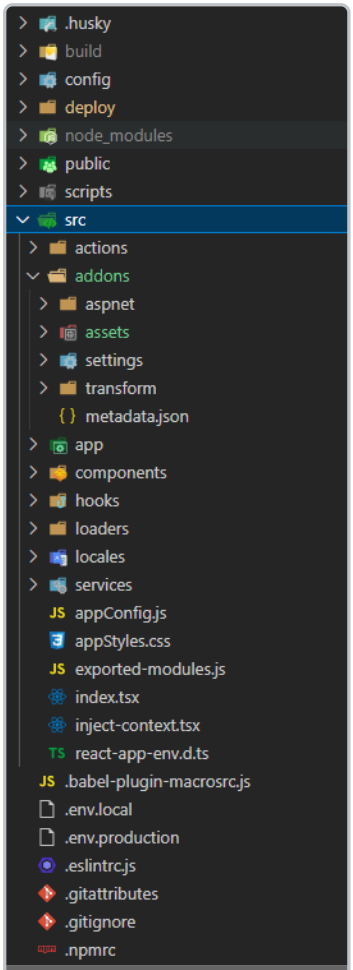
Template Project for Portal Customization

Last Modified on 11/10/2022 1:27 pm EST

Overview

Among other things, the **oneClick_Install_Env_Local.ps1** PowerShell function downloads and unpacks the customization project template. The customization template is a Visual Studio project that enables you to manage back-end features, such as shared resources, configuration transformation, custom C# code, among others and then apply your changes onto the portal application.



Customization template file structure

The unpacked folder structure includes several folders. Some are dedicated for common customization scenarios that don't require React development and others include templates for complex React customization scenarios.

Folders related to common customization scenarios	**Folders related to scenarios that require React development**
/deploy: Contains PowerShell functions for setting up the development environment for portal customization.	**/src/app/modules**: Contains containers ("pages") onto which you add React components. These containers are ultimately your pages, similar to aspx files that contain ascx files without any logic.

Folders related to common customization scenarios	Folders related to scenarios that require React development
/src: Contains code templates for the customizable components.	**/src/components**: Contains a React components folder, where each component is an independent component with single responsibility.

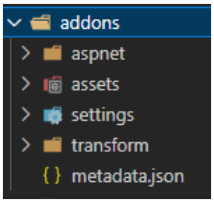
The addons folder

The templates under the **addons** folder are helpful for customization scenarios that do not require React development.

For example:

- Add a custom menu item transformation
- Add other configuration file transformations
- Add a Form Viewer control that loads a custom Cora Orchestration view within an aspx page, which opens as a portal tab
- Add C# code that compiles at runtime
- Add CSS files, fonts, images, and other assets to the Shared Resources folder based on the provided folder hierarchy

Folder location: **/src/addons**



addons folder contents

Folder	Contents
/aspnet/app_code	Contains custom **C#** code. Only ***.cs** files are copied into the customized application and compiled on demand at runtime.
/aspnet/pages	Contains custom **ASP.NET** pages. Only ***.aspx** files are copied into the customized application's root folder.
/assets	Contains **static and binary** contents that are copied as is into the customized application's root folder.
/settings	Contains **config files and resources** that are copied into the customized application's root folder.
/transform	Contains **partial XML configuration files** that are applied onto existing configuration files according to the Shared Resources hierarchy and copied to the customized application's root folder.

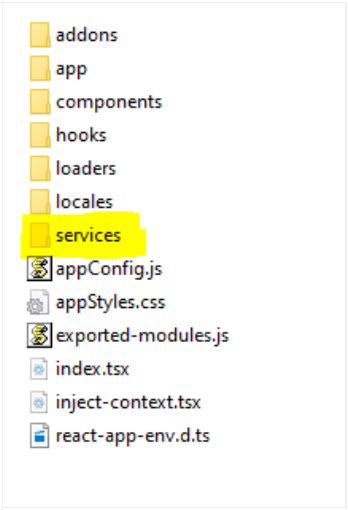
Folder	Contents
metadata.json	Contains the customization package name and version. This file is used by the application build scripts.

IMPORTANT
All the files and folders that you add to the **addons** folder keep their original file folder structure after they are copied to the customized application.
For example, if you add a 'mycustom' folder with an aspx file to **/aspnet/pages/custom/mypage.aspx**, then the path in the customized application looks like this: **inetpub\wwwroot\Cora SeQuence\Flowtime 1\Custom\mypage.aspx**.

The services folder

Application services are singletons and do not hold a state. They mainly contain common and shared functionality for daily usage, such as loggers, managers, and web clients, among others.

Folder location: **/src/services**



Available services

Folder	Description
client-service	Performs HTTP requests to API and web servers.
component-service	Renders React components from a URL.
config-service	Manages application config files.
context-service	Handles application session context.
inject-service	Allows dependency injection.

Folder	Description
log-service	Displays log messages on the browser console.
navigation-service	Handles application navigation between known and registered routes.
route-service	Manages components and routes.
shared-code	Shares code between the portal application and the customized application.
side-effect-service	Handles side effects and events.
state-service	Manages Redux store and application states.
translation-service	Enables the use of application localization resources.

Service usage example:

```
1  import { LogService } from 'services/log-service';
2
3  const AddWorkflow = (props) => {
4    LogService().log.Error('Some error occured');
5  };
```

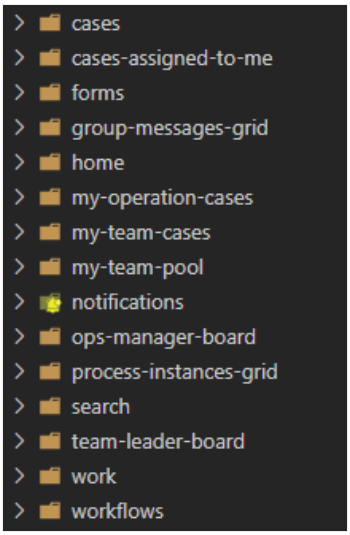
Line 1: Import service
Line 4: Call for service functionality

The modules folder

Application modules are similar to HTML pages, but in component-based React, they are called modules. A module represents a separate business logic. Each module can contain multiple inner components.

For example: **notifications** is a module that has Title, Grid, Search, and Filter components and performs HTTP requests to receive information and bind it into its components.

Folder location: **/src/app/modules**



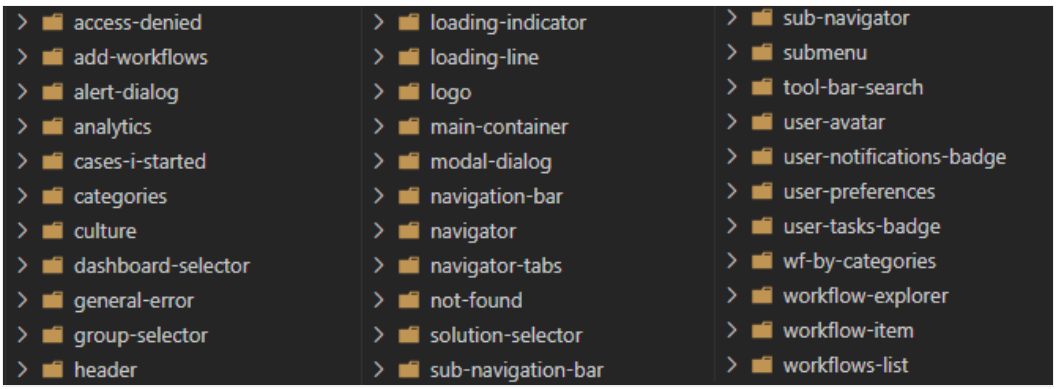
The components folder

Application components are a lower-level logic or UI representation code that can be reused multiple

times in different places without the need to copy paste. They act like classes in object-oriented languages. Each rendered component has its own instance reference.

For example: The Menu component has a few child components of type Link. Each Link component has its own text and navigation URL.

Folder location: **/src/components/common**



Component usage example:

```
1  import Logo from '../logo/';
2
3  const Header = props => {
4    return (
5      <div>
6        <Logo />
7        <ul>
8          <li>Home</li>
9          <li>About</li>
10         <li>Contact</li>
11       </ul>
12     </div>
13   );
14 };
```

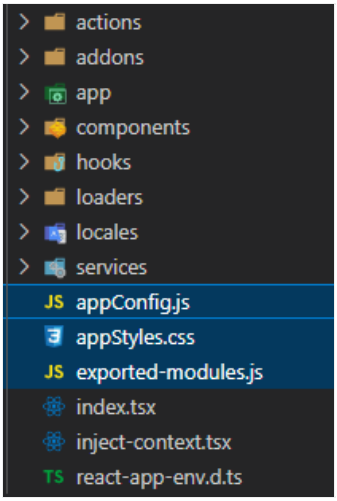
Line 1: Import component

Line 6: Render component

Global configurations

You can set global configurations and apply them to the main portal application.

Global configuration file location: **/src** (root level)



Global configuration JSON files

appConfig.js	Contains custom configuration that is applied to the main portal application at runtime.
appStyles.css	Contains custom styles that are applied to the main portal application at runtime.
exported-modules.js	Contains information about the customized components. The information is applied to the portal application during application build and packaging.