**NR-DAP-UI Requirements and Specifications**

**1. Overview:**

**nr-dap-ui** is a web-based user interface designed to streamline data extraction and automation processes for various business units. It enables users to interact with complex data sources, like databases, spreadsheets, and APIs, using a low-code/no-code approach. The platform automates the creation of Airflow Directed Acyclic Graphs (DAGs) for scheduling data extraction and processing workflows. With built-in user management, role-based access control, and integration with tools like Keycloak for SSO and Vault for secret management, the application simplifies data access, extraction, and automation tasks, providing an efficient interface for business users to create custom Power BI dashboards. The platform provides a low-code solution for users and administrators to manage data, run automated workflows, and generate reports.

Repo:

[bcgov/nr-dap-ui (github.com)](https://github.com/bcgov/nr-dap-ui)

Flow chart can be found here:

<https://miro.com/app/board/uXjVK5vMa6k=/?share_link_id=973215464778>

**2. Core Features:**

* User Authentication and Role Management:
  + Authentication: The app uses Keycloak for user authentication with SSO, offering session management and securing access to different resources based on user roles (admin or non-admin).
  + Role Management: Admin users have control over assigning roles and managing access to various databases and features. Admins can access all databases, while non-admin users are restricted to databases where they are the owner.
* Database Integration and Data Sync:
  + PostgreSQL Integration: PostgreSQL is used to store replication data and metadata such as user roles, applications, database configurations, and logs of user activity.
  + Oracle Integration: Oracle is the primary database used for extracting and replicating data tables. The app fetches table metadata (columns, types, etc.) and migrates them into PostgreSQL for analysis.
  + Vault Integration: Credentials for database access are securely stored and managed using Vault. These credentials are fetched dynamically when a database connection is established.
* Automated Data Extraction and Workflow Management:
  + Airflow DAG Integration: Airflow DAGs are triggered to automate workflows like data replication, schema creation, and syncing between Oracle and PostgreSQL.
  + Table Metadata Processing: The app allows users to select tables from Oracle databases, process their schema, and replicate them in PostgreSQL.
  + Schema and Database User Management: Automatically creates replication schemas and a team user in PostgreSQL based on the selected tables and grants role-based access to these schemas.
* Power BI Cloud API Integration:
  + File Uploads: Users can upload Power BI report files (.pbix format) and publish them directly to Power BI online workspaces.
  + Report Management: After publishing a report, the app retrieves report details (e.g., report ID, view link) and presents them to users. Users can also fetch and manage dataset access permissions.

**3. Detailed Functional Requirements:**

* User Registration and Login:
  + Users must log in using Keycloak authentication. The system should restrict access to non-logged-in users.
  + Admin users must have permissions to manage roles and users, ensuring restricted access to critical sections like user, role and database management.
* Database and Application Management:
  + Database Listing: Admins can view all databases and their details, including the application, owner(team user), schema, and Vault name.
  + Add/Edit/Delete Database: Admins can add new databases, edit existing database details, or delete databases. Non-admin users can only view databases they own.
  + Data Sync Management: Users (especially admins) can initiate data sync tasks that extract table metadata from Oracle and replicate it in PostgreSQL.
* Automation and Workflow Management:
  + Data Table Processing: Users can select specific tables from Oracle, view column metadata, and generate Data Definition Language (DDL) scripts for creating replicated tables in PostgreSQL.
  + Airflow DAG Triggering: After table selection and replication schema generation, users can trigger Airflow DAGs to automate data extraction and replication tasks. The DAG status must be monitored to ensure smooth execution.
  + PostgreSQL database user will be created and selected tables access will be granted.
  + CDC master table (Airflow will create/update DAGs based on this table.) rows will be inserted for selected tables.
* Power BI File Upload and Publishing:
  + Users should be able to upload Power BI reports (.pbix files) through the interface. Upon upload, the file should be published to the appropriate Power BI workspace based on the environment (dev, test, prod).
  + The application must handle authentication using MSAL to retrieve access tokens for Power BI.
  + After publishing, users should receive a link to view the report directly in the Power BI portal.

**4. Non-functional Requirements:**

* Scalability:
  + The app should handle multiple databases and numerous users efficiently.
  + The infrastructure should support expanding data sources or integrating additional BI tools in the future.
* Security:
  + Authentication via Keycloak ensures secure login and role-based access control.
  + All credentials are stored securely in Vault, with dynamic fetching during operations to minimize exposure.
* Performance:
  + The application must minimize latency in data extraction and replication processes.
  + Power BI report publishing should be optimized to handle large files and multiple environments.

**5. Key Features and Technologies:**

* Self-service data extraction for Power BI
* SSO via Keycloak for secure access
* Workflow automation with Airflow DAGs
* Low-code interface to reduce manual tasks
* Integration with multiple data sources including databases and object storage
* **Node.js**: Backend framework.
* **Express.js**: For handling routes.
* **PostgreSQL**: Primary database for storing data.
* **Vault API**: For secret management (e.g., database credentials).
* **Keycloak**: For Single Sign-On (SSO) and user authentication.
* **EJS (Embedded JavaScript Templates)**: For front-end rendering.
* **Docker**: For containerization.
* **Airflow DAG**: For automating data extraction workflows.
* **Power BI API**: Used to upload and publish Power BI reports, enabling business users to push data directly to Power BI and automate dashboard generation.
* **package.json** file for this project defines the application's dependencies, development tools, and scripts. Below is a detailed :

**Dependencies:**

* + **@azure/msal-node**: Handles Microsoft authentication for Power BI API integration.
  + **bcrypt**: Encrypts sensitive data like passwords.
  + **express**: Main framework for building the Node.js backend.
  + **axios**: Makes HTTP requests, e.g., for Power BI and Vault integration.
  + **multer**: Handles file uploads (e.g., Power BI files).
  + **pg**: PostgreSQL client.
  + **oracledb**: Oracle Database client.
  + **keycloak-connect**: Integrates Keycloak SSO for authentication.
  + **webpack**: JavaScript bundler for development.
  + **webpack-cli**: Provides CLI tools for Webpack.

**6. Project Setup**

Document the necessary steps for developers to set up the project from scratch:

1. **Clone the repository:**  
   git clone https://github.com/bcgov/nr-dap-ui.git
2. **Install dependencies:**  
   Install all dependencies.
3. **Environment setup:**  
   Provide instructions on setting up the .env file with the necessary configurations:

**.env**

*KEYCLOAK\_SECRET=""*

*KEYCLOAK\_URL=*[*https://dev/test/prod.loginproxy.gov.bc.ca/auth*](https://dev/test/prod.loginproxy.gov.bc.ca/auth)

*CONFIG\_BROKER\_JWT=" "*

*CONFIG\_ENV=dev*

*CONFIG\_VAULT\_ENV=dev*

*AIRFLOW\_USERNAME=*

*AIRFLOW\_PASSWORD=*

*DATABASE\_ODS\_IN\_VAULT=dapui-proxy-ods-dev*

*TENANT\_ID=" "*

*CLIENT\_ID=" "*

*CLIENT\_SECRET=" "*

*SESSION\_SEC=*

*REDIRECT\_URI=http://nr-dap-ui.apps.emerald.devops.gov.bc.ca*

1. **Database setup**:  
   Guide the team on initializing the PostgreSQL database using DDL:

<https://github.com/bcgov/nr-dap-ui/blob/main/dataModel/ddl.sql>

**7. Application Architecture**

The **nr-dap-ui** application architecture is divided into several layers and modules, each handling distinct responsibilities to ensure scalability, maintainability, and efficiency.

**7.1 Root Files and Important Directories**

1. **app.js**: The main entry point of the application. Initializes Express , routes, and starts the server.
2. **package.json**: Lists project dependencies and scripts.
3. **keycloak.js**: Configures Keycloak for authentication and Single Sign-On (SSO).
4. **Dockerfile**: Instructions for containerizing the application using Docker.
5. **dataModel/**: SQL schema and scripts for PostgreSQL, defining data structures and tables.
6. **modules/**: Contains key functionalities such as API communication with Vault and data extraction modules.
7. **public/**: Hosts static assets like CSS, and images.
8. **routes/**: Defines the various routes/endpoints in the application.
9. **views/**: EJS templates for rendering dynamic HTML pages.

**7.2 Modules Folder**

This folder contains helper modules for integrating external services (Vault API, Oracle DB, PostgreSQL) and for business logic execution:

1. **getCredentialsFromVault.js**:
   * Retrieves credentials from Vault for database connections (both Oracle and PostgreSQL).
2. **connectDatabase.js**:
   * Contains the logic for establishing and caching connections to databases based on credentials retrieved from Vault. It supports both Oracle and PostgreSQL connections.
   * **Functions**:
     + connectDatabase(secretName): Establishes and caches database connections using Vault credentials.
     + useCredentials(secretName): Retrieves and returns schema credentials from the cache.
3. **connectToOracleDatabase.js**:
   * Establishes a connection to Oracle databases.
   * **Functions**:
     + connectToOracleDatabase(credentials): Uses OracleDB library to connect using credentials retrieved from Vault.
4. **connectToPosgDatabase.js**:
   * Manages the PostgreSQL connection pooling and health checking.
   * **Functions**:
     + connectToPosgDatabase(credentials): Connects to PostgreSQL and verifies the connection.
     + getPool(credentials): Creates a new PostgreSQL connection pool if none exists or if the pool is unhealthy.
     + isPoolHealthy(): Checks if the connection pool is alive by performing a health check query.
5. **verifyUserEmail.js**:
   * Verifies if a user's email exists in the database.
   * **Function**:
     + verifyUserEmail(email, client): Queries the PostgreSQL database to verify the user's email.
6. **getUserDetails.js**:
   * Fetches user details such as role and associated applications from the database.
   * **Functions**:
     + getUserDetails(email, client): Retrieves user information and associated applications from PostgreSQL.
7. **logUserVisit.js**:
   * Logs user visits into the system.
   * **Function**:
     + logUserVisit(userDetails, client): Inserts a user visit log into the PostgreSQL database.
8. **triggerAirflowDAG.js**:
   * Communicates with the Airflow API to trigger DAGs and manage Airflow workflows.
   * **Functions**:
     + triggerAirflowDAG(): Triggers an Airflow DAG based on the configuration environment.
     + isDAGRunning(): Checks if an Airflow DAG is currently running before triggering a new instance.

**Application Architecture Documentation - Routes Breakdown**

**1. appRoutes.js**

* **Purpose**: Handles the management and display of applications associated with users.
* **Functions**:
  + **getApplications**: Retrieves the list of applications either for all users (if admin) or specific to the authenticated user.
  + **GET '/'**: Renders the appList view with applications based on the user's role (admin or non-admin).

**2. automation.js**

* **Purpose**: Manages automation tasks like table extraction, schema and tables creation, CDC master table updating, team user creation for ODS PostgreSQL database and integration with Oracle and PostgreSQL databases.
* **Functions**:
  + **getColumnsForTable**: Retrieves metadata (columns and comments) for Oracle tables.
  + **insertIntoDatatable**: Inserts table details into the PostgreSQL ODS datatable.
  + **POST '/generate-ddl'**: Generates the DDL (Data Definition Language) for creating tables based on the user's input and triggers Airflow DAG for automation.

**3. databaseRoutes.js**

* **Purpose**: Manages database configuration for applications, including viewing, adding, editing, and deleting databases.
* **Functions**:
  + **GET '/'**: Retrieves and lists all databases.
  + **GET '/add'**: Displays the form for adding a new database.
  + **POST '/add'**: Inserts a new database record into PostgreSQL.
  + **GET '/edit/**

**'**: Renders the form to edit a specific database.

* + **POST '/edit/**

**'**: Updates an existing database.

* + **POST '/delete/**

**'**: Deletes a database from PostgreSQL.

**4. dataTableRoutes.js**

* **Purpose**: Manages data tables within specific applications.
* **Functions**:
  + **isAdmin**: Checks if the logged-in user is an admin.
  + **getApplicationDetails**: Fetches details about tables in applications.
  + **GET '/'**: Lists tables within applications for either admin or regular users.
  + **POST '/delete/**

**'**: Deletes tables and revokes access to the application’s data.

**5. roleRoutes.js**

* **Purpose**: Manages user roles within the system.
* **Functions**:
  + **GET '/'**: Lists all roles.
  + **GET '/add'**: Displays the form to add a new role.
  + **POST '/add'**: Adds a new role to PostgreSQL.
  + **GET '/edit/**

**'**: Renders the form for editing a specific role.

* + **POST '/edit/**

**'**: Updates an existing role.

* + **POST '/delete/**

**'**: Deletes a role from PostgreSQL.

**6. uploadBI.js**

* **Purpose**: Handles file upload and Power BI report publishing.
* **Functions**:
  + **getPowerBIToken**: Retrieves an authentication token for Power BI API.
  + **publishToPowerBI**: Publishes the uploaded Power BI report to a specific workspace.
  + **GET '/'**: Renders the form for uploading a Power BI report.
  + **POST '/upload'**: Handles the report upload and triggers the publishing process.

**7. userRoutes.js**

* **Purpose**: Manages users within the system, including adding, editing, and deleting user accounts.
* **Functions**:
  + **GET '/'**: Lists all users with their roles.
  + **GET '/add'**: Renders the form to add a new user.
  + **POST '/add'**: Inserts a new user into PostgreSQL.
  + **GET '/edit/**

**'**: Renders the form for editing a user.

* + **POST '/edit/**

**'**: Updates user information.

* + **POST '/delete/**

**'**: Deletes a user from PostgreSQL.