THE GUIDELINE FOR RAPID SOFTWARE GAME DEVELOPMENT ON UNITY WEB-BASED ECOSYSTEM

Vatit Simakupt

A Thesis in Partial Fulfillment of the Requirements
for the Degree of Master of Science Program in Information Technology
Graduate Studies
Thai-Nichi Institute of Technology
Academic Year 2025

Thesis Topic	The Guideline For Rapid Software Game Development On Unity Web-Based Ecosystem
By	Vatit Simakupt
Field of Study	Information Technology
Thesis Advisor	Dr. Puwadol Sirikongtham

The Graduate Studies of Thai-Nichi Institute of Technology has been approved and accepted as partial fulfillment of the requirement for the Master's Degree.

……Vice President for Academic Affairs
(Assoc. Prof. Dr. Warakorn Srichavengsup)
Month…… Date…… Year……

Thesis Committees

……Chairman
(Assoc. Prof. Dr. Worapat Paireekreng)

……Committee
(Acting. Sub. L.T. Dr. Pichitchai Kamin)

……Committee
(Dr. Apichaya Nimkoompai)

……Advisor
(Dr. Puwadol Sirikongtham)

VATIT SIMAKUPT : THE GUIDELINE FOR RAPID SOFTWARE GAME DEVELOPMENT ON UNITY WEB-BASED ECOSYSTEM. ADVISOR : DR. PUWADOL SIRIKONGTHAM, 50 PP.

The rapid growth of content-driven games has created a pressing need for workflows that allow fast, flexible, and collaborative content development. Traditional approaches, which rely on Unity Editor and server-client architectures, often introduce bottlenecks: artists and designers must wait for programmers to implement changes, testing is limited to centralized environments, and granting access to source code poses security risks. These limitations hinder rapid iteration and reduce the efficiency of multi-disciplinary teams.

This study proposes a web-based content editing workflow for Unity games that leverages runtime content injection, sandboxed testing environments, and collaborative web interfaces. The system enables immediate testing and iteration of art assets, narrative sequences, and level designs without direct access to the game engine or source code. By integrating JavaScript with Unity Web-Target, the workflow allows real-time content adjustments while preserving security and asset integrity.

The proposed system was evaluated through quantitative measures, comparing it to traditional CMS-based workflows in terms of ease of use, collaboration efficiency, and task completion time. Results demonstrate a significant improvement in workflow efficiency, reducing both personnel requirements and iteration times. Weighted improvement ratios across different game genres—including casual, strategy, rhythm, and MMO—showed consistent enhancement in collaborative productivity.

Graduate Studies Student's Signature……

Field of Study Information Technology Advisor's signature……

Academic Year 2025

Acknowledgement

I would like to express my sincere gratitude to my thesis advisor, Dr. Puwadol Sirikongtham, for his valuable guidance, constructive suggestions, and continuous encouragement throughout the course of this research. His expertise and insightful feedback were essential to the successful completion of this thesis.

I would also like to thank the faculty members of the Graduate School, Thai-Nichi Institute of Technology, for providing the academic knowledge, resources, and supportive learning environment that contributed greatly to my studies and research development.

My appreciation is extended to all colleagues, classmates, and friends who offered helpful discussions, suggestions, and moral support during the research process. Their cooperation and shared experiences made this journey both productive and meaningful.

Finally, I would like to express my deepest gratitude to my family for their constant encouragement, understanding, and support throughout my academic journey. Their motivation and patience have been a vital source of strength in completing this work.

Vatit Simakupt

Table of Contents

Table of Contents (Continued)

Table of Contents (Continued)

List of Tables

List of Figures

Chapter 1
Introduction

1.1 Background

The contemporary game development industry is evolving at an increasingly rapid and complex pace, particularly in the realm of content-based games, which require the continuous creation and updating of game content to provide players with fresh and engaging experiences. In such games, content production cannot be delayed or slowed, as any disruption directly impacts game continuity and player satisfaction.

Most large-scale games today are designed using a server-client architecture, which allows the latest content updates to be delivered to players in real-time. While this architecture facilitates content distribution, it also introduces several challenges in content creation and management. One major challenge is the reliance on game engine editors, such as Unity Editor, for content development. Although these tools are powerful and comprehensive, they have inherent limitations: they often demand high computational resources, are difficult for non-technical team members to use, and lack the flexibility offered by web-based interfaces.

Moreover, granting direct access to the game's source code for artists or content creators—solely to enable Unity Editor usage—poses potential intellectual property risks and increases management complexity. Modern web technologies, however, provide an effective solution to these challenges. Web applications are easy to access, deploy, and update, while enabling the creation of content editing tools that operate independently of the main game engine. This separation allows programmers and artists to work concurrently without creating bottlenecks in the development process.

Leveraging the capabilities of Unity Web-Target in combination with JavaScript, developers can control and modify game behavior at runtime, enabling immediate testing of new content. This approach allows the use of data from both servers and local machines, and supports content creation in a sandboxed environment, enhancing collaborative efficiency.

A review of related research indicates that there has been no comprehensive, flexible workflow specifically designed for content-driven game development on a Unity Web-based ecosystem. This study, therefore, focuses on exploring and defining a system for web-based content editing in Unity games, investigating runtime data injection methods, supporting sandboxed content testing, and establishing workflows suitable for different game genres. The expected outcomes include faster development cycles, improved collaboration efficiency, and reductions in both time and personnel required for joint tasks.

1.2 Problem Statement

Despite the rapid advancement of game development tools, content-based game production remains time-consuming and resource-intensive. Large-scale games often rely on traditional CMS workflows combined with game engine editors like Unity, which require technical expertise and centralized server access. This setup creates several bottlenecks: artists and content designers must wait for programmers to implement changes, testing content in real-time is limited, and sharing updates across teams often leads to delays.

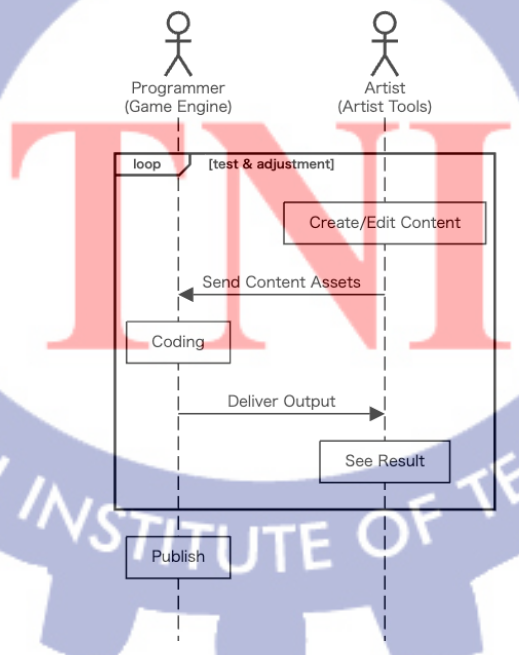

Figure 1.1 Sequence Diagram of Traditional Workflow

Furthermore, granting direct access to the game's source code poses security and intellectual property risks, restricting the flexibility of workflow design. While server-client architectures allow real-time content delivery, there is no established workflow that integrates runtime content injection, sandboxed testing, and collaborative web-based editing. This gap hinders rapid iteration and reduces the efficiency of cross-disciplinary teams, particularly when multiple content types—such as art assets, narrative sequences, and level design—must be tested and refined concurrently.

The lack of a flexible, web-based workflow for Unity games thus limits the potential for streamlined content creation, rapid prototyping, and scalable collaboration, making it a critical problem for studios aiming to maintain a competitive edge.

1.3 Research Objectives

This study aims to address the challenges identified above by developing a web-based content editing workflow for Unity games.

The key objectives are:

A) To design and implement a system that allows runtime content injection, enabling immediate testing and iteration of game assets.

B) To develop a sandboxed testing environment where artists and designers can experiment with new content independently of the main game engine.

C) To evaluate the system's impact on collaboration efficiency, including reductions in personnel requirements and task completion time.

D) To provide guidelines for integrating web-based content editing into different game genres, ensuring that workflow benefits are applicable across casual, strategy, rhythm, and MMO games.

By meeting these objectives, the study seeks to create a workflow that enhances productivity, reduces development friction, and preserves the security and integrity of game assets.

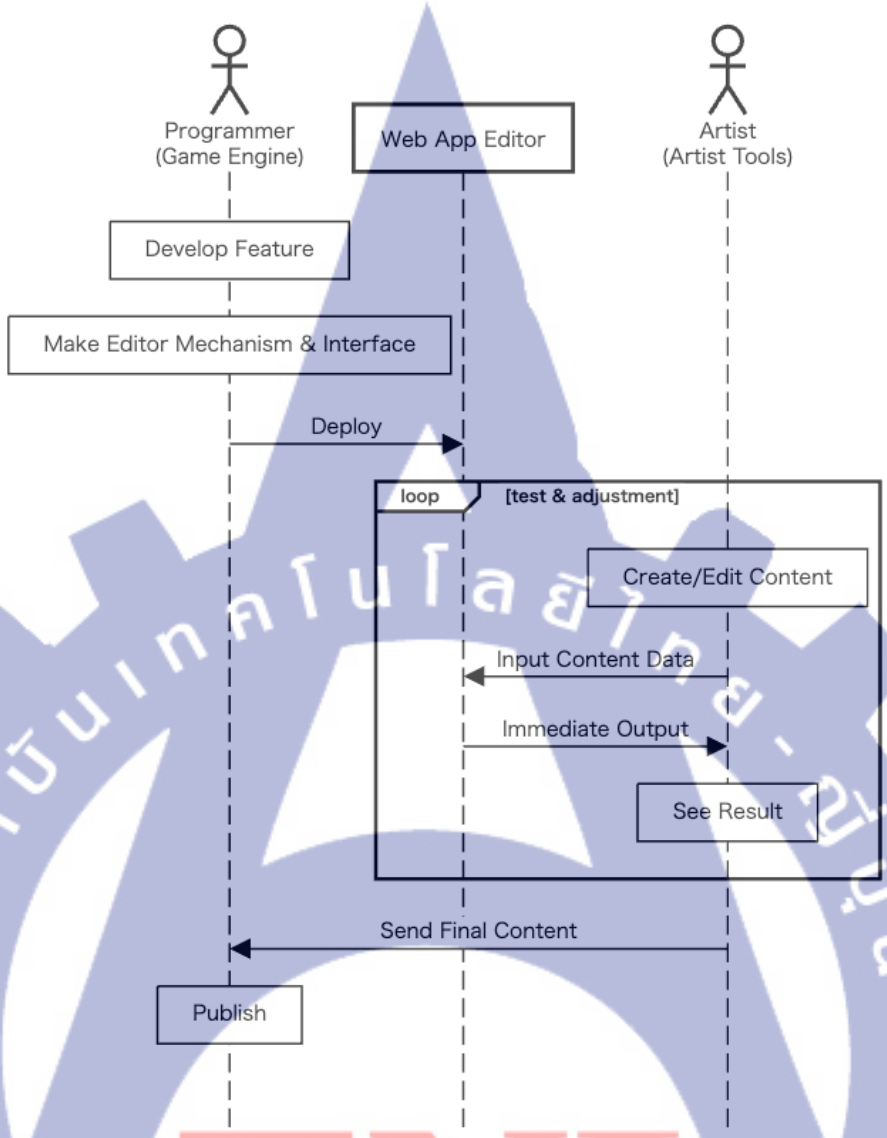

Figure 1.2 Sequence Diagram of Our Proposing Workflow

1.4 Scope and Limitations

The scope of this study is specifically focused on Unity-based, content-driven games and the creation of a web-based content editing workflow. The research will focus on three primary content types:

A) Art assets, including 2D graphics and 3D models.

B) Narrative content, such as dialogues, story sequences, and in-game text.

C) Stages or levels, encompassing layout, objectives, and gameplay mechanics.

Limitations of the study include:

A) The workflow will be implemented and tested in a controlled sandbox environment, not in a fully deployed live game.

B) Performance metrics will focus on collaboration efficiency and development time, rather than direct player experience or server scalability.

C) Integration with other game engines besides Unity is not covered, though principles may be adaptable.

These boundaries ensure that the study remains feasible while providing meaningful insights into workflow design for Unity games.

1.5 Significance of the Study

This study contributes to both academic research and practical game development by proposing a flexible, web-based workflow for content creation. For developers and studios, the system promises to accelerate iteration cycles, reduce dependency on technical personnel, and minimize bottlenecks in asset and content testing. For artists and designers, the workflow offers independent and sandboxed access to content modification tools, increasing creative freedom and reducing delays.

From a research perspective, the study demonstrates a methodology for runtime content injection, web-based editing, and collaborative sandbox environments, which can inform future studies in game development processes, CMS design, and workflow optimization. Ultimately, the study seeks to bridge the gap between technical implementation and content creation, providing a scalable, efficient, and secure approach for modern game development teams.

Chapter 2
Related Theory and Literature Review

This chapter expands on the theoretical foundations and practical concepts that underpin the proposed web-driven workflow for Unity-based, content-focused game development. It synthesizes ideas from software engineering, game production pipelines, web technologies, and human–computer interaction to explain why a web-based editor and runtime injection model can materially improve speed, collaboration, and safety in content creation. Where appropriate, key terms are defined and their practical implications are illustrated with short examples so the later design decisions and evaluations are easier to follow.

2.1 Theoretical Foundations

2.1.1 Game Development Workflow

Game development is an inherently multidisciplinary, multi-phase activity that requires tight coordination between programmers, designers, artists, sound engineers, QA, and often non-technical content creators. To reason about improvements to that process we separate the workflow into two complementary perspectives: the temporal phases that any game project traverses, and the domain-specific areas of work that run in parallel across those phases.

A) Phases of Game Development

Concept Phase

The concept phase establishes the high-level vision: the genre, target audience, core mechanics, and the emotional or experiential goals of the title. A clear concept reduces wasted work later by constraining the required content types and the technical architecture. For content-heavy games this phase also identifies update cadence (how often new content will be released), which in turn affects decisions about tooling and deployment.

Design Phase

Design translates concept into concrete plans: game design documents, data schemas (how levels, items, dialogue are represented), content

pipelines, and tooling needs. A design that anticipates frequent content updates will deliberately favor modular data formats (e.g., JSON, CSV, scriptable assets) and APIs that allow partial updates without full rebuilds.

Implementation Phase

Implementation is the construction phase: core systems, engine-side logic, networking, and integration with asset pipelines. Implementation must produce stable, well-documented hooks (APIs, events, data contracts) that content tools can rely on. In a web-driven model, implementers also expose safe runtime extension points so editors can inject content or swap assets during execution.

Testing Phase

Testing validates gameplay, balance, and technical correctness. For content-focused workflows, testing must include the ability to preview content in realistic game contexts without full builds: iteration speed is critical. This motivates local sandboxes and runtime data injection so artists and designers can test content in-situ.

Publishing Phase

Publishing covers release to players and ongoing live operations: content updates, hotfixes, and telemetry-driven iteration. A web-based content toolchain simplifies continuous updates and remote configuration because the central server can push new UI/tool versions and content without requiring end users to re-download clients.

Each phase benefits from tools and interfaces tailored to the team's needs. The central thesis is that web interfaces reduce friction across several phases — particularly design, implementation, testing, and live-ops — by enabling lightweight, role-specific tools that do not require direct access to engine projects or heavy local installs.

Figure 2.1 Visualizing the Flow of Development Phases

B) Areas of Game Development

Work in game projects also divides by specialty; each area has distinct constraints that affect how tooling should be designed:

Gameplay Mechanics

Encompasses code that implements movement, combat, AI, and rule systems. Mechanics require precise, deterministic behavior and often live in engine code. For content-driven iteration it is beneficial to separate pure data (tunable parameters, ability definitions) from code so designers can iterate without changing compiled logic.

Level Design

Designers create levels, scenarios, and progression. Level data tends to be hierarchical and spatial; designers need WYSIWYG editors, visual previews, and the ability to quickly validate changes in context. Web UIs can offer intuitive drag-and-drop editors and previews that interface with a runtime sandbox.

Secondary Content Editing

Includes dialogue, cutscenes, UI text, localizations, audio cues, and particle tweaks. These tend to be lower risk and more amenable to web-based editing because they seldom require changes to engine logic. When designed well, secondary content can be fully authored via the web without programmer intervention.

By mapping these areas to lightweight web tooling and clearly defined interfaces to the engine, teams reduce coupling and bottlenecks: artists and content creators can perform many iterations independently while programmers focus on core systems and exposed integration points.

2.1.2 Game System Architecture

Client Authoritative

In a client-authoritative architecture, the primary execution of game logic and state management occurs on the client side, meaning that the player's local device assumes responsibility for processing the majority of interactions and updates within the game environment. This architectural model is typically employed in single-player applications or multiplayer systems that demand extremely low-latency performance, where immediate responsiveness is prioritized over centralized control. By delegating authority to the client, this approach minimizes communication delays with a remote server, thereby enhancing the smoothness and immediacy of user experience. However, such decentralization also introduces significant challenges related to data integrity, synchronization accuracy, and vulnerability to manipulation, as the client holds substantial control over the game state.

Within the context of the proposed workflow, it is imperative to ensure that content produced by non-technical users—for example, through intuitive game design interfaces or visual scripting tools—remains fully compatible with client-side execution systems. The workflow must incorporate robust validation and compatibility checks to prevent desynchronization, logical conflicts, or performance degradation when user-generated content is integrated into the game. Ultimately, the design of this workflow should safeguard against inconsistencies between locally generated content and the broader gameplay experience, maintaining both stability and fidelity of interaction across all clients.

Server Authoritative

Conversely, a server-authoritative architecture centralizes control of game logic and state management on the server, which operates as the definitive arbiter of all gameplay actions and world updates. This model is predominantly utilized in multiplayer or competitive online games, where maintaining fairness, consistency, and protection against cheating are of paramount importance. In such systems, client devices primarily function as interfaces for capturing player input and displaying the server's validated outcomes, while all authoritative computations occur on the server itself. This configuration substantially enhances security, consistency, and synchronization across all connected players, as it prevents unauthorized client-side modifications that could compromise gameplay integrity.

Within the scope of the proposed workflow, this architecture necessitates the development of tools and processes that enable non-technical users to create or modify content that integrates seamlessly with the server's authoritative logic. The workflow must ensure that user-generated content can be efficiently synchronized with the server state, maintaining logical coherence and avoiding discrepancies between the player's local representation and the central game environment. Consequently, the design must facilitate an alignment between user-driven creativity and the server's regulatory mechanisms, preserving both the integrity of the multiplayer experience and the technical consistency of the overall system.

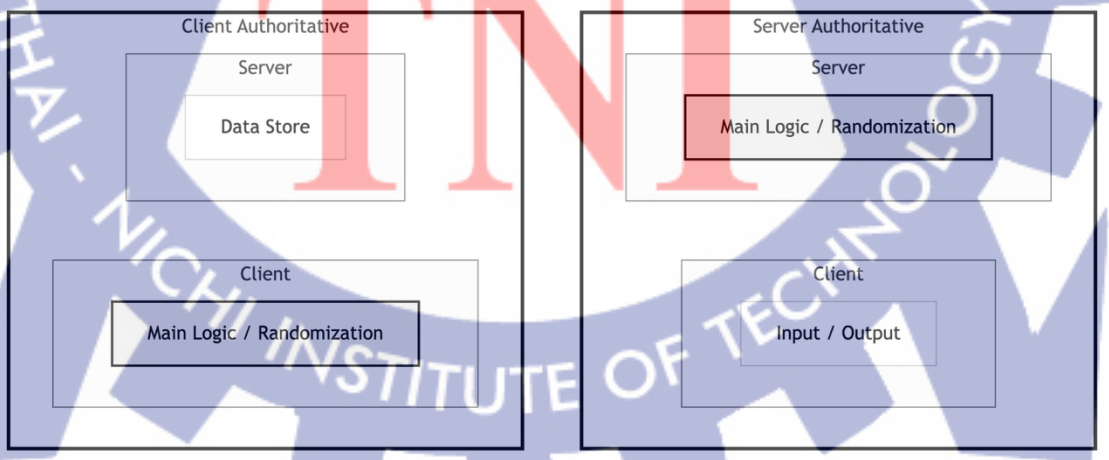

Figure 2.2 Comparison of Server vs Client Authoritative Models

2.2 Related Technologies

2.2.1 Game Engine and Platforms

A game engine is the integrated software stack that provides rendering, physics, input, animation, audio, scripting, and content management facilities. Engines differ from primitive multimedia libraries in scope and tooling: engines offer a higher-level, cohesive environment designed specifically for interactive real-time experiences.[1]

Game Engines vs. Primitive Multimedia Libraries

Scope and integration: A game engine bundles subsystems (rendering, physics, animation, audio, networking, UI) and coordinates them through a common lifecycle. Multimedia libraries typically expose isolated functionality (e.g., audio playback, 2D canvas) but leave the orchestration to the developer.

Content tools: Engines ship with editors (scene editors, animation editors, asset importers) that speed up content creation. These editors are powerful but are often heavyweight and assume technical familiarity. Web-based editors trade the full feature set for accessibility and role-specific simplicity.

Runtime facilities: Engines provide optimized pipelines (asset streaming, memory management, serializing game state) that multimedia libraries lack. Any web-based content solution must interoperate with these pipelines safely — for example by using prescribed asset formats and caches rather than attempting to bypass engine invariants.

Figure 2.3 Example of Major Multimedia Libraries

Major Game Engines

Unity — Widely used for both 2D and 3D projects [2,3]; supports multiple platforms including WebGL/WebAssembly builds. Unity's flexible component model and scripting in C# make it a practical candidate for exposing a

limited, safe interop surface to web-based editors (for example, native function calls and controlled WebRequest interception).

Unreal Engine — Strong in high-fidelity graphics and complex simulation; often used for AAA titles where web deployment is less common. Unreal's C++-centric architecture implies different integration strategies, but the same principle applies: expose constrained, well-documented interfaces to external tooling.

Godot — Open-source, light-weight engine with an approachable scripting environment; well-suited for small-to-medium projects and for teams that want to design custom editors. Godot's open nature makes it straightforward to add custom web-interfacing layers.

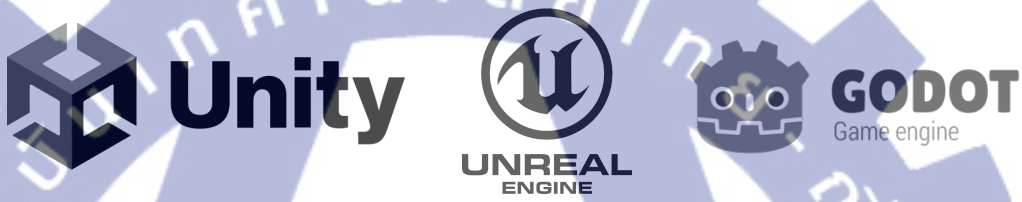

Figure 2.4 Example of Major Game Engines

Choice of engine should factor in the team's target platforms, required graphics fidelity, and the engine's extensibility for web interop (ability to expose safe runtime hooks, customizable asset loaders, and efficient WebAssembly/WebGL builds).

2.2.2 Web Technologies and Interoperability

Web Application and Frameworks

Web applications provide an accessible and platform-agnostic approach for building editing tools that complement — rather than replace — engine editors. They are particularly advantageous for enabling non-technical users to author, preview, and manage content without needing a local installation of heavy software.

Modern Complex UI/UX Ecosystem

Modern web frameworks allow creation of sophisticated, domain-specific interfaces [4]: visual level editors, node-based behavior editors, timeline editors for cutscenes, and asset manager dashboards. Because web UI libraries

are modular, teams can design role-based experiences (artist, designer, writer) that surface only the controls each role needs, improving learnability and reducing errors.

For example, a web level editor might provide:

- A simplified 2D/3D viewport for quick placement and snapping of objects.
- Parameter panels for configuring enemy waves or item spawn rules.
- Previews that call into a runtime sandbox to show the effect of changes immediately.

These features reduce iteration time and the cognitive load on content creators.

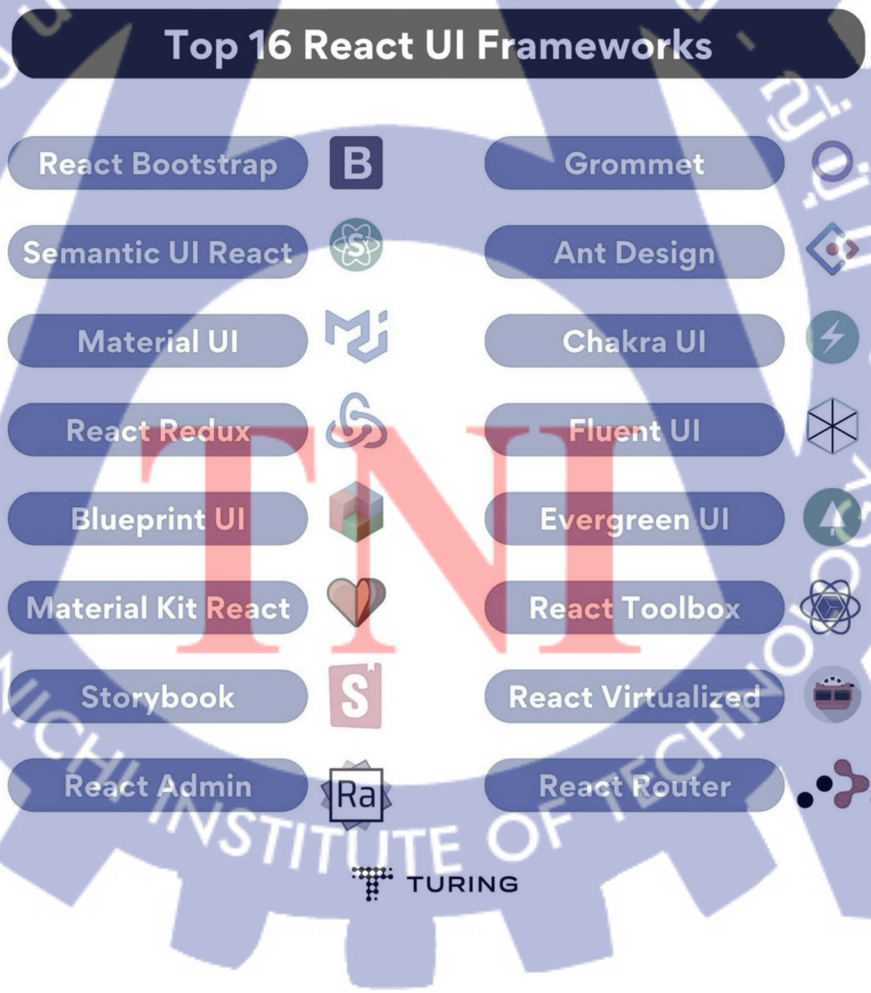

Figure 2.5 List of Top JavaScript UI Frameworks

Easy Access and Deployment

Web apps run in browsers across desktop and tablet devices, eliminating the friction of installing or updating local software. Centralized deployment means improvements to the editor reach all users immediately; permissions and access control can be managed server-side. For distributed teams, browser-based tools also facilitate remote collaboration, reviewing, and QA.

Integration with Modern Web Technologies

Web applications can interoperate with backend services, CDN storage, and browser-native APIs (drag & drop, file APIs, WebSockets). Important techniques for a content workflow include:

- Data injection / runtime overrides — Temporarily injecting content into the running client so creators can test changes without committing them to the central server. This is often implemented by sending JSON payloads or Blob URLs to the runtime and instructing the engine to load from an in-memory cache rather than the production CDN.
- HTTP-based workflows — Using REST or GraphQL APIs to persist content, fetch metadata, and trigger validation pipelines.
- Local sandboxing — Allowing the editor to store a local copy of modified content and test it in an isolated client instance so creators can iterate privately before committing.

These integrations make web-based editors powerful and flexible while still preserving the engine's authoritative code and performance characteristics.

Programming Language Interoperability

For a web editor to communicate robustly with a compiled engine, multiple interoperability strategies can be combined. Interoperability enables the engine and web UI to exchange data and call functions across language boundaries safely and efficiently.[5]

ABI – Application Binary Interface

ABI specifies how functions are called and how data structures are laid out in memory at the binary level. When embedding compiled modules (such as WebAssembly that originated from C#), respecting ABI conventions is necessary to

call into engine-provided functions reliably. In practice, content tools avoid low-level binary operations and instead use higher-level APIs, but ABI considerations remain relevant when designing low-latency native bindings.

API – Application Programming Interface

APIs form the contract between the web tool and the engine: endpoints for loading levels, applying parameter overrides, requesting previews, and receiving validation feedback. Designing a stable, versioned API with backward compatibility guarantees prevents editor updates from breaking older builds. Good APIs also provide clear error semantics and validation rules so editors can give immediate, actionable feedback to content authors.

FFI – Foreign Function Interface

FFI enables code written in one language (JavaScript) to call functions in another (C# compiled to WebAssembly) and vice versa. In Unity Web builds this is commonly realized by:

- Exposing specific C# methods to JavaScript through Unity's runtime interop hooks.

- Registering callback functions for events (e.g., asset loaded, validation passed) so the web UI can react to runtime state changes.

FFI must be carefully constrained to avoid allowing arbitrary code execution or state corruption; typically only a thin, well-audited set of functions are exposed.

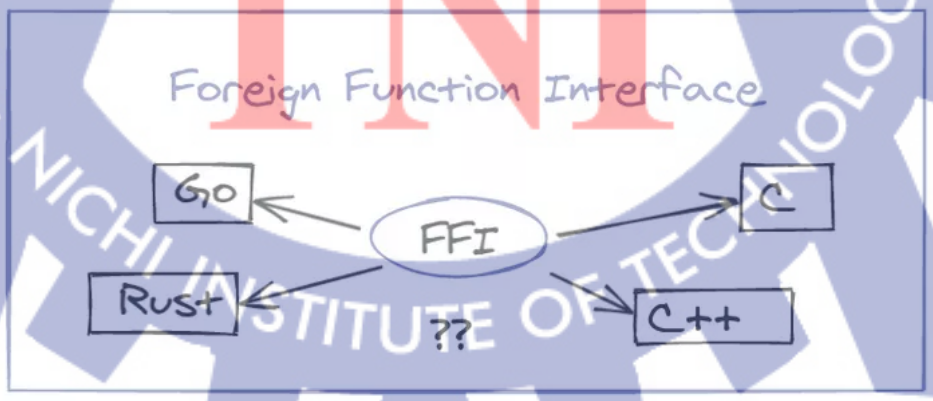

Figure 2.6 Diagram of FFI concept

Combining ABI, API, and FFI considerations yields a robust interoperability strategy: well-defined high-level APIs for routine content operations, and a limited low-level interop surface for performance-critical calls.

JavaScript and ECMAScript

JavaScript is the lingua franca of the web and forms the backbone of browser-based editors[6]. Understanding its capabilities and limitations is essential when designing a web-based content workflow.

JavaScript roles in the workflow

UI logic — Handling user interactions, form validation, and orchestrating editor components.

Data transformation — Converting UI inputs into engine-compatible payloads (e.g., serializing a level layout to a JSON schema).[7]

Interop — Sending and receiving messages to/from the engine runtime (via FFI or WebSocket channels).

Local persistence — Using browser storage or file APIs to maintain temporary sandboxes and caches.

ECMAScript standards

ECMAScript provides the evolving standard for JavaScript[8], introducing features such as modules, async/await, and modern data structures which make codebases more maintainable and expressive. Using recent ECMAScript features improves the clarity of editor code (e.g., async/await for network requests), but teams should consider transpilation or polyfills when supporting older browsers.[9]

Overall, JavaScript enables rapid UI prototyping and continuous deployment cycles that align well with fast content iteration.

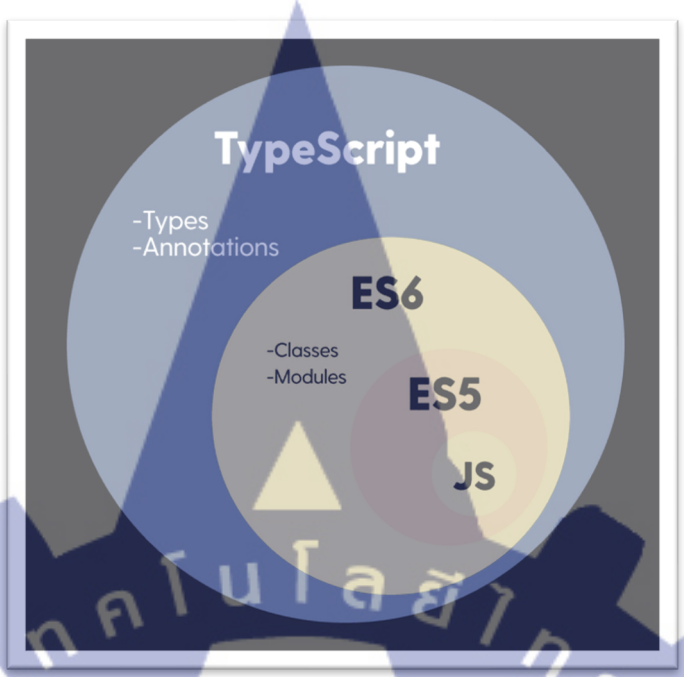

Figure 2.7 Diagram of JavaScript/ECMAScript Standards

WebAssembly

WebAssembly is a low-level bytecode designed for safe, fast execution in the browser. It allows languages like C# (via toolchains) to be executed with near-native performance, which is crucial for running engine logic in a web environment.[10]

Key Capabilities

Performance: Wasm enables computationally intensive subsystems (physics simulations, complex AI) to run efficiently in the browser, allowing closer parity between local native builds and web previews.

Sandboxing: Wasm executes in the browser sandbox, preserving security boundaries. Interactions with the host are mediated via explicit imports/exports, which simplifies security reasoning.

Interop with JavaScript: Wasm modules can export functions and memory buffers accessible to JavaScript, facilitating the FFI layer mentioned earlier.

By leveraging Wasm, Unity Web builds can expose a reliable runtime for previewing content produced in the web editor while keeping heavy lifting within the compiled module.

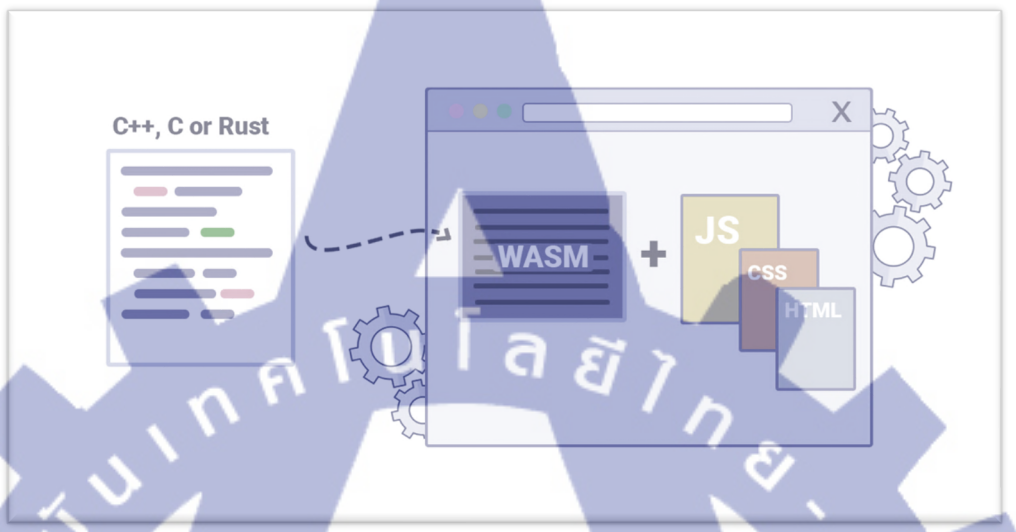

Figure 2.8 Architecture of WebAssembly

WebGL and WebGPU

Graphics APIs determine what can be rendered in a browser preview. WebGL and WebGPU are the two primary technologies for in-browser rendering.

WebGL

WebGL (based on OpenGL ES) is widely supported and allows real-time 2D/3D rendering in the browser without plugins. Unity's WebGL builds translate engine rendering into WebGL calls so a game can run inside a web page. While it provides broad compatibility, some advanced rendering features or shader models may need adaptation.[11]

WebGPU

WebGPU is a modern graphics API that offers improved shader expressiveness, compute shader support, and better multi-threading potential compared to WebGL [12,13]. As browsers adopt WebGPU, web previews can approach native-level graphics fidelity and performance, enabling more accurate visual validation of assets in the editor.

For content tooling, supporting both technologies — falling back to WebGL when WebGPU is unavailable — allows wider compatibility and better fidelity where possible.

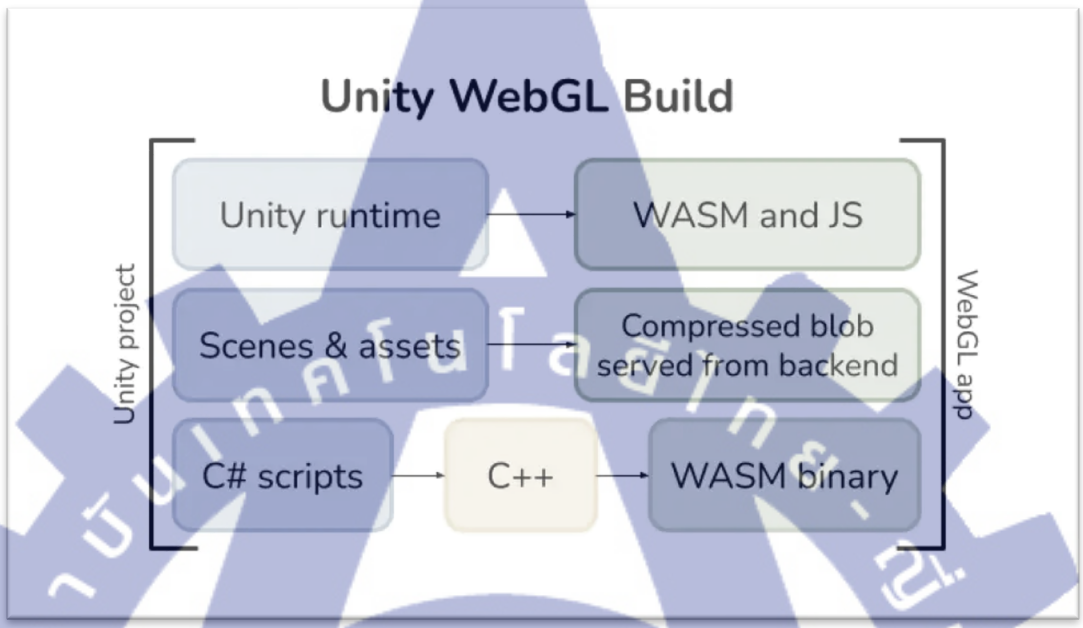

Figure 2.9 Architecture of Unity Web Target

2.3 Literature Review

2.3.1 Related Research on Game Development Tools

Creating Audio Games Online with a Browser-Based Editor [14] This research focuses on the development of a web-based editor for creating game content, specifically targeting audio-based games. Unlike the current study, which aims to develop a generalized workflow for various game genres, this research is narrowly focused on building a unique game system and editor that does not rely on established game engines like Unity or Unreal. The scope is limited to one specific genre, making it less versatile than the proposed approach of using web technologies for broader content creation needs across multiple game types.

Design and Implementation of Game Scenario Editor [15] This study proposes an external content editor for game engines, focusing on a tool for designing narrative-driven or scenario-based games. However, it does not use a web-

based platform and primarily emphasizes the creation and management of XML data for saving and loading game scenarios. The editor is disconnected from real-time game interaction, limiting its utility compared to the proposed workflow, which allows for direct integration with running game instances. This research is valuable for understanding non-web-based external editors but falls short in supporting live, collaborative game development.

Generative Audio and Real-Time Soundtrack Synthesis in Gaming Environments [16] This study proposes a workflow for creating content, but its focus is restricted to music and sound design within gaming environments, specifically for rhythm or music-based games. It does not involve web-based tools or broader content creation workflows applicable to other game genres. While it contributes to understanding how specialized workflows can enhance game content, its lack of focus on web applications or non-technical collaboration limits its relevance to the goals of this research.

User-Generated Content and Editors in Video Games: Survey and Vision [17] This research provides a survey of user-generated content (UGC) tools and editors used in major commercial and open-source video games. The study focuses on how these tools are used in established game titles rather than exploring workflows for teams developing their own games. While it highlights the importance of content editors in the gaming industry, it does not address the need for web-based, accessible workflows that empower non-technical team members, which is a key focus of the current study.

2.3.2 Related Research on Web Technologies in Games

Development of Video Games with WebGL Format That Allows You to Play Video Games Without Need for a Device With High Specifications [18] This research explores the potential of WebGL for creating video games that do not require high-specification hardware. However, the study does not leverage any established game engines like Unity or Unreal, instead opting for simpler, custom-built systems. While it demonstrates the power of WebGL for game deployment in browsers, it does not delve into the collaboration and content creation aspects that the current

study addresses, particularly in the context of non-technical user involvement in game development.

Using Unity and Shield UI for Displaying 3D Medical Education Objects in the Web Browser [19] This study utilizes Unity's WebGL capabilities to display 3D educational content for medical applications, highlighting the potential of web technologies for rendering complex visual data. However, the research is not focused on game development, nor does it aim to enable content creation workflows. The use of WebGL for non-game purposes underscores the flexibility of web technologies but is limited in its application to the collaborative game development context that this thesis seeks to address.

2.3.3 Research Comparison Table

Each of researches in the previous sections can be sorted and compared into the following comparison table. Each one are categorized by the game engine usages, method of game content editing, and types of content being focused.

Table 2.1 Comparison Table of Literature Review

Papers	Game Engine	External Editor	Focusing Content
Creating Audio Games Online with a Browser-Based Editor	<not using>	Web-Based Editor	Audio Game
Design and Implementation of Game Scenario Editor	Not Specified	Not Web-Based	Scenario-based Game
Generative Audio and Real-Time Soundtrack Synthesis in Gaming Environments	Unity	DAW (Digital Audio Workstation)	Audio Game
User-generated content and editors in video games: Survey and vision	Proprietary Game Engines	Proprietary Game Editor	Proprietary Games
Development of Video Games With WebGL Format That Allows You to Play Video Games Without Need for a Device With High Specifications	Pure WebGL	N/A	Game
Using Unity and Shield UI for Displaying 3D Medical Education Objects in the Web Browser	Unity	Web-Based Editor	Medical Education Applications
Our Approach (this thesis)	Unity	Web-Based Editor	Common Genres of Game

2.3.4 Research Gap Identification

Therefore, currently, there is a lack of research that distinctively addresses all three of the following concerns:

A) Platform-specific focus on Unity Game Engine: Some of existing studies adopt a general or platform-agnostic approach, rather than focusing specifically on Unity, which presents unique architecture, and being used by most of the game industry

B) Utilization of web-based capabilities for content editing: While game development tools are widely studied, few investigations explore the integration of web technologies to create web-based editors, enabling content creation, modification, and collaboration directly through a browser interface.

C) Application to game content data: Even in cases where platform-specific tools or web editors exist, those researches did not emphasize their application to game content data—including assets, levels, scripts, and interactive elements—which is one of the hardest to handle kind of data management system

Chapter 3
Research Methodology

3.1 Overall Research Process

Our research methodology involves a comprehensive approach that encompasses the evaluation of existing systems, design and implementation of new workflows, and thorough testing and analysis. This section outlines the key steps undertaken in the research process.

Figure 3.1 Overall Research Process

3.2 Define Evaluation Factors

To rigorously analyze and assess the efficiency and usability of the proposed system, it is essential to first define clear, measurable evaluation factors. These factors serve as benchmarks for comparing the new web-based workflow with traditional CMS approaches, ensuring that the evaluation is objective, systematic, and reproducible. The evaluation is primarily divided into two complementary perspectives: Ease of Use and Collaboration Efficiency, each containing multiple sub-factors for more granular assessment. This dual-perspective evaluation allows a holistic understanding of how the proposed framework impacts both individual user experience and team productivity.

3.2.1 Ease of Use

This factor focuses on assessing the characteristics of the system that directly impact how easily users can access, manipulate, and manage game content without encountering technical obstacles. The evaluation criteria under this category include:

A) **Accessibility**: This criterion examines the differences in access methods between conventional Unity Editor deployment or mobile/desktop installations versus a web-based interface. The web approach allows instant access through a browser, eliminating installation barriers, dependency issues, or version inconsistencies, which significantly improves user onboarding and workflow continuity. Accessibility also includes considerations for cross-platform compatibility, allowing team members to access tools on desktops, laptops, and tablets, thereby reducing workflow fragmentation.

B) **Customization**: Customization evaluates the system's ability to adapt to the unique requirements of each game or content type. Unlike traditional Unity Editor setups, which are often rigid and require technical expertise to modify UI elements or workflows, the web-based approach allows designers and developers to dynamically adjust interface layouts, data fields, and workflow steps according to project-specific needs. This flexibility empowers teams to optimize the system for their workflow rather than being constrained by default editor configurations.

C) **Isolation**: Isolation assesses whether developers and content creators can work independently without interfering with other processes or team members. The web-based system enables the creation of sandbox environments where content can be tested locally or on isolated data sets, reducing the risk of conflicts or accidental corruption of the main project. Independent testing fosters innovation and experimentation while maintaining system integrity.

D) **Data Transfer Efficiency**: This criterion evaluates how efficiently content and assets are exchanged or synchronized between users and servers. Traditional CMS workflows often require bulk uploads to a central server, which can become a bottleneck and slow down iteration cycles. By contrast, the web-based approach allows selective data injection and localized testing, minimizing network overhead and enabling rapid iterative development.

E) **Source Code Protection**: Protecting the intellectual property of the game's core code is a critical concern. Traditional workflows often necessitate giving content creators or artists direct access to the Unity project, increasing the risk of inadvertent modifications or leaks. The web-based workflow isolates content creation from source code access, allowing real-time content editing and testing without exposing sensitive code, which ensures higher security and IP protection.

3.2.2 Collaboration Efficiency

Collaboration efficiency measures the system's effectiveness in reducing redundancy and enhancing coordination among multi-role teams. Efficient collaboration is especially critical for content-driven games where designers, artists, and programmers frequently interact. Sub-criteria for this factor include:

A) **Average Task Duration**: This metric compares the time required for executing specific development tasks, such as testing graphic assets, creating narrative content, or designing levels, under both traditional CMS workflows and the proposed web-based workflow. Reductions in task duration indicate improved system efficiency and smoother team interactions.

B) **Task Weight per Game Genre**: Different game genres rely on various workflows to varying extents. For instance, rhythm games may depend heavily on sound asset testing, while strategy games prioritize level design and data validation. This criterion assigns a weight (1–5) to indicate the relative importance of each task for specific game genres, ensuring that evaluation reflects real-world priorities rather than a uniform, one-size-fits-all metric.

C) **Improvement Ratio**: This metric quantifies the efficiency gains achieved by the web-based workflow compared to conventional methods. It calculates the relative reduction in combined time and manpower required to complete tasks, accounting for both individual efficiency and team-level coordination. This ratio provides a clear, quantifiable measure of the system's practical impact across different game types.

3.3 Establishing Targeting Common Game Types

When designing workflows and systems for content creation, understanding the characteristic requirements of each game genre is crucial. Differences in architecture, content structure, and data management have direct implications for how much the system will affect workflow efficiency and tool usability.

The system is therefore evaluated across several representative game types:

3.3.1 Casual / Puzzle Games

These games typically employ a Client-Authoritative architecture, where the primary processing occurs on the player's device. Game data often consists of level/stage information and basic gameplay configurations. Due to the relatively small data size, content management can be efficiently handled using lightweight formats such as JSON or small-scale databases. The system should allow content creators to operate independently within a local sandbox, facilitating rapid experimentation without dependency on a central server.

Figure 3.2 Example of Casual / Puzzle Game

3.3.2 Turn-based Strategy Games

These games use a Hybrid Client-Server architecture to manage multi-player interactions and maintain consistent game states. Game data includes Master Data (e.g., items, characters) and Content Data (e.g., levels, maps). A web-based content management system must support isolated testing environments, enabling

updates to content or gameplay mechanics without immediately impacting the main server. Once verified, changes can be committed centrally.

Figure 3.3 Example of Turn-based Strategy Game

3.3.3 Music/Rhythm Games

Music and rhythm games also rely on a Client-Authoritative architecture, but require high-fidelity content such as audio tracks, synchronized graphics, and timed events. Real-time preview and testing capabilities are critical for content creators to evaluate the synchronization and quality of assets in a way that mimics actual gameplay.

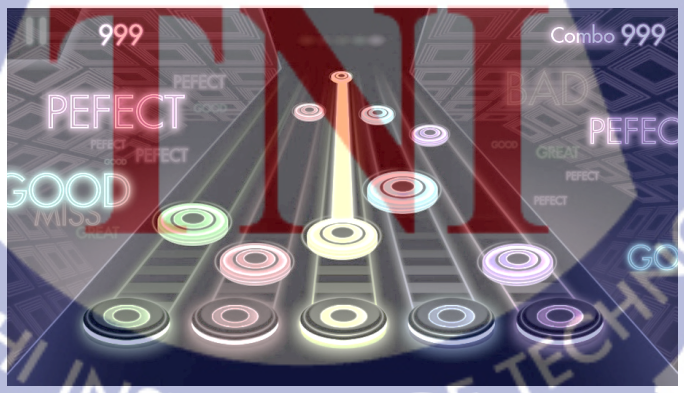

Figure 3.4 Example of Music/Rhythm Game

3.3.4 Real-time MMO Games

MMOs operate under a Server-Authoritative architecture, where all critical gameplay operations are controlled by the server. Given the large and complex data sets, creating and testing new content requires local sandbox environments, as well as mechanisms for temporary server-side data injection to immediately observe the effects of changes without impacting active players.

Figure 3.5 Example of Real-time MMO Game

3.4 Analyze Patterns of Game System Architectures

Different game system architectures impose distinct requirements on development workflows. The proposed system must be flexible enough to support multiple architectural patterns, ensuring it can integrate with a wide range of game types.

3.4.1 Client-Authoritative vs Server-Authoritative Architecture

In client-authoritative architecture, the majority of game logic runs on the client side. This architecture is often used in single-player games or multiplayer games with low-latency requirements. The proposed workflow must ensure that the content created by non-technical users is compatible with client-side systems and does not introduce synchronization issues.

In server-authoritative games, the game logic resides on the server, which ensures fairness and reduces cheating in multiplayer games. For server-

authoritative games, the workflow needs to enable content creation in such a way that it can be easily synced with server-side logic, including maintaining consistency between what non-technical users create and the server's authority over the game state.

3.4.2 Game Data Components and Patterns

Game data is one of the most complex data tables designs. Effective management of game content requires careful classification and structuring of data. Each game genre presents unique data complexity, necessitating the identification of core components and standardized patterns. Properly defined components allow for efficient content creation, testing, and modification. For example, casual games may prioritize simple dynamic content structures, while MMOs require real-time handling of player states and game events. This component and pattern analysis establishes the foundation for scalable, flexible content workflows across diverse game genres.

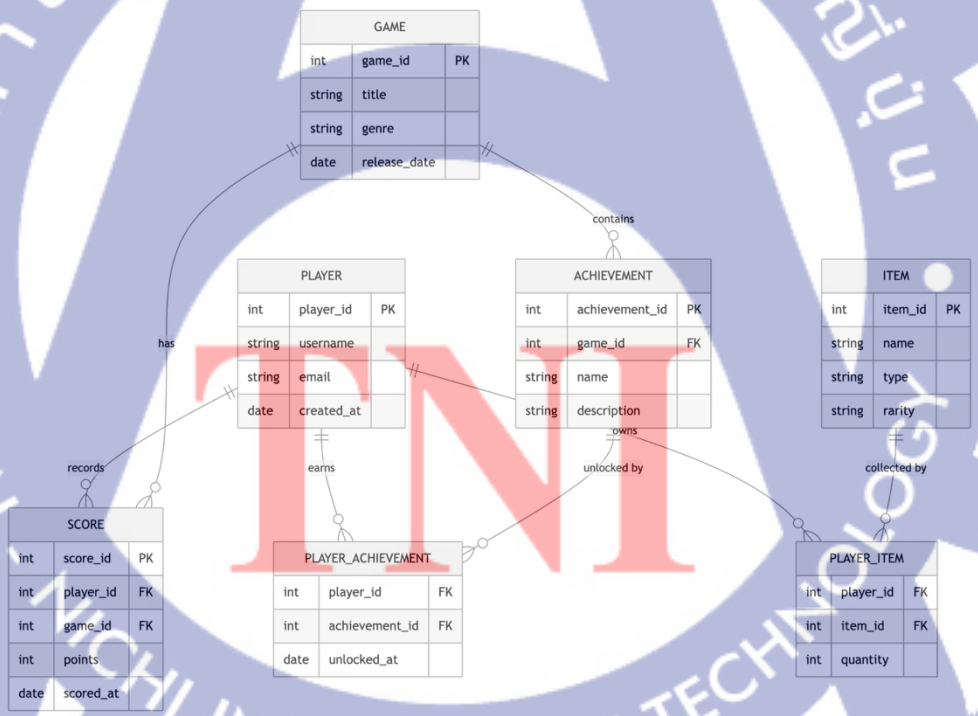

Figure 3.6 Example of Complex Game Data Structure Designing

3.5 Developing the Framework and Workflow

Following the analysis of game architectures and target genres, the next step is developing a framework and workflow compatible with Unity Web builds.

Key techniques we will be utilizing in our workflow include:

3.5.1 Native Function Calls

By enabling direct invocation of Unity functions from a web interface, content creators can test and adjust asset behavior in real-time, including asset loading, game state updates, and gameplay control. This eliminates the need for repeated Unity Editor builds, help create a customizable controlling behaviors, reducing development iteration times.

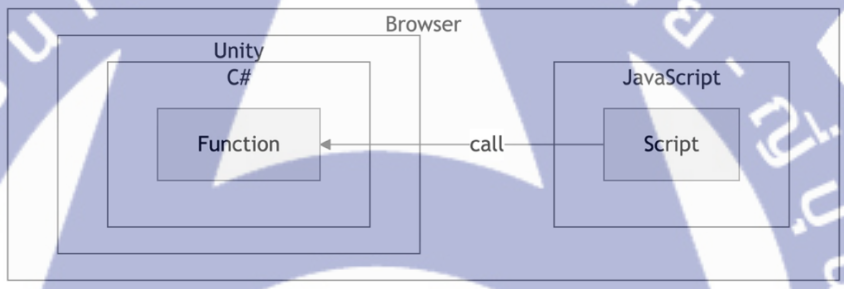

Figure 3.7 Architecture Diagram of Native Function Calls on Unity Web

3.5.2 Web Request Injection

Web Request Injection allows dynamic modification and overriding of game data during runtime via HTTP requests or web forms. Content creators can update levels, items, or narrative elements directly from the web UI, without modifying the underlying communication or system architecture of the game. This method also enable testing of new game art assets in isolation, to ensures seamless integration between web-based editing and live game logic.

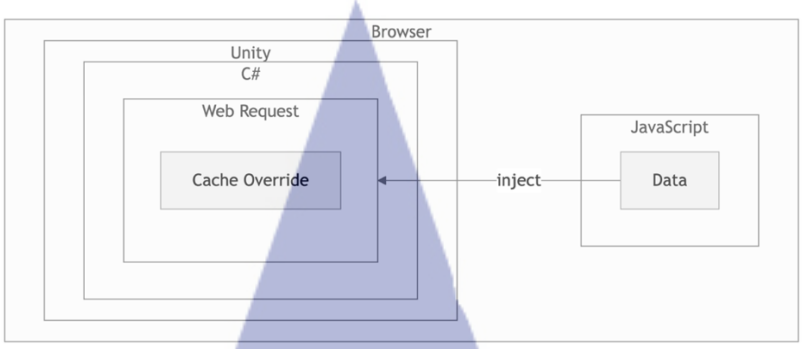

Figure 3.8 Architecture Diagram of Web Request Injection on Unity Web

3.6 Results Evaluation & Analysis

Once the framework and workflow are implemented, a thorough evaluation is conducted to verify their effectiveness. This involves testing the system across all target game types, measuring Ease of Use and Collaboration Efficiency, and comparing performance against traditional CMS-based workflows.

Metrics include task completion times, team workload reduction, and flexibility in isolated testing environments. The results provide empirical evidence of the framework's practical benefits, confirming its ability to streamline content creation, reduce dependencies, and enhance team collaboration.

Chapter 4
Research Results

4.1 Analysis of Common Game Data Architectures

4.1.1 High Level Categorization of Game Data

From the study and survey of different game genres, it was found that the structure of game data and system architectures exhibits distinct patterns as follows:

A) **Master Data / Meta Data**: This includes core game information such as characters, items, abilities, and fundamental parameters. Typically, these are stored separately from level-specific content to allow centralized control, prevent duplication, and enable consistent updates across stages or scenarios.

B) **Content Data (Stages / Levels)**: Level or stage data varies depending on the game type. For Puzzle games, the data structure is relatively simple, often limited to grid layouts, puzzles, and progression rules. For Strategy or MMO games, data is far more complex, including maps, equipment, missions, and multi-player interactions, requiring robust data organization and server-side synchronization.

C) **Player Progression Data**: This tracks player advancement, including scores, accumulated items, and the completion state of levels. The volume of this data grows proportionally with the number of players, necessitating efficient storage, retrieval, and synchronization mechanisms.

D) **Gameplay State / Snapshot**: This refers to the real-time state of the game, such as character positions, active events, and outcomes of player actions. Storing this separately allows fast processing, supports immediate testing, and enables modifications at any point without affecting the main data flow.

4.1.2 Overall Data Architecture

Each of game data categories can be put into logical hierarchy structure as following depiction, Major separation is whether that data only belong to each of player separately, or it is same for the entire game system regardless of player. Further separation into more detailed components is for differences in logic and handling in our proposing workflow.

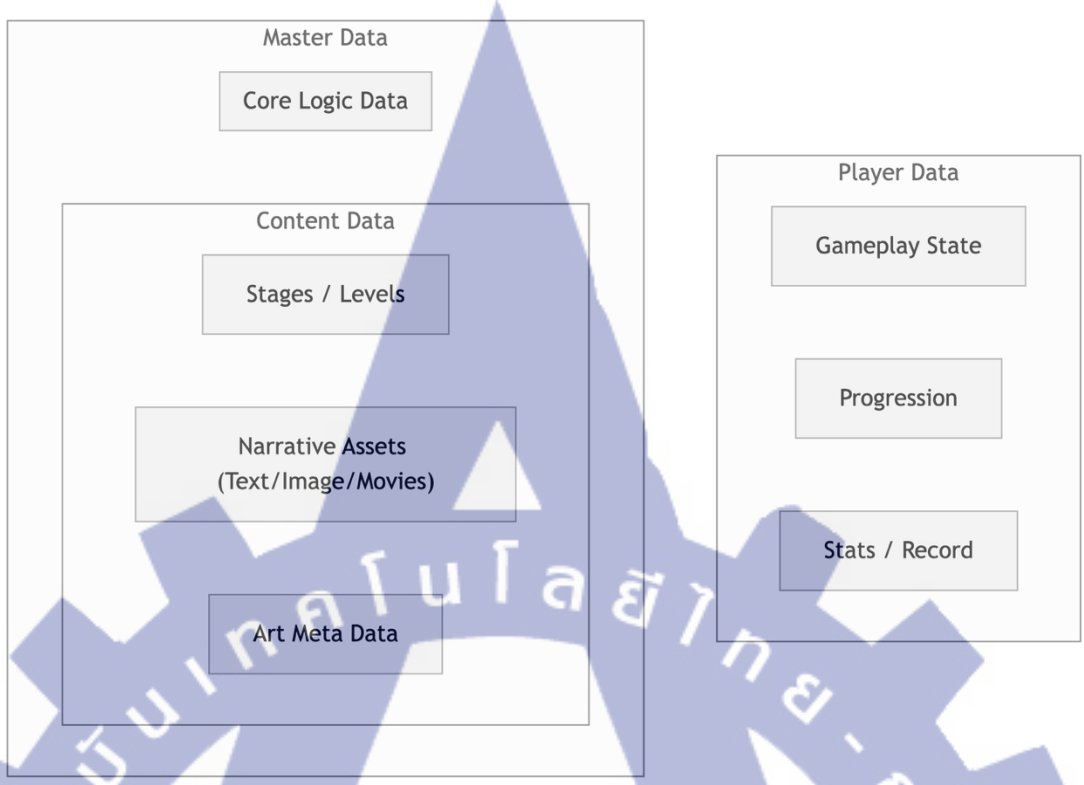

Figure 4.1 Logical Hierarchy of Game Data Architecture

4.2 System and Workflow Development Results

Based on the requirements and the analysis of game data architectures, a system was developed with components in C# and JavaScript working collaboratively. The core mechanism was the creation of a Custom Wrapper Class for Unity WebRequest, enabling diverse control over game operations via HTTP requests. Key features include:

4.2.1 Major Components of the System

The developed system contains the followings components

A) **Unity Wrapper Component Class**: To be able to customize and override the behavior of WebRequest of Unity, we choose an approach to create a wrapper component class that any code in the game system can use throughout the entire project, and get our custom subsystem features

B) **HTTP Request Injection Subsystem**: JavaScript dynamically generates JSON API requests and responses based on adjustments made in the Web UI. These data entries are stored in the internal cache, similar to asset overrides, and can manipulate both outgoing requests and incoming responses. This enables modification of client-server communication without touching the underlying game logic.

C) **CDN Asset Overriding Subsystem**: JavaScript commands direct Unity to load files through a Blob URL. The data is first processed through Unity WebRequest's format conversion and then stored in an Internal Cache. This allows the asset to be preloaded based on the URL before it is actually called during runtime, supporting dynamic testing and rapid iteration.

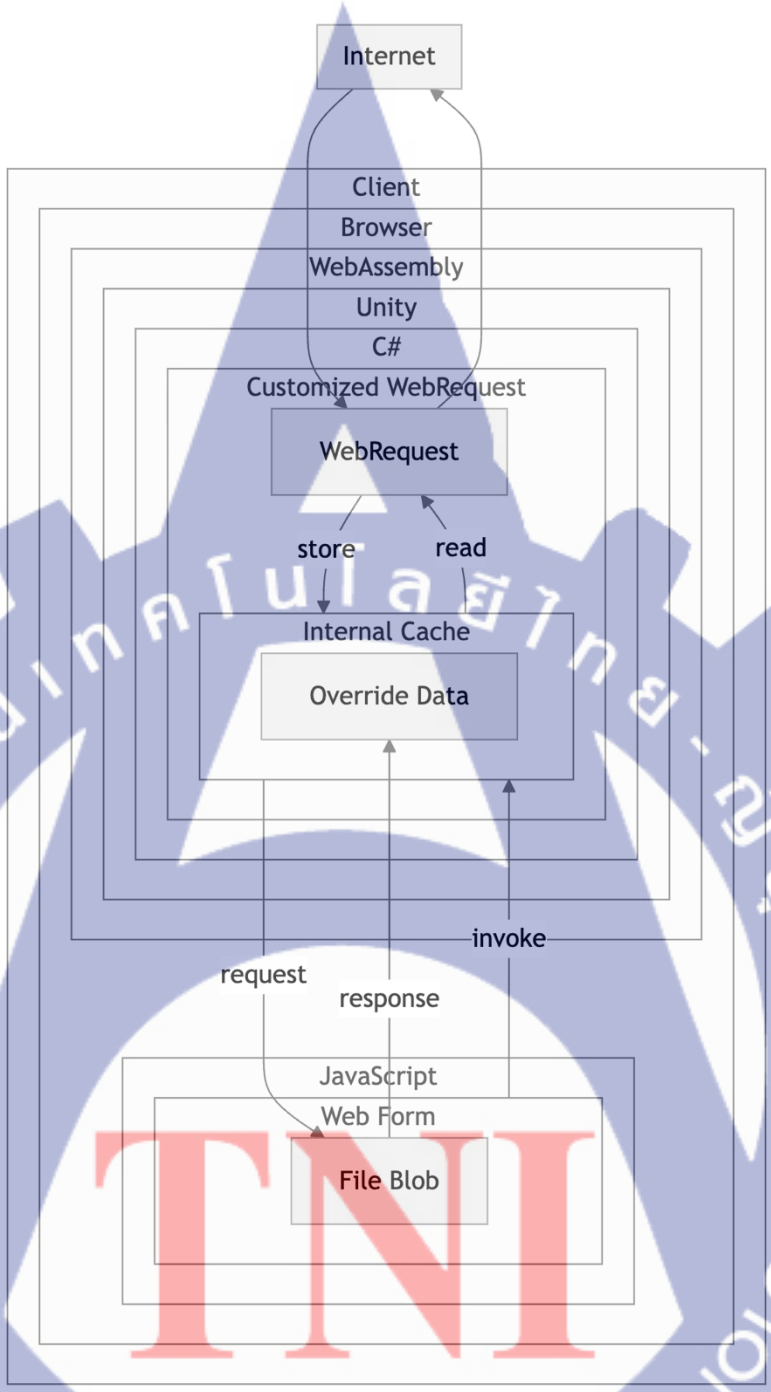

Figure 4.2 Architecture of Developed System

4.2.2 Interoperability of C# and JavaScript

We utilize the ability of Unity Web target that provide language interoperability of C# inside Unity with JavaScript running at web browser level.

Resulting in communication via a function and string parameter. Any text-based data types can be send over plain text via JSON data serialization.

The main FFI function is SendMessage function provided on JavaScript side to side data to Unity game object in scene via 2 parameters, game object's name, script component method name, and string parameter.

```
// Assume UnityInstance is your WebGL build instance
function injectContent(jsonData) {
    // Convert JSON object to string
    const jsonString = JSON.stringify(jsonData);

    // Send the content to Unity runtime via SendMessage
    // GameObject: "WebContentManager", Method: "ApplyContent", Param:
jsonString
    UnityInstance.SendMessage('WebContentManager', 'ApplyContent',
jsonString);
}

// Example usage
const newContent = {
    stageName: "Level_01",
    dialogues: ["Welcome!", "Collect all items to proceed."],
    assetUrls: ["assets/character.png", "assets/item.png"]
};

// Trigger content injection
injectContent(newContent);
```

Figure 4.3 Example of JavaScript Calling to Unity C#

4.2.3 Improved Workflow

The developed system supports various workflows:

A) **Art Asset Testing**: Artists can instantly load and test graphic files or 3D models locally. Master/Meta Data values can be overridden directly through the Web UI, enabling immediate validation without waiting for central server deployment.

B) **Narrative Content Creation**: Designers can create dialogues, story sequences, and test their results in real-time on a local sandbox. The system can also simulate interactions with Player Data, enabling immediate feedback on content effects.

C) **Stage/Level Creation**: The system allows independent editing of levels and stage data, including server-side overrides for instant result verification. For server-calculated data, operations leverage the State Snapshot, simplifying content overrides and testing without affecting live gameplay.

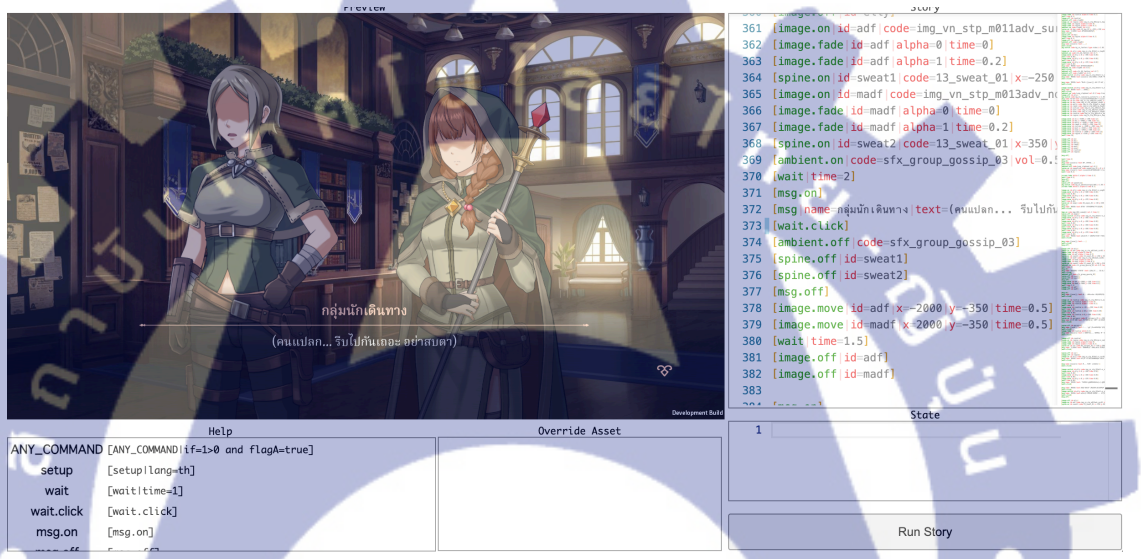

Figure 4.4 Sample UI of Developed System

From Figure 4.4, the developed system user interface comprises of Unity Viewport Canvas, HTML Form for Text Input, File Upload, Input Button. And an adaption of Monaco Editor library for Code Syntax Highlighting & Checking

Compared to Unity's built-in internal systems such as Addressable and Scriptable Objects, our system is more flexible because it operates on standard HTTP web resources. It can function without being confined to Unity's GUI Editor and is not limited to only the pre-build stage of game development.

4.3 Evaluation and Analysis

This section presents quantitative evaluation results for the developed framework, compared against traditional CMS workflows. Metrics were measured in terms of Ease-of-Use and Collaboration Efficiency, summarized in tables with analytical commentary.

4.3.1 Ease of Use Evaluation

We evaluated the ease of use by each factors established in Chapter 3, compared each availability of each factors in the following table.

Table 4.1 Ease-of-Use Comparison: Traditional CMS vs. Web UI

Evaluation Criteria	Traditional CMS	Web UI
Accessibility	Access via Unity Editor or Mobile App; updating and deployment are complex	Browser-based, always using the latest version
Customization	Limited ability to customize UI in Unity Editor	Full, flexible, and convenient UI creation and design
Isolation	Requires central server for testing	Can test independently on local machine
Data Transfer	Must upload data to server for testing	Immediate local testing
Source Code Protection	Cannot protect source code via Unity Editor	Achievable through build version isolation

From Table 4.1, it is evident that the new system satisfies all criteria outlined in Section 3.1.1.

4.3.2 Collaboration Efficiency Evaluation

We then evaluate the collaboration efficiency by taking account of time and number of persons (pax) need to perform each kind of tasks. Data was

collected by having 10 moderate-skilled small development teams perform same simple tasks using both the traditional and new workflows.

Table 4.2 Average Collaboration Time per Task

Task Type	Traditional CMS			Web UI			n	t	p
	PAX	Avg Time $\bar{x}$ (min)	s.d.	PAX	Avg Time $\bar{x}$ (min)	s.d.			
Art Assets	2	58.2	4.2	1	45	2.4	10	8.63	0
Narrative Content	2	119.3	5.6	1	86	4.1	10	15.17	0
Stages / Levels	2	126.1	5.9	1	62	3.2	10	30.20	0

Table 4.2 results indicate a reduction in team size per task from 2 (editor + programmer) to 1, along with shorter completion times due to the elimination of waiting and coordination overhead.

4.3.3 Weighted Improvement Ratio Calculation

Using the time and number of persons involved from previous section, we then calculate the weighted scores by incorporating the weight for each task among each game types

Table 4.3 Task Importance Weight per Game Type

Game Type	Art Assets	Narrative Content	Stages / Levels
Casual / Puzzle	4	1	5
Turn-based Strategy	2	3	5
Music / Rhythm	4	1	5
Real-time MMO	3	3	4

Table 4.3 assigns weight scores (1–5) to indicate the relative importance of each task for each game type as discussed in Section 3.2. These weights reflect the influence of each task on overall workflow efficiency.

The Weighted Improvement Ratio (WIR) was then calculated using:

$$WIR = \sum \left(\frac{T_{baseline} \times P_{baseline}}{T_{proposed} \times P_{proposed}} \times W \right) \quad (1)$$

Where:

- T baseline = Average task time using the traditional CMS
- T proposed = Average task time using the Web-based workflow
- P baseline = Number of personnel involved using the traditional CMS
- P proposed = Number of personnel involved using the Web-based workflow
- W = Task importance weight for each game type

Table 4.4 Weighted Improvement Ratio (WIR) per Game Type

	Weight per Game Type			**WIR**
Game Type	**Art Assets**	**Narrative Content**	**Stages / Levels**	
Casual / Puzzle	40%	10%	50%	3.34x
Turn-based Strategy	20%	30%	50%	3.37x
Music / Rhythm	40%	10%	50%	3.34x
Real-time MMO	30%	30%	40%	3.22x

Table 4.4 demonstrates the weighted improvement in task efficiency when comparing the traditional CMS with the proposed Web-based workflow. This calculation accounts for reduced personnel requirements (halved per task) and shows a significant increase in collaborative efficiency.

The primary gains come from reducing of number of personnel, which then directly affect the idle time and coordination delays, with slight variations across tasks and game genres.

4.4 Overall Results Analysis

This section presents an integrated discussion of the findings from the system evaluation. The analysis focuses on overall efficiency gains, genre-specific results, and key observations about workflow strengths and constraints.

4.4.1 Efficiency Improvements

The primary contributing factor of the significant outcomes was the reduction in required personnel. In the traditional CMS workflow, most content updates demanded close cooperation between a programmer and a designer or artist. Under the new system, a single user could perform the entire process—from data input to testing—directly through the browser interface. This shift effectively cut team size per task in half. It also gave artists and designers more autonomy, reducing idle waiting time and improving morale within multidisciplinary teams.

Time efficiency improved dramatically as well. The data showed average task durations dropping between 35 % and 50 %, depending on the content type. The main factor behind this gain was the removal of slow build cycles and the need for technical intervention before testing new content. With instant runtime injection and local previews, small edits that previously took hours to validate could now be verified in minutes. This acceleration not only shortened iteration loops but also encouraged more creative experimentation, since teams could test ideas without the penalty of long deployment delays.

4.4.2 Results by Game Genre

Casual and puzzle titles gained the most from the proposed system. Their relatively light data structures and client-authoritative logic allowed nearly complete independence from the central server. Level designers could create and adjust puzzles directly through the web UI, testing difficulty and layout instantly.

In strategy games, the benefits were equally clear but manifested differently. Because these games often rely on synchronized data across multiple players, the workflow's isolated testing environments became crucial.

For rhythm games, timing precision was the deciding factor. The ability to preview synchronized audio-visual content in real time allowed creators to

perfect beat alignment and animation cues directly in the browser. Previously, even minor changes required rebuilding the project to test playback synchronization. The new workflow eliminated that barrier completely. Artists could now iterate continuously, checking rhythm timing instantly and achieving a much tighter creative feedback loop.

MMO titles presented the most complex challenge due to their server-authoritative structure. Complete autonomy was not possible because many game states depend on centralized validation. However, the system still enabled substantial improvement through controlled server-side injection for non-critical data—such as event scripts, quest content, and UI configurations.

4.4.3 Key Observations and Discussion

Overall, the workflow proved both flexible and secure. Its browser-based accessibility meant that anyone with permission could work instantly, without installing or maintaining a local Unity environment. The isolation of content editing from source code protected intellectual property while allowing safe experimentation—a combination rarely achieved in collaborative pipelines. Moreover, the framework scaled effectively across genres, confirming its adaptability for both small independent projects and larger online systems.

A few limitations were identified during testing. For large MMO systems, fully synchronized server simulation remained a technical hurdle and required periodic programmer supervision. Web-based previews also depended on hardware performance and network stability, which occasionally affected asset-heavy scenes. Finally, complex Unity Editor features—like advanced shader customization or multi-scene editing—were still beyond the scope of the web interface, though they could be addressed in future iterations.

In practice, the results suggest that a web-based workflow can transform how content teams operate. The approach lowers technical barriers, accelerates production cycles, and enables non-programmers to participate in real-time content creation. For studios pursuing live-service models or frequent updates, this translates into tangible savings in both manpower and turnaround time.

Chapter 5
Conclusion and Future Works

5.1 Conclusion

The research has demonstrated that a web-based workflow integrated with the Unity ecosystem can substantially improve the efficiency and flexibility of game content development. The study identified and analyzed key data architecture patterns across multiple genres, revealing that game data can be effectively separated into four main categories: Master/Meta Data, Content Data, Player Progression Data, and Gameplay State. This modular structure enables independent updates and localized testing, providing a foundation for rapid iteration without compromising system integrity.

Performance evaluation confirmed that the developed system successfully reduced human resource requirements by approximately 50% per task, while maintaining or improving overall quality and coordination. The Weighted Improvement Ratio (WIR) across all tested game genres ranged between 3.2× and 3.4×, demonstrating significant gains in both productivity and workflow fluidity. These improvements stemmed from the integration of runtime content injection, web-based sandboxing, and dynamic request handling—all of which minimized dependencies between programmers and content creators.

Equally important, the results highlight enhanced ease of use and collaboration efficiency. Artists, writers, and designers could test and refine their work immediately through the browser interface, eliminating the need for engine access or technical mediation. The workflow effectively bridged creative and technical domains, turning what were once sequential tasks into concurrent, parallel processes. Overall, the system offers a concrete demonstration of how web technologies can modernize game production pipelines while ensuring data safety and scalability.

5.2 Recommendations

Based on these findings, several practical recommendations are proposed for studios and development teams seeking to implement similar systems.

First, adopt web-based editing tools as part of the standard development pipeline. By integrating these tools directly with the engine runtime, creative personnel can operate independently, reducing communication overhead and allowing faster feedback cycles.

Second, teams should design modular data architectures from the outset. Separating Master Data, Stage Data, and Player Data not only improves scalability but also makes it easier to perform isolated tests and hot-swaps during live operations.

Third, the use of Custom Wrappers and Request Handling systems—as introduced in this study—should be prioritized. These mechanisms allow developers to simulate API calls and server interactions locally, thereby reducing reliance on production servers and enabling realistic testing environments during development.

5.3 Future Works

While the current research has achieved promising results, there are several directions for continued exploration.

Future studies could expand sandbox capabilities to support multiplayer simulations, allowing collaborative and networked content testing under controlled conditions. This would be particularly valuable for MMO and co-op genres that rely on synchronized data validation.

Another avenue involves integrating automated testing tools directly within the web interface. Automating asset validation, script checks, and version control could further shorten iteration cycles and reduce the burden on QA teams.

In addition, the system could benefit from real-time data analytics integration, enabling live monitoring of player progression, content performance, and feedback loops. Such analytics would help developers dynamically adjust balance and difficulty while observing real user behavior.

Finally, further experimentation should explore applications in other game types—such as simulation, action-adventure, or educational games—to verify the generalizability of the workflow and uncover domain-specific adjustments.

5.4 Final Remarks

In summary, this research reaffirms the growing potential of web-based workflows in the field of game development. By reducing both manpower and development time, the proposed system demonstrates a tangible pathway toward more efficient and collaborative production pipelines.

Beyond technical advantages, the workflow encourages a cultural shift within development teams—promoting transparency, independence, and creative empowerment across roles. Programmers, artists, and designers can now contribute on equal footing, supported by tools that align with modern agile principles.

The findings also carry broader implications for the game industry. As games increasingly rely on continuous updates and live content delivery, the adoption of scalable, web-based frameworks can become a decisive factor in maintaining production quality and speed. The approach proposed here represents not just a technical improvement, but a practical step toward more sustainable and collaborative digital content creation in the years ahead.

References

References

[1] L. Etaiwi et al., "Consensus in game engine architectures: An overview of subsystem coupling in game engines," *Empirical Software Engineering*, vol. 31, no. 1, pp. 22, January 2026.

[2] Unity Technologies, "*Unity Documentation,*" Unity Documentation [Online]. Available: https://docs.unity.com/. [Accessed: September 10, 2025].

[3] A. Hussain et al., "Unity game development engine: A technical survey," *University of Sindh Journal of Information and Communication Technology*, vol. 4, no. 2, pp. 73–81, July 2020.

[4] R. Vyas, "Comparative analysis on front-end frameworks for web applications," *International Journal for Research in Applied Science and Engineering Technology*, vol. 10, no. 7, pp. 298–307, July 2022.

[5] Microsoft Corporation, "*Cross-Language Interoperability,*" [Online]. Available: https://msdn.microsoft.com/en-us/subscriptions/50b25433-59f2-4c6f-9774-7d777cde6b0a%28v=vs.90%29. [Accessed: September 10, 2025].

[6] Mozilla Corporation, "*JavaScript | MDN,*" [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript. [Accessed: September 10, 2025].

[7] Microsoft Corporation, "*Monaco Editor,*" [Online]. Available: https://microsoft.github.io/monaco-editor/. [Accessed: September 10, 2025].

[8] Ecma International, "*ECMA-262,*" Ecma International [Online]. Available: https://www.ecma-international.org/publications-and-standards/standards/ecma-262/. [Accessed: September 10, 2025].

[9] Microsoft Corporation, "*JavaScript With Syntax For Types,*" [Online]. Available: https://www.typescriptlang.org/. [Accessed: September 10, 2025].

[10] World Wide Web Consortium, "*Specifications - WebAssembly,*" [Online]. Available: https://webassembly.org/specs/. [Accessed: September 10, 2025].

[11] Khronos Group, "*WebGL 2.0 Specification,*" [Online]. Available: https://registry.khronos.org/webgl/specs/latest/2.0/. [Accessed: September 10, 2025].

[12] A. Aldahir, "*Evaluation of the Performance of WebGPU in a Cluster of WEB-Browsers for Scientific Computing,*" Thesis B.Sc. (Computing Science), Faculty of Science and Technology, Umeå University, Sweden, 2022.

[13] World Wide Web Consortium, "*WebGPU,*" [Online]. Available: https://www.w3.org/TR/webgpu/. [Accessed: September 10, 2025].

[14] M. Urbanek et al., "Creating audio games online with a browser-based editor," *AM'19: Proceedings of the 14th International Audio Mostly Conference: A Journey in Sound*, Nottingham, United Kingdom, September 18, 2019, pp. 272–276.

[15] W.-S. Park et al., "Design and implementation of game scenario editor," *Journal of Korea Game Society*, vol. 10, no. 1, pp. 115-125, February 2010.

[16] C. Bossalini et al., "Generative audio and real-time soundtrack synthesis in gaming environments: An exploration of how dynamically rendered soundtracks can introduce new artistic sound design opportunities and enhance the immersion of interactive audio spaces," *OzCHI '20: 32nd Australian Conference on Human-Computer-Interaction*, Australia, December 2, 2020, pp. 281–292.

[17] H. Duan et al., "User-generated content and editors in video games: Survey and vision," *2022 IEEE conference on games (CoG)*, Beijing, China, August 21-24, 2022, pp. 536–543.

[18] D. Prayudi et al., "Development of video games with WebGL format that allows you to play video games without need for a device with high specifications," *Brilliance: Research of Artificial Intelligence*, vol. 3, no. 2, pp. 117–121, November 2023.

[19] A. Nikolova et al., "Using unity and Shield UI for displaying 3D medical education objects in the web browser," *Digital Presentation and Preservation of Cultural and Scientific Heritage*, vol. 8, pp. 221-224, September 2018.