att optimera tillväxten och uppnå så hög avkastning som möjligt [4].

För att mäta mängden näringsämnen i näringslösningen används parametrarna elektrisk konduktivitet (Electrical Conductivity, EC) och totalt lösta fasta ämnen (Total Dissolved Solids, TDS). EC mäter vattnets förmåga att leda elektrisk ström, vilket direkt påverkas av koncentrationen av lösta joner såsom kalium, kalcium, magnesium och nitrater i näringslösningen [5], [6]. Eftersom dessa joner bär elektrisk laddning ökar EC-värdet när jonkoncentrationen stiger

TDS anger den totala mängden upplösta fasta ämnen i vatten och uttrycks vanligtvis i milligram per liter (mg/L), vilket motsvarar parts per million (ppm). TDS omfattar både organiska och oorganiska ämnen som är lösta i lösningen och dessa bidrar till vattenkvalitetens egenskaper [7]. Eftersom de flesta lösta mineraler i en hydroponisk näringslösning är joniserade kan TDS approximativt beräknas från EC vid 25 °C med hjälp av en konversionsfaktor (k) enligt ekvation (1).

$$\mathrm{TDS} = k \cdot \mathrm{EC}_{25^\circ\mathrm{C}} \tag{1}$$

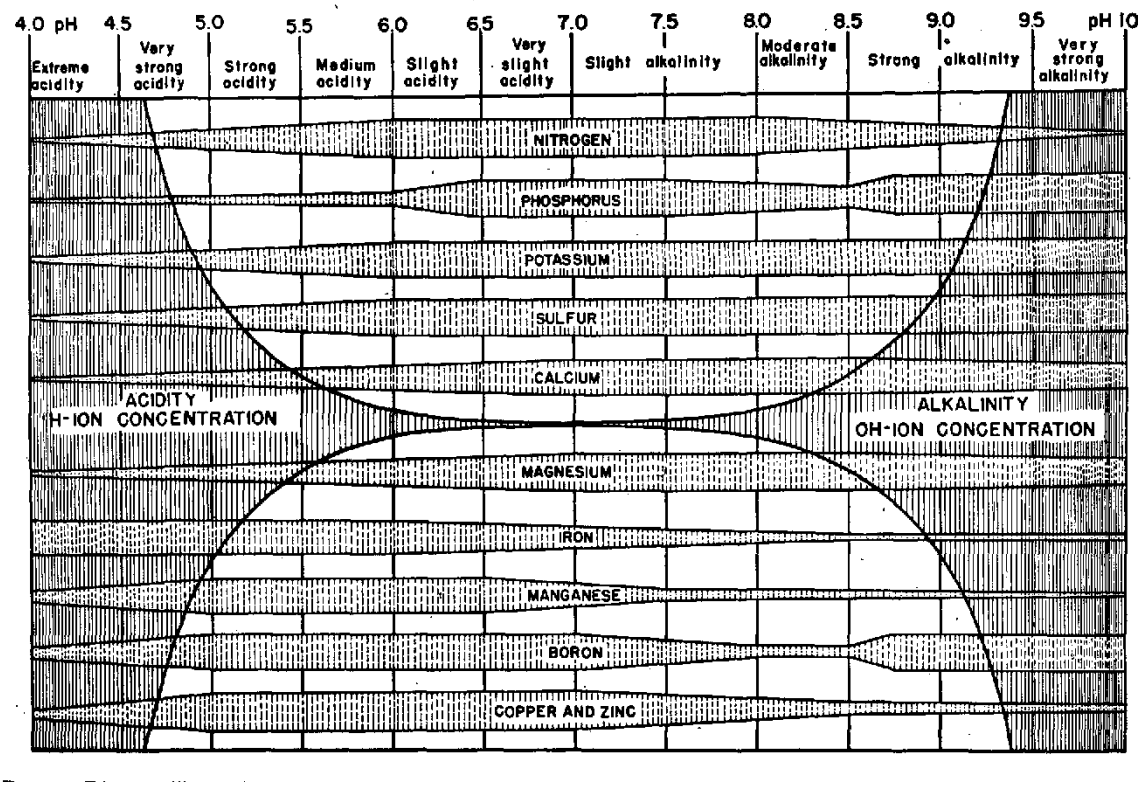
där (k) vanligen ligger i intervallet 0.55–0.8 beroende på näringsämnenas sammansättning och konverteringsmetod [8].

För höga EC-värden kan skada växtens rötter och försvåra upptaget av vatten, vilket hämmar tillväxten. För låga EC-värden innebär istället att växten får för lite näring, vilket kan leda till näringsbrist och sämre tillväxt [5, 6]. Rekommenderade EC-nivåer varierar beroende på växtart och odlingsstadium. Bladgrönsaker trivs generellt vid lägre EC-nivåer, omkring 0,8–1,8 mS/cm. Örter behöver ofta något högre nivåer kring 1,0–1,6 mS/cm medan grödor som tomat och paprika kan kräva högre EC-nivåer från 2 mS/cm upp till 5 mS/cm [9], [4]. Rekommenderade EC-värden återfinns i tabellerna i Bilaga A.

2.1.3.2 pH

pH (Potential of Hydrogen) är ett mått på hur surt eller basiskt ett ämne är. pH anger koncentrationen av vätejoner (H^+) i en lösning, desto fler vätejoner desto surare är lösningen. pH mäts på en logaritmisk skala från 0 till 14, där pH 7 är neutralt, värden under 7 är sura och värden över 7 är basiska [4].

pH-värdets betydelse för näringsupptagning illustreras i figur 1. Diagrammet visar hur upptagningsförmågan av olika näringsämnen varierar med pH. I diagrammet representeras varje näringsämne av en linje där bredden indikerar hur absorberbar ämnet är för växten vid ett visst pH-värde.



Figur 1: Tillgänglighet av växtnäringsämnen som funktion av pH (E. Truog, 1947 [10]).

Diagrammet visar att makronäringsämnen såsom kväve, fosfor och kalium har god absorption inom ett basiskt till svagt surt intervall medan mikronäringsämnena järn, magnesium och zink har god absorption inom ett neutralt till surt intervall [10]. Utifrån diagrammet kan man utläsa att det finns ett begränsat pH-intervall där växten samtidigt kan absorbera flera näringsämnen optimalt, vilket uppskattningsvis ligger mellan pH 5,5 och 6,5 [4]. Om pH avviker från detta intervall kan växten ha svårare att ta upp de näringsämnen som finns i lösningen. Vid lågt pH (<5,5) ökar lösligheten av metaller vilket kan leda till toxicitet, medan ett högt pH (>7,5) minskar tillgängligheten av viktiga mikronäringsämnen och därmed kan orsaka näringsbrist och fysiologisk stress hos växterna [4]. För optimala pH-värden för specifika växter hänvisas till tabellerna i Bilaga A.

2.1.3.3 Vatten- och lufttemperatur

Lufttemperaturen påverkar växternas tillväxt genom att styra fotosyntesens effektivitet och växtens energibalans. När temperaturen ligger inom ett optimalt intervall gynnas fotosyntesen, vilket främjar tillväxten. Vid för höga temperaturer ökar växtens energiförbrukning, vilket innebär att en mindre del av energin kan avsättas för tillväxt. Detta kan även leda till att bladen torkar ut samt att blomning och fruktsättning försämras [11]. Om temperaturen sjunker för lågt hämmas fotosyntesen, vilket reducerar tillväxthastigheten. Vid mycket låga temperaturer kan fotosyntesen upphöra, vilket kan leda till att tillväxten avstannar och att växten skadas om temperaturen är låg under en längre tid [11]. Skillnader mellan dag- och nattemperatur är också viktiga, eftersom för höga natttemperaturer kan påverka växtens energibalans negativt och minska tillväxten. Därför hålls dagstemperaturen ofta något högre än nattemperaturen. Ett lämpligt intervall är cirka 25–28 °C under dagen och 18–22 °C under natten [11].

Temperaturen i näringslösningen är en viktig faktor eftersom den påverkar växternas näringsupptag och syretillgång. När vattentemperaturen ligger inom ett optimalt intervall kan rötterna effektivt absorbera näringsämnen och syre, vilket främjar en stabil tillväxt [12]. Om vattnet blir för varmt minskar syrehalten [12], vilket kan leda till syrebrist runt rötterna. Samtidigt kan transporten av näringsämnen försämras och växten reagerar med en stressrespons där energi omfördelas från tillväxt till skyddande processer, vilket hämmar utvecklingen [12]. Om vattnet i stället är för kallt sänks rotens ämnesomsättning och näringsupptag, vilket också begränsar tillväxten. Låga temperaturer kan dessutom hämma rotutvecklingen och påverka de metaboliska processerna negativt, vilket ytterligare reducerar växtens tillväxt. Ett lämpligt temperaturintervall att utgå från är cirka 18–27 °C [12].

2.1.3.4 Luftfuktighet och Vapor Pressure Deficit (VPD)

För att växten ska kunna transportera näringsämnen från rötterna till sina olika delar måste vatten ständigt röra sig upp genom växten. Denna rörelse drivs av det undertryck som uppstår när vatten avdunstar från bladen, en process som kallas transpiration [13].

Luftfuktigheten i omgivningen påverkar denna process direkt. Relativ luftfuktighet (Relative Humidity, RH) är ett mått på hur mycket vattenånga luften innehåller i förhållande till dess maximala kapacitet vid en given temperatur. Genom att mäta RH kan man uppskatta hur mycket vatten växten potentiellt kan avdunsta. Vid hög RH är luften närmare mättad med vattenånga vilket minskar transpirationen, medan vid låg RH är luften torrare vilket ökar transpirationen [13]. Att kontrollera RH är därför viktigt för att undvika både vattenstress och försämrad transport av näringsämnen.

För att beskriva drivkraften bakom transpirationen används ångtrycksunderskott (Vapour Pressure Deficit, VPD) vilket är skillnaden mellan luftens mättnadsångtryck (maximalt vatten som luften kan hålla, ρ_{max}) och det faktiska vattenångtrycket i luften (ρ_{luft}). Denna skillnad skapar ett undertryck vid bladytan, vilket fungerar som drivkraften för transpiration och kan beräknas enligt följande [14]:

$$VPD = \rho_{\max} - \rho_{\text{luft}} \tag{2}$$

där:

$$\rho_{max} = 0.6108 \cdot e^{\frac{17.27 \cdot T}{T+237.3}} \tag{3}$$

och:

$$\rho_{\text{luft}} = \rho_{\max} \cdot \frac{RH}{100} \tag{4}$$

vilket ger:

$$VPD = 0.6108 \cdot e^{\frac{17.27 \cdot T}{T+237.3}} \cdot \left(1 - \frac{RH}{100}\right) \tag{5}$$

Där T är bladets temperatur i grader Celsius (°C), RH är den relativa luftfuktigheten i procent (%), $\rho_{\max}$ är mättnadsångtrycket vid en given temperatur i kilopascal (kPa), och ρ_{luft} är lufttrycket i kilopascal (kPa).

I kontrollerade odlingsmiljöer är skillnaden mellan bladets temperatur och lufttemperaturen ofta mycket liten, vanligtvis ±1–2 °C. Därför kan bladtemperaturen approximativt antas vara lika med lufttemperaturen vid beräkning av VPD [14]. Det optimala VPD-värdet varierar beroende på växtens utvecklingsstadium. Under tidiga stadier, såsom groning (när ett frö börjar växa) och vegetativ tillväxt, rekommenderas ett lägre VPD på cirka 0,3–0,6 kPa. Under senare stadier, såsom blomning och fruktsättning, är ett VPD på 0,8–1,2 kPa ofta mer lämpligt, eftersom det främjar transpiration och näringsupptag samt minskar risken för kondensrelaterade sjukdomar [13], [14].

2.1.3.5 Ljus och fotoperiod

Ljusintensitet definieras som mängden ljusenergi som når växtens yta per tidsenhet, ofta mätt som fotosyntetiskt aktiv strålning (Photosynthetically Active Radiation, PAR) [15]. Ökad ljusintensitet stimulerar fotosyntesen och därmed tillväxten fram till en mättnadspunkt [15]. Vid intensiteter över mättnadspunkten ger ytterligare ljus ingen ökning av fotosynteshastighet utan kan istället orsaka fotoinhibering (nedsatt fotosyntes till följd av starkt ljus). Detta skadar de pigment och ljussystem i växten som omvandlar ljus till energi [15]. Låg ljusintensitet begränsar energitillgången, vilket resulterar i reducerad biomassa, långsammare bladutveckling och lägre avkastning [15].

Ljuskvalitet definieras som ljusets spektrala sammansättning, det vill säga vilka våglängder som finns i det ljus som växterna exponeras för. Klorofyll a och b, de främsta fotosyntetiska pigmenten hos växter, absorberar ljus i specifika våglängder.

Detta gör att vissa våglängder bidrar mer effektivt till fotosyntesen än andra. Klorofyll absorberar främst blått ljus (cirka 400–500 nm) och rött ljus (cirka 600–700 nm), medan ljus inom andra delar av spektrumet såsom grönt ljus reflekteras i större utsträckning vilket ger växterna deras gröna färg. Blått ljus främjar växtens strukturella egenskaper såsom bladtjocklek och stomatafunktion (klyvöppningarnas öppning och stängning för gasutbyte och vattenreglering). Rött ljus främjar istället tillväxt och blomning. Genom att kombinera rött och blått ljus skapas ett spektrum som utnyttjar klorofylls ljusabsorptionsförmåga och därigenom stärker fotosyntes och tillväxten [16].

Ljusduration (fotoperiod) hänvisar till den totala tiden växter exponeras för ljus under ett dygn. Ljuset påverkar växternas naturliga cykel, vilket styr hur de växer och blommar. Växter delas i regel in i långdagsväxter, kortdagsväxter eller dagneutrala beroende på hur deras blomning och fysiologiska processer reagerar på ljusperiodens längd [17], [18]. Långdagsväxter har utvecklats i områden där dagarna är långa under växtsäsongen och de blommar därför när daglängden överstiger dess artspecifika gränsvärde. Kortdagsväxter är anpassade till miljöer där blomning sker när dagarna blir kortare och de blommar därför när ljusperioden understiger dess artspecifika gränsvärde. Dagneutrala växter påverkas inte av daglängden när det gäller blomning, utan blommar oberoende av hur lång dagen är. Däremot påverkar ljuset fortfarande deras fotosyntes och tillväxt [17], [18].

2.2 Beskrivning av elektriska komponenter

I detta avsnitt presenteras vilka elektriska komponenter som används.

2.2.1 Mikrokontroller

ESP32U är en mikrokontroller baserad på 32-bitars processorkärnor med inbyggt stöd för Wi-Fi och Bluetooth. Det inbyggda Wi-Fi-stödet är begränsat till 2,4 GHz-bandet och saknar stöd för 5 GHz-bandet. Detta innebär att mikrokontrollern endast kan ansluta till trådlösa nätverk som är tillgängliga på 2,4 GHz [19]. Dessutom är enheten utrustad med stöd för flera kommunikationsprotokoll såsom I^2C, SPI och UART, vilket möjliggör kommunikation med externa enheter [19]. Genom mikrokontrollerns digitala och analoga I/O-portar kan externa enheter anslutas, vilket möjliggör både avläsning och styrning. Detta gör ESP32U lämplig för inbyggda system och Internet of Things (IoT)-applikationer [19].

2.2.2 TDS-sensorn

En TDS-sensor (Total Dissolved Solids) fungerar genom att mäta vattnets elektriska ledningsförmåga med hjälp av två elektroder som sänks ned i vätskan [20, 21]. När en växelspänning appliceras mellan elektroderna skapas ett elektriskt fält i vattnet som får de laddade jonerna att röra sig, vilket ger upphov till en ström som är proportionell mot jonkoncentrationen och därmed mot den elektriska konduktiviteten (EC) [20], [21]. Den analoga signalen kan omvandlas till ett EC vilket därefter kan omvandlas till ett TDS-värde genom ekvation (1).

2.2.3 Ultraljudssensor

Ultraljudssensor A02YYUW är en sensor som använder ultraljud (ljudvågor över 20 kHz) för att mäta avståndet till ett objekt. Den fungerar genom att avge en ultraljudspuls och vänta på returekot. Avståndet beräknas internt av sensorn med ekvation (6) [22].

$$Avstånd = \frac{Tid \cdot \text{Ljudets hastighet}}{2} \tag{6}$$

Divisionen med två görs eftersom ljudpulsen färdas både från sensorn till objektet och tillbaka till sensorn. Den uppmätta tiden motsvarar därför ljudets totala färdväg fram och tillbaka, medan avståndet som ska beräknas endast är avståndet mellan sensorn och objektet.

Ultraljudssensorn har två kommunikationsledningar: TX (Transmit), som används för att skicka data och RX (Receive), som används för att ta emot data. Genom dessa anslutningar kan sensorn kommunicera seriellt och överföra det uppmätta avståndet digitalt via kommunikationsprotokollet UART (Universal Asynchronous Receiver/Transmitter) [22]. A02YYUW är även vattentålig och kan mäta mellan områdena 3 till 450 cm.

2.2.4 Ljussensor

Ljussensorn TSL2591 är en digital ljussensor för mätning av belysningsstyrka (lux) [23]. Sensorn mäter ljus genom två interna fotodioder. Den ena fotodioden mäter både synligt och infrarött ljus medan den andra huvudsakligen mäter infrarött ljus. Genom att jämföra signalerna från dessa två fotodioder kan sensorn kompensera för påverkan från infrarött ljus och därmed ge en mer noggrann uppskattning av den omgivande ljusintensiteten [24].

TSL2591 använder interna 16-bitars ADC-enheter (Analog-to-Digital Converters) för att omvandla de analoga signalerna från fotodioderna till digitala värden. Dessa digitala värden används sedan internt för att beräkna luxvärdet enligt ekvation 7 [24].

$$LUX = \frac{(Fullspektrum - Infraröd) \cdot Gain}{Integrationstid} \tag{7}$$

Fullspektrum representerar det totala uppmätta ljuset, Infraröd den infraröda komponenten, Gain den valda förstärkningen och Integrationstid sensorns integrationstid. Sensorn kommunicerar via I^2C-gränssnittet och använder på I^2C-bussen standardadressen 0x29 [24]. Sensorn kan mäta mellan 188μLux upp till 88 000 Lux [24].

2.2.5 pH-sensor

pH-sensorn mäter surhetsgraden i en vätska genom en elektrokemisk elektrod som genererar en spänning proportionell mot koncentrationen av vätejoner [25]. Denna elektrod är kopplad till en analog signalbehandlingsmodul som förstärker den svaga signalen och omvandlar den till en analog spänning (0–5 V) [25]. För att sensorn ska ge korrekta värden krävs kalibrering med standardlösningar med kända pH-värden där mätfelet justeras [25].

2.2.6 Temperatur- och luftfuktighetssensor

SHT30 är en digital temperatur- och luftfuktighetssensor som använder sig av I^2C-gränssnittet för kommunikation. Sensorn har en fast I^2C-adress på 0x44 (alternativt 0x45 beroende på adresskonfiguration) [26].

- **Luftfuktighet:** Sensorn kan mäta mellan 0–100 % RH [26] och använder sig av ett fuktkänsligt dielektriskt material som fungerar som en kondensator. När luftfuktigheten ökar absorberar materialet vattenmolekyler från omgivningen vilket får kapacitansen att öka. Kapacitansförändringen omvandlas till en elektrisk signal som skickas vidare via I^2C-gränssnittet [26].
- **Temperatur:** Sensorn kan mäta mellan $-40\,^{\circ}\mathrm{C}$ till $125\,^{\circ}\mathrm{C}$ [26] och använder sig av ett resistivt temperaturkänsligt material där materialets elektriska motstånd förändras med temperaturen. När temperaturen i omgivningen stiger ökar motståndet i materialet och när temperaturen sjunker minskar motståndet. Denna förändring omvandlas till en elektrisk signal som skickas vidare via I^2C-gränssnittet [26].

2.2.7 Vattentemperatursensor

Temperatursensorn består av DS18B20-chipet som är själva temperaturgivaren. Chipet sitter inne i ett vattentätt metallhölje som leder värme från vätskan till sensorn. Sensorn mäter temperaturen med silicon bandgap-teknik vilket innebär att chipet har halvledarkomponenter vars elektriska spänning förändras linjärt med temperaturen. Dessa förändringar omvandlas till en elektrisk signal som skickas vidare via 1-Wire-protokollet [27].

2.3 Kommunikationsprotokoll

I detta avsnitt presenteras de kommunikationsprotokoll som används.

2.3.1 I^2C

Inter-integrated circuit (I^2C) protokollet är ett seriellt kommunikationsprotokoll som används för dataöverföring mellan olika enheter [28]. Dataöverföringen sker via ett tvåtrådigt gränssnitt baserat på master-slave-arkitekturen [28].

Master-slave-arkitekturen fungerar genom att en masterenhet styr en eller flera slavenheter. Dessa är sammankopplade via två signalledningar: klocksignalen (Serial Clock Line, SCL) som genereras av masterenheten för att synkronisera dataöverföringen på bussen samt SDA (Serial Data Line) som används för dataöverföring i båda riktningarna mellan master och slav [28].

2.3.2 UART

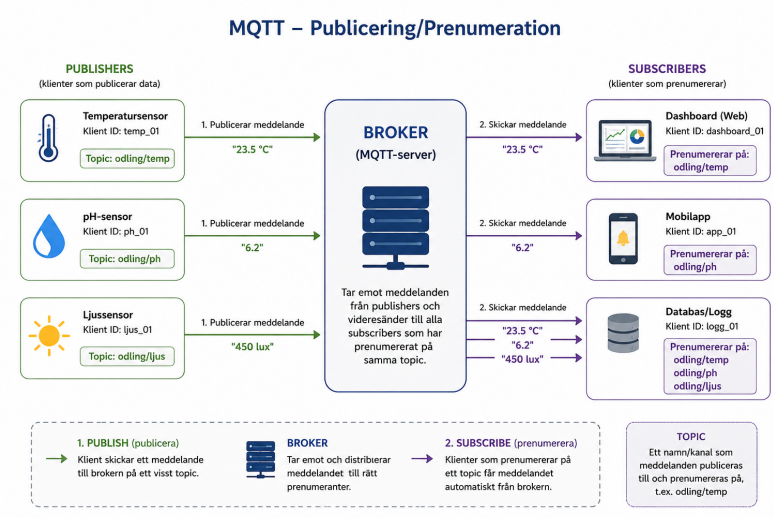
UART (Universal Asynchronous Receiver/Transmitter) är ett seriellt kommunikationsprotokoll som används för att överföra data mellan elektroniska enheter. Kommunikation sker genom två signalledningar: TX (Transmit), som skickar data och RX (Receive) som tar emot data. UART är asynkront vilket innebär att ingen gemensam klocksignal används mellan enheterna. I stället måste båda enheterna vara inställda på samma överföringshastighet (baud rate) för att kommunikationen ska fungera korrekt [29].

2.3.3 1-Wire

1-Wire-protokollet är ett kommunikationssystem där en master, exempelvis en mikrokontroller, kommunicerar med en eller flera slav-enheter via en enda dataledning [30]. All kommunikation styrs av mastern som först skickar en reset-signal för att synkronisera bussen varpå anslutna enheter svarar med en presence pulse för att visa att de finns. Varje slav har en unik 64-bitars adress vilket gör det möjligt att ha flera enheter på samma ledning. Data överförs en bit i taget genom tidsstyrda signaler där längden på pulser avgör om en etta eller nolla skickas [30].

2.3.4 MQTT

Figur 2 visar Message Queuing Telemetry Transport (MQTT) som är ett kommunikationsprotokoll för IoT.



Figur 2: Dataflöde mellan publicerande klienter, MQTT-broker och prenumererande klienter.

MQTT bygger på en publicering/prenumerationsmodell [31]. Alla enheter som ansluter till systemet kallas för klienter och kommunicerar via en central server som kallas "broker". När en klient publicerar ett meddelande skickar den data till brokern under ett specifikt "topic". Alla klienter som har prenumererat på samma topic får automatiskt meddelandet från brokern [31]. På detta sätt kan sensorer, IoT-enheter, appar eller andra klienter både skicka och ta emot information utan att behöva kommunicera direkt med varandra.

2.3.5 DHCP

Dynamic Host Configuration Protocol (DHCP) är ett applikationslagerprotokoll som automatiskt tilldelar nödvändiga nätverksparametrar såsom IP-adress, subnätmask, standardgateway och DNS-servrar vilket eliminerar behovet av manuell tilldelning [32]. Tilldelningen sker från en fördefinierad adresspool vilket innebär att antalet tillgängliga IP-adresser är begränsat och styrs av nätverkets konfiguration [32].

Kommunikationen mellan klient och server i DHCP sker enligt en fyrastegsprocess benämnd DORA (Discover, Offer, Request, Acknowledge) [32]. Processen startar när en klient sänder ett broadcast-meddelande (DHCP Discover) för att lokalisera tillgängliga DHCP-servrar i nätverket [32]. En server svarar därefter med ett erbjudande (DHCP Offer) som innehåller en föreslagen IP-adress och tillhörande konfigurationsparametrar [32]. Klienten bekräftar sitt val genom en DHCP Request varefter servern slutligen skickar en DHCP Acknowledge som bekräftar tilldelningen [32]. En central del av DHCP är användningen av leasingtider vilket innebär att en IP-adress tilldelas temporärt [32]. När en betydande del av leasingtiden har passerat begär klienten en förnyelse av sin tilldelning vilket säkerställer att IP-adresser kan återanvändas inom nätverket [32].

2.3.6 DNS

Domain Name System (DNS) är ett system som möjliggör för användare att använda lättlästa domännamn i stället för numeriska IP-adresser [33]. När en användare anger ett domännamn, exempelvis i en webbläsare, initieras en DNS-uppslagning (en stegvis sökprocess) för att hitta motsvarande IP-adress [33]. Först kontrollerar klienten om svaret redan finns i en lokal cache [33]. Finns ingen information lagrad skickas en förfrågan till en DNS-resolver som ofta erhålls via DHCP från nätverkets router eller internetleverantör [33].

Om resolvern inte har svaret i sin cache påbörjas en hierarkisk DNS-uppslagning [33]. Den första kontakten sker med en rotserver, som inte känner till den exakta IP-adressen men kan peka vidare till rätt toppdomänserver (TLD-server), exempelvis för ".se" eller ".com" [33]. Toppdomänservern dirigerar sedan vidare till en auktoritativ namnserver som ansvarar för den specifika domänen [33]. Den auktoritativa namnservern innehåller normalt de faktiska DNS-posterna för domänen och återsänder den korrekta IP-adressen till resolvern [33]. Svaret skickas därefter tillbaka till klienten som kan använda IP-adressen för att ansluta till den önskade servern [33].

2.3.7 TCP

Transmission Control Protocol (TCP) är ett transportlagerprotokoll som används för att överföra data mellan två enheter i ett IP-baserat nätverk [34]. TCP är en av de centrala komponenterna i den så kallade TCP/IP-modellen och används i många nätverkstjänster, exempelvis webbläsarkommunikation och filöverföring [34].

När en anslutning ska upprättas mellan två enheter sker detta genom en process som kallas en trevägshandskakning (three-way handshake) [34]. Processen inleds med att klienten skickar ett SYN-meddelande för att begära en anslutning [34]. Servern svarar med ett SYN-ACK-meddelande som bekräftar begäran och samtidigt skickar en egen synkronisering [34]. Slutligen skickar klienten ett ACK-meddelande som bekräftar anslutningen [34]. När detta är klart är en stabil TCP-anslutning etablerad [34].

TCP säkerställer även att data levereras korrekt genom att dela upp information i segment som numreras och bekräftas av mottagaren [34]. Om ett segment förloras eller anländer felaktigt begärs en ny överföring [34]. Protokollet hanterar dessutom flödeskontroll, vilket innebär att mängden data som skickas anpassas efter mottagarens kapacitet [34].

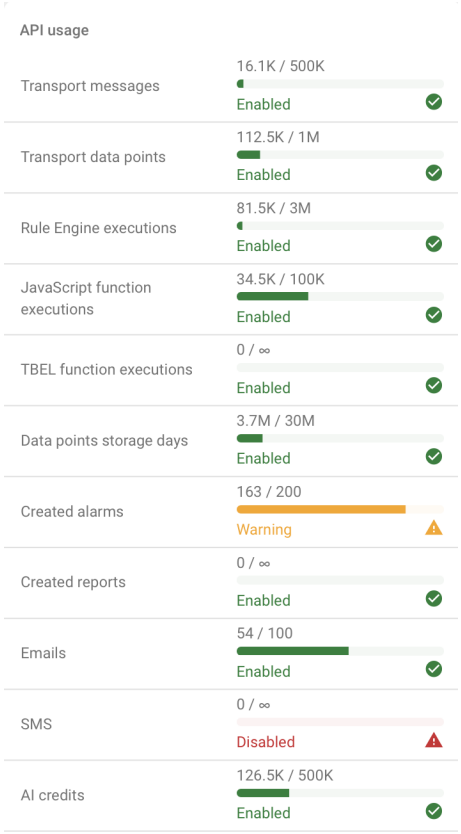
2.4 Zephyr

Zephyr är ett öppet operativsystem för inbyggda system klassificerat som ett realtidsoperativsystem (RTOS) och är avsett för tidskritiska tillämpningar i resursbegränsade enheter [35]. Det används bland annat i sensorer, IoT-enheter, wearables och industriella mikrokontrollersystem. Zephyr utvecklas inom Linux Foundation och har en modulär arkitektur som möjliggör anpassning efter specifika applikationskrav. Systemet stöder flera hårdvaruarkitekturer däribland ARM och RISC-V, samt innefattar stöd för kommunikationsprotokoll som Bluetooth och Wi-Fi [35].

2.5 ThingsBoard

ThingsBoard är en öppen plattform för Internet of Things (IoT) utformad för att hantera, visualisera och analysera data från uppkopplade enheter i realtid [36]. Plattformen används primärt för att samla in sensordata, bearbeta data i molnet samt skapa instrumentpaneler för övervakning och styrning. ThingsBoard utvecklades som ett open source-projekt med en skalbar arkitektur som kan användas både lokalt och i molnet. Plattformen har inbyggd funktionalitet för enhetsregistrering, regelbaserad datahantering och larmhantering. Den stöder även kommunikationsprotokoll som MQTT, CoAP och HTTP samt integration med databaser som PostgreSQL och Cassandra, vilket möjliggör hantering av stora datamängder från IoT-enheter [36].

ThingsBoard erbjuder olika prenumerationsnivåer med skilda användningsgränser för bland annat enheter, dashboards, rule chains, transporterade meddelanden, lagrade datapunkter, larm och e-postmeddelanden [36]. Gratisversionen har dokumenterade begränsningar för dessa resurser samt tidsbegränsad lagring, där telemetrydata och larm lagras i 30 dagar innan de automatiskt rensas, medan betalversionerna erbjuder högre gränser och längre lagringstid [37]. Plattformen erbjuder även en vy där användningen av dessa funktioner kan följas [36], vilket visas i Figur 3



Figur 3: Översikt över API-användning i ThingsBoard Cloud.

3

Metod

I detta kapitel beskrivs de metoder som användes för att planera, utveckla, testa och utvärdera systemet. Kapitlet behandlar även metodens begränsningar och hur dessa påverkar tolkningen av resultatet.

3.1 Arbetsmetodik

Arbetet bedrevs iterativt, där delsystem utvecklades och verifierades stegvis innan de integrerades i den färdiga lösningen. Detta arbetssätt valdes eftersom projektet omfattade flera tekniska områden, såsom sensorer, datakommunikation och visualisering i ThingsBoard.

Systemet delades in i tre huvuddelar: sensorer för datainsamling, ESP32 med Zephyr för behandling av mätvärderna och MQTT-kommunikation samt ThingsBoard för visualisering, loggning och larmhantering. Arbetet inleddes med en förstudie där hydroponiska system, sensorer, Zephyr och ThingsBoard studerades. Därefter valdes hårdvara utifrån mätområden, signaltyper och integrationsmöjligheter med ESP32. I samband med detta togs även en övergripande systemarkitektur fram.

Datainsamlingen bestod av sensorbaserade mätningar från det hydroponiska systemet, teknisk informationsinsamling från datablad och dokumentation för Zephyr samt ThingsBoard. Sensorerna användes för att övervaka vattenmiljön och den omgivande odlingsmiljön, medan dokumentationen låg till grund för korrekt koppling, konfigurering och implementation av systemet.

Implementeringen och testningen genomfördes stegvis. Först verifierades kommunikationen mellan dator och ESP32. Därefter testades varje sensor separat med egen programkod för att kontrollera att koppling och signalavläsning fungerade som förväntat. När en sensor fungerade korrekt integrerades den i huvudprogrammet. Detta möjliggjorde tidig identifiering av fel och förenklade felsökningen.

Efter de enskilda sensortesterna genomfördes integrationstester där flera sensorer användes samtidigt. Detta gjordes för att säkerställa att sensorerna fungerade tillsammans utan konflikter mellan kommunikationsgränssnitten eller datahantering. Mätvärdena jämfördes med förväntade värden, exempelvis från en termometer eller kalibreringslösningar.

När sensorerna hade verifierats integrerades kommunikationen med ThingsBoard. Sensordata strukturerades i ett format som kunde skickas via MQTT och visualiseras i en dashboard. Slutligen genomfördes systemtester där hela kedjan, från sensoravläsning till visualisering och larmhantering i ThingsBoard, verifierades.

3.2 Metodbegränsningar

Den valda metoden gav en strukturerad utvecklingsprocess och gjorde det möjligt att successivt bygga upp och verifiera systemet. Samtidigt finns det begränsningar som påverkar hur resultaten bör tolkas.

Testningen genomfördes i begränsad skala och systemet har inte långtidsverifierats över flera odlingscykler. Det innebär att slutsatser om långsiktig driftsäkerhet, sensorernas stabilitet över tid och systemets robusthet vid kontinuerlig drift inte kan fastställas med säkerhet. Detta är särskilt relevant eftersom hydroponiska system utsätter sensorer och elektronik för fukt och kondens, vilket kan orsaka korrosion, kortslutningar och försämrad mätnoggrannhet över tid. Därför behöver systemets elektronik skyddas och placeras på ett sätt som minimerar risken för exponering mot vätska och hög luftfuktighet.

En annan begränsning är att vissa sensorer har en noggrannhet som är starkt beroende av kalibrering och omgivningsförhållanden. Detta gäller särskilt analoga sensorer, där mätvärden kan påverkas av elektriskt brus, matningsspänning och temperatur.

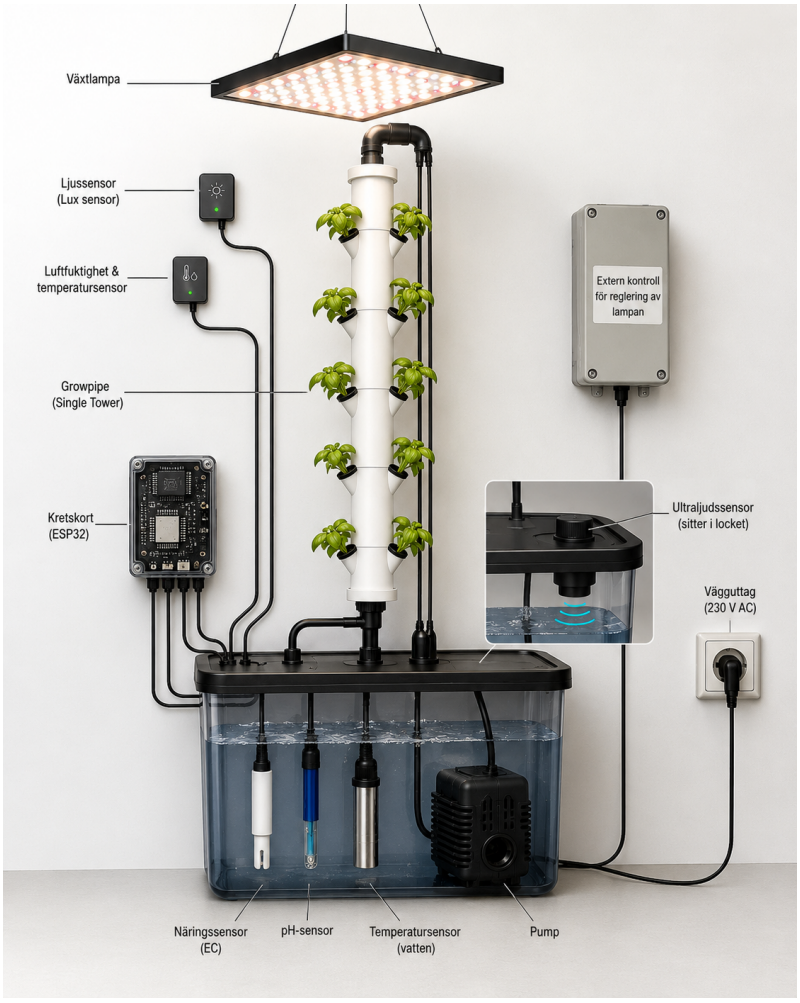
4

Genomförande

I detta kapitel beskrivs hur systemet implementerades i praktiken. Kapitlet omfattar sensorplacering, elektrisk koppling, testning och kalibrering av sensorer, mjukvaruimplementation i Zephyr samt integration med ThingsBoard för visualisering och larmhantering.

4.1 Sensorplacering

Sensorernas placering i det hydroponiska systemet valdes utifrån vilken parameter respektive sensor skulle mäta. Systemet omfattar både vattenrelaterade och luftrelaterade mätparametrar därför placerades sensorerna på olika delar av GrowPipes Single Tower, vilket illustreras i figur 4.



Figur 4: Placering av sensorer på GrowPipes Single Tower.

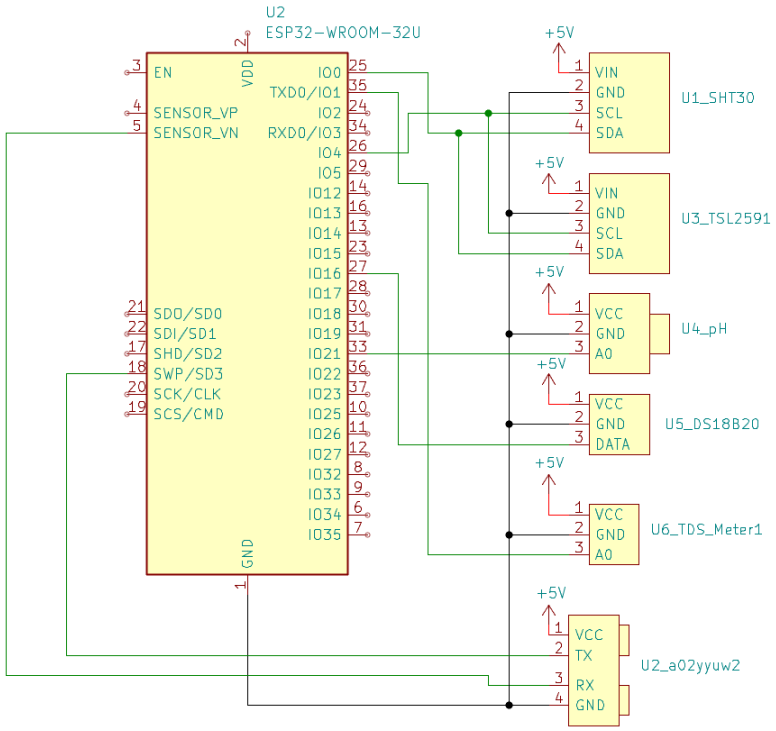
Lufttemperatur- och luftfuktighetssensorn placerades i anslutning till växterna för att mäta den odlingsmiljö som växterna exponeras för. Placeringen valdes så att sensorn inte skulle utsättas direkt för vattenstänk, samtidigt som den skulle vara tillräckligt nära växterna för att mäta den odlingsmiljön.

Vattentemperatursensorn placerades i näringslösningen för att mäta temperaturen i vattnet. pH- och TDS-sensorerna placerades också i näringslösningen eftersom dessa sensorer behöver vara nedsänkta i vätskan för att kunna mäta lösningens egenskaper. Avståndssensorn monterades på insidan av locket på vattenbehållaren och riktades ned mot vattenytan för att kunna uppskatta vattennivån.

Ljussensorn placerades i närheten av växternas bladnivå för att ge en uppskattning av den ljusexponering som växterna får. Placeringen är viktig eftersom mätvärdet påverkas av avståndet till ljuskällan, skuggning från växter och sensorns riktning.

4.2 Elektrisk koppling av sensorer

Figur 5 visar kopplingen mellan samtliga sensorer och mikrokontrollern. Mikrokontrollern fungerar som en central styrenhet som samlar in mätdata från olika sensorer via flera kommunikationsgränssnitt, beroende på respektive sensors tekniska krav.



Figur 5: Kopplingschema för samtliga sensorer.

Två av sensorerna är anslutna via ESP32:ns ADC-gränssnitt, där analoga signaler omvandlas till digitala värden. pH-sensorn är kopplad till GPIO33, motsvarande ADC1 kanal 5, där den analoga utsignalen representerar pH-värdet i lösningen.

TDS-sensorn är kopplad till GPIO35, motsvarande ADC1 kanal 7 och mäter den totala halten lösta ämnen i vattnet via en analog signal.

Temperatur- och fuktsensorn SHT30 kommunicerar i stället via I^2C-gränssnittet, vilket möjliggör digital kommunikation mellan sensorn och mikrokontrollern. I denna koppling används SDA på GPIO25 och SCL på GPIO26. Även ljussensorn TSL2591 är ansluten via samma I^2C-buss, där SDA ligger på GPIO25 och SCL på GPIO26. Det innebär att båda sensorerna delar samma kommunikationslinjer. Avståndssensorn är ansluten via UART-gränssnittet (UART2), där kommunikationen sker via GPIO18 (TX) och GPIO5 (RX). Kopplingen följer standarden för asynkron seriell kommunikation, vilket innebär korskoppling mellan sensorns och mikrokontrollerns sänd- och mottagarpinnar, där sensorns TX är kopplad till ESP32:ns RX och vice versa. Vidare är vattentemperatursensorn DS18B20 ansluten via 1-Wire-gränssnittet, där en enda dataledning används för kommunikationen. Datalinjen är kopplad till GPIO27 på mikrokontrollern. Alla sensorer är anslutna till en extern strömkälla, och för att mikrokontrollern och sensorerna ska kunna kommunicera med varandra krävs en gemensam referensjord. Därför är den externa strömkällans referensjord sammankopplad med sensorerna och mikrokontrollerns referenspunkter.

4.3 Testning och kalibrering av sensorerna

I detta avsnitt redogörs hur testning och kalibrering av de olika sensorerna utfördes.

4.3.1 Lufttemperatur och fuktighet

Sensorn användes för att mäta lufttemperatur och den relativa luftfuktigheten i odlingsmiljön. Sensorn valdes eftersom den mäter båda parametrarna i samma komponent och kan enkelt komunicera med mikrokontrollern.

4.3.1.1 Testning

Temperaturtestning genomfördes genom att jämföra sensorns uppmätta värden med en extern termometer. För att undersöka sensorns respons vid temperaturförändringar utsattes sensorn även för en lokal temperaturökning genom att placera ett finger mot sensorn under en kort tidsperiod. Därefter observerades hur mätvärdet förändrades när fingret avlägsnades.

Luftfuktighetstestning utfördes genom att först mäta den relativa luftfuktigheten i testmiljön. Därefter testades sensorns respons genom att placera ett fuktigt finger nära sensorn för att tillföra fukt. Efteråt observerades hur mätvärdet påverkades av det fuktiga fingret.

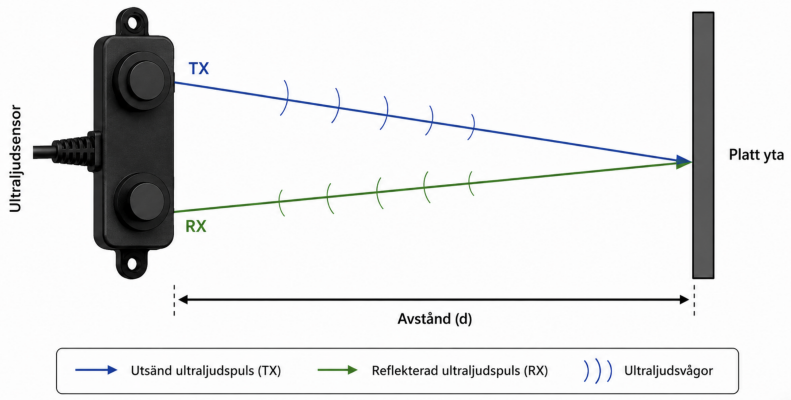
4.3.2 Avståndssensor

Avståndssensorn användes för att uppskatta vattennivån i behållaren. Detta var relevant eftersom en låg vattennivå kan göra att pumpen körs torr, vilket kan orsaka

skador på pumpen och avbryta cirkulationen av näringslösningen i systemet. Sensorn valdes eftersom den är vattentålig och är enkel att integrera med en mikrokontroller.

4.3.2.1 Testning av sensorn

Figur 6 illustrerar hur det första testet av avståndssensor genomfördes.



Figur 6: Testanordning för ultraljudssensorn.

För att testa att avståndssensorn kunde mäta avstånd korrekt sattes en testrigg upp där sensorn fixerades på ett känt avstånd från en plan yta. Syftet var att skapa stabila reflektioner av ljudvågorna och därmed utesluta påverkan från ojämna ytor som annars hade kunnat ge felaktiga mätningar.

Vid test två monterades sensorn på insidan av locket till GrowPipen för att undersöka hur mätvärdena påverkades under drift. Testet genomfördes medan pumpen var aktiv och vatten cirkulerade i systemet, vilket skapade rörelser och ytstörningar på vattenytan. Under testet undersöktes både en tänkt miniminivå och en tänkt maxnivå för vattenvolymen i behållaren. Miniminivån valdes så att vattennivån fortfarande låg ovanför pumpens insug för att minska risken för torrkörning. Maxnivån valdes strax under vattenbehållarens maximala kapacitet, då detta låg vid gränsen för sensorns mätområde. Sensorns mätvärden registrerades kontinuerligt under drift för att observera hur ytstörningrna påverkade mätvärdena. Resultatet av testerna återfinns i Bilaga E.

4.3.3 Vattentemperatur

DS18B20 användes för att mäta temperaturen i näringslösningen. Sensorn valdes eftersom den är vattentät och enkel att integrera med en mikrokontroller.

4.3.3.1 Testning och kalibrering

Testningen av sensorn genomfördes genom att jämföra temperaturen mot en extern termometer. Temperaturen mättes i både kallt och varmt vatten för att undersöka om sensorn följde temperaturförändringarna korrekt. Sensorn flyttades mellan kallt

och varmt vatten samtidigt som förändringar i det uppmätta temperaturvärdet observerades. Testet visas i Bilaga E.

4.3.4 Näringssensor

För att uppskatta koncentrationen av lösta ämnen i näringslösningen användes EC som ett mer konsekvent värde än TDS. Anledningen till att EC används istället för TDS är att EC är ett direkt mått på lösningens elektriska ledningsförmåga, medan TDS bygger på en konverteringsfaktor som är varierande.

4.3.4.1 Testning

Testningen av sensorn genomfördes genom att två behållare fylldes med vatten, varav den ena behållaren tillsattes salt medan den andra innehöll rent vatten. Sensorn placerades därefter i respektive behållare för att undersöka hur de uppmätta EC-värdena förändrades mellan lösningarna. Loggutdrag från testerna visas i Bilaga E.

4.3.4.2 Kalibrering

Kalibrering av EC-sensorn genomfördes med destillerat vatten och egenframställda natriumkloridlösninga med olika koncentrationer. Destillerat vatten användes eftersom det i princip inte har någon mätbar konduktivitet, vilket gjorde det lämpligt som utgångspunkt vid framställningen av lösningarna. Genom att tillsätta salt i kända mängder kunde lösningar med olika beräknade konduktivitetsvärden skapas och användas som referens vid kalibreringen. Först framställdes en stamlösning genom att blanda 1 dl destillerat vatten med 1g bordssalt. Lösningen rördes om tills allt salt lösts upp. Detta gav en koncentration på cirka 10 g/L NaCl.

EC-värdet för stamlösningen uppskattades genom att först beräkna lösningens molaritet enligt ekvation 8.

$$c = \frac{m}{M} \tag{8}$$

där c är molariteten (mol/L), m är masskoncentrationen (g/L) och M är molmassan för NaCl, vilket är $58,44$ g/mol.

Koncentrationen för stamlösningen blev därmed

$$\frac{10}{58,44} \approx 0,171 \text{ mol/L} \tag{9}$$

vilket motsvarar ungefär 171 mM.

En approximation användes därefter där

$$10 \text{ mM NaCl} \approx 1 \text{ mS/cm} \tag{10}$$

vilket gav ett uppskattat EC-värde för stamlösningen på cirka 17 mS/cm.

Tre olika kalibreringslösningar framställdes därefter genom utspädning av stamlösningen med destillerat vatten. Samtliga lösningar hade en total volym på 10 teskedar. EC-värdet för respektive lösning beräknades proportionellt enligt

$$EC_{\text{ny}} = EC_{\text{stam}} \times \frac{V_{\text{stam}}}{V_{\text{total}}} \tag{11}$$

där EC_{stam} är stamlösningens EC-värde, V_{stam} är mängden stamlösning och V_{total} är den totala volymen. Det framställda Kalibreringslösningarna ses i Tabell 1. Vid testningen mättes varje lösning med EC-sensorn och motsvarande ADC-värde registrerades vilket ses i Tabell 1.

Tabell 1: Framställda kalibreringslösningar för EC-sensorn med ADC-värdena i Zephyr.

Testlösning	Blandning	ADC-värde
Lösning 1	1:9	1600
Lösning 2	2:8	2186–2200
Lösning 3	3:7	2300

4.3.5 pH-sensor

pH används för att övervaka om näringslösningen blir för sur eller för basisk, eftersom avvikande pH-värden kan hindra växterna från att ta upp näringsämnen och i vissa fall leda till toxiska förhållanden i lösningen.

4.3.5.1 Testning

För att verifiera att pH-sensorn fungerade fylldes tre behållare med vatten, vinäger och diskmedel. Vatten har normalt ett neutralt pH, vinäger ett surt pH och diskmedel ett basiskt pH. Genom att mäta i respektive behållare kunde förändringar i de uppmätta pH-värdena observeras.

4.3.5.2 Kalibrering

Kalibrering av pH-sensorn genomfördes med hjälp av buffertlösningar med kända pH-värden på 4, 7 och 9. Genom att avläsa mikrokontrollerns ADC-värden för dessa lösningar erhölls tre kända punkter, vilka därefter användes för att relatera pH-värdet till motsvarande ADC-värden. vilket ses i Tabell 2.

Tabell 2: kalibreringslösningar för pH-sensorn med ADC-värdena i Zephyr.

Testlösning	pH	ADC-värde
Lösning 1	4,0	2258
Lösning 2	7,0	1773
Lösning 3	9,0	1482

4.3.6 Ljussensor

Ljussensorn användes för att mäta relativa förändringar i ljusnivån vid odlingssystemet. Sensorn kan inte mäta PAR, men den kan mäta ljusstyrka i lux, vilket gör den lämplig för att följa variationer i ljusförhållandena över tid.

4.3.6.1 Testning

Ljussensorn testades genom att exponera den för en ljuskälla och därefter observera förändringar i mätvärdet. Sensorn jämfördes med en luxmätarapp i mobilen för att undersöka om mätvärdena stämde överens.

4.3.6.2 Kalibrering

Kalibrering av ljussensorn genomfördes eftersom sensorn visade lägre värden vid starkt ljus än den externa luxmätarappen. För att kompensera för avvikelsen beräknades en kalibreringsfaktor på 1,75 genom att dividera referensvärdet med sensorns uppmätta värde enligt ekvation 9.

$$\frac{80000}{45443} \approx 1.76 \tag{12}$$

4.4 Zephyr

I detta avsnitt presenteras vad Zephyr-koden gör och hur den är uppbyggd.

4.4.1 Koduppbyggnad

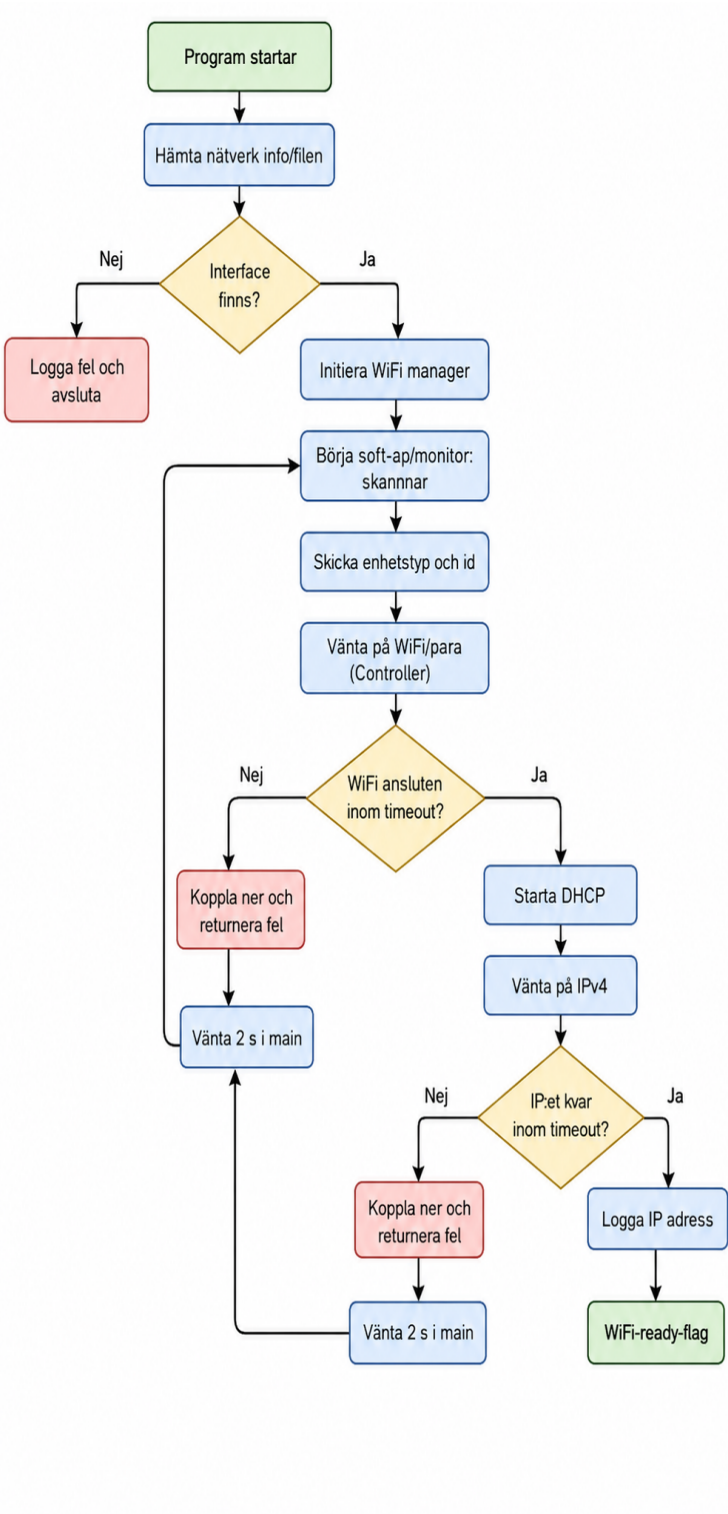
Koden följer en förutbestämd startsekvens där varje steg exekveras i en fast ordning. Wifi-anslutningen måste vara helt upprättad innan uppkopplingen till ThingsBoards MQTT-server sker. Därefter måste MQTT-anslutningen vara aktiv innan sensordata kan läsas och skickas till ThingsBoard.

Koden är uppdelad i moduler där varje del ansvarar för en specifik funktion. Varje modul består av en "`.c-fil`" med själva implementationen och en "`.h-fil`" som beskriver dess gränssnitt. Sensorerna är uppdelade i egna moduler och det finns även separata moduler för wifi samt ThingsBoard/MQTT-kommunikation. Alla viktiga inställningar är samlade i "`app_config.h`", dessa inkluderar wifi-uppgifter, MQTT-konfiguration, ThingsBoard access token, sändningsintervall för MQTT-medelanden, uppdateringsintervall för trådarna och fallbackvärden. Genom att samla alla konfigurerbara inställningar i en "`.h-fil`" förenklas ändringar eftersom uppdateringar inte behöver göras i flera olika filer.

4.4.2 WiFi

I figur 7 visas flödeschemat för systemets uppstartssekvens och etablering av wifi-anslutning. Figuren illustrerar stegen som tas från uppstart till lyckad wifi-anslutning samt hur systemet hanterar misslyckade anslutningar.

Vid uppstart initierar systemet nätverket genom att först hämta ett tillgängligt nätverksinterface. Detta är en grundförutsättning för all vidare kommunikation och om inget interface hittas avbryts processen direkt. När ett interface finns tillgängligt startas wifi-hanteraren. I detta steg registreras semaforer och händelsecallbacks vilket gör det möjligt att hantera asynkrona nätverkshändelser som anslutning, frånkoppling och tilldelning av IP-adress. Därefter konfigureras anslutningen med nätverksnamn (SSID) och lösenord varefter en anslutningsbegäran skickas till nätverket. Systemet väntar sedan på att wifi-länken ska etableras. Denna väntan sker både genom kontinuerlig statuskontroll och via händelser vilket gör att processen fungerar även om nätverket svarar långsamt eller varierande.Om anslutningen misslyckas försöker systemet återansluta tills den lyckas. När wifi-länken är upprättad startas DHCP-processen för att tilldela IPv4-adress, nätmask, standardgateway, DNS-serveradress och leasingtid. Om DHCP-parametrarna inte erhålls inom 35s avbryts försöket, kopplas ner och startas om efter en fördröjning på 2s. Detta skapar en stabil återkopplingsslinga som säkerställer att varje nytt försök börjar från ett korrekt tillstånd. När giltiga DHCP-parametrar har erhållits loggas de, interna statusflaggor uppdateras och nätverksinitialiseringen anses vara klar. Vid denna punkt är både wifi-länken och nätverkskonfigurationen etablerade, vilket innebär att systemet är redo för vidare kommunikation.

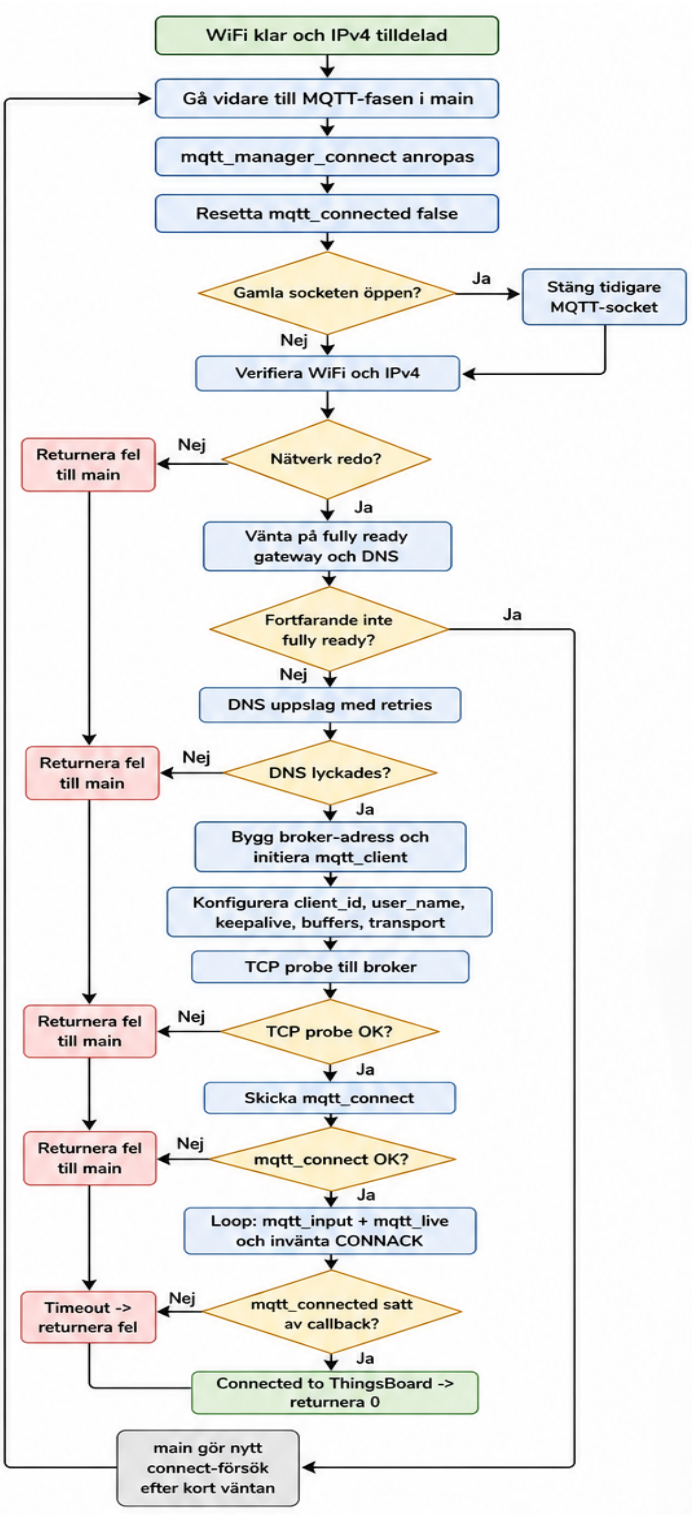


Figur 7: Flödesschema för wifi-anslutning.

4.4.3 MQTT-anslutning och enhetsaktivering

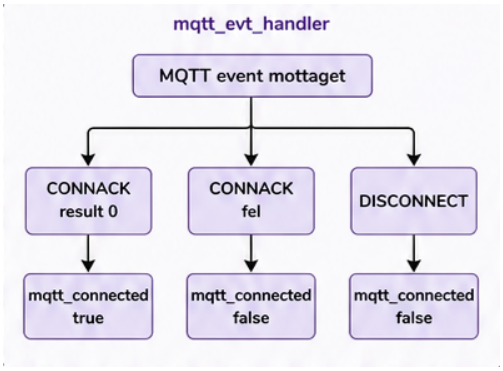
I Figur 8 visas flödesschemat från etablerad nätverksanslutning till lyckad anslutning mot ThingsBoard samt den efterföljande kommunikationsloopen mot ThingsBoards MQTT-brokern. Figuren visar även hur nätverksberedskap verifieras, anslutningen etableras och hur fel, återförsök och drift hanteras.

När wifi-anslutning och DHCP-processen är klar övergår systemet till MQTT-fasen. Anslutningsprocessen inleds med att nollställa tidigare anslutningstillstånd. Om en tidigare nätverksanslutning fortfarande är öppen stängs den innan en ny skapas så att systemet startar från ett definierat utgångsläge. Därefter verifieras att nätverket är redo för vidare kommunikation genom att kontrollera att WiFi-anslutningen är aktiv, en giltig IP-adress har tilldelats samt att DNS-information finns tillgänglig för att kunna slå upp adressen till ThingsBoards MQTT-broker. Om nätverket inte bedöms stabilt avbryts processen och ett nytt anslutningsförsök initieras efter två. När nätverket är redo görs en DNS-uppslagning mot ThingsBoards molnserver för att erhålla brokerns IP-adress. Vid misslyckande gör systemet upp till fem nya försök med tre sekunders mellanrum innan processen avbryts och ett nytt anslutningsförsök initieras efter en två sekunder. Vid lyckad uppslagning kombineras den erhållna IP-adressen med den konfigurerade porten (1883) för att skapa broker-adressen. Systemet initierar sedan MQTT-klienten och konfigurerar den med klient-ID, inloggningsuppgifter för autentisering, samt inställningar för att hålla anslutningen aktiv och överföra data på ett tillförlitligt sätt. Därefter görs en TCP-kontroll mot MQTT-brokern, och om kontrollen lyckas skickas en MQTT CONNECT begäran. Systemet går sedan in i ett vänteläge där inkommande MQTT-data behandlas och sessionen hålls vid liv genom löpande anrop av input- och keepalive-funktioner tills CONNACK (MQTT-brokerns anslutningsbekräftelse) mottas.



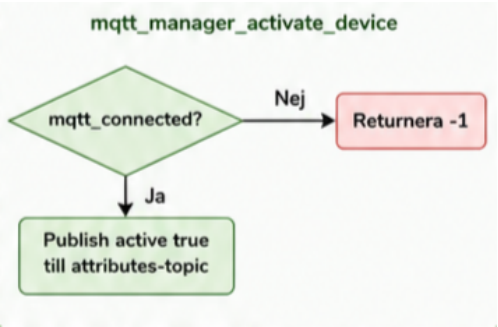
Figur 8: Flödesschema för MQTT-anslutning.

Figur 9 visar hur MQTT-händelser hanteras i en separat callback-funktion där anslutningsresultatet avgör om anslutningen markeras som aktiv eller om den ska betraktas som misslyckad. Vid lyckad anslutning markeras systemet som anslutet till brokern och MQTT-anslutningssteget betraktas som slutfört.



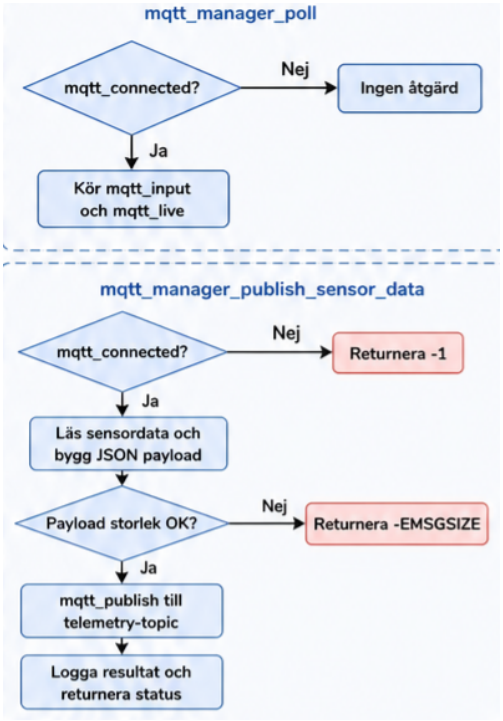
Figur 9: Hantering av MQTT-händelser.

Efter att MQTT-anslutningen har etablerats körs ett separat aktiveringssteg som illustreras i figur 10 där "devicen" i ThingsBoard aktiveras. Aktiveringen sker genom att en active-flagga publiceras till ThingBoard vilket bekräftar att enheten är registrerad, uppkopplad och klar för fortsatt datainsamling och publicering.



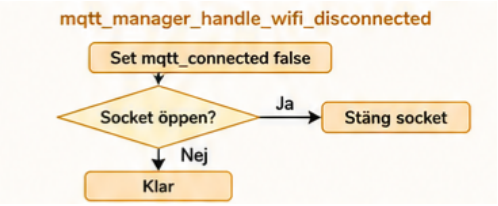
Figur 10: Aktivering av enhet i ThingsBoard.

Vid publicering av sensordata som illustreras i figur 11 kontrolleras först att anslutningen fortfarande är aktiv. Därefter hämtas sensordata och paketeras i ett JavaScript Object Notation (JSON) -meddelande där storleken verifieras innan publicering. Om payloaden överskrider tillåtna gränser avbryts sändningen, annars publiceras data till en telemetry-topic och resultatet loggas.



Figur 11: Publicering av sensordata

Systemet innehåller även hantering för frånkoppling av wifi under drift som visas i figur 12. Vid en sådan händelse markeras MQTT-anslutningen som inaktiv och eventuell öppen nätverksanslutning stängs vilket säkerställer att systemet återgår till ett definierat tillstånd och kan återansluta vid behov.



Figur 12: Hantering av frånkopplad Wi-Fi-anslutning.

4.4.4 Insamling av sensorvärden i Zephyr

Datainsamlingen i systemet är uppbyggd med en multithreading-struktur, vilket innebär att varje sensor hanteras i en egen periodisk tråd. I varje cykel läser tråden in sensordata via rätt gränssnitt och omvandlar värdet till ett internt skalat heltal, exempelvis tiondelar eller hundradelar. Detta gör att systemet får en enhetlig intern representation av mätvärdena. Eftersom varje sensor körs i en separat tråd kan sensorerna uppdateras oberoende av varandra. Om en sensor skulle vara långsammare påverkar det därför inte de andra sensorerna eller resten av datainsamlingen. De senaste mätvärdena samlas i en gemensam snapshot-struktur som fungerar som ett delat minne för sensordata. För att flera trådar inte ska läsa eller skriva i samma datastruktur samtidigt används en mutex. En mutex fungerar som ett lås och ser till att bara en tråd åt gången får åtkomst till snapshot-strukturen, vilket minskar risken för fel i datan. Vid MQTT-publicering används sedan denna snapshot-struktur i stället för att sensorerna läses direkt. Det innebär att systemet endast hämtar de senast lagrade värdena, medan själva sensoravläsningen sker separat i respektive tråd. På så sätt hålls insamling, lagring och överföring åtskilda i systemet, samtidigt som den senaste tillgängliga datan alltid kan skickas vidare.

4.4.5 Fallbackvärden

För att kunna identifiera om en sensor inte fungerar används fallbackvärden. Om sensor inte svarar används ett fördefinierat ersättningsvärde i stället. På så sätt kan systemet fortsätta publicera sensorvärden i samma format även vid partiella fel. För att undvika att ett fallbackvärde misstolkas som ett verkligt mätvärde valdes värden som är utanför sensorernas mätområde och därav fysiskt omöjliga för sensorerna att uppnå. Dessa värden gör det möjligt att upprätta larm för icke-fungerade sensorer i ThingsBoard. Följande fallbackvärden används i systemet:

- Lufttemperatur: -99
- Luftfuktighet: -99
- Vattentemperatur: -99
- EC: -99
- pH: -99
- Ljus: -99
- Avstånd: -99

4.4.6 Sensormoduler

I detta avsnitt redogörs hur varje sensor avläses.

4.4.6.1 Lufttemperatur och luftfuktighet

Systemet försöker först att nå sensorn på adress 0x44. Om sensorn inte svarar görs ett nytt försök på adress 0x45. Vid uppstart initieras först I^2C-gränssnittet och systemet kontrolleras sedan att sensorn svarar genom en första validerande mätning. Vid varje mätning läses totalt 6 byte från sensorn. Byte 0-1 innehåller temperaturdata, byte 2 innehåller CRC för temperaturen, byte 3-4 innehåller luftfuktighetsdata och byte 5 innehåller CRC för luftfuktigheten. CRC (Cyclic Redundancy Check) är ett kontrollvärde som används för att upptäcka datafel vid överföring. Båda datablocken valideras mot respektive CRC innan värdena accepteras. För att starta en ny mätning skickar systemet ett mätkommando till sensorn via I2C-bussen. Efter 20 ms läses 6 byte data från sensorn.

När datan har validerats omvandlas råvärdet för temperatur till tiondelar av grader Celsius och råvärdet för luftfuktighet till tiondelar av procent. Dessa skalade heltalsvärden används sedan vidare i systemet. Om kommunikationen misslyckas markeras sensorn som otillgänglig. Systemet försöker då initiera om sensorn och återställa I^2C-bussen vid nästa mätcykel. Om ingen giltig mätning kan genomföras används fallback-värdet (-99) enligt systemets felhantering.

4.4.6.2 Avståndssensor

Distanssensorn kommunicerar med en överföringshastighet på 9600 baud. Inkommande data läses sekventiellt och tolkas som en dataram på fyra byte, där den första byten utgör startmarkör (0xFF). Därefter verifieras ramen med checksumma för att säkerställa att den mottagits korrekt och utan överföringsfel. När ramen har validerats beräknas avståndet i centimeter utifrån de två databyte som representerar mätvärdet. Genom att endast acceptera validerade ramar minskar systemet risken för brus eller tillfälliga störningar. Vid en giltig mätning uppdateras det senast registrerade avståndsvärdet.

4.4.6.3 Vattentemperatur

Vid avläsning hämtas först DS18B20-enheten från Zephyrs sensorlager och systemet kontrollerar att enheten är redo. Därefter startas en ny temperaturmätning, följt av avläsning av sensorns temperaturkanal. Temperaturen erhålls i två delar, där den ena delen representerar heltalsdelen av temperaturen och den andra delen representerar decimaldelen. Dessa två värden kombineras och omvandlas därefter till systemets interna representation i tiondels grader Celsius. Om något steg i processen misslyckas används i stället ett fördefinierat fallbackvärde.

4.4.6.4 TDS-/EC-Sensor

EC-sensorn avläses analogt via mikrokontrollerns ADC-gränssnitt. Vid den första avläsningen initieras ADC-enheten och ADC1 ställs in för att använda kanal 7, vilket motsvarar GPIO35 på ESP32. Kanalen används med 12-bitars upplösning, intern referensspänning och en gain-inställning på $\frac{1}{4}$. Vid varje mätning samlas totalt tio ADC-prover in från sensorn. Mellan varje prov görs en kort fördröjning på 10 ms för att minska påverkan av tillfälligt brus. Därefter beräknas ett medelvärde av de tio mätpunkterna, vilket används som sensorns råvärde. Det uppmätta ADC-medelvärdet jämförs sedan med ADC-värdena från de uträknade lösningarna för att fastställa vilket EC-värde mätningen motsvarar. De råa ADC-värdena 420, 1600, 2193 och 2300 motsvarar EC-värdena 0,0, 1,7, 3,4 och 5,0 mS/cm. Om det uppmätta ADC-värdet ligger mellan två kalibreringspunkter beräknas EC-värdet med linjär interpolation. Om ADC-värdet överstiger den högsta kalibreringspunkten används extrapolering för att uppskatta EC-värden över 5,0 mS/cm.

Efter att EC-värdet har bestämts temperaturkompenseras det till referenstemperaturen 25 °C. För detta hämtas aktuell vattentemperatur från vattentemperatursensorn. Om någon giltig vattentemperatur inte finns tillgänglig används 25 °C som standardvärde. Temperaturkompenseringen beräknas enligt ekvation 13.

$$EC_{25} = \frac{EC_{mätt}}{1 + 0.02 \cdot (T - 25)} \tag{13}$$

Det slutliga resultatet omvandlas därefter till systemets interna representation i hundradelar av mS/cm. Exempelvis representerar värdet 123 ett EC-värde på 1,23 mS/cm. Om det slutliga värdet blir ogiltigt, om ADC-avläsningen misslyckas eller om sensorn inte kan läsas korrekt, används fallback-värdet -99.

4.4.6.5 pH

pH-sensorn läses av analogt via mikrokontrollerns ADC-gränssnitt. Vid den första avläsningen initieras ADC-enheten och ADC1 ställs in för att använda kanal 5, vilket motsvarar GPIO33 på ESP32. Kanalen används med 12-bitars upplösning, intern referensspänning och en gain-inställning på $\frac{1}{4}$. Vid varje mätning samlas totalt tio ADC-prover in från sensorn. Mellan varje prov görs en kort fördröjning på 10 ms för att minska påverkan av tillfälligt brus. Därefter beräknas ett medelvärde av de tio mätpunkterna, vilket används som sensorns råvärde. Det uppmätta ADC-medelvärdet jämförs sedan med ADC-värdena från buffertlösningarna för att fastställa vilket pH-värde mätningen motsvarar. De råa ADC-värdena 2258, 1773 och 1482 motsvarar pH-värdena 4,0, 7,0 och 9,0. Om det uppmätta ADC-värdet ligger mellan två kalibreringspunkter beräknas pH-värdet med linjär interpolation. Om ADC-värdet hamnar utanför kalibreringsområdet används extrapolering med samma lutning som närmaste intervall, vilket gör att både värden under pH 4,0 och över pH 9,0 kan uppskattas. Det slutliga resultatet omvandlas därefter till systemets interna representation i hundradelar av pH-enheter. Exempelvis representerar värdet 723 ett pH-värde på 7,23. Om det slutliga värdet blir ogiltigt, om ADC-avläsningen misslyckas eller om sensorn inte kan läsas korrekt, används fallback-värdet -99.

4.4.6.6 Ljussensor

Vid avläsning hämtas först TSL2591-enheten från Zephyrs sensorlager. Därefter kontrolleras att sensorn är redo för mätning. Sensorn konfigureras med låg gain, vilket motsvarar en förstärkning på 1 och används vid mätning av starkt ljus, samt en integrationstid på 100 ms. Därefter initieras en ny ljusmätning där både fullspektrumljus och infrarött ljus avläses samtidigt. Luxvärdet beräknas sedan internt enligt ekvation 7. Resultatet erhålls som två delar, där den ena delen utgör heltalsdelen och den andra decimaldelen. Dessa sammanfogas därefter till systemets interna representation, uttryckt i hundradelar av lux enligt ekvation 14.

$$\text{Lux\_Hundradels} = \text{val1} \times 100 + \frac{\text{val2}}{10000} \tag{14}$$

Resultatet skalas sedan med en kalibreringsfaktor på 1,75 för att matcha uppmätta värden mot en referensmätare ekvation 15.

$$\text{Luxkalibrerat} = \text{Luxhundradels} \times 1.75 \tag{15}$$

Om avläsningen misslyckas försöker systemet initiera sensorn på nytt och göra en ny läsning. Om det fortfarande inte fungerar används i stället fallbackvärdet.

4.5 ThingsBoard

ThingsBoard användes som plattform för visualisering, datalagring och larmhantering. Plattformen fungerar som mottagare av sensordata som skickades från mikrokontrollern. Genom ThingsBoard kunde sensordata presenteras i realtid, lagras över tid och användas som grund för larm vid avvikande värden.

I detta avsnitt beskrivs hur sensordata överfördes från ESP32 till ThingsBoard, hur data kopplades till dashboarden, hur layouten av dashboarden utformades samt hur larmfunktionerna implementerades.

4.5.1 Dataflöde i ThingsBoard

För att ansluta systemet skapades en enhet i ThingsBoard. Enheten tilldelades en access token som användes av systemet vid MQTT-anslutningen. När mikrokontrollern har etablerat anslutning mot ThingsBoards MQTT-broker publiceras sensordata till enhetens telemetry-topic i JSON-format.

Varje nyckel i JSON-meddelandet motsvarade ett mätvärde från systemet, exempelvis lufttemperatur, luftfuktighet, pH-värde, EC, ljusnivå eller vattennivå. När datan tagits emot av ThingsBoard kopplades respektive telemetry key till widgets i dashboarden. På så sätt kunde mätvärdena visualiseras utan att ytterligare manuell databehandling behövdes på plattformen.

4.5.2 Dashboardens layout

Dashboarden utformades för att ge en tydlig överblick över odlingssystemets aktuella status. Realtidsvärden presenterades med separata widgets för respektive sensor, medan historiska värden visades med linjediagram. Detta gjorde det möjligt att både övervaka systemets aktuella tillstånd och analysera förändringar över tid.

Layouten delades upp så att vattenrelaterade parametrar, såsom pH, EC, vattentemperatur och vattennivå, kunde särskiljas från omgivningsparametrar såsom lufttemperatur, luftfuktighet och ljusnivå. Syftet med denna uppdelning var att göra dashboarden mer överskådlig och underlätta felsökning vid avvikande mätvärden.

4.5.3 VPD uträkning i ThingsBoard

Utöver de sensorvärden som skickades direkt från ESP32 användes även ThingsBoards rule chain-funktion för att beräkna VPD. VPD användes eftersom det ger en mer heltäckande bild av odlingsmiljön än enbart luftfuktighet.

Beräkningen implementerades i en rule chain i ThingsBoard. När ett MQTT-meddelande togs emot skickades meddelandet vidare till ett script-block där temperatur och luftfuktighet hämtades från meddelandets payload. Därefter beräknades VPD enligt ekvation (5).

Det beräknade VPD-värdet lades in som en ny telemetry key. På så sätt kunde ThingsBoard behandla VPD på samma sätt som övriga sensorvärden. Värdet kunde därefter visualiseras i dashboarden och vid behov användas som grund för larm eller vidare analys.

4.5.4 Larmfunktioner

Larmfunktioner implementerades i ThingsBoard för att upptäcka avvikande mätvärden och fel på sensorerna. Två typer av larm användes. Den första typen aktiverades när ett mätvärde hamnade utanför det fördefinierade intervallet för respektive sensorparameter.

Den andra typen av larm användes för att upptäcka om en sensor var trasig eller urkopplad. Detta baserades på systemets fallbackvärden. Eftersom fallbackvärdena valdes utanför respektive sensors normala mätområde kunde ThingsBoard identifiera detta som ett sensorfel.

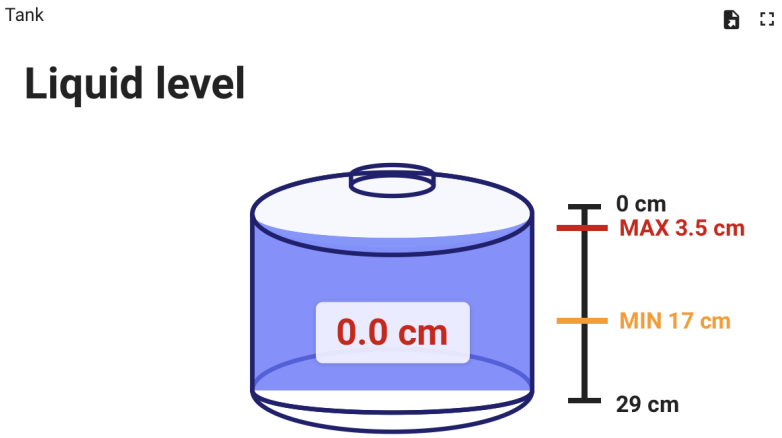
För att skilja mellan driftavvikelser och sensorfel skickades två separata e-postmeddelanden. Ett meddelande användes för larm när ett mätvärde låg utanför det tillåtna intervallet, medan det andra meddelandet användes när systemet möjligtvis identifierade en trasig eller urkopplad sensor. Detta gjorde larmhanteringen tydligare eftersom mottagaren kunde avgöra om problemet berodde på odlingsmiljön eller på själva mätsystemet.

4.5.5 Egna widgets i dashboarden

Utöver ThingsBoards standardwidgets utvecklades två egna widgets för att bättre anpassa visualiseringen till projektets behov. De egna widgetarna togs fram eftersom standardwidgets inte gav den presentation av mätvärden som önskades för odlingssystemet. Syftet var att göra dashboarden mer överskådlig och att tydligare visa systemets aktuella status vid övervakning. Den första widgeten utvecklades för att visualisera pH-värdet i näringslösningen, se figur 13. I stället för att enbart presentera pH som ett numeriskt värde visar widgeten hela pH-skalan med färgade intervall. Detta gör det lättare att snabbt tolka om lösningen är sur, neutral eller basisk. Det aktuella pH-värdet visas ovanför skalan, vilket ger användaren en tydligare visuell indikation än en vanlig numerisk indikator. Under pH-widgeten används även ett linjediagram för att visa hur pH-värdet förändras över tid.

Figur 13: Egenutvecklad widget för visualisering av pH-värde i ThingsBoard.

Den andra widgeten utvecklades för att visualisera vattennivån i tanken, se figur 14. Widgeten visar en förenklad illustration av behållaren tillsammans med det aktuella avståndsvärdet från avståndssensorn. Till höger om behållaren visas även markerade nivågränser för max- och miniminivå. Detta gör det enklare att snabbt bedöma om vattennivån ligger inom ett acceptabelt område eller om den närmar sig en kritisk nivå. Eftersom låg vattennivå kan leda till att pumpen körs torr var denna visualisering viktig för systemets övervakningsfunktion.



Figur 14: Egenutvecklad widget för visualisering av vattennivå i tanken.

Widgetarna byggdes upp med hjälp av HTML, CSS och Javascript i ThingsBoard

och kopplades till relevanta telemetry keys från ESP32. När nya sensorvärden togs emot av ThingsBoard uppdaterades widgetarna automatiskt i dashboarden. På så sätt kunde inkommande sensordata visualiseras direkt med ett utseende och en funktion som var anpassad för projektets användningsområde. Koden för de egenutvecklade widgetarna redovisas i Bilaga C.

4.6 Systemöversikt

Det färdiga systemet består av sensorer, mikrokontroller och molnbaserad visualisering. Sensorerna samlar in data från både vattenmiljön och den omgivande odlingsmiljön. ESP32 läser in dessa värden via ADC, I2C, UART och 1-Wire beroende på sensor. Sensoravläsningen sker med hjälp av multithreading där alla mätvärden samlas i ett gemensamt minne.

När nätverksanslutningen är etablerad skickas den senast tillgängliga sensordatan till ThingsBoard via MQTT. I ThingsBoard visualiseras mätvärdena i en dashboard och används för larmhantering. Systemets övergripande dataflöde kan därmed beskrivas som sensoravläsning, lokal databehandling, MQTT-publicering och molnbaserad visualisering. Systemet byggdes modulärt, vilket möjliggör att nya sensorer eller funktioner kan läggas till utan att hela systemet behöver konstrueras om.

5

Resultat

I detta kapitel presenteras resultaten från projektet. Projektet resulterade i ett fungerande prototypsystem för övervakning av ett hydroponiskt odlingssystem. Resultaten visar att hela kedjan från sensoravläsning till visualisering i ThingsBoard fungerade enligt projektets syfte. Systemet bedöms därför kunna användas som grund för vidare utveckling mot ett mer avancerat styr- och övervakningssystem.

5.1 Elektriska komponenter

Samtliga sensorer kunde integreras med mikrokontrollern och gav mätvärden som bedömdes vara tillräckligt stabila för projektets syfte.

Lufttemperatur- och luftfuktighetssensorn reagerade tydligt på förändringar i omgivningen. Vid temperaturtest ökade temperaturvärdet när sensorn utsattes för kroppsvärme och sjönk sedan gradvis tillbaka till rumstemperatur när värmekällan togs bort. Vid test mot en extern termometer överensstämde även de uppmätta temperaturerna med referensmätningar. Vid test av luftfuktigheten uppmättes 46,47% relativ luftfuktighet i testmiljön. När sensorn utsattes för fukt ökade luftfuktigheten tydligt. Resultatet visar därmed att sensorn reagerar korrekt på förändringar i temperatur och luftfuktighet och kan därmed anses fungera. Utdraget från testet återfinns i Bilaga E.

Vattentemperatursensorn gav temperaturvärden som överensstämde med referensmätningar från den externa termometern. När sensorn flyttades från kallt till varmt vatten ökade temperaturvärdet vilket visade att sensorn reagerade korrekt på temperaturförändringar. Resultatet indikerade att sensorn fungerade stabilt och gav tillförlitliga mätvärden. Utdraget från testet återfinns i Bilaga E.

Ljussensorn gav luxvärden som överensstämde väl med den externa luxmätarappen efter att kalibreringsfaktorn 1,75 infördes. Kompenseringen fungerade väl och vid låga luxnivåer, omkring 100–500 lux, uppgick avvikelsen mellan appen och sensorn vanligtvis till cirka 20 lux, vilket är en marginell skillnad. Vid höga luxnivåer, omkring 80 000 lux, var avvikelsen inte heller särskilt stor då skilnaden endast var mellan 1 000–3 000 lux. Kalibreringsfaktorn gjorde det dessutom möjligt att mäta värden över sensorns ursprungliga mätområde och få resultat över 100 000 lux som stämde väl överens med appens värden. Sensorn visade dock viss känslighet för vilken vinkel den hade mot ljuset, vilket innebär att placeringen av sensorn är viktig för att den ska ge korrekta värden. Testet återfinns i Bilaga E.

Avståndssensorn kunde mäta avstånd korrekt vid test mot en plan yta med känt avstånd. Testet i Bilaga D visade att sensorn gav stabila mätvärden under kontrollerade förhållanden. När sensorn monterades på locket och testades under normal drift påverkades mätvärdena av ytstörningarna. Vid miniminivån uppmättes ett avstånd

på cirka 21 cm från sensorpositionen, vilket motsvarade 3 L vätska i behållaren. Under kontinuerliga mätningar låg majoriteten av mätvärdena omkring 20 cm med vissa avstickare ner mot cirka 17 cm. Vid test av maxnivån uppmättes 16L vätska i behållaren, vilket med mätningar motsvarade omkring 3,7cm från sensorpositionen med fluktuerande värden ner mot omkring 3,3 cm. Trots variationerna visar resultaten att sensorn kan användas för att uppskatta vattennivån i behållaren och identifiera kritiskt låga nivåer. Sensorn bedöms därför uppfylla sitt syfte att minska risken för att pumpen körs torr och cirkulation av näringslösningen avtas. Testerna visas i Bilaga E.

pH-sensorn visade att den kunde reagera på förändringar i lösningens surhetsgrad. Vid test i lösningar med olika pH-värden förändrades det uppmätta värdet i förväntad riktning, vilket indikerar att sensorn fungerade och kunde användas för att uppskatta pH-värdet i näringslösningen. Efter kalibreringen mot buffertlösningarna visade sensorn mycket god överensstämmelse med referensvärdena. Eftersom pH-sensorn är analog påverkas dock mätningen av hur väl programvaran hanterar värden mellan kalibreringspunkterna samt av elektriskt brus. Resultatet visar därför att sensorn var användbar för att indikera om pH-värdet låg inom ett rimligt intervall, men att den inte bör betraktas som tillräckligt noggrann för exakta laboratoriemätningar. För projektets syfte bedömdes sensorn vara tillräcklig eftersom den kunde ge en indikation på om pH-värdet avvek från önskat område. Testet återfinns i Bilaga E.

TDS-sensorn visade att den kunde reagera på olika EC-nivåer, vilket indikerar att sensorn fungerar. Resultatet av kalibreringen visar att sensorn följer de uträknade lösningarna relativt väl, men eftersom ADC-värdena mellan 3,4 och 5,1 mS/cm var nästan identiska kan sensorn inte förväntas ge exakta resultat inom detta intervall. För ett generellt spann mellan dessa värden fungerar sensorn däremot tillräckligt bra för att ge en uppskattning. ADC-intervallet mellan 1,7 och 3,4 mS/cm är större, vilket ger bättre noggrannhet än det högre spannet, men även där kan sensorn inte förväntas ge exakta värden. Sammanfattningsvis lämpar sig sensorn inte för precisa EC-mätningar, men den fungerar tillfredsställande om syftet endast är att skilja mellan olika koncentrationsnivåer. På det lägre spannet fungerar den bättre än på det högre, eftersom ADC-värdena vid kalibreringen låg så nära varandra i det övre intervallet.

ESP32 visade sig fungera väl som plattform för styrning och insamling av sensordata, men dess begränsning till 2,4 GHz-nätverk innebär att användbarheten påverkas i miljöer där sådana nätverk saknas. Detta gör att mikrokontrollern inte är fullt lämplig i alla typer av nätverksmiljöer.

5.2 Zephyr-tester

ESP32 användes tillsammans med Zephyr för hantering och överföring av sensordata. Testerna i Bilaga E visar att systemet kunde ansluta till det trådlösa nätverket och erhålla nätverksparametrar via DHCP. Efter detta etablerades MQTT-anslutningen och sensordata kunde skickas vidare till ThingsBoard. Vid samtidig användning av

flera sensorer kunde systemet läsa in och överföra data utan observerade konflikter mellan sensorerna. Resultatet visar att trådhanteringen och kommunikationen fungerade som avsett även vid parallell datainsamling. Tester visade även att systemet kunde identifiera när en eller flera sensorer inte fungerade och då skicka det definierade fallbackvärdet för den aktuella sensorn. Vid förlorad eller misslyckad nätverksanslutning kunde systemet dessutom automatiskt försöka återansluta. I testfallet som visas i Bilaga E genomfördes återanslutningen framgångsrikt. Resultatet indikerar att kommunikationen mellan sensorer, ESP32 och ThingsBoard fungerade som avsatt samt att systemet har tillräcklig felhantering för att kunna användas i ett hydroponiskt övervakningssystem.

5.3 ThingsBoard

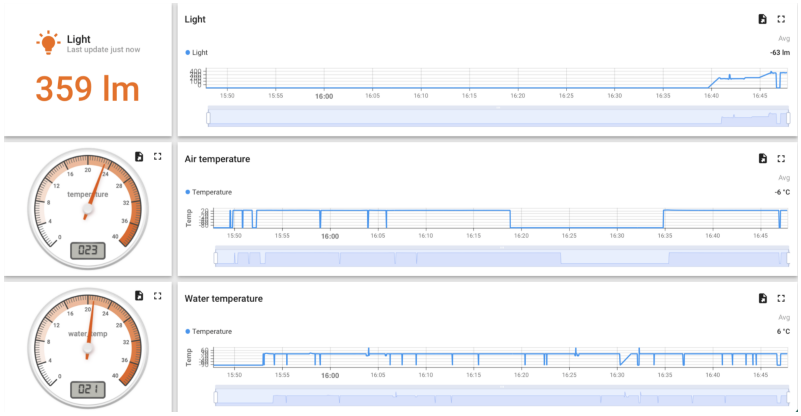
ThingsBoard användes för att visualisera och övervaka sensordata från det hydroponiska systemet. I figur 15 visar hur plattformen kunde ta emot MQTT-meddelanden från mikrokontrollern.

Telemetry

Last update time	Key ↑	Value
2026-05-25 16:47:26	distance	4.5
2026-05-25 16:47:26	ec	4.41
2026-05-25 16:47:26	humidity	52.7
2026-05-25 16:47:26	light	359.03
2026-05-25 16:47:26	pH	0.31
2026-05-25 16:47:26	temperature	23.2
2026-05-25 16:47:26	vpd	1.345024173084773
2026-05-25 16:47:26	water_temp	21.3

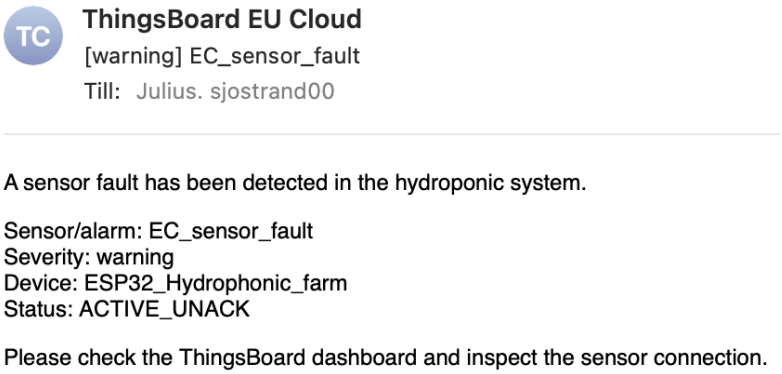
Figur 15: Inkommande telemetridata från ESP32 i ThingsBoard.

Medan figur 15 visar hur en del av hur Dashboarden byggdes upp med flera widgets och linjediagram för visualisering av temperatur, luftfuktighet, vattennivå, pH-värde, ljusnivå och TDS-värden. Detta gjorde det möjligt att övervaka systemets tillstånd över tid och identifiera förändringar i odlingsmiljön. Den fullständiga dashboarden redovisas i Bilaga D.



Figur 16: Dashboard för visualisering av sensorvärden i ThingsBoard.

Plattformens larmfunktioner kunde även användas för att indikera när uppmätta värden överskred det definierade gränsvärden eller när en sensor slutade att fungera. Som visas i figur 17 skickar systemet ut två olika larm beroende på situation. Ett av larmen är för när exempelvis sensorvärderna indikerar låg vattennivå eller avvikande temperatur. Det andra är för när en sensor slutar fungera.



Figur 17: Exempel på larmmeddelanden för driftavvikelse och sensorfel.

ThingsBoard visade sig fungera väl som plattform för fjärrövervakning av det hydroponiska systemet. Plattformen gjorde det möjligt att övervaka systemet utan kontinuerlig manuell insyn och möjliggjorde även att felmeddelanden skickades via e-post när avvikelser eller onormala tillstånd upptäcktes.

6

Slutsats

I detta kapitlet presenteras slutsatserna av examensarbetet

6.1 Sammanfattning av resultat

Syftet med examensarbetet var att utveckla, konfigurera och utvärdera ett inbyggt system för mätning och övervakning av ett hydroponiskt växtodlingssystem baserat på Zephyr RTOS. Resultatet visar att det utvecklade systemet uppfyller de grundläggande funktionella krav som ställdes för examensarbetet. Systemet kunde samla in data från flera sensorer, överföra informationen i realtid via MQTT till ThingsBoard, visualisera mätvärden och hantera larmfunktioner. Examensarbetet visar därmed att ett hydroponiskt odlingssystem kan övervakas med hjälp av ett inbyggt IoT-baserat sensorsystem baserat på Zephyr.

Arbetet visar även att IoT-baserad övervakning kan förenkla kontrollen av hydroponiska odlingssystem genom kontinuerlig datainsamling och visualisering. Detta kan bidra till att avvikelser upptäcks tidigare, exempelvis felaktiga pH-värden, låg vattennivå eller ogynnsamma temperaturer. Ett sådant system kan därför användas både i mindre hobbyodlingar och som grund för vidare utveckling mot mer avancerade och automatiserade odlingslösningar.

6.2 Begränsningar

Projektets resultat blev i huvudsak det som förväntades. Sensorerna reagerade på förändringar i omgivningen och systemet kunde etablera stabil kommunikation med ThingsBoard. Samtidigt identifierades vissa begränsningar. ESP32 visade sig fungera väl som plattform för styrning och insamling av sensordata, men dess begränsning till 2,4 GHz-nätverk innebär att användbarheten påverkas i miljöer där sådana nätverk saknas. Detta gör att mikrokontrollern inte är fullt lämplig i alla typer av nätverksmiljöer.

Även sensorerna hade vissa begränsningar. Vissa av sensorerna gav tillräckligt stabila värden för projektets syfte, men deras noggrannhet påverkades av hur väl de kalibrerades och hur de placerades. Ljussensorn var känslig för vinkeln mot ljuskällan, avståndssensorn påverkades av ytstörningar i behållaren och pH- samt TDS-sensorn bedöms främst vara lämpliga för att indikera avvikelser snarare än för exakta mätningar. Detta innebär att sensorerna fungerade tillräckligt bra för övervakning, men att de inte bör betraktas som fullt precisa mätinstrument.

Begränsningarna i gratisversionen av ThingsBoard bedöms inte utgöra något problem för det utvecklade hydroponiska systemet. En av de relevanta begränsningarna är antalet JavaScript-exekveringar, men för att nå gränsen skulle systemet behöva

skicka två meddelanden per minut varje dag, vilket är en mycket hög meddelandefrekvens för en hydroponisk odling. Även begränsningen för e-postmeddelanden bedöms vara tillräcklig, eftersom tre larm per dag under en månad snarare skulle indikera ett viktigare problem med hårdvaran än en begränsning i ThingsBoard. Slutsatsen är att gratisversionen bedöms vara tillräcklig för ett hydroponiskt system.

6.3 Vidareutveckling

Det finns flera möjliga vägar för vidareutveckling av systemet. En viktig förbättring är att använda noggrannare och bättre sensorer om systemet ska vidareutvecklas till en kommersiell lösning. Nästa steg kan vara att lägga till flera Growpipes och koppla ihop dem med rör, pumpar och aktuatorer för att skapa ett mer komplett system. Dessutom kan automatisk styrning införas så att vatten och näring fylls på i behållarna automatiskt samt att pH-värdet regleras automatiskt. Om systemet ska behållas och användas på hobbynivå bör långtidstester genomföras under flera odlingscykler för att undersöka driftsäkerhet och sensorernas stabilitet över tid.

6.4 Hållbarhetsaspekter

Ur ett hållbarhetsperspektiv har arbetet potential att bidra till en mer resurseffektiv odling. I takt med att jordens befolkning ökar och vattenresurserna blir allt mer begränsade, samtidigt som behovet av mat ökar, kan hydroponiska system vara ett fördelaktigt alternativ. Dessa system använder generellt mindre vatten än traditionell jordodling och möjliggör dessutom odling på höjden, vilket gör dem särskilt lämpade i situationer där odlingsyta är begränsade.Genom att vatten och näringsämnen kan återcirkuleras och kontrolleras mer noggrant kan odlingen även bli mer effektiv. Under rätt förutsättningar kan detta bidra till snabbare tillväxt och större avkastning per odlingsyta jämfört med traditionella odlingsmetoder. Möjligheten att kontinuerligt övervaka näringsnivåer och vattenförbrukning kan bidra till att minska resursslöseri. Genom att samla in och visualisera data får användaren bättre kontroll över odlingsprocessen, vilket minskar risken för överdosering av näringsämnen och onödig energianvändning. Samtidigt innebär användningen av elektronik och sensorer att material och komponenter måste tillverkas och hanteras som avfall, vilket i sig medför en miljöpåverkan. Trots detta kan den prototyp som utvecklats i detta examensarbete anses vara ett steg mot ett mer hållbart och kontrollerat odlingssystem, särskilt i miljöer där vatten och odlingsyta är begränsade.

Litteraturförteckning

[1] U.S. Department of Agriculture, "Hydroponics," National Agricultural Library. [Online]. Tillgänglig: https://www.nal.usda.gov/farms-and-agricultural-production-systems/hydroponics. [Hämtad: 25 mars 2026].

[2] Hydrogarden, "Odlingsguiden." [Online]. Tillgänglig: https://www.hydrogarden.se/odlingsguiden. [Hämtad: 25 mars 2026].

[3] R. S. Velazquez-Gonzalez et al., "A Review on Hydroponics and the Technologies Associated for Medium- and Small-Scale Operations," Agriculture, vol. 12, no. 5, p. 646, 2022. [Online]. Tillgänglig: https://www.mdpi.com/2077-0472/12/5/646. [Hämtad: 25 mars 2026].

[4] M. A. Al Meselmani, "Nutrient Solution for Hydroponics," in Recent Research and Advances in Soilless Culture, M. Turan, S. Argin, E. Yildirim, and A. Güneş, Eds. IntechOpen, 2022. doi:10.5772/intechopen.101604. [Online]. Tillgänglig: https://www.intechopen.com/chapters/80089. [Hämtad: 1 april 2026].

[5] S. E. Wortman, "Crop physiological response to nutrient solution electrical conductivity and pH in an ebb-and-flow hydroponic system," Scientia Horticulturae, vol. 194, pp. 34–42, 2015. [Hämtad: 1 april 2026].

[6] H. Marschner, *Marschner's Mineral Nutrition of Higher Plants*, 3rd ed. London, UK: Academic Press, 2012. [Hämtad: 1 april 2026].

[7] "What Is the Acceptable Total Dissolved Solids (TDS) Level in Drinking Water," The Berkey, [Online]. Tillgänglig: https://theberkey.com/blogs/water-filter/what-is-the-acceptable-total-dissolved-solids-tds-level-in-drinking-water. [Hämtad: Apr. 1, 2026].

[8] E. A. Atekwana et al., "The relationship of total dissolved solids measurements to bulk electrical conductivity in an aquifer contaminated with hydrocarbon," Journal of Contaminant Hydrology, vol. 70, no. 1–2, pp. 1–18, 2004. [Hämtad: 1 april 2026].

[9] Hydroponics Venture, "Hydroponic Nutrient Chart." [Online]. Tillgänglig: https://hydroponicsventure.com/hydroponic-nutrient-chart. [Hämtad: 1 april 2026].

[10] E. Truog, "Soil reaction influence on availability of plant nutrients," Soil Science Society of America Proceedings, vol. 11, pp. 305–308, 1947.

[11] A. O. Taylor and J. A. Rowley, "Plants under climatic stress: I. Low temperature, high light effects on photosynthesis," Plant Physiology, vol. 47,

no. 5, pp. 713–718, 1971. doi:10.1104/pp.47.5.713. [Hämtad: 7 april 2026].

[12] C. P. Levine et al., “Controlling root zone temperature improves plant growth and pigments in hydroponic lettuce,” Annals of Botany, vol. 132, no. 3, 2023. doi:10.1093/aob/mcad127. [Hämtad: 7 april 2026].

[13] R. R. Shamshiri et al., “Review of optimum temperature, humidity, and vapour pressure deficit,” Computers and Electronics in Agriculture, vol. 144, pp. 279–303, 2018. doi:10.1016/j.compag.2017.12.012. [Hämtad: 7 april 2026].

[14] H. Noh and J. Lee, “The Effect of Vapor Pressure Deficit Regulation on the Growth of Tomato Plants,” Applied Sciences, vol. 12, no. 7, p. 3667, 2022. doi:10.3390/app12073667. [Hämtad: 7 april 2026].

[15] M. J. P. Odjugo et al., “Effect of LED light intensity and spectral quality on spinach,” Plants, vol. 10, no. 4, 2022. [Hämtad: 7 april 2026].

[16] J. Liu and M. W. van Iersel, “Photosynthetic physiology of blue, green, and red light,” Frontiers in Plant Science, vol. 12, 2021. doi:10.3389/fpls.2021.619987. [Hämtad: 7 april 2026].

[17] U. Lagercrantz, “At the end of the day: a common molecular mechanism for photoperiod responses in plants?,” *Journal of Experimental Botany*, vol. 60, no. 9, pp. 2501–2515, 2009. doi:10.1093/jxb/erp139.

[18] J. M. Martínez-Gil and R. A. Kliebenstein, “Photoperiod control of plant growth,” Frontiers in Plant Science, vol. 12, 2021. doi:10.3389/fpls.2021.805635. [Hämtad: 7 april 2026].

[19] Espressif Systems, “ESP32-WROOM-32 Datasheet,” v2.6. [Online]. Tillgänglig: `https://documentation.espressif.com/esp32-wroom-32d_esp32-wroom-32u_datasheet_en.pdf`. [Hämtad: 23 mars 2026].

[20] Electronics Tree, “How the TDS Sensor Module Works,” 2023. [Online]. Tillgänglig: `https://electronicstree.com/how-the-tds-sensor-module-works`. [Hämtad: 24 mars 2026].

[21] DFRobot, “TDS Sensor Datasheet,” 2022. [Online]. Tillgänglig: `https://www.farnell.com/datasheets/3161977.pdf`. [Hämtad: 24 mars 2026].

[22] ESPBoards, “A02YYUW Waterproof Ultrasonic Sensor.” [Online]. Tillgänglig: `https://www.espboards.dev/sensors/a02yyuw`. [Hämtad: 24 mars 2026].

[23] R. Keim, “Ambient-Light Sensors Mimic the Human Eye,” DigiKey, Jan. 2014. [Online]. Tillgänglig: `https://www.digikey.co.uk/en/articles/ambient-light-sensors-mimic-the-human-eye`. [Hämtad: 7 april 2026].

[24] AMS AG, “TSL2591 Light-to-Digital Converter,” Datasheet, version 2-04, Jun. 2018. [Online]. Tillgänglig:

https://asset.conrad.com/media10/add/160267/c1/-/en/001516629DS01/datasheet-1516629-adafruit-1980-1-pcs.pdf. [Hämtad: 7 april 2026].

[25] DFRobot, "Gravity Analog pH Sensor." [Online]. Tillgänglig: https://www.mouser.com/pdfDocs/ProductOverview_DFRobot-Gravity-Analog-pH-Sensor.pdf. [Hämtad: 7 april 2026].

[26] Sensirion AG, "SHT3x-DIS Datasheet," 2022. [Online]. Tillgänglig: https://www.sensirion.com/products/catalog/SHT30-DIS-B/. [Hämtad: 7 april 2026].

[27] Analog Devices, "DS18B20 Datasheet," 2019. [Online]. Tillgänglig: https://www.analog.com/media/en/technical-documentation/data-sheets/DS18B20.pdf. [Hämtad: 7 april 2026].

[28] VPN Unlimited, "What is I2C communication?" [Online]. Tillgänglig: https://www.vpnunlimited.com/help/cybersecurity/i2c. [Hämtad: 23 mars 2026].

[29] Analog Devices, "UART Communication Protocol," 2020. [Online]. Tillgänglig: https://www.analog.com/en/resources/analog-dialogue/articles/uart-a-hardware-communication-protocol.html. [Hämtad: 7 maj 2026].

[30] Maxim Integrated, "1-Wire Bus System Description." [Online]. Tillgänglig: https://www.maximintegrated.com/en/design/technical-documents/tutorials/1/1796.html. [Hämtad: 23 mars 2026].

[31] EMQX Team, "MQTT Guide." [Online]. Tillgänglig: https://www.emqx.com/en/blog/the-easiest-guide-to-getting-started-with-mqtt. [Hämtad: 23 mars 2026].

[32] R. Droms, "DHCP," RFC 2131, 1997. [Online]. Tillgänglig: https://www.rfc-editor.org/rfc/rfc2131. [Hämtad: 8 april 2026].

[33] P. Mockapetris, RFC 1034/1035, 1987. [Online]. Tillgänglig: https://www.rfc-editor.org/rfc/rfc1034 och https://www.rfc-editor.org/rfc/rfc1035. [Hämtad: 8 april 2026].

[34] J. Postel, RFC 793, 1981. [Online]. Tillgänglig: https://www.rfc-editor.org/rfc/rfc793. [Hämtad: 8 april 2026].

[35] Zephyr Project. [Online]. Tillgänglig: https://www.zephyrproject.org/learn-about/. [Hämtad: 8 april 2026].

[36] ThingsBoard. [Online]. Tillgänglig: https://thingsboard.io/. [Hämtad: 8 april 2026].

[37] ThingsBoard, "Subscription Plans," ThingsBoard Cloud Documentation. [Online]. Tillgänglig:

https://thingsboard.io/docs/paas/eu/reference/subscriptions/.
[Hämtad: 24 maj 2026].

A

Bilaga

Här presenteras rekommenderade intervall för flera grönsaker, örter och frukter. Tabellerna är sammanställda utifrån Hydroponics Venture [9]. Temperatur-, ljus- och luxintervall bör ses som generella riktvärden och kan variera beroende på sort, tillväxtstadium och odlingsförhållanden.

A.1 Tabell för Grönsaker

Tabell 3: Tabell för Grönsaker

Växt	pH	EC (dS/m)	Lufttemp (°C)	Vattentemp (°C)	Ljusduration	Optimal luxnivå
Artichoke	6.5–7.5	0.8–1.8	20–25	18–22	12–14 h/dag	15000–25000
Asparagus	6.0–6.8	1.4–1.8	20–25	18–22	12–14 h/dag	12000–20000
Bean (Common)	6.0	2.0–4.0	22–28	18–24	14–16 h/dag	20000–30000
Broad Bean	6.0–6.5	1.8–2.2	18–24	18–22	12–14 h/dag	12000–18000
Broccoli	6.0–6.5	2.8–3.5	18–24	18–22	12–14 h/dag	10000–18000
Cabbage	6.5–7.0	2.5–3.0	18–24	18–22	12–14 h/dag	10000–18000
Capsicum	6.0–6.5	1.8–2.2	22–28	20–25	14–16 h/dag	25000–40000
Cauliflower	6.0–7.0	0.5–2.0	18–24	18–22	12–14 h/dag	10000–18000
Cucumber	5.8–6.0	1.7–2.5	22–28	20–25	14–16 h/dag	25000–35000
Eggplant	5.5	2.5–3.5	24–30	20–26	14–16 h/dag	25000–40000
Lettuce	5.5–6.5	0.8–1.2	18–24	18–22	12–16 h/dag	8000–15000
Pea	6.0–7.0	0.8–1.8	16–22	16–20	12–14 h/dag	10000–18000
Tomato	5.5–6.5	2.0–5.0	22–28	20–26	14–18 h/dag	25000–50000
Peppers	5.8–6.3	2.0–3.0	22–28	20–25	14–16 h/dag	25000–40000
Squash (Summer)	5.0–6.5	1.8–2.4	22–28	20–25	14–16 h/dag	20000–35000
Pumpkin	5.5–7.5	1.8–2.4	22–28	20–25	14–16 h/dag	20000–35000

A.2 Tabell för Rotgrönsaker

Tabell 4: Tabell för Rotgrönsaker

Växt	pH	EC (dS/m)	Lufttemp (°C)	Vattentemp (°C)	Ljusduration	Optimal luxnivå
Beetroot	6.0–6.5	0.8–5.0	18–24	18–22	12–14 h/dag	10000–18000
Carrots	6.3	1.6–2.0	18–24	18–22	12–14 h/dag	10000–18000
Garlic	6.0	1.4–1.8	18–22	18–20	12–14 h/dag	12000–20000
Leek	6.5–7.0	1.4–1.8	16–22	18–20	12–14 h/dag	10000–18000
Onions	6.0–6.7	1.4–1.8	18–24	18–22	12–14 h/dag	12000–20000
Parsnip	6.0	1.4–1.8	16–22	18–20	12–14 h/dag	10000–18000
Potato	5.0–6.0	2.0–2.5	18–24	18–22	14–16 h/dag	18000–25000
Radish	6.0–7.0	1.6–2.2	16–22	16–20	10–12 h/dag	8000–15000
Silverbeet	6.0–7.0	1.8–2.3	18–24	18–22	12–14 h/dag	10000–18000
Sweet Potato	5.5–6.0	2.0–2.5	22–28	20–25	14–16 h/dag	20000–30000
Turnip	6.0–6.5	1.8–2.4	16–22	18–20	12–14 h/dag	10000–18000

A.3 Tabell för Örter och frukt

Tabell 5: Tabell för Örter och frukt

Växt	pH	EC (dS/m)	Lufttemp (°C)	Vattentemp (°C)	Ljusduration	Optimal luxnivå
Basil	5.5–6.5	1.0–1.6	20–26	20–24	14–16 h/dag	10000–20000
Chicory	5.5–6.0	2.0–2.4	18–24	18–22	12–14 h/dag	10000–18000
Chives	6.0–6.5	1.8–2.4	18–24	18–22	12–14 h/dag	10000–15000
Fennel	6.4–6.8	1.0–1.4	18–24	18–22	12–14 h/dag	12000–18000
Lavender	6.4–6.8	1.0–1.4	20–28	18–22	14–16 h/dag	20000–30000
Lemon Balm	5.5–6.5	1.0–1.6	18–24	18–22	12–14 h/dag	10000–15000
Marjoram	6.0	1.6–2.0	20–26	18–22	12–14 h/dag	12000–18000
Mint	5.5–6.0	2.0–2.4	18–24	18–22	12–14 h/dag	8000–15000
Mustard Cress	6.0–6.5	1.2–2.4	18–24	18–22	12–14 h/dag	10000–18000
Parsley	5.5–6.0	0.8–1.8	18–24	18–22	12–14 h/dag	10000–18000
Rosemary	5.5–6.0	1.0–1.6	20–28	18–22	14–16 h/dag	20000–30000
Sage	5.5–6.5	1.0–1.6	20–28	18–22	14–16 h/dag	18000–28000
Thyme	5.5–7.0	0.8–1.6	20–28	18–22	14–16 h/dag	20000–30000
Watercress	6.5–6.8	0.4–1.8	16–22	16–20	12–14 h/dag	8000–15000
Banana	5.5–6.5	1.8–2.2	24–30	22–26	12–14 h/dag	25000–40000
Black Currant	6.0	1.4–1.8	18–24	18–22	12–14 h/dag	12000–20000
Blueberry	4.0–5.0	1.8–2.0	18–24	18–22	12–14 h/dag	12000–20000
Melon	5.5–6.0	2.0–2.5	22–30	20–25	14–16 h/dag	25000–40000
Passionfruit	6.5	1.6–2.4	22–28	20–24	12–14 h/dag	20000–35000
Paw-Paw	6.5	2.0–2.4	24–30	22–26	12–14 h/dag	25000–40000
Pineapple	5.5–6.0	2.0–2.4	22–30	20–25	12–14 h/dag	20000–35000
Red Currant	6.0	1.4–1.8	18–24	18–22	12–14 h/dag	12000–20000
Rhubarb	5.0–6.0	1.6–2.0	14–20	16–20	10–12 h/dag	8000–15000
Strawberries	5.5–6.5	1.8–2.2	18–24	18–22	14–16 h/dag	20000–30000
Watermelon	5.8	1.5–2.4	22–30	20–25	14–16 h/dag	25000–40000

Notering: Rekommenderade temperaturer, ljusdurationer och luxnivåer är generaliserade intervall baserade på odlingsprinciper och kan variera beroende på sort och tillväxtstadium.

B

Bilaga

Här visas all kod som gjorts i Zephyr

B.1 CMakeList.txt

```
# SPDX-License-Identifier: Apache-2.0

cmake_minimum_required(VERSION 3.20.0)

find_package(Zephyr REQUIRED HINTS $ENV{ZEPHYR_BASE})
project(hello_world)

target_sources(app PRIVATE
src/main.c
src/wifi_manager.c
src/mqtt_manager.c
src/sensor_manager.c
src/temperature_and_humidity_sensor.c
src/i2c_bus_lock.c
src/water_temp_param.c
src/ec_param.c
src/ph_param.c
src/light_param.c
src/distance_param.c
)
```

B.2 prj.conf

```
# =====================
# SERIAL OUTPUT (ONLY PRINTK)
# =====================
CONFIG_CONSOLE=y
CONFIG_UART_CONSOLE=y
CONFIG_PRINTK=y
CONFIG_STDOUT_CONSOLE=y


# Disable LOG for clean serial output
CONFIG_LOG=n


# =====================
# I2C (for SHT31 sensor)
# =====================
CONFIG_I2C=y
```

```
CONFIG_I2C_ESP32=y


# =====================
# ADC (for TDS/EC sensor on GPIO35)
# =====================
CONFIG_ADC=y


# =====================
# 1-Wire / DS18B20 (water temperature)
# =====================
CONFIG_SENSOR=y
CONFIG_W1=y
CONFIG_W1_NET=y
CONFIG_W1_ZEPHYR_GPIO=y
CONFIG_DS18B20=y
CONFIG_TSL2591=y
CONFIG_TSL2591_FETCH_WAIT=y


# =====================
# NETWORK CORE
# =====================
CONFIG_NETWORKING=y
CONFIG_NET_IPV4=y
CONFIG_NET_TCP=y
CONFIG_NET_UDP=y
CONFIG_NET_SOCKETS=y
CONFIG_NET_DHCPV4=y
CONFIG_DNS_RESOLVER=y
CONFIG_NET_CONFIG_SETTINGS=y
CONFIG_NET_CONFIG_NEED_IPV4=y
CONFIG_NET_INTERFACE_NAME=y
CONFIG_NET_INTERFACE_NAME_LEN=8


# =====================
# WIFI (ESP32)
# =====================
CONFIG_WIFI=y
CONFIG_WIFI_ESP32=y
CONFIG_NET_L2_WIFI_MGMT=y


# =====================
# MQTT
# =====================
CONFIG_MQTT_LIB=y
```

```
# =====================
# OPTIONAL STABILITY
# =====================
CONFIG_MAIN_STACK_SIZE=4096
CONFIG_SYSTEM_WORKQUEUE_STACK_SIZE=2048
```

B.3 app.overlay

```
#include <zephyr/dt-bindings/pinctrl/esp32-pinctrl.h>

&pinctrl {
    i2c0_custom: i2c0_custom {
        group1 {
            pinmux = <I2C0_SDA_GPIO25>,
                     <I2C0_SCL_GPIO26>;
            bias-pull-up;
            drive-open-drain;
            output-high;
        };
    };

    uart2_custom: uart2_custom {
        group1 {
            pinmux = <UART2_TX_GPIO18>;
        };

        group2 {
            pinmux = <UART2_RX_GPIO5>;
            bias-pull-up;
        };
    };
};

&i2c0 {
    pinctrl-0 = <&i2c0_custom>;
    pinctrl-names = "default";
    sda-gpios = <&gpio0 25 (GPIO_PULL_UP | GPIO_OPEN_DRAIN)>;
    scl-gpios = <&gpio0 26 (GPIO_PULL_UP | GPIO_OPEN_DRAIN)>;
    clock-frequency = <I2C_BITRATE_STANDARD>;
    status = "okay";

    tsl2591: tsl2591@29 {
        compatible = "ams,tsl2591";
        reg = <0x29>;
        status = "okay";
    };
};
```

```
&uart2 {
    current-speed = <9600>;
    pinctrl-0 = <&uart2_custom>;
    pinctrl-names = "default";
    status = "okay";
};

&adc0 {
    status = "okay";
};

/ {
    w1_0: w1-gpio-0 {
        compatible = "zephyr,w1-gpio";
        gpios = <&gpio0 27 (GPIO_ACTIVE_HIGH |
           GPIO_OPEN_DRAIN | GPIO_PULL_UP)>;
        #address-cells = <1>;
        #size-cells = <0>;
        status = "okay";

        ds18b20_0: ds18b20@0 {
            compatible = "maxim,ds18b20";
            reg = <0>;
            family-code = <0x28>;
            resolution = <12>;
            status = "okay";
        };
    };
};
```

B.4 app_config.h

```
#ifndef APP_CONFIG_H
#define APP_CONFIG_H

#define WIFI_SSID "YOURE WIFI_SSID"
#define WIFI_PASSWORD "WIFI PASSWORD"

#define MQTT_BROKER_HOST "eu.thingsboard.cloud"
#define MQTT_BROKER_PORT 1883
#define ACCESS_TOKEN "THINGSBOARD ACCESS TOKEN"

#define APP_MQTT_BUFFER_SIZE 256
#define APP_MQTT_PUBLISH_INTERVAL_MS 1000 // milliseconds
#define APP_SENSOR_UPDATE_INTERVAL_MS 1000//milliseconds

/* Per-parameter fallback values (scaled integers used across
    the app). */
```

```
#define APP_FALLBACK_AIR_TEMP_TENTHS       -99
#define APP_FALLBACK_AIR_HUMIDITY_TENTHS   -99
#define APP_FALLBACK_WATER_TEMP_TENTHS     -99
#define APP_FALLBACK_EC_HUNDREDTHS         -99
#define APP_FALLBACK_PH_HUNDREDTHS         -99
#define APP_FALLBACK_LIGHT_RAW             -99
#define APP_FALLBACK_DISTANCE_MM           -99

/* Fallback values above are used only on real read failure/
   unavailable sensor. */
#define APP_FORCE_FALLBACK_AIR_TEMP        0
#define APP_FORCE_FALLBACK_AIR_HUMIDITY    0
#define APP_FORCE_FALLBACK_WATER_TEMP      0
#define APP_FORCE_FALLBACK_EC              0
#define APP_FORCE_FALLBACK_PH              0
#define APP_FORCE_FALLBACK_LIGHT           0
#define APP_FORCE_FALLBACK_DISTANCE        0

#endif
```

B.5 main.c

```
/*
 * Main orchestration: WiFi + MQTT managers handle protocol
    details.
 */

#include <zephyr/kernel.h>
#include <zephyr/sys/printk.h>
#include <zephyr/net/net_if.h>
#include <zephyr/device.h>
#include <zephyr/devicetree.h>

#include "app_config.h"
#include "wifi_manager.h"
#include "mqtt_manager.h"
#include "sensor_manager.h"
#include "temperature_and_humidity_sensor.h"
#include "water_temp_param.h"
#include "light_param.h"
#include "distance_param.h"

static void print_sensor_status(const char *name, bool
   forced_fallback, bool active)
{
        if (forced_fallback) {
                printk("%s forced fallback value active\n",
                   name);
                return;
```

```
        }

        if (active) {
                printk("%s active\n", name);
        } else {
                printk("%s fallback value active\n", name);
        }
}

static void print_sensor_connection(const char *name, bool
   connected)
{
        printk("%s: %s\n", name, connected ? "connected" : "
           not connected");
}

static bool is_adc_ready(void)
{
        const struct device *adc = DEVICE_DT_GET(DT_NODELABEL
           (adc0));
        return device_is_ready(adc);
}

static bool is_distance_uart_ready(void)
{
        const struct device *uart_dev = DEVICE_DT_GET(
           DT_NODELABEL(uart2));
        return device_is_ready(uart_dev);
}

static bool is_ds18b20_ready(void)
{
        const struct device *dev = DEVICE_DT_GET_ANY(
           maxim_ds18b20);
        return dev && device_is_ready(dev);
}

static bool is_water_temp_connected(void)
{
        if (!is_ds18b20_ready()) {
                return false;
        }

        return param_water_temp_tenths(0) !=
           APP_FALLBACK_WATER_TEMP_TENTHS;
}

static bool is_light_connected(void)
{
```

```
        return param_light_hundredths(0) !=
           APP_FALLBACK_LIGHT_RAW;
}

static bool is_distance_connected(void)
{
        if (!is_distance_uart_ready()) {
                return false;
        }

        return param_distance(0) != APP_FALLBACK_DISTANCE_MM;
}

static void print_sensor_connection_summary(void)
{
        printk("\n=======Check Sensor connection========\n");

        /* Temperature/Humidity share the SHT31 sensor. */
        (void)sht31_init();
        bool sht31_connected = sht31_is_available();
        bool adc_connected = is_adc_ready();
        bool ds18b20_connected = is_water_temp_connected();
        bool light_connected = is_light_connected();
        bool distance_connected = is_distance_connected();

        print_sensor_connection("Temperature", !
           APP_FORCE_FALLBACK_AIR_TEMP && sht31_connected);
        print_sensor_connection("Humidity", !
           APP_FORCE_FALLBACK_AIR_HUMIDITY && sht31_connected
           );
        print_sensor_connection("Water temp", !
           APP_FORCE_FALLBACK_WATER_TEMP && ds18b20_connected
           );
        print_sensor_connection("EC", !APP_FORCE_FALLBACK_EC
           && adc_connected);
        print_sensor_connection("Light", !
           APP_FORCE_FALLBACK_LIGHT && light_connected);
        print_sensor_connection("Distance", !
           APP_FORCE_FALLBACK_DISTANCE && distance_connected)
           ;
        print_sensor_connection("pH", !APP_FORCE_FALLBACK_PH
           && adc_connected);

        printk("\n=======Sensor status========\n");
        print_sensor_status("Temperature",
           APP_FORCE_FALLBACK_AIR_TEMP, sht31_connected);
        print_sensor_status("Humidity",
           APP_FORCE_FALLBACK_AIR_HUMIDITY, sht31_connected);
```

```
        print_sensor_status("Water temp",
           APP_FORCE_FALLBACK_WATER_TEMP, ds18b20_connected);
        print_sensor_status("EC", APP_FORCE_FALLBACK_EC,
           adc_connected);
        print_sensor_status("Light", APP_FORCE_FALLBACK_LIGHT
           , light_connected);
        print_sensor_status("Distance",
           APP_FORCE_FALLBACK_DISTANCE, distance_connected);
        print_sensor_status("pH", APP_FORCE_FALLBACK_PH,
           adc_connected);
        printk("\n=======Publish Sensordata=======\n");
}

int main(void)
{
printk("=== ESP32 WiFi Connect Test ===\n");
printk("Program started successfully\n");

struct net_if *iface = net_if_get_default();
if (!iface) {
printk("ERROR: No network interface available\n");
return 0;
}

printk("Network interface found\n");

wifi_manager_init();

/* Hard lock startup sequence: WiFi -> ThingsBoard -> Sensor
   check -> Publish */
while (wifi_manager_connect() != 0) {
        k_sleep(K_SECONDS(2));
}

printk("\n=========ThingsBoard MQTT Connection & Device
   Activation============\n");
while (mqtt_manager_connect() != 0) {
        k_sleep(K_SECONDS(2));
}
(void)mqtt_manager_activate_device();

print_sensor_connection_summary();

sensor_manager_start();

int status_check_counter = 0;
int reconnect_delay_counter = 0;
int publish_timer_ms = APP_MQTT_PUBLISH_INTERVAL_MS;
bool was_connected = false;
```

```
while (1) {
k_sleep(K_SECONDS(1));
status_check_counter++;
        publish_timer_ms -= 1000;

        if (mqtt_manager_is_connected()) {
                mqtt_manager_poll();
                if (publish_timer_ms <= 0) {
                        (void)
                           mqtt_manager_publish_sensor_data()
                           ;
                        publish_timer_ms =
                           APP_MQTT_PUBLISH_INTERVAL_MS;
                }
        }

if (status_check_counter >= 5) {
status_check_counter = 0;

int connection_status = wifi_manager_check_connection();

if (connection_status == 0) {
was_connected = true;
reconnect_delay_counter = 0;

if (!mqtt_manager_is_connected()) {
// Use fully_ready to ensure DNS is actually functional
if (wifi_manager_is_network_fully_ready() &&
   mqtt_manager_connect() == 0) {
(void)mqtt_manager_activate_device();
        publish_timer_ms = APP_MQTT_PUBLISH_INTERVAL_MS;
}
}
} else {
// WiFi check FAILED - connection is actually lost
if (was_connected) {
printk("WiFi connection LOST - detected by active check\n");
was_connected = false;
}

// Implement reconnection delay to avoid spamming reconnect
   attempts
if (reconnect_delay_counter == 0) {
printk("Will attempt WiFi reconnection in 15 seconds...\n");
reconnect_delay_counter = 3;  // 3 * 5 seconds = 15 seconds
}

reconnect_delay_counter--;
```

```
if (reconnect_delay_counter == 0) {
printk("Attempting WiFi reconnection now...\n");
wifi_manager_reconnect();
        publish_timer_ms = APP_MQTT_PUBLISH_INTERVAL_MS;
}
}
}
}

return 0;
}
```

B.6 sensor_manager.c

```
#include "sensor_manager.h"

#include <zephyr/kernel.h>

#include "app_config.h"
#include "temperature_and_humidity_sensor.h"
#include "water_temp_param.h"
#include "ec_param.h"
#include "ph_param.h"
#include "light_param.h"
#include "distance_param.h"

#define SENSOR_THREAD_STACK_SIZE 1536
#define SENSOR_THREAD_PRIORITY 7

static struct sensor_snapshot latest_snapshot = {
        .temperature_tenths = APP_FALLBACK_AIR_TEMP_TENTHS,
        .humidity_tenths = APP_FALLBACK_AIR_HUMIDITY_TENTHS,
        .water_temp_tenths = APP_FALLBACK_WATER_TEMP_TENTHS,
        .ec_hundredths = APP_FALLBACK_EC_HUNDREDTHS,
        .ph_hundredths = APP_FALLBACK_PH_HUNDREDTHS,
        .light_hundredths = APP_FALLBACK_LIGHT_RAW,
        .distance_mm = APP_FALLBACK_DISTANCE_MM,
};

static struct k_mutex snapshot_lock;
static bool manager_started;

K_THREAD_STACK_DEFINE(air_sensor_stack,
   SENSOR_THREAD_STACK_SIZE);
K_THREAD_STACK_DEFINE(water_temp_stack,
   SENSOR_THREAD_STACK_SIZE);
K_THREAD_STACK_DEFINE(ec_stack, SENSOR_THREAD_STACK_SIZE);
K_THREAD_STACK_DEFINE(ph_stack, SENSOR_THREAD_STACK_SIZE);
K_THREAD_STACK_DEFINE(light_stack, SENSOR_THREAD_STACK_SIZE);
```

```
K_THREAD_STACK_DEFINE(distance_stack,
   SENSOR_THREAD_STACK_SIZE);

static struct k_thread air_sensor_thread_data;
static struct k_thread water_temp_thread_data;
static struct k_thread ec_thread_data;
static struct k_thread ph_thread_data;
static struct k_thread light_thread_data;
static struct k_thread distance_thread_data;

static void update_air_values(int temperature_tenths, int
   humidity_tenths)
{
        k_mutex_lock(&snapshot_lock, K_FOREVER);
        latest_snapshot.temperature_tenths =
           temperature_tenths;
        latest_snapshot.humidity_tenths = humidity_tenths;
        k_mutex_unlock(&snapshot_lock);
}

static void update_single_value(int *field, int value)
{
        k_mutex_lock(&snapshot_lock, K_FOREVER);
        *field = value;
        k_mutex_unlock(&snapshot_lock);
}

static void air_sensor_thread(void *arg1, void *arg2, void *
   arg3)
{
        ARG_UNUSED(arg1);
        ARG_UNUSED(arg2);
        ARG_UNUSED(arg3);

        while (1) {
                int temperature_tenths =
                   APP_FALLBACK_AIR_TEMP_TENTHS;
                int humidity_tenths =
                   APP_FALLBACK_AIR_HUMIDITY_TENTHS;

                if (!sht31_is_available()) {
                        (void)sht31_init();
                }

                if (sht31_is_available()) {
                        (void)
                           sht31_read_temperature_and_humidity_tenths
                           (&temperature_tenths, &
                           humidity_tenths);
```

```
                }

                update_air_values(temperature_tenths,
                    humidity_tenths);
                k_sleep(K_MSEC(APP_SENSOR_UPDATE_INTERVAL_MS)
                    );
        }
}

static void water_temp_thread(void *arg1, void *arg2, void *
    arg3)
{
        ARG_UNUSED(arg1);
        ARG_UNUSED(arg2);
        ARG_UNUSED(arg3);

        while (1) {
                int value = param_water_temp_tenths(0);
                update_single_value(&latest_snapshot.
                    water_temp_tenths, value);
                k_sleep(K_MSEC(APP_SENSOR_UPDATE_INTERVAL_MS)
                    );
        }
}

static void ec_thread(void *arg1, void *arg2, void *arg3)
{
        ARG_UNUSED(arg1);
        ARG_UNUSED(arg2);
        ARG_UNUSED(arg3);

        while (1) {
                int value = APP_FORCE_FALLBACK_EC
                        ? APP_FALLBACK_EC_HUNDREDTHS
                        : param_ec_hundredths(0);
                update_single_value(&latest_snapshot.
                    ec_hundredths, value);
                k_sleep(K_MSEC(APP_SENSOR_UPDATE_INTERVAL_MS)
                    );
        }
}

static void ph_thread(void *arg1, void *arg2, void *arg3)
{
        ARG_UNUSED(arg1);
        ARG_UNUSED(arg2);
        ARG_UNUSED(arg3);

        while (1) {
```

```
                int value = APP_FORCE_FALLBACK_PH
                        ? APP_FALLBACK_PH_HUNDREDTHS
                        : param_ph_hundredths(0);
                update_single_value(&latest_snapshot.
                   ph_hundredths, value);
                k_sleep(K_MSEC(APP_SENSOR_UPDATE_INTERVAL_MS)
                   );
        }
}

static void light_thread(void *arg1, void *arg2, void *arg3)
{
        ARG_UNUSED(arg1);
        ARG_UNUSED(arg2);
        ARG_UNUSED(arg3);

        while (1) {
                int value = param_light_hundredths(0);
                update_single_value(&latest_snapshot.
                   light_hundredths, value);

                k_sleep(K_MSEC(APP_SENSOR_UPDATE_INTERVAL_MS)
                   );
        }
}

static void distance_thread(void *arg1, void *arg2, void *
   arg3)
{
        ARG_UNUSED(arg1);
        ARG_UNUSED(arg2);
        ARG_UNUSED(arg3);

        while (1) {
                int value = param_distance(0);
                update_single_value(&latest_snapshot.
                   distance_mm, value);
                k_sleep(K_MSEC(APP_SENSOR_UPDATE_INTERVAL_MS)
                   );
        }
}

void sensor_manager_start(void)
{
        if (manager_started) {
                return;
        }

        k_mutex_init(&snapshot_lock);
```

```
k_thread_create(&air_sensor_thread_data,
    air_sensor_stack,
        K_THREAD_STACK_SIZEOF(air_sensor_stack),
            air_sensor_thread,
        NULL, NULL, NULL, SENSOR_THREAD_PRIORITY, 0,
            K_NO_WAIT);
k_thread_name_set(&air_sensor_thread_data, "
    air_sensor");

k_thread_create(&water_temp_thread_data,
    water_temp_stack,
        K_THREAD_STACK_SIZEOF(water_temp_stack),
            water_temp_thread,
        NULL, NULL, NULL, SENSOR_THREAD_PRIORITY, 0,
            K_NO_WAIT);
k_thread_name_set(&water_temp_thread_data, "
    water_temp");

k_thread_create(&ec_thread_data, ec_stack,
        K_THREAD_STACK_SIZEOF(ec_stack), ec_thread,
        NULL, NULL, NULL, SENSOR_THREAD_PRIORITY, 0,
            K_NO_WAIT);
k_thread_name_set(&ec_thread_data, "ec_sensor");

k_thread_create(&ph_thread_data, ph_stack,
        K_THREAD_STACK_SIZEOF(ph_stack), ph_thread,
        NULL, NULL, NULL, SENSOR_THREAD_PRIORITY, 0,
            K_NO_WAIT);
k_thread_name_set(&ph_thread_data, "ph_sensor");

k_thread_create(&light_thread_data, light_stack,
        K_THREAD_STACK_SIZEOF(light_stack),
            light_thread,
        NULL, NULL, NULL, SENSOR_THREAD_PRIORITY, 0,
            K_NO_WAIT);
k_thread_name_set(&light_thread_data, "light_sensor")
    ;

k_thread_create(&distance_thread_data, distance_stack
    ,
        K_THREAD_STACK_SIZEOF(distance_stack),
            distance_thread,
        NULL, NULL, NULL, SENSOR_THREAD_PRIORITY, 0,
            K_NO_WAIT);
k_thread_name_set(&distance_thread_data, "
    distance_sensor");

manager_started = true;
```

```
}

bool sensor_manager_is_started(void)
{
        return manager_started;
}

void sensor_manager_get_snapshot(struct sensor_snapshot *
   snapshot)
{
        if (!snapshot) {
                return;
        }

        k_mutex_lock(&snapshot_lock, K_FOREVER);
        *snapshot = latest_snapshot;
        k_mutex_unlock(&snapshot_lock);
}
```

B.7 sensor_manager.h

```
#ifndef SENSOR_MANAGER_H
#define SENSOR_MANAGER_H

#include <stdbool.h>

struct sensor_snapshot {
        int temperature_tenths;
        int humidity_tenths;
        int water_temp_tenths;
        int ec_hundredths;
        int ph_hundredths;
        int light_hundredths;
        int distance_mm;
};

void sensor_manager_start(void);
bool sensor_manager_is_started(void);
void sensor_manager_get_snapshot(struct sensor_snapshot *
   snapshot);

#endif
```

B.8 wifi_manager.c

```
#include "wifi_manager.h"

#include <zephyr/kernel.h>
#include <zephyr/sys/printk.h>
```

```c
#include <zephyr/net/net_if.h>
#include <zephyr/net/wifi_mgmt.h>
#include <zephyr/net/net_mgmt.h>
#include <zephyr/net/dhcpv4.h>

#include <string.h>

#include "app_config.h"
#include "mqtt_manager.h"

static struct k_sem wifi_sem;
static struct k_sem ip_sem;
static int wifi_connect_status = -1;
static bool wifi_connected = false;
static bool ipv4_ready = false;
static bool wifi_connect_in_progress = false;
static struct net_mgmt_event_callback wifi_mgmt_cb;

static void log_ipv4_address(struct net_if *iface, const char
    *prefix)
{
    struct in_addr *addr = net_if_ipv4_get_global_addr(iface,
        NET_ADDR_PREFERRED);

    if (addr) {
        printk("%s: %d.%d.%d.%d\n",
               prefix,
               addr->s4_addr[0], addr->s4_addr[1],
               addr->s4_addr[2], addr->s4_addr[3]);
    } else {
        printk("%s\n", prefix);
    }
}

static bool wait_for_wifi_completed(struct net_if *iface, int
    timeout_s)
{
    struct wifi_iface_status status;

    for (int i = 0; i < timeout_s; i++) {
        memset(&status, 0, sizeof(status));

        if (net_mgmt(NET_REQUEST_WIFI_IFACE_STATUS, iface,
                     &status, sizeof(status)) == 0) {
            if (status.state == WIFI_STATE_COMPLETED) {
                wifi_connected = true;
                net_dhcpv4_start(iface);
                printk("DHCPv4 start requested\n");
                return true;
```

```
            }

            if ((i % 5) == 0) {
                printk("WiFi state: %d\n", status.state);
            }
        }

        k_sleep(K_SECONDS(1));
    }

    return false;
}

static bool wait_for_ipv4_address(struct net_if *iface, int
   timeout_s)
{
    for (int i = 0; i < timeout_s; i++) {
        struct in_addr *addr = net_if_ipv4_get_global_addr(
           iface, NET_ADDR_PREFERRED);
        if (addr != NULL) {
            ipv4_ready = true;
            return true;
        }

        k_sleep(K_SECONDS(1));
    }

    return false;
}

static bool wait_for_ipv4_ready(struct net_if *iface, int
   sem_timeout_s, int poll_timeout_s)
{
    /* Clear stale semaphore state from previous connection
       attempts. */
    while (k_sem_take(&ip_sem, K_NO_WAIT) == 0) {
    }

    /* IP may already be assigned before we start waiting for
        event. */
    if (wait_for_ipv4_address(iface, 0)) {
        return true;
    }

    if (k_sem_take(&ip_sem, K_SECONDS(sem_timeout_s)) == 0) {
        if (wait_for_ipv4_address(iface, 0)) {
            return true;
        }
    }
```

```
    return wait_for_ipv4_address(iface, poll_timeout_s);
}

static void wifi_mgmt_event_handler(struct
   net_mgmt_event_callback *cb,
                                    unsigned long long
                                       mgmt_event,
                                    struct net_if *iface)
{
    switch (mgmt_event) {
    case NET_EVENT_WIFI_CONNECT_RESULT:
    {
        const struct wifi_status *status = (const struct
           wifi_status *)cb->info;

        wifi_connect_in_progress = false;
        wifi_connect_status = status->status;
        if (wifi_connect_status == 0) {
            wifi_connected = true;
            net_dhcpv4_start(iface);
            printk("DHCPv4 start requested\n");
            printk("WiFi connect result: %d - CONNECTED\n",
               wifi_connect_status);
        } else {
            wifi_connected = false;
            printk("WiFi connect result: %d - FAILED\n",
               wifi_connect_status);
        }
        k_sem_give(&wifi_sem);
        break;
    }
    case NET_EVENT_WIFI_DISCONNECT_RESULT:
    {
        const struct wifi_status *status = (const struct
           wifi_status *)cb->info;

        wifi_connect_in_progress = false;
        wifi_connected = false;
        ipv4_ready = false;
        mqtt_manager_handle_wifi_disconnected();
        printk("WiFi DISCONNECTED! Reason: %d\n", status->
           disconn_reason);
        break;
    }
    case NET_EVENT_IPV4_ADDR_ADD:
        ipv4_ready = true;
        log_ipv4_address(iface, "IPv4 address assigned");
```

```
        // Log gateway to verify DNS will work
        struct net_if *iface_gw = net_if_get_default();
        if (iface_gw) {
            struct in_addr gw = net_if_ipv4_get_gw(iface_gw);
            if (gw.s_addr != 0) {
                printk("Gateway configured (DNS ready): %d.%d
                    .%d.%d\n",
                        gw.s_addr & 0xFF, (gw.s_addr >> 8) & 0
                            xFF,
                        (gw.s_addr >> 16) & 0xFF, (gw.s_addr
                            >> 24) & 0xFF);
            } else {
                printk("WARNING: Gateway not yet configured\n
                    ");
            }
        }

        k_sem_give(&ip_sem);
        break;
    default:
        break;
    }
}

void wifi_manager_init(void)
{
    k_sem_init(&wifi_sem, 0, 1);
    k_sem_init(&ip_sem, 0, 1);
    net_mgmt_init_event_callback(&wifi_mgmt_cb,
        wifi_mgmt_event_handler,
                                 NET_EVENT_WIFI_CONNECT_RESULT
                                     |
                                     NET_EVENT_WIFI_DISCONNECT_RESULT
                                         |
                                     NET_EVENT_IPV4_ADDR_ADD)
                                        ;
    net_mgmt_add_event_callback(&wifi_mgmt_cb);
}

int wifi_manager_connect(void)
{
    struct net_if *iface = net_if_get_default();
    if (!iface) {
        printk("ERROR: No network interface available\n");
        return -ENODEV;
    }

    struct wifi_connect_req_params wifi_params = {
        .ssid = WIFI_SSID,
```

```
        .ssid_length = strlen(WIFI_SSID),
        .psk = WIFI_PASSWORD,
        .psk_length = strlen(WIFI_PASSWORD),
        .channel = WIFI_CHANNEL_ANY,
        .security = WIFI_SECURITY_TYPE_PSK,
        .mfp = WIFI_MFP_OPTIONAL,
    };

    printk("WiFi parameters configured for SSID: %s\n",
       WIFI_SSID);
    printk("Connecting to WiFi SSID '%s'...\n", WIFI_SSID);

    wifi_connect_in_progress = true;

    int ret = net_mgmt(NET_REQUEST_WIFI_CONNECT, iface,
                       &wifi_params, sizeof(wifi_params));
    if (ret && ret != -EALREADY && ret != -EINPROGRESS) {
        wifi_connect_in_progress = false;
        printk("Failed to send WiFi connect request: %d\n",
           ret);
        return ret;
    }

    if (ret == -EALREADY || ret == -EINPROGRESS) {
        printk("WiFi connect already in progress (%d)\n", ret
           );
    }

    printk("WiFi connect request sent, waiting for response
       ...\n");
    if (!wait_for_wifi_completed(iface, 30)) {
        (void)net_mgmt(NET_REQUEST_WIFI_DISCONNECT, iface,
           NULL, 0);
        wifi_connect_in_progress = false;
        wifi_connected = false;
        printk("WiFi connection timed out\n");
        return -ETIMEDOUT;
    }

    wifi_connect_in_progress = false;
    printk("\n=== WIFI CONNECTED ===\n");
    printk("SSID: %s\n", WIFI_SSID);
    printk("Waiting for IP address assignment...\n");

    if (wait_for_ipv4_ready(iface, 20, 15)) {
        log_ipv4_address(iface, "IP assignment detected");
    } else {
        printk("IP assignment timeout\n");
```

```
        (void)net_mgmt(NET_REQUEST_WIFI_DISCONNECT, iface,
            NULL, 0);
        wifi_connected = false;
        ipv4_ready = false;
        return -ETIMEDOUT;
    }

    return 0;
}

void wifi_manager_reconnect(void)
{
    struct net_if *iface = net_if_get_default();
    if (!iface) {
        printk("ERROR: No network interface for reconnection\
            n");
        return;
    }

    if (wifi_connect_in_progress) {
        printk("WiFi connect already in progress, waiting...\
            n");
        return;
    }

    // Reset IP ready flag before reconnection attempt
    ipv4_ready = false;

    struct wifi_connect_req_params wifi_params = {
        .ssid = WIFI_SSID,
        .ssid_length = strlen(WIFI_SSID),
        .psk = WIFI_PASSWORD,
        .psk_length = strlen(WIFI_PASSWORD),
        .channel = WIFI_CHANNEL_ANY,
        .security = WIFI_SECURITY_TYPE_PSK,
        .mfp = WIFI_MFP_OPTIONAL,
    };

    printk("Attempting WiFi reconnection...\n");

    wifi_connect_in_progress = true;

    int ret = net_mgmt(NET_REQUEST_WIFI_CONNECT, iface,
                       &wifi_params, sizeof(wifi_params));
    if (ret && ret != -EALREADY && ret != -EINPROGRESS) {
        wifi_connect_in_progress = false;
        printk("Reconnection request failed: %d\n", ret);
    } else {
        if (ret == -EALREADY || ret == -EINPROGRESS) {
```

```
            printk("Reconnect already in progress (%d)\n",
                ret);
        }

        if (wait_for_wifi_completed(iface, 30)) {
            wifi_connect_in_progress = false;
            printk("WiFi reconnected successfully!\n");

            // IMPORTANT: Start DHCP and wait for IP address
                just like in wifi_manager_connect()
            printk("Waiting for IP address assignment after
                reconnect...\n");

            if (wait_for_ipv4_ready(iface, 20, 20)) {
                log_ipv4_address(iface, "IP assignment
                    detected after reconnect");
            } else {
                printk("IP assignment timeout after reconnect
                    \n");
                (void)net_mgmt(NET_REQUEST_WIFI_DISCONNECT,
                    iface, NULL, 0);
                wifi_connected = false;
                ipv4_ready = false;
            }
        } else {
            (void)net_mgmt(NET_REQUEST_WIFI_DISCONNECT, iface
                , NULL, 0);
            wifi_connect_in_progress = false;
            wifi_connected = false;
            printk("WiFi reconnection timed out\n");
        }
    }
}

int wifi_manager_check_connection(void)
{
    struct net_if *iface = net_if_get_default();
    if (!iface) {
        return -1;
    }

    /* First check: interface must be up */
    if (!net_if_is_up(iface)) {
        if (wifi_connected) {
            wifi_connected = false;
            ipv4_ready = false;
        }
        return -1;
    }
```

```
    /* Second check: local flag status */
    if (!wifi_connected) {
        return -1;
    }

    /* Third check: actively query WiFi status from driver */
    struct wifi_iface_status status;
    memset(&status, 0, sizeof(status));

    if (net_mgmt(NET_REQUEST_WIFI_IFACE_STATUS, iface, &
       status, sizeof(status)) == 0) {
        if (status.state != WIFI_STATE_COMPLETED) {
            wifi_connected = false;
            ipv4_ready = false;
            return -1;
        }
    }

    /* Fourth check: verify IP address still exists */
    struct in_addr *addr = net_if_ipv4_get_global_addr(iface,
        NET_ADDR_PREFERRED);
    if (!addr) {
        ipv4_ready = false;
        return -1;
    }

    return 0;
}

bool wifi_manager_is_connected(void)
{
    return wifi_connected;
}

bool wifi_manager_is_ipv4_ready(void)
{
    return ipv4_ready;
}

bool wifi_manager_is_network_fully_ready(void)
{
    // Check core readiness needed before MQTT/DNS attempts
    struct net_if *iface = net_if_get_default();
    if (!iface) {
        return false;
    }

    if (!wifi_connected || !ipv4_ready) {
```

```
        return false;
    }

    // Verify IP address is assigned
    struct in_addr *addr = net_if_ipv4_get_global_addr(iface,
        NET_ADDR_PREFERRED);
    if (!addr) {
        return false;
    }

    return true;
}
```

B.9 wifi_manager.h

```
#ifndef WIFI_MANAGER_H
#define WIFI_MANAGER_H

#include <stdbool.h>

void wifi_manager_init(void);
int wifi_manager_connect(void);
void wifi_manager_reconnect(void);
int wifi_manager_check_connection(void);
bool wifi_manager_is_connected(void);
bool wifi_manager_is_ipv4_ready(void);
bool wifi_manager_is_network_fully_ready(void);

#endif
```

B.10 mqtt_maneger.c

```
#include "mqtt_manager.h"

#include <zephyr/kernel.h>
#include <zephyr/sys/printk.h>
#include <zephyr/net/net_if.h>
#include <zephyr/net/mqtt.h>
#include <zephyr/net/socket.h>

#include <string.h>

#include "app_config.h"
#include "wifi_manager.h"
#include "sensor_manager.h"

static struct mqtt_client client_ctx;
static uint8_t rx_buffer[APP_MQTT_BUFFER_SIZE];
static uint8_t tx_buffer[APP_MQTT_BUFFER_SIZE];
```

```
static bool mqtt_connected = false;
static struct sockaddr_storage broker_storage;
static char mqtt_if_name[16];

static void format_scaled_value(char *buf, size_t buf_size,
   int value, int scale)
{
    if (value < 0 || scale <= 1) {
        (void)snprintf(buf, buf_size, "%d", value);
        return;
    }

    if (scale == 10) {
        (void)snprintf(buf, buf_size, "%d.%01d", value / 10,
           value % 10);
        return;
    }

    if (scale == 100) {
        (void)snprintf(buf, buf_size, "%d.%02d", value / 100,
            value % 100);
        return;
    }

    (void)snprintf(buf, buf_size, "%d", value);
}

static int probe_tcp_connect(const struct sockaddr *addr)
{
    int sock = zsock_socket(addr->sa_family, SOCK_STREAM,
       IPPROTO_TCP);
    if (sock < 0) {
        return -errno;
    }

    size_t addr_len = (addr->sa_family == AF_INET) ?
                          sizeof(struct sockaddr_in) : sizeof
                             (struct sockaddr_in6);

    int ret = zsock_connect(sock, addr, addr_len);
    int err = 0;

    if (ret < 0) {
        err = -errno;
    }

    (void)zsock_close(sock);
    return err;
}
```

```c
static void mqtt_evt_handler(struct mqtt_client *const client
    ,
                             const struct mqtt_evt *evt)
{
    ARG_UNUSED(client);

    switch (evt->type) {
    case MQTT_EVT_CONNACK:
        if (evt->result == 0) {
            mqtt_connected = true;
            printk("MQTT connected successfully!\n");
        } else {
            mqtt_connected = false;
            printk("MQTT connection failed: %d\n", evt->
               result);
        }
        break;
    case MQTT_EVT_DISCONNECT:
        mqtt_connected = false;
        printk("MQTT disconnected\n");
        break;
    default:
        break;
    }
}

int mqtt_manager_connect(void)
{
    mqtt_connected = false;

    // Clean up any previous MQTT connection before
       establishing a new one
    if (client_ctx.transport.tcp.sock >= 0) {
        printk("Closing previous MQTT connection\n");
        zsock_close(client_ctx.transport.tcp.sock);
        client_ctx.transport.tcp.sock = -1;
        k_sleep(K_MSEC(500));  // Brief delay to ensure
           socket closure
    }

    if (!wifi_manager_is_connected() || !
       wifi_manager_is_ipv4_ready()) {
        printk("Network not ready for MQTT (wifi=%d ipv4=%d)\
           n",
               wifi_manager_is_connected(),
                  wifi_manager_is_ipv4_ready());
        return -1;
    }
```

```
    // Wait for network to be FULLY ready (including gateway/
       DNS configuration)
    int wait_count = 0;
    while (!wifi_manager_is_network_fully_ready() &&
       wait_count < 20) {
        printk("Waiting for full network readiness (gateway +
            DNS)...\n");
        k_sleep(K_MSEC(500));
        wait_count++;
    }

    if (!wifi_manager_is_network_fully_ready()) {
        printk("Network failed to fully initialize (router/
           gateway not ready)\n");
        return -1;
    }

    printk("Network fully ready for DNS resolution\n");

    // Clear broker storage for fresh DNS resolution
    memset(&broker_storage, 0, sizeof(broker_storage));

    struct zsock_addrinfo hints = {
        .ai_family = AF_INET,
        .ai_socktype = SOCK_STREAM,
    };
    struct zsock_addrinfo *res;

    int ret = -1;
    for (int i = 0; i < 5; i++) {
        ret = zsock_getaddrinfo(MQTT_BROKER_HOST, NULL, &
           hints, &res);
        if (ret == 0) {
            printk("DNS resolution succeeded on attempt %d\n"
               , i + 1);
            break;
        }
        printk("Hostname resolve retry %d failed: %d ", i +
           1, ret);
        if (ret == -11) {
            printk("(EAGAIN - DNS not ready yet)\n");
        } else if (ret == -101) {
            printk("(-101 error)\n");
        } else {
            printk("\n");
        }
```

```
        // Longer wait between DNS attempts to give system
           time to stabilize
        k_sleep(K_SECONDS(3));
    }
    if (ret != 0) {
        printk("Failed to resolve hostname after all retries:
            %d\n", ret);
        return -1;
    }

    struct sockaddr_in *broker4 = (struct sockaddr_in *)&
       broker_storage;
    broker4->sin_family = AF_INET;
    broker4->sin_port = htons(MQTT_BROKER_PORT);
    memcpy(&broker4->sin_addr,
           &((struct sockaddr_in *)res->ai_addr)->sin_addr,
           sizeof(struct in_addr));
    zsock_freeaddrinfo(res);

    mqtt_client_init(&client_ctx);

    client_ctx.broker = (struct sockaddr *)&broker_storage;
    client_ctx.evt_cb = mqtt_evt_handler;

    client_ctx.client_id.utf8 = (uint8_t *)ACCESS_TOKEN;
    client_ctx.client_id.size = strlen(ACCESS_TOKEN);

    static struct mqtt_utf8 username;
    username.utf8 = (uint8_t *)ACCESS_TOKEN;
    username.size = strlen(ACCESS_TOKEN);
    client_ctx.user_name = &username;

    client_ctx.password = NULL;

    client_ctx.protocol_version = MQTT_VERSION_3_1_1;
    client_ctx.keepalive = 60;
    client_ctx.clean_session = 1;

    client_ctx.rx_buf = rx_buffer;
    client_ctx.rx_buf_size = sizeof(rx_buffer);
    client_ctx.tx_buf = tx_buffer;
    client_ctx.tx_buf_size = sizeof(tx_buffer);

    client_ctx.transport.type = MQTT_TRANSPORT_NON_SECURE;
    client_ctx.transport.tcp.sock = -1;

    struct net_if *iface = net_if_get_default();
    if (iface && net_if_get_name(iface, mqtt_if_name, sizeof(
       mqtt_if_name)) == 0) {
```

```
        client_ctx.transport.if_name = mqtt_if_name;
    } else {
        client_ctx.transport.if_name = NULL;
    }

    ret = probe_tcp_connect((struct sockaddr *)&
       broker_storage);
    if (ret < 0) {
        printk("TCP probe to broker failed: %d\n", ret);
        return ret;
    }

    ret = mqtt_connect(&client_ctx);
    if (ret != 0) {
        printk("MQTT connect failed: %d\n", ret);
        return -1;
    }

    for (int i = 0; i < 30 && !mqtt_connected; i++) {
        (void)mqtt_input(&client_ctx);
        (void)mqtt_live(&client_ctx);
        k_sleep(K_MSEC(100));
    }

    if (mqtt_connected) {
        printk("Connected to ThingsBoard MQTT broker\n");
        return 0;
    }

    printk("Failed to connect to MQTT broker\n");
    return -1;
}

void mqtt_manager_poll(void)
{
    if (mqtt_connected) {
        (void)mqtt_input(&client_ctx);
        (void)mqtt_live(&client_ctx);
    }
}

int mqtt_manager_publish_sensor_data(void)
{
    if (!mqtt_connected) {
        return -1;
    }

    static int counter = 0;
    struct sensor_snapshot snapshot;
```

```
if (!sensor_manager_is_started()) {
    return -EAGAIN;
}

sensor_manager_get_snapshot(&snapshot);

char temperature_str[16];
char humidity_str[16];
char water_temp_str[16];
char ec_str[16];
char ph_str[16];
char light_str[16];
char distance_str[16];

format_scaled_value(temperature_str, sizeof(
   temperature_str), snapshot.temperature_tenths, 10);
format_scaled_value(humidity_str, sizeof(humidity_str),
   snapshot.humidity_tenths, 10);
format_scaled_value(water_temp_str, sizeof(water_temp_str
   ), snapshot.water_temp_tenths, 10);
format_scaled_value(ec_str, sizeof(ec_str), snapshot.
   ec_hundredths, 100);
format_scaled_value(ph_str, sizeof(ph_str), snapshot.
   ph_hundredths, 100);
format_scaled_value(light_str, sizeof(light_str),
   snapshot.light_hundredths, 100);
format_scaled_value(distance_str, sizeof(distance_str),
   snapshot.distance_mm, 10);

char payload[256];
int payload_len = snprintf(payload, sizeof(payload),
                           "{\"temperature\": %s, \"
                              humidity\": %s, "
                           "\"water_temp\": %s, \"ec\": %
                              s, "
                           "\"pH\": %s, \"light\": %s, \"
                              distance\": %s}",
                           temperature_str, humidity_str,
                               water_temp_str,
                           ec_str, ph_str, light_str,
                              distance_str);

if (payload_len < 0 || payload_len >= (int)sizeof(payload
   )) {
    printk("Payload truncated!\n");
    return -EMSGSIZE;
}
```

```
    struct mqtt_publish_param param = {
        .message.topic.qos = MQTT_QOS_0_AT_MOST_ONCE,
        .message.topic.topic.utf8 = (uint8_t *)"v1/devices/me
           /telemetry",
        .message.topic.topic.size = strlen("v1/devices/me/
           telemetry"),
        .message.payload.data = payload,
        .message.payload.len = payload_len,
        .message_id = counter++,
        .dup_flag = 0,
        .retain_flag = 0,
    };

    int ret = mqtt_publish(&client_ctx, &param);
    if (ret == 0) {
        printk("Published sensor data:\n%s\n\n", payload);
    } else {
        printk("Failed to publish sensor data: %d\n", ret);
    }

    return ret;
}

int mqtt_manager_activate_device(void)
{
    if (!mqtt_connected) {
        return -1;
    }

    char payload[] = "{\"active\": true}";

    struct mqtt_publish_param param = {
        .message.topic.qos = MQTT_QOS_0_AT_MOST_ONCE,
        .message.topic.topic.utf8 = (uint8_t *)"v1/devices/me
           /attributes",
        .message.topic.topic.size = strlen("v1/devices/me/
           attributes"),
        .message.payload.data = payload,
        .message.payload.len = strlen(payload),
        .message_id = 999,
        .dup_flag = 0,
        .retain_flag = 0,
    };

    int ret = mqtt_publish(&client_ctx, &param);
    if (ret == 0) {
        printk("Device activated in ThingsBoard\n");
    } else {
        printk("Failed to activate device: %d\n", ret);
```

```
    }

    return ret;
}

bool mqtt_manager_is_connected(void)
{
    return mqtt_connected;
}

void mqtt_manager_handle_wifi_disconnected(void)
{
    mqtt_connected = false;

    // Properly close MQTT socket when WiFi disconnects
    if (client_ctx.transport.tcp.sock >= 0) {
        printk("Closing MQTT socket due to WiFi disconnect\n"
           );
        zsock_close(client_ctx.transport.tcp.sock);
        client_ctx.transport.tcp.sock = -1;
    }
}
```

B.11 mqtt_manager.h

```
#ifndef MQTT_MANAGER_H
#define MQTT_MANAGER_H

#include <stdbool.h>

int mqtt_manager_connect(void);
void mqtt_manager_poll(void);
int mqtt_manager_publish_sensor_data(void);
int mqtt_manager_activate_device(void);
bool mqtt_manager_is_connected(void);
void mqtt_manager_handle_wifi_disconnected(void);

#endif
```

B.12 I2C_buss_lock.c

```
#include "i2c_bus_lock.h"

#include <zephyr/kernel.h>

static K_MUTEX_DEFINE(i2c_lock);

void i2c_bus_lock(void)
{
```

```
    k_mutex_lock(&i2c_lock, K_FOREVER);
}

void i2c_bus_unlock(void)
{
    k_mutex_unlock(&i2c_lock);
}
```

B.13 I2C_buss_lock.h

```
#ifndef I2C_BUS_LOCK_H
#define I2C_BUS_LOCK_H

void i2c_bus_lock(void);
void i2c_bus_unlock(void);

#endif
```

B.14 temprature_and_humidity_sensor.c

```
#include "temperature_and_humidity_sensor.h"

#include <zephyr/kernel.h>
#include <zephyr/device.h>
#include <zephyr/devicetree.h>
#include <zephyr/drivers/i2c.h>
#include <errno.h>

#include "app_config.h"
#include "i2c_bus_lock.h"

/* SHT31 possible I2C addresses and commands */
#define SHT31_I2C_ADDR_LOW  0x44
#define SHT31_I2C_ADDR_HIGH 0x45
#define SHT31_MEAS_HIGHREP 0x2400
#define SHT31_SOFT_RESET 0x30A2

#define SHT31_RESPONSE_LEN 6
#define SHT31_INIT_RETRIES 3

static const struct device *i2c_dev;
static bool sensor_available;
static uint16_t sht31_i2c_addr = SHT31_I2C_ADDR_LOW;
static int last_temperature_tenths;
static int last_humidity_tenths;

static uint8_t crc8(const uint8_t *data, int len)
{
    uint8_t crc = 0xFF;
```

```
    for (int i = 0; i < len; i++) {
        crc ^= data[i];
        for (int j = 0; j < 8; j++) {
            if (crc & 0x80) {
                crc = (crc << 1) ^ 0x31;
            } else {
                crc = (crc << 1);
            }
        }
    }
    return crc;
}

static int sht31_send_command(uint16_t cmd)
{
    uint8_t buf[2];
    buf[0] = (uint8_t)((cmd >> 8) & 0xFF);
    buf[1] = (uint8_t)(cmd & 0xFF);

    return i2c_write(i2c_dev, buf, sizeof(buf),
       sht31_i2c_addr);
}

static int sht31_read_data(uint8_t *data, uint8_t len)
{
    return i2c_read(i2c_dev, data, len, sht31_i2c_addr);
}

static int sht31_recover_bus(void)
{
    if (!i2c_dev) {
        return -ENODEV;
    }

    return i2c_recover_bus(i2c_dev);
}

static int sht31_validate_measurement(void)
{
    uint8_t response[SHT31_RESPONSE_LEN];
    int ret;

    i2c_bus_lock();

    ret = sht31_send_command(SHT31_MEAS_HIGHREP);
    if (ret != 0) {
        i2c_bus_unlock();
        return -EIO;
    }
```

```
    k_sleep(K_MSEC(20));

    ret = sht31_read_data(response, sizeof(response));
    i2c_bus_unlock();

    if (ret != 0) {
        return -EIO;
    }

    if (crc8(&response[0], 2) != response[2] || crc8(&
       response[3], 2) != response[5]) {
        return -EIO;
    }

    return 0;
}

static int sht31_try_init_at_addr(uint16_t addr)
{
    sht31_i2c_addr = addr;

    /* Some power-up sequences fail STATUS probe even when
       measurement works. */
    (void)sht31_send_command(SHT31_SOFT_RESET);
    k_sleep(K_MSEC(40));

    for (int i = 0; i < 2; i++) {
        if (sht31_validate_measurement() == 0) {
            return 0;
        }
        k_sleep(K_MSEC(50));
    }

    return -ENODEV;
}

int sht31_init(void)
{
    i2c_dev = DEVICE_DT_GET(DT_NODELABEL(i2c0));
    sensor_available = false;

    if (!i2c_dev || !device_is_ready(i2c_dev)) {
        return -ENODEV;
    }

    for (int i = 0; i < SHT31_INIT_RETRIES; i++) {
        (void)sht31_recover_bus();
```

```
        if (sht31_try_init_at_addr(SHT31_I2C_ADDR_LOW) == 0
           ||
            sht31_try_init_at_addr(SHT31_I2C_ADDR_HIGH) == 0)
                {
            sensor_available = true;
            return 0;
        }

        k_sleep(K_MSEC(120));
    }

    return -ENODEV;
}

static int sht31_perform_read(void)
{
    int cmd_ret;

    if (!sensor_available) {
        if (sht31_init() != 0) {
            return -ENODEV;
        }
    }

    i2c_bus_lock();
    cmd_ret = sht31_send_command(SHT31_MEAS_HIGHREP);
    i2c_bus_unlock();

    if (cmd_ret != 0) {
        sensor_available = false;
        i2c_bus_lock();
        (void)sht31_recover_bus();
        i2c_bus_unlock();

        if (sht31_init() != 0) {
            return -EIO;
        }

        i2c_bus_lock();
        cmd_ret = sht31_send_command(SHT31_MEAS_HIGHREP);
        i2c_bus_unlock();
        if (cmd_ret != 0) {
            return -EIO;
        }
    }

    k_sleep(K_MSEC(20));

    uint8_t response[SHT31_RESPONSE_LEN];
```

```
    i2c_bus_lock();
    int read_ret = sht31_read_data(response, sizeof(response)
       );
    i2c_bus_unlock();

    if (read_ret != 0) {
        sensor_available = false;
        i2c_bus_lock();
        (void)sht31_recover_bus();
        i2c_bus_unlock();

        if (sht31_init() != 0) {
            return -EIO;
        }

        i2c_bus_lock();
        read_ret = sht31_read_data(response, sizeof(response)
           );
        i2c_bus_unlock();
        if (read_ret != 0) {
            return -EIO;
        }
    }

    if (crc8(&response[0], 2) != response[2] || crc8(&
       response[3], 2) != response[5]) {
        sensor_available = false;
        i2c_bus_lock();
        (void)sht31_recover_bus();
        i2c_bus_unlock();
        if (sht31_init() != 0) {
            return -EIO;
        }

        i2c_bus_lock();
        cmd_ret = sht31_send_command(SHT31_MEAS_HIGHREP);
        i2c_bus_unlock();
        if (cmd_ret != 0) {
            sensor_available = false;
            return -EIO;
        }
        k_sleep(K_MSEC(20));

        i2c_bus_lock();
        read_ret = sht31_read_data(response, sizeof(response)
           );
        i2c_bus_unlock();
        if (read_ret != 0) {
            sensor_available = false;
```

```
            return -EIO;
        }
        if (crc8(&response[0], 2) != response[2] || crc8(&
           response[3], 2) != response[5]) {
            sensor_available = false;
            return -EIO;
        }
    }

    uint16_t raw_temp = (uint16_t)((response[0] << 8) |
       response[1]);
    uint16_t raw_humidity = (uint16_t)((response[3] << 8) |
       response[4]);

    last_temperature_tenths = -450 + (int)((1750U * raw_temp)
        / 65535U);
    last_humidity_tenths = (int)((1000U * raw_humidity) /
       65535U);

    return 0;
}

int sht31_read_temperature_and_humidity_tenths(int *
   temperature_tenths,
                                               int *
                                                  humidity_tenths
                                                  )
{
    if (!temperature_tenths || !humidity_tenths) {
        return -EINVAL;
    }

    if (sht31_perform_read() != 0) {
        *temperature_tenths = APP_FALLBACK_AIR_TEMP_TENTHS;
        *humidity_tenths = APP_FALLBACK_AIR_HUMIDITY_TENTHS;
        return -EIO;
    }

    *temperature_tenths = last_temperature_tenths;
    *humidity_tenths = last_humidity_tenths;
    return 0;
}

int sht31_read_temperature_tenths(void)
{
    if (sht31_perform_read() != 0) {
        return APP_FALLBACK_AIR_TEMP_TENTHS;
    }
    return last_temperature_tenths;
```

```
}

int sht31_read_humidity_tenths(void)
{
    if (sht31_perform_read() != 0) {
        return APP_FALLBACK_AIR_HUMIDITY_TENTHS;
    }
    return last_humidity_tenths;
}

bool sht31_is_available(void)
{
    return sensor_available;
}

int param_temperature_tenths(int counter)
{
    (void)counter;
    return sht31_read_temperature_tenths();
}

int param_humidity_tenths(int counter)
{
    (void)counter;
    return sht31_read_humidity_tenths();
}
```

B.15 temprature_and_humidity_sensor.h

```
#ifndef TEMPERATURE_AND_HUMIDITY_SENSOR_H
#define TEMPERATURE_AND_HUMIDITY_SENSOR_H

#include <stdint.h>
#include <stdbool.h>

/* Initialize SHT31 sensor on I2C bus */
int sht31_init(void);

/* Read temperature in tenths of degree (e.g. 250 = 25.0C) */
int sht31_read_temperature_tenths(void);

/* Read humidity in tenths of percent (e.g. 550 = 55.0%) */
int sht31_read_humidity_tenths(void);

/* Read both values from one SHT31 measurement cycle. */
int sht31_read_temperature_and_humidity_tenths(int *
   temperature_tenths,
        int *humidity_tenths);
```

```c
/* App parameter wrappers for temperature/humidity in tenths.
    */
int param_temperature_tenths(int counter);
int param_humidity_tenths(int counter);

/* Check if sensor is initialized and available */
bool sht31_is_available(void);

#endif
```

B.16 water_temp_param.c

```c
#include "water_temp_param.h"
#include "app_config.h"

#include <zephyr/device.h>
#include <zephyr/devicetree.h>
#include <zephyr/drivers/sensor.h>
#include <zephyr/sys/printk.h>

static const struct device *get_ds18b20_device(void)
{
    static const struct device *dev;
    static bool initialized;

    if (!initialized) {
        initialized = true;
        dev = DEVICE_DT_GET_ANY(maxim_ds18b20);
        if (!dev || !device_is_ready(dev)) {
            dev = NULL;
        }
    }

    return dev;
}

int param_water_temp_tenths(int counter)
{
    (void)counter;

    const struct device *dev = get_ds18b20_device();
    if (!dev) {
        return APP_FALLBACK_WATER_TEMP_TENTHS;
    }

    struct sensor_value temp;
    int ret = sensor_sample_fetch(dev);
    if (ret != 0) {
        return APP_FALLBACK_WATER_TEMP_TENTHS;
```

```
    }

    ret = sensor_channel_get(dev, SENSOR_CHAN_AMBIENT_TEMP, &
       temp);
    if (ret != 0) {
        return APP_FALLBACK_WATER_TEMP_TENTHS;
    }

    int64_t micro_c = (int64_t)temp.val1 * 1000000LL + temp.
       val2;
    int water_temp_tenths = (int)(micro_c / 100000LL);

    if (water_temp_tenths == 0) {
        return APP_FALLBACK_WATER_TEMP_TENTHS;
    }

    return water_temp_tenths;
}
```

B.17 water_temp_param.h

```
#ifndef WATER_TEMP_PARAM_H
#define WATER_TEMP_PARAM_H

int param_water_temp_tenths(int counter);

#endif
```

B.18 EC_param.c

```
#include "ec_param.h"
#include "water_temp_param.h"
#include "app_config.h"

#include <zephyr/kernel.h>
#include <zephyr/device.h>
#include <zephyr/devicetree.h>
#include <zephyr/drivers/adc.h>
#include <zephyr/sys/printk.h>

#define TDS_ADC_CHANNEL    7   /* GPIO35 = ADC1 channel 7 on
   ESP32 */
#define TDS_ADC_RESOLUTION 12
#define TDS_SAMPLES        10

/* Empirical calibration points (raw ADC -> EC mS/cm) from
   standard solutions. */
#define EC_CAL_RAW_0 420
#define EC_CAL_RAW_1 1600
```

```
#define EC_CAL_RAW_2 2193
#define EC_CAL_RAW_3 2300

#define EC_CAL_MS_0  0.0f
#define EC_CAL_MS_1  1.7f
#define EC_CAL_MS_2  3.4f
#define EC_CAL_MS_3  5.0f

static const struct device *adc_dev;
static bool adc_ready;

static void init_adc(void)
{
    if (adc_ready) {
        return;
    }

    adc_dev = DEVICE_DT_GET(DT_NODELABEL(adc0));
    if (!device_is_ready(adc_dev)) {
        printk("EC: ADC device not ready\n");
        return;
    }

    struct adc_channel_cfg ch_cfg = {

        .gain             = ADC_GAIN_1_4,
        .reference        = ADC_REF_INTERNAL,
        .acquisition_time = ADC_ACQ_TIME_DEFAULT,
        .channel_id       = TDS_ADC_CHANNEL,
    };

    if (adc_channel_setup(adc_dev, &ch_cfg) != 0) {
        printk("EC: ADC channel setup failed\n");
        return;
    }

    adc_ready = true;
}

static int read_adc_average(void)
{
    int16_t sample;
    struct adc_sequence seq = {
        .channels    = BIT(TDS_ADC_CHANNEL),
        .buffer      = &sample,
        .buffer_size = sizeof(sample),
        .resolution  = TDS_ADC_RESOLUTION,
    };
```

```
    long sum = 0;

    for (int i = 0; i < TDS_SAMPLES; i++) {
        if (adc_read(adc_dev, &seq) != 0) {
            return -1;
        }
        sum += sample;
        k_msleep(10);
    }

    return (int)(sum / TDS_SAMPLES);
}

static float linear_interp(int x, int x0, int x1, float y0,
   float y1)
{
    if (x1 <= x0) {
        return y0;
    }

    float ratio = (float)(x - x0) / (float)(x1 - x0);
    return y0 + ratio * (y1 - y0);
}

static float ec_from_raw_calibrated(int raw)
{
    if (raw <= EC_CAL_RAW_0) {
        return EC_CAL_MS_0;
    }

    if (raw <= EC_CAL_RAW_1) {
        return linear_interp(raw, EC_CAL_RAW_0, EC_CAL_RAW_1,
            EC_CAL_MS_0, EC_CAL_MS_1);
    }

    if (raw <= EC_CAL_RAW_2) {
        return linear_interp(raw, EC_CAL_RAW_1, EC_CAL_RAW_2,
            EC_CAL_MS_1, EC_CAL_MS_2);
    }

    if (raw <= EC_CAL_RAW_3) {
        return linear_interp(raw, EC_CAL_RAW_2, EC_CAL_RAW_3,
            EC_CAL_MS_2, EC_CAL_MS_3);
    }

    return linear_interp(raw, EC_CAL_RAW_2, EC_CAL_RAW_3,
       EC_CAL_MS_2, EC_CAL_MS_3);
}
```

```c
/* Returns EC in hundredths of mS/cm, temperature-compensated
    to 25 C.
 * Example: 123 means 1.23 mS/cm. */
int param_ec_hundredths(int counter)
{
    (void)counter;

    init_adc();
    if (!adc_ready) {
        return APP_FALLBACK_EC_HUNDREDTHS;
    }

    int raw = read_adc_average();

    if (raw < 0) {
        return APP_FALLBACK_EC_HUNDREDTHS;
    }

    int water_temp_tenths = param_water_temp_tenths(0);
    float water_temp = (water_temp_tenths >= 0) ? ((float)
       water_temp_tenths / 10.0f) : 25.0f;

    /* Temperature compensation to 25 C. */
    float compensation_coefficient = 1.0f + 0.02f * (
       water_temp - 25.0f);
    if (compensation_coefficient <= 0.0f) {
        compensation_coefficient = 1.0f;
    }

    float ec_measured = ec_from_raw_calibrated(raw);
    float ec_compensated = ec_measured /
       compensation_coefficient;

    int ec_hundredths = (int)(ec_compensated * 100.0f + 0.5f)
       ;

    if (ec_hundredths == 0) {
        return APP_FALLBACK_EC_HUNDREDTHS;
    }

    return ec_hundredths;
}
```

B.19 EC_param.h

```c
#ifndef EC_PARAM_H
#define EC_PARAM_H

int param_ec_hundredths(int counter);
```

```
#endif
```

B.20 ph_param.c

```
#include "ph_param.h"
#include "app_config.h"

#include <zephyr/kernel.h>
#include <zephyr/device.h>
#include <zephyr/devicetree.h>
#include <zephyr/drivers/adc.h>
#include <zephyr/sys/printk.h>

#define PH_ADC_CHANNEL      5   /* GPIO33 = ADC1 channel 5 on
    ESP32 */
#define PH_ADC_RESOLUTION   12
#define PH_SAMPLES          10

/* Empirical calibration points (raw ADC -> pH) from standard
    solutions. */
#define PH_CAL_RAW_4  2258  /* Average of the measured pH 4
   readings (2255-2260). */
#define PH_CAL_RAW_7  1773
#define PH_CAL_RAW_9  1482

#define PH_CAL_PH_4   4.0f
#define PH_CAL_PH_7   7.0f
#define PH_CAL_PH_9   9.0f

static const struct device *adc_dev;
static bool adc_ready;

static void init_adc(void)
{
    if (adc_ready) {
        return;
    }

    adc_dev = DEVICE_DT_GET(DT_NODELABEL(adc0));
    if (!device_is_ready(adc_dev)) {
        return;
    }

    struct adc_channel_cfg ch_cfg = {
        /* Match Arduino ADC_11db style range for GPIO33
           (~0-3.3V). */
        .gain = ADC_GAIN_1_4,
        .reference = ADC_REF_INTERNAL,
```

```
        .acquisition_time = ADC_ACQ_TIME_DEFAULT,
        .channel_id = PH_ADC_CHANNEL,
    };

    if (adc_channel_setup(adc_dev, &ch_cfg) != 0) {
        return;
    }

    adc_ready = true;
}

static int read_adc_average(void)
{
    int16_t sample;
    struct adc_sequence seq = {
        .channels = BIT(PH_ADC_CHANNEL),
        .buffer = &sample,
        .buffer_size = sizeof(sample),
        .resolution = PH_ADC_RESOLUTION,
    };

    long sum = 0;

    for (int i = 0; i < PH_SAMPLES; i++) {
        if (adc_read(adc_dev, &seq) != 0) {
            return -1;
        }
        sum += sample;
        k_msleep(10);
    }

    return (int)(sum / PH_SAMPLES);
}

static float linear_interp(int x, int x0, int x1, float y0,
   float y1)
{
    if (x1 <= x0) {
        return y0;
    }

    float ratio = (float)(x - x0) / (float)(x1 - x0);
    return y0 + ratio * (y1 - y0);
}

static float ph_from_raw_calibrated(int raw)
{
    /* Interpolate/extrapolate in the pH 4-7 range and beyond
       . */
```

```
    if (raw >= PH_CAL_RAW_7) {
        return linear_interp(raw, PH_CAL_RAW_7, PH_CAL_RAW_4,
            PH_CAL_PH_7, PH_CAL_PH_4);
    }

    /* Interpolate between pH 7 and 9. */
    if (raw >= PH_CAL_RAW_9) {
        return linear_interp(raw, PH_CAL_RAW_9, PH_CAL_RAW_7,
            PH_CAL_PH_9, PH_CAL_PH_7);
    }

    /* Extrapolate above pH 9 (raw < PH_CAL_RAW_9) using the
       pH 7-9 slope. */
    return linear_interp(raw, PH_CAL_RAW_9, PH_CAL_RAW_7,
       PH_CAL_PH_9, PH_CAL_PH_7);
}

/* Returns pH in hundredths using a piecewise-linear
   calibration curve.
 * Example: 723 means pH 7.23.
 */
int param_ph_hundredths(int counter)
{
    (void)counter;

    init_adc();
    if (!adc_ready) {
        return APP_FALLBACK_PH_HUNDREDTHS;
    }

    int raw = read_adc_average();
    if (raw < 0) {
        return APP_FALLBACK_PH_HUNDREDTHS;
    }

    float ph_value = ph_from_raw_calibrated(raw);
    int ph_hundredths = (int)(ph_value * 100.0f + 0.5f);

    if (ph_hundredths == 0 || ph_hundredths > 1400) {
        return APP_FALLBACK_PH_HUNDREDTHS;
    }

    return ph_hundredths;
}
```

B.21 ph_param.h

```
#ifndef PH_PARAM_H
#define PH_PARAM_H
```

```c
int param_ph_hundredths(int counter);

#endif
```

B.22 light_param.c

```c
#include "light_param.h"
#include "app_config.h"

#include <zephyr/device.h>
#include <zephyr/drivers/sensor.h>
#include <zephyr/drivers/sensor/tsl2591.h>
#include <zephyr/kernel.h>
#include <zephyr/sys/printk.h>

#include <errno.h>
#include <stdint.h>

#include "i2c_bus_lock.h"

/* Match Arduino reference behavior. */
#define LIGHT_CALIB_FACTOR_X100 175

static const struct device *light_dev;
static bool light_ready;

static int map_lux_to_hundredths(const struct sensor_value *
   lux)
{
    int64_t hundredths = ((int64_t)lux->val1 * 100LL) + ((
       int64_t)lux->val2 / 10000LL);

    if (hundredths < 0) {
        return 0;
    }

    return (int)hundredths;
}

static void init_light_sensor(void)
{
    if (light_ready) {
        return;
    }

    light_dev = DEVICE_DT_GET(DT_NODELABEL(tsl2591));
    if (!device_is_ready(light_dev)) {
        return;
```

```
    }

    /* LOW gain is more robust in strong light. */
    struct sensor_value gain = {
        .val1 = TSL2591_SENSOR_GAIN_LOW,
        .val2 = 0,
    };
    i2c_bus_lock();
    int ret = sensor_attr_set(light_dev, SENSOR_CHAN_LIGHT,
       SENSOR_ATTR_GAIN_MODE, &gain);
    i2c_bus_unlock();
    if (ret != 0) {
        return;
    }

    /* Short integration time reduces saturation risk. */
    struct sensor_value integration = {
        .val1 = 100,
        .val2 = 0,
    };
    i2c_bus_lock();
    ret = sensor_attr_set(light_dev, SENSOR_CHAN_LIGHT,
       SENSOR_ATTR_INTEGRATION_TIME,
                          &integration);
    i2c_bus_unlock();
    if (ret != 0) {
        printk("Light: failed to set TSL2591 integration time
           \n");
        return;
    }

    light_ready = true;
}

static int light_read_lux_hundredths(void)
{
    i2c_bus_lock();
    int ret = sensor_sample_fetch_chan(light_dev,
       SENSOR_CHAN_ALL);
    if (ret != 0) {
        i2c_bus_unlock();
        return -EIO;
    }

    struct sensor_value lux;
    ret = sensor_channel_get(light_dev, SENSOR_CHAN_ALL, &lux
       );
    i2c_bus_unlock();
```

```
    if (ret == -EOVERFLOW) {
        /* Saturated sensor reading: treat as invalid for
           this cycle. */
        return -EOVERFLOW;
    }

    if (ret != 0) {
        return -EIO;
    }

    int64_t raw_hundredths = map_lux_to_hundredths(&lux);
    int64_t calibrated = (raw_hundredths *
       LIGHT_CALIB_FACTOR_X100) / 100LL;

    if (calibrated < 0) {
        calibrated = 0;
    }

    return (int)calibrated;
}

int param_light_hundredths(int counter)
{
    (void)counter;

    init_light_sensor();
    if (!light_ready) {
        return APP_FALLBACK_LIGHT_RAW;
    }

    int lux_hundredths = light_read_lux_hundredths();
    if (lux_hundredths == -EOVERFLOW) {
        return APP_FALLBACK_LIGHT_RAW;
    }

    if (lux_hundredths < 0) {
        printk("Light: TSL2591 read failed\n");
        light_ready = false;
        return APP_FALLBACK_LIGHT_RAW;
    }

    return lux_hundredths;
}

int param_light(int counter)
{
    return param_light_hundredths(counter) / 100;
}
```

B.23 light_param.h

```
#ifndef LIGHT_PARAM_H
#define LIGHT_PARAM_H

int param_light_hundredths(int counter);
int param_light(int counter);

#endif
```

B.24 distance_param.c

```
#include "distance_param.h"
#include "app_config.h"

#include <zephyr/device.h>
#include <zephyr/devicetree.h>
#include <zephyr/drivers/uart.h>
#include <zephyr/sys/printk.h>

#define DISTANCE_MAX_BYTES_PER_CALL 64

static const struct device *get_distance_uart(void)
{
    static const struct device *uart_dev;
    static bool initialized;

    if (!initialized) {
        initialized = true;
        uart_dev = DEVICE_DT_GET(DT_NODELABEL(uart2));
        if (!device_is_ready(uart_dev)) {
            uart_dev = NULL;
        }
    }

    return uart_dev;
}

int param_distance(int counter)
{
    (void)counter;

    const struct device *uart_dev = get_distance_uart();
    static int last_valid_mm = APP_FALLBACK_DISTANCE_MM;

    if (!uart_dev) {
        printk("Distance fallback active (UART2 unavailable)\
           n");
        return last_valid_mm;
    }
```

```
    uint8_t byte;
    uint8_t frame[4] = {0};
    int idx = 0;
    int bytes_processed = 0;
    bool got_valid_frame = false;

    while (bytes_processed < DISTANCE_MAX_BYTES_PER_CALL &&
       uart_poll_in(uart_dev, &byte) == 0) {
        bytes_processed++;

        if (idx == 0) {
            if (byte != 0xFF) {
                continue;
            }
            frame[idx++] = byte;
            continue;
        }

        frame[idx++] = byte;
        if (idx < 4) {
            continue;
        }

        idx = 0;

        uint8_t sum = (uint8_t)((frame[0] + frame[1] + frame
           [2]) & 0xFF);
        if (sum != frame[3]) {
            continue;
        }

        int distance_mm = (frame[1] << 8) | frame[2];
        if (distance_mm > 30) {
            last_valid_mm = distance_mm;
            got_valid_frame = true;
        }
    }

    if (!got_valid_frame) {
        return APP_FALLBACK_DISTANCE_MM;
    }

    return last_valid_mm;
}
```

B.25 distance_param.h

```
#ifndef DISTANCE_PARAM_H
```

```
#define DISTANCE_PARAM_H

int param_distance(int counter);

#endif
```

C

Kod för egenutvecklade ThingsBoard-komponenter och rule chain

Här redovisas koden för de egenutvecklade komponenterna i ThingsBoard.

C.1 HTML för pH

```
    <div class="ph-widget">
  <div id="phValue" class="ph-value">--</div>

  <div class="ph-scale">
    <div class="ph-segment ph-0"><span>0</span></div>
    <div class="ph-segment ph-1"><span>1</span></div>
    <div class="ph-segment ph-2"><span>2</span></div>
    <div class="ph-segment ph-3"><span>3</span></div>
    <div class="ph-segment ph-4"><span>4</span></div>
    <div class="ph-segment ph-5"><span>5</span></div>
    <div class="ph-segment ph-6"><span>6</span></div>
    <div class="ph-segment ph-7"><span>7</span></div>
    <div class="ph-segment ph-8"><span>8</span></div>
    <div class="ph-segment ph-9"><span>9</span></div>
    <div class="ph-segment ph-10"><span>10</span></div>
    <div class="ph-segment ph-11"><span>11</span></div>
    <div class="ph-segment ph-12"><span>12</span></div>
    <div class="ph-segment ph-13"><span>13</span></div>
    <div class="ph-segment ph-14"><span>14</span></div>
  </div>

  <div id="phPointer" class="ph-pointer"></div>
</div>
```

C.2 CSS för pH

```
    .ph-widget {
  width: 100%;
  height: 100%;
  padding: 25px 10px 10px 10px;
  box-sizing: border-box;
  font-family: Roboto, Arial, sans-serif;
  position: relative;
}

.ph-value {
```

```
  text-align: center;
  font-size: 42px;
  font-weight: 700;
  color: red;
  margin-bottom: 20px;
}

.ph-scale {
  display: flex;
  width: 100%;
  height: 130px;
  border-radius: 8px;
  overflow: hidden;
}

.ph-segment {
  flex: 1;
  position: relative;
  display: flex;
  align-items: flex-end;
  justify-content: center;
  color: white;
  font-size: 14px;
  font-weight: 600;
  padding-bottom: 8px;
  box-sizing: border-box;
}

.ph-pointer {
  position: absolute;
  bottom: 0px;
  width: 0;
  height: 0;
  border-left: 10px solid transparent;
  border-right: 10px solid transparent;
  border-bottom: 18px solid #444;
  transform: translateX(-50%);
  transition: left 0.3s ease;
}

/* pH-colours */
.ph-0  { background: #ff0000; }
.ph-1  { background: #ff2a00; }
.ph-2  { background: #ff5a00; }
.ph-3  { background: #ff9500; }
.ph-4  { background: #ffc400; }
.ph-5  { background: #c8ff00; }
.ph-6  { background: #55ff22; }
.ph-7  { background: #06c85f; }
```

```
.ph-8  { background: #0bc78e; }
.ph-9  { background: #12c2c0; }
.ph-10 { background: #109bbc; }
.ph-11 { background: #126cc4; }
.ph-12 { background: #3836c7; }
.ph-13 { background: #6933c6; }
.ph-14 { background: #a032c7; }
```

C.3 Javascript för pH

```
  self.onInit = function() {
    self.updatePhValue();
};

self.onDataUpdated = function() {
    self.updatePhValue();
};

self.updatePhValue = function() {
    var phValueElement = self.ctx.$container.find('#phValue')
       ;
    var phPointerElement = self.ctx.$container.find('#
       phPointer');

    var ph = 0;

    if (self.ctx.data && self.ctx.data.length > 0) {
        var dataKey = self.ctx.data[0];

        if (dataKey.data && dataKey.data.length > 0) {
            var latestData = dataKey.data[dataKey.data.length
                - 1];
            ph = parseFloat(latestData[1]);
        }
    }

    if (isNaN(ph)) {
        ph = 0;
    }

    if (ph < 0) {
        ph = 0;
    }

    if (ph > 14) {
        ph = 14;
    }

    phValueElement.text(ph.toFixed(2));
```

```
    var positionPercent = (ph / 14) * 100;
    phPointerElement.css('left', positionPercent + '%');

    if (ph < 6.5) {
        phValueElement.css('color', 'red');
    } else if (ph >= 6.5 && ph <= 7.5) {
        phValueElement.css('color', '#06c85f');
    } else {
        phValueElement.css('color', '#126cc4');
    }
};

self.onResize = function() {
    self.updatePhValue();
};
```

C.4 HTML för Vattentanken

```
    <div class="tank-widget">
  <div class="tank-title">Liquid level</div>

  <div class="tank-area">

    <div class="scale-wrapper">
      <div class="scale-line"></div>

      <div class="scale-label scale-top">0 cm</div>
      <div class="scale-label scale-bottom">29 cm</div>

      <div class="limit-marker max-marker">
        <span class="marker-line"></span>
        <span class="marker-text">MAX 3.5 cm</span>
      </div>

      <div class="limit-marker min-marker">
        <span class="marker-line"></span>
        <span class="marker-text">MIN 17 cm</span>
      </div>
    </div>

    <div class="tank-container">
      <svg class="tank-svg" viewBox="0 0 260 260"
         preserveAspectRatio="xMidYMid meet">

        <!-- Tank outline -->
        <ellipse cx="130" cy="70" rx="95" ry="28" class="tank
           -top-outline"/>
```

```
        <path d="M35 70 L35 190 C35 210 225 210 225 190 L225
           70" class="tank-side-outline"/>
        <ellipse cx="130" cy="190" rx="95" ry="28" class="
           tank-bottom-outline"/>

        <!-- Neck -->
        <ellipse cx="130" cy="44" rx="28" ry="7" class="tank-
           neck"/>
        <path d="M102 44 L102 51 C102 58 158 58 158 51 L158
           44" class="tank-neck-side"/>
        <ellipse cx="130" cy="51" rx="28" ry="7" class="tank-
           neck-bottom"/>

        <!-- Water clip area -->
        <clipPath id="tankClip">
          <path d="M35 70 L35 190 C35 210 225 210 225 190
             L225 70 C225 92 35 92 35 70 Z"/>
        </clipPath>

        <!-- Water group -->
        <g clip-path="url(#tankClip)">
          <rect id="waterFill" x="35" y="190" width="190"
             height="0" class="water-fill"></rect>
          <ellipse id="waterSurface" cx="130" cy="190" rx="95
             " ry="23" class="water-surface"></ellipse>
        </g>

        <!-- Front outline on top -->
        <path d="M35 70 L35 190 C35 210 225 210 225 190 L225
           70" class="tank-front-outline"/>
        <ellipse cx="130" cy="70" rx="95" ry="28" class="tank
           -top-front"/>

      </svg>

      <div id="levelText" class="level-text">-- cm</div>
    </div>

  </div>
</div>
```

C.5 CSS för Vattentanken

```
    .tank-widget {
  width: 100%;
  height: 100%;
  box-sizing: border-box;
  font-family: Roboto, Arial, sans-serif;
  padding: 18px 22px;
```

```
  background: #ffffff;
  color: #222;
}

.tank-title {
  font-size: 38px;
  font-weight: 700;
  margin-bottom: 10px;
}

.tank-area {
  position: relative;
  width: 100%;
  height: calc(100% - 60px);
  display: flex;
  align-items: center;
  justify-content: center;
}

/* Tanken ligger kvar i mitten */
.tank-container {
  position: relative;
  width: 52%;
  max-width: 560px;
  min-width: 320px;
  margin: 0 auto;
}

.tank-svg {
  width: 100%;
  height: auto;
  display: block;
}

.tank-top-outline,
.tank-side-outline,
.tank-bottom-outline,
.tank-front-outline,
.tank-top-front,
.tank-neck,
.tank-neck-side,
.tank-neck-bottom {
  fill: none;
  stroke: #20206f;
  stroke-width: 3;
}

.tank-top-outline {
  fill: #f7f8ff;
```

```
}

.tank-neck {
  fill: #ffffff;
}

.tank-neck-side {
  fill: #ffffff;
}

.tank-neck-bottom {
  fill: #f7f8ff;
}

.water-fill {
  fill: rgba(95, 115, 255, 0.78);
  transition: y 0.4s ease, height 0.4s ease;
}

.water-surface {
  fill: rgba(130, 150, 255, 0.85);
  transition: cy 0.4s ease;
}

.level-text {
  position: absolute;
  left: 50%;
  top: 58%;
  transform: translate(-50%, -50%);
  padding: 8px 18px;
  border-radius: 6px;
  background: rgba(245, 245, 255, 0.9);
  color: #d50000;
  font-size: 32px;
  font-weight: 700;
  box-shadow: 0 1px 4px rgba(0,0,0,0.18);
}

/* Scale moved further to the right without changing the idea
    */
.scale-wrapper {
  position: absolute;
  right: -10%;
  top: 23%;
  height: 55%;
  width: 185px;
}

.scale-line {
```

```
  position: absolute;
  left: 22px;
  top: 0;
  height: 100%;
  width: 5px;
  background: #222;
}

.scale-line::before,
.scale-line::after {
  content: "";
  position: absolute;
  left: -12px;
  width: 29px;
  height: 5px;
  background: #222;
}

.scale-line::before {
  top: 0;
}

.scale-line::after {
  bottom: 0;
}

.scale-label {
  position: absolute;
  left: 52px;
  font-size: 20px;
  font-weight: 700;
  color: #222;
  white-space: nowrap;
}

.scale-top {
  top: -12px;
}

.scale-bottom {
  bottom: -12px;
}

.limit-marker {
  position: absolute;
  left: 0;
  display: flex;
  align-items: center;
  transform: translateY(-50%);
```

```
}

.marker-line {
  width: 45px;
  height: 5px;
  display: inline-block;
  margin-right: 10px;
}

.marker-text {
  font-size: 20px;
  font-weight: 700;
  white-space: nowrap;
}

.max-marker {
  top: 12.07%;
}

.max-marker .marker-line {
  background: #d50000;
}

.max-marker .marker-text {
  color: #d50000;
}

.min-marker {
  top: 58.62%;
}

.min-marker .marker-line {
  background: #ff9800;
}

.min-marker .marker-text {
  color: #ff9800;
}
```

C.6 Javascript för Vattentanken

```
    self.onInit = function() {
    self.updateTankLevel();
};

self.onDataUpdated = function() {
    self.updateTankLevel();
};
```

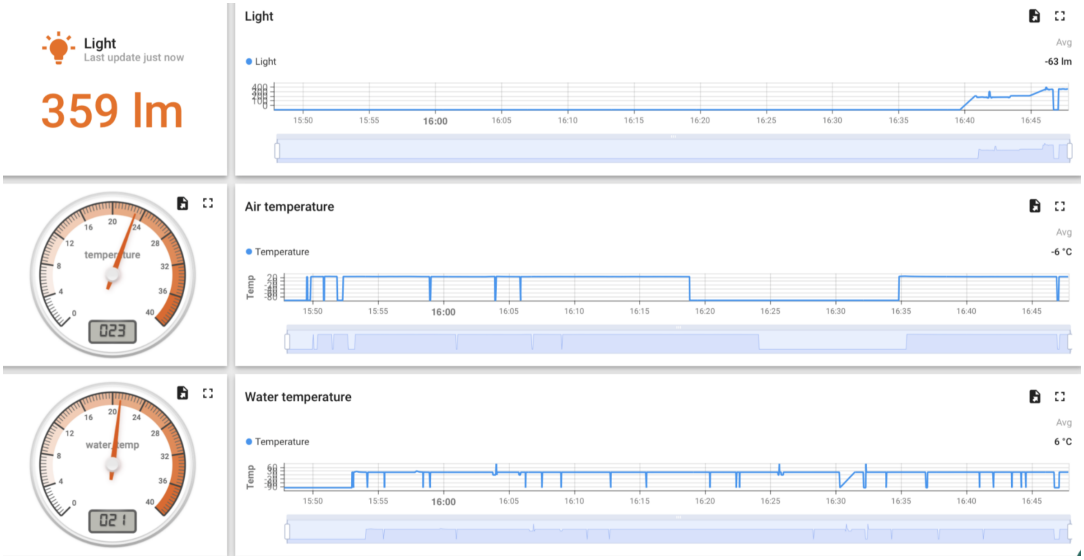
```
self.onResize = function() {
    self.updateTankLevel();
};

self.updateTankLevel = function() {
    var maxDistanceCm = 29;
    var topCm = 0;
    var bottomCm = 29;

    var levelCm = null;

    if (self.ctx.data && self.ctx.data.length > 0) {
        var dataKey = self.ctx.data[0];

        if (dataKey.data && dataKey.data.length > 0) {
            var latestData = dataKey.data[dataKey.data.length
                - 1];
            levelCm = parseFloat(latestData[1]);
        }
    }

    if (levelCm === null || isNaN(levelCm)) {
        levelCm = bottomCm;
    }

    if (levelCm < topCm) {
        levelCm = topCm;
    }

    if (levelCm > bottomCm) {
        levelCm = bottomCm;
    }

    /*
      The ThingsBoard value is assumed to be the distance
         from the sensor to the water surface.
      0 cm betyder full tank.
      29 cm betyder tom tank.
    */
    var fillRatio = (bottomCm - levelCm) / maxDistanceCm;

    if (fillRatio < 0) {
        fillRatio = 0;
    }

    if (fillRatio > 1) {
        fillRatio = 1;
    }
```

```
    /*
      SVG-koordinater:
      The tanks top roughly at y = 70
      The tanks bottom roughly at  y = 190
    */
    var tankTopY = 70;
    var tankBottomY = 190;
    var tankHeight = tankBottomY - tankTopY;

    var waterHeight = tankHeight * fillRatio;
    var waterY = tankBottomY - waterHeight;

    var waterFill = self.ctx.$container.find('#waterFill');
    var waterSurface = self.ctx.$container.find('#
       waterSurface');
    var levelText = self.ctx.$container.find('#levelText');

    waterFill.attr('y', waterY);
    waterFill.attr('height', waterHeight);
    waterSurface.attr('cy', waterY);

    levelText.text(levelCm.toFixed(1) + ' cm');

    /*
      coulor logic:
      Under 3.5 cm = close to full / over max
      betwene 3.5 and 17 cm = normal level
      over 17 cm = low water level
    */
    if (levelCm <= 3.5) {
        levelText.css('color', '#d50000');
    } else if (levelCm >= 17) {
        levelText.css('color', '#ff9800');
    } else {
        levelText.css('color', '#333');
    }
};
```

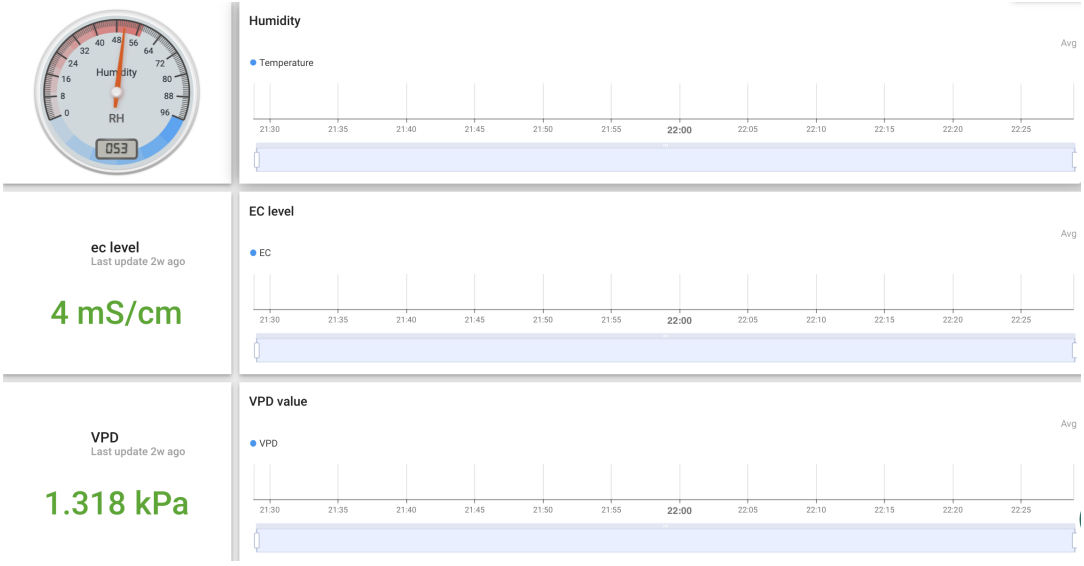
D

Dashboard i ThingsBoard

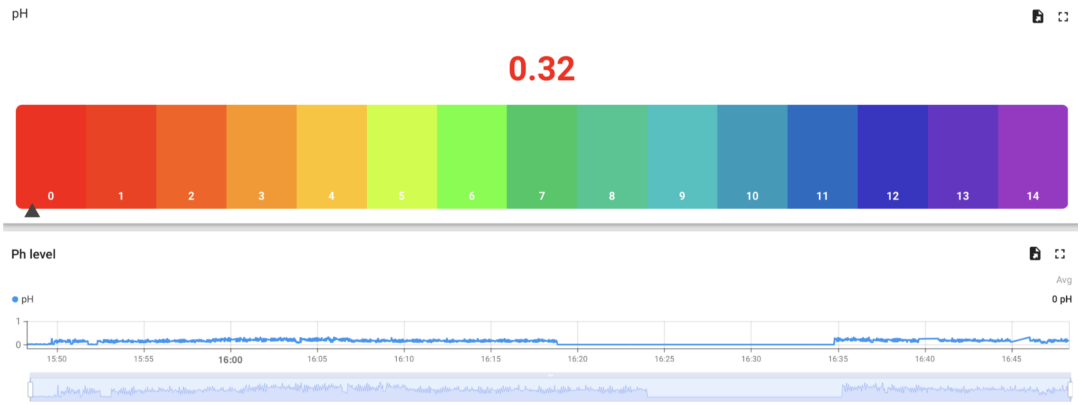
Denna bilaga visar dashboarden i ThingsBoard. Eftersom dashboarden inte ryms i en och samma skärmbild redovisas den i flera delar.



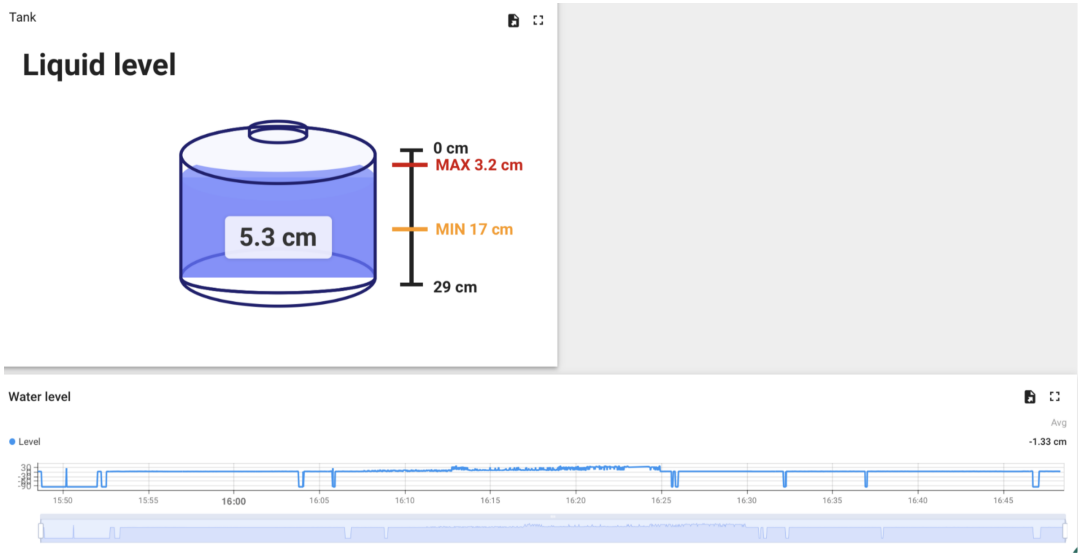
Figur 18: Första delen av dashboarden i ThingsBoard.



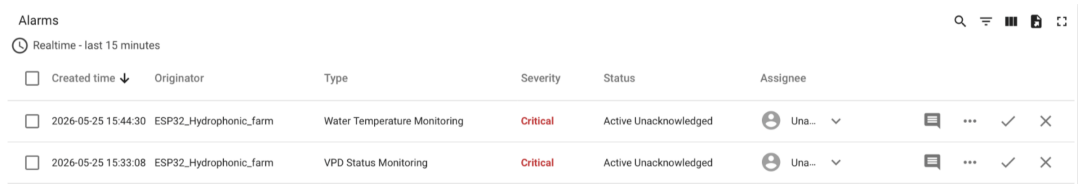
Figur 19: Andra delen av dashboarden i ThingsBoard.



Figur 20: Tredje delen av dashboarden i ThingsBoard.



Figur 21: Fjärde delen av dashboarden i ThingsBoard.



Figur 22: Femte delen av dashboarden i ThingsBoard.

E

Bilaga

Loggutdrag av Zephyr koden.

E.1 Test av Temperatur och Luftfuktighet

Test att lägga ett finger på sensorn.

```
Temperature: 22.38  C   |  Humidity: 46.47 %
Temperature: 22.41  C   |  Humidity: 46.50 %
Temperature: 22.40  C   |  Humidity: 46.65 %
Temperature: 22.40  C   |  Humidity: 46.58 %
Temperature: 22.38  C   |  Humidity: 46.54 %
Temperature: 22.43  C   |  Humidity: 46.44 %
Temperature: 22.40  C   |  Humidity: 46.46 %
Temperature: 22.40  C   |  Humidity: 46.19 %
Temperature: 22.38  C   |  Humidity: 46.07 %
Temperature: 25.05  C   |  Humidity: 46.54 %
Temperature: 27.21  C   |  Humidity: 61.32 %
Temperature: 28.12  C   |  Humidity: 69.40 %
Temperature: 28.81  C   |  Humidity: 73.39 %
Temperature: 29.31  C   |  Humidity: 75.43 %
Temperature: 29.73  C   |  Humidity: 76.50 %
Temperature: 30.01  C   |  Humidity: 77.02 %
Temperature: 30.25  C   |  Humidity: 77.28 %
Temperature: 30.44  C   |  Humidity: 77.42 %
Temperature: 29.67  C   |  Humidity: 69.94 %
Temperature: 29.63  C   |  Humidity: 77.43 %
Temperature: 30.49  C   |  Humidity: 81.04 %
Temperature: 30.82  C   |  Humidity: 87.03 %
Temperature: 30.35  C   |  Humidity: 90.24 %
Temperature: 29.66  C   |  Humidity: 90.08 %
Temperature: 29.17  C   |  Humidity: 82.42 %
Temperature: 28.77  C   |  Humidity: 73.96 %
Temperature: 28.42  C   |  Humidity: 66.08 %
Temperature: 28.05  C   |  Humidity: 59.07 %
Temperature: 27.50  C   |  Humidity: 53.70 %
Temperature: 27.24  C   |  Humidity: 52.05 %
Temperature: 26.81  C   |  Humidity: 51.22 %
Temperature: 26.47  C   |  Humidity: 50.07 %
Temperature: 26.32  C   |  Humidity: 50.43 %
Temperature: 26.14  C   |  Humidity: 49.85 %
Temperature: 25.98  C   |  Humidity: 48.15 %
Temperature: 25.84  C   |  Humidity: 46.70 %
Temperature: 25.73  C   |  Humidity: 46.05 %
Temperature: 25.60  C   |  Humidity: 45.02 %
```

```
Temperature: 25.49  C   |  Humidity: 44.31 %
Temperature: 25.39  C   |  Humidity: 44.29 %
Temperature: 25.27  C   |  Humidity: 44.00 %
Temperature: 25.18  C   |  Humidity: 43.93 %
Temperature: 25.08  C   |  Humidity: 44.41 %
Temperature: 24.98  C   |  Humidity: 44.21 %
Temperature: 24.89  C   |  Humidity: 43.72 %
Temperature: 24.84  C   |  Humidity: 43.40 %
Temperature: 24.79  C   |  Humidity: 43.55 %
Temperature: 24.74  C   |  Humidity: 43.61 %
Temperature: 24.68  C   |  Humidity: 43.55 %
Temperature: 24.63  C   |  Humidity: 43.70 %
Temperature: 24.57  C   |  Humidity: 43.87 %
Temperature: 24.54  C   |  Humidity: 44.46 %
Temperature: 24.47  C   |  Humidity: 44.59 %
Temperature: 24.43  C   |  Humidity: 44.57 %
Temperature: 24.10  C   |  Humidity: 43.98 %
Temperature: 24.05  C   |  Humidity: 43.96 %
Temperature: 24.03  C   |  Humidity: 43.76 %
Temperature: 23.99  C   |  Humidity: 43.80 %
Temperature: 23.98  C   |  Humidity: 43.80 %
Temperature: 23.98  C   |  Humidity: 43.83 %
Temperature: 23.96  C   |  Humidity: 44.07 %
Temperature: 23.94  C   |  Humidity: 44.31 %
Temperature: 23.92  C   |  Humidity: 44.73 %
Temperature: 23.92  C   |  Humidity: 45.08 %
Temperature: 23.88  C   |  Humidity: 45.19 %
Temperature: 23.86  C   |  Humidity: 45.57 %
Temperature: 23.81  C   |  Humidity: 45.32 %
Temperature: 23.79  C   |  Humidity: 45.20 %
Temperature: 23.81  C   |  Humidity: 46.57 %
Temperature: 23.79  C   |  Humidity: 47.48 %
Temperature: 23.75  C   |  Humidity: 47.16 %
Temperature: 23.78  C   |  Humidity: 46.58 %
Temperature: 23.72  C   |  Humidity: 46.14 %
Temperature: 23.51  C   |  Humidity: 44.34 %
Temperature: 23.48  C   |  Humidity: 44.33 %
Temperature: 23.48  C   |  Humidity: 44.34 %
Temperature: 23.46  C   |  Humidity: 44.42 %
Temperature: 23.44  C   |  Humidity: 44.52 %
Temperature: 23.43  C   |  Humidity: 44.52 %
Temperature: 23.43  C   |  Humidity: 44.54 %
Temperature: 23.40  C   |  Humidity: 44.48 %
Temperature: 23.40  C   |  Humidity: 44.44 %
Temperature: 23.40  C   |  Humidity: 44.47 %
Temperature: 23.39  C   |  Humidity: 44.64 %
Temperature: 23.34  C   |  Humidity: 44.64 %
Temperature: 23.31  C   |  Humidity: 44.67 %
Temperature: 23.34  C   |  Humidity: 44.64 %
```

```
Temperature: 23.33  C   |  Humidity: 44.70 %
Temperature: 23.31  C   |  Humidity: 44.73 %
Temperature: 23.31  C   |  Humidity: 44.85 %
Temperature: 23.28  C   |  Humidity: 44.86 %
Temperature: 23.26  C   |  Humidity: 45.02 %
Temperature: 23.21  C   |  Humidity: 45.04 %
Temperature: 23.20  C   |  Humidity: 45.07 %
Temperature: 23.21  C   |  Humidity: 45.22 %
Temperature: 23.20  C   |  Humidity: 45.32 %
Temperature: 23.21  C   |  Humidity: 45.38 %
Temperature: 23.20  C   |  Humidity: 45.39 %
Temperature: 23.20  C   |  Humidity: 45.37 %
Temperature: 23.19  C   |  Humidity: 45.39 %
Temperature: 23.21  C   |  Humidity: 45.33 %
Temperature: 23.17  C   |  Humidity: 45.14 %
Temperature: 23.20  C   |  Humidity: 45.10 %
Temperature: 23.17  C   |  Humidity: 45.06 %
Temperature: 23.17  C   |  Humidity: 45.11 %
Temperature: 23.13  C   |  Humidity: 45.09 %
Temperature: 23.13  C   |  Humidity: 45.10 %
Temperature: 23.14  C   |  Humidity: 45.03 %
Temperature: 23.13  C   |  Humidity: 44.97 %
Temperature: 23.13  C   |  Humidity: 45.06 %
Temperature: 23.12  C   |  Humidity: 45.05 %
Temperature: 22.99  C   |  Humidity: 45.10 %
Temperature: 23.00  C   |  Humidity: 45.16 %
Temperature: 22.99  C   |  Humidity: 45.12 %
Temperature: 23.00  C   |  Humidity: 45.11 %
Temperature: 22.96  C   |  Humidity: 45.06 %
Temperature: 22.96  C   |  Humidity: 45.11 %
Temperature: 22.95  C   |  Humidity: 45.09 %
Temperature: 22.99  C   |  Humidity: 45.09 %
Temperature: 22.98  C   |  Humidity: 45.08 %
Temperature: 22.99  C   |  Humidity: 45.07 %
Temperature: 22.96  C   |  Humidity: 45.12 %
Temperature: 22.95  C   |  Humidity: 45.08 %
Temperature: 22.93  C   |  Humidity: 45.07 %
Temperature: 22.92  C   |  Humidity: 45.01 %
Temperature: 22.92  C   |  Humidity: 45.12 %
Temperature: 22.89  C   |  Humidity: 45.23 %
Temperature: 22.93  C   |  Humidity: 45.35 %
Temperature: 22.96  C   |  Humidity: 46.06 %
Temperature: 22.98  C   |  Humidity: 46.11 %
Temperature: 22.95  C   |  Humidity: 46.18 %
Temperature: 22.95  C   |  Humidity: 46.23 %
Temperature: 22.96  C   |  Humidity: 46.33 %
Temperature: 22.96  C   |  Humidity: 46.32 %
Temperature: 22.93  C   |  Humidity: 46.32 %
Temperature: 22.96  C   |  Humidity: 46.33 %
```

```
Temperature: 22.93  C   |  Humidity: 46.33 %
Temperature: 22.96  C   |  Humidity: 46.39 %
Temperature: 22.93  C   |  Humidity: 46.53 %
Temperature: 22.96  C   |  Humidity: 46.58 %
Temperature: 22.95  C   |  Humidity: 46.56 %
Temperature: 22.96  C   |  Humidity: 46.51 %
Temperature: 22.95  C   |  Humidity: 46.52 %
Temperature: 22.95  C   |  Humidity: 46.40 %
Temperature: 22.93  C   |  Humidity: 46.39 %
Temperature: 22.95  C   |  Humidity: 46.41 %
Temperature: 22.95  C   |  Humidity: 46.40 %
Temperature: 22.91  C   |  Humidity: 46.11 %
Temperature: 22.89  C   |  Humidity: 46.04 %
Temperature: 22.89  C   |  Humidity: 46.17 %
Temperature: 22.91  C   |  Humidity: 46.38 %
Temperature: 22.88  C   |  Humidity: 46.47 %
Temperature: 22.91  C   |  Humidity: 46.56 %
Temperature: 22.89  C   |  Humidity: 46.63 %
Temperature: 22.89  C   |  Humidity: 46.72 %
Temperature: 22.91  C   |  Humidity: 46.80 %
Temperature: 22.92  C   |  Humidity: 46.81 %
Temperature: 22.89  C   |  Humidity: 46.64 %
Temperature: 22.89  C   |  Humidity: 46.53 %
Temperature: 22.89  C   |  Humidity: 46.47 %
Temperature: 22.91  C   |  Humidity: 46.29 %
Temperature: 22.88  C   |  Humidity: 46.18 %
Temperature: 22.89  C   |  Humidity: 46.05 %
Temperature: 22.88  C   |  Humidity: 46.02 %
Temperature: 22.88  C   |  Humidity: 46.01 %
Temperature: 22.81  C   |  Humidity: 46.18 %
Temperature: 22.81  C   |  Humidity: 46.15 %
Temperature: 22.81  C   |  Humidity: 46.15 %
Temperature: 22.82  C   |  Humidity: 46.10 %
Temperature: 22.84  C   |  Humidity: 46.04 %
Temperature: 22.82  C   |  Humidity: 46.08 %
Temperature: 22.81  C   |  Humidity: 46.06 %
Temperature: 22.84  C   |  Humidity: 46.08 %
Temperature: 22.85  C   |  Humidity: 46.81 %
Temperature: 22.84  C   |  Humidity: 47.46 %
Temperature: 22.85  C   |  Humidity: 47.67 %
Temperature: 22.85  C   |  Humidity: 47.24 %
Temperature: 22.86  C   |  Humidity: 46.75 %
Temperature: 22.84  C   |  Humidity: 46.41 %
Temperature: 22.84  C   |  Humidity: 46.26 %
Temperature: 22.84  C   |  Humidity: 46.40 %
Temperature: 22.85  C   |  Humidity: 46.64 %
Temperature: 22.81  C   |  Humidity: 46.16 %
Temperature: 22.81  C   |  Humidity: 46.10 %
Temperature: 22.82  C   |  Humidity: 46.16 %
```

```
Temperature: 22.82  C   |  Humidity: 46.16 %
Temperature: 22.82  C   |  Humidity: 46.27 %
Temperature: 22.82  C   |  Humidity: 46.29 %
Temperature: 22.82  C   |  Humidity: 46.15 %
Temperature: 22.81  C   |  Humidity: 46.12 %
Temperature: 22.81  C   |  Humidity: 46.17 %
Temperature: 22.79  C   |  Humidity: 46.27 %
Temperature: 22.81  C   |  Humidity: 46.30 %
Temperature: 22.82  C   |  Humidity: 46.29 %
Temperature: 22.79  C   |  Humidity: 46.21 %
Temperature: 22.79  C   |  Humidity: 46.15 %
Temperature: 22.81  C   |  Humidity: 46.04 %
Temperature: 22.81  C   |  Humidity: 46.12 %
Temperature: 22.77  C   |  Humidity: 46.76 %
```

E.2 Test av Näringssensor

Test att byta från vanligt vatten till vatten med salt löst i sig.

```
EC: 0.119 mS/cm
EC: 0.118 mS/cm
EC: 0.118 mS/cm
EC: 0.116 mS/cm
EC: 0.118 mS/cm
EC: 0.117 mS/cm
EC: 2.176 mS/cm
EC: 2.176 mS/cm
EC: 2.175 mS/cm
EC: 2.176 mS/cm
EC: 2.175 mS/cm
EC: 2.179 mS/cm
EC: 0.169 mS/cm
EC: 0.144 mS/cm
EC: 0.144 mS/cm
EC: 0.145 mS/cm
EC: 0.147 mS/cm
```

E.3 Test av Näringssensor

Test efter kalibrering med 1,7mS/cm, 3.4mS/cm och 5mS/cm

```
ec 0.15
ec 0.15
ec 0.00
ec 0.00
ec 0.00
ec 0.00
ec 0.00
ec 0.00
```

```
ec 1.44
ec 1.44
ec 0.19
ec 0.19
ec 0.22
ec 0.35
ec 0.35
ec 0.30
ec 0.30
ec 3.66
ec 3.66
ec 4.53
ec 4.53
ec 4.20
ec 4.20
ec 4.15
ec 4.15
ec 4.60
ec 4.95
ec 4.95
ec 4.35
ec 4.35
ec 0.00
ec 0.00
ec 0.00
ec 0.00
ec 0.08
ec 0.08
ec 0.00
ec 0.00
ec 0.00
ec 0.00
ec 2.65
ec 2.65
ec 3.10
ec 3.10
ec 2.81
ec 2.81
ec 3.01
ec 3.01
ec 3.24
ec 3.15
ec 3.15
ec 3.11
ec 3.11
ec 3.15
ec 3.15
ec 3.08
ec 3.08
```

```
ec 3.28
ec 3.28
ec 3.30
ec 3.30
ec 3.20
ec 3.30
ec 3.30
ec 0.00
ec 0.00
ec 0.00
ec 0.00
ec 0.00
ec 0.00
ec 0.00
ec 0.00
ec 0.00
ec 0.00
ec 0.00
ec 0.00
ec 0.00
ec 0.00
ec 1.84
ec 1.84
ec 1.84
ec 1.84
ec 1.88
ec 1.88
ec 1.95
ec 1.95
ec 1.97
```

E.4 Test av Vattentemperaturen

Test genom att flytta sensorn från klat till varmt vatten och tillbaka

```
Water temperature: 19.00  C
Water temperature: 19.00  C
Water temperature: 19.00  C
Water temperature: 19.12  C
Water temperature: 24.44  C
Water temperature: 28.37  C
Water temperature: 31.06  C
Water temperature: 33.06  C
Water temperature: 34.63  C
Water temperature: 35.88  C
Water temperature: 36.94  C
Water temperature: 37.81  C
Water temperature: 38.44  C
Water temperature: 38.94  C
```

```
Water temperature: 39.56  C
Water temperature: 37.50  C
Water temperature: 33.06  C
Water temperature: 30.12  C
Water temperature: 28.00  C
Water temperature: 26.37  C
Water temperature: 25.12  C
Water temperature: 24.12  C
Water temperature: 23.31  C
Water temperature: 22.62  C
Water temperature: 22.12  C
Water temperature: 21.62  C
Water temperature: 21.31  C
Water temperature: 21.00  C
Water temperature: 20.75  C
Water temperature: 20.56  C
Water temperature: 20.37  C
Water temperature: 20.19  C
Water temperature: 20.12  C
Water temperature: 20.00  C
Water temperature: 19.87  C
Water temperature: 19.81  C
Water temperature: 19.81  C
Water temperature: 19.75  C
Water temperature: 19.69  C
Water temperature: 19.62  C
Water temperature: 19.62  C
Water temperature: 19.56  C
Water temperature: 19.56  C
Water temperature: 19.56  C
Water temperature: 19.50  C
Water temperature: 19.50  C
Water temperature: 19.50  C
```

E.5 Test av Avståndssenorn

Test av avståndssensorn – fast avstånd

```
Distance: 74.20 cm
Distance: 74.20 cm
Distance: 74.20 cm
Distance: 74.20 cm
Distance: 74.20 cm
Distance: 74.20 cm
Distance: 74.20 cm
Distance: 74.20 cm
Distance: 74.20 cm
Distance: 74.20 cm
Distance: 74.20 cm
```

```
Distance: 74.20 cm
Distance: 74.20 cm
Distance: 74.20 cm
Distance: 74.20 cm
Distance: 74.20 cm
Distance: 74.20 cm
Distance: 74.20 cm
Distance: 74.20 cm
Distance: 74.20 cm
Distance: 74.20 cm
Distance: 74.20 cm
Distance: 74.20 cm
Distance: 74.20 cm
Distance: 74.20 cm
Distance: 74.20 cm
Distance: 74.20 cm
Distance: 74.20 cm
```

E.6 Test av Avståndssenorn

Test av avståndssensorn – miniminivå

```
Distance: 23.90 cm
Distance: 24.00 cm
Distance: 24.30 cm
Distance: 20.00 cm
Distance: 20.40 cm
Distance: 19.50 cm
Distance: 23.40 cm
Distance: 20.80 cm
Distance: 25.30 cm
Distance: 24.80 cm
Distance: 20.40 cm
Distance: 20.80 cm
Distance: 20.00 cm
Distance: 19.50 cm
Distance: 20.40 cm
Distance: 18.30 cm
Distance: 20.40 cm
Distance: 20.40 cm
Distance: 20.80 cm
Distance: 20.90 cm
Distance: 21.70 cm
Distance: 21.70 cm
Distance: 23.90 cm
Distance: 21.60 cm
Distance: 21.70 cm
Distance: 21.70 cm
Distance: 18.70 cm
```

```
Distance: 19.60 cm
Distance: 19.50 cm
Distance: 17.80 cm
Distance: 21.70 cm
Distance: 21.70 cm
Distance: 22.10 cm
Distance: 22.60 cm
Distance: 21.70 cm
Distance: 22.60 cm
Distance: 24.40 cm
Distance: 23.50 cm
Distance: 23.00 cm
Distance: 23.80 cm
Distance: 23.50 cm
Distance: 24.30 cm
```

E.7 Test av Avståndssenorn

Test av avståndssensorn – maxnivå

```
Distance: 3.70 cm
Distance: 3.70 cm
Distance: 3.70 cm
Distance: 3.70 cm
Distance: 3.70 cm
Distance: 3.70 cm
Distance: 3.70 cm
Distance: 3.70 cm
Distance: 3.70 cm
Distance: 3.70 cm
Distance: 3.70 cm
Distance: 3.70 cm
Distance: 3.70 cm
Distance: 3.30 cm
Distance: 3.70 cm
Distance: 3.70 cm
Distance: 3.70 cm
Distance: 3.70 cm
Distance: 3.70 cm
Distance: 3.30 cm
Distance: 3.30 cm
Distance: 3.30 cm
Distance: 3.30 cm
Distance: 3.30 cm
Distance: 3.30 cm
Distance: 3.30 cm
Distance: 3.30 cm
Distance: 3.70 cm
Distance: 3.30 cm
Distance: 3.80 cm
```

```
Distance: 3.30 cm
Distance: 3.30 cm
Distance: 3.70 cm
Distance: 3.30 cm
Distance: 3.30 cm
Distance: 3.30 cm
Distance: 3.30 cm
Distance: 3.30 cm
Distance: 3.30 cm
Distance: 4.20 cm
Distance: 3.30 cm
Distance: 3.30 cm
Distance: 3.30 cm
Distance: 3.80 cm
Distance: 3.30 cm
Distance: 3.70 cm
Distance: 3.30 cm
Distance: 3.40 cm
Distance: 3.70 cm
Distance: 3.80 cm
Distance: 3.30 cm
Distance: 3.80 cm
Distance: 3.30 cm
Distance: 3.30 cm
Distance: 3.30 cm
Distance: 4.10 cm
Distance: 3.30 cm
Distance: 3.30 cm
Distance: 4.20 cm
Distance: 3.30 cm
Distance: 3.30 cm
Distance: 3.30 cm
Distance: 3.30 cm
Distance: 3.30 cm
Distance: 3.30 cm
Distance: 3.70 cm
Distance: 3.70 cm
Distance: 3.30 cm
Distance: 4.10 cm
Distance: 4.10 cm
Distance: 3.70 cm
Distance: 3.70 cm
Distance: 3.70 cm
Distance: 3.70 cm
Distance: 3.30 cm
Distance: 3.70 cm
Distance: 3.30 cm
Distance: 3.70 cm
```

E.8 Test av Ljussensorn

Test att täcka för sensorn för att aproximera mörkt rum samt test i starkt solljus

```
94.41 Lux
92.12 Lux
92.12 Lux
88.60 Lux
79.32 Lux
87.46 Lux
85.08 Lux
17.85 Lux
7.14 Lux
7.14 Lux
7.14 Lux
7.14 Lux
7.14 Lux
0.00 Lux
0.00 Lux
0.00 Lux
0.00 Lux
0.00 Lux
85.08 Lux
85.08 Lux
85.08 Lux
85.08 Lux
85.08 Lux
79.32 Lux
87.46 Lux
87.46 Lux
87.46 Lux
87.46 Lux
81.58 Lux
87.46 Lux
87.46 Lux
84.00 Lux
68.77 Lux
96.84 Lux
81.58 Lux
79.32 Lux
169.03 Lux
317.22 Lux
960.75 Lux
1987.68 Lux
4608.46 Lux
45771.44 Lux
108014.63 Lux
106491.36 Lux
105219.92 Lux
9933.21 Lux
```

```
12029.69 Lux
105537.81 Lux
106152.46 Lux
105618.71 Lux
6814.20 Lux
3407.00 Lux
1955.81 Lux
422.06 Lux
187.75 Lux
78.10 Lux
68.77 Lux
```

E.9 Test av pH sensorn

Test efter kalibreringen av sensorn. Sensorn spolades med avjoniserat vatten mellean mätningarna därav lite konstiga värden mellan mätningarna.

```
pH 6.00
pH 5.99
pH 5.99
pH 5.97
pH 5.95
pH 5.94
pH 5.94
pH 5.95
pH 5.95
pH 5.95
pH 5.95
pH 5.95
pH 5.94
pH 5.89
pH 5.84
pH 5.66
pH 5.63
pH 5.55
pH 5.53
pH 6.21
pH 6.21
pH 6.15
pH 5.84
pH 5.64
pH 5.59
pH 5.46
pH 5.37
pH 5.42
pH 7.08
pH 7.06
pH 7.05
pH 7.05
```

pH 7.05
pH 7.06
pH 7.02
pH 7.03
pH 7.01
pH 7.03
pH 7.03
pH 7.01
pH 7.02
pH 7.01
pH 7.01
pH 7.03
pH 7.01
pH 7.01
pH 7.02
pH 7.01
pH 7.01
pH 7.01
pH 7.01
pH 7.00
pH 7.00
pH 7.00
pH 7.02
pH 7.03
pH 7.01
pH 7.01
pH 7.01
pH 7.16
pH 8.38
pH 8.66
pH 8.65
pH 8.62
pH 8.62
pH 8.40
pH 8.31
pH 8.22
pH 6.23
pH 6.26
pH 6.29
pH 6.32
pH 6.38
pH 6.41
pH 6.42
pH 6.42
pH 6.44
pH 6.46
pH 6.47
pH 6.47
pH 6.46

pH 6.51
pH 6.48
pH 6.48
pH 6.49
pH 6.52
pH 6.52
pH 6.38
pH 6.38
pH 6.39
pH 6.34
pH 6.26
pH 4.19
pH 4.15
pH 4.11
pH 4.11
pH 4.12
pH 4.12
pH 4.12
pH 4.09
pH 4.11
pH 4.09
pH 4.08
pH 4.09
pH 4.08
pH 4.07
pH 4.07
pH 4.07
pH 4.07
pH 4.10
pH 4.08
pH 4.09
pH 4.06
pH 4.09
pH 4.09
pH 4.18
pH 4.48
pH 5.50
pH 5.66
pH 5.66
pH 5.68
pH 5.74
pH 5.67
pH 5.53
pH 5.39
pH 4.54
pH 4.66
pH 4.67
pH 4.67
pH 4.70

```
pH 4.70
pH 4.69
pH 4.67
pH 4.67
pH 4.66
pH 4.64
pH 4.64
pH 4.65
pH 4.63
pH 4.63
pH 4.61
pH 4.61
pH 4.61
pH 4.51
pH 8.97
pH 9.00
pH 9.00
pH 9.00
pH 9.00
pH 9.00
pH 9.00
pH 9.00
pH 9.00
pH 9.00
pH 9.00
pH 9.00
pH 9.00
pH 9.00
pH 9.00
pH 9.00
pH 9.00
pH 9.00
pH 9.00
pH 9.00
pH 9.00
pH 9.00
pH 9.00
pH 9.00
pH 9.00
pH 9.00
pH 9.00
pH 9.00
pH 9.00
pH 9.00
pH 9.00
pH 9.00
pH 9.00
pH 9.00
pH 9.00
```

```
pH 9.00
pH 9.00
pH 9.00
pH 9.00
pH 9.00
pH 9.00
pH 9.00
pH 9.00
pH 9.00
pH 9.00
pH 9.00
pH 9.00
```

E.10 Stänga av och på sensorer under drift

Test genom att stänga av samtliga sensorer och sätta på dem igen för att se om Fallbackvärderna aktiveras korrekt.

```
=========ThingsBoard MQTT Connection & Device Activation
   =============
Closing previous MQTT connection
Network fully ready for DNS resolution
DNS resolution succeeded on attempt 1
MQTT connected successfully!
Connected to ThingsBoard MQTT broker
Device activated in ThingsBoard

=======Check Sensor connection========
Temperature: connected
Humidity: connected
Water temp: connected
EC: connected
Light: connected
Distance: connected
pH: connected

=======Sensor status========
Temperature active
Humidity active
Water temp active
EC active
Light active
Distance active
pH active

=======Publish Sensordata=======
Published sensor data:
{"temperature": 23.1, "humidity": 53.0, "water_temp": 21.3, "
   ec": 4.40, "pH": 0.39, "light": 354.90, "distance": 4.4}
```

```
Published sensor data:
{"temperature": 23.1, "humidity": 53.0, "water_temp": 21.3, "
   ec": 4.40, "pH": 0.28, "light": 349.86, "distance": 4.8}

Published sensor data:
{"temperature": 23.1, "humidity": 53.0, "water_temp": 21.3, "
   ec": 4.35, "pH": 0.28, "light": 337.13, "distance": 4.0}

Published sensor data:
{"temperature": 23.1, "humidity": 53.0, "water_temp": 21.3, "
   ec": 4.35, "pH": 0.29, "light": 337.13, "distance": 4.4}

Published sensor data:
{"temperature": 23.1, "humidity": 52.9, "water_temp": 21.3, "
   ec": 4.35, "pH": 0.29, "light": 340.69, "distance": 4.8}

Published sensor data:
{"temperature": 23.1, "humidity": 52.9, "water_temp": 21.3, "
   ec": 4.35, "pH": 0.26, "light": 342.75, "distance": 4.5}

Published sensor data:
{"temperature": 23.1, "humidity": 52.9, "water_temp": 21.3, "
   ec": 4.40, "pH": 0.32, "light": 339.20, "distance": 4.0}

Published sensor data:
{"temperature": 23.1, "humidity": 52.8, "water_temp": 21.3, "
   ec": 4.40, "pH": 0.33, "light": 330.03, "distance": 4.4}

Published sensor data:
{"temperature": 23.1, "humidity": 52.8, "water_temp": 21.3, "
   ec": 4.35, "pH": 0.28, "light": 322.94, "distance": 4.9}

Published sensor data:
{"temperature": 23.1, "humidity": 52.8, "water_temp": -99, "
   ec": 4.35, "pH": 0.25, "light": 348.39, "distance": 4.4}

Light: TSL2591 read failed
Published sensor data:
{"temperature": 23.1, "humidity": 52.8, "water_temp": -99, "
   ec": 0.42, "pH": 0.25, "light": -99, "distance": 4.5}

Published sensor data:
{"temperature": 23.1, "humidity": 52.8, "water_temp": -99, "
   ec": -99, "pH": 0.24, "light": -99, "distance": -99}

Published sensor data:
{"temperature": 23.1, "humidity": 52.8, "water_temp": -99, "
   ec": -99, "pH": 0.21, "light": -99, "distance": -99}
```

```
Published sensor data:
{"temperature": 23.1, "humidity": 52.8, "water_temp": -99, "
   ec": -99, "pH": 0.23, "light": -99, "distance": -99}

Published sensor data:
{"temperature": 23.1, "humidity": 52.8, "water_temp": -99, "
   ec": -99, "pH": 0.23, "light": -99, "distance": -99}

Published sensor data:
{"temperature": 23.1, "humidity": 52.8, "water_temp": -99, "
   ec": -99, "pH": 0.23, "light": -99, "distance": -99}

Published sensor data:
{"temperature": 23.1, "humidity": 52.8, "water_temp": -99, "
   ec": -99, "pH": 0.23, "light": -99, "distance": -99}

Published sensor data:
{"temperature": 23.1, "humidity": 52.8, "water_temp": -99, "
   ec": -99, "pH": 0.22, "light": -99, "distance": -99}

Published sensor data:
{"temperature": 23.1, "humidity": 52.8, "water_temp": -99, "
   ec": -99, "pH": 0.21, "light": -99, "distance": -99}

Published sensor data:
{"temperature": 23.1, "humidity": 52.8, "water_temp": -99, "
   ec": -99, "pH": 0.24, "light": -99, "distance": -99}

Published sensor data:
{"temperature": 23.1, "humidity": 52.8, "water_temp": -99, "
   ec": -99, "pH": 0.28, "light": -99, "distance": -99}

Published sensor data:
{"temperature": 23.1, "humidity": 52.8, "water_temp": -99, "
   ec": -99, "pH": 0.25, "light": -99, "distance": -99}

Published sensor data:
{"temperature": 23.1, "humidity": 52.8, "water_temp": -99, "
   ec": -99, "pH": 0.25, "light": -99, "distance": -99}

Published sensor data:
{"temperature": 23.1, "humidity": 52.8, "water_temp": -99, "
   ec": -99, "pH": 0.22, "light": -99, "distance": -99}

Published sensor data:
{"temperature": 23.1, "humidity": 52.8, "water_temp": -99, "
   ec": -99, "pH": 0.22, "light": -99, "distance": -99}
```

```
Published sensor data:
{"temperature": -99, "humidity": -99, "water_temp": -99, "ec"
   : -99, "pH": 0.26, "light": -99, "distance": -99}

Published sensor data:
{"temperature": -99, "humidity": -99, "water_temp": -99, "ec"
   : -99, "pH": 0.24, "light": -99, "distance": -99}

Published sensor data:
{"temperature": -99, "humidity": -99, "water_temp": -99, "ec"
   : -99, "pH": 0.24, "light": -99, "distance": -99}

Published sensor data:
{"temperature": -99, "humidity": -99, "water_temp": -99, "ec"
   : -99, "pH": 0.24, "light": -99, "distance": -99}

Published sensor data:
{"temperature": -99, "humidity": -99, "water_temp": -99, "ec"
   : -99, "pH": 0.20, "light": -99, "distance": -99}

Published sensor data:
{"temperature": 23.1, "humidity": 53.0, "water_temp": -99, "
   ec": -99, "pH": 0.23, "light": -99, "distance": -99}

Published sensor data:
{"temperature": 23.1, "humidity": 53.0, "water_temp": -99, "
   ec": -99, "pH": 0.22, "light": -99, "distance": -99}

Published sensor data:
{"temperature": 23.1, "humidity": 53.0, "water_temp": 85.0, "
   ec": -99, "pH": 0.28, "light": -99, "distance": -99}

Published sensor data:
{"temperature": 23.1, "humidity": 53.0, "water_temp": 85.0, "
   ec": -99, "pH": 0.30, "light": 355.47, "distance": -99}

Published sensor data:
{"temperature": 23.1, "humidity": 53.0, "water_temp": 21.3, "
   ec": 4.40, "pH": 0.30, "light": 351.94, "distance": 4.0}

Published sensor data:
{"temperature": 23.1, "humidity": 53.0, "water_temp": 21.3, "
   ec": 4.40, "pH": 0.23, "light": 353.39, "distance": 4.4}

Published sensor data:
{"temperature": 23.1, "humidity": 53.0, "water_temp": 21.3, "
   ec": 4.40, "pH": 0.31, "light": 359.03, "distance": 4.8}
```

E.11 Uppstart med sensorer av

Uppstart med sensorer frånkopplade

```
=== ESP32 WiFi Connect Test ===
Program started successfully
Network interface found
WiFi parameters configured for SSID: Johan
Connecting to WiFi SSID 'Johan'...
WiFi connect request sent, waiting for response...
WiFi state: 3
DHCPv4 start requested

=== WIFI CONNECTED ===
SSID: Johan
Waiting for IP address assignment...
IP assignment detected: 192.168.1.212

=========ThingsBoard MQTT Connection & Device Activation
   =============
Closing previous MQTT connection
Network fully ready for DNS resolution
DNS resolution succeeded on attempt 1
MQTT connected successfully!
Connected to ThingsBoard MQTT broker
Device activated in ThingsBoard

=======Check Sensor connection========
Temperature: not connected
Humidity: not connected
Water temp: not connected
EC: not connected
Light: not connected
Distance: not connected
pH: not connected

=======Sensor status========
Temperature fallback value active
Humidity fallback value active
Water temp fallback value active
EC forced fallback value active
Light fallback value active
Distance fallback value active
pH forced fallback value active

=======Publish Sensordata=======
Published sensor data:
{"temperature": -99, "humidity": -99, "water_temp": -99, "ec"
   : -99, "pH": -99, "light": -99, "distance": -99}
```

```
Published sensor data:
{"temperature": -99, "humidity": -99, "water_temp": -99, "ec"
   : -99, "pH": -99, "light": -99, "distance": -99}

Published sensor data:
{"temperature": -99, "humidity": -99, "water_temp": -99, "ec"
   : -99, "pH": -99, "light": -99, "distance": -99}
```

E.12 Tappad anslutning

Test genom att stänga av wifi och sedan sätta på det igen.

```
=== ESP32 WiFi Connect Test ===
Program started successfully
Network interface found
WiFi parameters configured for SSID: Johan
Connecting to WiFi SSID 'Johan'...
WiFi connect request sent, waiting for response...
WiFi state: 3
DHCPv4 start requested

=== WIFI CONNECTED ===
SSID: Johan
Waiting for IP address assignment...
IP assignment detected: 192.168.1.212

=========ThingsBoard MQTT Connection & Device Activation
   =============
Closing previous MQTT connection
Network fully ready for DNS resolution
DNS resolution succeeded on attempt 1
MQTT connected successfully!
Connected to ThingsBoard MQTT broker
Device activated in ThingsBoard

=======Check Sensor connection========
Temperature: not connected
Humidity: not connected
Water temp: not connected
EC: not connected
Light: not connected
Distance: not connected
pH: not connected

=======Sensor status========
Temperature fallback value active
Humidity fallback value active
Water temp fallback value active
EC forced fallback value active
```

```
Light fallback value active
Distance fallback value active
pH forced fallback value active

=======Publish Sensordata=======
Published sensor data:
{"temperature": -99, "humidity": -99, "water_temp": -99, "ec"
   : -99, "pH": -99, "light": -99, "distance": -99}

Published sensor data:
{"temperature": -99, "humidity": -99, "water_temp": -99, "ec"
   : -99, "pH": -99, "light": -99, "distance": -99}

Published sensor data:
{"temperature": -99, "humidity": -99, "water_temp": -99, "ec"
   : -99, "pH": -99, "light": -99, "distance": -99}

Published sensor data:
{"temperature": -99, "humidity": -99, "water_temp": -99, "ec"
   : -99, "pH": -99, "light": -99, "distance": -99}

Published sensor data:
{"temperature": -99, "humidity": -99, "water_temp": -99, "ec"
   : -99, "pH": -99, "light": -99, "distance": -99}

Published sensor data:
{"temperature": -99, "humidity": -99, "water_temp": -99, "ec"
   : -99, "pH": -99, "light": -99, "distance": -99}

MQTT disconnected
Failed to publish sensor data: -128
WiFi connection LOST - detected by active check
Will attempt WiFi reconnection in 15 seconds...
Attempting WiFi reconnection now...
Attempting WiFi reconnection...
WiFi state: 3
WiFi state: 0
WiFi state: 0
WiFi state: 0
WiFi state: 0
WiFi state: 0
WiFi reconnection timed out
Will attempt WiFi reconnection in 15 seconds...
Attempting WiFi reconnection now...
Attempting WiFi reconnection...
WiFi state: 3
WiFi state: 0
WiFi state: 0
WiFi state: 0
```

```
WiFi state: 0
WiFi state: 0
WiFi reconnection timed out
Will attempt WiFi reconnection in 15 seconds...
Attempting WiFi reconnection now...
Attempting WiFi reconnection...
WiFi state: 3
WiFi state: 0
WiFi state: 0
WiFi state: 0
WiFi state: 0
WiFi state: 0
WiFi reconnection timed out
Will attempt WiFi reconnection in 15 seconds...
Attempting WiFi reconnection now...
Attempting WiFi reconnection...
WiFi state: 3
WiFi state: 0
WiFi state: 0
WiFi state: 0
WiFi state: 0
WiFi state: 0
WiFi reconnection timed out
Will attempt WiFi reconnection in 15 seconds...
Attempting WiFi reconnection now...
Attempting WiFi reconnection...
WiFi state: 3
DHCPv4 start requested
WiFi reconnected successfully!
Waiting for IP address assignment after reconnect...
IP assignment detected after reconnect: 192.168.1.212
Closing previous MQTT connection
Network fully ready for DNS resolution
Hostname resolve retry 1 failed: -11 (EAGAIN - DNS not ready
   yet)
Hostname resolve retry 2 failed: -11 (EAGAIN - DNS not ready
   yet)
Hostname resolve retry 3 failed: -11 (EAGAIN - DNS not ready
   yet)
Hostname resolve retry 4 failed: -11 (EAGAIN - DNS not ready
   yet)
Hostname resolve retry 5 failed: -11 (EAGAIN - DNS not ready
   yet)
DNS still not ready, using cached broker IP
MQTT connected successfully!
Connected to ThingsBoard MQTT broker
Device activated in ThingsBoard
Published sensor data:
{"temperature": -99, "humidity": -99, "water_temp": -99, "ec"
```

```
   : -99, "pH": -99, "light": -99, "distance": -99}

Published sensor data:
{"temperature": -99, "humidity": -99, "water_temp": -99, "ec"
   : -99, "pH": -99, "light": -99, "distance": -99}

Published sensor data:
{"temperature": -99, "humidity": -99, "water_temp": -99, "ec"
   : -99, "pH": -99, "light": -99, "distance": -99}
```

INSTITUTIONEN FÖR ELEKTRTEKNIK
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige
www.chalmers.se

CHALMERS