Findings for FirstSpirit Third-Party-Preview API Integration with Salesforce Lightning Community

Approach: Include TPP Script in Global Head Markup

Implementation of Global Head Markup

- In the Community *Settings* go to section *Security* and configure the following:

  - *Content Security Policy Script Security Level:* Allow Inline Scripts and Script Access to Whitelisted Third-party Hosts
  - Add this to the *Trusted Sites for Scripts*: `https://unpkg.com`
  - Go to *CSP Trusted Sites* and add the trusted site URL: `unpkg.com`

- In the Community *Settings* go to section *Advanced* and add the following to the *Head Markup*:

```
<script src="https://unpkg.com/fs-tpp-api@1.2.14/snap.js"></script>
```

Issues with Global Head Markup

- `snap.js` seems to use Webpack style-loader which attempts to add stylesheets as `Blob` to the document. Even with the relaxed CSP browsers reject loading these stylesheets because the `blob:` scheme is not in the list of allowed sources.

  - CSP response header of the Community page:

```
Content-Security-Policy: default-src 'self'; script-src 'self' 'unsafe-
eval' 'unsafe-inline' https://service.force.com/embeddedservice/
https://unpkg.com; object-src 'self' www.google.com; style-src 'self'
'unsafe-inline' www.sto.de shop.sto.de unpkg.com; img-src 'self' data:
blob: https://b2commerce-mms--devgew1.my.salesforce.com
http://b2commerce-mms--devgew1--c.documentforce.com
https://img.youtube.com https://i.ytimg.com https://i.vimeocdn.com
https://login.salesforce.com/icons/ https://cs84.salesforce.com/icons/
```

```
www.sto.de shop.sto.de unpkg.com; media-src 'self' blob: www.sto.de
shop.sto.de unpkg.com; frame-ancestors *; frame-src 'self'
https://service.force.com/embeddedservice/ https://cs84.salesforce.com
https://sfdc-link-preview-staging.sfdc.sh https://sfdc-link-
preview.hk.salesforce.com https://cdn.embedly.com
https://www.youtube.com https://player.vimeo.com
https://play.vidyard.com www.sto.de shop.sto.de unpkg.com; font-src
'self' data: www.sto.de shop.sto.de unpkg.com; connect-src 'self'
www.sto.de shop.sto.de unpkg.com
```

  - The `style-src` is only: `'self' 'unsafe-inline' www.sto.de shop.sto.de unpkg.com` but would have to be `'self' 'unsafe-inline' blob: www.sto.de shop.sto.de unpkg.com` in order to make it work. See https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy/style-src
  - The *CSP Trusted Sites* configuration of Salesforce does not accept just `blob:` as a trusted site URL. So the `Blob` support cannot be added like a usual external domain.
  - Possible solution: FirstSpirit should provide separate JavaScript and CSS files, so that the script does not need to embed CSS using `Blob`.

- `snap.js` is evaluated in the context of the `window` which is not handled by Lightning Locker. Thus, the global `TPP_SNAP` is defined on this outer `window` object. Lightning Web Components in the page are managed by Lightning Locker and therefore are rendered in the context of a secured `window` object. This object does not inherit the global `TPP_SNAP`. The library is not accessible from within Lightning Web Components:

```
console.info(typeof TPP_SNAP); // Logs 'object' in browser window but
'undefined' inside web component
```

Approach: Include TPP Script in Theme Layout Component

Implementation of Theme Layout Component

- Obtain the NPM module for the TPP API: https://www.npmjs.com/package/fs-tpp-api
- Upload the files contained in the NPM module as a *Static Resource* to Salesforce (e.g. ZIP file with subdirectory `fs-tpp-api` containing the JavaScript files).
- Create a custom *Theme Layout Component* and load the `snap.js` file from the *Static Resource*:

```
<aura:component implements="forceCommunity:themeLayout">
  <ltng:require scripts="{!$Resource.StoPocCMSResources + '/fs-tpp-
api/snap.js'}" />
  {!v.body}
</aura:component>
```

Issues with Theme Layout Component

- Evaluation of `snap.js` fails with the following error:

```
{
  "actions": [
    {
      "id": "63;a",
      "descriptor": "aura://ComponentController/ACTION$reportFailedAction",
      "callingDescriptor": "UNKNOWN",
      "params": {
        "failedAction": "c:stoPocCMSThemeLayout",
        "failedId": "-1958337896",
        "clientError": "AuraFriendlyError: Custom Script Eval error in
'ltng:require' [SecureDOMEvent: [object Event]{ key: {\"namespace\":\"c\"}
}]",
        "clientStack": "Proxy.eval()@https://devgew1-
demoshop.cs84.force.com/components/ltng/require.js:7:361",
        "componentStack": "[c:stoPocCMSThemeLayout] > [ltng:require]",
        "stacktraceIdGen": "injectScript:require:ltng",
        "level": "ERROR"
      }
    }
  ]
}
```

- Lightning Locker Console does not really help with error analysis. It just displays: *Error: Unsupported MIME type.*

- `snap.js` is evaluated in the context the `window` object secured by Lightning Locker. When the script attempts to query HTML elements using `document.querySelectorAll('[data-preview-id]')` it will not obtain those elements generated by child components.

```
document.querySelectorAll('[data-preview-id]'); // Does not return elements
generated by child components
```

Approach: Include TPP Script in Web Components

Note: Not implemented so far.

Implementation of Web Components

- Obtain the NPM module for the TPP API: https://www.npmjs.com/package/fs-tpp-api

- Upload the files contained in the NPM module as a *Static Resource* to Salesforce (e.g. ZIP file with subdirectory `fs-tpp-api` containing the JavaScript files).

- Create a custom *Lightning Web Component* and load the `snap.js` file from the *Static Resource*:

```
import { loadScript } from 'lightning/platformResourceLoader';
import StoPocCMSResources from '@salesforce/resourceUrl/StoPocCMSResources';

export default class FirstSpiritCMSComponent extends LightningElement {
  renderedCallback() {
    loadScript(this, StoPocCMSResources + '/fs-tpp-api/snap.js').then(() =>
{
      // TPP_SNAP will be defined here
    });
  }
}
```

Issues with Web Components

- `snap.js` will be loaded only once even if imported in multiple instances of the web component but still needs to be evaluated in the context of each of the components. Potential performance issue.
- Access to HTML elements generated by the web component using `document.querySelectorAll()` is not possible.

```
document.querySelectorAll('[data-preview-id]'); // Does not return elements
generated by web component
```