TEC | Tecnológico de Costa Rica

TECHNICAL MANUAL

“GJ+ PLATFORM”

MADE BY

JEFRY CUENDIZ CUENDIZ
DAVID PASTOR BARRIENTOS
SHARON CHACON RODRIGUEZ
PAUBLO ALEXANDER AVILA RAMIREZ

SAN JOSÉ
2024

SUMMARY

The GameJam+ organization has to face challenges around the organization of its video game events globally since they do not have a centralized platform that allows them to properly manage their data. In order to attack this problem, it is proposed to develop a platform that allows the registration of jammers, organizers and judges who can provide anonymous feedback. All users will be able to manage their time zone so that it is that of their country and thus there is better coordination between global and local organizers. In addition, the platform will also have internal and customizable forms that will allow it to be traceable and thus not depend on external sites, so that all the information is centralized.

As for jammer users, this platform will allow you to upload video games and their respective pitches within the established deadlines so that there are no unfair disqualifications. In turn, they will receive feedback, as mentioned above, from the judges anonymously for an impartial process. In itself, through the implementation of this project, developed in the MEAN webstack, the efficiency of operations, transparency in the process will be improved and there will be better collaboration so that GameJam+ can continue to grow and provide more opportunities to lovers of video game development and video games in general who want to be part of the process due to its volunteer nature.
Keywords: gamejam, jammers, organization, video games

Contents

List of Figures

Chapter 1

Introduction

1.1 About this manual

The purpose of this manual is to give a description of the system about its functionalities and scope, and also provides an installation guide for both the development environment and the production environment, so that the system can be deployed correctly. and can be maintained and configured correctly.

1.2 Description and Scope of the project

1.2.1 Background

The GJ+ is a non-profit organization which is responsible for organizing gaming events on a global level. However, it has to face various challenges that are affecting both the efficiency and administration of its events. These challenges are due to the fact that there is no centralized platform that allows participants to be registered, provide feedback to them anonymously, manage time zones so that there are no disqualifications due to this issue, and allow tasks to be coordinated between organizers at a local and global level.

By virtue of the above, we seek to centralize and streamline the corresponding processes, and that is why the implementation of a Web system is proposed. This platform will offer centralized and easy-to-use tools for different types of users of the system, ranging from participants to judges and local and global administrators.

In the same way, we seek to minimize the impact caused by using forms from external sites, such as those currently used in Coda.io, since one of the problems that currently exists is providing traceability to the forms of a specific team, due to the high quantities. That is why it ensures that the system has the capacity to resolve these concerns by implementing forms within the same platform and in the same way, stop using external sites for this.

On the other hand, the platform will allow crucial tasks to be carried out for the development of the events, such as uploading games and pitches, in an organized manner and within the established deadlines, so as to avoid unfair disqualifications due to lack of clarity with administrative aspects such as They are the dates and time zones. Additionally, including an anonymous feedback system will allow all teams to receive impartial evaluations of the projects, thus allowing for a transparent process.

Among the functionalities, there is also improved communication and collaboration between organizers by providing tools to assign tasks, monitor processes and manage grants. All of this will help to meet the objectives established by the global organization, giving greater cohesion and effectiveness to the team that organizes the tasks.

Therefore, the implementation of this platform developed on the MEAN Webstack for the GJ+ is

a comprehensive solution to combat the challenges faced by the organization in managing events by providing a centralized and easy-to-use platform. Through this project, operational efficiency will be improved, more impartial and transparent evaluations of projects will be given, and communication between users, as well as their collaboration, will benefit, thus allowing greater success and growth of the GJ+.

1.2.2 Goals

- **General Objective**
  - Develop a platform that allows the management of the GameJam + event, where all participants use the page for the registration process, uploading files and for the organizers to carry out the process of tracking the progress of the teams on their site, allowing a better flow of information about what's going on as well as the rating process for games from other sites by using MEAN as a development Webstack.
- **Specific Objectives**
  - Develop a platform that allows users to authenticate through Magiclink and contains an exclusive interface for organizers that provides them with access to administrative and event monitoring tools.
  - Implement an interface where the judges rate the jammers' games and that these ratings reach the jammers anonymously, as well as a space for each party to upload games and pitches on the platform.
  - Implement an online store system where local GameJam+ organizers earn points by meeting objectives proposed by global organizers, so they can redeem these points for store items.

1.2.3 Contact

For questions and support, contact one of the developers:

- Paulo Ávila: pavila@estudiantec.cr
- Sharon Chacón: sharon.chacon.r@estudiantec.cr
- Jefry Cuendiz: jefrycc99@estudiantec.cr
- David Pastor: davidpastorb@estudiantec.cr

Chapter 2

Functionality Specification

2.1 Product Description

The third product is a complete platform to manage the entire GameJam process. It allows users to complete the tasks necessary to participate in the event, from creating accounts to managing teams and rating games. The platform is intuitive, easy to use and has special features for different user roles.
Organizers can create events and locations to host them, manage teams and games, assign users, and track event progress. They also have the ability to set categories and themes for games and teams, which helps organize and categorize content.
Game developers (jammers) can register on the platform, create groups and submit their games for review. They can view the status of their entry, receive feedback from judges, and collaborate with other team members.
The judge is responsible for evaluating and scoring the matches presented. They have access to special tools to evaluate games and provide detailed feedback to developers. In addition to these basic functions, the platform offers additional functions such as reports, user management and the ability to download information in PDF format. Particular attention has been paid to ease of use and user experience thanks to a clear interface and intuitive system navigation.

2.1.1 User Stories

MVP 1

Code	Description	Quality Criteria
GJP-1	Register user. Develop an interface that allows users to register in the system by entering their personal data.	Registration must allow the entry of name, email and password. The entered data must be stored correctly in the database and a confirmation message must be displayed after successful registration.

Code	Description	Quality Criteria
GJP-2	Create a login for user entry. Develop a login interface that allows users to enter the system using their email and password.	The system must validate the credentials entered and allow access only if they are correct. It should display clear error messages in case of incorrect or missing credentials.
GJP-3	CRUD sites. Develop an interface that allows administrators to create, read, update and delete sites in the database.	Administrators should be able to view a list of all sites, create new sites, edit existing sites, and delete them. Changes should be reflected immediately in the user interface.
GJP-4	View jammer games by site. Develop an interface that allows local organizers to view games from jammers associated with their site.	Local organizers should be able to see a list of games associated with their site, with details such as game name, team, and creation date.
GJP-5	CRUD users. Develop an interface that allows administrators to create, read, update and delete users in the database.	Administrators should be able to view a list of all users, create new users, edit existing users, and delete them. Changes should be reflected immediately in the user interface.
GJP-6	Global can assign role to users. Develop functionality that allows global administrators to assign roles to users.	Global administrators must be able to select a user and assign them a specific role. Role changes should be reflected immediately in the system.
GJP-14	Develop the conceptual model of the database (Entity - relationship). Create a conceptual model that represents the entities and their relationships in the database.	The model must include all relevant entities and the relationships between them. It must be clear and understandable, and serve as a basis for the implementation of the logical model.

Code	Description	Quality Criteria
GJP-15	Transfer the conceptual model to the logical model (Relational). Convert the conceptual model into a logical model that can be implemented in a relational database.	The logical model must include tables, columns, and relationships that faithfully represent the conceptual model. It should be optimized for efficiency and to avoid redundancies.
GJP-17	Write user stories. Create user stories that describe the required functionalities from the end user's perspective.	User stories should be clear, concise, and follow the format "As [role], I want [functionality], for [benefit]." They must be reviewed and approved by the development team and stakeholders.
GJP-18	Establish the project story estimate. Estimate the effort required to complete each user story.	Estimates must be accurate and based on the team's experience and the complexity of each story. They must be reviewed and adjusted periodically.
GJP-20	Establish test cases for the first MVP. Define test cases that verify the functionalities of the first MVP.	Test cases should cover all possible scenarios and validate that the functionalities behave as expected. They must be executed and documented.
GJP-21	Class diagram. Create a class diagram that represents the structures and relationships of the classes in the system.	The diagram should be clear and detailed, including all classes, their attributes and methods, and the relationships between them. It should be used as a reference during development.
GJP-22	User interface prototype. Create a functional prototype of the user interface that allows displaying the visual elements for the second sprint.	The prototype should provide a preview of the system layout and navigation, allowing for validation and feedback before final implementation.
GJP-25	View organizers by sites. Develop an interface that allows you to see the organizers associated with each site.	The interface should display a list of organizers for each site, with details such as name and role. It should be easy to use and allow quick identification of organizers.

Code	Description	Quality Criteria
GJP-26	View judges by sites. Develop an interface that allows you to see the judges associated with each site.	The interface should display a list of judges for each site, with details such as name and role. It must be easy to use and allow quick identification of judges.
GJP-27	Filter games-teams by category. Develop a functionality that allows you to filter games and teams by category.	Users should be able to select a category and see only the games and teams that belong to that category. The filter must be fast and accurate.
GJP-28	Filter games-teams by themes. Develop a functionality that allows you to filter games and teams by theme.	Users should be able to select a topic and see only the games and teams that belong to that topic. The filter must be fast and accurate.
GJP-29	CRUD categories. Develop an interface that allows administrators to create, read, update and delete categories in the database.	Administrators should be able to view a list of all categories, create new categories, edit existing categories, and delete them. Changes should be reflected immediately in the user interface.
GJP-30	Registration window. Develop an interface that allows the registration of a new jammer in the database, entering their name, email and the site to which they belong.	The interface should allow users to enter their name, email, and select their site. A confirmation message should be displayed upon successful registration.
GJP-31	Login window. Develop an interface that allows a user to log in using Magiclink, requiring them to enter their email.	The interface should allow users to enter their email and receive a login link by email. The link must allow immediate access to the system.
GJP-32	Sites info window. Develop an interface that shows detailed information about the sites.	The interface should display details such as site name, location, and associated organizers. It should be easy to navigate and understand.

Code	Description	Quality Criteria
GJP-33	Window for information manipulation. Develop an interface that allows administrators to manage all system information.	Administrators must be able to create, read, update and delete information from all catalogs (sites, users, games, etc.). Changes should be reflected immediately.
GJP-34	CRUD themes. Develop an interface that allows administrators to create, read, update and delete topics in the database.	Administrators should be able to view a list of all topics, create new topics, edit existing topics, and delete them. Changes should be reflected immediately in the user interface.
GJP-39	Create burndown chart. Develop an interface that displays a burndown graph of the project.	The chart should show the progress of the project in terms of work completed and work remaining. It should be updated regularly and be easy to interpret.

MVP 2

Code	Description	Quality Criteria
GJP-3	**CRUD SITES** Develop an interface in Angular that allows the global organizer to create, read, update and delete sites from the database, along with the Mongo methods necessary for each of these functions	A global organizer user can view available sites, create new sites, edit existing sites, and delete existing sites. The changes made are immediately displayed on the screen.
GJP-4	**View jammer games by site** Develop an interface in Angular that allows a local organizer to see the games of the site to which it belongs, using calls to Mongo methods	By logging into the system as a local organizer it is possible to see and interact with a interface that allows you to see the games on the site to which it belongs
GJP-5	**CRUD users** Develop an interface in Angular that allows the global organizer to create, read, update and delete users from the database, along with the Mongo methods necessary for each of these functions	A global user organizer can view available users, create new users, edit existing users, and delete existing users. The changes made are immediately displayed on the screen.

Code	Description	Quality Criteria
GJP-29	**CRUD categories** Develop an interface in Angular that allows the global organizer to create, read, update and delete categories from the database, along with the Mongo methods necessary for each of these functions	A global organizer user can view available categories, create new categories, edit existing categories, and delete existing categories. The changes made are immediately displayed on the screen.
GJP-30	**Registration window** Develop an interface in Angular that allows the registration of a new jammer in the database, through Mongo methods, requiring that you enter your name, your email and the site to which it belongs.	A user is able to enter their name, email address and the site at that belongs. A globalorganizer is able to verify that a new user has registered from the information manipulation window
GJP-31	**Login Window** Develop an interface in Angular that allows a user to log in using Magiclink, requiring them to enter their email	By entering an email and clicking on the send email button, an email is sent with a link that allows you to enter the system.
GJP-32	**CRUD SITES** Develop an interface in Angular that allows the global organizer to create, read, update and delete sites from the database, along with the Mongo methods necessary for each of these functions	A global organizer user can view available sites, create new sites, edit existing sites, and delete existing sites. The changes made are immediately displayed on the screen.
GJP-33	**Window for information manipulation** Develop an interface in Angular that allows the global organizer to create, read, update and delete all the database catalogs, along with the Mongo methods necessary for each of these functions	A global organizer user can view available sites, create new ones, edit and delete site catalogs, regions, users, teams, submission and existing topics and categories. The changes made are immediately displayed on the screen.
GJP-34	**CRUD themes** Develop an interface in Angular that allows the global organizer to create, read, update and delete themes from the database, along with the Mongo methods necessary for each of these functions	A global organizer user can view available themes, create new themes, edit existing themes, and delete existing themes. The changes made are immediately displayed on the screen.

Code	Description	Quality Criteria
GJP-40	**Interface - Jammer Home:** A home interface should be developed that is focused on jammers that allows quick access to the functions and features of the system according to what its role can perform.	The interface offers participants a user experience according to their needs, in such a way that they intuitively have access to each of the functionalities they require as well as the use of key tools to participate in the GameJam effectively.
GJP-49	**Interface - Jammer Submit Game:** Develop a nice looking interface that allows jammers to submit their games for evaluation or update the submit if necessary.	The interface for submitting games provides a fluid and easy-to-use experience, so that games can be submitted efficiently and without any complications.
GJP-50	**Backend-Jammer block submit after time limit:** Implement a restriction on the backend so that jammers cannot submit their games after the time available in the current phase has expired.	The restriction allows for better integrity and fairness in the judging process so that users can submit their games on time so that teams that submit late are not given an unfair advantage.
GJP-51	**Interface - Jammer View Team:** Program a visually pleasing interface so that jammers can see and manage all their team information.	The interface allows the participants to view the teams in a clear and orderly manner with all the important information, thus allowing better collaboration and proper coordination of the members of a team.
GJP-54	**Interface - Judge Rate Game:** Develop an interface that allows judges to evaluate and rate games in a simple and accurate way.	The game rating interface provides a pleasant, smooth and efficient user experience to perform your task of evaluating and rating games accurately.
GJP-57	**Backend - Jammer Time Limit to Submit:** Set a time limit on the back-end so that jammers can send the submit. Additionally, a counter is displayed with this	The time limit is appropriate for the current GameJam and allows the missing time to send the submit to be made up

Code	Description	Quality Criteria
GJP-59	**Backend - Judge Assign game to rate:** Develop the functionality in the back-end to assign a judge the game to develop.	The game is correctly assigned to the judge, ensuring that it is a local organizer that does not evaluate the same site. In addition, a fair and organized evaluation is allowed.
GJP-60	**Backend - Judge send Email with rating:** Allows that when a judge enters a rating in the system, an email with the evaluation information is sent to each of the team members.	The email sent to the user does not contain the name of the judge for transparency and the information is sent in a pleasant way to the user so that the information presented is better understood.
GJP-61	**Interface Prototype 2:** Create a functional prototype of the user interface that allows displaying the visual elements for the second sprint.	The interface prototype provides a preview of the design and navigation that the system will have, allowing the end user and developers to validate the proposal and obtain feedback before passing it to code.
GJP-62	**Technical Manual 2:** Prepare the detailed technical manual for the second sprint that shows all the changes made.	The technical manual documents in detail each of the points developed, then explaining the structure and operation of the system for its proper maintenance and administration in the future.
GJP-63	**User Manual 2:** Prepare the user manual for the second sprint that has clear and precise instructions on how to use each of the components of the interface as well as the features from the perspective of the end user.	The user manual explains in detail each of the tasks that end users can perform, thus allowing you to get the most out of the interface with clear and precise guidance.
GJP-64	**Sprint 2 report:** Write a detailed report that documents the specific progress and results obtained in the second sprint, adequately showing the tasks completed as well as the time it took to complete them.	The report is completely attached to what happened and the progress of the project, showing the achievements that were achieved, adding new information to the system.

Code	Description	Quality Criteria
GJP-65	**Angular - Jammer Team:** Develop a graphical interface in Angular that allows a jammer to view his team and manage it properly during the course of a GameJam	The interface is easy on the eyes and functional at the same time, so team members can easily and quickly manage their information from there.
GJP-66	**Angular - Jammer Submit:** Develop a graphical interface in Angular that allows jammers to submit their juices for evaluation as long as there is time available.	The interface is functional and pleasant to look at, it also shows the time remaining until the phase is finished and also allows jammers to send the games and their evaluations more easily and quickly.
GJP-67	**Angular - Judge:** Develop a graphical interface through Angular that allows judges to access its functionalities quickly and accessible.	The interface is functional and pleasing to the eye while being intuitive and allows judges to perform all their game evaluation tasks properly.
GJP-68	**Backend-Edit User Role:** Implement functionality in the back-end that allows the global organizer to modify a user's role in the system.	This feature gives administrators the ability to manage user roles in a flexible and efficient way while ensuring that the user with the appropriate permissions can do so.
GJP-69	**Define test cases:** Write a series of test cases that cover the different cases and scenarios that can happen when using the system to guarantee the operation and quality of the system.	The test cases allow you to exhaustively test all the functionalities implemented in the second sprint, thus identifying problems that may arise while providing a clear guide to evaluate compliance with the requirements and so that errors can be detected appropriately.
GP-70	**Execute tests:** Based on the established test cases, the state of the test case is executed and classified appropriately in the excel sheet to know the current state of the system.	Developers can execute all tests set in the system properly without the need for third party intervention and the status of test cases is filled honestly and transparently.

Code	Description	Quality Criteria
GJP-71	**Get the games that a judge evaluates:** Implement the functionality in the backend so that the judges can obtain a list of the games that they should evaluate.	This function allows judges to quickly and accurately access information on the games evaluated by a specific judge so that they can carry out the evaluation process.

MVP 3

Code	Description	Quality Criteria
GJP-00	**Upload PDF of category manuals (backend)** Develop an interface to upload category manuals in PDF format	Users can select and upload PDF files of category manuals through the interface, and these are successfully stored in the system.
GJP-01	**Upload PDF of category manuals (Frontend)** Develop an interface to upload category manuals in PDF format	Users can select and upload PDF files of category manuals through the interface, and these are successfully stored in the system.
GJP-02	**Upload PDF of topic manuals (Frontend)** Develop an interface to upload topic manuals in PDF format	Users can select and upload PDF files of topic manuals through the interface, and these are successfully stored in the system.
GJP-03	**Upload PDF of topic manuals (backend)** Develop an interface to upload topic manuals in PDF format	Users can select and upload PDF files of topic manuals through the interface, and these are successfully stored in the system.
GJP-04	**Fix page numbers in Global UI** Fix display of page numbers in global UI	Page numbers are displayed correctly and consistently across all sections of the global interface.
GJP-05	**Resize global things** Resize global interface elements to improve usability	The global interface elements are appropriately sized, improving usability and user experience.
GJP-06	**User Dashboard** Develop a dashboard for users that displays relevant information in an accessible way	The control panel displays relevant information for the user in a clear and organized manner, with shortcuts to key functionalities.

Code	Description	Quality Criteria
GJP-07	**Translate the technical manual into English** Translate all contents of the technical manual into English	The technical manual is completely translated into English, maintaining the coherence and technical precision of the original.
GJP-09	**Refactor PDF export filter** Improve existing filter for exporting data to PDF	The filter is more efficient and allows you to export data to PDF quickly and accurately, with additional customization options.
GJP-72	**Remove IDs from the global organizer UI** Remove visible IDs in the global organizer interface to improve presentation	IDs are no longer displayed in the global organizer interface, improving presentation and usability.
GJP-72	**Change topic and category language acronyms** Update language acronyms in topic and category sections	Language acronyms are displayed correctly and updated in all topic and category sections.
GJP-73	**Refactor global interface** Improve the global organizer user interface for greater usability	The global organizer interface is more intuitive and easy to use, allowing administrators to perform their tasks efficiently.
GJP-74	**Plan Tests** Develop a detailed plan to perform extensive system testing	The test plan covers all aspects of the system, ensuring that extensive testing is performed to ensure quality.
GJP-75	**Run tests** Run planned tests and document results	Tests are executed according to plan, and results are accurately documented to identify and correct problems.
GJP-76	**Technical manual** Prepare the detailed technical manual for the system	The technical manual documents in detail each of the points developed, explaining the structure and operation of the system for its proper maintenance and administration in the future.
GJP-77	**User Manual** Prepare the detailed user manual for the system	The user manual explains in detail each of the tasks that end users can perform, allowing them to get the most out of the interface with clear and precise guidance.

Code	Description	Quality Criteria
GJP-78	**Sprint report** Write a detailed report documenting the specific progress and results obtained in the sprint	The report accurately documents the progress and results of the sprint, showing the achievements achieved and the time spent on each task.
GJP-79	**Define test cases** Write a series of test cases that cover the different use scenarios of the system	Test cases allow you to test all functionality exhaustively, identifying problems and ensuring that requirements are met.
GJP-88	**Translate the user manual into English** Translate the entire content of the user manual into English	The user manual is completely translated into English, maintaining the consistency and precision of the original.
GJP-90	**Refactor judge window** Improve the user interface of the judge window for greater usability	The interface is more intuitive and easy to use, allowing judges to perform their evaluation tasks efficiently.
GJP-91	**List of topics and categories in the submission** Implement a drop-down list of topics and categories in the submission section of submissions	Users can clearly and easily select the topic and category in the submission section.
GJP-92	**Show regions even if they are empty** Set the interface to show all regions, even if they have no data	All regions are displayed in the interface, improving visibility and navigation for the user.
GJP-93	**Local UI creates site** Develop a user interface that allows a local organizer to create a new site	The local organizer can create a new site through an intuitive interface, and the site is successfully saved to the system.
GJP-94	**Local backend creates site** Implement functionality in the back-end so that a local organizer can create a new site	The local organizer can create a new site, which is correctly saved in the database and visible in the corresponding interface.
GJP-95	**Job BACK auto-assign judges for the end of the submission** Implement an automatic process that assigns judges to the submissions at the end of the submission period	The system automatically assigns judges to submissions, ensuring equitable distribution and avoiding conflicts of interest.

Code	Description	Quality Criteria
GJP-96	**Job to order submissions by number of evaluations** Implement an automatic process that orders submissions according to the number of evaluations received	The system automatically executes the process that orders the submissions by the number of evaluations, updating the information in real time.
GJP-97	**Change CRUDS confirmation component** Update the confirmation component for CRUD	operations The new confirmation component is more intuitive and provides a better user experience for CRUD operations.
GJP-99	**Refactor jammer window** Improve the jammer window UI for better usability	The interface is more intuitive and easy to use, allowing jammers to perform their tasks efficiently.
GJP-103	**Search filters and download CRUD information in PDF (Global)** Implement search filters and the option to download CRUD information in PDF format	Users can efficiently filter information and download the results in PDF format from the global interface.
GJP-104	**View PDF from user** Implement functionality so that users can view PDF files directly from their dashboard	Users can open and view PDF files from their user panel, without needing to download them.
GJP-105	**Add PDF from global interface** Implement functionality so that the global organizer can upload PDF files from the	interface The global organizer can select and upload PDF files, and these are correctly stored on the system, available for download.
GJP-106	**Refactor registration** Improve the user registration interface for greater usability and efficiency	The registration interface is more intuitive and efficient, facilitating the registration process for new users.
GJP-107	**Always view submit** Ensure that users can view your submissions at any time	Users can access and view their submissions at any time, without time restrictions.
GJP-108	**Jammer sees submissions** Implement a feature so that jammers can see submissions	Jammers can access and view submissions in a clear and organized way in the interface.
GJP-109	**Chat between local and jammer** Develop a chat interface that allows communication between a local organizer and a jammer	Local organizer and jammer users can send and receive messages in real time within the platform, with notifications of new messages.

Code	Description	Quality Criteria
GJP-110	**Average and send final evaluation (back)** Implement the functionality to calculate the average of the evaluations and send the final result	The system automatically calculates the average of the evaluations and sends the final result to the corresponding users.

Chapter 3

System architecture

3.1 General system design

The design of the system follows an MVC architectural pattern that allows for greater order in the project by identifying each part of it, thus following an order that is easy to follow in the code. MVC stands for model, view, and controller. Model is the backend that contains all the data logic, the view is the frontend or graphical user interface (GUI) and the controller is the brain of the application that controls how the data is displayed. The MVC pattern helps to divide the frontend and backend code into separate components, making it much easier to manage and make changes to either side without interfering with each other. The above explained according to Hernandez (2021).

The MEAN web stack will be used for development. The MEAN stack is a JavaScript-based framework for developing web applications. MEAN is named after MongoDB; non-relational document database for data management, Express; nodejs library for communication between the front and the base, Angular; client-side JavaScript framework and Node; for the server side, the four key technologies that make up the layers of the (MongoDB, n.d.-b) stack.

The Angular framework follows a very orderly MVC architecture, which adapts and simplifies the follow-up from design to development as both follow the same pattern. Going into more detail, the design of the system architecture is presented as follows.

The model part contains the different data components which are the different entities of the system. MongoDB is used for data management.

For the controller part, it is the part that is in charge of the logic and is the medium between the front and the back, it is the intermediary between the view and the data, it contains the different functions that the application requires and that respond to the requests that the user makes through the interface. For this part we work with Nodejs, using ExpressJS for the server to develop the API that is responsible for handling client requests and communication with the database.

Finally, the view part is the user interface, which the user interacts with directly. For the interface part, Angular is used.

3.1.1 Persistence design

The persistence design is shown in the image below, it is explained like this: There is the GameJam event, which is related to the themes for games and categories, these events occur on different sites; These are the places where events occur, which belong to regions. These events are led by global organizers who are related to a gamejam as they coordinate its data. Then there are the local organizers who are related to the teams, who are in charge of coordinating the teams on your site. The other user or role is the judge, who is then responsible for grading the team games, so this is related to the teams' deliveries. The team is made up of Jammers who are also another user in the system, they are the participants, the team is related to a GameJam, they are related to the deliveries, since they must make deliveries for each phase in which they advance in the GameJam. There is chat made up

of participants and messages, this chat is for jammers and local organizers, a local organizer can start a chat with a jammer or whith a team, so it is related to User to refer the jammer and local organizer and related with team.

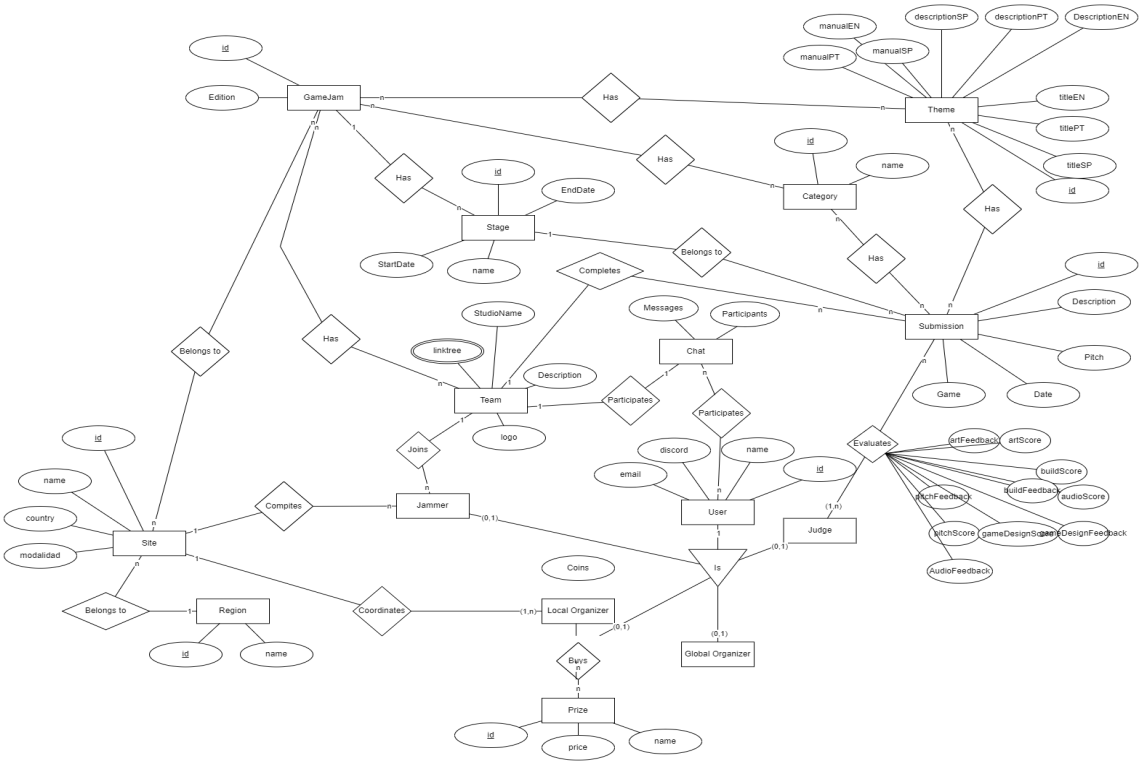

Figure 3.1: Entity-Relationship Model

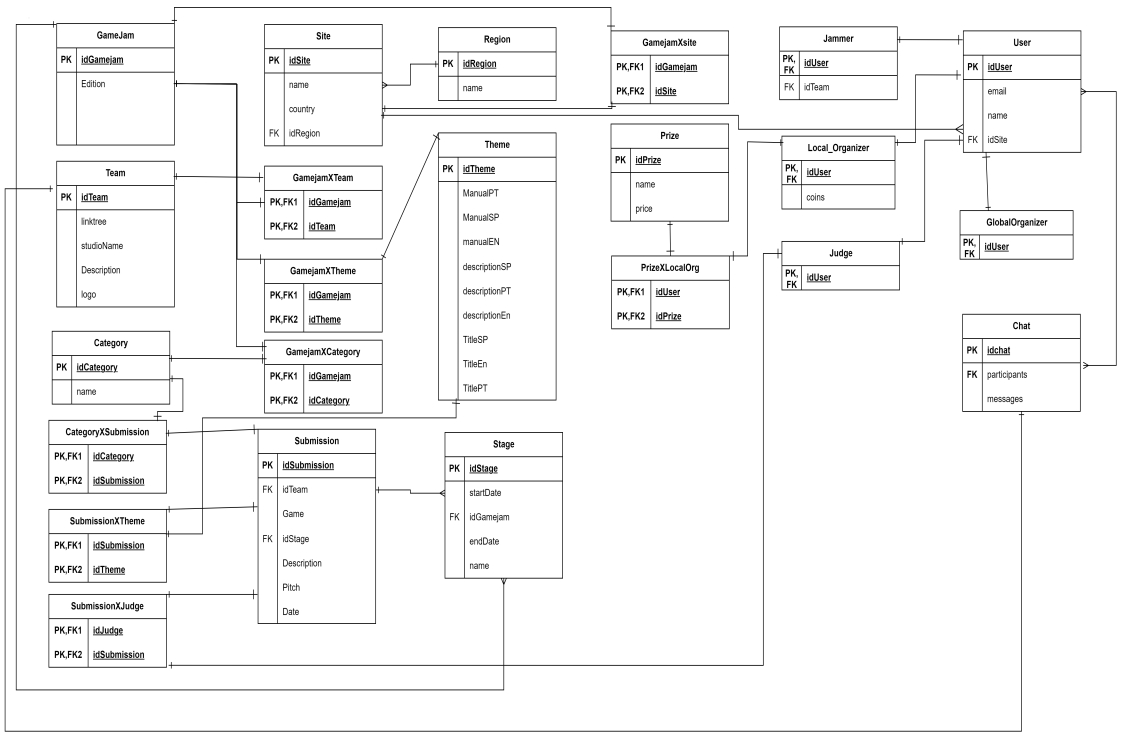

Figure 3.2: Logical Model

Now the logical model is presented, following what was previously explained in the entity-relationship model. In the following diagram you can see what the connection between these entities

is like, using a logical model to present the persistence design in greater detail, showing at a logical level how the tables should look or at least continue with the logic presented here to follow the same idea and respect the data requirements for this system to be developed. The data logic follows the following layout and relationships.

Below the abbreviation FK will be used to refer to foreign key and PK to refer to primary key. For entities or simple tables, their primary key is an id, a unique identifier for each data in a table, so for the following tables this attribute is omitted with the exception of certain tables that have a different key.

There is a GameJam with its edition.

You have a team with its name, a possible logo (optional), description of the team and linktree for links of interest that the group wishes to add. A category with your name.

A site with its name, country to which it belongs, and id of the region it belongs to, which is a foreign key to the Region table.

A delivery with an id of the team to which the delivery belongs (FK), link to the game, id of the stage in which the delivery is uploaded (FK), description of the delivery given by the team, link to the pitch and date of Delivery.

Region with name.

Topic with the manual, description and title, there are fields for three different languages.

The stage has its start date, id to the GameJam it is related to (FK), end date and name.

A prize with your name, description and price.

A user with his or her email (unique), site id (FK) and name, from this user the other users inherit: Jammer, Judge, Local Organizer and Global Organizer.

Jammer adding the id of the team it belongs to (FK).

And the organizer with one more attribute of coins, which are those used to purchase in the store, is currently assigned to the local organizer but you may want to expand it to other users, it is possible to manage it in the general user.

CategoriaxDelivery, each delivery has n categories associated with it, there is a primary key composed of the foreign keys of the category id and the delivery id.

TemaxDelivery, each delivery has at least one topic associated with it, there is a primary key composed of the foreign keys of the topic id and the delivery id.

DeliveryxJudge, each judge qualifies n deliveries, there is a primary key composed of the id of the judge and delivery.

GamejamxTeam, PK composed of the ids of the current GameJam and the team that joined the event, both FK.

GamejamxTema, each event has n themes that are revealed in the live competition, for this a relationship is added where its PK is the composition of the FK, GameJam id and theme id.

GamejamxCategoria, list of the categories associated with each gamejam, the PK is the composition of the FK, GameJam id and category id.

GamejamxSite, each event has n non-fixed associated sites for each edition, PK composed of the FK, GameJam id and site id.

PremioxOrganizadorLocal, record of prize exchanges, PK composed of the FK id of the user and id of the prize they requested.

3.1.2 Class diagram

The model section of the third deliverable presents a detailed structure of the data that makes up the video game development event management platform. A GameJam, the central event on the platform, can take place on one or more sites and consists of one or more stages. Each region encompasses one or more aggregated sites. Users, who can be GlobalOrganizer, LocalOrganizer, Jammer or Judge, are related to a site and have different roles within the system.

Teams, made up of one or more Jammers who are already registered in the system, generate one or more Submissions, which are closely linked to the team. Judges are associated with the Submissions and are responsible for evaluating the games. Each Submission can be associated with one or more topics and no or more categories, which is reflected in the aggregation relationship with Theme and Category. Prizes are associated with LocalOrganizers, although this association may change in the future.

The controller, divided into several parts based on model classes, manages user requests and data-related operations. Each controller handles basic operations such as create, modify, get, and delete for its respective data type. Although some controllers have more specific functionalities, such as the user controller, which is responsible for logging in and managing users associated with sites and regions.

The project, in its third deliverable, has already completed the development of views for user login, visualization of local and global organizers, as well as game and team information. Specific components have been designed for event data management, display of site and equipment information, and the login window.

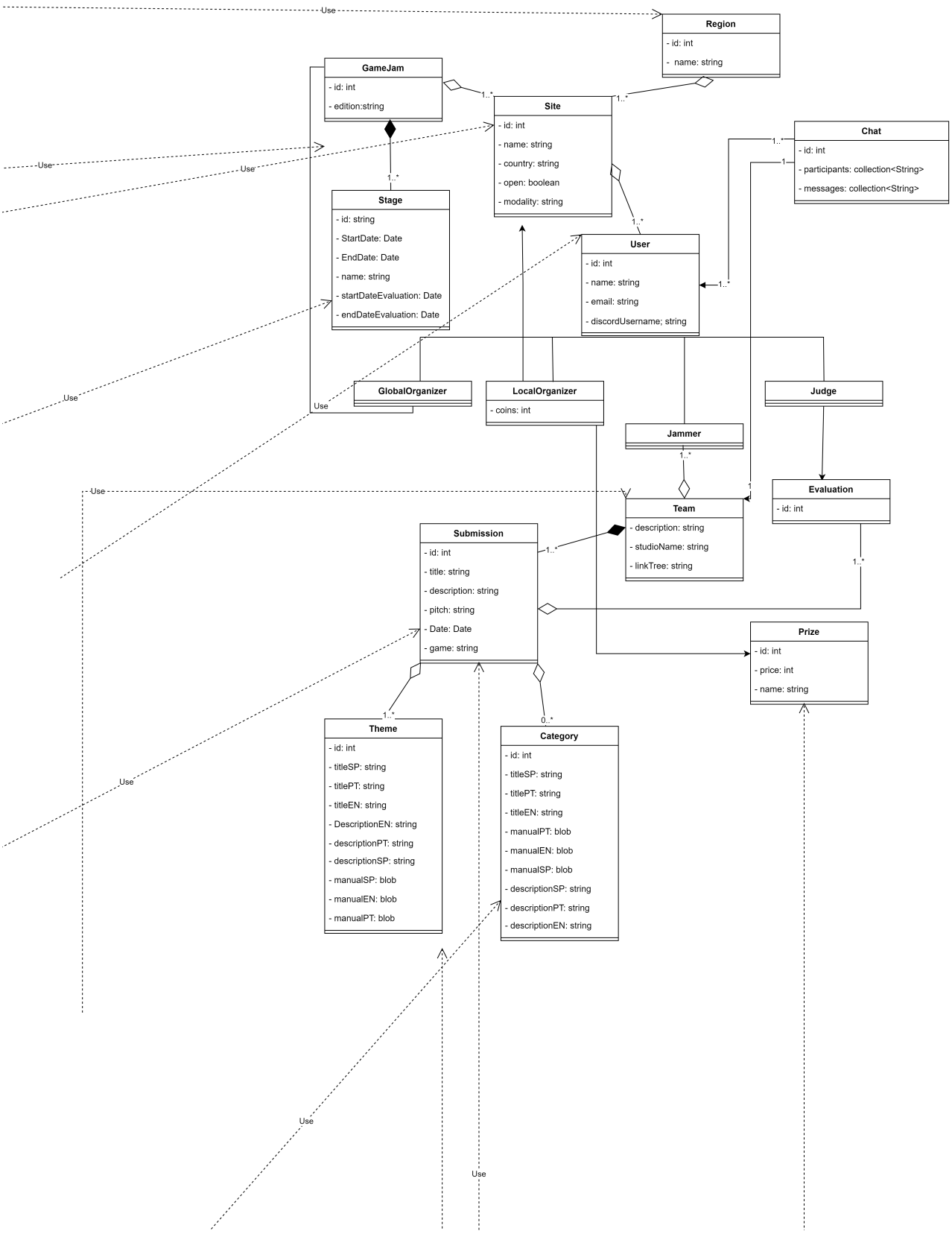

Figure 3.3: Class Diagram Model

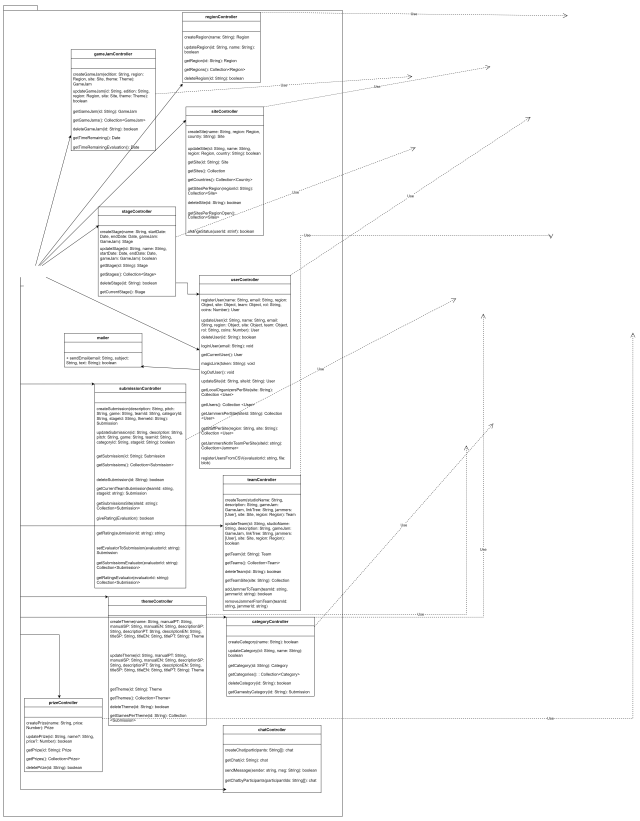

Figure 3.4: Controller

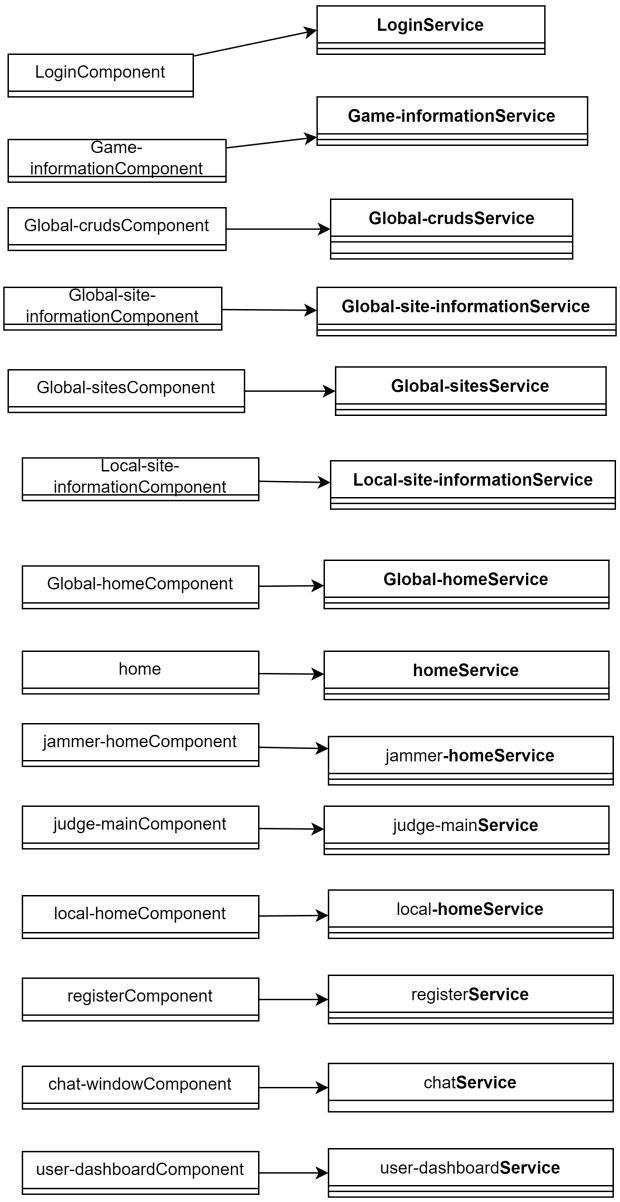

Figure 3.5: View

Chapter 4

Installation Instructions

4.1 Development Environment

4.1.1 Dependencies

Describe the development environment and then list each of the dependencies in the order in which you should install them.

- **MongoDB:** Taking as reference what was explained in MongoDB (n.d.-a).

  1. Configure the package management system (yum).

  Use the following command to create a file to install MongoDB directly using yam.

  vi /etc/yum.repos.d/mongodb-org-7.0.repo

  Edit the file with the following information

  [mongodb-org-7.0]
  name=MongoDB Repository
  baseurl=https://repo.mongodb.org/yum/redhat/8/mongodb-org/7.0/x86$_6$4/
  $gpgcheck = 1$
  $enabled = 1$
  $gpgkey = https://pgp.mongodb.com/server-7.0.asc$

2. Install MongoDB packages

sudo yum install -y mongodb-org

- **Nodejs** Steps were taken on the Jethva (2022) site to install it using NVM.

1. Install NVM. This command will download the NVM installation script and run it to install NVM on your system. After the err command and reopen the terminal.

curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/master/install.sh — Bash

2. Verify installation

nvm –version

3. Install nodejs specific version

nvm install 14.17.0

4. Set the version of Nodej.js to use

nvm use 14.17.0

5. Verify node

node –version

6. Check npm

npm –version

- **ANGULAR** Steps taken on Bello (2021) site

  1) Install angular version 17.2.3

  sudo npm install -g @angular/ cli@17.2.3

  2) Check installation and version

  ng –version

4.1.2 Source code

The code is in a Github repository, so you need to clone the code.

- **Github:** Taken the steps on the Oracle (2023) site to install it using NVM.

  1. Install Git

  sudo dnf install git -y

  2. Verify installation

  git –version

  3. Git clone https://github.com/Pr-Ing-S-Gamma/GJ-Platform.git

- **Express** It is already a dependency within the project, but if necessary.

  1) Go to the project folder and install express in the project.

  npm install express –save

- **Start angular** Once you have the project, angular is started and the necessary npm dependencies are installed.

  npm install

4.1.3 Deployment in test environment

Describe the steps to build or deploy the system for live testing in a test environment.

4.2 Production Environment

4.2.1 Dependencies

- **MongoDB:** Taking as reference what was explained in MongoDB (n.d.-a).

  1. Configure the package management system (yum).

  Use the following command to create a file to install MongoDB directly using yam.

  vi /etc/yum.repos.d/mongodb-org-7.0.repo

  Edit the file with the following information

  [mongodb-org-7.0]
  name=MongoDB Repository
  baseurl=https://repo.mongodb.org/yum/redhat/8/mongodb-org/7.0/x86$_6$4/
  $gpgcheck = 1$

$enabled = 1$
$gpgkey = https://pgp.mongodb.com/server-7.0.asc$

2. Install MongoDB packages

sudo yum install -y mongodb-org

- **Nodejs** Steps were taken on the Jethva (2022) site to install it using NVM.

1. Install NVM. This command will download the NVM installation script and run it to install NVM on your system. After the err command and reopen the terminal.

curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/master/install.sh — Bash

2. Verify installation

nvm –version

3. Install nodejs specific version

nvm install 14.17.0

4. Set the version of Nodej.js to use

nvm use 14.17.0

5. Verify node

node –version

6. Check npm

npm –version

4.2.2 Installation instructions

Once the dependencies mentioned in the previous section have been installed, the following must be done:

Clone the repository if you have not done so in the previous section. The main branch will be cloned, which is the one with the production version.//

Git clone https://github.com/Pr-Ing-S-Gamma/GJ-Platform.git

Run the following command to install the project dependencies

cd GJ-Platform/back-end

npm i

Then you must go to the following path and install the dependencies for the angular project

GJ-Platform/Frontend/GJ-Platform

npm i

Before running the server, you must go to the .env variable, located at the following path

`back-end / .env`

In the third line, in the PORT variable, change it to where the server is going to be installed, generally it is the IP or the domain if you have it, which will be used to access the web system.

This same thing needs to be done in the path

`Frontend GJ-Platform src environments environment.prod.ts`

And here, you need to change the ApiURL to the address of the domain where the system will be installed.

Then run ng build –configuration production

Finally, it remains to configure PM2 as follows: In case of errors that pm2 was not found, it is installed as follows

npm install pm2

or to install it globally

npm install -g pm2

Next, you must locate the index.js file in the backend folder and start the pm2 service in this way pm2 start index.js –name ”production”

And the system will be available for use.

Now at least 1 user of type GlobalOrganizer must be added as follows. This is so that you can use the system for the first time and configure everything you need, otherwise you will not be able to access the system in any way. It is done in the following way:

`http://localhost:3000/api/users/register-user?name=Name&email=mail@example.com&role=GlobalOrganizer&coins=100&discordUsername=UserName`

Or if you have a tool to make calls to APIs, you can send the following JSON using the following route along with a json following this format.

```
//localhost:3000/api/users/register-user

{
"name": "Name",
"email": " email@example.com ",
"role": "GlobalOrganizer",
"coins": 100,
"discordUsername": "UserName"
}
```

Finally the system will be loaded, you must go to the server IP or DNS in which the system was installed.

Chapter 5

API Methods

This section details all the methods published in the back-end section of the system. Likewise, a description, formats, parameters, results and any other procedure necessary to understand and correctly use its API are attached.

5.1 For all methods

All database tables have audit fields. This specifically includes saving the user who modifies and creates catalogs. That is why to ensure that the system methods can be executed correctly, it is important to have a session logged into the system. To log in to the system you must:

- Register a user, if you do not have one (refer to section 5.2.1)
- Log in to the system. 5.2.3
- Get a Magic Link, to get the token. 5.2.4

Once logged in, you can perform all the other actions, put delete post and get of each section detailed in this document.

5.2 Users

This section will detail the methods that involve users.

5.2.1 register-user

This method allows you to register a new user in the system.

Name	Type	Required	Description
name	String	Yes	User name.
discordUsername	String	Yes	Discord User.
email	String	Yes	User email.
region	Object	Yes	Object containing the ID and name of the user's region.
site	Object	Yes	Object containing the user's site ID and name.
team	Object	No	Object containing the user's team ID and name.
role	String	Yes	User role.
coins	Number	Yes	Amount of user's coins.

It is important to clarify that the region and the site must have been created in the system. Example of use:

```
POST localhost:3000/api/user/register-user
Content-Type: application/json

{
"name": "User Name",
"discordUsername" : "user#127",
"email": " email@example.com ",
"region": { "_id": "Region ID", "name": "Region Name" },
"site": { "_id": "Site ID", "name": "Site Name" },
"team": { "_id": "Team ID", "name": "Team Name" },
"role": "LocalOrganizer",
"coins": 100
}
```

Response messages:

Validation	Description	HTTP Code	Message
Invalid email address error	The email address provided is invalid	403	Invalid email address.
Duplicate error	The email is already in use	409	The email is already in use.
System error	Error executing request	400	Error message
Action successful	User registered successfully	200	Registered successfully

5.2.2 update-user

This method allows you to update the information of an existing user in the system.

Name	Type	Required	Description
id	String	Yes	ID of the user to update.
name	String	No	New user name.
discordUsername	String	Yes	Discord User.
email	String	No	New user email.
region	Object	No	New object containing the ID and name of the user's region.
site	Object	No	New object containing the user's site ID and name.
team	Object	No	New object containing the user's team ID and name.
role	String	No	New user role.
coins	Number	No	New amount of user's coins.

Example of use:

```
PUT localhost:3000/api/user/update-user/1
Content-Type: application/json

{
"name": "New Name",
"discordUsername" : "user#127",
"email": " new_mail@example.com ",
"region": { "_id": "New Region ID", "name": "New Region Name" },
"site": { "_id": "New Site ID", "name": "New Site Name" },
"team": { "_id": "New Team ID", "name": "New Team Name" },
"role": "New Role",
coins: 150
}
```

Response messages:

Validation	Description	HTTP Code	Message
Invalid ID error	The provided user ID is invalid	400	Invalid user ID.
User not found	User not found	404	User not found.
Invalid email address	The email address provided is not valid	403	Invalid email address.
Duplicate error	The email is already in use	409	The email is already in use.
System error	Error executing request	400	Error message
Action successful	User updated successfully	200	User updated successfully.

5.2.3 login-user

This method allows you to log in to the GameJam platform.

Name	Type	Required	Description
email	String	Yes	User email.

Example of use:

```
POST localhost:3000/api/user/login-user
Content-Type: application/json

{
"email": " email@example.com "
}
```

Response messages:

Validation	Description	HTTP Code	Message
User not found	User not found	404	User not found.
Action successful	Magic link sent to user	200	Magic link sent successfully.

5.2.4 magic-link

This method processes the received token and generates a magic link to log in.

Name	Type	Required	Description
token	String	Yes	JWT token received.

Response messages:

Validation	Description	HTTP Code	Message
Error processing token	Error decoding JWT token	400	Error processing token.
Successful action	Redirect user to home page	302	Redirection to home page.

5.2.5 logout-user

This method allows you to log out of the GameJam platform.

Name	Type	Required	Description
None	-	-	-

Response messages:

Validation	Description	HTTP Code	Message
Action successful	Session cookie has been deleted	200	Cookie deleted successfully.

5.2.6 get-current-user

This method obtains the user currently authenticated on the GameJam platform.

Name	Type	Required	Description
None	-	-	-

Response messages:

Validation	Description	HTTP Code	Message
User not found	User not found	404	User Not Found.
Action successful	User data returned	200	User data returned successfully.

5.2.7 update-site

This method allows you to update a user's site on the GameJam platform.

Name	Type	Required	Description
id	String	Yes	User identifier.
siteId	String	Yes	Identifier of the new site.

Example of use:

```
PUT localhost:3000/api/user/update-site/:id
Content-Type: application/json

{
"siteId": "siteId"
}
```

Response messages:

Validation	Description	HTTP Code	Message
User not found	User not found	404	User not found.
Action successful	User site has been updated	200	User site updated successfully.

5.2.8 get-local-organizers-per-site

This method obtains local organizers for a specific site on the GameJam platform.

Name	Type	Required	Description
siteId	String	Yes	Site Identifier.

Response messages:

Validation	Description	HTTP Code	Message
Organizers found	Local organizers found	200	Organizers found for site.
Request failed	Error searching for organizers	400	Error searching for organizers.

5.2.9 get-staff-per-site

This method obtains the personnel associated with a specific site.

Name	Type	Required	Description
region	String	Yes	Site region ID.
site	String	Yes	site ID.

Example of use:

```
GET localhost:3000/api/user/staff/:region/:site
```

Response messages:

Validation	Description	HTTP Code	Message
Search successful	Staff found in the system	200	Staff have been found in the system.
Request error	Error processing request	400	Specific error message.

5.2.10 get-users

This method gets all the users registered in the system.

Name	Type	Required	Description
None	-	-	-

Example of use:

```
GET localhost:3000/api/user/all
```

Response messages:

Validation	Description	HTTP Code	Message
Search successful	Users found in the system	200	Users have been found in the system.
Request error	Error processing request	400	Specific error message.

5.2.11 get-jammers-per-site

This method returns all users with the ”Jammer” role on a specific site.

Name	Type	Required	Description
siteId	String	Yes	ID of the site for which users will be obtained.

Example of use:

```
GET localhost:3000/api/user/getJammers/:siteId
```

Response messages:

Validation	Description	HTTP Code	Message
Search successful	Users with the role ”Jammer” found	200	Users with the role ”Jammer” have been found in the system.
Request error	Error processing request	400	Specific error message.

5.2.12 get-jammers-not-in-team-per-site

This method gets users with the ”Jammer” role who are not on any computer associated with a specific site.

Name	Type	Required	Description
siteId	String	Yes	site ID.

Example of use:

```
GET localhost:3000/api/user/jammers/notinteam/:siteId
```

Response messages:

Validation	Description	HTTP Code	Message
Search successful	Users with the role ”Jammer” who are not in any team have been found in the system.		
Request error	Error processing request	400	Specific error message.

5.2.13 delete-user

This method removes a user from the system.

Name	Type	Required	Description
id	String	Yes	ID of the user to delete.

Example of use:

```
DELETE localhost:3000/api/user/delete/:id
```

Response messages:

Validation	Description	HTTP Code	Message
Deletion Successful	User deleted successfully	200	User deleted successfully.
Request error	Error processing request	400	Specific error message.

5.2.14 register-users-from-csv

This method registers users to the system from a provided CSV file.

Name	Type	Required	Description
file	CSV	Yes	CSV file with user data.

Example of use:

```
POST localhost:3000/api/user/registerFromCSV
```

Response messages:

Validation	Description	HTTP Code	Message
Registration successful	User registration completed	200	User registration completed.
Request error	Error processing request	400	Specific error message.

5.2.15 add-role

This method adds a role to an existing user in the system.

Name	Type	Required	Description
id	String	Yes	ID of the user to whom the role will be added.
role	String	Yes	Role to add.

Example of use:

```
POST localhost:3000/api/user/addRol/:id
```

Response messages:

Validation	Description	HTTP Code	Message
Successful operation	Role added	200	Role added.
Request error	Error processing request	400	Specific error message.

5.2.16 delete-role

This method removes a role from an existing user in the system.

Name	Type	Required	Description
id	String	Yes	ID of the user whose role will be removed.
role	String	Yes	Role to delete.

Example of use:

```
DELETE localhost:3000/api/user/deleteRol/:id
```

Response messages:

Validation	Description	HTTP Code	Message
Operation successful	Role removed	200	Role removed.
Request error	Error processing request	400	Specific error message.

5.3 Category

In this section the methods that involve categories will be detailed.

5.3.1 create-category

This method creates a new category with the information provided.

Name	Type	Required	Description
titleSP	String	Yes	Category title in Spanish.
titleEN	String	Yes	Category title in English.
titlePT	String	Yes	Category title in Portuguese.
descriptionSP	String	Yes	Category description in Spanish.
descriptionEN	String	Yes	Category description in English.
descriptionPT	String	Yes	Category description in Portuguese.
manualSP	File	Yes	Category manual in Spanish (PDF).
manualEN	File	Yes	Category manual in English (PDF).
manualPT	File	Yes	Category manual in Portuguese (PDF).

Example of use:

```
POST localhost:3000/api/category/createCategory
```

Response messages:

Validation	Description	HTTP Code	Message
Category created	The category has been created successfully	200	Category created successfully.
Request error	Error processing request	400	Specific error message.

5.3.2 update-category

This method updates information for an existing category.

Name	Type	Required	Description
id	String	Yes	ID of the category to update.
titleSP	String	No	New category title in Spanish.
titleEN	String	No	New category title in English.
titlePT	String	No	New category title in Portuguese.
descriptionSP	String	No	New category description in Spanish.
descriptionEN	String	No	New category description in English.
descriptionPT	String	No	New category description in Portuguese.
manualSP	File	No	New category manual in Spanish (PDF).
manualEN	File	No	New category manual in English (PDF).
manualPT	File	No	New category manual in Portuguese (PDF).

Example of use:

```
PUT localhost:3000/api/category/updateCategory/:id
```

Response messages:

Validation	Description	HTTP Code	Message
Category updated	Category has been updated successfully	200	Category updated successfully.
Request error	Error processing request	400	Specific error message.

5.3.3 get-category

This method retrieves the details of a specific category based on its ID.

Name	Type	Required	Description
id	String	Yes	ID of the category to retrieve.

Example of use:

```
GET localhost:3000/api/category/getCategory/:id
```

Response messages:

Validation	Description	HTTP Code	Message
Category found	Category details found	200	Category found successfully.
Invalid category ID	The provided category ID is invalid	400	The provided category ID is invalid.
Category not found	The category with the given ID was not found	404	That category does not exist.
Request error	Error processing request	400	Specific error message.

5.3.4 get-categories

This method retrieves all categories stored in the system.

Example of use:

```
GET localhost:3000/api/category/getCategories
```

Response messages:

Validation	Description	HTTP Code	Message
Categories found	Categories were found in the system	200	Categories were found in the system.
Request error	Error processing request	400	Specific error message.

5.3.5 delete-category

This method removes a category from the system based on its ID.

Example of use:

```
DELETE localhost:3000/api/category/deleteCategory/:id
```

Response messages:

Validation	Description	HTTP Code	Message
Category deleted	Category was successfully deleted	200	Category successfully deleted.
Request error	Error processing request	400	Specific error message.

5.3.6 get-games-by-category

This method gets all the games associated with a specific category.

Example of use:

```
GET localhost:3000/api/category/getGamesbyCategory/:id
```

Response messages:

HTTP Code	Message	Description
200	Success	Successfully obtained game list.
404	Not found	The category does not exist.

5.3.7 get-PDF

This method obtains the PDF file associated with a category in a specific language.
Example of use:

```
GET localhost:3000/api/category/getPDF/:id/:language
```

URL parameters:

Name	Type	Required	Description
id	String	Yes	Category ID.
language	String	Yes	PDF language (SP, EN, PT).

Response messages:

HTTP Code	Message	Description
200	Success	PDF file sent successfully.
400	Error	Language not supported or category not found.
404	Not found	Manual not found for the specified language.
500	Server Error	Internal server error while getting the PDF.

5.4 Team

This section will detail the methods that involve a team

5.4.1 create-team

This method creates a new computer with the information provided.
Example of use:

```
POST localhost:3000/api/team/createTeam
```

Request parameters:

Name	Type	Required	Description
studioName	String	Yes	Studio Name.
description	String	Yes	Equipment description.
gameJam	Object	Yes	Data of the associated Game Jam.
linkTree	String	Yes	Team related links.
jammers	Array	Yes	List of team members (Jammers).
site	Object	Yes	Site associated with the device.
region	Object	Yes	Region associated with the team.

Response messages:

HTTP Code	Message	Description
200	Success	Successfully created team.
400	Error	Error creating team.
403	Forbidden	The user is already assigned to a team.
404	Not found	The Game Jam, site or region does not exist.

5.4.2 update-team

This method updates information for an existing computer.

Example of use:

```
PUT localhost:3000/api/team/updateTeam/:id
```

Request parameters:

Name	Type	Required	Description
studioName	String	Yes	New studio name.
description	String	Yes	New equipment description.
gameJam	Object	Yes	New associated Game Jam.
linkTree	String	Yes	New links related to the team.
jammers	Array	Yes	New list of team members (Jammers).
site	Object	Yes	New site associated with the team.
region	Object	Yes	New region associated with the team.

Response messages:

HTTP Code	Message	Description
200	Success	Team successfully updated.
400	Error	Error updating your computer.
403	Forbidden	The user is already assigned to a different computer.
404	Not found	The device does not exist.

5.4.3 get-team

This method obtains information from a specific computer.
Example of use:

```
GET localhost:3000/api/team/getTeam/:id
```

URL parameters:

Name	Type	Required	Description
id	String	Yes	Team ID.

Response messages:

HTTP Code	Message	Description
200	Success	Equipment found correctly.
400	Error	Invalid computer ID.
404	Not found	The device does not exist.

5.4.4 get-teams

This method gets all the computers stored on the system.
Example of use:

```
GET localhost:3000/api/team/getTeams
```

Response messages:

HTTP Code	Message	Description
200	Success	Equipment found in the system.
400	Error	Error obtaining equipment.

5.4.5 delete-team

This method deletes an existing computer.
Example of use:

```
DELETE localhost:3000/api/team/deleteTeam/:id
```

URL parameters:

Name	Type	Required	Description
id	String	Yes	Team ID.

Response messages:

HTTP Code	Message	Description
200	Success	Team successfully eliminated.
400	Error	Error deleting the computer.
404	Not Found	A computer with the given ID was not found.

5.4.6 get-team-site

This method gets all computers associated with a specific site.

Example of use:

```
GET localhost:3000/api/team/getTeamSite/:site
```

URL parameters:

Name	Type	Required	Description
site	String	Yes	site ID.

Response messages:

HTTP Code	Message	Description
200	Success	Equipment found for the specified site.
400	Error	Error getting site computers.

5.4.7 add-jammer-to-team

This method adds a member (Jammer) to an existing team.

Example of use:

```
POST localhost:3000/api/team/addJammerToTeam/:teamId/:jammerId
```

URL parameters:

Name	Type	Required	Description
teamId	String	Yes	ID of the team to which the jammer will be added.
jammerId	String	Yes	ID of the jammer to be added to the team.

Response messages:

HTTP Code	Message	Description
200	Success	Jammer successfully added to the team.
400	Error	Error adding the jammer to the computer.
404	Not found	Equipment or jammer not found.
409	Conflict	The jammer already belongs to a team.

5.4.8 remove-jammer-from-team

This method removes a member (Jammer) from an existing team.

Example of use:

```
DELETE localhost:3000/api/team/removeJammerFromTeam/:teamId/:jammerId
```

URL parameters:

Name	Type	Required	Description
teamId	String	Yes	ID of the team to remove the jammer from.
jammerId	String	Yes	ID of the jammer to be removed from the computer.

Response messages:

HTTP Code	Message	Description
200	Success	Jammer successfully removed from the team.
400	Error	Error removing the jammer from the computer.
404	Not Found	Computer, jammer or jammer not found on the computer.

5.5 Gamejams

This section will detail the methods that involve a GameJam

5.5.1 create-game-jam

This method creates a new Game Jam.

Example of use:

```
POST localhost:3000/api/gamejam/createGameJam
```

Request body (JSON):

```
{
"edition": "Game Jam Edition",
"theme": {
"_id": "ID of the selected topic"
}
}
```

Response messages:

HTTP Code	Message	Description
200	Éxito	Game Jam created successfully.
400	Error	Error creating the Game Jam.
404	Not found	The specified topic does not exist.

5.5.2 update-game-jam

This method updates an existing Game Jam.

Example of use:

```
PUT localhost:3000/api/gamejam/updateGameJam/:id
```

URL parameters:

Name	Type	Required	Description
id	String	Yes	ID of the Game Jam to be updated.

Request body (JSON):

```
{
"edition": "New edition of the Game Jam",
"theme": {
"_id": "ID of the new selected topic"
}
}
```

Response messages:

HTTP Code	Message	Description
200	Success	Game Jam successfully updated.
400	Error	Error updating Game Jam.
404	Not found	The Game Jam does not exist.

5.5.3 get-game-jam

This method gets a Game Jam for your ID.

Example of use:

```
GET localhost:3000/api/gamejam/getGameJam/:id
```

URL parameters:

Name	Type	Required	Description
id	String	Yes	ID of the Game Jam you want to obtain.

Response messages:

HTTP Code	Message	Description
200	Success	Game Jam found correctly.
400	Error	Error searching for Game Jam.
404	Not found	The Game Jam does not exist.

5.5.4 get-current-game-jam

This method gets the Game Jam currently in progress.

Example of use:

```
GET localhost:3000/api/gamejam/getCurrentGameJam
```

Response messages:

HTTP Code	Message	Description
200	Hit	Current Game Jam found.
404	Not Found	No current Game Jam found.

5.5.5 get-game-jam-to-evaluate

This method obtains the Game Jam currently in the evaluation period.

Example of use:

```
GET localhost:3000/api/gamejam/getGameJamToEvaluate
```

Response messages:

HTTP Code	Message	Description
200	Success	Current Game Jam found for evaluation.
404	Not Found	No current Game Jam was found for evaluation.

5.5.6 get-game-jams

This method fetches all Game Jams on the system.

Example of use:

```
GET localhost:3000/api/gamejam/getGameJams
```

Response messages:

HTTP Code	Message	Description
200	Success	Game Jams found on the system.
400	Error	Error getting Game Jams.

5.5.7 delete-game-jam

This method deletes a Game Jam.

Example of use:

```
DELETE localhost:3000/api/gamejam/deleteGameJam/:id
```

URL parameters:

Name	Type	Required	Description
id	String	Yes	ID of the Game Jam to be deleted.

Response messages:

HTTP Code	Message	Description
200	Success	Game Jam successfully eliminated.
400	Error	Error deleting Game Jam.
404	Not found	The Game Jam does not exist.

5.5.8 get-time-remaining

This method calculates the time remaining until the start or end of the active stage of a Game Jam.

Example of use:

```
GET localhost:3000/api/gamejam/getTimeRemaining
```

Response messages:

HTTP Code	Message	Description
200	Success	Time remaining correctly calculated.
400	Error	Error calculating remaining time.

5.5.9 get-time-remaining-evaluation

This method calculates the time remaining until the start or end of the evaluation period of a Game Jam.

Example of use:

```
GET localhost:3000/api/gamejam/getTimeRemainingEvaluation
```

Response messages:

HTTP Code	Message	Description
200	Success	Time remaining correctly calculated.
400	Error	Error calculating remaining time.

5.6 Stage

This section will detail the methods that involve a stage of the gamejam.

5.6.1 create-stage

This method creates a new stage for a Game Jam.

Example of use:

```
POST localhost:3000/api/stage/createStage
```

Input parameters:

Name	Type	Required	Description
name	String	Yes	Stage name.
startDate	Date	Yes	Stage start date.
endDate	Date	Yes	End date of the stage.
startDateEvaluation	Date	Yes	Start date of the stage evaluation period.
endDateEvaluation	Date	Yes	End date of the stage evaluation period.
gameJam	Object	Yes	Object containing the ID of the associated Game Jam.

Response messages:

HTTP Code	Message	Description
200	Success	Stage created successfully.
400	Error	Error creating the stage.

5.6.2 update-stage

This method updates an existing stage of a Game Jam.

Example of use:

```
PUT localhost:3000/api/stage/:id/updateStage
```

Input parameters:

Name	Type	Required	Description
name	String	Yes	New stage name.
startDate	Date	Yes	New start date of the stage.
endDate	Date	Yes	New end date of the stage.
startDateEvaluation	Date	Yes	New start date of the stage evaluation period.
endDateEvaluation	Date	Yes	New end date of the stage evaluation period.
gameJam	Object	Yes	Object containing the ID of the associated Game Jam.

Response messages:

HTTP Code	Message	Description
200	Success	Stage updated successfully.
400	Error	Error updating stage.

5.6.3 get-current-stage

This method gets the current stage of all Game Jams.

Example of use:

```
GET localhost:3000/api/stage/getCurrentStage
```

Response messages:

HTTP Code	Message	Description
200	Success	Current stage found correctly.
400	Error	Error getting current stage.

5.6.4 get-stage

This method gets a specific stage by its ID.
Example of use:

```
GET localhost:3000/api/stage/:id/getStage
```

Route parameters:

Name	Type	Required	Description
id	String	Yes	Stage ID.

Response messages:

HTTP Code	Message	Description
200	Success	Stage found correctly.
400	Error	Error getting stage.

5.6.5 get-stages

This method gets all the stages in the system.
Example of use:

```
GET localhost:3000/api/stage/getStages
```

Response messages:

HTTP Code	Message	Description
200	Success	Stages found correctly.
400	Error	Error getting stages.

5.6.6 delete-stage

This method removes an existing stage from a Game Jam.
Example of use:

```
DELETE localhost:3000/api/stage/:id/deleteStage
```

Route parameters:

Name	Type	Required	Description
id	String	Yes	Stage ID.

Response messages:

HTTP Code	Message	Description
200	Success	Stage successfully removed.
400	Error	Error deleting stage.

5.7 Region

This section will detail the methods that involve the regions.

5.7.1 create-region

This method creates a new region.

Example of use:

```
POST localhost:3000/api/region/createRegion
```

Input parameters:

Name	Type	Required	Description
name	String	Yes	Region name.

Response messages:

HTTP Code	Message	Description
200	Success	Region created successfully.
400	Error	Error creating region.
409	Error	The region already exists.

5.7.2 update-region

This method updates an existing region.

Example of use:

```
PUT localhost:3000/api/region/:id/updateRegion
```

Input parameters:

Name	Type	Required	Description
name	String	Yes	New region name.

Response messages:

HTTP Code	Message	Description
200	Success	Region updated correctly.
400	Error	Error updating region.
409	Error	A region with that name already exists.

5.7.3 get-region

This method gets a specific region by its ID.

Example of use:

```
GET localhost:3000/api/region/:id/getRegion
```

Route parameters:

Name	Type	Required	Description
id	String	Yes	region ID.

Response messages:

HTTP Code	Message	Description
200	Success	Region found correctly.
400	Error	Error getting region.
404	Error	The region does not exist.

5.7.4 get-regions

This method gets all the regions in the system.

Example of use:

```
GET localhost:3000/api/region/getRegions
```

Response messages:

HTTP Code	Message	Description
200	Success	Regions found correctly.
400	Error	Error getting regions.

5.7.5 delete-region

This method deletes an existing region.

Example of use:

```
DELETE localhost:3000/api/region/:id/deleteRegion
```

Route parameters:

Name	Type	Required	Description
id	String	Yes	region ID.

Response messages:

HTTP Code	Message	Description
200	Success	Region successfully removed.
400	Error	Error deleting region.

5.8 Site

This section will detail the methods that involve the Site.

5.8.1 create-site

This method creates a new site.

Example of use:

```
POST localhost:3000/api/site/createSite
```

Input parameters:

Name	Type	Required	Description
name	String	Yes	Site name.
region	String	Yes	Site Region.
country	String	Yes	Country of the site.
modality	String	Yes	Site Modality.

Response messages:

HTTP Code	Message	Description
200	Success	The site has been created successfully.
400	Error	Error creating site.
409	Error	The site already exists.

5.8.2 update-site

This method updates an existing site.

Example of use:

```
PUT localhost:3000/api/site/:id/updateSite
```

Input parameters:

Name	Type	Required	Description
name	String	Yes	New site name.
region	String	Yes	New site region.
country	String	Yes	New site country.
modality	String	Yes	New site modality.

Response messages:

HTTP Code	Message	Description
200	Success	The site has been successfully updated.
400	Error	Failed to update the site.
404	Error	The site does not exist.

5.8.3 get-site

This method gets a specific site by its ID.

Example of use:

```
GET localhost:3000/api/site/:id/getSite
```

Route parameters:

Name	Type	Required	Description
id	String	Yes	Site ID.

Response messages:

HTTP Code	Message	Description
200	Success	Site found correctly.
400	Error	Error getting site.
404	Error	The site does not exist.

5.8.4 change-status

This method changes the open status of a site.

Example of use:

```
PUT localhost:3000/api/site/changeStatus
```

Response messages:

HTTP Code	Message	Description
200	Success	Site Status successfully updated.
400	Error	Error changing site status.

5.8.5 get-sites

This method gets all the sites on the system.

Example of use:

```
GET localhost:3000/api/site/getSites
```

Response messages:

HTTP Code	Message	Description
200	Success	Sites found successfully.
400	Error	Error getting the sites.

5.8.6 get-countries

This method gets all available countries.

Example of use:

```
GET localhost:3000/api/site/getCountries
```

Response messages:

HTTP Code	Message	Description
200	Success	Country data obtained correctly.
400	Error	Error getting countries.

5.8.7 get-sites-per-region

This method gets all sites in a specific region.

Example of use:

```
GET localhost:3000/api/site/:regionId/getSitesPerRegion
```

Route parameters:

Name	Type	Required	Description
regionId	String	Yes	region ID.

Response messages:

HTTP Code	Message	Description
200	Success	Sites found successfully.
400	Error	Error getting the sites.
404	Error	The region does not exist.

5.8.8 get-sites-per-region-open

This method gets all open sites in a specific region.

Example of use:

```
GET localhost:3000/api/site/:regionId/getSitesPerRegionOpen
```

Route parameters:

Name	Type	Required	Description
regionId	String	Yes	region ID.

Response messages:

HTTP Code	Message	Description
200	Success	Sites found successfully.
400	Error	Error getting the sites.
404	Error	The region does not exist.

5.8.9 delete-site

This method deletes an existing site.

Example of use:

```
DELETE localhost:3000/api/site/:id/deleteSite
```

Route parameters:

Name	Type	Required	Description
id	String	Yes	Site ID.

Response messages:

HTTP Code	Message	Description
200	Success	Site successfully deleted.
400	Error	Failed to delete site.
404	Error	The site does not exist.

5.9 Submission

This section will detail the methods that involve the delivery of equipment.

5.9.1 create-submission

This method creates a new delivery.

Example of use:

```
POST localhost:3000/api/submission/createSubmission
```

Input parameters:

Name	Type	Required	Description
description	String	Yes	Delivery description.
pitch	String	Yes	Pitch of delivery.
game	String	Yes	Delivery game.
teamId	String	Yes	delivery team ID.
categoryId	String	Yes	Category ID of the delivery.
stageId	String	Yes	Stage ID of the delivery.
themeId	String	Yes	Delivery theme ID.
title	String	Yes	Title of the delivery.

Response messages:

HTTP Code	Message	Description
200	Success	Delivery successfully created.
400	Error	Error creating delivery.

5.9.2 update-submission

This method updates an existing delivery.

Example of use:

```
PUT localhost:3000/api/submission/:id/updateSubmission
```

Input parameters:

Name	Type	Required	Description
description	String	Yes	New delivery description.
pitch	String	Yes	New delivery pitch.
game	String	Yes	New game delivery.
teamId	String	Yes	New delivery team ID.
categoryId	String	Yes	New ID of the delivery category.
themeId	String	Yes	New delivery theme ID.
stageId	String	Yes	New ID of the delivery stage.
title	String	Yes	New delivery title.

Route parameters:

Name	Type	Required	Description
id	String	Yes	delivery ID.

Response messages:

HTTP Code	Message	Description
200	Success	Delivery successfully updated.
400	Error	Error updating delivery.
404	Error	Delivery does not exist.

5.9.3 get-current-team-submission

This method obtains the current delivery of a computer.

Example of use:

```
GET localhost:3000/api/submission/:teamId/:stageId/getCurrentTeamSubmission
```

Route parameters:

Name	Type	Required	Description
teamId	String	Yes	team ID.
stageId	String	Yes	stage ID.

Response messages:

HTTP Code	Message	Description
200	Success	Delivery found correctly.
404	Error	No delivery found for the specified team and stage.

5.9.4 get-submission

This method gets a specific delivery by its ID.

Example of use:

```
GET localhost:3000/api/submission/:id/getSubmission
```

Route parameters:

Name	Type	Required	Description
id	String	Yes	delivery ID.

Response messages:

HTTP Code	Message	Description
200	Success	Delivery found correctly.
400	Error	Error getting delivery.
404	Error	Delivery does not exist.

5.9.5 get-submissions

This method gets all deliveries into the system.

Example of use:

```
GET localhost:3000/api/submission/getSubmissions
```

Response messages:

HTTP Code	Message	Description
200	Success	Deliveries found correctly.
400	Error	Error getting deliveries.

5.9.6 get-submissions-site

This method gets all deliveries to a specific site.

Example of use:

```
GET localhost:3000/api/submission/:id/getSubmissionsSite
```

Route parameters:

Name	Type	Required	Description
id	String	Yes	Site ID.

Response messages:

HTTP Code	Message	Description
200	Success	Deliveries found correctly.
400	Error	Error getting deliveries.

5.9.7 delete-submission

This method deletes an existing delivery.

Example of use:

```
DELETE localhost:3000/api/submission/:id/deleteSubmission
```

Route parameters:

Name	Type	Required	Description
id	String	Yes	delivery ID.

Response messages:

HTTP Code	Message	Description
200	Success	Delivery successfully removed.
400	Error	Error deleting delivery.
404	Error	Delivery does not exist.

5.9.8 give-rating

This method assigns a grade to a submission.

Example of use:

```
POST localhost:3000/api/submission/giveRating
```

Input parameters:

Name	Type	Required	Description
submissionId	ObjectID	Yes	submission ID.
continuityPotential	Number	Yes	Continuity Potential.
audienceCompetitorsAwareness	NumberValue	Yes	Audience and competitors awareness value.
marketPositioningValue	Number	Yes	Market Positioning Value.
gameDesignCoreLoopValue	Number	Yes	Game Design Core Loop Value.
gameDesignHookValue	Number	Yes	GameDesignHookValue.
gameDesignBalanceValue	Number	Yes	Game Design Balance Value.
artVisualsCoherenceQuality	NumberValue	Yes	Art visual quality and coherence value.
audioDesignCoherenceQuality	NumberValue	Yes	Audio Design Coherence and Quality Value.
buildQualityValue	Number	Yes	Build Quality Value.
UIUXQualityValue	Number	Yes	UI/UX Quality Value.
narrativeWorldBuildingValue	Number	Yes	Narrative WorldBuilding Value.
pitchFeedback	String	Yes	Pitch Feedback.
gameDesignFeedback	String	Yes	Game Design Feedback.
artVisualsFeedback	String	Yes	Visual Art Feedback.
audioDesignFeedback	String	Yes	Audio Design Feedback.
buildFeedback	String	Yes	Build Feedback.
personalFeedback	String	Yes	Personal Feedback.

Response messages:

HTTP Code	Message	Description
200	Success	Correctly rated game.
400	Error	Error processing game rating.

5.9.9 get-rating

This method obtains the grade assigned to a delivery by an evaluator.

Example of use:

```
GET localhost:3000/api/submission/getRating/:submissionId
```

Route parameters:

Name	Type	Required	Description
submissionId	ObjectID	Yes	submission ID.

Response messages:

HTTP Code	Message	Description
200	Success	Rating found correctly.
400	Error	Error getting grade.
404	Error	The delivery is not assigned to the current reviewer user.

5.9.10 set-evaluator-to-submission

This method assigns a rater to a delivery for evaluation.

Example of use:

```
POST localhost:3000/api/submission/setEvaluatorToSubmission
```

Input parameters:

Name	Type	Required	Description
id	ObjectID	Yes	Evaluator ID.

Response messages:

HTTP Code	Message	Description
200	Success	Evaluator correctly assigned to the delivery.
400	Error	Error assigning the tester to the delivery.
404	Error	No delivery was found available for evaluation at this stage.

5.9.11 get-submissions-evaluator

This method gets all deliverables assigned to a rater for evaluation.

Example of use:

```
GET localhost:3000/api/submission/getSubmissionsEvaluator/:id
```

Route parameters:

Name	Type	Required	Description
id	ObjectID	Yes	Evaluator ID.

Response messages:

HTTP Code	Message	Description
200	Success	Deliveries found correctly.
400	Error	Error processing the request.

5.9.12 get-ratings-evaluator

This method obtains all ratings assigned to a rater.

Example of use:

```
GET localhost:3000/api/submission/getRatingsEvaluator/:id
```

Route parameters:

Name	Type	Required	Description
id	ObjectID	Yes	Evaluator ID.

Response messages:

HTTP Code	Message	Description
200	Success	Ratings found correctly.
400	Error	Error processing the request.

5.10 Theme

This section will detail the methods that involve the themes of a gamejam.

5.10.1 create-theme

This method creates a new topic.
Example of use:

```
POST localhost:3000/api/theme/createTheme
```

Input parameters:

Name	Type	Required	Description
titleEN	String	Yes	English topic title.
titleSP	String	Yes	Title of the topic in Spanish.
titlePT	String	Yes	Theme title in Portuguese.
descriptionEN	String	Yes	Description of the topic in English.
descriptionSP	String	Yes	Description of the topic in Spanish.
descriptionPT	String	Yes	Description of the topic in Portuguese.
manualEN	File	Yes	Subject manual in English (attached file).
manualSP	File	Yes	Subject manual in Spanish (attached file).
manualPT	File	Yes	Theme manual in Portuguese (attached file).

Response messages:

HTTP Code	Message	Description
200	Success	Topic created successfully.
400	Error	Error creating theme.
409	Error	The topic already exists.

5.10.2 get-theme

This method gets a specific topic by its ID.
Example of use:

```
GET localhost:3000/api/theme/:id/getTheme
```

Route parameters:

Name	Type	Required	Description
id	String	Yes	topic ID.

Response messages:

HTTP Code	Message	Description
200	Success	Topic found correctly.
400	Error	Error getting theme.
404	Error	The topic does not exist.

5.10.3 get-themes

This method gets all the topics in the system.
Example of use:

```
GET localhost:3000/api/theme/getThemes
```

Response messages:

HTTP Code	Message	Description
200	Success	Topics successfully found.
400	Error	Error getting themes.

5.10.4 update-theme

This method updates an existing theme.
Example of use:

```
PUT localhost:3000/api/theme/:id/updateTheme
```

Route parameters:

Name	Type	Required	Description
id	String	Yes	topic ID.

Input parameters:

Name	Type	Required	Description
titleEN	String	Yes	New English theme title.
titleSP	String	Yes	New title of the song in Spanish.
titlePT	String	Yes	New topic title in Portuguese.
descriptionEN	String	Yes	New theme description in English.
descriptionSP	String	Yes	New description of the topic in Spanish.
descriptionPT	String	Yes	New topic description in Portuguese.
manualEN	File	No	New topic manual in English (optional, attached file).
manualSP	File	No	New topic manual in Spanish (optional, attached file).
manualPT	File	No	New topic manual in Portuguese (optional, attached file).

Response messages:

HTTP Code	Message	Description
200	Success	Theme successfully updated.
400	Error	Error updating theme.
404	Error	The topic does not exist.

5.10.5 delete-theme

This method deletes an existing theme.

Example of use:

```
DELETE localhost:3000/api/theme/:id/deleteTheme
```

Route parameters:

Name	Type	Required	Description
id	String	Yes	topic ID.

Response messages:

HTTP Code	Message	Description
200	Success	Topic successfully removed.
400	Error	Error deleting topic.
404	Error	The topic does not exist.

5.10.6 get-games-per-theme

This method gets all the games associated with a specific topic.

Example of use:

```
GET localhost:3000/api/theme/:id/getGamesPerTheme
```

Input parameters:

Name	Type	Required	Description
id	String	Yes	topic ID.

Response messages:

HTTP Code	Message	Description
200	Success	Games obtained correctly.
400	Error	Internal server error.

5.10.7 get-PDF

This method obtains the manual of a topic in PDF format.

Example of use:

```
GET localhost:3000/api/theme/:id/getPDF/:language
```

Input parameters:

Name	Type	Required	Description
id	String	Yes	ID of the topic from which you want to obtain the manual.
language	String	Yes	Manual language (SP, EN, PT).

Response messages:

HTTP Code	Message	Description
200	Success	Manual obtained correctly.
400	Error	Internal server error.
404	Error	Topic or manual not found.

5.11 Chat

This section will detail the methods that involve the chat.

5.11.1 create-chat

This method creates a new chat.

Example of use:

```
POST localhost:3000/api/chat/create-chat
```

Request body (JSON):

```
{"participants": [
    {
        "participantType": "Team or User",
        "participantId": "id of user or team"
    },
    {
        "participantType": "Team or User",
        "participantId": "id of user or team"
    }
]
}
```

Response messages:

HTTP Code	Message	Description
200	Success	chat created successfully.
400	Error	Error creating chat.

5.11.2 get-chat

This method obtains a chat by id.

Example of use:

```
POST localhost:3000/api/chat/get-chat/:id
```

Input parameters:

Name	Type	Required	Description
id	String	Yes	chat ID.

Response messages:

HTTP Code	Message	Description
200	Success	chat founded successfully.
400	Error	Error looking for chat.
409	Error	The chat id does not exists.

5.11.3 send-chat

This method sends a message to a chat.

Example of use:

```
POST localhost:3000/api/chat/send-chat
```

Input parameters:

Name	Type	Required	Description
id	String	Yes	chat ID.

Request body (JSON):

```
{
    "sender": {
        "Id": "id of sender (User or Team)",
        "Type": "User or Team"
    },
    "msg": "messsage to send"

}
```

Response messages:

HTTP Code	Message	Description
200	Success	message sent successfully.
400	Error	Error sending message.

5.11.4 get-chat-by-participants

This method obtains a chat by its participants.

Example of use:

```
POST localhost:3000/api/chat/get-chat-by-participants
```

Request body (JSON):

```
{
    "participantsIds" : ["id", "id"]

}
```

Response messages:

HTTP Code	Message	Description
200	Success	chat founded successfully.
400	Error	Error looking for chat.
409	Error	There in no match with the participants.

Chapter 6

Conclusions and future work

6.1 Conclusions

- When planning a Sprint, it is best to subdivide the tasks in the best possible way so that too much responsibility does not fall on one person and that all the story points are distributed appropriately. In this way, it is avoided that there are stories that include much more work than they should and thus all team members are able to work at the same pace without so many inconveniences.
- It is very important that when using the GitHub tool, you have separate branches, especially one focused on the production environment and another for deployment, since sometimes small changes in a small part of the code can greatly affect the rest, so it is best to have this separation so that there are no problems with the program that is deployed to the end user.
- When designing a system, it is best to separate it into modules as much as possible so that if changes must be made to a module, they do not end up affecting other sections of the code. Using this work with a modular approach also allows the distribution in a work team to be easier and more effective, since there are more specific tasks according to the module being worked on.
- It is recommended to configure and use a development and production environment using CI/CD, since it speeds up the integration of changes in real time. In the same way, it automates the environment configuration process each time a change is uploaded to the system, thus improving the productivity of the development team and in the production area.

6.2 Problems and limitations

As for problems, according to the established test cases, no problems were found in the system. On the other hand, the only limitation present is that for this first Sprint the graphical interface was designed to view the games by site but in the end The scope of this first phase of development did not include the profile of the jammers or the teams within the platform to upload the games, so a limitation is that the games that appear within the graphical interface are examples , you still cannot view games by site.

On the other hand, there are the platforms for which the system was designed, starting from the fact that it is using Angular, the interface is responsive and there is a wide variety of tools, there is the following:

6.3 Platforms for which the system is designed

- **Operating System:**
  - Windows 10 or later.
  - macOS 10.12 Sierra or later.
  - Linux distributions supported by modern web browsers, such as Ubuntu 21.10.
  - Mobile devices with operating systems iOS 11 or later (for iPhone and iPad) and Android 7.0 Nougat or later.
- **Minimum requirements:**
  - A modern, supported web browser, such as Google Chrome (version 70 or later), Mozilla Firefox (version 63 or later), Safari (version 11 or later), Microsoft Edge (version 70 or later), Opera (version 57 or later later), etc.
  - Internet connection to load the web page and its associated resources.
- **Recommended resolution:** The page itself is responsive, so it adapts to different resolutions but to view it in the best way on a computer, a minimum screen resolution of 1024x768 pixels is recommended to guarantee optimal viewing Of content. For mobile devices, a resolution of at least 400x561 pixels is recommended.
- **Versions used for development:**
  - Angular: 17.0
  - MongoDB: 7.0
  - NodeJs: 20.11.1
  - NPM: 10.5.0

6.4 Future work

In this case, this is the last Sprint that this development team executes, so the future work are recommendations that are provided to developers who will continue maintaining this system. Among the main points are the following:

- In the case of this system, chat is presented as the form of user communication. However, it is worth highlighting the fact that this can be broken down and separated further, by having a separate notification system so that you do not necessarily depend on an organizer, for example, to send the message but rather this is handled automatically.
- It is necessary to polish the graphical interface issues, since in some parts it is not completely responsive, so additional refactoring is needed in the graphic section so that it is completely adaptable for mobile devices and thus can reach a greater Number of users.
- An integration of the external platform of the store must be done with the system that was created since the development of the store was omitted because the end user said that this was being worked on separately, so it is necessary to Talk to the client to find out how the integration should be done and what changes should be made to the system.
- Given that the company for which the system is developed does not speak native Spanish, it is best that the rest of the documentation that continues to be generated as well as the program continues to be all in English, so that it does not have to be generated in two different languages everything and the development time can be used in a better way.

Referencias bibliográficas

Bello, F. (2021, Noviembre 22). *Instalando angular 13.* Retrieved from https://fbellod.medium.com/instalando-angular-13-78f162e89dd

Hernandez, R. (2021, Junio 28). *El patrón modelo-vista-controlador: Arquitectura y frameworks explicados.* Retrieved from https://www.freecodecamp.org/espanol/news/el-modelo-de-arquitectura-view-controller-pattern/

Jethva, H. (2022, Junio 15). *How to install node.js and npm on oracle linux.* Retrieved from https://www.atlantic.net/dedicated-server-hosting/how-to-install-node-js-and-npm-on-oracle-linux/

MongoDB. (n.d.-a). *Install mongodb community edition on red hat or centos.* Retrieved from https://www.mongodb.com/docs/manual/tutorial/install-mongodb-on-red-hat/#footnote-oracle-linux

MongoDB. (n.d.-b). *What is the mean stack?* Retrieved from https://www.mongodb.com/mean-stack

Oracle. (2023, octubre 19). *Comience con git en oracle linux.* Retrieved from https://docs.oracle.com/en/learn/ol-git-start/index.html#install-git-on-oracle-linux