Proceedings of the 8th European Conference on Industrial Engineering and Operations Management
Paris, France, July 2-4, 2025

Proceedings of the International Conference on Industrial Engineering and Operations Management

Publisher: IEOM Society International, USA DOI: 10.46254/EU08.20250378
Published: July 2, 2025

Integrating External APIs Into Unity: Methods and Applications

Francesca Giuliani and Leonardo Frizziero
Department of Industrial Engineering, Drawings and Methods
Università degli Studi di Bologna
Bologna, Italy
francesca.giuliani14@unibo.it, leonardo.frizziero@unibo.it

Andrea Detratti
IT Department Bonfiglioli Spa
Bologna, Italia
andrea.detratti@bonfiglioli.com

Abstract

The present study investigates methods for integrating external APIs into the Unity development environment, with a particular focus on private API communication for real-time data visualisation from industrial IoT sensors. The research evaluates four approaches—UnityWebRequest, HttpWebRequest, HttpClient, and native JSON parsing—assessing their viability in secure, session-based enterprise scenarios. A detailed case study involving the development of an interactive Unity-based platform serves as the experimental basis for evaluating each method. The results of the study indicate significant limitations with UnityWebRequest, which consistently fails when used with private APIs, returning HTTP 403 errors due to inadequate session and cookie management. Despite the fact that HttpWebRequest offers greater flexibility and manual configuration options, it also exhibited a similar degree of unreliability, repeatedly resulting in HTTP 401 Unauthorized responses, even in instances where session cookies were explicitly configured.
The HttpClient approach was the only one to demonstrate consistent success across all of the tested scenarios. The system exhibited consistent reliability in authentication processes, ensuring session persistence through effective cookie management. Furthermore, it integrated seamlessly into Unity's asynchronous architecture, thereby further enhancing its functionality. Its compact syntax and modern design further enhance its suitability for use in complex, data-driven Unity applications.The findings provide practical insights for the design of immersive, XR-enabled systems and digital twins developed in Unity within the framework of Industry 5.0. Future research directions should include the following: firstly, the identification and resolution of interoperability challenges; secondly, the enhancement of security through the adoption of decentralised architectural models.

Keywords
Unity, API, C#, Data Management, Digital Twin.

1. Introduction
In the contemporary industrial landscape, the evolution towards connected and smart products has become inevitable. In order to maintain competitiveness in the global marketplace, various industries, including mechanical engineering, automotive manufacturing and personal care, must incorporate services into their products. The rapid expansion of the Internet of Things (IoT) has accelerated this transformation, resulting in a demand for user-centred, financially sustainable, and technically feasible solutions that add value for both consumers and producers (Vitali et al. 2017).

The advent of Industry 5.0 has precipitated a paradigm shift in industrial operations, characterised by the integration of human intelligence with advanced technologies. The Internet of Things (IoT) is instrumental in this paradigm, as it facilitates the collection and analysis of real-time data from distributed sensors and devices, thereby enhancing operational efficiency (Blinova et al., 2024). However, the increasing complexity of interconnected devices gives rise to challenges in terms of interoperability, innovation management and security (Rizzo et al., 2016; Del Giudice, 2016). In this context, Application Programming Interfaces (APIs) emerge as vital tools for extending platform capabilities and enhancing flexibility. These tools enable Unity applications to access external data sources, connect with cloud services, and create user-centric interfaces, a capability that is especially valuable in environments such as industrial simulation, virtual reality (VR), augmented reality (AR), and business analytics (Miorandi et al., 2012).

Unity, which was originally developed for the purpose of creating video games, has undergone a significant evolution, resulting in its current status as a versatile cross-platform tool. This transformation has led to its utilisation in a wide range of domains, including enterprise-level applications. The native capabilities of the system, such as real-time rendering and physics simulation, are significantly enhanced through API integration, especially in the context of IoT-related functionalities (Haas, n.d.).

Despite the extensive documentation on Unity's API usage, the majority of resources are focused on open or public APIs. However, in the context of enterprise environments, there is often a requirement to interact with private APIs, which introduce unique challenges, including restricted access and authentication constraints. The present paper addresses these challenges by examining and comparing several API integration methods in Unity, focusing particularly on private business applications.

1.1 Objectives

The primary objective of this study is to identify and validate the most suitable method for integrating private APIs into Unity-based systems, particularly within industrial and enterprise contexts where secure data transmission, session management, and real-time performance are essential. Despite the fact that Unity provides native tools for HTTP communication, these mechanisms frequently prove to be inadequate when interacting with private cloud infrastructures that require robust authentication and persistent session handling.
The objective of this work is to:

- Analyze and compare the behavior of different API integration techniques available in Unity, specifically UnityWebRequest, HttpWebRequest, and HttpClient, with attention to their compatibility with secure endpoints.
- Assess the practical advantages and constraints of each method in handling authentication, session persistence, and data visualisation within private IoT environments.
- Develop and test a Unity platform capable of retrieving and rendering real-time sensor data through API calls secured by session-based authentication.
- Provide concrete implementation strategies and reusable code structures that facilitate integration and reduce development overhead in similar industrial use cases.
- Offer actionable insights to support the development of XR-enabled systems, digital twins, and data-driven industrial applications aligned with the vision of Industry 5.0.

By pursuing these objectives, the study seeks to define a practical and extensible integration model that enables Unity to interact reliably with secure APIs, supporting the broader adoption of immersive and intelligent systems in industrial innovation.

2. Literature Review

The integration of Application Programming Interfaces (APIs) and the Internet of Things (IoT) plays a fundamental role in the development of real-time, data-driven platforms across industries. APIs function as intermediaries, thereby facilitating communication between software components and external data sources. This enables the modular design of scalable systems.

2.1 Internet of Things (IoT)

The Internet of Things (IoT) concept signifies a technological evolution in which devices, sensors, and systems are interconnected to collect, process, and act on real-time data. In industrial environments, the Internet of Things (IoT) plays a pivotal role in the automation of processes, the enhancement of efficiency, and the facilitation of informed decision-making. In the contemporary era of Industry 5.0, the Internet of Things (IoT) has emerged as a pivotal

technology, instrumental in the transformation of operational strategies through the integration of real-time analytics with human-machine collaboration (Blinova et al., 2024).

These smart systems are composed of four layers: data collection via sensors, data transmission through networks, data processing using intelligent software, and actionable implementation based on insights (Batsi and Tennina 2024). The capacity to visualize this data through platforms such as Unity enhances control and user interaction in monitoring systems and simulations (Kok, 2021).

However, the increasing volume and heterogeneity of connected devices give rise to critical challenges, including data security, system interoperability, and scalability (Fatima et al. 2022; Weber 2010). It is imperative that solutions are adaptable, capable of interfacing with diverse hardware and services while ensuring secure and reliable data exchange.

2.2 Application Programming Interfaces (APIs)

APIs facilitate structured communication across applications without disclosing internal logic, thereby enabling the secure exchange of information. These technologies are foundational in the fields of cloud computing, data analytics and Internet of Things (IoT) systems (Dorairajan, 2023). In the context of the Internet of Things (IoT), APIs facilitate the retrieval and processing of real-time data, enabling lightweight and secure interactions optimised for bandwidth-constrained networks through protocols such as the Message Queue Telemetry Transport (MQTT).

REST APIs are particularly favoured for their lightweight architecture and scalability, relying on standard HTTP methods (GET, POST, PUT, DELETE) to exchange JSON- or XML-formatted data (Kumar and Kelly 2021). As demonstrated in Figure 1, RESTful APIs function through structured request-response cycles, wherein clients interact with specific endpoints and receive responses typically in JSON format.

Unity, through scripting in C#, supports various API interaction methods—including UnityWebRequest, HttpWebRequest, and HttpClient—thereby enabling the import and visualisation of data for XR and simulation-based platforms. These capabilities enable Unity to function not only as a development engine, but also as a conduit between IoT data and immersive digital experiences (Miorandi et al., 2012; Kok, 2021).

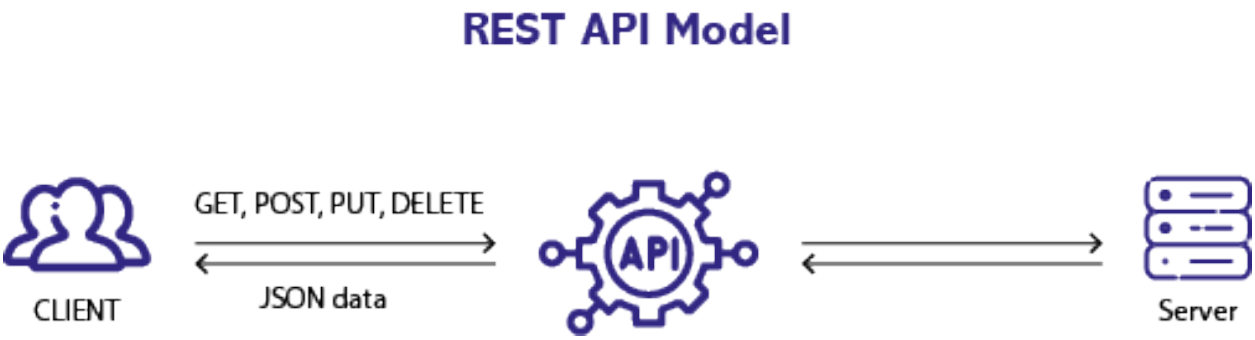

Figure 1 - REST API model. The diagram illustrates the communication process between client and server using RESTful APIs. Each request (e.g., GET, POST, PUT, DELETE) is directed to a specific endpoint, and the server responds with data typically formatted in JS

2.3 Challenges in API Integration

While Unity offers built-in tools for accessing public APIs, integrating private or enterprise-grade APIs remains challenging due to its limited support for secure authentication and persistent session management. UnityWebRequest frequently fails in these contexts, returning HTTP 403 errors due to poor handling of cookies and authentication headers. Similarly, HttpWebRequest, despite offering manual control via CookieContainer, proved unreliable, consistently yielding HTTP 401 errors.

In contrast, HttpClient emerged as the only effective solution, successfully managing session-based POST and GET requests through its built-in asynchronous support and automatic cookie handling. This significantly reduces implementation complexity and enhances session reliability.

Literature and practice both suggest that enterprise-grade IoT platforms demand tailored integration strategies. These require modular, reusable frameworks capable of addressing secure communication, session control, and cross-platform API access—particularly in Unity applications for XR systems, digital twins, and real-time monitoring.

3. Methods

The present methodological framework has been developed for the purpose of evaluating the performance and reliability of different API integration techniques within the Unity environment. The focus of the present study is on the secure and efficient retrieval of data from private servers. The investigation focuses on a detailed examination of three Unity-compatible HTTP methods: UnityWebRequest, HttpWebRequest, and HttpClient. These methods are employed in conjunction with JSON parsing, a process utilised for the management of structured data returned by API endpoints.

All testing was conducted in a controlled Unity environment simulating a representative industrial IoT scenario, where secure authentication and real-time data flow are critical. In order to ensure consistency in interface structure and visualisation components, each API method was integrated within the same Unity scene. This approach enabled a comparison to be made, thus isolating the impact of the communication logic.

The Unity application was organised into three main interface panels (Figure 2):

- A login screen, where user credentials are submitted to the API via POST requests.
- A main menu, providing navigation across different modules for data visualisation.
- A data panel, which dynamically updates in response to API data retrieval.

Figure 2 .From left to right: login screen, main menù screen, data panel menù all developed in Unity. Those are the UI elements that are useful for the correct usage of application

Scene management and API logic were centralised through a dedicated GameObject, Scene Manager, which contained scripts for authentication, request dispatching, and session tracking. The development of all scripts was conducted utilising the C# programming language within the Microsoft Visual Studio environment, thereby ensuring compatibility with standard Unity development practices.

The user interaction flow was structured sequentially:

1. The user initiates a login request through the interface.
2. Following the successful authentication process, a POST request is initiated and a session cookie is stored.
3. Subsequently, authenticated GET requests are initiated to retrieve sensor data or system metrics.

Each API method was evaluated using both public and private endpoints under consistent environmental conditions. The comparison focused on several key performance dimensions:

- The ability to complete authentication and retrieve data successfully.
- The handling and persistence of session cookies across requests.
- The degree of compatibility with private API configurations and infrastructure.

The behaviour that has been observed can be defined in terms of three key areas: execution reliability, response consistency, and error management.
This methodological approach is intended to establish a reproducible benchmark for Unity developers seeking to integrate secure APIs into simulation platforms, XR systems, or Industry 5.0 applications requiring real-time, authenticated data flows.

4. Data Collection

This section describes the experimental framework adopted to evaluate and compare multiple methods for integrating private APIs into Unity. The primary objective was to assess each method's ability to perform secure authentication and reliable data retrieval from a private server, reflecting typical requirements found in industrial IoT and digital twin environments.

4.1 Unity Interface Setup

The Unity application was structured into three primary scenes: a login interface, a navigation menu, and a data visualisation panel. User interface elements—including Canvas, Input Fields, and Buttons—were created using Unity's Legacy UI system, accessible via the Hierarchy > UI > Legacy menu path (see Figure 3).

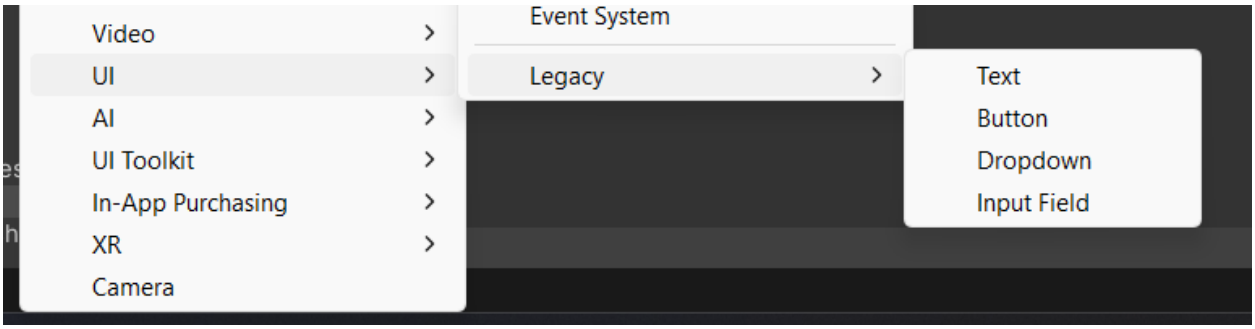

Figure 3. Unity interface to manage the insertion of new UI component

To manage application logic and API interactions, a dedicated GameObject called SceneManager was employed. This object coordinated user flow and hosted scripts responsible for authentication, session handling, and HTTP request dispatching. User credentials entered in the login interface were processed using three distinct communication methods, allowing for consistent comparative evaluation.

4.2 API Authentication Methods

The study evaluated three approaches for integrating private APIs into Unity by testing their ability to perform secure authentication via HTTP POST requests and to retrieve protected data through session-based HTTP GET requests. The objective was to build a Unity-based platform capable of real-time visualisation of IoT data originating from enterprise cloud systems. Those are:

a) UnityWebRequest, Unity's native HTTP interface, offers basic functionality but lacks support for advanced session management. In all tests involving private APIs, it consistently failed to maintain valid authentication sessions, returning HTTP 403 Forbidden errors. The requirement to manually insert cookies proved unreliable, confirming its incompatibility with secure enterprise environments.

b) HttpWebRequest allows greater control through headers and a CookieContainer, but it also proved ineffective in this context. Despite correctly extracting and reattaching session cookies, every authenticated request using HttpWebRequest returned HTTP 401 Unauthorized errors. These failures indicate that the method does not properly preserve session integrity when interacting with private APIs, and its use within Unity appears fundamentally incompatible with the authentication expectations of modern secure servers.

c) HttpClient, by contrast, was the only method that successfully handled both authentication and data retrieval. POST requests correctly initiated sessions, and subsequent GET requests consistently retrieved protected data without error. HttpClient's automatic cookie management and compatibility with Unity's asynchronous programming model made it a robust and developer-friendly solution. As shown in the next paragraphs, it achieved stable authentication flows and it is able to elaborate a successful GET request and live data

integration into Unity's UI components. As shown in Figure 4, it achieved stable authentication flows; Figure 5 illustrates a successful GET request and live data integration into Unity's UI components.

```
public async Task<bool> Login(string email, string password)
{
    string uriPath = "identity/users/login";
    string json = $@"
    {{
        \"email\": \"{email}\",
        \"password\": \"{password}\"
    }}";

    var request = new HttpRequestMessage(HttpMethod.Post, "https://example.com/api/login?apiKey=YOUR_API_KEY")
    {
        Content = new StringContent(json, Encoding.UTF8, "application/json")
    };

    HttpResponseMessage response = await client.SendAsync(request);

    if (response.IsSuccessStatusCode)
    {
        string cookie = response.Headers.GetValues("Set-Cookie").FirstOrDefault();
        PlayerPrefs.SetString("cookie", cookie);
        return true;
    }
    else
    {
        return false;
    }
}
```

Figure 4. HttpClient POST method using C# in unity to access the platform and the cloud

```
public async Task FetchData(string thingId, string metricName)
{
    string uriPath = $"v2/identity/users/me/things";
    string url = $"https://example.com/api/{uriPath}";

    var request = new HttpRequestMessage(HttpMethod.Get, url);

    string cookie = PlayerPrefs.GetString("cookie");
    if (!string.IsNullOrEmpty(cookie))
    {
        request.Headers.Add("Cookie", cookie);
    }

    HttpResponseMessage response = await client.SendAsync(request);

    if (response.IsSuccessStatusCode)
    {
        string resBody = await response.Content.ReadAsStringAsync();
        Debug.Log("Response Code: " + (int)response.StatusCode);
        Debug.Log("Response Body: " + resBody);

        JObject data = JObject.Parse(resBody);
        DisplayData(data);
    }
    else
    {
        Debug.LogError("Error: " + response.StatusCode);
    }
}
```

Figure 5. Data fetching using GET request and session token using HttpRequest to fetch data

All retrieved data—typically in JSON format—was parsed using the Newtonsoft.Json library and dynamically mapped to Unity interface panels for real-time display. For reproducibility, the complete implementation code for all three methods is publicly available at: https://github.com/thatsfrancigiuli/POST-GET-Unity.git

5. Results and Discussion

This section presents an analysis of the API integration methods tested in Unity, with particular focus on their applicability in secure, session-based environments such as industrial IoT platforms and digital twin systems.

5.1 Numerical Results

While all three integration strategies—UnityWebRequest, HttpWebRequest, and HttpClient—were evaluated, quantitative metrics were collected only for the functional implementation based on HttpClient. UnityWebRequest and HttpWebRequest consistently failed to complete the authentication process and were therefore excluded from performance benchmarking.

For the working method, HttpClient, precise timings were recorded for both login (POST) and data retrieval (GET) operations across multiple test runs. The average duration for authentication requests was 175 ms, while data retrieval requests completed in 387 ms (Figure 6). These values reflect consistent and reliable behavior under the test conditions, validating the efficiency of HttpClient in managing session-based communication.

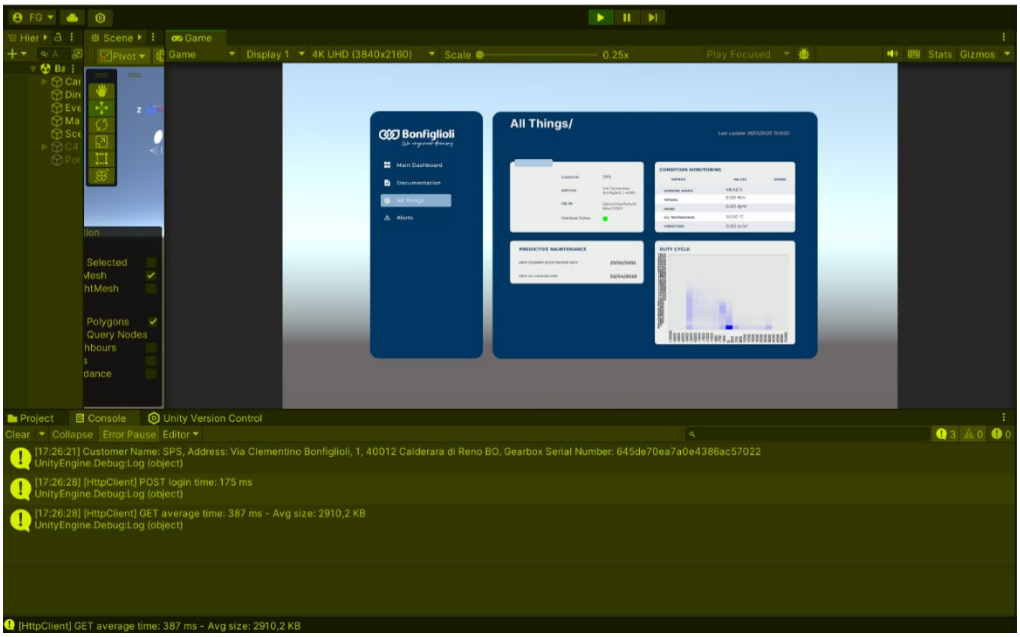

Figure 6 .Debug of HttpClient timing in POST and GET method. POST method is used for authentication process and GET method is used for fetching data.

In contrast, UnityWebRequest and HttpWebRequest encountered critical errors—HTTP 403 and HTTP 401 respectively—which prevented any successful session establishment or data access. Consequently, no valid performance metrics could be obtained for these methods.

5.2 Graphical Results

To visually demonstrate these outcomes, a comparative analysis is presented. As shown in Figure 7, UnityWebRequest failed during the GET phase, returning repeated HTTP 403 errors due to its inability to manage session cookies. Figure 8 captures the response from HttpWebRequest, which returned HTTP 401 Unauthorized, highlighting its failure to maintain valid session state despite proper cookie configuration.

In contrast, Figure 9 displays the console output for a successful HttpClient GET request. The retrieved JSON data is correctly parsed and rendered within Unity's interface, confirming session persistence and end-to-end authentication success.

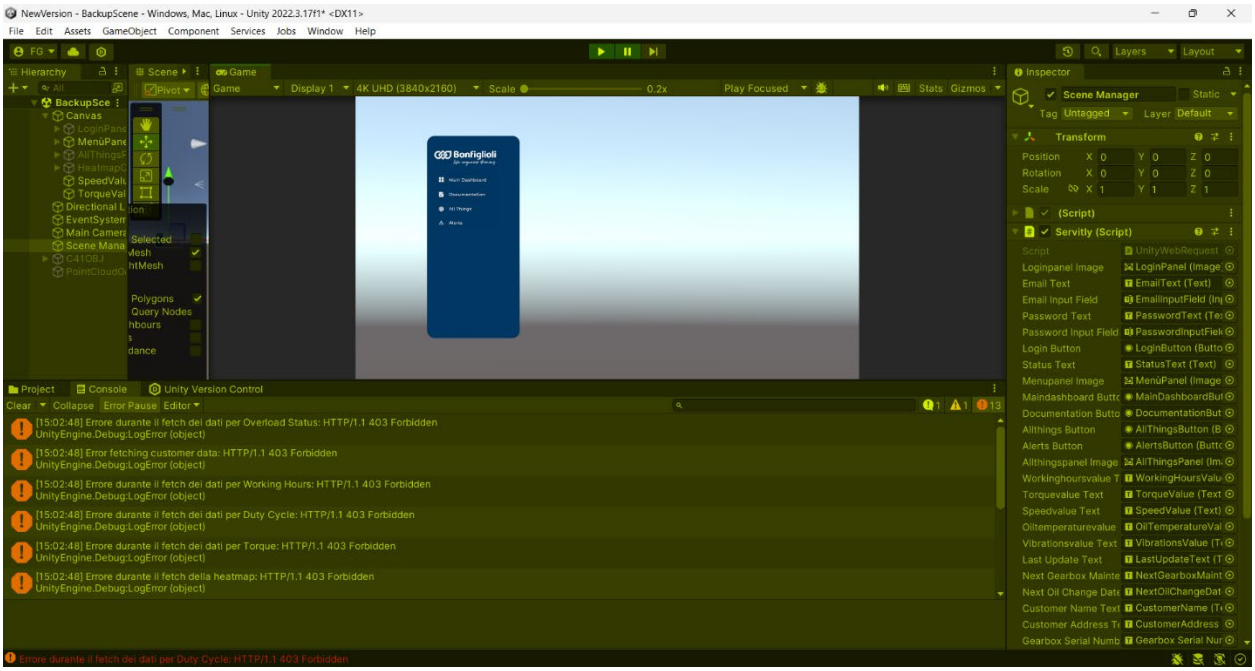

Figure 7. UnityWebRequest, the process fail to gain data with 403 error.

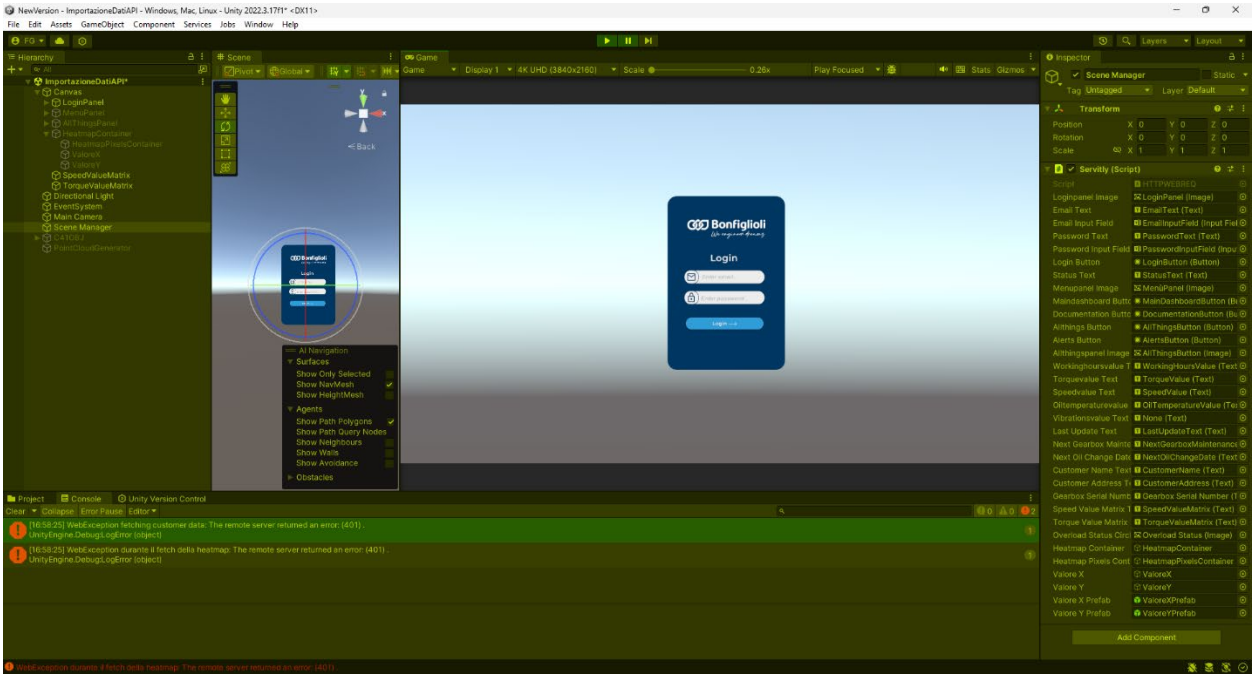

Figure 8 . HttpWebRequest method fails in authentication process with an 401 error.

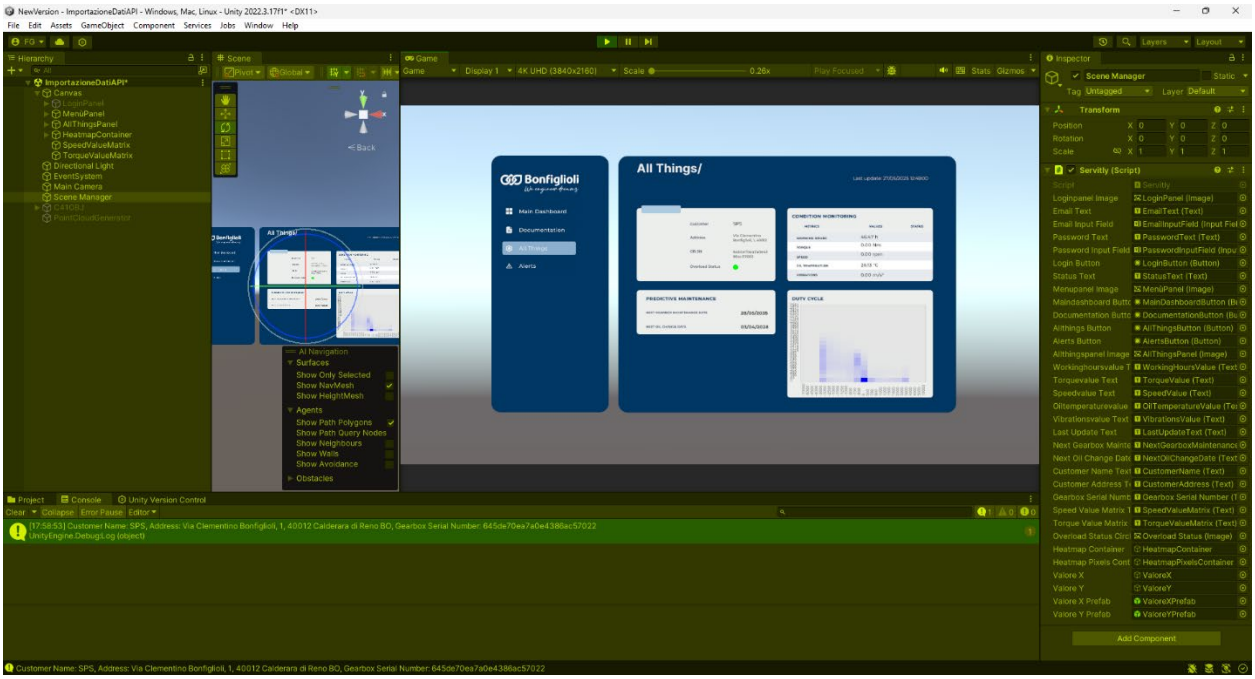

Figure 9. HttpClient is the suggest method for its ease of use and reliability.

Although no formal statistical tests were applied, the results were consistent and reproducible across all test iterations. The numerical and visual evidence clearly supports the conclusion that HttpClient is the only viable method for private API integration in Unity environments requiring secure session management and real-time performance, also supported by debug output.

5.3 Proposed Improvements

The experimental results revealed critical limitations in Unity's native and low-level .NET-based API methods. UnityWebRequest lacks proper cookie management and fails authentication with private endpoints, while HttpWebRequest is unable to maintain valid sessions despite explicit configuration.
To overcome these limitations, the integration workflow was restructured around the use of HttpClient.
Key improvements included:

- Construction of API requests using HttpRequestMessage for granular control over headers and payloads
- Automated management of authentication cookies and session state
- Use of PlayerPrefs to store tokens or session identifiers across runtime sessions

This approach enabled the Unity application to:

- Authenticate securely using POST requests
- Retrieve protected data via authenticated GET requests
- Visualize structured JSON content in real time within Unity's user interface

The HttpClient-based implementation not only resolved the issues observed with other methods, but also introduced a scalable and modular foundation for secure API integration in Unity.

5.4 Validation

To validate the consistency of the implementation, a cross-platform verification was conducted using Postman, replicating the exact API calls and credentials employed in Unity, after the debug text. The validation process confirmed:

- The structure and content of JSON responses in Unity matched those returned via Postman
- Authentication cookies were handled identically across both environments
- Status codes and error messages were consistently aligned

Although formal hypothesis testing was not applied, the repeated success of authenticated GET and POST requests using HttpClient strongly supports the robustness and generalisability of the proposed method. This confirms its applicability to a wide range of secure, session-based Unity applications in industrial or enterprise contexts.

6. Conclusion

This study systematically evaluated different strategies for integrating external APIs into Unity, with a focus on environments that require secure authentication, session persistence, and real-time data exchange—conditions typical of industrial IoT and enterprise platforms.

Three HTTP communication methods were tested: UnityWebRequest, HttpWebRequest, and HttpClient. UnityWebRequest was found to be inadequate for private API integration, consistently failing to handle session-based authentication and returning HTTP 403 errors. HttpWebRequest, despite its configurability via CookieContainer, also failed to preserve valid sessions, resulting in repeated HTTP 401 Unauthorized responses. Only HttpClient proved capable of successfully completing both authentication and data retrieval operations.

HttpClient's automatic cookie handling, native asynchronous support, and streamlined syntax made it the only reliable and maintainable solution for secure API integration within Unity. These findings offer practical guidance for developers working on digital twins, XR interfaces, and real-time dashboards in industrial contexts.

Aligned with the goals of Industry 5.0, this research supports the integration of intelligent systems with immersive technologies, contributing to the development of more secure, flexible, and adaptive Unity applications. Future directions include the implementation of token-based authentication schemes such as OAuth2, the adoption of real-time protocols like WebSocket or MQTT, and deeper integration with Unity's XR toolkits to support interactive and scalable industrial systems.

References

Aldoseri, A., Al-Khalifa, K. and Hamouda, A., A roadmap for integrating automation with process optimization for AI-powered digital transformation, Preprints, 2023. https://doi.org/10.20944/preprints202310.1055.v1

Aslam, F., Aimin, W., Li, M. and Rehman, K. U., Innovation in the era of IoT and Industry 5.0: Absolute Innovation Management (AIM) framework, *Information (Switzerland)*, vol. 11, no. 2, pp. 1–17, 2020. https://doi.org/10.3390/info11020124

Blinova, T., Singh, D., Kaur, N., Prasanna, Y. L. and Acharya, P., IoT-driven innovations: A case study experiment and implications for Industry 5.0, *BIO Web of Conferences*, vol. 86, 2024. https://doi.org/10.1051/bioconf/20248601071

Del Giudice, M., Discovering the Internet of Things (IoT) within the business process management: A literature review on technological revitalization, *Business Process Management Journal*, vol. 22, no. 2, pp. 263–270, 2016. https://doi.org/10.1108/BPMJ-12-2015-0173

Dorairajan, A., What is an API? Application Program Interface Definition, *Axway*, 2023. Available: https://www.axway.com/en/api-management

Fatima, Z., Tanveer, M. H., Waseemullah, Zardari, S., Naz, L. F., Khadim, H., Ahmed, N. and Tahir, M., Production plant and warehouse automation with IoT and Industry 5.0, *Applied Sciences (Switzerland)*, vol. 12, no. 4, pp. 2053, 2022. https://doi.org/10.3390/app12042053

Haas, J., A history of the Unity game engine, Worcester Polytechnic Institute, IQP Report, 2014

Hussain, F., Hussain, A., Shakeel, H., Uddin, N. and Ghouri, T. L., Unity Game Development Engine: A Technical Survey, 2020 https://sujo.usindh.edu.pk/index.php/USJICT/article/view/1800

Kelly, S. and Kumar, K., RESTful APIs, in Unity Networking Fundamentals: Creating Multiplayer Games with Unity, *Apress*, pp. 55–89, 2022. https://doi.org/10.1007/978-1-4842-7358-6_3

Kok, B., Beginning Unity Editor Scripting: Create and Publish Your Game Tools, *Apress Media LLC*, 2021. https://doi.org/10.1007/978-1-4842-7167-4

Lombardi, M., Pascale, F. and Santaniello, D., Internet of Things: A general overview between architectures, protocols and applications, Information (Switzerland), vol. 12, no. 2, pp. 1–21, 2021. https://doi.org/10.3390/info12020087

Miorandi, D., Sicari, S., De Pellegrini, F. and Chlamtac, I., Internet of Things: Vision, applications and research challenges, *Ad Hoc Networks*, vol. 10, no. 7, pp. 1497–1516, 2012. https://doi.org/10.1016/j.adhoc.2012.02.016

Sai Reddy Dwarampudi, V. S. and Mandhala, V. N., Social media login authentication with Unity and Web Sockets, *Proceedings of the 2023 International Conference on Computer Science and Emerging Technologies (CSET)*, pp. 1–5, 2023. https://doi.org/10.1109/CSET58993.2023.10346940

Strousopoulos, P., Troussas, C., Krouska, A. and Sgouropoulou, C., Architecting immersive education: Designing an intelligent online virtual university, in *Proceedings of the 4th International Conference on Novel and Intelligent Digital Systems (NiDS 2024)*, Springer, pp. 346–356, 2024. https://doi.org/10.1007/978-3-031-73344-4_29

Tuhaise, V. V., Tah, J. H. M. and Abanda, F. H., Technologies for digital twin applications in construction, *Automation in Construction*, vol. 152, pp. 104931, 2023. https://doi.org/10.1016/j.autcon.2023.104931

Unity Technologies, Unity - Manual: Retrieving text or binary data from an HTTP Server (GET), 2021. Available: https://docs.unity3d.com/6000.0/Documentation/Manual/web-request-retrieving-text-binary-data.html

Unity Discussions, UnityWebRequest vs HTTPClient - Unity Engine, 2022. Available: https://discussions.unity.com/t/unitywebrequest-vs-httpclient/886948

Biographies

Francesca Giuliani is a PhD student in Mechanical and Advanced Sciences of Engineering (DIMSAI) at the Department of Industrial Engineering (DIN), University of Bologna, in collaboration with Bonfiglioli S.p.A. She holds a Bachelor's degree in Industrial Product Design and a Master's degree in Advanced Design from the University of Bologna (2018–2023). Her academic background combines design thinking with engineering methodologies, with a particular interest in innovation and sustainability. Her Master's thesis focused on the redesign of the Lancia Beta HPE with electric propulsion, applying the TRIZ and IDeS methods to explore innovation strategies in the automotive sector. This project allowed her to deepen her understanding of car design processes, system evolution, and product-service integration. Her current research focuses on the development of an Internet of Things (IoT) platform for industrial applications. She is actively studying Digital Twin technologies, artificial intelligence (AI), and augmented reality (AR) for monitoring, diagnostics, and predictive maintenance of mechanical systems. Her work explores how real-time data can be imported into digital environments to enhance human interaction with complex technical models, particularly in industrial and manufacturing contexts. Francesca is also interested in methods for optimizing industrial production systems, organisational models, and human-centred design approaches. She is committed to bridging the gap between design and engineering to develop more intelligent, responsive, and user-oriented industrial technologies.

Leonardo Frizziero, a mechanical engineer, is a family man, having been married and fathered three children. He is an advocate for scientific discourse on subjects pertaining to industrial engineering design and methods. Since 2005, he has been employed at Ferrari Spa, where his role has involved the coordination of methods, times and costs for new car projects. He obtained his PhD in 2009 through collaboration with the Industrial Engineering Design and Methods research group and was promoted to the position of Junior Researcher in 2013. The primary areas of research focus on two distinct domains: The relationship between car design and health is a subject that has been the focus of numerous studies and research. In these fields, he collaborates with automotive companies and health facilities, using advanced technologies for 3D virtual representation and digital twin, static and dynamic. Since 2018, he has been employed as a Senior Researcher. Since 2017, he has been qualified as an Associate Professor of Industrial Engineering Design and Methods, assuming his position in 2021. In 2021, he assumed the role of Seminar Chair at Lawrence Tech University in Detroit, where he delivered a lecture series on the subject of 'Car Design'. Prior to assuming academic appointments, he accumulated significant professional experience in relevant industry roles. As of June 2023, he has attained the designation of Full Professor. In February 2024, he was elevated to the status of Fellow of the IEOM Society International, Michigan, USA.

Andrea Detratti is an IoT Cloud Development Specialist at Bonfiglioli Riduttori S.p.A., where he designs and implements innovative software solutions to drive industrial digitalization. He holds a Bachelor's degree in Electronic and Information Engineering and a Master's degree in Computer and Automation Engineering from the University of Ferrara, developing a strong expertise in the integration of computer science and industrial automation. During his academic journey, Andrea focused on industrial connectivity and 5G applications, exploring solutions such as multi-connectivity and offloading to enhance system performance. His Master's thesis investigated the use of emerging technologies to optimize network traffic management in industrial contexts.
In his current role, Andrea combines technical skills and organizational capabilities to develop cloud and edge computing solutions. Leveraging tools like Microsoft Azure and Grafana, he contributes to the monitoring and optimization of business processes, with a particular focus on scalability and efficiency. His experience also includes collaborating with internal teams and external partners to implement technological innovation strategies. Passionate about automation and digital transformation, Andrea is dedicated to exploring new technological frontiers

to improve human-machine interaction, contributing to the development of increasingly intelligent and connected industrial systems.