

UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

BACHELOR IN COMPUTER SCIENCE

Development of a PowerPoint Add-in for Launching and Managing Interactive Quizzes through TapClassroom

Rodrigo Collado Díaz

June, 2026

Development of a PowerPoint Add-in for Launching and Managing Interactive Quizzes through TapClassroom



UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

Department of Technology and Information Systems

BACHELOR IN COMPUTER SCIENCE

SOFTWARE ENGINEERING

Development of a PowerPoint Add-in for Launching and Managing Interactive Quizzes through TapClassroom

Author: Rodrigo Collado Díaz

Advisor: David Vallejo Fernández

Co-advisor: Santiago Sánchez Sobrino

June, 2026

Rodrigo Collado Díaz

Ciudad Real – Spain

E-mail: Rodrigo.Collado@alu.uclm.es
Telephone: 608 81 10 42

© 2026 Rodrigo Collado Díaz

This work is licensed under CC BY-SA 4.0. To view a copy of this license, visit
`https://creativecommons.org/licenses/by-sa/4.0/`

TRIBUNAL:

Presidente:

Vocal:

Secretario:

FECHA DE DEFENSA:

CALIFICACIÓN:

PRESIDENTE	VOCAL	SECRETARIO
Fdo.:	Fdo.:	Fdo.:

Abstract

In recent years, gamification has established itself as a key methodological strategy in the educational field to foster student motivation and engagement. Interactive quiz platforms allow teachers to integrate real-time participation dynamics, evaluating learning continuously and keeping the audience involved.

In this context, Furious Koalas S.L., a spin-off company of the University of Castilla-La Mancha, develops TapClassroom, a web-based gamification platform for interactive sessions. However, its practical usage presented three main limitations: the session control forced the presenter to remain at a desktop computer, it lacked tools for post-session data analysis, and presenters had no way to integrate quizzes directly into presentation slides, meaning all sessions had to be run exclusively from the standalone web browser.

The aim of this work is to design, develop, and validate three new interrelated subsystems that extend the TapClassroom platform: a mobile-first control interface that grants the host complete freedom of movement, a comprehensive reports module for generating post-session metrics and feedback (Web, CSV, and PDF), and an Office Add-in that natively embeds the system directly into Microsoft PowerPoint presentations.

During its development, guided by an agile methodology, complex architectural challenges were addressed. These included real-time state management via WebSockets under a server-authoritative model, and adapting the system to the strict iframe security policies of PowerPoint Online. All of this was achieved while maintaining compatibility with the pre-existing Node.js/Express backend server and PostgreSQL database, without altering the core quiz-state API.

The system was successfully validated through real-device operational runs and automated End-to-End testing, proving its technical robustness and highly responsive real-time control. Ultimately, these extensions not only enrich the user experience and pedagogical analysis but also establish the technological foundation for a comprehensive future redesign of the application.

Resumen

En los últimos años, la gamificación se ha consolidado como una estrategia metodológica clave en el ámbito educativo para fomentar la motivación y el compromiso del alumnado. Las plataformas de cuestionarios interactivos permiten integrar dinámicas de participación en tiempo real, evaluando el aprendizaje de forma continua y manteniendo la atención de la audiencia.

En este contexto, Furious Koalas S.L., una spinoff de la Universidad de Castilla-La Mancha, desarrolla TapClassroom, una plataforma web para la realización de sesiones interactivas. Sin embargo, su uso práctico presentaba tres limitaciones principales: el control de la sesión obligaba al docente a permanecer frente a un ordenador de escritorio, carecía de herramientas para el análisis de datos tras la sesión, y los presentadores no tenían forma de integrar los cuestionarios directamente en sus diapositivas, por lo que toda la sesión debía ejecutarse exclusivamente desde el navegador web.

El presente trabajo tiene como objetivo diseñar, desarrollar y validar tres nuevos subsistemas interrelacionados que extienden la plataforma TapClassroom: una interfaz de control móvil que otorga al presentador total libertad de movimiento, un módulo de informes para la generación de métricas y retroalimentación post-sesión (Web, CSV y PDF), y un complemento (*Office Add-in*) que incrusta el sistema nativamente en las presentaciones de Microsoft PowerPoint.

Durante su desarrollo, guiado por una metodología ágil, se abordaron desafíos arquitectónicos complejos. Estos incluyeron la gestión del estado en tiempo real mediante WebSockets bajo un modelo de servidor autoritativo, y la adaptación del sistema a las estrictas políticas de seguridad *iframe* de PowerPoint Online. Todo ello se logró manteniendo la compatibilidad con el servidor *backend* preexistente (Node.js/Express) y la base de datos (PostgreSQL), sin alterar la API central del estado del cuestionario.

El sistema fue validado con éxito mediante pruebas operativas en dispositivos reales y automatización *End-to-End*, demostrando su robustez técnica y un control en tiempo real altamente responsivo. En definitiva, estas extensiones no solo enriquecen la experiencia de usuario y el análisis pedagógico, sino que sientan las bases tecnológicas para un futuro rediseño integral de la aplicación.

Acknowledgement

Entregar este Trabajo de Fin de Grado es una de esas fechas que uno imagina desde el primer día de carrera, sin saber muy bien todo lo que vendrá antes de llegar a ella. Ahora que ese momento es real, lo primero que siento es gratitud. Porque detrás de estas páginas hay mucho más que horas de trabajo: hay personas que han estado ahí cuando más las he necesitado.

A mi familia, gracias por ser el punto de apoyo constante que nunca ha fallado. No solo durante estos años de universidad, sino a lo largo de toda una vida. Habéis celebrado cada pequeño logro como si fuera enorme, y habéis restado importancia a cada tropiezo para que no me quedara paralizado en él. Sin vuestro cariño y vuestra confianza incondicional, este camino habría sido mucho más duro. Estos logros también son vuestros.

A mis amigos, tanto a los que llevan conmigo de toda la vida como a los que he tenido la suerte de encontrar durante la carrera: gracias por los momentos de desconexión que tanto se agradecen cuando uno lleva demasiado tiempo mirando una pantalla, y gracias también por la confianza que habéis depositado en mí incluso cuando yo dudaba. Esos ratos compartidos han sido un respiro imprescindible.

Por último, quiero dedicar un agradecimiento especial a mis tutores, David y Santiago. Gracias por abrirme las puertas de la empresa y por confiar en mí para llevar este proyecto adelante desde el primer momento. Vuestra forma de guiar, de estar disponibles cuando surgían dudas, y vuestra cercanía ha marcado la diferencia. De esta experiencia me llevo no solo lo aprendido sobre el trabajo en sí, sino también una forma de enfocar los problemas que espero llevarme conmigo mucho tiempo.

A todos vosotros, gracias por haber formado parte de este camino.

Rodrigo

A la familia que se elige y la que no

Contents

Abstract **v**

Resumen **vii**

Acknowledgement **ix**

Contents **xiii**

List of Tables **xvii**

List of Figures **xix**

1 Introduction **1**

1.1 Context 1

1.1.1 Interactive quiz platforms in education 2

1.1.2 Furious Koalas and the TapClassroom baseline 2

1.1.3 Limitations detected in the original workflow 3

1.2 Project proposal 4

1.3 Architectural overview 5

1.3.1 Mobile interface in the system flow 6

1.3.2 Reports module in the system flow 6

1.3.3 PowerPoint add-in in the system flow 7

1.4 Scope of the work 7

1.5 Document structure 8

2 Objectives **9**

2.1 General objective 9

2.2 Specific objectives 9

3 State of the Art **11**

3.1 Gamification in the Classroom 11

3.1.1 Introduction to Gamification 11
3.1.2 Theoretical Foundations of Educational Gamification 12
3.1.3 Game Elements in Learning Environments 13
3.1.4 Benefits and Limitations of Gamification in Education 14
3.1.5 Existing Gamification Platforms and Tools 15
3.1.6 Comparative Analysis of Interactive Quiz Solutions 18
3.1.7 Evidence-Based Selection and Implementation 20
3.2 Web Development Technologies 21
3.2.1 Modern Web Application Architectures 22
3.2.2 Backend Technologies 23
3.2.3 Frontend Technologies 26
3.2.4 Microsoft Office Extension Development 27
3.2.5 Containerisation and Deployment 29
3.2.6 Development Tools and Version Control 31

4 Methodology **33**
4.1 Methodological Approach 33
4.2 Software Development: Agile Practices 34
4.3 Project Management: Scrum-Inspired Framework 34
4.3.1 Roles 34
4.4 Software Development Methodology 35
4.5 Work Planning 35
4.6 Validation Strategy 37
4.7 Development Workflow 37
4.8 Hardware and Software Resources 37
4.8.1 Hardware Resources 37
4.8.2 Operating System 38
4.8.3 Software Resources 38

5 Architecture **39**
5.1 General Overview 39
5.2 Backend 41
5.2.1 Layered architecture and request handling 41
5.2.2 Data model and session isolation 43
5.2.3 Real-time state synchronisation 44
5.3 Frontend 45

5.3.1 Rendering model and view coordination . . . 46
5.3.2 Desktop views and interaction roles . . . 47
5.4 Mobile Interface . . . 48
5.4.1 Serving a dedicated interface per device type . . . 49
5.4.2 Single-page architecture and view management . . . 49
5.4.3 Modular client architecture . . . 50
5.4.4 Real-time communication and clicker bridge . . . 51
5.5 Reports and Feedback System . . . 52
5.5.1 Output formats and the shared data pipeline . . . 53
5.5.2 Separating computation from presentation . . . 54
5.5.3 Web view and parallel data loading . . . 55
5.5.4 Calculated metrics . . . 56
5.6 PowerPoint Add-in Integration . . . 57
5.6.1 Choosing the integration mode . . . 57
5.6.2 Adapting an interface not designed for an iframe . . . 59
5.6.3 PowerPoint Online hardware constraints . . . 61
5.6.4 Deployment and distribution . . . 63

6 Results **65**
6.1 Mobile Control Interface . . . 67
6.1.1 Session linking and live monitoring . . . 68
6.1.2 Live control and session closure . . . 68
6.2 PowerPoint Add-in . . . 68
6.2.1 Embedded execution in the slide canvas . . . 68
6.2.2 In-slide control and cross-host delivery . . . 69
6.3 Reports and Feedback Module . . . 71
6.3.1 Participant analytics and presenter notes . . . 71
6.3.2 Evidence export . . . 72
6.4 Testing, Verification, and Project Assessment . . . 72
6.4.1 Functional and end-to-end verification . . . 72
6.4.2 Control responsiveness and latency measurements . . . 75
6.4.3 Code Statistics . . . 76
6.4.4 Project Costs and Resources . . . 76

7 Conclusions **77**
7.1 Reached objectives . . . 77

7.2 Addressed competences 78
7.3 Personal conclusion 79
7.4 Future work 80

A Appendix A **83**
A.1 User Stories 83

B Appendix B **87**
B.1 Evolution through iterations 87

C Appendix C **93**
C.1 Deployment Instructions 93
C.2 User Manual 95

References **101**

List of Tables

3.1 Feature comparison between main educational gamification platforms. . . . 19

5.1 Metrics computed by the reports module, grouped by reporting scope. All metrics are derived from the same query and reused across the CSV, PDF, and web view outputs. 56

6.1 Simplified compatibility matrix for final host delimitation. 70

6.2 Condensed evidence matrix for mobile-flow checks under real-device operation. 73

6.3 Browser-side RTT per control action in the automated E2E walkthrough on the local TapClassroom deployment. 76

6.4 Lines of code added per language, measured with `cloc`. 76

B.1 Summary of results – Iteration 1 87

B.2 Summary of results – Iteration 2 88

B.3 Summary of results – Iteration 3 88

B.4 Summary of results – Iteration 4 89

B.5 Summary of results – Iteration 5 90

B.6 Summary of results – Iteration 6 90

B.7 Summary of results – Iteration 7 91

B.8 Summary of results – Iteration 8 91

B.9 Summary of results – Iteration 9 92

List of Figures

1.1 High-level information flow of the extended TapClassroom ecosystem. Elements highlighted in blue correspond to subsystems developed or adapted as part of this project. 6

3.1 Gamification strategies and elements in education. 12

3.2 Gamification key features. 14

3.3 Kahoot! editing interface for a quiz session. 16

3.4 Socrative teacher desktop interface. 17

3.5 Socrative Space Race mode. 17

3.6 Mentimeter templates and interface. 18

3.7 Abstract representation of the client-Server Model. 22

3.8 How Socket.IO works with several clients. 23

3.9 A visual example of a PostgreSQL data model. 25

3.10 Add-in Overview. 28

3.11 A PowerPoint Add-in created by Kurious Koalas as a test. 29

4.1 Sprint-based work plan for the project. 36

5.1 High-level component diagram of TapClassroom. The three subsystems developed in this project (mobile interface, reports module, and PowerPoint add-in) that are highlighted in blue, extend the platform as independent clients of the existing server infrastructure, without modifying the core API modules or the data layer beneath them. 40

5.2 Layered architecture of the TapClassroom server. Each incoming HTTP request passes through the middleware pipeline before reaching the corresponding controller, which delegates to a service that operates on the database through MikroORM. The blue modules are the contributions of this project. 42

5.3 Sequence diagram of a real-time state update. The host's action triggers a `PATCH` request; the server updates its in-memory state and broadcasts the new state to all connected clients, which re-render their views accordingly. . 44

5.4 Server-side rendering pipeline for TapClassroom views. The server assembles data and renders the full HTML before delivery; the embedded client script then subscribes to Socket.IO events and patches the DOM in place, so the server remains the authority for both the initial render and all subsequent state changes. 46

5.5 Interaction roles of the desktop views. The host control panel is the main administrative bidirectional view; the participant view submits answers via WebSocket and reacts to state updates; the stream view subscribes to a dedicated Socket.IO room; the ranking view is rendered server-side on demand, with no real-time dependency. 47

5.6 Initial mockups for the mobile interface of TapClassroom. The selection view (left) lets the host choose an event and open registration; the control view (right) provides full session control during an active quiz. 48

5.7 Navigation between the five views of the mobile interface. The operational views coexist in a single HTML document; in-place transitions are managed by CSS class toggling so the WebSocket connection remains alive throughout the active session. 50

5.8 ES6 module structure of the mobile interface. `ContestApp` is the root orchestrator; all modules share a single `state` object and a common `elements` map. `ContestSocket` is the main consumer of external real-time events, while write requests are distributed across the UI modules that own each part of the control flow. 51

5.9 Internal data flow of the reports module. The computation layer queries the database and builds statistics; a view mapper transforms them into a presentation-ready model consumed by the PDF and CSV renderers; a controller streams the final response. 53

5.10 First design for the Reports web view, developed during early stages. 55

5.11 Early prototype using the task pane model. The interface appears as a side panel in edit mode, but PowerPoint hides it during presentation, making session control inaccessible. This motivated the switch to the content add-in model. 58

5.12 Content add-in execution model. Office reads the manifest, injects Office.js into the iframe, and loads the `SourceLocation` from the TapClassroom server. The returned HTML page calls `Office.onReady()` to confirm host context and activate add-in behaviour. 58

5.13 Participant management view opening as a floating browser window over PowerPoint. Because the add-in runs inside an iframe, navigation calls escape the embedded context and produce a detached window the host must manage separately. 59

5.14 Layout and scaling issues inside the add-in iframe before CSS corrections. Fixed pixel dimensions cause elements to overflow and content to be cut off within the constrained iframe, resolved by replacing fixed sizes with relative units and flexible layouts. 60

5.15 Mentimeter, a Microsoft-validated content add-in with similar functionality to TapClassroom, explicitly warns users that presentation mode is unavailable in PowerPoint Online due to hardware API restrictions imposed by the platform's iframe permission policy. 62

5.16 Sideloading the TapClassroom add-in Online. Using the *Upload My Add-in* option. 64

5.17 Sideloading the TapClassroom add-in for PowerPoint desktop. The manifest XML file is loaded directly into PowerPoint without going through the Microsoft AppSource store, using the *Shared Folder* trusted catalogue tab in PowerPoint Desktop. 64

6.1 High-level overview of the TapClassroom platform. The host controls sessions via the PowerPoint Add-in, web interface, or mobile control interface; the platform core manages real-time state, persistence, and analytics; students participate through mobile or web clients. 65

6.2 Mobile interface checkpoints: selection and monitoring (top-left), active question control (top-right), advanced settings (bottom-left), and session closure (bottom-right). 67

6.3 Add-in loaded in PowerPoint Desktop. 69

6.4 Add-in loaded in PowerPoint Online. 69

6.5 Desktop slideshow with the in-slide control panel and integrated action buttons available in the same presentation surface. 70

6.6 Participant performance report with scores and accuracy rates. For example, Rodrigo answered 56% of questions correctly (5/9), earning 42.0 pts. 71

6.7 Presenter notes panel with automatically generated alert about question difficulty. 71

6.8 Generated PDF report page with per-question analysis and response-distribution chart. 72

6.9 Synchronisation trace: mobile control action, PATCH payload inspection, and stream view rendered in question mode. 74

6.10 RTT (Round-Trip Time) distribution extracted from the Fetch API instrumentation acting as the state controller from the actual host device (iPhone 14). The 95th percentile (P95) boundary is highlighted. 75

C.1 Content Add-in embedded directly within a PowerPoint slide showing the live streaming view of the contest 96

C.2 Embedded Add-in displaying the clean stream view and the minimalist floating control panel 96

C.3 Mobile remote control interface displaying live session metrics and polling commands 97

C.4 TapClassroom dashboard highlighting the button to access the reports section 98

C.5 Reports interface displaying event selection, expandable analysis panels, and export options 99

Chapter 1

Introduction

In this chapter, the general context and motivation behind the project are presented, followed by the limitations detected in the original TapClassroom platform, the proposed subsystems to extend, a high-level architectural view, and the structure of this document.

1.1 Context

Question-and-answer contests have maintained a strong presence in popular culture, particularly on television, where they combine suspense, time pressure, and audience participation. Gameshows like *Who Wants to Be a Millionaire?* and *Pasapalabra* have enjoyed longevity because they combine factual knowledge with spectacle, individual competition with collective viewing, and entertainment with a sense of shared cultural values. Their appeal does not depend only on the questions themselves, but also on the rhythm of the interaction, the visibility of progress, and the suspense generated by limited time.

This format is especially relevant in educational contexts because it transforms assessment into an active experience. In a traditional lecture, the teacher often receives limited information about whether students are following the explanation until a formal exercise or exam is performed. In a quiz-based session, however, each answer becomes an immediate signal. The teacher can detect misconceptions, reinforce concepts, and adjust the pace of the class while students remain involved in the activity.

In recent years, educational paradigms have increasingly embraced active learning methods to foster student motivation [OSBSW20]. The use of game mechanics in non-game contexts, an approach generally known as gamification [Kap12], has become a relevant methodological strategy. By introducing elements typical of games into academic environments, educators can transform traditional lectures into interactive experiences that reward participation and provide immediate feedback. This is particularly useful when the goal is to keep a group attentive, identify weak points, and encourage students to test their understanding without the formality of a conventional assessment.

Gamification is effective when it is integrated into the learning activity with a clear pedagogical intention. Points, rankings, countdowns, and progression indicators can increase engagement, but their value is greater when they are connected to meaningful feedback. In live quizzes, students see that their answers matter, teachers receive information about the state of the group, and the class as a whole shares a common rhythm.

1.1.1 Interactive quiz platforms in education

The spread of interactive electronics and mobile devices has naturally extended gamification to the digital realm. This digitalisation of entertainment and education has led to the widespread adoption of quiz-based platforms such as *Kahoot!*, *Socrative*, *Quizizz*, and *Mentimeter*. These tools leverage competition, time pressure, and progression systems to keep students engaged while reducing the technical barrier to participation.

In classroom environments, quiz-based gamification plays a dual role. It provides the instructor with an immediate mechanism to evaluate comprehension and adjust the pacing of the lecture, while simultaneously keeping the audience attentive. However, empirical evidence suggests a complexity to these immediate outcomes; while the integration of gamification elements like points, leaderboards, and badges can boost motivation and perceived competence in the short term, this drive is frequently extrinsic and tends to decline as the novelty effect wears off [RM23]. To foster more sustainable engagement, the educational focus often shifts towards the broader ecosystem of delivery. Within this context, *mobile learning*, characterised by its spontaneous, informal and situated nature, suggests a shift away from rigid, institutionalised teaching formats towards more flexible and personalised learning experiences [Tra05].

In alignment with this shift, interactive quiz platforms have altered the expectations of teachers and students. A modern classroom participation system is expected to operate in real time, support different device types, provide a simple entry mechanism, and produce usable feedback after the activity. It must also fit the presenter's workflow. In face-to-face contexts, interruptions, window switching, or configuration steps can quickly damage the continuity of the explanation.

1.1.2 Furious Koalas and the TapClassroom baseline

In this context, Furious Koalas S.L., a spin-off company of the University of Castilla-La Mancha, developed its own infrastructure for real-time quiz-based contests with live production and audience interaction. The company designs customised interactive experiences based on popular TV shows and deploys them in corporate settings, conventions, trade fairs, and educational activities [1].

[1] https://www.furiouskoalas.com/actividades-para-empresas/

Building upon the success of these productions, the company developed **TapClassroom**[2]. TapClassroom is a gamification platform designed to bring that same high-engagement dynamic into educational and corporate classrooms. It allows presenters to create events, manage audiences, and project real-time feedback. Participants can join via their own mobile devices or by using proprietary physical response clickers connected to the host's machine. The use of such clicker-based response systems has been repeatedly linked to enhanced classroom interactivity [WS20].

From a technical perspective, TapClassroom is a web platform centred on a server-based quiz state. The host controls the progress of the session, participants submit answers, and projected views react to the state broadcast by the server. This design is appropriate for live contests because all views depend on a shared source of truth.

Before this project, the baseline platform already supported the essential lifecycle of an interactive contest. Hosts could create events, configure questions, register participants, start a session, collect answers, display results, and show rankings. It also supported different participant modalities, including web users and physical clickers. This foundation was robust enough to run live activities, but its interaction model was still strongly tied to a desktop browser and to the operational habits of the original web application.

1.1.3 Limitations detected in the original workflow

Prior to the developments introduced in this project, TapClassroom performed solidly but presented three significant usability and operational constraints for the presenter.

The first limitation was related to mobility. The host's control panel was built exclusively for desktop browsers, so the teacher had to remain at the classroom computer to advance questions, open or close the poll, monitor participants, and trigger visual transitions. This reduced mobility and made the session dependent on the physical location of the workstation.

The second limitation concerned the pedagogical lifecycle of the quiz. Although the platform collected answers and computed scores in real time, there was no native mechanism to analyse the session data once the event concluded. The information remained stored in the database, but it was not transformed into a form that a teacher could interpret, export, or use to detect poorly understood concepts.

The third and most disruptive limitation involved presentation flow. In standard teaching environments, instructors present material through slideshow software like Microsoft PowerPoint. Because TapClassroom operated only as a standalone web application, the presenter had to interrupt the lecture, leave presentation mode, open a browser, and control the session. This context-switching disrupted class rhythm and increased mistakes under pressure.

[2] `https://www.tapclassroom.com/`

These limitations are closely related. They do not describe isolated missing features, but a broader problem of integration between the live teaching situation and the software environment. A classroom quiz platform must be technically synchronised, but it must also fit the presenter's physical and cognitive workflow.

This distinction is important because live classroom tools are used under conditions that differ from ordinary administrative software. The presenter cannot pause the audience while recovering a lost window, searching for a report, or returning to a control panel hidden behind the slideshow. Each interruption affects attention, reduces confidence in the activity, and may discourage teachers from adopting the tool regularly. Therefore, improving TapClassroom required more than adding isolated functions. It required reducing friction before, during, and after the quiz session. The subsystems developed in this project respond to that need by connecting live control, later analysis, and presentation delivery into the same operational ecosystem.

From the company's perspective, this also makes the platform easier to deploy in varied classrooms, where available hardware, teacher habits, and presentation routines are rarely identical.

1.2 Project proposal

This Final Degree Project was carried out within the framework of an internship at Furious Koalas S.L. It proposes the design, development, and validation of three new interrelated subsystems that extend the TapClassroom ecosystem, strictly targeting the usability and operational limitations previously described.

The central objective is to provide a cohesive suite of tools that gives presenters complete control over the live session, delivers actionable post-event feedback, and integrates the platform into standard presentation workflows without altering the existing core functionality. The project therefore extends TapClassroom through new interfaces, services, and integration mechanisms that reuse the existing server, data model, and real-time synchronisation infrastructure.

The first contribution is a mobile control interface. By providing a responsive single-page interface tailored for smartphones, the host can monitor the audience, launch questions, open and close polls, trigger answer reveals, and control the pace of the session with freedom of movement around the classroom.

The second contribution is a comprehensive reports module. This system aggregates session data and exposes it through summary views, allowing teachers to analyse student performance and question difficulty. Results can be exported as CSV files or PDF reports. The module converts raw answer data into participant-level summaries, question-level metrics, response distributions, response times, success rates, and presenter notes.

The third contribution is a content add-in for Microsoft PowerPoint. This integration embeds TapClassroom into the presentation canvas, eliminating the need to interact with external browsers during delivery. The contribution required the creation of the Office Add-in manifest and the adaptation of the existing web application to the iframe-based execution environment imposed by Office.

Although the three contributions address different user-facing problems, they share a common design principle. Each one integrates with the existing TapClassroom platform instead of creating a parallel system. The mobile interface uses the same control endpoints as the desktop interface, the reports module reads the same persisted entities, and the PowerPoint add-in loads the existing web views inside the Office host. This approach reduces duplication and preserves the stability of the pre-existing product.

The expected impact is therefore threefold. The presenter gains more flexible control during the live session, the teacher obtains post-session evidence for pedagogical review, and Furious Koalas S.L. validates that TapClassroom can evolve through compatible extensions while maintaining a coherent technical foundation.

1.3 Architectural overview

The TapClassroom subsystems developed during this project are built upon a server-based architecture that separates presentation, application logic, real-time synchronisation, and data persistence responsibilities. This allows the new subsystems to interact with the pre-existing logic without fragmenting the system's core stability.

The core backend operates on Node.js and Express, using PostgreSQL for persistence and Socket.IO for real-time bidirectional communication. The server acts as the single source of truth for the event's progress. Whenever the host issues a command from the desktop browser, the mobile interface, or the embedded PowerPoint canvas, the server updates its state and pushes the synchronised result to participants and projected views.

The three new modules integrate into this workflow in different ways. The mobile interface acts as a new control client over HTTP and WebSockets. The reports module computes analytics from historical session records. The Office Add-in bridges the Microsoft Office.js constraints with the underlying web framework, allowing the same TapClassroom web application to be executed inside PowerPoint.

Figure 1.1 summarises the high-level flow of information across the system. Solid arrows represent request-driven actions, while dashed arrows represent real-time broadcasts.

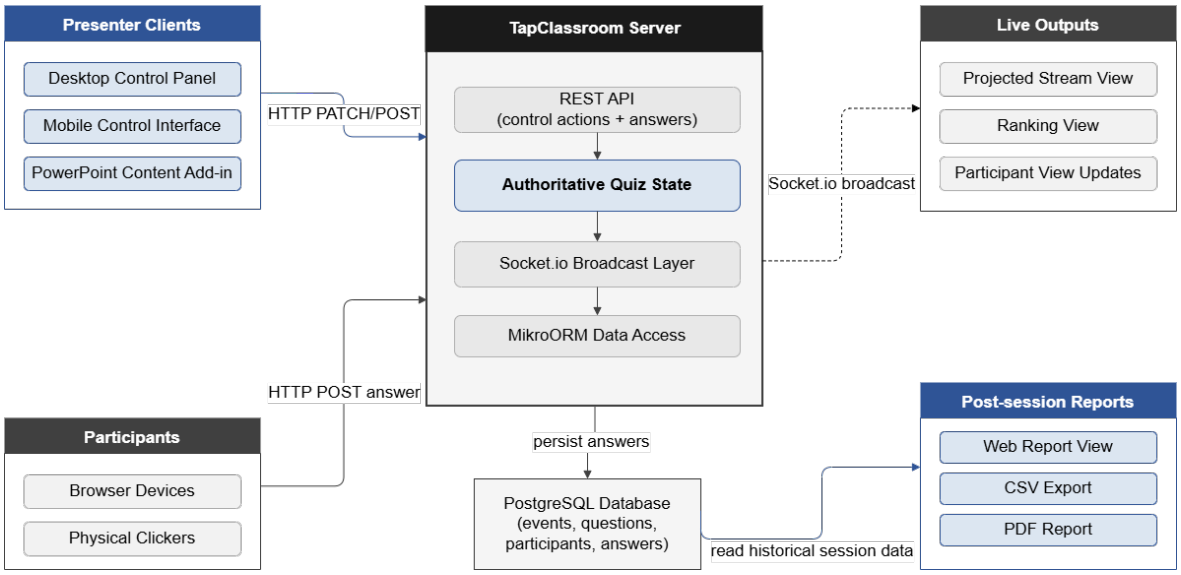


Figure 1.1: High-level information flow of the extended TapClassroom ecosystem. Elements highlighted in blue correspond to subsystems developed or adapted as part of this project.

This view shows how the project contributions fit into the same operational loop. The presenter controls the session, participants answer from their devices or clickers, and the server coordinates the state, persists the results, and feeds both live views and post-session reports.

1.3.1 Mobile interface in the system flow

The mobile interface is introduced as an additional host view, not as a separate backend. It receives the same quiz state that feeds the desktop control panel and sends commands to the same state-management endpoints. Advancing to the next question, opening a poll, applying a multiplier, or displaying a ranking therefore has the same server-side effect regardless of the client used.

This contribution also required adapting the interaction model to touch screens. The result is a mobile-first control surface with larger targets, clear separation between frequent and advanced actions, predictable modals, and a layout that remains readable in portrait orientation.

1.3.2 Reports module in the system flow

The reports module occupies a different point in the lifecycle. It is not involved in the real-time progression of the quiz, but it relies on the data generated by that progression. After the event, it loads stored answers and derives higher-level metrics from them.

The module serves human-readable and machine-readable needs. The web view lets teachers inspect participants and presenter notes, the PDF provides a printable document, and the CSV export supports external analysis. All formats are generated from the same pipeline.

1.3.3 PowerPoint add-in in the system flow

The PowerPoint add-in addresses the most complex integration challenge because it places TapClassroom inside a host application controlled by Microsoft Office. Office Add-ins are web applications embedded in an iframe and described through an XML manifest. This means that TapClassroom can be loaded inside PowerPoint, but must respect Office restrictions involving navigation, authentication, iframe permissions, content security policies, and presentation-mode behaviour.

The solution uses a content add-in because this type of add-in is inserted directly onto the slide surface. A task pane was not appropriate, since task panes are not visible in the same way during slideshow delivery. The content add-in keeps the quiz view and the control experience inside the presentation frame.

Several adaptations were necessary in this embedded context. Navigation actions that previously opened new browser windows had to be redirected inside the add-in frame, selected elements had to be conditionally shown or hidden, and security headers had to be adjusted so that TapClassroom pages could be framed by Office domains. PowerPoint Online also imposed restrictions on hardware APIs, affecting physical clicker communication in the browser-based version.

1.4 Scope of the work

The scope of this project is limited to the extension and integration of the existing TapClassroom platform. It does not attempt to redesign the complete product, replace the core quiz engine, or introduce a new data model. The work focuses on three concrete product surfaces and their supporting backend functionality.

The mobile contribution covers the presenter-side control flow for live sessions, including event selection, participant monitoring, poll control, question navigation, advanced settings, and session termination or restart actions. It does not replace the participant-facing mobile view.

The reports contribution covers post-session analysis and export through summary statistics, participant breakdowns, question metrics, presenter notes, CSV files, and PDF documents. It does not attempt to build a full learning analytics platform.

The PowerPoint contribution covers the creation of a content add-in and the adaptations required for TapClassroom to operate inside PowerPoint Desktop and PowerPoint Online as far as the host environment allows. It does not include publication through Microsoft AppSource.

1.5 Document structure

The remainder of this document is structured in accordance with the standards for Final Degree Projects of the *School of Computer Science* at the *University of Castilla-La Mancha* (UCLM).

Chapter 2: Objectives

This chapter defines the primary and specific objectives of the project, establishing the goals for the new TapClassroom subsystems, including the mobile interface, reports module, and presentation add-in.

Chapter 3: State of the Art

This chapter provides a review of the theoretical foundations of educational gamification, its benefits and limitations, existing interactive quiz platforms, and the web technologies and technical patterns underlying modern real-time applications.

Chapter 4: Methodology

This chapter explains the Scrum-inspired agile framework adopted during development, the continuous validation strategies, the work plan followed across the project, and the hardware and software resources used.

Chapter 5: Architecture

This chapter provides a detailed overview of the system's architecture. It explains the reasoning behind the key design decisions, the resolution of integration challenges, and the internal structure of the pre-existing system alongside the mobile interface, reports module, and PowerPoint add-in.

Chapter 6: Results

This chapter presents the outcomes of the development, evaluating the system's operational continuity under real-world scenarios and discussing the performance impact of the integrations.

Chapter 7: Conclusions

The final chapter assesses the fulfilment of the objectives, outlines the competencies addressed, provides a personal reflection, and suggests directions for future refinements and application iterations.

Chapter 2

Objectives

In this chapter, the objectives of the project are outlined. Specifically, the main objective is explained, followed by the specific sub-goals that support it.

2.1 General objective

This project aims to design, develop, and validate three new interrelated subsystems that extend the TapClassroom platform, a web-based gamification platform for interactive sessions developed by Furious Koalas S.L. The primary goal is to provide a cohesive suite of tools that solves existing operational constraints without altering the core functionality of the platform.

The main advantage of achieving this objective is that TapClassroom becomes better aligned with the complete lifecycle of a classroom quiz. Presenters gain control over live sessions from any location in the classroom, teachers receive actionable post-event pedagogical feedback, and the interactive experience can be embedded into standard presentation workflows, specifically Microsoft PowerPoint. In this way, the project seeks to reduce operational friction before, during, and after the session while preserving the existing server-based architecture and the real-time behaviour of the platform. This is especially relevant in live educational contexts, where the presenter must keep the rhythm of the explanation while simultaneously managing questions, participants, projected content, and feedback.

This objective is not limited to adding independent features. It also requires ensuring that the new capabilities behave as a coherent extension of the existing ecosystem. The resulting system must remain understandable for presenters, technically compatible with the current backend, and useful in realistic educational and corporate presentation contexts.

2.2 Specific objectives

To fulfil the general objective, the work is structured around a set of specific goals derived from the usability, feedback, integration, compatibility, and validation needs identified in the original TapClassroom workflow. These objectives define what the project must achieve from the perspective of the platform and its users:

- **To facilitate system usability and mobility for presenters**. The system must free the host from the main classroom computer. The presenter should be able to manage questions, monitor participants, open and close polls, and control the pacing of the session with freedom of movement. This objective seeks to make TapClassroom compatible with more dynamic teaching situations, where the teacher may need to move around the classroom while keeping control of the activity and maintaining direct interaction with students.
- **To provide actionable post-session feedback**. The platform must offer native mechanisms to analyse session data once the event concludes. The objective is to transform the answers collected during the quiz into useful information for teachers, including performance metrics, participant summaries, and question-level indicators. This should help identify both individual difficulties and questions that may require further explanation in future sessions. The results must also be exportable in structured formats (CSV and PDF) for academic grading, administrative records, and pedagogical review.
- **To integrate seamlessly with presentation flows**. The interactive quizzes must run directly on the presentation layer itself to prevent disruptive context-switching. The objective is to allow TapClassroom to operate inside Microsoft PowerPoint so that presenters can keep the quiz, the projected content, and the control experience within the same delivery environment. This must consider both Desktop and Online execution contexts, as well as the restrictions imposed by the Office Add-in execution model.
- **To ensure architectural compatibility and real-time synchronisation**. The new capabilities must work harmoniously with the existing server-based architecture based on Node.js, Express, and PostgreSQL. They must preserve the server as the single source of truth and maintain immediate state synchronisation across all connected clients through WebSockets. This objective is essential to avoid fragmenting the platform or introducing inconsistent session states.
- **To validate the system in a real-world operational context**. The project must be tested under realistic live-session conditions to verify its technical robustness and practical usefulness. This includes real-device operational runs and automated end-to-end testing to ensure control responsiveness, integration stability, and correct behaviour across the main user workflows.

Chapter 3

State of the Art

THIS chapter presents an overview of the technologies, methodologies, and pedagogical approaches related to gamification applied to the educational field. It begins by exploring the theoretical foundations and core elements that make game-based learning effective, alongside a comparative analysis of existing interactive platforms used in classrooms today. Subsequently, the chapter delves into the modern web development paradigms, system architectures, and deployment strategies necessary to build scalable, real-time educational tools, with a particular focus on Microsoft Office Add-ins and containerised environments.

3.1 Gamification in the Classroom

3.1.1 Introduction to Gamification

Gamification has established itself as one of the most promising methodological strategies in 21st-century education. The term refers to the application of elements, mechanics, and dynamics typical of games in non-ludic contexts, with the aim of motivating, engaging, and improving participants' performance.

The concept began to gain popularity in the early 2010s, although its roots trace back to earlier practices of applying game mechanics in various fields. According to Deterding et al. (2011) [DDKN11], gamification differs from complete games in that it uses only specific elements of game design, without necessarily creating a complete game experience.

Gamification can be understood as a methodological approach that employs game elements (such as narratives, challenges, missions, and feedback) to motivate and engage students to achieve specific educational goals. As a methodological approach, it reorganises learning spaces and times, establishes the ways in which information will be provided, and reorients educational activities. Implementation in the classroom does not simply mean *playing* during classes, but strategically incorporating elements that promote active participation, sustained engagement, and meaningful learning. Figure 3.1 provides further insights into the core strategies and elements that constitute this educational approach.



Figure 3.1: Gamification strategies and elements in education.

Source:
`https://www.pickcel.com/blog/gamification-in-education-complete-guide/`

3.1.2 Theoretical Foundations of Educational Gamification

Gamification as a pedagogical strategy is based on various theories of learning and motivation that provide the conceptual framework necessary for its effective implementation. From a learning perspective, gamification is primarily grounded in constructivism [Fos13], where students actively build their knowledge through experience and reflection. Constructivism, developed by authors such as Piaget and Vygotsky, holds that learning is an active process where students construct new knowledge from their previous experiences. Gamification facilitates this process by providing environments where students can experiment, make mistakes, and learn iteratively [Kap12].

In the context of the digital age, connectivism, a theory proposed by Siemens, emphasises the importance of knowledge networks. Gamified experiences naturally align with this perspective by allowing students to connect concepts and collaborate in digital environments [KH08].

Regarding motivation, gamification is strongly framed within the self-determination theory by Deci and Ryan [DR13], which privileges the development of intrinsic motivation. This theory identifies three basic psychological needs:

- **Autonomy**: The need to feel that one controls their own actions.
- **Competence**: The need to feel capable and effective in the activities performed.
- **Relatedness**: The need to feel connected with others.

Intrinsic motivation is fundamental to engaging students and achieving meaningful learning. Well-designed gamified systems must preserve student autonomy and not abuse extrinsic rewards that may undermine intrinsic motivation. Experimental research has demonstrated that specific game elements, such as badges, leaderboards, and avatars, satisfy different basic psychological needs [SHMM17].

Complementing self-determination theory, the flow theory by Csikszentmihalyi [Csi90] describes a mental state of complete concentration where a person is fully absorbed in an activity. To achieve this optimal state, the activity must present an appropriate balance between the challenge it poses and the participant's skills. If the challenge is too high, anxiety is generated; if it is too low, boredom appears. Gamification allows for dynamic adjustment of this balance through progressive difficulty levels, creating conditions where students can experience flow and, consequently, deeper learning.

3.1.3 Game Elements in Learning Environments

Effective implementation of gamification requires understanding and properly applying the fundamental elements that compose game experiences. Within gamification, three fundamental elements intervene that provide a framework for designing coherent experiences: **dynamics**, **mechanics**, and **components** [WH20].

Game dynamics represent the high-level conceptual aspects that guide the player's experience. These include the narrative or story that gives meaning to activities, the sense of progression and continuous improvement, the social interactions and collaboration through relationships, and the emotions that arise during gameplay such as curiosity, competitiveness, and satisfaction. These dynamics create the overarching context within which learning activities unfold.

Game mechanics, on the other hand, are the processes that drive action and generate player engagement. They encompass challenges, tasks with clear objectives and progressive difficulty, as well as elements of competition that enable performance comparison among participants and cooperation that fosters teamwork towards common goals. Crucially, mechanics also include feedback mechanisms that provide immediate information about performance, and rewards that offer recognition for achievements reached. These mechanics translate abstract dynamics into concrete player actions.

Finally, game components are the specific, tangible elements associated with mechanics and dynamics. Points provide a numerical representation of progress or achievements, while badges offer visual recognition for specific accomplishments. Leaderboards create rankings that promote healthy competition, and levels indicate progression stages that reflect increasing mastery. Avatars allow for personalised visual representation of participants, and quests structure sets of related tasks. These components are the building blocks through which players interact with the gamified system.

Effective design of gamified experiences requires following a systematic process that integrates these elements coherently:

1. Context analysis: Identify student characteristics, content, and objectives
2. Objective definition: Establish clear and measurable educational goals
3. Element selection: Choose appropriate dynamics, mechanics, and components
4. Narrative design: Create an engaging story that provides context
5. Implementation: Put the design into practice in the educational environment
6. Evaluation and adjustment: Measure results and make iterative improvements

Figure 3.2 summarises the key features and design principles that make up these gamification elements.



Figure 3.2: Gamification key features.

Source: `https://mambo.io/gamification-guide/gamification-components-and-design-principles`

3.1.4 Benefits and Limitations of Gamification in Education

The application of gamification in educational contexts presents both significant benefits and important challenges that must be carefully considered. Studies demonstrate the substantial impact that gamification has on educational variables such as **motivation** and **student engagement** [SH20]. Students achieve a high level of commitment when they are motivated, even preferring to continue with the playful activity rather than ending the class. The interactive and playful nature of gamified experiences captures attention and maintains interest for extended periods.

This increased engagement often translates into improved academic performance. For instance, in a study involving students from the Faculty of Education in Albacete, results revealed that students exposed to gamification reported increased motivation and achieved better academic results compared to those taught through traditional methods [CMPRGP19]. However, other empirical evidence suggests that these benefits are not uniform across all contexts. Li et al. [LMS23] found that the effectiveness of gamification varies considerably depending on the subject area, with some disciplines benefiting far more than others, and that certain design approaches can even produce negative outcomes. This suggests that gamification may not be equally suited to all types of content, particularly those requiring factual knowledge retention or theoretical understanding.

Beyond academic metrics, gamification has the potential to foster creativity and research skills, as well as fostering collaborative learning. Among the competencies that are boosted are critical thinking and problem-solving, teamwork and collaboration, autonomy and self-regulation of learning, resilience to failure, and digital competencies. The immediate feedback characteristic of gamified systems allows students to know their results instantly and adjust their learning, contrasting with traditional assessments where feedback can take days or weeks.

However, the implementation of gamification is not without challenges and potential pitfalls. There is a notable risk of superficiality when gamification is based solely on points and badges without sufficient pedagogical foundation. The superficial application of game elements can result in experiences that entertain but do not educate.

Closely related to this is the danger of excessive reliance on extrinsic rewards, since when poorly designed, gamification can produce a sense of external control rather than genuine motivation. The abuse of external rewards (points, badges) can undermine intrinsic motivation, making students only interested in rewards and not in learning itself.

From a practical standpoint, designing effective gamified experiences requires pedagogical and game design knowledge, considerable preparation time, appropriate technological resources, and specific teacher training. Furthermore, not all students respond equally to gamification. Some may feel uncomfortable with competition or lack access to necessary technology, making it essential to design inclusive experiences that consider student diversity. Additionally, the motivational effect may diminish over time if not constantly renewed and adapted, presenting a sustainability challenge for long-term implementation.

3.1.5 Existing Gamification Platforms and Tools

Numerous digital platforms facilitate the implementation of gamified strategies in the classroom. The most relevant and widely used in the educational field are described below.

Kahoot!

Kahoot! [1] is a game-based learning platform that facilitates the creation, sharing, and use of educational games or trivia quizzes in a matter of minutes. Founded at the Norwegian University of Science and Technology and launched in 2013, it has established itself as one of the leading tools in the digital learning sector.

The platform offers several types of activities including Quiz, True/False, Discussion, and Survey formats, with multiple game modes that transform the same questionnaire into different experiences depending on whether participation is group-based or individual. Users can customise their activities by adding questions with images, videos, and up to four answer options. Figure 3.3 shows how to create a quiz. The teacher creates questionnaires and issues PIN codes for students to access from Kahoot.it, with scoring determined by the number of correct answers and response speed, the sooner one answers correctly, the more points can be earned.

Kahoot!'s ability to motivate and engage students has been widely documented, with the majority of studies reporting a positive effect on participation, motivation, and academic performance [WT20]. Teachers can use Kahoot! for any subject and educational level from Early Childhood, Primary, and Secondary Education, with results that can be exported to Excel or Google Drive for detailed analysis.

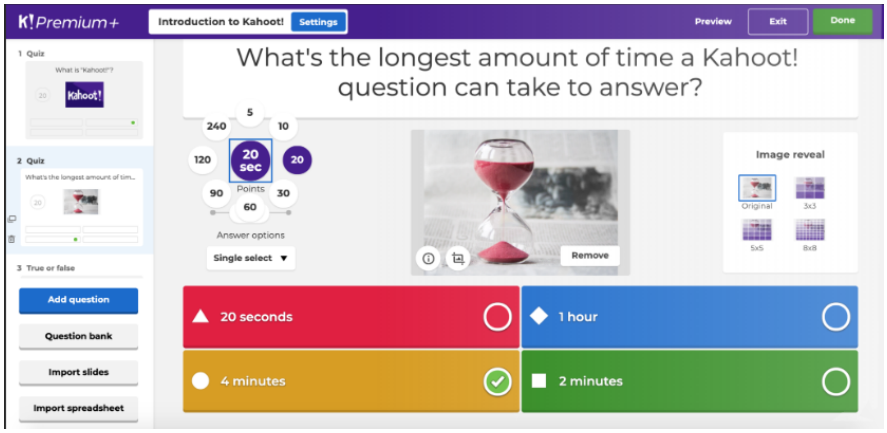


Figure 3.3: Kahoot! editing interface for a quiz session.

Source: `https://harwickmc.com/introduction-to-kahoot-video/`

Socrative

Socrative [2] is a free application that helps teachers motivate class participation and track student progress through interactive assessments with real-time response collection.

[1] `https://kahoot.com/es/`
[2] `https://www.socrative.com/`

The platform is compatible with any device, requires no personal profile, and supports up to 50 students per room in its free version. It offers multiple question types (multiple choice, true/false, and short answer) with no character limits. Activity modes include Quiz, Space Race, and Exit Ticket formats. Figures 3.4 and 3.5 show the teacher's interface and the competitive Space Race mode, respectively.

A distinctive advantage of Socrative is its instant feedback, allowing teachers to gauge understanding in seconds. Unlike timed platforms, it imposes no time limit, enabling students to answer without pressure. The platform generates downloadable reports in Excel or Google Drive, making it ideal for Flipped Classroom, collaborative learning, and BYOD (Bring Your Own Device) approaches.

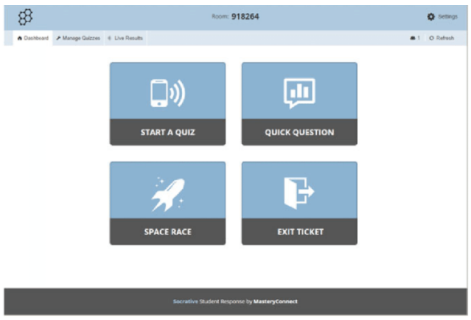


Figure 3.4: Socrative teacher desktop interface.

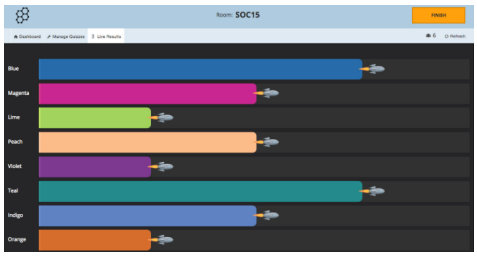


Figure 3.5: Socrative Space Race mode.

Sources: (left) https://www.researchgate.net/figure/Socrative-teacher-desktop-interface-with-different-features-for-sending-and-receiving_fig1_314253693, (right) https://www.techlearning.com/how-to/what-is-socrative-and-how-does-it-work-best-tips-and-tricks.

Mentimeter

Mentimeter [3] is an interactive presentation platform that incorporates gamification elements to encourage real-time audience participation. The platform supports live polls and voting, interactive word clouds, multiple choice questions, and rating scales, all with immediate visualisation of results. Its integration with PowerPoint allows seamless incorporation into existing presentations.

Mentimeter is especially effective for initial diagnostic assessment, comprehension checking during class, opinion gathering and debates, brainstorming activities, and satisfaction surveys. Its attractive and professional visual interface, combined with ease of integration into existing presentations, makes it particularly suitable for large groups. Recent studies have documented its effectiveness in improving participation and learning in classes with large numbers of students [MKP22]. A sample of its interface and available templates is shown in Figure 3.6.

[3] https://www.mentimeter.com/es-ES



Figure 3.6: Mentimeter templates and interface.

Source: `https://digi-ed.uk/support/article/what-are-the-mentimeter-templates`

Other market solutions

Other relevant platforms include: Quizizz [4] (individual pace, homework assignments), Classcraft [5] (complete role-playing game with avatars and teams), Gimkit [6] (economic mechanics), and Edpuzzle [7] (gamification through interactive video).

3.1.6 Comparative Analysis of Interactive Quiz Solutions

To facilitate the selection of the most appropriate platform according to specific educational needs, a comparative analysis of the main available solutions is shown in Table 3.1.

[4] `https://webflow-dev.quizizz.com/homepage-au?lng=es-ES`
[5] `https://www.hmhco.com/programs/classcraft`
[6] `https://www.gimkit.com/`
[7] `https://edpuzzle.com/`

Feature	Kahoot!	Socrative	Mentimeter	Quizizz
Launch year	2013	2010	2014	2015
Cost	Limited free version, paid plans from €4/month	Free up to 50 students	Limited free version, plans from €11.99/month	Robust free version, flexible quote to obtain pro plan
Game pace	Primarily synchronous (live quizzes)	Flexible: synchronous or self-paced activities	Synchronous interaction during live presentations	Synchronous or asynchronous (student-paced mode)
Edge	High (points, speed, leaderboard)	Low to medium	Low (focus on participation rather than scoring)	Medium (leaderboards with reduced time pressure)
Question types	Multiple choice, true/false, short answer, puzzles	Multiple choice, true/false, short answer	Multiple choice, scales, word clouds, open questions	Multiple choice, true/false, open-ended, fill in the blanks
Time limit	Configurable per question	Optional	Session-based rather than per-question	Configurable per question
Feedback	Immediate after each question	Immediate	Immediate visual feedback	Immediate, optionally with explanations
Reports	Detailed reports, exportable	Exportable reports (e.g., spreadsheets)	Basic reports in free version	Detailed reports per student and activity
Offline use	No (internet connection required)	No (internet connection required)	No (internet connection required)	No (internet connection required)
Multimedia	Images and videos	Limited multimedia support	Images (and multimedia depending on plan)	Images, videos, audio, and GIFs
Ease of use	Very high	High	Very high	High
LMS Integration	Yes (e.g., Google Classroom, LMS platforms)	Limited LMS integration	LMS integration depending on plan	Yes (Google Classroom and LMS platforms)

Table 3.1: Feature comparison between main educational gamification platforms.

Selection criteria

The selection of an educational platform is a multifaceted decision-making process that must transcend mere technical availability to ensure pedagogical effectiveness. According to the **TPCK framework** [MK06], the choice should be rooted in the intersection of technology, pedagogy, and specific subject content, rather than viewing the tool in isolation. Furthermore, the adoption of mobile technologies by teachers is strongly influenced by perceived usefulness and perceived ease of use, as well as by the school environment in which they are integrated [SPOMGP16]. Finally, to ensure inclusivity and flexibility, the selection must align with the principles of **Universal Design for Learning** [RM06], providing multiple means of engagement.

Thus, the choice of a platform should be based on several factors:

- Pedagogical objectives: Defining whether the focus is formative vs. summative assessment, or the reinforcement vs. introduction of concepts [MK06].
- Teacher profile: Variables such as age, years of teaching experience, gender, and type of school act as moderating variables in the acceptance of mobile technologies [SPOMGP16].
- Context of use: The heterogeneity of technological resources available across schools conditions the feasibility of integrating mobile devices into teaching practice [SPOMGP16].
- Type of content: Matching the platform to theoretical, practical, or conceptual learning materials [MK06].
- Learner variability: Providing multiple means of engagement and representation to accommodate differences in students' abilities, disabilities, and prior knowledge, ensuring equitable access to learning content [RM06].
- Available resources: The availability of time, access to materials, and human resources needed to develop educational activities with mobile devices constitute facilitating conditions that positively influence teachers' behavioural intention to use such technologies [SPOMGP16].

3.1.7 Evidence-Based Selection and Implementation

The effectiveness of gamification in educational contexts has been documented in multiple empirical studies. Meta-analyses examining educational gamification conclude that most studies report significant positive effects on cognitive, motivational, and behavioural learning outcomes, with effect sizes larger in school settings than in higher education or informal training contexts. Moreover, combining competitive and collaborative interaction proves more effective than competition alone, and benefits are more pronounced when game fiction is incorporated into the learning environment [SH20].

These findings are corroborated by empirical studies across different educational levels, from primary school to higher education. Research across various subjects and educational levels using Kahoot! has demonstrated significant improvements in learning performance and motivation compared to traditional teaching approaches [WT20], while studies at the university level have shown statistically significant improvements in academic performance, particularly among intrinsically motivated students [SC18, CMPRGP19].

Drawing from this empirical evidence and the comparative analysis presented in Table 3.1, several critical factors emerge for selecting and implementing gamification platforms effectively. First, the alignment between the platform's capabilities and the specific learning objectives is paramount. For instance, Kahoot's time-pressured format and competitive leaderboards make it particularly suitable for knowledge reinforcement and formative assessment where quick recall is valued, whereas Socrative's untimed approach may better serve deeper conceptual exploration where students need time to reflect.

Second, the student profile significantly influences platform effectiveness. Younger students or those with lower digital competence may benefit from the intuitive, streamlined interface of Kahoot!, while more advanced students might appreciate the flexibility and varied question types offered by Quizizz. Equity considerations are also crucial, given that platforms requiring premium features or high-bandwidth connections may create barriers for students with limited resources, making the robust free versions of Socrative or Quizizz preferable in such contexts.

Third, the context of use shapes platform selection. Synchronous classroom environments benefit from platforms like Kahoot! and Mentimeter that create shared, real-time experiences fostering social presence and collective engagement. In contrast, hybrid or asynchronous learning contexts require platforms like Quizizz that support student-paced modes and homework assignments.

The literature identifies additional success factors, including a balance between challenge and skill, the preservation of student autonomy, the provision of effective feedback, the integration of social components, and the consideration of students' individual traits [RM23]. Ultimately, effective selection requires moving beyond simple feature comparison to a holistic consideration of how the platform's design aligns with pedagogical goals, learner characteristics, and implementation context.

3.2 Web Development Technologies

The development of modern educational gamification platforms requires a solid understanding of contemporary web technologies. This section explores the fundamental technologies and architectures that enable the creation of interactive, real-time web applications suitable for classroom environments.

3.2.1 Modern Web Application Architectures

Modern web applications have evolved significantly from simple static pages to complex, interactive systems capable of handling real-time communication and processing large volumes of concurrent users.

Client-Server Model

The client-server architecture represents the foundational paradigm of web application design. In this model, the application is divided into two main components: the client, which handles the user interface and user interactions, and the server, which manages data processing, business logic, and storage.

The client, typically a web browser, sends requests to the server using the HTTP protocol. The server processes these requests, performs necessary operations (database queries, computations, validations), and returns responses to the client. An abstract representation of this model is shown in Figure 3.7. This separation of concerns provides advantages in scalability, maintainability, security, and resource optimisation. Modern implementations often employ three-tier architecture, which introduces a middle tier for application logic, separating it from both the presentation layer and the data layer, enhancing modularity and facilitating testing [Fow12].

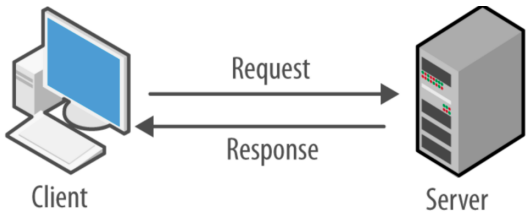


Figure 3.7: Abstract representation of the client-Server Model.

Source: `https://madooei.github.io/cs421_sp20_homepage/client-server-app/`

Real-Time Architectures (WebSockets)

Traditional HTTP communication follows a request-response pattern where the client must initiate every interaction. However, educational gamification applications, particularly quiz platforms, require real-time bidirectional communication to synchronise questions, display live results, and update leaderboards instantaneously.

WebSockets [8] provide a persistent, two-way communication channel over a single TCP connection. Unlike HTTP polling, WebSockets maintain an open connection that allows both client and server to send messages at any time without the overhead of establishing new connections [FM11].

[8] `https://datatracker.ietf.org/doc/html/rfc6455`

The WebSocket protocol begins with an HTTP handshake where the client sends an upgrade request, the server responds with HTTP 101 (Switching Protocols), and bidirectional communication proceeds over the persistent connection. For educational quiz applications, this architecture enables several critical capabilities. Teachers can broadcast questions simultaneously to all participants, ensuring synchronised quiz experiences across the classroom. Student responses are transmitted instantly, allowing for real-time answer collection without polling delays. Leaderboards refresh dynamically as answers arrive, maintaining engagement through immediate competitive feedback. Furthermore, all participants see identical timers and receive immediate feedback, creating a cohesive shared experience essential for interactive classroom activities.

Socket.IO [9] is a popular library that abstracts WebSocket communication, providing automatic reconnection, fallback to HTTP long-polling, and room-based communication ideal for classroom scenarios. Figure 3.8 shows an abstract representation of this connection.

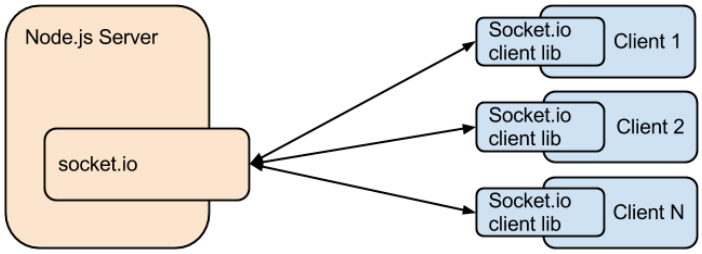


Figure 3.8: How Socket.IO works with several clients.

Source: `https://softwareengineeringdaily.com/2016/03/03/socket-io-and-realtime-applications-with-guillermo-rauch/`

3.2.2 Backend Technologies

The backend constitutes the server-side infrastructure responsible for data management, business logic execution, and API provision. Modern backend development emphasises performance, scalability, and developer productivity.

Node.js and Express

Node.js [10] is a JavaScript runtime built on Chrome's V8 engine that enables server-side JavaScript execution. Its event-driven, non-blocking I/O model makes it particularly suitable for real-time applications with high concurrent connections. The platform employs a single-threaded event loop that handles thousands of concurrent connections efficiently, while non-blocking I/O operations prevent the main thread from blocking, thereby improving overall throughput.

[9] `https://socket.io/`
[10] `https://nodejs.org/`

Additionally, Node.js provides access to an extensive ecosystem of over one million packages through the Node Package Manager (npm), facilitating rapid development and code reuse. Perhaps most significantly for full-stack development, Node.js enables the use of JavaScript across both frontend and backend, creating a unified development experience and allowing teams to share code and expertise across the entire application stack.

Building upon this foundation, Express.js [11] has emerged as the standard web framework for Node.js, providing a minimal yet flexible foundation for building web applications and APIs. Express middleware functions execute sequentially, processing requests before they reach route handlers, enabling modular implementation of authentication, logging, error handling, CORS configuration, and body parsing.

HTTP Security and Session Management

Web applications often rely on middleware to harden HTTP security and manage user sessions. Libraries such as Helmet [12] can set secure HTTP headers (e.g., Content Security Policy, X-Frame-Options), while session managers like Express-session [13] handle the creation and validation of session cookies. When applications are embedded in iframes or use third-party authentication, additional considerations like cookie `SameSite` [14] flags and secure transport become important. Server-side utilities such as `archiver` are commonly used to package multiple files (e.g., CSV report exports) into a downloadable ZIP archive.

Object-Relational Mapping (ORM) Tools

Object-relational mappers (ORMs) provide an abstraction layer that maps programming language objects to database tables and query results. They help developers work with data using language-native constructs, reducing redundant SQL and improving type safety. In the Node.js/TypeScript ecosystem, popular ORMs include TypeORM, Prisma, and MikroORM. TapClassroom uses MikroORM [15], a TypeScript-first ORM that supports entity definitions via decorators, automated schema synchronisation, and a unit-of-work pattern that scopes database operations to the lifetime of a request.

Relational Databases (PostgreSQL)

PostgreSQL [16] is an advanced open-source relational database management system known for its reliability, feature robustness, and standards compliance [Mom01]. It has been actively developed for over 35 years and supports both SQL (relational) and JSON (non-relational) querying [OH17].

[11] `https://expressjs.com/`
[12] `https://helmetjs.github.io/`.
[13] `https://www.npmjs.com/package/express-session`.
[14] `https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie/SameSite`.
[15] `https://mikro-orm.io/`.
[16] `https://www.postgresql.org/`

The system offers ACID compliance ensuring data integrity, along with support for complex queries including joins, subqueries, and window functions. Its native JSONB type provides flexibility for semi-structured data, while extensibility through custom functions allows for application-specific logic at the database level. Multi-Version Concurrency Control (MVCC) enables efficient handling of multiple simultaneous transactions without lock contention.

For quiz applications specifically, relational databases like PostgreSQL excel at maintaining referential integrity between interconnected entities such as quizzes, questions, participants, and responses, ensuring data consistency across complex educational workflows. A visual example of this data model is provided in Figure 3.9

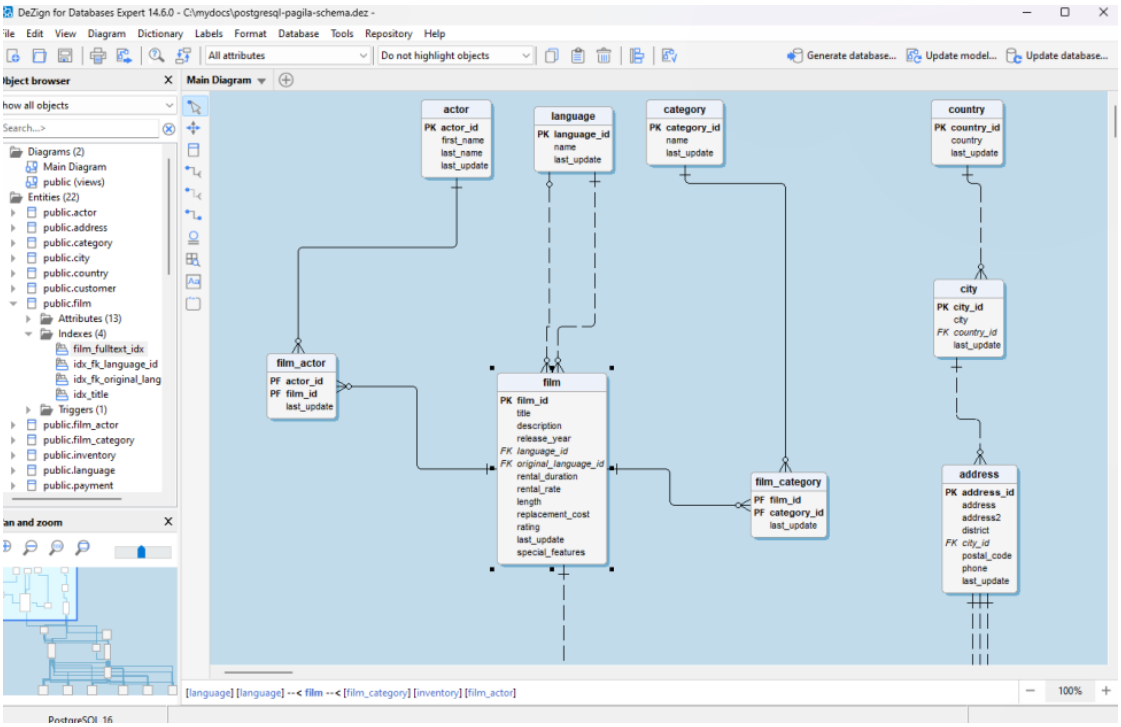


Figure 3.9: A visual example of a PostgreSQL data model.

Source: `https://www.datanamic.com/postgresql/`

RESTful APIs

Representational State Transfer (REST) is an architectural style for designing networked applications [Fie00]. RESTful APIs use HTTP methods to perform operations on resources identified by URIs.

The fundamental REST principles include statelessness, uniform interface, client-server separation, and layered system architecture. HTTP methods map to CRUD operations: `GET` retrieves resources, `POST` creates new resources, `PUT/PATCH` updates existing resources, and `DELETE` removes resources. RESTful APIs facilitate integration between systems, enabling quiz platforms to expose functionality consumable by web clients, mobile applications, or third-party services.

3.2.3 Frontend Technologies

Frontend technologies comprise the tools and languages used to create user interfaces that run in web browsers. The frontend is responsible for presenting information and capturing user interactions.

HTML, CSS, and JavaScript

The frontend technology stack is built upon three foundational technologies:

HTML (HyperText Markup Language) [17] provides the structural foundation of web pages. HTML5 introduced semantic elements (`<header>`, `<nav>`, `<article>`, `<section>`) that improve accessibility and search engine optimisation, along with APIs for multimedia, local storage, and canvas graphics [Pil10].

CSS (Cascading Style Sheets) [18] controls the visual presentation of HTML elements, with modern features including Flexbox and Grid for responsive layouts, custom properties (variables), animations and transitions, and media queries for adaptive design.

JavaScript [19] enables dynamic behaviour and interactivity. As the programming language of the web, JavaScript executes in browsers to manipulate the Document Object Model (DOM), handle events, communicate with servers, and manage application state. ECMAScript 6 (ES6) and subsequent versions introduced modern features such as arrow functions, classes, modules, promises, and async/await syntax [Fla20].

TypeScript

TypeScript [20] is a statically typed superset of JavaScript developed by Microsoft. It adds optional type annotations that enable compile-time error detection and improved tooling support.

Benefits of TypeScript include type safety (catching errors before runtime), enhanced IDE support with autocompletion and refactoring, self-documenting code through type annotations, and gradual adoption since JavaScript code is valid TypeScript. TypeScript compiles to plain JavaScript, ensuring compatibility with all browsers and runtimes.

Templating Engines and Server-Side Rendering

Server-side rendering (SSR) with templating engines generates HTML on the server and sends fully formed pages to the browser. Engines like EJS (Embedded JavaScript) [21], Pug, and Handlebars enable embedding dynamic data and control flow into HTML templates.

[17] `https://html.spec.whatwg.org/`
[18] `https://www.w3.org/Style/CSS/`
[19] `https://tc39.es/ecma262/`
[20] `https://www.typescriptlang.org/`
[21] `https://ejs.co/`.

This approach reduces the need for heavy client-side JavaScript frameworks and simplifies initial page loads, which can be advantageous for applications that need to work reliably across a wide range of devices and environments.

Browser Security and Iframe Constraints

Modern browsers enforce strict security policies when loading content inside cross-origin iframes. Content Security Policy (CSP) directives such as `frame-ancestors` determine which origins may embed a page [RBC+20], while Permissions Policy [22] (formerly Feature Policy) controls access to APIs such as the Web Serial API, screen wake lock, camera, and microphone. Additionally, browsers block third-party cookies unless they are marked `SameSite=None; Secure` [23], which affects authentication flows within embedded frames. These constraints are especially relevant for Office Add-ins and other embedded web experiences.

Headless Browsers and Server-Side PDF Generation

Generating rich, printable reports on the server often involves rendering HTML and chart graphics in a browser engine without a visible UI. Headless browser tools such as Puppeteer [24] automate Chromium in headless mode, allowing server-side code to load HTML, execute JavaScript, and export PDFs. Charting libraries such as Chart.js can be rendered in this environment using helper packages like chartjs-node-canvas [25], enabling dynamic generation of charts that are embedded in the final PDF output.

3.2.4 Microsoft Office Extension Development

Beyond standalone web applications, modern educational platforms can leverage the widespread use of productivity software by integrating directly into tools like Microsoft PowerPoint. Microsoft Office Add-ins extend Office applications with custom functionality, enabling integration with external services and enhanced user experiences within familiar productivity tools.

Introduction to Office Add-ins

Office Add-ins [26] are web applications that run inside Office applications (Word, Excel, PowerPoint, Outlook). Unlike traditional COM/VSTO add-ins, Office Add-ins use standard web technologies (HTML, CSS, JavaScript) and run in a sandboxed browser control within the Office application.

[22] `https://www.w3.org/TR/permissions-policy/`.
[23] `https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-rfc6265bis`.
[24] `https://pptr.dev/`.
[25] `https://github.com/SeanSobey/ChartjsNodeCanvas`.
[26] `https://learn.microsoft.com/en-us/office/dev/add-ins/overview/office-add-ins`

The web-based architecture provides cross-platform compatibility (Windows, Mac, web), familiar technologies for web developers, centralised deployment, sandboxed security, and automatic updates, as conceptually illustrated in Figure 3.10. Add-ins can appear as task panes (side panels), content add-ins (embedded within documents), or add-in commands (ribbon buttons and menu items).

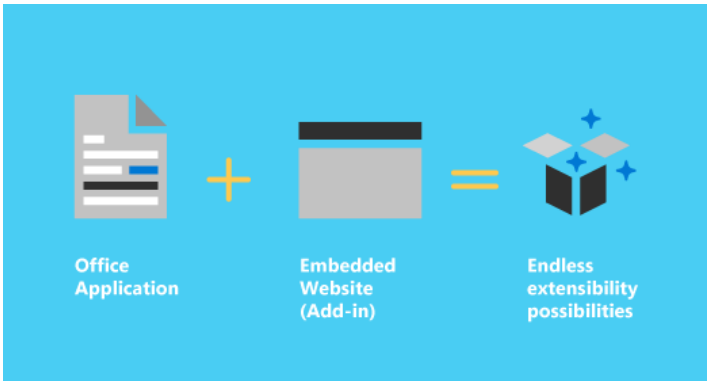


Figure 3.10: Add-in Overview.

Source:
`https://learn.microsoft.com/ko-kr/office/dev/add-ins/overview/office-add-ins`

Office.js: JavaScript API for Office

Office.js [27] is the JavaScript library that provides programmatic access to Office document content and application features. The API follows a promise-based asynchronous pattern and uses a proxy object model for efficient communication with the Office host.

The API is organised into the Common API (shared functionality across all Office applications) and Host-specific APIs (application-specific object models for Excel ranges, Word paragraphs, PowerPoint slides). Key capabilities include reading and modifying document content, responding to events, displaying custom UI, storing settings, and communicating with external web services.

Add-in Architecture for PowerPoint

PowerPoint Add-ins interact with presentations through the PowerPoint JavaScript API [28]. This API provides access to slides, shapes, text, and presentation structure.

The PowerPoint object model hierarchy includes Application, Presentation, SlideCollection, Slide, ShapeCollection, and Shape objects. For educational quiz integration, PowerPoint add-ins can navigate between slides programmatically, read slide content to generate questions, insert dynamic content such as QR codes, and synchronise presentation flow with external quiz engines. An example of a custom task pane integrated into PowerPoint is shown in Figure 3.11.

[27] `https://learn.microsoft.com/en-us/javascript/api/overview`
[28] `https://learn.microsoft.com/en-us/javascript/api/powerpoint`

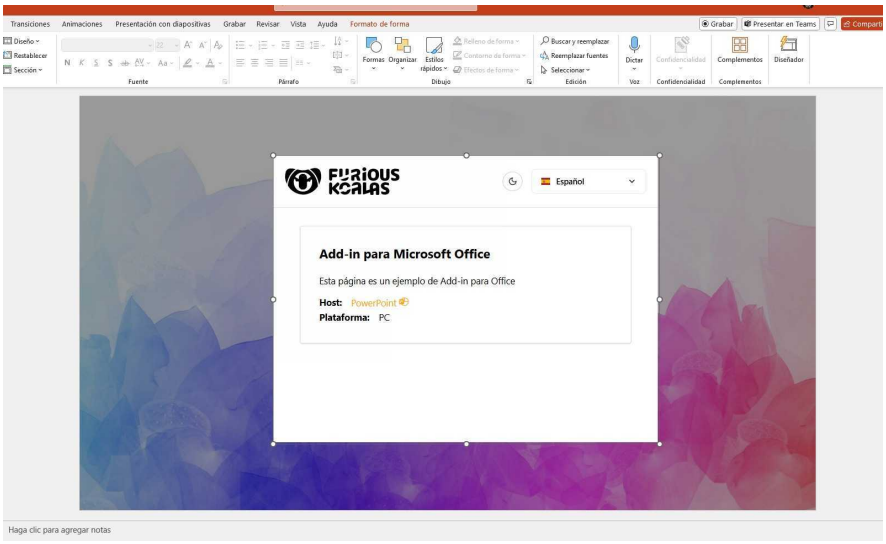


Figure 3.11: A PowerPoint Add-in created by Kurious Koalas as a test.

XML Manifest and Configuration

Every Office Add-in requires a manifest file that defines its properties, capabilities, and entry points. The XML manifest specifies identity information (ID, version, display name), target hosts, API requirements, source location, permissions (ReadDocument, WriteDocument, ReadWriteDocument), and UI commands. The manifest serves as the contract between the add-in and the Office host.

Integration of Add-ins with External Web Applications

Office Add-ins communicate with external web services through REST API calls, WebSocket connections for real-time features, OAuth authentication, and the Dialog API for authentication flows. For quiz applications, the add-in can transmit slide content to a quiz server, receive real-time responses, display live results within PowerPoint, and synchronise question timing with presentation navigation.

3.2.5 Containerisation and Deployment

Modern application deployment relies on containerisation technologies that ensure consistency across development, testing, and production environments.

Docker and Docker Compose

Docker [29] is a platform for developing, shipping, and running applications in containers. Containers package applications with their dependencies into standardised units that run consistently across any infrastructure [Mer14].

[29] `https://www.docker.com/`

Docker containers differ from virtual machines in that they share the host operating system kernel, making them lightweight and fast to start. A Dockerfile defines the container image through instructions: `FROM` (base image), `COPY` (add files), `RUN` (build commands), `EXPOSE` (ports), and `CMD` (startup command).

For applications requiring multiple interconnected services, Docker Compose [30] extends Docker's capabilities to multi-container applications. A `docker-compose.yml` file defines services, networks, and volumes, enabling complex application stacks to be started with a single command. A typical quiz application deployment might orchestrate a web server container running Node.js and Express for application logic, a database container with PostgreSQL for data persistence, a reverse proxy container using Nginx for load balancing and SSL termination, and optionally a caching layer with Redis to improve response times for frequently accessed data.

Secure Tunnels for Local Development

Some platforms (notably Microsoft Office Add-ins and webhook-based services) require publicly accessible HTTPS URLs. During local development, this can be achieved using tunneling services such as ngrok [31] and Cloudflare Tunnel [32]. These tools create an HTTPS endpoint that forwards requests to a local HTTP server, allowing developers to test integrations that depend on secure origins or iframe embedding without deploying to a remote server. The generated URLs are typically temporary, which has implications for manifest management and security.

Infrastructure as Code (IaC)

Infrastructure as Code treats infrastructure configuration as software, enabling version control, automated testing, and reproducible deployments [Mor16]. IaC tools define infrastructure in declarative configuration files rather than manual processes.

Popular IaC tools include Terraform (cloud-agnostic provisioning), AWS CloudFormation, Ansible (configuration management), and Kubernetes (container orchestration). Applications are typically deployed on cloud virtual machines (e.g., AWS EC2) or managed container platforms, using reverse proxies such as Nginx for TLS termination and routing. Benefits for educational platforms include reproducibility, version-controlled infrastructure changes, self-documenting configurations, automation reducing manual errors, and easy scalability.

[30] `https://docs.docker.com/compose/`
[31] `https://ngrok.com/`.
[32] `https://developers.cloudflare.com/cloudflare-one/connections/connect-apps/`.

3.2.6 Development Tools and Version Control

Effective software development requires tools for code management, collaboration, and quality assurance.

Version control systems track changes to source code over time, enabling collaboration and maintaining project history. Git [33] has become the dominant system, offering distributed architecture, flexible branching, and robust merging capabilities [CS14]. The system operates through key concepts including repositories (project containers), commits (snapshots of changes), branches (parallel development lines), merges (integration of changes), and remotes (synchronised copies).

Platforms such as GitHub [34], GitLab [35], and Bitbucket [36] extend Git's core functionality with collaborative features including pull requests for code review, issue tracking for project management, and continuous integration/continuous deployment (CI/CD) pipelines for automated testing and deployment.

Coding environments have come a long way to match the demands of modern development workflows. Visual Studio Code [37] has emerged as a leading code editor, offering an integrated terminal, sophisticated debugging capabilities, seamless Git integration, and an extensive extension ecosystem. Node.js projects typically utilise npm [38] or Yarn [39] for dependency management, with the `package.json` file serving as the central configuration document defining project metadata, dependencies, and scripts.

Quality assurance through automated testing has become integral to professional development practices. Testing operates at multiple levels: unit tests verify individual function behaviour, integration tests validate component interactions, and end-to-end tests simulate complete user workflows. Testing frameworks such as Jest [40] for JavaScript and pytest [41] for Python integrate with CI/CD pipelines to automatically validate changes before deployment, catching regressions and ensuring code quality throughout the development lifecycle.

[33] `https://git-scm.com/`
[34] `https://github.com/`
[35] `https://about.gitlab.com/`
[36] `https://bitbucket.org/`
[37] `https://code.visualstudio.com/`
[38] `https://www.npmjs.com/`
[39] `https://yarnpkg.com/`
[40] `https://jestjs.io/`
[41] `https://docs.pytest.org/`

Chapter 4

Methodology

THIS chapter describes the methodology followed to organise, develop, and validate the project. It distinguishes between the project management methodology, based on a Scrum-inspired agile framework, and the software development methodology, based on iterative and incremental implementation. It also summarises the work plan, validation strategy, repository workflow, and the main hardware and software resources used.

4.1 Methodological Approach

The project combines two complementary methodological dimensions. The first one concerns *project management*, which defines how work is planned, reviewed, and adapted over time. For this purpose, a Scrum-inspired framework is adopted, since the project involves evolving requirements, weekly feedback, and progressive clarification of technical constraints [SS20].

The second dimension concerns *software development*, which defines how the system is implemented. In this case, the work follows an iterative and incremental approach aligned with agile software engineering principles [FH+01]. Instead of attempting to specify the complete solution in advance, each contribution is developed through functional increments that can be tested, reviewed, and refined before moving to the next stage.

This distinction is important because the project includes three different product surfaces with different risks: a mobile control interface, a reports module, and a Microsoft PowerPoint add-in. The PowerPoint integration introduces uncertainty around Office.js, iframe restrictions, authentication inside embedded contexts, and PowerPoint Online limitations. The mobile interface and reports module, in turn, require adapting the existing TapClassroom platform without breaking its real-time behaviour. For these reasons, a rigid sequential methodology such as Waterfall is less suitable than an agile process that supports inspection and adaptation [TKB22].

4.2 Software Development: Agile Practices

The development approach is aligned with the agile values of feedback, collaboration, working software, and adaptation [FH+01]. The priority is to build demonstrable increments, review them frequently with the academic supervisor and the company contacts, and adjust technical decisions when new constraints appear.

This is especially relevant for the PowerPoint integration. The initial exploration of Office Add-ins requires comparing alternatives, validating assumptions inside the real host environment, and reacting to restrictions that are not fully visible before implementation. For the mobile interface and reports module, the same agile approach allows each user-facing capability to be checked independently before being integrated into the full TapClassroom workflow.

4.3 Project Management: Scrum-Inspired Framework

Scrum organises work into short iterations called sprints and encourages regular inspection of progress and adaptation of priorities [SS20]. In this project, the framework is adapted to an individual Final Degree Project developed within the context of an internship at Furious Koalas S.L. Therefore, formal Scrum ceremonies such as daily stand-ups or planning poker are not applied in their strict form, but the core principles of transparency, inspection, and adaptation are preserved.

The project is organised into 2-week sprints across an 18-week period. Each sprint has a limited set of goals, usually linked to one of the three main contributions or to the analysis and validation work required to support them. Weekly review meetings or asynchronous updates are used to present progress, discuss blockers, and adjust the next tasks. This cadence allows newly discovered constraints, especially those related to the Office Add-in environment, to be incorporated without redefining the entire project plan.

4.3.1 Roles

Although formal Scrum events such as daily stand-ups, planning poker, or retrospectives were not held, the key Scrum roles were represented informally:

- **David Vallejo**, the academic supervisor and co-founder of Furious Koalas S.L., the company where the internship for this project was conducted, took on a role similar to that of the **Scrum Master**. He provided guidance on technical strategy, facilitated the resolution of blockers, and helped refine goals at the start of each iteration.

- **Santiago Sánchez**, technologies director at Furious Koalas S.L. with in-depth knowledge of the TapClassroom platform, fulfilled a role equivalent to that of the **Product Owner**. He defined functional priorities, validated implemented features against the actual needs of the platform, and provided technical feedback on integration decisions.

- **Carlos González**, co-founder of Furious Koalas S.L., acted as the **client**. His periodic feedback shaped the higher-level direction of the product, ensuring that the development aligned with the company's vision and the practical requirements of their educational clients.

4.4 Software Development Methodology

From a development perspective, the project follows an iterative and incremental strategy. Each subsystem is developed as a functional increment that is prototyped, integrated, and validated under realistic conditions, keeping the focus on user-visible capabilities rather than isolated technical layers.

Consequently, implementation avoids a linear backend-to-frontend workflow. Instead, features are built as complete vertical slices encompassing server behaviour, view adaptation, and client interaction. This ensures early detection of integration issues, which is critical since multiple clients share the same quiz-state logic.

The PowerPoint add-in is treated as the highest-risk increment because it depends on an external host environment. Initial activities focus on feasibility, specifically evaluating add-in models, testing iframe execution, and identifying host restrictions. Once confirmed, development proceeds to navigation, host detection, authentication, and embedded rendering. Each step must be validated directly inside the Office context, as standard browser testing is insufficient.

The mobile interface and reports module are developed as platform extensions. The mobile interface reuses existing endpoints and WebSocket events to adapt the presenter experience for touch devices, progressing from essential controls to secondary management flows. The reports module reuses persisted data, adding a processing layer for participant summaries, question metrics, and file exports. In both cases, development remains incremental to ensure each functional path is checked before its scope is expanded.

4.5 Work Planning

The work plan is expressed as nine 2-week sprints. This representation is consistent with the Scrum-inspired management model and avoids treating the project as a set of independent phases. The expected sprint organisation is shown in Figure 4.1.

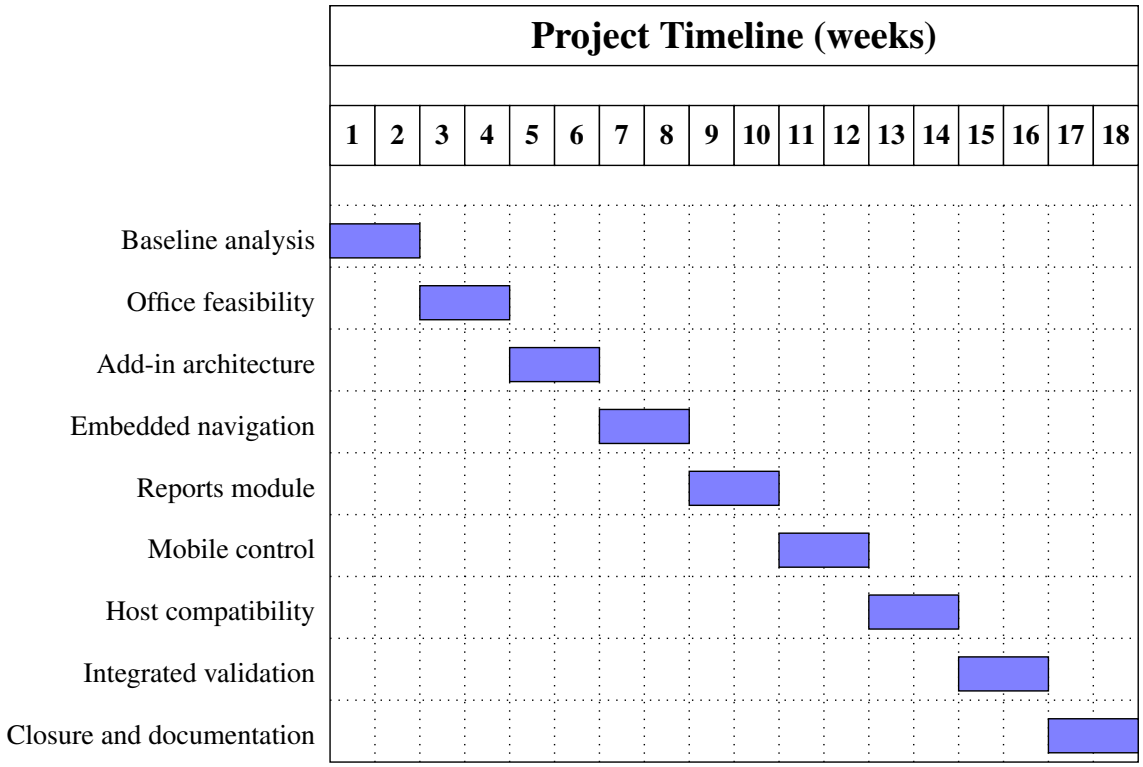


Figure 4.1: Sprint-based work plan for the project.

The planned sprints are briefly described below:

1. **Baseline analysis:** Understand the TapClassroom architecture, review the existing quiz workflow, and clarify the project requirements.
2. **Office feasibility:** Study the Office Add-in model, create initial prototypes, and identify the limitations of PowerPoint Desktop and PowerPoint Online.
3. **Add-in architecture:** Define the content add-in approach and the architectural adaptations required to embed TapClassroom inside a slide.
4. **Embedded navigation:** Adapt navigation and rendering behaviour so that browser-oriented views can operate inside the add-in frame.
5. **Reports module:** Implement post-session feedback, including participant summaries, question metrics, and exportable report formats.
6. **Mobile control:** Develop the mobile presenter interface and connect it to the existing quiz-state endpoints and real-time updates.
7. **Host compatibility:** Review authentication, iframe restrictions, sideloading, and native behaviour in PowerPoint.
8. **Integrated validation:** Test the main workflows across desktop, mobile, reports, stream views, and the PowerPoint add-in.

9. **Closure and documentation:** Consolidate the supported scope, refine the final deliverables, and complete the dissertation documentation.

4.6 Validation Strategy

Validation runs continuously across increments, not as a final phase. Every change is first verified locally and then in an integrated scenario with the server and all relevant client views (desktop control, mobile control, stream view, reports, and PowerPoint add-in), since all share the same quiz-state logic.

Each surface has its own validation focus. The PowerPoint add-in is checked in both Desktop and Online for navigation, auth persistence, and iframe/host constraints; the mobile interface is tested on a real iPhone 14 and via automated end-to-end flow; reports are validated by cross-checking web, CSV, and PDF outputs against the same event data.

The strategy mixes manual and automated execution. Manual testing covers environment-dependent risks like touch interaction and PowerPoint behaviour, while automation handles regression-prone paths such as mobile control and report generation.

4.7 Development Workflow

The project is distributed across two Git repositories. The first one, `simple-quiz-server`, is hosted on Bitbucket and contains the TapClassroom server, database model, real-time layer, and EJS-rendered views. The second one, `add-in-contenido-Tapclassroom`, is hosted on GitHub and contains the Office Add-in manifest and its associated TypeScript / Webpack source code.

A simplified branching model is used. Development related to this project is isolated from the stable TapClassroom production branch through a dedicated integration branch, while the add-in is versioned in its own repository. This organisation keeps the production codebase protected while allowing iterative development and validation.

4.8 Hardware and Software Resources

This section summarises the main resources required to carry out the project. The list is intentionally concise, but it preserves the distinction between hardware, operating system, programming languages, development tools, libraries, and documentation tools.

4.8.1 Hardware Resources

- **Personal computer**: The project is developed on a personal computer with an Intel Core i7 processor and 32 GB of RAM. This machine is used to run the TapClassroom stack locally, including the Node.js server, PostgreSQL database, Docker containers, and the Office Add-in development environment.

- **iPhone 14**: This device is used to validate the mobile control interface under realistic conditions. It allows the layout, touch interaction, viewport behaviour, and control responsiveness to be checked on an actual smartphone rather than only through browser emulation.

4.8.2 Operating System

- **Windows 11**: The development and validation environment is based on Windows 11. This operating system is especially relevant because PowerPoint Desktop sideloading and shared-folder catalogue configuration are primarily tested through the native Windows client.

4.8.3 Software Resources

Programming Languages

The main languages are TypeScript, JavaScript, HTML/EJS, CSS, and SQL. TypeScript is used in the backend and the Office Add-in project; JavaScript supports browser-side behaviour; EJS renders server-side views; CSS defines responsive and embedded layouts; and SQL underlies the PostgreSQL data model.

Development Tools

The main development tools are Visual Studio Code, Git, Docker, Node.js, npm, Webpack, Bitbucket, and GitHub. Ngrok, Cloudflare Tunnel, and Netlify [1] are used when HTTPS exposure or hosted test assets are required for PowerPoint Online and Office Add-in validation.

Libraries and Frameworks

The most relevant libraries and frameworks are Express, Socket.IO, MikroORM, Office.js, Puppeteer, Helmet, and `express-session`. Together, they support the HTTP backend, real-time communication, database access, PowerPoint host integration, PDF generation, automated validation, and embedded security configuration.

Design, Planning, and Documentation Tools

Adobe XD [2] is used to prototype the mobile interface; Draw.io [3], Mermaid [4], and Visual Paradigm Online [5] are used for diagrams and use case models; Slack [6] supports communication with the company; and Overleaf is used to write and compile the dissertation in LaTeX.

[1] `https://www.netlify.com`
[2] `https://helpx.adobe.com/xd/get-started.html`
[3] `https://app.diagrams.net`
[4] `https://mermaid.js.org`
[5] `https://online.visual-paradigm.com/`
[6] `https://slack.com`

Chapter 5

Architecture

THIS chapter provides a technical discussion of the internal structure of TapClassroom. It describes the main components of the system, explains how they interact, and highlights the design decisions behind the three subsystems developed during this project. A general overview is presented first, followed by more detailed descriptions of the backend, the frontend, and each of the contributed features.

5.1 General Overview

TapClassroom is a web-based gamification platform designed to support interactive quiz sessions in classroom environments. A host creates an event, loads a set of questions, and controls the flow of the session in real time, while participants join from their devices, submit answers, and receive immediate feedback through a live ranking. The platform also supports physical clicker devices, which are dedicated hardware units that participants use to submit answers via a USB-to-serial bridge connected to the host's machine, providing an alternative to smartphone-based participation in environments where device access is limited. The system is developed and maintained by Furious Koalas S.L., and this project extends it with three new subsystems: a mobile-friendly interface for session control, a reports module for post-session feedback, and an Office Add-in that integrates the platform directly into Microsoft PowerPoint.

Figure 5.1 shows a high-level component diagram of the system. The core infrastructure (the Express server, the PostgreSQL database, the Socket.IO layer, and the desktop web interface) was already in place before this project. The three contributions developed here are highlighted within that context: **the mobile interface**, which gives hosts full control of the session from a smartphone or tablet; **the reports module**, which generates detailed feedback for both hosts and participants after the session ends; and the set of adaptations that enable TapClassroom to run as an **embedded add-in inside PowerPoint**, including the state management required by PowerPoint Online.

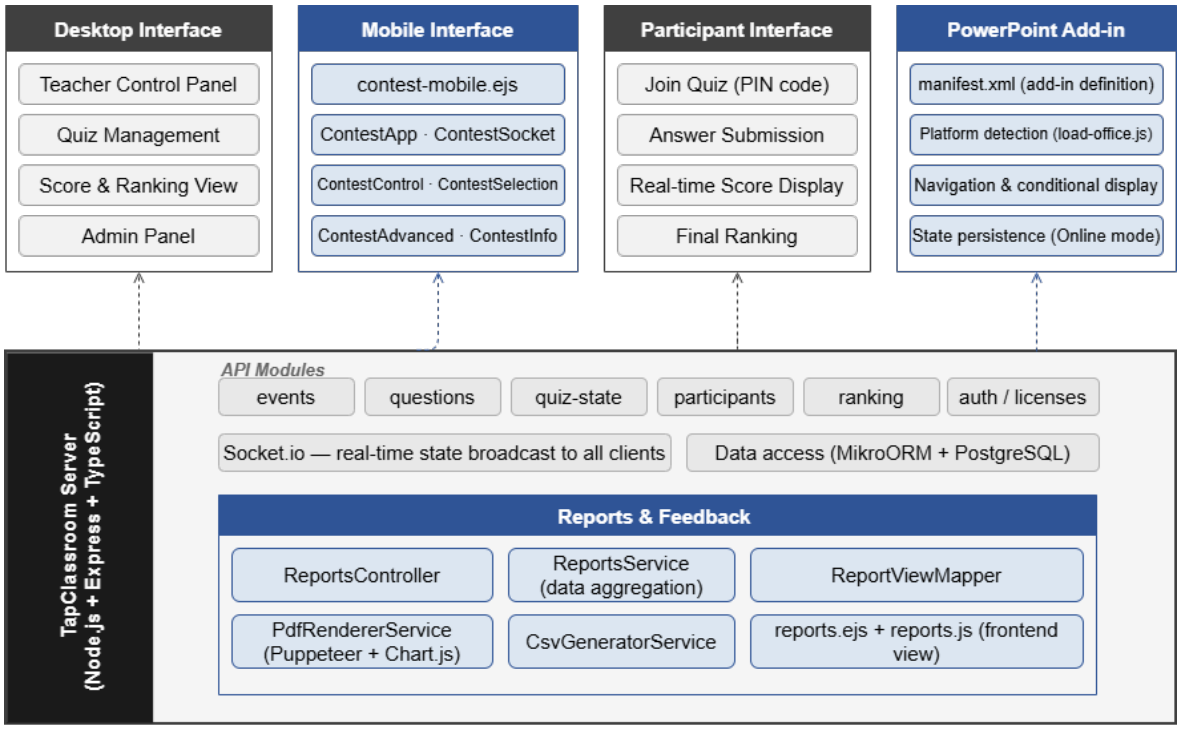


Figure 5.1: High-level component diagram of TapClassroom. The three subsystems developed in this project (mobile interface, reports module, and PowerPoint add-in) that are highlighted in blue, extend the platform as independent clients of the existing server infrastructure, without modifying the core API modules or the data layer beneath them.

Baseline System and Key Enhancements

Prior to this project, TapClassroom provided a functional quiz session system accessible from a desktop browser. Hosts could manage events, questions, and participants, and control the session flow through a web interface designed for large screens. Participants could join in two ways: from their own devices via a web browser, or through physical clicker devices connected to the host's machine, which the server registers as a separate participant type alongside web participants. The system worked, but three constraints limited how it could be used in practice.

The first was **mobility**. The host's control panel was not usable on a smartphone, which meant the host had to remain at a workstation throughout the session. A **mobile-first control interface** was the direct response, as it gives the host full session control from any device, without altering the underlying quiz-state API or replacing the existing desktop interface.

The second was the absence of **post-session insight**. Once a session ended, the data stayed in the database with no way to review performance, identify difficult questions, or export results. The **reports module** was built to close this gap, generating structured feedback automatically from the existing data without requiring any schema changes.

The third arose when the company wanted to begin delivering sessions from within PowerPoint presentations. **Context-switching between two applications** in front of an audience is disruptive, and the existing web interface was not designed to run inside an embedded iframe. The **PowerPoint add-in** embeds TapClassroom directly onto the slide surface, so the quiz content and session controls remain accessible throughout the presentation with no secondary window.

Each contribution was designed to extend the platform independently, without altering the existing architecture.

5.2 Backend

A live quiz platform poses a structural challenge. The server must simultaneously handle request-response interactions (creating events, loading questions, submitting answers) and push state changes to every connected client the instant the host acts. These two communication patterns have different requirements and are typically served by different mechanisms. The question is how to unify them under a single server without duplicating state or introducing synchronisation bugs.

The architecture adopted here follows a *server-based* model, where the server acts as the single, authoritative source of truth for the session state, a paradigm commonly used in networked game architectures [Ber01]. Under this model, the server updates its in-memory representation and broadcasts the state to all connected clients. While this approach often requires client-side prediction to hide latency in high-speed games, the nature of a quiz application allows for a simplified implementation. Therefore, clients act as passive observers that do not maintain independent mutable states, eliminating the need for complex reconciliation logic and ensuring that all participants remain strictly synchronised.

5.2.1 Layered architecture and request handling

Keeping this model correct as the application grows requires that business logic, request handling, and data access remain cleanly separated. Mixing them would make it easy for a controller to bypass validation, for a service to leak persistence concerns, or for a database query to end up scattered across unrelated files [Par72]. A layered architecture is the standard response to this problem, and it is the one adopted here.

The server is organised into three distinct layers, ensuring that concerns remain strictly isolated:

- **Routing and controllers**: The outermost layer receives every incoming HTTP request, runs it through a middleware pipeline (handling authentication, licence checks, and context injection), and delegates to a service.

- **Services**: The middle layer contains the core business logic, such as computing results, coordinating entities, and enforcing rules.
- **Data access**: The innermost layer maps TypeScript objects to database rows and scopes every database operation to the request lifecycle, ensuring transactions do not bleed across concurrent requests.

Figure 5.2 shows this structure from entry point to database.

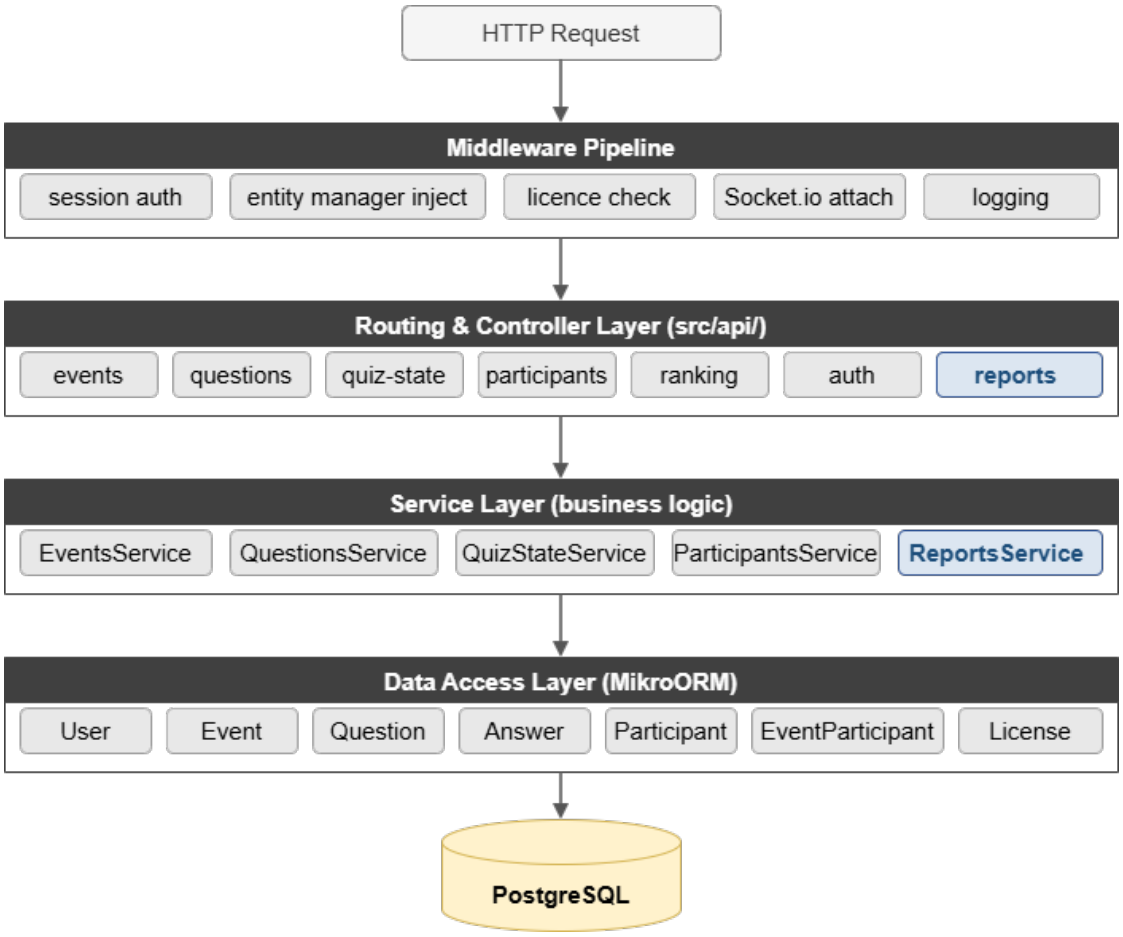


Figure 5.2: Layered architecture of the TapClassroom server. Each incoming HTTP request passes through the middleware pipeline before reaching the corresponding controller, which delegates to a service that operates on the database through MikroORM. The blue modules are the contributions of this project.

The result is a server where each functional domain is self-contained, so adding new subsystems only requires introducing their own controllers, services, and client-side modules, with cross-cutting changes confined to well-defined integration points [RF20]. The thin routing layer also allows the service layer to be tested without spinning up an HTTP server.

Domain organisation and middleware pipeline

The server is implemented in TypeScript on top of Node.js and Express. Each functional domain (events, questions, quiz state, participants, ranking, authentication, and reports) is encapsulated in its own module with a dedicated Express Router object. Each controller defines the HTTP routes for its domain and delegates all logic to a corresponding service, keeping the routing layer free of business concerns.

The middleware pipeline runs before every request reaches its route handler. The middlewares in use are session-based authentication, entity manager injection, and Socket.IO context attachment. This pipeline enforces input validation, session checks, and licence rules at the system boundary, so that no service needs to repeat those checks internally. The data access layer is built on MikroORM, an object-relational mapper for TypeScript; each request receives its own entity manager instance injected by the pipeline, which scopes all database operations to the request lifecycle and prevents transactions from bleeding across concurrent requests.

5.2.2 Data model and session isolation

A secondary structural question concerns how session data should be organised. Each quiz event involves the same participants, but scores must remain isolated per event. Duplicating participant records per session creates inconsistency, while a flat design without a junction entity cannot represent a score independently from a participant's identity.

The solution is an associative relationship that records each participant's association with a specific event along with their accumulated score as a relationship attribute [EN16], allowing core entities to be reused across events while per-session data stays cleanly separated.

This design also determined the scope of this project's contributions. The schema was not modified, since the reports module and the mobile interface read aggregated data already written by the quiz logic, operating entirely on existing entities. Extending the platform through new subsystems rather than altering its data foundation kept regression risk low and integration straightforward.

Entity mapping and association loading

The entities are defined as TypeScript classes and mapped to PostgreSQL tables through MikroORM decorators. Relationships between entities, such as the association between an `Event` and its `Questions`, or between a `Participant` and its `EventParticipant` records, are declared as typed properties and resolved by the ORM at query time. MikroORM's eager population is used when a service needs to load an entity together with all its associations in a single query, as the reports service does when assembling the full dataset for a session, including event metadata, questions, answers, and participant records, which are all loaded together before any statistics are computed, avoiding repeated round-trips to the database.

5.2.3 Real-time state synchronisation

The server-based model depends on a reliable broadcast mechanism. When the host advances a question or opens the poll, every connected client (potentially dozens of participant devices and a projected stream view) must receive the update within milliseconds and render consistently. Polling would introduce latency and unnecessary load; a persistent connection that lets the server push updates is the appropriate tool.

The server maintains an in-memory representation of the current quiz state for each active session, containing the active question, poll status, screen mode, participant counts, score multiplier, and other session parameters. This object is the single source of truth for that session. When a host action modifies it, the server serialises the updated state and broadcasts it to all clients subscribed to that session's channel. Clients re-render their views from the broadcast and may keep transient local state for rendering purposes, but they do not own the authoritative quiz state. This design also simplifies the addition of new client types. The mobile interface and the PowerPoint add-in, both contributed by this project, reuse the same broadcast events and REST endpoints as the existing desktop interface, so the core API did not need to change.

Supporting simultaneous sessions without interference requires that each host's state be isolated. The state object is keyed per authenticated user and active event, so two hosts running sessions in parallel operate on entirely separate state trees. Figure 5.3 illustrates the full message flow for a single state update.

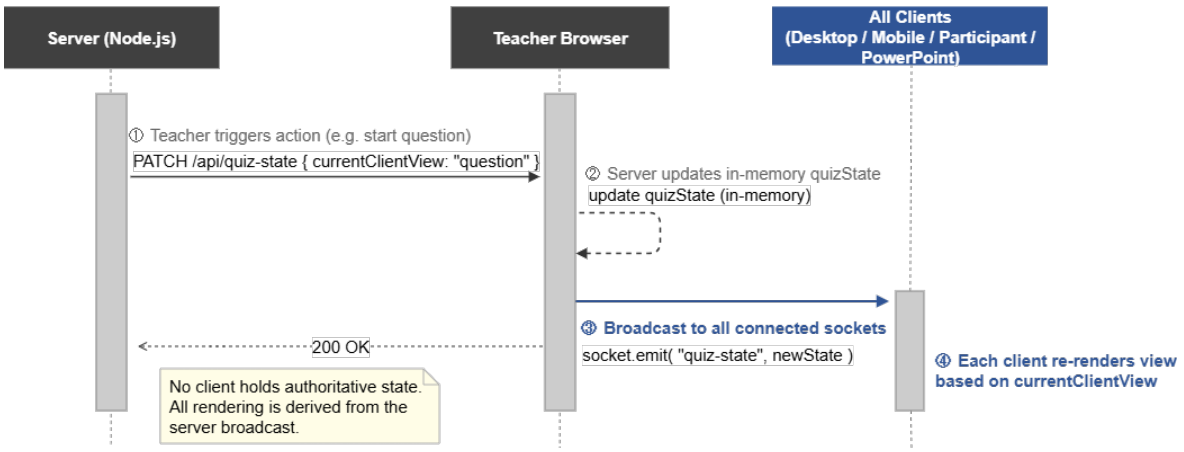


Figure 5.3: Sequence diagram of a real-time state update. The host's action triggers a `PATCH` request; the server updates its in-memory state and broadcasts the new state to all connected clients, which re-render their views accordingly.

Socket.io rooms and scoped state

The state exists at two levels. The original desktop interface relies on a global `quizState` object; the scoped state map, managed by `scoped-quiz.ts`, isolates state per authenticated host and active event. Each write operation (opening the poll, advancing a question, ending the session) goes through this module, which updates the correct state tree and emits the updated object to the corresponding Socket.IO room. Clients join the room for their session on connection and leave it on disconnect; the server never pushes state to clients outside the relevant room.

The real-time layer is implemented with Socket.IO, which provides persistent bidirectional connections over WebSockets with an automatic fallback to long-polling for environments that block WebSocket traffic. The Socket.IO instance is made available inside request handlers through the middleware pipeline, without being imported directly by the service layer. This keeps services free of real-time concerns and allows them to trigger broadcasts as a natural side effect of handling an HTTP request, with no coupling to the transport mechanism.

5.3 Frontend

The frontend of TapClassroom must serve several audiences simultaneously: a host controlling the session, participants submitting answers, and a projected stream visible to the whole room. Each of these has a fundamentally different relationship with the server, ranging from full bidirectional interaction to purely passive display. The challenge is how to support all of them consistently without duplicating infrastructure or introducing separate build pipelines.

The existing system addressed this through server-side rendering (SSR), where all pages are produced by the Node.js server and delivered as complete HTML documents, with lightweight client-side scripts handling only the dynamic behaviour that cannot be resolved at request time. Given that the SSR strategy was already in place, the decision for this project was to extend it rather than introduce a parallel client-side framework, for the following reasons: first, this approach keeps the frontend tightly integrated with the backend; and second, it avoids the complexity of maintaining two independent build systems over a codebase that was not designed for that split. Consequently, the three contributions of this project were built on top of this foundation.

5.3.1 Rendering model and view coordination

The core challenge in a server-based frontend is keeping views consistent with server state without making each client responsible for managing that state independently. If every view maintained its own copy of the quiz state, divergence would be inevitable. The solution is a clear separation between the initial render, handled entirely by the server before anything reaches the browser, and subsequent updates, pushed by the server and applied in place by a lightweight client script. This keeps the server as the single source of truth for both the initial state and every change that follows, without requiring a client-side framework or a separate build pipeline.

The advantage of this approach over a fully client-side architecture is that the server can enforce authentication, licence checks, and data access rules at render time, before any HTML is delivered. There is no risk of a client rendering a view it is not authorised to see, because the server simply does not produce it. Figure 5.4 shows the resulting server-side rendering pipeline.

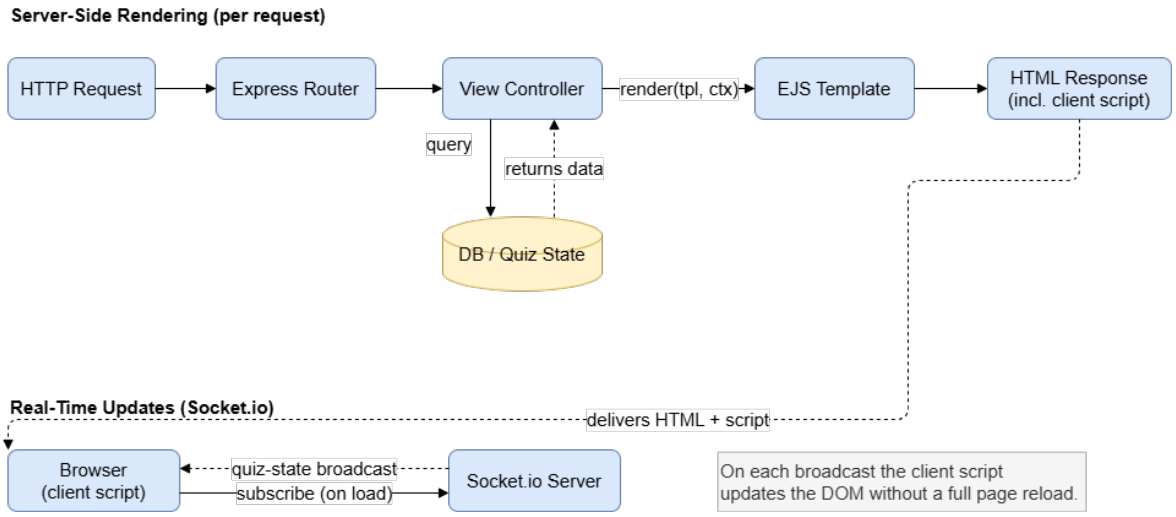


Figure 5.4: Server-side rendering pipeline for TapClassroom views. The server assembles data and renders the full HTML before delivery; the embedded client script then subscribes to Socket.IO events and patches the DOM in place, so the server remains the authority for both the initial render and all subsequent state changes.

View-controller pattern and EJS rendering

Each page is served by a dedicated view-controller that assembles the required data and passes it to an EJS template as a context object. The template produces a complete HTML document that is immediately displayable without any client-side processing. Once loaded in the browser, an embedded JavaScript module subscribes to Socket.IO events and updates only the DOM elements that need to change, without triggering a full page reload. The view-controller therefore handles both the data assembly and the template rendering, keeping those concerns out of the service layer.

5.3.2 Desktop views and interaction roles

A quiz session involves several views that differ not just in their content but in their role within the system. Some views send commands to the server, some receive state and react to it, and some do both. If these roles are not made explicit, there is a risk of a view accumulating responsibilities that belong elsewhere, a passive display caching state it should not own, or a control view rendering content that should be driven by the server. Making the role of each view a first-class design decision keeps the system predictable. A view that never sends requests cannot produce side effects, and a view that never holds local state cannot diverge from the server.

The desktop interface organises its views along this axis. Some are fully bidirectional, some are mixed, and some are purely passive. The purely passive views are particularly important in a classroom context, because they are projected on a shared screen, where the guarantee that they cannot trigger any server-side action is not just a design preference, but a safety property. Figure 5.5 illustrates these interaction roles.

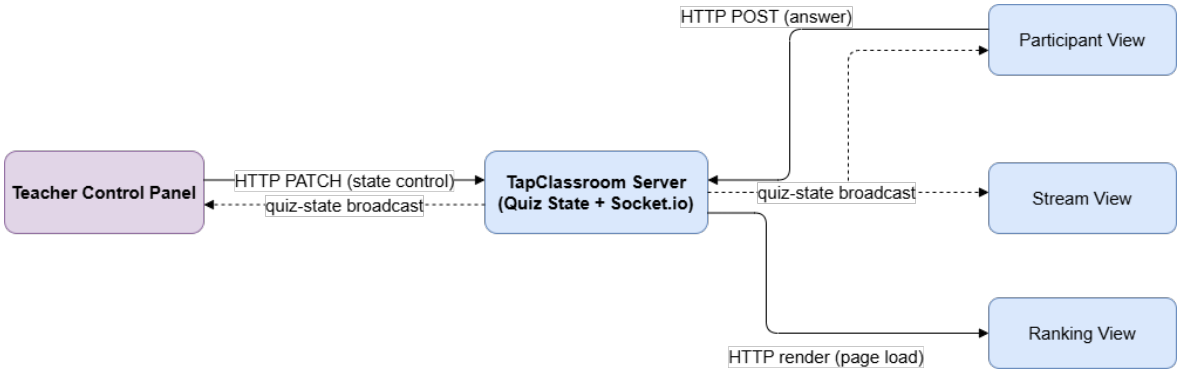


Figure 5.5: Interaction roles of the desktop views. The host control panel is the main administrative bidirectional view; the participant view submits answers via WebSocket and reacts to state updates; the stream view subscribes to a dedicated Socket.IO room; the ranking view is rendered server-side on demand, with no real-time dependency.

View implementation and client scripts

Each desktop view is implemented as an EJS template rendered by a dedicated view-controller. This separation of responsibilities defines four distinct interaction roles:

- **Host control panel** (*fully bidirectional*): Delegates session commands to the quiz-state API module and receives real-time updates via Socket.IO events (`change view`, `change question`, `change poll state`).
- **Participant view** (*mixed role*): Submits answers by emitting an `answer` event over WebSocket and re-renders on broadcasts, but does not own the authoritative quiz state.
- **Stream view** (*purely passive*): Joins a dedicated Socket.IO room and reads the `EClientView` mode from each broadcast to determine which content to display, without issuing any requests.

- **Ranking view** (*server-rendered*): Generated on demand after each question and at session end; it requires no Socket.IO connection, making it safe to project on a shared screen with no side effects.

5.4 Mobile Interface

The mobile interface is one of the three contributions of this project. Before it existed, the only way for a host to control a TapClassroom session was from a desktop browser. The existing control panel assumed a wide viewport, relied on mouse-precision interactions, and was not usable on a smartphone. In practice, this forced the host to remain at a fixed workstation throughout the session, which limited the dynamics of classroom delivery.

The starting point was a minimal mobile footprint, as a reduced version of the desktop interface that only exposed event selection and participant registration, with no session control capabilities. Furious Koalas S.L. provided the visual design for the full interface as a set of mockups covering each screen and its interactive states.

The contribution described in this section is the implementation of those mockups and their integration with the existing backend, without modifying the underlying session-control API. The result is a fully functional session control panel that the host can use from any smartphone or tablet, freeing them from the workstation without replacing the existing server-side architecture or breaking any existing interface, as shown in Figure 5.6.

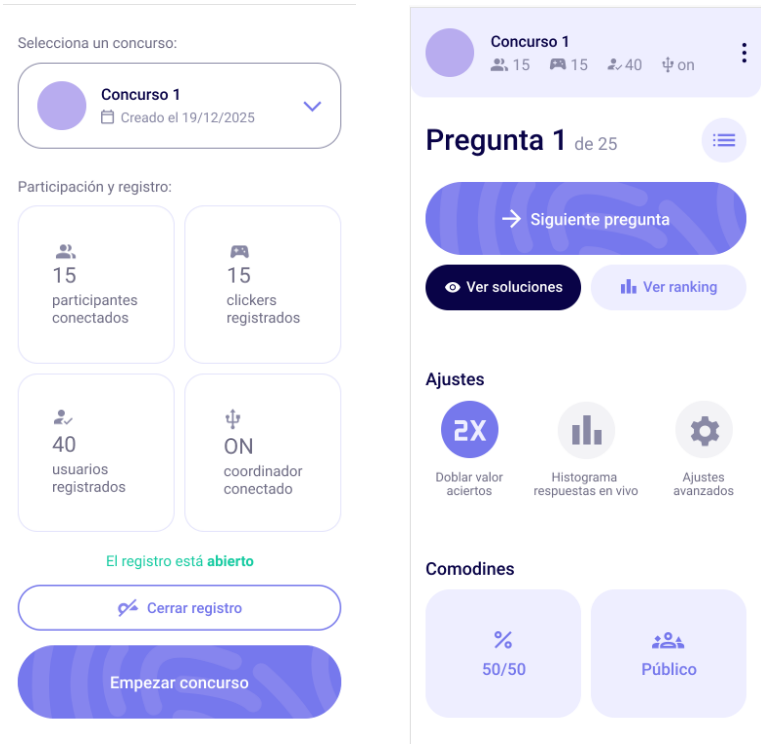


Figure 5.6: Initial mockups for the mobile interface of TapClassroom. The selection view (left) lets the host choose an event and open registration; the control view (right) provides full session control during an active quiz.

5.4.1 Serving a dedicated interface per device type

The mobile interface is not a responsive adaptation of the desktop interface, but rather a completely different UI with a different layout, different interaction model, and different view structure. A responsive approach using CSS media queries would allow the same HTML to reflow at different screen sizes, but it cannot produce two structurally different interfaces from the same template. Serving a dedicated template per device type was the only design that matched the scope of the difference between the two interfaces, while keeping the backend and business logic completely shared.

This approach also has a clear advantage over maintaining a separate frontend application. The mobile interface is served by the same Node.js server, uses the same session authentication, and receives the same server-rendered context as the desktop interface. By reusing these components, it explicitly avoids duplicating the backend infrastructure, the user session management, the real-time WebSocket communication channels, the data access logic, and the contextual data, ensuring there is no duplication of API logic and no need for an additional deployment target.

Device detection and template routing

Detection is performed on the server by inspecting the `User-Agent` header of each incoming request through a regular expression that matches common mobile device identifiers. When a match is found, the server renders `contest-mobile.ejs` instead of the desktop template. The query parameter `force_mobile=true` bypasses detection for development purposes, allowing the mobile interface to be tested from a desktop browser. The same context object is passed to both templates, so the mobile interface has access to the same server-side data (current quiz state, event list, participant counts, question metadata) without requiring any additional endpoints.

5.4.2 Single-page architecture and view management

A session control interface on mobile has a hard constraint, since the WebSocket connection to the server must remain alive throughout the session. A host who loses the connection mid-session cannot control the quiz until the connection is re-established, and any ongoing state (poll open, active question, participant counts) would need to be re-fetched. Navigating between separate pages would tear down and re-establish the connection on every transition, making that failure mode a routine occurrence rather than an exception.

The solution was to deliver the entire interface as a single-page application [MP13], managing all view transitions client-side while keeping the Socket.IO connection uninterrupted for the duration of the session. Transitions between views produce no network activity and are imperceptible to the user, and since the full application state remains in memory, switching between views requires no re-fetching from the server.

View structure and transitions

The interface presents five distinct views within the same mobile workflow. During active session control they are navigated in place inside a single HTML document, although some actions in the selection phase still trigger a reload to rehydrate server-rendered data:

- **ContestSelection**: The entry point, shown when no session is active. It exposes event selection, participant registration controls, and current participant counts.
- **ContestControl**: The main operational view during a running session, grouping all controls the host needs during playback: question navigation, poll toggle with auto-close timer, score multiplier, wildcard actions, and session management.
- **ContestAdvanced**: Provides specific configuration fields for the poll timeout and the hangman auto-reveal interval.
- **ContestInfo**: A summary panel accessible from the header menu showing active participant and physical clicker statistics.
- **ContestTransition**: A minimal confirmation screen shown when the session ends, with a manual return path to the main selection view.

Only one view is visible at a time; `ContestApp` manages the in-place transitions by toggling CSS visibility classes on the view containers. Figure 5.7 illustrates the navigation between views.

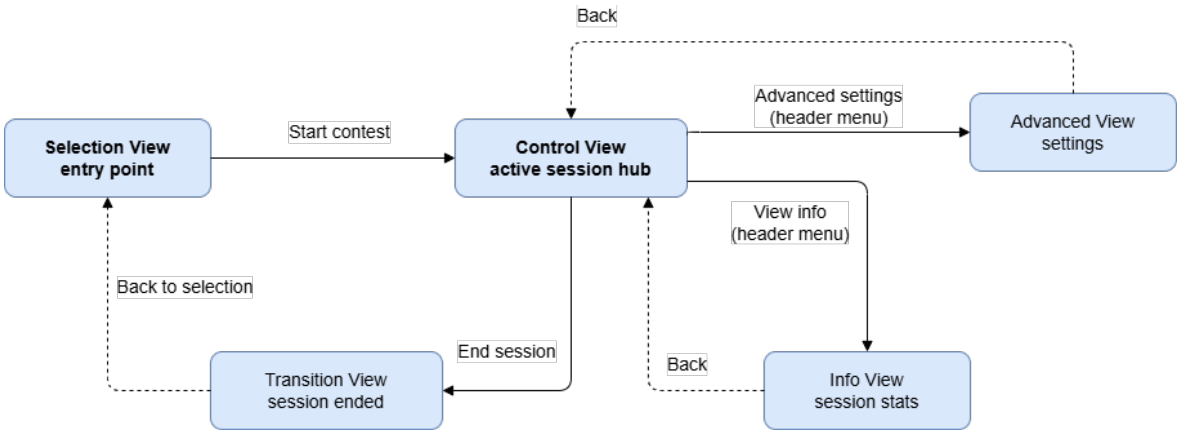


Figure 5.7: Navigation between the five views of the mobile interface. The operational views coexist in a single HTML document; in-place transitions are managed by CSS class toggling so the WebSocket connection remains alive throughout the active session.

5.4.3 Modular client architecture

A single-page interface with five views, real-time state updates, and multiple interactive controls accumulates enough complexity that a monolithic script would violate the cohesion principles [YC79]. Mixing Socket.IO event handling, DOM update logic, and view transition code in a single file would make changes error-prone. The interface was therefore structured as a set of ES6 modules from the start, each owning a clearly bounded responsibility.

This separation also solves the data flow problem. With multiple modules reacting to the same server events and updating the same DOM elements, there is a risk of conflicting updates if each module maintains its own copy of the state. Centralising the state in a single shared object, owned by the root orchestrator, ensures that all modules work from the same source of truth and that state changes are predictable and traceable.

Module structure and data flow

`ContestApp` is the root orchestrator. Instantiates all other modules, wires them together, and owns the shared `state` object and `elements` map. It reads the initial server state from a `window.contestMobileData` object injected by the EJS template at render time, and exposes itself on `window.contestApp` for cross-module communication. Each module receives a reference to the shared state and elements map at initialisation, so there is one source of truth throughout the client. Figure 5.8 shows the module dependency structure.

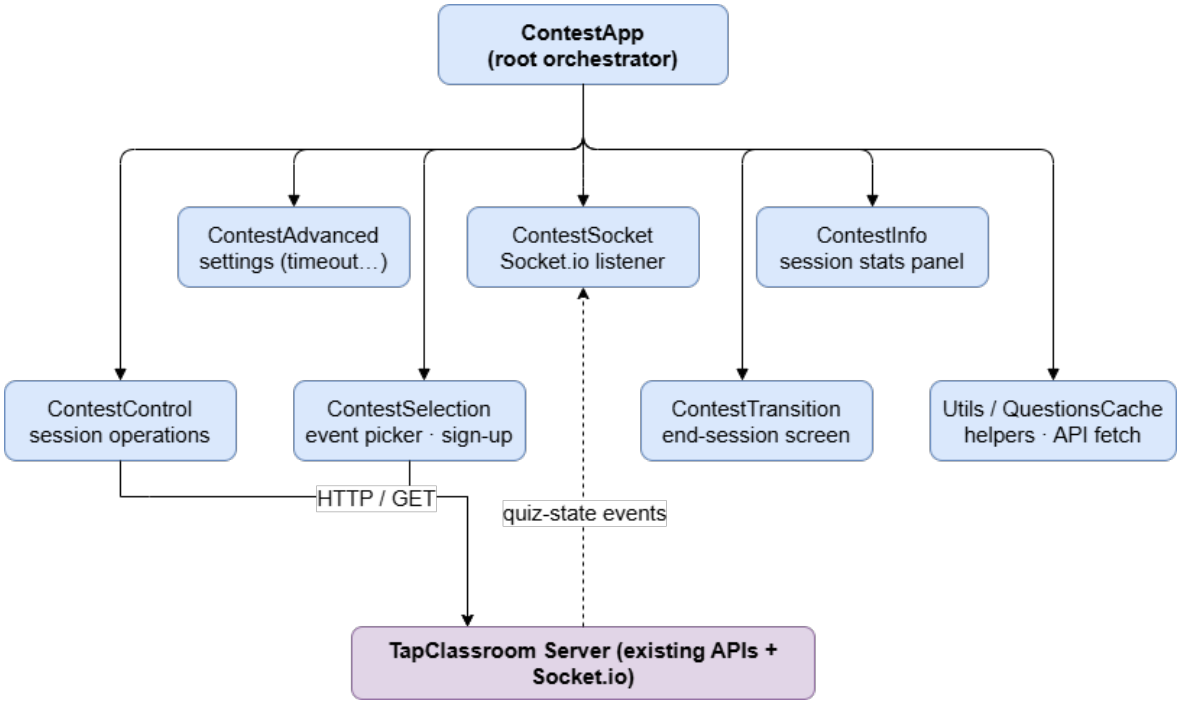


Figure 5.8: ES6 module structure of the mobile interface. `ContestApp` is the root orchestrator; all modules share a single `state` object and a common `elements` map. `ContestSocket` is the main consumer of external real-time events, while write requests are distributed across the UI modules that own each part of the control flow.

5.4.4 Real-time communication and clicker bridge

The mobile interface needs to stay synchronised with the server state and also reflect the status of the physical clicker hardware. These are two distinct communication channels with different origins and different reliability profiles. Specifically, the Socket.IO connection is a persistent network channel managed by the server, while the clicker bridge is a local hardware connection managed by a separate script on the client. Mixing them in the same communication layer would couple the reliability of one to the other.

The solution was to treat them as independent event sources and let the UI react to both through the same DOM-event model. Network events arrive via Socket.IO and are handled by one module; hardware events are dispatched as custom DOM events and handled by another. The UI layer does not need to know the origin of an event, only how to respond to it. This design also means that if the clicker bridge is unavailable, the Socket.IO layer continues to function normally, and vice versa.

The broader advantage is the same one that applies throughout the mobile interface. The server-based model described in Section 5.2 treats all clients equally, the mobile interface required no changes to the underlying quiz-state API. It reuses the same Socket.IO events and REST endpoints as the desktop interface, although dedicated server-side routing and template rendering were added for the mobile view.

Socket.IO events, clicker bridge, and touch adaptation

The `ContestSocket` module opens a Socket.IO connection when the page loads and binds handlers for the server-sent real-time events. Among them, `quiz-state-changed` and `change question` are the most important for the control flow, as they deliver state updates that are merged into the shared local state and reflected immediately in the UI. The `participant-joined` and `participant-left` events are used only to keep the connected participant count in sync.

The clicker bridge is handled by the `CoordinatorStatus` component, a separate export from the same module. It does not communicate via Socket.IO; instead it listens for the custom DOM event `clicker-bridge:status`, dispatched by the script that manages the serial connection. The status indicator in the header and in the info view is updated whenever this event fires, independently of the network state.

All interactive controls are standard HTML elements sized for finger input, which is sufficient for the tap-based interaction model of a session control panel. The only device-specific adaptations required were at the viewport level, where the `meta` tag sets `viewport-fit=cover` and `user-scalable=no` to prevent accidental zoom during session control, and CSS environment variables handle the safe-area insets on iOS devices with a notch or home indicator, ensuring that controls are never obscured by the system UI.

5.5 Reports and Feedback System

The reports module is the second contribution of this project. Before it existed, a TapClassroom session produced no persistent output beyond the raw data stored in the database. Once a session ended, there was no way to review individual performance, identify difficult questions, or export results for further use. The session ended with a ranking, but with no explanation of what it meant.

The need for a feedback mechanism was identified from two directions. Hosts needed to understand which questions were hard, which participants struggled, and how response times varied. Furious Koalas S.L. needed to export raw session data in a machine-readable format for external tools. These two requirements point to different output formats with different audiences, which shaped the module's architecture from the start.

The reports module addresses both needs through a single shared data pipeline that computes the session statistics once and routes them to three independent output formats: an interactive web view for the host, a printable PDF report, and a machine-readable CSV export. The result is that every session produces structured, actionable feedback automatically, without requiring any additional input from the host.

5.5.1 Output formats and the shared data pipeline

The central design challenge was that two structurally different outputs (a machine-readable export and a human-readable report) needed to be derived from the same underlying data. Building two independent data pipelines would have duplicated the query logic and risked inconsistencies between the CSV and PDF versions of the same session. The solution was to compute the statistics once and route the results through separate rendering paths depending on the requested format.

This separation also determined the development order. CSV generation is the simplest output. The data pipeline computes the statistics, and a serialiser writes them to flat files. It was the natural starting point because it validated the data model and the query logic with minimal rendering complexity. The PDF and web view were added once the data layer was stable, reusing the same computed statistics through a view mapper that formats them for presentation. Figure 5.9 shows the resulting data flow.

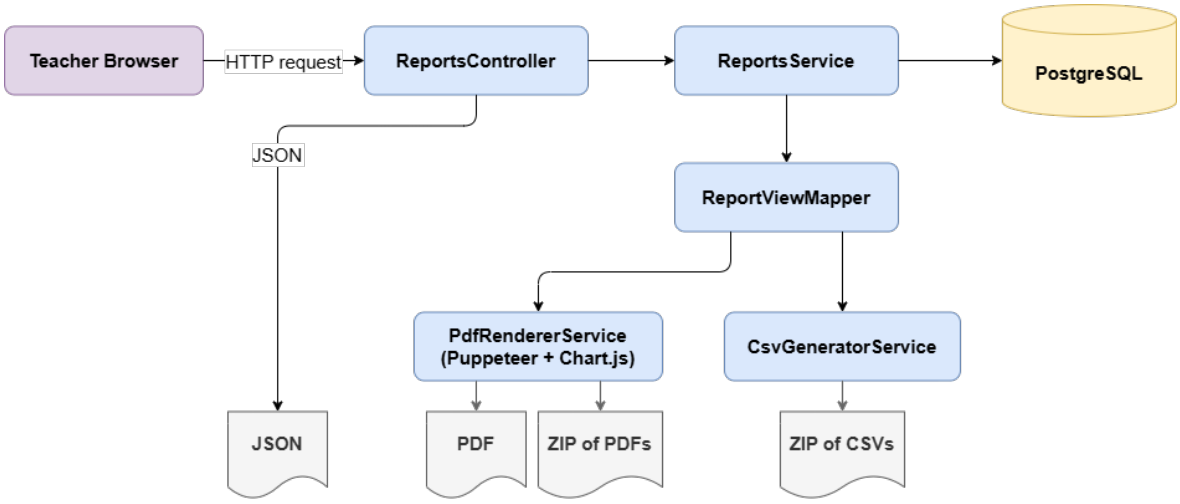


Figure 5.9: Internal data flow of the reports module. The computation layer queries the database and builds statistics; a view mapper transforms them into a presentation-ready model consumed by the PDF and CSV renderers; a controller streams the final response.

5.5.2 Separating computation from presentation

The pipeline involves five collaborating classes, each with a distinct responsibility. The core question in designing this pipeline was where to draw the boundary between data computation and data presentation. Mixing them, having the service format percentages as strings or resolve option UUIDs to display text, would couple the statistical logic to the rendering format and make it harder to reuse the same data for different outputs.

The solution was a strict layering: a computation layer that calculates statistics without any knowledge of how they will be displayed, a transformation layer that converts those raw structures into a presentation-ready format, and two independent rendering layers that consume the transformed data to produce their respective outputs [Fow12]. Each layer can evolve independently. Adding a new output format requires only a new rendering layer, without touching the computation or transformation logic.

Pipeline implementation

`ReportsService` is the computational core. It loads the event from the database with all associated entities using MikroORM eager population, then builds a unified participant map that resolves the two participant kinds in the system: web participants identified by email, and physical clicker participants identified by device ID and coordinator UUID. Only participants who submitted at least one answer are included. From this map it computes three sets of statistics: event-level aggregates, per-participant statistics, and per-question statistics including full response distributions. It also generates the presenter notes by scanning the statistics for threshold violations and positive outcomes, classifying each observation as `danger`, `warning`, `info`, or `success`.

`ReportViewMapper` transforms the raw statistics into a `ReportViewModel`, a serialisable data transfer object with fields formatted as strings, percentages rounded to two decimal places, times expressed in seconds, and option UUIDs resolved to their display text.

`PdfRendererService` takes the `ReportViewModel` and a list of requested sections and produces a PDF buffer using Puppeteer. This approach, rendering HTML and CSS with a headless browser rather than constructing the document with a low-level PDF library, was chosen because the report layout includes tables, colour-coded cards, and embedded bar charts generated with `chartjs-node-canvas`. Replicating that layout with an API like pdfkit would have required rebuilding the design logic from scratch; Puppeteer renders the same HTML used in the web view and produces a faithful, print-ready document.

`CsvGeneratorService` has no external dependencies. It serialises the `EventReportData` structure into four flat CSV strings (event metadata, general statistics, per-participant data, and per-question analysis) assembled in memory and streamed as a ZIP via `archiver` in the controller, without buffering the full archive.

`ReportsController` exposes five HTTP endpoints, all protected by `adminAuthMiddleware`: CSV ZIP download, PDF generation with optional per-section splitting, individual participant report, paginated participant list, and presenter notes.

5.5.3 Web view and parallel data loading

The reports web view presents two independent datasets to the host: the participant list and the presenter notes. These have no dependency on each other and neither dataset needs the other to be loaded before it can be displayed. Loading them sequentially would introduce unnecessary latency, since the host would wait for the first request to complete before the second even started.

The solution was to issue both requests in parallel and let each section of the UI populate independently as its data arrives. This keeps the interface responsive, since if one endpoint is slower than the other, the faster section is already usable while the other is still loading.

Web view implementation

The reports page is rendered by a dedicated view-controller that loads the event list and passes it to `reports.ejs`. When the host selects an event, the client script fires two fetch calls simultaneously using `Promise.all`, specifically one to the participants endpoint and one to the presenter notes endpoint. Each accordion section populates independently as its response arrives.

The participant section renders a searchable and sortable list of cards, one per participant, with name, score, ranking, correct answer count, and average response time. The presenter notes section renders a prioritised list of colour-coded observations: red for `danger`, amber for `warning`, blue for `info`, and green for `success`. Download buttons fixed to the bottom of the page trigger the CSV ZIP and PDF exports. Figure 5.10 shows the first design for this interface.

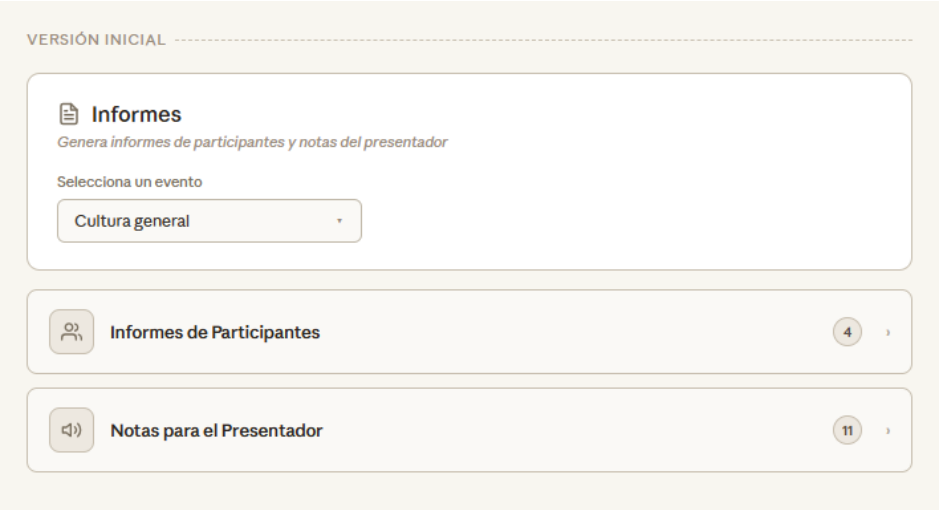


Figure 5.10: First design for the Reports web view, developed during early stages.

5.5.4 Calculated metrics

The module computes statistics at three scopes. **Event-level metrics** summarise session health: engagement, participation, and general difficulty. **Participant-level metrics** support grading, feedback, and detection of struggling attendees. **Question-level metrics** highlight confusing or problematic questions for future revision. All three are derived from the same database query and reused across the CSV, PDF, and web outputs. Table 5.1 summarises the full set of metrics by scope.

Scope	Metric	Description
Event	Participants (Total / Active)	All attendees vs. subset with ≥ 1 recorded answer
Event	Total questions	Number of questions in the event
Event	Overall success rate	Correct answers / total answers across all participants (%)
Event	Average score	Mean score across all active participants
Event	Average response time	Mean `timeToAnswer` across all answers
Event	Participation rate	Actual answers / possible answers (%)
Event	Clickers (Registered / Active)	Assigned physical devices vs. those with ≥ 1 answer
Participant	Correct / incorrect answers	Count of `isCorrect` flags
Participant	Success rate	Correct / total answered (%)
Participant	Average response time	Mean `timeToAnswer` across their answers
Participant	Participation rate	Questions answered / total questions (%)
Participant	Ranking	Position by score after sorting all active participants
Participant	Performance flags	Highlights outliers (success $< 30\%$ or $\geq 80\%$, time > 18 s)
Participant	Per-question breakdown	Answer text, correctness, and response time for each question
Question	Total / correct responses	Aggregate answer counts
Question	Success rate	Correct / total (%)
Question	Average response time	Mean `timeToAnswer` for the question
Question	Response distribution	Count and percentage per chosen option
Question	Quality flags	Highlights outliers (success $< 40\%$ or $\geq 90\%$, time > 15 s)
Question type	Success rate by type	Per-type aggregate of `ONE_IN_FOUR` or `HANGMAN`

Table 5.1: Metrics computed by the reports module, grouped by reporting scope. All metrics are derived from the same query and reused across the CSV, PDF, and web view outputs.

The presenter notes are generated by scanning these metrics for threshold violations and positive outcomes. Questions with a success rate below 40% generate a `danger` observation; those whose average response time exceeds 15 seconds generate a `warning`. Participants with fewer than 30% correct answers generate individual or aggregate warnings depending on their count. Excellent results generate `success` notes. All observations are sorted by severity before being returned.

5.6 PowerPoint Add-in Integration

The PowerPoint add-in integration is the central contribution of this project. Before it existed, TapClassroom had no presence inside PowerPoint. Hosts used the web application both to project quiz content and to control the session from a browser window, running alongside the presentation in a separate application. Every interaction required switching context in front of an audience, which was disruptive and increased the risk of errors. Furious Koalas S.L. requested the development of a PowerPoint integration as the primary objective of this project, with the goal of embedding TapClassroom directly onto the slide surface so that quiz content and session controls remain accessible throughout the presentation without any secondary window.

Building this integration from scratch required solving a set of technical problems that only emerged progressively during development: choosing the right integration mode, adapting an interface that was never designed to run inside an iframe, and handling the hardware API restrictions imposed by PowerPoint Online.

5.6.1 Choosing the integration mode

Microsoft provides two integration modes for PowerPoint add-ins. A task pane add-in appears as a side panel next to the slide, while a content add-in is inserted directly onto the slide surface as an embedded object. The distinction matters in a presentation context: during slideshow mode, PowerPoint hides the task pane entirely, making it invisible to both the host and the audience.

This was not discovered from documentation alone. An early prototype was built using the task pane model and demonstrated to Furious Koalas S.L. in a working session. When the presentation mode was activated, the task pane disappeared and the interface became inaccessible. The company immediately identified this as a blocking issue. The content add-in model, which remains visible on the slide surface in both edit and presentation mode, was the only viable path forward and the basis on which the rest of the integration was built. The quiz content and session controls are always visible to the host during the presentation, with no mode transition or context switch required.

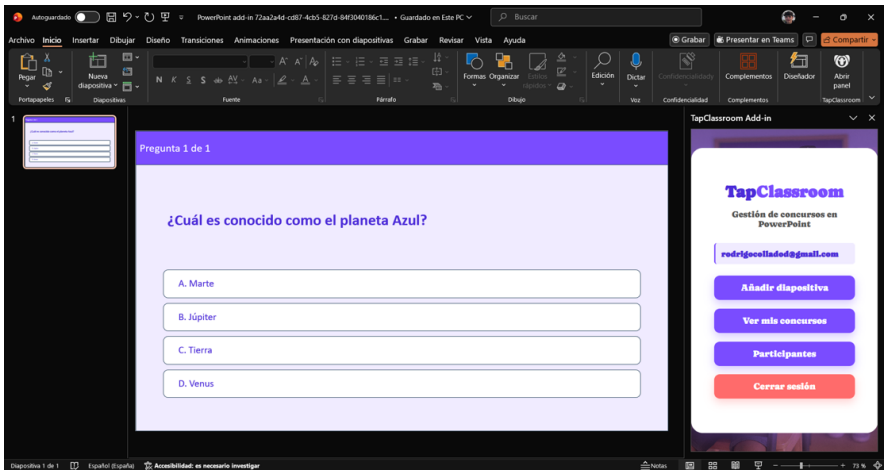


Figure 5.11: Early prototype using the task pane model. The interface appears as a side panel in edit mode, but PowerPoint hides it during presentation, making session control inaccessible. This motivated the switch to the content add-in model.

Manifest and execution model

An Office Add-in is a web application loaded inside an embedded iframe, described by an XML manifest declaring metadata, the URL to load (`SourceLocation`), permitted navigation domains (`AppDomains`), initial iframe dimensions, and the required permissions level; no store registration is needed for development or institutional distribution.

The API limits dimensions to 32–1000 pixels, a legacy of the 4:3 slide format, so for widescreen 16:9 slides the add-in is inserted at 1000x550 pixels and the host can extend it manually. On insertion, Office injects Office.js into the iframe and loads the `SourceLocation` URL, which the TapClassroom server handles as a normal HTTP request returning a complete HTML page with no separate client-side build. Figure 5.12 illustrates this flow.

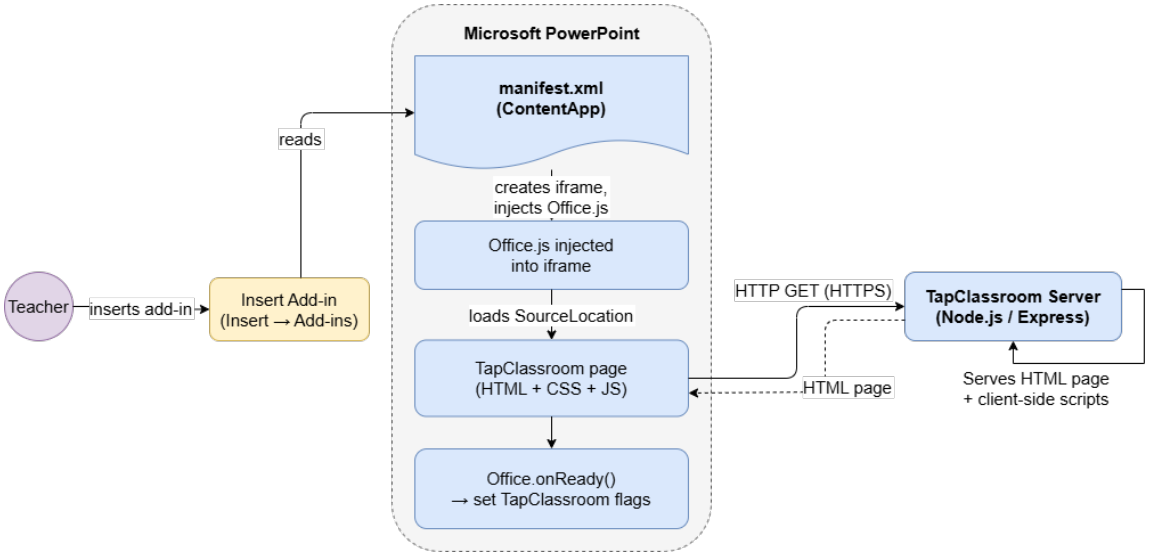


Figure 5.12: Content add-in execution model. Office reads the manifest, injects Office.js into the iframe, and loads the `SourceLocation` from the TapClassroom server. The returned HTML page calls `Office.onReady()` to confirm host context and activate add-in behaviour.

5.6.2 Adapting an interface not designed for an iframe

TapClassroom was built as a standalone web application. Running it inside an Office-hosted iframe exposed three categories of incompatibility that had to be resolved without modifying the existing server routes or breaking the desktop and mobile interfaces.

The first problem was navigation. Several views were designed to open auxiliary pages in separate browser windows. Inside an iframe, that behaviour either escapes the embedded context or is blocked entirely, producing a detached window the host would have to manage separately during a live session. Figure 5.13 shows this problem in practice.

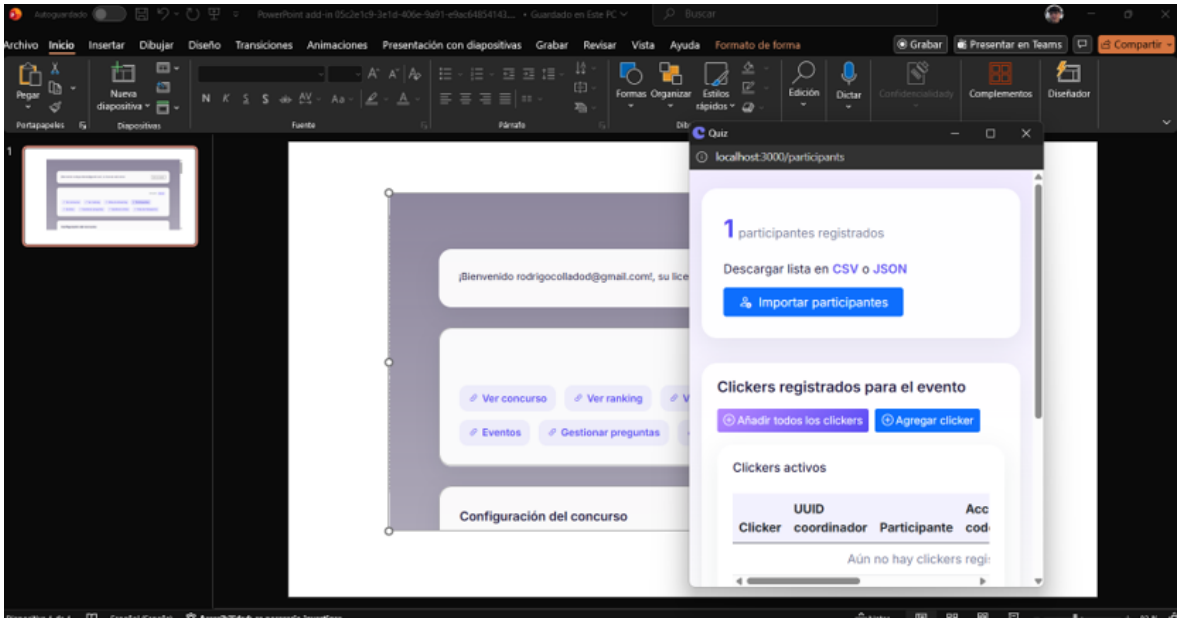


Figure 5.13: Participant management view opening as a floating browser window over PowerPoint. Because the add-in runs inside an iframe, navigation calls escape the embedded context and produce a detached window the host must manage separately.

The second problem was layout. The existing CSS was designed for a full browser viewport. Inside the constrained iframe, elements overflowed and content was cut off. Figure 5.14 shows this before the CSS was corrected.

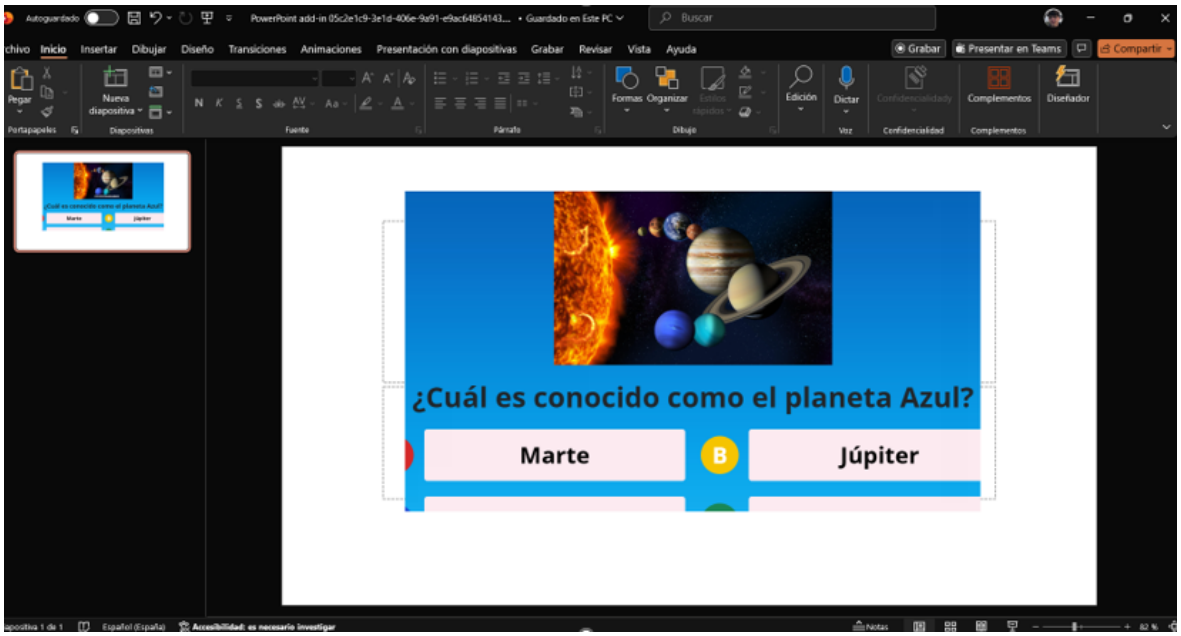


Figure 5.14: Layout and scaling issues inside the add-in iframe before CSS corrections. Fixed pixel dimensions cause elements to overflow and content to be cut off within the constrained iframe, resolved by replacing fixed sizes with relative units and flexible layouts.

The third problem was context detection. Without knowing whether the page was running inside PowerPoint or in a regular browser, it was impossible to apply the correct behaviour selectively. Detecting the context asynchronously also risked showing an incorrect layout to the host before detection completed.

The solution to all three problems was a suite of lightweight client-side scripts injected into every page, each addressing one category of incompatibility. No server routes were modified and no existing templates were duplicated. This means the add-in behaviour is entirely additive, where removing it would leave the desktop and mobile interfaces completely unaffected.

Adaptation scripts and server configuration

`load-office.js` is the foundation of the detection system. It calls `Office.onReady()`, the entry point for Office Add-in, waits for the Promise to resolve with the host and platform identifiers, and sets two boolean flags on a global `window.TapClassroom` object: `isInPowerPoint` and `isPowerPointOnline`. A 3-second fallback timer ensures that if Office.js never calls the callback, as happens in regular browsers, the flags default to `false` and normal operation continues. Once detection is complete, the script dispatches the DOM element `tapclassroom:office-ready` that all other add-in scripts wait for before activating.

To prevent a flash of unstyled content caused by this asynchronous detection, all pages apply `body { visibility: hidden }` by default. Once `load-office.js` completes detection it adds the class `loaded` to `<body>`, restoring visibility. This ensures the host never sees an intermediate layout before the correct mode is applied.

`addin-navigation.js` resolves the navigation problem by overriding `window.open()` with a function that redirects within the same iframe instead of opening a new window. It also intercepts `target="_blank"` link clicks at capture phase, removes the `target` attribute, and appends the `inAddin=true` URL parameter so that subsequent pages are also served in add-in mode. A `MutationObserver` applies the same transformation to links inserted dynamically after the initial page load.

`addin-conditional-display.js` handles the layout differences between the add-in and browser contexts using a CSS-class convention. Elements with the attribute `addin-hide-if-powerpoint` are hidden inside the add-in, and elements with `addin-show-if-powerpoint` are shown only inside the add-in. This avoids duplicating templates while allowing the two contexts to present different UI elements. The same audit that introduced this system identified the fixed-dimension CSS issues; these were resolved by replacing fixed-size rules with relative units and flexible layouts.

Three server-side changes were also required. The Content Security Policy was updated to allow Office domains (`*.office.com`, `*.officeapps.live.com`, `*.sharepoint.com`, `*.live.com`, `*.microsoftonline.com`) in the `frame-ancestors` directive, without which the browser refuses to embed the page inside the Office iframe. The `Permissions-Policy` header was set to grant `screen-wake-lock` and `storage-access` to the iframe. The session cookie was configured with `SameSite=None; Secure` to allow authentication to persist inside a cross-origin iframe, which is required for PowerPoint Online.

5.6.3 PowerPoint Online hardware constraints

PowerPoint Online is the browser-based version of PowerPoint available through Microsoft 365, widely used in educational institutions. It was identified as a primary deployment target, but it imposes hardware API restrictions that the desktop application does not.

Two capabilities required by TapClassroom (communicating with physical clicker devices via Web Serial, and preventing the screen from sleeping via Screen Wake Lock) both require explicit permission delegation from the parent iframe. PowerPoint Online does not grant these permissions to embedded add-ins, which means they are blocked unconditionally regardless of what the add-in requests. This is a deliberate architectural constraint of the platform, consistent with minimising the permissions granted to embedded third-party content in a SaaS environment.

The same restriction affects other Microsoft-validated add-ins. Mentimeter, a content add-in with similar functionality to TapClassroom, explicitly warns users that presentation mode is unavailable in the browser version for the same reason. Figure 5.15 shows this warning.

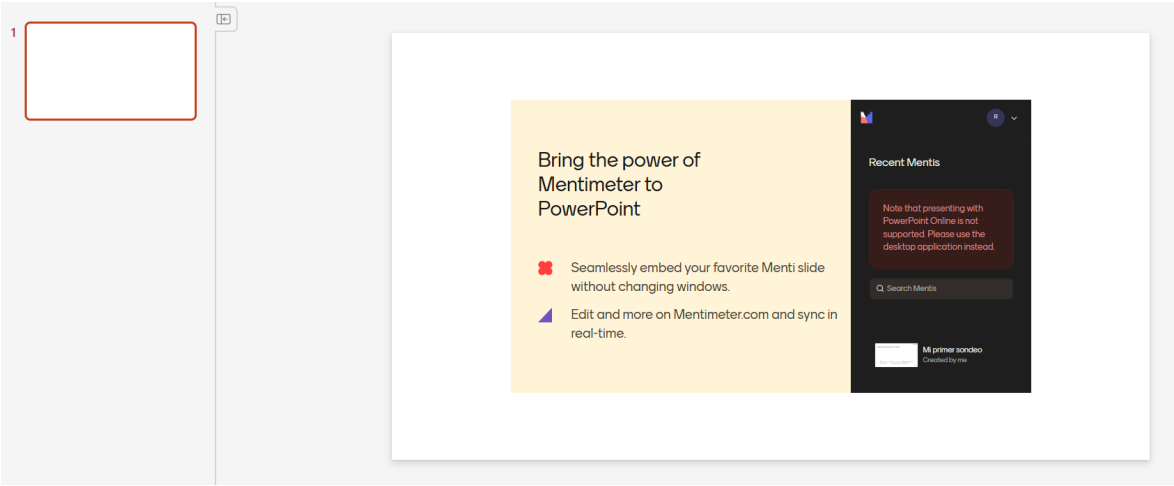


Figure 5.15: Mentimeter, a Microsoft-validated content add-in with similar functionality to TapClassroom, explicitly warns users that presentation mode is unavailable in PowerPoint Online due to hardware API restrictions imposed by the platform's iframe permission policy.

The consequence for TapClassroom is that the clicker bridge functions only in PowerPoint Desktop. In PowerPoint Online the session can be fully controlled and participants can join from their devices, but physical clicker hardware is unavailable. This is an architectural ceiling imposed by the host platform, not a limitation of the add-in itself.

For live classroom sessions, PowerPoint Desktop is therefore the recommended deployment environment, as it supports physical clicker devices and allows the add-in to be expanded to fill the slide surface for projection. PowerPoint Online remains a valid option for editing and configuring sessions, or for environments where participants join exclusively from their own devices and projection is not required.

Permission policy inspection

To confirm that the required permissions were absent, the `allow` attribute of the iframe embedding the add-in was inspected directly in Chrome DevTools. The attribute grants permissions for `autoplay`, `microphone`, `camera`, `clipboard-read`, `clipboard-write`, `display-capture`, and `local-network-access`, but does not include `serial` or `screen-wake-lock`. Under the W3C Permissions Policy specification, a capability absent from the `allow` attribute of a cross-origin iframe is blocked unconditionally, regardless of what the embedded document requests.

5.6.4 Deployment and distribution

The Office Add-in API requires all manifest URLs to be served over HTTPS. During development this creates a mismatch. The local server runs on plain HTTP, which PowerPoint Desktop tolerates but PowerPoint Online rejects. A secure public endpoint was therefore needed to test the add-in in PowerPoint Online without deploying to production. In production, HTTPS is satisfied natively through a reverse proxy with a valid TLS certificate.

Distribution introduces a separate challenge. Publishing through the official Microsoft AppSource store requires a paid verified publisher account, which was not available for this project. The alternative is sideloading, delivering the manifest file directly to the end user. This avoids the store entirely but requires the user to perform a manual installation process that varies by platform and is unfamiliar to most end users. For PowerPoint Desktop on Windows, the process involves several steps through Windows settings including registry edits and folder configuration. Left manual, this would be a barrier to adoption for non-technical users. A scripted installer was therefore developed to automate the process entirely. The remaining design challenge was that Windows Defender and corporate firewalls often block unsigned `.bat` scripts, which would prevent the installer from running on the target machines. Wrapping the script in a self-extracting executable was the solution, as it sidesteps the firewall warnings while keeping the installation experience to a single double-click.

Development tunnel and production infrastructure

During development, Cloudflare Tunnel and ngrok were used to expose the local HTTP server over an ephemeral HTTPS URL, enabling the add-in to be tested in PowerPoint Online without deploying to production. In production, the TapClassroom server runs on an AWS EC2 instance behind an Nginx reverse proxy with a valid TLS certificate; no tunnel is involved.

Distribution via sideloading

For PowerPoint Online, the host uploads the manifest via *Insert* > *Add-ins* > *Upload My Add-in*. Figure 5.16 shows this process. For PowerPoint Desktop on Windows, the mechanism requires registering a trusted network share catalogue in PowerPoint's Trust Center and placing the manifest in that folder.

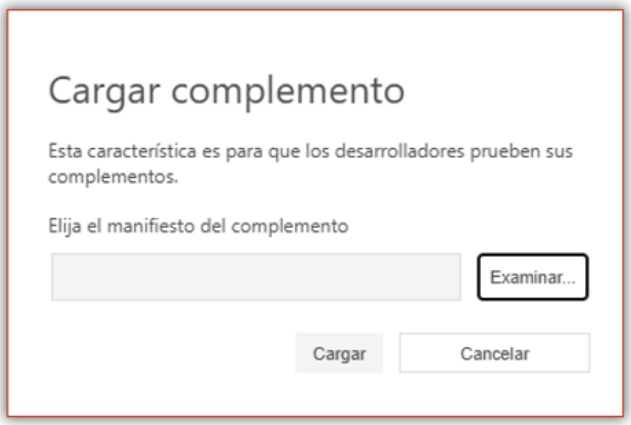


Figure 5.16: Sideloading the TapClassroom add-in Online. Using the *Upload My Add-in* option.

To automate the Desktop installation, the registry edits and folder setup were packaged into a batch script. To avoid firewall blocking of unsigned `.bat` files, the script and the manifest were wrapped together into a self-extracting `.exe` using the Windows built-in tool `iexpress`. This produces a single file the user can run without interacting with the command line, and without triggering the security warnings that an unsigned script would raise, as illustrated in Figure 5.17.

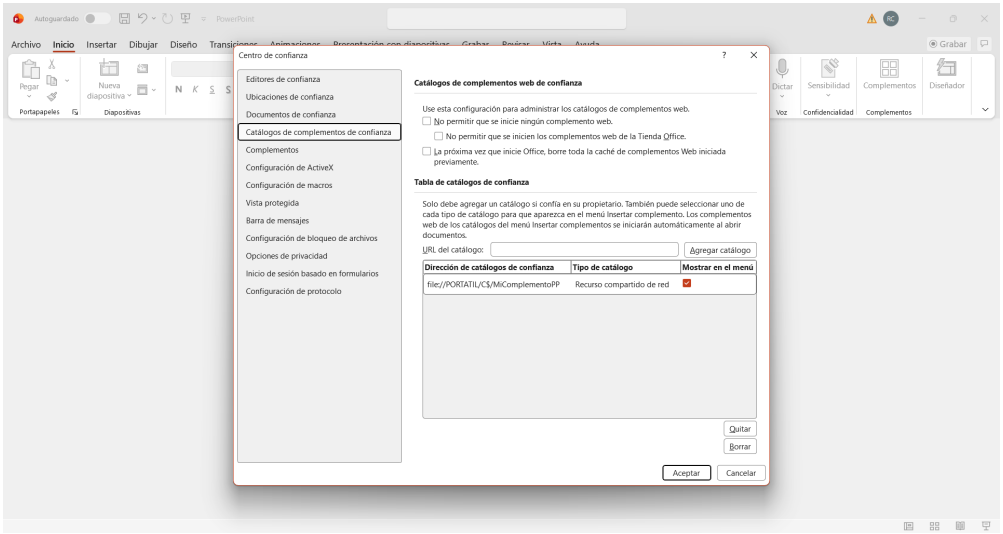


Figure 5.17: Sideloading the TapClassroom add-in for PowerPoint desktop. The manifest XML file is loaded directly into PowerPoint without going through the Microsoft AppSource store, using the *Shared Folder* trusted catalogue tab in PowerPoint Desktop.

Chapter 6

Results

THIS chapter presents the results of the TapClassroom contributions developed in this project. By the end of development, the delivered scope included three integrated contributions over the existing TapClassroom platform: a **mobile interface** that provides wireless live session control for the host directly from a smartphone; a **PowerPoint integration** that embeds the ecosystem seamlessly into the presentation flow based on the content add-in model; and a **reports and feedback module** that enables deep post-session analysis, automated presenter observations, and data export.

These elements were not conceived as independent prototypes, but as connected components of the same operational sequence, from quiz execution and supervision to performance data interpretation. This holistic approach ensured modular additions did not fragment the user experience. The mobile interface shares the same authenticated host context as the platform, so live control and post-session review remain part of the same ecosystem. The PowerPoint add-in co-exists with these mobile controls, allowing presenters to choose their interaction channel in real-time. Figure 6.1 provides an overview of host-student interaction across all three contributions.

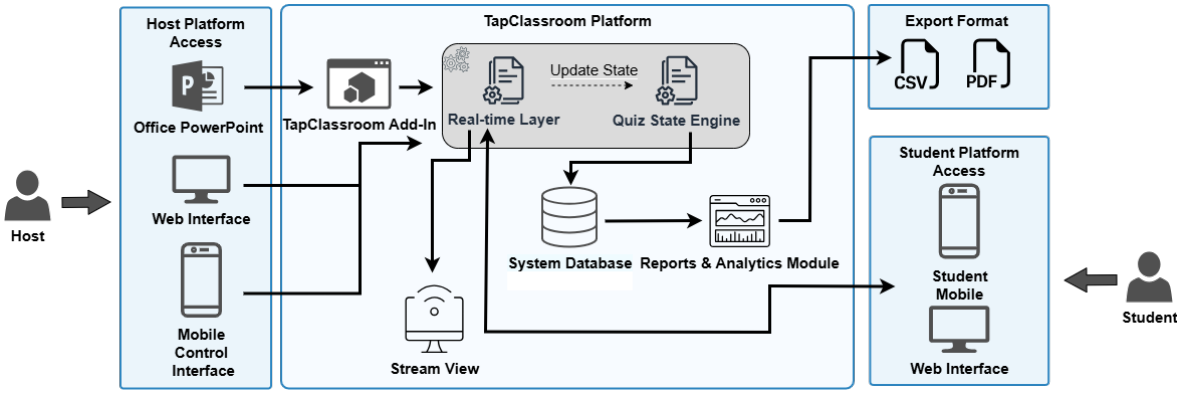


Figure 6.1: High-level overview of the TapClassroom platform. The host controls sessions via the PowerPoint Add-in, web interface, or mobile control interface; the platform core manages real-time state, persistence, and analytics; students participate through mobile or web clients.

That progression followed the nine-iteration structure established in Chapter 4, spanning from October to March across the planned 18-week period.

The first two iterations were dedicated to technology research and prototyping: the initial Office.js exploration and the content add-in architecture decision that would later support the PowerPoint integration were established here, fixing the architectural and integration foundations of the project before any user-facing module was committed.

Iterations 3 to 7 concentrated functional delivery, each targeting a specific interface, view, or module: the embedded stream view, the responsive navigation, and the in-slide control panel of the add-in were built progressively across iterations 3 to 5; the reports and feedback module was developed in iteration 6; and the mobile control interface was delivered in iteration 7.

The final two iterations were dedicated to deployment preparation and cross-environment refinement, finalising add-in distribution and PowerPoint Online compatibility and refining the mobile interface, so that the delivered contributions operated coherently in the target live-session context.

No significant re-planning of scope was required; the main adjustments occurred within iterations 3 and 5, where integration complexity between the content add-in boundary and the existing runtime views required additional coordination cycles before functional stability was reached. The detailed breakdown of each iteration is provided in Appendix B.

The chapter first presents the delivered functionality of each contribution: the mobile control interface, the PowerPoint add-in, and the reports and feedback module. Within each section, the subsections anchor the delivered functionality to the User Stories it fulfils and the iterations in which it was built, walking through it with reference to the corresponding interface captures. The final section then consolidates the project assessment, kept deliberately separate from the functional walkthrough so that the reader first gains the full picture of what was built. It combines the functional and end-to-end verification of the three contributions, which pairs manual real-device runs with reproducible browser-automated regression scripts, with the quantitative performance instrumentation, the code statistics, and the material and human resource costs.

6.1 Mobile Control Interface

The mobile interface gives the host full control of a live TapClassroom session from any smartphone browser, replacing the need to remain physically tied to the desktop computer. Figure 6.2 shows its four main states, which together span the full session lifecycle: from linking and live monitoring before the contest, through real-time control during play, to a clean closure at the end.

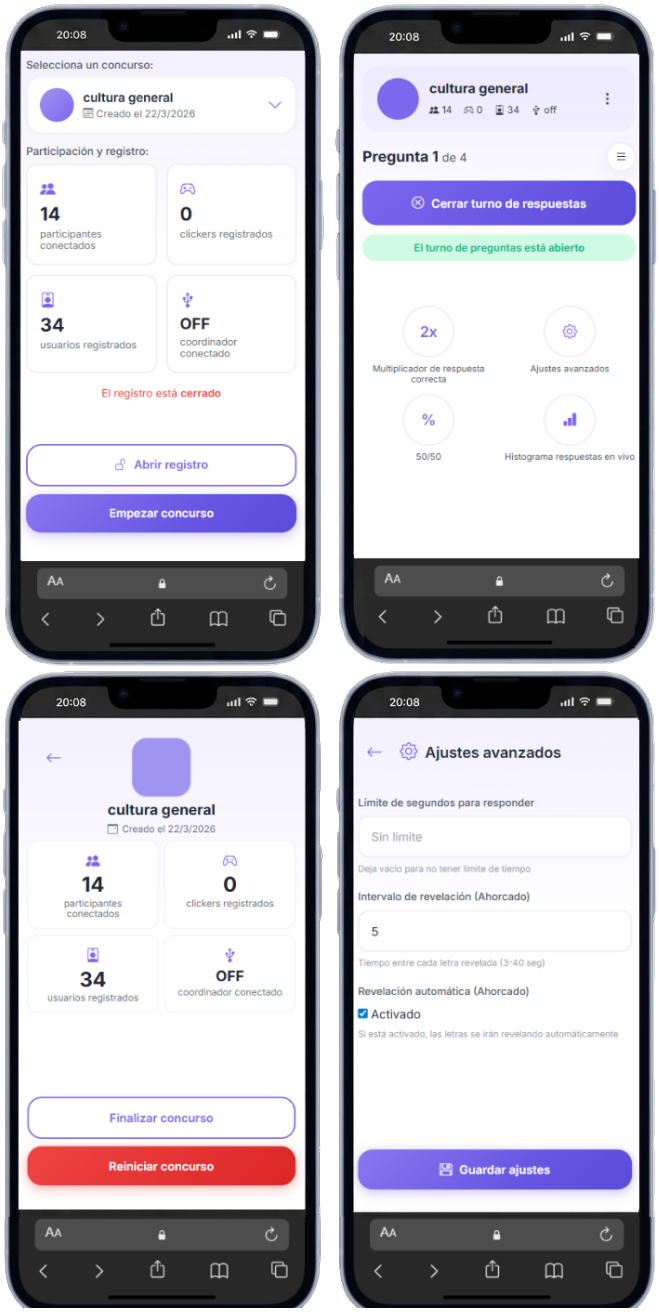


Figure 6.2: Mobile interface checkpoints: selection and monitoring (top-left), active question control (top-right), advanced settings (bottom-left), and session closure (bottom-right).

6.1.1 Session linking and live monitoring

Built in Iteration 7 to satisfy US04 (remote access and linking), the selection and monitoring view is the host's entry point into a live session. During the initial registration phase, the host monitors incoming connections directly from this view. As participants join via the web application or physical clicker devices, the interface updates the active connection counters in real-time. This immediate visual feedback lets the host decide exactly when to close registration and launch the first question, remaining entirely untethered from the main presentation computer.

6.1.2 Live control and session closure

Once participants are connected, the control state becomes the primary interaction hub. It brings together question-cycle control (US05) and event management (US06), both built in Iteration 7 and refined in Iteration 9. High-frequency actions such as advancing questions, toggling the poll status, and activating score multipliers or wildcards stay within immediate reach, and triggering a visual clue or displaying the leaderboard reflects instantly on the projected stream view. This degree of responsiveness is essential for maintaining the pacing of a gamified classroom activity, as any noticeable lag between the host's input and the visual output could disrupt the engagement of the participants.

The same control state also governs the closure sequence, providing a clean exit from the active session. When the final question concludes, the interface presents a deliberate confirmation step to prevent accidental terminations; upon confirming, the session state is finalised in the database and the mobile view transitions to a summary screen that marks the end of the live control flow. The result is a cohesive lifecycle that spans from preparation to closure without breaking the mobile user experience.

6.2 PowerPoint Add-in

The PowerPoint integration embeds the TapClassroom ecosystem directly into the presentation surface, so that the host can run a live session without leaving the slide canvas. It is the contribution that spans the longest arc of the project, evolving from early technology exploration into a workflow delivered across both target hosts.

6.2.1 Embedded execution in the slide canvas

The embedded execution model is the foundation of the integration and addresses native embedding (US07). Its groundwork was laid by the Office.js exploration and the content add-in architecture decision taken in Iteration 1 and Iteration 2, and realised as the in-frame stream view in Iteration 3. The add-in is inserted into the slide canvas and loads its page in-frame, preserving authenticated operation across classroom-like sequences. Figure 6.3 shows the loaded add-in in the PowerPoint Desktop environment, while Figure 6.4 shows the equivalent result in PowerPoint Online.

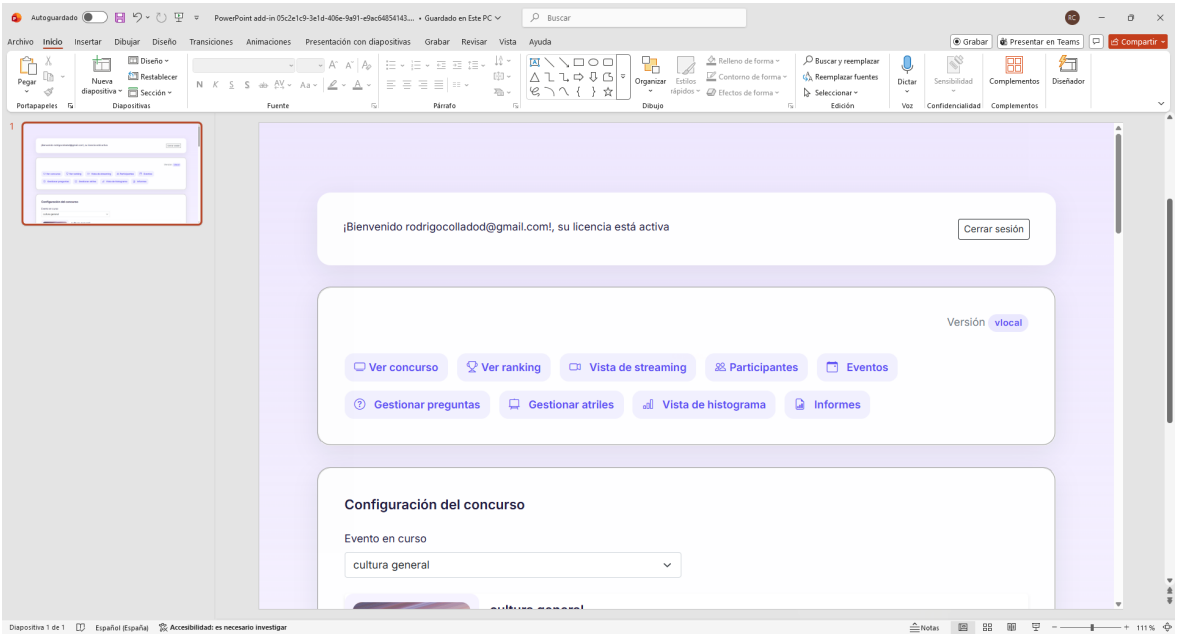


Figure 6.3: Add-in loaded in PowerPoint Desktop.

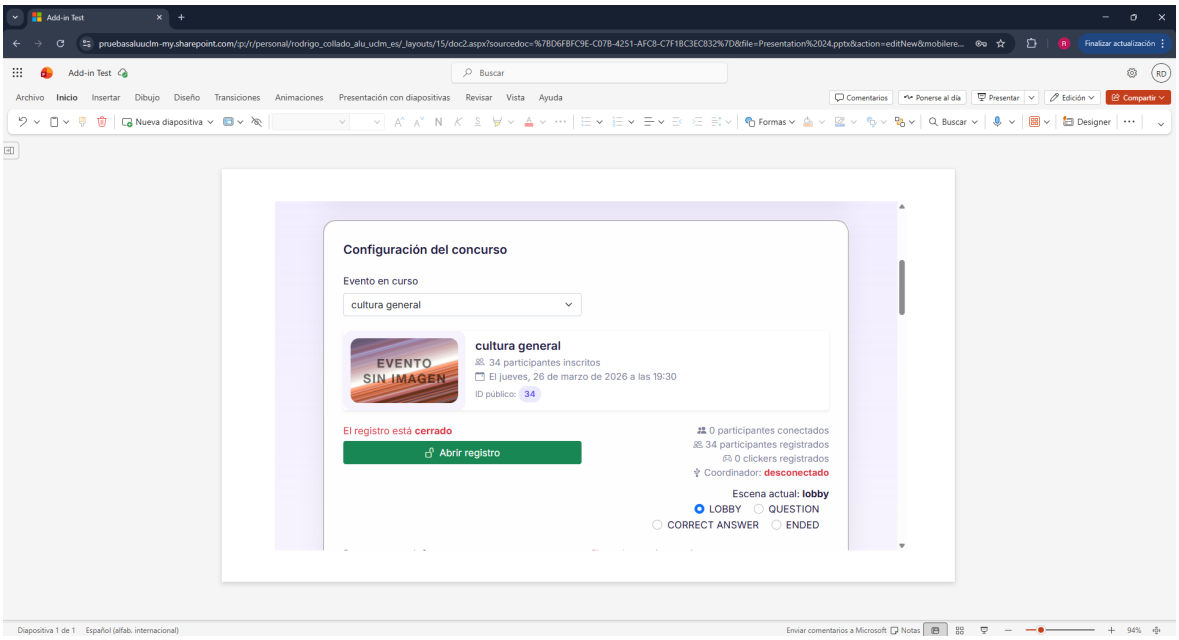


Figure 6.4: Add-in loaded in PowerPoint Online.

6.2.2 In-slide control and cross-host delivery

Building on that foundation, the in-slide control panel developed in Iteration 4 and Iteration 5 delivers slide operability (US08) together with immediate reflection and session isolation (US10). From the embedded panel, the host performs event context selection, control-state changes, and the rest of the host-facing actions, all of which remain available during slideshow execution within the same presentation surface, as shown in Figure 6.5.



Figure 6.5: Desktop slideshow with the in-slide control panel and integrated action buttons available in the same presentation surface.

Distribution and Online compatibility, finalised in Iteration 8 and Iteration 9, complete the requirement of stability across Office environments (US09). The delivered workflow operates across both target hosts in which classroom use was expected, although a small set of capabilities depend on browser features that PowerPoint Online restricts inside the embedded frame, which led to replacing popup-dependent navigation with an iframe-safe path. Table 6.1 summarises the delivered status of each capability per host.

Capability	**Desktop**	**Online**	**Delivered status**
Embedded add-in execution in slide canvas	Yes	Yes	Delivered in both hosts
In-frame control-panel interaction path	Yes	Yes	Delivered in both hosts
Popup-dependent navigation behaviour	Adapted	Restricted	Replaced with iframe-safe path
Web Serial clicker bridge	Yes	No	Desktop-only scope
Screen wake lock in embedded context	Yes	No	Desktop-only scope

Table 6.1: Simplified compatibility matrix for final host delimitation.

6.3 Reports and Feedback Module

Post-session evidence is most useful when explored and exported quickly. Once a contest ends, selecting it opens a view that consolidates participant analytics, presenter observations, and exportable evidence in one place.

6.3.1 Participant analytics and presenter notes

Developed in Iteration 6, the on-screen analytics deliver performance analysis (US01). They present a ranking with per-user summary metrics, together with a navigable accordion whose participant-detail modals expose the per-row data for each participant, as shown in Figure 6.6.



Figure 6.6: Participant performance report with scores and accuracy rates. For example, Rodrigo answered 56% of questions correctly (5/9), earning 42.0 pts.

Alongside the raw analytics, and covering difficulty detection and alerts (US02), the module generates a presenter notes panel that automatically surfaces difficulty alerts and teaching recommendations derived from the session results, giving the host an interpreted view of where participants struggled. A brief example is shown in Figure 6.7.

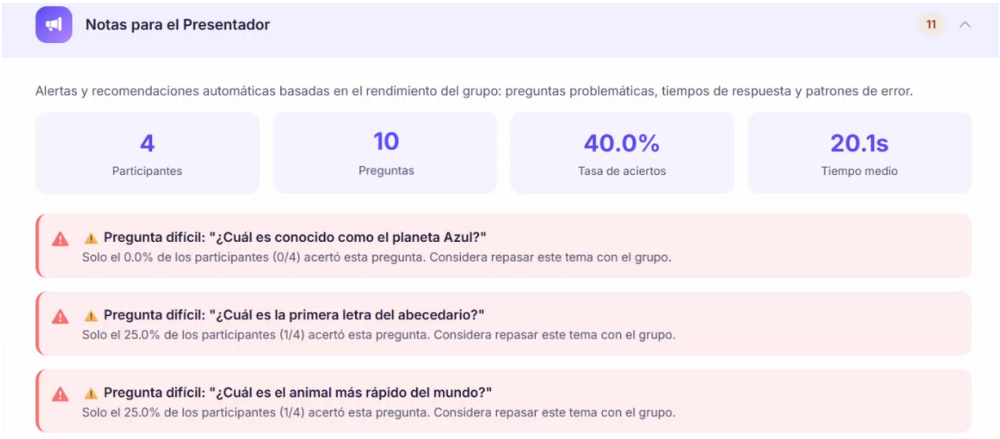


Figure 6.7: Presenter notes panel with automatically generated alert about question difficulty.

6.3.2 Evidence export

Also delivered in Iteration 6, the export feature fulfils evidence export (US03). The same event run can be exported in two complementary formats, a CSV output as machine-readable evidence for external processing, and a PDF report as presentation-oriented evidence for teaching review, including per-question analysis and response-distribution charts. Figure 6.8 shows a generated PDF report page.

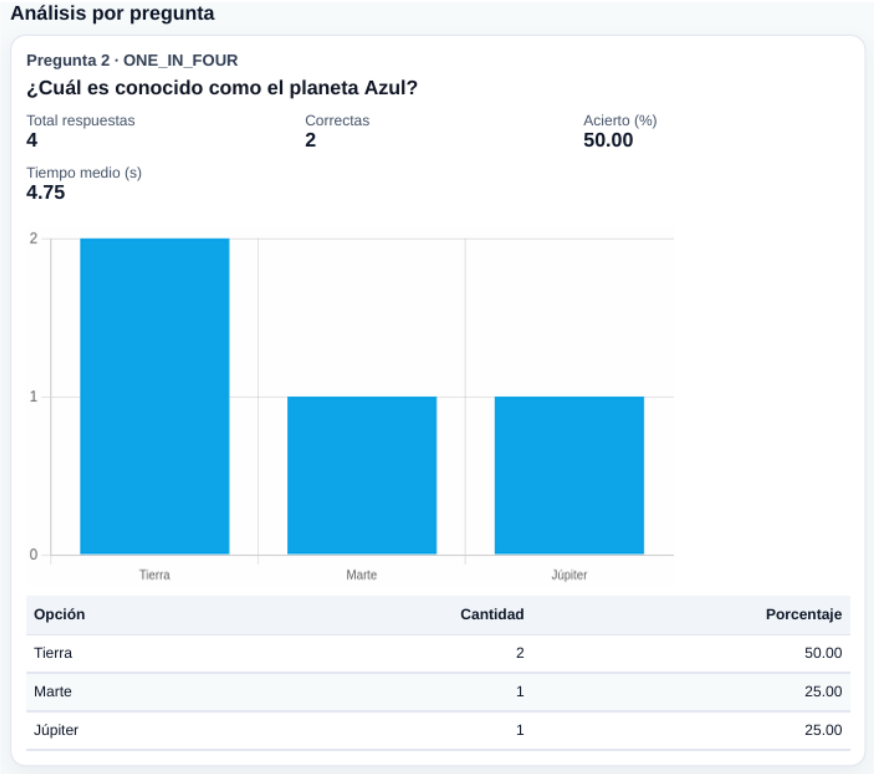


Figure 6.8: Generated PDF report page with per-question analysis and response-distribution chart.

6.4 Testing, Verification, and Project Assessment

This section consolidates the project assessment. It first reports the functional and end-to-end verification performed on the three contributions, then the quantitative performance instrumentation, and finally the code statistics and the material and human resource costs of the project.

6.4.1 Functional and end-to-end verification

Each contribution was verified as a complete operational sequence rather than as a set of isolated UI checks, combining manual real-device runs with browser-automated End-to-End (E2E) regression scripts.

Mobile control flow

The mobile flow was exercised as a continuous state machine traversing selection, control, settings, and closure, confirming that controls remained reachable and session context was preserved across transitions. This is critical in a live classroom environment, where any forced page reload or loss of context could severely disrupt the presenter's delivery; for example, a host can navigate into advanced settings to review the current configuration and immediately return to the active control panel without losing the session bindings or disconnecting from the underlying WebSocket channels. Table 6.2 summarises the outcomes for each checkpoint.

Checkpoint	**Observed outcome**
Selection and monitoring state	Session context, registration status, and live counters remained available at entry and after returning from other views.
Active question control state	Turn-control actions remained reachable and reflected the expected turn state across repeated runs.
Advanced settings state	The advanced-settings view remained reachable, and returning to control preserved the active session context.
Session closure state	End and restart actions remained available after full control sequences, preserving operational continuity.

Table 6.2: Condensed evidence matrix for mobile-flow checks under real-device operation.

In addition to real-device runs on an iPhone 14, the same operational path was exercised through an automated E2E walkthrough based on Puppeteer with user-agent emulation, used strictly as a reproducible regression check rather than as a substitute for the real-device observations. In the documented successful execution, the script entered through the selection view, selected the contest, opened registration, started the contest, closed the poll, advanced to the next question, entered and exited advanced settings, and finalised the session, reaching the transition screen without errors. Figure 6.9 provides a representative trace where a PATCH update sets `currentClientView="question"` and the stream view converges to question mode in the same sequence [1].

[1] A screen recording illustrating this synchronisation sequence in `https://youtu.be/S9rcjvAjBOM`.

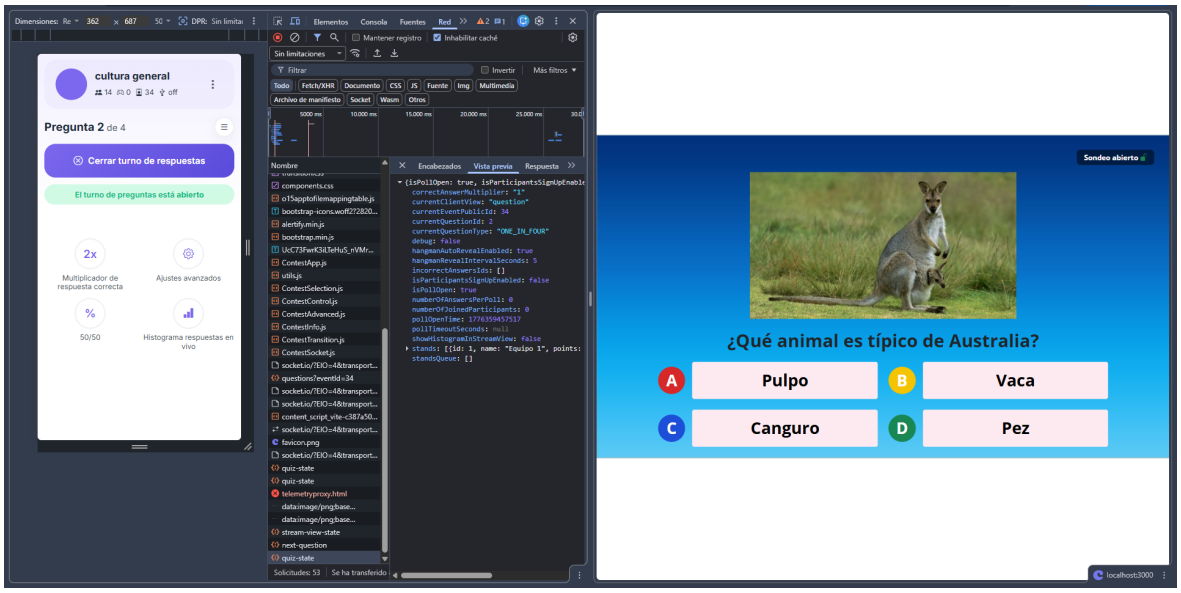


Figure 6.9: Synchronisation trace: mobile control action, PATCH payload inspection, and stream view rendered in question mode.

Across repeated cycles, propagation remained coherent: turn-state updates were reflected consistently, question navigation preserved session identity, and settings changes persisted after returning to control. The host can thus execute critical actions from one channel with fewer context switches, an operational observation rather than a formal usability study. Quantitative latency measurements are reported in Section 6.4.2.

PowerPoint add-in across hosts

In the documented desktop runs, the embedded path remained stable during repeated use: control actions were reachable from within the slide, context switches to external browser windows were avoided, and interface rendering remained readable under classroom projection conditions, delimiting Desktop as the fully operational target. PowerPoint Online checks confirmed that core embedded interaction is available and that the add-in loading path remains consistent with the desktop host, delimiting the Online target to the supported in-frame workflow captured in Table 6.1.

Reports module

Beyond the visual checks across output formats, the stability of the reports module was ensured through a dedicated browser-automated E2E test. Using Puppeteer, an operational script authenticated as a host, selected events from the UI, expanded the data layout accordions, and intercepted the network responses triggered by the export actions, verifying that the reports view renders correctly and that the CSV and PDF endpoints return valid `application/zip` and `application/pdf` MIME types respectively. Cross-format consistency was confirmed against three direct, observable criteria for the same event run:

- **Event identity:** The originating session ID and timestamp were consistent across both formats.
- **Participant scope:** Both outputs contained the exact same list of answering participants.
- **Per-question totals:** Aggregate response counts aligned across the PDF visual charts and the CSV raw columns.

In the documented checks, no inconsistencies were found between CSV and PDF outputs for the observed runs.

6.4.2 Control responsiveness and latency measurements

To quantify control responsiveness, command latencies were instrumented on the mobile client (iPhone 14). Action-triggering requests (HTTP `PATCH`/`POST`) to the `/api/quiz-state` endpoints were timed with the high-resolution Performance API. Figure 6.10 shows the Round-Trip Time (RTT) distribution across 25 samples.

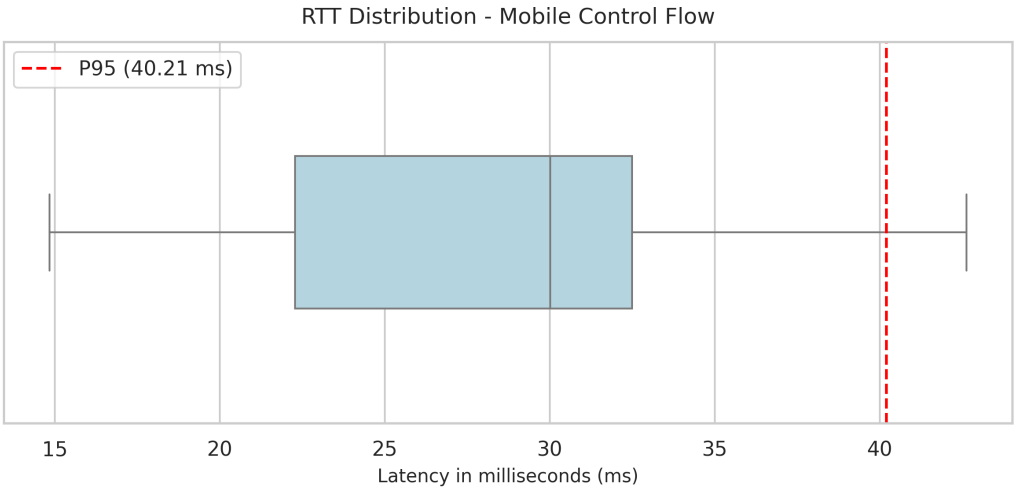


Figure 6.10: RTT (Round-Trip Time) distribution extracted from the Fetch API instrumentation acting as the state controller from the actual host device (iPhone 14). The 95th percentile (P95) boundary is highlighted.

These latencies, well below human-noticeable thresholds ($\sim$ 100 ms, median 30.02 ms, P95 40.21 ms), confirm that offloading control through REST operations lets the mobile client direct real-time views without overloading the WebSocket channels.

A complementary automated E2E run completed the full control path on the local deployment, methodologically distinct from the iPhone 14 runs. Table 6.3 reports the browser-side RTT per control action; all values stayed within the same low-latency band as the on-device measurements.

Control action	Endpoint	RTT (ms)
Contest selection	`PATCH /api/quiz-state`	15.48
Open registration	`PATCH /api/quiz-state/participants-signup`	12.13
Contest start	`PATCH /api/quiz-state`	23.83
Poll closure	`PATCH /api/quiz-state`	20.24
Next-question navigation	`POST /api/quiz-state/next-question`	32.22
Contest finalisation	`POST /api/control-panel/end-quiz`	22.42
Stream-view reset	`PATCH /api/stream-view-state`	18.88

Table 6.3: Browser-side RTT per control action in the automated E2E walkthrough on the local TapClassroom deployment.

6.4.3 Code Statistics

The contributed code was measured with `cloc`[2] as the diff against the pre-existing baseline. Table 6.4 reports the lines of code added per language, totalling 8 845 lines (8 221 excluding documentation).

Language	Files	Code
JavaScript	15	3 214
TypeScript	4	1 322
CSS	9	2 922
EJS	2	763
Markdown (docs)	1	624
Total	**31**	**8 845**

Table 6.4: Lines of code added per language, measured with `cloc`.

6.4.4 Project Costs and Resources

The project's cost is essentially human, expressed as a *company cost* (the full hourly cost to the employer, not the net or gross salary). For a junior developer in Spain, a representative gross salary of 25 000 €/year[3] plus the 2026 employer social-security contributions (≈30.65% of the gross base)[4] gives an annual company cost of about 32 660 €, or roughly 18.15 €/h over an effective 1 800-hour year. The 400+ hours of development therefore amount to approximately 7 260 €, the dominant cost of the project. Material and licensing costs were negligible: the stack relied entirely on open-source and free-tier tools, and the hardware used (an iPhone 14 for real-device testing and a development laptop) was already available at no additional cost, keeping the overall project budget notably lean.

[2] `https://github.com/AlDanial/cloc`
[3] `https://desarrollosoftware.es/salarios-desarrolladores-espana-2026/`
[4] `https://gsmb.es/cotizacion-a-la-seguridad-social/`

Chapter 7

Conclusions

In this chapter, the architecture of the proposed solution for TapClassroom is assessed from a reflective perspective. First, the extent to which the initial objectives have been fulfilled is evaluated. Next, the competences acquired during the process are analysed. The text then moves on to a personal reflection, in which the challenges faced and the lessons learnt are highlighted. Finally, it proposes several directions for future work, emphasising how this project lays the foundations for the next evolution of the application.

7.1 Reached objectives

The work carried out during this project has successfully met the initial objectives at both the general and the specific level. Beyond the delivered functionality, the main achievement lies in having designed, developed, and validated three interrelated subsystems, a mobile control interface, a reports and feedback module, and a PowerPoint add-in, and in having integrated them coherently into a platform that was still under active development inside a company. This required working across a wide range of technical fronts, from real-time web communication and responsive single-page interfaces to the Office.js add-in model, server-side data aggregation, and document generation, while respecting the constraints of an existing, production-bound codebase that could not be destabilised.

At the general level, the three subsystems were proven to operate as a cohesive whole rather than as isolated features. They share the same authenticated host context, preserve the server as the single source of truth, and keep the real-time behaviour of the platform intact, so that live control, presentation embedding, and post-session analysis form a single continuous workflow.

Regarding the specific objectives, their degree of achievement is discussed below.

- **To facilitate system usability and mobility for presenters**: This objective was accomplished through a lightweight mobile web interface that frees the host from the main classroom computer, allowing questions, participant monitoring, poll control, and session pacing to be managed with full freedom of movement.

- **To provide actionable post-session feedback**: The reports module aggregates session data into a clear, navigable interface, surfaces question-level difficulty indicators and participant summaries, and produces standardised CSV and PDF exports for academic grading, administrative records, and teaching review.
- **To integrate seamlessly with presentation flows**: The PowerPoint add-in embeds the classroom controls directly into the slide canvas, keeping the quiz, the projected content, and the control experience within the same delivery environment across both Desktop and Online execution contexts, within their respective capabilities.
- **To ensure architectural compatibility and real-time synchronisation**: The new subsystems operate harmoniously with the existing Node.js, Express, and PostgreSQL architecture, preserving the server as the single source of truth and maintaining immediate state synchronisation across all connected clients through WebSockets, without fragmenting the platform or introducing inconsistent session states.
- **To validate the system in a real-world operational context**: The subsystems were exercised under realistic live-session conditions, combining real-device operational runs with automated end-to-end testing to confirm control responsiveness, integration stability, and correct behaviour across the main user workflows.

7.2 Addressed competences

This section focuses on the specific competences developed throughout the project, based on the guidelines of the Degree in Computer Engineering, in the Software Engineering specialisation.

- **[IS1] Ability to develop, maintain, and evaluate software services and systems that meet user requirements, behave reliably and efficiently, are affordable to maintain, and comply with quality standards**: This competence was developed through the implementation of a complete software solution, covering everything from identifying requirements to architectural design and testing. The architecture was structured to ensure reliability and maintainability, using modern web technologies like TypeScript and Express. Thorough testing and validation processes, including automated E2E checks with Puppeteer, ensured the system met the established quality standards and behaved efficiently under real operational conditions.
- **[IS3] Ability to solve integration problems according to available strategies, standards, and technologies**: One of the main technical challenges was integrating the new subsystems with the existing TapClassroom platform. The solution required a deep understanding of web integration standards, particularly when embedding the PowerPoint add-in and managing cross-origin communication. Furthermore, the synchronisation of the mobile interface with the main server via REST APIs and WebSockets demanded robust integration strategies to ensure real-time consistency.

- **[IS4] Ability to identify and analyse problems and design, develop, implement, verify and document software solutions on the basis of adequate knowledge of current theories, models and techniques**: This competence was exercised throughout the entire project lifecycle. The initial limitations of the platform were analysed, leading to the design of specific subsystems (reports, mobile interface, and PowerPoint add-in). These were then developed, implemented, and systematically verified through operational runs and latency instrumentation. The extensive documentation of the architecture, methodologies, and results in this document serves as a testament to the rigorous engineering approach applied.

7.3 Personal conclusion

This project has been a demanding and transformative stage of my undergraduate studies. One of the primary challenges was joining an already developed project maintained by Furious Koalas S.L., which required a deep effort to comprehend its existing layered architecture built on Node.js, Express, TypeScript, and Socket.IO, before any new functionality could be implemented. The challenge of building on this existing platform with new interfaces and integration boundaries, without breaking the core quiz-state API or the underlying data model, required a significant shift from theoretical knowledge to practical, applied engineering.

One of the most valuable aspects of this experience was learning to navigate the complexities of real-world software development under an agile methodology. Collaborating with the company's team through a Scrum-inspired framework allowed the project to pivot when necessary. A prime example was the early discovery that the initial PowerPoint task pane prototype would be hidden during a presentation, prompting a crucial architectural shift to a content add-in model. Dealing with these complexities of the Office.js environment, as well as the strict iframe permission policies in PowerPoint Online, forced me to make informed architectural decisions and troubleshoot unexpected behaviours independently.

Furthermore, developing the mobile single-page interface taught me how to maintain persistent real-time connections while managing client-side state, and building the reports module highlighted the importance of cleanly separating data computation from presentation pipelines to support multiple outputs. Having to reason about three subsystems at once, each with its own runtime constraints but sharing a single source of truth, gave me a much more holistic view of how the pieces of a live, distributed system fit together.

Although the three subsystems have not yet been deployed to a production environment, the process has been incredibly rewarding. It has transformed my approach to software development, helping me to think more holistically about user experience, system architecture, and code maintainability. Most importantly, the results of this project have proven the viability of these new concepts, paving the way for the next evolution of the TapClassroom application, building on these foundations rather than discarding them.

7.4 Future work

While the current implementation fulfils its intended scope, several areas for improvement and extension have been identified. The fact that this project has served as a foundation for a broader evolution of the platform opens up numerous possibilities for future development.

Potential architectural and functional improvements

- **Production deployment and live monitoring**: The immediate next step is the deployment of the subsystems to a production environment. This will require setting up robust logging and live monitoring tools to track the system's performance during actual classroom use, as well as gathering real usage data to guide subsequent priorities.
- **Evolution towards a commercial-grade experience**: Rather than rebuilding the platform from scratch, the natural next step is to pivot the current development towards a new iteration whose interface and user experience are aligned with those of established commercial products such as Kahoot. This entails modernising the frontend and the UI/UX layer and progressively decoupling the backend, while reusing the architecture and the three subsystems delivered here as the starting point.
- **Full host parity for the PowerPoint add-in**: A focused line of work is to close the remaining gap between Desktop and PowerPoint Online, finding iframe-safe alternatives to the capabilities currently restricted in the embedded Online frame, such as the Web Serial clicker bridge and the screen wake lock, so that both hosts offer an equivalent experience.
- **Enhanced analytics and recommendations**: The reports module could be extended with advanced analytics, providing deeper insights into student performance trends over time and automated, personalised recommendations to help presenters adapt their content for future sessions.
- **Broader device and accessibility coverage**: The mobile interface could be validated across a wider range of devices and screen sizes, and reviewed against accessibility and internationalisation guidelines, broadening the range of classrooms and presenters the platform can serve.

Appendices

Appendix A

A.1 User Stories

The key functionalities have been defined in the form of user stories, as set out in the following tables, which reflect the main functional objectives validated during the development and testing phases.

Id	**US01 - Performance Analysis**
Role	**As a** Presenter
Functionality	**I want** to visualize the global metrics of a finished session and break down individual student performance
Utility	**To** evaluate the general level of comprehension
Acceptance criteria	• The system displays summary metrics (e.g., total participants, overall correctness). • Performance can be viewed per individual student.

Id	**US02 - Difficulty Detection (Alerts)**
Role	**As a** Presenter
Functionality	**I want** to see automatic alerts or notes highlighting the questions with the highest failure rate
Utility	**To** quickly identify which concepts need to be reviewed
Acceptance criteria	• The system highlights questions with a high percentage of incorrect answers. • Insights are generated automatically from session data.

Id	US03 - Evidence Export
Role	**As a** Administrator or Presenter
Functionality	**I want** to download session reports in both raw data (CSV) and a formal formatted document (PDF)
Utility	**To** keep institutional records
Acceptance criteria	• Reports can be exported in CSV format. • Reports can be exported in PDF format. • Exported documents contain accurate session data.

Id	US04 - Remote Access and Linking
Role	**As a** Presenter
Functionality	**I want** to have a dedicated mobile control interface that connects instantly to the active session
Utility	**To** present and move freely around the classroom without being tied to the main computer
Acceptance criteria	• A standalone, responsive mobile interface is available for session control. • The mobile interface connects seamlessly to the active quiz session. • Session state is synchronized upon connection.

Id	US05 - Question Cycle Control
Role	**As a** Presenter
Functionality	**I want** to navigate through questions, open the voting phase, and reveal correct answers from my smartphone
Utility	**To** maintain absolute control of the class pacing
Acceptance criteria	• Controls are available to advance to the next question. • Voting phase can be started and stopped from the mobile device. • Correct answers can be revealed.

Id	US06 - Event Management
Role	**As a** Presenter
Functionality	**I want** to end the session or restart it completely from my mobile device
Utility	**To** leave the activity ready for a new group of students
Acceptance criteria	• The session can be formally closed from the mobile interface. • The session can be restarted, clearing previous state.

Id	US07 - Native Integration (Embedding)
Role	**As a** Presenter
Functionality	**I want** to insert the interactive TapClassroom quiz directly on my slide as a PowerPoint component
Utility	**To** avoid relying on external browser windows during the class
Acceptance criteria	• The TapClassroom interface runs inside a PowerPoint content add-in. • External browser windows are not required for normal operation.

Id	US08 - Slide Operability
Role	**As a** Presenter
Functionality	**I want** to operate the quiz smoothly using menus that visually adapt to the available size within my slide
Utility	**To** prevent covering other important information
Acceptance criteria	• The interface is responsive to the add-in iframe dimensions. • Core functionalities remain accessible regardless of the slide's aspect ratio.

Id	US09 - Stability in Office Environments
Role	**As a** Presenter
Functionality	**I want** to start my session in the add-in and have it remain secure and stable whether I project the presentation from the Office desktop application or from PowerPoint Online
Utility	**To** ensure reliability across different platforms
Acceptance criteria	• The add-in functions correctly in PowerPoint Desktop. • The add-in functions correctly in PowerPoint Online, handling iframe restrictions.

Id	US10 - Immediate Reflection and Isolation
Role	**As a** Presenter
Functionality	**I want** any action launched on my mobile (next question, show answers) to appear instantly on the PowerPoint projection for the students, and for this data to be completely isolated
Utility	**To** avoid crossing information with other teachers simultaneously teaching
Acceptance criteria	• Actions trigger immediate state updates across all connected clients via WebSockets. • Event data and WebSockets channels are strictly isolated per session.

Appendix B

B.1 Evolution through iterations

This appendix provides a detailed breakdown of the nine development iterations conducted between October 1 and March 31. The process followed a Scrum-inspired timeline consisting of 18 active weeks, mapped into 2-week iterations.

Iteration 1 – October 1 to October 14	
Main activity	System exploration and technology research. Initial setup of the development environment, exploration of the Office Add-in API (Office.js), and Webpack configuration for the PowerPoint add-in.
Related User Stories	US07
Obstacles and solutions	Initial unfamiliarity with the Office.js security sandbox and iframe boundaries. This was resolved by extensive review of Microsoft's documentation and building a baseline "hello world" add-in to validate the local dev-server.
Key outcomes	Foundational understanding of the Office Add-in lifecycle.

Table B.1: Summary of results – Iteration 1

Iteration 2 – October 15 to October 28	
Main activity	Initial prototype development using a Task Pane approach for the PowerPoint integration, attempting to control the session from a side panel.
Related User Stories	US07, US08
Obstacles and solutions	It was discovered that PowerPoint Task Panes are automatically hidden in Presentation Mode, completely breaking the intended use case of controlling a live session while presenting. The decision was made to discard the Task Pane and move to a Content Add-in model.
Key outcomes	The technical limitation of Task Panes was identified early. The architectural path shifted definitively toward embedding the platform directly into the slide surface as a Content Add-in.

Table B.2: Summary of results – Iteration 2

Iteration 3 – October 29 to November 11	
Main activity	Transition to the Content Add-in and streaming view. Embedding the TapClassroom interactive quiz directly onto the slide canvas and linking the existing WebSockets.
Related User Stories	US07, US10
Obstacles and solutions	Integration complexity between the content add-in boundary and the existing views required additional coordination cycles before functional stability was reached. Addressed by isolating the connection lifecycle and ensuring the add-in properly hooks into the slide-load events.
Key outcomes	Successful projection of the interactive quiz inside the slide. Real-time synchronization of participants and answering components operated smoothly within the embedded container.

Table B.3: Summary of results – Iteration 3

Iteration 4 – November 12 to November 25	
Main activity	Navigation, popups and responsive design. Adapting the existing web interface to fit varied slide aspect ratios and resolving constraints related to the iframe's limited viewport.
Related User Stories	US08
Obstacles and solutions	Standard navigation popups and modals were cut off by the iframe's strict boundaries. This was solved by rerouting popups into an internal overlay system that respects the content add-in's dimensions.
Key outcomes	A highly responsive UI behavior inside PowerPoint. Modals, dialogs, and core functions remained accessible and visually consistent regardless of the dimensions of the slide component.

Table B.4: Summary of results – Iteration 4

Iteration 5 – November 26 to December 9	
Main activity	Development of the integrated control panel inside the content add-in. Since Task Panes were discarded, it was necessary to embed session controls directly onto the slide canvas to ensure the presenter could manage the quiz flow without context switching.
Related User Stories	US08, US10
Obstacles and solutions	The desktop control view contained too many elements; it was necessary to simplify the interface and select only the essential controls for the presentation mode. The server had to be adapted to distinctively detect the PowerPoint environment to serve this specific panel. Furthermore, the panel had to be superimposed on the stream view without obstructing the quiz content, which required carefully finding the appropriate size and positioning.
Key outcomes	A cohesive hybrid view was established inside the slide, enabling the host to fully manage the session directly from PowerPoint without external tracking windows.

Table B.5: Summary of results – Iteration 5

Iteration 6 – December 10 to December 23	
Main activity	Reports view integration. Aggregating stored session data and building the post-session analytics module directly from the main server.
Related User Stories	US01, US02, US03
Obstacles and solutions	Structuring reliable, dynamically generated PDF and CSV exports without overly increasing backend complexity. Addressed by cleanly separating data access from the template-generation pipeline.
Key outcomes	Presenters gained the ability to automatically identify difficult questions, analyze participant performance, and export clean, formatted documents for local storage and grading.

Table B.6: Summary of results – Iteration 6

Iteration 7 – January 20 to February 2	
Main activity	Creation of the Mobile interface for session control. Developing a lightweight, single-page application focused solely on quick execution and session pacing from a smartphone, avoiding separate page navigation to preserve the WebSocket connection.
Related User Stories	US04, US05, US06
Obstacles and solutions	Preventing duplicate events or loss of socket connections due to rapid view switching and sleeping typical of mobile browsers. Fixed by implementing automated sync-checks upon returning to the tab.
Key outcomes	A robust, mobile-first control interface enabling the presenter to roam freely. It immediately reflected any action directly back into the live PowerPoint presentation.

Table B.7: Summary of results – Iteration 7

Iteration 8 – February 3 to February 16	
Main activity	Deployment and distribution. Researching and testing different sideloading techniques (network share vs local catalog) for Windows environments to allow clients to use the add-in without requiring it to be published on the Microsoft Partner Center.
Related User Stories	US09
Obstacles and solutions	Distributing the ‘.xml‘ manifest manually was too complex for final users. An automated installer was developed by wrapping a batch script (‘.bat‘) inside an executable (‘.exe‘) to configure the trust catalog automatically, alongside using Ngrok to test real remote connectivity.
Key outcomes	A valid and cost-effective distribution method was achieved. Users can now install and insert the TapClassroom add-in directly via a Shared Folder in PowerPoint without relying on the public Microsoft Store.

Table B.8: Summary of results – Iteration 8

Iteration 9 – February 17 to March 2	
Main activity	Online compatibility, refinement, and system consolidation. Testing across PowerPoint Desktop and PowerPoint Online environments with final End-to-End checks.
Related User Stories	US09, US10
Obstacles and solutions	PowerPoint Online imposed much stricter cross-origin restrictions than the desktop client, requiring adjustments to authentication mechanisms and CORS policies. Carefully configured server response headers resolved the issues.
Key outcomes	Total confirmation of system readiness. Operational and latency testing successfully recorded satisfying metrics, proving the mobile interface operates reliably alongside the projected slide.

Table B.9: Summary of results – Iteration 9

Appendix C

C.1 Deployment Instructions

This section provides the necessary steps to deploy the main TapClassroom project and the PowerPoint add-in from scratch.

System Requirements

Before running the complete TapClassroom ecosystem, ensure that the following system components are installed:

- Docker and Docker Compose (to containerise the TapClassroom server)
- Make utility (for environment shortcuts)
- Node.js and npm (to handle the add-in packages)
- Microsoft PowerPoint (desktop version for local debugging)
- A modern web browser to access the frontend

Execution Modes

The complete TapClassroom experience relies on two distinct but interrelated architectures that must be launched separately. The main server handles all network requests, database transactions and core business logic, whereas the Office add-in embeds the user interface directly inside the presentation software.

Running the TapClassroom Server

To initialise the backend ecosystem, follow these deployment steps:

1. Open a terminal window and navigate to the project directory containing the server architecture:

```
cd proyecto-TapClassroom
```

2. Launch the containerisation process using the provided Makefile. This automatically builds all the required Docker images for the local environment:

```
make start
```

3. Wait until the terminal indicates that the databases and backend endpoints have correctly booted. The main application is now operating.
4. You can access the participant web interface by opening a browser and navigating to `http://localhost:3000`.
5. To stop the environment once the session concludes, execute:

```
make stop
```

Running the PowerPoint Add-in

To configure the presenter functionality, the extension requires its own node initialisation:

1. Open a separate terminal and enter the add-in folder:

```
cd powerpoint-ContentAddIn
```

2. Resolve and install all necessary javascript dependencies. This step is only strictly needed the first time the project is opened:

```
npm install
```

3. Execute the local development toolchain:

```
npm start
```

4. The Office debugging suite will instantly process the manifest setup, launch a secure local server, and open Microsoft PowerPoint with the TapClassroom tool ribbon enabled automatically.
5. To detach and stop the debugging service securely, run:

```
npm run stop
```

C.2 User Manual

User Interface Overview

The system provides an integrated ecosystem bridging Microsoft PowerPoint, a mobile web client for presenters, and an analytical dashboard for educators. This connected structure allows the user to manage interactive polls and sessions dynamically, whether from inside their presentation or remotely via a mobile device, and later review engagement data.

Main Features

- **Presentation Integration**: Users can start sessions and manage live polls directly from embedded tools inside a PowerPoint slide.
- **Remote Session Control**: Presenters can govern session states, progress quizzes, and trigger questions externally via their mobile browsers.
- **Real-time Monitoring**: The system provides immediate statistics and connection statuses directly to the presenter's interfaces.
- **Session Analytics**: Users can access detailed post-session reports, including participation rates and response latencies.

Visual Guidance

PowerPoint Add-in Integration

The system allows presenters to control the classroom flow without ever leaving the Microsoft PowerPoint environment, embedding the necessary tools directly into the slides.

Inserting the Add-in

Before starting a presentation, the educator must insert the TapClassroom component into their chosen slides. To do this, open PowerPoint, navigate to the 'Add-ins' tab on the top ribbon and select the TapClassroom module. A placeholder frame will appear on the slide canvas, which can be resized to fill the desired area.

Initiating a Session

To start an interactive lecture, the presenter must play the slideshow and reach a slide containing the inserted add-in. Because the add-in mirrors the full web browser experience, the presenter first logs in with their credentials to access their complete user dashboard. From this familiar interface, the educator navigates to 'Stream view' with the quiz previously selected. Once launched, the embedded interface proyects the live contest content to the classroom, as shown in Figure C.1. Participants join the session by independently following the session link or scanning a related QR code, as there is no static code screen displayed by default.

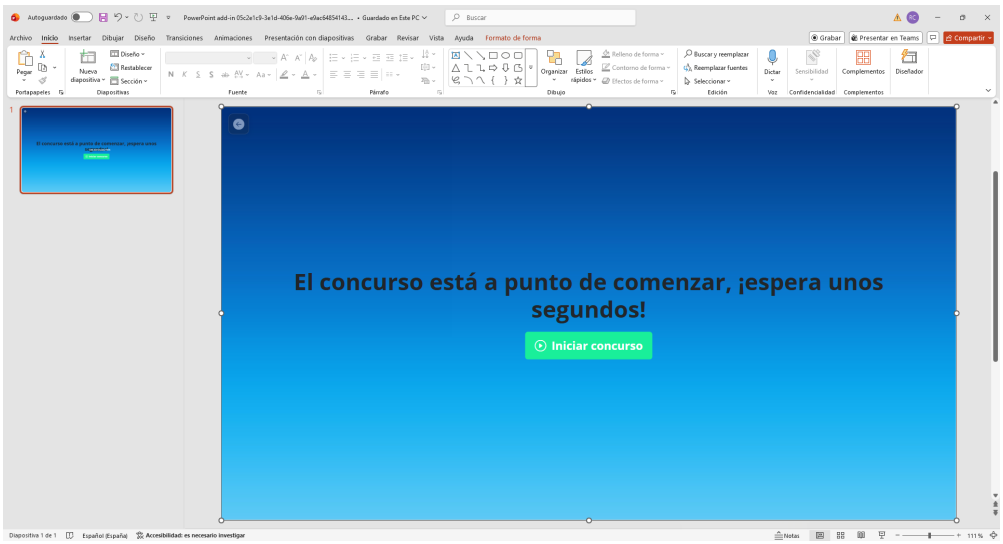


Figure C.1: Content Add-in embedded directly within a PowerPoint slide showing the live streaming view of the contest

Controlling the Polling Phase

Once the presentation advances to a question slide, the system updates the live stream to reflect the active question. To prevent visual clutter and distraction, the add-in does not project raw statistics or complex interfaces onto the main canvas. Instead, it superimposes a minimalist, floating control panel over the streaming view, as shown in Figure C.2. This condensed interface strips away the bulk of the desktop control dashboard, leaving only the essential commands required for presentation mode. Using this subtle overlay, the instructor can click to manually trigger the opening of the poll, reveal the correct answer, or proceed to the next phase without obstructing the quiz content.

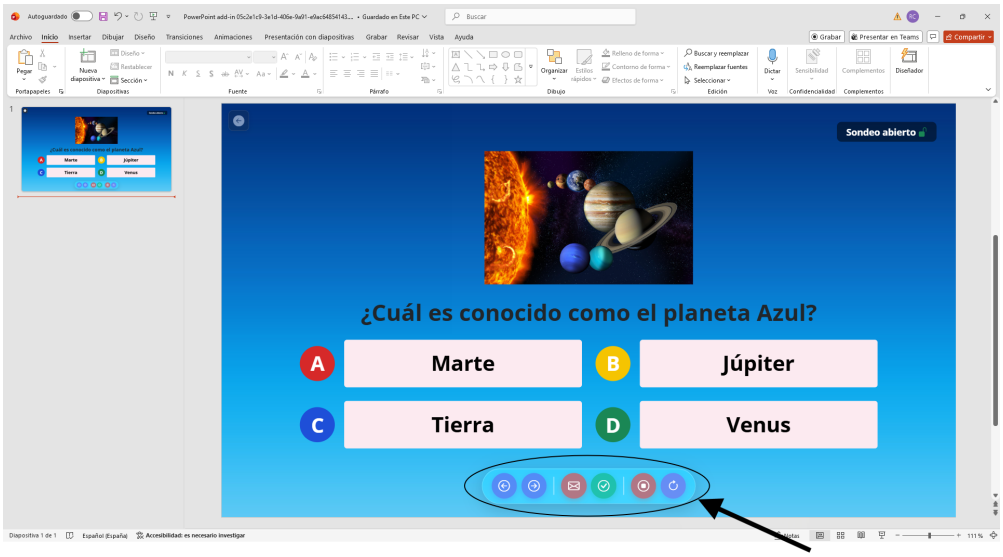


Figure C.2: Embedded Add-in displaying the clean stream view and the minimalist floating control panel

Mobile Presenter Interface

The mobile interface is designed for the instructor to have remote control over the active quizzes and session flow without being tied to the main computer.

Connecting to the Active Session

From any mobile browser, the presenter logs into their account at the TapClassroom platform URL. Upon logging in, the user can navigate directly to the mobile remote control panel. The interface synchronizes with the active PowerPoint broadcast, displaying the current status of the lecture.

Session and Quiz Management

Rather than answering questions, the mobile UI acts as a remote control.

- Using the prominent control buttons on the screen, the presenter can tap to manually trigger the opening of a poll or force-close a question phase directly from their smartphone.
- The interface provides a clear overview of connected participants and live response counters, ensuring the instructor holds full control of the pacing while moving freely around the classroom, as seen in Figure C.3.



Figure C.3: Mobile remote control interface displaying live session metrics and polling commands

Analytics Dashboard

After the lecture, the educator can review detailed metrics regarding student performance and engagement.

Accessing the Reports

To access these metrics, the educator must log into the main TapClassroom web platform through a desktop browser. On the main dashboard, the instructor selects the Reports module from the central navigation menu, as guided by the arrow in Figure C.4.

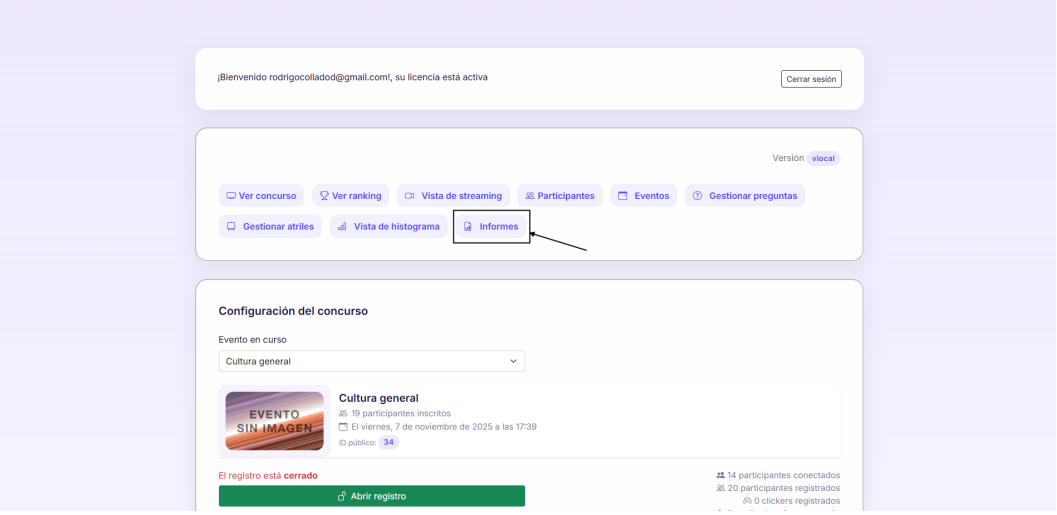


Figure C.4: TapClassroom dashboard highlighting the button to access the reports section

Reviewing Post-Session Data

Once inside the Reports view, the instructor first selects the specific finalized event from a dropdown menu. The interface then populates several expandable panels:

- **Participant Reports**: detailed insights into individual student performance.
- **Presenter Notes**: automated pedagogical alerts highlighting questions with high failure rates or significant abstention.

As shown in Figure C.5, the system also provides dedicated buttons to export this raw session data in universally accessible formats, allowing for further external analysis via `CSV` or `PDF` downloads.

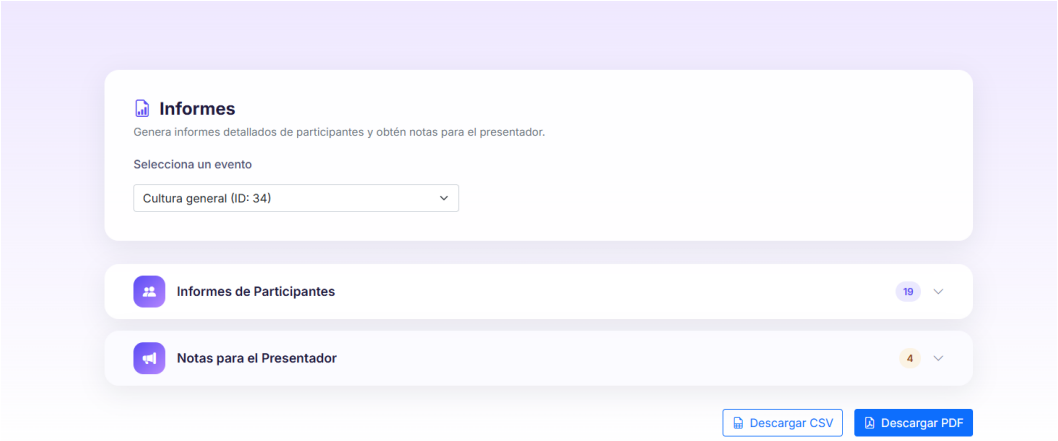


Figure C.5: Reports interface displaying event selection, expandable analysis panels, and export options

References

[Ber01] Yahn W Bernier. Latency compensating methods in client/server in-game protocol design and optimization. In *Game developers conference*, 2001.

[CMPRGP19] Antonio Cebrián Martínez, Ascensión Palomares Ruiz, y Ramón García Perales. Gamificación del aprendizaje y sus efectos en el rendimiento académico de alumnos universitarios. (Estudio de caso intrasujetos). In Elia M. Fernández-Díaz, Carlos Rodríguez-Hoyos, y Adelina Calvo Salvador, editors, *Jornadas Universitarias de Tecnología Educativa (XXVII Edición): Activismo y Tecnología, hacia una universidad comprometida con la educación crítica y emancipadora. Libro de actas*, pages 120–126, Santander, 2019. Universidad de Cantabria.

[CS14] Scott Chacon y Ben Straub. *Pro git*. Springer Nature, 2014.

[Csi90] Mihaly Csikszentmihalyi. *Flow: The psychology of optimal experience*. Harper & Row, 1990.

[DDKN11] Sebastian Deterding, Dan Dixon, Rilla Khaled, y Lennart Nacke. From game design elements to gamefulness: defining "gamification". In *Proceedings of the 15th international academic MindTrek conference: Envisioning future media environments*, pages 9–15, 2011.

[DR13] Edward L Deci y Richard M Ryan. *Intrinsic motivation and self-determination in human behavior*. Springer Science & Business Media, 2013.

[EN16] Ramez Elmasri y Shamkant B Navathe. *Fundamentals of database systems seventh edition*. Pearson, 2016.

[FH+01] Martin Fowler, Jim Highsmith, et al. The agile manifesto. *Software development*, 9(8):28–35, 2001.

[Fie00] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000.

[Fla20] David Flanagan. *JavaScript: The Definitive Guide: Master the World's Most-Used Programming Language*. O'Reilly Media, 7th edition, 2020.

[FM11] Ian Fette y Alexey Melnikov. The websocket protocol. RFC 6455, Internet Engineering Task Force (IETF), 2011.

[Fos13] Catherine Twomey Fosnot. *Constructivism: Theory, perspectives, and practice*. Teachers College Press, 2013.

[Fow12] Martin Fowler. *Patterns of enterprise application architecture*. Addison-Wesley, 2012.

[Kap12] Karl M Kapp. *The gamification of learning and instruction: game-based methods and strategies for training and education*. John Wiley & Sons, 2012.

[KH08] Rita Kop y Adrian Hill. Connectivism: Learning theory of the future or vestige of the past? *International Review of Research in Open and Distributed Learning*, 9(3):1–13, 2008.

[LMS23] Minzi Li, Siyu Ma, y Yuyang Shi. Examining the effectiveness of gamification as a tool promoting teaching and learning in educational settings: A meta-analysis. *Frontiers in psychology*, 14:1253549, 2023.

[Mer14] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014(239):2, 2014.

[MK06] Punya Mishra y Matthew J Koehler. Technological pedagogical content knowledge: A framework for teacher knowledge. *Teachers college record*, 108(6):1017–1054, 2006.

[MKP22] Ma Mohin, Leonine Kunzwa, y Sagar Patel. Using mentimeter to enhance learning and teaching in a large class. *International Journal of Educational Policy Research and Review*, 9(2):48–57, 2022.

[Mom01] Bruce Momjian. *PostgreSQL: introduction and concepts*. Addison-Wesley New York, 2001.

[Mor16] Kief Morris. *Infrastructure as code: managing servers in the cloud*. O'Reilly Media, 2016.

[MP13] Michael Mikowski y Josh Powell. *Single page web applications: JavaScript end-to-end*. Manning Publications, 2013.

[OH17] Regina O Obe y Leo S Hsu. *PostgreSQL: up and running: a practical guide to the advanced open source database*. O'Reilly Media, 2017.

[OSBSW20] David C Owens, Troy D Sadler, Angela T Barlow, y Cindi Smith-Walters. Student motivation from and resistance to active learning rooted in essential science practices. *Research in Science Education*, 50(1):253–277, 2020.

[Par72] David Lorge Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058, 1972.

[Pil10] Mark Pilgrim. *HTML5: up and running: dive into the future of web development*. O'Reilly Media, 2010.

[RBC+20] Sebastian Roth, Timothy Barron, Stefano Calzavara, Nick Nikiforakis, y Ben Stock. Complex security policy? a longitudinal analysis of deployed content security policies. In *Proceedings of the 27th Network and Distributed System Security Symposium (NDSS)*, 2020.

[RF20] Mark Richards y Neal Ford. *Fundamentals of software architecture: an engineering approach*. O'Reilly Media, 2020.

[RM06] David H Rose y Anne Meyer. *A practical reader in universal design for learning*. Harvard Education Press, Cambridge, MA, 2006.

[RM23] Elias Ratinho y Cátia Martins. The role of gamified learning strategies in student's motivation in high school and higher education: A systematic review. *Heliyon*, 9(8):e19033, 2023.

[SC18] Sujit Subhash y Elizabeth A Cudney. Gamified learning in higher education: A systematic review of the literature. *Computers in human behavior*, 87:192–206, 2018.

[SH20] Michael Sailer y Lisa Homner. The gamification of learning: A meta-analysis. *Educational psychology review*, 32(1):77–112, 2020.

[SHMM17] Michael Sailer, Jan Ulrich Hense, Sarah Katharina Mayr, y Heinz Mandl. How gamification motivates: An experimental study of the effects of specific game design elements on psychological need satisfaction. *Computers in human behavior*, 69:371–380, 2017.

[SPOMGP16] José Carlos Sánchez-Prieto, Susana Olmos-Migueláñez, y Francisco J García-Peñalvo. Informal tools in formal contexts: Development of a model to assess the acceptance of mobile technologies among teachers. *Computers in human behavior*, 55:519–528, 2016.

[SS20] Ken Schwaber y Jeff Sutherland. The scrum guide. Scrum.org, 2020.

[TKB22] Frank Tsui, Orlando Karam, y Barbara Bernal. *Essentials of software engineering*. Jones & Bartlett Learning, 2022.

[Tra05] John Traxler. Defining mobile learning. In *IADIS International Conference Mobile Learning 2005*, pages 261–266. IADIS, 2005.

[WH20] Kevin Werbach y Dan Hunter. *For the win, revised and updated edition: The power of gamification and game thinking in business, education, government, and social impact*. University of Pennsylvania Press, 2020.

[WS20] Ruth Wood y Shaista Shirazi. A systematic review of audience response systems for teaching and learning in higher education: The student experience. *Computers and Education*, 153:103896, 2020.

[WT20] Alf Inge Wang y Rabail Tahir. The effect of using Kahoot! for learning–A literature review. *Computers & Education*, 149:103818, 2020.

[YC79] Edward Yourdon y Larry L Constantine. *Structured design: fundamentals of a discipline of computer program and systems design*. Prentice-Hall, Inc., 1979.

Este documento fue editado y tipografiado con LaTeX empleando
la clase **esi-tfg** (versión 0.20181017) que se puede encontrar en:
`https://bitbucket.org/esi_atc/esi-tfg`