**Setting up local development environment for JSS and Sitecore**

**Background:**

For this documentation, we would be working with **Sitecore JSS**. **Sitecore JavaScript SDK (JSS)** is a set of npm packages that enable implementing Sitecore apps using modern JavaScript frameworks.

If you are starting with **Sitecore JSS** app development, It is recommended that you also read this [**documentation**](https://jss.sitecore.com/docs/introduction/why-jss)

**Prerequisites:**

1. Install Sitecore 9.3.
2. Sitecore 9.3 environment up and running.
3. Sitecore 9.3 environment must install [JSS (Sitecore JSS 13.x for Sitecore 9.3)](https://dev.sitecore.net/Downloads/Sitecore_JavaScript_Services/130/Sitecore_JavaScript_Services_1300.aspx) in order to support JSS apps.
4. Create ***API Key*** *(a one time task)* for your app in Sitecore at ***"/sitecore/system/Settings/Services/API Keys/"*** path.
5. nodejs 14.11 or later.
6. Any IDE such as VSCode.
7. JSS Cli node module package (Check the documentation here for recommended version of JSS for Sitecore 9.3).

**Install required prerequisites:**

1. Sitecore 9.3 access Url (Considering Sitecore 9.3 is installed and configured to use JSS): **https://sc9.local.com/sitecore/login**
2. Create ***API Key*** for your app in Sitecore.
3. Download and install nodejs. You can download nodejs from [here](https://nodejs.org/download/release/v14.11.0/). You can of course manage these versions via [nvm](https://github.com/nvm-sh/nvm). If you want to use nvm for windows, check this [out](https://github.com/coreybutler/nvm-windows/releases/tag/1.1.8).
4. Download and install VSCode. You can download VSCode from [here](https://code.visualstudio.com/download)
5. Install JSS Cli, refer JSS Cli documentation [here](https://jss.sitecore.com/docs/fundamentals/cli).
   1. **Sitecore JSS** supports developing [React](https://jss.sitecore.com/docs/client-frameworks/react/react-overview), [Angular](https://jss.sitecore.com/docs/client-frameworks/angular/angular-overview), [Vue](https://jss.sitecore.com/docs/client-frameworks/vue/vue-overview) and [Nextjs](https://jss.sitecore.com/docs/nextjs/getting-started-nextjs/why-nextjs) applications with Sitecore. JSS provides boilerplate templates such as "react", "angular", "vue" and "nextjs" to get started with your frontend project. **For this documentation, we will be using "react" template to get** [**started**](https://jss.sitecore.com/docs/client-frameworks/getting-started/quick-start)**.**
   2. To install **Sitecore JSS** CLI node module package's latest version use following command:

|  |
| --- |
| **$ npm install -g @sitecore-jss/sitecore-jss-cli** |

* 1. To install a specific version of JSS CLI for example 13.0.0, use the following command:

|  |
| --- |
| ***$ npm install -g @sitecore-jss/sitecore-jss-cli@13.2.2*** |

**Note:** As per **Sitecore JSS** documentation, it is recommended to use JSS CLI version 13.x. Check here for more details: [**Sitecore JSS 13.2 for Sitecore 9.3**](https://jss.sitecore.com/release-notes)

**Before we get started, let's look at what would be the developer workflow after required tools and stuffs are ready.**

**What is the developer workflow?**

When it comes to developer workflow, there are two approaches that comes into picture.

* Sitecore First
  + Steps:
    - Create API key pertaining to your <appName> in Sitecore at "/sitecore/system/Settings/Services/API Keys/" path. **[This prerequisite step should have been completed as described above]**
    - Setup the app with initial data setup in Sitecore and handover API key and app location to tenant.
    - Create a boilerplate app based on your preferred template (react, vue, angular and nextjs)
    - Setup your local jss app to generate the configs (***jss setup*** cli command will be used to do this, we will talk about it later in this documentation).
      * scjssconfig.json (contains configs to connect to sitecore for running connected mode and app/items deployment)
      * <appName>.deploysecret.config (contains a random key for app/items deployment)
      * <appName>.config (contains Site definitions, rendering host configuration, Jss app configuration etc.)
    - Deploy the configs to Sitecore server.
    - Start running connected mode via [**jss cli**](https://jss.sitecore.com/docs/fundamentals/cli).
* Code First (***Recommended in most cases***)
  + Steps:
    - Create a boilerplate app based on your preferred template (react, vue, angular and nextjs)
    - Get started with disconnected mode and make your app ready to move to Sitecore for transition into connected mode (We will throw light on these modes later in this documentation).
    - Create API key pertaining to your <appName> in Sitecore at "/sitecore/system/Settings/Services/API Keys/" path and handover to tenant. This is a one time setup.**[This prerequisite step should have been completed as described above]**
    - Setup your local jss app to generate the configs (***jss setup*** cli command will be used to do this, we will talk about it later in this documentation).
      * scjssconfig.json (contains configs to connect to sitecore for running connected mode and app/items deployment)
      * <appName>.deploysecret.config (contains a random key for app/items deployment)
      * <appName>.config (contains Site definitions, rendering host configuration, Jss app configuration etc.)
    - Deploy the configs to Sitecore server.
    - Deploy your app to sitecore using [**jss cli**](https://jss.sitecore.com/docs/fundamentals/cli) (***jss deploy*** cli command will be used to do this, we will talk about it later in this documentation).
    - Start running connected mode via [**jss cli**](https://jss.sitecore.com/docs/fundamentals/cli).

***Which one to choose from the two? →*** It depends the team requirement as to how team wants to proceed, whether they want to setup app, basic templates and content within Sitecore initially and then start connecting local code base to Sitecore for further development (also called connected mode development) or they want to get started locally first (also called disconnected mode development).

Now let's assume you choose "***Sitecore First***" workflow, its a more traditional way to use **Sitecore JSS**.  In this workflow the JSS app consumes data from Sitecore, but has no responsibility for defining the structure of that data, which components are registered with Sitecore, etc. In this workflow all developers will need their own development instance of Sitecore, and the JSS site will connect to it to acquire its content and layout data.

It is most suited when:

* Frontend developer with experience in Sitecore and they know creating complex content or backend. (e.g. adding a JSS site into an existing Sitecore instance)
* JSS apps will be deployed using the same deployment process. (e.g. same source repository and Sitecore CI/CD process)

Learn more about "Sitecore First" workflow [here](https://jss.sitecore.com/docs/fundamentals/dev-workflows/sitecore-first).

When it comes to "**Code First**" workflow, you start with disconnected mode and then transition to connected mode. You create basic templates/dictionary/content etc. to get started and deploy your app (along with templates/dictionary/content etc.) to Sitecore your self. No manual intervention needed except one time config. In a code-first workflow the JSS app creates a manifest of its content data and data schema, from a set of files. This enables the JSS app to execute with local mock content, without a Sitecore instance. In this mode the JSS app is the master copy of all artifacts. The manifest gets imported into Sitecore which creates the necessary structures to support the app when you deploy the app to Sitecore.

It is most suited when:

* For early prototyping of a design without connecting to Sitecore.
* Frontend developers who do not have their own Sitecore instances.
* Frontend developers are not Experienced in Sitecore and they are well versed in Javascript development.
* JSS apps will be deployed independent of Sitecore CI/CD process.

Learn more about "Code First" workflow [here](https://jss.sitecore.com/docs/fundamentals/dev-workflows/code-first).

**Eventually you move to "Sitecore First" workflow when everything is setup in Sitecore via "Code First" workflow.**

**Sitecore JSS Application Modes**

There are several JSS application modes pertaining to Development and Production. For this documentation which is specific to development, the two basic modes are:

1. **Disconnected mode**
   1. You don't need Sitecore to get started
   2. Local host and local data source (content) is used. It is available with boilerplate app under "*<appName>/data*" directory.
2. **Connected mode**
   1. You need sitecore to connect
   2. Local host and remote data source is used. Remote data source refers to Sitecore content data.

**To learn more about the various modes in detail, refer this** [**documentation**](https://jss.sitecore.com/docs/fundamentals/application-modes).

**Getting Started**

As aforementioned, we will be using JSS's ***react***template generate the boilerplate app, deploy the app to Sitecore and run the app in connected mode. We will using "***Code First***" approach for this documentation. Assuming you have already installed and completed all pre-requisites. Let's get started.

**Step 1:** **Create a boilerplate app using react template** (Assuming you have already installed JSS CLI node module as described in pre-requisites above.)

Use the following commands to create the boilerplate app :

|  |
| --- |
| **$ jss create jss-demo-app react** |

To create boilerplate app for a specific JSS version, e.g.: 15.0.2 (recommended as per Sitecore documentation), use the following commands:

|  |
| --- |
| **$ jss create jss-demo-app react -b release/15.0.0** |

*Where, 'jss-demo-app' is the appName and 'react' is the template. Checkout* [*this*](https://jss.sitecore.com/docs/client-frameworks/getting-started/quick-start) *documentation for more info. This boilerplate app was bootstrapped with help of*[*https://github.com/facebook/create-react-app*](https://github.com/facebook/create-react-app)*.*

**Step 2: Start with disconnected mode**

Use the following commands to start the disconnected mode:

|  |
| --- |
| **$ cd jss-demo-app**  **$ jss start**  **OR**  **$ npm run start** |

Sitecore is not required to be installed, and content data is pulled from static local sources. Disconnected development is appropriate when no Sitecore instance is available, or when frontend developers do not wish to run a local copy of Sitecore. Content data is mocked using local files (json/yaml/js) instead of a Sitecore instance. You local app connects to mock sitecore proxy (you don't need to setup this, jss framework does it all for you) and delivers the mocked content data.

At this point you would cleanup your boilerplate app, add/remove templates/components etc. and test it locally in disconnected mode and make it ready for transition to connected mode.

**Step 3: Keep the Sitecore 'API key' handy, so that you can use it during config setup on the next step.**

**Step 4: Setup the jss configs (assuming you already have the API key for your app, check the prerequisites) for transition to connected mode.**

* Use the following commands to setup jss configs:

|  |
| --- |
| **$ jss setup** |

* Provide the details as asked on the prompt, keeping in mind that you need to connect to Sitecore 9.3. At the end it would generate two files. Example:

|  |
| --- |
| **Is your Sitecore instance on this machine or accessible via network share? [y/n]: y Path to the Sitecore folder (e.g. c:\inetpub\wwwroot\**[**my.siteco.re**](http://my.siteco.re)**):  Sitecore hostname (e.g.** [**http://myapp.local.siteco.re**](http://myapp.local.siteco.re)**; see /sitecore/config; ensure added to hosts): https://sc9.local.com Sitecore import service URL [https://sc9.local.com/sitecore/api/jss/import]: https://sc9.local.com/sitecore/api/jss/import Sitecore API Key (ID of API key item): {A0F9FC32-EBE5-46A4-A556-1FAB7F35DCEA} Please enter your deployment secret (32+ random chars; or press enter to generate one): Deployment secret has been generated. Ensure the JSS app config on the Sitecore end has the same secret set. Deploy secret Sitecore config written to C:\sitecoreFrontend\jss-demo-app\sitecore\config\jss-demo-app.deploysecret.config Ensure this configuration is deployed to Sitecore. JSS connection settings saved to C:\sitecoreFrontend\jss-demo-app\scjssconfig.json**  **NEXT STEPS \* Ensure the hostName in /sitecore/config/\*.config is configured as sc9.local.com, and in hosts file if needed. \* Deploy your configuration (i.e. 'jss deploy config') \* Deploy your app (i.e. 'jss deploy app -c -d') \* Test your app in integrated mode by visiting https://sc9.local.com** |

* **"*scjssconfig.json*"** at the root of the project/app with content based on inputs provided. Example config:

|  |
| --- |
| **{**  **"sitecore": {**  **"instancePath": "",**  **"apiKey": "{A0F9FC32-EBE5-46A4-A556-1FAB7F35DCEA}",**  **"deploySecret": "8zd8ca6rfdm7mygupa3pg6yq7buvcfdkceuz6x9g43f4",**  **"deployUrl": "https://sc9.local.com/sitecore/api/jss/import",**  **"layoutServiceHost": "https://sc9.local.com"**  **}**  **}** |

Note: Above config is just a sample and not for copying for your app. Generate the config for your app and compare with this example. It is just for reference.

* **"<appName>.deploysecret.config"** (in this case ***jss-demo-app.deploysecret.config***) at **"/sitecore/config"** path in the project/app with content based on inputs provided. Example config:

|  |
| --- |
| **<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">**  **<sitecore>**  **<javaScriptServices>**  **<apps>**  **<app name="jss-demo-app"**  **deploymentSecret="8zd8ca6rfdm7mygupa3pg6yq7buvcfdkceuz6x9g43f4" debugSecurity="false" />**  **</apps>**  **</javaScriptServices>**  **</sitecore>**  **</configuration>** |

Note: Above config is just a sample and not for copying for your app. Generate the config for your app and compare with this example. It is just for reference.

* When you create the boilerplate app, it by default generates another config **"*<appName>.config*"** (in this case ***jss-demo-app.config***) under ***"/sitecore/config"****.* Make sure you update the hostName (***hostName="sc9.local.com"***) to match the hostname under site definition settings. Example config:

|  |
| --- |
| **<!--**  **JSS Sitecore Configuration Patch File**  **This configuration file registers the JSS site with Sitecore, and configures the Layout Service**  **to work with it. Config patches need to be deployed to the Sitecore server.**  **Normally `jss deploy config` can do this for local development. To manually deploy, or to deploy via CI,**  **this file can be placed in the `App\_Config/Include` folder, or a subfolder of it, within the Sitecore site.**  **-->**  **<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/" xmlns:set="http://www.sitecore.net/xmlconfig/set/" xmlns:role="http://www.sitecore.net/xmlconfig/role/">**  **<sitecore>**  **<settings>**  **<!--**  **PROXY CONFIGURATION**  **When running JSS with the headless proxy (node-headless-ssr-proxy),**  **the original IP address of clients is obscured by the proxy.**  **This setting tells Sitecore to read the forwarded header the proxy sends,**  **thus making analytics track the correct original client IP address.**  **-->**  **<setting name="Analytics.ForwardedRequestHttpHeader" set:value="X-Forwarded-For" />**  **</settings>**  **<sites>**  **<!--**  **JSS Site Registration**  **This configures the site with Sitecore - i.e. host headers, item paths.**  **If your JSS app lives within an existing Sitecore site, this may not be necessary.**  **IMPORTANT: JSS sites ship in 'live mode', which makes development and testing easy,**  **but disables workflow and publishing. Before going to production, change the `database`**  **below to `web` instead of `master`.**  **-->**  **<site patch:before="site[@name='website']"**  **inherits="website"**  **name="jss-demo-app"**  **hostName="sc9.local.com"**  **rootPath="/sitecore/content/jss-demo-app"**  **startItem="/home"**  **database="master" />**  **</sites>**  **<javaScriptServices>**  **<apps>**  **<!--**  **JSS App Registration**  **The JSS app needs to be registered in order to support layout service and import services.**  **There are many available attributes, and they inherit the defaults if not explicitly specified here.**  **Defaults are defined in `/App\_Config/Sitecore/JavaScriptServices/Sitecore.JavaScriptServices.Apps.config`**  **NOTE: graphQLEndpoint enables \_Integrated GraphQL\_. If not using integrated GraphQL, it can be removed.**  **-->**  **<app name="jss-demo-app"**  **sitecorePath="/sitecore/content/jss-demo-app"**  **useLanguageSpecificLayout="true"**  **graphQLEndpoint="/api/jss-demo-app"**  **inherits="defaults"**  **/>**  **</apps>**  **<!--**  **IMAGE RESIZING WHITELIST**  **Using Sitecore server-side media resizing (i.e. the `imageParams` or `srcSet` props on the `<Image/>` helper component)**  **could expose your Sitecore server to a denial-of-service attack by rescaling an image with many arbitrary dimensions.**  **In JSS resizing param sets that are unknown are rejected by a whitelist.**  **Sets of image sizing parameters that are used in app components must be whitelisted here.**  **If a param set is not whitelisted, the image will be returned \_without resizing\_.**  **To determine the image parameters being used, look at the query string on the `src` of the rendered image, i.e. '/img.jpg?mw=100&h=72' -> mw=100,h=72**  **Note: the parameter sets defined here are comma-delimited (,) instead of &-delimited like the query string. Multiple sets are endline-delimited.**  **-->**  **<allowedMediaParams>**  **<!-- XML element name is arbitary, useful for organizing and patching -->**  **<styleguide-image-sample>**  **mw=100,mh=50**  **</styleguide-image-sample>**  **<styleguide-image-sample-adaptive>**  **mw=300**  **mw=100**  **</styleguide-image-sample-adaptive>**  **</allowedMediaParams>**  **</javaScriptServices>**  **<api>**  **<!--**  **Define the app's Sitecore GraphQL API endpoint**  **Note: this can be removed if you are not using GraphQL.**  **Note: the endpoint must be defined both for integrated and connected type GraphQL queries.**  **-->**  **<GraphQL>**  **<endpoints>**  **<jss-demo-appGraphQLEndpoint url="/api/jss-demo-app" type="Sitecore.Services.GraphQL.Hosting.DatabaseAwareGraphQLEndpoint, Sitecore.Services.GraphQL.NetFxHost" resolve="true">**  **<url>$(url)</url>**  **<enabled>true</enabled>**  **<enableSubscriptions>true</enableSubscriptions>**  **<!-- lock down the endpoint when deployed to content delivery -->**  **<graphiql role:require="ContentDelivery">false</graphiql>**  **<enableSchemaExport role:require="ContentDelivery">false</enableSchemaExport>**  **<enableStats role:require="ContentDelivery">false</enableStats>**  **<enableCacheStats role:require="ContentDelivery">false</enableCacheStats>**  **<disableIntrospection role:require="ContentDelivery">true</disableIntrospection>**  **<schema hint="list:AddSchemaProvider">**  **<content type="Sitecore.Services.GraphQL.Content.ContentSchemaProvider, Sitecore.Services.GraphQL.Content">**  **<!-- scope typed template generation to just this app's templates -->**  **<templates type="Sitecore.Services.GraphQL.Content.TemplateGeneration.Filters.StandardTemplatePredicate, Sitecore.Services.GraphQL.Content">**  **<database>context</database>**  **<paths hint="list:AddIncludedPath">**  **<templates>/sitecore/templates/Project/jss-demo-app</templates>**  **</paths>**  **<fieldFilter type="Sitecore.Services.GraphQL.Content.TemplateGeneration.Filters.StandardFieldFilter, Sitecore.Services.GraphQL.Content">**  **<exclusions hint="raw:AddFilter">**  **<!--**  **Remove system fields from the API (e.g. \_\_Layout) to keep the schema lean**  **-->**  **<exclude name="\_\_\*" />**  **</exclusions>**  **</fieldFilter>**  **</templates>**  **<queries hint="raw:AddQuery">**  **<!-- enable querying on items via this API -->**  **<query name="item" type="Sitecore.Services.GraphQL.Content.Queries.ItemQuery, Sitecore.Services.GraphQL.Content" />**  **</queries>**  **<fieldTypeMapping ref="/sitecore/api/GraphQL/defaults/content/fieldTypeMappings/standardTypeMapping" />**  **</content>**  **</schema>**  **<!-- Enables the 'jss' graph nodes that are preformatted to use with JSS rendering components, and the datasource resolving queries for JSS -->**  **<extenders hint="list:AddExtender">**  **<layoutExtender type="Sitecore.JavaScriptServices.GraphQL.JssExtender, Sitecore.JavaScriptServices.GraphQL" resolve="true" />**  **</extenders>**  **<!-- Determines the security of the service. 'publicService' is open to anonymous access, but requires an SSC API key. -->**  **<security ref="/sitecore/api/GraphQL/defaults/security/publicService" />**  **<!-- Determines how performance is logged for the service. Defaults are defined in Sitecore.Services.GraphQL.config -->**  **<performance ref="/sitecore/api/GraphQL/defaults/performance/standard" />**  **</jss-demo-appGraphQLEndpoint>**  **</endpoints>**  **</GraphQL>**  **</api>**  **</sitecore>**  **</configuration>** |

Note: Above config is just a sample and not for copying for your app. Generate the config for your app and compare with this example. It is just for reference.

* Review the application config patch file(s) in ***"/sitecore/config"*** to ensure that it is configured appropriately for your Sitecore installation.

**Step 5: Deploy following configs to Sitecore server.**

1. jss-demo-app.config
2. jss-demo-app.deploysecret.config

**Further changes to these configs would be done via devops pipelines to which tenants would be given access and instructions**

**Step 6: Deploy your app to sitecore using** [**jss cli**](https://jss.sitecore.com/docs/fundamentals/cli)**(as a one time step, later you would be deploying only components/templates etc.)**

Use following command to deploy the app to Sitecore:

|  |
| --- |
| **$ jss deploy items --includeContent --includeDictionary --appName <appName>** |

**Here:**

* *<appName>* refers to the app name you are deploying. e.g. jss-demo-app.
* *--includeContent* is an instruction to the 'jss deploy app' command to include the content data locally available.
* *--includeDictionary* is an instruction to the 'jss deploy app' command to include the available dictionary items.

**Example:**

|  |
| --- |
| ***$ jss deploy items --includeContent --includeDictionary --appName jss-demo-app*** |

***"jss deploy app"*** is shorthand for ***"jss deploy items && jss deploy files"***. Used to deploy both the app's manifest items (routes, templates, etc) and a production build of the app's code. ***jss deploy app*** accepts all options supported by ***jss deploy items*** or ***jss deploy files***.

**Step 6: Start running connected mode via**[**jss cli**](https://jss.sitecore.com/docs/fundamentals/cli)

Use the following command to start the connected mode:

|  |
| --- |
| **$ jss start:connected**  **OR**  **$ npm rum start:connected** |

**Learn more on JSS CLI commands here:**[**https://jss.sitecore.com/docs/fundamentals/cli**](https://jss.sitecore.com/docs/fundamentals/cli)

**Creating a component and deploying to Sitecore, An example:**

To create a component, you can use [**JSS CLI scaffold**](https://jss.sitecore.com/docs/client-frameworks/getting-started/first-component) which will create component sample with the name you provide.

Use the below command to create a component named "***Header***" (for example):

|  |
| --- |
| **$ jss scaffold <componentName>** |

**Example**:

|  |
| --- |
| **$ jss scaffold Header** |

|  |
| --- |
| **> npm run scaffold -- Header**  **> jss-demo-app@18.0.0 scaffold C:\sitecoreFrontend\jss-demo-app > node scripts/scaffold-component.js "Header"**  **Component Header has been scaffolded. Next steps: \* Define the component's data in sitecore\definitions\components\Header.sitecore.js \* Implement the React component in src\components\Header\index.js \* Add the component to a route layout (/data/routes) and test it with jss start** |

Use the following command to deploy the component to Sitecore:

|  |
| --- |
| **$ jss deploy items** |

***jss deploy items*** has many options that enable modifying how the package deploys, such as excluding content items with **--noContent**. Run ***jss deploy items --help*** to see all the options with descriptions.

When run with default options, ***jss deploy items*** also runs ***jss manifest***, and ***jss package***. Options for all of these commands may also be passed to deploy items.

Content, media, or dictionary items are excluded by default for safety, in case a content editor has changed the data. To deploy these items, run ***jss deploy items --includeContent --includeDictionary*** to deploy everything.

**Learn more on creating components** [**here**](https://jss.sitecore.com/docs/client-frameworks/getting-started/first-component).