Unlimited.

Zhijie Qian Martin Soto Junhan Li Dong Xu

Create

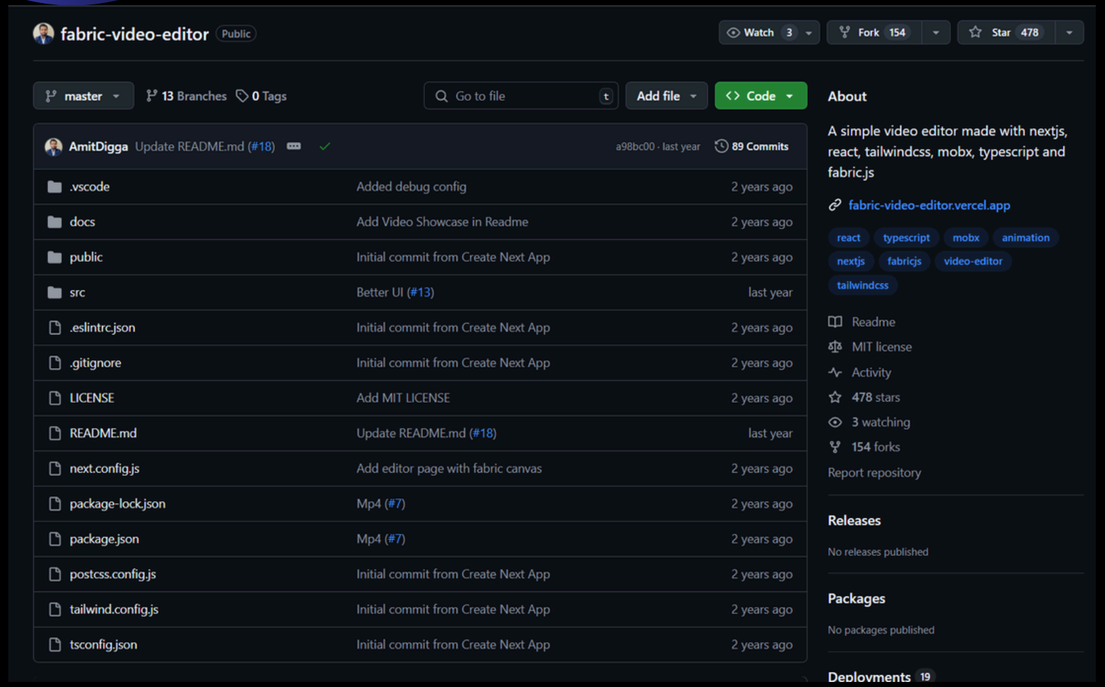
Next Genertaion

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

Cloud Video Editor

Login Signup

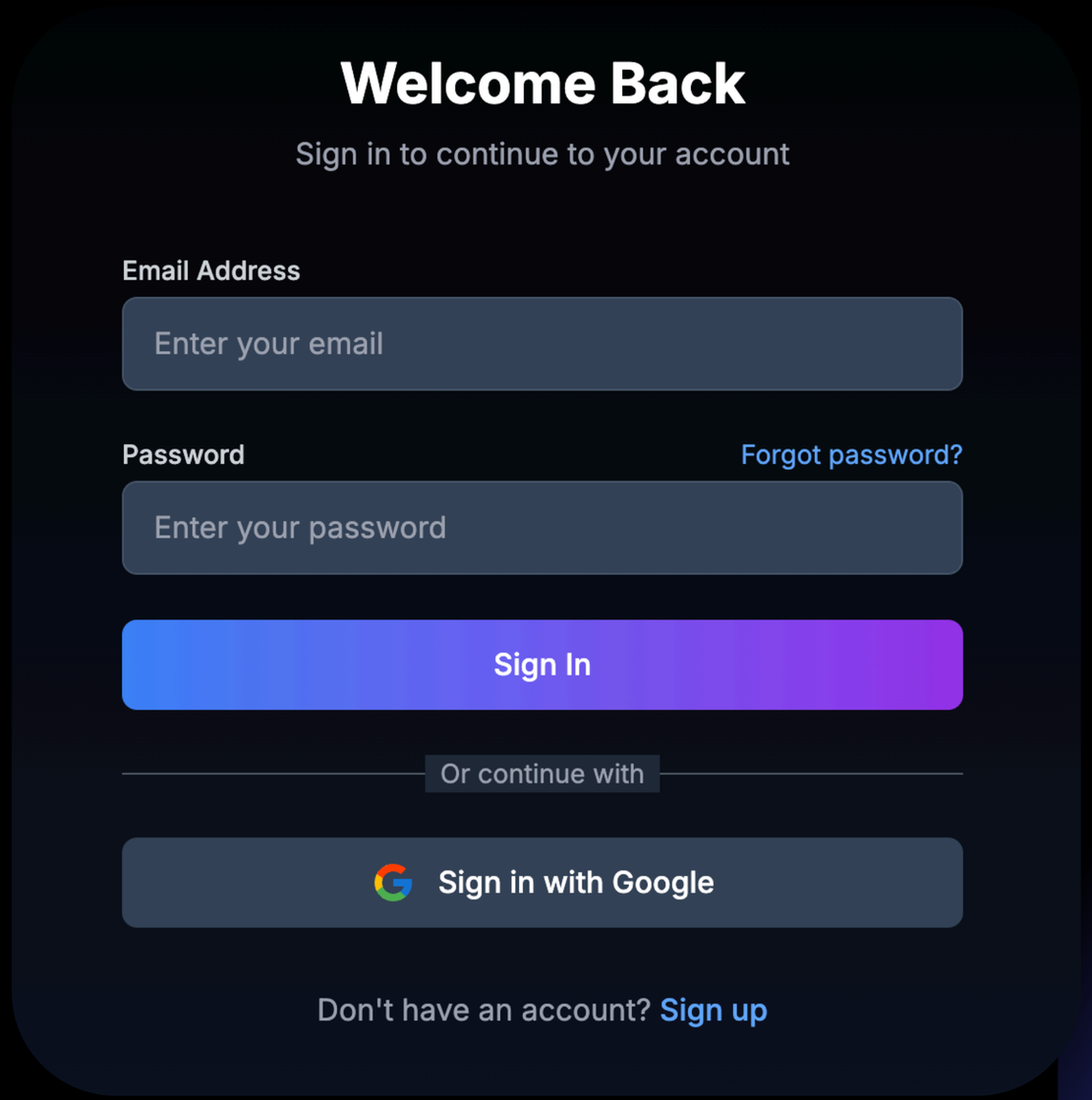
The live share video editor

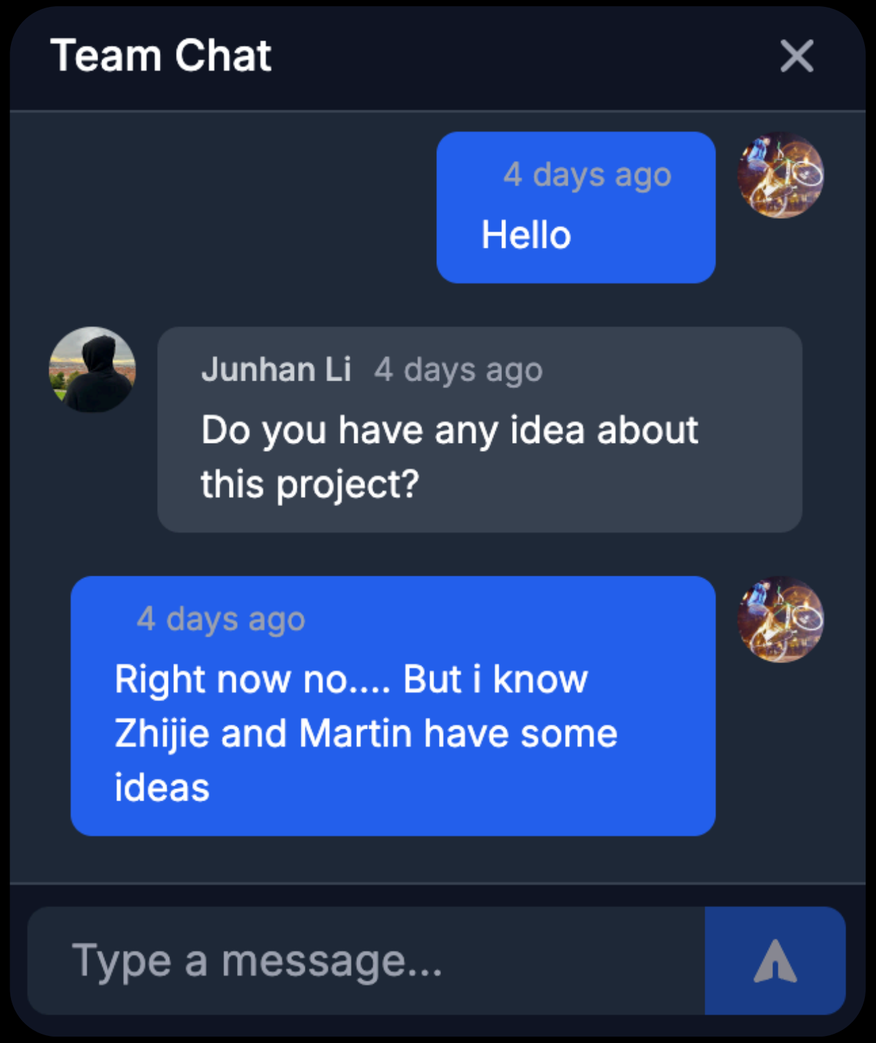
A colaborative web-based video editor. Edit your videos anywhere, anytime wtih others.

Start to Use Login Demo Editor

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.

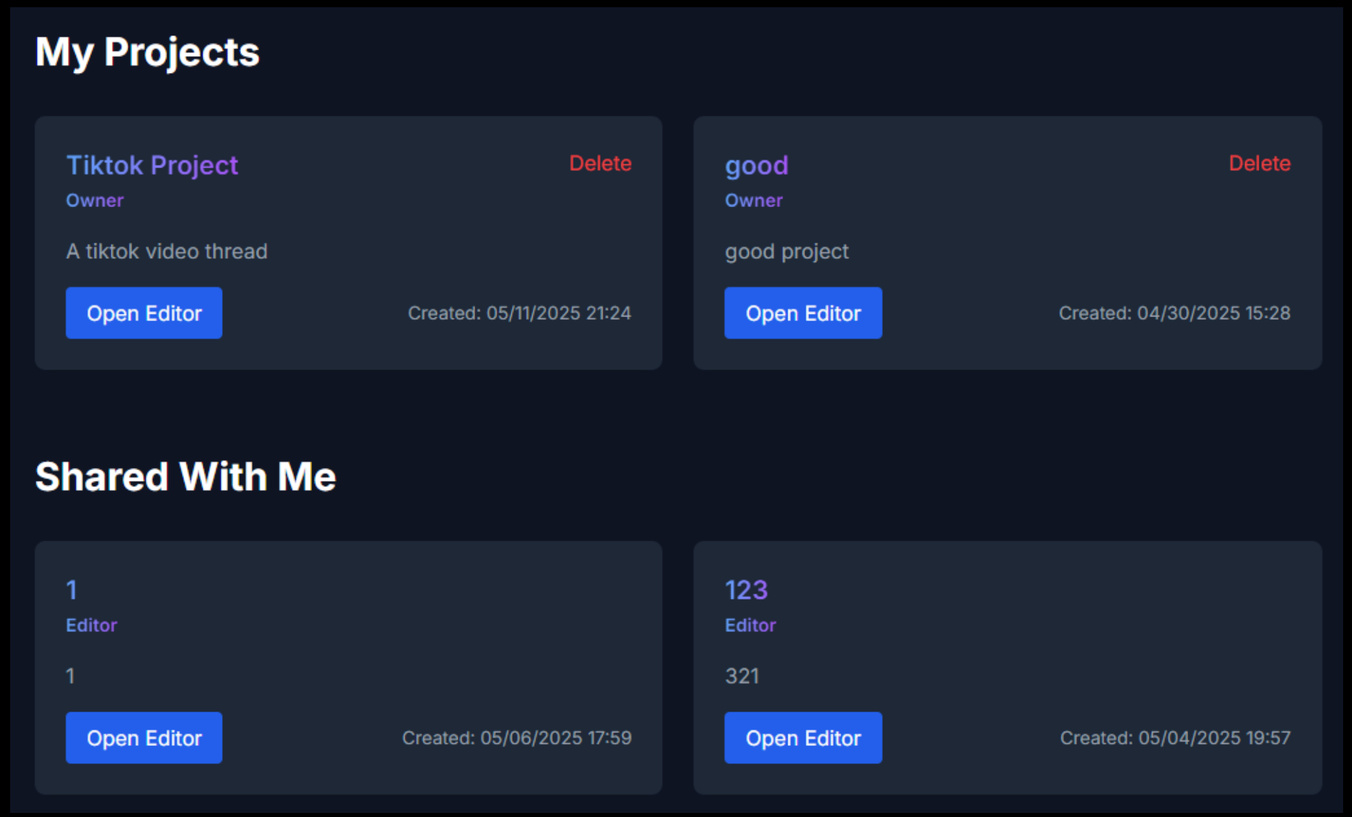




Online chat box

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.

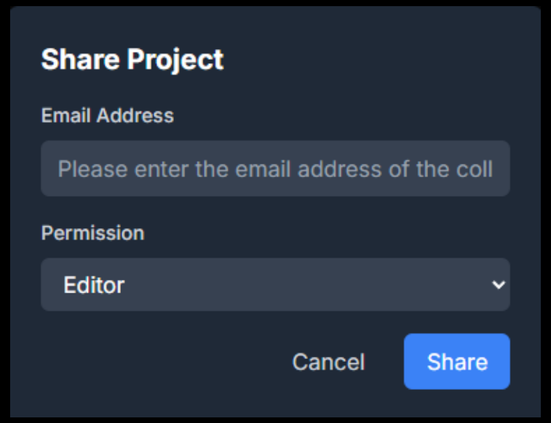
Give it a try and start chatting with your friends anytime!



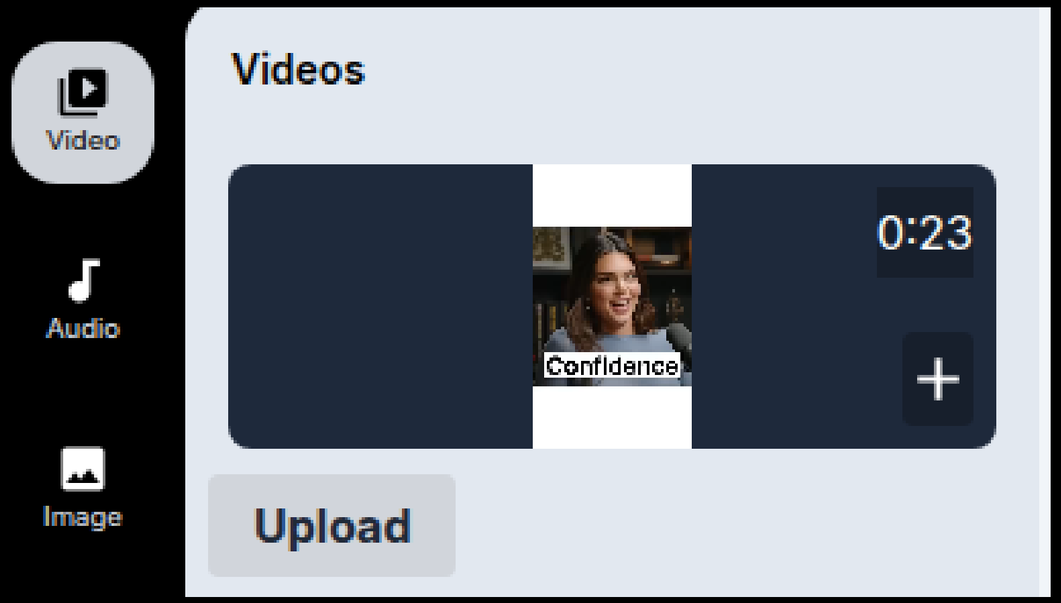
Workspace

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




```tsx
const uploadFile = async (file: File, folder: string = "uploads"): Promise<string> =>
    {
  const storageRef = ref(storage, `${folder}/${file.name}`);
  try {
    // Upload the file to Firebase Storage
    await uploadBytes(storageRef, file);

    // Get the file's download URL
    const downloadURL = await getDownloadURL(storageRef);
    return downloadURL;
  } catch (error) {
    console.error("Error uploading file:", error);
    throw error;
  }
};

const getFilesFromFolder = async (folder: string = "uploads"): Promise<string[]> => {
  const folderRef = ref(storage, folder);
  try {
    const result = await listAll(folderRef);
    const urls = await Promise.all(result.items.map(async (itemRef) => {
      return getDownloadURLFromRef(itemRef);
    }));
    return urls;
  } catch (error) {
    console.error("Error retrieving files from folder:", error);
    throw error;
  }
};
```

Sincronización - Ficheros

```
onSnapshot(collection(db, `projects/${this.projectId}/videos`), (snapshot) => {
  snapshot.docChanges().forEach((change) => {
    const data: string = change.doc.data().url as unknown as string;

    if (change.type === "added") {
      this.addVideoResource(data, false);
      console.log("New animation: ", change.doc.data());
    }
  });
});

onSnapshot(collection(db, `projects/${this.projectId}/audios`), (snapshot) => {
  snapshot.docChanges().forEach((change) => {
    const data: string = change.doc.data().url as unknown as string;

    if (change.type === "added") {
      this.addAudioResource(data, false);
      console.log("New animation: ", change.doc.data());
    }
  });
});

onSnapshot(collection(db, `projects/${this.projectId}/images`), (snapshot) => {
  snapshot.docChanges().forEach((change) => {
    const data: string = change.doc.data().url as unknown as string;

    if (change.type === "added") {
      this.addImageResource(data, false);
      console.log("New animation: ", change.doc.data());
    }
  });
});
```

Sincronización - Animaciones

```ts
type AnimationBase<T, P = {}> = {
  uid: string | null,
  id: string;
  targetId: string;
  duration: number;
  type: T;
  properties: P;
}
```

```ts
onSnapshot(collection(db, 'projects/${this.projectId}/animati
  snapshot.docChanges().forEach((change) => {
    const data: Animation = {
      ...change.doc.data(),
      uid: change.doc.id,
    } as Animation;

    if (change.type === "added") {
      this.addAnimation(data, false);
      console.log("New animation: ", change.doc.data());
    }
    if (change.type === "modified") {
      this.updateAnimation(data.id, data, false);
      console.log("Modified animation: ", change.doc.data());
    }
    if (change.type === "removed") {
      this.removeAnimation(data.id);
      console.log("Removed animation: ", change.doc.data());
    }
  });
});
```

```ts
// Store.ts
async addAnimation(animation: Animation, localChange: boolean = true) {
    // ...
    addAnimationToFirestore(animation, this.projectId);
    this.animations = [...this.animations, animation];
    this.refreshAnimations();
}
updateAnimation(id: string, animation: Animation, localChange: boolean =
    // ...
    uploadAnimationToFirebase(animation, this.projectId);
    const index = this.animations.findIndex((a) => a.id === id);
    this.animations[index] = animation;
    this.refreshAnimations();
}
```

Sincronización - Elementos

```ts
export type EditorElementBase<T extends string, P> = {
  uid: string | null;
  id: string;
  conflitId: string | null;
  fabricObject?: fabric.Object;
  name: string;
  readonly type: T;
  order: number;
  placement: Placement;
  timeFrame: TimeFrame;
  properties: P;
};

export type EditorElement =
  | VideoEditorElement
  | ImageEditorElement
  | AudioEditorElement
  | TextEditorElement;
```

```ts
function deepCopy(element: EditorElement): EditorElement {
  switch (element.type) {
    case "video":
      return {
        ...element,
        fabricObject: undefined, // Exclude fabricObject
        properties: {
          ...element.properties,
          imageObject: undefined // Exclude imageObject
        },
        placement: { ...element.placement },
        timeFrame: { ...element.timeFrame },
      } as VideoEditorElement;
    case "image":
      return {
        ...element,
        fabricObject: undefined, // Exclude fabricObject
        properties: {
```

```
setSelectedElement(selectedElement: EditorElement | null) {
    var refresh = false;
    if (this.canvas) {
      if (selectedElement?.fabricObject){
        this.canvas.setActiveObject(selectedElement.fabricObject);
      }
      else{
        if(this.selectedElement != null){
          const element = this.mergeElement(
            this.pendingMerge[this.selectedElement.id]?.from,
            this.selectedElement,
            this.pendingMerge[this.selectedElement.id]?.to,
            this.pendingMerge[this.selectedElement.id]?.type
          );
          if(element){
            if(this.pendingMerge[this.selectedElement.id]){
              delete this.pendingMerge[this.selectedElement.id];
              this.updateEditorElement(element);
              // Refresh to update the element's appearance (remove editor color)
              this.refreshElements();
            }
          }else{
            const ele = removeUndefinedFields(deepCopy(this.selectedElement));
            ele.conflitId = this.selectedElement.id;
            ele.id = getUid();
            ele.name = `${this.selectedElement.name} (conflit)`;
            this.conflit[ele.id] = ele;
            selectedElement = this.pendingMerge[this.selectedElement.id]?.to;
            refresh = true;

            alert("There is a conflict with the element. Pls, review the conflict
    track and delete one of them to sryncronize all data.")
```

```
const mergeElementUpdate = function (original: EditorElement, from: EditorElement,
    : EditorElement) {
  const diffFrom: Record<string, any> = diff(original, from);
  const diffTo: Record<string, any> = diff(original, to);
  if ('fabricObject' in diffFrom) {
    delete diffFrom.fabricObject;
  }
  if ('fabricObject' in diffTo) {
    delete diffTo.fabricObject;
  }

  const element = removeUndefinedFields(deepCopy(original));
  const normalChanges = ['order', 'placement', 'timeFrame', 'properties'];
  for (const change of normalChanges) {
    if (
      !mergeField(
        element,
        from,
        to,
        change,
        diffFrom,
        diffTo
      )
    ) {
      return null;
    }
  }

  return element;
};
```

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

```
updateEditorElement(editorElement: EditorElement, localChange: bo
    if(this.conflit[editorElement.id] != undefined){
      this.conflit[editorElement.id] = editorElement;
      return;
    }

    if(this.pendingMerge[editorElement.id] == undefined) {
      if(!localChange){
        const ele = this.editorElements.find((e) => e.id === edit
        if (!ele) {
          return;
        }
        const dif = diff(ele, editorElement);
        if ('fabricObject' in dif) {
          delete dif.fabricObject;
        }

        if (Object.keys(dif).length === 0) {
          return;
        }
      }else{
        uploadElementToFirebase(editorElement, this.projectId);
      }
    }
    this.setEditorElements(this.editorElements.map((element) =>
      element.id === editorElement.id ? editorElement : element
    ));
}
```

```
onSnapshot(collection(db, 'projects/${this.projectId}/videoEdi
  snapshot.docChanges().forEach((change) => {
    const data = change.doc.data();
    const element: EditorElement = {
      uid: change.doc.id,
      id: data.id,
      conflitId: null,
      name: data.name,
      type: data.type,
      order: data.order,
      placement: data.placement,
      timeFrame: data.timeFrame,
      properties: data.properties,
      editPersonsId: data.editPersonsId,
    };
    if(data.order >= this.order){
      this.order = data.order + 1;
    }
    if (change.type === "added") {
      this.addEditorElement(element, false);
      console.log("New element: ", change.doc.data());
    }
    if (change.type === "modified") {
      if (this.selectedElement?.id === element.id) {
        const dif = diff(this.selectedElement, element);
        if ("fabricObject" in dif) {
          delete (dif as { fabricObject?: unknown }).fabricObj
        }
        if (Object.keys(dif).length === 0) {
          return;
        }
        if (this.pendingMerge[element.id] == undefined) {
          this.pendingMerge[element.id] = {
            from: this.selectedElement,
            to: element,
            type: "updated",
          };
        } else {
          this.pendingMerge[element.id].to = element;
          this.pendingMerge[element.id].type = "updated";
        }
      } else {
        this.updateEditorElement(element, false);
```

Sincronización - Otros

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

  const db = getFirestore();
  const projectDocRef = doc(db, `projects/${projectId}`);
  try {
    await updateDoc(projectDocRef, { background });
    console.log('Background updated successfully.');
  } catch (error) {
    console.error('Error updating background in Firestore:', error);
  }
};
```

```ts
// Store.ts
const projectDocRef = doc(db, `projects/${this.projectId}`);
onSnapshot(projectDocRef, (docSnapshot) => {
  if (docSnapshot.exists()) {
    const data = docSnapshot.data();

    if (data.background !== undefined) {
      this.setBackgroundColor(data.background, false);
      console.log("Background updated: ", data.background);
    }

    if (data.times !== undefined) {
      this.setMaxTime(data.times, false);
      console.log("Times updated: ", data.times);
    }
  } else {
    console.error("Project document does not exist.");
  }
});
```

Unlimited. Zhijie Qian Martin Soto Junhan Li Dong Xu Create

Demostración

Hosting

Vercel

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

Unlimited.

Zhijie Qian Martin Soto Junhan Li Dong Xu

Create

Thank **You**

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