UAB
Universitat Autònoma de Barcelona

Dipòsit digital de documents de la UAB

This is the **published version** of the bachelor thesis:

Ali Ali, Muniba; Cortes Comellas, Oriol, tut. Xarxa social efímera basada en esdeveniments. 2025. (Enginyeria Informàtica)

This version is available at https://ddd.uab.cat/record/317579

under the terms of the BY-NC-ND license

Ephemeral Event-Based Social Network: Secure and Contextual User Interaction Platform

Muniba Liaqat Ali Ali

June 30, 2025

Resum– Aquest Treball de Fi de Grau presenta el desenvolupament d'una xarxa social efímera centrada en la interacció durant esdeveniments. La plataforma permet que els usuaris es connectin i comuniquin únicament mentre dura l'esdeveniment, evitant la sobrecàrrega de continguts i afavorint la privacitat. Està desenvolupada amb el conjunt tecnològic MERN (MongoDB, Express.js, React, Node.js) i incorpora autenticació segura, rols diferenciats, creació i cerca d'esdeveniments, i un sistema de xat en temps real. Els esdeveniments poden ser públics o privats; només en aquests darrers es requereix escanejar un codi QR físicament per accedir al xat. El projecte segueix una metodologia àgil, amb desenvolupament iteratiu i testatge continu. Els resultats mostren una plataforma funcional i escalable, amb aplicacions reals en el networking i la comunicació efímera en entorns segurs.

Paraules clau– Xarxa efímera, xat en temps real, sistema basat en esdeveniments, MERN stack, autenticació QR

Abstract– This Final Degree Project presents the development of an ephemeral social network for event-based interactions. The platform allows users to connect only during active events, avoiding content overload and enhancing privacy. Built with the MERN stack (MongoDB, Express.js, React, Node.js), it includes secure authentication, user roles, event creation and discovery, and real-time chat. Events can be public or private; private chats require scanning a QR code at the venue to verify attendance. The project adopts an agile methodology with iterative development and testing. Results confirm a robust and scalable system with applications in networking, secure communication, and post-event analysis.

Keywords– Ephemeral network, real-time chat, event-based system, MERN stack, QR authentication

1 Introduction and Project Context

In today's hyperconnected world, digital interactions are constant and often overwhelming. While traditional social networks provide spaces for continuous communication, they frequently lack contextual relevance, especially in temporary situations like live events. This Final Degree Project introduces an innovative ephemeral social network specifically designed for real-time interaction during scheduled events such as conferences, concerts, or festivals.

The platform is structured around two user roles: **Administrators** and **Users**. Administrators are responsible for creating, configuring, and managing events, while Users can browse, join, and participate in any public event listed on the platform. This clear separation of roles ensures both organizational control and seamless participation.

A core feature of the system is a **real-time chat** linked to each event. This chat becomes active only during the event's scheduled time, allowing attendees to interact in a focused and contextual way. Once the event concludes, the chat automatically enters a read-only archived state, where users can still review the conversation, but cannot send new messages. This ensures relevance and prevents post-event

• Contact e-mail: 1635042@uab.cat
• Specialization: Information Technologies
• Supervised by: Oriol Cortés Comellas (Department of Computer Science and Artificial Intelligence)
• Academic year: 2024/25

noise.

To enhance security and authenticity, **private events** require attendees to scan a **QR code** displayed at the physical location in order to access the chat. [17] This mechanism guarantees that only verified participants can join the discussion.

Another distinctive feature is the **event review system**. After an event ends, users who attended can leave feedback, helping organizers improve future experiences and enabling other users to discover high-quality events based on community insights.

Overall, the project addresses the need for focused, temporary digital spaces where interactions are meaningful and time-bound, combining real-time communication, access control, and community-driven feedback. This work lays the foundation for a scalable and versatile event-based platform with strong potential in both social and professional contexts.

2 Objectives

2.1 General Objective

To design and implement an ephemeral social networking platform that enables meaningful, time-bound interaction between event participants, combining secure real-time communication, access control, and post-event feedback mechanisms.

2.2 Specific Objectives

- **SO1**: Develop a **role-based authentication system** distinguishing administrators and users, with secure login and session control. [23]
- **SO2**: Allow administrators to **create, configure, edit and delete events** through an intuitive management interface.
- **SO3**: Enable users to **browse, search and join public events**, and view detailed event information.
- **SO4**: Implement a **favourites system** that allows both administrators and users to bookmark events and monitor their activity.
- **SO5**: Implement a **real-time event chat** that is only available during the event and transitions to an **archived read-only state** afterward.
- **SO6**: Integrate **QR code-based access control** for private events, ensuring only verified physical attendees can join the event's chat.
- **SO7**: Develop a **notification system** that enables administrators to send real-time updates or announcements to users interested in their events.
- **SO8**: Allow users to **submit event reviews** after attending, including ratings and comments to improve organizer accountability and guide other participants.
- **SO9**: Apply an **agile development methodology**, structuring the project into iterative sprints with continuous testing and refinement.

3 State of the Art

In recent years, a wide range of digital platforms have emerged to facilitate communication and interaction during events. Applications such as **WhatsApp**, **Telegram**, and **Discord** are often used to create temporary group chats, while platforms like **Facebook Events** or **Meetup** help organize and promote social gatherings. However, these tools are not designed specifically for ephemeral interaction. They retain data permanently, lack contextual boundaries, and often result in fragmented or uncontrolled communication spaces.

Despite the growing demand for more **context-aware digital spaces**, few platforms have focused on creating truly **temporary and event-specific communication environments**. Most solutions rely on generic chat functionalities or require extensive manual setup, which does not scale well for event organizers or provide sufficient access control. Additionally, they rarely distinguish between types of participants or integrate post-event feedback mechanisms in a meaningful way.

What sets this project apart is its **event-centric and time-limited architecture**, combined with a well-defined **role system** (admins vs. users), **QR-based private access**, **real-time chat activation**, and a **post-event review system**, all of which are tightly integrated into a single, user-friendly platform. Unlike traditional tools, this solution **automatically deactivates chats when events conclude**, preserving the ephemeral nature of interactions while still allowing participants to access archived conversations in read-only mode.

Furthermore, features such as **personalized favourites**, and **notification broadcasting for event updates** place this platform closer to the needs of modern organizers and attendees alike, enabling a smooth and secure experience from discovery to post-event engagement.

To the best of my knowledge, no existing platform combines this level of **temporal interaction design**, **security**, **feedback collection**, and **user segmentation** in one cohesive system. This project fills that gap by offering an innovative approach to ephemeral digital networking tailored specifically for real-world events.

4 Methodology

This project followed an **agile development methodology**, structured in short sprints to allow for continuous progress tracking, flexibility in responding to challenges, and iterative improvements. The agile approach was key to managing complexity and adapting the scope as the project evolved.

Frontend and backend development were carried out in **parallel**, following a modular architecture to ensure separation of concerns and ease of integration. The system was implemented using the **MERN stack** (MongoDB, Express.js, React, Node.js), which enabled rapid development with reusable components and scalable APIs.

For task and version management, **GitHub Projects** was used to organize sprints, assign priorities, and track completion. All code was managed using **GitHub** for version control, with commits aligned to development milestones and delivery checkpoints. [11] **Postman** was used to test

and validate RESTful endpoints, [14] [3] while **Figma** supported early UI/UX prototyping and visual planning.

Manual testing, log-based debugging, and validation of user interactions were carried out continuously to ensure functionality, usability, and data integrity. This combination of agile development and modern tooling resulted in a highly flexible and well-structured workflow.

5 PLANNING

The development of this project was structured into four main phases, each aligned with the core functionality and evolution of the application (**see Fig. 1**). These phases were organized according to an agile sprint methodology and focused on delivering incremental progress throughout the semester. **A more detailed breakdown of the task board and timeline can be found in Appendix A.**

5.1 Phase 1: Requirements and Initial Setup

The project began with the definition of functional requirements and the design of wireframes using Figma to establish a clear visual structure. The backend was initialized using Express.js [5] [6] and MongoDB, [19] [18] and the first modules related to user registration, login, and role-based authentication were implemented. This phase also included the basic profile system and the foundation for event creation.

5.2 Phase 2: Core Functionality

In this stage, key modules were implemented to give shape to the platform's primary functionalities. These included event creation, editing, and deletion by administrators, public event discovery by users, and the implementation of the favourites system for both roles.

5.3 Phase 3: Real-Time Interaction and Access Control

The third phase focused on enabling ephemeral interaction during events. A real-time chat system was developed using WebSockets, configured to activate only while the event is ongoing and transition to a read-only state once the event concludes. QR-based authentication was introduced to secure access to private event chats, restricting it to verified attendees.

5.4 Phase 4: Notifications, Reviews and Final Adjustments

The last phase added features aimed at improving user engagement and event monitoring. A notification system was implemented to allow administrators to communicate with users who showed interest in their events. A review module was also added to collect post-event feedback. The project concluded with interface polishing, testing, and the drafting of the final report.

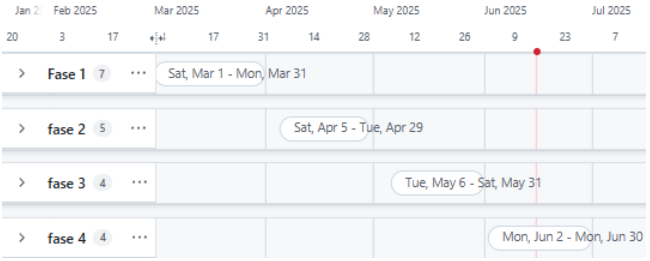


Fig. 1: *Visual representation of the planning phases and their timeline using GitHub Projects.*

6 REQUIREMENTS

A solid understanding of the system requirements is essential to guide development and ensure that the final product meets both technical expectations and user needs. This section details the **functional**, **non-functional**, and **technical requirements** of the ephemeral event-based social network, as defined during the initial planning and refined throughout the project's iterations.

Each requirement is assigned a **priority** level: High, Medium, or Low, to reflect its criticality for the platform's core functionality and user experience.

6.1 Functional Requirements

These requirements define what the system must do from the perspective of user interaction and behaviour.

ID	Description	Priority
FR1	Users can register, log in, and edit their profile	High
FR2	Users can browse and join public events	High
FR3	Administrators can create, edit, and delete events	High
FR4	Events can be public or private	High
FR5	Users and admins can mark events as favourites	Medium
FR6	Private event access requires QR code scanning	High
FR7	Real-time chat is enabled only during the event	High
FR8	Chat becomes read-only after the event ends	Medium
FR9	Admins can send notifications to users interested in their events	Medium
FR10	Users can leave reviews (rating and comments) after attending an event	Medium

TABLE 1: *Functional requirements of the platform and their priority.*

6.2 Non-Functional Requirements

These specify how the system should behave, focusing on performance, security, usability, and reliability.

ID	Description	Priority
NFR1	The application is responsive and mobile-friendly	High
NFR2	Protected routes and user data follow security best practices (e.g. encryption, access control)	High
NFR3	The system performs reliably with moderate concurrent user load	Medium
NFR4	Real-time communication has minimal latency	High
NFR5	The user interface is intuitive and consistent with the ephemeral nature of the platform	Medium
NFR6	Only authenticated users can access protected views	High

TABLE 2: *Non-functional requirements of the platform and their priority.*

6.3 Technical Requirements

These define the technology stack, tools, and architectural decisions used to implement the platform.

ID	Description	Priority
TR1	Frontend developed with React and modular components	High
TR2	Backend built with Node.js and Express, exposing RESTful APIs	High
TR3	MongoDB used as the database, accessed through Mongoose	High
TR4	Real-time chat implemented with Socket.IO	High
TR5	QR scanning integrated via html5-qrcode	Medium
TR6	Authentication handled with JWT and bcrypt password hashing	High
TR7	Development workflow managed with GitHub Projects	Medium
TR8	UI prototyped and styled with Figma and CSS Modules	Medium

TABLE 3: *Technical requirements of the platform and their priority.*

7 Use Cases

Understanding and documenting key use cases is essential to ensure that the system meets user expectations and behaves correctly in real scenarios. The following table outlines the main interactions between users and the platform, detailing the purpose and responsible roles. These use cases guided both the feature design and the implementation of core functionalities.

ID	Title	Description	Role
UC1	User Registration	A user registers by filling in personal data and credentials to create an account.	User
UC2	Event Creation	An administrator creates and configures a new event, setting title, date, privacy, and description.	Admin
UC3	Event Discovery	A user searches or browses public events and views details before joining.	User
UC4	Join Event	A user joins a selected public event and adds it to their participation list.	User
UC5	Event Chat	During the event, users can send and receive real-time messages in the event chat.	User/Admin
UC6	QR Verification	A user accesses the chat of a private event by scanning a physical QR code.	User
UC7	Notifications	An administrator sends updates or messages to users interested in a specific event.	Admin
UC8	Leave Review	After the event ends, a user leaves a rating and comment to review the event.	User
UC9	Mark Favourite	A user or admin marks an event as favourite to monitor its activity or return later.	User/Admin

TABLE 4: *Main use cases and interactions within the platform.*

8 Development

The development of the platform was carried out entirely in a local environment, ensuring a full control over both frontend and backend functionalities. The architecture followed a modular and component-based design to facilitate maintainability, scalability, and clarity during implementation.

The stack selected for this project was the **MERN stack** (MongoDB, Express.js, React, Node.js), which enabled a fast and modern web development process. This section details the technologies, tools, and structure used throughout the development process, divided between the backend and frontend subsystems.

8.1 Backend

The backend was implemented using **Node.js** [20] with the **Express.js** framework, responsible for managing the application's logic and RESTful API endpoints. The server exposes different route groups, each corresponding to a specific feature such as user authentication, events, chat messages, QR access, notifications, and reviews.

8.1.1 Architecture and Folder Structure

The backend follows a clear separation of concerns, organized into four main directories:

- `controllers/`: Contains the logic for handling each route (e.g., `loginController.js`, `joinQRController.js`, `reviewController.js`).
- `models/`: Includes the Mongoose schemas for MongoDB collections such as `users`, `events`, `messages`, and `reviews`.
- `routes/`: Groups the routing logic for each resource (e.g., `userRoutes.js`, `eventRoutes.js`, `notificationRoutes.js`).
- `middleware/`: Contains reusable logic like authentication validation.

8.1.2 Key Technologies and Tools

- **MongoDB** was used as the primary database, managed via **MongoDB Compass**, a graphical interface to view and query collections.
- **Mongoose** enabled schema-based modeling and validation of documents, as well as easy population of references across collections.
- **dotenv** was used for environment variable management and secure separation of credentials.
- **cors** was configured to handle cross-origin resource sharing between the frontend and backend.
- **socket.io** powered the real-time chat system, allowing events to create ephemeral chat rooms tied to the event lifecycle.
- **jsonwebtoken** and **bcrypt** were used for user authentication, [4] [16] [28] token validation, and password hashing. [1] [2]

8.2 Frontend

The frontend was developed using **React**, [22] with a modular folder structure and component-based design for reusability and clarity. Each feature of the application is encapsulated in its own folder under `src/components/`.

8.2.1 Component Structure

The main functional areas are organized into self-contained components such as:

- **Login**, **Signup**, **EditProfile**: User authentication and profile management
- **CreateEvent**, **EventDetails** and **SearchEvents**: Event creation, discovery, and management
- **ChatEvent**: Real-time chat linked to each event
- **QRCodeScanner**: Private QR-based access validation using **html5-qrcode**
- **EventNotifications**, **UserNotifications**: Notification interface for admins and users
- **EventReviews**: Submission of reviews after event conclusion
- **Inbox**, **Main**, **Navbar**: Navigation and user experience flow

8.2.2 Styling and User Interface

All styling was implemented using **CSS Modules**, allowing for scoped styles and better maintenance. The interface was designed with usability and visual clarity in mind, following the ephemeral nature of the platform.

8.2.3 Real-Time Features

The platform integrates real-time chat through **socket.io-client**, [24] which connects each user to their event-specific room. Additionally, QR scanning is implemented using the **html5-qrcode** library, enabling physical verification for private event participation.

8.3 Execution Environment

The entire application was developed and tested locally. The backend server runs on a local instance using Express, while the frontend runs via React's development server. Communication between both layers is handled via HTTP [15] and WebSocket connections on localhost.

8.3.1 Cloud Deployment

After completing development and testing locally, the platform was deployed to the cloud to make it publicly accessible and simulate a real-world production environment.

The final version of the application is available online at: https://tfg-theta-two.vercel.app/

To deploy the system, the following services were used:

- **MongoDB Atlas** – Hosted NoSQL database used to store users, events, messages, and reviews.
- **Render** – Used to deploy and host the Express-based backend server.
- **Vercel** – Used to host the React-based frontend with continuous integration and easy deployment.

These services allowed for smooth deployment, secure data handling, and responsive access from any device.

9 RESULTS

The final implementation of the platform, named **Ephegram**, delivers a complete and functional ephemeral social network tailored for event-based interaction. This section provides a comprehensive overview of the achieved results, comparing the initial design wireframes with the implemented system, and outlining the testing and validation procedures used to ensure platform reliability, security, and usability.

9.1 From Wireframes to Functional System

The project began with the creation of wireframes in **Figma**, which served as visual references to define the platform's navigation, user flow, and main components. [7] [8] While the final web design diverged significantly from these initial mockups, they provided essential guidance during the planning and early development stages.

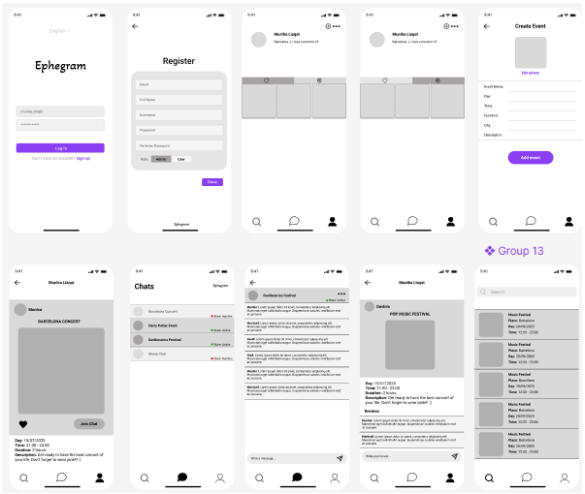


Fig. 2: *Initial wireframes in Figma.*

These wireframes helped define structural elements such as the login screen, event feed, chat view, and profile layout. However, during development, the visual design and component distribution evolved significantly to improve clarity, usability, and mobile responsiveness. The final application adopts a modern look with role-based navigation and fully functional interaction layers.

Screenshots of the final implementation are available in the Appendix (Annex B for Admin view, Annex C for User view).

9.2 Implemented Features by Role

The platform is structured around two main roles: **Admin** and **User**. The following subsections describe the functionalities developed for each role. Visual references for each section are included in the annexes.

Admin Functionality

- **Authentication:** Secure login using hashed passwords and JWT tokens. The system validates the admin role upon login and redirects accordingly to the admin dashboard.
- **Event Management:** Admins can create, edit, and delete events. They are able to upload images, define whether the event is public or private, and set key parameters such as title, description, location, date, and time.
- **Chat Oversight:** Admins have full visibility over the real-time chat of every event they have created. This allows them to monitor conversations, encourage participation, and ensure proper conduct during the event.
- **Notification Management:** Admins can send customized notifications to users who have shown interest in their events—either by joining them or marking them as favourites. These notifications help communicate updates, reminders, or last-minute changes. Admins can also review and delete previously sent notifications.
- **Review Access:** After an event concludes, admins can view the feedback and ratings left by users. This allows them to gather insights and improve future event planning.
- **Event Favourites:** Admins can mark any public event as a favourite. This does not grant them access to user-specific content or chats, but allows them to monitor events of interest, including those created by other admins (e.g., for benchmarking or inspiration). However, admins cannot join events as participants.

User Functionality

- **Authentication:** Users can register and log in, choosing their role during the process.
- **Dashboard:** Users see upcoming events, favourite events, and receive real-time notifications from events they follow or joined.
- **Event Discovery:** Public events can be browsed, searched, and filtered. Users can join events or mark them as favourites.
- **Event Details:** Full event information is displayed in a dedicated view, including creator, date, location and description.
- **Real-Time Chat:** During an active event, users can access a real-time chat, powered by Socket.IO. When the event ends, the chat becomes read-only.[25] [26]
- **QR Access for Private Events:** Private events require users to scan a QR code at the venue using the html5-qrcode library to access the chat. [13]
- **Review Submission:** After attending an event, users can submit a rating and comment, which they can later delete if desired.

All screenshots of Admin views are included in **Annex B**, and User views in **Annex C**.

9.3 Testing and Validation

Backend Testing (Postman)

All backend API endpoints were tested using **Postman**. [21] [27] The testing covered:

- Authentication and token validation
- CRUD operations for events
- Message sending and retrieval in chat rooms
- QR-based access validation
- Notification dispatch and retrieval

The system successfully handled expected responses and edge cases, returning appropriate HTTP status codes.

Frontend Testing

Frontend behavior was tested manually with:

- Real-time alerts (success/error) using visual UI notifications
- Console logs to trace state transitions
- Role-based navigation guards
- Mobile responsiveness on different screen sizes

The application showed stable behavior, with smooth transitions between screens and proper access control for protected views.

9.4 Summary of Achievements

The development and implementation of the platform culminated in the successful fulfilment of all planned objectives, delivering a robust, scalable, and context-aware social network tailored to ephemeral interaction. Below is a breakdown of the main milestones achieved and the value they bring to the project:

- **Complete Implementation of Functional Requirements:** All features outlined in the functional specifications (authentication, event creation, real-time chat, QR access control, notifications, reviews, etc.) were developed and fully integrated, ensuring a complete user journey from onboarding to post-event engagement.
- **Role-Based Separation with Tailored Experiences:** The system provides two clearly differentiated flows for **Administrators** and **Users**, each with their own interface and functionality. Administrators can manage all aspects of the event lifecycle, while users can explore, participate in and evaluate events with ease.
- **Real-Time Communication and Ephemeral Chatrooms:** Socket.IO-based chat was successfully deployed with lifecycle control, active only during the event and archived afterward. This ensures that communication remains relevant and context-bound, embodying the core principle of ephemerality.
- **Secure and Contextual Access Control:** Private events implement QR-code based access via physical validation. This guarantees that only verified attendees can join restricted event chats, reinforcing authenticity and privacy.
- **Full Notification Pipeline:** A flexible notification system was implemented. Administrators can issue announcements and updates to all users who have expressed interest in an event, while users can review these notifications in a dedicated section.
- **Review System for Event Quality Feedback:** Users can leave a rating and textual feedback for attended events. These reviews are stored, displayed, and can be managed by the user. This encourages accountability and future improvement by event organizers.
- **Frontend and Backend Successfully Decoupled and Integrated:** The frontend and backend were developed in parallel with a clean API contract. The MERN stack proved highly efficient, allowing seamless communication between client and server, with clearly defined REST endpoints.
- **Testing and Validation Across Layers:** Postman was used to rigorously test backend endpoints under multiple scenarios, including edge cases. On the frontend, functionality was verified through alert messages, real-time logs, and role-based view testing. Access control and chat behavior were especially scrutinized for correctness.
- **Responsive, Modular and Maintainable Codebase:** The platform is responsive across devices and follows modular best practices. CSS Modules were used for clean styling, and all frontend components were clearly scoped. The backend follows a model-controller-route separation that facilitates future scalability and extension.
- **Visual and Interaction Design:** While the initial Figma wireframes served as conceptual guidance, the final application evolved into a more refined and dynamic interface. User feedback loops and real use-case constraints guided this evolution, resulting in an aesthetically clean and functionally rich platform. [9] [10]

In conclusion, the platform meets, and in many aspects exceeds, the original expectations. The system not only demonstrates technical feasibility and user-centric design, but also provides a solid foundation for future deployment or academic extension.

9.5 Appendix Reference

Annex B: Admin interface screenshots

Annex C: User interface screenshots

10 Economic Evaluation

The economic evaluation of this project is based on an estimation of working hours, software and hardware requirements, and potential licensing costs. As this Final Degree Project (TFG) corresponds to 12 ECTS credits, it implies an approximate dedication of 300 hours.

10.1 Cost of Roles and Working Hours

The table below estimates the theoretical cost of the project using junior-level hourly rates for each role involved in software development. The figures are illustrative and do not represent real payments but help quantify the professional workload.

Role	Hours	€/h	Cost
UI/UX Designer	20h	22€/h	440€
Frontend Developer	110h	25€/h	2750€
Backend Developer	80h	26€/h	2080€
Project Manager	40h	28€/h	1120€
Tester and QA	50h	24€/h	1200€
Total	300h	—	**7590€**

TABLE 5: *Estimated cost of roles and working hours.*

10.2 Software and Licensing Costs

Although the project was mainly developed using free and academic tools, its final deployment required the use of additional cloud services. Below is a list of the main tools involved:

- **Visual Studio Code** – Main code editor for frontend and backend development.
- **GitHub** – Version control and remote repository for collaboration.
- **GitHub Projects** – Used for agile task tracking and progress monitoring.
- **Postman** – Used to test and validate backend API endpoints under multiple scenarios.
- **MongoDB Compass** – GUI client to visualize and manage MongoDB collections.
- **Figma** – Used during the design phase to create wireframes and define UI structure.

For deployment and testing purposes, the following platforms were also used:

- **MongoDB Atlas** – Cloud-hosted NoSQL database used in production.
- **Render** – Used to deploy backend services.
- **Vercel** – Used to host the frontend application.

Although no actual money was spent, the estimated commercial value of the cloud resources used (storage, hosting, and bandwidth) during the final stages of the project is approximately **120€**.

10.3 Hardware Costs

The project was developed and tested using:

- A personal laptop with sufficient specs for fullstack development – estimated value: 1000€
- A mobile device (iOS) for responsive testing – estimated value: 800€

The total estimated cost for hardware usage is **1800€**.

10.4 Total Project Cost

Taking all relevant factors into account:

- Human resources: 7590€
- Software and licensing: 120€
- Hardware: 1800€

Total estimated cost: 9510€

11 Conclusions

The development of **Ephegram** has successfully demonstrated the feasibility and impact of an ephemeral, event-based social network. From the initial idea and wireframes to the final deployed system, the project has evolved into a complete and coherent platform that integrates technical robustness, user-centric design, and real-time functionality.

11.1 Fulfilment of Objectives

All the goals set at the beginning of the project were met, both at the functional and technical levels. The system offers:

- A clear role-based architecture (Admin vs. User) with tailored functionality for each profile.
- A responsive and intuitive frontend built with modern web technologies.
- A secure backend with full API coverage and real-time communication through Socket.IO.
- Context-aware interaction flows, such as QR-protected chats and time-bound messaging.
- Integration of notifications and post-event reviews, enriching user experience and platform quality.

11.2 Key Learnings

This project has provided an exceptional opportunity to apply knowledge from multiple disciplines, including:

- **Full-stack development:** Handling both frontend and backend logic, including API design, database modelling, authentication, and UI rendering.
- **Agile methodologies:** Managing tasks with GitHub Projects and planning iterations based on user roles and feedback loops. [12]

- **Security and privacy:** Implementing JWT, bcrypt, role-based access control, and ephemeral communication principles.
- **System design:** Structuring scalable components, ensuring modularity, and managing dependencies effectively.

11.3 Real-World Value

Ephegram goes beyond being a student prototype. Its design makes it suitable for real-life deployment in contexts such as:

- Temporary events or conferences where attendees interact only during the live session.
- Private meetups or workshops that require invitation control (QR validation).
- Social networking among people with shared interests that are limited in time or space.

The concept of ephemeral interaction, combined with real-time technologies and solid user control, proves to be not only technically viable but socially relevant.

11.4 Future Improvements

While the current version is fully functional, there are several ideas that could extend the platform in the future:

- Implementing image sharing and media in the chat.
- Adding push notifications via mobile (Firebase Cloud Messaging).
- Enabling multi-language support.
- Providing user analytics for admins to track event engagement.

11.5 Final Reflection

Working on this project has been a challenging yet rewarding experience. It has helped to consolidate not only technical skills but also critical thinking, time management, and problem-solving capabilities. Delivering a real, working product that mirrors real-world requirements is the best possible outcome for a Final Degree Project.

Ephegram reflects the essence of what a final project should achieve: technical depth, creativity, and real-world applicability.

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to all the people who have supported me throughout this journey and contributed, in one way or another, to the completion of this project.

First of all, I would like to thank my supervisor for proposing such a creative and inspiring idea. The concept provided not only technical depth but also personal motivation, making this project a meaningful and fulfilling experience.

To my parents for everything. As immigrants without a privileged academic or professional background, they have always done their absolute best to provide me with a better life and ensure that I never lacked access to education. Their efforts, sacrifices, and unconditional love have been the foundation of every step I have taken. This achievement is as much theirs as mine.

To my professors throughout the degree, thank you for sharing your knowledge, your dedication to teaching, and for guiding us along the way. Each class, comment, and challenge has helped shape the professional I am becoming.

To my classmates and friends, thank you for your support, your encouragement, and for making this intense academic journey more bearable, even in the toughest moments.

And finally, to myself, for never giving up despite all the obstacles, for pushing through every challenge, and for reaching this milestone in my journey to become a computer engineer. This project represents not only academic success, but also a personal victory built on resilience and determination.

As the Qur'an reminds us:

Fa-inna ma'al usri yusrā
"Indeed, with hardship comes ease."
(Qur'an 94:6)

These words have been a source of strength and hope during difficult times, and this achievement is proof that perseverance bears fruit.

I am confident this is not the final step, but rather the first of many in the journey toward the bright future I am determined to build.

REFERENCES

[1] NPM. "bcrypt," *npmjs.com*. [Online]. Available: https://www.npmjs.com/package/bcrypt [Accessed: 3-Apr-2025].

[2] GeeksforGeeks. "Password Encryption in Node.js using bcryptjs Module," *GeeksforGeeks.org*. [Online]. Available: https://www.geeksforgeeks.org/password-encryption-in-node-js-using-bcryptjs-module/ [Accessed: 3-Apr-2025].

[3] Dev.to. "How to validate Node + Express requests with Joi middleware," *dev.to*. [Online]. Available: https://dev.to/mattiamalonni/how-to-validate-node-express-requests-with-joi-middleware-b2c [Accessed: 4-Apr-2025].

[4] DigitalOcean. "Implementing JWT Authentication in Node.js," *DigitalOcean Community*. [Online]. Available: https://www.digitalocean.com/community/tutorials/nodejs-jwt-expressjs [Accessed: 2-Apr-2025].

[5] Express.js. "Express - Node.js web application framework," *expressjs.com*. [Online]. Available: https://expressjs.com/ [Accessed: 14-Mar-2025].

[6] Express.js. "Error handling," *Express.js Guide*. [Online]. Available: https://expressjs.com/en/guide/error-handling.html [Accessed: 5-Apr-2025].

[7] Figma. “Design system best practices,” *Figma Blog*. [Online]. Available: https://www.figma.com/best-practices/ [Accessed: 21-Mar-2025].

[8] Figma. “Create layout grids,” *Figma Help Center*. [Online]. Available: https://help.figma.com/hc/en-us/articles/360040450513-Create-layout-grids-with-grids-columns-and-rows [Accessed: 21-Mar-2025].

[9] Figma. “Getting started in Figma,” *Figma Help Center*. [Online]. Available: https://help.figma.com/hc/en-us/categories/360002051613-Get-started [Accessed: 20-Mar-2025].

[10] Figma. “Create prototypes,” *Figma Help Center*. [Online]. Available: https://help.figma.com/hc/en-us/articles/360040451373-Create-prototypes [Accessed: 20-Mar-2025].

[11] GitHub. “Help documentation,” *GitHub Docs*. [Online]. Available: https://docs.github.com/en [Accessed: 1-Mar-2025].

[12] GitHub. “About projects,” *GitHub Docs*. [Online]. Available: https://docs.github.com/en/issues/planning-and-tracking-with-projects/learning-about-projects/about-projects [Accessed: 1-Mar-2025].

[13] html5-qrcode. “html5-qrcode documentation.” GitHub: *mebjas/html5-qrcode*. [Online]. Available: https://github.com/mebjas/html5-qrcode [Accessed: 10-May-2025].

[14] Joi. “Object schema validation,” *joi.dev*. [Online]. Available: https://joi.dev/api/?v=17.9.1 [Accessed: 4-Apr-2025].

[15] MDN Web Docs. “HTTP Headers – Authentication and Security,” *developer.mozilla.org*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers [Accessed: 6-Apr-2025].

[16] MDN Web Docs. “Working with JSON Web Tokens (JWT),” *developer.mozilla.org*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Authentication [Accessed: 22-Mar-2025].

[17] MDN Web Docs. “MediaDevices.getUserMedia(): Web APIs,” *developer.mozilla.org*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices/getUserMedia [Accessed: 20-May-2025].

[18] MongoDB. “MongoDB Documentation,” *mongodb.com*. [Online]. Available: https://www.mongodb.com/docs/ [Accessed: 16-Mar-2025].

[19] Mongoose. “Getting started with Mongoose,” *mongoosejs.com*. [Online]. Available: https://mongoosejs.com/docs/index.html [Accessed: 16-Mar-2025].

[20] NPM. “express-session,” *npmjs.com*. [Online]. Available: https://www.npmjs.com/package/express-session [Accessed: 6-Apr-2025].

[21] Postman. “Postman Learning Center,” *postman.com*. [Online]. Available: https://learning.postman.com/ [Accessed: 5-Apr-2025].

[22] React. “Quick start – React,” *react.dev*. [Online]. Available: https://react.dev/learn [Accessed: 14-Mar-2025].

[23] Section.io. “Creating Secure Login System with Node.js,” *section.io*. [Online]. Available: https://www.section.io/engineering-education/secure-login-system-node-js/ [Accessed: 5-Apr-2025].

[24] Socket.IO. “Getting started: Socket.IO documentation,” *socket.io*. [Online]. Available: https://socket.io/docs/v4 [Accessed: 15-May-2025].

[25] Socket.IO. “Emit cheatsheet: How to use emit and on,” *socket.io*. [Online]. Available: https://socket.io/docs/v4/emit-cheatsheet/ [Accessed: 1-May-2025].

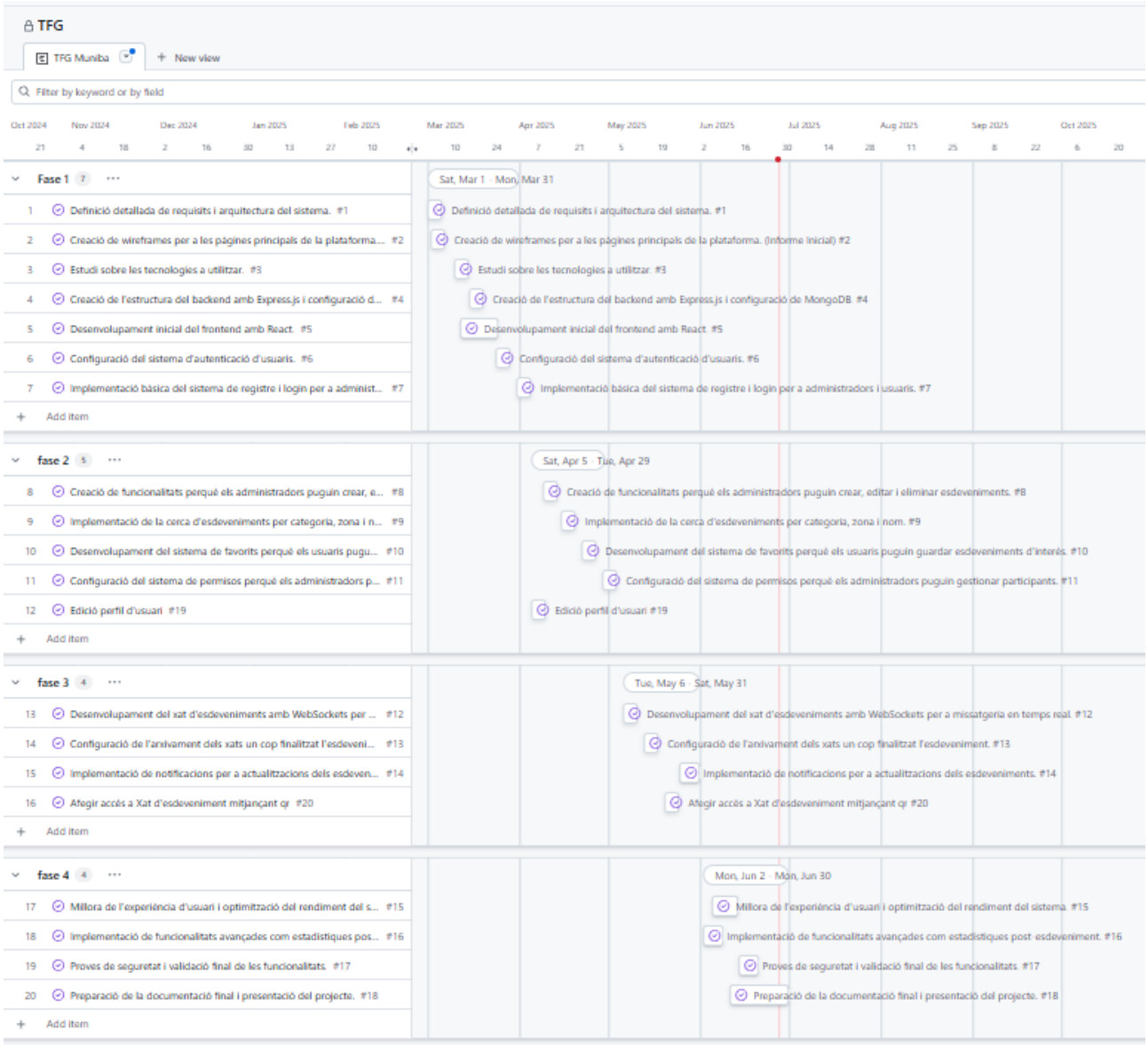
[26] Socket.IO. “Managing rooms: Rooms and namespaces,” *socket.io*. [Online]. Available: https://socket.io/docs/v4/rooms/ [Accessed: 12-May-2025].

[27] ToolsQA. “How to use Postman to test APIs,” *toolsqa.com*. [Online]. Available: https://www.toolsqa.com/postman/postman-tutorial/ [Accessed: 6-Apr-2025].

[28] JWT.io. “JSON Web Tokens Introduction,” *jwt.io*. [Online]. Available: https://jwt.io/introduction [Accessed: 2-Apr-2025].

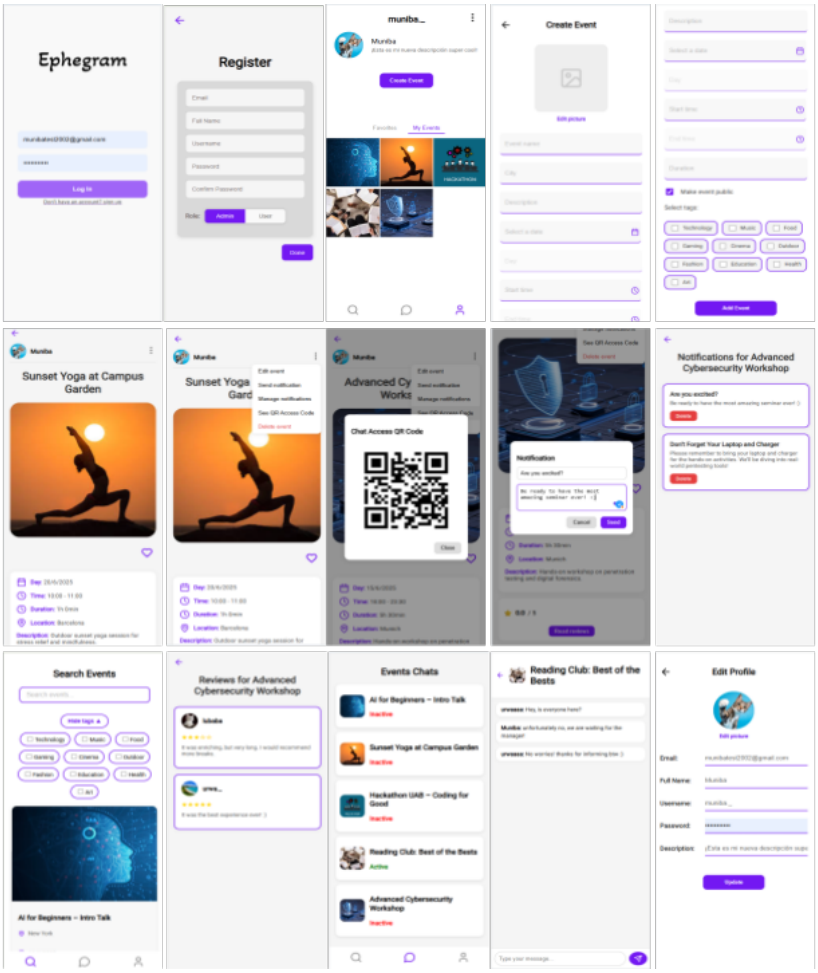
APPENDIX

A. GitHub Projects Timeline



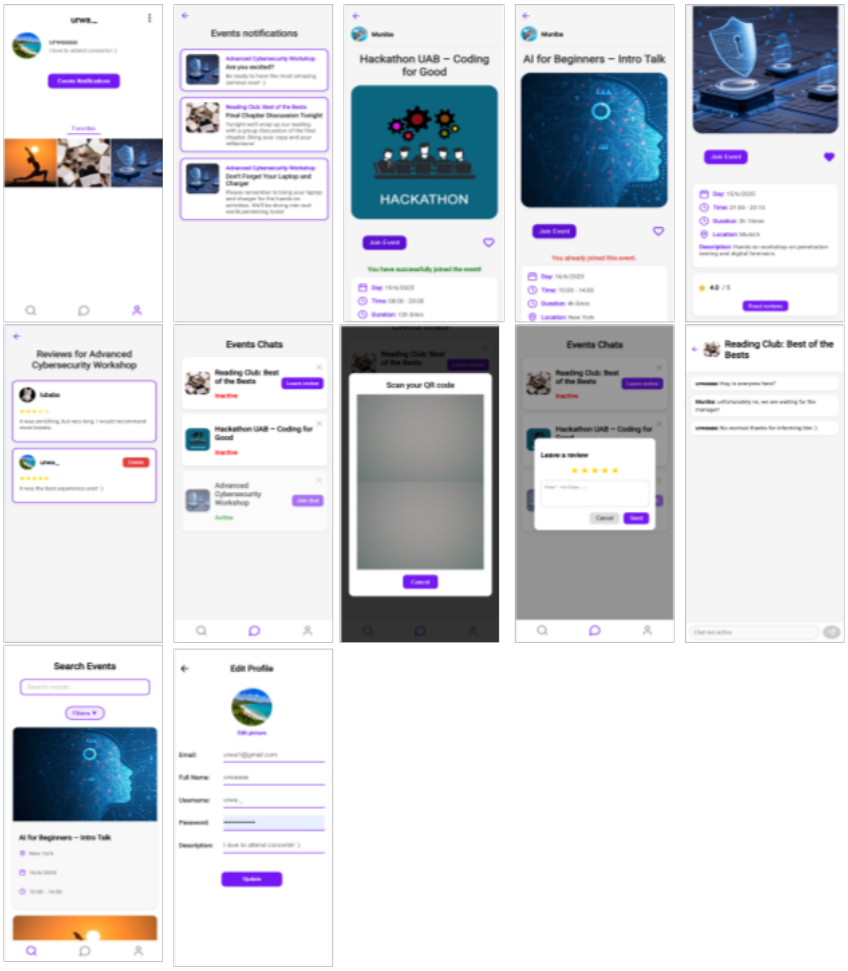
GitHub Projects board.

B. Admin Interface Screenshots



Admin view.

C. User Interface Screenshots



User view.