DICOM Viewer — Software Design Document (SDD)

Project: `dicom_viewer`

Version: 0.1.0

Tech Stack: Python 3.11, Reflex 0.8.24.post1, pydicom, numpy, Pillow, Tailwind (via Reflex plugin)

Primary Goal: Browse a local folder of DICOM files, display images, and interactively adjust visualization (window/level presets, zoom/pan) with a metadata side panel.

—

1. Executive Summary

This project is a lightweight **local DICOM image viewer** built with **Reflex** (Python-first web app framework). The application runs a local web server (frontend at `:3000`, backend at `:8000`) and provides:

- A directory selection workflow using a server-side directory browser UI
- Background scanning of a selected folder for valid DICOM files
- Image navigation (list + slider + prev/next)
- Image processing: DICOM pixel decoding + (optional) HU rescale + window/level + PNG render + Base64 delivery
- Viewer controls: presets, manual windowing, zoom/pan, reset
- Metadata side panel with a simple password-protected “unlock” for sensitive fields

—

2. Scope

2.1 In Scope

- Local directory browsing and scanning
- DICOM validation (best-effort) and sorting
- Single-slice image display (first slice for multi-frame)
- Window leveling (center/width)

- Zoom and pan via CSS transform
- Metadata extraction for common tags
- Tutorial dialog rendering from same-origin static HTML

2.2 Out of Scope (Current)

- PACS / DICOMweb networking (QIDO/WADO/STOW)
- Multi-series hierarchy browsing (studies/series separation)
- 3D volume rendering, MPR, cine playback for multi-frame
- Annotation tools, measurements, ROI
- User accounts, multi-user collaboration, persistent database

—

3. User Stories

1. **Select a DICOM folder** - As a user, I want to pick a folder containing DICOM files so I can view them.
2. **Scan results** - As a user, I want to see a list of detected DICOM images so I can choose where to start.
3. **View and navigate** - As a user, I want to browse images quickly using slider and next/prev.
4. **Adjust visualization** - As a user, I want presets (lung/bone/soft tissue…) and manual WW/WL controls.
5. **Inspect metadata** - As a user, I want to see study/series/image details and optionally unlock protected fields.
6. **Learn windowing** - As a user, I want an in-app tutorial for presets via embedded HTML pages.

—

4. System Overview

4.1 High-Level Architecture

Reflex apps are conceptually split into:
- **UI (React) generated from Python component definitions**
- **State & events** handled server-side (Python), pushed to the frontend

This project follows the typical Reflex model:
- `dicom_viewer/states/dicom_state.py` = source of truth for data + actions

- `dicom_viewer/dicom_viewer.py` + `dicom_viewer/components/*.py` = UI pages and components bound to state/events

—

5. Repository Structure

```
dicom_viewer-main/
  README.md
  rxconfig.py
  pyproject.toml
  assets/
    placeholder.svg
    tutorials/
      windowing_en.html
      windowing_es.html
      windowing_zh_cn.html
      windowing_zh_tw.html
  dicom_viewer/
    dicom_viewer.py       # app + routes + landing page UI
    components/
      loading_spinner.py  # scanning indicator
      viewer.py           # /viewer layout + panels + controls
    states/
      dicom_state.py      # DicomViewerState: directory scan + image processing
  docs/images/            # product screenshots referenced by README
```

—

6. Key Design Decisions

6.1 Reflex State as the Single Source of Truth

All UI derives from `DicomViewerState` fields. User actions trigger Reflex events that mutate state and optionally redirect routes.

6.2 Server-Side Image Rendering

DICOM is read and processed on the server:

- Decode pixel array (pydicom)
- Apply rescale slope/intercept (when present)
- Apply window/level mapping to 8-bit
- Encode PNG and send to browser as Base64 data URL

This keeps the frontend thin and avoids implementing DICOM decoding in JavaScript.

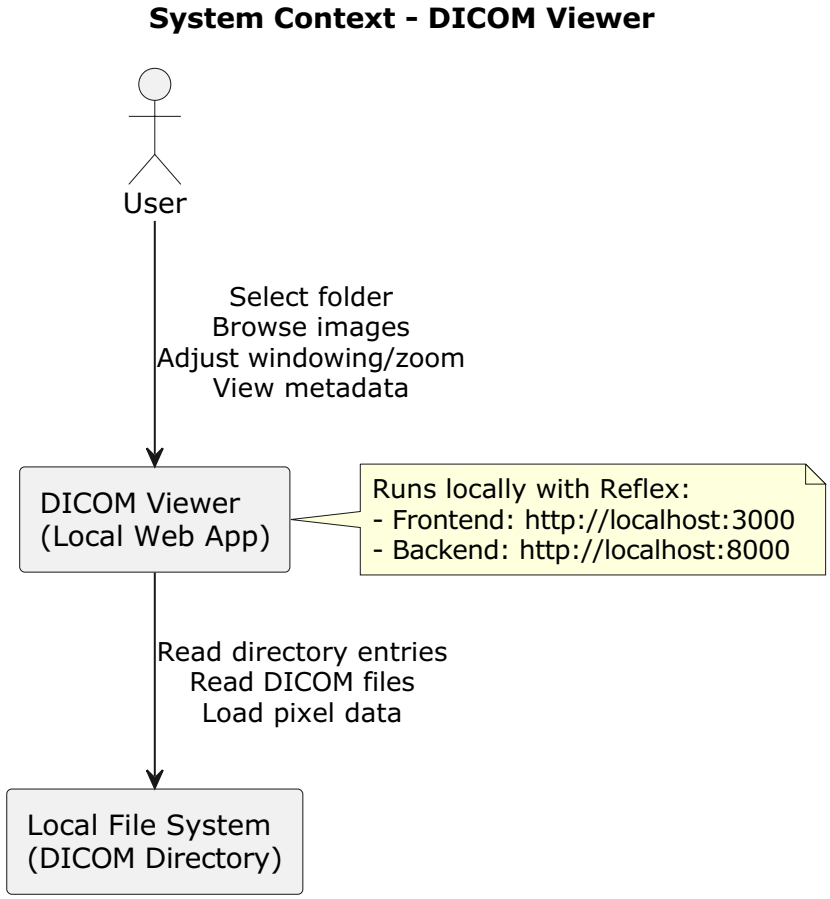
6.3 UX Workflow Steps

A single string (`workflow_step`) drives the “Select → List → Viewer” progression and the step indicator UI.

—

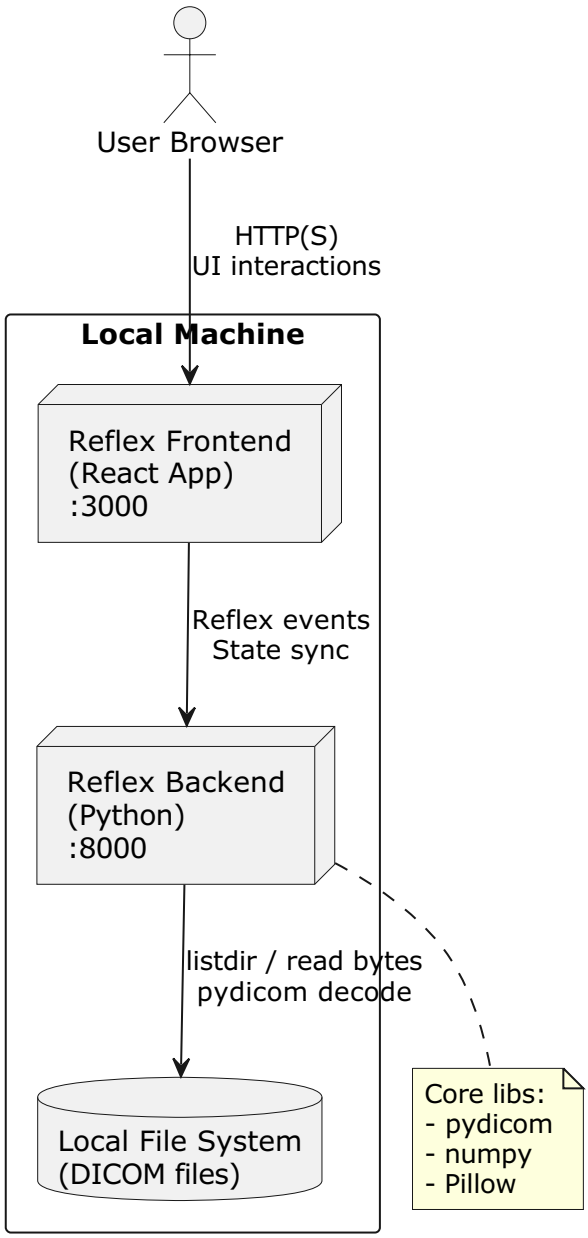
7. Architecture Diagrams (PlantUML)

7.1 System Context Diagram

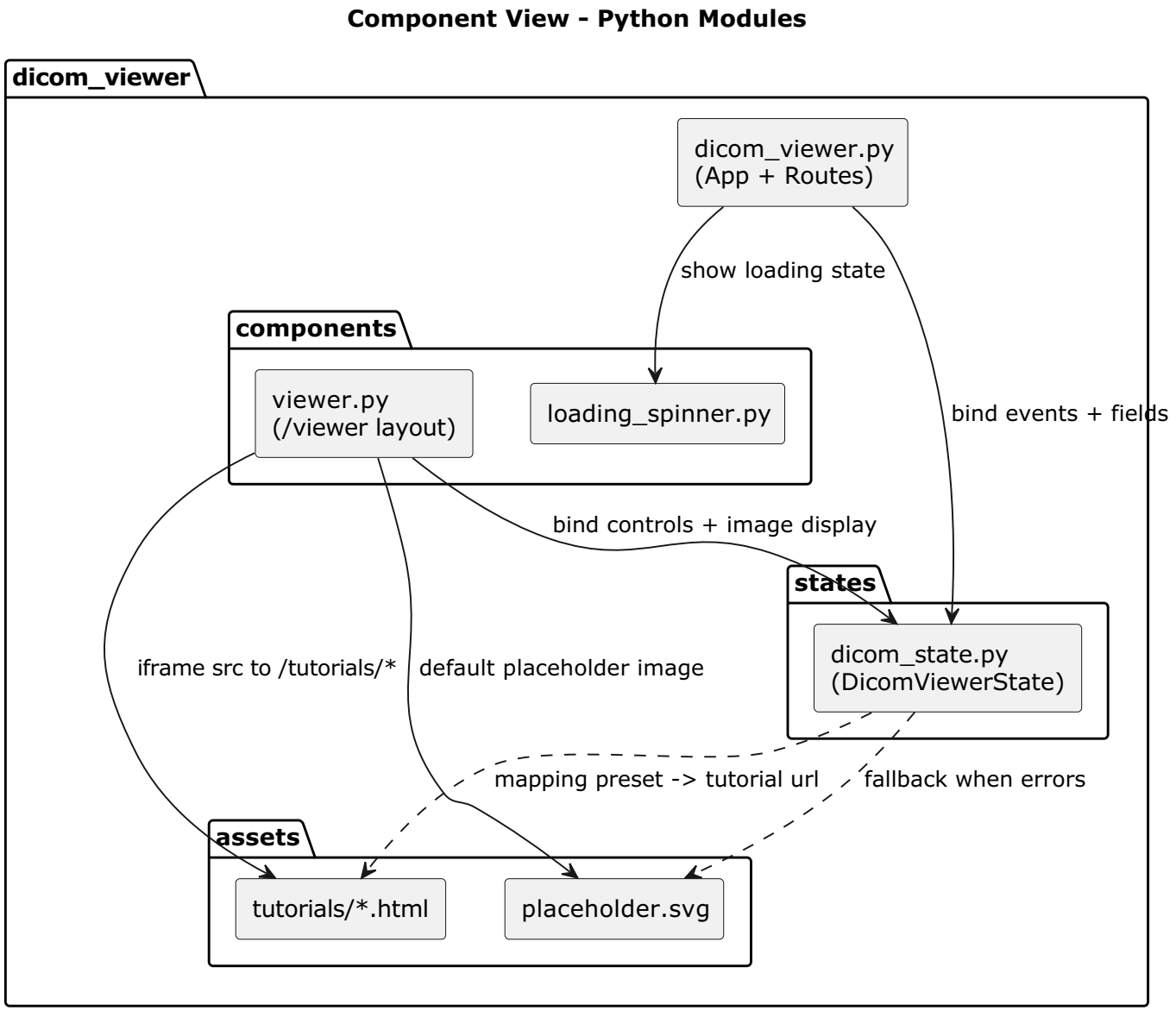


7.2 Container Diagram (Frontend/Backend within Reflex)

Container View - Reflex App



7.3 Component Diagram (Code-Level)



—

8. Core Runtime Model

8.1 Main State Object

Class: `DicomViewerState(rx.State)`

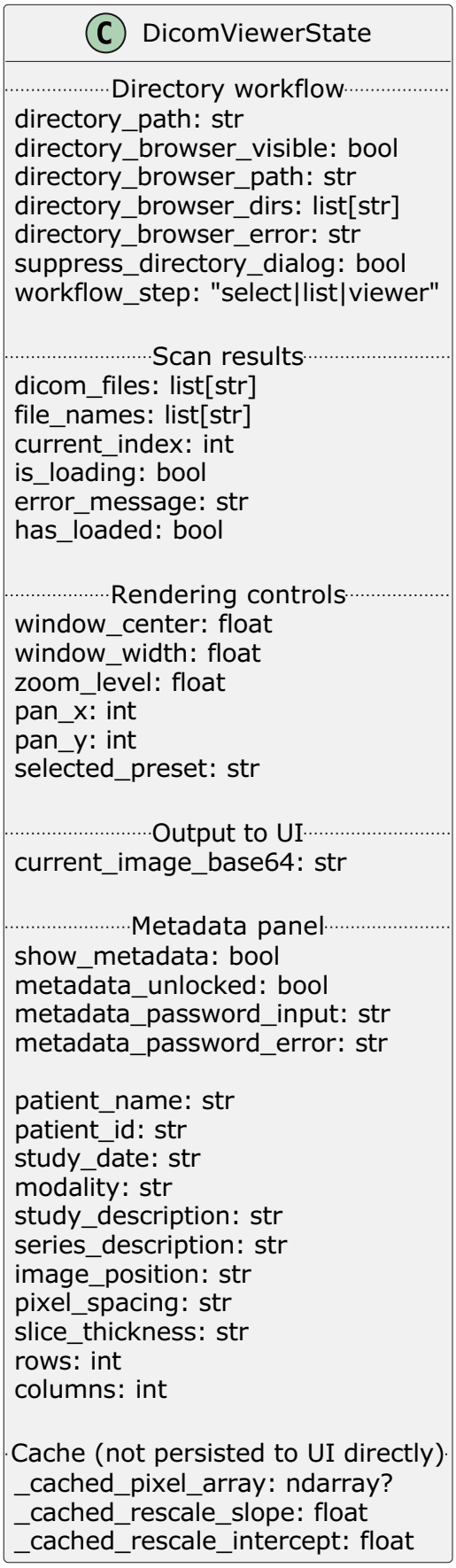
Responsibilities:

- Track workflow state and UI state (visible dialogs, selection indices, errors)
- Scan directories and build a list of DICOM files
- Load and cache pixel arrays and key rescale values
- Extract metadata fields for display
- Render current image to Base64 PNG
- Apply viewing controls (windowing, zoom, pan, presets, reset)

• Gate “protected metadata” behind a password

8.2 State Data (Conceptual)

Class View - DicomViewerState (Conceptual)



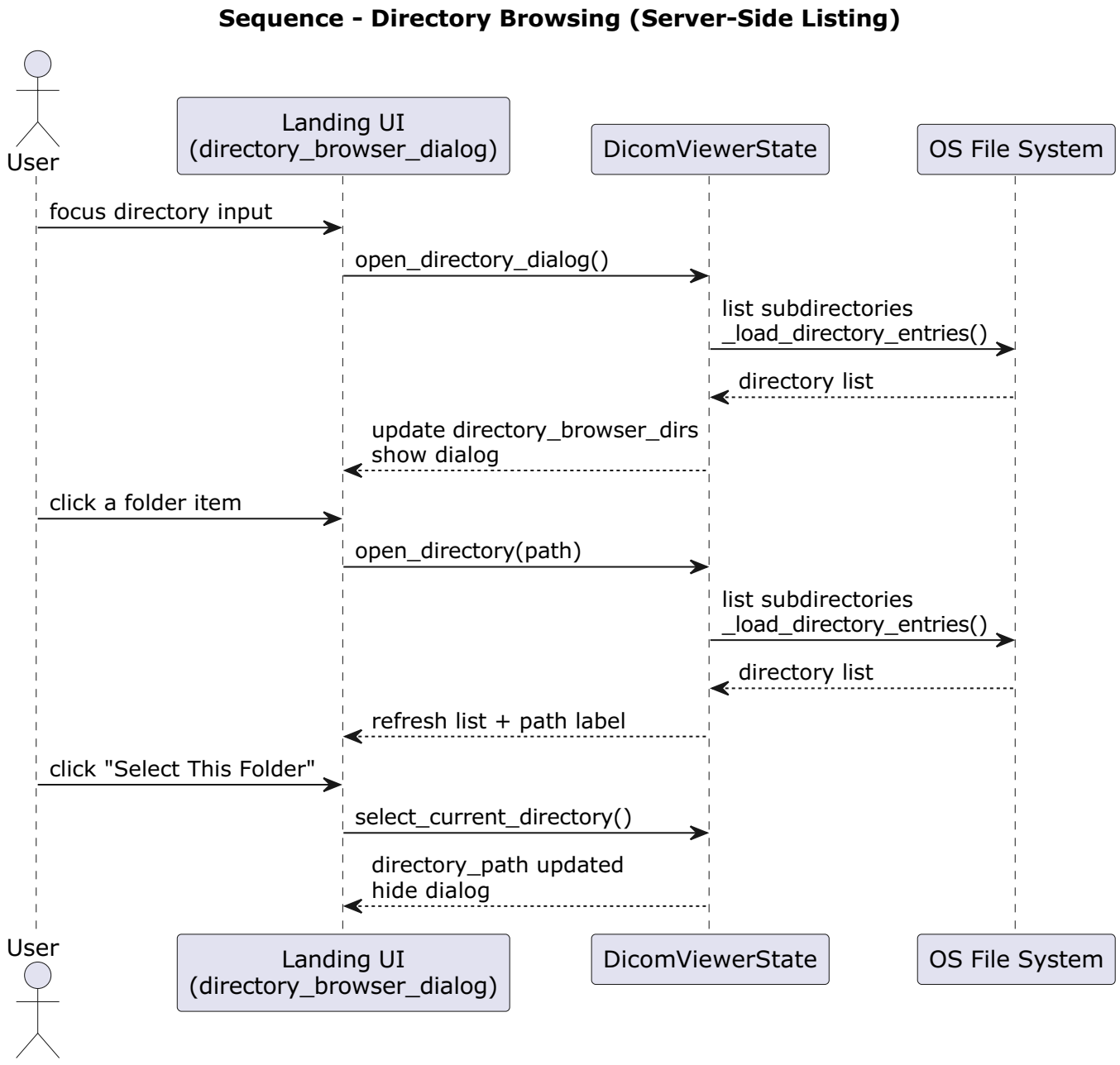
—

9. Key Workflows

9.1 Directory Browse & Selection

The “directory selector” is implemented as a server-side directory listing dialog:

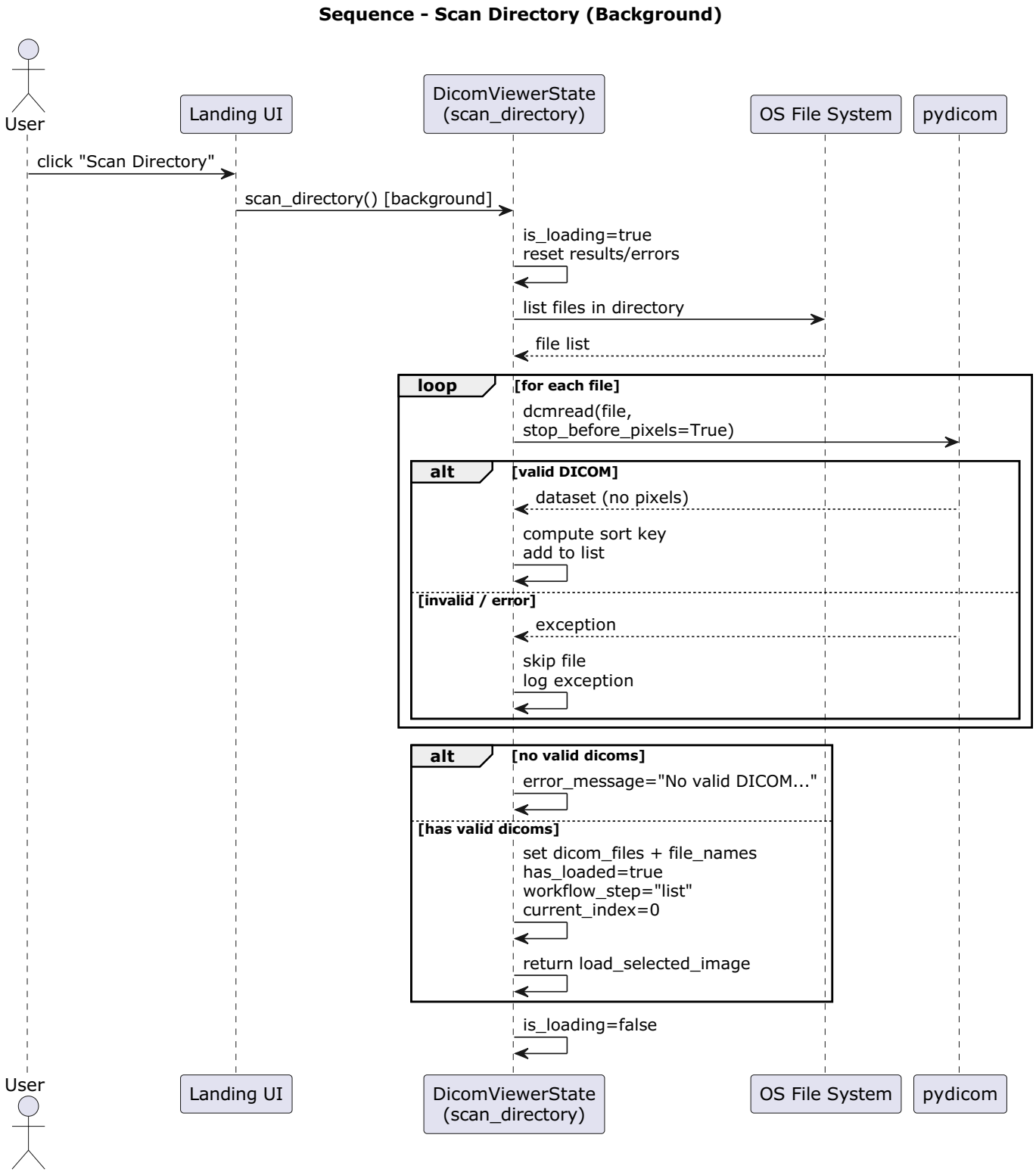
- `open_directory_dialog()` shows the dialog and lists subdirectories
- `open_directory(path)` moves into the chosen directory
- `go_up_directory()` navigates to parent
- `select_current_directory()` commits the browser path to `directory_path`



9.2 Scan Directory for DICOM Files (Background Event)

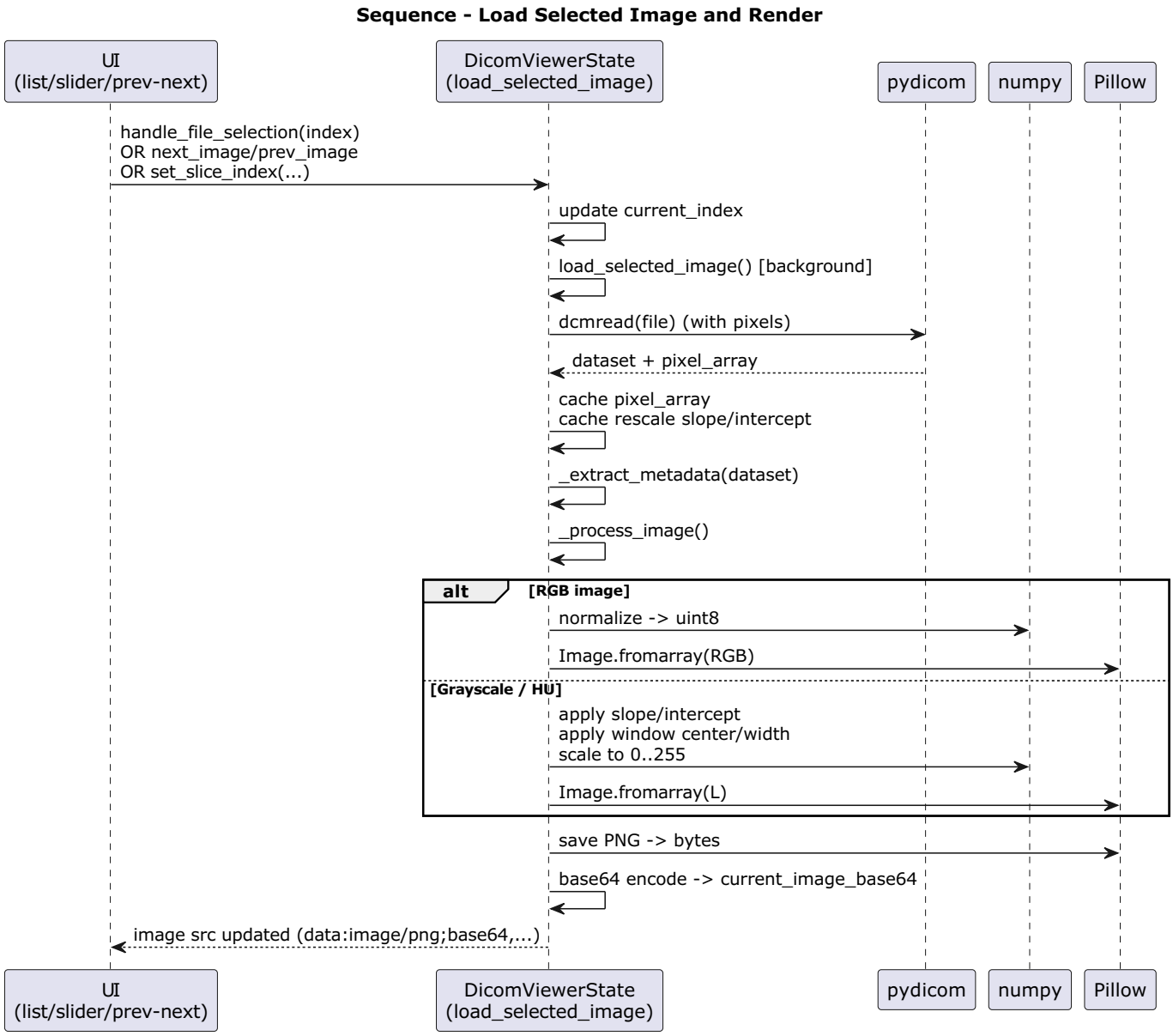
Scanning is a background event (`@rx.event(background=True)`) to keep UI responsive.

- Reads files in directory
- Attempts `pydicom.dcmread(..., stop_before_pixels=True)` for quick validation
- Sorts files using `_dicom_sort_key(...)`
- Sets `has_loaded`, `file_names`, `dicom_files`, then triggers `load_selected_image`



9.3 Load & Render Current Image

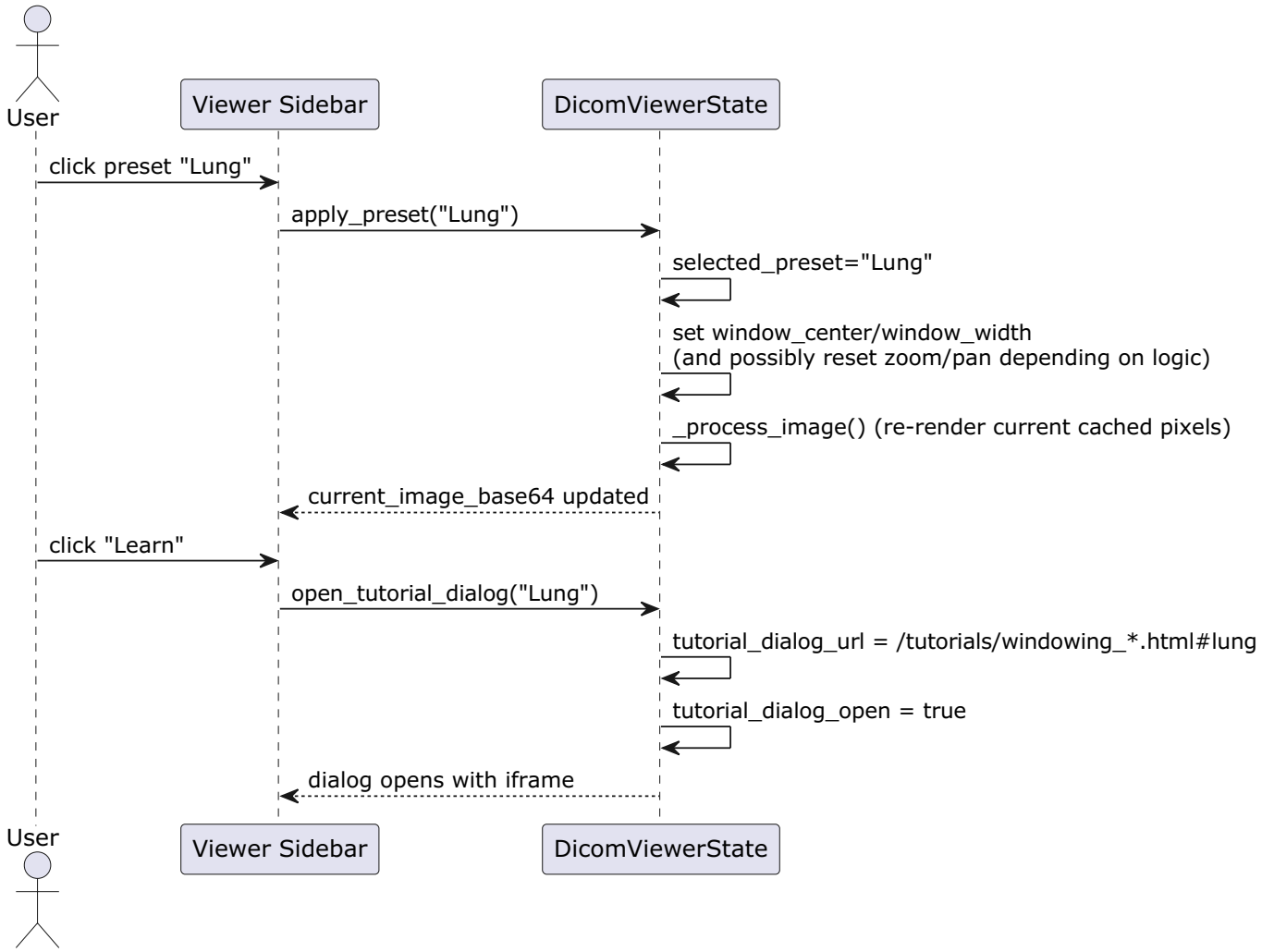
On selection or navigation, the app loads a DICOM, caches pixel data, extracts metadata, and renders Base64 PNG.



9.4 Window/Level + Presets + Tutorial Dialog

Presets set `window_center` and `window_width`, and optionally open a tutorial dialog with an iframe URL.

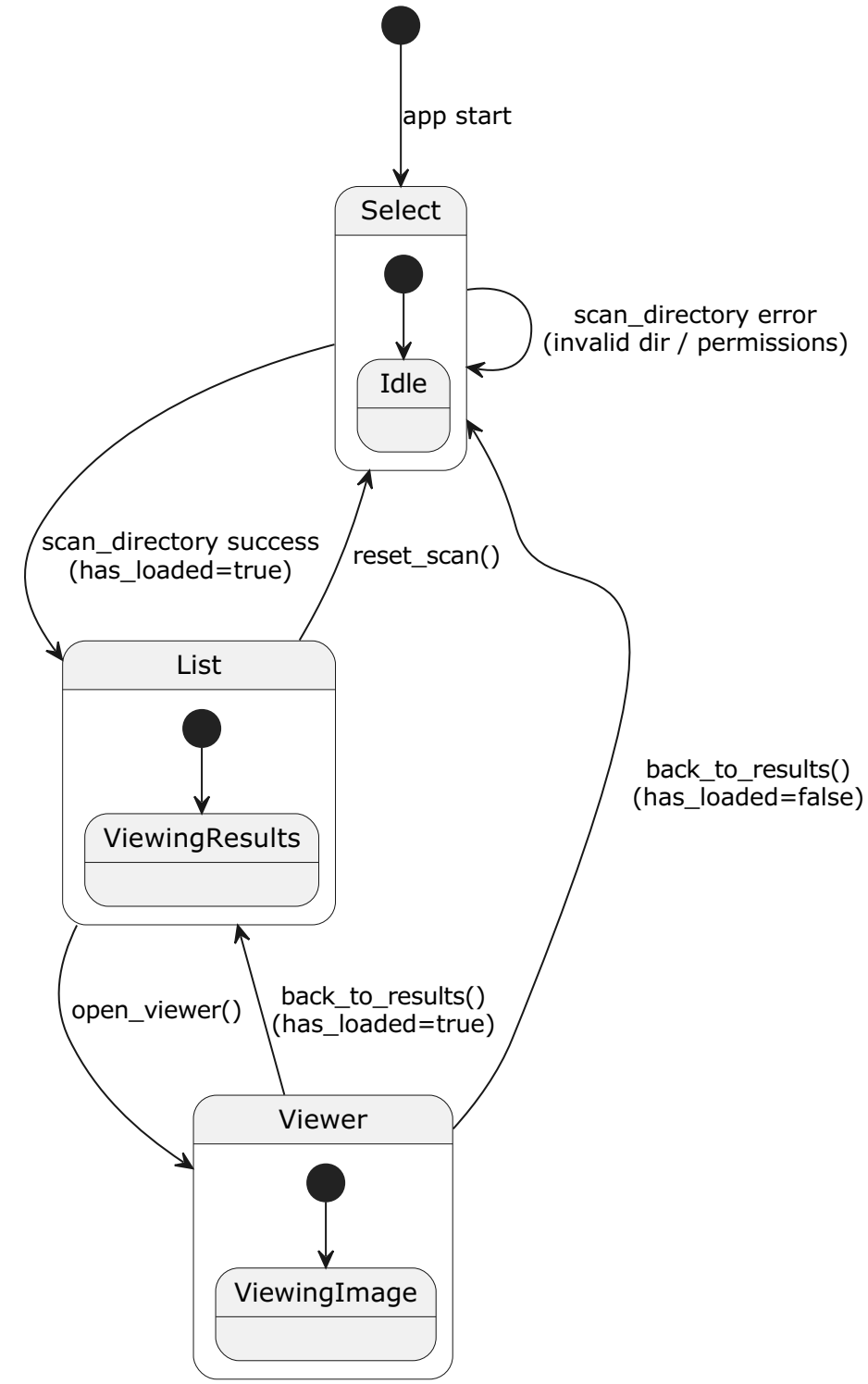
Sequence - Apply Preset and Re-render



9.5 Workflow State Machine

`workflow_step` drives the main user journey and UI indicator.

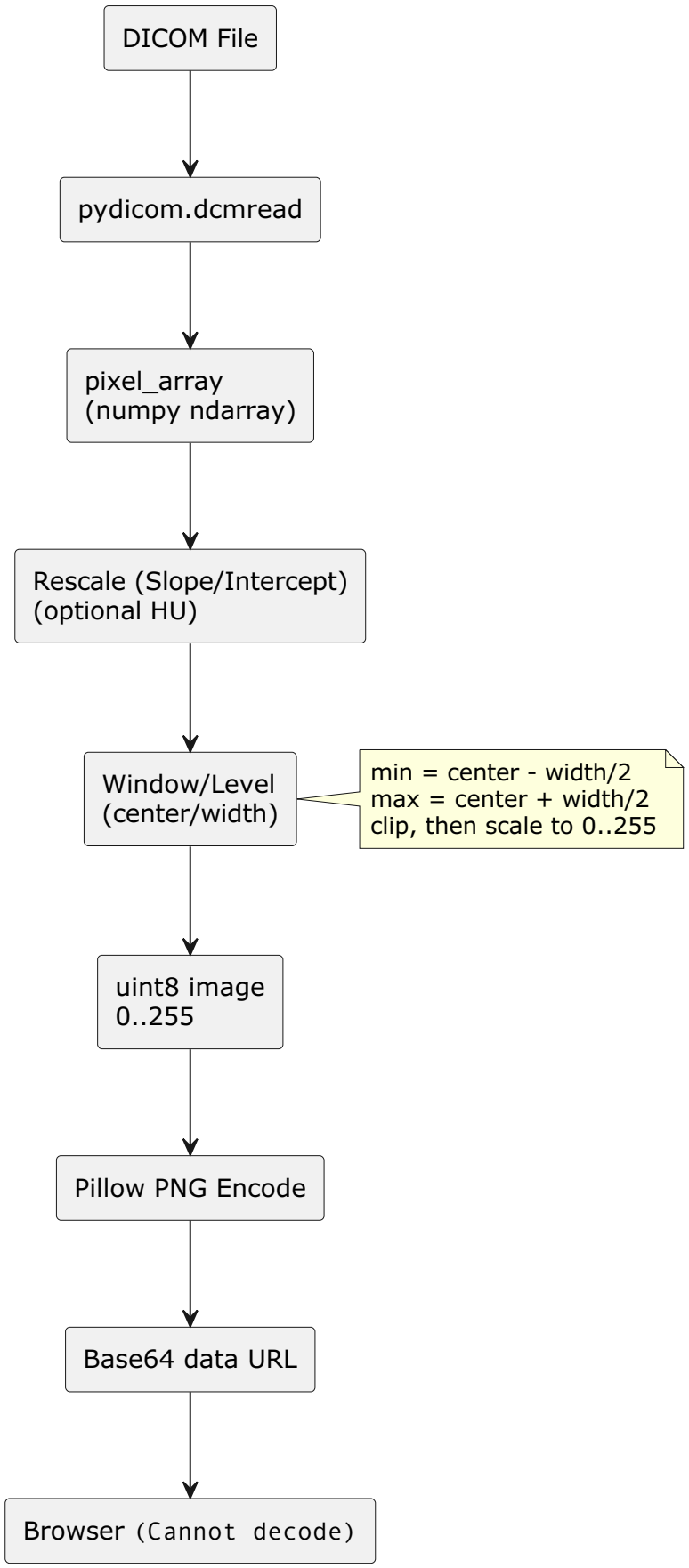
State Machine - workflow_step



—

10. Image Processing Pipeline (Data Flow)

Data Flow - DICOM to Displayable PNG

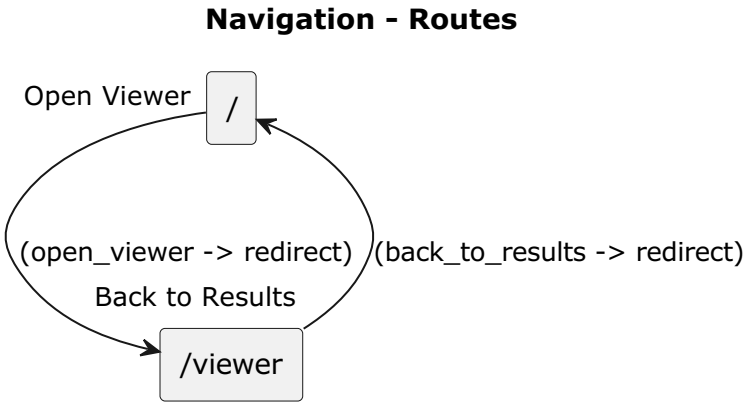


—

11. UI Pages and Navigation

11.1 Routes

- `/` : Landing page (directory selection, scan results)
- `/viewer` : Viewer page (image + sidebars)



—

12. Configuration

12.1 Environment Variables

- `PUBLIC_DICOM_DIR`
- Used to initialize `directory_path` and `directory_browser_root` defaults.
- `DICOM_METADATA_PASSWORD`
- Default: `"dicom"`
- Controls unlocking protected metadata fields.

12.2 Runtime Defaults

- Default DICOM directory on macOS: `/Users/Shared/DICOM`
- Otherwise default browser dir: `Path.home()` (or `/` fallback)

—

13. Error Handling & Edge Cases

13.1 Directory Access

- Non-existent or invalid folder: user-facing error
- Permission errors: special message guiding macOS privacy settings

13.2 Invalid DICOM Files

- Skipped during scan (logged, not fatal)
- Rendering failures show placeholder (`/placeholder.svg`) and error message where appropriate

13.3 Multi-frame / 3D

- If pixel array is not RGB but has extra dimensions, the implementation attempts to take the first slice/frame.

—

14. Security & Privacy Considerations

1. **Local-first design**

- By default, data is read from local disk by the backend process.
  2. **Metadata gating**

- Patient name/ID and some date fields can be hidden until password unlock.
  3. **Directory browsing**

- The directory browser is server-driven. Consider restricting it to an allowed root (e.g., enforce `PUBLIC_DICOM_DIR`) if the app is used in shared environments.
  4. **PHI**

- Even with gating, PHI may still be present in DICOM pixel data overlays or other tags not displayed. Treat the app as PHI-capable and follow local compliance requirements.

—

15. Performance Characteristics

- **Scan time** scales with number of files in the chosen folder:

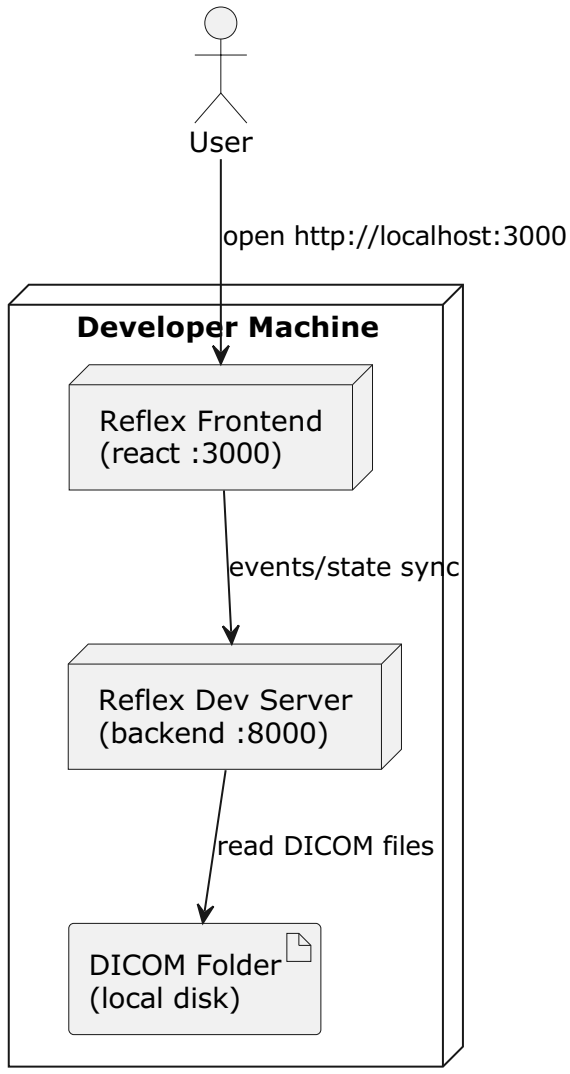
- Uses `stop_before_pixels=True` during scan for speed

- **Rendering** does full PNG encode each time windowing changes:
- Uses cached pixel array to avoid repeated DICOM decode for the same image
- **Base64 images** increase memory/bandwidth vs. serving binary PNG:
- Acceptable for local workflows, but can be optimized later

—

16. Deployment Diagram

Deployment - Local Development



—

17. Testing Strategy (Recommended)

> The repository currently focuses on app functionality; below is a recommended testing approach.

17.1 Unit Tests

- `_dicom_sort_key(ds, path)` sorting correctness
- `_process_image()` windowing math & edge cases (width=0, RGB vs grayscale)
- `_extract_metadata()` field extraction with mocked pydicom datasets

17.2 Integration Tests

- Scan a fixture directory with:
- valid DICOMs
- invalid files
- permission denied directory simulation (where possible)

17.3 E2E Tests (Playwright)

- Landing page: open dialog, select directory, scan, list appears
- Viewer: open viewer, slider changes image, presets change image, metadata panel toggle, unlock validation

—

18. Future Enhancements

1. **Series/Study grouping**
   - Build a hierarchy: Study → Series → Instances, rather than a flat directory list.
2. **Root restriction for directory browser**
   - Enforce a safe root directory for browsing.
3. **Image serving optimization**
   - Serve PNG via backend endpoint + cache by (file, windowing, zoom) instead of Base64 inline.
4. **Multi-frame support**
   - Cine controls for multi-frame DICOMs.
5. **More metadata**
   - Offer a “full tag view” with search/filter, with PHI controls.
6. **Export**
   - Export current view as PNG, export metadata as JSON.

—

Appendix A — Mapping of Important Files to Responsibilities

- `dicom_viewer/dicom_viewer.py`
- Landing page UI, directory dialog, results list, app routes (`/`, `/viewer`)
- `dicom_viewer/components/viewer.py`
- Viewer page layout, metadata panel, sidebar controls, tutorial dialog
- `dicom_viewer/states/dicom_state.py`
- All state, directory scan logic, DICOM decode, windowing, zoom/pan, metadata extraction
- `assets/tutorials/*.html`
- Embedded windowing explanations (same-origin for iframe compatibility)

—

—

Appendix B — Event I/O Reference (Reflex `@rx.event`)

> Notes:
> - “Background = Yes” means the event runs asynchronously in the backend (`background=True`).
> - Some events call internal helpers (e.g., `_process_image()`), which **also** update `current_image_base64`.
> - “UI Binding (file:line)” points to where the event is wired from the UI layer.

Directory & Workflow

Event	Background	Inputs	UI Binding (file:line)	State Outputs (mutations)
`set_directory`	No	path: str	dicom_viewer.py:33	directory_path, error_message
`suppress_directory_dialog_once`	No	—	dicom_viewer.py:42	suppress_directory_dial

Event	Background	Inputs	UI Binding (file:line)	State Outputs (mutations)
`open_directory_dialog`	No	—	dicom_viewer.py:34	directory_browser_dirs (_load_directory_entries) directory_browser_error (via _load_directory_entries) directory_browser_path directory_browser_visib suppress_directory_dial
`close_directory_dialog`	No	—	dicom_viewer.py:118	directory_browser_visib
`go_up_directory`	No	—	dicom_viewer.py:107	directory_browser_dirs (_load_directory_entries) directory_browser_error (via _load_directory_entries) directory_browser_path
`open_directory`	No	path: str	dicom_viewer.py:146	directory_browser_dirs (_load_directory_entries) directory_browser_error (via _load_directory_entries) directory_browser_path
`select_current_directory`	No	—	dicom_viewer.py:113	directory_browser_visib directory_path, error_message
`scan_directory`	Yes	—	dicom_viewer.py:43	current_image_base64, current_index, dicom_fil error_message, file_names, has_loaded, is_loading, metadata_password_er metadata_password_inp metadata_unlocked, workflow_step
`reset_scan`	No	—	dicom_viewer.py:198	current_image_base64, current_index, dicom_fil error_message, file_names, has_loaded, metadata_password_er

Event	Background	Inputs	UI Binding (file:line)	State Outputs (mutations)
				metadata_password_inp metadata_unlocked, workflow_step
`open_viewer`	No	—	dicom_viewer.py:204	workflow_step
`back_to_results`	No	—	components/ viewer.py:411	workflow_step

Scan Results Selection

Event	Background	Inputs	UI Binding (file:line)	State Outputs (mutations)	Ret
`handle_file_selection`	No	index: int	dicom_viewer.py:168	current_index	Dico
`load_selected_image`	Yes	—	Not directly wired in UI (internal/utility or called by other events).	_cached_pixel_array, _cached_rescale_intercept, _cached_rescale_slope, current_image_base64 (via _process_image), error_message, image_position, modality, patient_id, patient_name, pixel_spacing, rows, columns, series_description, slice_thickness, study_date, study_description	—
`next_image`	No	—	components/ viewer.py:237	current_index	Dico

Event	Background	Inputs	UI Binding (file:line)	State Outputs (mutations)	Ret
`prev_image`	No	—	components/viewer.py:219	current_index	Dic
`set_slice_index`	No	value: str	components/viewer.py:230	current_index	Dic

Viewer Controls

Event	Background	Inputs	UI Binding (file:line)	State Outputs (mutations)	Return / Next
`update_window_center`	No	value: str	components/viewer.py:344	current_image_base64 (via _process_image), selected_preset, window_center	—
`update_window_width`	No	value: str	components/viewer.py:366	current_image_base64 (via _process_image), selected_preset, window_width	—
`apply_preset`	No	preset_name: str	components/viewer.py:60	current_image_base64 (via _process_image), selected_preset, window_center, window_width	—
`reset_view`	No	—	components/viewer.py:302	current_image_base64 (via _process_image), pan_x, pan_y,	—

Event	Background	Inputs	UI Binding (file:line)	State Outputs (mutations)	Return / Next
				selected_preset, window_center, window_width, zoom_level	
`set_zoom`	No	value: float	Not directly wired in UI (internal/utility or called by other events).	zoom_level	—
`zoom_in`	No	—	components/viewer.py:262	zoom_level (via set_zoom)	—
`zoom_out`	No	—	components/viewer.py:252	zoom_level (via set_zoom)	—
`reset_zoom`	No	—	components/viewer.py:282	pan_x, pan_y, zoom_level	—
`pan_control`	No	dx: int, dy: int	components/viewer.py:271, components/viewer.py:277, components/viewer.py:288, components/viewer.py:295	pan_x, pan_y	—

Metadata Panel

Event	Background	Inputs	UI Binding (file:line)	State Outputs (mutations)	Return / Next
`toggle_metadata`	No	—	components/viewer.py:111, components/viewer.py:420	show_metadata	—
`update_metadata_password`	No	value: str	components/viewer.py:153	metadata_password_input, metadata_password_error	—

Event	Background	Inputs	UI Binding (file:line)	State Outputs (mutations)	Return / Next
`unlock_metadata`	No	—	components/viewer.py:158	metadata_password_error, metadata_password_input, metadata_unlocked	—
`lock_metadata`	No	—	components/viewer.py:140	metadata_password_error, metadata_password_input, metadata_unlocked	—

Tutorial Dialog

Event	Background	Inputs	UI Binding (file:line)	State Outputs (mutations)	Return / Next	Note
`set_tooltip_language`	No	value: str	components/viewer.py:386	tooltip_language	—	Switc toolti langu supp
`open_tutorial_dialog`	No	preset: str	components/viewer.py:75	tutorial_dialog_open, tutorial_dialog_title, tutorial_dialog_url	—	Sets URL origin `asse` `tuto` `*.ht` anch
`close_tutorial_dialog`	No	—	components/viewer.py:22	tutorial_dialog_open, tutorial_dialog_title, tutorial_dialog_url	—	Close and c state

—

Appendix C — Window/Level Presets (WC/WL and WW)

> Notes:
> - Values are the project's built-in defaults in `apply_preset()`.
> - WC/WL and WW are typically interpreted in **HU** (Hounsfield Units) for CT when rescale slope/intercept are present.
> - Default reset values: **WC=40, WW=400** (same as "Soft Tissue").

Preset	Window Center (WL)	Window Width (WW)
Abdomen	50	400
Adrenal	40	300
Angio Bone Sub	300	1200
Arterial	150	600
Body	40	450
Bone	300	1500
Brain	40	80
Cardiac	75	350
Colon/Bowel	50	400
CTA Head/Neck	180	700
CTA/Vascular	150	600
Extremity/MSK	40	350
Gallbladder	30	200
Head/Neck	50	250
Kidney	30	300
Liver	60	150
Lung	-600	1500
Lung HRCT	-700	1200
Mediastinum	40	350
Orbits	50	300
Pancreas	50	200
Pelvis	50	450
Sinus	50	300
Skin/Subcutaneous	50	250
Soft Tissue	40	400

Preset	Window Center (WL)	Window Width (WW)
Spine	30	300
Stroke	35	30
Subdural	50	130
Temporal Bone	700	4000
Trauma	50	500
Venous	100	500

—