Video Segment Splitter (ClipShift) — Software Design Document (SDD)

1. Document Control

- Project: Video Segment Splitter (UI branding: **ClipShift**)
- Repository layout: `https://github.com/milochen0418/video_segment_splitter`
- Primary runtime: Python 3.11 + Reflex
- Purpose: Upload a video, choose a segment count, split into N clips, and download clips individually or as a ZIP bundle.

2. Executive Summary

This system is a browser-based video splitting application built with **Reflex** (Python full-stack web framework). Users upload a video file; the backend extracts metadata (duration, resolution, size), then performs splitting by invoking **ffmpeg** as an asynchronous subprocess for each segment. Results are written to the server upload directory and served back to the browser via download URLs (including a “Zip All Clips” bundle).

Key architectural choices:

- **Reflex State** is the core application controller (upload, metadata extraction, splitting, zipping).
- Video splitting uses **ffmpeg subprocesses** rather than in-process Python encoding, reducing GIL contention and keeping the event loop responsive.
- A lightweight “System Monitor” UI (powered by `psutil`) can be toggled during processing to observe CPU/memory/load.

3. Goals and Non-Goals

3.1 Goals

- Simple UX: upload → choose segments (1–20) → split → download.
- Accurate metadata display (duration, resolution, file size).
- Server-side splitting with progress updates.
- Provide individual download links per clip.
- Provide “ZIP all clips” download.

3.2 Non-Goals (current scope)

- No timeline editor / manual cut points.
- No authentication, per-user storage isolation, or multi-tenant quota management.
- No persistent database for jobs/history.
- No cloud object storage integration (S3/GCS/etc).
- No automatic cleanup/retention policy for uploaded/generated files.

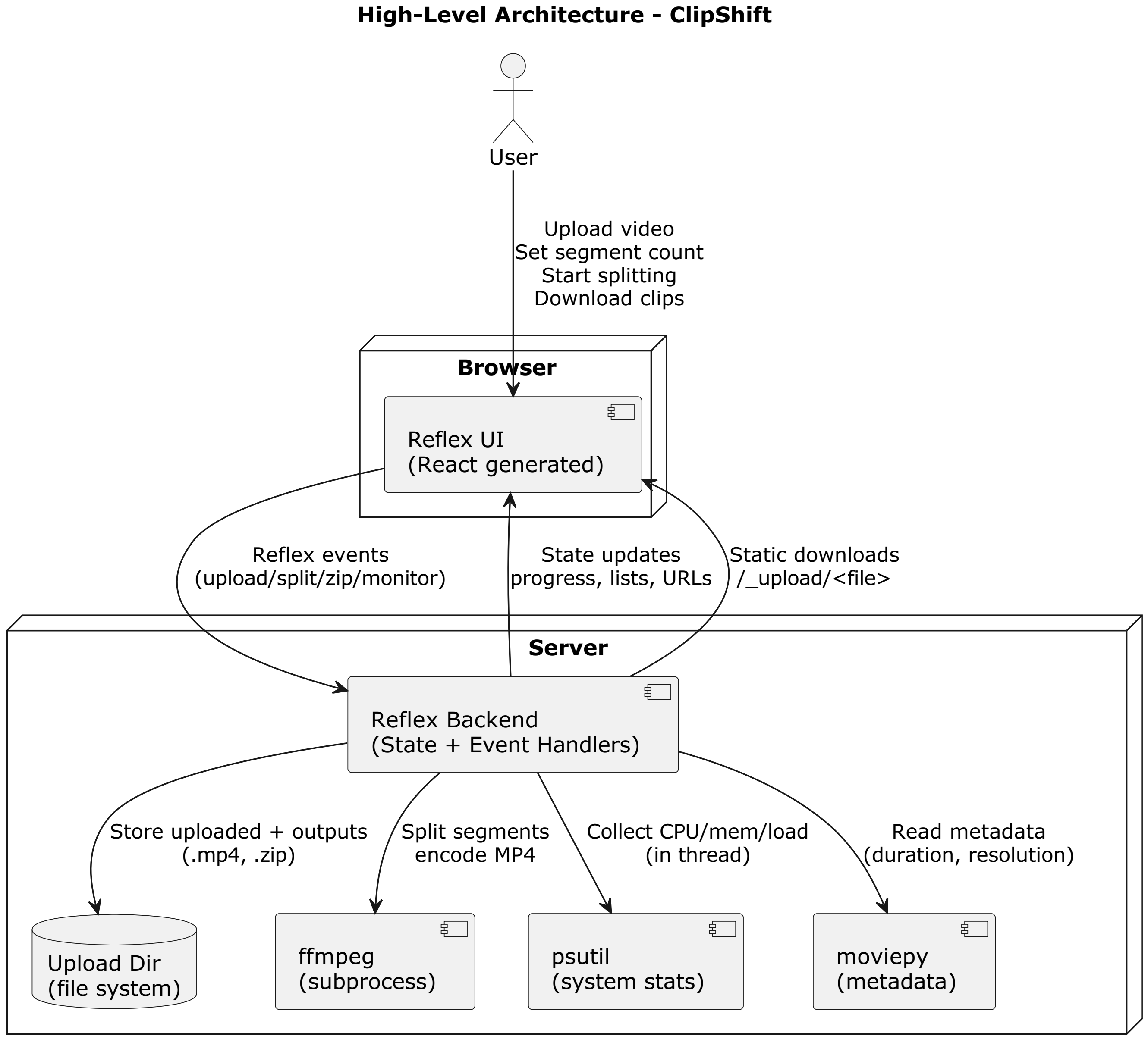
4. System Overview

4.1 High-Level Architecture

- **Frontend (Browser)**: Reflex-generated React UI components.
- **Backend (Reflex server)**: Hosts state/event handlers and serves uploaded/generated files.
- **Video Tooling**:

- Metadata extraction: `moviepy.VideoFileClip`

- Splitting/encoding: `ffmpeg` invoked via `asyncio.create_subprocess_exec`
- **Monitoring**: `psutil` sampled in a background thread via `asyncio.to_thread`.

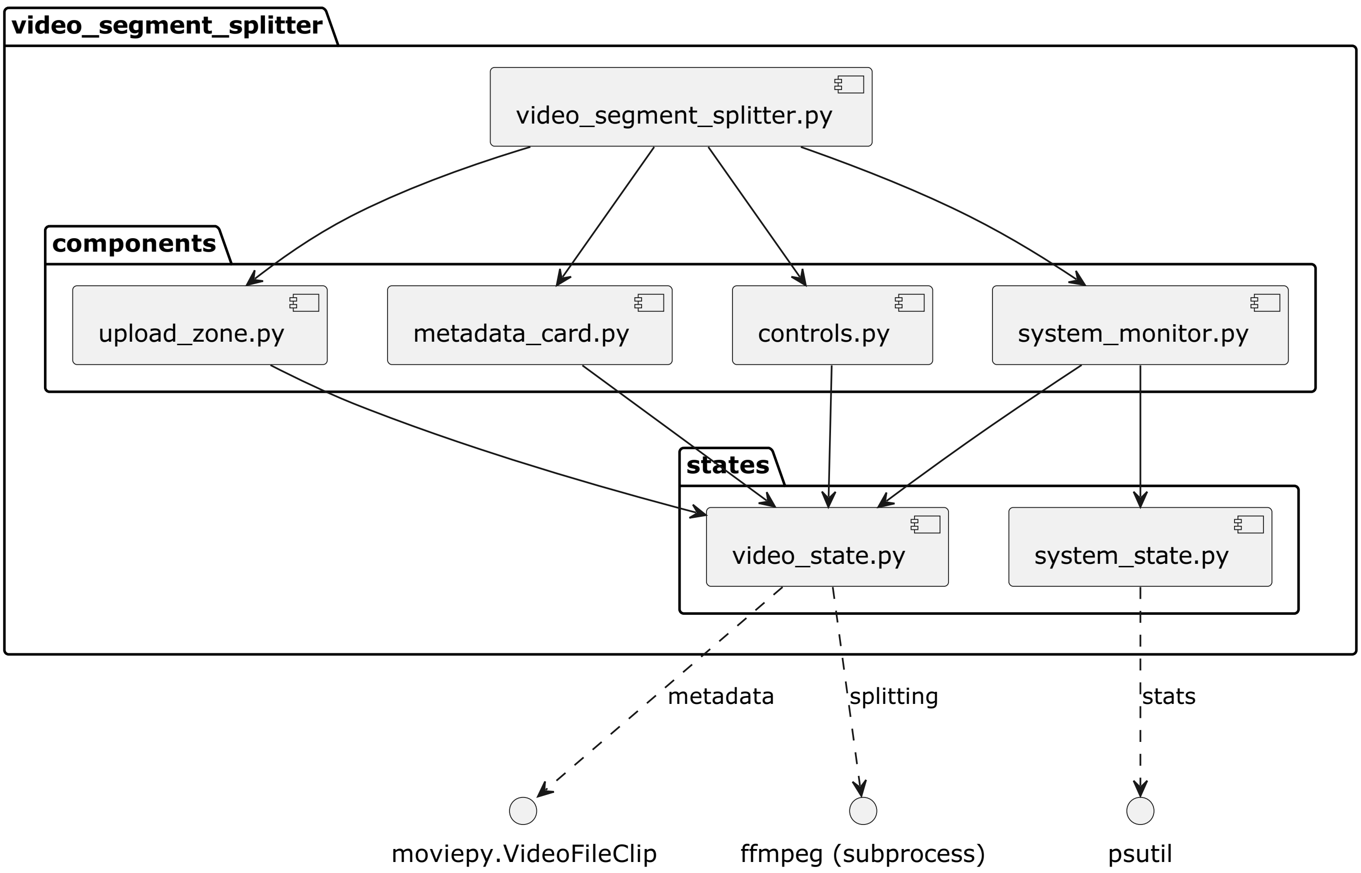


5. Repository Structure

Top-level important files:

- `video_segment_splitter/video_segment_splitter.py`: main page composition and app initialization.
- `video_segment_splitter/states/video_state.py`: upload, metadata, splitting, zipping.
- `video_segment_splitter/states/system_state.py`: system monitor stats collection & refresh loop.
- `video_segment_splitter/components/*`: UI components.
- `rxconfig.py`: Reflex app config (api_url, plugins).
- `pyproject.toml`: dependency and Python version constraints.
- `reflex_rerun.sh`, `proj_reinstall.sh`: dev tooling scripts.

Package / Module Diagram



6. Runtime & Dependencies

6.1 Runtime

- Python: `~3.11` (strict, enforced by Poetry)
- Reflex: `0.8.24.post1`
- Local dev ports:
- Frontend: `http://localhost:3000`
- Backend/API: `http://localhost:8000` (also used to build download URLs)

6.2 Core Dependencies (from `pyproject.toml`)

- `reflex`: UI + backend event/state framework
- `moviepy`: video metadata extraction (and historically could do cutting)
- `psutil`: system load/CPU/memory monitoring
- `pydantic`: typed models (`VideoMetadata`, `VideoSegment`)
- `asyncio`: async subprocess and background tasks
- `ffmpeg`: required binary available on PATH (fallback: `imageio-ffmpeg` if installed)

6.3 Configuration (`rxconfig.py`)

- `app_name="video_segment_splitter"`
- `api_url="http://localhost:8000"`
- Tailwind v4 plugin + sitemap plugin enabled

7. Core Domain Model

Defined in `video_state.py`:

7.1 Data Classes

- `VideoMetadata`

- `filename`

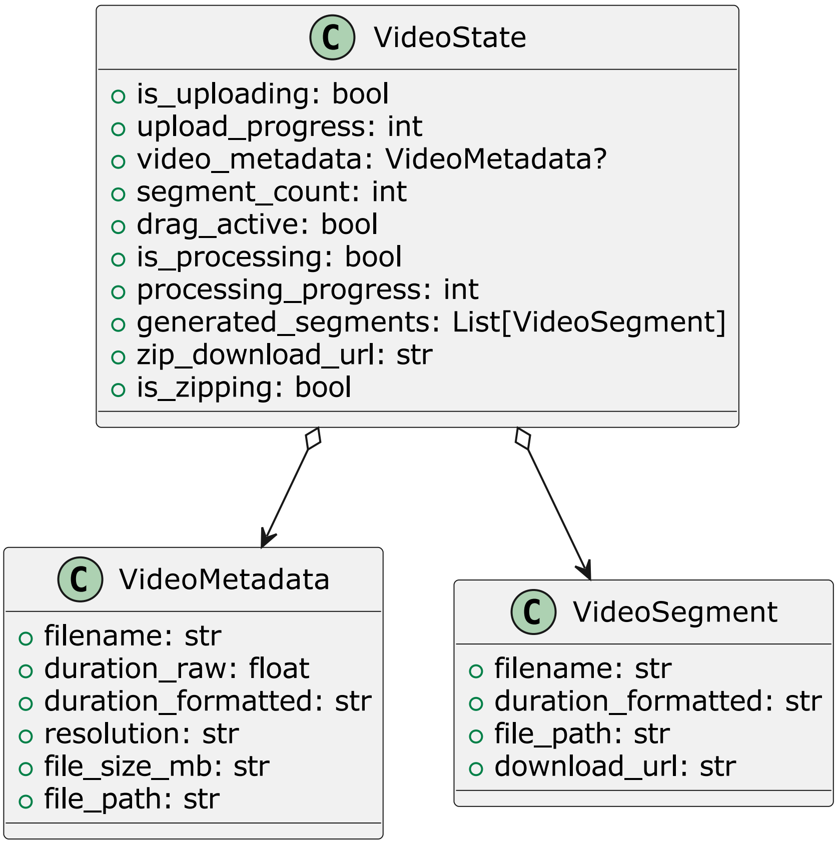
- `duration_raw` (seconds)
- `duration_formatted` (HH:MM:SS)
- `resolution` (WxH string)
- `file_size_mb`
- `file_path` (server-side path)

- `VideoSegment`

- `filename`

- `duration_formatted`
- `file_path`
- `download_url`

Domain Model (Pydantic Models)



8. Main User Journeys and Flows

8.1 Upload & Metadata Extraction Flow

Trigger

User drops a file into `rx.upload.root` (component `upload_zone.py`), which calls:

- `VideoState.handle_upload(rx.upload_files(upload_id="video_upload"))`

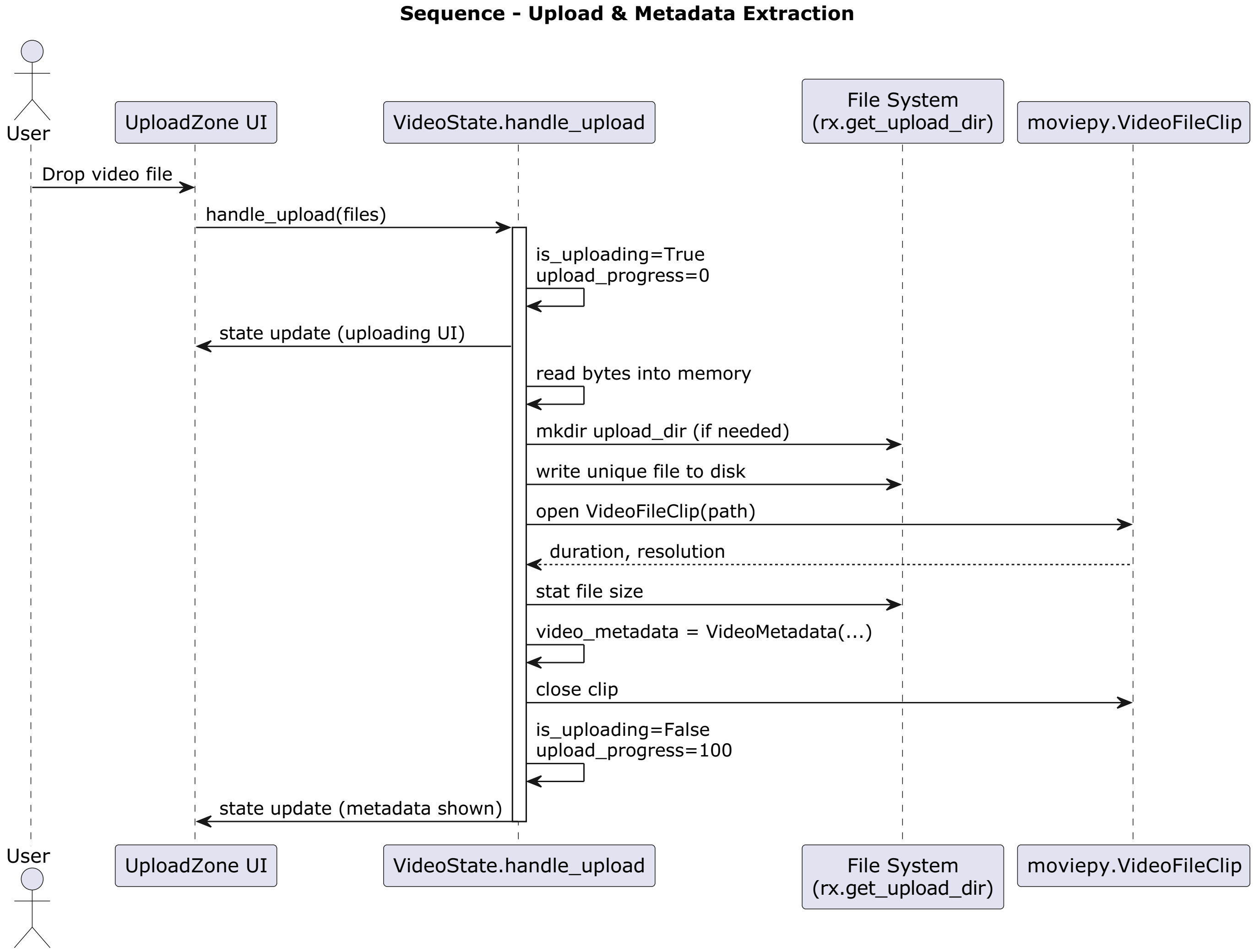
Steps (as implemented)

1. Set `is_uploading=True`, reset `upload_progress=0`.
2. For each file:

- Read entire file into memory (`upload_data = await file.read()`).
- Ensure upload directory exists (`rx.get_upload_dir()`).
- Create `unique_name = <random10>_<originalName>`.
- Write bytes to disk.
  3. Extract metadata using `VideoFileClip(file_path)`:

- Duration (`clip.duration`)

- Resolution (`clip.size`)
- File size (`os.path.getsize`) 4. Save `VideoMetadata` into state. 5. Set `is_uploading=False`, `upload_progress=100`. 6. On exceptions, log and show a toast error.



Notes / Implications

- Upload reads the full file into memory. Large uploads can spike memory usage.
- No explicit validation beyond file extension accept list and the ability of MoviePy to parse.

8.2 Split Video Flow (N segments)

Trigger

User clicks **Start Splitting Video** (component `controls.py`):

- `on_click=VideoState.split_video`

`split_video` is marked as `@rx.event(background=True)`.

Steps (as implemented)

1. Acquire state lock and validate:

- Must have `video_metadata`
- Must not already be processing
  2. Initialize:

- `is_processing=True`

- `processing_progress=0`
- clear `generated_segments`
  3. Compute:

- `segment_duration = total_duration / segment_count`
  4. For each segment `i`:

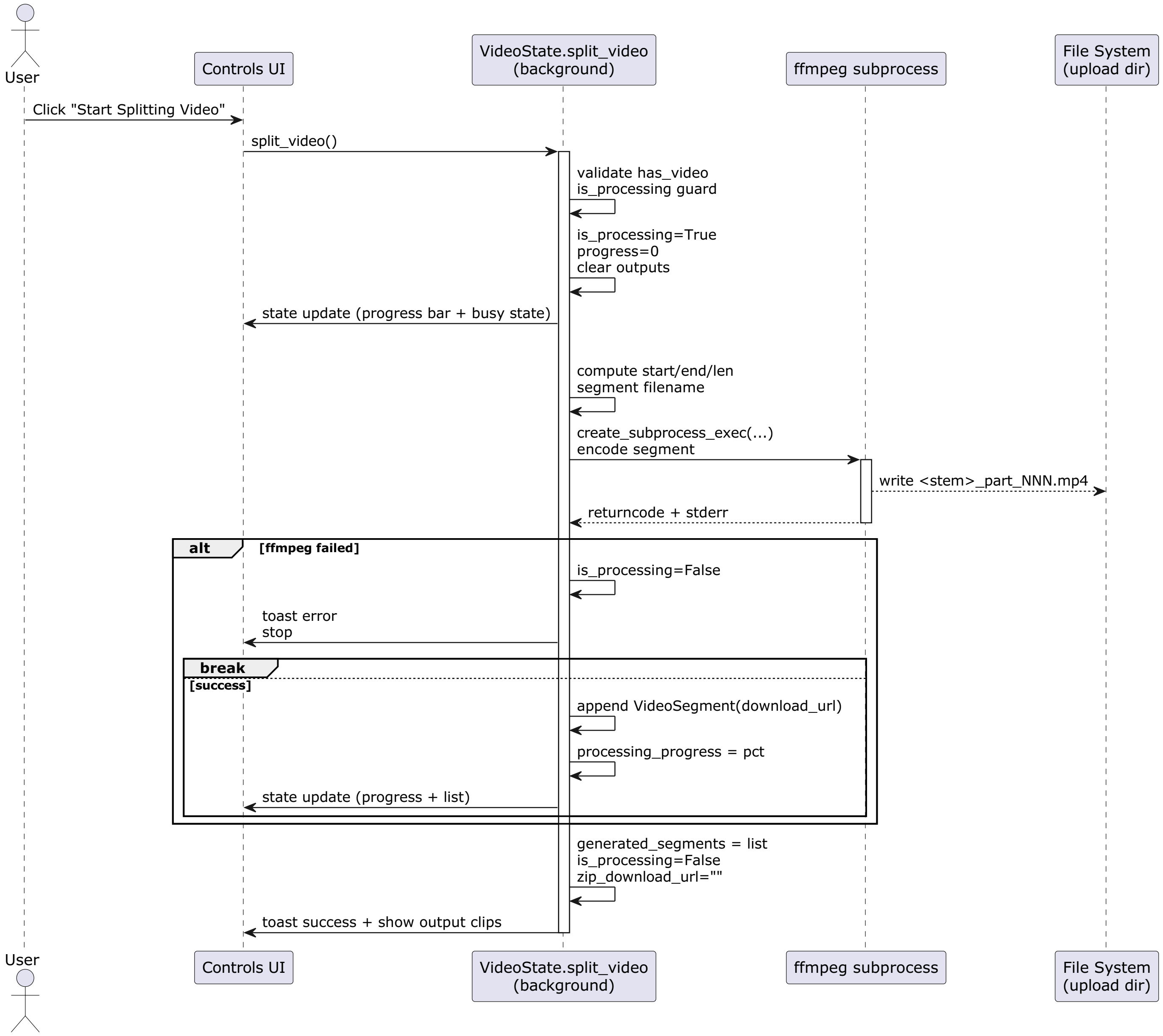
- Calculate `start_time`, `end_time`, `seg_len`

- Output filename: `<original_stem>_part_<NNN>.mp4`
- Invoke `ffmpeg` as async subprocess:
  - `-ss <start>`
  - `-i <input>`
  - `-t <length>`
  - `-c:v libx264`, `-c:a aac`
  - `-threads <max_threads>` where `max_threads = max(1, cpu_count//2)`
  - `-loglevel error`
  - If ffmpeg fails: toast error, stop.
  - Build download URL as `{api_url}/_upload/{segment_filename}`
  - Append `VideoSegment` and update `processing_progress` 5. After loop:

- Save generated list

- `is_processing=False`
- Clear `zip_download_url`
- Toast success

Sequence - Split Video into Segments (ffmpeg)



Notes / Implications

- Segments are created sequentially (one ffmpeg process at a time).
- Thread limiting (`cpu_count//2`) is a deliberate UX choice to preserve server responsiveness.
- Output format fixed to MP4 with H.264 + AAC.

8.3 Zip All Clips Flow

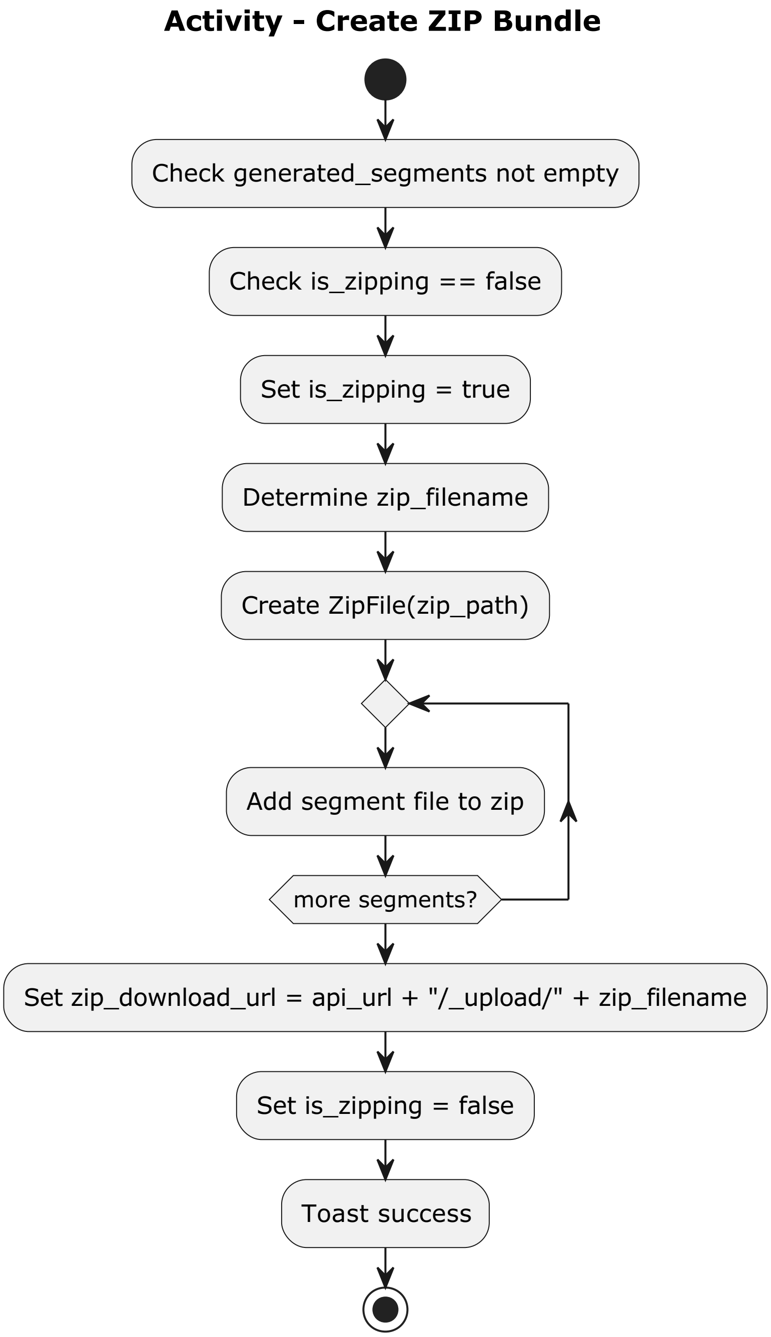
Trigger

User clicks **Zip All Clips** (main page, in output header):

- `on_click=VideoState.create_zip_download`

Steps

1. Validate `generated_segments` not empty and not already zipping.
2. Create `<original_stem>_all_parts.zip` in upload dir.
3. Add each segment file into ZIP with `arcname=seg.filename`.
4. Set `zip_download_url = {api_url}/_upload/{zip_filename}`.
5. Toast success.


Activity - Create ZIP Bundle
Check generated_segments not empty
Check is_zipping == false
Set is_zipping = true
Determine zip_filename
Create ZipFile(zip_path)
Add segment file to zip
more segments?
Set zip_download_url = api_url + "/_upload/" + zip_filename
Set is_zipping = false
Toast success

8.4 System Monitor Flow

Trigger

During `is_processing=True`, a floating **System Busy** button appears (`system_monitor.py`). Clicking toggles a panel.

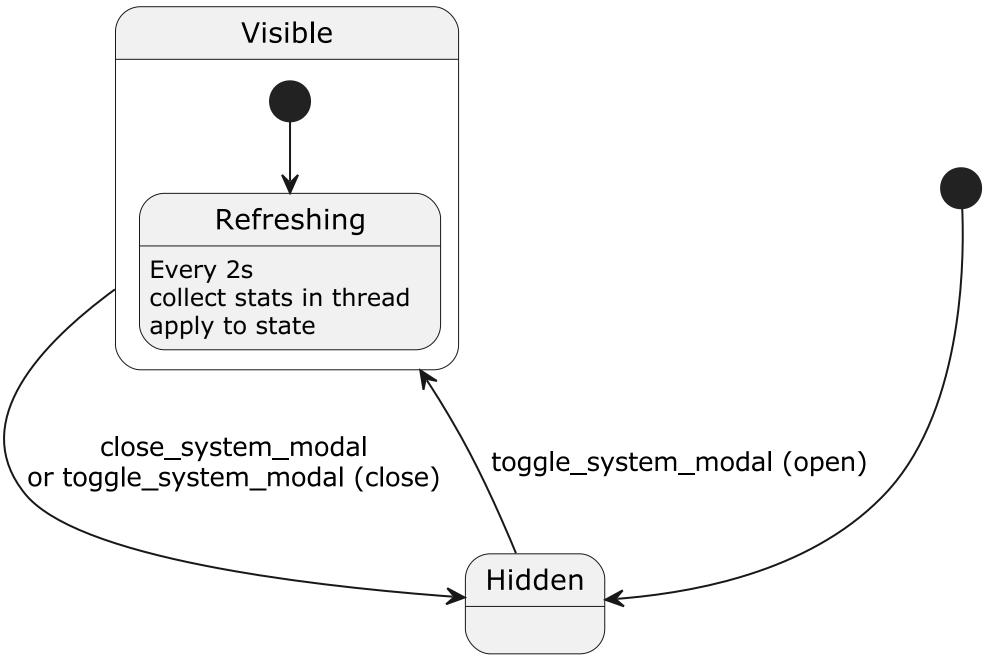
Implementation (`SystemState`)

- `toggle_system_modal()` toggles UI state; if opening, it returns `SystemState.auto_refresh` event.
- `auto_refresh` runs in background:

- Every 2 seconds: `asyncio.to_thread(_collect_system_stats)`

- Applies stats to state variables
- Stops when modal closes

State Machine - System Monitor Panel



9. UI Composition

Main page is defined in `video_segment_splitter.py`:

- Header with branding and GitHub link (to GitHub).
- Hero section.
- Core controls:

- `upload_zone()` (drop area)

- `metadata_card()` (only when `has_video`)
- `controls()` (segment count + start button)
- Output clips section:

- Appears when `has_video`

- Shows either placeholder or list of segments
- ZIP button appears after segments exist
- Footer and system monitor floating button.

10. Backend / Serving Downloads

The code constructs download URLs as:

- `{api_url}/_upload/{filename}`

This assumes the Reflex backend serves the upload directory via a route like `/_upload/` (typical Reflex behavior for uploaded assets).

Design implications:

- Generated files become directly downloadable without additional API handlers.
- Access control is not enforced; any user who knows the URL can fetch the file (in the current design).

11. Concurrency, State, and Responsiveness

11.1 Reflex State Concurrency

- Long-running operations are run with `@rx.event(background=True)` and use `async with self:` blocks to safely mutate state and yield UI updates.
- `split_video` uses sequential steps with periodic state updates (`processing_progress`).

11.2 Event Loop Responsiveness

- Video splitting is delegated to an OS process (ffmpeg), avoiding Python CPU-bound work.
- System monitoring uses `asyncio.to_thread` to keep `psutil` sampling off the event loop.

12. Error Handling Strategy

- Upload metadata extraction is wrapped in try/except:

- Logs exception

- Shows toast: “Failed to process video metadata”
- Splitting:

- If ffmpeg returns non-zero:

    - Log stderr
    - Stop processing
    - Toast error: “ffmpeg failed on segment X”
    - General exceptions:

    - Log and toast error with message

    - Zipping:

- Any exception leads to toast: “Failed to create ZIP archive”

- `is_zipping` is reset in finally-like behavior

13. Security and Risk Considerations

13.1 File Handling Risks

- Files are stored on server disk; there is no cleanup on:

- clearing UI selection

- finishing a job
- Filename stem is used to create output names; unusual/unicode names may produce awkward filenames.

13.2 Resource Exhaustion

- Upload reads full file bytes into memory; large videos may cause memory spikes.
- Multiple concurrent users can spawn multiple ffmpeg subprocesses (even if each job is sequential internally), saturating CPU/disk.

13.3 Access Control

- No authentication and no per-user isolation.
- Direct download URLs can be shared.

13.4 Mitigations (recommended)

- Stream uploads to disk instead of `await file.read()` (chunked write).

- Add maximum upload size and duration limits.
- Introduce cleanup policy (TTL, job-based folders, or explicit “delete outputs”).
- Add auth and isolate outputs by session/user folder if needed.
- Consider randomizing output clip filenames (not only prefix for upload).

14. Performance Considerations

- Encoding settings are fixed (`libx264`, `aac`) which can be CPU intensive.
- `-threads` is capped to half cores; good for UX but slower throughput.
- Current pipeline re-encodes segments; for some inputs, a faster “stream copy” approach might be possible (`-c copy`) but would need careful handling of keyframes and accuracy.

15. Deployment Architecture

15.1 Local Development

- Install system packages:

- Python 3.11

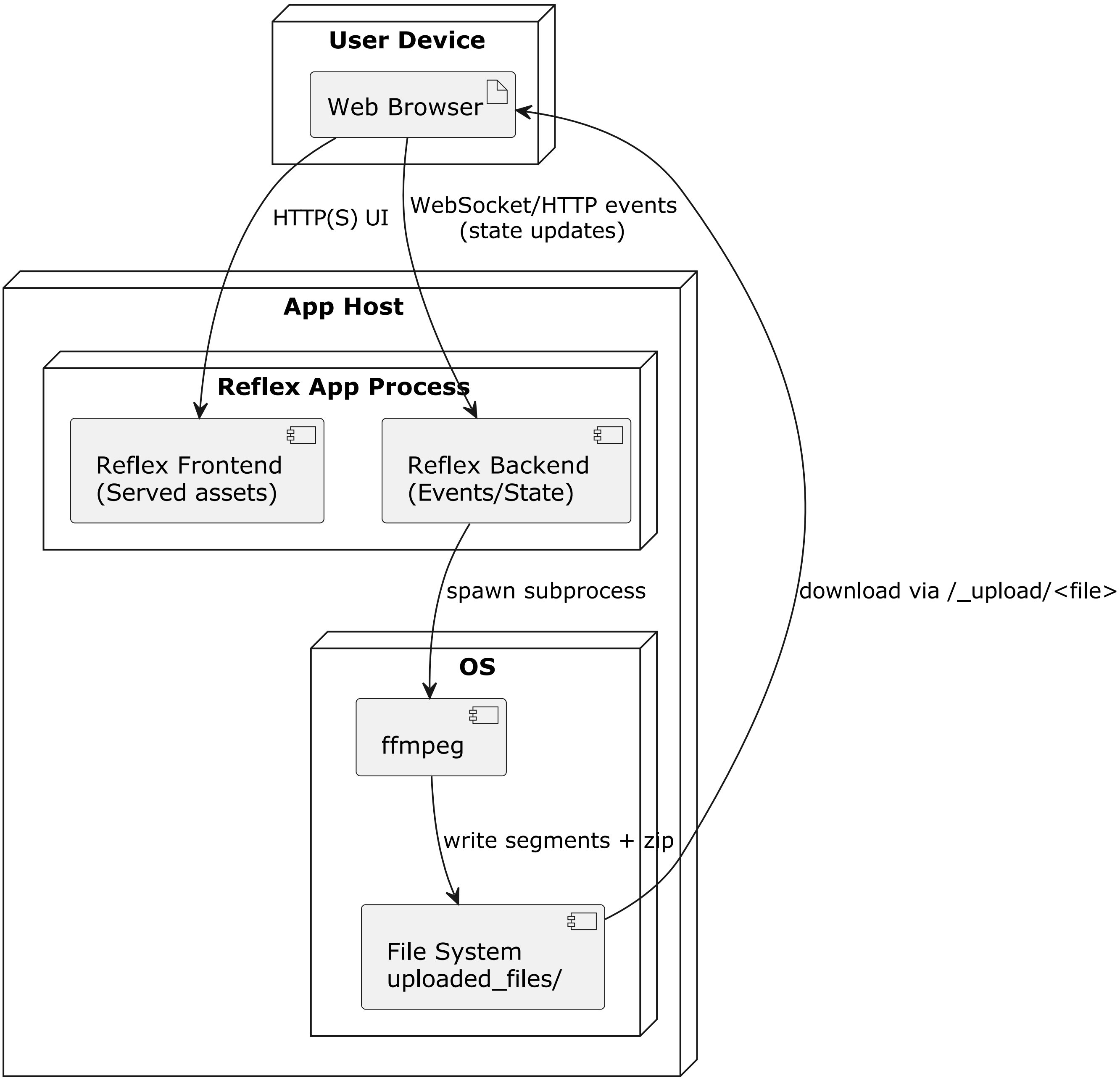
- ffmpeg
- Poetry
- `poetry install`
- Start with:

- `poetry run ./reflex_rerun.sh`

15.2 Production (Typical)

- Build environment includes ffmpeg binary.
- Run Reflex backend/server behind a reverse proxy.
- Persist upload directory on durable storage if downloads must survive restarts.

Deployment Diagram (Typical)



16. Extensibility Roadmap (Practical Next Steps)

1. **Custom split points**

- UI for timeline markers
- Backend accepts list of (start, end) ranges

2. **Job queue**

- Server-wide queue to limit concurrent ffmpeg executions

3. **Persistence**

- Store job metadata in DB (SQLite/Postgres)

- Rehydrate job history on reload

4. **Storage backends**

- Upload to object storage + signed URLs

5. **Observability**

- Structured logs for ffmpeg command + duration
- Basic metrics (job count, average time, failures)

17. Appendix A — Key Implementation Notes (from code)

- Metadata extraction uses `moviepy.VideoFileClip(file_path)` then `.duration` and `.size`.
- Splitting uses ffmpeg with `-ss` (seek), `-t` (duration), and re-encode parameters.
- Segment count constrained to `[1..20]`.
- Download URL assumes server exposes `/_upload/` mapped to upload dir.
- System monitor is only shown while `VideoState.is_processing` is true.