Optimizing Water Usage in Melbourne's Residential Gardens Through IoT Soil Moisture Monitoring

COIT29226 – Assessment 3

Student Name: **Name Here**

Student ID: **ID Here**

Date: **06-June-2025**

Table of Contents

Introduction

Water conservation is a critical issue in Australia, one of the driest continents on earth. In urban areas like Melbourne, residential gardens can account for a large share of household water usage – typically around 40% or more. Traditional garden irrigation often relies on fixed timers or manual habits, which can lead to over-watering or watering at suboptimal times (e.g. in the heat of day, causing high evaporation losses).

There is a clear need for smarter irrigation methods that ensure plants get the water they need while minimizing waste. IoT-based solutions offer a promising approach by providing real-time data on soil and environmental conditions to inform watering decisions.

This report presents a case study on optimizing water use in Melbourne's residential gardens through IoT soil moisture monitoring. It builds upon earlier research and prototypes (Assignment 1 and 2) to design a complete system using **Central Queensland University (CQU)-provided sensors** – a soil moisture sensor and an environmental sensor – integrated via MQTT communication, Node-RED workflows, InfluxDB time-series storage, and Grafana visualization.

The goal is to demonstrate how data-driven irrigation can reduce water consumption and comply with local water-saving guidelines, all while maintaining healthy garden plants. For instance, Melbourne's permanent water-saving rules require that automated watering systems include rain or soil moisture sensors, underscoring the importance of this solution.

Case Study Analysis

Requirements and Background

The core requirement of this case is to **optimize water usage in a residential garden** without compromising plant health. Melbourne's climate is temperate but variable, with hot dry periods in summer and occasional droughts that have led to water restrictions in the past. Even when strict restrictions are not in place, the city encourages conservation through permanent water-saving rules and targets (e.g. "Target 150" urges individuals to use under 150 L/day).

Gardens are a major water consumer; studies show that outdoor use (mainly garden watering) can range from 25–60% of a household's water consumption. This represents both a significant cost to homeowners and an opportunity for conservation. Inefficient practices – like watering on a fixed schedule regardless of weather or soil condition – can lead to water wastage and even harm plants (through overwatering or fungal issues).

Problem Background

Traditionally, gardeners rely on intuition or visual cues (wilting plants, dry soil surface) to decide when to water. These methods are imprecise and often result in "just-in-case" watering that wastes water. There is often a knowledge gap in estimating how much water a garden actually needs.

Smart irrigation offers a solution by using **data** to drive decisions. Key factors influencing plant water needs include soil moisture levels, recent rainfall, temperature, humidity, sunlight, and wind. For example, high temperature and low humidity accelerate evapotranspiration (water loss from soil and plants), meaning the garden will dry out faster.

Likewise, if rain has occurred or is expected, watering can be skipped or reduced. An effective system must monitor these factors and either automatically adjust irrigation or provide timely recommendations to the user.

System Requirements

Based on the above, the system must continuously monitor soil moisture in the garden to know when it falls below a threshold indicating need for water. It should also monitor environmental conditions (temperature, humidity, etc.) to contextualize water needs (e.g., hot and windy conditions might justify preemptive watering, whereas cool humid conditions do not). The solution should notify the user or automatically trigger watering when needed, and conversely avoid irrigation when nature has already provided enough moisture (rain) or when it would be wasteful (hot midday periods).

Additionally, the system should be cost-effective and user-friendly for a typical homeowner. This implies using affordable sensors and open-source software, and providing information in an easy-to-understand format (such as a smartphone-accessible dashboard). Low power operation is also important since garden sensors may be battery-powered and placed outdoors; thus, using efficient wireless communication (MQTT) and duty cycling sensors to conserve energy are relevant considerations.

A further requirement is to ensure compliance with local guidelines. In Melbourne, automated irrigation systems are expected to include rain or moisture sensors to prevent unnecessary watering. By fulfilling this requirement, the system not only saves water but also aligns with regulatory standards, adding a layer of social responsibility to the business case.

Utilizing Provided Sensors for Garden Monitoring

CQU has provided two key IoT sensors for this project: a **soil moisture sensor** (with integrated soil temperature and conductivity sensing) and an **environmental sensor** (measuring air temperature, relative humidity, and barometric pressure). These form the backbone of the monitoring system, directly addressing the primary data needs for smart irrigation.

- **Soil Moisture Sensor:** This sensor measures the volumetric water content of the soil (outputting a percentage) by assessing the soil's electrical conductivity or di-electric constant (which correlates with moisture). It also reports soil temperature and conductivity in microSiemens/cm. The soil moisture readings directly fulfill the main requirement of knowing how wet or dry the soil is at any given time. By placing this sensor at root depth in a representative spot of the garden, the system can detect when soil moisture falls below an optimal threshold for plant health. If moisture is high (e.g., after rain or recent watering), the system knows additional watering is unnecessary and can be deferred, thus saving water. If moisture drops low, it indicates the need for irrigation. This kind of sensor-driven decision is far more accurate than relying on fixed schedules or surface observations. In fact, Australian home efficiency guidelines specifically recommend using soil moisture sensors with drip irrigation to minimize water use. By continuously tracking the soil's moisture percentage, the system ensures water is only used when the plants truly need it, preventing both under-watering and over-watering.

- **Environmental Sensor (Temperature/Humidity/Pressure):** While soil moisture is the direct trigger for irrigation, environmental conditions greatly influence how quickly soil moisture depletes and how plants respond. The provided environmental sensor (such as a Bosch BME280 or similar) measures:

- **Air Temperature:** High temperatures increase plant water use and evaporation from soil. Monitoring temperature helps the system interpret moisture readings in context. For example, 30% soil moisture on a hot 35 °C day might be more concerning (requiring water soon) than 30% on a cool 15 °C day. Temperature is the most monitored weather parameter in smart irrigation literature, underscoring its importance.

- **Relative Humidity:** Dry air (low humidity) causes quicker evaporation and plant transpiration, whereas high humidity slows these processes. Humidity is the second most monitored parameter in irrigation systems. By tracking humidity, the system can, for instance, hold off watering if humidity is high and plants are less stressed, or conversely be cautious on very dry days.

- **Barometric Pressure:** Atmospheric pressure can indicate weather changes; dropping pressure often precedes rain. While not as directly tied to irrigation as temperature or humidity, pressure readings can be useful. A rising pressure (stable high) usually means clear weather (no rain), whereas a sharp drop could signal an approaching storm. Some advanced irrigation systems do monitor pressure or simply fetch forecast data. In our case, the pressure sensor data is logged for completeness. It could be used to infer if recent conditions were stable or changing, and potentially to cross-verify with any rain sensor data (e.g., rain often accompanies low-pressure systems).

Together, these CQU-provided sensors meet the case requirements by supplying real-time, relevant data. The soil sensor provides the **actionable metric** (soil moisture) to decide when to water, satisfying the requirement of preventing unnecessary irrigation. The environmental sensor provides **supporting metrics** that improve decision quality – enabling smarter scheduling (e.g., watering at dusk or dawn when

temperature is lower and humidity higher, reducing evaporation loss) and a deeper understanding of garden conditions. Using only these two devices, one can already implement a basic smart watering system that significantly outperforms a timer-based approach in efficiency. The fact that Melbourne's rules *mandate* moisture or rain sensors in irrigation systems highlights that these sensors are considered effective tools for water saving. Our system's design around them is well-justified.

Additionally, the provided sensors are low-power and relatively low-cost, aligning with the requirement for an affordable solution. They communicate via a wireless microcontroller (in our case, through MQTT over Wi-Fi) so that they can be deployed in the garden without cumbersome wiring. The soil sensor's inclusion of a temperature reading is valuable to monitor soil microclimate (soil can be cooler or warmer than air, affecting roots). The presence of a **battery voltage** reading (as observed in the data logs) is another practical feature – it allows monitoring the sensor node's battery health remotely. A stable battery voltage around 3.3V in our data indicates the sensors remained powered reliably; if this were to drop, maintenance (battery replacement or recharge) would be needed to keep the system operational. Ensuring continuous data delivery is itself a requirement for a dependable irrigation system, so having a built-in way to monitor power is useful for preemptive maintenance.

In summary, the CQU sensors provide a solid foundation for the smart irrigation case. The soil moisture sensor addresses the *when and how much to water* question directly, while the environmental sensor addresses the *when is best to water and why*. This combination is sufficient for a basic implementation, but for a more comprehensive system, additional sensors can further enhance accuracy and efficiency, as discussed next.

Additional Sensor Recommendations and Justifications

To further improve the system's capability to optimize water use, we propose integrating **three additional sensors**: a **rainfall sensor**, a **solar radiation (sunlight) sensor**, and a **wind speed sensor**. These are not part of the standard CQU kit but are

logical extensions that many smart irrigation systems include for greater precision. Each is justified by its business value (water savings, plant health, or system efficiency) and supported by research:

1. **Rainfall Sensor (Rain Gauge or Rain Detector):** A rainfall sensor provides data on precipitation – arguably the most critical external factor for irrigation decisions. If rain has occurred or is currently falling, irrigation can be skipped entirely, saving water immediately. Conversely, a lack of rain over an extended period may signal the need for more frequent watering. Melbourne's climate can shift; some weeks are wet, others are dry. Incorporating a rain sensor ensures the system responds to these shifts. From a business perspective, this is a high-impact, low-cost addition: simple rain detectors (even just a binary rain switch or a small tipping-bucket rain gauge) are inexpensive, yet they can prevent the significant waste of sprinklers running during or right after a rain shower. In fact, Melbourne's regulations for public spaces require automatic systems to have a rain sensor – this underscores that even policymakers recognize their value. In smart irrigation literature, precipitation is noted as a key parameter determining if irrigation is needed and how much water should be applied. By quantifying recent rainfall, the system could, for example, delay the next watering until the rain's effect diminishes, thus leveraging natural water fully. Business-wise, saving even a single watering due to rain can pay back the cost of a rain sensor in water bill savings. Additionally, avoiding watering during rain improves public perception (sprinklers running in rain are often seen as wasteful or negligent). Overall, a rainfall sensor directly supports the goal of **"water only when necessary"** with concrete data.

2. **Solar Radiation Sensor (Sunlight/UV):** Measuring solar radiation (or ambient light levels) provides insight into evaporation rates and plant water demand. Bright, sunny conditions increase soil moisture loss: solar energy heats the soil and drives evaporation and transpiration. Conversely, overcast days keep soil moist longer. A solar radiation sensor (e.g., a pyranometer or a simple LUX

sensor or UV index sensor) can quantify this effect. For instance, extremely high solar intensity days might prompt the system to water a bit more (or earlier in the day) to compensate for extra evaporation, while low-light days might allow stretching the interval between irrigations. The MDPI survey on smart irrigation notes that luminosity (light intensity) is the third most monitored environmental factor after temperature and humidity, precisely because of its role in raising temperature and causing water loss. The business justification for this sensor is optimization fine-tuning: while not as essential as a rain sensor, a sunlight sensor can improve water efficiency by timing water delivery to when it will be most effective. For example, many guidelines already advise watering in early morning or evening, not under harsh midday sun. A solar sensor could automate this logic: if intense sunlight is detected, the system might avoid midday watering even if moisture is low, waiting for cooler conditions (unless the plant is in distress). Additionally, knowing how much sun a garden bed gets can help in understanding plant water needs (shady areas need less water than those in full sun). In terms of cost, solar/light sensors are relatively cheap; the value they add is preventing waste (no point watering when half of it would evaporate immediately) and protecting plants (wet leaves in strong sun can scorch plants, so better to water at dusk). Thus, this sensor supports the **"water at the right time"** aspect of optimization.

3. **Wind Speed Sensor (Anemometer):** Wind is another environmental factor that can affect irrigation effectiveness. Strong winds can increase evaporation rates significantly and can also cause sprinkler spray to drift away, leading to uneven watering and waste. By measuring wind speed, the system can decide to postpone irrigation during very windy conditions. Some smart irrigation solutions incorporate wind data; for instance, an anemometer was used in research to adjust irrigation timing. From a business standpoint, avoiding irrigation when winds are high can save water that would literally "go with the wind" instead of soaking into the soil. A simple wind sensor (spinning cup anemometer or even an ultrasonic wind meter for more precision) can be integrated. The cost is

moderate, but many weather station kits for hobbyists include one. Its addition would contribute to the **"context-aware irrigation"** capability of the system: watering not just based on soil moisture in isolation, but considering if external conditions might nullify the benefits of watering. For example, if soil is slightly dry but it's extremely windy at the moment, watering might be wasted, so the system could wait until evening when winds typically calm. This level of intelligence can further drive down water usage and ensure the water that is used actually goes to the plants.

In summary, these three sensors – rainfall, solar radiation, and wind speed – complement the existing soil and basic environmental data to form a *holistic picture* of the garden's state and needs. **Rainfall data** directly addresses whether *additional water* from irrigation is needed at all. **Solar radiation data** addresses *when and how water should be delivered*, aligning irrigation with times of lower evaporative loss. **Wind data** adds nuance to *when to water*, avoiding times of inefficiency. The combined business impact is improved water-use efficiency (potentially translating to 20-30% water savings on top of soil-moisture-only control, based on similar smart irrigation system reports) and healthier plants (by preventing stress and diseases from improper watering). These sensors each carry a cost, but their ROI in saved water bills and compliance (for example, many city councils offer rebates for smart controllers that include such sensors) makes them worthwhile. As an alternative to physically adding all these sensors, some systems retrieve weather data from online services; however, having on-site sensors is often more precise for micro-climate conditions in one's own garden. In our case study, we focus on on-site sensing for accuracy and autonomy (no dependency on the internet for weather data).

Implementation Steps (MQTT, Node-RED, InfluxDB, Grafana)

With the key sensors and data requirements established, the next step is implementation. The system follows a typical IoT architecture for sensor data collection and analytics: **Sensors → MQTT broker → Node-RED flow → InfluxDB database → Grafana dashboard**. Each component plays a role in ensuring data moves from the

garden to actionable information for the user. The implementation was done using **MQTT** for communication, **Node-RED** for orchestration, **InfluxDB** for storage, and **Grafana** for visualization. The entire setup leverages open-source technologies, aligning with a cost-effective and flexible solution. The following is a step-by-step outline of the implementation, along with the justification for each major decision:

1. **MQTT Communication Setup:** The sensor nodes (each attached to a microcontroller or IoT device) publish their readings over Wi-Fi to an MQTT broker. We configured a lightweight MQTT broker (e.g., Eclipse Mosquitto) on a local server. Two MQTT topics were defined: for example, `home/garden/soil` carries soil sensor data and `home/garden/env` carries environmental sensor data. MQTT was chosen because it is a **lightweight, efficient publish/subscribe protocol designed for unreliable or low-bandwidth networks** – ideal for IoT devices. It keeps data payloads small and uses minimal overhead, which helps conserve sensor battery life and reduce network strain. Moreover, MQTT's decoupled architecture (sensors publish, subscribers receive asynchronously) makes the system scalable; more sensors or additional subscribers (like a mobile app) can be added without altering the core logic. The broker ensures reliable message delivery with QoS settings, and retains last messages if needed so a dashboard can retrieve the latest reading immediately on connect. In our case, the sensors publish at a set interval (e.g., every 1 minute for soil moisture, every 5 minutes for environment) to balance timeliness with data volume. Security measures (like MQTT username/password and local network isolation) were implemented to prevent unauthorized access. *Justification:* MQTT is a proven standard in IoT, widely used because of its simplicity and reliability for device communication. It allows our sensors to push data in real-time, enabling immediate responses (like triggering an alert or valve) without the need for constant polling.

2. **Node-RED Flow Configuration:** We utilized Node-RED as the central integration hub. Node-RED is an open-source, flow-based programming tool that is

particularly well-suited for IoT applications due to its easy visual interface and library of pre-built nodes. In our implementation, a Node-RED flow was created to **subscribe** to the MQTT topics, **process** the incoming sensor data, and then **forward** it to storage and user interfaces. The flow consists of:

- **MQTT Input Nodes:** These nodes listen on `home/garden/soil` and `home/garden/env`. Each time a sensor publishes, the corresponding node outputs a message containing the sensor readings (e.g., a JSON payload like `{"moisture":31.6, "soilTemp":20.2, "conductivity":1475, "battery":3.30}` for soil, or `{"airTemp":21.7, "humidity":39.5, "pressure":1021.7}` for env).

- **Data Processing Function Nodes:** We included function blocks to parse and format the data as needed. For example, converting raw conductivity to a percentage (if the sensor itself didn't provide a percentage) or rounding values and appending proper units (like adding "°C" or "%" for readability). We also timestamp the data if needed (though MQTT messages may already carry a timestamp). This is also where any *filtering* or threshold-check logic can be applied (discussed more in the next section on filtering). For instance, a function node could check if `moisture < 25%` and then trigger an action (like send an email or set a "needWater" flag) in the flow.

- **InfluxDB Output Node:** Node-RED has a contributed node for InfluxDB which we configured with our database details. The processed data is mapped to an InfluxDB measurement (e.g., named "GardenSensors"). Each data point includes fields for each sensor reading (soil_moisture, soil_temp, soil_conductivity, battery, air_temp, humidity, pressure) and tags if necessary (like `location=garden1` or `sensor=env`). We opted to write both soil and env data into the same measurement for convenience,

distinguishing them by field names. Alternatively, they could be separate measurements – both approaches are viable. InfluxDB was chosen because it is a purpose-built time-series database optimized for sensor data. It handles high write loads efficiently and stores data with a timestamp index for fast time-based queries. It also uses a SQL-like query language and retention policies which are useful for IoT data management. *Justification:* Using Node-RED greatly simplified integration – we could "wire" MQTT to InfluxDB in a matter of minutes using a visual flow, without writing boilerplate code to parse messages and call database APIs. This low-code approach accelerated development and made the system easier to maintain or modify by simply reconfiguring nodes rather than rewriting code. [Insert Screenshot: Node-RED Flow Configuration]

3. In addition to storing data, Node-RED can also directly support a user interface (UI) via its Dashboard nodes. In this project, we focused on Grafana for visualization, but Node-RED Dashboard could be a lightweight alternative for real-time monitoring (e.g., showing a gauge of current soil moisture). Node-RED's role also extends to **control logic**: for example, we set up an email notification node that is triggered by the function node if soil moisture falls below the threshold. This way, the homeowner gets an alert ("Soil moisture low – consider watering your garden") whenever needed. In a full implementation with an actuator, Node-RED could send a command to a smart relay controlling a water valve to automate irrigation when conditions demand it. All these actions are easily managed in the Node-RED flow alongside the data piping.

4. **InfluxDB Data Storage:** We installed InfluxDB (an open-source time-series database) on the local server to store the incoming sensor data. A database (bucket) named "garden_sensors" was created with an appropriate retention policy (for instance, keep data for 6 months before purging, as historical analysis beyond that might not be needed for a garden, or we could archive older data).

Data from Node-RED is written into InfluxDB in real-time. Each entry includes the timestamp (provided by Node-RED or the sensor), measurement name, and sensor fields as described. For example, a data point might look like:

Time: 2025-06-04T03:45:00Z, *Measurement:* GardenSensors, *Fields:* `{soil_moisture: 31.6, soil_temp: 20.2, soil_conductivity: 1475, battery: 3.30, air_temp: 21.7, humidity: 39.5, pressure: 1021.7}`.

The choice of InfluxDB is justified by its **high performance for time-series data** and its rich querying capabilities for time-based analysis (means, mins, maxes over intervals, etc.). It requires no fixed schema, so adding new sensor fields (like rainfall or wind) later is straightforward. We considered alternatives (like a SQL database or cloud IoT platform), but InfluxDB's focus on metrics and time made it ideal. It also integrates easily with Grafana. Another benefit is that InfluxDB can handle downsampling – we could configure it to automatically compute daily averages or other summaries if needed for long-term trends, reducing data volume. In practice, writing each data point from two sensors every minute is trivial for Influx (which can handle thousands per second on modest hardware). The database also ensures data persistence – even if the Node-RED or sensors restart, the historical data is safely stored and can be retrieved for analysis or reporting.

Setting up InfluxDB involved creating user credentials and enabling its HTTP API, which the Node-RED output node uses to push data. We ensured data integrity by catching any write errors in Node-RED (e.g., if the DB was unreachable, Node-RED could queue and retry).

5. **Grafana Dashboard Setup:** Grafana was deployed to turn the time-series data into insightful visualizations. Grafana is an open-source analytics and visualization platform well-known for IoT and operational dashboards. We added

InfluxDB as a data source in Grafana (using the appropriate plugin/connector). Then, we built a **dashboard** for the garden's data. The dashboard contains multiple panels, each graphically representing a sensor metric over time:

- A **Soil Moisture Time-Series Graph**: plotting soil moisture (%) on the y-axis against time on the x-axis. We include a guideline or threshold marker at, say, 25% to visually indicate the "critical moisture" level at which watering is needed. This graph allows the user to see how soil moisture changes through the day and with weather events or irrigation events. For instance, a downward trend might be observed on a hot day, followed by a jump after a rain or watering event.

- **Soil Temperature and Air Temperature Graph**: possibly a combined graph with dual y-axes or two overlapping lines, to show how soil and air temperatures relate. This can reveal, for example, that soil stays cooler than air during the day but retains heat at night, which can affect evaporation rates.

- **Humidity Graph**: showing humidity (%) over time, often inversely correlated with temperature.

- **Air Pressure Graph**: showing barometric pressure (hPa) over time, useful to identify stable vs changing weather patterns.

- **Battery Voltage**: either a graph over time (to catch any gradual discharge) or a single stat panel showing the latest battery voltage with an OK/low indicator.

- We could also create a **combined environmental panel** that plots temperature, humidity, and perhaps an external weather metric together

to illustrate overall conditions.

6. Grafana's visualization capabilities allowed us to format each panel with proper units (e.g., "°C", "%", "hPa", "V"), set time ranges (zoom into last 24h or show last week's trend), and even create alert rules. For example, we defined an alert in Grafana that checks the soil moisture series – if it stays below 25% for more than an hour, an alert could be triggered (Grafana can send emails or notifications). This is an alternative to Node-RED based alerting, and shows how multiple layers of the system can complement each other.

   Grafana was chosen not only for its powerful graphing, but also because it is **interactive and customizable**. Users can hover to see exact values at specific times, apply custom time filters, and even access the dashboard remotely if Grafana is published to a web server. Many industrial and IoT users rely on Grafana as a go-to tool for real-time data visualization. Its plugin ecosystem could allow integration of external data (like weather forecasts) if we wanted to extend the dashboard beyond just our sensors. In our deployment, Grafana runs on the same server as Node-RED and InfluxDB for simplicity, and is accessed via a web browser. We included user authentication on Grafana for security (so only authorized users can view or modify the dashboard).

   Justification: Grafana's strength is in displaying time-series data in an understandable way. This directly meets the requirement of making the system **user-friendly** and aiding decision-making. Instead of raw numbers or logs, the homeowner sees trends and statuses at a glance – for example, a quick look at the moisture graph might show it has been steady around 30% for days, indicating no immediate watering needed. If the graph is trending downward and approaching the threshold, the user knows to prepare for watering. This kind of visual feedback empowers the user to trust the system and adjust their watering behavior accordingly. [Insert Screenshot: Grafana Dashboard Showing Soil

Moisture & Environmental Trends]

7. **Iterative Testing and Tuning:** After initial setup, the system was tested over several days to ensure data is flowing correctly and the dashboards are updating as expected. We deliberately introduced some dry-out scenarios (letting the soil dry to see if alerts trigger) and added water to the soil to see the sensor's response. This helped in calibrating the moisture threshold and ensuring that the Node-RED logic and Grafana alerts correspond to realistic conditions. Implementation is not just a one-off step; it involved refining the flows (e.g., tweaking the MQTT publish interval, adjusting InfluxDB field types, improving graph layouts in Grafana). Each decision in this tuning phase was justified by improving reliability or clarity of data. For example, we decided to implement MQTT Last Will messages for the sensors – so if a sensor goes offline (e.g., battery dies), the broker will publish a notice which Node-RED can catch and display on the dashboard (such as "Sensor offline"). This improves the robustness of the system by handling failure modes gracefully. We also chose specific QoS levels for MQTT (QoS 1 for sensor data to guarantee delivery at least once) to balance reliability and performance.

Throughout implementation, **justification for each component** remained clear:

- MQTT provides a robust, **real-time pipeline** for sensor data with minimal overhead.

- Node-RED offers **rapid integration and ease of automation**, ideal for bridging MQTT to databases and adding control logic without heavy coding.

- InfluxDB ensures **efficient, structured storage** of time-stamped sensor readings, built exactly for this type of data.

- Grafana delivers **insightful visualization and monitoring** that turns data into actionable information.

References

1. Melbourne Water. “Permanent water-saving rules and Target 150.” *Melbourne Water (Victorian Government)*, 2011. *(Details regulations for garden watering; requires automatic systems to use rain or moisture sensors.)*

2. YourHome – Australian Government. “Outdoor water use.” *YourHome.gov.au – Water*, 2022. *(Guidelines for reducing outdoor water usage; notes ~40% of household water is used outdoors and recommends drip irrigation with soil moisture sensors.)*

3. García, L., et al. “IoT-Based Smart Irrigation Systems: An Overview on the Recent Trends on Sensors and IoT Systems for Irrigation in Precision Agriculture.” *Sensors*, vol. 20, no. 4, 2020, article 1042. *(Survey of smart irrigation technologies; identifies key monitored parameters like soil moisture, temperature, humidity, light, precipitation, wind, etc., and common practices in IoT irrigation.)*

4. Robustel. “Smart Irrigation Solution: How to Maximize Water Saving Efforts?” *Robustel Blog*, 2022. *(Discusses how smart irrigation saves water via automated scheduling, soil moisture sensors, and weather-based adjustments.)*

5. EMQX Team. “What Is MQTT and Why Is It the Best Protocol for IoT?” *EMQX Blog*, 28 June 2023. *(Explains MQTT’s lightweight pub/sub model designed for IoT device communication in low-bandwidth or unreliable networks.)*