UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

BACHELOR IN COMPUTER SCIENCE

Automatic LED Control System for Podiums in a Television Quiz Show Platform

Carmen Cabañas Bógalo

July, 2025

Automatic LED Control System for Podiums in a Television Quiz Show Platform

UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

Department of Technology and Information Systems

BACHELOR IN COMPUTER SCIENCE

SOFTWARE ENGINEERING

Automatic LED Control System for Podiums in a Television Quiz Show Platform

Author: Carmen Cabañas Bógalo

Advisor: David Vallejo Fernández

Co-advisor: Francisco Manuel García Sánchez-Belmonte

July, 2025

Carmen Cabañas Bógalo

Ciudad Real – Spain

E-mail: Carmen.Cabanas1@alu.uclm.es
Telephone: 638 51 02 70

TRIBUNAL:

Presidente:

Vocal:

Secretario:

FECHA DE DEFENSA:

CALIFICACIÓN:

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

Abstract

In recent years, programmable LED lighting has established itself as a key expressive resource in interactive and entertainment environments, thanks to its energy efficiency, versatility and ability to integrate with digital systems. At the same time, TV quiz shows have remained one of the most recognisable cultural formats, especially in prime time. Their popularity has been reinforced both in traditional media and on digital platforms. This is due to the promotion of real-time participation dynamics and gamification elements, which encourage the active involvement of the audience.

In this context, Furious Koalas S. L., a spin-off company of the University of Castilla-La Mancha, has developed its own infrastructure for live competitions with audience interaction. This platform combines physical lecterns, mobile applications and audiovisual components to create immersive experiences. However, the management of the LED strips integrated in the lecterns was carried out using external tools that were too complex for the specific requirements of the system.

The aim of this work is to design, develop and validate a technological solution for the dynamic and synchronised control of LED lighting in interactive competitions. The proposal consists of a modular application composed of two main elements: on the one hand, a web interface that allows managing videos and actions in an intuitive way, and on the other hand, a back-end server in charge of coordinating the projection of animations in real time. This architecture communicates with the physical controller (Deskontroller 16) via the Art-Net protocol to project content onto addressable LED strips integrated into customised lecterns.

The previous tool used, MadMapper, offered excessive coverage in relation to the specific use required by the platform, as it is a professional solution designed for more complex artistic environments. In contrast, the system developed in this project significantly reduces the operational burden, eliminates dependence on external licenses and precisely fits the functional needs of Furious Koalas S.L. Moreover, it integrates with the main server of the competition via HTTP requests, allowing direct control from the central system.

During its development, challenges typical of real-time systems were addressed, such as audiovisual synchronisation, asynchronous task management or network communication between the computer and the controller. The tool was successfully deployed in a real environment, the XVIII Castilla-La Mancha Informatics Olympiads, where it proved its technical robustness and its ability to adapt to a live production situation.

Resumen

En los últimos años, la iluminación LED programable se ha consolidado como un recurso expresivo clave en entornos interactivos y de entretenimiento, gracias a su eficiencia energética, versatilidad y capacidad de integración con sistemas digitales. Paralelamente, los concursos de preguntas y respuestas tipo televisivo han mantenido su vigencia como uno de los formatos culturales más reconocibles, especialmente en franjas de máxima audiencia (*prime time*). Su popularidad se ha reforzado tanto en los medios tradicionales como en las plataformas digitales. Esto se debe al impulso de dinámicas de participación en tiempo real y elementos de gamificación, que fomentan la implicación activa del público.

En este contexto, Furious Koalas S. L., una spinoff de la Universidad de Castilla-La Mancha creada en 2016, ha desarrollado una infraestructura propia para la realización de concursos en directo con interacción del público. Esta plataforma combina atriles físicos, aplicaciones móviles y componentes audiovisuales para crear experiencias inmersivas. Sin embargo, la gestión de las tiras LED integradas en los atriles se realizaba mediante herramientas externas que resultaban excesivamente complejas para los requisitos concretos del sistema.

El presente trabajo tiene como objetivo diseñar, desarrollar y validar una solución tecnológica para el control dinámico y sincronizado de la iluminación LED en concursos interactivos. La propuesta consiste en una aplicación modular compuesta por dos elementos principales: por un lado, una interfaz web que permite gestionar vídeos y acciones de forma intuitiva, y por otro, un servidor backend encargado de coordinar la proyección de animaciones en tiempo real. Esta arquitectura se comunica con el controlador físico (Deskontroller 16) mediante el protocolo Art-Net para proyectar contenidos en tiras LED direccionables integradas en atriles personalizados.

La herramienta previamente empleada, MadMapper, ofrecía una cobertura excesiva en relación con el uso concreto que requería la plataforma, ya que se trata de una solución profesional diseñada para entornos artísticos más complejos. En contraste, el sistema desarrollado en este proyecto reduce significativamente la carga operativa, elimina la dependencia de licencias externas y se ajusta con precisión a las necesidades funcionales de Furious Koalas S.L. Además, se integra con el servidor principal del concurso mediante peticiones HTTP, lo que permite su control directo desde el sistema central.

Durante su desarrollo se abordaron desafíos propios de los sistemas en tiempo real, como la sincronización audiovisual, la gestión de tareas asíncronas o la comunicación en red entre el ordenador y el controlador. La herramienta se desplegó con éxito en un entorno real, las XVIII Olimpiadas de Informática de Castilla-La Mancha, donde demostró su robustez técnica y su capacidad para adaptarse a una situación de producción en vivo.

Acknowledgement

Finalizar este Trabajo de Fin de Grado no solo marca el cierre de una etapa académica, sino también un momento de reflexión y gratitud hacia todas las personas que me han acompañado en este camino.

En primer lugar, quiero expresar mi agradecimiento más profundo a mi familia. Gracias por haber estado siempre ahí, no solo durante estos años de universidad, sino desde mucho antes: en el colegio, en el instituto, en cada pequeño paso que me ha traído hasta aquí. Vuestra confianza en mí, vuestro apoyo incondicional y los valores que me habéis transmitido han sido fundamentales para convertirme en la persona que soy hoy. Todo este recorrido, con sus logros y dificultades, no habría sido posible sin vuestra presencia constante.

También quiero dar las gracias a mis amigos de siempre, con quienes he compartido gran parte de mi vida, y a los que he tenido la suerte de conocer durante la carrera. Vuestro apoyo, vuestras palabras de ánimo y esos momentos compartidos, tan necesarios en los días complicados, han sido un pilar imprescindible para seguir adelante.

Por último, pero no menos importante, quiero dedicar una mención especial a mis tutores de TFG, David y Paco. Gracias por brindarme la oportunidad de desarrollar este proyecto junto a vosotros y por confiar en mí desde el primer momento. Agradezco sinceramente vuestra guía, vuestras enseñanzas y la disponibilidad que siempre habéis mostrado para ayudarme en cada fase del proceso.

A todos vosotros, gracias por formar parte de este camino.

Carmen

A los que siempre estuvieron

Contents

List of Tables

List of Figures

List of code listings

Chapter 1

Introduction

In this chapter, the general context and motivation behind the project are presented, followed by an overview of the proposed system and the structure of this document.

1.1 Context

Today, LED lighting systems are widely used in all types of environments, including industrial and cultural spaces. Their low energy consumption, colour customisation capabilities and ease of integration with electronic and digital systems have made these systems a key tool for designing visual experiences. From stage shows and artistic installations to urban environments, road signs and educational spaces, programmable light is increasingly being used as a functional and expressive resource.

1.1.1 The popularity of quiz-based formats across media and platforms

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?*, *Des chiffres et des lettres* and *Pasapalabra* have had a massive audience for a years. The television quiz show is one of the earliest and most enduring forms of broadcast entertainment, based on the idea of *ordinary* people performing under pressure [Su 08].

Such formats have enjoyed longevity thanks to their hybrid nature: combining factual knowledge with spectacle, individual competition with collective viewing and entertainment with a sense of shared cultural values. This combination has enabled quiz shows to remain relevant despite changes in the media landscape, such as the shift towards digital interaction and mobile entertainment.

In parallel, the digitalisation of entertainment has led to the emergence of trivia-based apps and interactive quiz platforms such as *Kahoot!* [1] or *Trivial Crack* [2]. These platforms use competition, instant feedback and game-based progression systems to engage users in educational and recreational contexts.

[1] `https://kahoot.com`
[2] `http://preguntados.com/`

1.1.2 Gamification and its benefits for interactive events

The use of game mechanics in non-game contexts, known as *gamification*, has become more common in areas like education, health, productivity and entertainment media. According to Deterding et al., this phenomenon can be defined as "the use of game design elements in non-game contexts" [DDea11]. Typical mechanisms include points, badges, leaderboards, and instant feedback systems, which aim to increase user engagement and participation. The ability of these mechanics to stimulate intrinsic motivation has been proved, particularly when they are designed to support users' sense of autonomy, competence, and relatedness, as outlined by Self-Determination Theory [RD00].

In interactive events, gamification plays a dual role: it transforms the structure of participation, encouraging users to remain attentive and engaged, and it amplifies the memorability and emotional impact of the experience. Empirical studies confirm that providing users with immediate feedback in the form of scores, badges or performance indicators enhances their engagement and perceived competence [HKea14].

Moreover, interactive lighting has been identified as an effective form of visual feedback. LED-based light displays can notify remote users of the system's status or ambient interaction without the need for on-screen messages, creating a more natural and spatially distributed interface [FBea15]. Similarly, light stimuli with colour, intensity or animation are successfully used in educational contexts to signal progress, collaboration or task completion [LRea20].

1.1.3 Furious Koalas' infrastructure

In this context, Furious Koalas Ltd. [3], a spinoff of the University of Castilla-La Mancha, has developed its own infrastructure for real-time quiz-based contests with live production and audience interaction. This system has been used at events like the Informatics Olympiads and in special quiz shows for companies, weddings, trade fairs, and other public or company events. The company makes interactive experiences based on TV shows like *Pasapalabra*, *Who Wants to Be a Millionaire?* or *¡Boom!*, and can customise them to suit the customer. These activities combine live hosting, digital devices and audiovisual elements, including LED lighting, to enhance engagement and participation. Participants can join in person via lecterns or remotely through mobile apps that allow them to answer questions in real time. [4]

As shown in Figure 1.1, Furious Koalas' infrastructure has been successfully implemented in corporate settings, such as at the ABB team-building convention held in April 2023 at the Oceanogràfic Auditorium in Valencia. There, more than 250 employees took part in a live quiz designed to reinforce internal knowledge through gamified interaction.

[3] `https://www.furiouskoalas.com`

[4] See for example: `https://www.youtube.com/watch?v=OFo6m7AlgOY`

Figure 1.1: Live quiz competition organised by Furious Koalas for the company ABB. The quiz included company-related questions and used digital lecterns with LED lighting and real-time ranking display.

This infrastructure involves custom-designed lecterns equipped with buttons, integrated screens and distributed LED strips. These lecterns serve as the main physical interface for participants, enabling direct and synchronous interaction with the quiz system. The logic and scoring of each game session is managed by the main coordination platform developed in-house, which in this document is referred to as the *Main Quiz System*. The aim is to make the experience of participants and attendees more exciting by using visual elements that are in sync with the event's progression.

Until now, lectern lighting was managed using MadMapper software[5], a professional tool that is widely used in artistic projection environments. However, in this case, it was used exclusively to control the LEDs by the spinoff staff, which was not worth the operational complexity involved in configuring it. Although the system worked properly during events, the prior preparation was impractical and tool itself proved to be far more ambitious and complex than the specific needs of this project required.

These limitations motivated the development of a more flexible software solution, specifically adapted to the Furious Koalas lectern system. The aim was to offer a more accessible, efficient, and fully integrated control in the contest logic.

[5] `https://madmapper.com/`

1.2 Project proposal

This Final Degree Project was carried out within the framework of an internship at Furious Koalas. It proposes the design, development, and validation of a customised software solution for LED lighting control in interactive environments, which is applied to an existing infrastructure of lecterns. Based on the system's functional requirements, a modular application has been developed that can receive projection orders from a web interface and the main quiz platform. Both sources of interaction are integrated transparently.

The system enables dynamic animations to be projected onto LED strips. During its development, challenges associated with real-time control systems were addressed, including accurate synchronisation of audiovisual content, concurrent task management, and maintaining fluidity in playback under high-load conditions.

With a user-friendly interface and the ability to validate in real environments, the system proved its operational feasibility at the XVIII Castilla-La Mancha Informatics Olympiads held in April 2025 at the School of Computer Science (ESI) in Ciudad Real.

In comparison to the previous MadMapper-dependant setup, the new implementation brings several practical advantages. Firstly, it eliminates dependency on third-party software for control over integration with the event's main logic. Secondly, the solution's modular structure makes maintenance and future extensions, such as new animation formats or interaction modes, easier. Lastly, the web interface allows non-technical users to operate the system with minimal training.

Overall, this project shows the development of a technically robust and versatile control system. This integrates software engineering, real-time audiovisual synchronisation, and direct application in live production environments. The successful deployment of the system at a public event highlights its maturity and potential for reuse in similar interactive scenarios.

1.2.1 Architectural overview

The LED control system developed in this project has a modular, layered architecture which separates responsibilities into three main categories: presentation, domain logic, and persistence. This high-level organisation ensures the system is maintainable and scalable, and provides a clean separation of concerns between the user interface, the core functionality, and the data layer. At a more detailed level, the domain layer is subdivided into three conceptual packages following *clean architecture* principles [Mar17]: interface adapters (HTTP endpoints); an application coordination layer (controllers); and a domain core containing business logic and entities.

The system is composed of two main parts. The first part is a *web interface* developed in TypeScript, which lets users select videos or predefined actions and send them to specific

lecterns. These videos and actions contain the animations to be projected on one or more LED lecterns, according to the chosen configuration. The second part is a *back-end service* made in Python using Flask. This is in charge of the playback logic and handles real-time communication with the Art-Net LED controller model (Deskontroller 16) used in Furious Koalas' quiz infrastructure.

The back-end consists of three logical subsystems that handle video pre-processing, double buffering, and DMX data transmission. These subsystems have distinct functional responsibilities within the system. LED animations are encoded into RGB frames, loaded into a buffer structure and transmitted in synchrony with the event flow.

A simplified diagram of the system architecture is shown in Figure 1.2. It shows the main functional modules, which are grouped into three conceptual layers: presentation, domain and persistence. It also highlights the separation of concerns that guided the design of the system. This architectural structure has been designed to facilitate future extensions and promote modular development, ensuring that each subsystem can evolve independently without compromising the system's overall functionality.

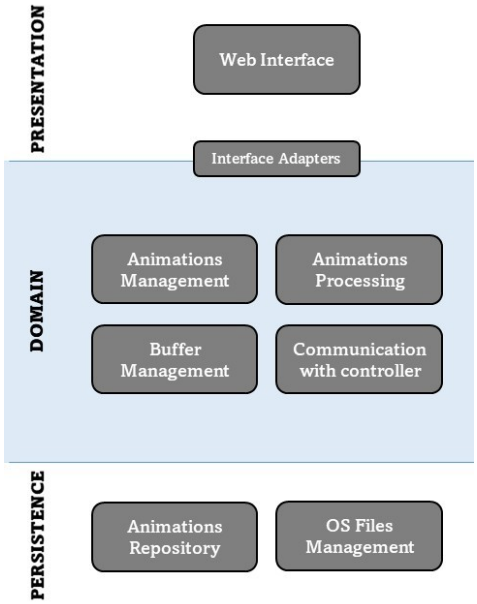


Figure 1.2: Simplified high-level overview of the LED Management System.

1.3 Document structure

The remainder of this document is structured as follows, 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 main functional goals of the system, aligned with the specific needs identified in the project's context.

Chapter 3: State of the Art

This chapter reviews the main technologies and engineering principles relevant to system development. It is divided into three major thematic areas: programmable LED lighting and its control protocols; video processing techniques for LED-based animations; and software engineering methods applied to real-time lighting control systems.

Chapter 4: Methodology

The methodology followed here includes the planning made and also other resources used in this project.

Chapter 5: Architecture

This chapter provides a detailed overview of the architecture of the developed system. It explains the reasoning behind the key design choices, the evolution of the system, and its internal structure.

Chapter 6: Results

This chapter presents the main outcomes of the project, highlighting how the system was used in a real contest environment.

Chapter 7: Conclusions and future work

The final chapter discusses proposed objectives have been achieved and suggests possible improvements and directions for future development.

Chapter 2

Objectives

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

2.1 General objectives

This project aims to design a customised system for the automatic control of programmable LEDs integrated into lecterns used in a TV quiz show, whose platform belongs to the UCLM spin-off Furious Koalas S.L. Also, the system must be able to operate in synchronisation with the main quiz-control server and be fluid in their interactions. This means that lighting effects must be accurately timed and coordinated with events such as contestant responses, score changes, or game phases.

Thus, this system aims to provide an efficient and practical solution for displaying video content on the LED strips attached to the lecterns. The solution must be integrated with the Deskontroller 16, a model of the PIX CONTROL 16 controller, which is compatible with the Art-Net protocol. It must provide a simplified user interface to facilitate the configuration of the animations.

2.2 Specific objectives

The LED control system developed in this project is designed to serve as a key component within the quiz show platform developed by Furious Koalas S.L. The specific objectives of the system are as follows:

- **Facilitate system usability for non-technical users**: The system must be easy for users without programming knowledge to operate. This requires a simplified interaction model and an intuitive interface so that lighting animations can be triggered and configured without any technical expertise.
- **Ensure visual fidelity and synchronisation**: The colour output of the LEDs must accurately reflect the original video associated with each animation to be projected onto the lecterns. This includes minimal latency and frame loss, which is crucial for guaranteeing a seamless visual experience during the quiz show.

- **Integrate the system with the quiz platform**: The LED control module must be compatible with the company's existing infrastructure. This involves receiving instructions from the main server in standard communication formats and responding to external events in real-time.
- **Validate the system in a real-world context**: The project must be tested in a realistic environment using the company's actual hardware. A first version of the system is expected to be deployed and used at a public event to demonstrate its effectiveness and robustness.
- **Contribute to cost reduction and platform competitiveness**: The solution is intended to replace the use of licenced external software, which is currently employed in the production of quiz shows and requires a paid subscription. By developing a LED control module, the company aims to reduce its long-term operational costs, as well as to increase the autonomy and competitiveness of its platform.

Chapter 3

State of art

In this chapter, it is presented a comprehensive review of the technologies, methodologies, and architectural approaches related to LED lighting control and video-driven visual systems. The subject is divided into three primary sections: lighting control protocols and hardware, video analysis techniques for dynamic lighting, and software engineering strategies applied to real-time multimedia systems. This fundamental knowledge underpins the subsequent development of the system outlined in subsequent chapters.

3.1 LED Lighting Control Technologies

3.1.1 Introduction to Programmable LED Lighting

LED technology history

The technology related to light-emitting diodes has experienced significant evolution since its first discoveries, becoming one of the main lighting solutions these days.

1907 – First contact with electroluminescence by Henry Joseph Round

The first documented observation of electroluminescence, a phenomenon in which semiconductors emit light, was made in 1907 by the British engineer Henry J. Round. During testing carborundum-based cat whisker rectifiers (polycrystalline SiC), he observed visible light at the metal–semiconductor junction [DK08]. He published this in a brief note in *Electrical World* [Rou07].

Although similar effects in materials like silicon and germanium were sporadically observed in the following decades, their erratic nature and unclear physical basis limited further progress [Sch03].

1920s - Oleg Lossev's investigations

Between 1923 and the 1940s, Russian physicist Oleg V. Lossev studied electroluminescence in point-contact silicon carbide (SiC) diodes at the A. F. Ioffe Physical-Technical Institute in Leningrad. He investigated their electrical and optical behaviour in detail and correctly identified the light emission as originating from a surface 'active layer' associated with the n-type region, although the emission mechanism remained unclear [DK08].

Lossev envisioned optical data transmission, thus anticipating modern photonics. Despite the limitations imposed by resource scarcity and political repression (he died during the Siege of Leningrad in 1942), his contributions stimulated interest in semiconductor physics and provided the basis for future LED technology [DK08].

1962 - First visible LED by Nick Holonyak

In 1962, while working at General Electric, Nick Holonyak Jr. developed the first LED to emit visible red light using a gallium arsenide phosphide (GaAsP) alloy [DK08]. This marked a step change from infrared-only emissions, earning him the recognition of being the 'father of visible LEDs'.

The use of direct-bandgap semiconductors, like GaAsP, allowed efficient light emission and made multicolour and high-brightness LEDs possible, expanding their application from basic indicators to electronic displays [DK08].

1980-1990 - Evolution to high-efficiency and high-brightness LEDs

In the 1980s and 1990s, LED technology advanced with the introduction of compound semiconductors like aluminium-gallium-arsenide (AlGaAs) and indium-gallium-aluminium-phosphide (InGaAlP), enabling greater luminous efficiency, intensity, and colour range, as described in [DK08].

These improvements emerged from a better understanding of heterostructures and the adoption of epitaxial techniques such as metal-organic chemical vapour deposition (MOCVD), which enabled the precise control of crystalline layers at the nanoscale level [DK08, Sch03]. Consequently, LEDs evolved from simple indicators to general-purpose lighting elements used in road signs, automotive systems and electronic displays [DK08]

From conventional to programmable LEDs

The transition from fixed LEDs to programmable systems was driven by the convergence of digital electronics and control protocols such as DMX512 [EST24], standardised in the 1980s. This enabled the individual control of each light point, allowing for dynamic lighting installations.

The later spread of addressable LED strips—based on chips like WS2812 [1] or APA102 [AEC16], and their integration with platforms such as Arduino [BS14] or Raspberry Pi [2] made programmable lighting accessible to both professionals and amateurs. Programmable LEDs differ from conventional ones in that they can interpret real-time instructions to adjust colour, intensity, or behaviour individually.

[1] Adafruit NeoPixel Überguide: `https://learn.adafruit.com/adafruit-neopixel-uberguide`
[2] `https://projects.raspberrypi.org/en/projects/physical-computing/3`

Modern applications of programmable LEDs

Thanks to their efficiency, durability and programmable behaviour, LEDs have evolved into core components in diverse sectors [DK08, KSea07]. In **transport and automotive**, they are used in traffic signals, variable signage and vehicle lighting for their visibility, low consumption and fast response [SBea00, Yu13]. In **consumer electronics**, they serve as backlighting in displays, mobile devices and large-format panels [Tsa05].

Figure 3.1 illustrates some of these applications, including traffic signage, automotive lighting, and stage performance environments, where programmable LEDs have become commonplace.

Figure 3.1: Examples of modern applications of programmable LEDs: traffic signage, automotive lighting, and LED panels in live concerts.
Sources: (left) `https://tecnivial.com/catalogo/carreteras/senalizacion-luminosa/senalizacion-luminosa-definitiva-y-radares/senales-y-paneles-led/`, (center) `https://recambiosloeches.com/nueva-normativa-para-instalacion-de-led-en-automoviles/` and (right) `https://pin.it/7dViRTafb`

Beyond industrial use, programmable LEDs are central to **stage and artistic environments**. Their synchronisation capabilities via DMX512 [EST24] or Art-Net [Art24] enable dynamic light effects in concerts, plays or audiovisual installations [GBea24, Dun15]. Their small size makes them suitable for integration into costumes, scenery, or instruments.

In the field of **interactive art and immersive installations**, programmable LEDs respond to external stimuli (e.g., motion, sound), generating personalised visual experiences [SCea23]. These systems are found in museums, festivals and urban interventions, where light becomes a material of perception. A notable example is shown in Figure 3.2, which depicts an immersive environment created by James Turrell.

Finally, they are increasingly employed in **dynamic architectural lighting and decoration**, both indoor and outdoor, transforming spaces through animated light patterns [Xu24]. Software such as MadMapper, TouchDesigner or Resolume[3] allows full creative control over each pixel's behaviour.

[3] `https://madmapper.com`, `https://derivative.ca`, `https://resolume.com`

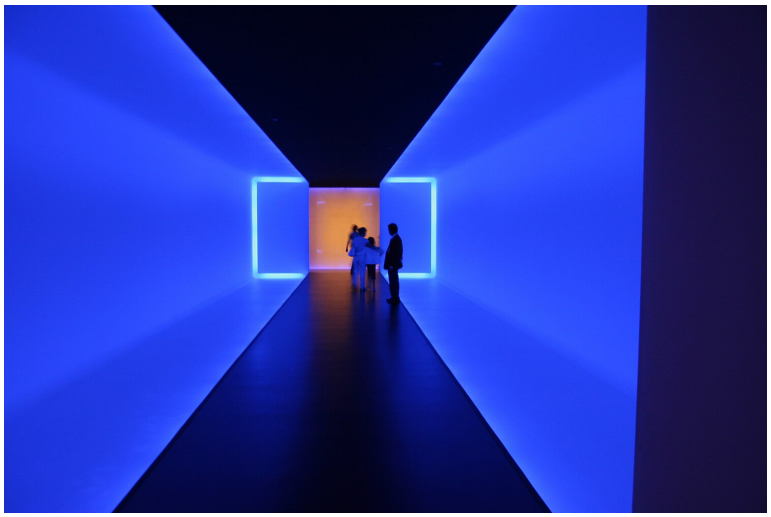


Figure 3.2: Immersive installation by James Turrell, where carefully calibrated lighting and geometric framing are used to transform architectural space through light perception. Image: 'The Light Inside', Museum of Fine Arts, Houston. Photo by Ed Schipul (Houston, TX, USA)
Source: `https://www.archdaily.cl/cl/998736/la-luz-como-materia-10-artistas-que-transforman-el-espacio-con-la-iluminacion`

Feature	**Conventional LEDs**	**Programmable LEDs**
Control	On/off switching or basic colour variation	Individual control of colour, brightness, and timing
Directionality	Uniform behaviour across the circuit	Addressable: each LED responds independently
Protocols	No data interpretation; voltage-based	Support DMX512, SPI, Art-Net
Hardware	Simple drivers or resistors	Requires controllers and digital interfaces
Applications	Lighting, indicators, displays	Media art, installations, reactive environments
Interactivity	No feedback or adaptation	Real-time reaction to sensors, sound, or video

Table 3.1: Functional comparison between conventional and programmable LEDs.

Differences between conventional LEDs and programmable LEDs

Although both types of LEDs operate by injecting current across a p–n junction [Sch03], their control capabilities and application domains differ significantly. As shown in Table 3.1, programmable LEDs incorporate digital interfaces, support communication protocols, and enable real-time, per-pixel control—features absent in conventional implementations.

Advantages of LED programming and control over traditional lighting

Compared to traditional technologies like incandescent or halogen lamps, programmable LEDs offer substantial benefits [GBea24]:

- **Dynamic and individualised control**: Support for dynamic effects, colour transitions, and synchronisation through DMX512 [EST24] or Art-Net [Art24].
- **High energy efficiency**: Lower power consumption and adaptive brightness [Tsa05].
- **Longer lifespan**: Reduced maintenance costs due to extended durability [Tsa05].
- **Instant response**: Immediate on/off switching, crucial for real-time visual effects [SCea23].
- **Integration flexibility**: Available in strips, arrays, or panels, adaptable to architecture, costumes, or furniture [Tsa05].
- **Interactive potential**: Integration with sensors enables intelligent, responsive environments [SCea23].

3.1.2 Communication Protocols for LED Control

The control of programmable LEDs relies on digital communication protocols that transmit lighting data precisely and reliably. Different protocols suit different needs in terms of scale, complexity and latency. This section reviews the most commonly used standards: DMX512, Art-Net, sACN (E1.31) and SPI, focusing on their technical characteristics, limitations and typical applications in modern LED systems.

DMX512: Standard in stage lighting

The DMX512 protocol was developed in 1986 by the United States Institute for Theatre Technology (USITT) and later standardised by ESTA [EST24]. Designed for stage lighting control, it uses unidirectional communication over RS-485, continuously transmitting digital data through a single line. Each 'universe' can manage up to 512 channels, each representing a value between 0 and 255.

In RGB LED systems, each LED usually needs three channels (R, G and B), enabling control of up to 170 RGB LEDs per universe. Larger setups require multiple universes and distribution hardware, such as Art-Net nodes or DMX splitters [Art24].

Furthermore, the typical refresh rate of DMX512 is 44 Hz. Using many channels increases the update interval per LED, which can introduce latency, especially problematic in synchronised audiovisual applications. [EST24].

Despite its limited bandwidth and unidirectional nature, DMX512 remains the industry standard for medium-scale lighting setups where full-pixel precision is not required. Its reliability and widespread adoption make it ideal for use in concerts, theatre productions and architectural lighting.

As shown in Figure 3.3, DMX fixtures are commonly connected in a daisy-chained topology, with a 120 Ω termination resistor at the end to maintain signal integrity.

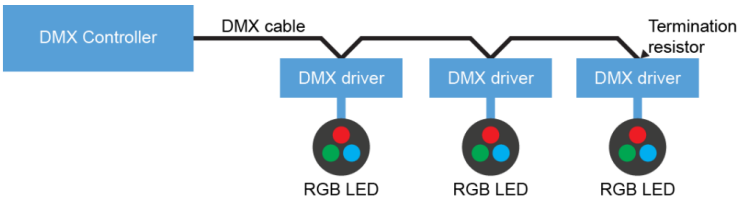


Figure 3.3: Typical DMX512 wiring setup: a DMX controller sends unidirectional data to a sequence of DMX drivers, each connected to an RGB LED fixture. The resistor is placed at the end of the line to ensure signal integrity.
Source: *How to wire DMX/RDM lighting systems*, eldoLED, 2020. Available at: `https://www.soliled.com/wp-content/uploads/2020/06/Learning-Center_Application-note_How-to-wire-DMX-lighting-systems.pdf`

Art-Net: DMX through the Ethernet

Art-Net is a protocol developed in 1998 by Artistic Licence to transmit DMX512 data over Ethernet networks using UDP/IP [Art24]. Taking advantage of standard infrastructure (e.g. switches and routers) enables the management of hundreds of DMX universes simultaneously, equivalent to tens of thousands of channels. This scalability makes Art-Net ideal for large-scale installations involving numerous addressable LEDs, such as LED walls, synchronised shows or interactive projections.

Unlike serial DMX512, it benefits from the high bandwidth and speed of Ethernet, offering lower latency and higher refresh rates, which facilitates real-time synchronisation with audio or video content [Art24]. The use of IP routing allows flexible universe distribution across multiple nodes, simplifying the organisation of complex systems. Furthermore, software tools like MadMapper, Resolume, QLC+ and TouchDesigner[4] are natively compatible with Art-Net, enabling direct control from a computer without external DMX hardware.

In short, Art-Net overcomes the limitations of the DMX512 structure in terms of channel count, speed and scalability by extending it over a modern network. It has therefore become the reference protocol for advanced lighting control.

sACN: Modern evolution of Art-Net

The sACN protocol (Streaming Architecture for Control Networks), standardised as ANSI E1.31, was developed by Entertainment Services and Technology Association (ESTA) to transmit DMX512 data over Ethernet more efficiently than earlier protocols such as Art-Net

[4] `https://madmapper.com`, `https://resolume.com`, `https://www.qlcplus.org`, `https://derivative.ca`

or DMX512 itself [Ent18]. It supports up to 63,999 universes, nearly doubling Art-Net's capacity.

A key improvement is its use of *multicast* transmission, which sends data only to subscribed devices. This reduces network load and enhances efficiency, as illustrated in Figure 3.4. However, proper network configuration (e.g., IGMP snooping) is essential to take full advantage of this feature [Ent18].

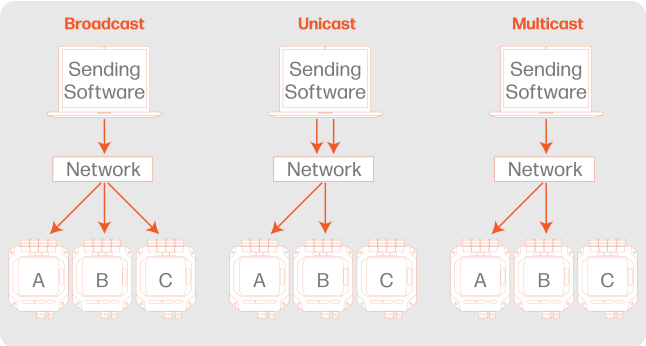


Figure 3.4: Comparison of data delivery modes in network-based lighting protocols: *broadcast* (used by Art-Net), *unicast*, and *multicast* (used by sACN). Only multicast sends data exclusively to devices that request it, improving efficiency.
Source: Advatek Lighting, *Art-Net vs sACN: which should I use?*, available at `https://www.advateklighting.com/blog/guides/art-net-vs-sacn`

While sACN offers superior scalability and network optimisation, Art-Net remains popular for its simplicity and broad device compatibility [Ent18].

SPI: Direct and efficient low-level control

SPI (Serial Peripheral Interface) is a synchronous protocol widely used in embedded systems to communicate between microcontrollers and peripherals [Mic03, Tex12]. In LED control, SPI enables direct pixel-level addressing in strips or matrices [iSk24].

Unlike protocols such as DMX512 or Art-Net, which operate over networked or serial infrastructures, SPI uses direct electrical lines (typically a data line and, optionally, a clock line) to connect the microcontroller (e.g. Arduino, ESP32 or Raspberry Pi) to the LEDs [Tex12, Mic03].

Its advantages include high transmission speed, precise timing and low hardware requirements, as it eliminates the need for intermediary controllers or converters [Tex12, SVea20]. However, signal degradation over long distances and dependence on the microcontroller's processing capacity limit its scalability [iSk24].

SPI allows accurate control of each peripheral at high transmission speeds [Tex12], making it suitable for precise and fast communication. Moreover, without conversion or intermediate protocols required, SPI is a direct and efficient solution in terms of processing [SVea20].

Feature	DMX512	Art-Net	sACN	SPI
Transmission	RS-485 (XLR)	Ethernet (UDP)	Ethernet (UDP, multicast)	Direct GPIO lines
Scalability and channels	1 universe (512 ch)	100+ universes	Up to 63,999 universes	Depends on clock and wiring
Speed and latency	Moderate (44 Hz)	High	High, optimised via multicast	Very high, real-time control
Hardware required	DMX controller	Art-Net node or compatible software	sACN-capable devices	Microcontroller only
Use cases	Theatre, small setups	Video mapping, large shows	Complex installs, architectural control	Makers, DIY, education

Table 3.2: Summary of key differences between LED control protocols.

Due to its simplicity and accessibility, SPI is frequently used in maker projects, education, experimental media art and prototyping, where control and flexibility are prioritised over scale or standardisation.

Comparing DMX512, Art-net, sACN and SPI protocols

To better understand the characteristics and practical implications of the protocols presented in the previous sections, table 3.2 provides a concise comparison of the four main communication protocols used in programmable LED systems. It includes core aspects such as transmission method, scalability, speed, and typical use cases.

3.1.3 Controllers and Hardware for LED Management

In programmable LED systems, controllers or drivers act as intermediaries between control software and the LEDs. They interpret digital signals from protocols like DMX512 [EST24], Art-Net [Art24], sACN [Ent18] or SPI [SVea20], converting them into electrical signals that activate each LED or group.

Depending on the installation's complexity, controllers range from simple DMX decoders to advanced multi-universe interfaces with real-time synchronisation capabilities. Choosing the right hardware depends on the LED type, communication protocol, project scale, and control precision required.

PIX CONTROL 16: operation, features and capabilities

The term *PIX CONTROL 16* refers to multi-output controllers compatible with Art-Net, capable of handling 16 DMX universes (8,192 channels). These devices are suitable for medium to large-scale LED installations where efficient management of multiple universes is required.

In this project, the **Deskontroller 16** was used as a representative model. It supports 16 Art-Net universes via 8 physical ports (2 universes per port) and outputs SPI (TTL) signals directly to addressable LED strips, such as WS2812, SK6812 or APA102 [Des17]. It is not suitable for conventional DMX512 fixtures.

More advanced variants, such as the *Deskontroller LITE V3*[5], manage up to 32 universes (16384 channels) and offer for more precise control and enhanced mapping.

General Operation

These controllers:

- Receive Art-Net packets via Ethernet (UDP/IP) [Art24].
- Convert the data into SPI signals, emitted through physical outputs for direct connection to digital strips.

Configuration Interface

The Deskontroller 16 includes a web-based interface to configure:

- Universe assignment, channel count and LED layout.
- Colour order (RGB/RGBW), gamma correction and timing sync.
- Network settings (IP, subnet mask).

Performance and Applications

With refresh rates up to 50 fps, it supports smooth animations and synchronised lighting. Typical use cases include:

- Interactive installations and digital art.
- LED façades and museum displays.
- Video-to-light synchronisation in multimedia events.

In summary, the Deskontroller 16 is a robust and efficient option for pixel-based LED control in Art-Net environments, particularly when traditional DMX512 fixtures are not involved.

Alternatives to Deskontroller 16

There are multiple alternatives to the Deskontroller 16, adapted to different budgets, project scales, and user profiles. These solutions can be broadly categorised into two groups: professional-grade controllers and DIY-oriented or educational devices.

[5] `https://www.deskontrol.net/es/controlador-led-art-net/384-deskontroller-lite-v3-32-universos-artnet.html`

1. Professional Controllers

Enttec DMX USB Pro: A compact USB-to-DMX interface widely used in theatrical and educational settings. Although it only handles one DMX universe, it is valued for its reliability, compatibility with software like QLC+, LightKey or Resolume, and ease of setup. It requires a computer to remain connected during operation. [6]

Advatek PixLite Mk3: A high-end Art-Net/sACN controller with direct SPI output, designed for medium and large-scale installations. It offers advanced features such as pixel mapping, integrated effects and redundancy, as well as a comprehensive web interface. [7]

Philips Colour Kinetics Data Enabler Pro: An all-in-one solution for architectural lighting that combines data and power delivery over a single line. Typically used in permanent, large-scale installations, it integrates smoothly with lighting design platforms and high-durability fixtures. [8]

2. DIY (Do It Yourself), educational or low-cost solutions

Arduino + TLC5940: Combining Arduino with the TLC5940 driver from Texas Instruments enables control of up to 16 high-resolution PWM channels. [9] Although it is ideal for educational or small-scale projects, it lacks native support for DMX or Art-Net unless additional modules or custom code are used. [10]

Raspberry Pi + SPI / Art-Net (with libraries such as rpi_ws281x or OLA): Raspberry Pi can control addressable strips via SPI or function as an Art-Net node using libraries like OLA.[11] It is a flexible platform for experimental setups or as an intermediate bridge between software and hardware.[12]

ESP32 with WLED or FastLED: This WiFi-enabled microcontroller supports real-time LED control via apps or remote servers. Firmware like WLED [13] allows Art-Net and E1.31 reception, making it suitable for mobile or temporary installations.[14]

T-1000S / K-1000C / K-8000C: Low-cost offline controllers that play preloaded animations from SD cards. Commonly used in façades or storefronts, they support chips like WS2812 but lack real-time responsiveness [88L17, LED23].

[6] https://www.enttec.com/es/product/dmx-usb-interfaces/dmx-usb-pro-professional-1u-usb-to-dmx512-converter/
[7] https://www.advateklighting.com/products/collections/professional-pixel-control
[8] https://www.colorkinetics.com/global/products/pds/dataenablerpro
[9] https://www.ti.com/product/TLC5940
[10] https://www.arduino.cc
[11] https://www.openlighting.org/ola
[12] https://www.raspberrypi.com
[13] https://kno.wled.ge
[14] https://www.espressif.com/en/products/socs/esp32

Practical Considerations in Large LED Installations

When it comes to large-scale addressable LED systems, selecting a suitable controller requires an evaluation of the installation's physical, electrical and logical characteristics. Most controllers operate with addressable chips that embed control logic in each pixel, enabling individual data processing. The most widely used controllers are:

WS2812 – Popular RGB chip using a single PWM data line without a clock [Wor13]. [15]

APA102 – Also known as DotStar, uses SPI (data + clock), enabling higher refresh rates [AEC16]. [16]

SK6812 – Variant compatible with WS2812, but offering improved colour consistency and RGBW versions [DOOC15].

These LEDs are wired in a daisy-chain: data is passed from one LED to the next. While this simplifies cabling, it introduces two key limitations:

- Signal degradation with long strips or high pixel counts, causing artefacts or loss of synchronisation [iSk24].
- Chain interruption if one LED fails.

Electrically, each RGB pixel can draw up to 60mA. In long strips, this may cause voltage drop, leading to dimmer LEDs at the far end. To mitigate this, power injection points are distributed along the strip, ensuring uniform brightness and avoiding system instability.[17]

3.2 Video Processing for LED Animations

3.2.1 Video Analysis Techniques for Lighting

LED lighting systems can respond to video content using computer vision techniques to extract relevant visual features in real time. The most common techniques are motion detection, colour analysis and image segmentation.

Motion detection identifies dynamic regions within a scene and is useful for triggering reactive lighting or highlighting movement. Techniques include background subtraction, optical flow, and interframe difference analysis [GW20, Cap. 10].

Colour analysis determines the chromatic composition of a frame, enabling synchronisation between screen content and ambient lighting. Systems that map dominant colours to RGB LEDs help create immersive environments. The use of colour sensors in cyber-physical systems has been studied in the context of real-time image segmentation based on colour similarity in RGB space, which is applicable in interactive environments with dynamic lighting [XSea18].

[15] https://learn.adafruit.com/adafruit-neopixel-uberguide

[16] https://learn.adafruit.com/adafruit-dotstar-leds

[17] https://learn.adafruit.com/rgb-led-strips/current-draw, https://www.ledyilighting.com/the-ultimate-guide-to-addressable-led-strip/#elementor-toc__heading-anchor-12

Image segmentation divides a frame into meaningful regions using methods such as thresholding, k-means clustering or deep learning [PP93]. This facilitates the assignment of different LED groups to specific areas. Recent approaches also decompose lighting into direct and indirect components to improve spatial accuracy [MSea19].

These techniques form the basis of intelligent lighting systems that adapt to visual stimuli and constitute the foundation for the pixel mapping methods discussed in the next section.

3.2.2 Video-to-LED Mapping: Approaches and Algorithms

Once the relevant visual information has been extracted from the video analysis, it needs to be transformed into useful commands for LED systems. This process is divided into three levels: the conceptual mapping model, the pixel mapping logic and the algorithm used to execute this mapping.

Conceptual models of LED mapping–video

Mapping models define how video pixels are associated with physical LEDs, depending on the installation topology and intended effects (see Figure 3.5):

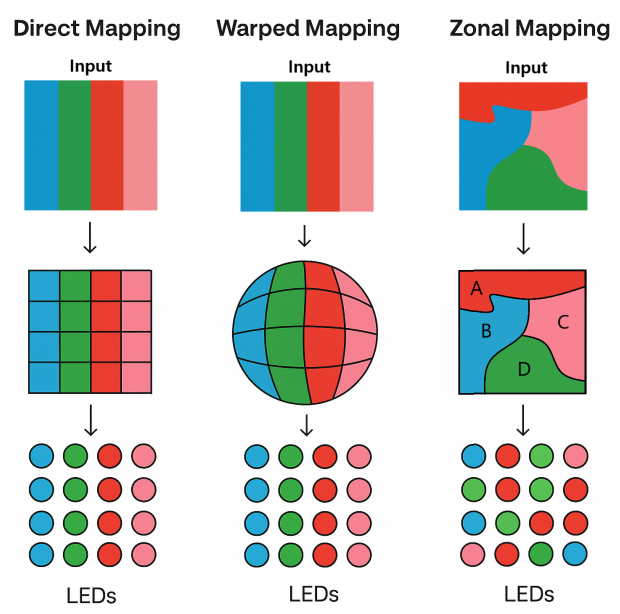


Figure 3.5: Conceptual LED mapping models: direct 1:1 (left), warped for irregular surfaces (centre), and zonal mapping by image regions (right).

- **Direct Mapping (1:1)**: Each LED represents exactly one pixel of the video. It is useful when the video resolution and the arrangement of the LEDs match.

$$\text{LED}(i, j) = \text{Pixel}(i, j)$$

- **Calibrated or Warped Mapping**: It is applied when LEDs are placed on non-planar or irregular surfaces. A geometric transformation (e.g., homography) adapts the content to physical coordinates [BEea07, Maj04].
- **Zonal Mapping (by Regions)**: The image is divided into regions (LED groups). Common in adaptive systems like segmented automotive headlights [WMea21].

Pixel mapping logics

Mapping logic defines how video pixels are associated with physical LEDs, depending on the spatial layout of the installation. The main approaches are:

- **Linear Mapping:** Frequently used in LED strip installations arranged as one-dimensional matrices. Each LED corresponds to a fixed horizontal or vertical strip of the video, and receives the average colour or a representative pixel from that region.[18]
- **Snake (Zigzag) Scan:** Common in matrix-like LED panels where strips are wired in alternating directions, allowing efficient allocation without rewiring. This pattern requires handling row direction inversions and is analogous to scanning strategies used in 2D imaging systems [VNea20].
- **Coordinate-Indexed Access:** The LED layout is stored as a two-dimensional matrix where each `(x, y)` coordinate maps to a video pixel or region. This approach is especially effective when the LED arrangement matches or is proportional to the resolution of the source content.[19]
- **Spatial Proximity Mapping:** In non-uniform or three-dimensional LED layouts (e.g., light sculptures, curved surfaces, façades), spatial structures like Voronoi diagrams or k-d trees assign each LED to its closest pixel in the projection space. This ensures that each emitter reflects the dominant colour of its surrounding visual region [LG07, CXea21].

Algorithms for efficient video processing

Transforming video into LED control signals efficiently requires mapping colour data and optimising visual quality, speed, and system responsiveness. Several strategies are employed in real-time lighting systems:

- **Bilinear and bicubic interpolation**: Applied when adapting video to lower-resolution LED matrices (e.g., projecting HD onto a 16×16 grid). Bilinear interpolation computes the average of four neighbouring pixels [PM17], while bicubic uses sixteen for smoother transitions and greater detail retention [AFea18]. The latter improves edge definition and spatial accuracy at the cost of higher processing load.

[18] `https://learn.sparkfun.com/tutorials/addressable-led-strip-hookup-guide/all`

[19] `https://www.hackster.io/news/displaying-patterns-on-an-irregular-led-matrix-cb0cb4e8bfc6`

- **Spatial Clustering (K-means, DBSCAN)**: These unsupervised methods group pixels by visual similarity (e.g., colour or intensity) to reduce content complexity while preserving coherence. K-means is suited to regular regions [JMea99], while DBSCAN excels in irregular or noisy scenes [EKea].
- **Active Region Mapping (ROI)**: The concept of regions of interest (ROI) enables the selective processing of visual content by focusing on particularly noticeable areas, such as faces or moving objects, which are detected via background subtraction or saliency models [PHMea22]. By ignoring peripheral areas, it reduces computational load and enhances the expressiveness of the LED output in perceptually meaningful regions.

Real-time performance in large installations requires low-latency algorithms and optimised implementations, which often use parallel computation to maintain fluidity.

Time synchronisation between video and LEDs

Accurate synchronisation between video frames and the LED output is crucial in order to avoid visual artefacts, such as flickering, latency and desynchronisation. Several strategies are commonly employed for this purpose:

- **Timestamps (Presentation Timestamps – PTS)**: Timed video streams contain timestamps that indicate when each frame should be displayed. These markers ensure that the LED output remains in sync with the temporal flow of the video [Tek14].
- **External events**: Sensors (e.g., depth cameras, accelerometers) can trigger lighting changes in response to gestures or movement, turning the LED system into a reactive interface [Zou20].
- **Network protocols such as Art-Net and sACN**: To synchronise multiple universes or distributed devices, Art-Net uses the `ArtSync` packet, which coordinates simultaneous playback across nodes [Art24]. sACN includes built-in priority and timing mechanisms [Ent18].
- **Frame buffers**: Intermediate buffers store RGB data before emission, regulating output rate and preventing inconsistencies or overlaps between frames [GW20, Sec. 1.5].

To ensure smooth and coherent playback, the chosen strategy must be adapted to the system's hardware capabilities and the real-time constraints of the content.

3.2.3 Software Tools for Video-to-LED Mapping

LED video mapping software tools enable visual content to be transformed into control signals compatible with protocols such as DMX512, Art-Net, SPI and sACN. Typically, these applications allow users to import video files, define output regions, map LEDs spatially, and configure synchronisation parameters. While some tools prioritise ease of use

through graphical interfaces, others offer advanced features oriented towards professional workflows.

MadMapper: Professional tool for LED mapping

MadMapper is a widely used tool for creating interactive audiovisual installations involving LED mapping. It enables users to import visual content and assign regions of videos or images to physical LED fixtures via an intuitive graphical interface, as shown in Figure 3.6 [Gar24a].

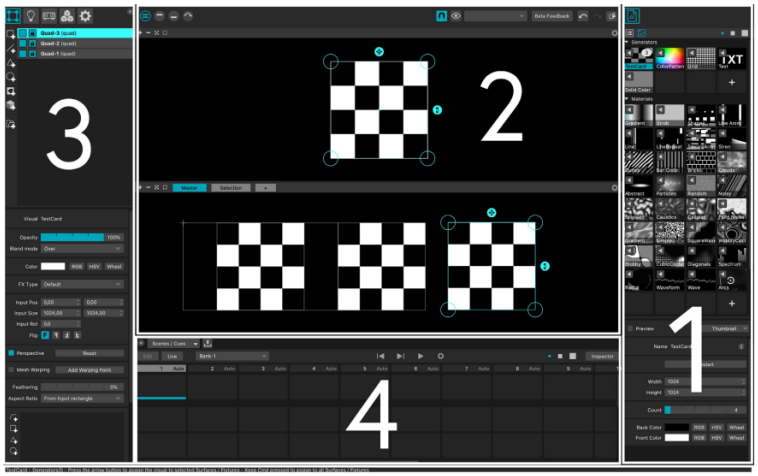


Figure 3.6: MadMapper interface for mapping visual content onto LED surfaces [Gar24a].

It supports the DMX512, Art-Net and sACN protocols and allows multiple LED universes [Gar24a] to be controlled with real-time synchronisation via MIDI, OSC and external sensors [Gar24b, Gar24d]. Real-time effects can be created using OpenGL shaders written in GLSL, leveraging GPU acceleration [Gar24c]. This makes it suitable for dynamic and low-latency environments.

While MadMapper excels in visual flexibility and user interface design, it is not intended for complex logic or conditional processing. Such tasks are better addressed by node-based systems like TouchDesigner. Nevertheless, MadMapper has become a standard in concerts, exhibitions and interactive scenography where coordinated control of lighting and media is essential.

Its project gallery documents professional works in over 40 countries, showcasing its widespread adoption in the creative industry[20]. Moreover, its Instagram account offers a wide selection of recent installations with visual examples and real setups [21].

[20] `https://madmapper.com/gallery/`
[21] `https://www.instagram.com/mad_mapper`

Resolume Arena

Resolume Arena is a real-time video mixing platform widely used by VJs and audiovisual performers. Though not designed specifically for LED mapping, it supports Art-Net output, enabling the assignment of visual "slices" to LED universes or physical outputs [22].

Its layered architecture makes it easy to combine video clips, effects, and generative content. Integrated audio analysis tools allow you to synchronise visuals with rhythm and frequency. Resolume supports MIDI, OSC, FFGL and GLSL, as well as real-time texture sharing via Syphon on macOS and Spout on Windows.

While it is powerful for screen-based manipulation, its LED mapping capabilities are more limited than those of dedicated tools such as MadMapper. An overview of its interface is shown in Figure 3.7.

Figure 3.7: Resolume Arena interface showing layered clips, effects, and the output monitor. Source: `https://resolume.com/software`

TouchDesigner

TouchDesigner is a node-based visual programming environment designed for use in interactive installations, real-time performances and advanced multimedia systems. Unlike tools that focus solely on video output, TouchDesigner allows the integration of sensor data, audio inputs and conditional logic, and can be controlled via Art-Net, DMX512, sACN, MIDI, OSC or NDI [23].

Thanks to its modular architecture, dynamic data flows can be constructed using nodes (operators) that perform specific tasks, such as transformation, filtering or rendering. This flexibility supports the creation of generative content and real-time interaction. Figure 3.8 shows an example of the interface.

[22] `https://resolume.com`
[23] `https://docs.derivative.ca/Main_Page`

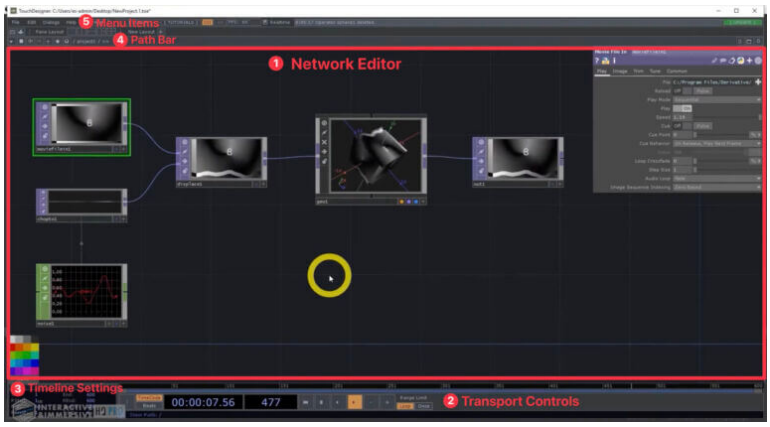


Figure 3.8: TouchDesigner interface showing its node-based programming environment and modular structure.
Source: `https://interactiveimmersive.io/touchdesigner-user-interface/`

TouchDesigner is especially suitable for complex, responsive environments and data-driven visual systems [Li24]. It supports Python scripting, GLSL shaders, and protocols such as Spout, Syphon, OSC and MIDI. Although it is powerful, its complex nature makes it more suitable for those with technical experience in multimedia engineering or creative coding.

QLC+

Q Light Controller Plus (QLC+) is a free, open-source lighting control application compatible with DMX512 and Art-Net [oL23]. It is structured around functional views such as Functions, Devices, Inputs/Outputs and the Virtual Console, supporting scene programming, live execution and real-time control.[24]

Although it was not designed for video mapping, QLC+ can control addressable LED strips, moving heads and RGB fixtures with high temporal precision. It supports MIDI, OSC, USB-DMX and audio-based triggers and enables users to create custom control widgets on the Virtual Console.

Thanks to its open-source nature, QLC+ is particularly useful for educational purposes, low-budget projects and permanent installations, where the primary goal is not audiovisual integration. An example of its interface is shown in Figure 3.9.

[24] `https://www.qlcplus.org`

Figure 3.9: QLC+ Virtual Console with user-defined widgets for live DMX lighting control. Source: `https://www.qlcplus.org`

Other Open-source Solutions

Several open-source tools offer flexible LED control to users with programming knowledge. This makes them ideal for educational purposes, generative art, prototyping and small-scale installations:

- **FastLED:** C++ library for microcontrollers like Arduino or ESP32, supporting addressable strips (WS2812, APA102, SK6812) with animation and brightness control.[25]
- **rpi_ws281x:** C library for Raspberry Pi, enabling LED control via PWM or SPI. Compatible with Python/C++ for high-volume LED arrays.[26]
- **WLED:** Firmware for ESP8266/ESP32, offering wireless LED control via web interface, app, Art-Net, E1.31, MQTT, or Alexa.[27]

While lacking graphical interfaces, these tools stand out for their low cost, adaptability and integration with custom-coded systems.

Table 3.3 compares the tools and libraries discussed above in terms of usability, protocol compatibility, extensibility, and typical application context.

[25] `https://fastled.io/`
[26] `https://github.com/jgarff/rpi_ws281x`
[27] `https://kno.wled.ge`

Software / Library	Ease of use	Protocol support	Extensibility	Typical use
MadMapper	High (graphical interface)	Art-Net, DMX, sACN, OSC, MIDI	Moderate (OSC, modules, shaders)	Professional LED installations, mapping
Resolume Arena	High (VJ-oriented)	Art-Net, OSC, MIDI, Syphon/Spout	Limited (effects and clips)	VJ performances, audio-visual effects
TouchDesigner	Medium-Low (node-based environment)	Art-Net, DMX, OSC, MIDI, NDI	High (Python, shaders, custom nodes)	Complex interactive systems
QLC+	Medium (technical lighting interface)	DMX, Art-Net, OSC, MIDI	Limited (open-source)	Theatrical and architectural lighting
FastLED	Low (code-based)	N/A (direct control)	High (fully programmable)	Educational and generative art
rpi_ws281x	Low (code-based)	N/A (PWM/SPI)	High (script integration)	Data visualisation, custom Pi setups
WLED	High (web/app interface)	Art-Net, E1.31, MQTT, Alexa	Medium (API and presets)	WiFi installations, creative lighting

Table 3.3: Comparison between software tools and libraries for addressable LED control.

As summarised in Table 3.3, tool selection depends on project scale, user expertise and functional requirements. MadMapper and Resolume both offer intuitive interfaces for live performance contexts. TouchDesigner, on the other hand, enables modular, logic-based control for complex setups. QLC+ is ideal for stage lighting, while FastLED, rpi_ws281x and WLED are versatile, low-cost solutions for custom-coded or embedded systems.

3.3 Software Engineering Applied to Lighting Control Systems

3.3.1 Software Architectures in Real-Time Control Systems

Real-time systems are essential when the correctness of a system depends on both the output and the timing. They are generally classified as *hard* or *soft real-time*, depending on how critical it is to meet deadlines [LO11].

LED lighting control is usually soft real-time, meaning that small delays will not cause failures, but may reduce visual quality. In live or interactive contexts, however, maintaining low latency is crucial to avoid desynchronisation. Therefore, architectural decisions must consider not only functionality, but also concurrent execution, device synchronisation and responsiveness to time-sensitive events [Bab12].

Characteristics of Real-Time Systems Applied to Light Control

Designing an LED-based lighting system requires adopting core characteristics of real-time systems (RTS). This is particularly important for the ability to respond to events with low latency [LO11]. The most relevant functional and non-functional requirements are outlined below:

Temporal determinism

Temporal determinism is particularly important in stage and architectural lighting. A deterministic system ensures that each operation is completed within a known maximum time, even under heavy load. In this context, scenes, effects or colour transitions must occur precisely at the scheduled time. Flickering, delays or desynchronisation between DMX universes must be avoided [LO11].

Multi-source integration and real-time response

Modern lighting systems operate as cyber-physical environments, integrating multiple data streams (video, sensors, audio and manual inputs) in parallel. In order to maintain perceptual synchronisation and visual coherence, these heterogeneous signals must be processed with low latency. Common input sources include:

- **Real-time video**: Visual content (e.g., RGB pixels) is mapped to lighting commands via temporary buffers and refresh-rate synchronisation [GW20].
- **Sensors (light, motion, sound)**: In interactive contexts, lighting responses are adapted based on sensor inputs. Technologies such as I2C, SPI and MQTT facilitate the acquisition of real-time data that triggers scene changes [KDB16].
- **User Interfaces**: From consoles to web applications. The user has direct control of the system, especially in live contexts.
- **Audio or music**: FFT-based analysis and peak detection align light effects with musical rhythm and dynamics [LO11].

To ensure parallel, real-time processing, several **architectural mechanisms** are typically employed:

- **Concurrent buffers**: Temporary storage that supports simultaneous read/write operations, preventing data loss [LO11] [GW20, Sec. 1.5].
- **Device synchronisation**: Mechanisms such as synchronised clocks or event triggers coordinate multiple inputs [LO11].
- **Priority Planning**: Priority queues ensure time-critical events are processed preferentially [LO11].
- **Parallel execution threads**: Handling external events in separate threads prevents blocking the main loop and reduces scheduling overhead [LO11].

- **Event buses**: Intermediate distribution layers decouple producers (e.g., sensors) from consumers (e.g., output modules), improving scalability and modularity.

It is this capacity for coordinated multi-source integration that enables the creation of immersive and adaptive lighting for both live performances and interactive installations [Bab12, LO11].

Architectural models used in real-time lighting control

Modern lighting control systems, particularly those intended for interactive or high-resolution environments, are increasingly adopting distributed architectures. These systems achieve greater scalability, fault tolerance and proximity to actuators by delegating computation to autonomous modules rather than relying on a central controller [TVS06]. Four architectural styles are illustrated in Figure 3.10:

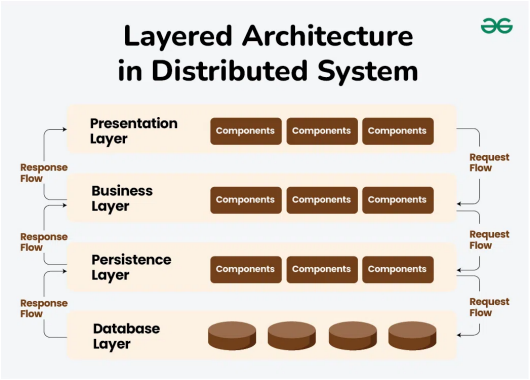
- **Layered**: Divide functionality into hierarchical levels (e.g., interface, logic, hardware abstraction), enabling clear separation between user-facing elements (e.g., dashboards) and low-level protocols (e.g., Art-Net, DMX) [TVS06].
- **Object-based architectures**: Components act as self-contained objects that interact with each other via APIs or RPC mechanisms. This is ideal for modular lighting systems, where controllers and visualisation modules can coordinate flexibly. Service-Oriented Architectures (SOAs) are a common implementation that enables reuse and interoperability in heterogeneous environments [TVS06].[28]
- **Data-centred**: It relies on a shared repository or publish-subscribe middleware. In lighting, subsystems can access shared timelines, buffers or animation banks either synchronously or asynchronously [TVS06].
- **Event-based**: Modules operate independently and communicate asynchronously, reacting to stimuli such as sensor inputs or video frames. Processing is only triggered when needed, ensuring low latency, energy efficiency, and loose coupling [TVS06].

Due to the interactive and time-sensitive nature of LED animations, event-based architectures are a particularly good fit. These architectures provide immediate responsiveness and integrate seamlessly with interrupt-driven or observer-based software patterns, which are commonly used in sensor-enhanced lighting systems [LO11].

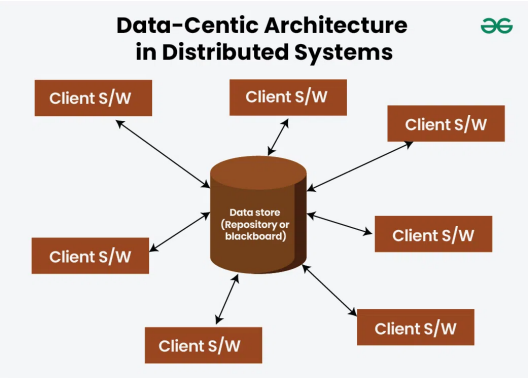
Processing Models: Pipelines and Parallel Processing

Interactive lighting systems greatly benefit from *pipelines*, where data flows through sequential stages (acquisition, analysis and rendering) allowing each stage to process different data elements concurrently. This technique, known as **pipeline parallelism**, enables

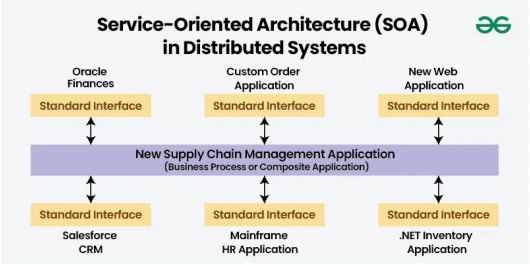
[28] `https://www.geeksforgeeks.org/architecture-styles-in-distributed-systems/`



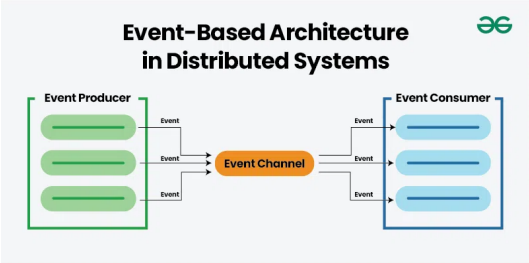
(a) Layered architecture



(b) Data-centered architecture



(c) Object-based architecture (represented via SOA)



(d) Event-based architecture

Figure 3.10: Architectural models in distributed lighting control systems.
Images from *GeeksforGeeks: Architecture Styles in Distributed Systems* `https://www.geeksforgeeks.org/architecture-styles-in-distributed-systems/`.

continuous throughput with minimal latency and is widely used in high-rate vision systems. [CJ09].

For instance, Sabater et al. [SBea17] implemented a multi-stage pipeline for light-field video processing, including geometric calibration, colour homogenisation, depth estimation, and rendering. As illustrated in Figure 3.11, pipeline and data parallelism can be combined to process high-throughput pixel data efficiently.

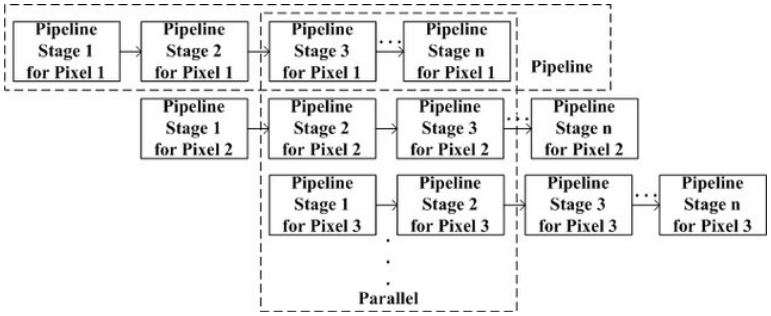


Figure 3.11: Combined pipeline and data-parallel processing for pixel-wise operations. Each pixel progresses through a multistage pipeline, allowing concurrent processing at different pipeline stages. Image from *A Parallel Reconfigurable Architecture for Real-Time Stereo Vision* [CJ09].

For large-scale LED arrays, *data parallelism* becomes essential. Frames are divided into smaller blocks processed in parallel across threads or cores, distributing computational load and reducing latency.

This strategy is exemplified in Guthe et al.[GWea02], where volumetric datasets are split into cubic blocks that are compressed, decompressed, and rendered concurrently using GPU-based texture mapping:

> *"We divide the data sets into cubic blocks... we apply the wavelet filters to each block... This results in a lowpass filtered block and wavelet coefficients... We render these blocks using hardware texture mapping..."* [GWea02].

These models significantly enhance the performance and responsiveness of real-time lighting, enabling continuous video or sensor input to be transformed into precise control signals [SBea17].

Handling large volumes of data in real time

In reactive lighting systems with hundreds or thousands of output channels, it is essential to manage high-throughput data streams with minimal latency. In addition to the architectural model, specific optimisation techniques are required to ensure stable and timely performance [LO11]. Common strategies include:

- **Downsampling:** This reduces computational load by adapting the resolution of the source content (e.g., a 20×16 video) to match the LED array, preserving spatial correspondence. Some learning-based algorithms retain essential spatial and temporal patterns, enhancing visual coherence [XTea22].
- **Ring buffers**: Fixed-size circular buffers allow concurrent reads and writes, avoiding blocking and ensuring consistent data flow in animations or audio-reactive lighting [Tan09].
- **Task parallelisation**: Reading, processing and transmission are carried out in separate threads. Synchronised queues or semaphores prevent race conditions and ensure deterministic execution [SBea17, GWea02].
- **Compact data structures**: Using binary arrays, vectorised buffers (e.g. NumPy) or memory-aligned blocks in C/C++ minimises memory usage and accelerates computation, which is key for high-frame-rate scenarios [The25].

These techniques minimise the delay between input and output, ensuring smooth and synchronised performance even under high load, and avoiding visual artefacts such as flickering or glitches [LO11].

3.3.2 Development of User Interfaces for LED Animation Configuration

User interfaces (UIs) in programmable lighting systems combine technical capabilities with creative intent. They enable users to configure animations, colour schemes, timing and spatial mappings, thereby simplifying the design and deployment of lighting effects. Well-designed UIs enhance workflows and facilitate the creation of precise visual compositions.

Tools and languages for interface development

Interfaces for programmable lighting systems are usually created using cross-platform tools that convert user input into control commands such as DMX scenes or RGB values, while also providing clear feedback on the system's status.

The choice of development environment depends on the application type (local or distributed), hardware constraints, and user profile (technician, artist, or operator). The most widely used tools are outlined below:

Qt (C++): A high-performance framework widely used for professional UI development. Its *signal-slot* architecture simplifies event handling and synchronisation, which is ideal for managing DMX universes and animation control panels. [29]

Qt offers a rich set of components (e.g., RGB sliders, timelines, layout managers) suitable for complex lighting interfaces [30]. Several specialised modules support real-time, multisensory installations:

- **Media and Sensor Integration**: Modules such as `Qt Multimedia` and `Qt Sensors` support video, audio and environmental input [31].
- **Hardware Communication**: `Qt Serial Port` and `Serial Bus` provide communication with hardware through serial ports or industrial buses. They enable integration with DMX512 and Art-Net controllers [32].
- **Data Visualisation**: `Qt Charts` and `Qt Data Visualization` offer real-time plots of intensity levels and sensor inputs [33].
- **Animation and Remote Control**: `Qt Quick Timeline`, `Lottie Animation`, and `Remote Objects` allow the generation of keyframe-based animations and distributed control [34].

[29] https://doc.qt.io/

[30] https://doc.qt.io/qt-6/index.html

[31] https://doc.qt.io/qt-6/qtmultimedia-index.html; https://doc.qt.io/qt-6/qtsensors-index.html

[32] https://doc.qt.io/qt-6/qtserialport-index.html; https://doc.qt.io/qt-6/qtserialbus-index.html

[33] https://doc.qt.io/qt-6/qtcharts-index.html; https://doc.qt.io/qt-6/qtdatavisualization-index.html

[34] https://doc.qt.io/qt-6/qtquicktimeline-index.html; https://doc.qt.io/qt-6/qtlottieanimation-index.html; https://doc.qt.io/qt-6/qtremoteobjects-index.html

- **Web and Audio Extensions**: `Qt WebSockets`, `WebChannel`, `TextToSpeech` and `Spatial Audio` support remote control via browser and immersive audio design [35].

Thanks to its modular design and extensive ecosystem of libraries, Qt remains a versatile platform for developing responsive, real-time lighting interfaces.

JavaFX: This is a Java-based framework for building modern user interfaces. It features a hierarchical *scene graph*, declarative animations and integration with sensors or external devices via third-party libraries. [36]

Thanks to its cross-platform architecture, it can be deployed on desktops, embedded systems and touch-enabled devices. This makes it suitable for educational tools, custom LED configurators and visually interactive environments. [37]

Tkinter and PyQt: Both are popular for prototyping in Python. `Tkinter`, included in the standard library, offers limited graphical capabilities but allows quick development of simple tools with minimal code [38].

By contrast, `PyQt` [39] offers full access to the Qt ecosystem from Python, enabling the development of sophisticated UIs without leaving the interpreted environment. Its compatibility with libraries like `pyartnet` and `OLA` makes it particularly effective for LED and DMX control [40].

Figure 3.12 shows a PyQt interface for controlling LEDs and a servo via Arduino. It shows real-time interaction without requiring Qt Designer.

PyQt interfaces can incorporate background processing using `QThread` [41], enabling responsive UIs even with ongoing Art-Net communication. Its modular structure supports scalable and reusable component design.

Web frameworks (React, Electron): These are well-suited for distributed or multi-device interfaces. Combining a Python backend (e.g., `Flask` or `FastAPI`) with a React or Vue frontend enables interactive dashboards for real-time control via browsers, which is ideal for multi-user environments or LAN/WiFi access without installation [42].

With `Electron`, these interfaces can be packaged as cross-platform desktop apps for Windows, macOS, and Linux, maintaining a single codebase while avoiding browser de-

[35] `https://doc.qt.io/qt-6/qtwebsockets-index.html`; `https://doc.qt.io/qt-6/qtwebchannel-index.html`;`https://doc.qt.io/qt-6/qttexttospeech-index.html`; `https://doc.qt.io/qt-6/qtspatialaudio-index.html`

[36] `https://docs.oracle.com/javase/8/javase-clienttechnologies.htm`

[37] `https://openjfx.io/`

[38] `https://docs.python.org/3/library/tkinter.html`

[39] `https://www.riverbankcomputing.com/software/pyqt/intro`

[40] `https://pyartnet.readthedocs.io/en/latest/pyartnet.html`; `https://www.openlighting.org/ola`

[41] `https://doc.qt.io/qt-6/qthread.html`

[42] `https://react.dev/`

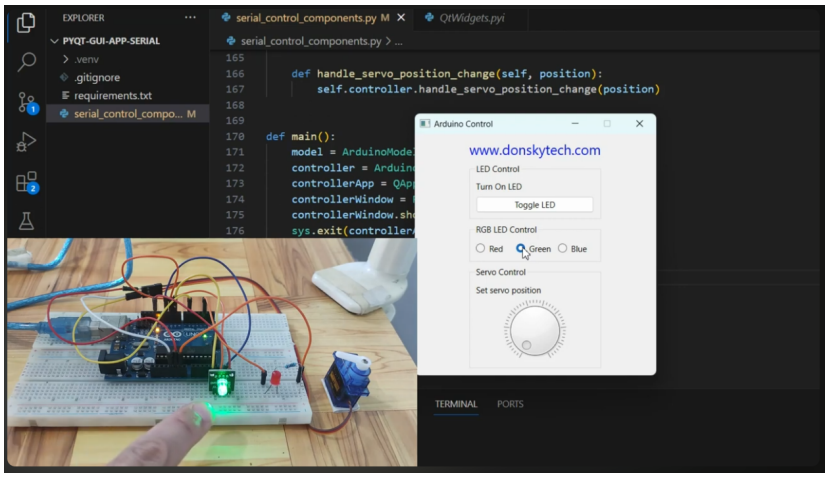


Figure 3.12: GUI built with PyQt for LED and servo control via Arduino.
Screenshot from: *DonskyTech, YouTube* `https://www.youtube.com/watch?v=5uqwzu8wT3A`.

pendency [43]. This simplifies distribution, updates, and compatibility across heterogeneous systems [44].

When choosing between web-based and native frameworks, it is important to consider factors such as latency tolerance, the target environment, graphical complexity and long-term maintainability.

Usability Principles in Lighting Control Interfaces

Jakob Nielsen's usability heuristics [Nie94] offer a solid foundation for designing lighting control interfaces, particularly in real-time contexts. These principles are particularly important for ensuring precision, low latency and adaptability in LED installations of different sizes. The most pertinent principles are:

- **System Status Visibility**: Real-time feedback on animations, colour values, and active channels enhances control and situational awareness.
- **Match Between System and the Real World**: Interfaces should use familiar visual metaphors (e.g., sliders, timelines) and terminology (e.g., scene, fade) for intuitive interaction.
- **User Control and Freedom**: Users should be able to edit, pause, or undo actions safely, with options to save and restore presets during live use.
- **Error Prevention**: Validation mechanisms should avoid misconfigurations, such as channel overlaps or out-of-range values.
- **Recognition Rather Than Recall**: Visible, labelled options should be favoured over memorised commands, with tooltips or tags used when helpful.

[43] `https://www.electronjs.org/docs`
[44] `https://rocketbuild.com/native-app-vs-web-app/`

- **Flexibility and Efficiency of use**: Both novice and expert users should be able to customise layouts and shortcuts.
- **Help and Documentation**: Integrated help buttons, hover guides, and visual aids improve user autonomy.
- **Low latency and immediate response**: Immediate visual feedback ensures smooth interaction, requiring efficient backend and frontend coordination.
- **Visual scalability**: Interfaces should scale to different LED setups, supporting zooming, grouping, and layered views.
- **Dark mode and high contrast**: High legibility is essential in dark technical environments to reduce visual fatigue.

Examples of Effective Interfaces in Specialised Software

Various software tools apply usability principles to support intuitive control of LED animations. While prominent solutions like **MadMapper** and **QLC+** have already been discussed (see Sections 3.2.3 and 3.2.3), the following examples illustrate additional specialised interfaces designed for efficient lighting configuration and deployment:

- **xLights**: A free, open-source tool for designing music-synchronised light shows, especially in festive contexts. Its timeline-based interface supports animation sequencing, audio integration, and network configuration via protocols like Art-Net and sACN [45]. A built-in simulator allows previewing installations before deployment.Figure 3.13 illustrates the interface, showing the animation editor, audio waveform, and layout preview.

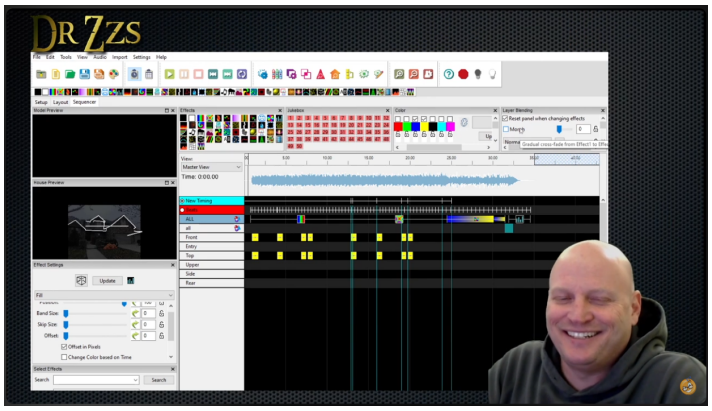


Figure 3.13: The xLights interface, with timeline editor and audio waveform.
Source: *DrZzs & GrZzs*, YouTube `https://www.youtube.com/watch?v=p7wV6A26Gak`.

- **LED Matrix Studio:** This is a minimalist but effective editor for designing static or animated LED patterns. Users can draw pixel by pixel and export data in formats com-

[45] `https://xlights.org`

patible with Arduino and similar platforms, making it ideal for educational purposes and prototyping [46]. Figure 3.14 shows the interface alongside the resulting output on a physical LED matrix.

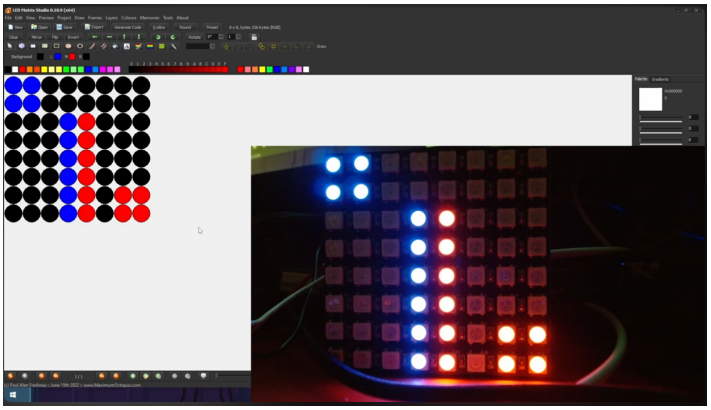


Figure 3.14: LED Matrix Studio interface and physical LED output.
Screenshot from: *Boaztheostrich*, YouTube `https://www.youtube.com/watch?v=fHhKf1lGWx0`.

- **Lightkey**: A macOS application for scenic and architectural lighting, supporting DMX, Art-Net, and MIDI [47]. It offers a visual, user-friendly interface for managing devices and effects, as shown in Figure 3.15.

Figure 3.15: Lightkey interface in a worship environment.
Screenshot from: *Felty Studios*, YouTube `https://www.youtube.com/watch?v=y_ZebL1PAeU`.

Low-level tools such as `FastLED` or `rpi_ws281x` (see Section 3.2.3) are suited to firmware-level control, while graphical platforms like `xLights` or `Lightkey` (see Section 3.3.2) offer intuitive interfaces for live use. Both types play complementary roles and are often integrated in hybrid lighting setups.

[46] `https://sourceforge.net/projects/led-matrix-studio/`
[47] `https://lightkeyapp.com/en/specs`

Chapter 4

Methodology

In this chapter, it is explained how and why this project has followed an agile methodology [Som16]. Furthermore, the final section of the chapter defines the specifications of the used hardware and software resources.

4.1 Development methodology

Software engineering offers various methodological approaches for managing and implementing projects [Som16]. Among these, agile methodologies have emerged as a flexible alternative to traditional models, such as the Waterfall model, which require predefined phases and the full specification of requirements from the outset.

Given the exploratory nature of this project and the use of unfamiliar technologies (Art-Net, Flask, `pyartnet` library), an agile approach was considered more appropriate than a traditional one. The project requires a high degree of flexibility as well as the ability to adapt the implementation strategy in response to new technical constraints or design opportunities. Throughout the development, a number of architectural decisions might be re-evaluated in order to enhance performance and ensure system responsiveness. Such decisions may include coordination between LED playback and video frames, as well as the internal data handling mechanisms. These adjustments will be guided by continuous testing and feedback during weekly review sessions. Agile methodologies promote incremental development, continuous feedback, and iterative decision making, which makes them particularly well suited for solo development in dynamic and partially defined environments.

The final product is intended to be part of a larger system, and its design will be shaped by trial, feedback, and technical discoveries. That is why the project will be developed following an agile mindset, with a Scrum-inspired structure tailored to the specific needs of an internship.

4.1.1 Project management: Scrum framework

Scrum is an agile project management framework that involves working in short cycles, known as Sprints, which last between one and four weeks [SS20]. Although it was originally designed for teams, its principles have been adapted for individual developers.In this case,

weekly iterations were chosen to align with the rhythm of academic meetings. This approach maximises learning opportunities while also limiting the cost of change. Shorter sprints provide quicker feedback loops and reduce the risks associated with long-term planning in uncertain environments [SS20].

To initiate the workflow, the internship is scheduled to begin with two initial meetings: the first to clarify the project's objectives and the second to introduce the technical infrastructure, including the LED lecterns and the current setup. From that point onwards, development is organised around short-term goals reviewed weekly, enabling continuous adaptation of priorities as new challenges arise.

In this structure, the tutor, David Vallejo, takes on a role similar to that of the Scrum Master, providing guidance and helping to define strategies. Meanwhile, Francisco Manuel García, a colleague with in-depth knowledge of the company's systems, acts as the Product Owner, providing technical insight and functional validation. Although formal Scrum events like daily stand-ups, planning poker or retrospectives are not done, the pillars of transparency, inspection, and adaptation are still respected. Occasional feedback from Carlos González, a co-founder of Furious Koalas S.L., who acts as the client throughout the project, will reinforce the collaborative nature of the process.

4.1.2 Software development: agile practices

The development approach followed in this project is aligned with the four core values of the Agile Manifesto [1]:

- **Individuals and interactions over processes and tools**: As a solo developer, my workflow will involve continuous interaction with a tutor and a technical colleague from Furious Koalas S.L. Rather than relying on rigid procedures, decisions will be taken and adapted to emerging challenges.
- **Working software over comprehensive documentation**: Throughout the development, the primary focus will be building a functioning system capable of rendering video content on addressable LEDs rather than producing extensive documentation at an early stage. Technical reports and design descriptions will be prepared once the prototype is stable.
- **Customer collaboration over contract negotiation**: The direction of the work will not dictate by contractual specifications, but rather by ongoing collaboration with the client. Together with the project tutor, they are expected to provide continuous feedback that helps validate technical decisions and ensures that the system evolves according to the actual needs of the quiz platform.

[1] `https://agilemanifesto.org/`

- **Responding to change over following a plan**: Several design changes might be introduced during the project as technical constraints became clearer. The process must remain flexible enough to accommodate these decisions without compromising the project's overall coherence.

4.1.3 Work planning

Although there is no fixed initial planning, the project is structured into a set of work packages that guide its development. These packages are progressively defined based on feedback, technical exploration, and implementation needs. Each unit of work corresponds to a functional milestone that contributes to the gradual construction of the system.

1. **System exploration and tool research**. The initial phase involves a thorough analysis of the architecture of the existing quiz platform, as well as an examination of the relevant protocols and technologies. These include Art-Net, the PIX CONTROL 16 controller, and Python-based web development tools such as `Flask`.
2. **LED control logic and content mapping**. Once the system is understood, the next step focusses on implementing the logic for controlling the LED lecterns and developing mapping algorithms to project static and dynamic content onto the physical LED layout.
3. **Back-end development and API design**. A web server is developed using Flask to expose endpoints for video playback and LED control. The API is designed to support different playback modes and facilitate communication with external components.
4. **User interface and video preview**. A front-end interface is implemented so that users can preview and select videos for projection.
5. **System integration and coordination**. The LED control system is integrated with the quiz main system by adapting request formats and containerising the solution.
6. **Refinement, optimisation and validation**. The final phase involves performance tuning, synchronisation adjustments, and system validation under realistic conditions. Continuous testing ensures that the solution behaves reliably and meets its functional goals.

These stages are also represented graphically on the Gantt chart in Figure 4.1, which provides an overview of the expected duration and sequencing of each work package throughout the 16-week development period.

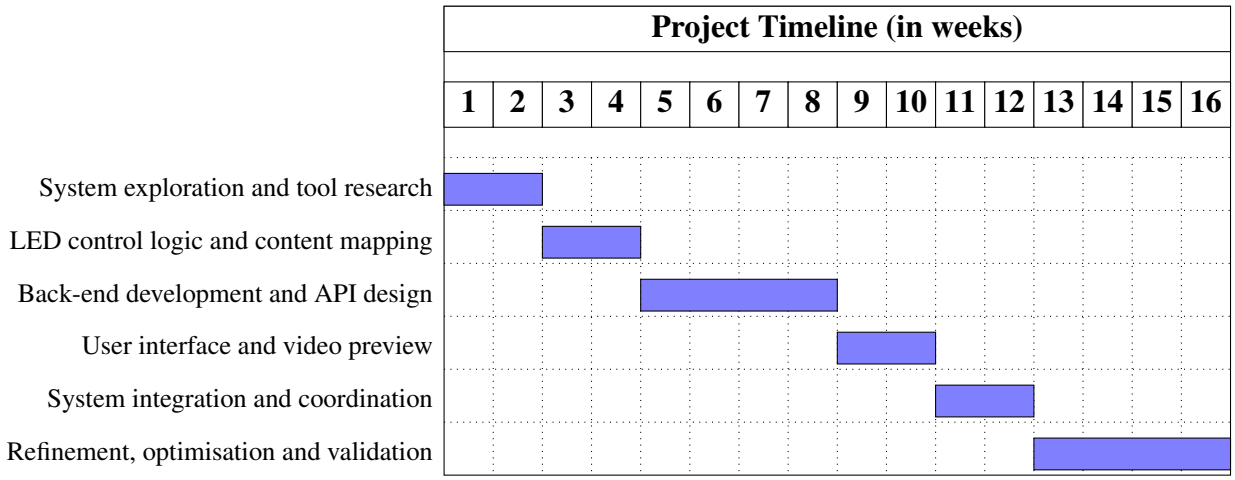


Figure 4.1: Estimated work plan for the project

4.2 Development workflow

Throughout the project, the version control system used will be Git [2], with the remote repository hosted on Bitbucket [3]. The branching model to be adopted will be inspired by the Git Flow methodology but simplified for the needs of a single developer. The main structure includes a "main" branch containing the first stable release of the system. In parallel, the 'develop' branch can be used to integrate ongoing work. Several feature branches will be created from the 'develop' branch, each dedicated to a specific functionality or module (e.g. video processing, video previewing, user interface). Once completed, these feature branches will be merged back into the 'develop' branch as required.

Although no strict commit message convention will be enforced, the use of feature branches will help maintain clarity and modularity in the development process. This workflow provides enough flexibility and organisation for individual projects, ensuring the clean integration of new functionalities while keeping the production branch stable.

4.3 Hardware and software resources

This section describes the hardware components, operating systems, programming languages, tools, and libraries that have been used throughout the development of the project.

4.3.1 Hardware resources

- **My Personal Computer**: The project will be fully developed on my laptop, which is powerful enough to run several threads simultaneously. It has an Intel Core i7-14650HX processor and 32 GB of RAM.

[2] `https://git-scm.com`
[3] `https://bitbucket.org`

- **ITSI computer**: This computer will be used to understand the entire system I am given, including how to start up the hardware and software components, control the LEDs with Madmapper and analysing the universe mapping of the LED strips. The desktop computer belongs to Furious Koalas S.L., whose offices are located in the Instituto de Tecnologías y Sistemas de la Información (ITSI) building.
- **Deskontroller 16**: This controller supports up to 16 universes and a maximum of 8,192 Art-Net channels. The device receives lighting instructions via the Art-Net protocol over a network and converts them into SPI (TTL) signals suitable for addressable LED strips.
- **Addressable RGB LED Strips (LED RGB DIG 30PIX, 12V)**: The lighting elements the system is currently using are addressable LED strips based on the WS2811 chip, which operate at 12V and are designed to control groups of RGB LEDs through an external driver. The WS2811 chip controls three LEDs per pixel, but the strips are commercially referred to as having 30 addressable "pixels" per metre. They are compatible with controllers that use digital signalling protocols, such as SPI or Art-Net, via the appropriate interface. In total, there are 1,170 LEDs, distributed equally between three lecterns.

4.3.2 Operating systems

- **Windows 11**: Both the development and testing environments will be based on Windows 11. As there is no specific requirement for an alternative operating system, I opted for Windows 11 because I am more familiar with it. All hardware components, drivers, and software tools will be throughout the project were fully compatible with Windows 11, ensuring a stable and efficient workflow.

4.3.3 Software resources

Programming languages

- **Python**: This is the language I will use to develop the system's back-end. I chose Python for several reasons: its simplicity and readability facilitate rapid development; the Flask framework is based on Python; and there are specific libraries available for Art-Net communication, such as OLA (Open Lighting Architecture) [4] and `pyartnet`.
- **TypeScript**: This language will be used to develop the logic of the user interface. This is due to recent experience with TypeScript in previous projects, as well as the need to organise the system with a clear distinction between front-end and back-end code. Although both parts are going to be located in the same repository, they will be organised into separate folders within the project. Flask will manage the back-end and provide the API, while the front-end will be built using Node.js and TypeScript for better modularity and development efficiency.

[4] `https://www.openlighting.org/ola/`

- **HTML**: HyperText Markup Language will be used to structure the content of the front-end web pages. It will determine the layout and elements displayed to users in the graphical interface of the LED animation management.
- **CSS**: Cascading Style Sheets will be used to define the visual appearance of the front-end interface, including colour schemes, spacing, fonts, and responsive design. This ensures a consistent and user-friendly visual experience.

Development tools

- **Visual Studio Code**: It is a cross-platform source code editor developed by Microsoft. Its key features include a lightweight design, extensibility through plugins, and support for multiple programming languages. In this project, it will be used as the primary editor for both the back-end (Python) and the front-end (TypeScript, HTML, and CSS).
- **Git**: It is a free, open source version control system that allows you to save and track changes to source code. It also enables you to access the entire history of those changes and create branches to isolate feature development. Git will be used throughout this project to maintain a consistent development workflow.
- **Bitbucket**: Bitbucket is a Git-based repository hosting service. It offers additional features, such as issue tracking and access control. For this project, Bitbucket is going to be the only repository for managing the source code, providing version control and backups throughout the development process.
- **Docker** [5]: Docker is a containerisation platform which uses Linux kernel features, such as namespaces and cgroups, to create containers, that are lightweight, isolated environments. In this project, Docker will be used to encapsulate the LED control system in its own container, enabling it to connect to the internal network of the main contest server, which also runs in a separate container.
- **Node.js**: Node.js is a JavaScript runtime environment built on the V8 engine. In this project, it will be used to manage and compile the front-end code written in TypeScript. It also provides access to a wide range of packages via its default package manager, `npm`, which is crucial for installing and configuring the front-end toolchain.

Project planning and design tools

- **Microsoft Planner**: This is a web-based task management application that is integrated into Microsoft 365. It will be used to track project progress, define tasks, and maintain visibility of the overall development timeline. It facilitates personal organisation and task prioritisation throughout the different stages of the project.
- **Visual Paradigm**: This software design and modelling tool supports the creation of UML diagrams, flowcharts, and system architecture models. In this project, it will be

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

used to design system diagrams and illustrate architecture, data flow, and component interactions.

- **Figma**: It is a collaborative design platform primarily used for prototyping user interfaces. In this project, it will be used to design and preview the layout and behaviour of the web interface prior to implementation. Using Figma helps validate the user experience (UX) and refine the visual structure of the front-end.

Software libraries and frameworks

- **Flask (v3.1.0)**: This is a lightweight Python web framework that will be used to implement the back-end server. Provides a REST API that enables external clients or user interfaces to activate LED animations and interact with the system.
- **pyartnet (v1.0.1)**: This is a Python library that facilitates the transmission of lighting control data via the Art-Net protocol. It will be used to communicate with the PIX CONTROL 16 controller, enabling LED strips to be controlled in real time.
- **numpy (v2.2.3)**: It is a library for numerical computation and array manipulation. It will be to handle RGB data arrays extracted from video frames and to perform operations on them efficiently.
- **opencv-python (v4.11.0.86)**: This is a widely used computer vision library. In this project, it will be used to load videos and extract individual frames for LED colour mapping.
- **pillow (v11.1.0)**: This is a Python imaging library used for handling and manipulating static images. In this project, it will be to extract a single frame from a video and convert it into a static image.
- **pytest (v8.3.5)**: It is a testing framework used to implement unit and integration tests. It will allow for automated verification of the system's internal components and API routes.
- **pytest-asyncio (v0.26.0)**: This is a plugin for `pytest` that supports the testing of asynchronous code. It is essential for testing coroutines and `async def` functions that will be used in the system.
- **pytest-flask (v1.3.0)**: This is an extension of `pytest` made for Flask applications. It will be used to test the behaviour and accuracy of the web routes defined in the back-end.
- **pytest-cov (v6.1.1)**: This is a plugin for measuring code coverage during testing. It will be used to determine which parts of the application are covered by the test suite and to identify any untested code paths.

Documentation tools

- **Overleaf**: It is a widely used collaborative online LaTeX editor for writing technical and scientific documents. In this project, Overleaf will be used to write the entire final thesis, ensuring correct referencing, a consistent document structure, and high-quality typography.

Chapter 5

Architecture

In this chapter, a technical discussion of the system's internal structure is provided. It details its main components and explains how they interact to ensure synchronised playback of audiovisual content on LEDs. First, a general overview of the system is given, and then the most important subsystems are described more in depth. These descriptions highlight the most relevant design decisions and their impact on the efficiency, robustness, and scalability of the system.

5.1 General overview

This system addressed in this project is a functional subsystem within a larger question-based quiz control system, the *Quiz Main System*. Although this project focuses exclusively on the design and development of the LED Management Subsystem, it is important to highlight that its functionalities are used in a larger framework, developed by the company Furious Koalas S.L., where it operates in coordination with other subsystems.

The interaction and integration between this subsystem and the rest that compose the *Quiz System* are described more deeply in the deployment and communication sections (see Section 5.7).

Figure 5.1 provides a high-level view of the overall architecture of the *Quiz Main System*. It shows the different subsystems involved in quiz coordination and visual presentation, and highlights the position of the LED Management Subsystem within the entire infrastructure. This architecture is composed of specialised modules that coordinate to provide real-time interaction and visual feedback during quiz sessions.

The **LED Management System**, developed in this project, is responsible for video playback, LED control, and media preview functionalities projected onto the lecterns. It operates in synchrony with other components such as the **Moving Heads Control System**, which manages scenic lighting animations via DMX, and the **Question Viewer**, which displays questions and auxiliary content on screen. The **Quiz Maker** and **Simple-Quiz Server** handle question management and execution logic. The last one allows the audience to participate by voting from their mobile devices: when a question is launched, it opens a survey, collects answers in real time, and generates a public ranking based on their responses. The

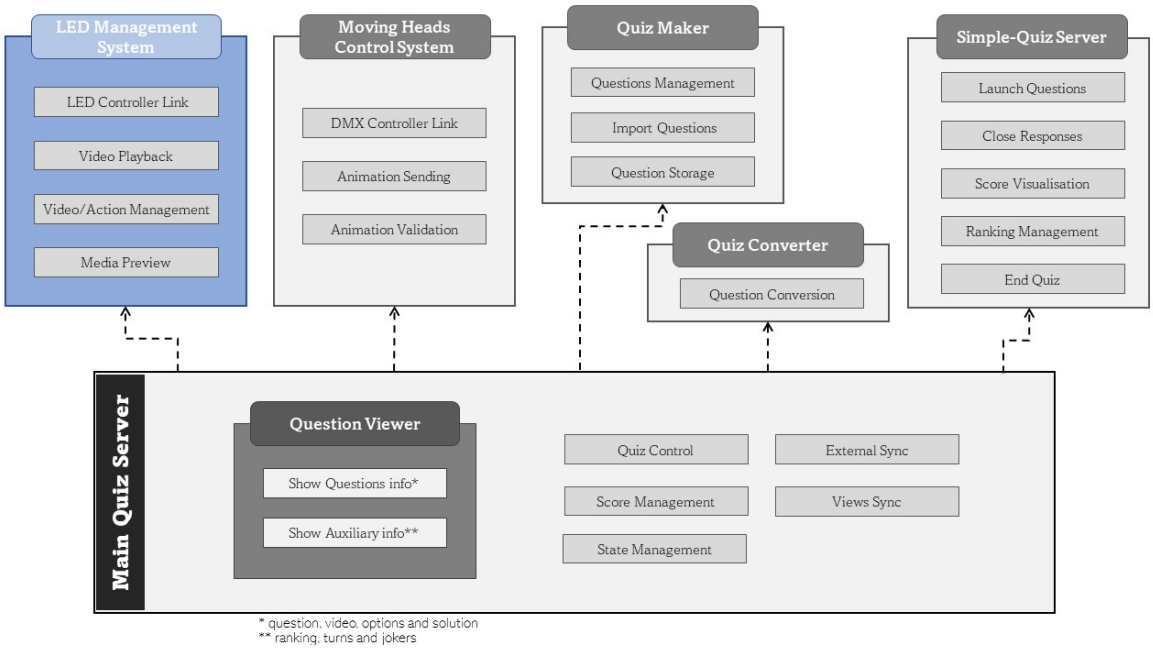


Figure 5.1: High-level component diagram of the Quiz Main System. The LED Management System is one of the coordinated subsystems alongside question display, score management (Simple-Quiz Server), and moving-head lighting control.

Quiz Converter acts as an intermediary layer to unify question formats. All of them are coordinated by the central **Main Quiz Server**, which governs the quiz state, scoring, view synchronisation, and external interactions.

In this subsection, the focus will be on the internal architecture of the LED management system and its division into two core components: front-end and back-end.

Front-End. The front-end enables users to manage available videos and its associated actions. It is possible to create a preview that is proportional to the actual dimensions of the LED lecterns, making it easy to check the result before running the projection. Additionally, the provided interface enables users to send playback commands directly to the system without requiring the main quiz server to be active, thereby providing the lighting system with functional autonomy.

Back-End. The back-end provides the logical core of the lighting system. Its main functionality is to process the video content, map the resulting RGB values to the correct LED layout, and dispatch synchronised DMX signals to the hardware controller. It is responsible for receiving commands from the user interface or from the main quiz server and executing them by orchestrating all the internal components.

The system has three logical subsystems to clarify its internal structure. All functionalities are part of a unified back-end service. This logical separation makes it easy to understand responsibilities and design decisions related to video processing, buffer management and communication with the controller. These subsystems are explained below.

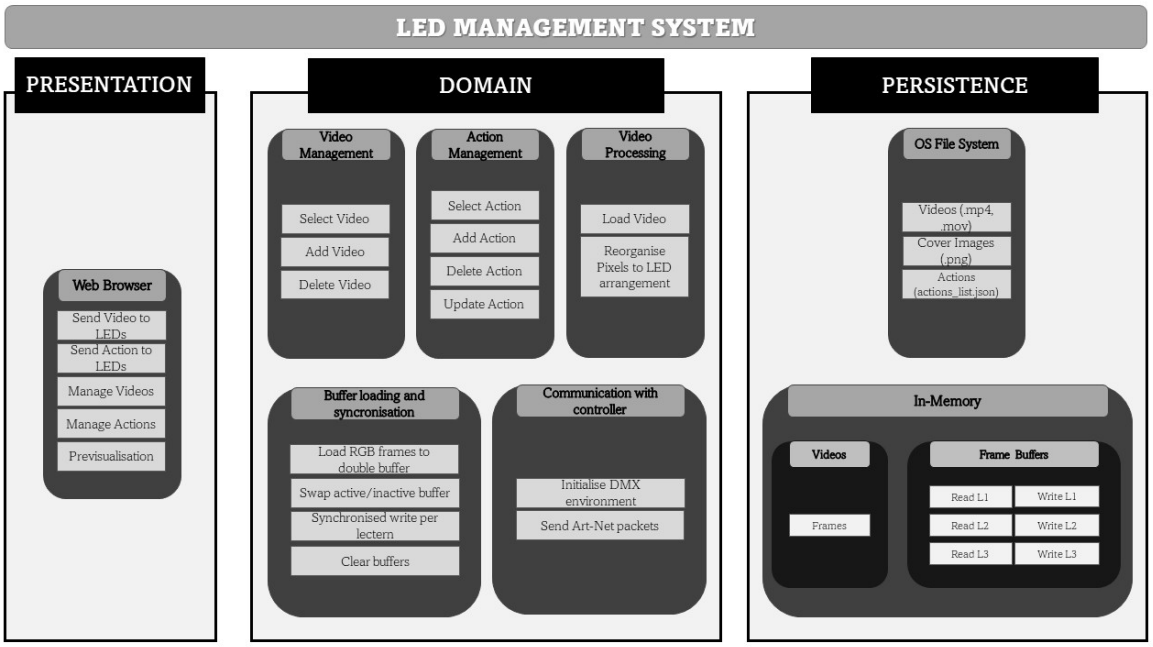


Figure 5.2: Layered architecture of the LED Management System. The components are grouped into presentation, domain, and persistence layers according to their functional responsibilities.

- **Video processing and actions**: This subsystem is responsible for the extraction, rearrangement, and storage of the RGB frames required for each lectern. Moreover, it establishes the link between each video and its associated action, so that the user can have predefined behaviours.
- **Buffer loading and synchronisation**: This subsystem handles the orderly transfer of RGB data to the corresponding buffer. Asynchronous techniques are employed to maintain temporal synchronisation between the video frames and the visual output displayed on the LEDs.
- **Communication with the Art-Net controller**: This subsystem is in charge of ensuring a stable and responsive lighting output. It must respect the strict timing constraints imposed by the controller and minimise packet loss. It uses the `pyartnet` library to transmit Art-Net packets to the controller.

An overview of the layered structure of the LED management system is shown in Figure 5.2.

5.2 Order coordination and business logic (back-end)

The back-end is the logical core of the system. Its main function is to orchestrate the flow of information between the user interface and the subsystems responsible for video processing and hardware control. It has been developed using the Flask framework in Python. Since the main quiz server is also developed in Python, using Flask ensured consistency

and compatibility between the two systems. Furthermore, Flask provides a lightweight and flexible framework for developing web applications.

Internal structure and architectural approach

The back-end has been designed using a **layered architecture** inspired by Clean Architecture principles [Mar17]. This promotes the separation of responsibilities and decoupling between components. This structure improves maintainability and facilitates unit testing, as well as allowing the system to be scaled progressively [Pil17]. The code is organised into four distinct conceptual layers:

- **Interface Adapters Layer**: It includes the HTTP endpoints handlers that receive incoming requests from the front-end. These modules perform initial validation and convert data into formats suitable for the application layer. All HTTP routes are defined using `Blueprints`.
- **Application Layer**: It contains the coordination logic through `VideoController` and `DMXController`. These classes act as intermediaries between the API endpoints and the core services of the domain layer. They are responsible for directing the execution flow by invoking the relevant methods according to the type of request received, without carrying out any validation or processing logic themselves. This design maintains a clear separation of concerns, ensuring that orchestration tasks are decoupled from domain models and business logic, both of which are handled by lower layers.
- **Domain Layer**: It defines the fundamental system entities (`Video`, `Frame`, `WideFrame` and `FrameBuffer`) and the associated business logic services, such as video processing, buffer synchronisation and sending data to hardware. These functionalities are accessed through structured service classes (`VideoService` and `DMXService`) that follow the principles of the facade pattern [Mar17], offering a simplified interface to higher layers while encapsulating complex internal operations.
- **Persistence Layer**[1]: It manages access to external resources, such as the file system, videos, cover images and the `actions_list.json` configuration file, using components like `VideoRepository` and `VideoFileLoader`.

Figure 5.9 shows a simplified class diagram of the back-end.

This final structure was not present from the beginning. Initially, the development focused on experimental validation: turning on a single LED, displaying an image, and later projecting a video on the LED lecterns. All of these tests were implemented within a single module, as the priority was to verify basic functionality rather than applying architectural principles. As development progressed, the system underwent several iterations, driven by

[1] Also referred to as the *Infrastructure* layer in Clean Architecture. In this system, it includes access to the file system rather than a database.

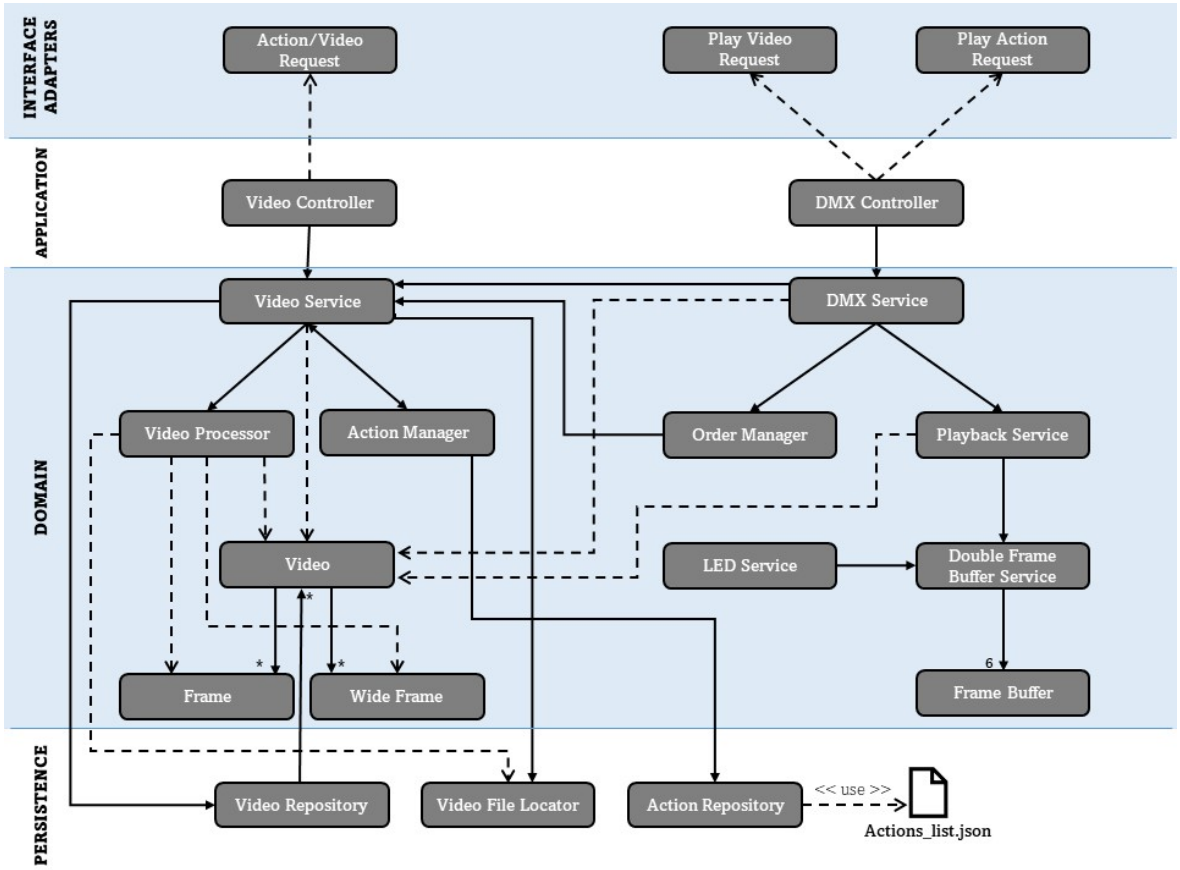


Figure 5.3: Logical structure of the back-end organised in four conceptual layers: presentation, application, domain, and persistence. Each class is grouped according to its role and dependency level within the system.

the progressive refinement of requirements and the emergence of unexpected constraints. These changes required a gradual restructuring of the back-end. This ultimately led to a modular and maintainable design.

In the *initial design*, all responsibilities (video processing, data buffering and DMX transmission) were concentrated in a single module. At that stage, it was essential to carry out preliminary tests and resolve any uncertainties regarding the control of the underlying hardware. While this enabled a quick functional prototype to be created, the structure exhibited high coupling, low maintainability and poor code clarity. These limits became clear as the system grew more complex.

The *first reorganisation* introduced a basic modular structure by splitting the functionality into two classes: The `VideoService` processes and prepares video data for transmission, and the `LEDService` sends the data to the hardware. While this change achieved an initial level of functional decoupling, it also revealed a performance bottleneck. Each video frame was processed and transmitted sequentially, which introduced latency and degraded the system's responsiveness under load.

The *current architecture* addresses these limitations through a clear division of responsibilities into three logical stages: (1) video processing, which is performed once at server startup; (2) intermediate memory buffering using a double-buffering mechanism, and (3) continuous transmission of RGB data to the Art-Net controller. A dedicated set of classes is responsible for implementing each stage, following the principle of single responsibility [Mar17]. This approach improves modularity, enhances maintainability, and facilitates testing and future extensions of the system.

Receipt and Management of Orders

In the initial version of the system, separate routes were defined for each order type, e.g.`/OneOn`, `/Wide`, `/Different` resulting in a rigid and unscalable structure. In a later stage, these routes were unified. They were put under a single URL: /api/play. This accepts POST requests that have a flexible structure. This allows you to specify which videos and on which lectern should be played.

To enable integration with the main quiz server, an alternative route was enabled via POST: `/playJSON`, which accepts orders in the standard format `(action, device)`. The system uses the action identifier to access the `action_list.json` configuration file to determine which video should be played and how. This same path is also used with the GET method to enable communication between the system and the OBS software, which is responsible for event relay.

5.3 Communication with hardware subsystem (Art-Net)

The main objective of this subsystem is to establish and maintain communication between the lighting control system and the hardware that activates the physical LEDs. The transmission of DMX data over an IP network is made possible by the Art-Net protocol, which is the basis of this communication. The conversion of Art-network packets into real SPI signals distributed by universes is carried out by the physical controller `Deskontroller 16` [Des17].

The module must ensure reliable, lossless data transmission at a constant refresh rate and correct alignment of the DMX channels with the physical arrangement of the LEDs on each lectern.

Implementation details

The communication process with the Art-Net driver is given by the `pyartnet` library, a lightweight API written in Python. This library enables interaction with Art-Net nodes and the transmission of DMX data to designated universes and channels. [2] Upon initiation of the system, a connection is established with the node corresponding to the IP of the controller,

[2] `https://pyartnet.readthedocs.io/en/latest/index.html`

designated as `Deskontroller 16`. The required universes are then defined in accordance with the physical distribution of the LEDs across the three lecterns.

Each universe is represented by a single object, which contains a set of DMX channels. The maximum number of channels that can be accommodated is 512, with each channel arranged in a block of three channels per RGB pixel. The data of the read buffer is continuously read by this subsystem and sent sequentially to the configured channels.

In the system's preliminary design, `OLA` (Open Lighting Architecture) was considered a solution with wide application in professional lighting control environments. [3] The selection of `pyartnet` was ultimately determined by the following factors: its superior simplicity of use, its native integration within Python, and its utilisation in the remaining back-end components. While `OLA` offers greater flexibility and support for multiple protocols and simultaneous nodes, `pyartnet` was more suitable for a closed and controlled system such as the one proposed in this project, where scalability requirements are limited and reliability can be ensured by direct control of the data flow.

Refresh rate and stability

In order to maintain visual stability and avoid both flickering and latency, it is vital that data transmission to the DMX universes is carried out continuously, even when no video is being displayed. This strategy enables the LEDs to be maintained in a constant state, thus preventing intermittent power outages that could potentially lead to the generation of visual artefacts during the playback process.

The transmission of data occurs in regular cycles, using a dedicated asynchronous loop that iterates over the reading buffer of each lectern and transmits the RGB values to the corresponding channels. Initially, the configuration of the system was set to deliver 50 *frames per second* (FPS), representing the maximum capability of the `Deskontroller 16` driver. However, this frequency was later reduced to 30 FPS as an efficiency measure. Since the videos played rarely exceed 24 FPS, maintaining a rate lower than maximum allows reducing the system load without compromising visual fluidity.

During the preliminary testing phase, it was observed that the LEDs exhibited erratic behaviour, characterised by intermittent flickering [4], even when the data submitted were technically correct. Using analytical tools, including `Wireshark` [5], allowed the validation of the hypothesis that the controller would discard DMX packets if multiple consecutive transmissions had identical content. This behaviour was especially evident in videos featuring slow transitions or static scenes, where the RGB values remained constant over several frames.

[3] `https://www.openlighting.org/ola/`

[4] This phenomenon manifested as intermittent illumination with sporadic colour variations, partial deactivation, and unexpected activation of the LEDs. It is evident that these effects did not correspond to the content of the video, and were produced in a non-deterministic way.

[5] `https://www.wireshark.org/`

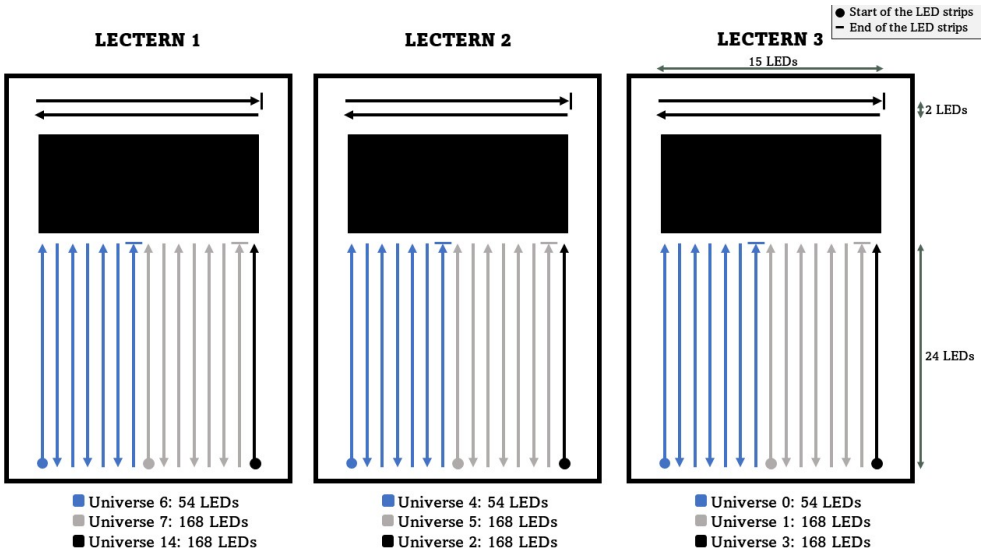


Figure 5.4: The allocation of LEDs by DMX universes and their physical distribution

To mitigate this effect, a slight controlled variation was introduced in the sent data: at each iteration, the value of the red channel of the first LED in the universe 0 is modified by alternating between adding or subtracting one unit. This minimal variation does not affect visual perception, but ensures that each package is interpreted as distinct, thus avoiding its disposal by the controller. The implementation of this solution, based on a Boolean variable, has proven to be effective and stable in all scenarios tested.

Channel and universe allocation

The allocation of DMX channels is dependent on the physical configuration of each lectern and the distribution of its LED strips. Each area of the lectern (upper, left and middle) corresponds to a specific sequence of channels within a designated universe.

This correspondence is established during the initialisation phase and remains unchanged throughout the system execution. The logical mapping of pixels (internal memory order) corresponds exactly to the physical mapping of LEDs (actual channels in the controller), facilitating efficient transmission and accurate projection with no need of intermediate calculations. Figure 5.4 shows how the universes and channels are distributed in each area of the lectern.

Robustness and error tolerance

The communication system does not implement an active mechanism for verifying the status of the network or the packets sent. Instead, it continues to send data periodically across the defined universes, under the assumption that the receiving node will process them if available. Despite the absence of specific retries in case of transmission error, the design exhibits robust resilience against temporary disconnections. The absence of confirmations does not result in system crashes if the controller is disconnected or not responding.

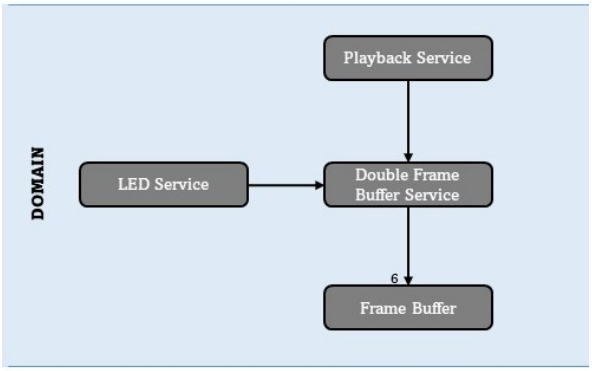


Figure 5.5: Simplified class diagram of the hardware communication subsystem. It includes the classes responsible for buffer loading (`PlaybackService`), double-buffer management (`DoubleFrameBufferService`), and RGB data transmission (`LEDService`).

Furthermore, it is possible to manually force the complete shutdown of the LEDs by sending a frame with all RGB values set to zero. This functionality is advantageous in critical failure situations and enables the controlled termination of a projection session.

Internal structure

Figure 5.5 illustrates the interaction between the playback logic, the double-buffer mechanism, and the LED emission service involved in the communication with the Art-Net controller.

5.4 Buffer loading and synchronisation subsystem

This subsystem coordinates the playback of pre-processed videos on physical displays and ensures that colour data (RGB) is transmitted in the correct order and with accurate timing. It functions as an intermediary between the storage of the pre-prepared `Frame` objects and the module responsible for transmitting the data to the Art-Net controller, using dedicated buffers. It serves two purposes: firstly, it maintains the temporal correspondence between the video content and its display on the LEDs; secondly, it ensures that data loading and transmission to the hardware are carried out efficiently and without interference from concurrent tasks.

Buffer management per lectern

Each lectern has a pair of fixed buffers (read and write) that are managed by the `DoubleFrameBufferService` class. When a playback task loads a new frame, it does so in the write buffer. Once this process is complete, the roles of the buffers are *swapped*, so that the controller always accesses the read buffer without risk of collisions or inconsistencies.

Unlike the video processing phase, this subsystem does not use dynamic structures such as `deque` (double-ended queue) at runtime. Instead, a fixed list of 390 RGB values is contained

Stage	Main Characteristics	Drawbacks / Advantages
Initial Design	Real-time processing and direct sending from `VideoService` to `LEDService`.	*Drawbacks*: High CPU usage, latency, desynchronisation between lecterns.
Intermediate design with JSON	Frames written to a shared JSON file and read continuously by the sending module.	*Advantage*: Functional decoupling. *Drawbacks*: Race conditions, disk overhead, and performance degradation due to the use of synchronisation mechanisms such as `locks`.
Preload in memory	Videos fully processed and stored in memory upon server boot.	*Advantage*: Improved stability and speed. *Drawback*: Still relies on file reading to access RGB frames.
Double Buffering	Concurrent access via active/passive buffers using `DoubleFrameBufferService`.	*Advantage*: High consistency, no locks, optimal synchronisation, real-time performance.

Table 5.1: Summary of the architectural evolution towards a buffer-based real-time playback model.

in each `FrameBuffer`, corresponding to the total number of LEDs per lectern. This decision allows direct access by index. It also provides greater temporal stability. This minimises the computational overhead and possible errors for concurrent access.

This behaviour follows the classic *double-buffering* pattern [HAM02], in which two buffers are used per lectern: one for writing and one for reading. Controlled exchanges occur after each update. This technique ensures that the controller always accesses a fully consistent data state, thus avoiding race conditions. A race condition arises when two concurrent operations access a shared resource simultaneously and at least one of them performs a write operation, leading to unpredictable results depending on the timing of execution [HLea15]. In concurrent systems, such issues can cause severe desynchronisation, data corruption, or latency peaks. Although mechanisms such as locks or semaphores are often used to handle race conditions, they introduce significant complexity in the management of concurrency, including the risk of deadlocks or contention [Tan09]. In this system, the use of double buffering removes the need for such synchronisation primitives, ensuring deterministic and conflict-free access to the data.

To better understand the reasons behind the current design, Table 5.1 summarises the different architectural strategies explored for transferring video data to the Art-Net controller, from the initial design to the current implementation of the *Double Buffer* pattern.

Double Buffer pattern

The final version of the system implements the *Double Buffer* pattern, a widely adopted technique in multimedia systems and real-time rendering [HAM02]. The pattern relies on

maintaining two buffers: one dedicated to writing (active) and another to reading (passive). Both operations never act on the same structure at once. Once the writing of a frame is complete, a *swap* takes place, allowing the reading process to access a stable and updated data set.

This approach has been shown to be effective in multimedia systems where synchronisation of data flows is critical. Studies have showed its effectiveness in preserving the quality of service in continuous video transmissions on low-speed networks [OOea13].

In our system, each lectern is assigned two `FrameBuffer` objects, managed by the `DoubleFrameBufferService` class. The playback task loads frames into the write buffer, while the emission task retrieves RGB data from the read buffer and sends it to the appropriate DMX universes. This design guarantees consistency, determinism, and responsiveness even if there are multiple lecterns receiving orders concurrently. A simplified diagram illustrating how the pattern operates within the system is shown in Figure 5.6.

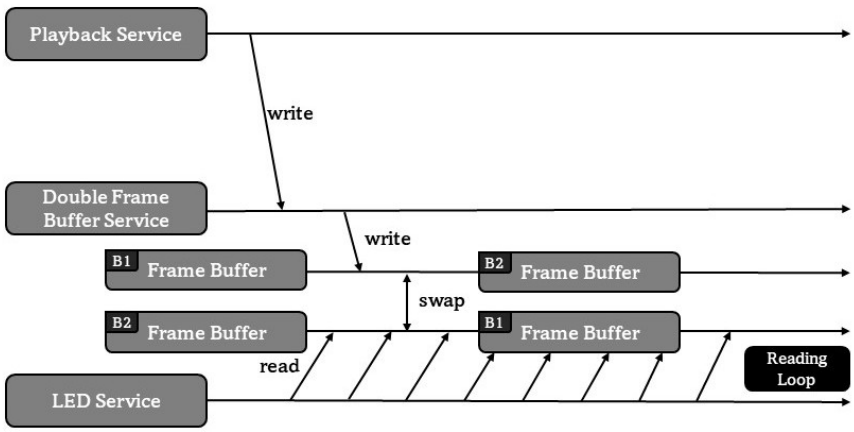


Figure 5.6: Diagram illustrating the double-buffering mechanism. The `PlaybackService` writes frames to the active buffer, managed by `DoubleFrameBufferService`. Upon completion, a swap operation takes place, making the written buffer available for reading. The `LEDService` continuously reads from the passive buffer to transmit RGB data to the Art-Net controller.

Projection modes and playback logic

The system has four projection modes (see Figure 5.7), each of them with a different running logic:

- **`Individual`**: One video is projected onto a single lectern. A single asynchronous task associated with the corresponding buffer is launched.
- **`Different`**: Three different videos are played, one per lectern. The process is divided into three independent asynchronous tasks, with each one managing its own `FrameBuffer`. They do not need to coordinate with each other.

- `AllLecterns`: A single video is shown simultaneously on the three lecterns. An asynchronous task iterates over the video frames and updates the three buffers with the same content in each iteration. This approach ensures synchronisation without the need for multiple cores.
- `Wide`: Each panoramic video is divided into three zones (left, middle and right) during the pre-processing phase. The playback task's function is simply to extract each `WideFrame` and update the three buffers with their respective sections. Segmentation at runtime is not performed as part of the task's function.

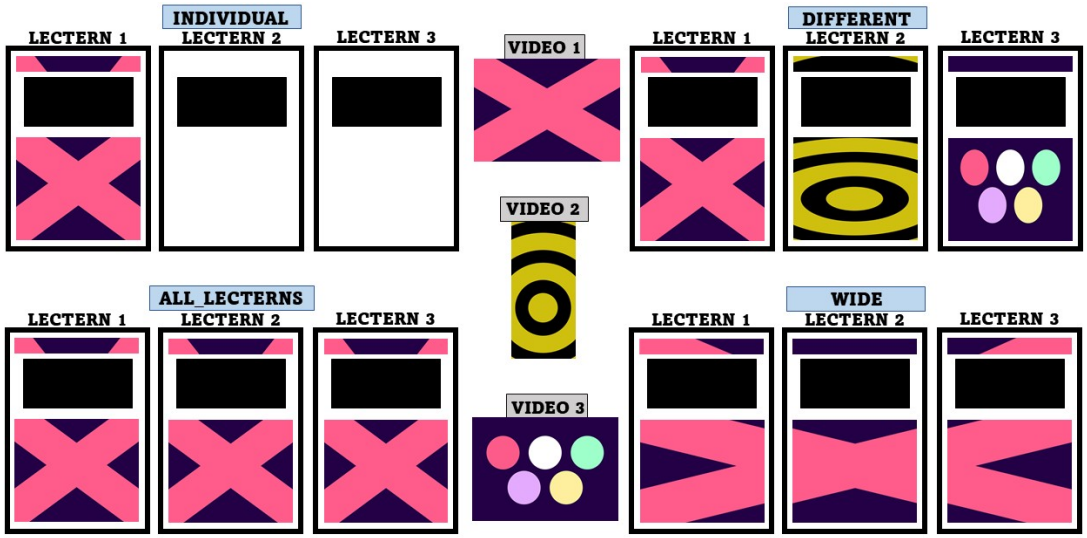


Figure 5.7: Visual representation of the four playback modes supported by the system. From top-left to bottom-right: `Individual`, `Different`, `AllLecterns`, and `Wide`. Each configuration determines how the video content is distributed and synchronised across the three lecterns.

Timing and task control

In this system, visual synchronisation depends directly on accurate frame timing. Any deviation in the playback rhythm can cause noticeable visual mismatches, especially when projecting animations across multiple devices. To address this, the system must manage frame delivery intervals with high precision.

In this implementation, playback speed is controlled using high-precision timers based on `time.perf_counter()`, which offer greater resolution and stability than traditional methods such as `time.time()`. Each playback task operates its own timing loop to calculate the exact moment when each frame should be delivered. If a frame is detected as delayed beyond tolerance, it is skipped to maintain synchronisation and avoid the accumulation of mismatches.

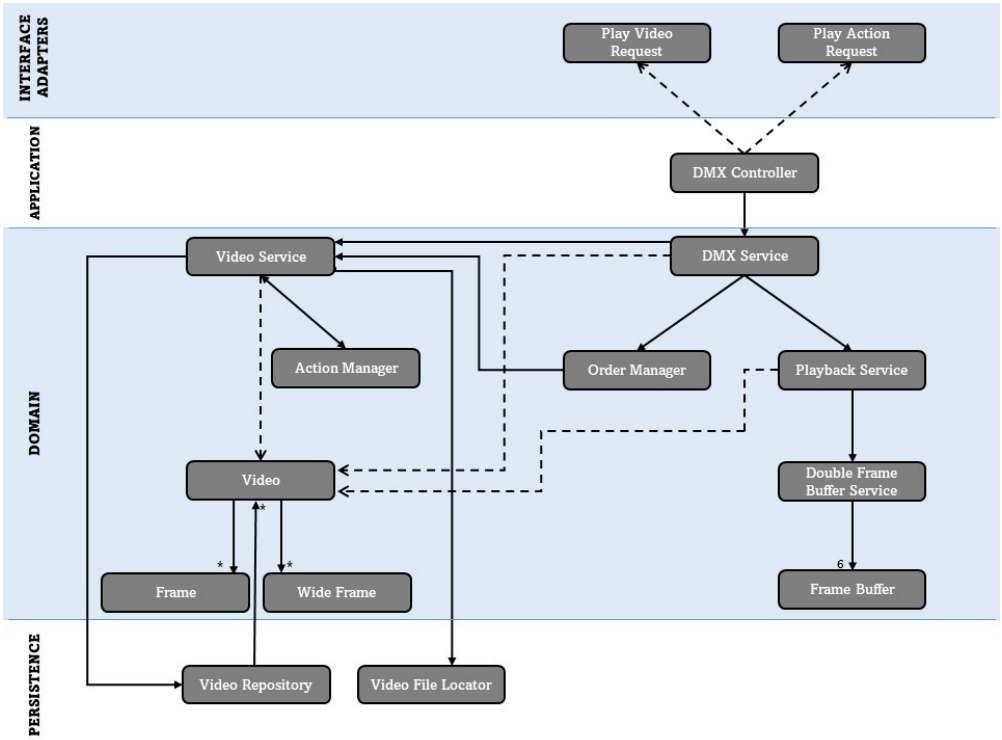


Figure 5.8: The buffer loading and synchronisation subsystem simplified. It shows which classes involved from the reception of an order (video or action) until the processed RGB frame is written to the corresponding `FrameBuffer`.

Furthermore, when a new playback command is received, any previous active tasks on the corresponding lectern are automatically cancelled. This ensures that each lectern only reflects the most recent order, avoiding the accumulation of pending orders or overlay projections.

Asynchronous coroutines[6] manage all this mechanism, allowing concurrency maintenance without resorting to multiple threads. This approach eliminates the need for complex synchronisation primitives, such as `locks` or `semaphores`.

Internal structure

The internal workflow of the buffer loading and synchronisation subsystem is illustrated in Figure 5.8. This class diagram summarises the interaction between key components involved in handling a playback or action request from its reception via the Art-Net controller to the final delivery of RGB data into the appropriate `FrameBuffer`.

[6]Asynchronous coroutines are special functions in Python defined with `async def`, which allow non-blocking execution by yielding control during long I/O operations or delays. They are executed within an event loop and facilitate concurrency without the need for traditional multithreading. See: `https://docs.python.org/3/library/asyncio-task.html`

5.5 Video processing and management subsystem

The video processing subsystem is designed to take the visual content of a video and turn it into a structured RGB data stream. This is so that the information can be accurately and reliably represented when it's projected onto the LED strips. The transformation is adapted to the physical layout of the lecterns and takes place before they are transmitted to the hardware.

In addition to this pre-processing, the module manages the system's audiovisual resources. In particular, it is responsible for:

- Receiving, storing and organising videos uploaded from the web interface, as well as their cover images (covers).
- Maintaining the system resource directory structure, specifically the folders `resources/videos` and `resources/covers`, where the necessary files are located for later access and use.
- Managing the `action_list.json` configuration file, which associates action identifiers with specific videos and specifies whether they should be played in a loop and in what mode in case they are sent to the three lecterns simultaneously (panoramic or multiple).
- Providing mechanisms for adding, editing, and deleting videos and actions, ensuring consistency between the stored data and visible elements in the interface.

This subsystem acts as both a conversion engine for visual content and a central point for managing and storing the multimedia resources used by the system. The decoupling of these tasks from the synchronisation, timing and sending logic, which are handled by other independent subsystems, is made possible by its modular design.

Image extraction and resizing

The first step in processing a video file is to sequentially extract its frames. For this purpose, the `OpenCV` (`cv2`) library has been used, as it is widely used in image and video processing due to its efficiency and versatility. [7] Earlier versions of the system also evaluated the use of `Pillow`, although it proved more suitable for processing static images (*covers*) than video sequences.

Each extracted frame is automatically resized to a fixed resolution depending on the selected projection mode. For projection onto a single lectern, the target resolution is **15 pixels wide by 26 pixels high**, corresponding to the number of LEDs actually placed on each lectern. For panoramic videos (`Wide` mode), the frame is adjusted to a resolution of **45 pixels wide by 26 pixels high**, which is then divided into three vertical zones, with one zone per lectern measuring 15 pixels in width.

[7] https://opencv.org/

Reducing the resolution establishes a direct correspondence between video pixels and physical LEDs, eliminating the need for interpolation or complex mapping. The system's computational load is also reduced by this, since the amount of information that must be processed and rearranged in each frame is decreased. This is achieved without significantly compromising the visual quality perceived in the LED matrix.

Pixel spatial rearrangement

Once the frames have been extracted and resized, the resulting data structure must be transformed to reflect the physical arrangement of the LEDs on the lecterns. By default, frames are stored as pixel arrays in a Cartesian format, from left to right and from top to bottom. However, this arrangement does not coincide with the order of the physical LED connections, so the data must be completely reorganised.

Figure 5.4 illustrates the physical arrangement of the LEDs on each lectern. It distinguishes between the horizontal (top) and vertical (bottom) areas and shows the connection direction and the Art-Net universe assigned to each strip. This setting forms the basis of the pixel reordering algorithm, which is described in the next section.

Pixel rearrange algorithm

The implemented algorithm rearranges the pixels in each frame according to the physical layout of the LEDs on the lecterns. First, the frame is resized to an array of 15×26 pixels. Three lists are then generated from this: one for the upper horizontal strip (`upLEDs`), and two for the vertical strips (`leftLEDs` and `middleLEDs`).

The key steps are:

1. Extract the first two rows for the top area and apply alternating inversion (zigzag).
2. Rotate the image 90º to process the columns vertically.
3. Go through the columns by grouping pixels in serpentine mode (from bottom to top and vice versa) to build lists corresponding to the sides.

The result is not a single structure, but three `deque` lists that reflect the actual order in which the physical LEDs should be lit.

Below is a simplified version of the pseudocode algorithm:

```
function reorder_pixels(frame):
    resized_frame = resize(frame, width=constants.WIDTH, height
       =constants.HEIGHT)

    // Top side of the lectern
    top_leds = deque()
    top_leds.extend(pixels from first row)
```

```
7       top_leds.extendleft(pixels from second row) // Inverse
            order

9       // Rotation for working with vertical strips
10      rotated_frame = rotate_90_clockwise(resized_frame)
11      data = flatten(rotated_frame)

13      // Vertical zigzag process
14      left_leds = deque()
15      middle_leds = deque()
16      for strip in [left_leds, middle_leds]:
17          direction = up
18          while len(strip) < constants.LONG_STRIP_LENGTH:
19              if direction == up:
20                  strip.extend(next_column_top_to_bottom(data))
21              else:
22                  strip.extend(next_column_bottom_to_top(data))
23              data = remove_used_column(data)
24              direction = toggle(direction)

26      // Completion of the top if there are remaining LEDs
27      while len(top_leds) < constants.TOP_STRIP_LENGTH:
28          data = flip(data) // inverts the matrix on the x-axis
29          top_leds.extendleft(next_row(data))
30          data = remove_used_row(data)

32      return top_leds, left_leds, middle_leds
```

Code listing 5.1: Algorithm for Pixel Rearrangement

Once the algorithm has been applied to all the frames in a video, the `deque` of `Frames` is added as an attribute to the `Video` object. In panoramic mode, this process is repeated three times. The frame is divided into three vertical zones measuring 15×26, generating a total of nine rearranged `deques`. This data is stored in a `WideFrame` object, which is then added to a `deque` and stored in the same `Video` object. This avoids the need for additional segmentation or transformation during execution.

Use of efficient structures

In the early versions of the system, rearranged data for each frame was stored in standard Python lists (`list`), due to their simplicity of use and intuitive representation of RGB sequences. However, as the system evolved and more demanding timing mechanisms were

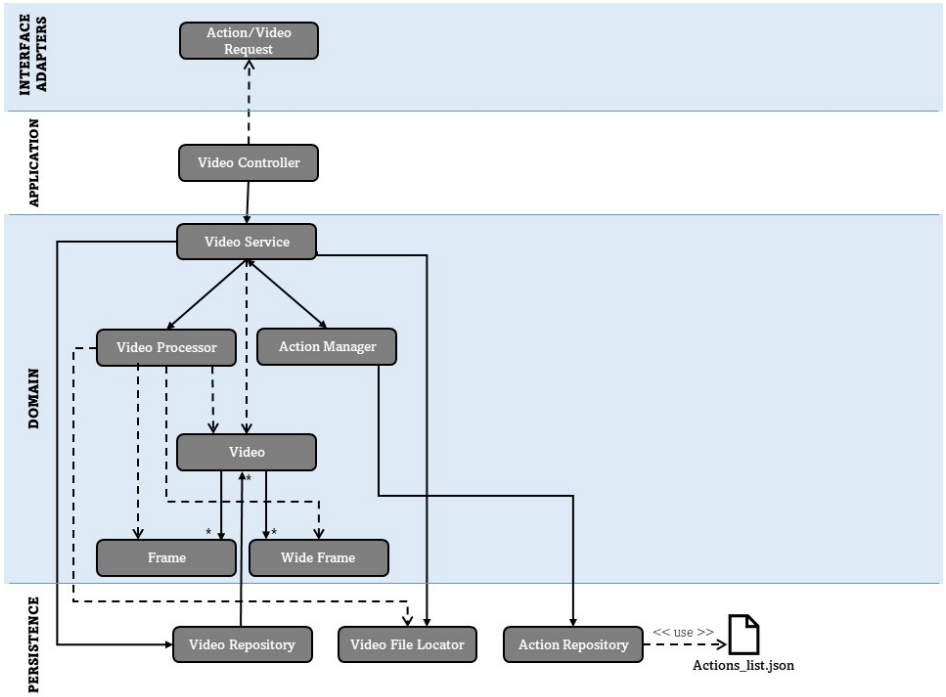


Figure 5.9: Class diagram of the Video processing and management subsystem. It shows which classes are used by this subsystem

integrated, the need to improve performance arose. To achieve this, I adopted the `deque` data structure provided by the standard Python `collections` library. This structure allows insertion and removal operations to be carried out at both the start and end of the sequence. This is particularly useful in the context of LED projection, as the data for each frame is consumed in order. No intermediate access is required.

This decision was in line with the principle of sequential access that had already been adopted when pixels were rearranged into one-dimensional lists. Thus, each `Frame` object contains three `deques` representing the RGB data to be sent to a specific lectern. This data can be efficiently consumed by the synchronisation and sending system without the need for additional transformations.

The use of `deque`, therefore, not only optimises system performance, but simplifies memory management by avoiding costly operations associated with traditional lists, such as erasing multiple items or internal reordering.

Internal structure

Figure 5.9 provides a structural overview of the classes involved in the video processing and management subsystem. The diagram highlights how user requests are handled through the `VideoController`, and how responsibilities such as video decoding, action association, and storage are distributed across components.

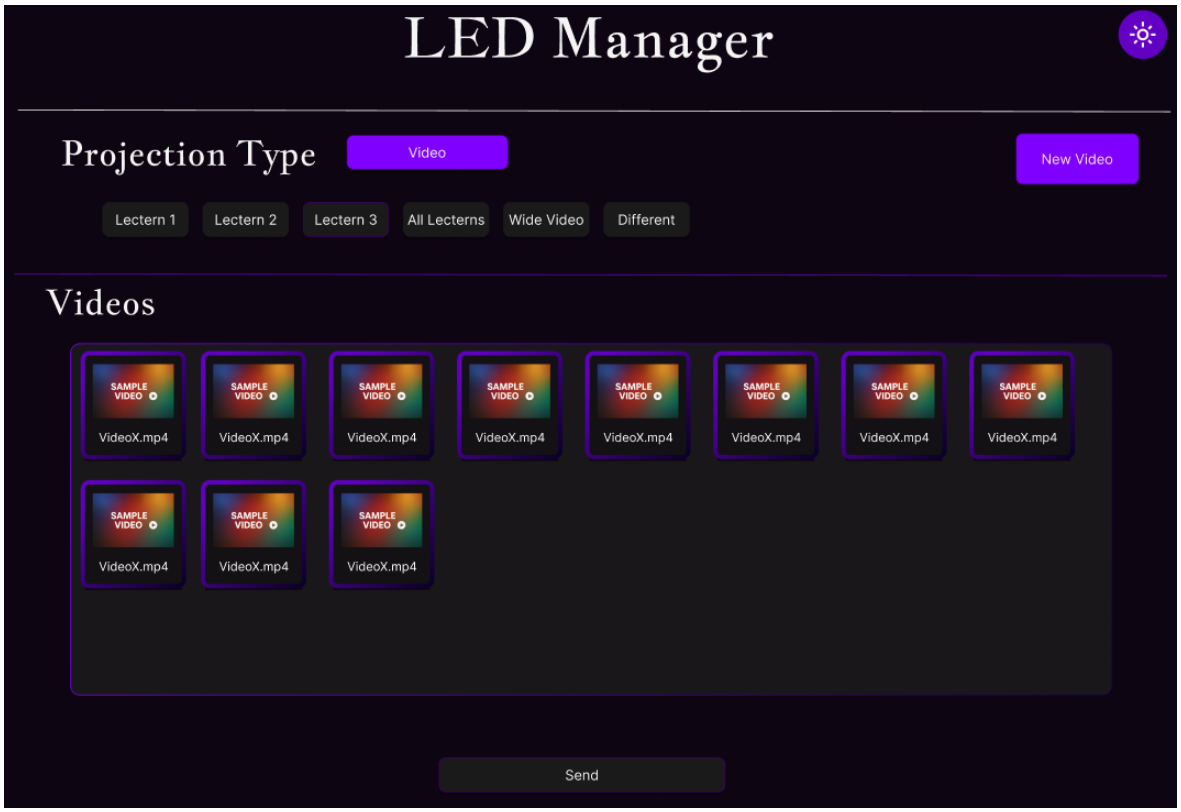


Figure 5.10: Early-stage prototype of the LED Manager interface, designed in Figma. It shows the planned layout for video selection, projection mode configuration, and playback control.

5.6 Web interface (Front-End)

The web interface is the main point of access for users to the system. It allows to manage the available videos, assign actions to each one, select the target lectern, preview the animations, and run the projection on the LEDs. Several interface concepts were evaluated using visual prototypes before implementing the final version. Figure 5.10 shows one of the early design drafts developed in Figma, which helped to define the initial distribution of components and the main interaction flows.

Split decision between Back-End and Front-End

In the early stages of development, server-side rendering using `Jinja2` templates was adopted as a practical choice. At that time, the system was quite simple, and adding HTML rendering directly to the Flask back-end meant that it was possible to create a prototype more quickly and test its functionality immediately. However, as the interface became more complex, it became clear that this approach had important limitations. The close integration of logic and presentation made it difficult to create a modular design and restricted the ability to evolve the user interface independently.

Therefore, I chose to completely decouple the front-end, developing it with **HTML**, **CSS** and **TypeScript**, which was later compiled into JavaScript. To manage this environment,

Node.js was used, allowing the front-end to be compiled, served, and debugged autonomously. The same root project is used for both environments, but they are organised in separate folders. This makes for a more maintainable and scalable architecture.

This decision has made it possible to add new visual features, improve the design, and develop the interface in modules without affecting the Flask server's logic.

Main interface functionalities

The interface is divided into two main sections: one for video management and the other for configuring actions. The aesthetic is coherent in both cases, and reusable elements such as buttons and modes allow the user to interact intuitively with the system. The implemented functionalities that stand out are:

- Upload new videos.
- Association of videos to custom actions.
- Edit and delete existing videos and actions.
- Preview animations with a size proportional to the actual dimensions of the lecterns.
- Select the projection mode: individual, simultaneous, or panoramic.

Each projection order can be sent directly from the interface. This means that the main quiz server is not needed. This gives the system an autonomous operational capacity, which is for independent tests or demonstrations.

Visual preview

A key feature of the front end is the preview viewer, which is designed to enable users to check how an animation will be projected before sending it to the LEDs. At first, the idea was to use an HTML `<canvas>` element, as this gives a lot more control over how frames are rendered. This would have enabled direct manipulation of the video pixels, resizing, deformation, and simulation of the physical arrangement of the LEDs via graphical operations programmed in JavaScript.

Nevertheless, this solution was more complex to implement. It would have been necessary to capture each frame of the video, convert it into an image or a pixel buffer, and then draw it manually on the canvas. All of this had to be done while maintaining a consistent refresh rate. Given the functional purpose of the preview, which was simply to show the video at a reduced scale, this computational and development load was considered unnecessary. Therefore, a simpler solution was chosen that was equally valid for the purpose of the interface: the HTML `<video>` element. This component makes integration easy, manages playback automatically and offers acceptable performance to represent a preview of content. Playback can be performed for either a single video or three simultaneously, one per lectern.

Front-End structural evolution

During the early stages of development, all front-end logic was centralised in a single TypeScript file, which was manageable for a small project. However, as new features like action and video management were added, the code got bigger and more complicated, which made it hard to maintain.

In view of this situation, it was decided to restructure the front-end by dividing it according to functionality. Each system module is now encapsulated in an independent file, each of which is responsible for a specific task (video handling, action management, interface control and sending requests, among others). This organisation improves readability and enables more ordered and modular work.

Furthermore, improvements were made with the user experience in mind. These included the addition of visual feedback to highlight selected items, minimal validations to avoid errors in usage (e.g. the mandatory selection of a video and a lectern before being sent), and a clear and accessible distribution of buttons and information.

5.7 Deployment architecture and integration with the quiz server

This section focuses on the technical integration between the LED management system and the main quiz server. It describes the deployment environment based on Docker containers and the validation mechanisms implemented to ensure correct behaviour during the event.

Dockerisation and network compatibility

In the course of the integration process with the primary quiz system, the system was containerised with a view to facilitating deployment in controlled and reproducible environments, as well as ensuring compatibility with existing network infrastructure.

At this stage, conflicts related to network ports occupied by other services hosted in different containers were detected. The solution to this issue was to modify the default port of the back-end and connect its container to the internal network defined by Docker. This modification allowed for the establishment of direct and stable communication between the different components of the system.

Validation and control mechanisms

In order to ensure the maintenance of functional consistency during the event, validation mechanisms were incorporated to ensure compliance with the rules of the competition system. The following aspects stand out:

- **Control of mode settings**: The system uses the `all_Lecterns` attribute defined in each action to determine how an action should be projected between the three lecterns. Despite the absence of any logical restriction that would prevent the simultaneous

reproduction of a single animation on multiple lecterns, this configuration functions as a consistency mechanism. In essence, it ensures that the commands transmitted adhere to the mode designed by the user.

- **Structural order validation**: The system avoids accumulation or delayed execution of multiple actions through the use of a list indexed by lectern, where each position maintains exclusively the active playback task at that instant. Upon receiving a new order, the existing order is overwritten, thereby ensuring that the most recent action is always projected, and unwanted execution queues are avoided.

These validations are based on a system of actions, the definition of which is provided in a JSON configuration file (action_list.json). Each action determines the key attributes of the expected behaviour in a declarative way. These include the video to be played, the looping of the video, and the playback logic to be applied to all three lecterns (`Wide`, `AllLecterns` or `Different`). This model achieves a flexible integration between the back-end and the main server of the quiz by abstracting the presentation logic into a configurable and decoupled layer from the control code.

Chapter 6

Results

In this chapter, the LED control system developed from the architecture discussed is presented. This system has been validated through three complementary approaches: (i) iterative and incremental testing during development, (ii) systematic automated testing to guarantee internal consistency, and (iii) deployment in a real-world contest scenario. These validations confirm that the system meets the functional objectives initially proposed, and that it behaves robustly under interactive and time-sensitive conditions. The source code is available at the following URL:

`https://github.com/carmencbog/leds-management-project`

6.1 Development Context

The system aims to enable the synchronised control of LED lighting using video content as the source of dynamic visual data. This section will briefly outline the functional scope of the system from the user's perspective, as illustrated in the use case diagram showed in Figure 6.1.

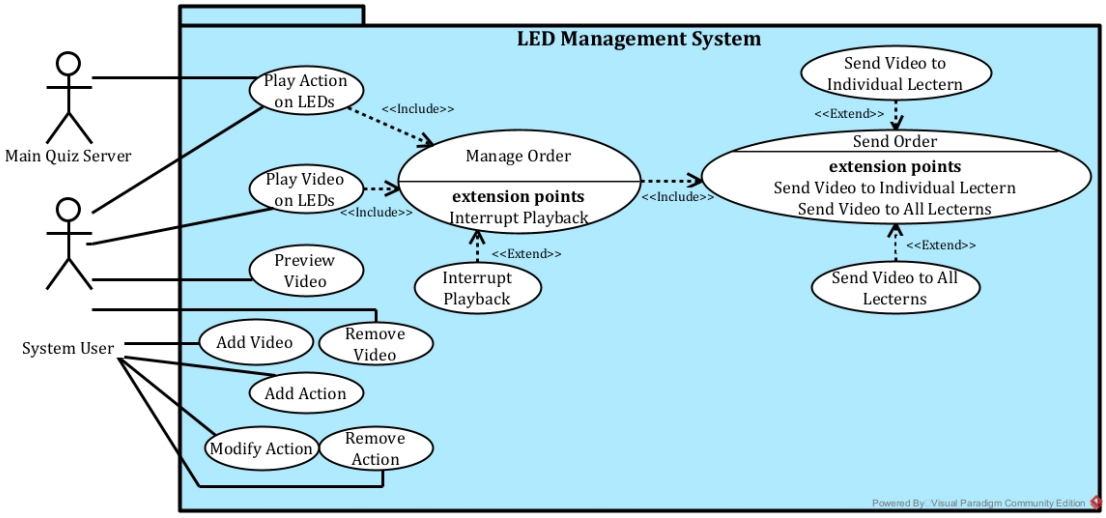


Figure 6.1: Use case diagram of the LED Management System. It shows the interactions between the System User and the Main Quiz Server with the LED module. Internal cases such as *Send Video to Lecterns* are handled by the system at a lower level of abstraction.

A detailed description of the iterative progress can be found in Appendix B, where each development week is documented in terms of activities, technical decisions, challenges encountered, and solutions adopted. This chronological overview helps trace the system's design evolution and reflects the practical decisions taken to ensure its stability, modularity, and compatibility. Figure 6.2 provides a visual summary of the main activities carried out in each iteration, grouped chronologically by week.

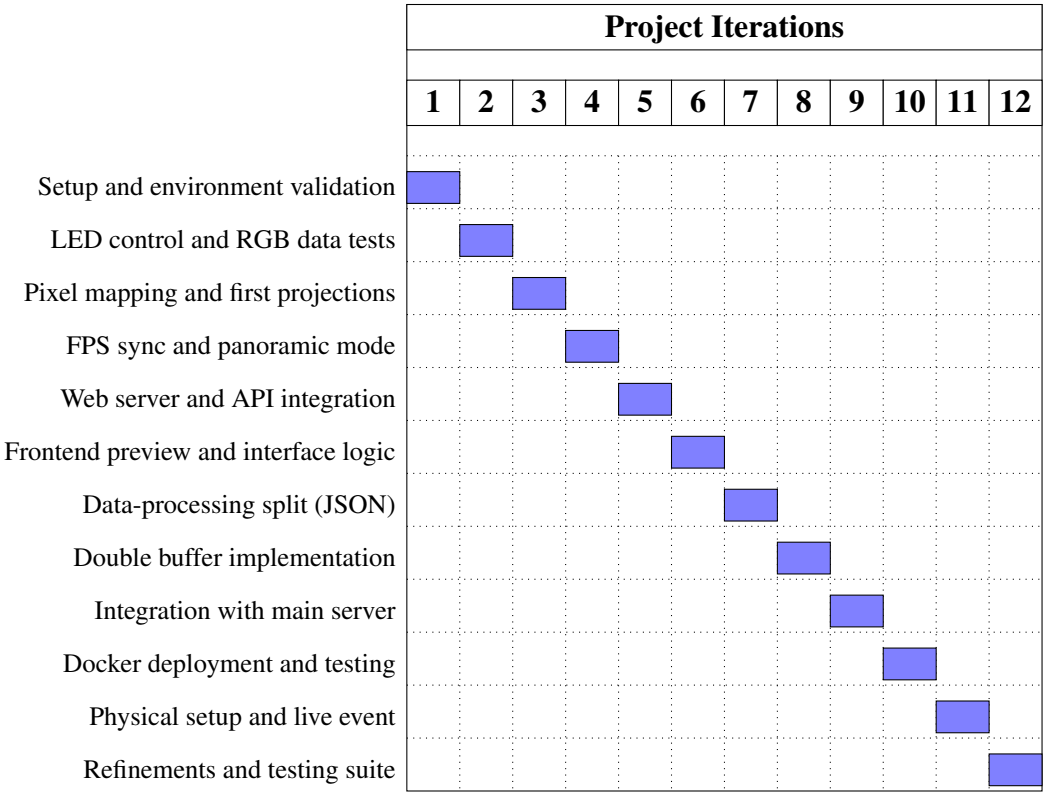


Figure 6.2: Chronological overview of the development iterations and main milestones.

6.2 Incremental validation, systematic testing

Throughout development, the system's functionalities were progressively tested in close alignment with the evolving requirements. These empirical tests not only ensured the correct execution of each feature, but also served as a foundation to verify the system's responsiveness and robustness under diverse operational conditions. Special attention was given to aspects such as synchronisation precision, performance stability, and correct task management during concurrent execution.

Playback synchronisation and timing accuracy

One of the system's most important features is the synchronised playback of videos on LED lecterns, which maintains fluidity and visual correspondence with the original frames. During the initial tests, significant misalignments were evident: discrepancies in the time

taken to read the frames, process them and send them to the DMX channels resulted in jumps, flickering and colour distortion in the lecterns.

The use of `time.perf_counter()` for high-precision timing loops significantly improved playback smoothness. Moreover, the strategy of preloading video data at startup eliminated runtime delays caused by pixel read and transformation operations.

Latency reduction by using the *Double-Buffer* pattern

The implementation of the *Double-Buffer* mechanism substantially improved visual fluidity. Independent buffers at each lectern enabled frames to be delivered in parallel and without interference.

Significant improvements in playback fluidity were observed after implementing this pattern. Tests carried out with panoramic videos and in `Different` mode showed a drastic reduction in perceived latency and a near-complete disappearance of visual errors. Even under conditions where resources were limited, such as with laptops without direct power, the system maintained virtually stable playback throughout the test sequence.

Additionally, random flickering caused by static frames was mitigated by introducing minimal controlled variation in the data sent, a fix verified through iterative testing.

Dynamic management of playback

The system was tested to ensure that it could interrupt an ongoing animation and start a new one on demand. This behaviour is crucial in interactive environments. Initial designs based on threads caused race conditions, later resolved through the adoption of asynchronous coroutines using `asyncio`.

Tests confirmed that the system responded correctly to requests from both the main contest system and the graphical interface. It executed new actions without any conflicts or the need to restart components. This dynamic responsiveness improves the system's reliability in interactive and demonstration environments.

Systematic testing

Automated testing was gradually introduced as development progressed to ensure maintainability and prevent regressions. In order to do this, a suite of automated tests was implemented covering both unit and integration levels:

- **Unit tests**: Focused on verifying the expected behaviour of the classes in isolation.
- **Integration tests**: Validated the coordination between endpoints, controllers and services, ensuring that requests from the front-end or quiz server triggered the appropriate internal workflows.

6.3 Physical assembly of the LED lecterns

LED projection was integrated into custom-built lecterns designed to meet the event's needs. Each lectern contains internal wiring and LED strips with connectors distributed across three illuminated surfaces. The following images show one of the lecterns, both inside and out, and the overall external appearance of the lecterns used during the Olympiads. The physical design and assembly of these lecterns was carried out by Furious Koalas S.L.

(a) Internal wiring of a lectern, showing power supplies and signal routing.

(b) External panel of the same lectern, with arranged LED strips.

(c) Front view of the three custom-built lecterns used during the Olympiads.

Figure 6.3: Physical assembly of the LED lecterns: internal components, external LED distribution, and overall visual design.

As shown in Figure 6.3a, the internal structure of each lectern consists of several key components. The circular device located at the top centre corresponds to the internal wiring for the buzzer button, which allows participants to signal when taking their turn. The red and black wire emerging from the top right corner connects to the external contestant microphone mounted on the lectern surface.

Multiple power supplies, adapters, and signal switchers are installed to ensure proper energy distribution and interface compatibility among components. In addition, a compact monitor is mounted inside the lectern to display the contestant's current score. It is connected via an HDMI cable and powered through the internal distribution system.

Finally, the LED wiring is routed through the bottom edge of the lectern. These connections enable individual control of the LED strips visible on the external front panel (see Figure 6.3b).

6.4 Web interface for testing and configuration

To support the development, debugging and validation of the LED control system outside the constraints of the live contest environment, a dedicated web interface has been implemented. This interface enables developers and testers to manage videos, configure playback actions and send animation commands to specific lecterns independently of the main quiz control infrastructure.

As illustrated in Figure 6.4, the interface supports a wide range of features that proved essential during the iterative development phase. These include video management (uploading, previewing and deleting videos), configuring actions (defining playback logic per lectern) and controlling animations in real time. Users can preview the outcome of any configuration without triggering projection, identify synchronisation issues in advance, and test animations individually or in coordinated patterns across lecterns.

The tool also supports looped playback and multi-mode actions, such as playing the same video on all lecterns or different videos on each lectern. It also provides a complete set of controls to interrupt or restart animations individually or globally. This flexibility enabled the backend logic and video pipeline to be refined efficiently before deployment.

In the long term, it is expected that this interface will scale with the system. If the LED infrastructure grows, for example by introducing new projection modes, the web interface can serve as the operational centre for orchestration and testing. Its modular architecture and decoupled design will facilitate integration with automated testing tools or higher-level scheduling systems in the future.

Further details and user guidance are available in the user manual (see Annex C).

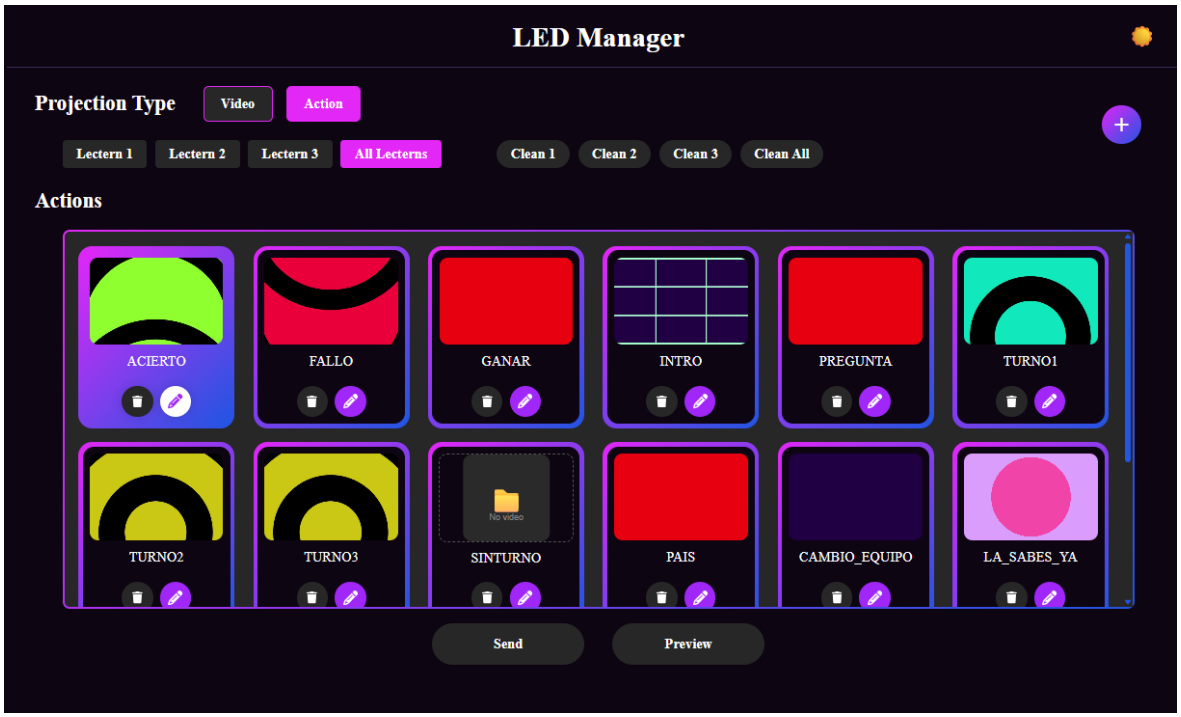


Figure 6.4: Web interface for testing and configuring animation projections. It enables the selection of target lecterns, definition of playback modes, and triggering of animations.

6.5 Production deployment: XVIII Castilla-La Mancha Informatics Olympiads

The system was deployed and used during the closing act of the **XVIII Castilla-La Mancha Informatics Olympiads** [1], held on Friday, April 25, 2025, in the hall of the Superior School of Informatics (ESI) of Ciudad Real. Organised by the University of Castilla-La Mancha, this event brought together teams of students from the ESO (secondary school), upper secondary and Higher Level Education Cycle programmes from all the region's provinces. They competed in several phases involving programming challenges, databases and operating systems.

On the final day, the three teams with the highest scores in each category advanced to a live stage competition inspired by televised quiz shows. The developed LED projection system was fully integrated into this final event as a key visual element, accompanying each phase with synchronised lighting and animations.

Unlike isolated demonstrations, this deployment required the LED system to operate in coordination with several live elements. Some of which are audio from the presenter's and contestants' microphones, background music, timed question displays, audience interaction tools, and real-time updates to the scoreboard and mobile app rankings. The system was also required to remain in sync with the use of jokers: *The Wise One*, *Fifty-Fifty*, and *Audience*

[1] Official article of the event available at: `https://esi.uclm.es/index.php/2025/04/30/cierre-de-las-xviii-olimpiadas-en-informatica-de-clm/`

Joker by the contestants. Their activation had to be tightly coordinated with visual feedback on the main screen and the internal game logic (e.g. disabling two incorrect answers or displaying public statistics).

The time constraints were also challenging. When a contestant pressed the buzzer, the central server initiated a countdown that was visible to the audience and players. Meanwhile, the LEDs responded immediately with contextual animations. This entire sequence had to remain tightly synchronised in real time, with no margin for visual desynchronisation or delay.

The LED control server received real-time commands from the main quiz server via HTTP and responded with seamless playback across the three lecterns, either individually or in panoramic mode. Furthermore, the system remained stable and performed consistently throughout the event, even under challenging lighting and acoustic conditions.

Use Cases at the Olympiads

The functionalities of the LED system required during the event were as follows:

- Projection of synchronised panoramic videos on the three lecterns using `Wide` mode.
- Video projection on the lecterns individually.
- Real-time reception of orders from the main quiz server via HTTP.
- Stable and flicker-free LED behaviour throughout the event.

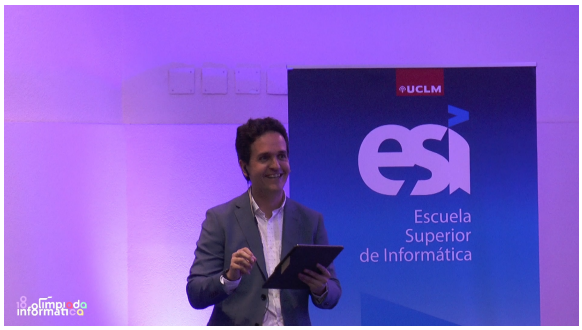
The system worked continuously throughout the event with no incidences. The experience confirmed both the robustness of the system and its suitability for real-world scenarios. This goal was achieved in front of a substantial audience and under the conditions of theatrical lighting.

Visual evidence of deployment

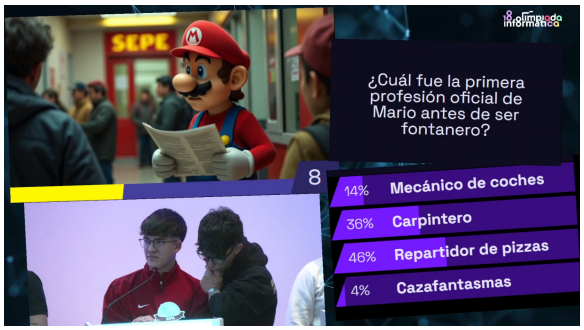
The following are some images from videos recorded during the event, where you can see the live installation and use of the system in real time:

(a) LED lecterns active during a live round.

(b) Audience during the contest final.



(c) Presenter coordinating the final round.



(d) Question display after using the audience joker during the constest.

Figure 6.5: Images of the XVIII Castilla-La Mancha Informatics Olympiads.

This video shows highlights from the XVIII Castilla-La Mancha Informatics Olympiads, where the developed LED projection system was deployed live during the final stage:

`https://www.youtube.com/watch?v=Ihtwd-TnYl0`

Chapter 7

Conclusions

In this chapter, the development of the project is assessed from a reflexive 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 lessons learned are highlighted. Finally, it proposes several directions for future improvements and possible extensions to the system.

7.1 Reached objectives

The LED control system developed during this project has successfully met the initial objectives at both the general and specific levels.

The creation of a custom system capable of controlling programmable LEDs integrated into quiz lecterns has been achieved. The implemented system supports synchronised operation with the main quiz system and ensures smooth interaction throughout the entire lighting sequence. This was shown at the XVIII Castilla-La Mancha Informatics Olympiads, which was held on April 25, 2025 where the system was used as part of a system deployed on production.

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

- **Facilitate system usability for non-technical users**: This objective has been achieved by developing a lightweight web interface designed using HTML, CSS and TypeScript. This interface enables users to preview animations in a visual modal before triggering them on the LED lecterns with a single interaction. Furthermore, it enables the management of actions and video and display user feedback directly on the interface.
- **Ensure visual fidelity and synchronisation**: Minimising latency and ensuring that the LED output closely matched the original video content proved to be a significant technical challenge. Although no formal latency measurements were recorded, qualitative testing revealed a significant decrease in frame delay and enhanced synchronisation with each successive iteration. This was successfully addressed by implementing a double buffering system that separates the writing and reading of frames. A smooth projection that preserves visual fidelity even under high demand conditions has been

enabled by this mechanism, coupled with time-relative synchronisation strategies using `time.perf_counter()`.

- **Integrate the system with the quiz platform**: Commands originally transmitted through sockets can now be responded to by the system, which is fully integrated with the main server via HTTP requests. A mapping between internal actions and video routes has been created using a JSON configuration file. Moreover, the server was containerised using Docker to enable integration within the internal network of the quiz platform.
- **Validate the system in a real-world context**: This goal was achieved by deploying the prototype at the XVIII Castilla-La Mancha Informatics Olympiads, where it ran continuously throughout the contest. Its behaviour was checked and enhanced during testing sessions before the event, which showed that it was ready to be used on production environments.
- **Contribute to cost reduction and platform competitiveness**: The previously used licenced software for LED control software (MadMapper) was successfully replaced, proving that the required technical functionalities are provided by the developed solution. This means that the company will not need to rely on external tools and will have more control over its technical infrastructure.

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 from start to finish, covering everything from identifying requirements to real-world deployment. The system was validated at the XVIII Castilla-La Mancha Informatics Olympiad, where it proved to be stable under event conditions. The architecture was structured in layers to ensure reliability, enabling independent testing of modules and reducing coupling between them. Moreover, code documentation and modular design principles contributed to the system's long-term maintainability.

  During the process, the development workflow was in line with real-world practices used at Furious Koalas S.L., where agile methodologies and iterative feedback cycles played a key role. This dynamic allowed for progressive refinement and adaptation to evolving requirements, which reinforced the robustness and extensibility of the system.

- **[IS3] Ability to solve integration problems according to available strategies, standards, and technologies**: One of the main technical challenges was integrating the LED control module with the main quiz platform. At first, it was made as a Python application on its own. However, it was changed to receive requests from the main server using HTTP-based APIs instead of raw sockets.

  The solution involved using Docker to containerise the application, thereby simplifying its deployment within the existing infrastructure. Clear internal interfaces were also designed to manage the video-to-action mapping logic. These integration tasks involved dealing with system-level dependencies, network configuration and other external constraints.

7.3 Personal conclusion

This project has been the most demanding and transformative stage of my undergraduate studies so far. From the start, the aim of creating a real-time LED control system able to interact with external hardware and adapt to a live quiz environment, felt ambitious and perhaps slightly beyond reach. Nonetheless, this initial uncertainty gradually evolved into a structured process of exploration, experimentation, and consolidation.

One of the things I value most about this experience is that it forced me to use unfamiliar tools and protocols without any predefined instructions. Working with Art-Net, managing asynchronous video streaming and integrating everything into the main quiz system meant I had to learn to make informed decisions based on observation, performance and constraints, rather than just theory.

The process was not linear. There were frustrating moments caused by unexpected latency and incompatibilities between components, as well as architectural decisions that later had to be reconsidered. However, it was precisely these challenges that offered the greatest learning opportunities. Each obstacle introduced a new layer of complexity, requiring me to take a step back and reconsider my assumptions.

The project has taught me much more than how to deliver a functional system. It has transformed my approach to software development, helping me to think more independently, structure code more professionally and find solutions before seeking assistance.

Although I had not previously worked closely with hardware, I found real satisfaction in the visual feedback that came with each step forward. Seeing the LEDs react to the code I had written made each small advance feel like a tangible result, almost like a reward. This made progress visible in a very literal sense and kept me motivated, even through the most frustrating stages.

Even though the system could still be improved, I feel genuinely proud of what I achieved. Seeing the system work in real time during the contest made all the effort worthwhile and proved that I can overcome complex challenges independently.

7.4 Future work

Throughout the development and testing phases of the system, several areas for improvement and extension have been identified. The current implementation fulfils its intended purpose under the given conditions. Nonetheless, adjustments or enhancements to the architecture may be required to support more demanding scenarios or improve long-term usability. These suggestions are categorised below as either structural constraints or potential functionality upgrades.

Potential architectural and performance-related improvements

- **Scalability in multi-lectern environments**: The system currently handles three lecterns concurrently without issues, but scaling it to larger setups could stress the current asynchronous model. Performance degradation or desynchronisation may occur due to packet saturation. However, such scenarios are unlikely to be experienced within the expected use cases of Furious Koalas S.L., which typically involve a fixed number of three lecterns. Future iterations might consider either distributing tasks across processes or using lightweight microservices to mitigate these risks.
- **Frame rate saturation**: Playback performance deteriorates when the system attempts to process videos exceeding 30 FPS. This is in part due to an intentional cap imposed to preserve the controller's durability and in part due to a hardware limitation: the Art-Net controller cannot reliably handle more than 50 FPS. For applications requiring higher refresh rates, although unlikely, this represents an insuperable barrier that would require alternative hardware or architectures.
- **Universe and channel limitations**: Deskontroller 16 supports up to 16 universes (8,192 channels), nine of which are already occupied in the current configuration. This restricts the maximum number of controllable LEDs unless additional controllers are deployed or the distribution of universes is further optimised.
- **Format compatibility**: Currently, the system only supports MP4 and MOV formats due to incompatibilities with other codecs at an early stage. Adding automatic format validation or integrating a conversion pipeline could improve flexibility and user experience.
- **Security foundations**: As the current deployment is strictly local and limited to internal use by FK employees, no specific security measures were adopted for the initial version. However, future deployments in broader or more exposed environments may benefit from the implementation of security aspects.

Improvements could include serving the application over HTTPS using Nginx, enabling basic authentication for sensitive endpoints and validating user input to prevent script injection or data corruption. Furthermore, the implementation of technical measures such as static code analysis, firewall rules, and CORS restrictions for trusted local IP addresses could reduce the attack surface and improve isolation. Finally, applying secure HTTP headers (such as `Content-Security-Policy` or `Strict-Transport-Security`) and enabling access logs may provide useful mechanisms for protection and traceability in shared environments.

Functional upgrades and usability features

- **Real-time visualisation**: Adding a real-time view of the LED states to the web interface, could provide valuable feedback to users if the lecterns are not in their field of vision when projecting an animation.
- **Action import**: Allowing the direct import of actions from JSON files via the interface would simplify complex setups. This feature should include validation to detect malformed or incompatible files before changes are applied.
- **User-defined organisation**: Future versions could enable users to organise videos and actions into folders within the interface, which would improve clarity when using the software on a large scale. Other small details, such as enabling custom image covers for each item, could also enhance usability, particularly in collaborative or multi-user environments.

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 - Video Playback on LEDs**
Rol	**As** system user
Funcionality	**I want** to select a video and send it to the lecterns
Utility	**To** play it back synchronously on the LEDs
Acceptance criteria	• The video must be projected while preserving the original FPS. • There must be no delays or flickering in the LEDs. • The image must be adapted to the physical proportions of the lectern.

Id	**US02 – Sending Videos to Individual Lecterns**
Rol	**As** system user
Funcionality	**I want** to send a specific video to a single lectern
Utility	**To** experiment with independent lighting effects
Acceptance criteria	• Each lectern must be able to receive a video command individually. • Videos must play back in a stable and synchronised manner.

Id	US03 – Panoramic Video Playback
Rol	**As** system user
Funcionality	**I want** to play a panoramic video across the three lecterns
Utility	**To** generate a continuous image
Acceptance criteria	• The video must be automatically resized and segmented. • Playback must be simultaneous and synchronised across all lecterns.

Id	US04 – Intuitive Graphical Interface
Rol	**As** system user
Funcionality	**I want** a clear user interface
Utility	**To** view all available actions and videos
Acceptance criteria	• Actions must be clearly labelled. • It must allow uploading videos, selecting lecterns, and sending playback commands.

Id	US05 – Integration with the Main Quiz System
Rol	**As** system user
Funcionality	**I want** the system to work along with with the main quiz system
Utility	**To** send projections from a single control point
Acceptance criteria	• The application must expose HTTP routes to trigger projections. • The system must remain operational independently if no external command is received.

Id	US06 – Video Preview Before Launch
Rol	**As** system user
Funcionality	**I want** to see how the video resized before projection
Utility	**To** verify it does not become too deformed
Acceptance criteria	• The system must allow uploading and previewing videos through the interface. • The preview ratio must simulate the actual format of the lecterns.

Id	US07 – Controlled Interruption of Ongoing Projection
Rol	**As** system user
Funcionality	**I want** to interrupt the playback of a video
Utility	**To** play a new animation without waiting for the current one to end
Acceptance criteria	• The projection must stop upon receiving a new command. • No concurrency errors or loss of synchronisation must occur. • The new video must start immediately after the interruption.

Id	US08 – Management of Available Videos
Rol	**As** system user
Funcionality	**I want** to be able to add and remove videos through the interface
Utility	**To** easily organise the system's multimedia resources
Acceptance criteria	• The interface must allow the addition of `.mp4` or `.mov` files from a specific folder. • Files in unsupported formats must be rejected. • Videos must be displayed in a list that updates in real time.

Id	US09 – Action Management
Rol	**As** system user
Funcionality	**I want** to be able to define, modify and delete actions
Utility	**To** organise the available commands and associate them with specific videos
Acceptance criteria	• The ID of each action must be unique and validated before creation. • It must be possible to assign a representative name to each action. • Each action must be linked to zero, one, or three videos existing in the system. • The interface must only allow modifying and deleting actions that have already been defined.

Appendix B

B.1 Evolution through iterations

Iteration 1 – February 3 to 7	
Main activity	General understanding of the contest system and the LED controller. Initial setup of `pyartnet` and validation of the environment through connectivity tests.
Related User Stories	-
Obstacles and solutions	Art-Net packets were being sent to an incorrect IP address, preventing communication with the controller. The issue was resolved by reviewing and correcting the network configuration.
Key outcomes	Successful connection to the DMX controller and first LEDs switched on via Python code. Initial understanding of the `pyartnet` library. Feasibility of LED control without MadMapper confirmed.

Table B.1: Summary of results – Iteration 1

Iteration 2 – February 10 to 14	
Main activity	Further exploration of the `pyartnet` library and analysis of the system's behaviour when attempting to light up a specific LED. Initial design decisions regarding RGB data representation and transmission.
Related User Stories	-
Obstacles and solutions	Incomplete packet transmission was causing random LED activations. This was resolved by generating all the channels of each universe, even those not actively used. The refresh rate was also adjusted until a stable value was found (0.02s).
Key outcomes	A single LED was successfully switched on in a precise and controlled manner. Further understanding of the controller's architecture was achieved, and design alternatives for data organisation were considered (lists vs arrays, lists vs deques).

Table B.2: Summary of results – Iteration 2

Iteration 3 – February 17 to 21	
Main activity	Development of the pixel rearrangement algorithm to adapt images to the physical layout of the LEDs. First tests with video files. Transition towards using `deque`-based structures.
Related User Stories	US01, US02
Obstacles and solutions	The order of the pixels retrieved from the image did not match the physical LED layout. This was solved by applying an algorithm to divide it into zones consistent with the physical topology.
Key outcomes	Successful projection of images and first videos across the three lecterns. The rearrangement algorithm proved viable. The need to synchronise playback speed between the original video and the LED output was recognised.

Table B.3: Summary of results – Iteration 3

Iteration 4 – February 24 to 28	
Main activity	Implementation of playback speed control to match the original video's FPS. Projection of panoramic video (wide mode). Initial development of the Flask web server and first architectural decisions (separation of frontend and backend).
Related User Stories	US01, US2, US03
Obstacles and solutions	Playback delays persisted when using `time.time()`, which were mitigated by switching to `time.perf_counter()`. Additional LED flickering issues due to controller overload and packet loss were identified using Wireshark. The use of threads and `asyncio` tasks was evaluated as a potential solution.
Key outcomes	Smoother playback achieved with synchronisation adjusted to the original FPS. The first panoramic projection was completed. The structural separation between frontend and backend was established as a long-term architectural decision.

Table B.4: Summary of results – Iteration 4

Iteration 5 – March 3 to 7	
Main activity	Initial implementation of the Flask server and first functional design of the frontend. Reception of commands through specific routes and later refactoring into a single POST route for greater flexibility.
Related User Stories	US01, US02, US03, US04
Obstacles and solutions	Difficulties arose in executing coroutines due to lack of full control over the event loop[1]. This was resolved by either creating new loops or reusing existing ones as appropriate. A setter was also introduced in the controller to manage active tasks when receiving new commands.
Key outcomes	First fully functional workflow from the web interface to LED video playback. Successful refactoring into a single POST route `/play`, optimising control logic.

Table B.5: Summary of results – Iteration 5

Iteration 6 – March 10 to 14	
Main activity	Design of the video previewer in the frontend using TypeScript and Node.js. Implementation of a playback modal with behaviour adapted to the number of videos sent. Improvement of the visual feedback provided to the user.
Related User Stories	US04, US06
Obstacles and solutions	Synchronising three independent videos proved challenging, especially when using the playback bar. The issue was resolved by handling `seeking` events to avoid loops between `stop` and `play`. However, in future versions this synchronisation is removed, as I found it unnecessary.
Key outcomes	A more refined frontend ready for real testing scenarios. Functional visual previewer incorporated, along with selective video sending to individual lecterns.

Table B.6: Summary of results – Iteration 6

[1] See asyncio event loops documentation in `https://docs.python.org/3/library/asyncio-event loop.html`

Iteration 7 – March 17 to 21	
Main activity	Separation between data processing and transmission. Implementation of a workflow in which the `VideoService` writes the frames to a JSON file and the `LEDService` reads them continuously.
Related User Stories	US01
Obstacles and solutions	Concurrent access to the JSON file was causing exceptions and frame loss. A `lock` mechanism was introduced to ensure that reading did not occur during writing. Strategies for preloading videos were also considered to prevent real-time processing bottlenecks.
Key outcomes	Effective separation between content generation and emission, enabling asynchronous control. The `Video` and `Frame` classes were introduced, and the system was prepared for a future transition to more efficient memory-based buffers.

Table B.7: Summary of results – Iteration 7

Iteration 8 – March 24 to 28	
Main activity	Replacement of the JSON-based system with a DoubleBuffer pattern. Each lectern now has its own independent read and write buffer. Active task cancellation is also managed.
Related User Stories	US01, US02, US07
Obstacles and solutions	Tasks accessing the same buffer simultaneously were causing visual artefacts and data fragmentation. This was resolved by assigning an independent double buffer to each lectern, eliminating conflicts. Frame loss issues due to shared video objects were also addressed using `deepcopy`.
Key outcomes	Optimised buffer system without the need for disk writing. Each task now handles its own data flow, with immediate interruption capability. The system became more stable, accurate, and modular.

Table B.8: Summary of results – Iteration 8

Iteration 9 – March 31 to April 4	
Main activity	Redesign of the system to support standard commands from the main server using the `(action, device)` format. An intermediate JSON file was created to map actions to videos. Looping playback was also implemented for selected actions.
Related User Stories	US01, US05
Obstacles and solutions	Difficulty in correctly translating socket-based commands to the REST model used by the LED server. This was resolved by creating a `/playJSON` route and an `action_list.json` file containing metadata for each action.
Key outcomes	The system was integrated with the main server and became compatible with external action commands. It gained the ability to loop videos until a new command was received. Internal task and shared resource handling was improved.

Table B.9: Summary of results – Iteration 9

Iteration 10 – April 7 to 11	
Main activity	Full integration of the LED system into the Docker environment of the main server. Real-world testing was made. Flickering issues were resolved.
Related User Stories	US01, US05
Obstacles and solutions	Network issues between Docker containers were resolved by configuring a shared internal network. Persistent flickering was mitigated by introducing minimal continuous changes to a single LED, preventing extended inactivity.
Key outcomes	The system became fully operational within the final deployment environment. Critical issues were proved under real conditions. The LED server's robustness and reliability improved significantly in response to simultaneous commands and uniform video playback.

Table B.10: Summary of results – Iteration 10

Iteration 11 – April 21 to 25	
Main activity	Physical installation of the LED system on the event stage. Integration with additional hardware (StreamDeck) and coordination with the main server. Rehearsals were carried out, along with live support during the XVIII Informatics Olympiads.
Related User Stories	US01, US05
Obstacles and solutions	Errors occurred when pressing multiple buttons simultaneously. This was resolved by improving the task queuing system, thereby avoiding overlaps and loss of order.
Key outcomes	The system operated successfully during the event. Animations were projected synchronously and reliably, even when multiple commands were issued simultaneously. The backend's robustness and the system's modularity were validated in a real-world setting.

Table B.11: Summary of results – Iteration 11

Beyond the structured development iterations, additional work was carried out throughout the month of May to consolidate and prepare the system for long-term maintainability. This included the refactoring of the backend into modular components using Flask Blueprints, the development of unit and integration tests for the main services, and the extension of the frontend interface with tools for adding, editing and deleting LED actions (US04, US 08, US09). These tasks strengthened the reliability, testability, and usability of the system, ensuring its readiness for future adaptations and deployments.

Appendix C

C.1 Deployment Instructions

System Requirements

Before running the LED control system, ensure that the following components are installed:

- Python 3.11 or higher (only for manual execution)
- Docker and Docker Compose (for containerised deployment)
- Web browser to access the frontend

Execution Modes

The system can be launched in two ways:

- **Integrated Mode:** running as part of the main quiz server network
- **Standalone Mode:** running independently for local control

Running with the Main Server (Integrated Mode)

1. Open a terminal and navigate to the Docker project folder:

   ```
   cd leds-management-project/led_management_docker
   ```

2. Remove any existing container (if needed):

   ```
   docker rm led_management_app
   ```

3. Build the Docker image:

   ```
   docker build -t led_management_app .
   ```

4. Launch the container within the main server network:

```
docker run -d --name led_management_app \
  --network=yoroppa-main-system-server_default \
  -p 4000:4000 led_management_app
```

5. Confirm both containers are on the same Docker network:

```
docker network inspect yoroppa-main-system-server_default
```

6. You can access to the interface searching `http://localhost:4000` on your web browser.

Note: `yoroppa-main-system-server` must be already built in Docker.

Running Independently (Standalone Mode)

There are two alternatives for running the system locally without the main server:

A) Using Docker (recommended)

1. Open a terminal and navigate to the Docker project folder:

```
cd leds-management-project/led_management_docker
```

2. Build the Docker image:

```
docker build -t led_management_app .
```

3. Run the container, exposing port 4000 to the host:

```
docker run -d --name led_management_app -p 4000:4000 led_management_app
```

4. Open your browser and navigate to `http://localhost:4000`

B) Running Manually with Python (for development)

1. Open a terminal and navigate to the Docker project folder:

```
cd leds-management-project/led_management_docker
```

2. Create and activate a virtual environment:

```
python -m venv ledenv
source ledenv/bin/activate  # On Windows: ledenv\Scripts\activate
```

3. Install the required dependencies:

```
pip install -r requirements.txt
```

4. Run the Flask server:

```
python .\app.py
```

5. Open your browser and go to `http://localhost:4000`

C.2 User Manual

User Interface Overview

The system provides a web interface accessible from any browser. This interface allows the user to manage and trigger LED animations associated with quiz actions or video content.

Main Features

- **Video Management**: Users can add or delete videos from the interface.
- **Action Management**: Users can add, edit or delete actions directly from the interface.
- **Animation Preview**: Users can preview available videos and actions from the interface.
- **Animation Playback**: Users can send videos and actions to the LEDs.

Visual Guidance

Video Management

The system allows users to manage the available videos used for LED projection. This includes uploading and deleting files, previewing them, and sending them to specific lecterns.

Adding a New Video

To add a video, the user must click the `New Video` button, as shown in Figure C.1.

- Only files with extension `.mp4` or `.mov` are accepted.

- It is not allowed to upload videos with a name that already exists in the system, regardless of the file extension. For example, if `a.mp4` has already been added, it will not be possible to upload `a.mov` (Figure C.2).
- After a successful upload, a confirmation modal will be displayed (see Figure C.3), which disappears automatically after 5 seconds.

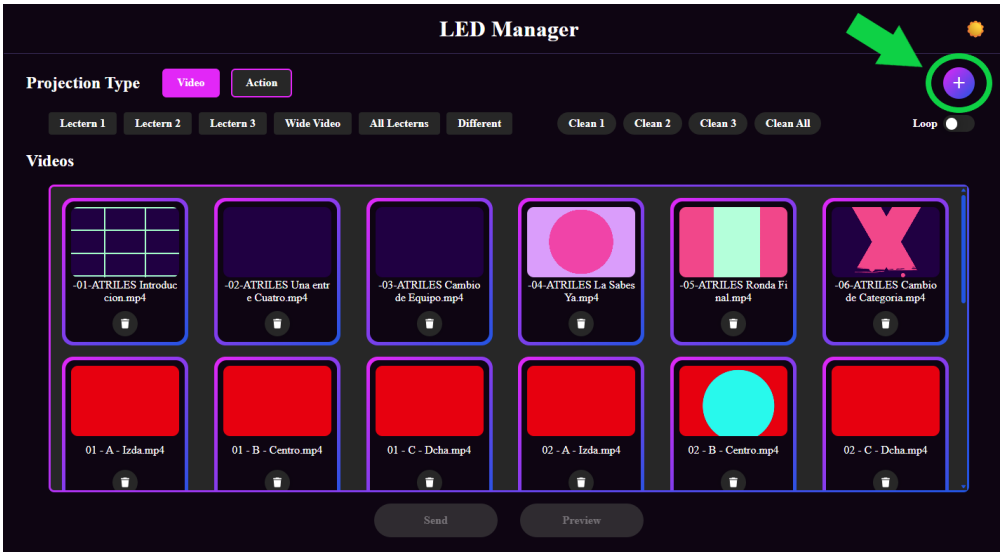


Figure C.1: Button to add a new video to the system (top-right of the screen)

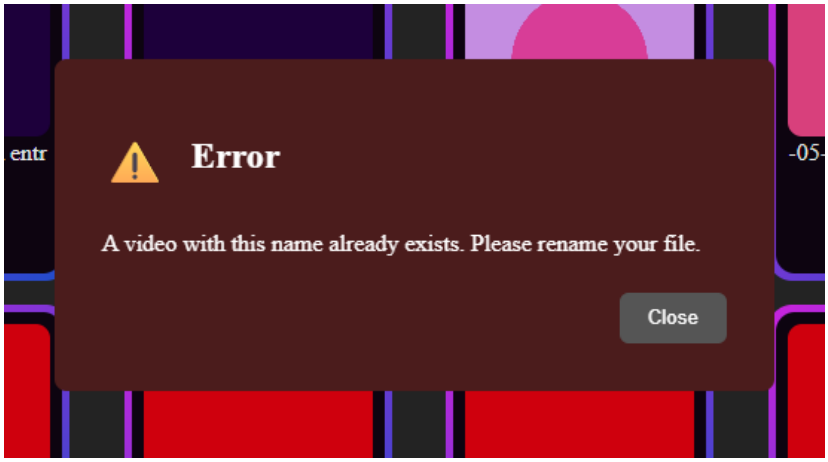


Figure C.2: Modal shown after successful video upload

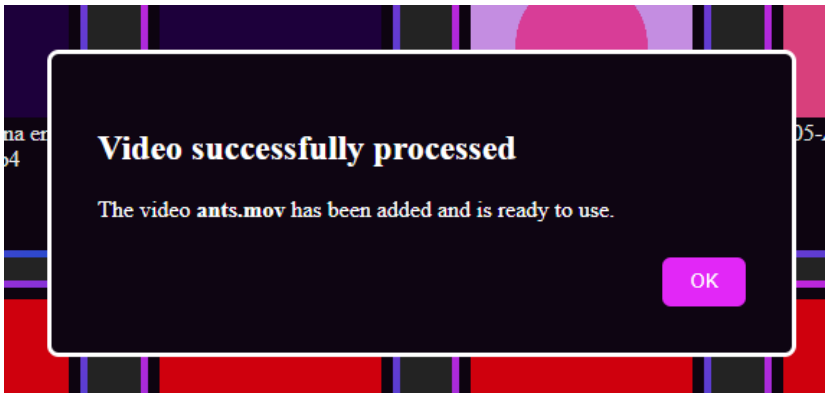


Figure C.3: Modal shown after successful video upload

Deleting a Video

To delete a video from the system, the user must click the delete icon associated with the desired file, as shown in Figure C.4.

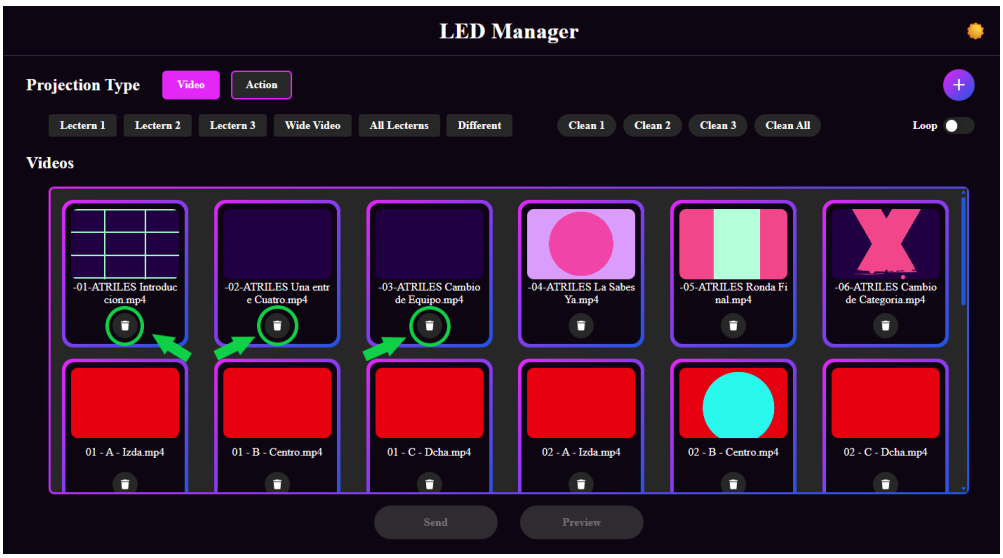


Figure C.4: Delete button associated with each uploaded video

The behaviour of the system varies depending on whether the video is currently linked to any configured actions:

- **If the video is not associated with any action:** a confirmation modal will be displayed asking the user to confirm the deletion (As seen in Figure C.5).
- **If the video is associated with one or more actions:** the system will prevent deletion and display a modal indicating that the video cannot be removed. This modal also shows the list of associated actions, as illustrated in Figure C.6.

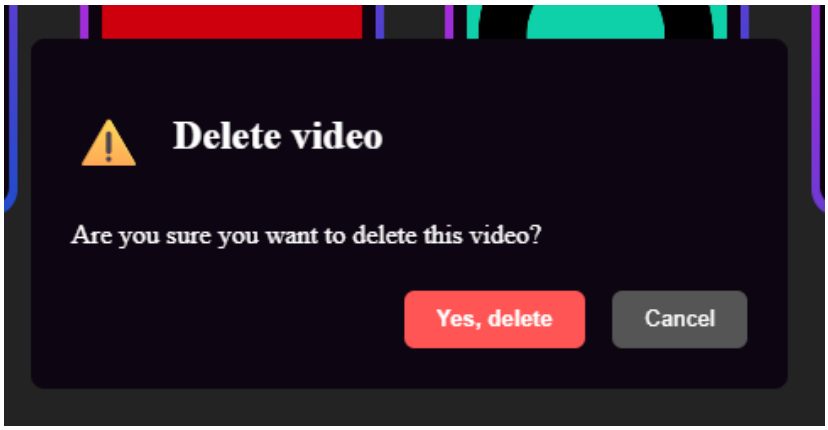


Figure C.5: Warning modal shown when attempting to delete a video linked to one or more actions

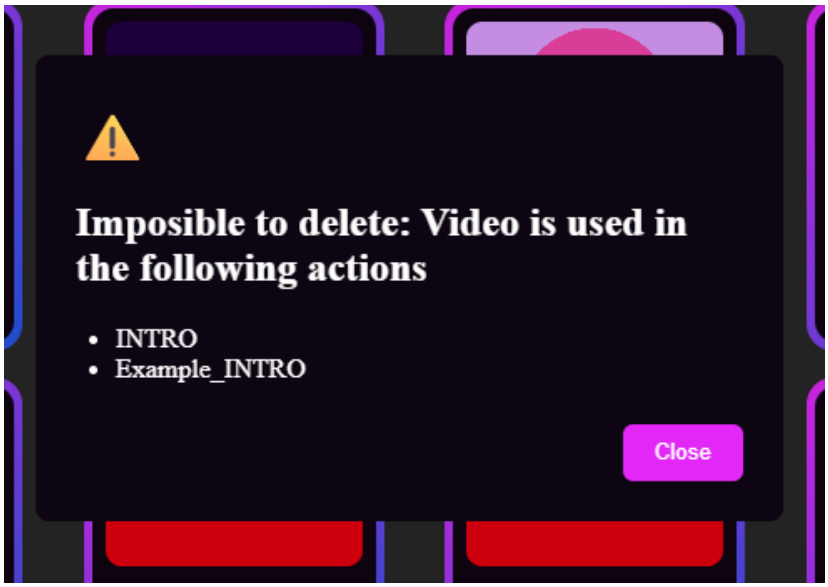


Figure C.6: Warning modal shown when attempting to delete a video linked to one or more actions

Action Management

The system allows users to create, edit, and manage actions that can later be triggered from the main interface or by external systems.

Adding or Editing an Action

To add a new action, the user must click on the `New Action` button, which is placed at the top right of the screen. To edit an existing one, the edit icon of the action must be used. Both options open the same modal form. The placement of the buttons is illustrated in Figure C.7.

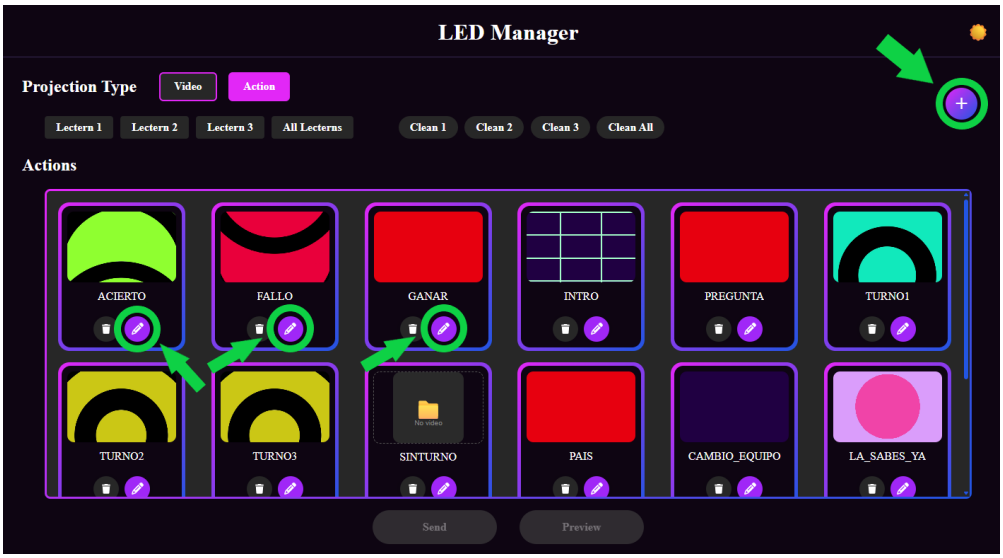
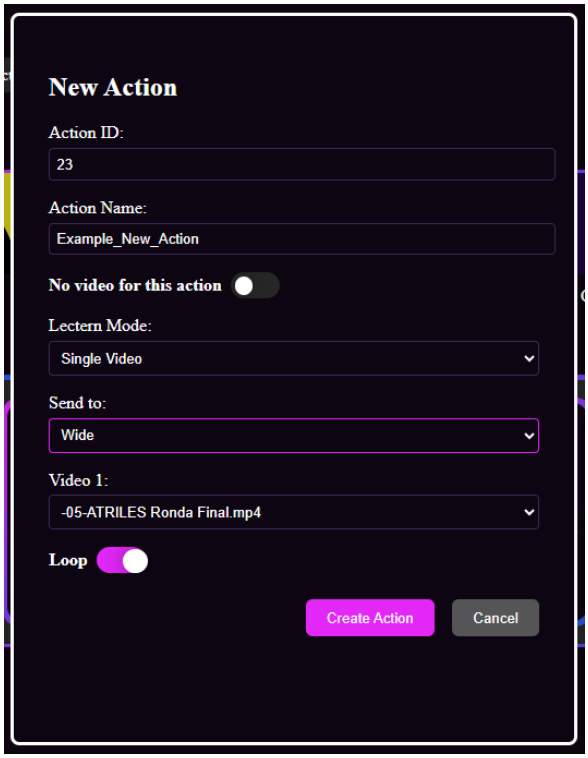


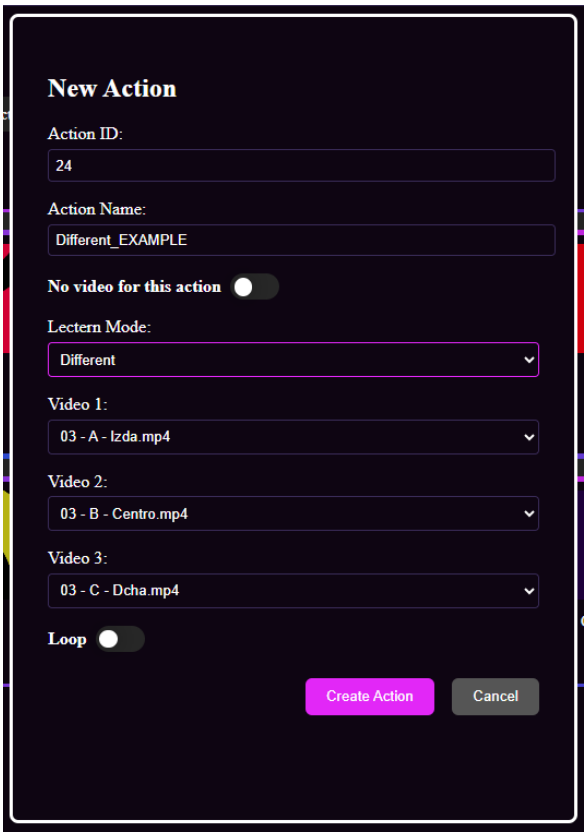
Figure C.7: Action Mode, highlighting add and edit action buttons.

Depending on the type of action, the modal shows different input configurations:

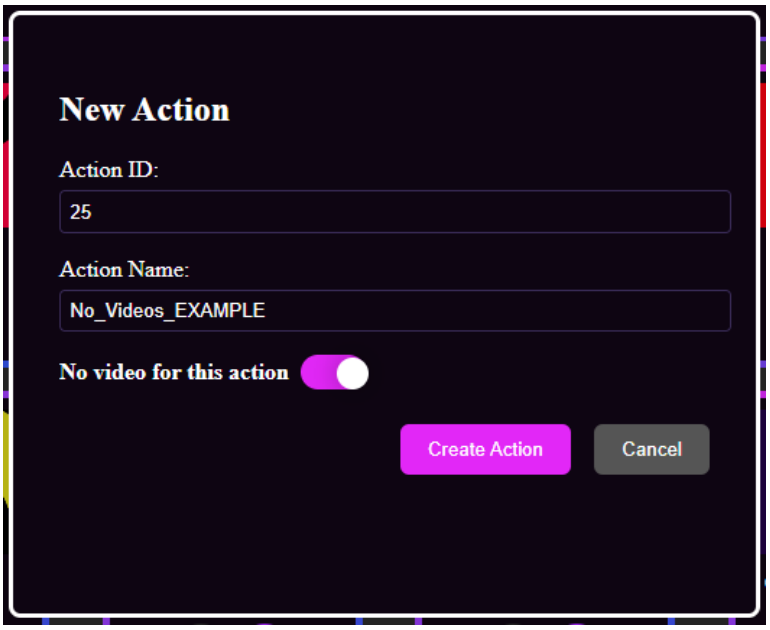
- **Single Video Action:** An action assigned to all lecterns or to a wide projection mode (`AllLecterns` or `Wide`). This action is associated with a single video. The user can also enable or disable looping for this video. See Figure C.8a.
- **Different Action:** A special type of action that allows each lectern to play a different video simultaneously. In this case, the modal displays three fields, one per lectern. The set of videos can be configured to be in loop mode. See Figure C.8b.
- **Empty Action:** An action that is initially created without any video associated. This configuration allows defining an interruptive action. See Figure C.8c.



(a) Single Video action with one video for all lecterns.



(b) Different action with one video per lectern.



(c) Action without any associated video.

Figure C.8: Modal view for the configuration of actions, showing different assignment types.

The system enforces the following constraints:

- **Unique ID:** Each action must have a unique identifier. The system prevents saving an action whose ID is already in use.
- **Video requirements:** When saving an action of type `Single Video` or `Different`, it is mandatory to select the appropriate number of associated videos.

Note: All actions with associated videos can be send to a specific lectern. Choosing between `AllLecterns`, `Wide` and `Different` will define how the video will be played when sending the action to all lecterns.

Deleting an Action

To delete an existing action, the user must click on the corresponding delete icon, as shown in Figure C.9.

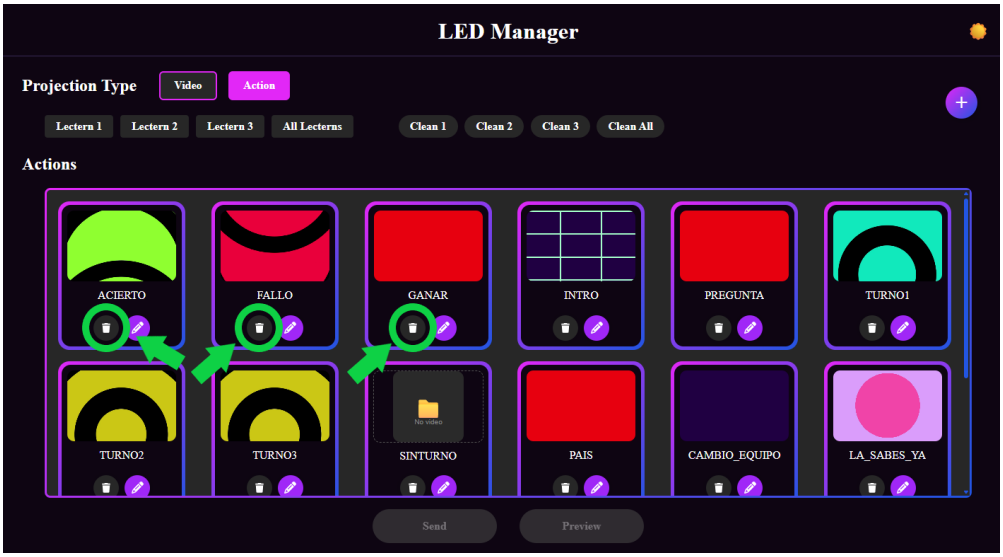


Figure C.9: Delete button associated with each action

When you click, a confirmation modal will appear, asking you to confirm the deletion:

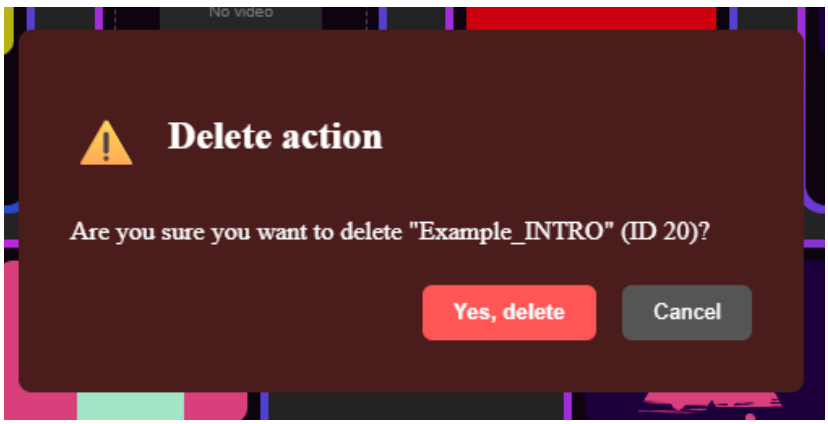


Figure C.10: Confirmation modal for deleting an action

Preview and Playback Controls

The system allows users to either preview the selected video(s) before sending them to the LED lecterns, or directly trigger playback on one or more lecterns. This functionality is identical in both `Video Mode` and `Action Mode`.

Selection Requirements

To use either the preview or playback options, the user must first configure the following:

- **In Video Mode:** The user must choose the projection type and a video. In case of selecting `Different`, three videos must be chosen.
- **In Action Mode:** The user must choose both a target lectern and an action previously defined in the system.

Previewing a Video

Once the configuration is complete, pressing the `Preview` button opens a modal that shows a visual representation of the selected content before it is sent to the LEDs. This helps the user verify the result without triggering a live projection.

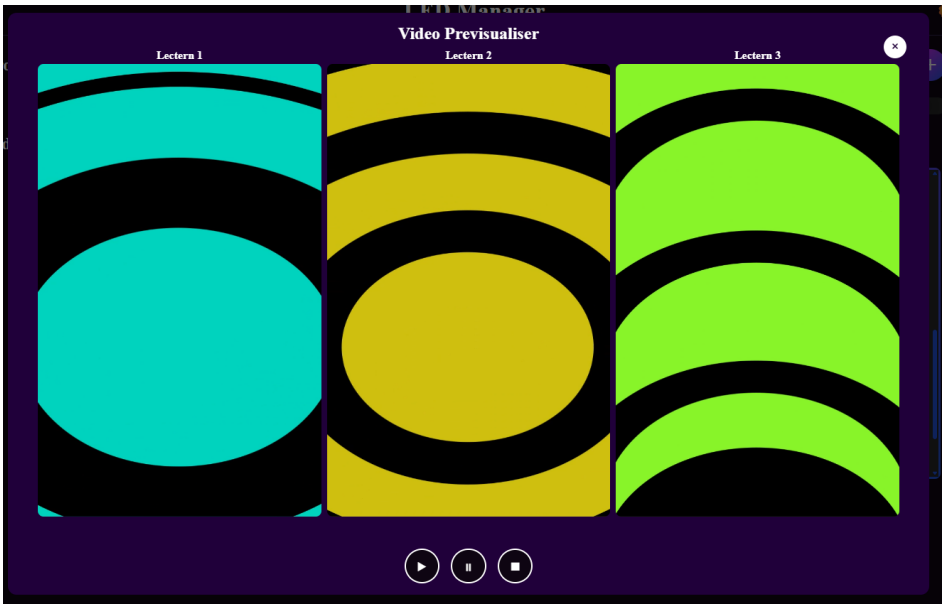


Figure C.11: Modal for video preview before LED projection

Sending a Video or Action to the Lecterns

Once the selection is correctly set, clicking the `Play` button will send the chosen video(s) or action to the specified lectern(s).

Looping Option: For video mode, the user may enable a loop option via a checkbox. When enabled, the selected video(s) will repeat indefinitely until stopped manually. In case of the actions, looping is defined when the action is created or modified.

Interrupting Playback: Clean Buttons

The system includes control buttons that allow the user to interrupt the ongoing projection at any moment. There are individual controls for each lectern as well as a global one:

- `Clean 1`: Stops playback on Lectern 1
- `Clean 2`: Stops playback on Lectern 2
- `Clean 3`: Stops playback on Lectern 3
- `Clean All`: Stops playback on all lecterns at once

A full example of a selected configuration with the associated playback and clean controls is shown in Figure C.12.

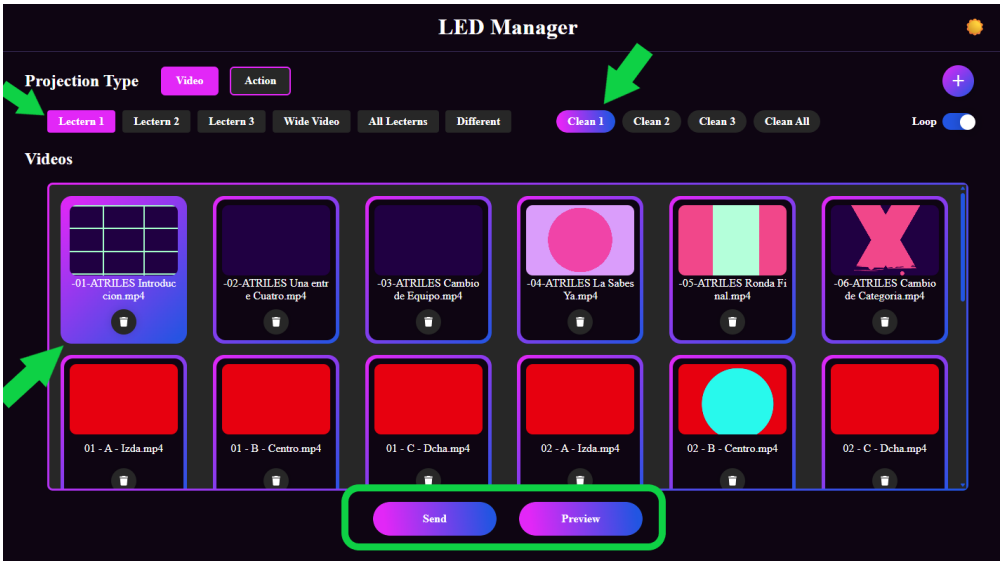


Figure C.12: Selection of video and action, with preview/play buttons and clean controls highlighted

Light Mode Option

The interface also supports a light theme for improved visibility in bright environments. Users can switch between dark and light modes using the theme toggle button located in the top-right corner of the interface.

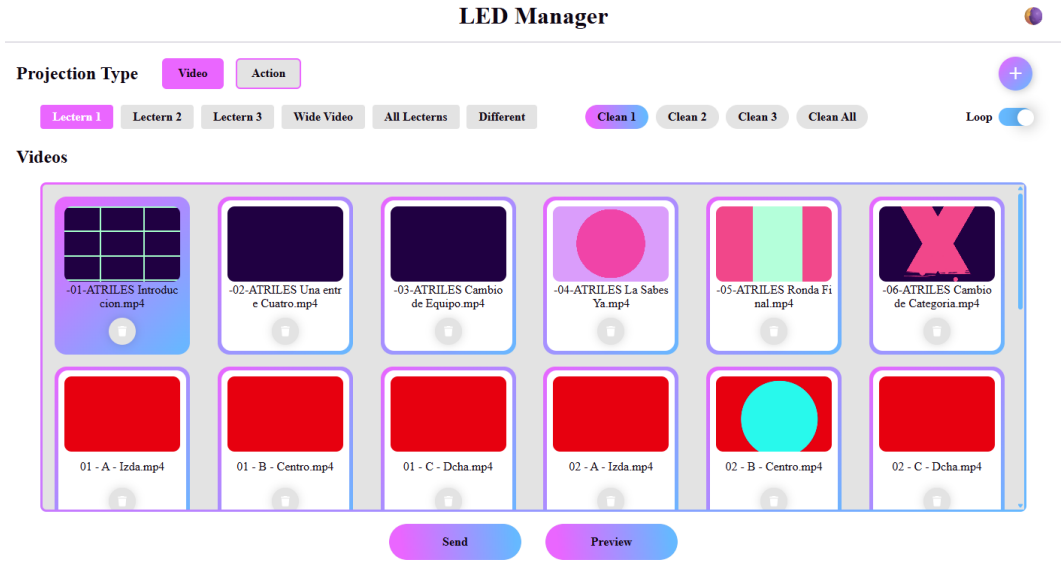


Figure C.13: Example of the interface in light mode

References

[88L17] 88Light. LED SD Card Controller T1000S - Datasheet. https://www.88light.com/Datasheet/88L-%20SD%20Card%20CONTROLLER%20T1000S%20DATASHEET%20-170217.pdf, 2017.

[AEC16] Ltd. APA Electronic Co. APA102 Addressable RGB LED Datasheet, 2016.

[AFea18] Dahlan Abdullah, F Fajriana, y et al. Application of Interpolation Image by using Bi-Cubic Algorithm. En *Journal of Physics: Conference Series*, volume 1114, página 012066. IOP Publishing, 2018.

[Art24] Artistic Licence. Specification for the Art-Net 4 Ethernet Communication Protocol, 2024.

[Bab12] Seyed Morteza Babamir. *Real-Time Systems: Architecture, Scheduling, and Application*. IN-TECH, 2012.

[BEea07] Oliver Bimber, Andreas Emmerling, y et al. Passive-Active Geometric Calibration for View-Dependent Projection. *Journal of Virtual Reality and Broadcasting*, 4(1), 2007.

[BS14] Massimo Banzi y Michael Shiloh. *Getting Started with Arduino*. Maker Media, Inc., edición 3rd, 2014.

[CJ09] Lei Chen y Yunde Jia. A Parallel Reconfigurable Architecture for Real-Time Stereo Vision. En *2009 International Conference on Embedded Software and Systems*, 2009.

[CXea21] Yixi Cai, Wei Xu, y et al. ikd-Tree: An Incremental K-D Tree for Robotic Applications, 2021.

[DDea11] Sebastian Deterding, Dan Dixon, y et al. From Game Design Elements to Gamefulness: Defining Gamification. En *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments, MindTrek 2011*, volume 11, 2011.

[Des17] Deskontrol Electronics. *Deskontroller 16 – User Manual*, 2017. url: https://www.deskontrol.net/descargas/manuales/deskontroller-8X2-16-user-manual.pdf.

[DK08] Russell D. Dupuis y Michael R. Krames. History, Development, and Applications of High-Brightness Visible Light-Emitting Diodes. *Journal of Lightwave Technology*, 26(9):1154–1163, 2008.

[DOOC15] Ltd. Dongguan OPSCO Optoelectronics Co. SK6812RGBW LED Datasheet. https://cdn-shop.adafruit.com/product-files/2757/p2757_SK6812RGBW_REV01.pdf, 2015.

[Dun15] Richard E. Dunham. *Stage Lighting: Fundamentals and Applications*. Routledge, 2015.

[EKea] Martin Ester, Hans-Peter Kriegel, y et al. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise.

[Ent18] Entertainment Services and Technology Association (ESTA). ANSI E1.31 – 2018: Lightweight streaming protocol for transport of DMX512 using ACN, 2018.

[EST24] ESTA – Entertainment Services and Technology Association. ANSI E1.11 – 2024 Entertainment Technology—USITT DMX512-A Asynchronous Serial Digital Data Transmission Standard for Controlling Lighting Equipment and Accessories. Technical report, ESTA, 2024.

[FBea15] Euan Freeman, Stephen Brewster, y et al. Interactive Light Feedback: Illuminating Above-Device Gesture Interfaces. En *Human-Computer Interaction – INTERACT 2015*, Cham, 2015. Springer International Publishing.

[Gar24a] GarageCube & 1024 Architecture. *Introduction to the User Interface*, 2024. Official MadMapper Guide.

[Gar24b] GarageCube & 1024 Architecture. *Modules Guide*, 2024. Official MadMapper Guide.

[Gar24c] GarageCube & 1024 Architecture. *Realtime Reactive Visuals*, 2024. Official MadMapper Guide.

[Gar24d] GarageCube & 1024 Architecture. *Scenes and Cues Guide*, 2024. Official MadMapper Guide.

[GBea24] Estelle Guerry, Élodie Bécheras, y et al. The Use of LED Technology in Stage Lighting. A Literature Review. En *2024 IEEE Sustainable Smart Lighting World Conference Expo (LS24)*, páginas 1–4, 2024.

[GW20] Rafael C. Gonzalez y Richard E. Woods. *Digital Image Processing*. Pearson, edición 4th, 2020.

[GWea02] S. Guthe, M. Wand, y et al. Interactive rendering of large volume data sets. En *IEEE Visualization, 2002. VIS 2002.*, 2002.

[HAM02] Eric Haines y Tomas Akenine-Möller. *Real-Time Rendering*. CRC Press LLC, Boca Raton, Florida, edición 1st, 2002.

[HKea14] Juho Hamari, Jonna Koivisto, y et al. Does Gamification Work? – A Literature Review of Empirical Studies on Gamification. En *2014 47th Hawaii International Conference on System Sciences*, 2014.

[HLea15] Shan He, Sizhao Li, y et al. Uncertainty Analysis of Race Conditions in Real-Time Systems. En *2015 IEEE International Conference on Software Quality, Reliability and Security*, 2015.

[iSk24] iSkyDance Lighting LTD. *SPI Controllers – Installation Guide*, 2024.

[JMea99] A. K. Jain, M. N. Murty, y et al. Data clustering: a review. *ACM Computing Surveys*, 31, 1999.

[KDB16] F. Khodadadi, A. V. Dastjerdi, y R. Buyya. Internet of Things: Principles and Paradigms. En R. Buyya y A. V. Dastjerdi, editors, *Internet of Things: Principles and Paradigms*. Morgan Kaufmann, 2016.

[KSea07] Michael R. Krames, Oleg B. Shchekin, y et al. Status and Future of High-Power Light-Emitting Diodes for Solid-State Lighting. *Journal of Display Technology*, 3(2):160–175, 2007.

[LED23] LED Lighting Hut. *K-8000C RGB DMX LED Controller Manual*, 2023. url: `https://www.ledlightinghut.com/files/K-8000C.PDF`.

[LG07] Hugo Ledoux y Christopher M. Gold. The 3D Voronoi Diagram: A Tool for the Modelling of Geoscientific Datasets. En *GéoCongrès*, Québec, Canada, 2007.

[Li24] Yiyuan Li. An Exploration of Touchdesigner's Applications in the Digital Innovation of Ink Art. EAI, 2024.

[LO11] Phillip A. Laplante y Seppo J. Ovaska. *Real-Time Systems Design and Analysis: Tools for the Practitioner*. Wiley, United Kingdom, edición 4th, 2011.

[LRea20] Yanhong Li, Beat Rossmy, y et al. Tangible Interaction with Light: A Review. *Multimodal Technologies and Interaction*, 4:72, 2020.

[Maj04] A. Majumder. Camera based evaluation of photometric compensation methods on multi-projector displays. En *2004 International Conference on Image Processing, 2004. ICIP '04.*, volume 5, 2004.

[Mar17] R.C. Martin. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Robert C. Martin Series. Pearson Education, 2017.

[Mic03] Microchip Technology Inc. Getting Started: SPI – Overview and Use of the PICmicro Serial Peripheral Interface, 2003. url: `https://ww1.microchip.com/downloads/en/DeviceDoc/spi.pdf`.

[MSea19] Abhimitra Meka, Mohammad Shafiei, y et al. Real-Time Global Illumination Decomposition of Videos. *arXiv preprint arXiv:1908.01961*, 2019.

[Nie94] Jakob Nielsen. Ten Usability Heuristics for User Interface Design, 1994.

[oL23] University of Leicester. QLC+ Raspberry Pi Jessie Guide. `https://www.studocu.com/en-gb/document/university-of-leicester/introducito n-to-computing/qlc-raspberry-pi-jessie-guide/75067066`, 2023. Used in the course "Introduction to Computing", MA2252.

[OOea13] Ayodeji Oluwatope, Abiodun Odedoyin, y et al. Buffer Occupancy of Double-Buffer Traffic Shaper in Real-Time Multimedia Applications across Slow-Speed Links. *Communications and Network*, 05, 2013.

[PHMea22] Dewi Anggraini Puspa Hapsari, Sarifuddin Madenda, y et al. A Novel Approach to Video Compression using Region of Interest (ROI) Method on Video Surveillance Systems. *International Journal of Advanced Computer Science and Applications*, 13(6):126–132, 2022.

[Pil17] Anand Balachandran Pillai. *Software Architecture with Python: Design and Architect Highly Scalable, Robust, Clean, and High Performance Applications in Python*. Packt Publishing, Birmingham, UK, 2017.

[PM17] Kaushick Parui y Arun Emil Minj. Image Interpolation techniques in digital image processing. *ResearchGate*, 2017.

[PP93] Nikhil R. Pal y Sankar K. Pal. A review on image segmentation techniques. *Pattern Recognition*, 26(9):1277–1294, 1993.

[RD00] Richard Ryan y Edward Deci. Self-Determination Theory and the Facilitation of Intrinsic Motivation, Social Development, and Well-Being. *The American psychologist*, 55, 2000.

[Rou07] Henry Joseph Round. A note on carborundum. *Electrical World*, 49:309, 1907.

[SBea00] Margaret Suozzo, Nils Borg, y et al. LED Traffic Lights: Signaling a Global Transformation. En *Proceedings of the 2000 ACEEE Summer Study on Energy Efficiency in Buildings*. American Council for an Energy-Efficient Economy, 2000.

[SBea17] Neus Sabater, Guillaume Boisson, y et al. Dataset and Pipeline for Multi-View Light-Field Video. En *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, July 2017.

[SCea23] Jirawat Sookkaew, Nakarin Chaikaew, y et al. Interactive Light Art: The Illumination of Art and Technology Merging. *International Journal of Engineering Trends and Technology (IJETT)*, 71(12):308–323, 2023.

[Sch03] E. Fred Schubert. History of light-emitting diodes. En *Light-Emitting Diodes*. Cambridge University Press, Cambridge, 2003.

[Som16] Ian Sommerville. *Software Engineering*. Pearson Education Limited, edición 10th, 2016.

[SS20] Ken Schwaber y Jeff Sutherland. The Scrum Guide – The Definitive Guide to Scrum: The Rules of the Game, 2020.

[Su 08] Su Holmes. *The Quiz Show*. Edinburgh University Press, 2008.

[SVea20] Pallavi S., Priyanka V., y et al. Design and Verification of Serial Peripheral Interface (SPI) Protocol. *International Journal of Recent Technology and Engineering (IJRTE)*, 8(6):793–797, 2020.

[Tan09] Andrew S. Tanenbaum. *Sistemas operativos modernos*. Pearson, edición 3rd, 2009.

[Tek14] Tektronix, Inc. *A Guide to MPEG Fundamentals and Protocol Analysis*, 2014. url: `https://download.tek.com/document/25W-11418-10.pdf`.

[Tex12] Texas Instruments. *Serial Peripheral Interface (SPI) for KeyStone Devices – User's Guide*, 2012. url: `https://www.ti.com/lit/ug/sprugp2a/sprugp2a.pdf`.

[The25] The NumPy Community. *NumPy User Guide*. NumPy Developers, edición 2.2.0, January 2025. url: https://numpy.org/doc/2.2/numpy-user.pdf.

[Tsa05] Jeff Y. Tsao. Solid-stage lighting: Lamps, chips, and materials for tomorrow. *IEEE Circuits and Devices Magazine*, 21(3):27–37, 2005.

[TVS06] Andrew S. Tanenbaum y Maarten Van Steen. *Distributed Systems: Principles and Paradigms*. Pearson Prentice Hall, edición 2nd, 2006.

[VNea20] Abner Velazco, Magnus Nord, y et al. Evaluation of different rectangular scan strategies for STEM imaging, 2020.

[WMea21] Liping Wang, Jianshe Ma, y et al. High-Resolution Pixel LED Headlamps: Functional Requirement Analysis and Research Progress. *Applied Sciences*, 11(8), 2021. url: https://www.mdpi.com/2076-3417/11/8/3368.

[Wor13] World Semi. *WS2812 Intelligent Control LED Integrated Light Source*, 2013.

[XSea18] Neal N. Xiong, Yang Shen, y et al. Color sensors and their applications based on real-time color image segmentation for cyber physical systems. *EURASIP Journal on Image and Video Processing*, 2018(1):23, 2018.

[XTea22] Xiaoyu Xiang, Yapeng Tian, y et al. Learning Spatio-Temporal Downsampling for Effective Video Upscaling. En *European Conference on Computer Vision (ECCV)*, 2022. url: https://www.ecva.net/papers/eccv_2022/papers_ECCV/papers/136780159.pdf.

[Xu24] Huili Xu. Study of Lighting Design for Interior Spaces. *Journal of Education and Educational Research*, 8(1):97–101, 2024.

[Yu13] Jyh-Cheng Yu. Design of LED Edge-Lit Light Bar for Automotive Taillight Applications. En *Proceedings of SPIE - LED-based Illumination Systems*, volume 8835. SPIE, 2013.

[Zou20] Xuedan Zou. PIXEL: Interactive Light System Design Based On Simple Gesture Recognition, 2020.

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`