JSUI Improvement Plan

Reference Implementations

- **Rails**: `/Users/Shared/Development/Source/Library/org/datagraph/dydra-rails-20161201`
- **SPOCQ**: `/Users/Shared/Development/Source/Library/org/datagraph/spocq` — `src/spocq-server/resource.lisp` (endpoint patterns), `src/spocq-server/admin/` (admin handlers)
- **JSUI**: `/Users/Shared/Development/Source/Library/org/datagraph/jsui`

A. Architecture: Local Routes vs. Remote Endpoints

The current documentation conflates client-side SPA routes with server API endpoints. These are distinct concerns:

- **Local routes** control client-side presentation. They are path patterns matched by the SPA router (`router.js`) to select a page class for rendering. They exist only in the browser and do not generate server requests by themselves.
- **Remote endpoints** are HTTP URLs served by the SPOCQ backend. Page classes issue `fetch()` calls to these endpoints to retrieve or modify data.

User App Local Routes → Remote Endpoints

Local Route	Page Class	Remote Endpoint(s)
`/login`	LoginPage	`POST /system/accounts/{acct}/configuration` (auth)
`/`	HomePage	(none — static content)
`/signup`	SignupPage	`POST /system/accounts`
`/reset_password`	ResetPasswordPage	*Currently: Rails `GET /reset_password`*
`/confirmations/new`	ConfirmationsPage	*Currently: Rails `GET /confirmation/new`*
`/unlocks/new`	UnlocksPage	*Currently: Rails `GET /unlock/new`*
`/invite`	InvitationsNewPage	`POST /invitations`
`/invite/success`	InvitationsSuccessPage	(none)
`/invitations`	InvitationsIndexPage	`GET /invitations`
`/account/:acct`	AccountShowPage	`GET /system/accounts/{acct}/configuration`
`/account/:acct/edit`	AccountEditPage	`GET/POST /system/accounts/{acct}/configuration`

Local Route	Page Class	Remote Endpoint(s)
`/account/:acct/auth_token`	AccountAuthTokenPage	*Currently: Rails `GET /:acct/auth_token`*
`/account/:acct/repositories`	RepositoriesIndexPage	`GET /system/accounts/{acct}/repositories`
`/account/:acct/repositories/:repo`	RepositoryShowPage	`GET /system/accounts/{acct}/repositories/{repo}/configura` `/storage`, `/history`, `/service_statistics`, `/service_history`, `/revisions`
`/account/:acct/repositories/:repo/edit`	RepositoryEditPage	`GET/POST .../configuration`, `GET/POST .../collaboration`
`/account/:acct/repositories/:repo/import`	RepositoryImportPage	`POST /{acct}/{repo}/service` (graph store)
`/account/:acct/repositories/:repo/query`	SparqlPage	`POST /{acct}/{repo}/sparql`
`/account/:acct/repositories/:repo/views/:view`	QueryShowPage	`GET /system/accounts/{acct}/repositories/{repo}/views/{vi` `POST /{acct}/{repo}/sparql`
`/account/:acct/repositories/:repo/query_logs`	TemplatePage	*Stub — see section F*
`/account/:acct/repositories/:repo/status`	TemplatePage	*Stub — see section F*
`/account/:acct/repositories/:repo/size`	TemplatePage	*Stub — see section F*
`/account/:acct/repositories/:repo/meta`	TemplatePage	*Stub — see section F*

Admin App Local Routes → Remote Endpoints

Local Route	Page Class	Remote Endpoint(s)
`/login`	AdminLoginPage	`POST /system/accounts/{acct}/configuration`, `GET /system/users/{acct}/configuration`
`/`	AdminDashboardPage	`GET /system/accounts`, per-account repo/history fetches
`/manage/accounts`	ManageAccountsPage	`GET /system/accounts`
`/manage/accounts/:acct`	ManageAccountPage	`GET /system/accounts/{acct}/configuration`
`/manage/repositories`	ManageRepositoriesPage	`GET /system/accounts/{acct}/repositories` (per checked account)
`/invitations`	AdminInvitationsPage	`GET /invitations`
`/invite`	AdminInviteNewPage	`POST /invitations`
`/history/queries`	QueryHistoryPage	`GET /system/accounts/{acct}/repositories/{repo}/service_h` (per checked repo)

Local Route	Page Class	Remote Endpoint(s)
`/history/transactions`	TransactionHistoryPage	`GET /system/service_history/transactions`

Graph Store Protocol Endpoints

The graph store endpoints at `/{acct}/{repo}` and `/{acct}/{repo}/service` support the full HTTP method set:

Method	Endpoint	Purpose
GET	`/{acct}/{repo}`	Export repository (content-negotiated: Turtle, N-Triples, N-Quads, RDF/XML, JSON-LD, TriG)
POST	`/{acct}/{repo}`	Add statements (import — merges with existing data)
PUT	`/{acct}/{repo}`	Replace all statements
DELETE	`/{acct}/{repo}`	Clear repository (remove all statements)
GET	`/{acct}/{repo}/service?graph={uri}`	Export named graph
POST	`/{acct}/{repo}/service?graph={uri}`	Add to named graph
PUT	`/{acct}/{repo}/service?graph={uri}`	Replace named graph
DELETE	`/{acct}/{repo}/service?graph={uri}`	Clear named graph

B. Server-Side Authentication Endpoints

B.1 Password Reset

Current state: The JSUI `ResetPasswordPage` renders a form that submits to the Rails Devise endpoint `POST /password`. This depends on the Rails application being deployed.

Rails implementation: Uses `accounts.reset_password_token` database field. Flow: 1. `POST /password` with email — generates token, stores in DB, sends email with link 2. `GET /password/edit?reset_password_token=...` — renders new-password form 3. `PUT /password` with token + new password — validates token, updates password

Proposed SPOCQ implementation: Add two endpoints to the account resource, following the existing `respond-to-account-configuration-get/post` pattern in `resource.lisp`:

```
POST /system/accounts/{acct}/password-reset
  Request: { "email": "user@example.com" }
  Action:  Generate random token, store in accounts.reset_password_token,
           send email with link to SPA reset page
  Response: 202 Accepted

POST /system/accounts/{acct}/password-reset/{token}
  Request: { "password": "newpassword", "password_confirmation": "newpassword" }
  Action:  Validate token matches accounts.reset_password_token,
           hash and store new password, clear token
  Response: 200 OK or 422 Unprocessable (invalid/expired token)
```

JSUI changes: - `ResetPasswordPage`: Update form to POST to `/system/accounts/{acct}/password-reset` - Add new route/page for the token-based password update form - SPOCQ email: Use CL-SMTP or equivalent for sending reset instructions

B.2 Email Confirmation

Finding: The Rails application does **not** implement email confirmation. The Devise `:confirmable` module is not enabled, and no `confirmation_token` or `confirmed_at` fields exist in the database schema. The confirmation email template exists but is unused.

Recommendation: Drop as a deficiency. If confirmation is desired as new functionality:

```
POST /system/accounts/{acct}/confirmation
  Action: Generate confirmation token, send email

GET /system/accounts/{acct}/confirmation/{token}
  Action: Validate token, set confirmed_at timestamp
```

This would require adding `confirmation_token` and `confirmed_at` columns to the accounts table.

B.3 Auth Token Reset

Current state: The JSUI `AccountAuthTokenPage` issues a POST to the Rails endpoint `POST /:acct/auth_token`, which calls Devise's `reset_authentication_token!`.

Rails implementation: The `accounts.authentication_token` field stores the API token. `reset_authentication_token!` generates a new random token and saves it.

Proposed SPOCQ implementation: Extend the existing account configuration endpoint:

```
POST /system/accounts/{acct}/configuration
```

```
  Request: { "reset_authentication_token": true }
  Action:  Generate new random token, store in accounts.authentication_token
  Response: { "authentication_token": "new-token-value", ... }
```

Alternatively, a dedicated endpoint:

```
POST /system/accounts/{acct}/auth-token/reset
  Response: { "authentication_token": "new-token-value" }
```

JSUI changes: - `AccountAuthTokenPage.afterRender()`: Change the reset POST target from Rails endpoint to the SPOCQ endpoint - Display the returned token value

---

C. Invitation Flow

Current state: The "Request Invite" form is on the home page and at `/invite`. A link from the invitation list page is not useful since unauthenticated users who need invitations cannot access it.

Rails implementation: Two-phase system: 1. User requests invite at `POST /invitations` with email. `InviteCode` generates a 5-character alphanumeric code. Admin notified via `InviteMailer.invite_signup_notification`. 2. Admin sends invite via `POST /invitations/:id/send_invite`. Code assigned, email sent via `InviteMailer.send_invite` with signup link containing `?invite_code=...`. 3. Signup validates invite code against `invitations` table.

Proposed changes:

C.1 Add invite link to login page

Add a "Request an invitation" link to the user app login page, below the login form. This is the natural discovery point for new users who don't have accounts.

C.2 SPOCQ invitation endpoints

These may already exist. Verify and ensure:

```
POST /invitations              - Create invitation request (public, requires email)
GET  /invitations              - List all invitations (authenticated)
POST /invitations/{id}/send    - Generate invite code and send email (admin)
DELETE /invitations/{id}       - Delete invitation (admin)
```

C.3 Email sending

SPOCQ needs SMTP capability for: - Invitation request notification to admin - Invite code email to user - Password reset instructions

Reference: Rails templates in `app/views/invite_mailer/`: - `send_invite.html.haml`: "Come give Dydra.com a try!" with invite code and signup link - `invite_signup_notification.html.haml`: "New Dydra.com Invitation Signup" to admin

---

D. Account-Account Collaboration

Finding: Does not exist in Rails. The `AccountCollaborationsController` has empty `index` and `destroy` stubs. This is not a deficiency to rectify — remove from the deficiency list.

E. Graph Store Protocol

Already documented in section A (table). The JSUI currently uses: - `GET /{acct}/{repo}` for export (`handleRepositoryExport`) - `POST /{acct}/{repo}/service` for import (`handleRepositoryImport`) - `DELETE /{acct}/{repo}` for clear (`handleRepositoryClear`)

The `PUT` method (full replacement) is available but not exposed in the UI. This could be added as an "Replace All" import option if needed.

F. Repository Metadata (Replacing TemplatePage Stubs)

Current state: Four user-app routes map to `TemplatePage` placeholders: - `/:acct/:repo/query_logs` - `/:acct/:repo/status` - `/:acct/:repo/size` - `/:acct/:repo/meta`

Available SPOCQ endpoints: - `/system/accounts/{acct}/repositories/{repo}/configuration` — returns JSON with `quad_count` and other metadata - `/system/accounts/{acct}/repositories/{repo}/st` — returns disk size attributes - `/system/accounts/{acct}/repositories/{repo}/service_statistics` — aggregated query statistics - `/system/accounts/{acct}/repositories/{repo}/service_history` — query performance time-series

Proposed resolution: These are not separate pages — they are aspects of the repository detail view. Integrate them into `RepositoryShowPage`:

1. **Meta/Status**: The repository show page sidebar already displays metadata from the configuration endpoint. Remove the separate `/meta` and `/status` routes. If direct URL access is needed, redirect to the repository show page.

2. **Size**: Display `quad_count` (from configuration) and disk size (from `/storage`) in the existing repository sidebar. Remove the separate `/size` route.

3. **Query logs**: Replace the `TemplatePage` with a real page that fetches from `/system/accounts/{acct}/repositories/{repo}/service_history` and displays a sortable table (same pattern as the admin query history pane). Or add a "Query History" tab to the repository pane.

G. View Execution Links

Current state: The repository page lists views in a query catalog. Each view has a link that opens the SPARQL editor (`QueryShowPage`) where the user can run the query interactively.

Proposed addition: Add an "Execute" link next to each view entry that returns results directly:

```
GET /{acct}/{repo}/{view}
  Accept: text/html
```

SPOCQ supports view execution with content type negotiation. The `text/html` response wraps SPARQL results in an HTML document.

Implementation: - In `RepositoryShowPage`, add an execution link to each view entry in the query catalog: `html <a href="/{acct}/{repo}/{view}" data-external target="_blank" title="Execute">Run</a>` - The `data-external` attribute prevents the SPA router from intercepting the link - Opens in a new tab since the response is a standalone HTML document from the server

---

H. Import UI Improvements

H.1 Disable during request

Apply the same disable pattern used by export: disable both the import and export buttons while either operation is in progress. Currently `handleRepositoryImport` disables the import button and changes text to "Importing…" — extend this to also disable the export button, and vice versa.

H.2 Asynchronous import with progress

Replace `fetch()` with `XMLHttpRequest` in `handleRepositoryImport` to access upload progress events:

```
const xhr = new XMLHttpRequest();
xhr.open("POST", url);
xhr.setRequestHeader("Content-Type", contentType);
xhr.setRequestHeader("Authorization", `Bearer ${token}`);
```

```
// Progress tracking
xhr.upload.onprogress = (e) => {
  if (e.lengthComputable) {
    const pct = Math.round((e.loaded / e.total) * 100);
    button.textContent = `Importing... ${formatBytes(e.loaded)}/${formatBytes(e.total)} (${p
  } else {
    button.textContent = `Importing... ${formatBytes(e.loaded)}`;
  }
};

xhr.onload = () => { /* handle success/error */ };
xhr.onerror = () => { /* handle error */ };
xhr.send(file);
```

H.3 Byte count display

Before upload begins, show the file size:

```
button.textContent = `Importing ${fileName} (${formatBytes(file.size)})...`;
```

H.4 Abort capability

Store the `xhr` reference and add a Cancel mechanism:

```
// Replace button text with Cancel during upload
button.textContent = "Cancel Import";
button.onclick = () => xhr.abort();

xhr.onabort = () => {
  showStatusMessage("Import cancelled.");
  button.textContent = originalText;
  button.onclick = null;
};
```

H.5 Import history tab

SPOCQ provides `GET /system/service_history/imports` with query parameters `account`, `agent`, `limit`, `offset`. Response fields: `uuid`, `timestamp`, `account_key`, `repository_key`, `agent_key`, `source_uri`, `base_uri`, `context_uri`, `quad_count`, `success`.

Add an "Import History" tab to the admin dashboard, following the same pattern as the existing Transaction History tab. This data comes from the same PostgreSQL `repository_import_logs` table used by the Rails application.

I. Authorization Model

Current state

The admin SPA requires administrator privileges at login. Non-admin users cannot access any admin features.

Proposed change

Allow any authenticated user to access the admin SPA, but conditionally display side-effect controls:

1. **Login flow**: Remove the admin privilege check from `AdminLoginPage`. Authenticate normally via `/system/accounts/{acct}/configuration`.
2. **Role detection**: After successful authentication, query `/system/users/{acct}/configuration` and check for `administrator_of`. Store the result as `state.isAdmin` (boolean).
3. **UI gating**:
   - **Always visible** (for all authenticated users): Account list, repository list, invitations list, query history, transaction history, import history. The server will return only what the user is authorized to see.
   - **Admin-only** (hidden when `!state.isAdmin`): Delete account button, Delete repository button, Create account button, Send invitation button, Delete invitation button.
4. **Server enforcement**: The SPOCQ server already enforces authorization. If a non-admin user attempts a restricted operation, the server returns 403. The UI gating is a convenience, not a security boundary.

J. Admin Invitations

Current state: The admin SPA has invitation management (list, send, delete, create). The user app has an invitation request page at `/invite`.

Recommendation: - Keep the admin SPA's invitation list with send/delete for managing existing invitations - Remove the "New Invitation" creation from the admin SPA (it duplicates the user app's `/invite`) - Or: keep it for admin use (creating invitations on behalf of users) - Add a "Request Invite" link on the user login page pointing to `/invite`

K. SEO

K.1 Meta tags

Add to `index.html` `<head>`:

```
<!-- Open Graph -->
<meta property="og:title" content="Dydra - Cloud-hosted RDF Graph Database" />
<meta property="og:description" content="Store, query, and manage RDF data in the cloud with
<meta property="og:image" content="https://dydra.com/images/dydra-icon.png" />
<meta property="og:url" content="https://dydra.com" />
<meta property="og:type" content="website" />

<!-- Twitter Card -->
<meta name="twitter:card" content="summary" />
<meta name="twitter:title" content="Dydra - Cloud-hosted RDF Graph Database" />
<meta name="twitter:description" content="Store, query, and manage RDF data in the cloud wit

<!-- Improved description -->
<meta name="description" content="Dydra is a cloud-hosted RDF graph database with SPARQL que

<!-- Canonical -->
<link rel="canonical" href="https://dydra.com" />
```

K.2 Structured data

Add JSON-LD to `index.html`:

```
<script type="application/ld+json">
{
  "@context": "https://schema.org",
  "@type": "WebApplication",
  "name": "Dydra",
  "url": "https://dydra.com",
  "description": "Cloud-hosted RDF graph database with SPARQL support",
  "applicationCategory": "Database",
  "operatingSystem": "Web"
}
</script>
```

K.3 Noscript fallback

Add to `index.html` `<body>`:

```
<noscript>
  <h1>Dydra</h1>
  <p>Dydra is a cloud-hosted RDF graph database. JavaScript is required to use this applicat
  <ul>
    <li><a href="https://docs.dydra.com">Documentation</a></li>
    <li><a href="https://blog.dydra.com">Blog</a></li>
  </ul>
</noscript>
```

K.4 Server-side pre-rendering

For the public home page, consider serving a static HTML version that includes the full page content. The SPA can then hydrate on top of it. This requires the SPOCQ server to serve `index.html` with pre-rendered content for crawlers (detect via `User-Agent` or use a pre-rendering service).

This is lower priority — the authenticated pages (account, repository, query) do not need SEO.

L. CSS Dependency Elimination

L.1 Bootstrap

Finding: Bootstrap CSS files are loaded but **no Bootstrap classes are used** anywhere in the application. All layout and styling uses custom CSS in `stylesheets/style.css` and `stylesheets/jsui-overrides.css`.

Action: Delete the following files: - `css/bootstrap.css` - `css/bootstrap.min.css` - `css/bootstrap-theme.css` - `css/bootstrap-theme.min.css` - `css/bootstrap-datepicker.css` - `css/tablesorter/theme.bootstrap.css`

No HTML or JS changes required.

L.2 jQuery UI / Aristo

Finding: No jQuery UI JavaScript is loaded. The Aristo CSS theme is loaded for styling only. Actual usage is limited to:

Icon classes (8 distinct icons, ~18 total references): - `ui-icon-alert` (5x) — error messages - `ui-icon-circle-arrow-w` (4x) — back navigation - `ui-icon-link` (3x) — link indicator - `ui-icon-disk` (2x) — save action - `ui-icon-trash` (1x) — delete action - `ui-icon-plusthick` (1x) — add/new action - `ui-icon-play` (1x) — run query - `ui-icon-help` (1x) — help button

State/layout classes (~25 references): - `ui-state-error` (6x) — error message styling - `ui-state-hover` (4x) — hover effects - `ui-state-highlight` (2x) — success messages - `ui-corner-all` (7x) — rounded corners - `ui-widget` (7x) — widget container

Replacement strategy:

1. Create 8 SVG icon files in `images/icons/` (or use inline SVG in the templates)
2. Add replacement CSS classes to `stylesheets/jsui-overrides.css`:

```
/* Replace ui-widget + ui-state-error */
.message-error {
  border: 1px solid #c44;
```

```
  background: #fef0f0;
  padding: 8px 12px;
  border-radius: 4px;
}
/* Replace ui-state-highlight */
.message-success {
  border: 1px solid #4a4;
  background: #f0fef0;
  padding: 8px 12px;
  border-radius: 4px;
}
/* Replace ui-icon with SVG background */
.icon { display: inline-block; width: 16px; height: 16px; vertical-align: middle; }
.icon-alert { background: url('../images/icons/alert.svg') no-repeat center; }
.icon-back { background: url('../images/icons/back.svg') no-repeat center; }
/* etc. */
```

3. Update class references in:
   - `ui/pages/index.js` (~30 references)
   - `signup.html` (~3 references)
   - `reset_password.html` (~3 references)
   - `admin.html` (~3 references)
4. Remove from all HTML files:

   ```
   <link rel="stylesheet" href="./stylesheets/jquery-ui/aristo/jquery-ui-1.8.7.custom_comp
   ```

5. Delete:
   - `stylesheets/jquery-ui/` directory (Aristo theme CSS and images)
   - `stylesheets/jquery-ui-1.8.24.min.css`

M. SPOCQ Server Implementation References

For server-side implementations of password reset and email, the relevant SPOCQ code patterns are:

M.1 Account resource pattern

In `resource.lisp`, account configuration endpoints follow this pattern:

```
/system/accounts/{account}/configuration
  GET  → respond-to-account-configuration-get
  POST → respond-to-account-configuration-post
```

Password reset should follow the same resource model:

```
/system/accounts/{account}/password-reset
```

```
  POST → respond-to-account-password-reset-request  (generate token, send email)

/system/accounts/{account}/password-reset/{token}
  POST → respond-to-account-password-reset-complete  (validate token, set password)
```

M.2 Database access

SPOCQ accesses MySQL/PostgreSQL for import logs and usage stats. The same database connection can be used to: - Read/write `accounts.reset_password_token` - Read/write `accounts.authentication_token` - Read/write `invitations` table

M.3 Email sending

SPOCQ does not currently send email. Options: - **CL-SMTP**: Common Lisp SMTP client library - **Delegate to external service**: Use a mail relay (Postfix, SendGrid API, etc.) - **Minimal template**: Plain text email with reset link / invite code — no need for HTML templates initially

---

N. Import and Query History

N.1 Import history

SPOCQ provides `GET /system/service_history/imports` serving data from the `repository_import_logs` table (same table used by Rails).

Response fields: `uuid`, `timestamp`, `account_key`, `repository_key`, `agent_key`, `source_uri`, `base_uri`, `context_uri`, `quad_count`, `success`

Query parameters: `account`, `agent`, `limit`, `offset`

JSUI implementation: Add an "Import History" tab to the admin dashboard, between "Transaction History" and the end. Use the same sortable table pattern as the existing history tabs. Columns: timestamp, account, repository, source URI, quad count, success.

N.2 Query history

Already implemented in the admin SPA via the "Query History" tab, fetching from `/system/accounts/{acct}/repositories/{repo}/service_history` per selected repository.

The SPOCQ server also provides aggregate endpoints: - `GET /system/service_history/queries` — system-wide query history - `GET /system/service_statistics/queries` — aggregated query statistics

These could supplement the per-repository view currently implemented.

O. Payment Mechanism

No payment mechanism is intended. Remove `PaymentHistoryPage` or leave as a no-op that redirects to the dashboard (current behavior).

Implementation Priority

Phase 1: Client-side only (no server changes)

1. **L.1** Delete unused Bootstrap CSS files
2. **F** Replace TemplatePage stubs with real metadata display using existing SPOCQ endpoints
3. **G** Add view execution links to repository page
4. **H.1** Disable import/export buttons during requests
5. **H.2-H.4** Import progress tracking with byte count and abort
6. **I** Change admin authorization to allow any authenticated user, hide admin-only buttons
7. **K.1-K.3** Add SEO meta tags, structured data, noscript fallback
8. **J** Add invite link to user login page
9. **N.1** Add Import History tab to admin dashboard

Phase 2: Client + CSS refactor

10. **L.2** Replace jQuery UI/Aristo classes with custom SVG icons and CSS

Phase 3: Server-side (SPOCQ changes required)

11. **B.1** Password reset endpoints
12. **B.3** Auth token reset endpoint
13. **C.2-C.3** Invitation endpoints with email sending
14. **M.3** SPOCQ email sending capability

Not applicable

- **B.2** Email confirmation — not implemented in Rails, not a regression
- **D** Account-account collaboration — does not exist in any implementation
- **O** Payment history — not intended