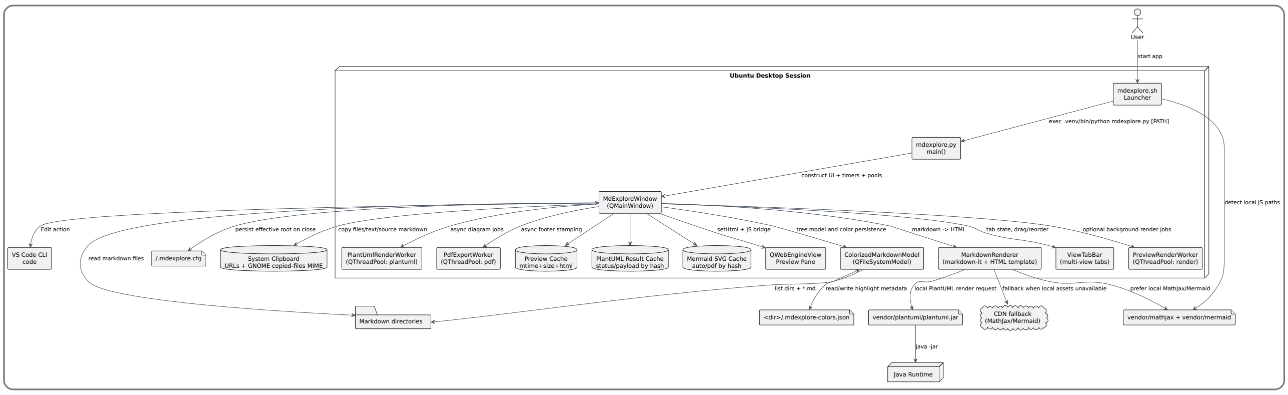
mdexplore UML

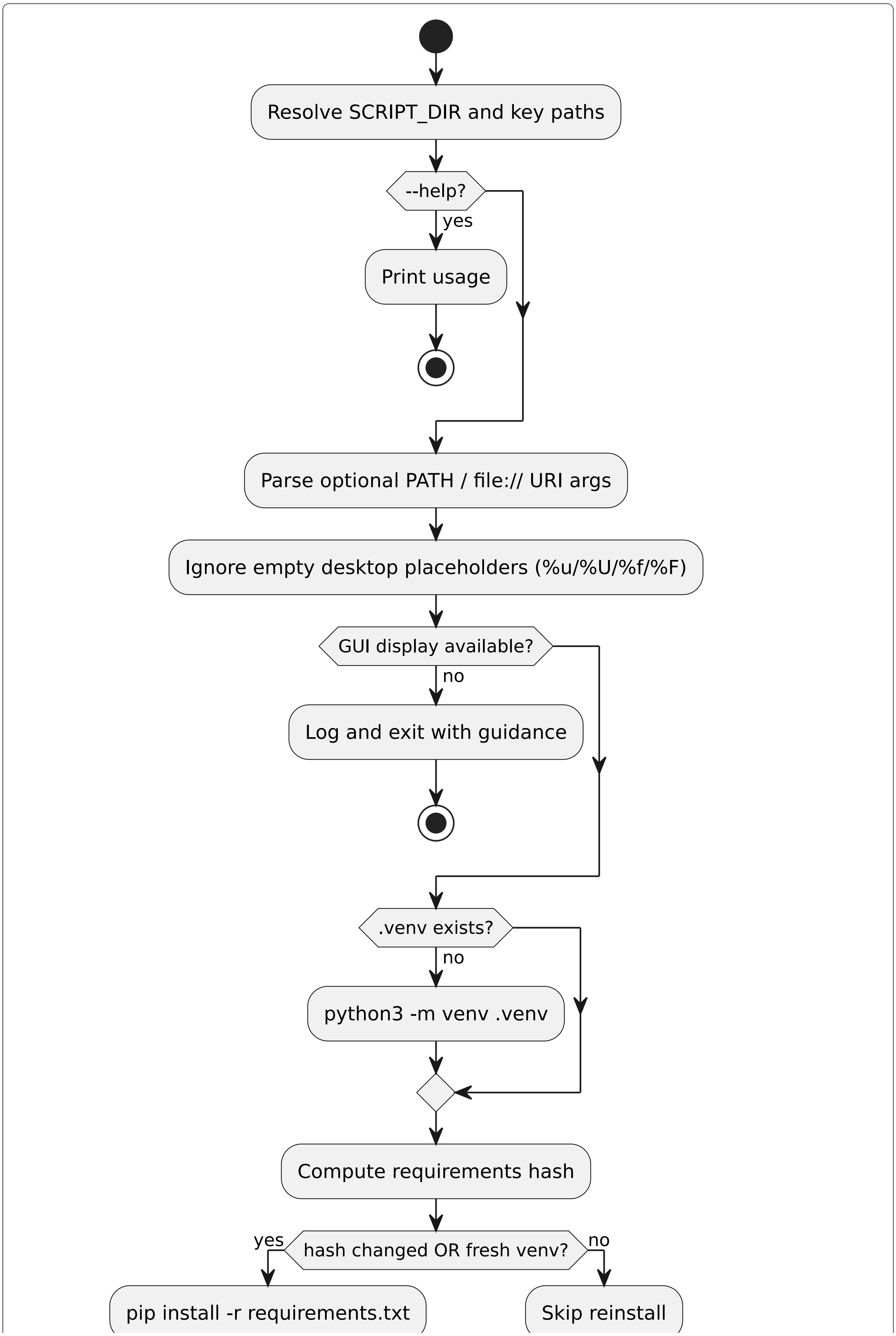
This document provides a comprehensive PlantUML view of the current `mdexplore` implementation. All diagrams are embedded so they can be rendered directly by mdexplore (or any Markdown viewer with PlantUML support).

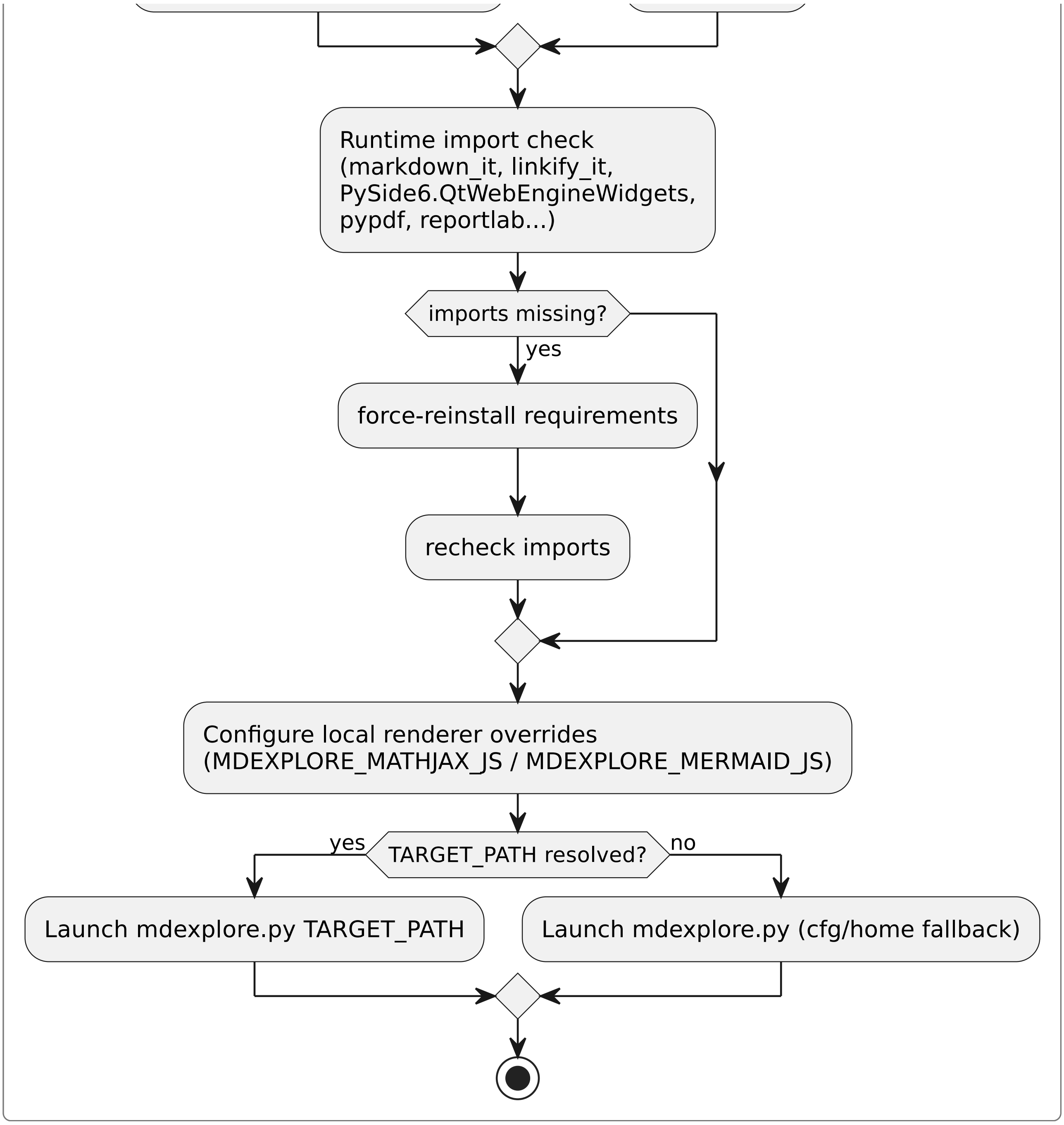
Detailed render/caching forks (GUI vs PDF, JS Mermaid vs Rust Mermaid, cache mode ownership, and restore behavior) are intentionally documented in `RENDER-PATHS.md`. This UML file keeps those areas abstracted at system/class boundaries to avoid duplicating low-level render logic across two docs.

1. System Architecture

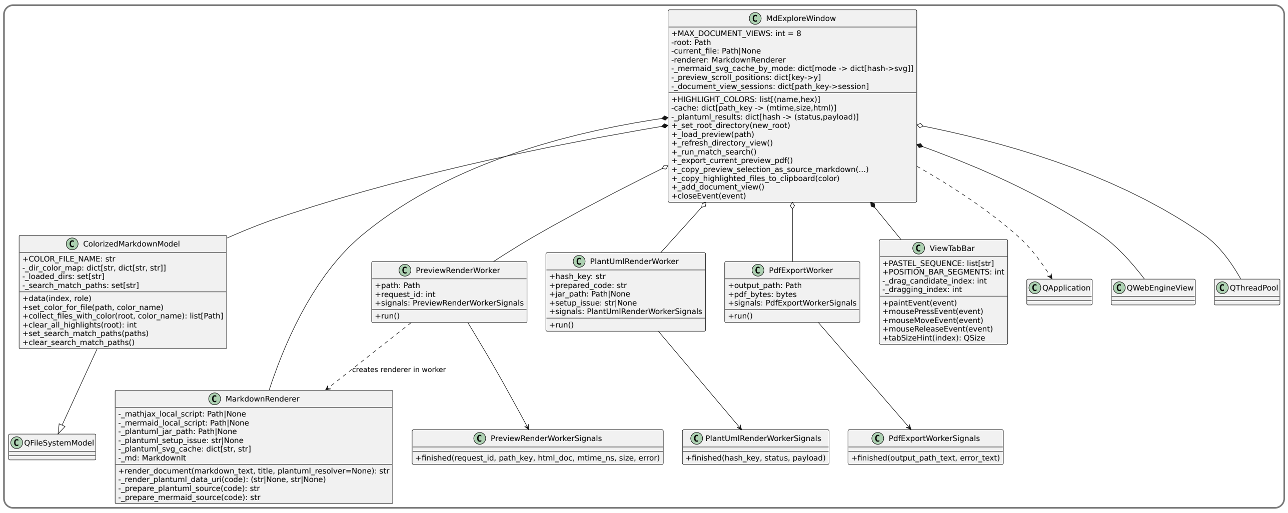


2. Launcher Runtime Activity (`mdexplore.sh`)


Resolve SCRIPT_DIR and key paths
--help?
yes
Print usage
Parse optional PATH / file:// URI args
Ignore empty desktop placeholders (%u/%U/%f/%F)
GUI display available?
no
Log and exit with guidance
.venv exists?
no
python3 -m venv .venv
Compute requirements hash
yes
hash changed OR fresh venv?
no
pip install -r requirements.txt
Skip reinstall

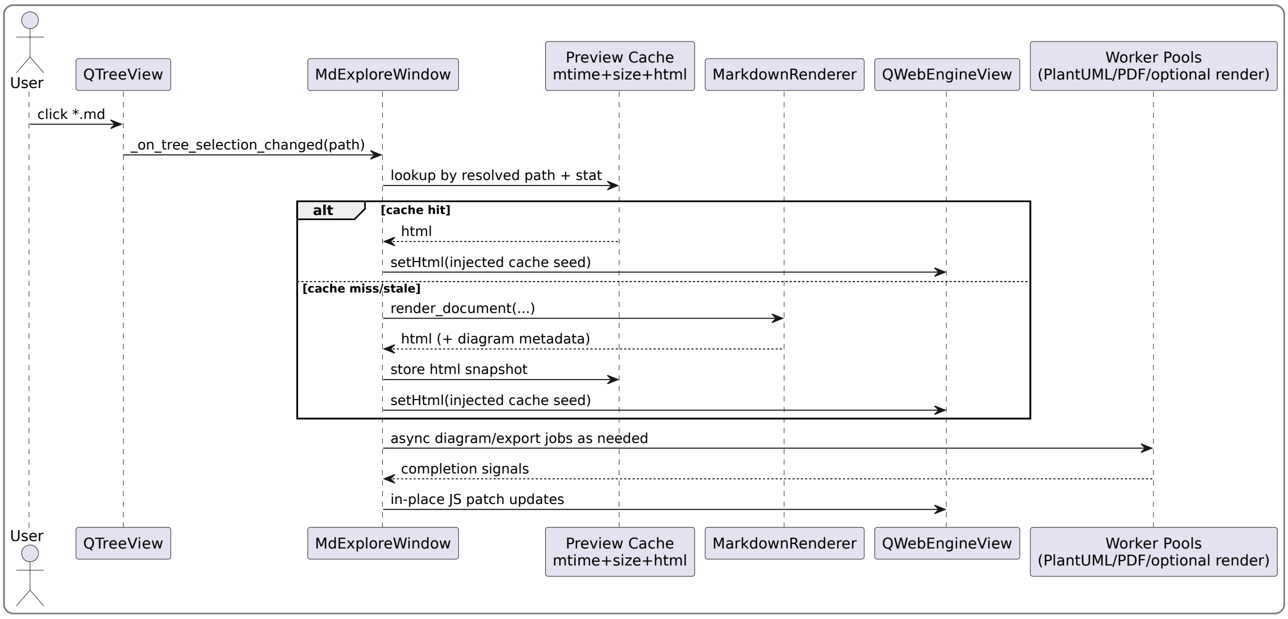

Runtime import check
(markdown_it, linkify_it,
PySide6.QtWebEngineWidgets,
pypdf, reportlab...)
imports missing?
yes
force-reinstall requirements
recheck imports
Configure local renderer overrides
(MDEXPLORE_MATHJAX_JS / MDEXPLORE_MERMAID_JS)
TARGET_PATH resolved?
yes
no
Launch mdexplore.py TARGET_PATH
Launch mdexplore.py (cfg/home fallback)

3. Core Class Diagram

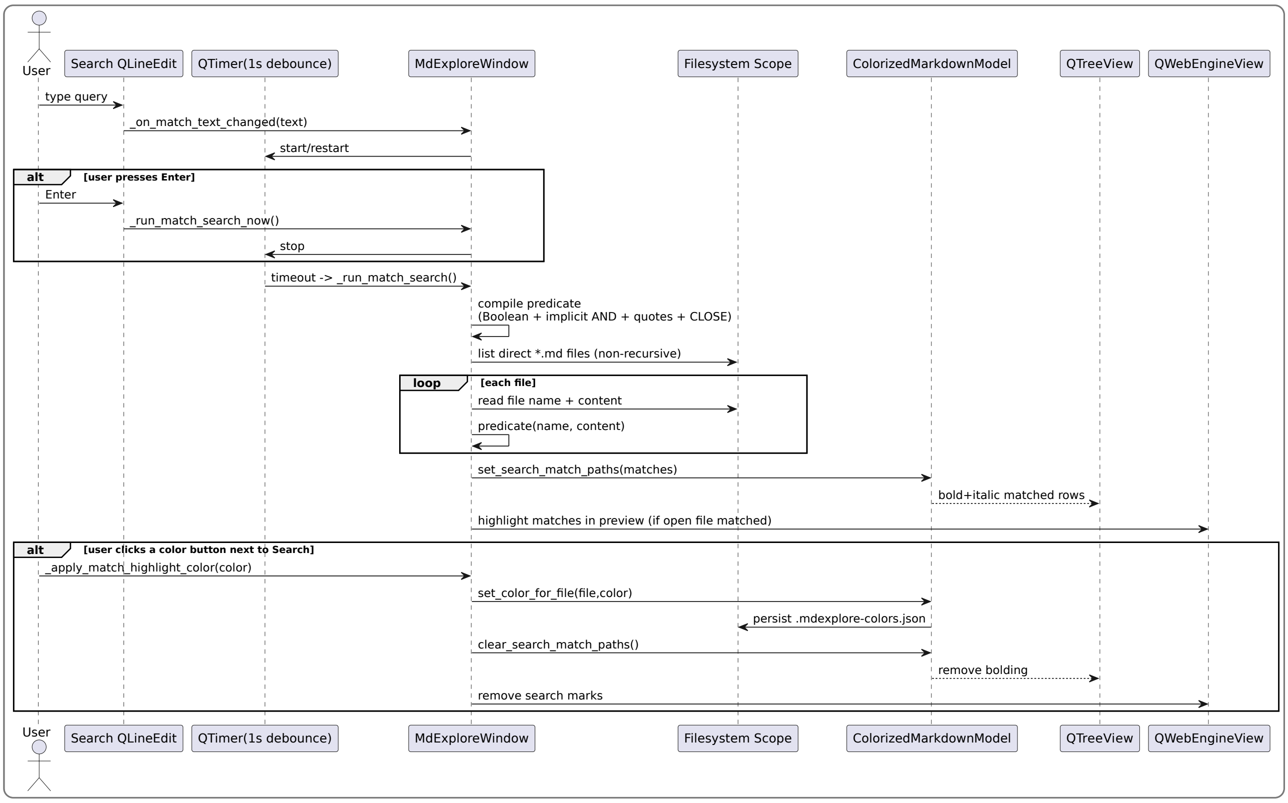

MdExploreWindow
+MAX_DOCUMENT_VIEWS: int = 8
-root: Path
-current_file: Path|None
-renderer: MarkdownRenderer
-_mermaid_svg_cache_by_mode: dict[mode -> dict[hash->svg]]
-_preview_scroll_positions: dict[key->y]
-_document_view_sessions: dict[path_key->session]
+HIGHLIGHT_COLORS: list[(name,hex)]
-cache: dict[path_key -> (mtime,size,html)]
-_plantuml_results: dict[hash -> (status,payload)]
+_set_root_directory(new_root)
+_load_preview(path)
+_refresh_directory_view()
+_run_match_search()
+_export_current_preview_pdf()
+_copy_preview_selection_as_source_markdown(...)
+_copy_highlighted_files_to_clipboard(color)
+_add_document_view()
+closeEvent(event)
ColorizedMarkdownModel
+COLOR_FILE_NAME: str
-_dir_color_map: dict[str, dict[str, str]]
-_loaded_dirs: set[str]
-_search_match_paths: set[str]
+data(index, role)
+set_color_for_file(path, color_name)
+collect_files_with_color(root, color_name): list[Path]
+clear_all_highlights(root): int
+set_search_match_paths(paths)
+clear_search_match_paths()
PreviewRenderWorker
+path: Path
+request_id: int
+signals: PreviewRenderWorkerSignals
+run()
PlantUmlRenderWorker
+hash_key: str
+prepared_code: str
+jar_path: Path|None
+setup_issue: str|None
+signals: PlantUmlRenderWorkerSignals
+run()
PdfExportWorker
+output_path: Path
+pdf_bytes: bytes
+signals: PdfExportWorkerSignals
+run()
ViewTabBar
+PASTEL_SEQUENCE: list[str]
+POSITION_BAR_SEGMENTS: int
-_drag_candidate_index: int
-_dragging_index: int
+paintEvent(event)
+mousePressEvent(event)
+mouseMoveEvent(event)
+mouseReleaseEvent(event)
+tabSizeHint(index): QSize
QApplication
QWebEngineView
QThreadPool
creates renderer in worker
QFileSystemModel
MarkdownRenderer
-_mathjax_local_script: Path|None
-_mermaid_local_script: Path|None
-_plantuml_jar_path: Path|None
-_plantuml_setup_issue: str|None
-_plantuml_svg_cache: dict[str, str]
-_md: MarkdownIt
+render_document(markdown_text, title, plantuml_resolver=None): str
-_render_plantuml_data_uri(code): (str|None, str|None)
-_prepare_plantuml_source(code): str
-_prepare_mermaid_source(code): str
PreviewRenderWorkerSignals
+finished(request_id, path_key, html_doc, mtime_ns, size, error)
PlantUmlRenderWorkerSignals
+finished(hash_key, status, payload)
PdfExportWorkerSignals
+finished(output_path_text, error_text)

4. Preview Pipeline (Abstracted)

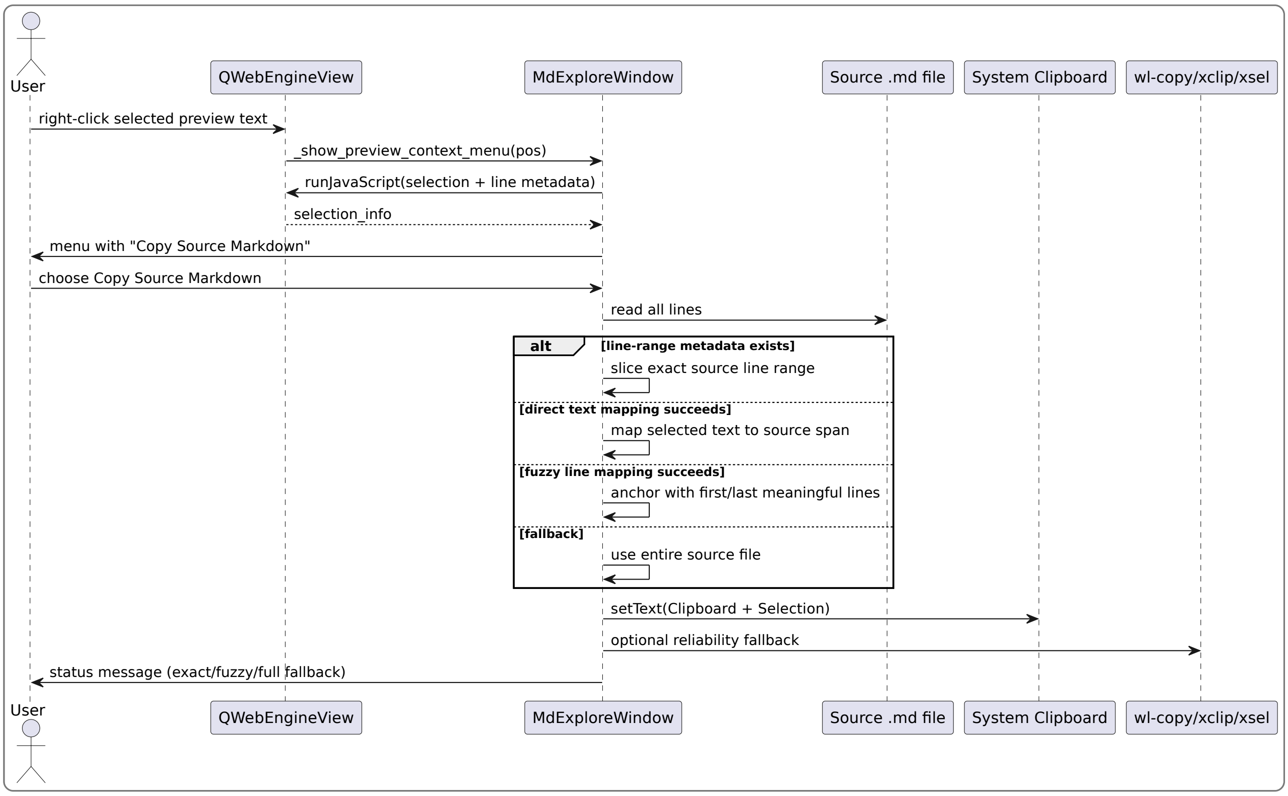
The detailed render/cache branches are covered in `RENDER-PATHS.md`. This section intentionally keeps an architectural boundary view only.



5. Search + Highlight + Apply Color Flow

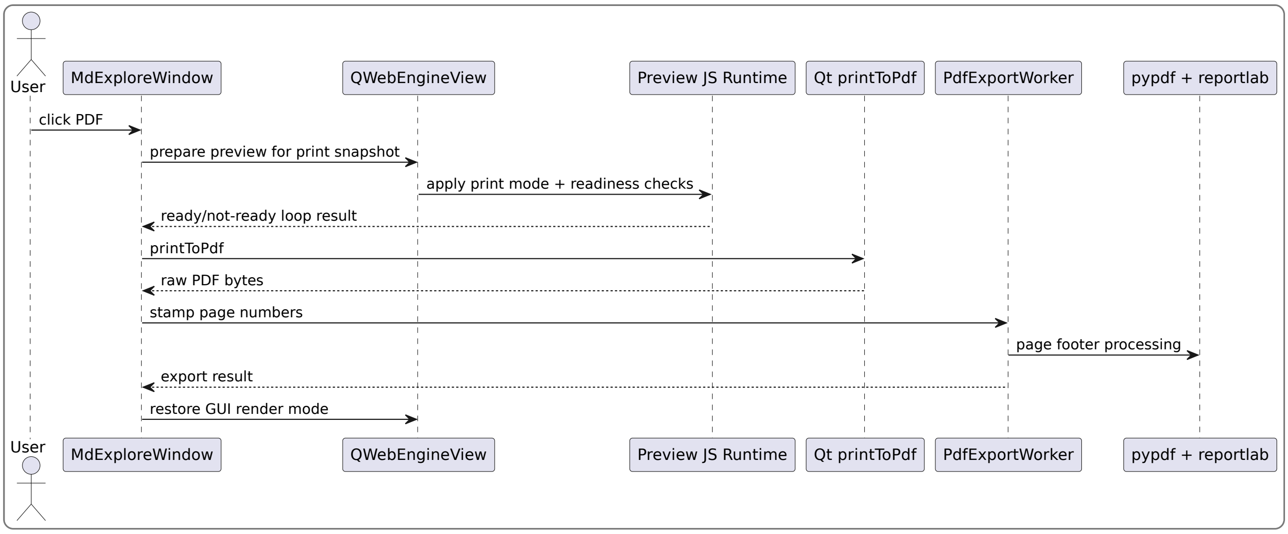


6. Preview Context Menu: Copy Source Markdown

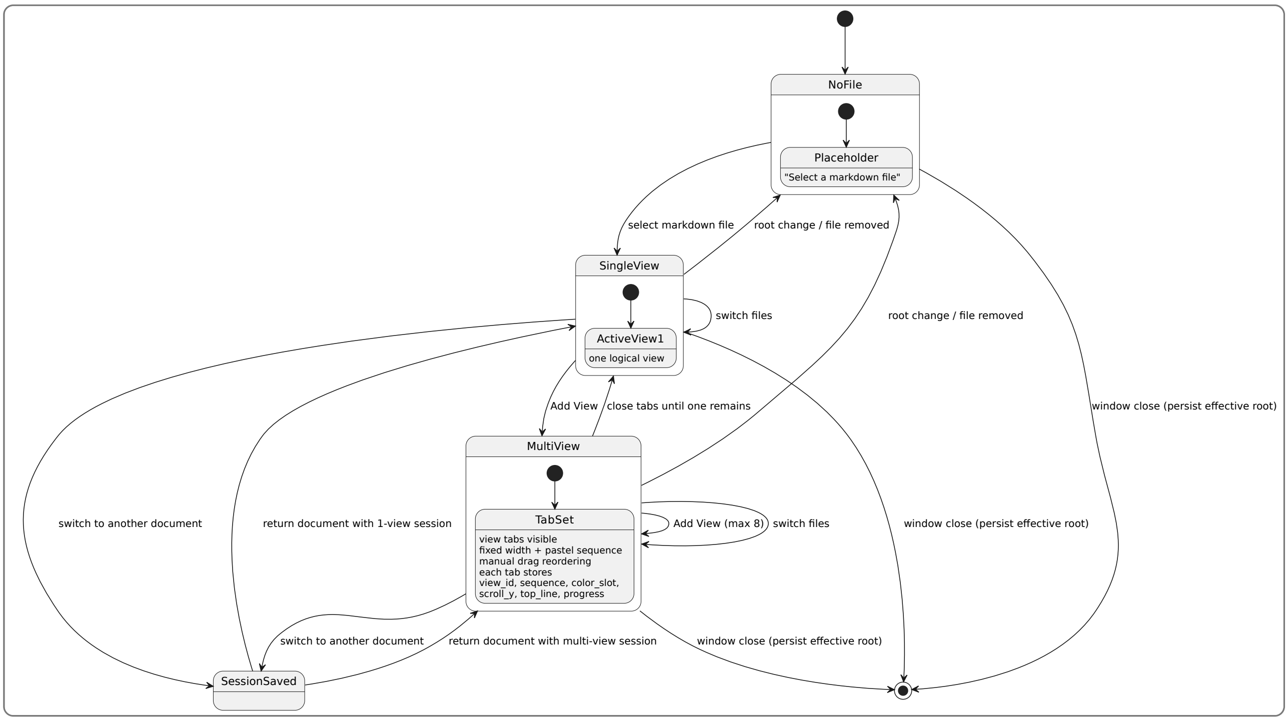

User
QWebEngineView
MdExploreWindow
Source .md file
System Clipboard
wl-copy/xclip/xsel
right-click selected preview text
_show_preview_context_menu(pos)
runJavaScript(selection + line metadata)
selection_info
menu with "Copy Source Markdown"
choose Copy Source Markdown
read all lines
alt
[line-range metadata exists]
slice exact source line range
[direct text mapping succeeds]
map selected text to source span
[fuzzy line mapping succeeds]
anchor with first/last meaningful lines
[fallback]
use entire source file
setText(Clipboard + Selection)
optional reliability fallback
status message (exact/fuzzy/full fallback)
User
QWebEngineView
MdExploreWindow
Source .md file
System Clipboard
wl-copy/xclip/xsel

7. PDF Export Pipeline (Abstracted)

Detailed PDF mode branching (JS Mermaid grayscale path vs Rust Mermaid default-themed PDF cache path) is documented in `RENDER-PATHS.md`.



8. Document View/Tab Lifecycle


NoFile
Placeholder
"Select a markdown file"
select markdown file
root change / file removed
SingleView
ActiveView1
one logical view
switch files
root change / file removed
window close (persist effective root)
Add View
close tabs until one remains
MultiView
TabSet
view tabs visible
fixed width + pastel sequence
manual drag reordering
each tab stores
view_id, sequence, color_slot,
scroll_y, top_line, progress
Add View (max 8)
switch files
switch to another document
return document with 1-view session
window close (persist effective root)
switch to another document
return document with multi-view session
window close (persist effective root)
SessionSaved

Notes

- Diagrams are based on current code in `mdexplore.py` and `mdexplore.sh`.
- Render/caching branch internals are intentionally abstracted here and documented in `RENDER-PATHS.md` to keep a single authoritative deep map.
- Worker/threadpool usage is intentionally separated by concern:
  - render pool (preview HTML jobs),
  - PlantUML pool (diagram jobs),
  - PDF pool (post-processing/stamping).
- PlantUML rendering is non-blocking in UI flow: placeholders are rendered first, then patched in place as jobs finish.
- TODO (known issue): diagram zoom/pan restore (Mermaid and PlantUML) is not yet consistently reliable when leaving a document and returning in the same app run.