Munich Re Markets Portfolio Dashboard Widget

Installation

There are two ways to obtain the dependencies needed for using the Portfolio Dashboard widget:

1. From a package hosted on https://www.npmjs.com/ via a package manager such as NPM or Yarn

   Recommended for integration into web applications that already use a package manager and rely on a build process using a bundler such as Webpack

2. From a ready-to-use artifact distributed via the Munich Re Markets Content Delivery Network (CDN)

   Recommended for integration into websites that are built without the use of a package manager and/or do not rely on a bundler

Installation from NPM

In order to install the widget, run the following commands:

If you use NPM as a package manager:

```
npm install --save @munichremarkets/portfolio-dashboard
```

And if you use Yarn:

```
yarn add @munichremarkets/portfolio-dashboard
```

Installation from CDN

In this case, there is no dedicated installation step as the necessary dependency is already available as a ready-to-use artifact on the Munich Re Markets CDN.

Integration

The Portfolio Dashboard widget provides a (plain JavaScript) rendering API in the form of the `renderPortfolioDashboard` function that is used to display the widget on the page and pass data to the widget.

Importing from NPM

The Portfolio Dashboard package contains JavaScript (ES5) code in both ECMAScript and UMD (Universal Module Definition) module formats, so you need to use a bundler that supports either of these formats, such as Webpack or Rollup. Using ES6-style imports, your code might look like this:

```
import { renderPortfolioDashboard } from '@munichremarkets/portfolio-dashboard';
```

TypeScript declaration files (`.d.ts`) are also included in the package, providing typings and enhanced IDE support for the rendering API.

Importing from CDN

The CDN resource contains JavaScript (ES5) code in a format that does not make any assumptions on a module system. You can just include it into your website using a `<script>` tag like this (replacing `1.0.0` with the required version number):

```
<script src="https://releases.munichremarkets.com/portfolio-dashboard-1.0.0/widgets-portfolio-dashboard.js"></script>
```

This script provides its functionality under the dedicated global namespace `sriPortfolioDashboard` in order to avoid name clashes with other global JavaScript variables. The rendering API is made available as `sriPortfolioDashboard.renderPortfolioDashboard` .

Rendering the widget

In order to render the widget, you need to invoke the `renderPortfolioDashboard` rendering function that is made available as described above. The following data needs to be provided when invoking the rendering function:

- A target DOM element (typically a `<div>` element) into which the widget should be rendered

- A configuration object providing data to the widget, see section "Configuring the widget" below

The rendering function returns a promise that resolves when the rendering is complete.

For example, a call to the rendering function might look like this (ES6):

```
const target = document.getElementById('widget-target');

renderPortfolioDashboard(target, {
    /* configuration data, see below */
}).then(() => {
    /* code to be executed when the rendering is complete */
});
```

Please make sure to invoke the rendering function after the DOM has been loaded, e.g. by putting the call into a `<script>` tag at the end of the `<body>` or into a `DOMContentLoaded` event handler.

- **Note:** You need to make sure that the `process.env` object is available (not just `process.env.NODE_ENV`), e.g. using Webpack's Define Plugin.

Unmounting the widget

The promise returned by the rendering function resolves to a cleanup function. This function can be used to unmount the widget like this:

```
const target = document.getElementById('widget-target');

renderPortfolioDashboard(target, {
    /* configuration data, see below */
}).then(unmountWidget => {
    /* use `unmountWidget` e.g. in an event listener */
});
```

Configuring the widget

The configuration object passed to `renderPortfolioDashboard` can contain the following properties (example values):

```
{
    // Whether to include widget-scoped Bootstrap 4.x CSS into the page
    // Optional, default: false
    // Set this to `true` if you are not already using Bootstrap on your website
    includeBootstrap4Css: true,

    // Base URL of fund data backend
    // Optional, default: ''
    // May optionally end in '/'
    backendBaseUrl: 'https://some-url',

    // A token provided in JWT (JSON Web Token) format (see https://jwt.io).
    // The token is required for rendering the widget and permits access to the index data backend.
    // Mandatory
    jwt: '<some-token>',

    // Contact data of the insurer
    // Optional - if not specified, no contact information is displayed
    contactData: {
        // Name of the contact at the insurer
        // Mandatory
        name: 'Maximilian Mustermann',

        // Base64 encoded image representing the contact at the insurer
        // Optional - if not specified, no image is displayed
        // MIME type 'image/png', will be turned into a data URL by prepending ''
        imageBase64Encoded:
'PHN2ZyB2aWV3Qm94PSIwIDAgMTIwIDgwIiB4bWxucz0iaHR0cDovL3d3dy53My5vcmcvMjAwMC9zdmciPgogICAgPHRleHQ+RXhhbXBsZTwvdGV4dD4KPC9zdmc+',
    },

    // End date of the customer's insurance contract
    // Mandatory
    contractEndDate: new Date('2030-12-31'),
```

```
    // Configuration of which indicators are shown when selecting funds or viewing the detailed information for a specific fund
    // Optional - if not specified, a default indicator configuration is used (Sectors, Asset Classes, Regions)
    //
    // There are three slots, all of which are optional:
    // * On narrow screens, all slots that are specified are shown from top to bottom in order
    // * On wide screens, slot 1 is shown on the left, slot 2 in the center, and slot 3 on the right
    // * If none of the slots is specified, no indicator area is shown
    //
    // The following indicators are available for each slot:
    // IndicatorType.Sectors, IndicatorType.AssetClasses, IndicatorType.Regions, IndicatorType.ReturnRisk, IndicatorType.TopHoldings
    indicatorConfig: {
        slot1: IndicatorType.Sectors,
        slot2: IndicatorType.AssetClasses,
        slot3: IndicatorType.Regions,
    },

    // Language to use for localized texts
    // Mandatory
    // Currently available languages: Language.DE, Language.EN
    language: Language.DE,

    // Monthly contribution to the customer's portfolio in EUR
    // Mandatory
    monthlyContribution: 350,

    // List of funds in the customer's portfolio
    // Mandatory
    portfolioFunds: [
        {
            // ISIN of the fund
            // Mandatory
            isin: 'LU0323357649',

            // Share of the fund within the portfolio, from 0 to 1
            // Mandatory
            share: 0.5,

            // Monthly contribution to the fund in EUR
            // Mandatory
            monthlyContribution: 350,
        },
    ],

    // Total value of the customer's portfolio in EUR
    // Mandatory
    totalValue: 100000,

    // Custom texts per language
    // Optional - if not specified, default texts are used
    textOverrides: {
        'widgets.portfolioDashboard.title': {
            // Texts can be provided in any supported language (see `language` option)
            [Language.DE]: 'Vertragsübersicht',
            [Language.EN]: 'Contract Overview',
        },
    },
}
```

The following things should be considered in order to ensure that the widget works as intended.

Note: Providing data that does not adhere to the documented API of the widget may lead to unexpected behavior or the widget not working altogether.

Text override mechanism

Some texts that are displayed by the widget can be overridden by providing the `textOverrides` option in the configuration object. It expects an object in the form of

```
{
    'some.overridable.key': {
        [Language.DE]: 'Some german text',
        [Language.EN]: 'Some english text',
```

```
    },
    'some.other.key': { /* ... */ }
}
```

where `some.overridable.key` is a localization key specified by the widget.

Note: The override mechanism is entirely optional. If it is omitted, default texts are used instead.

Semantic markup

Some texts can contain basic semantic HTML markup that is sanitized before displaying. The following tags are allowed and will be displayed as bold, italic, underlined, etc.: `strong`, `b`, `em`, `i`, `u`, `br`, `p`.

Note: HTML tags cannot contain attributes. If provided anyway, they are removed during sanitization (e.g. `<em class="some-classname">some text</em>` will be converted to `<em>some text</em>`).

Interpolation

In some cases, texts might need to include dynamic values (e.g. monetary values or percentages). Such values can be referenced in the text using `{{someVariable}}`, where `someVariable` is a predefined name specific to the respective text key and will be replaced with the actual value during runtime.

Empty default text

In some cases default texts might be empty, which will cause the enclosing markup element (e.g. an information icon with accompanying text label) to purposefully not be displayed in the widget. This is a mechanism to remove markup elements entirely when the text that they usually display is missing. In these cases, a text override can be supplied to make the element visible again.

Grouping of sectors

The Portfolio Dashboard widget provides a fine-grained set of sectors that can be shown in the sector indicator and sector filter. The full set of available sectors is given by the text keys starting with `general.enums.fundSectors`:

```
general.enums.fundSectors.equityBasicMaterials
general.enums.fundSectors.equityCommunicationServices
general.enums.fundSectors.equityConsumerCyclical
general.enums.fundSectors.equityConsumerDefensive
general.enums.fundSectors.equityEnergy
general.enums.fundSectors.equityFinancialServices
general.enums.fundSectors.equityHealthServices
general.enums.fundSectors.equityIndustrials
general.enums.fundSectors.equityRealEstate
general.enums.fundSectors.equityTechnology
general.enums.fundSectors.equityUtilities
general.enums.fundSectors.fixedIncomeCashAndEquivalents
general.enums.fundSectors.fixedIncomeCorporate
general.enums.fundSectors.fixedIncomeDerivatesOther
general.enums.fundSectors.fixedIncomeGovernment
general.enums.fundSectors.fixedIncomeMunicipal
general.enums.fundSectors.fixedIncomeSecuritized
general.enums.fundSectors.other
```

In case you don't need this fine granularity, you can group multiple sectors together using the text override mechanism described above. More precisely, if you define the same text overrides for several of these text keys, the corresponding sectors are shown as one sector (using the given text) and their respective shares are added together.

For example, if both `equityConsumerCyclical` and `equityConsumerDefensive` are assigned the same text "Consumer Goods" and there is a share of 3% for `equityConsumerCyclical` and 4% for `equityConsumerDefensive`, the widget will group both together into a single sector called "Consumer Goods" with a share of 7%.

In fact, the default texts provided by the widget already group some sectors together using this mechanism.

As a special case, this means you can subsume sectors under the generic "Others" sector by assigning them the same text as `general.enums.fundSectors.other`.

Note that since texts are language-dependent, this grouping of sectors may also depend on the language. It is recommended to use the same grouping for all languages to avoid confusion.

Mandatory and optional fields in the configuration

- When using TypeScript, your IDE should automatically indicate which fields are mandatory/optional via the provided typings.
- When using plain JavaScript, please refer to the section above regarding mandatory/optional fields.
- The widget will perform a schema validation on the provided configuration and will fail to render if the validation fails. In this case you can then find additional information on why the validation failed in the console of your browser.

Providing enumeration values in the configuration

Libraries imported from `node_modules` (NPM packages)

Enumerations can be directly imported from the widget library. This works with both plain JavaScript and TypeScript. For instance, the `Language` enumeration can be used like this:

```
import { renderPortfolioDashboard, Language } from '@munichremarkets/portfolio-dashboard';

const target = document.getElementById('widget-target');
renderPortfolioDashboard(target, { language: Language.EN, ... });
```

Libraries included via CDN

Enumerations are accessible under the global namespaces provided by the widget library. For instance, the `Language` enumeration can be used like this:

```
const target = document.getElementById('widget-target');
sriPortfolioDashboard.renderPortfolioDashboard(target, { language: sriPortfolioDashboard.Language.EN, ... });
```

Although other ways of providing an enumeration value are technically possible (e.g. numerical or string values), it is considered improper usage and may stop working at any point.

Styling

Framework

The Portfolio Dashboard widget uses version 4.x of the Bootstrap framework for styling. When integrating the widget into your website, you need to do different things depending on whether your website uses Bootstrap, too:

Case 1: Your website uses Bootstrap

In this case, Bootstrap CSS rules are already present on your website and all the Bootstrap theme colors defined by your website will automatically be used by the widget. Note that the supported range of Bootstrap versions is from 4.2 to 4.6 (inclusive).

Case 2: Your website does not use Bootstrap

In this case, you are required to set the `includeBootstrap4Css` flag to `true` when invoking the widget rendering function, so that the Bootstrap 4.x CSS rules shipped with the widget are included on your page. Those CSS rules will not affect the appearance of elements outside the widget because they are scoped to the `.sri-widget` CSS class that is assigned to a DOM element the widget is rendered into.

Adaptation of styling

By default, the styling of the widget automatically adapts to the styling of your website to a certain extent. However, you can still customize the appearance if necessary.

Automatic styling

Fonts

Since the widget does not define its own font, any font you define in the DOM tree above the widget will automatically be used within the widget.

Spacings

Font sizes and most of the spacings are defined in units of `rem` (root `em`), so these sizes will automatically be set relative to the font size of your website's `<body>` element.

Width

The widget will automatically fill the full width of its parent element.

Theming

PRIMARY THEMING API: OVERRIDING BOOTSTRAP THEME COLORS & CSS VARIABLES

Regardless of whether you are providing a custom-themed Bootstrap or not, certain colors (*SRI colors*) need to be customized. The minimal set of CSS variables that should be overridden are `emphasis`, `element`, and the shades of `primary`, `secondary` and `emphasis`:

```
primary-[50-900]
secondary-[50-900]
emphasis
emphasis-[50-900]
element-[0-23]
element-other
```

The `element-*` colors are used within indicators and charts (funds, categories, ...). The `element-other` color is used specifically for those instances where it's necessary to display an "*Other*" category in an indicator or chart.

By default, all additional colors are derived from those colors so overriding them should already provide a good baseline for theming.

Case 1: Your website uses Bootstrap

Bootstrap theme colors: Bootstrap components used throughout the Munich Re Markets widgets will be styled correctly, CSS variables for those customized colors are expected to be available on `:root`. This should require no further action.

Case 2: Your website does not use Bootstrap

Bootstrap theme colors: The CSS variables referring to the Bootstrap theme colors need to be overridden:

```
--primary
--secondary
--success
--danger
--warning
--info
--light
--dark
```

To make sure that your settings for these variables take precedence over the defaults defined by the widget, use the CSS selector `.sri-widget.sri-external` like this:

```
.sri-widget.sri-external {
    --primary: red;
}
```

Note:

- Bootstrap internally uses SCSS variables, which are only present at build time. Since you will use the Bootstrap CSS rules shipped with the widget if you don't provide your own, components provided by Bootstrap will not be themed properly by just overriding the CSS variables. Also, classes like `.text-primary`, `.bg-primary` etc. will use wrong color values. In those cases the according CSS classes have to be overridden individually (see next section).
- Any style overrides except changing Bootstrap theme colors may stop working at any point and hence need to be thoroughly tested on every upgrade.

SECONDARY THEMING API: OVERRIDING CSS CLASSES

You can override each style of the application via CSS classes individually. Therefore, it is important to know the basic markup of the widget, which looks like this:

```
<!-- Passed into the rendering function by your website -->
<div>
    <div class="sri-widget sri-external">
        <div class="sri-portfolio-dashboard">
            <!-- Widget content -->
        </div>
    </div>
</div>
```

If you want to style an element within the main content of the widget, prefix all CSS rules with `.sri-widget.sri-external` to ensure that your CSS rule is more specific than the ones shipped with the widget and hence overrides the latter. For example, to style a button (`.sri-btn`), use a CSS rule like this:

```
.sri-widget.sri-external .sri-btn {
    /* ... */
}
```

If you use other widgets on the same page and need to scope a rule specifically to the Portfolio Dashboard widget, additionally include the widget-specific class like this:

```
.sri-widget.sri-external.sri-portfolio-dashboard .sri-btn {
    /* ... */
}
```

There is one special case: Some elements need to be rendered directly into the `<body>` of your website to ensure that they cover other elements, e.g. modal dialogs and tooltips. For these elements, the markup looks like this:

```
<div class="sri-widget sri-external sri-portfolio-dashboard sri-modal">
    <!-- Modal content -->
</div>
```

In this case, you can override styles within the modal using CSS rules like this:

```
.sri-widget.sri-external.sri-portfolio-dashboard.sri-modal .sri-btn {
    /* ... */
}
```

Notes:

- Use `!important` for rules originating from Bootstrap. This is because Bootstrap already declares all rules as `!important`, so your rules need to be `!important`, too, in order to override the Bootstrap rules.
- Any style overrides except changing Bootstrap theme colors may stop working at any point and hence need to be thoroughly tested on every upgrade.

Print view

In the print view, you might want to hide certain elements on your page (e.g. header and footer), reduce page margins etc. You can do this using a media query like this:

```
@media print {
    header {
        display: none !important;
    }
}
```

Recommendations for mobile viewports

Viewports in the "Extra small" (`xs`) and "Small" (`sm`) ranges of the responsive breakpoints defined by Bootstrap, i.e. narrower than `768px`, are considered "mobile" viewports and larger ones are considered "desktop" viewports.

Although the Portfolio Dashboard widget is generally responsive across all breakpoints, there are significant changes in layout between mobile and desktop viewports. Hence it is recommended to test your website in both variants.

Due to the scrolling behavior of the widget, it is recommended that you allow the widget to take the full height of the page with possible exceptions being a header and a footer. The widget assumes that a sticky header may be present on mobile viewports but not on desktop viewports. This distinction comes into play when the widget automatically scrolls to the top because this means scrolling to the top of the page on mobile viewports but to the top of the widget for desktop viewports.

Browser support

The following browsers are supported:

- Chrome (latest version)
- Firefox (latest version)
- Safari on macOS and iOS (latest two major versions)