Video Duration Adjuster (TimeShift) - Software Design Document (SDD)

0. Document Control

- Project: Video Duration Adjuster (UI branding: **TimeShift**)
- Repository layout: `https://github.com/milochen0418/video_duration_adjuster`
- Primary runtime: Python 3.11 + Reflex
- Purpose: lets users change a video's total duration by specifying a **target duration**.

1. Overview

1.1 Purpose

Video Duration Adjuster is a **Reflex** web application that lets users change a video's total duration by specifying a **target duration**. The system then:

- computes the required playback speed,
- adjusts **video timestamps**,
- optionally performs **optical-flow frame interpolation** when slowing down,
- and adjusts **audio tempo** while preserving pitch (prefer **Rubber Band**, otherwise fallback to `atempo` chaining).

1.2 Primary Use Cases

1. Upload a video and read metadata (duration/resolution/FPS/size/audio existence).
2. Set a target duration (either time format or total seconds).
3. Generate a **5-second preview** to validate pacing/audio quality.
4. Process and download the **full output**.

1.3 Non-Goals

- User accounts/authentication
- Persistent asset management (TTL/cleanup policy is not implemented)
- Multi-file batch pipelines
- Deep-learning interpolation as default (RIFE exploration exists only as a referenced PR)

—

2. Tech Stack

2.1 Application

- Python `~3.11` (per `pyproject.toml`)
- Reflex `0.8.24.post1`
- granian `2.7.1`
- fastapi
- psutil
- reflex-mouse-track

2.2 Video Tooling

- `ffmpeg` + `ffprobe`
- On macOS, project recommends **Homebrew `ffmpeg-full`** because it typically includes the `rubberband` filter (compiled with `--enable-librubberband`).

—

3. Repository Layout

```
video_duration_adjuster/
  rxconfig.py
  pyproject.toml
  README.md
  reflex_rerun.sh
  proj_reinstall.sh
  run_test_suite.sh
  apt-packages.txt
  assets/
  video_duration_adjuster/
    video_duration_adjuster.py
    states/
      video_state.py
    components/
      navbar.py
      upload_zone.py
      time_controls.py
      video_preview.py
```

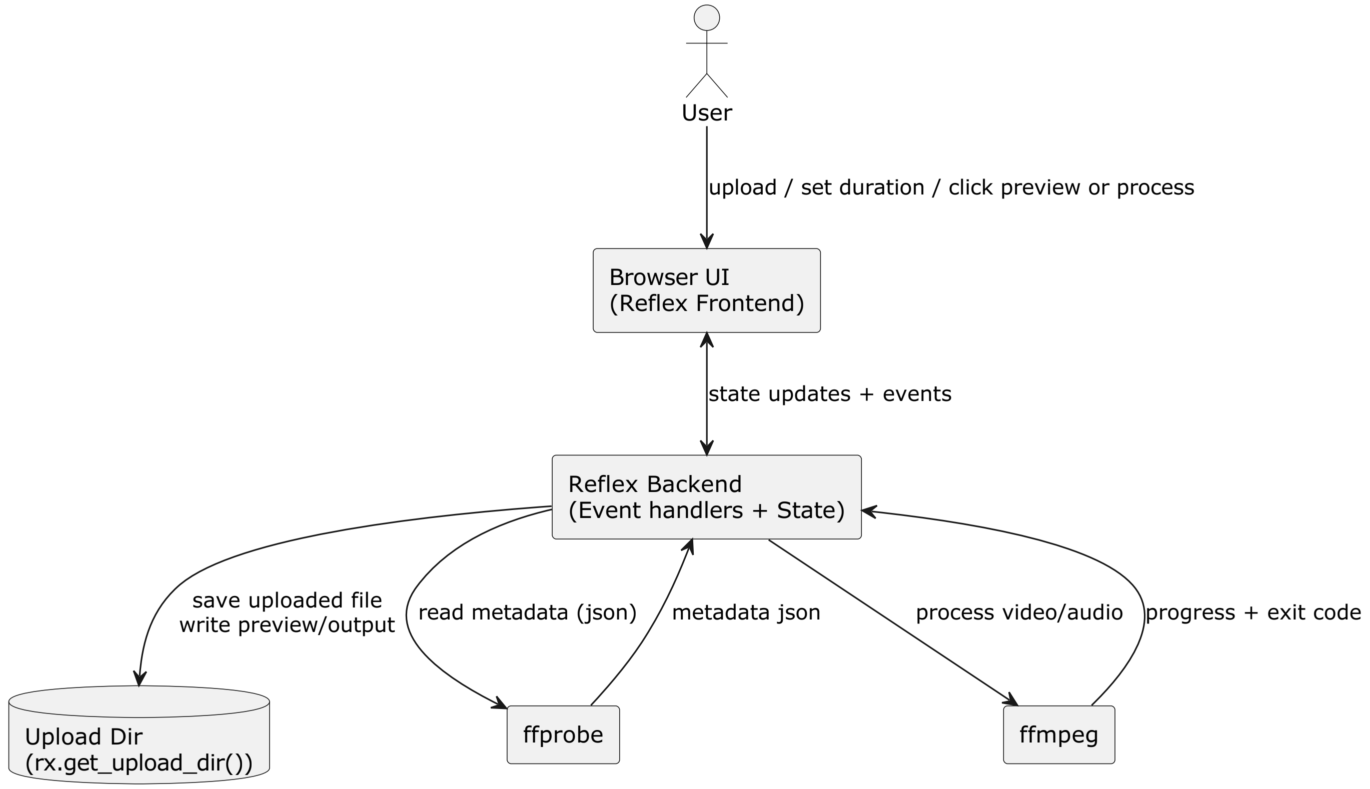
Key files:

- `rxconfig.py`: app name, API URL, plugins (Tailwind, Sitemap).
- `video_duration_adjuster/video_duration_adjuster.py`: main page layout + routing.
- `states/video_state.py`: core state machine, ffprobe metadata, ffmpeg processing.
- `components/*`: UI composition.

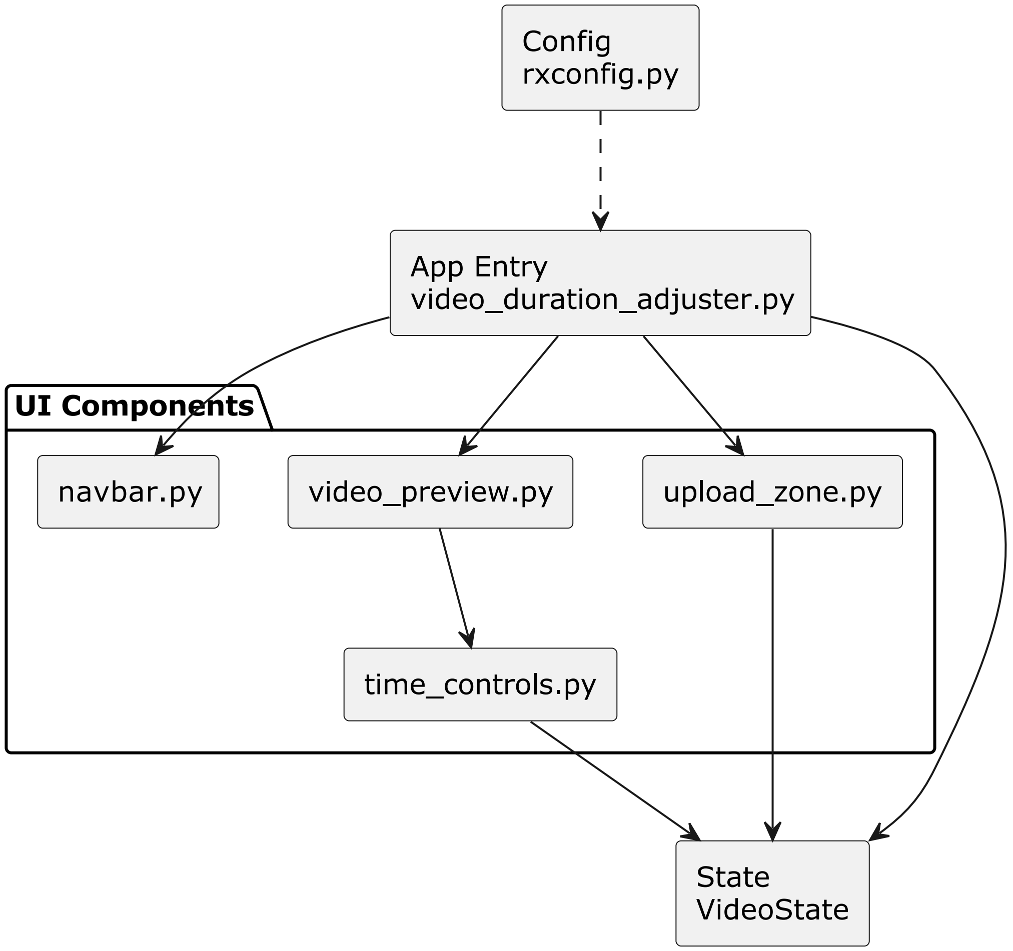
—

4. Architecture

4.1 High-Level System Context



4.2 Module Diagram



—

5. UX Flow (5 Steps)

The UI is explicitly step-based (also shown in the Documentation dialog):

1. **Original Video**: upload + original preview
2. **Video Information**: duration/resolution/size
3. **Set Target Duration**: time format or total seconds
4. **Preview Result**: generate 5-second preview
5. **Final Output**: process full video and download

—

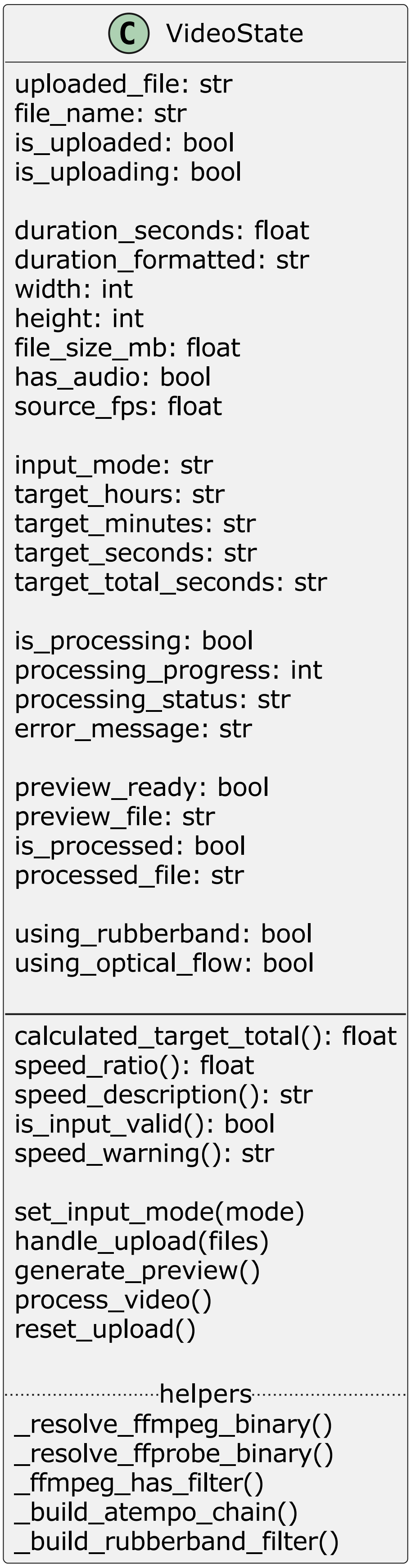
6. Core State Design (VideoState)

`VideoState` (in `states/video_state.py`) is the single source of truth for upload, metadata, processing, and UI status.

6.1 Key State Fields

- Upload:
- `uploaded_file`, `file_name`, `file_size_mb`
- `is_uploaded`, `is_uploading`
- Metadata:
- `duration_seconds`, `duration_formatted`
- `width`, `height`, `source_fps`
- `has_audio`
- Target duration input:
- `input_mode` = `"time"` or `"seconds"`
- `target_hours`, `target_minutes`, `target_seconds`
- `target_total_seconds`
- Processing:
- `is_processing`, `processing_progress`, `processing_status`
- `preview_ready`, `preview_file`
- `is_processed`, `processed_file`
- `error_message`
- Feature flags (runtime detection):
- `using_rubberband`, `using_optical_flow`

6.2 Class Diagram



7. Data and Processing Flow

7.1 Upload + Metadata Extraction (ffprobe)

When user uploads:

1. Server reads bytes from `rx.UploadFile`.
2. Saves into `rx.get_upload_dir()` using a random 8-char prefix and a sanitized filename (`isalnum` plus `._-`).
3. Runs:

- `ffprobe -print_format json -show_format -show_streams <file>`
  4. Parses:

- duration from `format.duration`

- video stream width/height
- FPS from `avg_frame_rate` or `r_frame_rate` via `Fraction`
- presence of audio stream

If ffprobe fails, it sets safe defaults and populates:

- `error_message = "Could not read video metadata. Please verify ffprobe/ffmpeg setup."`

7.2 Target Duration and Speed Ratio

- `calculated_target_total` is computed from:

- time mode: hours/minutes/seconds

- seconds mode: `target_total_seconds`
- `speed_ratio = duration_seconds / target_seconds` (rounded to 3 decimals)

- ratio > 1.0 means faster output (shorter)

- ratio < 1.0 means slower output (longer)
- Video timestamp scale factor used by ffmpeg:

- `video_speed_factor = 1.0 / speed_ratio`

- filter: `setpts=PTS*video_speed_factor`

7.3 Optical Flow Interpolation (minterpolate)

Enabled only when all are true:

- slowing down (`video_speed_factor > 1.0`)
- `ffmpeg` supports the `minterpolate` filter
- `source_fps > 0`

Filter becomes:

- `setpts=PTS*factor,minterpolate=fps=<target_fps>:mi_mode=mci:mc_mode=aobmc:vsbmc=1`

`target_fps` is clamped to `1.0..120.0`, with preference to keep at least the original fps.

7.4 Audio Tempo Handling

If the input has audio:

- If `ffmpeg` has `rubberband`:

- uses `rubberband=tempo=<tempo>:...:formant=preserved:...`

- sets `using_rubberband = True`
- else uses `atempo` chaining:

- builds multiple `atempo` filters because single `atempo` supports only about `0.5..2.0`

Audio tempo uses:

- `tempo = speed_ratio`

7.5 Filter Graph

- Video only:

- `[0:v]setpts=... [v]`

- Video + audio:

- `[0:v]... [v]; [0:a]<audio_filter> [a]`

Mapping:

- always `-map [v]`
- plus `-map [a]` if audio exists

—

8. FFmpeg Execution and Progress Tracking

8.1 Command Execution

Processing spawns:

- `ffmpeg -progress pipe:1 -nostats -i <input> -filter_complex <graph> -map [v] (-map [a]) -y <output>`

Preview mode adds:

- `-t 5`

8.2 Progress Updates

The app reads ffmpeg output lines and parses:

- `out_time_ms=<int>`
- converts to seconds and computes percent based on:

- preview: expected 5 seconds

- full: expected `calculated_target_total` (min 0.001)
- clamps to `1..99` during processing, sets `100` on success.

On `progress=end`, it sets progress to at least 99.

8.3 Output Artifacts

All stored in Reflex upload directory:

- preview: `preview_<uploaded_file>`
- full: `processed_<uploaded_file>`

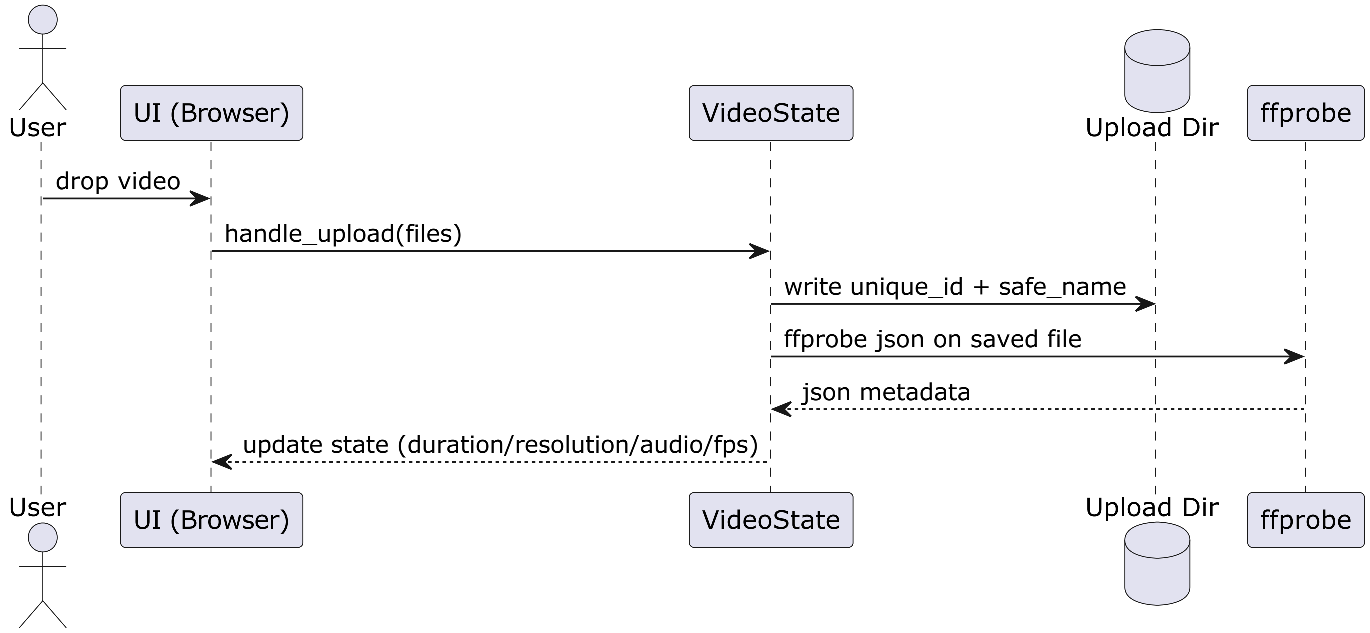
Downloads are provided by:

- `rx.get_upload_url(filename)` and `<a download=...>`.

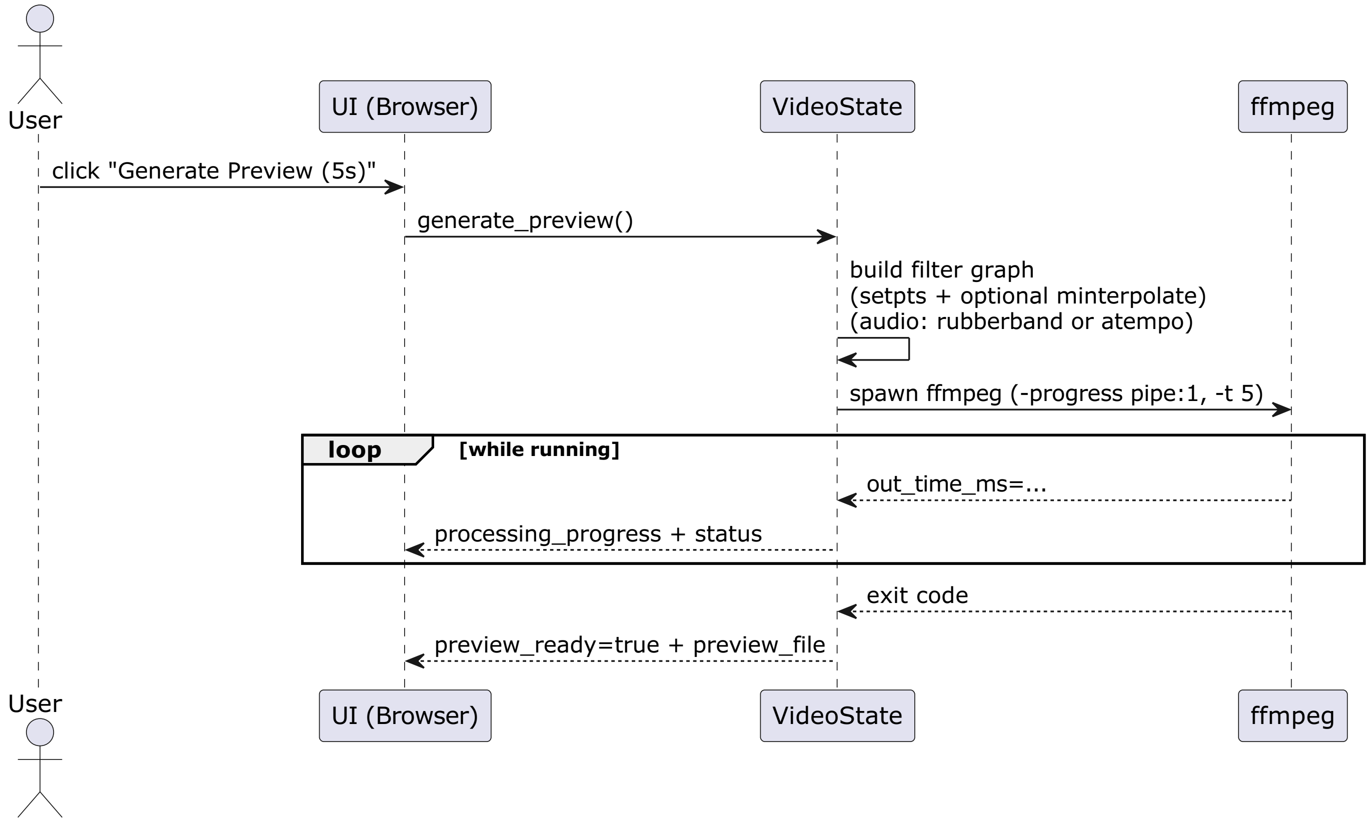
—

9. Sequence Diagrams

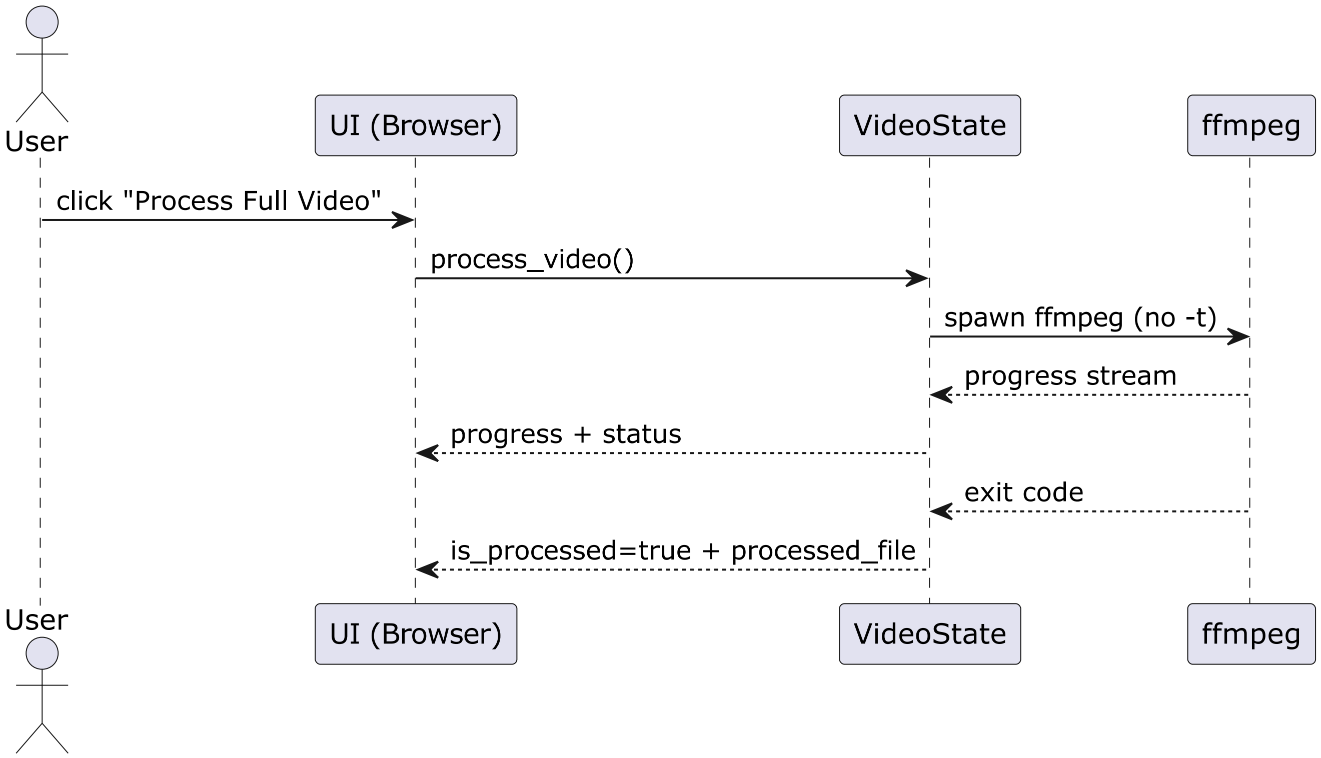
9.1 Upload and Metadata



9.2 Preview Generation


User
UI (Browser)
VideoState
ffmpeg
click "Generate Preview (5s)"
generate_preview()
build filter graph
(setpts + optional minterpolate)
(audio: rubberband or atempo)
spawn ffmpeg (-progress pipe:1, -t 5)
loop
[while running]
out_time_ms=...
processing_progress + status
exit code
preview_ready=true + preview_file
User
UI (Browser)
VideoState
ffmpeg

9.3 Full Processing



—

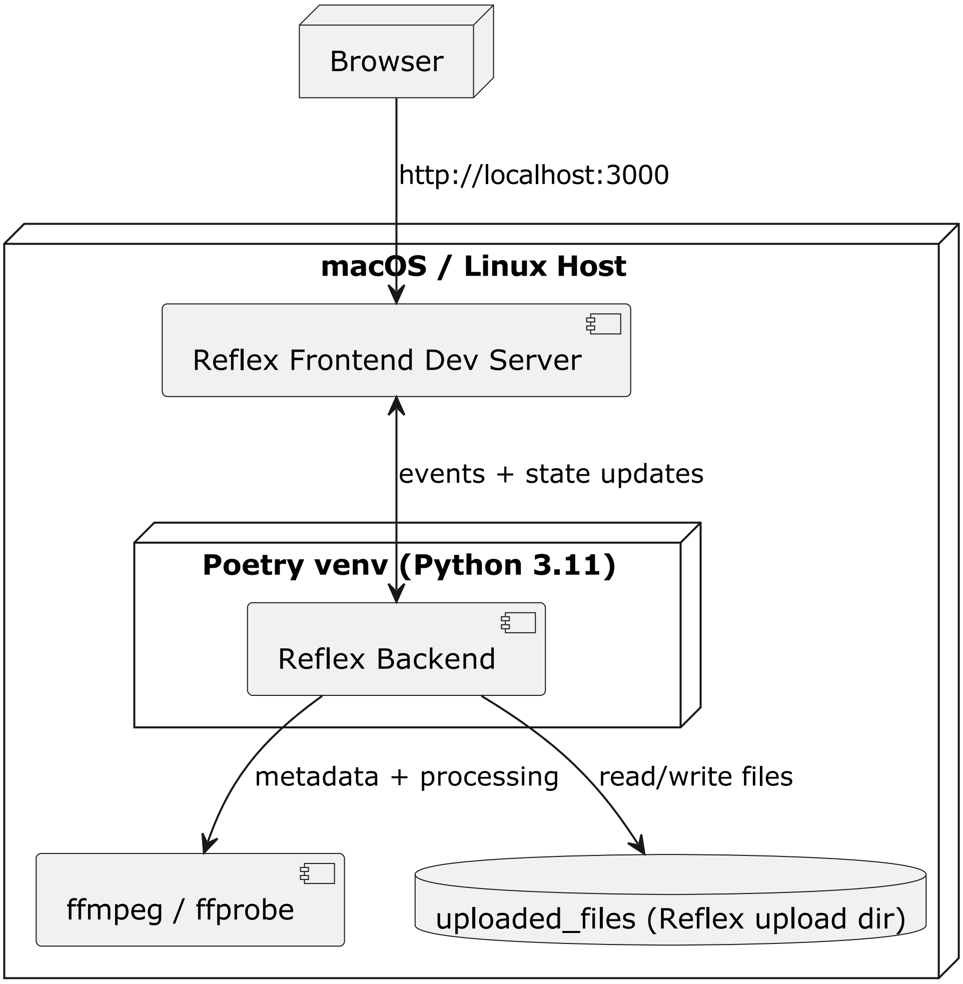
10. Deployment

10.1 Local Development

Recommended (per README):

- `brew install python@3.11 ffmpeg-full poetry`
- `poetry env use python3.11`
- `poetry install`
- `poetry run ./reflex_rerun.sh`
- App at `http://localhost:3000`

10.2 Deployment Diagram



–

11. Error Handling and Edge Cases

11.1 Upload Failures

- File list empty: return early.
- Write error: shows toast ”Failed to process video file.”

11.2 Metadata Failures

- ffprobe non-zero exit or empty output:

- sets safe defaults

- sets `error_message` explaining setup issue

11.3 Processing Failures

- Missing file: `FileNotFoundError("Source file not found")`

- Invalid ratio: `ValueError("Invalid speed ratio")`
- ffmpeg non-zero exit:

- logs last ~80 lines of output

- user sees `error_message = "Error: ..."` and status ”Failed”

11.4 Extreme Ratio Warnings

- ratio > 4.0: warns about visual artifacts
- ratio < 0.25: warns about heavy interpolation

–

12. Notable Design Decisions

1. **Prefer `ffmpeg-full`** on macOS to enable `rubberband` filter.
2. **Feature detection at runtime**:

- checks if `rubberband` and `minterpolate` exist in current ffmpeg build. 3. **Optical-flow interpolation only when slowing down** and filter is supported. 4. **Preview-first workflow** to reduce user cost before full processing.

–

13. Testing

13.1 Present in Repo

- `run_test_suite.sh` exists as a harness (Playwright-based per dependencies).
- `pytest` and `playwright` are in dev dependencies.

13.2 Recommended Tests

- Unit tests:

- `_build_atempo_chain(tempo)` correctness

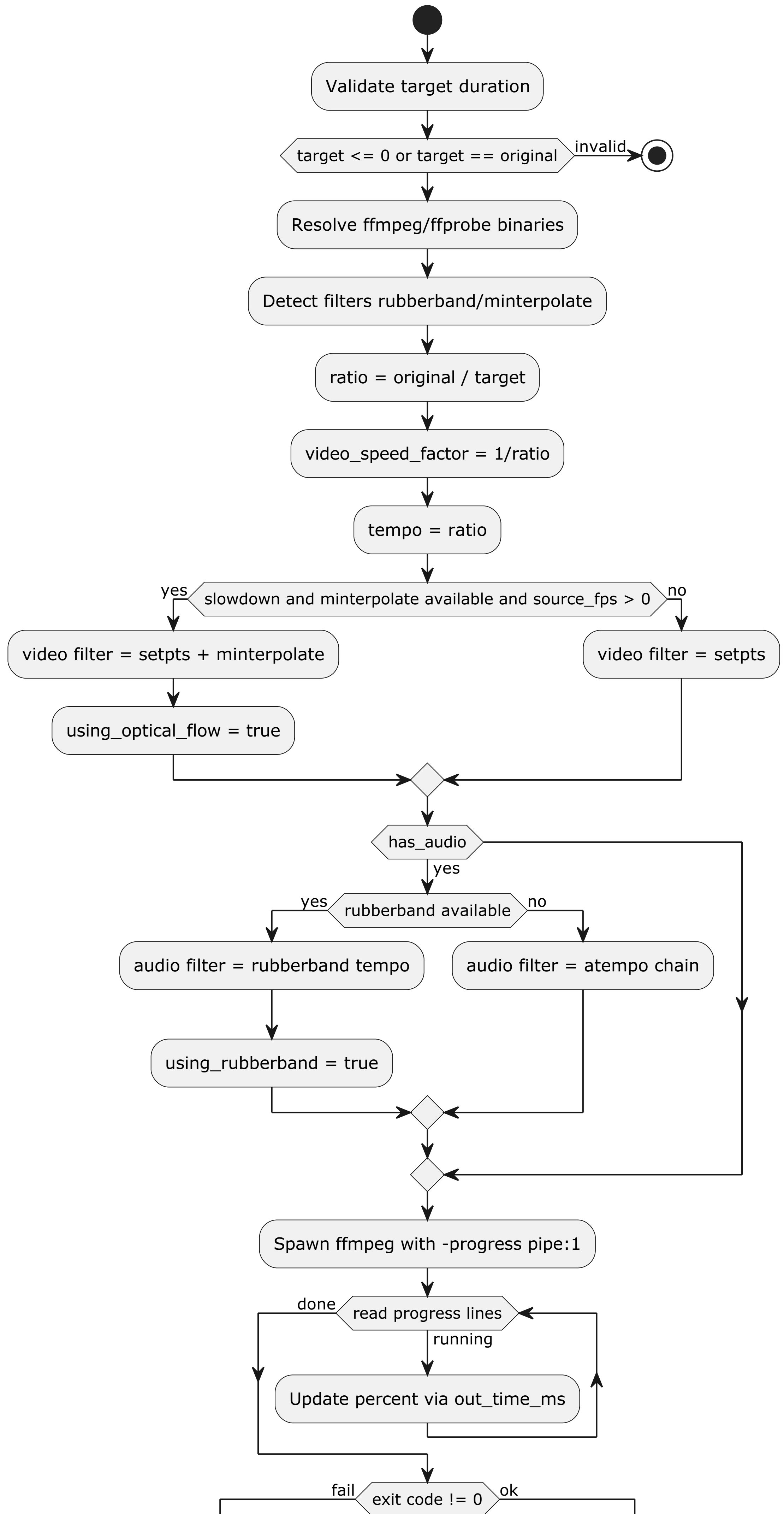
- `_parse_fps("30000/1001")` behavior
- `speed_ratio` and validation rules
- Integration tests:

- ffprobe metadata parsing on known sample files

- ffmpeg processing for:
    - video-only
    - video+audio (rubberband and atempo fallback)
    - slow-down path with minterpolate
    - E2E tests (Playwright):
- upload -> metadata visible
- set target -> preview -> download exists
- full process -> download exists

—

14. Appendix: Processing Activity Diagram


Validate target duration
target <= 0 or target == original
invalid
Resolve ffmpeg/ffprobe binaries
Detect filters rubberband/minterpolate
ratio = original / target
video_speed_factor = 1/ratio
tempo = ratio
yes
slowdown and minterpolate available and source_fps > 0
no
video filter = setpts + minterpolate
video filter = setpts
using_optical_flow = true
has_audio
yes
yes
rubberband available
no
audio filter = rubberband tempo
audio filter = atempo chain
using_rubberband = true
Spawn ffmpeg with -progress pipe:1
done
read progress lines
running
Update percent via out_time_ms
fail
exit code != 0
ok

—

15. Reference Note: RIFE Exploration (Non-Default)

The README documents an experimental path using **RIFE v4.22** (Metal/CoreML acceleration on Apple Silicon) explored in PR #1, but it was not adopted as default due to output quality comparison against the CPU-based optical-flow interpolation approach.

—