

Engineering  
Awesome  
Conference.

# Mashreq Bank & Sitecore JSS

An abstract 3D geometric composition on a dark background. It features a central teal sphere with a dark red, multi-layered, flower-like structure on its surface. Two bright cyan lines intersect at the center, forming an 'X' shape. Scattered around are several small, glowing red spheres and red polyhedrons, some with small orange lights at their vertices.

# Agenda

## Intro

Nodes and Mashreq Bank

## Sitecore JSS

Intro to Sitecore JSS

## Our Approach

How we implemented the solution

## Our Learnings

Can we do it better next time?

## Maturity of JSS

Can Sitecore do it better with the next JSS release?

Nodes

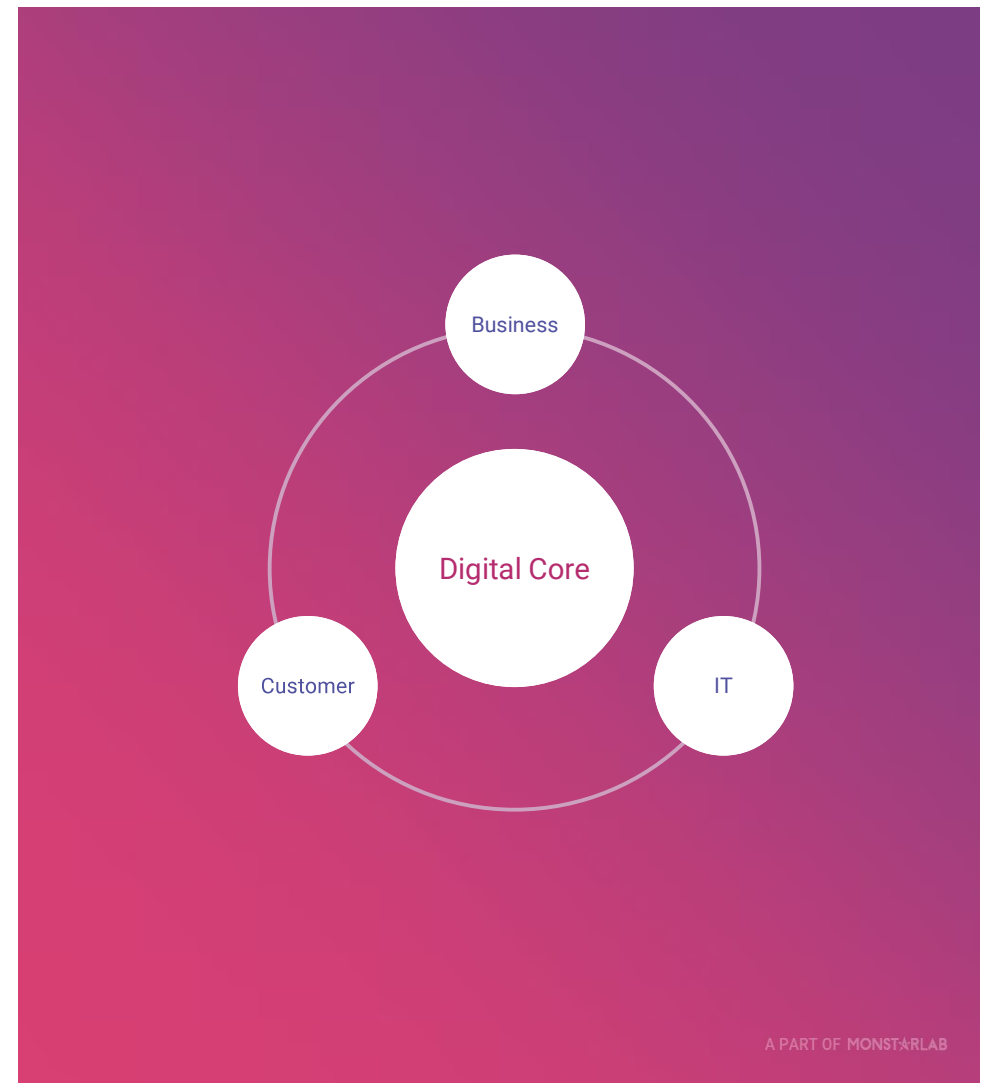
# Nodes and Sitecore

**Sitecore Platinum Partner**  
2016

**Previously worked at Sitecore**

**Experience Awards**  
2016, 2 x 2018, 2019

**Sitecore MVP**





Mashreq Bank

# Mashreq Bank

**Founded in 1967 in Dubai**  
EU, Asia, Africa and US

## **Core values**

Socially Responsible, Passionated about clients,  
Innovative, Respect for colleagues and integrity  
transparent

## **Euromoney award**

“Most innovative bank in Middle East”

A PART OF MONSTARLAB

# Setting the scene

New website

Wanted to use content across dif. channels

Full control of the development

First workshop i Dubai start Marts

# Sitecore JSS

Go headless with full support  
of Sitecore Experience  
Platform functionality

# Why Headless

Deliver content across digital channels



# Why Headless

Deliver content across digital channels

Ressources from Server to Client





# Why Headless

**Deliver content across digital channels**

**Ressources from Server to Client**

**Better performance and better UX flow**

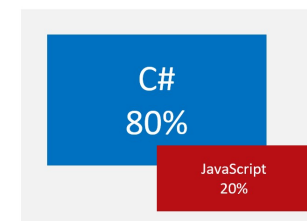
# Why Headless

**Deliver content across digital channels**

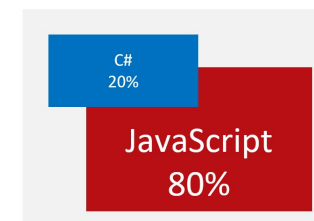
**Ressources from Server to Client**

**Better performance and better UX flow**

**Competences from backend and platform to frontend**



Sitecore MVC



Sitecore JSS

# Why Sitecore JSS for Mashreq Bank

## **Tracking, personalization and optimization and A/B testing**

Sitecore Experience Platform offerings

## **Require cutting edge technology and React**

Support Internal developer department.

## **No vendor specific framework**

Wanted standard framework

# Sitecore and headless

## 2001, Sitecore released

Sitecore started in 2001 with separation of content and presentation

## 2012, Sitecore Item Web API

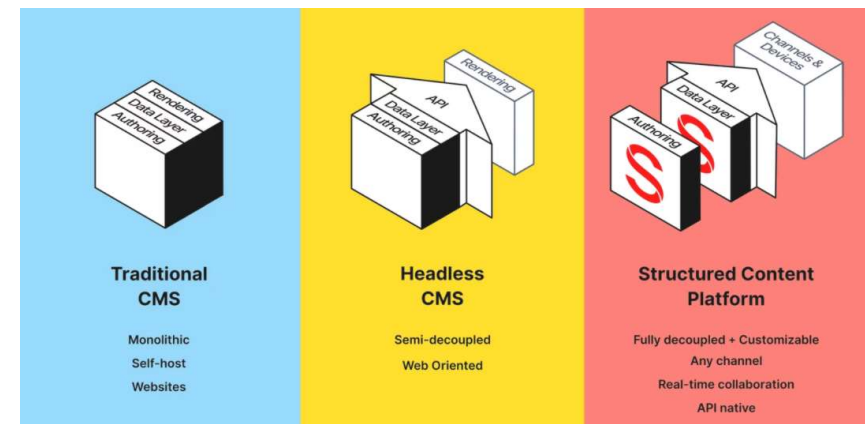
Sitecore introduced their first take on exposing content in JSON in Sitecore version 6.5 from 2012.

## 2015, Sitecore Service Client API

In Sitecore version 7.5 from 2015 they introduced Sitecore.Service.Client API for ItemService and EntityService

## 2017, Sitecore JSS

In Sitecore version 9.0 from 2017 Sitecore introduced their first take on Sitecore Headless with Sitecore JSS



# Introducing Sitecore JSS



## JSS Library

NPM Packages that offer Sitecore Layout capabilities in JavaScript



## Sitecore Layout Service

Physically decouples rendering from Content Delivery



## Server-side JavaScript rendering

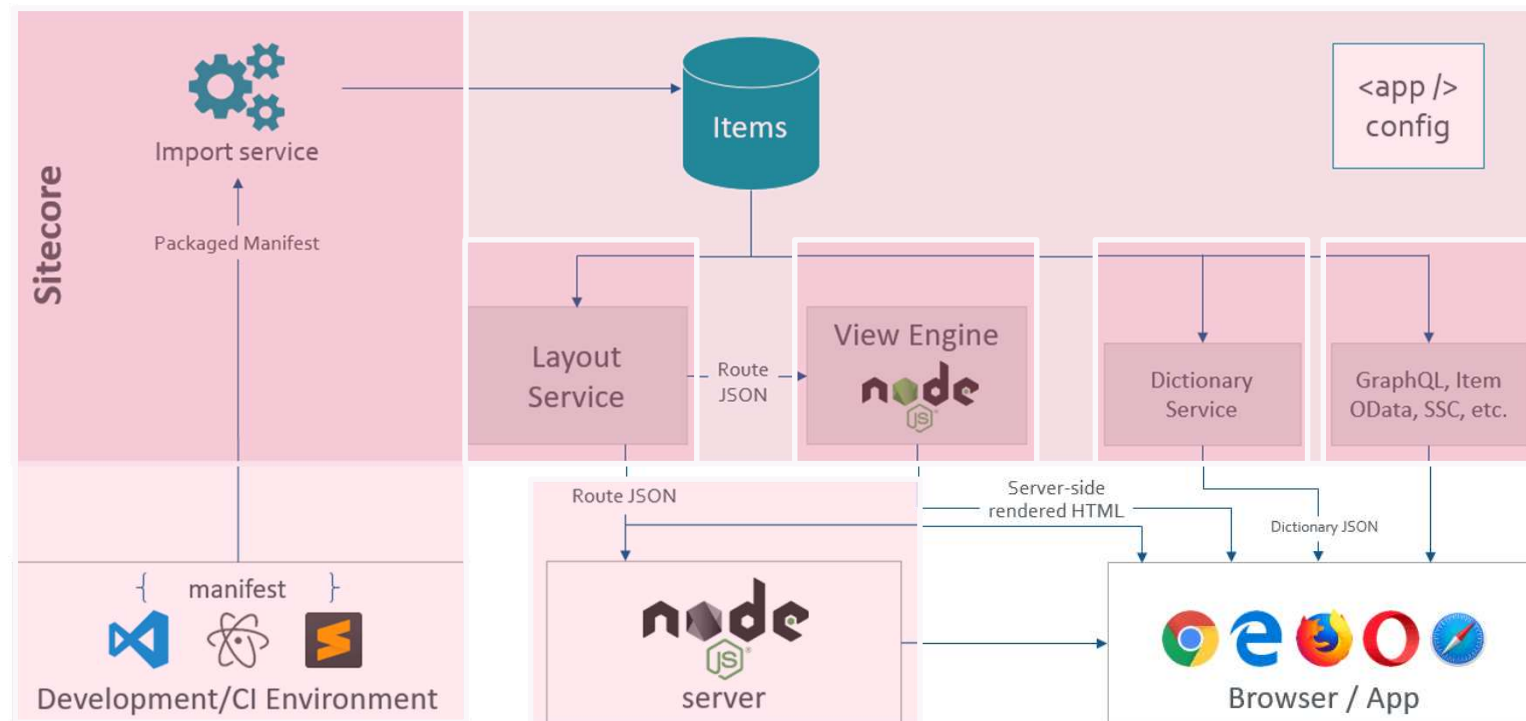
Using node.js integrated with Sitecore rendering



## Application Import

Generate Sitecore artifacts from JavaScript source

# Introducing Sitecore JSS

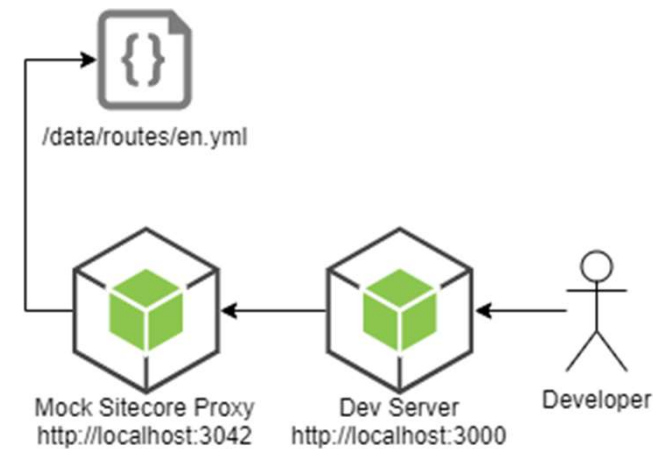


# JSS Application Mode

## Disconnected Developer Mode

Rendering performed by the browser

Data comes from local json file



# JSS Application Mode

## Disconnected Developer Mode

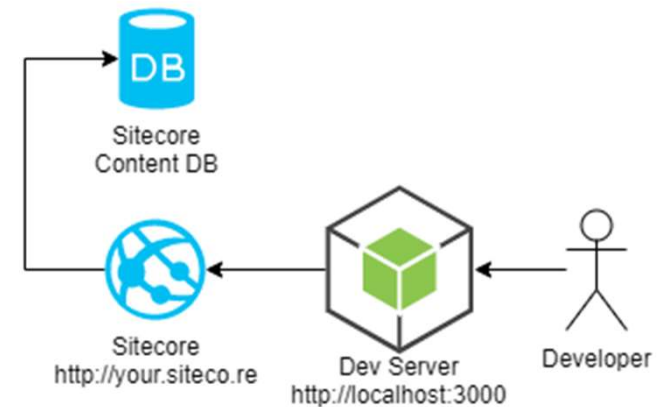
Rendering performed by the browser

Data comes from local json file

## Connected Developer Mode

Rendering performed by the browser

Data comes from Sitecore via LayoutService/DictionaryService/GraphQL via HTTP





# JSS Application Mode

## Disconnected Developer Mode

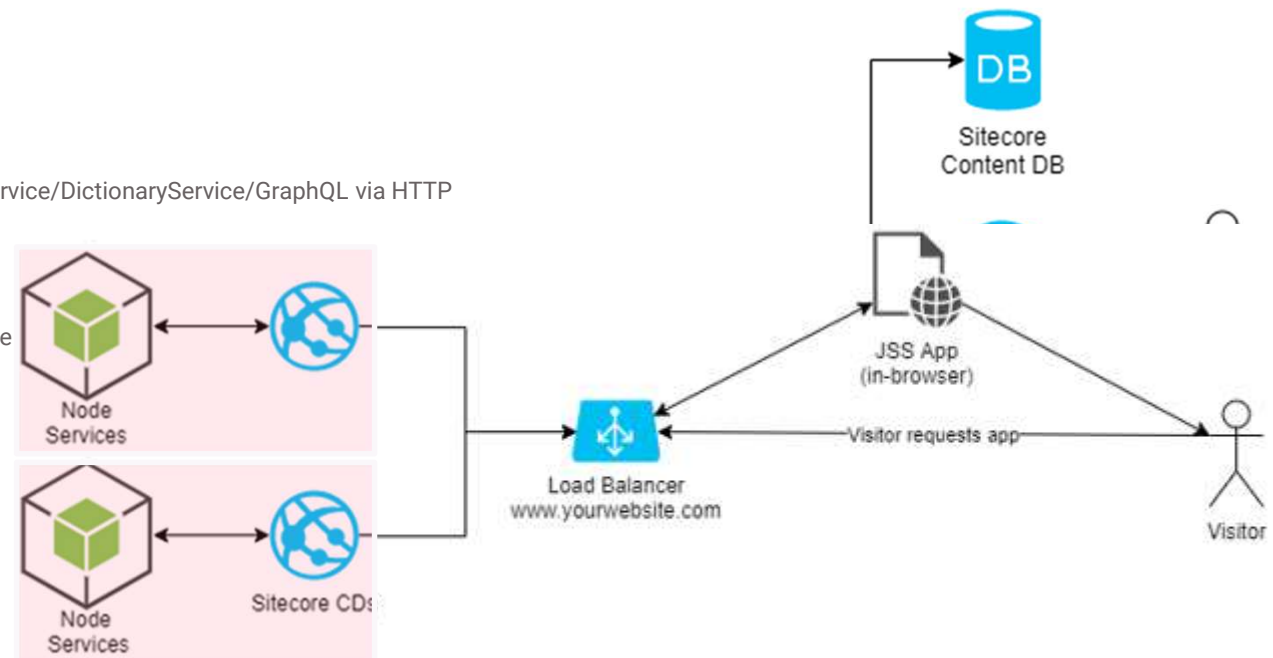
Rendering performed by the browser  
Data comes from local json file

## Connected Developer Mode

Rendering performed by the browser  
Data comes from Sitecore via LayoutService/DictionaryService/GraphQL via HTTP

## Integrated Mode

Rendering performed by server  
Data comes from Sitecore Layoutservice  
Requires configuration of the app



# JSS Application Mode

## Disconnected Developer Mode

Rendering performed by the browser  
Data comes from local json file

## Connected Developer Mode

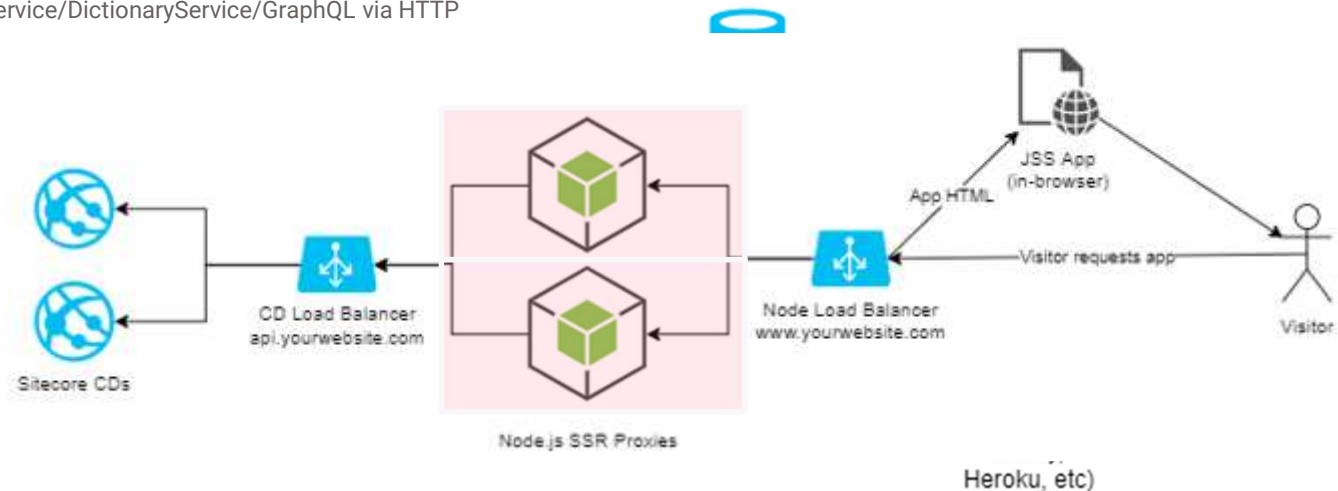
Rendering performed by the browser  
Data comes from Sitecore via LayoutService/DictionaryService/GraphQL via HTTP

## Integrated Mode

Rendering performed by server  
Data comes from Sitecore Layoutser  
Requires configuration of the app

## Headless Server-Side Rendering Mode

Rendering performed by server  
Data comes from Sitecore CD



# JSS Application Mode

## Disconnected Developer Mode

Rendering performed by the browser  
Data comes from local json file

## Connected Developer Mode

Rendering performed by the browser  
Data comes from Sitecore via LayoutService/DictionaryService/GraphQL via HTTP

## Integrated Mode

Rendering performed by server  
Data comes from Sitecore Layoutservice  
Requires configuration of the app

## Headless Server-Side

### Rendering Mode

Rendering performed by server  
Data comes from Sitecore CD

## API-Only Mode (Sitecore Service Client)

Consume Sitecore APIs returning JSON

# Our Approach

Efficiency.  
Without compromise Sitecore  
OOTB functionality.

# Our Objectives

## **Intencify backend and frontend development**

Better and more smooth collaboration

## **Avoid dependencies**

Between Backend and Frontend developers

## **Efficient implementation process**

Tight deadline

## **Don't compromise Sitecore OOTB functionality**

We are highly focused on following best practices and guidelines. The customer want to use DXP

# The developing process

## Frontenders not used to work with Sitecore

... And don't want to learn Sitecore

## No need to learn how the platform works

Between Backend and Frontend developers

## Heavy load time for Sitecore

Feeling of wasting time



Loading...



Loading...



Loading...

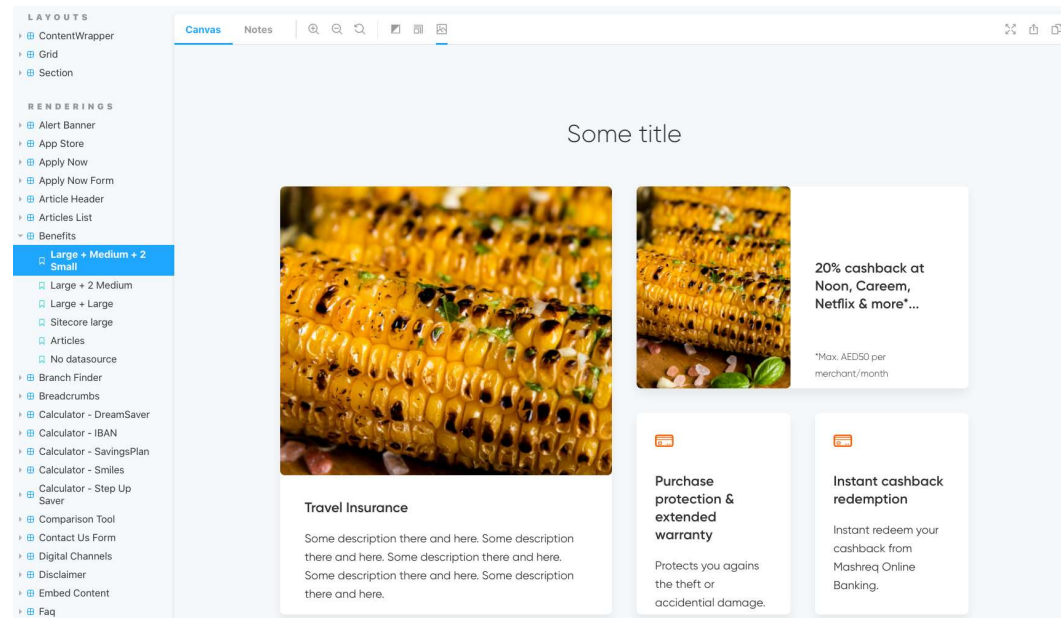


Loading...

# Our Approach

## Frontend developers used to work with Storybook

Component development without the entire webpage



# Our Approach

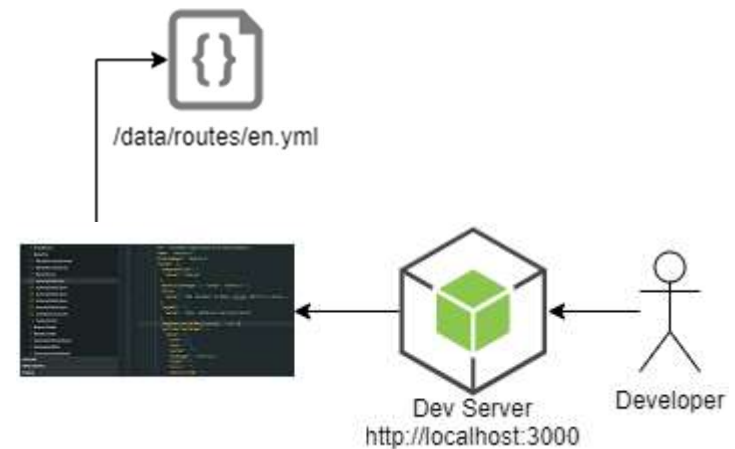
## Disconnected mode, but not

Component development without the entire webpage

## First come has the responsibility

Contract for Content

Defines dummy data





# Our Approach

## Field Definition

Component development without the entire webpage

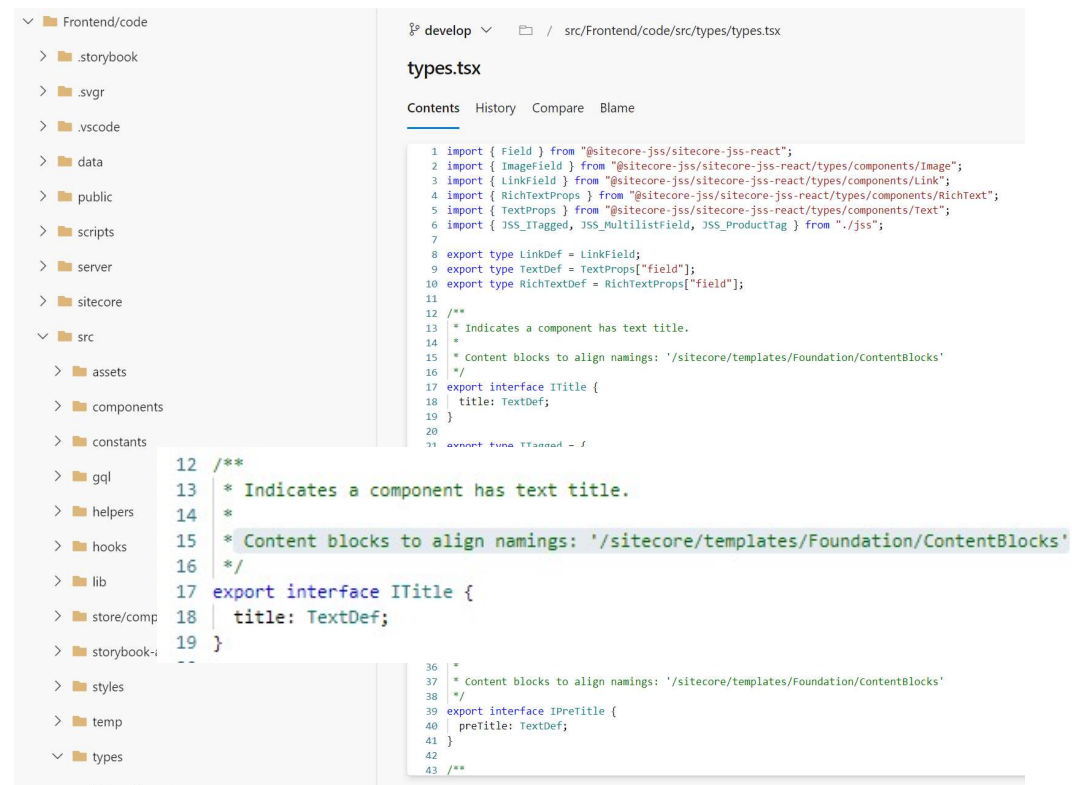
## Respect to Sitecore Helix

Following Sitecore Helix conventions

## Speed up development due to agree on this process

Contract for Content

Defines dummy data



```
1 import { Field } from "@sitecore-jss/sitecore-jss-react";
2 import { ImageField } from "@sitecore-jss/sitecore-jss-react/types/components/Image";
3 import { LinkField } from "@sitecore-jss/sitecore-jss-react/types/components/Link";
4 import { RichTextProps } from "@sitecore-jss/sitecore-jss-react/types/components/RichText";
5 import { TextProps } from "@sitecore-jss/sitecore-jss-react/types/components/Text";
6 import { JSS_ITagged, JSS_MultilistField, JSS_ProductTag } from "../jss";
7
8 export type LinkDef = LinkField;
9 export type TextDef = TextProps["field"];
10 export type RichTextDef = RichTextProps["field"];
11
12 /**
13  * Indicates a component has text title.
14  *
15  * Content blocks to align namings: '/sitecore/templates/Foundation/ContentBlocks'
16  */
17 export interface ITitle {
18   title: TextDef;
19 }
20
21 export type TTagged = JSS_ITagged & ITitle;
```

# Our Approach

## Field Definition

Agreement how the data look like

Typescript to able to define the most common field.

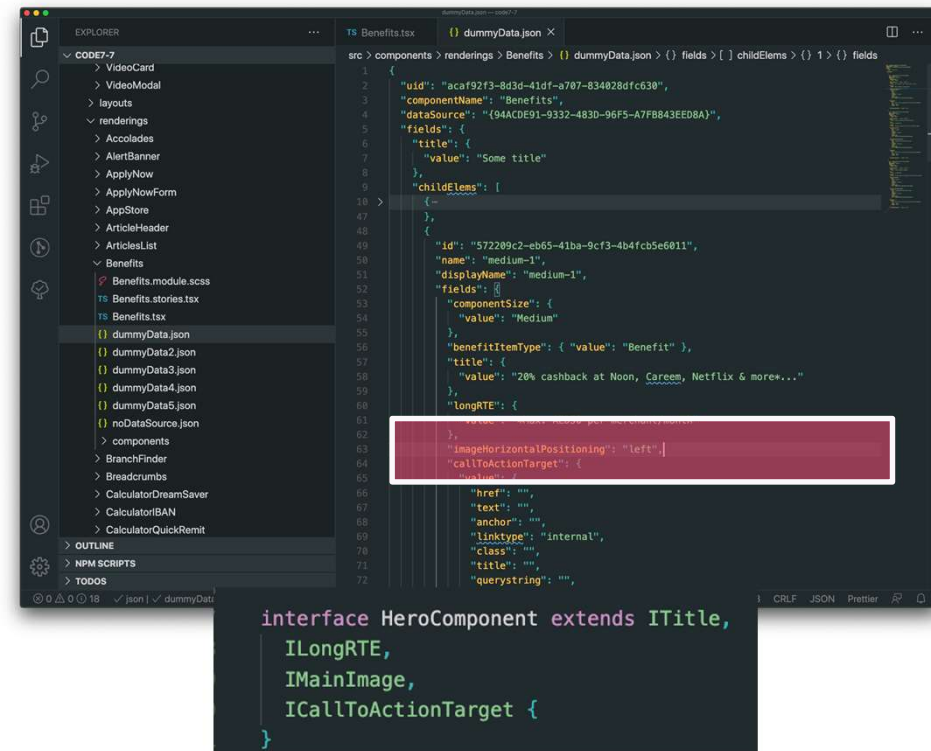
FE compose dummy data by field definition

## Dummy Data

The one first come would create the dummy data and the contract for the content contained in the component

## Test environment for frontend testing

For QA frontend implementation before deploy



```
1 {
2   "uid": "acaf92f3-8d3d-41df-a707-834028dfc630",
3   "componentName": "Benefits",
4   "dataSource": "{94ACDE91-9332-483D-96F5-A7FB843EED8A}",
5   "fields": {
6     "title": {
7       "value": "Some title"
8     },
9     "childElems": [
10    {
11      "id": "572209c2-eb65-41ba-9cf3-4b4fcb5e6011",
12      "name": "medium-1",
13      "displayName": "medium-1",
14      "fields": {
15        "componentSize": {
16          "value": "Medium"
17        },
18        "benefitItemType": { "value": "Benefit" },
19        "title": {
20          "value": "20% cashback at Noon, Careem, Netflix & morex..."
21        },
22        "longRTE": {
23          "value": "20% cashback at Noon, Careem, Netflix & morex..."
24        },
25        "imageHorizontalPositioning": "left",
26        "callToActionTarget": {
27          "value": "20% cashback at Noon, Careem, Netflix & morex..."
28        },
29        "href": "",
30        "text": "",
31        "anchor": "",
32        "linkType": "internal",
33        "class": "",
34        "title": "",
35        "queryString": ""
36      }
37    }
38  ]
39 }

interface HeroComponent extends ITitle,
  ILongRTE,
  IMainImage,
  ICallToActionTarget {
}
```

# Our Approach

## **Basic implementation of Rendering Engine Javascript**

Instead of MVC rendering engine

## **GraphQL to consume data from Sitecore**

Advantages of OOTB Sitecore

## **No Dictionary Service in use**

implemented at item level instead

## **Server Side Renderings**

Limitations for frontenders, they don't have window object

## **Integrated Mode CM and CD**

CM for supporting Experience Editor. CD based on Sitecore recommendation

## **If performance issues, we could move NodeJS**

If we got performance issues, we could move NodeJS away from Sitecore CD

# Our Learnings

Experience. Reflect. Conclude.  
Apply.

# Our Learnings

## **Performance, performance, performance**

Search rankings

Impact your bounce rate

Ultimately impact your bottom line

# Performance

## CPU

Number of users: 1,100 concurrent users

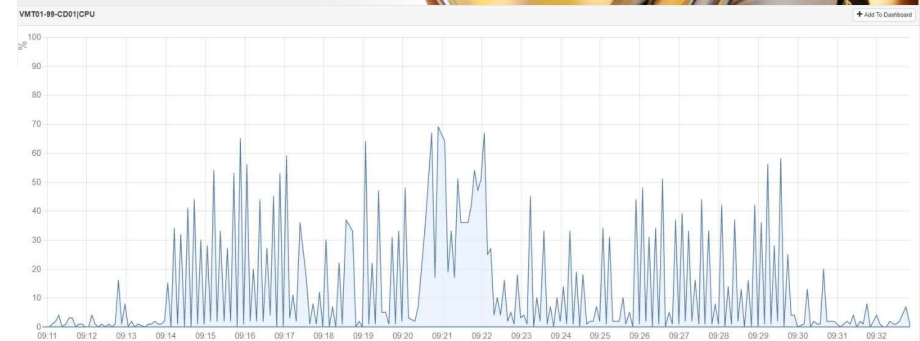
Thread delay: 10s

Duration: 15 min

## Sitecore JSS



## Sitecore MVC



# Performance

## AppPool Memory

Number of users: 1,200 concurrent users

Thread delay: 10s

Duration: 15 min

## Default Sitecore JSS configuration

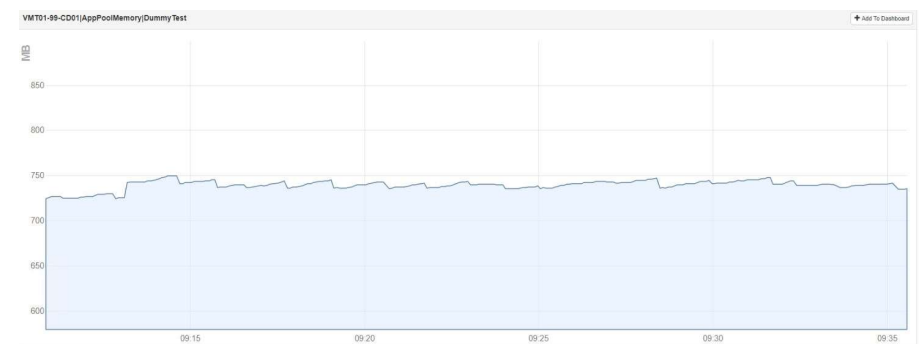
"/App\_Config/Sitecore/JavaScriptServices/Sitecore.JavaScriptServices.Apps.Config"

ServerSideRenderingWorkerProcesses="2"

## Sitecore JSS



## Sitecore MVC



# Our Learnings

## Caching

Sitecore Cache Instance  
Automatic Persisted Queries

## Security

Query Complexity  
Authoritative Cahce

```
<cache type="Sitecore.Services.GraphQL.Hosting.QueryTransformation.Caching.WhitelistingGraphQLQueryCache, Sitecore.Services.GraphQL">  
  <!-- path can be virtual (/... or ~/...) or physical (c:\...) -->  
  <param desc="path">~/App_Data/GraphQL/myendpoint-whitelist</param>  
  <!-- learning allows the whitelist to add incoming queries; then disable this in production -->  
  <learningEnabled>true</learningEnabled>  
</cache>
```



# Maturity of JSS

Expectations meeting reality?

# Maturity of JSS

## **By Default Sitecore JSS comes with some limitations**

Multisite limitations

QS limitations

## **Simple request for filtering items don't need coding**

Get content based on GraphQL query

Easily extending JSON from Sitecore

## **Specific item resolver per component**

By default you need to implement a specific item resolver per component.

Expose a model to re-use most used fields 80% of the time

Enhanced templating at component level

# Thank you!

Engineering  
Awesome  
Conference.

# Any questions?

