

Unlimited.

Zhijie Qian

Martin Soto

Junhan Li

Dong Xu

Create

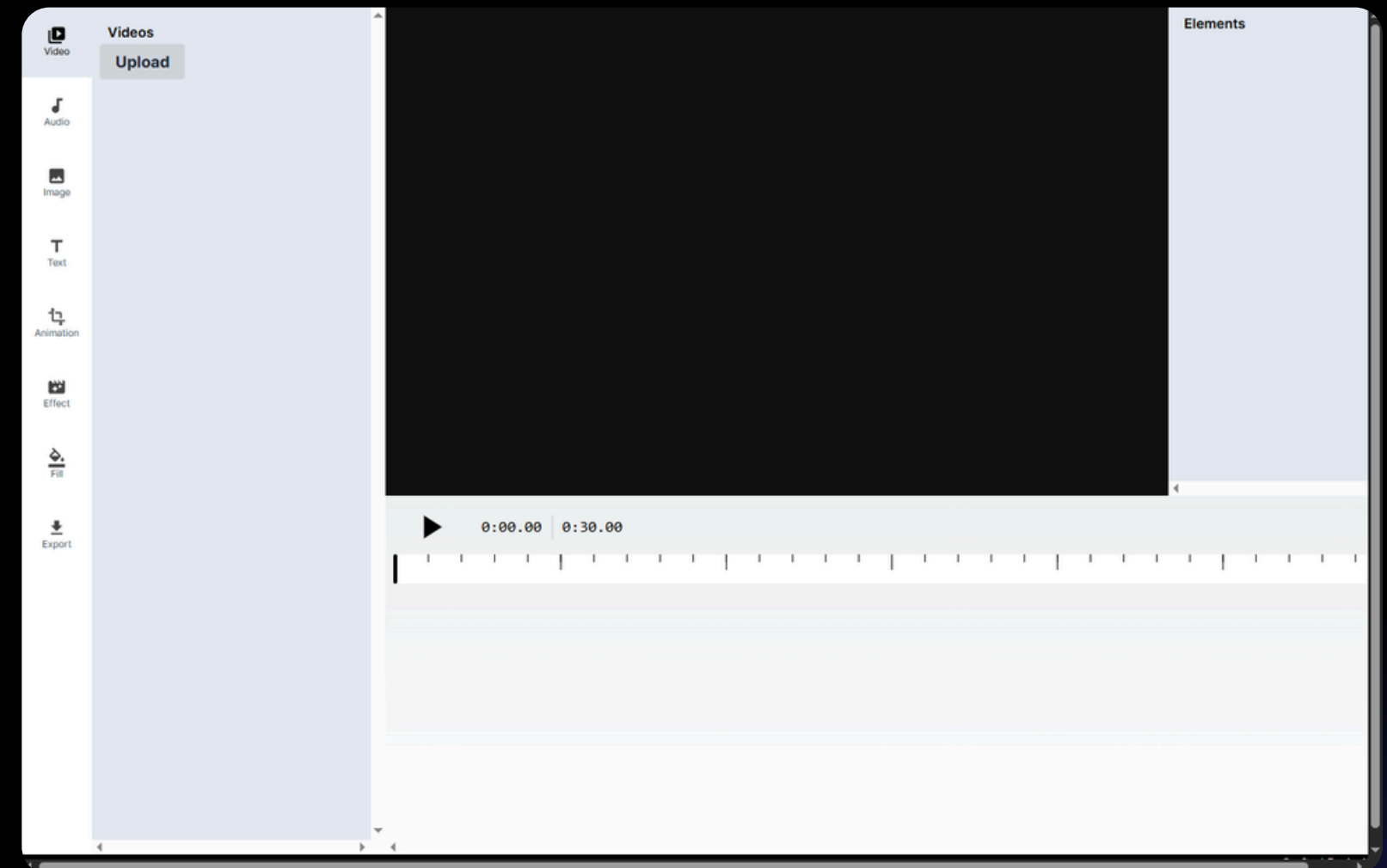
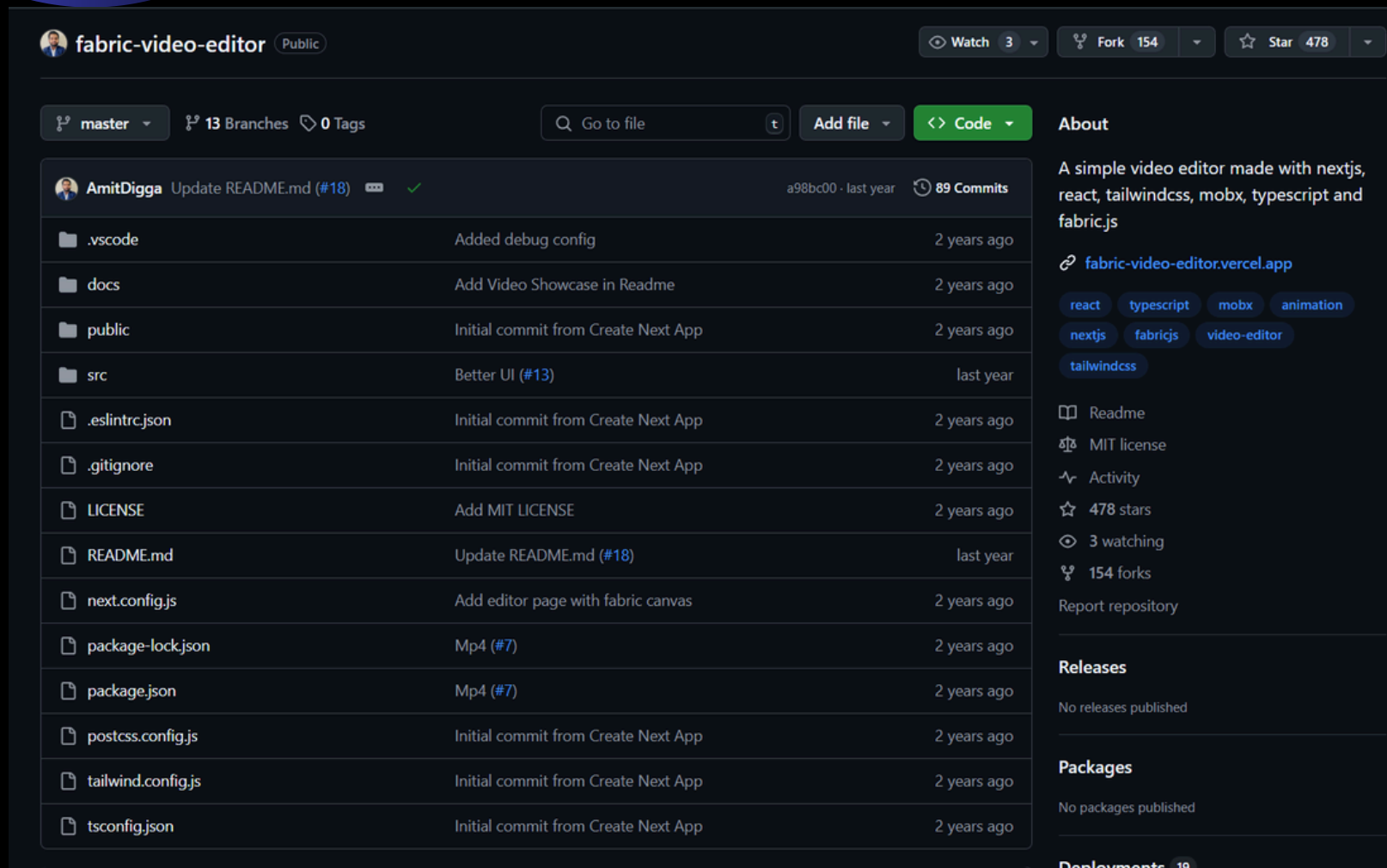
Next Genertaion

Cloud Video Editor

Next Slide

En el inicio

At the beginning, we were just chatting about whether it might be possible to edit videos online.



<https://github.com/AmitDigga/fabric-video-editor>

The live share video editor

A colaborative web-based video editor. Edit your videos anywhere, anytime wtih others.

Start to Use

Login

Demo Editor



Powerful Features
Animations, Effects, Timeline & More

USER LOGIN

We follow an Agile development methodology, working in sprints to deliver high-quality products efficiently. This approach allows for continuous feedback and iterative improvement, ensuring our projects stay aligned with client expectations.

Welcome Back

Sign in to continue to your account

Email Address

Password

[Forgot password?](#)

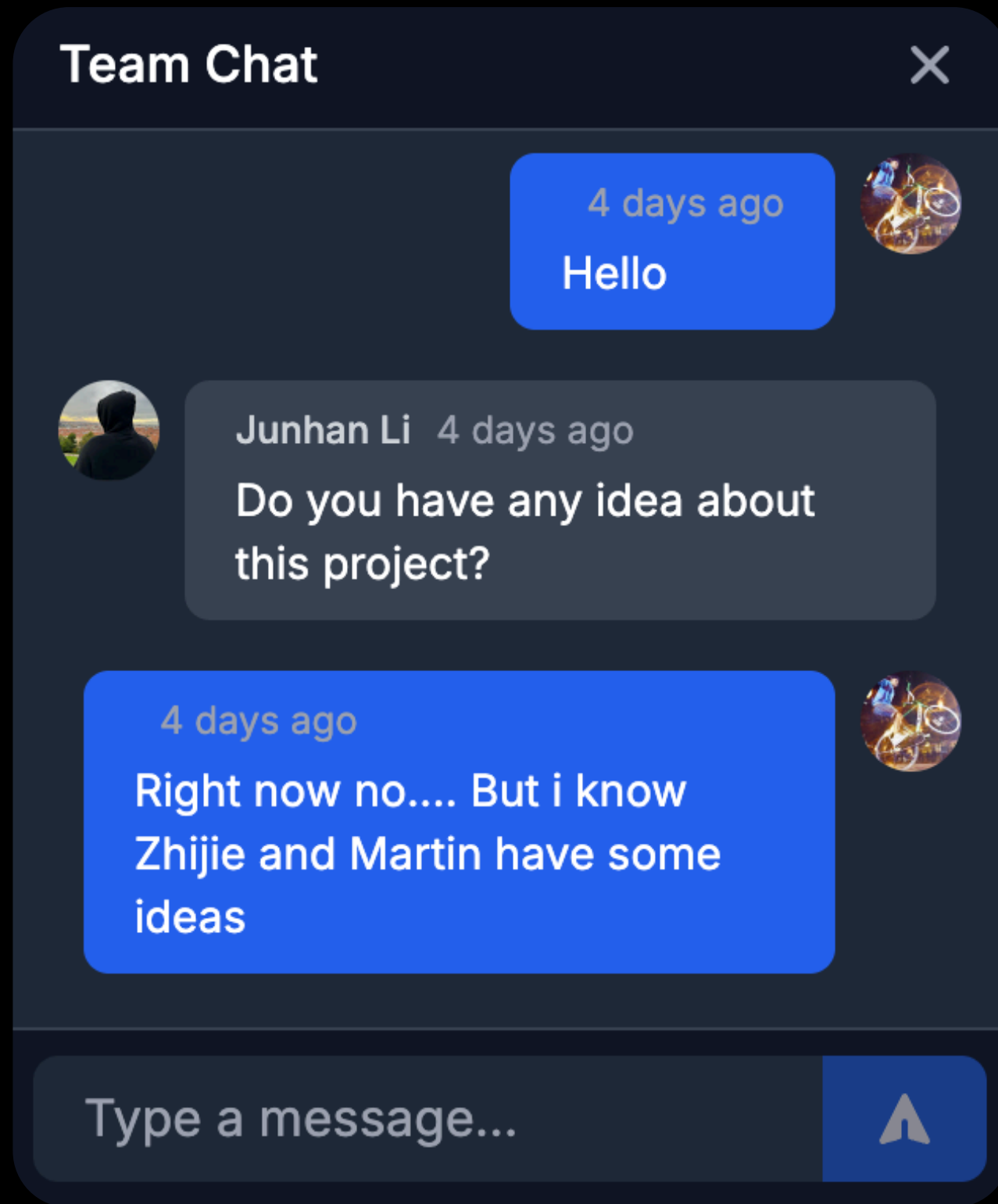
Sign In

Or continue with



Sign in with Google

Don't have an account? [Sign up](#)



Online chat box

At any time, you can communicate with your team members about the project through the online chat box located at the right corner. This chat box supports group chats with multiple users online simultaneously. Every project you create will include this feature.

Give it a try and start chatting with your friends anytime!

My Projects

Tiktok Project

Owner

A tiktok video thread

Open Editor

Created: 05/11/2025 21:24

Delete

good

Owner

good project

Open Editor

Created: 04/30/2025 15:28

Delete

Shared With Me

1

Editor

1

Open Editor

Created: 05/06/2025 17:59

123

Editor

321

Open Editor

Created: 05/04/2025 19:57

Share Project

Email Address

Please enter the email address of the coll

Permission

Editor

CancelShare

Tiktok Project

Owner

Share Project

En Workspace, el usuario puede ver sus propios proyectos y los proyectos compartidos con él

El usuario puede compartir el proyecto en la pagina de proyecto (si es propietario OWNER)

Almacenamiento - Firestore



Video

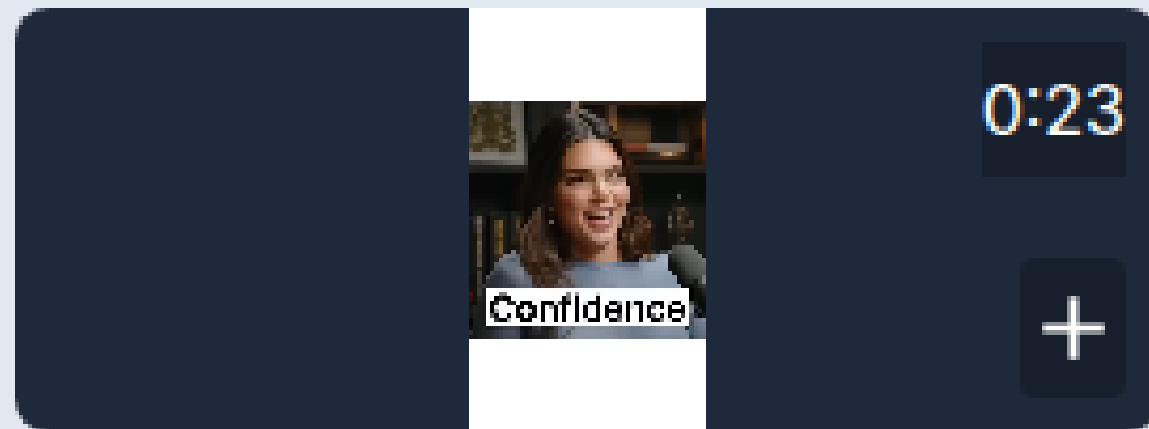


Audio



Image

Videos



Upload

```
1 const uploadFile = async (file: File, folder: string = "uploads"): Promise<string> =>
2   {
3     const storageRef = ref(storage, `${folder}/${file.name}`);
4     try {
5       // Upload the file to Firebase Storage
6       await uploadBytes(storageRef, file);
7
8       // Get the file's download URL
9       const downloadURL = await getDownloadURL(storageRef);
10      return downloadURL;
11    } catch (error) {
12      console.error("Error uploading file:", error);
13      throw error;
14    }
15  };
16
17 const getFilesFromFolder = async (folder: string = "uploads"): Promise<string[]> => {
18   const folderRef = ref(storage, folder);
19   try {
20     const result = await listAll(folderRef);
21     const urls = await Promise.all(result.items.map(async (itemRef) => {
22       return getDownloadURLFromRef(itemRef);
23     }));
24     return urls;
25   } catch (error) {
26     console.error("Error retrieving files from folder:", error);
27     throw error;
28   }
29 };
```

Sincronización - Ficheros

```
1 onSnapshot(collection(db, 'projects/${this.projectId}/videos'), (snapshot) => {
2   snapshot.docChanges().forEach((change) => {
3     const data: string = change.doc.data().url as unknown as string;
4
5     if (change.type === "added") {
6       this.addVideoResource(data, false);
7       console.log("New animation: ", change.doc.data());
8     }
9   });
10 });
11
12 onSnapshot(collection(db, 'projects/${this.projectId}/audios'), (snapshot) => {
13   snapshot.docChanges().forEach((change) => {
14     const data: string = change.doc.data().url as unknown as string;
15
16     if (change.type === "added") {
17       this.addAudioResource(data, false);
18       console.log("New animation: ", change.doc.data());
19     }
20   });
21 });
22
23 onSnapshot(collection(db, 'projects/${this.projectId}/images'), (snapshot) => {
24   snapshot.docChanges().forEach((change) => {
25     const data: string = change.doc.data().url as unknown as string;
26
27     if (change.type === "added") {
28       this.addImageResource(data, false);
29       console.log("New animation: ", change.doc.data());
30     }
31   });
32 });
```


Sincronización - Animaciones

```
1 type AnimationBase<T, P = {}> = {
2   uid: string | null,
3   id: string;
4   targetId: string;
5   duration: number;
6   type: T;
7   properties: P;
8 }
```

```
1 onSnapshot(collection(db, 'projects/${this.projectId}/animati
2   snapshot.docChanges().forEach((change) => {
3     const data: Animation = {
4       ...change.doc.data(),
5       uid: change.doc.id,
6     } as Animation;
7
8     if (change.type === "added") {
9       this.addAnimation(data, false);
10      console.log("New animation: ", change.doc.data());
11    }
12    if (change.type === "modified") {
13      this.updateAnimation(data.id, data, false);
14      console.log("Modified animation: ", change.doc.data());
15    }
16    if (change.type === "removed") {
17      this.removeAnimation(data.id);
18      console.log("Removed animation: ", change.doc.data());
19    }
20  });
21 });
```

```
1 // Store.ts
2 async addAnimation(animation: Animation, localChange: boolean = true) {
3   // ...
4   addAnimationToFirestore(animation, this.projectId);
5   this.animations = [...this.animations, animation];
6   this.refreshAnimations();
7 }
8 updateAnimation(id: string, animation: Animation, localChange: boolean =
9   // ...
10  uploadAnimationToFirebase(animation, this.projectId);
11  const index = this.animations.findIndex((a) => a.id === id);
12  this.animations[index] = animation;
13  this.refreshAnimations();
14 }
```

Sincronización - Elementos

```
1 export type EditorElementBase<T extends string, P> = {
2   uid: string | null;
3   id: string;
4   conflictId: string | null;
5   fabricObject?: fabric.Object;
6   name: string;
7   readonly type: T;
8   order: number;
9   placement: Placement;
10  timeFrame: TimeFrame;
11  properties: P;
12 };
13
14 export type EditorElement =
15   | VideoEditorElement
16   | ImageEditorElement
17   | AudioEditorElement
18   | TextEditorElement;
```

```
1 function deepCopy(element: EditorElement): EditorElement {
2   switch (element.type) {
3     case "video":
4       return {
5         ...element,
6         fabricObject: undefined, // Exclude fabricObject
7         properties: {
8           ...element.properties,
9           imageObject: undefined // Exclude imageObject
10        },
11        placement: { ...element.placement },
12        timeFrame: { ...element.timeFrame },
13      } as VideoEditorElement;
14     case "image":
15       return {
16         ...element,
17         fabricObject: undefined, // Exclude fabricObject
18         properties: {
19           ...element.properties,
```

```

91 setSelectedElement(selectedElement: EditorElement | null) {
92     var refresh = false;
93     if (this.canvas) {
94         if (selectedElement?.fabricObject){
95             this.canvas.setActiveObject(selectedElement.fabricObject);
96         }
97         else{
98             if(this.selectedElement != null){
99                 const element = this.mergeElement(
100                     this.pendingMerge[this.selectedElement.id]?.from,
101                     this.selectedElement,
102                     this.pendingMerge[this.selectedElement.id]?.to,
103                     this.pendingMerge[this.selectedElement.id]?.type
104                 );
105                 if(element){
106                     if(this.pendingMerge[this.selectedElement.id]){
107                         delete this.pendingMerge[this.selectedElement.id];
108                         this.updateEditorElement(element);
109                         // Refresh to update the element's appearance (remove editor color)
110                         this.refreshElements();
111                     }
112                 }else{
113                     const ele = removeUndefinedFields(deepCopy(this.selectedElement));
114                     ele.conflictId = this.selectedElement.id;
115                     ele.id = getUid();
116                     ele.name = `${this.selectedElement.name} (conflict)`;
117                     this.conflict[ele.id] = ele;
118                     selectedElement = this.pendingMerge[this.selectedElement.id]?.to;
119                     refresh = true;
120
121                     alert("There is a conflict with the element. Pls, review the conflict
track and delete one of them to synchronize all data.")

```

```

36 const mergeElementUpdate = function (original: EditorElement, from: EditorElement, to
: EditorElement) {
37     const diffFrom: Record<string, any> = diff(original, from);
38     const diffTo: Record<string, any> = diff(original, to);
39     if ('fabricObject' in diffFrom) {
40         delete diffFrom.fabricObject;
41     }
42     if ('fabricObject' in diffTo) {
43         delete diffTo.fabricObject;
44     }
45
46     const element = removeUndefinedFields(deepCopy(original));
47     const normalChanges = ['order', 'placement', 'timeFrame', 'properties'];
48     for (const change of normalChanges) {
49         if (
50             !mergeField(
51                 element,
52                 from,
53                 to,
54                 change,
55                 diffFrom,
56                 diffTo
57             )
58         ) {
59             return null;
60         }
61     }
62
63     return element;
64 };

```

```

66 const mergeElementDelete = function (original: EditorElement, from: EditorElement, to
: EditorElement, projectId: string | null) {
67     addElementToFirestore(to, projectId);
68     return to;
69 };

```

```

1 updateEditorElement(editorElement: EditorElement, localChange: bo
2     if(this.conflit[editorElement.id] != undefined){
3         this.conflit[editorElement.id] = editorElement;
4         return;
5     }
6
7     if(this.pendingMerge[editorElement.id] == undefined) {
8         if(!localChange){
9             const ele = this.editorElements.find((e) => e.id === edit
10             if (!ele) {
11                 return;
12             }
13             const dif = diff(ele, editorElement);
14             if ('fabricObject' in dif) {
15                 delete dif.fabricObject;
16             }
17
18             if (Object.keys(dif).length === 0) {
19                 return;
20             }
21         }else{
22             uploadElementToFirebase(editorElement, this.projectId);
23         }
24     }
25     this.setEditorElements(this.editorElements.map((element) =>
26         element.id === editorElement.id ? editorElement : element
27     ));
28 }

```

```

1 onSnapshot(collection(db, 'projects/${this.projectId}/videoEdi
2     snapshot.docChanges().forEach((change) => {
3         const data = change.doc.data();
4         const element: EditorElement = {
5             uid: change.doc.id,
6             id: data.id,
7             conflitId: null,
8             name: data.name,
9             type: data.type,
10            order: data.order,
11            placement: data.placement,
12            timeFrame: data.timeFrame,
13            properties: data.properties,
14            editPersonsId: data.editPersonsId,
15        };
16        if(data.order >= this.order){
17            this.order = data.order + 1;
18        }
19        if (change.type === "added") {
20            this.addEditorElement(element, false);
21            console.log("New element: ", change.doc.data());
22        }
23        if (change.type === "modified") {
24            if (this.selectedElement?.id === element.id) {
25                const dif = diff(this.selectedElement, element);
26                if ("fabricObject" in dif) {
27                    delete (dif as { fabricObject?: unknown }).fabricObj
28                }
29                if (Object.keys(dif).length === 0) {
30                    return;
31                }
32                if (this.pendingMerge[element.id] == undefined) {
33                    this.pendingMerge[element.id] = {
34                        from: this.selectedElement,
35                        to: element,
36                        type: "updated",
37                    };
38                } else {
39                    this.pendingMerge[element.id].to = element;
40                    this.pendingMerge[element.id].type = "updated";
41                }
42            } else {
43                this.updateEditorElement(element, false);

```

Sincronización - Otros

```
1 // Utils.ts
2 const addBackgroundToFirestore = async function (background: string, projectId:
3   string | null) {
4   if (!projectId) {
5     console.error('Project ID is null. Cannot update background in Firestore.');
```

```
6   }
7
8   const db = getFirestore();
9   const projectDocRef = doc(db, 'projects/${projectId}');
```

```
10  try {
11    await updateDoc(projectDocRef, { background });
12    console.log('Background updated successfully.');
```

```
13  } catch (error) {
14    console.error('Error updating background in Firestore:', error);
15  }
16 };
```

```
53 // Store.ts
54 const projectDocRef = doc(db, 'projects/${this.projectId}');
```

```
55 onSnapshot(projectDocRef, (docSnapshot) => {
56   if (docSnapshot.exists()) {
57     const data = docSnapshot.data();
58
59     if (data.background !== undefined) {
60       this.setBackgroundColor(data.background, false);
61       console.log("Background updated: ", data.background);
62     }
63
64     if (data.times !== undefined) {
65       this.setMaxTime(data.times, false);
66       console.log("Times updated: ", data.times);
67     }
68   } else {
69     console.error("Project document does not exist.");
70   }
71 });
```

Unlimited.

Zhijie Qian

Martin Soto

Junhan Li

Dong Xu

Create

Demostración

Hosting



<https://cloud-video-editor.vercel.app>



Unlimited.

Zhijie Qian

Martin Soto

Junhan Li

Dong Xu

Create

Thank You

<https://cloud-video-editor.vercel.app>