Overview

2022-07-04

# Dependency Injection

All plugin logic should be injected via the host, and interfaces are used to describe the properties and function within the injection. This is to allow an agnostic host, where we can inject the functionality through the Angular website, mock it out in the SDK, or implement 'real' functionality within the SDK or Website.

# Plugin based architecture

Plugins needs to be developed in a 'standalone' manner, where each plugin can run independently of any other, and should work correctly under any host environment. To accurately simulate that environment the SDK implements a PluginHost which all plugins need to be registered to in order to see them in the SDK website.

# Section of work

This describes a single task, or a group of tasks, that share common features an functionality. Any new tasks should first consider what section of work they fall under to understand which conventions to use.

eg. Developing tasks for "Login screens", "Registration screens", "Route guarding", etc, will all fall under the "Authentication" section of work.

# Interfacing Dependency Injection

All injections from the LBO into Vue happen through Dependency Injection (for now - the idea is to have all functionality within the Vue project). In order to create a type-safe solution for TypeScript we need to generate valid interfaces which accurately describe the injection type.

The root interface for dependency injection can be found in the following location: src/core/interface/ScopeInterface.ts

The RootScopeProvideInterface should **only** house secondary provider interfaces that cover a section of work to help with proper abstraction and code modularity.

All secondary provider interfaces should be stored in the following location: src/core/interface/provider

**Updating Root Provide in LBO** ui-network-admin/src/main/resources/static/scripts/app.js

# Mock Dependency Injection

While in development, the SDK has no access to LBO to have data and functionality injected in. This means we need to create mock functions and data as a temporary workaround. All mock data should follow the Interface standards.

The mock data entry point can be found in the following location: src/mock/RootScopeMock.ts

All secondary provider mocks should be stored in the following location: src/mock/provider

# Encapsulation

When creating plugins there may be a time where you need to split out code to its own file, create interfaces, or simply abstract some functionality or styles. The idea of encapsulation is defining where exactly those files should be located relative to the .vue plugin file.

There are two types of plugins you can generate, and the encapsulation definition changes for each;

**Standalone Plugin** A standalone plugin shares no functionality or resources with any other plugins, and therefor needs to external references. Encapsulation is then bound by the plugin structure itself, and all code needed for that plugin should fall within that plugin folder.

**Shared Plugin** Any plugin created that derives from another plugin, or requires shared code from the SDK is known as a shared plugin. While the rules remain the same as a Standalone Plugin, the main difference is that any shared code or interfaces will reside in the /core folder.

# Provider vs Service

A **provider** is primarily an injected piece of functionality that sits within RootScope.provide.\*. It’s mostly used to leverage existing functionality within the LBO which is then simply interfaced into the plugin.

A **service** is primarily a standalone set of functions that can be shared between plugins. This may be created in the SDK or LBO and would always have one defined instantiation which all plugins can reuse.

In most cases you will be working with Providers, as Services usually provide core functionality needed.