Dydra JSUI Summary

February 7, 2026

---

1. Overview

The JSUI is a complete single-page application replacement for the Dydra web interface, comprising two SPAs:

- **User App** (`index.html`): Account management, repository operations, SPARQL queries, data import/export
- **Admin App** (`admin.html`): System administration with account/repository management, invitation handling, and history monitoring

Both applications share core infrastructure (router, authentication, state management) and communicate with the SPOCQ backend via a RESTful `/system/` API.

---

2. Key Capabilities

2.1 Repository Management

- Create, edit, delete repositories
- Clear all triples with confirmation
- Export to multiple RDF formats (Turtle, N-Triples, N-Quads, RDF/XML, JSON-LD, TriG, CSV)
- Import with progress tracking, byte counts, and abort capability
- Asynchronous import option (`AcceptAsynchronous: notify` header)
- Collaborator management with read/write permissions

2.2 SPARQL Interface

- YASQE-based editor with syntax highlighting
- 12+ result format options via content negotiation
- Save/Save As/Clear query operations
- Saved views with execute and delete actions
- Query history with signature links to query text
- Multiple query tabs within a single editor instance
- **Standalone editor windows** — Drag query editor tabs outside the application window to open them as independent windows
- **State preservation** — Query text, tabs, results, and authentication state are preserved when opening standalone windows
- **View management** — Views can be opened in current pane, new pane, or new window with results table

2.3 Administration

- Account listing with bulk selection, details modal, create/delete
- Repository management filtered by selected accounts
- Invitation management (list, send, delete, create)
- **Query history** with table and graph views:
  - Table view with sortable columns (timestamp, account, repository, signature, runtime)
  - Graph view plotting runtime (in seconds) against timestamp
  - Clickable graph datapoints showing query text with timestamp, account, and repository
- **Transaction history** (system-wide) with sortable table
- **Import history** (system-wide) with table and graph views:
  - Table view with pagination support
  - Graph view plotting quad count against timestamp using logarithmic scale
- **Tabbed sub-panes** — History views support switching between table and graph views
- Role-based UI: non-admin users see data but not destructive controls

2.4 User Experience

- Multi-account/multi-host authentication support
- Pane/tab system for simultaneous views
- Editable location bar showing current route
- Cross-app navigation between user and admin SPAs
- Field-level change tracking with dirty-state detection
- Lazy-loaded admin tabs for reduced initial load
- **Drag-and-drop interface** — Query editor tabs can be dragged to create standalone windows
- **Session storage** — State transfer between windows using `sessionStorage` and URL hash parameters

2.5 Data Visualization

- **SVG graph rendering** for time-series data visualization
- **Automatic scaling** — Horizontal scale adjusts to data range; vertical scale rounds to powers of ten
- **Logarithmic scaling** — Support for log scale Y-axis (used for import history)
- **Dynamic time grids** — Grid markers adapt to data range (minutes, hours, days, or 6-month intervals)
- **Axis labeling** — Clear labels for both axes (e.g., “Run Time (s)”, “Quads”, “Time”)
- **Clickable datapoints** — Graph points can be interactive (e.g., query history shows query details)

- **Visual styling** — Small dots (1px radius) and lines (1px width) for clean appearance

---

3. Advantages Over Rails Implementation

1. **Richer SPARQL editor** — YASQE-based with syntax highlighting, inline execution, multiple result formats
2. **Repository analytics** — Events, resources, statistics, series, revisions tabs
3. **Multi-account support** — Simultaneous authentication against multiple Dydra instances
4. **Field-level change tracking** — Replication system tracks edits with dirty-state detection
5. **Pane/tab system** — View multiple accounts and repositories simultaneously
6. **Location bar** — Editable address bar showing and accepting route input
7. **Static deployment** — No server dependency for UI; can be served from any web server or CDN
8. **Broader import formats** — Client-side detection for .ttl, .rdf, .xml, .nt, .nq, .trig, .jsonld, .json, .csv, .hdt
9. **RDF export with format selection** — Multiple serialization formats via content negotiation
10. **Repository clear** — One-click removal of all triples with confirmation
11. **Transaction history** — System-wide transaction monitoring (not available in Rails)
12. **Admin SPA with filtered views** — Account checkbox filtering cascades to repositories and query history
13. **Sortable admin tables** — Column-header click sorting with directional indicators
14. **Lazy-loaded admin tabs** — Content fetched on demand
15. **Cross-app navigation** — Bidirectional links between user and admin SPAs
16. **SVG branding** — Vector logos with PNG fallbacks
17. **Data visualization** — Interactive SVG graphs for query and import history (not available in Rails)
18. **Standalone editor windows** — Drag-and-drop interface for opening editors in separate windows
19. **Tabbed sub-panes** — Multiple view modes (table/graph) within admin dashboard panes

---

4. Data Models

4.1 Account

Field	Description
id	Unique identifier
name	Account name (slug)
email	Contact email
fullname	Display name
phone, skype, jabber	Contact fields
workinfo	Work information
balance	Account balance
authentication_token	API access token

4.2 Repository

Field	Description
id	Unique identifier
account_id	Owning account
name	Repository name (slug)
summary	Short description
description	Full description
homepage	External URL
license	License identifier
quad_count	Number of quads stored
disk_size	Storage used

4.3 Query/View

Field	Description
id	Unique identifier
repository_id	Parent repository
name	Query name (slug)
query	SPARQL query text
running	Execution status flag

4.4 Invitation

Field	Description
id	Unique identifier
email	Invitee email
invite_code	Generated code
http_referrer	Referral source
account_name	Linked account (if accepted)

4.5 Session

Field	Description
accountName	Logged-in account
login()	Start session
logout()	End session
isLoggedIn()	Check status

---

5. API Endpoints

5.1 Authentication

Endpoint	Method	Purpose
`/system/accounts/{acct}/configuration`	POST	Authenticate, retrieve token
`/system/users/{acct}/configuration`	GET	Check admin privileges

5.2 Accounts

Endpoint	Method	Purpose
`/system/accounts`	GET	List all accounts
`/system/accounts`	POST	Create account
`/system/accounts/{acct}`	DELETE	Delete account
`/system/accounts/{acct}/configuration`	GET/POST	Read/update settings
`/system/accounts/{acct}/password-reset`	POST	Password reset
`/system/accounts/{acct}/token-reset`	POST	Reset auth token

5.3 Repositories

Endpoint	Method	Purpose
`/system/accounts/{acct}/repositories`	GET	List repositories
`/system/accounts/{acct}/repositories`	POST	Create repository
`/system/accounts/{acct}/repositories/{repo}`	DELETE	Delete repository
`/system/accounts/{acct}/repositories/{repo}/configuration`	GET/POST	Settings
`/system/accounts/{acct}/repositories/{repo}/collaboration`	GET/POST	Collaboration
`/system/accounts/{acct}/repositories/{repo}/history`	GET	Events
`/system/accounts/{acct}/repositories/{repo}/storage`	GET	Disk usage
`/system/accounts/{acct}/repositories/{repo}/service_statistics`	GET	Query stats
`/system/accounts/{acct}/repositories/{repo}/service_history`	GET	Query history

Endpoint	Method	Purpose
`/system/accounts/{acct}/repositories/{repo}/revisions`	GET	Revisions
`/system/accounts/{acct}/repositories/{repo}/views/{view}`	GET/PUT/DELETE	View CRUD

5.4 History

Endpoint	Method	Purpose
`/system/service_history/transactions`	GET	Transaction history
`/system/service_history/imports`	GET	Import history
`/system/service_history/queries/{acct}/{sig}`	GET	Query text

5.5 Invitations

Endpoint	Method	Purpose
`/invitations`	GET	List invitations
`/invitations`	POST	Create/send invitation
`/invitations/{email}`	DELETE	Delete invitation

5.6 Graph Store Protocol

Endpoint	Method	Purpose
`/{acct}/{repo}`	GET	Export (content-negotiated)
`/{acct}/{repo}`	POST	Import (merge)
`/{acct}/{repo}`	PUT	Replace all
`/{acct}/{repo}`	DELETE	Clear
`/{acct}/{repo}/sparql`	POST	Execute SPARQL

---

6. File Structure

```
jsui/
  index.html                   # User SPA entry
  admin.html                   # Admin SPA entry
  app.js                       # User app bootstrap
  admin-app.js                 # Admin app bootstrap
  router.js                    # Shared client-side router
  doc/
     requirements-20260207.md  # Current requirements
     summary-20260207.md       # This document
```

```
    summary.md                # Previous summary (2026-02-05)
    analysis-20260204.md      # Comparative analysis
js/
    sparql-editor.js          # SPARQL editor (2,345+ lines)
    yasqe-wrapper.js          # YASQE wrapper
lib/
    app_state.js              # Global state
    auth.js                   # Authentication
    auth_store.js             # Token storage
    config.js                 # Configuration
    models/                   # Data models
    persistence/              # Storage adapters
    replication/              # Change tracking
ui/
    app.js                    # User App controller
    routes.js                 # User routes (30+)
    utils.js                  # Shared utilities
    components/               # Layout, Header, Footer, etc.
    pages/
        base_page.js         # Abstract base
        index.js             # User pages (~5,460 lines)
    admin/
        app.js                # Admin App controller
        layout.js             # Admin layout
        routes.js             # Admin routes (10)
        pages.js              # Admin pages (~2,160 lines)
stylesheets/
    style.css                 # Base styles
    jsui-overrides.css        # Custom overrides
images/
    Dydra Logo-user.svg       # User logo
    Dydra Logo-admin.svg      # Admin logo
    trash.svg                 # Delete icon
    edit.svg                  # Edit icon
    link.svg                  # External link icon
    logs.svg                  # Table/logs icon
    chart-line.svg            # Graph/chart icon
fonts/, webfonts/              # Font assets
```

---

7. Technology Stack

- **Framework**: Vanilla ES6 modules, no build tooling
- **Editor**: YASQE (Yet Another SPARQL Query Editor)
- **HTTP**: Fetch API, XMLHttpRequest (for upload progress)

- **Storage**: LocalStorage for session persistence, SessionStorage for window state transfer
- **Styling**: Static CSS (no preprocessor)
- **Icons**: SVG with PNG fallbacks
- **Visualization**: Custom SVG graph rendering
- **Drag and Drop**: HTML5 Drag and Drop API

---

8. New Features Since February 5, 2026

8.1 Standalone Editor Windows

- **Drag-and-drop interface**: Query editor tabs can be dragged outside the application window
- **State preservation**: Query text, query tabs, results, and authentication are preserved when opening standalone windows
- **Session management**: Uses `sessionStorage` and URL hash parameters to transfer state between windows
- **New route**: `/standalone-editor` page for rendering standalone editor instances
- **Editor API enhancements**: Added `getQueryTabs()`, `getCurrentTabId()`, and `restoreQueryTabs()` methods

8.2 Data Visualization

- **SVG graph rendering**: Custom graph component for time-series data
- **Query history graphs**: Runtime (seconds) vs timestamp with clickable datapoints
- **Import history graphs**: Quad count vs timestamp with logarithmic scale
- **Dynamic time grids**: Adaptive grid markers based on data range (minutes, hours, days, 6-month intervals)
- **Automatic scaling**: Horizontal and vertical axes adjust to data automatically
- **Visual styling**: Clean appearance with 1px dots and lines

8.3 Enhanced View Management

- **Multiple view modes**: Views can be opened in current pane, new pane, or new window
- **View results window**: New window displays view results in table format with multiple output formats
- **Explicit event handlers**: Improved event handling for view operations (edit, pane, window, delete)
- **Editor instance tracking**: `editorInstances` Map tracks active editor instances for state transfer

8.4 Admin Dashboard Enhancements

- **Tabbed sub-panes**: History views support switching between table and graph views
- **Icon-based navigation**: Table view uses `logs.svg`, graph view uses `chart-line.svg`
- **Graph integration**: Query and import history panes include both table and graph views
- **Interactive graphs**: Query history graph datapoints show query details on click

8.5 Code Organization

- **StandaloneEditorPage**: New page class for standalone editor windows
- **renderGraph()**: New method in AdminDashboardPage for SVG graph generation
- **Enhanced editor API**: SPARQL editor now supports state serialization and restoration
- **Improved event handling**: Explicit event handlers with capture phase to prevent router interference

---

9. Component Architecture

9.1 User App Components

- **App** (`ui/app.js`): Main application controller with tab management and drag-and-drop support
- **Pages** (`ui/pages/index.js`): 20+ page classes including new `StandaloneEditorPage`
- **Router** (`router.js`): Client-side routing with History API support
- **State** (`lib/app_state.js`): Global application state management

9.2 Admin App Components

- **AdminDashboardPage** (`ui/admin/pages.js`): Dashboard with lazy-loaded panes and graph rendering
- **Graph rendering**: Custom SVG generation with automatic scaling and grid generation
- **History panes**: Query, transaction, and import history with table and graph views

9.3 SPARQL Editor

- **createSparqlEditor** (`js/sparql-editor.js`): Unified editor component
- **State management**: Query tabs, results, and editor state

- **API methods**: `getQueryTabs()`, `getCurrentTabId()`, `restoreQueryTabs()`
- **Multiple modes**: Standalone, integrated, and window-based

---

10. Implementation Details

10.1 Drag-and-Drop State Transfer

1. **Drag start**: Editor state (query text, tabs, results) stored in `sessionStorage` with unique session ID
2. **Drag end**: If dropped outside window, new window opened with session ID and token in URL hash
3. **Window initialization**: StandaloneEditorPage reads session ID, retrieves state from `sessionStorage`, and restores editor
4. **Cleanup**: Session storage cleaned up after state transfer

10.2 Graph Rendering

1. **Data extraction**: Time-series data extracted from history entries
2. **Scale calculation**: Horizontal scale based on time range; vertical scale rounded to powers of ten
3. **Grid generation**: Time grid markers generated based on data range (minutes/hours/days/6-month intervals)
4. **SVG generation**: Custom SVG paths, circles, and text elements for graph visualization
5. **Interactivity**: Click handlers attached to datapoints for additional information display

10.3 Tabbed Sub-Panes

1. **Tab structure**: Sub-tabs rendered within admin dashboard panes
2. **State management**: Active sub-tab tracked in pane dataset
3. **Content switching**: Table and graph content toggled based on active tab
4. **Icon integration**: Visual indicators for table vs graph views

---

11. Future Enhancements (Out of Scope)

- Real-time collaboration features
- Advanced query optimization suggestions
- Data visualization beyond basic table display
- Mobile native applications
- Offline mode with local data caching
- Additional graph types (bar charts, pie charts, etc.)
- Graph export functionality (PNG, SVG download)