# Introduction

Prof. Riccardo Berta

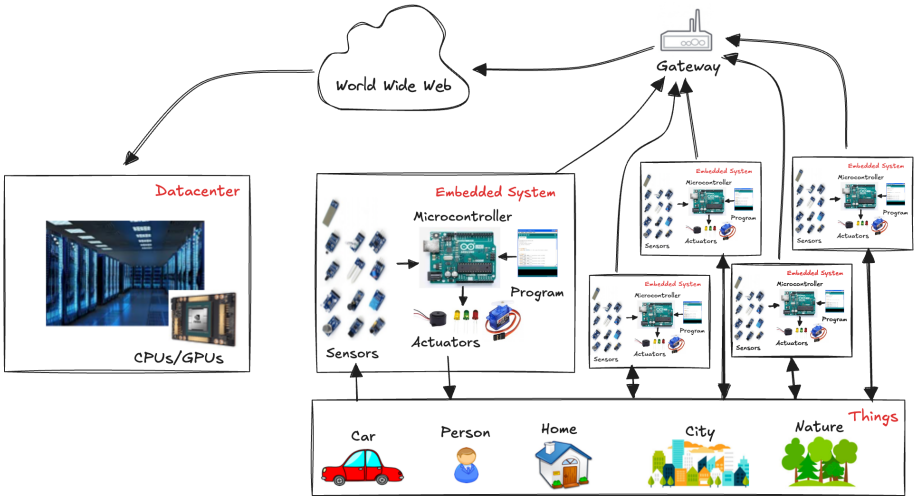
2025-02-26

## Contents

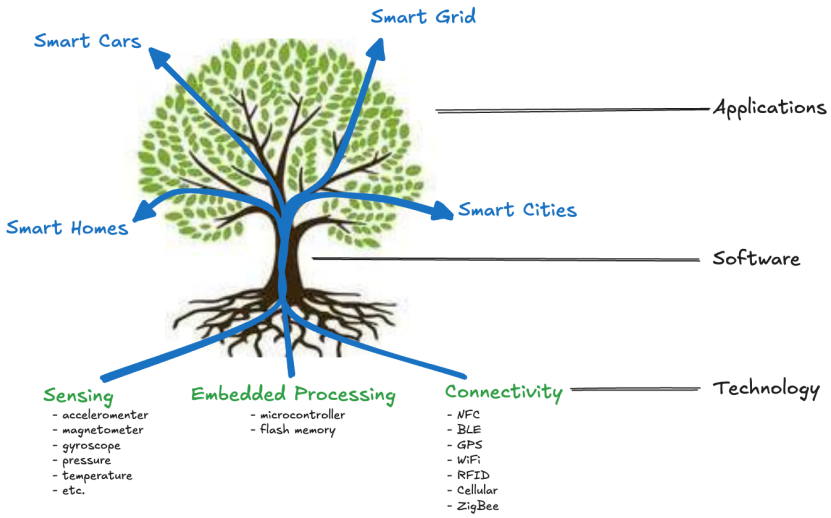
**1 Introduction** **1**
1.1 Issues . . . 4
1.2 Cloud vs Fog computing . . . 5
1.3 Scenario example: the connected hotel . . . 6
1.4 Adopting Web technologies . . . 8
1.4.1 The Web as GUI . . . 9
1.4.2 The Web as an API . . . 10
1.4.3 A GUI for the ”My Office” IoT system . . . 11
1.5 Semantic Gap . . . 15
1.6 Landscape . . . 16
1.7 Hands-on Activity . . . 16

# 1 Introduction

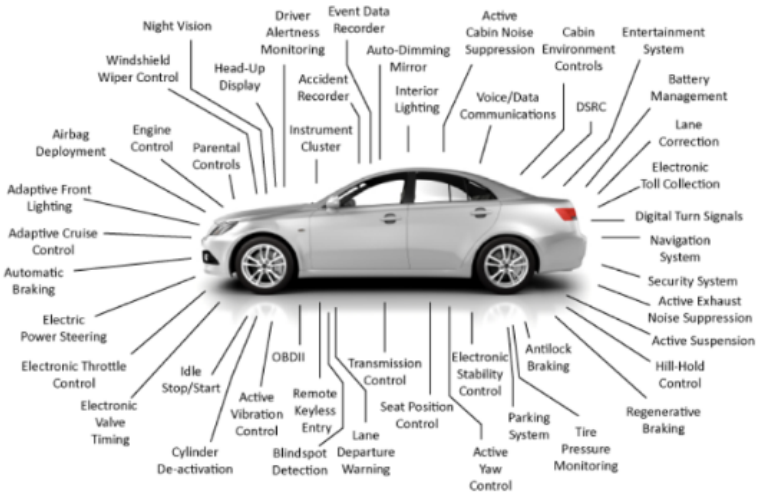
The Internet of Things (IoT) is a concept that describes **a vast network of interconnected physical devices** that are embedded with **sensors**, **actuators**, **software**, and **connectivity** technologies. These devices are capable of **collecting data from their surroundings**, **exchanging it over the internet**, and **acting upon this data to influence the environment**. IoT enables automation, monitoring, and control of systems, making it a powerful tool for creating smarter and more efficient processes across various industries and applications. The following image illustrates a generic IoT system, emphasizing the interconnections of devices and systems across various domains. At the center is the Internet, the network that enables global communication. A gateway serves as the intermediary, connecting the global network to a localized systems. The local network comprises multiple **embedded systems**, each driven by a microcontroller. These microcontrollers execute programs that manage sensors (which gather data from the environment) and actuators (which perform actions based on that data). The data collected by sensors is often transmitted to a **datacenter**, where it is stored and analyzed:




The IoT paradigm consists of three main components: **technologies**, **software**, and **services**.




At the roots, the **foundational technologies**, including sensing components such as accelerometers, magnetometers, gyroscopes, pressure sensors, and temperature sensors, are supported by embedded processing elements like microcontrollers and flash memory, along with connectivity technologies such as NFC, BLE, GPS, Wi-Fi, RFID, Cellular, and ZigBee, which facilitate data exchange. The trunk represents the **software layer**, which acts as the central framework connecting the hardware to higher-level functions. From this foundation, the branches extend to various **IoT applications**, such as smart homes, smart cars, smart grids, and smart cities, symbolizing the diverse use cases and advancements driven by IoT technology. As an exmaple, the integration of IoT in **smart cars** enhances connectivity, enabling real-time data exchange, advanced driver assistance, and seamless interaction with smart infrastructure for improved safety and efficiency:

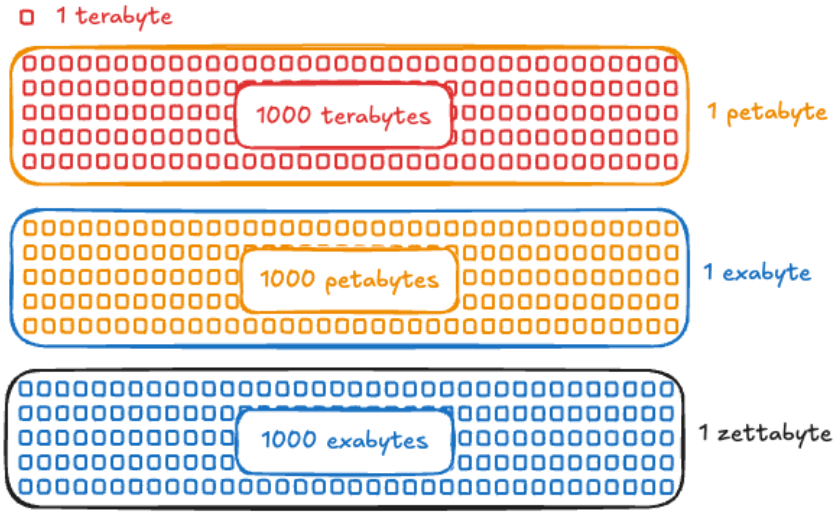



A modern car is today surrounded by an array of advanced features and technologies. Several systems work together to enhance safety, comfort, performance, and connectivity in vehicles. These features include safety systems like airbags, adaptive cruise control, lane departure warning, and blindspot detection. There are also performance-enhancing technologies, such as electronic stability control, regenerative braking, and adaptive front lighting. Additionally, comfort and entertainment are addressed through systems like active cabin noise suppression, voice/data communications, and entertainment controls. Connectivity and monitoring are represented by components like event data recorders, navigation systems, and tire pressure monitoring.

The IoT offers numerous **advantages** that make it a transformative technology. One major benefit is **automation and efficiency**, as IoT systems reduce the need for human intervention while optimizing resource utilization. **Data-driven decision-making** is another significant advantage, as IoT provides real-time insights that can improve operations and planning. Additionally, IoT enhances **quality of life** by introducing smart systems into homes, cities, and healthcare environments, ensuring greater convenience, safety, and overall well-being. As of 2023, the IoT landscape is marked by **remarkable growth and adoption**. There are approximately **16.7 billion IoT devices worldwide**, a figure that surpasses twice the global population and is anticipated to exceed 27 billion by 2025. The global IoT market reflects this rapid expansion, with a valuation of **$662.21 billion in 2023**, projected to surge to approximately $3.3 trillion by 2030. A significant contributor to this growth is the increasing popularity of smart home devices; in 2022 alone, 857 million units were shipped globally, with this number expected to rise to 1.09 billion by 2027. These statistics highlight the pervasive impact of IoT across industries and households, underscoring its role in shaping the future of technology.

## 1.1 Issues

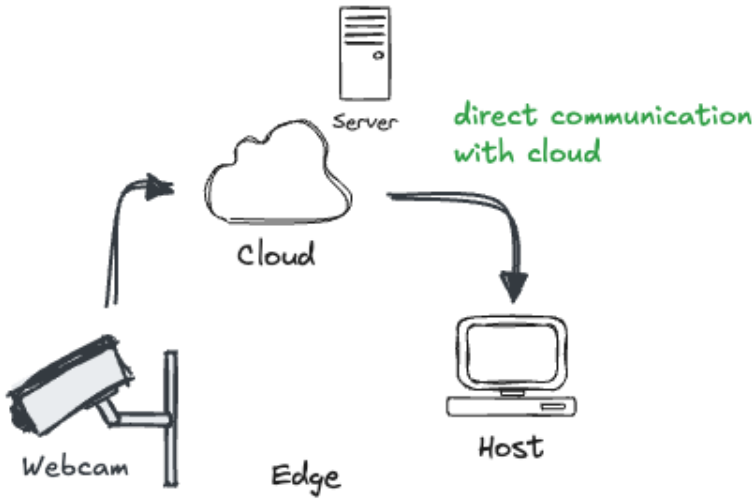
One major issue is the data problem, involving challenges in **volume, variety, velocity, veracity, and value**. IoT devices generate **enormous amounts of data**, often referred to as **big data**, which poses significant challenges in terms of storage, processing, and analysis. The sheer volume of data can overwhelm traditional data management systems:



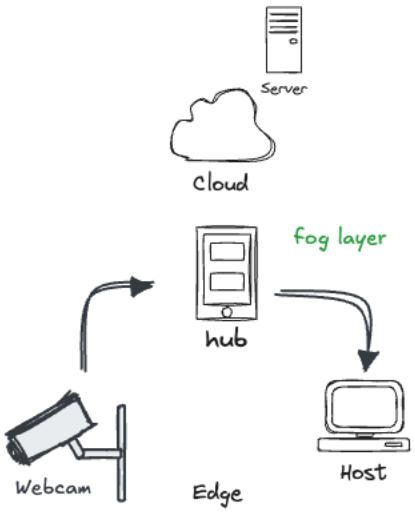

The variety of data is another issue, as IoT devices produce **diverse types of information**, such as sensor readings, video streams, and logs, often in different formats. This diversity makes integration, interoperability, and standardization difficult to achieve, especially when devices come from different manufacturers or use **incompatible protocols**. The **speed at which IoT data is generated** and needs to be processed, is also a significant challenge. Many IoT applications, such as real-time monitoring or autonomous vehicles, require immediate processing of data to make time-critical decisions. The veracity of IoT data, which refers to its **accuracy and reliability**, is another problem. Data collected by IoT devices can be noisy, incomplete, or prone to errors due to hardware malfunctions or environmental factors. Poor data quality can lead to inaccurate analyses, jeopardizing the effectiveness of IoT applications. Finally, the value of IoT data depends on the **ability to extract actionable insights** from it. Raw data often requires extensive preprocessing, aggregation, and analysis to reveal meaningful patterns, trends, or predictions. Additionally, concerns around **data privacy and security** compound these problems. IoT systems collect sensitive information, which, if not adequately protected, could lead to breaches or misuse. Striking a balance between data utility and privacy is an ongoing challenge in the IoT domain.

## 1.2 Cloud vs Fog computing

The existing cloud computing paradigm cannot address the challenges of the rapidly growing IoT ecosystem. **Cloud computing** refers to the delivery of computing services, including storage, servers, databases, and software, over the internet. It allows users to access and pay for services on demand without owning physical infrastructure. Data from devices, such as webcams, is transmitted directly to the cloud for processing and storage. While cloud computing provides a scalable and flexible solution, it can face challenges such as high latency, network bandwidth limitations, and security issues, especially in IoT applications requiring real-time responses:



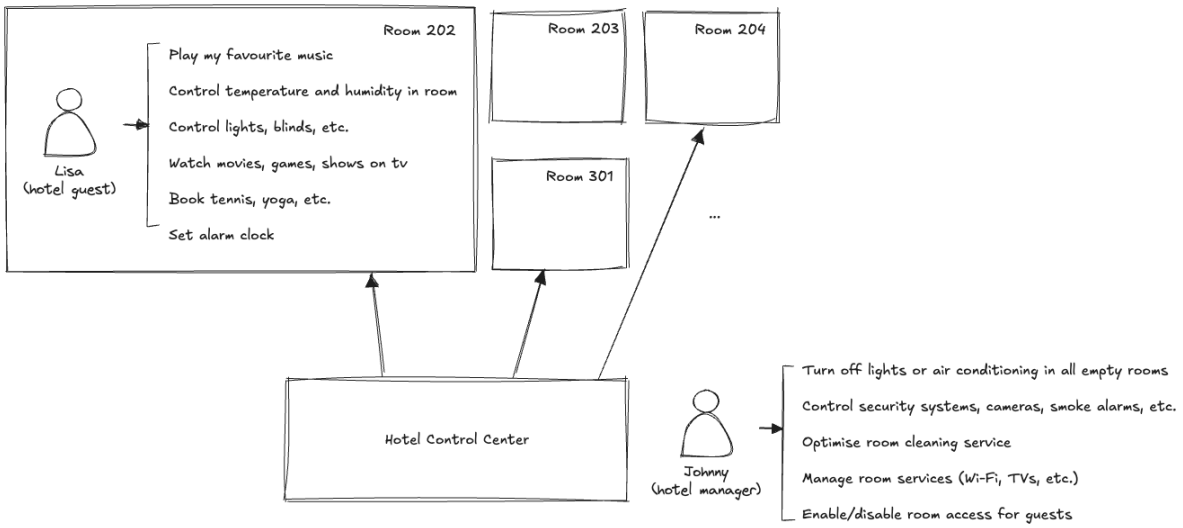

To address these challenges, **fog computing** introduces a **distributed paradigm** that extends cloud computing closer to **the edge of the network**. In this model, an intermediate ”fog layer,” represented by devices such as hubs or fog nodes, **processes data locally** before sending it to the cloud. This approach reduces latency and alleviates network congestion by handling tasks closer to data sources. For example, a webcam in a fog computing system would send its data first to a fog node for initial processing, minimizing the need to transmit large amounts of raw data to a remote cloud:




The growth of IoT underscores the need for fog computing, as the traditional cloud model struggles to sustain the increasing demands of scalability, reliability, and security. By adding a hierarchical layer between devices and the cloud, fog computing optimizes data management and supports the real-time processing capabilities required for modern IoT applications. This hybrid approach integrates the strengths of both cloud and edge computing, providing a more sustainable and efficient solution for the IoT-driven future.

## 1.3 Scenario example: the connected hotel

We consider a smart hotel scenario, where various systems are interconnected to provide enhanced guest experiences and operational efficiency:

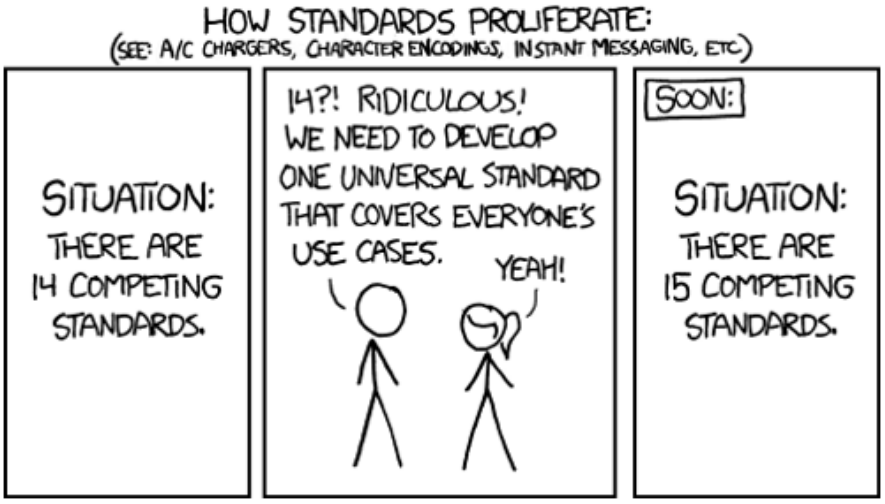


**Guest Perspective**: Lisa can interact with her room environment using a mobile device or in-room controls. She can customize her room settings, including music playback, temperature control, lighting, and entertainment options. She can also book services like tennis or yoga and set alarms.

**Hotel Control Center Perspective**: Johnny has access to a central control center where he can manage various aspects of the hotel. This includes: energy management, security and safety, room services, cleaning efficiency and guest access.

The diagram shows how the guest interface and the hotel control center are connected, enabling two-way communication and data exchange. This allows for real-time monitoring, personalized services, and proactive management of hotel operations. This is a simplified representation of a connected hotel. In reality, the system could be integrated with other hotel systems like reservation, billing, and customer relationship management for a more comprehensive solution.

The smart connected hotel faces **several challenges**. One of the primary issues is the **complexity and time required to integrate disparate systems** and devices into a cohesive network. This difficulty stems from the need to manage varying protocols, data formats, and integration methods unique to each vendor solutions. Additionally, the lack of standardized data formats often leads to **data silos**, where valuable information (such as guest preferences, energy consumption, or maintenance requests) remains isolated, limiting its potential for analysis, sharing, and actionable insights. This fragmentation not only hampers operational efficiency but also introduces **security vulnerabilities**. Systems with differing security protocols and levels of protection increase the risk of cyberattacks and data breaches. A promising solution lies in the **adoption of common standards**, however, a significant risk remains: the **proliferation of competing standards**, a dilemma humorously captured in the following comic:

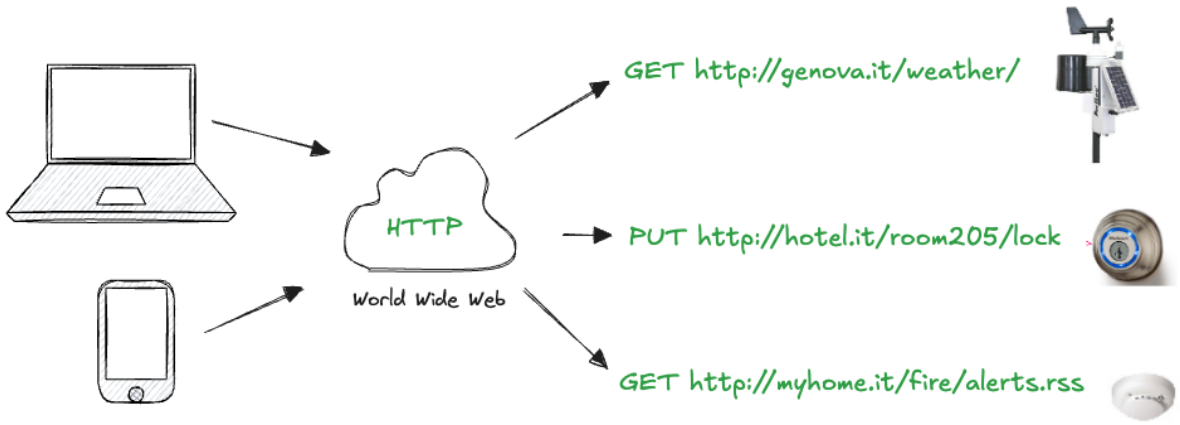



This risk stems from the fact that various organizations and industries have independently developed their own IoT standards, leading to a lack of uniformity across the ecosystem. This lack of consis-

tency poses several challenges. Firstly, it hinders interoperability between devices and systems from different manufacturers, making it difficult for them to communicate and work together seamlessly. Secondly, it complicates the integration process, as systems architects must navigate a maze of disparate standards to connect various IoT components. Finally, this lack of standardization can slow down the adoption of IoT solutions on a broader scale, as businesses and consumers may be hesitant to invest in technologies that lack compatibility and create uncertainty around future upgrades and expansions.

## 1.4 Adopting Web technologies

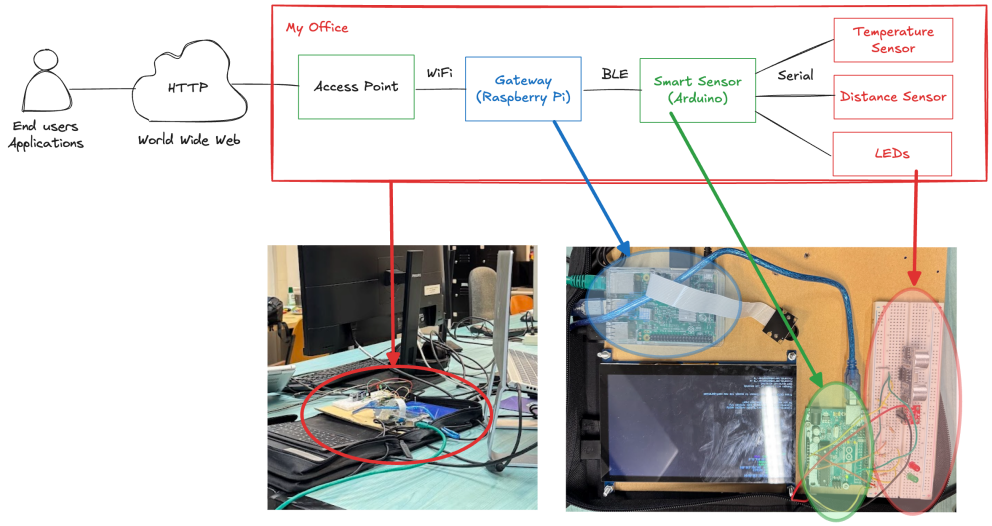
The **Web of Things (WoT)** is a concept designed to address the standardization issues in the IoT ecosystem. IoT encompasses a wide range of devices and technologies, often with different communication protocols, data formats, and functionalities, making it difficult to interconnect and integrate these devices seamlessly. The WoT idea relies on **reusing existing web standards** (such as HTTP, WebSockets, and RESTful APIs), which are already **well-established and widely supported**. By using these standards, IoT devices can communicate over the web, allowing for easier integration with other web-based services and applications.




To ensure devices can understand and exchange meaningful data, WoT emphasizes **semantic interoperability**, which involves defining common data models and vocabularies. This helps ensure that devices with different technical architectures can interpret each other's data in a standardized way. API can be used to allows devices to **expose functionalities in a web-friendly manner**. It provides a high-level interface that simplifies the interaction between devices and applications, making it easier to control and monitor IoT devices from different manufacturers. By adopting open web technologies and common standards, WoT can facilitate device integration across various industries, reducing the complexity and cost of implementing IoT solutions.

Consider a typical end-to-end IoT system, we call it ”My Office”, designed to collect data from various

sensors and transmit it to the cloud for analysis and remote access:




At the core of the system is the **Gateway** (represented by a Raspberry Pi board), which acts as a bridge between sensors and the internet. The gateway gathers data from a smart sensor (and Arduino board) using Bluetooth Low Energy (BLE) wireless communication. The smart sensor provides environmental data (temperature) and detects motion (PIR sensor). Once aggregated, the data is transmitted to the cloud, enabling remote access and control for end-users and applications. The system also supports device control, such as managing leds. This ”My Office” example highlights a fundamental IoT architecture, showcasing the key components involved in collecting, transmitting, and utilizing sensor data.

### 1.4.1 The Web as GUI

In the ”My Office” IoT system, the web can serve as **a user-friendly interface** for interacting with sensors and devices. Imagine a web page displaying real-time data from the temperature sensor, the PIR motion sensor, or even a live video feed from a connected camera:

This web page, accessible from anywhere with an internet connection, enables users to monitor the environment, control devices, and analyze data seamlessly. Users can view current temperature readings, detect motion activity, and check sensor statuses. They can also remotely control devices, such as turning LEDs on or off, adjusting appliance settings, or automating actions like turning on lights when motion is detected. Additionally, the web interface offers tools for analyzing data trends, generating reports, and gaining insights into the system’s performance. The interface is **dynamically generated**, reflecting real-time data and device status. This web-based approach offers several advantages:

- **Accessibility**: it can be accessed from any device with a browser, such as computers, smartphones, or tablets.
- **Ease of use**: the familiar web interface ensures intuitive navigation
- **Remote control**: users can manage and monitor the system from anywhere
- **Data visualization**: information is presented clearly, using charts and graphs for better insights.

By leveraging the web, the ”My Office” IoT system becomes more accessible, interactive, and powerful, offering users a seamless way to monitor and control their environment.

### 1.4.2 The Web as an API

While we often interact with the web through browsers, designed for human consumption, it can also serve as a powerful **Application Programming Interface (API)**. An API defines a set of rules and tools that allow different software applications to communicate and interact with each other. In the context of the ”My Office” IoT system, this means enabling other software applications to interact with the sensors and devices. Imagine a weather forecasting application that wants to access temperature data from the ”My Office” system. Instead of a human browsing the web page to see the temperature, the weather application can directly request this data via an API. This involves sending HTTP requests to specific URLs, similar to how a web browser would, but with the key difference being that the application is requesting data in a machine-readable format like **JSON (JavaScript Object Notation)**, which is a lightweight data-interchange format that uses human-readable text to represent structured data. It's derived from JavaScript, but it's language-independent and can be used by various programming languages. This machine-to-machine communication allows for seamless integration between different systems. The ”My Office” system could expose its data and functionality through a well-defined API, enabling other applications to:

- **Retrieve sensor data**: access real-time temperature readings, humidity levels, motion detection events, and other sensor data.
- **Control devices**: remotely turn on/off lights, adjust thermostat settings, and trigger other actions based on sensor readings or user commands.
- **Build custom applications**: developers can create new applications that leverage the ”My Office” system's data and functionality, such as home automation systems, energy management tools, or security monitoring systems.

The web browser is a tool for humans to surf the web, we need a different tool to interact with APIs. **Postman** provides a user-friendly interface for sending HTTP requests, inspecting responses, and test-

ing API interactions. This allows developers to quickly prototype and test their applications before deploying them. The role of web as an API extends its functionality beyond human interaction, it enables seamless communication between different software systems, opening up a world of possibilities for integration and automation. As an example, with Postman, we can send a GET request to the ”My Office” system to retrieve the list of available sensors or a GET on a specific sensor to get its current temperature reading:

### 1.4.3 A GUI for the ”My Office” IoT system

Let's put on our developer hat and start building an application that interacts with the ”My Office” WoT device. It is not important to grasp all the details of the following source code, but only to get a general idea. We create a simple web page that retrieves temperature data from ”My Office” and displays it to the user (see *01.polling-sensor.html*). In general, we have a page described in HTML and made interactive with **JavaScript** code:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Polling Temperature</title>
  </head>
  <body>
    <h1>Current Temperature</h1>
    <h2 id="temp"></h2>
    <script>
    ....
    </script>
  </body>
</html>
```

The JavaScript code fetches the temperature data from the ”My Office” system using an HTTP GET request and updates the web page with the latest reading:

```
// Ensures the code executes only after the HTML document is fully loaded
document.addEventListener("DOMContentLoaded", () => {

  // Use an asynchronous call to fetch data without reloading the page
  // The endpoint is expected to return a JSON object with temperature data.
  function fetchTemperature() {

    // Use the Fetch API to make an HTTP request to an endpoint,
    // specifying the URL and headers to accept JSON data
    fetch('https://makers.diten.unige.it/iot/sensors/temperature',
          {headers: {'Accept': 'application/json'}})

    // Once the response arrives, the callback functions convert the data in JSON,
    .then(response => response.json())

    // Then pdate the temperature value displayed on the webpage and set a timeout to
        fetch
    // the temperature again
    .then(data => {
```

```
        document.getElementById('temp').textContent = `${data.value} ${data.unit}`;
        setTimeout(fetchTemperature, 1000);
      })

      // If an error occurs, log the error to the console
      .catch(error => console.error('Error:', error));
    }

    // Call the fetchTemperature function to start fetching temperature data
    fetchTemperature();
});
```

This simple example demonstrates how the web can serve as a powerful interface for interacting with IoT devices, enabling real-time data visualization and user engagement. By leveraging the potential and wide availability of web technologies, we can effortlessly develop rich and highly sophisticated GUI that connect seamlessly with IoT systems. For example we can visualize real-time data exploiting the Google Charts library (see *02.polling-sensor-chart.html*):

```
// Loads the Google Charts library and specifies the callback function
// drawChart() to be called once the library is loaded
google.charts.load('current', { packages: ['corechart'] });
google.charts.setOnLoadCallback(drawChart);

function drawChart() {
    // Define the maximum number of data points to display on the chart
    const maxDataPoints = 10;

    // Create a new LineChart object and specify the HTML element to render the chart
    const chart = new google.visualization.LineChart(document.getElementById('chart'));

    // Initialize a table for storing chart data. First row defines the column headers:
    // Time (x-axis) and Temperature (y-axis)
    // Second row adds an initial data point with the current time and a placeholder value
    const data = google.visualization.arrayToDataTable([
        ['Time', 'Temperature'],
        [new Date().toLocaleTimeString(), 0]
    ]);

    // Configure chart properties: title, how to smooths the line between points,
        animation
    // when the chart updates and the position of the legend
    const options = {
        title: 'Temperature',
        curveType: 'function',
        animation: { duration: 1000, easing: 'in' },
        legend: { position: 'bottom' }
    };

    // Add a new data point to the chart. If the number of rows exceeds the maximum,
    // remove the oldest data point before adding the new one
    function addDataPoint(value) {
        if (data.getNumberOfRows() > maxDataPoints)
            data.removeRow(0);
        data.addRow([new Date().toLocaleTimeString(), value]);
        chart.draw(data, options);
    }

    // Fetch temperature data from the server and add it to the chart
    function fetchTemperature() {
        fetch('https://makers.diten.unige.it/iot/sensors/temperature',
              {headers: {'Accept': 'application/json'}})
```

```
        .then(response => response.json())
        .then(result => {
            addDataPoint(result.value);
            setTimeout(fetchTemperature, 2000);
        })
        .catch(error => console.error('Error fetching data:', error));
}

// Start fetching temperature data to update the chart
fetchTemperature();
```

While polling the sensor in the previous code works adequately, it introduces **inefficiency**: rather than repeatedly fetching the temperature from the device every few seconds, it would be far more efficient if the system were **notified** only when a temperature change occurs, and only when it happens. This highlights one of the major impedance mismatches between the **Web traditional request-response model** and the **event-driven architecture that IoT applications** often rely on. The ideal solution to address this challenge is the use of **WebSockets**, which allows for real-time, bi-directional communication between the server and client, enabling the system to receive updates only when a change occurs (see *03.websockets-temp-graph.html*):

```
// Create a WebSocket connection to the temperature sensor.
// Note the wss:// protocol, used for WebSocket connections, which allows for
// persistent, two-way communication between the client and the server.
var socket = new WebSocket('wss://makers.diten.unige.it/iot/sensors/temperature');

// The "onmessage" event handler is registered to listen for incoming messages
// from the WebSocket. When a message is received, the data is parsed from JSON
// and added to the chart data
socket.onmessage = function (event) {
  const cleanedData = event.data.replace(/"/g, '').trim();
  const result = parseFloat(cleanedData);
  addDataPoint(result);
};

// The "onerror" event handler is registered to log any errors that occur during
// the WebSocket connection. If an error occurs, the message 'WebSocket error!' is logged
socket.onerror = function (error) {
  console.error('WebSocket error:', error);
};
```

This setup efficiently addresses the inefficiencies of polling, providing a more scalable and real-time solution. The web page dynamically updates the temperature chart whenever new data is received, ensuring that users have access to the latest information without unnecessary delays. By leveraging WebSockets, we can create responsive, real-time applications that seamlessly interact with IoT devices.

We explored some ways to read sensor data, but what about **writing data to a device**? For example, we may want to send a command to control a device, such as toggling an LED on or off. This introduces the need to send instructions or data to the device, not just read it. One way to do this is by using a simple **HTML form** that sends a **POST request** to the device (see *04.actuator-form.html*):

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Send Message to Actuator</title>
</head>
<body>
  <h1>Send Message to Actuator</h1>

  <!-- The form uses the POST method to send data directly to the device -->
  <form action="https://makers.diten.unige.it/iot/actuators/leds/1" method="post">
    <label for="message">Enter a message for the Led 1</label>
    <input type="text" id="message" name="value" placeholder="{ 'value': false }" required
        >
    <button type="submit">Send to Pi</button>
  </form>
</body>
</html>
```

While this method works well in principle, there's a small issue: browsers do not natively submit a JSON payload body in a POST request. Instead, due to legacy reasons, browsers use a format called "application/x-www-form-urlencoded" when submitting form data. This format is fine for sending simple key-value pairs, but it isn't ideal for sending structured data like JSON objects. Moreover, HTML forms not supports PUT method, only support GET and POST methods. To handle this, we can implement a JavaScript function that overrides the default behavior of the form submission. This function gathers the form data, convert it into a JSON object, and then send it as the body of the POST request to the API. By doing so, we ensure the data is sent in the desired JSON format. Additionally, the function can provide feedback by displaying the result of the operation upon a successful request (refer to *05.actuator-ajax-json.html*):

```
// Add an event listener to the form to handle the 'submit' event
document.getElementById('message-form').addEventListener('submit', function (event) {

  // Prevent the form's default behavior, which is to submit and reload the page
  event.preventDefault();

  // Get the value entered in the input field with id 'message'
  const message = document.getElementById('message').value;

  // Use the Fetch API to send a POST request to the server
  fetch('https://makers.diten.unige.it/iot/actuators/leds/1', {
    method: 'PUT',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ value: message }),
  })

  // Check if the response was successful
  .then(response => {
    console.log(response);
  })

  // Handle the successful response (e.g., log the server's response)
  .then(data => {
    console.log('Message sent successfully:', data);
  })

  // Handle any errors that occurred during the fetch
  .catch(error => {
```

```
29      console.error('Error sending message:', error);
30    });
31  });
```

Abbiamo appena visto come il web possa essere utilizzato per creare un'interfaccia verso i dispositivi, permettendo di interagire con essi in modo semplice ed efficace. Tuttavia, questo non è l'obiettivo principale del corso. Nei prossimi moduli, ci concentreremo su quello che è il nostro scopo principale: esplorare come le stesse tecnologie web possano essere applicate non solo per consumare le API, ma anche per implementarle. In questo modo, acquisiremo una comprensione approfondita dei principi e delle pratiche necessarie per creare API scalabili, universali e applicabili al contesto IoT, utilizzando strumenti e metodologie ampiamente diffusi, non solo HTTP, JavaScript e JSON, ma anche altre tecnologie web.

## 1.5 Semantic Gap

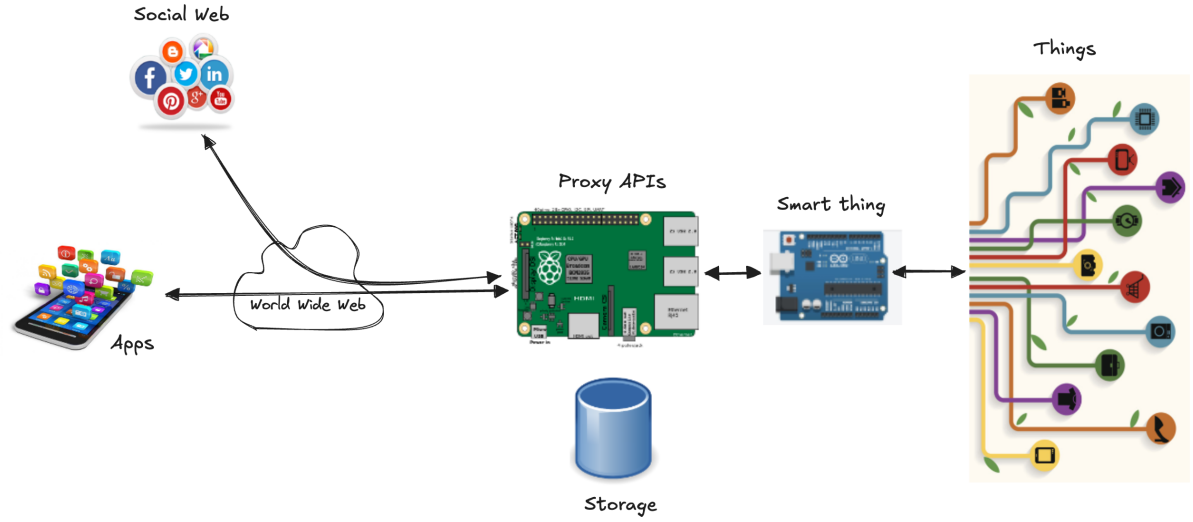
We can consider the challenge of **understanding and interpreting** the meaning of data exchanged between devices and applications. For example, when working with JSON objects representing sensors, we assumed that we already know **what each field means and how to use them**. However, what happens if the only information we have about a device is its internet endpoint and no further details? This lack of shared understanding highlights the **semantic gap problem**. It begins at the **network level** with device discovery: how can an application locate all the devices on a network and determine their endpoints? Without this initial step, communication cannot even begin. Once devices are discovered, the next challenge is for the application to **understand" the devices**, specifically, to determine what sensors or actuators the device offers, the data formats it uses, and the meaning of its properties and fields. One (partial) solution to address this issue is to exploit a fundamental feature of the web: **links**. Just as links in web pages provide a way to navigate and discover content, they can also help applications explore and understand devices. By embedding links and metadata within APIs, devices can guide applications to discover their capabilities and interpret the data they provide. This approach leverages existing web mechanisms to reduce the semantic gap, enabling more seamless and meaningful interaction between devices and applications.

Summarizing, we interact with an embedded device, potentially located on the other side of the world (like in "My Office"), through a simple web page. This page regularly fetches data from a connected sensor, displaying it on a graph, a remarkable capability achieved with just a few lines of HTML and JavaScript. Now, imagine if our Raspberry Pi didn't provide its data via HTTP, JSON, or WebSockets, but instead relied on a machine-to-machine IoT protocol like ZigBee. In such a case, direct communication from a browser would be impossible. We would be forced to write our application in a lower-level language, like C or Java, and forego web technologies such as URLs, HTML, CSS, and JavaScript. This is the essence of the **Web of Things (WoT)** idea: creating APIs for devices that are universally ac-

cessible. By bridging the gap between IoT and the web, WoT leverages widespread web development tools and practices, enabling broader innovation and accessibility in building connected solutions.

## 1.6 Landscape

The following image depicts the system architecture that covers the overall scopes of the course:




At the heart of the architecture is the ”World Wide Web,” the network of interconnected websites and services where web applications and social media platforms reside. A ”Mobile Apps” on a device enable users to access and interact with these web-based services. A ”Proxy API” serve as intermediary, translating data between different systems and protocols, effectively bridging the gap between the web and physical devices. The ”Smart Thing” is a physical devices equipped with sensors and actuators, allowing them to sense their environment and take action. The ”Storage” component refers to a database, which manages and stores data generated by the system, such as sensor readings or user interactions. This system supports diverse interactions, such as a smart thermostat connected to a social media platform, enabling remote control and energy usage notifications, or a sensor-equipped device collecting data and transmitting it to a web application for analysis and visualization. The diagram illustrates the growing phenomenon of the WoT, where everyday objects are becoming increasingly connected to the internet, opening up new opportunities for automation, remote management, and data-driven insights.

## 1.7 Hands-on Activity

1 - Experiment with the several URLs provided by the API http://devices.webofthings.io using Postman. Examine how these URLs are structured, how they differ from one another, and explore the device's

endpoints to understand the data provided by each sensor, including its format and structure.

2 - Consider the electronic devices around you: like the appliances in your kitchen, your TV or sound system, the ordering system at a café, or the train notification board at the station and so on. Imagine a world where the services and data offered by these devices follow a unified structure: consistent endpoints, names, and content formats. Using the JSON structure we've introduced, try to design a similar system for these devices. Map out the names they might use and the JSON objects they could return to represent their data and services.