Custom Codes in React Blocks

Global variables

There is a `<div>` element on top of the document `<body>` that stores application id and page id on it's `data-appid` and `data-pageid` attributes.

If there is a logged in user `window.logged_in_user` object is available with `softr_user_email` , `softr_user_full_name` properties and in case the user is connected to Airtable it may have other fields coming from Airtable user record.

Block specific data is stored on `window` object with `hrid` as an identifier of that block. (ex. `window['table1']` in case hrid is table1). The `window[hrid]` object differs based on block type.

- List, List details, Kanban, Chart, Organization chart, Calendar, Twitter and Map blocks have `baseId` and `tableName` properties.
- Form block has `airtableBaseUrl` property
- Map block has `google_map_api_key` property

There is `jQuery` or `$` available but we will get rid of it soon, so try to not use it much.

There is `openSwModal` global method that opens given url in modal.

ex. `openSwModal('https://softr.io/')`

▼ Chart colors

To change chart default colors we should set

```
window['chart1-colors'] = ['#FE8070', '#DD7E6B', '#EA9999', '#F4CCCC', '#24A37D', '#AEAEB5', '#E25B5B', '#FFF974', '#4BAEAE', '#E5E5EA',
```

where `chart1` is the block hrid.

Browser Events

Adding **event listeners** to **elements** that were rendered by **React** is tricky.
You may **add** an **event listener** but after **React re-renders** the **component** it might **loose** your **listener** because React can (in

some cases) remove and re-create element that you added the listener on.

To handle this the `event listener` should be added on `parent element` that doesn't re-render and `check` the element `selector` after `event trigger`.

ex.

```
const handler = (e) => {
    if (e.target.closest('#table1 .ag-row')) {
       // handle click event on ag-row
    }
};

document.body.addEventListener('click', handler);
```

This way you can handle `click` event on table rows with `table1` hrid.

Generic Custom Events

There are custom events that Softr blocks trigger or listen to.
We also add **hrid** to the event name to identify the block the event refers to.

▼ block-loaded

`block-loaded` event is triggered when React mounts the block into DOM. It can be used instead of `DOMContentLoaded` event that is used in old custom codes.

ex.

```
window.addEventListener('block-loaded-table1', () => {
  console.log('Block loaded');
});
```

▼ get-record

`get-record` event is triggered on every single data response from softr data service. It is used on list-details blocks and it can be used as for getting the data and using it for other 3rd party calls or as a indicator that after some small interval time the block will be fully rendered.
ex.

```
const onRecord = (e) => {
  // we got new data under e.details
  console.log(e.detail);
  //console.log {id: '***', fields: {...}}
};

window.addEventListener('get-record-list-details1', onRecord);
```

```
const onRecord = (e) => {
  setTimeout(() => {
    // The block finish rendering
    // I may do some staff here.
  }, 50);
};

window.addEventListener('get-record-list-details1', onRecord);
```

```
window.addEventListener('get-record-list-details1', (e) => {
  // hide list details block if no record found
  if (!e.detail.id) {
    document.getElementById('list-details1').classList.add('d-none');
```

```
  }
});
```

▼ get-records

`get-records` event is triggered on every data response from softr data service. It can be used as for getting the data and using it for other 3rd party calls or as a indicator that after some small interval time the block will be fully rendered.
ex.

```
const onRecords = (e) => {
  // we got new data under e.details
  console.log(e.detail);
  //console.log [{id: '***', fields: {...}}]
};

window.addEventListener('get-records-table1', onRecords);
```

```
const onRecords = (e) => {
  setTimeout(() => {
    // The block finish rendering
    // I may do some staff here.
  }, 50);
};

window.addEventListener('get-records-table1', onRecords);
```

Also there is an `get-records:before` event that is triggered before sending request, It can be used to catch the inline filter or search field changes.

```
const onFilterChange = (e) => {
  console.log(e.detail);
  //console.log { search: '', filter: [ { field: 'size', value: 'M' } ] }
};

window.addEventListener('get-records-table1:before', onFilterChange);
```

▼ update-records

`update-records` event is listened by all blocks that use external data.
It can be used to change/add/remove the data that should be rendered. Mostly it can be used in pair with `get-records`.
ex.

```
const onRecords = (e) => {
  const modifiedRecords = e.detail.map(({fields, ...other}) => ({
    ...other,
    fields: {
      ...fields,
      phone: fields.phone ? fields.phone.replace('+374', '0') : '',
    }
  }));

  const modify = new CustomEvent('update-records-table1', { detail: modifiedRecords });
  setTimeout(() => window.dispatchEvent(modify), 1);
};

window.addEventListener('get-records-table1', onRecords);
```

Form Custom Events

▼ update-fields

`update-fields` event is listened by blocks that use form inputs with Formik library. Currently it's blocks under Form and User Accounts categories. It can be used to update input values with custom code.

ex.

```
window.addEventListener('block-loaded-form1', () => {
  const updateFields = new CustomEvent('update-fields-form1', {
      detail: {
        'Full Name': 'Softr',
        'Email': 'info@softr.io'
      }
  });
  window.dispatchEvent(updateFields);
});
```

▼ submit-form, submit-form-success, submit-form-failure

`submit-form` event with field values as an attribute is triggered before sending form submission request.
`submit-form-success` event with field values is triggered after getting response with success status code.
`submit-form-failure` event is triggered after getting response with error code.

ex.

```
window.addEventListener('submit-form-form1', (e) => {
  // e.detail is an object with form field names and values
  console.log('form submit', e.detail);
  // form submit { "Full Name": "Softr", "Email": "info@softr.io" }
});

window.addEventListener('submit-form-success-form1', (e) => {
  // e.detail is an object with form field names and values
  console.log('form submit success', e.detail);
  // form submit success {
  //                   payload: { "Full Name": "Softr" ... },
  //                   response: { headers: {...}, data: {...}, status: 200 }
  //                 }
});

window.addEventListener('submit-form-failure-form1', (e) => {
  console.log('form submit failure', e.detail);
  // form submit failure 'Email field is required.'
});
```

▼ set-form-dropdown-values

`set-form-dropdown-values` can be used to set dropdown values dynamically.

ex.

```
window.addEventListener('block-loaded-form1', () => {
  const updateOptions = new CustomEvent('set-form-dropdown-values-form1', {
      detail: {
        size: ['S', 'M', 'L'],
        color: ['Red', 'Blue', 'Yellow']
      }
  });
  window.dispatchEvent(updateOptions);
});
```

Header Custom Events

▼ set-logo-link

`set-logo-link` event is used to set header logo link to custom url.

ex.

```
window.addEventListener('block-loaded-header1', () => {
  const detail = { link: 'https://google.com' };
  window.dispatchEvent(new CustomEvent('set-logo-link-header1', { detail }));
});
```

▼ trigger action before logout

first will trigger custom action and after 300 ms will continue to Sign Out

ex.

```
window.addEventListener('user-sign-out', (e) => {

  // do some actions before logout - (300 ms)

});
```

Styling

There is still bootstrap included in the page, but it will be removed in the near future. So try not to use it.

Material-ui React component library is used in the new blocks. It adds classes to all small to large components with the prefix of `Mui` those classes can be used to add custom styles to the elements, but we may also change it to `Softr` or something similar in the near future (ex. `MuiInputBase` to `SoftrInputBase`).

Try wrapping selector with hrid to add more priority to your selector.
ex.

```
#table1 .MuiInputBase-input {
  border-left: none;
}
```

There are also some attributes that expose the content of the element. It can help identify and style them based on it.

- data-content attribute on tag elements
- data-rating attribute on rating elements

Because CSS supports attribute selectors, they can be used to style the elements based on content.

ex.

```
// tags with content "Low"
#table1 .tag-item[data-content="Low"] {
    background-color: #F00 !important;
}
// add span to the selector if you want to customize text color
#table1 .tag-item[data-content="Low"] span {
    color: #FFF !important;
}

// tags that start with word "Low"
#table1 .tag-item[data-content^="Low"] {
    background-color: #F00 !important;
}
// tags that have substring "low"
#table1 .tag-item[data-content*="low"] {
    background-color: #F00 !important;
}

// ratings that are 1 start
```

```
#table1 [data-rating="1"] span {
    color: #F00 !important;
}
```