IBM Cúram Social Program Management 8.0.2

IBM Social Program Management Design System 6.0.0

Note

Before using this information and the product it supports, read the information in Notices *(on page liii)*

Edition

This edition applies to IBM Cúram Social Program Management 8.0.2.

Contents

Chapter 1. IBM Social Program Management Design System

You can use the design system to develop your own custom web applications in addition to the standard IBM® Cúram Social Program Management web client. The design system provides the foundational packages for building accessible and responsive web applications. It consists of a React UI component library, React development resources, and a style guide for creating web applications.

The design system incorporates the US Web Design Standards and also supports additional CSS, utility classes, and a layout framework to enable teams to quickly build Section 508 compliant, responsive, and production-ready web applications.

Documentation versions

The online documentation applies only to the most recent version of the design system. To read the documentation in PDF format for earlier versions, see the IBM Cúram Social Program Management PDF library.

Chapter 2. IBM Social Program Management Design System release notes

Read about enhancements and bug fixes in the IBM Social Program Management Design System.

For more information about changes that depend on server-side updates, see the release notes for your specific version of IBM Cúram Social Program Management at https://www.ibm.com/support/docview.wss?uid=swg27037963.

For more information about compatibility with IBM Cúram Social Program Management versions, see Prerequisites and supported software *(on page 8)*.

6.0.0 (15 December 2022)

No changes in the IBM Social Program Management Design System for this release.

Chapter 3. Prerequisites and supported software

Before you install or upgrade, review the prerequisites and supported software to ensure compatibility.

IBM® Cúram Social Program Management Platform

IBM® Cúram Social Program Management Platform (SPM) is a prerequisite for developing and deploying your web application.

The IBM Social Program Management Design System asset is released at more frequent intervals than SPM and requires specific SPM versions to benefit from the most recent server-side enhancements and bug fixes.

Note:

- From IBM Social Program Management Design System 5.0.0 onwards, new features, server-side enhancements, and defect fixes are supported only in the most recent IBM Cúram Social Program Management version lines. Security fixes and defect fixes are supported on IBM Cúram Social Program Management 7.0.10-7.0.11.
- The IBM Social Program Management Design System 3.x.x version line continues to be supported for security updates and critical defect fixes only on the older compatible version lines of IBM Cúram Social Program Management, 7.0.10 -7.0.11.
- The IBM Social Program Management Design System 2.6 version line continues to be supported for security updates and critical defect fixes only on the older compatible version lines of IBM Cúram Social Program Management, 7.0.4 -7.0.9.

For more information about the support strategy, see Citizen Engagement support strategy *(on page 50)*.

Note:
From October 2019, new features, and server enhancements and defect fixes, are delivered only in the most recent version line.

Table 1. Compatibility with Social Program Management

A list of the asset versions and their compatible Social Program Management versions.

Asset versions	Compatible Social Program Management versions
6.0.0 5.3.2 5.3.1 5.3.0 5.2.2 5.2.1 5.2.0 5.1.0	• 8.0.2 for all new features, enhancements, and defect fixes. • 7.0.10-7.0.11 for security fixes and defect fixes.
5.0.0	• 8.0.1 for all new features, enhancements, and defect fixes. • 7.0.10-7.0.11 for security fixes and defect fixes.
4.1.4 4.1.3 4.1.2 4.1.1 4.1.0	• 8.0.1 for all new features, enhancements, and defect fixes.
4.0.3 4.0.2 4.0.1	• 8.0.0 for all new features, enhancements, and defect fixes.

Table 1. Compatibility with Social Program Management

A list of the asset versions and their compatible Social Program Management versions.

(continued)

Asset versions	Compatible Social Program Management versions
4.0.0	
3.0.8 3.0.7 3.0.6	• 7.0.11 iFix 5 for essential maintenance, security updates and critical defect fixes. • 7.0.10 iFix 8 for essential maintenance, security updates and critical defect fixes.
3.0.5 3.0.4	• 7.0.11 iFix 3 for all new features, enhancements, and defect fixes. • 7.0.10 iFix 7 for essential maintenance, security updates and critical defect fixes.
3.0.3	• 7.0.11 iFix 3 for all new features, enhancements, and defect fixes. • 7.0.10 iFix 6 for essential maintenance, security updates and critical defect fixes.
3.0.2	• 7.0.11 iFix 2 for all new features, enhancements, and defect fixes. • 7.0.10 iFix 5 for essential maintenance, security updates and critical defect fixes.
3.0.1 3.0.0	• 7.0.11 iFix 1 for all new features, enhancements, and defect fixes. • 7.0.10 iFix 4 essential maintenance, for security updates and critical defect fixes.
2.9.1	• 7.0.11 for all new features, enhancements, and defect fixes. • 7.0.10 iFix 3 for essential maintenance, security updates and critical defect fixes.
2.9.0	

Table 1. Compatibility with Social Program Management

A list of the asset versions and their compatible Social Program Management versions.

(continued)

Asset versions	Compatible Social Program Management versions
2.8.6	• 7.0.10 iFix 3 for all new features, enhancements, and defect fixes.
2.8.5 (Including the 2.8.4 internal release)	
2.8.3	
2.8.2	
2.8.1	
2.8.0	
2.7.0	

Node.js

Node.js is a prerequisite for installing the React application and for developing and deploying your web application.

Compatible Node.js versions.

Supported software	Version	Prerequisite minimum	Operating system restrictions
Node.js	16 LTS (latest) 14 LTS	14.16.0 LTS (Fermium)	No

Note:
By default, Node 16 uses Node Package Manger (npm) 7. To use this configuration, you must specify the npm option `legacy-peer-deps` in your project. The way that npm 7 treats peer dependencies changed. The Universal Access Responsive Web Application is using the `legacy-peer-deps` option as a temporary fix while we work to remove this constraint. For more information about `legacy-peer-deps`, see npm Docs. The following steps outline how to configure the `legacy-peer-deps` option:

1. Create a `.npmrc` file at the root of your project.
2. Add the `legacy-peer-deps=true` content to the file.

Application server, web server, and database

Deploying the web application requires a web server in the IBM Cúram Social Program Management topology. The following application server, web server, and database combinations are supported for developing and deploying your custom application.

- IBM® WebSphere® Application Server, IBM® HTTP Server or Apache HTTP Server, and IBM® Db2®
- IBM® WebSphere® Application Server, IBM® HTTP Server or Apache HTTP Server, and Oracle Database
- Oracle WebLogic Server, Oracle HTTP Server or Apache HTTP Server, and Oracle Database

For more information about installing an application server or database for SPM, see Installing prerequisite products.

HTTP servers

These HTTP servers are supported for deployment.

Compatible HTTP server versions

Supported software	Version	Prerequisite minimum	Operating system restrictions
IBM® HTTP Server	9.0	9.0.0.5	No
	8.5.5	8.5.5.9	No

Supported software	Version	Prerequisite minimum	Operating system restrictions
Oracle HTTP Server	12.2.1.3.0 and future fix packs	12.2.1.3.190808	No
Apache HTTP Server	2.4 and future patches	2.4	No

Web browsers

The following browsers are supported for developing and deploying your web application.

New versions of Chrome, Firefox, Edge, and Safari are released more frequently than Internet Explorer, and updates are installed automatically by default for these browsers. Design system releases are tested on the latest versions of the browsers that are available at the start of IBM's development cycle;

Note:
Only stable Chrome releases are tested.

If a browser is tested and no issues are found, IBM® certifies that version. The prerequisites advise the version that is certified at each new product release. If IBM® cannot certify a version, you might need to revert to a previous, fully certified version. While IBM® supports customers with newer versions of the browsers than the last certified version, customers must know that those versions of the browsers are not fully tested.

Supported software	Version	Prerequisite minimum	Operating system restrictions
Apple Safari	14 and future fix packs	13	No
Google Chrome	91 and future fix packs	80	No
Microsoft™ Edge	91 and future fix packs	44	No
Microsoft™ Internet Explorer	11 and future fix packs	11	No

Supported software	Version	Prerequisite minimum	Operating system restrictions
Mozilla Firefox	89 and future fix packs	73	No

Accessibility

This accessibility software is supported.

Supported software	Version	Prerequisite minimum	Operating system restrictions	Browser
Freedom Scientific JAWS screen reader	2020 and future fix packs	2020	No	Microsoft™ Internet Explorer 11
Apple VoiceOver	Any version and future fix packs	Any version	Any version	Microsoft Edge and JAWS 2020 is the only certified screen reader and browser combination.

Note:
The combination of Internet Explorer 11 and JAWS 2019 is the only certified screen reader and browser combination.

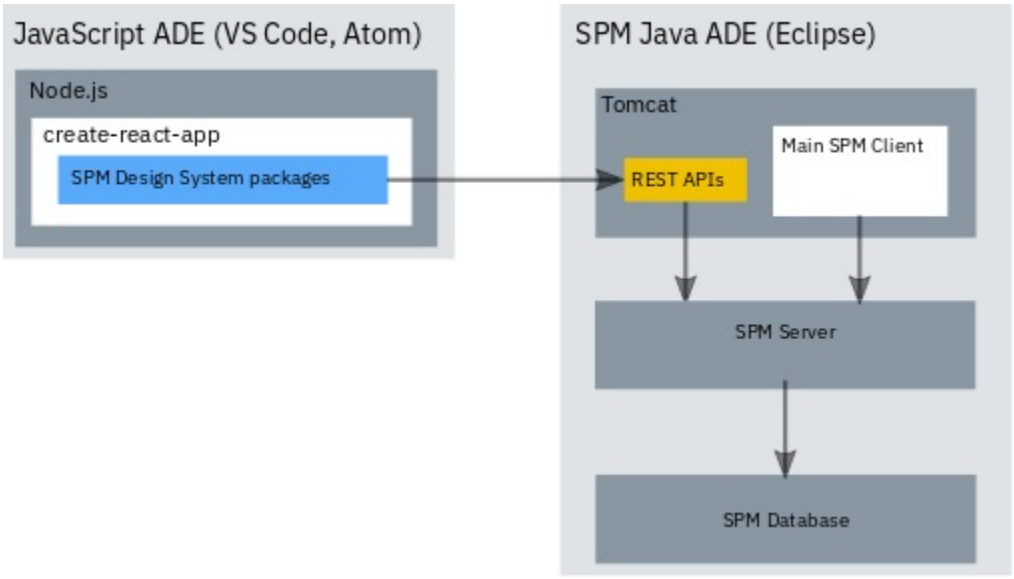
Previous versions

To see the prerequisites and supported software for previous versions, see the IBM Cúram Social Program Management PDF library.

Chapter 4. Installing and getting started with the design system

To get started quickly, install Node.js, install a React application, install the Social Program Management Design System Node packages, and complete the short tutorial. When it is time to develop and test your REST APIs, install the IBM Cúram Social Program Management Java Application Development Environment (ADE).

You need a React application into which you can install the Social Program Management Design System node packages. You can use any React application. However, the Facebook `create-react-app` contains some useful tools that simplify getting started with React development.



The Social Program Management Design System consists of the following Node packages:

- `@spm/core`
- `@spm/core-ui`
- `@spm/core-ui-locales`
- `@spm/intelligent-evidence-gathering`
- `@spm-intelligent-evidence-gathering-locales`
- `@govhhs/govhhs-design-system-core`
- `@govhhs/govhhs-design-system-react`
- `@spm/eslint-config`
- `@spm/test-framework`
- `@spm/web-dev-accelerator-scripts`
- `@spm/web-dev-accelerator`

Attention:
When working with npm packages, it is important that you familiarize yourself with the npm ecosystem and how package dependencies work, so that you can adopt a suitable security strategy for your project needs.

1. Download and install Node.js from https://nodejs.org. The installation includes the npm (Node package manager), which you can use to install your Node packages.
   For more information about Node.js, see this Node.js developerWorks article.
2. Create your React application by using Facebook's `create-react-app`:

   a. From the directory where you want to create your React application, enter the following command, where *`my-app`* is the name that you want to call your application. A *my-app* directory is created to contain the application files.

   ```
   npx create-react-app my-app
   ```

   For more information about `create-react-app`, see the create-react-app user guide.

   b. You now have a basic React application in the *`my-app`* directory. You can run the application by entering the following commands:

   ```
   cd my-app
   npm start
   ```

   If the local host does not start automatically, browse to http://localhost:3000/ to see the running application.

3. Download the IBM Social Program Management Design System Node packages. Open the Merative Support Community, under **Software Downloads**, select **Go to Downloads**, and follow the instructions to download the `SPM_DS_<version>.zip` archive file. Then extract the packages in the archive file to any directory.
4. Install and configure the IBM Social Program Management Design System Node packages.

   a. From your *`my-app`* React application directory, enter the following command for each design system package to install the packages.

   ```
   npm install <path>/<package-name>-<version>.tgz
   ```

   Where *`<path>`* is the download path, *`<package-name>`* is a downloaded package, and *`<version>`* is the package version.

> **Note:**
> Ignore any Node package dependency warnings for now. If needed, you can resolve them later.

b. To import the minified JavaScript and CSS files for the design system components, edit the *my-app*`\src\App.js` file and insert the following lines:

```
import '@govhhs/govhhs-design-system-core/dist/js/@govhhs/govhhs-wds.min';

import '@govhhs/govhhs-design-system-core/dist/css/govhhs-wds.min.css';
```

5. Now, you can do the short tutorial to get started, or you can install the IBM Cúram Social Program Management Java ADE that you need to develop and test your REST APIs, and do the tutorial later.
   a. Complete the short tutorial to get started, see Tutorial: Creating a page in your application *(on page 19)*.
   b. Install the IBM Cúram Social Program Management Java ADE, see Installing a Development Environment.

Chapter 5. Upgrading the design system

The design system is released on a frequent schedule. To upgrade to a new version, replace the Social Program Management Design System Node packages in your React application with the newer versions. Only the most recent version and two previous versions are supported. Ensure that you are reading the documentation for your version of the design system.

1. Download the IBM Social Program Management Design System Node packages. Open the Merative Support Community, under **Software Downloads**, select **Go to Downloads**, and follow the instructions to download the `SPM_DS_<version>.zip` archive file. Then extract the packages in the archive file to any directory.
2. Install the IBM Social Program Management Design System Node packages. From your React application directory, enter the following command for each design system package to install the packages.
   In the command, *<path>* is the download path, *<package-name>* is a downloaded package, and *<version>* is the package version.

   > **Note:**
   > Ignore any Node package dependency warnings for now. If needed, you can resolve them later.

   ```
   npm install <path>/<package-name>-<version>.tgz
   ```
3. Run the following command to install the package dependencies.

   ```
   npm install
   ```
4. Complete any upgrade steps in the IBM Social Program Management Design System release notes *(on page 7)*.

Chapter 6. Developing your web application

Create your web application with the help of the development resources that are included in the IBM Social Program Management Design System.

Tutorial: Creating a page in your application

In this short tutorial, you update the create-react-app `App.js` file in your React application to build an application page with the Social Program Management Design System components. For more information about how to design and use Social Program Management Design System components, see the Storybook documentation in `@govhhs/govhhs-design-system-react/doc/index.html`.

1. Let's run the application. While it is running, you can see your changes as you make them. Run the application by entering the following commands:

   ```
   cd my-app
   npm start
   ```

   Browse to http://localhost:3000/ to see the running application.
2. Edit *`my-app`*`/src/App.js`.

First, let's add a header, which includes a nav bar, logo, and the agency name. For the header to load, we must import four components.

- Header
- Primary Navigation
- NavigationLink
- Link

3. We need to import the image for the header logo, so add the following line after the JavaScript and CSS imports lines we added during installation.

   ```
   import logo from '@govhhs/govhhs-design-system-core/dist/img/logo-mark.svg';
   ```

   If a duplicate `import logo` statement is already included in the `App.js` file, delete it.
4. To make the components available for us to use, insert the following component import statement after the logo import statement.

   ```
   import {
     Header,
   ```

```
  PrimaryNavigation,
  SecondaryNavigation,
  NavigationLink,
  Link,
  HeaderOverflowMenu,
  OverflowMenuOption
} from "@govhhs/govhhs-design-system-react"
```

5. Replace the `<div>` code inside the create-react-app render function with the following code.

```
<div>
        <Header
          title="Agency name"
          logo={<img src={logo} alt="logo"/>}
        >
          <PrimaryNavigation>
            <NavigationLink
              title="Section One"
              id="basic-nav-section-one"
            >
              <Link>
              Sub-Link One
              </Link>
              <Link>
              Sub-Link Two
              </Link>
              <Link>
              Sub-Link Three
              </Link>
            </NavigationLink>
            <NavigationLink
              title="Section Two"
              id="basic-nav-section-two"
            >
              <Link>
              Sub-Link One
              </Link>
              <Link>
              Sub-Link Two
```

```
              </Link>
              <Link>
              Sub-Link Three
              </Link>
            </NavigationLink>
            <NavigationLink
              title="Link Three"
              id="basic-nav-section-three"
            />
            <NavigationLink
              title="Link Four"
              id="basic-nav-section-four"
            />
          </PrimaryNavigation>
        </Header>
      </div>
```

Add the Hero component.

6. In the component import statement, insert Hero into the list of imported components.

```
import {
  Hero,
  Header,
  PrimaryNavigation,
  SecondaryNavigation,
  SecondaryNavigationList,
  NavigationLink,
  Link,
  HeaderOverflowMenu,
  OverflowMenuOption
} from "@govhhs/govhhs-design-system-react"
```

7. When you add more than one component in the `return()` function, the JSX code must be wrapped in a surrounding `<div>` element. We want the Hero component to load underneath the header, so insert the `<Hero />` tag after the `</Header>` tag in the `return()` function. The `return()` function now has the following structure:

```
<div>
        <Header>
```

```
      ...
          <NavigationLink
            title="Link Four"
            id="basic-nav-section-four"
          />
        </PrimaryNavigation>
      </Header>
      <Hero />
    </div>
```

The hero component is displayed in the running application.

Now let's add some cards to our page. Cards are a great way to provide an entry point to more details, or access to actions on card content.

We must add some layout components to our list of imports:

- Grid
- Column
- Section

And add the card components to our list of imports:

- Card
- CardHeader
- CardBody
- CardFooter

8. Update your component import statement by inserting the following card components:

```
import {
  .....
  Card,
  CardHeader,
  CardBody,
  CardFooter,
  Grid,
  Column,
  Section,
} from "@govhhs/govhhs-design-system-react"
```

9. To display the cards underneath the `</Hero>` component, append the following JSX code to lay three link cards out in a grid:

```
<Section>
         <Grid>
         <h2>Card Actions</h2>
         <p>Before using cards in your project, have a look at the guidance in Storybook on how to use cards
correctly.</p>
           <Column width="1/3">
             <Card href="/404" className="wds-u-mr--small wds-u-mt--small">
               <CardHeader title="Card header" highlight/>
                 <CardBody>
                   <p>
                     Neque porro quisquam est qui dolorem ipsum quia dolor sit amet, consectetur, adipisci
velit.
                   </p>
                 </CardBody>
               <CardFooter>
               </CardFooter>
             </Card>
             </Column>
             <Column width="1/3">
             <Card href="" className="wds-u-mt--small">
               <CardHeader title="Card header" highlight/>
                 <CardBody>
                   <p>
                     Neque porro quisquam est qui dolorem ipsum quia dolor sit amet, consectetur, adipisci
velit.
                   </p>
                 </CardBody>
               <CardFooter>
               </CardFooter>
             </Card>
             </Column>
             <Column width="1/3">
             <Card href="" className="wds-u-mt--small">
               <CardHeader title="Card header" highlight/>
                 <CardBody>
```

```
                  <p>
                    Neque porro quisquam est qui dolorem ipsum quia dolor sit amet, consectetur, adipisci
velit.
                  </p>
                </CardBody>
              <CardFooter>
              </CardFooter>
            </Card>
          </Column>
        </Grid>
      </Section>
```

The cards are displayed in the running application.

Next, add a section that contains a form, an image, and a content area. For this section, we need to add some more components to our component import statement. Hopefully you're getting the hang of this!

- FieldSet
- TextInput
- Content
- TextArea
- Button
- Image

10. Add the following items to your component import statement:

```
import {
  ....
  FieldSet,
  TextInput,
  Content,
  TextArea,
  Button,
  Image,
  Form
} from '@govhhs/govhhs-design-system-react';
```

11. Because we are using an image in this part of the page, we need to import that too. Add the following line before our component import statement:

```
import img from '@govhhs/govhhs-design-system-core/dist/img/hero-480.jpg';
```

12. For the last step, to append the JSX code to our `return()` function, add the following code after the last `</Section>` tag:

```
<Section className="wds-c-section--light">
          <Grid>
            <Column width="1/2">
            <Content>
              <h3 className="wds-u-mt--small">About</h3>
              <p>All design system components are fully responsive and will work on mobile, tablet and desktop
 browsers.
              Try it out by resizing the browser window.</p>
              <Image
                src={img}
                alt="Error loading Image"
                fallback={<span />}
              />
            </Content>
            </Column>
            <Column width="1/2">
              <h3 className="wds-u-mt--small">Form title</h3>
                <form>
                  <FieldSet>
                    <TextInput
                      label="Text input"
                      value="Input text"
                    />
                    <TextInput
                      label="Text input"
                      value="Input text"
                    />
                    <TextArea label="Text area label"></TextArea>
                    <Button category="primary" style={{float: "right"}}>Sign up</Button>
                  </FieldSet>
                </form>
            </Column>
          </Grid>
        </Section>
```

That's it, your page is now complete, you have created a fully responsive page in very little time! Hopefully you enjoyed this tutorial and see the benefits of using our design system.

Here's the final App.js file for reference:

```
import React, { Component } from 'react';

import './App.css';

import '@govhhs/govhhs-design-system-core/dist/js/@govhhs/govhhs-wds.min';

import '@govhhs/govhhs-design-system-core/dist/css/govhhs-wds.min.css';

import logo from '@govhhs/govhhs-design-system-core/dist/img/logo-mark.svg';

import img from '@govhhs/govhhs-design-system-core/dist/img/hero-480.jpg';

import {

  Hero,

  Header,

  PrimaryNavigation,

  SecondaryNavigation,

  SecondaryNavigationList,

  NavigationLink,

  Link,

  HeaderOverflowMenu,

  OverflowMenuOption,

  Card,

  CardHeader,

  CardBody,

  CardFooter,

  Grid,

  Column,

  Section,

  FieldSet,

  TextInput,

  Content,

  TextArea,

  Button,

  Image,

  Form

} from "@govhhs/govhhs-design-system-react";


class App extends Component {
```

```
render() {
  return (
    <div>
      <Header
        title="Agency name"
        logo={<img src={logo} alt="logo"/>}
      >
        <PrimaryNavigation>
          <NavigationLink
            title="Section One"
            id="basic-nav-section-one"
          >
            <Link>
            Sub-Link One
            </Link>
            <Link>
            Sub-Link Two
            </Link>
            <Link>
            Sub-Link Three
            </Link>
          </NavigationLink>
          <NavigationLink
            title="Section Two"
            id="basic-nav-section-two"
          >
            <Link>
            Sub-Link One
            </Link>
            <Link>
            Sub-Link Two
            </Link>
            <Link>
            Sub-Link Three
            </Link>
          </NavigationLink>
          <NavigationLink
```

```
              title="Link Three"
              id="basic-nav-section-three"
            />
            <NavigationLink
              title="Link Four"
              id="basic-nav-section-four"
            />
          </PrimaryNavigation>
        </Header>
        <Hero />
        <Section>
         <Grid>
         <h2>Card Actions</h2>
         <p>Before using cards in your project, have a look at the guidance in Storybook on how to use cards
correctly.</p>
           <Column width="1/3">
             <Card href="/404" className="wds-u-mr--small wds-u-mt--small">
               <CardHeader title="Card header" highlight/>
                 <CardBody>
                   <p>
                     Neque porro quisquam est qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit.
                   </p>
                 </CardBody>
               <CardFooter>
               </CardFooter>
             </Card>
             </Column>
             <Column width="1/3">
             <Card href="" className="wds-u-mt--small">
               <CardHeader title="Card header" highlight/>
                 <CardBody>
                   <p>
                     Neque porro quisquam est qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit.
                   </p>
                 </CardBody>
               <CardFooter>
               </CardFooter>
```

```
            </Card>
            </Column>
            <Column width="1/3">
            <Card href="" className="wds-u-mt--small">
              <CardHeader title="Card header" highlight/>
                <CardBody>
                  <p>
                    Neque porro quisquam est qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit.
                  </p>
                </CardBody>
              <CardFooter>
              </CardFooter>
            </Card>
          </Column>
        </Grid>
      </Section>
      <Section className="wds-c-section--light">
         <Grid>
           <Column width="1/2">
           <Content>
             <h3 className="wds-u-mt--small">About</h3>
             <p>All design system components are fully responsive and will work on mobile, tablet and desktop
browsers.
             Try it out by resizing the browser window.</p>
             <Image
               src={img}
               alt="Error loading Image"
               fallback={<span />}
             />
           </Content>
           </Column>
           <Column width="1/2">
             <h3 className="wds-u-mt--small">Form title</h3>
               <form>
                 <FieldSet>
                   <TextInput
                     label="Text input"
```

```
                    value="Input text"
                  />
                  <TextInput
                    label="Text input"
                    value="Input text"
                  />
                  <TextArea label="Text area label"></TextArea>
                  <Button category="primary" style={{float: "right"}}>Sign up</Button>
                </FieldSet>
              </form>
            </Column>
          </Grid>
        </Section>
      </div>
    );
  }
}

export default App;
```

JavaScript™ development environment

You can use any JavaScript™ development environment to develop your application, for example, Microsoft™ Visual Studio Code, Atom, or Sublime. Choose the tools that suits you best.

The IBM Social Program Management Design System does not depend on any specific tools, so you can choose your own environment. However, Microsoft™ Visual Studio Code supports many plug ins that make development faster and easier, for example:

- Linting tools (ESLint)
- Code formatters (Prettier)
- Debugging tools (Debugger for Chrome)
- Documentation tools (JSDoc)

IBM® does not own, develop, or support any of the tools.

Design system packages

Use the design system packages to help you to develop and test your web client.

govhhs-design-system-core

This package contains a style guide, a library of user interface components, and front-end development resources that you can use to create Section 508-compliant, responsive, consistent web applications. The design system provides CSS, utility classes, and a grid framework so that you can quickly build accessible, responsive, production-ready websites.

govhhs-design-system-react

This package contains a React component library in Storybook to help you to build your application. It provides a collection of React components that align with the IBM Social Program Management Design System.

For more information about the design guidelines, utility classes, and React components, see `govhhs/govhhs-design-system-react/doc`.

core

This package provides JavaScript™ utilities to help you develop your application. For example, use the *RESTService* utility to connect to a IBM Cúram Social Program Management server-side REST API. Use `IntlUtils` to format numbers and dates for globalization.

For more information about the `core` package utilities, see the JSDoc API documentation in `spm/core/doc`.

core-ui

This package provides common React UI components to help you develop your application. For example, use the AppSpinner component to display a spinning animation while a page loads, or use the Toaster component to display notifications to the user.

For more information about the `core-ui` components, see the JSDoc API documentation in `spm/core-ui/doc`.

core-ui-locales

This package provides translated artifacts for the `core-ui` components.

intelligent-evidence-gathering

This package enables IEG scripts that are configured in the IBM Cúram Social Program Management application to run in your application. An API is provided to call the IEG scripts.

For more information, see the API documentation in `spm/intelligent-evidence-gathering/doc`.

`intelligent-evidence-gathering-locales`

This package contains translated artifacts for the `intelligent-evidence-gathering` package.

`spm-web-dev-accelerator`

This package contains the Social Program Management Web Development Accelerator rapid feature development tool, which generates Redux modules to handle the communication between your application and IBM Cúram Social Program Management REST APIs.

`spm-web-dev-accelerator-scripts`

This package contains a Swagger parser to retrieve information from IBM Cúram Social Program Management REST APIs, and scripts to generate the features and modules code from configuration information in the `spm-web-dev-accelerator` package.

`spm-test-framework`

This package contains a number of reusable files to help you to set up a test environment for testing with Test Cafe, Jest, and Enzyme. You can use the provided helper files to help you to develop and write end-to-end tests, unit tests, or snapshot tests for your project.

`spm-eslint-config`

This package contains an ESLint configuration with predefined coding style rules and an EditorConfig configuration file.

Connecting to REST APIs

You can connect your web application to REST APIs, such as the IBM Cúram Social Program Management REST APIs.

For more information about customizing IBM Cúram Social Program Management REST APIs, see *Developing Cúram REST APIs*.

Related information

Developing Cúram REST APIs

The RESTService utility

The `@spm/core` package provides the `RESTService` utility, which you can use to connect your application to a REST API. The `RESTService` utility provides important functions for securing and connecting to IBM Cúram Social Program Management REST APIs, such as CSRF protection and SSO support. You can fetch resources with alternatives such as Fetch API, SuperAgent, or Axis, but you must consider implementing functionality that is handled by the `RESTService` utility, like CSRF protection and SSO support.

The `RESTService` utility supports the GET, POST, and DELETE HTTP methods through the following JavaScript methods:

- `RESTService.get(url, callback, params)`
- `RESTService.post(url, data, callback)`
- `RESTService.del(url, callback)`

See the full `RESTService` class documentation in the `doc` folder in the `@spm/core` package.

The `RESTService` utility hides details of calls, such as passing credentials, language, and errors. The callback that is passed to the GET, POST, or DELETE methods is started after the API calls return. API calls are asynchronous, so write your code to expect and handle a delay in receiving a response.

The `RESTService` utility provides functions during communications for authentication, handling responses, and user language.

Authentication

Authentication of the user is handled transparently by the `RESTService` utility. After a user is authenticated, the REST APIs automatically send the needed 'credentials', that is, the authentication cookies, with each request. For information about how authentication is handled for REST, see Cúram REST API security.

If a user's session is invalidated before a new request is made to a REST API, then the '401 unauthorized' response is returned by the server. The `RESTService` utility relays the response to the callback function passed by the caller.

Handling responses

The `RESTService` utility formats the response from the server to ensure that callbacks receive the response in a consistent manner.

Each GET, POST, and DELETE method accepts a callback function from the caller. When called by the `RESTService` utility, the callback function receives a Boolean value that indicates the success or failure of the API call and the response. The callback function can then deal with the result. For example, a failure

can be used to trigger your code to throw an error with the response data that can be used to trigger an error boundary. For more information about the callback function parameters, see the API documentation for the `RESTService` utility.

User Language

The 'Accept-Language' HTTP header is automatically set by the `RESTService` utility based on the user's selected language, which the user can select with the language picker in the application. This approach lets the server respond in the correct locale where locale sensitive information is being handled on the server.

The locale that is passed in the header is set in the transaction that is initiated by that REST request, and is used for the duration of that transaction. For more on transactions, see Transaction control.

Cross-Site Request Forgery (CSRF)

The `RESTService` utility manages REST API CSRF protection for Universal Access that includes:

- Managing conditions on when to fetch a CSRF token.
- Pausing requests to fetch the CSRF token from the SPM server when needed.
- Storing the CSRF token in the application.
- Appending the CSRF token to the HTTP request header when appropriate.

Handling timeouts

The `RESTService` utility can manage unresponsive calls to the server. You can set environment variables in the `.env` files to set thresholds for timeouts.

- `REACT_APP_RESPONSE_TIMEOUT=10` Wait 10 seconds for the server to start sending.
- `REACT_APP_RESPONSE_DEADLINE=60` but allow 1 minute for the file to finish loading.

Simulating slow responses

During development, it is important to test that your application continues to operate in an acceptable way even when network responses are slow. You can simulate a slow network connection by setting a property in the `.env.development` file in the root of your project.

For example, set `REACT_APP_DELAY_REST_API=2` to delay the response from all GET requests for 2 seconds. The value can be set to any positive integer to adjust the delay.

Communicating with multiple API servers

IBM Social Program Management Design System is configured to connect and communicate with a single Social Program Management server. However, it is possible to communicate with multiple API servers through the `RestService` utility.

To define where resource requests are directed, you can register a mapping function that is started by the `RESTService` utility before it makes the eventual call to the server. You can use the mapping function to map the requested resource, for example, `/submitted_applications`, to a specific endpoint, for example, `https://spm-server1.com`. For all other resources, the default server that is specified in the `REACT_APP_REST_URL` property is called. How the mapping is achieved depends on the mapping function, as shown in the following example.

Note:
Complexities with authentication and session management exist that you must resolve separately from configuring the server that requests are directed to. Resolving the authentication and session management issues are project-specific tasks, and are outside the scope of this documentation.

1. Edit the `.env` configuration file in the root of your application, and add the **REACT_APP_REST_URL1** and **REACT_APP_REST_URL2** environment variables with the hostname and port of the server where the REST services are deployed, for example:

```
REACT_APP_REST_URL=/Rest

REACT_APP_REST_URL1=https://spm-server1.com/Rest

REACT_APP_REST_URL2=https://spm-server2.com/Rest
```

2. Update the `App.js` file to enable users to register and hook in a custom `apiEndPoints` function to determine the API endpoint URL for a particular resource, as shown in the following example:

```
import get from 'lodash.get';
   import REACT_APP_REST_URL from './REACT_APP_REST_URL';


   function isResourceMatchingDomainList(resource, domainItem) {
     const domainsDetailsList = JSON.parse(JSON.stringify(domainItem));
     const resArr = domainsDetailsList.map(x => x[Object.keys(x)[0]]);
     return resArr.find(y => resource.match(y));
   }


   const apiEndPoints = resource => {
```

```
    if (process.env.REACT_APP_REST_URL1 &&
      get(REACT_APP_REST_URL, 'data[0].REACT_APP_REST_URL1') &&
      isResourceMatchingDomainList(resource, REACT_APP_REST_URL.data[0].REACT_APP_REST_URL1) ) {
      return `${process.env.REACT_APP_REST_URL1}/${resource}`;
    }
    if (process.env.REACT_APP_REST_URL2 &&
      get(REACT_APP_REST_URL, 'data[0].REACT_APP_REST_URL2') &&
      isResourceMatchingDomainList(resource, REACT_APP_REST_URL.data[0].REACT_APP_REST_URL2) ) {
      return `${process.env.REACT_APP_REST_URL2}/${resource}`;
    }
    return `${process.env.REACT_APP_REST_URL}/${resource}`;
  };


  RESTService.registerApiEndPointsFunction(apiEndPoints);
```

3. Create the `SPM-WebApps\packages\universal-access-sample-app\src\REACT_APP_REST_URL.js` file that contains the resource URL per domain to be retrieved, as shown in the following example:

```
const REACT_APP_REST_URL = {
    data: [
     {
        REACT_APP_REST_URL1: [
          {
            url: '^v1/activities',
          },
         {
           url: '^v1/ua/messages$',
         },
        {
          url: '^v1/ua/public_messages$',
         },
         {
         url: 'v1/ua/communications/',
        },
      ],
       REACT_APP_REST_URL2: [
         {
          url: '^v1/ua/appeals',
```

```
        },
      ],
    },
  ],
};
export default REACT_APP_REST_URL;
```

4. Use regex expressions to match the URL to the JSON defined content.
5. Enter the following command to install dependent packages:

   ```
   npm install
   ```

6. Enter the following command to build the application into a build folder in the `universal-access-starter-pack`:

   ```
   npm run build
   ```

7. Copy the build folder to the HTTP server and deploy it.
   For more information, see Deploying your web application to a web server *(on page 40)*.

Connecting to REST APIs on Tomcat

If you have deployed REST APIs to Tomcat in your Eclipse development environment, you can connect to them from your application.

For more information about building and deploying IBM Cúram Social Program Management REST APIs, see *Creating a Cúram REST API* and *Deploying Cúram REST APIs on Tomcat*.

Use the API for Tomcat in the URL for your call to the RESTService utility. See the following example RESTService call.

```
RESTService.get('http://localhost:9080/Rest/v1/myAPI', (success, response) => {
  if (success) {
    // deal with response containing the json body
  } else {
    // deal with error contained in response
  }
});
```

Related information

Creating a Cúram REST API

Deploying Cúram REST APIs on Tomcat

Authenticating against REST APIs

You can authenticate against REST APIs by using the RESTService utility or by using the IBM Cúram Social Program Management REST API security feature.

Authenticating against REST APIs by using the RESTService utility

Start the authentication URL *j_security_check* by using the POST function on the RESTService utility.

The following example shows a sample log-in invocation:

```
const callbackAfterLogin = (success, response) => {
  if (success) {
    // Login succeeded
  } else {
    // Login failed
  }
};
const loginUrl = 'http://localhost:9080/Rest/j_security_check';
const loginData = {
  j_username: username,
  j_password: password,
  user_type: 'EXTERNAL',
};
RESTService.post(loginUrl, loginData, callbackAfterLogin, 'form');
```

Authenticating against REST APIs by using the IBM Cúram Social Program Management REST API security feature

For more information, see Cúram REST API security.

Authenticating against REST APIs in Tomcat

A JAAS authentication mechanism is not exposed by the Tomcat and Eclipse environment, so you cannot authenticate your web application through the RESTService utility. To authenticate against a REST API in a Tomcat and Eclipse environment, use the Eclipse RMILoginClient class.

For more information about authenticating against REST APIs in a Tomcat and Eclipse environment by using the Eclipse RMILoginClient class, see *Starting the clients*.

When you authenticate, log in by using your external user name and password to authenticate against the server. Subsequent calls to your REST API simulate this user.

Related information

Starting the clients

Chapter 7. Deploying your web application to a web server

You can deploy your web application on a web server in a production-like environment as part of your development process. Deployment in a production environment is outside the scope of this documentation, but you can use the instructions in this section for guidance.

Install and configure IBM® HTTP Server with WebSphere® Application Server

Install and configure IBM® HTTP Server either on the same server as WebSphere® Application Server or on a remote server. To enable cross-origin resource sharing (CORS), you can set the **curam.rest.allowedOrigins** property for the REST application on your application server, or install the IBM® HTTP Server plug-in for WebSphere® Application Server.

WebSphere® Application Server must be installed and configured.

You must install IBM® Installation Manager. For more information, see the IBM® Installation Manager documentation. You can download IBM® Installation Manager from Installation Manager and Packaging Utility download documents.

Note:

When the React application and the SPM server are deployed in different hosts that don't share the same top-level domain+1, and the web server where the React app is hosted doesn't run a proxy plug-in towards the SPM application servers, you must change the Cross-Site Request Forgery (CSRF) and session cookies for cross-origin requests, from the default `Samesite=Lax` to `Samesite=None`.

An alternative solution is to deploy a gateway web server in front of SPM to modify the cookie by using this directive:

```
Header edit Set-Cookie ^(.*)$ $1;SameSite=None;Secure
```

For SPM clusters, place this directive in the web servers where SPM applications are mapped.

To enable cross-origin resource sharing (CORS), choose one of the following options:

- Set the **curam.rest.allowedOrigins** property for the REST application that is deployed on the application server. For more information about setting the **curam.rest.allowedOrigins** property, see Cúram REST configuration properties.
- Install and configure the IBM® HTTP Server plug-in for WebSphere® Application Server to enable IBM® HTTP Server to communicate with WebSphere® Application Server. WebSphere® Customization Toolbox is needed to configure the plug-in.

1. Install IBM® HTTP Server. For more information, see Migrating and installing IBM HTTP Server.
2. **Optional:** If you don't set the **curam.rest.allowedOrigins** property, you must install the following software:

    a. Install the IBM® HTTP Server plug-in for WebSphere® Application Server.
    For more information, see Installing and configuring web server plug-ins.

    b. Install the WebSphere® Customization Toolbox.
    For more information, see Installing and using the WebSphere Customization Toolbox.

3. Start IBM® HTTP Server. For more information, see Starting and stopping the IBM HTTP Server administration server.
4. To secure IBM® HTTP Server, see Securing IBM® HTTP Server.

Generating an IBM® HTTP Server plug-in configuration

This task is needed only if you install the IBM® HTTP Server plug-in for WebSphere® Application Server. Use WebSphere® Customization Toolbox to generate a plug-in configuration.

Start WebSphere® Application Server. For more information, see Starting a WebSphere® Application Server traditional server.

To generate the IBM® HTTP Server plug-in configuration, complete the steps at the WebSphere® Application Server Network Deployment plug-ins configuration topic.

Configuring the IBM® HTTP Server plug-in

Configure the IBM® HTTP Server plug-in for WebSphere® Application Server and WebSphere® Customization Toolbox. This task is necessary only if you have chosen to install the IBM® HTTP Server plug-in, instead of setting the **curam.rest.allowedOrigins** property for the REST application that is deployed on the application server. Also, for information about how to configure the web server's HTTP verb permissions to mitigate verb tampering, see Enabling HTTP verb permissions.

You can run the `configurewebserverplugin` target to complete the following tasks:

- Add the web server virtual hosts to the client hosts configuration in WebSphere® Application Server.
- Propagate the plug-in key ring for the web server.
- Map the modules of any deployed applications to the web server.

1. Start IBM® HTTP Server.
   For more information, see Starting and stopping the IBM® HTTP Server administration server.
2. On the remote WebSphere® Application Server, run the following command.

   ```
   build configurewebserverplugin -Dserver.name=server_name
   ```

   The `configurewebserverplugin` target requires a mandatory `server.name` argument that specifies the name of the server when the target is invoked. For more information about the `configurewebserverplugin` target, see Configuring a web server plug-in in WebSphere® Application Server.
3. Consider adding extra aliases to the *client_host*, as shown in the following examples:
   - For WebSphere® Application Server, add port number `9044`.
   - For the default HTTP port, add port number `80`.
   - For HTTPS ports, add port number `433`.

   For more information about client host setup, see step 19 in the WebSphere® Application Server port access setup topic.
4. To avoid port mapping issues from web applications, restart WebSphere® Application Server and IBM® HTTP Server.
   For more information, see Starting and stopping the IBM® HTTP Server administration server.

Related information

Starting, stopping, and restarting IBM® WebSphere Application Server

Install and configure Oracle HTTP Server with Oracle WebLogic Server

Install and configure Oracle HTTP Server on either the same server as Oracle WebLogic Server or on a remote server.

Oracle WebLogic Server must be installed and configured. For more information, see Installing and Configuring Oracle WebLogic Server and Coherence for Oracle HTTP Server 12.1.3, and Installing and Configuring Oracle HTTP Server for Oracle HTTP Server 12.2.1.3.

Note:

When the React application and the SPM server are deployed in different hosts that don't share the same top-level domain+1, and the web server where the React application is hosted doesn't run a proxy plug-in towards the SPM application servers, you must change the Cross-Site Request Forgery (CSRF) and session cookies for cross-origin requests, from the default `Samesite=Lax` to `Samesite=None`.

An alternative solution is to deploy a gateway web server in front of SPM to modify the cookie by using this directive:

```
Header edit Set-Cookie ^(.*)$ $1;SameSite=None;Secure
```

For SPM clusters, place this directive in the web servers where SPM applications are mapped.

Installing Oracle HTTP Server and its components

Install and configure Oracle HTTP Server in either a stand-alone domain, or in an Oracle WebLogic Server domain. You must install and configure an Oracle web server plug-in for proxying requests.

The Oracle web server plug-in allows requests to be proxied from Oracle HTTP Server to Oracle WebLogic Server. If you install and configure the Oracle web server plug-in, requests that are delegated to Oracle WebLogic Server still appear to originate from the Oracle HTTP Server, even if Oracle HTTP Server and Oracle WebLogic Server are hosted on two different servers.

Because of the web browser same-origin policy, cross-origin resource sharing (CORS) is restricted in many browsers by default. The web server plug-in enables CORS where Oracle HTTP Server and Oracle WebLogic Server are installed on different computers.

CORS enables an instance of your web application that is deployed on Oracle HTTP Server in one domain to request the REST services that are deployed on Oracle WebLogic Server in another domain.

1. Install Oracle HTTP Server for Oracle WebLogic Server. For more information, see Installing and Configuring Oracle HTTP Server for Oracle HTTP Server 12.1.3, and Installing and Configuring Oracle HTTP Server for Oracle HTTP Server 12.2.1.3.
2. To configure Oracle HTTP Server, choose one of the following options:

    - To configure Oracle HTTP Server in a stand-alone domain, follow the instructions at Configuring Oracle HTTP Server in a Standalone Domain for Oracle HTTP Server 12.1.3, or Configuring Oracle HTTP Server in a Standalone Domain for Oracle HTTP Server 12.2.1.3.
    - To configure Oracle HTTP Server in an Oracle WebLogic Server domain, follow the instructions at Configuring Oracle HTTP Server in a WebLogic Server Domain for Oracle HTTP Server 12.1.3, or Configuring Oracle HTTP Server in a WebLogic Server Domain for Oracle HTTP Server 12.2.1.3.
3. If Oracle HTTP Server and Oracle WebLogic Server are installed in different domains, to enable CORS, install a web server plug-in.
   For more information about configuring an Oracle WebLogic Server proxy plug-in, see Configuring the Plug-In for Oracle HTTP Server for Oracle HTTP Server 12.1.3, or Configuring the Plug-In for Oracle HTTP Server for Oracle HTTP Server 12.2.1.3.
4. To secure Oracle HTTP Server, follow the procedure at Managing Application Security 12.1.3, or Managing Application Security for Oracle HTTP Server 12.2.1.3.

The Oracle HTTP Server instance is now ready to for you to deploy the application. The default location for deploying the application is *`OHS_INSTANCE`*`/config/fmwconfig/components/${COMPONENT_TYPE}/instances/${COMPONENT_NAME}/htdocs`. However, you can configure the default location value to a different location.

Start Oracle HTTP Server. For more information, see Next Steps After Configuring an Oracle HTTP Server Domain for Oracle HTTP Server 12.1.3, and Next Steps After Configuring the Domain for Oracle HTTP Server 12.2.1.3.

Configuring the Oracle HTTP Server plug-in

If a web server such as Oracle HTTP Server is configured in the topology, you must configure a web server plug-in in Oracle WebLogic Server. The web server plug-in enables Oracle WebLogic Server to communicate with Oracle HTTP Server. Also, for information about how to configure the web server's HTTP verb permissions to mitigate verb tampering, see Enabling HTTP verb permissions.

To enable an Oracle HTTP Server web server plug-in in Oracle WebLogic Server, you can run the `configurewebserverplugin` target.

1. Ensure the remote Oracle WebLogic Server Oracle WebLogic Server is running.
   For more information, see Starting and stopping Web Logic servers.
2. On the remote Oracle WebLogic Server, run the following command.
   The `configurewebserverplugin` target requires a mandatory `server.name` argument that specifies the name of the server when the target is invoked.

```
build configurewebserverplugin -Dserver.name=server_name
```

For more information about the `configurewebserverplugin` target, see Configuring a web server plug-in in Oracle WebLogic Server.

3. Restart the remote Oracle WebLogic Server.
   For more information, see Starting and stopping Web Logic servers .

Installing and configuring Apache HTTP Server

Install and configure Apache HTTP Server on either the same server as the application server or on a remote server. To enable cross-origin resource sharing (CORS), you can set the **curam.rest.allowedOrigins** property for the REST application on your application server, or install the appropriate plug-in for your web server. Also, for information about how to configure the web server's HTTP verb permissions to mitigate verb tampering, see Enabling HTTP verb permissions for guidance.

An application server must be installed and configured.

Note:

When the React application and the SPM server are deployed in different hosts that don't share the same top-level domain+1, and the web server where the React application is hosted doesn't run a proxy plug-in towards the SPM application servers, you must change the Cross-Site Request Forgery (CSRF) and session cookies for cross-origin requests, from the default `Samesite=Lax` to `Samesite=None`.

An alternative solution is to deploy a gateway web server in front of SPM to modify the cookie by using this directive:

```
Header edit Set-Cookie ^(.*)$ $1;SameSite=None;Secure
```

For SPM clusters, place this directive in the web servers where SPM applications are mapped.

To enable cross-origin resource sharing (CORS), choose one of the following options:

- Set the **curam.rest.allowedOrigins** property for the REST application that is deployed on the application server. For more information about setting the **curam.rest.allowedOrigins** property, see Cúram REST configuration properties.
- Install and configure the plug-in for your server.

1. Install Apache HTTP Server. For more information, see Compiling and Installing in the Apache HTTP Server documentation.
2. **Optional:** If you don't set the **curam.rest.allowedOrigins** property, you must choose one of the following options:
    - WebSphere® Application Server

      Install the plug-in for WebSphere® Application Server, see Installing and configuring web server plug-ins.

      Install the WebSphere® Customization Toolbox, see Installing and using the WebSphere Customization Toolbox.

      To configure Apache HTTP Server with WebSphere® Application Server, see Configuring Apache HTTP Server.

    - Oracle WebLogic Server 12cR1 (12.1.3):

      For more information about configuring an Oracle WebLogic Server proxy plug-in, see Configuring the Plug-In for Oracle HTTP Server.

      To configure Apache HTTP Server with Oracle WebLogic Server, see Configuring the Plug-In for Apache HTTP Server.

    - Oracle WebLogic Server 12cR2 (12.2.1.3):

      For more information about configuring an Oracle WebLogic Server proxy plug-in, see Configuring the Plug-In for Oracle HTTP Server.

      To configure Apache HTTP Server with Oracle WebLogic Server, see Configuring the Plug-In for Apache HTTP Server.
3. Start Apache HTTP Server. For more information, see Starting Apache in the Apache HTTP Server documentation.
4. To secure Apache HTTP Server server, see Security Tips and Apache SSL/TLS Encryption in the Apache HTTP Server documentation.

Building your web application for deployment

You must build your web application for deployment on a web server. A `build` directory is created that contains all of the required files for your web application.

For more information about npm build and deployment, see npm run build in the `create-react-app` GitHub documentation.

From your application root directory, run the following command to create the `build` directory.

```
npm run build
```

Deploying your web application to a web server

To test your web application against an existing IBM Cúram Social Program Management application that is deployed on an enterprise application server, you can deploy the web application on one of the supported web servers. The supported web servers are all based on Apache HTTP server so the deployment procedure is similar.

You must have built your application for deployment.

The `universal-access-starter-pack` package includes a preconfigured `.htaccess` file under the public folder that gets added to the built application. This file contains comments to explain the web server configuration requirements for React Router `BrowserRouter` enablement.

For more information about how to configure `.htaccess` files in a web server, see the *Apache HTTP Server Tutorial: .htaccess files* related link.

For more information about React Router `BrowserRouter`, see Serving Apps with Client-Side Routing.

1. Copy the contents of the `build` directory to the appropriate directory for your HTTP server.

   For more information about the `<directory>` directive, see the related links.

2. Configure the web server.

   - If you use `.htaccess`, enable the directives in `.htaccess` by editing `httpd.conf` and setting an appropriate value for the `AllowOverride` directive in the Directory section for the HTTP server's `DocumentRoot`, or the corresponding directory where the resources are being deployed.

     In addition, you must load the `mod_rewrite` module for the React Router `BrowserRouter`.

     ```
     # Enables mod_rewrite for React Router's BrowserRouter directives
     <IfModule !mod_rewrite.c>
         LoadModule rewrite_module modules/mod_rewrite.so
     </IfModule>
     # "/opt/IBM/HTTPServer/htdocs/universal" is the location
     # where the web application is deployed under the DocumentRoot.
     # Alternatively you can specify the DocumentRoot "/opt/IBM/HTTPServer/htdocs"
     <Directory "/opt/IBM/HTTPServer/htdocs/universal">
     ```

```
    AllowOverride FileInfo Options=MultiViews
</Directory>
```

- If you do not use `.htaccess`, you can copy the directives in `.htaccess` and put them in a `LocationMatch` section for your application in `httpd.conf`.

```
# Enables mod_rewrite for React Router's BrowserRouter directives
<IfModule !mod_rewrite.c>
    LoadModule rewrite_module modules/mod_rewrite.so
</IfModule>
# Below LocationMatch is set to "/universal" because the application
# will be served from https://youhostname.com/universal
<LocationMatch /universal>
    #
    # place here your .htaccess directives
    #
</LocationMatch>
```

3. Tune your HTTP server for improved performance, see the Performance Tuning guide.

Related information

GitHub documentation: npm run build

Content Security Policy Quick Reference Guide

Apache core features V2.0: <Directory> Directive

Apache core features V2.4: <Directory> Directive

Apache HTTP Server Tutorial: .htaccess files

Chapter 8. Troubleshooting and support

Use this information to help you to troubleshoot issues with the IBM® Cúram Universal Access Responsive Web Application or IBM Social Program Management Design System.

The IBM Cúram Social Program Management supported assets can be installed, customized, and deployed separately from IBM Cúram Social Program Management, before being integrated into the system.

When troubleshooting web applications that are integrated with IBM Cúram Social Program Management, use this troubleshooting information in conjunction with the troubleshooting information for IBM Cúram Social Program Management. For more information, see the *Troubleshooting and support* related link.

Related information

Troubleshooting and support

Citizen Engagement components and licensing

You can use and customize the IBM Universal Access Responsive Web Application for your organization, or develop your own custom web applications in addition to the standard IBM Cúram Social Program Management application. Use this information to understand the IBM Cúram Social Program Management components, supported assets, and licenses that you need.

Installable components

IBM Social Program Management Design System supported asset

The design system provides foundational packages for building accessible and responsive web applications. It consists of a React UI component library, React development resources, and a style guide for creating web applications.

IBM Universal Access Responsive Web Application supported asset

The IBM Universal Access Responsive Web Application provides a reference web application, which you can use and customize for your organization. The IBM Universal Access Responsive Web Application requires the IBM Social Program Management Design System and the Universal Access application module.

Universal Access application module

The Universal Access (UA) application module provides the Universal Access administrator application and the Universal Access REST APIs that expose interfaces to Universal Access

functions for consumption by the IBM Universal Access Responsive Web Application. Universal Access requires the IBM® Cúram Social Program Management Platform.

Licensing Universal Access

You can buy the Universal Access application module, which entitles the IBM Universal Access Responsive Web Application asset, and IBM® Cúram Social Program Management Platform, which entitles the IBM Social Program Management Design System asset.

Alternatively, you can buy Citizen Engagement, which includes the Universal Access application module, the IBM® Cúram Social Program Management Platform, and both assets.

Licensing the IBM Social Program Management Design System

To develop custom web applications to complement the IBM® Cúram Social Program Management Platform, you can buy the IBM® Cúram Social Program Management Platform, which entitles the IBM Social Program Management Design System asset.

Citizen Engagement support strategy

The Citizen Engagement assets are typically released monthly, and they can be upgraded independently of IBM Cúram Social Program Management (SPM) . Each release is a full release and not a delta release.

The assets are supported for the lifetime of the latest supported SPM version available at the time of the asset release.

- The main asset line is released monthly and contains new features, enhancements, security updates, defects, and support for the latest SPM version.
- IBM Universal Access Responsive Web Application 2.6 continues to be supported with security updates and critical defect fixes for older compatible SPM versions.

Although new features can be delivered in any asset release, they are typically delivered at the same time as the Universal Access application module release that contains the new APIs for those features. Where possible, Universal Access REST API changes are delivered in refresh pack or other impact-free releases that impose no forced upgrade impact.

Semantic versioning

The assets use semantic versioning. As a general guideline, this means:

- MAJOR version for incompatible API changes
- MINOR version for adding functionality in a backwards-compatible manner
- PATCH version for backwards-compatible bug fixes

Examining log files

Log files are a useful resource for troubleshooting problems.

Examining the browser console logs

For JavaScript applications, you can examine the browser console logs for errors that might be relevant to investigating problems. For the exact details about how to locate the console logs within the browser, see your browser documentation.

Note:
When you are developing applications with the IBM Social Program Management Design System, console logging information might also be displayed in the console that runs the start process for the application.

Examining the HTTP Server log files

When you deploy a built application on an HTTP Server, the built application introduces a new point with which logging is captured in your system topology. The IBM® HTTP Server, Oracle HTTP Server, and the Apache HTTP Server include comprehensive logging system and related information.

For more information about troubleshooting the IBM® HTTP Server, see Troubleshooting IBM HTTP Server.

For more information about troubleshooting the Oracle HTTP Server, see Managing Oracle HTTP Server Logs.

For more information about troubleshooting the Apache HTTP Server, see Log Files.

Examining the IEG log files

System administrators can enable improved logging by setting the **curam.trace system** administration property to `trace_on` or higher, and you can then check the server logs after you call the datastore prepopulation feature. You can view detailed logs that are generated during the population of data during screening, application intake, and life events to better explain what interactions have taken place. Information is output to the server logs during datastore prepopulation to describe which code path was taken and why.

The following information is written to the server logs during datastore prepopulation:

- Information about which code path was taken and why.
- The values of the relevant system administration properties.
- The schema names of the relevant IEG scripts.
- The number of records in the ViewProcessor table.

Notices

This information was developed for products and services offered in the United States.

IBM® may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM® representative for information on the products and services currently available in your area. Any reference to an IBM® product, program, or service is not intended to state or imply that only that IBM® product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM® intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM® product, program, or service.

IBM® may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM® Director of Licensing IBM® Corporation North Castle Drive, MD-NC119 Armonk, NY 10504-1785 US

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM® Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing Legal and Intellectual Property Law IBM® Japan Ltd. 19-21, Nihonbashi-Hakozakicho, Chuo-ku Tokyo 103-8510, Japan

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM® may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM® websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM® product and use of those websites is at your own risk.

IBM® may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM® Director of Licensing IBM® Corporation North Castle Drive, MD-NC119 Armonk, NY 10504-1785 US

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM® under terms of the IBM® Customer Agreement, IBM® International Program License Agreement or any equivalent agreement between us.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM® products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM® has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM®products. Questions on the capabilities of non-IBM® products should be addressed to the suppliers of those products.

Statements regarding IBM®'s future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM® prices shown are IBM®'s suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM®, for the purposes of developing, using, marketing or distributing

application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM®, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM® shall not be liable for any damages arising out of your use of the sample programs.

Privacy Policy considerations

IBM® Software products, including software as a service solutions, (“Software Offerings”) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering’s use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session cookies or other similar technologies that collect each user’s name, user name, password, and/or other personally identifiable information for purposes of session management, authentication, enhanced user usability, single sign-on configuration and/or other usage tracking and/or functional purposes. These cookies or other similar technologies cannot be disabled.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM®’s Privacy Policy at http://www.ibm.com/privacy and IBM®’s Online Privacy Statement at http://www.ibm.com/privacy/details the section entitled “Cookies, Web Beacons and Other Technologies” and the “IBM® Software Products and Software-as-a-Service Privacy Statement” at http://www.ibm.com/software/info/product-privacy.

Trademarks

IBM®, the IBM® logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM® or other companies. A current list of IBM® trademarks is available on the Web at “ Copyright and trademark information ” at http://www.ibm.com/legal/copytrade.shtml.