Table of Contents

Basic Information

The Pega React Starter Pack application provides sample code which illustrates leveraging Version 1 of the [Pega Digital Experience (DX) APIs] (https://community.pega.com/digital-experience-api) to build a custom React front-end experience for Pega applications from which users can view, create, and update Pega cases and assignments.

It implements a simple case worker portal which should work against most simple Pega applications. It currently leverages [Semantic UI React] (https://react.semantic-ui.com/) as its UI component library.

The starter pack also provides an example of an embedding experience where a Pega case is embedded within a React self-service portal experience at some point during the interactions. For more information, see the Embedding example and configuration section.

A front-end developer will need to install the React Starter Pack on their local desktop computer and configure the source code to access either a simple Pega application (or you can deploy and leverage the provided CableConnect sample app).

Development Environment

Node.js and NPM are critical for installation and execution of the Pega React Starter Pack. NVM is optional, but heavily recommended. See https://docs.npmjs.com/downloading-and-installing-node-js-and-npm for info on installing Node.js and NPM and also discusses NVM.

This application was tested using:

- Node.js version: 16.13.2
- NPM version: 8.1.2
- NVM version: 0.37.2

Installation Instructions

1. Retrieve the latest Pega React Starter Pack zip file from Pega Community (https://community.pega.com/marketplace/components/react-starter-pack).

2. Unzip the PegaReactStarterPack87.zip, it will create a PegaReactStarterPack87 directory that contains the following sub-directories:
   - `React App` - source code to allow building and executing the Pega React Starter Pack app
   - `Documents` - contains this README file and the instructions for installing the CableConnect Pega sample app
   - `CableConnect App` - contains the zip file to import from Pega Dev Studio to install the CableConnect Pega sample app
   - `Repeat App` - contains the zip file to import from Pega Dev Studio to install the Repeat Pega sample app
3. Move `pega-react.zip` from `React App` directory into a directory of your choosing and unzip it.
   - This will create a subdirectory called `pega-react`
4. Execute the following commands from within the `pega-react` directory within a terminal window (or command prompt):
   - `npm install`
   - This will retrieve all the dependent modules required by the Pega React Starter Pack application.
   - WARNING: If npm install encounters errors, it might be related to an incompatibility with the version of npm you are using and the package-lock.json file. Deleting the package-lock.json file and then running npm install again will typically resolve this issue.
   - WARNING: DO NOT use `npm audit fix --force` to try to fix vulnerabilities enumerated during a npm install. This has been known to frequently make a breaking change by lowering the version of react-scripts within package.json.
5. Setup the CableConnect Pega sample application by following the instructions within CableConnectSampleApp.pdf (located within the Documents sub-directory)
6. Make sure that your Pega application server and Pega application are setup properly. See Pega Setup considerations
7. Review the Pega server connection information within the file `pega-react/src/_services/endpoints.js`. See Application Configuration section for more details on updating this to point to the Pega server which has been properly configured for DXAPI access.
8. Execute the following commands from within the `pega-react` directory within a terminal window (or command prompt):
   - `npm start` OR `npm run starthttps`
9. This should open your browser to http://localhost:3000 (OR to https://localhost:3000), which is where the application will be served. If you don't see the browser open automatically, look at the output from npm start and enter the provided url into a browser.

Basic Usage

By default, the starter pack application is configured to access a local Pega server http://localhost:1080/prweb. To login to the CableConnect sample application, you can use any of the following credentials: _ operator: rep.cableco, password: pega - case worker

- operator: customer.cableco, password: pega - case worker _ operator: tech.cableco, password: pega – case worker _ operator: manager.cableco, password: pega – case manager _ operator: admin.cableco, password: pega – developer/admin

Once logged in, you can create cases from the CaseType list, open WorkObjects from the WorkList, and perform end-to-end flows, based on the data returned from the API.

WARNING: When importing the CableConnect sample application, we recommend checking the "Enable advanced mode to provide more granular control over the import process" option on the 2nd import screen. This will eventually provide a checkbox to enable the imported operators. If "Enable advanced mode to provide more granular control over the import process" checkbox is not checked, the imported operators will likely be disabled by default. Also, the Pega server may be configured in a manner that you may be required to change the operator's password before using that operator. On a non-Production Pega development server, one way to stop this forced password change from happening is by using DEV STUDIO and navigating to Records/SysAdmin/Authentication Service and edit the "Platform Authentication" record. Within the "Security Policies" tab, the Password policy record may be removed. Remember to save the record after making such a change. Removing the security policy from This will stop Pega Infinity from forcing a password change on the first login.

Application Configuration

The CableConnect example Pega application RAP includes a OAuth 2.0 Client Registration record named "CableCoV1Oauth". The default endpoints.js file is configured by default with the client id value within that record. If you wish to access the sample app with OAuth, remember to set use_OAuth property within endpoints.js to `true`.

If importing the CableConnect application onto a 8.7 or better server, also enable refresh token functionality for the "Authorization code" grant flow within the "CableCoV1Oauth" OAuth 2.0 Client Registration record. This is a newer capability available for Public client registrations with Infinity 8.7 and better servers, and allows for access tokens to expire more rapidly and be silently refreshed with the refresh token.

If you want to configure the React Starter Pack application to access a different server follow these steps:

- Open `pega-react/src/_services/endpoints.js` and modify the `PEGAURL` property to the value appropriate for your desired system.
- If using a Pega server prior to 8.5, set use_v2apis to `false`
- Decide whether you wish to use Basic or OAuth with the starter pack.
- Configuring to use Basic authentication:
  - Set use_OAuth to `false`
  - Using Dev Studio:
    - confirm the "api" and "application" Service Package's "Authentication type" is set to "Basic".
    - Note: Setting the "application" Service Package Authentication type to Basic IS NOT compatible with leveraging Constellation design system applications. It is heavily recommended to Configure the starter pack to use OAuth or to disable the use_v2apis (disable screen flow capability)
- Configuring for OAuth (preferred):
  - Set use_OAuth to `true`
  - If using a 8.7 or better Pega server, set use_revoke to `true`
  - Using Dev Studio, confirm the "api" and "application" Service Package's "Authentication type" is set to "OAuth 2.0"

- Create an OAuth 2.0 Client Registration record for this Starter Kit app (if CableConnect app was not imported with the OAuth 2.0 Client registration record named "CableCoV1OAuth"):
    - Create a new "Security/OAuth 2.0 Client Registration" record for this app
    - You might name it "PegaReactSPA" for both the Short description and the Client Name.
    - Specify "Public" for the type of client (as browser apps are not able to prevent any "Client secret" from being compromised)
    - Select "Authorization Code" for the Grant type
    - Add a RedirectURI value based on the url used to access the deployed Pega React Starter Pack Application (e.g., http://localhost:3000/auth)
        - Note: The "auth" route in the path is significant for this sample app
        - If you want to have a local http and https version of the app running, you will have to create and configure an additional RedirectURI for the additional initial url you want OAuth to redirect to after the authentication succeeds.
    - Make sure "Enable refresh token" is enabled (this option is not available for 8.6 and earlier servers)
    - Enable the "Enable Proof Key for Code Exchange" option
    - Review and possibly adjust "Token expiry settings
        - When Refresh Tokens are not available and enabled, the "Access token lifetime" determines how long the user may keep using the application without encountering another visible authentication challenge.
        - When Refresh Tokens are enabled, the "Refresh token lifetime" determines how long the user may keep using the application without encountering another visible authentication challenge, and the "Access token lifetime" can be a much shorter duration value.
- Enter the appropriate values (within endpoints.js) for client_id from the appropriate client registration record
    - Set loginExperience to either `loginBoxType.Main` or `loginBoxType.Popup`

- Ensure that your desired end user operator access group includes the :PegaAPI role.

Pega Setup considerations

This section covers some important Pega application and Pega server setup/configuration requirements for properly using this starter pack against a particular Pega server.

- When you login as a particular operator, the default application access group specified for the operator will be utilized. This access group must contain the "PegaAPI" role to allow proper access to the Pega DX API. (This is also mentioned within the CableConnect sample application documentation.) Typically, :PegaAPI role would be added to the application access group and that role will have PegaRules:PegaAPI specified as a dependent role.

- The Confirm harness/page has to be explicitly available within the application as the default one will not currently work with the DX API.

- The API Service Package record needs to have the proper Authentication mechanism configured to permit proper access (either Basic or OAuth) and the Starter Pack app needs to be configured to match. See Application Configuration.

- The Integration/Services/Endpoint-CORS policy mapping much be configured properly so that the "api/" endpoint is configured to utilize the "APIHeadersAllowed" CORS policy.

Embedding example and configuration

The React starter pack's embedded experience enables you to embed a Pega Case experience within another React-based application. This implies that you can use components provided by the starter pack to display the case-related UI within the host application. The current example is based on the sample Pega application (CableConnect) provided with the full starter pack package.

To configure an embedded application, modify the following EMBEDCFG settings within the endpoints.js file:

- *caseType*: The case type instance to create.
- *userIdentifier*: The operator to utilize when creating the case.
- *password*: Base-64 encoded password for operator specified by the userIdentifier.
- *passContent*: true or false value to indicate whether or not to pass content based on the selection within the prior screen.

Access the embedded version of the application using the following URL:

```
http://localhost:3000/embedded
```

Note: This example also posts some content specific to the case as part of the package selected from the self-service portal page. If utilizing another Pega application or case type, then those structures within src/_apps/Embedded/**Embedded.js** should be modified as well (or set EMBEDCFG.passContent to *false* to disable the content from being posted).

Creating a production build to deploy on an HTTP web server

This section covers how to build a production build and also outlines how to build it such that it may be deployed on some specific path within your hosting web server (e.g, /PegaReact).

Make sure the **endpoints.js** file is setup and configured properly to reference your Pega server.

The homepage property within **package.json** is initially "/". It should be modified to reference the desired deployment application path (e.g, "/PegaReact"). Do not specify a trailing "/" when specifying an alternate deployment path.

The npm run build command will build an optimized production build within the /pega-react/build output directory.

If OAuth is configured, specify the full path to the application as an additional redirect URL within the OAuth 2.0 Client registration record used for the application. For example, if the app is deployed on a server named lab001.foo.com, then the redirect URI specified for a **PegaReact** chosen app name would be https://lab001.foo.com/PegaReact/auth.

The files within the build directory may now be transferred to the appropriate web server path using sftp or alternate tooling used to transfer files to the web server.

To simplify deployment to any web server, the **postbuild** script within package.json currently includes the creation of directories to correlate to the same client-side routes supported by the application and the copying of the built root **index.html** file to these directories as well. An alternate (and more complex approach) is to not create the subdirectories which correlate to the client-side routes utilized and instead make web server-specific changes to serve up the root **index.html** page when unknown paths are encountered at the server. For example, for Tomcat servers, the Rewrite Valve may be enabled and an explicit rule specified for this app. This would involve 2 steps.

1. Enable *RewriteValve* by adding the following XML within the **<Host name="localhost">** XML element within the **conf/server.xml** config file:

```
<Valve className="org.apache.catalina.valves.rewrite.RewriteValve" />
```

2. Add , or if present, modify the **conf/Catalina/localhost/rewrite.config** file for this specific deployment:

```
RewriteCond %{REQUEST_PATH} !-f
RewriteRule ^/PegaReact/(.*) /PegaReact/index.html
```

If employing such a web server based rules approach, the **postbuild** script may be eliminated within the **package.json** file.

Some deployment related references with info on other web servers:

- Create-React App: Deployment
- React Router Configuration - Apache Http Server, Nginx, Tomcat

Testing the application

You can test rep.cableco scenario by executing the following commands in the terminal:

1.
   ```
   $ npm start (and leave it running)
   ```
2. Open a different terminal window or tab (since start is still running)
3. **Executing the tests**:

   ```
   $ npm run test
   ```
4. **Getting the test report**:

   To get the test report of last run:

```
$ npm run test-report
```

> **NOTE**: These tests execute the sample **CableConnect** application.

Application Settings within the Settings Dialog

There are several settings presently within the Settings dialog which are present to illustrate alternate sequences of API usage

- `Use Page Instructions for Embedded Pages`
  - When checked generates pageInstructions content for all fields within Embedded Pages
- `Use Page Instructions for Page Lists/Page Groups`
  - When checked generates pageInstructions content for all fields within repeating Page Groups and Page Lists
- `Autocomplete/Dropdown use local options for Data Page`
  - When not checked, the options which are populated within the field structure (since Pega Infinity 8.3+) are ignored and separate GET /data/ endpoint transactions result to retrieve the options for the field
- `Autocomplete/Dropdown use local options for Clipboard`
  - When not checked, the options which are populated within the field structure (since Pega Infinity 8.3+) are ignored and the data is retrieved from the appropriate "content" portion from the earlier retrieve data from the GET /cases/{ID} post
- `Save assignment (preferred) (vs. Save Case)`
  - When checked, invokes POST /assignments/{ID} rather than PUT /cases/{ID}
  - The POST assignments endpoint is available since Pega Infinity 8.4 and offers additional validation against the flow action properties.
- `Screen flow`
  - Indicates if the endpoints use_v2Apis is enabled or not. When enabled, the application is able to properly display the Previous button for Muti-step form steps
    - Note: With the CableCo sample app, the customer.cableco operator is the one which has a screen flow example
- `Show Case Details area`
  - When checked, will display the Case Details area on the right of the work object screens
- `Show Save action`
  - When checked, will offer a Save acton button within the work object screens
- `Show Workgroup Workbaskets`
  - When checked, will include workbaskets within the current logged in operator's workgroups
- `Create Case Starting Fields (must be JSON-compliant)`
  - Used only by case types within more complex apps like Customer Service, which require additional context to be specified when invoking the case type. Leave blank otherwise.

Application Structure

If you are familiar with React / Redux, then many of the components and classes will be straightforward.

```
PegaApp/
  README.md
  docs/
  node_modules/
  package.json
  public/
  src/
    _actions/
    _apps/
      AppSelector/
      Embedded/
      PegaApp/
    _components/
    _constants/
    _helpers/
    _reducers/
    _services/
    _styles/
    AppHeader/
    assets/
    Dashboard/
    DashboardWidget/
    IframePage/
    LoginPage/
    PegaForm/
    SilentPage
    Workarea/
    Worklist/
    WorkObject/
```

Some of the most important directories and files are highlighted below:

`_actions/`

These files contain action creators, used to dispatch actions via Redux.

There are separate files for actions related to:

- Alerts
- Assignments
- Cases
- Errors
- Users
- Workqueues

`_reducers/`

Redux reducers. Used to update state in the store after actions are dispatched.

There are separate reducers for:

- Alerts
- Assignments
- Cases
- Errors
- Users
- Workqueues

`_services/`

Functions used to issue AJAX requests and manage responses. All of the included methods use the Axios library for Promise-based requests.

There are separate service files for:

- Assignments
- Cases
- Data Pages
- Users

`PegaForm/PegaForm.js`

This is a React component used to generate forms for assignments, views, and pages based on data returned from the Pega API. Form generation for assignments, views, and pages are all based on a nested UI data structure returned from the API.

- Views / pages contain groups.
- Each element of a Group array can contain a view, layout, field, paragraph or caption.
- Layouts determine the UI structure of the form.
    - Supported layouts (layout.groupFormat values) are:
        - Dynamic (Simple layout)
        - Grid (Table)
        - Stacked
        - Inline middle
        - Inline grid double
        - Inline grid double (70 30)
        - Inline grid double (30 70)
        - Inline grid triple
    - Additional supported structural components:
        - Repeating (Dynamic) layout
        - Repeating (Grid) row
        - Embedded Section
    - Unsupported layouts and structural components include:
        - Layout group
        - Column layout
        - Dynamic layout group
        - Hierarchical table
        - Navigational tree
        - Dynamic container

        - AJAX container
        - Non-auto generated sections
        - Sections that include scripts
        - Non guardrail-compliant sections
- Fields contain information about the property, including reference, current value, outstanding validation messages, and attached actions.
- Supported fields:
    - pxTextInput
    - pxDropdown
    - pxCheckbox
    - pxTextArea
    - pxURL
    - pxEmail
    - pxDateTime
    - pxInteger
    - pxNumber
    - pxPhone
    - pxDisplayText
    - pxHidden
    - pxButton
    - label
    - pxLink
    - pxIcon
    - pxRadioButtons
    - pxCurrency
    - pxAutoComplete
- Supported actions:
    - setValue
    - postValue
    - refresh
    - takeAction
    - runScript
    - openUrlInWindow
    - localAction (replaceCurrent and modalDialog)

PageGroups and PageLists are supported.

When changing values on the form (checking a checkbox, typing into an input, etc...) the changes in value are reflected in state.values for PegaForm. When doing a POST to submit an assignment or refresh fields, the state.values object is translated into a nested representation of the data based on page structure, and sent to the server. Note: When Page Instructions are enabled, fields within constructs that are represented by Page Instructions are sent within the pageInstructions as a sequence of directives to play back (rather than as specific values in the content)

`_helpers/ReferenceHelper.js`

- Class to handle translating Pega's fully qualified property paths to a nested Object structure, and vice versa.
- Also some utility methods for:
  - Handling initial PegaForm state given View from API
  - Finding correct PageGroup/List based on property reference
  - Getting blank entry for PageGroup/List when adding new element
- When posting data to the server via API, it must be nested.
- When retrieving field information from the server, the property paths are flat and qualified.

`_components/DataPageDropdown.js`

This is an example of a self-contained React component that has some Pega-API-specific logic. In this case, the DataPageDropdown creates a dropdown form element that sources its options from a DataPage.

`_components/PegaAutoComplete.js`

An example of using a function component. Implements the auto complete field logic, and also provides an example of leveraging DXAPI custom attributes.

`_app/AppSelector`

Establishes main routes to either render the sample PegaApp or Embedded app.

`_app/Embedded`

Example of a self-service page which then instantiates a WorkObject component related to having created a new case. It can be reached with a route to "/embedded" (such as http://localhost:3000/embedded)

`_app/PegaApp`

Example of a Pega Case Worker portal which displays a worklist and allows opening up various cases/assignments within separate managed tabs. It can be reached with any route other than "/embedded"

Other Resources

- This project was bootstrapped with Create React App.
- Semantic UI React
- Pega Digital Experience (DX) API Overview
- Traditional Starter Packs Documentation
- Supported Features within Starter Packs
- Info on which Pega application layouts are supported by DX API
- Troubleshooting the Starter Packs
- Pega API topic within Pega 8.5 Help File