bloomreach

Bloomreach Discovery

PWA Integration Guide

Version 1.0.0

Table of Contents

1. Summary

The Bloomreach Discovery PWA integration is built for the Salesforce Commerce Cloud PWA. Often referred to as Salesforce composable. It allows you to quickly kickstart integration of Bloomreach Discovery into a Salesforce PWA storefront. This documentation contains instructions for integration, deploying and running the project on a Managed Runtime Environment (MRT). It includes instructions, code samples and best practices for an integration project.

Bloomreach Discovery offers AI-Driven product discovery focused on ROI. It leverages Loomi, our industry leading AI for ecommerce, to further drive product discovery. The Bloomreach Discovery x Shopify connector significantly simplifies your implementation to make sure pixel data from your storefront is fed appropriately into Bloomreach Discovery. Build your storefront quicker with a templated search UI and components for search results pages, facets and autosuggest.

2. Component Overview

Functional Overview

The Bloomreach service allows the customer to use a search engine on your PWA.

Limitations, Constraints

Use of this PWA requires credentials and API keys from Bloomreach. You will also need to set up the necessary environment variables. You also need to upload, install and configure the CORE BLM cartridge (from the github: https://github.com/bloomreach/discovery-sfcc-b2c) in the sandbox.

Contract

To work with the PWA, you need to contact the Bloomreach support service to get your personal credentials to gain access to the Bloomreach services.

Compatibility

The PWA is designed for Salesforce Commerce Cloud PWA version 3.0.0 and above.

Privacy, Payment

The PWA doesn't collect and process user profile information or billing information. For additional privacy information, please contact your Bloomreach Account Manager.

3. Implementation Guide

Set-up of CORE Cartridge

In order to set-up your account correctly, please use the document "Core Cartridge Integration guide.pdf"

Required Configurations

Open **./config/default.js**

You will need to populate the highlighted sections with your own sandbox data, this will point the PWA to your commerce instance.

```
// The default site for your app. This value will be used when a siteRef
defaultSite: 'Bloomreach_SFRA',
// Provide aliases for your sites. These will be used in place of your si
throughout the application.
// siteAliases: {
//     RefArch: 'us'
// },
// The sites for your app, which is imported from sites.js
sites,
// Commerce api config
commerceAPI: {
    proxyPath: '/mobify/proxy/api',
    parameters: {
        clientId: '17b5ae6b-2245-47e3-8460-becfe2570ad5',
        organizationId: 'f_ecom_zzrk_005',
        shortCode: 'kv7kzm78',
        siteId: 'Bloomreach_SFRA'
    }
},
```

```
// Additional parameters that configure Express app behavior.
ssrParameters: {
    ssrFunctionNodeVersion: '18.x',
    proxyConfigs: [
        {
            host: 'kv7kzm78.api.commercecloud.salesforce.com',
            path: 'api'
        },
        {
            host: 'zzrk-005.dx.commercecloud.salesforce.com',
            path: 'ocapi'
        },
```

More information can be found here:
https://developer.salesforce.com/docs/commerce/pwa-kit-managed-runtime/guide/setting-up-your-local-environment.html#configuration-values

Local Environment Variables

At first if you checkout the project and do “npm install” and you try to run “npm start” you will get the following error:

```
Error: Environment variable BLM_ACCOUNT_ID is required.
```

This is because we NEED to set-up environment variables to run the project.

Open project folder **./bloomreach/settings/credentials.template.json** file. It should look like this:

```
{
    "BLM_ACCOUNT_ID": {"value": ""},
    "BLM_AUTH_KEY": {"value": ""},
    "BLM_DOMAIN_KEY": {"value": ""},
    "BLM_CATALOG_NAME": {"value": ""},
    "BLM_WIDGET_BEST_SELLER_ID": {"value": ""},
    "BLM_WIDGET_RECOMMEND_ID": {"value": ""},
    "BLM_WIDGET_FREQUENTLY_VIEWED_TOGETHER_ID": {"value": ""},
    "BLM_WIDGET_FREQUENTLY_BOUGHT_TOGETHER_ID": {"value": ""},
    "BLM_DEBUG": {"value": ""},
    "BLM_TEST_DATA": {"value": ""}
}
```

As you can see we need to enter value for ALL the fields. Lets see what each is

#	Field Title	Description
1	BLM_ACCOUNT_ID	This is part of personal credentials to gain access to the Bloomreach services. Contact the Bloomreach support service to get your Account ID.
2	BLM_AUTH_KEY	This is part of personal credentials to gain access to the Bloomreach services. Contact the Bloomreach support service to get your Authentication Key.
3	BLM_DOMAIN_KEY	Domain Key is used as a product catalog identifier on the Bloomreach side. This is part of personal credentials to gain access to the Bloomreach services. Contact the Bloomreach support service to get your Domain Key.
4	BLM_CATALOG_NAME	The name of the products catalog
5	BLM_WIDGET_BEST_SELLER_ID	The id of the widget that will be used for best selling products on the Homepage.
6	BLM_WIDGET_RECOMMEND_ID	The id of the widget that will be used for the recommended products on PDP.
7	BLM_WIDGET_FREQUENTLY_VIEWED_TOGETHE	The id of the widget that will be used for the frequently viewed together products on PDP.

	R_ID	
8	BLM_WIDGET_FREQUENTLY_BOUGHT_TOGETHER_ID	The id of the widget that will be used for the frequently bought together products on PDP.
8	BLM_DEBUG	This is Pixel tracking settings for Staging and Development environments. It should be set to “false” for Production. More information here: https://documentation.bloomreach.com/discovery/docs/global-page-view-pixel
10	BLM_TEST_DATA	This is Pixel tracking settings for Staging and Development environments. It should be set to “false” for Production. More information here: https://documentation.bloomreach.com/discovery/docs/global-page-view-pixel

NOTE: Environment variables starting with “BLM_WIDGET_” are NOT mandatory and you can run the project without them.

Populate the fields with the necessary data AND delete the “.template” string from the **credentials.template.json** file, so that it becomes **credentials.json**

Next let's export the variables so we can run the project locally. We have two scripts in the **./bloomreach/settings/** folder:

1. export_env.sh - for Linux and MacOS users
2. export_env.ps1 - for Windows Users

MacOS / Linux

Open **./bloomreach/settings/** directory in terminal

We need to install “jq” package first

# For Debian/Ubuntu
sudo apt-get install jq

For macOS using Homebrew
brew install jq

Then run the **export_env.sh** script with:
source ./export_env.sh

Now, in the root directory of the project, you should be able to run:
npm start

Windows

Open **./bloomreach/settings/** directory in PowerShell

Run the Windows script in PowerShell with administrator privileges
.\export_env.ps1

Next add the variables(if they are not already added) in the **./config/default.js** file:

```
app: {
    blm: {
        accountId: parseEnvVar('BLM_ACCOUNT_ID') || null,
        authKey: parseEnvVar('BLM_AUTH_KEY') || null,
        userId: parseEnvVar('BLM_DOMAIN_KEY') || null,
        catalogName: parseEnvVar('BLM_CATALOG_NAME') || null,
        widgetBestSellerId: parseEnvVar('BLM_WIDGET_BEST_SELLER_ID') || null,
        widgetRecommendId: parseEnvVar('BLM_WIDGET_RECOMMEND_ID') || null,
        widgetFrequentlyViewedTogetherId:
            parseEnvVar('BLM_WIDGET_FREQUENTLY_VIEWED_TOGETHER_ID') || null,
        widgetFrequentlyBoughtTogetherId:
            parseEnvVar('BLM_WIDGET_FREQUENTLY_BOUGHT_TOGETHER_ID') || null,
        debug: parseEnvVar('BLM_DEBUG') || null,
        testData: parseEnvVar('BLM_TEST_DATA') || null
    }
},
```

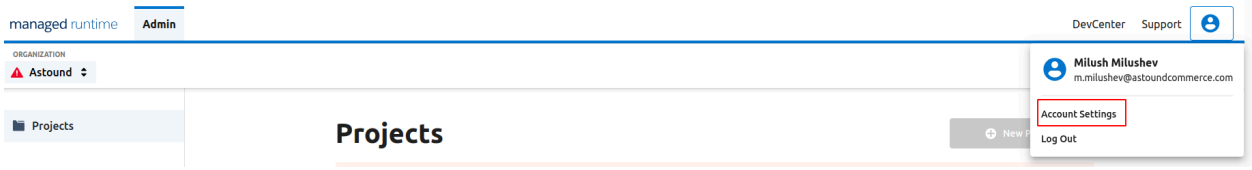
Now, in the root directory of the project, you should be able to run:
npm start

Server Environment Variables

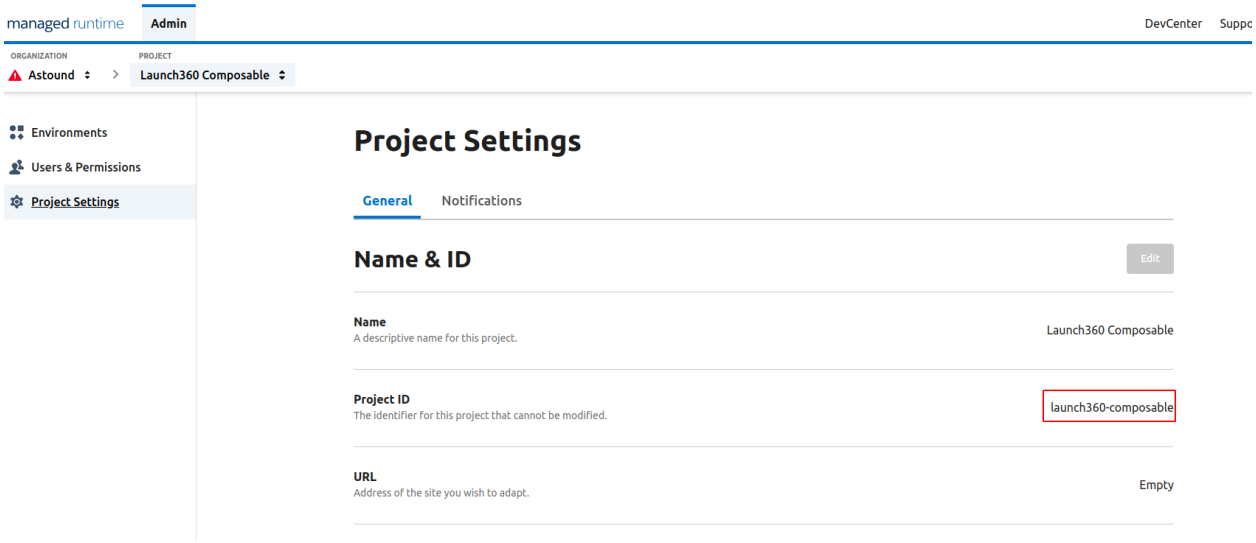
Utils

Lets set-up first some common variables that we are going to use to better illustrate requirements for environment variables:

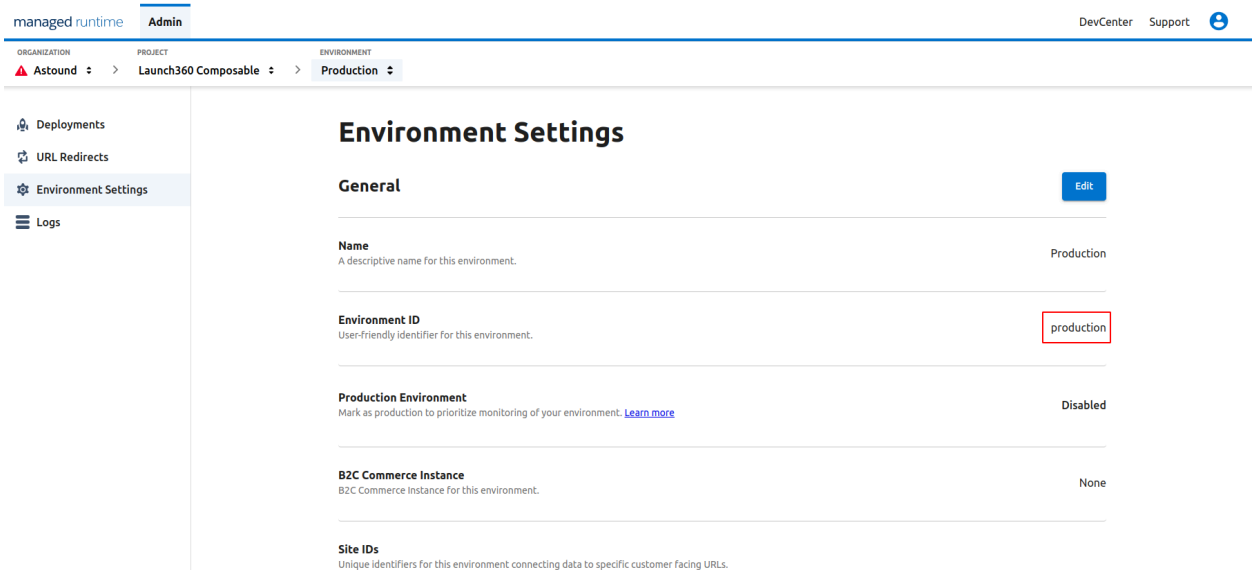
mobify_key="<your-own-mobify-key-in-the-mrt-environment>"
Can be found in here:



project_slug="<the-name-of-the-project-in-the-mrt-environment>"
Can be found in:



target_slug="<environment-id>"
Can be found here:



Check for already present env variables

Open **./bloomreach/settings/** directory in terminal and run the following:

```
curl "https://cloud.mobify.com/api/projects/$project_slug/target/$target_slug/env-var/" \
 --header "Authorization: Bearer $mobify_key" \
 --header 'Accept: application/json'
```

If you run it the first time it should return only “{}” empty JSON

Set env variables

Open **./bloomreach/settings/** directory in terminal and run the following:

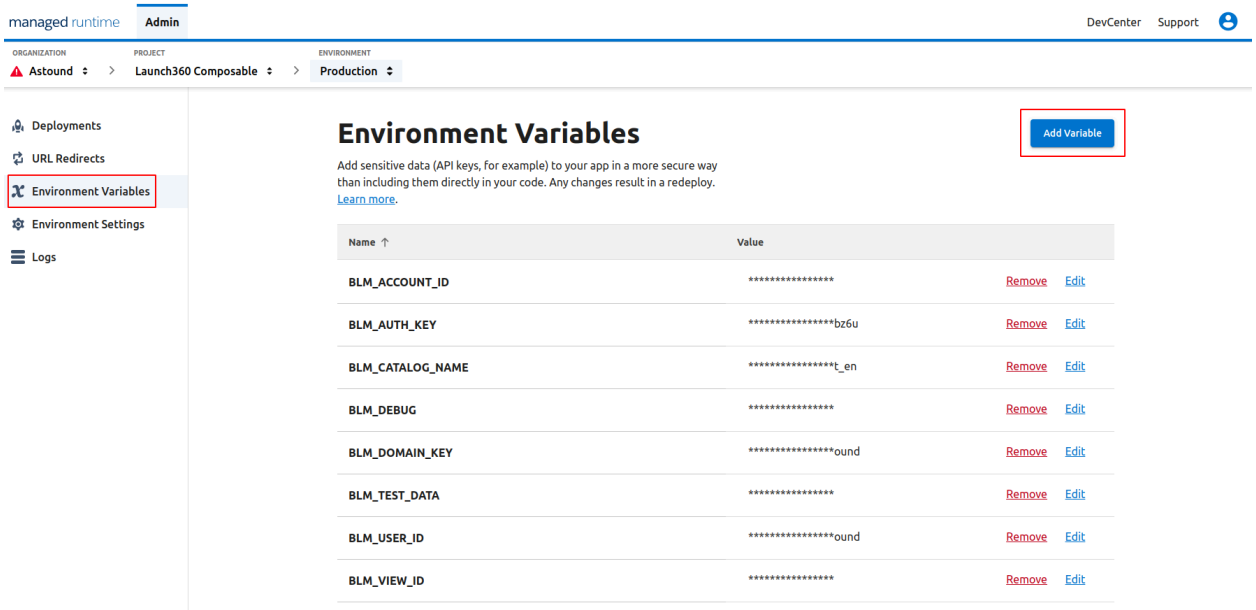
```
curl "https://cloud.mobify.com/api/projects/$project_slug/target/$target_slug/env-var/" \
--request 'PATCH' \
--header 'Content-Type: application/json' \
--header 'Accept: application/json' \
--header "Authorization: Bearer $mobify_key" \
--data @credentials.json
```

NOTE: credentials.json should already be populated with the necessary values

Now if you check again for present variables the JSON object should contain information about the variables. And keep in mind that their value will be asterisked like “**********”

More information about the environment variables can be found here:
https://developer.salesforce.com/docs/commerce/pwa-kit-managed-runtime/guide/env-var-feature-access.html

You can also use the MRT editor:

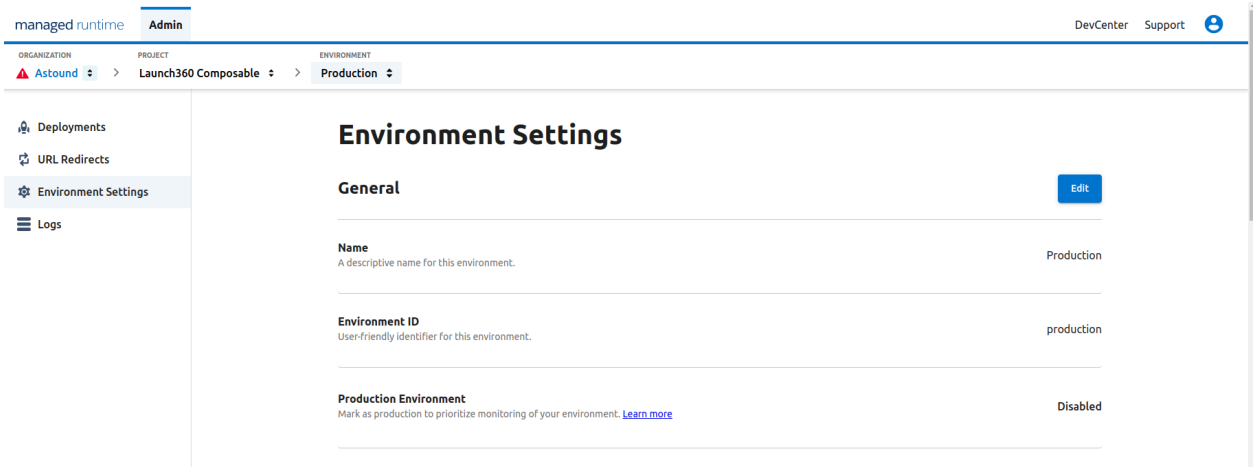


Proxy Configs

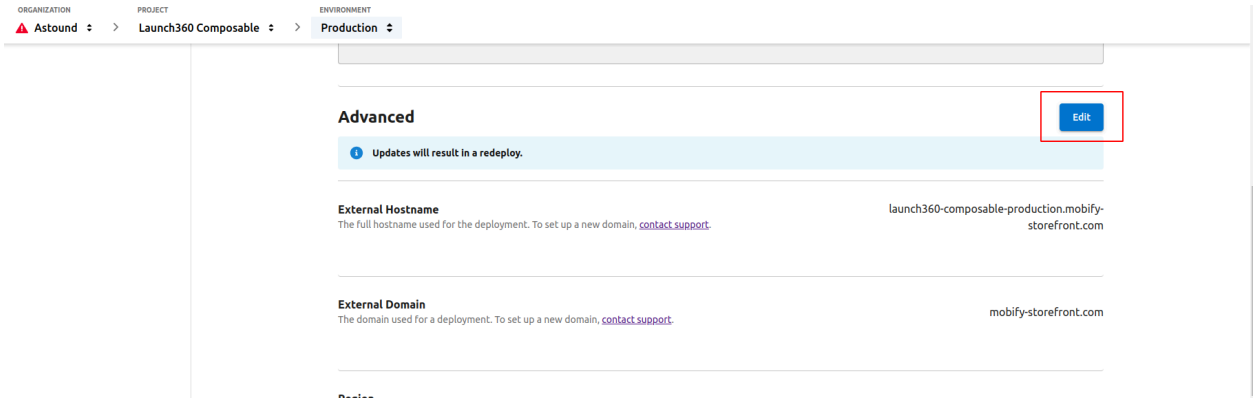
Now we need to set up the proxy configs. While in the root directory, open **./config/default.js** file and scroll to **proxyConfigs** object key:

```
module.exports = {
    ssrShared: [
        /*.json
    ],
    // Additional parameters that configure Express app behavior.
    ssrParameters: {
        ssrFunctionNodeVersion: '18.x',
        proxyConfigs: [
            {
                host: 'kv7kzm78.api.commercecloud.salesforce.com',    "commercecloud": Unknown word.
                path: 'api'
            },
            {
                host: 'zzrk-005.dx.commercecloud.salesforce.com',    "zzrk": Unknown word.
                path: 'ocapi'    "ocapi": Unknown word.
            },
            {
                host: 'staging-core.dxpapi.com',    "dxpapi": Unknown word.
                path: 'bloomreach',
                protocol: 'https'
            },
            {
                host: 'cdn.brcdn.com',    "brcdn": Unknown word.
                path: 'bloomreach-cdn',      Madison Dickson, 5 months ago • Refactor Pixel tracking. Still need
                protocol: 'https'
            },
            {
                host: 'staging-suggest.dxpapi.com',    "dxpapi": Unknown word.
                path: 'bloomreach-autosuggest',
                protocol: 'https'
            },
            {
                host: 'pathways-staging.dxpapi.com',    "dxpapi": Unknown word.
                path: 'bloomreach-recommends',
                protocol: 'https'
            }
        ]
    }
}
```

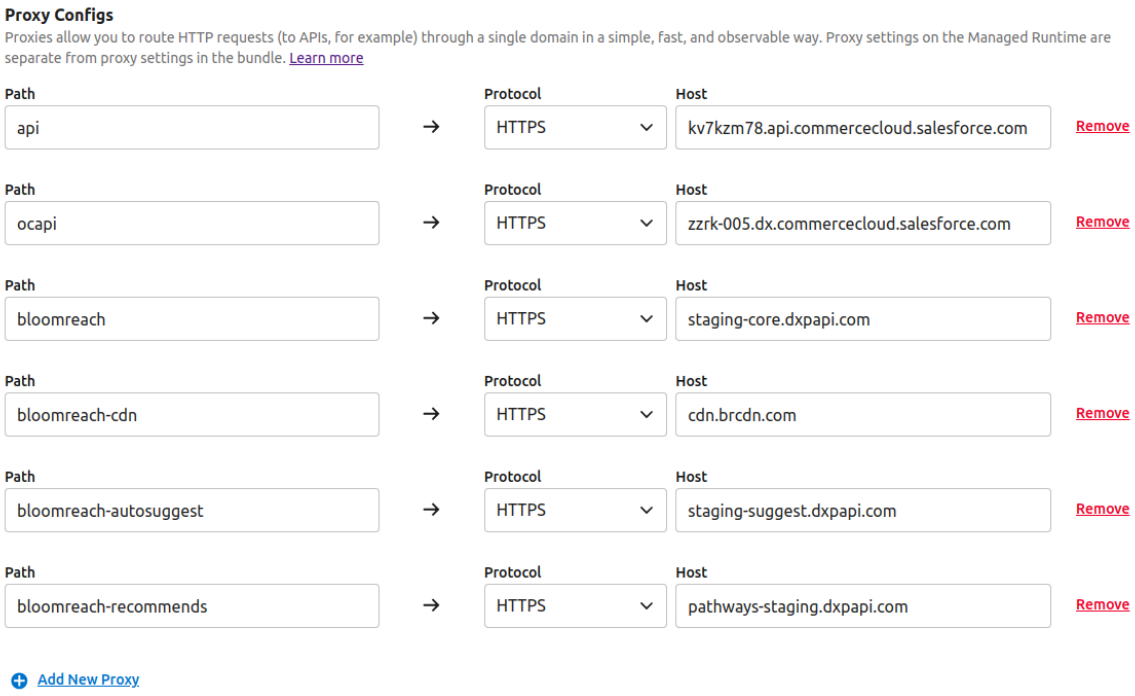
Now open Managed Runtime and open the environment you wish to deploy to and go to its settings.



Next scroll to the “Advanced” section and click on “Edit” button



Add each proxy config as it is in the **./config/default.js** file and click the "Update" button afterwards:



NOTE:
We are using the staging urls here, namely:
staging-core.dxpapi.com
staging-suggest.dxpapi.com
pathways-staging.dxpapi.com

The production ones are the following:
core.dxpapi.com
suggest.dxpapi.com
pathways.dxpapi.com

if you want to use staging urls locally but production in the MRT you will need to set some environment variables as well. Here is more information:
https://developer.salesforce.com/docs/commerce/pwa-kit-managed-runtime/guide/proxying-requests.html#override-proxy-configurations-with-environment-variables

Deploy a build

Remember the common variable that we used initially in Server Environment Variables section, namely:
project_slug="<the-name-of-the-project-in-the-mrt-environment>"
target_slug="<environment-id>"

We will need them again.
You can deploy a build by going to the root directory of the project and executing the following command in your terminal:

```
npm run push -- -s $project_slug --message "PWA kit build"  --target "$target_slug"
```

Log errors

You can log errors and debug by going to the root directory of the project and executing the following command in your terminal:

```
npx @salesforce/pwa-kit-dev@latest tail-logs --project $project_slug --environment $target_slug
```

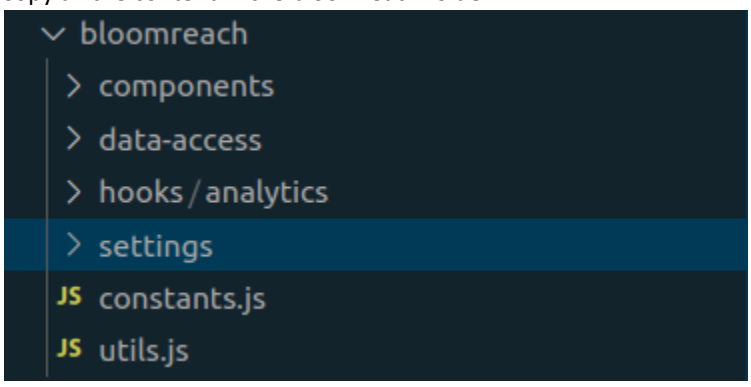
4. Implementing in already existing PWA

You still will need to follow all the steps in the above section, including:

- Set-up core cartridge
- Local Environment Variables
- Server Environment Variables
- Proxy Configs

Initial Configs

Copy all the content in the bloomreach folder:



Next open **./jsconfig** and add the following line:

```
// for development convienience, this file helps IDE's find relative paths prefixed with    "convienience":
// the `@salesforce/retail-react-app' for Template Extensibility projects
// If any aliases are defined in the webpack.config.js file, they need to be mirrored here.

// jsconfig.json
{
    "compilerOptions": {
        "target": "ES6",
        "baseUrl": "./",
        "paths": {
            // Add @salesforce files for better DX
            "@salesforce/*": ["./node_modules/@salesforce/*", "./node_modules/@salesforce/*/index"],
            // Add local bloomreach override
            "@bloomreach/*": ["./bloomreach/*", "./bloomreach/*/index"],
            "~/*": ["./overrides/*", "./overrides/*/index"]
        }
    }
}
```

Next open **./webpack.config**.js the important lines are highlighted so add the following and their dependencies:

```
/* eslint-disable @typescript-eslint/no-var-requires */
const configs = require('@salesforce/pwa-kit-dev/configs/webpack/config.js')
const path = require('path')

configs.forEach((c) => {
    c.resolve = {
        ...c.resolve,
        alias: {
            ...c?.resolve?.alias,
            '@bloomreach': path.resolve(__dirname, './bloomreach'),
            '~': path.resolve(__dirname, './overrides')
        }
    }
})
module.exports = configs
```

The Bloomreach functionality can be split into the following:
Product Search, Content Search, Search Suggestions, Autocorrect, Redirects, Recommendations and Pixel Analytics

Product Search

We use template extensibility for the product listing page since we will need to override the default behavior. If you are using the default search page template you will need to implement it. Instructions on how to do this are available here:
https://developer.salesforce.com/docs/commerce/pwa-kit-managed-runtime/guide/customize-a-page.html

Open **./overrides/app/pages/product-list/index.jsx**
You will need to add all the code and its dependencies from “Query Actions” comment section until the “Content Listing” section

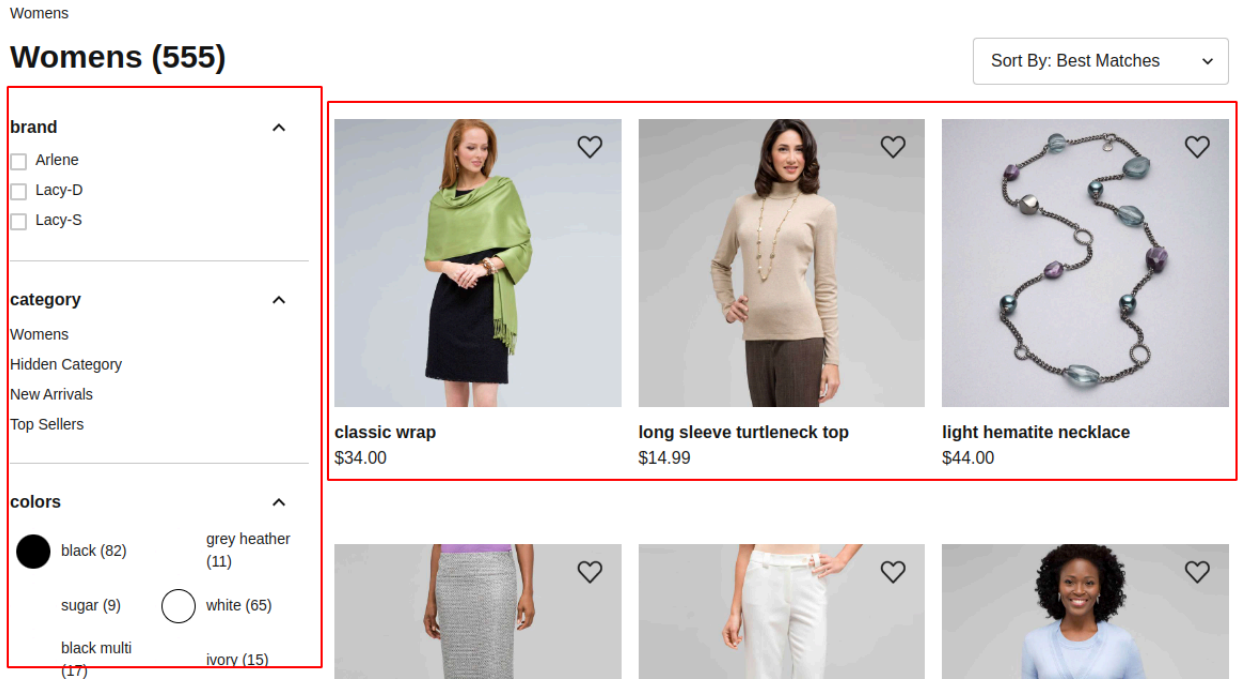
```
/***************** Query Actions ****************/
const {currency} = useCurrency()

const queryParams = {
    search_type: isSearch ? 'keyword' : 'category',
    q: isSearch ? searchQuery : `${params.categoryId}`,
    rows: DEFAULT_LIMIT_VALUES[0],
    start: searchParams?.offset || 0,
    view_id: currency.toLowerCase()
}

if (!isEmpty(searchParams?.refine)) {
    const {price, ...attributesExceptPrice} = searchParam

    // Price
    if (price) {
        queryParams.fq = `price:${price}`
    }

    // Other than price attributes
```

```
/***************** Content Listing ***************/
const contentSettings = {
    q: isSearch ? searchQuery : '',
    rows: DEFAULT_LIMIT_VALUES[0],
    start: searchParams?.offset || 0
}

const {isLoading: isContentLoading, data: contentData} = useGetContent(contentSettings, {
    enabled: isSearch
})
```

The Bloomreach hook that is responsible for retrieving the product data is:

```
const {
    isLoading,
    isRefetching,
    status,
    data: productData
} = useGetProductsByFilters(queryParams)
```

You should see product results for category and search as well as their refinements if implemented correctly:



Content Search

Open **./overrides/app/pages/product-list/index.jsx**
You will need to add all the code and its dependencies from “Content Listing” comment section until the “Error Handling” section

```
/**************** Content Listing ****************/
const contentSettings = {
    q: isSearch ? searchQuery : '',
    rows: DEFAULT_LIMIT_VALUES[0],
    start: searchParams?.offset || 0
}

const {isLoading: isContentLoading, data: contentData} = useGetContent(contentSettings, {
    enabled: isSearch
})
```

```
/**************** Error Handling ****************/
const errorStatus = error?.response?.status
```

Also you will need to alter the code in the render function to match this:

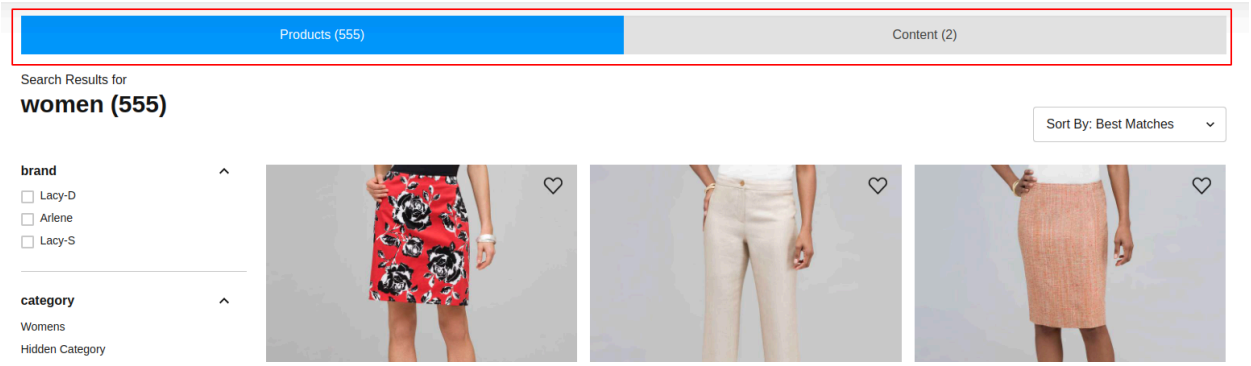
```
{showNoResults ? (
    <EmptySearchResults searchQuery={searchQuery} category={category} />
) : (
    <>
        {/* If it is a search we need a way to switch between products and content */}
        {isSearch && contentResultsCount > 0 && ( …
        )}

        {/* Custom content result handling */}
        {isShowingContentSearch ? (
            <ContentList
                headerSettings={{
                    searchQuery,
                    searchMeta: {
                        didYouMean: contentData?.did_you_mean,
                        total: contentResultsCount
                    },
                    isLoading: isContentLoading
                }}
                results={contentData?.response?.docs}
                paginationSettings={{
                    currentURL: basePath,
                    urls: contentPageUrls
                }}
            />
        ) : (
            <> …
            </>
        )}
    </>
)}
```

Note <ContentList/> component will handle listing of the content results

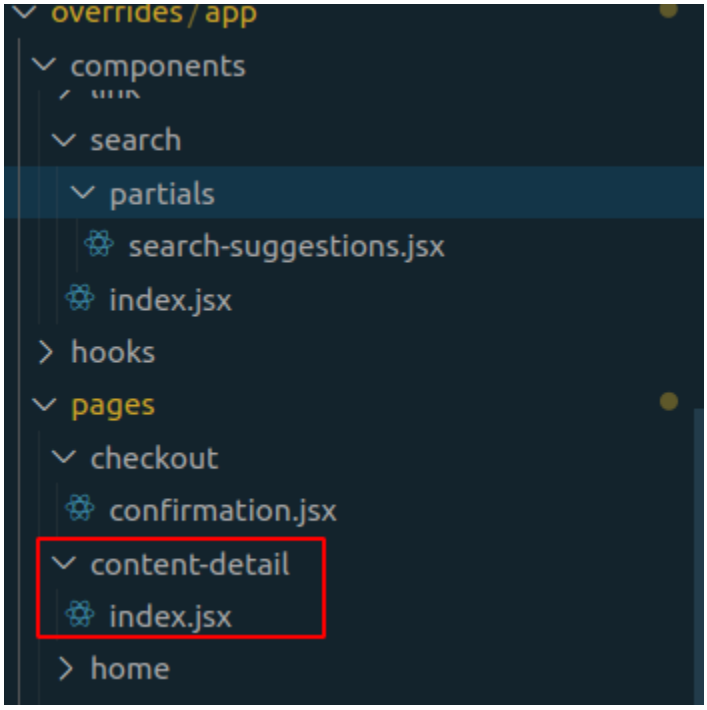
The Bloomreach hook that is responsible for retrieving the content data is:

```
const {isLoading: isContentLoading, data: contentData} = useGetContent(contentSettings, {
    enabled: isSearch
})
```

When searching and you have content results you should see this



To handle single pages you need to add **./overrides/app/pages/content-detail** file to your project directory:



As well as modifying the **./overrides/app/routes.jsx** to handle content single page:

```
const ContentDetail = loadable(() => import('./pages/content-detail'))
```

```
{
    path: '/content/:contentId',
    component: ContentDetail
},
```

Search Suggestions

If you already have **./overrides/app/components/search/index.jsx** file, open it and add the following:

```
// Bloomreach import
import SearchSuggestions from './partials/search-suggestions'
import {useGetAutosuggestions} from '@bloomreach/data-access/client/queries'
import {getQueryParamsSettings} from '@bloomreach/data-access/client/queries/helpers'
import {apiURL} from '@bloomreach/constants'
import {useLocation} from 'react-router'
import {getAppOrigin} from '@salesforce/pwa-kit-react-sdk/utils/url'
import {navigateRedirect} from '@bloomreach/utils'
```

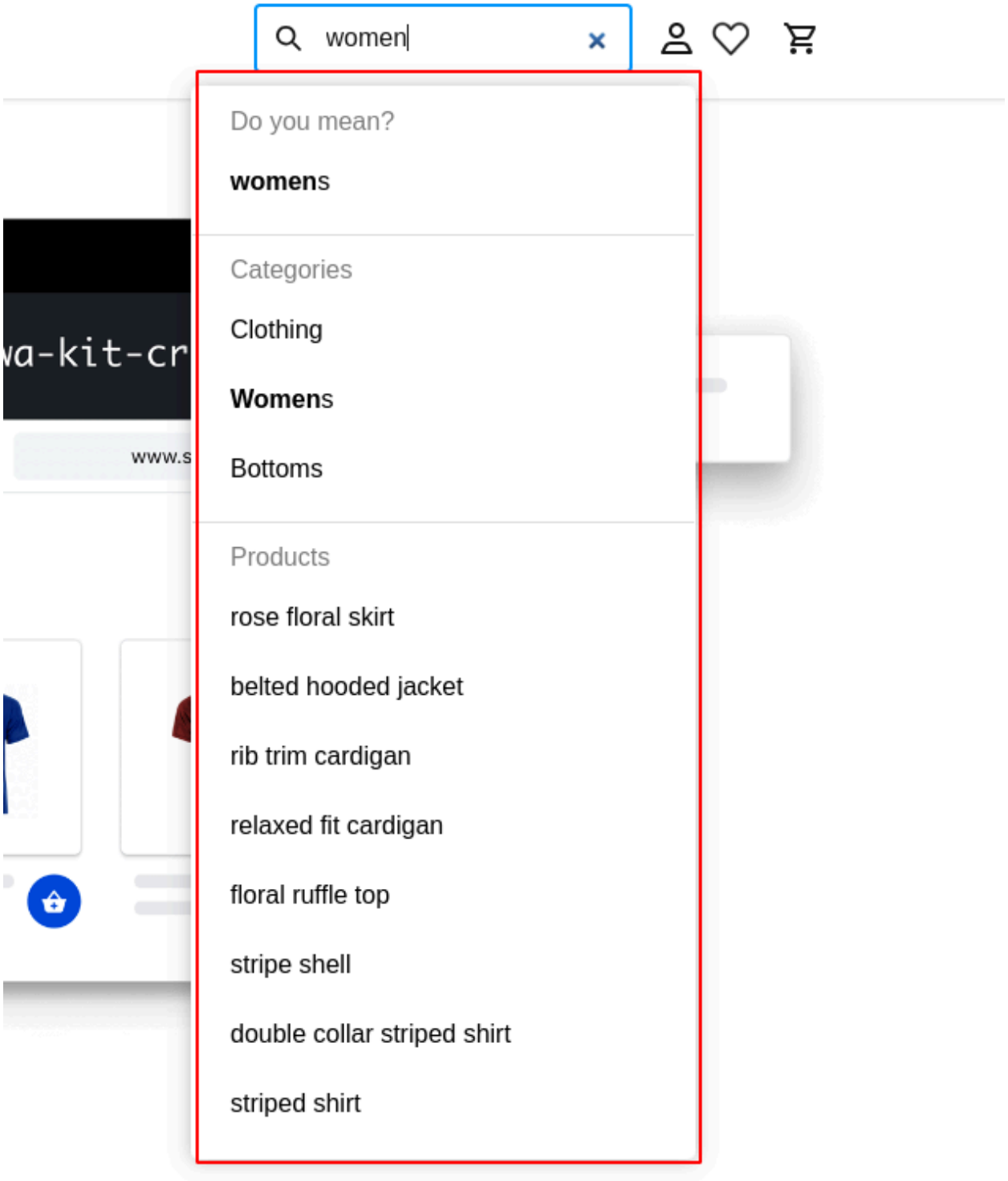
```
// Bloomreach Search Suggestions
const location = useLocation()
const appOrigin = getAppOrigin()

const searchSuggestion = useGetAutosuggestions(
    {
        q: searchQuery
    },
    {enabled: searchQuery?.length >= RECENT_SEARCH_MIN_LENGTH}
)
// End of Bloomreach Search Suggestions
```

Note this includes the **./overrides/app/components/search/partials/search-suggestions.jsx** file

The hook responsible for the date is:

```
const searchSuggestion = useGetAutosuggestions(
    {
```

Successfully integrating it would look like this:



Autocorrect

First you need to add the **./overrides/app/pages/product-list/partials/page-header.jsx** component and import it

```
import PageHeader from './partials/page-header'
```

in the **./overrides/app/pages/product-list/index.jsx** file and use it like this:

```
{/* Custom content result handling */}
{isShowingContentSearch ? (…
) : (
    <>
        {/* Below are the contents of the default ProductList template */}
        <AbovePageHeader />
        {/* Header */}
        <Stack
            display={{base: 'none', lg: 'flex'}}
            direction="row"
            justify="flex-start"
            align="flex-start"
            spacing={4}
            marginBottom={6}
        >
            <Flex align="left" width="287px">
                <PageHeader
                    searchQuery={searchQuery}
                    category={category}
                    productSearchResult={productSearchResult}
                    isLoading={!isLoadingFinished}
                />
            </Flex>

            <Box flex={1} paddingTop={'45px'}>…
            </Box>
            <Box paddingTop={'45px'}>…
            </Box>
        </Stack>

        <HideOnDesktop>
            <Stack spacing={6}>
                <PageHeader
                    searchQuery={searchQuery}
                    category={category}
                    productSearchResult={productSearchResult}
                    isLoading={isLoading}
                />
```

Note this is both for Desktop and mobile.

The actual data come from this hook in the variable **productSearchResult**:

```
const {
    isLoading,
    isRefetching,
    status,
    data: productData
} = useGetProductsByFilters(queryParams)
```

Redirects

If you already have **./overrides/app/components/search/index.jsx** file, open it and the following functions with all its dependencies:

```
// Bloomreach redirect handler
const handleSubmit = async (event) => {
    event.preventDefault()

    let searchText = searchInputRef.current.value.trim()

    // Replace the placeholders with actual values
    const queryParams = new URLSearchParams({
        ...getQueryParamsSettings({appOrigin, location}),
        request_type: 'search',
        search_type: 'keyword',
        fl: 'pid',
        q: searchText,
        // The number of products is not important here
        rows: '1',
        start: '0'
    })

    const apiUrlWithParams = `${apiURL}?${queryParams}`

    try {
        const response = await fetch(apiUrlWithParams)

        if (!response.ok) {
            throw new Error(`HTTP error! Status: ${response.status}`)
        }

        const data = await response.json()
        if (data?.keywordRedirect) {
            navigateRedirect(navigate, data?.keywordRedirect?.['redirected url'])
        } else {
            onSubmitSearch(event)
        }
    } catch (error) {
        console.error('Error fetching data:', error)
    } finally {
        searchInputRef.current.value = ''
    }
}
```

As well as passing it as a handler to the form:

```
return (
    <Box>
        <Popover isOpen={isOpen} isLazy initialFocusRef={searchInputRef}>
            <PopoverTrigger>
                <form onSubmit={handleSubmit}>
                    <HStack>
                        <InputGroup>
                            <InputLeftElement pointerEvents="none">
                                <SearchIcon />
                            </InputLeftElement>
                            <Input
                                autoComplete="off"
                                id="search-input"
                                onChange={(e) => onSearchInputChange(e)}
                                onFocus={() => shouldOpenPopover()}
                                onBlur={() => setIsOpen(false)}
                                type="search"
                                ref={searchInputRef}
                                {...props}
                                variant="filled"
                            />
                        </InputGroup>
```

You possibly already have(if you have implemented the Bloomreach Product Search), but it must be mentioned that you will need to also have the useEffect function in the **./overrides/app/pages/product-list/index.jsx** file:

```
useEffect(() => {
    if (!isServer) {
        if (!productData?.keywordRedirect) {
            if (status === 'success' && !isRefetching) {
                setProductSearchResult(productData)
                setIsLoadingFinished(true)
            } else {
                setIsLoadingFinished(false)
            }
        } else {
            navigateRedirect(navigate, productData?.keywordRedirect?.['redirected url'])
        }
    }
}, [status, productData, isRefetching])
```

Success should be indicated by correctly redirecting when submitting the search form with the term configured in the dashboard settings

Recommendations

Open your **./overrides/app/pages/product-detail/index.jsx** file and import the following component:

```
import Recommended from '@bloomreach/components/Recommended/'
```

And use it, for example like that:

```
{/* Product Recommendations */}
<Stack spacing={16}>
    {/* Removed other Product Recommendations in favor of Specifically Bloomreach recommender
    component below */}
    <Recommended productId={product?.id} mx={{base: -4, md: -8, lg: 0}} />
</Stack>
```

Success would look something like this when you open PDP:

Pixel Analytics

In the PWA we have 6 page type events: homepage, product, category, search, content, conversion.

Generally In order to trigger page event you need to import the following:

```
import {useBloomreachAnalytics} from '@bloomreach/hooks/analytics'
```

Get the track method:

```
  const {track} = useBloomreachAnalytics()
```

And for add the object date in useEffect:

Please, note that each track object must have “ptype” key in order to be valid and be registered as Page View.

The easiest way to confirm that is working is installing the testing extension here:
https://documentation.bloomreach.com/discovery/docs/validating-pixels

Here there are examples of how to integrate your own.

Homepage

```
useEffect(() => {
    einstein.sendViewPage(pathname)

    track({
        ptype: 'homepage',    "ptype": Unknown word.
        title: 'Home Page'
    })
}, [])
```

Product

```
/*************** Bloomreach Pixel ***************/
useEffect(() => {
    if (!product) return

    track({
        ptype: 'product',    "ptype": Unknown word.
        title: product.pageTitle,
        prod_id: product.id,
        prod_name: product.name
    })
}, [product?.id])
```

Note this depends here on product.id

Category

```
/*************** Bloomreach Pixel ***************/
useEffect(() => {
    if (!category) return

    const cat = category.parentCategoryTree?.reduce((acc, el, index, array) => {
        const delim = index === array.length - 1 ? '' : '|'    "delim": Unknown word.
        return acc + el?.name + delim    "delim": Unknown word.
    }, '')

    track({
        ptype: 'category',    "ptype": Unknown word.
        title: category?.pageTitle,
        cat_id: params.categoryId,
        cat
    })
}, [category?.id])
```

Search

```
useEffect(() => {
    if (!isSearch) return

    track({
        ptype: 'search',    "ptype": Unknown word.
        title: searchQuery,
        search_term: searchQuery,
        catalogs: [{name: CATALOG_NAME}]
    })
}, [isSearch, searchQuery])
```

Content

This is added in the **./overrides/app/pages/content-detail/index.jsx**

```
useEffect(() => {
    track({
        ptype: 'content',    "ptype": Unknown word.
        title: contentId,
        item_id: contentId,
        item_name: contentId,
        catalogs: [{name: CATALOG_NAME}]
    })
}, [])
```

Conversion

```
useEffect(() => {
    if (order) {
        track({
            ptype: 'conversion',    "ptype": Unknown word.
            is_conversion: 1,
            basket_value: order?.orderTotal,
            order_id: order?.orderNo,
            currency: order?.currency,
            basket: {
                items: order?.productItems.map((el) => {
                    return {
                        prod_id: el?.productId,
                        sku: el?.itemId,
                        name: el?.productName,
                        quantity: el?.quantity,
                        price: el?.price
                    }
                })
            }
        })
    }
}, [order])
```

Note that this is added in the **./overrides/app/pages/checkout/confirmation.jsx** component