

Ingeniería de Software

2024-2

## **Documento de diseño**

Pizarra colaborativa para computólogos

### **Equipo I**

Diego Sebastián Sánchez Correa

Mauro Emiliano Chávez Zamora

Ulises Josué Anaya Pérez

Daniel Linares Gil

Karyme Ivette Azpeitia García

Creado: 15/02/2024

Última actualización: 03/04/2024

## Objetivo

---

El objetivo de este documento es diseñar una aplicación web para la creación y edición de pizarras que permita la colaboración entre usuarios en tiempo real.

## Contexto

---

Existen múltiples aplicaciones para la edición de pizarras, como Miro, Figjam y Jamboard. Nuestro objetivo es desarrollar una herramienta similar que permita utilizar funcionalidades para específicas a nuestra área de conocimiento; lo cual incluye el poder crear gráficas y expresar expresiones/fórmulas matemáticas con facilidad.

## Herramientas elegidas

---

En la sección del front-end elegimos la herramienta VUE, la cual es un *framework* de JavaScript, que nos permite trabajar a través de componentes que nos den un diseño web modular, la idea es que VUE cargue los componentes de manera predeterminada para simular la navegación en la aplicación, esto se conoce como *SPA (Single page application)* y nos permitiría que la aplicación se ejecutara con fluidez para generar una buena impresión en la persona usuaria.

Por otro lado, para el back-end, decidimos utilizar NestJS debido a que es uno de los frameworks más populares para Javascript, además, ofrece soporte para Typescript, que, como es también usado dentro del front-end, facilita el desarrollo conjunto (si la herramienta es compartida entre back-end y front-end, el desarrollo se torna una tarea más simple) así como la incorporación de bibliotecas dentro de ambas.

## Edición concurrente de pizarras

---

Existen dos métodos principales para colaboración en tiempo real, *Operational transformation* (OT) y *Conflict free replicated data types* (CRDT).

*Operational transformation* es el primer método que surgió para colaboración en tiempo real. Consiste en que un servidor central se encarga de aplicar los cambios a los documentos y funciona como la fuente de verdad. Cuando múltiples usuarios modifican el documento de manera concurrente, el servidor se encarga de transformar las operaciones concurrentes para que se apliquen.

Por ejemplo, se tiene el texto inicial 'ttexto', y se realizan dos operaciones concurrentes. Alice elimina la letra en la posición 1, y de manera concurrente, Bob inserta una 'o' a la derecha de la posición 4. Si el servidor procesa la operación realizada por Alice antes que la de Bob, entonces debe modificar el índice de la inserción de Bob para que esta ocurra en la posición 3, porque al eliminarse la letra en la posición 1, las letras a su derecha se movieron una posición a la izquierda. OT es el método más utilizado en la actualidad y lo utilizan aplicaciones como Google Docs. Sin embargo, este método tiene algunos inconvenientes. El principal inconveniente es que necesita un servidor central que funcione como fuente de verdad, por lo cual es más difícil de escalar, y se dificulta la edición de documentos sin acceso a internet.

En los últimos años se ha desarrollado un nuevo método para la edición de documentos colaborativa, conocido como *Conflict free replicated data types* (CRDT). La idea principal de CRDT es que los documentos se almacenan en una estructura de datos que permite que los cambios realizados por múltiples usuarios se puedan sincronizar de manera consistente.

La principal ventaja de los CRDT es que están diseñados para sistemas distribuidos, y no necesitan un servidor central. Cada usuario edita su copia del documento y cuando los usuarios sincronizan los cambios, existe la garantía de que todas las copias convergen al mismo estado final. Los CRDT facilitan la edición de documentos sin conexión y también permiten el intercambio de la información en esquemas P2P donde no existe un servidor central.

La principal desventaja de los CRDT es que el tamaño del documento crece rápidamente por la información extra que almacenan. Sin embargo, en la actualidad existen implementaciones de CRDT como Yjs [3] que realizan múltiples optimizaciones. En [2], se muestra que Yjs puede manejar documentos de gran tamaño, con una complejidad en tiempo lineal para procesar el documento respecto al número de operaciones. Además, el espacio utilizado por Yjs para almacenar los documentos, es solamente de 1.5 a 2 veces mayor al utilizado en el documento original. Podría parecer que un aumento de 2 veces en el tamaño del documento puede incrementar significativamente el tráfico de red cuando cada vez que se abre el documento. Sin embargo, se puede almacenar una copia del documento de manera local, y que cuando se vuelva a abrir solo se transfieran las actualizaciones que se realizaron.

Se utilizará Yjs para manejar la edición concurrente de las pizarras. Yjs es uno de los protocolos de CRDT más populares y es utilizado por defecto por Blocksuite [1], la biblioteca que se utilizará para la edición de texto.

## Autenticación

---

La autenticación utilizará *JSON web tokens* (JWT). JWT permite verificar la identidad de un usuario sin necesidad de almacenar información de la sesión en la base de datos, por lo cual es más eficiente que si fuera necesario buscar la sesión en la base de datos. Por otro lado, la principal desventaja de utilizar JWT es que no permite un control de las sesiones de usuario, una vez se genere el *token JWT*, no es posible invalidar la sesión hasta que expire. Para mitigar lo anterior, los *tokens* se generan con un tiempo de expiración corto (tentativamente un día).

Se implementará una ruta en la API para registrar al usuario, y otra para iniciar sesión, las cuales se muestran a continuación.

### ■ POST /api/auth/login

```
1 Request:
2 {
3   "email": "user1@mail.com",
4   "password": "password"
5 }
6
7 Response:
8 STATUS: 200 OK
9 {
10   "token": "json-web-token"
11 }
12
13 Response Error:
14 STATUS: 404 NOT FOUND
```

#### ■ POST /api/auth/register

```
1 Request:
2 {
3     "username": "username"
4     "email": "user1@mail.com",
5     "password": "password"
6 }
7
8 Response:
9 STATUS: 200 OK
10 {
11     "token": "json-web-token"
12 }
13
14 Response Error:
15 STATUS: 404 NOT FOUND
```

## Autorización

---

Inicialmente, antes de la introducción del concepto de espacios de trabajo, parecía natural asociar a los usuarios con las pizarras directamente; pero la administración de permisos de pizarras, en conjunto con aquellos definidos para los espacios de trabajo, se tornó una tarea compleja e innecesaria. Si las pizarras se trataban de subconjuntos de espacios de trabajo, por qué lidiar con cada uno de los subconjuntos si podíamos establecer a los espacios de trabajo como la unidad básica de trabajo.

De esta manera, llegamos a la conclusión de que la mejor opción sería definir los permisos de escritura, lectura y administrador entre los usuarios y un espacio de trabajo. Estos serán administrados a partir de URLs (generados por el creador del espacio de trabajo) que contendrán Json Web Tokens que permitirá asociar al usuario con el espacio de trabajo particular.

## Colaboración

---

La aplicación debe ser capaz de compartir un enlace a las personas que el usuario desee se integren a su pizarra, utiliza el identificador de manera que se pueda designar permisos de lectura/edición. Si bien se generan permisos por defecto y otros se pueden asignar personalmente a cada usuario, también es necesario que la aplicación revise los permisos de acceso más restrictivos de manera que no caigamos en irregularidades de seguridad, el link poseerá la referencia al atributo *whiteboardID* en la tabla que lleva el registro, de manera que sólo será necesario revisar el acceso cuando un usuario trate de acceder a una pizarra en concreto.

La pizarra lleva registro de las personas que tienen permitido editarla a través del JWT (*Jason Web Token*), el cual sólo identifica perfiles activos, sino que también se articula con los identificadores de la pizarra para verificar qué permisos de colaboración poseen las personas usuarias. La idea es que las modificaciones también tengan un seguimiento a través del *token* de identificación.

Blocksuite será nuestra herramienta que gestionará la colaboración desde el lado del frontend a través de su integración con los componentes que muestran la pizarra en pantalla.

## Comentarios

---

La idea es que al tener una pizarra colaborativa, muchos usuarios pueden hacer cambios de manera simultanea por esa razón , nos pareció muy útil poder incluir comentarios ya que al ser un espacio de trabajo compartido , necesitamos que exista una manera de poder comunicar a los demás colaboradores alguna observación o sugerencia sobre alguna parte del contenido en la pizarra. Entonces tenemos algunas opciones para implementar los comentarios

- **Opción back-end:** La primera opción que tenemos para usar comentarios en la pizarra es crear una entidad dentro de la base de datos.
- **Opcion blocksuite:** Esta opción se basa en la posibilidad de definir los comentarios de un documento como datos embebidos en el formato del documento.

## Servidor de WebSockets

---

La naturaleza dinámica de la pizarra colaborativa hace de los *websockets* una buena opción que facilita la sincronización de cambios entre varios usuarios que trabajan en ella al mismo tiempo. Las bases de datos comunes no son lo suficientemente rápidas para administrar actualizaciones en tiempo real de aplicaciones cuyo cambio sucede muy rápidamente; la pizarra colaborativa es un ejemplo ideal de este tipo de aplicaciones.

Buscando algunas bibliotecas pensamos que es una buena opción hacer uso de *y-redis* que es una alternativa de back-end para *y-websockets* .

*Redis* (Remote Dictionary Server) es una base de datos multimodelo que se creó a partir de la idea de que el caché también puede ser una unidad de almacenamiento no volátil. Se trata de un sistema en el que la información siempre es modificada o leída de la memoria principal (RAM); esto, naturalmente, induce una latencia de milisegundos. En conjunto, escribe la información al disco para que otorgar la capacidad de reconstrucción.

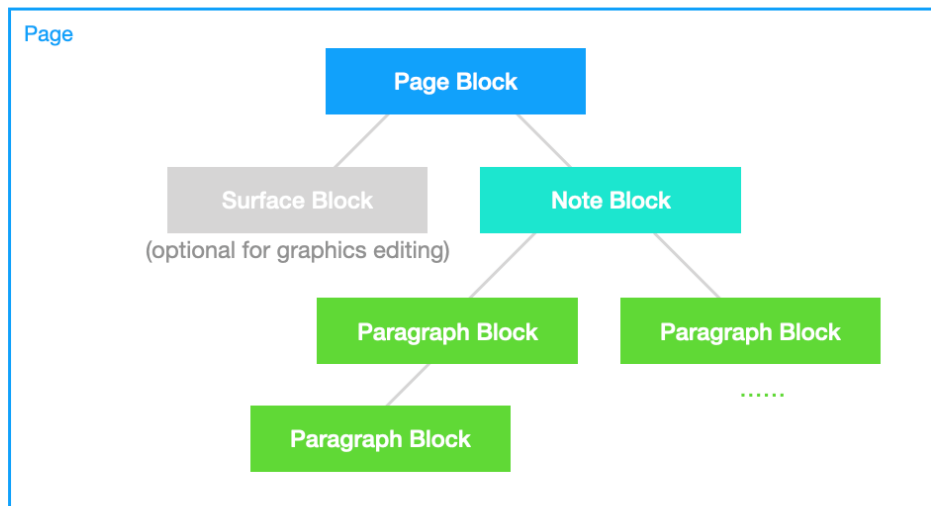
Redis es usada como caché y canal de distribución para actualizaciones en documentos, al tener una pizarra colaborativa , se generan muchísimas operaciones cada momento, por lo que hace sentido usar *redis-cache* para almacenamiento temporal para distribuir documentos lo más rápido posible.

## Edición de documentos

---

Para la edición de documentos se utilizará *Blocksuite* [1]. *Blocksuite* es un *framework* para edición de documentos moderno que incluye soporte para colaboración en tiempo real a través de *Yjs*. Además, *Blocksuite* incluye funciones de edición de texto avanzadas y soporte para dibujo sobre el canvas, lo cual lo hace muy conveniente para el desarrollo de una pizarra colaborativa.

En *Blocksuite* el código se organiza en bloques, los cuales son similares a la noción de componentes en *Vue*. El documento se representa como un árbol de bloques a partir de un bloque raíz. Por ejemplo, la imagen debajo muestra un ejemplo de la estructura del árbol de bloques.

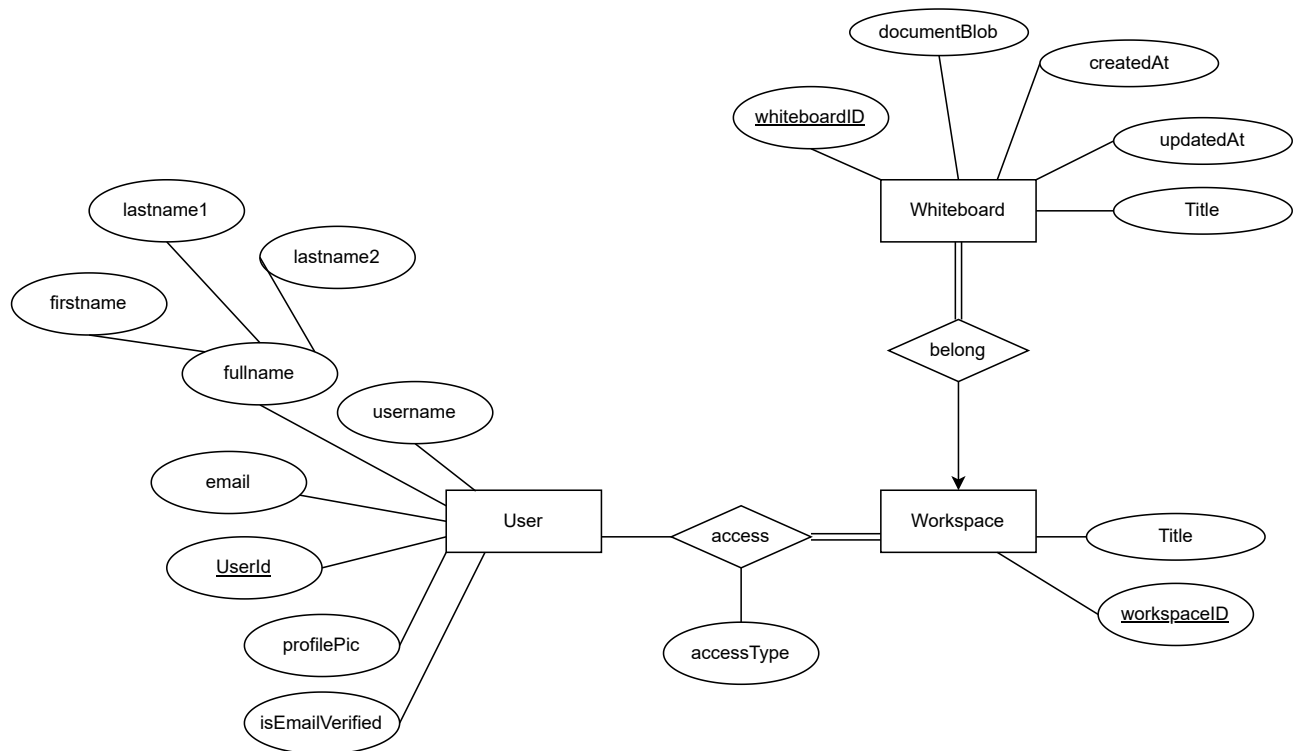


Cada bloque se compone por *schema*, *service* y *view*, los cuales corresponden al modelo, controlador y vista de MVC. Además, los bloques pueden tener asociados *fragments*, los cuales contienen funcionalidad que puede ser reutilizada en distintos bloques, por ejemplo, una barra de opciones flotante que aparece al seleccionar un bloque. La estructura de los bloques permite que se puedan extender de manera sencilla.

Blocksuite también provee una biblioteca de bloques que se pueden utilizar para la pizarra. Estos bloques se renderizan como componentes web, utilizando Lit. La principal razón por la cual utilizan componentes web es porque se pueden integrar en la mayoría de *frameworks* (Vue tiene buen soporte), y los componentes web se renderizan directamente en el DOM, a diferencia de *frameworks* como Vue, en los cuales se utiliza un DOM virtual.

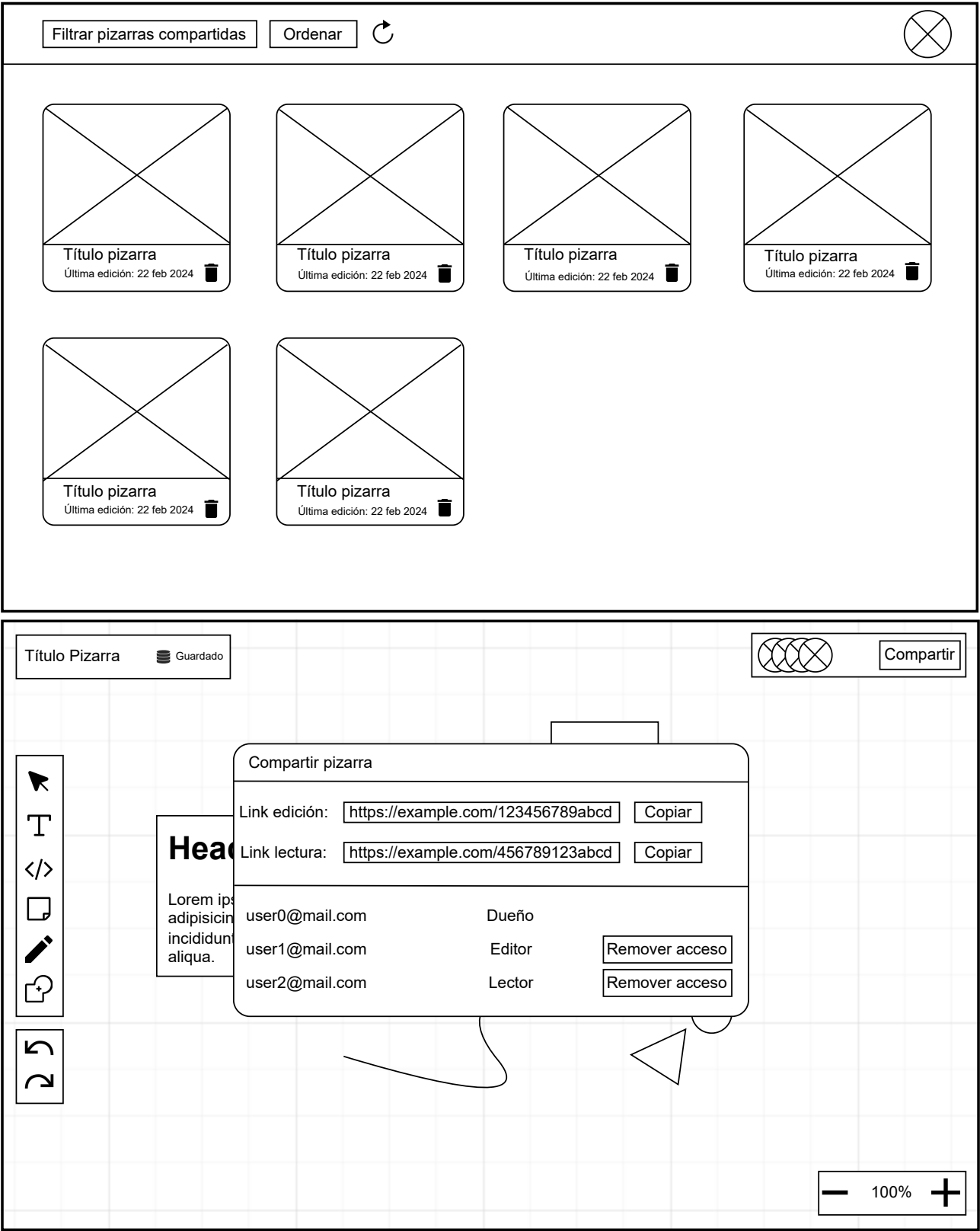
Se podría implementar un renderizado de Blocksuite para Vue, pero sería necesario reimplementar al menos la vista de todos los componentes a utilizar. Por lo anterior, se utilizará Lit para implementar los bloques de Blocksuite, y Vue para todo lo demás. Los componentes que requieran interactuar con el editor pero no sean dinámicos, como por ejemplo la barra de herramientas de edición, se implementarán en Vue.

Para configurar Blocksuite dentro de la aplicación de Vue se utilizarán dos componentes. El componente `EditorProvider` será el encargado de inicializar el editor de Blocksuite, y proveerlo como estado de la aplicación utilizando `provide`. Todos los componentes que necesitan acceder al editor necesitan ser hijos del componente `EditorProvider`. El componente `EditorContainer` se encarga de inyectar el componente web raíz del editor en el DOM.




**Nota:** Inicialmente se tiene pensado almacenar el documento en formato binario como un atributo de la entidad *Whiteboard*. Se iterará a partir del diseño inicial para optimizar cómo se almacena el documento.


Wireframes Pizarra













Título Pizarra

 Guardado



Compartir

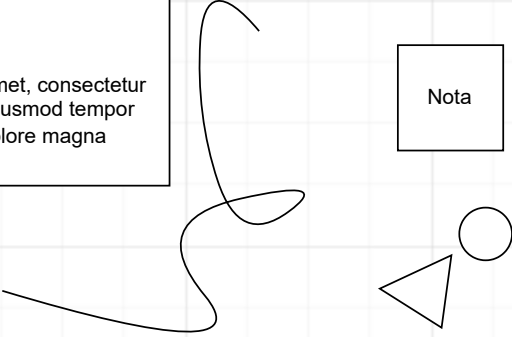




# Heading

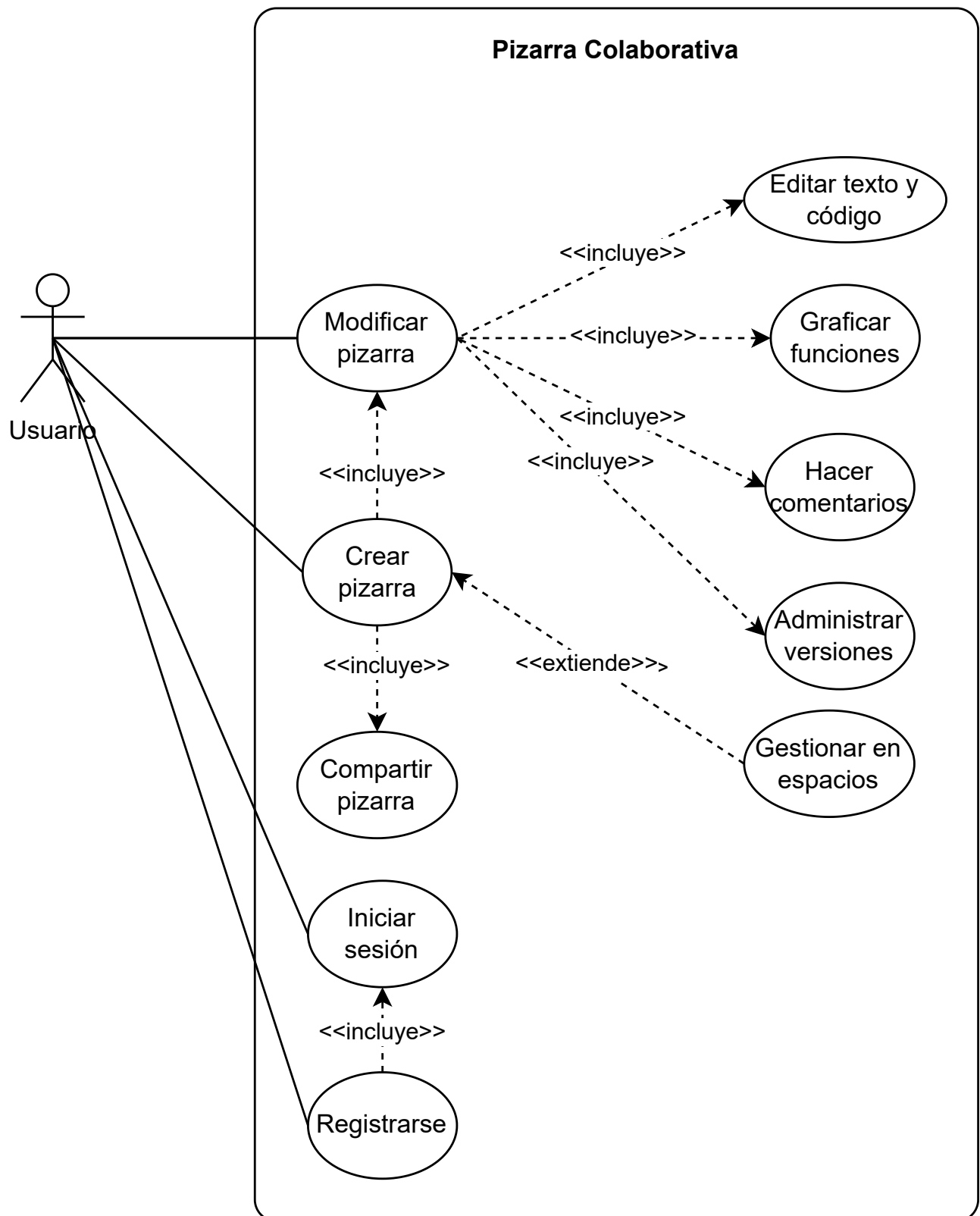
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

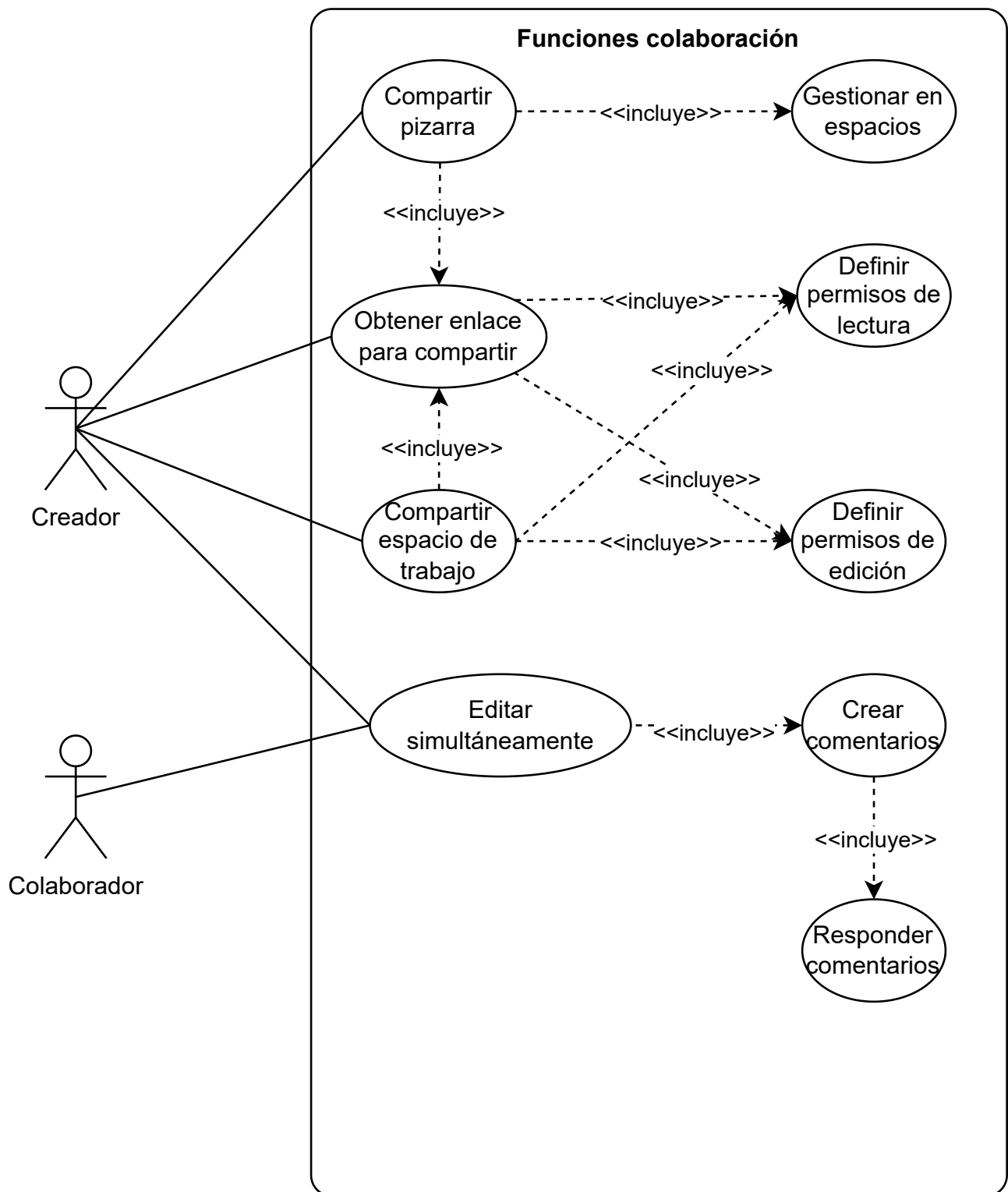
Bloque de código

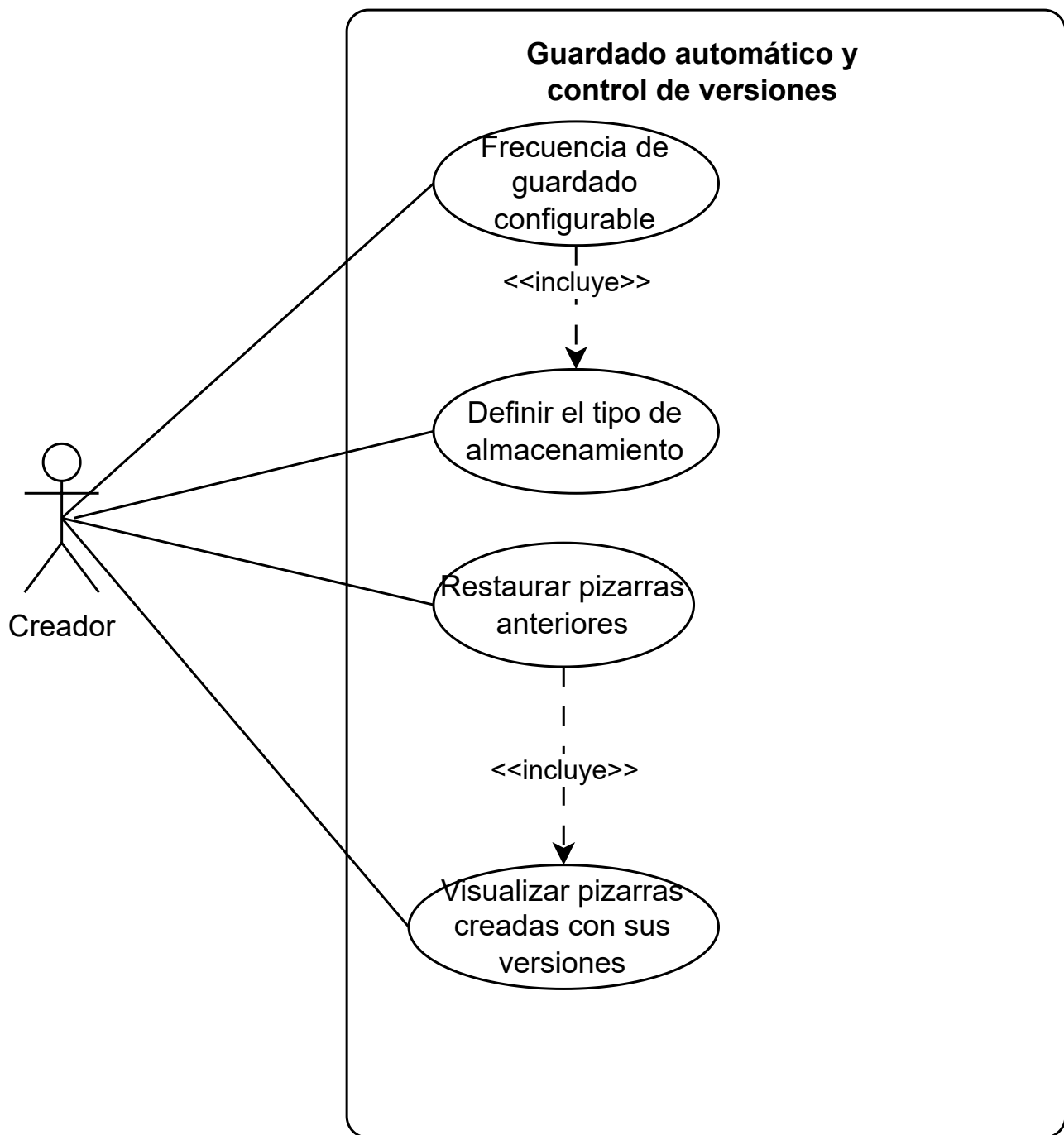
Nota



 100% 







## Referencias

- [1] BlockSuite | Content Editing Tech Stack.
- [2] Kevin Jahns. Are CRDTs suitable for shared editing?, 12 2020.
- [3] Yjs. GitHub - yjs/yjs: Shared data types for building collaborative software.