# SEO Implementation

Search engine optimization (SEO) is the process of affecting the online visibility of a website or a web page in a web search engine's unpaid results.

To enable SEO in Virtual Branch Phase 2 we have handled the following things:

1. Angular Universal Starter Project:

The project is based on Universal Starter kit provided by Angular. This is the base to start SEO implementation in Angular. This helps in server side rendering of the Angular app, which generates a static page on the node server and then presents the same on the browser.

To achieve this the node server makes webservice calls required for the page to be rendered and binds data to the page.

Following changes are made in “home.service” to fetch data from node:

* URL is formed using the “AppConfig.settings.env.node” which holds the tomcat url.
* Headers are created using the browser cookies in function “parseCookies”. The cookies are read on the node server using the following code:

let req = this.injector.get(REQUEST);

* The above code helps to get SM\_USER after login to fetch user specific data.
* “parseCookies” is an important function to create server side rendered page as it creates the headers which are read by the webservices to give response
* createUrl is the function to create specific URL if the page is rendered on node server or browser

Node Server Start: In order to start the angular app on node server, the file “local.js” is used. This file holds the code to start the node server in SSL enabled mode.

1. Resolvers:

For SEO it is very important that the application view page source gives proper result with entire content. For this to work we have used **Resolvers** for every page**.**Using Resolvers the data will be fetched before navigating to the component. Hence in the component we do not need to fetch data from webservice in the constructor. But within the page if data changes then we need to refresh by calling the webservice.

The resolvers are present in the “resolvers” folder.

1. Browser Specific API

SEO build creates the page on the node server and not on the browser. Hence browser objects like “window”,”sessionStorage” are not accessible. However if we still need to use these objects we need to use the following code format which exists in our app currently:

import { [PLATFORM\_ID](https://angular.io/api/core/PLATFORM_ID), [APP\_ID](https://angular.io/api/core/APP_ID), [Inject](https://angular.io/api/core/Inject) } from '@angular/core';

import { [isPlatformBrowser](https://angular.io/api/common/isPlatformBrowser" \t "_blank) } from '@angular/common';

constructor( @[Inject](https://angular.io/api/core/Inject)([PLATFORM\_ID](https://angular.io/api/core/PLATFORM_ID)) private platformId: Object,@[Inject](https://angular.io/api/core/Inject)([APP\_ID](https://angular.io/api/core/APP_ID)) private appId: string)

{

const [platform](https://angular.io/api/core/testing/TestBed#platform) = [isPlatformBrowser](https://angular.io/api/common/isPlatformBrowser" \t "_blank)(platformId) ? 'in the[browser](https://angular.io/api/animations/browser)' : 'on the server';

console.log(`Running ${[platform](https://angular.io/api/core/testing/TestBed#platform)} with appId=${appId}`);

if(platform)

{

      //Access browser objects

}

Else

{

      //option to be executed if it is server

}

}

1. Meta Tag:

For SEO to work, meta tags are added in the page source file:

e.g.

<meta name="keywords" content="Johnson Controls, Virtual Branch, Johnson Controls Contact US, Selectors, Product &amp; Solutions"><meta name="description" content="Johnson Controls Virtual Branch">

For product pages the content for keywords and description are fetched from the respective product services. For all other generic pages it is configured using the seo.json file

* 1. Seo.json:

{

"seo":{

"keywords":"Johnson Controls, Virtual Branch, Johnson Controls Contact US, Selectors, Product & Solutions",

"description":"Johnson Controls Virtual Branch"

}

}

* 1. Webservice: The Product services have the key “seo\_keywords” and “seo\_description”. These are binded to the page at runtime in series.component.ts & item.component.ts files

1. Multiple Language Support using hreflang:

The app supports multiple languages. In order to provide the required support we have added canonical tags in every page for all supported languages.