ClipForge

Build a Desktop Video Editor in 72 Hours

Background

CapCut transformed video editing by making it accessible and intuitive. What used to require complex desktop software and hours of learning could now be done in minutes on your phone or computer. The secret? A streamlined interface focused on the essentials: trim, splice, add effects, export.

Now imagine building that experience from scratch for the desktop. A native application where creators can record their screen, import clips, arrange them on a timeline, and export professional-looking videos — all without leaving the app.

This project challenges you to build a production-grade desktop video editor in just 3 days, shipping before you relocate to Austin on Thursday, October 30th.

Why This Matters

Video is the dominant content format. Every creator, educator, and professional needs video editing tools. By building a desktop editor, you'll understand:

- Working with media streams and file handling
- Building performant timeline interfaces
- Desktop app development with Electron or Tauri
- Real-time preview and rendering pipelines
- Exporting and encoding video formats

Project Overview

This is a **compressed 3-day sprint** with two key deadlines:

1. **MVP**: Tuesday, October 28th at 10:59 PM CT
2. **Final Submission**: Wednesday, October 29th at 10:59 PM CT

Project introduction: Monday morning, October 27th

You'll build a desktop video editor using **Electron or Tauri** that handles screen recording, webcam capture, clip import, timeline editing, and video export.

MVP Requirements (Tuesday 10:59 PM CT)

This is a hard gate. To pass the MVP checkpoint, you must have:

- Desktop app that launches (Electron or Tauri)

- Basic video import (drag & drop or file picker for MP4/MOV)
- Simple timeline view showing imported clips
- Video preview player that plays imported clips
- Basic trim functionality (set in/out points on a single clip)
- Export to MP4 (even if just one clip)
- Built and packaged as a native app (not just running in dev mode)

The MVP proves you can handle media files in a desktop context. Focus on the fundamentals: import, display, trim, export.

Example Architecture

At minimum, you should have:

- Desktop framework (Electron with React/Vue or Tauri with your frontend choice)
- Media processing library (FFmpeg via fluent-ffmpeg, @ffmpeg/ffmpeg, or native Tauri commands)
- Timeline UI component (canvas-based or DOM-based with draggable clips)
- Video player (HTML5 video element or VideoJS)
- File system access for import/export

Core Features (Full Submission)

Recording Features

Your app must support native recording:

- Screen recording (full screen or window selection)
- Webcam recording (access system camera)
- Simultaneous screen + webcam (picture-in-picture style)
- Audio capture from microphone
- Record, stop, and save recordings directly to timeline

Technical hint:

For Electron: Use desktopCapturer API to list available screens/windows, then pass source to getUserMedia(). For webcam, use standard navigator.mediaDevices.getUserMedia().

For Tauri: You'll need to invoke Rust commands to access screen capture (using platform-specific APIs like AVFoundation on macOS or Windows.Graphics.Capture on Windows). For webcam, use web APIs via getUserMedia() in the frontend.

Alternative: Use navigator.mediaDevices.getDisplayMedia() in the renderer process for screen sharing (works in both frameworks but may have limitations on window selection).

Import & Media Management

Support multiple ways to add content:

- Drag and drop video files (MP4, MOV, WebM)
- File picker for importing from disk
- Media library panel showing imported clips
- Thumbnail previews of clips
- Basic metadata display (duration, resolution, file size)

Timeline Editor
This is the heart of your application:

- Visual timeline with playhead (current time indicator)
- Drag clips onto timeline
- Arrange clips in sequence
- Trim clips (adjust start/end points)
- Split clips at playhead position
- Delete clips from timeline
- Multiple tracks (at least 2: main video + overlay/PiP)
- Zoom in/out on timeline for precision editing
- Snap-to-grid or snap-to-clip edges

Preview & Playback

- Real-time preview of timeline composition
- Play/pause controls
- Scrubbing (drag playhead to any position)
- Audio playback synchronized with video
- Preview window shows current frame at playhead

Export & Sharing
Users need to get their videos out:

- Export timeline to MP4
- Resolution options (720p, 1080p, or source resolution)
- Progress indicator during export
- Save to local file system
- Bonus: Upload to cloud storage (Google Drive, Dropbox) or generate shareable link

Technical hint: FFmpeg is essential for encoding. You'll need to stitch clips, apply cuts, and render to final format.

Additional Features (Stretch Goals)
If you finish core features early:

- Text overlays with custom fonts and animations
- Transitions between clips (fade, slide, etc.)
- Audio controls (volume adjustment, fade in/out)
- Filters and effects (brightness, contrast, saturation)
- Export presets for different platforms (YouTube, Instagram, TikTok)
- Keyboard shortcuts for common actions
- Auto-save project state
- Undo/redo functionality

Testing Scenarios
We'll test your app with:

- Recording a 30-second screen capture and adding it to timeline

- Importing 3 video clips and arranging them in sequence
- Trimming clips and splitting at various points
- Exporting a 2-minute video with multiple clips
- Using webcam recording and overlay on screen recording
- Testing on both Mac and Windows if possible

Performance Targets

- Timeline UI remains responsive with 10+ clips
- Preview playback is smooth (30 fps minimum)
- Export completes without crashes
- App launch time under 5 seconds
- No memory leaks during extended editing sessions (test for 15+ minutes)
- File size: Exported videos should maintain reasonable quality (not bloated)

Technical Stack

Recommended (not required):

Desktop Framework: Electron (more mature, larger bundle) or Tauri (Rust-based, smaller, faster)

Frontend: React, Vue, Svelte, or Vanilla JS

Media Processing: FFmpeg (fluent-ffmpeg for Node, @ffmpeg/ffmpeg for browser context, or native commands in Tauri)

Timeline UI: HTML5 Canvas, Fabric.js, Konva.js, or custom CSS/DOM solution

Video Player: HTML5 <video> element, Video.js, or Plyr

Use whatever stack helps you ship fastest. Time is your constraint.

Build Strategy

1. ### Start with Import and Preview
   Get video files loading and displaying first. This validates your media pipeline.
2. ### Build the Timeline
   Timeline is your core interface. Get clips draggable, trimmable, and deletable before adding complex features.
3. ### Add Recording Last
   Recording is not critical for MVP. Once import/timeline/export works, add screen and webcam capture.
4. ### Test Export Early
   FFmpeg encoding can be tricky. Test export with a single clip as soon as possible to avoid surprises.
5. ### Package and Test on Real Hardware
   Don't wait until the last minute to build your distributable. Test the packaged app, not just dev mode.

Submission Requirements

Submit the following by Wednesday, October 29th at 10:59 PM CT:

- GitHub Repository with setup instructions and architecture overview

- Demo Video (3–5 minutes) showing import, recording, editing, and export
- Packaged desktop app (provide download link or instructions to build)
- README with clear instructions for running and building the app

For desktop apps: Host your distributable on GitHub Releases, Google Drive, or Dropbox. Include build instructions so we can compile if needed.

Final Note

72 hours isn't long at all. This project is about velocity and pragmatism. You're building a real desktop application in under three days.

A simple, working video editor that can record, arrange clips, and export beats a feature-rich app that crashes or doesn't package correctly.

Focus on the core loop: Record → Import → Arrange → Export.

Desktop apps are hard. Video processing is hard. You're doing both in 3 days.

Remember: Just submit. Don't miss a submission.