Refactoring your Helix solution to JSS

Vitalii Tylyk

About me

- From Ukraine
- ex-Sitecorian, worked on several positions/projects:
  - As developer: *Social Connected*, *Komfo Connector*, *CMS Core*
  - As technical support: *CMS Core, Search&Indexing, Azure*
- Overall 7 years of Sitecore experience

@vitalii_tylyk
https://blog.vitaliitylyk.com

History | Motivation

- Technical innovation:
  - Enable modern FE development frameworks in the Sitecore development stack
  - Monolithic architecture => microservices
- Development process improvement: parallel independent development of BE & FE

History | The challenge

- A chance to innovate existing project running on Sitecore MVC.
  - It is a bank
  - No big infrastructure changes
  - Maximize content reuse, **transparent to content editors**
  - Pressure from skeptics. As usual.
  - JSS was still in Beta phase ☺
- Time for POC: 2 weeks

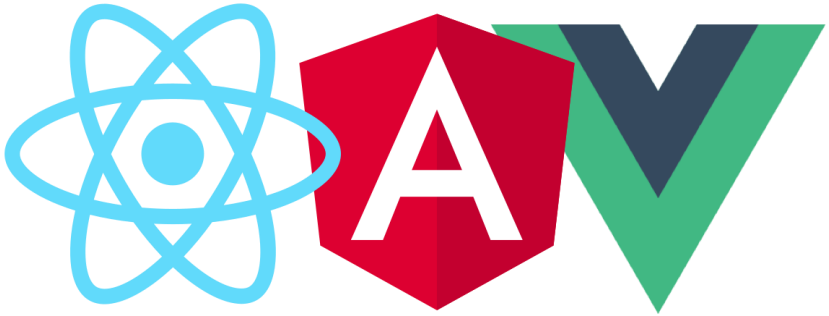
History | Technology stack

- Existing solution
  - AngularJS 1.7 traditional Sitecore MVC app
  - Glass Mapper
  - Helix
  - A monolith
- Target solution
  - SPA on top of Angular 7
  - JSS
  - Helix
  - Separated API microservices

Solution | Overview

1. Choose FE framework
2. Choose development workflow
3. Define code repository structure
4. Refactor BE code
5. Implement frontend application (*can be done in parallel*)
6. Choose deployment topology
7. Update build and deployment process

FE framework choice



- Sitecore is not giving a preference to any of these 3 and will equally support all of them.
- Sitecore builds its own UIs in Angular (SPEAK 3, new Horizon)
  - Kill 2 birds with one stone

Development flow

- FE First & Sitecore First
- Migration => Sitecore first:
  - **Templates reuse** => faster for refactoring scenario
    - *Ready-made contract*
  - Component separation reuse
  - Easier transition, **this is how most companies are used to work**
- **Contract** first**!**
  - Make sure to align with FE on data structure before doing implementation

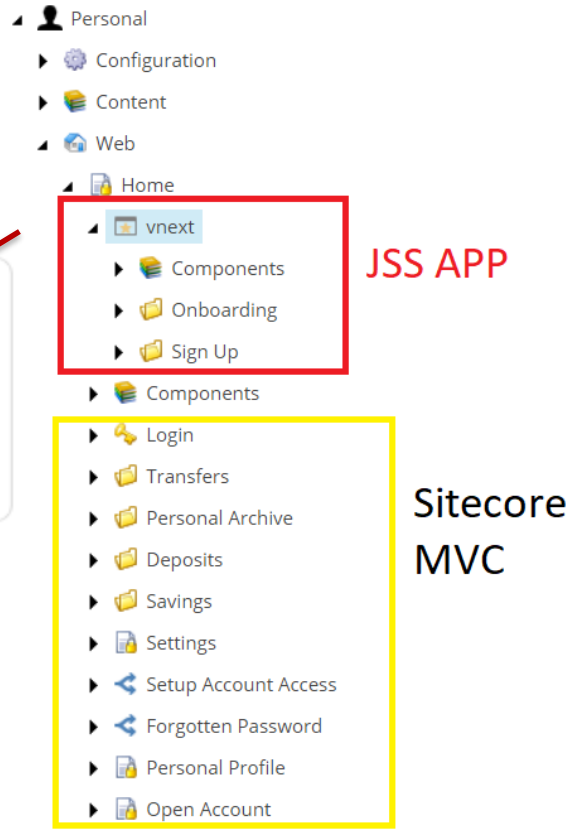
Code repository structure

- Sitecore MVC:
  - FE and BE code normally reside in same repo and are tightly coupled
- Sitecore JSS: 2 options
  - Separate repositories for FE and BE code
  - **Single repository (recommended by Sitecore)**
    - *Easier to version-control/synchronize/revert changes*
    - *Simpler to manage*
    - ***Do** keep FE code in a separate folder.*

Backend | JSS APP setup

- No need to rewrite it all at once

```
<apps>
  <app name="MyJSSapp"
     sitecorePath="/sitecore/content/MyClient/home/vnext"
     inherits="defaults"
  />
</apps>
```

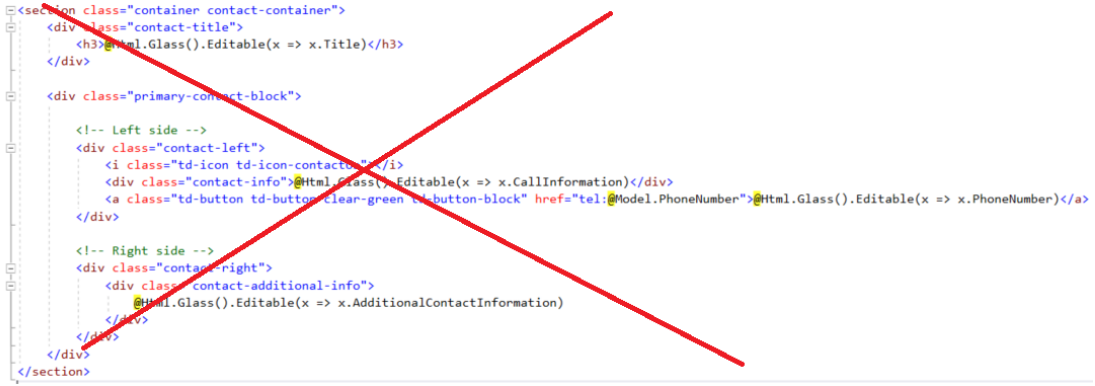


Backend | Sitecore templates

- Inherit your **Page Templates** from **/sitecore/templates/Foundation/JavaScript Services/Route**
- Most of the **Datasource Templates** templates can be reused
  - You don’t have to rebuild them ☺
  - But you might consider reviewing them / removing unused fields

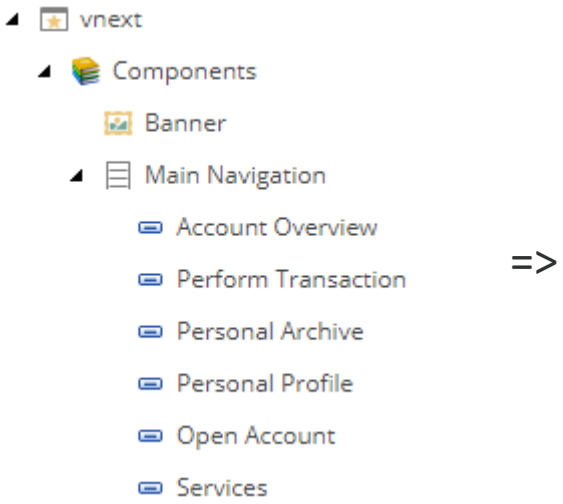
Backend | Refactoring existing Sitecore MVC code

- MVC Controllers, Razor views, TDS Generated code / Glass Mapper / etc



Backend | Refactoring existing Sitecore MVC code

- ~~MVC Controllers, Razor views, TDS Generated code / Glass Mapper / etc~~
- JSS Layout service serializes Sitecore items to JSON directly



=>

Backend | Creating JSS renderings

GraphQL

Component GraphQL Query - shapes the JSON data passed to this component. Variables $datasource and $contextItem are always available. If blank, normal JSS shaping is used:

Open xGraph Browser

```
query IntegratedPageQuery($datasource: String!, $contextItem: String!) {
  # Datasource query
  # $datasource will always be set to the ID of the rendering's datasource item.
  datasource(value: $datasource) {
    ... on IntegratedPage {
      title {
        jss
      }
      text {
        jss
      }
      logoImage {
        jss
      }
    }
  }

  # Context item query
  # $contextItem will always be set to the ID of the current context item (the route item)
  contextItem: item(path: $contextItem) {
    id
    children(requirePresentation: true) {
      displayName
      url(options: { disableLanguageEmbedding: true })
    }
  }
}
```

Backend | Extra data for renderings

- Extend LayoutService context data

  https://jss.sitecore.com/docs/techniques/extending-lay

  service/layoutservice-extending-context

- Integrated GraphQL => **no code required**
- Custom resolvers for complex cases


```
{
  "context": {
    "securityInfo": {
      "isAnonymous": true
    },
    "pageEditing": false,
    "site": {
      "name": "JssReactWeb"
    },
    "navigation": [
      {
        "name": "Home",
        "path": "/",
        "children": [
          {
            "name": "About",
            "path": "/about"
          },
          {...},
          {...}
        ]
      }
    ]
  }
}
```

Backend | Analytics

- Use JSS tracking service
  - Goals
  - Events
  - Outcomes
  - Campaigns
  - Views

```
trackingApi
  // note the events are an array - batching is supported
  .trackEvent([{ eventId: 'Download' }], trackingApiOptions)
  .then(() => console.log('Page event pushed'))
  .catch((error) => console.error(error));
```

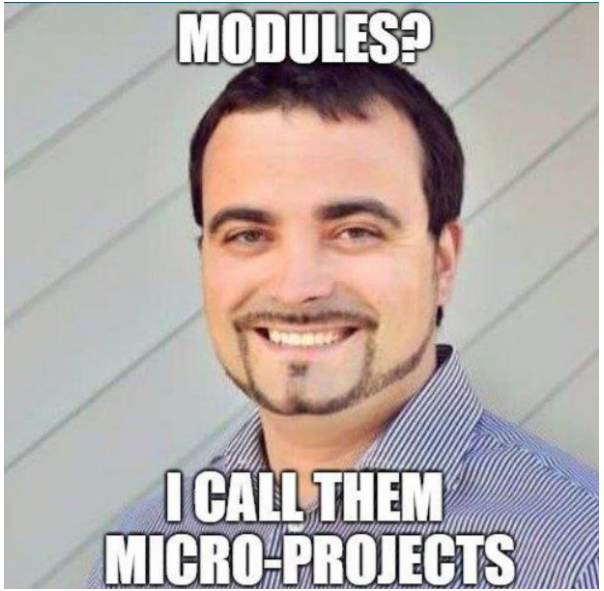
- Custom API for contact identification, etc

Backend | Helix & JSS

- Classic Sitecore MVC Helix Feature module:
  - MVC Controllers (controller renderings)
  - API Controllers
  - Models
  - Views
  - Serialized items
  - Services, Repositories, Infrastructure Code
- Sitecore JSS Helix module:
  - Serialized items
  - *Custom rendering contents resolvers*
  - *API Controllers => preferably separate microservice*
  - *Services, Repositories, Infrastructure Code => preferably separate microservice*

Backend | Helix & JSS

- How useful are separate VS projects per Helix module?
  - In Helix architecture, it is **the logical boundary** and the proper **dependency direction** which matters, **not the physical boundary**
- Consider using single VS project for your Feature layer
- Coming soon: **Helix.Examples** => Helix mutations, including Helix Guidance for JSS



Frontend | Helix

2 ways:

- Give FE developers freedom to use their own best practices, tooling and structure
  - https://blog.vitaliitylyk.com/sitecore-jss-meets-helix-introduction

- Enforce Helix to FE world
  - https://www.jflh.ca/2018-10-13-helix-and-sitecore-javascript-services

Helix + JSS | Summary

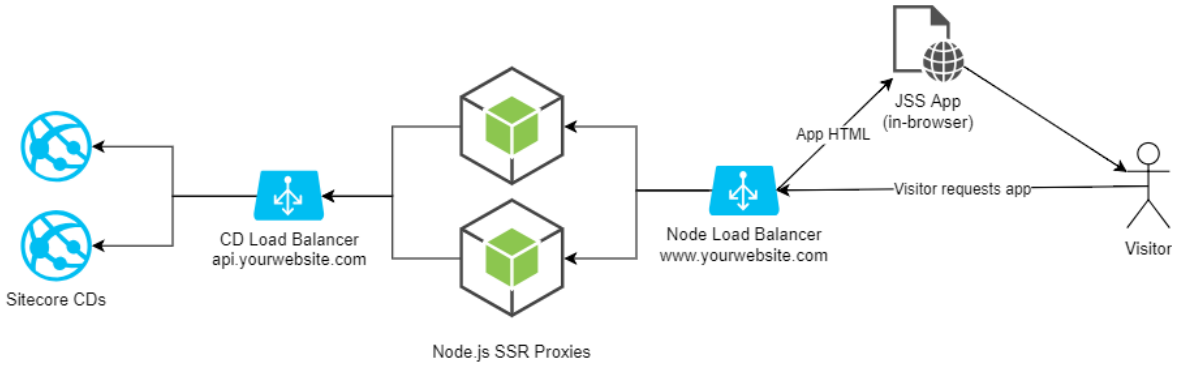
- Organize your Sitecore items in Helix way
- Consider alternative VS solution structures
  - Project-per-layer
  - Project-per-module Foundation
- Follow best practices of chosen FE framework
  - Keep FE code separate
  - Split code into modules
  - Align naming with BE

Frontend | Implementation tips

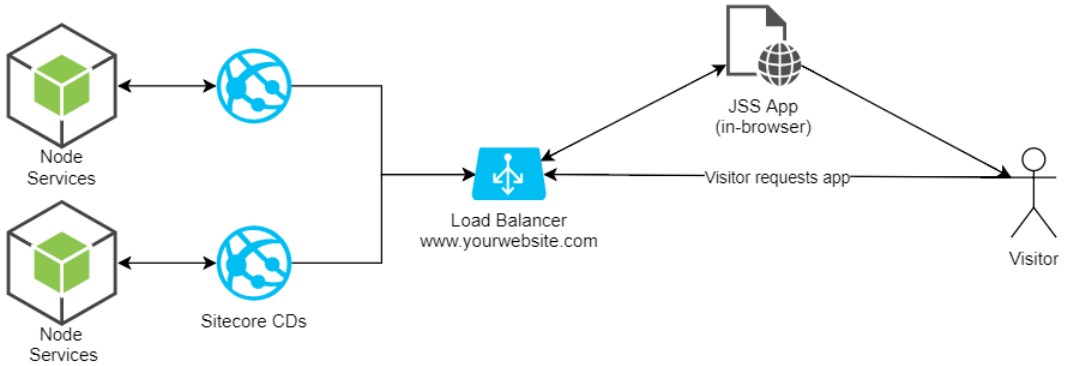
- Consider UX changes
- Use helpers for EE support
  - Test EE support from the early stage
- Check the **Client Frameworks** section in docs
- Convert links to router links
  - https://kamsar.net/index.php/2018/09/Routing-Sitecore-links-with-JSS/
- JSS development team gladly accepts pull requests ;)

Deployment topology

- Headless topology



- Integrated topology
  - **Good starting point**



Build process changes

- Configuration:
  - Layout Service host URL
  - API key
- Build process:
  - ```
    jss setup --nonInteractive --apiKey="#{ApiKey}" --layoutServiceHost="#{LayoutServiceHost}"
    ```
  - ```
    jss deploy files --destination="path\to\your\website\build\output"
    ```

Deployment process changes

- Prerequisites:
  - Install Node.JS on CM/CD servers
  - Install JSS server package on your environments
- Copy JSS app build artifacts to CM/CD
  - Deploy Node.js instance(s) in case of Headless topology
- *[Optional]* Replace #{LayoutServiceHost} and #{ApiKey} tokens in *.js files
- **No changes** in item serialization/deployment: Unicorn/TDS

Results

- Modern FE development stack
- Enabled parallel BE/FE development (contract first)
- FE development speed +50%
  - Modern tooling & framework
  - No BE dependency (using mocks, disconnected mode)
- BE development speed +30%
  - No need to build Views, Models, Controllers

Thanks!

- More details in https://blog.vitaliitylyk.com/guide-on-refactoring-your-sitecore-solution-to-sitecore-jss/
- Questions?

@vitalii_tylyk
https://blog.vitaliitylyk.com