DICOM Data Explorer (Intro) — Software Design Document

> **Purpose**: Help users explore public DICOM datasets, inspect **Series metadata**, and download DICOM files to a local folder, **without writing code**.
> **Current data source**: **IDC (NCI Imaging Data Commons)** via the `idc_index_data` parquet index + public S3 objects.

—

Table of Contents

—

1. Scope

This document describes the architecture and design for a Reflex-based web app that provides:

- A **Welcome** screen with onboarding/tutorial.
- An **IDC Search** screen to filter and browse **Series**.

- A **Downloads** screen to manage a cart and download DICOM files from public S3.

—

2. Goals and Non-Goals

Goals

- Beginner-friendly exploration of public DICOM datasets.
- Simple filters and browsing of IDC series.
- Local download of DICOM files with progress feedback.

Non-Goals (current version)

- Authentication / user accounts.
- Persisting cart/history across browser refresh or server restart.
- Multi-source download beyond IDC (legacy TCIA modules are placeholders).

—

3. Tech Stack

- **Python 3.11**
- **Reflex** (web UI + backend state/events)
- **DuckDB** for querying IDC parquet index
- **idc_index_data** providing `IDC_INDEX_PARQUET_FILEPATH`
- **requests** for S3 listing and file downloads
- **Tailwind CSS** for styling

—

4. Repository Layout

```
dicom_data_explorer_intro-main/
├── dicom_data_explorer_intro/
│   ├── dicom_data_explorer_intro.py      # App entry: routes
│   ├── pages/
│   │   ├── welcome.py                    # Landing page + data source cards
│   │   ├── idc_search.py                 # Filters + results + metadata panel
│   │   ├── downloads.py                  # Cart + progress + history
│   │   ├── collections.py                # placeholder/legacy
│   │   └── search.py                     # placeholder/legacy
│   ├── components/
│   │   ├── layout.py                     # Sidebar + main + onboarding modal
│   │   ├── sidebar.py                    # Navigation + tutorial/help buttons
│   │   ├── onboarding.py                 # Tutorial modal content
```

```
│   │   └── tooltip.py                    # Helper tooltip component
│   ├── services/
│   │   ├── idc_service.py                # DuckDB queries over IDC parquet
│   │   └── tcia_service.py               # deprecated placeholder
│   └── states/
│       ├── ui_state.py                   # Onboarding UI state
│       ├── idc_state.py                  # Filters/search/sort/pagination
│       ├── download_state.py             # Cart/download/progress/history
│       └── tcia_state.py                 # deprecated placeholder
├── rxconfig.py
├── pyproject.toml / poetry.lock
├── reflex_rerun.sh
└── docs/images/                           # screenshots for README
```

—

5. Functional Requirements

FR-1 Welcome + Tutorial

- User can open the Welcome page (`/`) and start an interactive onboarding modal (tutorial).

FR-2 IDC Search and Browse

- User can select filters:
- Collection
- Modality
- Body Part
- User can run a search and view a results table.
- User can apply client-side:
- free-text search query
- min/max image count filters
- sort field and direction
- pagination (10 items per page)

FR-3 Series Details

- User can select a series and see detailed metadata for that series.

FR-4 Download Cart + Download

- User can add series to a cart.
- User can remove items / clear cart.

- User can start download for all cart items.
- UI shows:
- progress percent
- total files and downloaded files
- current series UID (in progress)

FR-5 Download History (session)

- After downloads, user sees a history list in the same session.

—

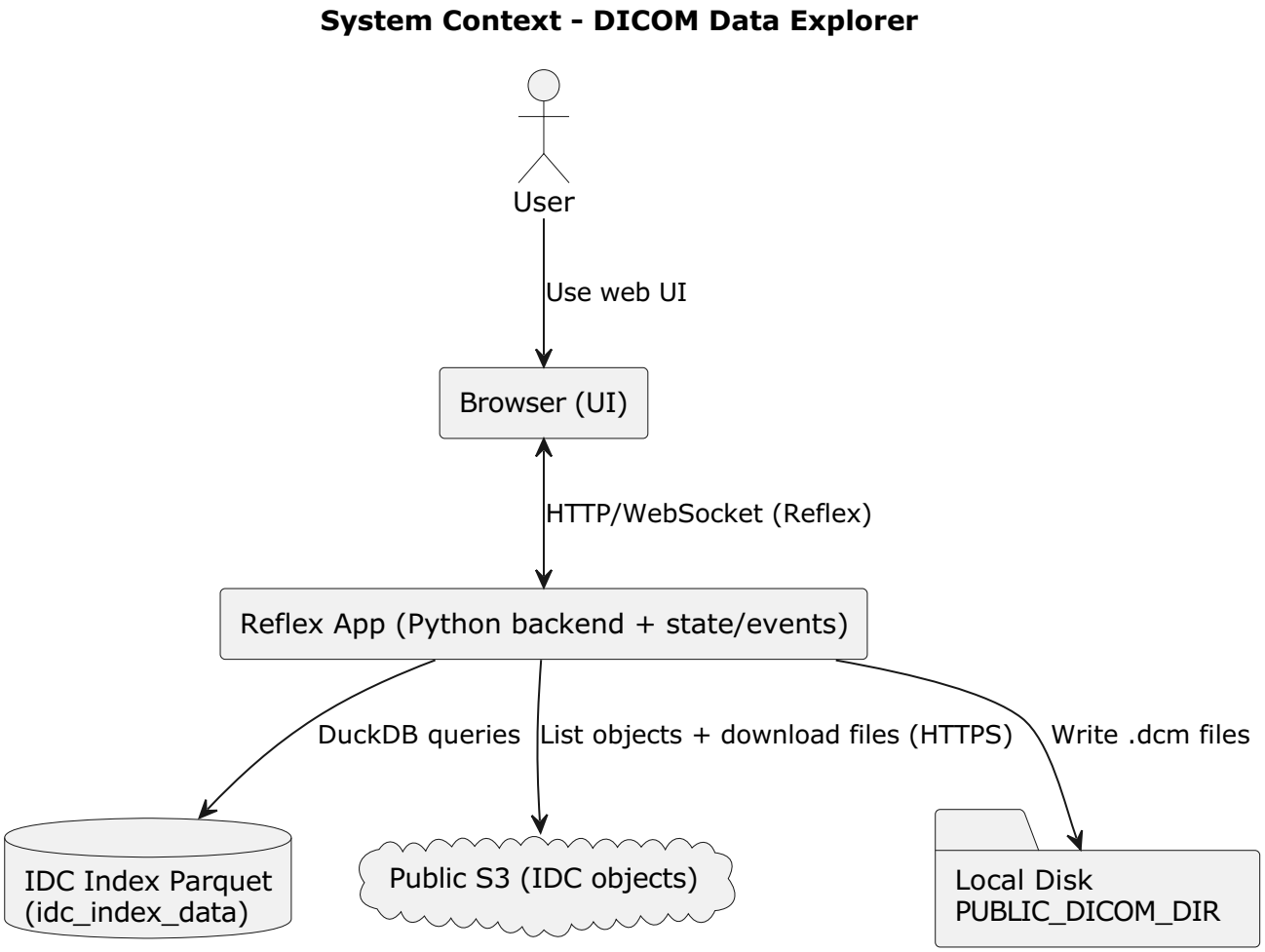
6. Non-Functional Requirements

- **Usability**: onboarding steps + tooltips for medical imaging terms.
- **Reliability**: download errors should not crash the entire app.
- **Maintainability**: clear separation: pages/components/states/services.
- **Portability**: configurable download folder via env var.

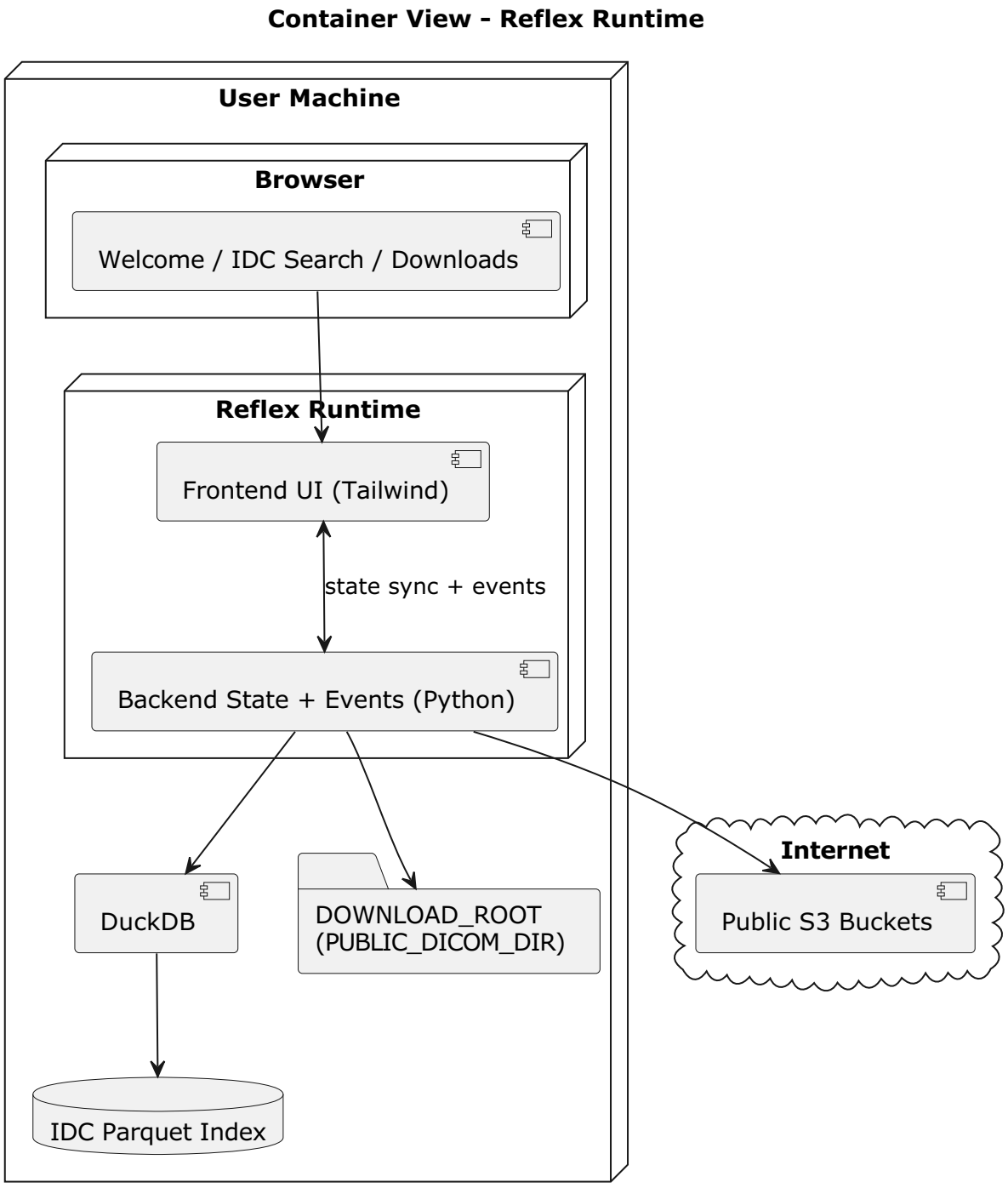
—

7. High-Level Architecture

7.1 System Context Diagram



7.2 Container View



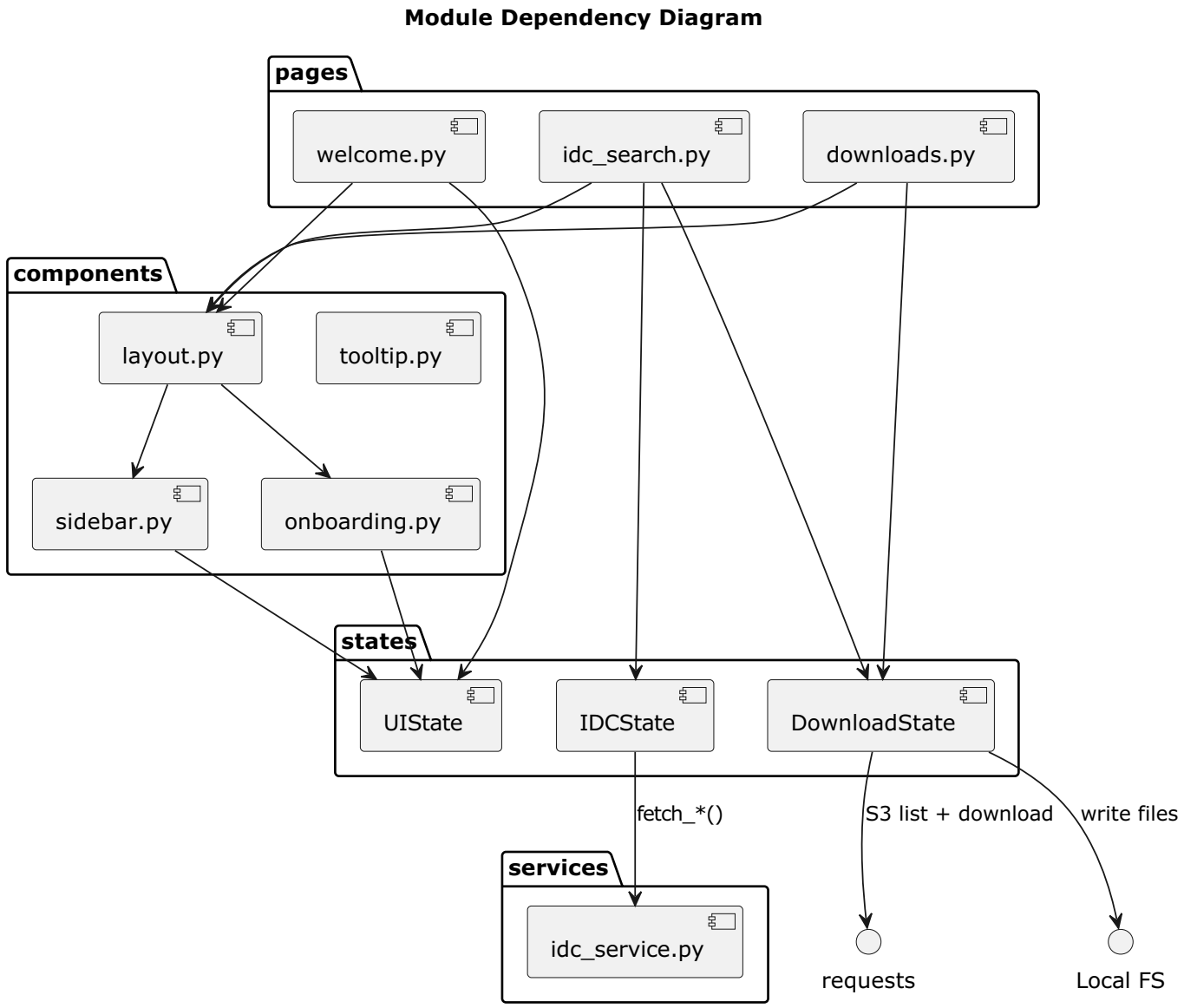
—

8. Module Design

8.1 Layering Principles

- **pages/**: page-level UI layout and user interactions.
- **components/**: reusable UI (layout, sidebar, onboarding, tooltip).
- **states/**: application logic/state (filters, pagination, downloads).
- **services/**: data-access logic (DuckDB queries for IDC).

8.2 Module Dependency Diagram



—

9. Data Model

9.1 IDC Series Record (dict)

From `idc_service.fetch_series()` (DuckDB -> DataFrame -> `to_dict(orient="records")`):

- `SeriesInstanceUID`
- `Collection` (from `collection_id`)
- `Modality`
- `BodyPartExamined`
- `SeriesDate`
- `SeriesDescription`
- `ImageCount` (from `instanceCount`)

- `series_size_MB`
- `series_aws_url`

9.2 Cart and History Items

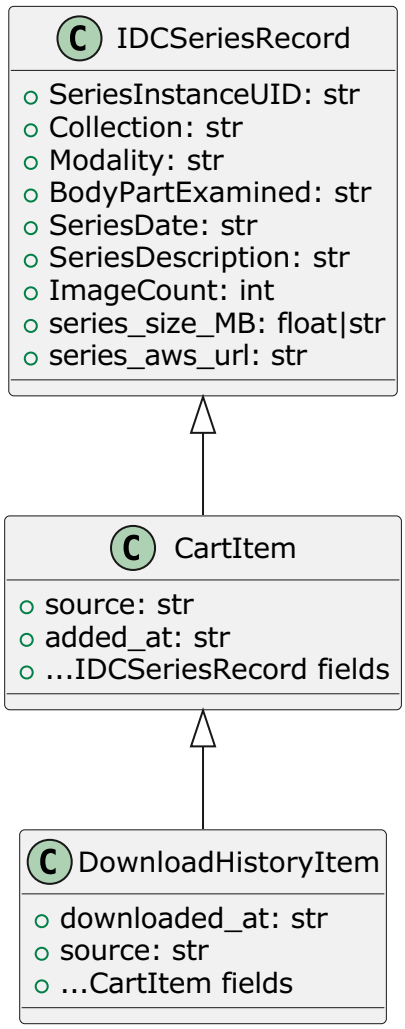
Cart items are series dict + additional fields:

- `source` (expected `"IDC"`)
- `added_at`

History items are cart dict +:

- `downloaded_at`

Data Model (Conceptual)



9.3 Local Output Paths

- Download root: `DOWNLOAD_ROOT = Path(os.getenv("PUBLIC_DICOM_DIR", "/Users/Shared/DICOM"))`
- Series folder:

- `DOWNLOAD_ROOT / sanitize(Collection) / sanitize(SeriesInstanceUID) / <relative s3 key path>`

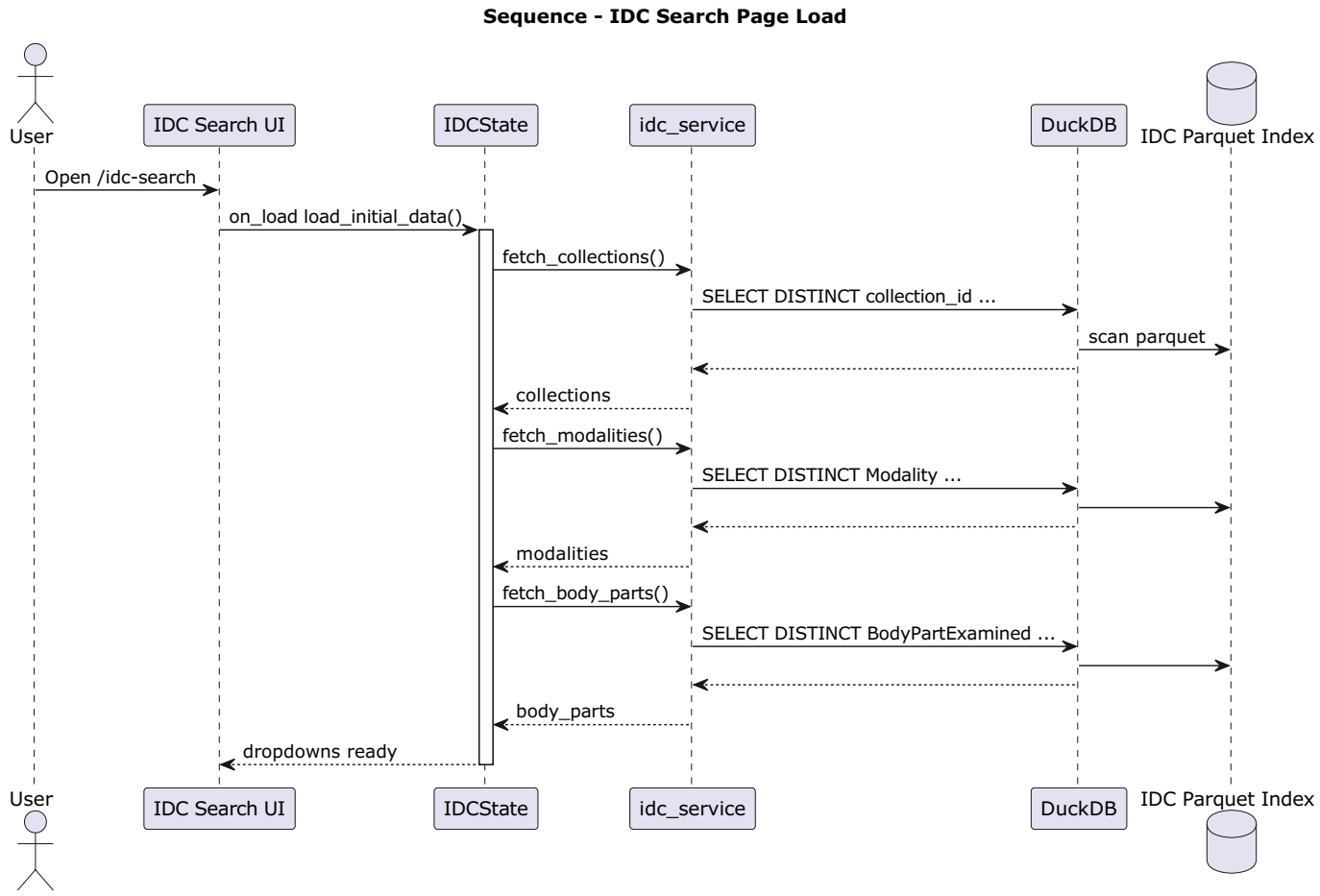
Sanitization (`_sanitize_segment`):

- empty -> `"unknown"`
- `/` and `\` replaced with `_`

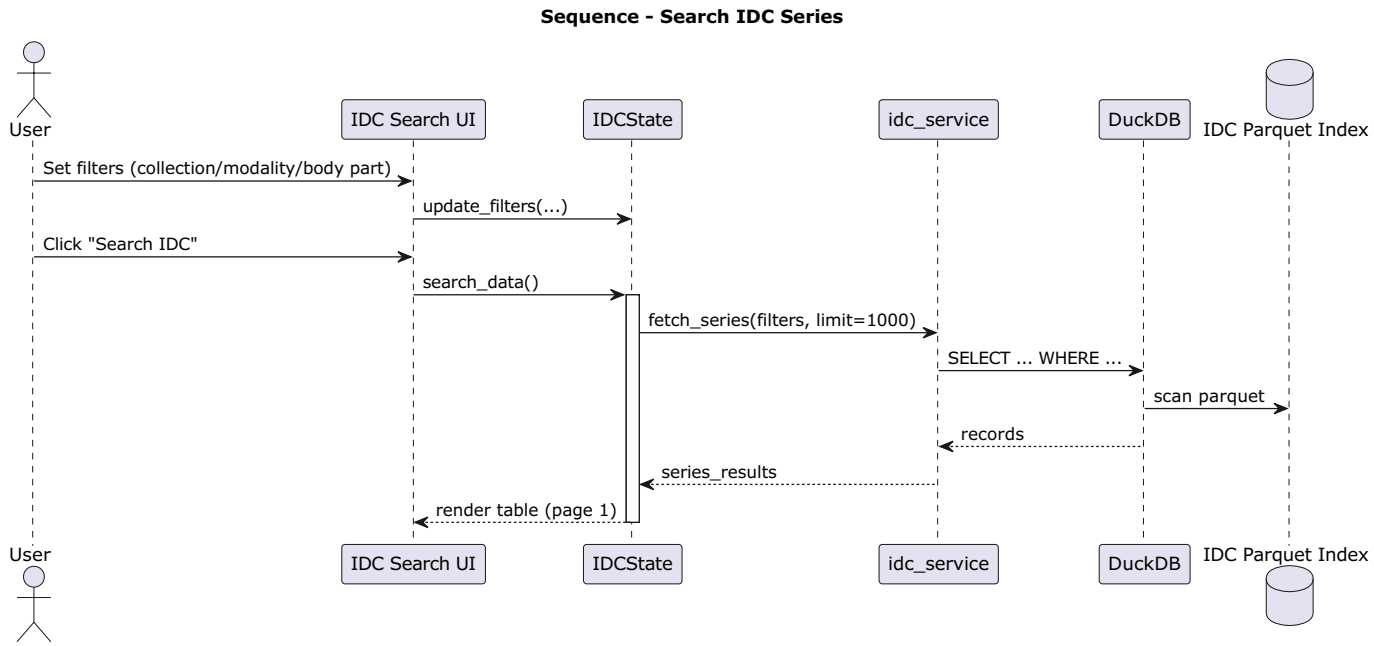
—

10. Key Flows (Sequence Diagrams)

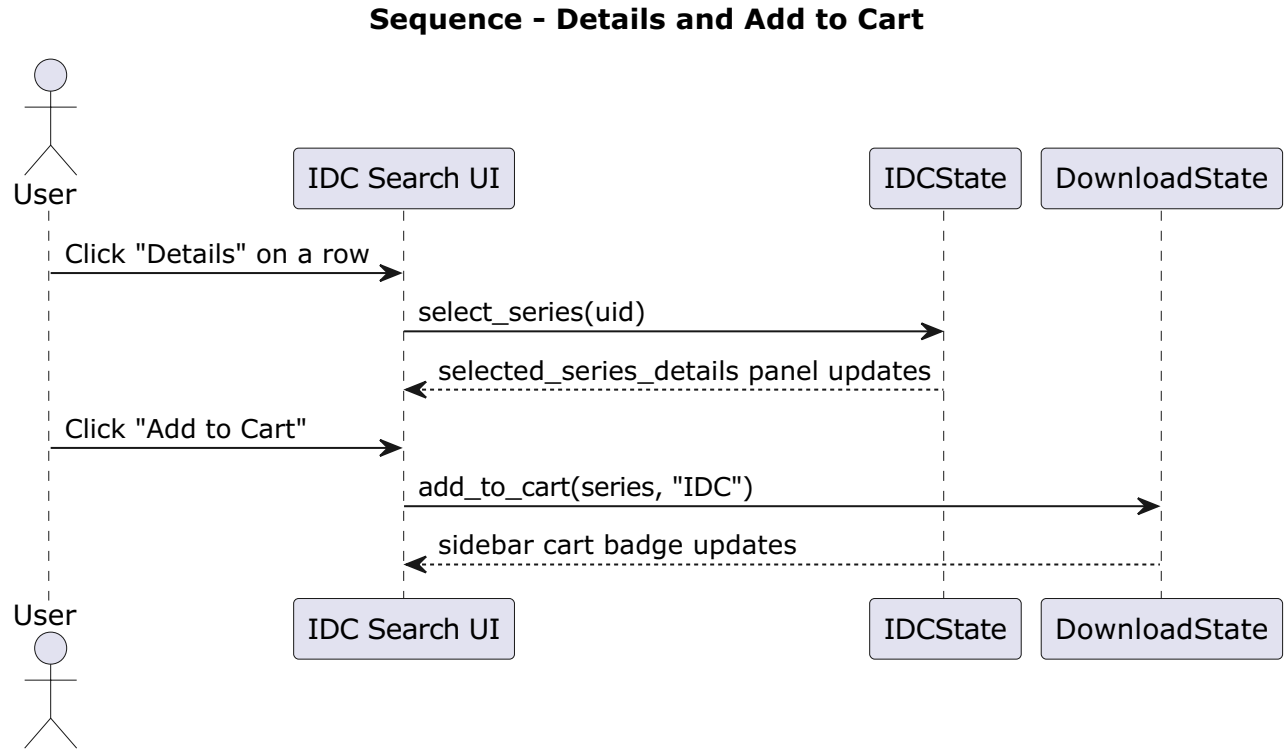
10.1 Page Load: `/idc-search` initializes dropdowns



10.2 Search Series



10.3 Select Details + Add to Cart



—

11. Download Pipeline Design

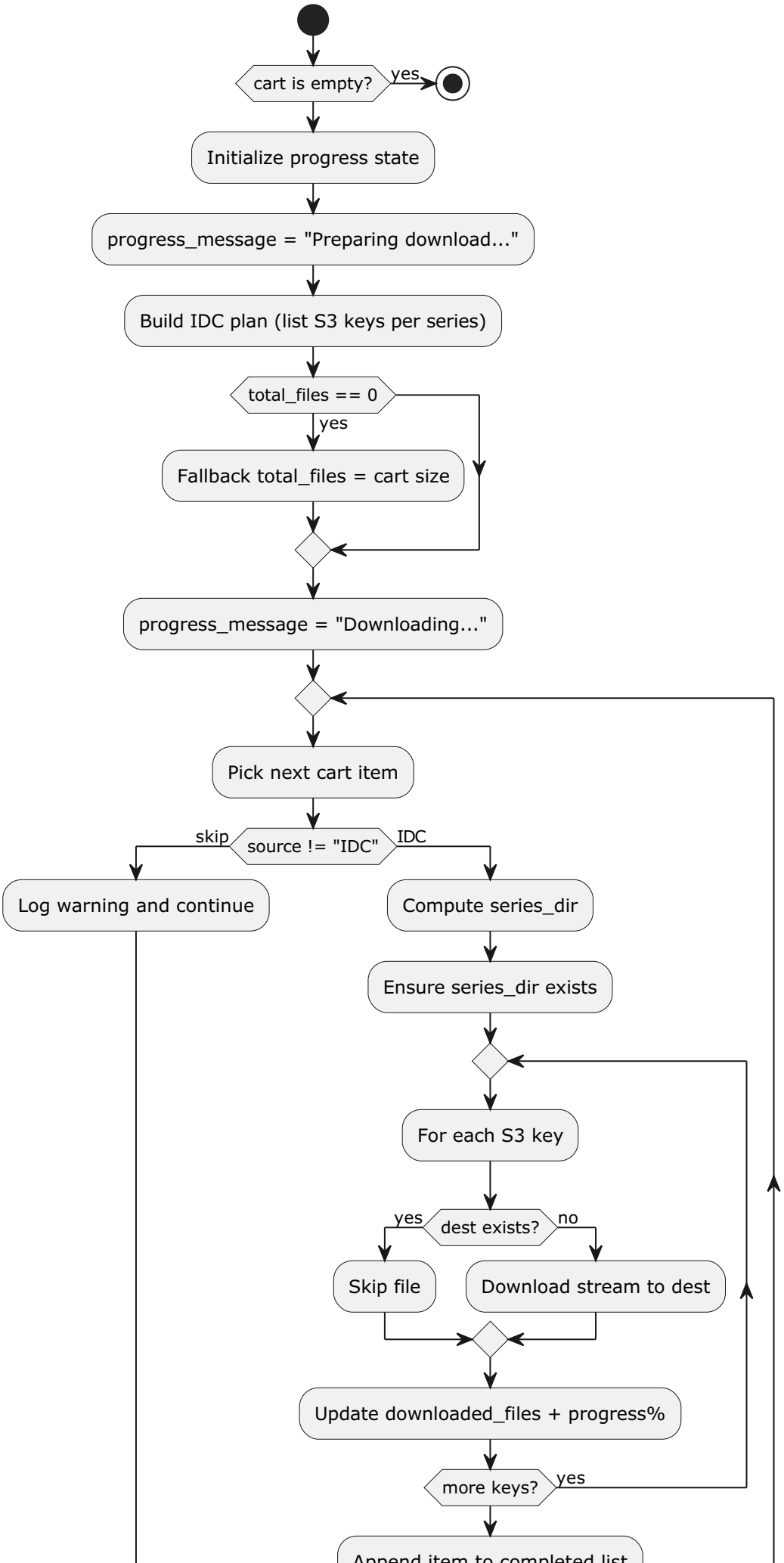
11.1 Design Summary

Download is implemented in `DownloadState.start_download()` as an **async event**:

1. Preparation:
   - Set flags: `is_downloading=true`, reset progress counters
   - Build an **IDC plan** by listing S3 keys per series (`_get_idc_keys`)
   - Accumulate `total_files`
1. Download:
   - For each IDC series in cart:
     - Determine output folder
     - For each S3 key:
     - download via streaming to local path (skip existing file)
     - update counters and progress percent
1. Finalization:
   - Add successful items to `download_history`
   - Clear cart and reset flags

11.2 Activity Diagram

Activity - Download All Series



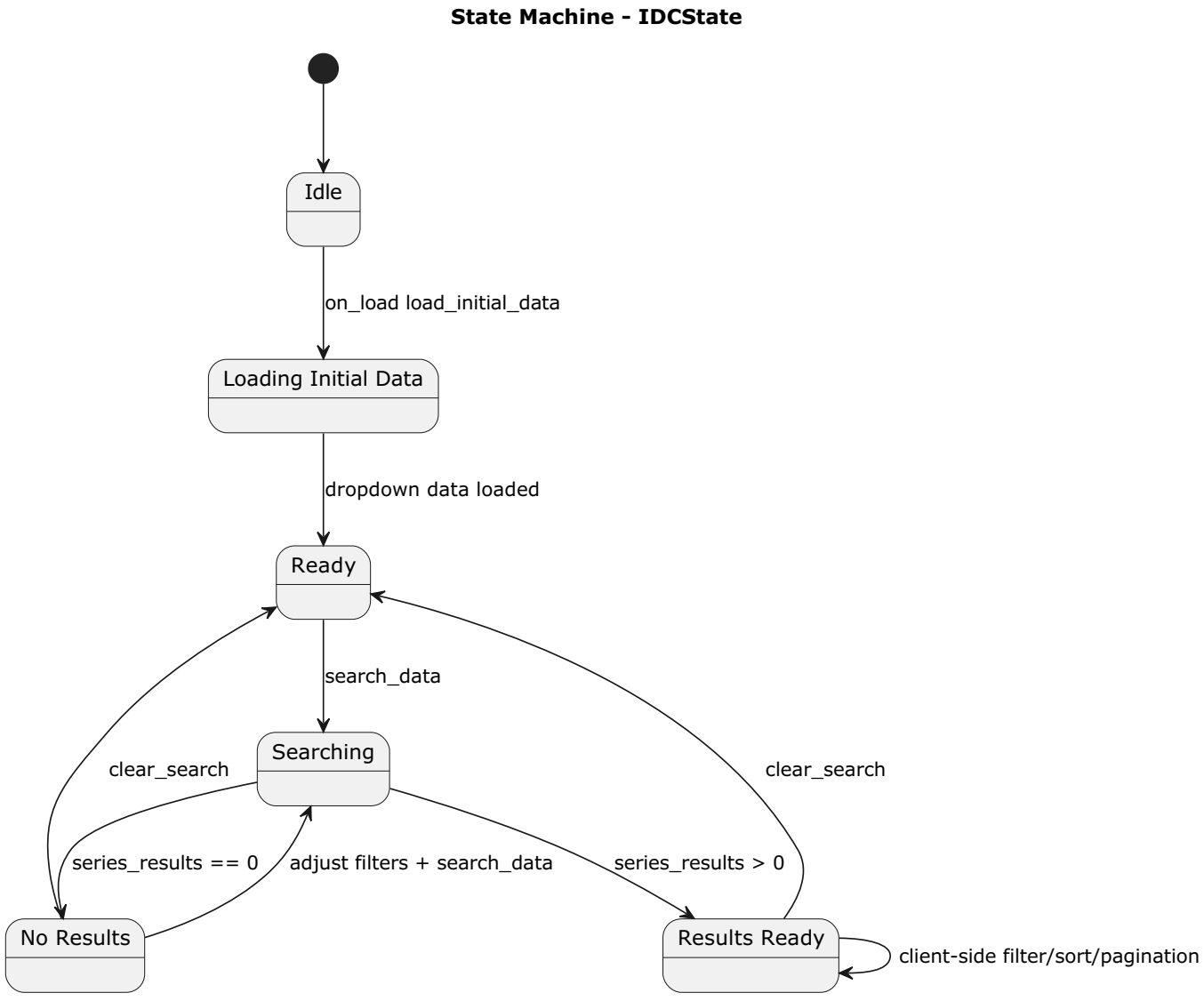
11.3 S3 URL Parsing + Listing

- `_parse_s3_url()` supports:
- `s3://bucket/prefix...`
- `https://bucket.s3.amazonaws.com/prefix...`
- `https://s3.amazonaws.com/bucket/prefix...`
- `_list_s3_objects()` uses **S3 ListObjectsV2** via XML response and continuation token.

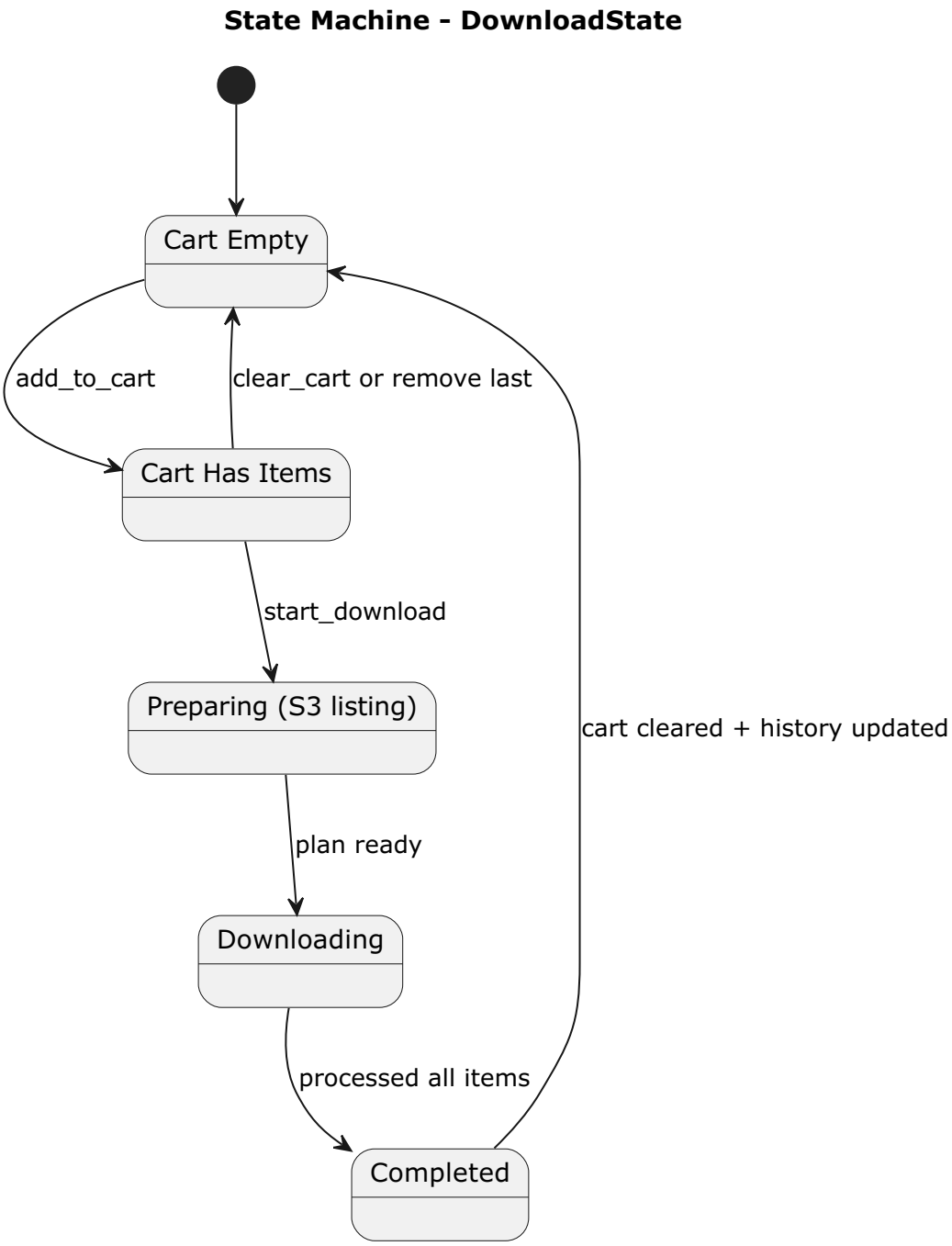
—

12. State Machines

12.1 IDCState



12.2 DownloadState



—

13. Error Handling and Observability

Current behavior

- Services and download pipeline wrap key operations in try/except and log exceptions.
- Failures for one item should not abort other items (best-effort behavior).

Recommended improvements

- Surface `IDCState.error_message` in UI (currently defined but not displayed).

- Track per-series download result (`success/failed`, error summary) in history.
- Add a cancel mechanism (user-initiated stop).

—

14. Security Considerations

Input safety

- DuckDB queries in `idc_service` build SQL with string interpolation for filters.
- Risk is reduced because filter values typically come from dropdown lists.
- Still recommended to enforce strict whitelisting or parameterization.

Filesystem safety

- `_sanitize_segment()` reduces path traversal risks by removing separators.
- Consider verifying that `DOWNLOAD_ROOT` is writable and not a sensitive location.

—

15. Performance and Scalability

Current

- `fetch_series(limit=1000)` returns up to 1000 records, then all filtering/sorting/pagination is client-side.
- Downloads are sequential per key; may be slow for large series.

Recommended

- Implement server-side pagination/sorting for large result sets.
- Add limited concurrency for downloads (e.g., N parallel files per series).
- Cache S3 listing results per `SeriesInstanceUID` during a session.

—

16. Configuration and Deployment

Run (Poetry)

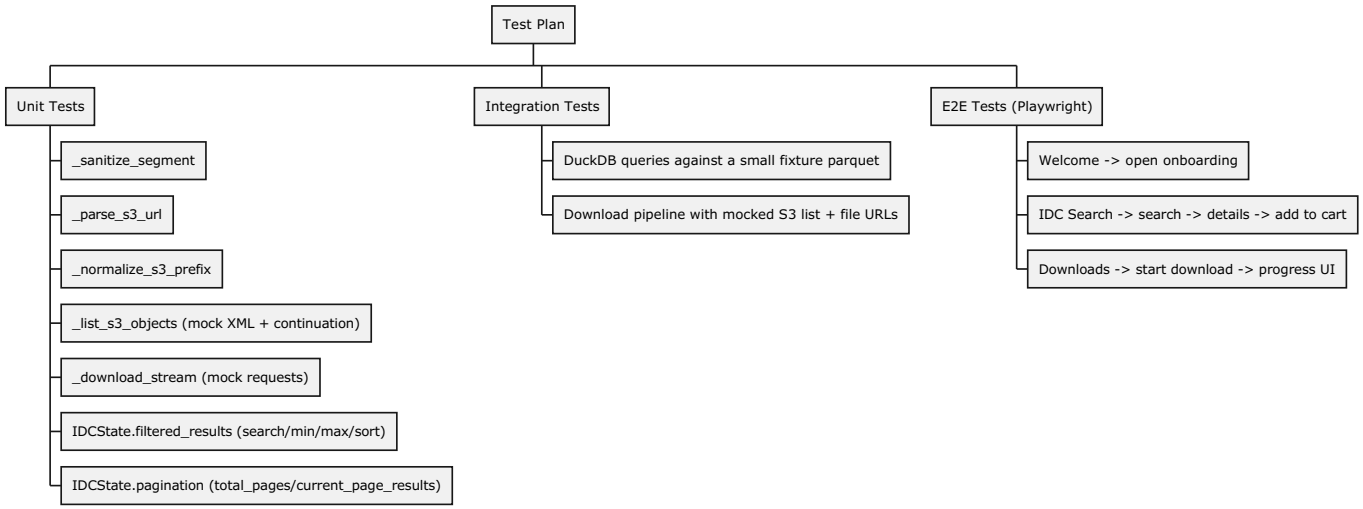
```
poetry env use python3.11
poetry install
poetry run ./reflex_rerun.sh
# http://localhost:3000
```

Environment Variables

- `PUBLIC_DICOM_DIR`
- default: `/Users/Shared/DICOM`
- used as download root folder

—

17. Testing Strategy



—

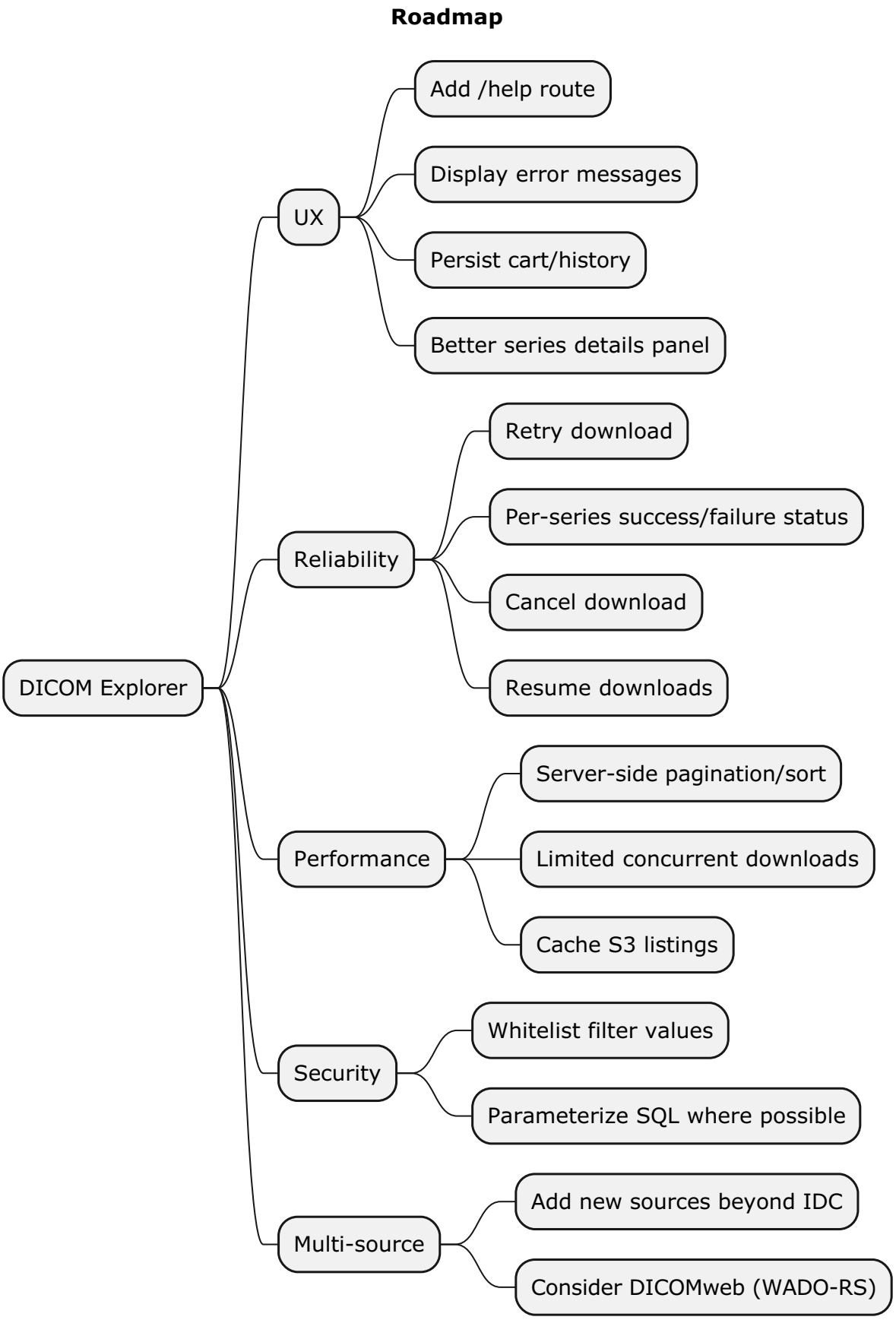
18. Known Gaps and Improvements

Known gaps (from code)

1. Sidebar contains a `Help & FAQ` link to `/help`, but **no route is registered** for `/help` (will 404).
2. `IDCState.error_message` is defined but not rendered.
3. Cart and history are in-memory state only (lost on reload/restart).

4. Only `"IDC"` is supported for downloading; other sources are skipped with a warning.
5. S3 XML listing can be slow for large prefixes and may be subject to throttling.

Improvement roadmap (high-level)



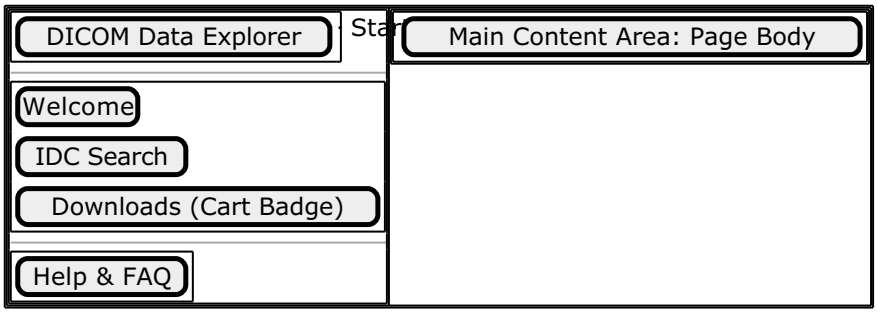
—-

—

19. UI Wireframes (PlantUML salt)

19.1 Global Layout (Sidebar + Content)

UI Wireframe - Global Layout (Sidebar + Content)



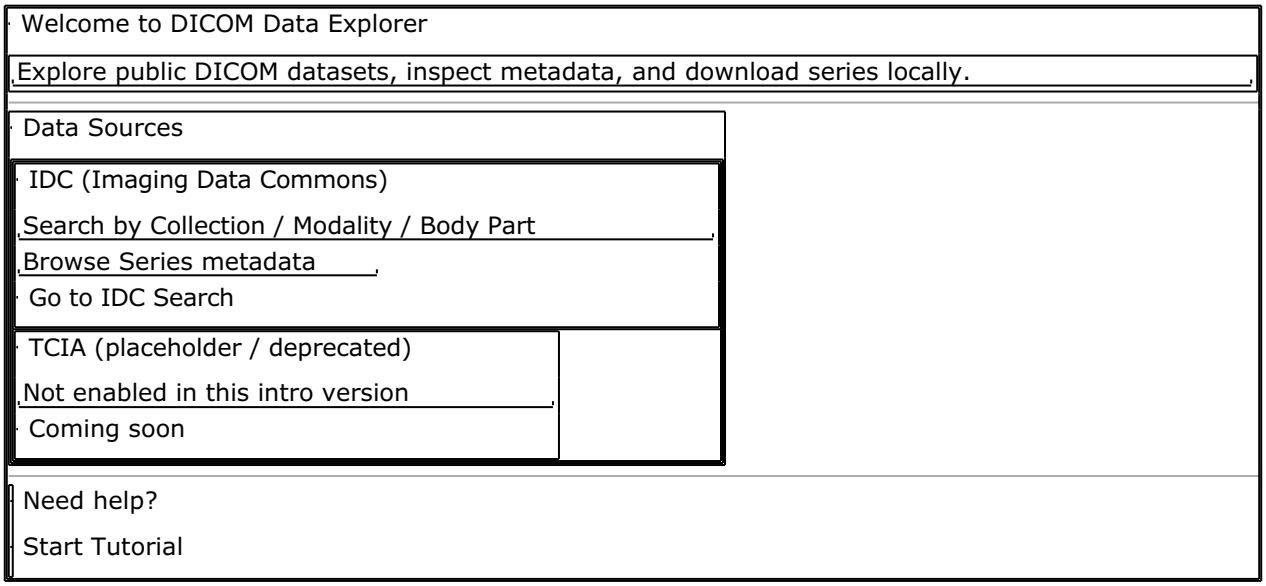
Notes:

- Left = sidebar navigation + tutorial button
- Right = active page content

—

19.2 Welcome Page Wireframe

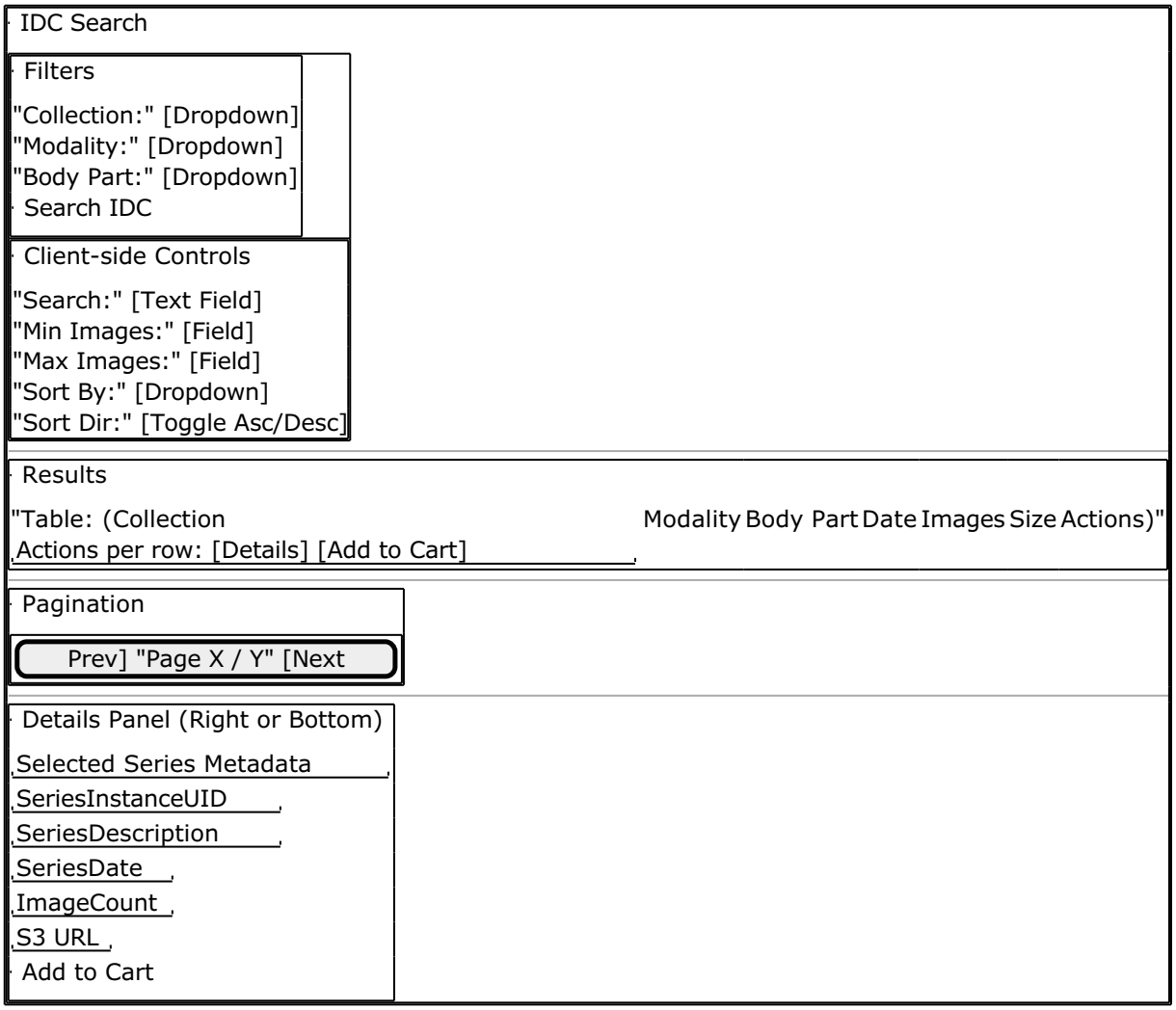
UI Wireframe - Welcome Page



—

19.3 IDC Search Page Wireframe (Filters + Table + Details Panel)

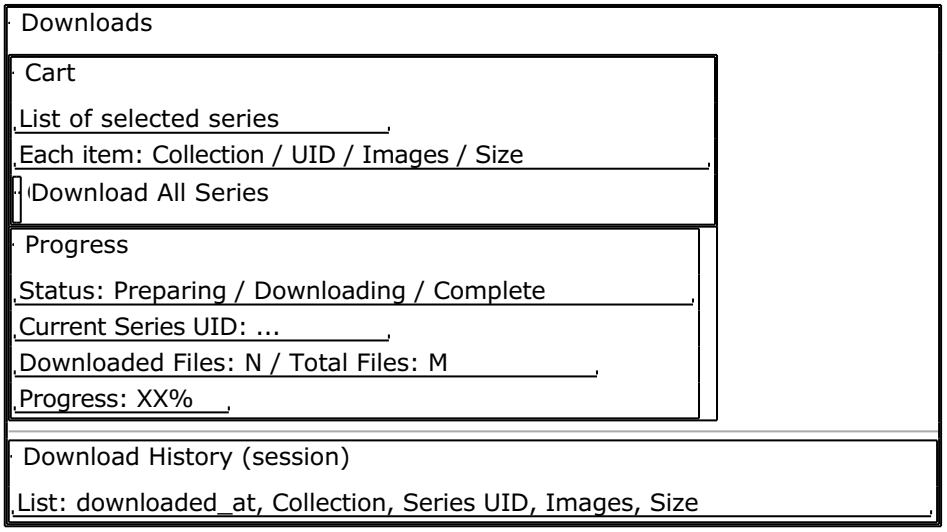
UI Wireframe - IDC Search Page



—

19.4 Downloads Page Wireframe (Cart + Progress + History)

UI Wireframe - Downloads Page

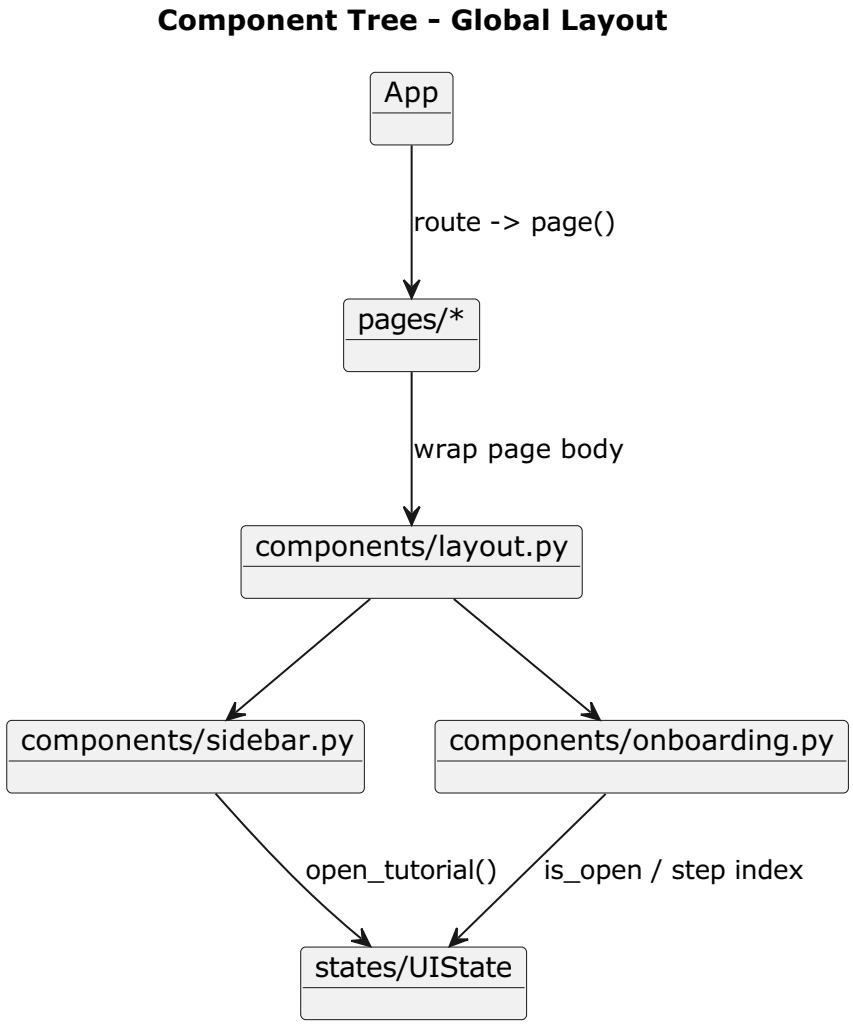


—

20. Page Component Trees (Welcome / IDC Search / Downloads)

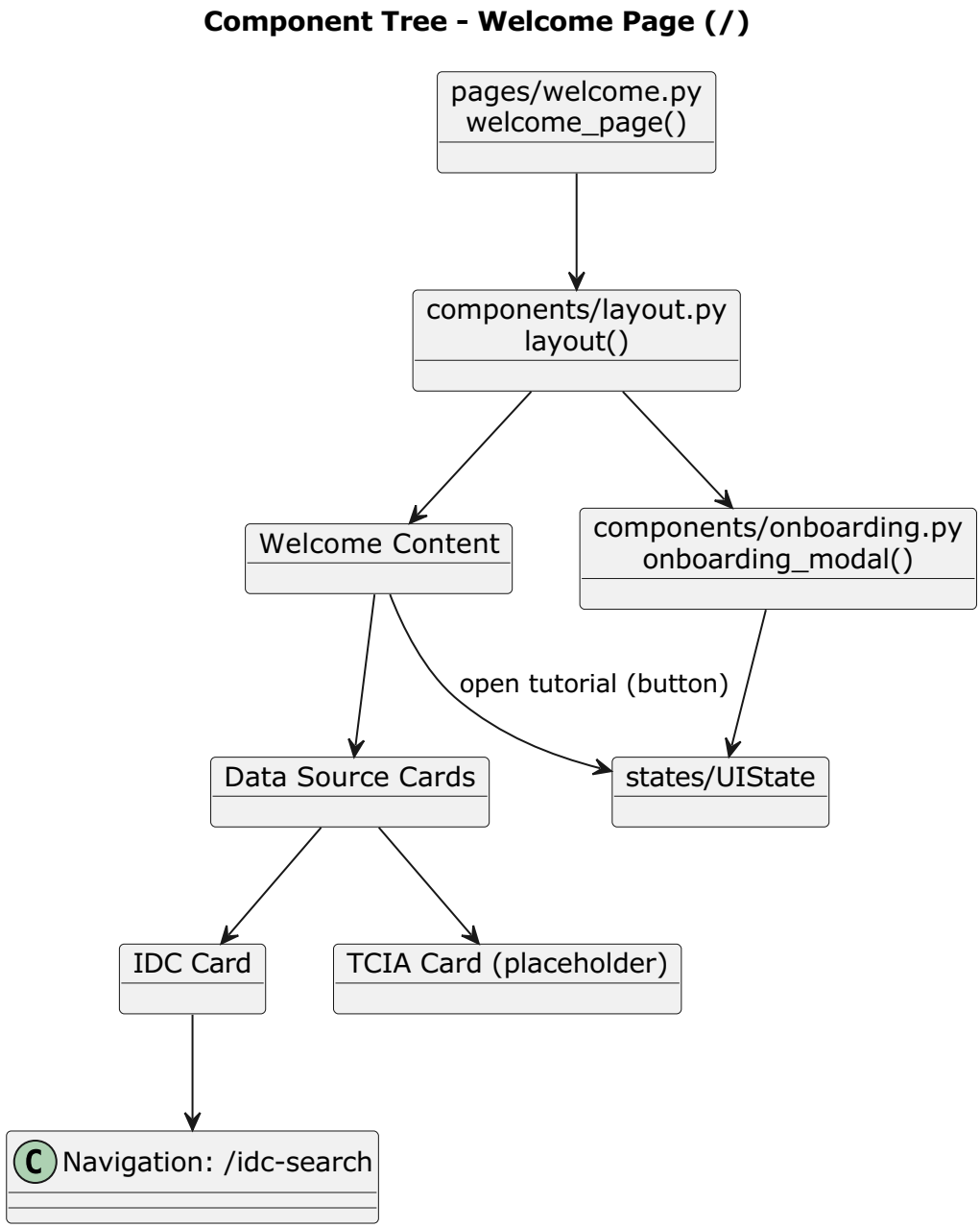
This section shows a **logical component tree** (how the UI composes layout + page elements). It is not strict React code, but a design-level view aligned with the code structure: `pages/`, `components/`, `states/`.

20.1 Global Component Tree (Shared Layout)



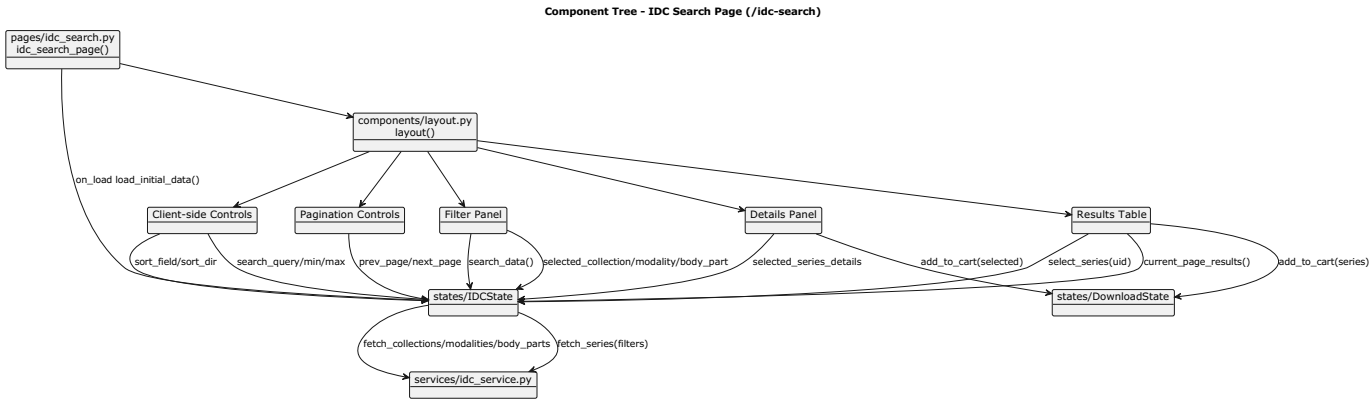
—

20.2 Welcome Page Component Tree



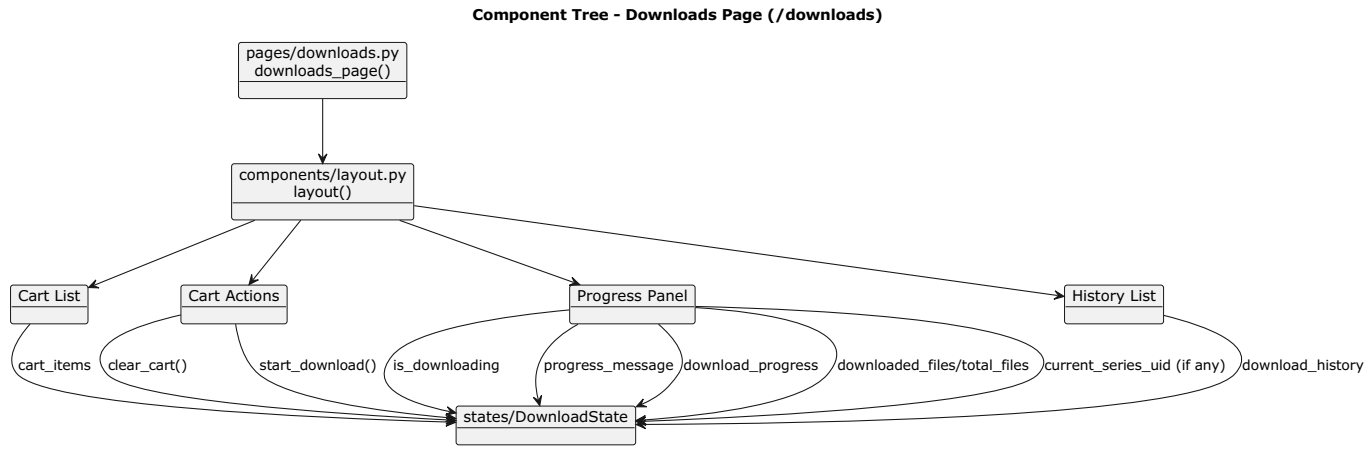
—

20.3 IDC Search Page Component Tree



—

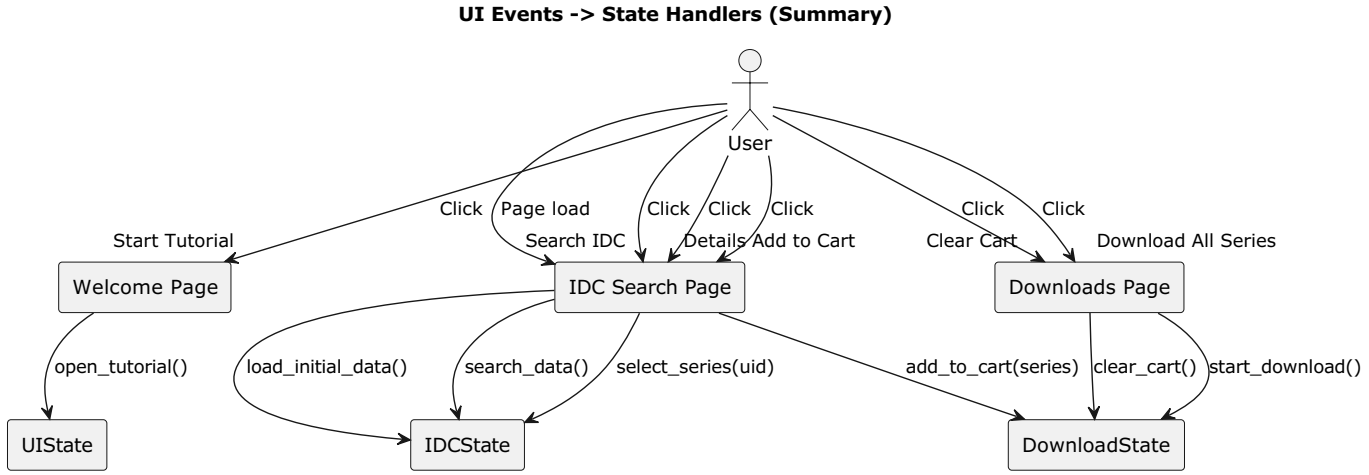
20.4 Downloads Page Component Tree



—

21. Optional: UI Interaction Map (Events -> State)

If you want an explicit mapping of “User action” to “State event handler”:



—