UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

DESARROLLO DE UNA APLICACIÓN WEB PARA MODELADO COLABORATIVO

Andrés Navarro Blanco
Tutor: Sara Pérez Soler
Ponente: Esther Guerra Sánchez

JULIO 2019

DESARROLLO DE UNA APLICACIÓN WEB PARA MODELADO COLABORATIVO

AUTOR: ANDRÉS NAVARRO BLANCO
TUTOR: SARA PÉREZ SOLER
PONENTE: ESTHER GUERRRA SÁNCHEZ

Escuela Politécnica Superior
Universidad Autónoma de Madrid
JULIO de 2019

Resumen (castellano)

Este Trabajo de Fin de Grado (en adelante TFG), consiste en la creación de una aplicación web para modelado colaborativo, para el chatbot *SOCIO*, que ya funciona en la aplicación de mensajería instantánea *Telegram* y la red social de microblogging *Twitter*. Este chatbot permite, mediante lenguaje natural, realizar tareas de modelado en diversos entornos con múltiples usuarios para el tratamiento y diseño de modelos.

Para ello, se desarrollará una aplicación (*SOCIO-APP*) que adapte esta misma funcionalidad en un entorno web y a la vez proporcione un ambiente menos verboso y más intuitivo visualmente para el usuario a la hora de hacer uso del bot.

Los chatbots, a menudo, al ser mayormente utilizados en redes sociales o servicios de mensajería instantánea pecan de un sobreuso de comandos escritos que, para un usuario no especializado, puede llegar a resultar extenuante/reiterativo.

SOCIO-APP se encargará de que esta experiencia sea mucho más amena tanto para usuarios expertos como para usuarios no acostumbrados a tratar con chatbots, proponiendo una interfaz de usuario agradable e intuitiva, para facilitar y mejorar en todo momento el arte de desarrollar un modelo de software de forma conjunta.

Con relación a posibilitar estos objetivos, *SOCIO-APP* contará con un servicio de mensajería instantánea grupal, un sistema para guardar todo el histórico de mensajes de la aplicación, así como una red de usuarios privada con perfiles modificables, para dar una mayor sensación de naturalidad a la tarea que se desea conseguir con este proyecto, proporcionar mecanismos para la discusión y coordinación en la gestión de modelos.

Está aplicación ha sido implementada con *Django*, un framework de *Python* a alto nivel para el desarrollo de aplicaciones web, con una base de datos *SQLite* y se encuentra totalmente integrada con todos los módulos que tienen tratamiento con *SOCIO*, es decir, *Telegram* y *Twitter*, para que cualquier usuario del bot, independientemente de la plataforma pueda acceder a cualquier proyecto. Para la realización de este proyecto se ha partido de una *API REST* existente, la cual proporciona la funcionalidad para la creación y gestión de los modelos mediante llamadas y peticiones al bot *SOCIO*, que a su vez se comunica con *PlantUML* para la generación gráfica de los modelos.

En conclusión, está aplicación pretende mejorar la experiencia del usuario a la hora de desarrollar modelos simultáneamente, de una forma totalmente innovadora y hasta ahora nunca vista en el mundo del desarrollo colaborativo de productos software.

Abstract (English)

This final degree's dissertation (from now onwards TFG-standing for the Spanish Trabajo de Fin de Grado) consists in the creation of a web application for a collaborative model, for the *SOCIO* chatbot, which is now working in the messenger server for instant messaging *Telegram* and the microblogging social network *Twitter*. This chatbot allows, by dint of natural language, to perform modeling tasks in several settings with multiple users for the treatment and design of models.

To that end, the application (*SOCIO-APP*) will be developed with the aim of adapting this same functionality to a web context that provides at the same time a less wordy setting that is on top of that more visually intuitive for the user to make us of the aforementioned bot.

Chatbots are mainly used in social networks or instant texting services and they often err, precisely because of that use, on the side of the overuse of the written commands which can be seen as strenous for the non-specialized user and even for the ones with more expertice.

SOCIO- APP will count on a group instant texting service, a system to keep all the history of the application messages as well as a net of users with changeable profiles, to provide a bigger feeling of naturality to the prospective task that is supposed to be attainable with this Project, that is, providing discussion and coordination mechanisms for the management of models.

This application has been implemented with *Django*, a *Python* framework at a high level to develop web applications, with a database *SQLite* and it is found fully integrated in all the modules that are related to *SOCIO*, that is to say, *Telegram* and *Twitter*, for any bot user to access to any project no matter the platform they are using. For the creation of this project I have taken as a starting point an existant *API REST* which supplies the funcionality for the creation and management of the models through calls and requests to the *SOCIO* bot, which communicates in turn with *PlantUML* for the graphic generation of the models.

In conclusion, this application intends to improve the user experience when developing models simultaneously in a totally innovative way never seen up to now in the collaborative development of software products area.

Palabras clave (castellano)

Aplicación web, modelado colaborativo, chatbot, lenguaje natural.

Keywords (inglés)

Web application, collaborative modeling, chatbots, natural language

Agradecimientos

En primer lugar, quisiera agradecer a mi tutora Sara Pérez Soler, por darme la oportunidad de realizar este proyecto, con el cual, a pesar de la presión de realizar un TFG, he disfrutado mucho.

En segundo lugar, agradecer a todos mis compañeros y amigos de la carrera, por su apoyo y compañía, tanto en los buenos como en los malos momentos, sin duda alguna lo mejor que uno puede llevarse de su paso por la escuela.

En tercer lugar quisiera darle un millón de gracias a mi familia, por su apoyo incondicional en todos los aspectos posibles de la vida.

Por último, aunque no por ello menos importante, quisiera darle las gracias a mi novia por creer en mí, pues sin su apoyo, llegar hasta aquí habría sido imposible.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

1 Introducción

En esta sección haremos una breve introducción al contenido de este TFG, hablando sobre la motivación para realizar este trabajo (apartado 1.1), de los objetivos a cumplir durante su creación (apartado 1.2) y de la organización del documento (apartado 1.3).

1.1 Motivación

El auge de las tecnologías web ha supuesto un gran avance en cuanto a las comunicaciones a nivel global. Este progreso se ha visto acompañado con la aparición de herramientas y mecanismos de comunicación y discusión como las aplicaciones de mensajería instantánea, las redes sociales, etc., así como desarrollo de herramientas software más eficientes y ajustadas a los tiempos en los que vivimos, en los que la necesidad de comunicación entre personas va mucho más allá de enviar correos electrónicos o mantener una conversación cara a cara. La continua evolución de las ciencias informáticas conlleva una constante aparición de nuevas ideas para mejorar las aplicaciones existentes y convertirlas en productos que se amolden perfectamente al usuario actual.

En este entorno surgió la idea de *SOCIO* [1], un chatbot para redes sociales gracias al cual, usuarios de todo tipo y lugar pueden asociarse para realizar desarrollos colaborativos de modelos software. Este bot se utiliza en las redes sociales y aplicaciones de mensajería *Twitter* y *Telegram*, siguiendo una serie de comandos variables según la aplicación desde la que se solicite su uso. El bot va creando un modelo gráfico interpretando los requisitos que los usuarios le especifican mediante lenguaje natural.

Llegados a este punto, surge de nuevo la necesidad de seguir avanzando en busca de satisfacer lo mejor posible las necesidades de los usuarios. En el entorno de las redes sociales la utilización de bots está muy normalizada y su uso es constante hoy en día, como podemos ver con los socialbots, chatbots, etc. En el caso particular del bot *SOCIO* su uso en redes sociales y herramientas de mensajería instantánea supone un gran avance en el mundo del modelado colaborativo, pero dado el entorno no especializado en el modelado colaborativo de estas aplicaciones, su potencial se ve bastante limitado.

Esta limitación tiene que ver, tanto con la obligación de tener que usar constantemente los comandos de cada red social para realizar cualquier acción, ya sea de edición para obtener cualquier tipo de información, como por la falta de una interfaz gráfica adecuada que acompañe al uso del bot. La interfaz gráfica facilitaría cumplir el objetivo principal de *SOCIO*, que es llegar a una gran cantidad de usuarios sin tener en cuenta la especialización de estos en el mundo del modelado y que estos puedan fácil y cómodamente colaborar en el diseño de modelos conjuntamente.

Desde el punto de vista de la utilización del chatbot *SOCIO* en las redes sociales, la facilidad de uso es excelente, y es una gran simplificación de la ardua tarea de crear un modelo de forma colaborativa, pero en cuanto a la comodidad del usuario para el acceso a los datos generados por el bot y la gestión de estos, es muy mejorable, así como el hecho de aprovechar todo el potencial latente que puede llegar a abarcar esta herramienta.

Así surge la idea de desarrollar la aplicación *SOCIO-APP*, una aplicación web creada única y exclusivamente para sacar la máxima capacidad de este bot de modelado

colaborativo y poder así brindar al usuario una experiencia realmente satisfactoria y completa, cumplimentando esta falta de comodidad e incluso de facilidad (ahorrando en el uso de comandos), de la que peca el uso del bot en redes sociales y herramientas de mensajería instantánea no especializadas.

1.2 Objetivos

El objetivo principal de este TFG es el análisis, diseño y desarrollo de una aplicación web de modelado colaborativo online que se integre con el bot *SOCIO* y con todos sus módulos, para conseguir una herramienta de uso sencillo y agradable para la mayoría de los usuarios sin tener en cuenta su experiencia en el mundo del modelado.

Los objetivos por cumplimentar serán los siguientes:

- Desarrollo de la lógica de negocio para usuarios, estándar en las aplicaciones web de hoy en día (registro de usuarios, perfiles modificables, login, etc.).
- Creación de una interfaz gráfica de usuario (GUI), intuitiva y fácil de utilizar para cualquier tipo de usuario, con todas las funcionalidades ofrecidas por el bot integradas en ella.
- Implementación de un chat en tiempo real para múltiples usuarios, con capacidad para ejecutar todas las posibles operaciones de modelado que ofrece *SOCIO*.
- Establecer comunicaciones con el bot *SOCIO* y representar sus respuestas en una interfaz gráfica de una manera agradable y sencilla.
- Persistencia de los datos que maneje la aplicación (usuarios, proyectos, mensajes, etc.).
- Mostrar un historial de los últimos mensajes recibidos por proyecto, teniendo especialmente en cuenta los mensajes que hayan realizado acciones de modificación del modelado en algún proyecto.
- Integración de la aplicación web con el resto de las herramientas que hacen uso de *SOCIO*, como son *Telegram* y *Twitter*.

1.3 Organización de la memoria

El contenido de esta memoria será el expuesto a continuación:

- **Estado del arte** (Sección 2)**.** En esta sección hablaremos sobre las diferentes herramientas que nos encontramos hoy en día para el diseño colaborativo de software y otros productos y las herramientas para el modelado colaborativo.
- **Análisis y diseño** (Sección 3)**.** Se lleva a cabo un análisis de los requisitos y funcionalidades para implementar de manera correcta la aplicación y se presenta el diseño realizado para cumplir esta tarea.
- **Desarrollo** (Sección 4)**.** Se expondrán todas las tecnologías utilizadas en el desarrollo del proyecto y como han sido utilizadas para lograr los objetivos de este TFG.

- **Herramienta** (Sección 5)**.** En este apartado, se hará una introducción a la herramienta y sus usos.

- **Conclusiones y trabajo futuro** (Sección 6)**.** Esta sección contendrá las conclusiones sacadas de la realización de este proyecto, así como las posibles derivaciones y mejoras que se puedan llevar a cabo en el futuro.

2 Estado del arte

El software colaborativo (también conocido como *groupware*), es un concepto de trabajo que surgió a finales de la década de 1980.

En 1987, Louis S. Richman y Julianne Slovak escribieron su artículo *"Software catches the team spirit"* [2], en el que describían al software colaborativo como un nexo electrónico de unión entre los trabajadores dedicados al desarrollo de software. También sugerían que el denominado *groupware* aspiraba con revolucionar la manera de trabajar de los equipos, utilizando el software como elemento central de la comunicación y así mejorar notablemente la eficiencia y calidad, tanto del proceso de desarrollo, como de los productos finales.

A continuación, hablaremos sobre las herramientas más significativas hoy en día para trabajar en entornos colaborativos relacionadas con el desarrollo de proyectos (Sección 2.1) y de las herramientas especializadas en el modelado colaborativo (Sección 2.2).

2.1 Herramientas con entornos colaborativos

Los entornos colaborativos son básicos para el desarrollo de proyectos, sin tener en cuenta el ámbito en el que se quiera desarrollar el mismo. El poder sumar conocimientos con cualquier usuario supone una más que probable mejora del trabajo realizado, además de posibilitar el acceso a estas tareas a usuarios no especializados en la misma. En este apartado se presentan cinco herramientas para desarrollar software de forma colaborativa.

- *Slack* [3]: es una aplicación de comunicación y colaboración en equipo para llevar a cabo proyectos de cualquier índole. Esta divide las conversaciones en grupos o canales específicos para cada tarea que se quiera realizar, guardando el historial completo de las mismas para cuando se quiera consultar. Una de sus mayores virtudes reside en su alta capacidad de integración con cualquier herramienta.

- *GitHub* [4]: es una plataforma de desarrollo colaborativo de proyectos software con control de versiones de forma mayoritariamente pública y gratuita. En esta plataforma múltiples usuarios pueden trabajar de forma paralela en un mismo proyecto, desarrollando diferentes ramas de un proyecto raíz para luego fusionar el trabajo realizado por cada uno de los usuarios.

- *Google Docs* [5]: es una herramienta para crear documentos de forma simultánea con otros usuarios en línea de una manera sencilla. Además, incluye un chat de mensajería instantánea y un servicio de correo electrónico incorporado, para ayudar a la comunicación e interacción entre los usuarios.

- *Asana* [6]: es una herramienta para el seguimiento y organización de trabajadores de proyectos de desarrollo de software, la cual fue diseñada para optimizar al máximo el trabajo de los miembros de un equipo. En esta plataforma se pueden crear esquemas con el trabajo futuro a realizar, proyectos en marcha, poner recordatorios para fechas límite de entregas, enviar peticiones a tus compañeros y comentar los posts que se publican dentro de la aplicación. También tiene un potente motor de búsqueda para poder encontrar cualquier trabajo realizado anteriormente.

- *MockPlus iDoc* [7]: es una herramienta para el desarrollo de proyectos de diseño colaborativo a nivel de front-end. Está da al usuario un entorno perfecto para las necesidades de un desarrollador o diseñador gráfico de plantillas HTML, que además se encuentra interconectado con todo el equipo que vaya a trabajar en el proyecto.

Como podemos observar, las herramientas *Slack, Google Docs y Asana* dan una gran importancia al uso de chats internos para la comunicación entre los usuarios, puesto que esto es primordial en el desarrollo colaborativo de cualquier producto. En las demás herramientas (*MockPlus iDoc y Github*), sustituyen los chats internos, por un ambiente totalmente colaborativo.

También podemos ver cómo es clave que la herramienta se adapte en fondo y forma a las necesidades de los usuarios que van a hacer uso de ella, por lo que las interfaces de usuario son un elemento crucial a la hora de mejorar la experiencia del desarrollo simultáneo.

2.2 Herramientas para modelado colaborativo

El modelado colaborativo supone un gran avance en lo que al modelado de software se refiere. La tarea de realizar un modelado supone, desde las fases más tempranas del desarrollo de un producto software, un elemento crucial, que no puede recaer en manos de expertos en una sola materia. El poder fusionar múltiples ideas de varios usuarios, cada uno con su punto de vista y formación, enriquece mucho el producto final a conseguir y además ameniza el proceso de creación. En este apartado se presentan cinco herramientas para modelado colaborativo online.

- *Visual Paradigm* [8]: es una herramienta online para el desarrollo de modelos. Tiene una tecnología de repositorios propios, en los que se puede versionar y ramificar proyectos, así como realizar comparaciones y ver los cambios acometidos a lo largo del desarrollo. Tiene una plataforma gráfica para el diseño de diagramas de modelado con la posibilidad de comentar en un sistema dechaten los mismos posibles errores cometidos o discutir posibles mejoras.

- *Gliffy* [9]: es un software online para crear cualquier tipo de diagrama UML, ya sean diagramas de flujo, diagramas de casos de uso, etc. Tiene una interfaz conformada por un panel al que se pueden arrastrar los distintos elementos que constituyen el diagrama a realizar, así como diferentes plantillas base. Estos diagramas se pueden compartir vía enlace con diferentes permisos de usuario como ver, editar y comentar. Para finalizar esta totalmente integrado con las herramientas *Jira* [10] (herramienta para la administración de tareas) y *Atlassian* (empresa desarrolladora de software con múltiples productos) [11] para tener así comunicación y colaboración total con los miembros del equipo de desarrollo.

- *GenMyModel* [12]: es una plataforma de modelado online con editores gráficos los cuales se pueden usar de manera simultánea invitando a los usuarios que quieras para colaborar. En estos editores online existe la posibilidad de abrir un chat de proyecto para poder estar en contacto con los otros usuarios de la herramienta. Tiene un repositorio de modelos centralizado para subir los proyectos y una interfaz intuitiva.

- *Cacoo* [13]: es un software para el desarrollo de modelos online. Gracias a su poderoso editor gráfico puede crear diagramas de flujo, diagramas de red, diagramas de bases de datos y también tiene la posibilidad de realizar esquemas

de seguimiento de proyectos para poder optimizar el trabajo realizado. También se puede añadir mensajes en los editores gráficos para comentar el trabajo que se esté realizando. Tiene una alta capacidad de integración con cualquier herramienta.

- *Creately* [14]: es un editor gráfico de modelos online, se utiliza para la creación, de una manera simple, de modelos y estructuras de bases de datos, con la posibilidad de invitar a colaboradores al proyecto y poder compartir con ellos las ideas de desarrollo y que ellos puedan ayudarte a mejorar el diseño del modelo.

Como podemos observar, todas estas herramientas de modelado colaborativo online se centran en un gestor gráfico de modelos. Este punto, como se ha expuesto en apartados anteriores limita el tipo de usuario que va a poder hacer uso de la herramienta dado que este debe tener unos conocimientos básicos de modelado para poder realizarlo de manera correcta. Por otra parte, el uso de servicios de mensajería o el envío de comentarios relacionados con los proyectos, supone la piedra angular de cualquier herramienta de modelado colaborativo, como podemos apreciar con estos cinco ejemplos, los cuales poseen esta clase de servicios o similares.

3 Análisis y Diseño

3.1 Análisis de la aplicación

A continuación, se presenta el análisis elaborado para la herramienta *SOCIO-APP*. En esta sección, se expondrá el alcance de la aplicación (apartado 3.1.1) y se realizará un análisis de requisitos, tanto funcionales (apartado 3.1.2) como no funcionales (apartado 3.1.3), todo ello desglosado en los diferentes subsistemas que componen la aplicación final, los cuales son: subsistema de usuarios, subsistema de proyectos, subsistema de acciones (relacionado con la comunicación con el chatbot *SOCIO*) y el subsistema de mensajería.

3.1.1 Alcance

Está aplicación tiene como objetivo el facilitar el diseño colaborativo de modelados mediante la ayuda de un chatbot, para ello se necesitará acoplar al funcionamiento principal del bot, además de aportar a los usuarios herramientas que faciliten y hagan atractivo su uso. A continuación, procederemos a ver el alcance de la aplicación en cada uno de los diferentes subsistemas que la componen.

En primer lugar, consideraremos el **subsistema de usuarios**:

- Los usuarios podrán registrarse y entrar en el sistema con un nombre de usuario y contraseña.
- Los usuarios podrán editar los datos de su perfil, nombre, correo electrónico y foto de perfil.
- Se permitirá a usuarios de otras plataformas el acceso a los proyectos propios de la aplicación y viceversa.

En segundo lugar, analizaremos el alcance del **subsistema de proyectos**:

- Crear nuevos proyectos, con sus salas de chat correspondientes, indicando el nombre y la visibilidad de este. La visibilidad puede ser pública (usuarios con permisos de escritura y lectura), protegida (todos los usuarios con permisos de lectura, pero permiso de escritura restringido) y privada (usuarios con restricciones en permisos de escritura y lectura).
- Modificar la visibilidad de los proyectos.
- Eliminar proyectos.
- Crear vínculos con proyectos externos de otras aplicaciones que usen *SOCIO*.
- Eliminar los vínculos a proyectos externos.

En tercer lugar, veremos el **subsistema de acciones,** que es el encargado principal de las comunicaciones con la API de *SOCIO*. Este subsistema se divide en diferentes módulos:

- **Configuración**
  - Crear, eliminar, modificar proyectos cuando lo indique la acción del subsistema de proyectos.

    - Validar el estado de modelado de un proyecto.
    - Dar, eliminar y modificar permisos de lectura y escritura a usuarios.
    - Crear ramas (branches) de proyectos existentes, asociados a un grupo de organización.
    - Cambiar el grupo de organización de las ramas (branches).
- **Edición**
    - Realizar acciones de modelado en proyectos mediante lenguaje natural.
    - Deshacer y rehacer acciones de modelado en proyectos.
- **Información**
    - Obtener todos los proyectos de *SOCIO*.
    - Obtener los proyectos de un usuario.
    - Obtener los proyectos en los que un usuario puede leer o escribir.
    - Obtener los proyectos con cierta visibilidad, pública o privada.
- **Estadísticas**
    - Obtener el número de acciones realizadas sobre un proyecto.
    - Obtener el número de acciones por usuario realizadas sobre un proyecto.
    - Obtener el número de mensajes por usuarios enviados en un proyecto.
    - Obtener el porcentaje de autoría de cada usuario en un proyecto.
- **Historial**
    - Acciones de modelado realizadas sobre un proyecto.
    - Acciones de gestión de un proyecto.
- **Obtener el modelo**
- **Obtener ayuda**
- **Obtener la imagen del proyecto**
- **Actualizaciones**
    - Registrar el canal de la aplicación.
    - Obtener todos los proyectos que han sido actualizados.
    - Obtener la imagen con la última actualización del proyecto.

Y en cuarto y último lugar, hablaremos del **subsistema de mensajería:**

- Los mensajes enviados en los proyectos se mostrarán en tiempo real.

- Estos mensajes deberán ser diferenciados entre mensajes simples o mensajes dirigidos al bot *SOCIO*.
- Habrá un historial de mensajes, que se actualizará con cada mensaje enviado. En este historial se tendrán especialmente en cuenta los mensajes que modifiquen alguno de los modelos de los proyectos.

3.1.2 Requisitos funcionales

En este apartado veremos el análisis de requisitos funcionales realizado para la aplicación separados por los subsistemas vistos anteriormente.

Subsistema de usuarios

1. **Registrar un usuario**: Se permitirá a los nuevos usuarios registrarse en la aplicación.

   Entrada: El nuevo usuario utilizará la plataforma de registro introduciendo sus datos, nombre de usuario, contraseña y correo electrónico.

   Proceso: El sistema comprobará que los datos introducidos son correctos, si es así procederá a dar de alta al usuario en la aplicación.

   Salida: Si los datos son correctos, el sistema redirigirá al usuario a la plataforma de login desde la cual tendrá que introducir sus credenciales. Si los datos no son correctos se informará al usuario en el mismo formulario de registro.

2. **Login de usuario**: Se permitirá al usuario entrar en la aplicación.

   Entrada: El usuario introduce su nombre y contraseña

   Proceso: El sistema comprueba que las credenciales son correctas, de ser así, el usuario entrará en la aplicación.

   Salida: Si los datos introducidos son correctos, se redirigirá al usuario a la página de inicio de la aplicación, en caso contrario se informará al usuario, en el mismo formulario de login de que sus credenciales no son correctas.

3. **Modificar datos de perfil**: Se permitirá al usuario modificar los datos de su perfil.

   Entrada: El usuario introduce modificaciones en los datos de su perfil.

   Proceso: El sistema comprobará los nuevos datos modificados, si no contienen ningún error, los modificará en la base de datos.

   Salida: Si los datos son correctos, el sistema mostrará de nuevo el perfil del usuario con los datos modificados, de no serlos indicará un mensaje de error.

Subsistema de proyectos

4. **Crear un proyecto**: El usuario podrá crear nuevos proyectos.

   Entrada: El usuario introducirá el nombre del proyecto y la visibilidad que desea.

Proceso: El sistema comprobará que los datos son correctos, si es así creará el proyecto en la base de datos y se enviará la petición al bot *SOCIO* para crear el proyecto.

Salida: Si los datos introducidos son correctos, el sistema redirige al usuario a la sala de chat creada para el proyecto, De no ser correctos el sistema mostrará el error. Si hubiera algún error en la petición a la API se mostraría un mensaje de error.

5. **Modificar un proyecto**: El usuario podrá modificar la visibilidad de un proyecto, siempre y cuando sea el administrador.

   Entrada: El usuario selecciona la opción de modificar proyecto, se le redirige a la página para asegurar que desea modificarlo y selecciona la opción afirmativa.

   Proceso: El sistema comprueba que el usuario es el administrador del proyecto, de ser así modifica el proyecto en la base de datos y se envía la petición de modificación a *SOCIO.*

   Salida: Si la modificación se ha realizado correctamente, se redirigirá al usuario a la página de selección de proyectos. Si hubiera algún error en la petición a la API se mostraría un mensaje de error.

6. **Eliminar un proyecto**: El usuario podrá eliminar proyectos, siempre y cuando sea el administrador.

   Entrada: El usuario selección la opción de eliminar un proyecto, se le redirige a la página para asegurar que desea eliminar el proyecto y selecciona la opción afirmativa.

   Proceso: El sistema comprueba que el usuario es el administrador del proyecto, de ser así elimina el proyecto de la base de datos y se envía la petición de eliminación a *SOCIO.*

   Salida: Si la eliminación se ha realizado correctamente, se redirigirá al usuario a la página de selección de proyectos. Si hubiera algún error en la petición a la API se mostraría un mensaje de error.

7. **Crear vínculo a proyecto externo**: El usuario podrá crear vínculos a proyectos externos a la aplicación.

   Entrada: El usuario introduce el nombre y el canal del proyecto externo al que desea acceder.

   Proceso: El sistema comprueba que ese proyecto existe con ese nombre y con el canal especificado por el usuario mediante una petición a *SOCIO*, si existiera crea un proyecto en base de datos como vinculo al proyecto externo.

   Salida: Si el vínculo ha sido creado de manera correcta, se redirigirá al usuario a la sala de chat creada para el proyecto. Si el proyecto no existiera se mostraría un mensaje de error.

8. **Eliminar vínculo a proyecto externo**: El usuario podrá eliminar vínculos a proyectos externos a la aplicación.

Entrada: El usuario selección la opción de eliminar el vínculo al proyecto externo, se le redirige a la página para asegurar que desea eliminar el vínculo de proyecto y selecciona la opción afirmativa.

Proceso: El sistema comprueba que el usuario es el administrador del proyecto dentro de la aplicación, de ser así elimina el proyecto de la base de datos.

Salida: Si la eliminación se ha realizado correctamente, se redirigirá al usuario a la página de selección de proyectos.

Subsistema de acciones

9. **Validar un proyecto**: La aplicación mostrará la evaluación hecha sobre un proyecto.

   Entrada: El usuario selecciona la opción de validar proyecto en el chat de proyecto.

   Proceso: El sistema envía la petición correspondiente a *SOCIO*.

   Salida: Se mostrará un mensaje con la evaluación. Si hubiera algún error en la petición a la API se mostraría un mensaje de error.

10. **Dar permisos a usuario**: El usuario podrá dar permisos de lectura o escritura a un usuario, siempre y cuando sea el administrador del chat.

    Entrada: El usuario selecciona la opción de dar permisos a usuarios en el chat de proyecto. A continuación, indicará en el formulario el nombre de usuario, el tipo de permiso y el canal al que pertenece el usuario al que se desea dar permiso.

    Proceso: El sistema envía la petición correspondiente a *SOCIO*.

    Salida: Se mostrará un mensaje de aceptación de solicitud de permisos de usuario. Si hubiera algún error en la petición a la API se mostraría un mensaje de error.

11. **Eliminar permisos a usuario**: El usuario podrá eliminar permisos a otro usuario, siempre y cuando sea el administrador del chat.

    Entrada: El usuario selecciona la opción de eliminar permisos a un usuario en el chat de proyecto. A continuación, indicará en el formulario el nombre de usuario y el canal al que pertenece el usuario al que se desea eliminar los permisos.

    Proceso: El sistema envía la petición correspondiente a *SOCIO*.

    Salida: Se mostrará un mensaje con la correcta eliminación de permisos de usuario. Si hubiera algún error en la petición a la API se mostraría un mensaje de error.

12. **Cambiar permisos a usuario**: El usuario podrá cambiar permisos a un usuario, siempre y cuando sea el administrador del chat.

    Entrada: El usuario selecciona la opción de cambiar los permisos a un usuario en el chat de proyecto. A continuación, indicará en el formulario el nombre de usuario y el canal al que pertenece el usuario al que se desea modificar los permisos.

Proceso: El sistema envía la petición correspondiente a *SOCIO*.

Salida: Se mostrará un mensaje con la correcta modificación de permisos de usuario. Si hubiera algún error en la petición a la API se mostraría un mensaje de error.

13. **Crear un branch**: El usuario podrá crear distintas ramas de un proyecto.

    Entrada: El usuario selecciona la opción de crear un branch. A continuación, indicará en el formulario el nombre que desea darle al nuevo branch.

    Proceso: El sistema envía la petición correspondiente a *SOCIO*, si resulta satisfactoria, se creará un proyecto de tipo branch en la base de datos.

    Salida: Si la petición ha sido exitosa se redirigirá al usuario a la sala de chat para el nuevo branch. Si hubiera algún error en la petición a la API se mostraría un mensaje de error.

14. **Cambiar el grupo de un branch**: El usuario podrá cambiar el grupo al que pertenezca un branch.

    Entrada: El usuario, dentro del chat del branch, selecciona la opción de cambiar el grupo de un branch. A continuación, indicará en el formulario el nombre del grupo al que desea que pertenezca branch.

    Proceso: El sistema envía la petición correspondiente a *SOCIO*.

    Salida: Se mostrará un mensaje indicando la correcta modificación del grupo del branch. Si hubiera algún error en la petición a la API se mostraría un mensaje de error.

15. **Realizar acción de modelado en el proyecto**: Se podrán realizar acciones de modelado dentro del chat de proyecto.

    Entrada: El usuario selecciona la opción para comunicarse con *SOCIO*, introduce el mensaje con la acción de modelado que desee realizar y lo envía.

    Proceso: El sistema envía la petición correspondiente a *SOCIO*, se almacena el mensaje en la base de datos.

    Salida: Si la petición se ha realizado con éxito se muestra la imagen con el modelo actualizado en el chat. Si hubiera algún error en la petición a la API se mostraría un mensaje de error.

16. **Deshacer o rehacer acción de modelado en el proyecto**: Se podrán deshacer o rehacer acciones de modelado dentro del chat de proyecto.

    Entrada: El usuario selecciona la opción de rehacer o deshacer la acción de modelado.

    Proceso: El sistema envía la petición correspondiente a *SOCIO*.

    Salida: Si la petición se ha realizado con éxito se muestra la imagen con el estado modificado del modelo. Si hubiera algún error en la petición a la API se mostraría un mensaje de error.

17. **Obtener los proyectos**: Se podrán obtener todos los proyectos pertenecientes a *SOCIO*.

    E ntrada: El usuario selecciona la opción de obtener todos los proyectos relacionados con *SOCIO*.

    Proceso: El sistema envía la petición correspondiente a *SOCIO*.

    Salida: Si la petición se ha realizado con éxito se muestran todos los proyectos. Si hubiera algún error en la petición a la API se mostraría un mensaje de error.

18. **Obtener proyectos por usuario, visibilidad y permisos**: Se podrán obtener los proyectos por usuario, los proyectos dependiendo de la visibilidad y tipos de permisos que tengan los usuarios.

    Entrada: El usuario selecciona la opción de visualizar proyectos, indicando el usuario y canal al que pertenezca si es el caso de búsqueda por usuario, la visibilidad si es el caso de búsqueda por tipo de visibilidad, o el nombre de usuario, canal y tipo de permiso para la búsqueda por permisos de usuario.

    Proceso: El sistema envía la petición correspondiente a *SOCIO*.

    S alida: Si la petición se ha realizado con éxito se muestran los proyectos indicados en los filtros. Si hubiera algún error en la petición a la API se mostraría un mensaje de error.

19. **Obtener información de un proyecto**: Se podrá obtener la información de un proyecto.

    Entrada: El usuario selecciona la opción de obtener información.

    Proceso: El sistema envía la petición correspondiente a *SOCIO*.

    Salida: Si la petición se ha realizado con éxito se muestra la información del proyecto. Si hubiera algún error en la petición a la API se mostraría un mensaje de error.

20. **Obtener las acciones totales de un proyecto:** Se podrán obtener todas las acciones realizadas sobre un proyecto.

    Entrada: El usuario selecciona la opción de mostrar estadísticas totales de un proyecto.

    Proceso: El sistema envía la petición correspondiente a *SOCIO*.

    Salida: Si la petición se ha realizado con éxito se muestra en el chat la imagen de las estadísticas totales de un proyecto. Si hubiera algún error en la petición a la API se mostraría un mensaje de error.

21. **Obtener las acciones y mensajes por usuario de un proyecto**: Se podrán obtener las acciones y mensajes por usuario de un proyecto

    Entrada: El usuario selecciona la opción de mostrar estadísticas de acción o mensajes por usuario, indicando el usuario y el canal al que pertenece.

    Proceso: El sistema envía la petición correspondiente a *SOCIO*.

Salida: Si la petición se ha realizado con éxito se muestra en el chat la imagen con las estadísticas solicitadas Si hubiera algún error en la petición a la API se mostraría un mensaje de error.

22. **Obtener el porcentaje de autoría de un proyecto**: Se podrá obtener el porcentaje de autoría según las acciones de modelado llevadas a cabo por cada usuario.

    Entrada: El usuario selecciona la opción de obtener autoría de un proyecto.

    Proceso: El sistema envía la petición correspondiente a *SOCIO*.

    Salida: Si la petición se ha realizado con éxito se muestra la imagen con el porcentaje de autoría del proyecto. Si hubiera algún error en la petición a la API se mostraría un mensaje de error.

23. **Obtener todas las acciones de modelado de un proyecto**: Se podrán solicitar todas las acciones de modelado que cumplan determinados filtros.

    Entrada: El usuario selecciona la opción de obtener todas las acciones de modelado de un proyecto, y completa los filtros que quiera consultar en el formulario, estos pueden ser: usuario, fecha de inicio, fecha de fin, tipos de acciones o elementos que aparezcan en los mensajes de modelado.

    Proceso: El sistema envía la petición correspondiente a *SOCIO*.

    Salida: Si la petición se ha realizado con éxito se muestra las acciones de modelado que cumplan los filtros especificados. Si hubiera algún error en la petición a la API se mostraría un mensaje de error.

24. **Obtener las acciones de gestión de un proyecto**: Se podrán obtener todas las acciones de gestión de un proyecto

    Entrada: El usuario selecciona la opción de obtener acciones de gestión.

    Proceso: El sistema envía la petición correspondiente a *SOCIO*.

    Salida: Si la petición se ha realizado con éxito se muestra una imagen en el chat con las acciones de gestión llevadas a cabo durante el proyecto. Si hubiera algún error en la petición a la API se mostraría un mensaje de error.

25. **Obtener el modelo de un proyecto**: Se podrá obtener el modelo en formato .ecore

    Entrada: El usuario selecciona la opción obtener modelo.

    Proceso: El sistema envía la petición correspondiente a *SOCIO*.

    Salida: Si la petición se ha realizado con éxito se descarga un archivo en formato .ecore con el modelo. Si hubiera algún error en la petición a la API se mostraría un mensaje de error.

26. **Obtener ayuda**: Se podrá solicitar ayuda e información de *SOCIO*.

    Entrada: El usuario selecciona la opción de obtener ayuda

    Proceso: El sistema envía la petición correspondiente a *SOCIO*.

Salida: Si la petición se ha realizado con éxito se muestra el mensaje de ayuda. Si hubiera algún error en la petición a la API se mostraría un mensaje de error.

Subsistema de mensajería

27. **Envío en tiempo real de mensaje en chat**: Los usuarios enviarán mensajes en los chats y estos se mostrarán en tiempo real en la interfaz.

    Entrada: El usuario introduce en el cuadro de texto el mensaje que desea enviar al chat, ya sea un mensaje normal o para realizar una acción de modelado.

    Proceso: El sistema guardará el mensaje en base de datos, independientemente del tipo de mensaje que sea.

    Salida: Inmediatamente se mostrará el mensaje enviado en la sala de chat.

28. **Historial de mensajes**: El usuario podrá acceder al historial de mensajes enviados en los proyectos de la aplicación.

    Entrada: El usuario accede a la página del historial de mensajes.

    Proceso: El sistema dirigirá al usuario a la página del historial de mensajes.

    Salida: Se mostrarán al usuario los últimos mensajes por proyecto, los últimos mensajes de modificación y los últimos mensajes en proyectos propios. Si hubiera notificaciones de otros proyectos pendientes se lanzaría un pop-up indicando los proyectos que han sido modificados.

3.1.3 Requisitos no funcionales

En este apartado veremos el análisis de requisitos no funcionales realizado para la aplicación desglosados por las capacidades del sistema.

Seguridad

1. A la hora del registro de usuarios, se exigirán contraseñas con un mínimo de ocho caracteres, no podrán ser completamente numéricas, ni demasiado similares al resto de información personal del usuario

2. Las contraseñas serán encriptadas mediante el algoritmo SHA-256.

Plataformas

3. La aplicación funcionará en los navegadores *Google Chrome*, *Mozilla Firefox*, *Microsoft Edge* e *Internet Explorer* (versión 10 y posteriores), aunque ha sido testeado completamente en *Google Chrome*.

Usabilidad

4. El sistema de mensajería contará con un WebSocket especializado, que se reconectará automáticamente si el servicio sufriera alguna caída.

5. El sistema proporcionara mensajes de error, claros e informativos al usuario.

6. El sistema poseerá una interfaz gráfica sencilla e intuitiva, con el fin de lograr los objetivos de la aplicación.

Rendimiento

7. Las operaciones con la base datos no tardarán más de dos segundos en ser realizadas.

8. El muestreo de los mensajes en la sala de chat, una vez enviados, no tardará más de 0.1 segundos.

3.2 Diseño de la aplicación

A continuación, se presenta el diseño escogido para la herramienta *SOCIO-APP*. Este diseño, es un diseño simple, pensado para adaptarse a la perfección con las prestaciones del bot *SOCIO*.

Dado que *SOCIO* ya almacena la mayoría de los datos en una base de datos externas, en la base de datos local de la aplicación solo queremos almacenar los elementos que sean indispensables para el correcto funcionamiento de esta, consiguiendo así un mejor rendimiento. Siguiendo esta premisa, el modelo de *SOCIO-APP* se basará en los objectos proyecto, usuario y mensaje.

A continuación, en la Figura 3.1, se muestra el diagrama de clases realizado para la aplicación. Los proyectos estarán compuestos por un nombre de proyecto, un canal que indicará la aplicación desde la que ha sido creado el proyecto (*SOCIO-APP*, *Telegram o Twitter*), visibilidad, que puede ser pública, protegida o privada, el autor original, para que en proyectos externos a *SOCIO-APP* se tenga en cuenta el administrador externo y un timestamp, que indicará la fecha de creación del proyecto. Los usuarios están compuestos por el nombre de usuario, una contraseña, un correo electrónico y una imagen de perfil. Los mensajes tendrán los siguientes atributos, el texto del mensaje, un booleano que indicará si el mensaje va dirigido al bot o no y una fecha de envío del mensaje. Además, habrá un tipo especial de proyecto, que serán los branches, los cuales, además de todos los atributos de proyecto, también tendrán el atributo de proyecto padre.

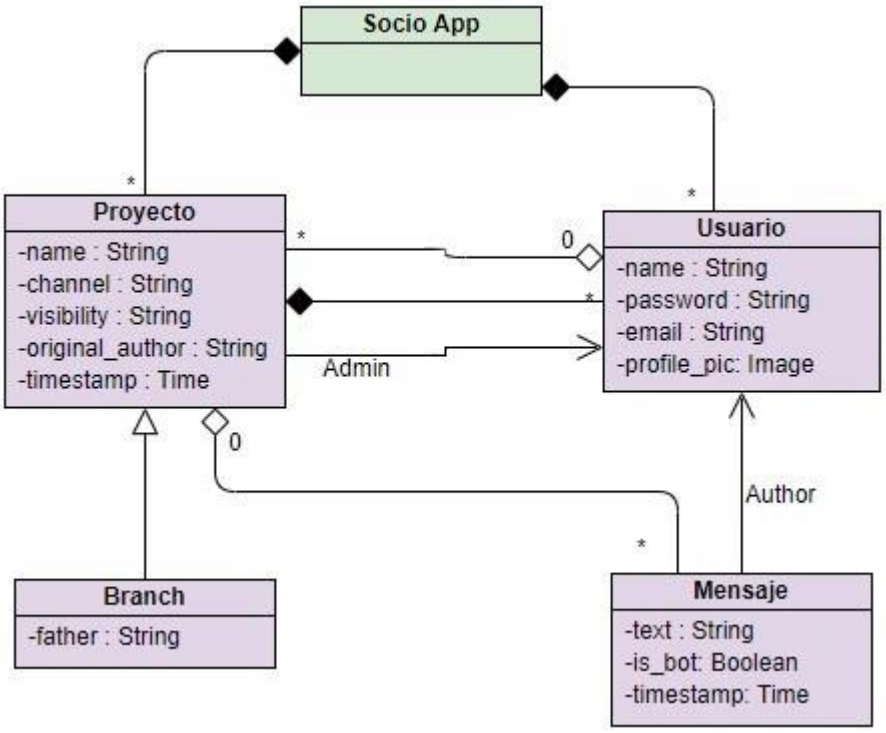

Figura 3.1: Diagrama de clases de la aplicación

Este diseño estará en constante comunicación con el servicio de acciones de la aplicación, el cual enviará sus peticiones a la API REST en la que está ubicado *SOCIO*.

Los datos de los proyectos como pudieran ser la lista de usuarios que pueden editar o que pueden leer determinado proyecto se encuentran en la base de datos de *SOCIO*. A estos datos se accederá y se tendrá constancia de sus valores mediante las peticiones (del tipo HTTP-Request) enviadas a la API.

En la Figura 3.2 se muestra la arquitectura de la aplicación y su comunicación con *SOCIO*.

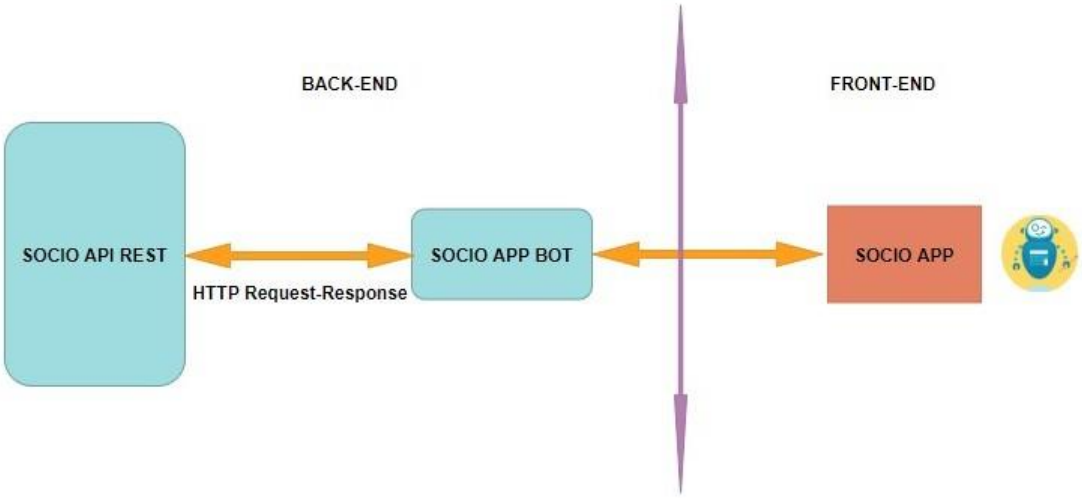

Figura 3.2: Arquitectura de la aplicación

El punto de conexión de esta arquitectura se basa en el subsistema de acciones de la aplicación. Estas acciones podrán ser llevadas a cabo por los usuarios desde el subsistema de proyectos. Estás acciones podrán ser externas al subsistema de proyecto como son la creación, modificación y eliminación, las cuales se realizan desde fuera de la propia estructura del proyecto, o las acciones internas las cuales solo pueden ser realizadas desde el interior de la plantilla de un proyecto concreto y que suponen la gran mayoría de las acciones. Cada una de las acciones supondrá una petición a la API de *SOCIO*.

4 Desarrollo

4.1 Estructura de la aplicación

Esta aplicación, al haber sido desarrollada con *Django*, sigue ciertas características propias de este framework. La aplicación estará dividida en una aplicación principal y dos sub-aplicaciones dentro del proyecto *Django*, estas sub-aplicaciones, serán la aplicación de proyectos y la aplicación de usuarios. La aplicación de usuarios no tiene mucha complicación dado que solo contiene un modelo, que es el propio usuario. La aplicación de proyecto es la que contiene toda la lógica de negocio de la herramienta. Está estará conformada por el modelo proyecto, además de los mensajes enviados en cada proyecto.

Los proyectos básicos en *Django,* están divididos por aplicaciones, y estas a su vez se organizan en varios módulos, estos son: el módulo de modelos, donde se almacenan los objetos previamente vistos, el módulo de plantillas, que es donde se almacenan los documentos HTML que se mostrarán en la aplicación web y el módulo de vistas, que ejercen como nexo de unión entre las plantillas y los modelos, es decir, entre el back-end y el front-end de la aplicación.

La aplicación principal será la encargada de mostrar las páginas iniciales y servir de base, en cuanto a estilos, para el resto de las aplicaciones que conforman *SOCIO-APP*.

La aplicación de usuarios será la que lleve la lógica de registro, login y modificaciones en usuarios. Está se encargará de almacenar nuevos usuarios en base de datos, realizar comprobaciones de contraseñas y modificar los datos del perfil de usuario.

La aplicación de proyectos será la más importante, dado que contendrá la parte más crucial de la lógica de negocio, es decir, todo lo relacionado con acciones realizadas sobre un proyecto, el sistema de mensajería, y las acciones enviadas al bot *SOCIO*. En esta sección expondremos únicamente las acciones realizadas sobre proyectos, dado que, en apartados posteriores, haremos hincapié en estos dos últimos puntos, el sistema de mensajería (apartado 4.3) y las acciones de conexión con el bot (apartado 4.4).

La aplicación proyecto, contiene dos plantillas principales que dan acceso al módulo del chat, las cuales se han decidido separar para no confundir al usuario, que son la plantilla de creación general de proyectos, y la plantilla de creación de referencias a proyectos externos. Esta decisión se ha tomado dado que, la creación de una referencia a un proyecto conlleva una lógica y datos muy diferentes a la de la creación de un proyecto al uso y así conseguir rebajar la complejidad de este proceso para el usuario. Desde estas plantillas se podrán realizar las acciones conectadas con las vistas de la aplicación proyecto de crear, eliminar y modificar un proyecto.

A continuación, en la figura 4.1, se muestra un ejemplo del código específico para crear u obtener un proyecto, este código se ejecuta dentro del administrador de proyectos contenido en el modelo de proyecto, cuando el usuario solicitar crear un nuevo proyecto.

```
def get_or_new(self, user, room_name, room_private, room_channel, branch_name, original_author):
    qlookup = Q(name=room_name)
    qlookupbranch = Q(name=branch_name)
    if branch_name != ROOM_NAME:
        qsbr = self.get_queryset().filter(qlookupbranch).distinct()
        if qsbr.count() == 1 and qsbr.first().private == PRIVATE:
            if qsbr.first().author == user or qsbr.first().member_set.filter(user=user):
                return qsbr.first(), False
            return None, False
        elif qsbr.count() == 1 and qsbr.first().private == PUBLIC:
            return qsbr.first(), False
        else:
            obj = Room(name=branch_name, author=user, private=room_private, channel=room_channel, branch=True,
                       project=room_name, original_author=original_author)
            obj.save()
            all_users = User.objects.all()
            for usr in all_users:
                obj.member_set.create(user=usr)
            obj.save()
            return obj, True
        return None, False
    else:
        qs = self.get_queryset().filter(qlookup).distinct()
        if qs.count() == 1 and qs.first().private == PRIVATE:
            if qs.first().author == user or qs.first().member_set.filter(user=user):
                return qs.first(), False
            return None, False
        elif qs.count() == 1 and qs.first().private == PUBLIC:
            return qs.first(), False
        else:
            obj = Room(name=room_name, author=user, private=room_private, channel=room_channel, original_author=original_author)
            obj.save()
            all_users = User.objects.all()
            for usr in all_users:
                obj.member_set.create(user=usr)
            obj.save()
            return obj, True
        return None, False
```

Figura 4.1: Creación u obtención de un proyecto

La modificación y eliminación de proyectos se llevan a cabo de una manera muy sencilla, gracias al sistema de vistas personalizadas que ofrece *Django*. Estas se conectan con la plantilla desde la que son llamadas y realizan las acciones que conlleve, según el tipo de vista personalizada del que hagan uso. En la figura 4.2, se muestran las vistas de modificación y borrado de un proyecto.

```
from django.views.generic import DetailView, ListView, CreateView, DeleteView, UpdateView
class RoomUpdateView(LoginRequiredMixin, UserPassesTestMixin, UpdateView):
    model = Room
    fields = ['private']

    def form_valid(self, form):
        form.instance.author = self.request.user
        return super().form_valid(form)

    def test_func(self):
        room = self.get_object()
        if self.request.user == room.author:
            return True
        return False

class RoomDeleteView(LoginRequiredMixin, UserPassesTestMixin, DeleteView):
    model = Room
    success_url = '/chatroom/'

    def test_func(self):
        room = self.get_object()
        if self.request.user == room.author:
            return True
        return False
```

Figura 4.2: Modificación y eliminación de un proyecto

4.2 Introducción

En esta sección, se expondrán las tecnologías utilizadas para el desarrollo de la herramienta, el proceso de construcción de la aplicación y las decisiones tomadas con diversos aspectos de esta.

La base de la aplicación *SOCIO-APP*, está hecha con *Django* [15], un framework del lenguaje de programación *Python* [16], para desarrollo web a alto nivel. *Django* tiene como ventajas la escalabilidad de los proyectos desarrollados, ayudar a evitar errores comunes en cuanto a la seguridad web de tu aplicación y su máxima, que es el flujo de trabajo, el cual es muy alto. Tomando la siguiente cita de su página web, en la que se realiza una definición de *Django* [15], "*creado por desarrolladores experimentados, se encarga de gran parte de las molestias que surgen en el desarrollo web, dejando que el creador se concentre en programar su aplicación, sin necesidad de reinventar la rueda*". Además, *Django* incluye un gestor de bases de datos *SQLite*.

Una vez determinada la tecnología en la que se basará la aplicación, ahora comentaremos el proceso de desarrollo de los módulos que componen la herramienta. En el apartado 4.2, veremos cómo se ha desarrollado la estructura de la aplicación, vista anteriormente en el apartado de diseño, en el apartado 4.3, conoceremos el desarrollo para llevar a cabo el chat en tiempo real y, por último, en el apartado 4.4 hablaremos de la implementación de la conexión con el bot *SOCIO*.

El front-end de la aplicación ha sido desarrollado en HTML, utilizando *Bootstrap*, una herramienta de libre uso para desarrollar con HTML, CSS y JavaScript, para mejorar visualmente la aplicación.

4.3 Chat en tiempo real

Este módulo se encuentra incluido en la aplicación de proyecto y es la columna vertebral de la aplicación, dado que la plantilla principal de un proyecto se basa en el chat. Para la realización de un chat estable y en tiempo real, se ha utilizado la librería *Django Channels* [17], para su definición veremos una cita de su página web, *"Channels es un proyecto que hace que Django extienda sus habilidades sobre HTTP, manejando WebSockets, protocolos de chat, protocolos IoT (Internet de las cosas) y mucho más. Está construido en una interfaz llamada ASGI (Asynchronous Server Gateway Interface)". Django Channels* se basa en la unión de varios módulos o paquetes:

- *Channels*: Es la capa de integración con *Django.*
- *Daphne*: Es el servidor terminal de WebSockets y HTTP.
- *Asgiref:* La librería base de *ASGI*.
- *Redis*: Es el canal base para la lógica del back-end.

Además, para asegurar aún más la estabilidad del chat, se ha hecho uso de la librería de *JavaScript*, *ReconnectingWebSocket* [18], la cual mejora sustancialmente el funcionamiento de la API de WebSockets, haciendo que, si alguna vez se cae la conexión, se reconecte automáticamente.

Para fusionar correctamente el módulo de *Channels*, con el proyecto *Django*, tenemos que habilitar un nuevo módulo, que será el módulo de consumidores, el cual, como su propio nombre indica, tendrá la función de consumir todas las conexiones y acciones del WebSocket desde el back-end de la aplicación.

A continuación, en las figuras 4.3 y 4.4, se muestran ejemplos de las funciones más importantes de este módulo, que son la conexión o apertura del WebSocket y la llegada de mensajes al WebSocket.

```
async def websocket_connect(self, event):
    room_name = self.scope['url_route']['kwargs']['room_name']
    room_private = self.scope['url_route']['kwargs']['room_private']
    room_channel = self.scope['url_route']['kwargs']['room_channel']
    room_author = self.scope['url_route']['kwargs']['room_author']
    branch_name = self.scope['url_route']['kwargs']['branch_name']
    user = self.scope['user']
    thread_obj = await self.get_thread(user, room_name, room_private, room_channel, branch_name, room_author)
    self.thread_obj = thread_obj
    chat_room = f"thread_{thread_obj.id}"
    self.chat_room = chat_room
    await self.channel_layer.group_add(
        chat_room,
        self.channel_name
    )
    await self.send({
        "type": "websocket.accept"
    })
```

Figura 4.3: Función de apertura o conexión del WebSocket

```
async def websocket_receive(self, event):
    front_text = event.get('text', None)
    if front_text is not None:
        loaded_dict_data = json.loads(front_text)
        msg = loaded_dict_data.get('message')
        bot = loaded_dict_data.get('bot')
        user = self.scope['user']
        username = 'default'
        if user.is_authenticated:
            username = user.username
        myResponse = {
            'message': msg,
            'username': username,
            'bot': bot
        }
        await self.create_chat_message(user, msg, bot)
        # broadcast the message
        await self.channel_layer.group_send(
            self.chat_room,
            {
                "type": "chat_message",
                "text": json.dumps(myResponse)
            }
        )
```

Figura 4.4: Función de llegada de mensaje al WebSocket

Estas funciones son llamadas desde el módulo de JavaScript integrado dentro de la plantilla del chat. Desde este módulo, se referenciarán todas estas llamadas y se transportarán los datos ingresados por el usuario desde el front-end al back-end de la aplicación. Este módulo estará basado en la librería *ReconnectingWebSocket*, vista anteriormente en esta sección. Este WebSocket especializado, estará apuntando en todo momento al "endpoint" del chat en cuestión, y estará conformado por las funciones de apertura, de llegada de mensaje, de cierre y de error en el socket.

En la figura 4.5, se muestra el ejemplo de la función JavaScript que envía el mensaje, cuando el usuario completa el formulario de envío de mensaje dentro del chat, para que sea tratado por la función de llegada de mensaje al WebSocket vista anteriormente en la figura 4.4.

```
socket.onopen = function (e) {
  formData.submit(function (event) {
    event.preventDefault()
    var msgText = msgInput.val()
    if (checkBox.checked == true){
      var finalData = {
        'message': msgText,
        'bot': true
      }
    }
    else{
      var finalData = {
        'message': msgText,
        'bot': false
      }
    }
    socket.send(JSON.stringify(finalData))
    formData[0].reset()
  })
```

Figura 4.5: Función de envío de mensajes al WebSocket

4.4 Conexión con SOCIO

La conexión con el bot *SOCIO*, también está contenida dentro de la aplicación de proyecto. Las acciones para la creación, modificación y eliminación de proyectos se llevarán a cabo desde la página principal de proyecto. El resto de las acciones se realizarán desde dentro de la ventana del chat de proyecto. Cada una de estas acciones supondrá una petición HTTP a la API REST de *SOCIO.*

Habrá una función que lanzará continuamente una petición para buscar actualizaciones en los proyectos de interés para nuestro canal, y así, detectar e informar sobre posibles cambios realizados en los proyectos desde fuera de la aplicación (cambios que ejecute un usuario de *Telegram*, por ejemplo, en un proyecto de nuestra aplicación).

Estas peticiones se lanzarán mediante la interfaz *XMLHTTPRequest* (XHR), empleada bajo el lenguaje de programación *JavaScript* y la técnica de desarrollo web interactiva *Ajax.* La mayoría de las peticiones serán de tipo POST, devolviendo datos de tipo: texto, JSON y Octet-Stream.

A continuación, en las figuras 4.6 y 4.7, se muestran respectivamente, una petición del tipo GET enviada a la API de *SOCIO* para obtener información de ayuda sobre el bot, y una petición de tipo POST que devuelve una imagen con el estado del modelo de un proyecto.

```
$("#HelpPopUp").click(function() {
  var me = $("#myUsername").val();
  var url = "http://dimo1.ii.uam.es:8080/Socio-rest/help";
  var xhr = new XMLHttpRequest();
  xhr.onreadystatechange = function(){
      if(xhr.readyState == 4) {
          if(xhr.status == 200) {
            alertify.alert(xhr.response.text + xhr.response.url, function(){
              alertify.message('Help');
            });
          } else if(xhr.responseText != "") {
              alertify.error(xhr.responseText);
          }
      } else if(xhr.readyState == 2) {
          if(xhr.status == 200) {
              xhr.responseType = "json";
          } else {
              xhr.responseType = "text";
          }
      }
  }
  xhr.open('GET', url);
  xhr.send();
});
```

Figura 4.6: Petición XHR de tipo GET

```
var url = "http://dimo1.ii.uam.es:8080/Socio-rest/editor/do/Socio-App/" + me + "/" + room;
var xmlhttp = new XMLHttpRequest();   // new HttpRequest instance
xmlhttp.onreadystatechange = function(){
    if(xmlhttp.readyState == 4) {
        if(xmlhttp.status == 200) {
        var img = document.getElementById("img");
        var url = window.URL || window.webkitURL;
        img.src = url.createObjectURL(this.response);
        document.getElementById("img").style.display = "block";
        document.getElementById("close-btn").style.display = "block";
        } else if(xmlhttp.responseText != "") {
            alertify.error(xmlhttp.responseText);
        }
    } else if(xmlhttp.readyState == 2) {
        if(xmlhttp.status == 200) {
            xmlhttp.responseType = "blob";
        } else {
            xmlhttp.responseType = "text";
        }
    }
}
xmlhttp.open("POST", url);
xmlhttp.setRequestHeader("Content-Type", "application/json");
xmlhttp.send(JSON.stringify(msg_to_send));
```

Figura 4.7: Petición XHR de tipo POST

5 Herramienta

En esta sección, veremos el front-end de la aplicación, sus posibles usos e interacciones por parte del usuario, y como se han llevado a cabo todos los requisitos y acciones descritos previamente en este documento.

5.1 Introducción de la aplicación

La aplicación recibirá a los usuarios, con una página de inicio en la que se podrá visualizar el video del uso de *SOCIO*, desde esta página se tendrá acceso a las páginas de login y registro de usuario, como podemos ver en la figura 5.1.

Figura 5.1: Página de inicio de *SOCIO-APP*

Las páginas de login y registro de usuario tienen conexión con el subsistema de usuarios, y su perspectiva es la que se muestra en las figuras 5.2 y 5.3 respectivamente.

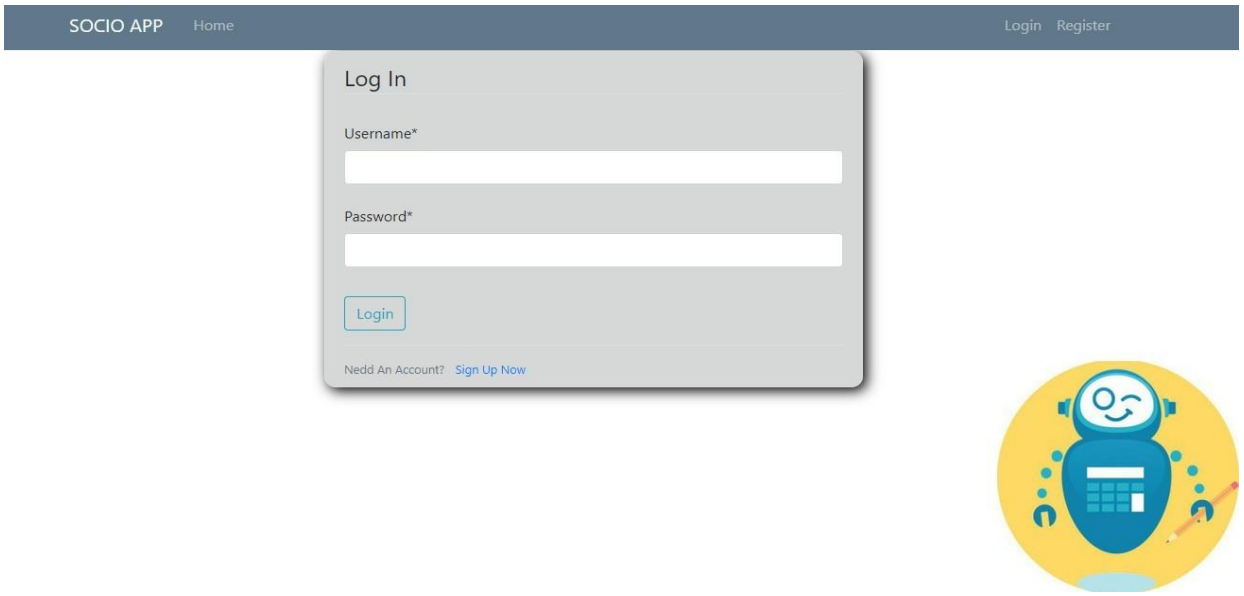

Figura 5.2: Página de login de *SOCIO-APP*

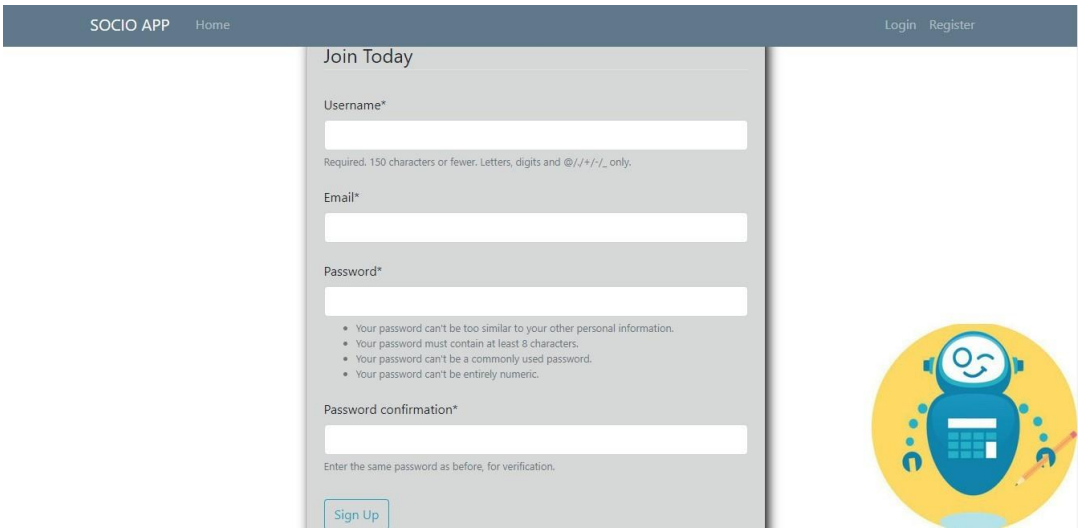

Figura 5.3: Página de registro de *SOCIO-APP*

Una vez que el usuario haya entrado al sistema se le recibirá en la página de inicio nuevamente, pero con diferentes opciones desplegadas, como son el acceso al sistema de notificaciones, el acceso al perfil de usuario, y el acceso a las ventanas de proyectos. Estos cambios se muestran en la figura 5.4.

Figura 5.4: Página de inicio de *SOCIO-APP* para usuario registrado

5.2 Administración proyectos

A continuación, veremos el módulo de proyectos, principalmente las páginas de administración y acceso.

En primer lugar, tenemos la página de creación y administración de proyectos internos de la aplicación.

Esta página constará, de un formulario para introducir el nombre del nuevo proyecto y su visibilidad, también se mostrarán los proyectos de los cuales somos administrador y los proyectos de otros usuarios. Para los proyectos de los que somos administrador se podrán realizar acciones de modificación y borrado, aparte de la de acceso al proyecto, mientras que para los proyectos de otros usuarios solo tendremos opción de acceso. Podemos ver la vista de esta página en la figura 5.5.

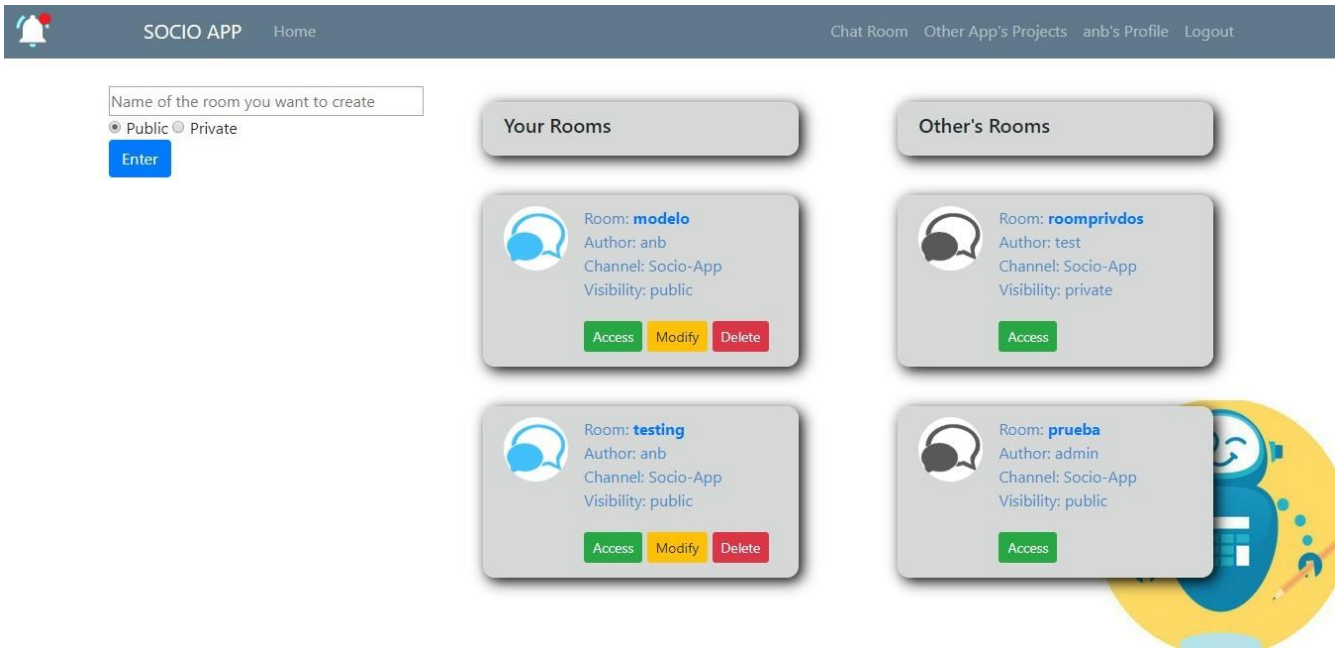

Figura 5.5: Página administración de proyectos

En segundo lugar, veremos la página de creación y administración de referencias a proyectos externos a la aplicación.

Esta página también constará, de un formulario para introducir el nombre del proyecto a buscar, además del canal en el que existe ese proyecto y se mostrarán los proyectos externos a los cuales se ha hecho referencia. Para los proyectos de los que somos administrador se podrá acceder a ellos y eliminar el vínculo. Como se ha comentado en apartados anteriores la lógica de esta página es diferente a la de la creación estándar de un proyecto, dado que aquí, primero se debe buscar el nombre del proyecto y su canal, si este no existiera se lanzaría un error y si existiera solo se crearía el proyecto en la base de datos de *SOCIO-APP*, no se lanzaría la petición de creación a *SOCIO*, dado que este proyecto ya existe. Podemos ver la vista de esta página en la figura 5.6.

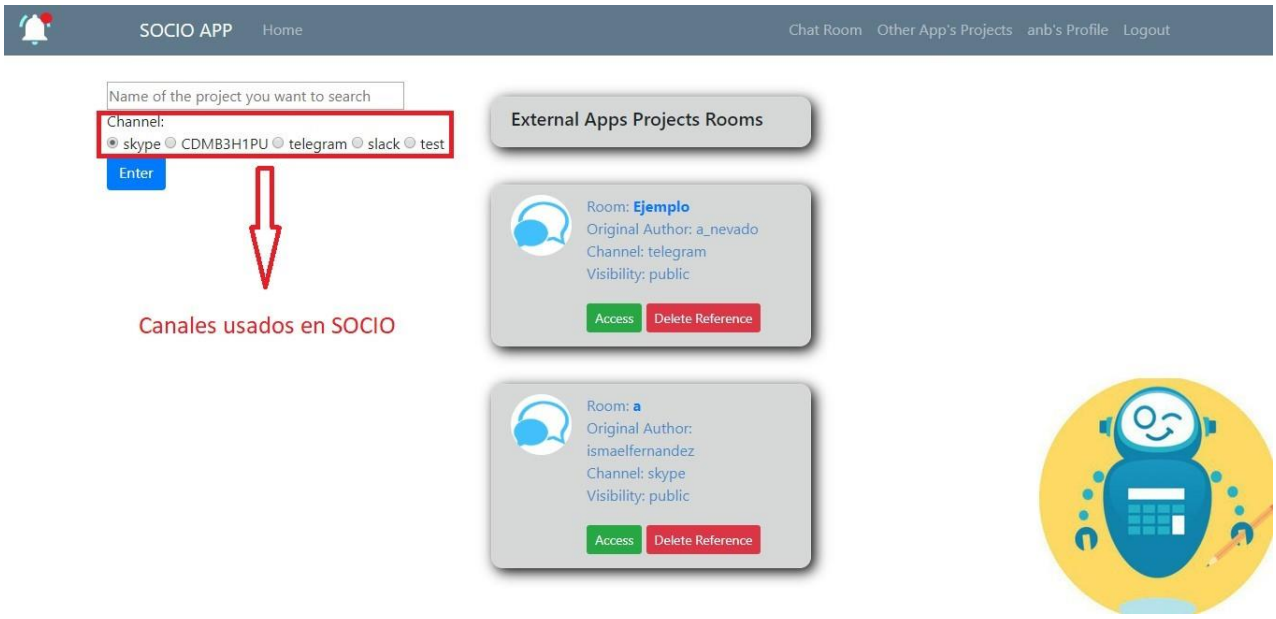

Figura 5.6: Página administración de proyectos externos

5.3 Chat

En este apartado veremos el sistema de chat desarrollado para los proyectos, el acceso a este se hará desde las páginas de administración de proyectos vistas en el punto anterior.

En esta vista, accederemos a una habitación de chat, con visualización de la lista de usuarios de la aplicación, un panel de acciones desplegable para realizar peticiones a *SOCIO* y el formulario para el envío de mensajes, así como el cuadro de chat donde se visualizarán estos mensajes. En la figura 5.7 se muestra la vista previa de la página de chat.

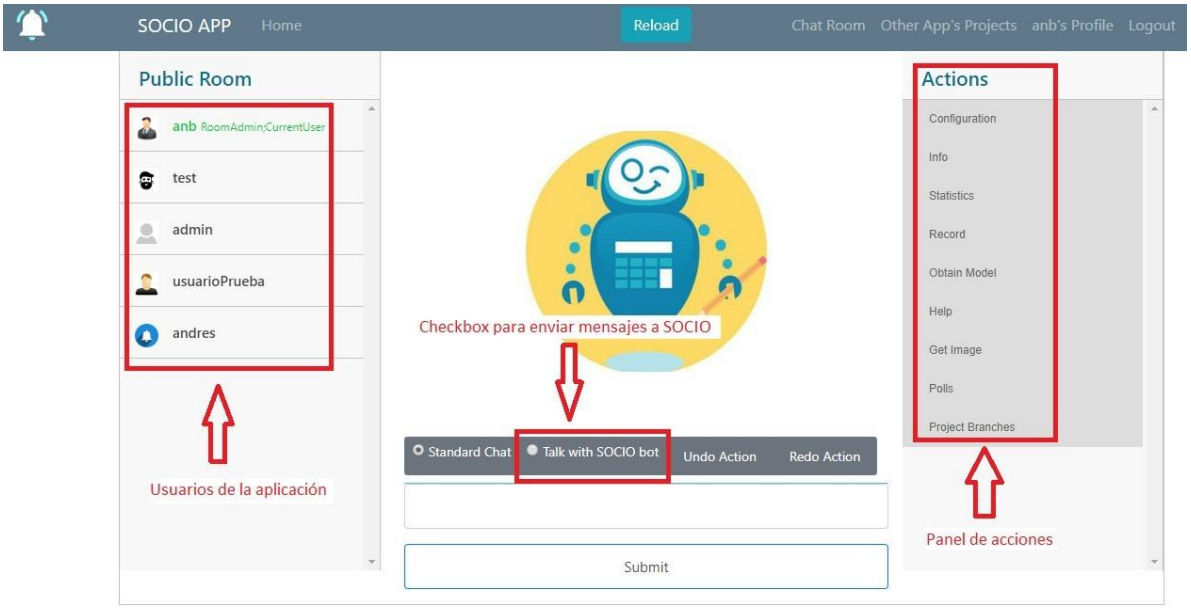

Figura 5.7: Página de una habitación de chat para proyecto

Para ver mejor la dinámica del chat, a continuación, se mostrarán unos ejemplos del uso del chat de proyecto, para la creación de un modelo. Para realizar una acción de modelado se debe seleccionar el checkbox de “Talk with SOCIO bot” y escribir en el formulario de texto el mensaje de modelado que se desea enviar al bot. Cabe resaltar que los mensajes tendrán estilos diferentes si son mensajes de modelado, mensajes estándar o mensajes externos. Estos ejemplos se mostrarán en las siguientes dos capturas sacadas de la sala de chat de un proyecto (figura 5.8 y figura 5.9).

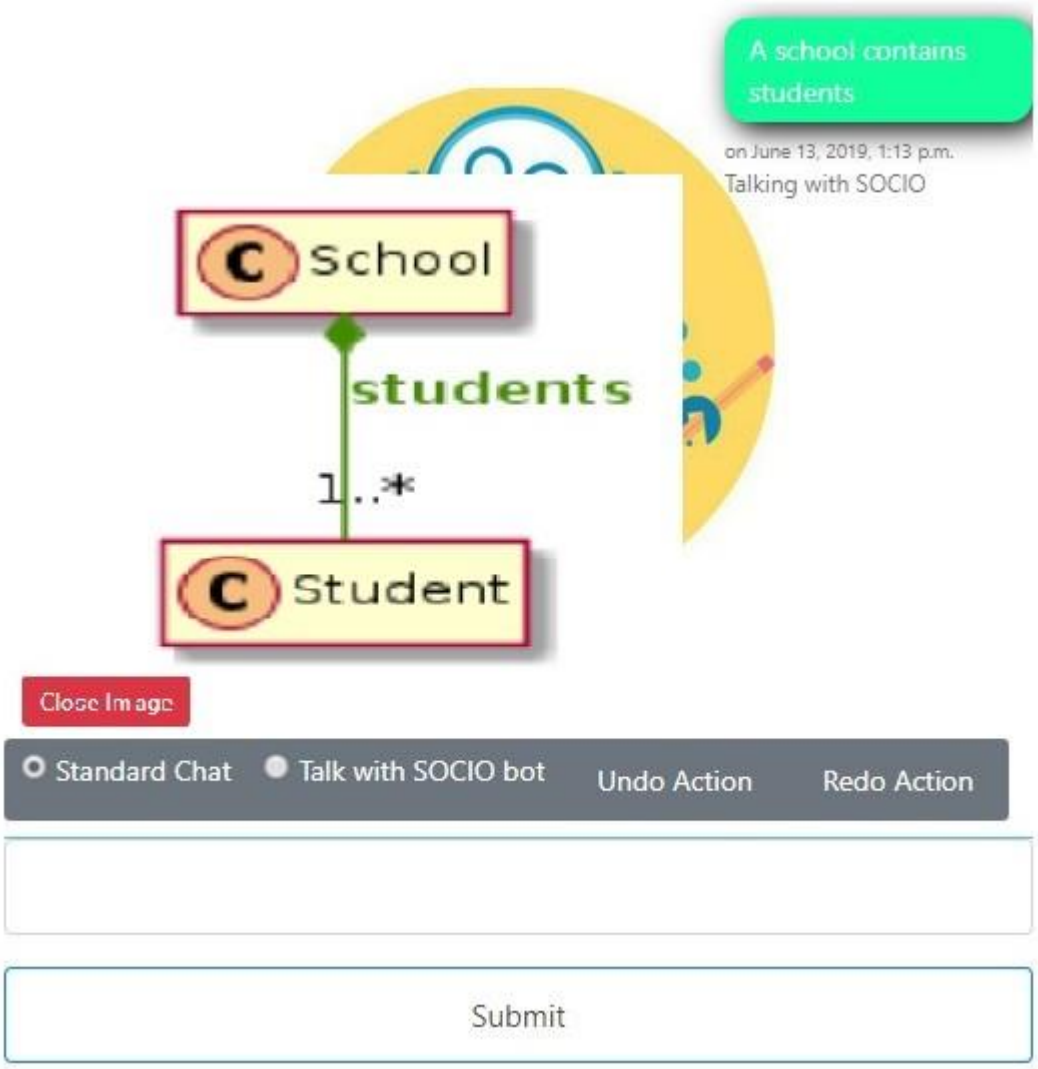

Figura 5.8: Ejemplo 1 de acción de modelado en chat

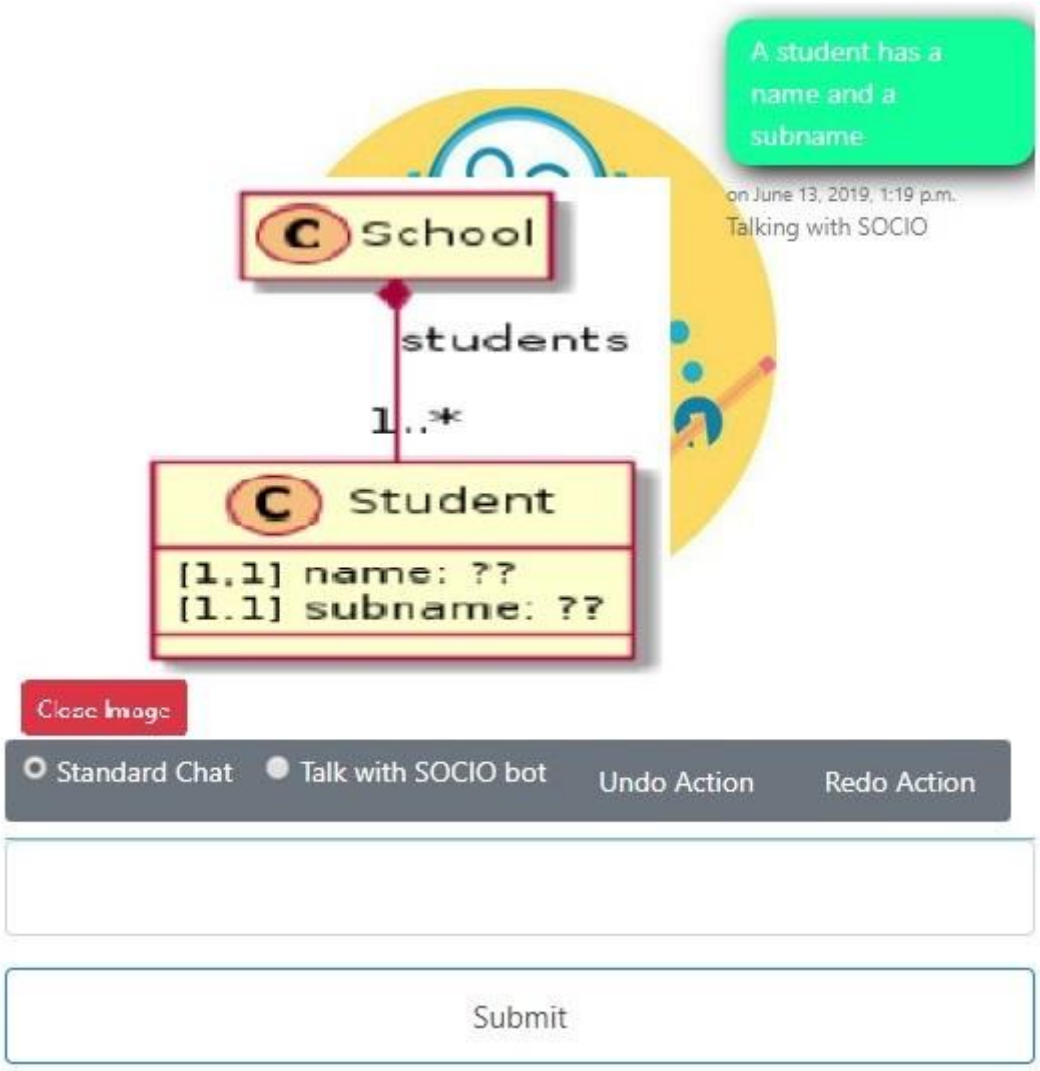

Figura 5.9: Ejemplo 2 de acción de modelado en chat

Siguiendo con el modelado mediante lenguaje natural, podemos llegar a obtener un modelo completo, como podría ser el siguiente, mostrado en la figura 5.10, tras realizar sucesivas acciones de modelado en la sala de chat del proyecto utilizado como ejemplo.

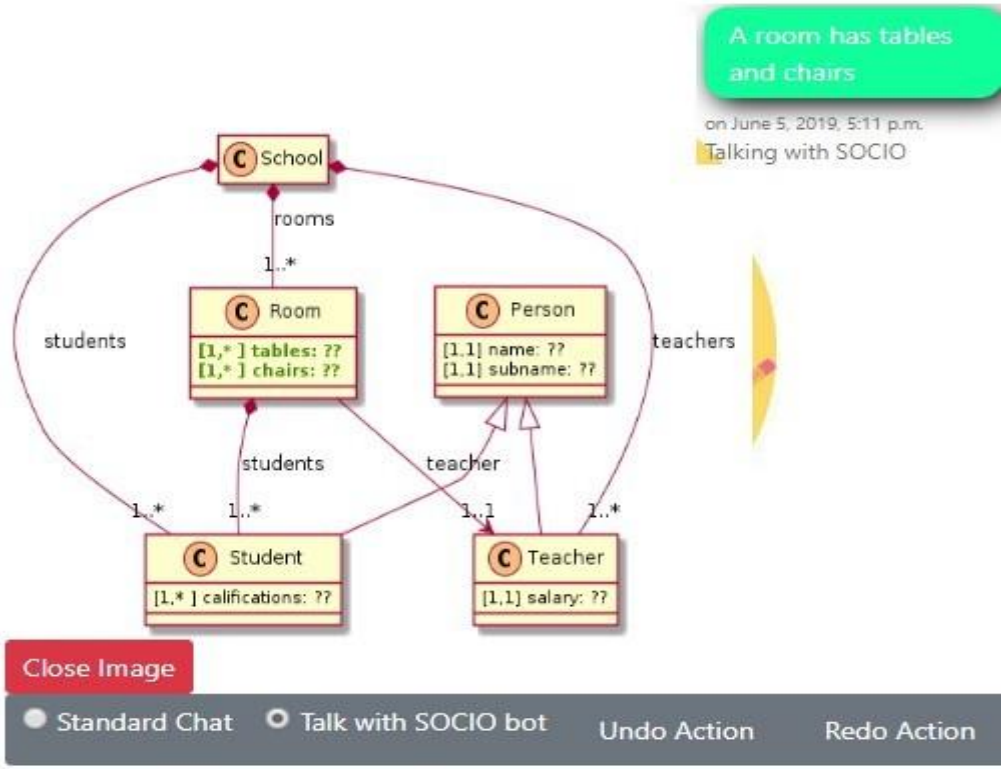

Figura 5.10: Ejemplo 3 de acción de modelado en chat

En la siguiente imagen, (figura 5.11), vemos la llegada de un mensaje externo a un proyecto de la aplicación. En este caso en concreto se ha recibido un mensaje de modelado desde la herramienta *Telegram.* En esta situación, se mostrará un pop-up de aviso al usuario que este en el chat de que ha habido una modificación externa y se mostrará en mensaje en un recuadro amarillo, indicando el usuario y la aplicación desde la que se está realizando la acción de modelado.

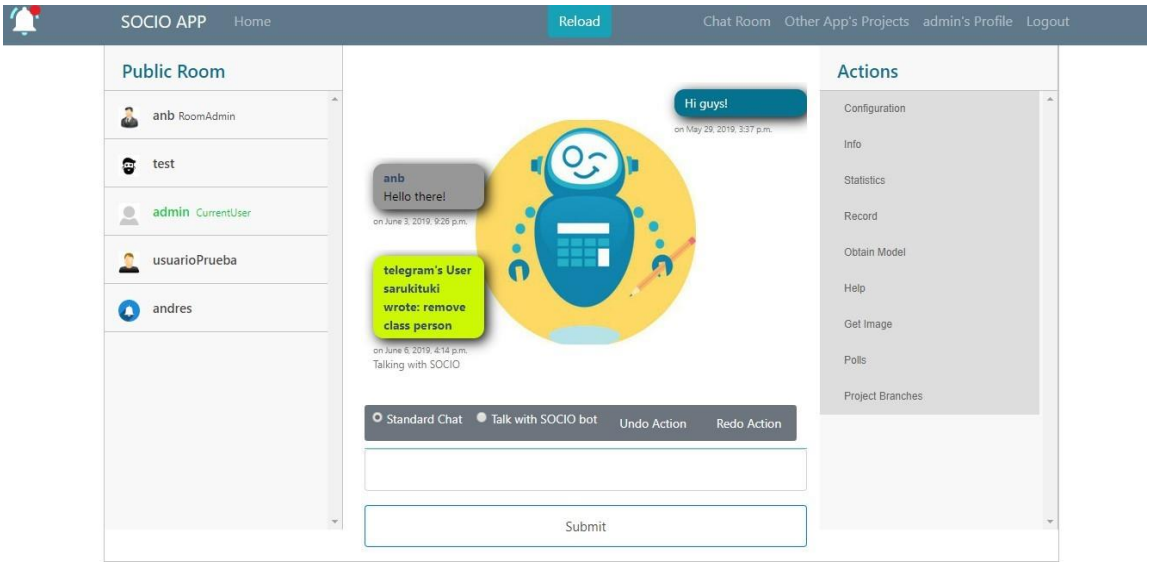

Figura 5.11: Ejemplo de llegada de mensaje externo

A continuación, se mostrarán ejemplos de otras acciones, que no sean de modelado por lenguaje natural, ejecutadas seleccionando los botones del panel de acciones situado a la derecha del chat.

En la figura 5.12 podemos ver el formulario que aparece cuando se selecciona la opción de dar permisos de lectura o escritura a un usuario, a la cual se accede desde el desplegable de configuración. Como podemos observar este formulario da la opción de seleccionar el canal al que pertenece el usuario, pudiendo así dar permiso a usuarios de otras aplicaciones a editar nuestro proyecto, también te permite elegir el tipo de permiso a asignar, ya sea lectura o escritura y un cuadro de texto para introducir el nombre del usuario al que se desean dar los permisos. Una vez se envíe el formulario el sistema responderá con una alerta de éxito, o de error en caso de que algo haya ido mal.

Este formulario será el estándar de cualquier operación seleccionada en la que haya que introducir más de un dato o seleccionar entre varios en forma de ratio-button o checkbox.

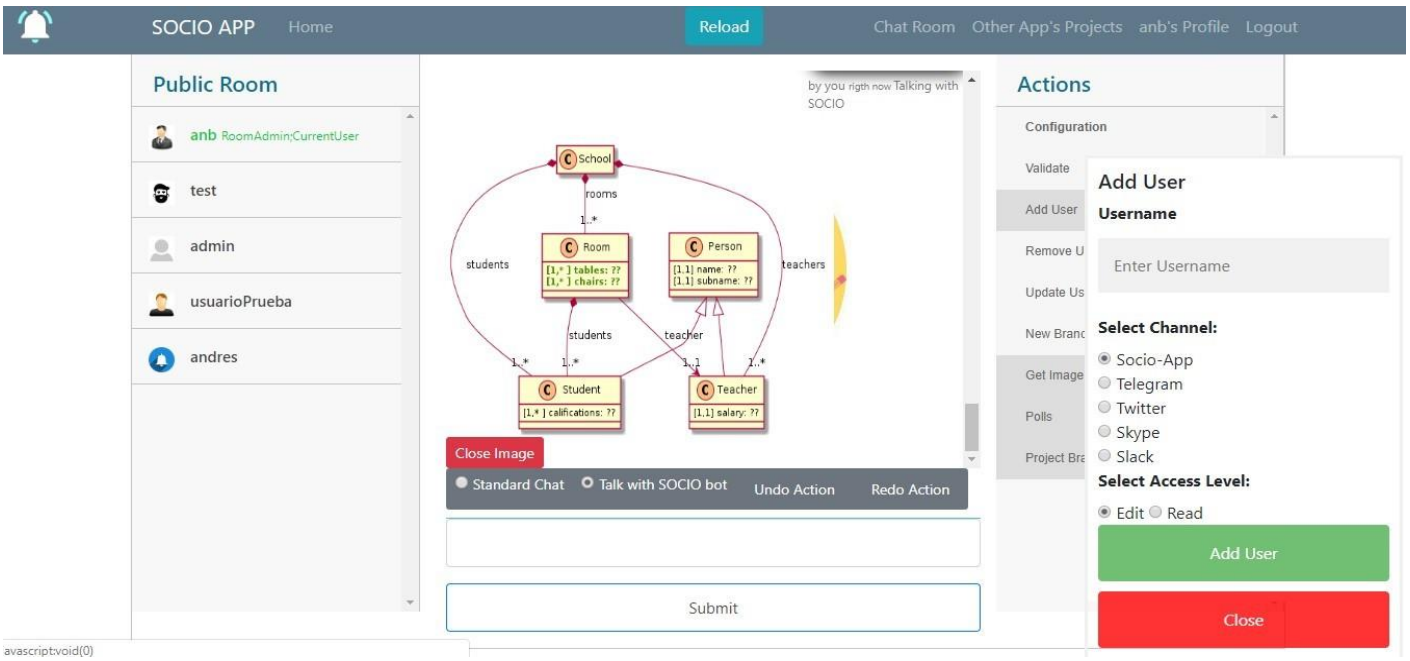

Figura 5.12: Acción de dar permisos a un usuario

Otra operación interesante, es la creación de branches de un proyecto. Está operación se seleccionará desde el desplegable de configuración. Se nos mostrará un formulario en el que debemos introducir el nombre que deseamos dar al branch (figura 5.13), una vez aceptemos la operación se nos llevará a la sala de chat del nuevo branch, con un vínculo al proyecto padre en el panel de acciones (figura 5.14). Una vez un proyecto tenga branches creados, estos se mostrarán en el desplegable del panel de acciones "Project Branches" (figura 5.15).

En la figura 5.13 se muestra el formulario en forma de pop-up, que salta una vez seleccionamos la opción de crear un nuevo branch, dentro del cuadro de texto indicaremos el nombre que deseamos darle al nuevo branch.

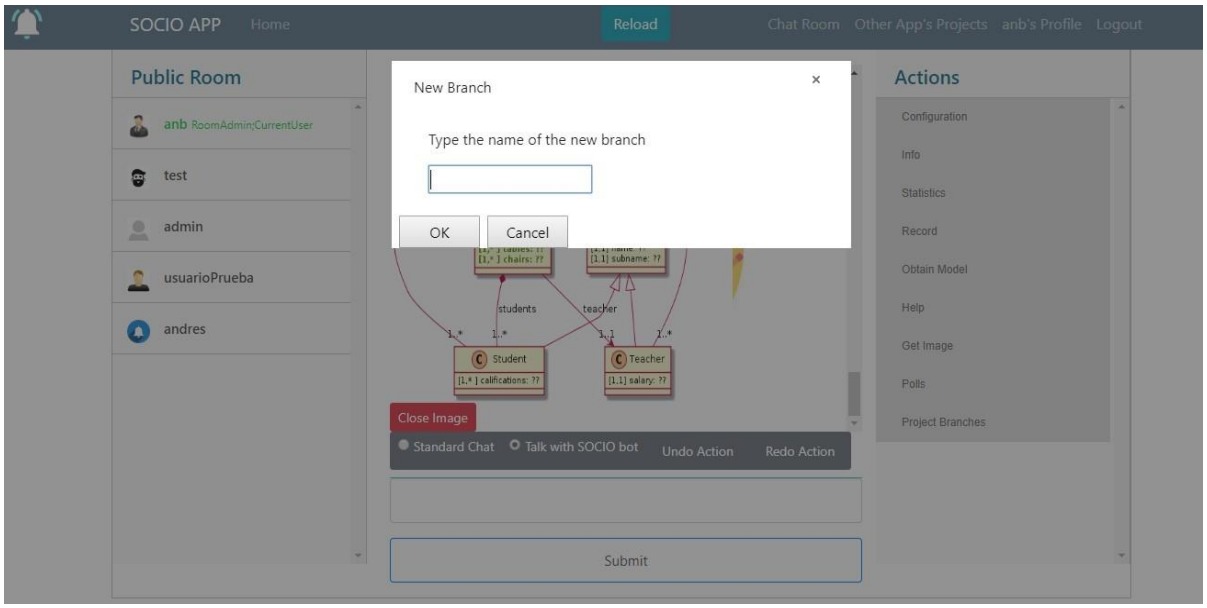

Figura 5.13: Formulario de creación de branch

En la figura 5.14, se muestra la sala de chat para un branch. Esta es idéntica a la sala de chat normal de un proyecto, con la única diferencia del muestreo de los branches. Al ser una función inservible para este tipo de proyecto, se sustituye por un botón de redireccionamiento al proyecto padre del branch.

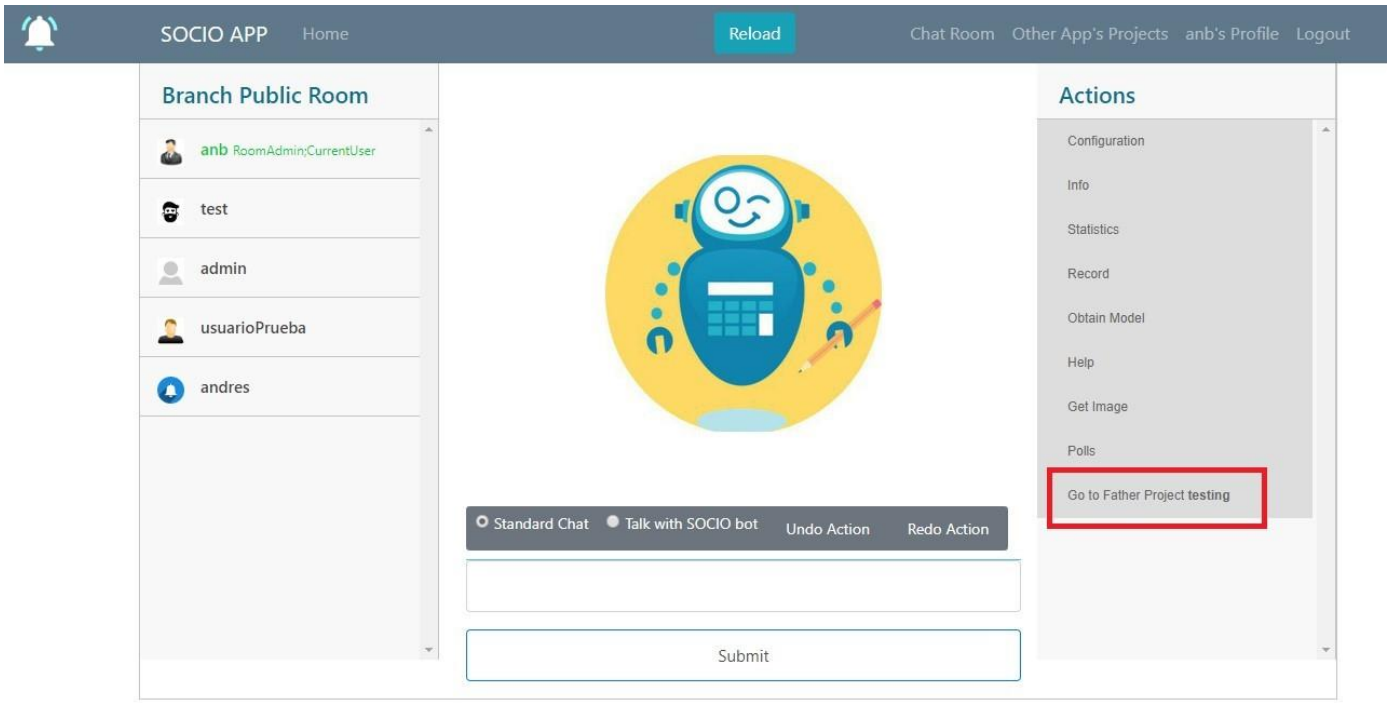

Figura 5.14: Sala de chat para un branch

En la figura 5.15, se muestra la sala de chat de un proyecto con múltiples branches creados. Como podemos ver en el cuadro de acciones, se podrá acceder a los branches del proyecto mediante un menú desplegable en el que se desglosarán las ramificaciones del proyecto.

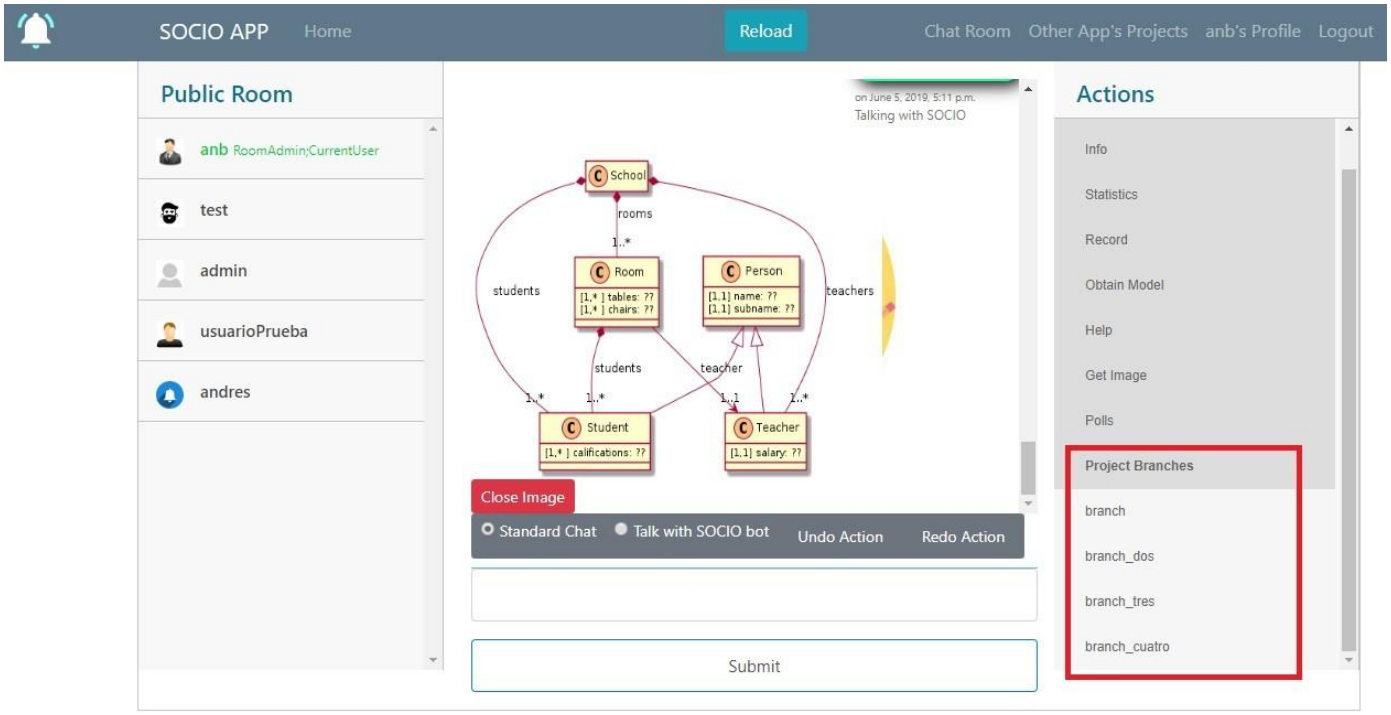

Figura 5.15: Sala de chat de proyecto con branches

5.4 Notificaciones (Historial)

Desde esta ventana se podrá hacer un seguimiento de las actualizaciones y mensajes recibidos en los proyectos. La imagen de la campana que hay en la barra principal de navegación de la aplicación, mostrará un círculo rojo cuando alguno de los proyectos de la aplicación haya sido modificado.

Esta página nos recibirá, si hubiera notificaciones en los proyectos de la aplicación, con un pop-up informándonos de que proyectos han sido modificados, especificando su nombre, autor y canal.

En esta página, habrá tres tipos de listas, en las que se diferenciará los mensajes de mayor interés. En la primera lista se mostrarán los últimos mensajes escritos en los proyectos del usuario. En la segunda lista se mostrarán las últimas acciones de modelado de cada proyecto, esta lista será especial dado que cada mensaje contendrá un botón para poder visualizar la imagen del modelo tras esa última acción de modelado. En la tercera y última lista se mostrarán los últimos mensajes de cada proyecto. Los mensajes se diferenciarán entre mensajes de modelado o mensajes estándar según el color, verde para los mensajes de modelado, gris para los normales y amarillo para los mensajes externos a la aplicación.

En las siguientes figuras se pueden ver las notificaciones de chat (figura 5.16), así como la imagen obtenida de una de las últimas acciones de modelado efectuadas sobre un proyecto (figura 5.17) y el pop-up de aviso de modificación en proyecto (figura 5.18).

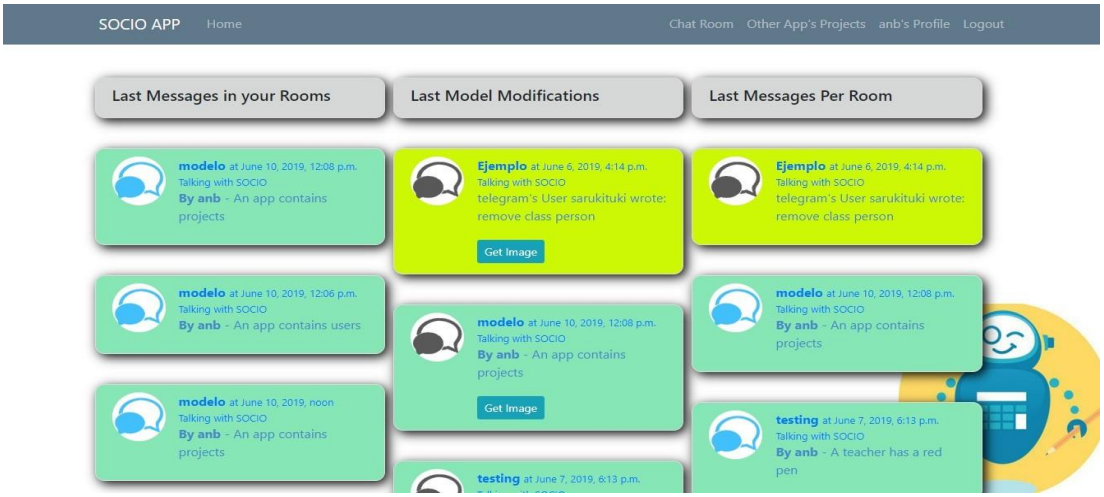

Figura 5.16: Página de notificaciones o historial

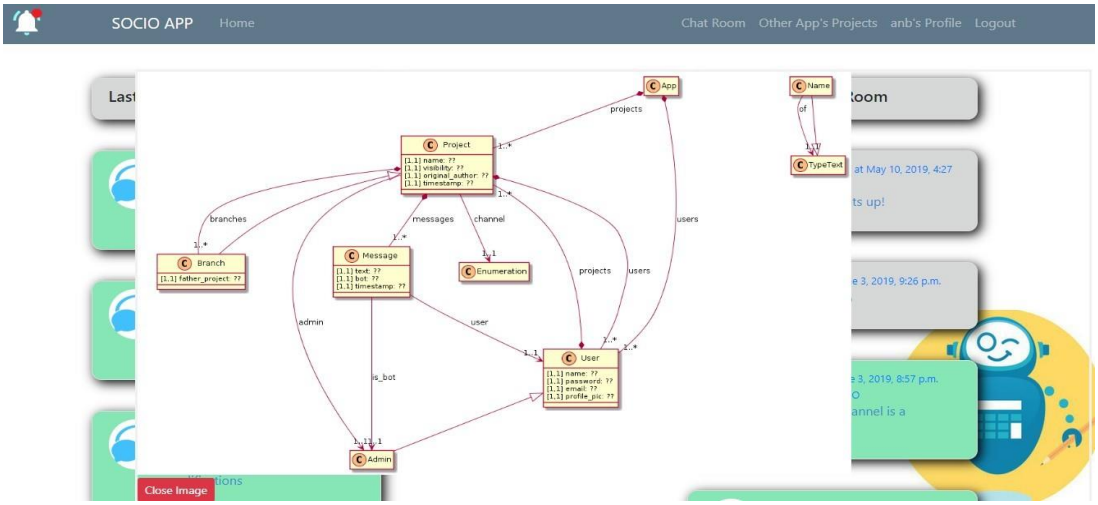

Figura 5.17: Imagen de modelo en página de notificaciones o historial

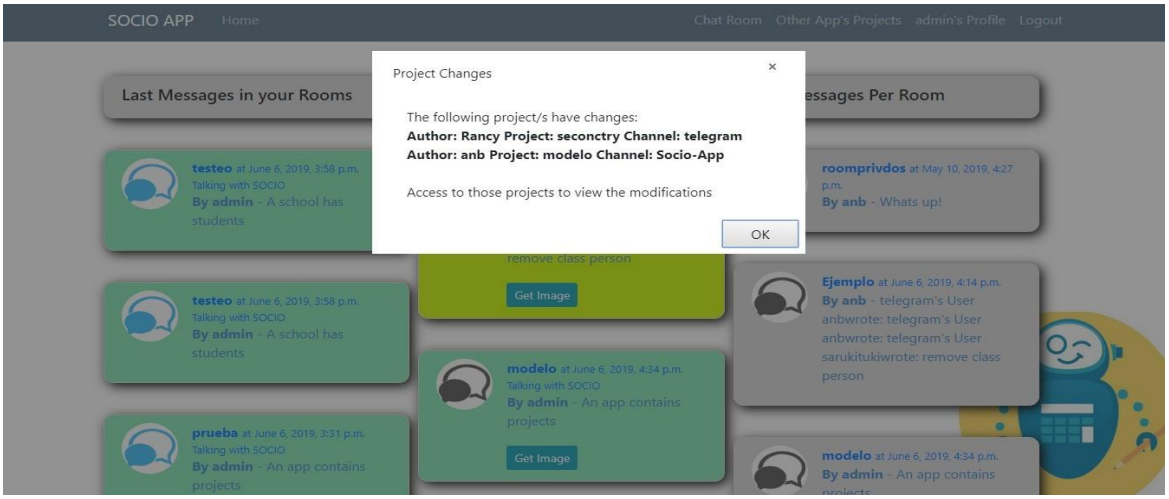

Figura 5.18: Pop-up informativo en página de notificaciones o historial

6 Conclusiones y trabajo futuro

6.1 Conclusiones

El desarrollo colaborativo es una forma de trabajo básica en los tiempos en los que vivimos y el gran número de herramientas creadas e incluso, la adaptación de muchas plataformas, para esta modalidad son prueba de ello. Dentro del modelado colaborativo, la posibilidad de unir los conocimientos de expertos en el modelado, con otro tipo de expertos, como pudieran ser expertos en dominio, daría lugar a modelos mejor enfocados desde fases tempranas del desarrollo.

Este trabajo ha hecho uso de un proyecto de modelado colaborativo, enfocado para redes sociales, mediante el uso de chatbots e interpretación de lenguaje natural, y lo ha convertido en una aplicación propia, singular y especializada, para la ardua tarea de modelar colaborativamente.

El fusionar la funcionalidad del bot *SOCIO*, con una aplicación totalmente dedicada a explotar todo su potencial, mediante un sistema de mensajería propio, fácil acceso a todas sus funcionalidades y una interfaz de usuario adecuada y creada exclusivamente para su uso con el bot consigue que la experiencia de uso de *SOCIO* sea mucho más satisfactoria.

Llegados a cumplir el objetivo de desarrollar la aplicación, podemos decir que *SOCIO-APP* es una aplicación con un gran potencial, además, teniendo en cuenta que no existe ninguna herramienta para modelado colaborativo mediante lenguaje natural, esta aplicación podría considerarse pionera en este tipo de desarrollos, consiguiendo así una gran capacidad de inclusión entre los usuarios, pudiendo ser utilizada tanto por usuarios noveles como expertos en la materia.

6.2 Trabajo futuro

Como posible trabajo futuro para este TFG se exponen los siguientes puntos:

- Exportación del proyecto como aplicación para dispositivos móviles. Aunque el desarrollo web es totalmente *responsive*, para llevarlo a otras plataformas habría que trabajar más el diseño de la herramienta.
- Inclusión en la aplicación de un sistema de edición gráfica online, para poder tratar los modelos de una forma más directa.
- Desarrollo de sistema de votaciones para las ramas de un proyecto.

Referencias

[1] Sara Pérez Soler, "Modelado colaborativo en lenguaje natural a través de redes sociales", UAM. Departamento de Ingeniería Informática, junio de 2018 http://hdl.handle.net/10486/648809

[2] Louis S. Richman, Julianne Slovak, "Software Catches the Team Spirit", IEEE Fortune Magazine, nº8, páginas 93-96, 8 de junio de 1987 http://archive.fortune.com/magazines/fortune/fortune_archive/1987/06/08/69109/index.htm

[3] https://slack.com/intl/es-es/ [Última fecha de acceso: 08/06/2019]

[4] https://github.com/ [Última fecha de acceso: 17/06/2019]

[5] https://www.google.es/intl/es/docs/about/ [Última fecha de acceso: 14/05/2019]

[6] https://asana.com/ [Última fecha de acceso: 08/06/2019]

[7] https://idoc.mockplus.com/ [Última fecha de acceso: 08/06/2019]

[8] https://www.visual-paradigm.com/ [Última fecha de acceso: 08/06/2019]

[9] https://www.gliffy.com/ [Última fecha de acceso: 08/06/2019]

[10] https://es.atlassian.com/software/jira [Última fecha de acceso: 18/06/2019]

[11] https://es.atlassian.com/ [Última fecha de acceso: 19/06/2019]

[12] https://www.genmymodel.com/ [Última fecha de acceso: 08/06/2019]

[13] https://cacoo.com/es/ [Última fecha de acceso: 08/06/2019]

[14] https://creately.com/ [Última fecha de acceso: 08/06/2019]

[15] https://www.djangoproject.com/ [Última fecha de acceso: 10/06/2019]

[16] https://www.python.org/ [Última fecha de acceso: 12/06/2019]

[17] https://channels.readthedocs.io/en/latest/ [Última fecha de acceso: 15/06/2019]

[18] https://github.com/joewalnes/reconnecting-websocket [Última fecha de acceso: 23/03/2019]

Glosario

API	Application Programming Interface
TFG	Trabajo de Fin de Grado
GUI	Graphical User Interface
SHA	Secure Hash Algorithm
REST	Representational State Transfer
HTTP	HyperText Transfer Protocol
HTML	HyperText Markup Language
ASGI	Asynchronous Server Gateway Interface
XHR	XMLHttpRequest