



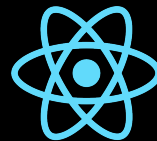
EUROPE'S LEADING AEM DEVELOPER CONFERENCE
28th – 30th SEPTEMBER 2020

Components as a Service

Tony Schumacher, Teclead

Agenda

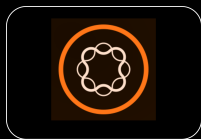
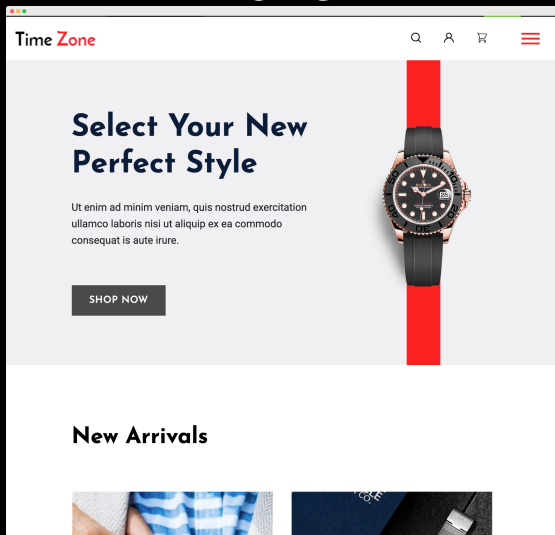
- Project Situation
- Solution Concept
- Frontend Architecture
- Authoring
- Demo / Live coding
- FAQ



Project Situation

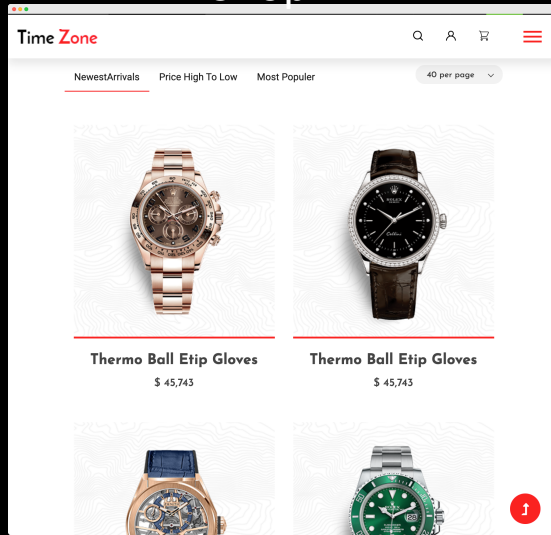
Project Situation

Home



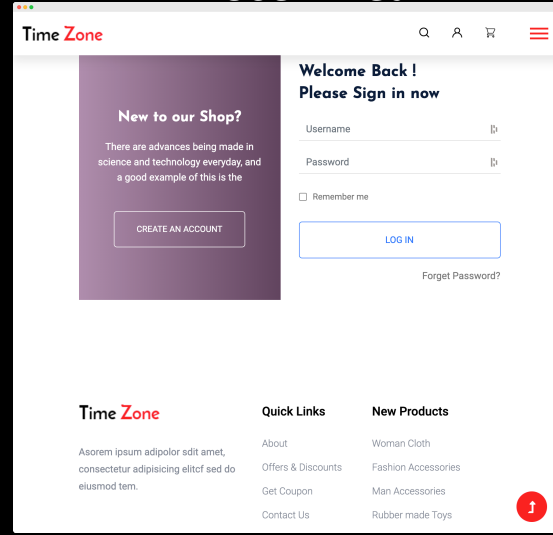
brand.com

Shop



brand.com/shop

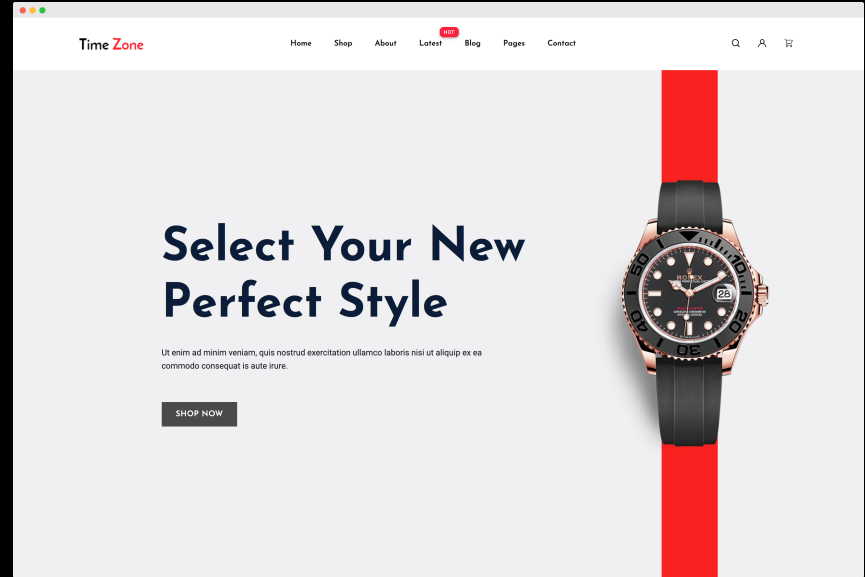
User Area



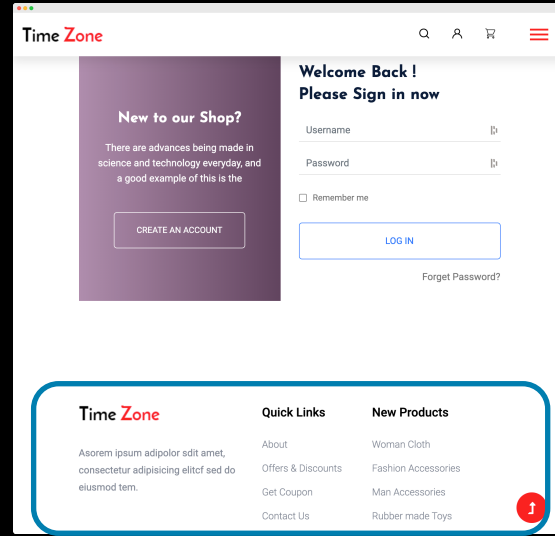
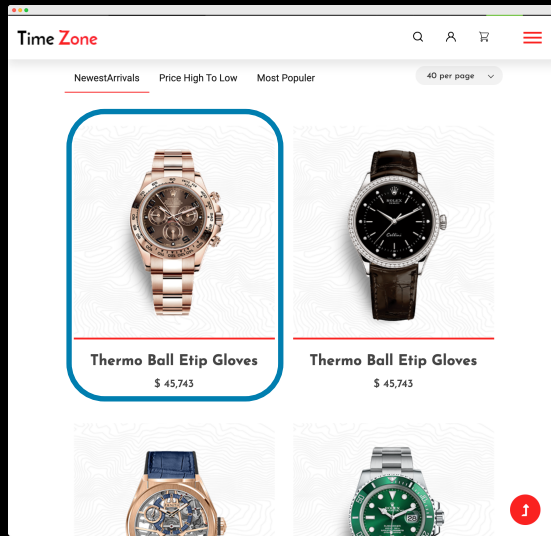
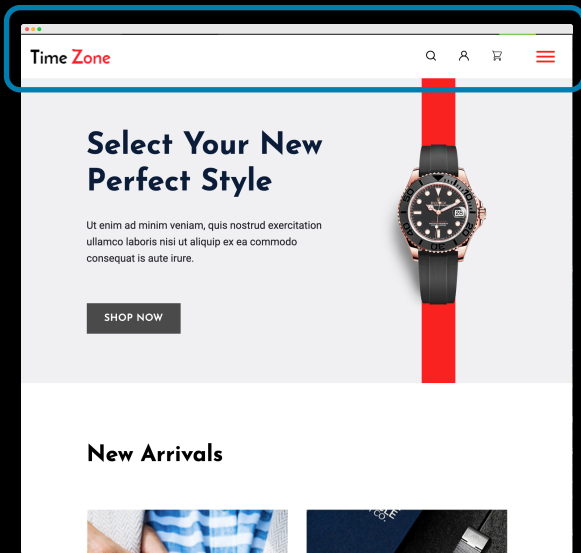
brand.com/user-area

Project Situation

- company uses multiple systems in one platform
- redundant development work
- slow and synchronized release processes
- no option for system migration
- independent teams



Project Situation



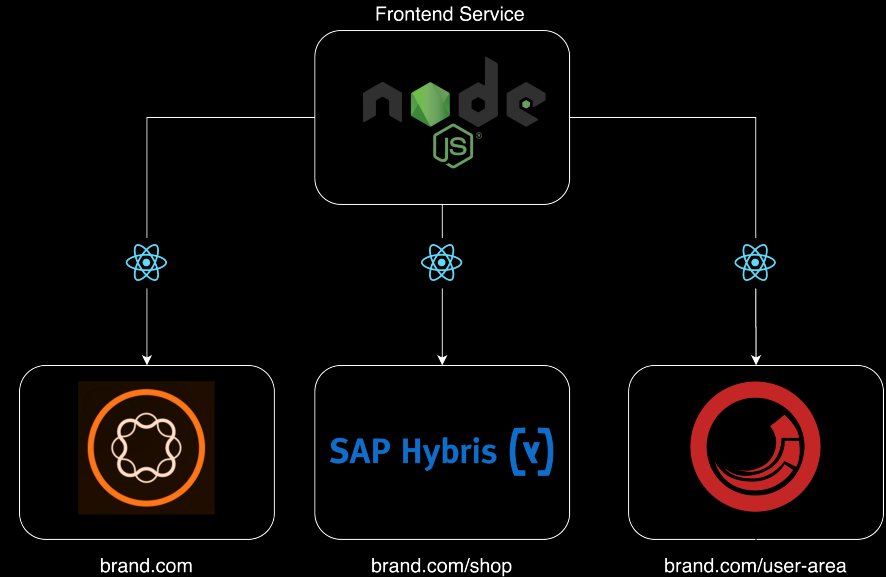
- Redundant implementations for multiple components
- Synced deployment processes

Solution Concept

Solution Concept

Specs:

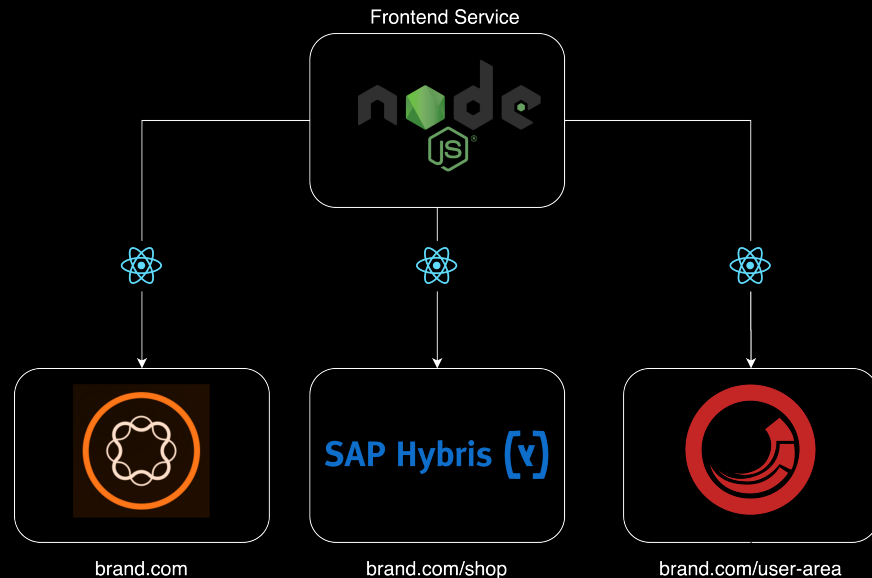
- decouple frontend development (Frontend Service)
- One common frontend stack through the systems
- Frontend components ship dialogs
- „Hybrid headless“ set up

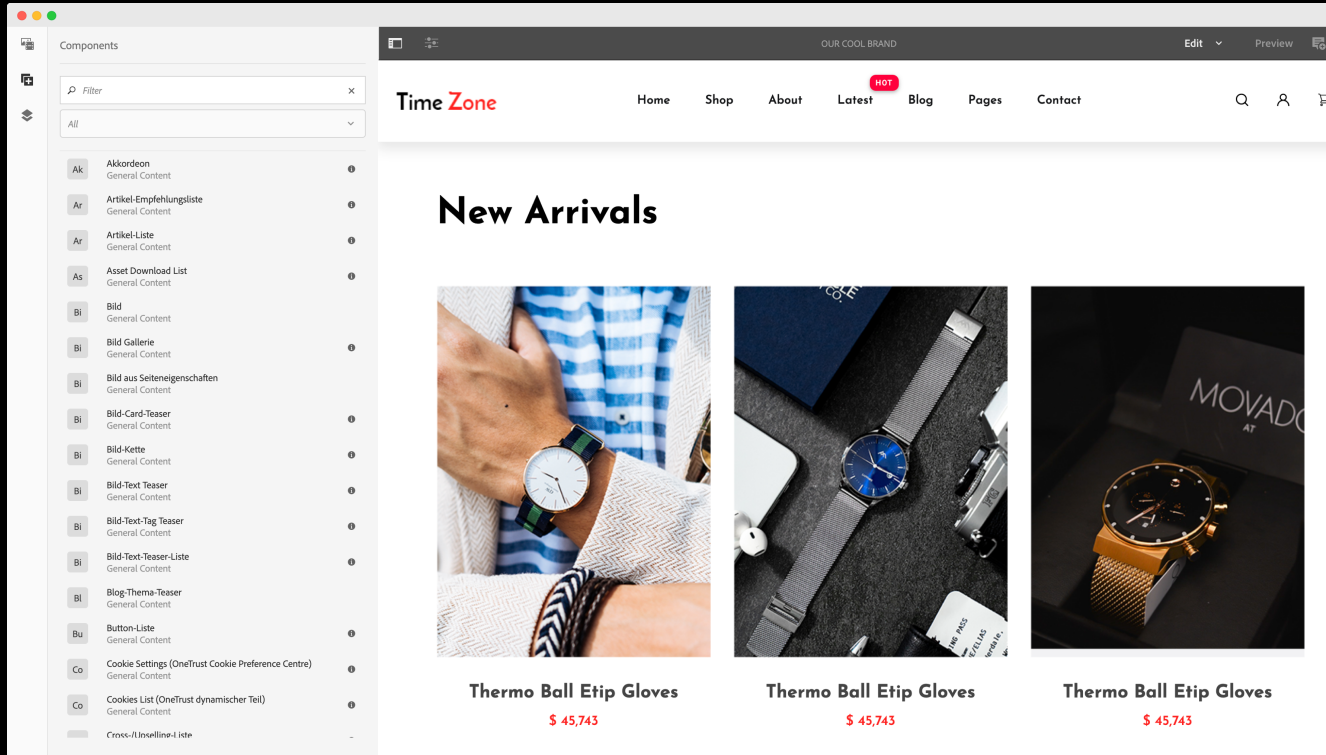


Solution Concept

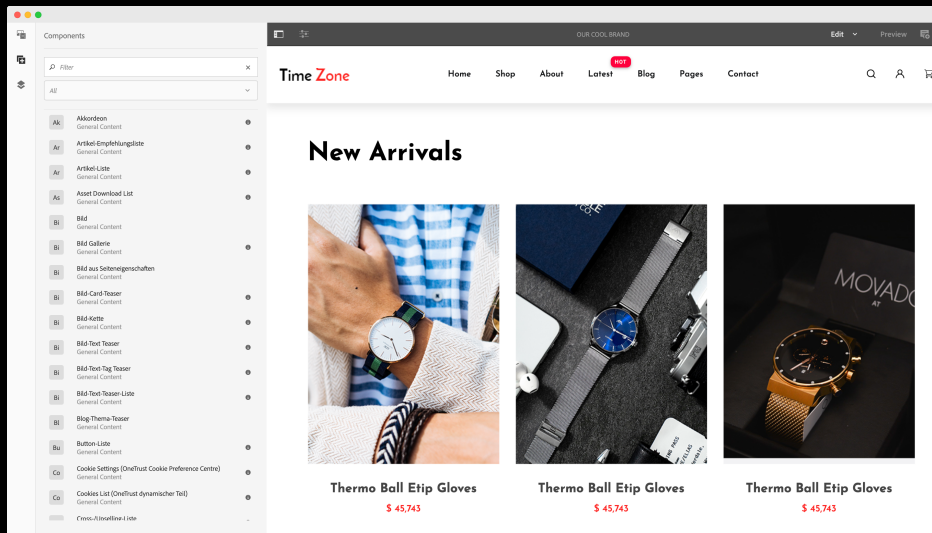
What it solves:

- ✓ One release process for all systems
- ✓ deployment is synced
- ✓ no redundancies
- ✓ Technology agnostic
- ✓ no AEM deployment needed for new frontend components

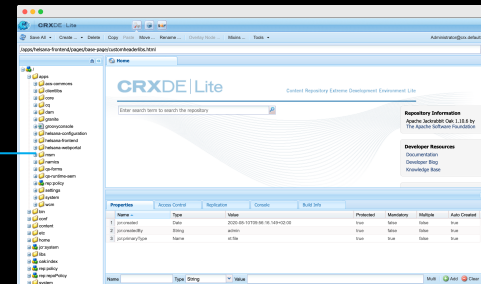
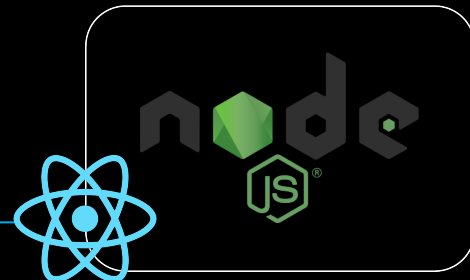




Solution Concept

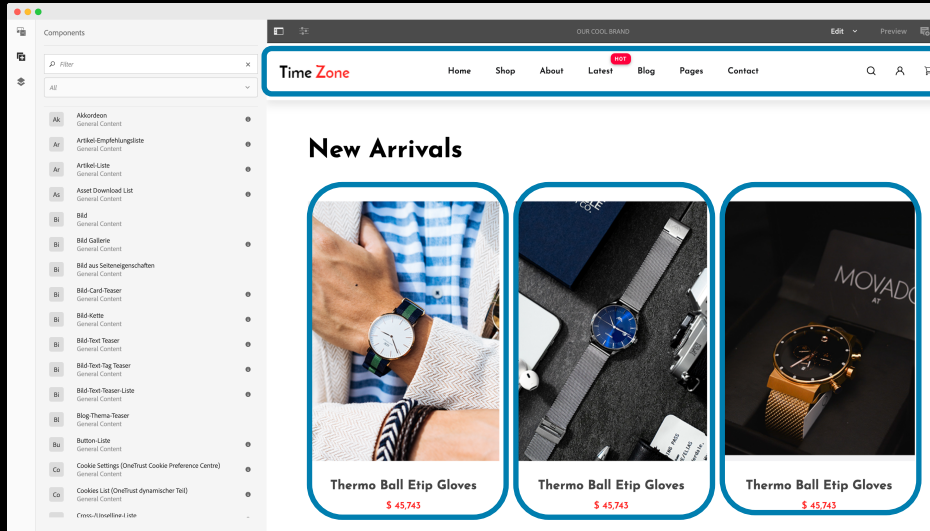


OSGI Service

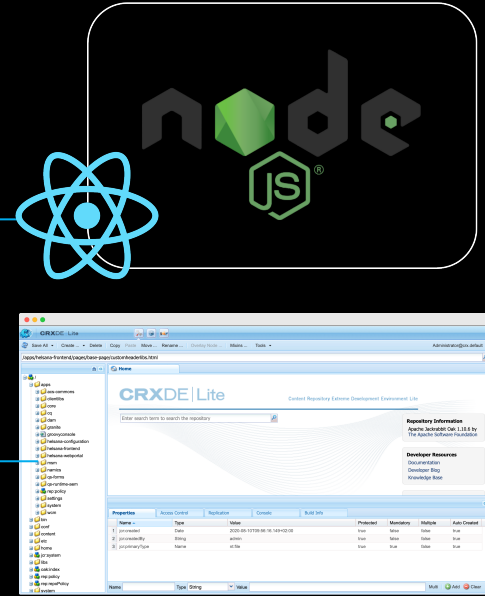


Solution Concept

- OSGI Service checks which components are used (in AEM)

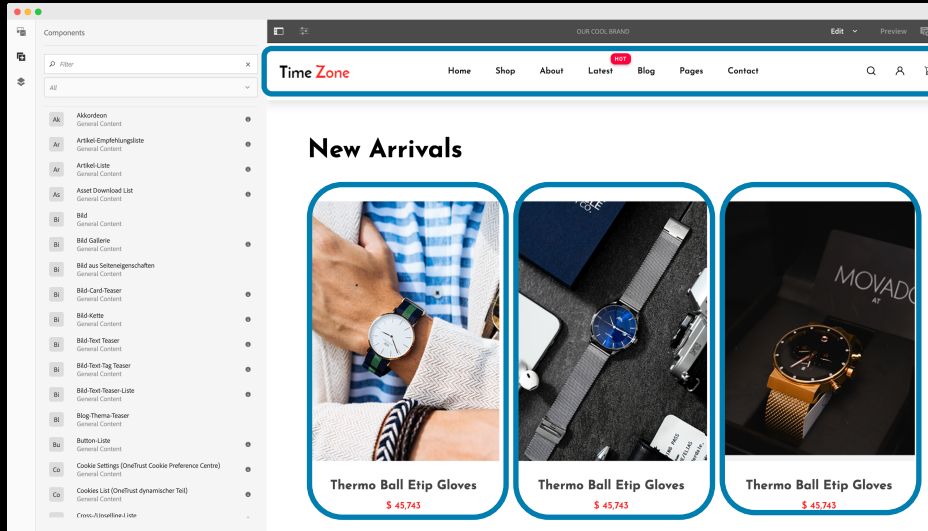


OSGI Service

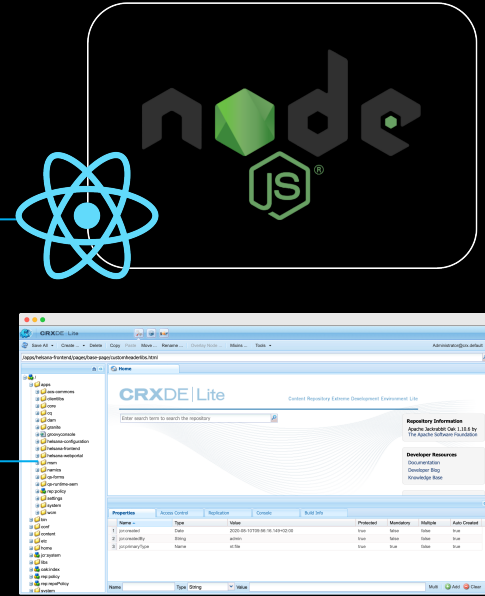


Solution Concept

- OSGI Service checks which components are used (in AEM)
- Calls Frontend Service to get needed assets (JS / CSS)

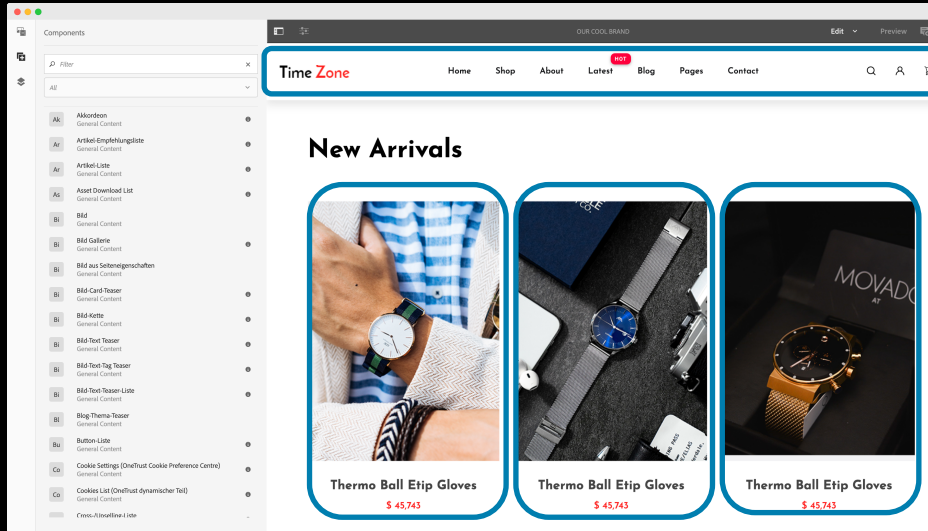


OSGI Service

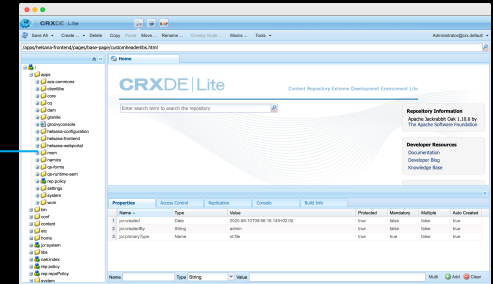
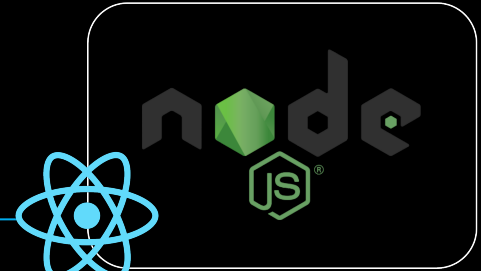


Solution Concept

- OSGI Service checks which components are used (in AEM)
- Calls Frontend Service to get needed assets (JS / CSS)
- Same Concept for all other systems



OSGI Service



Frontend Architecture

- OSGI Service reads page content and gets all apps
- Calls Frontend Service to get assets

```
<sly data-sly-use.fs="de.teclead.FrontendService"/>
<head>
    ${fs.getAppCSS}
</head>

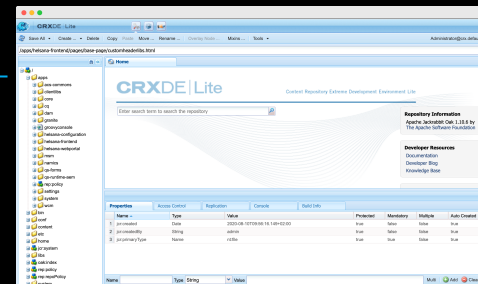
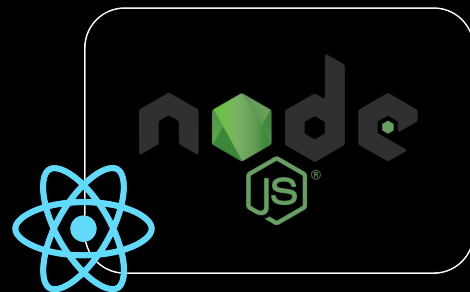
<body>
    <div class="fs-navigation" data-params={...}>
    <div class="fs-product-teaser" data-params={...}>
    <div class="fs-fooooter" data-params={...}>
    ${fs.getAppJS}
</body>
```

Frontend Architecture

- Micro Apps concept is used
- OSGI Service injects JCR data into the components as JSON
- Sling Model can be used for adding business logic

```
<head>
  <link rel="stylesheet" href="fs.brand.com/apps/navigations.css">
  <link rel="stylesheet" href="fs.brand.com/apps/product-teaser.css">
  <link rel="stylesheet" href="fs.brand.com/apps/fs-foooter.css">
</head>
<body>
  <div class="fs-navigation" data-params={...}>
  <div class="fs-product-teaser" data-params={...}>
  <div class="fs-foooter" data-params={...}>

  <!-- dynamic by sightly -->
  <script src="fs.brand.com/apps/navigations.js">
  <script src="fs.brand.com/apps/product-teaser.js" />
  <script src="fs.brand.com/apps/fs-foooter.js" />
</body>
```

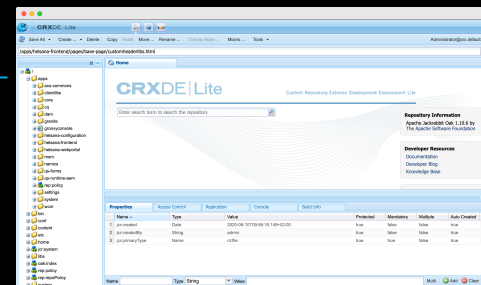
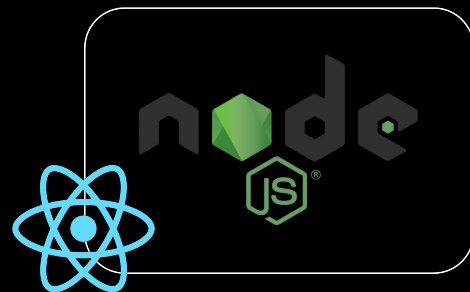


Frontend Architecture

- Inline option available => no external Domain calls
- OSGI service injects assets into the page directly

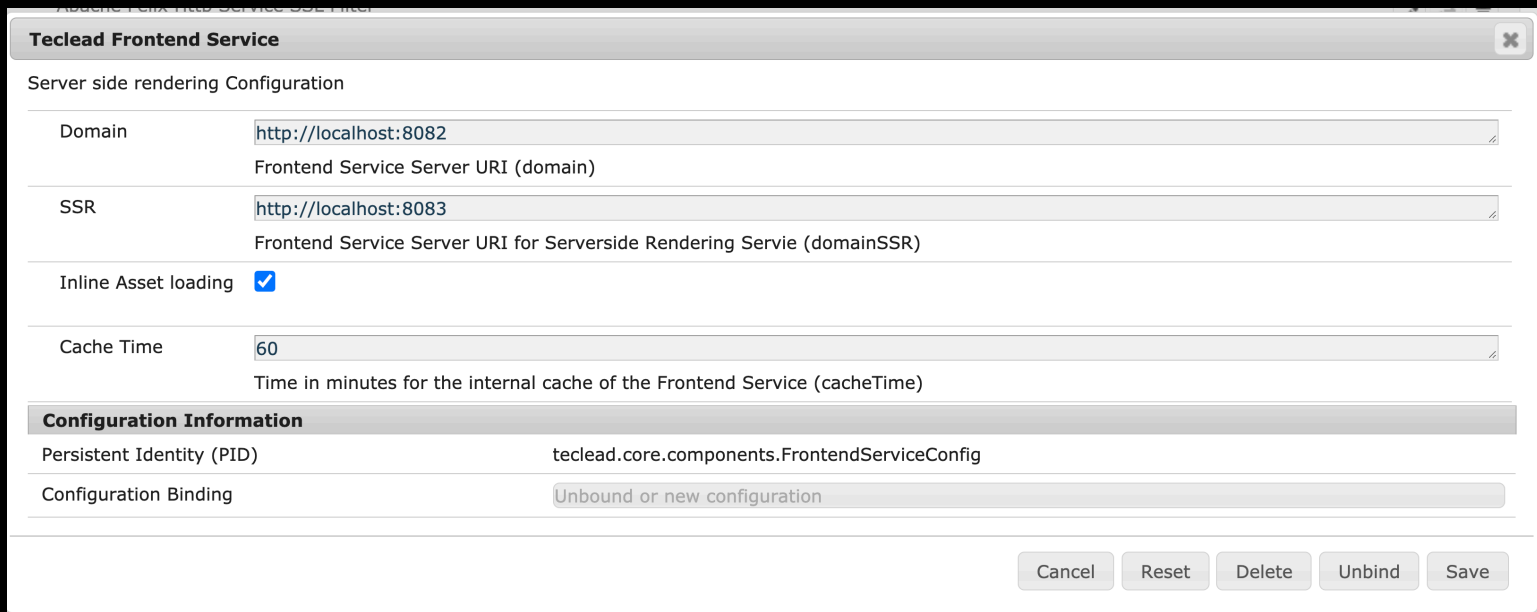
```
<head>
  <style>
    {inline css from service}
  </style>
</head>
<body>
  <div class="fs-navigation" data-params={...}>
  <div class="fs-product-teaser" data-params={...}>
  <div class="fs-foooter" data-params={...}>

  <!-- dynamic by sightly -->
  <script>
    {inline js from service}
  </script>
</body>
```



Frontend Architecture

- OSGI Config available
- Inline options for internal loading



Teclead Frontend Service

Server side rendering Configuration

Domain
Frontend Service Server URI (domain)

SSR
Frontend Service Server URI for Serverside Rendering Service (domainSSR)

Inline Asset loading ☒

Cache Time
Time in minutes for the internal cache of the Frontend Service (cacheTime)

Configuration Information

Persistent Identity (PID)

Configuration Binding

Cancel Reset Delete Unbind Save

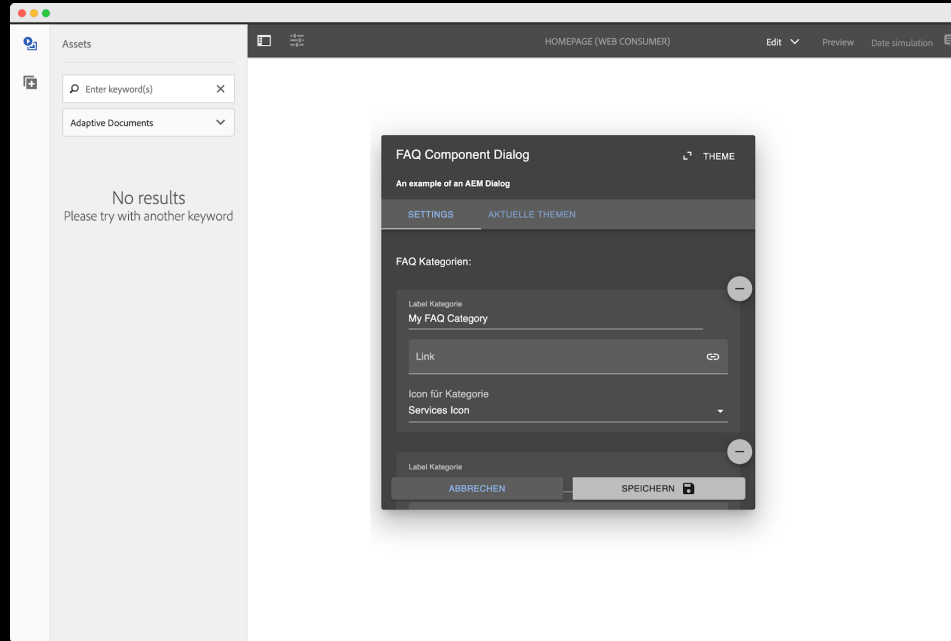
Sightly Component

- All components use the same ResourceSuperType
- Removes complexity out of Sightly templates
- Can be used together with Serverside Rendering
- More Details can be found in last years presentation:
- <https://www.youtube.com/watch?v=Byfs7ptHhiU>

```
<div data-sly-use.fs="de.teclead.FrontendService" class="aem-react-component ${fs.componentName}"
    data-params="${fs.data}">
    <!-- Component render here, SSR is optional-->
</div>
```

Authoring

One Dialog to rule them all



brand.com



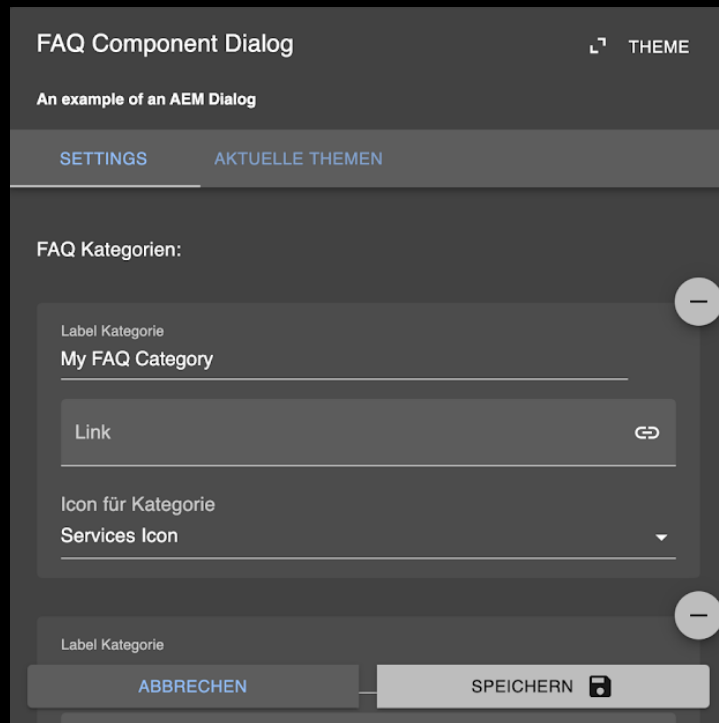
brand.com/shop



brand.com/user-area

Authoring

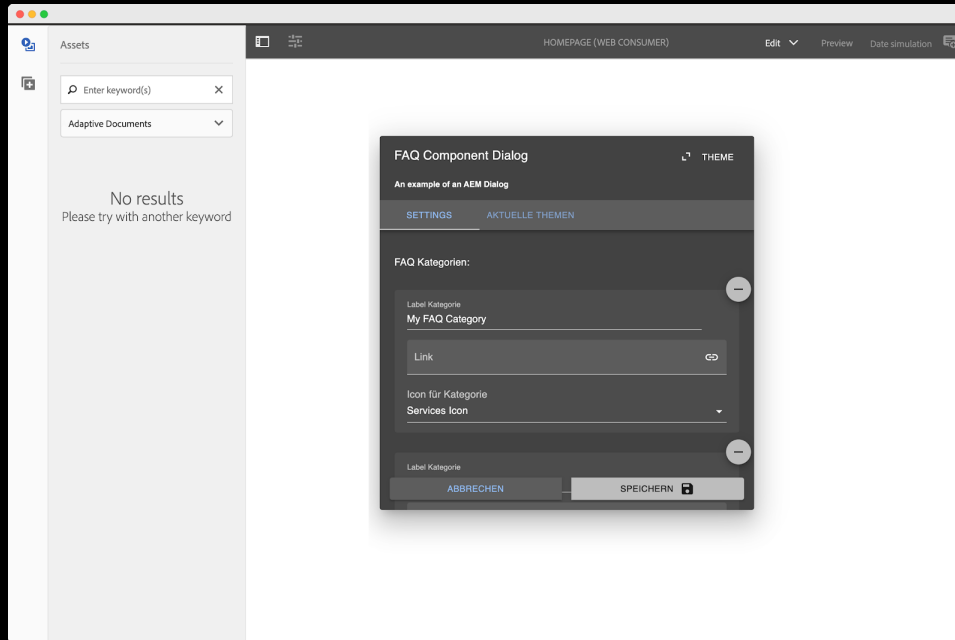
- Authors can decide: Unified vs. Touch UI
 - All dialogs are shipped from the Frontend Service
 - Data is stored in the JCR
 - Works with the standard Dialog as well
 - Dialog can be used in all systems
-
- <https://www.npmjs.com/package/@teclead/aem-generator>
 - <https://www.npmjs.com/package/@teclead/dialog-generator-react>



The screenshot shows a web-based dialog titled "FAQ Component Dialog" with a "THEME" button in the top right corner. Below the title is the subtitle "An example of an AEM Dialog". The dialog has two tabs: "SETTINGS" and "AKTUELLE THEMEN", with "AKTUELLE THEMEN" being the active tab. The main content area is titled "FAQ Kategorien:" and contains a form for editing a category. The form includes a "Label Kategorie" field with the value "My FAQ Category", a "Link" field with a chain icon, and an "Icon für Kategorie" dropdown menu with "Services Icon" selected. At the bottom of the dialog, there are two buttons: "ABBRECHEN" (Cancel) and "SPEICHERN" (Save) with a save icon.

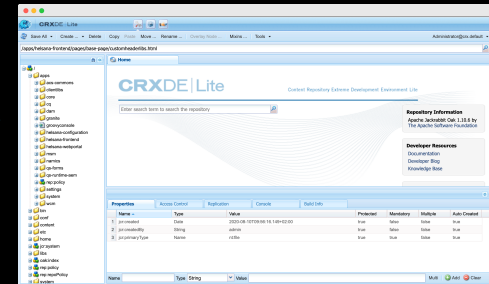
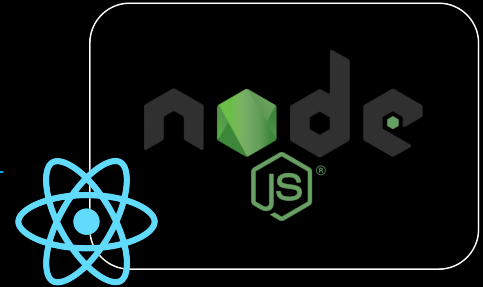
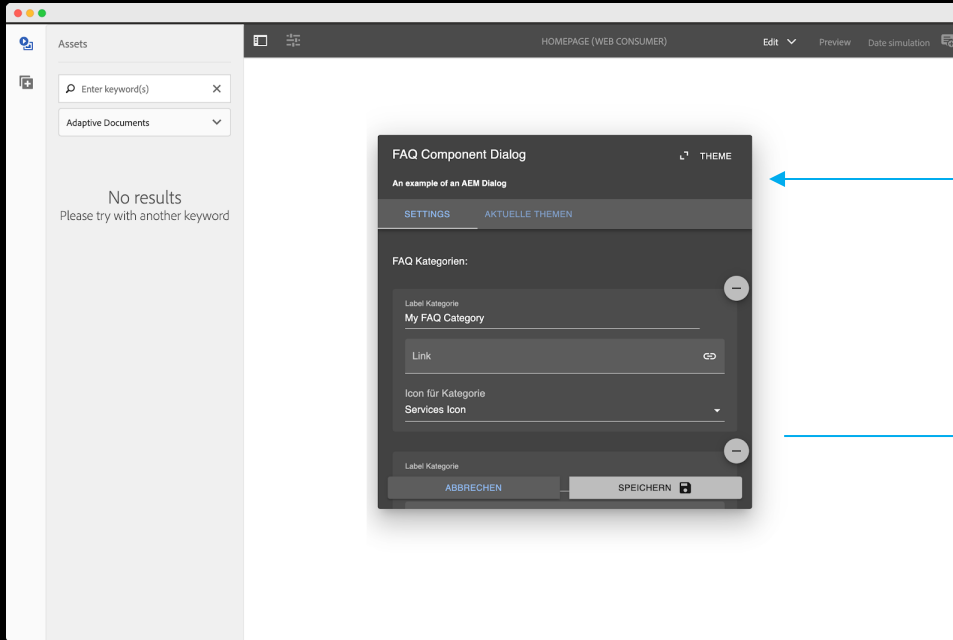
Authoring

- Dialogs POSTs data to an AEM Servlet
- Stores it like the normal AEM dialog in the JCR



Authoring

- Dialogs POSTs data to an AEM Servlet
- Stores it like the normal AEM dialog in the JCR



Demo

FAQ

Does everything needs to be in the Frontend Service?

- architecture can be mixed with normal AEM development

Does everything needs to be in the Frontend Service?

- architecture can be mixed with normal AEM development

Does it work with Core Components?

- React components still use Core Components and child parsys

Does everything needs to be in the Frontend Service?

- architecture can be mixed with normal AEM development

Does it work with Core Components?

- React components still use Core Components and child parsys

What about performance?

- components are cached in AEM / Dispatcher, no performance bottleneck

Does everything needs to be in the Frontend Service?

- architecture can be mixed with normal AEM development

Does it work with Core Components?

- React components still use Core Components and child parsys

What about performance?

- components are cached in AEM / Dispatcher, no performance bottleneck

Can we use touch UI Dialog only?

- Yes! The external dialog is optional for 3rd party systems

Tony Schumacher

Partner

Mail: tony.schumacher@teclead.de

HQ: Berlin

