Promptly Project: Technical Documentation & Architectural Report

Overview

Promptly is a fully browser-based asset management, collaboration, and documentation system engineered specifically for use by field engineers, site operators, project managers, and executive stakeholders. It facilitates real-time organization of engineering assets, contextual notes, secure file uploads, and QR-code tracking for fast retrieval—all while offering robust, structured access control. Promptly is optimized for portable, static deployment with no traditional server requirements, relying entirely on Supabase for backend services. The project emphasizes rapid deployment (via GitHub Pages), zero-maintenance hosting, and role-specific functionalities with advanced permission controls.

Tech Stack

Frontend (Client-Side Only)

- **HTML5 / CSS3 / JavaScript (Vanilla)**: Native browser interfaces, mobile-first responsive layout
- QRCode.js for in-browser QR generation (used to identify assets)
- GitHub Pages for static asset hosting, enabling free HTTPS distribution without backend servers

Backend (Supabase Platform)

- **Supabase** (as a complete Backend-as-a-Service stack)
- PostgreSQL for relational database management
- Supabase Auth for secure user authentication and email verification
- Supabase Storage for storing and retrieving uploaded files
- Row-Level Security (RLS) to restrict access based on user roles and hub membership

Note: Supabase Storage now handles all file management. Cloudflare R2 is not currently in use.

JavaScript SDKs & External Libraries

- `@supabase/supabase-js@2` – Full Supabase integration for client-side operations
- `qrcode.min.js` – Pure JavaScript QR code renderer used to associate each asset with a scannable identifier

---

HTML Files (Frontend Interfaces)

File Name	Description
`index.html`	Public-facing landing page showcasing the platform's features
`signup.html`	Sign-up portal for root users to create new hubs and accounts

File Name	Description
`field_login_ui.html`	Login interface for all user roles (admin, root, manager, operator)
`dashboard.html`	Primary interface for asset management, file uploads, and collaboration
`asset_page.html`	Displays asset details, attached files, QR codes, and contextual notes
`user_management_panel.html`	Available to root/admin users to manage user approvals, roles, and hubs

Each HTML file is modular and standalone, with embedded JavaScript logic that directly interacts with Supabase. They rely entirely on in-browser execution.

Supabase Database Tables (Schema)

1. `users`

Tracks authentication metadata and contextual roles.

- `id`: Primary key
- `auth_id`: UUID from Supabase Auth
- `email`: User email address
- `role`: Enum (`admin`, `root`, `manager`, `operator`)
- `status`: Enum (`pending`, `approved`)
- `hub_id`: Foreign key linking user to a hub

2. `hubs`

Each root user manages one hub; all assets and team members are scoped to this hub.

- `id`: Primary key
- `owner_id`: The root user who created the hub
- `name`: Display name for the hub

3. `assets`

Engineering assets created by managers or root users.

- `id`: Primary key
- `hub_id`: Foreign key to hubs table
- `name`: Human-readable asset name
- `identifier`: Unique string/code
- `description`: Summary of asset
- `context_prompt`: Background context or notes

4. `notes`

Textual notes and observations attached to an asset.

- `id`: Primary key
- `asset_id`: Foreign key to asset
- `author_id`: Foreign key to users table
- `content`: Note body
- `created_at`: Timestamp

5. `files`

Metadata for files uploaded to Supabase Storage.

- `id`: Primary key
- `asset_id`: Foreign key to asset
- `file_url`: Full public/private Supabase Storage URL
- `file_name`: Display name
- `created_at`: Timestamp

Authentication and Access Logic

Sign-Up Flow

1. User signs up via `signup.html`.
2. Supabase sends a confirmation email with redirect.
3. A new row is inserted into `users` table with:
4. `status = pending`
5. `role = operator` by default
6. Admin/root user must manually approve and optionally elevate role to manager or root.

Login Flow

1. Login handled through `field_login_ui.html` using Supabase Auth.
2. On successful login:
3. Supabase session is stored
4. Corresponding `users` row is validated (`status = approved`)
5. Approved users are routed to `dashboard.html`.

Role-Based Permissions Matrix

Role	Add Assets	View Assets	Add Users	Approve Users	Upload Files	Add Notes	Delete Items	Cross-Hub Access
Admin	✅	✅	✅	✅	✅	✅	✅	✅
Root	✅	✅	✅	✅	✅	✅	✅	❌

Role	Add Assets	View Assets	Add Users	Approve Users	Upload Files	Add Notes	Delete Items	Cross-Hub Access
Manager	✅	✅	❌	❌	✅	✅	❌	❌
Operator	❌	✅	❌	❌	❌	✅	❌	❌

Row-Level Security (RLS) Policies

`users`

```
CREATE POLICY "Allow self-view" ON users
  FOR SELECT USING (auth.uid() = auth_id);
```

`hubs`

```
CREATE POLICY "Owner can create and view hub" ON hubs
  FOR SELECT, INSERT
  USING (auth.uid() = owner_id)
  WITH CHECK (auth.uid() = owner_id);
```

`assets`

```
CREATE POLICY "Hub members manage assets" ON assets
  FOR SELECT, INSERT, UPDATE, DELETE
  USING (hub_id IN (SELECT hub_id FROM users WHERE auth_id = auth.uid()));
```

`notes`, `files`

Policies mirror the asset-level permissions using subqueries against the `users` and `hubs` tables.

Known Issues Prior to Rebuild

1. **Redirect Loop After Login**:

2. Possible delay in session availability.

3. Solution: Use `onAuthStateChange` to reliably detect login state before proceeding.

4. **Hub Not Being Created for New Root Users**:

5. `getOrCreateHub()` fails due to premature redirection or missing session.

6. Ensure session is loaded before this logic.

7. **Supabase Storage File Size Limitations**:

8. For free-tier accounts, files are capped per upload.

9. Workaround: Enforce upload limits client-side and show alerts.

10. **Browser Compatibility**:

11. GitHub Pages may serve cached versions, causing older JavaScript to run.

12. Recommend forcing reload or version query strings in URLs.

Rebuild Strategy & Best Practices

- Use modular JS files and shared utility functions
- Validate Supabase session early before executing logic
- Store hub creation timestamp for onboarding tracking
- Introduce debug mode to surface session or RLS policy issues
- Add support for inviting members by email, marking them as `pending`
- Add visual indicators per role on dashboard interface
- Include file previews, file size checks, and QR links that redirect to individual `asset_page.html?id=` routes

---

Final Notes

This document comprehensively captures all current architectural and technical considerations of the Promptly platform. It includes schema definitions, authentication workflows, role-based access enforcement, file handling strategies, and known edge cases. It is structured to support future developers, engineers, and ChatGPT-based planning for rebuilding or extending the system.

With this foundation, Promptly can be reengineered for greater resilience, improved UX, and support for new features like real-time collaboration, notifications, and dashboard analytics. This report should serve as the canonical blueprint for any follow-up implementation or restart of the project lifecycle.