Architectural UI Composition of DiscoveryStudio

Oleksandr Mostovyi (omostovyi)

7/08/2024

[Introduction 1](#_Toc171427319)

[Customer Ready SPA Requirements 2](#_Toc171427320)

[SPA Business Logic 2](#_Toc171427321)

[SPA Structure Overview 3](#_Toc171427322)

[Snapshot Management 5](#_Toc171427323)

[All Objects View mode 5](#_Toc171427324)

[DB Objects View mode 6](#_Toc171427325)

[DB Extensibility View mode 7](#_Toc171427326)

[Application Extensibility View mode 8](#_Toc171427327)

[File Objects View mode 9](#_Toc171427328)

[Cognos Reports View mode 10](#_Toc171427329)

[Cron Jobs View mode 11](#_Toc171427330)

[Full Text Search mode 12](#_Toc171427331)

[SPA Code Base 12](#_Toc171427332)

[Toolkit 12](#_Toc171427333)

[Folder Structure 12](#_Toc171427334)

[Appendices 14](#_Toc171427335)

[**Appendix 1**: Readiness Checklist 14](#_Toc171427336)

[**Appendix 2**: Framework Key Components’ and Their APIs 15](#_Toc171427337)

## Introduction

This document describes the DS UI composition that is part of \*\*\*. The composition is implemented in form of the browser-based Single Page Application **(SPA)** and is designed to enable users of the tool to operate, monitor and support a discovery process that precedes migration. The existing set of tools to analyze discovery snapshots widen the qualified personnel capabilities in screening the system in question, comparing its integral parts revisions, and adjusting accordingly the oncoming migration procedures.

The SPA, which also may be referred to in other documents as ***Studio UI***, is completely portable and may be deployed, virtually, on any webserver on top of any OS or in the cloud. The toolkit used for its implementation is up-to-date and free from any known vulnerability for the time of this writing.

## Customer Ready SPA Requirements

The main source of requirements for delivering DS SPA is the Customer. According \*\*\* *Acceptance Criteria* the SPA should meet the following requirements:

1. The SPA should be cross-platform.
2. The SPA is (and it should be) web based.
3. The SPA UI should be intuitive and easy to use.
4. The SPA should be well-documented.
5. The SPA toolkit and dependencies should be free from severe vulnerabilities: according to Snyk.
6. The SPA source code quality should be confirmed with SonarQube test coverage: over 90% for each work item (to be tested).
7. The SPA should be performant:
	1. Have a good/fair loading speed.
	2. Have a reasonable level of resource consumption.
	3. Be scalable.
8. The SPA should be maintainable.

Please, check [Appendix 1: Readiness Checklist](#_Appendix_1:_Readiness) below in this document for the current state of things.

## SPA Business Logic

**Studio UI** SPA contains mandatory features required to prepare and deliver any migration process successfully**.** Here’s the SPA features:

**Browse:** It makes it possible to browse collected entities from the source system(s) during the discovery process. The tree-like object browser UI block defines the entry point of one’s browsing as well as reflecting the inner structure of a snapshot. Any leaf or node in this tree is an object having properties or a sub-tree of its own children.

**View:** It helps to view an entity’s content in the event it is available in the human readable form. The SPA Viewer UI block opens readable files in read-only mode, recognizing plain text or programming language of a file. A built-in feature of its own is searching within the file. The block supports multi-tab, i.e. multiple open files, mode by means of added functionality.

**List:** It can list an entity property set. Listing of an entity property or a process list is implemented as a data grid. Extra controls of the customized ***Grid*** UI block facilitate User’s group operations on uniform data sets.

**Search:** The User may search through a selected snapshot for an entity or within its content. Searching UI blocks – basic full-text and extended ones with saving search results – makes navigation over a snapshot a breeze. The User has full control on what to search and how to save it.

**Snapshot management:** Adds the management flexibility: turning on and off snapshots for further analysis of its content or retrying discovery in the post-discovery phase. A snapshot download option makes local snapshot analysis possible and or within a specific environment.

**Snapshots analysis:** Allows users to run different analyzers on collected metadata and shows the results.

## SPA Structure Overview

The central container component shapes the whole SPA layout. The three parts of layout look as following:



*Diagram 1*

Block 1 (left-side vertical). The right vertical border is dynamic – moving horizontally and shrinking or growing Block 2 and 3. Additionally, there is another splitter between Block 2 and 3, a vertically moving border between the blocks.

Every Framework component may belong to virtually any Block of the SPA due to its nature – it is a React component, and this is exactly why it is self-sufficient and completely portable on its own.

All three blocks contain at least one placeholder for the static or dynamic React component(s). However, the layout may be overlaid with some other components to comfort specific UI blocks meeting particular business case requirements.

The SPA layout is tailored to fit key components of the Framework, and those are:

1. Object Tree Browser component- reflects hierarchy of objects included in the snapshot
2. Object Content Viewer component – shows content of object e.g. DDL scripts, source files text etc.
3. Object Property Viewer component – shows collected properties of selected object
4. Data Grid component – shows the collected metadata in the tabular format
5. Extended Search component - this component enables users to create and save custom search queries. It also displays the results of these searches

Please, check [Appendix 2: Framework Key Components’ and Their APIs](#_Appendix_2:_Framework) for the current state of things.

Typical layout of the SPA key components is like this:



*Diagram 2*

### Snapshot Management

***Snapshot Management*** chores add to the management flexibility: *turning on* and *off* snapshots for further analysis of its content or retrying discovery in the post-discovery phase. A snapshot download option makes local snapshot analysis possible and or within a specific environment.



Any snapshot operation logs out its own details; any snapshot reflected with Studio UI may be in one of three states: loaded (or imported), (a snapshot operation) being in progress, and unloaded (from the system) snapshot.

In addition, there is an option to interrupt a running snapshot operation unconditionally with *Cancel* control.

### All Objects View mode

***All Objects*** UI presents all imported for analysis snapshots constituents: high-level view. Every top-level node in the tree is browsable either here or within its corresponding section.



*Diagram 4*

Technically, ***All Objects*** UI is a one-for-all entry point retaining the functionality, which is available for every type of object given in a snapshot separately.

### DB Objects View mode

***DB Objects*** UI helps discovering database objects and everything related: objects, identifiers, DDLs, etc. Key property lists complete the picture with finding information on things like partitions, clusters, authentication, parallelism, etc.



*Diagram 5*

The top-level nodes of DB object structure are given on the diagram above: Users, Roles, Schemas, etc.; for more detailed information please refer to the corresponding back-end documentation.

### DB Extensibility View mode

***DB Extensibility*** UI is to serve a very specific purpose related to \*\*\* Banner (please refer to the corresponding document for more details). In fact, this UI is for browsing auxiliary queries of the said system. All the rest is very much the same as in any other section of Studio UI.



*Diagram 6*

### Application Extensibility View mode

***Application Extensibility*** UI helps to review the \*\*\* customers’ extensions they may have as a part of Banner custom pages, including metadata about them. The custom parts belonging to this section are virtual domains, custom pages and customized CSS files.



*Diagram 7*

### File Objects View mode

***File Objects*** UI makes browsable everything scanners may discover on file systems explored: folder, files, symlinks, etc. Please refer to the specific documentation on this type of object to go into more detail. All objects in this section are also presented in a tree-like fashion and have their own property list. Textual content view exists for plain text files with the corresponding syntax highlight.



*Diagram 8*

### Cognos Reports View mode

***Cognos Reports*** UI provides access to Banner predefined and customized reports. The stored list of analytical reports unfolds attributes and definitions, connections to data sources and data models, and their usage stats. The reports are presented in hierarchical form for browsing and reviewing their properties.



*Diagram 9*

### Cron Jobs View mode

***Cron Jobs*** UI deals with the existing cron job inventory on the system in question, including job details, schedules, creation and execution timestamps, plus, their dependencies. The object tree browsable navigation.



*Diagram 10*

### Full Text Search mode

***Full Text Search*** UI is a helper tool to lookup a literal or fuzzily defined term throughout a pre-selected snapshot content at any depth of object nesting.

*NB: Currently this section is under heavy development and may undergo significant changes in the future.*

## SPA Code Base

### Toolkit

The actual development kit of the ***Studio UI*** SPA consists of the following:

* React v18
* RxJS v7
* Typescript v5
* Tailwind CSS v3 (MIT license)
* Webpack v5
* ESLint v8
* Jest v29
* JS Testing library v16
* Monaco editor v4.6 (MIT license)

### Folder Structure

The SPA source code folder tree is ***feature-oriented*** with some sharable inclusions like services, custom hooks, icons, etc.

src/

+---**components**

| +---Badge

| +---ConfirmationModal

| +---SnapshotSelector

| +---SpaHeader

| +---SpaLayout

| +---SpaNavBar

| +---Spinner

| \---Splitter

+---constants

+---customTypes

+---**features**

| +---directory-tree

| +---table

| \---viewer

+---hooks

+---icons

| +---IconAllObjects

| +---IconAnnotation

| +---IconApp

| ...

+---**pages**

| +---AllObjects

| +---AppExtensibilityTree

| +---CognosReports

| +---CronJobTree

| +---DbExtensibilityTree

| +---DbTree

| +---ExtendedSearch

| +---FilesTree

| +---FtSearchUI

| +---Settings

| \---SnapshotsList

+---services

| \---tree-node-service

\---utils

The simplified structure presented above may be read as following:

* Components – SPA layout and overall view.
* Features – those are key components of the Framework
* Pages – high-level sections of the SPA

## Appendices

### **Appendix 1**: Readiness Checklist

|  |  |  |
| --- | --- | --- |
| **#** | **Requirement** | **Status** |
| 1 | SPA portability, i.e. if it is cross-platform | + |
| 2 | Web-/browser-based | + |
| 3 | UI/UIX is user-friendly and intuitive | + |
| 4 | SPA documentation | + |
| 5 | NPMs in use have no vulnerabilities (SNYK DB) | + |
| 6 | SonarQube test coverage | Planned |
| 7 | Performant SPA | + |
| 8 | Maintainable SPA | + |

### **Appendix 2**: Framework Key Components’ and Their APIs

The DS SPA framework has a few key components, but it’s growing. For the time of this writing there are such components on the list:

1. Object Tree Browser component
2. Object Content Viewer component
3. Data Grid component
4. Object Property Viewer component – under refactoring
5. Extended Search component – under development

Each component uses various data adapters – one or more – because of the components’ interactivity and high level of customization.

#### Object Tree Browser

|  |  |
| --- | --- |
| **export type** TreeNode = { id: **string**; elementId: **string**; title: **string**; tooltip?: **string**; icon: ReactNode | **null**; isLeaf: **boolean**; children: TreeNode[]; childrenCount: **number**; hasNextLoad: **boolean**; opened: **boolean**; visible: **boolean**; selected: **boolean**; depth: **number**; data: { [key: **string**]: **unknown**, parentID: **string** | **null**, pagination: Pagination, } | **null**}; | id – element idelementId – snapshot IDisLeaf – leaf or nodetitle – element titletooltip – tooltip text, if anyicon – element iconisLeaf – no children elementchildren – children list, if anychildrenCound – children count, if any hasNextLoad – if more siblings opened – if unfoldedvisible – if element visibleselected – if element selected depth – current element depthdata – element payload |
|  |  |

#### Object Content Viewer

|  |  |
| --- | --- |
| **type** MultiTabStateType = { tabsContentMap: TabContentMapType; allTabsIds: **string**[]; pinnedTabsIds: **string**[]; dynamicTabId: **string** | **null**; activeTabId: **string** | **null**; isLoadingTab: **boolean**;}; | tabsContentMap – tab contentallTabsIds – tab ID listpinnedTabsIds – pinned tab ID listdynamicTabId – current/dynamic tab IDactiveTabIf – tab in focus IDisLoadingTab – tab state |
|  |  |

#### Data Grid

|  |  |
| --- | --- |
| **export type** GridRow = { id: **string**; isExpanded: **boolean**; visible: **boolean**; isChildRow: **boolean**; childRowLast: **boolean**; childrenRows: GridRow[]; cells: GridRowCell[]; value: Omit<TableGridIncomingData, **'children'**>; onExpand?: (e: React.UIEvent<HTMLElement>) => **void**;}; | id – element idisExpanded – if unfolded visible – if element visibleisChildRow – if child rowchildRowLast – if the last onechildrenRows – element listcells – a row cellvale – conditional cell value |
|  |  |